Computer Science NEA
Jack Leverett

Candidate Number: 7714
Centre Number: 50639

Table of Contents

Analysis

Background to the project

What is BeReal?

BeReal is a new social media platform based around the idea of taking a single picture a day. You can be prompted to take the picture at any random point. It is supposed to give a real insight into what someone is doing day-to-day. This presents a very different kind of social media that gives people an un-altered view into their friends' lives.

It's considered to be a healthier kind of social media; it doesn't allow fabrication and stops users from so called "doom scrolling". Doom scrolling is where users endlessly scroll without purpose through seemingly endless content. BeReal doesn't allow this since there is only 1 post per person. This has also made it a fairly distraction free social media, most people will only look at it once a day for a brief amount of time at most.

Why do companies need a social media platform?

In the modern-day companies are constantly striving for a more engaged workforce. With the rise of technology and in recent times AI many jobs are being made obsolete. The workforce now has to be more connected and creative than ever; good communication and positive workplace relationships enable this.

Having a platform that acts similarly to BeReal, would promote friendly communication in the modern workplace, which intern allows for better collaboration and overall better work from all employees. Big players have noticed this too, Microsoft now include a social media platform in their office 365 suite. This productivity suite is the most popular in the world and its adoption of a sovereign social media platform shows there is a market for a better way to engage employees.

Modern companies also face the challenge of maintaining a workplace culture while many employees work from home. For these employees all their interactions will happen entirely over video platforms such as teams, and since these meetings are often set-up with a purpose in mind, have almost no time to connect with their co-workers. A system similar to BeReal could inspire conversation and communication between these work from home employees and their co-workers. This can vastly improve communication and so lead to better teamwork.

The BeReal model is ideal for companies since it can inspire this conversation and connection while keeping distractions to a minimum. An employee can't get distracted for hours when there is only a handful of posts to see.

Current System

BeReal

As explained above BeReal involves everyone getting a randomised notification once per day to take a picture. You are given a 2-minute interval to take this picture in, however, if missed, you don't lose out on the post instead your post is simply marked as being late.

After a post has been created users can comment and react on their friends' posts, as they appear in the users' feed. A user's post will remain public until the next day when a new one is created. Old posts are put into a different section and only viewable to the user themselves, here they can see the date each photo was taken as well as a preview.

Friend requests must be sent by one person and then accepted by the other. This means the only posts that appear on your feed will be posts from friends rather than people you "follow" or from "recommended posts".

Yammer

This is currently one of the few (if not only) intra-organisational social media solution. It acts like other forum or "community" based social media (reddit etc). Users from within the organisation can create "communities" and invite or allow other users from the organisation to join. From within the community, they can create posts, polls and announcements.

Whoever created the community has complete control over it as the community "administrator". They can also promote and remove members from the community. A user can join as many communities as they want and add friends from across their organisation.

The Problem with BeReal

For a company to have its own BeReal-like social platform, typically, they would have to rely on 3rd party-hosting, support, and moderation in the form of BeReal. Relying on a 3rd party for a service such as this has several implications.

Control

A company using the BeReal platform would likely be worried about the possibility of confidential data accidentally being posted by a user. Or one of their employees posting something that violated the companies code of conduct. This could result in the company wanting to take actions to remove said post to stop further damages both monetary and to the company's reputation.

However, all moderation and rules are created and enforced by BeReal themselves. If the company needs to remove a post from one of its employees, they would have to raise an

appeal with BeReal, but there is good chance that a certain post could breach the companies code of conduct but not BeReal's and in this situation the company has no power and the post will remain up.

Shut Down

In the case BeReal shuts down, the organisation using their social media loses all access to the service. So due to forces outside of their control they lose access to a platform that could become essential to their workplace culture. BeReal shutting down seems like an unlikely scenario but as BeReal currently has no monetisation, and there has been large recession within the technology market since the peak of COVID-19 it's not entirely outlandish to see BeReal having a short lifespan.

Features

BeReal also lacks a few features and settings that would be essential for a platform like this to work within a company. For instance, a company would likely want to make sure the daily post goes off within certain times of the day to not disturb employees during weekends or the evening. Larger organisations will also have a number of large teams, departments and even sites. In cases like this, employees likely don't want to see posts from someone who they never have or ever will meet. BeReal only has a simplistic friend system meaning employees can't be organised into their teams without each user manually friending and un-friending people as departments grow/shrink. This also doesn't allow users to switch their feed to be from a specific team, some employees may play a role in a number of departments and so not being able to organise their feed into separate teams could make their feeds completely irrelevant to them.

Security

Additionally due to organisations having no way to remove posts quickly and effectively, members who unwittingly capture confidential or sensitive information in the background of one of their posts could put company and individual security at risk. This has already been occurring as stated in this article:

https://www.worklife.news/technology/bereal-workplace/

To summarise, security experts have warned of the privacy implications that BeReal brings to the employee and customers of a business. If an image is shared that contains confidential information it could lead to potential data breaches and security incidents. Beyond security it could put the innocent employees at risk if they don't want themselves posted on social media without proper consent.

Once something has been put out on a platform it is out of the control of the people who use it, content becomes almost impossible to remove.

The Problem with Yammer

Currently an organisation looking for a social media platform is only left with the choice of yammer (viva engage). However, many system administrators have reported several moderation problems. For instance members have been “treating the platform like their personal Facebook”. Many organisations (including our school) have had to disable it entirely due to these moderation issues.

Bored employees given access to a mandated social media are likely to make multiple posts a day, wasting huge amounts of time and energy. Yammer being connected with Microsoft 365 also means communities generate huge amounts of notifications for every user. This can fill someone’s inbox and get in the way of actual work.

System admins response

One system admin commenting on this Reddit post:

https://www.reddit.com/r/sysadmin/comments/7ymxwg/anyone_using_yammer/

“within two days was people sharing memes, posting gifs, and basically shitposting all over the place... One month in, people are still using it like Facebook. We have a bit of a joke in IT, a bet if you will, on how long Yammer will last in our organization. I'm guessing 8 months before they pull the plug.”

This highlights how Yammer’s no limit approach to content creation from each user has sent IT personal to the frontlines of content moderation. Unable to fully tackle the tide of content that just a few users can post. Another admin said their small organisation had to designate:

“two or three admins (usually site HR being one) that are keepers of the page”

Another said:

“And no one wants to hire someone just to be the full time Yammer police”

All this shows how the no limit approach is overwhelming for small organisations. Additionally, only having Admin staff able moderate and remove content puts great load on a small subset of people. If the responsibility was shared among mangers to handle their own teams, then the content would be moderated more diligently.

Many organisations have subsequently had to turn off the feature of office 365 and settle for nothing. The small social organisation market has no practical platform to both engage employees and provides the tools to properly moderate the content. This has left a huge hole in the market, which has yet to be addressed.

Prospective Users

Organisation members

Who are they?

This is the main user base of the platform. These people expect the platform to run smoothly without a hitch. They also expect the system to be configured and moderated correctly by managers and admins.

How will they use the system?

Creating posts: Upon receiving their once-a-day notification these users will be encouraged to capture whatever they are doing in the moment.

Viewing and interacting with their team's posts: Once they have made (or missed) their post, team members will be able to like and interact with posts within their team. The number of likes will be viewable while comments give members the chance to engage with each other.

Editing their profile: Users will be able to choose what information they have displayed on their profile. This can be email, name, phone number, job role etc. This information may need to be changed and altered as time goes on and so its essential users are given the ability to change all of this.

Managers/Heads of departments

Who are they?

These are the heads of departments and managers of groups of employees. These people are liable for the actions of their team, this means they will be responsible for their team's content and use of the platform.

How will they use the system?

Moderating user content: They will need to be able to remove posts, comments and other content. They will also need to enforce their workplaces codes of conduct and resolve disputes. The key part here is they should only be able to enforce these powers within their team.

Managing different "departments": These users will also want to be able to manage who is in their team removing and adding employees as departments change size.

Admins

Who are they?

These are the people who set-up the software. They will be in charge of keeping the system working as well as modifying the different settings to suit the organisation needs. In most organisation this role will be taken on by an IT specialist or system administrator.

How will they use the system?

User "roles" and team creation: Admins will oversee assigning the powers and initial creation of teams. They will assign a team leader role to each one and from their allow the person fulfilling this role to take over.

Instance settings: Every organisation will have different needs and wants of BeOpen. Admins will have control over a number of server settings such as the time of day that notifications will be sent out.

Security settings: Certain organisations require higher levels of security; these admins will also be in charge of configuring that. For instance, how long until users will have to re-login? How secure should passwords be? Should users require a special code to register?

Questionnaire / Interviews

1) *What is your organizations current way of communicating/maintaining relationships with work from home employees?*

Prospective user 1:

"Currently, Teams - How ever it is seldom used and is incredibly one sided as organisers have majority of the control and i have even been in teams were messages from lower ranked people were only able to reply to the higher-ups"

Prospective user 2:

"Bright HR, whatsapp"

2) *What benefits and problems are there with your organizations current system?*

Prospective user 1:

"The file sharing elements of Teams is really inconsistent and often breaks/doesn't save. Benefit, a (slightly) more casual comms method than email and is easier to include multiple people"

Prospective user 2:

“Benefits: Its an app that almost everyone has anyway, I can use it for more than just work, Its simple. Problems: Lots of bugs, Boring”

3) *What functions would a social media for organizations need to make it sustainable/usable?*

Prospective user 1:

“Open 2 way comms Easy direct comms if needed Casual atmosphere - important things are harder to bring up if everyone is tense”

Prospective user 2:

“It would need to function as a benefit to the workplace environment, and hopefully increase workplace productivity.”

4) *What problems could arise out of a work place social media and how could they be addressed?*

Prospective user 1:

“Social media becomes a distraction if it lets people use it to often so a way of limiting that would benefit productivity”

Prospective user 2:

“HR Problems with inappropriate workplace posts. Would need a moderator/team of moderators.”

5) *What do you feel could be the best way of limiting distractions from a social media platform like this?*

Prospective user 1:

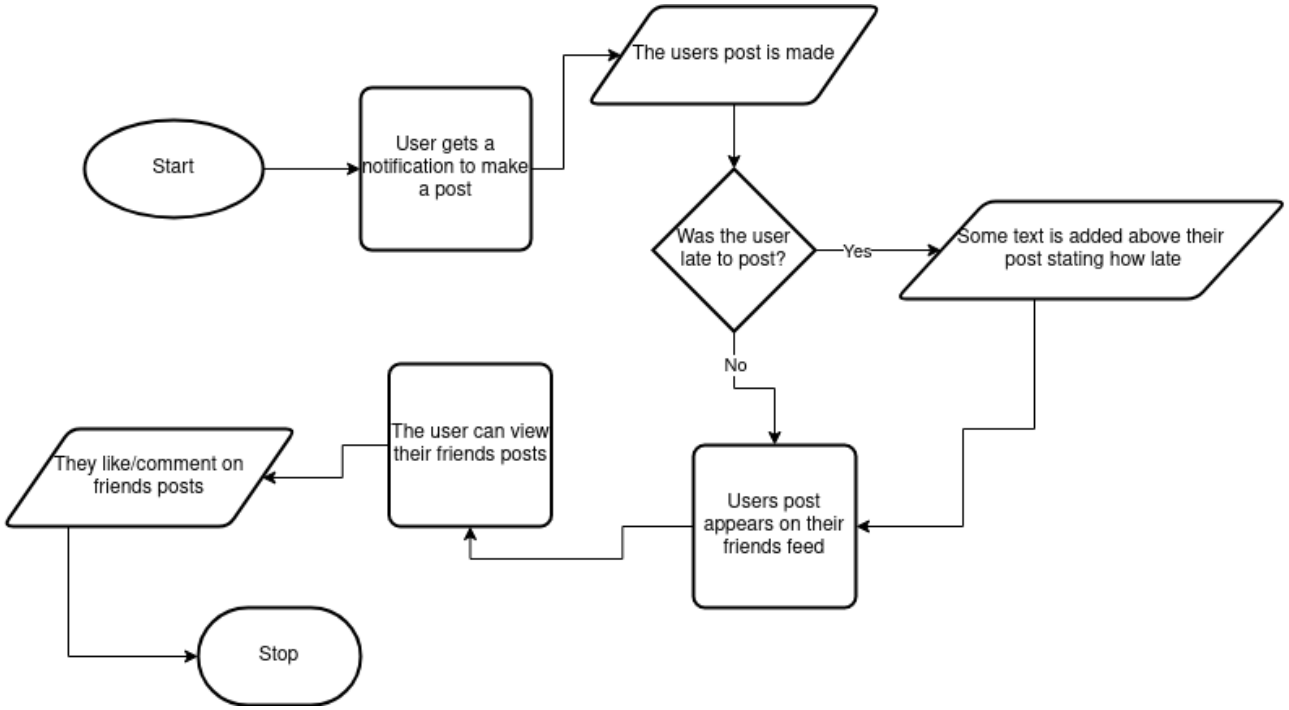
“You could make it so there is a limited amount of comments or screentime. Another way would be limiting people to only being able to follow people in their department”

Prospective user 2:

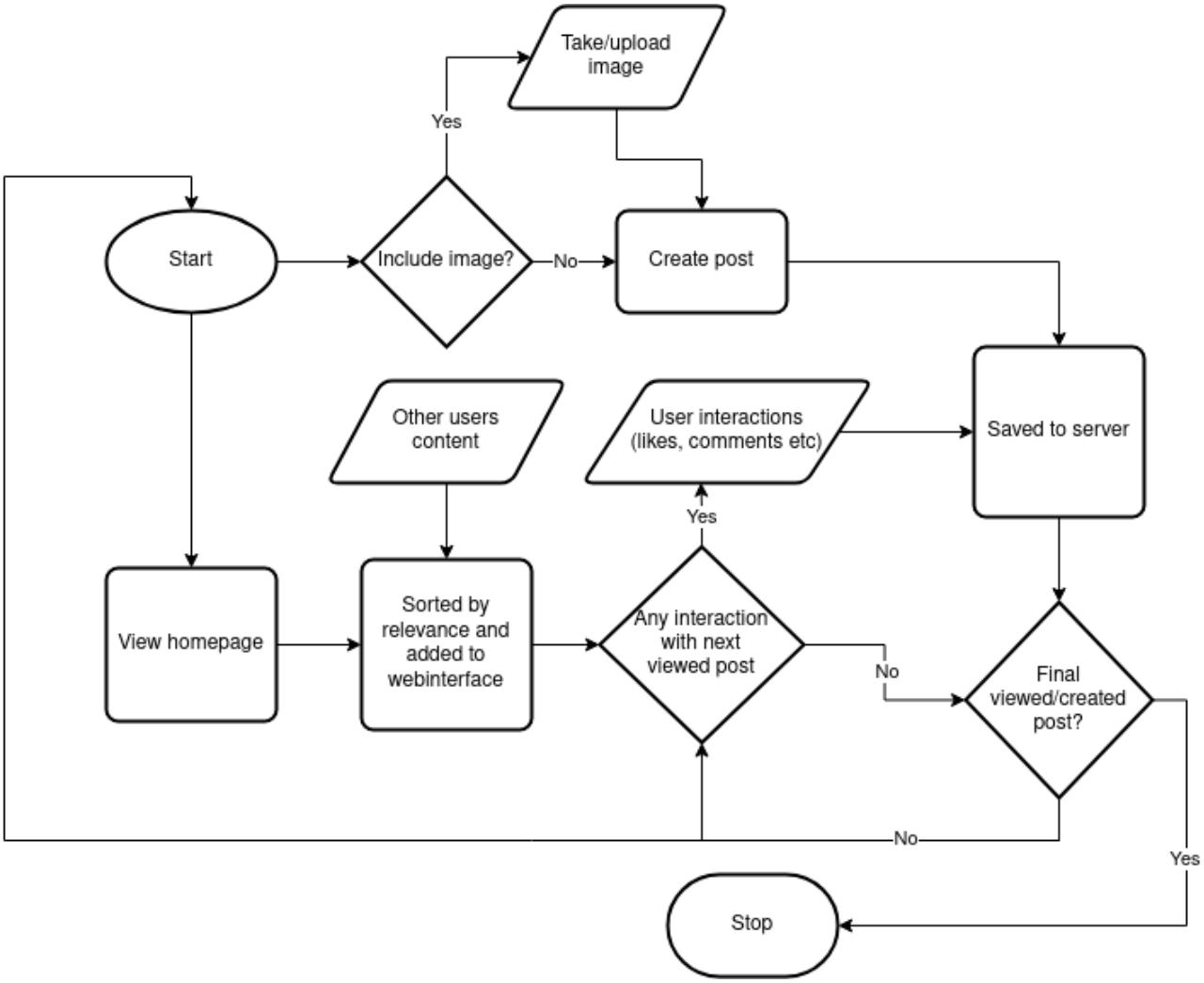
“Screentime”

Diagrams

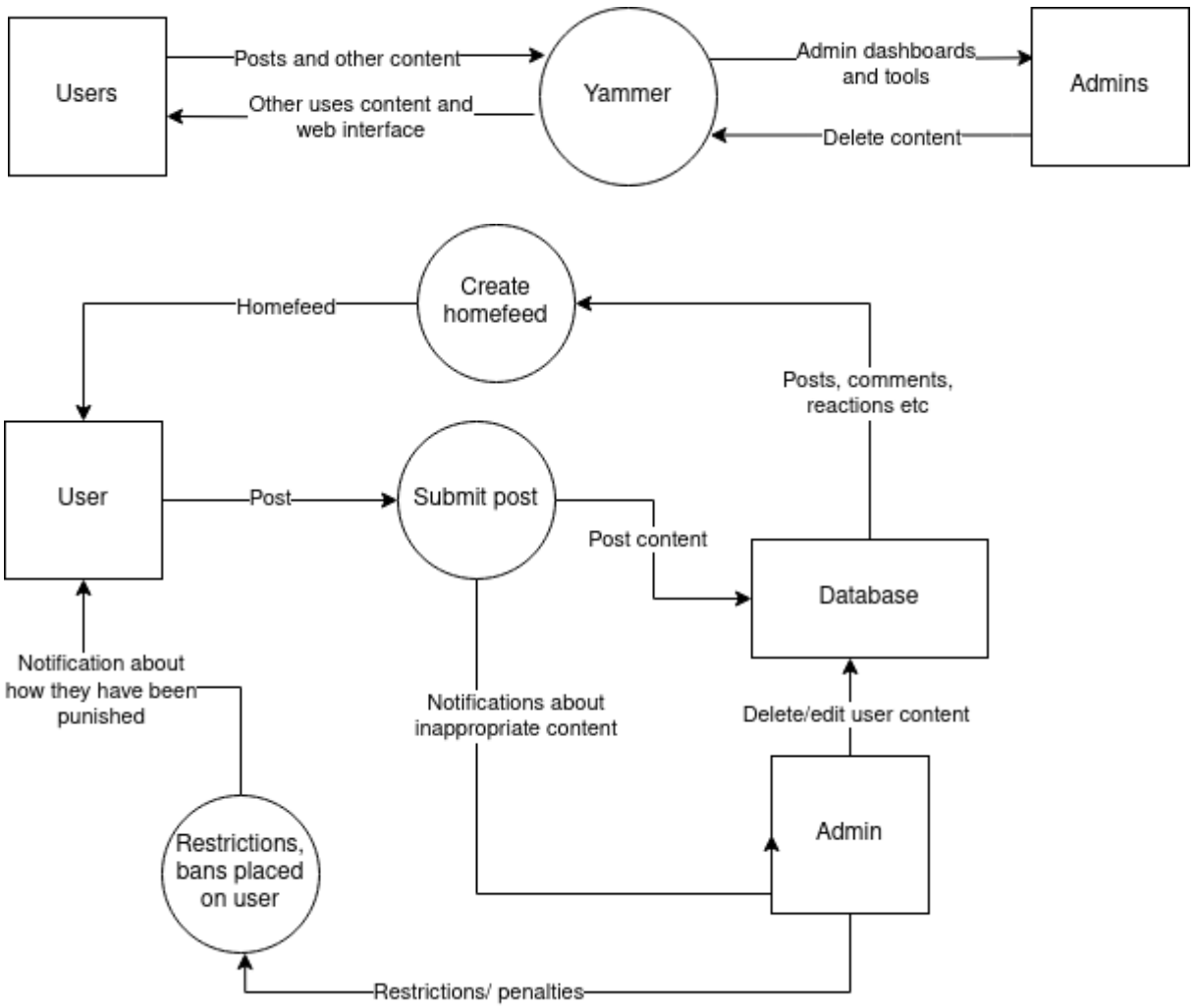
BeReal flow diagram



Yammer flow diagram

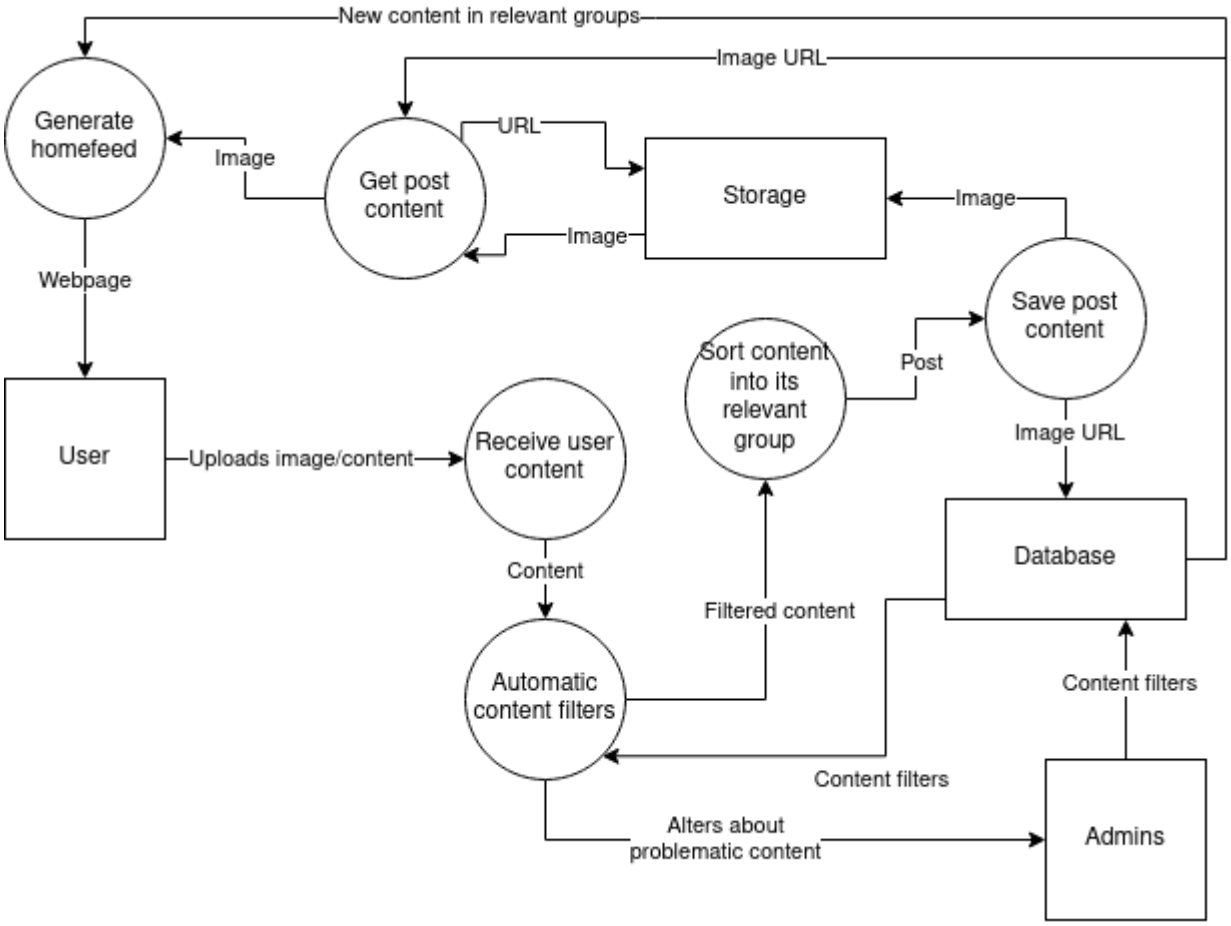


Yammer level 0 DFD



Yammer level 1 DFD

Shows simplistic insight into the yammer system itself how posts are submitted and added to the database as well as how home feeds are created and presented to the user



Yammer level 2 DFD

More detail into the “submit post” process and how images are saved and fetched for home-feed generation. Shows how images attached to text posts are actually saved and assigned a URL and that URL is then kept in a database along with other data about the post. It also shows a more in-depth look at how Admins can create filters and limit users.

User Requirements

Must

- Limit the number of posts a user can create within a day for instance limit it to one post a day.
- Send out a single notification at a random point in a day, to prompt all users to create a post.
- Place a time constraint on the users post, if a post is not created within 5 minutes of the notification remove that user's ability to post for the day.
- Allows users to log into existing accounts and register new accounts as needed.
- The assigning of different user levels based on the users position in the organisation. These different levels can give users access to functions and moderation tools.
- The creation of different teams and appointing of team leaders who will have superior management tools, to moderate their team's posts.
- Implement a friend system, where people can send friend requests and the recipient can either accept or reject said friend request.
- Each user has a profile with a number of different fields that can be edited by them, like bio, name, contact information, team role, etc
- Provide an easy to use and accessible mobile GUI for android as well as allowing user to take pictures via the app for their post.

Should

- Utilise a custom hash and UUID algorithm to reduce risk of malicious code being side loaded from 3rd party dependencies.
- Have a settings page with a number of options that should be saved to a local database.
- Have a team and organisation page where managers and administrators can create new occupations and view other teams.
- Create an algorithm for friend recommendations to users, these recommendations could use a form of searching algorithm and point system to try and gather relevant people on a number of factors and recommend friend requests to such users.

- Implement a notification service like Unified push (ntfy) or other API to allow real time notifications.

Could

- Create an algorithm to allow sorting home-feeds by relevance to the user. This could be dictated by matching up factors such as common friends, likes, comments and average number of post interactions
- Allow the organisation to customize a number of security features and rules.
- Allow database encryption for extra security including a Shamir secret sharing backup method for the encryption key
- Provide a section of the app dedicated to different stats about the users' teams. Providing leader boards and graphs of consistency.
- Limit screen time for users and make the amount of time customizable by Admins.

Won't

- Create an app for IOS devices or MacOS devices.
- Integrate **multiple** different notification APIs or services.
- Allows users to attach posts from an external source (like their camera roll)
- Allow users to create or add a profile picture.

Objectives

1) Each of a user's posts will contain an image taken using the BeOpen app which they took within the time frame provided by the service, along with an optional caption.
2) The system will prompt users to create a new post within a given time frame once per day. They will be prompted with a notification through the app, notifying them when they can first create a post and when they missed the time window.
3) Allow administrators to change several different security settings, such as the how long authentication tokens are valid for.
4) Creation of teams and the sorting of members into teams relevant to the member. This sorting is done by associating all people with the same occupation into a single team.
5) Teams should have an appointed "team leader" this person will have the access to tools to allowing them to moderate content and users (delete posts, comments or remove users) within their team.
6) Users should be able to accept and send friend requests to allow them to see posts from individual users who can be both internal or external to the users team.
7) Users should be able to like and comment on other user's posts. These other posts can originate from their team or from a friend. Comments should also be able to be liked, by other users. There should be a total likes counter displayed on posts and comments.
8) The BeOpen server should maintain well detailed logs of both user activity and server functionality. These logs should be accessible via any deployment method including container systems like docker.
9) Users should be able to customize certain features of their profile, for instance their legal name and role in the company. This information will be visible to anyone who clicks on their username.
10) Each user should have a username and password combination that they can use to login to their organization BeOpen instance. The client should also have the ability to store login tokens which can be used for automatic authentication.

Design

Overview

Friends

The system has 2 main ways for a user to make social connections, through their team and their friends. A friend connection is formed when a user sends a “friend request” to one user and it is accepted. Friends will be able to see each other’s posts on their home feed.

Teams and Occupations

A user joins a team by setting their occupation. A users occupation connects them to a team which is managed by assigned “team leaders” who have special permissions to delete content made by the members of that team. Users can change their occupation by creating an “occupation change request” via their profile page. Then management or an admin can accept or reject a change request.

For example, a school might setup a few occupations 2 of which are “maths teacher” and “computing teacher” each of these occupations will have an associated team named “maths teacher’s” and “computing teacher’s” by default (the name of a team can be changed and therefor different to the name of the occupation associated with it). If Mrs Nolan was currently a maths teacher but is changing profession and becoming a computing teacher, she can make an occupation change request to “computing teacher”. Once approved by management, for instance the headteacher, Mrs Nolan will change occupation and her new team would be “computing teacher’s” and so would start receiving posts from said team, as well as Mrs Nolan’s posts being received by all others in said team.

Posts and comments

A user can post once per day within the timeslot. The time slot for any given day is generated at the start of the day, it will pick a (by default) 5-minute slot at a random point in the day to allow users to post. Users are sent a notification of when this is using the “ntfy” implementation of “unified push” a push notification protocol that works across all devices. It also uses an internal notification system for when you’re on the app itself.

Posts contain 2 pieces of content an image taken within the given 5 minutes, and a caption also written in these 5 minutes. The user on the client will be prompted to take a picture using their devices camera, they are allowed to retake the photo if it is within the time limit. Then they can add a caption and click the “post” button. Once the post is on the server it cannot be edited, however it can be deleted by the user.

Once posted the users friends and team will be able to see their post in their own home feeds. From there each user can like and comment on the post.

Comments can be submitted to a post at any time. Each comment can also be liked itself.

The rules for who is privileged to delete a piece of content:

- If the content was made by the user themselves
- The user attempting to delete the content is the team leader of the user who created the content.
- The user attempting to delete the content is management or an admin.

Memories

A user can view all their previous posts from the “memories” page. Here they can select a year, month and a day, which will then show them the post made on that day. If a post wasn’t made on a particular day the date won’t appear in the list.

When viewing an old post, you are still able to delete, and view comments from said post. You are also able to still like both the post and comments underneath.

Client-Server

The system uses a client-server model, the idea is, each organisation would run their own server for maximum sovereignty. The server will be designed to do most of the heavy processing and all data storage, for the clients who will act as thin clients only really managing how the data fetched from the server is displayed on the UI.

Connection

The server and client use the web-socket protocol to exchange information, this can utilise HTTP or HTTPS and can run on all modern devices. This protocol was chosen since it allows for the server to issue real time updates to the client without the client having to setup background tasks to update its UI.

Server

Configuration

The server auto-generates a configuration file that is stored in the server’s “data” directory. It utilises an INI format for ease of use even by inexperienced admins. This configuration file allows organisation admins to adjust security features, post time constrains and database adjustments (a full list of configurations is below). Overall, the configuration file makes adjusting settings on the server easy for any semi-competent admin.

Configuration options list

Authorisation

Name	Default value	Description
AdminKey	Randomly generated string	The secret used to allow admin registration (generally only used to add the first admin to the server)
RegistrationKey	Randomly generated string	The secret used to restrict registration to people within the organisation, if publicly known anyone (even outside the organisation) could make an account

UsernameMaxLength	20	The maximum string length of a username
UsernameMinLength	5	The minimum string length of a username
PasswordMaxLength	30	The maximum string length of a password
PasswordMinLength	5	The minimum string length of a password
TokenExpiryTime	2592000 (30 days)	The time to expire on a authorisation token in seconds

Database

Name	Default value	Description
Path	data/database.db	The file path to the database from the main.py file
Encrypt	false	Whether to encrypt the database or not. Can be enabled even after initial creation of the database.
ShamirSecretSharing	false	Can only be enabled if the encrpt is enabled. This enables the creation of shamir secret shares, their use is explained the in configuration and administration documentation and in the security section of this write up.
NumberOfShares	5	This is the number of shares that will be generated if ShamirSecretSharing is enabled.
MinimumShares	3	This is the minimum number of shares needed to decrypt the database and deconstruct the master key if Shamir secret sharing is enabled.
KeyPath	data/key.txt	This is the path of the file holding the actual encryption key used to encrypt the database, if encrypt is enabled.
EncryptedPath	data/.cryptdatabase.db	The path of the encrypted database, if database encryption is enabled.

EncryptionConfigPath	data/encrpytconfig.txt	The path for the encryption configuration file used when database encryption is first set up. Currently this file only contains the master password to be used. Only relevant if encryption is enabled.
SaltPath	data/.salt.txt	The salt which is added to the master password when generating the encryption key for encryption of the key.txt file. Only relevant if encryption is enabled.
SharesPath	data/shares/	The directory where the share text files are placed after they have been generated. This path must be accessible to the administrator for reading and editing. This is only relevant if Shamir secret sharing is enabled.

User

Name	Default value	Description
DefaultLevel	Member	The default authorisation level of a newly registered user, So either: Member, Manager or Admin
DefaultOccupationID	Null	The default occupation of a newly registered user, you assign it using the occupation_id of the occupation

Posts

Name	Default value	Description
PostTimeLimit	5	The time limit after the post slot starts for users to be able to post in minutes.

Notifications

Name	Default value	Description
DefaultExpireTime	604800 (7 days)	The default time that a notification expires after if not set manually when the notification is created. The unit is seconds
ntfyUrl	https://ntfy.example.com	The URL leading to the organisations ntfy instance. See the notifications section of the write up to learn about how ntfy works.

Miscellaneous

Name	Default value	Description
ServerCode	Randomly Generated String	A prefix used for certain commands the client can issue and notifications that the client may receive. Generally is used in special cases such as for system notifications that shouldn't be displayed to the user.

Events, logging and development

The client communicates with the server and activates certain functions by calling "events". These all have a standard naming scheme and have a standard way of both receiving data and outputting information.

All data in is sent as one dictionary. The key of the dictionary should correspond to input data such as "username" or "post_id". The client can also define what data it wishes to receive back more specificity by specifying an "items" list. Your typical input dictionary may look like this:

data = {'username': "James", 'date': "19-12-22", 'items': ["post_id", "date"]}

The inputs are managed carefully through use of class properties on the server side. These act as an interface for user input but validating data is in a non-malicious form then testing to see if data (such as a username) exists in the database.
The inputs will also assume a few default values even if nothing is provided. For instance, when getting a post, a client can send no inputs, the server will assume they are referring to themselves (no need to specify your own username) and that you want today's post (no need to specify date) it also assumes you want all the available information about said post (no need to specify what information you want).

The inputs are setup this way to make for an easy and intuitive developer experience and make the server resistant to malicious inputs. This means a programmer going to develop a custom client (perhaps to make it more accessible to those with visual impairments) don't have to spend lots of time reading documentation for what inputs they need to provide to the server to get the needed data.

Another example of this intuitive design is in the team's information. To get information (say the name) of a team the server needs a team_id to make the SQL query. So, the client can either provide the specific team_id, the occupation_id, the username of a user in said team or no information at all (if the logged in user is in the team wanted). The idea is, if the data can be correlated to the specific ID needed it will be and so can be used by a client developer.

The events also follow a standard naming scheme. An event will generally start with what data you are targeting for instance profile, post, or team. Followed by get, set, or delete. Often there are additional nuances such as friend_delete_request, to delete a friend request. Overall, the base get, set and delete can be used with almost all data types and again make for an easier developer experience, although granted some more nuanced functions may need to be looked up in the documentation.

The server also uses a system of "status messages" which indicate the success, failure or potential problem with any event called. Each status message is received by an event on the client side and contains 3 pieces of information.

- Time – A date and (human readable) timestamp of when the status message was created.
- Level – Either INFO, WARN or FAIL. Indicates the general idea of the status.
- Message – A more specific message about exactly what went on for instance "Post(s) successfully fetched" or "Post couldn't be created invalid, or no data provided".

Each status message is also recorded in a sever side actionslog.txt file this log records both the status, event being called and the user_id of the user calling the event. This can be very helpful at debugging a complex issue caused by scale of users. Status messages can also be used as messages to the user themselves to assure an action had an effect in times where the UI can't provide a clear answer. Status messages will also be used by client developers to enable them to debug without opening the server log.

Status

To keep active communication with the client as easy as possible, creating status messages is also easy. The clients access all events through the "handler" classes, these classes all inherit from a root class and call their private methods through the "root handler". This root handler also creates the status message interface object.

The status system is made up of 3 classes, a "log" class which enables the basic functionality of log messages and formatting and so is the parent class of the other 2. The next class is called "status_interface", this acts as an interface which holds some technical

data about routing the status message and attaching some metadata, it also stores status messages on a secondary server log called the “actions log”. The 3rd part of the system is the “status” class, this is the class which status messages are made from. It takes 3 initialisation arguments: The status level, the status message, and the status interface. This status interface is passed as an object of the status_interface class.

Why I mentioned the structure of the handlers earlier is because the root handler method defines a status interface and subsequently adds it as an attribute called “statface”. This is assigned as an attribute of the table class being used as well as an attribute of the handler. This means at any point in either the table class or the handler’s, creating and sending a new status message is as simple as:

status(“INFO”, “This is a standard status message”, self.staface)

Everything else is left to be automated by the status class and status interface. Originally the status class was coded without an interface, and it required 6 init arguments and 2 method calls just to send a status message. This newer system allows for simple status message creation throughout the entire user facing code.

Notifications

As briefly mentioned before, for client notifications I have used “ntfy” which is an implementation of Unified push. This is a set of open standards used for push notifications and is compatible with all devices, see appendix for link. “ntfy” is a server application that I would recommend running on the same server as the BeOpen instance. Its very easy to setup through the docker container and can be up and running within a couple minutes. This keeps to the base objectives of BeOpen the first of them being that its sovereign.

Ntfy was chosen over any other system because of its sovereignty to the organization. If the organization is already willing to host their own social media platform a lightweight notification system isn’t much more of a stretch. Its also platform agnostic, it can work across both desktop and mobile. This has the advantage of not having to collect telemetry from clients just to send them a notification.

The only downside to using ntfy is the users must (if on mobile) download the ntfy app alongside BeOpen. The first-time login page instructs a user to do this and displays to them their “topic” name. A ntfy “topic” is essentially a notification stream, is someone knows the exact resource location of a topic they can also receive all the notifications. This is not a security flaw though since obscuring the topic names is easy, in my system topics are simply called “<username>-<first 8 characters of user_id>”. For example, if your username is “john” your notification topic is called: “john-a52b2jk8”. The chances of this topic being guessed and the fact that no sensitive information is enclosed in notifications means that this security is considered plenty. Element (an open source end-to-end encrypted messaging app) also uses ntfy for their notifications in this very fashion and consider it more than secure.

On desktop these notifications will be received after logging into and allowing notifications from the organisations respective ntfy site.

So overall for ease of development, security and sovereignty ntfy and the Unified Push protocol as a whole was an ideal solution for client native notifications. However the “in app” notifications are still available and use BeOpen’s web sockets for sending and receiving.

Distribution and Hardware support

Client

Android

On android apps can only be run if coded in Java or Kotlin and are packaged into APKs. However, it is possible to “translate” python code into an executable APK that can be run on an android phone. This was the main reason for picking Kivy as the UI library since it supported all platforms and is a sister project to “Buildozer” and “Pyjanus”. Buildozer is used to package python into APKs. Pyjanus is a python library for interacting with the Android API for things like the camera. Luckily, I personally don’t need to interact with Pyjanus since Kivy handles that. The only thing in the app that does need me to consider what platform I’m on is accessing the filesystem. This is as simple as importing the app path from the python android library, and making sure to use this “root path” anytime I interact with the filesystem.

Desktop

As said before Kivy is cross platform and so the only difference between the desktop and android client is how they interact with the filesystem. In terms of the UI, they are identical. This makes for an easier user experience since they won't have to re-learn the UI. The desktop version can be packaged into a .exe for windows using “auto-py-to-exe”, and into a flatpack using the “flatpack-builder” for use on (most) modern Linux distributions.

I did successfully package my application using both tools for Windows and Linux. I did not package a program for MacOS, (despite the python and UI libraries being fully compatible), since I have no way of testing the package once done. Additionally packaging for MacOS or IOS is near impossible on a non-apple platform.

IOS

IOS apps can only be compiled using a MacOS system. I do not have access to a MacOS system and so was unable to compile an application for IOS. Additionally, IOS lacks sideloading and requires a payment to apple to publish apps on the app store.

Theoretically though the client can be compiled and used for IOS, however I have no way of testing its performance and I haven’t attempted to implement platform specifics (such as camera access, image saving, or database storage).

Server

In both cases the cases below, there should be plenty of storage space available to the server. Since the server stores images, the server's data can start to take up a large amount of room depending on the scale of the organisation it is deployed to.

Bare-metal deployment

The python main file and the modules directory containing the imports can be run on bare metal. This should work on windows (untested) and works on Linux. Additionally on a Linux platform you can turn the running of the python scripts into a background systemd process. It should also be noted that a bare metal deployment could be unstable since it will have to utilise the host OS python install which (if modified) could cause the server to break or crash. A requirements.txt is included with the server files but an auto-installer is not. This means for an admin to set up a bare metal deployment they would have to manually download all files and folders into a suitable section of the filesystem with the correct write permissions. They would also have to manually install the correct python packages and versions. For these reasons a bare metal deployment is not recommended as its harder to setup, harder to maintain and less secure.

Docker deployment

A docker container is also available for distribution which makes running the server on any platform relatively simple. This docker container may be Linux exclusive however since the Windows version of docker is still limited in its capabilities. This was done using a DockerFile and so can be run using docker-compose, it is supposed to be run with a shared "./data" volume to allow administrators to configure the config file and have access to the logs. However, docker also supports the output of the log using

docker container <container-name> log

The docker-compose (or docker run command) should expose the port 9999, or if using a reverse proxy exposing 9999 as a virtual port. The port to be exposed can be configured however using the config file.

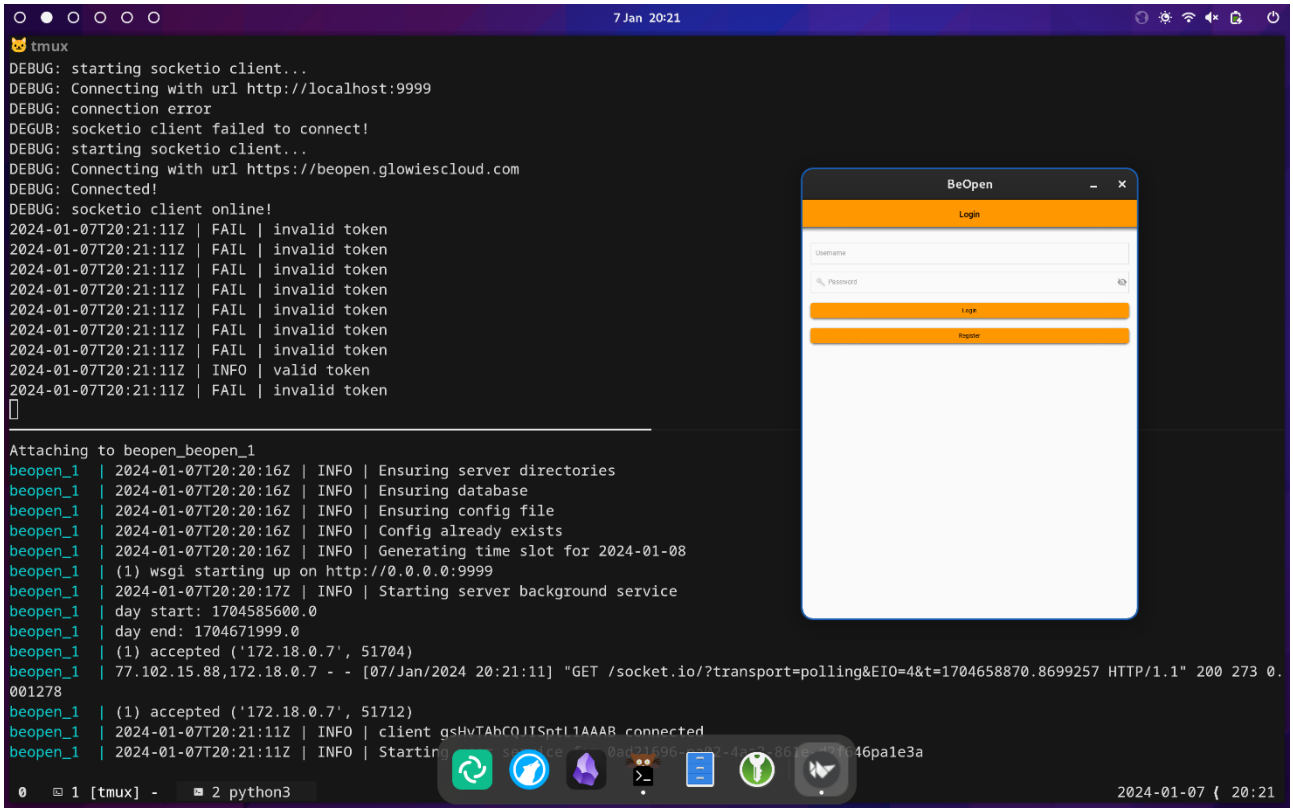
Docker also provides some security advantages as well, since the server is running in a containerised section on a slimmed down OS (alpine) there is limited attack surface when compared to a bare metal server potentially running multiple applications. In these ways the BeOpen server is at less risk of being compromised but additionally containerising BeOpen can prevent a server being compromised via BeOpen. Since the attacker first must compromise BeOpen, then the docker OS and then somehow escape the container to compromise the host OS.

Overall docker deployment is recommended over bare metal for improved security, ease of use, stability and re-producibility.

Deployment test

I tested both a bare-metal deployment and docker deployment on my raspberry-pi 4. Both times the system was being routed through a nginx reverse proxy that also enabled HTTPS, receiving certificates via LetsEncrypt. BeOpen was utilising port 9999 in both cases. I utilised my own domain, in this document ill refer to my domain as “mydomain.com” for security reasons, so the service was being hosted at “https://beopen.mydomain.com”. To test the functionality of the server I used a client running on my laptop.

Below are some screenshots of an SSH session (into a raspberry-pi) showing the logs of the docker container and a BeOpen client, which is connected.



Security

Levels

The systems security is mainly managed using “levels”. The 3 levels are listed in the table below (all levels below presume you are not a team leader):

Name	Description
Member	The basic level, users are assigned this level by default and most never change. Permissions allow you to delete your own content and your own content only. You can view friends posts and team members posts but not allowed to view any posts from outside your team or friend group. Additionally, occupation changes must be approved by a manager or admin. Members can also not view, accept or reject occupation change requests (they can view and cancel their own however).
Management	This level is designed for the upper management of an organisation. Management has additional privileges for instance they are allowed to delete any user’s content and can manage occupation change requests. Consequently, managers do not need to create occupation change requests but rather can just set their occupation (taking effect immediately). Management can also edit/delete teams (names, descriptions, and leaders)
Admin	Admins have all the same permissions as management but with a few additions: The server supports admin’s ability to edit user profiles, posts and comments however the client only supports admin’s ability to edit user profiles.

Any client is allowed to connect to the server and on initial connection a client is given the level None. Clients with this level only have access to a handful of miscellaneous events and register. This allows for client-side registration and then once a client is logged in they are assigned their respective level and from then on have access to all other events.

Team leaders

Any users of any level can be made into a team leader by an admin or management. The user must be in the team they are being promoted to lead. Team leaders have additional privileges over the posts and comments created by their team. Team leaders can delete team members posts, comments, and impressions (however deleting impressions is not supported on the client side).

Credential storing

A user is made up of 4 key bits of information:

- User ID – A randomly generated Universally Unique Identifier (UUID) not fully shared outside of the server.
- Username – A unique identifier (within the server instance) used by clients to identify users.
- Password – Secret phrase or word kept by the user to keep their account secure.

As mentioned before user IDs are not shared outside of the server this is done to act as a layer of obfuscation in case of a database leak, since the user ID is used throughout the server to identify the user. When a client passes a username to the server the server immediately converts it to a user ID using the “user_id” class. Additionally, as will be explained next, the user ID is utilized to secure the passwords, without the addition of corresponding user IDs a leak of all the database password entries would be useless.

Passwords

A password is setup on account registration, once the registration data has been received by the server and the data verification checks have taken place the password is first salted using the freshly and secret (outside the server) user ID, which is appended to the end of the string. Then using a custom hash algorithm (for additional obfuscation) the string is hashed for storage and the variable is re-assigned to an empty string (to minimise the damage of a ram dump attack).

Database encryption

The database itself can be encrypted at rest; this can be configured in the configuration file. There are many options in the configuration file for changing the paths of certain files etc, but the main options are “EncryptDatabase” (with can be true of false) and “ShamirSecretSharing” (also true or false). The “Guide to encrypting the database” (included as a markdown file) goes into detail about how to set up encryption of the database, but I’ll give a brief explanation here as well. The administrator provides a master password in the encryptionconfig.txt (path of file can be changed). This password is then used to encrypt a key.txt that stores the actual key that the database is encrypted with. This master password is then deleted automatically from the server’s filesystem. If Shamir Secret Sharing is also enabled, then using the values specified in the configuration for the number of shares and minimum shares required the server will create several shares as individual text files. If the correct number of shares are provided, the server can derive the master password and decrypt the databases encryption key.

Essentially the administrator can have both a master password which can be used to decrypt the database, or they can use a combination of the shares to decrypt the database. Once the database is decrypted all the other events open up for use. But if a

server has encryption enabled the server will launch in “decrypt” mode where the only event available is the “decrypt” event. All other event calls will return a status message saying they are currently not available. Any active client can call the decrypt event even if not logged in, but they still need to provide the correct credentials to unlock the database. Additionally, once decrypted the client will still have to log in and by no means has no further access to the database directly unless they have a terminal session on the server.

The Shamir secret shares can be used to distribute the master password, this system was implemented as a “backup” method. Say the previous administrator with the master password left the company or suddenly passed away the organisation would have no access to the database and lose their social platform. The immediate solution to this many might give is to simply tell lots of people the secret so that multiple people can use the master password for decryption, but this simply widens the attack surface for social engineering. This is the problem PayPal faced back in the early 2000s, I highly recommend reading the blog post below:

https://max.levch.in/post/7242894571440701 44/shamir-secret-sharing-its-3am-paul-the-head-of

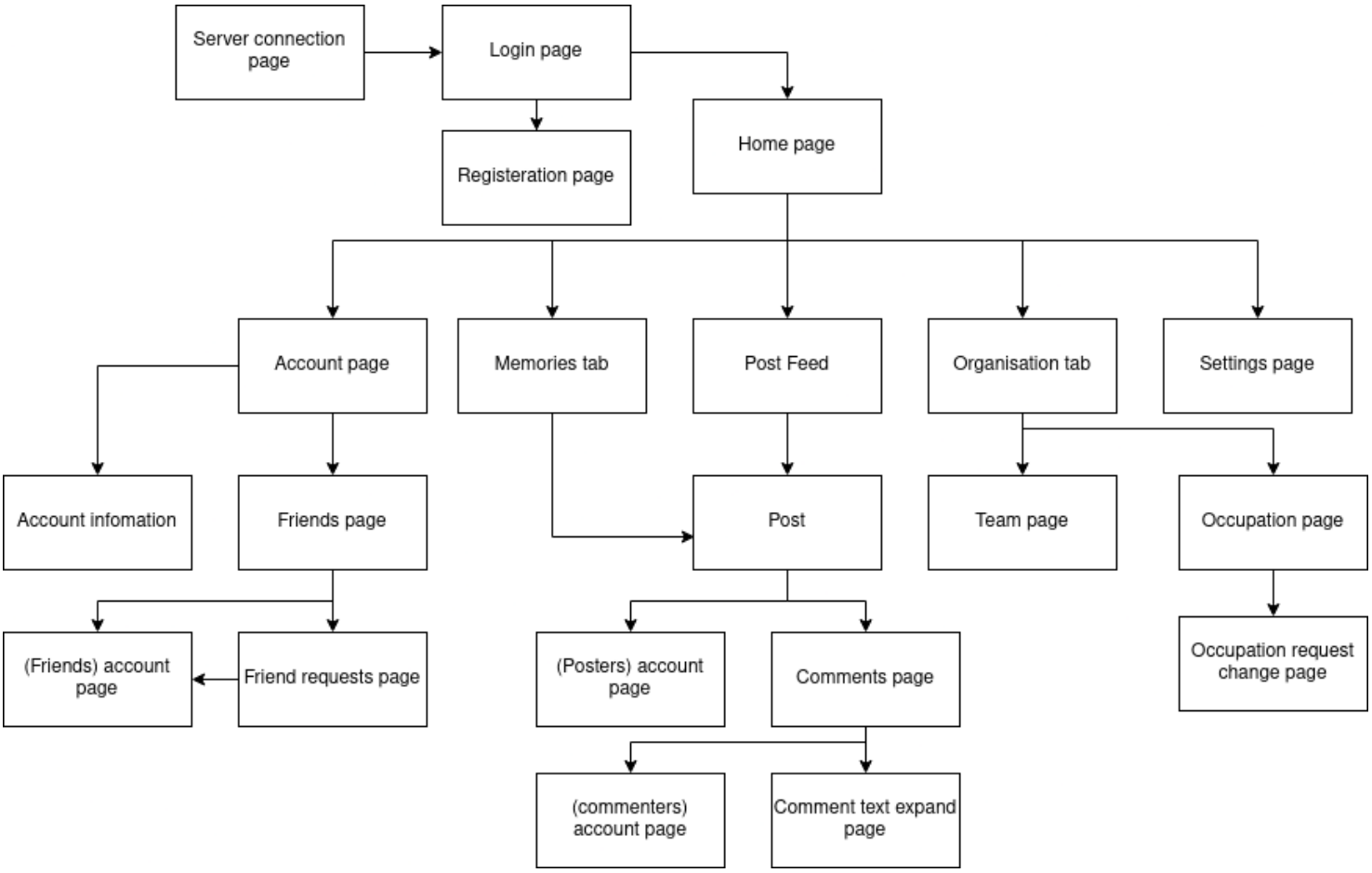
Shamir secret sharing solves this problem in a very mathematical way (which I go over in the algorithms section), here I will just go over how it works for the users. As said before if enabled the server will generate a series of text files, each text file is a “share”, it is the admins job to distribute these shares however they please and delete them from the server. When these shares are generated, the admin can set the minimum amount required to decrypt the database and the number of shares to be generated. What this means is that say the minimum amount required to decrypt is set to 3 the administrator can then generate 10 shares and hand them out to 10 people within the organization. If any 3 of these people come together and enter their shares into the decrypt event, they can decrypt the database. The essential idea is that an administrator can have the master password but if for any reason they lose this password the trusted people who they gave shares to can come together to decrypt the database and re-construct the master password.

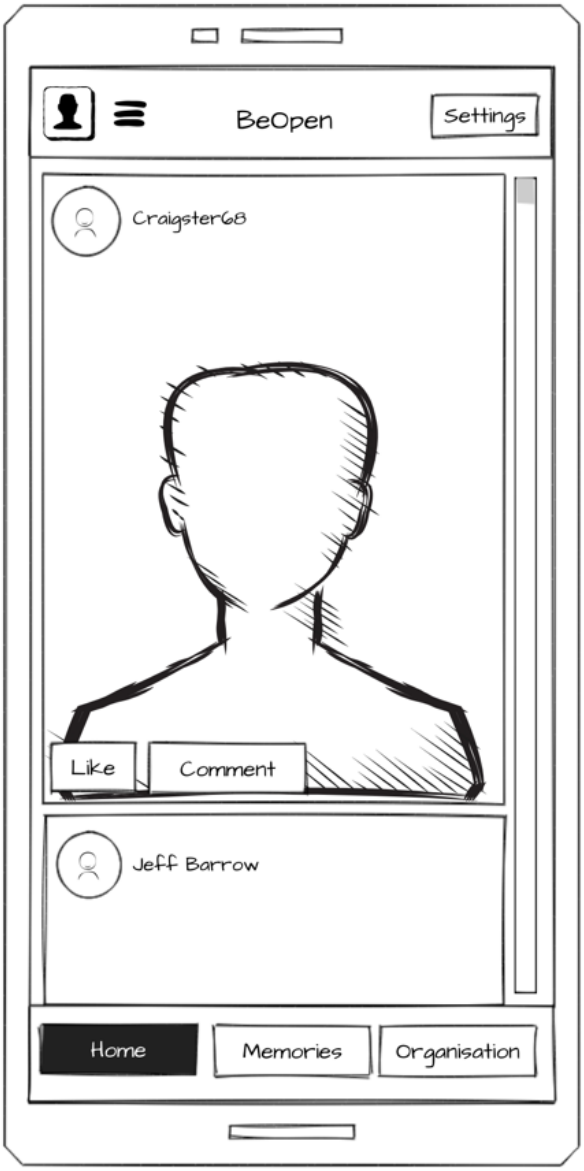
The database encryption scheme itself uses python’s cryptography library to generate a “fement” key. Then the database’s raw bytes are read, encrypted and stored in another file called “.cryptdatabase.db” (by default). This reading of raw bytes and using python’s cryptography library manually was the preferred method for a few reasons. Firstly, decreasing dependency on pip packages is a good idea since recently the python package library has seen a rise in malicious code taking over repositories and being used to distribute malware. Additionally, there are no good python libraries that support the encryption of a SQLite database, every library which did exist has been deprecated or not updated in 5+ years and so is not recommended for use. Thirdly it kept the code overhead minimal, reading and writing raw bytes keeps the code simple and safe.

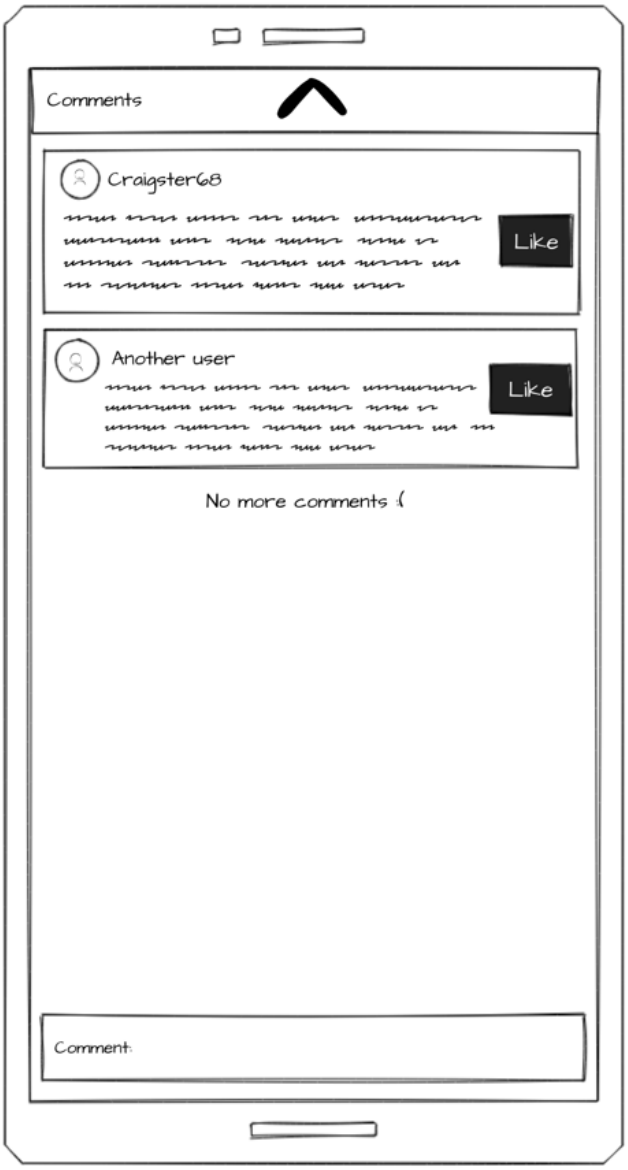
The one downside to my method is that 2 versions of the database are kept at any one time, the encrypted version and (when the server is running) the unencrypted version. This means that the server must have enough storage to store 2 versions of the database. The other reason this method is slightly flawed is we are reading raw bytes every time the database is encrypted. So, it has to re-read and re-encrypt the entire file, even if just one thing has changed. This means once a deployment becomes large enough the server could take a couple seconds to shutdown (depending on the hardware). However, neither of these factors are huge problem simply due to the size of the database. Even for large deployments, the database remains small. Even serving thousands of clients the database shouldn't grow larger than at most a couple GBs and if only serving a few hundred no larger than a few 100MBs. Additionally reading and writing raw bytes is extremely fast even low-end hardware can read and write GBs in just a couple seconds.

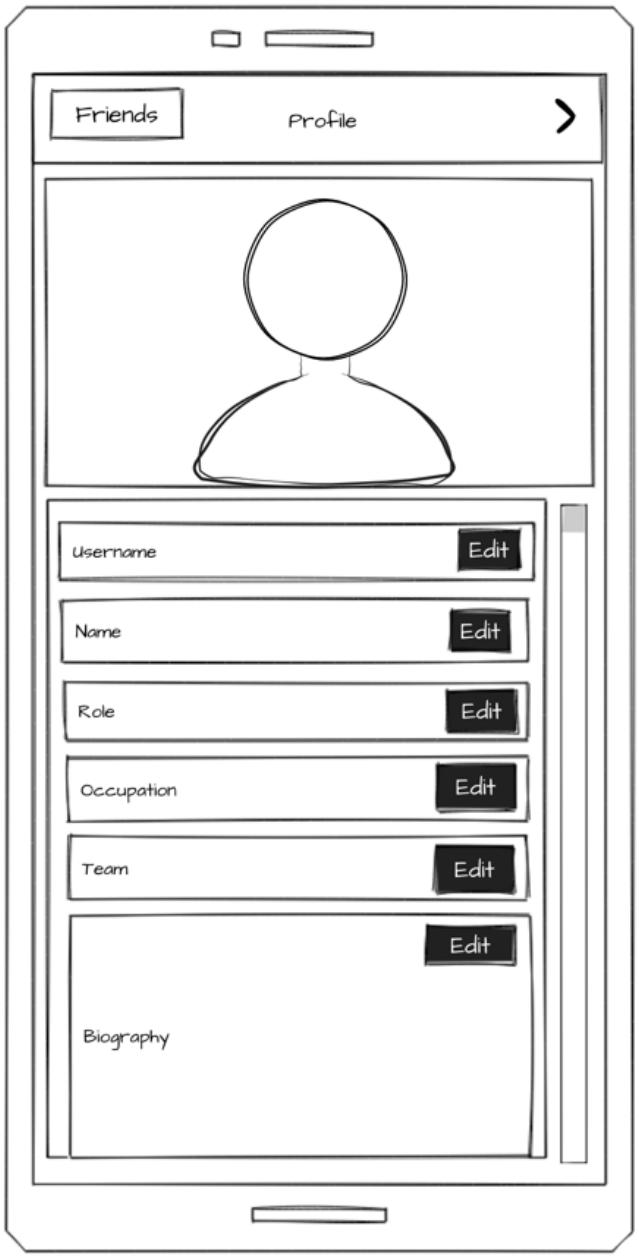
Overall, it was decided that this encryption method was more than suitable for this application. The security benefits provided by encrypting the database at rest means even if an attacker compromises the system all the administrator has to do is shutdown the BeOpen server (which they can do from a logged in admin client) and then all the attacker is left with is an encrypted database. Decryption of the database (if set with a correct password) should be computationally secure especially for this type of data. At the end of the day this is simply a social media platform and the computation power required to decrypt the database (by brute force) wouldn't be worth it even to a state actor.

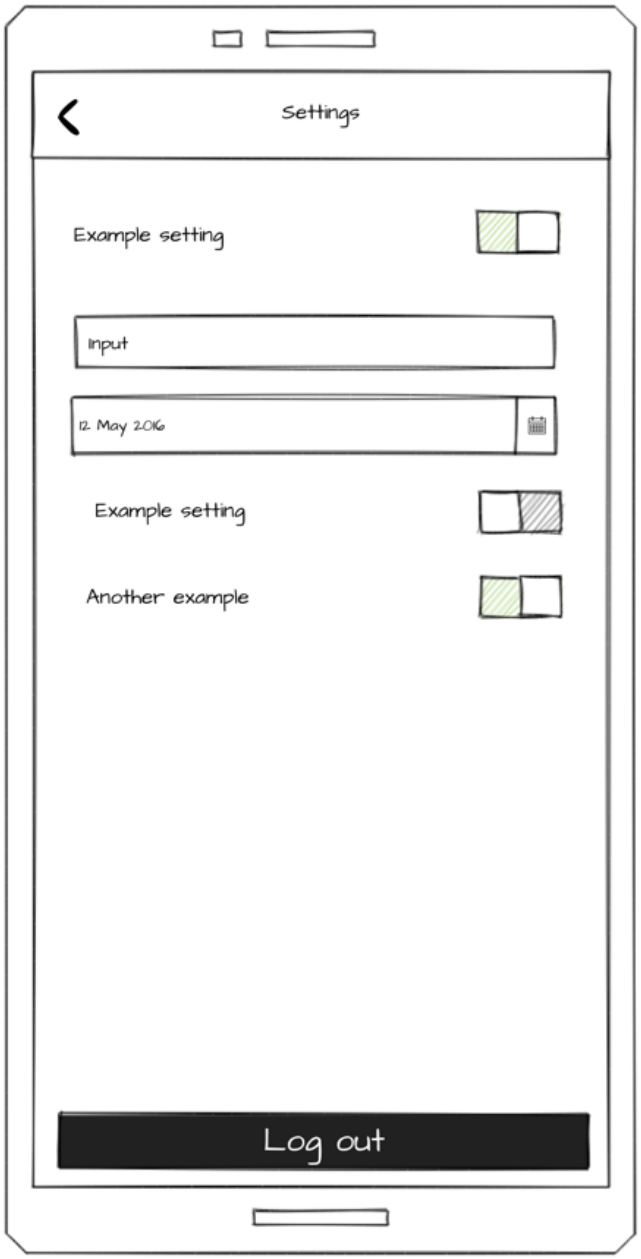
User Interface

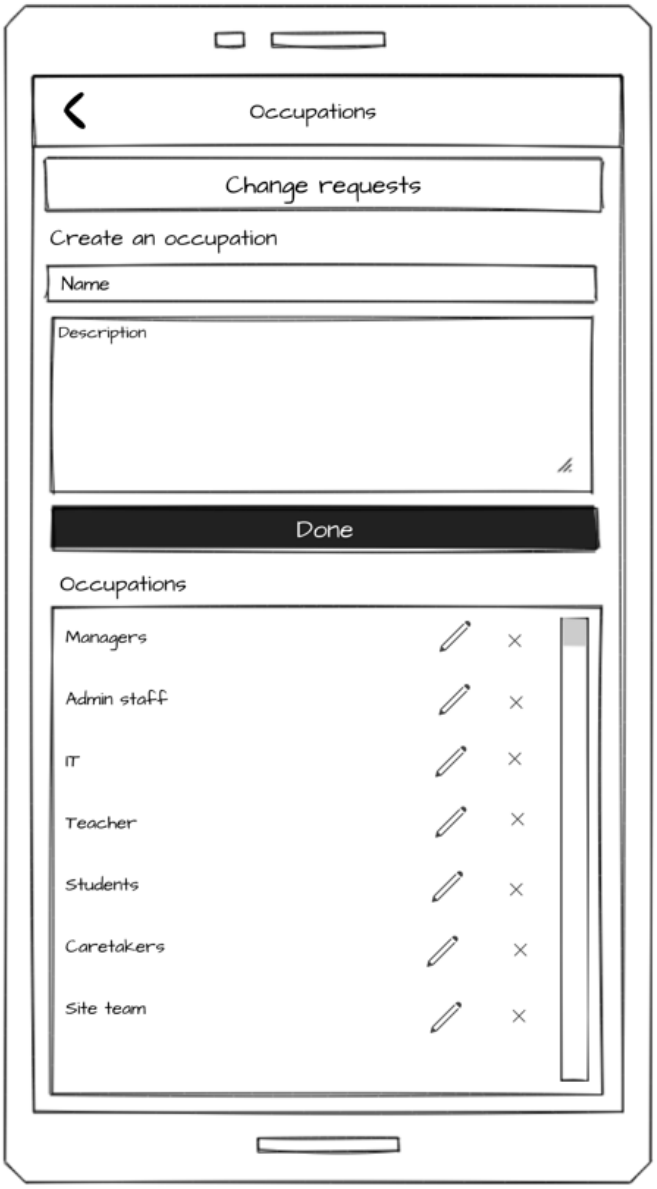


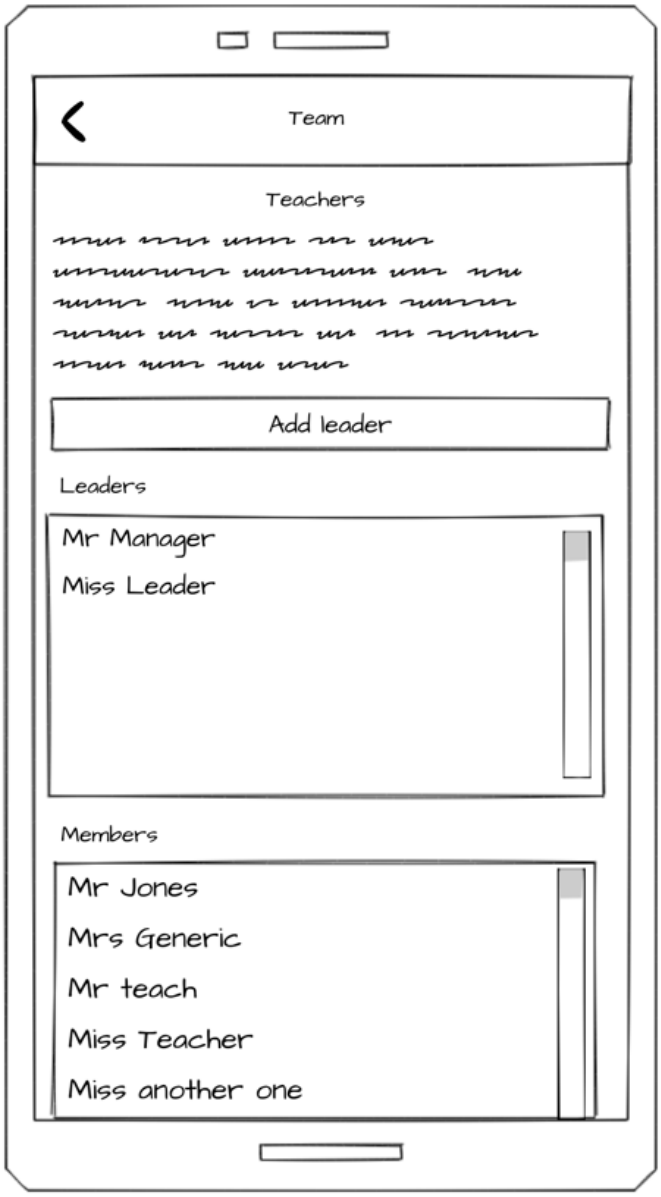

BeOpen
Settings
Craigster68
Like
Comment
Jeff Barrow
Home
Memories
Organisation

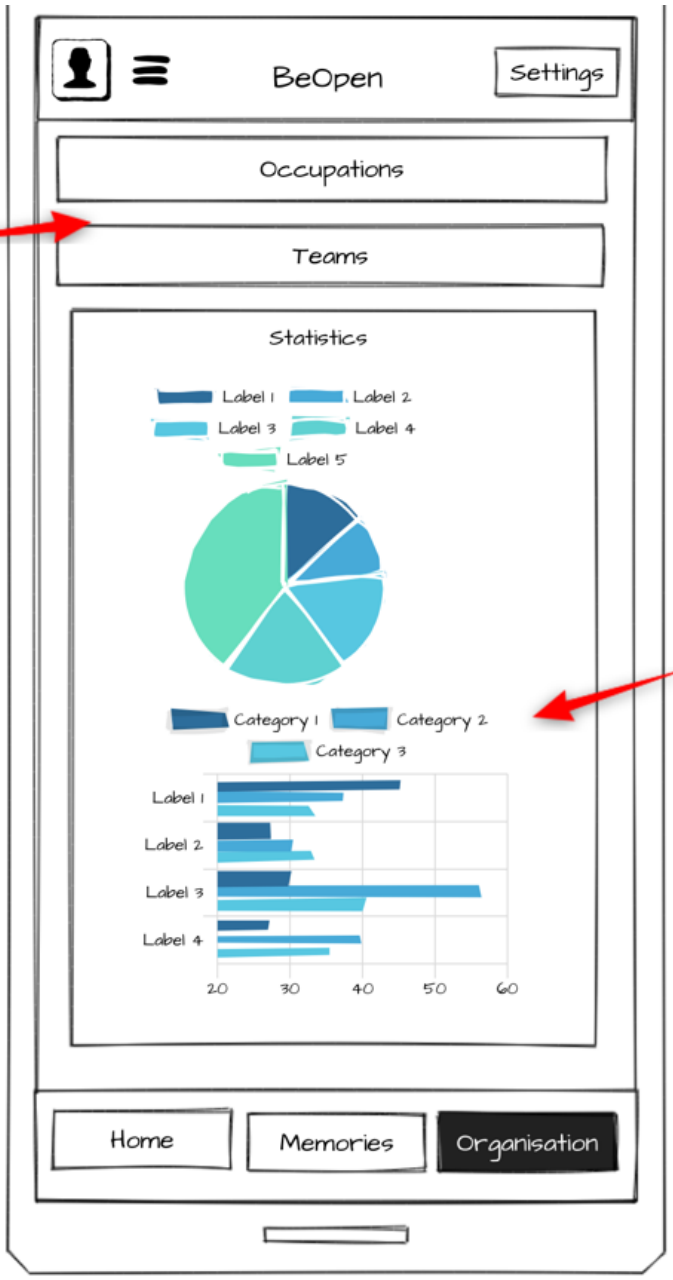

Comments
Craigster68
Like
Another user
Like
No more comments :(
Comment:


Friends
Profile
Username
Edit
Name
Edit
Role
Edit
Occupation
Edit
Team
Edit
Edit
Biography


Settings
Example setting
Input
12. May 2016
Example setting
Another example
Log out

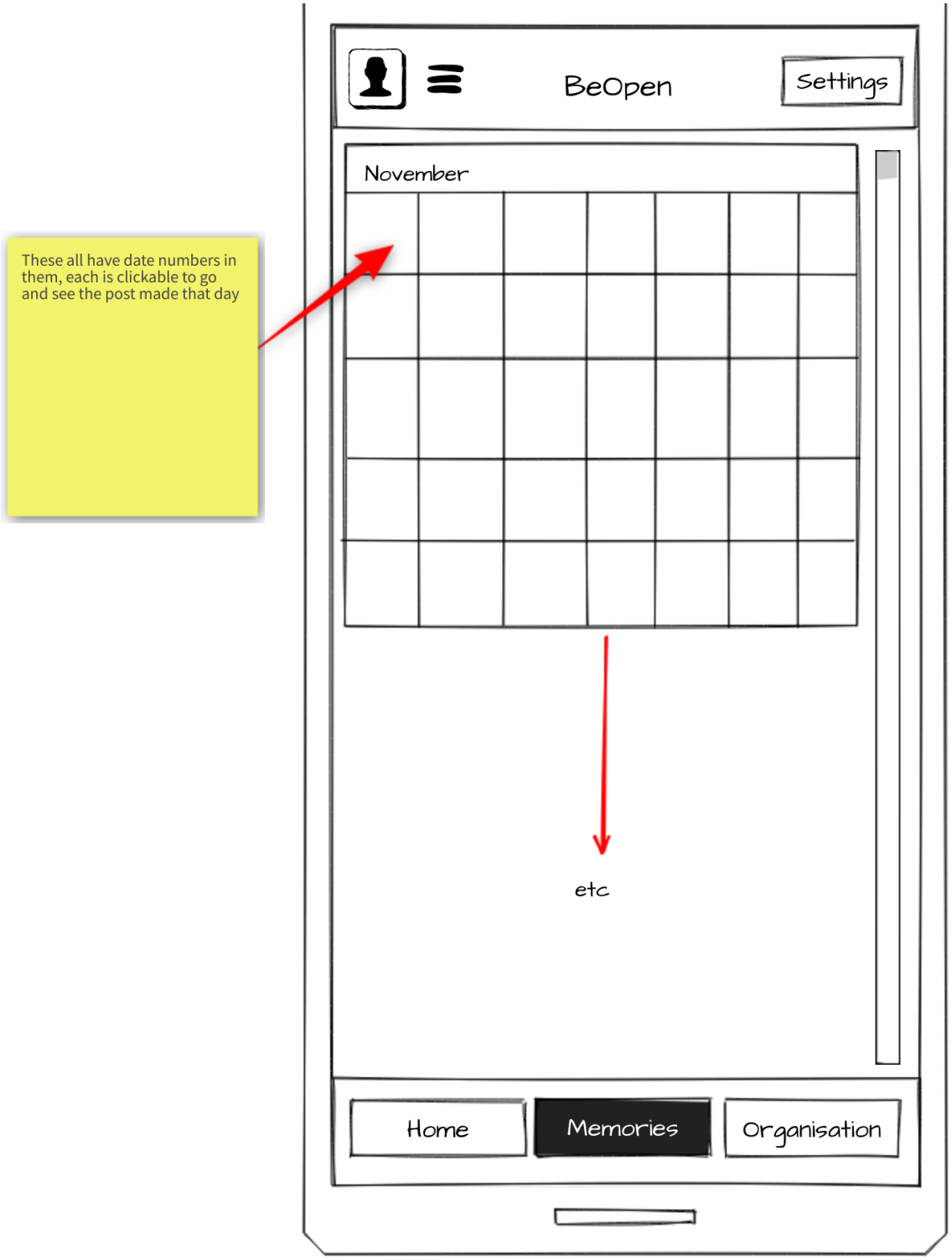

Occupations
Change requests
Create an occupation
Name
Description
Done
Occupations
Managers
Admin staff
IT
Teacher
Students
Caretakers
Site team


Team
Teachers
Add leader
Leaders
Mr Manager
Miss Leader
Members
Mr Jones
Mrs Generic
Mr teach
Miss Teacher
Miss another one



These buttons take you to the teams and occupations page respectively, If a user is not an admin only the teams button will appear

Only an idea likely wont be in the final implementation. Only done if lots of spare time


These all have date numbers in them, each is clickable to go and see the post made that day
BeOpen
Settings
November
etc
Home
Memories
Organisation

Design

The Design of the interface is meant to feel familiar to other social medias. The homepage displays the user a post feed, each post displays the username, a clickable profile button/picture as well as like and comment buttons at the bottom. It's very similar to other social medias. Its utilising the more modern design of "floating" action buttons with the picture taking up the entire area of the post with the action buttons (like, profile etc) are displayed as white icons above the photo. This is opposed to having each post have a chin and a header where such information is displayed. This makes the feed a mix between Instagram style free scrolling (not locked to seeing one post filling the frame at a time) and a TikTok style post card. I did want to have a locked scrolling, but I felt this may feel uncomfortable for the slightly older user base, the system is supposed to be used in workplaces, people of working age are generally more familiar with platforms like Facebook. Facebook utilizes a free scrolling method and so I kept this same free scrolling method, so that these users would easily start using the app.

Its designed this way to feel familiar to any user hopefully allowing anyone to pick up the app and immediately be able to start engaging with its basic functions.

Buttons (apart from top and bottom navigation buttons) take up the breadth of the screen. This was done as to keep the layout of the page as linear as possible. This minimises confusion and makes the app more accessible for one handed use, if you're either left-handed person or a right-handed person. The homepage itself consists of 3 main tabs; this is where the user spends most of their time. If it weren't for the common design of social media to have your own profile in the top left it would have been made a main tab along the bottom. However again like said before I wanted the app to feel familiar so left the profile page accessible from the top left only.

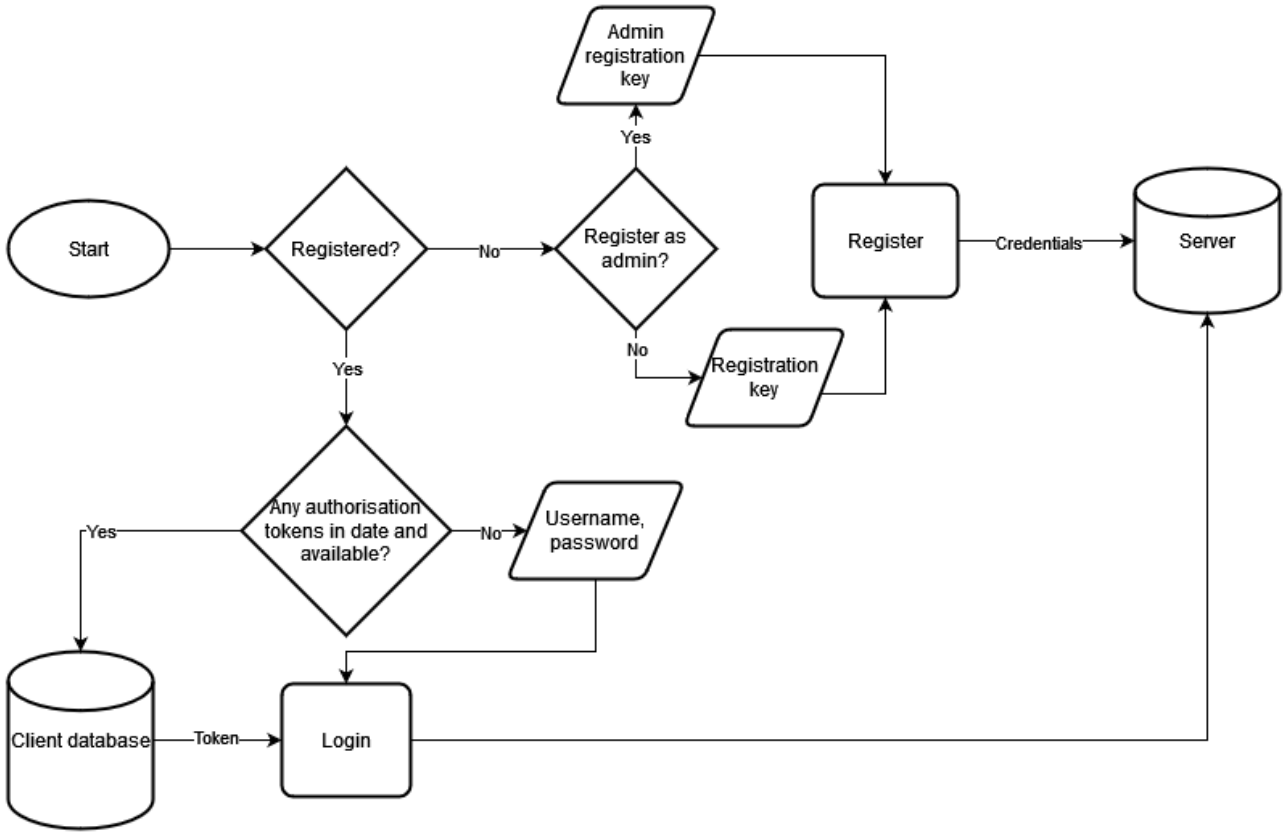
However other people's profiles are always accessible by clicking on their username anywhere in the app. Whether their a fellow team member, team leader, sending you a friend request, making a comment etc, if you see another user's username you can click it and see their profile. This again is very familiar to other social media and makes interacting and learning about other people in your organisation easier than ever.

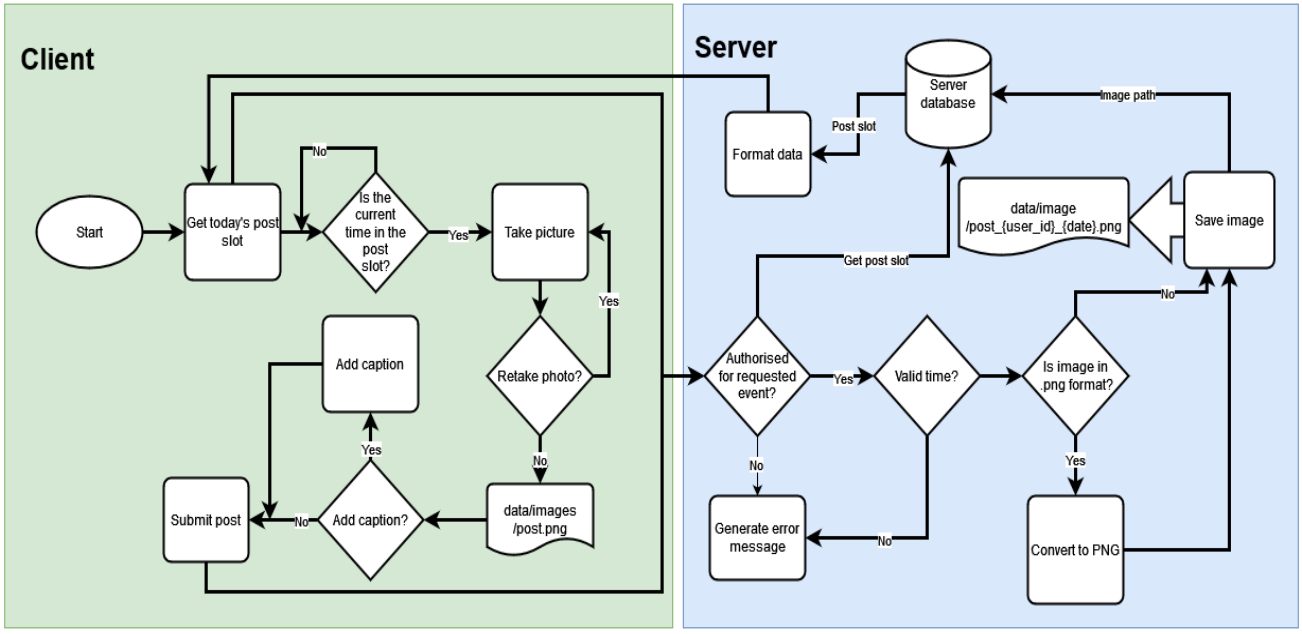
Overall, the UI is supposed to be initiative and familiar to anyone who has used any form of social media before. Preferably the final UI will be using a material design theme to keep a consistent modern UI. Material design will also easily integrate BeOpen (in terms of looks) into any android device. Since most android default apps use this theme BeOpen won't look out of place compared to the messaging app etc.

System diagrams

Flow charts

Login/registration diagram



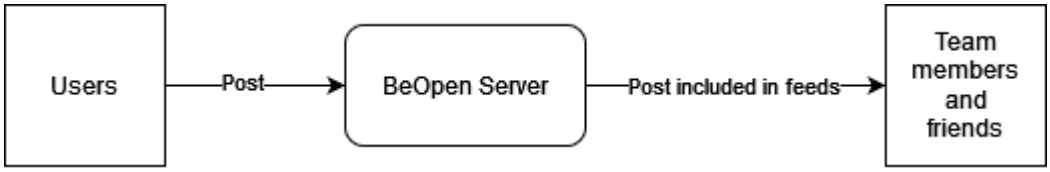


Posting

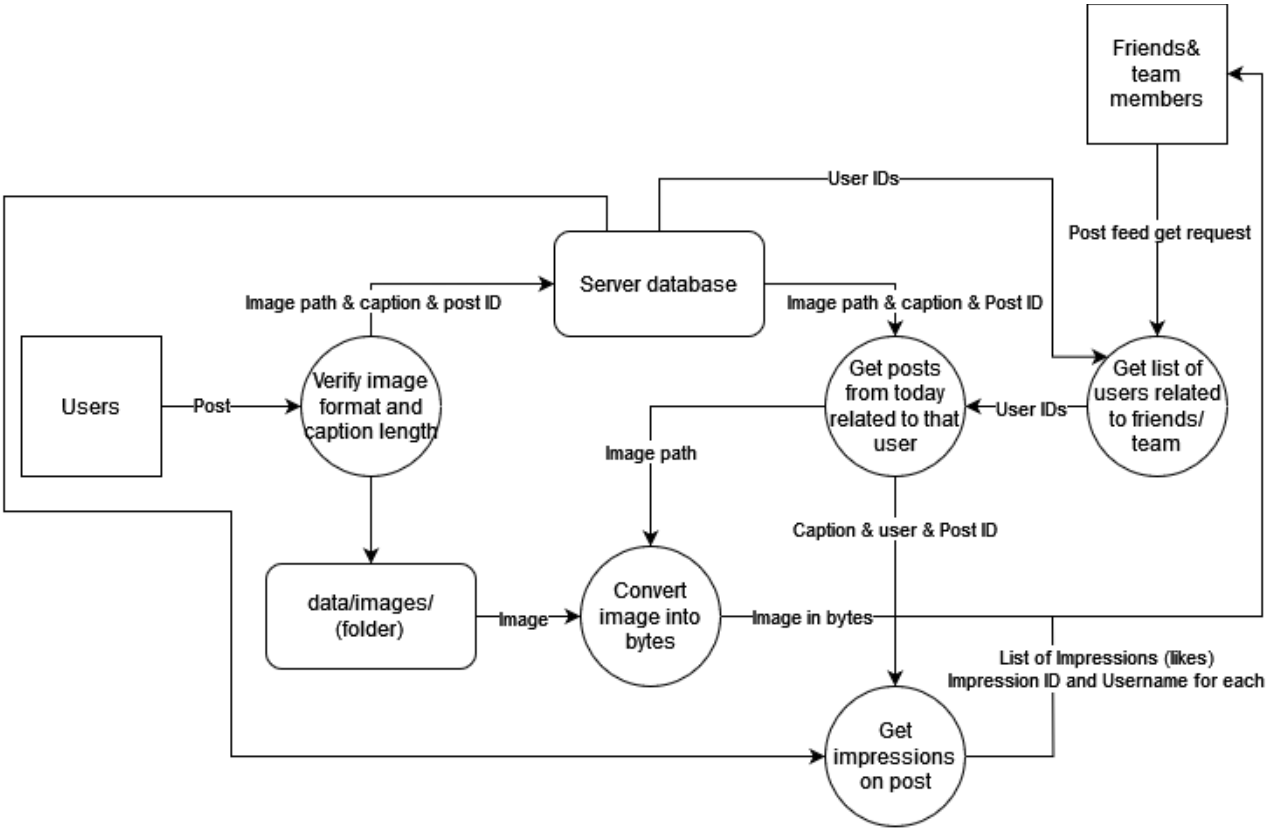
Data Flow Diagrams

Posting

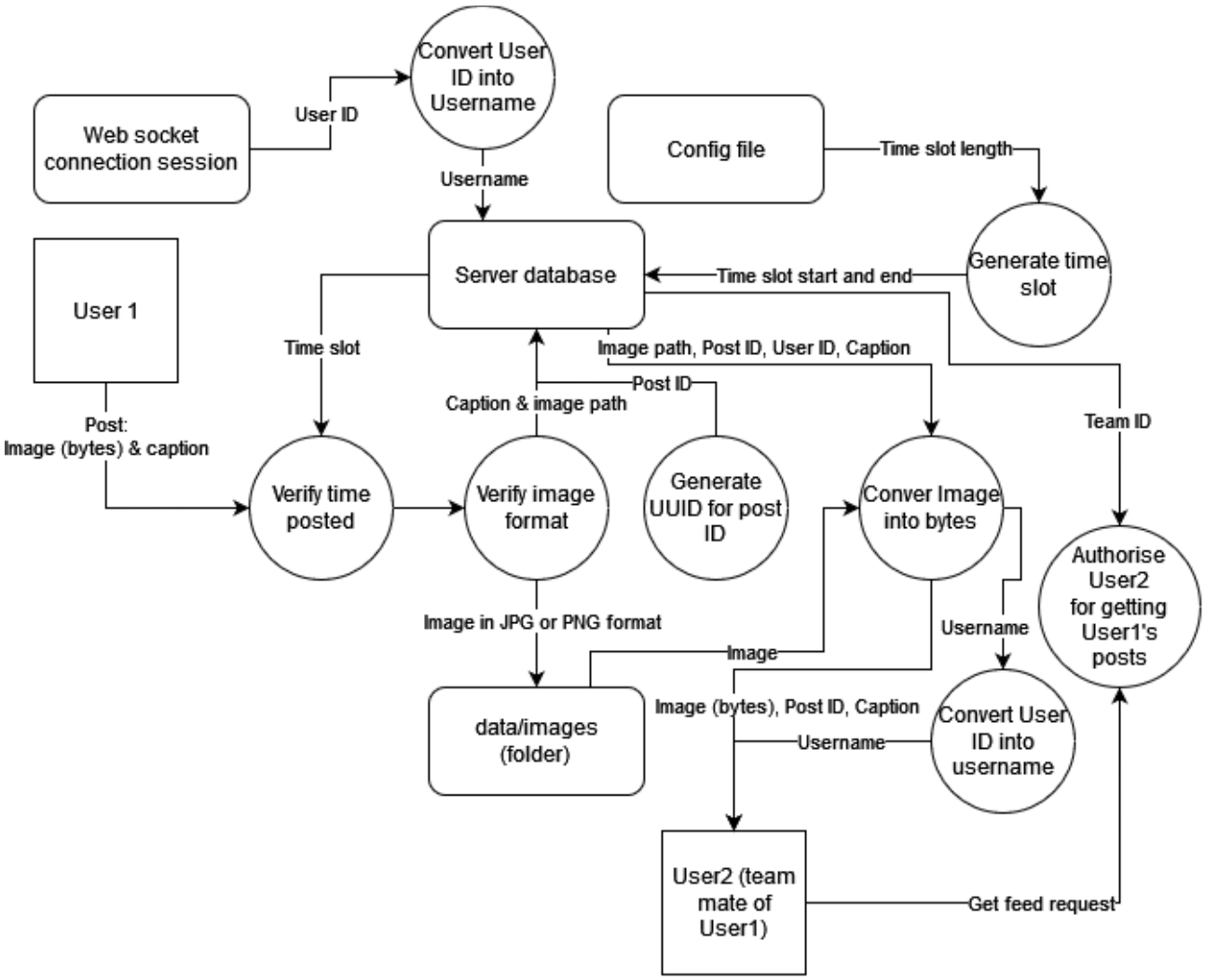
Level 0



Level 1

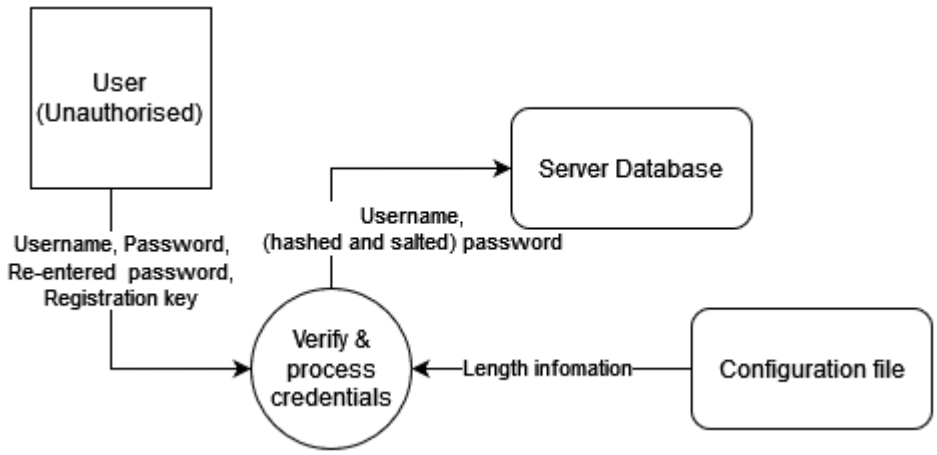

Friends& team members
User IDs
Post feed get request
Server database
Image path & caption & post ID
Image path & caption & Post ID
Users
Post
Verify image format and caption length
Get posts from today related to that user
User IDs
Get list of users related to friends/ team
Image path
Caption & user & Post ID
data/images/ (folder)
Image
Convert image into bytes
Image in bytes
List of Impressions (likes)
Impression ID and Username for each
Get impressions on post

Level 2

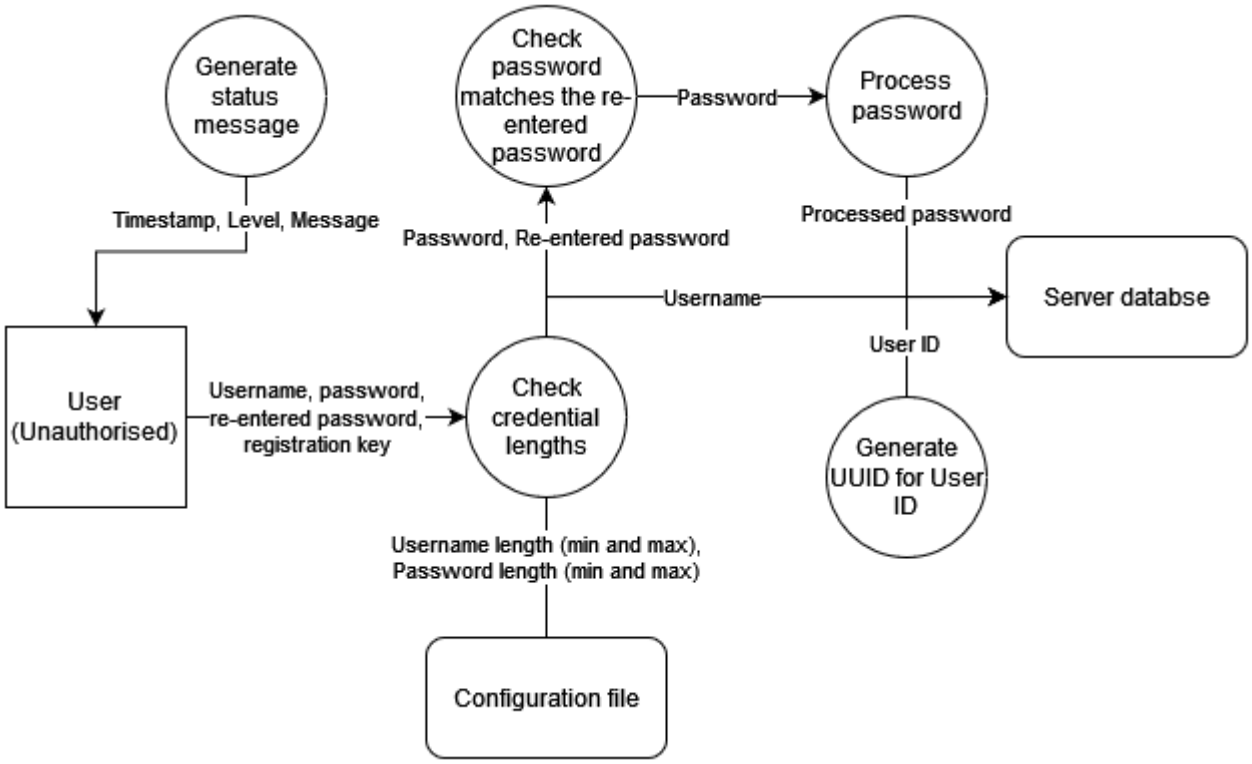


Register

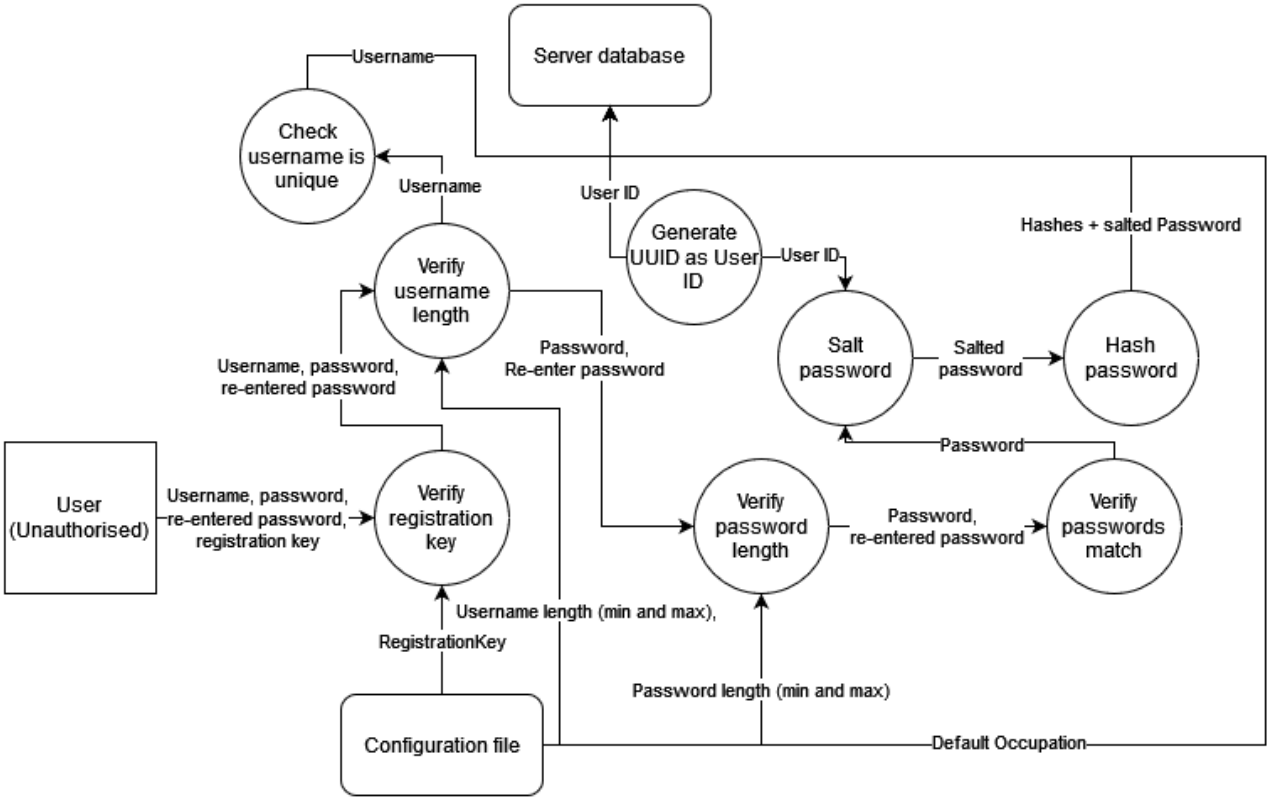
Level 0



Level 1


Generate status message
Check password matches the re-entered password
Password
Process password
Timestamp, Level, Message
Password, Re-entered password
Processed password
Username
Server databse
User ID
User (Unauthorised)
Username, password, re-entered password, registration key
Check credential lengths
Generate UUID for User ID
Username length (min and max), Password length (min and max)
Configuration file

Level 2


Server database
Username
Check username is unique
Username
User ID
Generate UUID as User ID
User ID
Hashes + salted Password
Verify username length
Password, Re-enter password
Salt password
Salted password
Hash password
Username, password, re-entered password
Password
User (Unauthorised)
Username, password, re-entered password, registration key
Verify registration key
Verify password length
Password, re-entered password
Verify passwords match
Username length (min and max),
RegistrationKey
Password length (min and max)
Configuration file
Default Occupation

IPSO Chart Client Side

Input	Process	Storage	Output
Creating post by clicking “post” button: - Photo from camera - Caption	- Validate that post was sent in correct time frame. - Validate caption length. - Save picture to server	Posts table Time_slots table (database)	- Status message
Opening personal profile page	- Get name, username, role, occupation and biography. - Check the user’s permissions for editing each category	Profile table (database) Dictionary (client side, post fetch)	- Display the user information
Clicking the edit button on a profile category (for instance name) And taking input for the new value	- Validate input - Reload profile page	Profile table (database)	- Status message - Changed value
Entering the homepage	- Fetch post data - Save the post images	Array of image paths, and dictionary	- The post image - Like, profile and comment buttons - The (posters) username

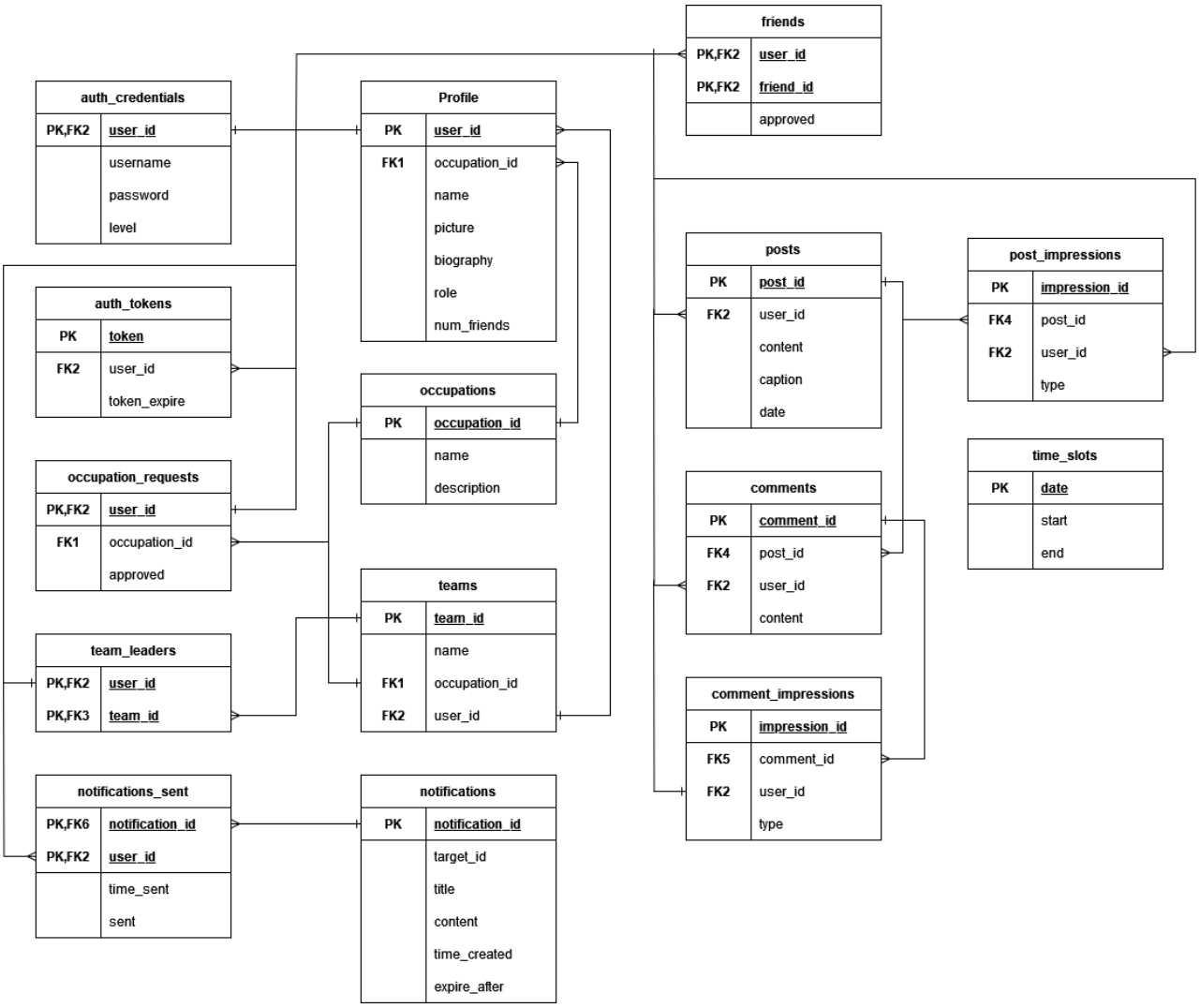
IPSO Chart Server Side

Inputs	Processes	Storage	Output
Registration details: - Username - Password - Re-written password - Registration key	- Validate username and password lengths - Check passwords match - Check for username uniqueness - Validate registration key - Generate user id - Hash and salt password	auth_credentials table (database)	- Successful registration message
Login details: - Username - Password	- Hash password - Fetch correct password hash - Compare password hash with correct password hash - Generate authentication token	auth_credentials table and auth_tokens table (database) Tokens (client_database)	- Authentication token - Successful login status message - Boolean to indicate if logged in or not
- Username of friend	- Verify user exists - Check for existing friend request to and from - Notify the requested user	Friends table Auth_credentials table (database)	- Status message for successfully creating request
- Post id Or - Date	- Verify data refers to real post, user and date - Convert post image to byte data	Post table Auth_credentials table (database)	- Post information (from one of the users own posts)

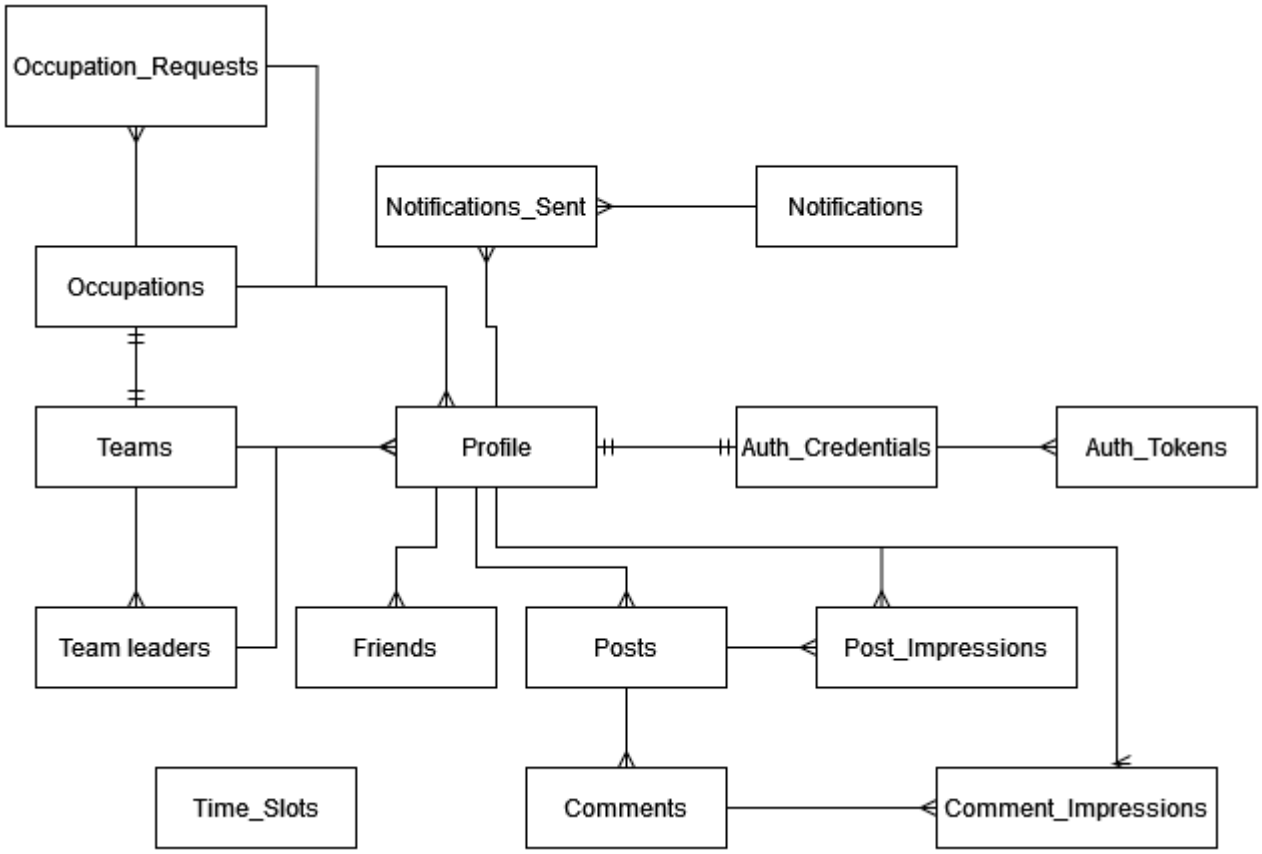
- Post id Or - Date - Username	- Validate the user is Admin, management or leader of requested users post - Convert post image to byte data	Post table Auth_credentials table (database)	- Post information about requested post
- Team id	- Verify that the user is an admin or management	Post table Auth credentials table Teams table (database)	- List of posts and their information
- Post picture - Caption	- Validate that post was sent in correct time frame - Validate caption length - Save picture to server	Posts table Time_slots table (database)	- Status message
- Occupation name - Description	- Validate name and description length - Validate user is management or admin	Occupation table Auth credentials (database)	- Success status message - Updated occupation list

Database

Tables and attributes (including primary and foreign keys), refer to key below


friends
PK,FK2 user_id
PK,FK2 friend_id
approved
auth_credentials
PK,FK2 user_id
username
password
level
Profile
PK user_id
FK1 occupation_id
name
picture
biography
role
num_friends
posts
PK post_id
FK2 user_id
content
caption
date
post_impressions
PK impression_id
FK4 post_id
FK2 user_id
type
auth_tokens
PK token
FK2 user_id
token_expire
occupations
PK occupation_id
name
description
time_slots
PK date
start
end
occupation_requests
PK,FK2 user_id
FK1 occupation_id
approved
comments
PK comment_id
FK4 post_id
FK2 user_id
content
teams
PK team_id
name
FK1 occupation_id
FK2 user_id
team_leaders
PK,FK2 user_id
PK,FK3 team_id
comment_impressions
PK impression_id
FK5 comment_id
FK2 user_id
type
notifications_sent
PK,FK6 notification_id
PK,FK2 user_id
time_sent
sent
notifications
PK notification_id
target_id
title
content
time_created
expire_after

Relationship diagram



Normalisation

All tables are normalised to 3NF except for “Teams” which has 2 attributes that depend on the primary key but both columns are never filled in together. The 3 attributes are: team_id, occupation_id and user_id. Team_id is the primary key, but when creating a team it will use an occupation_id but a group of friends for each user will be counted as a “friends” team and is associated with that user through the user_id column.

DDL

auth_credentials

```
CREATE TABLE IF NOT EXISTS auth_credentials (
        user_id TEXT NOT NULL PRIMARY KEY,
        username TEXT NOT NULL,
        password TEXT NOT NULL,
        level TEXT NOT NULL,
        FOREIGN KEY (user_id)
            REFERENCES profile (user_id)
```

```
        ON UPDATE CASCADE
        ON DELETE CASCADE
    )
```

auth_tokens

```
CREATE TABLE IF NOT EXISTS auth_tokens(
    user_id TEXT NOT NULL,
    token TEXT NOT NULL PRIMARY KEY,
    token_expire REAL NOT NULL,
    FOREIGN KEY (user_id)
      REFERENCES auth_credentials (user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
    )
```

profile

```
CREATE TABLE IF NOT EXISTS profile (
    user_id TEXT NOT NULL PRIMARY KEY,
    occupation_id TEXT,
    name TEXT,
    picture TEXT,
    biography TEXT,
    role TEXT,
    num_friends INTEGER DEFAULT 0,
    FOREIGN KEY (occupation_id)
      REFERENCES occupations (occupation_id)
        ON UPDATE CASCADE
        ON DELETE SET NULL
    )
```

friends

```
CREATE TABLE IF NOT EXISTS friends (
    user_id TEXT NOT NULL,
```

```
    friend_id TEXT NOT NULL,
    approved BOOLEAN,
    FOREIGN KEY (user_id)
      REFERENCES profile (user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
    FOREIGN KEY (friend_id)
      REFERENCES profile (user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
    PRIMARY KEY (user_id, friend_id)
    )
```

occupations

```
CREATE TABLE IF NOT EXISTS occupations (
    occupation_id TEXT NOT NULL PRIMARY KEY,
    name TEXT NOT NULL,
    description TEXT
    )
```

occupation_requests

```
CREATE TABLE IF NOT EXISTS occupation_requests (
    user_id TEXT NOT NULL PRIMARY KEY,
    occupation_id TEXT NOT NULL,
    approved BOOLEAN DEFAULT False NOT NULL,
    FOREIGN KEY (user_id)
      REFERENCES profile (user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
    FOREIGN KEY (occupation_id)
      REFERENCES occupations (occupation_id)
```

```
            ON UPDATE CASCADE
            ON DELETE CASCADE
        )
```

teams

```
CREATE TABLE IF NOT EXISTS teams (
        team_id TEXT NOT NULL PRIMARY KEY,
        name TEXT NOT NULL,
        occupation_id TEXT,
        user_id TEXT,
        FOREIGN KEY (occupation_id)
          REFERENCES occupations (occupation_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
        FOREIGN KEY (user_id)
          REFERENCES profile (user_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
        )
```

team_leaders

```
CREATE TABLE IF NOT EXISTS team_leaders (
        user_id TEXT NOT NULL,
        team_id TEXT NOT NULL,
        FOREIGN KEY (user_id)
          REFERENCES profile (user_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
        FOREIGN KEY (team_id)
          REFERENCES teams (team_id)
            ON UPDATE CASCADE
```

```
        ON DELETE CASCADE
    PRIMARY KEY (user_id, team_id)
    )
```

posts

```
CREATE TABLE IF NOT EXISTS posts (
    post_id TEXT NOT NULL PRIMARY KEY,
    user_id TEXT NOT NULL,
    content TEXT NOT NULL,
    caption TEXT,
    date TEXT NOT NULL,
    FOREIGN KEY (user_id)
      REFERENCES profile (user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
    )
```

comments

```
CREATE TABLE IF NOT EXISTS comments (
      comment_id TEXT NOT NULL PRIMARY  KEY,
      post_id TEXT NOT NULL,
      user_id TEXT NOT NULL,
      content TEXT NOT NULL,
      FOREIGN KEY (post_id)
        REFERENCES posts (post_id)
          ON UPDATE CASCADE
          ON DELETE CASCADE
      FOREIGN KEY (user_id)
        REFERENCES profile (user_id)
          ON UPDATE CASCADE
          ON DELETE CASCADE
```

)

post_impressions

```
CREATE TABLE IF NOT EXISTS post_impressions (
        impression_id TEXT NOT NULL PRIMARY KEY,
        post_id NOT NULL,
        user_id NOT NULL,
        type NOT NULL,
        FOREIGN KEY (post_id)
          REFERENCES posts (post_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
        FOREIGN KEY (user_id)
          REFERENCES profile (user_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
        )
```

comment_impressions

```
CREATE TABLE IF NOT EXISTS comment_impressions (
        impression_id TEXT NOT NULL PRIMARY KEY,
        comment_id NOT NULL,
        user_id NOT NULL,
        type NOT NULL,
        FOREIGN KEY (comment_id)
          REFERENCES comments (comment_id)
            ON UPDATE CASCADE
            ON DELETE CASCADE
        FOREIGN KEY (user_id)
          REFERENCES profile (user_id)
            ON UPDATE CASCADE
```

```
        ON DELETE CASCADE
    )
```

time_slots

```
CREATE TABLE IF NOT EXISTS time_slots (
    date TEXT NOT NULL PRIMARY KEY,
    start FLOAT NOT NULL,
    end FLOAT NOT NULL
    )
```

notifications

```
CREATE TABLE IF NOT EXISTS notifications (
    notification_id TEXT NOT NULL PRIMARY KEY,
    target_id TEXT NOT NULL,
    title TEXT NOT NULL,
    content TEXT,
    time_created FLOAT NOT NULL,
    expire_after FLOAT NOT NULL
    )
```

notifications_sent

```
CREATE TABLE IF NOT EXISTS notifications_sent (
    notification_id TEXT NOT NULL,
    user_id TEXT NOT NULL,
    time_sent FLOAT,
    sent BOOLEAN DEFAULT False NOT NULL,
    PRIMARY KEY (notification_id, user_id)
    FOREIGN KEY (notification_id)
      REFERENCES notifications (notification_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
    FOREIGN KEY (user_id)
      REFERENCES profile (user_id)
```

ON UPDATE CASCADE

ON DELETE CASCADE

)

SQL

SELECT

```
def count(self, data=None):
    info = {'impression_count': 0}

    if dict_key_verify(data, keys: "impression_type") and not self.impression_type:
        self.impression_type = data['impression_type']

    if self.impression_type:
        self.cur.execute(f"SELECT COUNT(*) FROM {self.table_name} WHERE type = ? AND {self.attr_name} = ?", (self.impression_type,self.attr_id))
        rez = self.cur.fetchall()
        if rez:
            info['impression_count'] = rez[0][0]

    else:
        info = None

    return info
```

The above SQL and processing around the statement, takes place in the impressions class. This class is used as a base class for both post_impression and comment_impression classes. Hence the attr_name (post_id or comment_id) varies as well as the table name (post_impressions or comment_impressions) changes.

The Statement itself is simply totalling the number of records that meet the requirements. This is typically used to count the number of likes on a comment or post for instance.

```
def get_content(self):
    info = {'impressions': None}

    if self.attr_name:
        self.cur.execute(f"SELECT impression_id FROM {self.table_name} WHERE user_id=? AND {self.attr_name}=?", (self.id,self.attr_id))
        rez = self.cur.fetchall()
        if rez:
            info['impressions'] = [{column: None for column in self.columns} for impression_id in rez]
            for i, impression_id in enumerate(rez):
                impression_info = self.class_type()
                impression_info.impression_id = impression_id[0]
                impression_info.columns = self.columns
                impression_info = impression_info.get()['impressions'][0]
                info['impressions'][i] = impression_info

    if not self.attr_id:
        info = None

    return info
```

The above SQL command again dynamically uses different tables and column names due to it being a method of a base class. It gets the impression_id of any impression that matches the parameters and then uses this impression id to generate a new object of the same class and perform a get() method to get exact details about the impression.

```
def get(self):
    info = {column: None for column in self.columns}

    if not self.occupation_id:
        self.cur.execute("SELECT occupations.occupation_id, occupations.name, description FROM profile INNER JOIN occupations USING(occupation_id) WHERE user_id = ?", (self.id,))
    else:
        self.cur.execute("SELECT occupation_id, name, description FROM occupations WHERE occupation_id = ?", (self.occupation_id,))
    rez = self.cur.fetchone()

    if rez:
        occupation = {'occupation_id':rez[0], 'name':rez[1], 'description':rez[2]}

        for column in self.columns:
            info[column] = occupation[column]

    if not rez and not self.id:
        info = None

    return info
```

The above method contains 2 SQL statements both executed in different circumstances. This method is intended to fetch details about an occupation, In the event a user has not provided an occupation ID the method assumes the user is referring to their own occupation. So using the occupation_id in their profile and an inner join of the occupations table we get the occupation details.

```
def get_members(self):
    info = {'members': None}

    # option 2
    self.cur.execute("""SELECT auth_credentials.username FROM auth_credentials
                INNER JOIN profile USING(user_id)
                CROSS JOIN teams ON profile.occupation_id = teams.occupation_id
                WHERE teams.team_id=?""", (self.team_id,))

    rez = self.cur.fetchall()
    if rez:
        info['members'] = [{'username': member[0]} for member in rez]

    if not self.team_id:
        info = None
```

The above method is from the teams class. It is to be used to get all the usernames of members in a specific team, when a team ID, occupation ID or user ID is specified by the user. The class properties convert occupation IDs and user IDs into their corresponding team ID. The SQL statement first matches up the username with a user ID using an inner join between profile and auth_credentials. Then using the occupation IDs from the profile table, we perform a cross join with the teams table. Then filtering the results by matching the team ID to the user input in the WHERE clause we get all the members usernames of a specified team. After this statement has been executed a list comprehension is used to format the output information.

```
def create(self, data={'name': None, 'description': None}):
    occupation_uuid = uuid_generate()
    team_uuid = uuid_generate()
    name = data['name']
    description = data['description']

    self.cur.execute("INSERT INTO occupations(occupation_id, name, description) VALUES (?, ?, ?)", (occupation_uuid, name, description))
    self.cur.execute("INSERT INTO teams (team_id, name, occupation_id) VALUES (?, ?, ?)", (team_uuid, name, occupation_uuid))
    self.db.commit()
```

INSERT

The above statements are used for creating a new occupation and its corresponding team. Simply creating a record of each in their corresponding tables after generating 2 UUIDs for each record.

Here 3 SQL statements are executed, the first is used to remove any previous occupation change requests made, since the new one is replacing it. The 2nd is used to check if the occupation ID being targeted exists, then the final statement is used to create a new record, for the occupation_request.

```
def set_request(self, data):
    occupation_id = data['occupation_id']

    self.cur.execute("SELECT approved FROM occupation_requests WHERE user_id = ?", (self.id,))
    if self.cur.fetchone():
        self.delete_request()

    self.cur.execute("SELECT name FROM occupations WHERE occupation_id = ?", (occupation_id,))
    if self.cur.fetchone():
        self.cur.execute("INSERT INTO occupation_requests (user_id, occupation_id, approved) VALUES (?, ?, ?)", (self.id, occupation_id, False))
    else:
        pass
        # add status message here

    self.db.commit()
```

UPDATE

```
def set(self, data):
    occupation_id = data["occupation_id"]

    self.cur.execute("SELECT name FROM occupations WHERE occupation_id = ?", (occupation_id,))
    if self.cur.fetchone():
        self.cur.execute("UPDATE profile SET occupation_id = ? WHERE user_id = ?", (occupation_id, self.id))
    else:
        pass
        # add status message here

    self.db.commit()
```

This function is used for updating information about a certain users occupation. There are 2 SQL statements here. The first is used to verify the occupation being referred to exists. The second is used to updated the users profile to the occupation ID provided.

```
def edit(self, data):
    if 'occupation_id' in data and not self.occupation_id:
        self.occupation_id = data['occupation_id']
    for column in self.columns:
        if column == "occupation_id":
            continue
        value = data[column]
        self.cur.execute(f"UPDATE occupations SET {column} = ? WHERE occupation_id = ?", (value, self.occupation_id))
    self.db.commit()
```

The method above is used for editing certain information about an occupation. The user can decide what is to be edited and does so through an abstracted method of modifying the self.columns list. This list is looped through in for loop and the SQL statement is executed separately per column. This allows us to change certain columns without going through several unreadable if statements. Instead, columns are static and baked into the statement. The columns list is heavily filtered there's only a handful of allowed columns and anything outside of this is excluded from the input. The SQL statement itself is very simple though just update the selected column with the corresponding value passed by the user where the occupation_id matches the user input.

DELETE

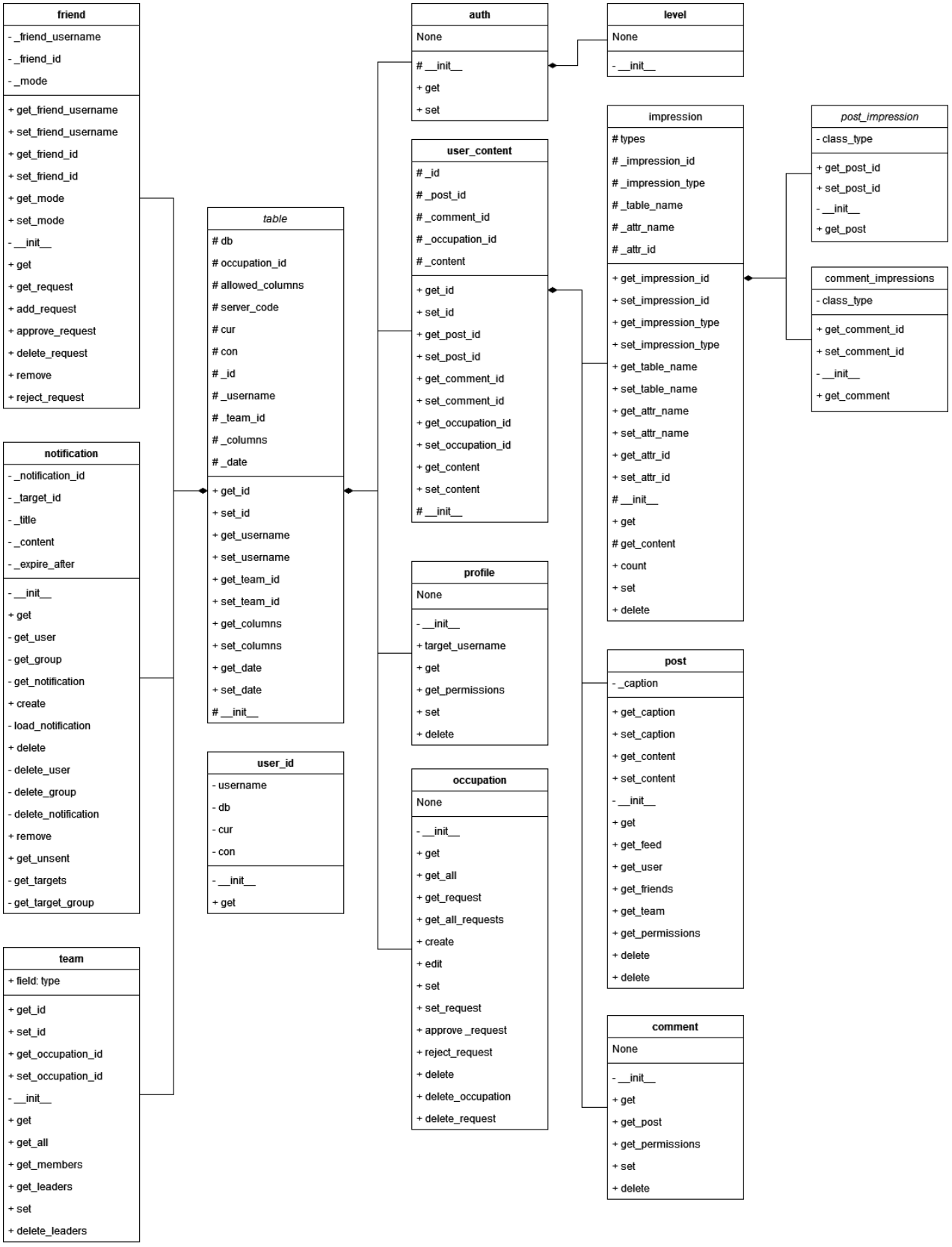
```
def delete(self):
    if self.id and self.date:
        self.cur.execute("SELECT post_id FROM posts WHERE user_id=? AND date=?", (self.id, self.date))
        rez = self.cur.fetchone()
        if rez and not self.post_id:
            self.post_id = rez[0]

    if self.post_id:
        self.cur.execute("DELETE FROM posts WHERE post_id=?", (self.post_id,))
        self.db.commit()
```

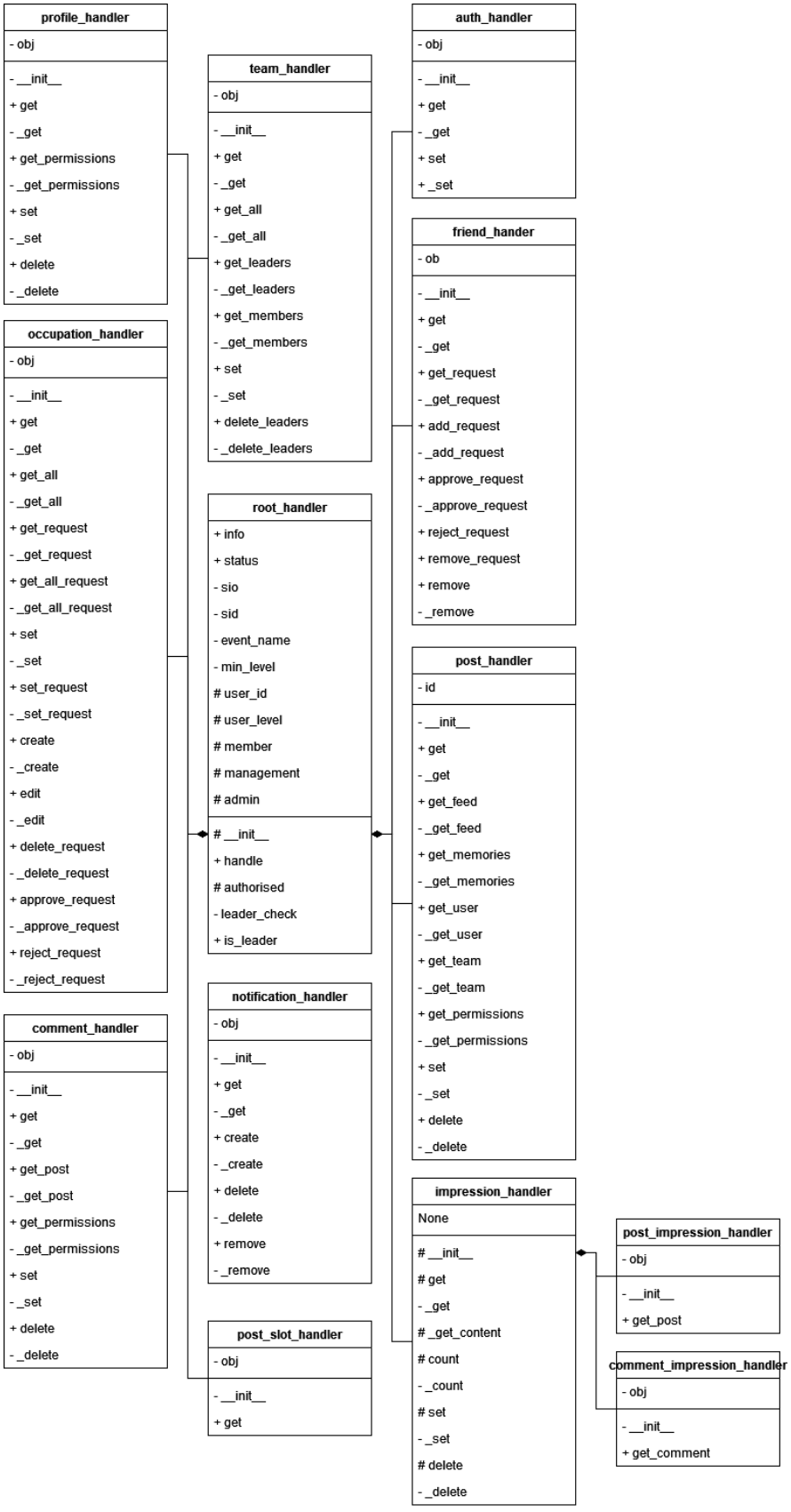
Here 2 scenarios play out depending on the user input. If the user provided a post ID then the input that post ID is respected and the required post is delete. If the user doesn't however the method will use their user ID and the current date to get their post and their post ID and use that to delete the post.

Class structure and diagrams

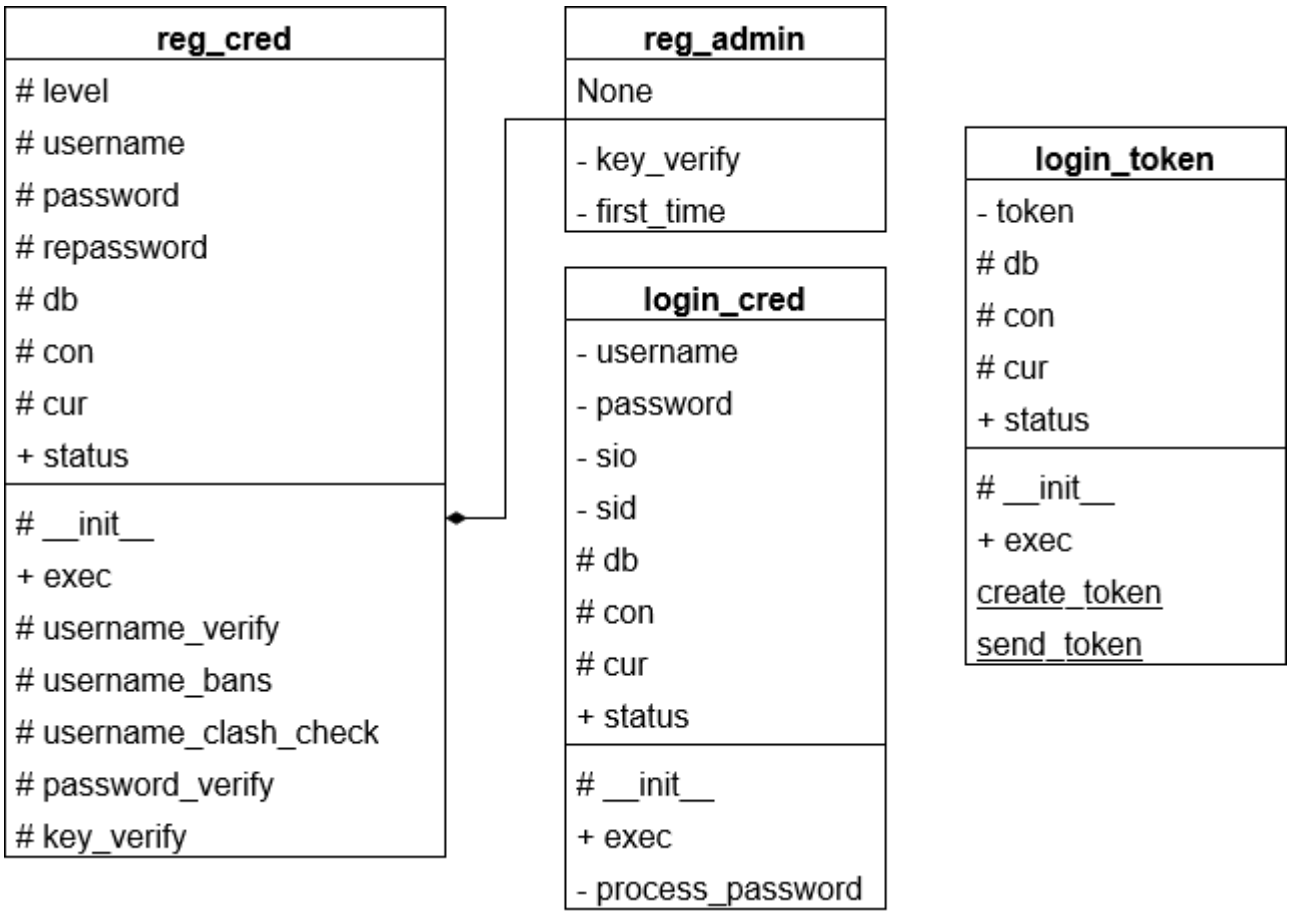
Table classes



Class handlers



Auth classes

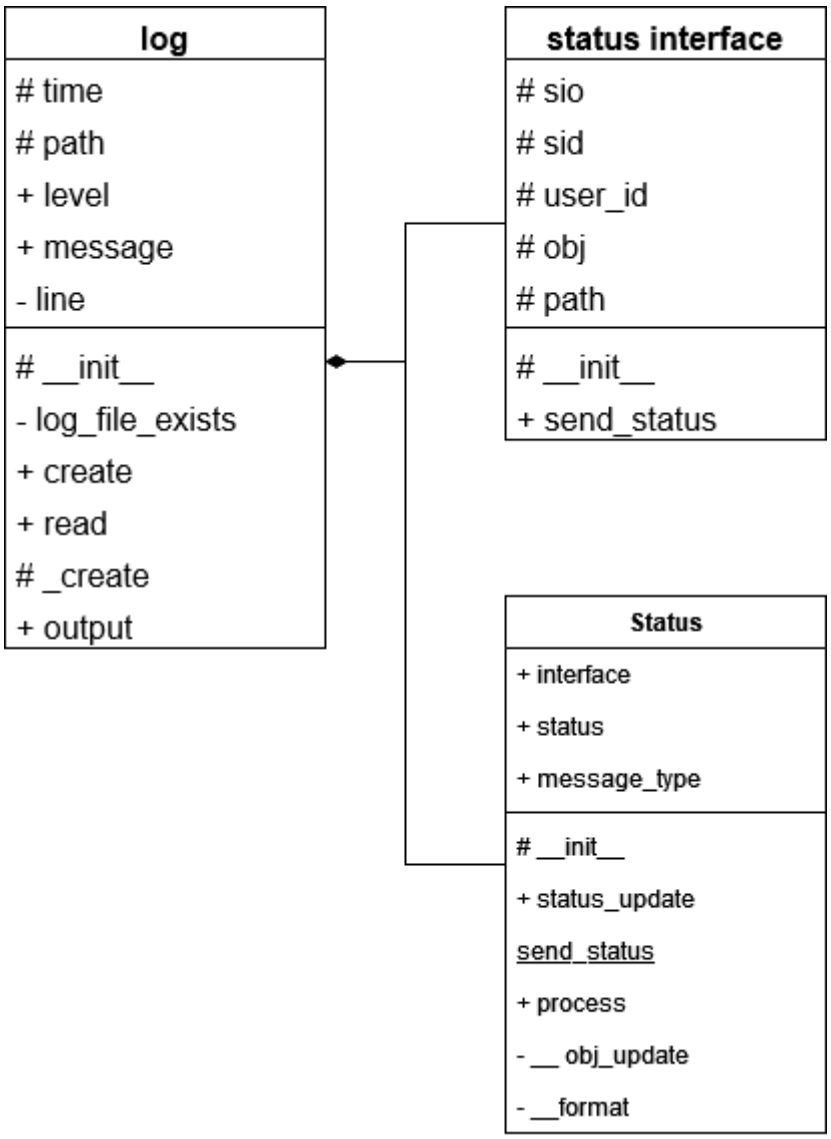


Database Classes

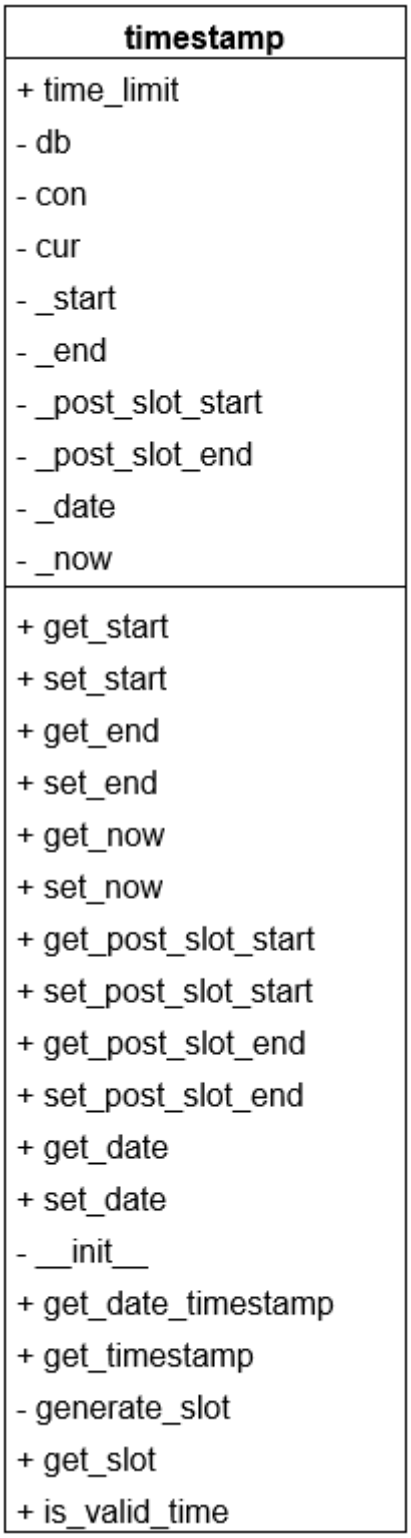
connect
+ con + cur - path
- __init__ + create + commit + close +execute

create
- path - cur - con
- __init__ + tables - auth_credentials - auth_tokens - profile - occupations - occupation_requests - teams - team_leaders - posts - comments - post_impressions - comment_impressions - notifications - notifications_sent

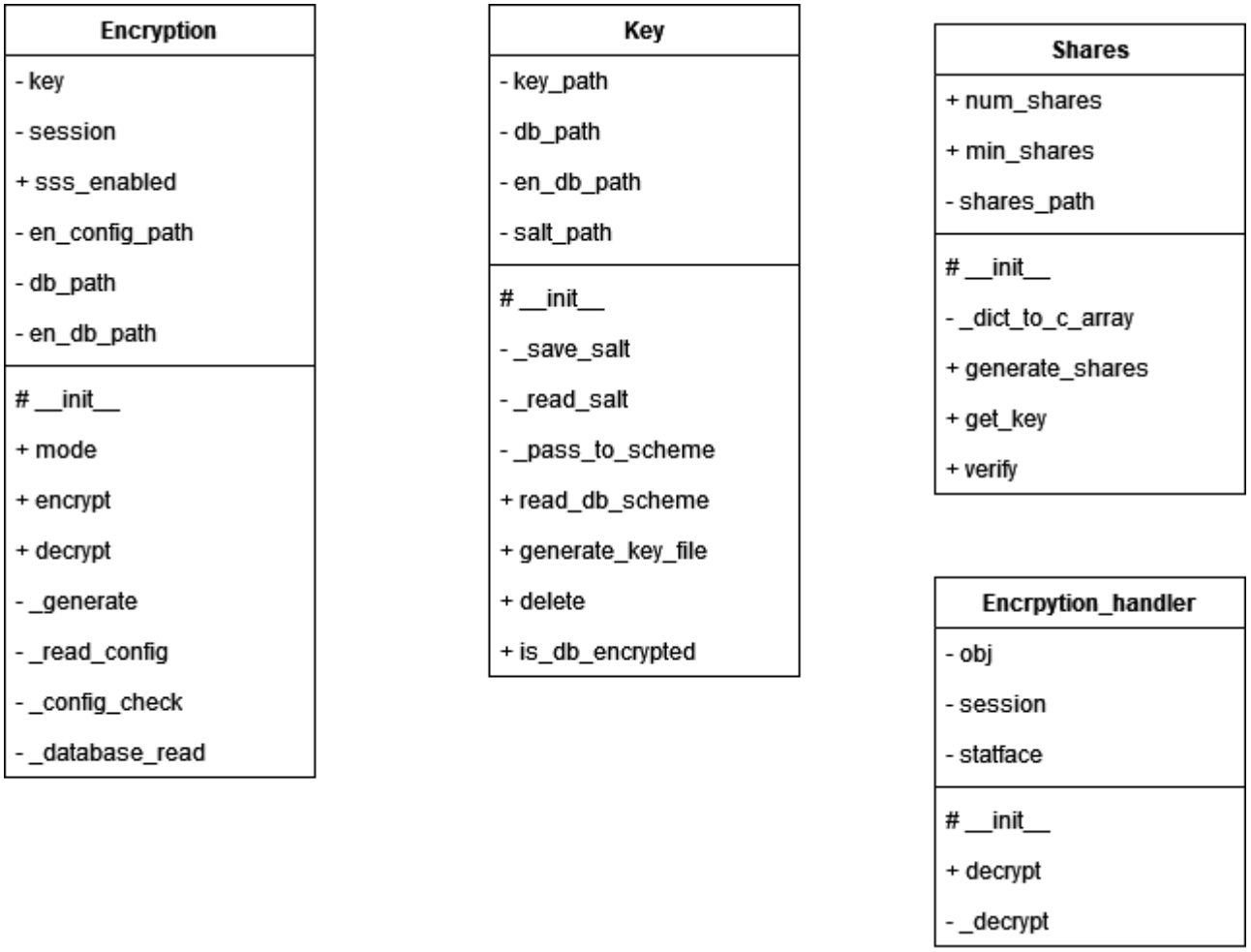
Logging classes



Datetime classes

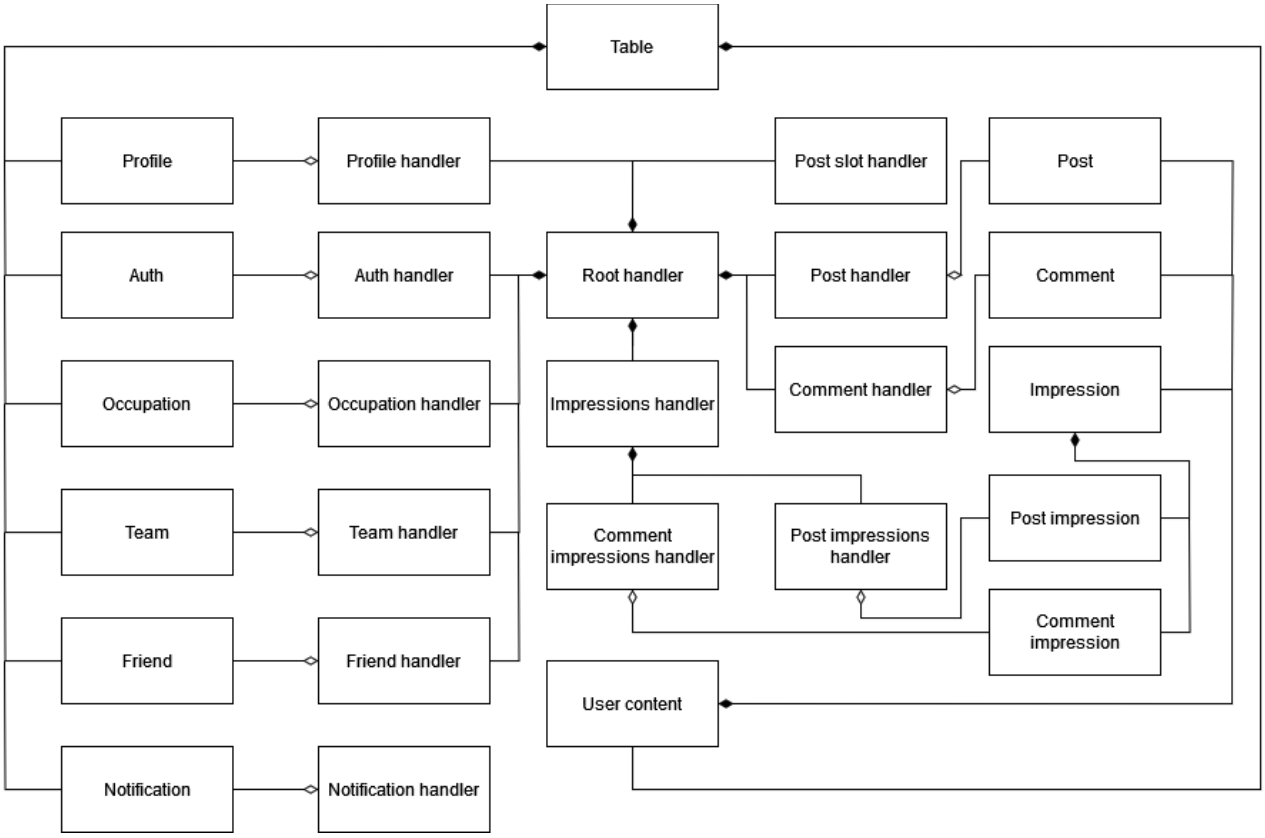


Encryption



Handler and Table classes

This is a diagram that shows the same classes shown in handler and table diagrams, but shows there relationship to each other. That is profile handler has a profile table etc.



Algorithms

Merge sort

A merge sort is used to sort user posts by the number of likes they have, a merge sort is chosen due to its time complexity (*O(n log n)*) when sorting large sets of data, depending on the size of the organisation sorting posts as such could save massively on server-response times when getting posts.

It uses 2 class methods however in the pseudo code they will be represented as 2 functions.

Pseudo code equivalent

```
FUNCTION merge(left, right)
        IF LENGTH(left) = 0 THEN
                RETURN right
ENDIF
        IF LENGTH(right) = 0 THEN
                RETURN left
ENDIF
```

```
        result = []
        index_left <- 0
        index_right <- 0
        WHILE LENGTH(result) < LENGTH(left) + LENGTH(right) THEN
                left_item <- left[index_left][‘impression_count’]
                right_item <- right[index_right][‘impression_count’]
                IF left_item <= right_item THEN
                        result.APPEND(left[index_left])
                        index_left <- index_left + 1
                ELSE THEN
                        result.APPEND(right[index_right])
                        index_right <- index_right + 1
ENDIF
                IF index_right = LENGTH(left) THEN
                        result <- result + left[index_left:]
                        BREAK
ENDIF
                IF index_left == LENGTH(right) THEN
                        result <- result + right[index_right:]
                        BREAK
ENDIF
ENDWHILE

FUNCTION sort (posts)
        FOREACH post IN posts
                num_likes <- post_impressions(post_id=post[‘post_id’]).count()
                post[‘impression_count’] = num_likes
NEXT
        IF LENGTH(post) < 2 THEN
                RETURN posts
ENDIF
```

```
    mid = LENGTH(posts) // 2
    sorted_posts <- merge(left=sort(posts[:mid]), right=sort(posts[mid:]))
    RETURN sorted_posts
```

Generating post list per month

This occurs when generating the memories page on the client. The client will receive several posts all with different dates. The client seeks to generate a number a list of months, then a list of days attached to each month along with their posts.

Pseudo code

```
post_months = {}  #empty dictionary
FOREACH post in posts THEN
    date = post['date']
    date_list = date.SPLIT("-") #splits the date into a 3 item list
    IF date IN post_months.KEYS() THEN
        post_months[date].APPEND(post)
    ELSE THEN
        post_months[date] = [post]
    ENDIF
    NEXT
```

UUID generation

This UUID generation scheme is used for generating all unique IDs. A custom algorithm adds to obscuration of the password salts. Its made up of 2 functions, one for generating the bytes and adding to the hex string. The other for converting binary to hexadecimal digits.

Pseudo code

```
FUNCTION bin_to_hex(byte)
    byte_hex <- ""
    total <- 0
    FOR i=1 to LENGTH(byte)
        total <- total + INT(byte[I]) * 2^I
```

```
        NEXT
        first_place <- total INTDIV 16
        second_place <- total – first_place * 16

        places = [first_place, second_place]
        FOREACH place IN places
            IF place < 10 THEN
                byte_hex <- byte_hex + STRING(place)
            ELSE THEN
                byte_hex = byte_hex + CHAR(65 + place – 10)
            ENDIF
            NEXT
        RETURN byte_hex

FUNCTION den_to_bin(number)
        Byte_string <- “”
        FOR i=7 TO 0
            bit <- number DIV 2^i
            number <- number – bit * 2^i
            byte_string <- byte_string + STRING(bit)
        NEXT
        RETURN byte_string

FUNCTION generate()
        Byte_list = []
        FOR i=1 TO 16
            number <- STR(RANDOMINT(0, 255))
            byte <- den_to_bin(number)
            byte_list.APPEND(byte)
```

```
        byte_list[6] <- byte_list[6][:4] + "0010"
        byte_list[8] <- byte_list[8][:6] + "01"
    NEXT
    hex_string = “”
    FOR i=1 TO LENGTH(byte_list)
        byte_hex <- bin_to_hex(byte_list[i])
        IF i IN [4, 6, 8, 10] THEN
            hex_string <- hex_string + “-“
        ENDIF
        hex_string <- hex_string + byte_hex
    NEXT
    RETURN hex_string
```

Username hash

This function is mainly used in the friend recommendation algorithm (depicted later in the write up). This function simply takes a username in and converts it into a unique number. This hash is not full proof, but the chance of a colliding hash is practically impossible. Especially for deployments on the singular organisation level. The hashes are evenly distributed throughout the range due to the use of the value “p” (described later), essentially using this value means that strings will be distributed across the whole 10^7 + 7 hash space. A custom hash function is used to minimise the size of the resulting hash. If the standard python in built hash was used it would require things like linked lists to be huge. Large lists can soak up lots of memory and force systems to utilise high swap space. This can then bring programs running to a screeching halt. So minimising the size of lists like this is very important for performance.

Originally the hash function was written in python, through some testing I found python could complete a hash of a string (of 25 characters) in ~0.127 seconds. The same program written in C took just ~0.001 seconds, (to 3 significant figures). This means for this huge time saving the hash algorithm was implemented in C and imported as a library into python.

This hash utilises 2 key numbers:

“m” is are large prime number, its size is considered sufficient here since m essentially defines are range of hash results. With an m this size we reduce the chances of a colliding hash. Technically we could make m larger, but m will also define the size of any linked list and hash map used with these hashes. So I decided that 10^7 + 7 was sufficient to optimise memory usage.

“p” is another prime that is as close to the number of string characters that are available for the string input. 97 allows for most ASCII symbols, all numbers and all characters both upper and lower case. In reality though usernames can’t include all ASCII characters I simply included these in the number in case symbol limits were removed in later versions.

Pseudo code

```
FUNCTION hash(String)
    M <- 10^7 + 7
    P <- 97
    Total <- 0
    FOR i=1 TO LENGTH(String)
        Total <- Total + (INT(String[i]) - 32) * p^i
    NEXT
    Result <- Total MOD M
    RETURN Result
```

Friend recommendation (Graph traversal)

The friend recommendation algorithm is made up of 2 classes:

User – Holds information about a user, the number of times they appear and their lowest depth in the graph. It also has methods for finding and organising its friends into other User objects.

Graph – This is where the graph is held, constructed, and traversed. It has methods for generating and traversing the graph as well as adding edges and is geared specifically towards being a friend graph.

There is also a function which is used as an interface for other parts of the system to interact with the Graph. This is not included in the below code since it is fairly un-interesting, just verifying certain parameters.

The code below also does not include the User class since again it is un-interesting and just made up of the components for getting friends from the database and converting them into objects which are added to a list attribute. The only thing to note about the User class is when “hash” is called on one of the objects it utilises a special method that uses the previously described hash function on the object’s “username” attribute.

What is included below is the Graph class, so this is made up of the interesting parts of the algorithm. Below I describe some of the attributes and data structures used in the algorithm:

Important attributes

graph – A 2D linked list that contains all nodes and there corresponding edges. Each node is stored as its position as depicted by its hash. At a nodes position there is another array that contains the hashes of each of its edges. These hashes can then be used to find their respective nodes in the graph and so on. The graph itself is a directed, unweighted graph. Technically it could have been represented as an undirected graph since it's a friend system, not a follow system. But it's converted into a directed graph because when a node is spawned from a previous node that previous node is removed from the adjacency list of the new nodes.

friend_directory - A hash map used to store User objects. The key for each object is the hash of the username. So, using a hash from the **graph** you can find the corresponding object in the **friend_directory**. This is used to manipulate attributes like depth while traversing the graph.

edge_queue – As said by its name this is a queue of the edges to be visited. Since it's a queue this means items can only be added on to the bottom and only removed from the top. So, when a node is visited its corresponding edges (retrieved from **graph**) are added onto the bottom of the **edge_queue**. The node itself is then removed from the top of the **edge_queue** and added to **visted**.

visited – An array of hashes, this is simply used to store the nodes that have already been visited by the algorithm.

Breath first search vs depth first search

Breadth first search (BFS) starts at an origin point in a graph and then visits each of its edges, as it visits each edge it adds that nodes edges to the bottom of the queue. This means it will then search all the edges of these visited nodes.

Depth first search (DFS) starts at the origin point in a graph and chooses its first edge to visit, this edge then chooses its first edge to visit and so on. Essentially the algorithm will traverse to the bottom of the graph and then backtrack to previously unvisited nodes. It does this by utilising a stack instead of a queue. So, when a node is visited its edges are added to the top of the stack, since you can only remove items from the top of a stack the next edge to be visited will be one of the edges that had just been added by the visited node.

I originally used DFS but on encountering the difficulty of calculating the current depth of the node being visited I switched to a BFS. Since using BFS I could assign the depth of an edge as it gets added to the bottom of a queue. I would simply take the origins depth minus one and assign this to the corresponding edge's depth attribute. Additionally using BFS allows for greater optimisations if needed later, for instance it's easier to cut of the graph once a certain depth has been reached. This could be done say if 5 friend recommendations have already been generated.

Converting the DFS to the BFS was incredibly easy though all I had to do was add the depth assignment as edges were added to the queue and swap out the stack data structure for a queue.

Pseudo code

```
CLASS Graph

FUNCTION __init__(self, username)
        self.origin_user <- User(username, True)
        self.graph <- [[]] * 10^7+7
        self.friend_directory <- [NULL] * 10^7+7
        self.friend_directory[hash(self.origin_user)] <- self.origin_user
        self.exclude <- []


FUNCTION PUBLIC generate(self, depth):
        self.origin_user.depth <- depth – 1
        self.add_user_friends(self, origin, source, depth)


FUNCTION add_user_friends(self, origin, source, depth)
origin.find_friends(self.exlude.APPEND(source.username))
                IF hash(self.origin_user) = hash(origin) THEN
                        self.exlude = self.exclude + origin.exlude
                ENDIF
                FOREACH friend IN origin.friend_list
                friend_hash = hash(friend)
                self.add_edge(hash(origin), friend_hash)
```

```
            friend_dir = self.friend_directory[friend_hash]
                IF friend_dir != NULL THEN
                    self.friend_directory[friend_hash].count <- friend_dir.count + 1
                ELSE THEN
                Self.friend_directory[friend_hash] <- friend
                ENDIF
                IF depth-1 > 0 THEN
                self.add_user_friends(friend, origin, depth-1)
                ENDIF
            NEXT

    FUNCTION add_edge(self, node, edge)
    self.graph[node] = self.graph[node].APPEND(edge)

FUNCTION PUBLIC bft(self)
    self.visited <- []
    self.edge_queue = [hash(self.origin_user)]
    self.visit(self.edge_queue[0])

FUNCTION visit(self, origin)
    start_pos <- self.graph[origin]
    self.on_visit(origin)
    self.edge_queue.REMOVE(LENGTH(self.edge_queue) - 1)
    self.visited.APPEND(origin)

    FOREACH neigbour IN start_pos
        neigbour_obj <- self.friend_directory[neigbour]
        origin_obj <- self.friend_directory[origin]
        in_visited <- neigbour IN self.visited
```

```
            in_queue <- neigbour IN self.edge_queue
            IF NOT in_visited AND NOT in_queue THEN
            neigbour_obj.depth <- origin_obj.depth - 1
            self.edge_queue.PREPEND(neigbour)
            ENDIF
        NEXT
            IF LENGTH(self.edge_queue) > 0 THEN
            next <- self.edge_queue[LENGTH(self.edge_queue)-1]
                self.visit(next)
            ENDIF

        FUNCTION on_visit(self, origin):
            origin_obj <- self.friend_directory[origin]
            origin_obj.score = origin_obj.depth * origin_obj.count
```

Shamir Secret Sharing

Shamir Secret Sharing is a method for sharing a secret in such a way that if any combination of those shares were put together you can derive the original secret. So, when the shares are first created you provide the program with 3 parameters:

- The secret – This must be turned into a number somehow, (for instance you can use an ASCII table to convert a string into a number)
- Minimum number of shares – This is the minimum number of shares needed to be combined for reconstruction of the secret.
- Number of shares to be generated – The number of shares to be generated this number must be bigger than the minimum number of shares needed for reconstruction.

This scheme in our system is used for distribution of the secret database encryption key. Shamir Secret Sharing has been used by several notable organizations for similar purposes for instance PayPal used Shamir Secret Sharing to secure their databases in the early 2000s.

Mathematical principles

This scheme works on a simple mathematical principle: A polynomial of power n can be found if provided with n+1 points that lie on that polynomial. For instance 2 points on a cartesian (2D coordinate system) set of axes perfectly define a line, there is no other

straight line that will go through the points (1, 3) and (-8, 8), there exists just one line. A straight line can be represented with the polynomial: y = ax + b

You can keep going with this given 3 points on a polynomial with a power of 2 you can find the values of a, b and c in which construct this polynomial:

y = ax^2 + bx + c

So, if we encoded our secret (which remember has to be converted into a number) as the y intercept of a polynomial we could use points on this polynomial as our “shares”. Since say we encoded our secret in a polynomial of power 3 this means that 4 points (aka shares) could be used to reconstruct that polynomial. Let’s say we generated 10 shares any 4 of these 10 shares can be used to perfectly reconstruct the polynomial f(x) and then the secret is equal to f(0).

Generating shares

To generate the shares as said before we need 3 inputs, the secret (we will call s) the minimum number of shares for reconstruction (we will call m) and the number of shares to be generated (we will call p). There is no theoretical limit to the size of m as long as m > 1 in the perfect system. However due to the limits of floating-point integers etc the maximum size of m in my system is 7. The size of p however only has the limit of p > m, since if it was not, we would never be able to reconstruct the secret. So to generate shares all we do is create a polynomial of size m-1, so if our m was 3, we generate:

y = ax^2 + bx + c

Since our secret is encoded in the y intercept c will equal our secret for instance lets say our secret is 520. So now c = 520. a and b have no limitations as to what they can be, however this system picks a and b randomly between the limits of 2^(n-1) + 1 and 2^(n) – 1. The system sets n to 50 by default but this can be altered in future versions as 50 is entirely arbitrary. We also generate a random prime number, we then perform prime MOD random number on each random number generated. This limits the size of the polynomial’s a and b.

So now we have values for a and b, this means we have a complete polynomial we will call f(x) and can start generating shares by using different x values. Our system simply counts linearly (increasing by 1 each time) starting from x = 1 all the way to x = p. The x values are considered public so using these simplistic x values is no issue.

Once the shares are generated the system outputs them to a set of text files (1 per share) each share text file contains 3 pieces of information. The “number share” (aka the x value of the point), the “share secret” (the y value of the point) and the minimum shares needed to reconstruct the secret. The “number share” and the minimum shares required are considered public however the share secret is the important part that should be kept secret

and safe. So someone keeping a share simply needs to keep 2 pieces of information: what number share they have and what their share secret is.

The math I have walked through is the actual math that the system uses. This part of the system isn't particularly interesting as it simply consists of some simple math operations and substitution of x values into a polynomial. So will not provide any pseudo code.

Reconstructing the secret

This is the interesting part. I will provide a worded step by step here as well as pseudo code for large segments of this algorithm.

To reconstruct the secret the system needs the number of the minimum number of shares required to reconstruct the secret (we will call this m) and shares (consisting of their secret and share number) the number of shares provided should be equal to m.

The first thing we do is construct a set of linear equations, essentially using m we can determine the power of our resulting polynomial (m – 1) this means we know how many unknown coefficients we have. For instance if m = 3 we know our polynomial is of power 2 and so has 3 unknown coefficients. y = ax^2 + bx + c where a, b and c are or unknowns. Since we have an x and y we can substitute these values in for all our points. Continuing with our m = 3 example lets say our 3 points are (1, 1872), (2, 4266) and (3, 7837). This means we can construct the following 3 linear simultaneous equations:

1872 = a + b + c

4266 = 4a + 2b + c

7837 = 9a + 3b + c

So, we have 3 simultaneous equations and 3 unknowns (note that c is our secret). Any GCSE level child could do this (especially this simple example) easy with a bit of time. But for a computer to reliably find a, b and c with even the most complex values we have to use matrices.

You can use matrices to solve systems of linear equations of any size and complexity, this makes them perfect for this application as the program needs a strict set of calculations it can perform to derive the answer consistently. So we can turn these linear equations into the following matrix system:

$$\begin{pmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1872 \\ 4266 \\ 7837 \end{pmatrix}$$

Essentially you take your integer coefficients put them into their own matrix and multiply this by a matrix of unknows setting this equal to the matrix of known y values (our share secrets). In this case are m = 3, so our first matrix is of size 3x3 the second of size 3x1 and

our resulting product of these two matrices is a 3x1. But in general terms (depending on the size of m you have a m x m matrix multiplied by an m x 1 which equals an m x 1 matrix. This is why matrices are ideal for this application they can scale no matter the size of the linear equations.

To find a, b and c we simply must multiply our right-hand matrix (the matrix of y values) by the inverse of our matrix of x results (the m x m) matrix. I will not explain here how the inverse of a matrix is found but I do show how this is done in the pseudo code below. Once you find the inverse and multiply it you will get 2 matrices of the same size are equal to each other. This is the procedure the system goes through to find these unknows. From there it reconstructs the polynomial equation called f(x) and then finds the result of f(0). This result is the secret.

The language choice

I chose to use C++ to tackle this algorithm, there are a few reasons it was chosen over python or any other language. The rest of the system is in python (other than the hash function) so whatever language I used had to be easily interfaceable with python. Since python is built on C the native library “ctypes” can be used to interact with C and C++ code. The question now is why C++ over python, simply this was down to speed and control. This was an intensive math heavy algorithm with numbers that varied widely in size. If done in python the program would run particularly slow when it started to deal with large numbers since it must dynamically allocate the memory for the numbers (since its an interpreted language), meanwhile C++ is a compiled language meaning all the memory required for the numbers is already assigned and overall, the program can run much faster. Additionally, the code has to do lots of division and large recursive computations so to not bog down the server I decide a C based; compiled language was best. C++ was chosen over C though despite not classes being used in this algorithm due to C++ having greater quality of life features which objective C still lacks.

Pseudo code

Here I will go over some of the key functions that are used in the processes outlined above, mainly focusing on how we manipulate the matrices once formed while reconstructing the secret. The below functions mainly focus on the process for inverting a matrix I will quickly outline this process to get the inverse of a matrix you:

1) Find the determinant of the matrix
2) Form a matrix of minors
3) Form the matrix of cofactors from the matrix of minors
4) Then multiply this matrix of cofactors by the reciprocal of the determinant
5) This gives you the inverse matrix

The below function is used to find the determinant of a matrix, it's a recursive function since these matrices can be as large as dimension 7. The final ELSE statement uses the calculations for getting the determinant of a matrix with a dimension above 2. If not done recursively this function would be huge and would have to contain lots of confusing math operations for every case of every dimension. This would harm the codes expandability and maintainability. Additionally in maths matrix determinants are found recursively even when doing it by hand. This makes the code easy to understand for anyone who understands the basics of matrices.

```
FUNCTION findDet(matrix, dimension):
    IF dimension == 0 THEN
        det <- 1
    ELSE IF dimension == 1 THEN
        det <- matrix[0][0]
    ELSE IF dimension == 2 THEN
        det <- matrix[0][0] * matrix[1][1] – matrix[0][1] * matrix[1][0]
    ELSE THEN
        FOR 0 TO dimension-1
            sub_matrix = findMinor(matrix, dimension, i, 0)
            sub_matrix_det = findDet(sub_matrix, dimension-1)
            term <- matrix[0][i] * sub_matrix_det
            IF (i+1) MOD 2 ==  0 THEN
                term <- 0 – term
            ENDIF
            det <- det + term
        NEXT
    ENDIF
    RETURN det
```

The function below is used for finding the minor of a matrix, this is also used when finding the determinant of any matrix larger than dimension 2. This function simply works by taking in the row and column that is not to be included in the matrix of minors. So by simply looping through the matrix and comparing x and y positions with the row and column that is not to be included in the minor we easily generate the minor of a matrix.

```
FUNCTION findMinor(matrix, dimension, pos_x, pos_y)
    # creates an array of size (dimension – 1)
    minor <- [] * (dimension – 1)
    minor_x <- 0
    minor_y <- 0

    FOR i FROM 0 TO (dimension – 1)
        # creates an array of size (dimension – 1)
        Line <- [] * (dimension – 1)

        FOR j FROM 0 TO (dimension – 1)
            IF i != pos_y and j != pos_x THEN
                Line[minor_x] <- matrix[i][j]
                minor_y <- minor_y + 1
            ENDIF
        NEXT
        IF minor_x != 0 THEN
            minor[minor_y] = line
            minor_y <- minor_y + 1
        ENDIF
        minor_x <- 0
    NEXT
RETURN minor
```

The function below takes in the original matrix as an input and coverts it into the matrix of cofactors. It uses 2 previous functions to find the minor and the determinant of said minor in any given section of the given matrix. Using this it constructs a new matrix (a 2D array). Note that since this is written in C++ originally, the 2D array is in actual fact an array of pointers. Each pointer leading to a 1D array. This is the same for all matricies throughout the code and is talked about further in the data structures section of this write up.

```
FUNCTION formMatrixCofactors(matrix, dimension)
```

```
    # creates an array of size dimension
    cofactors <- []*dimension
    FOR i FROM 0 TO dimension-1
        # creates an array of size dimension
        line <- [] dimension
        sign <- 1
        IF (i+1) MOD 2 == 0 THEN
            sign <- -1
        ENDIF
        FOR j FROM 0 TO dimension
            minor <- findMinor(matrix, dimension, j, i)
            cofactor <- findDet(minor, dimension-1) * sign
            sign <- - sign
            lint[j] <- cofactor
        NEXT
        cofactors[i] <- line
    NEXT
    RETURN cofactors
```

This next function “transposes” the matrix this is the process of turning each row into a column.

```
FUNCTION transposeMatrix(matrix, dimension)
    # creates an array of size dimension
    transposed_matrix  <- []*dimension
    FOR i FROM 0 TO dimension-1
        # creates an array of size dimension
        line  <- []*dimension
        FOR j FROM 0 TO dimension-1
            line[j] <- matrix[j][i]
        NEXT
```

```
            transposed_matrix[i] <- line
        NEXT

        RETURN transposed_matrix
```

The next function is used in the last stage of the process taking in the determinant and the transposed matrix. It then multiplies this matrix by the reciprocal of the determinant.

```
FUNCTION formInverse(matrix, dimension, det)
        # forms an array of size dimesion
        inverse <- [] * dimension
        FOR I FROM 0 TO dimension-1
            # forms an array of size dimesion
            line <- [] * dimension
            FOR j FROM 0 TO dimension
                Line[j] <- (1.0/det) * matrix[i][j]
            NEXT
            inverse[i] <- line
        NEXT
        RETURN inverse
```

So at this point in the program, we have our inverse now all we do is multiply this inverse by the matrix of y values (the matrix of the share secret values). The inverse is passed as matrixA and the matrix of y values as matrixB since they are multiplied in the order inverse x matrix of y values. Another thing to note is: in the C++ program matrices are passed as structs containing their x and y dimensions along with their actual matrix. Here though I have just represented the x and y dimension of matrixA as arguments.

```
FUNCTION multiplyMatrices(matrixA, matrixB, matrixA_x, matrixA_y)
        # forms an array of size matrixA_y
        result_matrix <- [] * matrixA_y
        FOR i FROM 0 TO matrixA_y
            # creates an array of length 1
```

```
            line <- []
            result <- 0
            FOR j FROM 0 TO matrixA_x
                result <- result + matrixA[i][j] * matrixB[j][0]
            NEXT
            Line[0] <- result
            result_matrix[i] <-line
        NEXT
        RETURN result_matrix
```

Control flow

These functions are all used by a function called solve which simply gets the result of one function and passes it to the next along with extra data like the dimension of the matrix etc. This function is very basic and boring all it really does is call this set of functions above. As for getting the final secret this solve function simply returns the last element of the resulting matrix as this is equivalent to the y intercept. I will go into detail about the structures and data types used throughout the C++ program as here the C style arrays, pointers and structures are turned into a pseudo code style “list” to keep the algorithm easy to understand.

Limitations

As said before there are limitations on the number of shares that can be generated (must be less than 20) and the number of shares required for reconstruction (must be less than 7). The reason there is a maximum on the number of shares needed for reconstruction is because as you increase this number, you increase the size of the matrix and the size of the determinant. Computers are bad at division and even with C++ largest floating point number after 7 shares for reconstruction the division becomes to minute and the program starts to lose accuracy. This loss in accuracy is only by a few decimal points but reconstruction of a encryption key needs extreme accuracy.

If this module was to be re-written and it needed to support reconstruction with greater than 7 shares it would need to use a 3rd party library (not included in the standard C++ distribution) to support more accurate numbers with larger bits. The downside to doing something like this is when the number of bits a number uses becomes larger than the page size of a CPU programs can start to slow down significantly. So doing this would likely make the algorithm slower. I consider 6 shares for reconstruction to be plenty for most use cases, any more than this is an edge case, and the organization can likely spare

the resources to re-code this one module. Additionally, it’s a very minor change if you wanted to do this all you have to do is install the header file and change the type definition of “Lint” (currently long long int) and “Ldouble” (currently long double).

Post scheduling and time slots

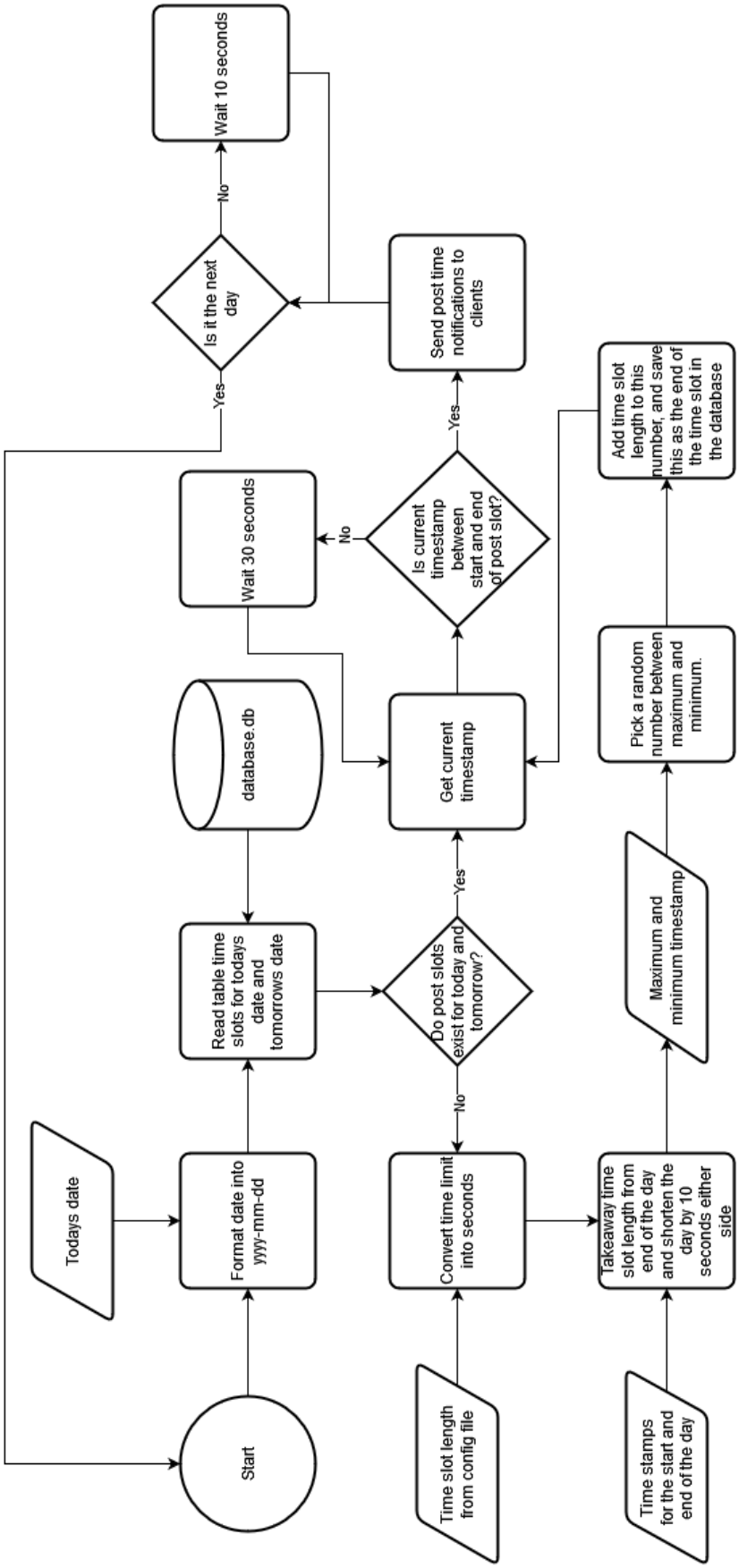
A key part of this system is the scheduling of when users can create posts. Users are only supposed to be able to post within a certain time frame within the day. This is enforced server side, not client side to prevent the creation of a malicious 3rd party client, however clients can (and are encouraged to) request the start and end times of the post slot for the current day. This is done so that clients can schedule a notification on the user’s device. However, when it is time to post an active client will receive an internal notification from the server.

On the server the creation and management of posts slots is given to a background process that runs separate from the main thread. The unit used throughout the process is seconds since epoch (Unix time). This is used to avoid constant conversion of time and for exact and globally agreed time across all systems. Its also the easiest to perform math operations on.

Time slots are stored in the database in the “time_slots” table, this is the only table with no foreign keys in the entire database. It contains 3 fields, the date in the yyyy-mm-dd format (this is the primary key) and is stored as a string. Time slot start is stored as a float number in Unix time, and time slot end stored the same way as time slot start. Another approach to this would have been storing the start of the time slot and the length of the time slot but that would require extra compute whenever a process needs the time of the slot end.

Flowchart

The flowchart does not have a stop point since this is an overview of the background process that runs from server launch until the server itself shuts down. The flowchart doesn’t include the other functions this background service performs like cleaning expired notifications, since this is not relevant to the post scheduling.


Start
Todays date
Format date into yyyy-mm-dd
Read table time slots for todays date and tomorrows date
database.db
Do post slots exist for today and tomorrow?
Yes
No
Time slot length from config file
Convert time limit into seconds
Get current timestamp
Is current timestamp between start and end of post slot?
Yes
No
Wait 30 seconds
Send post time notifications to clients
Is it the next day
Yes
No
Wait 10 seconds
Time stamps for the start and end of the day
Takeaway time slot length from end of the day and shorten the day by 10 seconds either side
Maximum and minimum timestamp
Pick a random number between maximum and minimum.
Add time slot length to this number, and save this as the end of the time slot in the database

Data structures

Recommendation graph

A graph is used in the friend recommendation algorithm. This graph is used to define the relationships between users (friends, friends-of-friends, etc) and traversed from an origin node to generate friend recommendations. The graph is generated on-the-fly, this is done instead of pre-generating the graph on boot to save on memory usage and enforce certain user exclusions. Users such as those already with friend requests from the origin user are excluded from the graph. The graph itself is a directed but unweighted graph, while it technically could be an undirected graph it is “converted” into a directed graph by removing the previous nodes from the adjacency list of the node it spawns. This is done to make traversal easier and prevent double counting of a node.

The graph is made up of “nodes” but essentially boils down to a large 2D array. A nodes hash (defined by a hash of the user’s username) is also its position in the array. Then at its position a second list containing the hash of all “edges” (friends of that user) is stored. This means you visit a node via its hash and from that position you can pull a list of other user hashes.

Some nodes have no users in their list this is because when generating the graph a certain depth is defined. A depth of 1 only allows for the users friends to be visited, depth of 2 allows for friends of friends to be visited and so on. This depth parameter is stated to limit the compute cost of the algorithm.

A graph was chosen for this task since recommending a friend was done based on a simple calculation:

Number of times a user appears in other friend lists x The “distance” from the origin user

So, the graph is used for calculating the distance from the origin user and calculating the score as each node is visited. The count for the number of times a single node appears is assigned to the node’s user object as the graph is generated.

Overall, a graph makes the algorithm way more efficient as there’s no need to hold several arrays of different users friends in memory instead it can all be handled in a single array or the “graph”. It also allows for functionality to be added later, since as each node is visited utilizing the friend directory and their hash you can make any alterations or calculations about that user’s relationship with other users that’s needed. A slightly modified version of the algorithm could be used to recommend posts to other users and sort a user’s feed by relevance.

This graph is found in the “Graph” class in “modules/algorithms/recommend.py

Recommendation queue

A queue is used in the friend recommendation algorithm. It's used to organise which nodes in the graph should be visited next. Nodes are placed at the tail of the queue (index 0) and the next node to be visited is taken from the head of the queue.

A queue was chosen for this task because I wanted to traverse the graph breadth first, a queue allowed me to add neighbours of the currently visited node to the end of the queue meaning once all of the ahead nodes (a layer above the neighbours) had been traversed these neighbours would be visited next and so on. A stack could have been an alternative data structure used but this would of forced the use of a depth first search which was slower and caused greater complexity when working out the distance of each node from the origin node (a central part of the friend recommendation).

The queue itself simply holds hashes which can be used as references to the hash table where the user objects are stored and used as positions in the graph. This means that the hash of each object can be used to point towards 2 different sets of values, in other data structures.

This queue is found in the "Graph" class in "modules/algorithms/recommend.py

Recommendation hash map

A hash map is used again in the friend recommendation algorithm in the Graph class. The hash map is used to allow the algorithm to look up a user's object via the user's hash (which is used in the graph, visited and edge queue). So, the key is the hash of the user, and the stored value is an object of the User class which contains information about the user like their username as well as methods for generating a list of their friends.

A hash map is used so that we don't have to convert the hash to a string for lookup in a python dictionary and so we can pull the raw integer of the key. It allows for more efficient use of compute however comes at a slight cost of memory usage. The memory usage cost though is minimal since most of the list is empty it's just that the list is the size of our largest hash: 10^7 + 7. But overall, a few kb of memory is worth it for the more readable and compute efficient code.

It also allows for other languages like C to integrate into the algorithm to manage some intensive parts. If it was left in a python dictionary C would be almost completely unable to interact, but as an array C can easily index as it normally would with C style vectors.

Notification queue

A user background service utilises the notifications table object to fetch unsent notifications. Notifications are only counted as "sent" once the server has sent them via the clients "notification event". Each notification is timestamped, and I decided that the older notifications should get priority in a queue.

So before hitting the user's personal notification service the "get_unsent" method queues the notifications with the oldest being at the front of the queue. Then once passed to the service it takes the notification of the top of the queue gathers some additional data before sending it off to the user. Then it loops back around onto the next notification.

A queue was chosen so that in real time new notifications created since the notification service started could be added to the back of the queue. Say while the notification service was sending lots of notifications to a user at once another was created, this notification can be dynamically added to the back of the shared notification queue to be sent while the service is running.

Images

Pictures and images are integral to the system, the whole idea of the platform stems around taking these pictures and providing them as posts to other users. As a result, the way in which images are received, stored, and sent is very important.

Images can be provided by the client in 2 formats, either png or jpg. I chose these formats since they are the most common formats for bitmap images additionally the libraries used to interact with the camera for both desktop and android defaulted to outputting png. If the clients provide any other type of image the post creation process will fail and provide back a status message saying as such.

Images are not compressed server side or client side; this could be a problem say if a client provides a particularly large image. However, the size of data is limited by socketIO which will limit the size of a single data transfer. This limit is high enough to not get in the way of any normal image, but an image file specifically designed to be maliciously large would be limited.

On the server side all images are stored in the "data/images/" path. Each image's name is stored simply as {post ID}.{format}. So, an example of this would be "2cd80607-5f29-490f-b666-81b94b6f8378.png". Using the post ID to identify images works perfectly since the post ID is unique per post and since there can only be one image per post it works for our current implementation. The path to the image isn't just derived from the post ID though since we could decide in the future to adopt a new naming scheme for images. So, for future proofing and simplicity of code the "posts" table in the database has a string field called "content", this is where the path for the image is stored.

Like I said before we could technically find a post by just using the post ID but a likely change to the system would be dividing the "data/images/" directory per user. So, the images directory would have a directory per user. So, in future versions the path to a user's post on the 5th of April might look like: "data/images/johnathon/05-04-24.png". Currently this sort of change to the pathing logic is unnecessary, modern filesystems can handle thousands of files per directory and the additional logical overhead of implementation is not worth the theoretical performance benefit on older systems.

However, if this change was to be made it could be quickly implemented, due to the dedicated field in the posts database.

Database

The database on the server side uses several techniques to keep data linked and organised according to the 3rd normal form standards. Since at large this is a database heavy application the database must be well maintained and well linked to keep the data inside it consistent.

For instance, the occupationID of an occupation is the primary key in the occupations table, it is also a field and foreign key in both the Teams table and Profile table. These foreign keys then have different properties. Following this same example for the teams and profile table, on the occupationID being updated in the occupations table the updates cascade. However, on deletion of occupationID (and its occupation since it's the primary key) the teams table will delete any entries with that occupationID (aka, cascade delete). The profile table however will simply turn that field of the record to NULL. Since we don't want to delete the user but just de-associate them with the now deleted occupation.

These kinds of structures are all over the database to keep the data as consistent as possible. The server already does enough work so the database should do its job and not leave any "cleanup" tasks to the scripting. Every instance of the same type of values uses a foreign key pair to maintain this methodology.

Matrices

In the Shamir secret sharing algorithm matrices are heavily used to solve a system of simultaneous equations. The entire algorithm is written using C style arrays and pointers, meaning static arrays and memory address pointers had to be used. Especially since many of these matrices were being returned from functions.

Matrices themselves were represented as a pointer to an array of pointers. This was the method for representing and passing a 2D array around the program. Since in C you cannot simply return an array, you must return a pointer to an array created inside a function. To make a 2D array you must create an array of pointers. Matrices also have some other important information that is crucial to their use, namely their dimension. Without explicitly passing the dimensions a function would have no way of knowing the size of a matrix. Since it's simply being passed as a pointer there is no operation that can be performed to get the size of an array.

To tackle this a Matrix structure was used so that the 2D array of the matrix values could be passed along side its x and y dimensions in one neat package. This meant that all you had to do was pass matrixA and the function could get all the information needed about matrixA.

Testing

Here I describe the testing methodology. The server's functions are divided up into classes. Each class has its relevant methods and a client facing handler function. For example, of these classes could be "posts" we will call this the "post module". Then "posts" will have a client facing handler class called "posts_handler". So, the testing methodology follows this same structure, first a module is written with all the relevant data then each method in that class is tested internally by calling it relevant file directly and utilizing a defined "test" function, with some base case values. Once these tests have passed, I write the handler and the client-side code to interact with the server events. This code is then tested together and where we test the edge cases for the original module, since the handler is built to clean inputs and handle erroneous data. So, there are 2 stages to testing:

Stage 1: Individual internal test on module (for instance "posts"), simple correct usage of the methods is tested here (mainly looking for syntax errors and very basic logic errors here)

Stage 2: "end-to-end" testing using the GUI client, which then goes through the event, handler, and module itself. Edge cases and wrong inputs are tested here to make sure the server and client can handle these cases.

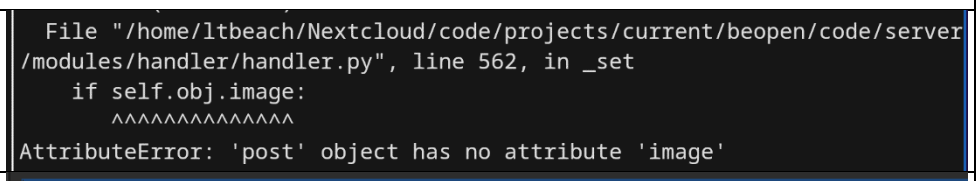
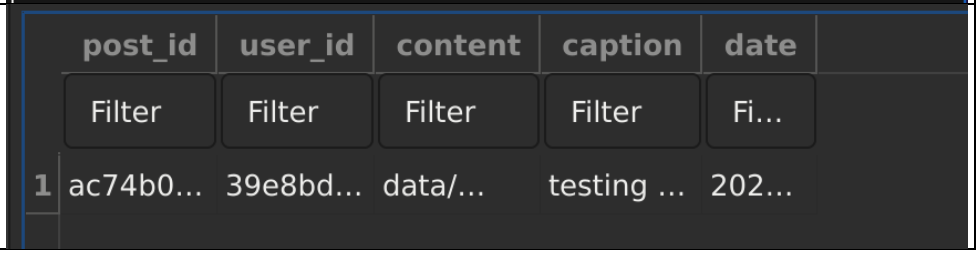
Stage 2 testing is the main thing documented in this write up since the Stage 1 testing is largely uninteresting and the only bugs, I encountered there were small syntax errors or minor logic bugs, nothing that took more than a minute to solve. Stage 2 testing though is far more interesting since it still deals with some of these minor bugs but also larger logical flaws and led to more interesting code dealing with edge cases etc. Stage 2 testing additionally led me to digging around source code from my python GUI library.

There were some exceptions to this stage 1, stage 2 methodology for instance when testing more complicated and staged algorithms as well as any C++ code written was tested more consistently as it was being written (as I am less experienced in this language). But stage 1 and 2 testing always followed this was also just some pre-liminary testing that we can call stage 0 testing.

Server tests

Test Number	Test Description	Expected	Observed	Action
PST.1A.I	Using the "post_set" event from the client including valid content and	The server should accept the post write the relevant information to	The server complains about there being no such	This is because I should be using self.obj.content content is the

	caption. The client is creating 1 post. Its also being done in the valid time slot	the database as well as save the post image.	thing as the attribute image	referance to the image to make the code more flexible for the future if it was altered to be a text post instead for instance.
PST.1A.I	Using the “post_set” event from the client including valid content and caption. The client is creating 1 post. Its also being done in the valid time slot	The server should accept the post write the relevant information to the database as well as save the post image.	As expected	

Test Number	Image Number	Image
PST.1A.I	1	
PST.1A.II	1	

Test Number	Test Description	Expected	Observed	Action
N.1A.I	When the current time is within the allocated time slot defined by the table time_slots entry, the notification service should generate the post time notification.	The notification to be generated and an entry added to the notifications table of the database containing a unique notification_id, a title that uses the post-servercode format, content stating the time until post as configured as well as its expiration and time created.	As expected	
N.2A.I	After the notification has been created in the notifications table load_notification should be called just after creation.	This function should then identify the targets of the notification and add their user_id, the notification_id and a status on whether it has been sent in the sent_notifications table.	The method that identifies the target_group was returning None. This should only happen if the notification id or target id was not valid.	When a notification needs to be sent to the entire server the target_id is set to None. But get_target_group checks if a target_id exists. Since the id is None it believes target id is doesn't exist. Addressing the entire server is now done by a special code

				“all-{server code}” this string is also made an illegal username, so the target_id is not mistaken.
N.2A.II			get_target_group continues to return None. Due to a misuse of the “is” built in function returning False.	The culprit line was remove and replaced with an if/else statement.
N.2A.III			As expected	
N.3A.I	Testing wether the user_notification_service can send a queued notification to a logged in client	The user_notification_service background task sends a emits the notifiation to a connected client and sets the time_sent to the current time and setting the column set to True.	As expected	
N.4A.I	Removal of notifications that are over their expiration time. By the server background service	The notifications from notifications_sent are removed as to stop the user_notification_service picking them up and sending	As expected	

		expired notifications.		

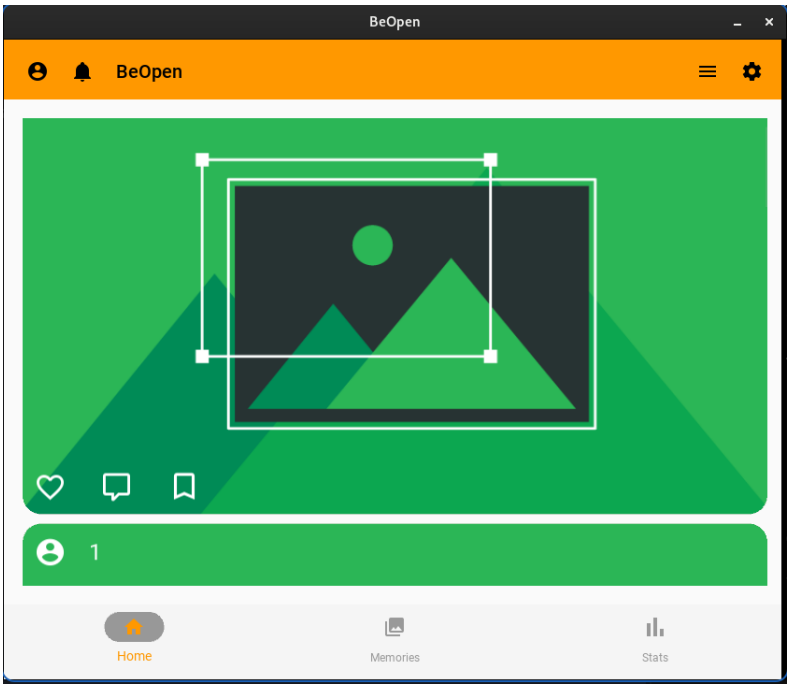
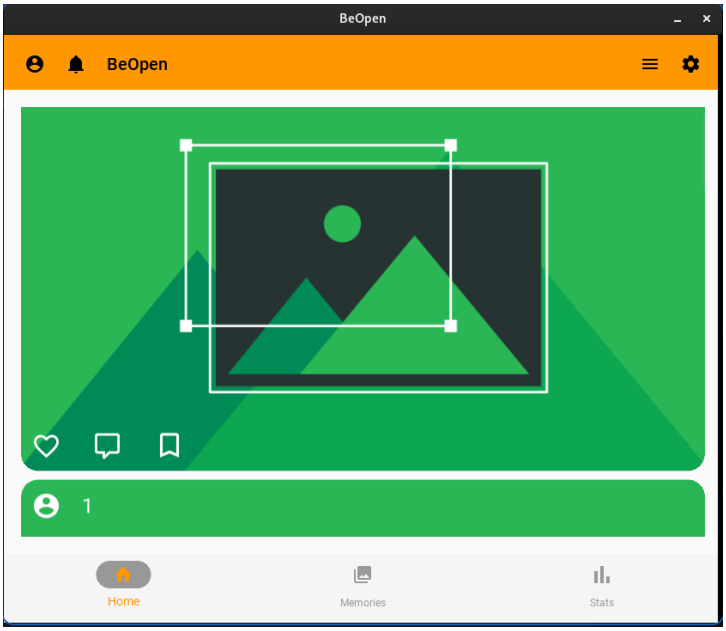
End to End tests

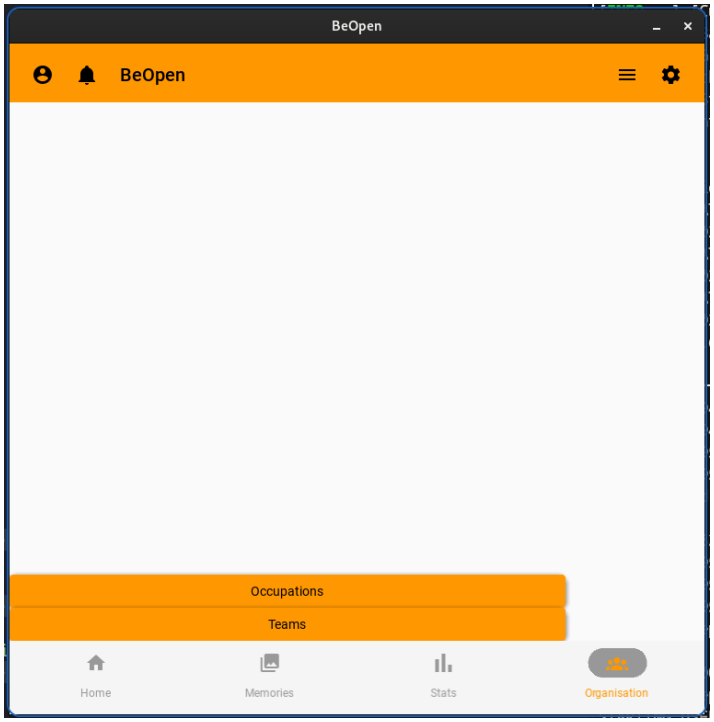
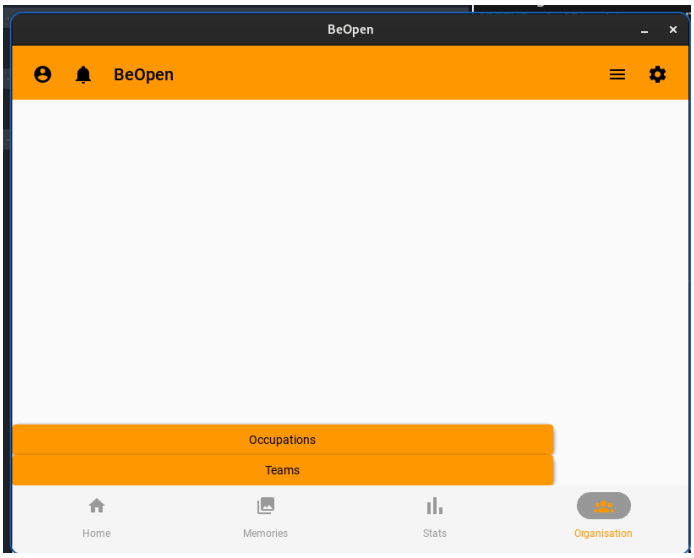
Organisation Tab

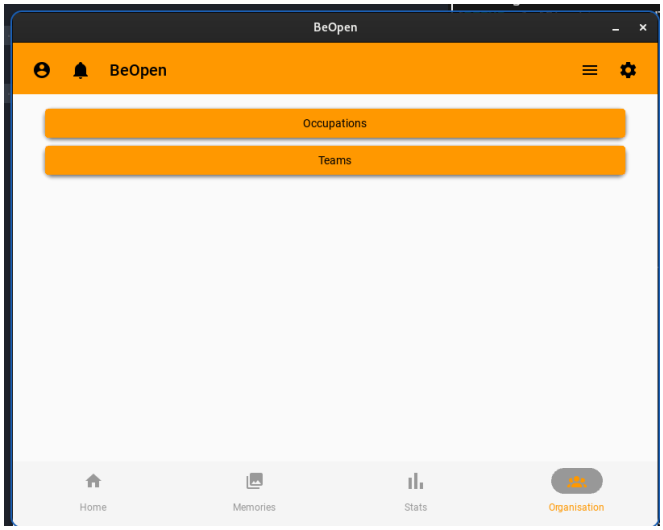
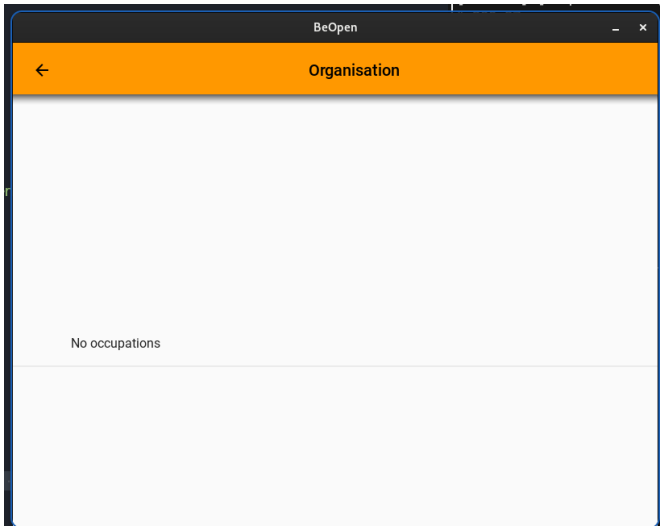
Test Number	Test Description	Expected	Observed	Action
ORG.1A.I	Checking if the organisation tab appears for a non-management (and above) and non-team leader. In this case the user will be of level **member** and not a **team leader**	Since the user is just a **member** and not a **team leader** the tab should not appear		
ORG.1B.I	Clicking the organisation tab at the bottom of the screen All users from here on will be **management** or **team leader**	Change the screen to the “organisation” screen	The tab for the organisation page did not appear	The function that adds the tab to the bottom of the screen (after checking the users level) was not called in the homepage init method
ORG.1B.II	Clicking the organisation tab at the bottom of the screen All users from here on will be **management** or	Change the screen to the “organisation” screen	As expected	

	team leader			
ORG.1C.I	Checking the UI and the look of the page	There should be a top bar displaying the app name, a settings icon, a profile icon and an additional menu for sorting the homefeed. The content of the page itself should contain 2 buttons teams and	The 2 buttons are squished down in the bottom corner of the page	Added some padding to the page itself and gave both buttons a position hint to be centred in the x direction. Also added scrolling functionality to the page to keep the pages consistent
ORG.1C.II	Checking the UI and the look of the page	There should be a top bar displaying the app name, a settings icon, a profile icon and an additional menu for sorting the homefeed. The content of the page itself should contain 2 buttons teams and	As expected	
ORG.2A.I	Pressing the “occupations” button	This should change your page to the occupations management page	As expected	
ORG.2B.I	Pressing the “teams” button	This should change your page to the		

		occupations management page		

Test Number	Image Number	Image
ORG.1A.I	1	
ORG.1B.I	1	

ORG.1B.II	1	
ORG.1C.I	1	

ORG.1C.II	1	
ORG.2A.I	1	

Login and Register

Test Number	Test Description	Expected	Observed	Action
U.1A.I	The login from the client ui, entering a **correct** username and password	After pressing login the client will receive a status message (not displayed on ui) and the screen will switch to the homepage	As expected	Added a popup message at the bottom of the screen on successful login
U.1B.I	The login from the client ui, entering a **Incorrecct** username and password	After pressing the login button the text field should go red indicating an error	As expected	Need to display what the problem is to the user (incorrect details or connection error)
U.1B.II	Implemented a popup that displays the error message from the server to the user.	The textfields should go red and an error message should popup at the bottom	As expected	
U.1C.I	The login from the client ui, entering a **Incorrecct** username and a **correct** password	The textfields should go red and an error message should popup at the bottom	As expected	
U.1D.I	The login from the client ui, entering a c**orrecct** username and a **incorrect** password	The textfields should go red and an error message should popup at the bottom	As expected	
U.1E.I	The login from	The textfields	As expected	

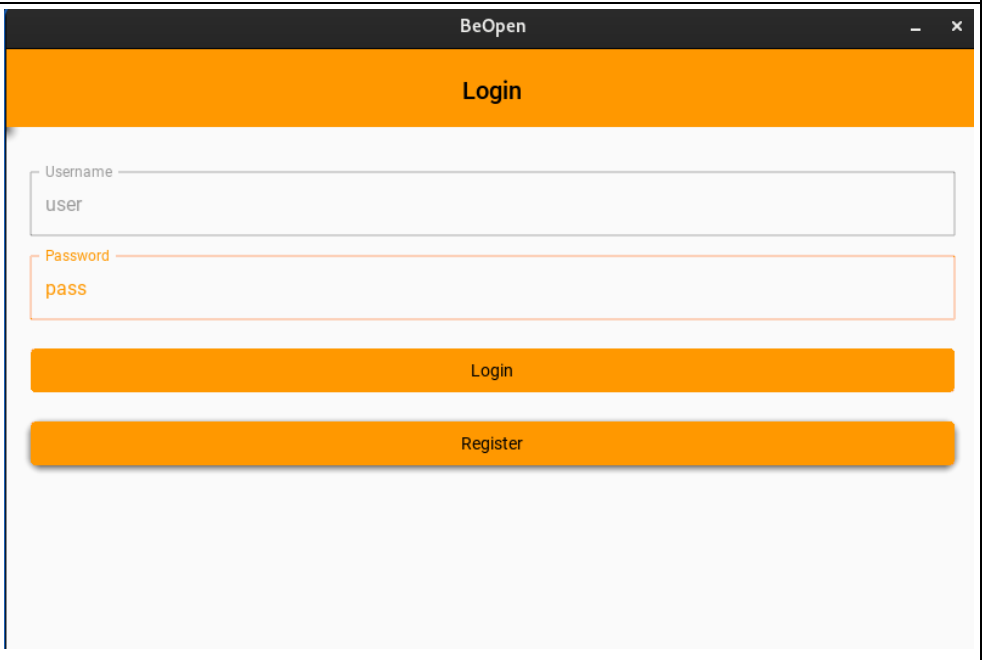
	the client ui, entering a **correct** username and **no** password	should go red and an error message should popup at the bottom		
U.1F.I	The login from the client ui, entering a **no** username and a **correct** password	The textfields should go red and an error message should popup at the bottom	As expected	
U.1G.I	The login from the client ui, entering a **no** username and a **no** password	The textfields should go red and an error message should popup at the bottom	As expected	
U.2A.I	Clicking the register button	Changes the screeen to the registration screen with all the correct fields	As expected	
U.2B.I	Clicking the Admin Register button	Changes the text in the Registration key field to display Admin Registration key and changes the button text to Member register	As expected	
U.2C.I	Member registration with valid credentials	After clicking register the page should change back to the login and a	As expected	

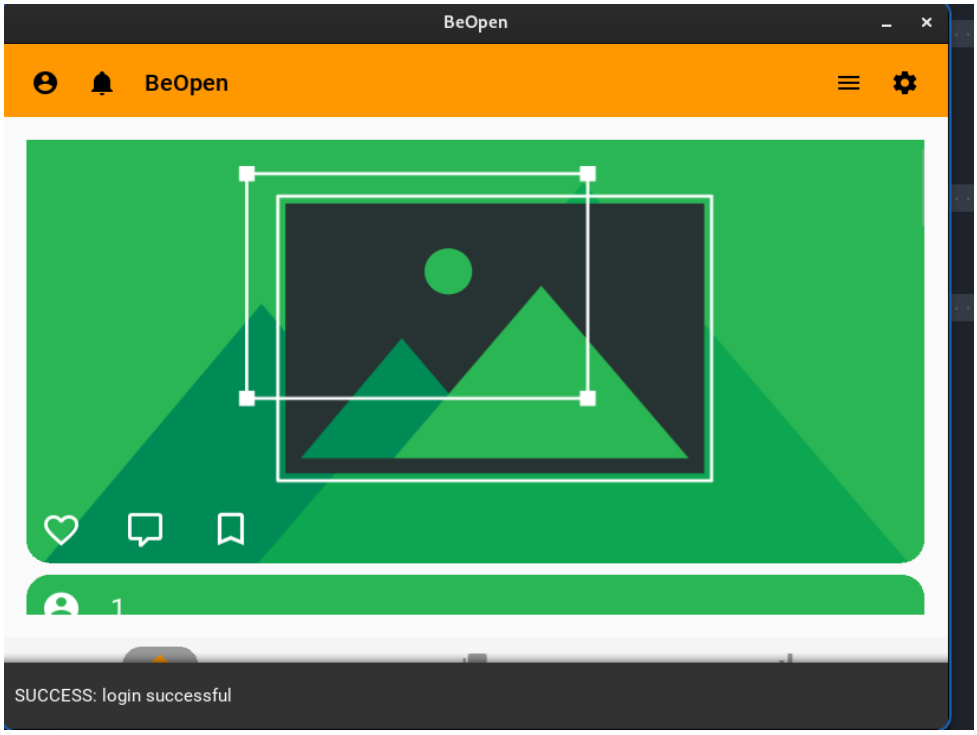
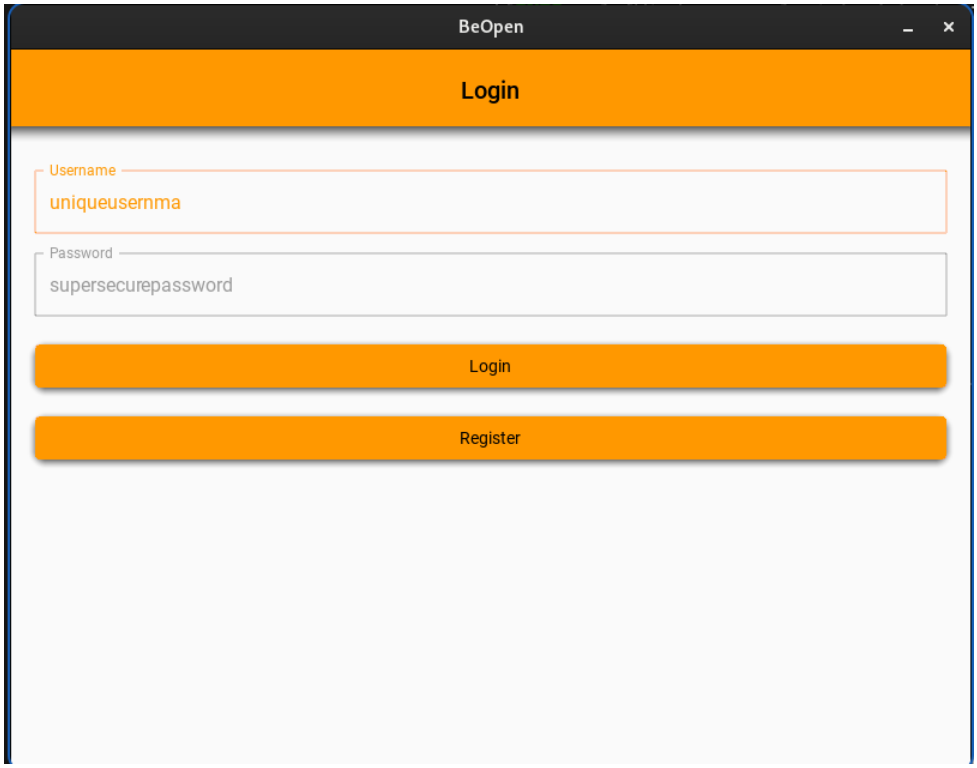
		popup message indicating a successful registration.		
U.2D.I	For these following tests the server is configured to accept usernames between 3 and 25 characters long (inclusive). Member registration with a username thats 3 characters long	After clicking register the page should change back to the login and a popup message indicating a successful registration.	As expected	
U.2E.I	Member registration with a username that is less than 3 characters long	After clicking register a popup message with a error stating an problem with the length of the username	As expected	
U.2F.I	Member registration with a username that is 25 characters long	After clicking register the page should change back to the login and a popup message indicating a successful registration.	As expected	
U.2G.I	Member registration with a username that is more than 25	After clicking register a popup message with a error stating an	As expected	

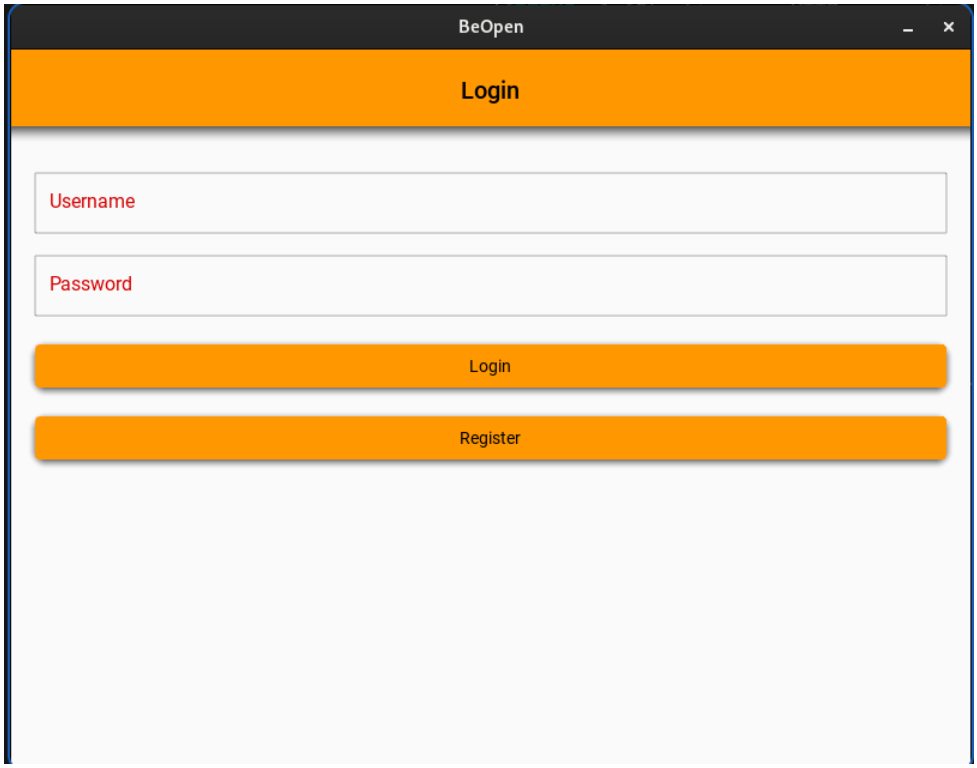
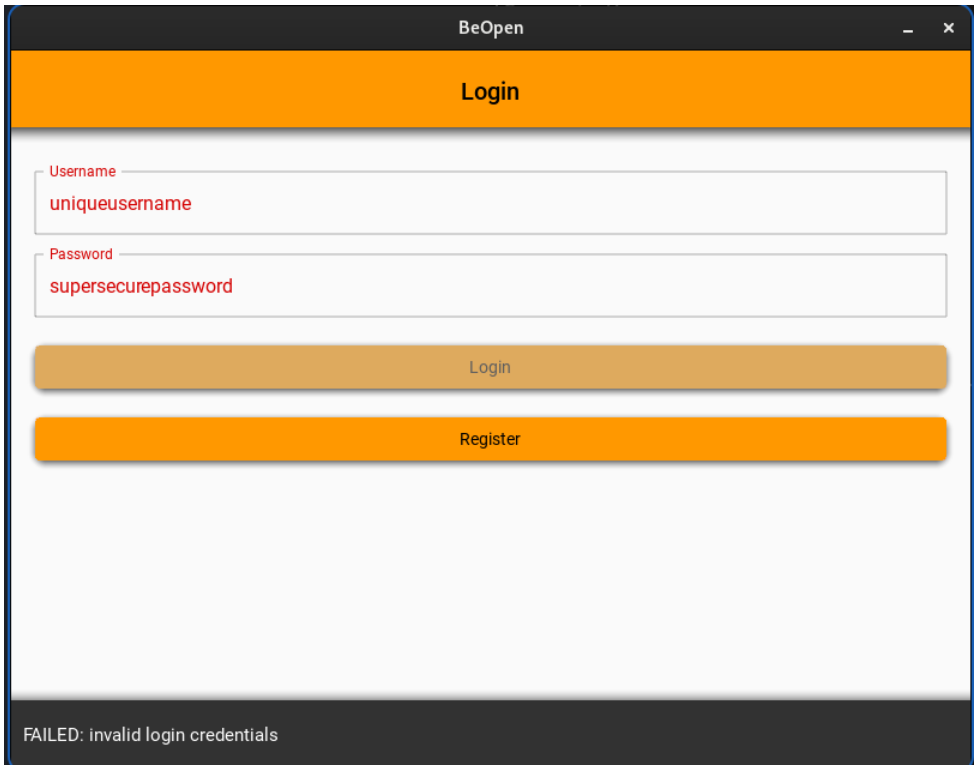
	characters long	problem with the length of the username		
U.3A.I	Admin registration with valid credentials	On register click their should be a popup for successful registration and the screen should switch back to the login screen. The server should also add corresponding entries in the tables	Didnt register with the correct label. Reason is the client is not emitting the reg_admin event	Added a class attribute to define the registration mode (member or admin) on registration submit, it sets the event call based on the mode
U.3A.II	Admin registration with valid credentials	On register click their should be a popup for successful registration and the screen should switch back to the login screen. The server should also add corresponding entries in the tables	As expected	
U.3B.I	Admin registration with no credentials	On register click their should be a popup for an unsuccessful registration. The server	As expected	

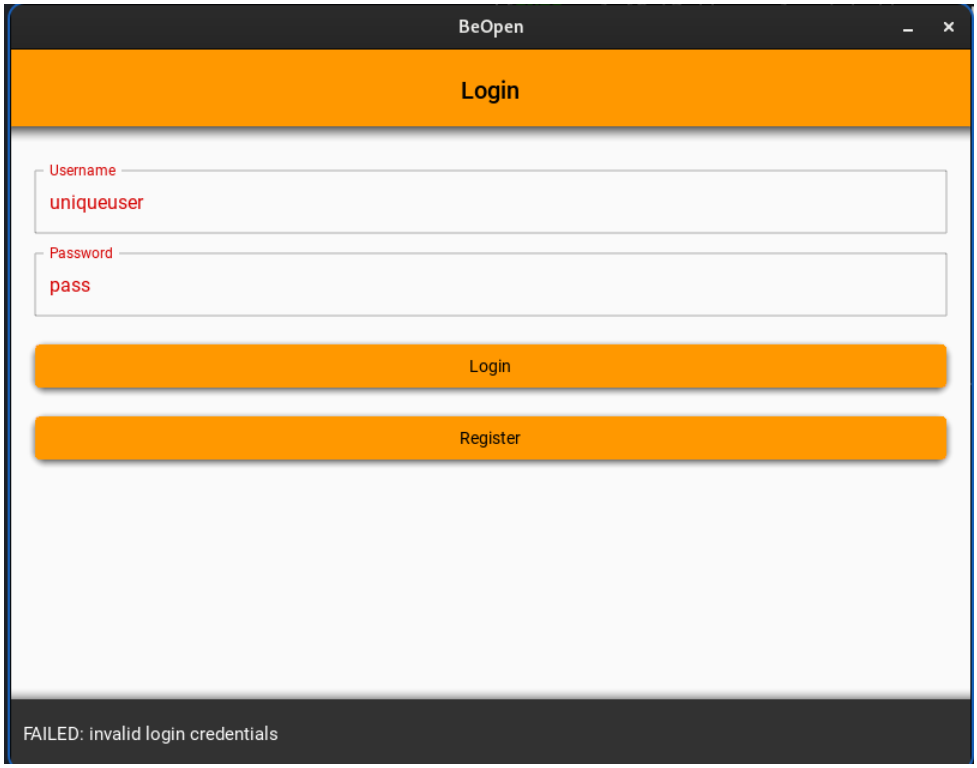
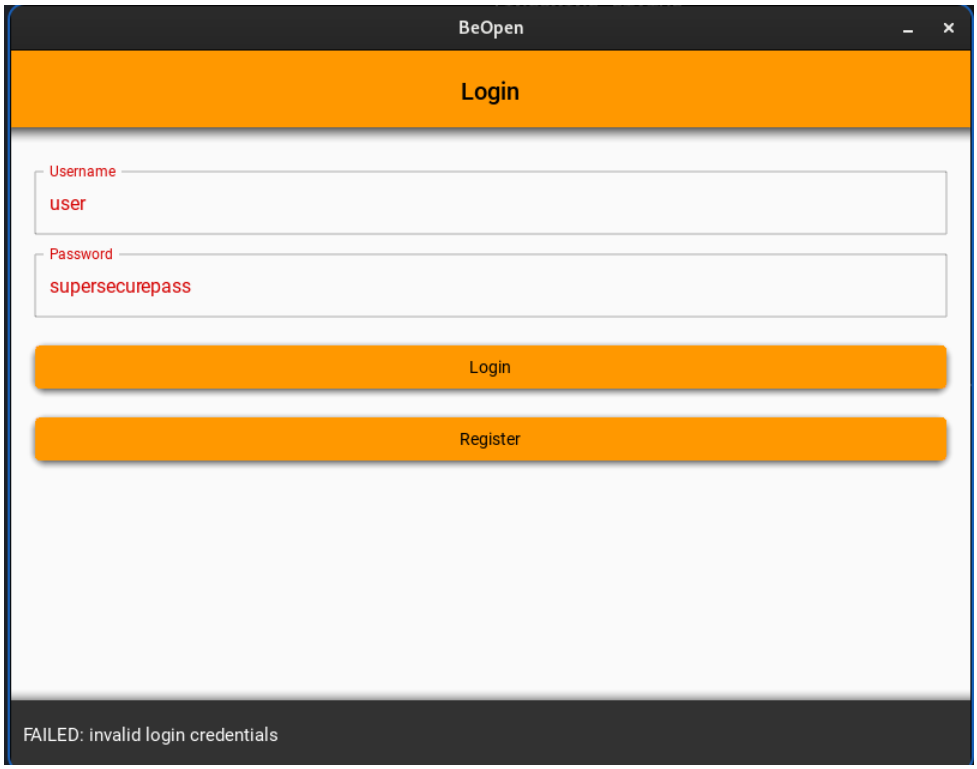
		should not create any table entries		
U.4A.I	Entering a **valid** url of the server (hosted locally) and clicking connect	The page should display an appropriate popup message at the bottom of the screen and change the page to the login page	The url was counted as invalid. This was because there was no way currently for the start_client function to communicate whether it was successful or not.	Added a try a return statement to start client so it returns True when successfully connected and False when unsuccessful
U.4A.II	Entering a **valid** url of the server (hosted locally) and clicking connect	The page should display an appropriate popup message at the bottom of the screen and change the page to the login page	The url continues to count as invalid. This is caused by the urllib error checking process which denys any urls that point to localhost.	This package was remove and replaced with a try and except in the sio connection
U.4A.III	Entering a **valid** url of the server (hosted locally) and clicking connect	The page should display an appropriate popup message at the bottom of the screen and change the page to the login page	There was no popup message at the bottom of the page.	Add a popup message for when both successful connection and unsuccessful connections
U.4A.IIII	Entering a **valid** url of the server (hosted locally) and clicking	The page should display an appropriate popup message	As expected	

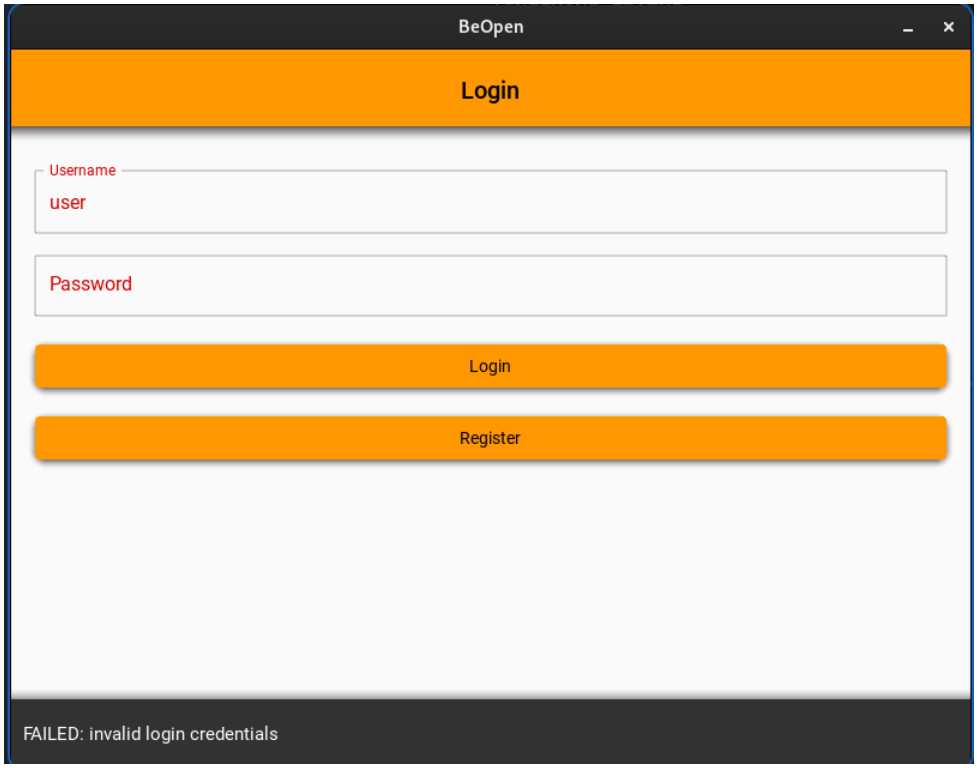
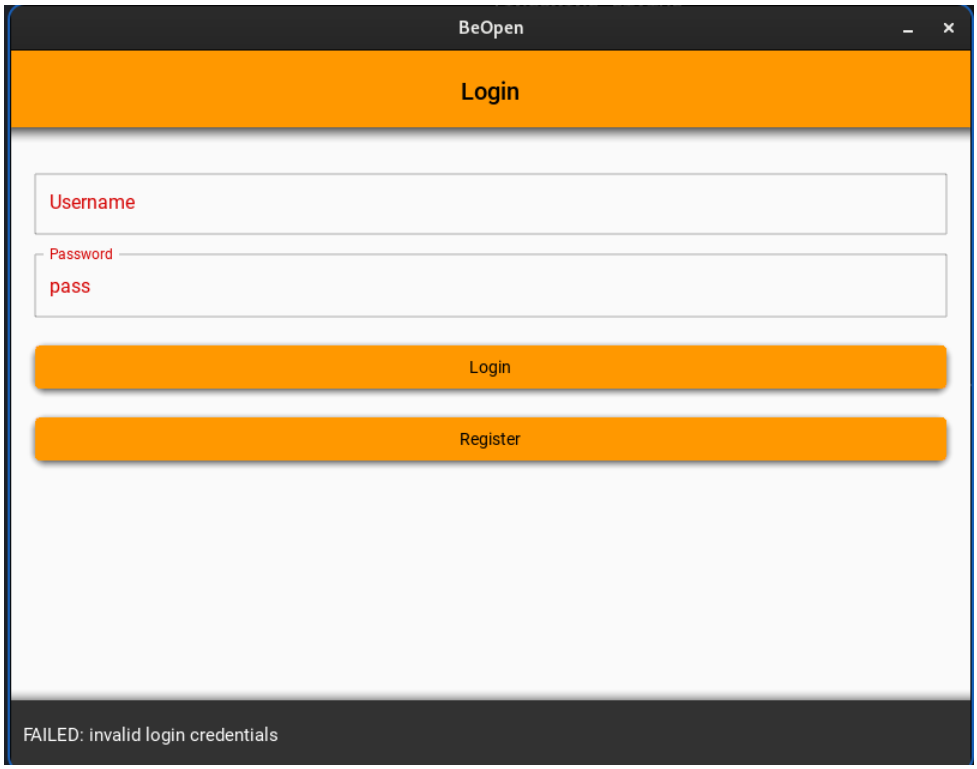
	connect	at the bottom of the screen and change the page to the login page		
U.4B.I	Entering an **invalid** url and clicking connect	The page should display an **unsuccessful** connection message. It should also make the textbox go red and remind the user to include http:// or https://	As expected	

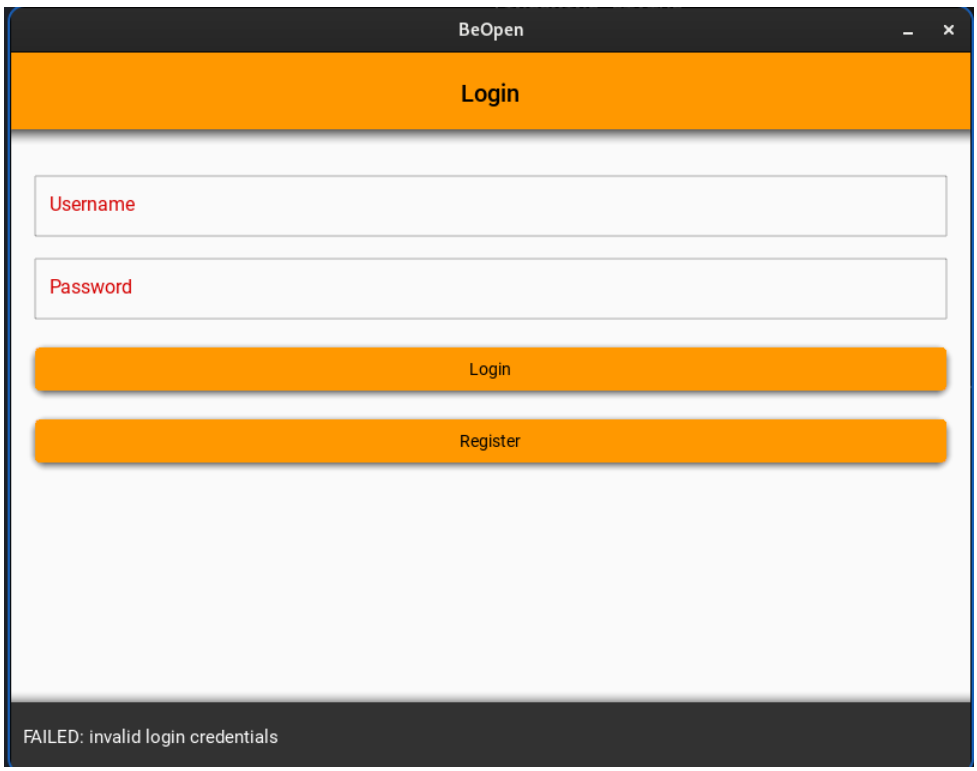
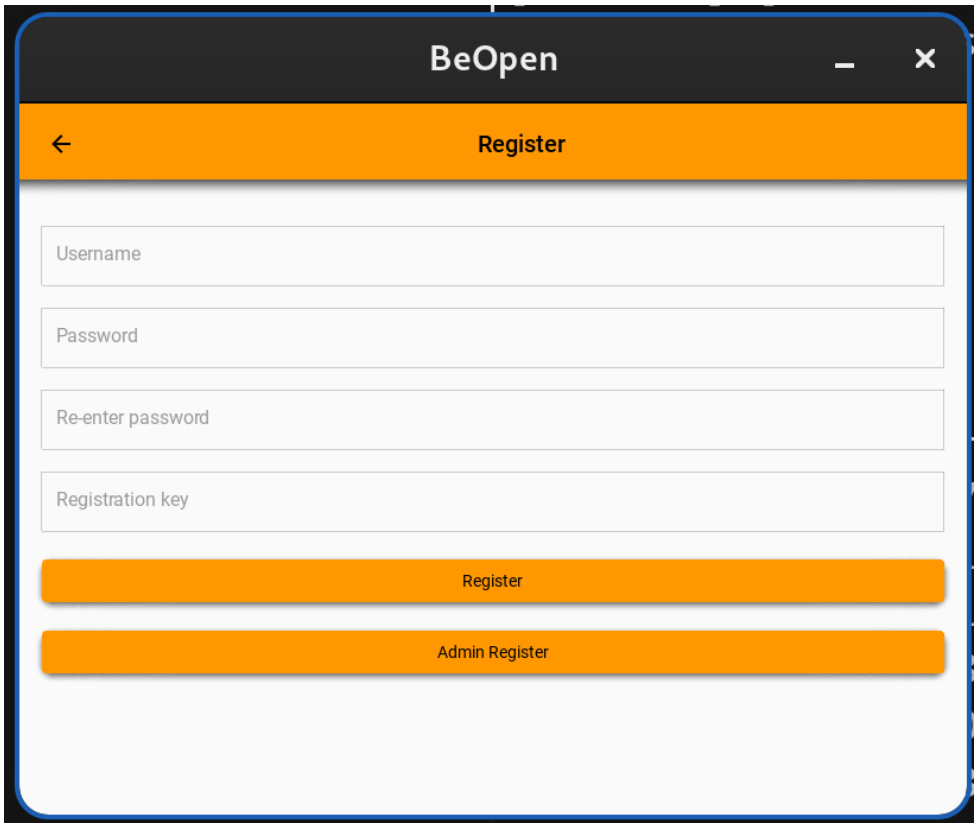
Test Number	Image Number	Image
U.1A.I	1	

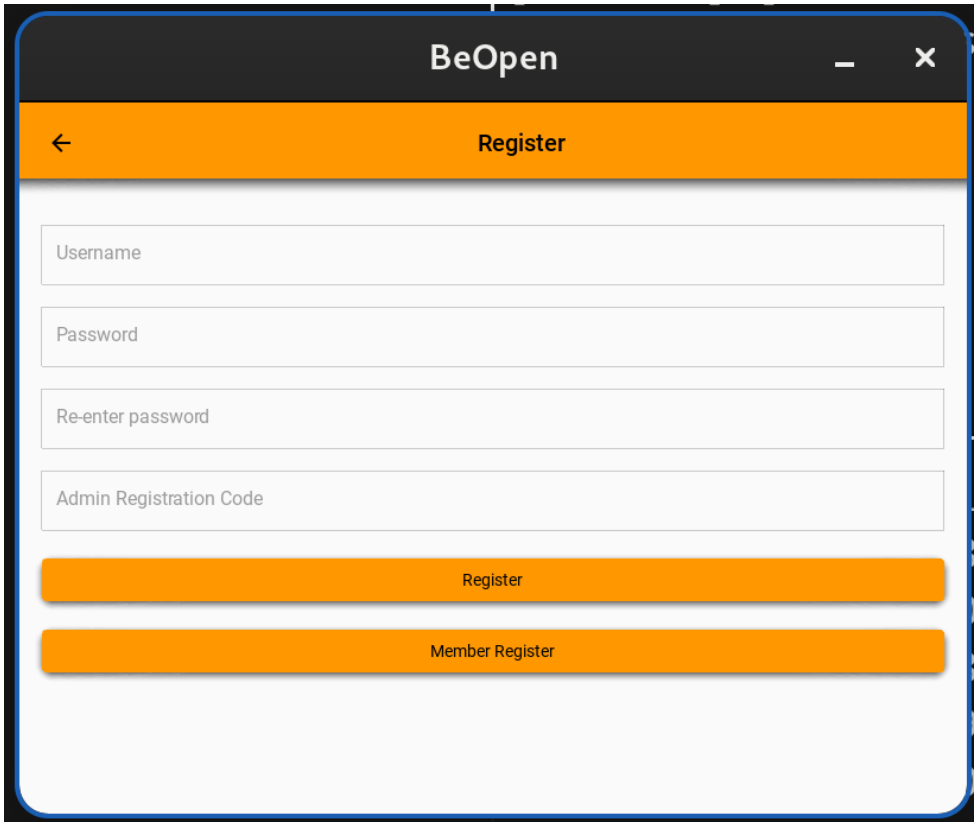
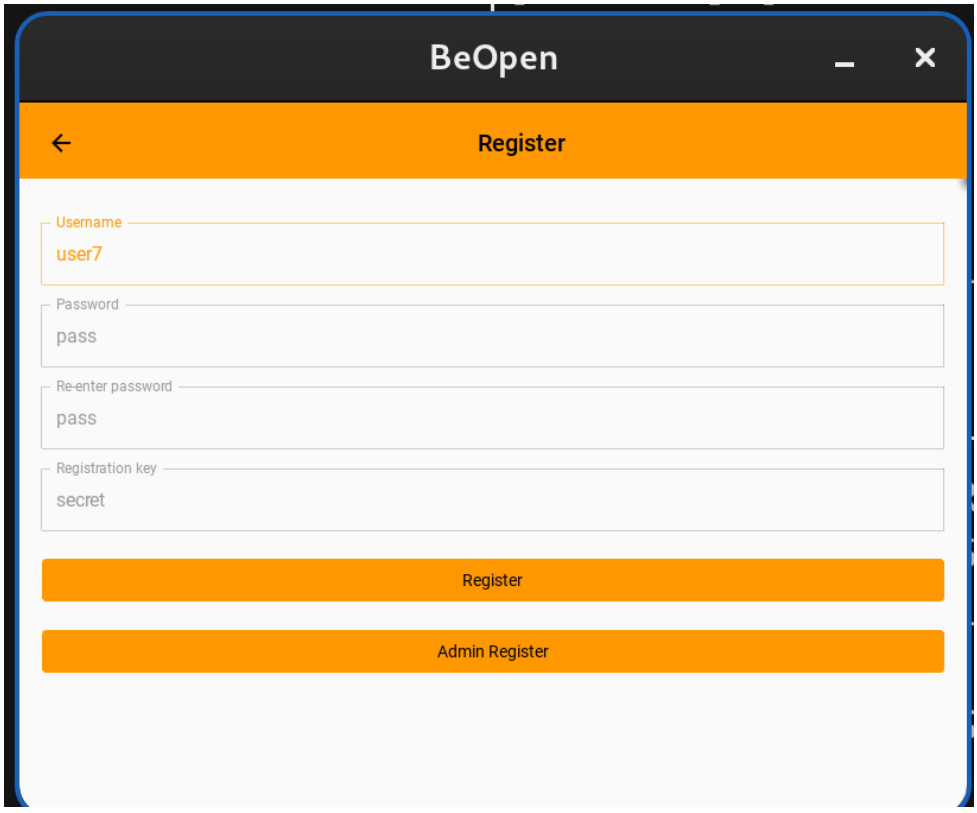
U.1A.I	2	 BeOpen BeOpen 1 SUCCESS: login successful
U.1B.I	1	 BeOpen Login Username uniqueusernma Password supersecurepassword Login Register

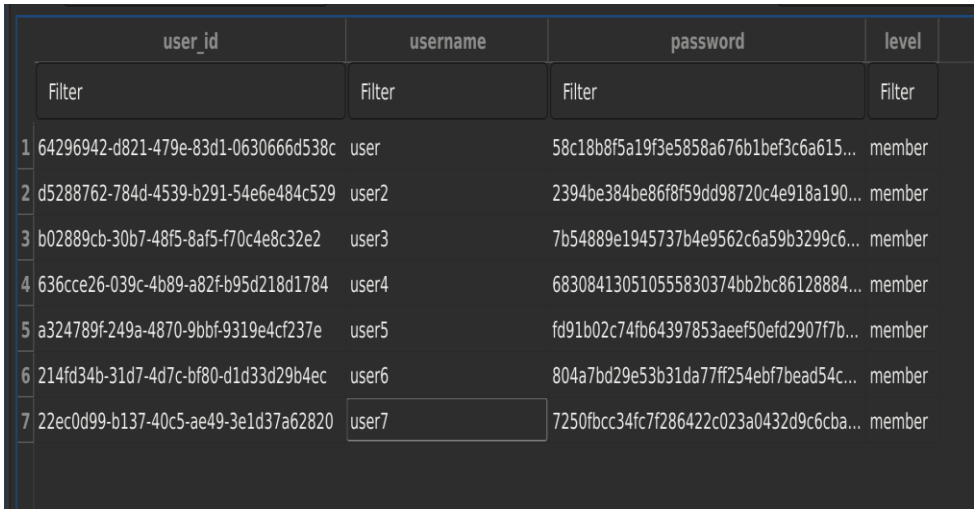
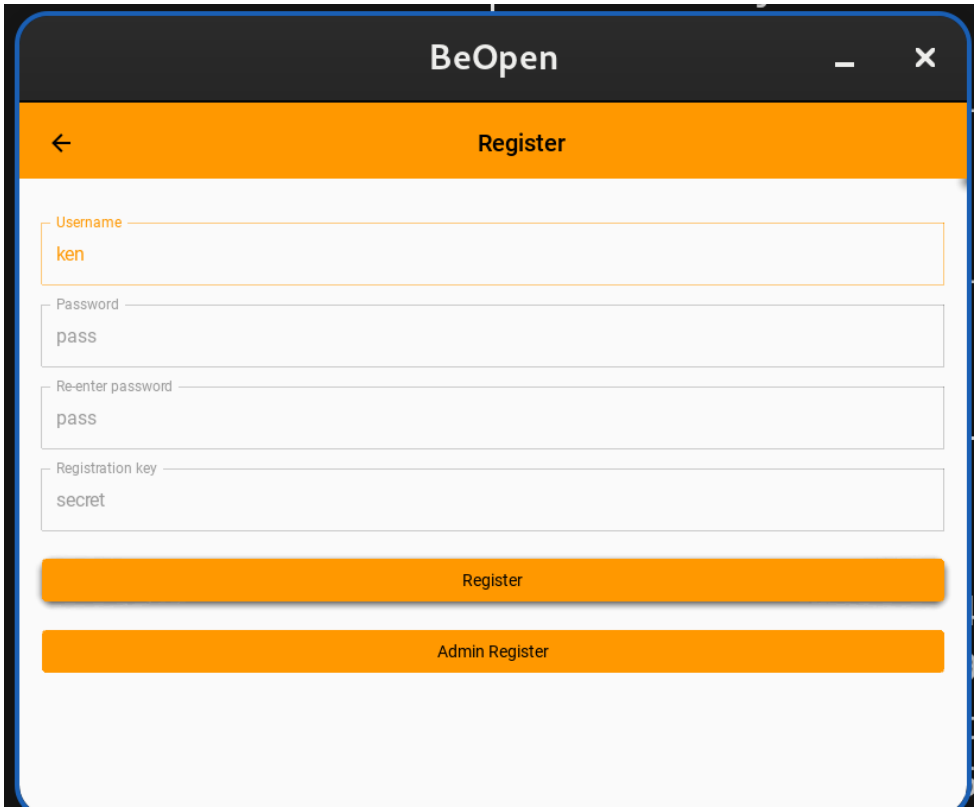
U.1B.I	2	
U.1B.II	1	

U.1C.I	1	 BeOpen Login Username uniqueuser Password pass Login Register FAILED: invalid login credentials
U.1D.I	1	 BeOpen Login Username user Password supersecurepass Login Register FAILED: invalid login credentials

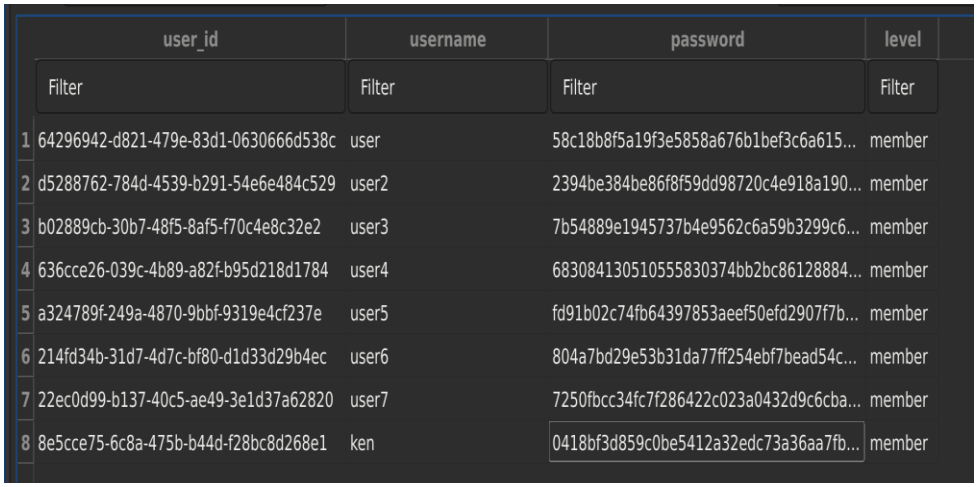
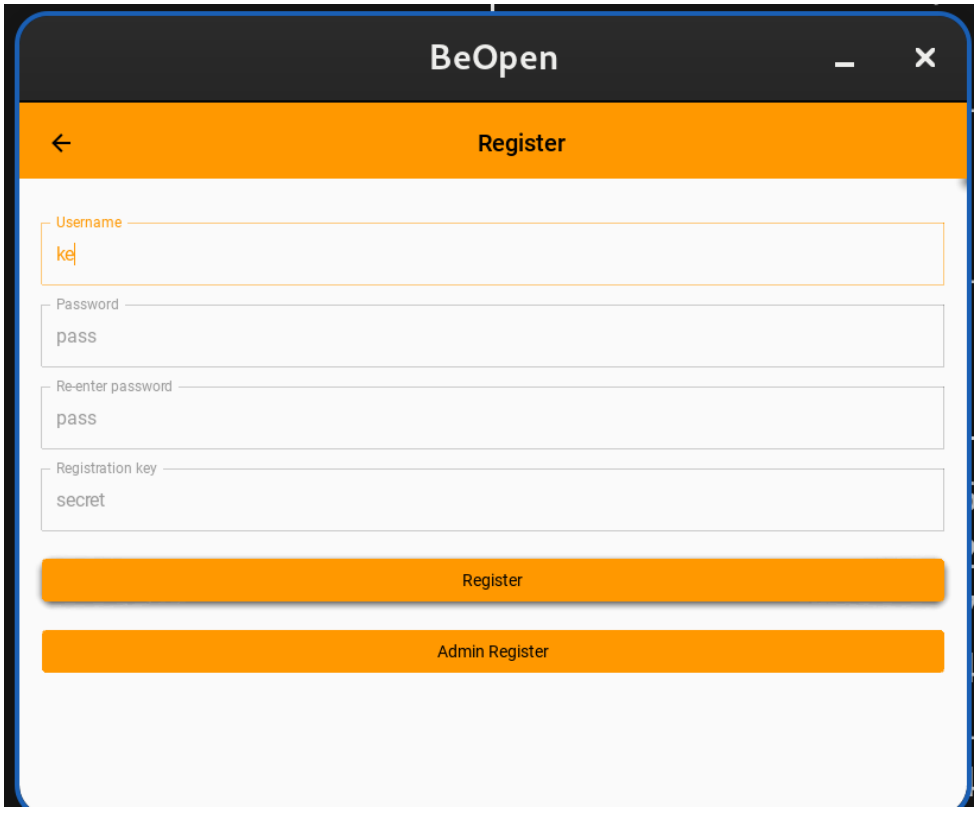
U.1E.I	1	 BeOpen Login Username user Password Login Register FAILED: invalid login credentials
U.1F.I	1	 BeOpen Login Username Password pass Login Register FAILED: invalid login credentials

U.1G.I	1	 BeOpen Login Username Password Login Register FAILED: invalid login credentials
U.2A.I	1	 BeOpen Register Username Password Re-enter password Registration key Register Admin Register

U.2B.I	1	 BeOpen Register Username Password Re-enter password Admin Registration Code Register Member Register
U.2C.I	1	 BeOpen Register Username user7 Password pass Re-enter password pass Registration key secret Register Admin Register

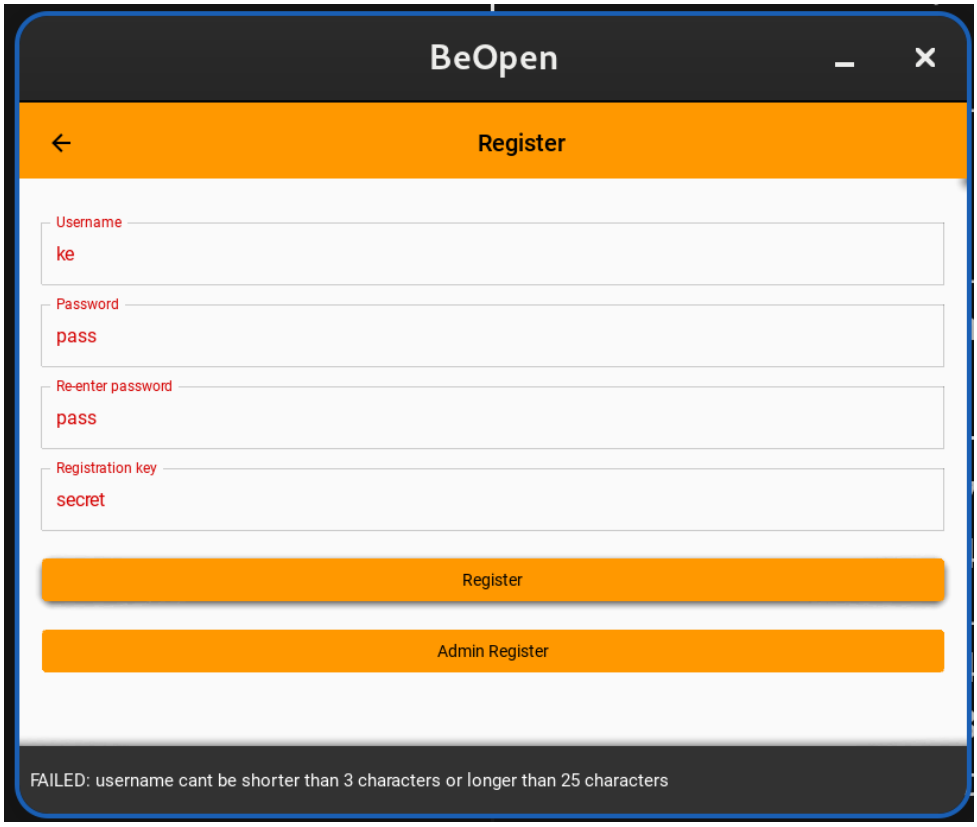
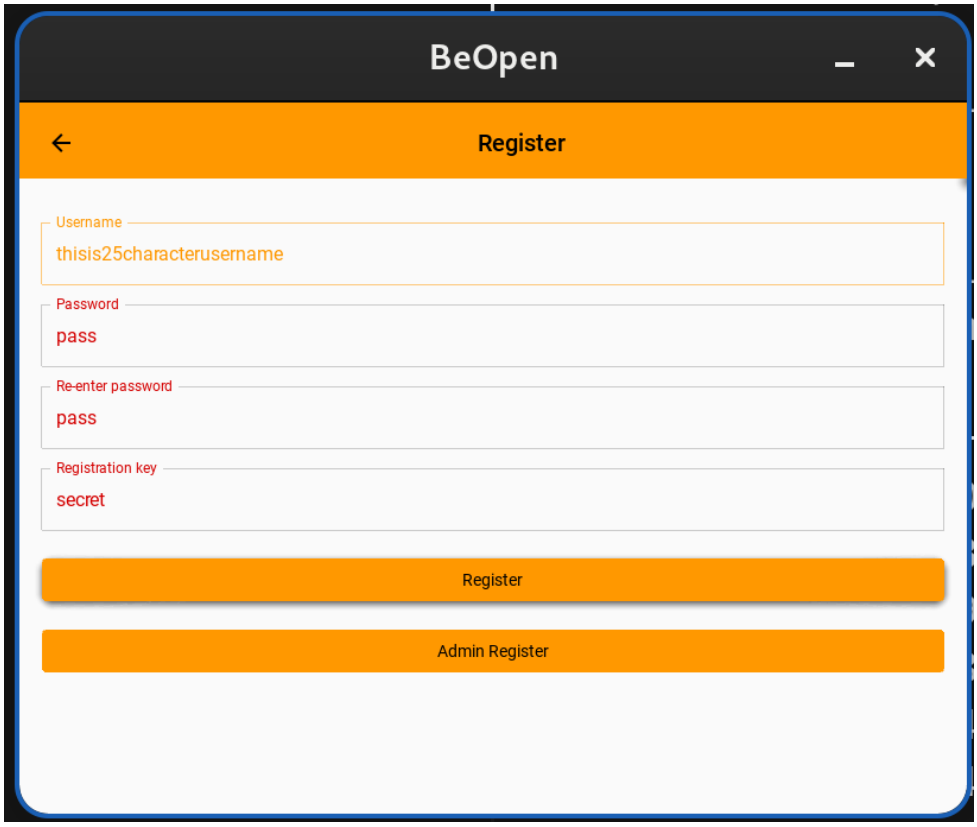
U.2C.I	2	
U.2D.I	1	

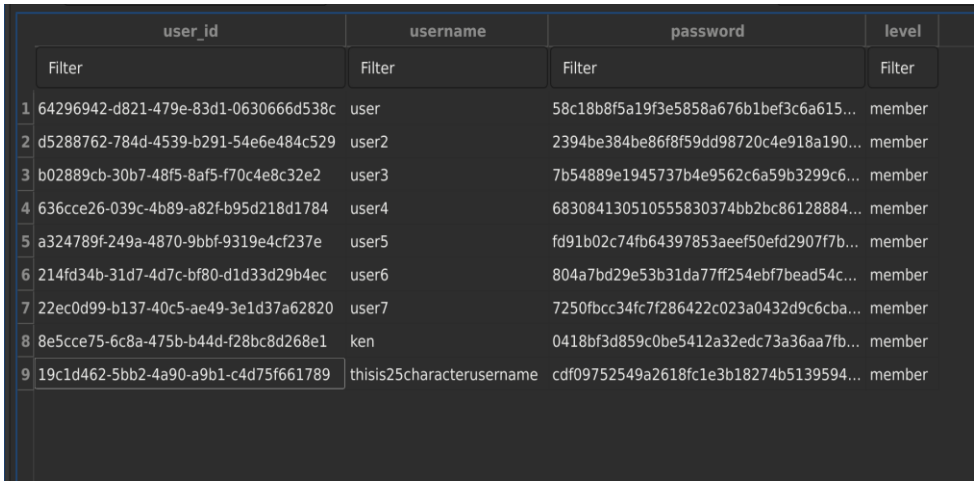
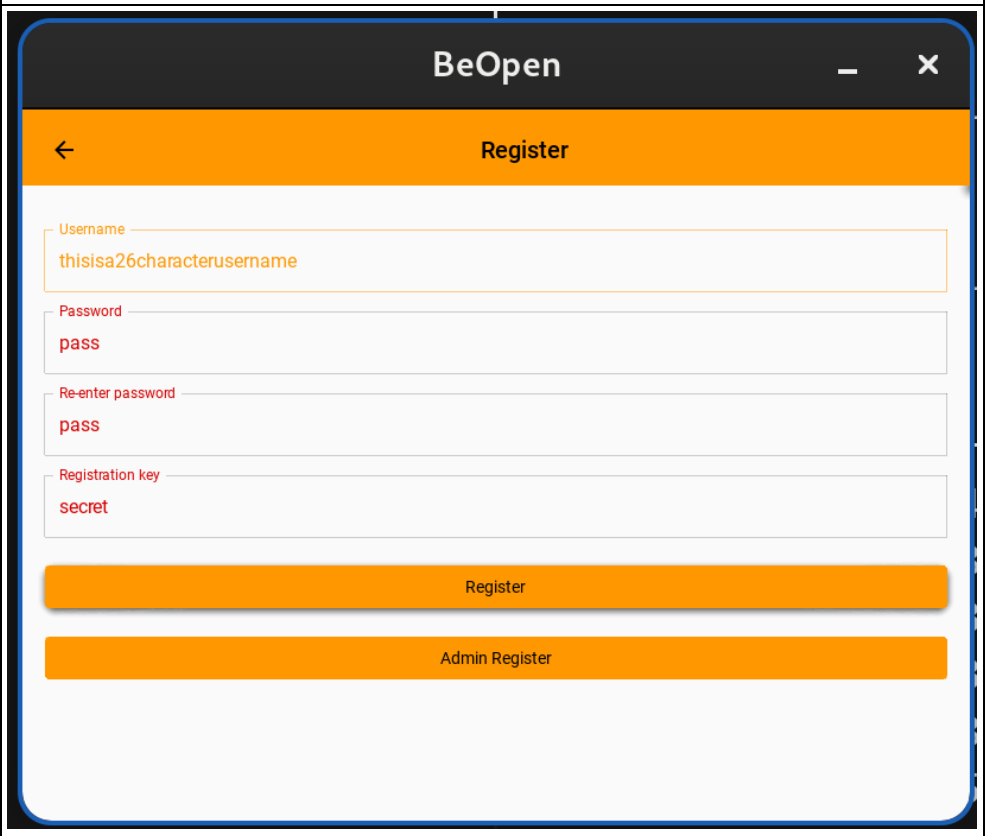
user_id
username
password
level
Filter
Filter
Filter
Filter
1 64296942-d821-479e-83d1-0630666d538c user 58c18b8f5a19f3e5858a676b1bef3c6a615... member
2 d5288762-784d-4539-b291-54e6e484c529 user2 2394be384be86f8f59dd98720c4e918a190... member
3 b02889cb-30b7-48f5-8af5-f70c4e8c32e2 user3 7b54889e1945737b4e9562c6a59b3299c6... member
4 636cce26-039c-4b89-a82f-b95d218d1784 user4 6830841305105558303 74bb2bc86128884... member
5 a324789f-249a-4870-9bbf-9319e4cf237e user5 fd91b02c74fb64397853aeef50efd2907f7b... member
6 214fd34b-31d7-4d7c-bf80-d1d33d29b4ec user6 804a7bd29e53b31da77ff254ebf7bead54c... member
7 22ec0d99-b137-40c5-ae49-3e1d37a62820 user7 7250fbcc34fc7f286422c023a0432d9c6cba... member
BeOpen
Register
Username
ken
Password
pass
Re-enter password
pass
Registration key
secret
Register
Admin Register

U.2D.I	2	
U.2E.I	1	

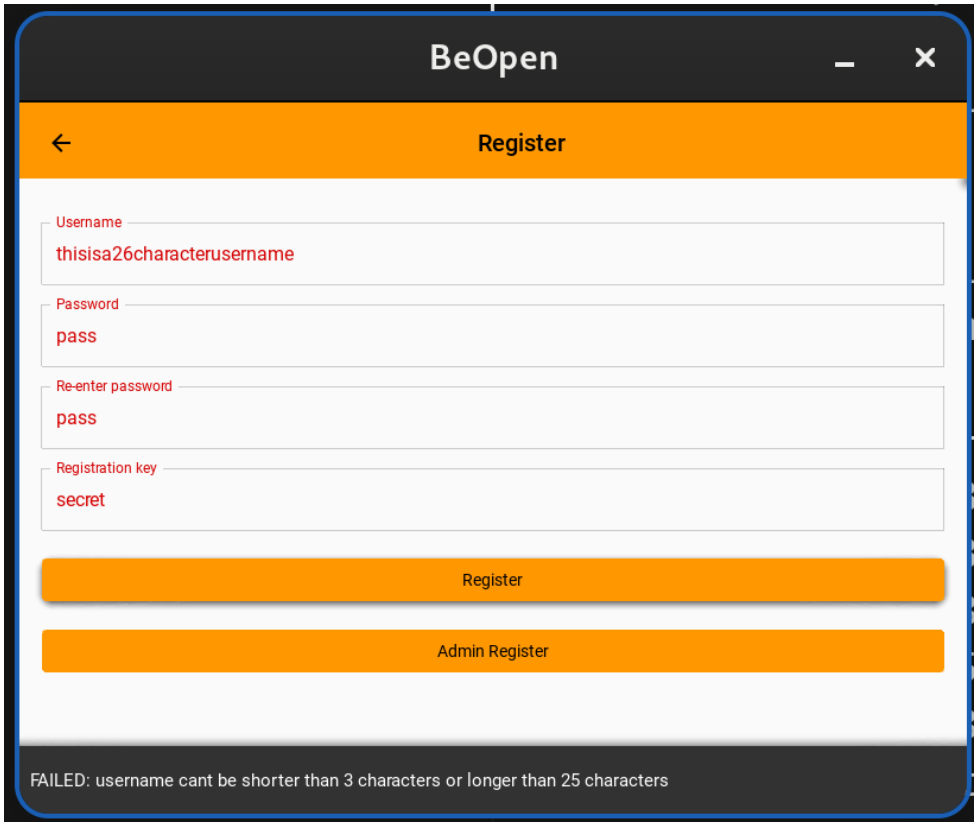
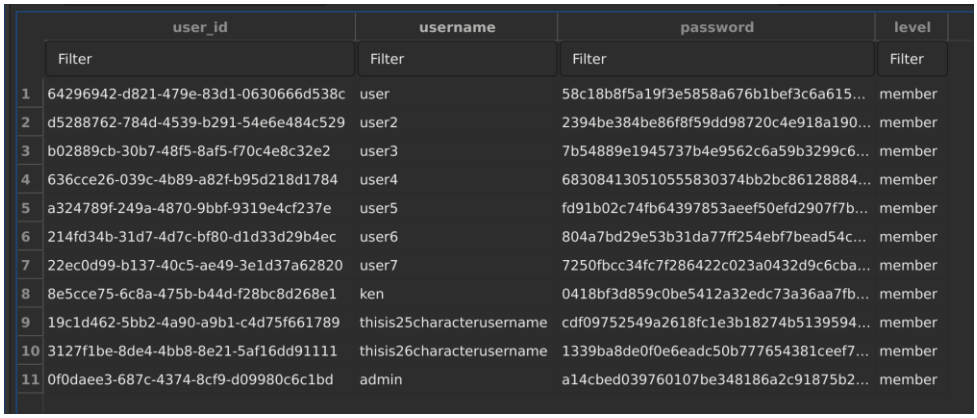
user_id
username
password
level
Filter
Filter
Filter
Filter
1 64296942-d821-479e-83d1-0630666d538c user 58c18b8f5a19f3e5858a676b1bef3c6a615... member
2 d5288762-784d-4539-b291-54e6e484c529 user2 2394be384be86f8f59dd98720c4e918a190... member
3 b02889cb-30b7-48f5-8af5-f70c4e8c32e2 user3 7b54889e1945737b4e9562c6a59b3299c6... member
4 636cce26-039c-4b89-a82f-b95d218d1784 user4 6830841305105558303 74bb2bc86128884... member
5 a324789f-249a-4870-9bbf-9319e4cf237e user5 fd91b02c74fb64397853aeef50efd2907f7b... member
6 214fd34b-31d7-4d7c-bf80-d1d33d29b4ec user6 804a7bd29e53b31da77ff254ebf7bead54c... member
7 22ec0d99-b137-40c5-ae49-3e1d37a62820 user7 7250fbcc34fc7f286422c023a0432d9c6cba... member
8 8e5cce75-6c8a-475b-b44d-f28bc8d268e1 ken 0418bf3d859c0be5412a32edc73a36aa7fb... member

BeOpen
Register
Username
ke
Password
pass
Re-enter password
pass
Registration key
secret
Register
Admin Register

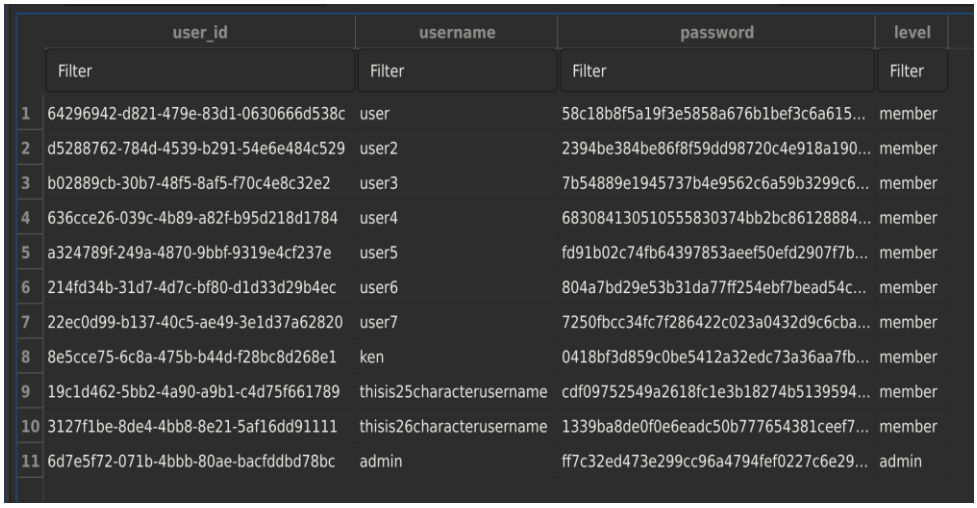
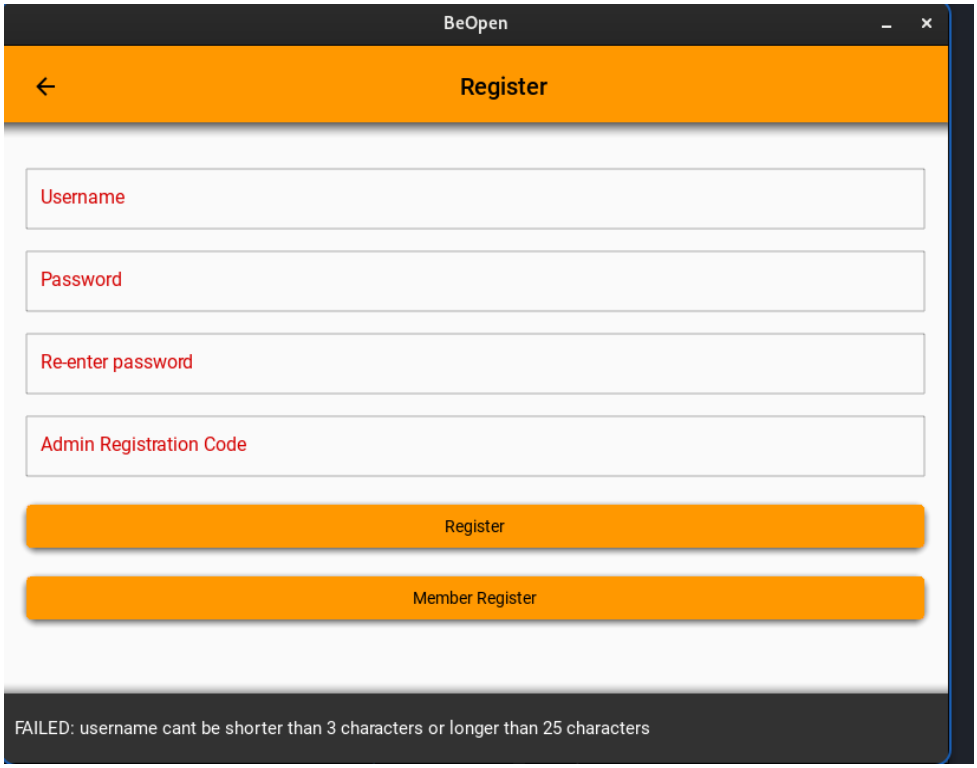
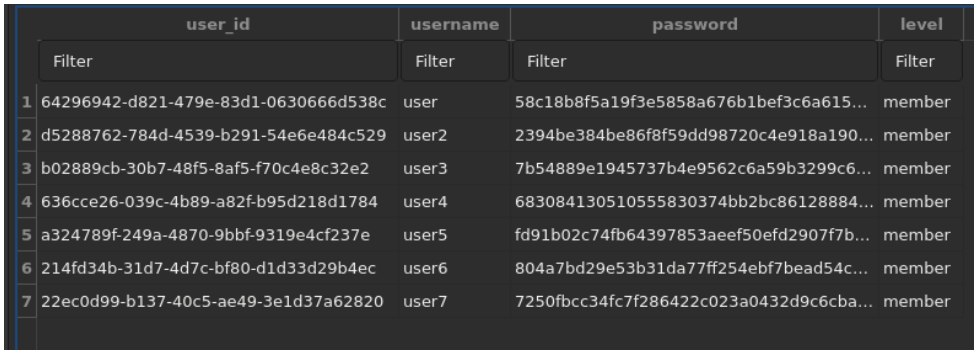
U.2E.I	2	 BeOpen Register Username ke Password pass Re-enter password pass Registration key secret Register Admin Register FAILED: username cant be shorter than 3 characters or longer than 25 characters
U.2F.I	1	 BeOpen Register Username thisis25characterusername Password pass Re-enter password pass Registration key secret Register Admin Register

U.2F.I	2	
U.2G.I	1	

	user_id	username	password	level
	Filter	Filter	Filter	Filter
1	64296942-d821-479e-83d1-0630666d538c	user	58c18b8f5a19f3e5858a676b1bef3c6a615...	member
2	d5288762-784d-4539-b291-54e6e484c529	user2	2394be384be86f8f59dd98720c4e918a190...	member
3	b02889cb-30b7-48f5-8af5-f70c4e8c32e2	user3	7b54889e1945737b4e9562c6a59b3299c6...	member
4	636cce26-039c-4b89-a82f-b95d218d1784	user4	6830841305105558303 74bb2bc86128884...	member
5	a324789f-249a-4870-9bbf-9319e4cf237e	user5	fd91b02c74fb64397853aeef50efd2907f7b...	member
6	214fd34b-31d7-4d7c-bf80-d1d33d29b4ec	user6	804a7bd29e53b31da77ff254ebf7bead54c...	member
7	22ec0d99-b137-40c5-ae49-3e1d37a62820	user7	7250fbcc34fc7f286422c023a0432d9c6cba...	member
8	8e5cce75-6c8a-475b-b44d-f28bc8d268e1	ken	0418bf3d859c0be5412a32edc73a36aa7fb...	member
9	19c1d462-5bb2-4a90-a9b1-c4d75f661789	thisis25characterusername	cdf09752549a2618fc1e3b18274b5139594...	member

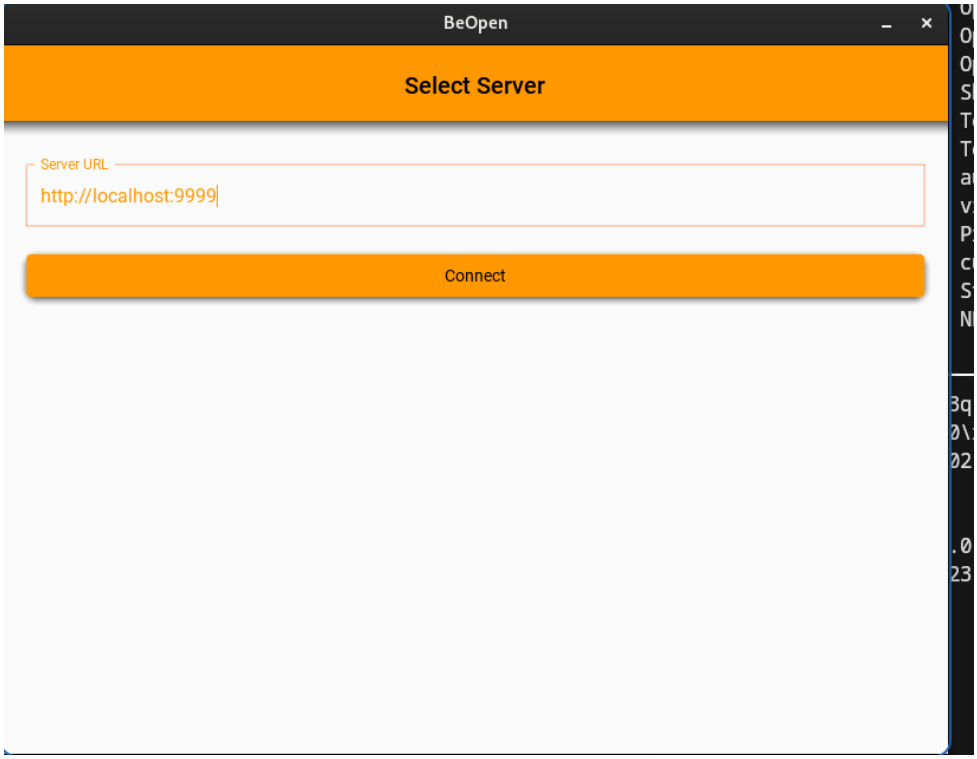
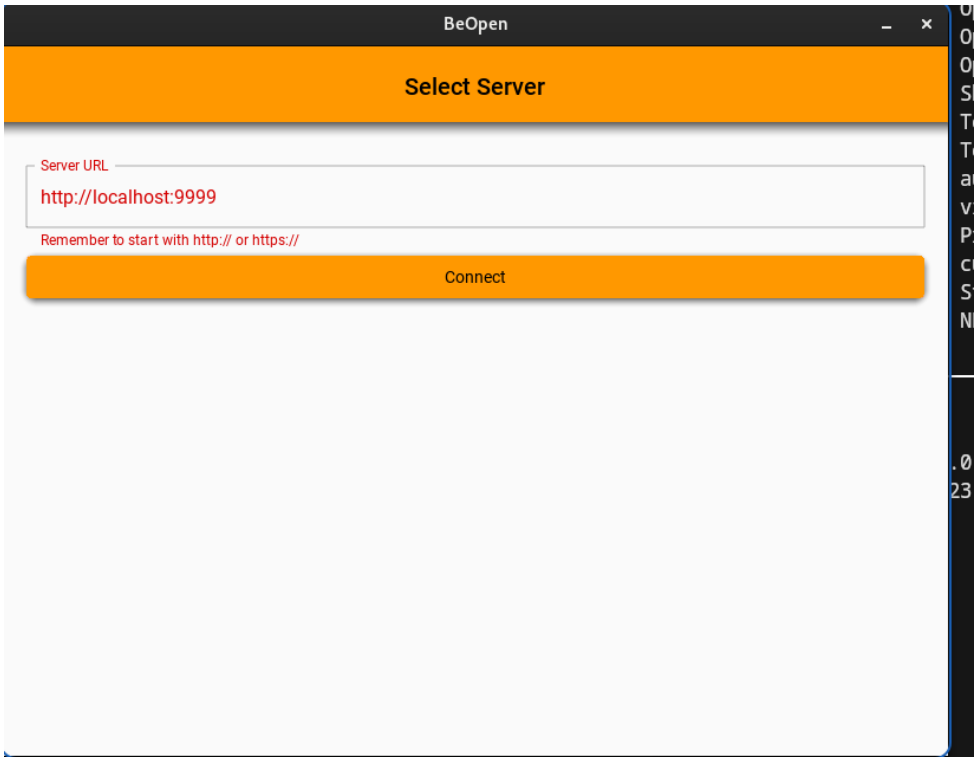
U.2G.I	2	
U.3A.I	1	

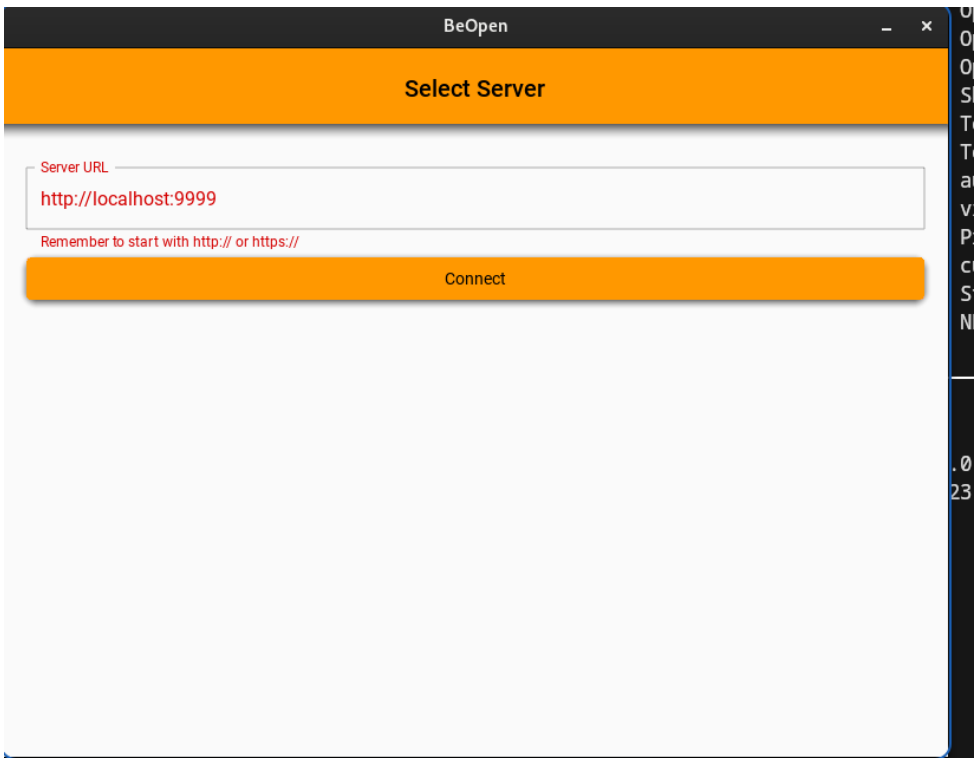
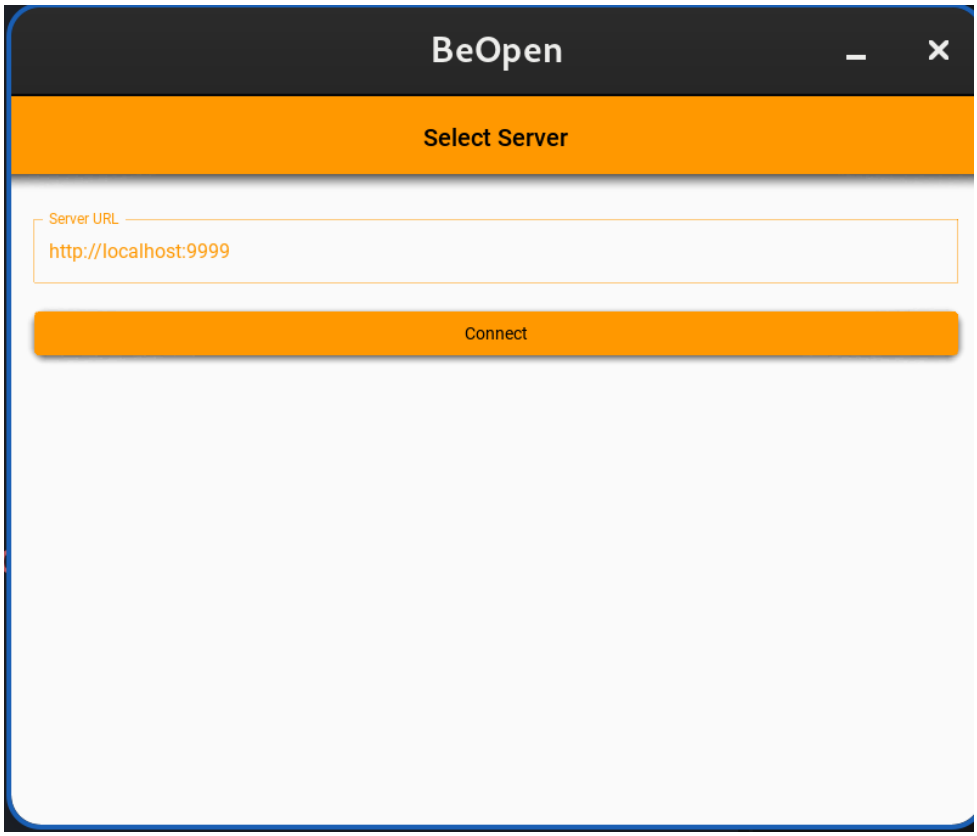
	user_id	username	password	level
	Filter	Filter	Filter	Filter
1	64296942-d821-479e-83d1-0630666d538c	user	58c18b8f5a19f3e5858a676b1bef3c6a615...	member
2	d5288762-784d-4539-b291-54e6e484c529	user2	2394be384be86f8f59dd98720c4e918a190...	member
3	b02889cb-30b7-48f5-8af5-f70c4e8c32e2	user3	7b54889e1945737b4e9562c6a59b3299c6...	member
4	636cce26-039c-4b89-a82f-b95d218d1784	user4	6830841305105558303 74bb2bc86128884...	member
5	a324789f-249a-4870-9bbf-9319e4cf237e	user5	fd91b02c74fb64397853aeef50efd2907f7b...	member
6	214fd34b-31d7-4d7c-bf80-d1d33d29b4ec	user6	804a7bd29e53b31da77ff254ebf7bead54c...	member
7	22ec0d99-b137-40c5-ae49-3e1d37a62820	user7	7250fbcc34fc7f286422c023a0432d9c6cba...	member
8	8e5cce75-6c8a-475b-b44d-f28bc8d268e1	ken	0418bf3d859c0be5412a32edc73a36aa7fb...	member
9	19c1d462-5bb2-4a90-a9b1-c4d75f661789	thisis25characterusername	cdf09752549a2618fc1e3b18274b5139594...	member
10	3127f1be-8de4-4bb8-8e21-5af16dd91111	thisis26characterusername	1339ba8de0f0e6eadc50b777654381ceef7...	member
11	0f0daee3-687c-4374-8cf9-d09980c6c1bd	admin	a14cbed039760107be348186a2c91875b2...	member

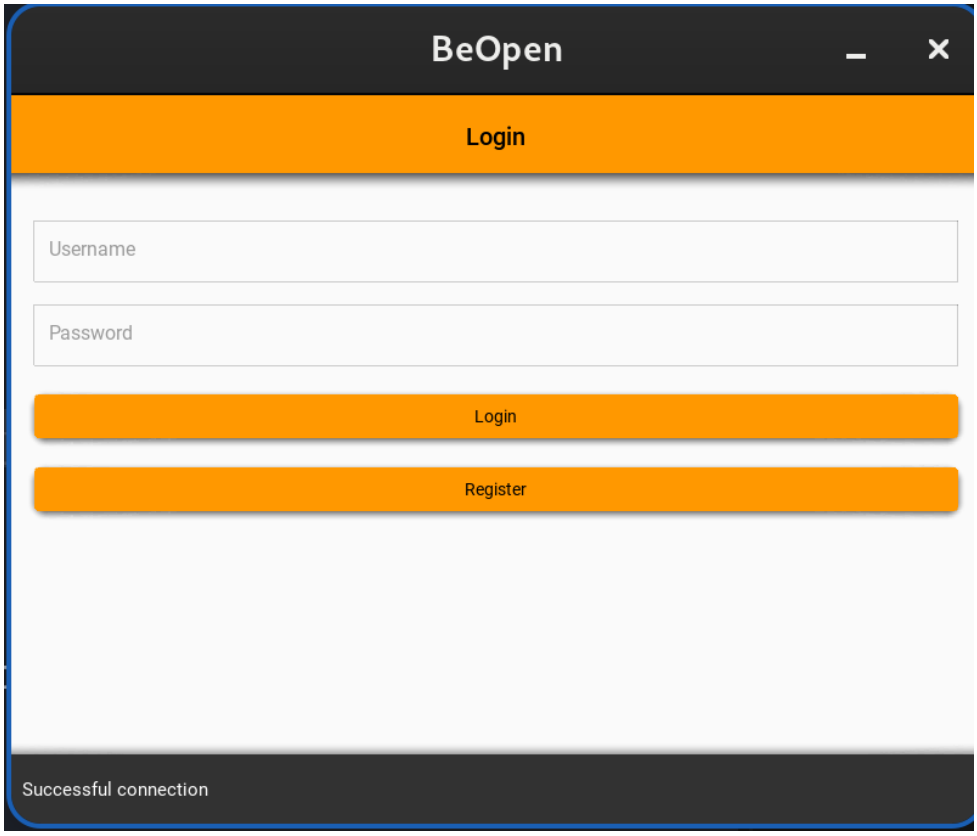
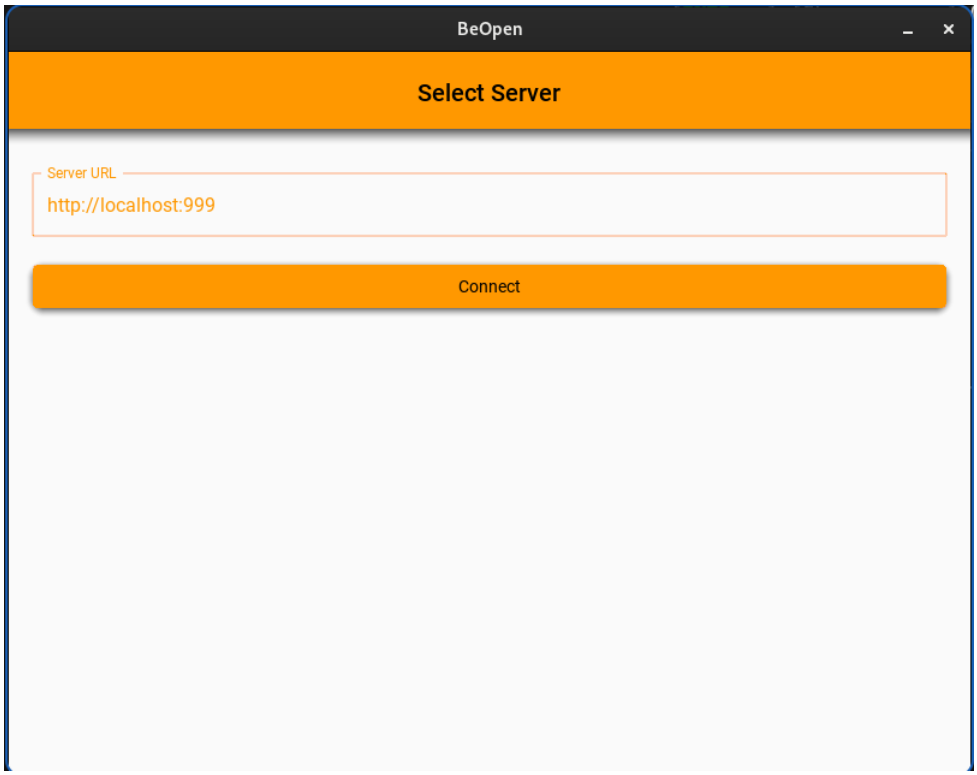
U.3A.II	1	
U.3B.I	1	
U.3B.I	2	

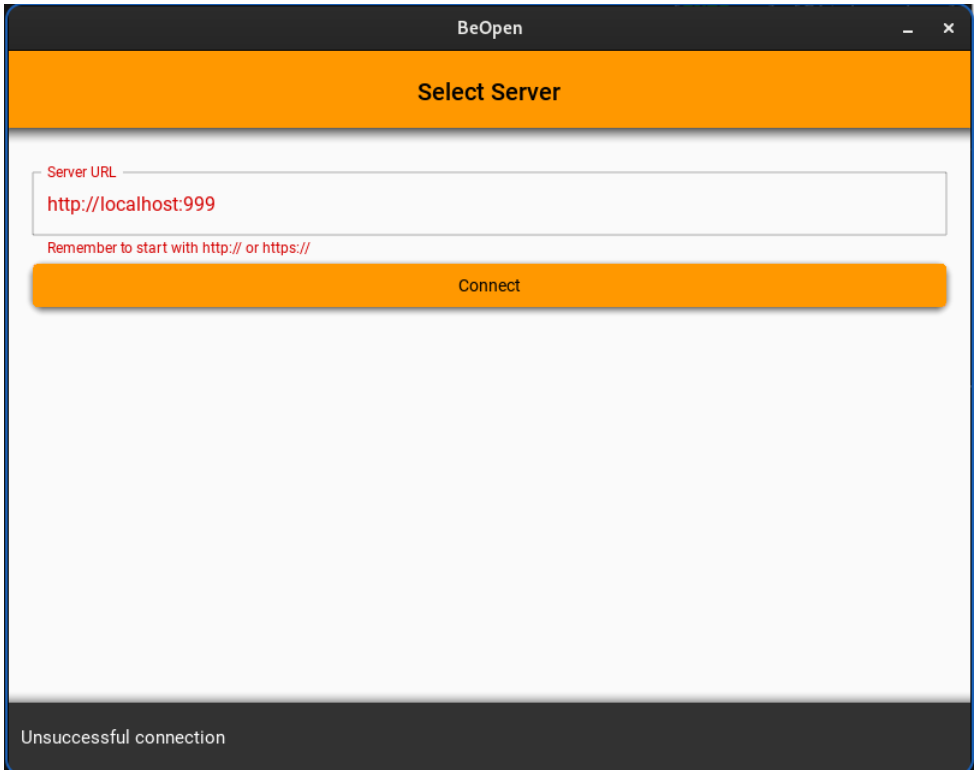
	user_id	username	password	level
	Filter	Filter	Filter	Filter
1	64296942-d821-479e-83d1-0630666d538c	user	58c18b8f5a19f3e5858a676b1bef3c6a615...	member
2	d5288762-784d-4539-b291-54e6e484c529	user2	2394be384be86f8f59dd98720c4e918a190...	member
3	b02889cb-30b7-48f5-8af5-f70c4e8c32e2	user3	7b54889e1945737b4e9562c6a59b3299c6...	member
4	636cce26-039c-4b89-a82f-b95d218d1784	user4	6830841305105558303 74bb2bc86128884...	member
5	a324789f-249a-4870-9bbf-9319e4cf237e	user5	fd91b02c74fb64397853aeef50efd2907f7b...	member
6	214fd34b-31d7-4d7c-bf80-d1d33d29b4ec	user6	804a7bd29e53b31da77ff254ebf7bead54c...	member
7	22ec0d99-b137-40c5-ae49-3e1d37a62820	user7	7250fbcc34fc7f286422c023a0432d9c6cba...	member
8	8e5cce75-6c8a-475b-b44d-f28bc8d268e1	ken	0418bf3d859c0be5412a32edc73a36aa7fb...	member
9	19c1d462-5bb2-4a90-a9b1-c4d75f661789	thisis25characterusername	cdf09752549a2618fc1e3b18274b5139594...	member
10	3127f1be-8de4-4bb8-8e21-5af16dd91111	thisis26characterusername	1339ba8de0f0e6eadc50b777654381ceef7...	member
11	6d7e5f72-071b-4bbb-80ae-bacfddbd78bc	admin	ff7c32ed473e299cc96a4794fef0227c6e29...	admin

	user_id	username	password	level
	Filter	Filter	Filter	Filter
1	64296942-d821-479e-83d1-0630666d538c	user	58c18b8f5a19f3e5858a676b1bef3c6a615...	member
2	d5288762-784d-4539-b291-54e6e484c529	user2	2394be384be86f8f59dd98720c4e918a190...	member
3	b02889cb-30b7-48f5-8af5-f70c4e8c32e2	user3	7b54889e1945737b4e9562c6a59b3299c6...	member
4	636cce26-039c-4b89-a82f-b95d218d1784	user4	6830841305105558303 74bb2bc86128884...	member
5	a324789f-249a-4870-9bbf-9319e4cf237e	user5	fd91b02c74fb64397853aeef50efd2907f7b...	member
6	214fd34b-31d7-4d7c-bf80-d1d33d29b4ec	user6	804a7bd29e53b31da77ff254ebf7bead54c...	member
7	22ec0d99-b137-40c5-ae49-3e1d37a62820	user7	7250fbcc34fc7f286422c023a0432d9c6cba...	member

U.4A.I	1	 BeOpen Select Server Server URL http://localhost:9999 Connect
U.4A.I	2	 BeOpen Select Server Server URL http://localhost:9999 Remember to start with http:// or https:// Connect

U.4A.II	1	
U.4A.IIII	1	

U.4A.IIII	2	
U.4B.I	1	

U.4B.I	2	 BeOpen Select Server Server URL http://localhost:999 Remember to start with http:// or https:// Connect Unsuccessful connection

Profile

Test Number	Test Description	Expected	Observed	Action
P.1A.I	Clicking the top account/profile button on the top bar of the app. This button is used to access the users own profile page where they can edit certain parts of their profile	Clicking the button changes the page to the profile page the title of the page should just be “Profile”	The page changes to the profile page but the title says “user’s Profile” (we are logged in as “user”) it should only display the username in the title if we are in another persons profile.	Setup an if statement in the load_content method of AccountPage to check if the username being passed is the same as the session.username (where the logged in username is stored)
P.1A.II	Clicking the top account/profile button on the top bar of the app. This button is used to access the users own profile page where they can edit certain parts of their profile	Clicking the button changes the page to the profile page the title of the page should just be “Profile”	As expected	
P.2A.I	Checking the contents of the profile page and that the ui only displays edit buttons on the correct categories. In this case we are logged in as just a **member**.	Clicking the button changes the page to the profile page the title of the page should just be “Profile” and edit buttons should be next to name, role, biography and occupation	As expected	

P.2B.I	Checking the contents of the profile page and that the ui only displays edit buttons on the correct categories. In this case we are logged in as just a **management**.	Clicking the button changes the page to the profile page the title of the page should just be “Profile” and edit buttons should be next to name, role, biography and occupation	As expected	
P.2C.I	Checking the contents of the profile page and that the ui only displays edit buttons on the correct categories. In this case we are logged in as just an **admin**.	Clicking the button changes the page to the profile page the title of the page should just be “Profile” and edit buttons should be next to name, role, biography and occupation	As expected	
P.2D.I	Checking the contents of the profile page and that the ui only displays edit buttons on the correct categories. In this case we are logged in as just an **team leader**.	Clicking the button changes the page to the profile page the title of the page should just be “Profile” and edit buttons should be next to name, role, biography, team name and occupation		
P.3A.I	Clicking the edit button on the **name** column	When clicking the edit button where the	As expected	

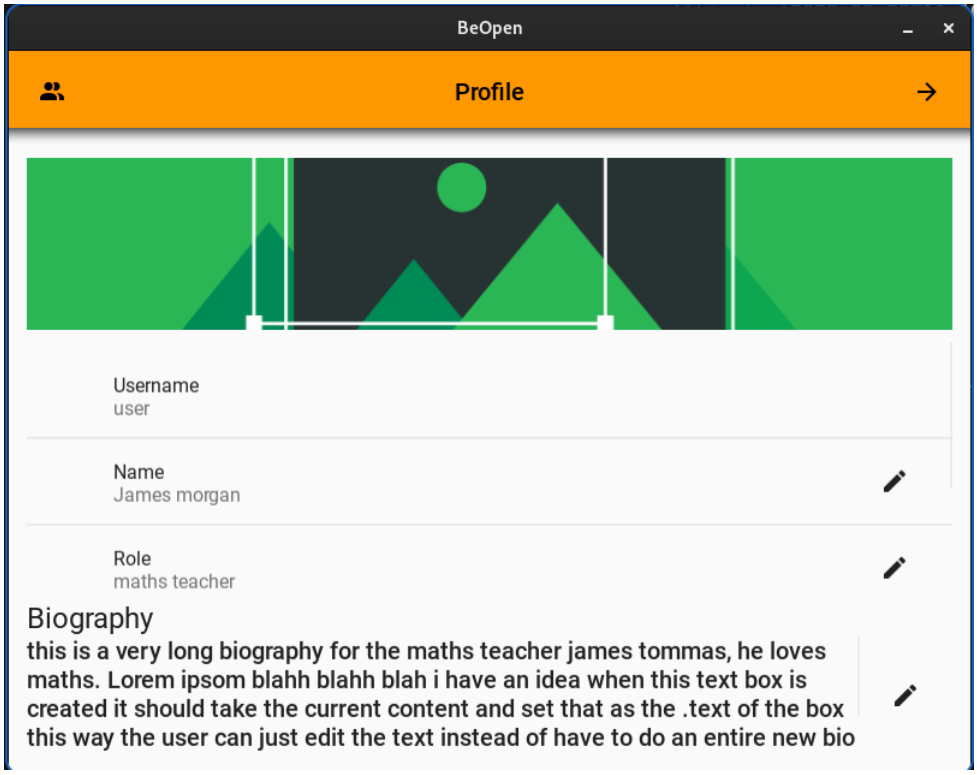
	and changing it	profile picture should be displayed will be replaced with an edit box. On submit the UI should update to reflect the change and the server side database should do the same. The UI should also replace the edit box back with the profile picture		
P.3B.I	Clicking the edit button on the **role** column and changing it	When clicking the edit button where the profile picture should be displayed will be replaced with an edit box. On submit the UI should update to reflect the change and the server side database should do the same. The UI should also replace the edit	As expected	

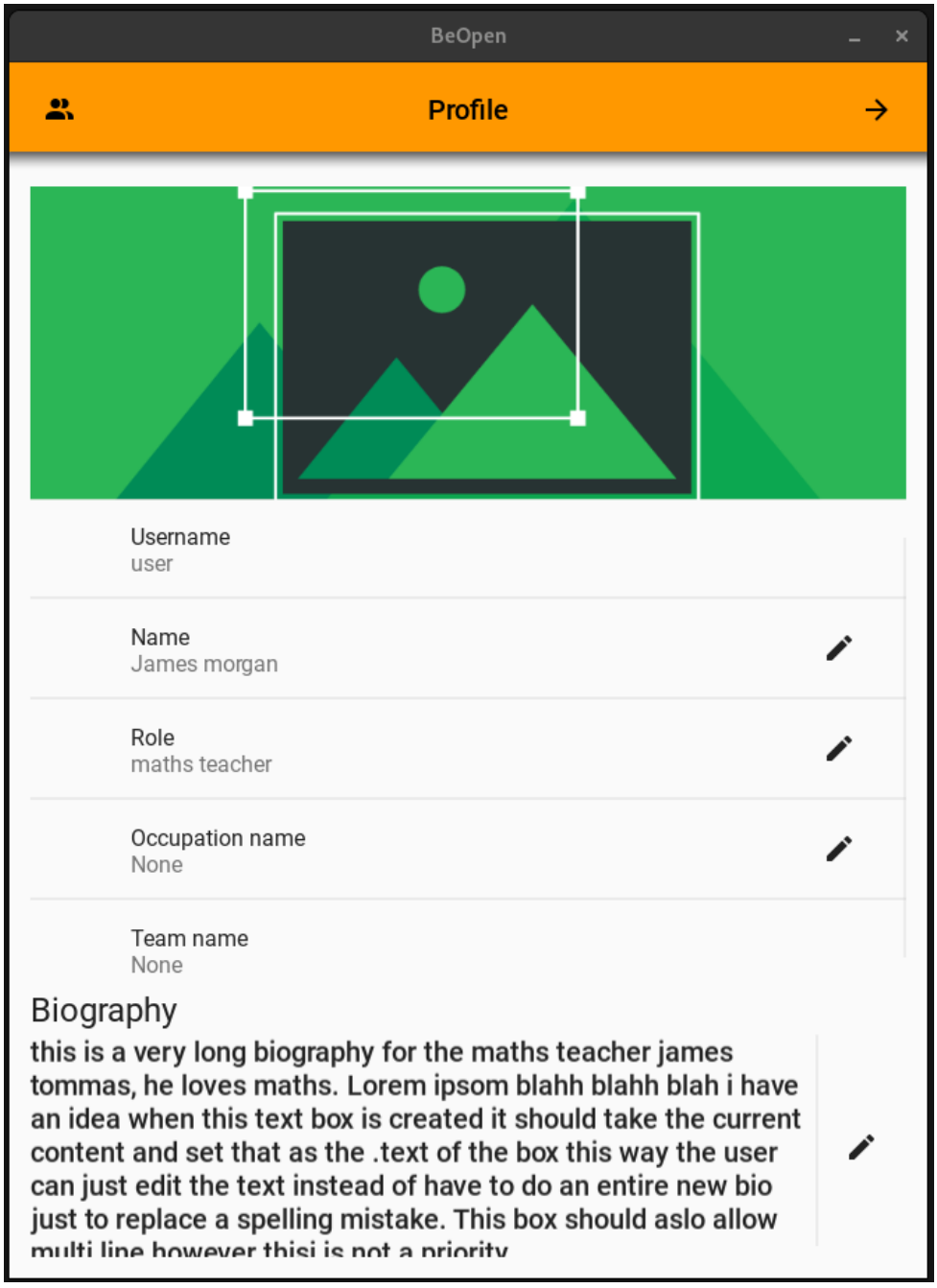
		box back with the profile picture		
P.3C.I	Clicking the edit button on the **biography** column and changing it	When clicking the edit button where the profile picture should be displayed will be replaced with an edit box. On submit the UI should update to reflect the change and the server side database should do the same. The UI should also replace the edit box back with the profile picture	As expected	
P.3D.I	Clicking the edit button on the **occupation name** column and creating an occupation change request. The user is a member in this case	After clicking the edit button the profile picture should be replaced with an occupation request creation area (in this case since the user is a member). The occupation change request	On clicking the occupation button an empty list would appear	This was a problem with how the list was implemented. There was another issue that would prevent proper selection from the list as well

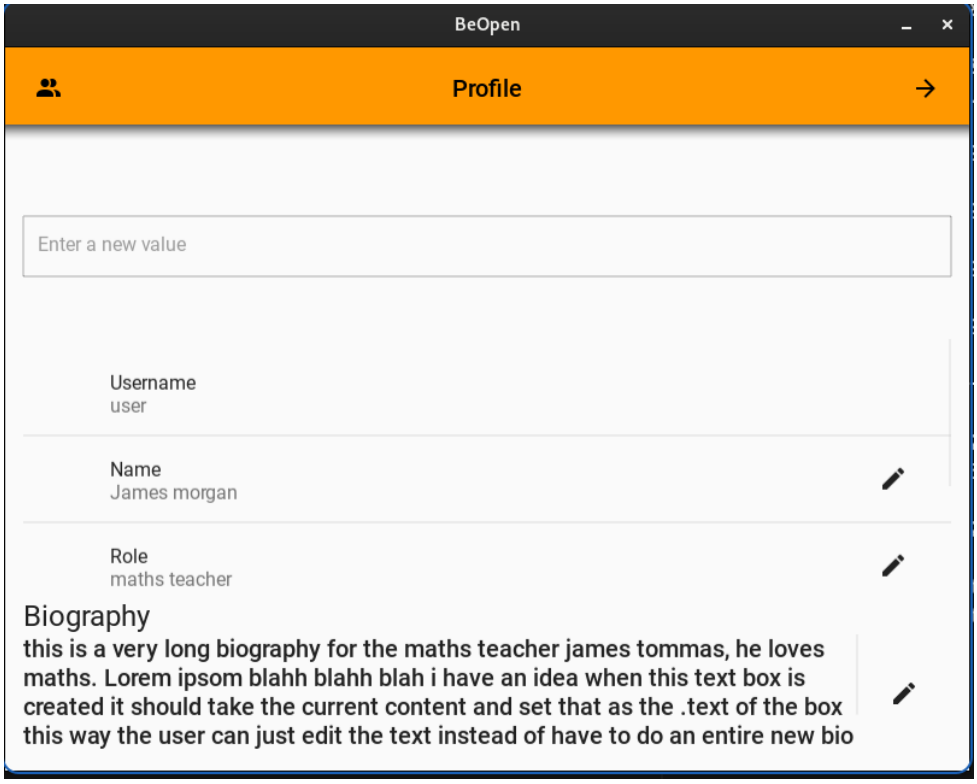
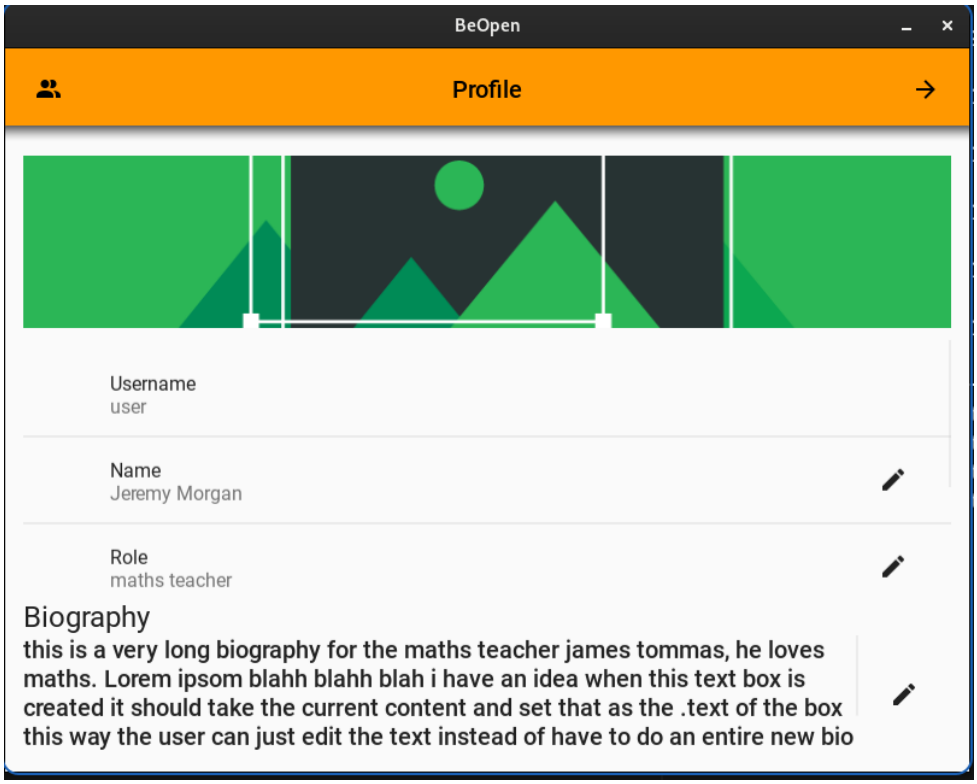
		should appear on the server side database as well.		
P.3D.II	Clicking the edit button on the **occupation name** column and creating an occupation change request. The user is a member in this case	After clicking the edit button the profile picture should be replaced with an occupation request creation area (in this case since the user is a member). The occupation change request should appear on the server side database as well.	On selection of the first option in the list and clicking “create request” an error message would appear stating “no selection made”. However picking any other option works fine	This came down to the check for a selection being made. Since the selection was the 0^{th} item the check would return false since in python “if 0” is false.
P.3D.III	Clicking the edit button on the **occupation name** column and creating an occupation change request. The user is a member in this case	After clicking the edit button the profile picture should be replaced with an occupation request creation area (in this case since the user is a member). The occupation change request should appear on the server side database as well.	Didn’t clean the text from the create request area. It also didn’t set the editing area back to the profile picture. There is also no way to close the area if your done.	

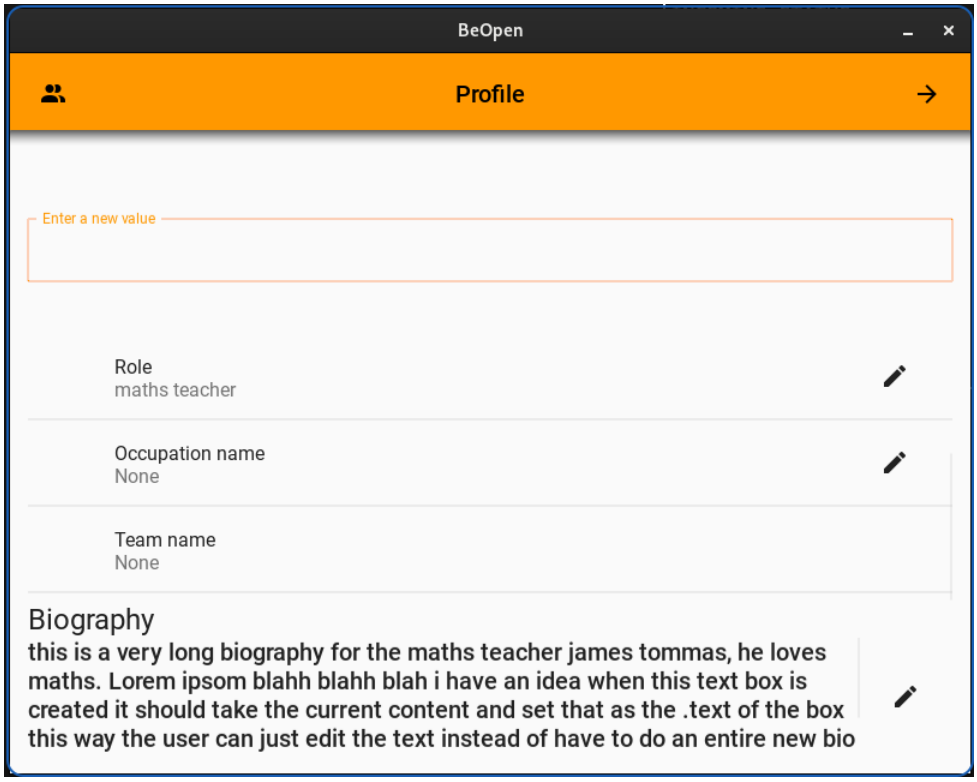
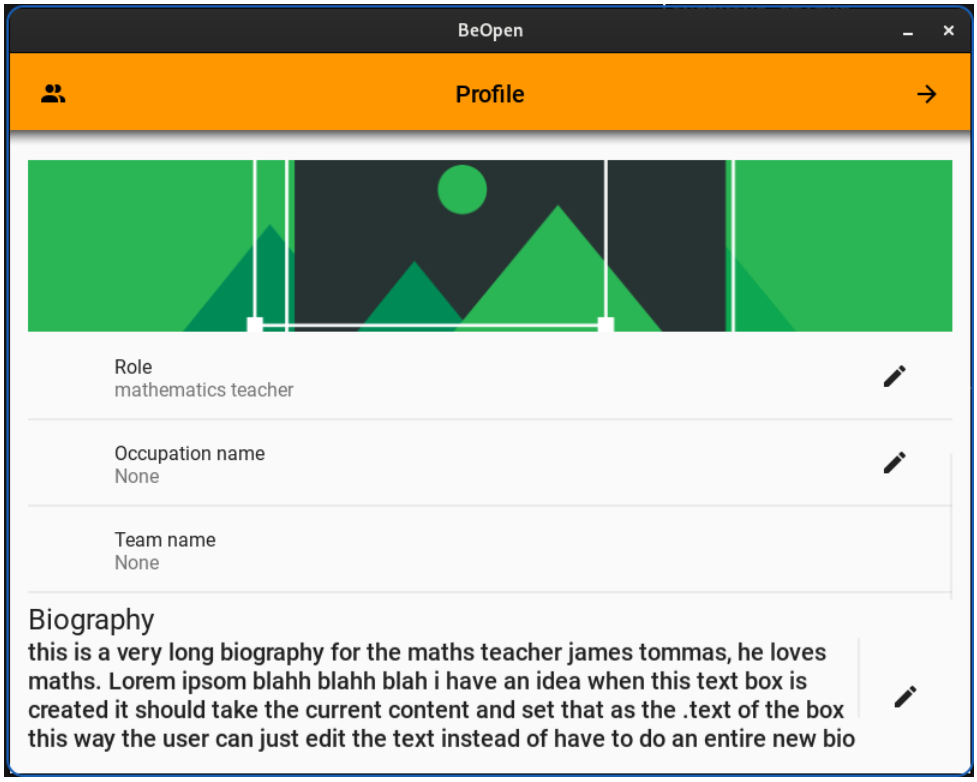
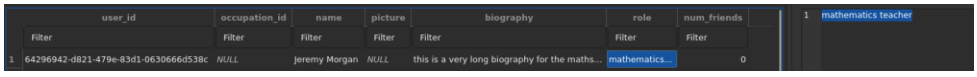
P.3D.IIII	Clicking the edit button on the **occupation name** column and creating an occupation change request. The user is a member in this case	After clicking the edit button the profile picture should be replaced with an occupation request creation area (in this case since the user is a member). The occupation change request should appear on the server side database as well.	As expected	
P.3E.I	Clicking the cancel button on a pending request	This should change the request status to Approved, this change should be reflected in the database	As expected	

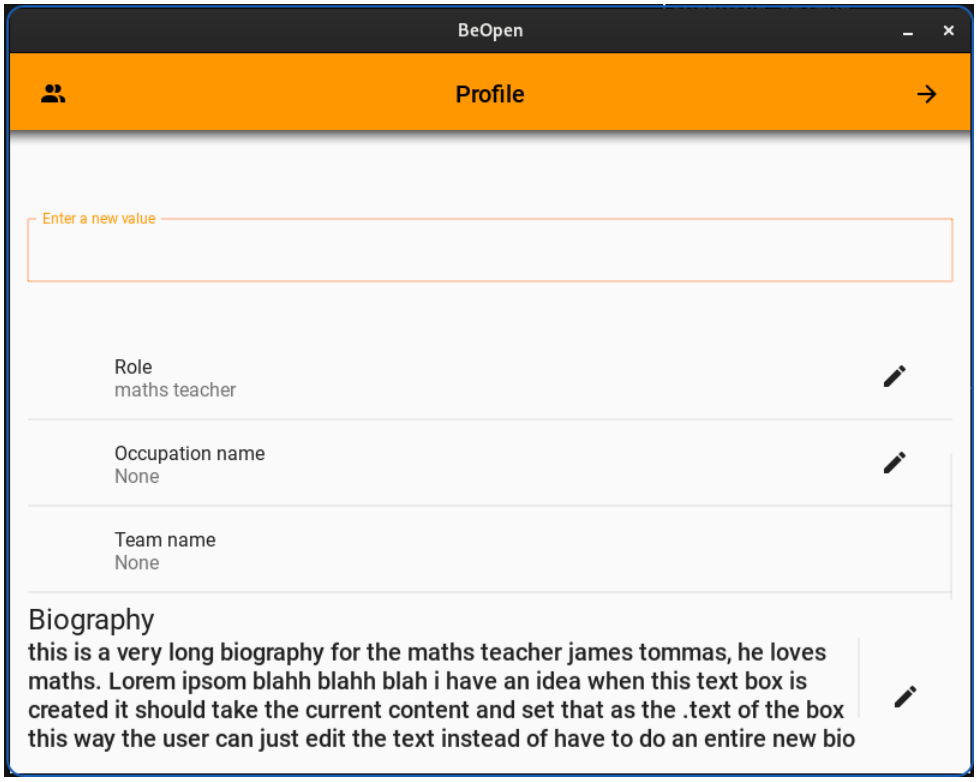
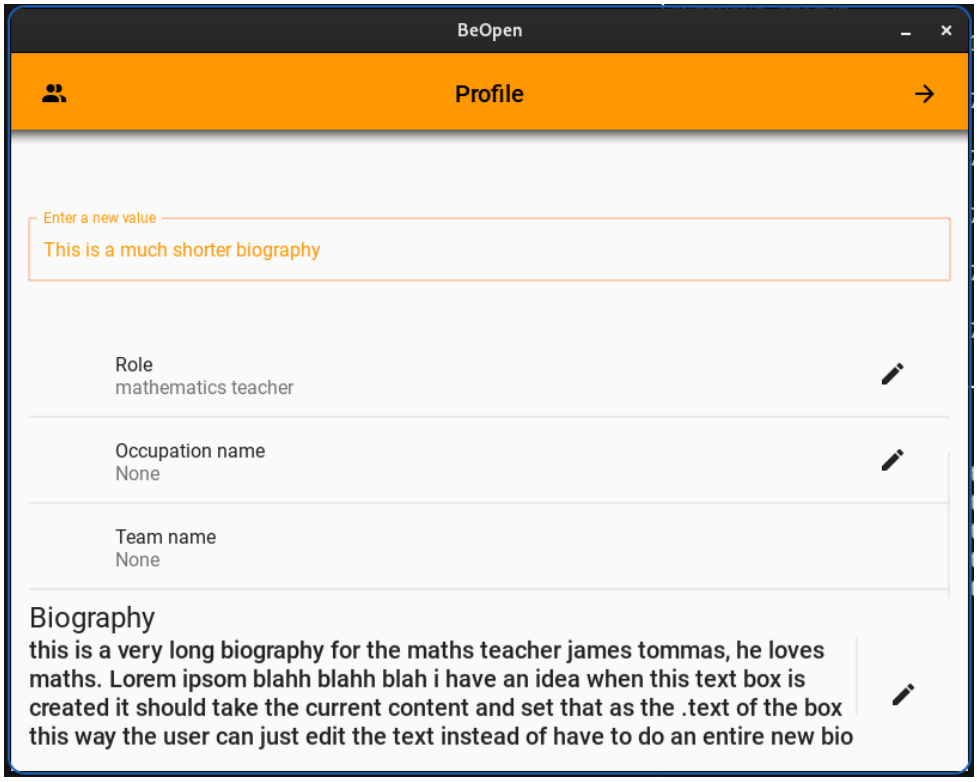
Test Number	Image Number	Image

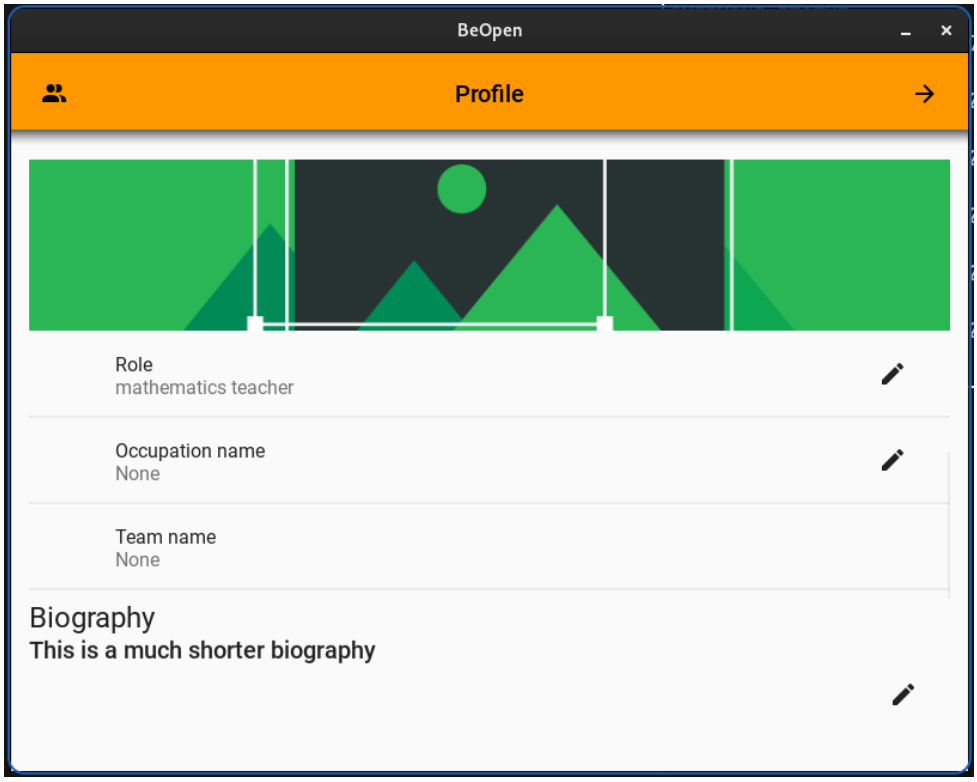
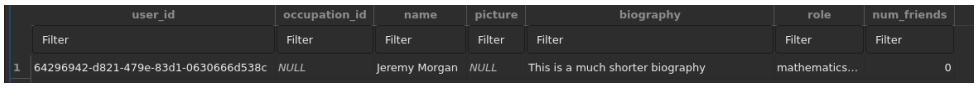
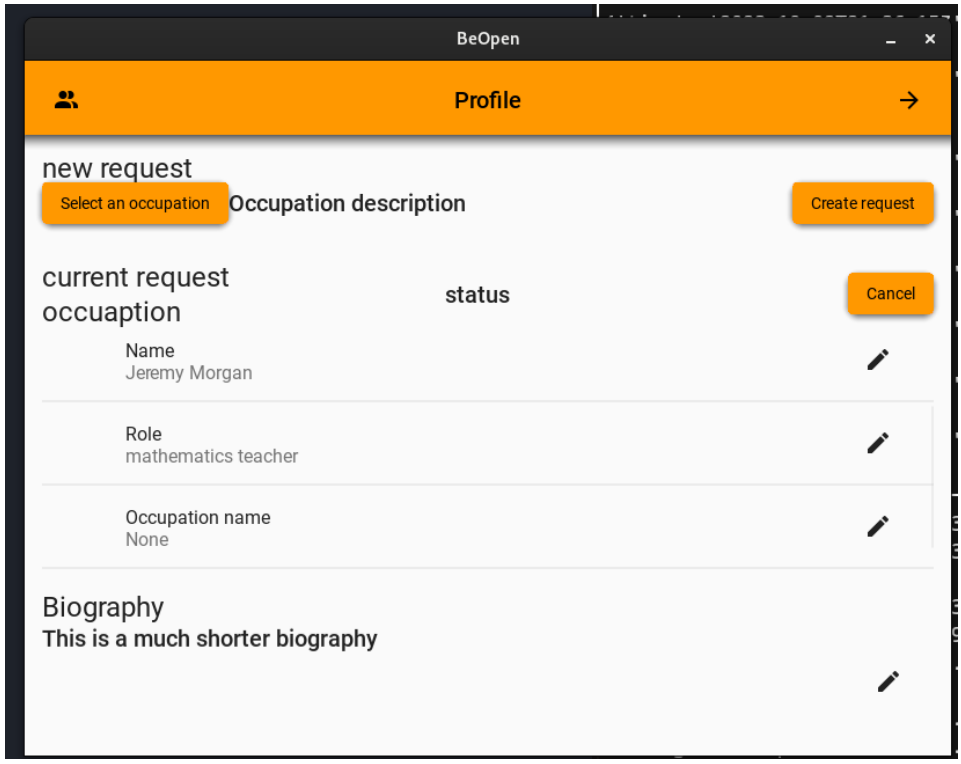
P.1A.I	I	 BeOpen Profile Username user Name James morgan Role maths teacher Biography this is a very long biography for the maths teacher james tommas, he loves maths. Lorem ipsom blahh blahh blah i have an idea when this text box is created it should take the current content and set that as the .text of the box this way the user can just edit the text instead of have to do an entire new bio

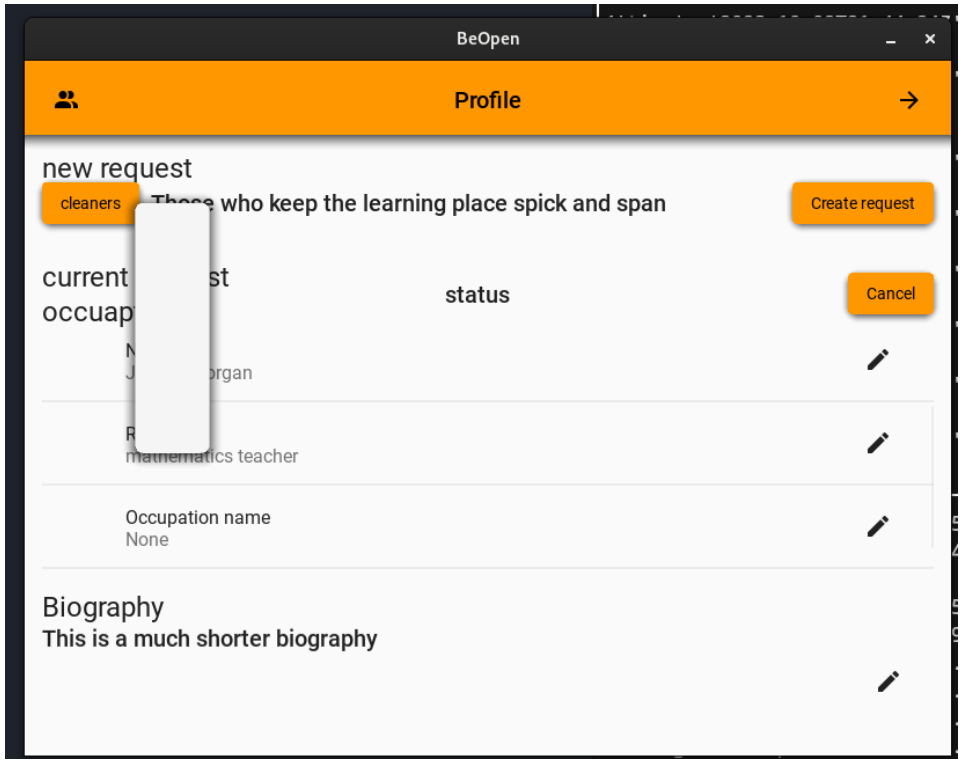
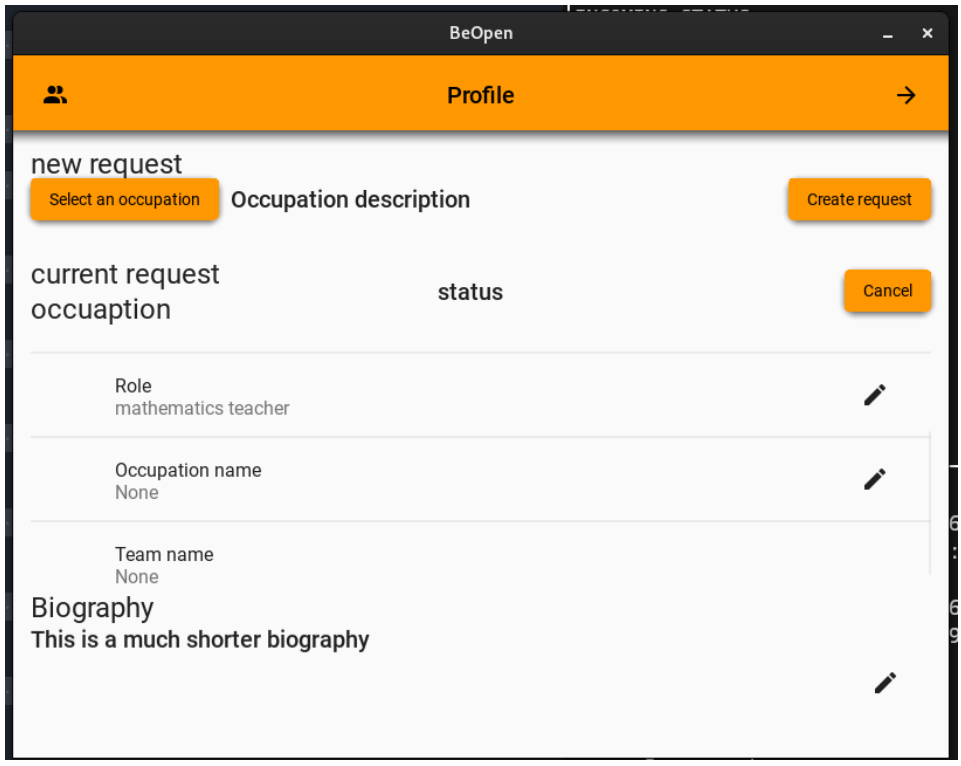
P.2A.I	1	 BeOpen Profile Username user Name James morgan Role maths teacher Occupation name None Team name None Biography this is a very long biography for the maths teacher james tommas, he loves maths. Lorem ipsom blahh blahh blah i have an idea when this text box is created it should take the current content and set that as the .text of the box this way the user can just edit the text instead of have to do an entire new bio just to replace a spelling mistake. This box should aslo allow multi line however thisi is not a priority

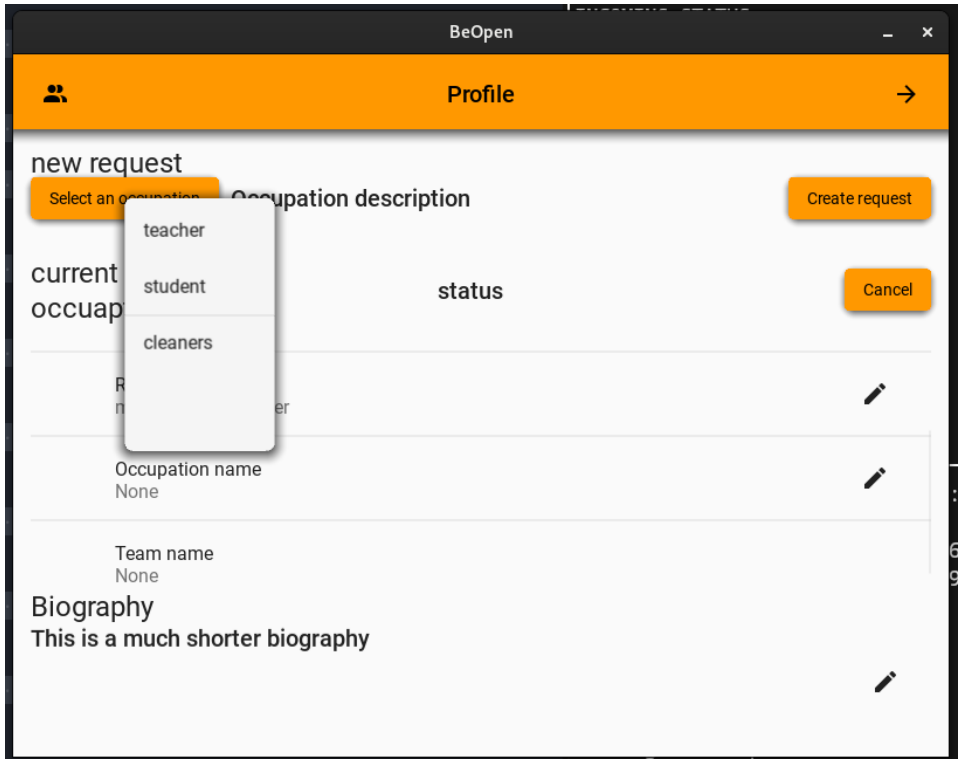
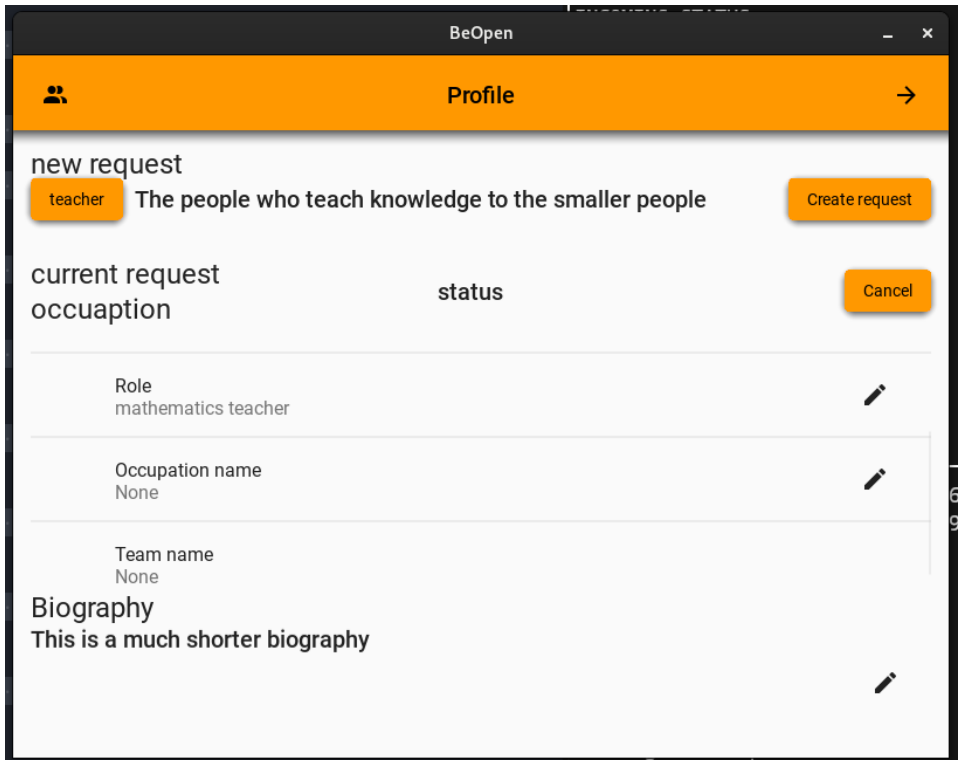
P.3A.I	1	
P.3A.I	2	
P.3A.I	3	

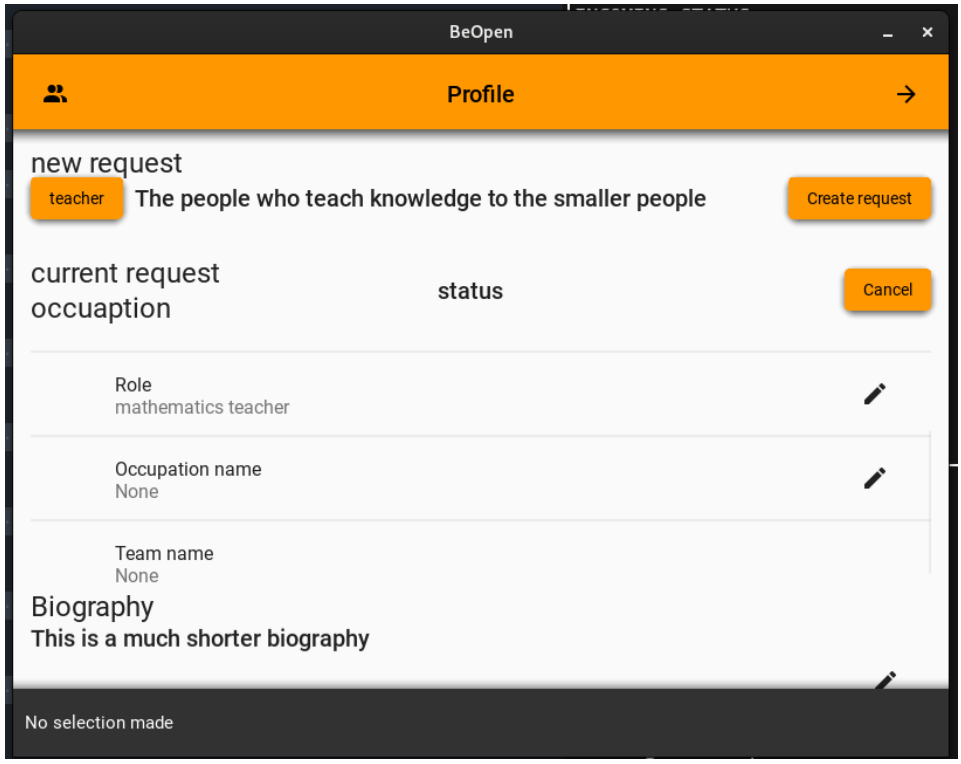
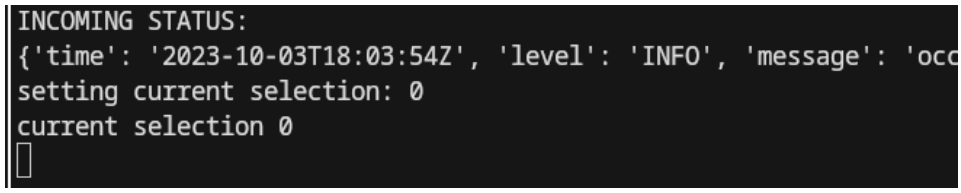
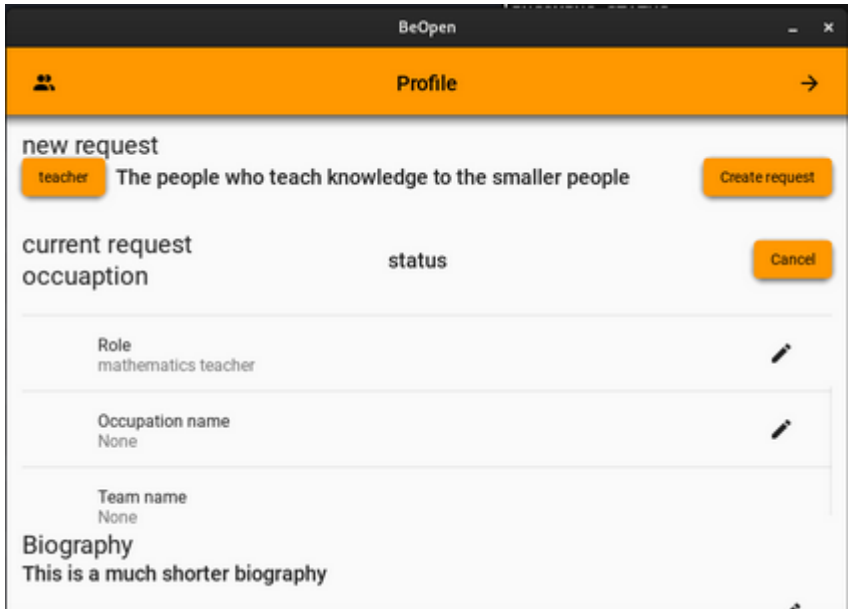
P.3B.I	1	
P.3B.I	2	
P.3B.I	3	

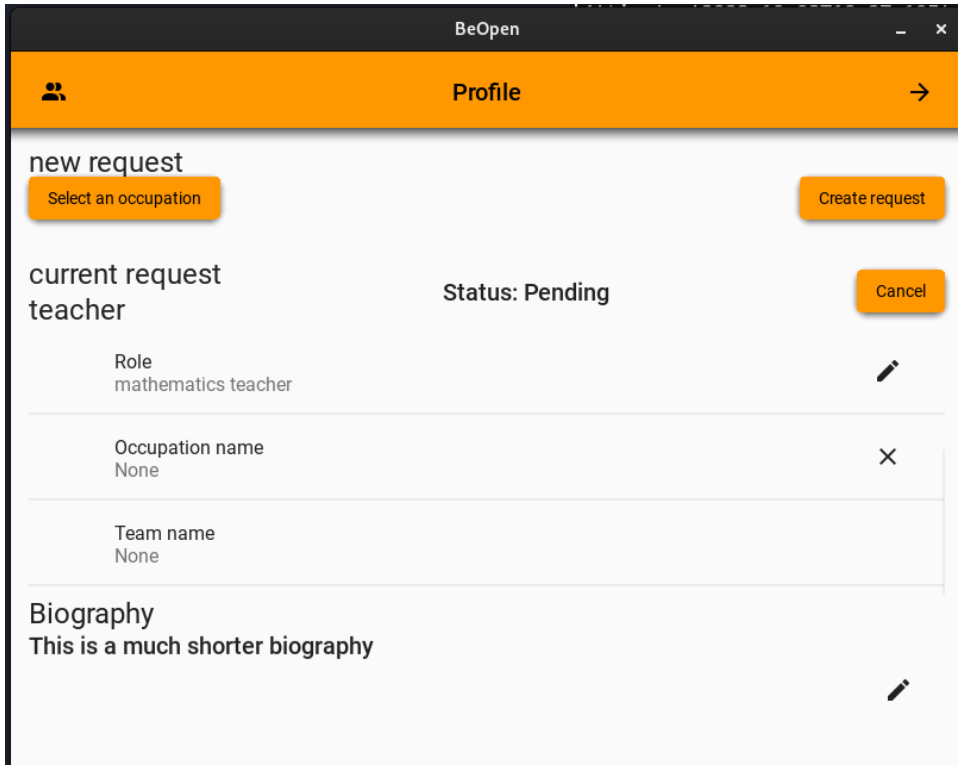
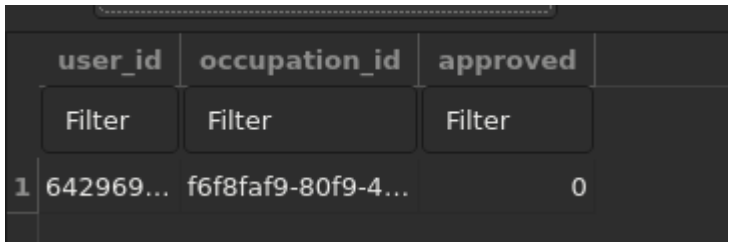
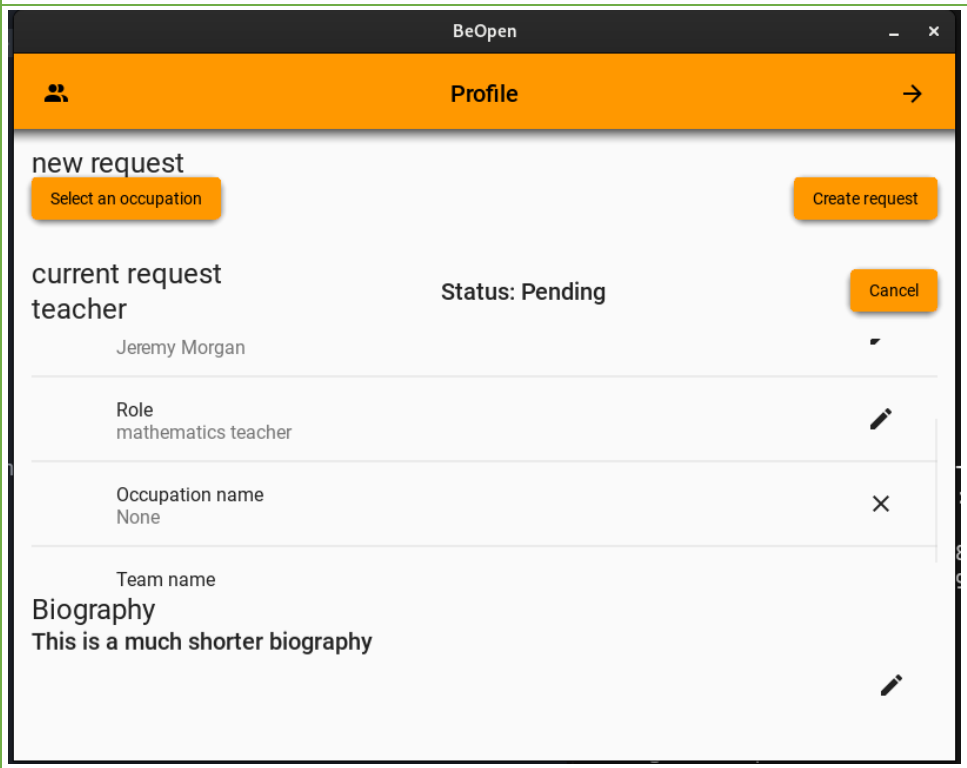
P.3C.I	1	 BeOpen Profile Enter a new value Role maths teacher Occupation name None Team name None Biography this is a very long biography for the maths teacher james tommas, he loves maths. Lorem ipsom blahh blahh blah i have an idea when this text box is created it should take the current content and set that as the .text of the box this way the user can just edit the text instead of have to do an entire new bio
P.3C.I	2	 BeOpen Profile Enter a new value This is a much shorter biography Role mathematics teacher Occupation name None Team name None Biography this is a very long biography for the maths teacher james tommas, he loves maths. Lorem ipsom blahh blahh blah i have an idea when this text box is created it should take the current content and set that as the .text of the box this way the user can just edit the text instead of have to do an entire new bio

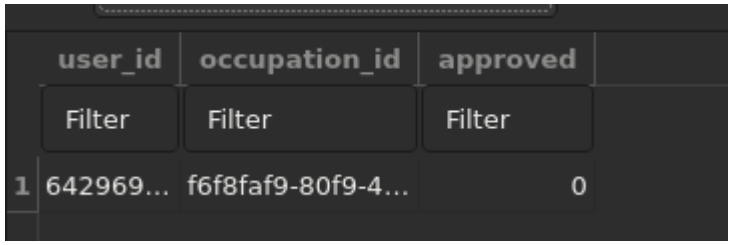
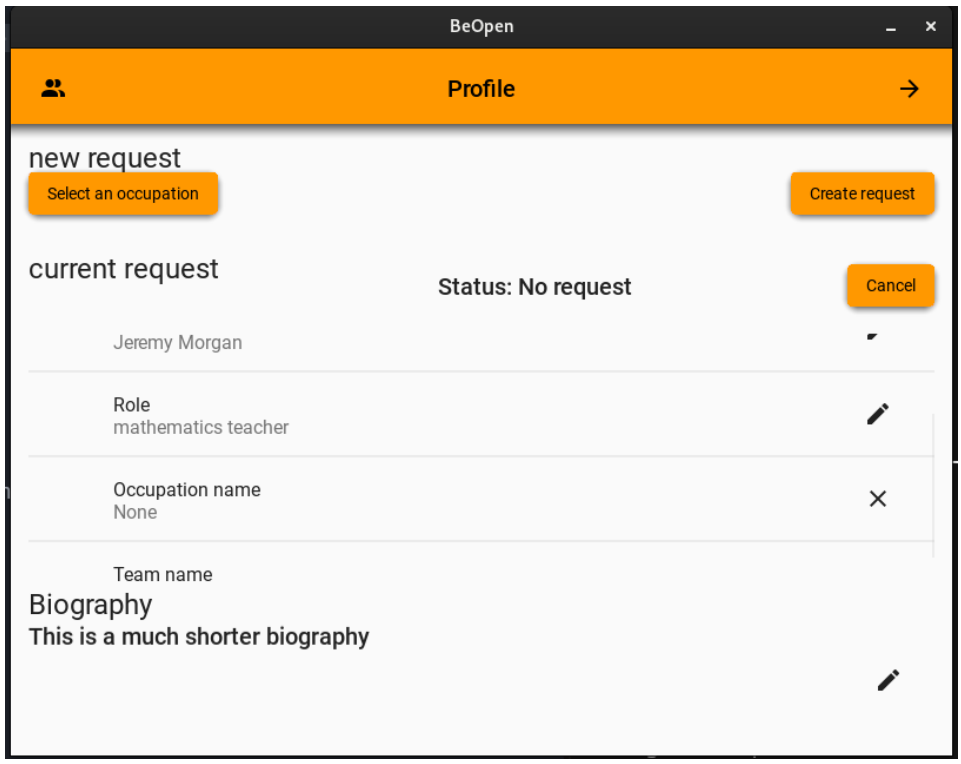
P.3C.I	3	
P.3C.I	4	
P.3D.I	1	

P.3D.I	2	
P.3D.II	1	

P.3D.II	2	
P.3D.II	3	

P.3D.II	4	
P.3D.II	5	
P.3D.III	1	

P.3D.IIII	1	
P.3D.IIII	2	
P.3E.I	1	

P.3E.I	2	
P.3E.I	3	
P.3E.I	4	

Friends

Test Number	Test Description	Expected	Observed	Action
F.1A.I	To see if the basic UI is displayed correctly	There should be a top bar with a back button. Then in the content area of the page there should be a button with the text “requests” and a list of friends. The user being tested here has no friends. The screen should also be scrollable.	The top button is being cut off as the scroll view seems to be clipping into the top bar	I removed the boxlayout that was surrounding the scrollview. And added the padding to the scrollview itself instead of the above boxlayout.
F.1A.II	To see if the basic UI is displayed correctly	There should be a top bar with a back button. Then in the content area of the page there should be a button with the text “requests” and a list of friends. The user being tested here has 3 friends. The screen should also be scrollable.	As expected	
F.2A.I	Removing a friend using the cross button	On release of the x button the server	As expected	

	displayed next to their username	database should be updated to remove this friend and the client UI should also update to show the friend has been removed		
F.3A.I	Pressing the back button on the friends page	This should bring you back to the account screen	As expected	
F.4A.I	Pressing the “requests” button	This should bring you to the friend requests screen.	As expected	
F.4B.I	General UI of the friend requests screen	There should be a top bar with a back button and 2 scrollable areas one outgoing friend requests each requests and the other incoming friend requests. The user being used here has 0 incoming and 0 outgoing requests	There is uneeded padding around the whole screen. There is no space between the username text bar and the add friend button. These items also cant be interacted with at all	Remove the padding and included the load_content method in the init (this is likely what was blocking the items from being intractable)
F.4B.II	General UI of the friend requests screen	There should be a top bar with a back button and 2 scrollable areas one outgoing	When the request button is pressed the program freezes and the needs to be	This is likely to do with the request functionality potentially the server is not

		friend requests each requests and the other incoming friend requests. The user being used here has 0 incoming and 0 outgoing requests	forced closed	returning the correct callback code. Turns out the client was calling the “friend_get_requests” event instead of “friend_get_request”
F.4B.III	General UI of the friend requests screen	There should be a top bar with a back button and 2 scrollable areas one outgoing friend requests each requests and the other incoming friend requests. The user being used here has 0 incoming and 0 outgoing requests	The server creates an error saying there is no “get_request” method in friend	In the friend class the method is called get_requests then in the handler it was called get_request. This was changed in the handler to conform with the info.py
F.4B.III	General UI of the friend requests screen	There should be a top bar with a back button and 2 scrollable areas one outgoing friend requests each requests and the other incoming friend requests. The user being used here has	The server complains that there is no such thing as an “accepted” column in the friends table	The column in the table is called “approved” so changed the SQL command to use this instead

		0 incoming and 0 outgoing requests		
F.4B.IV	General UI of the friend requests screen	There should be a top bar with a back button and 2 scrollable areas one outgoing friend requests each requests and the other incoming friend requests. The user being used here has 0 incoming and 0 outgoing requests	Complains that nonetype object is not iterable on the client. This suggests the server is returning None on incoming and or outgoing requets	This is because I didn't account for if the return on either requests was None (for no requests). So now if either one is None they are respectively set to []. Additionally the for loops were split up where they added the incoming and outgoing widgets
F.4B.V	General UI of the friend requests screen	There should be a top bar with a back button and 2 scrollable areas one outgoing friend requests each requests and the other incoming friend requests. The user being used here has 0 incoming and 0 outgoing requests	The lists are clipping into the text field and button	Seperated the page into the making a friend request area and 2 individually scrollable lists. This solves any problems with clipping. Additionally centred the title and fixed the direction of the back button
F.4B.VI	General UI of the friend	There should be a top bar with a back	As expected	

	requests screen	button and 2 scrollable areas one outgoing friend requests each requests and the other incoming friend requests. The user being used here has 0 incoming and 0 outgoing requests		
F.5A.I	Incoming and outgoing requests areas of the page The user being used here has 0 incoming and 0 outgoing requests.	So each area should instead show a single item reading that there are no requests	As expected	
F.5B.I	Incoming and outgoing requests areas of the page The user being used here has 2 incoming and 3 outgoing requests.	The incoming request area should display 2 incoming friend request each one should have the username, an accept button and a reject button The outgoing request area should show 3 outgoing request along with each ones username and	The incoming items go off the side of the screen and the tick icon is not displaying correctly	The icon name is wrong it should be called “check” not “tick”. The check icon is also being moved to the left

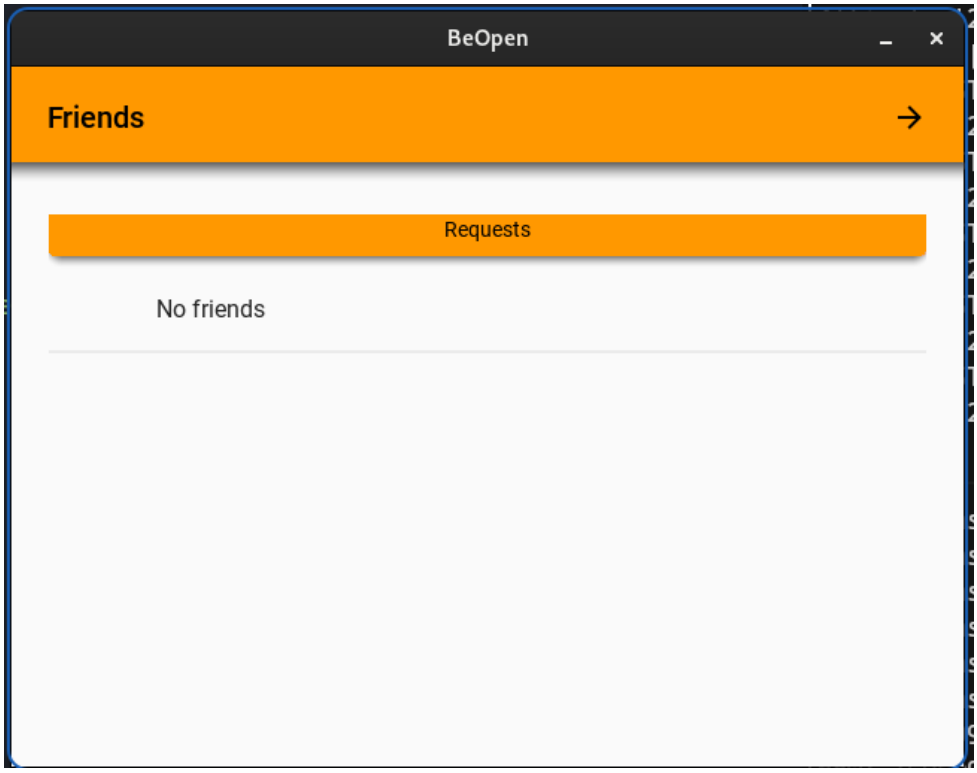
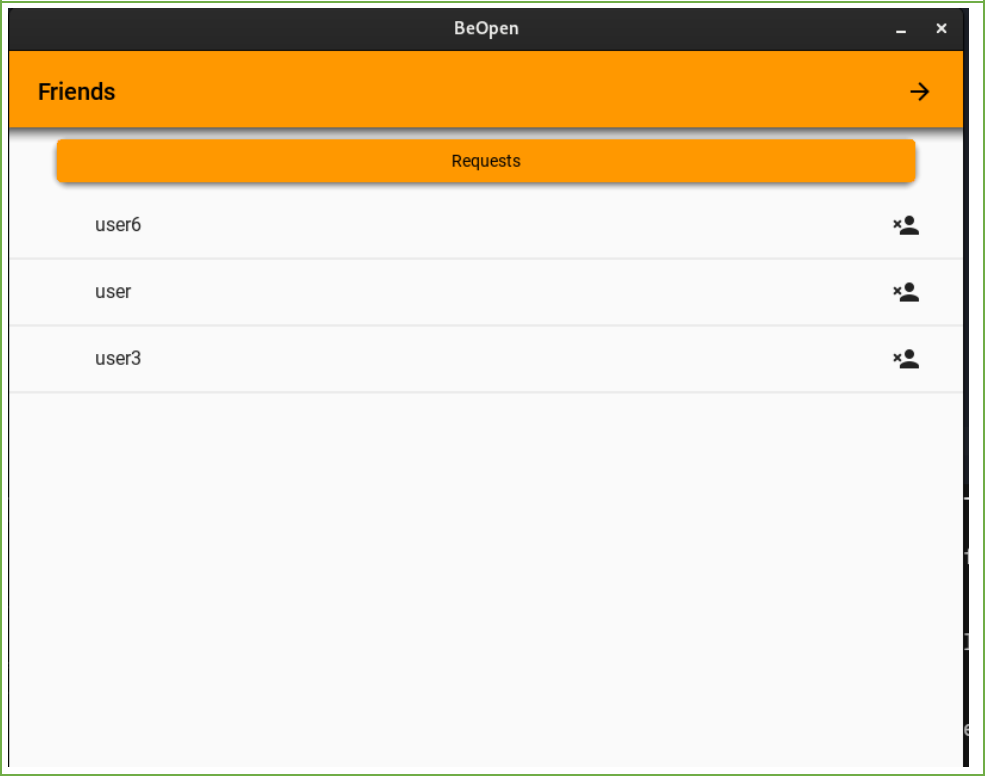
		a cancel button next to each.		
F.6A.I	Clicking the request itself (not its buttons) to get to the users profile who is making the request/is being requested	Clicking on a users request should switch screen to the account page and clicking the back button should then take you back to the friend request page.	It changed to the page but the page also displays the friends icon this should only be accessble to the user themselves or an admin	Validation on the creation of the friends icon and page
F.6A.II	Clicking the request itself (not its buttons) to get to the users profile who is making the request/is being requested	Clicking on a users request should switch screen to the account page and clicking the back button should then take you back to the friend request page.	As expected	
F.7A.I	Clicking the accept button on an incoming friend request	This should remove the request from the list, this person should then be added to the friends list on the “friends” page. And this person should be listed as an accepted friend in the server database	The server errors claiming that “friend_id” is never defined.	This is likely because this is meant to read self.friend_id

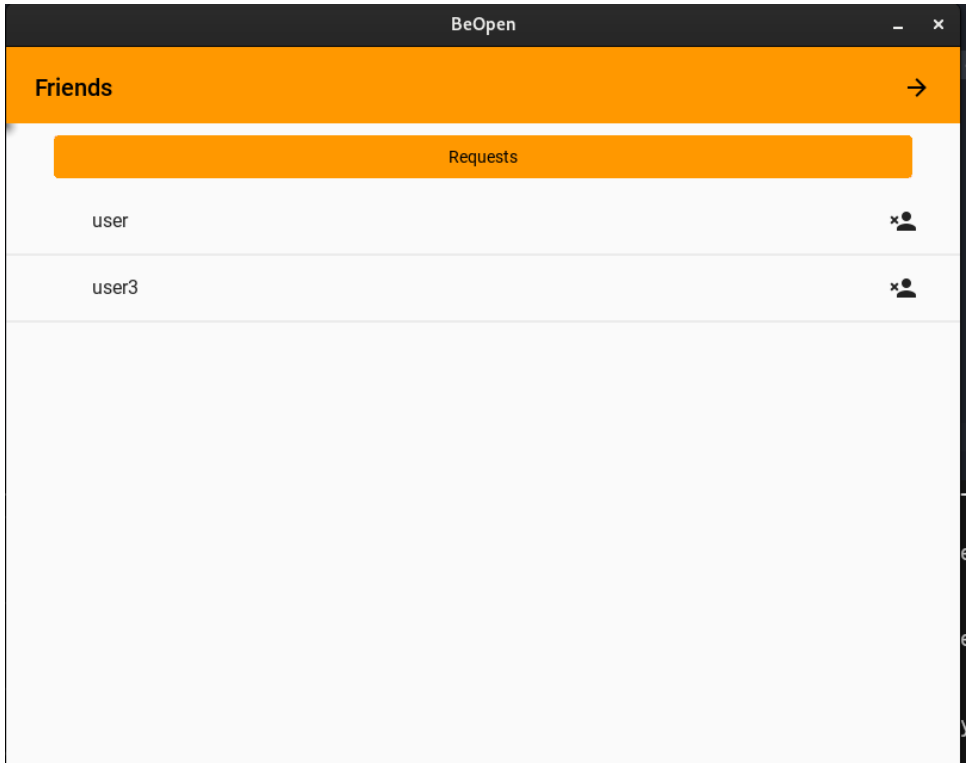
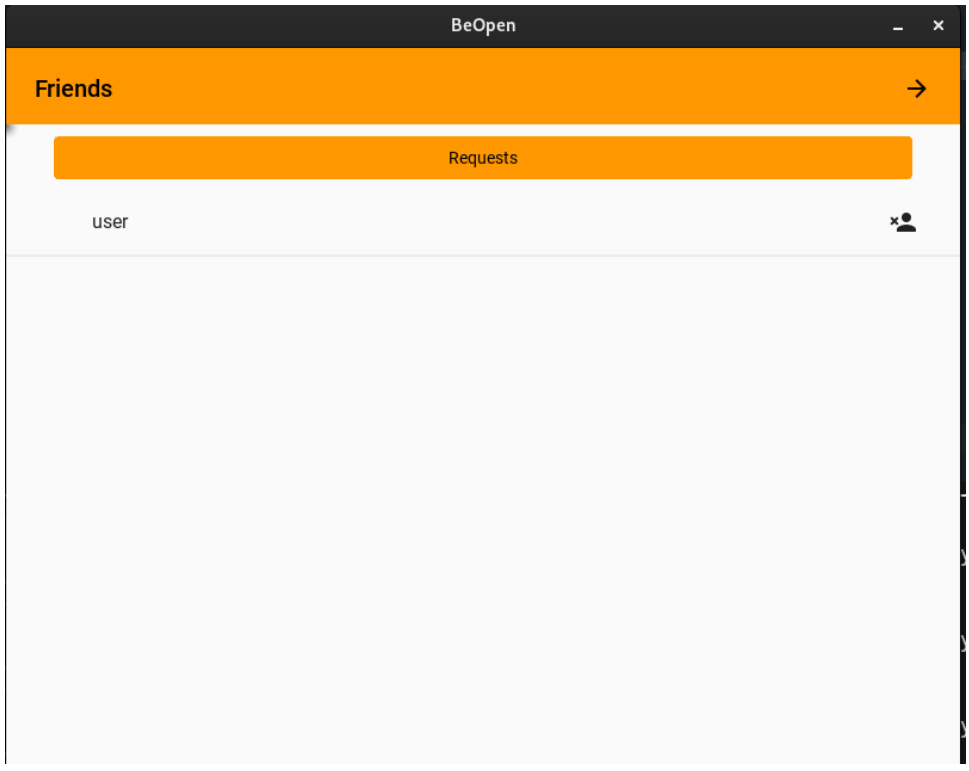
F.7A.II	Clicking the accept button on an incoming friend request	This should remove the request from the list, this person should then be added to the friends list on the “friends” page. And this person should be listed as an accepted friend in the server database	The object has no attribute “friend_id”.	This is a deeper logical problem. There is no friend_id attribute in the class ever created so I created a new property for it which is set when a friend username is assigned.
F.7A.II	Clicking the accept button on an incoming friend request	This should remove the request from the list, this person should then be added to the friends list on the “friends” page. And this person should be listed as an accepted friend in the server database	As expected	
F.7B.I	Clicking the reject button on an incoming friend request	This should remove the request from the list. And remove the request from the server side database.	No changes were made correctly. This is because the button was calling self.reject instead of root.reject	Changes the self.reject to root.reject method

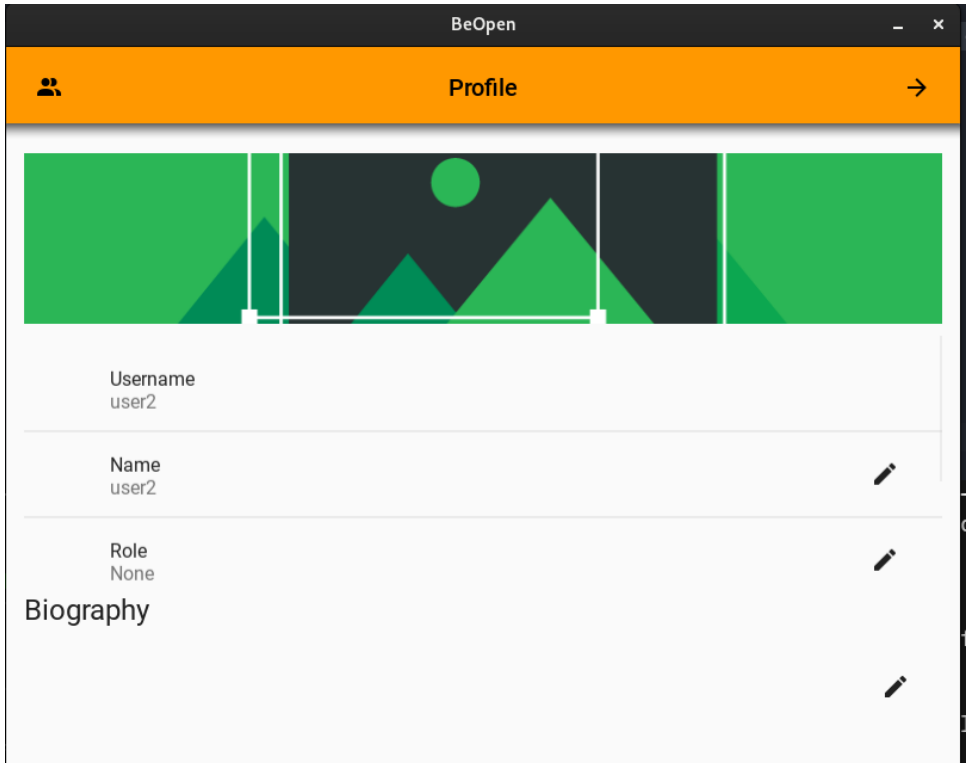
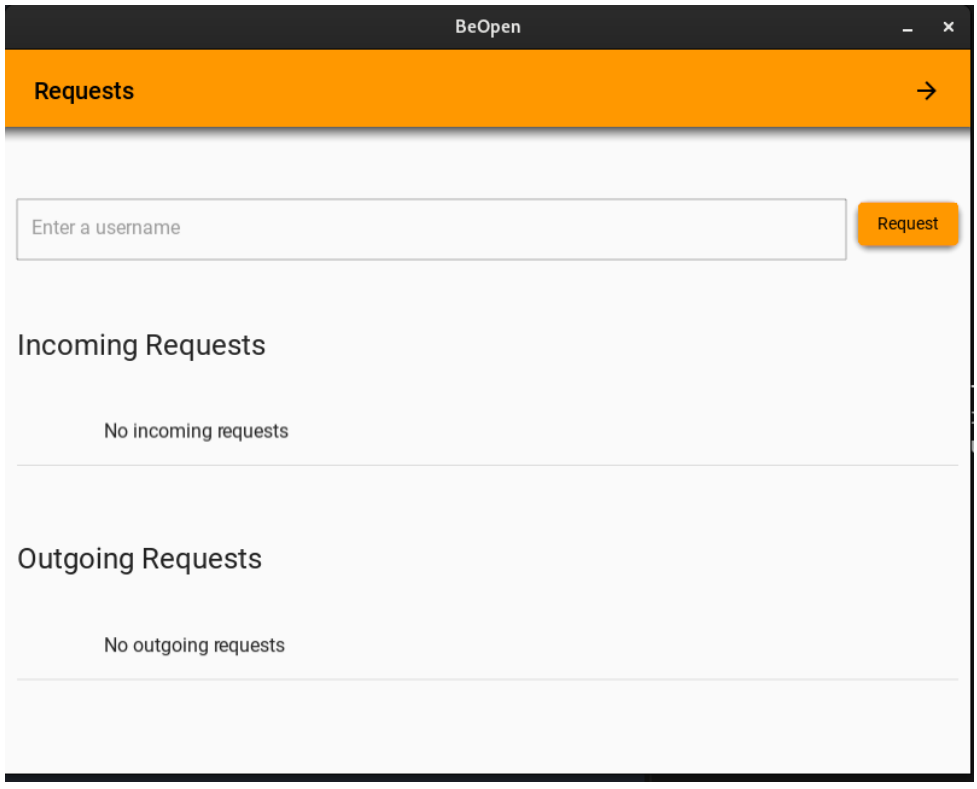
F.7B.II	Clicking the reject button on an incoming friend request	This should remove the request from the list. And remove the request from the server side database.	On clicking reject the UI refreshes correctly and doesn't freeze but even after refreshing the requests persists It seams the request isnt being correcly removed from the table as shown in the pictures	This is a server side issue and likely a permissions issue. It came down to a syntax error in the SQL command.
F.7B.III	Clicking the reject button on an incoming friend request	This should remove the request from the list. And remove the request from the server side database.	As expected	
F.7C.I	Clicking the cancel button on an outgoing friend request.	This should remove the request from the list and remove the request from the server side database	On clicking cancel the UI does refresh but there is no change to the list.	The cancel method wasn't sending any data about which request to cancel along with it. Hence the server did nothing
F.7C.II	Clicking the cancel button on an outgoing friend request.	This should remove the request from the list and remove the request from the server side	As expected	

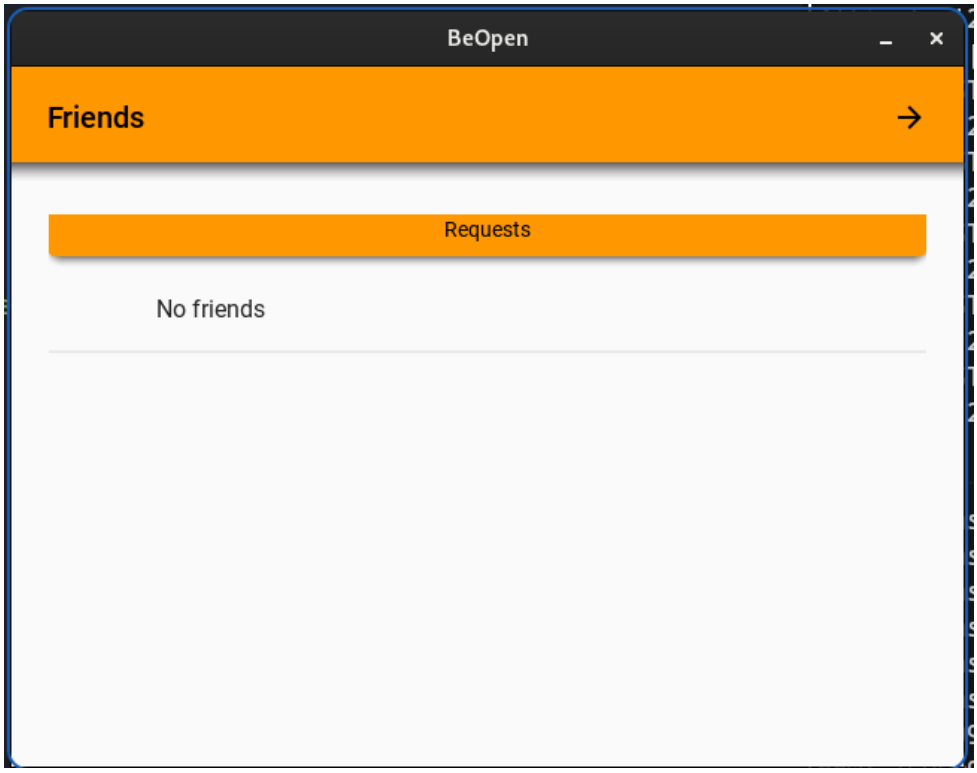
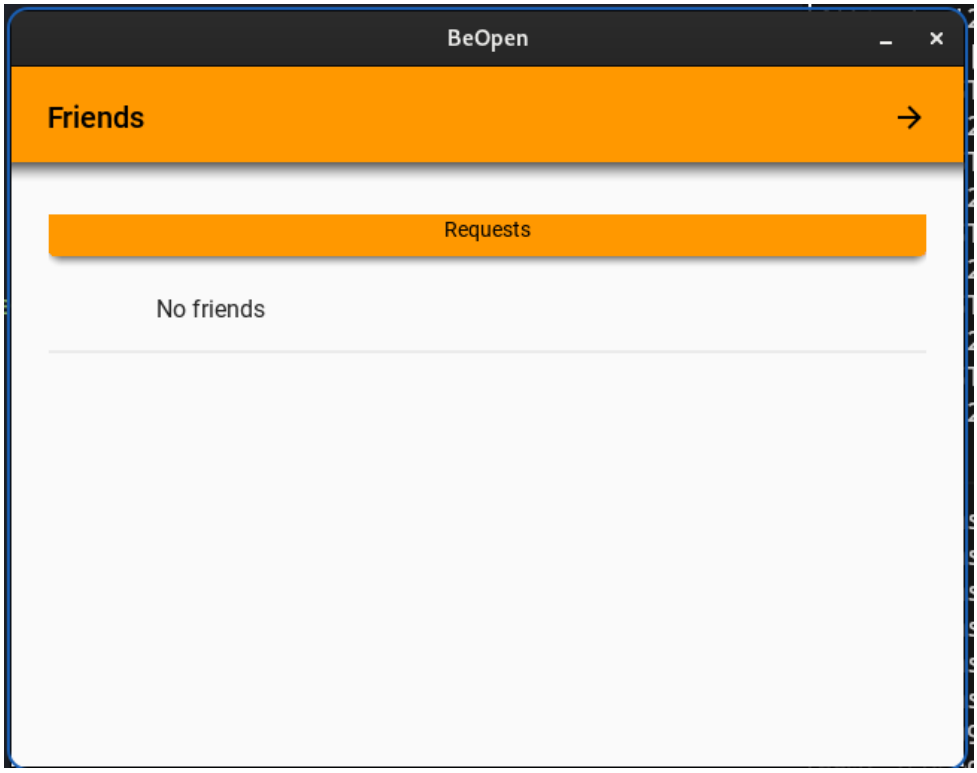
		database		
F.8A.I	Entering a valid username and pressing request	The box should clear and the request should be created on the server side database	The program freezes due to a dictionary key error	
F.8A.II	Entering a valid username and pressing request	The box should clear and the request should be created on the server side database	Shows the wrong status message and doesn't clear the text box	
F.8A.III	Entering a valid username and pressing request	The box should clear and the request should be created on the server side database	As expected	

Test Number	Image Number	Image

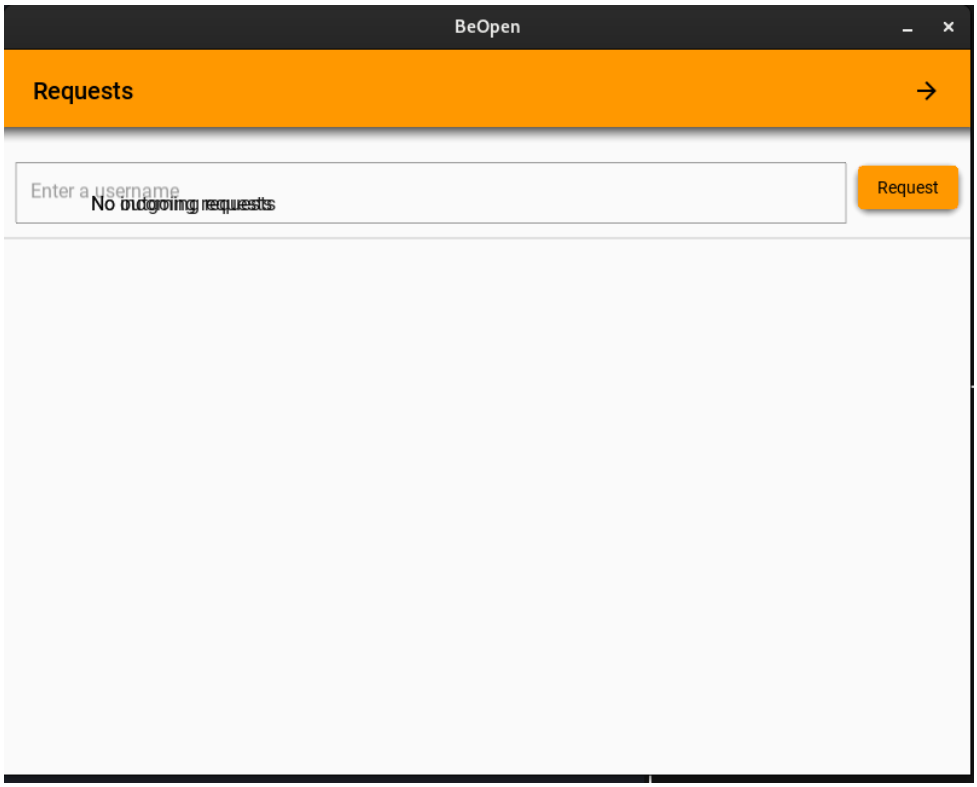
F.1A.I	1	
F.1A.II	1	

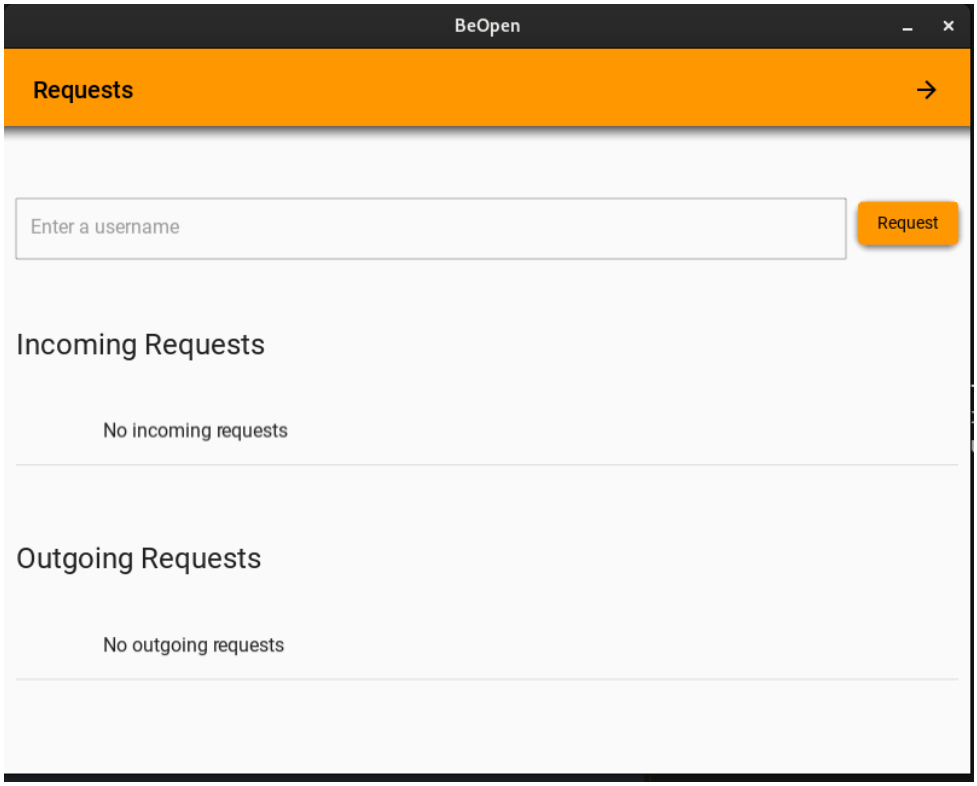
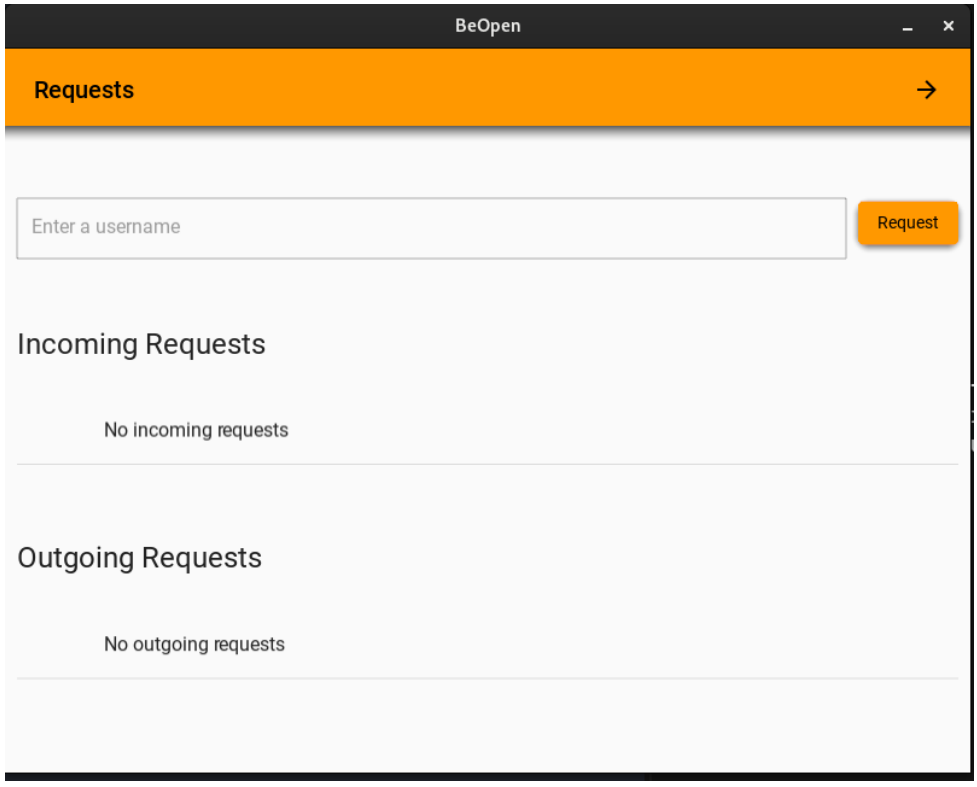
F.2A.I	1	 BeOpen Friends Requests user user3
F.2A.I	2	 BeOpen Friends Requests user

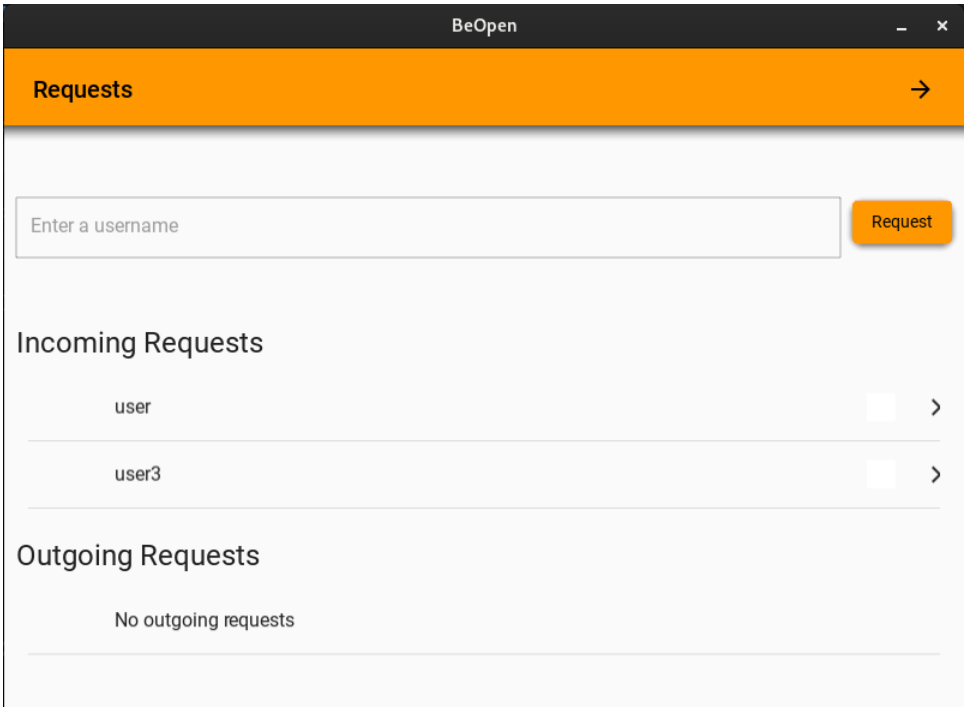
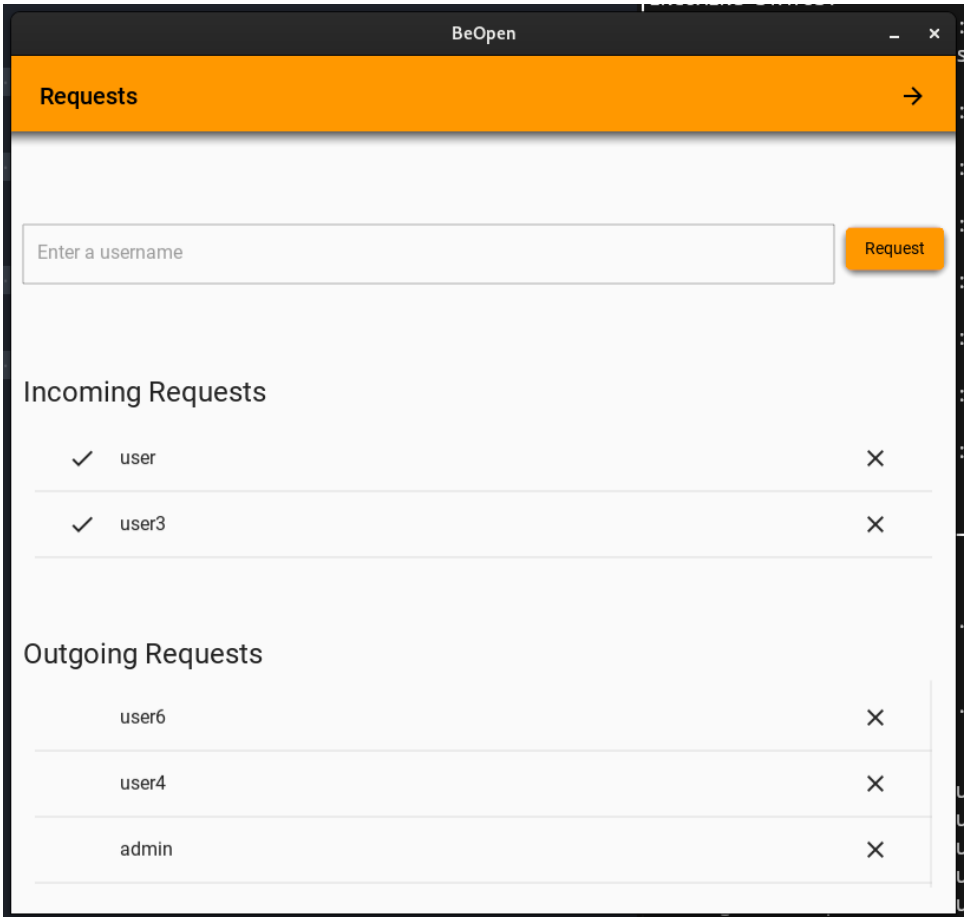
F.3A.I	1	 BeOpen Profile Username user2 Name user2 Role None Biography
F.FB.I	1	 BeOpen Requests Enter a username Request Incoming Requests No incoming requests Outgoing Requests No outgoing requests

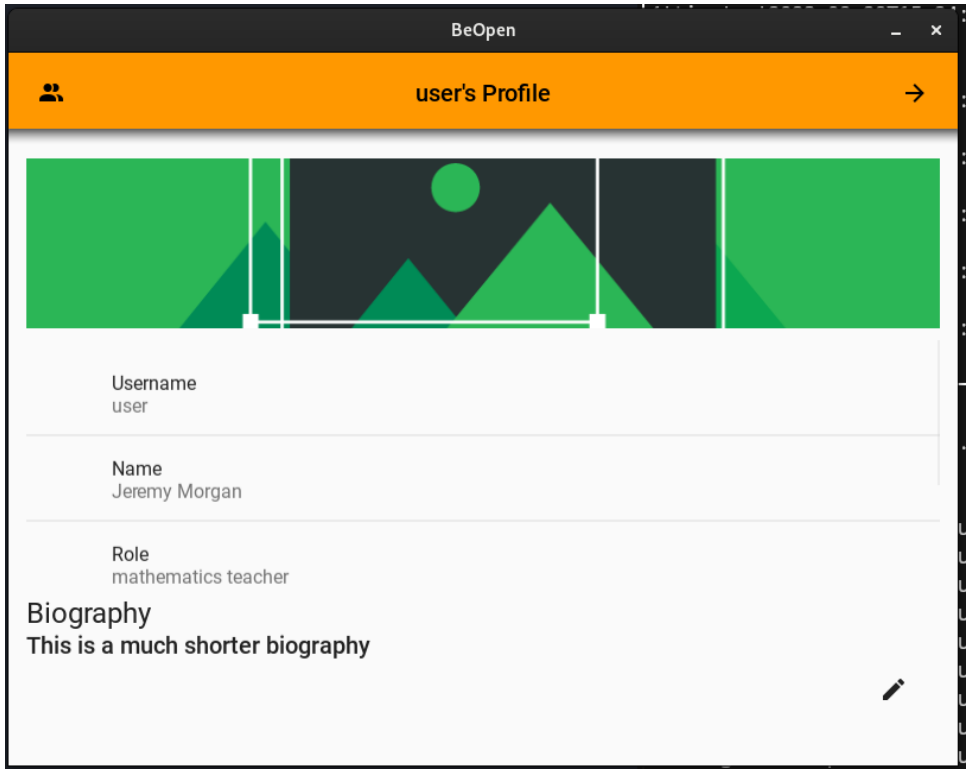
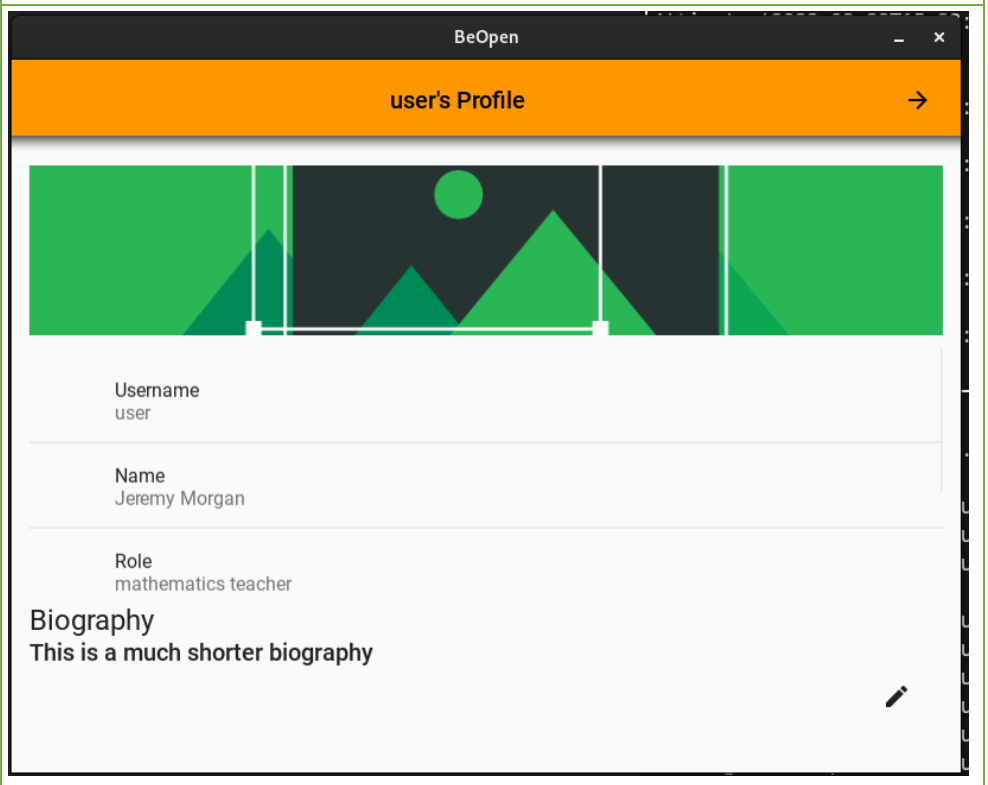
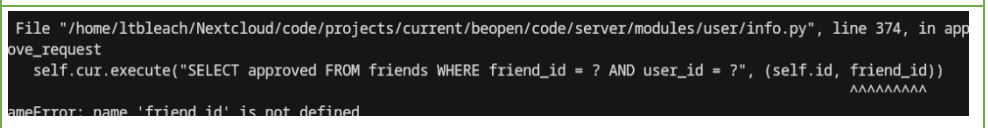
F.4B.I	1	
F.4B.II	1	

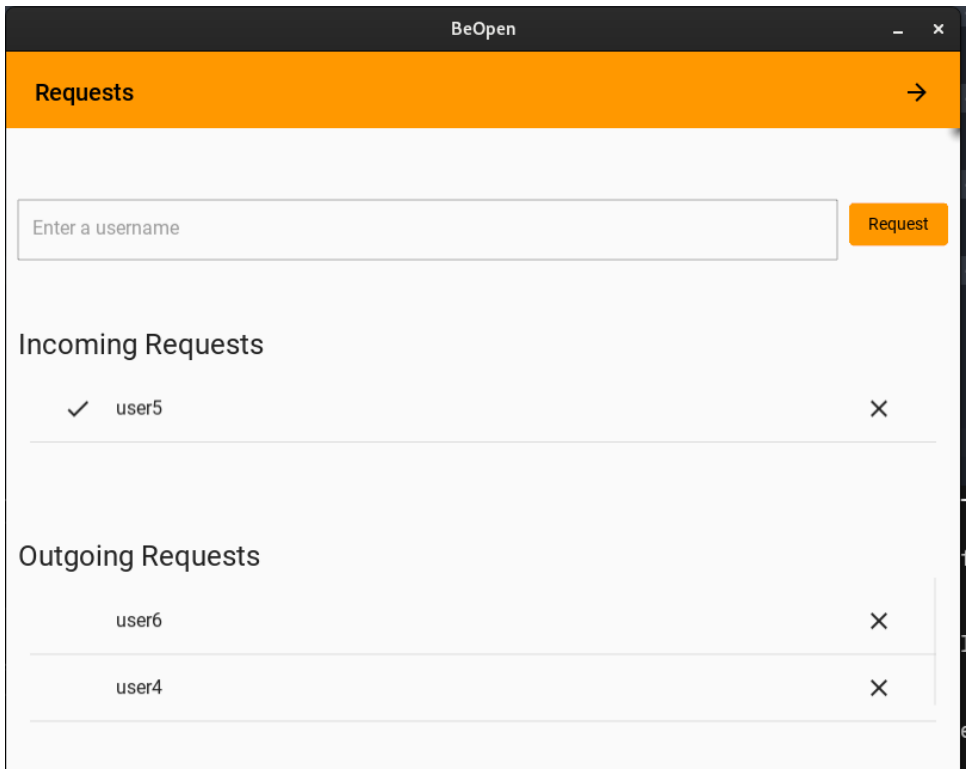
F.4B.II	2	

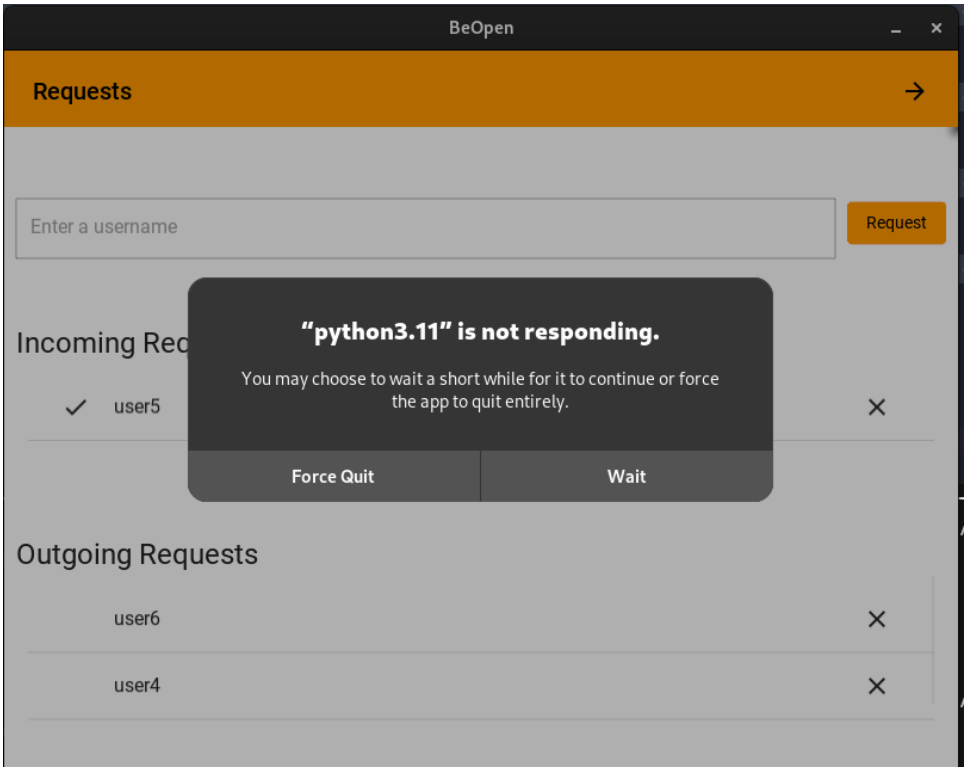
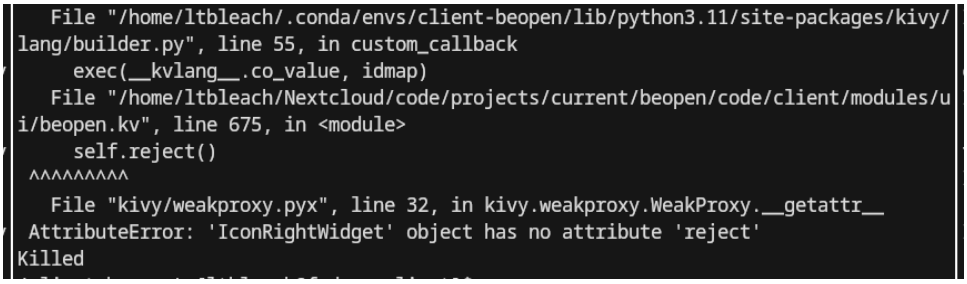
F.4B.III	1	
F.4B.V		

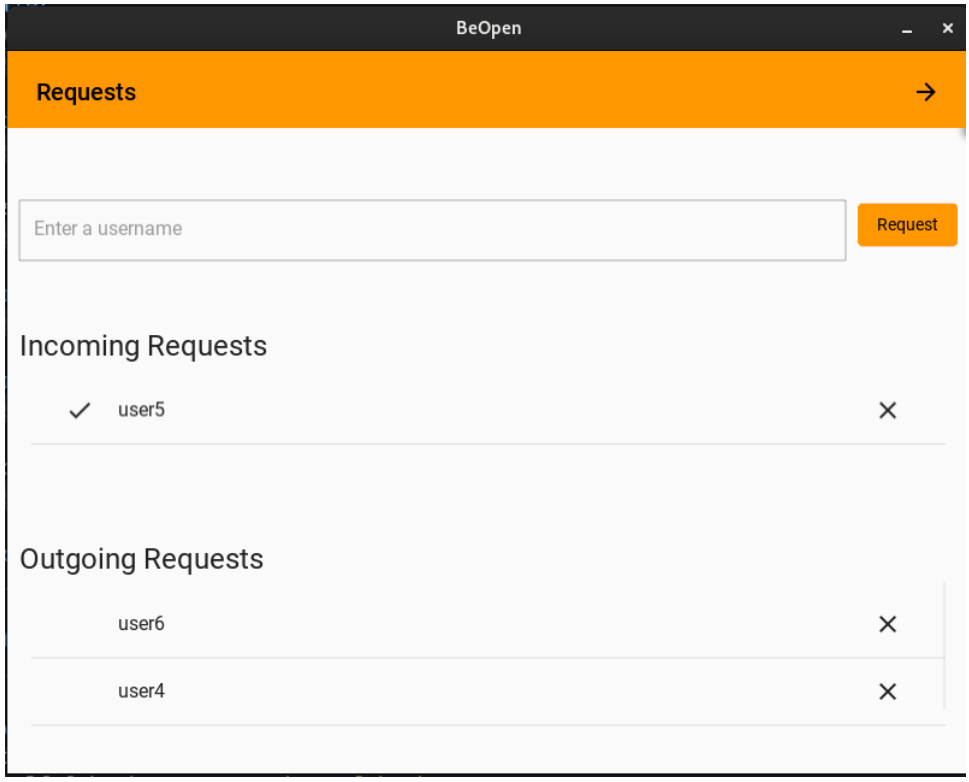
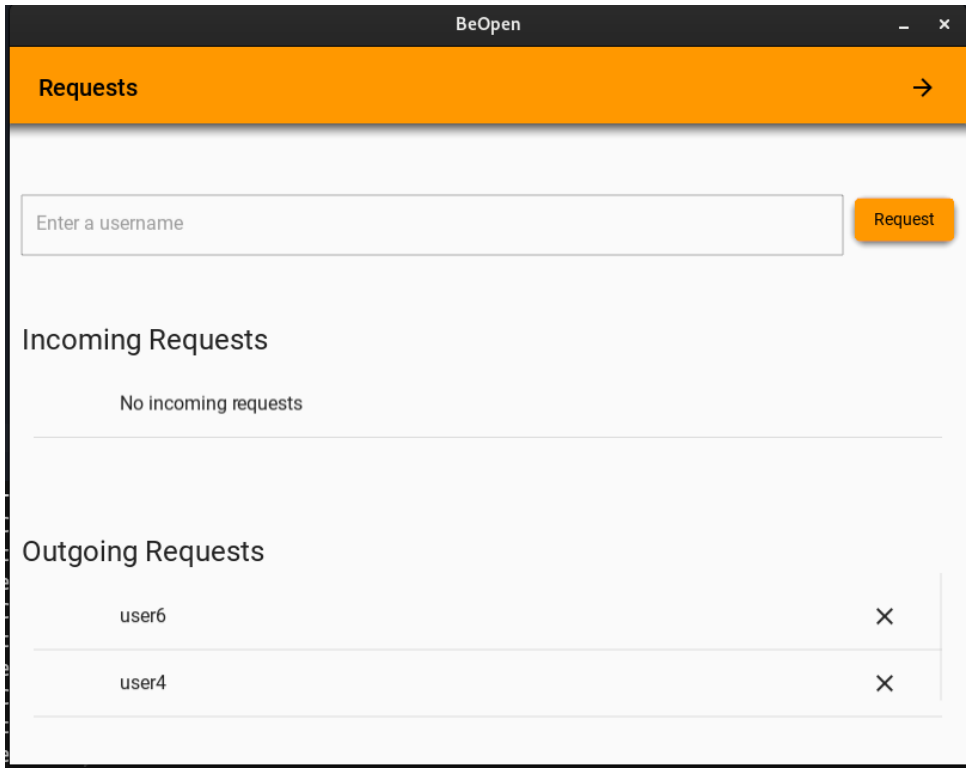
F.4B.VI	1	
F.5A.I	1	

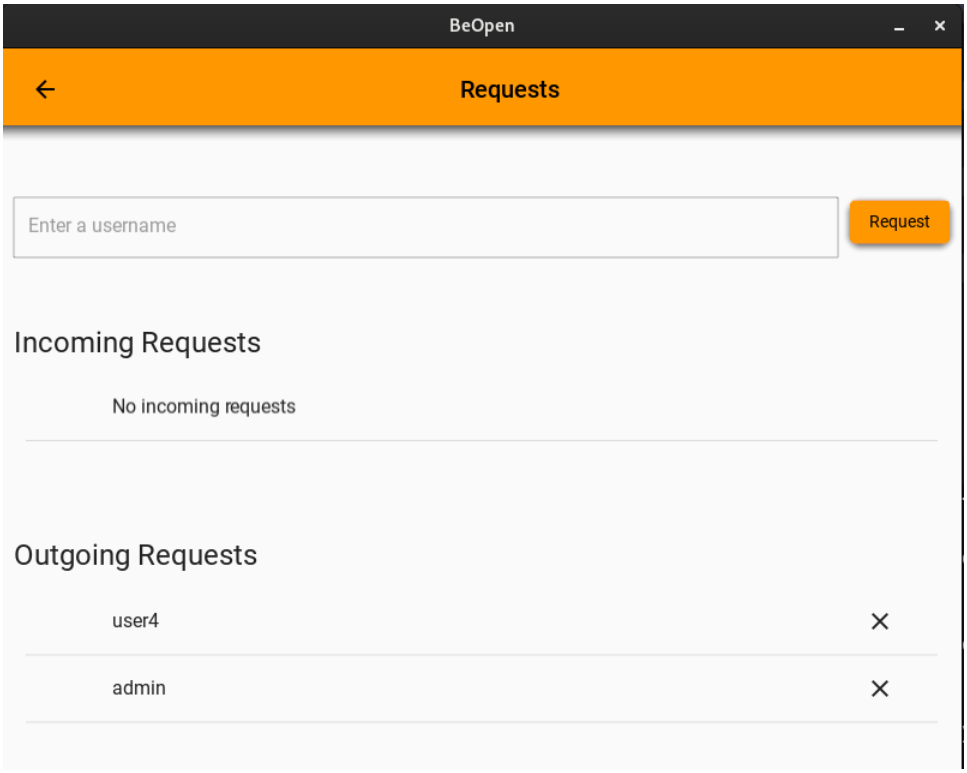
F.5B.I	1	
F.5B.II	1	

F.6A.I	1	
F.6A.II	1	
F.7A.I	1	

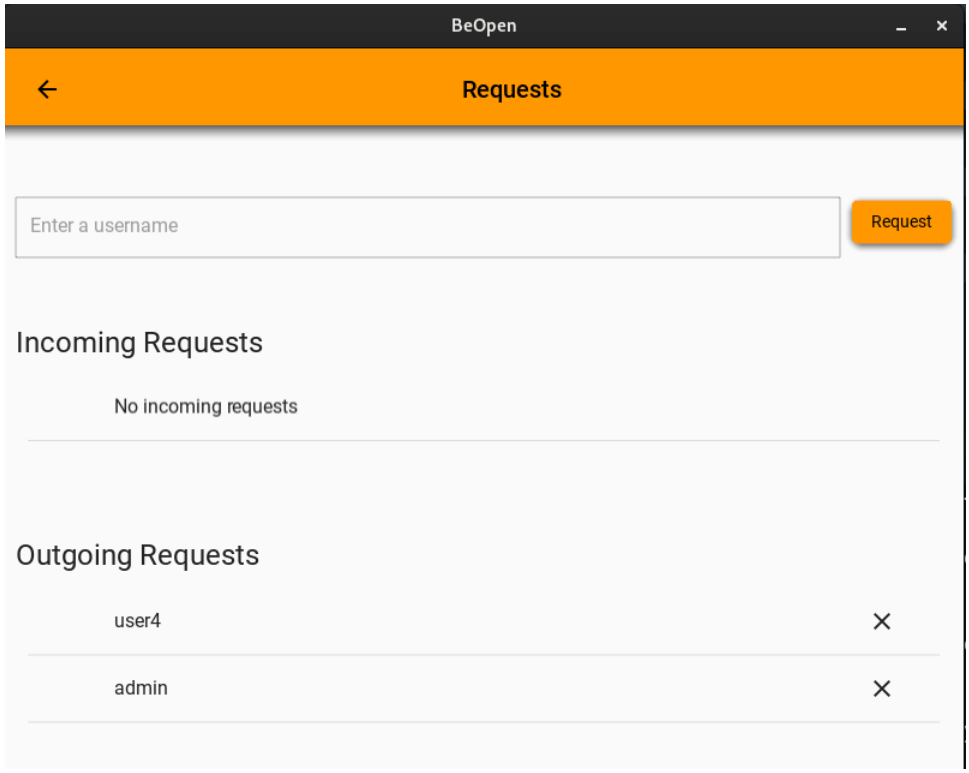
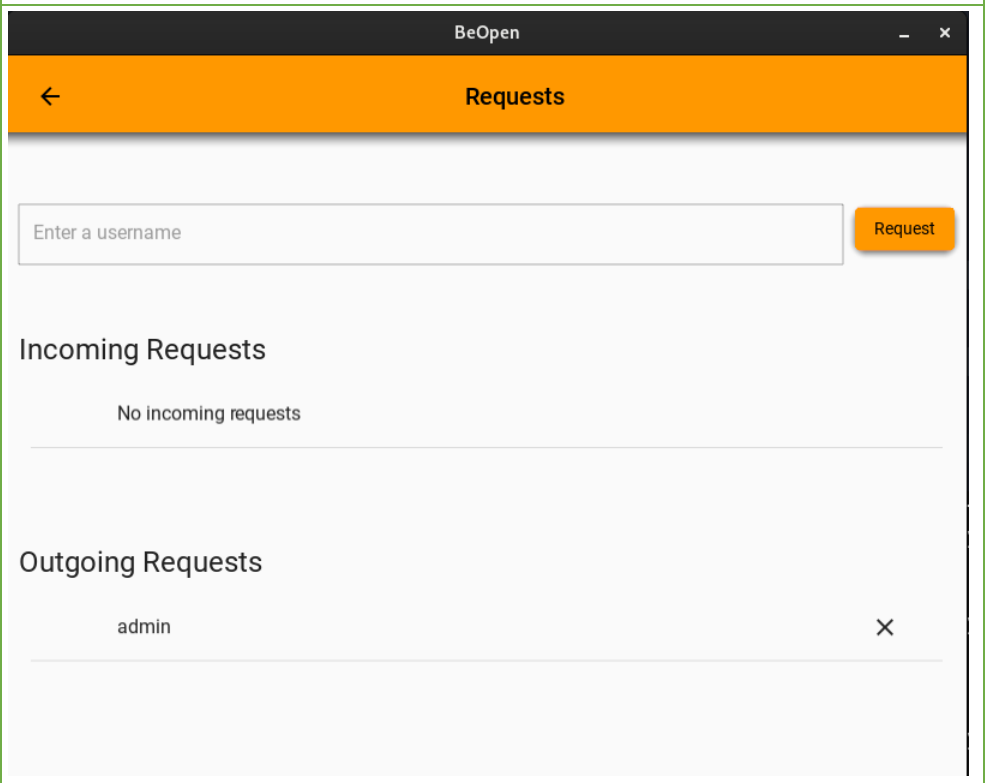
F.7A.II	1	
F.7A.III	1	
F.7B.I	1	

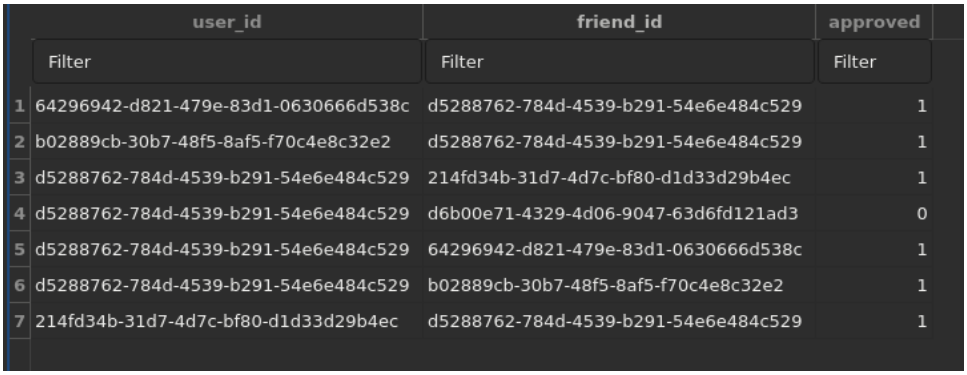
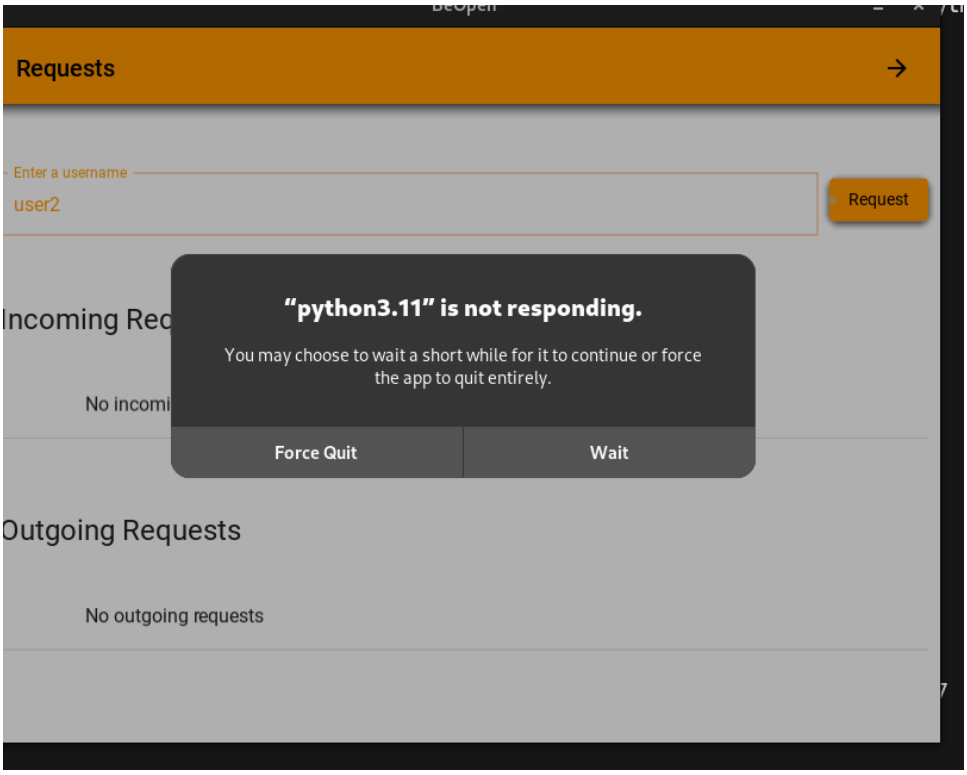
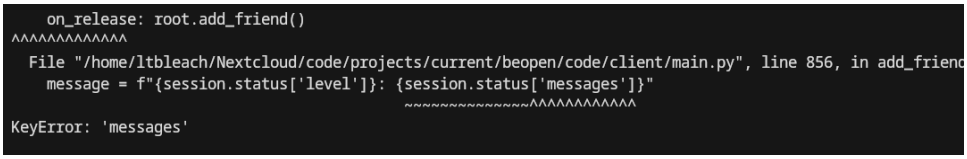
F.7B.I	2	 BeOpen Requests Enter a username Request Incoming Rec "python3.11" is not responding. You may choose to wait a short while for it to continue or force the app to quit entirely. user5 Force Quit Wait Outgoing Requests user6 user4
F.7B.I	3	 File "/home/ltbleach/.conda/envs/client-beopen/lib/python3.11/site-packages/kivy/lang/builder.py", line 55, in custom_callback exec(__kvlang__.co_value, idmap) File "/home/ltbleach/Nextcloud/code/projects/current/beopen/code/client/modules/ui/beopen.kv", line 675, in <module> self.reject() ^^^^^^^^^ File "kivy/weakproxy.pyx", line 32, in kivy.weakproxy.WeakProxy.__getattr__ AttributeError: 'IconRightWidget' object has no attribute 'reject' Killed

F.7B.II	1	
F.7B.III	1	

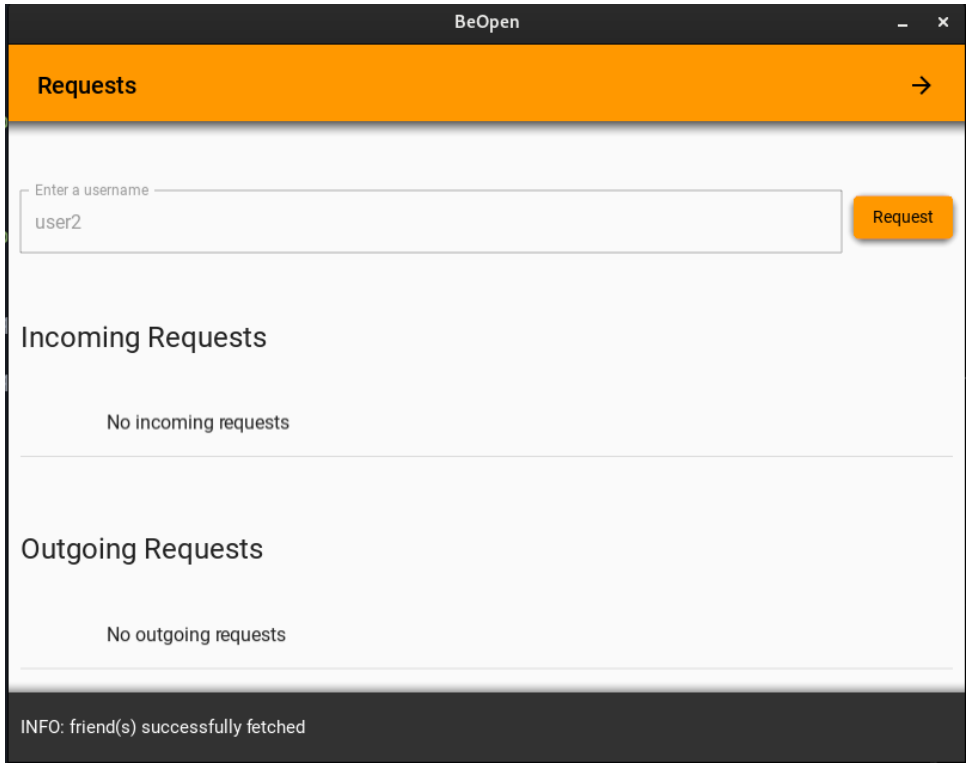
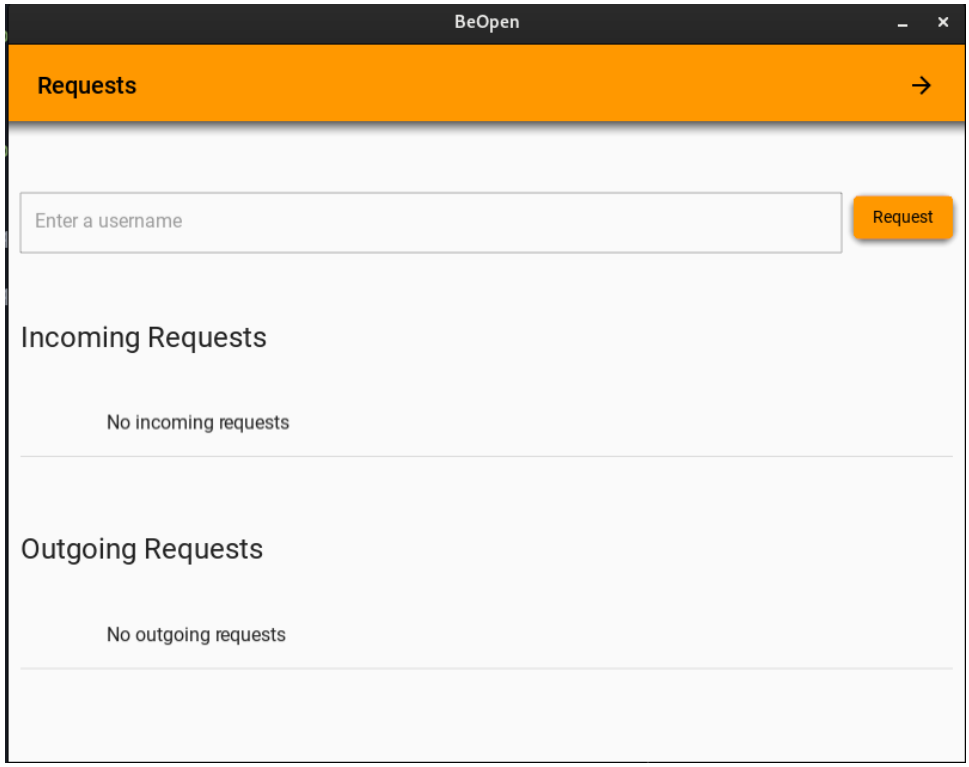
F.7B.III	2	
F.7C.I	1	

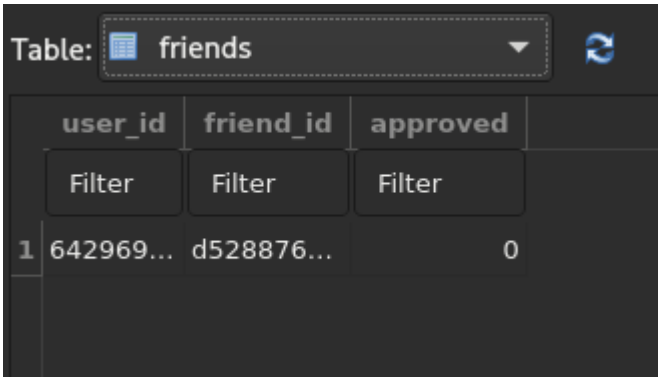
	user_id	friend_id	approved
	Filter	Filter	Filter
1	64296942-d821-479e-83d1-0630666d538c	d5288762-784d-4539-b291-54e6e484c529	1
2	b02889cb-30b7-48f5-8af5-f70c4e8c32e2	d5288762-784d-4539-b291-54e6e484c529	1
3	d5288762-784d-4539-b291-54e6e484c529	636cce26-039c-4b89-a82f-b95d218d1784	0
4	d5288762-784d-4539-b291-54e6e484c529	214fd34b-31d7-4d7c-bf80-d1d33d29b4ec	1
5	d5288762-784d-4539-b291-54e6e484c529	d6b00e71-4329-4d06-9047-63d6fd121ad3	0
6	d5288762-784d-4539-b291-54e6e484c529	64296942-d821-479e-83d1-0630666d538c	1
7	d5288762-784d-4539-b291-54e6e484c529	b02889cb-30b7-48f5-8af5-f70c4e8c32e2	1
8	214fd34b-31d7-4d7c-bf80-d1d33d29b4ec	d5288762-784d-4539-b291-54e6e484c529	1

F.7C.I	2	
F.7C.II	1	

F.7C.II	2	
F.8A.I	1	
F.8A.I	2	


```
    on_release: root.add_friend()
^^^^^^^^^^^^^
  File "/home/ltbleach/Nextcloud/code/projects/current/beopen/code/client/main.py", line 856, in add_friend
    message = f"{session.status['level']}: {session.status['messages']}"
                                           ~~~~~~~~~~~~~~~^^^^^^^^^^^^
KeyError: 'messages'
```

F.8A.II	1	
F.8A.III	1	

F.8A.III	2	 Table: friends user_id friend_id approved Filter Filter Filter 1 642969... d528876... 0

Notifications

Test Number	Test Description	Expected	Observed	Action
N2.1A.I	Entering the notification screen. Clicking the notification button at the top of the home screen	On release of the button the screen should change to the notification screen	The app crashed with no error messages displayed on the app itself or on the server	This was due to the client calling a non-existant event. This was fixed
N2.1A.II	Entering the notification screen. Clicking the notification button at the top of the home screen	On release of the button the screen should change to the notification screen	As expected	
N2.1B.I	UI elements and look of the page	There should be a topbar displaying “Notifications” or “{username}’s Notifications” if (for example an admin) is viewing another users notifications.	As expected	
N2.1C.I	Scrolling the notification list	On using the scroll wheel or swiping down the list of notifications should shift downward.	As expected	

N2.1D.I	Clicking the back button at the top of the notification page	This should change the page back to the homefeed (if accessing own notifications) or if an admin accessed the notifications of another user it should return the admin to the users profile.		
N2.2A.I	Displaying the notifications themselves. The user being used here has 0 notifications	The notification area should have a single item reading “no notifications”	As expected	
N2.2B.I	Displaying notifications on the notification screen. The user being used here has 5 notifications	There should be 5 notifications each displaying a title some without extra content. All should have a cross button next to them	The client crashed after clicking the notification button	No error message was displayed so this was likely a bad server call or server error. Error in the variable name for the notification information pull.
N2.2B.II	Displaying notifications on the notification screen. The user being used here has 5 notifications	There should be 5 notifications each displaying a title some without extra content. All should have a cross button	Notifications appeared with correct titles but the content of the notifications are is not correct	

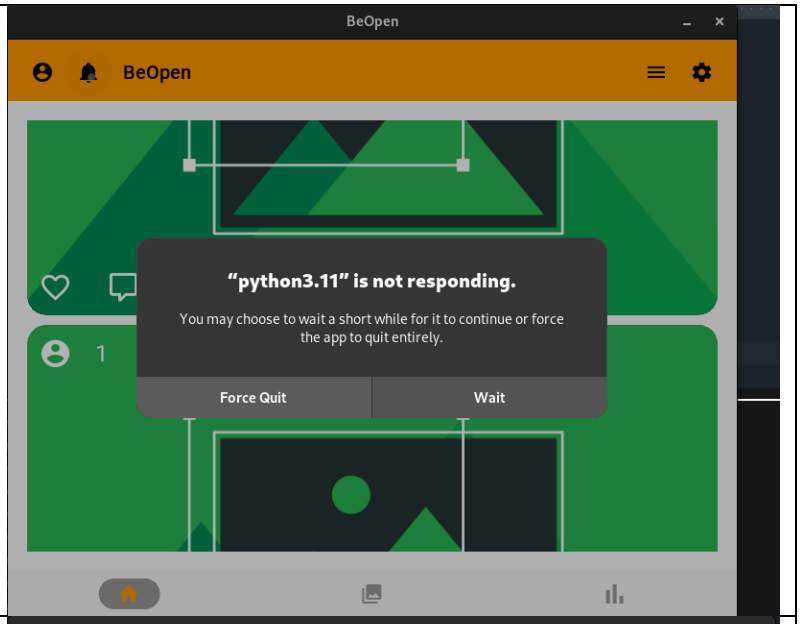
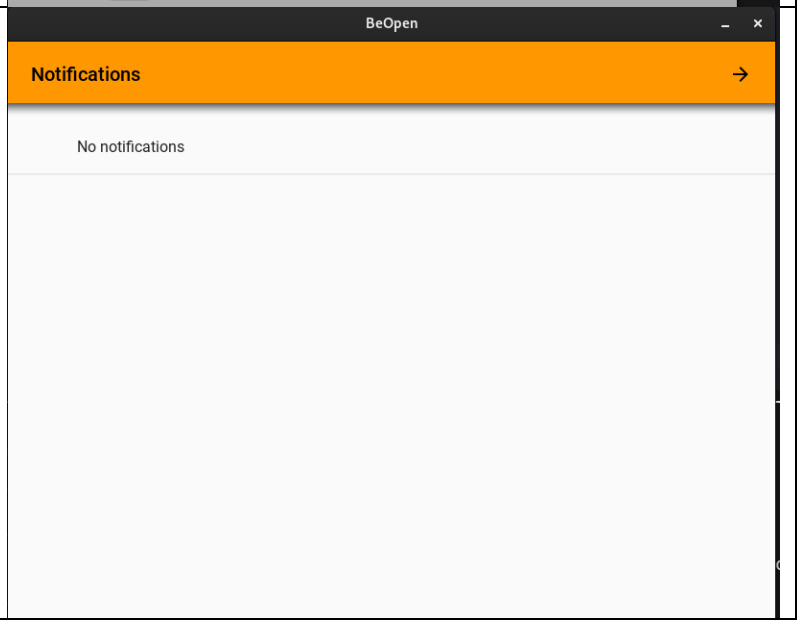
		next to them		
N2.2C.I	New notification being received by the client and added to the UI in real time. Here a user will be logged in on the notification screen when 5 test notifications are created, these notifications should then be put on the UI.	Here the user should see a new notification added to their already loaded notification page	The client errors out complaining about a dictionary key error.	This is because the notification_id isnt sent by the user notification service. This is a server side service that provides live notifications to logged in users. I will change this so the service also provides the notification ID to the client.
N2.2C.II	New notification being received by the client and added to the UI in real time.Here a user will be logged in on the notification screen when 5 test notifications are created, these notifications should then be put on the UI.	Here the user should see a new notification added to their already loaded notification page	The client errors, this time its that graphics are trying to be added outside of the main thread of the gui.	This is a limitation of the gui framework but will be substituted with a refresh button. For this reshresh button to work the load_content method was tweaked to clear notifications first before fetching all again. And the user will receive a live notification from their OS anyway.
N2.2C.III	While the user is on the notifications	The newly created notifications	As expected	

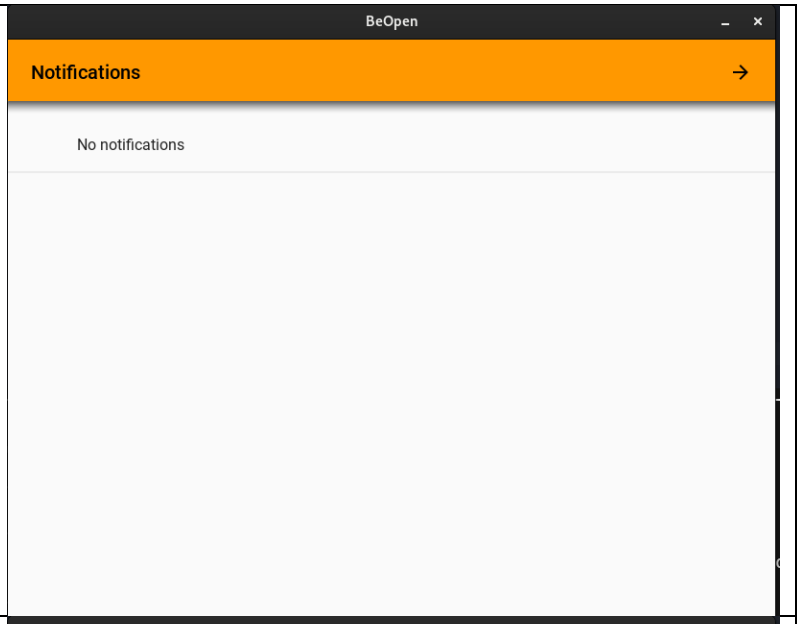
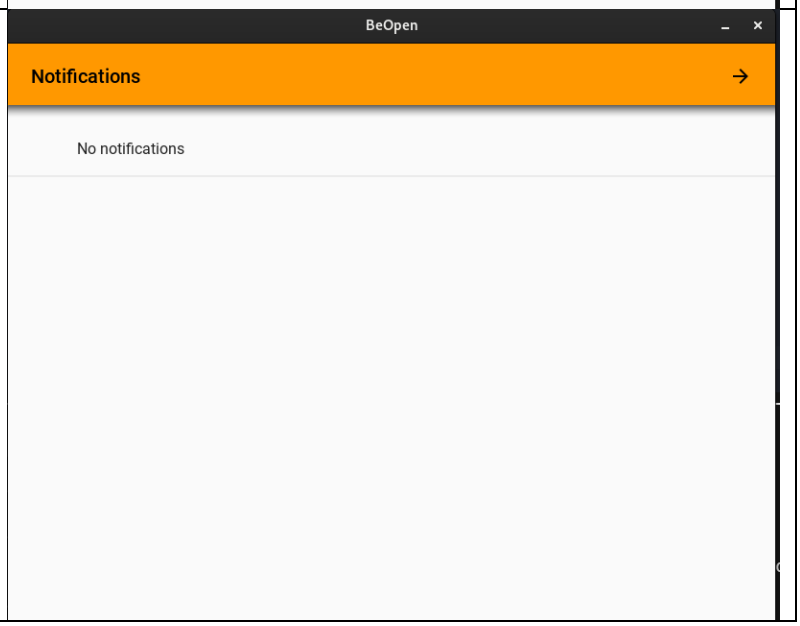
	page another client will create test notifications. The first user will then click the refresh button on their notifications page	should appear in the list.		
N2.3A.I	Pressing the cross button next to a notification, to remove the notification	The UI should update to remove the selected notification and this change should also be reflected on the server side database	Minor syntax error server side, other than that as expected.	
N2.3B.I	Pressing the cross button next to a notification, to remove the notification. In this case it's the last notifications	The UI should update to remove the selected notification, and display "No notifications" and this change should also be reflected on the server side database	The list goes blank after removing the last notifications	The UI is not refreshed after removing a notification. Now at the end of the delete method the load_content method is called again.
N2.4A.I	Users of 3 different levels will be logged into 3 clients. There will be a "member" user, "management" user and an	Of the 3 users only the **admin** should receive the notification.	As expected	

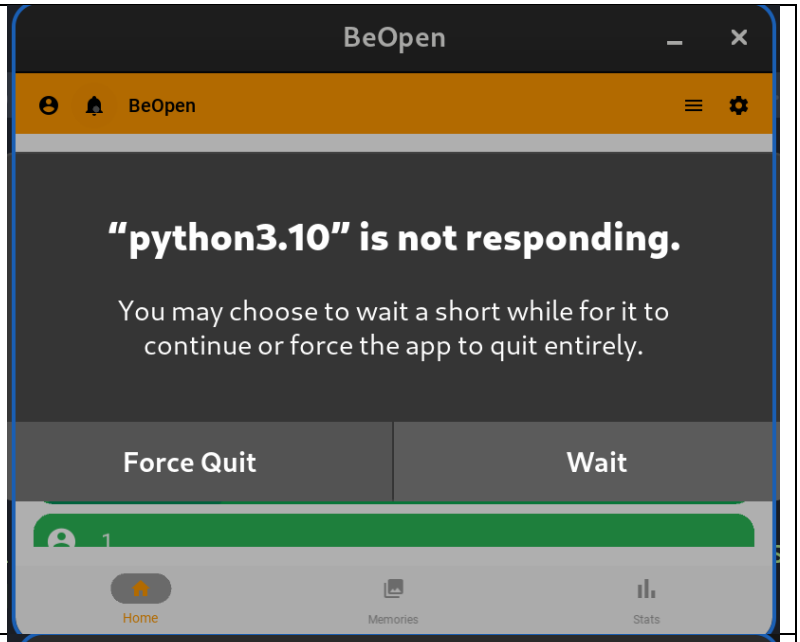
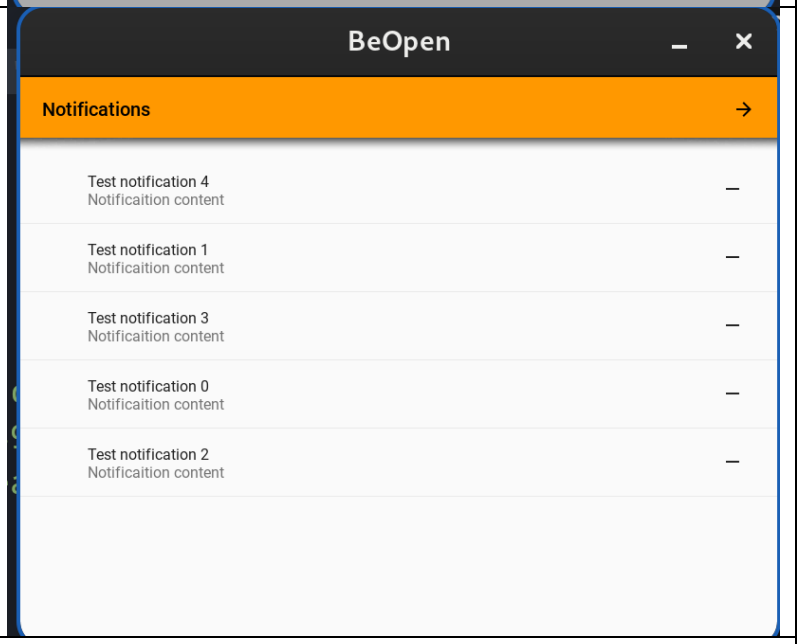
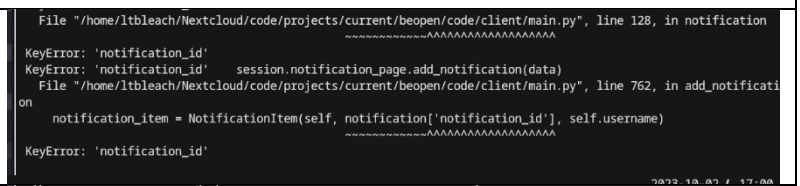
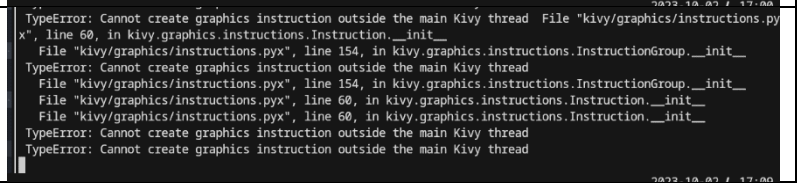
	“admin” user. A notification directed towards only **admins** will be generated			
N2.4B.I	Users of 3 different levels will be logged into 3 clients. There will be a “member” user, “management” user and an “admin” user. A notification directed towards only **management** will be generated	Of the 3 users only the **management** user should receive the notification.	As expected	
N2.4C.I	Users of 3 different levels will be logged into 3 clients. There will be a “member” user, “management” user and an “admin” user. A notification directed towards only **members** will be generated	Of the 3 users only the **member** user should receive the notification.	As expected	
N2.5A.I	2 users will be logged into 2 different clients one user will be part of the “teachers” team	Of the 2 users only the user apart of the “teachers” team should receive		

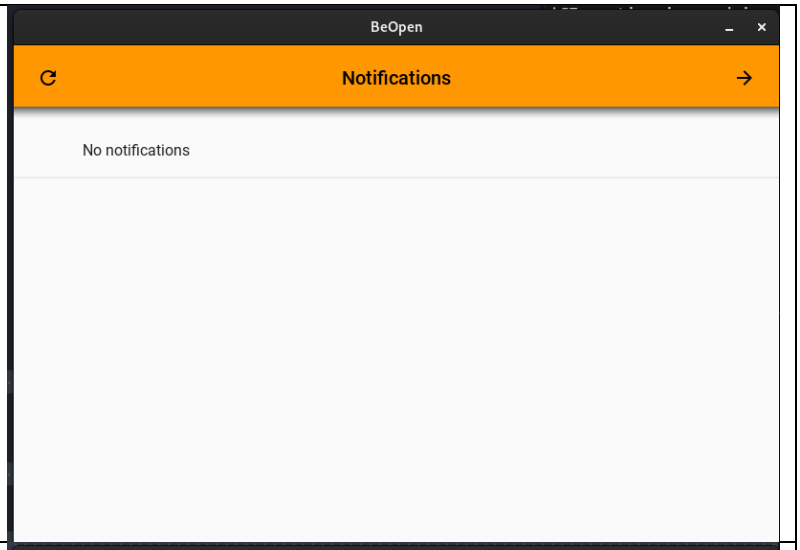
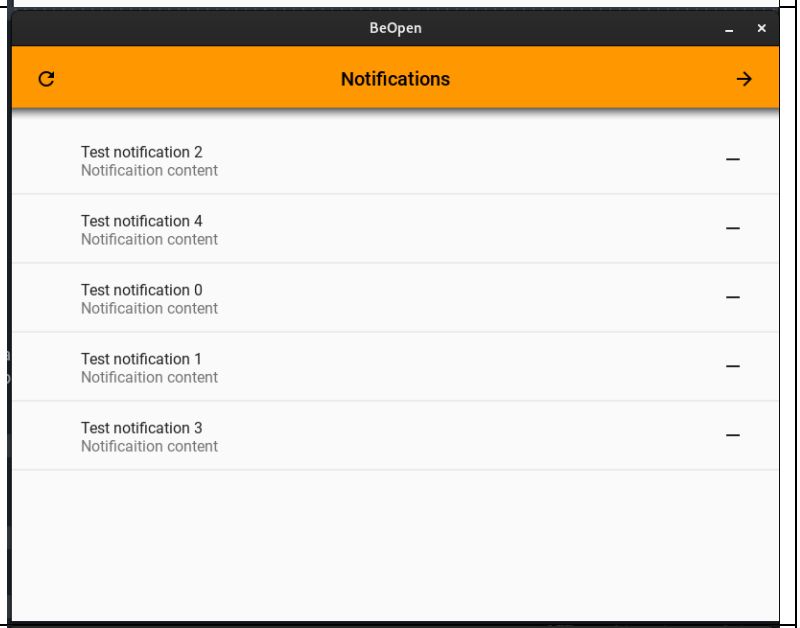
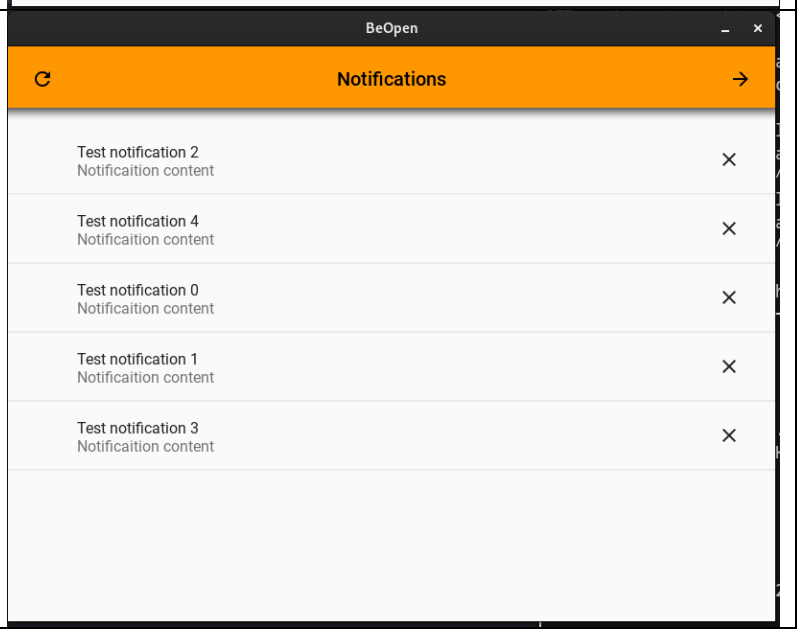
	and the other apart of the “students” team. A notification will be generated intended for “teachers” team members only	the notification		
N2.6A.I	2 users will be logged into 2 different clients. One with username Adam and the other with username Betty A notification will be generated intended for Betty	Of the 2 users only Betty should receive the notification.	As expected	
N2.7A.I	The client will receive a “post time” notification. The title of this type of notification is unique and so requires some processing before displaying to the user	The title of the notification should simply read “Post time” not displaying the server code (which is originally sent by the server in the title)		

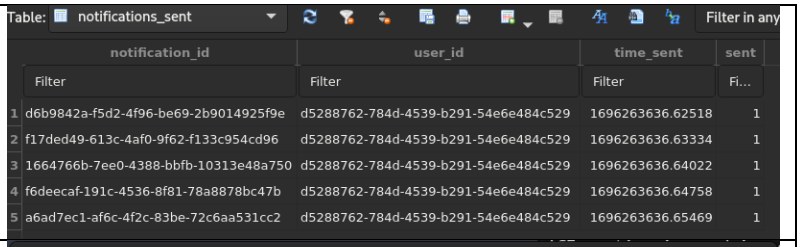
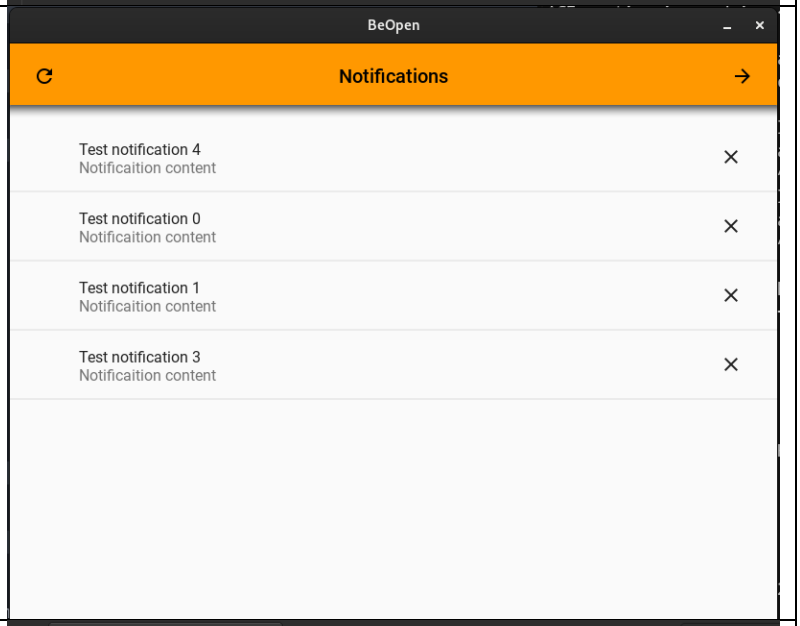
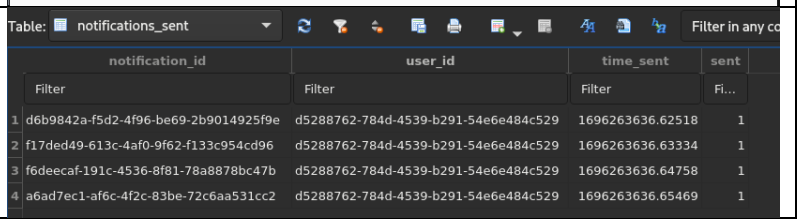
Test Number	Image Number	Image

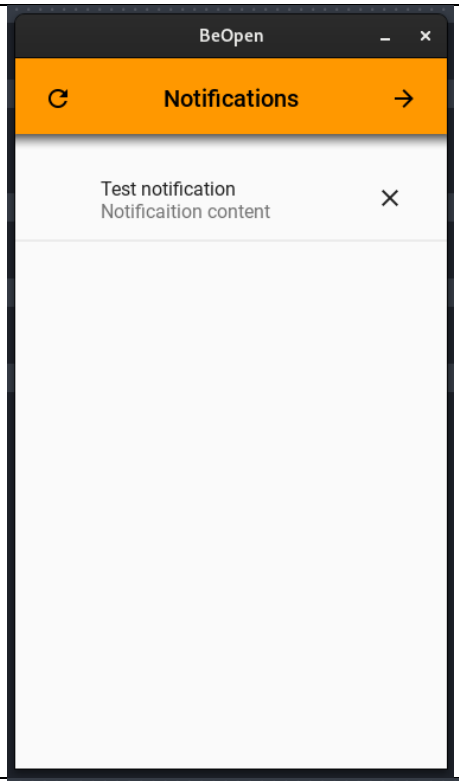
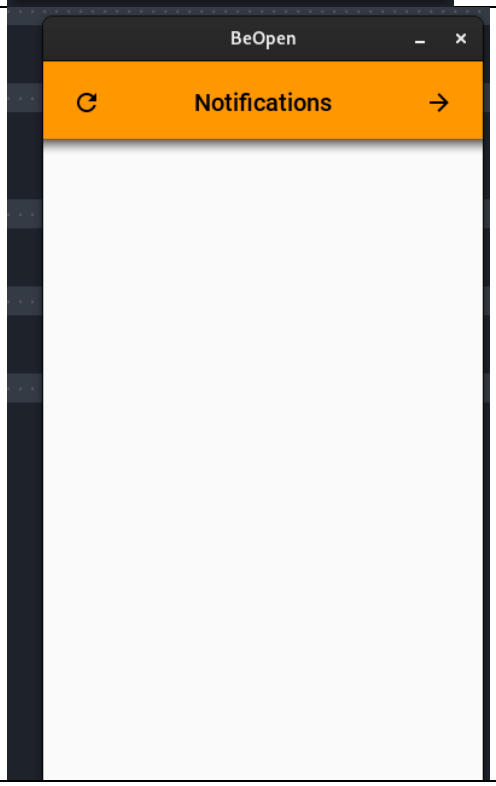
N2.1A.I	1	
N2.1A.II	1	

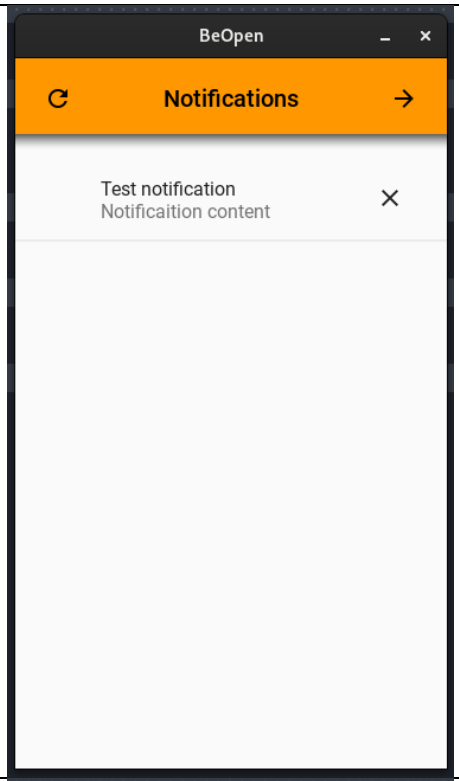
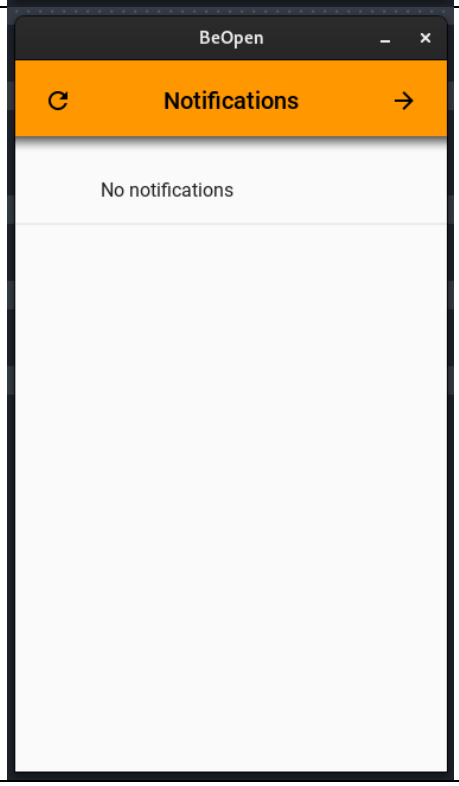
N2.1B.I	1	
N2.2A.I	1	

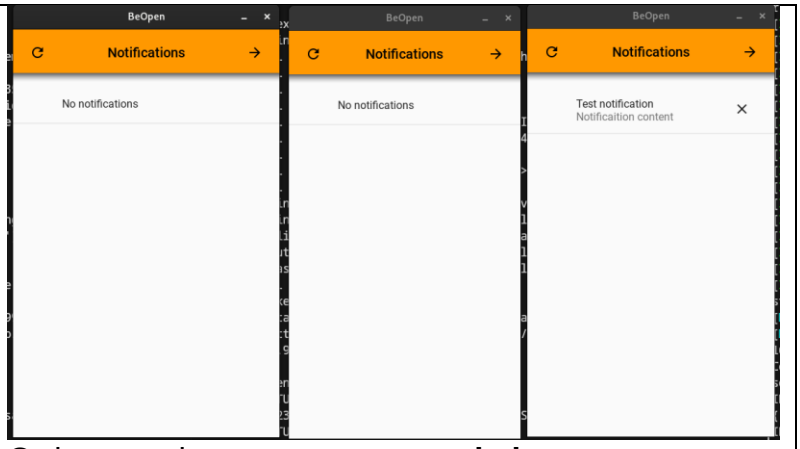
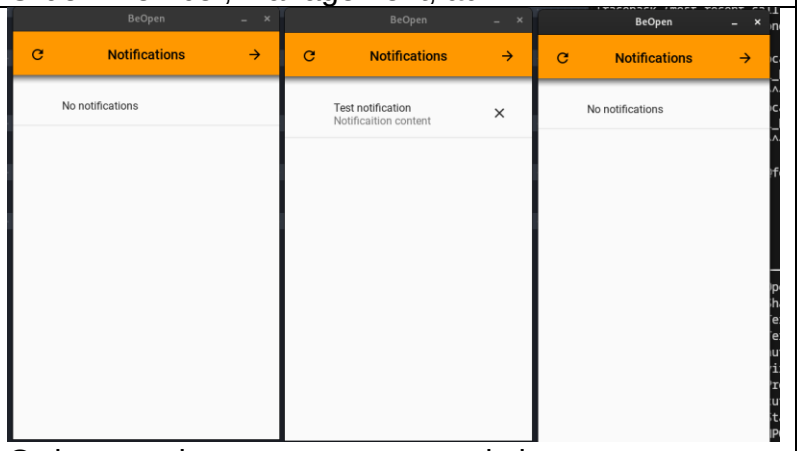
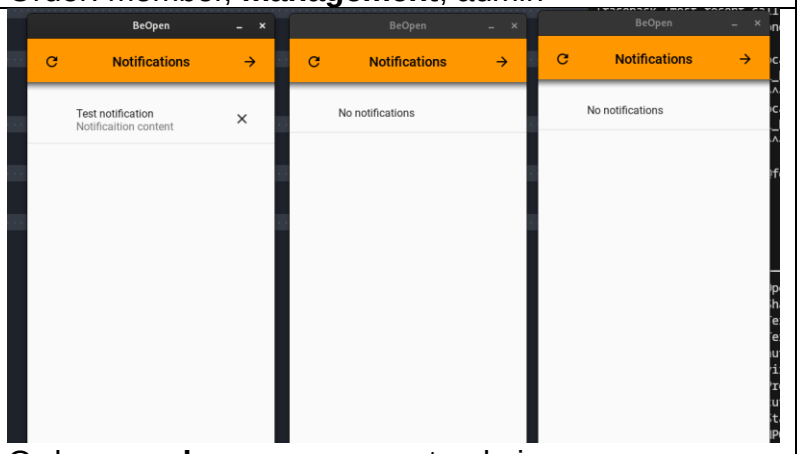
N2.2B.I	1	
N2.2B.II	1	
N2.2B.III	1	
N2.2C.I	1	
N2.2C.II	1	

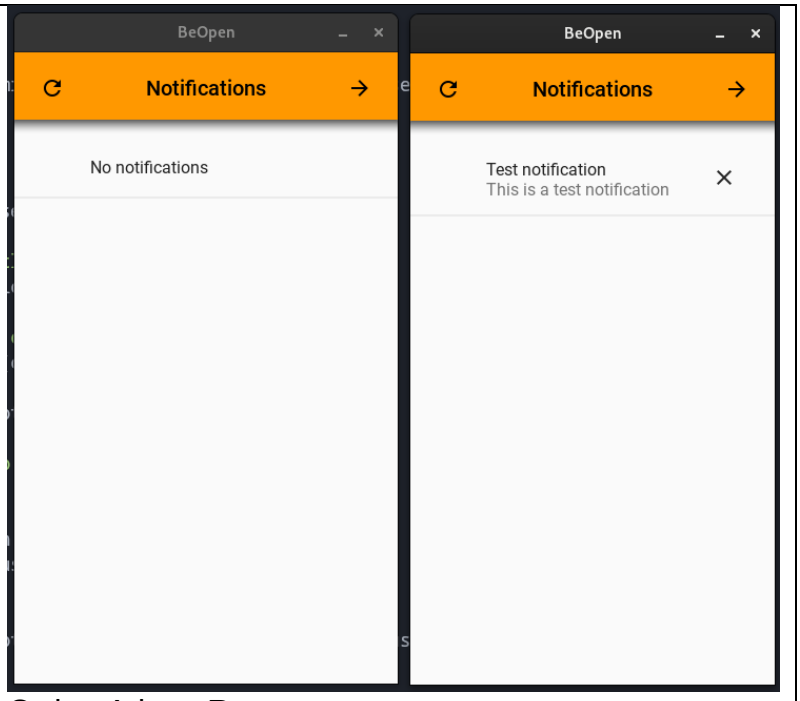
N2.2C.III	1	
N2.2C.III	2	
N2.3A.I	1	

N2.3A.I	2	
N2.3A.I	3	
N2.3A.I	4	

N2.3B.I	1	
N2.3B.I	2	

N2.3B.II	1		
N2.3B.II	2		

N2.4A.I	1	 Order: member, management, **admin**
N2.4B.I	1	 Order: member, **management**, admin
N2.4C.I	1	 Order: **member**, management, admin

N2.6A.I	1	 Order: Adam, **Betty**

Occupation requests

Test Number	Test Description	Expected	Observed	Action
O.1A.I	Clicking the occupations button on the organisation page	This should change the displayed screen to the “occupations page”	As expected	
O.1B.I	The page UI, look and navigation functionality. This involves scrolling	There should be a top bar displaying a back button the top bar should also read the name of the page. There should be a button at the top and a area for the list of occupations	The content starts halfway down the page. There is also no padding on the side of the page. There is also no requests button	
O.1B.II	The page UI, look and navigation functionality. This involves scrolling	There should be a top bar displaying a back button the top bar should also read the name of the page. There should be a button at the top and a area for the list of occupations	As expected	
O.1C.I	Pressing the back button at the top of the	This should bring the user back to the	As expected	

	page	management page		
O.2A.I	The occupation list itself Here the server instance has no occupations	The list should simply display that there are “no occupations”.	As expected	
O.2B.I	The occupation list itself. Here the server instance has 4 occupations	The list should display all 4 occupations with their name, description below and an edit button along side.		
O.3A.I	Clicking the edit button on an occupation	This should create a new “editing area” above the occupation list. It should contain 2 text boxes with text already inside displaying the title in one, displaying the description in the other. There should also be a “done” button		
O.3B.I	Editing an occupation name. Here the title (or name) of an occupation will be changed from	On the client side the “edit area” should disappear after clicking the done button and the relevant occupation	Nothing changed on the UI side and nothing changed on the serverside database. However neither	The wrong event was being called on the client side. The client was calling occupation_set instead of

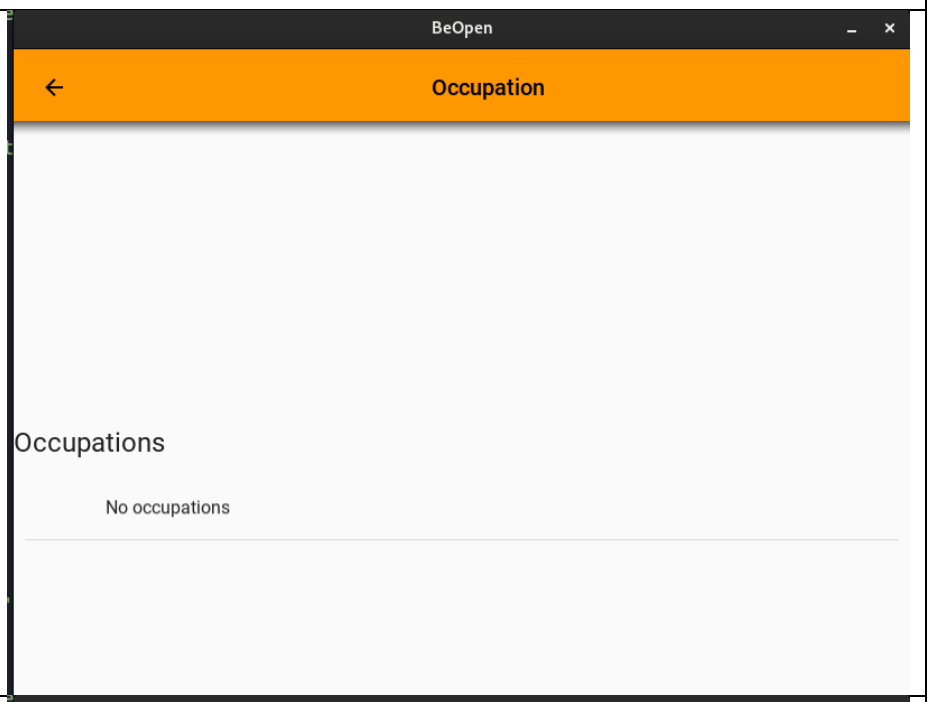
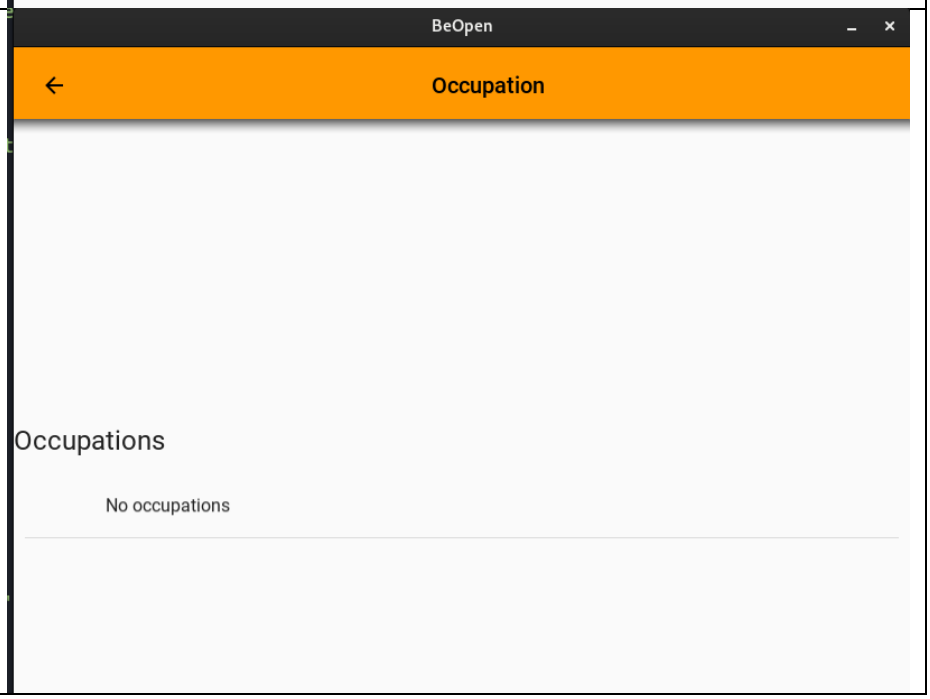
	“maths teacher” to “mathematics teacher”. Then the “done” button will be pressed	should be updated in the displayed list. On the server side the occupation should be updated in the database to reflect the change	created an error	occupation_edit
O.3B.II	Editing an occupation name. Here the title (or name) of an occupation will be changed from “maths teacher” to “mathematics teacher”. Then the “done” button will be pressed	On the client side the “edit area” should disappear after clicking the done button and the relevant occupation should be updated in the displayed list. On the server side the occupation should be updated in the database to reflect the change	As expected	
O.3C.I	Editing an occupation description. Here the description of an occupation will be changed from “teachers who teach maths” to	On the client side the “edit area” should disappear after clicking the done button and the relevant occupation should be	As expected	

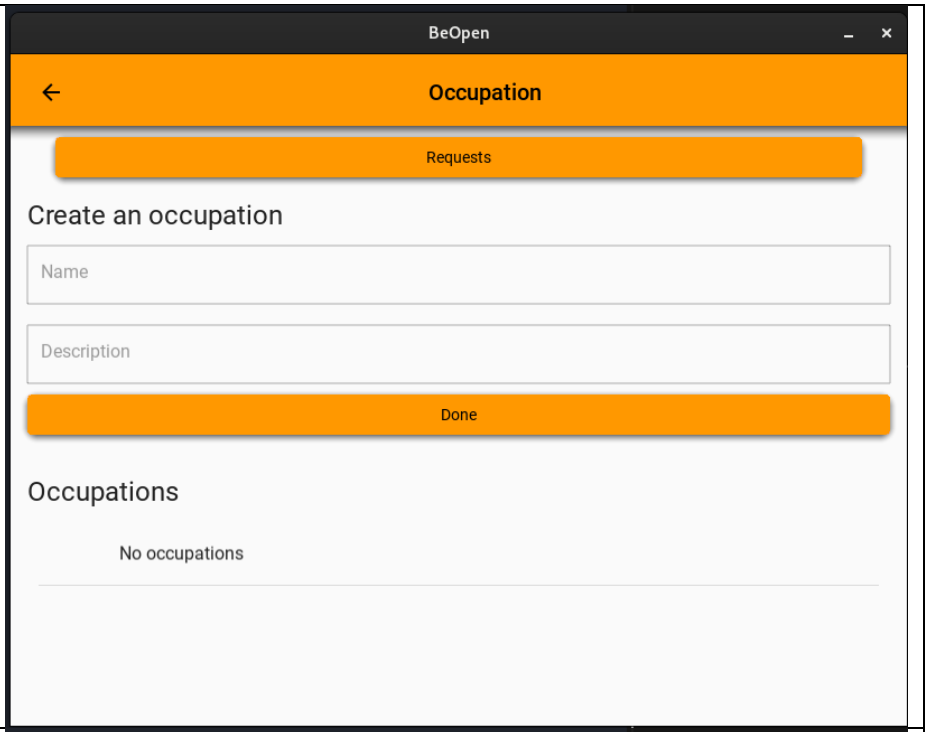
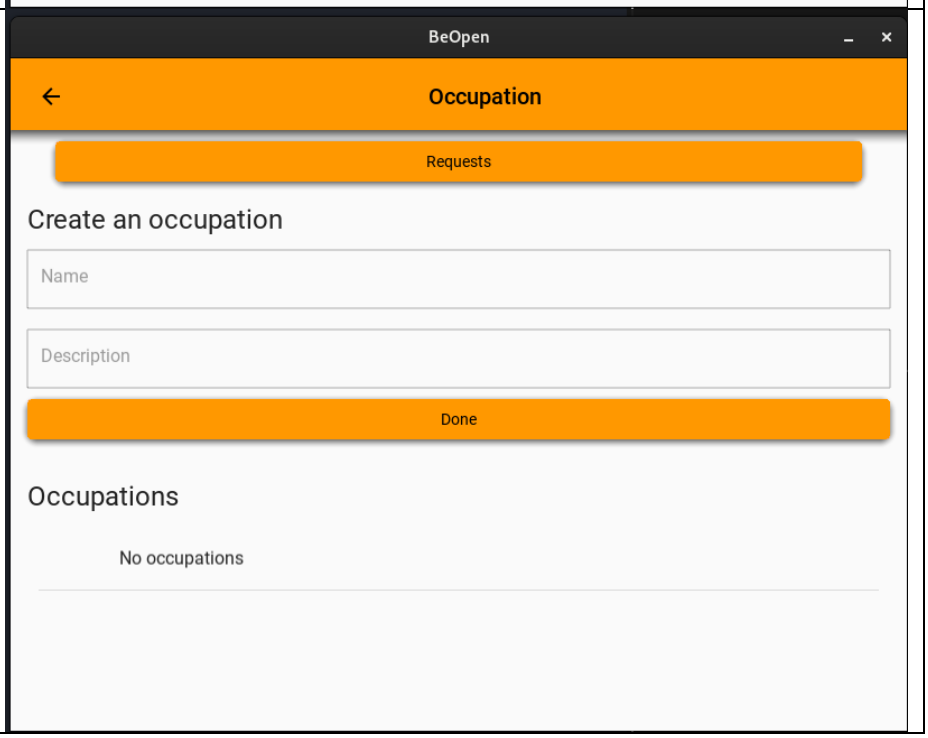
	“teachers who teach mathematics”. Then the “done” button will be pressed	updated in the displayed list. On the server side the occupation should be updated in the database to reflect the change		
O.3D.I	Editing an occupation name and description. Here the title (or name) of an occupation will be changed from “maths teacher” to “mathematics teacher”. And the description of an occupation will be changed from “teachers who teach maths” to “teachers who teach mathematics”. Then the “done” button will be pressed	On the client side the “edit area” should disappear after clicking the done button and the relevant occupation should be updated in the displayed list. On the server side the occupation should be updated in the database to reflect the change	As expected	
O.3E.I	Pressing the edit button not changing anything and clicking done	On the client side the “edit area” should disappear after clicking the done button and	As expected	

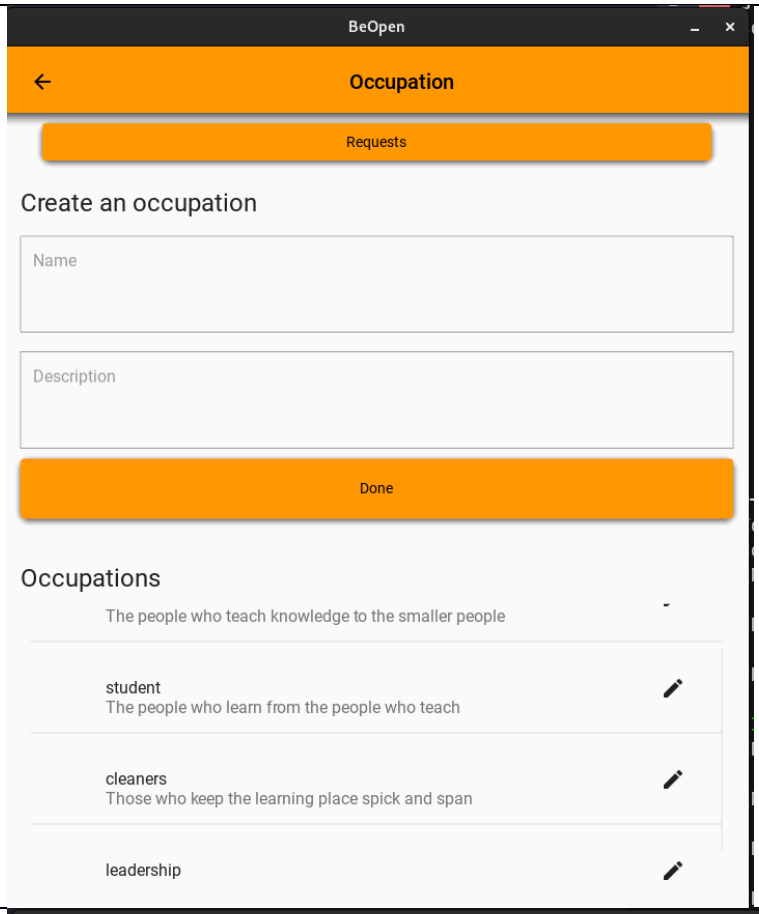
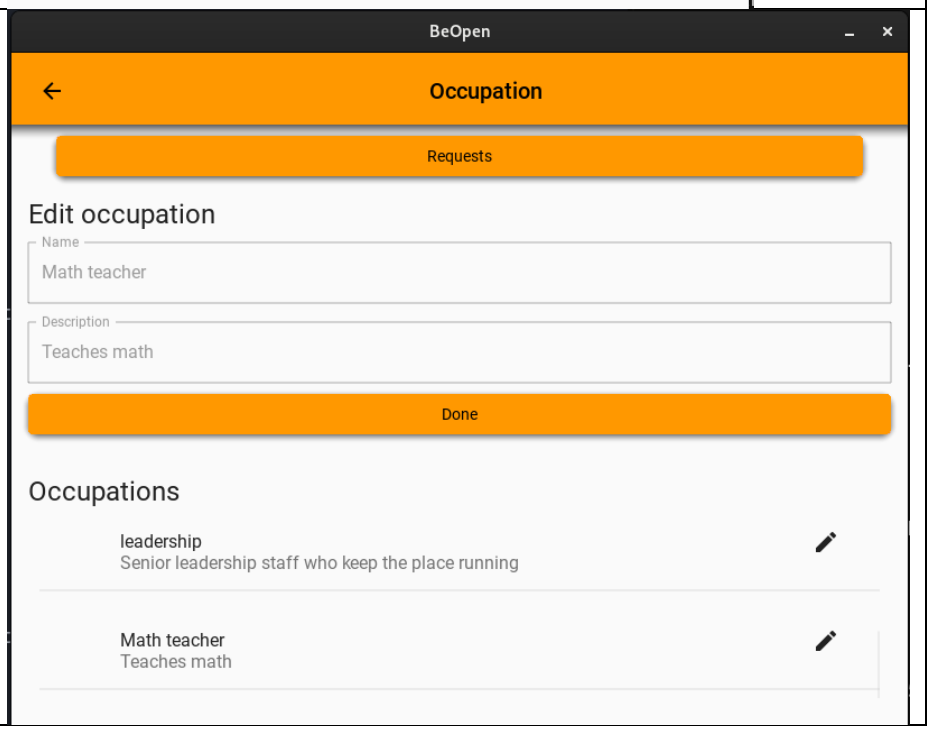
		nothing should be changed on server side database		
O.4A.I	Creating an occupation with a valid name and description and clicking done	This should clear the text boxes and add the new occupation to the server side databse and the occupations list should refresh.	As expected	
O.5A.I	Pressing the delete button on an occupation.	This should update the UI to remove the occupation and the change should be reflected in the server side database	As expected	
O.6A.I	Pressing the “requests” button	This should change the page to the occupation requests page.	As expected	
O.6B.I	The UI and look of the occupation requests page	There should be a top bar with a back button and the name of the page at the top. There should be a single scrollable list of occupation requests	Uneeded padding around the edges of the screen	

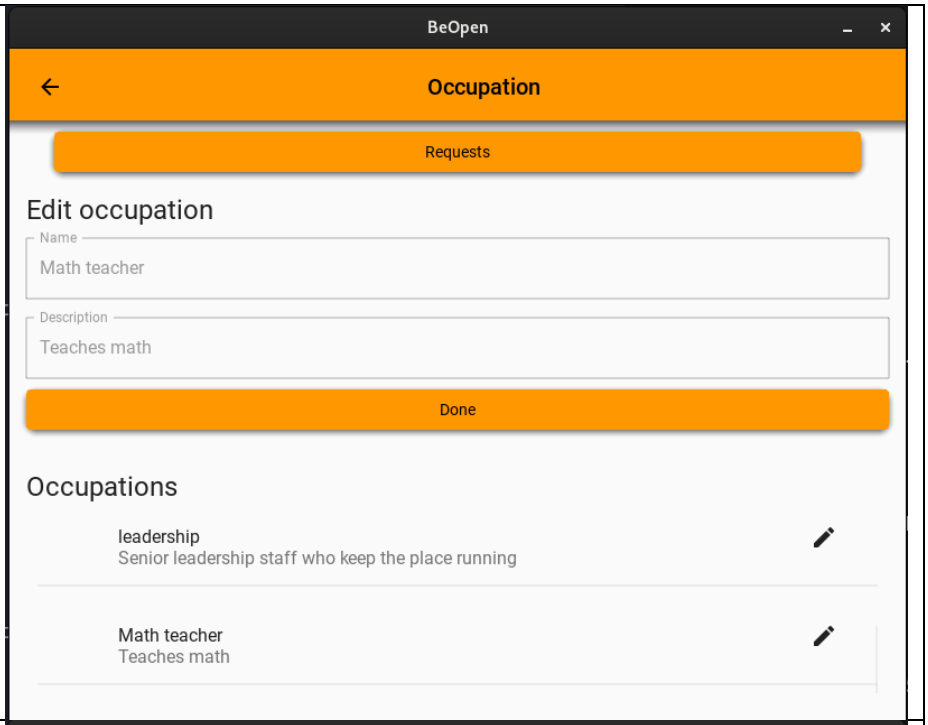
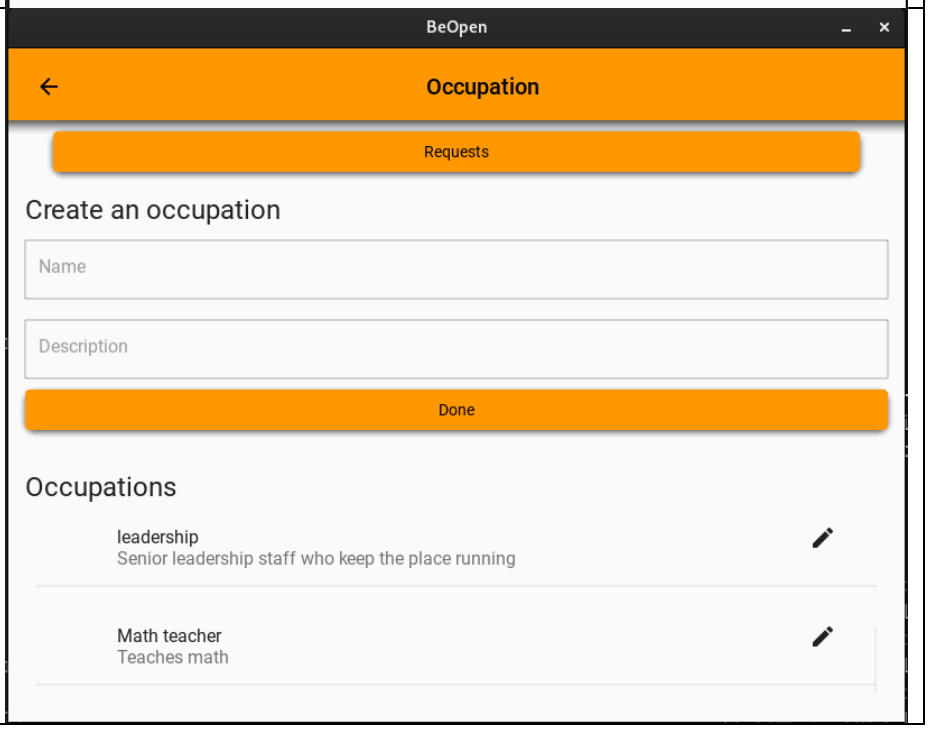
O.6B.II	The UI and look of the occupation requests page	There should be a top bar with a back button and the name of the page at the top. There should be a single scrollable list of occupation requests	As expected	
O.6C.I	Pressing the back button	This should bring us back to the occupations screen (the last screen we were on)	It brings us back to the organisation screen	Setup a back method that uses the previous pages obj to move back to it.
O.6C.II	Pressing the back button	This should bring us back to the occupations screen (the last screen we were on)	As expected	
O.7A.I	Checking to see if the occupation requests are being displayed correctly. In this instance there are no occupation change requests	The occupation requests area should simply display “no occupation change requests”	As expected	
O.7B.I	Checking to see if the occupation requests are being displayed correctly. In this instance	Each occupation requests should have an accept and reject button as well as display the username of the	The interaction buttons go off the side of the screen	The accept button will be moved to the right of the request keep the reject on the

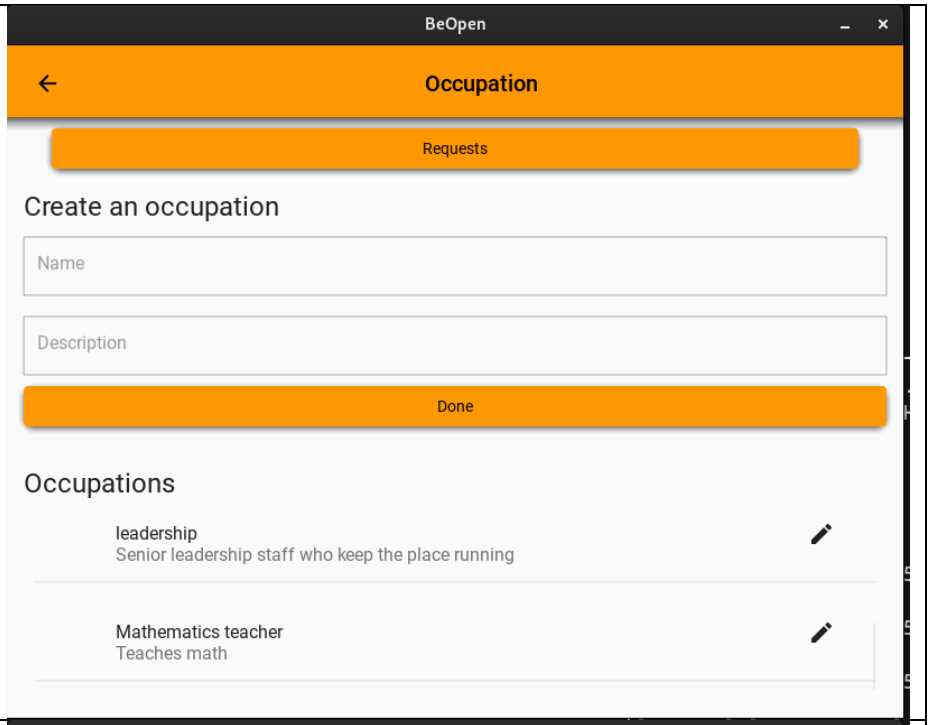
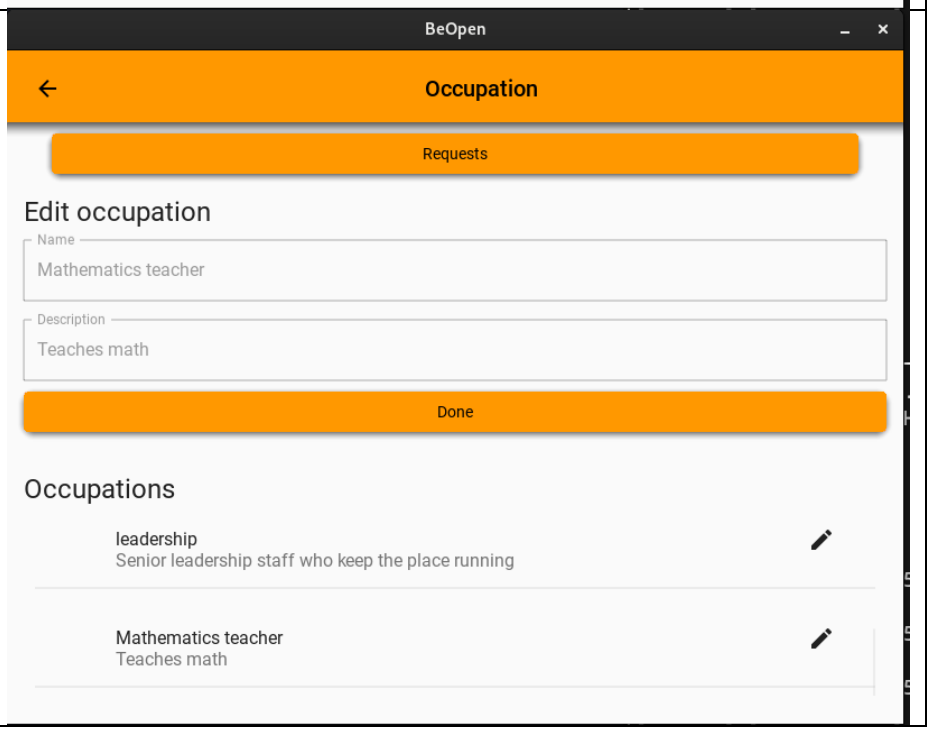
	there are 3 occupation change requests being made	person requesting the change and the occupation name they wish to change to.		right
O.7B.II	Checking to see if the occupation requests are being displayed correctly. In this instance there are 3 occupation change requests being made	Each occupation requests should have an accept and reject button as well as display the username of the person requesting the change and the occupation name they wish to change to.	As expected	
O.8A.I	Pressing the approve button (the tick) on an occupation change request	The request should disappear from the list and the user who made the request should have their occupation updated on the server side database	As expected	
O.8B.I	Pressing the reject button (the cross) on an occupation change request	The request should disappear from the list and the user who made the request should see no change to their occupation	As expected	

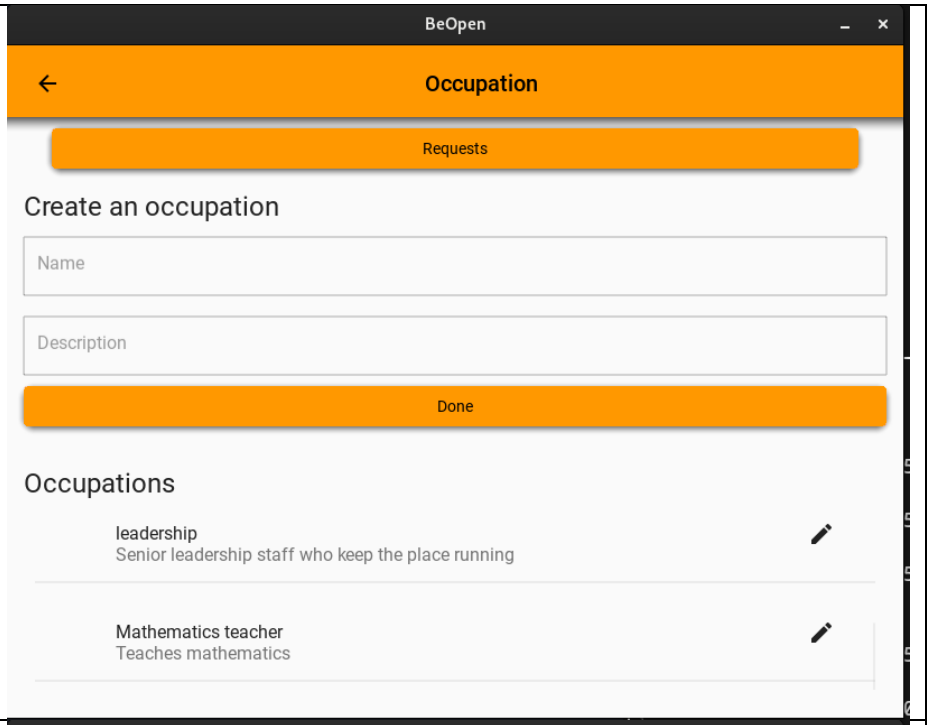
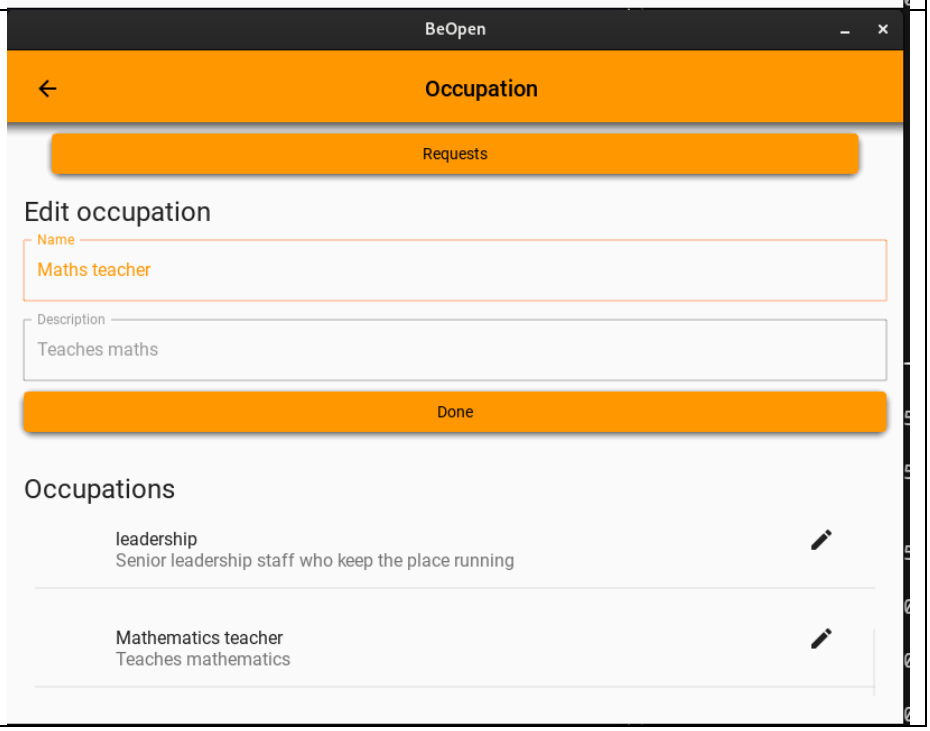
Test Number	Image Number	Image
O.1A.I	1	
O.1B.I	1	

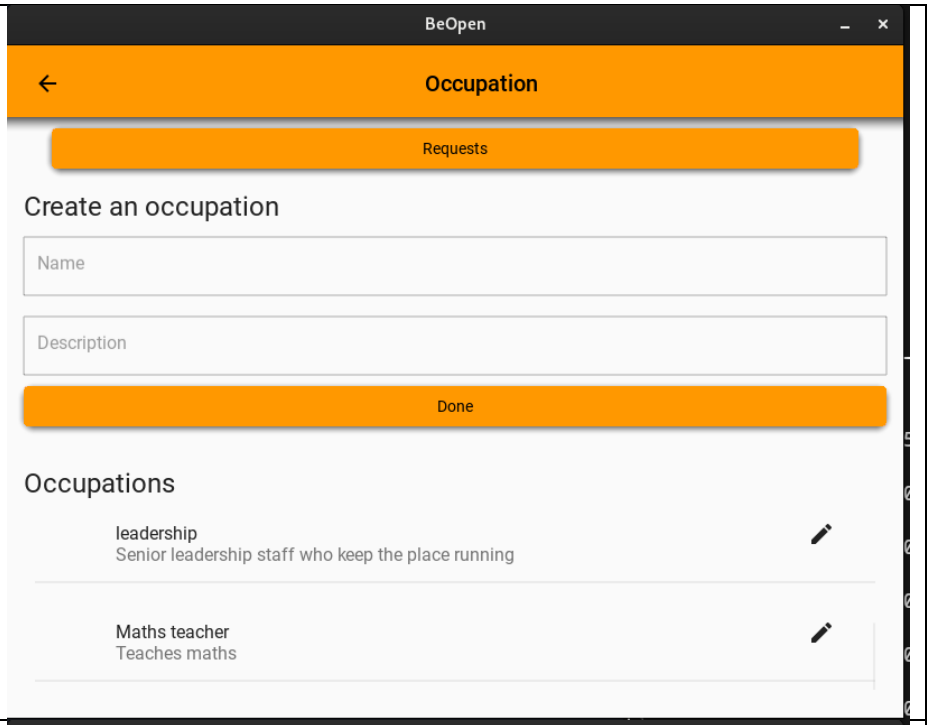
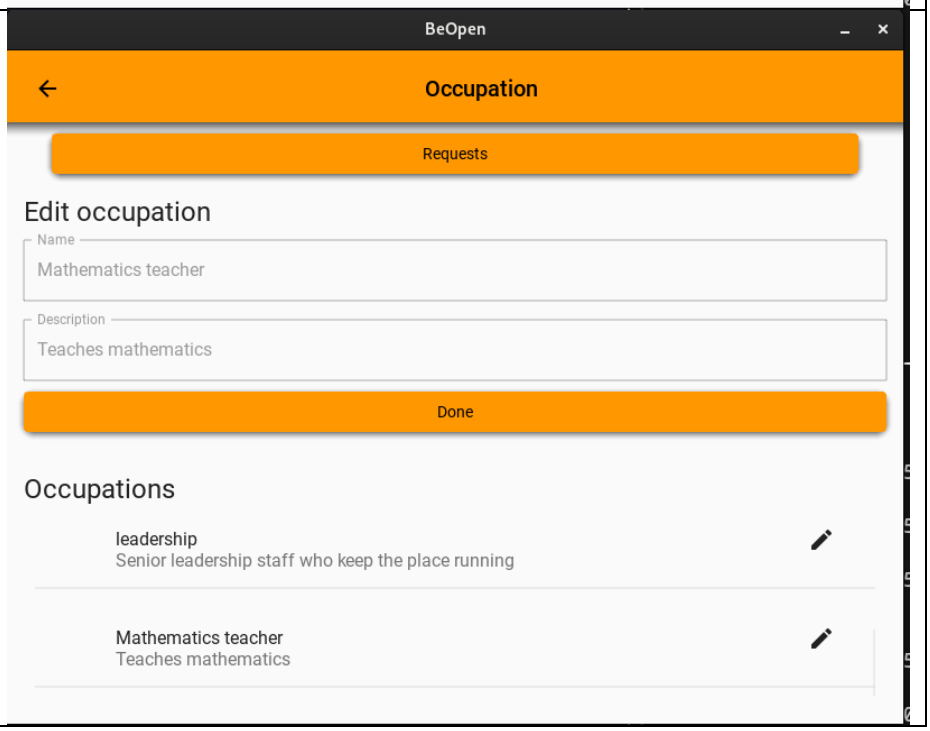
O.1B.II	1	
O.2A.I	1	

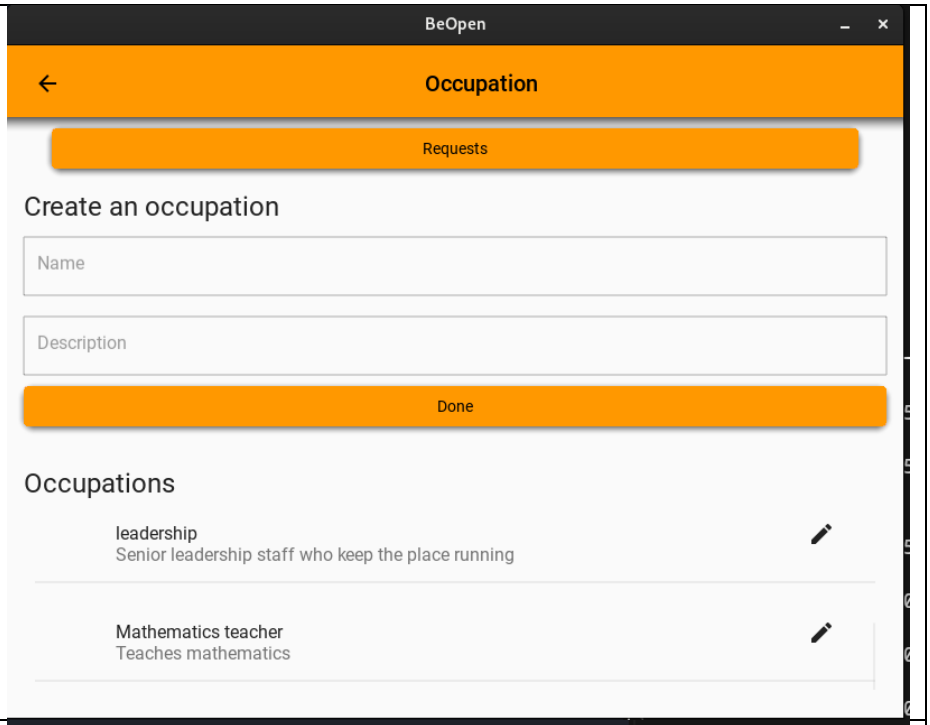
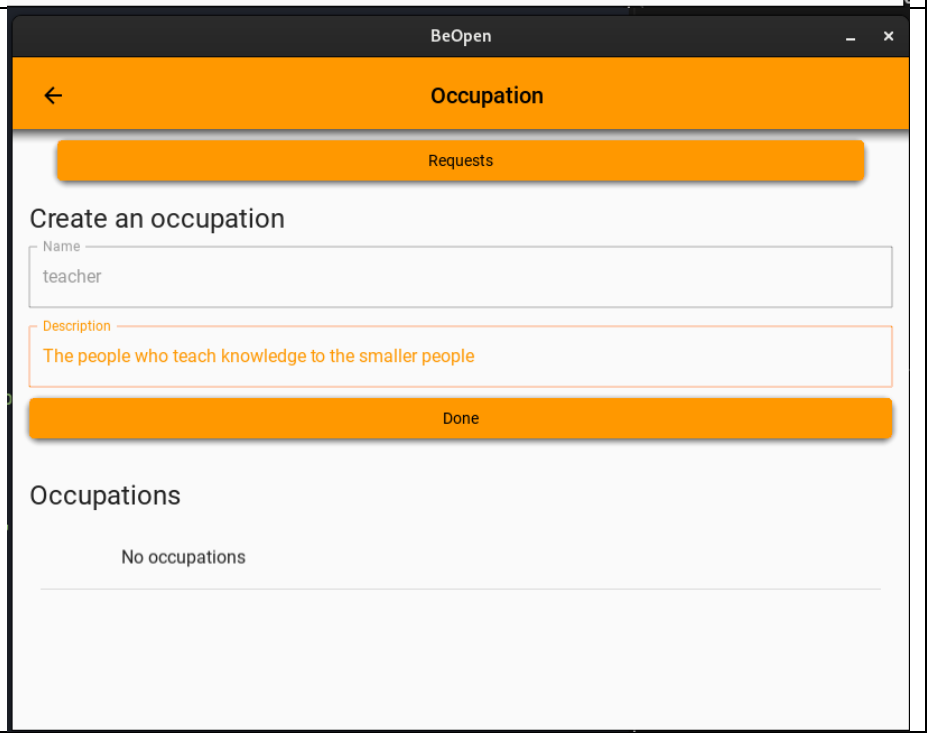
O.2B.I	1	 BeOpen Occupation Requests Create an occupation Name Description Done Occupations The people who teach knowledge to the smaller people student The people who learn from the people who teach cleaners Those who keep the learning place spick and span leadership
O.3A.I	1	 BeOpen Occupation Requests Edit occupation Name Math teacher Description Teaches math Done Occupations leadership Senior leadership staff who keep the place running Math teacher Teaches math

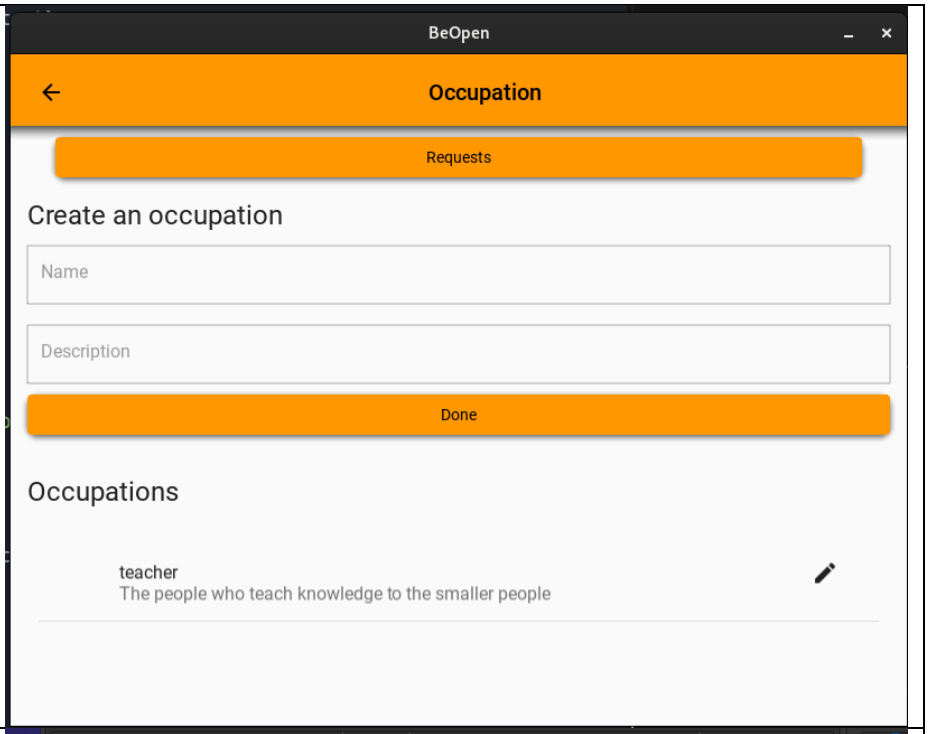
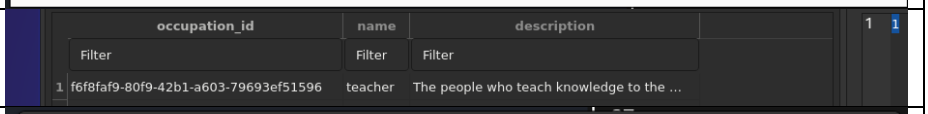
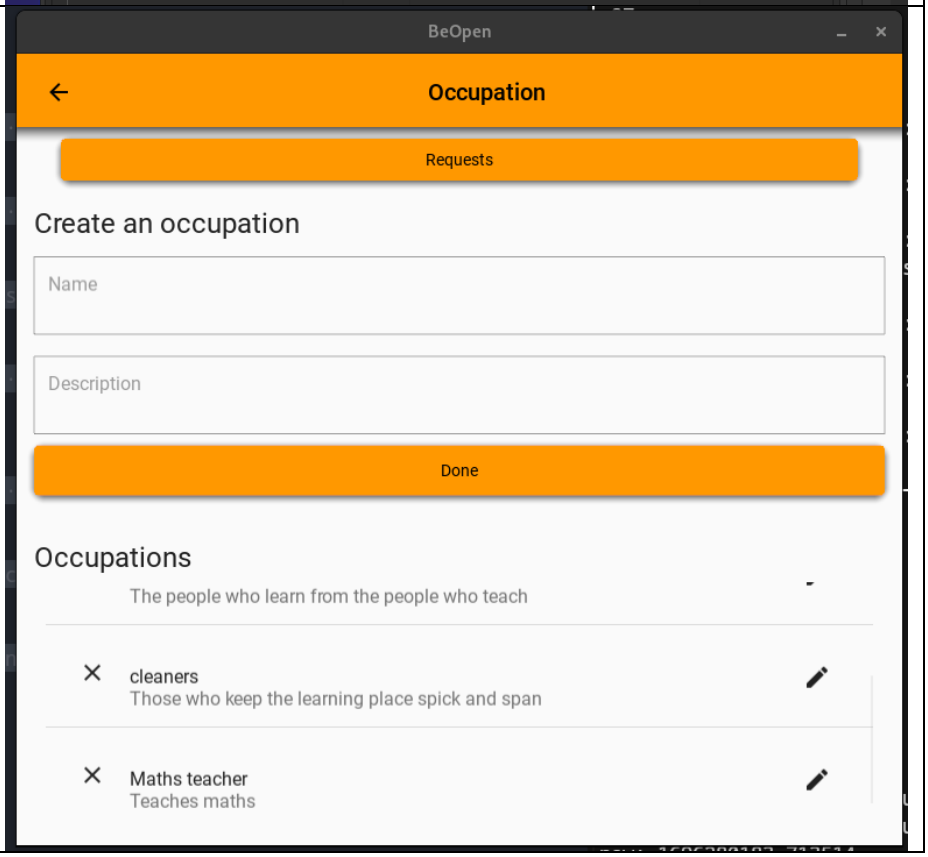
O.3B.I	1	 BeOpen Occupation Requests Edit occupation Name Math teacher Description Teaches math Done Occupations leadership Senior leadership staff who keep the place running Math teacher Teaches math
O.3B.I	2	 BeOpen Occupation Requests Create an occupation Name Description Done Occupations leadership Senior leadership staff who keep the place running Math teacher Teaches math

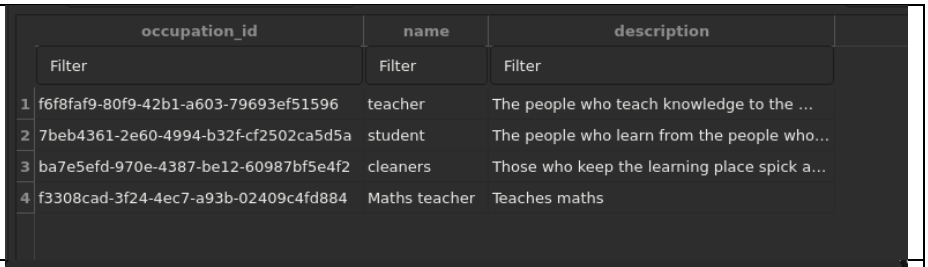
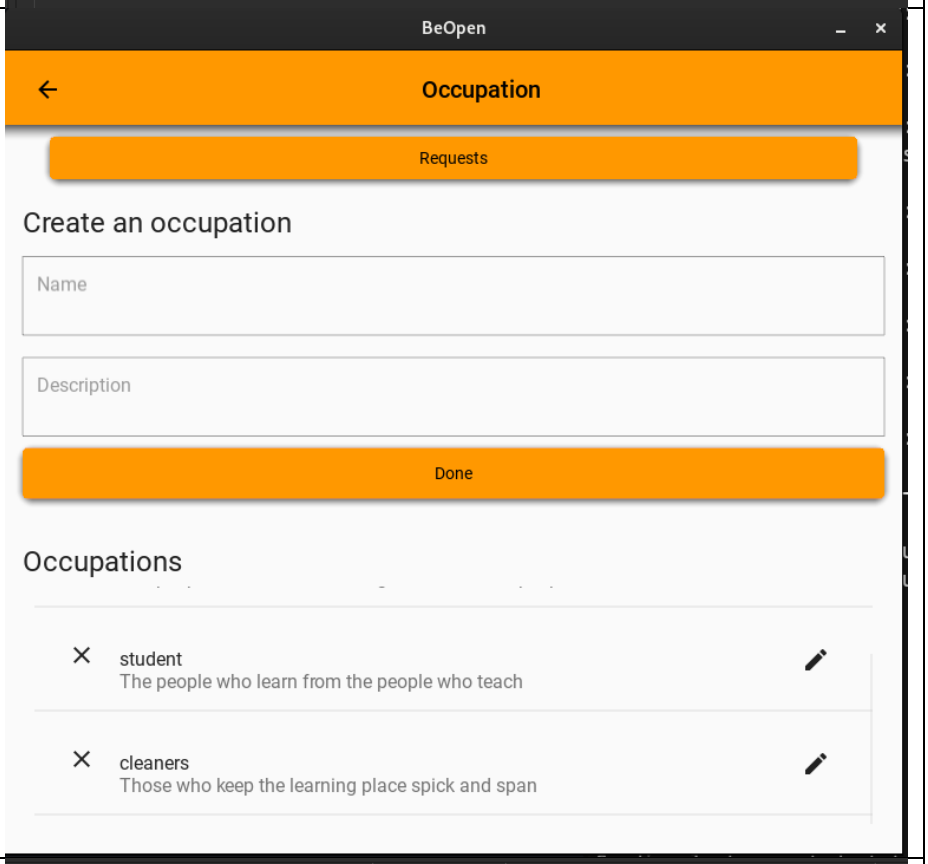
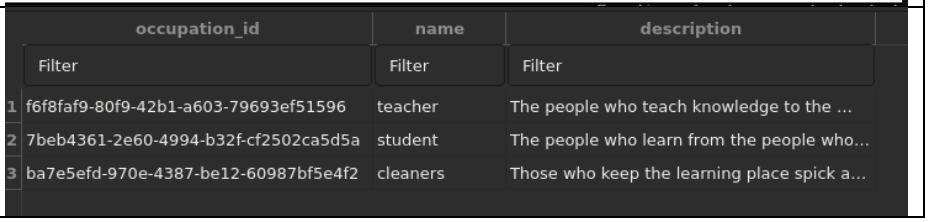
O.3B.II	1	
O.3C.I	1	

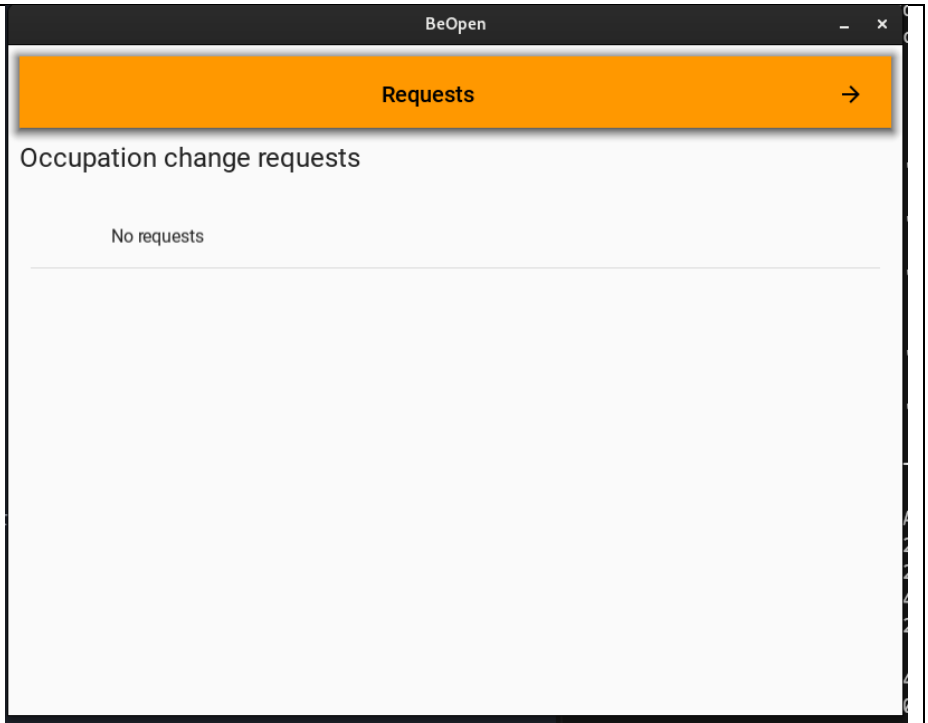
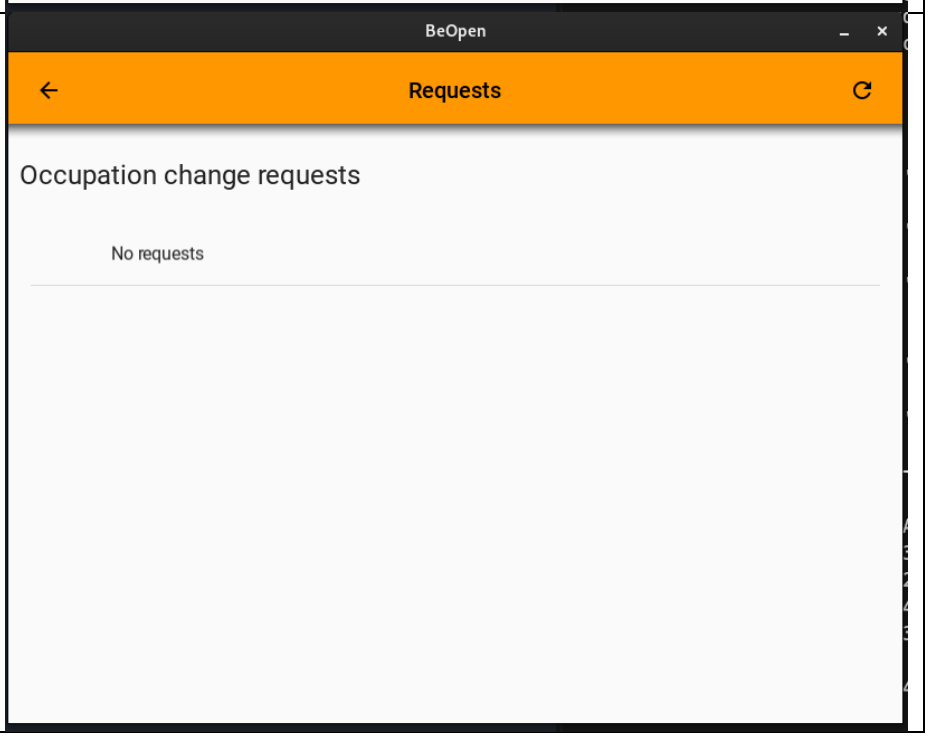
O.3C.I	2	 BeOpen Occupation Requests Create an occupation Name Description Done Occupations leadership Senior leadership staff who keep the place running Mathematics teacher Teaches mathematics
O.3D.I	1	 BeOpen Occupation Requests Edit occupation Name Maths teacher Description Teaches maths Done Occupations leadership Senior leadership staff who keep the place running Mathematics teacher Teaches mathematics

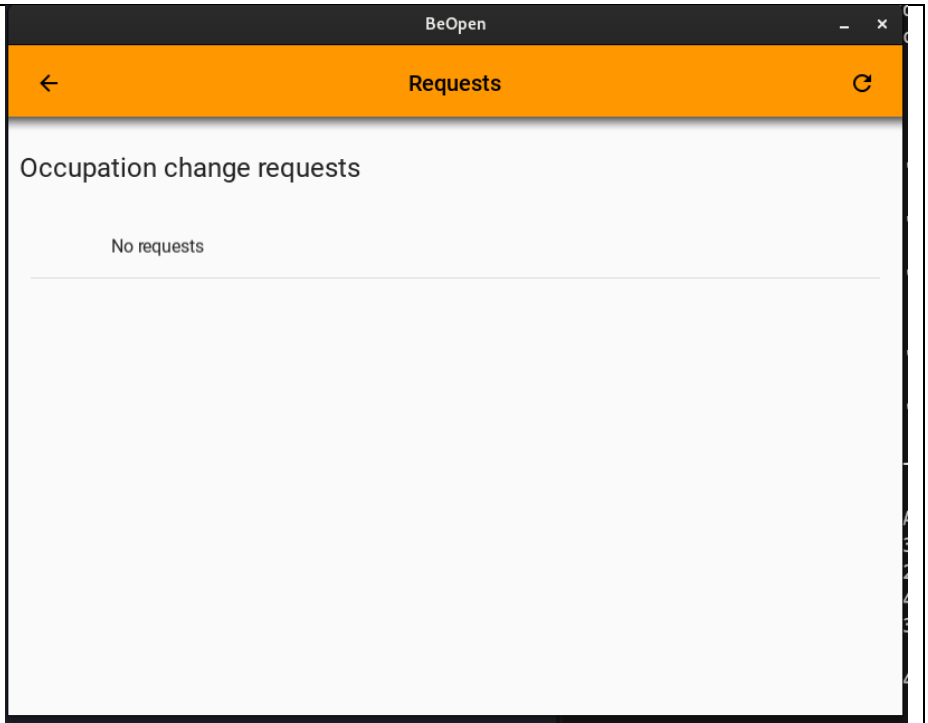
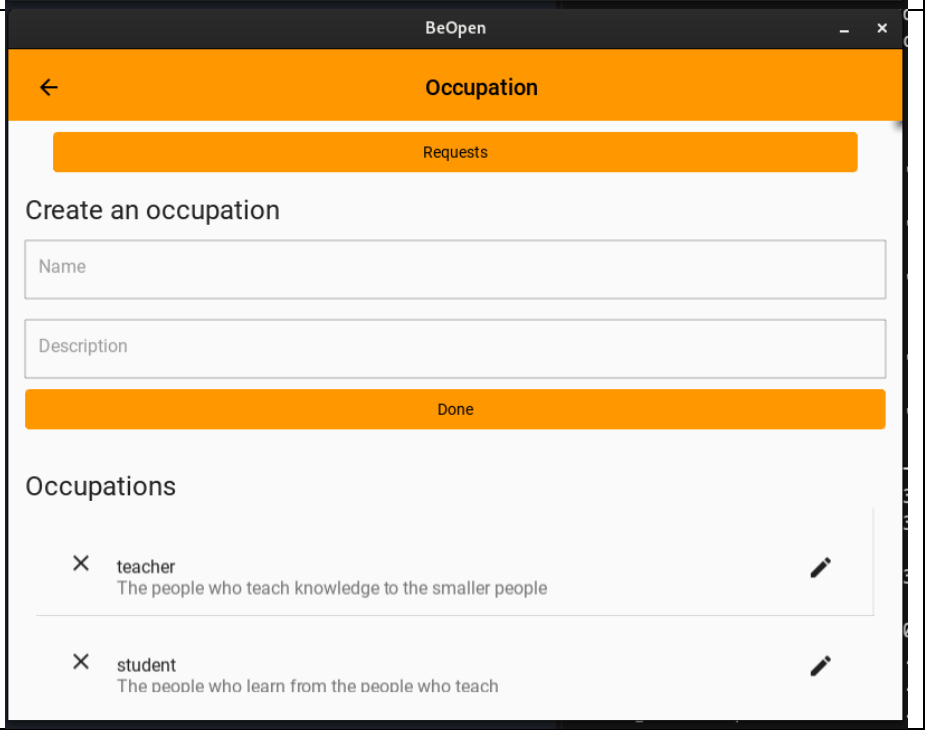
O.3D.I	2	 BeOpen Occupation Requests Create an occupation Name Description Done Occupations leadership Senior leadership staff who keep the place running Maths teacher Teaches maths
O.3E.I	1	 BeOpen Occupation Requests Edit occupation Name Mathematics teacher Description Teaches mathematics Done Occupations leadership Senior leadership staff who keep the place running Mathematics teacher Teaches mathematics

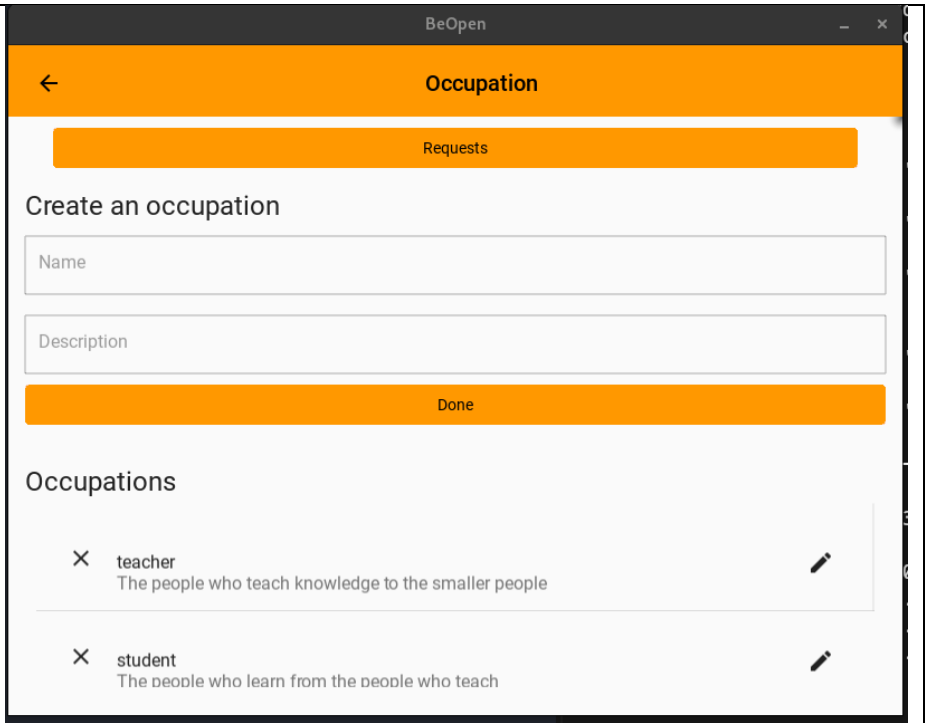
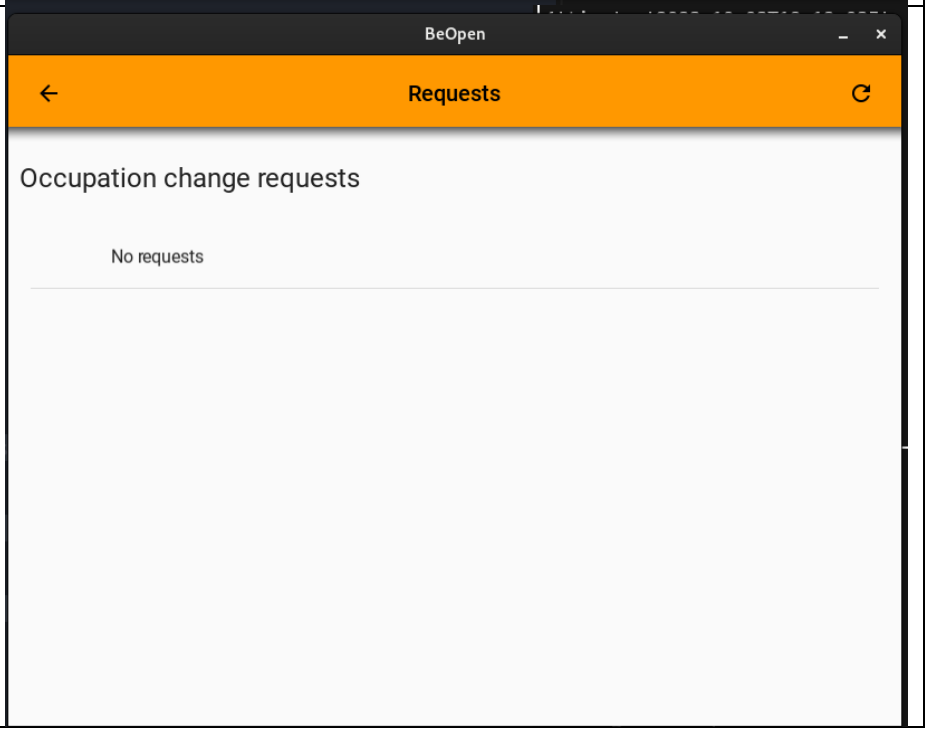
O.3E.I	2	 BeOpen Occupation Requests Create an occupation Name Description Done Occupations leadership Senior leadership staff who keep the place running Mathematics teacher Teaches mathematics
O.4A.I	1	 BeOpen Occupation Requests Create an occupation Name teacher Description The people who teach knowledge to the smaller people Done Occupations No occupations

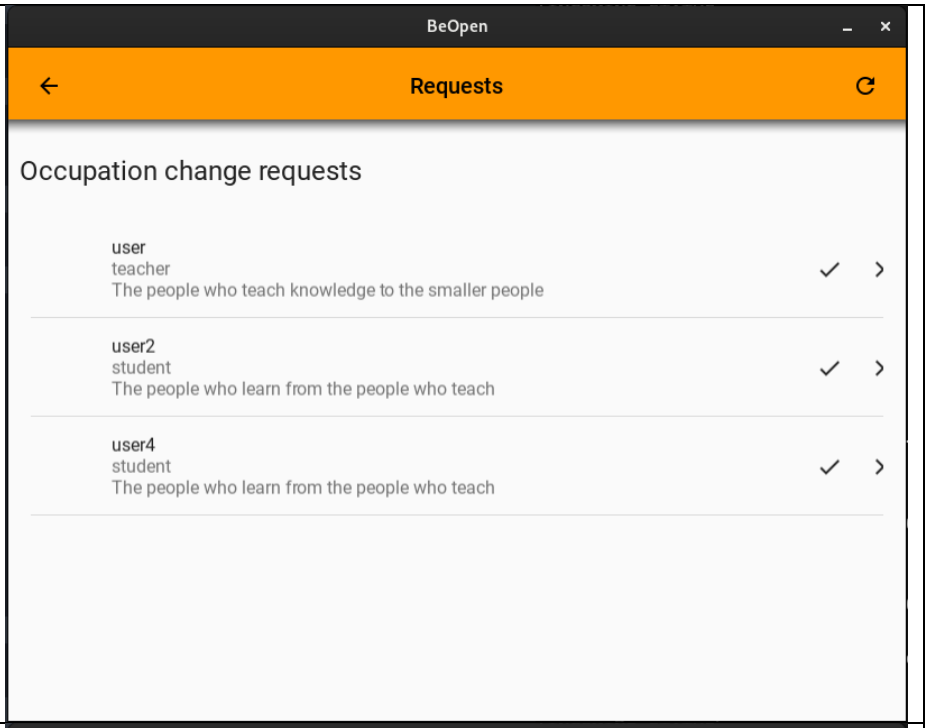
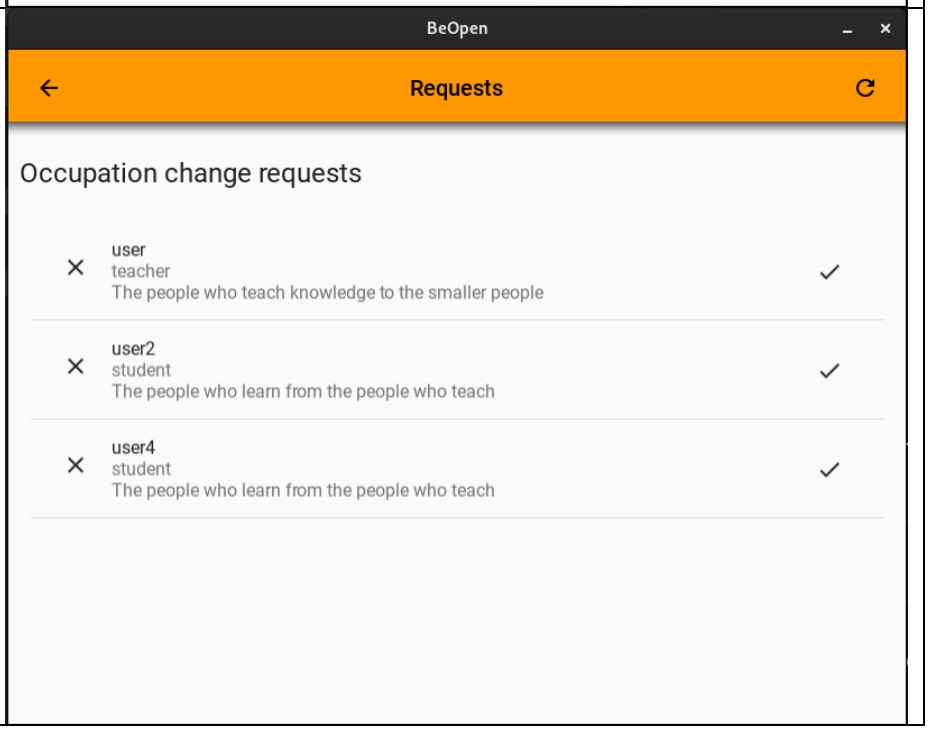
O.4A.I	2	
O.4A.I	3	
O.5A.I	1	

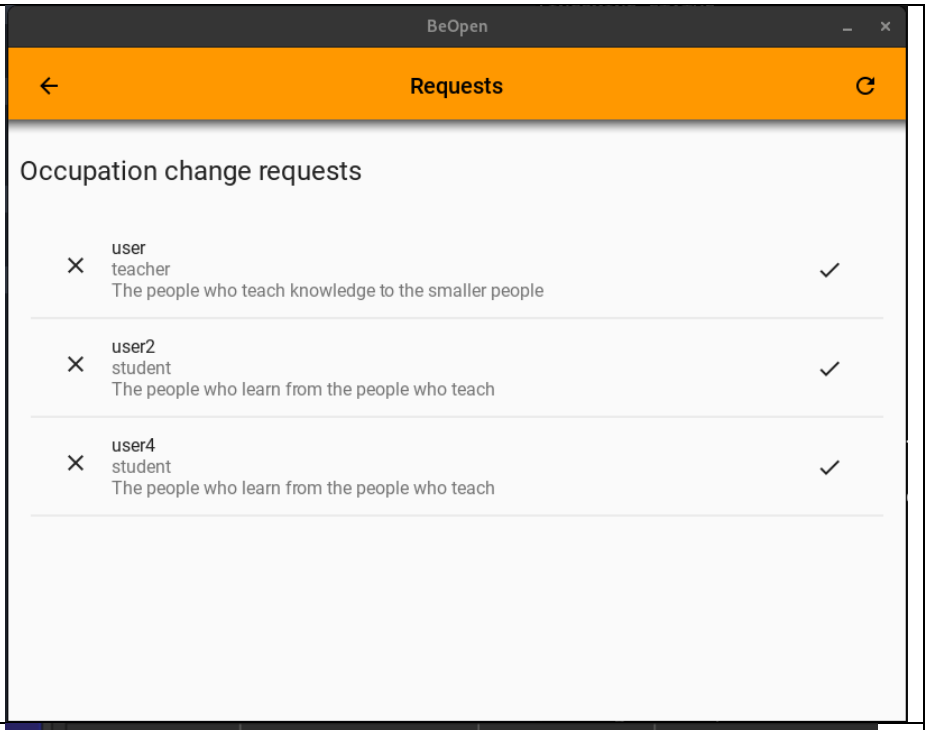
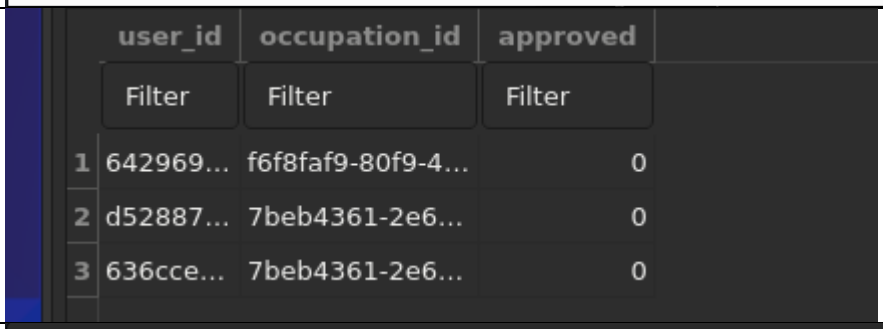
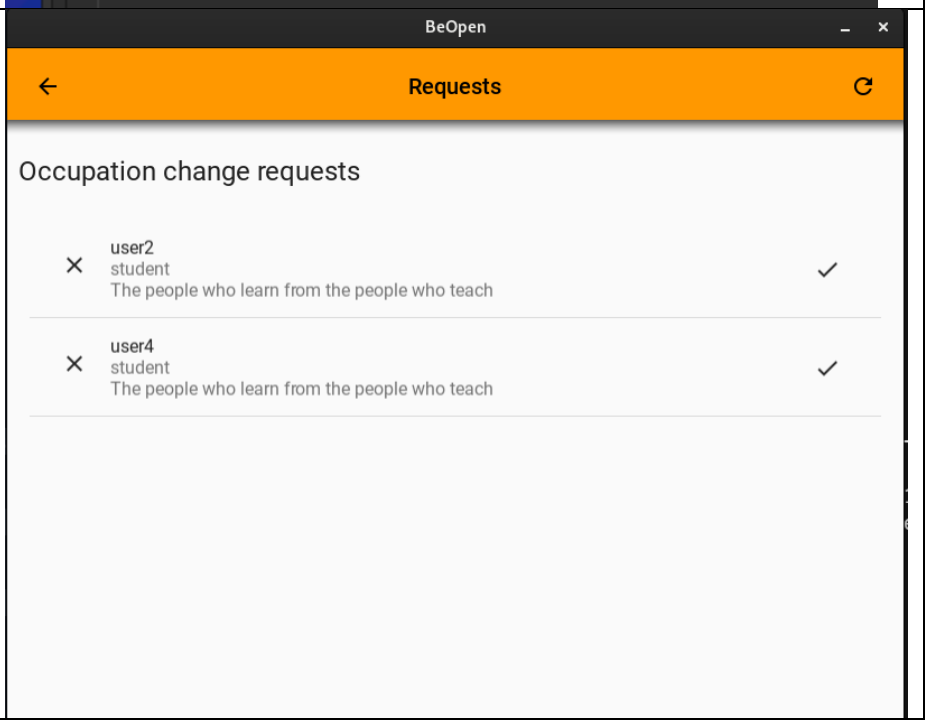
O.5A.I	2	
O.5A.I	3	
O.5A.I	4	

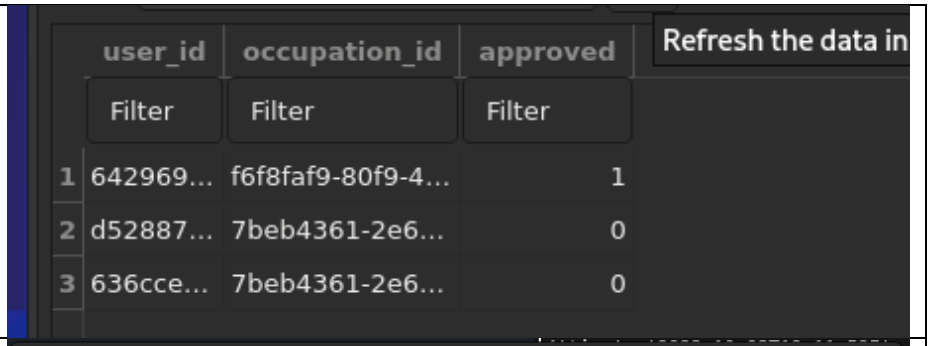
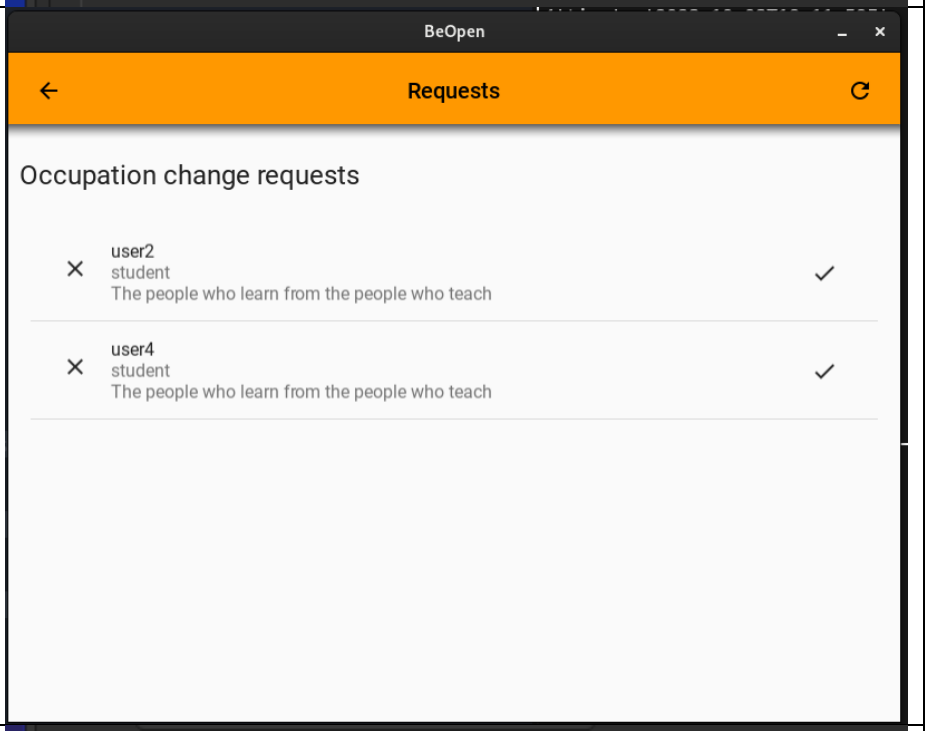
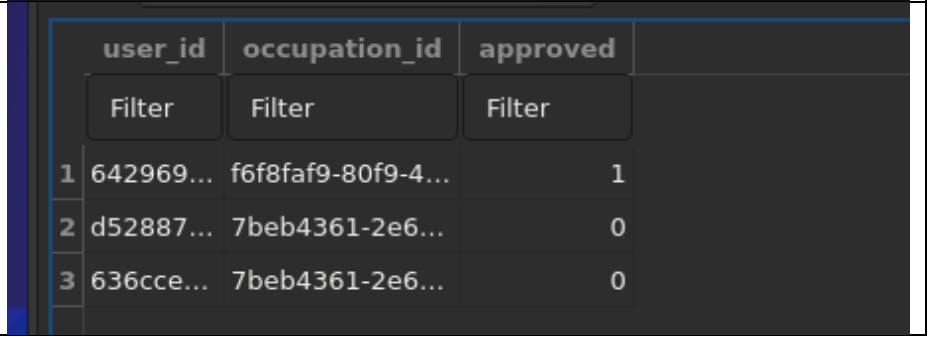
O.6A.I	1	 BeOpen Requests Occupation change requests No requests
O.6B.I	1	 BeOpen Requests Occupation change requests No requests

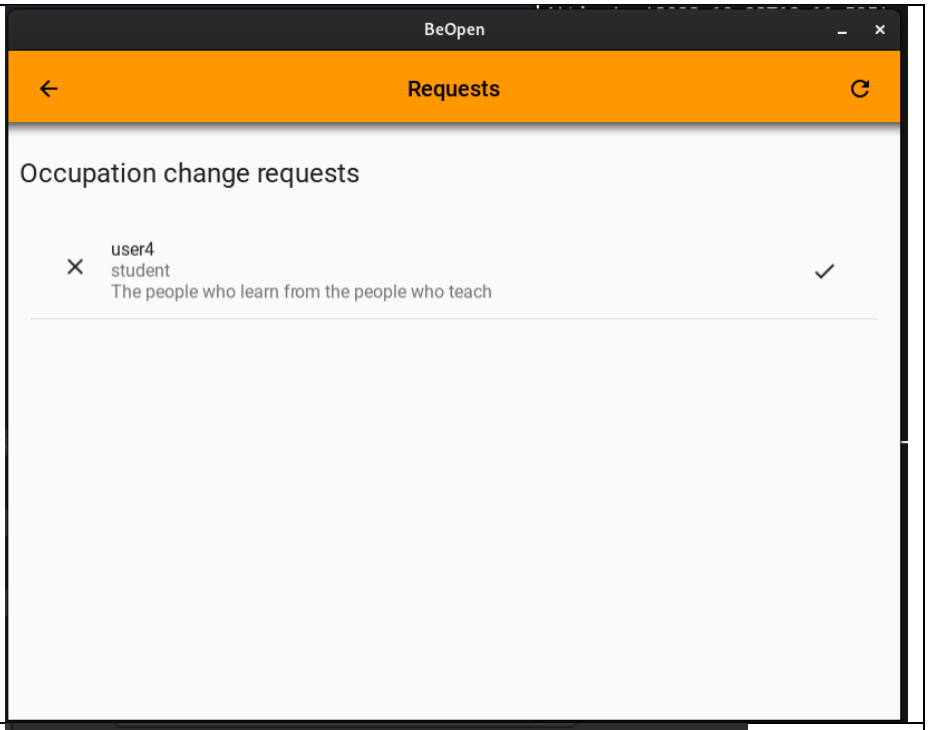
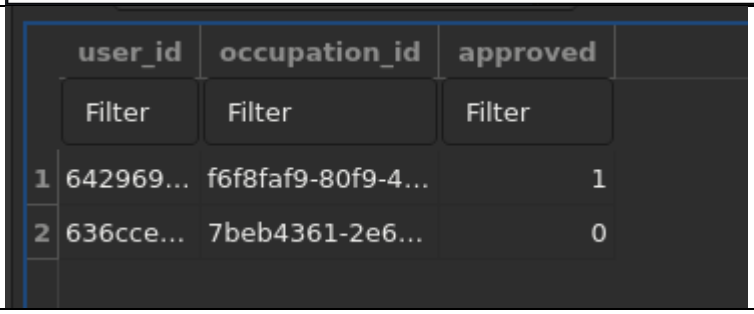
O.6B.II	1	
O.6C.I	1	

O.6C.II	1	
O.7A.I	1	

O.7B.I	1	
O.7B.II	1	

O.8A.I	1	
O.8A.I	2	
O.8A.I	3	

O.8A.I	4	
O.8B.I	1	
O.8B.I	2	

O.8B.I	3	 BeOpen Requests Occupation change requests user4 student The people who learn from the people who teach
O.8B.I	4	 user_id occupation_id approved Filter Filter Filter 1 642969... f6f8faf9-80f9-4... 1 2 636cce... 7beb4361-2e6... 0

Homepage and posts

Test Number	Test Description	Expected	Observed	Action
H.1A.I	From the login page getting to the homepage. Loading first 5 posts and displaying their content correctly	The page should load with a top bar, and in this case no posts so it should simply display “no posts” to the user	The page freezes and the server complains that the post object has no database connection attribute	The attribute is being created after its first used to set up other attributes in their setters. Moved these attributes
H.1A.II	From the login page getting to the homepage. Loading first 5 posts and displaying their content correctly	The page should load with a top bar, and in this case no posts so it should simply display “no posts” to the user	The client crashes but the server doesn’t have an error this time a string is being passed inplace of a dictiontary to the post swiper	The client was calling post_get which only returns a single post when you supply a post_id. The server has methods for getting friend posts and team posts but nothing so the server can construct the whole feed for the client. So I created a new event that post_get_feed that combines friend posts and team posts.
H.1A.III	From the login page getting to the homepage. Loading first 5	The page should load with a top bar, and in this case no posts so it should simply	Again the client crashes and the server complains that a Nonetype is	Proper checking on the get_friends and get_team methods as well as additional

	posts and displaying their content correctly	display “no posts” to the user	not subscribtable. This was because lack of checking for if the user is in a team or not and/or had any friends.	checks for get_feed to handle no posts being returned
H.1A.III	From the login page getting to the homepage. Loading first 5 posts and displaying their content correctly	The page should load with a top bar, and in this case no posts so it should simply display “no posts” to the user	As expected	
H.1B.I	From the login page getting to the homepage. Loading first 5 posts and displaying their content correctly	The page should load with a top bar, and in this case there are 5 posts on the server meant for this user. The client should first only load 5 posts, then only load the next 4 once the “load more” button is clicked.	On login the client crashes. The post_id is not being correclty passed to the comments section of a post.	Actually comes down to a server side error on how it was passing data between class methods. Now fixed
H.1B.II	From the login page getting to the homepage. Loading first 5 posts and displaying	The page should load with a top bar, and in this case there are 5 posts on the server meant for this user.	The posts are not displaying correctly just showing white space. The caption isnt being set correctly still	The client was not calling the load_content function would uses the fetched information to display it on each post.

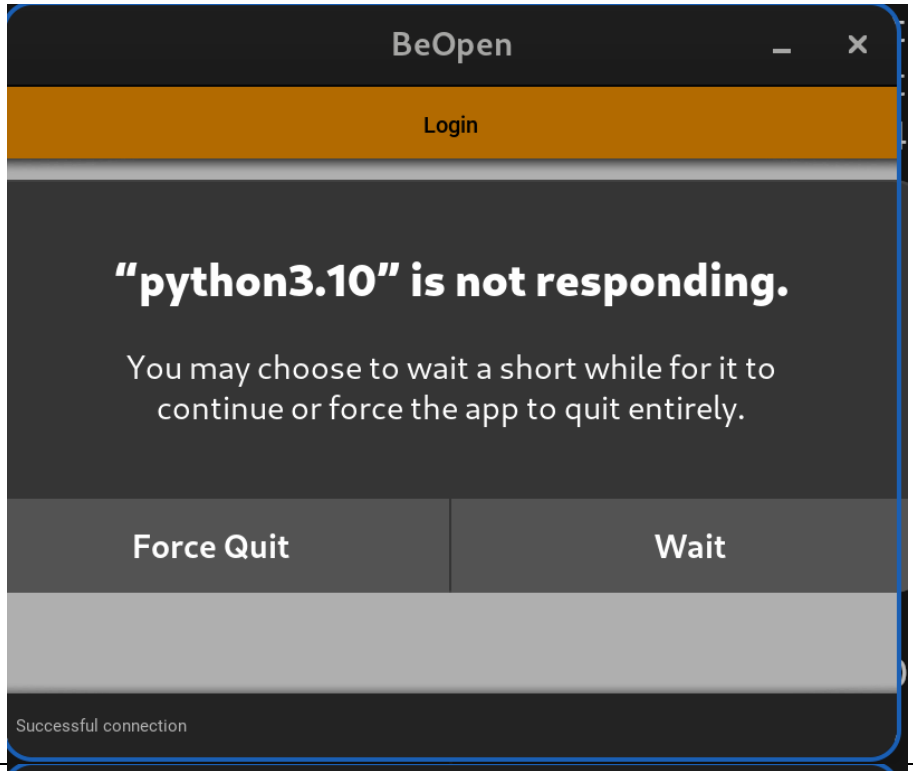
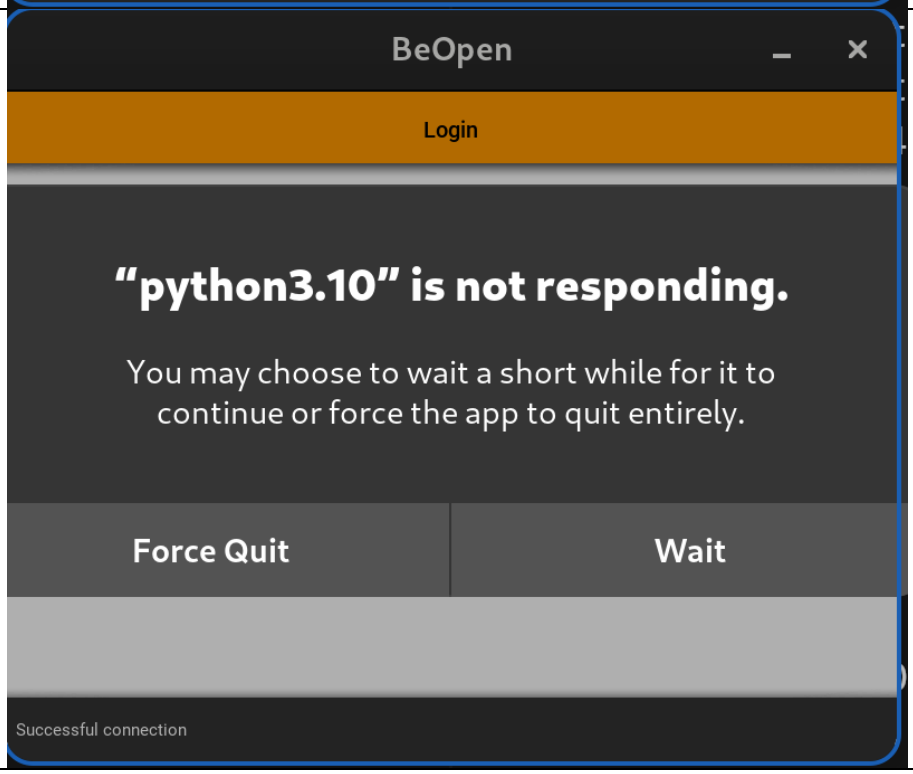
	their content correctly	The client should first only load 5 posts, then only load the next 4 once the “load more” button is clicked.	displaying the defualt text.	
H.1B.III	From the login page getting to the homepage. Loading first 5 posts and displaying their content correctly	The page should load with a top bar, and in this case there are 5 posts on the server meant for this user. The client should first only load 5 posts, then only load the next 4 once the “load more” button is clicked.	Client cant iterate over nonetype. As in the varaible post_likes is supposed to be a list but is coming up as a nonetype	The client does not anticipate a post to have no impressions on it. This is now fixed
H.1B.IV	From the login page getting to the homepage. Loading first 5 posts and displaying their content correctly	The page should load with a top bar, and in this case there are 5 posts on the server meant for this user. The client should first only load 5 posts, then only load the next 4 once the “load more” button is clicked.	The client only seams to be showing 3 posts, however when clicking load more it does load the remaining 2 posts one by one.	This is due to the list being use for the for loop being the same list that is being reduced by the for loop using .pop() Simply removing this .pop() fixes the issue as reducing the list as it is looped through is no longer needed.

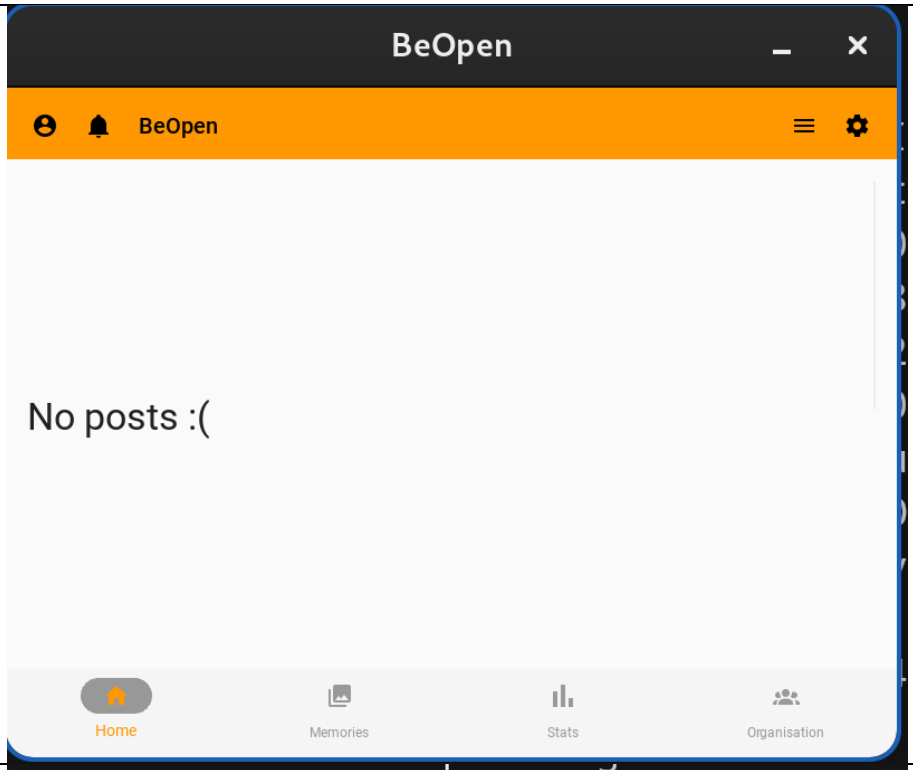
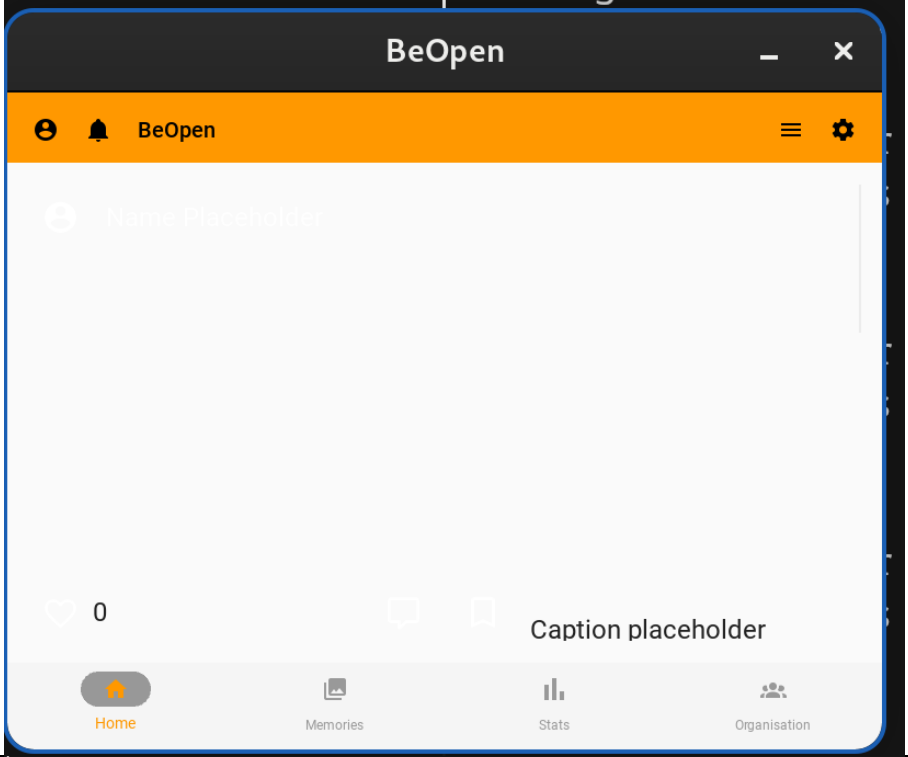
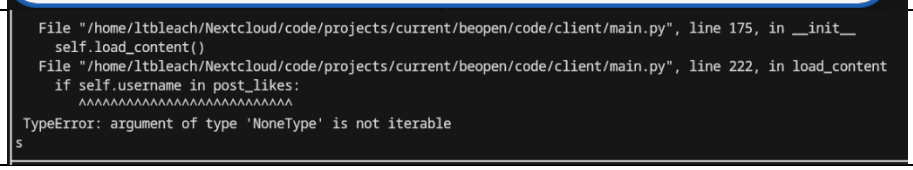
H.1C.I	From the login page have the homepage load the first 5 posts, one of them has been liked	The login page should switch to the homepage with 5 posts loaded the post from “user9” has a like from us on it already and so should have a full heart like icon	The post loads with the corret like count but the heart button isnt illuminated.	The username comparision in the HomeSwiper object was using the posters username and not the clients username.
H.1C.II	From the login page have the homepage load the first 5 posts, one of them has been liked	The login page should switch to the homepage with 5 posts loaded the post from “user9” has a like from us on it already and so should have a full heart like icon	As expected	
H.1D.I	Loading a set of posts of which all have long captions, the server imposes a character limit of 100	The captions should be displayed in a readable and resizable way	As expected	
H.2A.I	Clicking the “load more” button when there are no posts present.	It should simply set the scroll back to the top and keep the single label “No posts”	It creates another “No posts” label when clicked and sets the scroll of the screen to this new label.	Have the button check for new posts. And if posts/text already exist to consider every possible case. Additionally fixed issue with the number of posts displayed at once

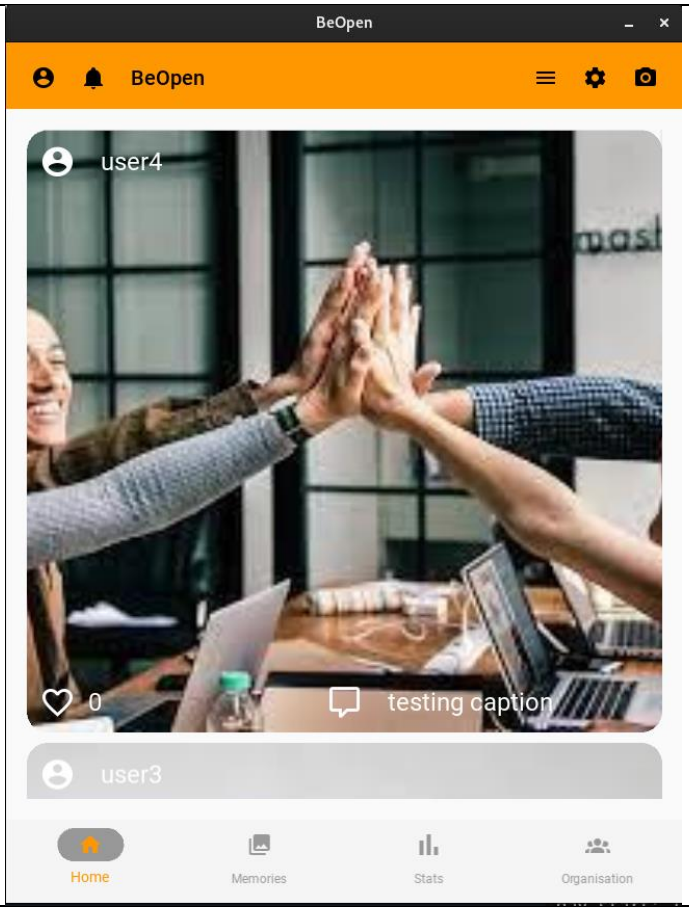
H.2A.II	Clicking the “load more” button when there are no posts present.	It should simply set the scroll back to the top and keep the single label “No posts”	As expected	
H.2B.I	Clicking the load more button when all available posts are already displayed. For this test the user has only 5 posts that should be displayed, they are already loaded.	On clicking the button a message should appear stating that all available posts are already loaded. Something like “Sorry, no more posts”	The warning message about there being no posts doesn’t appear	The message is created but forgot to add a condition to display the message so added an if statement to check for if thre are any more posts if not it will display the message.
H.2B.II	Clicking the load more button when all available posts are already displayed. For this test the user has only 5 posts that should be displayed, they are already loaded.	On clicking the button a message should appear stating that all available posts are already loaded. Something like “Sorry, no more posts”	As expected	
H.2C.I	Clicking the load more button when there are 6	The load more button should load just 5 more	As expected	

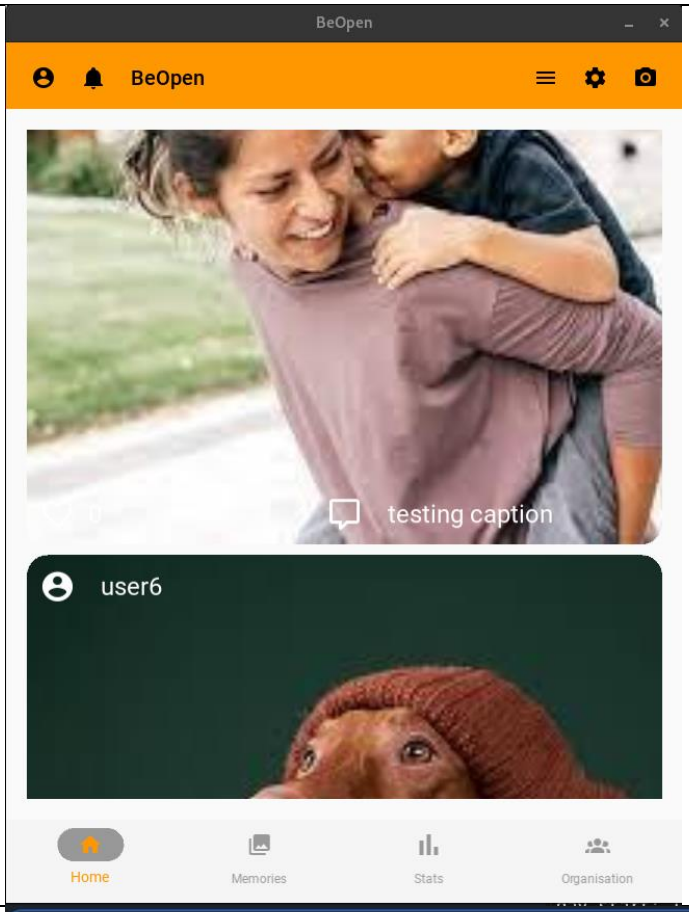
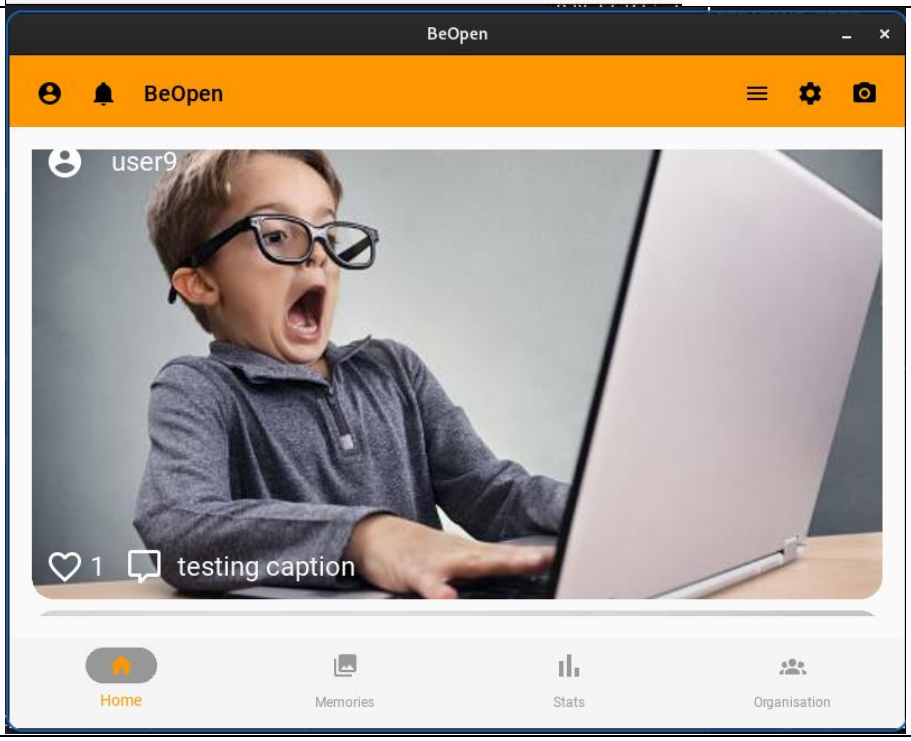
	more posts for the user to see.	posts, and then stop.		
H.2D.I	Clicking the load more button when there is 1 more posts for the user to see.	The load more button should load the last post and set the scroll of the page to this post.	As expected	
H.3A.I	Clicking the like button on a post	The heart should turn to be full, instead of just the outline and the server should save the impression on its database. The like count should update	As expected	
H.3B.I	Clicking the un-like button on a previously liked post	The heart should turn back to an outline the like count should decrease by 1 and the relevant impression should be removed from the post_impressions table on the server.	Its seamingly working in the moment but there is no change happening on the server side	The client is passing the wrong information for impression_delete which requires a impression_id
H.3B.II	Clicking the un-like button on a previously liked post	The heart should turn back to an outline the like count should decrease by 1	As expected	

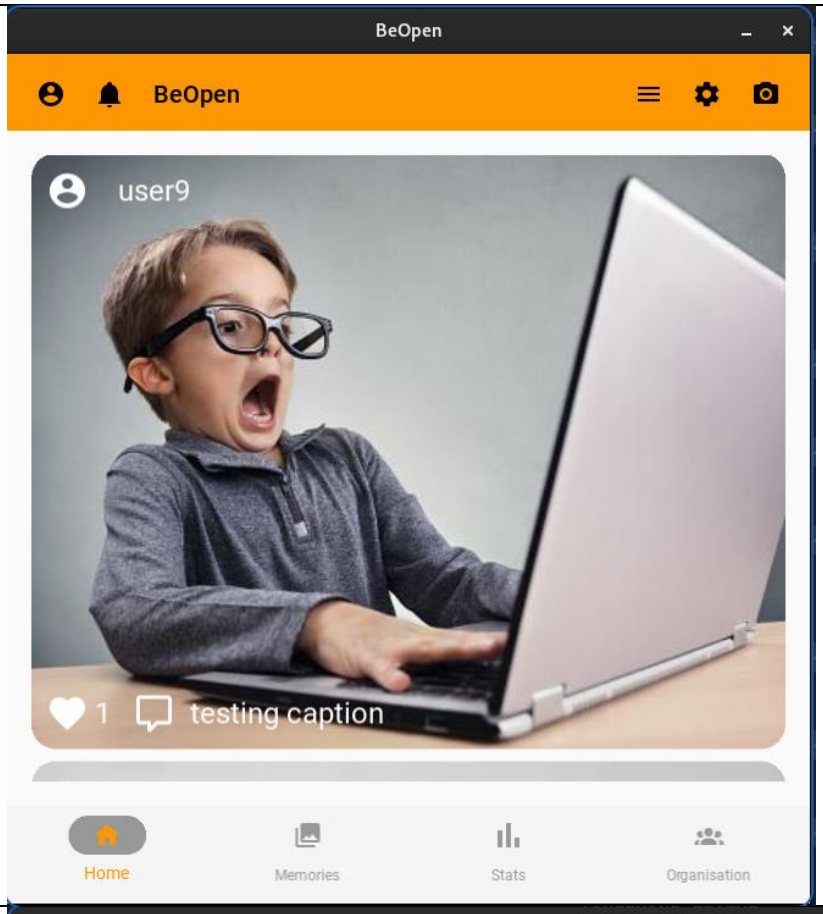
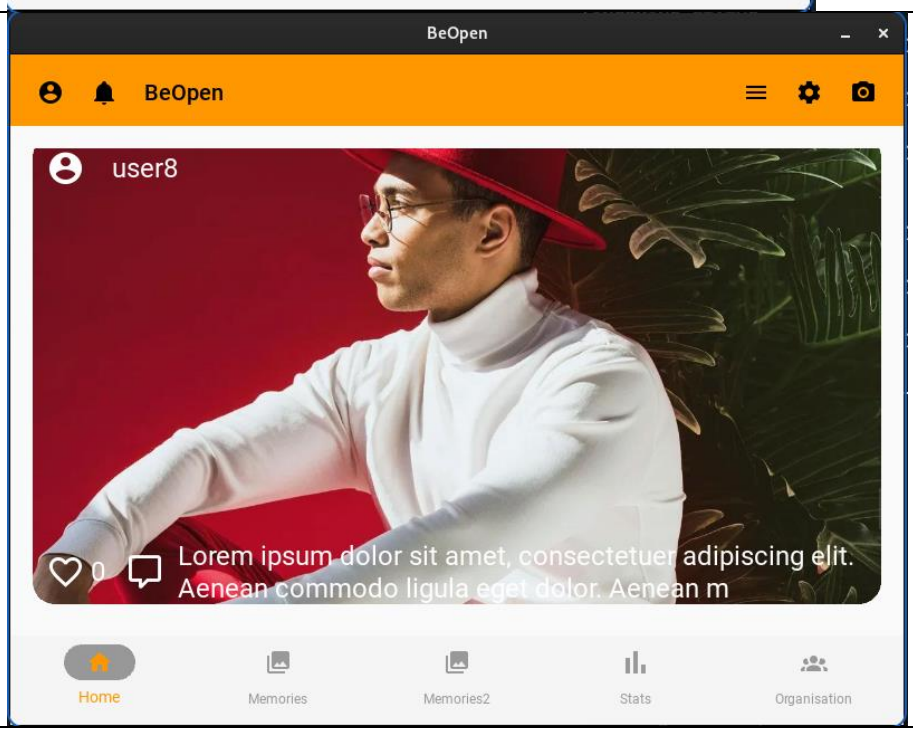
		and the relevant impression should be removed from the post_impressions table on the server.		
H.4A.I	Clicking the profile button as an admin an admin and clicking delete	This should remove the post from the feed and the change should be reflected on the server database	As expected	
H.4B.I	Clicking the profile button logged in as an Admin. Then clicking profile	This should take the user to the the posters profile, in this case its an admin so they should have edit buttons next to role, name, biography,	As expected	
H.4C.I	Clicking the profile button logged in a “user” who is a normal member level user	On clikcing the button it should just take you straight to the profile page of the poster	As expected	

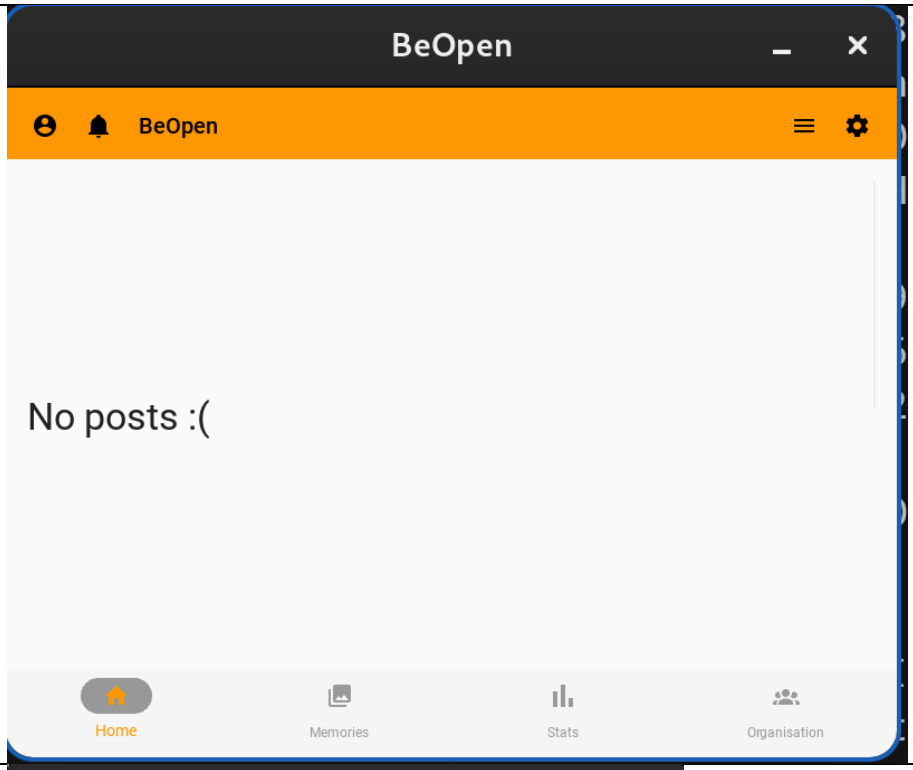
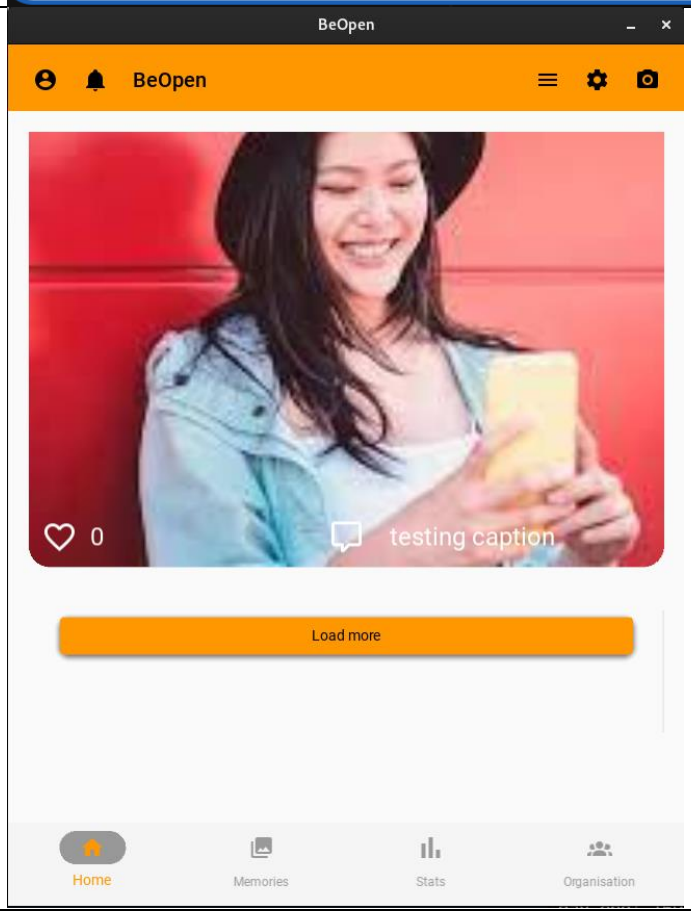
Test Number	Image Number	Image
H.1A.I	1	
H.1A.II	1	

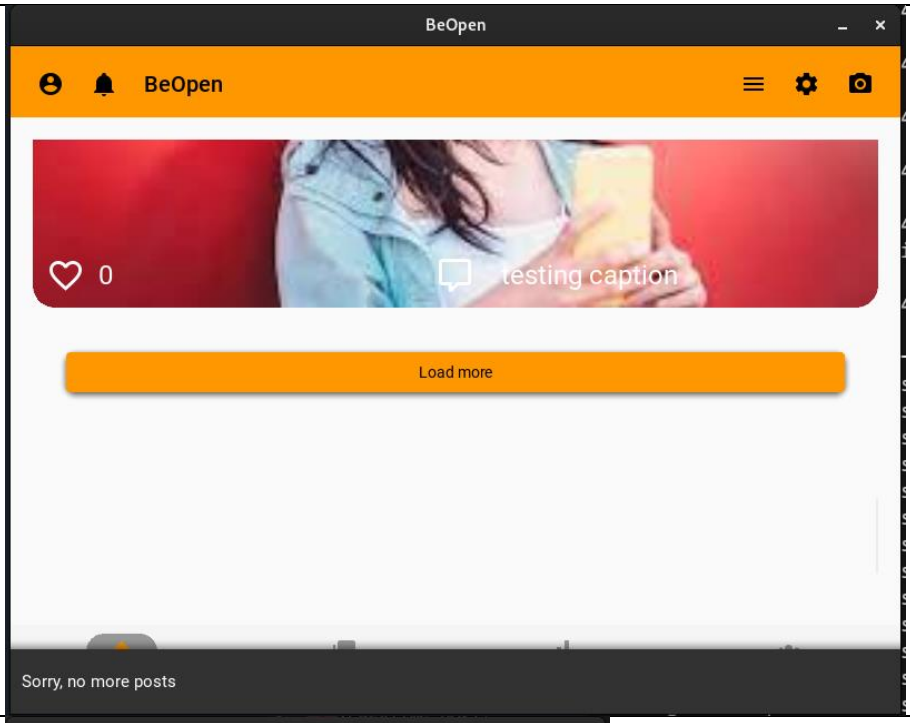
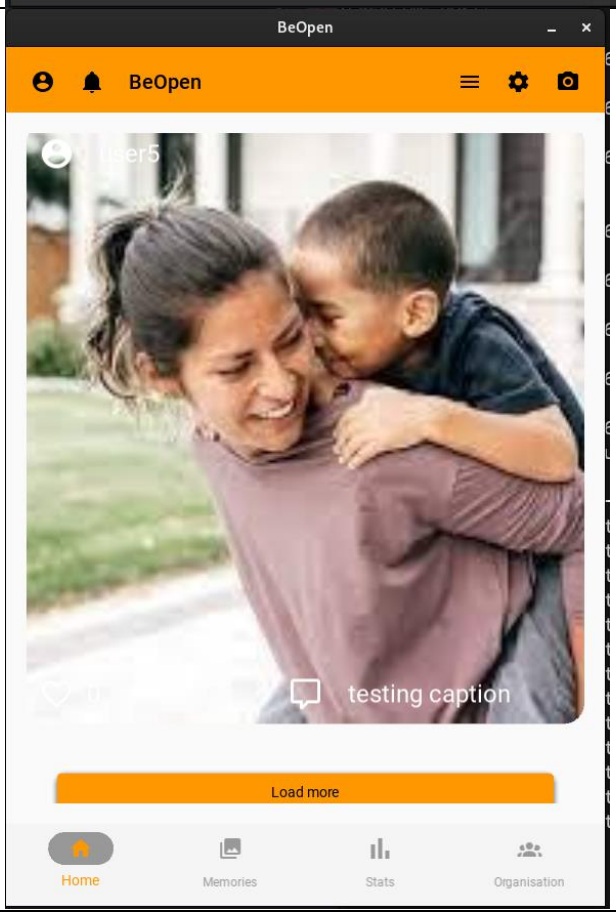
H.1A.III	1	
H.1B.II	1	
H.1B.III	1	

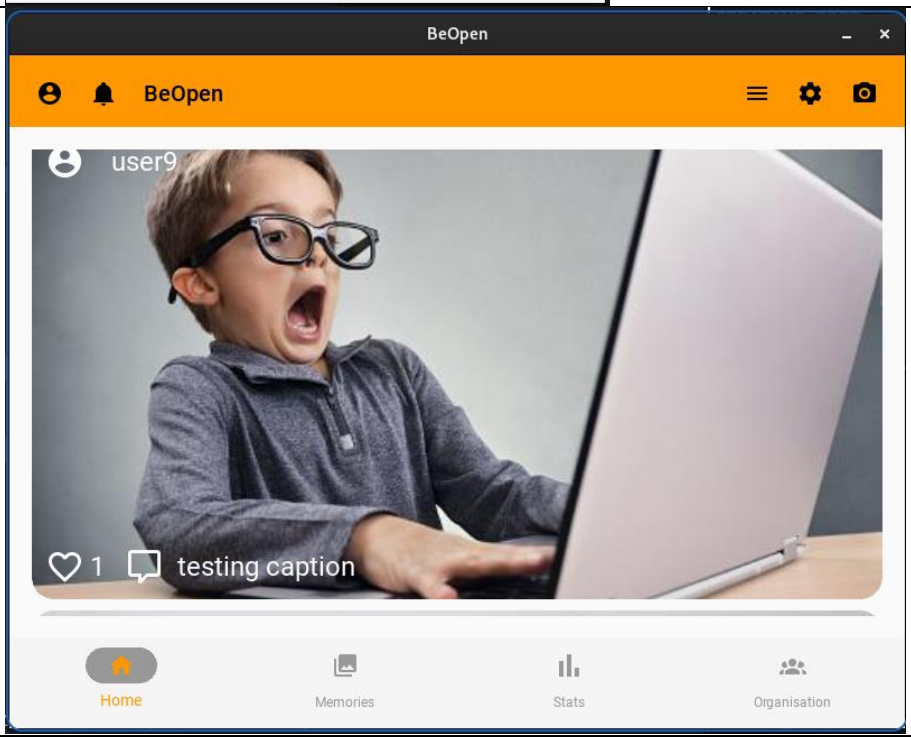
H.1B.IV	1	 BeOpen BeOpen user4 0 testing caption user3 Home Memories Stats Organisation

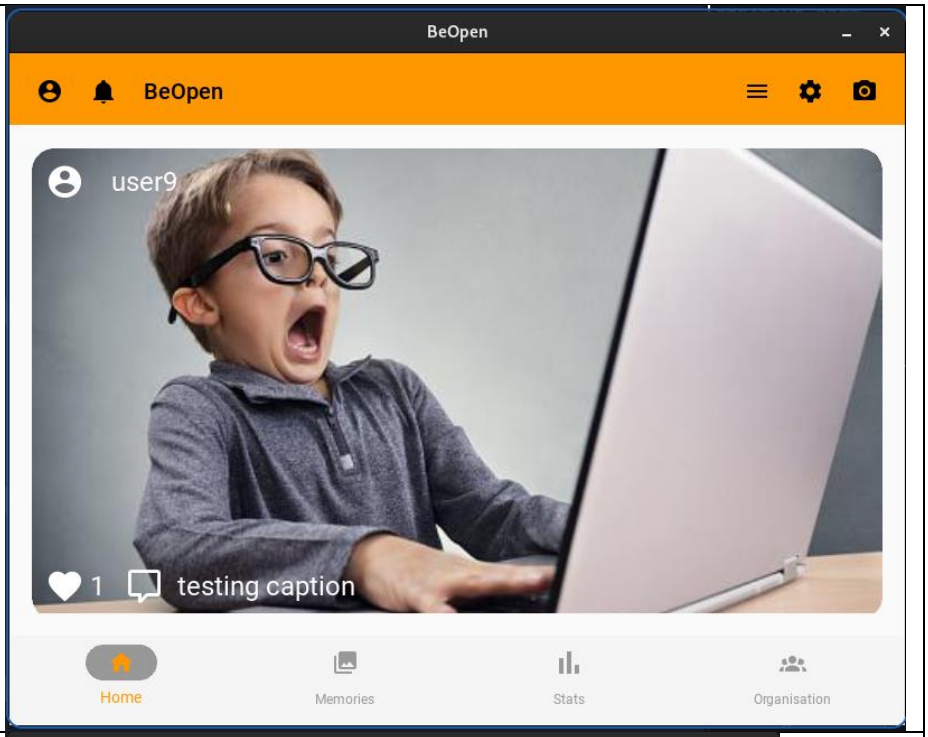
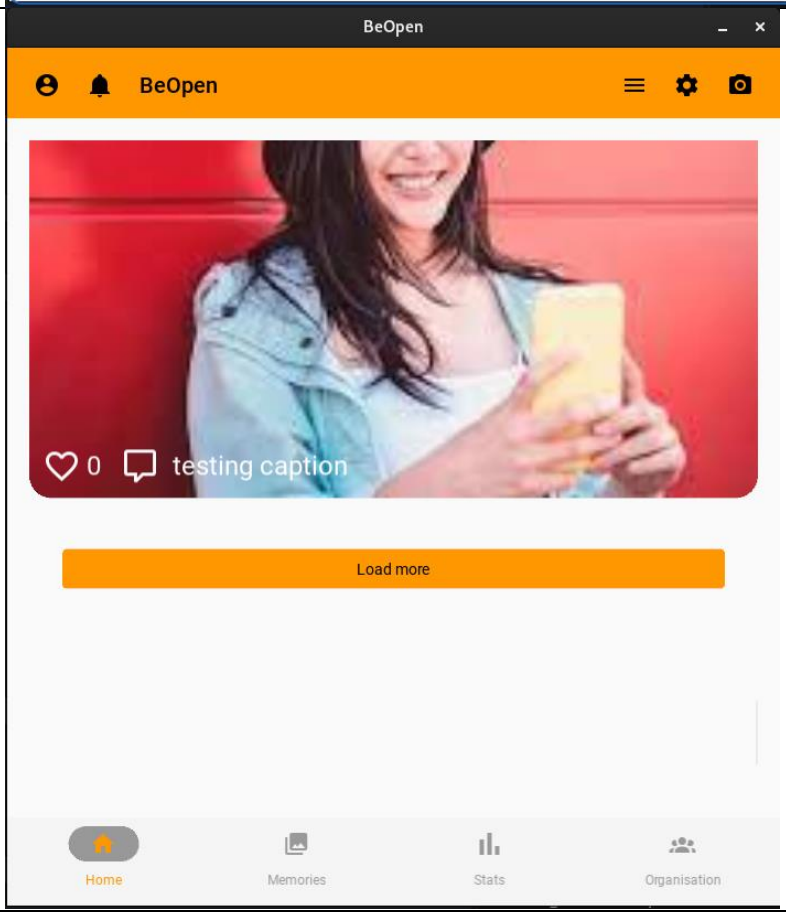
H.1B.V	1	 BeOpen BeOpen testing caption user6 Home Memories Stats Organisation
H.1C.I	1	 BeOpen BeOpen user9 1 testing caption Home Memories Stats Organisation

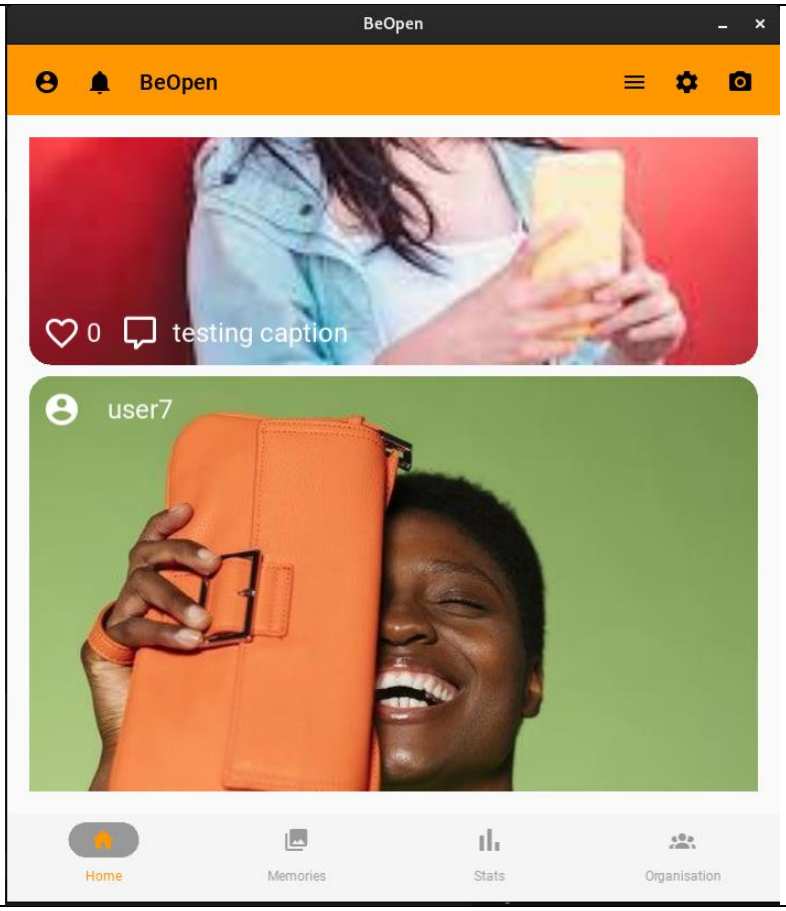
H.1C.II	1	
H.1D.I	1	

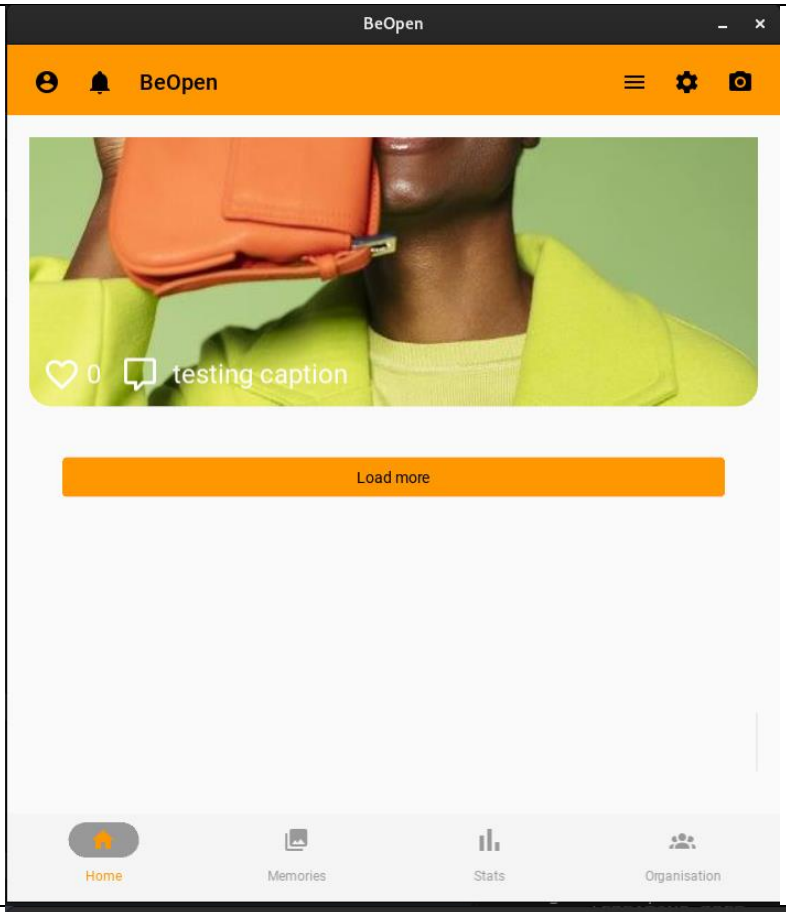
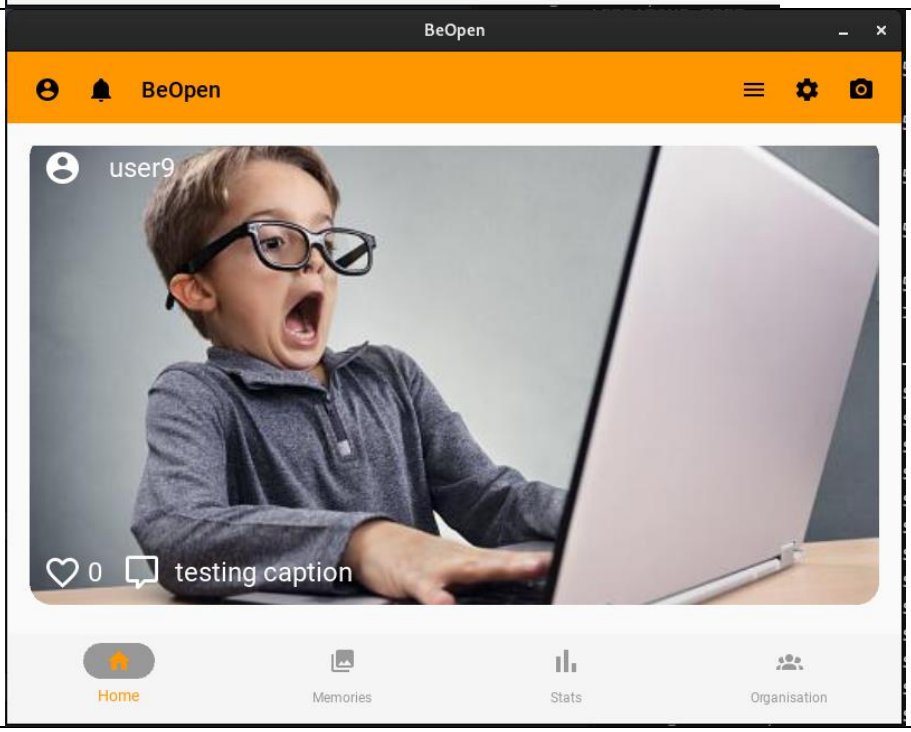
H.2A.II	1	
H.2B.I	1	

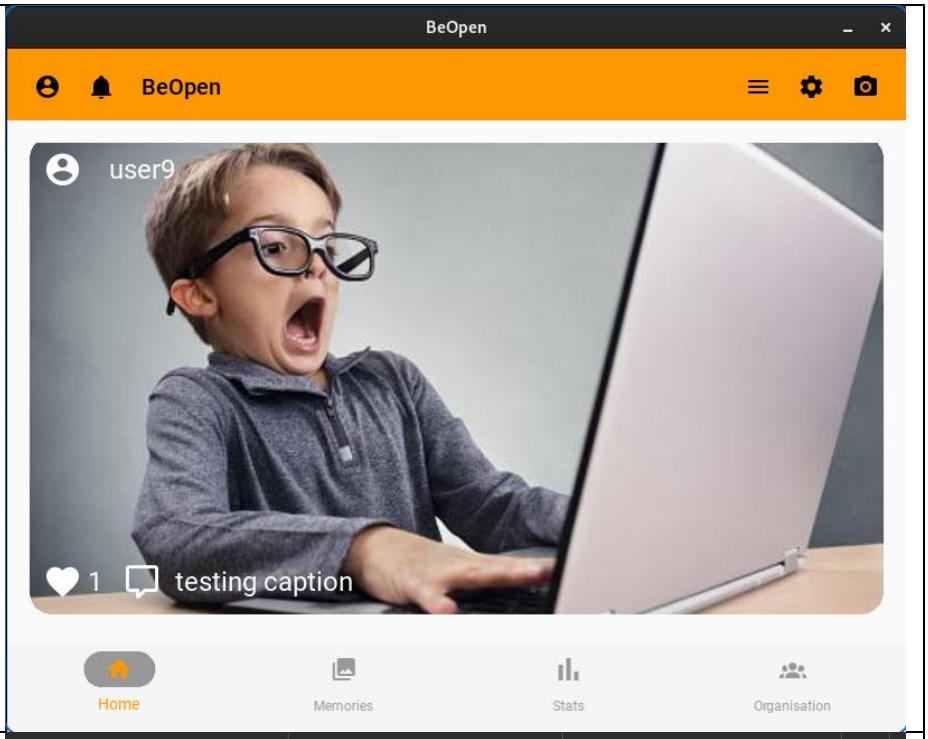
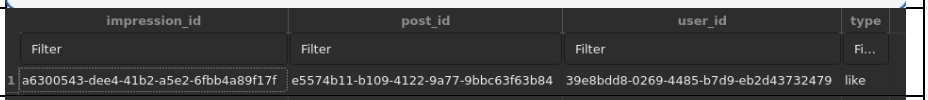
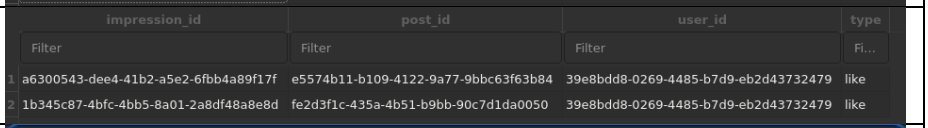
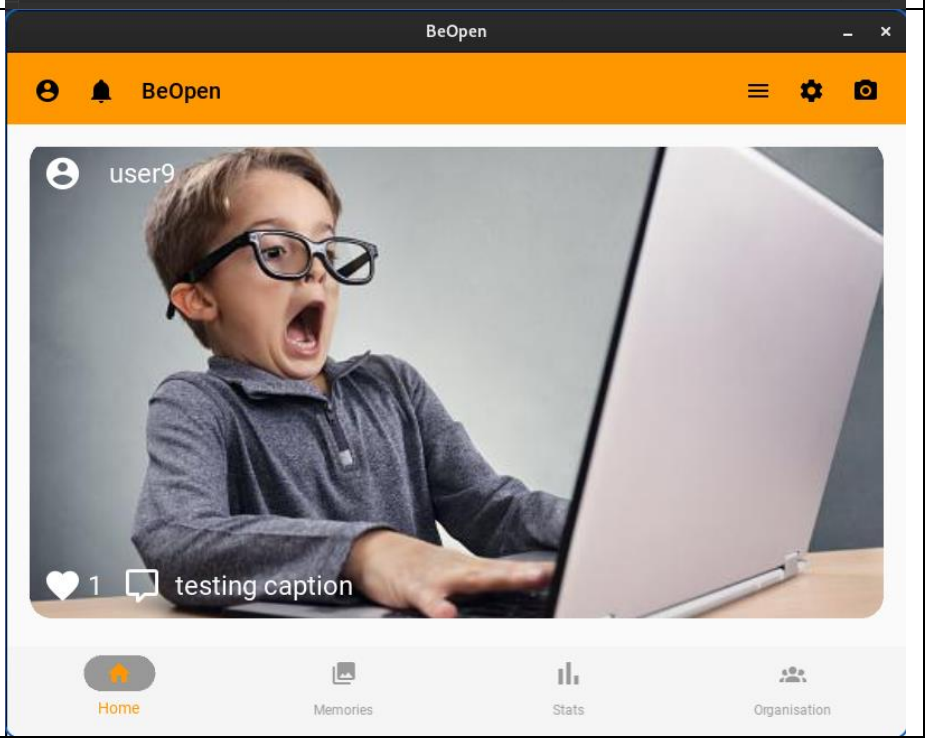
H.2B.II	1	
H.2C.I	1	

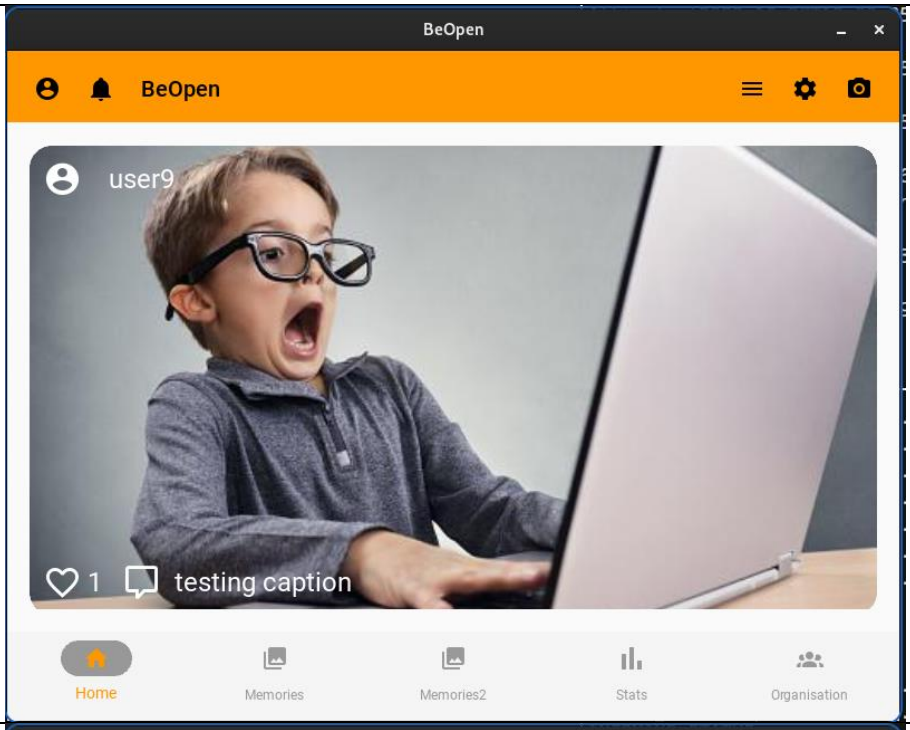
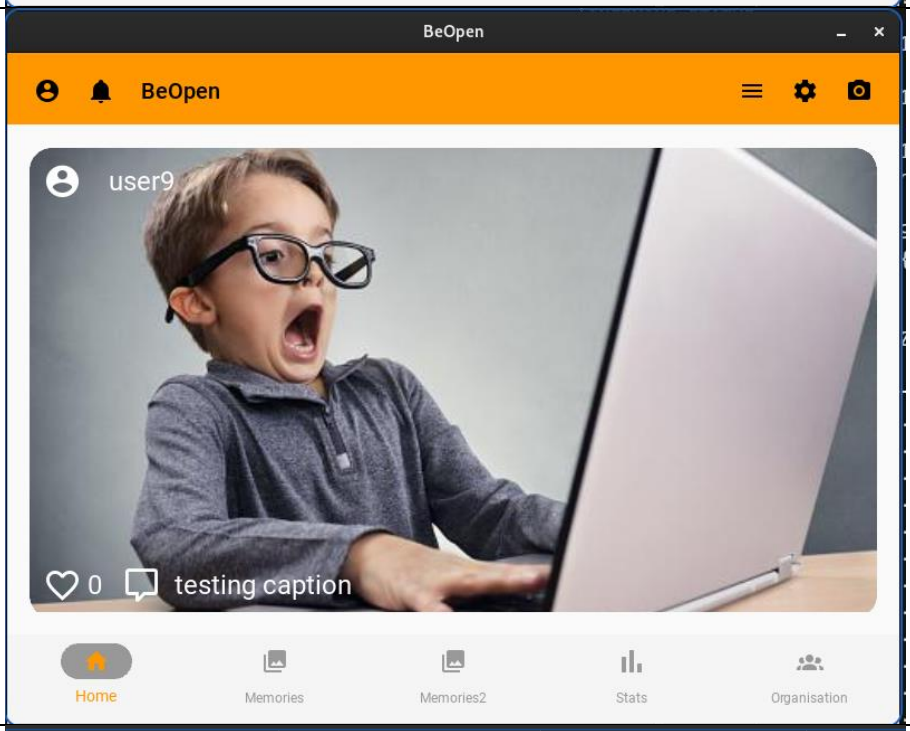
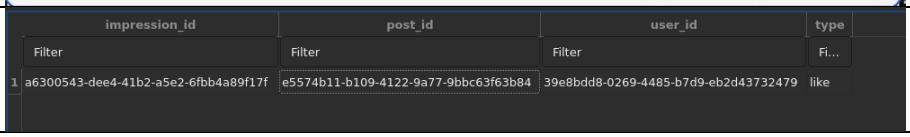
H.2C.I	2	 BeOpen BeOpen testing caption Home Memories Stats Organisation
H.1D.I	1	 BeOpen BeOpen user9 1 testing caption Home Memories Stats Organisation

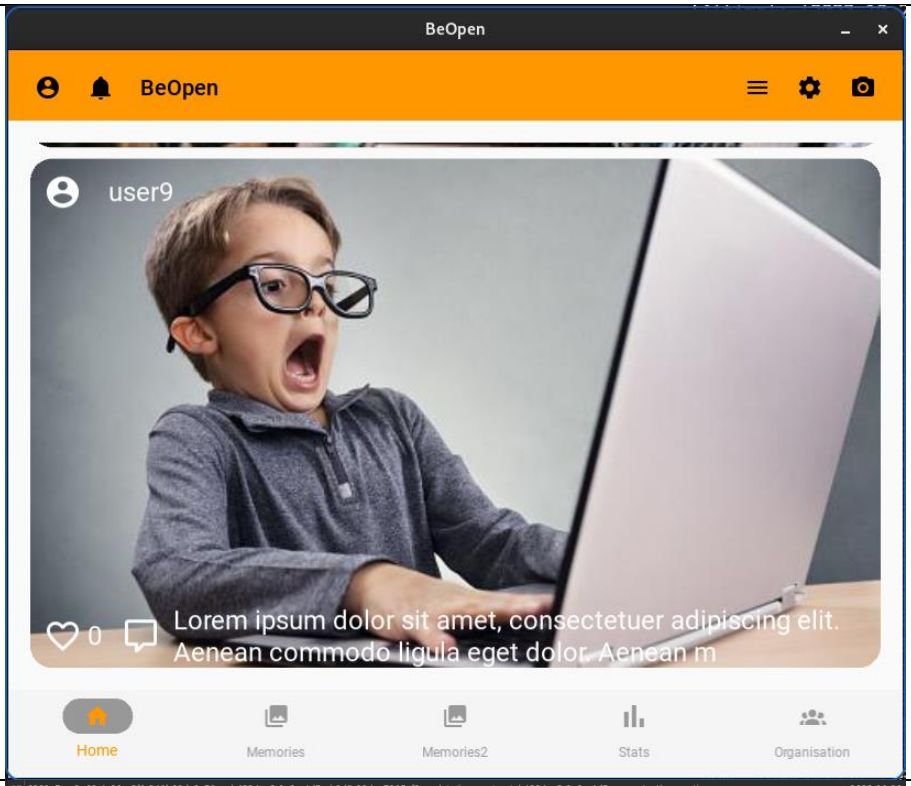
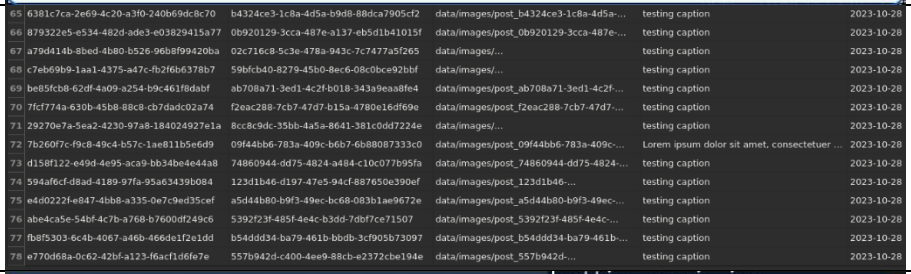
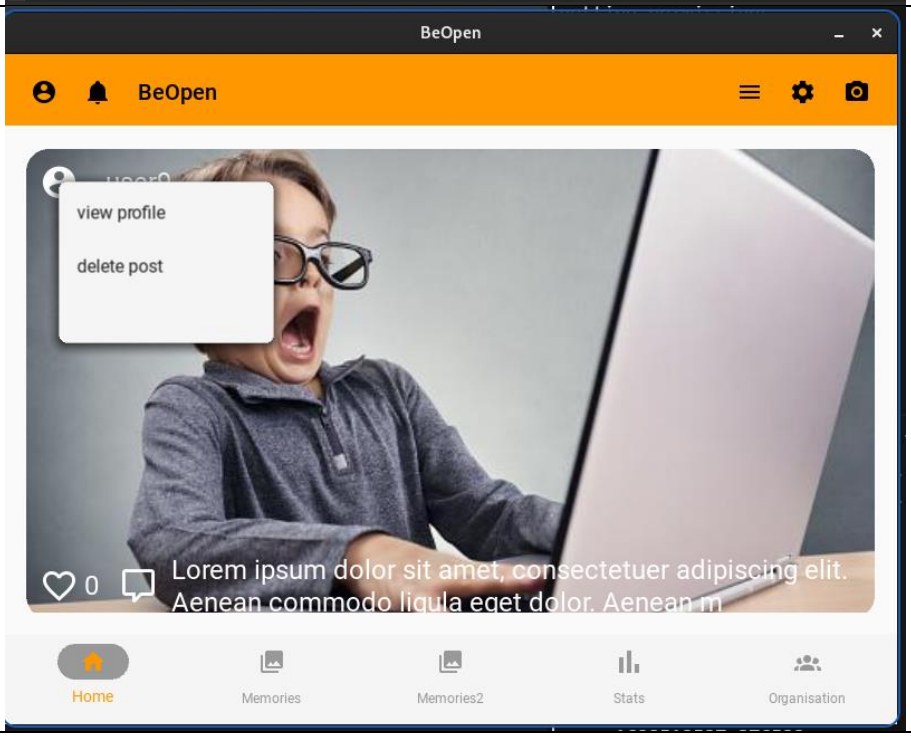
H.1D.II	1	
H.2D.I	1	

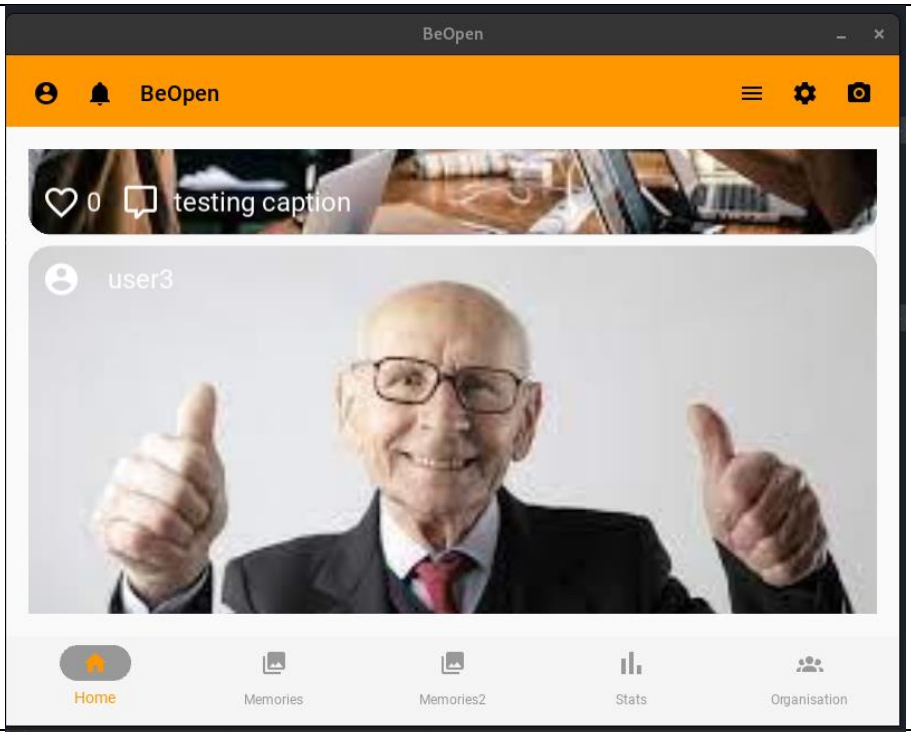
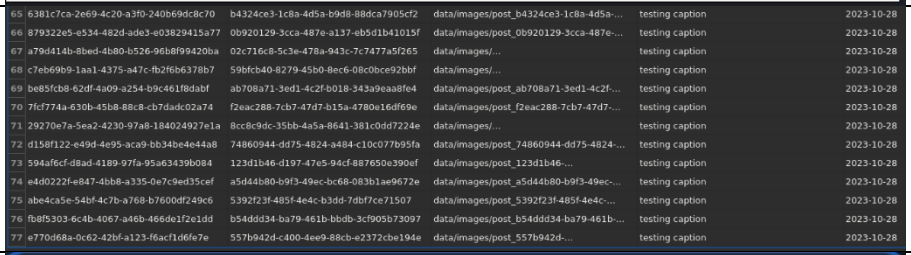
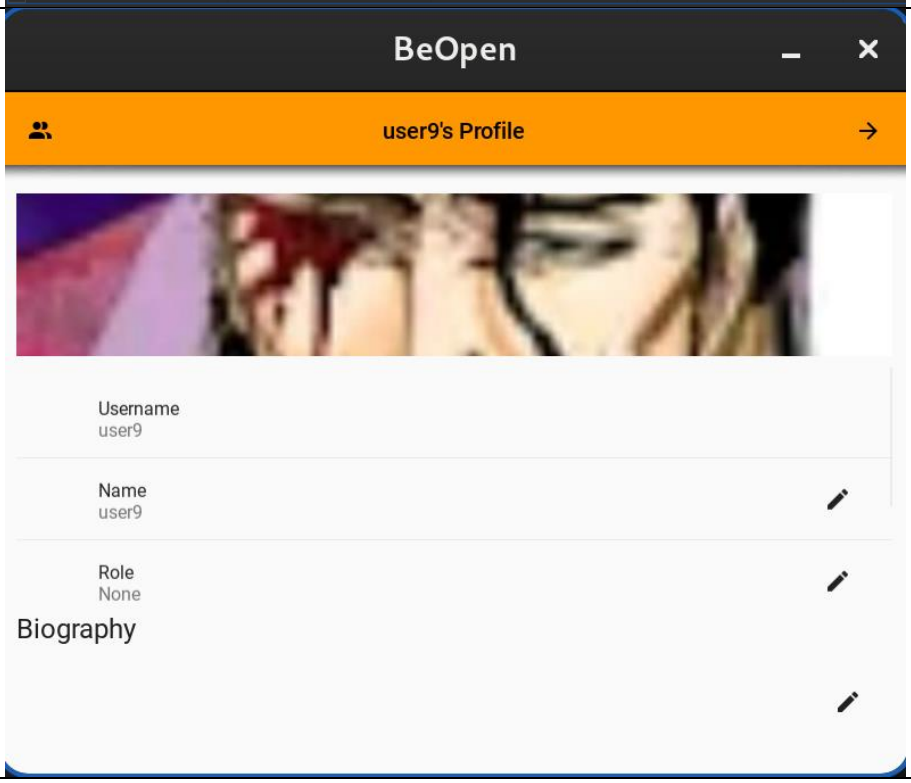
H.2D.I	2	 BeOpen BeOpen 0 testing caption user7 Home Memories Stats Organisation

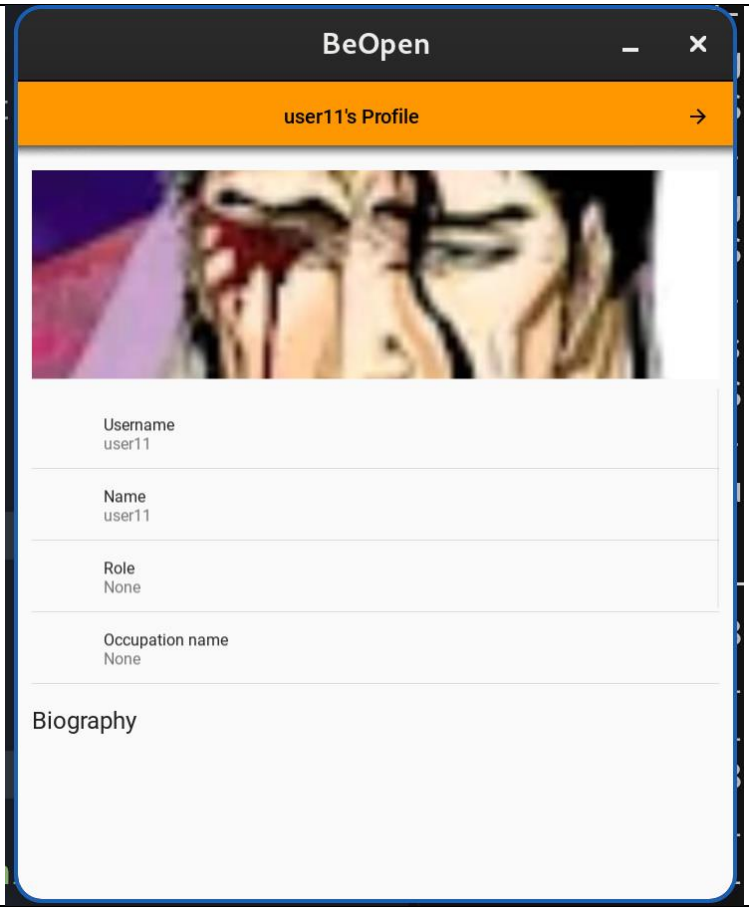
H.2D.I	3	 BeOpen BeOpen 0 testing caption Load more Home Memories Stats Organisation
H.3A.I	1	 BeOpen BeOpen user9 0 testing caption Home Memories Stats Organisation

H.3A.I	2	
H.3A.I	3	
H.3B.I	1	
H.3B.I	2	

H.3B.I	3	
H.3B.II	1	
H.3B.II	2	

H.4A.I	1	
H.4A.I	2	
H.4A.I	3	

H.4A.I	4	
H.4A.I	5	
H.4B.I	1	

H.4C.I	1	 BeOpen user11's Profile Username user11 Name user11 Role None Occupation name None Biography	

Posting

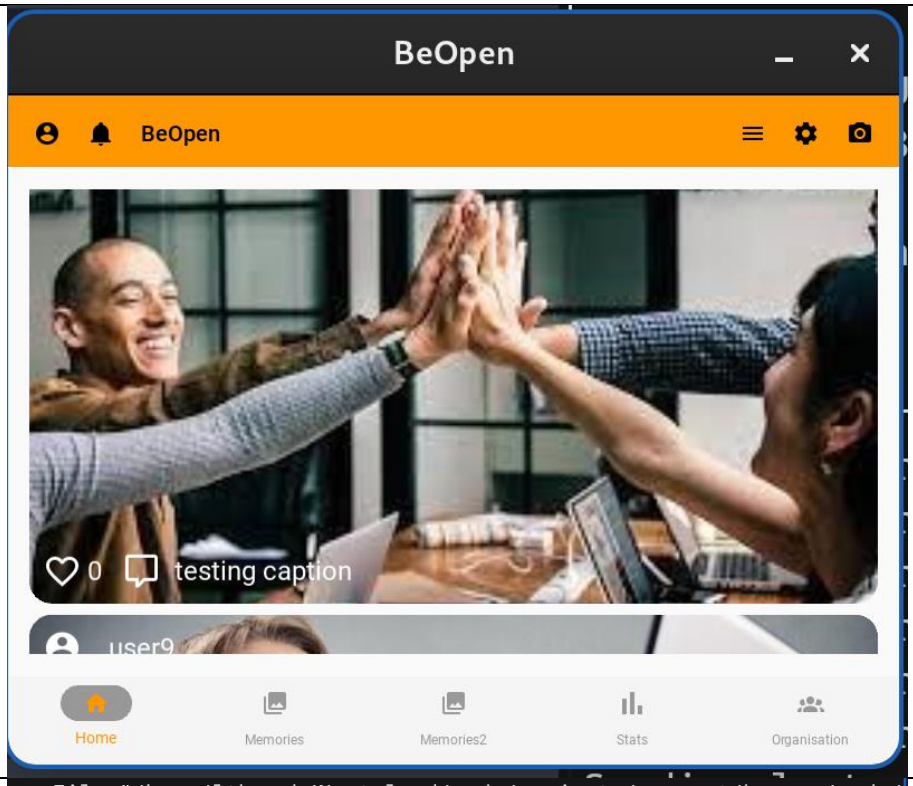
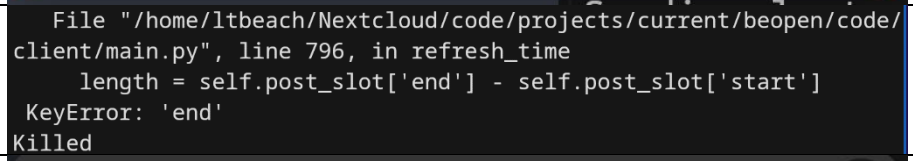
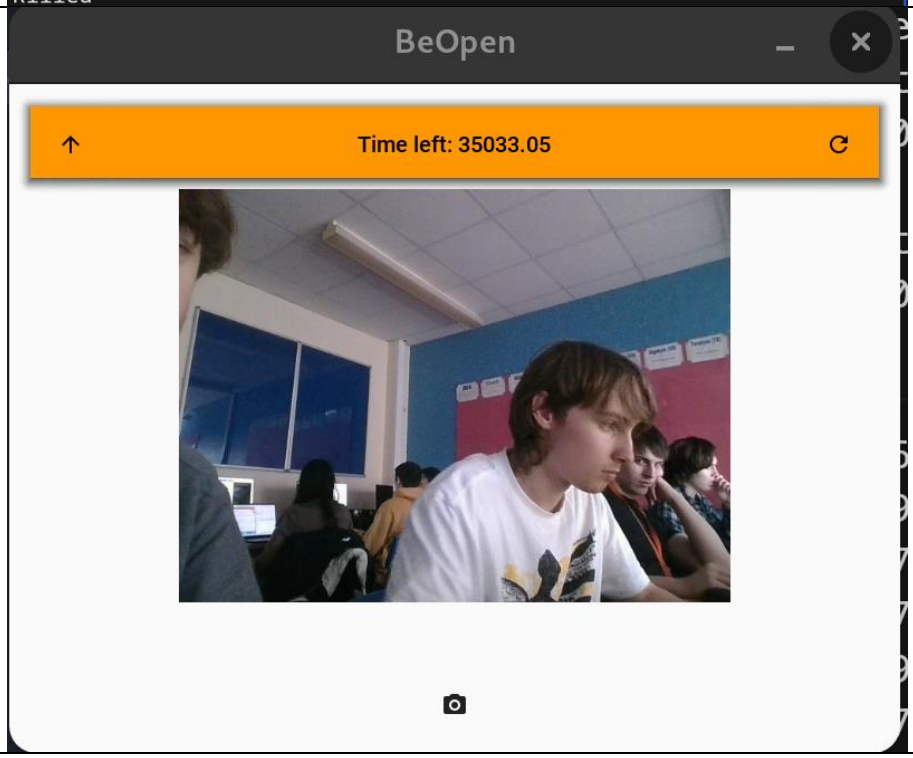
Test Number	Test Description	Expected	Observed	Action
PT.1A.I	When post time comes about in the day the UI should add the camera icon to the top bar on the homepage. For this test I manually set the time slot for the day on the server to be now for testing purposes	The camera icon should be added to the top bar on the hompage, this camera icon when clicked should lead to the posting page	As expected	
PT.1B.I	Clicking the camera button this should switch the view to the camera page where the user can take a picture for their post	This page should have a camera view a capture button and a countdown at the top	The client errored while loading the page, dictionary key error	Changed how the client fetched the day end and start times.
PT.1B.II	Clicking the camera button this should switch the view to the camera page where the user can take a picture for their post	This page should have a camera view a capture button and a countdown at the top	The page has wrong padding and the time countdown needs to be converted to minutes and hours	Created a method for converting the time into hours, minutes and seconds. As well as adding a clock.
PT.1B.III	Clicking the camera button this should	This page should have a camera view a	As expected	

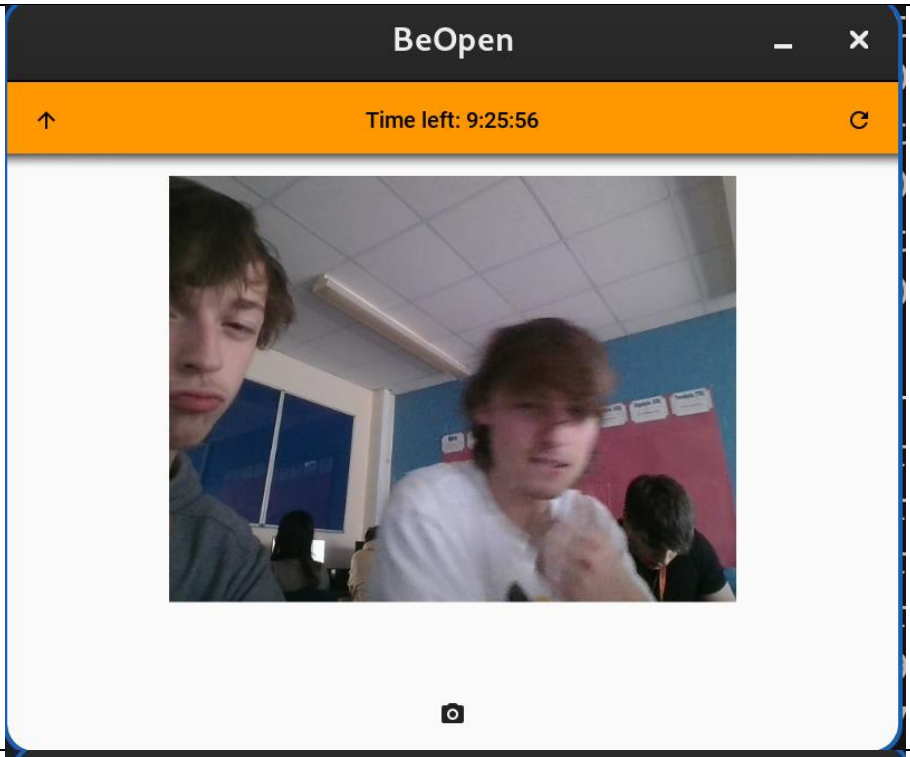
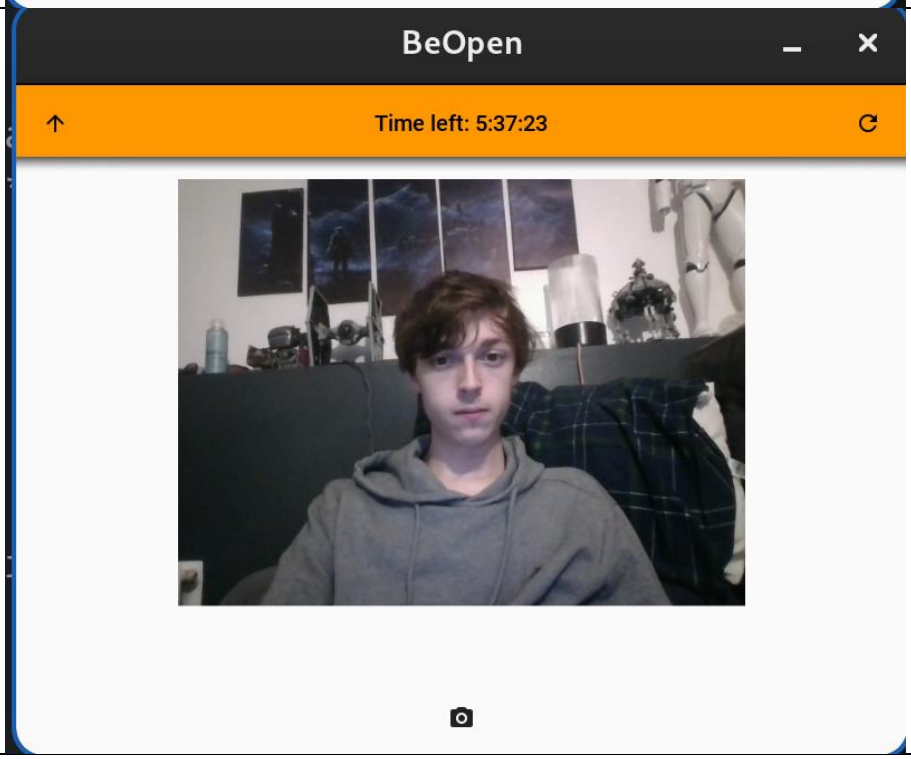
	switch the view to the camera page where the user can take a picture for their post	capture button and a countdown at the top		
PT.2A.I	Clicking the capture button	This should take the user to the “post review” page where they can add a caption and look at the photo they just took	The client complains the screen it tries to switch to doesn't exist.	Forgot to add the page object to the screen manager
PT.2A.II	Clicking the capture button	This should take the user to the “post review” page where they can add a caption and look at the photo they just took	As expected	
PT.3A.I	Pressing the back button from the camera page	This should take the user back to the homepage	As expected	
PT.3B.I	Pressing the back button on the post review page	his should take the user back to the homepage	As expected	
PT.3C.I	Pressing the retake photo button on the post review page	This should take the user back to the camera page to retake a photo	Minor spelling problem in variable “self” (put “sefl”)	Fixed and now working as expected
PT.3D.I	Pressing the retake photo	The post preview photo	As expected	

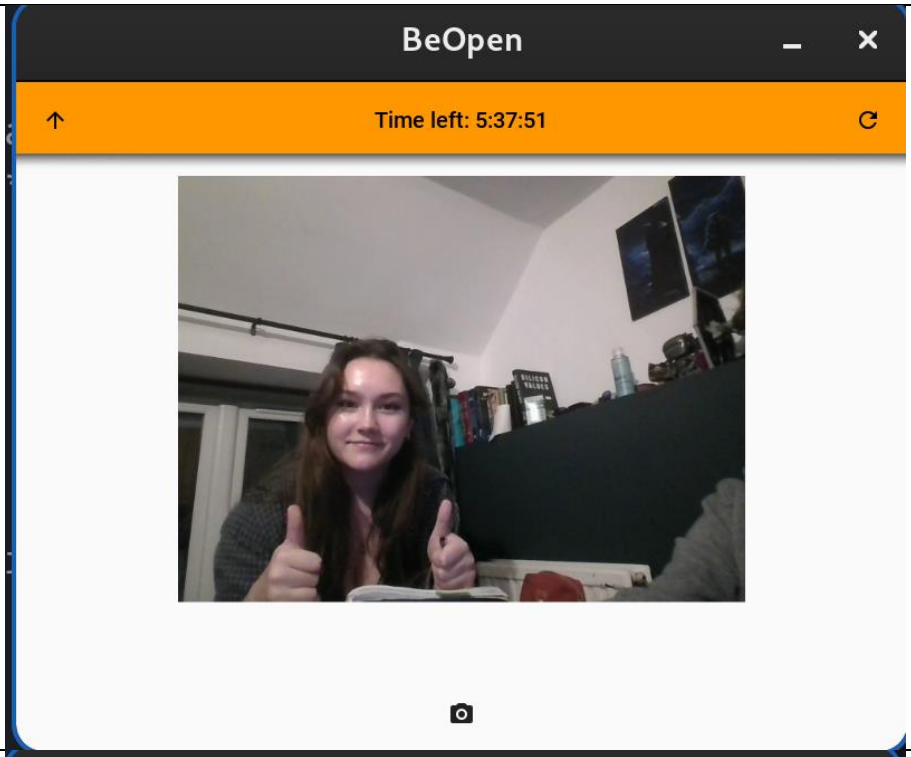
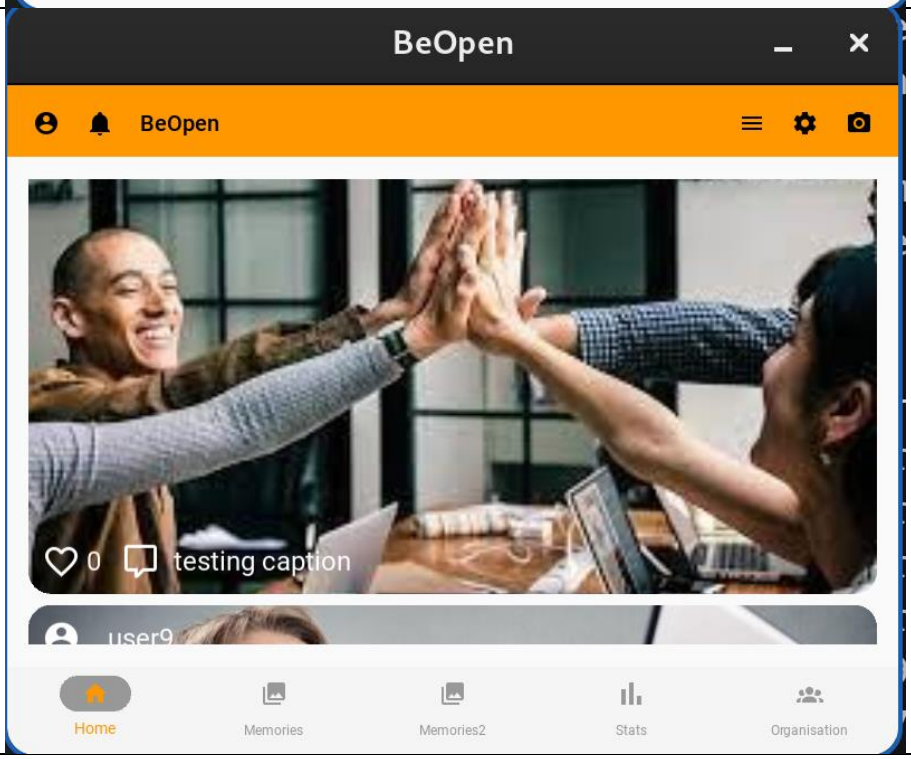
	and taking another photo for the post and seeing if it changes the photo on the post review page	should change to the retake photo		
PT.4A.I	Opening the photo page (as if going to take the photo) backing out to the home page and then going back into the take a photo page	It should go back to the homepage and then go back into the camera page as expected	The program crashes and complains it cant reactivate the camera	Instead of deleting the camera page on exit keep it created and just switch away from it. The camera even if deleted remains active on an OS level. Need to find a way to stop it on an OS level At the moment this seams to be a limitation of the kivy gui framework. The camera is badly documented. The documentation does list a stop() method but when using this the code claims the camera class has no such method. Will need to look

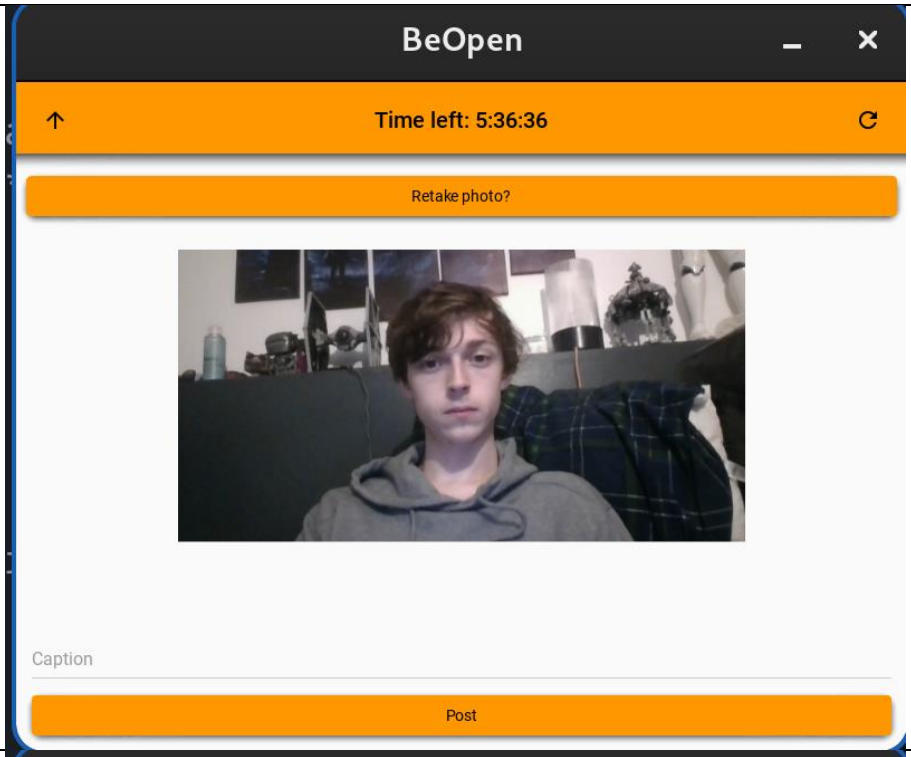
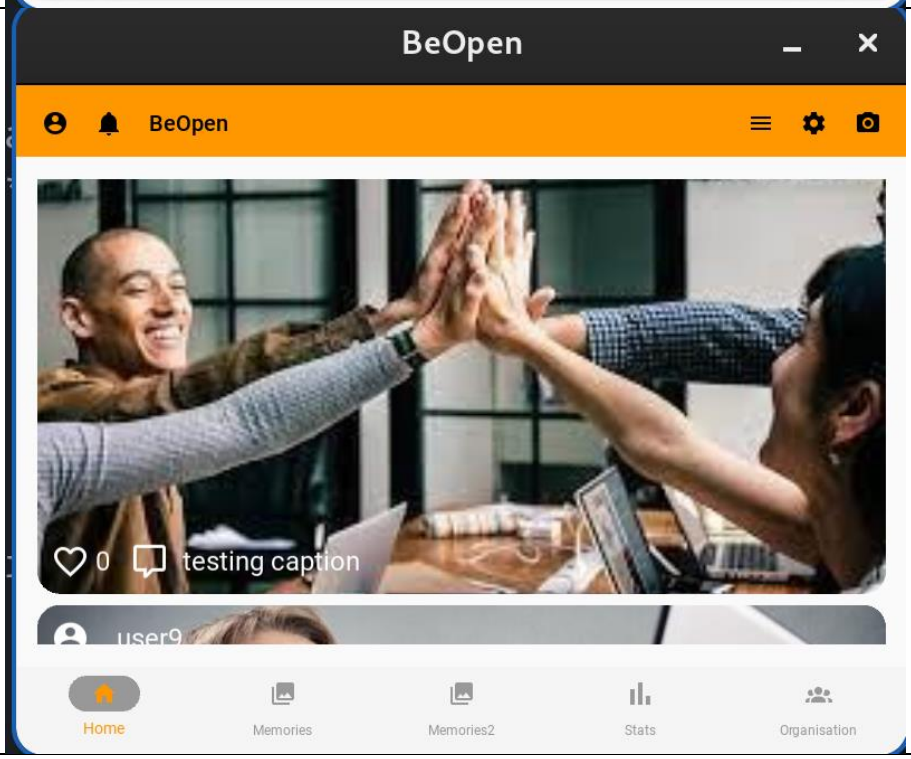
				into Kivy and OpenCV source code to find out how to accomplish this.
PT.5A.I	Taking a photo for the post, adding a caption and clicking the post button	This should send the user back to the homepage. The hompage top bar should no longer display the post button and the users post should appear on the server side database	The server could not make a post and the status message claimed it was due to insufficient/wrong data being provided.	This ended up being down to the server not being configured to accept both, png and jpg format images. The client takes images in the png format but the server could only do jpg, this has now been changed to allow both.
PT.5A.II	Taking a photo for the post, adding a caption and clicking the post button	This should send the user back to the homepage. The hompage top bar should no longer display the post button and the users post should appear on the server side database	As expected	

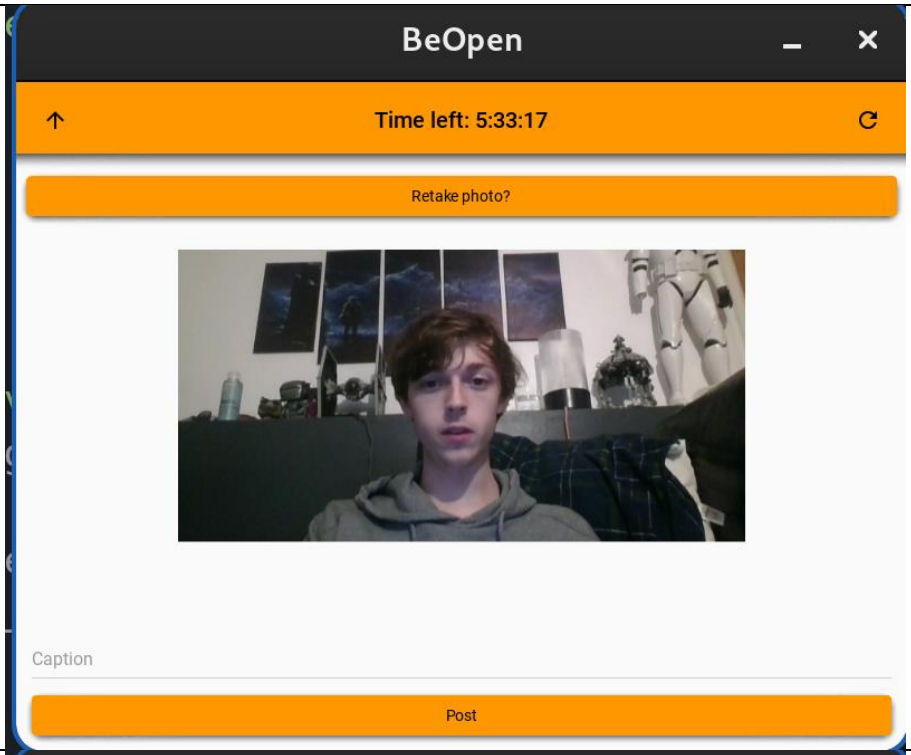
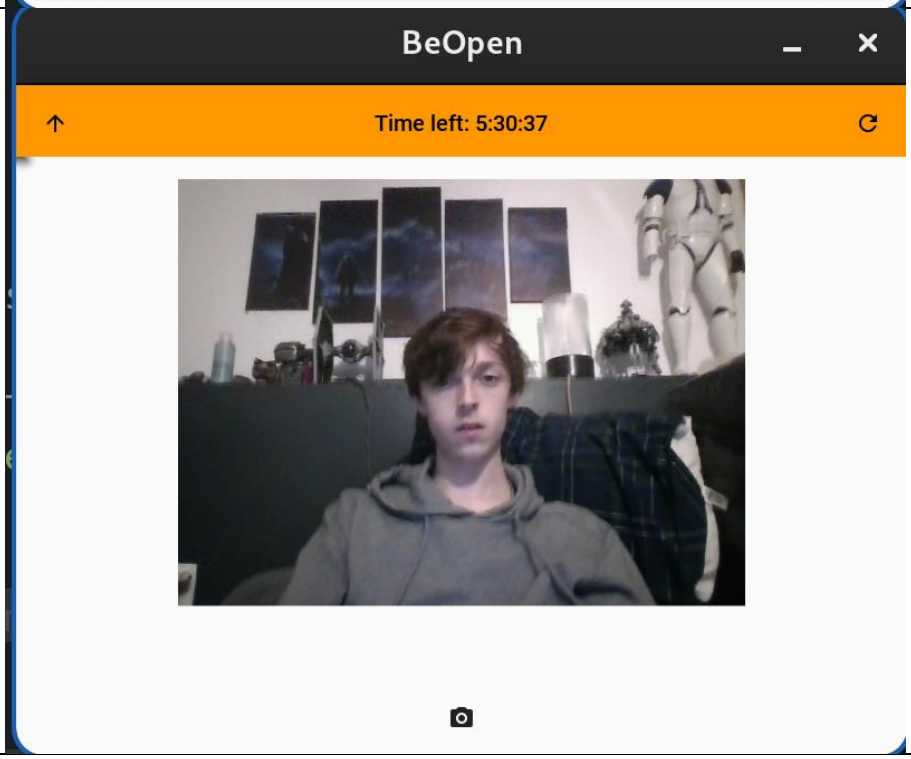
Test Number	Image Number	Image

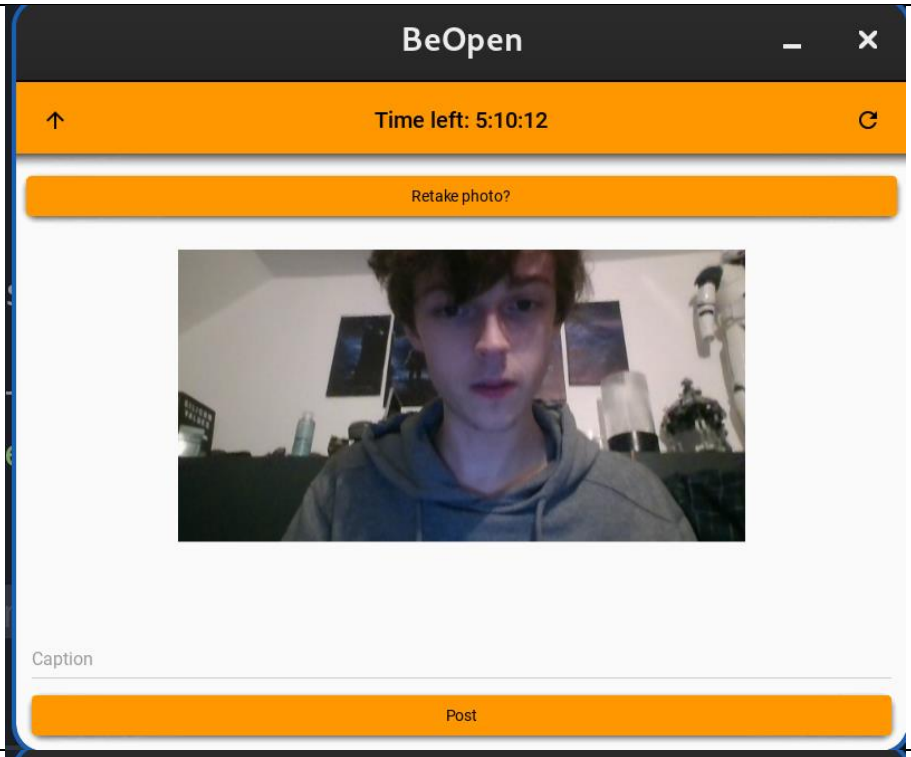
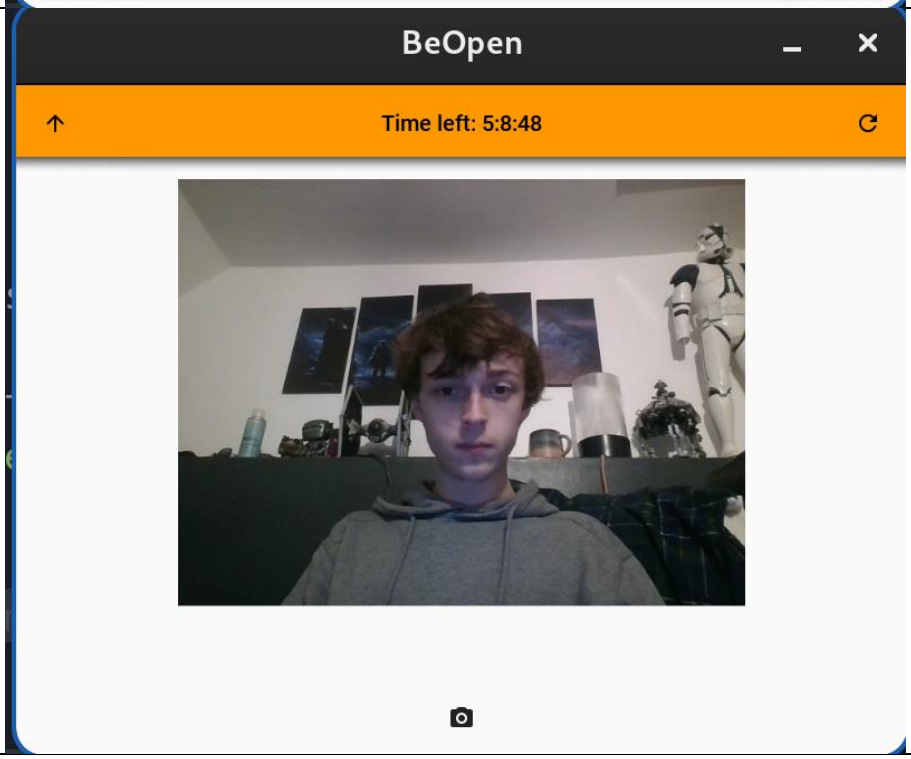
PT.1A.I	1	
PT.1B.I	1	
PT.1B.II	1	

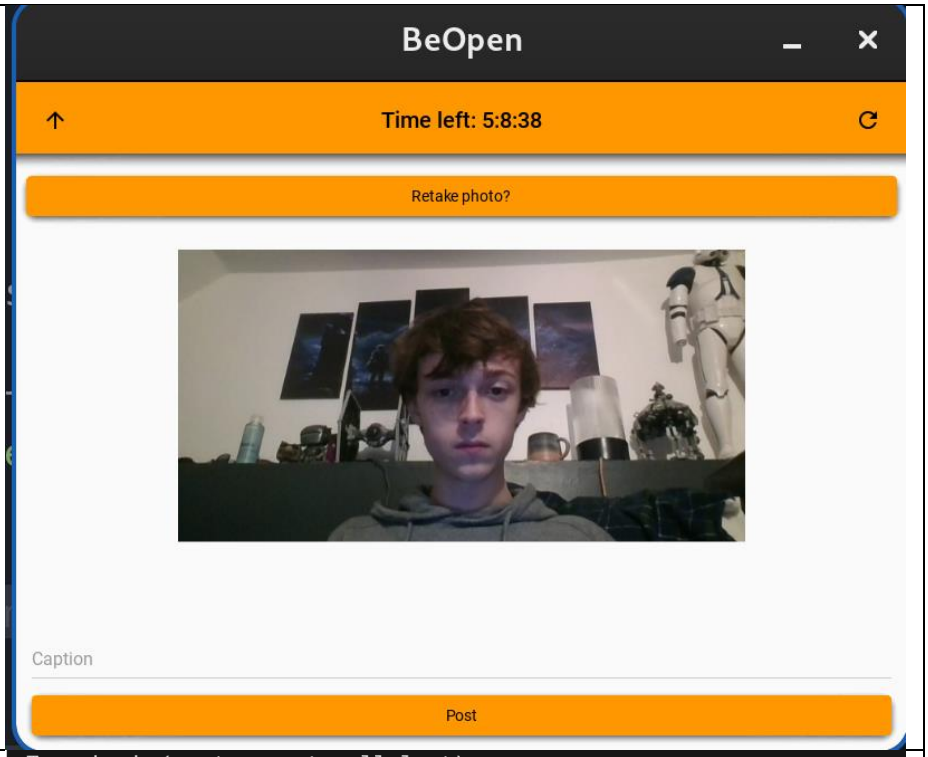
PT.1B.III	1	
PT.2A.I	1	

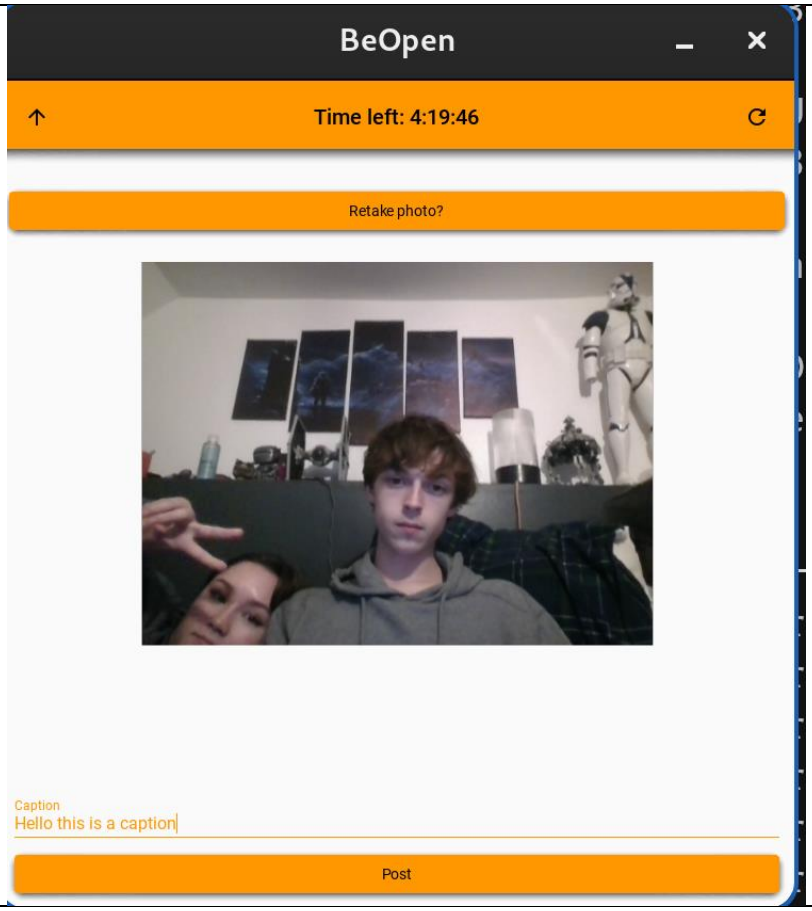
PT.3A.I	1	
PT.3A.I	2	

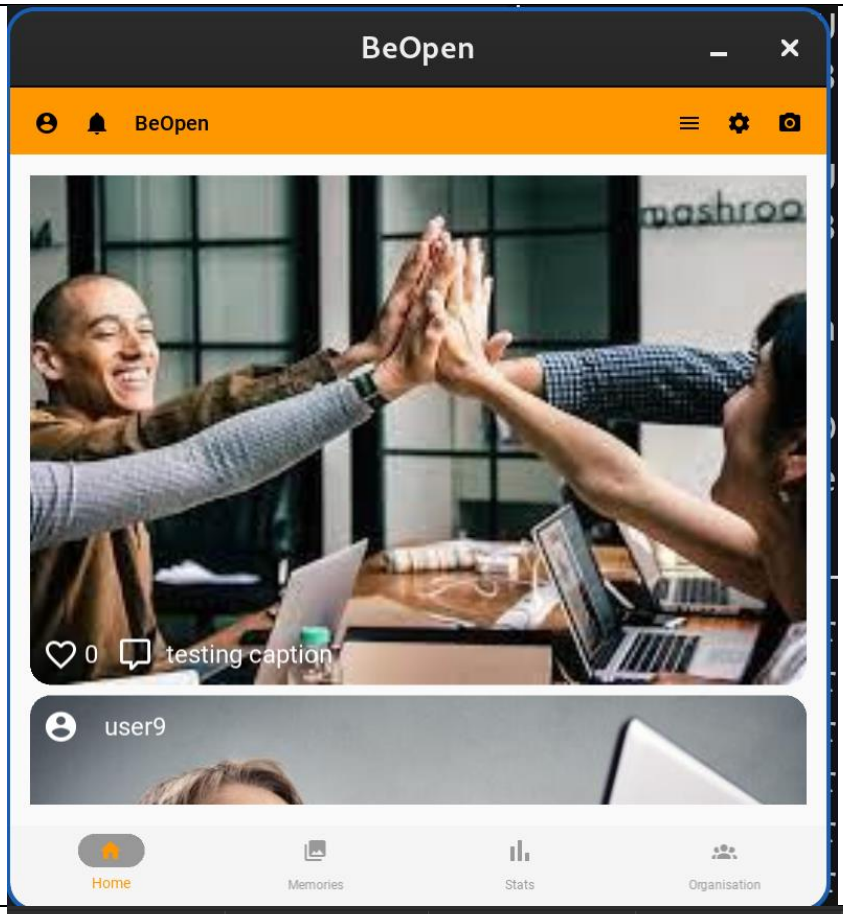
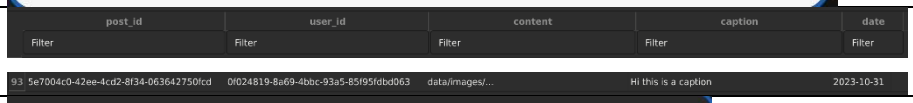
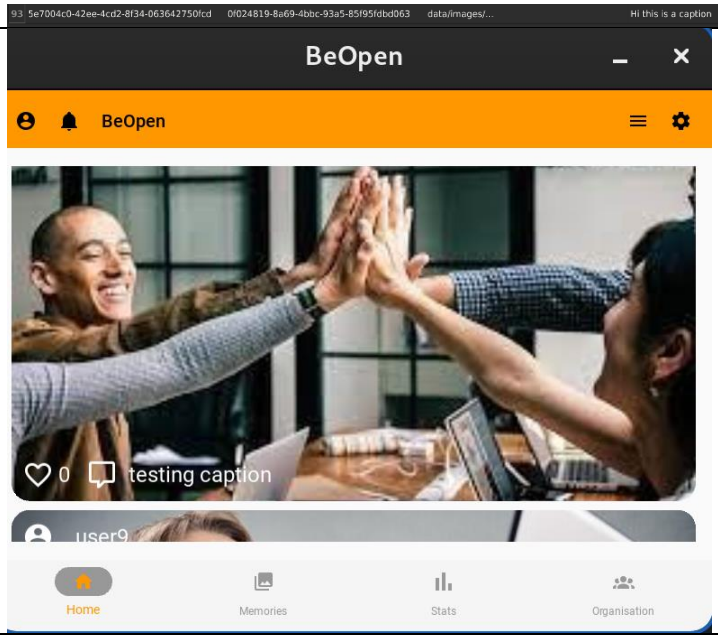
PT.3B.I	1	
PT.3B.II	2	

PT.3C.I	1	
PT.3C.I	2	

PT.3D.I	1	
PT.3D.I	2	

PT.3D.I	3	
PT.4A.I	1	Traceback (most recent call last): File "/home/ltbeach/.conda/envs/kivy/lib/python3.10/threading.py", line 1567, in _shutdown lock.acquire() File "/home/ltbeach/.conda/envs/kivy/lib/python3.10/site-packages/engineio/client.py", line 36, in signal_handler return original_signal_handler(sig, frame) File "/home/ltbeach/.conda/envs/kivy/lib/python3.10/site-packages/socketio/client.py", line 26, in signal_handler return original_signal_handler(sig, frame) KeyboardInterrupt: (kivy) [ltbeach@fedora client]$

PT.5A.I	1	 BeOpen Time left: 4:19:46 Retake photo? Caption Hello this is a caption Post	

PT.5A.I	2	
PT.5A.II	1	
PT.5A.II	2	

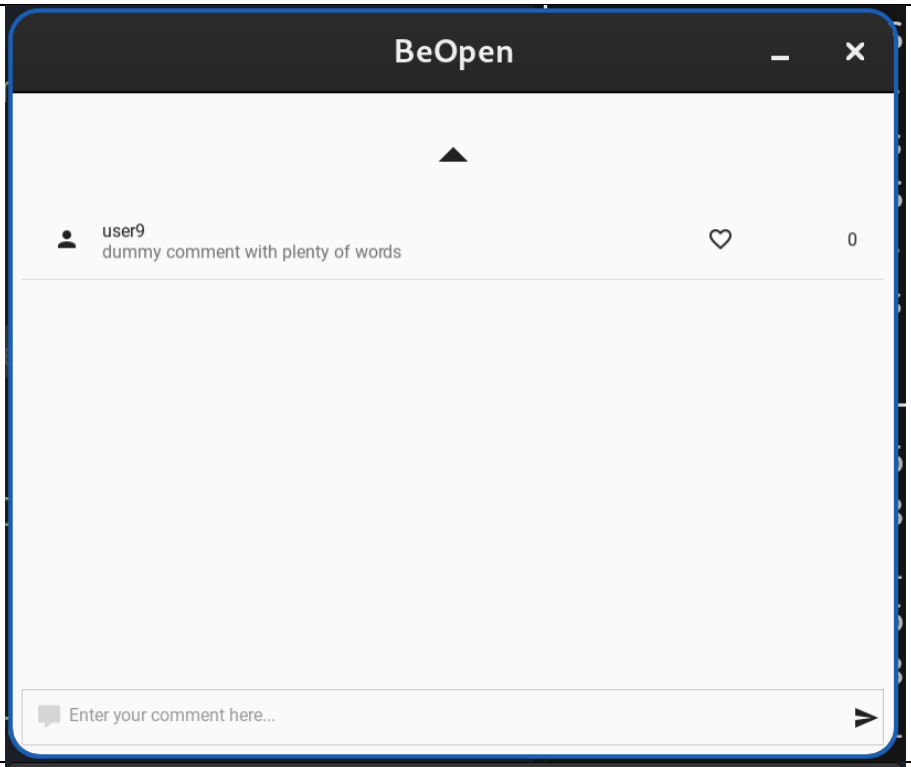
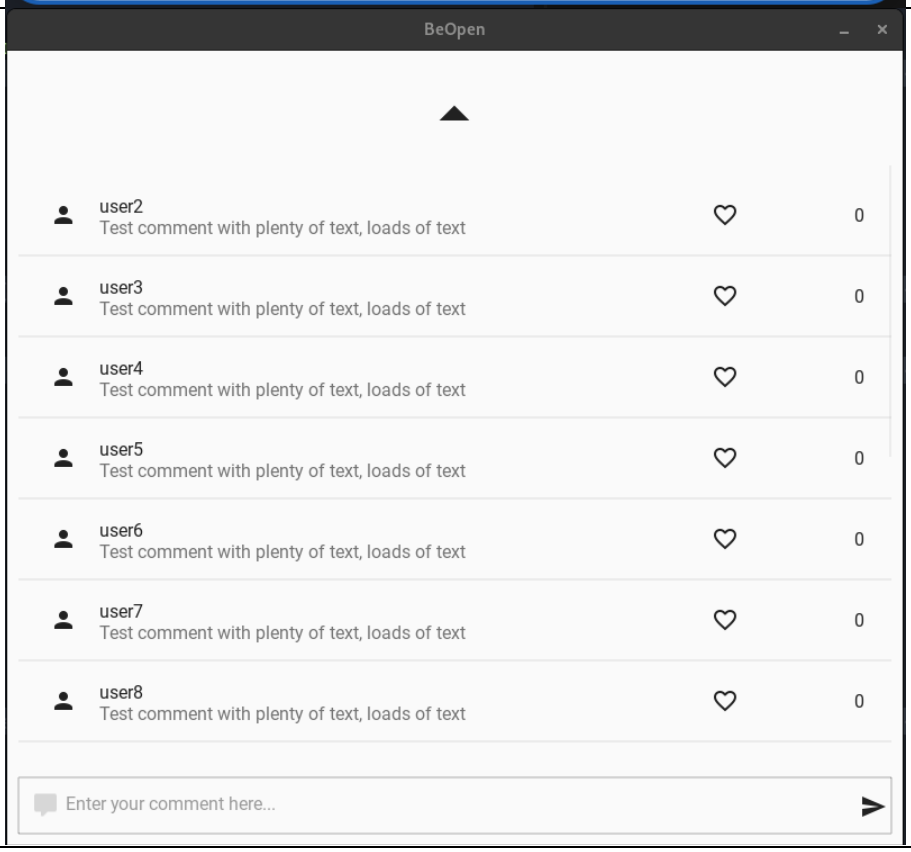
Comments

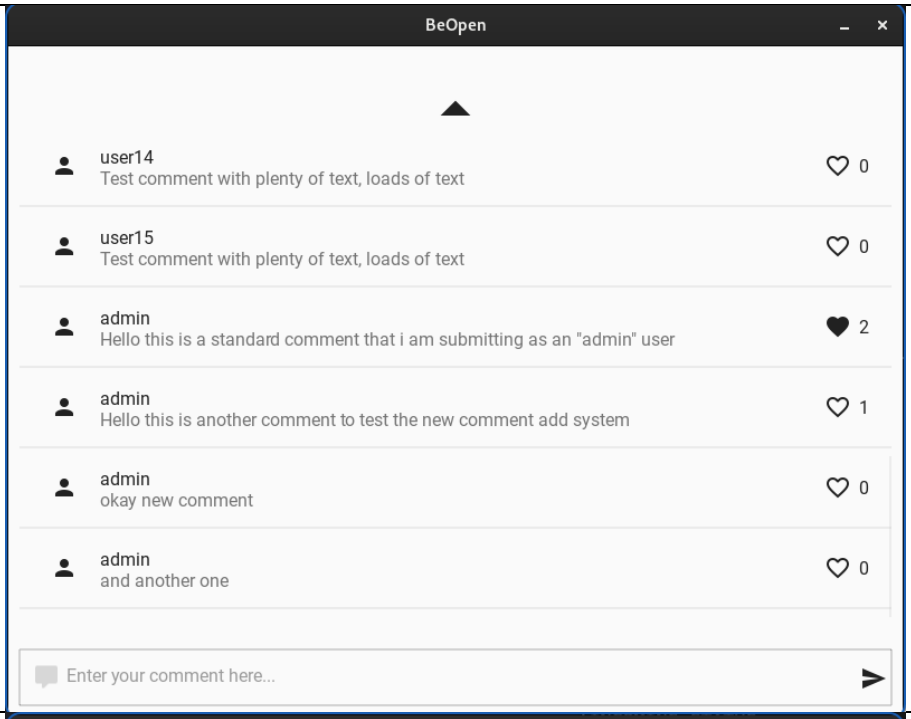
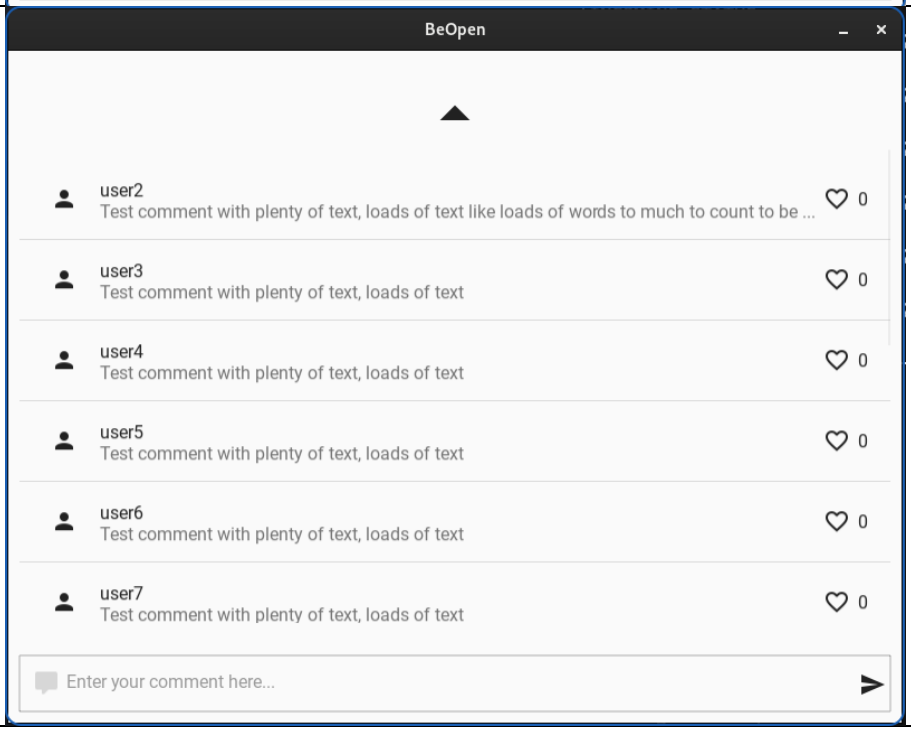
Test Number	Test Description	Expected	Observed	Action
C.1A.I	Going into the comments page	There should be an arrow at the top, this is the back button. In this case there should be just 1 comment in the comment list, and there should be comment box at the bottom	As expected	
C.1B.I	Have the comments page load a set of comments in this case 14 comments from 14 different users	Each comment should have a profile button, a like count and a like button	As expected	While this specific function was as expected I did find while setting up the test that the “comment_set” method never correctly set the post_id of the person commenting and defaults to a user commenting on their own post. This was subsequently fixed
C.1C.I	Have the comments	The page should display	Previously liked comments	This was due to the client

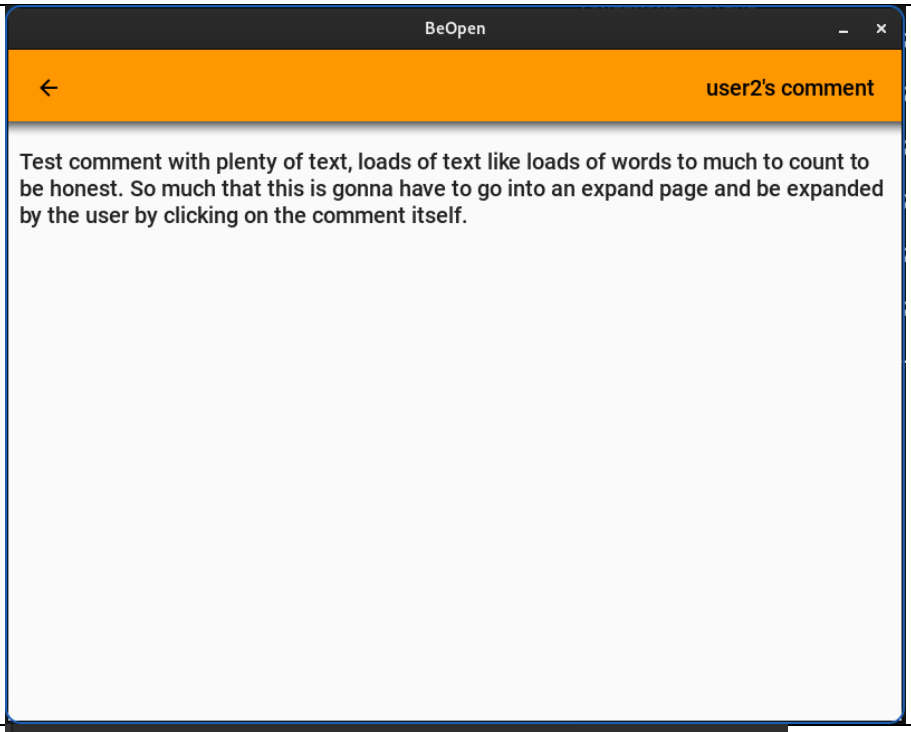
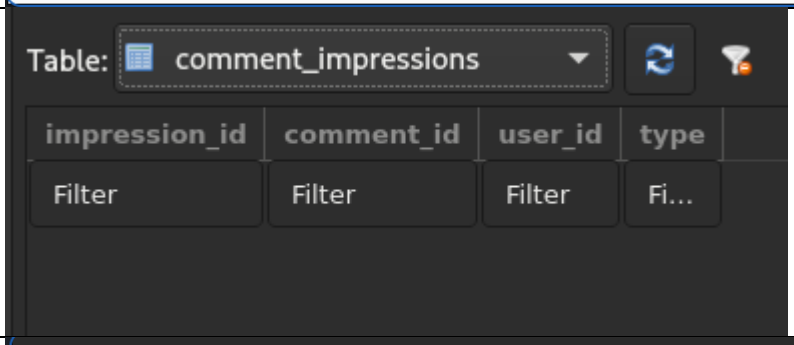
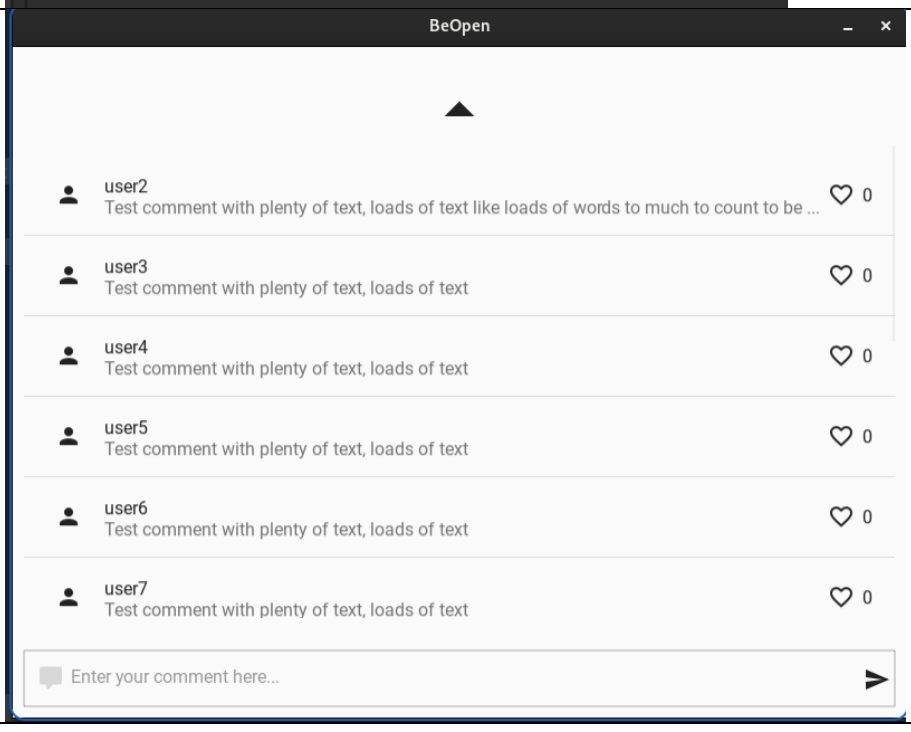
	page load comments but 1 comment has a like from another user, and a liked comment from the user viewing the comments	14 comments, 1 has a like count of 1 and unliked by the user viewing the page. Another comment as a like count of 2 and is liked by the user viewing the comments.	werent showing up as such, the count would be 1 but the heart was just an outline (as if you had not liked the comment yet)	incorrecly handling the data passed back by the server as well as using the wrong username to compare against the usernames that had liked the comment.
C.2A.I	Clicking on a long comment to expand it to the “expand page” and be able to read the full comment	On clicking the comment it should switch page to a page that displays the full text of the comment	As expected	
C.2B.I	Clicking the like button on a comment	It should turn the hear to a filled in heart and increase the like count by one. This change should be reflected on the server side database	The client side UI updated correctly and didn’t have any issues, the server didn’t run into any errors either but the database entry was never added for the like on user2’s comment	This is likely a data verification issue on the server side. Actually turned out to be the client not sending the comment_id correctly, it was left with a placeholder value.
C.2B.II	Clicking the like button on a comment	It should turn the hear to a filled in heart and increase the like count by one. This change should	As expected	

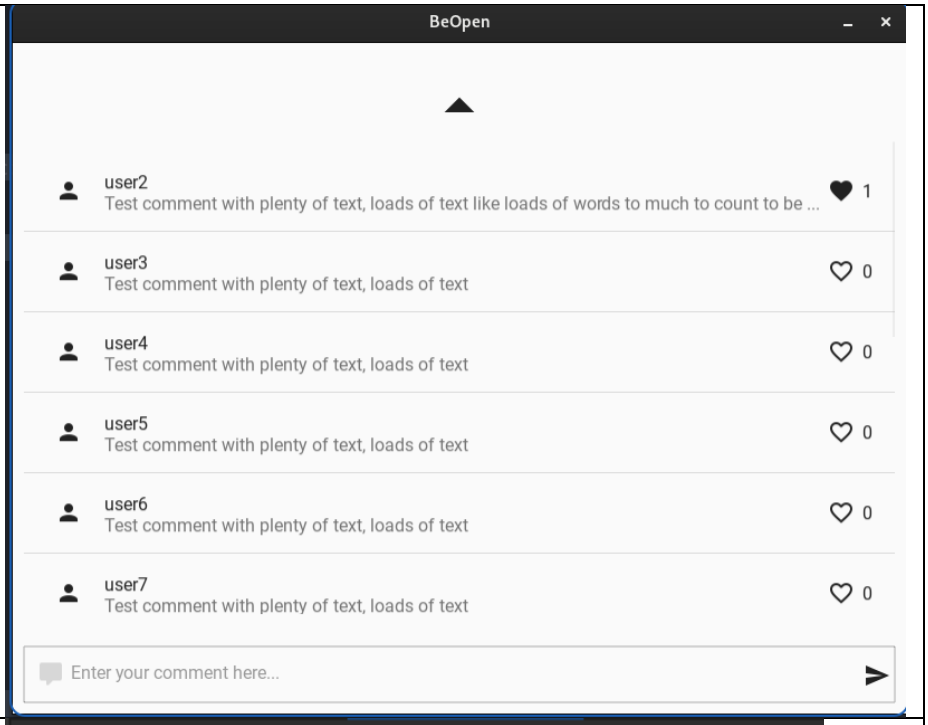
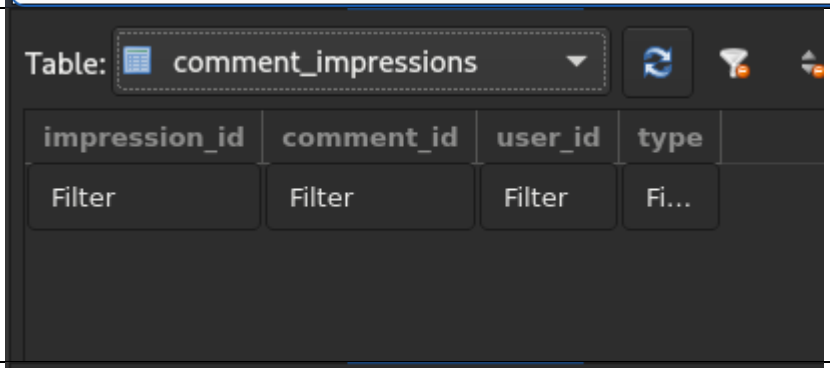
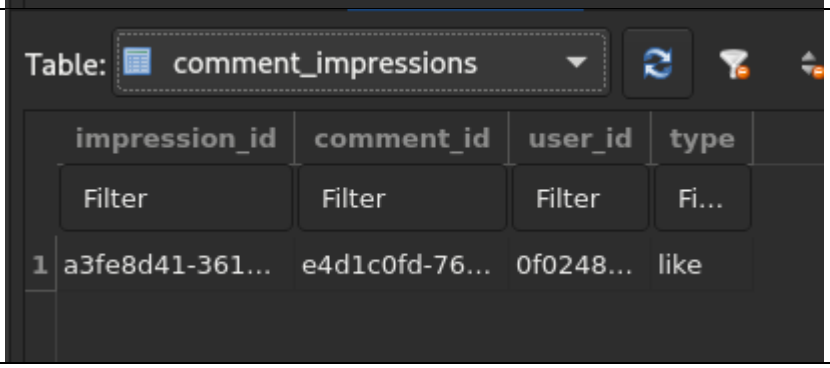
		be reflected on the server side database		
C.2C.I	Clicking the unlike button on a liked comment	The comment should change the heart to just the outline of a heart. The like count should also decrease and then then the change should be reflected in the server-side database	As expected	
C.3A.I	Clicking on the profile button on a comment as an admin. Then on the menu that should appear clicking view profile	On clicking the profile button initially, a menu should appear for the user to select delete or view profile. (here we select view profile) this should then take the user to the comment users	As exepected	
C.3B.I	Clicking on the profile button as an admin. Then selecting delete comment.	When initially clicking the profile button a menu should appear with options to view profile or delete comment. Here we click delete comment. This should remove	As expected	

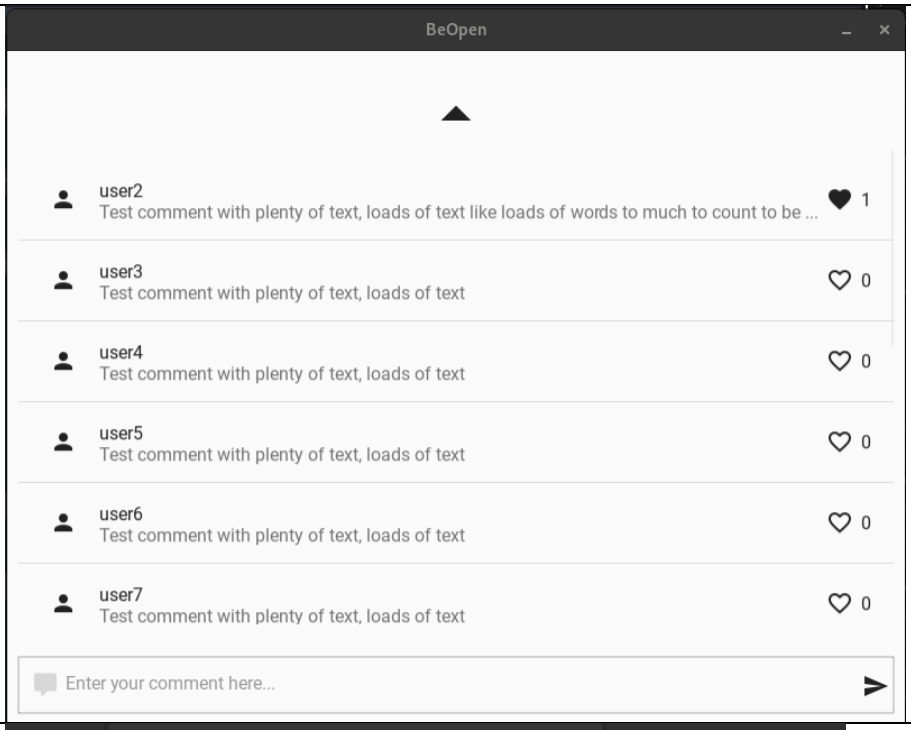
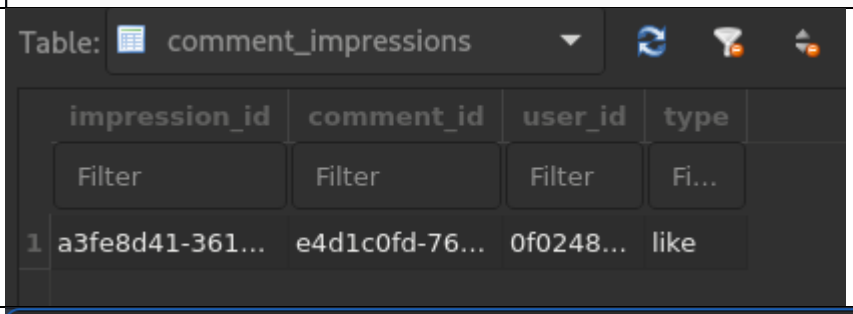
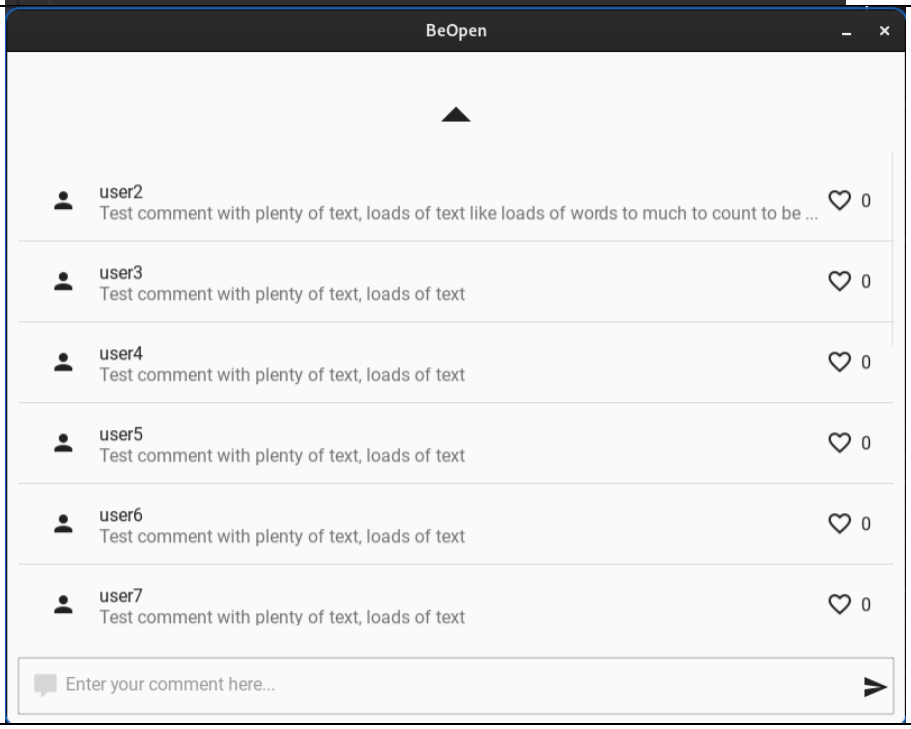
		the selected comment from the comment area, and removed on the server-side database		
C.4A.I	Typing some text into the comment box at the bottom of the page. Then clicking the send button to submit the comment	The comment should appear in the comment list above and the comment should appear in the server-side database	As expected	While there was no errors or major issues, the method for adding the comment to the interface itself was changed for performance reasons. Now the client instead of reloading all the comments just loads the new one in locally

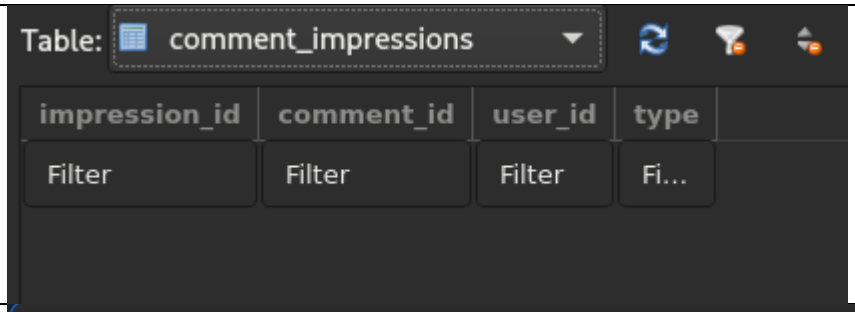
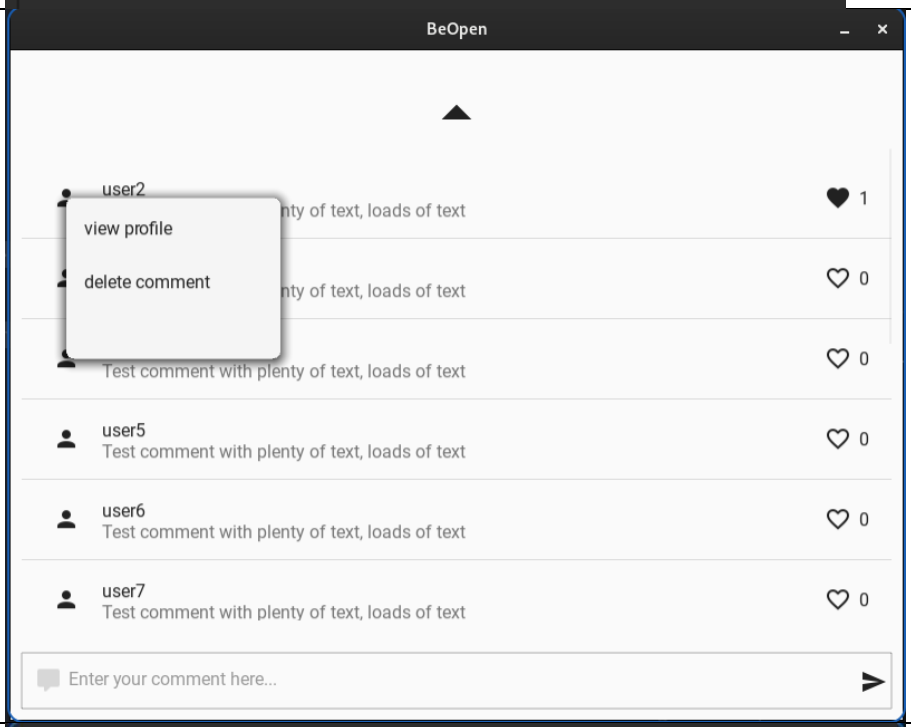
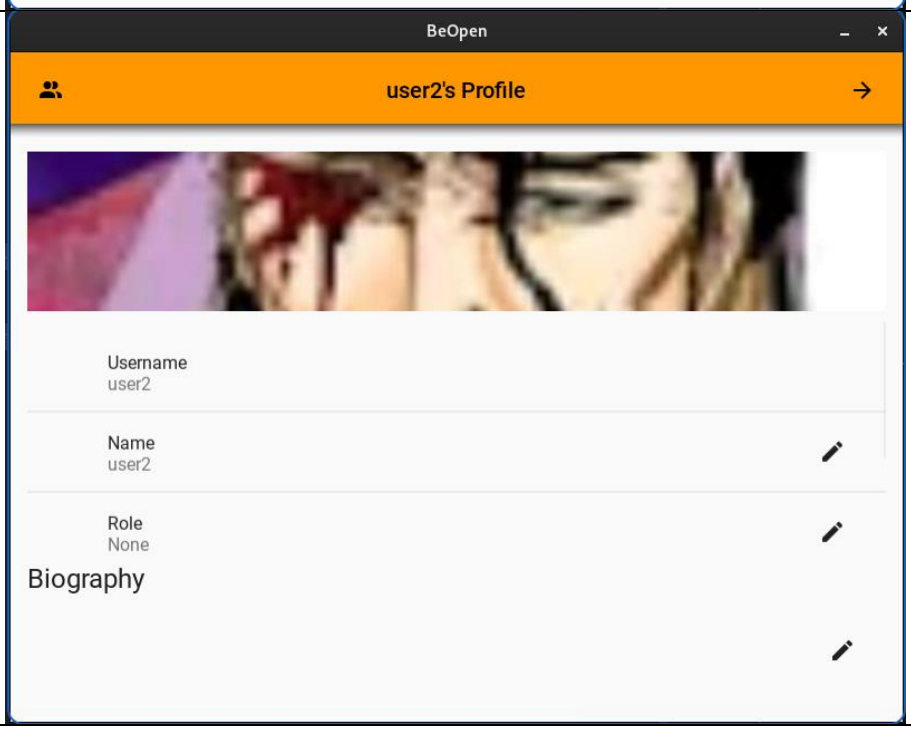
Test Number	Image Number	Image
C.1A.I	1	
C.1B.I	1	

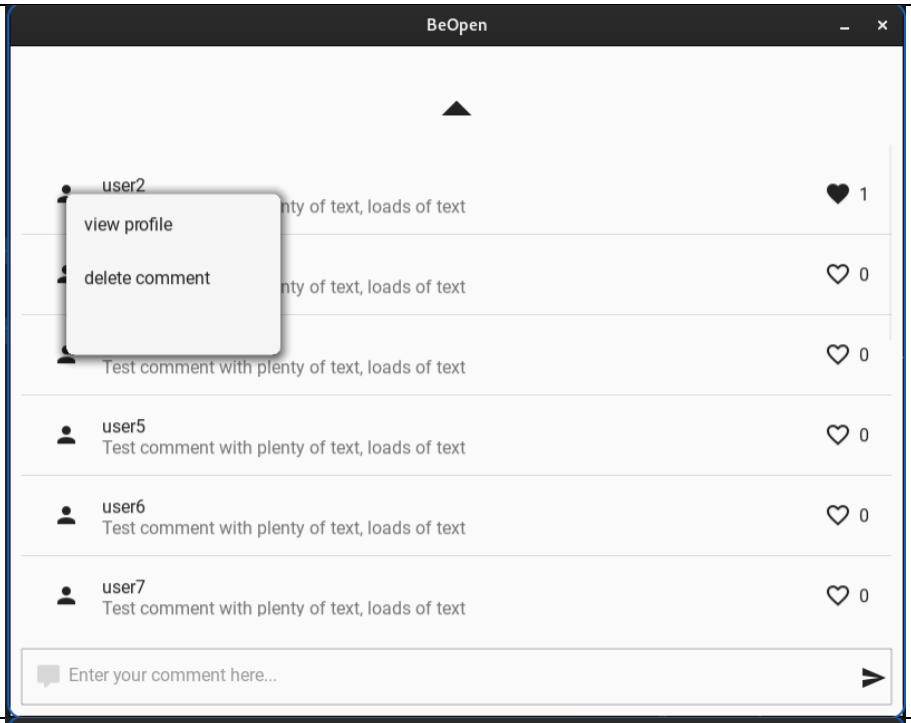
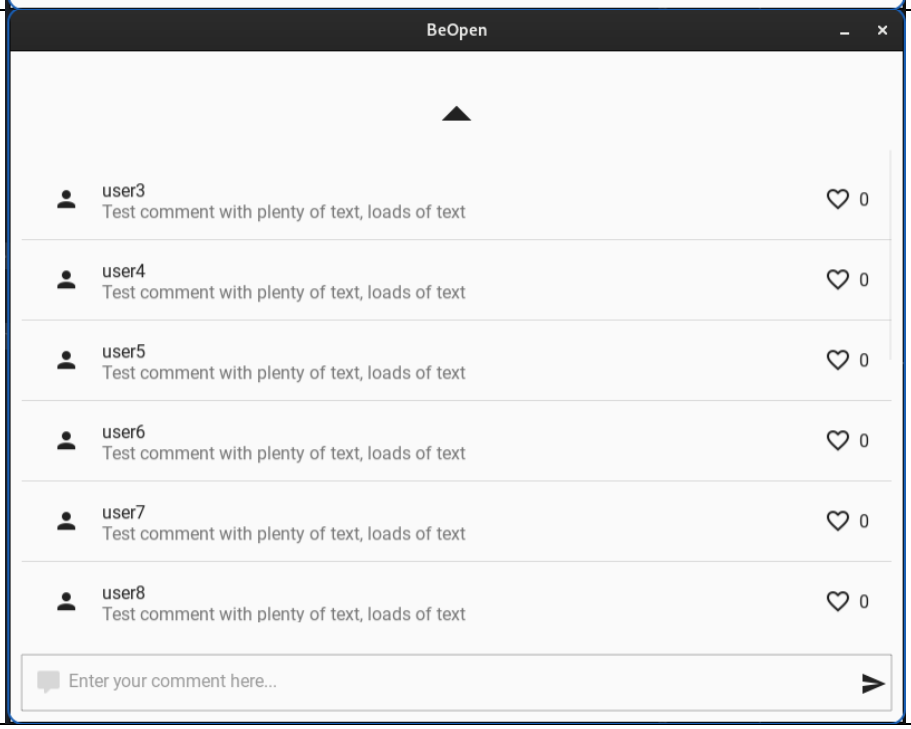
C.1C.I	1	 BeOpen user14 Test comment with plenty of text, loads of text 0 user15 Test comment with plenty of text, loads of text 0 admin Hello this is a standard comment that i am submitting as an "admin" user 2 admin Hello this is another comment to test the new comment add system 1 admin okay new comment 0 admin and another one 0 Enter your comment here...
C.2A.I	1	 BeOpen user2 Test comment with plenty of text, loads of text like loads of words to much to count to be ... 0 user3 Test comment with plenty of text, loads of text 0 user4 Test comment with plenty of text, loads of text 0 user5 Test comment with plenty of text, loads of text 0 user6 Test comment with plenty of text, loads of text 0 user7 Test comment with plenty of text, loads of text 0 Enter your comment here...

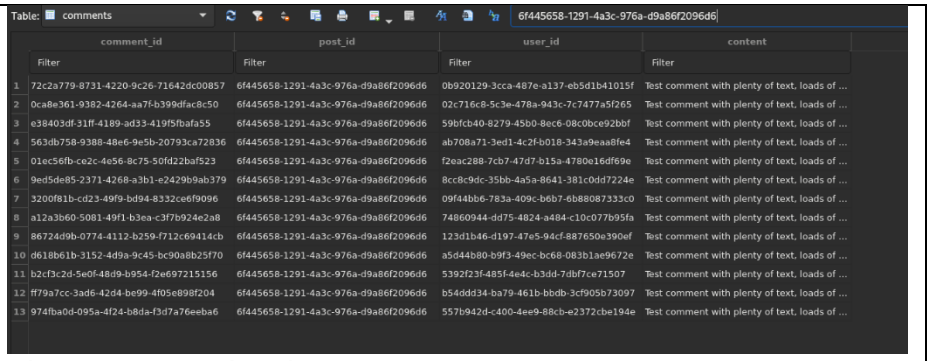
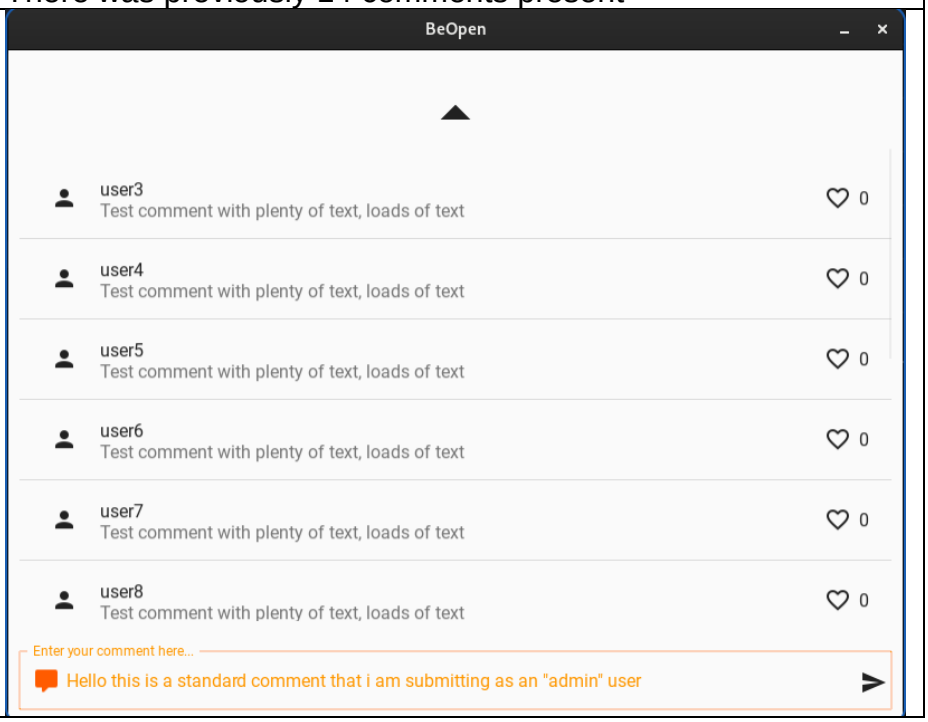
C.2A.I	2	
C.2B.I	1	
C.2B.I	2	

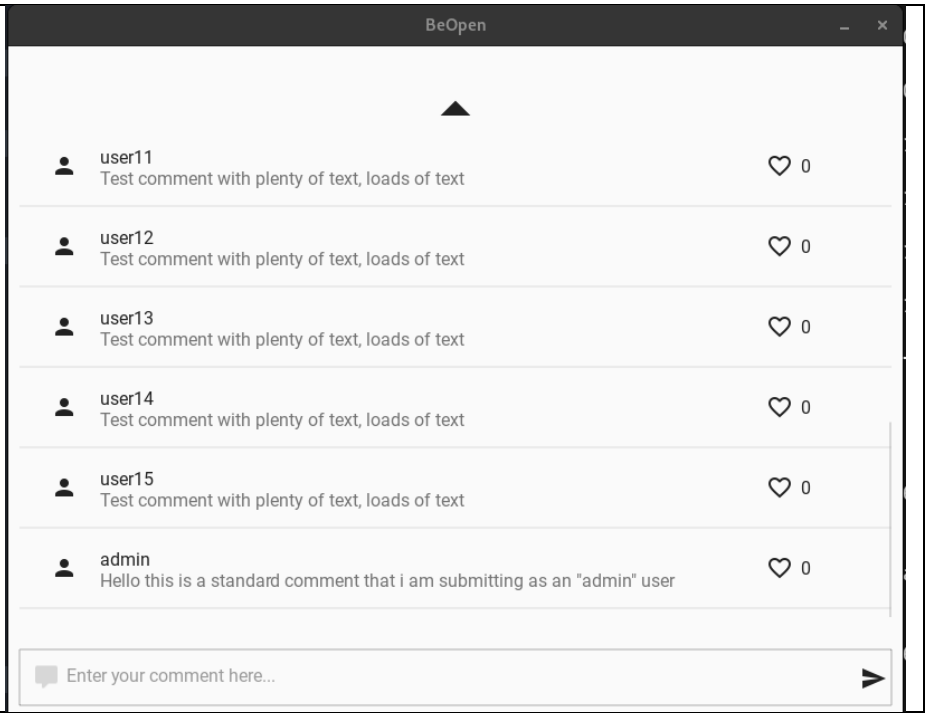
C.2B.I	3	
C.2B.I	4	
C.2B.II	1	

C.2C.I	1	
C.2C.I	2	
C.2C.I	3	

C.2C.I	4	
C.3A.I	1	
C.3A.I	2	

C.3B.I	1	 BeOpen user2 view profile delete comment 1 0 0 user5 Test comment with plenty of text, loads of text 0 user6 Test comment with plenty of text, loads of text 0 user7 Test comment with plenty of text, loads of text 0 Enter your comment here...
C.3B.I	2	 BeOpen user3 Test comment with plenty of text, loads of text 0 user4 Test comment with plenty of text, loads of text 0 user5 Test comment with plenty of text, loads of text 0 user6 Test comment with plenty of text, loads of text 0 user7 Test comment with plenty of text, loads of text 0 user8 Test comment with plenty of text, loads of text 0 Enter your comment here...

C.3B.I	3	 There was previously 14 comments present
C.4A.I	1	

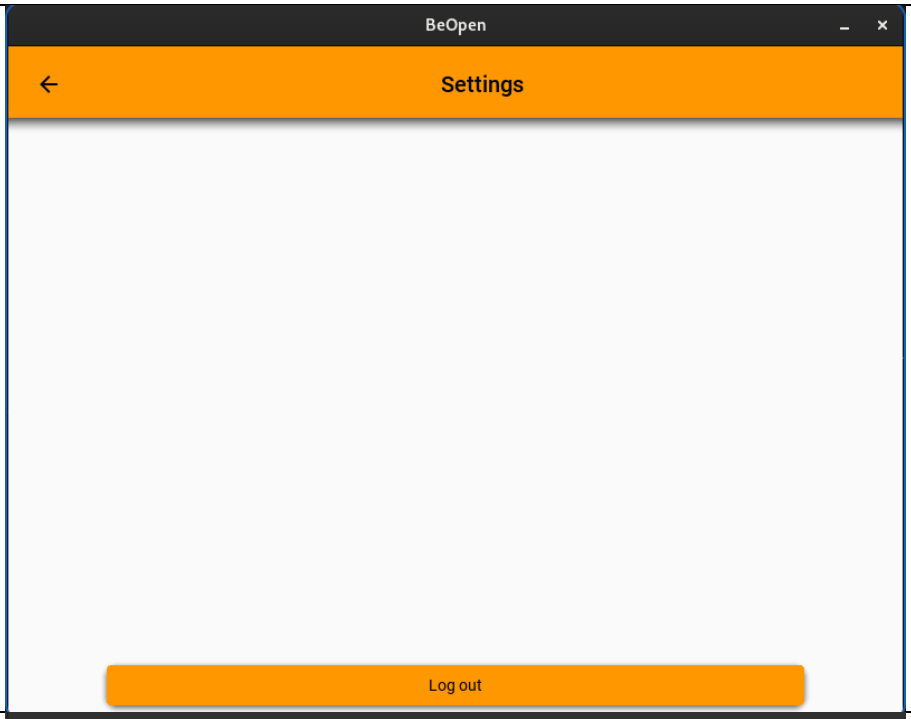
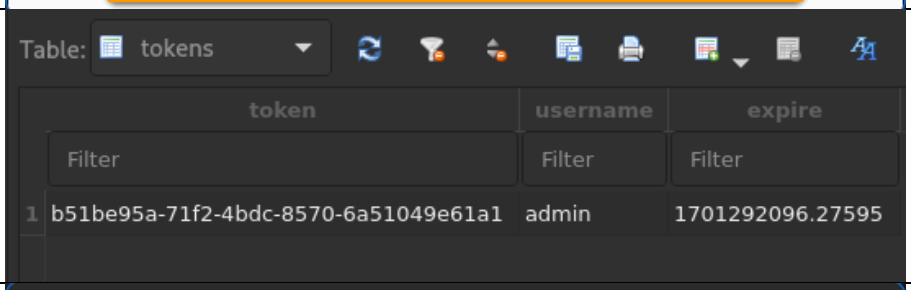
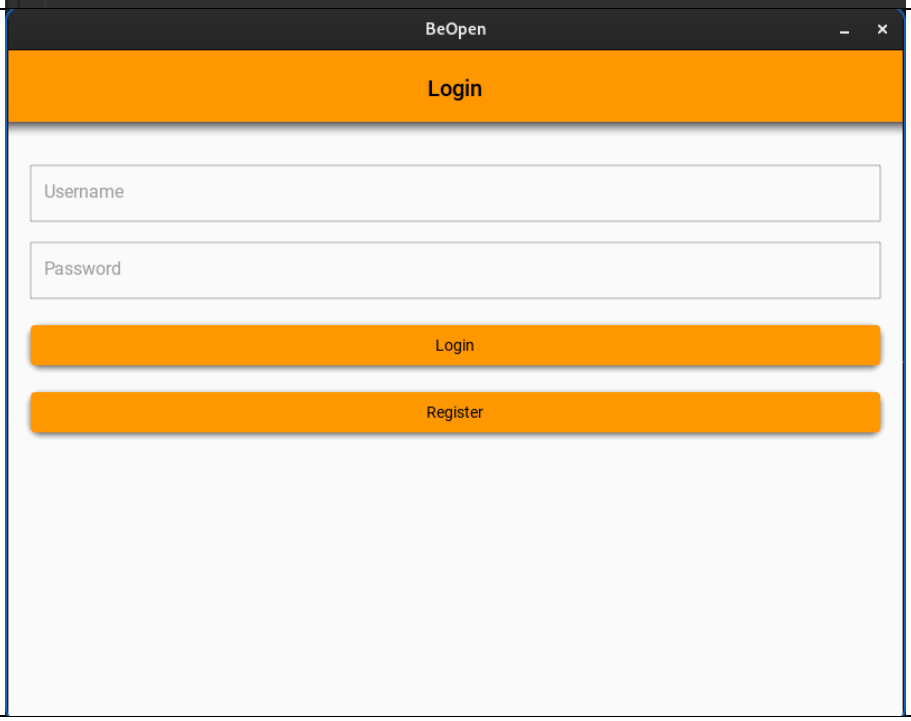
C.4A.I	2	

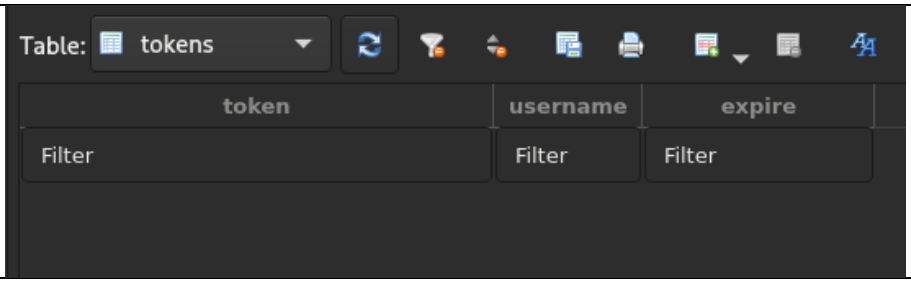
Settings

Test Number	Test Description	Expected	Observed	Action
S.1A.I	Clicking the settings icon at the top of the homepage	The page should switch to the settings page. There should be a top bar showing a back button and a title. Currently the only “setting” is the logout button so that should be the only thing display. The logout button should be at the bottom of the screen	As expected	

S.1B.I	Clicking the logout button	This should take the user back to the login screen and clear the authentication tokens from the client database.	As expected	
S.2A.I	General use after loggin out of Admin and logging in as user. Here im going to login as admin and then logout, next logging in to user and do some general navigation of the app on both accounts	There should be no errors, the main thing im looking out for is errors where there is duplicates of screens. This happens when screens are named the same thing in both sessions or aren't properly deleted when the other user logs out	Had small issue with accounts pages, they were using the old functions for switching screens and so were not properly removing themselves after use. This meant there was some overlap on the screens after users switched	This was quickly fixed and now works as expected

Test Number	Image Number	Image

S.1A.I	1	
S.1B.I	1	
S.1B.I	2	

S.1B.I	3	

Database Encryption

Test Number	Test Description	Expected	Observed	Action
E.1A.I	Launching the server with encryption and shamir secret sharing set to true. Minshares = 3 and numberofshares = 5. The master password being set is 520	The server should generate the shares into the specified share folder (via the config file). There should be 5 shares, 2 combinations of shares should have been tested, these tests should have been noted in the log. And the database should be encrypted stored at data/.cryptdatabase.db	As expected.	
E.1B.I	Launching the server with encryption and shamir secret sharing set to true. Minshares = 3 and numberofshares = 5. The master password being set is 625582934	The server should generate the shares into the specified share folder (via the config file). There should be 5 shares, 2 combinations of shares should have been tested, these tests should have been noted in the log. And the database should be encrypted stored at data/.cryptdatabase.db	As expected	
E.1C.I	Launching the server with encryption set to true. The	The database should be encrypted stored at the path data/.cryptdatabase.db.		

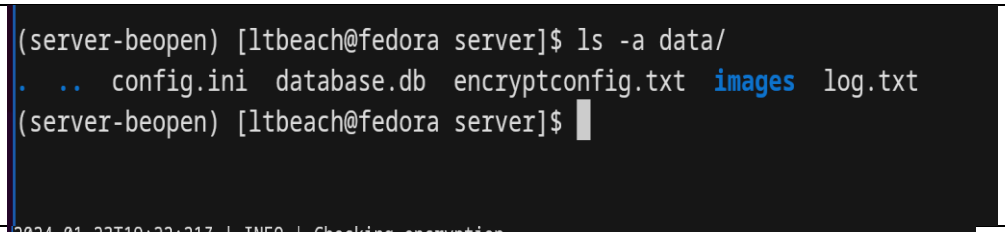
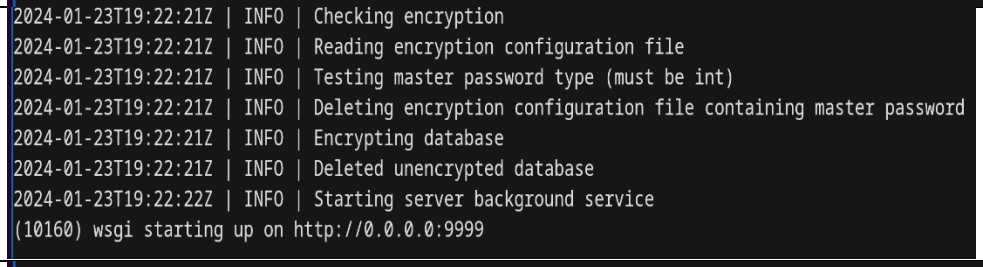
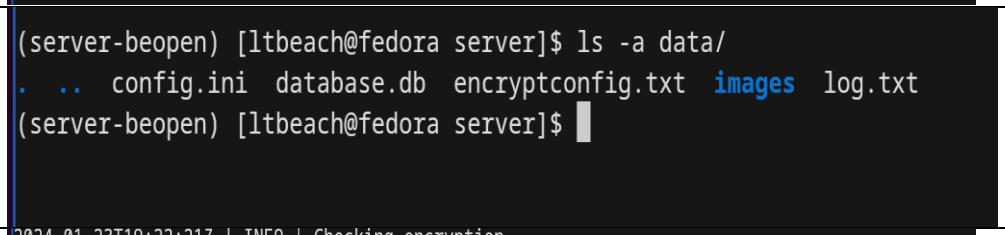
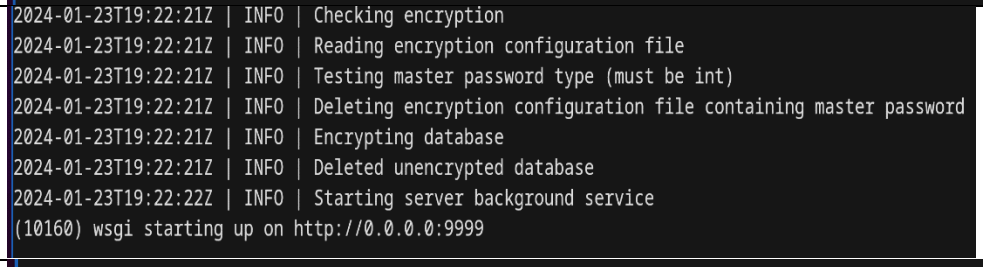
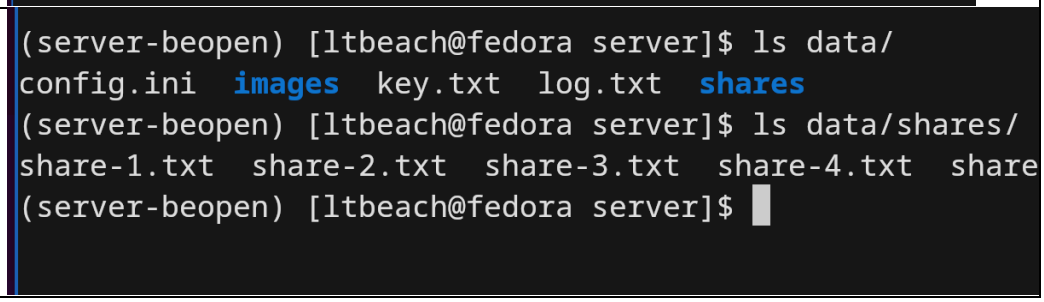
	master password being set is 625582934	The encrypt config should also be deleted and the server should be in decryption mode.		
E.1D.I	Launching the server with encryption and shamir secret sharing set to true. Minshares = 3 and numberofshares = 5. The master password being set is 520. However not providing a encryptconfig file.	The server should stop and the logs should show that the encryption config file could not be found at the specified path		
E.1E.I	Launching the server with encryption and shamir secret sharing set to true. Minshares = 3 and numberofshares = 5. The master password being set is 520. However the provided string in the encryptconfig path is the word “hello” (to be valid the master password has to be an integer).	The server should stop and the logs should state that the master password provided could not be read.		

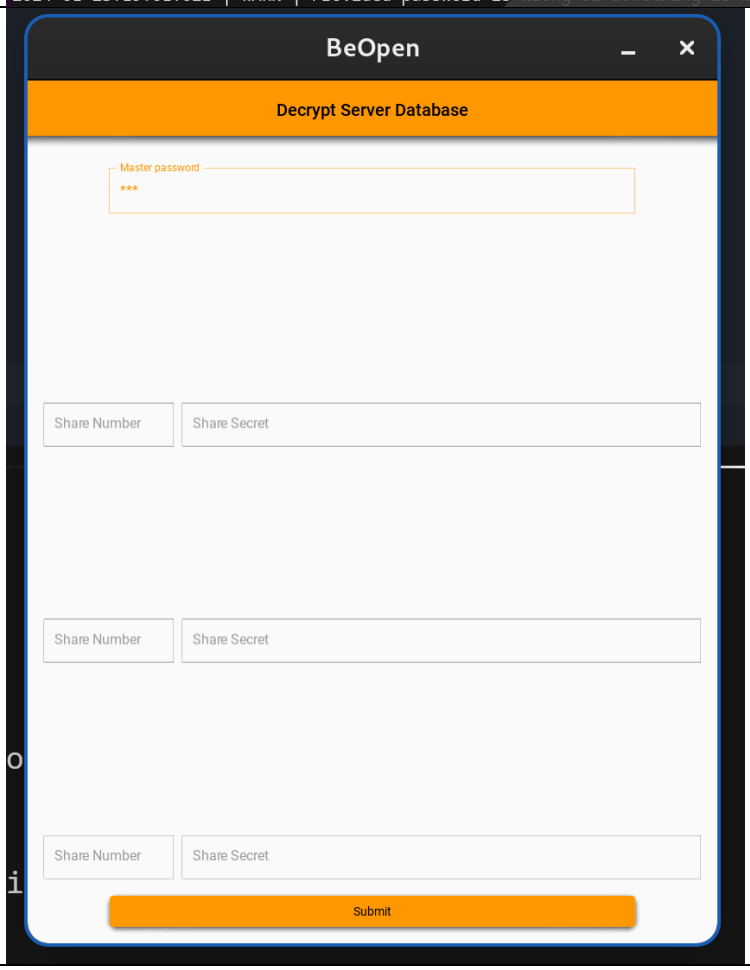
E.2A.I	Client test. Server settings: Encryption: enabled Shamir secret sharing: enabled Master password: 520 From a client the client after connecting to the server should bring up the decryption page. This time we will use the CORRECT master password	Client side: It should be successful and so bring us to the login page, from here you can continue to use the app as normal. Server side: The logs should indicate that the database has been decrypted. And the unencrypted database should be at the path “data/database.db”. It should be openable (here I use dbrowser to prove that the database has been decrypted). The server should enter normal mode		
E.2B.I	Client test. Server settings: Encryption: enabled Shamir secret sharing: enabled Master password: 520 From a client the client after connecting to the server should bring up	Client side: The input fields should error. Indicating unsuccessful decryption. Server side: The logs should show an attempt to decrypt the database had failed. The database should remain encrypted and the server in decrypt mode.		

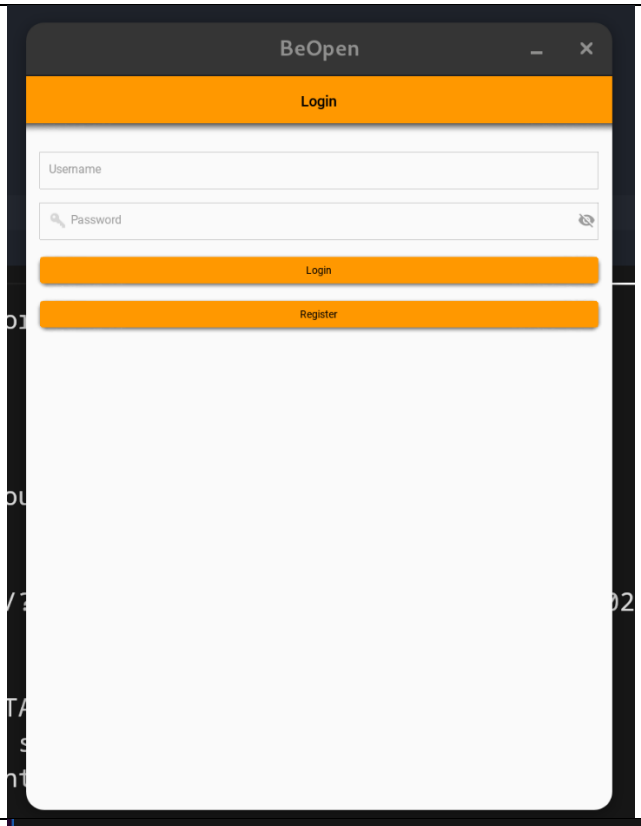
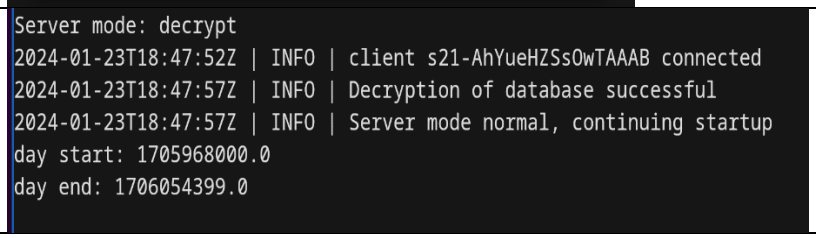
	the decryption page. This time we will use the INCORRECT master password			
E.3A.I	Client test. Server settings: Encryption: enabled Shamir secret sharing: enabled Master password: 520 From a client the client after connecting to the server should bring up the decryption page. This time we will a CORRECT set of shamir secret shares	Client side: It should be successful and so bring us to the login page, from here you can continue to use the app as normal. Server side: The logs should indicate that the database has been decrypted. And the unencrypted database should be at the path “data/database.db”. It should be openable (here I use dbrowser to prove that the database has been decrypted). The server should enter normal mode		
E.3B.I	Client test. Server settings: Encryption: enabled Shamir secret sharing: enabled Master password: 520	Client side: The input fields should error. Indicating unsuccessful decryption. Server side: The logs should show an attempt to decrypt the database had failed. The database		

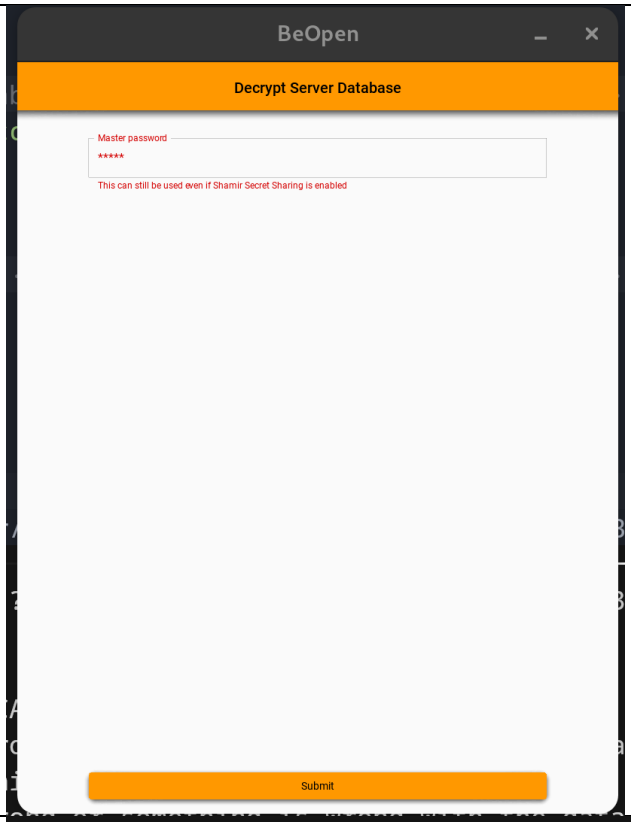
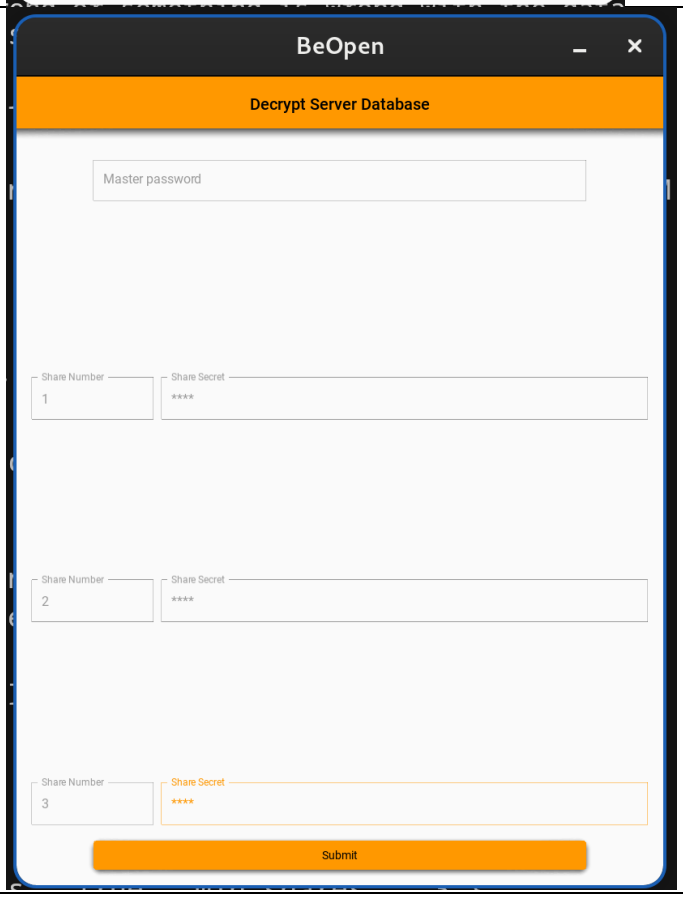
	From a client the client after connecting to the server should bring up the decryption page. This time we will an INCORRECT set of shamir secret shares	should remain encrypted and the server in decrypt mode.		
E.4A.I	Client test. Server settings: Encryption: enabled Shamir secret sharing: disabled Master password: 520 Just testing to see what inputs field the client dispalys. See previous test for inputting etc.	The client should only display the master password field.		
E.4B.I	Client test. Server settings: Encryption: enabled Shamir secret sharing: enabled Minshares: 5 Master	The client should display 5 input boxes for the Shamir secret sharing inputs. As well as the master password input box at the top and a submit button at the bottom.		

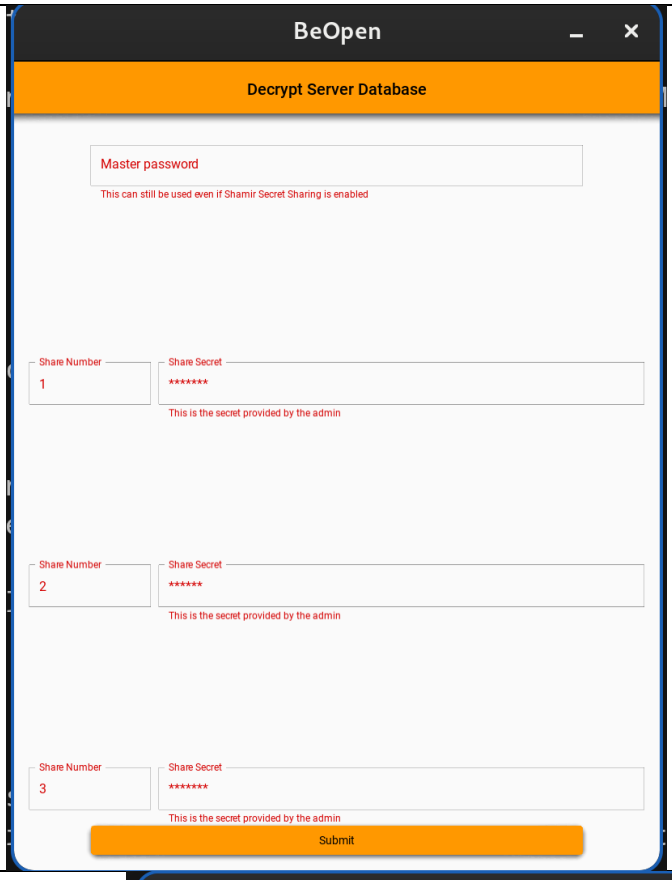
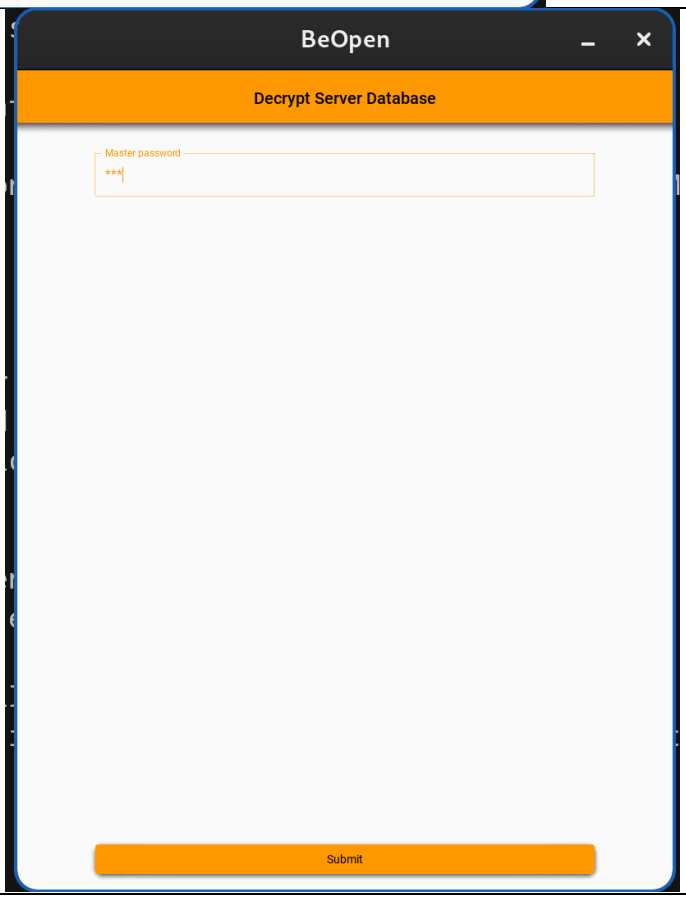
	password: 520 Just testing to see what inputs field the client dispalys. See previous test for inputting etc.			

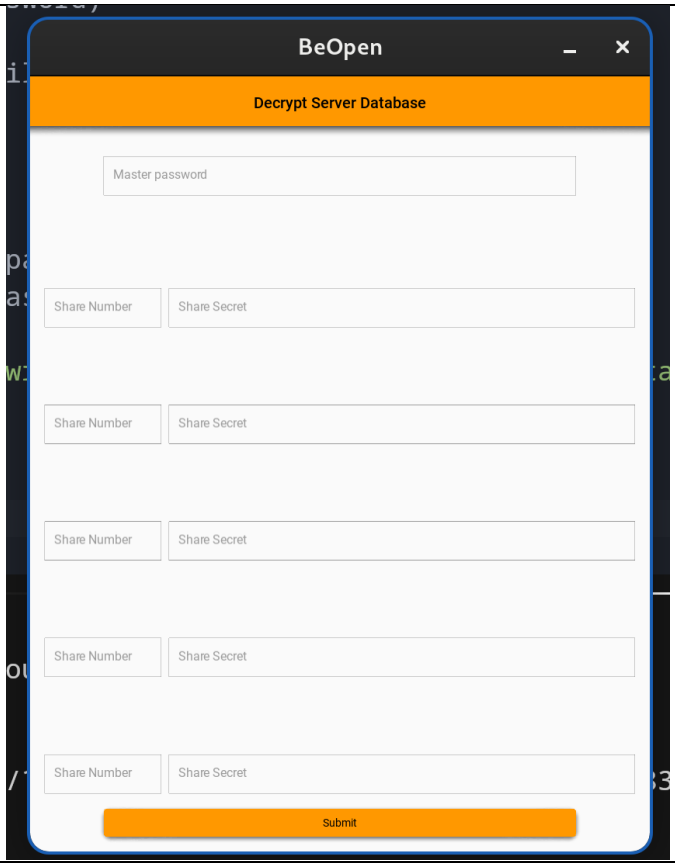
Test Number	Image Number	Image
E.1A.I	1	
E.1A.I	2	
E.1B.I	1	
E.1B.I	2	
E.1B.I	3	

E.1C.I	1	2024-01-23T19:22:21Z | INFO | Checking encryption 2024-01-23T19:22:21Z | INFO | Reading encryption configuration file 2024-01-23T19:22:21Z | INFO | Testing master password type (must be int) 2024-01-23T19:22:21Z | INFO | Deleting encryption configuration file containing master password 2024-01-23T19:22:21Z | INFO | Encrypting database 2024-01-23T19:22:21Z | INFO | Deleted unencrypted database 2024-01-23T19:22:22Z | INFO | Starting server background service (10160) wsgi starting up on http://0.0.0.0:9999
E.1D.I	1	2024-01-23T18:52:25Z | INFO | Ensuring server directories 2024-01-23T18:52:25Z | INFO | Ensuring config file 2024-01-23T18:52:25Z | INFO | Config already exists 2024-01-23T18:52:25Z | INFO | Ensuring database 2024-01-23T18:52:25Z | INFO | Checking encryption 2024-01-23T18:52:25Z | FAIL | Encryption config could not be found at data/encryptconfig.txt 2024-01-23T18:52:25Z | FAIL | Could not generate encryption scheme, something wrong in config file or 2024-01-23T18:52:25Z | FAIL | Could not encrypt database, something went wrong, see logs for details (server-beopen) [ltbeach@fedora server]$
E.1E.I	1	2024-01-23T19:00:57Z | INFO | client AjU6h18KvwtxiZXCAAAB connected 2024-01-23T19:01:01Z | WARN | Provided password is wrong or something is wrong with the database key 2024-01-23T19:01:01Z | FAIL | Something went wrong while decrypting the database 2024-01-23T19:01:02Z | WARN | Provided password is wrong or something is wrong with the database key
E.2A.I	1	

E.2A.I	2	 BeOpen Login Username Password Login Register	
E.2A.I	3	 Server mode: decrypt 2024-01-23T18:47:52Z | INFO | client s21-AhYueHZSsOwTAAAB connected 2024-01-23T18:47:57Z | INFO | Decryption of database successful 2024-01-23T18:47:57Z | INFO | Server mode normal, continuing startup day start: 1705968000.0 day end: 1706054399.0	

E.2B.I	1	 BeOpen Decrypt Server Database Master password ***** This can still be used even if Shamir Secret Sharing is enabled Submit	
E.3A.I	1	 BeOpen Decrypt Server Database Master password Share Number 1 Share Secret **** Share Number 2 Share Secret **** Share Number 3 Share Secret **** Submit	

E.3B.I		 BeOpen Decrypt Server Database Master password This can still be used even if Shamir Secret Sharing is enabled Share Number 1 Share Secret ******* This is the secret provided by the admin Share Number 2 Share Secret ****** This is the secret provided by the admin Share Number 3 Share Secret ******* This is the secret provided by the admin Submit	
E.4A.I	1		 BeOpen Decrypt Server Database Master password *** Submit

E.4B.I	1	

Final product video testing

Test number	Name	Description	Link
1	Admin 1	This clip shows how an Admin would go about registering their account, creating a number of occupations, altering some of their profile information and even set a team leader for the IT staff	https://youtu.be/S7qmnO0pqMo
2	Admin 2	This clip shows an Admin: logging in, accepting some occupation change requests and adding some team leaders to the Students team	https://youtu.be/hx0EX4n52wU
3	Member 1	In this clip a member creates an account, sets up some basic information, sends a friend request and creates an occupation change request. They also look at some provided help boxes	https://youtu.be/iFrqAxIeeTY

4	Member 2	In this clip a member logs into an account, changes some profile information, accepts a friend request, sends a friend request to a recommended friend and looks at their team	https://youtu.be/vvCI4Xd5Rk0
5	Member 3	In this clip a member: logs into their account, likes, comments on some other peoples posts. Then creates their own post with a caption. They also view some of their memories from a post they made a previous day.	https://youtu.be/cwaT_C9WV0I
6	Member 4	In this clip a member: Plays with the profile page closing info change panels (attempting to cause a bug), looks into someone else's profile, deletes and views some notifications, unlikes some posts and comments, changes the settings and the logs out	https://youtu.be/ChVRb_hM6Bg
7	Server 1	In this clip I show how an admin may go about setting up encryption on the database. I first show the guide that's available in the "docs" directory. Then I follow the process, creating an encryption config containing the master password, turning encryption on. I also show how the new database cannot be opened by a program used for viewing SQLite databases. Proving it is encrypted. I then unencrypted the database and again open it with the same program where it is successful. I then log in admin showing that the server is functioning as formal	https://youtu.be/rYbMYhRXPaw
8	Server 2	In this clip I show how the server database would be decrypted using Shamir secret sharing and how the server would re-create the master password in the filesystem. Then I prove the	https://youtu.be/vMhg5U_kdCM

		database is working as usual by logging in as an admin.	
9	Server 3	In this clip I show how the server database being permanently decrypted by setting encryption to false in the config before re-launching the server and then decrypting it one last time. I also show the database is decrypted and works as usual both using a program to view SQLite databases and logging in to the server.	https://youtu.be/VEp7ruA1FME

In case any links do not work for some reason all the showcase videos are in the playlist below: https://www.youtube.com/playlist?list=PLKQHu0l6LmP-OwUzidnlO_bxiQ3O1Ux2e

Evaluation

Potential user trials (pre-improvements)

Finley

Trial 1

Video of trial: https://youtu.be/bXM083W5ZE8

Finley is a very tech competent person, immediately he had no problem registering his account however noticed that he could see his password in full view. He would prefer if the password was hidden as he typed it in (both in the login and registration page) with a button to reveal it if he wanted to. He then went on to the main homepage since he wasn't part of any teams and had no friends, he couldn’t see any posts. Finley then immediately went to go make a post and did so with ease he said that making a post was intuitive and caused no friction. After posting he quickly said, “How do I see other people’s posts?” and began looking at the other tabs. He found the memories tab and found his own post again he found no real friction in this process. However, he couldn’t figure out how to see anyone else’s post and we ended the trial there.

Overall, though due to the application providing no explanation to the user on how to add friends or join a team he couldn’t engage with the main function of the system, viewing others posts. He also wanted passwords to be hidden by default when typed.

Things to be added as a result of this test:

- A first-time login page, this should help a first-time user do some basic things like join a team, set their name, and set their role.
- Password fields will be updated to be hidden by default with a button to show.

Izumi

Since Finley’s trial a first-time login page has been added to guide the user through setting up some basic information.

Trial 1

Video of trial: https://youtu.be/QOs6MUuFJNQ

Izumi had no problem registering the account however when she asked what the registration key was, I realised that a note or hint about how a user should get their registration key from their admin could be useful. Then she easily logged in, and since the Finley trial, a first-time login page had been added. This then prompted Izumi through setting her name and role. However, there was minor confusion about what a team was, so a better explanation to the user may need to be added to this screen. Izumi immediately went to create a new post again this was done with no friction or confusion. We stopped the trial here due to an interruption.

The main takeaways from this trial:

- A better explanation of teams needs to be added.
- Izumi didn’t like that she couldn’t see her own post in the homefeed

Trial 2

Video of trial: https://youtu.be/qaA-BGUFL2U

Izumi logged in to the account she created in her first trial and her occupation set request was accepted by an admin (me). This allowed her to view posts from the other 3 people in her team, Coops, Dan and santi. But first she clicked on to her account changed her bio and then went on to accept a friend request from coops. Then she went through liked and commented on the posts she could see. This time her own post appeared in the feed; however, she said it would have been better if it appeared at the top of the feed like in other social medias. At the end she went back into her profile page and attempted to change her profile picture, at which point I informed here this was not an option currently. The trial ended here.

Main takeaways from this trial:

- A user's own post for the day should appear at the top of the home feed.
- Izumi wanted to be able to add her own profile picture.

Improvements

Of these trials, 5 possible improvements were suggested. I'm going to go through each of them in more detail.

First time login page

This is relatively easy to add since it's just a simple page with a few input boxes. It's going to include adding a name, role creating an occupation change request (to join a team). For each of these things it's going to include an explanation of what each one is and at the end the user can hit "done" to go to the homepage. The page also includes an explanation of how to make a friend request.

Password fields

Hiding the inputs of the password field with a button to reveal is another easy implementation with huge upside to the user. This requires simply making a new custom text box widget and replacing the standard password text boxes with this widget.

Own post in homefeed

This is something that was easily added since all I had to do was remove the function that prevented this happening on the server side. I presumed at the time that this feature would be un-desirable but like Izumi noted it's a standard function in all other social media and so was re-added.

Making the post appear at the top of the feed however was more difficult. But was done since it was still a very low complexity task. It was to be performed on the client side since it was a UI focused issue. Before adding all other posts, the client now reaches out for the users own post. It adds this post the top of the post stack, and then carries on as normal.

Profile picture

This is a feature that is unlikely to be added for several reasons. The first reason is its complexity, the feature would require additional server-side code, including new profile events and class methods, and an update to the database tables to allow adding of an image path. It would also require significant client-side code. To properly receive and save the image as well as take/upload profile pictures.

The second reason is due to it not adding much to the user experience. These profile pictures would only be displayed on profile pages, and not at the top of a user's posts (due to UI library constraints). Generally, the benefit to the user is easy recognition of other people via their picture but since you would have to click into their profile to see their picture user's might as well just read the username (displayed next to posts, comments etc) or full name of the user (displayed in the profile page).

Overall, the effort put in to adding this feature would outweigh the benefits to the user.

Explanations

To better explain the system to the user several pages will now have a "?" at the top as a clickable button. A number of these "help" buttons will be added to some input fields to, for instance on the registration key input box. This button should help and explain some key parts of how the platform functions to the user. On clicking this button, a box will pop up containing an explanation, after being read this box can be dismissed.

These help boxes may even be added as a setting to be turned on and off. So once a user understands the platform they can go to settings and turn off the help buttons.

If for some reason the video links do not work all Trial videos are also in the playlist linked here: https://www.youtube.com/playlist?list=PLKQHu0l6LmP_QRCdAfwE0z0nlIFkgVu6W

If done again

If the system was to be designed again from scratch a few parts would be reworked and, in some cases, done in completely different languages.

Client UI

If I were to start with a clean slate, I would likely build the client utilising native Kotlin or Flutter. This is because while the python library Kivy does compile to an APK and run on an android phone (and in theory an iPhone), its not very performant. That is it can take a while to load and the app can feel sluggish. The python library runs non-native code by creating a python environment within the running app that allows the python code to run and then communicate with a java interface that then runs the actual code to generate the widgets. This means to simply display some text on the screen:

Python creates widget -> Java interface -> Java creates widgets -> Widget is displayed.

This is manageable on newer devices but when it comes to older devices the app could border on difficult to use if the user is part of a large organisation.

In my testing I used 3 different devices, I will briefly cover the performance and feel of the app on each. The first “device” is used was the pixel 13 pro emulator (android 13/14) on android studio, this was a virtual android device running with roughly the same specifications the actual phone itself would have. On here the app ran well generally, except for when generating “pop up lists” these lists are generated on the fly since their contents can be changed by another user at any time (such as if an admin deleted an occupation). The Kivy documentation does not recommend this due to this widgets particularly bad performance but ultimately, I had no choice. Even then to the user this simply appears as the pressing of a button not being very snappy. I also ran the app on my own phone a Motorola edge 20 (a low-end, 2020 phone, on android 13) and it also ran fine again apart from the drop down menus (again). I also ran it on a Samsung galaxy a5, 2017 (low end phone running android 11).

It was also hard to develop for since compiling from python to a native android APK does take significantly longer than compiling Flutter or Kotlin. So basic things like syntax and logic errors could be tested on a computer, however the look of the UI on a phone and the long compile times associated created a huge time sink. Faster UI development could of allowed for more experimentation and possibly a more intuitive design.

Flutter would probably be preferable for a redesign since like Kivy, flutter supports IOS, MacOS, Windows, Android, Linux but also additionally web. It also runs at near native speeds on every platform. This would make the apps a lot snappier and also allow for deployment of a web application. A downside to using something like flutter is I would likely have to write the socketio parser custom. Since flutter lacks a package for socketio and so I would likely have to work with raw HTTP WebSockets and the (poorly written documentation of python-socketio). But the overall benefits would be huge and once the parser is written it can simply be placed to work in the background, only having to be modified slightly if the server is ever updated to later python-socketio packages. Flutter is also written is a package that utilises the programming language Dart which is very similar to C syntax.

Status system

The current status system makes full use of WebSockets, while a client is connected, they can receive status messages, the client always has an open status receiver than handles and displays these status messages (where necessary) to the user. However, sometimes after calling an even the client needs to look at the status “level” (INFO, WARN or FAIL) to be able to determine if the result of the last even call was successful. With the open status

messages and multiple events and other server processes happening the last, status message from the last event can get lost. This means that additional checks need to be put in place.

If the system were to be designed again all events (even those who return no data) would have a callback that returns the last status message produced from that event. This would allow a client to perform actions purely based on this last status message as it tends to be a success or failure. This would allow for a slightly more streamlined client coding experience. All other features of the status mechanism would be kept the same just with this additional feature.

User service

The only other server-side part I would re-design is the background user service. Currently for every logged in client a background service is created, each of these services run on their own thread. This background service allows for the serving of notifications and updates without the client having to poll the server. This system works great as it reduces the clients' workloads and threads, however as a specific instance gets larger I would worry about the performance of every client having its own background service thread. These threads are not intensive as they only check for new notifications ever 5 seconds. But with enough and just the fact that you must have a thread per active client could soak up memory and clock cycles.

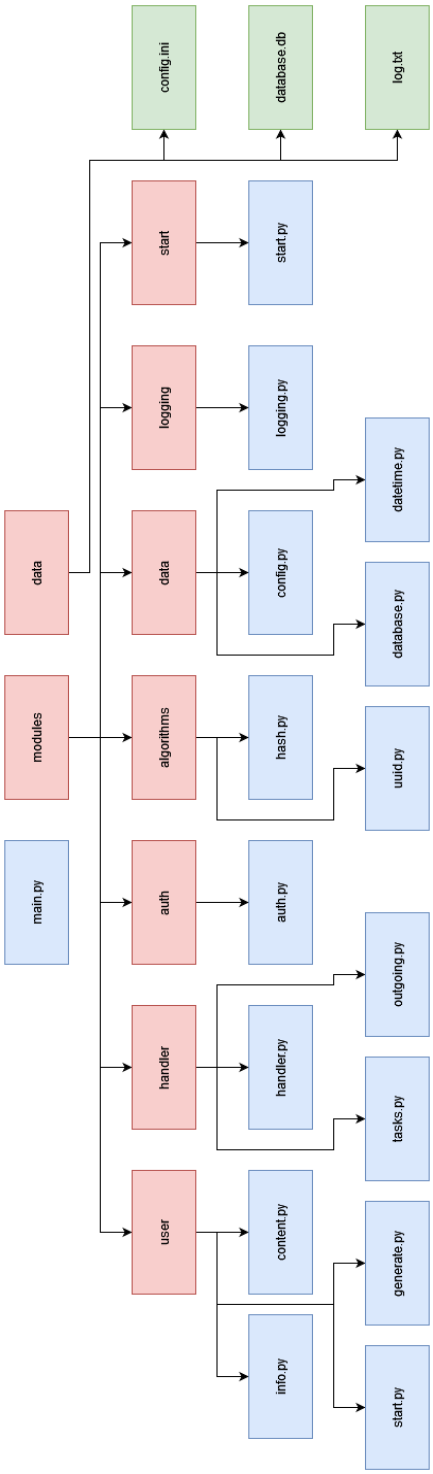
In a redesign of these background services, I would likely convert these into a singular background service or a handful. Perhaps having a background service active for every 10 active clients. So, if 30 clients were actively logged in and connected, 3 background services would be active handling their notifications and other "live" tasks. This would reduce the number of open threads and actually could be scaled as an option. Since for every client you have connected to one background service the less performant for the client while the more performant for the server. So higher grade server hardware could allow for less clients per background service. The reason this is less performant for the client is because these background services run on a single thread. So if client A and client B both needed to receive notifications, but client A has slower internet and is in the queue before B the background service would have to wait for the transfer to fully complete for A before moving on to send the notification for B. Basically you can have 29 people connected to one background service who all have perfect internet but if just 1 person joins that background service with slow internet notifications become slower for everyone.

However, to put this in perspective notifications generally consist of a timestamp, a title and a description (essentially the data to be transferred is very small) and so when we say "less performant" we only mean at most a fraction of a second. So having a background service handle even a few dozen clients should be completely fine and performance differences will be unobtrusive.

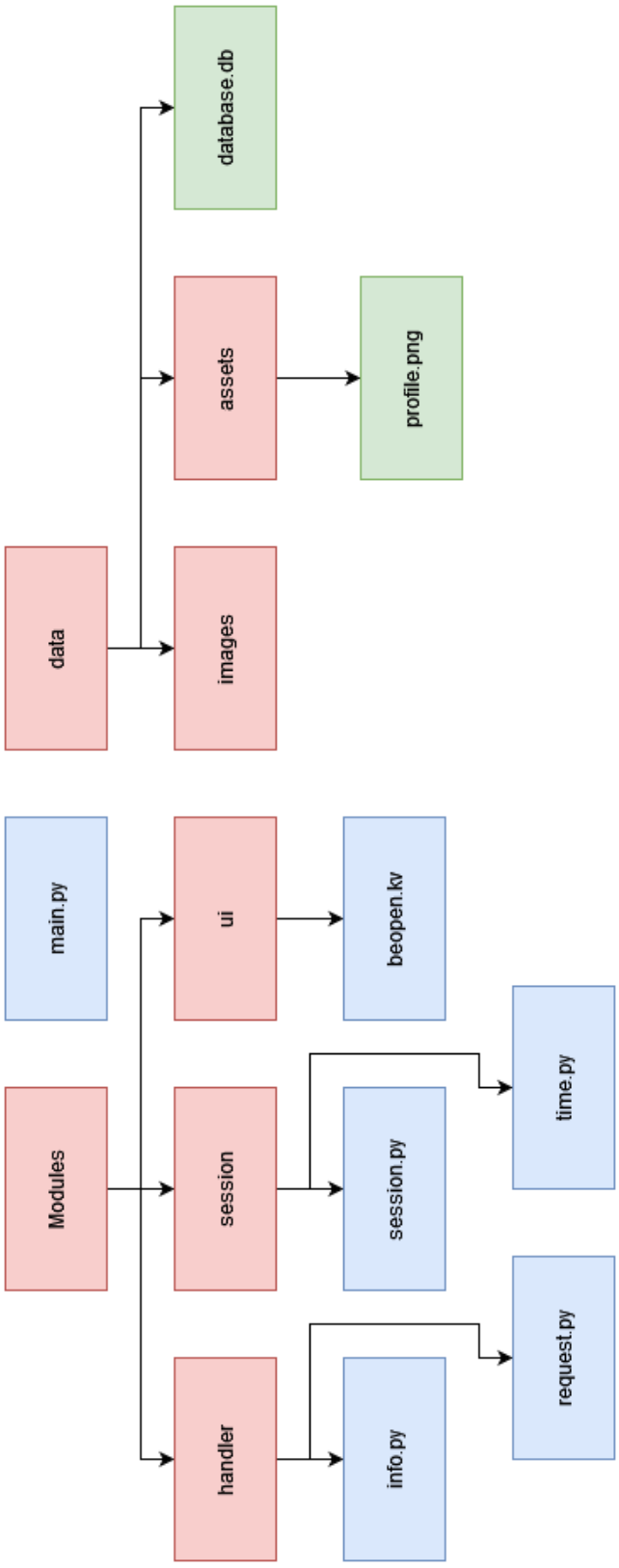
Code

File structure diagram

Server



Client


database.db
assets
profile.png
data
images
main.py
ui
beopen.kv
Modules
session
session.py
time.py
handler
info.py
request.py

Techniques

In this section I will lay out exactly what techniques are used from the top band (group A) of the AQA NEA mark scheme (technical skills section). Please note that this does not encompass all techniques used and in many cases these techniques are used on such a large scale (for instance the class structures) that I might reference many points. I will also note when the references I've given are just examples of the many times the techniques are used.

Algorithms

Cross-table parameterised SQL

- See SQL and database section of the write up for many examples of this.
- Server/modules/user/info.py and server/modules/user/content.py are the 2 main files that deal with almost every table in the database. Here you will see many examples of joins and complex aggregations being used

Aggregate SQL functions

- See SQL and database section of the write up.
- Also see sever/modules/user/content.py, line 979. In the impressions class the "count" method utilises an SQL count aggregate function.

User/CASE-generated DDL Scripts

- See the DDL section of the SQL and database section of this write up
- Or see server/modules/data/database.py for the code that creates the databases
- Here you will find all the generation scripts for the database. The database is made up of 15 tables all of them have at least 1 foreign key connection to another, some tables are almost entirely foreign keys. There is only one table without any sort of connection that being "time_slots".

Graph Traversal

- See under the algorithms section "friend recommendation algorithm". It utilizes a graph traversal to find common friend of friends and friends of friends of friends etc.
- Or see server/algorithms/recommend.py, lines 47 to 137 is the "Graph" class that stores the graph and performs the traversal.

Queue operations

- See under the algorithms section “friend recommendation algorithm”. A queue is used to facilitate a breadth first search of a graph. This includes queueing and dequeuing items.

Hashing

- See under the algorithms section “username hash”. This is used in the friend recommendation algorithm to generate hashes for peoples usernames. This allows theses data points to be put into the graph and stored in a hash map.
- Or see server/modules/algorithms/hash.cpp, this cpp file performs that actual hash algorithm itself but to see how this algorithm is put to use see: server/modules/algorithms/recommend.py, lines: 47-137

Advanced matrix operations and Complex mathematical operations

- See under the algorithms section “Shamir secret sharing”.
- Or see server/modules/data/sss.cpp this file contains all the matrix operations and polynomial mathematics used to generate polynomials given a y intercept and solve polynomial y intercept problems by reconstructing the curve of power n-1 using n points.

Recursive algorithms

- Under the algorithms section: “Shamir secret sharing” specifically the “det” function, “Friend recommendations” specifically the graph generation and the “__add_user_friends” function. These are just 2 examples, the Shamir secret sharing algorithm actually has several instances of recursive algorithms.
- Or see: sever/modules/data/sss.cpp and server/modules/data/recommend.py you will find the previously mentioned functions in these files along with other recursions.

Complex user-defined algorithms (scheduling)

- Algorithms section: “post scheduling and time slots”
- Or see: server/modules/data/datetime.py the “timestamp” class deals with the generation of post time slots, considering the length of days etc. The server/modules/handler/tasks.py handles reading the current time and comparing it to the slot.
- The client also has some slot management, it polls the server once for the time slot and uses a “kivy clock” to schedule a post button appearing. See: client/main.py line 552, and in edition 921 and 977 to see how scheduling is also used to repeatedly update the time widget on the post creation pages.

Complex user-defined algorithms and Complex mathematical operations

- UUID generation, see algorithms section: “UUID generation”. It uses binary and hex string mathematics alongside pre-assigning bits to generate Universally Unique IDs.
- Also see: server/modules/algorithms/uuid.py

Merge sort

- Algorithm section: “merge sort”, its used for post sorting on the server side. Its implemented to allow for sorting to be done on the server side rather than relying on a thin client. Posts are sorted by like count.
- Or see: server/modules/user/content.py lines: 691 to 726

Dynamic generation of objects based on a complex user-defined use of OOP model

- The main files containing large use of objects on the server side is: modules/user/info.py and modules/user/content.py, modules/handler/handler.py also makes extensive use of objects and passing them as attributes of other classes. Modules/algorithms/recommend.py also uses objects to generate the graph and each user generates subsequent user objects and adds them to its friends list attributes.
- On the server side you should look at main.py, here there is extensive use of objects due to all UI being handled through objects. This means heavy scripting how objects are added, passed to children of that object and so on. For instance page switching in the UI and going “back” is often done by passing the parent object to the child so they can call switch(parent).

Server-side scripting using request and response objects

- The entire model is a server client. All requests and responses are packaged the same way.
- Each request from the client contains a dictionary simply called “data”, this dictionary will contain key value pairs corresponding to data needed by the event called.
- All responses to client are also very simple, the server always returns: “True, info”. True being the Boolean value and info being a dictionary as well. This ensures that the server doesn’t error simply didn’t receive the correct number of items passed. The True simply states to the callback client function that what it is receiving is a callback from the server.
- Clients can call events on the server for complicated processes or just when they need some data or change some information on the system. This is done through a number of pre-defined server events. For instance instead of the client getting all

the information about friends and then generating a recommendation client side, it simply calls the friend_recommend event and the server does the computation.

Model (data structures)

Most data structures are also outlined in the data structures section of this write up.

Complex data model in database

- See database and SQL section for the server database and table structure
- The database structure results in almost every table having foreign keys, there are also a number of link tables, look at the database diagram to see where these exist.
- The DDL scripts show in the document also dictate where these foreign keys are how their behaviour varies (on delete cascade, on update set null etc).

Hash map

- Used in the friend recommendation algorithm as “friend directory”, Its searchable using the hash of a username. Look at the “hash map” section for more information
- Or see server/modules/algorithms/recommend.py lines 47 to 137 for the hash maps creation and use.

Queue

- See the data structures section or (for the best queue implementation) see server/modules/algorithms/recommend.py

Graph

- See the algorithms section for how the graph is used and generated for the friend recommendation algorithm, See the data structures “graph” section for more on the generation and why its was used.
- Or see: server/modules/algorithms/recommend.py

Files organised for direct access

- On the server and client side images must be saved, for the client side images must be saved locally before they can be displayed on the UI. While on the server side images must be saved to allow access to them by clients.
- The data structures section goes in depth about how this is done. Or see server/modules/user/content.py, lines 461 to 475 and the rest of the “post” class for how images are read, sent and stored.

Complex user-defined use of object-oriented programming models

- See the class diagram sections to see how classes are structured, inheritance and use both composition and aggregation.
- An example of composition is used in the graph and user relationship (users are part of the graph). An example of aggregation is in the handlers section (handlers cannot exist independently of their corresponding table classes) but their table classes can exist independently of the handers. Similarly with logs and status, status messages can exist separately from the log, the log cannot exist separately from the status.
- Looking at the class diagrams you will find more examples of this.
- Inheritance is also heavily used throughout, as well as objects being attributes of other classes, and class references being attributes of objects.
- The main files to look at (all on server side) modules/user/info.py, modules/user/content.py, modules/handler/handler.py, modules/algorithms/recommend.py, modules/track/log.py and more.

File descriptions

Server

main.py

This is the so-called “root” of the program, this file mainly just handles interaction with the web-socket connection and its events. Here event decorators are used to create events that clients can call. Here data comes in and information (where necessary) is returned to the client through callback functions.

Almost all events consist of taking 2 defined arguments “sid” as in server ID (an ID used for the span of a client connection) and “data”. “data” is typically a dictionary and is the inputs from the client. Then most events defined in this file will call a type of “handler” such as “post_handler” pass it some basic security arguments, such as the minimum level needed for access to the function, the event name (for error messages) and user inputs.

Then if the event needs to return some information it will have a return line that always returns 2 pieces of information the bool “True” and “info” again info is returned from the handler method and is typically a dictionary, or if an error occurred with the users inputs “None” will be returned.

handler/handler.py

This file contains all the “handler” classes that inherit from the base class “root_handler”. All these classes handle user inputs, status messages and security. Inputs are assigned systematically through use of a verification function to check a certain dictionary key exists in the inputs, then is assigned to an object. Each assignment also handles its own security

allowing for more nuanced security through an “authorisation” method inherited from the root handler.

The root handler also contains the most important method “handle”, this method manages the immediate assignment of a user_id to the obj as well as managing what data the client wants returned by managing the obj.columns that exists in every table class. The root class manages the basic security of what levels have access to which events.

Handlers ultimately filter user input and mange security but all processing and database modifications is done in other modules.

handler/outgoing.py

This file handles any outgoing data from the server that is not activated from a client even. Basically a client also has a set of events that can be called by the server for instance for receiving a notification or status message. Outgoing event calls like this from the server are done through functions in this file. To minimise the surface area of the system.

handler/tasks.py

This file contains all the background operations on the server that are separate from the main event loop. When the server first starts a background service starts for managing the deleting of expired notifications and the notification of post time. It runs every 10 seconds and is modular so any additional background tasks that need to take place in the future can be added.

Another main background task is one that is started whenever a new client joins and logs in. This background service is responsible for serving real time notifications and updates to the client. This allows real time notifications without the client having to devote recourses to polling the server for notifications as this can consume large amounts of battery on certain devices.

user/info.py

This manages all tables that relate to a user’s personal information a list is below:

- **table** – A root class for all other classes that relate to tables in the database
- **user_id** – Non-accessible to handlers, refers to the single attribute user_id
- **auth** – *auth_credentials* table
- **profile** – *profile* table
- **friend** – *friends* table
- **occupation** – *occupations* table
- **team** – *teams* table

Each of the above is a class for a table so each can be addressed as an object. These classes aggressively use python class “properties” which are equivalent to the “getters”

and “setters” in other languages. The getters don’t do much in most of these attributes but the setters filter inputs. These filters act as a additional defence against SQL injection (despite using prepared statements anyway). But also help the methods themselves handle errors and output correct status messages, if a value is found to not be valid in a setter the attribute is set to None instead.

Most classes here have a number of public methods that all correspond to events available to clients. Here permissions and security is abstracted away through the handlers.

user/content.py

This file is similar to the info but instead all classes here refer to the content that users produce. A full list of classes is below:

- **user_content** – a base class for all other content classes in this file (inherits from table)
- **post** – *posts* table
- **comment** – *comments* table
- **impression** – *impressions* table, a base class for post and comment impressions
- **post_impression** – *post_impressions* table
- **comment_impression** – *comment_impressions* table
- **notification** – *notifications* table

All the classes here act in the same way as info, using getters and setters to filter inputs and security abstraction through the handlers that mange the security side of things.

user/generate.py

These functions are used for generating a fresh user. This happens once a user has registered creating a friend’s team, logging credentials in the auth_credentials table and a profile.

start/start.py

Manages the startup operation of the server and some first time setup operations such as creating the database, configuration file. It also creates a time slot for the day if one has not been created.

logging/logging.py

Contains 2 classes, geared towards logging and status messages to clients. The logging class can read and write to the logs, all statuses are written to the logs as well alongside additional information to assist in debugging.

data/config.py

Manges the configuration file. It programmatically creates the configuration file and provides a proper interface for the rest of the server to read the configuration file.

data/database.py

Manages the programmatic creation of the database. Also has a connection class to make an easy interface for the info and content classes to execute commands to the database. It manages the connection, the cursor and the automatic committing of any SQL command to lighten the load on other classes.

data/datetime.py

Contains the timestamp class. This class is used to manage all timings within the system. It coordinates the timings of the client and the post slots. It also can be called at any point to get the exact Unix timestamp using the property *timestamp.now* and *timestamp. Date*. It also creates the post time notification and abstracts the complexities of assigning a post time and retrieving it.

data/sss.cpp

This file contains the logic for Shamir Secret Sharing. This is only used if enabled in the config but is mostly made up of matrix operations, all these operations in the correct sequence can invert a matrix of size n x n.

auth/auth.py

This file manages all registration, login and authentication token creation. Most security functions are present in this file and are used for authentication of users before they can run an event. It also manages the creation and distribution of authentication tokens; these tokens can be stored by clients and have an expiration date on them. After this date a client will be forced to log back in and get a new authentication token.

algorithms/recomend.py

Contains the logic for generating friend recommendations for users. Containing a method and the associated classes, such as User and Graph. This is called from the friend class in the user/info.py.

algorithms/hash.cpp

Contains the hashing logic. Very short and this type of hash is only really used by the friend recommendation algorithm since its unsuitable for long strings. C struggles with large strings and large numbers and often suffers from memory overflows.

algorithms/uuid.py

This is where the function that generates UUIDs is. This function is used plenty of times throughout the code for the generation of almost all IDs. User ID is the most important one generated out of this since this also acts as a salt for the password. So secure and random

UUID generation is very important for the security of passwords. It also manages the hash function which is used for hashing passwords. This makes passwords safe for storage since if a database leak were to happen an attacker would have to devote large amounts of compute to get username, password pairs.

Client

main.py

This is where all programable UI management happens. It's a large file since it is impractical to turn most of this code into modules, since most methods are executing within widget classes and manipulating the UI.

Changing, adding, and updating of widgets all takes place in parent widgets and so on. This class-based approach to widgets allows complex structures and in many cases objects of parent classes becoming attributes of child classes. Small background tasks also often take place, most of these background or scheduled functions are for updating countdowns and managing the post slot.

ui/beopen.kv

This file is directly tied to the UI library used for the project, Kivy. This library makes use of some implicit links between a python class running in a main file and a "kv" file that contains the expressions for many widgets. This custom KV language is very simple and is designed to make creating widgets and adding static widgets to the UI easy. Then the main.py file will have classes of the same names as widgets in the kv file. Since their names are the same, they are implicitly linked at runtime which means programmatically adding widgets to the UI is done from the python code.

Overall, this file mainly manages the looks of base widgets and the arrangement of static widgets.

session/session.py

This session file manages the current user session. A new session is created for each separate launch of the app/when a new user is logged in. This session stores some basic information that can then be easily used throughout the program through a constant "session" object present in the main file. This session object contains basic information. For example, the username of the current user session, whether the session is logged in any present auth_tokens, the server code etc.

Session also plays a key role in receiving information from the server callbacks. Since session is used as a point of transfer for data. This allows the client to apply any filtering to incoming data as well as authentication of the information.

This file also contains the db class used for seamlessly interacting with the client-side database in a safe way. The wait class is also present this is very important for safely

awaiting data from a server callback. Without this class and its methods, if a server didn’t respond fast enough a client would continue with invalid data causing errors. Wait facilitates poor connections likely when a device is constantly moving around.

session/time.py

This manages the timing of the client and keeps it in sync with the server. This file allows the rest of the client to abstract away the handling of time data, post slots and date. Its most vital for the “memories” function of the client which must handle time and dates in a complex manner.

handler/info.py

This file manages any complicated data that is passed back to the client. For instance, images need to be decoded and stored on the client side. They also need to be dynamically removed to save on performance of the application. Through the info class the complexities of data such as images is abstracted away to the rest of the code making it as simple as any other piece of data.

handler/request.py

This file manages all the events being called on the server side. So, every time the client calls a get, set, or delete it goes through this requests file and its aptly named request class. This class centralises the security of outgoing data and handles the data in an easy way so that any method calling the request class and its “emit” method doesn’t have to see the complexities of a callback function.

Code

I break down the code per file. Ill start from the root file of both the server and the client. The titles of each section will depict the path from these root files.

Server

main.py

```
import socketio
import eventlet

# SESSION
class server_session():
    def __init__(self):
        self.clients = []
        self.logged_in = []
        self.accepting_clients = True

        self.mode = "normal"
        self.flags = []

        self.encrypt_on_shutdown = True
        self.db_encrypted = True
        self.password = None

# this is a class object shared accross the server
# it allows access to some basic infomation about the server's current status
session = server_session()
# SESSION

# STARTUP
from modules.track.logging import log
from modules.start.start import main as server_startup
server_startup(session)
# STARTUP

# MODULES
from modules.auth import auth
from modules.track import *
send_status = logging.status.send_status

from modules.user import info as user_info
from modules.handler.handler import *
from modules.handler.tasks import user_service, server_service
from modules.algorithms.univ import dict_key_verify
from modules.data.datetime import timestamp
from modules.data.config import read as config_read
```

```
# MODULES

sio = socketio.Server()
app = socketio.WSGIApp(sio)


# CONNECT/DISCONNECT EVENTS
@sio.event
def connect(sid, environ, auth):
    if session.accepting_clients:
        sio.save_session(sid, {'id': None, 'level': None})
        log("INFO", f"client {sid} connected")
        session.clients.append(sid)
    else:
        # return status here, create interface etc
        sio.disconnect(sid)

@sio.event
def disconnect(sid):
    log("INFO", f"client {sid} disconnected")
    session.clients.remove(sid)
    if sid in session.logged_in:
        session.logged_in.remove(sid)
# CONNECT/DISCONNECT EVENTS


# AUTH EVENTS
@sio.event
def login(sid, data):
    info = {'logged_in': False}
    status, user_id, level = auth.login(sio, sid, data)

    with sio.session(sid) as client_session:
        # saves some infomation to the sid of a connected client
        # this sid can be passed to other functions to identify the client even if
they havent provided specfic info to that event
        # as long as they have logged in
        client_session['id'] = user_id
        client_session['level'] = level

    if auth.authorised(sio, sid, "member"):
        info['logged_in'] = True
        if sid not in session.logged_in:
            session.logged_in.append(sid)
        sio.start_background_task(user_service, sio, sid)

    send_status(sio, sid, status)
```

```
    return True, info

# any event that returns infomation will return True as its first parameter
# this is to let the client side function know that the infomation being returned
is a "callback" from the server
# without this the client side function would have no way of knowing if the func-
tion has been called by the server or the client itself

@sio.event
def register(sid, data):
    status = auth.register(data)
    send_status(sio, sid, status)
    if status['level'] == "INFO":
        return True, {'is_registered': True}
    return True, {'is_registered': False}

@sio.event
def admin_register(sid, data):
    status = auth.admin_register(data)
    send_status(sio, sid, status)
    if status['level'] == "INFO":
        return True, {'is_registered': True}
    return True, {'is_registered': False}

@sio.event
def auth_get(sid, data=None):
    info, status = auth_handler(sio, sid, session, min_level='member',
event_name='auth_get').get(data)
    return True, info

@sio.event
def auth_set(sid, data=None):
    info, status = auth_handler(sio, sid, session, min_level='member',
event_name='auth_set').set(data)
# AUTH EVENTS


# PROFILE EVENTS
@sio.event
def profile_get(sid, data=None):
    info, status = profile_handler(sio, sid, session, min_level='member',
event_name='profile_get').get(data)
    return True, info

@sio.event
def profile_get_permissions(sid, data=None):
```

```
    info, status = profile_handler(sio, sid, session, min_level='member',
event_name='profile_get_permissions').get_permissions(data)
    return True, info

@sio.event
def profile_set(sid, data=None):
    info, status = profile_handler(sio, sid, session, min_level='member',
event_name='profile_set').set(data)

@sio.event
def profile_delete(sid, data=None):
    info, status = profile_handler(sio, sid, session, min_level='member',
event_name='profile_delete').delete(data)
# PROFILE EVENTS END


# FRIEND EVENTS START
@sio.event
def friend_get(sid, data=None):
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='friend_get').get(data)
    return True, info

@sio.event
def friend_get_requests(sid, data=None):
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='friend_get_requests').get_requests(data)
    return True, info

@sio.event
def friend_get_recomendations(sid, data=None):
    friend_get_recomendations
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='friend_get_recomendations').get_recomendations(data)
    return True, info

@sio.event
def friend_add_request(sid, data=None):
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='friend_add_request').add_request(data)

@sio.event
def friend_approve_request(sid, data=None):
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='friend_approve_request').approve_request(data)

@sio.event
```

```
def friend_remove_request(sid, data=None):
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='friend_remove_request').remove_request(data)

@sio.event
def friend_reject_request(sid, data=None):
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='friend_reject_request').reject_request(data)

@sio.event
def friend_remove(sid, data=None):
    info, status = friend_handler(sio, sid, session, min_level='member',
event_name='remove').remove(data)
# FRIEND EVENTS END


# OCCUAPTION EVENTS
@sio.event
def occupation_get(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='member',
event_name='occupation_get').get(data)
    return True, info

@sio.event
def occupation_get_all(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='member',
event_name='occupation_get_all').get_all(data)
    return True, info

@sio.event
def occupation_set(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='member',
event_name='occupation_set').set(data)

@sio.event
def occupation_set_request(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='member',
event_name='occupation_set_request').set_request(data)

@sio.event
def occupation_get_request(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='member',
event_name='occupation_get_request').get_request(data)
    return True, info

@sio.event
def occupation_get_all_requests(sid, data=None):
```

```
    info, status = occupation_handler(sio, sid, session, min_level='member',
event_name='occupation_get_all_request').get_all_request(data)
    return True, info

@sio.event
def occupation_delete_request(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='member',
event_name='occupation_delete_request').delete_request(data)

@sio.event
def occupation_approve_request(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='management',
event_name='occupation_approve_request').approve_request(data)

@sio.event
def occupation_reject_request(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='management',
event_name='occupation_reject_request').reject_request(data)

@sio.event
def occupation_create(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='management',
event_name='occupation_create').create(data)

@sio.event
def occupation_edit(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='management',
event_name='occupation_edit').edit(data)

@sio.event
def occupation_delete_occupation(sid, data=None):
    info, status = occupation_handler(sio, sid, session, min_level='management',
event_name='occupation_delete_occupation').delete_occupation(data)
# OCCUAPTION EVENTS


# TEAM EVENTS
@sio.event
def team_get(sid, data=None):
    info, status = team_handler(sio, sid, session, min_level='member',
event_name='team_get').get(data)
    return True, info

@sio.event
def team_get_all(sid, data=None):
    info, status = team_handler(sio, sid, session, min_level='member',
event_name='team_get_all').get_all(data)
```

```
    return True, info

@sio.event
def team_get_leaders(sid, data=None):
    info, status = team_handler(sio, sid, session, min_level='member',
event_name='team_get_leaders').get_leaders(data)
    return True, info

@sio.event
def team_get_members(sid, data=None):
    info, status = team_handler(sio, sid, session, min_level='member',
event_name='team_get_members').get_members(data)
    return True, info

@sio.event
def team_set(sid, data=None):
    info, status = team_handler(sio, sid, session, min_level='member',
event_name='team_set').set(data)

@sio.event
def team_delete_leaders(sid, data=None):
    info, status = team_handler(sio, sid, session, min_level='member',
event_name='team_delete_leaders').delete_leaders(data)
# TEAM EVENTS


# POST EVENTS
@sio.event
def post_get_feed(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_get_feed').get_feed(data)
    return True, info

@sio.event
def post_get(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_get').get(data)
    return True, info

@sio.event
def post_get_memories(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_get').get_memories(data)
    return True, info

@sio.event
def post_get_user(sid, data=None):
```

```
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_get_user').get_user(data)
    return True, info

@sio.event
def post_get_friends(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_get_friends').get_friends(data)
    return True, info

@sio.event
def post_get_team(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_get_team').get_team(data)
    return True, info

@sio.event
def post_get_permissions(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_get_permissions').get_permissions(data)
    return True, info

@sio.event
def post_set(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_set').set(data)

@sio.event
def post_delete(sid, data=None):
    info, status = post_handler(sio, sid, session, min_level='member',
event_name='post_delete').delete(data)
# POST EVENTS


# COMMENT EVENTS
@sio.event
def comment_get(sid, data=None):
    info, status = comment_handler(sio, sid, session, min_level='member',
event_name='comment_get').get(data)
    return True, info

@sio.event
def comment_get_post(sid, data=None):
    info, status = comment_handler(sio, sid, session, min_level='member',
event_name='comment_get_post').get_post(data)
    return True, info
```

```
@sio.event
def comment_get_permissions(sid, data=None):
    info, status = comment_handler(sio, sid, session, min_level='member',
event_name='comment_get_permissions').get_permissions(data)
    return True, info

@sio.event
def comment_set(sid, data=None):
    info, status = comment_handler(sio, sid, session, min_level='member',
event_name='comment_set').set(data)
    return True, info

@sio.event
def comment_delete(sid, data=None):
    info, status = comment_handler(sio, sid, session, min_level='member',
event_name='comment_delete').delete(data)
    return True, info
# COMMENT EVENTS


# IMPRESSION EVENTS
@sio.event
def post_impression_get(sid, data=None):
    info, status = post_impression_handler(sio, sid, session, min_level='member',
event_name='post_impression_get').get(data)
    return True, info

@sio.event
def post_impression_get_post(sid, data=None):
    info, status = post_impression_handler(sio, sid, session, min_level='member',
event_name='post_impression_get_post').get_post(data)
    return True, info

@sio.event
def post_impression_count(sid, data=None):
    info, status = post_impression_handler(sio, sid, session, min_level='member',
event_name='post_impression_count').count(data)
    return True, info

@sio.event
def post_impression_set(sid, data=None):
    info, status = post_impression_handler(sio, sid, session, min_level='member',
event_name='post_impression_set').set(data)

@sio.event
def post_impression_delete(sid, data=None):
```

```
    info, status = post_impression_handler(sio, sid, session, min_level='member',
event_name='post_impression_delete').delete(data)

@sio.event
def comment_impression_get(sid, data=None):
    info, status = comment_impression_handler(sio, sid, session, min_level='mem-
ber', event_name='comment_impression_get').get(data)
    return True, info

@sio.event
def comment_impression_get_comment(sid, data=None):
    info, status = comment_impression_handler(sio, sid, session, min_level='mem-
ber', event_name='comment_impression_get_comment').get_comment(data)
    return True, info

@sio.event
def comment_impression_count(sid, data=None):
    info, status = comment_impression_handler(sio, sid, session, min_level='mem-
ber', event_name='comment_impression_count').count(data)
    return True, info

@sio.event
def comment_impression_set(sid, data=None):
    info, status = comment_impression_handler(sio, sid, session, min_level='mem-
ber', event_name='comment_impression_set').set(data)

@sio.event
def comment_impression_delete(sid, data=None):
    info, status = comment_impression_handler(sio, sid, session, min_level='mem-
ber', event_name='comment_impression_delete').delete(data)
# IMPRESSION EVENTS END


# NOTIFICATION EVENTS START
@sio.event
def notification_get(sid, data=None):
    info, status = notification_handler(sio, sid, session, min_level='member',
event_name='notification_get').get(data)
    return True, info

@sio.event
def notification_create(sid, data=None):
    status = notification_handler(sio, sid, session, min_level='member',
event_name='notification_create').create(data)

@sio.event
def notification_delete(sid, data=None):
```

```
    status = notification_handler(sio, sid, session, min_level='member',
event_name='notification_delete').delete(data)

@sio.event
def notification_remove(sid, data=None):
    status = notification_handler(sio, sid, session, min_level='member',
event_name='notification_remove').remove(data)
# NOTIFICATION EVENTS END


# OTHER EVENTS
@sio.event
def get_ntfy_topic(sid, data=None):
    info = {'topic': None}
    if sio.get_session(sid)['level']:
        user_id = sio.get_session(sid)['id']
        username = user_info.auth(user_id=user_id).get()['username']

        nfty_topic = f"{username}-{user_id[:8]}"
        info['topic'] = nfty_topic

    return True, info

@sio.event
def server_code_get(sid, data=None):
    code = config_read('miscellaneous', 'servercode')
    info = {'server_code': code}
    return True, info

@sio.event
def is_post_slot(sid, data=None):
    info = None
    if timestamp().is_valid_time():
        info = {'is_post_slot': True}
    else:
        info = {'is_post_slot': False}
    return True, info

@sio.event
def get_date(sid, data=None):
    info = {'date':timestamp().date}
    return True, info

@sio.event
def post_slot_get(sid, data=None):
    info, status = post_slot_handler(sio, sid, session, min_level='member',
event_name='post_slot_get').get(data)
```

```
    return True, info

@sio.event
def shutdown(sid, data=None):
    info, status = server(sio, sid, session, min_level='admin', event_name='shut-
down').shutdown(data)
# OTHER EVENTS

# ENCRYPTION EVENTS START
@sio.event
def decrypt(sid, data=None):
    success = encryption_handler(session).decrypt(data)
    return True, {'success': success}

@sio.event
def get_mode(sid, data=None):
    sss_enabled = config_read('database', 'ShamirSecretSharing')
    min_shares = config_read('database', 'MinimumShares')
    info = {'mode': session.mode, 'password': True, 'sss': sss_enabled,
'min_shares': min_shares}
    return True, info
# ENCRYPTION EVENTS END

def test():
    pass

def main():
    # add mode check + while loop to background tasks
    sio.start_background_task(server_service, session)
    open_port = int(config_read('networking', 'Port'))
    eventlet.wsgi.server(eventlet.listen(('', open_port)), app)
    server(sio, None, session, min_level='admin', event_name='shutdown').inter-
nal_shutdown({'time': 0.1})

if __name__ == "__main__":
    main()
```

modules/algorithms/hash.cpp

```
# include<string.h>
# include<string>
# include<cmath>
typedef long long int Lint;

extern "C" Lint hash(char* str) {
  Lint m = std::pow(10,7) + 7;
  int p = 97;
```

```
  Lint total = 0;

  for (int i=0; i<strlen(str); i++) {
    total += (int(str[i]) - 32) * pow(p,i);
  }

  Lint result = total % m;
  return result;
}

extern "C" Lint printc(char* str) {
  int num = strlen(str);
  return num;
}
```

modules/algorithms/univ.py

```
# checks a string for illegal characters
# string = string to be checked
# allow_chars = allowed characters should be passed as a string
def char_check(string, allow_chars):

    # default allow_chars value
    if allow_chars == None:
        allow_chars = ascii_letters + digits

    #allowed_char = ascii_letters + digits + "_" + "-"
    if set(string).difference(allow_chars):
        return True
    else:
        return False

def dict_key_verify(dictionary, keys, mode="and", *args, **kwargs):
# checks if the dictionary exists, if the key exists as a field and if that fields
value is not none
# can be used to check if multiple keys exist
    if mode != "and" and mode != "or":
        mode = "and"
    if type(keys) != list:
        keys = [keys]

    verified = []
    if type(keys) != list:
        keys = [keys]

    for key in keys:
```

```
        if type(dictionary) != dict or key not in dictionary or not diction-
ary[key]:
            verified.append(False)
        else:
            verified.append(True)

    if mode == "and":
        if all(verified) == True:
            return True
    if mode == "or":
        if True in verified:
            return True
    return False


if __name__ == "__main__":
    data = {'name': "joe", 'job': "cuck", 'age': "69"}
    answer = dict_key_verify(data, ['job', 'names'], "and")
    print(answer)
```

modules/algorithms/uuid.py

```
import random
import ctypes
import pathlib
import hashlib

# RBP
import time
# RBP

def bin_to_hex(byte):
    byte_hex = ""
    total = 0
    for i, bit in enumerate(byte):
        total += int(bit) * 2 ** i
    first_place = total // 16
    second_place = total - first_place * 16

    places = [first_place, second_place]
    for i, place in enumerate(places):
        if place < 10:
            byte_hex += str(place)
        else:
            byte_hex += chr(65 + place - 10)

    return byte_hex.lower()
```

```
def den_to_bin(number):
    byte_string = ""
    result = 2
    power = 0

    # finds the greatest power of 2 that can fit in the number
    # this defines the length of the binary number
    while result > 0:
        result = number // 2**power
        if result == 0:
            break
        power += 1

    for i in range(power-1, -1, -1):
        bit = number // 2**i
        number -= bit * 2**i
        byte_string += str(bit)

    return byte_string

def set_bits(binary, num_bits):
    for i in range(num_bits - len(binary)):
        binary += "0"
    return binary

#uuid START
def generate():
    byte_list = []

    # generates 16 8 bit numbers as strings
    for i in range(16):
        number = random.randint(0, 255)
        bits = den_to_bin(number)
        byte = set_bits(bits , 8)
        byte_list.append(byte)

    # setting certain places as pre-defined, as stated by the UUID4 spec (see apen-
dix)
    byte_list[6] = byte_list[6][:4] + "0010"
    byte_list[8] = byte_list[8][:6] + "01"

    # UUIDs are always shown in terms of hex
    hex_string = ""
    for byte_index, byte in enumerate(byte_list):
        byte_hex = bin_to_hex(byte)
        # adds the dashes in the indexes as required by the UUID4 spec
```

```
        if byte_index in [4, 6, 8, 10]:
            hex_string += "-"
        hex_string += byte_hex

    return hex_string
#uuid END

#string hash START
def hash_string(string):
    string = string.replace("-", "0")
    string = string.replace("_", "0")
    libname = pathlib.Path().absolute() / "modules/algorithms/libcpphash.so"
    c_lib = ctypes.CDLL(libname)

    charptr = ctypes.POINTER(ctypes.c_char)
    c_lib.printc.argtypes = [charptr]
    c_lib.printc.restypes = int

    result = c_lib.hash(ctypes.c_char_p(string.encode('utf-8')))
    return result

def long_hash(string):
    result = hashlib.sha256(string.encode('utf-8'))
    result = result.hexdigest()
    return result

# string hash END

if __name__ == "__main__":
    result = hash_string("hello")
    print(result)
```

modules/algorithms/recommend.py

```
from modules.user import info as user_info
from modules.algorithms.univ import dict_key_verify
from modules.algorithms.uuid import hash_string

class User():
    def __init__(self, username, origin=False):
        self.username = username
        self.friends = user_info.friend(username=username)
        self.origin = origin

        self.exclude = []
        self.count = 1
        self.depth = 0
```

```
        self.score = 0

        self.friend_list = []

    def find_friends(self, exclude=[], **kwargs):
        self.exclude += exclude
        friends = self.friends.get()
        if dict_key_verify(friends, "friends"):
            self.__organise_friends(friends['friends'])

        self.__find_excluded()

    def __organise_friends(self, friends, **kwargs):
        # used to create the user objects of friends
        for friend in friends:
            if friend['username'] not in self.exclude:
                self.friend_list.append(User(friend['username']))

    def __find_excluded(self):
        # gathers the users to be excluded from the next nodes neigbours and sets
this list = to self.exclude
        # this exclude list includes the previously passed exclude list
        if self.username not in self.exclude:
            self.exclude.append(self.username)
        if self.origin:
            self.exclude = self.exclude + [friend.username for friend in
self.friend_list]
            requests = self.friends.get_requests()
            if dict_key_verify(requests, "requests"):
                self.exclude = self.exclude + [request for request in requests["re-
quests"]]

    def __hash__(self):
        obj_hash = hash_string(self.username)
        return obj_hash

class Graph():
    def __init__(self, username):
        self.origin_user = User(username, True)
        self.graph = [[]] * (10**7+7)

        self.friend_directory = [None] * (10**7+7)
        self.friend_directory[hash(self.origin_user)] = self.origin_user
        self.exclude = []

    def generate(self, depth=1):
        self.origin_user.depth = depth-1
```

```
        self.__add_user_friends(self.origin_user, self.origin_user, depth)

    def __add_user_friends(self, origin, source, depth):
        origin.find_friends(self.exclude + [source.username])
        if hash(self.origin_user) == hash(origin):
            self.exclude += origin.exclude

        for friend in origin.friend_list:
            friend_hash = hash(friend)
            self.__add_edge(hash(origin), friend_hash)

            # if this user already exists in the graph add to their count in the
user's object
            # this count keeps track of how many other users friend lists a certain
user is
            if self.friend_directory[friend_hash]:
                self.friend_directory[friend_hash].count += 1
            else:
                self.friend_directory[friend_hash] = friend

            if depth-1 > 0:
                # recursively calls the function until the depth is 0.
                self.__add_user_friends(friend, origin, depth-1)

    def __add_edge(self, node, edge):
        # using the + operator on the lists since .append() has some undefined be-
haviour on large arrays.
        self.graph[node] = self.graph[node] + [edge]

    def bft(self):
        self.visted = []
        # adds the hash of the selected orgin user to the edge queue
        self.edge_queue = [hash(self.origin_user)]

        self.__visit(self.edge_queue[0])

    def __visit(self, origin):
        # the origin is a number and so can be used as an index for the graph array
        start_pos = self.graph[origin]
        self.__on_visit(origin)

        # adds the current node to the vistsed lists and removes it from the queue
        self.edge_queue.pop(len(self.edge_queue)-1)
        self.visted.append(origin)

        for neigbour in start_pos:
            neigbour_obj = self.friend_directory[neigbour]
```

```
            origin_obj = self.friend_directory[origin]

            # checks if the node has been visted yet, if not adds it to the edge
queue and assigns it a depth from the origin
            if neigbour not in self.visted and neigbour not in self.edge_queue:
                neigbour_obj.depth = origin_obj.depth - 1
                self.edge_queue = [neigbour] + self.edge_queue

        if len(self.edge_queue) > 0:
            # recursively calls this method until the edge_queue is empty
            self.__visit(self.edge_queue[len(self.edge_queue)-1])

    def __on_visit(self, origin):
        origin_obj = self.friend_directory[origin]
        # each node is only visited once in the graph so the count is calculated
when constructing the graph
        origin_obj.score = origin_obj.depth * origin_obj.count

    def recomend_friends(self):
        self.recomendations = []

        # removing the user requesting the recomendations and their friends from
the visited list
        # this is done so that the user or people who are already friends of the
user dont get recomended
        possible = []
        for user in self.visted:
            user_obj = self.friend_directory[user]
            if user_obj.username not in self.exclude:
                possible = possible + [user]

        while len(self.recomendations) != len(possible):
            largest = User(username="")
            largest.score = -1
            for friend in possible:
                friend_obj = self.friend_directory[friend]
                if friend_obj not in self.recomendations and friend_obj.score >
largest.score:
                    largest = friend_obj

            self.recomendations.append(largest)

def recomend_friend(username, amount=1, depth=1):
    if not (depth >= 1 and depth <= 4):
        depth = 4

    friend_graph = Graph(username)
```

```
    friend_graph.generate(depth)
    friend_graph.bft()
    friend_graph.recomend_friends()

    recomended = [{'username': recomended.username} for recomended in
friend_graph.recomendations[:amount]]
    return recomended

def main():
    result = recomend_friend("Jack", 3, 4)

if __name__ == "__main__":
    main()
```

modules/auth/auth.py

```
# BEFORE PRODUCTION PUSH
### Need to uncomment the try and exept build into fuctions:
#### login, register, admin_register

import sqlite3
import time
from string import ascii_letters, ascii_lowercase, digits

### MODULES
from modules.track import *

from modules.user.generate import main as user_generate
from modules.user import info as user_info

from modules.data.database import connect as db_connect
from modules.data.config import read as config_read
from modules.data.datetime import timestamp

from modules.algorithms.uuid import long_hash as hash_string
from modules.algorithms.uuid import generate as uuid_generate
from modules.algorithms.univ import char_check
### MODULES

# need to change this to path
database_name = config_read("database", "Path")

class reg_cred():
    def __init__(self, cred):
        self.level = config_read("user", "DefaultLevel")
```

```
        self.key = cred['key']
        self.username = cred['username']
        self.password= cred['password']
        self.repassword= cred['repassword']

        self.db = db_connect()
        self.db.create(self)

        logging.status("INFO", "registration initialised").status_update(self)

    def exec(self):
        # CHECKS
        check_processes = [self.username_verify, self.username_bans,
self.username_clash_check, self.password_verify]
        for check in check_processes:
            check()
            if self.status['level'] == "FAIL":
                return
        if not self.key_verify():
            return
        logging.status("INFO", "credential verification successful").status_up-
date(self)
        # CHECKS

        self.id = user_generate(self.username, self.password, self.level)
        #self.db.close()

        logging.status("INFO", "registration successful").status_update(self)

    def username_verify(self):
        # This will be configurable
        min_len = 3
        max_len = 25

        if self.username == None:
            logging.status("FAIL", "username cannot be null").status_update(self)

        elif len(self.username) < min_len or len(self.username) > max_len:
            logging.status("FAIL", f"username cant be shorter than {min_len} char-
acters or longer than {max_len} characters").status_update(self)

        elif char_check(self.username, ascii_letters + digits + "_" + "-") == True:
            logging.status("FAIL", f"username contains invalid characters").sta-
tus_update(self)

    def username_bans(self):
```

```
        servercode = config_read('miscellaneous', 'servercode')
        if servercode in self.username:
            logging.status("FAIL", "usernames contains servercode").status_up-
date(self)

    def username_clash_check(self):
        self.cur.execute("SELECT username FROM auth_credentials WHERE username =
?", (self.username,))

        if self.cur.fetchall():
            logging.status("FAIL", "username is already in use").status_up-
date(self)

    def password_verify(self):
        # This will be configurable
        min_len = 4
        max_len = 100

        if self.password == None:
            logging.status("FAIL", "password cannot be null").status_update(self)

        elif len(self.password) < min_len or len(self.password) > max_len:
            logging.status("FAIL", f"password cant be shorter than {min_len} char-
acters or longer than {max_len}").status_update(self)

        elif self.password != self.repassword:
            logging.status("FAIL", f"passwords do not match").status_update(self)

    def key_verify(self):
        if self.key == config_read('authorisation', 'RegistrationKey'):
            return True
        else:
            return False
            logging.status("FAIL", "registration code is incorrect").status_up-
date(self)

class reg_admin(reg_cred):
    def __init__(self, cred):
        super().__init__(cred)
        self.level = "admin"

        logging.status("INFO", "admin registration initialised").status_up-
date(self)

    def key_verify(self):
        if self.key == config_read('authorisation', 'AdminKey'):
            return True
```

```
        else:
            return False

    def first_time(self):
        self.cur.execute("SELECT user_id FROM auth_credentials WHERE level = ?",
(self.level,))
        value = self.cur.fetchone()

        if value:
            return False
        else:
            return True

class login_cred():
    def __init__(self, sio, sid, cred):
        self.username = cred['username']
        self.password = cred['password']

        self.sio = sio
        self.sid = sid

        self.db = db_connect()
        self.db.create(self)

        logging.status("INFO", "credential login initialised").status_update(self)

    def exec(self):
        self.process_password()

        self.cur.execute("SELECT user_id FROM auth_credentials WHERE username = ?
AND password = ?", (self.username, self.password_hash))
        self.id = self.cur.fetchone()

        if self.id:
            self.id = self.id[0]

            logging.status("INFO", "valid login credentials").status_update(self)
            login_token.create_token(self)
            login_token.send_token(self)
            logging.status("INFO", "login successful").status_update(self)
        else:
            logging.status("FAIL", "invalid login credentials").status_update(self)

        self.db.close()

    def process_password(self):
```

```
        self.cur.execute("SELECT user_id FROM auth_credentials WHERE username = ?",
(self.username,))
        user_id = self.cur.fetchone()
        if user_id:
            self.password_hash = hash_string(self.password + user_id[0])
        else:
            self.password_hash = None

class login_token():
    def __init__(self, cred):
        self.token = cred['token']

        self.db = db_connect()
        self.db.create(self)

        logging.status("INFO", "token login initialised").status_update(self)

    def exec(self):
        self.token_hash = hash_string(self.token)
        self.cur.execute("SELECT user_id, token_expire FROM auth_tokens WHERE token
= ?", (self.token_hash,))
        fetch_data = self.cur.fetchall()
        self.id = None

        if fetch_data:
            self.id, self.token_expire = fetch_data[0][0], fetch_data[0][1]

            if self.token_expire > timestamp().now:
                logging.status("INFO", "valid token").status_update(self)
            else:
                logging.status("FAIL", "invalid token").status_update(self)

        else:
            logging.status("FAIL", "invalid token").status_update(self)

        self.db.close()

    @staticmethod
    def create_token(self):
        expire_time = float(config_read("authorisation", "tokenexpirytime"))

        self.token = uuid_generate()
        self.token_hash = hash_string(self.token)
        self.token_expire = timestamp().now + expire_time

        ### ALL NEEDS CHANGING
```

```
        self.cur.execute("INSERT INTO auth_tokens (user_id, token, token_expire)
VALUES (?, ?, ?)", (self.id, self.token_hash, self.token_expire))
        self.db.commit()

        logging.status("INFO", "authentication token created").status_update(self)

    @staticmethod
    def send_token(self):
        self.sio.emit('recv_token', {'token':self.token, 'expire': self.token_ex-
pire}, room=self.sid)
        logging.status("INFO", "token sent").status_update(self)


class error_process():
    def __init__(self):
        logging.status("WARNING", "something went wrong").status_update(self)
        self.id = None

def login(sio, sid, cred):
    if "token" in cred:

        try:
            client = login_token(cred)
            client.exec()
        except:
            logging.status("FAIL", "token not authorised").status_update(client)


    elif all(param in cred for param in ['username', 'password']):

        try:
            client = login_cred(sio, sid, cred)
            client.exec()
        except:
            logging.status("FAIL", "login failed").status_update(client)

    else:
        client = error_process()
        logging.status("FAIL", "no credentials provided").status_update(client)

    client.level = user_info.level(user_id=client.id).get()
    if client.level:
        client.level = client.level['level']
    return client.status, client.id, client.level

def register(cred):
```

```
    if all(param in cred for param in ['username', 'password', 'repassword',
'key']):

        try:
            client = reg_cred(cred)
            client.exec()
        except:
            logging.status("FAIL", "registration failed").status_update(client)

    else:
        client = generic_process()
        logging.status("FAIL", "no credentials provided").status_update(client)

    return client.status

def admin_register(cred):
    if all(param in cred for param in ['username', 'password', 'repassword',
'key']):

        try:
            client = reg_admin(cred)
            if client.key_verify() == True and client.first_time() == True:
                client.exec()
            else:
                logging.status("FAIL", "admin key does not match/admin already ex-
ists").status_update(client)
        except:
            logging.status("FAIL", "registration failed").status_update(client)

    else:
        client = error_process()
        logging.status("FAIL", "no credentials provided").status_update(client)

    return client.status

def authorised(sio, sid, min_level='admin'):
    level_list = ['member', 'management', 'admin']

    allow_levels = level_list[level_list.index(min_level):]
    level = sio.get_session(sid)['level']

    if level in allow_levels:
        user_authorised = True
    else:
        user_authorised = False

    return user_authorised
```

```
def main():
    error = error_process()

if __name__ == "__main__":
    main()
```

modules/data/config.py

```
import configparser
from modules.track.logging import log

path = "data/config.ini"

def create():
    try:
        file = open(path, 'r')
        log("INFO", "Config already exists")
        return
    except FileNotFoundError as e:
        log("INFO", "Creating config file")
        pass

    config = configparser.ConfigParser()

    config.add_section('authorisation')
    # change this to a randomly generated string
    config.set('authorisation', 'AdminKey', 'secret')
    config.set('authorisation', 'RegistrationKey', 'secret')
    config.set('authorisation', 'UsernameMaxLength', '20')
    config.set('authorisation', 'UsernameMinLength', '5')
    config.set('authorisation', 'PasswordMaxLength', '30')
    config.set('authorisation', 'PasswordMinLength', '5')
    config.set('authorisation', 'TokenExpiryTime', '2592000')

    config.add_section('database')
    config.set('database', 'Path', 'data/database.db')
    config.set('database', 'Encrypt', 'false')
    config.set('database', 'ShamirSecretSharing', 'false')
    config.set('database', 'NumberOfShares', '5')
    config.set('database', 'MinimumShares', '3')
    config.set('database', 'KeyPath', 'data/key.txt')
    config.set('database', 'EncryptedPath', 'data/.cryptdatabase.db')
    config.set('database', 'EncryptionConfigPath', 'data/encryptconfig.txt')
    config.set('database', 'SaltPath', 'data/.salt.txt')
    config.set('database', 'SharesPath', 'data/shares/')
```

```
    config.add_section('user')
    config.set('user', 'DefaultLevel', 'member')
    config.set('user', 'DefaultOccupationID', 'Null')

    config.add_section('posts')
    config.set('posts', 'PostTimeLimit', '5') # miniutes
    config.set('posts', 'DayStart', '9') #24 hour time
    config.set('posts', 'DayEnd', '17') #24 hour time

    config.add_section('notifications')
    config.set('notifications', 'DefaultExpireTime', '604800')
    config.set('notifications', 'ntfyUrl', 'https://ntfy.example.com')

    config.add_section('networking')
    config.set('networking', 'Port', '9999')

    config.add_section('miscellaneous')
    config.set('miscellaneous', 'ServerCode', '12345')

    with open(path, 'w') as configfile:
        config.write(configfile)
    log("INFO", "Created config file")

def read(section, key, *args, **kwargs):
    config = configparser.ConfigParser()
    config.read(path)

    if section not in config:
        return None
    if key not in config[section]:
        return None

    info = config[section][key]
    if info == "false":
        info = False
    if info == "true":
        info = True

    return info

def main():
    create()
    info = read("users", "DefaultOccupation")
    print(info)

if __name__ == "__main__":
```

```
    main()
```

modules/data/database.py

```
import sqlite3
import os
import ctypes
import pathlib

import base64
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

# db encrypt
from cryptography.fernet import Fernet
#from pysqlcipher3 import dbapi2 as sqlite3
# db encrypt

from modules.track.logging import log
from modules.data.config import read as config_read
from modules.algorithms.uuid import generate as uuid_generate

class connect():
    def __init__(self):
        self.path = config_read("database", "Path")

    def create(self, obj):
        self.con = sqlite3.connect(self.path)
        self.cur = self.con.cursor()

        if obj != None:
            obj.con = self.con
            obj.cur = self.cur

    def commit(self):
        self.con.commit()

    def close(self):
        self.con.commit()
        self.con.close()

    def execute(self, command, values=None):
        cur = self.con.cursor()
        cur.execute(command, values)
        self.close()
```

```
# Table creation
class create():
    def __init__(self):
        self.path = config_read("database", "Path")
        self.en_path = config_read("database", "EncryptedPath")

    def tables(self):
        decrypted_database = os.path.exists(self.path)
        encrypted_database = os.path.exists(self.en_path)
        if decrypted_database or encrypted_database:
            return

        con = sqlite3.connect(self.path)
        self.cur = con.cursor()

        tables = [self.auth_credentials, self.auth_tokens, self.profile ,
self.friends, self.occupations, self.occupation_requests, self.teams,
self.team_leaders, self.posts, self.comments, self.post_impressions, self.com-
ment_impressions, self.time_slots, self.notifications, self.notifications_sent]
        for table in tables:
            table()

    def auth_credentials(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS auth_credentials (
                    user_id TEXT NOT NULL PRIMARY KEY,
                    username TEXT NOT NULL,
                    password TEXT NOT NULL,
                    level TEXT NOT NULL,
                    FOREIGN KEY (user_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    )
                    """)

    def auth_tokens(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS auth_tokens(
                    user_id TEXT NOT NULL,
                    token TEXT NOT NULL PRIMARY KEY,
                    token_expire REAL NOT NULL,
                    FOREIGN KEY (user_id)
                        REFERENCES auth_credentials (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    )
```

```
                """)

    def profile(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS profile (
                    user_id TEXT NOT NULL PRIMARY KEY,
                    occupation_id TEXT,
                    name TEXT,
                    picture TEXT,
                    biography TEXT,
                    role TEXT,
                    num_friends INTEGER DEFAULT 0,
                    FOREIGN KEY (occupation_id)
                        REFERENCES occupations (occupation_id)
                            ON UPDATE CASCADE
                            ON DELETE SET NULL
                    )
                    """)

    def friends(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS friends (
                    user_id TEXT NOT NULL,
                    friend_id TEXT NOT NULL,
                    approved BOOLEAN,
                    FOREIGN KEY (user_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    FOREIGN KEY (friend_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    PRIMARY KEY (user_id, friend_id)
                    )
                    """)

    def occupations(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS occupations (
                    occupation_id TEXT NOT NULL PRIMARY KEY,
                    name TEXT NOT NULL,
                    description TEXT
                    )
                    """)

    def occupation_requests(self):
```

```
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS occupation_requests (
                    user_id TEXT NOT NULL PRIMARY KEY,
                    occupation_id TEXT NOT NULL,
                    approved BOOLEAN DEFAULT False NOT NULL,
                    FOREIGN KEY (user_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    FOREIGN KEY (occupation_id)
                        REFERENCES occupations (occupation_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    )
                    """)

    def teams(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS teams (
                    team_id TEXT NOT NULL PRIMARY KEY,
                    name TEXT NOT NULL,
                    occupation_id TEXT,
                    user_id TEXT,
                    FOREIGN KEY (occupation_id)
                        REFERENCES occupations (occupation_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    FOREIGN KEY (user_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    )
                    """)

    def team_leaders(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS team_leaders (
                    user_id TEXT NOT NULL,
                    team_id TEXT NOT NULL,
                    FOREIGN KEY (user_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    FOREIGN KEY (team_id)
                        REFERENCES teams (team_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
```

```
                    PRIMARY KEY (user_id, team_id)
                    )
                    """)

    def posts(self):
        self.cur.execute("""
                    CREATE TABLE IF NOT EXISTS posts (
                    post_id TEXT NOT NULL PRIMARY KEY,
                    user_id TEXT NOT NULL,
                    content TEXT NOT NULL,
                    caption TEXT,
                    date TEXT NOT NULL,
                    FOREIGN KEY (user_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    )
                    """)

    def comments(self):
        self.cur.execute("""
                        CREATE TABLE IF NOT EXISTS comments (
                        comment_id TEXT NOT NULL PRIMARY  KEY,
                        post_id TEXT NOT NULL,
                        user_id TEXT NOT NULL,
                        content TEXT NOT NULL,
                        FOREIGN KEY (post_id)
                            REFERENCES posts (post_id)
                                ON UPDATE CASCADE
                                ON DELETE CASCADE
                        FOREIGN KEY (user_id)
                            REFERENCES profile (user_id)
                                ON UPDATE CASCADE
                                ON DELETE CASCADE
                         )
                         """)

    def post_impressions(self):
        self.cur.execute("""
                        CREATE TABLE IF NOT EXISTS post_impressions (
                        impression_id TEXT NOT NULL PRIMARY KEY,
                        post_id NOT NULL,
                        user_id NOT NULL,
                        type NOT NULL,
                        FOREIGN KEY (post_id)
                            REFERENCES posts (post_id)
                                ON UPDATE CASCADE
```

```
                        ON DELETE CASCADE
                    FOREIGN KEY (user_id)
                        REFERENCES profile (user_id)
                            ON UPDATE CASCADE
                            ON DELETE CASCADE
                    )
                     """)

    def comment_impressions(self):
        self.cur.execute("""
                        CREATE TABLE IF NOT EXISTS comment_impressions (
                        impression_id TEXT NOT NULL PRIMARY KEY,
                        comment_id NOT NULL,
                        user_id NOT NULL,
                        type NOT NULL,
                        FOREIGN KEY (comment_id)
                            REFERENCES comments (comment_id)
                                ON UPDATE CASCADE
                                ON DELETE CASCADE
                        FOREIGN KEY (user_id)
                            REFERENCES profile (user_id)
                                ON UPDATE CASCADE
                                ON DELETE CASCADE
                        )
                         """)

    def time_slots(self):
        self.cur.execute("""
                         CREATE TABLE IF NOT EXISTS time_slots (
                         date TEXT NOT NULL PRIMARY KEY,
                         start FLOAT NOT NULL,
                         end FLOAT NOT NULL
                         )
                         """)

    def notifications(self):
        self.cur.execute("""
                        CREATE TABLE IF NOT EXISTS notifications (
                        notification_id TEXT NOT NULL PRIMARY KEY,
                        target_id TEXT NOT NULL,
                        title TEXT NOT NULL,
                        content TEXT,
                        time_created FLOAT NOT NULL,
                        expire_after FLOAT NOT NULL
                        )
                          """)
```

```
    def notifications_sent(self):
        self.cur.execute("""
                        CREATE TABLE IF NOT EXISTS notifications_sent (
                        notification_id TEXT NOT NULL,
                        user_id TEXT NOT NULL,
                        time_sent FLOAT,
                        sent BOOLEAN DEFAULT False NOT NULL,
                        PRIMARY KEY (notification_id, user_id)
                        FOREIGN KEY (notification_id)
                            REFERENCES notifications (notification_id)
                                ON UPDATE CASCADE
                                ON DELETE CASCADE
                        FOREIGN KEY (user_id)
                            REFERENCES profile (user_id)
                                ON UPDATE CASCADE
                                ON DELETE CASCADE
                        )
                         """)

class encryption():
    def __init__(self, session):
        self.key = key()
        # needs to pass num_shares and min_shares
        self.session = session

        self.sss_enabled = config_read("database", "ShamirSecretSharing")
        self.en_config_path = config_read("database", "EncryptionConfigPath")
        self.db_path = config_read("database", "Path")
        self.en_db_path = config_read("database", "EncryptedPath")

    def mode(self):
        # uses a large amount of logic statements to figure out what mode the
server should enter on launch
        # additionally what flags it should launch with
        encryption_enabled = config_read("database", "Encrypt")
        db_encrypted = self.key.is_db_encrypted()

        mode = None
        flags = []
        if encryption_enabled:
            if db_encrypted:
                mode = "decrypt"
            else:
                success = self.encrypt()
                if success:
                    mode = "decrypt"
                else:
```

```
                    exit()
        else:
            if db_encrypted:
                mode = "decrypt"
                flags = ["forever"]
            else:
                mode = "normal"
                self.session.db_encrypted = False

        self.session.mode = mode
        self.session.flags = flags

    def encrypt(self, flags=[]):
        if self.session.password:
            password = self.session.password
        else:
            password = self._generate()

        if not password:
            log("FAIL", "Could not encrypt database, something went wrong, see logs
for details")
            return False

        scheme = self.key.read_db_scheme(password)
        with open(self.db_path, "rb") as db:
            db_data = db.read()

        # create new encrypted database
        log("INFO", "Encrypting database")
        en_db_data = scheme.encrypt(db_data)
        with open(self.en_db_path, "wb") as en_db:
            en_db.write(en_db_data)

        # delete unecrypted database
        os.remove("data/database.db")
        log("INFO", "Deleted unencrypted database")
        return True

    def decrypt(self, data, flags=[]):
        min_shares = config_read('database', 'MinimumShares')
        if "sss" in flags:
            password = int(shares(min_shares).get_key(data['shares']))
        else:
            password = int(data['password'])

        scheme = self.key.read_db_scheme(password)
        if not scheme:
```

```
            return False

        # decrypting the databsae raw bytes
        with open(self.en_db_path, "rb") as en_db:
            en_db_data = en_db.read()

        db_data = scheme.decrypt(en_db_data)
        with open(self.db_path, "wb") as db:
            db.write(db_data)

        if not self._database_read():
            log("FAIL", "Decryption of database failed, see logs for details")
            return False
        log("INFO", "Decryption of database successful")

        self.session.password = password
        for flag in flags:
            if flag == "forever":
                log("WARN", "Permanent decryption of the database")
                self.session.encrypt_on_shutdown = False
                self.key.delete()
            elif flag == "sss":
                with open(self.en_config_path, "w") as en_config:
                    en_config.write(str(password))
                log("WARN", f"You decrypted the database using Shamir secret
shares, your master password has been reconstructed and can be found on the server
at the location: {self.en_config_path}. Please remember to delete this file after
reading")

        self.session.db_encrypted = False
        self.session.mode = "normal"
        return True

    def _generate(self):
        options = self._read_config()
        if not self._config_check(options):
            log("FAIL", "Could not generate encryption scheme, something wrong in
config file or with maseter password")
            return None
        else:
            options['password'] = int(options['password'])
            if self.sss_enabled:
                options['num_shares'] = int(options['num_shares'])
                options['min_shares'] = int(options['min_shares'])

        self.key.generate_key_file(options['password'])
```

```
        if self.sss_enabled:
            log("INFO", "Shamir Secret Sharing enabled, generating shares")
            sss = shares(options['min_shares'], options['num_shares'])
            sss_success = sss.generate_shares(options['password'])
            if not sss_success:
                log("FAIL", "Something went wrong generating shamir secret shares,
see log for details")
                return None

        log("INFO", "Deleting encryption configuration file containing master pass-
word")
        os.remove(self.en_config_path)
        return options['password']

    def _read_config(self):
        num_shares = config_read("database", "NumberOfShares")
        min_shares = config_read("database", "MinimumShares")
        options = {}
        try:
            with open(self.en_config_path, "r") as config:
                log("INFO","Reading encryption configuration file")
                options['password'] = config.read()

            if self.sss_enabled:
                options['num_shares'] = num_shares
                options['min_shares'] = min_shares
        except:
            return None

        return options

    def _config_check(self, options):
        # checking if the file exists
        try:
            en_config = open(self.en_config_path, "r")
            en_config.close()
        except:
            log("FAIL", f"Encryption config could not be found at {self.en_con-
fig_path}")
            return False

        # check config contents
        try:
            log("INFO", "Testing master password type (must be int)")
            master_pass = int(options['password'])
            if len(options) == 1:
                return True
```

```
            elif self.sss_enabled and len(options) == 3:
                log("INFO", "Testing number of shares type (must be integer)")
                num_shares = int(options['num_shares'])
                log("INFO", "Testing minimum shares type (must be integer)")
                min_shares = int(options['min_shares'])

                if num_shares < 20 and min_shares < 7:
                    return True
                else:
                    log("WARN", "SSS number of shares is to large or minimum shares
is to large")
                    return False
            else:
                log("WARN", "Something went wrong reading config file, check the
docs for a guide")
                return False
        except:
            log("WARN", "The master password, number of shares and minimum shares
all must be integers")
            return False

    def _database_read(self):
        try:
            db = connect()
            db.create(self)
            db.cur.execute("SELECT * FROM time_slots")
            return True
        except:
            return False

class key():
    def __init__(self):
        self.key_path = config_read("database", "KeyPath")
        self.db_path = config_read("database", "Path")
        self.en_db_path = config_read("database", "EncryptedPath")
        self.salt_path = config_read("database", "SaltPath")

    def _save_salt(self, salt):
        with open(self.salt_path, "wb") as salt_file:
            salt_file.write(salt)

    def _read_salt(self):
        try:
            with open(self.salt_path, "rb") as salt_file:
                salt = salt_file.read()
            return salt
        except:
```

```
            return None

    def _pass_to_scheme(self, password):
        password = str(password).encode()
        salt = self._read_salt()
        if not salt:
            salt = os.urandom(16)
            self._save_salt(salt)

        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,
            salt=salt,
            iterations=480000,
        )
        key = base64.urlsafe_b64encode(kdf.derive(password))
        scheme = Fernet(key)

        return scheme

    def read_db_scheme(self, password):
        file_scheme = self._pass_to_scheme(password)

        with open(self.key_path, "r") as key_file:
            en_password = key_file.read()

        db_scheme = None
        try:
            password = file_scheme.decrypt(en_password)
            db_scheme = self._pass_to_scheme(password)
        except:
            log("WARN", "Provided password is wrong or something is wrong with the
database key")
        return db_scheme

    def generate_key_file(self, password):
        #db_password = bytes(uuid_generate().replace("-", "").encode())
        db_password = uuid_generate().replace("-", "").encode()
        file_scheme = self._pass_to_scheme(password)
        en_db_password = str(file_scheme.encrypt(db_password).decode())
        with open(self.key_path, "w") as key_file:
            key_file.write(en_db_password)

    def delete(self):
        os.remove(self.salt_path)
        os.remove(self.key_path)
        os.remove(self.en_db_path)
```

```
    def is_db_encrypted(self):
        try:
            db = open(self.en_db_path, "rb")
            return True
        except:
            return False

class ShareStruct(ctypes.Structure):
    __fields__ = [("y", ctypes.c_longlong), ("x", ctypes.c_int)]

# this class is mainly geared towards acting as an interface for hte c++ code
class shares():
    def __init__(self, min_shares, num_shares=None):
        if num_shares:
            self.num_shares = int(num_shares)
        self.min_shares = int(min_shares)
        self.shares_path = config_read("database", "SharesPath")

    def _dict_to_c_array(self, share_list):
        c_share_array = ((ctypes.c_longlong*2)*self.min_shares)
        share_array = []

        for i in range(len(share_list)):
            c_share = (ctypes.c_longlong*2)(*[share_list[i]['num'],
share_list[i]['secret']])
            share_array.append(c_share)

        c_share_array = ((ctypes.c_longlong*2)*len(share_list))(*share_array)
        return c_share_array

    def generate_shares(self, password):
        libname = pathlib.Path().absolute() / "modules/data/libcppsss.so"
        c_lib = ctypes.CDLL(libname)

        c_lib.newSecretInternal.argtypes = [ctypes.c_longlong, ctypes.c_int,
ctypes.c_int, ctypes.POINTER(ctypes.c_char)]
        c_lib.newSecretInternal.restypes = None

        path_ptr = ctypes.c_char_p(self.shares_path.encode('utf-8'))
        c_lib.newSecretInternal(password, self.num_shares, self.min_shares,
path_ptr)

        success = self.verify(password)
        return success

    def get_key(self, share_list):
```

```
        libname = pathlib.Path().absolute() / "modules/data/libcppsss.so"
        c_lib = ctypes.CDLL(libname)

        c_share_array = ((ctypes.c_longlong*2)*self.min_shares)
        c_share_array_pointer = ctypes.POINTER(c_share_array)

        c_lib.solveInternal.argtypes = [c_share_array_pointer, ctypes.c_int]
        c_lib.solveInternal.restypes = int

        new_share_array = ctypes.pointer(self._dict_to_c_array(share_list))
        result = c_lib.solveInternal(new_share_array, self.min_shares)
        return result

    def verify(self, password):
        # used to verify that the shamir secret shares generated can be used to re-
construct the original key
        log("INFO", "Verifying share integrity")
        # we essentially take a sample of the shares
        # if all these samples work we assume any combination of said samples will
        # this works well since we test the combination of all hte smallest numbers
and all teh largest
        # the only reason a set of shares wouldnt work is because they have become
to large and c++ starts to lose accuracy
        # if this doesnt happen then its safe to assume all shares work
        shifts = self.num_shares - self.min_shares

        for i in range(shifts):
            top = i + self.min_shares

            shares_used = ""
            for num_share in range(i, top):
                shares_used += str(num_share) + ", "
            shares_used = shares_used[:-2]

            log("INFO", f"Attempting to generate original password with shares:
{shares_used}")
            share_list = []

            for j in range(i, top):
                # reads the shares from their files
                path = self.shares_path + f"share-{j+1}.txt"
                with open(path, "r") as share:

                    try:
                        x = int((share.readline().split(": "))[1])
                        y = int((share.readline().split(": "))[1])
                        share_list.append({'num': x, 'secret': y})
```

```
                    except:
                        log("WARN", "Something went wrong reading one of the
shares, have they been altered?")
                        break

            result = self.get_key(share_list)

            if result != password:
                log("WARN", "A set of shares could not be used to generate the
original password, try again or use a diffrent password")
                return False
            else:
                log("INFO", f"{i+1}/{shifts} sets of shares successfully used to
generate the original password")

        return True


def main():
    db = create()
    db.path = "database.db"
    db.tables()

if __name__ == "__main__":
    main()

class retrieve():
    def __init__(self):
        self.db = db_connect()
        self.db.create(self)

    def level(self, identifier):

        self.cur.execute("SELECT level FROM auth_credentials WHERE username = ? OR
user_id = ?", (identifier, identifier))
        rez = self.cur.fetchone()
        if rez:
            return rez[0]

    def user_id(self, username):

        self.cur.execute("SELECT user_id FROM auth_credentials WHERE username = ?",
(username,))
        rez = self.cur.fetchone()
        if rez:
            rez = rez[0]
```

```
        return rez


    def username(self, user_id):

        self.cur.execute("SELECT username FROM auth_credentials WHERE user_id = ?",
(user_id,))
        rez = self.cur.fetchone()
        if rez:
            rez = rez[0]

        return rez

    def occupation_id(self, user_id):

        self.cur.execute("SELECT occupation_id FROM profile WHERE user_id = ?",
(user_id,))
        rez = self.cur.fetchone()
        if rez:
            rez = rez[0]

        return rez
```

modules/data/datetime.py

```
from datetime import date, timedelta, datetime
import random
import eventlet
# MODULES
from modules.data.config import read as config_read
from modules.data.database import connect as db_connect
from modules.handler import outgoing
from modules.track.logging import log

# MODULES

class timestamp():
    def __init__(self):
        self.time_limit = float(config_read("posts", "PostTimeLimit")) * 60
        self.db = db_connect()
        self.db.create(self)

    @property
    def start(self):
        value = self.get_date_timestamp()
        self._start = value
```

```
        return self._start
    @start.setter
    def start(self, value):
        value = self.get_date_timestamp()
        self._start = value

    @property
    def end(self):
        value = self.get_date_timestamp(day_mod=1) - 1
        self._end = value
        return self._end
    @end.setter
    def end(self, value):
        value = self.get_date_timestamp(day_mod=1) - 1
        self._end = value

    @property
    def now(self):
        value = self.get_timestamp()
        self._now = value
        return value
    @now.setter
    def now(self, value):
        value = self.get_timestamp()
        self._now = value

    @property
    def post_slot_start(self):
        value = self.get_slot()['start']
        self._post_slot_start = value
        return self._post_slot_start
    @post_slot_start.setter
    def post_slot_start(self, value):
        self._post_slot_start = self._post_slot_start

    @property
    def post_slot_end(self):
        value = self.get_slot()['end']
        self._post_slot_end = value
        return self._post_slot_end
    @post_slot_end.setter
    def post_slot_end(self, value):
        self._post_slot_end = self._post_slot_end

    @property
    def date(self):
        date = str(datetime.now().date())
```

```
        self._date = date
        return self._date
    @date.setter
    def date(self, value):
        self._date = value

    def get_date_timestamp(self, year_mod=0, month_mod=0, day_mod=0, *args,
**kwargs):
        modifier = [year_mod, month_mod, day_mod]

        now_mod = (datetime.now()+timedelta(days=day_mod))
        date = (str(now_mod.date()).replace("-0", "-")).split("-")
        date = [int(string) for string in date]

        timestamp = datetime(date[0], date[1], date[2]).timestamp()

        return timestamp

    def get_timestamp(self):
        now = (float(datetime.now().timestamp()))
        return now

    def generate_slot(self, data=None):
        for i in range(2):
            if i == 0:
                date = str(datetime.now().date())
                start = self.get_date_timestamp()
            else:
                now_mod = (datetime.now()+timedelta(days=1))
                date = (str(now_mod.date()))
                start = self.get_date_timestamp(0, 0, 1)

            self.cur.execute("SELECT date FROM time_slots WHERE date=?", (date,))
            if not self.cur.fetchone():
                log("INFO", f"Generating time slot for {date}")

                day_start = start + int(config_read("posts", "DayStart")) * 60 * 60
                day_end = start + int(config_read("posts", "DayStart")) * 60 * 60
                slot_start = random.randint(day_start, day_end)
                slot_end = slot_start + self.time_limit
                self.cur.execute("INSERT INTO time_slots (date, start, end) VALUES
(?, ?, ?)", (date, slot_start, slot_end))
                self.db.commit()

    def get_slot(self):
        info = None
```

```
        date = str(datetime.now().date())
        self.cur.execute("SELECT start, end FROM time_slots WHERE date=?", (date,))
        rez = self.cur.fetchone()
        if rez:
            info = {'start':rez[0], 'end':rez[1]}
        return info

    def is_valid_time(self):
        if (self.now < self.post_slot_end) and (self.now >= self.post_slot_start):
            return True
        return False
```

modules/data/sss.cpp

```
#include <cstdlib>
# include<iostream>
# include<string>
# include<random>
# include<cmath>
# include<array>
#include <fstream>
using namespace std;

typedef long long int Lint; // 64 bits
typedef double Ldouble;
struct security {
  int num_shares;
  int num_required;
};

struct shareStruct {
  int x;
  Lint y;
};

bool isPrime(Lint n) {
  int flag = 0;
  for (int i = 2; i <= n / i; ++i) {
    if (n % i == 0) {
      flag = 1;
      break;
    }
  }
  if (flag == 0) return true;
  else return false;
}
```

```
Lint genRandInt(int n) {
    // Returns a random number
    // between 2**(n-1)+1 and 2**n-1
    //long max = (long)powl(2, n) - 1;
    //long min = (long)powl(2, n - 1) + 1;
    long max = (long)pow(2, n) - 1;
    long min = (long)pow(2, n - 1) + 1;
    Lint result = min + (rand() % ( max - min + 1 ) );
    return result;
}

Lint genPrime() {
  Lint prime = 10;

  while (isPrime(prime) == false) {
    int complexity = 50;
    prime = genRandInt(complexity);
  }
  return prime;
}

int* encodeSecret(int* poly, const int secret, const int num_required) {
  poly[num_required-1] = secret;
  return poly;
}

Lint getPolyY(const int* poly, int poly_len, int poly_x, const Lint prime) {
  Lint total = 0;
  Lint poly_y = 0;

  for (int i=0; i<poly_len+1; i++) {
    int power = poly_len - i;
    int coefficient = poly[i];
    poly_y = coefficient * pow(poly_x, power);
    total = total + poly_y;
  }

  return total;
}

shareStruct* genShares(int num_shares, int num_required, const int* poly, const
Lint prime){
  shareStruct* shares = new shareStruct[num_shares];
  for (int i=1; i<=num_shares; i++) {
    shareStruct share;
    share.x = i;
```

```
    share.y = getPolyY(poly, num_required-1, share.x, prime);
    shares[i-1] = share;
  }
  return shares;
}

int* genPoly(int degree, const Lint prime, const Lint secret) {
  int* poly = new int[degree];

  for (int i = 0; i < degree; i++) {
    int random_num = genRandInt(10);
    poly[i] = prime % random_num;
  }
  return poly;
}

// solving polynomials
struct inputStruct {
  int required;
  shareStruct* shares;
};

struct polyTerm {
  Lint coefficient;
  int power;
};

struct linearEquation {
  shareStruct point;
  polyTerm* terms;
};

linearEquation* constructEquations(const int required, shareStruct shares[]) {
  linearEquation* equations = new linearEquation[required];
  shareStruct share;
  polyTerm term;

  for (int i = 0; i < required; i++) {
    share = shares[i];
    linearEquation equation;
    polyTerm* terms = new polyTerm[required];

    for (int j = 0; j < required; j++) {
      term.power = required - 1 - j;
      terms[j] = term;
    }
```

```
    equation.terms = terms;
    equation.point.x = share.x;
    equation.point.y = share.y;

    equations[i] = equation;
    // dont delete terms from memory as its referanced in equations
  }
  return equations;
}

struct matrix{
  Lint** matrix;
  int dimension_x;
  int dimension_y;
};
struct matrix_system {
  matrix A;
  matrix B;
  matrix X;
};

matrix_system formMatrix(const linearEquation* equations, int required) {
  Lint** matrixA = new Lint*[required];
  Lint** matrixB = new Lint*[required];

  for (int i=0; i < required; i++) {
    linearEquation equation = equations[i];
    Lint* lineA = new Lint[required];
    for (int j=0; j < required; j++) {
      lineA[j] = pow(equation.point.x, equation.terms[j].power);
    }
    matrixA[i] = lineA;

    Lint* lineB = new Lint[1];
    lineB[0] = equation.point.y;
    matrixB[i] = lineB;
  }

  matrix matrixA_data; matrix matrixB_data;
  matrixA_data.matrix = matrixA; matrixB_data.matrix = matrixB;

  matrixA_data.dimension_x = required; matrixB_data.dimension_x = 1;
  matrixA_data.dimension_y = required; matrixB_data.dimension_y = required;

  matrix_system matricies;
  matricies.A = matrixA_data; matricies.B = matrixB_data;
```

```
  return matricies;
}

Lint** findMinor(Lint** matrixA, const int dimension, const int pos_x, const int
pos_y) {
  Lint** matrixB = new Lint*[dimension-1];
  int matrixB_pos_x = 0; int matrixB_pos_y = 0;

  for (int i=0; i<dimension; i++) {
    Lint* line = new Lint[dimension-1];
    for (int j=0; j<dimension; j++) {
      if (i != pos_y and j != pos_x) {
        line[matrixB_pos_x] = matrixA[i][j];
        matrixB_pos_x++;
      }
    }
    if (matrixB_pos_x != 0) {
      matrixB[matrixB_pos_y] = line;
      matrixB_pos_y++;
    }
    else {
      delete[] line;
    }
    matrixB_pos_x = 0;
  }

  return matrixB;
}

Lint findDet(Lint** matrixA, const int dimension) {
  Lint det = 0;
  if (dimension == 0) {
    det = 1;
  }
  else if (dimension == 1) {
    det = matrixA[0][0];
  }
  else if (dimension == 2) {
    det = matrixA[0][0] * matrixA[1][1] - matrixA[0][1] * matrixA[1][0];
  }
  else {
    for (int i=0; i<dimension; i++) {
      // reuse form matrix? pottentially split it up into formMatrixA and formMa-
trixB?
      Lint** matrixB = findMinor(matrixA, dimension, i, 0);
      Lint matrixB_det = findDet(matrixB, dimension-1);
      Lint term = matrixA[0][i] * matrixB_det;
```

```
      if ((i+1)%2 == 0) {
        term = 0-term;
      }
      det = det + term;
    }
  }

  return det;
}

matrix formMatrixCofactors(Lint** matrixA, const int dimension) {
  Lint** matrixB = new Lint*[dimension];

  for (int i=0; i<dimension; i++) {
    Lint* line = new Lint[dimension];

    int sign = 1;
    if ((i+1)%2 == 0) {
      sign = -1;
    }
    for (int j=0; j<dimension; j++) {
      Lint** minor = findMinor(matrixA, dimension, j, i);
      Lint cofactor = findDet(minor, dimension-1) * sign;
      sign = -sign;
      line[j] = cofactor;
    }
    matrixB[i] = line;
  }

  matrix matrix_data; matrix_data.matrix = matrixB;
  matrix_data.dimension_x = dimension; matrix_data.dimension_y = dimension;
  return matrix_data;
}

matrix transposeMatrix(Lint** cofactors, const int dimension) {
  Lint** matrixB = new Lint*[dimension];

  for (int i=0; i<dimension; i++) {
    Lint* line = new Lint[dimension];
    for (int j=0; j<dimension; j++) {
      line[j] = cofactors[j][i];
    }
    matrixB[i] = line;
  }

  matrix matrixB_data; matrixB_data.matrix = matrixB;
```

```
  matrixB_data.dimension_x = dimension; matrixB_data.dimension_y = dimension;
  return matrixB_data;
}

struct float_matrix{
  Ldouble** matrix;
  int dimension_x;
  int dimension_y;
};
struct float_matrix_system {
  matrix A;
  matrix B;
  matrix X;
};

float_matrix multiplyConstant(matrix matrixA_data, const int dimension, const Lint
det) {
  Ldouble** matrixB = new Ldouble*[dimension];
  Lint** matrixA = matrixA_data.matrix;

  for (int i=0; i<dimension; i++) {
    Ldouble* line = new Ldouble[dimension];
    for (int j=0; j<dimension; j++) {
      line[j] = (1.0/det) * matrixA[i][j];
    }
    matrixB[i] = line;
  }
  float_matrix matrixB_data; matrixB_data.matrix = matrixB;
  matrixB_data.dimension_x = matrixA_data.dimension_x; matrixB_data.dimension_y =
matrixA_data.dimension_y;

  return matrixB_data;
}

float_matrix multiplyMatricies(float_matrix inverseA_data, matrix matrixB_data) {
  int dimension_x = inverseA_data.dimension_x;
  int dimension_y = inverseA_data.dimension_y;

  Ldouble** matrixC = new Ldouble*[matrixB_data.dimension_y];
  Ldouble** inverseA = inverseA_data.matrix;
  Lint** matrixB = matrixB_data.matrix;

  for (int i=0; i<dimension_y; i++) {
    Ldouble* line = new Ldouble[0];
    Ldouble result = 0;
    for (int j=0; j<dimension_x; j++) {
      result = result + inverseA[i][j] * matrixB[j][0];
```

```
    }
    line[0] = result;
    matrixC[i] = line;
  }
  float_matrix matrixC_data; matrixC_data.matrix = matrixC;
  matrixC_data.dimension_x = matrixB_data.dimension_x; matrixC_data.dimension_y =
matrixB_data.dimension_y;

  return matrixC_data;
}

Lint** StructToArray(shareStruct* struct_array, int len_array) {
  Lint** array = new Lint*[len_array];
  for (int i=0; i<len_array; i++) {
    array[i] = new Lint[2];
    array[i][0] = struct_array[i].x;
    array[i][1] = struct_array[i].y;

  }
  return array;
}

shareStruct* ArrayToStruct(Lint** array, int len_array) {
  shareStruct* share_array = new shareStruct[len_array];
  for (int i=0; i<len_array; i++) {
    shareStruct share;
    share.x = array[i][0];
    share.y = array[i][1];
    share_array[i] = share;
  }
  return share_array;
}

void writeShares(shareStruct* shares, const int num_shares, const int num_required,
string root_path) {
  cout << root_path << endl;
  for (int i=0; i<num_shares; i++) {
    shareStruct share = shares[i];
    string share_path = root_path + "share-" + to_string(share.x) + ".txt";
    ofstream share_file(share_path);
    share_file << "Share number: " << share.x << endl;
    share_file << "Share secret: " << share.y << endl;
    share_file << "Minimum share required: " << to_string(num_required) << endl <<
endl;
    share_file << "IMPORTANT: Please remind your admin that its there job to dis-
tribute and delete shares from the server";
  }
```

```
}

extern "C" Lint solveInternal(shareStruct* shares, int required) {
  inputStruct inputs;
  inputs.shares = shares;
  inputs.required = required;

  linearEquation* equations = new linearEquation[inputs.required];
  equations = constructEquations(inputs.required, inputs.shares);

  matrix_system matricies = formMatrix(equations, inputs.required);
  delete[] equations;
  Lint det = findDet(matricies.A.matrix, matricies.A.dimension_x);

  matrix cofactors = formMatrixCofactors(matricies.A.matrix, matricies.A.dimen-
sion_x);
  matrix transposition = transposeMatrix(cofactors.matrix, cofactors.dimension_x);

  float_matrix inverseA = multiplyConstant(transposition, transposition.dimen-
sion_x, det);
  float_matrix matrixC = multiplyMatricies(inverseA, matricies.B);

  Lint secret = matrixC.matrix[matrixC.dimension_y-1][0];
  return secret;
}

extern "C" void newSecretInternal(const Lint secret, const int num_shares, const
int num_required, char* root_path) {
  string str(root_path);
  const Lint prime = genPrime();
  int* poly = genPoly(num_required-1, prime, secret);

  poly = encodeSecret(poly, secret, num_required);
  shareStruct* shares = genShares(num_shares, num_required, poly, prime);

  writeShares(shares, num_shares, num_required, root_path);
}


int main() {
}
```

modules/handler/handler.py

```
from modules.auth.auth import authorised
from modules.track import *
send_status = logging.status.send_status
```

```
status = logging.status
log = logging.log
from modules.algorithms.univ import dict_key_verify
from modules.user import info as user_info
from modules.user import content as content_info
from modules.data.datetime import timestamp
from modules.data.config import read as config_read
from modules.data.database import key as db_key
from modules.data.database import encryption as db_encrpytion

import eventlet

# for this section client even calls call a handler. This handler then calls the
root handler passing the target method as an argument
# this allows for one method (the root handler) to handle overhead tasks taht are
required for every event called
# this is done to make the code simple and reduce boiler plate

class root_handler():
    def __init__(self, sio, sid, session, min_level='admin', event_name='event',
*args, **kwargs):
        self.info = None
        self.status = None
        self.sio = sio
        self.sid = sid
        self.event_name = event_name
        self.min_level = min_level
        self.session = session

        self.user_id = sio.get_session(sid)['id']
        self.user_level = sio.get_session(sid)['level']

        # permissions levels
        self.member = self.authorised(level='member')
        self.management = self.authorised(level='management')
        self.admin = self.authorised(level='admin')
        # permissions levels

        self.statface = logging.status_interface(sio, sid, self.user_id, self)

    def handle(self, method, data=None, auth=None):
        # bunch of boiler plate,
        if self.session.mode == "normal":
            self.obj.id = self.user_id
            # statface is the interface used for status messages, anytime a status
message is created this interface is also passed
            self.obj.statface = self.statface
```

```
            #checking the user calling the event is authorised based on their level
            if self.authorised(level=self.min_level):
                method(data=data)
            else:
                status("WARN", f"{self.event_name} - User not authorised",
self.statface)
        else:
            status("FAIL", f"Server is currently in {self.session.mode} and so is
not accepting calls to this event, try again later", self.statface)

    def authorised(self, level=None, username=None, mode="and", lead_username=None,
associated_username=None, *args, **kwargs):
        # an all in one function for verifying if a user is authorised for an ac-
tion in a number of diffretn ways:
        # - association
        # - level
        # - team lead
        auths = []
        if username:
            lead_username = username
            associated_username = username

        if level:
            # identifies if the user has level privalidges matchign that given
            level_auth = False
            level_list = ['member', 'management', 'admin']
            allow_levels = level_list[level_list.index(level):]

            if self.user_level in allow_levels:
                level_auth = True
            auths.append(level_auth)

        client_username = user_info.auth(user_id=self.user_id).get()['username']

        if lead_username:
            # checks if the user has leader privalidges over the target user
            leader_auth = False
            subjects_leaders = user_info.team(username=lead_username).get_lead-
ers()['leaders']

            if subjects_leaders and client_username in subjects_leaders:
                leader_auth = True
            auths.append(leader_auth)

        if associated_username:
            # your associated to a user if your in the same team or your friends
```

```
            # so if associated auth is true it means the target user is in the same
team or friends with the subject user
            associated_auth = False
            subjects_friends = user_info.friend(username=associ-
ated_username).get()['friends']
            subject_team = user_info.team(username=associated_username).get_mem-
bers()
            if subject_team:

                if client_username in subjects_friends or client_username in sub-
ject_team['members']:
                    associated_auth = True
                auths.append(associated_auth)

        if mode == "and":
            auth = all(auths)
        if mode == "or":
            auth = (True in auths)
        else:
            auth = auths
        # returns auth as either true or false
        return auth

    def _leader_check(self, leaders, username):
        leader = False
        for leader in leaders:
            if leader['username'] == username:
                leader = True
        return leader

    def is_leader(self, user_id, identifier):
        username = user_info.auth(user_id=user_id).get()['username']
        leader = False

        leaders = user_info.team(user_id=identifier, username=identifier, occupa-
tion_id=identifier, team_id=identifier).get_leaders()['leaders']
        if leaders:
            leader = self._leader_check(leaders, username)
        else:
            leaders = user_info.team(user_id=identifier).get_leaders()['leaders']
            if leaders:
                leader = self._leader_check(leaders, username)

        return leader

class auth_handler(root_handler):
```

```
    def __init__(self, sio, sid, session, min_level='admin', event_name='authenti-
cation info event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name=event_name, *args, **kwargs)
        self.obj = user_info.auth(user_id=self.user_id)

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'username') and self.admin:
            self.obj.username = data['username']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get()

    def set(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _set(self, data=None):
        if dict_key_verify(data, 'username'):
            profile.username = data['username']
        if dict_key_verify(data, 'items'):
            profile.columns = data['items']

        self.obj.set(data)

class profile_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='profile
event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name=event_name, *args, **kwargs)
        self.obj = user_info.profile(user_id=self.user_id)

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get()

    def get_permissions(self, data=None):
```

```
        self.handle(self._get_permissions, data)
        return self.info, self.status
    def _get_permissions(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
        if dict_key_verify(data, 'target_username'):
            self.obj.target_username = data['target_username']

        self.info = self.obj.get_permissions()

    def set(self, data=None):
        self.handle(self._set, data)
        return self.info, self.status
    def _set(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', data['username'], "or"):
                self.obj.username = data['username']

        self.obj.columns = list(data.keys())
        self.obj.set(data)
        status("INFO", "item(s) successfully set to provided value(s)")

    def delete(self, data=None):
        self.handle(self._delete, data)
        return self.info, self.status
    def _delete(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']

        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.obj.delete()

class occupation_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='occupation
event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level='admin',
event_name=event_name, *args, **kwargs)
        self.obj = user_info.occupation(user_id=self.user_id)

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
```

```
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get()

    def get_all(self, data=None):
        self.handle(self._get_all, data)
        return self.info, self.status
    def _get_all(self, data):
        self.info = self.obj.get_all()

    def set(self, data=None):
        self.handle(self._set, data)
        return self.info, self.status
    def _set(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']
        if dict_key_verify(data, 'occupation_id'):
            self.obj.set(data)
            status("INFO", "Occupation successfully set", self.statface)
        else:
            status("WARN", "Occupation couldnt be updated no value(s) provided",
self.statface)

    def set_request(self, data=None):
        self.handle(self._set_request, data)
        return self.info, self.status
    def _set_request(self, data=None):
        if dict_key_verify(data, 'username') and self.admin:
            self.obj.username = data['username']
        if dict_key_verify(data, 'occupation_id'):
            self.obj.set_request(data)
            status("INFO", "Occupation change request successfully created",
self.statface)
        else:
            status("WARN", "Occupation change request could not be created no
value(s) provided", self.statface)

    def get_request(self, data=None):
        self.handle(self._get_request, data)
        return self.info, self.status
    def _get_request(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']
```

```
        self.info = self.obj.get_request()

    def get_all_request(self, data=None):
        self.handle(self._get_all_request, data)
        return self.info, self.status
    def _get_all_request(self, data=None):
        self.info = self.obj.get_all_requests()

    def delete_request(self, data=None):
        self.handle(self._delete_request, data)
        return self.info, self.status
    def _delete_request(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']

        self.obj.delete_request()

    def approve_request(self, data=None):
        self.handle(self._approve_request, data)
        return self.info, self.status
    def _approve_request(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
            self.obj.approve_request()
            status("INFO", "Occupation change request successfully approved",
self.statface)
        else:
            status("FAIL", "Occupation change request unable to be approved invalid
data provided", self.statface)

    def reject_request(self, data=None):
        self.handle(self._delete_request, data)
        return self.info, self.status
    def _reject_request(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
            self.obj.reject_request()
            status("INFO", "Occupation change request successfully rejected",
self.statface)
        else:
            status("FAIL", "Occupation change request unable to be rejected invalid
data provided", self.statface)

    def create(self, data=None):
        self.handle(self._create, data)
        return self.info, self.status
    def _create(self, data=None):
```

```
        if dict_key_verify(data, "name"):
            self.obj.create(data)
            status("INFO", "Occupation successfully created", self.statface)
        else:
            self.status = logging.status("WARNING", "no value(s) provided").sta-
tus_update(None)
            status("FAIL", "Occupation could not be created invalid data provided",
self.statface)

    def edit(self, data=None):
        self.handle(self._edit, data)
        return self.info, self.status
    def _edit(self, data=None):
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
            self.obj.columns = list(data.keys())
            self.obj.edit(data)
            status("INFO", "Occupation successfully edited", self.statface)
        else:
            status("FAIL", "Occupation could not be edited invalid data provided",
self.statface)

    def delete_occupation(self, data=None):
        self.handle(self._delete_occupation, data)
        return self.info, self.status
    def _delete_occupation(self, data=None):
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
            self.obj.delete_occupation()
            status("INFO", "Occuaption successfully deleted", self.statface)

        else:
            status("FAIL", "Occupation could not be deleted invalid data provided",
self.statface)

class team_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='team
event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level='admin',
event_name=event_name, *args, **kwargs)
        self.obj = user_info.team(user_id=self.user_id)

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'username'):
```

```
            self.obj.username = data['username']
        if dict_key_verify(data, 'user_id'):
            self.obj.id = data['user_id']
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
        if dict_key_verify(data, 'team_id'):
            self.obj.team_id = data['team_id']

        if self.obj.team_id:
            self.info = self.obj.get()
        else:
            status("FAIL", "Team data could not be fetched, invalid data provided",
self.statface)

    def get_all(self, data=None):
        self.handle(self._get_all, data)
        return self.info, self.status
    def _get_all(self, data=None):
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get_all()

    def get_leaders(self, data=None):
        self.handle(self._get_leaders, data)
        return self.info, self.status
    def _get_leaders(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
        if dict_key_verify(data, 'id'):
            self.obj.id = data['id']
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
        if dict_key_verify(data, 'team_id'):
            self.obj.team_id = data['team_id']

        if self.obj.team_id:
            self.info = self.obj.get_leaders()
        else:
            status("FAIL", "Team leaders could not be fetched, invalid data pro-
vided", self.statface)

    def get_members(self, data=None):
        self.handle(self._get_members, data)
        return self.info, self.status
    def _get_members(self, data=None):
        if dict_key_verify(data, 'username'):
```

```
            self.obj.username = data['username']
        if dict_key_verify(data, 'id'):
            self.obj.id = data['id']
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
        if dict_key_verify(data, 'team_id'):
            self.obj.team_id = data['team_id']

        if self.obj.team_id:
            self.info = self.obj.get_members()
        else:
            status("FAIL", "Team members could not be fetched, invalid data pro-
vided")

    def set(self, data=None):
        self.handle(self._set, data)
        return self.info, self.status
    def _set(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
        if dict_key_verify(data, 'user_id'):
            self.obj.id = data['user_id']
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
        if dict_key_verify(data, 'team_id'):
            self.obj.team_id = data['team_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        if self.is_leader(self.user_id, self.obj.team_id) or self.management:
            if self.obj.team_id:
                self.obj.set(data)
            else:
                status("FAIL", "Team data not changed invalid data provided",
self.statface)
        else:
            status("FAIL", "Team data not changed, not authorised to alter this
team", self.statface)

    def delete_leaders(self, data=None):
        self.handle(self._delete_leaders, data)
        return self.info, self.status
    def _delete_leaders(self, data=None):
        if dict_key_verify(data, 'username'):
            self.obj.username = data['username']
        if dict_key_verify(data, 'user_id'):
            self.obj.id = data['user_id']
```

```
        if dict_key_verify(data, 'occupation_id'):
            self.obj.occupation_id = data['occupation_id']
        if dict_key_verify(data, 'team_id'):
            self.obj.team_id = data['team_id']

        if data['leaders']:
            if self.is_leader(self.user_id, self.obj.team_id) or management:
                self.obj.delete_leaders(data)
            else:
                status("FAIL", "Leader(s) was not deleted, not authorised to alter
this team", self.statface)
        else:
            status("FAIL", "Leader(s) was not deleted, not authorised to alter this
team", self.statface)

class friend_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='friend
event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level='admin',
event_name=event_name, *args, **kwargs)
        self.obj = user_info.friend(user_id=self.user_id)

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']

        self.info = self.obj.get()

    def get_requests(self, data=None):
        self.handle(self._get_requests, data)
        return self.info, self.status
    def _get_requests(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']
        if dict_key_verify(data, 'mode'):
            self.obj.mode = data['mode']
        else:
            status("WARN", "No mode provided defaulting to fetching incoming friend
request(s)", self.statface)

        self.info = self.obj.get_requests()

    def get_recomendations(self, data=None):
        self.handle(self._get_recomendations, data)
```

```
        return self.info, self.status
    def _get_recomendations(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']

        self.info = self.obj.get_recomendations(data)

    def add_request(self, data=None):
        self.handle(self._add_request, data)
        return self.info, self.status
    def _add_request(self, data=None):
        if dict_key_verify(data, 'username') and self.admin:
            self.obj.username = data['username']

        self.info = self.obj.add_request(data)

    def approve_request(self, data=None):
        self.handle(self._approve_request, data)
        return self.info, self.status
    def _approve_request(self, data=None):
        if dict_key_verify(data, 'username') and self.admin:
            self.obj.username = data['username']

        self.info = self.obj.approve_request(data)

    def reject_request(self, data=None):
        self.handle(self._remove, data)
        return self.info, self.status

    def remove_request(self, data=None):
        self.handle(self._remove, data)
        return self.info, self.status

    def remove(self, data=None):
        self.handle(self._remove, data)
        return self.info, self.status
    def _remove(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']

        self.info = self.obj.remove(data)

class post_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='post
event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name=event_name, *args, **kwargs)
```

```
        self.obj = content_info.post(user_id=self.user_id)

    def get_feed(self, data=None):
        self.handle(self._get_feed, data)
        return self.info, self.status
    def _get_feed(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', data['username'], "or"):
                post.username = data['username']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get_feed()

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', data['username'], "or"):
                post.username = data['username']
        if dict_key_verify(data, 'post_id'):
            self.obj.post_id = data['post_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get()

    def get_memories(self, data=None):
        self.handle(self._get_memories, data)
        return self.info, self.status
    def _get_memories(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            post.username = data['username']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get_memories()

    def get_user(self, data=None):
        self.handle(self._get_user, data)
        return self.info, self.status
    def _get_user(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', data['username'], "or"):
                self.obj.username = data['username']
        if dict_key_verify(data, 'items'):
```

```
            self.obj.columns = data['items']

        self.info = self.obj.get_user()

        return self.info, self.status

    def _get_friends(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get_friends()

    def get_team(self, data=None):
        self.handle(self._get_team, data)
        return self.info, self.status
    def _get_team(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', lead_username=data['username'],
mode="or"):
                self.obj.username = data['username']
        if dict_key_verify(data, 'team_id') and self.management:
            self.obj.team_id = data['team_id']
        if dict_key_verify(data, 'occupation_id') and self.management:
            self.obj.occupation_id = data['occupation_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info =self.obj.get_team()

    def get_permissions(self, data=None):
        self.handle(self._get_permissions, data)
        return self.info, self.status
    def _get_permissions(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', lead_username=data['username'],
mode="or"):
                self.obj.username = data['username']
        if dict_key_verify(data, 'post_id'):
            self.obj.post_id = data['post_id']

        self.info = self.obj.get_permissions()

    def set(self, data=None):
        self.handle(self._set, data)
        return self.info, self.status
```

```
    def _set(self, data=None):
        if dict_key_verify(data, 'username') and self.admin:
            self.obj.username = data['username']
        if dict_key_verify(data, 'content'):
            self.obj.content = data['content']
        if dict_key_verify(data, 'caption'):
            self.obj.caption = data['caption']

        if self.obj.content:
            self.obj.columns = list(data.keys())
            self.obj.set(data)
        else:
            status("FAIL", "Post could not be created, invalid image provided",
self.statface)

    def delete(self, data=None):
        self.handle(self._delete, data)
        return self.info, self.status
    def _delete(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', lead_username = data['username'],
mode="or"):
                self.obj.username = data['username']

        if dict_key_verify(data, 'post_id'):
            poster_info = content_info.post(post_id=data['post_id'],
items=['username']).get()
            if dict_key_verify(poster_info, 'posts'):
                poster_info = poster_info['posts']
                if dict_key_verify(poster_info, 'username'):
                    poster_username = poster_info['username']
                    if self.authorised('management', lead_username =
poster_username, mode="or"):
                        self.obj.post_id = data['post_id']

        self.obj.delete()

class comment_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='comment
event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name=event_name, *args, **kwargs)
        self.obj = content_info.comment(user_id=self.user_id)

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
```

```
    def _get(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', data['username'], "or"):
                self.obj.username = data['username']
        if dict_key_verify(data, 'comment_id'):
            self.obj.comment_id = data['comment_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get()

    def get_post(self, data=None):
        self.handle(self._get_post, data)
        return self.info, self.status
    def _get_post(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', data['username'], "or"):
                self.obj.username = data['username']
        if dict_key_verify(data, 'post_id'):
            self.obj.post_id = data['post_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get_post()

    def get_permissions(self, data=None):
        self.handle(self._get_permissions, data)
        return self.info, self.status
    def _get_permissions(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', lead_username=data['username'],
mode="or"):
                self.obj.username = data['username']
        if dict_key_verify(data, 'comment_id'):
            self.obj.comment_id = data['comment_id']

        self.info = self.obj.get_permissions()

    def set(self, data=None):
        self.handle(self._set, data)
        return self.info, self.status
    def _set(self, data=None):
        if dict_key_verify(data, 'username') and self.admin:
            self.obj.username = data['username']
        if dict_key_verify(data, 'post_id'):
            self.obj.post_id = data['post_id']
```

```
        self.obj.columns = list(data.keys())
        self.obj.set(data)

    def delete(self, data=None):
        self.handle(self._delete, data)
        return self.info, self.status
    def _delete(self, data=None):
        if dict_key_verify(data, 'username'):
            if self.authorised('management', lead_username=data['username'],
mode="or"):
                self.obj.username = data['username']

        if dict_key_verify(data, 'comment_id'):
            commenter_info = content_info.comment(comment_id=data['comment_id'],
items=['username']).get()
            if dict_key_verify(commenter_info, 'comments'):
                commenter_info = commenter_info['comments']
                if dict_key_verify(commenter_info, 'username'):
                    commenter_username = commenter_info['username']
                    if self.authorised('management', lead_username = com-
menter_username, mode="or"):
                        self.obj.comment_id = data['comment_id']

        self.obj.delete()

class impression_handler(root_handler):
    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'impression_id'):
            self.obj.impression_id = data['impression_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']

        self.info = self.obj.get()

    def get_comment(self, data=None):
        self.handle(self._get_content, data)
        return self.info, self.status
    def get_post(self, data=None):
        self.handle(self._get_content, data)
        return self.info, self.status
    def _get_content(self, data=None):
        if dict_key_verify(data, 'username'):
            if authorised(level='management', lead_username=data['username'],
mode='or'):
```

```
                self.obj.username = data['username']
        if dict_key_verify(data, 'post_id'):
            self.obj.post_id = data['post_id']
        if dict_key_verify(data, 'comment_id'):
            self.obj.comment_id = data['comment_id']
        if dict_key_verify(data, 'items'):
            self.obj.columns = data['items']
        if dict_key_verify(data, 'types'):
            self.obj.types = data['types']

        self.info = self.obj.get_content()

    def count(self, data=None):
        self.handle(self._count, data)
        return self.info, self.status
    def _count(self, data=None):
        if dict_key_verify(data, 'username'):
            if authorised(level='management', lead_username=data['username'],
mode='or'):
                self.obj.username = data['username']
        if dict_key_verify(data, 'impression_type'):
            self.obj.impression_type = data['impression_type']
        if dict_key_verify(data, 'post_id'):
            self.obj.post_id= data['post_id']
        if dict_key_verify(data, 'comment_id'):
            self.obj.comment_id = data['comment_id']

        self.info = self.obj.count(data)

    def set(self, data=None):
        self.handle(self._set, data)
        return self.info, self.status
    def _set(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']
        if dict_key_verify(data, 'impression_type'):
            self.obj.impression_type = data['impression_type']
        if dict_key_verify(data, 'post_id'):
            self.obj.post_id= data['post_id']
        if dict_key_verify(data, 'comment_id'):
            self.obj.comment_id = data['comment_id']

        self.info = self.obj.set(data)

    def delete(self, data=None):
        self.handle(self._delete, data)
        return self.info, self.status
```

```
    def _delete(self, data=None):
        if dict_key_verify(data, 'impression_id'):
            self.obj.impression_id = data['impression_id']

            username = self.obj.get()['username']

            if self.is_leader(self.obj.id, username) or self.admin or self.manage-
ment:
                self.info = self.obj.delete(data)

class post_impression_handler(impression_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='comment
impression event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name='event', *args, **kwargs)
        self.obj = content_info.post_impression(user_id=self.user_id)

class comment_impression_handler(impression_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='comment
impression event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name='event', *args, **kwargs)
        self.obj = content_info.comment_impression(user_id=self.user_id)

class notification_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='notifica-
tion event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name=event_name, *args, **kwargs)
        self.obj = content_info.notification(user_id=self.user_id)

    def get(self, data=None):
        self.handle(self._get, data)
        return self.info, self.status
    def _get(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']

        self.info = self.obj.get(data)
        if self.info:
            status("INFO", "Notification(s) successfully fetched", self.statface)
        else:
            status("FAIL", "Notification(s) unable to be fetched, something went
wrong", self.statface)

    def create(self, data=None):
        self.handle(self._create, data)
```

```
        return self.status
    def _create(self, data=None):
        allowed = False

        if dict_key_verify(data, 'target_id'):
            target_data = {'target_id': data['target_id']}
            target_info = self.obj.get_target_group(target_data)
            if target_info['type'] == "team" or "user":
                if self.is_leader(self.user_id, target_info['id']) or self.manage-
ment:
                    allowed = True

        if allowed:
            self.obj.create(data)
        else:
            status("FAIL", "Unable to create notification, you are unauthorised for
this action", self.statface)

    def delete(self, data=None):
        self.handle(self._delete, data)
        return self.status
    def _delete(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']

        self.obj.delete(data)

    def remove(self, data=None):
        self.handle(self._remove, data)
        return self.status
    def _remove(self, data=None):
        if dict_key_verify(data, 'username') and self.management:
            self.obj.username = data['username']
        if dict_key_verify(data, 'notification_id'):
            self.obj.notification_id = data['notification_id']

        self.obj.remove(data)

class post_slot_handler(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='post slot
event', *args, **kwargs):
        super().__init__(sio, sid, session, min_level=min_level,
event_name=event_name, *args, **kwargs)
        self.obj = timestamp()

    def get(self, data=None):
        self.info = {'post_slot_start': None, 'post_slot_end': None, 'date': None}
```

```
        slot_data = self.obj.get_slot()
        if slot_data:
            self.info['post_slot_start'] = slot_data['start']
            self.info['post_slot_end'] = slot_data['end']
            self.info['date'] = self.obj.date
            status("INFO", "Successfully fetched post slot", self.statface)
        else:
            status("CRIT", "Unable to fetch post slot, something went wrong, please
contact administrator", self.statface)

        return self.info, self.status

class server(root_handler):
    def __init__(self, sio, sid, session, min_level='admin', event_name='post slot
event', *args, **kwargs):
        if sid:
            super().__init__(sio, sid, session, min_level=min_level,
event_name=event_name, *args, **kwargs)
        self.session = session
        self.clients = self.session.clients
        self.logged_in = self.session.logged_in

        try:
            # to handle for the case where the database hasnt been decrypted but an
internal shutdown has been called
            self.obj = content_info.notification()
        except:
            pass

    def _estimate_shutdown_time(self):
        estimate = (len(self.clients) - len(self.logged_in)) +
len(self.logged_in)*2 + 10
        return estimate

    def _notify_clients(self, shutdown_by):
        server_code = config_read("miscellaneous", "ServerCode")
        notif_data = {'target_id': "all-"+server_code, 'expire_after':shutdown_by-
timestamp().now-1, 'title': "Server Shutting Down", 'content': "Its recomended to
disconnect"}
        self.obj.create(notif_data)
        status("INFO", "Shutdown notifications sent", self.statface)

    def _shutdown_services(self):
        # the below is in the documentation but doesnt work
        #self.sio.shutdown()
        status("INFO", "Background user and server services shutdown", self.stat-
face)
```

```
    def _close_new_clients(self):
        self.session.accepting_clients = False
        status("INFO", "Server closed to new clients", self.statface)

    def _disconnect_clients(self):
        status("INFO", "Disconnecting clients, bye", self.statface)
        for client in self.clients:
            self.sio.disconnect(client)

    def shutdown(self, data=None):
        self.handle(self._shutdown, data)
        return self.info, self.status
    def _shutdown(self, data=None):
        log("WARN", "Server recieved shutdown signal")
        tasks = {'notifs': False, 'disconnect_clients': False ,'background_ser-
vices': False, 'close_new_clients': False}

        if dict_key_verify(data, "time"):
            shutdown_by = timestamp().now + float(data['time'])
        else:
            shutdown_by = timestamp().now + 30
            status("WARN", "No shutdown time was provided, shutting down in 30 sec-
onds", self.statface)

        estimate = self._estimate_shutdown_time()

        sent_notifs = False
        while timestamp().now < shutdown_by - estimate:
            if not tasks['notifs']:
                self._notify_clients(shutdown_by)
                tasks['notifs'] = True
            if timestamp().now <= shutdown_by - estimate*1.2 and tasks['back-
ground_services'] == True:
                self._shutdown_services()
                tasks['background_services'] = True
                self._close_new_clients()
                tasks['close_new_clients'] = True
            eventlet.sleep(1)
        status("WARN", "Beginning final shutdown process, disconnect client",
self.statface)

        if not tasks['background_services']:
            self._shutdown_services()
        if not tasks['close_new_clients']:
            self._close_new_clients()
```

```
        self._disconnect_clients()
        log("INFO", "Server finished pre-shutdown process calling stop")

        if config_read("database", "encrypt"):
            if not self.session.db_encrypted:
                db_encrpytion(self.session).encrypt()
            else:
                log("WARN", "Not encrypting database, database was never de-
crypted")
        #self.sio.shutdown()

    def internal_shutdown(self, data):
        self.statface = None
        self._shutdown(data)

class encryption_handler():
    def __init__(self, session):
        self.obj = db_encrpytion(session)
        self.session  = session
        self.statface = None

    def decrypt(self, data=None):
        if data:
            success = self._decrypt(data)
            return success
        else:
            log("FAIL", "Could not decrypt database")
            return False
    def _decrypt(self, data):
        sss_enabled = config_read("database", "ShamirSecretSharing")
        flags = []
        if sss_enabled and dict_key_verify(data, "shares") and not dict_key_ver-
ify(data, "password"):
            flags = ["sss"]

        if dict_key_verify(data, "password") or ("sss" in flags):
            try:
                if dict_key_verify(data, "password"):
                    password = int(data['password'])
                success = self.obj.decrypt(data, flags)
                if success:
                    return True
                else:
                    log("FAIL", "Something went wrong while decrypting the data-
base")
            except:
                log("FAIL", "Something went wrong with decrypting the database")
```

```
        return False

# TESTING RBP
def test():
    pass
# TESTING RBP

if __name__ == "__main__":
    test()
```

modules/handler/outgoing.py

```
from modules.data.config import read as config_read

import requests

def post_slot(sio, sid=None):
    if sid:
        sio.emit("post_slot", room=sid)
    else:
        sio.emit("post_slot")

def send_ntfy(sio, info, sid, username):
    url = config_read("notifications", "ntfyUrl")
    if url == "https://ntfy.example.com":
        return

    user_id = sio.get_session(sid)['id']
    nfty_topic = f"{username}-{user_id[:8]}"
    if url[-1] != "/":
        url = url + "/"

    message = info['message'].encode(encoding='utf-8')
    title = info['title']

    print(f"ntfy: {nfty_topic}")
    try:
        requests.post(f"{url}{ntfy_topic}", data=message, headers={"Title": title})
    except:
        log("WARN", "Notification server cannot be reached, ensure ntfy is up and
that the provided url is correct")

def send_notification(sio, info, sid=None, username=None):
    if sid:
        send_ntfy(sio, info, sid, username)
        sio.emit("notification", info, room=sid)
```

```
    else:
        sio.emit("notification", info)
```

modules/handler/tasks.py

```
import eventlet
from modules.track.logging import log

from modules.data.database import connect as db_connect
from modules.data.config import read as config_read
from modules.data.datetime import timestamp
from modules.handler import outgoing
from modules.user.content import notification
from modules.user.info import auth as auth_info

from modules.start.start import final_startup

def user_service(sio, sid):
    user_id = sio.get_session(sid)['id']

    db = db_connect()
    db.create(None)
    log("INFO", f"Starting user service for {user_id}")

    while True:
        eventlet.sleep(30)
        user_notification_service(db, sio, sid, user_id)

def user_notification_service(db_con, sio, sid, user_id):
    notifications = notification(user_id=user_id)
    notifications.columns = ['notification_id','title', 'content']
    username = auth_info(user_id=user_id).get()['username']

    notif_queue = notifications.get_unsent()['notifications']
    if notif_queue:
        for notif in notif_queue:
            outgoing.send_notification(sio, notif, sid, username)
            db_con.cur.execute("UPDATE notifications_sent SET time_sent = ?, sent =
? WHERE user_id = ? AND notification_id = ?", (timestamp().now, True, user_id, no-
tif['notification_id']))
            db_con.commit()

def post_time_notification():
    if timestamp().is_valid_time():
        log("INFO", "Sending post time notifications")
        post_time_limit = int(config_read('posts', 'posttimelimit'))
        title = "post-" + config_read('miscellaneous', 'ServerCode')
```

```
        content = f"you have {post_time_limit} minutes to post"
        target = "all-" + config_read('miscellaneous', 'ServerCode')
        # notifications has a sepcial code for sending notifications accross the
server
        # if the target is set to "all-<unique server code>" the entire server is
notified

        notification_data = {'title': title, 'content': content, 'target_id': tar-
get, "expire_after": post_time_limit*60}
        notification().create(notification_data)
        notification_created = True
        log("INFO", "Sent post time notifications")
    else:
        notification_created = False

    return notification_created

def notification_remove(db_con):
    db_con.cur.execute("SELECT notification_id, time_created, expire_after FROM no-
tifications")
    rez = db_con.cur.fetchall()
    if rez:
        for notif in rez:
            if notif[1] + notif[2] < timestamp().now:
                notification(notification_id=notif[0]).delete()

def startup_notif():
    server_code = config_read('miscellaneous', 'servercode')
    notif_data = {'target_id': "all-"+server_code, 'title': "Server is up", 'con-
tent': "The server is now on and functioning", 'expire_after': 180}
    notification().create(notif_data)

def server_service(session):
    db = db_connect()
    db.create(None)
    log("INFO", f"Starting server background service")

    while session.mode != "normal":
        eventlet.sleep(1)
    log("INFO", "Server mode normal, continuing startup")

    final_startup(session)
    startup_notif()
    while True:
        # keeps the service running forever
        post_notification = False
        post_notif_title = "post-" + config_read('miscellaneous','ServerCode')
```

```
        db.cur.execute("SELECT time_created FROM notifications WHERE title=?",
(post_notif_title,))
        rez = db.cur.fetchall()
        if rez:
            for notif in rez:
                if timestamp().start < notif[0] and timestamp().end > notif[0]:
                    post_notification = True

        today_end = timestamp().end

        while timestamp().now < today_end:
            # keeps running this loop until the end of the day then it returns to
the while above on the start of the new day
            #print(f"now: {timestamp().now}")
            eventlet.sleep(10)
            # removes expired notifications
            notification_remove(db)

            if not post_notification:
                post_notification = post_time_notification()
```

modules/start/start.py

```
from modules.data.database import create as db_create
from modules.data.database import encryption
from modules.data.config import read as config_read
from modules.track.logging import log
import os

def main(session):
    create_directories()
    log("INFO", "Ensuring server directories")

    from modules.data.config import create as config_create
    log("INFO", "Ensuring config file")
    config_create()

    log("INFO", "Ensuring database")
    db_create().tables()

    if session.db_encrypted:
        log("INFO", "Checking encryption")
        encryption(session).mode()

def final_startup(session):
    from modules.data.datetime import timestamp as datetime_timestamp
```

```
    datetime_timestamp().generate_slot()

def create_directories():
    paths = ["data", "data/images"]
    if config_read("database", "ShamirSecretSharing"):
        paths = ["data", "data/images", "data/shares/"]
    for path in paths:
        if not os.path.exists(path):
            os.mkdir(path)
            log("INFO", f"Created new directory: {path}")

if __name__ == "__main__":
    main()
```

modules/track/logging.py

```
from datetime import datetime
from os.path import exists

class log():
    def __init__(self, level, message):
        if not hasattr(self, "message_type"):
            self.message_type = "log"
        self.time = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%SZ")
        self.level = level
        self.message = message
        self.path = "data/log.txt"

        if self.message_type == "log":
            self._create()

    def log_file_exist(self):
        file_exists = exists(self.path)
        return file_exists

    def create(self):
        # here for legacy support
        # old version required a specific call to log(*info).create
        # this has since been revambed
        pass

    def _create(self, log_string=None):
        if not log_string:
            log_string = f"{self.time} | {self.level} | {self.message}"

        if not self.log_file_exist():
```

```
            with open(self.path, 'w') as log_file:
                log_file.write(f"{self.time} | INFO | Log file created at
'{self.path}'")
        else:
            with open(self.path, 'a') as log_file:
                log_file.write(log_string)
        self.output(log_string)

    def read(self, amount):
        with open(self.path, 'r') as log_file:
            entries = log_file.readlines()
            if amount == None:
                return entries
            entries = entries[len(entries)-amount:]
            return entries

    def output(self, log_string):
        if self.message_type == "log":
            print(log_string)

class status(log):
    def __init__(self, level, message, interface=None):
        self.message_type = "status"
        super().__init__(level, message)
        self.status = {"time":self.time, "level":self.level, "message":self.mes-
sage}
        if interface:
            self.interface = interface
            self.process()

    # LEGACY METHODS
    def status_update(self, obj):
        status = {"time":self.time, "level":self.level, "message":self.message}
        if obj != None:
            obj.status = status
            obj.status_string = f"{self.time} | {self.level} | {self.message}"

        return status

    @staticmethod
    def send_status(sio, sid, status):
        sio.emit('recv_status', status, room=sid)
    # LEGACY METHODS

    def process(self):
        self.__format()
        self.__object_update()
```

```
        self._create(self.log_string)
        self.interface.send_status(self.status)

    def __object_update(self):
        if self.interface.obj != None:
            self.interface.obj.status = self.status
            self.interface.obj.status_string = self.status_string

    def __format(self):
        self.status = {"time":self.time, "level":self.level, "message":self.mes-
sage}
        self.status_string = f"{self.time} | {self.level} | {self.message}"

        user_id = self.interface.user_id
        sid = self.interface.sid
        self.log_string = f"{self.time} | {self.level} | {self.interface.user_id} |
{self.message}"

class status_interface(log):
    def __init__(self, sio, sid, user_id="Unknown", obj=None):
        self.sio = sio
        self.sid = sid
        self.user_id = user_id
        self.obj = obj
        self.path = "data/actions_log.txt"

    def send_status(self, status):
        self.sio.emit('recv_status', status, room=self.sid)

def main():
    entry = logging("INFO", "test log creation")
    entry.path = "log.txt"

if __name__ == "__main__":
    main()
```

modules/user/content.py

```
from modules.user.info import table, auth
from modules.user.info import user_id as info_user_id
from modules.user.info import auth as info_auth
from modules.user.info import friend as info_friends
from modules.user.info import team as info_team

from modules.algorithms.uuid import generate as uuid_generate
from modules.algorithms.univ import dict_key_verify
```

```
from modules.data.config import read as config_read
from modules.data.database import connect as db_connect
from modules.data.datetime import timestamp

from modules.track.logging import status

from PIL import Image
import io

class user_content(table):
    def __init__(self, user_id=None, username=None, occupation_id=None,
team_id=None, comment_id=None, post_id=None, content=None, caption=None, *args,
**kwargs):
        if not self.allowed_columns:
            self.allowed_columns = ['post_id', 'content', 'caption', 'username',
'team_id', 'date']
        super().__init__(user_id=user_id, username=username, occupation_id=occupa-
tion_id, team_id=team_id, post_id=post_id, content=content)
        self.post_id = post_id
        self.comment_id = comment_id
        self.content = content
        self.caption = caption

    @property
    def id(self):
        return self._id
    @id.setter
    def id(self, value):
        if type(value) == str:
            self.cur.execute("SELECT username FROM auth_credentials WHERE user_id =
?", (value,))
            if self.cur.fetchone():
                self.cur.execute("SELECT post_id FROM posts WHERE user_id=? AND
date=?", (value, self.date))
                rez = self.cur.fetchone()
                if rez:
                    self.post_id = rez[0]
            else:
                value = None
        else:
            value = None
        self._id = value

    @property
    def post_id(self):
        return self._post_id
```

```
    @post_id.setter
    def post_id(self, value):
        self.cur.execute("SELECT content FROM posts WHERE post_id = ?", (value,))
        if not self.cur.fetchone():
            value = None
        self._post_id = value

    @property
    def comment_id(self):
        return self._comment_id
    @comment_id.setter
    def comment_id(self, value):
        self.cur.execute("SELECT content FROM comments WHERE comment_id=?",
(value,))
        if not self.cur.fetchone():
            value = None
        self._comment_id = value

    @property
    def occupation_id(self):
        return self._occupation_id
    @occupation_id.setter
    def occupation_id(self, value):
        team_value = None
        self.cur.execute("SELECT name FROM occupations WHERE occupation_id = ?",
(value,))
        if not self.cur.fetchone():
            value = None
        else:
            self.cur.execute("SELECT team_id FROM teams WHERE occupation_id = ?",
(value,))
            rez = self.cur.fetchone()
            if rez:
                team_value = rez[0]

        self.team_id = team_value
        self._occupation_id = value

    @property
    def content(self):
        return self._content
    @content.setter
    def content(self, value):
        if type(value) != str:
            value = None
        self._content = value
```

```
class post(user_content):
    @property
    def caption(self):
        return self._caption
    @caption.setter
    def caption(self, value):
        if type(value) != str:
            value = None
        self._caption = value

    @property
    def content(self):
        return self._content
    @content.setter
    def content(self, value):
        image_formats = ['png', 'jpg']
        for form in image_formats:
            try:
                save_path = f"data/images/post_{self.id}_{self.date}.{form}"
                with Image.open(io.BytesIO(value)) as recieved:
                    recieved.save(save_path)
                break
            except:
                save_path = None
        self._content = save_path

    def __init__(self, user_id=None, username=None, occupation_id=None,
team_id=None, post_id=None, content=None, caption=None, *args, **kwargs):
        self.allowed_columns = ['post_id', 'content', 'caption', 'username',
'date']
        super().__init__(user_id=user_id, username=username, occupation_id=occupa-
tion_id, team_id=team_id, post_id=post_id, content=content, caption=caption)

    def get_feed(self):
        info = {"posts": None}
        post_feeds = []

        friend_posts_info = self.get_friends()
        if dict_key_verify(friend_posts_info, "posts"):
            friend_posts = friend_posts_info['posts']
            post_feeds.append(friend_posts)
            status("INFO", "Succesfully fetched friends' post(s)", self.statface)

        team_posts_info = self.get_team()
        if dict_key_verify(team_posts_info, "posts"):
            team_posts = team_posts_info['posts']
            post_feeds.append(team_posts)
```

```
            status("INFO", "Succesfully fetched team's post(s)", self.statface)

        for post_feed in post_feeds:
            info = {"posts": []}
            for post in post_feed:
                info['posts'].append(post)

        return info

    def get(self):
        info = {"posts":{column: None for column in self.columns}}

        for column in self.columns:
            # gots per column for the sql querey instead of quereying every field
            # this is so we can add only what the user requests as their is no easy
way of identifying onces fetched from a result
            # without hte user of "magic numbers" anyway
            if column == "username":
                column = "user_id"
            self.cur.execute(f"SELECT {column} FROM posts WHERE post_id=?",
(self.post_id,))
            rez = self.cur.fetchone()
            append_item = rez[0]
            if rez:
                if column == "user_id":
                    column = "username"
                    append_item = auth(user_id=append_item).get()['username']
                elif column == "content":
                    with open(append_item, "rb") as content:
                        append_item = content.read()
                info["posts"][column] = append_item
                status("INFO", "Succesfully fetched {column} for requested post",
self.statface)
            else:
                status("FAIL", "Could not fetch post data, invalid data provided",
self.statface)
        else:
            status("WARN", "No post data requested to be fetched, check your in-
puts", self.statface)

        return info

    def get_memories(self):
        info = {"posts":None}
        self.cur.execute(f"SELECT post_id FROM posts WHERE user_id=?", (self.id,))
        rez = self.cur.fetchall()
```

```
        if rez:
            info = {"posts":[{column: None for column in self.columns} for post in
rez]}
            for i, post_details in enumerate(rez):
                post_info = post(post_id=post_details[0])
                post_info.columns = self.columns
                info['posts'][i] = post_info.get()['posts']
            status("INFO", "Succesfully fetched post memories", self.statface)
        elif self.id:
            status("WARN", "No memories exist", self.statface)

        if not self.id:
            status("FAIL", "Could not fetch memory data, invalid data provided or
none exist", self.statface)

        return info

    def get_user(self):
        info = {"posts":{column: None for column in self.columns}}

        self.cur.execute(f"SELECT post_id FROM posts WHERE user_id=? AND date=?",
(self.id, self.date))
        rez = self.cur.fetchone()
        if rez:
            post_info = post(post_id=rez[0])
            post_info.columns = self.columns
            info = post_info.get()
            status("INFO", "Post(s) Succesfully fetched", self.statface)
        else:
            info["posts"] = None
            status("WARN", "No post(s) exist for that user", self.statface)

        if not self.id or not self.date:
            info = None
            status("FAIL", "No posts fetched, invalid inputs", self.statface)

        return info

    def get_friends(self):
        info = {'posts':None}
        friends = info_friends(user_id=self.id).get()
        if dict_key_verify(friends, 'friends'):
            friends = friends['friends']
            info = {"posts":[{column: None for column in self.columns} for friend
in friends]}

            for i, friend in enumerate(friends):
```

```
                friend_info = post(username=friend['username'])
                friend_info.columns = self.columns
                data = friend_info.get_user()['posts']
                info["posts"][i] = data
            else:
                status("WARN", "Could not fetch friend(s) post(s), no friends ex-
ist", self.statface)
        else:
            status("WARN", "Could not fetch friend(s) post(s), no friends exist",
self.statface)

        if not self.id:
            info = None
            status("FAIL", "Could not fetch friend(s) post(s), invalid data pro-
vided", self.statface)
        else:
            status("INFO", "Succesfully fetched friend(s) post(s)", self.statface)

        return info

    def get_team(self):
        info = {"posts": None}
        members_data = info_team(user_id=self.id, username=self.username, occupa-
tion_id=self.occupation_id, team_id=self.team_id)
        members_info = members_data.get_members()
        if members_info:
            members = members_info['members']
        else:
            status("WARN", "Team posts unable to be fetched, no other team mem-
bers", self.statface)
            members = None
            info = None

        if members:
            info = {"posts":[{column: None for column in self.columns} for member
in members]}

            for i, member in enumerate(members):
                member_info = post(username=member['username'])
                member_info.columns = self.columns
                data = member_info.get_user()['posts']

                info['posts'][i] = data

            if not members_data.team_id:
                status("FAIL", "Team posts unable to be fetched, invalid data pro-
vided", self.statface)
```

```
                info = None
            else:
                status("INFO", "Team posts Succesfully fetched", self.statface)
        else:
            status("WARN", "Team posts unable to be fetched, no team members pro-
vided", self.statface)

        return info

    def get_permissions(self):
        info = {"delete": False, "edit": False}

        subject = info_auth(user_id=self.id, items=['level', 'username']).get()
        if subject['level'] == "management" or subject['level'] == "admin":
            info['delete'] = True
        if subject['level'] == "admin":
            info['edit'] = True

        target_info = post(post_id=self.post_id, items=['username']).get()
        if dict_key_verify(target_info, 'username'):
            target_username = target_info['username']
            if subject['username'] == target_username:
                info['delete'] = True

            target_team_info = info_team(username=target_username).get_leaders()
            if dict_key_verify(target_team_info, 'leaders'):
                target_leaders = target_team_info['leaders']
                if subject['username'] in target_leaders:
                    info['delete'] = True

        if not self.id or not self.post_id:
            info = None
            status("FAIL", "Permissions could not be fetched, invalid data pro-
vided", self.statface)
        else:
            status("INFO", "Permissions succesfully fetched", self.statface)
        return info

    def set(self, data=None):
        self.content = data['content']
        self.caption = None
        caption_intended = False

        if dict_key_verify(data, 'caption'):
            caption_intended = True
            if len(data['caption']) <= 100:
                self.caption = data['caption']
```

```
        post_id = uuid_generate()

        valid_time = timestamp().is_valid_time()
        self.cur.execute("SELECT post_id FROM posts WHERE user_id=? AND date=?",
(self.id, self.date))

        if not self.cur.fetchone() and valid_time and ((caption_intended and
self.caption) or (not caption_intended and not self.caption)):
            self.cur.execute("INSERT INTO posts (post_id, user_id, content, cap-
tion, date) VALUES (?, ?, ?, ?, ?)", (post_id, self.id, self.content, self.caption,
self.date))
            self.db.commit()
            status("INFO", "Post successfully created", self.statface)
        else:
            status("FAIL", "Post could not be created, invalid data provided",
self.statface)

    def delete(self):
        if self.id and self.date:
            self.cur.execute("SELECT post_id FROM posts WHERE user_id=? AND
date=?", (self.id, self.date))
            rez = self.cur.fetchone()
            if rez and not self.post_id:
                self.post_id = rez[0]

        if self.post_id:
            self.cur.execute("DELETE FROM posts WHERE post_id=?", (self.post_id,))
            self.db.commit()
            status("INFO", "Post successfully deleted", self.statface)
        else:
            status("FAIL", "Post could not be deleted, invalid data provided",
self.statface)

    def _sort(self, posts):
        for post in posts:
            if dict_key_verify(post, "post_id"):
                num_likes = post_impressions(post_id=post['post_id']).count()
                post['impression_count'] = num_likes

        if len(posts) < 2:
            return posts
        mid = len(posts) // 2

        return __merge(
            left=_sort(posts[:mid]),
            right=_sort(posts[mid:]))
```

```
    def __merge(self, left, right):
        if len(left) == 0:
            return right
        if len(right) == 0:
            return left

        result = []
        index_left = index_right = 0
        while len(result) < len(left) + len(right):
            if left[index_left]['impression_count'] <= right[index_right]['impres-
sion_count']:
                result.append(left[index_left])
                index_left += 1
            else:
                result.append(right[index_right])
                index_right += 1

            if index_right == len(right):
                result += left[index_left:]
                break
            if index_left == len(left):
                result += right[index_right:]
                break

class comment(user_content):
    def __init__(self, user_id=None, username=None, occupation_id=None,
team_id=None, comment_id=None, post_id=None, content=None, *args, **kwargs):
        self.allowed_columns = ['comment_id','post_id','username','content']
        super().__init__(user_id=user_id, username=username, occupation_id=occupa-
tion_id, team_id=team_id, comment_id=comment_id, post_id=post_id, content=content)

    def get(self):
        info = {'comments':None}

        info['comments'] = {column: None for column in self.columns}
        for column in self.columns:
            if column == "username":
                column = "user_id"
            self.cur.execute(f"SELECT {column} FROM comments WHERE comment_id = ?",
(self.comment_id,))
            rez = self.cur.fetchone()
            if rez:
                comment_info = rez[0]
                if column == "user_id":
                    column = "username"
                    comment_info = auth(user_id=comment_info).get()['username']
                info['comments'][column] = comment_info
```

```
            else:
                status("FAIL", f"Comment {column} unable to be fetched, something
went wrong", self.statface)
        else:
            status("WARN", "No data requested to be fetched, check inputs",
self.statface)

        if not self.comment_id:
            info = None
            status("FAIL", "Comment unable to be fetched, invalid commentID pro-
vided", self.statface)
        else:
            status("INFO", "Succesfully fetched comment", self.statface)

        return info

    def get_post(self):
        info = {'comments':None}

        self.cur.execute("SELECT comment_id FROM comments WHERE post_id=?",
(self.post_id,))
        rez = self.cur.fetchall()
        if rez:
            info['comments'] = [{column: None for column in self.columns} for com-
ment in rez]

            for i, comment in enumerate(rez):
                self.comment_id = comment[0]
                data = self.get()
                info['comments'][i] = data['comments']
            else:
                status("WARN", "No comments related to requested post", self.stat-
face)
        else:
            status("FAIL", "Comment(s) unable to be fetched, something went wrong",
self.statface)

        if not self.post_id:
            info = None
            status("FAIL", "Comment(s) unable to be fetched, invalid commentID pro-
vided", self.statface)
        else:
            status("INFO", "Succesfully fetched comment(s)", self.statface)

        return info

    def get_permissions(self):
```

```
        info = {"delete": False, "edit": False}

        subject = info_auth(user_id=self.id, items=['level', 'username']).get()
        if subject['level'] == "management" or subject['level'] == "admin":
            info['delete'] = True
        if subject['level'] == "admin":
            info['edit'] = True

        target_info = comment(comment_id=self.comment_id, items=['username']).get()
        if dict_key_verify(target_info, 'username'):
            target_username = target_info['username']
            if subject['username'] == target_username:
                info['delete'] = True

            target_team_info = info_team(username=target_username).get_leaders()
            if dict_key_verify(target_team_info, 'leaders'):
                target_leaders = target_team_info['leaders']
                if subject['username'] in target_leaders:
                    info['delete'] = True

        if not self.id or not self.comment_id:
            info = None
            status("FAIL", "Unable to get comment permissions, invalid user or com-
mentID provided", self.statface)
        else:
            status("INFO", "Succesfully fetched comment permissions", self.stat-
face)
        return info

    def set(self, data=None):
        comment_id = uuid_generate()
        if dict_key_verify(data, 'content'):
            self.content = data['content']

        if self.content and self.post_id and self.id:
            self.cur.execute("INSERT INTO comments (post_id, comment_id, user_id,
content) VALUES (?,?,?,?)", (self.post_id, comment_id, self.id, self.content))
            self.db.commit()
            status("INFO", "Succesfully created comment", self.statface)
        else:
            status("FAIL", "Could not create comment, invalid content, postID or
user provided", self.statface)

    def delete(self):
        if self.comment_id:
            self.cur.execute("DELETE FROM comments WHERE comment_id=?", (self.com-
ment_id,))
```

```
            self.db.commit()
            status("INFO", "Succesfully deleted comment", self.statface)
        else:
            status("FAIL", "Could not delete comment, invalid commendID provided",
self.statface)

class impression(user_content):
    # this class is inherited by post_impression and comment_impression
    # because of this it uses attributes for the table names and the table fields
    # they are baked into the sql string because these are NOT user defined and so
therefor there is no security risk baking it into a string
    @property
    def impression_id(self):
        return self._impression_id
    @impression_id.setter
    def impression_id(self, value):
        self.cur.execute(f"SELECT impression_id FROM {self.table_name} WHERE im-
pression_id = ?", (value,))
        if not self.cur.fetchone():
            value = None
        self._impression_id = value

    @property
    def impression_type(self):
        return self._impression_type
    @impression_type.setter
    def impression_type(self, value):
        if not value in self.types:
            value = None
        self._impression_type = value

    @property
    def table_name(self):
        return self._table_name
    @table_name.setter
    def table_name(self, value):
        if value != "post_impressions" and value != "comment_impressions":
            value = None
        self._table_name = value
        if value:
            self.attr_name = value.replace("impressions", "id")

    @property
    def attr_name(self):
        return self._attr_name
    @attr_name.setter
    def attr_name(self, value):
```

```
        if value != "post_id" and value != "comment_id":
            value = None
        self._attr_name = value
        if not self.table_name and value:
            self.table_name = value.replace("id", "impressions")

    @property
    def attr_id(self):
        return self._attr_id
    @attr_id.setter
    def attr_id(self, value):
        root_table = "comments"
        if self.attr_name:
            if "post" in self.attr_name:
                root_table = "posts"
            self.cur.execute(f"SELECT user_id FROM {root_table} WHERE
{self.attr_name} = ?", (value,))
            if not self.cur.fetchone():
                value = None
        else:
            value = None
        self._attr_id = value

    def __init__(self, user_id=None, username=None, comment_id=None, post_id=None,
impression_id=None, impression_type=None, table_name=None, attr_name=None,
attr_id=None, *args, **kwargs):
        if not hasattr(self, "allowed_columns"):
            self.allowed_columns = ['impression_id','username','type']
        super().__init__(user_id=user_id, username=username, post_id=post_id, com-
ment_id=comment_id)
        self.impression_id = impression_id

        if not hasattr(self, "types"):
            self.types = ['like']
        if not hasattr(self, "table_name"):
            self.table_name = table_name
        if not hasattr(self, "attr_name"):
            self.attr_name = attr_name
        if not hasattr(self, "attr_id"):
            self.attr_id = attr_id

    def get(self):
        info = {'impressions':None}
        info['impressions'] = [{column: None for column in self.columns}]

        for column in self.columns:
            if column == "username":
```

```
                column = "user_id"
            self.cur.execute(f"SELECT {column} FROM {self.table_name} WHERE impres-
sion_id=?", (self.impression_id,))
            rez = self.cur.fetchone()
            if rez:
                rez_info = rez[0]
                if column == "user_id":
                    column = "username"
                    rez_info = auth(user_id=rez_info,
items=['username']).get()['username']
                info['impressions'][0][column] = rez_info
            else:
                status("FAIL", f"Impression {column} could not be fetched, some-
thing went wrong", self.statface)
        else:
            status("WARN", "No data requested to be fetched, check inputs",
self.statface)

        if not self.impression_id:
            info = None
            status("FAIL", "Impressions could not be fetched, invalid impressionID
provided", self.statface)
        else:
            status("INFO", "Impressions succesfully fetched", self.statface)

        return info

    def get_content(self):
        info = {'impressions': None}

        if self.attr_name:
            self.cur.execute(f"SELECT impression_id FROM {self.table_name} WHERE
user_id=? AND {self.attr_name}=?", (self.id,self.attr_id))
            rez = self.cur.fetchall()
            if rez:
                info['impressions'] = [{column: None for column in self.columns}
for impression_id in rez]
                for i, impression_id in enumerate(rez):
                    impression_info = self.class_type()
                    impression_info.impression_id = impression_id[0]
                    impression_info.columns = self.columns
                    impression_info = impression_info.get()['impressions'][0]
                    info['impressions'][i] = impression_info
                else:
                    status("WARN", "Post/comment has no impressions associated",
self.statface)
            else:
```

```
                status("WARN", "Impression(s) unable to be fetched, somethign went
wrong", self.statface)
        else:
            status("FAIL", "Impression(s) unable to be fetched, impression type un-
specified", self.statface)

        if not self.attr_id:
            info = None
            status("FAIL", "Impression(s) unable to be fetched, invalid post/com-
ment ID provided", self.statface)
        else:
            status("INFO", "Succesfully fetched impression(s)", self.statface)

        return info

    def count(self, data=None):
        info = {'impression_count': 0}

        if dict_key_verify(data, "impression_type") and not self.impression_type:
            self.impression_type = data['impression_type']

        if self.impression_type:
            self.cur.execute(f"SELECT COUNT(*) FROM {self.table_name} WHERE type =
? AND {self.attr_name} = ?", (self.impression_type,self.attr_id))
            rez = self.cur.fetchall()
            if rez:
                info['impression_count'] = rez[0][0]
                status("INFO", "Succesfully fetched impression count", self.stat-
face)
            else:
                status("FAIL", "Impression count unable to be fetched, something
went wrong", self.statface)

        else:
            info = None
            status("FAIL", "Impression count unable to be fetched, invalid impres-
sion type", self.statface)

        return info

    def set(self, data=None):
        if dict_key_verify(data, "impression_type"):
            self.impression_type = data['impression_type']
        impression_id = uuid_generate()

        exists = False
```

```
        self.cur.execute(f"SELECT type FROM {self.table_name} WHERE user_id=? AND
type=? AND {self.attr_name}=?", (self.id, self.impression_type, self.attr_id))
        if self.cur.fetchone():
            exists = True
        else:
            status("WARN", "Impression from this user of this type already exists
on this content", self.statface)

        if self.impression_type and self.id and self.attr_id:
            if not exists:
                self.cur.execute(f"INSERT INTO {self.table_name} (impression_id,
{self.attr_name}, user_id, type) VALUES (?, ?, ?, ?)", (impression_id,
self.attr_id, self.id , self.impression_type))
                self.db.commit()
                status("INFO", "Impression succesfully created", self.statface)
            else:
                status("FAIL", "Impression unable to be created, impression already
exists", self.statface)
        else:
            status("FAIL", "Impression unable to be created, invalid impression
type, user or post/comment ID provided", self.statface)

    def delete(self, data=None):
        if dict_key_verify(data, "impression_id"):
            self.impression_id = data['impression_id']

        if self.impression_id:
            self.cur.execute(f"DELETE FROM {self.table_name} WHERE impres-
sion_id=?", (self.impression_id,))
            self.db.commit()
            status("INFO", "Succesfully deleted impression", self.statface)
        else:
            status("FAIL", "Impression unable to be deleted, invalid impression ID
provided", self.statface)

class post_impression(impression):
    def __init__(self, user_id=None, username=None, post_id=None, impres-
sion_type=None, *args, **kwargs):
        self.allowed_columns = ['impression_id','post_id','username','type']
        self.types = ['like']
        self.table_name = "post_impressions"
        self.class_type = post_impression
        super().__init__(user_id=user_id, username=username, post_id=post_id, im-
pression_type=impression_type)

        if self.post_id:
            self.attr_id = self.post_id
```

```
    @property
    def post_id(self):
        return self._post_id
    @post_id.setter
    def post_id(self, value):
        self.cur.execute("SELECT content FROM posts WHERE post_id = ?", (value,))
        if not self.cur.fetchone():
            value = None
        self._post_id = value
        self.attr_id = value

    def get_post(self):
        info = self.get_content()
        return info

class comment_impression(impression):
    def __init__(self, user_id=None, username=None, comment_id=None, impres-
sion_type=None, *args, **kwargs):
        self.types = ['like']
        self.table_name = "comment_impressions"
        self.allow_columns = ['impression_id','comment_id','username','type']
        self.class_type = comment_impression
        super().__init__(user_id=user_id, username=username, comment_id=comment_id,
impression_type=impression_type)

        if self.comment_id:
            self.attr_id = self.comment_id

    def get_comment(self):
        info = self.get_content()
        return info

    @property
    def comment_id(self):
        return self._comment_id
    @comment_id.setter
    def comment_id(self, value):
        self.cur.execute("SELECT content FROM comments WHERE comment_id=?",
(value,))
        if not self.cur.fetchone():
            value = None
        self._comment_id = value
        self.attr_id = value

class notification(table):
    @property
```

```
    def notification_id(self):
        return self._notification_id
    @notification_id.setter
    def notification_id(self, value):
        self.cur.execute("SELECT notification_id FROM notifications WHERE notifica-
tion_id=?", (value,))
        if not self.cur.fetchone():
            value = None
        self._notification_id = value

    @property
    def target_id(self):
        return self._target_id
    @target_id.setter
    def target_id(self, value):
        self.cur.execute("SELECT user_id FROM auth_credentials WHERE user_id=?",
(value,))
        user = self.cur.fetchone()
        self.cur.execute("SELECT team_id FROM teams WHERE team_id=?", (value,))
        team = self.cur.fetchone()
        if value == "all-" + self.server_code:
            all_server = True
        else:
            all_server = False
        level = value in ['member', 'management', 'admin']

        if not (user or team or all_server or level):
            value = None
        self._target_id = value

    @property
    def title(self):
        return self._title
    @title.setter
    def title(self, value):
        if type(value) != str:
            value = None
        self._title = value

    @property
    def content(self):
        return self._content
    @content.setter
    def content(self, value):
        if type(value) != str:
            value = None
        self._content = value
```

```
    @property
    def expire_after(self):
        return self._expire_after
    @expire_after.setter
    def expire_after(self, value):
        if type(value) != float and type(value) != int:
            value = None
        self._expire_after = value

    def __init__(self, user_id=None, username=None, notification_id=None, tar-
get_id=None, title=None, content=None, expire_after=None):
        self.allowed_columns = ['notification_id', 'target_id', 'title', 'content',
'time_created']
        super().__init__(user_id=user_id, username=username)
        self.notification_id = notification_id
        self.target_id = target_id
        self.title = title
        self.content = content
        self.expire_after = expire_after

    def get_target_group(self, data=None):
        # finds out the type of id thats been provided
        # below the list displays the types that can be provided
        info = {'type': None, 'id': None}
        types = ['server', 'user', 'team', 'level']
        # "server" means that everyone is the target for the notificaion

        if dict_key_verify(data, 'target_id'):
            self.target_id = data['target_id']

        if self.target_id:
            if self.target_id == "all-"+self.server_code:
                # "all-<server code>" is the unique way of identifying a notifica-
tion to the entire server
                # its structured this way to stop any collisions with a user who
might call themselves "all" or "server"
                # as such the server code (usually a string of numbers or 12345 by
default) is banned from use in usernames
                info = {'type': types[0], 'id': self.target_id}
            elif self.target_id in ['member', 'management', 'admin']:
                info = {'type': types[3], 'id': self.target_id}
            else:
                self.cur.execute("SELECT username FROM auth_credentials WHERE
user_id = ?", (self.target_id,))
                rez = self.cur.fetchone()
                if rez:
```

```
                    username = auth(user_id=self.target_id).get()['username']
                    info = {'type': types[1], 'id': username}

                self.cur.execute("SELECT name FROM teams WHERE team_id = ?",
(self.target_id,))
                rez = self.cur.fetchone()
                if rez:
                    info = {'type': types[2], 'id': self.target_id}

        else:
            info = None

        return info

    def get_targets(self, data=None):
        # targets are the users that hte notifications should be sent to
        # targets can be specified on creation as a number of diffrent things, for
instance providing a team id allows the targeting of a  team
        # its implicit meaning its the servers job to find out what type of ID the
user is providing
        info = {'targets': None}

        if dict_key_verify(data, 'target_id'):
            self.target_id = target_id

        if self.target_id:
            target_group = self.get_target_group(self)

            if target_group['type'] == "user":
                info['targets'] = [{'user_id':info_user_id(username=tar-
get_group['id']).get()['user_id']}]
            else:

                if target_group['type'] == "server":
                    self.cur.execute("SELECT user_id FROM profile")
                elif target_group['type'] == "team":
                    self.cur.execute("SELECT user_id FROM profile INNER JOIN teams
USING(occupation_id) WHERE team_id=?", (target_group['id'],))
                elif target_group['type'] == "level":
                    self.cur.execute("SELECT user_id FROM auth_credentials WHERE
level=?", (target_group['id'],))
                rez = self.cur.fetchall()
                if rez:
                    info['targets'] = [{'user_id':target[0]} for target in rez]
        else:
            # status message
            info = None
```

```
        return info

    def get_unsent(self, data=None):
        info = {'notifications': None}

        #self.cur.execute("SELECT notification_id FROM notifications_sent WHERE
user_id=? AND sent=?",(self.id, False))
        self.cur.execute("SELECT notification_id, time_created FROM notifications
INNER JOIN notifications_sent USING(notification_id) WHERE user_id=? AND sent=?",
(self.id, False))
        rez = self.cur.fetchall()
        if rez:
            info['notifications'] = []

            queued_notifs = self._sort_notifications(rez)
            for unsent in queued_notifs:
                notification_info = notification(notification_id=unsent)
                notification_info.columns = self.columns
                notif_data = notification_info.get_notification()['notifica-
tions'][0]
                info['notifications'].append(notif_data)

        if not self.id:
            info = None
        return info

    def _sort_notifications(self, notifs):
        for i in range(len(notifs)):
            if i < len(notifs)-2:
                if notifs[i][1] < notifs[i+1][1]:
                    swap = notifs[i+1]
                    notifs[i+1] = notifs[i]
                    notifs[i] = swap

        notifs = [notif[0] for notif in notifs]
        return notifs

    def get(self, data=None):
        info = None

        if dict_key_verify(data, 'user_id'):
            self.id = data['user_id']
        if dict_key_verify(data, 'target_id'):
            self.target_id = data['target_id']
        if dict_key_verify(data, 'notification_id'):
            self.notification_id = data['notification_id']
```

```
        if self.notification_id:
            info = self.get_notification()
        elif self.id:
            info = self.get_user()
        else:
            status("WARN", "Unable to fetch notifications, must provide a valid
user or notification ID", self.statface)

        return info

    def get_user(self):
        info = {'notifications':None}

        self.cur.execute("SELECT notification_id FROM notifications_sent WHERE
user_id = ?", (self.id,))
        rez = self.cur.fetchall()
        if rez:
            info = {'notifications': []}
            # rez could be user_notifs
            for notif in rez:
                notif_id = notif[0]
                user_notification = notification(notification_id=notif_id)
                user_notification.columns = self.columns
                notification_info = user_notification.get_notification()['notifica-
tions'][0]
                info['notifications'].append(notification_info)
            else:
                status("WARN", "No notifications exist for this user", self.stat-
face)
        else:
            status("WARN", "No notifications exist for this user", self.statface)

        if not self.id:
            status("WARN", "Unable to fetch notifications, invalid user provided",
self.statface)
            info = None

        return info

    def get_group(self, data=None):
        info = {'notifications': None}

        if self.target_id:
            self.cur.execute("SELECT notification_id FROM notifications WHERE tar-
get_id=?", (self.target_id,))
            rez = self.cur.fetchall()
```

```
            if rez:
                info['notifications'] = [{column: None for column in self.columns}
for notif in rez]
                for i, notif in enumerate(rez):
                    notification_info = notification(notification_id=notif[0])
                    notification_info.columns = self.columns
                    notif_data = notification_info.get_notification()['notifica-
tions'][0]
                    info['notifications'][i] = notif_data
        else:
            # status message
            info = None
        return None

    def get_notification(self, data=None):
        info = {'notifications': None}
        info['notifications'] = [{column: None for column in self.columns}]

        if self.notification_id:
            for column in self.columns:
                self.cur.execute(f"SELECT {column} FROM notifications WHERE notifi-
cation_id = ?", (self.notification_id,))
                rez = self.cur.fetchone()
                if rez:
                    info_item = rez[0]
                    if column == "target_id":
                        info_item = self.get_target_group({'target_id':
info_item})['id']
                    info['notifications'][0][column] = rez[0]
                else:
                    status("WARN", "Notification {column} unable to be fetched,
something went wrong", self.statface)
            else:
                status("WARN", "No data requested to be fetched, check inputs",
self.statface)

        else:
            info = None
            status("WARN", "Notification unable to be fetched, invalid notification
ID provided", self.statface)
        return info

    def load_notification(self, data=None):
        # loads notifications into the "notification_sent" table. This is where no-
tifications are queued for sending when their target next logs in
        if dict_key_verify(data, 'notification_id'):
            self.notification_id = data['notification_id']
```

```
        if self.notification_id:
            notification_info = notification(notification_id=self.notification_id)
            notification_data = notification_info.get_notification()['notifica-
tions'][0]

            notification_info.target_id = self.target_id
            target_data = notification_info.get_targets()['targets']
            if target_data:
                for target in target_data:
                    self.cur.execute("INSERT INTO notifications_sent (notifica-
tion_id, user_id) VALUES (?, ?)", (self.notification_id, target['user_id']))
                    self.db.commit()

    def create(self, data=None):
        if dict_key_verify(data, 'target_id'):
            self.target_id = data['target_id']
        if dict_key_verify(data, 'title'):
            self.title = data['title']
        if dict_key_verify(data, 'content'):
            self.content = data['content']
        if dict_key_verify(data, 'expire_after'):
            self.expire_after = data['expire_after']

        notification_id = uuid_generate()
        time_created = timestamp().now

        if self.title and self.target_id:
            if not self.content:
                self.content = self.title
                status("WARN", "No notification content provided, setting content
to title", self.statface)
            if not self.expire_after:
                self.expire_after = float(config_read(section="notifications",
key="defaultexpiretime"))
                status("WARN", "No notification expire after time provided, setting
to default", self.statface)

            self.cur.execute("INSERT INTO notifications (notification_id, tar-
get_id, title, content, time_created, expire_after) VALUES (?, ?, ?, ?, ?, ?)",
(notification_id, self.target_id, self.title, self.content, time_created, self.ex-
pire_after))
            self.db.commit()
            status("INFO", "Notification successfully created", self.statface)
            pre_load_notification_id = self.notification_id
            self.load_notification({'notification_id': notification_id})
            self.notification_id = pre_load_notification_id
```

```
        else:
            status("FAIL", "Unable to create notification, invalid title or target
ID provided", self.statface)

    def delete(self, data=None):
        if dict_key_verify(data, 'target_id'):
            self.target_id = data['target_id']
        if dict_key_verify(data, 'notification_id'):
            self.notification_id = data['notification_id']

        if self.notification_id:
            self.delete_notification(data)
        elif self.target_id:
            self.delete_group(data)
        elif self.id:
            self.delete_user(data)
        else:
            status("FAIL", "Unable to delete notification, invalid user or tar-
get/notification ID provided", self.statface)

    def delete_user(self, data=None):
        if self.id:
            self.cur.execute("SELECT notification_id FROM notifications_sent WHERE
user_id=?", (self.id,))
            rez = self.cur.fetchall()
            if rez:
                for notif in rez:
                    notification_info = notification()
                    notification_info.delete_notification({'notification_id':no-
tif[0]})
                else:
                    status("WARN", "User has no notifications to be deleted",
self.statface)
            else:
                status("FAIL", "Unable to delete notification(s), something went
wrong", self.statface)
        else:
            status("FAIL", "Unable to delete notification(s), invalid user pro-
vided", self.statface)

    def delete_group(self, data=None):
        if self.target_id:

            self.cur.execute("SELECT notification_id FROM notifications WHERE tar-
get_id=?", (self.target_id,))
            rez = self.cur.fetchall()
```

```
            if rez:
                for notif in rez:
                    notification_info = notification(notification_id=notif[0])
                    notification_info.delete_notification()
                else:
                    status("WARN", "Target(s) have no notifications to be deleted",
self.statface)
            else:
                status("FAIL", "Notification(s) unable to be deleted somethign went
wrong", self.statface)
        else:
            status("FAIL", "Notification(s) unable to be deleted, invalid target ID
provided", self.statface)

    def delete_notification(self, data=None):
        if self.notification_id:
            self.cur.execute("DELETE FROM notifications_sent WHERE notifica-
tion_id=?", (self.notification_id,))
            self.db.commit()
            status("INFO", "Succesfully deleted notification", self.statface)
        else:
            status("FAIL", "Unable to delete notification, invali notification ID
provided", self.statface)

    def remove(self, data=None):
        if self.notification_id:
            self.cur.execute("DELETE FROM notifications_sent WHERE user_id = ? AND
notification_id=?", (self.id, self.notification_id,))
            self.db.commit()
            status("INFO", "Notification successfully removed", self.statface)
        else:
            status("FAIL", "Notification unable to be removed, invalid notification
ID provided", self.statface)
```

modules/user/generate.py

```
# This will essentially run one a successful registration happens
# This could become a class or function in the user_info file
## Since this file is essentially just going to be using that one
## However it will also create some of its own database entries

import sqlite3
from modules.data.config import read as config_read
from modules.data.database import connect as db_connect

from modules.algorithms.uuid import generate as uuid_generate
```

```
from modules.algorithms.uuid import long_hash as hash_string

from modules.user import info

db = db_connect()
db.create(None)
cur = db.cur


def auth_credentials(user_id, username, password, level):
    cur.execute("INSERT INTO auth_credentials (user_id, username, password, level)
VALUES (?, ?, ?, ?)", (user_id, username, password, level))

    return user_id

def profile(user_id):
    cur.execute("INSERT INTO profile (user_id) VALUES (?)", (user_id,))

    occupation_id = config_read("user", "DeafultOccupation")

    # finds wether or not the occupation_id exists
    cur.execute("SELECT name FROM occupations WHERE occupation_id = ?", (occupa-
tion_id,))
    rez = cur.fetchone()
    if rez:
        info.occupation(user_id).set({"occupation_id":occupation_id})

def team(user_id, name="friends"):
    team_id = uuid_generate()

    cur.execute("INSERT INTO teams (team_id, name, user_id) VALUES (?, ?, ?)",
(team_id, name, user_id))
    cur.execute("INSERT INTO team_leaders (team_id, user_id) VALUES (?, ?)",
(team_id, user_id))


def main(username, password, level):
    user_id = uuid_generate()
    password_hash = hash_string(password + user_id)

    auth_credentials(user_id, username, password_hash, level)
    profile(user_id)
    team(user_id)

    db.commit()
    return user_id
```

```
if __name__ == "__main__":
    main("test_user", "test_password")
```

modules/user/info.py

```
from modules.track.logging import log, status
from modules.data.config import read as config_read
from modules.data.database import retrieve
from modules.data.database import connect as db_connect
from modules.data.datetime import timestamp
from modules.algorithms.uuid import generate as uuid_generate
from modules.algorithms.univ import dict_key_verify
from modules.algorithms.recomend import recomend_friend

class table():
    def __init__(self, user_id=None, username=None, occupation_id=None, al-
lowed_columns=None, *args, **kwargs):
        self.statface = None
        self.db = db_connect()
        self.db.create(self)

        self.id = user_id
        self.username = username
        self.occupation_id = occupation_id
        if not self.allowed_columns:
            self.allowed_columns = allowed_columns
        self.columns = self.allowed_columns
        self.server_code = config_read('miscellaneous', 'servercode')

    @property
    def id(self):
        return self._id
    @id.setter
    def id(self, value):
        if type(value) == str:
            self.cur.execute("SELECT username FROM auth_credentials WHERE user_id =
?", (value,))
            if not self.cur.fetchone():
                value = None
        else:
            value = None
        self._id = value

    @property
    def username(self):
```

```
        return self._username
    @username.setter
    def username(self, value):
        self.cur.execute("SELECT user_id FROM auth_credentials WHERE username = ?",
(value,))
        if not self.cur.fetchone():
            value = None

        self._username = value

        if value:
            u_id = user_id(username=value).get()['user_id']
            if self.id != u_id:
                self.id = u_id

    @property
    def occupation_id(self):
        return self._occupation_id
    @occupation_id.setter
    def occupation_id(self, value):
        self.cur.execute("SELECT name FROM occupations WHERE occupation_id = ?",
(value,))
        if not self.cur.fetchone():
            value = None
        self._occupation_id = value

    @property
    def team_id(self):
        return self._team_id
    @team_id.setter
    def team_id(self, value):
        self.cur.execute("SELECT name FROM teams WHERE team_id = ?", (value,))
        if not self.cur.fetchone():
            value = None
        self._team_id = value

    @property
    def columns(self):
        return self._columns
    @columns.setter
    def columns(self, value):
        valid = []

        if type(value) == list:
            for column in value:
                if column in self.allowed_columns:
                    valid.append(column)
```

```
        self._columns = valid

    @property
    def date(self):
        self._date = timestamp().date
        return self._date
    @date.setter
    def date(self, value):
        self._date = value

class user_id():
    def __init__(self, username=None, *args, **kwargs):
        self.username = username
        self.db = db_connect()
        self.db.create(self)

    def get(self):
        info = {'user_id':None}

        self.cur.execute(f"SELECT user_id FROM auth_credentials WHERE username =
?", (self.username,))
        rez = self.cur.fetchone()

        if rez:
            info = {"user_id":rez[0]}

        else:
            info = None
        return info

class auth(table):
    def __init__(self, user_id=None, username=None, *args, **kwargs):
        self.allowed_columns = ["username", "level"]
        super().__init__(user_id=user_id, username=username)

    def get(self):
        info = {}

        for column in self.columns:
            info[column] = None

            self.cur.execute(f"SELECT {column} FROM auth_credentials WHERE user_id
= ?", (self.id,))
            rez = self.cur.fetchone()

            if rez:
```

```
                info[column] = rez[0]

        if not self.id:
            status("FAIL", "Invalid username provided", self.statface)
            info = None
        else:
            status("INFO", "Authorisation info successfully fetched", self.stat-
face)

        return info

    def set(self, data):
        for column in self.columns:
            value = data[column]
            rez = None

            if column == 'username':
                # the select statement was here instead of update i have no idea
why
                # ive replaced it with an update since updating a username is com-
pletetly fine
                #self.cur.execute("SELECT username FROM auth_credentials WHERE
username = ?", (value,))
                self.cur.execute("UPDATE auth_credentials SET username = ? WHERE
user_id = ?", (value, self.id))
                status("INFO", "Successfully changed username", self.statface)
            if column == 'level':
                self.cur.execute("UPDATE auth_credentials SET level = ? WHERE
user_id = ?", (value ,self.id))
                status("INFO", "Successfully changed level", self.statface)

        self.db.commit()

# V RBP: I think this is depricated and no longer in use
class level(auth):
    def __init__(self, user_id):
        super().__init__(user_id=user_id)
        self.columns = ["level"]
# ^ RBP: I think this is depricated and no longer in use

class team(table):
    def __init__(self, user_id=None, username=None, occupation_id=None,
team_id=None, *args, **kwargs):
        self.allowed_columns = ['team_id', 'name', 'occupation_id', 'user_id']
        super().__init__()
        if user_id:
            self.id = user_id
```

```
        if username:
            self.username = username
        if occupation_id:
            self.occupation_id = occupation_id
        if team_id:
            self.team_id = team_id

    @property
    def id(self):
        return self._id
    @id.setter
    def id(self, value):
        occupation_value = None
        self.cur.execute("SELECT username FROM auth_credentials WHERE user_id = ?",
(value,))
        if not self.cur.fetchone():
            value = None
        else:
            self.cur.execute("SELECT occupation_id FROM profile WHERE user_id = ?",
(value,))
            rez = self.cur.fetchone()
            if rez:
                occupation_value = rez[0]

        self.occupation_id = occupation_value
        self._id = value

    @property
    def occupation_id(self):
        return self._occupation_id
    @occupation_id.setter
    def occupation_id(self, value):
        team_value = None
        self.cur.execute("SELECT name FROM occupations WHERE occupation_id = ?",
(value,))
        if not self.cur.fetchone():
            value = None
        else:
            self.cur.execute("SELECT team_id FROM teams WHERE occupation_id = ?",
(value,))
            rez = self.cur.fetchone()
            if rez:
                team_value = rez[0]

        self.team_id = team_value
        self._occupation_id = value
```

```
    def get(self):
        info = {column: None for column in self.columns}
        for column in self.columns:
            self.cur.execute(f"SELECT {column} FROM teams WHERE team_id = ?",
(self.team_id,))
            rez = self.cur.fetchone()
            if rez:
                info[column] = rez[0]

        if not all(info.values()) and not(self.team_id):
            info = None
            status("FAIL", "Team data could not be fetched, invalid data provided",
self.statface)
        else:
            status("INFO", "Team data successfully fetched", self.statface)

        return info

    def get_all(self):
        info = {'teams': None}

        for column in self.columns:
            self.cur.execute(f"SELECT {column} FROM teams WHERE user_id IS NULL")
            rez = self.cur.fetchall()
            if rez:
                if not info['teams']:
                    info['teams'] = [{} for i in range(len(rez))]
                for i, items in enumerate(rez):
                    info['teams'][i][column] = items[0]
                status("INFO", "Team(s) successfully fetched", self.statface)
            else:
                status("FAIL", "Team(s) could not be fetched, something went
wrong", self.statface)

        return info

    def get_members(self):
        info = {'members': None}

        self.cur.execute("""SELECT auth_credentials.username FROM auth_credentials
                    INNER JOIN profile USING(user_id)
                    CROSS JOIN teams ON profile.occupation_id = teams.occupation_id
                    WHERE teams.team_id=?""", (self.team_id,))

        rez = self.cur.fetchall()
        if rez:
            info['members'] = [{'username': member[0]} for member in rez]
```

```
            status("INFO", "Team members successfully fetched", self.statface)

        if not self.team_id:
            status("FAIL", "Team members could not be fetched, invalid data pro-
vided")
            info = None

        return info

    def get_leaders(self):
        info = {'leaders': None}

        self.cur.execute("SELECT user_id FROM team_leaders WHERE team_id = ?",
(self.team_id,))
        rez = self.cur.fetchall()
        if rez:
            info['leaders'] =
[{'username':(auth(user_id=user_id).get())['username'] for user_id in leader} for
leader in rez]
            status("INFO", "Team leaders successfully fetched", self.statface)
        else:
            status("FAIL", "Team leaders could not be fetched, invalid data pro-
vided", self.statface)

        return info

    def set(self, data):
        for column in self.columns:
            if column == "name" and dict_key_verify(data, 'name'):
                self.cur.execute("UPDATE teams SET name=? where team_id=?",
(data['name'] ,self.team_id))
                self.db.commit()
                status("INFO", "Team data successfully changed", self.statface)

            if dict_key_verify(data, 'leaders'):
                current_leaders = (self.get_leaders())['leaders']
                for leader in data['leaders']:

                    exists = False
                    if current_leaders:
                        for current_leader in current_leaders:
                            if current_leader['username'] == leader['username']:
                                exists = True

                    if not exists:
                        self.cur.execute("SELECT user_id FROM auth_credentials
WHERE username = ?", (leader['username'],))
```

```
                        info = user_id(username=leader['username']).get()
                        if info:
                            self.cur.execute("INSERT INTO team_leaders (user_id,
team_id) VALUES (?, ?)", (info['user_id'], self.team_id))
                            self.db.commit()
                            status("INFO", "New leader successfully added to team",
self.statface)
                        else:
                            status("FAIL", "Leader not set, user does not exist",
self.statface)
                    else:
                        status("WARN", "This user already exists as a leader of the
team", self.statface)

    def delete_leaders(self, data):
        leaders = data['leaders']
        current_leaders = self.get_leaders()['leaders']
        if type(leaders) == str:
            leaders = [leaders]
        for leader in leaders:

            exists = False
            if current_leaders:
                for current_leader in current_leaders:
                    if current_leader['username'] == leader['username']:
                        exists = True

            if exists:
                self.cur.execute("DELETE FROM team_leaders WHERE user_id=? AND
team_id=?", (user_id(username=leader['username']).get()['user_id'],self.team_id,))
                self.db.commit()
                status("INFO", "User {leader['username']} removed as a leader from
this team", self.statface)
            else:
                status("WARN", "User {leader['username']} does not exist as a
leader of this team", self.statface)

class friend(table):
    @property
    def friend_username(self):
        return self._friend_username
    @friend_username.setter
    def friend_username(self, value):
        obj = friend(username=value)
        if not obj.username:
            value = None
        else:
```

```
            self.friend_id = user_id(username=value).get()['user_id']
        self._friend_username = value

    @property
    def friend_id(self):
        return self._friend_id
    @friend_id.setter
    def friend_id(self, value):
        obj = friend(user_id=value)
        if not obj.id:
            value = None
        self._friend_id = value

    @property
    def mode(self):
        return self._mode
    @mode.setter
    def mode(self, value):
        if value not in ['incoming', 'outgoing']:
            value = "incoming"
        self._mode = value

    def __init__(self, user_id=None, username=None, *args, **kwargs):
        self.allowed_columns = ['username', 'friend_username']
        self.mode = "outgoing"
        super().__init__(user_id=user_id, username=username)

    def get(self):
        info = {'friends':None}

        self.cur.execute("SELECT friend_id FROM friends WHERE user_id = ? AND ap-
proved = ?", (self.id, True))
        rez = self.cur.fetchall()

        info['friends'] = [auth(user_id=user[0]).get() for user in rez]

        if not self.id:
            info = None
            status("FAIL", "Friends not fetched, invalid data provided", self.stat-
face)
        else:
            status("INFO", "Friends successfully fetched", self.statface)

        return info

    def get_requests(self):
        info = {'requests': None}
```

```
        if self.mode == 'incoming':
            self.cur.execute("SELECT user_id FROM friends WHERE friend_id = ? AND
approved = ?", (self.id, False))
        else:
            self.cur.execute("SELECT friend_id FROM friends WHERE user_id = ? AND
approved = ?", (self.id, False))
        rez = self.cur.fetchall()

        if rez:
            users = [auth(user_id=user[0]).get()['username'] for user in rez]
            info['requests'] = users
            status("INFO", f"Successfully fetched {self.mode} friend request(s)",
self.statface)
        elif not self.id:
            status("FAIL", f"Could not fetch {self.mode} friend request(s), invalid
data provided", self.statface)
            info = None

        return info

    def get_recomendations(self, data):
        info = {'recomended': None}
        if dict_key_verify(data, 'amount') and isinstance(data['amount'], int):
            amount = data['amount']
        else:
            status("FAIL", "Could not fetch friend recomendation(s), invalid amount
provided or data is in wrong format", self.statface)
            return None

        depth = 3
        username = auth(user_id=self.id).get()['username']
        recomendations = recomend_friend(username, amount, depth)
        if recomendations:
            info['recomended']  = recomendations
            status("INFO", "Successfully fetched friend recomendations", self.stat-
face)
        else:
            status("FAIL", "Could not fetch friend recomendation(s), something went
wrong generating recomendation(s)", self.statface)

        return info

    def add_request(self, data):
        approved = False

        if dict_key_verify(data, 'friend_username'):
```

```
            self.friend_username = data['friend_username']
            friend_id = user_id(data['friend_username']).get()['user_id']

        if friend_id:
            # checks if the other person has added them as a friend
            # if so it accepts the other persons request and creates their own ap-
proved request
            self.cur.execute("SELECT user_id FROM friends WHERE friend_id = ? AND
user_id = ?", (self.id, friend_id))
            rez = self.cur.fetchone()
            if rez:
                self.cur.execute("UPDATE friends SET approved = True WHERE
friend_id = ? AND user_id = ?", (self.id, friend_id))
                approved = True

            # checks to see if this friend request already exists (wether accepted
or rejected)
            # if not then it makes a new unaproved friend request
            self.cur.execute("SELECT approved FROM friends WHERE user_id = ? AND
friend_id = ?", (self.id, friend_id))
            rez = self.cur.fetchone()
            if not rez:
                self.cur.execute("INSERT INTO friends (user_id, friend_id, ap-
proved) VALUES (?, ?, ?)", (self.id, friend_id, approved))
                status("INFO", "Friend request successfully created", self.stat-
face)
            elif rez[0] == False:
                status("WARN", "User already has an active friend request to this
user", self.statface)
            elif rez[0] == True:
                status("WARN", "User is already friends with other user",
self.statface)
        else:
            status("FAIL", "Could not create friend request, invalid data pro-
vided")

        self.db.commit()

    def approve_request(self, data):
        if dict_key_verify(data, 'friend_username'):
            self.friend_username = data['friend_username']
        self.cur.execute("SELECT approved FROM friends WHERE friend_id = ? AND
user_id = ?", (self.id, self.friend_id))
        rez = self.cur.fetchone()
        if rez:
            self.add_request(data)
        else:
```

```
            status("FAIL", "Friend request does not exist", self.statface)

    def reject_request(self, data):
        self.remove(data)

    def delete_request(self, data):
        self.remove(data)

    def remove(self, data):
        if dict_key_verify(data, 'friend_username'):
            self.friend_username = data['friend_username']

        if self.friend_id:
            self.cur.execute("DELETE FROM friends WHERE user_id = ? AND friend_id =
?", (self.id, self.friend_id))
            self.cur.execute("DELETE FROM friends WHERE friend_id = ? AND user_id =
?", (self.id, self.friend_id))
            status("INFO", "Friend/friend request successfully removed/rejected",
self.statface)
        else:
            status("FAIL", "Friend/friend request could not be removed/rejected,
invalid data provided", self.statface)

        self.db.commit()

class profile(table):
    @property
    def target_username(self):
        return self._target_username
    @target_username.setter
    def target_username(self, value):
        prof = profile(username=value)
        if not prof.username:
            value = None
        self._target_username = value

    def __init__(self, user_id=None, username=None, *args, **kwargs):
        self.allowed_columns = ["biography", "role", "name", "occupation_id"]
        super().__init__(user_id=user_id, username=username)

    def get(self):
        info = {}

        for column in self.columns:
            info[column] = None
```

```
            self.cur.execute(f"SELECT {column} FROM profile WHERE user_id = ?",
(self.id,))
            rez = self.cur.fetchone()
            if rez:
                info[column] = rez[0]

        if not self.id:
            status("FAIL", "Invalid username provided profile unable to be
fetched")
            info = None
        else:
            status("INFO", "Profile infomation successfully fetched")

        return info

    def get_permissions(self):
        info = {"delete": False, "edit": False}

        subject = auth(user_id=self.id, items=['level', 'username']).get()
        if subject['level'] == "management" or subject['level'] == "admin":
            info['delete'] = True
            info['edit'] = True

        if self.target_username:
            if subject['username'] == self.target_username:
                info['delete'] = True
                info['edit'] = True

            target_team_info = team(username=self.target_username).get_leaders()
            if dict_key_verify(target_team_info, 'leaders'):
                target_leaders = target_team_info['leaders']
                if subject['username'] in target_leaders:
                    info['delete'] = True

        if not self.id or not self.target_username:
            status("FAIL", "Invalid username or data provided", self.statface)
            info = None
        else:
            status("INFO", "Permissions successfully fetched", self.statface)

        return info

    def set(self, data):
        for column in self.columns:
            item = data[column]
```

```
            self.cur.execute(f"UPDATE profile SET {column} = ? WHERE user_id = ?",
(item, self.id))
            status("INFO", "Successfully changed/deleted {column}", self.statface)

        self.db.commit()

    def delete(self):
        data = {}

        for column in self.columns:
            data[column] = None
        self.set(data)

class occupation(table):
    def __init__(self, user_id=None, username=None, occupation_id=None, *args,
**kwargs):
        self.allowed_columns = ["occupation_id", "name", "description"]
        super().__init__(user_id=user_id, username=username, occupation_id=occupa-
tion_id)

    def get(self):
        info = {column: None for column in self.columns}

        if not self.occupation_id:
            self.cur.execute("SELECT occupations.occupation_id, occupations.name,
description FROM profile INNER JOIN occupations USING(occupation_id) WHERE user_id
= ?", (self.id,))
        else:
            self.cur.execute("SELECT occupation_id, name, description FROM occupa-
tions WHERE occupation_id = ?", (self.occupation_id,))
        rez = self.cur.fetchone()

        if rez:
            occupation = {'occupation_id':rez[0], 'name':rez[1], 'descrip-
tion':rez[2]}

            for column in self.columns:
                info[column] = occupation[column]

        if not rez and not self.id:
            status("FAIL", "Occupation could not be fetched: invalid data pro-
vided", self.statface)
            info = None
        else:
            status("INFO", "Occupation successfully fetched", self.statface)

        return info
```

```
    def get_request(self):
        info = {'occupation_id': None, 'approved': None}

        self.cur.execute("SELECT occupation_id, approved FROM occupation_requests
WHERE user_id = ?", (self.id,))
        rez = self.cur.fetchone()
        if rez:
            info['occupation_id'] = rez[0]
            info['approved'] = rez[1]
            status("INFO", "Occupation requests fetched successfully")
        else:
            status("FAIL", "Occupation requests could not be fetched invalid data
provided", self.statface)
            info = None

        return info

    def get_all_requests(self):
        info = {'requests': None}

        self.cur.execute("SELECT user_id, occupation_id FROM occupation_requests
WHERE approved = ?", (False,))
        rez = self.cur.fetchall()

        if rez:
            info['requests'] = [{'username': auth(user_id=re-
quest[0]).get()['username'], 'occupation_id': request[1]} for request in rez]
            status("INFO", "Occupation requests successfully fetched", self.stat-
face)
        else:
            status("FAIL", "Occupation requests could not be fetched something went
wrong", self.statface)
        return info

    def set(self, data):
        occupation_id = data["occupation_id"]

        self.cur.execute("SELECT name FROM occupations WHERE occupation_id = ?",
(occupation_id,))
        if self.cur.fetchone():
            self.cur.execute("UPDATE profile SET occupation_id = ? WHERE user_id =
?", (occupation_id, self.id))
            status("INFO", "Occupation successfully updated", self.statface)
        else:
            status("FAIL", "Occupation could not be updated invalid data provided",
self.statface)
```

```
        self.db.commit()

    def set_request(self, data):
        occupation_id = data['occupation_id']

        self.cur.execute("SELECT approved FROM occupation_requests WHERE user_id =
?", (self.id,))
        if self.cur.fetchone():
            self.delete_request()
            status("INFO", "Removing previous occupation change request",
self.statface)

        self.cur.execute("SELECT name FROM occupations WHERE occupation_id = ?",
(occupation_id,))
        if self.cur.fetchone():
            self.cur.execute("INSERT INTO occupation_requests (user_id, occupa-
tion_id, approved) VALUES (?, ?, ?)", (self.id, occupation_id, False))
        else:
            status("FAIL", "Occupation change request could not be made invalid oc-
cupation_id provided", self.statface)

        self.db.commit()

    def approve_request(self):
        self.cur.execute("SELECT occupation_id FROM occupation_requests WHERE ap-
proved = ? AND user_id = ?", (False, self.id,))
        if self.cur.fetchone():
            self.cur.execute("UPDATE occupation_requests SET approved = ? WHERE
user_id = ?", (True, self.id))
            self.cur.execute("SELECT occupation_id FROM occupation_requests WHERE
user_id = ?", (self.id,))
            rez = self.cur.fetchone()
            if rez:
                self.set({'occupation_id': rez[0]})
                status("INFO", "Occupation change request successfully approved",
self.statface)
            else:
                status("CRIT", "Occupation change request approved but not changed
in the user entry, contact admin", self.statface)
        else:
            status("FAIL", "Occupation change request from that user does not exist
or has already been approved", self.statface)

        self.db.commit()

    def reject_request(self):
```

```
        self.delete_request()

    def delete(self):
        self.cur.execute("UPDATE profile SET occupation_id = ? WHERE user_id = ?",
(None, self.id))
        self.db.commit()
        status("INFO", "Occupation no longer associated with user", self.statface)

    def delete_request(self):
        self.cur.execute("DELETE FROM occupation_requests WHERE user_id = ?",
(self.id,))
        self.db.commit()
        status("INFO", "Occupation change request successfully deleted", self.stat-
face)

    def get_all(self):
        info = {'occupations':None}

        self.cur.execute("SELECT occupation_id, name, description FROM occupa-
tions")
        rez = self.cur.fetchall()

        if rez:
            occupations = [{'occupation_id':occupation[0], 'name':occupation[1],
'description': occupation[2]} for occupation in rez]
            info['occupations'] = occupations
            status("INFO", "Occupation(s) successfully fetched", self.statface)
        else:
            status("FAIL", "Occupation(s) could not be fetched", self.statface)

        return info

    def create(self, data={'name': None, 'description': None}):
        occupation_uuid = uuid_generate()
        team_uuid = uuid_generate()
        name = data['name']
        description = data['description']

        self.cur.execute("INSERT INTO occupations(occupation_id, name, description)
VALUES (?, ?, ?)", (occupation_uuid, name, description))
        self.cur.execute("INSERT INTO teams (team_id, name, occupation_id) VALUES
(?, ?, ?)", (team_uuid, name, occupation_uuid))
        self.db.commit()

    def edit(self, data):
        if 'occupation_id' in data and not self.occupation_id:
            self.occupation_id = data['occupation_id']
```

```
        for column in self.columns:
            if column == "occupation_id":
                continue
            value = data[column]
            self.cur.execute(f"UPDATE occupations SET {column} = ? WHERE occupa-
tion_id = ?", (value, self.occupation_id))
        self.db.commit()

    def delete_occupation(self, data=None):
        if dict_key_verify(data, "occupation_id") and not self.occupation_id:
            self.occupation_id = data['occupation_id']
        self.cur.execute("DELETE FROM occupations WHERE occupation_id = ?",
(self.occupation_id,))
        self.db.commit()

def main():
    log("WARN", "modules/user/info.py has been called as main. This file is not in-
tended to run solo. Please use main.py or modules/handler/handler.py")

if __name__ == "__main__":
    main()
```

dockerfile

```
FROM python:3.11-alpine
WORKDIR /
ADD main.py .
ADD modules ./modules
RUN pip install python-socketio eventlet pathlib configparser datetime pillow py-
thon-dotenv
CMD python -u ./main.py
```

Docs/'Guide to encrypting the database.md'

```
# Overview
This document is designed to guide an administrator through setting up encryption
on their BeOpen database. This can be a good idea for increased security and ease
of response to a breach. If you have database encryption in event of a breach all
you have to do is shutdown the server application, this encrypts the database imme-
diately.

Remember you can (while logged in on an admin account) shutdown the server from
your settings panel.

# Options
Before enabling encryption and getting it setup you have to consider some options
available to you. Standard encryption simply utilises a single master password
```

which you can use to decrypt the database from any active client device while the server is in "decrypt" mode.

You also have the option of enabling Shamir secret sharing. This allows you to create a number of "shares", you can then hand out shares to trusted colleagues or friends, in the event you as the administrator ever loses the master password you can ask for a set number of these shares to be given back to you, inputting these shares into the "decrypt" screen of the client will decrypt the database and reconstruct your master password.

You can decide how many shares are required to reconstruct your master password and how many shares you want to create. Its completely up to you. The only limitations is that the number of shares created must be less than 20 and the number of shares needed for reconstruction must be less than 7. These parameters can be changed in the configuration file, under the database section.

 # Guide

1) Decide upon a master password, note your master password must be an integer. We recommend that this integer is made to be significantly large, short common integers may be easily guessed or easy to crack.
2) Create a text file at the path "data/encryptconfig.txt" (This path is configurable in the database section of your configuration file) and type your master password into this file.
3) Enable encryption in the configuration file by setting "EncryptDatabase" to "true".
4) If you want Shamir secret sharing enable this in the same section of your configuration file by setting "ShamirSecretSharing" to "true". Additionally change the values of "MinimumShares" and "NumberOfShares" to your preferred values.
5) Launch the server, if all goes according to plan the server will launch normally and you will be able to start any client and enter the decryption credentials.

Distribution of shares

If you used Shamir secret sharing your shares will now be sat as a collection of text files in (by default) data/shares. These text files will NOT be automatically deleted and so deleting these text files is left up to you as the administrator.

When you give someone a share make sure they remember their share number and share secret. If you know your share secret but cannot remember your share number it is not possible to use the share. The minimum shares required for reconstruction of the master password is considered public so this fact is also included on all shares. However this number is also stored in the configuration file of the server.

Fail

Encryption

If the encryption fails in anyway the server will log the problem and shutdown. Have a good read of the server logs, the most common issue may be that the shares

generated could not reconstruct the original key. If this is the case simply try again or use a shorter master password.

Server shutdown (ungraceful)
If the server suddenly lost power or was unable to perform a graceful shutdown for any reason the unencrypted database will be left on the system. This happens to avoid the risk of data loss. In this event:
1) Backup the encrypted database and the unencrypted database
2) Set encryption to false in the configuration file
3) Decrypt the database from any client
4) Shutdown the server again (gracefully)
5) Delete the database in the server directory and replace it with the previous version you backed up
6) You can then re-enable encryption and go through the process of setting that up again. If you use the same master password you do NOT have to re distribute the Shamir secret shares. However a set of new shares may be generated simply delete these files.

Dos and don'ts
If you use Shamir secret sharing do NOT change the "MinimumShares" configuration even after the encryption has successfully happened and the shares have been generated. If for some reason this option does change contact share holders to see if they or anyone else knows the correct value, without this value the master password cannot be re-constructed.

Do not manually change or alter any files unless instructed to do so by this guide. Changing configuration options while the server is running can lead to loss of data.

Do not share your master password with anyone else, if you wish to have a "backup" please use the Shamir secret sharing feature built into the server.

Client

main.py

```
import kivy
from kivymd.app import MDApp
from kivy.lang import Builder
from kivy.clock import Clock

from kivymd.uix.label import MDLabel
from kivymd.uix.button import MDIconButton, MDRaisedButton
from kivymd.uix.behaviors.magic_behavior import MagicBehavior
```

```
from kivymd.uix.textfield import MDTextField
from kivymd.uix.list import  OneLineAvatarIconListItem, TwoLineAvatarIconListItem,
ThreeLineAvatarIconListItem, IconRightWidget, MDList
from kivymd.uix.fitimage import FitImage
from kivymd.uix.snackbar import Snackbar
from kivymd.uix.menu import MDDropdownMenu
from kivymd.uix.bottomnavigation import MDBottomNavigationItem
from kivymd.uix.list import IRightBodyTouch
from kivy.uix.camera import Camera

from kivymd.uix.screen import MDScreen
from kivymd.uix.screenmanager import MDScreenManager
from kivy.core.window import Window
from kivymd.uix.controllers import WindowController
from kivymd.uix.dialog import MDDialog
from kivymd.uix.button import MDFlatButton

from kivymd.uix.boxlayout import MDBoxLayout
from kivymd.uix.relativelayout import MDRelativeLayout

import socketio
import os

## remove before production
import time
## remove before production

kivy.require('2.1.0')
__version__ = "0.0.2"

# IMPORTS
import uuid as uniqueid
from modules.session.session import session_info, wait, db
from modules.session.time import timestamp
from modules.session.session import setting as setting_info
from modules.handler.request import request, account_page
from modules.handler.info import image as image_info
def generate_uuid():
    uuid = str(uniqueid.uuid4())
    return uuid

def dict_key_verify(dictionary, keys, mode="and", *args, **kwargs):
    if mode != "and" and mode != "or":
        mode = "and"
    if type(keys) != list:
        keys = [keys]
```

```
    verified = []
    if type(keys) != list:
        keys = [keys]

    for key in keys:
        if type(dictionary) != dict or key not in dictionary or not diction-
ary[key]:
            verified.append(False)
        else:
            verified.append(True)

    if mode == "and":
        if all(verified) == True:
            return True
    if mode == "or":
        if True in verified:
            return True
    return False

def go_to(string, previous_line, file):
    line = previous_line
    while line != string:
        line = file.readline().strip()
    return line

def read_to(string, file):
    line = file.readline().strip()
    lines = []
    while line != string:
        lines.append(line)
        line = file.readline().strip()
    return lines

def get_dialog_content(dialog_title):
    with open("./data/assets/help.txt", "r") as f:
        line = f.readline().strip()
        line = go_to(f"[{dialog_title}:START]", line, f)
        line = go_to(f"(title:START)", line, f)
        title = read_to(f"(title:END)", f)
        line = go_to(f"(body:START)", line, f)
        body_lines = read_to(f"(body:END)", f)

    body = ""
    for line in body_lines:
        body += line + "\n"
    body = body[:-2]
    return {'title': title[0], 'body': body}
```

```
def open_help(app, page, dialog_title):
    content = get_dialog_content(dialog_title)
    page.dialog_help = HelpDialog(page, app, title=content['title'], text=con-
tent['body'])
    page.dialog_help.open()
# IMPORTS

#================== socketio START ==================
sio = socketio.Client()
session = session_info()

# http://localhost:9999
def start_client(sio, url):
    print("DEBUG: starting socketio client...")
    try:
        if not url:
            url = "http://localhost:9999"
        print(f"DEBUG: Connecting with url {url}")
        sio.connect(url)
        print("DEBUG: socketio client online!")
        return True
    except:
        print("DEGUB: socketio client failed to connect!")
        return False

def stop_client(sio):
    print("DEBUG: stopping socketio client...")
    sio.disconnect()

# connect/disconnect START
@sio.event
def connect():
    print("DEBUG: Connected!")

@sio.event
def connect_error(data):
    print("DEBUG: connection error")

@sio.event
def disconnect():
    print("Disconnected")
# connect/disconnect END

@sio.event
def recv_status(data):
    session.status = data
```

```
    string = f"{data['time']} | {data['level']} | {data['message']}"
    print(string)
#:import Snackbar kivymd.uix.snackbar.Snackbar

# auth START
@sio.event
def recv_token(data):
    wait(session).wait_username()
    session.auth_tokens.append(data['token'])
    db().execute("INSERT INTO tokens(token, username, expire) VALUES(?, ?, ?)",
(data['token'], session.username, data['expire']))

def login_cred(username="user", password="pass"):
    data = {'username': username, 'password': password}
    sio.emit('login', data, callback=auth)

def auth(callback, data):
    pass
# auth END

# other events START
@sio.event
def notification(data):
    pass
# other events END

#================== socketio END ==================


#================== kivy START ==================

# Utility START
class ExpandText(MDLabel):
    pass

class ExpandPage(MDScreen):
    def __init__(self, expand_text, banner, previous_page, **kwargs):
        super(ExpandPage, self).__init__(**kwargs)
        self.expand_text = expand_text
        self.previous_page = previous_page
        self.load_content()
        self.toolbar.title = banner

    @property
    def expand_text(self):
        return self._expand_text
    @expand_text.setter
```

```
    def expand_text(self, value):
        if type(value) != str or type(value) != list:
            if type(value) == str:
                value = [value]
        else:
            value=None
        self._expand_text = value

    def load_content(self):
        self.text_area.clear_widgets()
        for text in self.expand_text:
            item = ExpandText(text=text)
            self.text_area.add_widget(item)

    def back(self, app):
        app.set_screen(self.previous_page.name, "right")
        app.sm.remove_widget(self)

class HelpDialog(MDDialog):
    def __init__(self, page, app, **kwargs):
        kwargs["buttons"] = [MDFlatButton(text="Close", on_release=self.close,
theme_text_color = "Custom", text_color=app.theme_cls.primary_color), MDFlatBut-
ton(text="Turn off help?", on_release=self.settings, theme_text_color = "Custom",
text_color=app.theme_cls.primary_color)]
        super().__init__(**kwargs)
        self.page = page
        self.app = app

    def close(self, button):
        self.page.dialog_help.dismiss()

    def settings(self, button):
        self.close(button)
        if self.page.name == "SettingsPageScreen":
            return
        settings_screen_name = "SettingsPageScreen"
        settings_screen = SettingsPage(self.page, name=settings_screen_name)
        self.app.sm.add_widget(settings_screen)

        self.app.set_screen(settings_screen_name, "left")
# Utility END

# HomePage START
class HomeSwiper(MDBoxLayout):
    @property
    def username(self):
        return self._username
```

```
    @username.setter
    def username(self, value):
        self.ids.username.text = value
        self._username = value

    @property
    def caption(self):
        return self._caption
    @caption.setter
    def caption(self, value):
        if value == None:
            value = ""
        self.ids.caption.text = value
        self._caption = value

    @property
    def content(self):
        return self._content
    @content.setter
    def content(self, value):
        self.load_image(value)
        self.ids.content.source = self.image.path
        self._content = self.image.path

    def __init__(self, page, post_id, **kwargs):
        super().__init__(**kwargs)
        self.post_id = post_id
        self.page = page
        self.action_menu = None

        self.load_content()

    def load_image(self, value):
        self.image = image_info(self.post_id)
        self.image.load(value)
        self.ids.content.source = self.image.path

    def load_content(self):
        post_data = {'post_id': self.post_id}
        post_content = request(sio, session).emit('post_get', post_data)['posts']

        self.username = post_content['username']
        self.caption = post_content['caption']
        self.content = post_content['content']

        user_impression_data = {'items': ['username'], 'post_id': self.post_id,
'impression_type': "like"}
```

```
        post_likes = request(sio, session).emit('post_impression_get_post',
user_impression_data)['impressions']

        impression_data = {'post_id': self.post_id, 'impression_type': "like"}
        num_post_likes = request(sio, session).emit('post_impression_count', im-
pression_data)['impression_count']

        self.ids.like_number.text = str(num_post_likes)
        if post_likes:
            for like in post_likes:
                if self.page.username in like['username']:
                    self.ids.like.icon = "heart"

    def like(self):
        server = request(sio, session)

        if self.ids.like.icon == "heart-outline":
            self.ids.like.icon = "heart"

            data = {'impression_type': "like", 'post_id': self.post_id}
            server.emit('post_impression_set', data, None)
            self.ids.like_number.text = str(int(self.ids.like_number.text) + 1)
        else:
            self.ids.like.icon = "heart-outline"

            data = {'impression_type': "like", 'post_id': self.post_id, 'items':
['username','impression_id']}
            impression_info = server.emit('post_impression_get_post', data)

            if dict_key_verify(impression_info, 'impressions'):
                for impression in impression_info['impressions']:
                    if impression['username'] == self.page.username:
                        impression_id = impression['impression_id']

                data = {'impression_type': "like", 'impression_id': impression_id}
                server.emit('post_impression_delete', data, None)
                self.ids.like_number.text = str(int(self.ids.like_number.text) - 1)

    def switch_to_comments(self, app, direction='up'):
        comments_screen_name = "CommentsPageScreen_"+self.post_id
        comments_screen = CommentsPage(self.post_id, name=comments_screen_name)
        app.sm.add_widget(comments_screen)
        app.set_screen(comments_screen_name, direction)

    def post_options(self, app, direction='right'):
        data = {'post_id': self.post_id}
```

```
        delete_allowed = request(sio, session).emit("post_get_permissions",
data)['delete']

        if delete_allowed:
            profile_item = {'text': "view profile", 'viewclass': "OneLineListItem",
'on_release': lambda x=app: self.switch_to_account(app)}
            delete_item = {'text': "delete post", 'viewclass': "OneLineListItem",
'on_release': lambda: self.delete_post()}
            items = [profile_item, delete_item]
            self.action_menu = MDDropdownMenu(caller=self.account_button,
items=items, width_mult=3)
            self.action_menu.open()
        else:
            self.switch_to_account(app, direction)

    def delete_post(self):
        if self.action_menu:
            self.action_menu.dismiss()
        data = {'post_id': self.post_id}
        request(sio, session).emit("post_delete", data, None)
        self.page.home_swiper_grid.remove_widget(self)
        if len(self.page.home_swiper_grid.children) == 1:
            self.page.load_home()

    def switch_to_account(self, app, direction='right'):
        if self.action_menu:
            self.action_menu.dismiss()
        self.page.switch_to_account(app, self.username, direction)

class HomeLoadButton(MDBoxLayout):
    def __init__(self, **kwargs):
        if 'home_obj' in kwargs and kwargs['home_obj']:
            if kwargs['home_obj']:
                self.home_obj =  kwargs['home_obj']
            del kwargs['home_obj']
        super().__init__(**kwargs)

    def load_content(self):
        self.home_obj.home_swiper_grid.remove_widget(self)
        self.home_obj.load_home()

class NoPostLabel(MDBoxLayout):
    pass

class MemoriesMonth(MDBoxLayout):
    def get_memories_swiper_height(self):
        swiper_height = Window.height * 0.8 * 0.02
```

```
        return swiper_height

class SwiperMagicButton(MagicBehavior,MDIconButton):
    pass

class MemoriesSwiper(MDBoxLayout):
    pass

class OccupationPageButton(MDRaisedButton):
    def __init__(self, tab, page, **kwargs):
        super().__init__(**kwargs)
        self.tab = tab
        self.page = page

    def switch_to_occupation(self, app):
        self.tab.switch_to_occupation(app)

class OrganisationBottomItem(MDBottomNavigationItem):
    def __init__(self, page, username, **kwargs):
        super().__init__(**kwargs)
        self.page = page
        self.load_content()
        self.username = username

    def load_content(self):
        occupation_button = OccupationPageButton(self, self.page)
        level = request(sio, session).emit('auth_get')['level']
        if level != "member":
            self.occupation_button_area.add_widget(occupation_button)

    def switch_to_occupation(self, app, direction="left"):
        occupation_screen_name = "OccupationPageScreen"
        occupation_screen = OccupationPage(self.page, name=occupation_screen_name)

        app.sm.add_widget(occupation_screen)
        app.set_screen(occupation_screen_name, direction)

    def switch_to_team(self, app, direction="left"):
        team_screen_name = "TeamPageScreen"
        team_screen = TeamPage(self.page, self.username, name=team_screen_name)

        app.sm.add_widget(team_screen)
        app.set_screen(team_screen_name, direction)

class MonthListItem(OneLineAvatarIconListItem):
    def __init__(self, date, posts, month_list, **kwargs):
        super().__init__(**kwargs)
```

```
        self.month_list = month_list
        self.posts = posts
        self.date = date

    def day_view(self):
        day_list = DayList(self.date, self.posts, self.month_list)
        self.month_list.page.root_scroll.remove_widget(self.month_list)
        self.month_list.page.root_scroll.add_widget(day_list)

class MonthList(MDBoxLayout):
    def __init__(self, page, **kwargs):
        super().__init__(**kwargs)
        self.page = page
        self.back_stack = [self]
        self.load_content()

    def load_content(self):
        month_list = ["January", "February", "March", "April", "May", "June",
"July", "August", "September", "October", "November", "December"]
        data = {'items': ['post_id', 'date']}
        post_response = request(sio, session).emit("post_get_memories", data)

        if dict_key_verify(post_response, 'posts'):
            posts = post_response['posts']
            post_months = {}

            for post in posts:
                if dict_key_verify(post, 'date'):
                    date = post['date']
                    date_list = date.split("-")
                    if dict_key_verify(post_months, date):
                        post_months[date].append(post)
                    else:
                        post_months[date] = [post]

            for post_group in post_months:
                date = post_group.split("-")
                month_name = month_list[int(date[1])-1]
                date_string = date[0] + ": " + month_name

                item = MonthListItem(date, post_months[post_group], self,
text=date_string)
                self.scroll.add_widget(item)

        else:
            item = OneLineAvatarIconListItem(text="No memories :(")
            self.scroll.add_widget(item)
```

```
class DayListItem(OneLineAvatarIconListItem):
    def __init__(self, date, post, day_list, **kwargs):
        super().__init__(**kwargs)
        self.day_list = day_list
        self.post = post
        self.date = date

    def post_open(self):
        memory = MemoryLayout(self.post, self.day_list)
        area = self.day_list.month_list.page.root_scroll
        area.clear_widgets()
        area.add_widget(memory)

class DayList(MDBoxLayout):
    def __init__(self, date, posts, month_list, **kwargs):
        super().__init__(**kwargs)
        self.month_list = month_list
        self.posts = posts
        self.date = date
        self.back_stack = self.month_list.back_stack

        self.back_stack.append(self)
        self.load_content()

    def load_content(self):
        for post in self.posts:
            date_string = self.date[2]
            item = DayListItem(self.date, post, self, text=date_string)
            self.scroll.add_widget(item)

    def back(self):
        last = len(self.back_stack)-1
        self.month_list.page.root_scroll.clear_widgets()
        self.month_list.page.root_scroll.add_widget(self.back_stack[last-1])
        self.back_stack.pop(last)

class MemoryLayout(MDBoxLayout):
    def __init__(self, post, day_list, **kwargs):
        super().__init__(**kwargs)
        self.post = post
        self.day_list = day_list
        self.page = self.day_list.month_list.page
        self.back_stack = self.day_list.back_stack

        self.back_stack.append(self)
        self.load_content()
```

```
    def remove_username(self, post):
        post.ids.username.text = self.post['date']
        post.ids.profile_area.remove_widget(post.ids.account_button)

    def load_content(self):
        post = HomeSwiper(self.page, self.post['post_id'])
        self.remove_username(post)
        self.post_area.add_widget(post)

class HomePage(MDScreen, WindowController):
    def __init__(self, username=None, app=None, **kwargs):
        super().__init__(**kwargs)
        post_login()

        self.username = username
        self.app = app

        self.account_screens = []
        self.notifications_screens = []
        self.settings_screen = None
        self.settings_screen_name = None

        self.posts_displayed = []
        self.post_exist = False
        self.camera_widget_exists = False
        self.camera_page_screen = None

        self.organisation_item_exists = False

        self.post_slot = request(sio, session).emit("post_slot_get")
        self.posted_today()
        self.load_content()

        self.selected_tab = "Home"
        self.help_tool = ["help", lambda x: self.open_help(self.app)]

    def login(self):
        login_cred()

    def load_content(self):
        self.load_home()
        self.load_memories()
        self.load_organisation()
        Clock.schedule_interval(self.check_post_time, 1)
        self.load_toolbar()
```

```
    def load_toolbar(self):
        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)
        else:
            toolbar_len = len(self.toolbar.right_action_items)
            for i in range(toolbar_len):
                if self.toolbar.right_action_items[i][0] == "help":
                    return
            new_toolbar = [self.help_tool] + self.toolbar.right_action_items[0:]
            self.toolbar.right_action_items = new_toolbar

    def on_tab_press(self, name):
        self.selected_tab = name

    def open_help(self, app):
        level = request(sio, session).emit('auth_get')['level']

        if self.selected_tab == "Home":
            open_help(app, self, "Home")
        elif self.selected_tab == "Memories":
            open_help(app, self, "Memories")
        elif self.selected_tab == "Organisation" and level == "member":
            open_help(app, self, "Organisation")
        elif self.selected_tab == "Organisation" and level != "member":
            open_help(app, self, "Organisation-admin")

    # SWITCHING
    def switch_to_settings(self, app, direction='left'):
        settings_screen_name = "SettingsPageScreen"
        settings_screen = SettingsPage(self, name=settings_screen_name)
        app.sm.add_widget(settings_screen)

        app.set_screen(settings_screen_name, direction)

    def switch_to_account(self, app, username=None, direction='right', *args,
**kwargs):
        if not username:
            username = self.username
        account_screen_name = "AccountPageScreen_"+username

        account_screen = AccountPage(username, self, name=account_screen_name)
        app.sm.add_widget(account_screen)
        app.set_screen(account_screen_name, direction)
```

```
    def switch_to_notifications(self, app, username=None, direction='right', *args,
**kwargs):
        if not username:
            username = self.username
        notifications_screen_name = "NotificationsScreen-"+username
        notifications_screen = NotificationsPage(username, self, name=notifica-
tions_screen_name)
        app.sm.add_widget(notifications_screen)
        app.set_screen(notifications_screen_name, direction)

    # FETCHING DATA
    def fetch_posts(self):
        data = {'items': ['post_id']}
        posts = request(sio, session).emit('post_get_feed', data)['posts']
        post_list = []

        if self.post_made:
            post = request(sio, session).emit("post_get_user")
            if dict_key_verify(post, "posts"):
                post = post['posts']
                if dict_key_verify(post, "post_id"):
                    if post['post_id'] not in self.posts_displayed:
                        post_list.append(post)

        if posts:
            for i, post in enumerate(posts):
                if post:
                    post_list.append(post)

        return post_list

    # HOME
    def load_home(self):
        posts = self.fetch_posts()
        post_list = []

        if posts:
            for i, post in enumerate(posts):
                if post['post_id'] not in self.posts_displayed:

                    exists = False
                    for existing_post in post_list:
                        if existing_post['post_id'] == post['post_id']:
                            exists = True

                    if not exists:
                        post_list.append(post)
```

```
            posts = post_list

            for i, post in enumerate(posts):
                if not self.post_exist:
                    self.home_swiper_grid.clear_widgets()
                    self.post_exist = True
                home_swiper = HomeSwiper(self, post['post_id'])

                if i == 0:
                    first_home_swiper = home_swiper

                # adds your own post to the top of the post list
                # its done in this way below because there is no way to pre-pend
with kivy widgets
                # the only way is to manually modify the child list which is not
recomended
                if home_swiper.username == self.username:
                    # saves the previous post list
                    old_grid = self.home_swiper_grid.children[1:]
                    # clears the grid
                    self.home_swiper_grid.clear_widgets()
                    # adds your new post at the top
                    self.home_swiper_grid.add_widget(home_swiper)

                    # adds the rest of the previous posts
                    for old_post in old_grid:
                        self.home_swiper_grid.add_widget(old_post)
                    first_home_swiper = home_swiper
                else:
                    self.home_swiper_grid.add_widget(home_swiper)

                self.posts_displayed.append(post['post_id'])
                if i == 4:
                    break

            if "first_home_swiper" in locals():
                self.home_swiper_scroll.scroll_to(first_home_swiper)

            if not posts:
                Snackbar(text="Sorry, no more posts").open()

        else:
            if not self.post_exist:
                self.home_swiper_grid.clear_widgets()
                self.home_swiper_grid.add_widget(NoPostLabel())
```

```
        self.load_more_button = HomeLoadButton(home_obj=self)
        self.home_swiper_grid.add_widget(self.load_more_button)

    def get_home_swiper_height(self):
        swiper_height = Window.height * 0.70
        return swiper_height

    # MEMORIES
    def get_memories_swiper_height(self):
        month_height = self.get_memories_month_height()
        swiper_height = month_height * 0.8 * 0.02
        return swiper_height

    def get_memories_month_height(self):
        month_height = Window.height * 2
        return month_height

    def load_memories(self):
        self.root_scroll.clear_widgets()
        item = MonthList(self)
        self.root_scroll.add_widget(item)

    # STATS
    def load_stats(self):
        pass

    # ORGANISATION
    def load_organisation(self):
        if not self.organisation_item_exists:
            self.organisation_item_exists = True
            self.bottom_navigation.add_widget(OrganisationBottomItem(self,
self.username))

    # SIZE
    def update(self):
        self.home_swiper_grid.row_default_height = self.get_home_swiper_height()

    def on_size(self, *args):
        self.update()

    # Post time
    def posted_today(self):
        date = {'items': ['date']}
        posts = request(sio, session).emit("post_get_memories", date)['posts']
        if posts:
            date = request(sio, session).emit("get_date")['date']
            for post in posts:
```

```
                if date == post['date']:
                    self.post_made = True
                    return
        self.post_made = False

    def check_post_time(self, dt):
        now = timestamp().now
        if self.post_slot['post_slot_start'] < now and
self.post_slot['post_slot_end'] > now and not self.post_made:
            if not self.camera_widget_exists:
                self.toolbar.right_action_items.append(["camera", lambda x:
self.switch_to_camera(self.app, direction='up')])
                self.camera_widget_exists = True
        else:
            for i, action_item in enumerate(self.toolbar.right_action_items):
                if "camera" in action_item:
                    self.toolbar.right_action_items.pop(i)
                    break

    def switch_to_camera(self, app, direction="up"):
        camera_page_screen_name = "CameraPageScreen"
        self.camera_page_screen = CameraPage(self, name=camera_page_screen_name)
        app.sm.add_widget(self.camera_page_screen)
        app.set_screen(camera_page_screen_name, direction)

    # First time
    def check_first_time(self, app):
        occupation = request(sio, session).emit("occupation_get")
        profile = request(sio, session).emit("profile_get", {'items': ['role',
'name']})
        friends = request(sio, session).emit("friend_get")['friends']
        if not occupation['occupation_id'] and not friends and not profile['role']
and not profile['name']:
            first_time_page_screen_name = "FirstTimePage"
            first_time_page_screen =
FirstTimePage(name=first_time_page_screen_name)
            app.sm.add_widget(first_time_page_screen)
            app.set_screen(first_time_page_screen_name, "down")
# HomePage END


# Comments START
class CommentContainer(IRightBodyTouch, MDBoxLayout):
    pass

class Comment(TwoLineAvatarIconListItem):
    def __init__(self, username, comment_id, page, **kwargs):
```

```
        super().__init__(**kwargs)
        self.username = username
        self.comment_id = comment_id
        self.page = page
        self.screen_prefix = "comment"
        self.action_menu = None

        self.load_content()

    def liked_previously(self):
        data = {'comment_id': self.comment_id}
        impressions = request(sio, session).emit("comment_impression_get_comment",
data)['impressions']

        if impressions:
            for impression in impressions:
                if dict_key_verify(impression, "username"):
                    if session.username == impression['username']:
                        self.impression_id = impression['impression_id']
                        return True
        return False

    def load_content(self):
        if self.liked_previously():
            self.like_button.icon = "heart"

        data = {'impression_type': "like", 'comment_id': self.comment_id}
        count = request(sio, session).emit("comment_impression_count", data)['im-
pression_count']
        self.like_count.text = str(count)

    def expand(self, app):
        expand_page_name = f"{self.screen_prefix}_expand_page_{self.username}"
        expand_page = ExpandPage(self.secondary_text, self.text+"'s comment",
self.page, name=expand_page_name)
        app.sm.add_widget(expand_page)
        app.set_screen(expand_page_name, "left")

    def like(self):
        if self.like_button.icon == "heart-outline":
            data = {'impression_type': "like", 'comment_id': self.comment_id}
            request(sio, session).emit('comment_impression_set', data, None)
            if self.liked_previously():
                self.like_button.icon = "heart"
                self.like_count.text = str(int(self.like_count.text)+1)
        else:
            self.like_button.icon = "heart-outline"
```

```
            data = {'impression_id': self.impression_id}
            request(sio, session).emit("comment_impression_delete", data, None)
            self.like_count.text = str(int(self.like_count.text)-1)

    def profile(self, app):
        if self.action_menu:
            self.action_menu.dismiss()
        account_page_name = f"{self.screen_prefix}_account_page_{self.username}"
        account_page = AccountPage(self.username, self.page, remove_on_exit=True,
name=account_page_name)
        app.sm.add_widget(account_page)
        app.set_screen(account_page_name, "right")

    def delete_comment(self):
        if self.action_menu:
            self.action_menu.dismiss()
        data = {'comment_id': self.comment_id}
        request(sio, session).emit("comment_delete", data, None)
        self.page.comment_stack.remove_widget(self)
        if not self.page.comment_stack.children:
            self.page.load_content()

    def action_options(self, app):
        data = {'comment_id': self.comment_id}
        delete_allowed = request(sio, session).emit("comment_get_permissions",
data)['delete']

        if delete_allowed:
            profile_item = {'text': "view profile", 'viewclass': "OneLineListItem",
'on_release': lambda x=app: self.profile(app)}
            delete_item = {'text': "delete comment", 'viewclass': "OneLineLis-
tItem", 'on_release': lambda: self.delete_comment()}
            items = [profile_item, delete_item]
            self.action_menu = MDDropdownMenu(caller=self.profile_button,
items=items, width_mult=3)
            self.action_menu.open()
        else:
            self.profile(app)

class CommentsPage(MDScreen):
    def __init__(self, post_id, **kwargs):
        super(CommentsPage, self).__init__(**kwargs)
        self.post_id = post_id
        self.comments = []
        self.comments_exist = False
        self.load_content()
```

```
    def get_comments(self):
        data = {'post_id': self.post_id}
        comments = request(sio, session).emit("comment_get_post", data)['comments']
        return comments

    def add_comment(self, comment):
        if not self.comments:
            self.comment_stack.clear_widgets()

        comment_id = comment['comment_id']
        username = comment['username']

        comment_item = Comment(username, comment_id, self, text=username, second-
ary_text=comment['content'])
        self.comments.append(comment_id)
        self.comment_stack.add_widget(comment_item)

    # LOADING
    def load_content(self):
        self.comment_stack.clear_widgets()
        comments = self.get_comments()

        if comments:
            for comment in comments:
                self.add_comment(comment)
        else:
            item = OneLineAvatarIconListItem(text="No comments :(")
            self.comment_stack.add_widget(item)

    def submit(self):
        content = self.comment_field.text
        data = {'post_id': self.post_id, 'content': content}
        request(sio, session).emit("comment_set", data, None)
        self.comment_field.text = ""
        comments = self.get_comments()
        for comment in comments:
            if comment['username'] == session.username and comment['content'] ==
content and comment['comment_id'] not in self.comments:
                self.add_comment(comment)
                break

    # SWITCHING
    def switch_to_home(self, app):
        app.set_screen("HomePageScreen", 'down')
        app.sm.remove_widget(self)
# Comments END
```

```
# Post START
class CameraPage(MDScreen):
    def __init__(self, previous_page, **kwargs):
        super(CameraPage, self).__init__(**kwargs)
        self.path = "data/images/post.png"
        self.post_slot = request(sio, session).emit("post_slot_get")
        self.previous_page = previous_page
        Window.size = (800, 600)

        self.load_content()
        self.refresh_time()
        Clock.schedule_interval(self.refresh_time, 1)

    def load_content(self):
        self.camera = Camera(play=True)
        self.camera_area.add_widget(self.camera)

        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)

    def open_help(self, app):
        open_help(app, self, "Camera")

    def format_time(self, time_left):
        time_left = int(time_left)
        seconds = time_left%60
        minutes = time_left//60
        hours = 0
        if minutes > 60:
            hours = minutes//60
            minutes = minutes - hours*60
        time_format = f"{hours}:{minutes}:{seconds}"
        return time_format

    def refresh_time(self, dt=None):
        length = self.post_slot['post_slot_end'] -
self.post_slot['post_slot_start']
        time_in = timestamp().now - self.post_slot['post_slot_start']
        time_left = round(length - time_in, 2)
        time_format = self.format_time(time_left)
        self.toolbar.title = f"Time left: {time_format}"

    def capture(self, app):
        self.camera.export_to_png(self.path)
        self.camera_to_post(app)
```

```
    def camera_to_post(self, app):
        post_review_page_screen_name = "PostReviewPage"
        post_review_page_screen = PostReviewPage(self, self.path, name=post_re-
view_page_screen_name)
        app.sm.add_widget(post_review_page_screen)
        app.set_screen(post_review_page_screen_name, "left")

    def exit(self, app):
        app.set_screen("HomePageScreen", "down")
        app.sm.remove_widget(self)
        Window.size = (800, 1000)

class PostReviewPage(MDScreen):
    def __init__(self, camera_page, path, **kwargs):
        super(PostReviewPage, self).__init__(**kwargs)
        self.path = path
        self.camera_page = camera_page
        self.post_slot = request(sio, session).emit("post_slot_get")

        self.load_content()
        self.refresh_time()
        Clock.schedule_interval(self.refresh_time, 1)

    def load_content(self):
        self.image.source = self.path

        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)

    def open_help(self, app):
        open_help(app, self, "PostReview")

    def refresh_time(self, dt=None):
        length = self.post_slot['post_slot_end'] -
self.post_slot['post_slot_start']
        time_in = timestamp().now - self.post_slot['post_slot_start']
        time_left = round(length - time_in, 2)
        time_format = self.camera_page.format_time(time_left)
        self.toolbar.title = f"Time left: {time_format}"

    def post(self, app):
        with open(self.path, "rb") as image:
            image_data = image.read()
            data = {'content': image_data, 'caption': self.caption.text}
```

```
            request(sio, session).emit("post_set", data, None)
            self.camera_page.previous_page.post_made = True
            self.image.source = ""
            os.remove(self.path)
        self.exit(app, "down")

        self.camera_page.previous_page.load_memories()
        self.camera_page.previous_page.load_home()

    def retake(self, app):
        os.remove(self.path)
        self.post_to_camera(app)

    def post_to_camera(self, app, direction="right"):
        app.set_screen(self.camera_page.name, direction)
        self.ids.image_area.remove_widget(self.image)
        app.sm.remove_widget(self)

    def exit(self, app, direction="down"):
        app.set_screen("HomePageScreen", "down")
        if not self.camera_page.previous_page.post_made:
            os.remove(self.path)
        app.sm.remove_widget(self.camera_page)
        app.sm.remove_widget(self)
        Window.size = (800, 1000)
# Post END

# Acccount START
class ProfileInfo(TwoLineAvatarIconListItem):
    def __init__(self, account_page=None, info_type=None, **kwargs):
        super().__init__(**kwargs)
        self.info_type = info_type
        self.account_page = account_page

    def set_title(self, text):
        if text and type(text) == str:
            text = (text.replace("_", " ")).capitalize()
            self.text = text

    def set_content(self, text):
        if text and type(text) == str:
            self.secondary_text = text

    def make_editable(self):
        button = InfoEditButton(self.account_page, self.info_type)
        self.add_widget(button)
```

```
class InfoEditButton(IconRightWidget):
    def __init__(self, account_page_obj=None, info_type=None, **kwargs):
        super().__init__(**kwargs)
        self.info_type = info_type
        self.account_page_obj = account_page_obj
        self.account_page_obj.currently_editing = None

    def change_info(self):
        if self.icon == "pencil":
            if self.account_page_obj.currently_editing:
                self.account_page_obj.currently_editing.icon = "pencil"
            self.account_page_obj.currently_editing = self
            self.account_page_obj.picture_to_textbox(self.info_type)
        else:
            self.account_page_obj.textbox_to_picture()

class BioEditButton(MDIconButton):
    def __init__(self, account_page_obj=None, info_type="biography", **kwargs):
        super().__init__(**kwargs)
        self.info_type = info_type
        self.account_page_obj = account_page_obj
        self.account_page_obj.currently_editing = None

    def change_info(self):
        if self.icon == "pencil":
            if self.account_page_obj.currently_editing:
                self.account_page_obj.currently_editing.icon = "pencil"
            self.account_page_obj.currently_editing = self
            self.account_page_obj.picture_to_textbox(self.info_type)
        else:
            self.account_page_obj.textbox_to_picture()

class ProfilePicture(FitImage):
    pass

class InfoChangeBox(MDBoxLayout):
    def __init__(self, page, info_type, username=session.username, **kwargs):
        super().__init__(**kwargs)
        self.username = username
        self.info_type = info_type
        self.page = page

        self.load_content()

    def load_content(self):
        self.text_field.text = self.page.info[self.info_type]
```

```
    def save(self):
        new_value = self.text_field.text
        self.page.info[self.info_type] = new_value
        account_page(sio).set_profile(self.info_type, new_value, self.username)
        self.page.textbox_to_picture()

class OccupationChange(MDBoxLayout):
    def __init__(self, page=None, **kwargs):
        super().__init__(**kwargs)
        self.occupations = request(sio, session).emit('occupation_get_all')
        self.current_selection = None
        self.occupations_info = []
        self.menu = None
        if page:
            self.page = page

    def selection_menu(self):
        if dict_key_verify(self.occupations, 'occupations'):
            self.occupations_info = self.occupations['occupations']
            items = []
            for i, occupation in enumerate(self.occupations_info):
                item = {'text': occupation['name'], 'viewclass': "OneLineListItem",
'on_release': lambda x=i: self.selection(x)}
                items.append(item)
            self.menu = MDDropdownMenu(caller=self.occupation_select, items=items,
width_mult=2)
            self.menu.open()
        else:
            message = "No occupations"
            Snackbar(text=message).open()

    def selection(self, item_num):
        self.occupation_select.text = self.occupations_info[item_num]['name']
        self.occupation_description.text = self.occupations_info[item_num]['de-
scription']
        self.current_selection = item_num
        self.menu.dismiss()


class UserOccupationChange(OccupationChange):
    def __init__(self, page=None, **kwargs):
        super().__init__(page, **kwargs)
        self.load_content()

    def cancel(self):
        request(sio, session).emit('occupation_delete_request', {}, None)
        message = f"{session.status['level']}: {session.status['message']}"
```

```
        Snackbar(text=message).open()
        self.load_content()
        self.page.textbox_to_picture()

    def submit(self):
        if self.current_selection or self.current_selection == 0:
            data = {'occupation_id': self.occupations_info[self.current_selec-
tion]['occupation_id']}
            request(sio, session).emit('occupation_set_request', data, None)
        else:
            message = "No selection made"
            Snackbar(text=message).open()
        self.load_content()
        self.occupation_select.text = "Select an occupation"
        self.occupation_description.text = ""
        self.page.textbox_to_picture()

    def load_content(self):
        occupation_request = request(sio, session).emit('occupation_get_request')
        if dict_key_verify(occupation_request, 'occupation_id'):
            occupation = request(sio, session).emit('occupation_get', {'occupa-
tion_id': occupation_request['occupation_id']})

            if occupation:
                self.request_occupation.text = occupation['name']

                if occupation_request['approved']:
                    request_status = "Approved"
                else:
                    request_status = "Pending"

                self.request_status.text = "Status: "+request_status
            else:
                Snackbar(text=session.status).open()

class ManagementOccupationChange(OccupationChange):
    def __init__(self, page, **kwargs):
        super().__init__(**kwargs)
        self.page = page

    def submit(self):
        if self.occupations_info:
            data = {'occupation_id': self.occupations_info[self.current_selec-
tion]['occupation_id']}
            request(sio, session).emit('occupation_set', data, None)
            message = f"{session.status['message']}"
            self.occupation_select.text = "Select an occupation"
```

```
            self.occupation_description.text = ""
        else:
            message = f"No occupation selected"
        Snackbar(text=message).open()

        if self.occupations_info:
            self.page.textbox_to_picture()

class AccountPage(MDScreen):
    def __init__(self, username=session.username, previous_page=None, re-
move_on_exit=True, **kwargs):
        super(AccountPage, self).__init__(**kwargs)
        self.username = username
        self.info_objs = []
        self.bio_edit_button = None
        self.load_content()
        self.friend_page_screen = None
        self.remove_on_exit = remove_on_exit
        if previous_page == None:
            self.previous_page = "HomePageScreen"
        else:
            self.previous_page = previous_page.name

    def load_content(self):
        self.above_info.clear_widgets()
        self.profile_info_view.clear_widgets()
        if self.bio_edit_button:
            self.profile_bio_view.remove_widget(self.bio_edit_button)

        req = request(sio, session)
        data = {'username': self.username}
        self.info = req.emit("profile_get", data)

        permissions_data = {'username': session.username, 'target_username':
self.username}
        self.permissions = req.emit("profile_get_permissions", permissions_data)

        if self.username != session.username:
            self.toolbar.title = self.username+"'s Profile"

            level = req.emit('auth_get')['level']
            if level == "member":
                self.toolbar.left_action_items = []

        if dict_key_verify(self.info, "occupation_id"):
            team_data = {'occupation_id': self.info['occupation_id'], 'items':
["name"]}
```

```
            self.info['team'] = req.emit("team_get", team_data)['name']

            occupation_data = {'occupation_id': self.info['occupation_id'],
'items': ["name"]}
            self.info['occupation'] = req.emit("occupation_get", occupa-
tion_data)['name']
        else:
            self.info['team'] = ""
            self.info['occupation'] = ""
        del self.info['occupation_id']

        self.info['username'] = self.username
        order = ['username', 'name', 'role', 'occupation', 'team']

        self.bio_edit_button = BioEditButton(self)
        if self.info['biography']:
            self.biography_content.text = self.info['biography']
        else:
            self.biography_content.text = ""
            self.info['biography'] = ""
        if self.permissions['edit']:
            self.profile_bio_view.add_widget(self.bio_edit_button)

        for key in order:
            if not self.info[key]:
                self.info[key] = ""
            profile_info = ProfileInfo(self, key)
            if self.permissions['edit'] and key in ['name', 'role', 'occupation',
'team']:
                profile_info.make_editable()
            profile_info.set_title(key)
            profile_info.set_content(self.info[key])
            self.profile_info_view.add_widget(profile_info)
            self.info_objs.append(profile_info)

        self.profile_picture = ProfilePicture()
        self.above_info.add_widget(self.profile_picture)

        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)

    def open_help(self, app):
        open_help(app, self, "Profile")

    def refresh_content(self):
```

```
        self.load_content()

    def picture_to_textbox(self, info_type):
        self.currently_editing.icon = "close"
        self.above_info.clear_widgets()
        level = request(sio, session).emit('auth_get')['level']
        if info_type == 'occupation' or info_type == 'team':
            if level == "member":
                self.change = UserOccupationChange(self)
            else:
                self.change = ManagementOccupationChange(self)
        else:
            self.change = InfoChangeBox(self, info_type, self.username)
        self.above_info.add_widget(self.change)

    def textbox_to_picture(self):
        self.above_info.clear_widgets()
        self.above_info.add_widget(self.profile_picture)
        self.currently_editing.icon = "pencil"
        self.refresh_content()

    def switch_to_friend(self, app, direction="right"):
        friend_page_screen_name = "FriendPageScreen_" + self.username
        friend_page_screen = FriendPage(self, self.username,
name=friend_page_screen_name)
        app.sm.add_widget(friend_page_screen)
        app.set_screen(friend_page_screen_name, direction)

    def back(self, app, direction="left"):
        app.set_screen(self.previous_page, direction)
        if self.remove_on_exit:
            app.sm.remove_widget(self)
# Acccount END


# Settings START
class SettingRoot(MDBoxLayout):
    def __init__(self, title, page, **kwargs):
        super().__init__(**kwargs)
        self.title = title
        self.page = page
        self.load_content()

    def load_content(self):
        self.setting = setting_info(self.title)
        self.set_title(self.setting.title)
        self.set_description(self.setting.description)
```

```
        self.setting_icon.icon = self.setting.icon

    def set_title(self, text):
        if text and type(text) == str:
            self.setting_title.text = text

    def set_description(self, text):
        if text and type(text) == str:
            self.setting_description.text = text

class SettingSwitch(SettingRoot):
    def load_content(self):
        super().load_content()
        self.toggle.active = self.setting.value

    def on_toggle(self, app):
        self.page.help_tool = ["help", lambda  app: self.page.open_help(app)]
        self.setting.change_value(self.toggle.active)
        self.page.load_toolbar()
        app.homepage_screen.load_toolbar()

class SettingTextField(SettingRoot):
    def __init__(self, title, **kwargs):
        super().__init__(title, **kwargs)
        self.submission_func = self.submit_url

    def load_content(self):
        super().load_content()
        self.input_field.text = self.setting.value

    def submit_func(self):
        self.setting.change_value(self.input_field.text)
        self.submission_func()
        self.page.load_toolbar()
        app.homepage_screen.load_toolbar()

    def submit_url(self):
        self.error = True

        try:
            result = urlopen(self.text)
        except HTTPError as e:
            pass
        except URLError as e:
            pass
        except ValueError as e:
            pass
```

```
        else:
            self.error = False

class ShutdownButton(MDRaisedButton):
    def __init__(self, page, **kwargs):
        super().__init__(**kwargs)
        self.page = page

    def shutdown(self, app):
        request(sio, session).emit("shutdown", None, None)
        #app.disconnected()

class SettingsPage(MDScreen):
    def __init__(self, previous_page, **kwargs):
        super(SettingsPage, self).__init__(**kwargs)
        self.previous_page = previous_page

        self.load_content()

    def load_content(self):
        self.settings_stack.clear_widgets()
        settings = db().execute("SELECT title, state FROM settings")
        if settings:
            for setting in settings:
                if setting[1] != None:
                    setting_obj = SettingSwitch(setting[0], self)
                else:
                    setting_obj = SettingTextField(setting[0], self)

            self.settings_stack.add_widget(setting_obj)

        self.level = request(sio, session).emit('auth_get')['level']
        if self.level == "admin":
            button = ShutdownButton(self)
            self.static_buttons.add_widget(button)

        self.load_toolbar()

    def load_toolbar(self):
        if not self.toolbar.right_action_items:
            self.toolbar.right_action_items = [self.help_tool]
        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)
```

```
    def open_help(self, app):
        open_help(app, self, "Settings")

    def logout(self, app):
        clean_directories()
        session.clear()
        app.switch_to_login()

    def back(self, app):
        app.sm.remove_widget(self)
        self.previous_page.load_content()
        app.set_screen(self.previous_page.name, "right")
# Settings END


# Notification START
class NotificationItem(TwoLineAvatarIconListItem):
    def __init__(self, page, notification_id, username, **kwargs):
        super().__init__(**kwargs)
        self.notification_id = notification_id
        self.username = username
        self.page = page
        self.content = ""

    def set_title(self, text):
        if text and type(text) == str:
            self.text = text
            self.title = text

    def set_content(self, text):
        if text and type(text) == str:
            self.secondary_text = text
            self.content = text

    def delete(self):
        data = {'notification_id': self.notification_id, 'username': self.username}
        request(sio, session).emit('notification_remove', data, None)
        self.page.notification_stack.remove_widget(self)
        self.page.load_content()

    def expand(self, app):
        expand_page = ExpandPage([self.title, self.content], app)
        app.sm.add_widget(expand_page)

class NotificationsPage(MDScreen):
    def __init__(self, username=None, previous_page=None, **kwargs):
        super(NotificationsPage, self).__init__(**kwargs)
```

```
        session.notification_page = self
        self.username = username
        self.previous_page = previous_page
        self.notifications = None
        self.load_content()

    def load_content(self):
        self.notifications = request(sio, session).emit('notification_get',
{'username': self.username})['notifications']
        self.notification_stack.clear_widgets()

        if self.notifications:
            for notification in self.notifications:
                notification_obj = NotificationItem(self, notification['notifica-
tion_id'], self.username)

                if dict_key_verify(notification, 'title'):
                    notification_obj.set_title(notification['title'])
                    if dict_key_verify(notification, 'content'):
                        content_text = notification['content']
                        notification_obj.set_content(content_text)

                self.notification_stack.add_widget(notification_obj)
        else:
            item = OneLineAvatarIconListItem(text="No notifications")
            self.notification_stack.add_widget(item)

        if self.username and self.username != session.username:
            self.toolbar.title = self.username+"'s Notifications"

        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)

    def open_help(self, app):
        open_help(app, self, "Notifications")

    def add_notification(self, notification):
        notification_item = NotificationItem(self, notification['notification_id'],
self.username)
        content_text = notification['title']
        if dict_key_verify(notification, 'content'):
            notification_item.set_content(notification['content'])

        self.notification_stack.add_widget(notification_item)
```

```
    def back(self, app, direction="left"):
        app.set_screen(self.previous_page.name, direction)
        app.sm.remove_widget(self)
# Notification END


# Friend START
class BaseFriendItem(OneLineAvatarIconListItem):
    def __init__(self, obj=None, **kwargs):
        super().__init__(**kwargs)
        self.page_obj = obj
        self.friend_profile_screen_name = None
        self.friend_profile_screen = None

    def refresh_content(self):
        if self.page_obj:
            self.page_obj.load_content()

    def profile(self, app):
        new_friend_profile_screen_name = self.profile_prefix+"ProfileP-
age"+self.text
        if new_friend_profile_screen_name != self.friend_profile_screen_name:
            if self.friend_profile_screen_name:
                app.sm.remove_widget(self.friend_profile_screen)
        self.friend_profile_screen_name = new_friend_profile_screen_name
        self.friend_profile_screen = AccountPage(self.text, self.page_obj,
name=self.friend_profile_screen_name)
        app.sm.add_widget(self.friend_profile_screen)
        app.set_screen(self.friend_profile_screen_name, "right")

class IncomingRequestItem(BaseFriendItem):
    def __init__(self, obj=None, **kwargs):
        super().__init__(obj, **kwargs)
        self.profile_prefix = "IncomingRequest"

    def accept(self):
        self.verdict('approve')

    def reject(self):
        self.verdict('reject')

    def verdict(self, verdict):
        data = {'friend_username': self.text}
        request(sio, session).emit('friend_'+verdict+'_request', data, None)
        self.refresh_content()
        self.page_obj.friend_page.load_content()
```

```
class OutgoingRequestItem(BaseFriendItem):
    def __init__(self, obj=None, **kwargs):
        super().__init__(obj, **kwargs)
        self.profile_prefix = "OutgoingRequest"

    def cancel(self):
        data = {'friend_username': self.text}
        request(sio, session).emit('friend_remove_request', data, None)
        self.refresh_content()

class RecomendationItem(BaseFriendItem):
    def __init__(self, obj=None, **kwargs):
        super().__init__(obj, **kwargs)
        self.profile_prefix = ""

    def add_friend(self):
        self.page_obj.add_friend(self.text)

class FriendItem(BaseFriendItem):
    def __init__(self, obj=None, **kwargs):
        super().__init__(obj, **kwargs)
        self.profile_prefix = "Friend"

    def remove(self):
        data = {'friend_username': self.text}
        request(sio, session).emit('friend_remove', data, None)
        self.refresh_content()

class FriendPage(MDScreen):
    def __init__(self, account_page, username, **kwargs):
        super(FriendPage, self).__init__(**kwargs)
        self.account_page = account_page
        self.username = username
        self.friend_request_screens = []
        self.load_content()

    def load_content(self):
        data = {'username': self.username}
        self.friends = request(sio, session).emit("friend_get")['friends']
        self.friend_list.clear_widgets()

        if self.friends:
            for friend in self.friends:
                friend_info = FriendItem(self, text=friend['username'])
                self.friend_list.add_widget(friend_info)
        else:
            friend_info = OneLineAvatarIconListItem(text="No friends")
```

```
            self.friend_list.add_widget(friend_info)

        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.left_action_items):
                if option[0].lower() == "help":
                    self.toolbar.left_action_items.pop(i)
        self.account_page.load_content()

    def open_help(self, app):
        open_help(app, self, "Friends")

    def switch_to_friend_request(self, app, username=None, direction='right'):
        if not username:
            username = self.username
        friend_request_screen_name = "FriendRequestPageScreen_"+username
        friend_request_screen = FriendRequestPage(self, username, name=friend_re-
quest_screen_name)
        app.sm.add_widget(friend_request_screen)

        app.set_screen(friend_request_screen_name, direction)

    def switch_to_account(self, app):
        app.set_screen(self.account_page.name, "left")
        app.sm.remove_widget(self)

class FriendRequestPage(MDScreen):
    def __init__(self, friend_page, username, **kwargs):
        super(FriendRequestPage, self).__init__(**kwargs)
        self.username = username
        self.friend_page = friend_page
        self.load_content()

    def _get_request(self, mode="incoming", username=None):
        requests_data = request(sio, session).emit("friend_get_requests",
{'username': username, 'mode': mode})
        if dict_key_verify(requests_data, "requests"):
            requests = requests_data['requests']
        else:
            requests = []
        return requests

    def load_content(self):
        self.incoming_requests.clear_widgets()
        self.outgoing_requests.clear_widgets()
        self.recomendations.clear_widgets()

        self.incoming = self._get_request("incoming", self.username)
```

```
        if not self.incoming:
            self.incoming = []
            self.incoming_requests.add_widget(OneLineAvatarIconListItem(text="No
incoming requests"))

        self.outgoing = self._get_request("outgoing", self.username)
        if not self.outgoing:
            self.outgoing = []
            self.outgoing_requests.add_widget(OneLineAvatarIconListItem(text="No
outgoing requests"))

        for incoming in self.incoming:
            self.incoming_requests.add_widget(IncomingRequestItem(self, text=incom-
ing))
        for outgoing in self.outgoing:
            self.outgoing_requests.add_widget(OutgoingRequestItem(self, text=out-
going))

        data = {'amount': 5}
        self.friend_recomends = request(sio, session).emit('friend_get_recomenda-
tions', data)
        if dict_key_verify(self.friend_recomends, "recomended"):
            self.friend_recomends = self.friend_recomends["recomended"]
        else:
            self.friend_recomends = []
            self.recomendations.add_widget(OneLineAvatarIconListItem(text="No rec-
ommendations sorry"))

        for recomend in self.friend_recomends:
            self.recomendations.add_widget(RecomendationItem(self,
text=recomend['username']))

    def add_friend_search(self):
        message = "No username entered"
        if self.username_select.text:
            self.add_friend(self.username_select.text)
            if session.status['level'].lower() != "info":
                self.username_select.error = True
            else:
                self.username_select.text = ""
        else:
            self.username_select.error = True

    def add_friend(self, username):
        data = {'friend_username': username}
        request(sio, session).emit('friend_add_request', data, None)
        message = f"{session.status['level']}: {session.status['message']}"
```

```
        Snackbar(text=message).open()
        self.load_content()

    def switch_to_friend(self, app, username=None, direction='left'):
        app.set_screen(self.friend_page.name, direction)
        app.sm.remove_widget(self)
# Friend END


# OCCUPATION START
class ManageOccupationChange(OccupationChange):
    def submit(self):
        if self.current_selection:
            data = {'occupation_id': occupations[i]['occupation_id'], 'username':
self.username_select.text}
            request(sio, session).emit('occupation_set', data)
            message = f"{session.status['level']}: {session.status['message']}"
            Snackbar(text=message).open()
            if session.status['level'].lower() != "info":
                self.username_select.error = True
        else:
            message = "No selection made"
            Snackbar(text=message).open()

class BaseOccupationItem(ThreeLineAvatarIconListItem):
    def __init__(self, occupation_id, obj=None, **kwargs):
        super().__init__(**kwargs)
        self.occupation_id = occupation_id
        self.page_obj = obj
        data = {'occupation_id': self.occupation_id}

class OccupationItem(BaseOccupationItem):
    def edit(self):
        self.page_obj.occupation_edit(self.occupation_id, self.text, self.second-
ary_text)

    def delete(self):
        self.page_obj.occupation_delete(self.occupation_id)
        self.page_obj.load_content()

class OccupationRequestItem(BaseOccupationItem):
    def refresh_content(self):
        self.page_obj.load_content()

    def accept(self):
        self.verdict("approve")
```

```
    def reject(self):
        self.verdict("reject")

    def verdict(self, verdict):
        data = {'username': self.text}
        request(sio, session).emit('occupation_'+verdict+'_request', data, None)
        self.refresh_content()

class OccupationEdit(MDBoxLayout):
    def __init__(self, page, occupation_id, name, description, **kwargs):
        super().__init__(**kwargs)
        self.name = name
        self.description = description
        self.occupation_id = occupation_id
        self.page = page
        self.load_content()

    def load_content(self):
        self.ids.name.text = self.name
        self.ids.description.text = self.description

    def submit(self):
        data = {'name': self.ids.name.text, 'description': self.ids.descrip-
tion.text, 'occupation_id': self.occupation_id}
        request(sio, session).emit('occupation_edit', data, None)
        self.page.load_content()
        self.page.edit_area.clear_widgets()
        self.page.edit_area.add_widget(OccupationCreate(self))

class OccupationCreate(MDBoxLayout):
    def __init__(self, page, **kwargs):
        super().__init__(**kwargs)
        self.page = page

    def create(self):
        data = {'name': self.ids.name.text, 'description': self.ids.descrip-
tion.text}
        request(sio, session).emit('occupation_create', data, None)
        message = f"{session.status['level']}: {session.status['message']}"
        Snackbar(text=message).open()
        self.ids.name.text = ""
        self.ids.description.text = ""
        self.page.load_content()

class OccupationPage(MDScreen):
    def __init__(self, previous_page, **kwargs):
        super(OccupationPage, self).__init__(**kwargs)
```

```
        self.previous_page = previous_page
        self.load_content()

    def load_content(self):
        occupations = request(sio, session).emit('occupation_get_all')['occupa-
tions']
        self.occupations.clear_widgets()

        if occupations:
            for occupation in occupations:
                item = OccupationItem(occupation['occupation_id'], self, text=occu-
pation['name'], secondary_text=occupation['description'])
                self.occupations.add_widget(item)
        else:
            item = OneLineAvatarIconListItem(text="No occupations")
            self.occupations.add_widget(item)

        self.edit_area.clear_widgets()
        self.edit_area.add_widget(OccupationCreate(self))

        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)
        self.previous_page.load_toolbar()

    def open_help(self, app):
        open_help(app, self, "Occupation")

    def occupation_edit(self, occupation_id, name, description):
        edit_space = OccupationEdit(self, occupation_id, name, description)
        self.edit_area.clear_widgets()
        self.edit_area.add_widget(edit_space)

    def occupation_delete(self, occupation_id):
        data = {'occupation_id': occupation_id}
        request(sio, session).emit('occupation_delete_occupation', data, None)
        Snackbar(text="Occupation deleted").open()

    def switch_to_occupation_request(self, app, direction='left'):
        occupation_request_screen_name = "OccupationRequestPageScreen"
        occupation_request_screen = OccupationRequestPage(self, name=occupation_re-
quest_screen_name)
        app.sm.add_widget(occupation_request_screen)
        app.set_screen(occupation_request_screen_name, direction)

    def back(self, app, direction="right"):
```

```
        app.set_screen("HomePageScreen", direction)
        app.sm.remove_widget(self)

class OccupationRequestPage(MDScreen):
    def __init__(self, previous_page, **kwargs):
        super(OccupationRequestPage, self).__init__(**kwargs)
        self.previous_page = previous_page
        self.load_content()

    def load_content(self):
        requests = request(sio, session).emit('occupation_get_all_requests')['re-
quests']
        self.change_requests.clear_widgets()

        if requests:
            for request_info in requests:
                data = {'occupation_id': request_info['occupation_id']}
                occupation = request(sio, session).emit('occupation_get', data)
                occupation_request = OccupationRequestItem(request_info['occupa-
tion_id'], self, text=request_info['username'], secondary_text = occupa-
tion['name'], tertiary_text=occupation['description'])
                self.change_requests.add_widget(occupation_request)
        else:
            self.change_requests.add_widget(OneLineAvatarIconListItem(text="No re-
quests"))

    def back(self, app, direction='right'):
        app.set_screen(self.previous_page.name, direction)
        app.sm.remove_widget(self)
# OCCUPATION END


# TEAM START
class LeaderItem(OneLineAvatarIconListItem):
    def __init__(self, leader_username, page=None, **kwargs):
        super().__init__(**kwargs)
        self.page = page
        self.leader_username = leader_username
        self.text = leader_username

    def delete(self):
        data = {'leaders': [{'username': self.leader_username}]}
        request(sio, session).emit('team_delete_leaders', data, None)
        self.page.load_content()

class AddLeaderButton(MDRaisedButton):
    def __init__(self, page=None, **kwargs):
```

```
        super().__init__(**kwargs)
        self.page = page

    def add_leader(self):
        self.page.button_to_add()

class ChangeNameButton(MDRaisedButton):
    def __init__(self, page=None, **kwargs):
        super().__init__(**kwargs)
        self.page = page

    def change_name(self):
        self.page.button_to_add("name")

class AddLeader(MDBoxLayout):
    def __init__(self, page=None, **kwargs):
        super().__init__(**kwargs)
        self.page = page

    def submit(self):
        data = {'leaders': [{'username': self.ids.username.text}]}
        request(sio, session).emit('team_set', data, None)
        self.page.add_to_button()

class ChangeName(MDBoxLayout):
    def __init__(self, page=None, **kwargs):
        super().__init__(**kwargs)
        self.page = page

    def submit(self):
        team_data = {'name': self.ids.name.text}
        request(sio, session).emit('team_set', team_data, None)
        self.page.add_to_button()

class TeamPage(MDScreen):
    def __init__(self, previous_page, username, **kwargs):
        super(TeamPage, self).__init__(**kwargs)
        self.previous_page = previous_page
        self.username = username
        self.load_content()

    def load_content(self):
        server = request(sio, session)
        leaders = []
        members = []

        leaders_info = server.emit('team_get_leaders')
```

```
        if dict_key_verify(leaders_info, 'leaders'):
            leaders = leaders_info['leaders']

        members_info = server.emit('team_get_members')
        if dict_key_verify(members_info, 'members'):
            members = members_info['members']
        team = server.emit('team_get')

        self.members.clear_widgets()
        self.leaders.clear_widgets()
        self.edit_area.clear_widgets()

        if leaders:
            for leader in leaders:
                item = LeaderItem(leader['username'], self)
                self.leaders.add_widget(item)
        else:
            item = OneLineAvatarIconListItem(text="No members")
            self.leaders.add_widget(item)

        self.level = server.emit('auth_get')['level']
        if self.level != "member" or self.username in leaders:
            if members:
                self.edit_area.add_widget(ChangeNameButton(self))
                self.edit_area.add_widget(AddLeaderButton(self))

        if members:
            self.team_name.text = team['name']
            for member in members:
                item = OneLineAvatarIconListItem(text=member['username'])
                self.members.add_widget(item)
            if not self.members:
                item = OneLineAvatarIconListItem(text="No members")
                self.members.add_widget(item)
        else:
            item = OneLineAvatarIconListItem(text="No members")
            self.members.add_widget(item)
        if self.team_name.text == "":
            self.team_name.text = "No team"

        if not setting_info("Help boxes").value:
            for i, option in enumerate(self.toolbar.right_action_items):
                if option[0].lower() == "help":
                    self.toolbar.right_action_items.pop(i)
        self.previous_page.load_toolbar()

    def button_to_add(self, input_mode="leader"):
```

```
        self.edit_area.clear_widgets()
        if input_mode == "leader":
            add_widget = AddLeader(self)
        elif input_mode == "name":
            add_widget = ChangeName(self)
        self.edit_area.add_widget(add_widget)

    def add_to_button(self):
        self.load_content()

    def open_help(self, app):
        open_help(app, self, "Team")

    def back(self, app, direction="right"):
        app.set_screen("HomePageScreen", direction)
        app.sm.remove_widget(self)
# TEAM END


# Auth START
class Auth():
    def load_content(self, obj, obj_id):
        for field in obj.fields:
            if "password" in field.lower():
                auth_field = PasswordField(info_type=field)
            else:
                auth_field = AuthField(info_type=field)
            obj_id.add_widget(auth_field)
            obj.auth_field_objs.append(auth_field)

class PasswordField(MDRelativeLayout):
    def __init__(self, info_type=None, **kwargs):
        super().__init__(**kwargs)
        self.info_type = info_type.lower()
        self.load_content()

    def load_content(self):
        self.ids.password_field.hint_text = self.info_type.capitalize()

class AuthField(MDTextField):
    def __init__(self, info_type=None, **kwargs):
        super().__init__(**kwargs)
        self.info_type = info_type.lower()
        self.load_content()

    def load_content(self):
        self.hint_text = self.info_type.capitalize()
```

```
    def change_hint(self, text):
        self.hint_text = text

class AuthButton(MDRaisedButton):
    def __init__(self, function=None, text=None, **kwargs):
        super().__init__(**kwargs)
        self.function = function
        self.text = text

    def action(self, app):
        self.function(app)

    def change_text(self, text):
        self.text = text

class LoginPage(MDScreen):
    def __init__(self, app, **kwargs):
        super(LoginPage, self).__init__(**kwargs)
        self.app = app
        self.fields = ["Username", "Password"]
        self.auth_field_objs = []
        self.register_page = None
        self.logged_in = False
        self.load_content()
        self.login_token()

    def load_content(self):
        Auth().load_content(self, self.login_view)
        self.login_view.add_widget(AuthButton(self.login, 'Login'))
        self.login_view.add_widget(AuthButton(self.register, 'Register'))

    def login_token(self):
        results = db().execute("SELECT * FROM tokens")
        data = {'logged_in': False}

        if results:
            for result in results:
                token = result[0]
                expire = float(result[2])
                if timestamp().now > float(expire):
                    db().execute("DELETE FROM tokens WHERE token = ?", (token,))
                else:
                    info = {'token': token}
                    data = request(sio, session).emit("login", info)

        if data['logged_in']:
```

```
            username = request(sio, session).emit("auth_get", {'items':
['username']})['username']
            self.login_confirmation(self.app, data, username)

    def login(self, app):
        username = self.auth_field_objs[0].text
        password = self.auth_field_objs[1].text
        if not username and not password:
            username = "user"
            password = "pass"
        info = {'username': username, 'password': password}
        data = request(sio, session).emit('login', info)
        self.login_confirmation(app, data, username)

    def login_confirmation(self, app, data, username):
        message = f"{session.status['level']}: {session.status['message']}"
        if data['logged_in'] == True:
            app.switch_to_homepage("down", username)
            session.username = username
            session.level = request(sio, session).emit('auth_get')['level']
            self.logged_in = True
        else:
            for field in self.auth_field_objs:
                field.error = True
                self.logged_in = False
        Snackbar(text=message).open()

    def register(self, app):
        register_page = RegisterPage(name='RegisterPageScreen')
        app.sm.add_widget(register_page)
        app.set_screen('RegisterPageScreen', 'left')

class RegisterPage(MDScreen):
    def __init__(self, **kwargs):
        super(RegisterPage, self).__init__(**kwargs)
        self.fields = ["Username", "Password", "Re-enter Password", "Registration
Code"]
        self.auth_field_objs = []
        self.load_content()
        self.mode = "member"

    def load_content(self):
        Auth().load_content(self, self.register_view)
        self.register_view.add_widget(AuthButton(self.register, 'Register'))
        self.mode_button = AuthButton(self.mode, 'Admin Register')
        self.register_view.add_widget(self.mode_button)
```

```
    def register(self, app):
        info_points = ['username', 'password', 'repassword', 'key']
        info = {point: self.auth_field_objs[i].text for i, point in enumer-
ate(info_points)}
        if self.mode == "member":
            event = "register"
        else:
            event = "admin_register"
        data = request(sio, session).emit(event, info)
        self.register_confirmation(data, app)

    def register_confirmation(self, data, app):
        message = f"{session.status['level']}: {session.status['message']}"
        if data['is_registered'] == True:
            app.set_screen("LoginPageScreen", "right")
            app.sm.remove_widget(self)
        else:
            Snackbar(text=message).open()
            for field in self.auth_field_objs:
                field.error = True
        Snackbar(text=message).open()

    def mode(self, app):
        for field in self.auth_field_objs:
            if "code" in field.info_type:
                break
        if self.mode == "member":
            self.mode_button.change_text("Member Register")
            field.hint_text = "Admin Registration Code"
            self.mode = "admin"
        else:
            self.mode_button.change_text("Admin Register")
            field.hint_text = "Registration Code"
            self.mode = "member"

class ServerPage(MDScreen):
    def __init__(self, **kwargs):
        super(ServerPage, self).__init__(**kwargs)

    def get_server_code(self):
        response = request(sio, session).emit("server_code_get")
        if dict_key_verify(response, 'server_code'):
            session.server_code = response['server_code']
        else:
            message = "WARNING: Badly behaving server, please contact administra-
tor"
            Snackbar(text=message).open()
```

```
    def connect(self, app):
        connected = start_client(sio, self.url.text)
        if not connected:
            self.url.error = True
            message = "Unsuccessful connection"
        else:
            self.get_server_code()
            app.switch_to_decrypt()
            message = "Successful connection"
        Snackbar(text=message).open()

class ShareInput(MDBoxLayout):
    pass

class DecryptPage(MDScreen):
    def __init__(self, app, **kwargs):
        super(DecryptPage, self).__init__(**kwargs)
        self.share_inputs = []
        self.min_shares = None
        self.app = app
        self.load_content()

    def load_content(self):
        mode_info = request(sio, session).emit("get_mode")
        self.sss_enabled = mode_info['sss']
        if self.sss_enabled:
            self.min_shares = int(mode_info['min_shares'])
            for i in range(self.min_shares):
                share_input = ShareInput()
                self.share_inputs.append(share_input)
                self.input_area.add_widget(share_input)

    def submit(self):
        encrypt_data = {'shares': None, 'password': None}
        data = {'success': False}
        if self.en_password.text:
            encrypt_data['password'] = self.en_password.text
            data = request(sio, session).emit("decrypt", encrypt_data)
        elif self.sss_enabled:
            encrypt_data['shares'] = []
            for share in self.share_inputs:
                try:
                    share_num = int(share.share_num.text)
                    share_secret = int(share.share_secret.text)
                except:
                    share.share_num.error = True
```

```
                    share.share_secret.error = True
                    continue

                encrypt_data['shares'].append({'num': share_num, 'secret':
share_secret})

            if len(encrypt_data['shares']) >= self.min_shares:
                data = request(sio, session).emit("decrypt", encrypt_data)
        if data['success']:
            self.app.sm.remove_widget(self)
            self.app.switch_to_login()
        else:
            self.en_password.error = True
            if encrypt_data['shares']:
                for share in self.share_inputs:
                    share.share_num.error = True
                    share.share_secret.error = True
# Auth END

class FirstTimePage(MDScreen):
    def __init__(self, **kwargs):
        super(FirstTimePage, self).__init__(**kwargs)
        self.load_content()

    def load_content(self):
        data = {'items': ['level']}
        level = request(sio, session).emit("auth_get", data)['level']
        if level == "member":
            occupation_change = UserOccupationChange(self)
            occupation_change.ids.new_request_area.remove_widget(occupa-
tion_change.ids.new_request_title)
        elif level == "management" or level == "admin":
            occupation_change = ManagementOccupationChange(self)
        self.ids.step3.add_widget(occupation_change)

        ntfy_topic = request(sio, session).emit("get_ntfy_topic")['topic']
        self.ids.topic_name.text = ntfy_topic

    def set_role(self):
        if self.role_input.text:
            request(sio, session).emit("profile_set", {'role': self.role_in-
put.text}, None)

    def set_name(self):
        if self.name_input.text:
            request(sio, session).emit("profile_set", {'name': self.name_in-
put.text}, None)
```

```
    def textbox_to_picture(self):
        pass

    def done(self, app):
        self.set_name()
        self.set_role()
        app.set_screen("HomePageScreen", "down")
        app.sm.remove_widget(self)

class BeOpen(MDApp):
    def set_screen(self, screen, trans):
        self.sm.current = screen
        self.sm.transition.direction = trans

    def build(self):
        Builder.load_file('./modules/ui/beopen.kv')

        self.theme_cls.material_style = "M3"
        self.theme_cls.theme_style = "Light"
        self.theme_cls.primary_palette = "Orange"

        self.homepage_screen = None
        self.login_screen = None

        self.sm = MDScreenManager()
        self.sm.add_widget(ServerPage(name='ServerPageScreen'))

        Window.size = (800, 1000)

        return self.sm

    def disconnected(self):
        self.sm.clear_widgets()
        stop_client(sio)
        self.sm.add_widget(ServerPage(name='ServerPageScreen'))

    def switch_to_comments(self, transition='up'):
        self.comments_screen = CommentsPage(name='CommentsPageScreen')
        self.sm.add_widget(self.comments_screen)
        self.set_screen('CommentsPageScreen', transition)

    def switch_to_homepage(self, transition=None, username=None):
        if self.homepage_screen:
            self.sm.remove_widget(self.homepage_screen)
        self.homepage_screen = HomePage(username, self, name='HomePageScreen')
        self.sm.add_widget(self.homepage_screen)
```

```
        if self.first_time_login():
            first_time_page_screen_name = "FirstTimePage"
            first_time_page_screen =
FirstTimePage(name=first_time_page_screen_name)
            self.sm.add_widget(first_time_page_screen)
            self.set_screen(first_time_page_screen_name, "down")
        else:
            self.sm.switch_to(self.homepage_screen)

    def switch_to_decrypt(self):
        crypt_data = request(sio, session).emit("get_mode")
        if crypt_data['mode'] == "decrypt":
            decrypt_screen_name = 'DecryptPageScreen'
            decrypt_screen = DecryptPage(self, name=decrypt_screen_name)
            self.sm.add_widget(decrypt_screen)
            self.set_screen(decrypt_screen_name, "down")
        else:
            self.switch_to_login()

    def switch_to_login(self, transition="up"):
        login_screen_name = "LoginPageScreen"
        if self.login_screen:
            self.sm.remove_widget(self.login_screen)

        self.login_screen = LoginPage(self, name=login_screen_name)
        if not self.login_screen.logged_in:
            self.sm.add_widget(self.login_screen)
            self.set_screen(login_screen_name, transition)

    def comments_to_homepage(self, transition='down'):
        self.set_screen('HomePageScreen', transition)
        self.sm.remove_widget(self.comments_screen)

    def first_time_login(self):
        occupation = request(sio, session).emit("occupation_get")
        profile = request(sio, session).emit("profile_get", {'items': ['role',
'name']})
        friends = request(sio, session).emit("friend_get")['friends']
        if not occupation['occupation_id'] and not friends and not profile['role']
and not profile['name']:
            return True
        else:
            return False
            first_time_page_screen_name = "FirstTimePage"
            first_time_page_screen =
FirstTimePage(name=first_time_page_screen_name)
```

```
            self.sm.add_widget(first_time_page_screen)
            self.set_screen(first_time_page_screen_name, "down")
#================== kivy END ==================

def create_settings():
        settings = [{'title': "Help boxes",
                     'description': "Turn of the help buttons that appear as clicka-
ble question marks",
                      'default_value': True,
                      'icon': "help"}]

        settings_db = db()
        saved_settings = settings_db.execute("SELECT title FROM settings")

        for setting in settings:
            if saved_settings:
                if (setting['title'],) in saved_settings:
                    continue

            if isinstance(setting['default_value'], bool):
                settings_db.execute("INSERT INTO settings (title, description,
state, icon) VALUES (?, ?, ?, ?)", (setting['title'], setting['description'], set-
ting['default_value'], setting['icon']))
            else:
                settings_db.execute("INSERT INTO settings (title, description,
value, icon) VALUES (?, ?, ?, ?)", (setting['title'], setting['description'], set-
ting['default_value'], setting['icon']))

def create_directories():
    paths = ["data", "data/images"]
    for path in paths:
        if not os.path.exists(path):
            os.mkdir(path)

def clean_directories():
    paths = ["data/images"]
    for path in paths:
        for file in os.listdir(path):
            os.remove(os.path.join(path, file))

def setup():
    create_directories()
    clean_directories()
    create_settings()

def post_login():
    db().execute("DELETE FROM tokens WHERE username != ?", (session.username,))
```

```
def main():
    setup()
    BeOpen().run()
    stop_client(sio)

if __name__ == "__main__":
    main()
```

modules/handler/info.py

```
from PIL import Image
import io

class profile():
    def __init__(self):
        pass

    def username():
        pass

    def name(self, first_name=None, last_name=None):
        if first_name:
            self.first_name
        if last_name:
            self.last_name

        if first_name and last_name:
            pass

class image():
    def __init__(self, post_id):
        self.post_id = post_id
        self.path = None

    def load(self, image_bytes):
        image_formats = ['png', 'jpg']
        for form in image_formats:
            try:
                self.path = f"data/images/{self.post_id}.{form}"
                with Image.open(io.BytesIO(image_bytes)) as recieved:
                    recieved.save(self.path)
                break
            except:
                self.path = None
```

```
    def delete(self):
        os.remove(self.path)
```

modules/handler/request.py

```
# MODULES
from modules.session.session import wait

def dict_key_verify(dictionary, keys, mode="and", *args, **kwargs):
    if mode != "and" and mode != "or":
        mode = "and"
    if type(keys) != list:
        keys = [keys]

    verified = []
    if type(keys) != list:
        keys = [keys]

    for key in keys:
        if type(dictionary) != dict or key not in dictionary or not diction-
ary[key]:
            verified.append(False)
        else:
            verified.append(True)

    if mode == "and":
        if all(verified) == True:
            return True
    if mode == "or":
        if True in verified:
            return True
    return False
# MODULES
class request():
    def __init__(self, sio, session=None, username=None):
        self.sio = sio
        self.session = session
        self.username = username

    def callback(self, callback, data):
        self.session.transfer = data

    def emit(self, event, info=None, callback_func="self.callback"):
        if callback_func == "self.callback":
            callback_func = self.callback
        if callback_func == None:
            self.sio.emit(event, info)
```

```
            return
        else:
            self.sio.emit(event, info, callback=callback_func)
        wait(self.session).wait()
        return self.session.transfer

class account_page(request):
    def refresh(self):
        info = self.get_profile()
        return info

    def get_profile(self):
        info = {'username': self.username, 'name': self.username, 'role': "", 'oc-
cupation_name': "", 'team_name': "", 'biography': ""}

        profile_data = {'username': self.username, 'items': ['name', 'role', 'biog-
raphy']}
        profile_info = self.emit('profile_get', profile_data)

        occupation_data = {'username': self.username, 'items': ['name']}
        occupation_info = self.emit('occupation_get', occupation_data)
        if dict_key_verify(occupation_info, 'name'):
            info['occupation_name'] = occupation_info['name']

        team_data = {'username': self.username, 'items': ['name']}
        team_info = self.emit('team_get', team_data)
        if dict_key_verify(team_info, 'name'):
            info['team_name'] = team_info['name']

        team_leader_info = self.emit('team_get_leaders')
        if dict_key_verify(team_leader_info, 'leaders'):
            if self.username in team_leader_info['leaders']:
                info['team_name'] += " (team lead)"

        for key in profile_info.keys():
            if dict_key_verify(profile_info, key):
                info[key] = profile_info[key]

        return info

    def set_profile(self, item, new_value, username=None):
        profile = ['name', 'role', 'biography']
        occupation = ['occupation_name']
        team = ['team_name']

        if item in profile:
            event = 'profile_set'
```

```
            if item == 'name':
                new_values = new_value.split(" ")
                if len(new_values) == 2:
                    items = ['first_name', 'last_name']
        if item in occupation:
            event = 'occupation_set'
            item = 'name'
        if item in team:
            event = 'team_set'
            item = 'name'

        if type(item) != list:
            items = [item]
        if type(new_value) != list:
            new_values  = [new_value]

        for value, item in zip(new_values, items):
            data = {'username': None,'items': [item], item: value}
            if username:
                data['username'] = username
            self.emit('profile_set', data, None)
```

modules/session/session.py

```
import time
import sqlite3

class session_info():
    def __init__(self):
        self._logged_in = False
        self.username = None
        self.server_code = None
        self.cycle_count = 0
        self.auth_tokens = []
        self.transfer = None
        self.status = None

    @property
    def logged_in(self):
        return self._logged_in
    @logged_in.setter
    def logged_in(self, value):
        self._logged_in = value
        self.cycle_count += 1
```

```
    @property
    def transfer(self):
        return self._transfer
    @transfer.setter
    def transfer(self, value):
        self._transfer = value
        self.cycle_count += 1

    def clear(self):
        db().execute("DELETE FROM tokens")
        self.__init__()

class wait():
    def __init__(self, session):
        self.session = session
        self.update()

    def update(self):
        self.last = self.session.cycle_count
        self.current = self.session.cycle_count

    def current_update(self):
        self.current = self.session.cycle_count

    def wait(self, status=None):
        while self.last == self.current:
            time.sleep(0.05)
            self.current_update()
        return

    def wait_username(self):
        while not self.session.username:
            time.sleep(0.05)
        return

class db():
    def __init__(self):
        self.path = "./data/database.db"
        self._create()

    def _create(self):
        con, cur = self._connect()

        cur.execute("""CREATE TABLE IF NOT EXISTS tokens (
                    token TEXT NOT NULL PRIMARY KEY,
                    username TEXT NOT NULL,
                    expire TEXT NOT NULL
```

```
                    )""")

        cur.execute("""CREATE TABLE IF NOT EXISTS settings (
                    title TEXT NOT NULL PRIMARY KEY,
                    description TEXT,
                    state BOOL,
                    value TEXT,
                    icon TEXT
                    )""")

        self._close(con)

    def _connect(self):
        con = sqlite3.connect(self.path)
        cur = con.cursor()
        return con, cur

    def execute(self, command, values=None):
        rez = None
        con, cur = self._connect()

        if values:
            cur.execute(command, values)
        else:
            cur.execute(command)

        if "SELECT" in command:
            rez = cur.fetchall()

        self._close(con)
        if rez:
            return rez

    def _commit(self, con):
        con.commit()

    def _close(self, con):
        con.commit()
        con.close()

class setting():
    def __init__(self, title):
        self.title = title
        self.db = db()
        self.__fetch()

    def __fetch(self):
```

```
        setting_data = self.db.execute("SELECT * FROM settings WHERE title = ?",
(self.title,))
        if setting_data:
            self.title = setting_data[0][0]
            self.description = setting_data[0][1]
            if not self.description:
                self.description = ""

            if setting_data[0][3] != None:
                self.value = setting_data[0][3]
                self.type = "text_field"
            else:
                self.value = setting_data[0][2]
                self.type = "swtich"

            if setting_data[0][4] != None:
                self.icon = setting_data[0][4]
            else:
                self.icon = "cog"

    def change_value(self, new_value):
        if self.type == "swtich" and isinstance(new_value, bool):
            self.db.execute("UPDATE settings SET state = ?", (new_value,))
        elif self.type == "text_field" and isinstance(new_value, str):
            self.db.execute("UPDATE settings SET value = ?", (new_value,))
```

modules/session/time.py

```
from datetime import date, timedelta, datetime

class timestamp():
    @property
    def start(self):
        value = self.get_date_timestamp()
        self._start = value
        return self._start
    @start.setter
    def start(self, value):
        value = self.get_date_timestamp()
        self._start = value

    @property
    def end(self):
        value = self.get_date_timestamp(day_mod=1) - 1
        self._end = value
```

```
        return self._end
    @end.setter
    def end(self, value):
        value = self.get_date_timestamp(day_mod=1) - 1
        self._end = value

    @property
    def now(self):
        value = self.get_timestamp()
        self._now = value
        return value
    @now.setter
    def now(self, value):
        value = self.get_timestamp()
        self._now = value

    @property
    def date(self):
        date = str(datetime.now().date())
        self._date = date
        return self._date
    @date.setter
    def date(self, value):
        self._date = value

    def get_date_timestamp(self, year_mod=0, month_mod=0, day_mod=0, *args,
**kwargs):
        modifier = [year_mod, month_mod, day_mod]

        now_mod = (datetime.now()+timedelta(days=day_mod))
        date = (str(now_mod.date()).replace("-0", "-")).split("-")
        date = [int(string) for string in date]

        timestamp = datetime(date[0], date[1], date[2]).timestamp()

        return timestamp

    def get_timestamp(self):
        now = (float(datetime.now().timestamp()))
        return now

    def get_date(self, timestamp):
        date = datetime.utcfromtimestamp(timestamp).strftime('%Y-%m-%d')
        return date

    def get_days_month(self, month, year):
        pass
```

modules/ui/beopen.kv

```
#UTILITY Widgets
<ScrollingView@MDScrollView>
  do_scroll_y: True
  do_scroll_x: False

<ScrollArea@MDFloatLayout>
  padding: 20
  size_hint_y: None

<ScrollAreaGrid@MDGridLayout>
  padding: 10
  cols: 1
  size_hint_y: None

<ScrollAreaBox@ScrollAreaGrid>
  MDBoxLayout:
    orientation: "vertical"

<ScrollAreaBoxLayout@ScrollAreaBox>

<HelpDialog>
  text: "Help box"
  text_color: "Black"

<ExpandText>
  text: "Expanded text"
  font_style: "H6"

<ExpandPage>:
  toolbar: toolbar
  text_area: text_area

  MDBoxLayout:
    orientation: "vertical"

    MDTopAppBar:
      id: toolbar
      title: ""
      anchor_title: "right"
      left_action_items: [["arrow-left", lambda x: root.back(app)]]

    MDBoxLayout:
      orientation: "vertical"
      padding: 10
```

```
      ScrollingView:
        ScrollArea:
          id: text_area

<SwiperMagicButton>
  opposite_colors: True
  icon_size: 35

<MemoriesSwiper>
  RelativeLayout:
    orientation: "horizontal"

    FitImage:
      source: ""
      radius: [20,]

    MDBoxLayout:
      adaptive_height: True
      orientation: "horizontal"
      pos_hint: {'top': 1}
      spacing: 12
      padding: 10

      MDLabel:
        text: "99"
        font_style: "H5"
        font_size: 20
        size_hint_y: None
        height: self.texture_size[1]
        pos_hint: {"center_y": .5}
        opposite_colors: True

<MemoriesMonth>
  orientation: "vertical"
  spacing: 10

  MDLabel:
    text: "Month placeholder"
    font_style: "H5"
    font_size: 40
    size_hint_y: 0.1

  MDGridLayout:
    size_hint_y: 0.9
    cols: 5
    spacing: 10
    row_default_height: root.get_memories_swiper_height()
```

```
<HomeSwiper>
  account_button: account_button
  profile_area: profile_area
  RelativeLayout:
    orientation: "horizontal"

    FitImage:
      id: content
      source: ""
      radius: [20,]

    MDBoxLayout:
      adaptive_height: True
      orientation: "horizontal"
      spacing: 12

      MDBoxLayout:
        orientation: "horizontal"
        adaptive_width: True
        adaptive_height: True

        SwiperMagicButton:
          id: like
          icon: "heart-outline"
          on_release:
            root.like()

        MDLabel:
          id: like_number
          text: "0"
          font_style: "H5"
          text_color: "white"
          theme_text_color: "Custom"
          pos_hint: {'center_x': 0.5, 'center_y': 0.5}

      MDBoxLayout:
        adaptive_height: True
        orientation: "horizontal"

        SwiperMagicButton:
          icon: "comment-outline"
          on_release:
            root.switch_to_comments(app)

        MDLabel:
          id: caption
          text: "Caption placeholder"
```

```
        adaptive_height: True
        font_style: "H5"
        text_color: "white"
        theme_text_color: "Custom"
        pos_hint: {'center_x': 0.5, 'center_y': 0.5}

    MDBoxLayout:
      id: profile_area
      adaptive_height: True
      orientation: "horizontal"
      pos_hint: {'top': 1}
      spacing: "12dp"

      SwiperMagicButton:
        id: account_button
        icon: "account-circle"
        on_release: root.post_options(app)

      MDLabel:
        id: username
        text: "Name Placeholder"
        font_style: "H5"
        size_hint_y: None
        height: self.texture_size[1]
        pos_hint: {"center_y": .5}
        opposite_colors: True

<NoPostLabel>
  MDBoxLayout:
    orientation: "vertical"

    MDLabel:
      text: "No posts :("
      font_style: "H4"
      halign: "center"

<HomeLoadButton>
  padding: 30

  MDRaisedButton:
    size_hint_x: 1
    pos_hint: {"center_x": .5, "center_y": .95}
    text: "Load more"
    on_release:
      root.load_content()

<OccupationPageButton>
  text: "Occupations"
```

```
  pos_hint: {'center_x': 0.5}
  on_release: root.switch_to_occupation(app)
  size_hint_x: 0.8

<OrganisationBottomItem>
  occupation_button_area: occupation_button_area
  team_button_area: team_button_area
  name: "Organisation"
  text: "Organisation"
  icon: "account-group"
  on_tab_press: root.page.on_tab_press(self.name)

  MDBoxLayout:
    orientation: "vertical"
    padding: 20

    ScrollingView:
      ScrollAreaBoxLayout:
        spacing: 50

        MDBoxLayout:
          id: occupation_button_area

        MDBoxLayout:
          id: team_button_area

          MDRaisedButton:
            text: "Teams"
            pos_hint: {'center_x': 0.5}
            on_release: root.switch_to_team(app)
            size_hint_x: 0.8

<MemoryLayout>
  post_area: post_area
  orientation: "vertical"
  spacing: 10

  MDIconButton:
    icon: "arrow-up"
    on_release: root.day_list.back()

  MDBoxLayout:
    id: post_area
    orientation: "vertical"

<MonthListItem>
  on_release: root.day_view()
```

```
<MonthList>
  scroll: scroll
  orientation: "vertical"
  spacing: 10

  MDScrollView:
    do_scroll_x: False
    do_scroll_y: True

    MDList:
      id: scroll

<DayListItem>
  on_release: root.post_open()

<DayList>
  scroll: scroll
  orientation: "vertical"
  spacing: 10

  MDIconButton:
    icon: "arrow-up"
    on_release: root.back()

  MDScrollView:
    do_scroll_x: False
    do_scroll_y: True

    MDList:
      id: scroll

<HomePage>:
  home_swiper_scroll:home_swiper_scroll
  home_swiper_grid:home_swiper_grid
  bottom_navigation: bottom_navigation
  toolbar: toolbar
  root_scroll: root_scroll

  MDBoxLayout:
    orientation: "vertical"

    MDTopAppBar:
      title: "BeOpen"
      id: toolbar
      anchor_title: "left"
      right_action_items: [["help", lambda x: root.open_help(app)], ["cog", lambda x:
root.switch_to_settings(app, 'left')]]
```

```
      left_action_items: [["account-circle", lambda x: root.switch_to_account(app,
direction='right')],["bell", lambda x: root.switch_to_notifications(app, direction='right')]]
      md_bg_color: app.theme_cls.primary_color

    MDBottomNavigation:
      id: bottom_navigation

      MDBottomNavigationItem:
        name: "Home"
        text: "Home"
        icon: "home"
        on_tab_press: root.on_tab_press(self.name)

        MDBoxLayout:
          padding: 20
          size_hint: 1, 1

          MDScrollView:
            id:home_swiper_scroll
            do_scroll_x: False
            do_scroll_y: True

            MDGridLayout:
              id:home_swiper_grid
              cols: 1
              spacing: 10
              adaptive_height: True

      MDBottomNavigationItem:
        name: "Memories"
        text: "Memories"
        icon: "image-multiple"
        on_tab_press: root.on_tab_press(self.name)

        MDBoxLayout:
          padding: 20

          MDBoxLayout:
            id: root_scroll
            orientation: "vertical"

      MDBottomNavigationItem:
        name: "Stats"
        text: "Stats"
        icon: "poll"
        on_tab_press: root.on_tab_press(self.name)

        MDLabel:
```

```
        text: "Coming Soon!"
        font_size: "50sp"
        halign: "center"

# Comments Widgets
<Comment>
  like_button: like_button
  profile_button: profile_button
  like_count: like_count
  container: container
  text: ""
  secondary_text: ""
  on_release: root.expand(app)
  on_size:
    self.ids._right_container.width = container.width
    self.ids._right_container.x = container.width

  IconLeftWidget:
    id: profile_button
    icon: "account"
    on_release: root.action_options(app)

  CommentContainer:
    id: container
    adaptive_width: True

    MDIconButton:
      id: like_button
      icon: "heart-outline"
      on_release: root.like()

    MDLabel:
      id: like_count
      text: "3000"
      halign: "right"

<CommentsPage>:
  comment_stack: comment_stack
  comment_field: comment_field

  MDBoxLayout:
    orientation: "vertical"

    MDBoxLayout:
      padding: 10
      size_hint: 1, 0.15
      orientation: "vertical"
```

```
      AnchorLayout:
        anchor_x: 'center'
        anchor_y: 'top'

        MDIconButton:
          icon: "menu-up"
          icon_size: "64sp"
          icon_color: app.theme_cls.primary_color
          on_release:
            root.switch_to_home(app)

    MDBoxLayout:
      orientation: "vertical"
      padding: 10
      spacing: 20

      ScrollingView:
        MDList:
          id: comment_stack

      MDRelativeLayout:
        size_hint_y: None
        height: comment_field.height
        pos_hint: {"bottom": 0.5}

        MDTextField:
          id: comment_field
          hint_text: "Enter your comment here..."
          icon_left: "comment"
          mode: "rectangle"

        MDIconButton:
          icon: "send"
          post_hint: {"center_y": 0.5}
          pos: comment_field.width - self.width + dp(8), 0
          on_release: root.submit()

# account widgets
<ProfilePicture>
  size_hint_x: 0.4
  pos_hint: {"center_x": .5}
  source: "data/assets/profile.png"

<InfoChangeBox>
  pos_hint: {"center_y": .5}
  text_field: text_field
  spacing: 10
```

```
  MDTextField:
    id: text_field
    hint_text: "Enter a new value"
    on_text_validate: root.save()
    mode: "rectangle"
    size_hint_x: 0.8
    pos_hint: {"center_y": .5}

  MDRaisedButton:
    text: "Save"
    on_release: root.save()
    size_hint_x: 0.2
    pos_hint: {"center_y": .5}

<InfoChangeBoxOld>
  pos_hint: {"center_y": .5}
  hint_text: "Enter a new value"
  mode: "rectangle"
  on_text_validate:
    self.submit_func()

<InfoEditButton>
  icon: "pencil"
  on_release:
    self.change_info()

<BioEditButton>
  icon: "pencil"
  size_hint_x: 0.1
  pos_hint: {"center_y": .5}
  on_release:
    self.change_info()

<ProfileInfo>
  text: ""
  secondary_text: "None"

<UserOccupationChange>
  orientation: "vertical"
  spacing: 20
  occupation_select: occupation_select
  request_title: request_title
  request_occupation: request_occupation
  request_status: request_status
  request_cancel: request_cancel
  request_button: request_button
  occupation_description: occupation_description
```

```
  MDBoxLayout:
    id: new_request_area
    orientation: "vertical"

    MDLabel:
      id: new_request_title
      text: "new request"
      font_style: 'H5'
      pos_hint: {'center_x': .5}
      size_hint_y: 0.1

    MDBoxLayout:
      orientation: "horizontal"
      size_hint_y: 0.9
      spacing: 10

      MDRaisedButton:
        id: occupation_select
        text: "Select an occupation"
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: root.selection_menu()

      MDLabel:
        id: occupation_description
        text: ""
        font_style: 'H6'
        pos_hint: {'center_x': .5, 'center_y': .5}

      MDRaisedButton:
        id: request_button
        text: "Create request"
        pos_hint: {'center_y': .5}
        on_release: root.submit()

  MDBoxLayout:
    orientation: "horizontal"

    MDBoxLayout:
      orientation: "vertical"
      spacing: 10

      MDLabel:
        id: request_title
        text: "current request"
        font_style: 'H5'
      MDLabel:
        id: request_occupation
        text: ""
```

```
        font_style: 'H5'

    MDLabel:
      id: request_status
      text: "Status: No request"
      font_style: 'H6'
      pos_hint: {'right': 1}

    MDRaisedButton:
      id: request_cancel
      text: "Cancel"
      pos_hint: {"center_y": .5}
      on_release: root.cancel()

<ManagementOccupationChange>
  orientation: "vertical"
  spacing: 20
  occupation_select: occupation_select
  occupation_description: occupation_description
  set_button: set_button

  MDBoxLayout:
    orientation: "vertical"

    MDLabel:
      text: "Change occupation"
      font_style: 'H5'
      pos_hint: {'center_x': .5}
      size_hint_y: 0.1

    MDBoxLayout:
      orientation: "horizontal"
      size_hint_y: 0.9
      spacing: 10

      MDRaisedButton:
        id: occupation_select
        text: "Select an occupation"
        pos_hint: {'center_x': .5, 'center_y': .5}
        on_release: root.selection_menu()

      MDLabel:
        id: occupation_description
        text: "Occupation description"
        font_style: 'H6'
        pos_hint: {'center_x': .5, 'center_y': .5}

      MDRaisedButton:
```

```
        id: set_button
        text: "Create request"
        pos_hint: {'center_y': .5}
        on_release: root.submit()

<AccountPage>:
  name: 'account_page_screen'

  toolbar: toolbar
  profile_info_view: profile_info_view
  profile_bio_view: profile_bio_view
  biography_content: biography_content
  above_info: above_info

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Profile"
      anchor_title: "center"
      left_action_items: [["account-multiple", lambda x: root.switch_to_friend(app)]]
      right_action_items: [["help", lambda x: root.open_help(app)], ["arrow-right", lambda x:
root.back(app, "left")]]
      md_bg_color: app.theme_cls.primary_color

    MDBoxLayout:
      orientation: "vertical"
      size_hint_y: 0.8

      MDBoxLayout:
        orientation: "vertical"
        size_hint_y: 0.70
        padding: 15
        spacing: 10

        MDBoxLayout:
          id: above_info
          size_hint_y: 0.3

        MDScrollView:
          size_hint_y: 0.45

          MDList:
            id: profile_info_view

        MDBoxLayout:
```

```
        size_hint_y: 0.25
        orientation: "vertical"
        spacing: 10

        MDLabel:
          size_hint_y: 0.1
          text: "Biography"
          font_style: "H5"

        MDBoxLayout:
          id: profile_bio_view
          orientation: "horizontal"
          size_hint_y: 0.9

          ScrollView:
            size_hint_x: 0.9
            do_scroll_x: False
            do_scroll_y: True

            MDLabel:
              id: biography_content
              size_hint_y: None
              size: self.texture_size
              text: ""
              font_style: "H6"

# Settings widgets
<SettingSwitch>
  setting_title: setting_title
  setting_description: setting_description
  toggle: toggle
  setting_icon: setting_icon

  AnchorLayout:
    anchor_x: 'right'
    anchor_y: 'center'
    padding: 20

    MDBoxLayout:
      orientation: "horizontal"
      adaptive_height: True
      spacing: 20

      MDIcon:
        id: setting_icon
        icon: "language-python"

      MDBoxLayout:
```

```
        orientation: "vertical"
        spacing: 20

        MDLabel:
          id: setting_title
          text: "switch 1"
          font_size: "30sp"
          text_color: 0, 1, 1, 1

        MDLabel:
          id: setting_description
          text: "description text"
          text_color: 0, 0, 1, 1

    MDSwitch:
      id: toggle
      on_active: root.on_toggle(app)

<SettingTextField>
  setting_title: setting_title
  setting_description: setting_description
  input_field: input_field
  setting_icon: setting_icon

  AnchorLayout:
    anchor_x: 'right'
    anchor_y: 'center'
    padding: 20

    MDBoxLayout:
      orientation: "horizontal"
      adaptive_height: True
      spacing: 20

      MDIcon:
        id: setting_icon
        icon: "language-python"

      MDBoxLayout:
        orientation: "vertical"
        spacing: 20

        MDLabel:
          id: setting_title
          text: "switch 1"
          font_size: "30sp"
          text_color: 0, 1, 1, 1
```

```
        MDLabel:
          id: setting_description
          text: "description text"
          text_color: 0, 0, 1, 1

    MDTextField:
      id: input_field
      hint_text: "default hint text"
      helper_text_mode: "on_error"
      helper_text: "default helper text"
      on_text_validate: root.submit_func()

<SettingTextFieldOld>
  hint_text: "default hint text"
  helper_text_mode: "on_error"
  helper_text: "default helper text"
  on_text_validate: root.submit_func()

<ShutdownButton>
  text: "Shutdown server"
  size_hint_x: 0.8
  pos_hint: {'center_x': 0.5}
  on_release: root.shutdown(app)

<SettingsPage>:
  settings_stack: settings_stack
  static_buttons: static_buttons
  toolbar: toolbar

  MDBoxLayout:
    orientation: "vertical"

    MDTopAppBar:
      id: toolbar
      title: "Settings"
      anchor_title: "center"
      left_action_items: [["arrow-left", lambda x: root.back(app)]]
      right_action_items: [["help", lambda x: root.open_help(app)]]
      md_bg_color: app.theme_cls.primary_color

    MDBoxLayout:
      orientation: "vertical"
      padding: 20
      spacing: 10

      MDBoxLayout:
        id: settings_stack
        orientation: "vertical"
```

```
        spacing: 10

      MDBoxLayout:
        orientation: "vertical"
        spacing: 10
        id: static_buttons

      MDRaisedButton:
        text: "Log out"
        size_hint_x: 0.8
        pos_hint: {'center_x': 0.5}
        on_release: root.logout(app)

# notification widgets
<NotificationItem>
  text: ""
  secondary_text: ""
  on_release:
    root.expand()

  IconRightWidget:
    icon: "close"
    on_release:
      root.delete()

<NotificationsPage>:
  toolbar: toolbar
  notification_layout: notification_layout
  notification_stack: notification_stack

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Notifications"
      anchor_title: "center"
      left_action_items: [["refresh", lambda x: root.load_content()]]
      right_action_items: [["help", lambda x: root.open_help(app)], ["arrow-right", lambda x:
root.back(app)]]

    MDBoxLayout:
      id: notification_layout
      orientation: "vertical"
      size_hint_y: 0.9

      MDScrollView:
```

```
        size_hint_y: 1
        do_scroll_y: True

        MDList:
          id: notification_stack

# friends widgets
<FriendItem>
  text: "Friend"
  on_release: root.profile(app)

  IconRightWidget:
    icon: "account-remove"
    on_release:
      root.remove()

<FriendPage>:
  toolbar: toolbar
  friend_list: friend_list

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Friends"
      anchor_title: "center"
      left_action_items: [["help", lambda x: root.open_help(app)]]
      right_action_items: [["arrow-right", lambda x: root.switch_to_account(app)]]

    MDBoxLayout:
      orientation: "vertical"

      MDRaisedButton:
        text: "Requests"
        on_release: root.switch_to_friend_request(app)
        size_hint_x: 0.9
        pos_hint: {'center_x': 0.5}

      ScrollingView:
        MDList:
          id: friend_list

<IncomingRequestItem>
  text: "Incoming Request"
  on_release: root.profile(app)
```

```
  IconLeftWidget:
    icon: "check"
    on_release:
      root.accept()
  IconRightWidget:
    icon: "close"
    on_release:
      root.reject()

<OutgoingRequestItem>
  text: "Outgoing Request"
  on_release: root.profile(app)

  IconRightWidget:
    icon: "close"
    on_release:
      root.cancel()

<RecomendationItem>
  text: "Recomended friend"
  on_release: root.profile(app)

  IconRightWidget:
    icon: "account-plus"
    on_release:
      root.add_friend()

<FriendRequestPage>:
  toolbar: toolbar
  incoming_requests: incoming_requests
  outgoing_requests: outgoing_requests
  recomendations: recomendations
  username_select: username_select

  MDBoxLayout:
    orientation: "vertical"

    MDTopAppBar:
      id: toolbar
      title: "Requests"
      anchor_title: "center"
      left_action_items: [["refresh", lambda x: root.load_content()]]
      right_action_items: [["arrow-right", lambda x: root.switch_to_friend(app)]]

    MDBoxLayout:
      size_hint_y: 0.1
      orientation: "horizontal"
      spacing: 10
```

```
      padding: 10

      MDTextField:
        id: username_select
        mode: "rectangle"
        hint_text: "Enter a username"
        pos_hint: {'center_y': 0.5}

      MDRaisedButton:
        text: "Request"
        on_release: root.add_friend_search()
        pos_hint: {'center_y': 0.5}

    MDBoxLayout:
      orientation: "vertical"
      size_hint_y: 0.2

      MDBoxLayout:
        orientation: "vertical"
        padding: 10
        MDLabel:
          text: "Recomendations"
          font_style: "H5"
          size_hint_y: 0.1

        MDBoxLayout:
          size_hint_y: 0.9
          padding: 10
          ScrollingView:
            MDList:
              id: recomendations

    MDBoxLayout:
      orientation: "vertical"
      padding: 10
      size_hint_y: 0.7

      MDBoxLayout:
        orientation: "vertical"
        MDLabel:
          text: "Incoming Requests"
          font_style: "H5"
          size_hint_y: 0.1

        MDBoxLayout:
          size_hint_y: 0.9
          padding: 10
          ScrollingView:
```

```
            MDList:
              id: incoming_requests

        MDBoxLayout:
          orientation: "vertical"
          MDLabel:
            text: "Outgoing Requests"
            font_style: "H5"
            size_hint_y: 0.1

          MDBoxLayout:
            size_hint_y: 0.9
            padding: 10
            ScrollingView:
              MDList:
                id: outgoing_requests

# management widgets
<ManageOccupationChange>
  orientation: "vertical"
  change_button: change_button
  occupation_select: occupation_select
  occupation_description: occupation_description
  username_select: username_select

  MDLabel:
    text: "Change a users occupation (this will change their team)"
    font_style: 'H5'
    pos_hint: {'center_x': .5}
    size_hint_y: 0.1

  MDBoxLayout:
    orientation: "horizontal"
    size_hint_y: 0.9

    MDTextField:
      id: username_select
      hint_text: "Enter a username"

    MDRaisedButton:
      id: occupation_select
      text: "Select an occupation"
      pos_hint: {'center_x': .5, 'center_y': .5}
      on_release: root.selection_menu()

    MDLabel:
      id: occupation_description
      text: "Occupation description"
```

```
      font_style: 'H6'
      pos_hint: {'center_x': .5, 'center_y': .5}

    MDRaisedButton:
      id: change_button
      text: "Change occupation"
      pos_hint: {'center_y': .5}
      on_release: root.submit()

<OccupationItem>
  text: "Occupation name"
  secondary_text: "Description"

  IconLeftWidget:
    icon: "close"
    on_release:
      root.delete()
  IconRightWidget:
    icon: "pencil"
    on_release:
      root.edit()

<OccupationRequestItem>
  text: "Username"
  secondary_text: "Occupation name"
  tirtiary_text: "Description"

  IconRightWidget:
    icon: "check"
    on_release:
      root.accept()

  IconLeftWidget:
    icon: "close"
    on_release:
      root.reject()

<OccupationEdit>
  orientation: "vertical"
  spacing: 10

  MDLabel:
    text: "Edit occupation"
    font_style: "H5"
    size_hint_y: 0.1
  MDTextField:
    id: name
    mode: "rectangle"
```

```
    hint_text: "Name"
    size_hint_x: 1
    size_hint_y: 0.35
  MDTextField:
    id: description
    mode: "rectangle"
    hint_text: "Description"
    size_hint_x: 1
    size_hint_y: 0.35
  MDRaisedButton:
    text: "Done"
    size_hint_x: 1
    on_release: root.submit()
    size_hint_y: 0.2

<OccupationCreate>
  orientation: "vertical"
  spacing: 10

  MDLabel:
    text: "Create an occupation"
    font_style: "H5"
    size_hint_y: 0.1
  MDTextField:
    id: name
    mode: "rectangle"
    hint_text: "Name"
    size_hint_x: 1
    size_hint_y: 0.35
  MDTextField:
    id: description
    mode: "rectangle"
    hint_text: "Description"
    size_hint_x: 1
    size_hint_y: 0.35
  MDRaisedButton:
    text: "Done"
    size_hint_x: 1
    on_release: root.create()
    size_hint_y: 0.2

<OccupationPage>:
  toolbar: toolbar
  occupations: occupations
  edit_area: edit_area

  MDBoxLayout:
    orientation: "vertical"
```

```
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Occupation"
      anchor_title: "center"
      left_action_items: [["arrow-left", lambda x: root.back(app)]]
      right_action_items: [["help", lambda x: root.open_help(app)]]

    MDRaisedButton:
      text: "Requests"
      on_release: root.switch_to_occupation_request(app)
      size_hint_x: 0.9
      pos_hint: {'center_x': 0.5, 'center_y': 0.5}

    MDBoxLayout:
      id: edit_area
      orientation: "vertical"
      padding: 15

    MDBoxLayout:
      orientation: "vertical"
      padding: 15

      MDLabel:
        text: "Occupations"
        font_style: "H5"
        size_hint_y: 0.1

      MDBoxLayout:
        size_hint_y: 0.9
        padding: 10
        ScrollingView:
          MDList:
            id: occupations

<OccupationRequestPage>:
  toolbar: toolbar
  change_requests: change_requests

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Requests"
      anchor_title: "center"
```

```
      left_action_items: [["arrow-left", lambda x: root.back(app)]]
      right_action_items: [["refresh", lambda x: root.load_content()]]

    MDBoxLayout:
      orientation: "vertical"
      padding: 10

      MDLabel:
        text: "Occupation change requests"
        font_style: "H5"
        size_hint_y: 0.1

      MDBoxLayout:
        size_hint_y: 0.9
        padding: 10
        ScrollingView:
          MDList:
            id: change_requests

<LeaderItem>
  IconRightWidget:
    icon: "close"
    on_release: root.delete()

<AddLeaderButton>
  text: "Add leader"
  size_hint_x: 0.8
  on_release: root.add_leader()
  pos_hint: {'center_x': 0.5}

<AddLeader>
  orientation: "vertical"
  spacing: 10

  MDLabel:
    text: "Add a leader"
    font_style: "H5"
    size_hint_y: 0.2
  MDTextField:
    id: username
    mode: "rectangle"
    hint_text: "Name"
    size_hint_y: 0.5
  MDRaisedButton:
    text: "Done"
    on_release: root.submit()
    size_hint_y: 0.3
    size_hint_x: 1
```

```
<ChangeNameButton>
  text: "Change team name"
  size_hint_x: 0.8
  on_release: root.change_name()
  pos_hint: {'center_x': 0.5}

<ChangeName>
  orientation: "vertical"
  spacing: 10

  MDLabel:
    text: "Change team name"
    font_style: "H5"
    size_hint_y: 0.2
  MDTextField:
    id: name
    mode: "rectangle"
    hint_text: "Name"
    size_hint_y: 0.5
  MDRaisedButton:
    text: "Done"
    on_release: root.submit()
    size_hint_y: 0.3
    size_hint_x: 1

<TeamPage>:
  toolbar: toolbar
  members: members
  leaders: leaders
  team_name: team_name
  edit_area: edit_area

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Team"
      anchor_title: "center"
      left_action_items: [["arrow-left", lambda x: root.back(app)]]
      right_action_items: [["help", lambda x: root.open_help(app)], ["refresh", lambda x:
root.load_content()]]

    MDBoxLayout:
      orientation: "vertical"
      padding: 10
```

```
      spacing: 10

      MDBoxLayout:
        orientation: "vertical"

        MDLabel:
          id: team_name
          text: ""
          font_style: "H5"
          font_size: 30
          pos_hint: {'center_x': 0.93}
          size_hint_y: 0.1

        MDBoxLayout:
          id: edit_area
          orientation: "vertical"
          size_hint_y: 0.9
          spacing: 10

      MDBoxLayout:
        orientation: "vertical"
        MDLabel:
          text: "Leader"
          font_style: "H5"
          size_hint_y: 0.1

        MDBoxLayout:
          size_hint_y: 0.9
          padding: 10
          ScrollingView:
            MDList:
              id: leaders

      MDBoxLayout:
        orientation: "vertical"
        MDLabel:
          text: "Members"
          font_style: "H5"
          size_hint_y: 0.1

        MDBoxLayout:
          size_hint_y: 0.9
          padding: 10
          ScrollingView:
            MDList:
              id: members

# auth widgets
```

```
<PasswordField>
  size_hint_y: None
  height: password_field.height
  text: password_field.text

  MDTextField:
    id: password_field
    hint_text: "Password"
    text: ""
    password: True
    mode: "rectangle"
    icon_left: "key-variant"

  MDIconButton:
    icon: "eye-off"
    pos_hint: {"center_y": 0.45}
    pos: password_field.width - self.width + dp(8), 0
    theme_text_color: "Hint"
    on_release:
      self.icon = "eye" if self.icon == "eye-off" else "eye-off"
      password_field.password = False if password_field.password is True else True

<AuthField>
  adaptive_height: True
  mode: "rectangle"
  hint_text: ""

<AuthButton>
  text: "auth"
  pos_hint: {'center_x': 0.5}
  size_hint_x: 0.5
  on_release:
    self.action(app)

<LoginPage>:
  login_view: login_view

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      title: "Login"
      anchor_title: "center"

    MDGridLayout:
      id: login_view
      padding: 20
```

```
      spacing: 10
      row_default_height: 50
      cols: 1
      rows: 4

<RegisterPage>:
  register_view: register_view

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      title: "Register"
      anchor_title: "center"
      left_action_items: [["arrow-left", lambda x: app.set_screen("LoginPageScreen",
"right")]]

    MDGridLayout:
      id: register_view
      padding: 20
      spacing: 10
      row_default_height: 50
      cols: 1
      rows: 6

<ServerPage>:
  url: url
  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      title: "Select Server"
      anchor_title: "center"

    MDGridLayout:
      padding: 20
      spacing: 10
      row_default_height: 50
      cols: 1
      rows: 2

      MDTextField:
        id: url
        adaptive_height: True
        mode: "rectangle"
        hint_text: "Server URL"
```

```
        helper_text: "Remember to start with http:// or https://"

      MDRaisedButton:
        text: "Connect"
        pos_hint: {'center_x': 0.5}
        size_hint_x: 0.5
        on_release:
          root.connect(app)

<ShareInput>
  pos_hint: {'center_y': 0.5}
  spacing: 10
  share_num: share_num
  share_secret: share_secret

  MDTextField:
    id: share_num
    mode: "rectangle"
    hint_text: "Share Number"
    size_hint_x: 0.2

  MDTextField:
    id: share_secret
    password: True
    mode: "rectangle"
    hint_text: "Share Secret"
    helper_text: "This is the secret provided by the admin"
    size_hint_x: 0.8

<DecryptPage>:
  input_area: input_area
  en_password: en_password
  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      title: "Decrypt Server Database"
      anchor_title: "center"

    MDBoxLayout:
      orientation: "vertical"
      padding: 20
      spacing: 20

      MDTextField:
        id: en_password
        password: True
```

```
        mode: "rectangle"
        hint_text: "Master password"
        helper_text: "This can still be used even if Shamir Secret Sharing is enabled"
        size_hint_x: 0.8
        pos_hint: {'center_x': 0.5}

      MDBoxLayout:
        id: input_area
        orientation: "vertical"

      MDRaisedButton:
        text: "Submit"
        pos_hint: {'center_x': 0.5}
        size_hint_x: 0.8
        on_release: root.submit()

<CameraPage>:
  camera_area: camera_area
  toolbar: toolbar

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Time left: "
      anchor_title: "center"
      left_action_items: [["arrow-up", lambda x: root.exit(app)]]
      right_action_items: [["help", lambda x: root.open_help(app)]]

    MDBoxLayout:
      id: camera_area
      orientation: "vertical"
      size_hint_y: 0.7
      padding: 10

    MDBoxLayout:
      orientation: "vertical"
      size_hint_y: 0.2
      padding: 10

      MDIconButton:
        icon: "camera"
        on_release: root.capture(app)
        pos_hint: {'center_x': 0.5, 'center_y': 0.5}

<PostReviewPage>:
```

```
  image: image
  caption: caption
  toolbar: toolbar

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10

    MDTopAppBar:
      id: toolbar
      title: "Time left: "
      anchor_title: "center"
      left_action_items: [["arrow-up", lambda x: root.exit(app)]]
      right_action_items: [["help", lambda x: root.open_help(app)]]

    MDBoxLayout:
      orientation: "vertical"
      size_hint_y: 0.1
      padding: 10

      MDRaisedButton:
        text: "Retake photo?"
        on_release: root.retake(app)
        size_hint_x: 1
        pos_hint: {'center_y': 0.5}

    MDBoxLayout:
      id: image_area
      orientation: "vertical"
      size_hint_y: 0.5
      padding: 10

      FitImage:
        id: image
        source: ""

    MDBoxLayout:
      orientation: "vertical"
      size_hint_y: 0.3
      padding: 15
      spacing:10

      MDTextField:
        id: caption
        hint_text: "Caption"
        helper_text: "This text will appear alongside your photo"
        size_hint_x: 1
```

```
      MDRaisedButton:
        text: "Post"
        size_hint_x: 1
        on_release: root.post(app)

<FirstTimePage>:
  name_input: name_input
  role_input: role_input

  MDBoxLayout:
    orientation: "vertical"
    spacing: 10
    padding: 20

    MDBoxLayout:
      id: step0
      orientation: "vertical"
      size_hint_y: 0.1

      MDLabel:
        text: "Before doing anything else, please download the ntfy app or navigate to your
organisations ntfy site. Then subscribe to the topic:"
        font_style: "H6"

      MDLabel:
        id: topic_name
        text: "Could not fetch topic name"
        font_style: "H5"

    MDBoxLayout:
      id: step1
      orientation: "vertical"
      size_hint_y: 0.15

      MDLabel:
        text: "Step 1: Tell us your name"

      MDTextField:
        id: name_input
        hint_text: "Name"

    MDBoxLayout:
      id: step2
      orientation: "vertical"
      size_hint_y: 0.15

      MDLabel:
        text: "Step 2: Let everyone know your role in the organisation"
```

```
      MDTextField:
        id: role_input
        hint_text: "Role"

    MDBoxLayout:
      id: step3
      orientation: "vertical"
      size_hint_y: 0.3

      MDLabel:
        text: "Step 3: Request to set your occupation, people with the same occupation will be grouped together into the same team. People in the same team will be able to see eachothers posts"

    MDBoxLayout:
      id: summary
      orientation: "vertical"
      size_hint_y: 0.3

      MDLabel:
        text: "Management or an admin will approve your request soon!\nFor now head over to the friends page and send some friend requests"
      MDLabel:
        text: "To get there click the profile button in the top left hand corner the homepage, and then click the friends button in the top left hand corner on the profile page"

      MDRaisedButton:
        text: "Done"
        size_hint_x: 0.8
        pos_hint: {'center_x': 0.5}
        on_release: root.done(app)
```

data/assets/help.txt

```
[Home:START]
  (title:START)
    Home page
  (title:END)

  (body:START)
    This is your home page, below you will is where you will see your friends posts.
    In the top left corner is your "profile" there you will be able to see all about you, and find friends
    Next to that is yout notifications to keep you up-to-date with your colleges
    Your settings panel is in the top left and additional areas are available in the navigation panel at the bottom of the page
  (body:END)
```

[Home:END]

[Memories:START]
(title:START)
Memories page
(title:END)

(body:START)
Here is where you will see your own historical posts. This means posts from previous days.
Just select a month and then the day of the month and you will see your old post. You can still like
and comment on it but others will not be able to see these interactions.
(body:END)
[Memories:END]

[Organisation:START]
(title:START)
Organisation page
(title:END)

(body:START)
Here you will see the navigation to your teams panel, and soon some infomation about your organisations
Your teams panel is where you will be able to see all about your team.

You can only be part of one organisation, since your account is specifically tied to it.
(body:END)
[Organisation:END]

[Organisation-admin:START]
(title:START)
Organisation page
(title:END)

(body:START)
Here you will see the navigation to your teams panel, and soon some infomation about your organisations
Your teams panel is where you will be able to see all about your team.
Additionally since your are management/admin staff you can see the occupation area here you can accept and reject occupation change requests, create new occupations and edit or delete current ones.
(body:END)
[Organisation-admin:END]

[Profile:START]
(title:START)
Profile page

(title:END)

(body:START)
This page you can see all the infomation about the profile your viewing. For your own profile you should see edit buttons next to some of infomation this means you can change or request to change this info.

Your role is simply how you describe your role in your team for instance "assistant"
Your occupation determines what team you are put into, you can only be part of 1 team at a time.
(body:END)
[Profile:END]

[Notifications:START]
(title:START)
Notifications page
(title:END)

(body:START)
Here you will see notifications from your organisations admins and management, as well notifications about people interacting with your posts and when its time to make your post.
(body:END)
[Notifications:END]

[Settings:START]
(title:START)
Settings page
(title:END)

(body:START)
This is where you can configure some options, for instance to stop seeing these help buttons toggle "Help Dialogs".
(body:END)
[Settings:END]

[Team:START]
(title:START)
Teams page
(title:END)

(body:START)
Your sorted into a team via your occupation, each occupation will have a team associated with it.
People in the same team as you will see your posts and you will see theres. This means you will see posts from
your team alongside posts from your friends.

Team leaders act as the moderator for the team. This means they can delete your posts and comments even if your
commenting on your friends posts.
(body:END)
[Team:END]

[Occupation:START]
(title:START)
Occupations page
(title:END)

(body:START)
Here you can change edit and delete the occupations in your organisation, this panel is only available for management and admins.
You can also approve and deny occupation change requests here. For a member to change their occupation they have to submit an occupation change request including what occupation they want to switch to. This request must be approved by management or above. Once approved the member is switched to their new team.
(body:END)
[Occupation:END]

[Friends:START]
(title:START)
Friends page
(title:END)

(body:START)
In BeOpen you dont follow people but friend people, this means both people accepted to be friends with eachother. Meaning you both see eachothers posts in your feed.
On this page you will be able to see your current friends and remove them if wanted. On the requests page (button below) you can send new friend requests, and approve or deny current friend requests. On that page is also a list of recomended friends based on mutal friends.

To send a friend request to someone (who isnt in the recomended) type their username into the request box, their username has to match exactly.
(body:END)
[Friends:END]

[Camera:START]
(title:START)
Making a post
(title:END)

(body:START)
You will be able to make one of these posts once a day, you will see the countdown at the top this is how long you have to post. After this time has run out you will not be able to post for that day.

To make a post just take a picture of whatever your doing right now and head onto the next screen.
(body:END)
[Camera:END]

[PostReview:START]
(title:START)
Making a post
(title:END)

(body:START)
Here you can review the image you just took and retake it if needed.
You can also add a caption to your post, note that you cannot edit a post after its been created you can however delete it. So be sure this is what you want to post today since deleting means you will have to wait till tomorrow to post again.
When your ready just hit post!
(body:END)
[PostReview:END]

org.flatpak.BeOpen.yml

```yaml
id: org.flatpak.BeOpen
runtime: org.freedesktop.Platform
runtime-version: '23.08'
sdk: org.freedesktop.Sdk
command: runner.sh
modules:
  - python3-requirements.json
  - name: cpython
    sources:
      - type: archive
        url: https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tar.xz
        sha256: f434053ba1b5c8a5cc597e966ead3c5143012af827fd3f0697d21450bb8d87a6
  - name: runner
    buildsystem: simple
    build-commands:
      - install -D main.py /app/main.py
      - install -D modules /app/modules
      - install -D runner.sh /app/bin/runner.sh
    sources:
      - type: file
        path: ../runner.sh
      - type: file
        path: ../main.py
      - type: dir
        path: ../modules
```

python3-requirements.json


```
{
    "name": "python3-requirements",
    "buildsystem": "simple",
    "build-commands": [],
    "modules": [
        {
            "name": "python3-python-socketio",
            "buildsystem": "simple",
            "build-commands": [
                "pip3 install --verbose --exists-action=i --no-index --find-
links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"python-socketio==5.8.0\" --no-
build-isolation"
            ],
            "sources": [
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/b5/82/ce0b6380f35f49d3fe687979a324c342cfa3588380232f3801db9dd62f9e/bidict-
0.22.1-py3-none-any.whl",
                    "sha256":
"6ef212238eb884b664f28da76f33f1d28b260f665fc737b413b287d5487d1e7b"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/95/04/ff642e65ad6b90db43e668d70ffb6736436c7ce41fcc549f4e9472234127/h11-0.14.0-
py3-none-any.whl",
                    "sha256":
"e3fe4ac4b851c468cc8363d500db52c2ead036020723024a109d37346efaa761"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/4f/ca/b14136484c9a10230abbf44a89041ccd2c696d0cb425e53f48ca0de0d1e7/python_en-
gineio-4.8.2-py3-none-any.whl",
                    "sha256":
"a357f0aba275c311b66f22181472ed5b174bbc541742eea1d16feae2fa1afabd"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/5d/e9/f296186e2a91f1472b9da74346163411196dc1b17f425acf088f293b32cc/py-
thon_socketio-5.8.0-py3-none-any.whl",
```

```
                    "sha256":
"7adb8867aac1c2929b9c1429f1c02e12ca4c36b67c807967393e367dfbb01441"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/6d/ea/288a8ac1d9551354488ff60c0ac6a76acc3b6b60f0460ac1944c75e240da/simple_web-
socket-1.0.0-py3-none-any.whl",
                    "sha256":
"1d5bf585e415eaa2083e2bcf02a3ecf91f9712e7b3e6b9fa0b461ad04e0837bc"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/78/58/e860788190eba3bcce367f74d29c4675466ce8dddfba85f7827588416f01/wsproto-
1.2.0-py3-none-any.whl",
                    "sha256": "b9ac-
ddd652b585d75b20477888c56642fdade28bdfd3579aa24a4d2c037dd736"
                }
            ]
        },
        {
            "name": "python3-eventlet",
            "buildsystem": "simple",
            "build-commands": [
                "pip3 install --verbose --exists-action=i --no-index --find-
links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"eventlet==0.33.3\" --no-build-
isolation"
            ],
            "sources": [
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/f6/b4/0a9bee52c50f226a3cbfb54263d02bb421c7f2adc136520729c2c689c1e5/dnspython-
2.4.2-py3-none-any.whl",
                    "sha256": "57c6fbaae-
aaf39c891292012060beb141791735dbb4004798328fc2c467402d8"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/90/97/928b89de2e23cc67136eccccf1c122adf74ffdb65bbf7d2964b937cedd4f/eventlet-
0.33.3-py2.py3-none-any.whl",
                    "sha256":
"e43b9ae05ba4bb477a10307699c9aff7ff86121b2640f9184d29059f5a687df8"
                },
                {
```

```
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/17/14/3bddb1298b9a6786539ac609ba4b7c9c0842e12aa73aaa4d8d73ec8f8185/greenlet-
3.0.3.tar.gz",
                    "sha256":
"43374442353259554ce33599da8b692d5aa96f8976d567d4badf263371fbe491"
                }
            ]
        },
        {
            "name": "python3-pathlib",
            "buildsystem": "simple",
            "build-commands": [
                "pip3 install --verbose --exists-action=i --no-index --find-
links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"pathlib==1.0.1\" --no-build-iso-
lation"
            ],
            "sources": [
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/78/f9/690a8600b93c332de3ab4a344a4ac34f00c8f104917061f779db6a918ed6/pathlib-
1.0.1-py3-none-any.whl",
                    "sha256": "f35f95ab8b0f59e6d354090350b44a80a80635d22ef-
dedfa84c7ad1cf0a74147"
                }
            ]
        },
        {
            "name": "python3-configparser",
            "buildsystem": "simple",
            "build-commands": [
                "pip3 install --verbose --exists-action=i --no-index --find-
links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"configparser\" --no-build-isola-
tion"
            ],
            "sources": [
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/81/a3/0e5ed11da4b7770c15f6f319abf053f46b5a06c7d4273c48469b7899bd89/con-
figparser-6.0.0-py3-none-any.whl",
                    "sha256":
"900ea2bb01b2540b1a644ad3d5351e9b961a4a012d4732f619375fb8f641ee19"
                }
            ]
        },
```

```
        {
            "name": "python3-datetime",
            "buildsystem": "simple",
            "build-commands": [
                "pip3 install --verbose --exists-action=i --no-index --find-
links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"datetime\" --no-build-isolation"
            ],
            "sources": [
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/ff/d5/f508192a563ab7415d1efbbe8d39cb9f0e510a1f6aaee3ca7d4ffed2a194/DateTime-
5.4-py3-none-any.whl",
                    "sha256":
"88caf4d2441fe479038f4740a1071953686f7c1ed6c9e8c7df9ebe84e592f0c6"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/32/4d/aaf7eff5deb402fd9a24a1449a8119f00d74ae9c2efa79f8ef9994261fc2/pytz-
2023.3.post1-py2.py3-none-any.whl",
                    "sha256":
"ce42d816b81b68506614c11e8937d3aa9e41007ceb50bfdcb0749b921bf646c7"
                },
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/87/03/6b85c1df2dca1b9acca38b423d1e226d8ffdf30ebd78bcb398c511de8b54/zope.inter-
face-6.1.tar.gz",
                    "sha256":
"2fdc7ccbd6eb6b7df5353012fbed6c3c5d04ceaca0038f75e601060e95345309"
                }
            ]
        },
        {
            "name": "python3-pillow",
            "buildsystem": "simple",
            "build-commands": [
                "pip3 install --verbose --exists-action=i --no-index --find-
links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"pillow==10.0.1\" --no-build-iso-
lation"
            ],
            "sources": [
                {
                    "type": "file",
```

```
                    "url": "https://files.pythonhosted.org/pack-
ages/64/9e/7e638579cce7dc346632f020914141a164a872be813481f058883ee8d421/Pillow-
10.0.1.tar.gz",
                    "sha256":
"d72967b06be9300fed5cfbc8b5bafceec48bf7cdc7dab66b1d2549035287191d"
                }
            ]
        },
        {
            "name": "python3-python-dotenv",
            "buildsystem": "simple",
            "build-commands": [
                "pip3 install --verbose --exists-action=i --no-index --find-
links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"python-dotenv==1.0.0\" --no-
build-isolation"
            ],
            "sources": [
                {
                    "type": "file",
                    "url": "https://files.pythonhosted.org/pack-
ages/44/2f/62ea1c8b593f4e093cc1a7768f0d46112107e790c3e478532329e434f00b/py-
thon_dotenv-1.0.0-py3-none-any.whl",
                    "sha256":
"f5971a9226b701070a4bf2c38c89e5a3f0d64de8debda981d1db98583009122a"
                }
            ]
        }
    ]
}
```

Appendix

SocketIO (python-socketio) – Networking and WebSocket library - documentation - https://python-socketio.readthedocs.io/en/stable/
Eventlet – Socketio deployment method – Documentation - https://eventlet.net/doc/
ConfigParser – For parsing the server-side configuration file – Documentation - https://docs.python.org/3/library/configparser.html/
Kivy - Client UI library - documentation – https://kivy.org/doc/stable/
KivyMD - Extension onto the Kivy UI library for material themed widgets - https://kivymd.readthedocs.io/en/1.1.1/

Shamir Secret Sharing – Paypal - https://max.levch.in/post/724289457144070144/shamir-secret-sharing-its-3am-paul-the-head-of
Shamir Secret Sharing – Basics - https://www.youtube.com/watch?v=K54ildEW9-Q

Federation – Mastodon a federated social media - https://joinmastodon.org/
Federation – Basics and dominant protocol - https://activitypub.rocks/

Glossary

BeOpen – The name of the system

Instance – An instance of BeOpen is a single deployment of the server software. One organisation should have one instance of BeOpen

Federation – The act of 2 social platform instances “federating” means that the 2 instances trust each other and work together to share/server each of their user’s data. From the user’s point of view it would seem everyone is on the same instance even if it technically 2 separate servers.

Occupation – A type of job or generalisation of roles in the organisation. An occupation is made by an admin or management on BeOpen. Users can assign themselves an occupation and this determines which team they are in.

Team – A group or department of members managed by a manager or head of department. Users in the same team will be able to see each other’s posts

Levels – Different types of users and employees such as Admins, managers, and members

Registration key – A secret phrase provided by an organisation to users to allow them to register an account on the organisation’s BeOpen instance.

Shamir Secret Sharing – An method for sharing a secret in a set of “shares” only a combination of a set number of these shares can be used to find out what the secret is

Encryption – The process of using a “key” to scramble some data, this data can only then be unscrambled through use of that same key (in AES encryption).