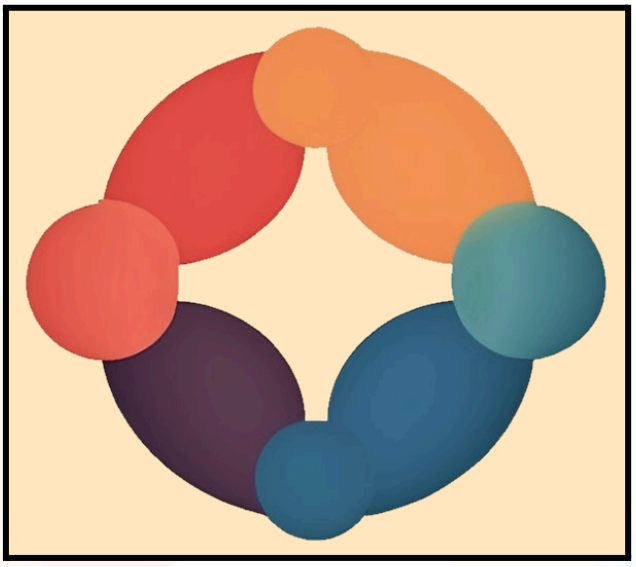
COMP3330 Interactive Mobile Application Design and Programming

Final Report: “HKUnexus”



Group 42

CHAN Yat Lam	3036064480
Benjamin Jun-jie Glover	3035962764
KIM Seho	3036069040
WONG Tang Yik	3036067781
WONG Shom	3036066555
WONG Kwun Yat	3036065915

1st December, 2024

1. Important Note

The code originally hides the Google Maps API key in local.properties so we don't expose the key on our GitHub repository. To make the source code connect to Google Maps API, you may need to put this line in 'local.properties (SDK Location)' under 'Gradle Scripts':
`"API_KEY = "AIzaSyC0bQYBMp7RoExDmHgKy_H98PGjBIjYYJI".`

We attempted to put the API key in the source code before submitting however just to be safe putting this in local.properties would probably be best., however, if any issues arise with connecting with Google Map servers then please place this API key in local.properties.

2. App Overview

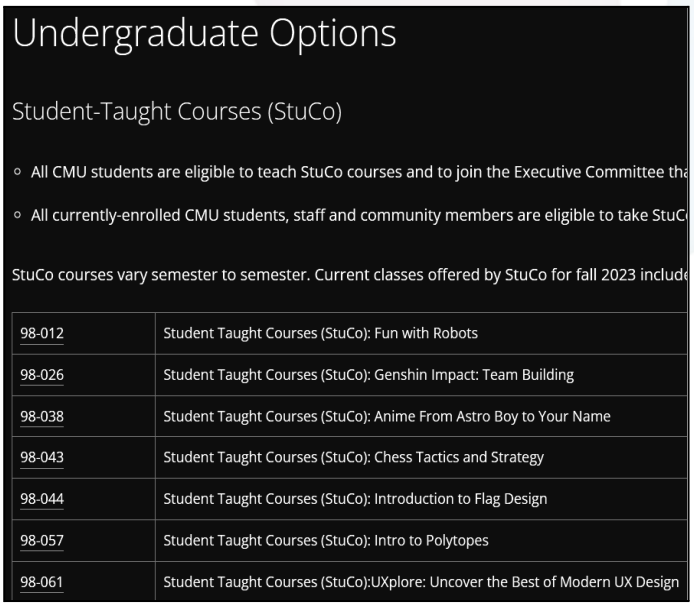
2.1 Main Idea / Goal

When HKU students want to find communities to share their interests with or engage as amateur hobby clubs (e.g. sports), they have the choice of either joining an HKU society or just making their own groups via promoting/asking around on different social platforms.

There may be niche interests or topics that are not covered by HKU societies, so many students choose the second option, but it is difficult to get many people to join your club by just repeatedly spamming on social platforms and hoping that somebody notices, and difficult to efficiently continue the group and manage activities.

Our app is designed to address this problem faced by these "amateur" clubs by serving as a centralised social hub for HKU students to organise their clubs and activities and handle all club-related functions. As a general "group/community-forming" app, it can also be extended to making any groups in general (e.g. study groups, groups for your major, etc).

2.2 Motivation



Undergraduate Options

Student-Taught Courses (StuCo)

- All CMU students are eligible to teach StuCo courses and to join the Executive Committee tha
- All currently-enrolled CMU students, staff and community members are eligible to take StuC

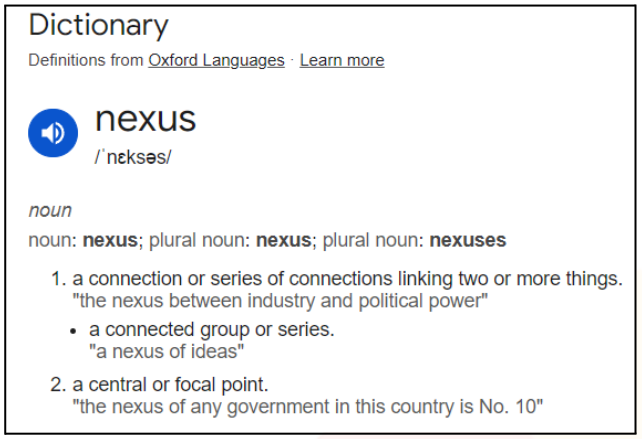
StuCo courses vary semester to semester. Current classes offered by StuCo for fall 2023 include

98-012	Student Taught Courses (StuCo): Fun with Robots
98-026	Student Taught Courses (StuCo): Genshin Impact: Team Building
98-038	Student Taught Courses (StuCo): Anime From Astro Boy to Your Name
98-043	Student Taught Courses (StuCo): Chess Tactics and Strategy
98-044	Student Taught Courses (StuCo): Introduction to Flag Design
98-057	Student Taught Courses (StuCo): Intro to Polytopes
98-061	Student Taught Courses (StuCo):UXplore: Uncover the Best of Modern UX Design

We personally have had our own experiences of facing the aforementioned problems of trying to find societies or groups that we were interested in, and also saw many examples within different platforms (e.g. promotions on WhatsApp and reddit). Seeing all the society stalls around HKU also sparked inspiration about the difficulties of forming hobby groups at HKU.

In addition, we were inspired by other existing resources like Carnegie Mellon University's StuCo, which allows students there to self-design and host their own educational courses of *any topic of their liking* that can be approved for real university credits.

3. Three Key Contributions of our App


Dictionary

Definitions from Oxford Languages · Learn more

nexus

/ˈnɛksəs/

noun

noun: **nexus**; plural noun: **nexus**; plural noun: **nexuses**

1. a connection or series of connections linking two or more things.
   "the nexus between industry and political power"
   - a connected group or series.
     "a nexus of ideas"
2. a central or focal point.
   "the nexus of any government in this country is No. 10"

3.1 Increased Social Connectivity

Students are able to connect over **any** specific interests they have that *are not represented in* any student societies, creating their own diverse communities that can become one of the easiest ways to encourage interaction and collaboration between students from different backgrounds within HKU.

3.2 Flexibility and Freedom

Students are able to form clubs over interests that *may otherwise be considered unsuitable* to be hosted as "societies", from small personal hobby clubs (sports meet, music band, etc) to large groups (e.g. major/course/field-specific clubs) to perhaps niche, trendy or "unprofessional" interests (e.g gaming) - **anytime, anywhere, about anything, for anyone.**

3.3 Centralization and Convenience

Our app provides tools to facilitate aspects of club management and social features - global club promotion and search/explore using categories, intra-club event/announcements, event timing and location, club members management, etc, such that *all clubs-related functionality is in one place/app* and it is convenient for users to manage their clubs and activities.

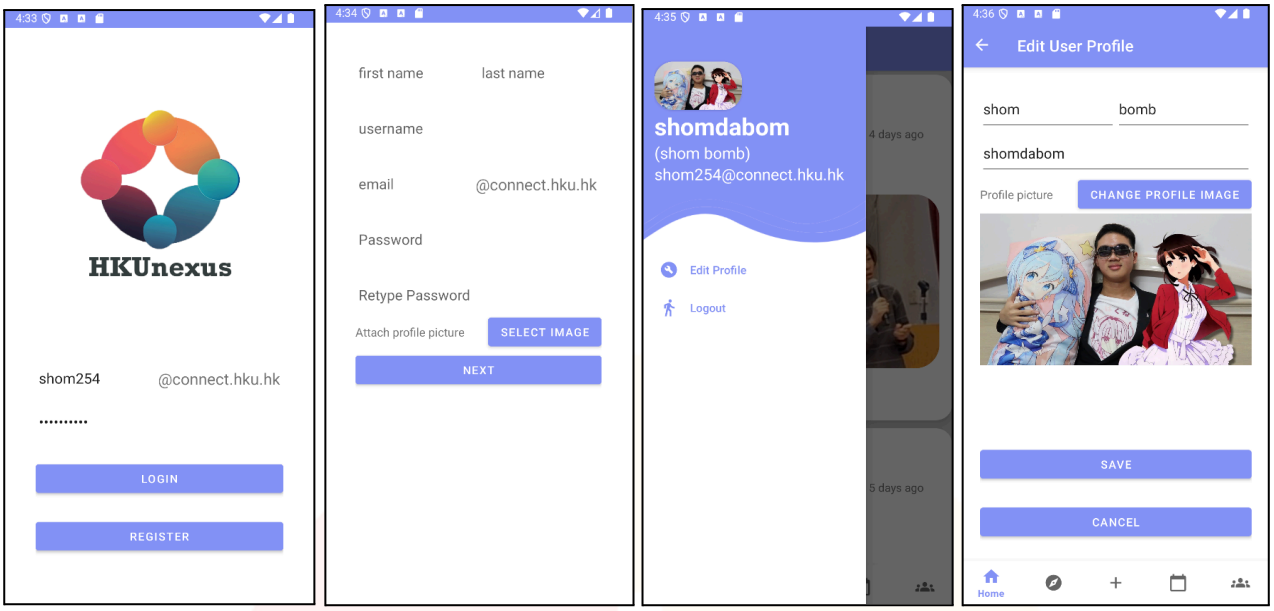
4. App Features

4.1 Main Features

4.1.1 User Accounts

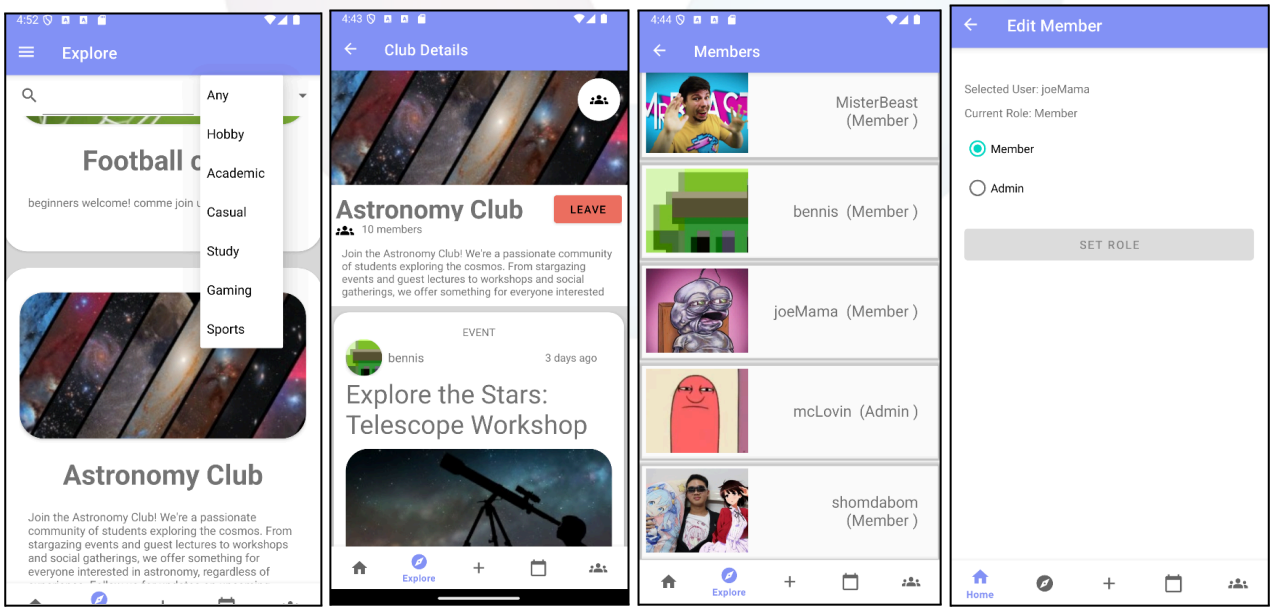
Users create accounts to login/logout and access the app.

- Students register their accounts (limited to HKU students only by restricting the email domain to "@connect.hku.hk") with their names, a username and a password, as well as an optional profile photo
- Registration process is similar to social media: first page for filling in account details, followed by a second page for 6-digit OTP code sent to the email registered by the user to activate the account in the backend.
- The account username and profile photo will be visible throughout the app publicly (e.g. in posts that the user created, or in the group member list)
- All the above user profile settings (name, username, profile image) can be changed in a dedicated "Edit Profile" page



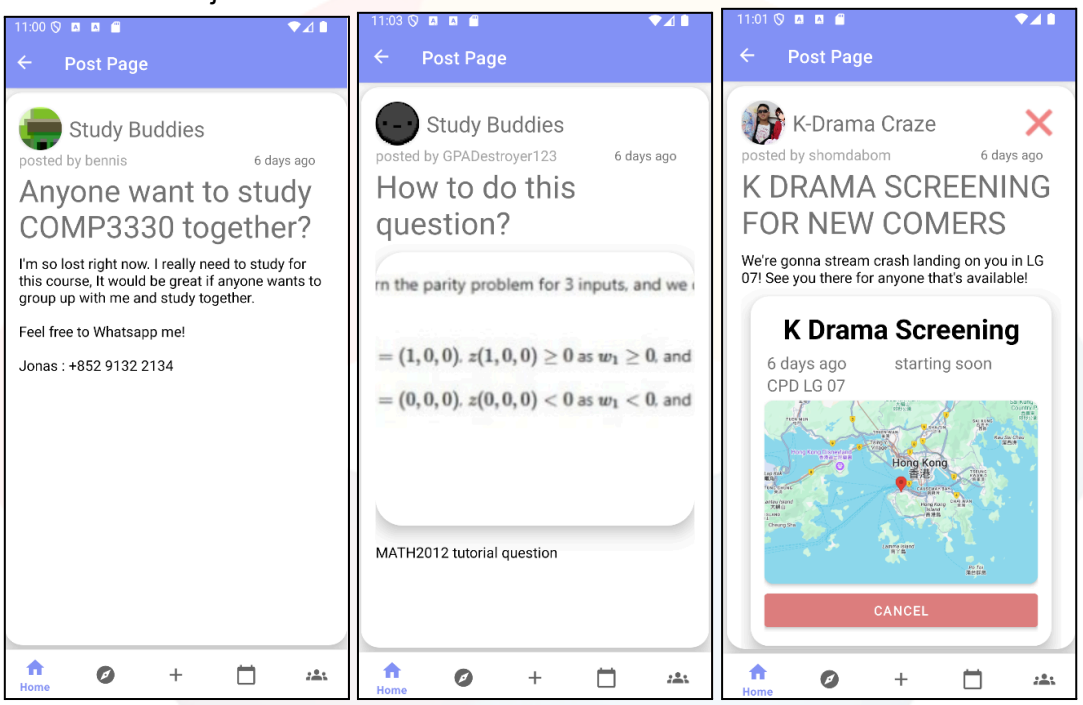
4.1.2 Club Discovery and Details

- Users can search for clubs either by club name (text search) or by categories/tags
- When users tap a club, they will be redirected to the club's landing page, which shows:
    - the name of the club, no. of members, description
    - a join/leave button and a members list button
    - a feed/list of all **posts** (including event posts)
- Users can tap the member list button to see all members and their roles (either "Admin" or "Member")
- Any users who are "Admin" (initially only the original group creator) can further tap on a user in the member list and make them "Admin":



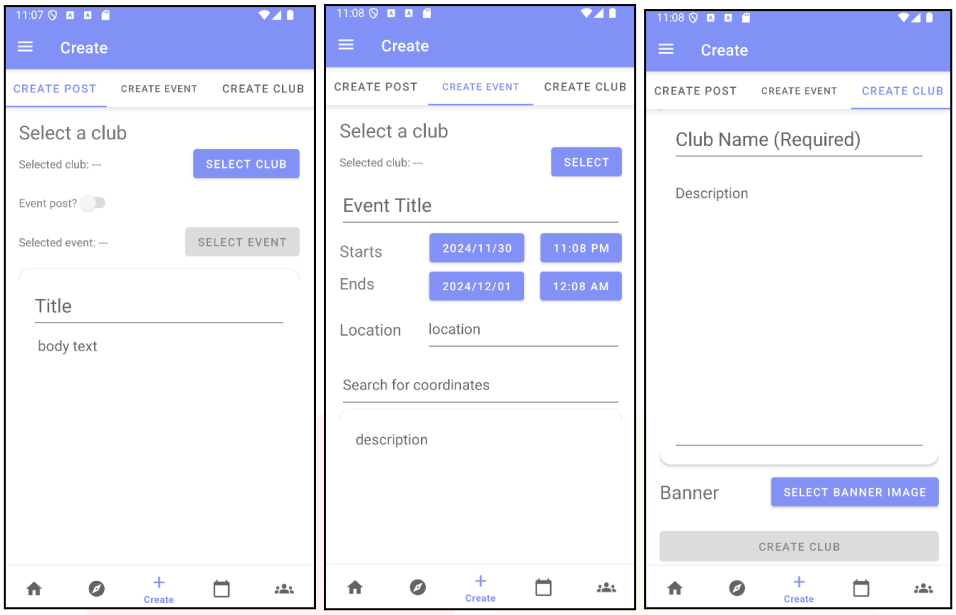
4.1.3 Posts and Events

- Users can tap on any post to see more details (title, description, date posted, user who posted)
- If the post was created by the user themself, a "X" button to allow the user to delete the post will appear
- Posts can optionally contain an image
- Tapping on the club name in the post will direct users to the club landing page
- Posts fall into 2 types:
    - **Regular Posts:** For simple announcements, questions, etc
    - **Event Posts:** Posts/announcements that specifically talk about an event. The event with its details will be embedded within the post as a separate card, and there is a button to join/leave the event.



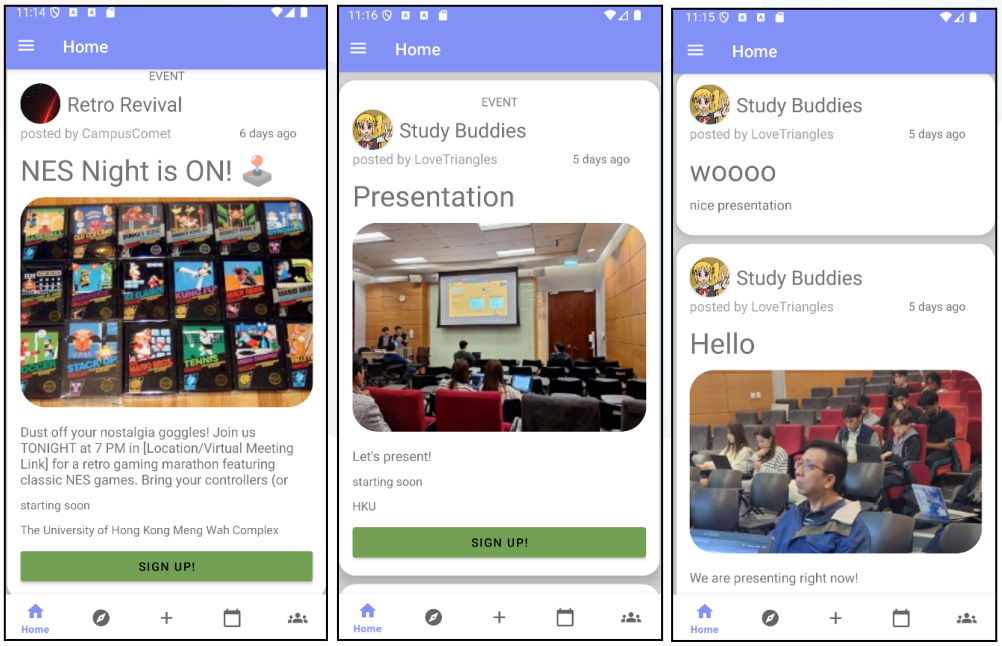
4.1.4 Club/Event/Post Creation

- Users who have been granted admin privileges can at anytime create their own events (for clubs they have joined), posts (for clubs they have joined), and clubs
- Events: title, start and end times, location (name), map coordinates (searchable via Google's Places SDK), description
- Posts:
    - **Regular Posts:** must belong to a club the user has joined, and can have a title, description and optional image
    - **Event Post**: regular post + embed an already created event
- Clubs: name, description, club banner



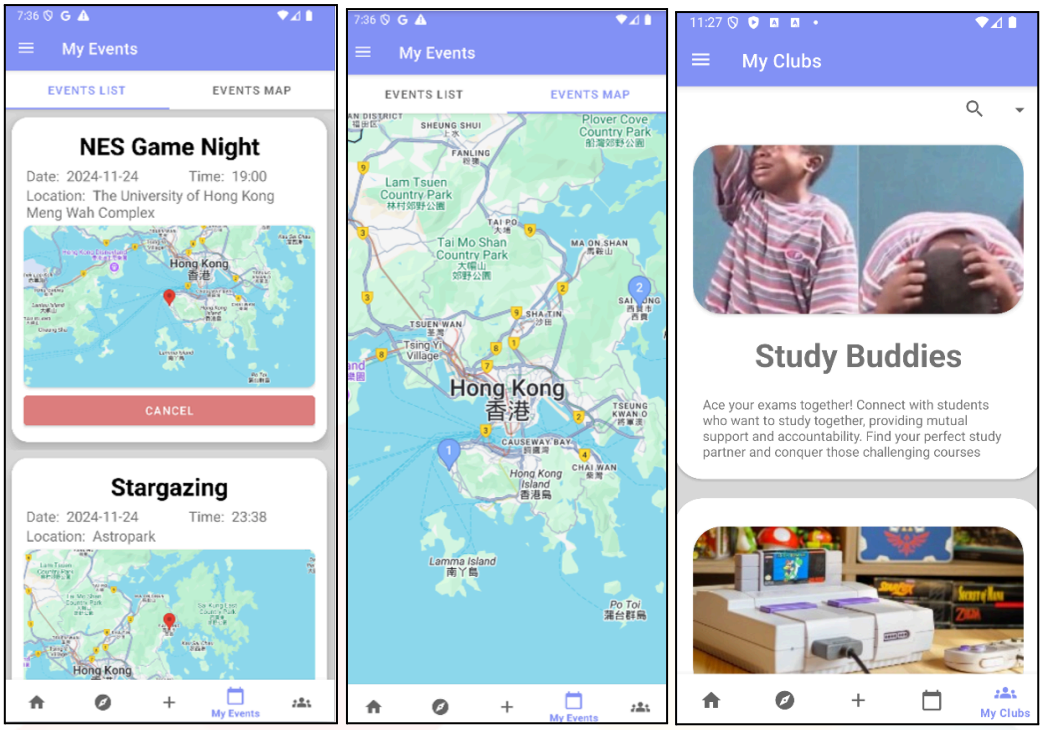
4.1.5 Personalised Home Feed

- Once users have joined clubs, the "Home" page will fetch all posts from their joined clubs (sorted by most recent) like a social-media-style "feed" of cards for brief information
- Event posts will show "**EVENT**" header, as well as event time, location & "JOIN" button



4.1.6 "My Clubs" / "My Events"

- Finally, users can view all their joined clubs and events in dedicated "My Clubs" and "My Events" pages
- "My Events" can show a list of all events at a glance, with option to leave the event using the button, or users can view a Map of all the events with markers of event locations

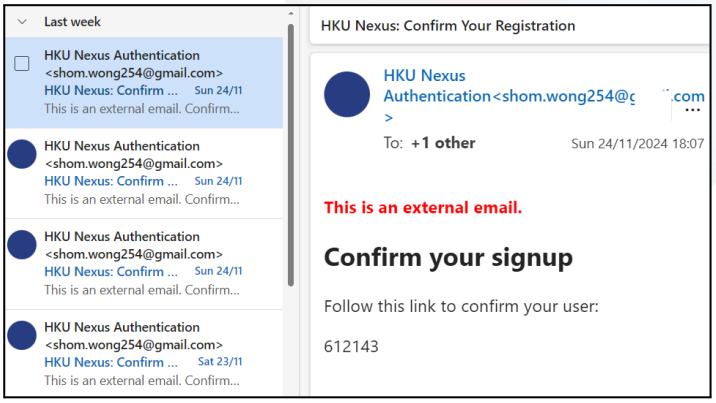


4.2 Technical Details

4.2.1 Backend Integration (Supabase)

We have used **Supabase**, an online Backend-as-a-Service platform that provides authentication, database and object storage for our app with direct access to the database using their own Kotlin SDK/HTTP Client, skipping the need for us to build a separate backend server.

1. User Accounts (Authentication)



We were able to set up a custom SMTP server using one of our member's own gmail accounts to be able to send OTP codes to verify accounts during registration.

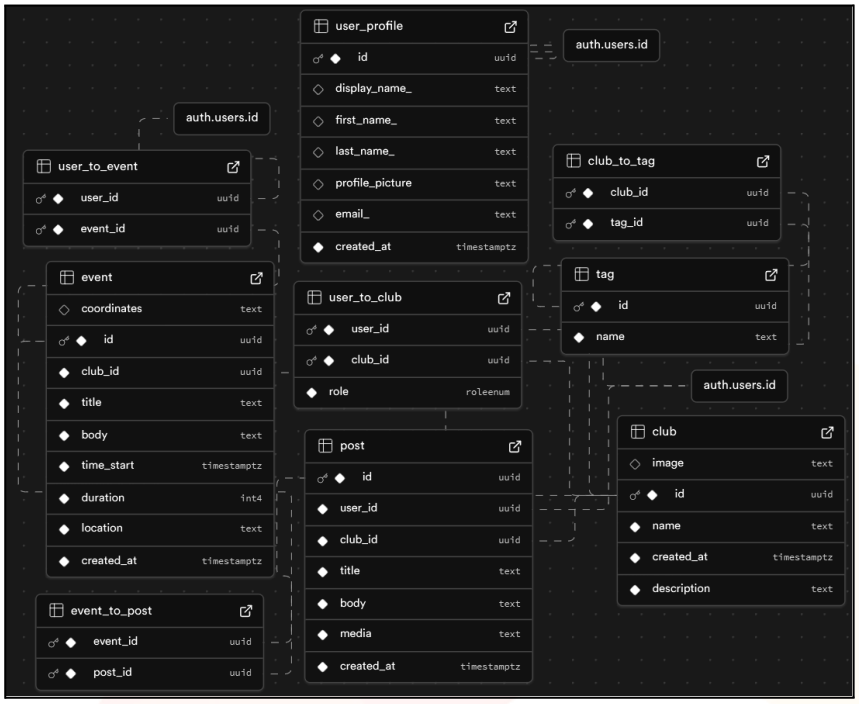
On the backend/database side, user authentication, credentials & session management is fully handled by Supabase's authentication module. We just had to call their APIs from our Kotlin code.

2. Database (Schema Design, SQL and RPC)

We used a PostgreSQL database to store app data, as the data was highly structured and we were familiar with relational databases.

The image here shows our complete database schema for app data, including all the entities our app uses (clubs, events, posts, tags) and their relations.

Users are assigned either a "Member" or "Admin" role which determines what he/she can or cannot do in that club (e.g. Creating posts, Deleting posts, Assigning members' permissions).



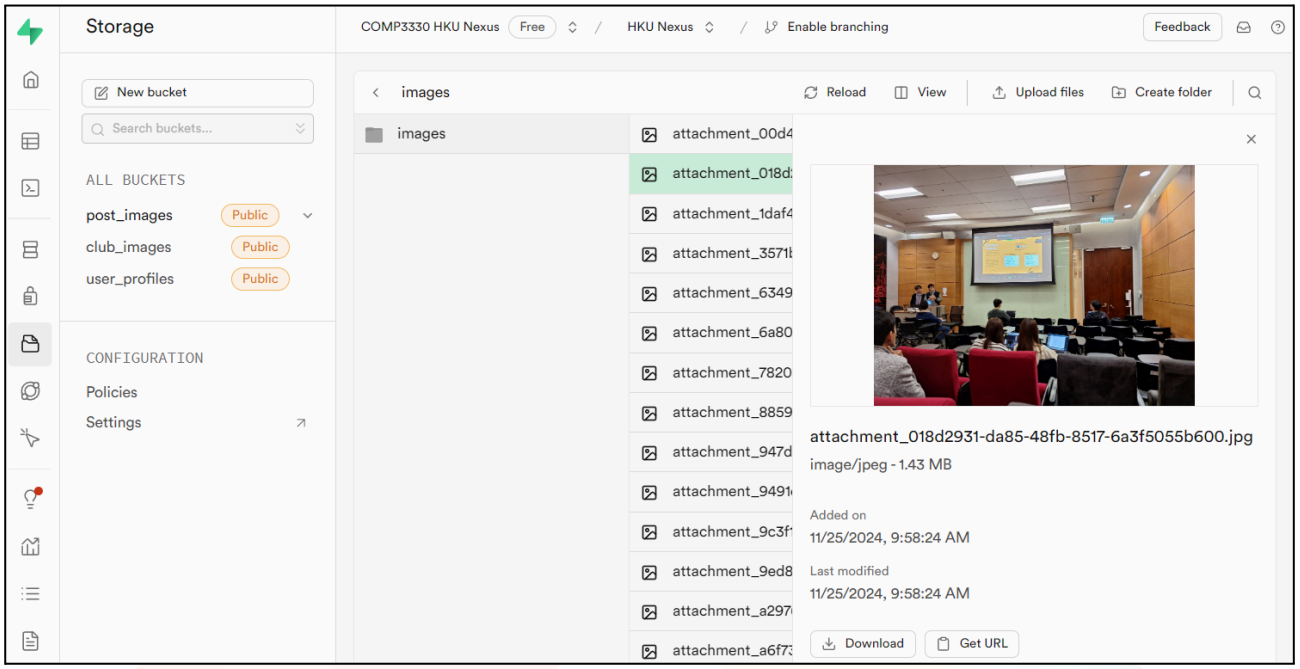
Instead of HTTP REST APIs, all the requests and queries to the database are done via RPC (remote procedure calls). We simply define database functions/procedures in SQL as direct endpoints and call them from our code with JSON and deserialize results into Kotlin classes.

As an example, here is the Kotlin RPC function `insertOrUpdateClub()` for calling `insert_or_update_club` (an SQL function in our database) and deserializing the result.

```kotlin
fun insertOrUpdateClub(
    idArg: String,
    nameArg: String,
    descriptionArg: String,
    imageArg: String
): ClubDto? {
    return runBlocking {
        val currentDateTime =
OffsetDateTime.now(ZoneOffset.UTC)
        val funcName = "insert_or_update_club"
        val funcParam = buildJsonObject {
            put("id_arg", idArg)
            put("name_arg", nameArg)
            put("description_arg", descriptionArg)
            put("image_arg", imageArg)
        }
        try {
            val result =
client!!.postgrest.rpc(funcName, funcParam)
            val output = result.decodeSingle<ClubDto>()
            return@runBlocking output
        } catch (e: Exception) {
            return@runBlocking null
        }
    }
}
```

```sql
create or replace function
insert_or_update_club(
    id_arg uuid,
    name_arg text,
    description_arg text,
    image_arg text
    )
returns setof club
language sql
as $$
insert into club (id, created_at, name,
description, image) values (id_arg, NOW(),
name_arg, description_arg, image_arg)
on conflict (id) do update set
created_at = created_at_arg,
name = name_arg,
description = description_arg,
image = image_arg;
```

3. Object Storage (Buckets)



For post images, club banners, & user profile pictures, we utilized Supabase's object storage service. Each type of media was separated into its own bucket, and images were named using the `post_id`, `user_id`, or `club_id` UUIDs from the database record they were associated with for easy fetching. We then utilised Glide on the app side to render these images using a HTTP GET request to the bucket.

4.2.2. Google Maps API

Finding exact locations can be difficult for users when planning or going to events. To better facilitate face-to-face meetings and minimize the time spent on finding your events, the app utilizes the Google Maps API to display event location.

1. Static Map

The app uses a static map displayed by a custom `'TouchImageView` rather than a traditional dynamic map. This was done to expand usability of the maps feature to lower-spec phones who may find the traditional dynamic map to be too computationally demanding. This loading changes the overhead from computational overhead to networking overhead as images need to be loaded from the Google servers. We find this tradeoff to be worthwhile as phones generally can load one image better than loading and constantly updating a live map in real-life scenarios.

Static maps are just images requested from the Google API by forming a link. Images are fine-tuned using parameters that are put into the URL. These parameters include the zoom in amount, image size, and type of map. The parameters with the most customization done are the markers which mark the actual event locations. These locations are obtained from the coordinates field where the location name from Places SDK is submitted.

```
if (eventCoordinates != null){
    val zoom = 11
    val size = "1080x1080"
    val mapType = "roadmap"
    val marker1 = "color:red|$eventCoordinates"
    val apiKey = BuildConfig.API_KEY // Replace with your actual API key
    val url = "https://maps.googleapis.com/maps/api/staticmap?center=$eventCoordinates&zoom=" +
            "$zoom&size=$size&maptype=$mapType&markers=$marker1&key=$apiKey"
```

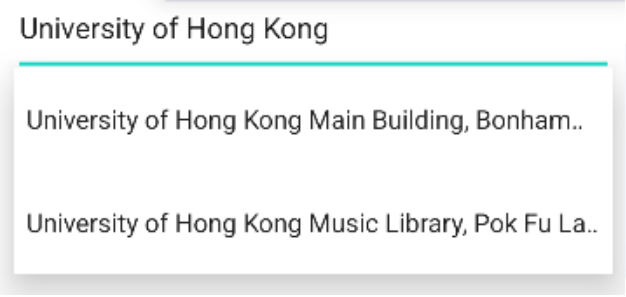
The markers for the 'Events Map' fragment are numbered and ordered in terms of how close they are to the event date. (Note: This ordering is done through the backend as the events are queried by the event's starting time in ascending order.)

These static images are then put into the `TouchImageView` which allows for panning and zooming of images and are a subset of ImageView. `TouchImageView` does not work with these map images traditionally, however, changing them to the Bitmap type allows for some control of images (this is a known bug).

This combination of `TouchImageView` and map images gives us a convincing map that allows for some zoom and panning abilities without the large performance overhead.

2. Places SDK

To facilitate location searching for marker placement, Places SDK autocomplete was used when creating an event to allow users to place markers at specific locations recognized by the Places SDK Autocomplete API by Google. The user will first type their desired location and the activity will attempt to predict the location and give several options for the user to choose from.



If the location is a common location (eg. 'Sai Kung') or is chosen from the list of predicted locations, then the marker will be shown on the map with no issue.

(To the right: An example of the autocomplete feature)

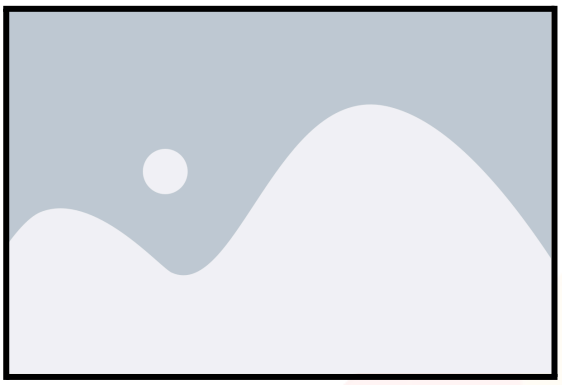
4.2.3. Glide Image Library

The Glide API is exclusively used as the interface for image handling for our posts and map display. It is an all-in-one solution for image handling that allows for efficient caching and on-the-fly image modification.

1. Fetching and Caching

Glide has features to load images from URLs through network loading, allowing the server to manage images and requests to our database with using links to manage table queries faster. After loading images, they are immediately cached on memory and disk. Whenever these images are needed to be loaded again, whether from accessing different pages or refreshing a page, they will be checked in memory first, then the disk, then finally will be loaded again from the source.

2. Image Modification (Bitmap Conversion/Resizing)



Glide allows for images to be displayed in unique ways before and after loading. Before loading, the application can place an image placeholder and also show an error image for invalid URLs after loading. ← We have chosen the following image on the left as the placeholder error image and loading image.

Images can also be transformed into different types to fit the developer's experience. We have chosen to transform it to a bitmap to better fit our Maps API as there are some issues with maps not displaying as another type due to the internal workings of TouchImageView. To overcome this issue, the image has to be transformed into a Bitmap then set as the image's bitmap

```
Glide.with(viewHolder.eventMap.context)
                .asBitmap()
                .load(url)
                .error(R.drawable.placeholder_view_vector) // Placeholder in case of error
                .into(object : SimpleTarget<Bitmap>() {
                override fun onResourceReady(resource: Bitmap, transition: Transition<in Bitmap>?) {
                        viewHolder.eventMap.setImageBitmap(resource)
                }

                override fun onLoadFailed(errorDrawable: Drawable?) {
                        super.onLoadFailed(errorDrawable)
                        viewHolder.eventMap.setImageDrawable(errorDrawable)
                }
                })
```

Legend:
Custom settings and parameters for image loading
Specifying how to load the image into the TouchImageView

5. Testing and Issue Analysis

Overall, the app is perfectly functional and all of the core features included in the proposal can be demonstrated. By the nature of the app, the major bottleneck comes down to the latency with each database call. We have employed several design considerations in order to mitigate the effects, such as cutting unnecessary API calls from the front end and using asynchronous methods for media fetching operations (rendering images).

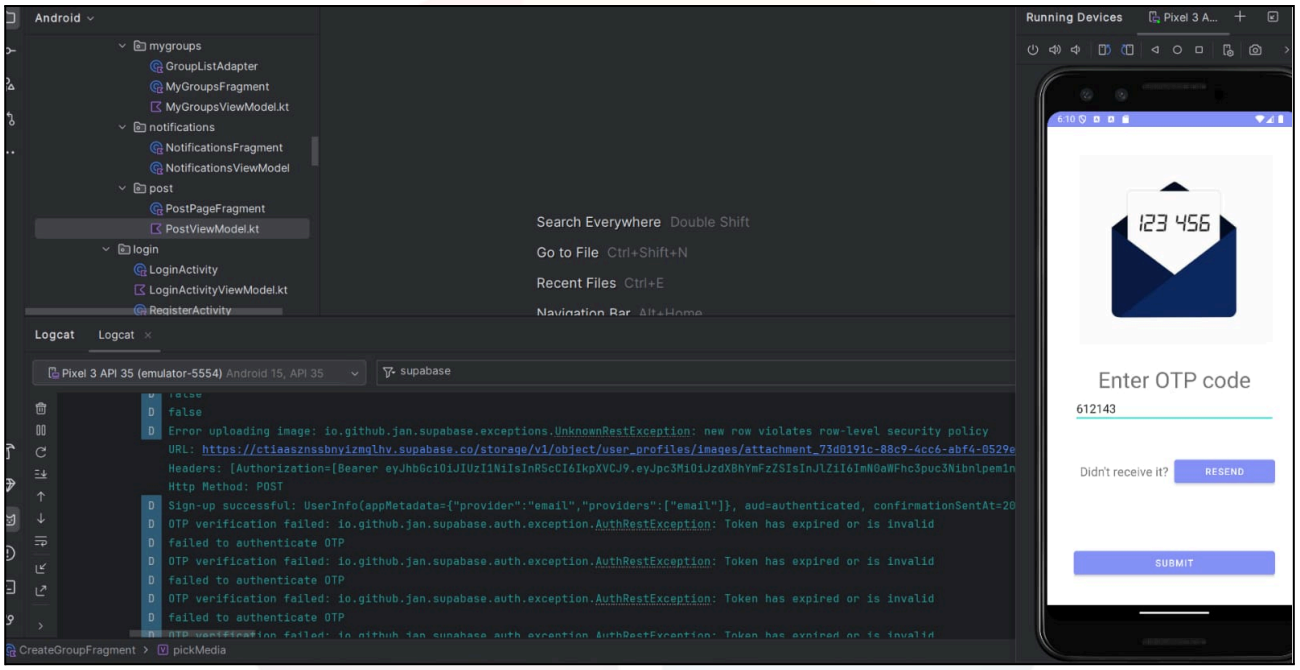
Provided the brief timeline of the project, the quality of the app is sufficient to be a prototype but not as polished to be a product ready for release.

- There is little error-handling for unexpected UI/UX interactions, such as submitting forms twice or performing actions before the completion of database methods.
- In the occurrence of errors, it would often crash and the app would close itself.
    - Fortunately, the bugs are rarely fatal and users can continue using the app after relogging in.
    - These exceptions can be entirely mitigated if one had put those checks in place (e.g. disabling buttons while fetching) or synchronising with the database after particular operations.

Additionally, in order for this type of social media application to be practically realized, it needs to handle the traffic for all its users. Assuming the app only serves the student community of a single university, the number of registered users of the platform could be in the range of hundreds if not a

few thousands, and up to an average of 10~25% users to be active at a time. With our limited resources, we have yet to conduct any tests about dealing with traffic of those magnitudes. It is not known whether our current architecture is reasonably optimized and practical under full-scale operation, or whether there are better ways of implementation that can achieve a similar vision.

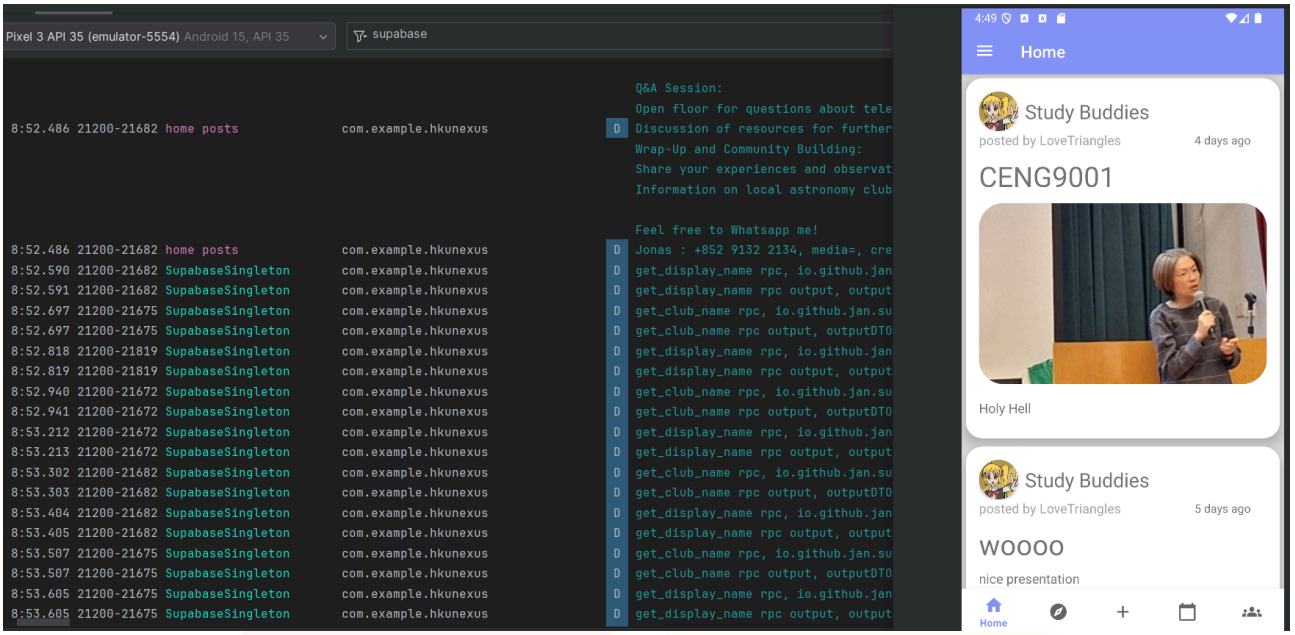
Going back to the development side, we did not have time nor saw the need for using testing tools on a small-scale app/course project. Instead, we used Android Studio's "Logcat" logging command-line tool extensively throughout our code (especially for fetching from supabase). Logcat supports log "tags" and multiple types of log entries which can prove useful for debugging.
For example, we can tag logs with "SupabaseSingleton" or "Glide" to show which related functionality is being logged, and we use `Log.DEBUG` and `Log.ERROR` in try-catch blocks to easily spot any fatal client-side errors.



Demonstration of using Android Studio's "Logcat" command line tool for logging
In this case, we are debugging the errors experienced during registration (OTP stage)
(The error was that the account had already existed within the database so the token was invalid)

6. Limitations and Remaining Issues

As mentioned earlier, the bottleneck in our app is fetching lots of data from the database. Unfortunately, as the complexity of our app's functions means we spent a lot of time implementing them and integrating them together, we did not have time or prioritized some caching solutions for posts. As a result, every time we loaded a page, the entire set of required data would be fetched, leading to some visible latency.



Although the UI/UX can achieve all the functionalities, it is rather crude at times and can be problematic in a practical setting. For example, the lack of filtering or searching functions when traversing a list of clubs or users. Those UI elements can be cumbersome to navigate once the number of items are over 20 or 30.

7. Future Plan

Indeed, as mentioned earlier, 1.5 months of work is definitely not enough to build a full scale app with every single one of our intended functions, but we are still very happy with the results achieved, since we were still able to implement the core functionality needed to demonstrate the intention and proof-of-concept of our app, especially considering the complexity of the app (a fully functioning social-media style app frontend *and* backend).

Therefore, this section will simply list features/enhancements we were not able to implement due to complexity and time constraints, and ones we intend to see actually implemented if this was a complete and marketable app.

7.1 Comments and Chat

It is crucial for a social media app to facilitate interactions among members. And there are prior real-life examples that the removal of a single core interactive feature can severely impact user experience and the platform as a whole (e.g. The removal of dislike in Youtube). That is why the most important feature that we would have added is Post Interactions.

Users can freely make comments under any posts, which can first allow users to express their sentiment of a posts as well as viewing the sentiments from others, secondly, it can serve as an area of back-to-back conversations, it can be from OP(original posts) to members or just between a group of members.

Any kind of rating system of posts shall also be considered. In addition to assigning a reputation value for user's reference, it can be integrated into the post feed system. Posts can be filtered or sorted by their reputation so users can browse the content that worth their attention

7.2 Notifications

Several functionalities of our app can be boosted by integrating with mobile notifications.

Since our app deals with event organization and management, it is reasonable to remind users when an event starts. For the hosts of an event, it is useful to give them notifications when someone has joined their event or a particular threshold of participant has reached. These information can help hosts to make changes to an activity when necessary

Notifications shall also integrate with posts and the interactions of posts (e.g. comments).

7.3 Other quality-of-life improvements

7.3.1 Calendar Integration

Similar to how events can be mapped on a single map, it is also useful to map all joined events on a single calendar for user's reference. There can also be functionality to export the events directly to the user's phone calendar or Google calendar, alongside all the other details.

7.3.2 Google Maps

The current maps implementation is rather limited as it is a free API. Google maps would have provided much better map accuracy and dynamic map navigation. Pairing location tracking with Google Maps would have been ideal to allow for users to find events on their own.

7.3.3 User Feedback

Adding a channel for users feedback about the app is useful for developers in maintaining the app and evolving the app in a positive way.

7.3.4 More supported types for posts

Currently, only single images are supported for posts, which are limiting the contents that can be posted to the app. Support for more media types, such as image gallery, videos and embedded links, can cater to different needs of users, and also enhance the communication between users.

Furthermore, new post types (such as polls) can expand the functionality of posts and allow users to interact within the platform.

7.3.5 Home page personalization/filter and sort

Home page is the main page that the user will get all the post information from, so it is important for the user to personalize their home page.
A filter and sorting function allows users to quickly select posts from their feed, enabling quicker access to relevant content and efficient navigation through their posts.

7.3.6 Custom roles for flexibility (self organisation)

Roles: A fully customizable role system where admins can set the right individually for each user or a group of users. Similar to the solution offered by Discord (An instant chat messaging app/platform).

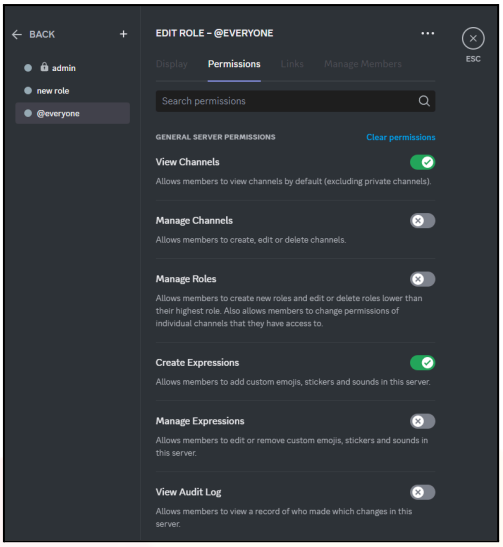


Figure : The role setting user interface from Discord