.

System Design Document

Group 7:

Andrew Dovale-Puig
Eli James
Jack Fejer
Lucy Childerhose
Lucas Komljenovic
Shyan Vilvarajah
Xinwei Lyu

TA: Elyas Rashno

November 7, 2024

Table of Contents

Table of Contents 2
1. Introduction **3**
Purpose of the System 3
2. Design Goals 4
Performance 4
Dependability 4
Availability 4
Security 4
Resilience 4
Safety 4
Cost 5
End User Criteria 5
Definitions, Acronyms and Abbreviations : 5
3. Use Cases **5**
Use Case 1: CreatePost 5
Participating Actors : 5
Flow of Events: 5
Entry Condition: 6
Quality Requirements: 6
Use Case 2: FriendPosts 6
Participating Actors : 6
Flow of Events: 6
Entry Condition: 7
Exit Condition: 7
Quality Requirements: 7
4. Object Model **7**
5. Sequence Diagrams **8**
5. Class Diagram **8**
6. State Chart Diagram **10**
7. Use Case Diagrams **12**
8. System Architecture **13**
Subsystem Decomposition and Services 13
GUI 13
Model 14
Controller 14
Database 14
Software/hardware Mapping 14
Persistent Data Management 14
Access Control and Security 14
Global Software Control 15
Boundary Conditions 15
9. GUI Interfaces **15**
The User Profile Page 15
The Home Page 16
10. Coding Assignments **17**
11. Timeline **18**

1. Introduction

Purpose of the System

In today's interconnected society, there exist many apps built with the intention of bringing people together through sharing experiences, although few are focused on the joy of sharing food and cruise recommendations with friends. Many current platforms focus on general food reviews, left by any customer. Here, there exists a gap for a personalized, or community-driven food review-sharing site either for restaurants or at-home recipes. The purpose of Snackmap is to provide an easy platform for food lovers to share and review the preparation process and experience of restaurant dishes or home-cooked dishes. Through the platform, users can explore new restaurants, share recipes, and interact with others who love food to expand their food experience. Have you ever received a restaurant recommendation, arrived at the restaurant and not known what to order? Snackmap will allow for reviews to be left on individual dishes or popular favourites. This project intends to create a dedicated platform where friends can easily exchange food reviews and uncover new dining experiences for users who are food enthusiasts or someone simply wanting to share a great meal recommendation with a friend. Snackmap seeks to build a personal network of users who can explore and share food together.

2. Design Goals

Performance

When interacting with the system, it should load all posts and reviews from the users' “friends” on the homepage within 3 seconds of logging in, ensuring that the platform remains responsive and efficient. Additionally, immediately after users publish a post, it should be seen with no more than a 3-second delay, ensuring all actions on the platform are reflected in real time. Overall, all actions from a user (like pressing a button, changing pages, etc.) should take no longer than 3 seconds. In terms of security, the system should automatically sign out of user accounts after 15 minutes of inactivity to ensure the account is secure if users forget to log out.

Dependability

Availability

The system will always be available to perform the basic functions of its design.

Security

The information of a user will be protected by their username and password login. The system will flag threatening content.

Resilience

The system will not be designed to run under erroneous conditions. This can include poor wifi, poor memory, etc.

Safety

The system’s intended design does not contain the possibility of human injury. Invalid or sensitive information will be filtered off the site.

Reliability
The system will not have any bugs hindering the app's performance. Edge cases will be tested to ensure the proper behaviour of the system

Cost

The cost of this system will be quite low or obsolete. There is no cost associated with development. The system does not require funds to deploy. Maintaining and upgrading the system will have no costs. Administering the system will not cost anything unless the data stored in the database becomes quite large. If so, a paid service may be needed to host this information.

End User Criteria

The GUI has been designed to be simple for users to interact with, improving their overall experience. Every page will have a navigation bar at the bottom of the screen allowing for

quick, intuitive navigation between pages. The user will also have a feed specifically dedicated to only show friends' posts, allowing them to filter down posts. Users can connect with friends and other community members by liking, sharing, and commenting on posts, creating an interactive and collaborative experience for exploring food options in the user's local area.

Definitions, Acronyms and Abbreviations:

- Graphical User Interface (GUI): Graphical user interface for visual presentation of interaction with users.
- Minimum Viable Product (MVP): A minimum viable product is a basic version of software that contains the necessary features.

3. Use Cases

Use Case 1: CreatePost

Participating Actors:

- User
- PostManagement
- Database

Flow of Events:

1. User initiates post creation: The user selects the "Posts" option on their Home page.
2. User add post: The user clicks the Add Post button
3. User fills in post details: The app prompts the user to:
    a. file the title
    b. Upload an image (optional).
    c. Enter a description of the dish or restaurant.
    d. Select a location (restaurant or homemade).
    e. Choose a review type (e.g., restaurant, homemade).
    f. Assign an overall rating (stars out of 5).
4. User submits the review: The user clicks "Post" to submit their review.
5. PostManagement processes the review: The PostManagement component validates the review data (ensuring no empty required fields) and prepares it for storage.

6. Database stores the review: The review data is stored in the Database, making it accessible for display on the user's and friends' feeds.
7. Review confirmation: PostManagement confirms the review's successful creation, displaying it on the user's homepage

Entry Condition:

The user has logged in and is on their Personal Homepage, ready to create a review. Exit Condition.

The user's review is successfully created and visible on their homepage and available for their friends to see, like, and comment on, depending on privacy settings.

Quality Requirements:

Response Time: The system must save and display the user's review within 3 seconds after submission.

User Experience: There is a clear prompt for each input. If the required fields (such as description, and rating) are empty, the interface should block the submission.

Use Case 2: FriendPosts

Participating Actors:

- User
- FriendsPage
- Database

Flow of Events:

1. User navigates to Friends Homepage: The user opens the Snackmap app and selects the "Friends" page from the navigation bar.
2. FriendsPage requests friend posts: FriendsPage retrieves posts made by the user's friends from the Database.
3. Database sends friend posts: The Database returns posts from friends, sorted by date or relevance, depending on system settings.

4. FriendsPage displays posts: Posts are shown on the user's screen, including images, descriptions, locations, and ratings. Each post includes options for interaction (like and comment).
5. User interacts with posts (optional): The user can like or comment on a friend's post.
    a. If the user likes a post, FriendsPage records the like and updates the post's like count.
    b. If the user comments, FriendsPage provides a text input, saves the comment, and displays it with the post.
6. Database updates interactions: Any likes or comments are stored in the Database, which associates them with the post for future reference.

Entry Condition:

The user is logged in and on the Friends Homepage, ready to view and interact with friends' posts.

Exit Condition:

The user has viewed and optionally interacted with friends' posts (liked or commented), and the interactions are saved.

Quality Requirements:

Response Time: Posts should load within 3 seconds of accessing the Friends Homepage.

User Experience: The interface should be simple and visually organized, making it easy for the user to view posts and interact with them.

Data Integrity: Any likes or comments must be accurately recorded in the Database, and interactions should update immediately to reflect real-time engagement.

4. Object Model

1. **Entities**: User, Post
2. **Controls:** RegistrationControl, LoginControl, PostControl, UserControl, ProfileControl, DatabaseControl
3. **Boundaries:**
    a. ProfileButton
        i. ViewPostsButton
            1. ViewPostsPopUpButton
                a. ViewPostsPopUpWindow

i. PopUpWindowCloseButton

ii. LanguageButton
    1. LanguagePopUpButton
        a. LanguageWindow
            i. PopUpWindowCloseButton

iii. PrivacySettingsButton
    1. PrivacySettingsPopUpButton
        a. PrivacySettingsP{opUpWindow
            i. PopUpWindowCloseButton

iv. EditProfileButton
    1. EditProfilePopUpWindow
        a. ChangeUsernameButton
            i. ChangeUsernamePopUpButton
                1. ChangeUsernamePopUpWindow
                    a. PopUpWindowCloseButton
        b. ChangeEmailIDButton
            i. ChangeEmailIDPopUpButton
                1. ChangeEmailIDPopUpWindow
                    a. PopUpWindowCloseButton
        c. ChangePhoneNumberButton
            i. ChangePhoneNumberPopUpButton
                1. ChangePhoneNumberPopUpWindow
                    a. PopUpWindowCloseButton
        d. ChangePasswordButton
            i. ChangePasswordPopUpButton
                1. ChangePasswordPopUpWindow
                    a. PopUpWindowCloseButton
        e. ChangeProfilePictureButton
            i. ChangeProfilePicturePopUpButton
                1. ChangeProfilePicturePopUpWindow
                    a. PopUpWindowCloseButton
        f. UpdateButton
            i. UpdatePopupWindow
                1. PopUpWindowCloseButton

b. PostsButton
    i. AddPostsButton
        1. AddPhotosButton
            a. AddPhotosPopUpButton
                i. AddPhotosPopUpWindow
                    1. PopUpWindowCloseButton
        2. TypeReviewButton
            a. TypeReviewPopUpButton
                i. TypeReviewPopUpWindow
                    1. PopUpWindowCloseButton
    ii. EditProfileButton
        1. EditPostButton
            a. EditPhotosButton
                i. EditPhotosPopUpButton

1. EditPhotosPopUpWindow
    a. PopUpWindowCloseButton

b. EditReviewButton
    i. EditReviewPopUpButton
        1. EditReviewPopUpWindow
            a. PopUpWindowCloseButton

c. DeleteReviewButton
    i. DeleteReviewPopUpButton
        1. DeleteReviewPopUpWindow
            a. PopUpWindowCloseButton

5. Sequence Diagrams

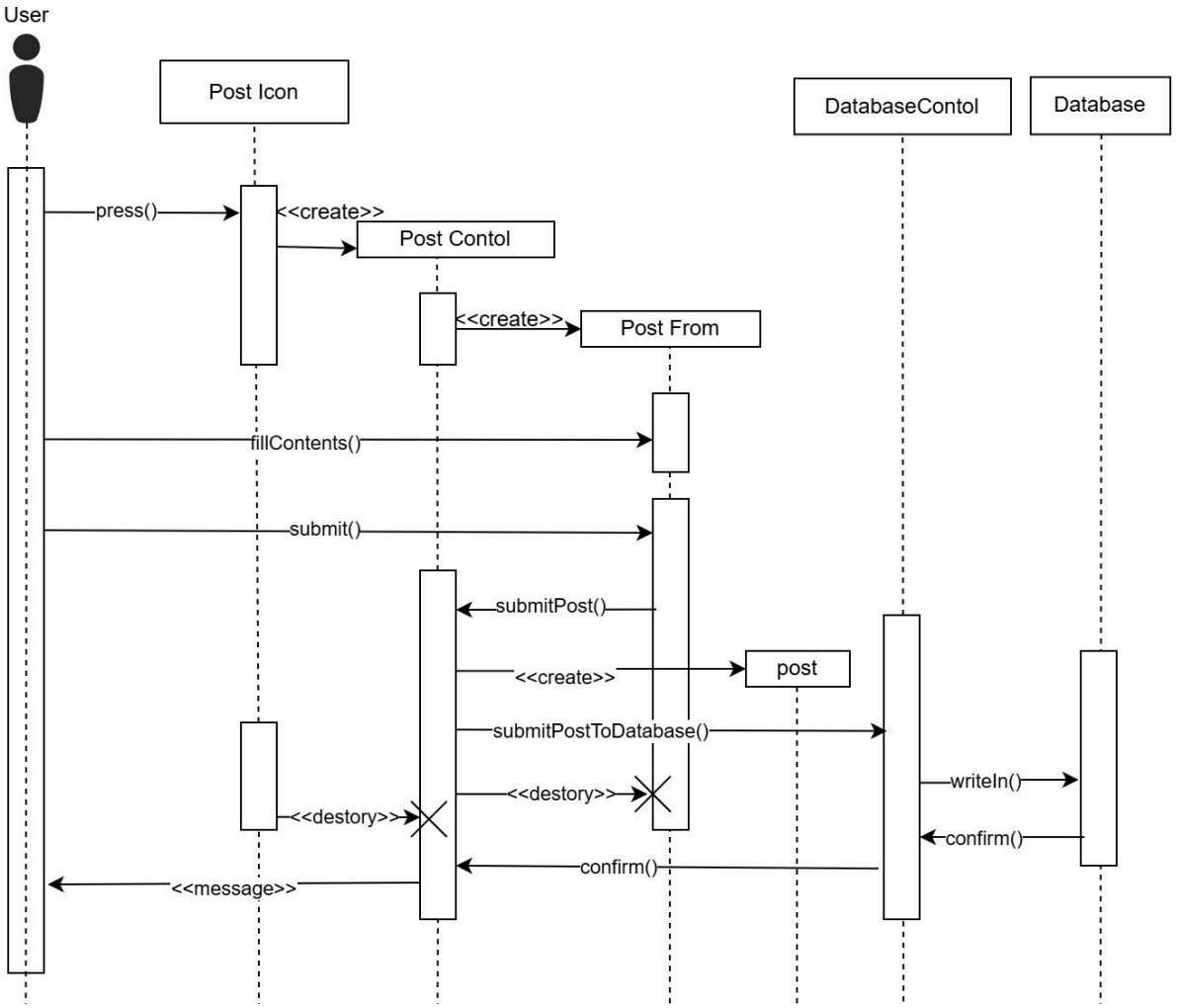

Figure 1: Sequence diagram for the first use case.

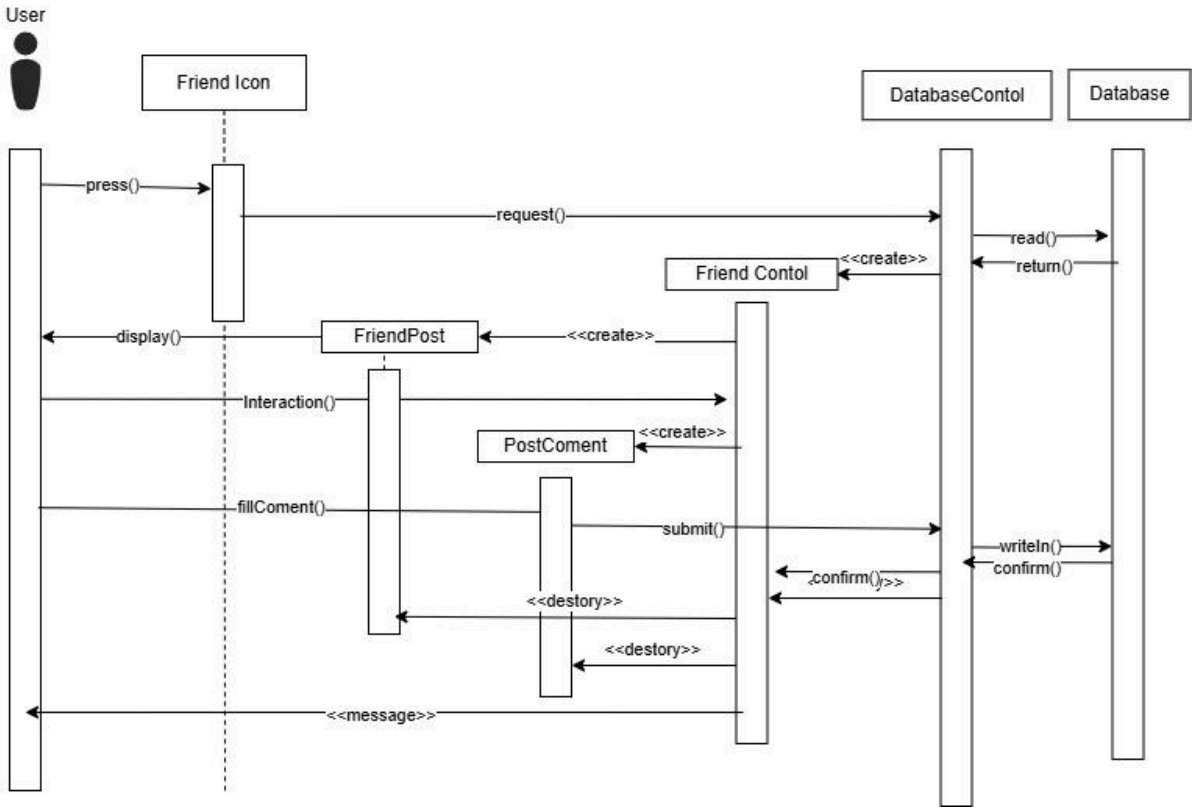

Figure 2: Sequence diagram for the second use case.

5. Class Diagram

The class diagram below shows the overall flow and interactions between objects. Each user will have an array of user IDs that represent their friends list and an array of Post objects representing their posts. Each post has three classes tied to it, respectively storing information regarding other users who like, comment and share a given user's post. The feed class will be used to generate the posts shown on the user’s main page and will pull all posts from the database filtering down the results based on the user’s location. This class will also be used to generate the posts shown on the user's friend page and will pull posts from the database if the post was made by other users on their friend list.

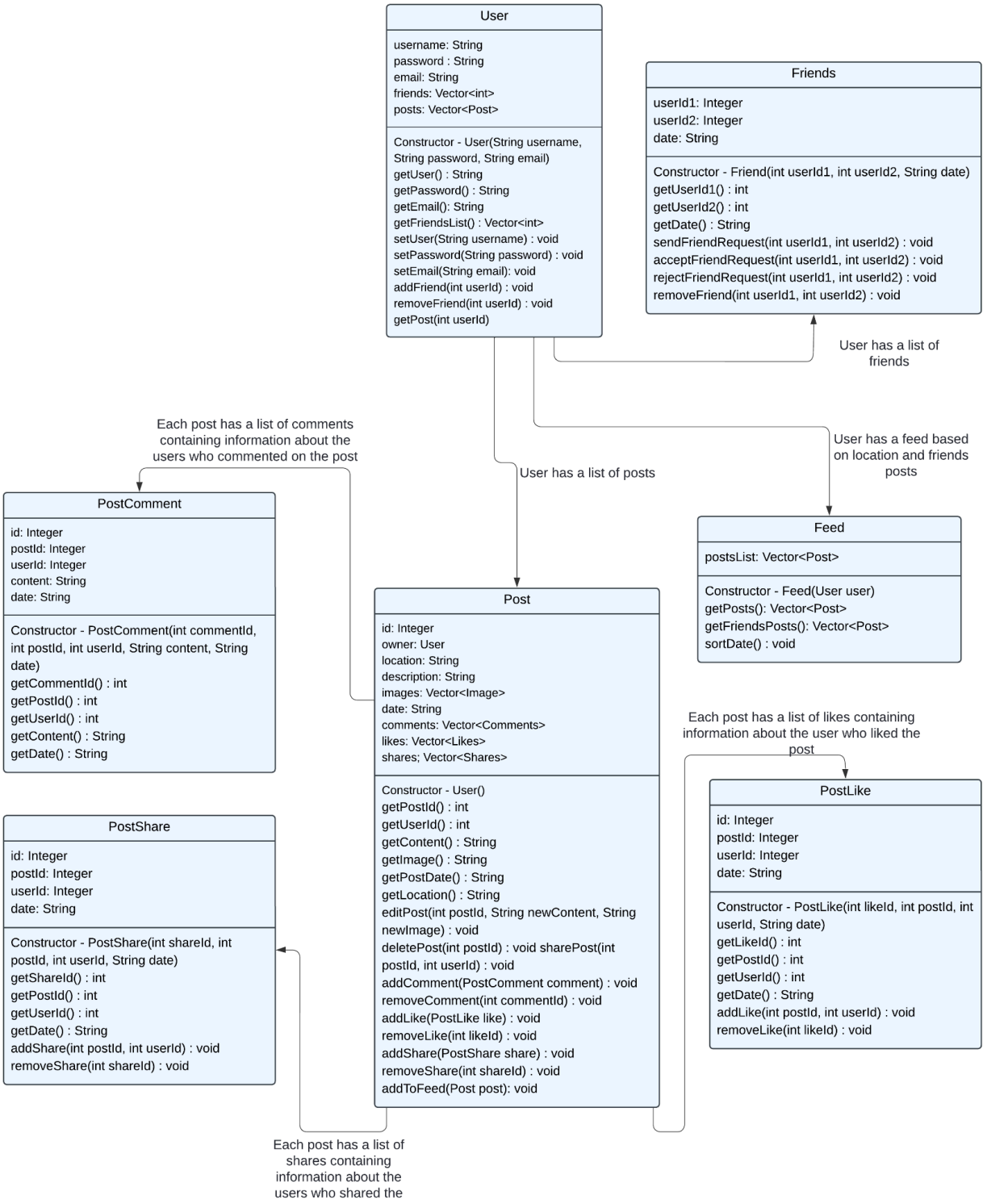

Figure 3: SnackMap Class diagram.

6. State Chart Diagram

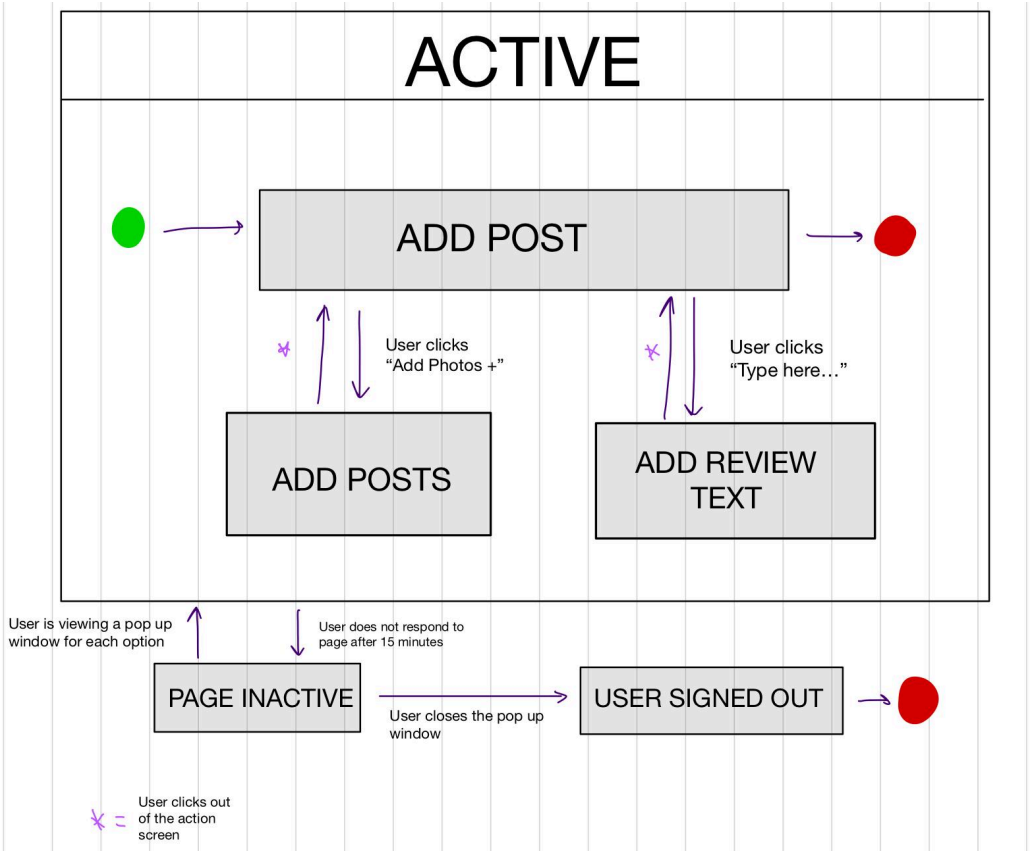

Figure 4: SnackMap “Add Post” State Chart Diagram.

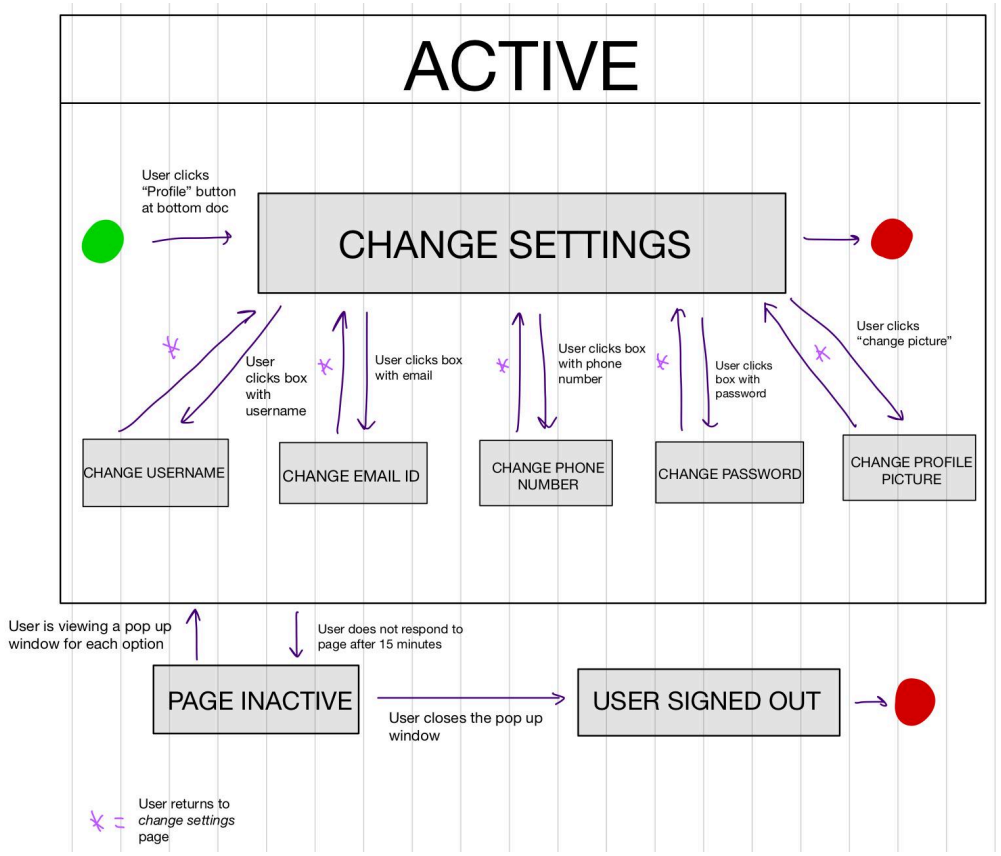

Figure 5: SnackMap "Change Settings" State Chart Diagram.

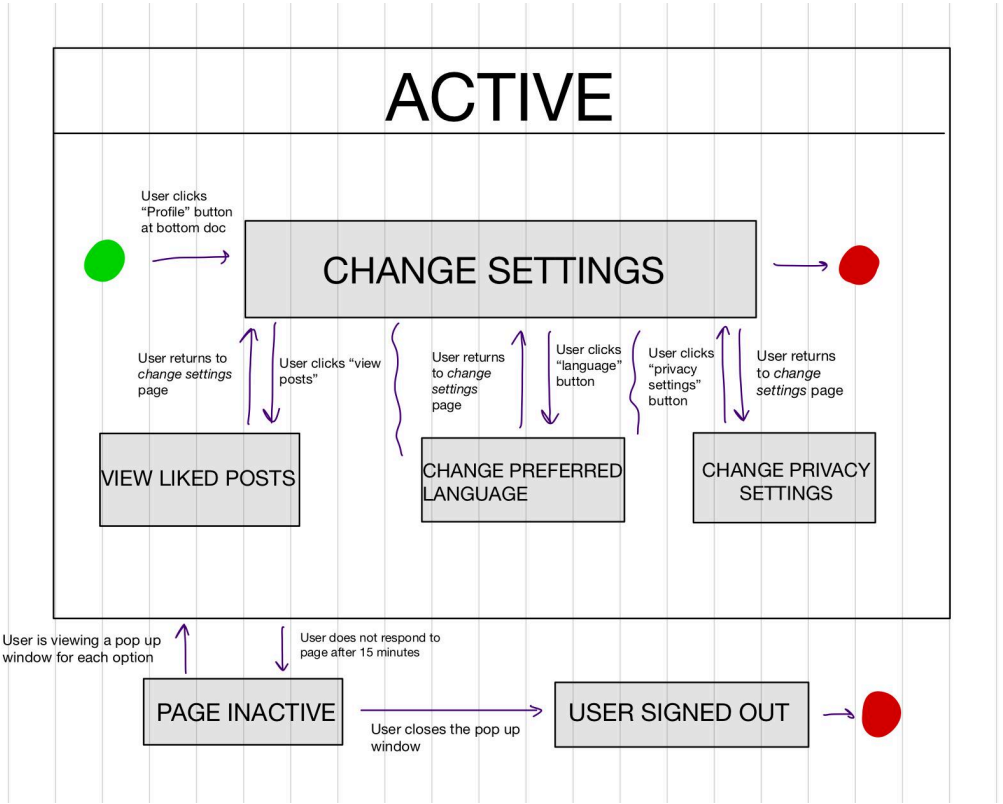

Figure 6: SnackMap "Change Settings" State Chart Diagram, continued.

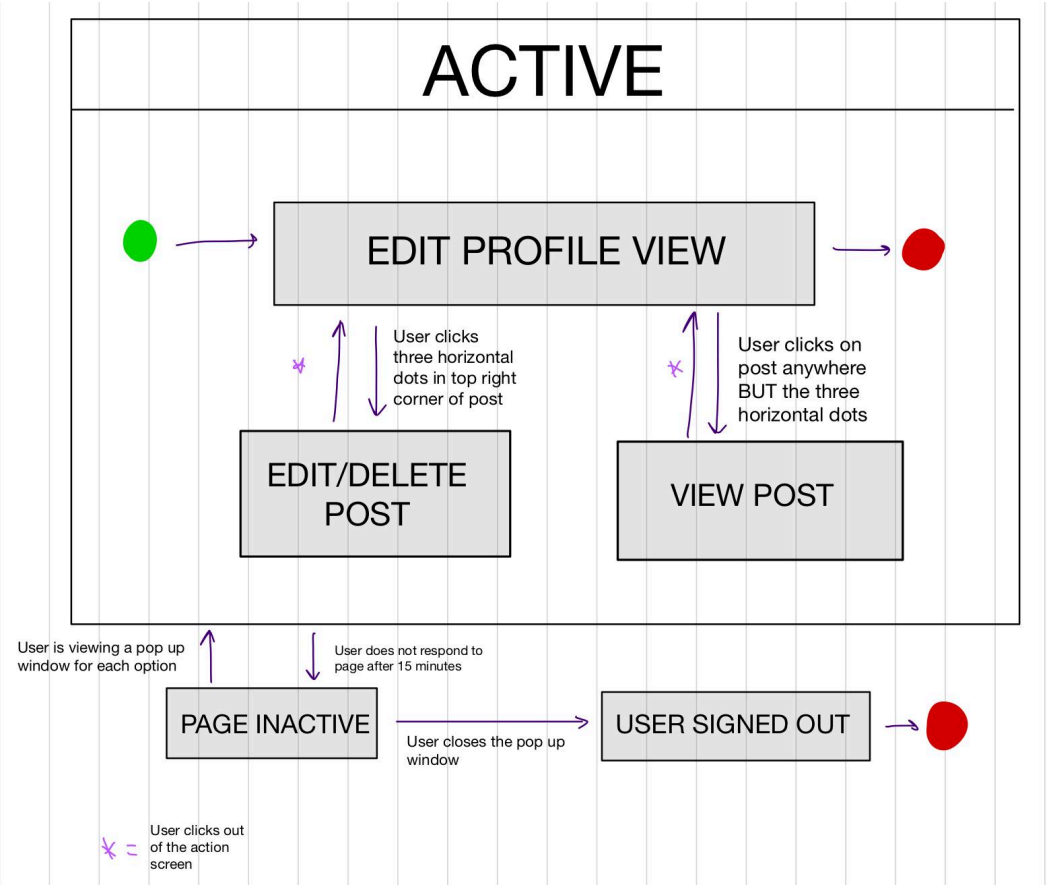

Figure 7: SnackMap "Edit Profile" State Chart Diagram.

7. Use Case Diagrams

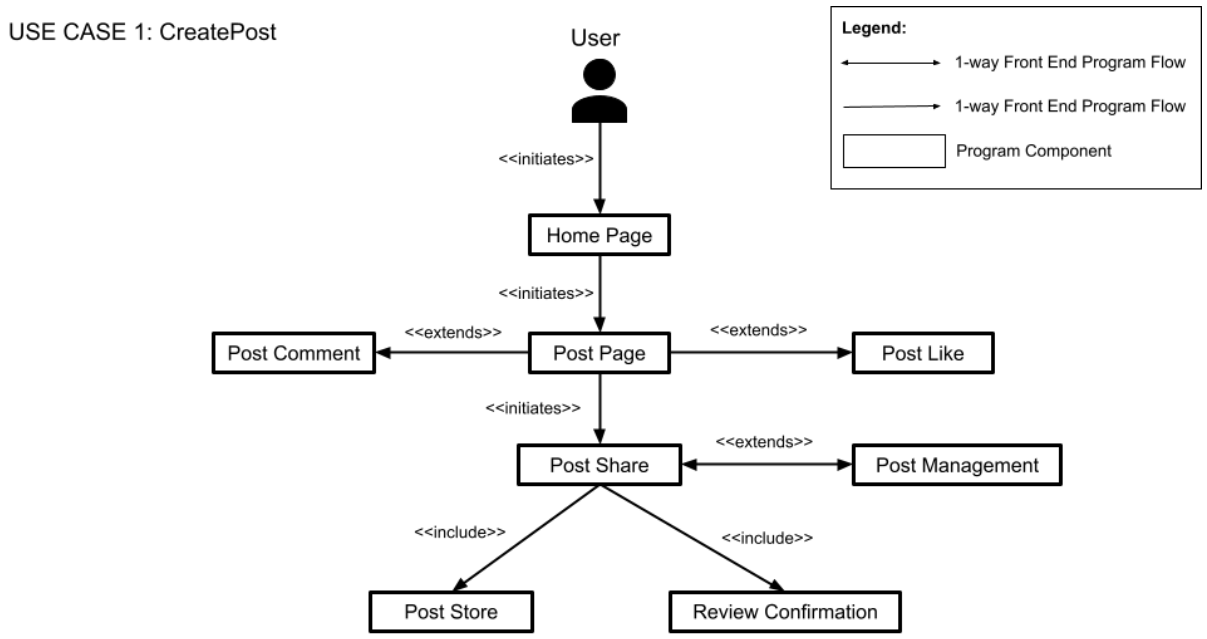

Figure 8: SnackMap CreatePost Use Case Diagram.

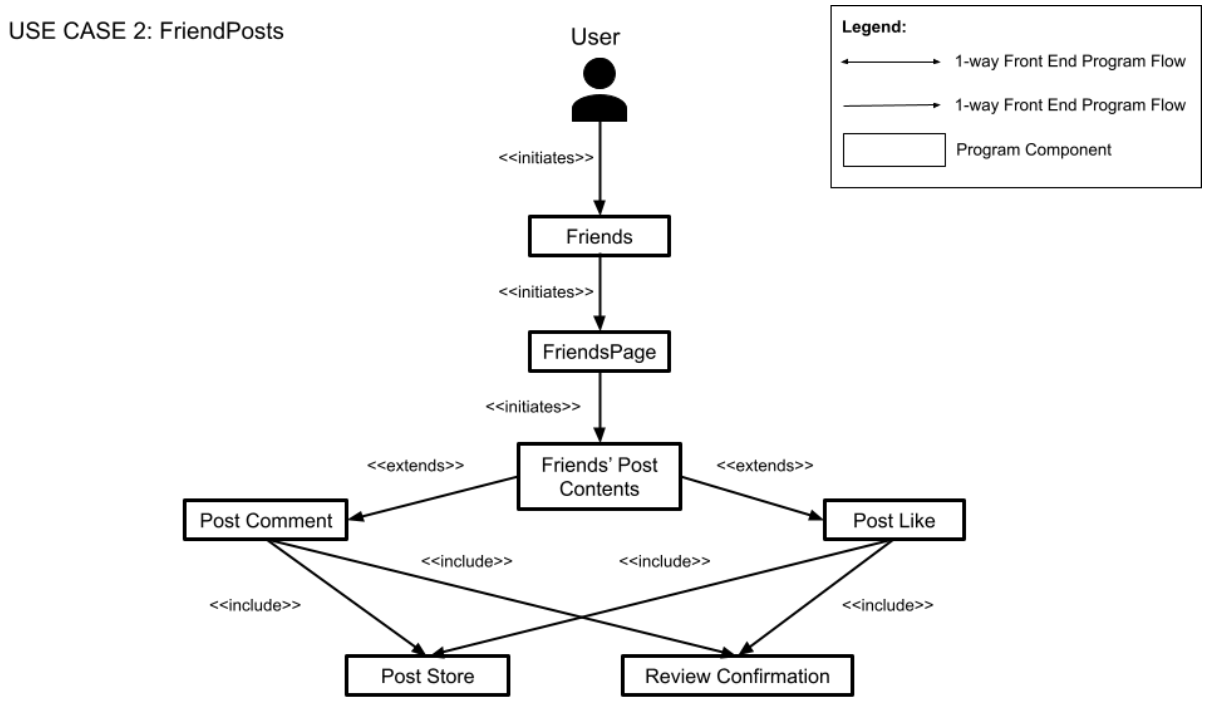

Figure 9: SnackMap FreindPost Use Case Diagram.

8. System Architecture

The system architecture follows a Model-View-Controller paradigm. The user interface is controlled by the view subsystem, which in turn uses the model subsystem to fetch the data pertinent to each view. The controller handles the querying and transferal of data between the database (which contains the information relevant to the user and their experience) and the GUI which handles the various views of the application.

Subsystem Decomposition and Services

Snackmap has 3, perhaps 4 subsystems.

- GUI (view)
- Model (content)
- Controller
- Database

GUI

The view subsystem handles the visual output painted to the screen for the various views the user may interact with the application through. It also handles collecting user input, which is then taken by the controller subsystem to update the content stored in the model and database subsystems. The view subsystem has moderate coupling as design changes to the user interface itself may require some alterations to how the controller interacts with it to update the database with information entered by the user. That said, once design decisions

about what field would be stored in the database, adjustments to the physical output of the various views do not require alterations to how data is queried and updated later on. The view subsystem has high cohesion as its elements are single-purposed.

Model

The model subsystem is responsible for database management, user profile and content changes, and the viewing of other users' content data. The model subsystem is responsible for asking the controller subsystem to fetch data from the database should user information be needed, as well as asking the controller to update the database based on information added by the user. This subsystem has moderate coupling as well, but slightly less than the view subsystem. If we wanted to add functionality to the model subsystem, little would be needed to be done to other subsystems, but restricting the scope of what the model does would require also restricting at least the database subsystem. It has high cohesion because the model class' methods are small in scope.

Controller

The controller subsystem handles state changes in the view as user data is changed. When new content needs to be shown, user input in the view subsystem would prompt the controller to ask the model to update its information and post it back to the user. Same for updating content, profile information, or even creating new users. It acts as the middle ground between the view and the model. It has pretty high coupling with the prior two systems, although the view would only ever be asking for information established as a field in the database early on, so design choices early about what user information will be stored improves this somewhat.

Database

The last subsystem is the database itself, responsible for storing the users' profile and content data. Information will be stored locally on an SQLite database. It has weak coupling as changes to the database do not require changes to the functions which query the database. It has strong cohesion because the information in the database is accessible only by SQLite's provided querying functions.

Software/hardware Mapping

Not applicable since Snackmap will not be mapped to any hardware.

Persistent Data Management

Data will be stored on a Sqlite server, which is a locally stored database queryable with SQL commands, thus eliminating the need for a third-party server/hosting service to store app and user data.

Access Control and Security

The user can access the app and its public user data if and only if they provide valid credentials, then allowing them to read their own data, and other users' public posts, write new data for their own new content and read/write their own profile data.

Global Software Control

The small scale of Snackmap and its few features means no synchronization will be needed to triage operations. Any requests for data instantiated by a reading use case or any need to update the information in the database can be carried out via a linear sequence of tasks carried out by the view, model and controller respectively.

Boundary Conditions

Upon startup, the view subsystem handles the display of the login page which cannot be progressed from until proper credentials are provided. When the system is shut down, users do not have to worry about state saving, the program will have all the required information upon its next startup as all pertinent data is stored in the SQLite server.

9. GUI Interfaces

The User Profile Page

The user can navigate to the profile page where past post interactions, preferred language, and privacy settings can be accessed and changed. Additionally, the user has the option to edit their profile. This brings the user to a different page where they can change their personal information for the account: profile picture, username, email ID, phone number, and password. These two pages can be seen below: the user profile page on the left, and the personal information page on the right.

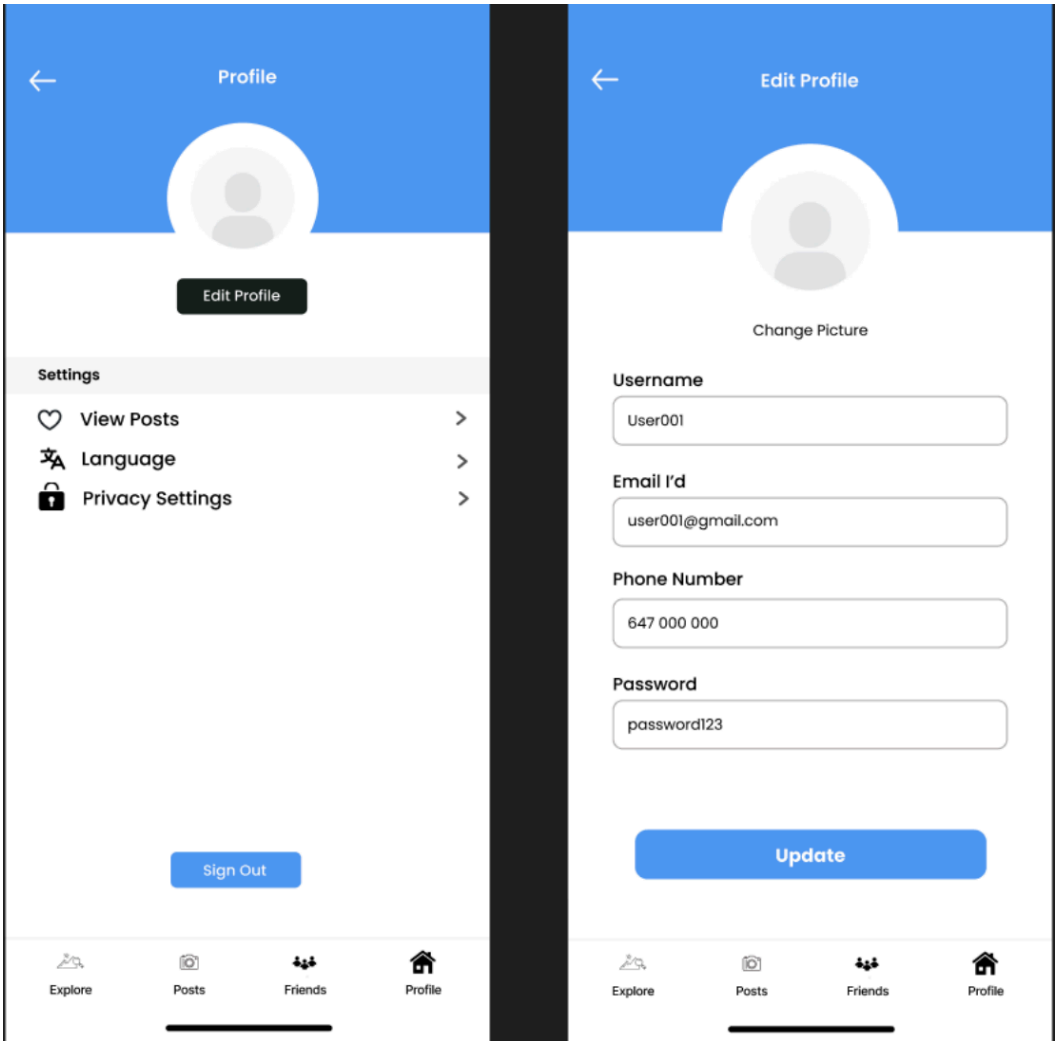

Figure 9: The User Profile page GUI interface.

The Home Page

Upon clicking the posts icon found on the bottom bar on the screen, the user is brought to a page where they can choose between “Add Post” and “Edit Profile”. If the user clicks “Add Post”, they are brought to the screen shown on the left below. Here, they can add “memories”, or photos, that give the user access to their camera roll to add pictures. The user is also prompted to write a review describing their “moments”. Once the user has finalized their post, they can then click the “Add Post” icon to post their review. The right screen is what appears when the user clicks “Edit Profile”. Here is where all posts a user uploads will be. The user can click on the three dots on a post to edit or delete their post.

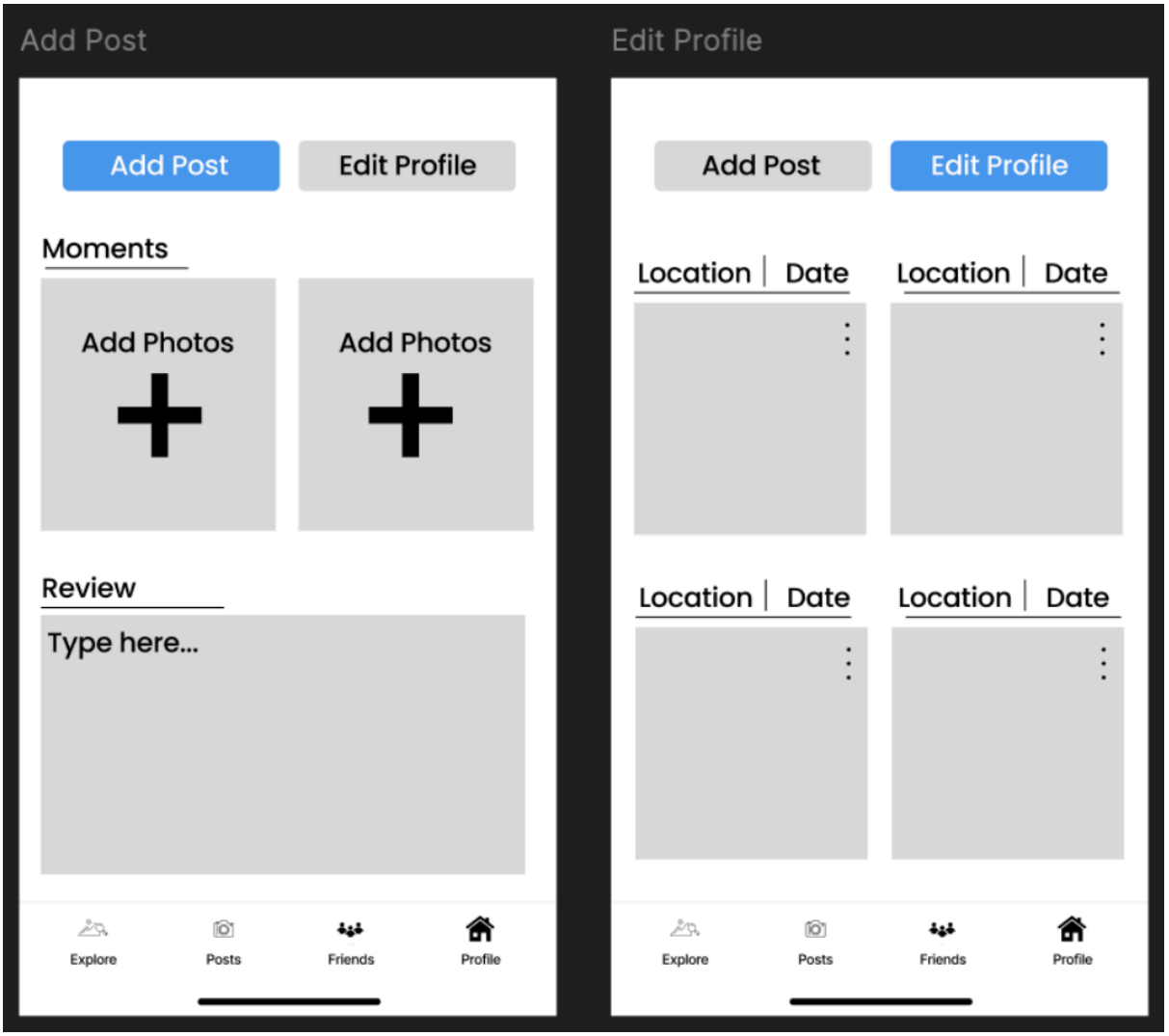

Figure 10: The Add Post and Edit Profile page GUI interface.

10. Coding Assignments

Member	**Code Assignment**
Andrea	Build and Develop User Classes
Jack	Post, Related, and Feed Classes
Lucy	Finalize and Implement GUI Design Framework
Eli	Post, Related, and Feed Classes
Xinwei	Build and Develop User Classes
Shyan	Build and Develop User Classes
Lucas	Finalize and Implement GUI Design

	Framework

11. Timeline