



**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

SC2006 Software Engineering

Project Title: OpenJio



Tutorial Group: DSAI1

Team Name: Team FIVVER

Team Members:

Tio Guo Yong (Leader) (U2123181B)

Chen Zihang (Asst. Leader) (U2121486H)

Zhang Jing Wen (U2121853G)

Jayden Yeo He (U2120348J)

Nema Aarushi (U2120814C)

1. Requirement	6
1.1 Introduction	6
1.2 Overall Description	7
1.3 External Interface Requirements	9
1.4 Functional Requirements	10
1.5 Non-Functional Requirements	15
1.6 Use Case Diagram	16
1.7 Use Case Descriptions	17
1.8 Data Dictionary	37
1.9 Initial UI Mockups	38
2. Design	43
2.1 Class Diagrams	43
2.2 Sequence Diagrams	43
2.3 Dialog Map	43
2.4 Architecture	44
3. Implementation	50
3.1 Application Skeleton	50
3.2 Source Code	50
3.3 Demo Script	50
3.4 Demo Video	53
4. Testing	54
4.1 Black-Box Testing	54
4.2 White-Box Testing	65
5. References	70
6. Appendix	71

1. Requirement

1.1 Introduction

1. Purpose

This documents the software requirements and specifications for Open-Jio. Open-Jio is a mobile application that allows users to self-initiate activities and join open invitations within Singapore. Activities can range from sports and movies, to study sessions and supper runs. Users can also add other users as friends, and chat with other users. This app can be installed from the Android or IOS App store.

This document will explain the purpose and features of the systems, including the functional and non-functional requirements, design considerations, UI mock-ups, test cases, use cases, and other pertinent information about how the system operates. This document is intended for both the stakeholders and the developers of the system.

2. Document Conventions

Text formats

- a. Font: Overpass
- b. Font-style: Normal for Body and second-level Subheadings, Semi-Bold for Subheadings and Bold for Headings.
- c. Font size: 11 for Body, 12 for second-level Subheadings, 14 for Subheadings and 16 for Headings.
- d. Line height: 1.15

3. Intended Audience and Reading Suggestions

The document is intended for all the stakeholder customers and developers – designers, coders, testers and maintainers. This document need not be read sequentially; users are encouraged to jump to any section they find relevant. Below is a brief overview for each part of the document.

Part 1 (Requirement)

This section offers a summary of the Open-Jio project, including goals, objectives, project scope, general system details.

Part 2 (Design)

This section describes the Open-Jio system class by class, including interface details, class hierarchies, performance/design constraints and process details.

Part 3 (Implementation)

This section covers all of the details related to the structure of the graphical user interface (GUI), including some preliminary mockups of the Open-Jio mobile application. Readers can view this section for a tentative glimpse of what the final product will look like.

Part 4 (Testing)

This section offers a list of test cases, testing and expected output, separated into black box and white box testing. Readers can use this section to evaluate the completeness and reliability of the product.

Part 5 (References)

This section includes all the documents and resources referenced and used in our project.

Part 6 (Appendices)

This section Includes all UML diagrams, links to mockups, source code, demo video and other deliverables.

4. Product Scope

Open-Jio is an utility application in daily life which facilitates physical social activities. Its objectives include

- a. Easing open-invitations and to create a convenient and easy-to-use application for users, trying to find an activity to do on the spot.
- b. Promoting the use of technology to bond with the community.

1.2 Overall Description

1. Product Perspective

- a. The concept of an “open-jio” or open-invitation is popularised in Singapore school hostels, where students plan impromptu sports activities that other students in the hostel are free to join. These invitations are entirely facilitated through Telegram group chats, and our app Open-Jio will be the first dedicated platform of its kind.

2. Product Functions

- a. Organise Activities
- b. The app allows users to organise activities, which are categorised into sports, hangout, etc.

3. Explore Activities

- a. Users can search for activities to join by category, location and time. Upcoming activities in their vicinity will also be automatically recommended to them.

4. Friends

- a. Users can add other users as friends, allowing them to chat and view what each other are participating in.

5. User Classes and Characteristics

- a. Youths: It includes students, and young adults who are more familiar with online social interactions.
- b. Adults: It includes daily workers or any other individuals who like to join social events.

6. Operating Environment

- a. Our mobile app will operate on the IOS and Android platforms, and will require location services.
- b. For Android platforms, the OS version must be at least Android 6 (Marshmallow).
- c. For IOS platforms, the OS version must be at least IOS 9.
- d. For location services, the device must have built-in support of Global Positioning System.

7. Design and Implementation Constraints

- a. User-interface/front-end: ReactJS version 18.2.0
- b. Database/back-end: Firebase version 9.12.1
- c. Map: OneMapAPI
- d. Navigation: React Router version 6.4.1
- e. Android OS: at least Android 6 (Marshmallow).
- f. IOS: at least IOS 9.
- g. Each page must have finished loading with complete details within 5 seconds.
- h. The maximum memory used by the app must not exceed 1GB.

8. User Documentation

- a. 'How to begin' section containing basic information on how to use the app in the app store.
- b. A promotional Youtube video on how to use the app.

9. Assumptions and Dependencies

- a. OneMap API requests do not exceed 250 calls every minute.
- b. The device has built-in support for the Global Positioning System.
- c. API calls to Firebase can successfully return complete data to the application.

1.3 External Interface Requirements

1. User Interfaces

- a. The buttons on the app must have a noticeable change in colour to indicate the user is hovering through it.
- b. Upon signing in, the app must allow the user to navigate to the home page, creation of activity page, saved activities page, and profile page from any pages.

2. Hardware Interfaces

- a. The device used to open this application must support Global Positioning System for the tracking of current user's location.
- b. The device must have at least 300MB free memory for the stability of the application.

3. Software Interfaces

- a. For Android devices, the OS version must be at least Android 6 (Marshmallow).
- b. For IOS devices, the OS version must be at least IOS 9.

4. Communications Interfaces

- a. The app requires internet access.
- b. The app must be able to send an email containing a verification link to change the user's password if that user forgets his or her own password.
- c. The app must be able to send notification to a user if:
- d. Another user sends a message to that user.
- e. Another user accepts a friend request from that user.
- f. The organiser (user) of an activity approves that user's join request.
- g. The organiser (user) of an activity removes that user as a participant.
- h. The organiser (user) cancels an activity.
- i. The participant (user) leaves an activity.

1.4 Functional Requirements

1. User Account & Authentication

- a. Registration
 - i. Description: The app must allow the current user to sign up for an account if there is not one associated with their email address.
 - ii. Input: Email, username, name, password, and birthdate.
 - iii. Output: Account successfully created and stored inside database.
- b. Location Permission
 - i. Description: The app must request for the current user's permission to access their location information.
 - ii. Input: User's approval of location request.
 - iii. Output: Current location of user.
- c. Login
 - i. Description: The app must allow the current user to log into the app using the account they created.
 - ii. Input: Email, password.
 - iii. Output: After login information validated, navigate to the home page.
- d. Logout
 - i. Description: The app must allow the current user to log out of their account.
 - ii. Input: Click on 'Logout' button.
 - iii. Output: The sign-in page.
- e. Reset password
 - i. Description: The app must allow the current user to reset their password by clicking on the verification they received through their registered email.
 - ii. Input: Registered email.
 - iii. Output: Email sent to user containing verification link to input new password.

2. User Profile

- a. Update Profile
 - i. Description: The app must allow the current user to edit the information displayed in their own profile page.
 - ii. Input: Profile photo, username, name and/ or description.
 - iii. Output: Profile information updated, navigates to profile page.

- b. My Friends
 - i. Description: The app must display a list of all other Users that current User is friends with, and the list of friend requests.
 - ii. Input: Click on the “My Friends” icon.
 - iii. Output: List of friends and friend requests.
- c. My Activities
 - i. Description: The app must display a list of all activities that User has saved, joined and
 - ii. organised.
 - iii. Input: Click on the “My Activities” icon.
 - iv. Output: List of activities.
- d. My Chats
 - i. Description: The app must display a list of all the chats User has with other users.
 - ii. Input: Click on the “My Chats” icon.
 - iii. Output: List of all chats .
- e. Chats Read
 - i. Description: Inside My Chats, the app must indicate whether the chat has been read by
 - ii. current User.
 - iii. Input: -
 - iv. Output: A blue banner under all unread chats.
- f. Other User Profile
 - i. Description: The app must allow the current user to view the profile page of another user.
 - ii. Input: Click on another user’s profile photo/name.
 - iii. Output: The profile page of that user.

3. Explore Activities

- a. Search Activity
 - i. Description: The app must allow the current user to search for upcoming activities initiated by other users.
 - ii. Input: Name of activities.
 - iii. Output: List of relevant activities filtered by above.
- b. Nearby Activities
 - i. Description: The app must display to the current user all the upcoming activities in their vicinity (within 10km).

- ii. Input: Location permission.
 - iii. Output: List of relevant activities will be displayed. Location markers of activities will be shown on the map.
- c. Sort by Time (Default)
- i. Description: For all the upcoming activities in the user's vicinity (within 10km), the app will display them in order of earliest to latest.
 - ii. Input: Default, or click on Location Symbol to switch to Time Symbol, to change it from Sort by Location to Sort by Time.
 - iii. Output: Filtered activities will be displayed in the order of earliest to latest.
- d. Sort by Location
- i. Description: For all the upcoming activities in the user's vicinity (within 10km), the app will display them in order of nearest to farthest.
 - ii. Input: Click on Time Symbol to switch to Location Symbol, to change it from Sort by Time to Sort by Distance.
 - iii. Output: Filtered activities will be displayed in the order of nearest to farthest.

4. Activity Organiser

- a. Create Activity
- i. Description: The app must allow the current user to create activities that other users can join.
 - ii. Input: Activity name, description, location, time, maximum number of participants and/or activity image.
 - iii. Output: New activity created and searchable by other users.
- b. Manage Participants
- i. Description: The app must allow an organiser to approve or reject participants.
 - ii. Input: Organiser's selection of users.
 - iii. Output: Participants are rejected or approved.
- c. Update Details
- i. Description: The app must allow an organiser to change details of the activity.
 - ii. Input: Activity name, description, location, time, maximum number of participants and/or activity image.
 - iii. Output: Activity's details successfully updated.
- d. Cancel Activity

- i. Description: The app must allow an organiser to cancel the activity that is at least 30 minutes before commencing.
- ii. Input: Click on the 'Cancel' button.
- iii. Output: The activity is cancelled for all participants.

5. Activity Participant

a. Join Activity

- i. Description: The app must allow the current user to join upcoming activities initiated by other users.
- ii. Input: Click on the 'Join' button.
- iii. Output: Organiser receives a request from that user to join the activity.

b. Withdraw Join Request

- i. Description: The app must allow the current user to withdraw his or her join request for upcoming activities initiated by other users.
- ii. Input: Click on the 'Pending' button.
- iii. Output: User's request will be withdrawn and removed from pending requests in organiser's activity

c. Save Activity

- i. Description: The app must allow the current user to save an activity that the user does not intend to join.
- ii. Input: Click on the 'Save' button.
- iii. Output: The saved activity will appear in the user's list of activities.

d. Leave Activity

- i. Description: The app must allow the current user to leave an activity that he/ she has indicated their participation for.
- ii. Input: Click on the 'Leave' button.
- iii. Output: User leaves the activity successfully and the organiser is notified.

6. Chat Functions

a. Message Users.

- i. Description: The app must allow the current user to send or receive messages to and from other users.
- ii. Input: Select a user to chat with.
- iii. Output: Users successfully send messages to other users or receive messages from other users.

b. Delete Message

- i. Description: The app must allow the current user to delete messages they've sent.
- ii. Input: Click on the message User will like to delete.
- iii. Output: Message is removed from database .

7. Friend Management

a. Add Friend

- i. Description: The app must allow the current user to add another user as a friend.
- ii. Input: Send friend request to another user.
- iii. Output: The recipient receives a friend request notification, that they can then approve/ reject.

b. Remove Friend

- i. Description: The app must allow the current user to remove a friend.
- ii. Input: Click on the 'Remove friend' button.
- iii. Output: Friend relationship removed.

c. Friend's Activities

- i. Description: The app must allow the current user to view the upcoming activities participated by a friend.
- ii. Input: Click on the 'View Activity' button on a friend's profile page.
- iii. Output: List of upcoming activities participated by a friend.

8. Notifications

a. Notifications

- i. Description: User must receive notification for:
 - 1. Incomplete fields when registering, editing profile, creating activity and update activity
 - 2. Incorrect username/password
 - 3. Location permission
 - 4. New Message
 - 5. Friends:
 - a. New friend request
 - b. Accepted friend request
 - 6. Activities:
 - a. New activity Join request
 - b. Accepted as participant
 - c. Rejected as participant
 - d. Removed from event
 - e. Activity cancellation
- ii. Input: None

- iii. Output: Notification alert

1.5 Non-Functional Requirements

1. Performance Requirements

- a. The system should be able to support 1000 simultaneous users.
- b. The database should be able to store up to 10 thousand active activities, each with an image.
- c. The map should be able to render all locations within Singapore.
- d. The mean time to view a page over a 56Kbps modem connection will not exceed 3 seconds.
- e. The application will run on all web browsers.

2. Safety Requirements

- a. Users under the age of 16 will not be able to register for an account.

3. Security Requirements

- a. Users are required to login with their email and password.
- b. Users are to keep their password safe and not share it with any other people, applications, or websites under any circumstances.
- c. Users are reminded not to leak their personal information such as addresses over the app.
- d. User accounts must be verified in order to post or join activities on the application.

4. Software Quality Attributes

- a. Adaptability
 - i. The system will be able to operate on all web browsers.
- b. Maintainability
 - i. The system will be opened for extension but closed for modification.
 - ii. The system code will be easy to read and understand.
- c. Reliability
 - i. There will be backup servers in place for redundancy.

5. Business Rules

- a. Developer
 - i. Developers can remove users that do not abide by the community guidelines.

- ii. Developers can remove activities that breach the community guidelines.
- b. User
 - i. Initiators can make adjustments to the details of the activity, such as date, description and location and time.
 - ii. Initiators can manage the participants of their activity.
 - 1. Accept or reject participants.
 - 2. Remove participants once they have joined the activity.
 - iii. Initiators can cancel the activity due to unforeseen circumstances.
 - iv. Initiators can also pass on the role of host to another user.

1.6 Use Case Diagram

The complete use case diagram can be found at the following links.

	Appendix	GitHub
Use-Case	Link	Link

1.7 Use Case Descriptions

Use Case ID:	UC0001		
Use Case Name:	Register Account		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	The user creates an account by filling in information for username, name, password, birthdate, and email.
Preconditions:	-
Postconditions:	User's account successfully created.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. The user presses the "Create account" button. 2. The user fills in the required information for username, name, password, birthdate, and email. 3. The user presses the "Create account" button. 4. App authenticates the information. 5. App redirects to the home page ui. 6. App displays the message "Account successfully created."
Alternative Flows:	<ol style="list-style-type: none"> 4. App finds the information is invalid. 5. Back to step 2.
Exceptions:	The user already has an account.
Includes:	-
Special Requirements:	App must be able to check information validity within 5 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0002		
Use Case Name:	Login Account		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	The user logs into the app using their registered email and password.
Preconditions:	Valid email and password.
Postconditions:	The user's account successfully logged in.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. In the login page, the user fills in their registered email and password. 2. The user presses the "Login" button. 3. App authenticates the username and password. 4. The user logs into their account's home page.
Alternative Flows:	<ol style="list-style-type: none"> 3. App finds that the email or the password is invalid. 4. Back to step 1.
Exceptions:	-
Includes:	-
Special Requirements:	Users must be able to log in to their account within 5 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0003		
Use Case Name:	Logout Account		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	The user logs out of his/her account on the application if he/she wishes to log in to a different account
Preconditions:	-
Postconditions:	The user is able to log in using the same or another account
Priority:	-
Frequency of Use:	By the intention of the user
Flow of Events:	<ol style="list-style-type: none"> 1. In the profile page, the user presses the 'Logout' button. 2. The application system will log the user out of his/her current account.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	Users must be able to log out of their account within 5 seconds
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0004		
Use Case Name:	Reset password		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	On the login page, if the user does not remember their password, they can reset it through email.
Preconditions:	The user has an account.
Postconditions:	The user's password is reset.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. The user presses the "Forgot password" button. 2. App sends an email with a verification link to the user. 3. The user clicks the verification link. 4. The link opens the app. 5. The user is prompted to enter a new password. 6. The user enters a new password. 7. App checks that the password is valid. 8. Password is reset.
Alternative Flows:	<ol style="list-style-type: none"> 7. App checks that the password is invalid. 8. The user is prompted to enter a new password.
Exceptions:	6.EX.1 - The user's email is not linked to any valid account. 6.EX.2 - The user does not have access to the email he/she inputted.
Includes:	-
Special Requirements:	App must send the email within 10 seconds.
Assumptions:	The email belongs to the User's account.
Notes and Issues:	-

Use Case ID:	UC0005		
Use Case Name:	Display Activities Nearby		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	Application will display a list of activities that are located near the user's current location, available for him/her to join.
Preconditions:	-
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. Activities are queried wherever the user navigates to the home page. 2. Maps API is invoked and nearby activities within a 10km radius is displayed on the home page.
Alternative Flows:	<ol style="list-style-type: none"> 1. If the user has not previously granted permission to access location, the app will prompt the user again to grant permission. 2. If it is past 9pm, the app will display activities happening on the same day, as well as the following day.
Exceptions:	-
Includes:	UC0002
Special Requirements:	The user's device needs to have location tracking capability.
Assumptions:	The user has granted permission to the App to access location.
Notes and Issues:	-

Use Case ID:	UC0006		
Use Case Name:	Search Activity		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User searches for upcoming activities initiated by other users (organisers).
Preconditions:	Valid activity name.
Postconditions:	List of relevant activities filtered by above.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. The user clicks on the 'search bar' on home page. 2. The user clicks on 'activity' from drop down menu to search by activity. 3. The app displays the list of relevant activities.
Alternative Flows:	<ul style="list-style-type: none"> • In step 3, if the list is empty (no activities). <ol style="list-style-type: none"> 1. The app displays 'No activities found.'
Exceptions:	-
Includes:	-
Special Requirements:	The app must display the list of relevant activities filtered within 10 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0007		
Use Case Name:	Search User		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User searches for other users' profiles.
Preconditions:	Valid username.
Postconditions:	List of users filtered by above.
Priority:	-
Frequency of Use:	By the intention of the user
Flow of Events:	<ol style="list-style-type: none"> 4. The user clicks on the 'search bar' on the home page. 5. The user clicks on 'user' from the drop down menu to search by activity. 6. The app displays the list of relevant users.
Alternative Flows:	<ul style="list-style-type: none"> ● In step 3, if the list is empty (no users). 2. The app displays 'No users found.'
Exceptions:	-
Includes:	-
Special Requirements:	The app must display the list of users filtered within 10 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0008		
Use Case Name:	Create Activity		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User (Organiser) can organise an activity that other Users can then search for and join.
Preconditions:	-
Postconditions:	The user has created an activity, and activity is added to the activities collection in the database.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User presses the “Create Activity” button on the navigation bar. 2. The user inputs the information for Activity title and description, category, location, time, the maximum number of participants or picture (optional). 3. App checks that all information is valid. 4. User clicks on ‘Post Activity’ button 5. Activity is created.
Alternative Flows:	<ol style="list-style-type: none"> 2. Information inputted is invalid. 3. The user is prompted to input the information again.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	<ol style="list-style-type: none"> 1. User enters one of the registered locations on the OneMap API.
Notes and Issues:	-

Use Case ID:	UC0009		
Use Case Name:	Update Activity		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User (Organiser) can update an activity that they have created.
Preconditions:	Activity must exist in the database.
Postconditions:	The user has updated an activity, and activity is updated in the database.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User opens an activity that they have created. 2. User clicks on the 'update' button. 3. User can modify any field (title, description, location, time, max capacity).
Alternative Flows:	<ol style="list-style-type: none"> 4. Information inputted is invalid. 5. The user is prompted to input the information again.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	User enters one of the registered locations on the OneMap API.
Notes and Issues:	-

Use Case ID:	UC0010		
Use Case Name:	Cancel Activity		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	The user (Organiser) cancels the activity that he/she has organised.
Preconditions:	The user has created an activity that is still upcoming.
Postconditions:	Activity is cancelled.
Priority:	-
Frequency of Use:	By the intention of the user
Flow of Events:	<ol style="list-style-type: none"> 1. The user (Organiser) clicks the "Cancel Event" button in the page of the Activity he/ she has created. 2. The user is prompted to confirm the cancellation. 3. Activity is cancelled. 4. App sends a notification to all participants about the activity cancellation.
Alternative Flows:	<ol style="list-style-type: none"> 3. User chooses to not confirm the cancellation. 4. Activity is not cancelled.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	The activity being cancelled was created by the User.
Notes and Issues:	-

Use Case ID:	UC0011		
Use Case Name:	Save Activity		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	Users can save activities that they wish to attend but are not sure if they want to attend.
Preconditions:	-
Postconditions:	Activity is saved.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User clicks on the 'save' button on the activity card. 2. Activity gets added to saved activities of the user.
Alternative Flows:	<ol style="list-style-type: none"> 1. Activity is not saved. 2. Activity gets deleted by the organiser.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	User has not joined the activity yet.
Notes and Issues:	-

Use Case ID:	UC0012		
Use Case Name:	Join Activity		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User (Participant) joins an activity that another user posted.
Preconditions:	Other users have created activities to join.
Postconditions:	The participant joins the activity.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. The user (participant) searches for an activity to join. 2. The user clicks the desired activity and clicks on the 'join' button. 3. App sends a notification to the organiser who will approve/reject the request. 4. When a request is approved, the user will receive a notification that they have joined the event..
Alternative Flows:	<ol style="list-style-type: none"> 1. User clicks on activity on the Home UI or Search List UI. 2. App sends a notification to the organisers and the user who just joined.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0013		
Use Case Name:	Leave Activity		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User (Participant) can leave an activity that he/ she has joined.
Preconditions:	The user has joined as a participant in an activity.
Postconditions:	The user is no longer a participant in the activity.
Priority:	-
Frequency of Use:	-
Flow of Events:	<ol style="list-style-type: none"> 1. User (Participant) chooses from a list of activities he/she has indicated to join. 2. User clicks the “Leave Activity” button. 3. App prompts the user to confirm the withdrawal. 4. User is no longer a participant for the activity. 5. (Assuming Organiser has approved User as Participant) App sends a notification to the Organiser to inform of participant leaving.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	Activity is still upcoming and not canceled yet. The user is a Participant and not Organiser.
Notes and Issues:	-

Use Case ID:	UC0014		
Use Case Name:	Add Friend		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User can request to add another User as Friend.
Preconditions:	Users are not Friends with one another.
Postconditions:	Users become one another's Friends.
Priority:	-
Frequency of Use:	By the intention of the user
Flow of Events:	<ol style="list-style-type: none"> 1. User A clicks on "Add Friend" button on the profile page of User B. 2. User B receives requests from User A and clicks on "Accept". 3. User A and User B become Friends.
Alternative Flows:	<ol style="list-style-type: none"> 1. User B receives requests from User A and clicks on "Reject". 2. User A is notified that User B has rejected his/her request.
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0015		
Use Case Name:	Remove Friend		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User can remove another User who is a 'Friend'.
Preconditions:	Users are Friends with one another.
Postconditions:	Users are no longer one another's Friends.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User A clicks on "Remove Friend" button on the profile page of User B. 2. User B is removed from User A's Friend list and User A is removed from User B's Friend list.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	User A and User B are friends from before.
Notes and Issues:	-

Use Case ID:	UC0016		
Use Case Name:	Reject Friend Request		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User can remove another User who is a 'Friend'.
Preconditions:	Users are Friends with one another.
Postconditions:	Users are no longer one another's Friends.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User A clicks on "Reject Friend Request" button on the profile page of User B. 2. User B is not added to User A's Friend list.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0017		
Use Case Name:	View Friends' Activities		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User can view friends' activities.
Preconditions:	Users have friends, friends have joined an activity.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User clicks the "Friends" tab to view all friends and their current activity. 2. If the friend is participating in an activity, it will be reflected under his username.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0018		
Use Case Name:	Chat		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	The user can send messages to another User.
Preconditions:	-
Postconditions:	The user sends a message to another User.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User selects another user to send a message to. 2. The user is prompted to type the message. 3. User presses the "Send" button. 4. The message is sent to the selected User. 5. App sends a notification to the selected User that he/she has a new message.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0019		
Use Case Name:	Edit Profile		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User
Description:	User edits the information displayed in their own profile page.
Preconditions:	-
Postconditions:	The user's information on the profile page is successfully updated.
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. The user presses the "Edit profile" button. 2. The user selects the required section to be changed: Profile photo, username, name and/ or description. 3. The user fills in the new information. 4. The user presses the "Save changes" button. 5. App authenticates the format of the information. 6. App redirects to profile page. 7. The user's information on the profile page is successfully updated.
Alternative Flows:	<ol style="list-style-type: none"> 5. App finds that the information is invalid. 6. Back to step 3.
Exceptions:	-
Includes:	-
Special Requirements:	App must be able to update information within 5 seconds.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	UC0020		
Use Case Name:	Manage Participants		
Created By:	Fivver	Last Updated By:	Fivver
Date Created:	30th Aug 2022	Date Last Updated:	6th Nov 2022

Actor:	User (Organiser)
Description:	User (Organiser) can accept or reject requests to join the activity from other users. The user (Organiser) can also remove users that are already taking part in the activity.
Preconditions:	The user has created an activity.
Postconditions:	-
Priority:	-
Frequency of Use:	By the intention of the user.
Flow of Events:	<ol style="list-style-type: none"> 1. User (Participant) sends a request to join the activity by pressing the "Join Activity" button on the activity details page. 2. The user (Organiser) can choose to "Accept" the request from the other user. 3. The user (Participant) will be added to the list of participating Users on the activity page. 4. App sends a notification to the participant of the acceptance. 5. The user (Organiser) can choose to remove a participating User by pressing the "Remove" button next to the name. 6. App sends a notification to the user of the removal.
Alternative Flows:	<ol style="list-style-type: none"> 2. User (Organiser) can choose to "Reject" the request from the other user. 3. App sends a notification to the user of the removal.
Exceptions:	10.EX.1 - User (Organiser) cannot accept a request as the participant's limit of the activity has been reached.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

1.8 Data Dictionary

1. User Account

A user account is a location on a network server used to store a computer username, password, and other information. A user account allows or does not allow a user to connect to a network, another computer, or other share. Any network that has multiple users requires user accounts.

2. Mobile Operating System

A mobile operating system, also called a mobile OS, is an operating system that is specifically designed to run on mobile devices such as mobile phones, smartphones, PDAs, tablet computers and other handheld devices.

3. Database

A database is a collection of information that is organised so that it can easily be accessed, managed, and updated. In one view, databases can be classified according to types of content.

4. Friend

A friend of a user is another user that has been connected to him/her. Adding a friend means the users can see one another's upcoming activities.

5. Activity

An activity is an event initiated and administered by the organiser. Other users can request to join the activity but only the organiser can accept their requests. The activity will include details such as location, date, type of activity as well as participants limit.

6. Organiser

An organiser is an user that initiates the activity. He will provide the details of the activity and be able to accept or reject other users' requests to participate in the activity. He will also be able to edit the details of the activity.

7. Participant

A participant is a user whose request has been approved by the organiser and is taking part in the activity. The participant can withdraw from the activity if he/she wants to do so.

8. Profile page

A page displaying the details of a user that he/she wants to share with other users. These details include profile photo, username, name and/ or description.

9. UI (User Interface)

The means by which the user interacts with the software program. This can include different pages of the application, including (but not limited to) Home page UI, Profile page UI.

1.9 Initial UI Mockups

The complete initial UI mockups can be found at the following Figma link:

<https://www.figma.com/file/67DBMydzZOW3QdWo10AR0p/SC2006-OpenJio?node-id=0%3A1>

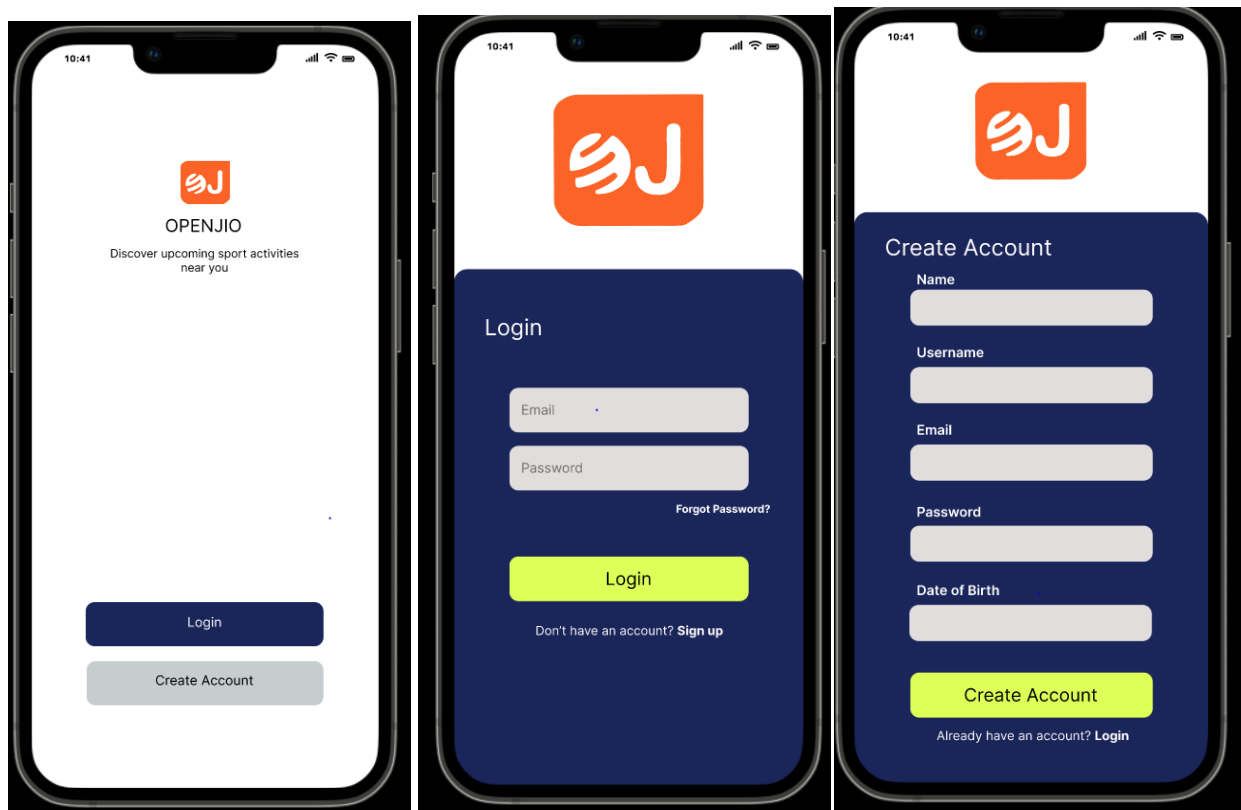


Fig. 1.1: *Splash screen, Login screen, and Registration screen*

The figure above shows the user interface for login and registration. The design of the screen is kept simple and only essential information is displayed on the screen. This will enable the user to have a seamless onboarding experience with the OpenJio App.

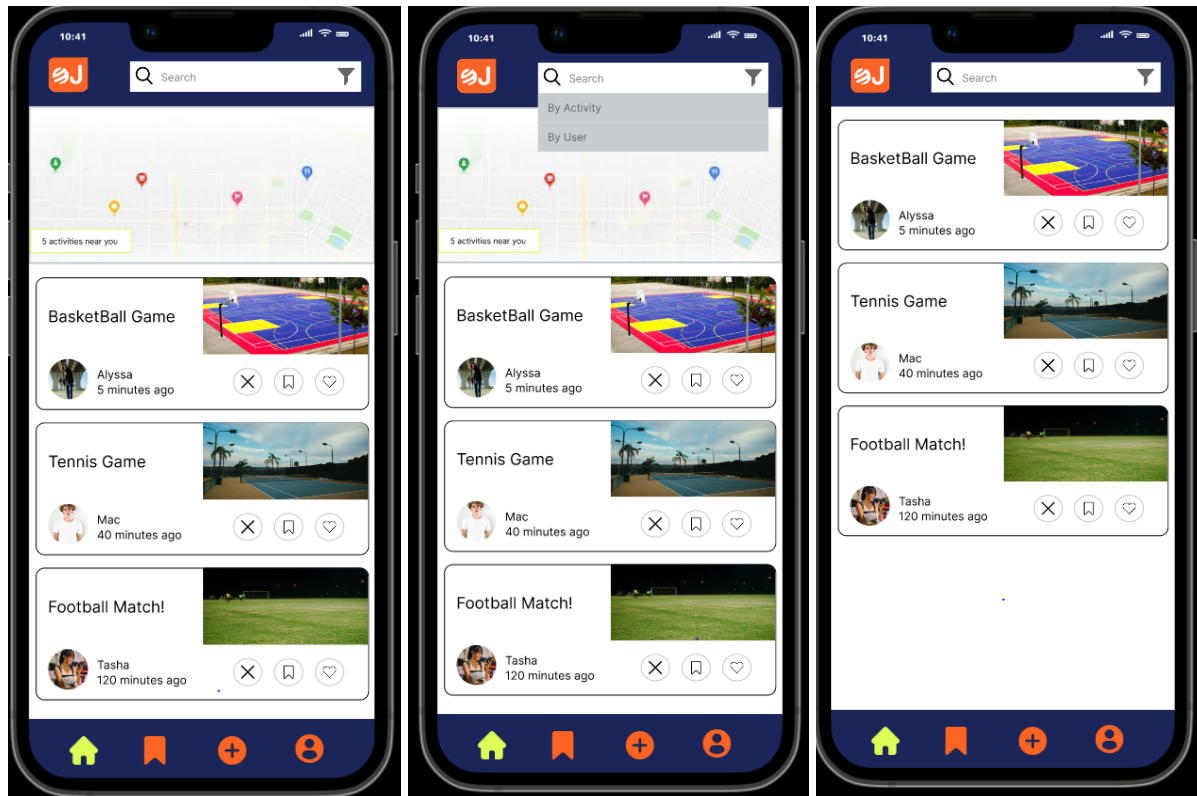


Fig. 1.2: *Home Page UI, Search UI, and Search Results UI*

The Home Page UI contains recommended activities based on the user's current location and the search bar. The user interface of these screens has appealing colours and informative activity cards. Users can search based on activity and use. A list of matching search results is displayed once the user hits 'enter.'

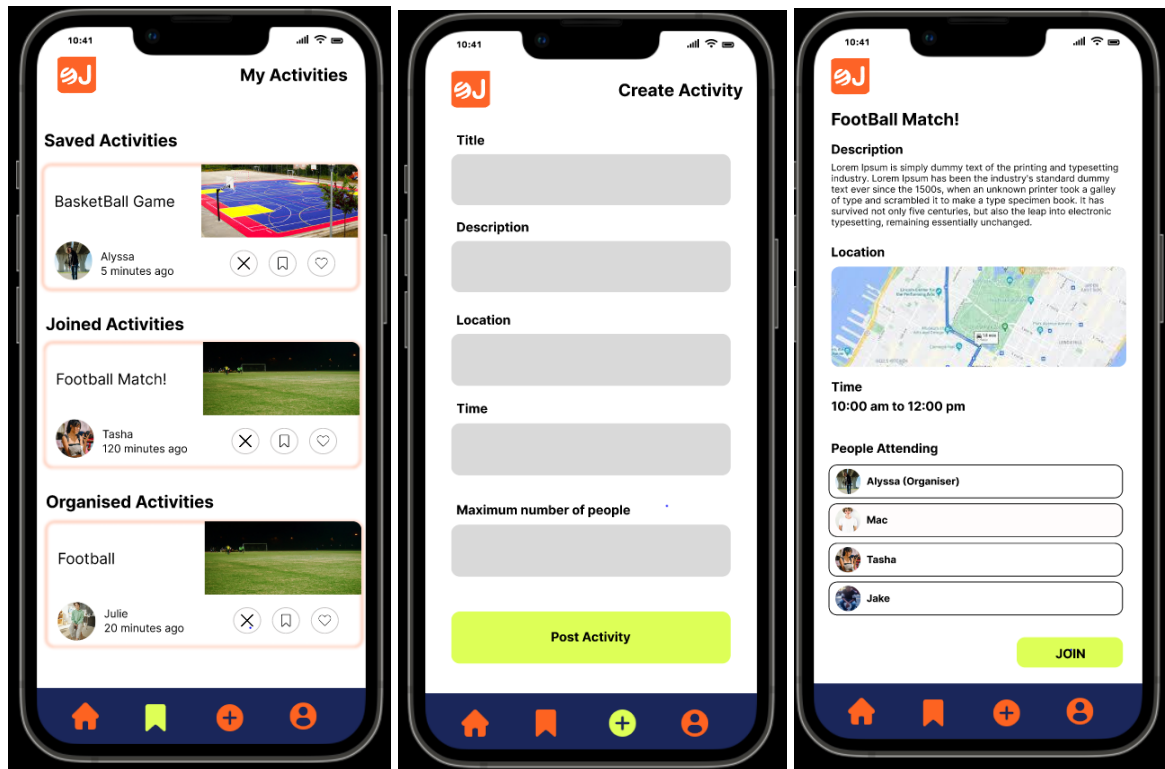


Fig.1.3: *My Activities UI, Create Activities UI, and Activity UI*

In MyActivitiesUI, users can view their “saved, joined and organised” activities all conveniently in one location. Users can also initiate activities themselves, and both of these are tabs located right in the navigation bar. In the ActivityUI, the activity’s location is clearly shown on a live map, providing a visual guide for users. All the other attendees are also listed along with their names and profile photos.

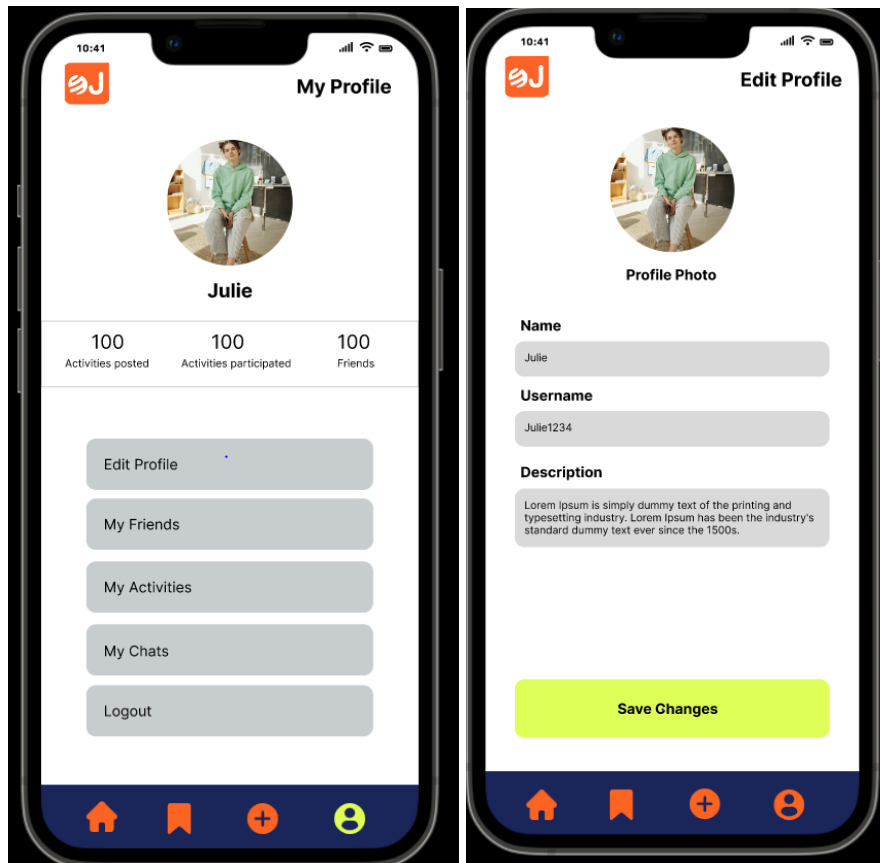


Fig.1.4: *My Profile UI and Edit Profile UI*

In Profile UI, users can view their number of activities participated, activities posted and friends. They can see their friends list, activity (saved, joined, and organised) list, and chats list. The logout feature can also be found in this UI. The user can also edit their profile details in the edit profile UI.

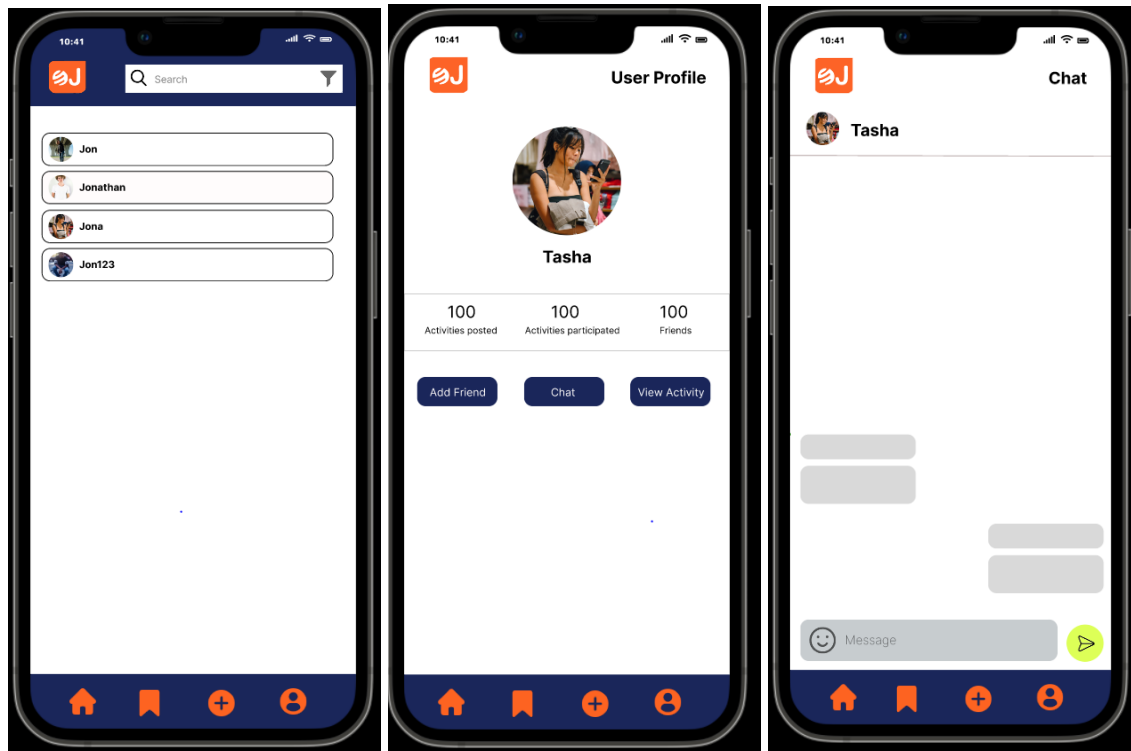


Fig.1.5: *SearchUI, Friends Profile UI, Chat UI*

In SearchUI, users can search for friends, and then view their profile. User profiles will display their number of friends, activities posted and participated. These numbers will be a basis for comparison between friends, and incentivise them to use the app more actively. Users can also add friends, view their activities, and initiate a message.

2. Design

2.1 Class Diagrams

The complete class diagrams can be found at the following links.

Type	Appendix	GitHub
Conceptual	Link	Link
Dynamic	Link	Link

2.2 Sequence Diagrams

The complete sequence diagrams can be found at the following links.

Type	Appendix	GitHub
Login, Registration & Reset Password	Link	Link
Edit Profile	Link	Link
Organise Activity	Link	Link
Join & Leave Activity	Link	Link
View & Add Friends	Link	Link
Chat	Link	Link

2.3 Dialog Map

The complete dialog map can be found at the following links.

	Appendix	GitHub
Dialog Map	Link	Link

2.4 Architecture

1. System Architecture Diagram

The complete system architecture diagram can be found at the following links.

	Appendix	GitHub
SA Diagram	Link	Link

2. Design Pattern

a) Reusability

Our application was developed using ReactJS as the front-end's framework. This framework was based on component-based programming. This allowed us to create reusable components such as a navigation bar. Hence, the same component could be applied in different components.

For example, the navigation bar component (Fig. 2.1) had been used in 13 other components of the application such as the HomeUI (Fig. 2.2) and ProfileUI (Fig. 2.3) components.

```

1  import './index.css'
2  import HomeLogo from '../Images/HomeLogo'
3  import MyActivityLogo from '../Images/MyActivityLogo'
4  import CreateActivityLogo from '../Images/CreateActivityLogo'
5  import ProfileLogo from '../Images/ProfileLogo'
6  import { memo } from 'react'
7  import { Link } from 'react-router-dom'
8
9  /**
10   * The navigation bar at the bottom of all pages once user signs in.
11   * @returns The HTML elements to be displayed on the website.
12   */
13  const NavBar = () => {
14    return (
15      <div className='NavBar'>
16        <Link to='/home'><HomeLogo /></Link>
17        <Link to='/myactivity'><MyActivityLogo /></Link>
18        <Link to='/createactivity'><CreateActivityLogo /></Link>
19        <Link to='/profile'><ProfileLogo /></Link>
20      </div>
21    )
22  }
23
24  export default memo(NavBar)

```

Fig. 2.1: The code for the navigation bar component.

```

231         {sortBy === 'distance' ?
232             <MdLocationOn onClick={() => setSort('time')} /> :
233             <MdAccessTimeFilled onClick={() => setSort('distance')} />}
234         </div>
235         {activities?.map((activity, index) =>
236             <ActivityCard key={index}
237                 uid={activity.uid}
238                 name={activity.title}
239                 bgImage={activity.bgImage}
240                 userImage={activity.userImage}
241                 organiser={activity.organiser}
242                 time={activity.time}
243                 rerender= {(()) => fetchActivities()}
244                 saved = {activity.saved}
245             />
246         )}
247         <div className='space'></div>
248         <NavBar />
249     </div>
250
251 )
252 }
253
254
255 export default memo(HomeUI)

```

Fig. 2.2: The code for the HomeUI component containing the navigation bar component at line 248.

```

85         <h3>{myProfile.numPost}</h3>
86         <p>Activities posted</p>
87     </div>
88     <div className='participated'>
89         <h3>{myProfile.numParticipate}</h3>
90         <p>Activities participated</p>
91     </div>
92     <div className='friend'>
93         <h3>{myProfile.numFriends}</h3>
94         <p>Friends</p>
95     </div>
96 </div>
97 <div className='content'>
98     <Link to='/editprofile'><button id='edit'>Edit profile</button></Link>
99     <Link to='/friend'><button id='friend'>My Friends</button></Link>
100    <Link to='/myactivity'><button id='activities'>My Activities</button></Link>
101    <Link to='/mychat'><button id='chat'>My Chats</button></Link>
102    <button id='Logout' onClick={signOut}>Logout</button>
103 </div>
104 <NavBar />
105 </div>
106 )
107 }
108
109 export default memo(ProfileUI)

```

Fig. 2.3: The code for the ProfileUI component containing the navigation bar component at line 105

b) Traceability

As our requirements were atomised, we could easily trace each requirement across the Software Development Life Cycle (SDLC).

For example, the use case of updating a user's profile can be easily traced from each phase back to the requirement and from the requirement.

- 1) Requirements phase: verifiable requirements for the use case of updating a user's profile ([User Profile](#)).
- 2) Design phase: sequence diagram to depict the flow of users updating their profile ([Edit Profile](#))
- 3) Implementation phase: Specific component and user interface for the users to update their profile. (Fig. 2.5 and Fig. 2.6)

```
10  /**
11   * The boundary class for edit profile page.
12   * @returns The HTML elements to be displayed on the website.
13   */
14  const EditProfileUI = () => {
15    const user = auth.currentUser
16    const navigate = useNavigate();
17    const fetched = useRef(false)
18
19    /**
20     * type : User's profile object.
21     * profile : The user profile's details.
22     * setProfile : Update user profile's details.
23     * name : The name of the user.
24     * username : The username of the user.
25     * description : The description of the user.
26     */
27    const [profile, setProfile] = useState({name: 'Enter new name',
28                                             username: 'Enter new username',
29                                             description: 'Enter new description'})
30    /**
31     * imageURL : The profile photo of the user.
32     */
33    const [imageURL, setImageURL] = useState('https://firebasestorage.googleapis.com/v0/b/openjio-560ed
```

Fig. 2.5: The code for the EditProfileUI component.

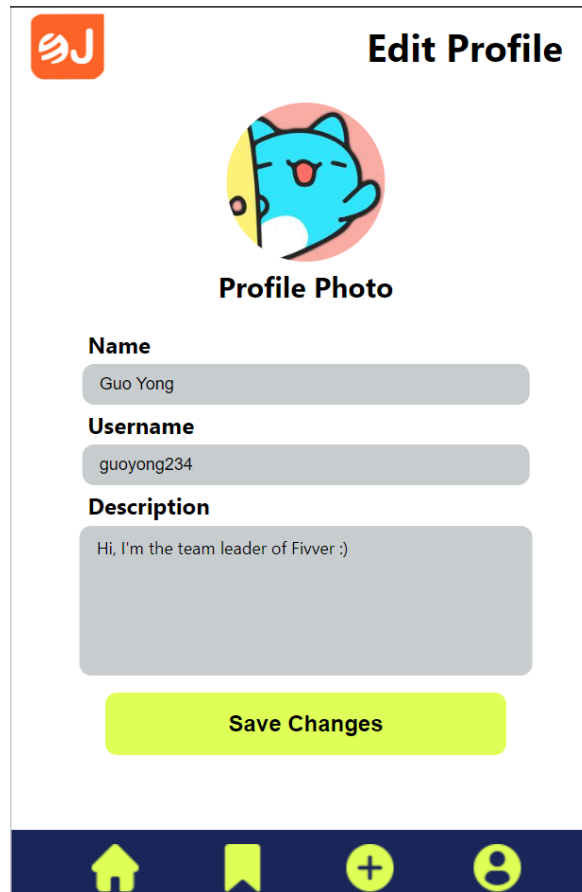


Fig. 2.6: The user interface for the *EditProfileUI* component.

- 4) Testing phase: Specific test cases to test the functionality of updating a user's profile through black-box testing (Functionality m under [Black-Box Testing](#)) and white-box testing (Functionality g under [White-Box Testing](#)).

c) Extensibility

As we used model-view-controller architecture in the software development, the interaction between boundary classes and the entity classes or databases must be performed through the control classes. This allowed us to add more method calls in the control classes to increase the functionalities easily without modifying boundary classes and entity classes. Different boundary classes can also interact with the control classes. Control classes can also interact with different entity classes.

For example, the *ActivityUI* component (boundary class) and the *HomeUI* component (boundary class) can call the methods inside *ActivityController* class (control class) to fetch the relevant activities data from the database. (Fig. 2.8, Fig. 2.9, Fig. 2.10) As *ActivityController* class is independent of boundary classes and entity classes, we can add in more methods inside it.

```

2  * ActionController class managing all activity-related actions.
3  */
4  class ActionController{
5      /**
6       * Fetch the activity info by activity ID.
7       * @param {String} activityID The activity ID.
8       */
9      async fetchInfo(activityID){ }
10
11     /**
12      * Fetch the activities by activity ID.
13      * @param {String[]} activityIDs An array of activity IDs
14      */
15     async fetchList(activityIDs){ }
16
17     /**
18      * The activity action that can be performed by a general user/participant.
19      * such as join, leave and save activity.
20      * @param {String} type Type of action.
21      * @param {String} activityID Activity ID.
22      * @param {String} userID User ID performing the action.
23      */
24     async activityActionByParticipant(type, activityID, userID){ }
25
26     /**
27      * The activity action that can be performed by the organiser.
28      * such as update, approve, reject, and delete activity.
29      * @param {String} type Type of action.
30      * @param {String} activityID Activity ID.
31      * @param {String} organiserID Organiser's user ID.
32      */
33     async activityActionByOrganiser(type, activityID, organiserID){ }
34 }

```

Fig. 2.8: *ActivityController class containing methods used by boundary classes.*

```

19  const ActivityUI = () => {
20
21      /**
22       * Fetch the activity's info from the database.
23       * @param {String} activityID The activity ID.
24       */
25      const fetchActivityInfo = async (activityID) => {
26          // Call the fetch activity info in the ActionController class.
27          fetchInfo(activityID)
28      }
29
30      /**
31       * The organiser/participant performs an action related to this activity.
32       * such as join, leave, approve, reject
33       * @param {String} type Type of action.
34       * @param {String} activityID Activity ID.
35       * @param {String} userID User ID performing the action.
36       * @param {String} role The role of the user performing the action
37       */
38      const activityAction = async (type, activityID, userID, role) => {
39          // Call the activity action in the ActionController class.
40          if(role === 'participant')
41              activityActionByParticipant(type, activityID, userID)
42          else if(role === 'organiser')
43              activityActionByOrganiser(type, activityID, userID)
44      }

```

Fig. 2.9: *The methods inside ActivityUI calling methods inside the ActionController class.*

```
17  /**
18   * The boundary class for home page.
19   * @returns The HTML elements to be displayed on the website.
20   */
21  const HomeUI = () => {
22
23    /**
24     * Fetch the activities from the database.
25     * @param {String[]} activitiesID An array of activity IDs.
26     */
27    const fetchActivities = async (activitiesID) => {
28      // Call the fetch activity list in the ActivityController class.
29      fetchList(activitiesID)
30    }
```

Fig. 2.10: *The methods inside HomeUI calling methods inside the ActivityController class.*

3. Implementation

3.1 Application Skeleton

The complete application skeleton can be found at the following GitHub link:

https://github.com/YoNG-Zaii/OpenJIO_Fivver/tree/main/Implementation/Application%20Skeleton

3.2 Source Code

The complete source code can be found at the following GitHub link:

https://github.com/YoNG-Zaii/OpenJIO_Fivver/tree/main/Implementation/Source%20Code

3.3 Demo Script

1. Registration
 - a. Navigate to sign up
 - b. Insert Name: *Thomas*
 - c. Insert Username: *thomas*
 - d. Insert Email: *thomas@gmail.com*
 - e. Insert Password: *123456*
 - f. Choose D.O.B: *1st January 2002*
 - g. Click on Create Account
 - h. **Redirected to Home**
2. Location Permission
 - a. Allow Permission Access
3. Edit Profile
 - a. Navigate to profile
 - b. Click on Edit Profile
 - c. Upload photo: *chicken.jpg*
 - d. Change Description: *I like trains!*
 - e. Click on Save Changes
 - f. **Redirected to Profile**
 - g. Show new profile with name

4. Home

- a. Navigate to Home
- b. Show all activities within a 5km radius
- c. Show all the filtered activities sorted by time
- d. Toggle Hourglass Logo to Location Logo
- e. Show all the filtered activities sorted by distance
- f. Show the map with the activity markers

5. Search Activity

- a. Input search bar: ball
- b. Search by Activity
- c. Click on search
- d. **Redirected to Search Results**

6. Activity Actions

- a. Save activity: volleyball organised by Caleb
- b. Click on activity: football organised by Charles
- c. Navigate into the activity
- d. Click on Join to request to join
- e. Click on Pending to retract request
- f. Click on Join to request to join
- g. **Approved by Charles**
- h. Click on Leave to leave activity
- i. Click on Join to request to join
- j. **Approved by Charles**

7. Create Activity

- a. Navigate to activity creation page
- b. Insert Name: *Group Run*
- c. Insert Description: *Run with me!*
- d. Insert Location: *nanyang community club*
- e. Choose Time: *04/11/2022*
- f. Insert Maximum capacity: 5
- g. Click on Post Activity
- h. **Redirected to Home**

8. Update Activity

- a. Navigate to activity page
- b. Click on Update

- c. Upload Photo: *run.jpeg*
- d. Click on Update
- e. **Redirected to Activity**

9. Organised Activity

- a. **Charles, Jayden, Jayden Chua join activity**
- b. Reject Jayden
- c. Approve Charles, Jayden Chua
- d. Remove Charles
- e. Navigate to Profile to show number of activities

10. My Activities

- a. Navigate to My Activities
- b. Navigate to Group Run Activity

11. Cancel Activity

- a. Click on cancel to cancel activity
- b. **redirected to Home**

12. Adding Friend

- a. Navigate to softball activity
- b. Click on Caleb
- c. **Redirected to Caleb's Profile page**
- d. Click on add friend

13. Search User

- a. Input search bar: Jayden
- b. Search by User
- c. Click on search
- d. **Redirected to Search Results**
- e. Click on Jayden Chua
- f. **Redirected to Jayden Chua's Profile page**

14. Friend Actions

- a. Click on add friend
- b. Navigate to My Friends
- c. **Charles, Jayden Pang send friend requests**

- d. *Reject Jayden Pang*
- e. *Approve Charles*
- f. **Jayden Chua accepts friend request**

15. Friend's Activities

- a. Navigate to Jayden Chua's Profile
- b. Click on View Activity to view Jayden Chua's upcoming activities

16. Chat

- a. Navigate to Profile
- b. Click on My Chats
- c. **Jayden Chua sends message: hello who are you**
- d. Opens chat with Jayden Chua
- e. Reply: were you at soccer yesterday?
- f. **Jayden Chua sends message: it wasn't me lol**
- g. Delete message: were you at soccer yesterday?

17. Delete Friend

- a. Navigate to My Friends
- b. Delete Jayden Chua

18. Logout

- a. Navigate to profile
- b. Click on Logout
- c. **Redirected to Login page**

3.4 Demo Video

The complete demo video can be found at the following Youtube link:

<https://youtu.be/QcDMia40I3M>

4. Testing

4.1 Black-Box Testing

a) Functionality: Register Account

Boundary values:

1. Password length must be ≥ 6 characters
2. Username length must ≤ 15 characters
3. Date of birth at least five years before ≤ 2017

a. Generic cases:

Test ID	Test Case	Expected Result	Actual Result
1.a.1	User registers with incomplete input fields	The system prompts the user to enter missing fields	The system prompts the user to enter missing fields
1.a.2	User registers with already registered email	The system displays 'email-already-in-use' error message	The system displays 'email-already-in-use' error message
1.a.3	User registers with password length < 6 characters	The system displays 'weak-password' error message; user prompted to enter password of length ≥ 6 characters	The system displays 'weak-password' error message; user prompted to enter password of length ≥ 6 characters
1.a.4	User registers with name, valid username, valid email, password and date of birth	The system directs the user to the home page ui	The system directs the user to the home page ui

b. Specific cases:

Test ID	Test Case	Expected Result	Actual Result
1.b.1	Input fields: 1. Name: Julie 2. Username: Julie123 3. Email: EMPTY 4. Password: Juser123	The system prompts the user to enter missing fields	The system prompts the user to enter missing fields

	5. Date of birth: 26-07-2002 Click on 'Create Account' button		
1.b.2	Input fields: 1. Name: Julie 2. Username: Julie123 3. Email: newuser@gmail.com 4. Password: Juser123 5. Date of birth: 26-07-2002 Click on 'Create Account' button	The system displays 'email-already-in-use' error message	The system displays 'email-already-in-use' error message
1.b.3	Input fields: 1. Name: Julie 2. Username: Julie123 3. Email: newuser@gmail.com 4. Password: Juser 5. Date of birth: 26-07-2002 Click on 'Create Account' button	The system displays 'weak-password' error message; user prompted to enter password of length ≥ 6 characters	The system displays 'weak-password' error message; user prompted to enter password of length ≥ 6 characters
1.b.4	Input fields: 1. Name: Julie 2. Username: Julie123 3. Email: Julie@gmail.com 4. Password: Juser123 5. Date of birth: 26-07-2002 Click on 'Create Account' button	The system directs the user to the home page ui	The system directs the user to the home page ui

b) Functionality: Logout Account

Steps: Click on user profile → Click on 'Logout' button

Test ID	Test Case	Expected Result	Actual Result
3.1	User clicks on 'Logout' under 'My Profile'	The system logs out user from application and directs them to 'Login' ui	The system logs out user from application and directs them to 'Login' ui
3.2	User clicks on 'back' after	User is redirected to the	User is redirected to

	logging out	login page	the login page
--	-------------	------------	----------------

c) Functionality: Reset Password

The user is on the 'Login' UI

Test ID	Test Case	Expected Result	Actual Result
4.1	User clicks on forgot password	User receives an email containing password reset link in his/her registered email	User receives an email containing a password reset link in his/her registered email

d) Functionality: Login Account

a. Generic cases:

Test ID	Test Case	Expected Result	Actual Result
2.a.1	User logs in with incomplete input fields	The system prompts the user to enter missing fields	The system prompts the user to enter missing fields
2.a.2	User logs in with unregistered email or password	The system displays 'user-not-found' error message	The system displays 'user-not-found' error message
2.a.3	User logs in with registered email and password	The system directs the user to the home page ui	The system directs the user to the home page ui

b. Specific cases:

Test ID	Test Case	Expected Result	Actual Result
2.b.1	Input Fields: 1. Email: Julie@gmail.com 2. Password: EMPTY	The system prompts the user to enter missing fields	The system prompts the user to enter missing fields
2.b.2	Input Fields:	The system displays	The system displays

	1. Email: J@gmail.com 2. Password: JUser123	'user-not-found' error message	'user-not-found' error message
2.b.3	Input Fields: 1. Email: <u>Julie@gmail.com</u> 2. Password: JUser123	The system directs the user to the home page ui	The system directs the user to the home page ui

e) Functionality: Display activities nearby

Test ID	Test Case	Expected Result	Actual Result
5.1	User lands on 'Home Page' UI and provides location permission	Activities that will happen within 10 km of user's current location will be displayed	Activities that will happen within 10 km of user's current location will be displayed
5.2	User lands on 'Home Page' UI and does not provide location permission	Upcoming activities within the next five days will be displayed	Upcoming activities within the next five days will be displayed

f) Functionality: Create Activity

The user clicks on the 'Create Activity' button on the navigation bar.

Boundary value:

1. I. Date: \geq current date
2. II. Max capacity ≥ 1
3. III. Description ≤ 100 character
4. IV. Title ≤ 15 characters

a. Generic cases:

Test ID	Test Case	Expected Result	Actual Result
8.1	User posts activity with incomplete input fields	User is prompted to enter missing field	User is prompted to enter missing field
8.2	User enters activity title, location, time, maximum capacity	New activity is posted	New activity is posted

8.3	User enters activity image, title, description, location, time, maximum capacity	New activity is posted	New activity is posted
-----	--	------------------------	------------------------

b. Specific cases:

Test ID	Test Case	Expected Result	Actual Result
8.3	Input fields: 1. Title: Swimming Relay 2. Location: EMPTY 3. Time: 08-11-2022 5:00pm 4. Maximum capacity: 10	User is prompted to enter location	User is prompted to enter location
8.1	Input fields: 1. Title: Swimming Relay 2. Location: NTU SRC 3. Time: 08-11-2022 5:00pm 4. Maximum capacity: 10	New activity is posted	New activity is posted
8.2	Input fields: 1. Image 2. Title: Swimming relay 3. Description: Let's have a replay! 4. Location: NTU SRC 5. Time: 08-11-2022 5:00pm 6. Maximum capacity: 10	New activity is posted	New activity is posted

g) Functionality: Update Activity

Steps: Go to 'My activities' → Under 'Organising activities' click on the activity card that you want to update

a. Generic cases:

Test ID	Test Case	Expected Result	Actual Result
9.1	User deletes required field but does not enter new value	The system prompts the user to enter missing field	The system prompts the user to enter missing field

9.2	User updates any/all activity field(s)	When user clicks on Activity card, updated fields is displayed	When user clicks on Activity card, updated fields is displayed
-----	--	--	--

b. Specific cases:

Test ID	Test Case	Expected Result	Actual Result
9.1	Field edited: 1. Location: NTU SRC → EMPTY	The system prompts the user to enter location	The system prompts the user to enter location
9.2	Field edited: 1. Time: 08-11-2022 5:00 pm → 10-11-2022 5:00 pm	When user clicks on Activity card, updated fields is displayed	When user clicks on Activity card, updated fields is displayed

h) Functionality: Search Activity

Flow of events: The user clicks on the search bar on the Home Page UI and select 'ByActivity'

a. Generic cases:

Test ID	Test Case	Expected Result	Actual Result
6.1	User clicks on the search bar on the home page and enters posted activity name	Activities whose title matches the searched title is displayed	Activities whose title matches the searched title is displayed
6.2	User clicks on the search bar on the home page and enters invalid/nonexistent activity name	No activity is displayed	No activity is displayed

b. Specific cases:

Test ID	Test Case	Expected Result	Actual Result
---------	-----------	-----------------	---------------

6.1	Input: 1. Badminton 2. Table Tennis	Activities whose title matches the searched title is displayed	Activities whose title matches the searched title is displayed
6.2	Input: 1. Pool	No activity is displayed	No activity is displayed

i) Functionality: Save Activity/ Join Activity/ Withdraw Join Request & Leave Activity (Participant Side)

Test ID	Test Case	Expected Result	Actual Result
10.1	User clicks on 'Save' button on the activity card (Prerequisite: User has not already saved the activity)	Activity is saved and can be viewed under 'My Activities'	Activity is saved and can be viewed under 'My Activities'
10.2	User clicks on 'Join' button on the Activity UI (Prerequisite: User hasn't already joined the activity)	Join request is sent to the organiser and once organiser accepts the request, user is added to the activity	Join request is sent to the organiser and once organiser accepts the request, user is added to the activity
10.3	User clicks on the 'Pending' button on the page of activity which user has previously joined (Prerequisite: User has previously sent join request to organiser)	Join request is withdrawn	Join request is withdrawn
10.4	User clicks on the 'Leave' button on the activity page (Prerequisite: User joined activity and request got approved)	User is no longer participant of the activity	User is no longer participant of the activity

- j) Functionality: Approve/ Reject Join Request / Remove Participant & Cancel Activity (Organiser Side)

Test ID	Test Case	Expected Result	Actual Result
11.1	Organiser of an activity receives join request from User A. Organiser clicks on 'Approve' button	User A is now a participant of the activity	User A is now a participant of the activity
11.2	Organiser of an activity receives join request from User A. Organiser clicks on 'Reject' button	User A is is not a participant of the activity	User A is is not a participant of the activity
11.3	Organiser of an activity clicks on the 'Cancel' button on the activity page	Activity is cancelled	Activity is cancelled
11.4	Organiser of an activity clicks on '-' remove participant button	Participant is removed from activity	Participant is removed from activity

- k) Functionality: Search User

The user clicks on the search bar on the Home Page UI

- a. Generic cases:

Test ID	Test Case	Expected Result	Actual Result
7.1	User enters name of an existing user	User whose name matches searched username is displayed	User whose name matches searched username is displayed
7.2	User enters name that is invalid/nonexistent user	No user is displayed	No user is displayed

b. Specific cases:

Test ID	Test Case	Expected Result	Actual Result
7.1	Input: 1. Guo Yong 2. yong	User whose name matches searched username is displayed	User whose name matches searched username is displayed
7.2	Input: 1. Jacey	No user is displayed	No user is displayed

l) Functionality: Add Friend/ Remove Friend & Approve/ Reject Friend Request

Steps: User A goes to User B's profile

Test ID	Test Case	Expected Result	Actual Result
13.1	User A clicks on "Add friend" button	Status changes to requested Target User will see User's Name under Friend Requests	Status changes to requested Target User will see User's Name under Friend Requests
13.2	User B clicks on "Reject" button (Prerequisite: User A sends a friend request)	Selected User's name is removed from Friend Requests	Selected User's name is removed from Friend Requests
13.3	User B clicks on "Approve" button (Prerequisite: User A sends a friend request)	Selected User's name moves from Friend Requests to Current Friends	Selected User's name moves from Friend Requests to Current Friends
13.4	User A clicks on "Remove friend" button (Prerequisite: User A and User are friends from before)	Selected User's name is removed from Current Friends	Selected User's name is removed from Current Friends

m) Functionality: Edit Profile

Steps: User clicks on 'Edit Profile' under 'My Profile'

a. Generic cases:

Test ID	Test Case	Expected Result	Actual Result
13.1	User deletes required field but does not enter new value	The system prompts the user to enter missing field	The system prompts the user to enter missing field
13.2	User updates any/all activity field(s)	When user views 'My Profile', updated fields are displayed	When user views 'My Profile', updated fields are displayed

b. Specific cases:

Test ID	Test Case	Expected Result	Actual Result
13.1	Input: 1. Username: Julie123 → EMPTY	The system prompts the user to enter missing field	The system prompts the user to enter missing field
13.2	Input: 2. Username: Julie123 → Julie12345	When user views 'My Profile', updated fields are displayed	When user views 'My Profile', updated fields are displayed

n) Functionality: View User's Profile & Friend's Activity

Test ID	Test Case	Expected Result	Actual Result
14.1	User clicks on 'View Activity' on Friend's profile page (Prerequisite: User must be Friend of the other user)	System displays friend's upcoming participating and organising activities	System displays friend's upcoming participating and organising activities
14.2	User A clicks on User B's card through: 1. Participants list on Activity Page	System displays User B's Profile	System displays User B's Profile

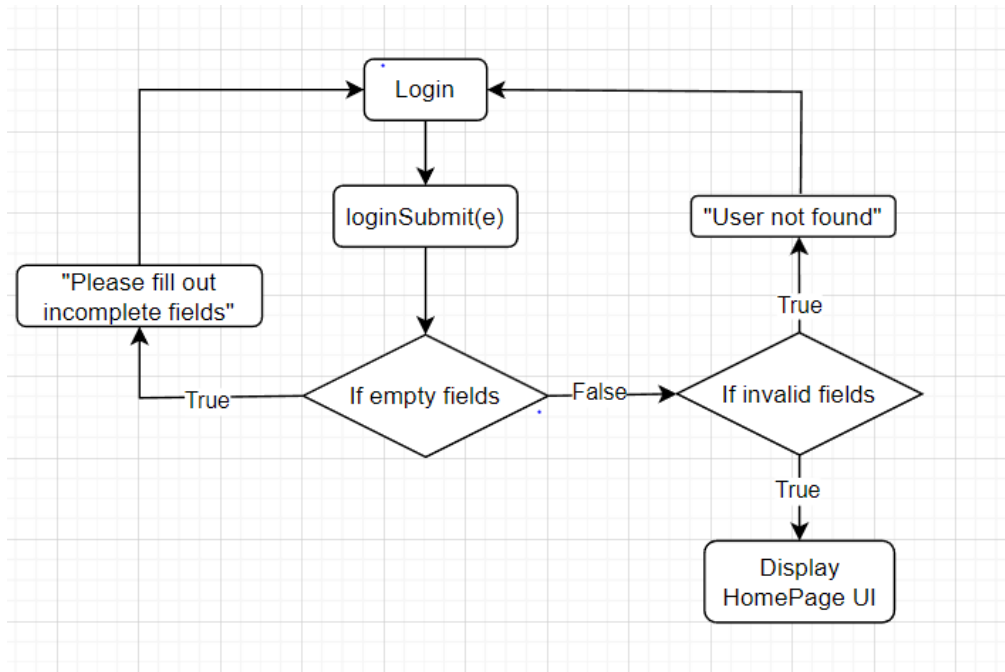
	2. Searching UI after searching for User B 3. User's Friends list UI		
--	---	--	--

o) Functionality: Chat

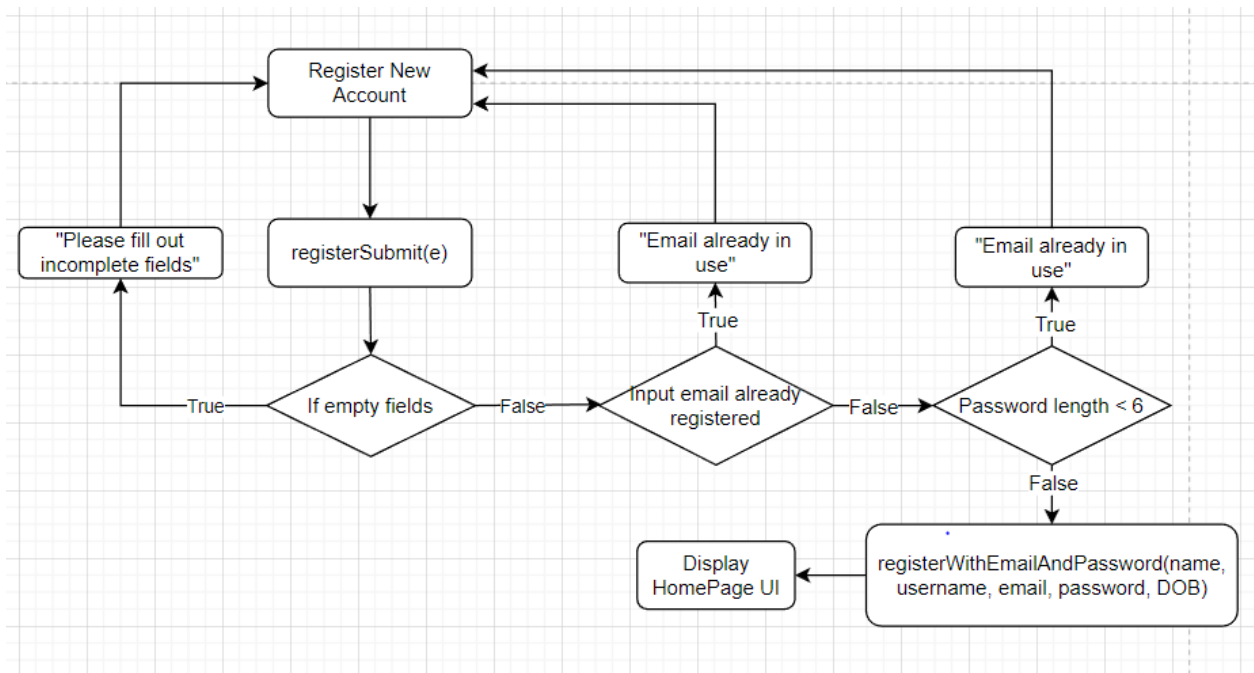
Test ID	Test Case	Expected Result	Actual Result
15.1	User starts chat with no previous Chat History	No messages will be displayed	No messages will be displayed in ChatUI
15.2	User opens chat with existing messages	Previously sent messages will be displayed	Previously sent messages will be displayed
15.3	User sends a message to Friend	Message is instantly displayed in Chat Window of User and Friend	Message is instantly displayed in Chat Window of User and Friend
15.4	User sends a message to non-Friend	Message is instantly displayed in Chat Window of User [Notification is sent to non-Friend]	Message is instantly displayed in Chat Window of User [Notification is sent to non-Friend]
15.5	Friend sends a message to User	Message is instantly displayed in Chat Window	Message is instantly displayed in Chat Window
15.6	User clicks on their message and clicks on 'delete'	Message is deleted	Message is deleted

4.2 White-Box Testing

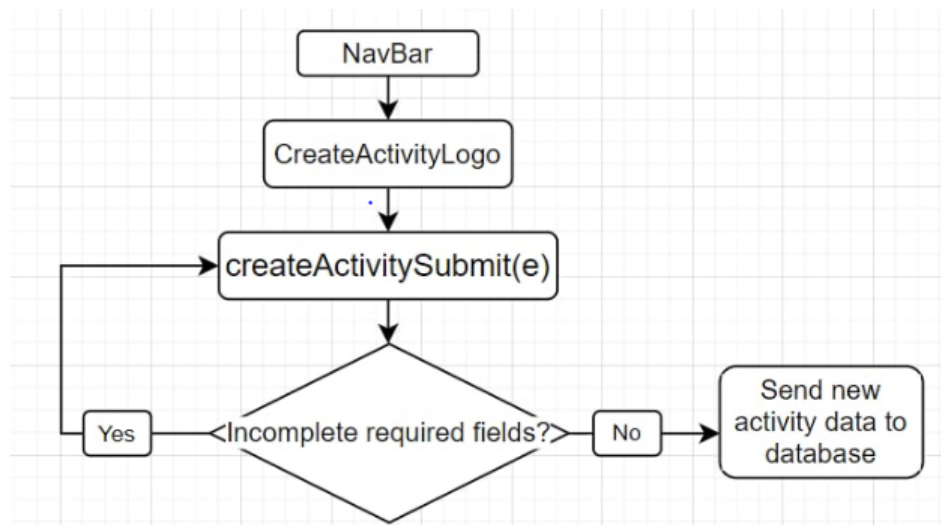
a) Functionality: Login Account



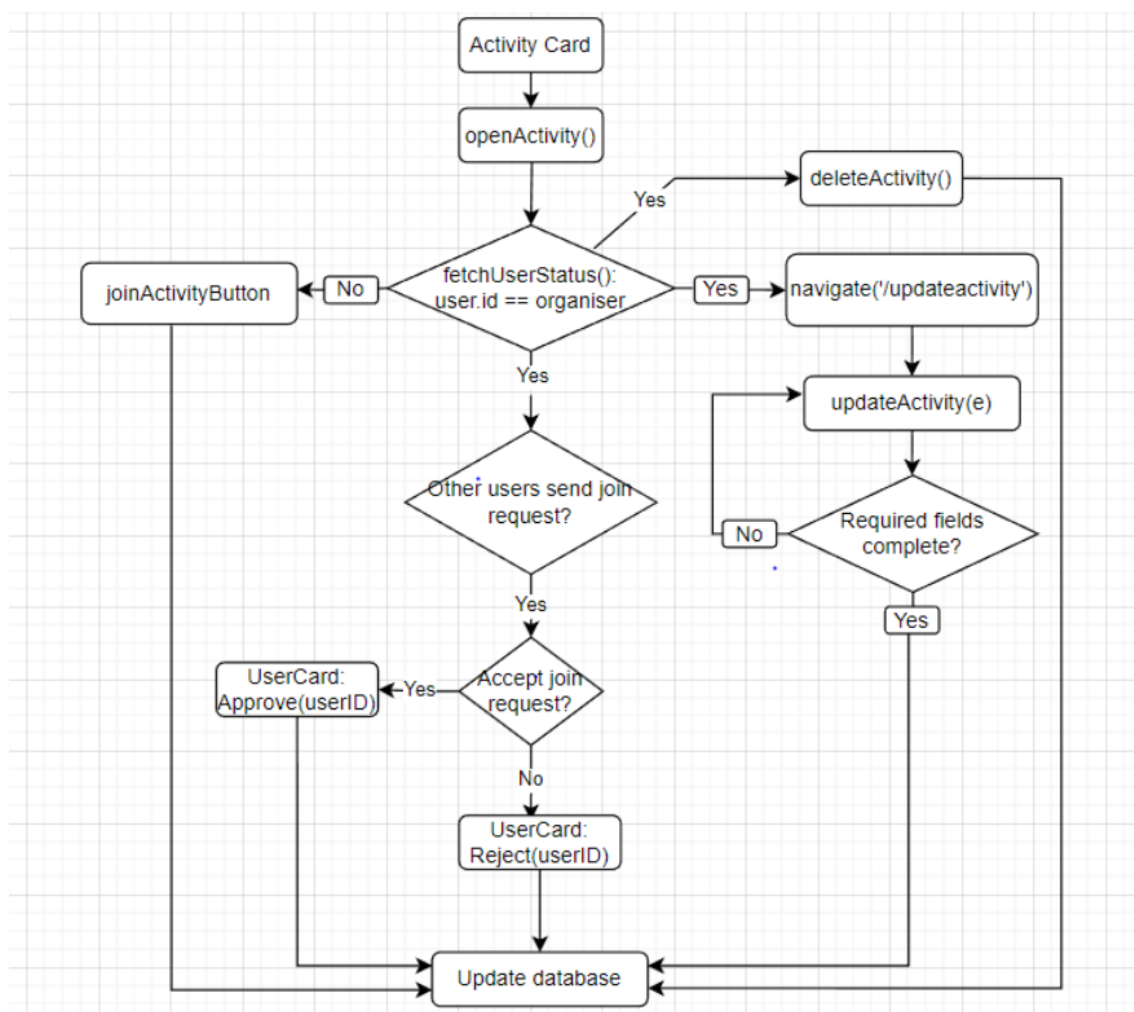
b) Functionality: Register Account



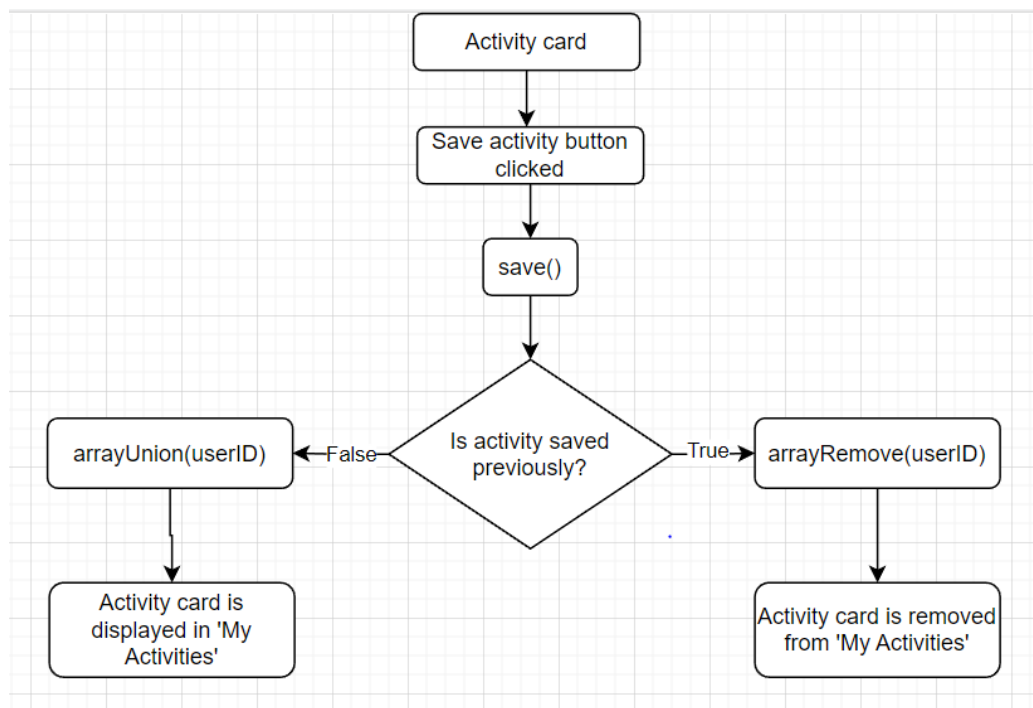
c) Functionality: Create Activity



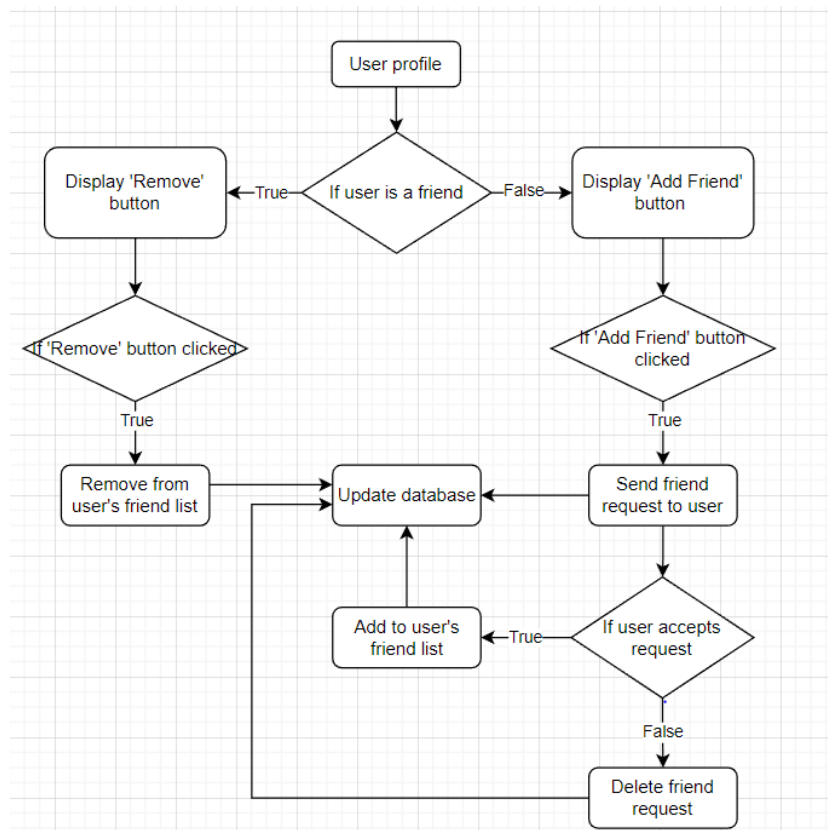
d) Functionality: Join/cancel/update activity and reject/approve join request



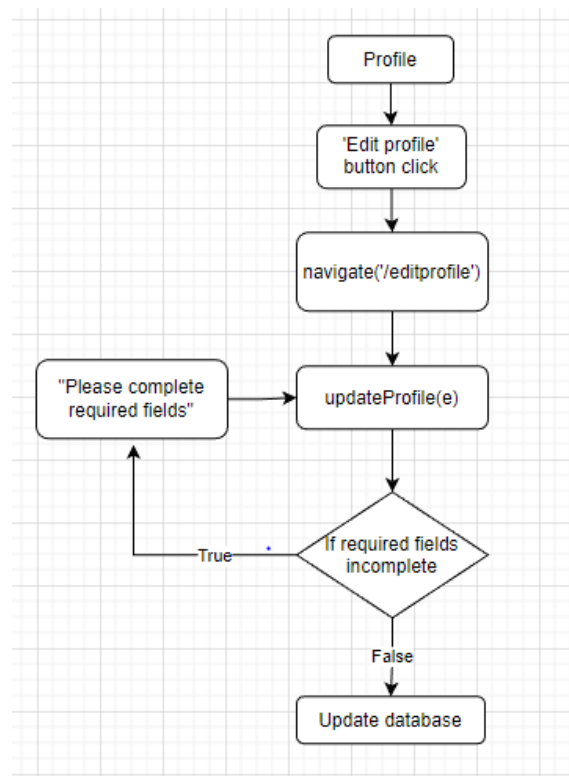
e) Functionality: Save/unsave Activity



f) Functionality: Add/remove friend or accept/reject friend request



g) Edit profile



5. References

Bruegge, & Dutoit, A. H. (2010). Object-oriented software engineering : using UML, patterns, and Java (3rd ed.). Prentice Hall.

caleb cheam. (2022, April 10). *CZ2006 DSAI group 1 Rainer's Rabbids*. [Video]. Youtube.
<https://www.youtube.com/watch?v=rEJBVSnsShE>

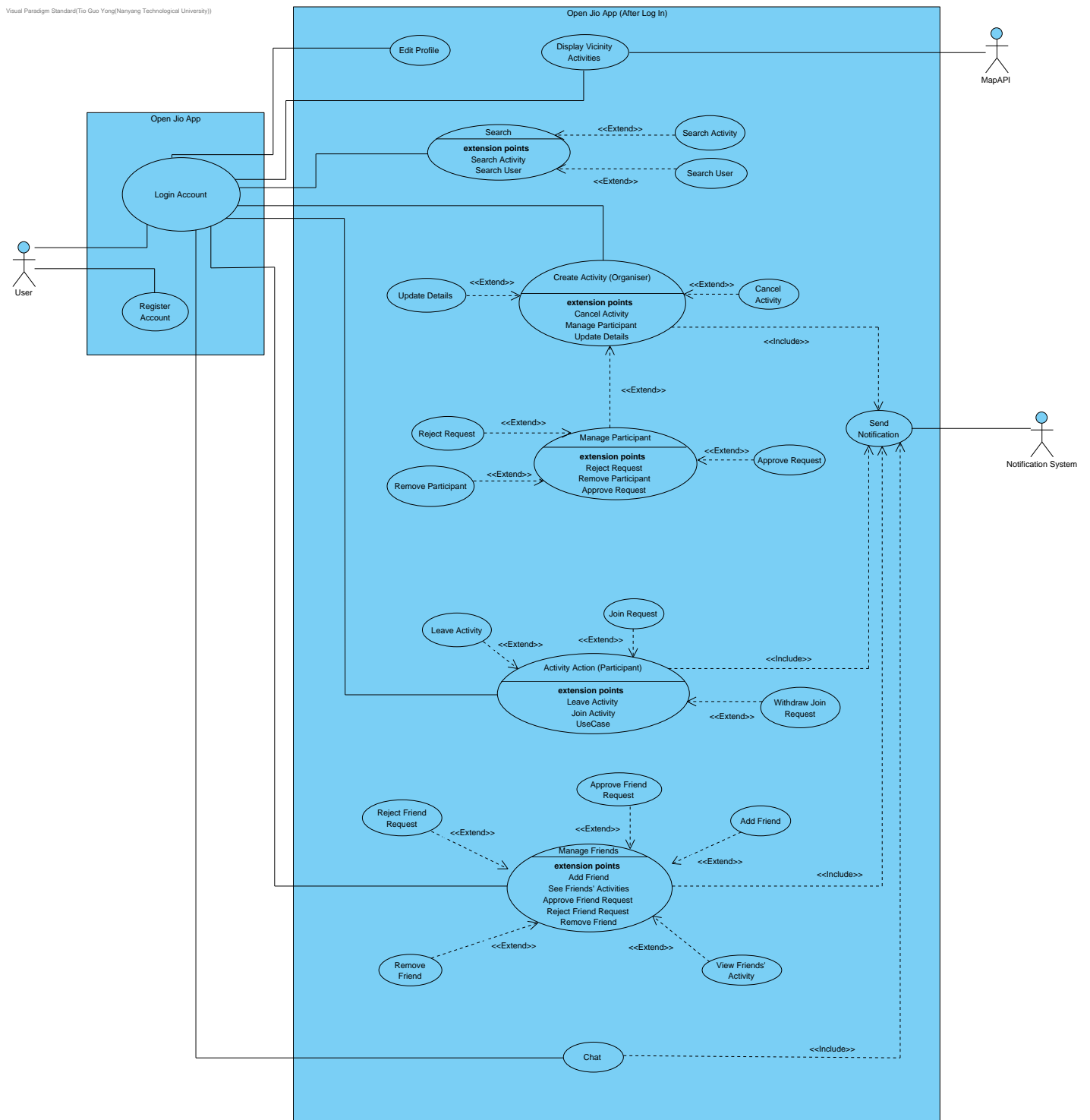
Htet Naing. (2019). CZ2006_Software_Engineering. GitHub.
https://github.com/Javelin1991/CZ2006_Software_Engineering

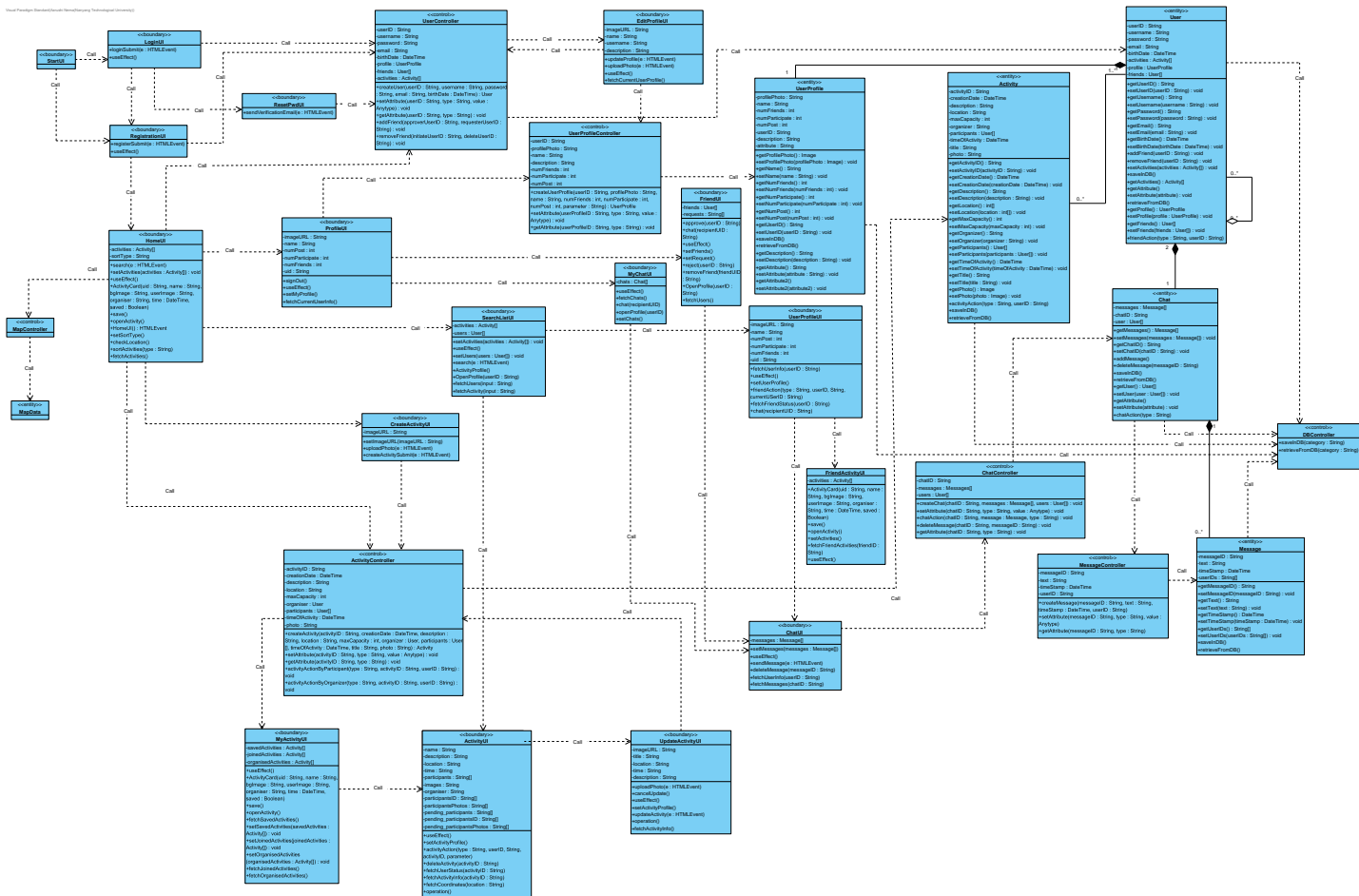
Sommerville, I. (2016). Software Engineering. 10th Edition, Pearson Education Limited, Boston.

6. Appendix

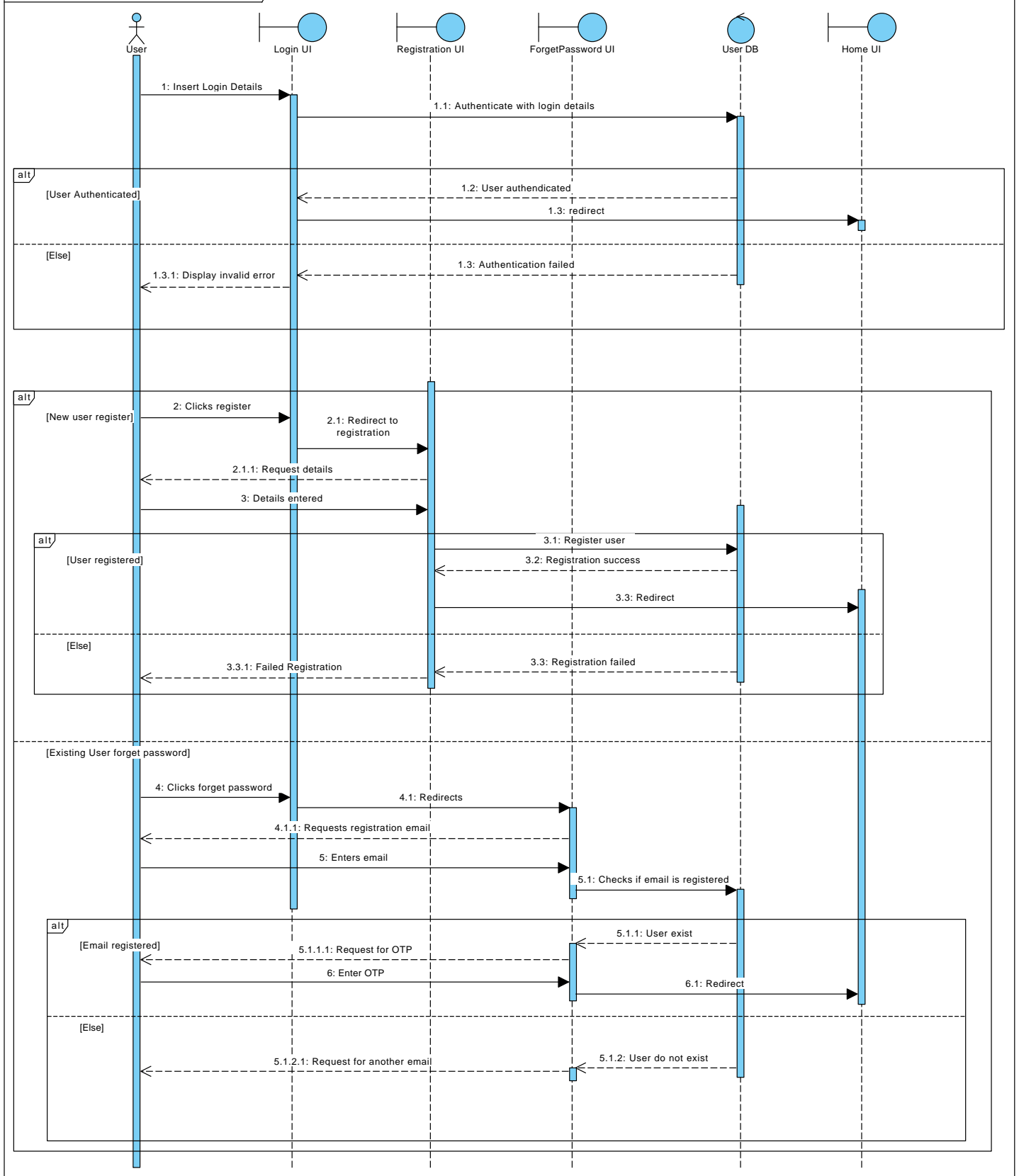
This section Includes all UML diagrams, links to mockups, source code and other deliverables.

1. [Use Case Diagram](#)
2. Class Diagrams
 - a) [Conceptual](#)
 - b) [Dynamic](#)
3. Sequence Diagrams
 - a) [Login, Registration & Reset Password](#)
 - b) [Edit Profile](#)
 - c) [Organise Activity](#)
 - d) [Join & Leave Activity](#)
 - e) [View & Add Friends](#)
 - f) [Chat](#)
4. [Dialog Map](#)
5. [System Architecture Diagram](#)
6. [Initial UI Mockups](#)
7. [Application Skeleton](#)
8. [Source Code](#)
9. [Demo Video](#)
10. [GitHub Repository](#)

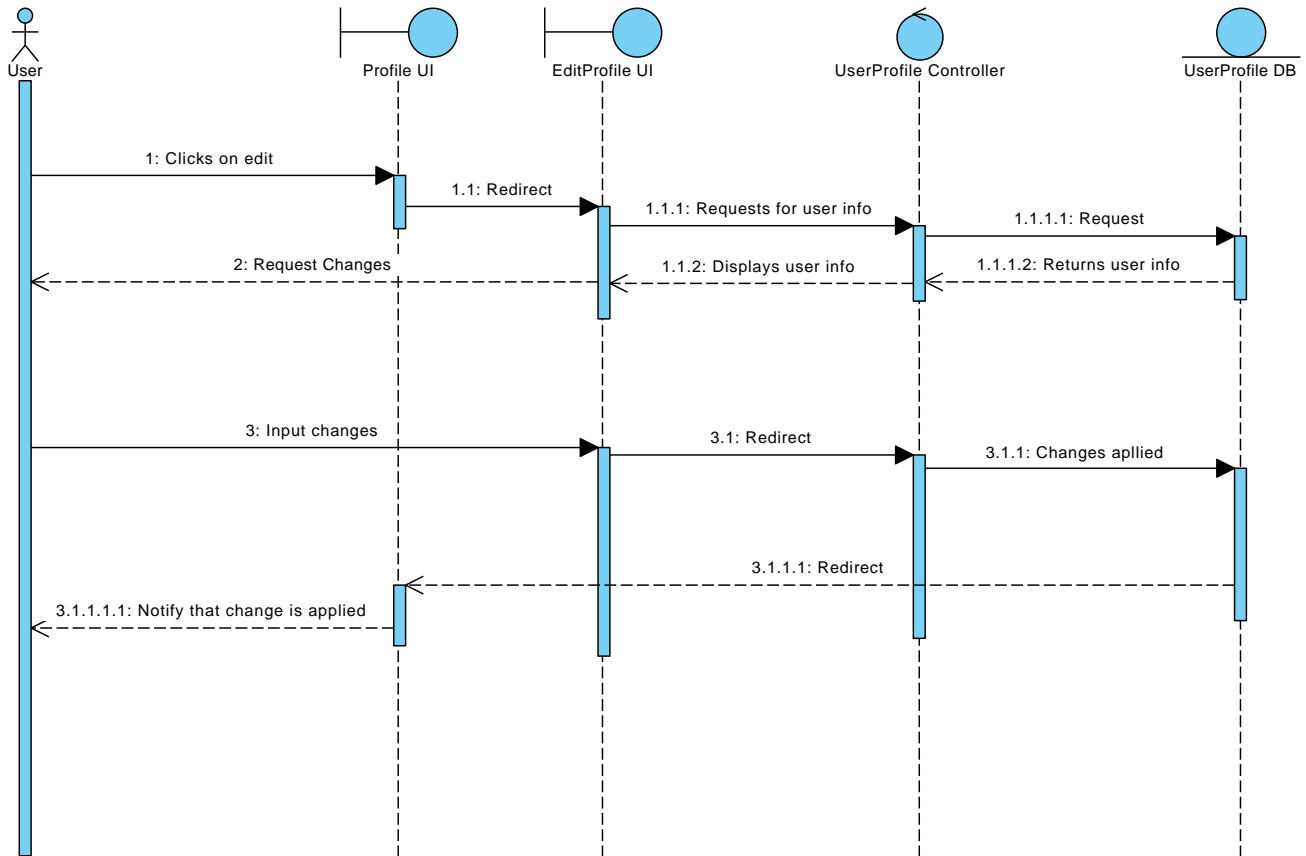


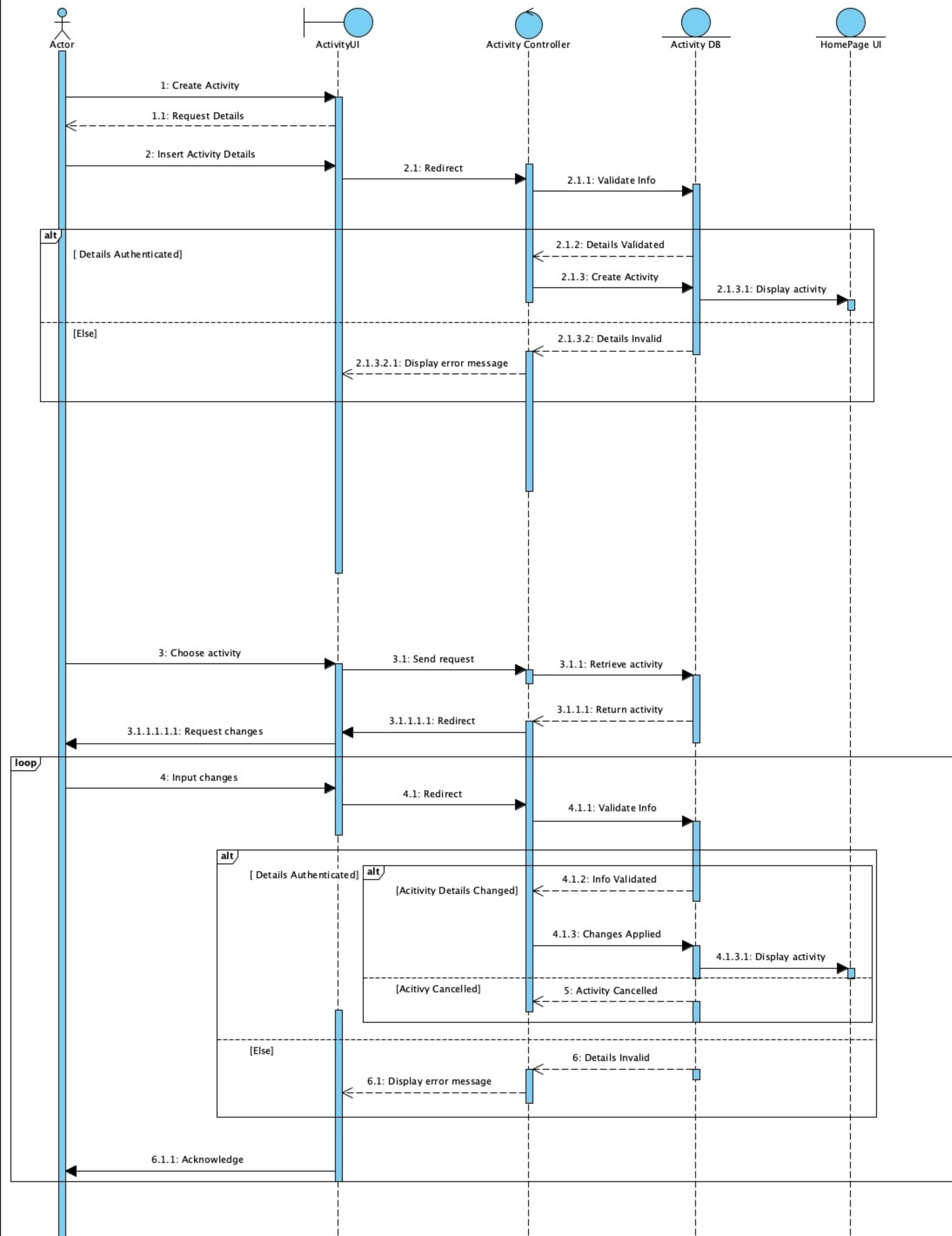


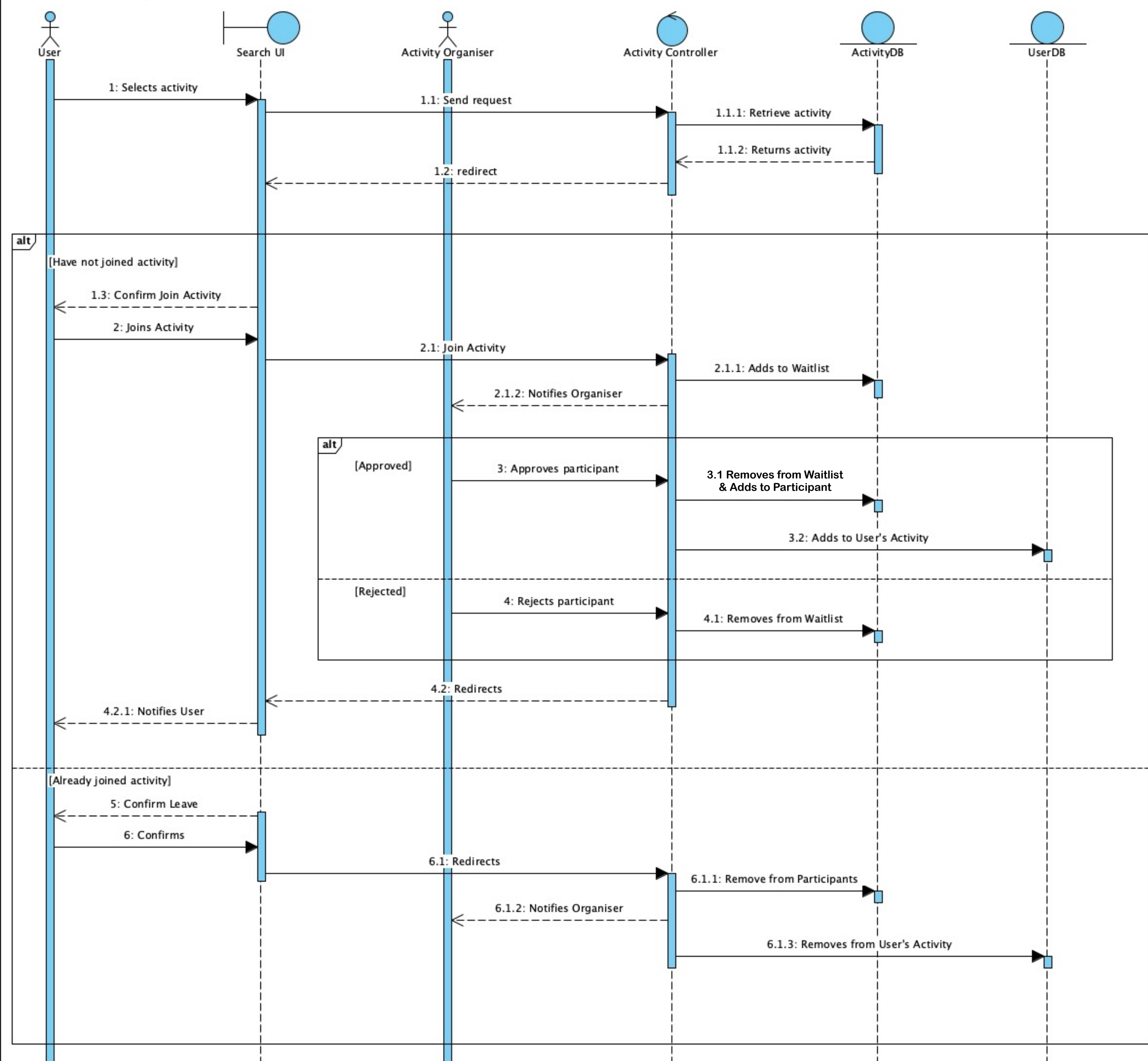
sd [Login, Registration & Reset/Forget Password]

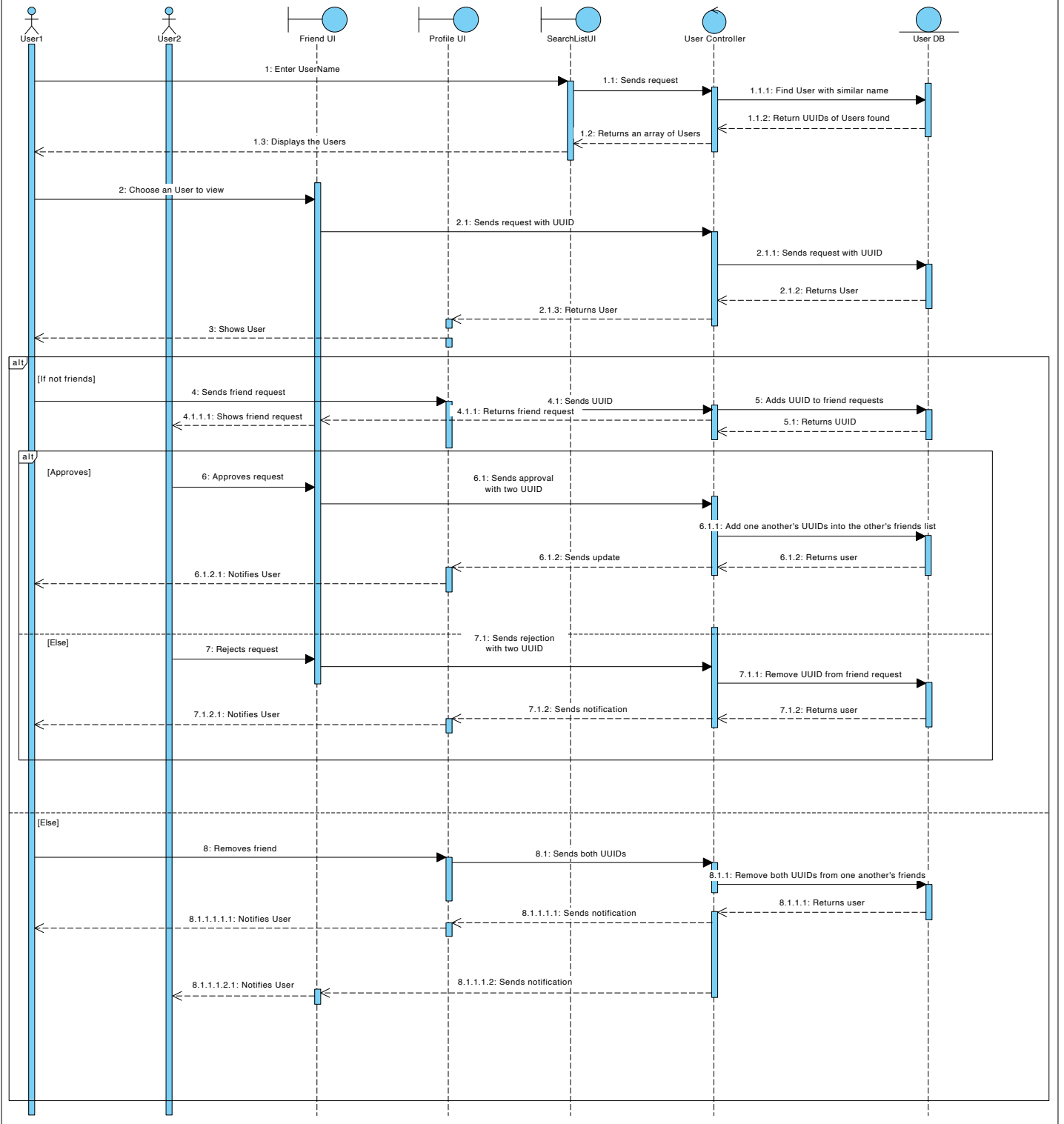


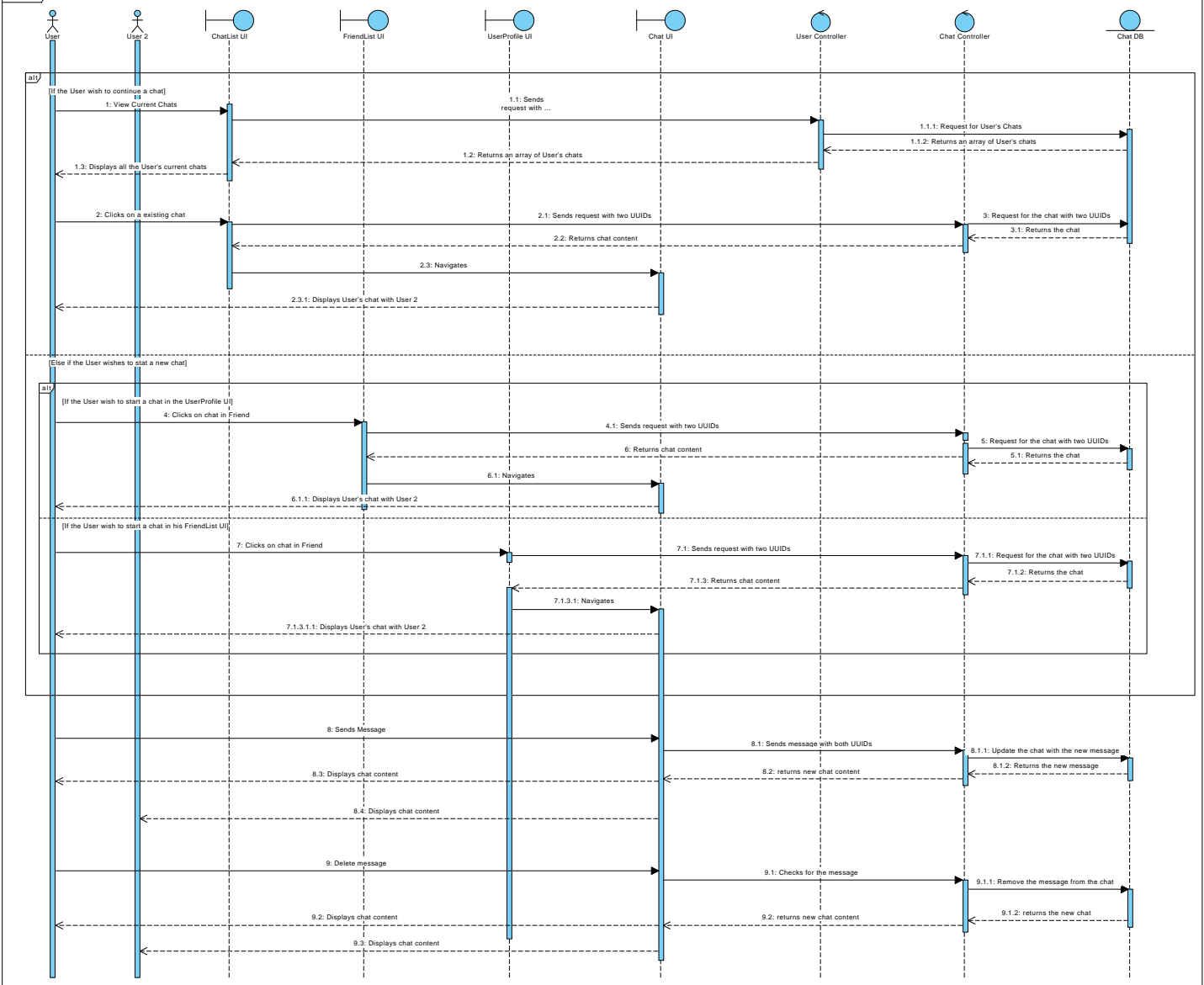
sd [Edit Profile]

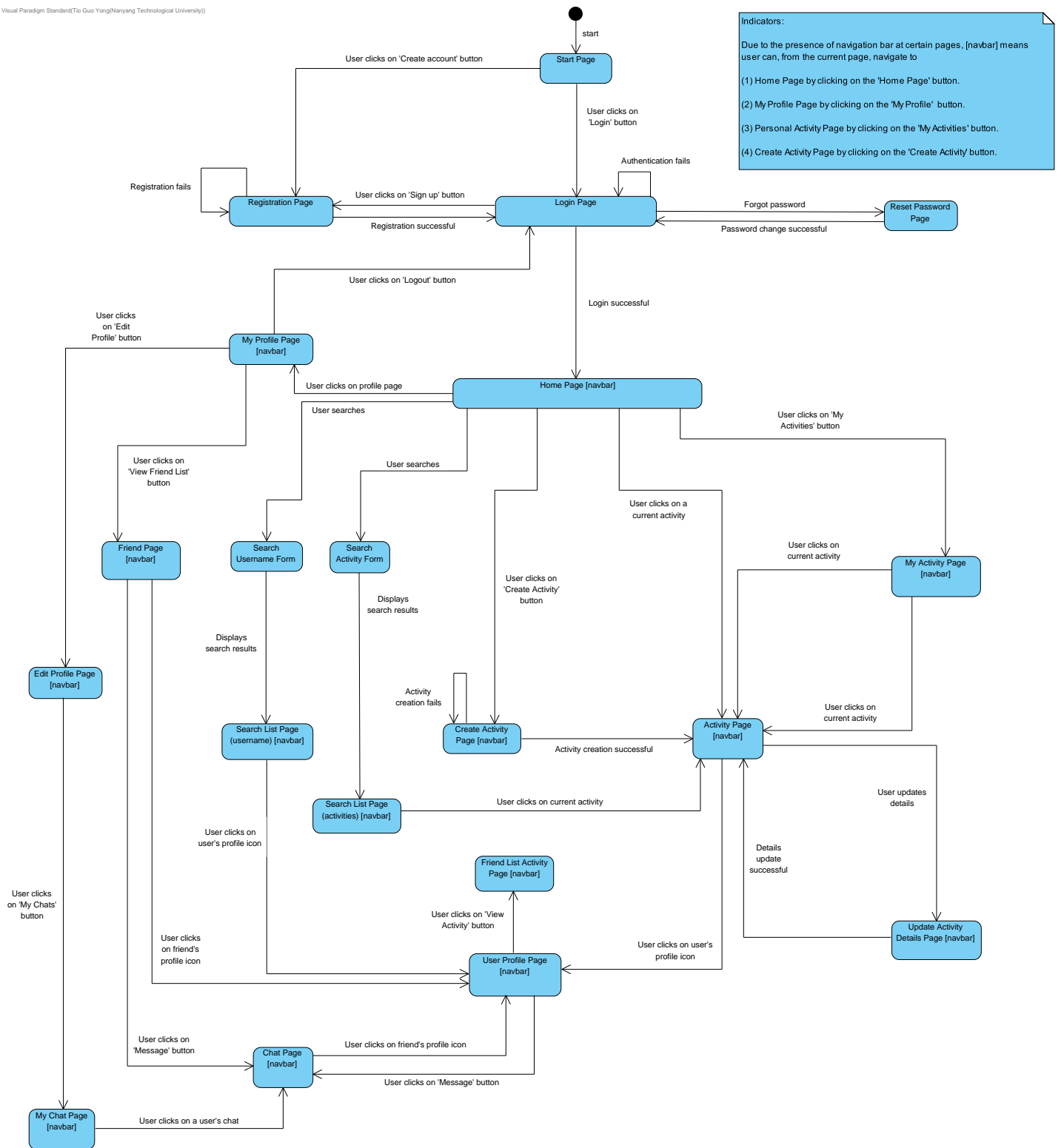












System Architecture

View

Controller

Model

