Milestone III Submission

Team Name:

Bora Papa

Proposed Level:

Apollo 11

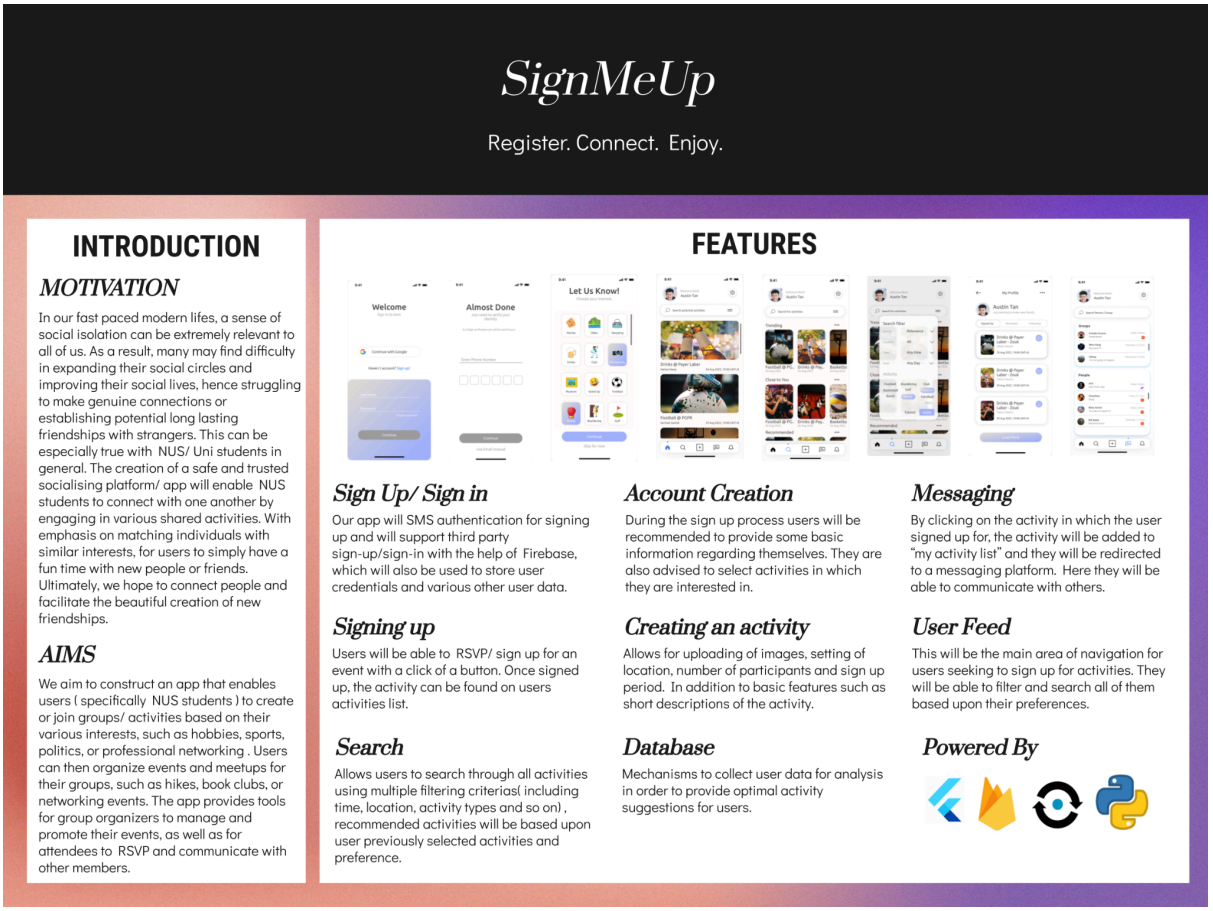
Deployment:

Refer to the attached apk file: Our Current App

Email: user1@gmail.com

Password: P@ssword123

Project Poster:



Motivation:

In our fast-paced modern lives, a sense of social isolation has become increasingly relevant to people from all walks of life. The advancement of technology and the prevalence of social media paradoxically have made it more challenging for individuals to form genuine connections and expand their social circles. This struggle is particularly true for university students, including those at the National University of Singapore (NUS), who often face the pressures of academic demands and the transition to a new environment. Recognizing this issue,

the creation of a safe and trusted social platform or app specifically tailored for NUS students can play a crucial role in addressing social isolation and fostering meaningful connections.

Our proposed social platform/app aims to bridge the gap between students, allowing them to connect with one another and engage in various shared activities. By focusing on matching individuals with similar interests, our platform provides a space for users to have a fun time and potentially form long-lasting friendships with like-minded peers. Through this platform, NUS students will have the opportunity to expand their social networks, discover new hobbies, and engage in meaningful conversations.

Moving away from NUS students, our socialising platform/app can further serve as a valuable resource for incoming freshmens or exchange students. It can facilitate the integration of students into the university community, enabling them to meet people from various backgrounds and forge connections based on shared interests. This can be particularly beneficial for international students who may face additional challenges in adapting to a new culture and making friends.

In conclusion, the development of our proposed platform for NUS students has the potential to address the prevalent issue of social isolation and enhance the social lives of university students. By leveraging technology and emphasising the importance of shared interests, our platform can facilitate connections among NUS students, leading to the creation of new friendships and a sense of community. As we navigate the challenges of our fast-paced modern lives, fostering genuine connections and meaningful relationships remains crucial, and we believe our platform can play a significant role in achieving that goal.

Vision:

One of the key benefits of our platform is the emphasis on creating a *safe and trusted environment*. Online safety is paramount, especially when interacting with strangers. The platform will implement robust security measures, such as *user verification*, to ensure the safety and well-being of its users. This will help establish a sense of trust among NUS students, encouraging them to actively participate and connect with others.

Additionally, our socializing platform/app will offer a wide range of features to enhance the user experience. Including *interest-based groups, event listings,* and hopefully even *chat functions*. By providing a diverse range of activities and opportunities for interaction, our platform aims to cater to different preferences and ensure that users find it enjoyable and engaging.

User Stories:

1. Creating and joining activities: This feature allows users to create their own activities and post them on the platform for others to see. Users can indicate their interests and preferences, such as hobbies, sports, or social events, to join a wide range of activities posted by other users. The activity space is an exploratory forum-like interface where users can browse and discover various activities that align with their interests.
2. Event scheduling and RSVP: Activity organisers have the ability to schedule specific events within the platform. They can set a date, time, and location for the event, providing all the necessary details. Members who are interested in attending can RSVP to indicate their intention to participate in the event.
3. Event notifications: Users can opt-in to receive notifications regarding new activities that match their interests. When a new activity is announced, users who have indicated their interest in similar activities will receive a notification, ensuring that they are informed about relevant opportunities. Additionally, users will receive notifications about upcoming activities they have RSVP'd to, reminding them of the event and any important updates.
4. Messaging: This feature provides a platform for users to communicate with each other. It facilitates event planning and coordination between participants. Users can exchange messages to discuss event details, coordinate logistics, or simply interact with others who share similar interests. It also serves as a means to consolidate friendships and maintain connections with people they have met through shared activities.
5. Account creation: To access the platform, users need to create an account. The account creation process may involve user authentication via SMS (Short Message Service) for verification purposes. Alternatively, users can sign up or log in using various third-party authentication services, such as Google or Facebook, to streamline the registration process.
6. Member profiles: Each user has the option to create a personal profile on the platform. This profile includes a personal bio or description, a profile picture, and areas of interest or activities they enjoy. Member profiles provide a snapshot of each user's background, preferences, and personality, making it easier for others to find like-minded individuals and establish connections.
7. Search: The search feature enables users to find specific activities or events based on their preferences. Users can search and filter through activities using various criteria such as location, time, interest, or other relevant filters.
8. User feed: The user feed is a customised feed that displays activities deemed of interest to the user. An algorithm analyses the user's preferences, past activities, and interactions on the platform to curate a personalised feed. This feed showcases activities, events, or updates that are most likely to appeal to the user, enhancing their browsing experience and increasing the likelihood of finding relevant and engaging activities.

Software Engineering Practices:

MVC Architecture Pattern:

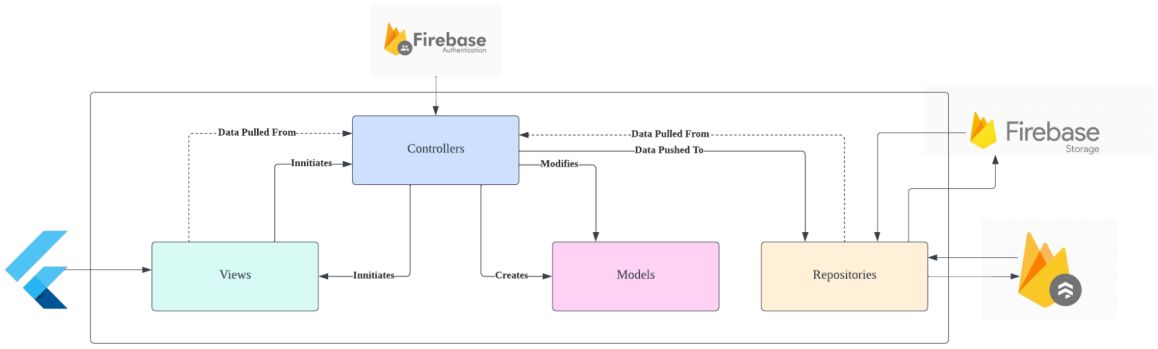
Our application follows the Model - View - Controller architecture. In which there exists a separation between software logic and code handling for visual displays.

The model in our application represents the data and the business logic of our application. It encapsulates the data from API calls, thus allowing for methods to manipulate and access them. In our case for example, the UserModel class is defined to represent the user data. It encapsulates properties such as name, username, email, password, profile image, description, and other related information. The UserModel acts as a data container and provides methods for serialisation and deserialization (toJson and fromSnapShot) to interact with Firestore. Essentially, the UserModel serves as the central entity to handle user-related operations and data.

The view is responsible for the user interface (UI) presentation of our application. It encompasses the visual elements that the users interact with. All the various widgets are used to build the UI components of different screens such as the SignUpForm, SignInForm, UserDashboardhomescreen, and others. These widgets define the layout, styling, and input fields to capture user data. The view is responsible for displaying the data from the model and receiving user input, and it does not contain any business logic.

Lastly, in our application, the controller acts as the intermediary between the model and the view. It handles user input, updates the model based on user actions, and synchronises the changes with the view. Controllers such as SignUpController, SignInController, UpdateProfileController, and others are used to manage the state and perform actions related to their respective screens. They handle user interactions, validate user input using the ValidationController, and communicate with the authentication,user, and event repositories for data operations. The controllers ensure that the model and view are in sync and facilitate the flow of data and actions within the application.

To handle the API calls, repositories are created within our application to better manage the handling of these data operations. By separating the data operations into repositories, our codes follow the principle of separation of concerns and maintain a clear distinction between data handling and other application logic.



Software Architecture Diagram

Application Features:

Splash Screen: [Completed]

User Stories:

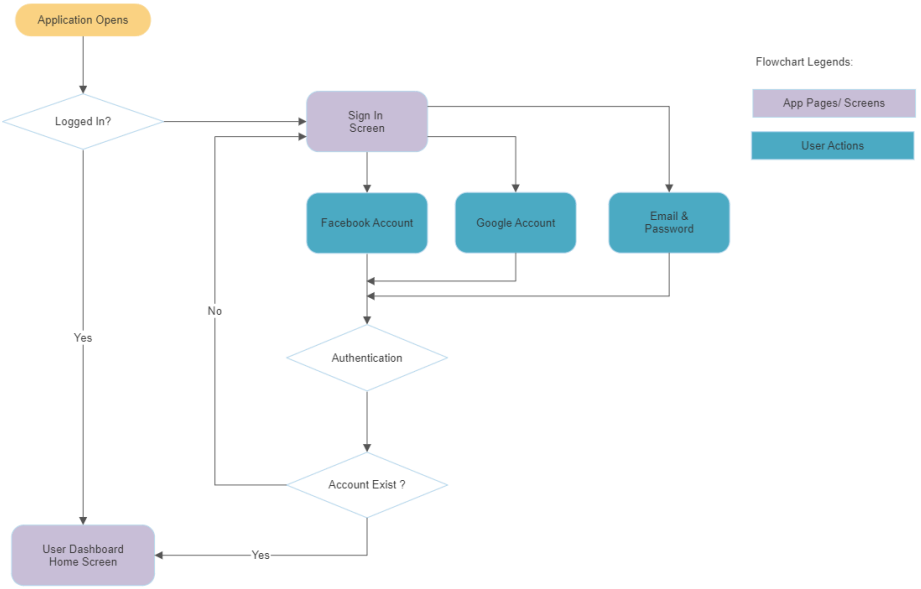
- As a user, I want to see a visually appealing splash screen when launching the application, creating a positive first impression and setting the tone for the app's design.
- As a user, I expect the splash screen to load quickly and efficiently, minimising any waiting time and providing a smooth transition to the main app interface.
- As a user, I appreciate seeing the app's logo or branding on the splash screen, helping me quickly identify and recognize the application.

Proposed Feature Description:

The splash screen serves as an introduction to our app, providing users with immediate feedback that the app is loading. It helps reduce perceived wait times and creates a seamless transition into our main app interface. While the splash screen is visible, our app is working in the background to load necessary data, set up services, and prepare the user interface.

By incorporating a splash screen, we enhance the user experience, giving our app a professional and polished look. Our soon to be carefully designed app logo displayed during the splash screen reinforces our brand identity and sets the visual tone for the rest of the app.

User Authentication: [Ongoing - Facebook Remaining]



User Authentication/ Sign In Flowchart

User Stories:

- As a user, I want to be able to sign in to the application using my email and password so that I can access my personalised account and data.
- As a user, I want the sign-in process to be secure and protected, ensuring that my login credentials are encrypted and kept confidential.
- As a user, I want to receive clear and informative error messages if I enter incorrect login credentials, helping me troubleshoot and resolve any issues.
- As a user, I want the sign-in process to be fast and efficient, allowing me to access my account without unnecessary delays or complications.
- As a user, I want the option to stay signed in even after closing and reopening the application, so that I don't have to repeatedly enter my login credentials.
- As a user, I want the sign-in form to be user-friendly and intuitive, with clear labels and input fields for my email and password.
- As a user, I want the sign-in process to support alternative login methods, such as signing in with Google or Facebook, to provide convenience and flexibility.
- As a user, I want to receive confirmation or feedback after successfully signing in, reassuring me that I am now authenticated and can access my account.

Proposed Feature Description:

Our proposed app will allows for 3 main sign in methods.

- Utilisation of a Google Account **[Completed]**
- Utilisation of a Facebook Account **[Completed]**
- The Standard Email & Password **[Completed]**

Our app will offer three main sign-in methods: Google Account, Facebook Account, and the standard Email & Password. The Google Account and Email & Password authentication methods have already been completed. We are currently working on integrating the Facebook Account sign-in method to provide users with more options and convenience.

Upon successful login, the user will be routed to the home page, where they can access the app's features and content. To ensure smooth user interactions, we have implemented appropriate error handling tools. These tools provide meaningful error messages and guidance to users in case of login failures or other authentication issues.

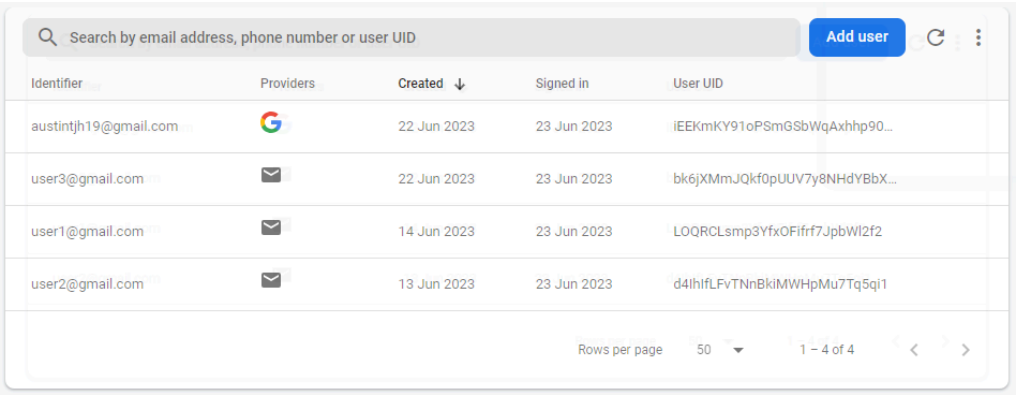
Proposed Feature Implementation Philosophy:

To handle the authentication process, we are leveraging the Firebase Authentication SDK. This SDK provides a reliable and secure way to authenticate users and manage their login credentials. It simplifies the authentication process, allowing us to focus on delivering a smooth and seamless user experience.

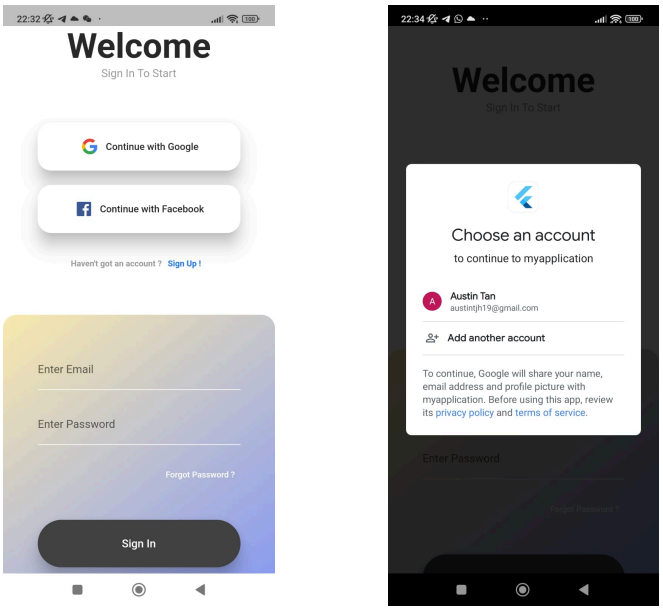
When the user opens the sign-in screen, they are presented with the option to enter their email and password. The SignInForm widget handles the input fields and validation. Upon clicking the "Sign In" button, the SignInController is invoked, and it accesses the AuthenticationRepository to initiate the sign-in process. The SignInController retrieves the email and password from the respective TextEditingController instances. It then calls the signInUserViaEmailAndPassword method from the AuthenticationRepository, passing the email and password as arguments. This method uses Firebase authentication to sign in the user with the provided email and password.

Firebase authentication handles the verification of the user's credentials and generates a unique user ID token. If the sign-in is successful, the user remains logged in until they explicitly sign out or their session expires. Firebase keeps the user logged in even if they exit the app and relaunch it, thanks to the persistence of the authentication state. The AuthenticationRepository acts as an intermediary between the controllers and Firebase authentication. It uses the Firebase authentication APIs to handle sign-in requests. The user accounts will be stored in the registered Firebase Authentication database associated with our app.

For signing in with Google, the tapping of its button triggers the signInWithGoogle method in the AuthenticationRepository. The Firebase authentication APIs for Google sign-in are then utilised to display a Google sign-in interface to the user. The user can then choose their Google account and grant permission to sign in. Once the sign-in process is successful, Firebase generates a unique user ID token and handles the authentication process. The user remains logged in using their Google credentials until they explicitly sign out or their session expires.

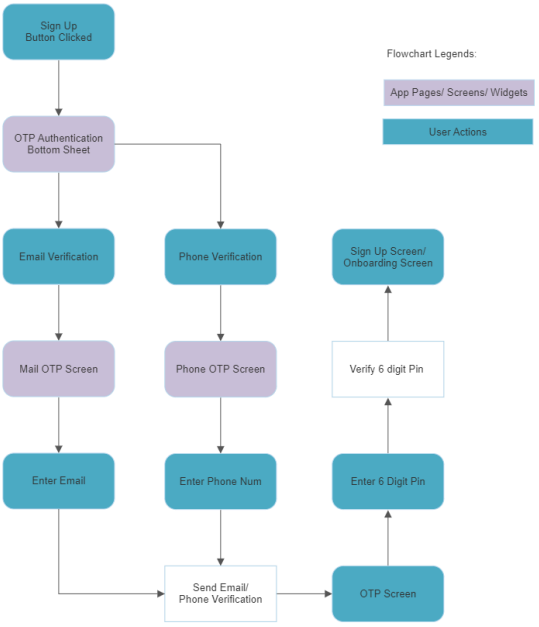


Firebase Authentication database



User Authentication/ User Sign In Screen

Phone/ Email OTP Authentication: [Ongoing Phone OTP]



Phone/ Email OTP Authentication Flowchart

User Stories:

- As a user, I want to receive a verification code via SMS or email when signing up for an account to ensure the security of my account and protect against unauthorised access.
- As a user, I expect the verification code to be delivered quickly and reliably, allowing me to complete the sign-up process without significant delays.
- As a user, I want the option to choose between receiving the verification code via SMS or email, depending on my preference and convenience.
- As a user, I expect the verification code to be valid for a sufficient amount of time, giving me enough time to enter it accurately and complete the sign-up process.
- As a user, I appreciate clear instructions on where to enter the verification code during the sign-up process, ensuring that I provide the code correctly and without confusion.
- As a user, I appreciate clear and concise error messages if there are any issues with the verification process, allowing me to understand and resolve the problem effectively.

Proposed Feature Description:

This feature will only be deployed when new users wish to create an account using the standard email and password. A pop-up option will be provided to the users to select their preferred authentication method.

When users provide the necessary details for account creation, a 6-digit pin will be sent to their mobile number or email address. This pin serves as a one-time password that users need to enter to authenticate themselves. Once the authentication is complete, users can proceed with the onboarding process.

Proposed Feature Implementation Philosophy:

In terms of implementation philosophy, for email OTP authentication:

- The AuthenticationRepository class handles email authentication operations using Firebase Authentication. It uses the EmailOTP package to send and verify OTPs via email.

- The emailAuthentication method in the repository sets up the email OTP configuration and sends the OTP to the user's email.
- The MailOTPScreen widget represents the screen where the user enters their email for OTP verification. It displays an image, text inputs, and a form for submitting the OTP.
- The MailOTPForm widget contains the logic for handling the OTP form submission. It uses the OTPController to call the verifyOTP method in the repository to verify the entered OTP.

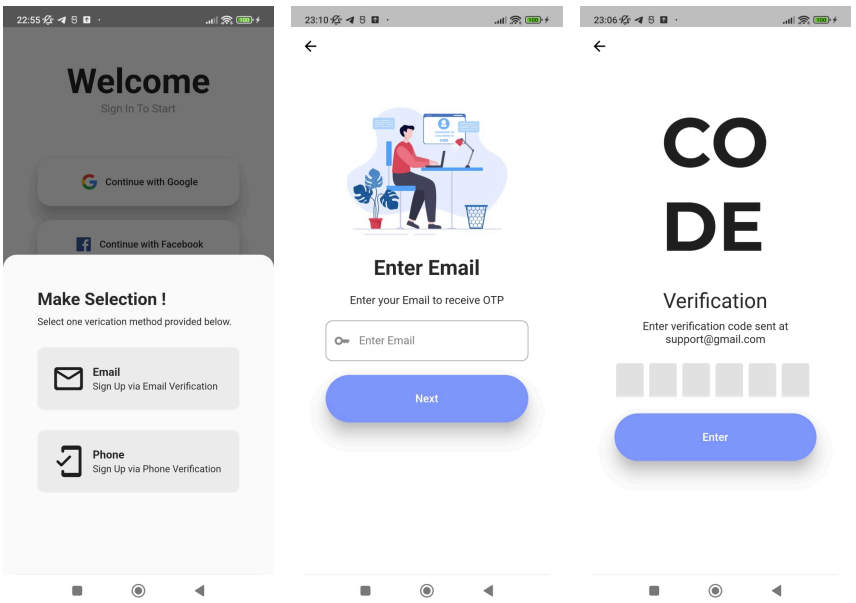
Similarly, for phone number OTP authentication:

- The AuthenticationRepository class also handles phone authentication operations. It uses Firebase Authentication's verifyPhoneNumber method to send OTPs to the user's phone number.
- The phoneAuthentication method in the repository triggers the OTP verification process by calling the verifyPhoneNumber method with the provided phone number.
- The OTPController class manages the logic for phone number OTP authentication. It holds references to the phoneNum and phoneCode (country code) text controllers.
- The phoneNumAuthentication method in the controller combines the phone code and number to authenticate the phone number via the repository's phoneAuthentication method.
- The PhoneOTPScreen widget represents the screen where the user enters their phone number for OTP verification. It displays an image, text inputs, and a form for submitting the OTP.
- The PhoneOTPForm widget contains the logic for handling the OTP form submission. It uses the OTPController to call the verifyOTP method in the repository to verify the entered OTP.

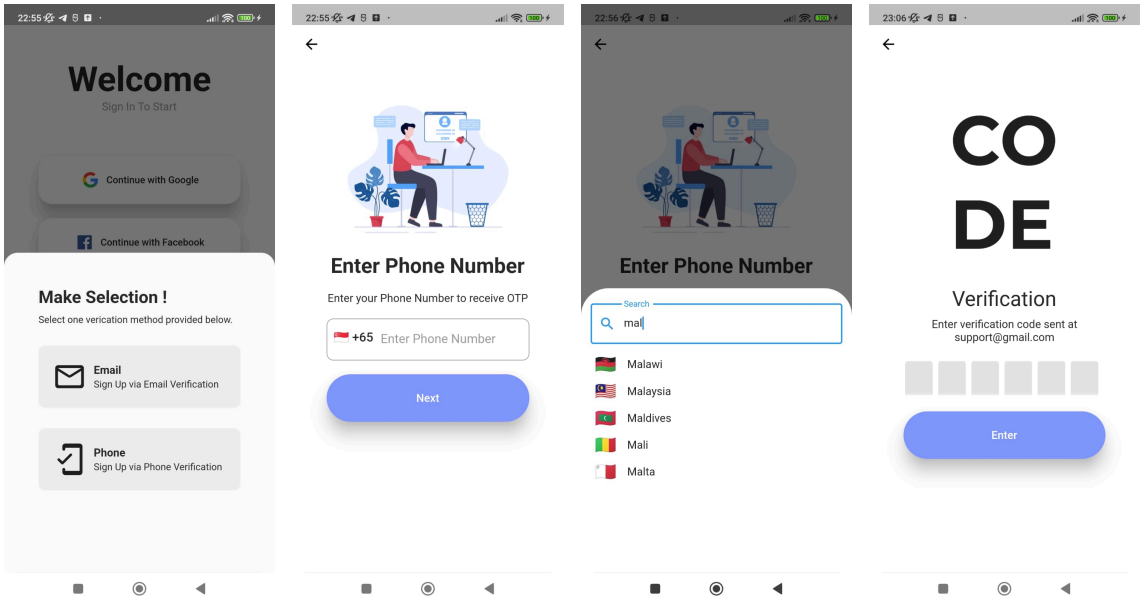
In summary, we utilised a special package called "email_otp" to facilitate the sending of the 6-digit pin for authentication via email. This package will handle the generation of random pins and provide the necessary functionality to authenticate them.

For phone number authentication, we relied entirely on Firebase Authentication. The process will begin with verifying the phone number provided by the user. Once the phone number is verified, a randomly generated pin will be sent to the user's phone. Firebase Authentication will handle the verification of the 6-digit pin entered by the user. It's worth noting that our implementation takes into account phone numbers from various nations, allowing seamless access regardless of users' location.

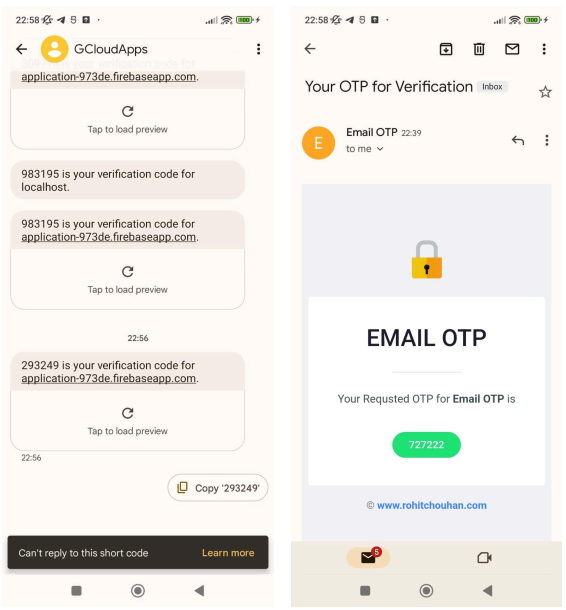
In both cases, the controllers and repositories play a crucial role. The controllers, such as OTPController, provide the necessary logic and actions for handling user input and interacting with the repository. The repository, represented by the AuthenticationRepository class, acts as an interface between the Flutter app and Firebase Authentication. It encapsulates the authentication operations, such as email and phone verification, OTP verification, and user creation or sign-in.



Via Email Authentication Process

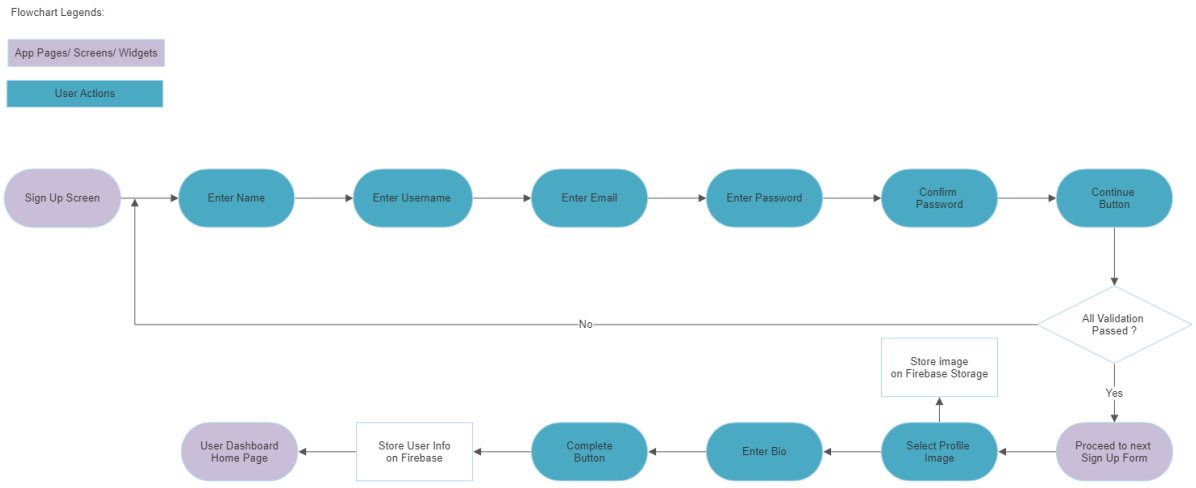


Via Phone Authentication Process



Sending of the 6 digit pins

User Registration: [Completed]



User Registration Flowchart

<u>User Stories:</u>

- As a user, I want to sign up for an account by providing my name, email, username, and password, enabling me to create a unique and personalised profile.
- As a user, I expect the sign-up form to validate my inputs, ensuring that I provide the required information and meet any specified criteria for each field.
- As a user, I want to select a profile image during the sign-up process, allowing me to customise my profile and make it more visually appealing.
- As a user, I expect the profile image selection to provide options such as uploading an image from my device.
- As a user, I appreciate an intuitive and user-friendly interface for selecting a profile image, making it easy for me to browse and choose an image that represents me accurately.

- As a user, I want the system to handle the profile image upload securely and efficiently, ensuring that my image is stored and displayed correctly on my profile.
- As a user, I expect the system to provide visual feedback or a preview of the selected profile image, allowing me to confirm that it is the desired image before completing the sign-up process.
- As a user, I want the option to skip the profile image selection if I prefer not to add one immediately, allowing me to complete the sign-up process without a profile image.
- As a user, I expect the system to handle any errors or issues related to profile image selection gracefully, providing clear error messages or instructions to help me troubleshoot and resolve the problem.

Proposed Feature Description:

Our proposed app will allow new users to register for a new account. To do so, they will be prompted to fill in the following details as displayed below. Some details are left optional to accommodate for any privacy concerns, ensuring that user's comfortability is prioritized.

- Name & Username **[Completed]**
- Email **[Completed]**
- Password **[Completed]**
- Description **[Completed]**
- Profile Image (Optional) **[Completed]**

To ensure a smooth account creation process, all user inputs are validated. Certain fields, such as the profile image, have default entries to accommodate users who may have privacy concerns or choose not to upload an image initially. The registration process via email and password takes place on two easily navigable scenes, allowing users to go back and forth if they wish to make changes to their entries.

For first-time sign-ups/ sign-ins using Google accounts, some fields, including username, name, and email, will be automatically populated with the respective information retrieved from the user's Google account. Other fields will have temporary placeholders until the user updates them.

For both registration processes, profile images can only be selected from the phone's gallery, and all images will be stored in Firebase Storage for retrieval and updating purposes.

Proposed Feature Implementation Philosophy:

As mentioned above the registration process via email and password takes place on two easily navigable scenes, we did this by splitting the form into 2 parts: Form 1 and Form 2. Form 1 collects the user's name, username, email, and password, while Form 2 allows them to provide a bio/description and profile image. Essentially Form 2 are the input fields which are not mandatory.

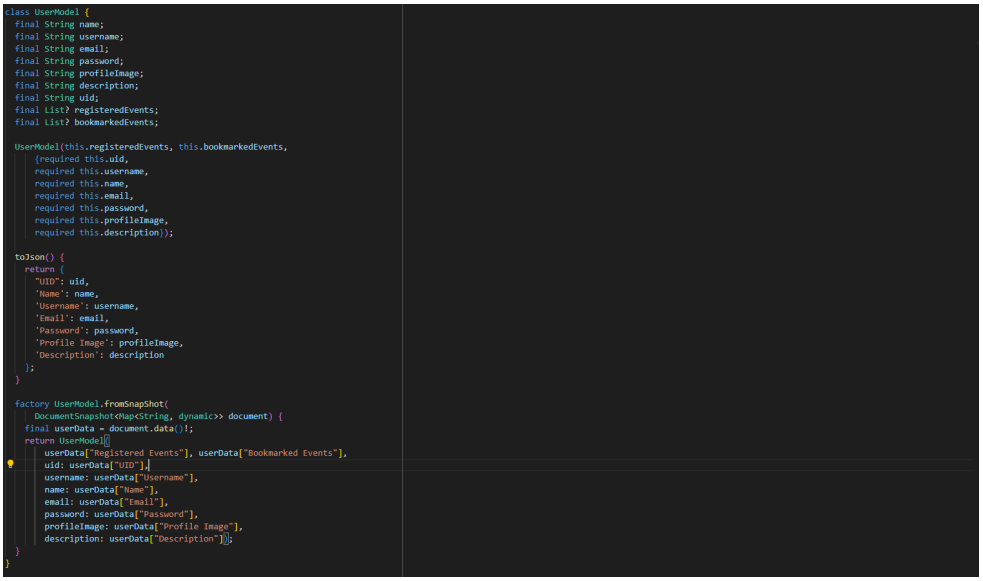
To handle the sign-up process, the SignUpController is responsible for coordinating the registration flow. It utilises the AuthenticationRepository and UserRepository to interact with Firebase Authentication and Firestore, respectively.

Form 1 uses the SignUpController to manage the text editing controllers for each input field. It also uses the ValidationController to perform validation on the form fields. Whereas Form 2 allows the user to select a profile image by tapping on the SelectProfileImageWidget. This widget uses the ImagePicker package to open the gallery and choose an image. The selected image is then uploaded to Firebase Storage using a unique ID

generated for the image. The image URL is retrieved and stored in the SignUpController's profileImage text controller. The bio input field is also managed by the SignUpController.

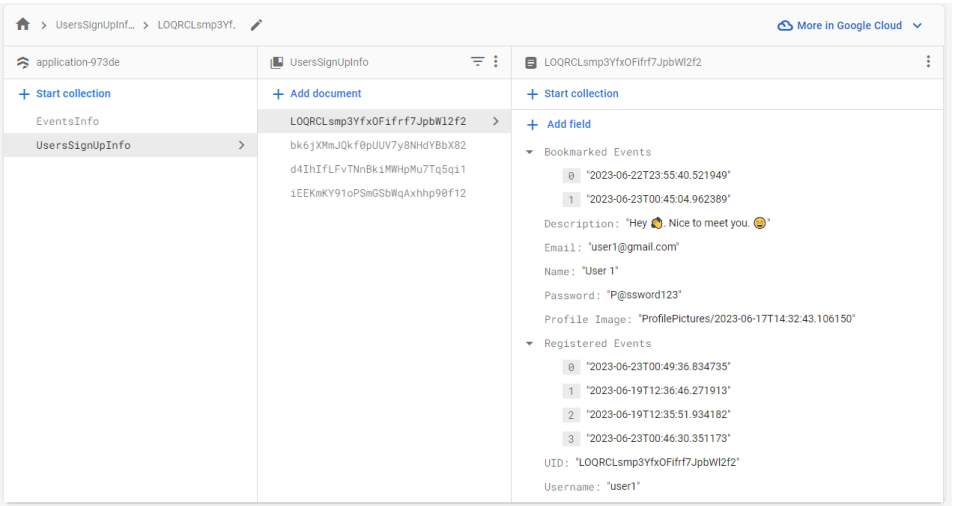
When the registration form is complete, the registerUser method in the SignUpController is invoked. This method utilises the AuthenticationRepository to create a new user account using the provided email and password. Once the user account is created, a UserModel is constructed using the input values from Form 1. The UserModel represents the user's profile information and is stored in Firestore via the UserRepository. (Note that profile image is stored as an image URL).

In summary, the controllers facilitate the interaction between the UI, Firebase services, and repositories, ensuring a smooth registration process for the user. (It's important to note that a UserModel class was created to temporarily store all data from the user's inputs. This approach improves data management by avoiding the need to pass large amounts of data via parameters.)

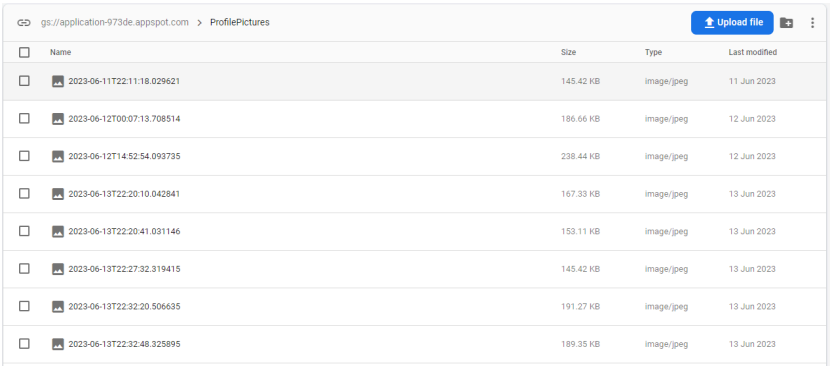


```dart
class UserModel {
  final String name;
  final String username;
  final String email;
  final String password;
  final String profileImage;
  final String description;
  final String uid;
  final List? registeredEvents;
  final List? bookmarkedEvents;

  UserModel(this.registeredEvents, this.bookmarkedEvents,
      {required this.uid,
      required this.username,
      required this.name,
      required this.email,
      required this.password,
      required this.profileImage,
      required this.description});

  toJson() {
    return {
      "UID": uid,
      'Name': name,
      'Username': username,
      'Email': email,
      'Password': password,
      'Profile Image': profileImage,
      'Description': description
    };
  }

  factory UserModel.fromSnapShot(
      DocumentSnapshot<Map<String, dynamic>> document) {
    final userData = document.data()!;
    return UserModel(
        userData["Registered Events"], userData["Bookmarked Events"],
        uid: userData["UID"],
        username: userData["Username"],
        name: userData["Name"],
        email: userData["Email"],
        password: userData["Password"],
        profileImage: userData["Profile Image"],
        description: userData["Description"]);
  }
}
```

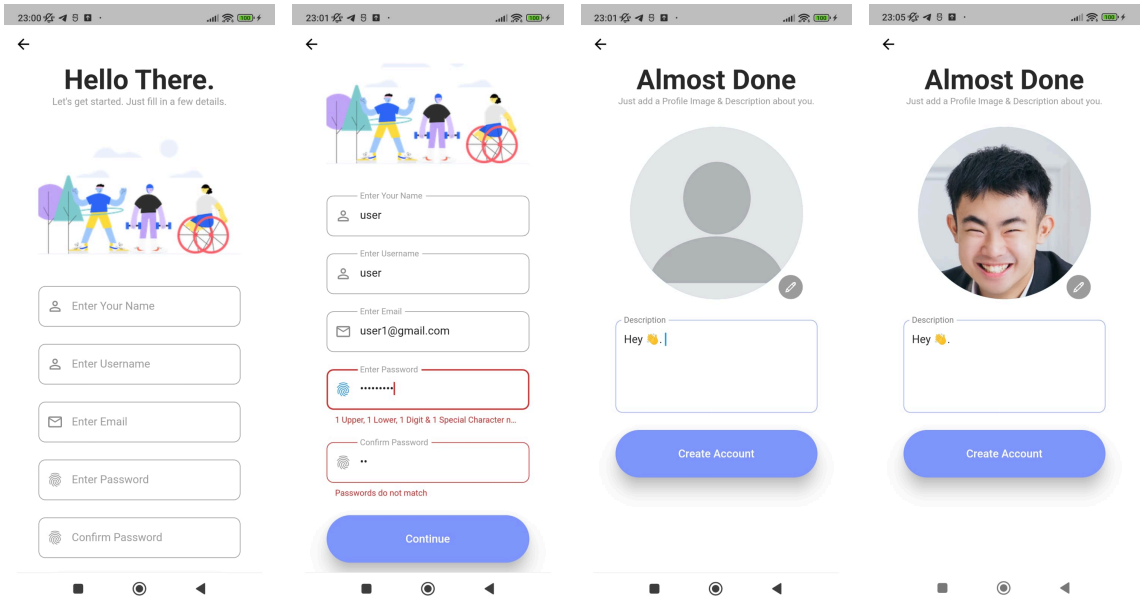
UserModel Class



UsersSignUpInfo Collection

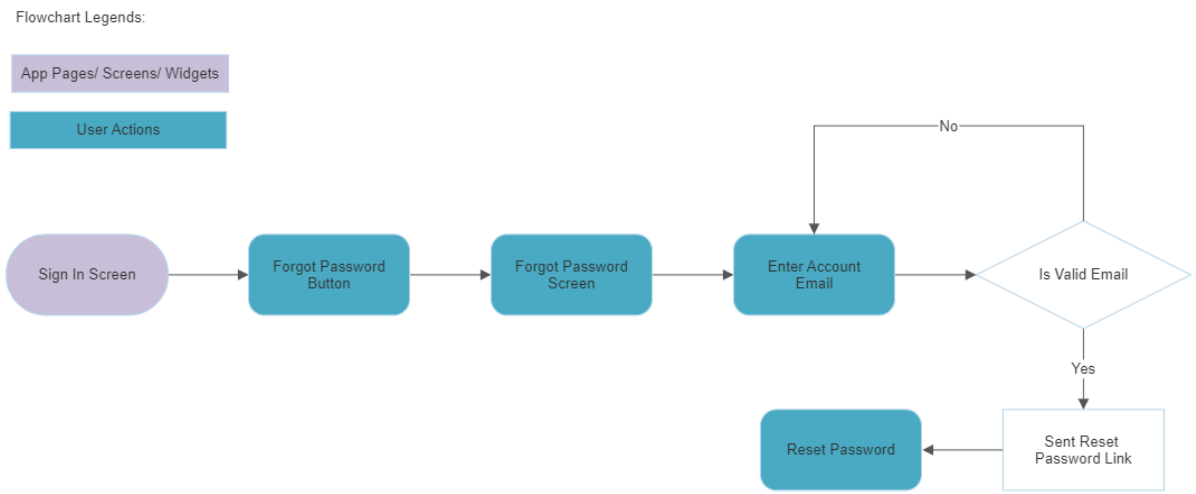


Profile Images Storage Location



Users Sign Up Process

Forgot Password: [Completed]



Reset Password Flowchart

User Stories:

- As a user, I want to have the ability to recover my account if I forget my password, using features such as password reset.
- As a user, I expect the "Forgot Password" feature to be easily accessible from the login screen, providing a clear and visible option to initiate the password recovery process.
- As a user, I want the system to prompt me to enter my registered email address when I click on the "Forgot Password" option, ensuring that the password reset instructions are sent to the correct email.

Proposed Feature Description:

This feature allows users to reset their password via their registered email. A button is provided on the sign-in page to initiate the password reset process. When a valid email is provided, a reset password email will be sent to the user.

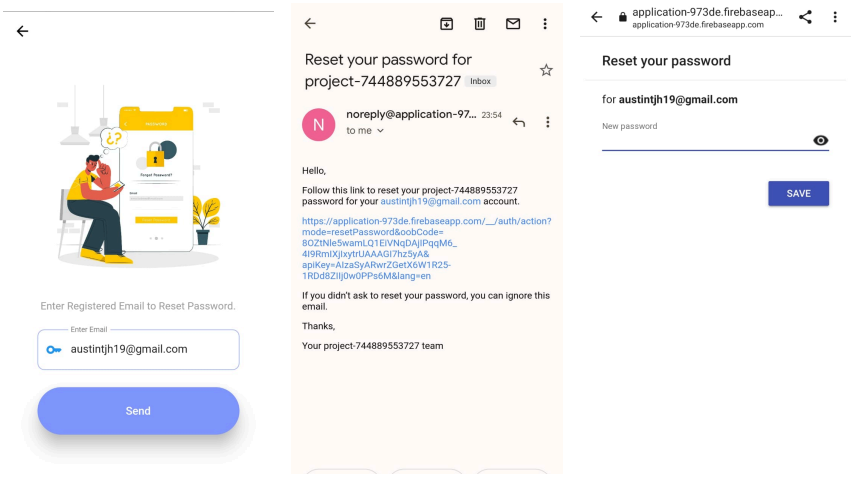
Once the user receives the password reset email, they can follow the instructions provided to reset their password. This typically involves clicking on a link or accessing a reset password page. The user can then enter a new password of their choice. Once the password reset process is completed, the user will be able to log in to their account using the new password.

Proposed Feature Implementation Philosophy:

The UI is composed of two main components: the ForgotPasswordScreen and the ForgotPasswordForm. The screen widget represents the overall layout of the "Forgot Password" screen, including the app bar, background colour, and the form itself. The form widget, on the other hand, contains the form fields for entering the email and a "Send" button. The form fields are validated using the ValidationController, which ensures that the email is entered in a valid format.

When the user enters their email and clicks the "Send" button, the form validation is triggered. If the email is valid, the resetPasswordViaEmail method is called, which in turn communicates with the AuthenticationRepository to send the password reset email using Firebase. The user is then directed to follow the instructions provided in the email to reset their password.

Overall, the "Forgot Password" process involves capturing the user's email, communicating with Firebase Authentication through the repository, and utilising Firebase's password reset functionality to enable users to reset their passwords securely.



Forgot Password Process

App Bar: [Completed]

User Stories:

- As a user, I want my profile image and name to be displayed prominently on the app bar, allowing me to quickly identify the account I am currently on.
- As a user, I expect my profile image to be displayed in a visually appealing and recognizable manner on the app bar, providing a personalised touch to the overall app interface.
- As a user, I expect the app bar to display my profile image and name consistently across different screens and sections of the app, ensuring a cohesive and familiar user experience.
- As a user, I want the app bar to provide easy access to additional profile-related features or actions, such as editing my profile or accessing account settings.
- As a user, I appreciate the responsiveness of the app bar, ensuring that my profile image and name are displayed properly and proportionally on different screen sizes and orientations.

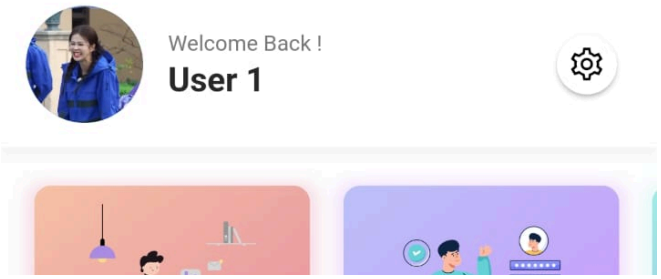
Proposed Feature Description:

The default app bar is available on all main feature screens, ensuring consistent navigation and user experience throughout the app. It provides users with important details of their current account, including their selected name and profile image, if any.

In addition to displaying account information, the app bar incorporates a dedicated button that allows users to access the Settings screen. This button serves as a central hub for various account management options. Within the Settings screen, users have the flexibility to perform actions such as signing out, editing their user profile, and modifying their registered email or password.

Proposed Feature Implementation Philosophy:

A general file Dashboard was utilised to improve readability of the code as well as its scalability. This general file Dashboard allows for the Appbar to be only written once even as users navigate to other pages. Essentially, when users navigate to other pages in the app, the app bar will continue to be visible at the top of the screen. The content of the app bar, represented by a DashboardAppBar widget, will also persist and remain consistent across different pages. Thus to update, only the DashboardAppBar widget has to be changed.



Default App Bar

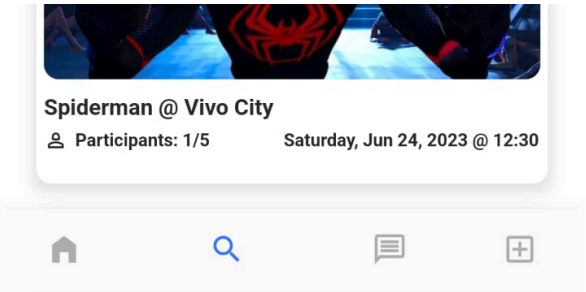
Bottom Navigation Bar: [Completed]

User Stories:

- As a user, I want to easily navigate to different pages by tapping on the corresponding icon in the bottom navigation bar, allowing me to quickly access the main content and features of the app.
- As a user, I expect the home page to be the default destination when I open the app or switch back to the app, providing a seamless and familiar user experience.
- As a user, I value a visually appealing and intuitive design for the bottom navigation bar, with clear and recognizable icons, consistent color schemes, and responsive interaction for enhanced usability.

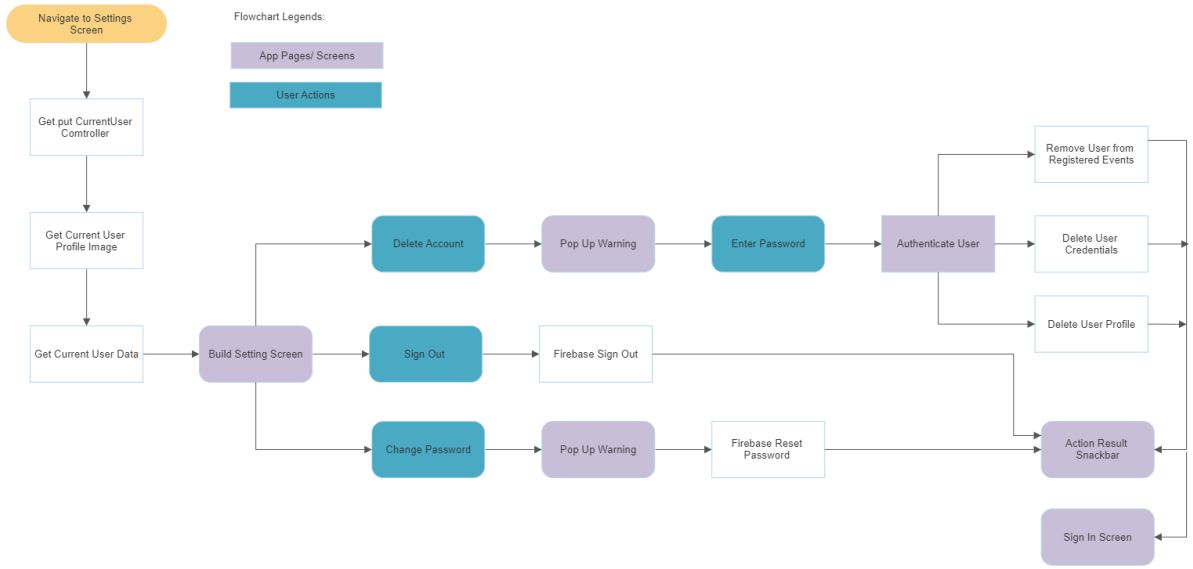
Proposed Feature Description:

It acts as the app's main navigation system for users, providing them with seamless access to four essential pages: Home, Search, Messages, and Create. Each page is thoughtfully labelled with built-in icons, enhancing the app's visual appeal and making it more intuitive for users to navigate. To ensure a user-friendly experience, a focused colour scheme is utilised, allowing for easier interaction and a clear understanding of the current viewing page. Furthermore, animated transitions are implemented to create a smooth and engaging navigation flow, enhancing the overall usability of the app.



Bottom Navigation Bar

Settings Screen: [Completed]



Settings Screen Flowchart

User Stories:

- As a user, I want the option to sign out or log out of my account when I'm done using the application, ensuring my account remains secure.
- As a user, I expect a smooth sign-out process that clears my session, removes any locally stored credentials, and redirects me to an appropriate screen, such as the Sign In Screen.
- As a user, I want to view my profile image, username, and email on the settings screen, providing me with a quick overview of my account information.
- As a user, I want the ability to edit my user profile by tapping on the "Edit User Profile" option in the settings screen, allowing me to update my profile image, username, name, or bio if needed
- As a user, I want the option to delete my account by selecting the "Delete Account" option in the settings screen, giving me control over permanently removing my account and associated data from the app.
- As a user, I appreciate a confirmation prompt or multi-step process when selecting the "Delete Account" option to ensure that I understand the consequences and can proceed with caution.
- As a user, I expect the app to securely handle the account deletion process and provide clear information about the actions that will be taken, such as permanently deleting my data and disabling access to the app's features.
- As a user, I want the ability to change my password by selecting the "Change Password" option in the settings screen, allowing me to update my account security and protect my information.

Proposed Feature Description:

The settings screen in our app serves as a central hub for users to manage their account and customise various aspects of their experience. It offers a range of actions and displays key user information to enhance usability and provide a visually appealing interface. The app settings screen will currently allow for users to completer 4 main actions:

- Edit User Profile **[Completed]**
- Delete Account **[Completed]**
- Change Password **[Completed]**
- Sign Out of Account **[Completed]**

Within the settings screen, users can access the option to edit their user profile. This feature allows them to update their profile information, such as their name, bio, or other relevant details. By providing users with the ability to customise their profile, we empower them to personalize their presence within the app and make it uniquely their own.

Users also have the option of signing out, deleting their account or changing their password. For both deleting of account and changing of password a pop up warning will be issued to the users when they perform these actions. For the deletion of an account, we further require users to enter their password as a percussion.

Proposed Feature Implementation Philosophy:

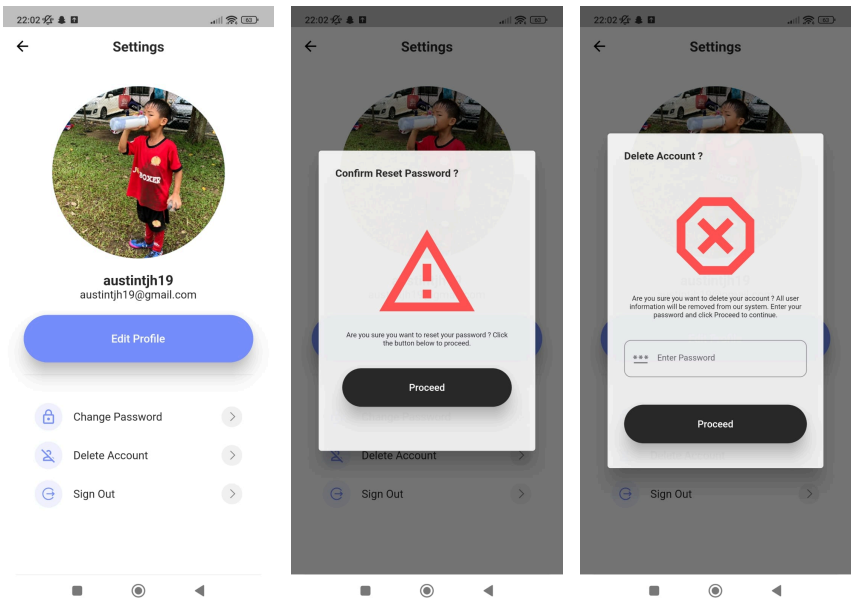
The implementation philosophy behind retrieving the user's email, username, and profile image follows a structured approach. The data retrieval process is facilitated by a CurrentUserController and UserRepository, utilising a combination of Firestore and Firebase Storage. To obtain the user's email, username, and profile image, a FutureBuilder widget is employed in the SettingsScreen. It calls the getUserData method from the CurrentUserController, which in turn invokes the corresponding method in the UserRepository. This method queries the Firestore database, retrieves the user document based on the UID, and maps the data to a UserModel object containing the email, username, and profile image.

The FutureBuilder widgets handle the asynchronous nature of the data fetching, displaying loading indicators while data is being retrieved. Once the data is available, it populates the UI components accordingly, showing the email, username, and profile image (if available).

For signing out, the built-in sign-out functionality provided by Firebase was utilised, ensuring that the user's session is properly terminated and they are no longer considered authenticated. This approach allows for consistent handling of user sign-out across different authentication methods (such as email/password, Google, and Facebook). Promoting code reusability and minimising potential issues that may arise from the implementation of a custom sign-out logic.

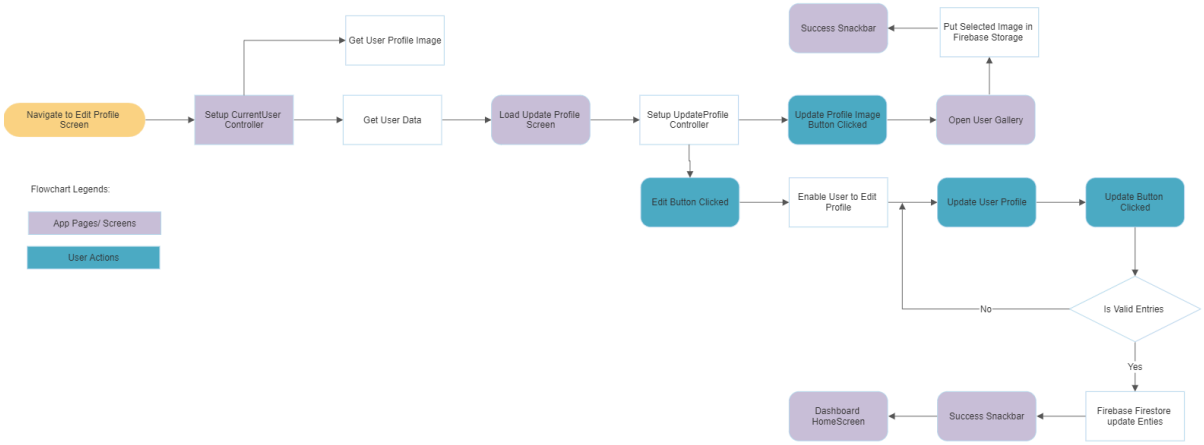
Similarly, for deleting user accounts, everything was largely handled by built in Firebase Authentication & Storage Functions. This action works by first re authenticating the user using their email and password. (For confirmation of account deletion) If authenticated, the app will then proceed to remove the user from any registered events by iterating through the registeredEvents list and invoking the _eventRepository.removeParticipant method. Following that, it deletes the user's account using the delete method provided by Firebase , and upon successful deletion, it deletes the user's profile using _userRepository.deleteUserProfile.

Regarding password resetting, its implementation mirrors that of the forgot password action at the beginning of the app. Furthermore, it is worth noting that the user actions described above involve API calls that are managed by repository files, each of which is called by respective controllers.



Settings Screens

Edit User Profile : [Completed]



Edit User Profile Flowchart

User Stories:

- As a user, I want to be able to edit my name in my user profile, so that I can keep my profile information up to date.
- As a user, I want the option to edit my username in my user profile, allowing me to personalise my account and make it more identifiable to others.
- As a user, I would like the ability to edit my bio in my user profile, enabling me to share a brief description about myself or any other relevant information.
- As a user, I expect to have the capability to upload or change my profile image in my user profile, allowing me to customise my visual representation in the app.
- As a user, I appreciate having an intuitive and user-friendly interface for editing my user profile, ensuring that the process is straightforward and easy to understand.
- As a user, I want the changes made to my user profile (name, username, bio, profile image) to be reflected across the app, so that others can see the updated information.

Proposed Feature Description:

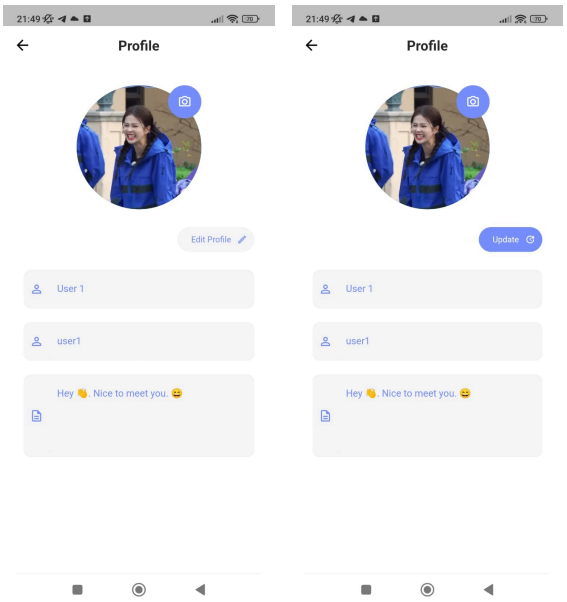
The "Edit User Profile" feature provides users with the ability to customise and personalise their profile within the platform, enhancing their sense of identity and allowing for a more tailored user experience. Users can modify various profile details such as their name, username, bio, and profile image. When a user initiates the profile editing process, they are presented with a user-friendly interface where they can input and update their desired changes. Once the user confirms the modifications, the updated profile details are immediately sent and persisted to the database, ensuring that the changes are reflected in real-time across the platform. For profile images, users currently only have the option to choose an image from their phone gallery.

Proposed Feature Implementation Philosophy:

We follow the principle of separating concerns by dividing the functionality into different methods within the UpdateProfileController and UserRepository classes. The UpdateProfileController handles user input and triggers the appropriate repository methods for updating the profile. The UserRepository class encapsulates the logic for interacting with Firebase Firestore and Firebase Storage to perform the necessary updates.

To expand on this matter, the user profile is updated in Firestore by accessing the appropriate document and using the update() method to modify specific fields. And Firebase Storage is used to update the user profile image by updating the corresponding field in the Firestore document.

We include error handling mechanisms to catch and handle potential exceptions that may occur during profile updates. This is done primarily through try-catch blocks to catch exceptions and displays appropriate error messages using Get.snackbar().



Edit Profile Screen

User Dashboard - Home Screen: [Completed]

User Stories:

- As a user, I want to see a visually appealing and easy-to-navigate home page when I open the app, so that I can quickly access the main features and content.

- As a user, I expect to find prominent buttons or icons on the home page that allow me to navigate to other key sections or features of the app, such as search, messaging, notifications, or user profile.
- As a user, I want the home page to display a horizontal scroll list of events that I have bookmarked, so that I can easily access and revisit those events.
- As a user, I expect the bookmarked events on the home page to be visually distinguishable and provide relevant information, such as event title, date, and location, to help me identify and select the events of interest.
- As a user, I want to be able to tap on a bookmarked event from the home page and be redirected to the event details page, so that I can view more information and register.
- As a user, I expect the registered events section on the home page to show relevant details, such as event title, date, and location, to help me easily identify the events I have registered for.
- As a user, I want the ability to tap on a registered event on the home page and be taken to the event details page, where I can view more information and manage my registration.
- As a user, I appreciate having a seamless and smooth scrolling experience on the home page, allowing me to browse through the bookmarked and registered events effortlessly.
- As a user, I want the home page to be responsive and regularly updated, ensuring that any changes to my bookmarked or registered events are reflected accurately.

Proposed Feature Description:

After the account is set up, or when a log-in event is authorised the user will be directed to the homepage where all the fun happens. The pictures below demonstrate the basic UI layout of the home page, it consists of a bottom navigation bar and on the top the user can manage their profile and app setting. On the home page, the users will be able to create a new event that is updated in realtime in firebase so after a new event is created, it will show up on the front page straight away. Apart from that there are two main sections, namely “your upcoming event” and “your bookmarked events” where the users can browse for the activities and events posted and perhaps join some of their liking. Each event displayed on home pages showcases event pictures, event name, location, time and participant counts, and all that information should be provided when posting a new event.

Proposed Feature Implementation Philosophy:

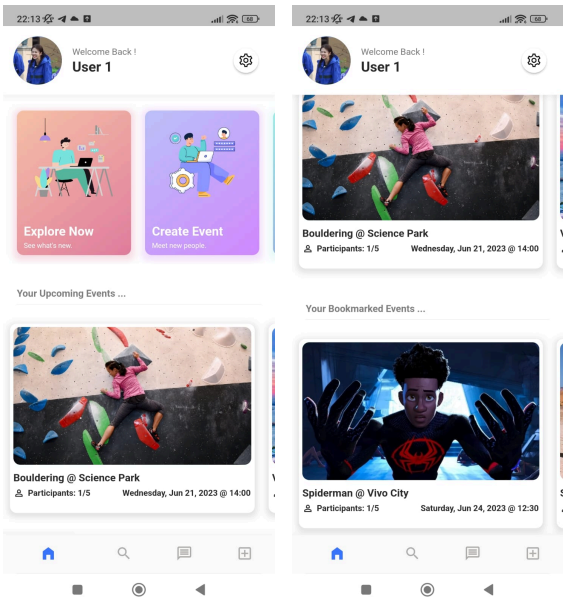
The User Dashboard Home page can be divided into three main widgets: the Actions List widget, Upcoming Events widget, and Bookmarked Events widget. Note that both the Navigation and App bar are constructed in a separate file, which calls a selected page, in this case the Dashboard Home Page when the user wishes to navigate there. This approach reduces code duplication and simplifies the implementation of page navigation.

The Actions List widget is a simple ListView of multiple buttons that direct users to specific screens within the app where they can perform various functions. The images used in the buttons are built-in assets of the app and remain constant regardless of the user.

The Upcoming Events widget and Bookmarked Events widget are implemented in a similar manner. Both widgets utilize the FutureBuilder widget. The event data, including the user's registered events and bookmarked events, are managed by the Current User Controller and General Event Controller. The controllers do not make direct API calls but instead request data from the User and Authentication Repository. The User Model stores only the IDs of the events, so the General Event Controller is responsible for making the API calls to retrieve the event data itself. These API calls to the database are implemented in the Event Repository.

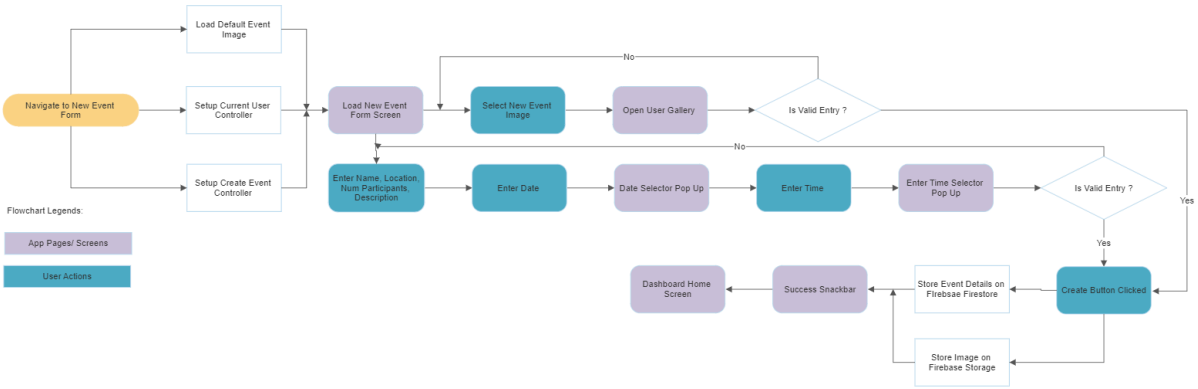
The Current User Controller is responsible for fetching user-specific data, such as the IDs of the registered and bookmarked events, from Firebase Firestore. It communicates with the User and Authentication Repository, which contains methods for retrieving user data from the database. For example, the getUserRegisteredEvents and getBookmarkedEvents methods in the Current User Controller utilize the UserRepository to fetch the event IDs associated with the current user. Once the event IDs are obtained, the General Event Controller takes over to fetch the event details from Firebase Firestore. It makes use of the EventRepository, which contains methods for retrieving event data based on the provided event IDs. For instance, the getEventsFromList method in the EventRepository filters the events based on the provided list of event IDs and returns a list of EventModel objects.

By separating the responsibilities between the controllers and repositories, we ensure a clear separation of concerns and maintain a modular architecture. The controllers handle user-specific data and interactions, while the repositories handle the data retrieval from the database. This approach promotes code reusability and easier maintenance in the long run.



User Dashboard (includes Registered & Bookmarked events)

Create Event: [Completed]



Create Event Flowchart

User Stories:

- As a user, I want to see an intuitive and user-friendly interface on the event creation page, so that I can easily provide the required information for the event.
- As a user, I appreciate the availability of options to choose an event image, allowing me to personalise the visual representation of the event.
- As a user, I want the ability to enter the event name and location fields, so that I can provide clear and specific details about the event.
- As a user, I expect the date selection process to be simple and convenient. When I click on the calendar icon, I want an overlaid date picker UI to appear, allowing me to easily choose a date for the event.
- As a user, I appreciate having a built-in time picker that allows me to select the event's start and end time. This ensures consistency in the UI design and provides an easy-to-use interface.
- As a user, I want the event creation page to include a description field, so that I can provide additional information and details about the event.
- As a user, I want to see an intuitive and user-friendly interface on the event creation page, so that I can easily provide the required information for the event.

Proposed Feature Description:

After tapping on the “create event” card, and then “Create Event”, the user will be navigated to the event creation page where they can choose the event image and name and location and so on. When setting the time and date, we choose to take advantage of the built-in DatePicker widget in flutter so when the user clicks on the calendar icon, an easy-to-use date picker UI would be overlaid on top of the current screen where the user can choose a date for the event. For choosing the time, similarly a TimePicker is also deployed to ensure the UI design consistency. At the bottom, the users can also add a description to the event.

Proposed Feature Implementation Philosophy:

A Create Event controller is utilised to manage the create event action. When a user initiates the create event action, the Create Event controller comes into play. It acts as a central hub, coordinating the necessary operations to ensure a smooth event creation process. Firstly, it handles any API calls required to fetch relevant data or interact with external services. These calls could include retrieving user information, updating user information or uploading event data.

The controller also performs validation checks to ensure that the input data is accurate and meets the necessary criteria. It verifies the completeness and correctness of the event details provided by the user, such as the event name, location, date, and time. Additionally, the Create Event controller takes responsibility for loading and storing of the uploaded event images. It manages the retrieval of images from user’s gallery and facilitates their storage for later use.

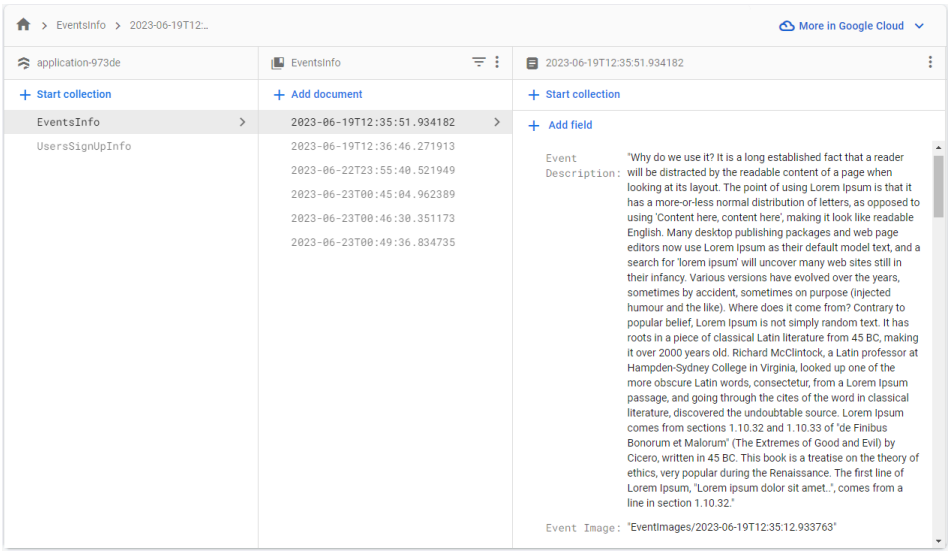
To streamline the process and avoid passing an excessive number of parameters, the controller utilises an Event Model. This model serves as a temporary container for event data, consolidating all relevant information into a single entity. For instance, the Event Model stores the event name and location as string arrays, allowing for efficient search capabilities using a search bar. The event date and time are represented as a timestamp, ensuring accurate and standardised time management. Furthermore, the participants are stored as a string array, containing the unique IDs associated with each participant. This organisation of data in the Event Model simplifies data manipulation and enhances code readability.

In terms of data storage, events' information is stored in the EventsInfo Collection. Each event is uniquely identified based on its creation time, with precision down to the microsecond. This approach ensures that

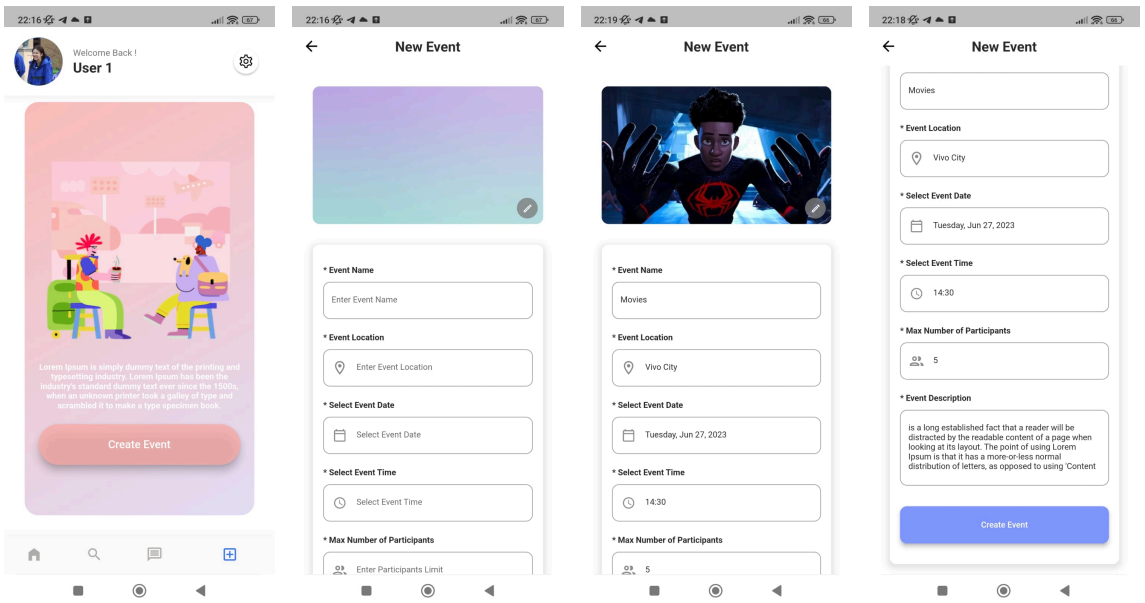
events can be easily retrieved, updated, or deleted as needed. Additionally, to handle the storage of event images, the Create Event controller leverages Firebase storage.

```dart
class EventModel {
  final String id;
  final String eventImage;
  final List eventName;
  final List eventLocation;
  final String eventDate;
  final String eventTime;
  final DateTime eventDateTime;
  final String participantsLimit;
  final String eventDescription;
  final List participants;

  EventModel(
      {required this.id,
      required this.eventImage,
      required this.eventName,
      required this.eventLocation,
      required this.eventDate,
      required this.eventTime,
      required this.eventDateTime,
      required this.participantsLimit,
      required this.eventDescription,
      required this.participants});

  toJson() {
    return {
      'ID': id,
      'Event Image': eventImage,
      'Event Name': eventName,
      'Event Location': eventLocation,
      'Event Date': eventDate,
      'Event Time': eventTime,
      'EventDateTime': eventDateTime,
      'Participants Limit': participantsLimit,
      'Event Description': eventDescription,
      'Participants': participants,
    };
  }
```

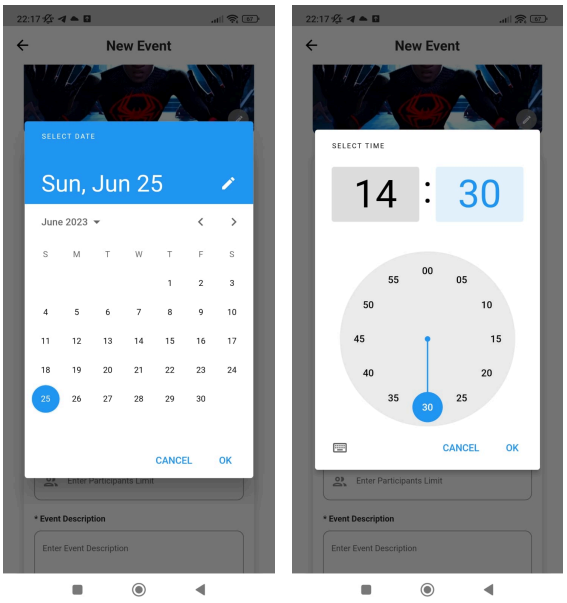
EventModel Class



EventsInfo Firebase Collection



Event Creation Process



Event Creation Entering Date & Time Forms

Event Details Screen: [Completed]

User Stories:

- As a user, I want to be able to access the event details screen by tapping on a specific event card or event listing, so that I can view more information about the event.
- As a user, I want to see relevant details about the event on the event details screen, such as the event name, date, time, location, and description, so that I can make an informed decision about whether to participate in the event.
- As a user, I want to see the number of participants who have already registered for the event, as it gives me an idea of the event's popularity and level of interest.

- As a user, I want to view the list of participants who have already registered for the event, as it allows me to see who else will be attending and potentially connect with them.
- As a user, I want the option to register for the event directly from the event details screen, so that I can easily indicate my interest in participating.
- As a user, I want the option to bookmark the event directly from the event details screen, so that I can save the event for future reference or easy access.
- As a user, I want the ability to unregister from the event if I change my mind or can no longer attend, ensuring that my participation status is accurately reflected.
- As a user, I appreciate having the option to contact the event organiser directly from the event details screen, facilitating communication and coordination.

Proposed Feature Description:

When accessing the event details screen, users will be presented with a visually appealing layout that prominently displays the event image, capturing the essence of the event and providing an enticing visual representation. Users can quickly grasp the event's atmosphere and theme, helping them make informed decisions about their participation.

The event details screen will also showcase important information such as the event name, location, and time, enabling users to easily understand the event's logistics and plan their attendance accordingly. Clear and concise presentation of these details ensures that users can quickly assess their interest and availability.

Furthermore, the feature allows users to register for the event directly from the details screen. By including a registration button or option, users can easily express their intent to participate and secure their spot at the event. Additionally, a bookmarking feature will be available, allowing users to save events of interest for future reference or follow-up.

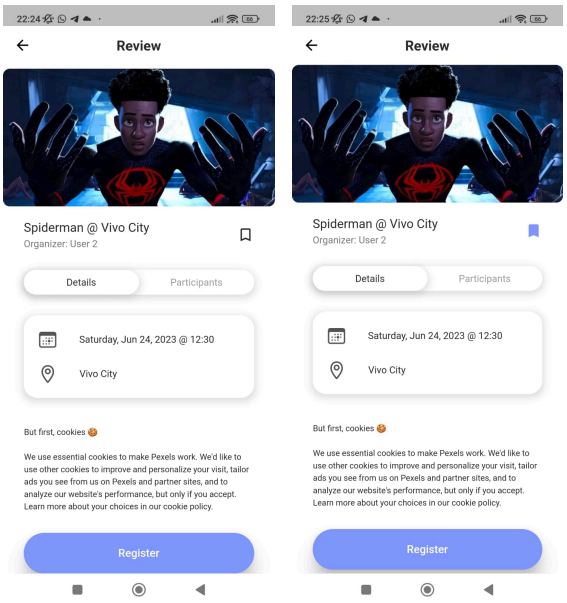
To promote community engagement and foster a sense of connection, the event details screen will showcase the number of participants already registered for the event. This provides social proof and helps users gauge the popularity and potential networking opportunities associated with the event. Moreover, a list of participants will be displayed, enabling users to see who else is attending. This enhances the networking aspect and encourages users to connect with like-minded individuals before the event. Users can view participant profiles, fostering a sense of community and facilitating interaction among event attendees.

Proposed Feature Implementation Philosophy:

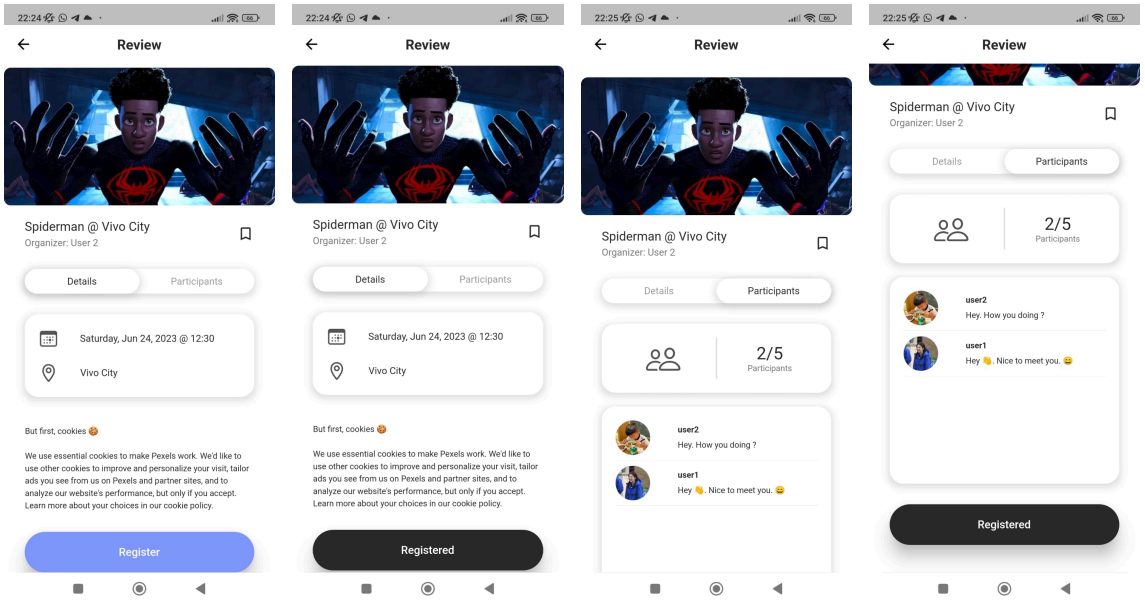
To enhance the readability and maintainability of the code, the event screen has been structured using multiple modular widgets. Each widget focuses on a specific actionable component, resulting in a more organised and comprehensible codebase. Moreover, the implementation follows a clear separation of concerns, with controllers managing user interactions and repositories handling API calls.

The General Event Controller plays a central role in the Event Details Screen. Its main responsibility is to fetch event data by utilising the getEventData method from the EventRepository. This method, which takes the event ID as a parameter, communicates with the database to retrieve detailed information about the event. The EventRepository acts as an intermediary between the controller and the database, handling the necessary API requests. It provides essential functionalities such as getEventData to fetch event details based on the event ID.

Furthermore, the General Event Controller is responsible for managing other important actions on the event screen. It facilitates the retrieval of event images through the getEventImage method, which is also implemented in the EventRepository. Additionally, the controller handles participant-related actions, including adding and removing participants, updating the number of participants, and checking if the event is full. These actions are carried out by invoking the corresponding methods in the EventRepository and AuthenticationRepository, which interact with the database and handle user authentication, respectively.



Event Bookmarking Process



Event Registration Process

Search (Search Bar + Filter): [Completed]

User Stories:

- As a user, I want to be able to search for events by their name, so I can quickly find specific events I'm interested in attending.
- As a user, I want to be able to search for events by location, so I can find events happening in a specific area or city.
- As a user, I want the search feature to provide accurate and relevant results based on the event name or location I enter.
- As a user, I want the search feature to be intuitive and easy to use, with a clear interface that allows me to enter my search criteria.
- As a user, I want the search feature to display search results in a visually appealing and organized manner, making it easy for me to browse through the events.
- As a user, I want to be able to refine my search results using filter criteria
- As a user, I want the search feature to provide real-time results, updating the search results as I enter or modify my search criteria.
- As a user, I want the search feature to be responsive and provide quick search results, ensuring a smooth and efficient user experience.

Proposed Feature Description:

The user can navigate to the search screen by clicking on the search icon at the bottom navigation bar and on the top there is a search bar and below it is a scrollable list of event cards. Search filters can be applied on the search result where the user can specify the location and time and number of participants, and also the order of the search result. After applying the filter, only the events that fit the filtering criteria will be displayed.

Proposed Feature Implementation Philosophy:

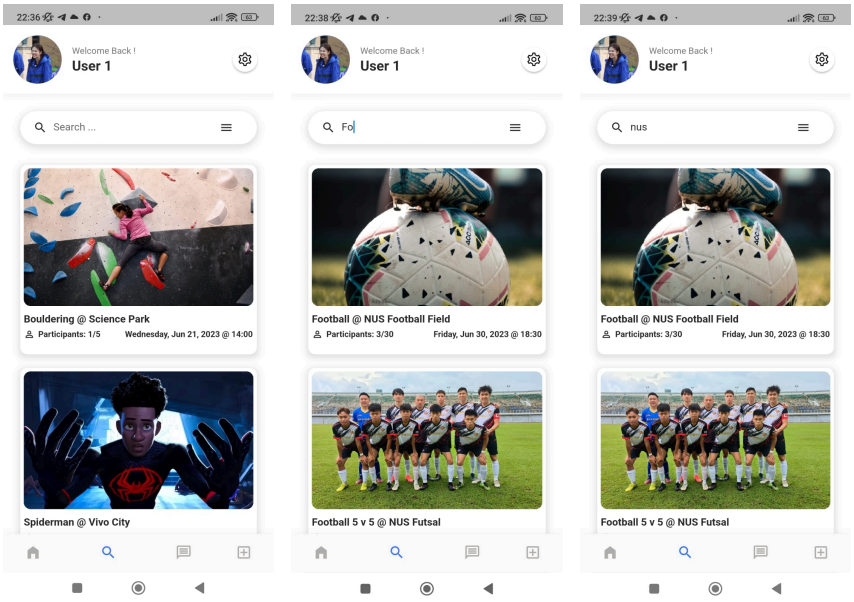
The searching for an event feature in the app is implemented using a combination of controllers, repositories, and Firebase calls. The key components involved in the search functionality are the SearchEventController, EventRepository, and Firebase Firestore.

The SearchEventController is responsible for handling the search functionality. It utilises the EventRepository to interact with Firebase Firestore to fetch the events data. The SearchEventController has a method called searchEventsFilteredBy that takes the filter criteria and order as parameters. This method makes a call to the EventRepository to fetch the events based on the specified filters. It passes the filter criteria and order to the repository's searchEventsFilteredBy method.

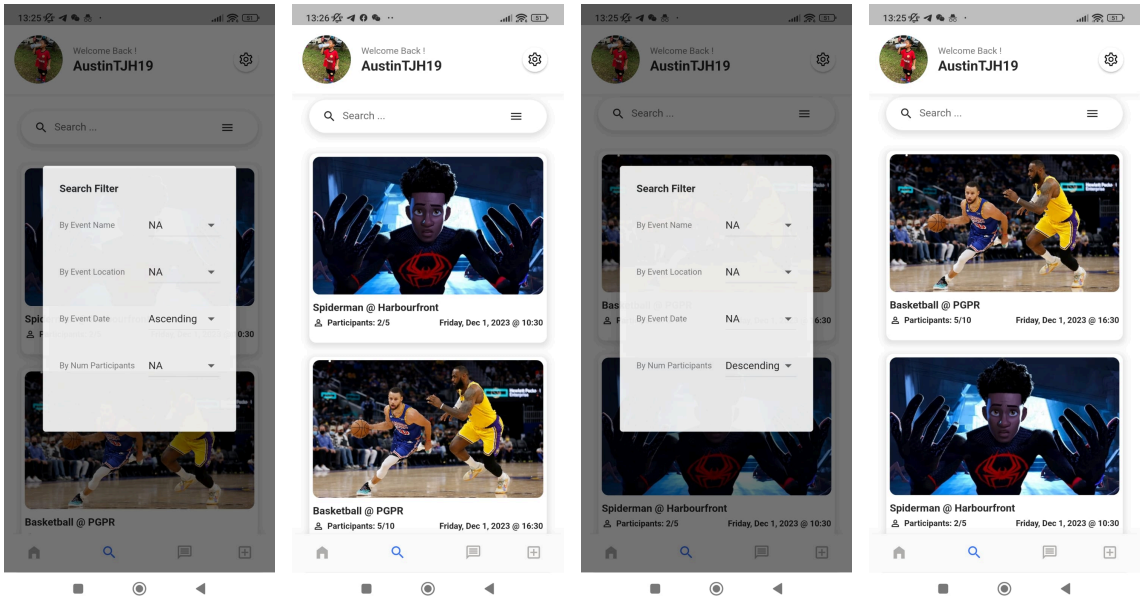
Inside the EventRepository, the searchEventsFilteredBy method performs a query on the Firebase Firestore collection to retrieve the events that match the search criteria. It uses the where method to filter the events based on the specified criteria. The EventRepository returns the list of matching events to the SearchEventController.

Back in the SearchEventController, the search results are returned to the DashboardSearchScreen, which is responsible for displaying the search results. The search results are displayed using the VerticalScrollEventsWidget. This widget takes the event details as input and displays them in a vertical scrollable format.

Overall, the search functionality follows a flow where the user input is captured in the DashboardSearchScreen, passed to the SearchEventController, and then used to query the EventRepository and fetch the relevant events from Firebase Firestore. The search results are then displayed in the UI using the VerticalScrollEventsWidget. This architecture allows for a flexible and efficient search feature in the app.



Searching Process (By Name & Location)



Filtering Search Data (By Date - Ascending to Num of Current Participants - Descending)

Messaging: [Ongoing]

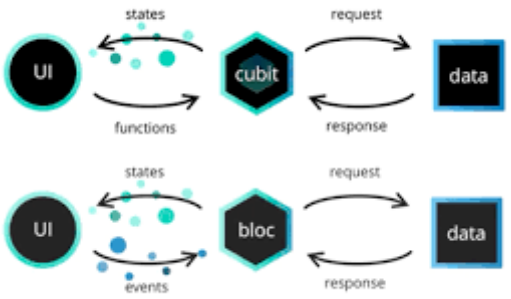


Diagram of Implementation Philosophy

User Stories:

- As a user, I am able to register for an event and chat with all the currently registered participants of the event
- As a user, I am able to join DM chat by clicking on another user's profile
- As a user, I am able to leave a group chat when leaving an event
- As a user, I am able to send messages in group chats where the messages are updated in real time
- As a user, I am able to see a list of chat rooms I'm in, that is also live updated

Proposed Feature Description:

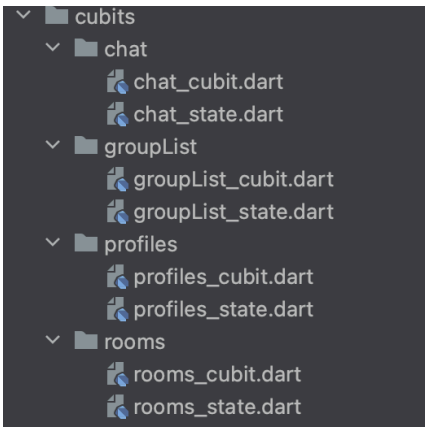
- Creating chat room (private DMs or group chat) **[Completed]**
- Displaying friends list **[Completed]**
- Search chat rooms **[Ongoing]**
- Real time messaging **[Completed]**
- Automatic chat group generation when an activity is created **[Completed]**
- Automatically leaving the event chat group when user deregisters for an event **[Completed]**

Messaging in our Project is done by following the BLoC architectural pattern. The BLoC (Business Logic Component) design pattern is a popular architectural pattern used in the development of software applications, particularly in the context of mobile and web applications. It was introduced by Google as part of the Flutter framework, but its principles can be applied to other platforms as well.

The primary goal of the BLoC design pattern is to separate the business logic of an application from the user interface, making the codebase more maintainable, testable, and scalable. It achieves this by dividing the application into three main components:

1) UI Layer: This layer is responsible for presenting the user interface and handling user interactions. It receives input from the user and sends it to the BLoC for processing.

2) BLoC Layer: The BLoC is the heart of the pattern. It acts as an intermediary between the UI layer and the data layer, containing all the business logic of the application. It takes input from the UI layer, processes it, and produces output, such as updated UI states or events. There are four cubits files and their respective state files created for state management.

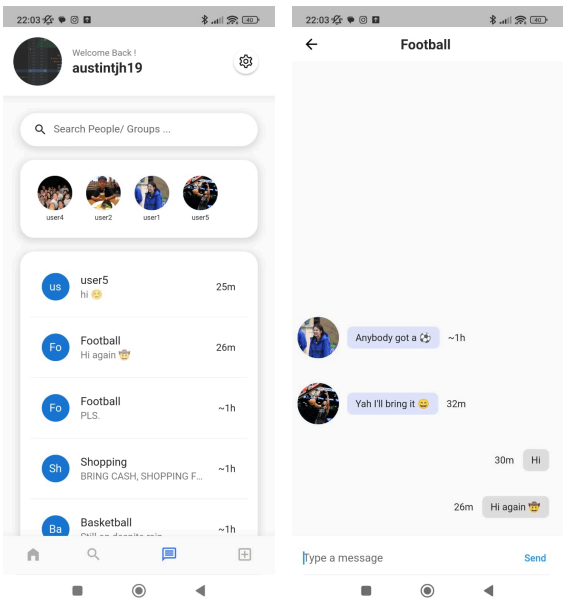


File Structure BLoC Layer

- chat_cubtis contains three methods : SetMemberListener(), SetMessageListener() and sendMessage(). SetMemberListener() is used to set up the stream for live updating the members in the group, SetMessageListener() sets up a stream that subscribes to messages table in the back end. sendMessage() is triggered when a user sends a message.
- Chat_state has 3 states: Chat_Loaded, Chat_Empty and Chat_error. Chat_loaded state is emitted when a chat is successfully updated or loaded. Chat empty is emitted when there are no previous messages.
- groupList_cubits is an a cubits that subscribes to the data fetched from supabase when searching for new groups to join. - ongoing
- Rooms_cubit contains methods such as initializeRooms(), getNewestMessages(), joinRoom() and createRoom(). initializeRoom() methods initialise the built-in messenger app by loading the current chat rooms and new users. it is lazily evaluated. getNewestMessages() method makes sure all messages sent in each room are updated in real time. joinRoom() is invoked when the user joins a chat room. createRoom() is invoked when the user creates a room or an event in which case the chat for that event would be automatically generated.
- Rooms_cubits has 4 main states: RoomsLoading (state emitted when the data is being fetched from the back end), RoomsLoaded (state emitted when the data is successfully loaded. RoomsEmpty (state emitted when there are no chat rooms found) and RoomsError that is emitted when an error is countered.
- Profiles_cubit also has two important attributes, first is user_profiles which is a cached map that stores all the preloaded userprofiles. The second is the get_profiles() method which is used to fetch room data from the back end.

3) Data Layer: The data layer is responsible for managing data sources, such as APIs, databases, or local storage. The BLoC interacts with the data layer to fetch or update data as required by the application's business logic. The Dart files that belong to this layer include classes such as Message, Profile and Room.

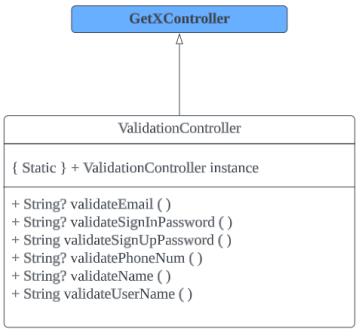
The communication between these layers typically follows a unidirectional flow. The UI layer sends events or commands to the BLoC, which then processes them and updates the UI with new states or emits further events.



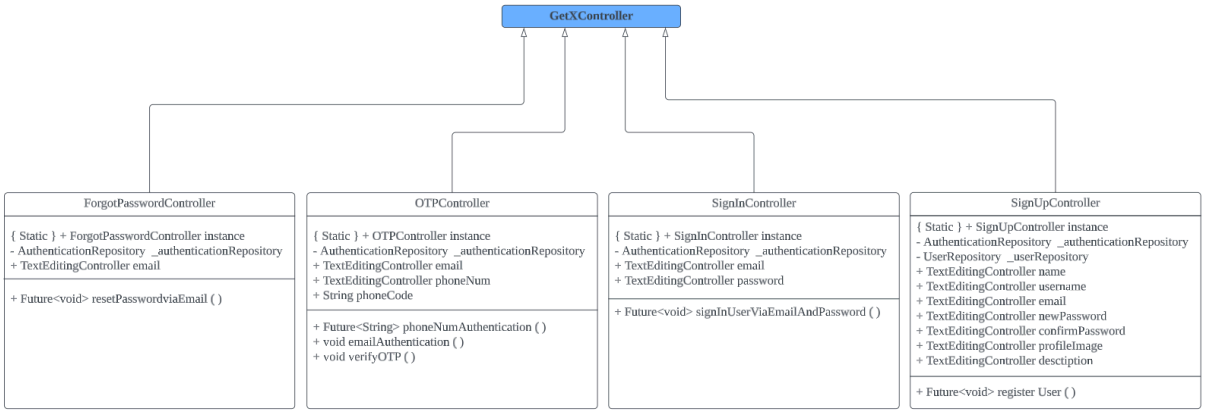
Class Diagrams (Key classes):

Application Controllers :

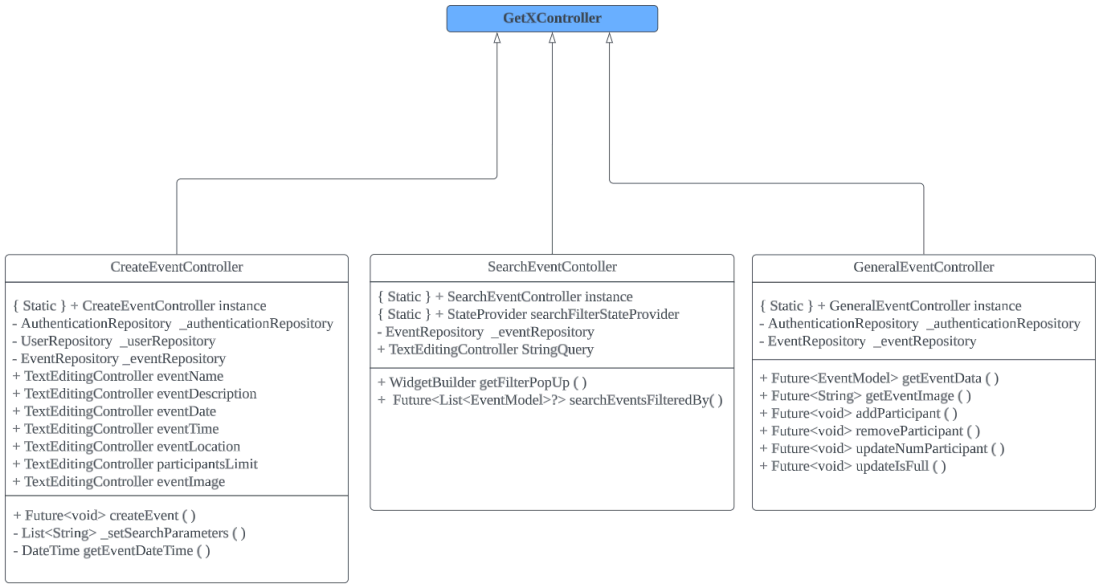
In our code base, an interesting observation can be made regarding the controllers. It is noteworthy that all controllers extend from a common base class known as GetXController. This inheritance relationship between GetXController and the various controllers is significant, as it establishes a hierarchical connection among them. To illustrate this relationship in a more visual manner, the diagrams presented below showcase the inheritance hierarchy, specifically highlighting the interplay between GetXController and the individual controllers.



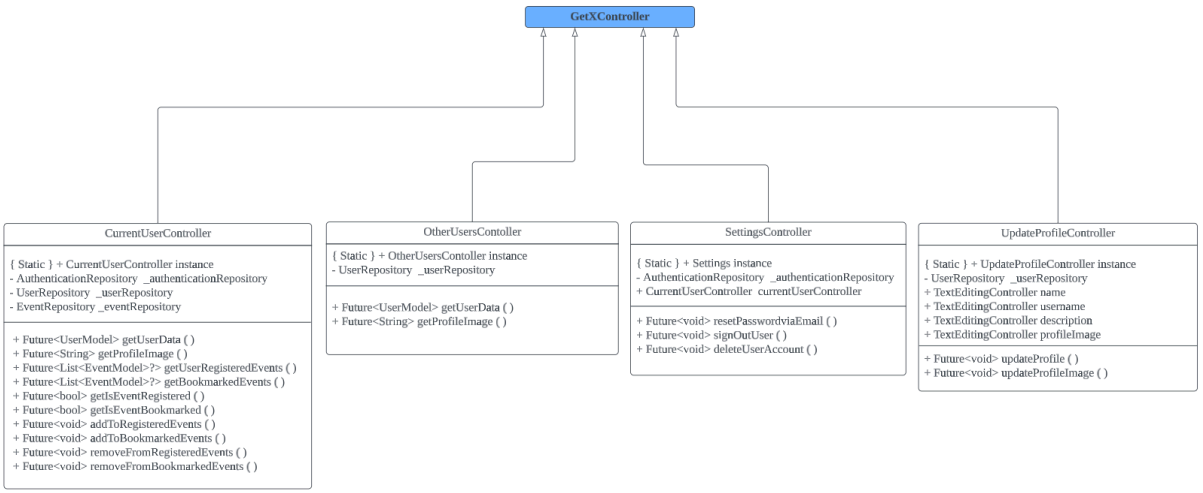
Class Diagram - ValidationController



Class Diagram - ForgotPasswordController, OTPController, SignInController, & SignUpController



Class Diagram - CreateEventController, SearchEventController, & GeneralEventController



Class Diagram - CurrentUserController, OtherUsersController, SettingsController & UpdateProfileController

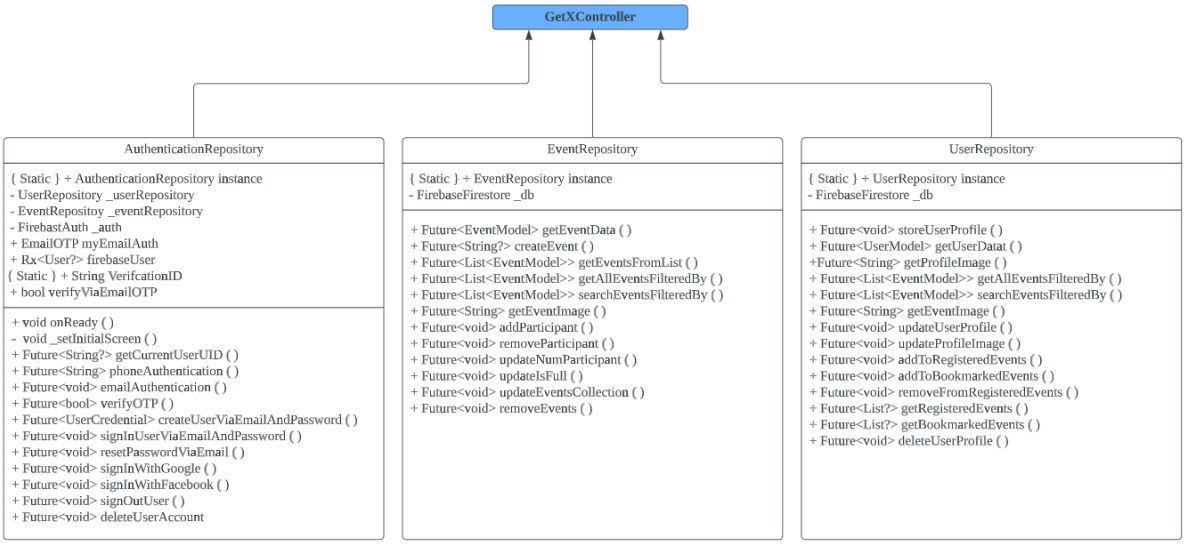
<u>Application Repositories:</u>

The repositories in our code are responsible for abstracting the data access layer and interacting with Firebase Firestore. They adhere to design principles such as separation of concerns and single responsibility.

The UserRepository handles all the operations related to user data, such as storing user profiles, updating user information, managing registered and bookmarked events, and retrieving user data. It encapsulates the logic and implementation details of Firebase Firestore operations specific to user-related data. By separating this logic into a repository, it ensures that the business logic and data access logic are decoupled, promoting better code organisation and maintainability.

Similarly, the EventRepository focuses on operations related to event data, including retrieving event details, adding and removing participants, and updating event information. It abstracts the interaction with Firebase Firestore specific to events. This separation allows for easier management and extensibility of the data access layer, making it simpler to replace the underlying data source or add caching mechanisms in the future.

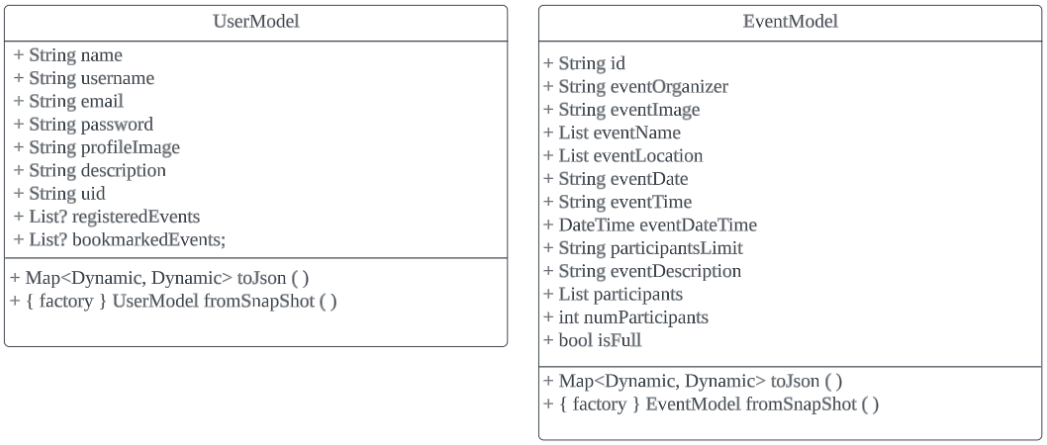
The AuthenticationRepository class is responsible for handling authentication-related operations in the app. It interacts with Firebase Authentication and other repositories to manage user authentication and account management.



Class Diagram - AuthenticationRepository, EventRepository, & UserRepository

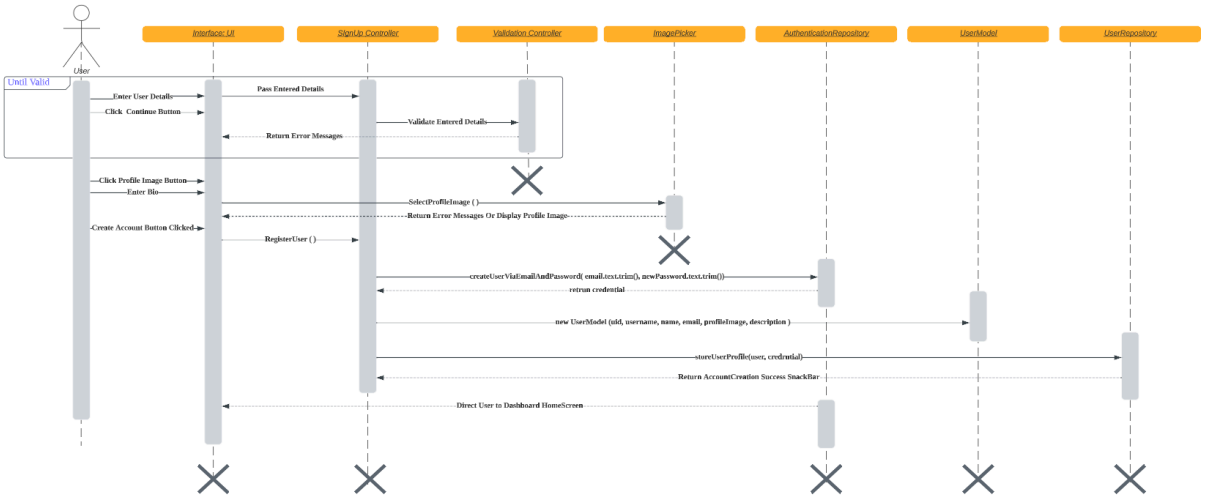
Application Models:

Models provide an abstraction layer between the raw API data and the application logic. By encapsulating the data within models, we can separate concerns and handle data-related operations separately, improving the overall maintainability and modularity of your codebase. Furthermore, these models help us with defining the structure of the data we receive from API calls. By mapping the incoming data to specific model fields, wecan organise and categorise the information, making it easier to access and work with. Within our application we currently have 2 key Models, the UserModel and the EventModel, both as stand alone classes, essentially they do not inherit from any parent class and have no child classes.

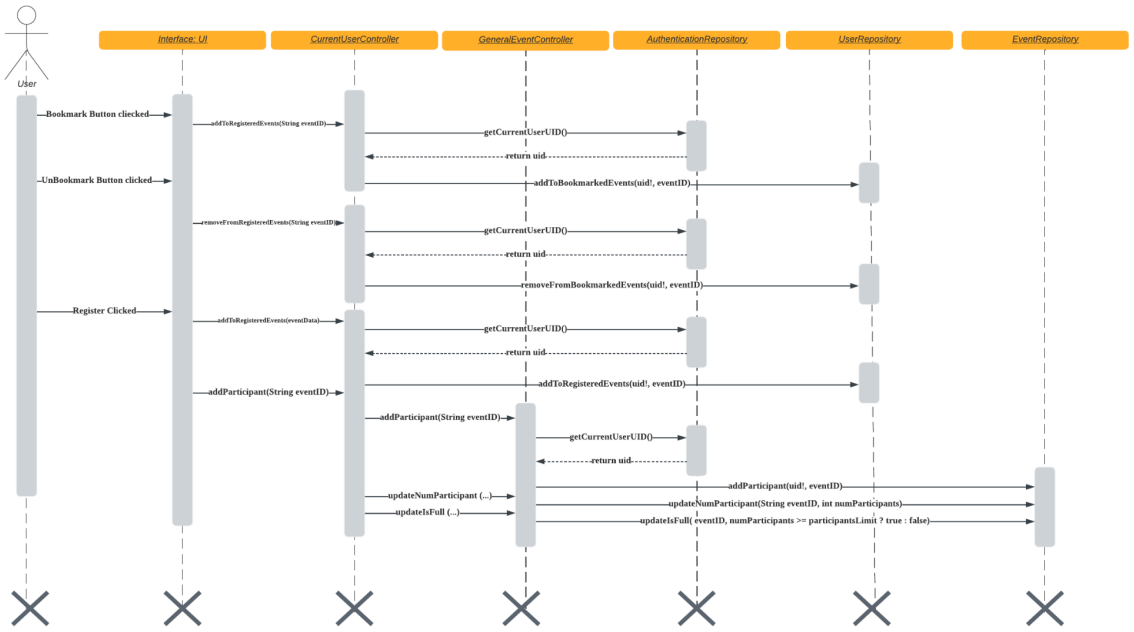


Class Diagram - UserModel & EventModel

Sequence Diagrams (Key features):

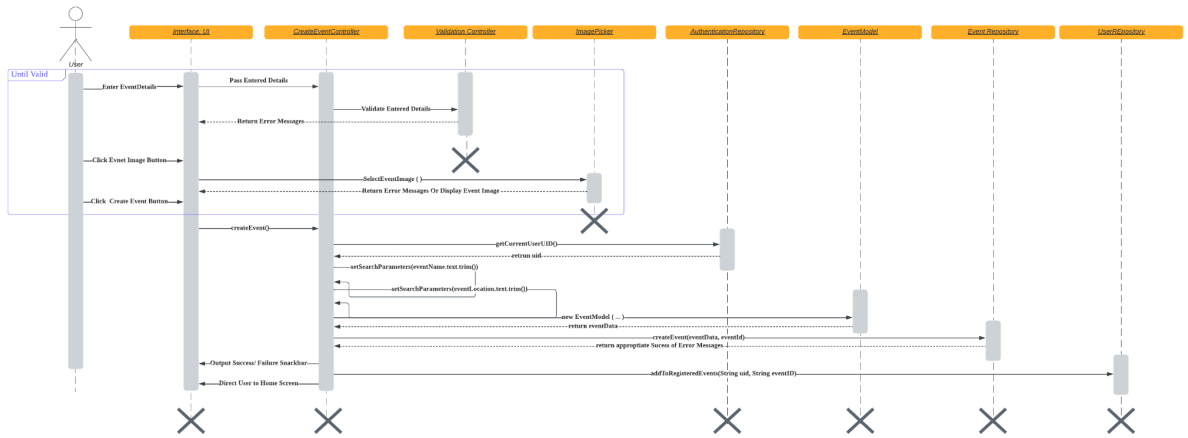


User Sign Up Sequence Diagram



User Bookmark/ Register Event Sequence Diagram

Search Event Sequence Diagram



Create Event Sequence Diagram

Quality Control :

Unit Testing: [Completed]

Unit testing plays a vital role in the automated testing process as it involves evaluating small units of code to ensure the reliability and functionality of specific parts of a program. In the case of our app, a comprehensive approach to unit testing was adopted to validate the logic behind selected functions within each of the controllers.

The unit tests for our app were carefully written to target critical functions within the controllers. These tests were designed to verify the expected behaviour of these functions, ensuring that they perform their intended tasks accurately and reliably

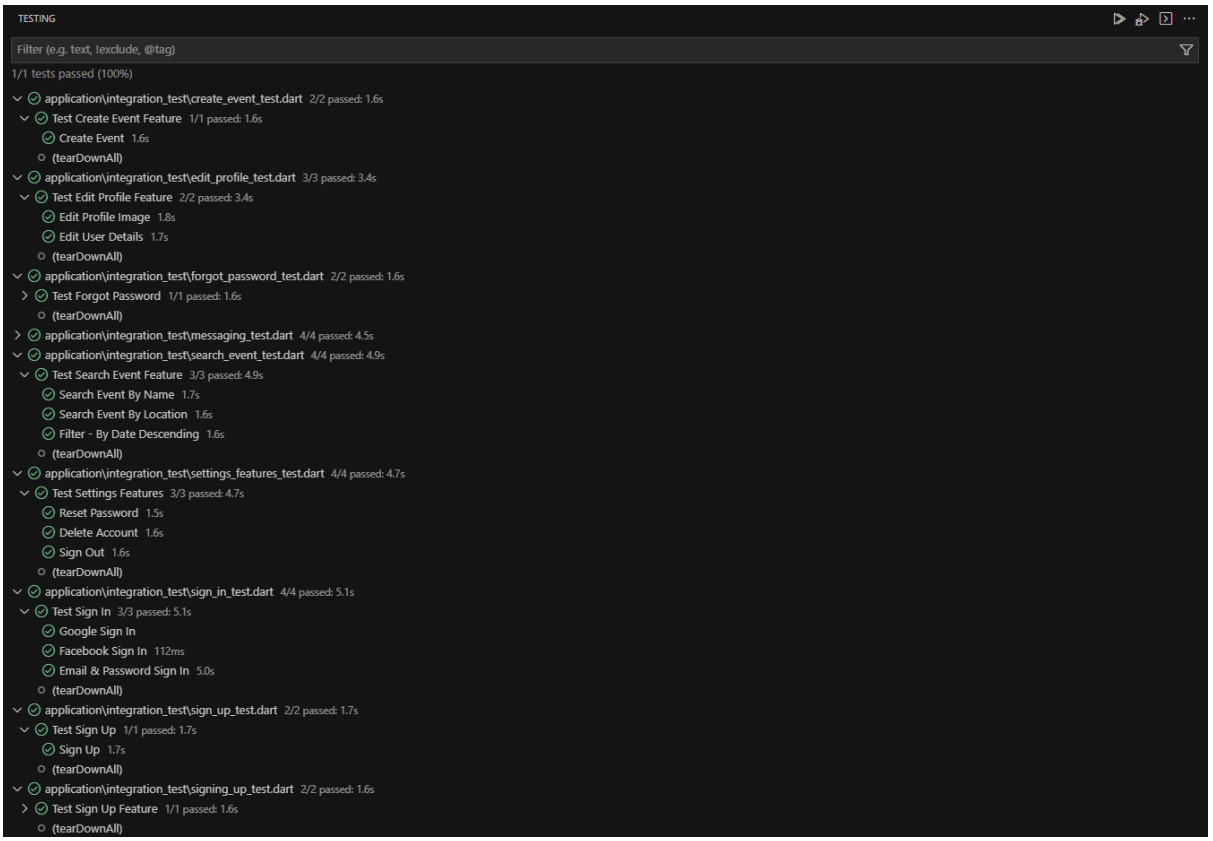
```
00:00 +1: Test Controllers Test Email Validation Empty String Handling
00:00 +2: Test Controllers Test Email Validation Invalid Email Handling
00:00 +3: Test Controllers Test Email Validation Valid Email Handling
00:00 +4: Test Controllers Test Sign In Password Validation Empty Password
00:00 +5: Test Controllers Test Sign In Password Validation Valid Password
00:00 +6: Test Controllers Test Sign Up Password Validation Empty Password
00:00 +7: Test Controllers Test Sign Up Password Validation Invalid Password Length
00:00 +8: Test Controllers Test Sign Up Password Validation Invalid Password
00:00 +9: Test Controllers Test Sign Up Password Validation Valid Password
00:00 +10: Test Controllers Phone Numeber Validation Empty String Handling
00:00 +11: Test Controllers Phone Numeber Validation Invalid Phone Number Handling
00:00 +12: Test Controllers Phone Numeber Validation Valid Phone Number Handling
00:00 +13: Test Controllers Name Validation Empty String Handling
00:01 +14: Test Controllers Name Validation Invalid Name Handling
00:01 +15: Test Controllers Name Validation Valid Name Handling
00:01 +16: Test Controllers Usename Validation Empty String Handling
00:01 +17: Test Controllers Usename Validation Invalid Name Handling
00:01 +18: Test Controllers Usename Validation Valid Name Handling
00:01 +19: Test Controllers (tearDownAll)
00:01 +20: All tests passed!
```

Test cases for Unit Testing & Their Outcomes

System Testing: [Completed]

System Testing		Outcomes	
System Test Case	Test Case Handling Mething	Fails Gracefully ?	Date
Invalid Credential Entered in any Component of the app.	Snackbar error output, telling user what exactly went wrong if identified. Allow for reentry.	Yes	07/19
Invalid image/ no image file selected for any features within the app that requires one.	Snackbar error output, telling user what exactly went wrong if identified. Requesting the user to reattempt.	Yes	07/19
Future Builder taking too long.	Utilisation of Linear Progress Bar or Circular Progress Indicator. In some cases a default text/ image.	Yes	07/19
OTP not sent via email/ phone number.	Snackbar error output, telling user that OTP code was not sent. Redirecting them to the sign in page for reattempt.	Yes	07/19
Invalid inputs entered for text form fields that needs to be validated (eg. password)	Snackbar notification, telling the user what went wrong. Allow for reentry.	Yes	07/19
Searching for an event that does not exist.	Text widget telling users that the event cannot be found. Should use other key words if any.	Yes	07/19
Registering for an event that is already full.	Snackbar notification, telling the user that the event is full. If really interested, should look to bookmark the event.	Yes	07/19
Looking for events that have just been uploaded.	Allow for users to refresh their screen.	Yes	07/19
Ensure designed screens fit all phone sizes.	Making the widget scrollable and dependent on height and width of device in the frist place.	Yes	07/19
Isn’t connected to the internet.	Default text and images in place. Snackbar warning about the cause of failure.	Yes	07/19

Integration Testing: [Ongoing]



Test cases for Integration Testing & Their Outcomes

Integration Test					Outcomes	
ID	User Story	Testing Objective	Actions	Expected Results	Pass/Fail	Date
1	As a user, I want to be able to sign in to the application using my email and password so that I can access my personalised account and data.	Be able to sign in using the standard email and password.	1. Launch the app 2. Enter existing user email and password. 3. Tap on Sign In button	Users directed to the User Dashboard Home Page	Pass	07/18
2	As a user, I want the sign-in process to support alternative login methods, such as signing in with Google or Facebook, to provide convenience and flexibility.	Be able to Sign In / Sign Up with either a Google account or Facebook Account	1. Launch the app 2. Click on Continue with Google 1. Launch the app 2. Click on Continue with Facebook	Users directed to the User Dashboard Home Page	Pass	07/18
3	As a user, I want to sign up for an account by providing my	Be able to sign up for an account.	1. Launch the app 2. Pump SignUpScreen on Integration Test.	Users directed to the User Dashboard Home Page	Pass	07/18

	name, email, username, and password, enabling me to create a unique and personalised profile.		3. Enter the required details. 4. Click on Continue 5. Key in the inputs to the bio. 6. Click on Create Account			
4	As a user, I want to have the ability to recover my account if I forget my password, using features such as password reset.	Reset password email sent to users when requested.	1. Launch the app 2. Enter existing email 3. Success Snackbar detected.	Success Snackbar detected and Users redirected to Sign In Screen.	Pass	07/18
5	As a user, I want the option to sign out or log out of my account when I'm done using the application, ensuring my account remains secure.	Be able to sign out of account once login	1. Launch the app 2. Sign in using an existing account. 3. Click on the Settings Button. 4. Click on the Sign Out button.	Users directed to the Sign In Screen	Pass	07/18
6	As a user, I want the option to delete my account by selecting the "Delete Account" option in the settings screen, giving me control over permanently removing my account and associated data from the app.	Be able to delete current logged in account.	1. Launch the app 2. Sign in using an existing account. 3. Click on the Settings Button. 4. Click on the Delete Account button. 5. Enter password of the current account.	Users directed to the Sign In Screen with success snackbar notification.	Pass	07/18
7	As a user, I want the ability to change my password by selecting the "Change Password" option in the settings screen, allowing me to update my account security and protect my information.	Reset password email sent to users when requested.	1. Launch the app 2. Sign in using an existing account. 3. Click on the Settings Button. 4. Click on the Change Password button.	Success sent email Snackbar detected.	Pass	07/19
8	As a user, I want the ability to edit my user profile by tapping on the "Edit User Profile" option in the settings screen, allowing me to	Be able to change user profile details once an account has been created.	1. Launch the app 2. Sign in using an existing account. 3. Click on the Settings Button.	Edit Profile Success Snackbar detected and Users redirected to the User Dashboard Home Page.	Pass	07/19

	update my profile image, username, name, or bio if needed		4. Click on Edit Profile Button 5. Click on Edit Button 6. Edit User Profile 7. Click on Save.			
9	As a user, I want to be able to create a new event, so that others who are keen can join.	Be able to create events and seen by other users	1. Launch the app 2. Sign in using an existing account. 3. Navigate to Create Page. 4. Click on Create Event Button. 5. Enter the required details. (With unique event name) 6. Click on Create Event Button. 7. Navigate to Search Events Page. 8. Search the created event by Name. 9. Check if Event is present.	Edit Profile Success Snackbar detected and can be found on Event Search page.	Pass	07/19
10	As a user, I want to be able to register for an event, so that I can easily indicate my interest in participating.	Be able to register for events	1. Launch the app 2. Sign in using an existing account. 3. Navigate to the Event Search Screen. 4. Select the first event. 5. Click on Register Button	Register Event Success Snackbar detected and Registered Button detected.	Pass	07/18
11	As a user, I want to be able to search for events by their name or location, so I can quickly find specific events I'm interested in attending.	Be able to search for event by name or location	1. Launch the app 2. Sign in using an existing account. 3. Navigate to Create Page. 4. Click on Create Event Button. 5. Enter the required details. (With unique event name and Location) 6. Click on Create Event Button. 7. Navigate to Search Events Page. 8. Search the created event by Name. 9. Check if Event is present.	Created Event found both by Location and Name.	Pass	07/19

			8. Search the created event by Location. 9. Check if Event is present.			
12	As a user, I want to be able to refine my search results using filter criteria	Able to filter events. (Only testing date)	1. Launch the app 2. Sign in using an existing account. 3. Navigate to Search Event Page. 4. Click on the Filter Button. 5. Select By Date & Descending 6. Exit Pop Up 7. Compare date of first 2 events	Event happening further away from the current Date and Time appears first.	Pass	07/19
13	Messaging …					

Usability Testing: [Complete]

Forewords - Usability Testing:

Our usability testing aimed to evaluate the user experience across various essential features of the app, including RSVPing, event creation, texting functionality, bookmarking events, and the effectiveness of the search feature. We carefully selected 5 participants who represent our primary target audience: NUS students. To comprehensively assess the app's usability, we designed realistic scenarios that closely align with its core functions. By crafting these tasks, we sought to replicate real-life interactions and gather valuable insights from the participants' experiences.

To ensure rich feedback, we prompted and encouraged our testers to think aloud while using the app, openly sharing their thoughts, feelings, and any challenges they faced. Throughout the test, we also closely observed participants' interactions with the app, meticulously documenting any issues, confusion, or positive experiences they encountered. Following the completion of the tasks, we conducted in-depth interviews to gather qualitative feedback. Through open-ended questions, we sought insights into their overall experience, the app's strengths, weaknesses, and suggestions for improvement.

Results Summary - Usability Testing:

A) RSVP: Asked participants to find and RSVP to a specific event happening within the next week. (Event has been created by us)

- Our observation: During the testing sessions, we noticed that users initially struggled with navigation on the User Dashboard's home page due to some horizontal scroll widgets. However, once they found the section to search for events and RSVP, the process became more straightforward. Some participants appeared unsure after registering, as if they were waiting for a confirmation, such as a vibration or a snackbar notification.

- Gathered User Feedback: Feedback from users indicated that it would be more efficient if they could search for events based on dates, in addition to event names or locations. A filtered search function allowing date selection was also suggested. Participants appreciated the clear visibility of the search functionality on the UI, finding it simple and easy to understand. However, some users felt that registering for an event was too simplistic, with just a single click of a button; and suggested making the process slightly more engaging or interactive. Overall, the feedback highlighted the importance of refining navigation elements, enhancing search capabilities, and adding a subtle confirmation or feedback mechanism to enhance the user experience during the RSVP process.

B) Event Creation: Instructed participants to create a new event, setting the event name, date, location, image, and adding event details.

- Our observation: The process of event creation was found to be quite user-friendly, with participants showing no hesitation while completing the task.

- Gathered User Feedback: Users expressed the need for the ability to delete and edit events, as well as set event end times, and link event locations to Google Maps for added convenience. They also suggested pre-adding participants during event creation and increasing the size of the event details text box, as viewing and editing text proved difficult. Additionally, participants recommended providing default event images for users without specific images and including a spellcheck feature to avoid errors.

  Additionally, participants preferred a more noticeable event creation confirmation, such as a comprehensive notification, in addition to the current snackbar. Lastly, users highlighted the importance of allowing event creators to access and keep track of their previously created events within the app.

C) Texting Functionality: Had participants send a message to the users attending the same event.

- Our observation: The texting functionality was found to be user-friendly and straightforward during the testing sessions.

- Gathered User Feedback: Participants provided valuable feedback on the app's texting functionality, highlighting several essential improvements. They expressed the desire for a feature to tag specific users within the group chat for more targeted communication. Additionally, users emphasised the importance of adding support for media attachments, enabling the sharing of images, audio, and videos. Some participants suggested enhancing the app's UI design to create a more cohesive and visually appealing experience. Moreover, they requested the inclusion of user-to-user texting for private conversations and the ability to block or mute other users when necessary. Finally, users recommended implementing app notifications for incoming messages, ensuring timely updates even when they are not actively using the app.

D) Bookmarking: Asked participants to bookmark their favourite upcoming event for quick access later.

- Our observation: Bookmarking was easily completed by participants, primarily due to the use of the universally recognizable bookmark icon.

- Gathered User Feedback: While completing the task of bookmarking their favourite upcoming event, users commended the ease of use, attributing it to the incorporation of universally recognizable icons. However, they expressed a desire for more default event images to be included, as during the event search process they noticed repetitive images that lacked visual variety. Another point of feedback highlighted the need for bookmarked events to appear automatically on the user dashboard home page, rather than requiring a manual page refresh. Participants also suggested the implementation of a comprehensive view where they can easily access and search all their bookmarked and registered events in one place.

E) Search: Requested participants to find existing events related to a specific interest or location.

- Our observation: The search feature was found to be easy to use during the testing sessions.

- Gathered User Feedback: One key recommendation was to incorporate a more detailed filter feature, allowing them to first filter events by date and then by name for example. This would enable more precise and relevant search results, making it easier for users to find events that match their preferences. Additionally, participants expressed interest in a personalised search page that presents events based on their interests and previous interactions with the app. Another enhancement suggested by users was the implementation of live location tracking, which would provide events close to their current area, offering a more localised event discovery experience.