Unit

University Social Media

Software Requirements Specification | Version 1.4

February 11, 2020

Team: Adeel Asghar, Tyler Gross, Palak Patel, and Hala Ali

Clients: Thomas Anter and Jahnu Best

GTA: Son Dang

REVISION HISTORY

Date	Description	Author	Comments
1-29-20	Version 1.0	Hala Ali Palak Patel Tyler Gross Adeel Asghar	First Draft
2-10-20	Version 1.1	Adeel Asghar	Edited formatting
2-10-20	Version 1.2	Hala Ali Palak Patel Tyler Gross Adeel Asghar	Finished sections and added pictures and fixed formatting
2-11-20	Version 1.3	Hala Ali	Edited table alignment/fields and indentations in 2.4 and 2.5 as per GTA feedback.
4-15-20	Version 1.4	Adeel Asghar Palak Patel	Updated screenshots and added new Functional Requirements

DOCUMENT APPROVAL

The following Software Requirements Specification has been accepted and approved by the following:

Date	Title	Printed Name	Signature
	Software Engineer	Tom Anter	
	Software Engineer	Jahnu Best	

TABLE OF CONTENTS

1. INTRODUCTION

The Software Requirements Specification (SRS) document is a formal written contract between the client and the service provider which provides the services being offered by the provider, so the client will know the full extent of what functionalities will be included in the software product. This information is crucial for business and technical teams since it is used to protect themselves from liability. It is also used as a guideline for designing, implementing, and testing the product.

1.1 Purpose

The purpose of this SRS document is to provide a list of requirements that will be used to develop the Collaborative University Android App. The intended audience for this document is the technical team who are the developers who will ensure that the requirements for the software product are met and the stakeholders of the app.

1.2 Scope

The scope for the software product, Collaborative University Android App, is to create an interactive forum for WSU students to gain insight on classes that interest them. This native mobile app will allow students to subscribe to the community feed that they want to join. The community feed will be comprised of WSU classes where students will be able to post on the feed and be able to comment on another person's post. In order to gain access to these features, the user must first create an account by inputting their email address, username, and password.

The benefits of this app will be to create an online collaborative space for WSU students to explore classes they are interested in and ask general questions that students who are currently taking this class or have taken it before could answer. This forum could be used for students subscribed to the same class feed who are trying to sell off their textbooks connect with other students who might be taking that class next semester who are also in the same feed.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
WSU	Wayne State University
Jenkins	Automation server
Native	This term is used in context of applications that are designed to work on a specific type of platform. Ex: Native to iOS means that a software product can only run on iOS platform
Feed/Community	This term is used to describe a central area where people can communicate with one another about a specific topic.
User	The person who this application is being designed for in order to use.
UI	User interface
API	Application Program Interface
CI/CD	Continuous Integration/Continuous Development
Thread	A process in execution. Threads are needed to perform complex functionalities that take a long time and cannot be handled in the main thread.
NoSQL	Non-Relational Structured Query Language
SQL	Structured Query Language
JSON	JavaScript Object Notation
MVVM	Model-View-View Model, this is the architectural design pattern we are using in our application
UX Design	User Experience Design
HTTP	Hyper Text Transfer Protocol
SDK	Software Development Kit

1.4 References

1. IEEE Recommended Practice for Software Requirements ... (2010, February 23). Retrieved January 29, 2020, from https://cse.msu.edu/~cse870/IEEEXplore-SRS-template.pdf
2. Chugh, A. (n.d.). Android MVVM Design Pattern. Retrieved February 10, 2020, from https://www.journaldev.com/20292/android-mvvm-design-pattern

1.5 Overview

The rest of the SRS document contains the following: a general description, specific requirements, analysis mode, and appendices. The SRS is organized with 4 major parts and an appendices list. The first major part is the introduction includes a brief description of an SRS document, the purpose, the scope, the definitions, references, and an overview. The second part is the general description which includes product perspective, product functions, user characteristics, general constraints, and assumptions and dependencies. The most important section is our specific requirements which consists of external interface requirements, functional requirements, nonfunctional requirements, design constraints, logical database requirements, and any other requirements. The fourth part is dedicated to all the analysis models, among them will be the over-arching data flow diagram. The Appendices section marks the end of the SRS document which contains helpful information of our developmental process.

2. GENERAL DESCRIPTION

The general description section will provide background information for the specific requirements that will be mentioned in this document. It will compare the emerging software product with similar, existing products that are currently available to use. This section will also give an overview of the functionalities that the software will perform as well as how the general characteristics of the software's end users will affect the requirements. After that, developer limitations in designing the software will be discussed in the general constraints' subsection. Assumptions and dependencies will be the last topic in the general description which will go over any factors that will affect the software requirements in this document.

2.1 Product Perspective

There are many similar products available that include the same functionalities and are more advanced in terms of the services they provide and their performance level. This software product is a mobile app that allows users to join communities and communicate with each other through an online forum. This app will closely resemble Reddit. In addition to creating large online forums based on topic, Reddit has additional features where users can direct message other users and users can create their own communities. Voat.co is another example of a similar application. The difference between our application and these examples is our application will be focused on creating an online environment focused on creating a helpful environment for student.

2.2 Product Functions

The software will perform these actions: register user account, login to account, search for all class feed titles in a particular subject, subscribe to a class by clicking that class feed and joining that community, being able to create a post on any class/community he or she is subscribed to, being able to delete a post he or she has created on any subscribed class/community, being able to comment on a user's post.

2.3 User Characteristics

Although the software product will not be deployed at the end of semester, the potential eventual users of the product will be WSU undergraduate students. These students will come from various technical and non-technical backgrounds, so the product must be designed to be user-friendly with easy navigation. The eventual users of the product are mostly young people within the age range of approximately 17-35 and use apps daily. Due to the app markets for mobile operating systems providing high quality apps, the UX design for the software product must be a top

priority. Any lagging in performance of the app will appear to look unprofessional and the end users will get impatient, resulting in them deleting the app.

2.4 General Constraints

Possible constraints of the system include:

- Limitations on Firebase database reads(queries) in the system. This will prevent the developer from designing the system to include functionality for users to comment on another user's comment. In order to display multiple comments of a comment in a thread, the developer must design nested layers in the NoSQL database and make multiple calls to receive one comment. This will be costly and result in poor performance as the user must wait for the database to retrieve the nested data for multiple posts and show it on the screen.
- Adhering to the MVVM Architecture throughout the project. The developers must keep in mind throughout their implementation that they must restructure their code to have every activity connected to a view model and a view model factory. The developer must also remember to follow the practices of MVVM: the activity layer should use shallow logic to describe UI triggered events, the view model layer should contain business logic to connect to the repository, the repository layer should contain database and network calls implemented, and the Firebase Realtime Database layer should have account data and user activity stored.

2.5 Assumptions and Dependencies

The assumptions of the system include:

- The specific operating system (Android) will be available on the hardware designated for the software product. If the operating system is not available, then the software product will not be launched. When running the application, the operating system must be at least an Android 5.0 version (this application was tested with Android 5.0 Lollipop) with the android device or emulator having an API of 21 installed.
- The user cannot forget their email address. If they do, then they will not be able to recover their account and must contact the developers for support in order to get the email associated with their account. The system currently accounts for users forgetting their password but does not account for users forgetting their email.

The dependencies of the system include:

- Adhering to dependency injection rules with dagger to implement class dependencies throughout the MVVM architecture. This involves constructor injections, method injections, and field injections being implemented throughout the project.

- The view models for the system can only depend on one view model factory and one module due to multi-binding.

3. SPECIFIC REQUIREMENTS

This section will inform the reader of all requirements. This includes information about the external interface, related hardware and well as a detailed description of functional and non-functional requirements.

3.1 External Interface Requirements

The purpose of this section is to provide an in-depth overview of external interface elements. It provides snapshots of the current user interface and mockups of the incomplete user interface. Also, necessary information is provided on how users will be interacting with the application.

3.1.1 User Interfaces

All users that have not already logged in will be prompted with the login screen. This screen is displayed in Figure 1. If they do not have an account, they will be prompted with the register screen in Figure 2. Once they have registered, they will automatically be directed to the login page. After they log in, they will be directed to the main forum page in Figure 4. Users cannot proceed to the forum page without logging in. If the user logs in once and closes the app they will still be logged in when they open it again.

In the case that the user forgets their password, they will have the option reset it by clicking the link under the login screen located. From there they will be redirected to the password reset screen displayed in Figure 3. They will receive an email with a link and from that link Firebase will take care of the password reset process.

The main screen of our application, visible in Figure 4, is where the user will be able to view every post from every forum that they are subscribed to. The navigation bar at the bottom of this screen will be visible from the home, class search and messaging screens. The home button on that navigation bar will redirect the user back to the main forum screen. The user will be able to click on whatever post that they want to view to open it. The navigation bar features links to the create text post screen, the subscribed forum screen, the messaging screen and the search screen. These can be viewed in Figures 6,5, 9 and 8. For additional options, the user will user a navigation drawer that can be accessed by the menu button in the upper left corner of the main screen. The navigation drawer, shown in Figure 14, gives the user four options, profile, setting, view blocked users and logout.

When a post is opened from anywhere in the application, the user will be able to view post information as well as comments. A mockup of this can be viewed in Figure 13. From this screen the user can make a comment if they are subscribed to the forum that post was made to. After the

comment is made the keyboard will fold up and the comment text box will clear. The user can also swipe left on a comment to report it or block the user that made it.

On the create text post screen the user will be able to enter a title as well as additional information for a post that they would like to make. On the bottom there is a navigation menu that lets the user switch between the image posts and text post screens. The image post screen is similar however it features an extra button to add an image. When the done button in ether screen is selected the user is navigated back the home screen. The create text and create image post screens are visible in Figures 6 and 7.

The search screen, visible in Figure 8, allows the user to search for a forum as well as subscribe/unsubscribe. When the button is blue that indicates a user is subscribed and they can click on the button to unsubscribe and that will turn the button white. Once a class is selected the user is directed to that forums screen. There the user can view all posts from that forum made by users they have not blocked. They can also click on posts which will redirect them to the clicked post screen and if they swipe left the can report a post or block a user.

When the user navigates to the messaging screen located in Figure 9, they will see list of conversations they are in. The user can select a conversation to view it and send more messages. If the user wishes to start a new conversation, then can select the button in the upper right-hand corner to be directed to the user's screen which can be viewed in Figure 10. From there they can search and select a user to message.

If the profile option on the side navigation drawer is selected the user will be directed to their profile visible in Figures 15 and 16. This screen allows the user to view, there profile image, username, email and all other their posts and comments. These posts and comments are clickable and direct the user to the clicked post screen. From this screen the user can select the edit button and be directed to the edit profile screen. From there the user can edit their username, bio or profile image. This is visible in Figure 17. The edit post/comments screens can also be accessed from the profile activity by swiping left on a post or comment. They are visible in Figures 18 and 19. The user can also delete comments and posts. The swipe buttons are visible in Figures 21 and 22.

If logout is selected the user will be logged out. And if the user selects the settings option, they will be directed to the settings screen viewable in Figure 23. From the settings screen the user can select the theme of the application. If they wish they can switch to night mode. Some samples of the application in nigh mode can be found in Figures 23 – 25. If the user selects the

blocked user's option, they will be navigated to a page where they can unblock any user that they have blocked. This is shown in Figure 20.

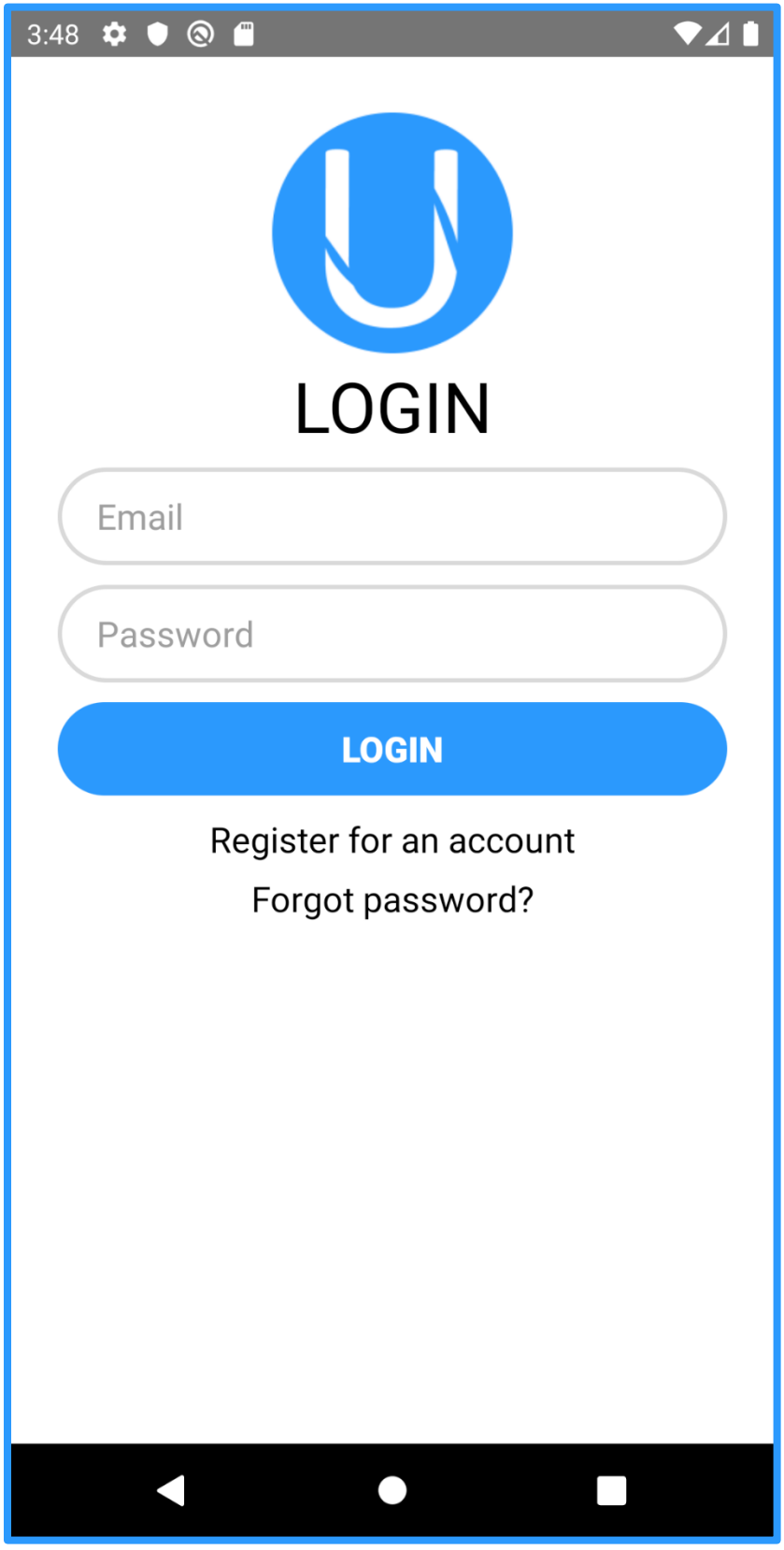


Figure 1: Login screen

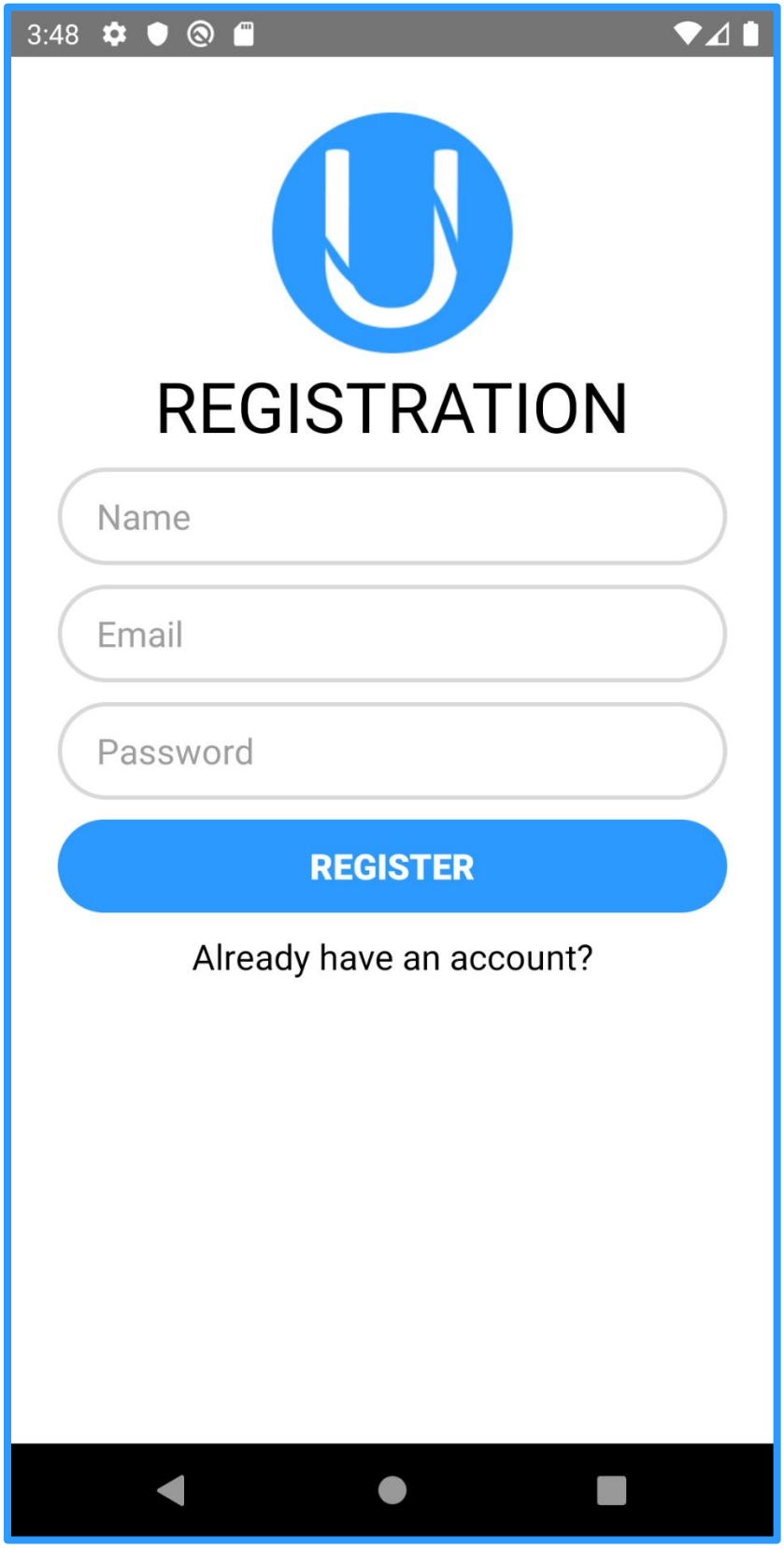


Figure 2: Registration screen

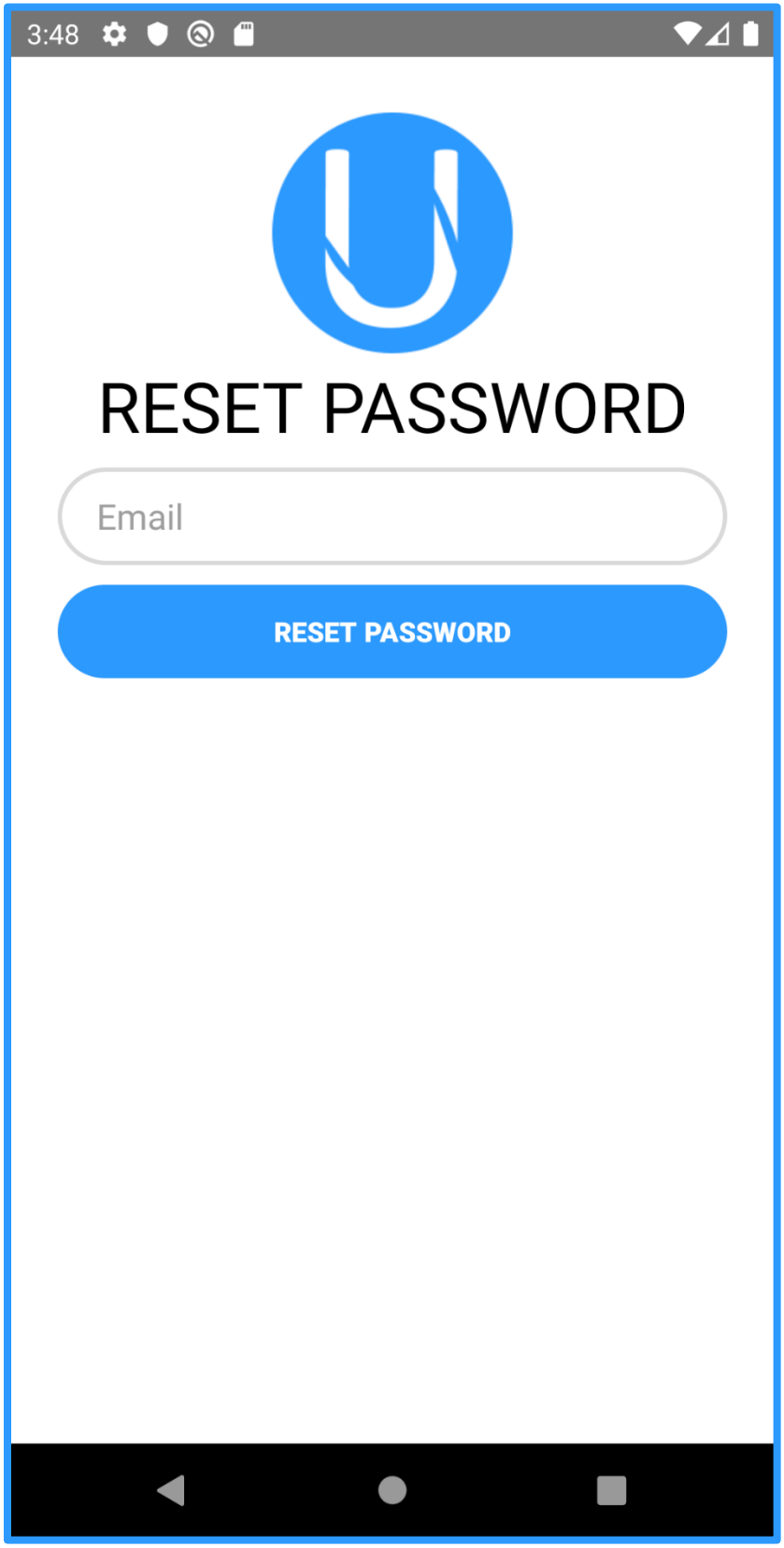


Figure 3: Password Reset Screen

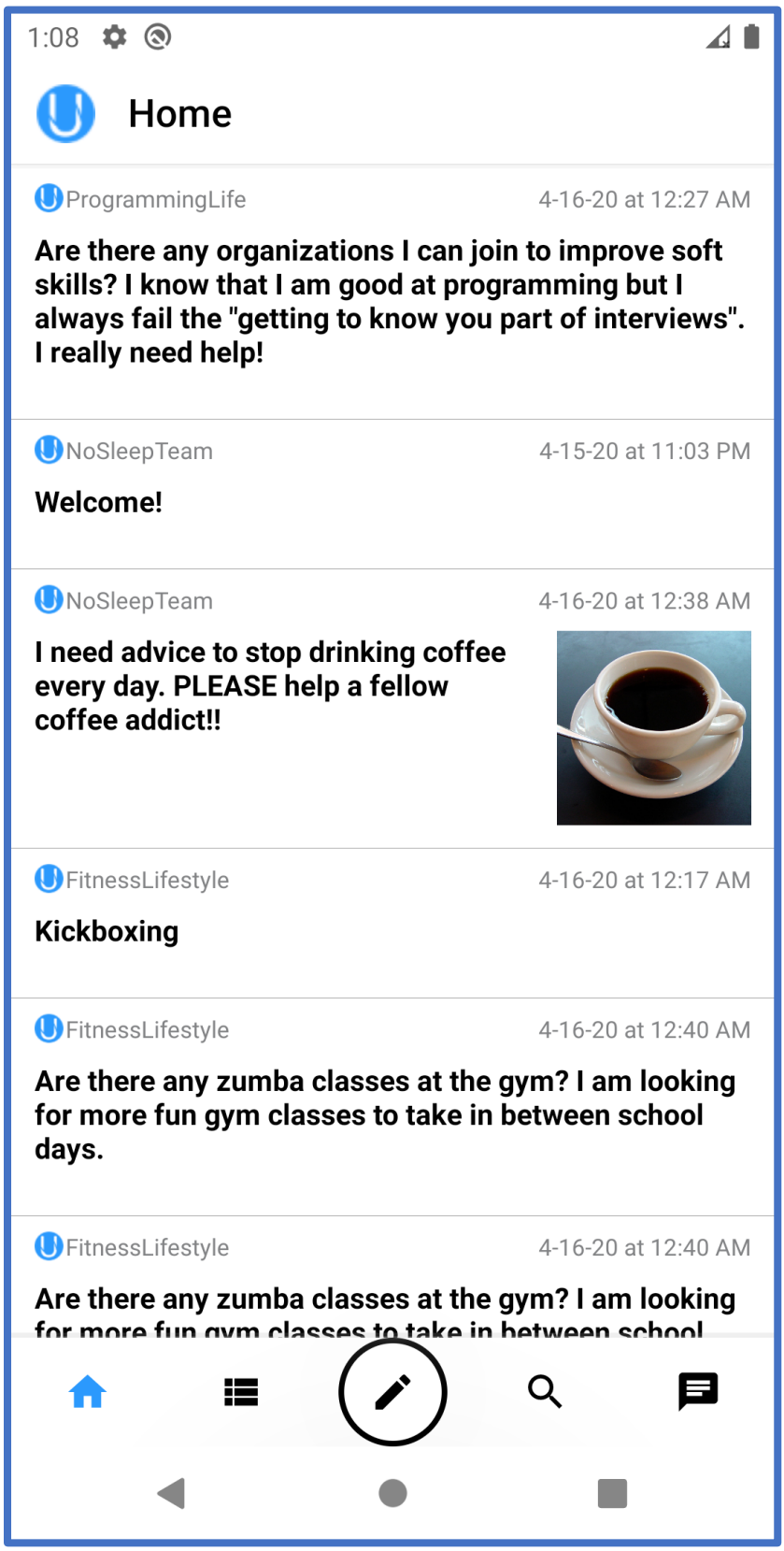


Figure 4: Home Page

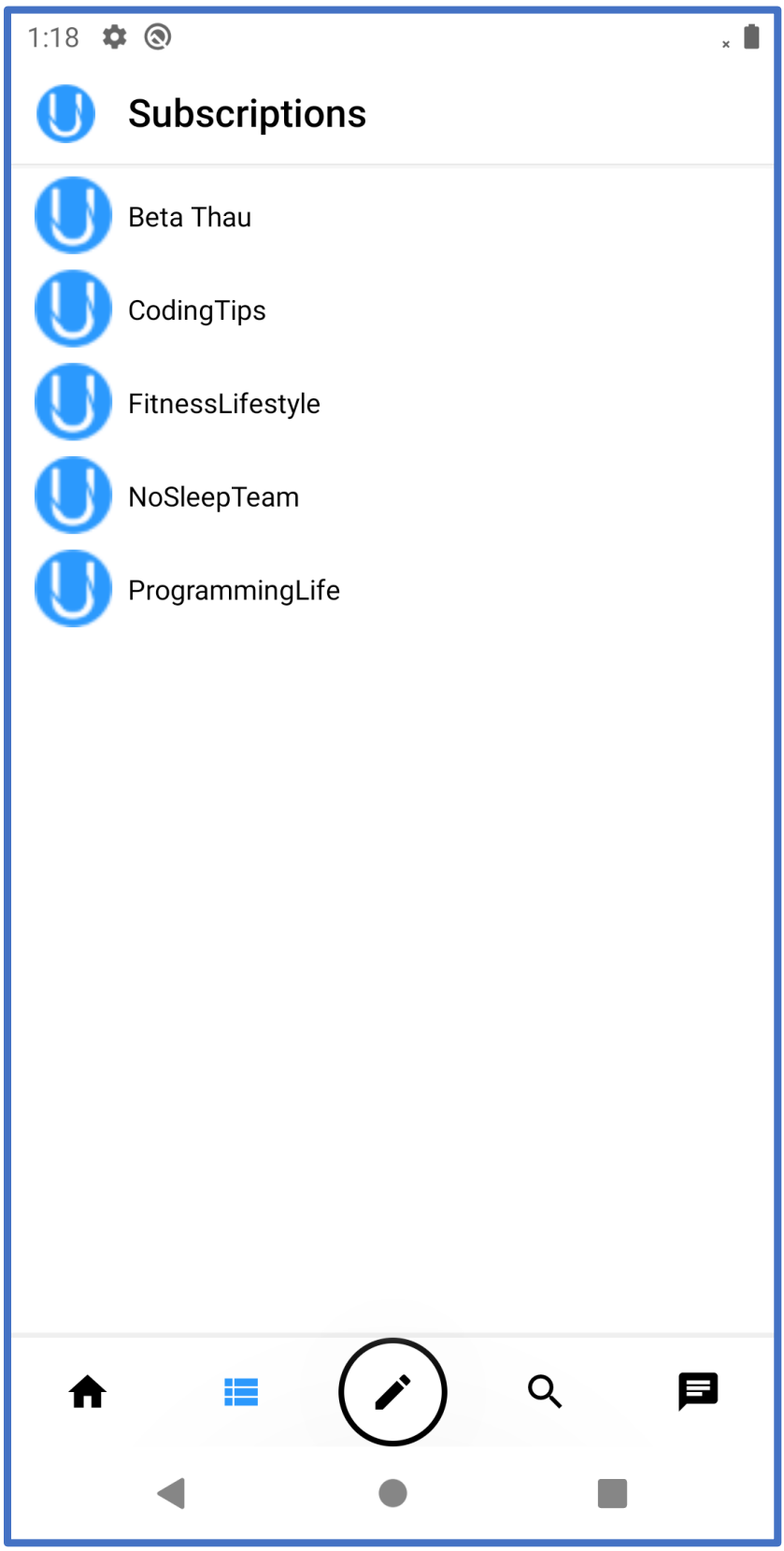


Figure 5: User's Subscriptions

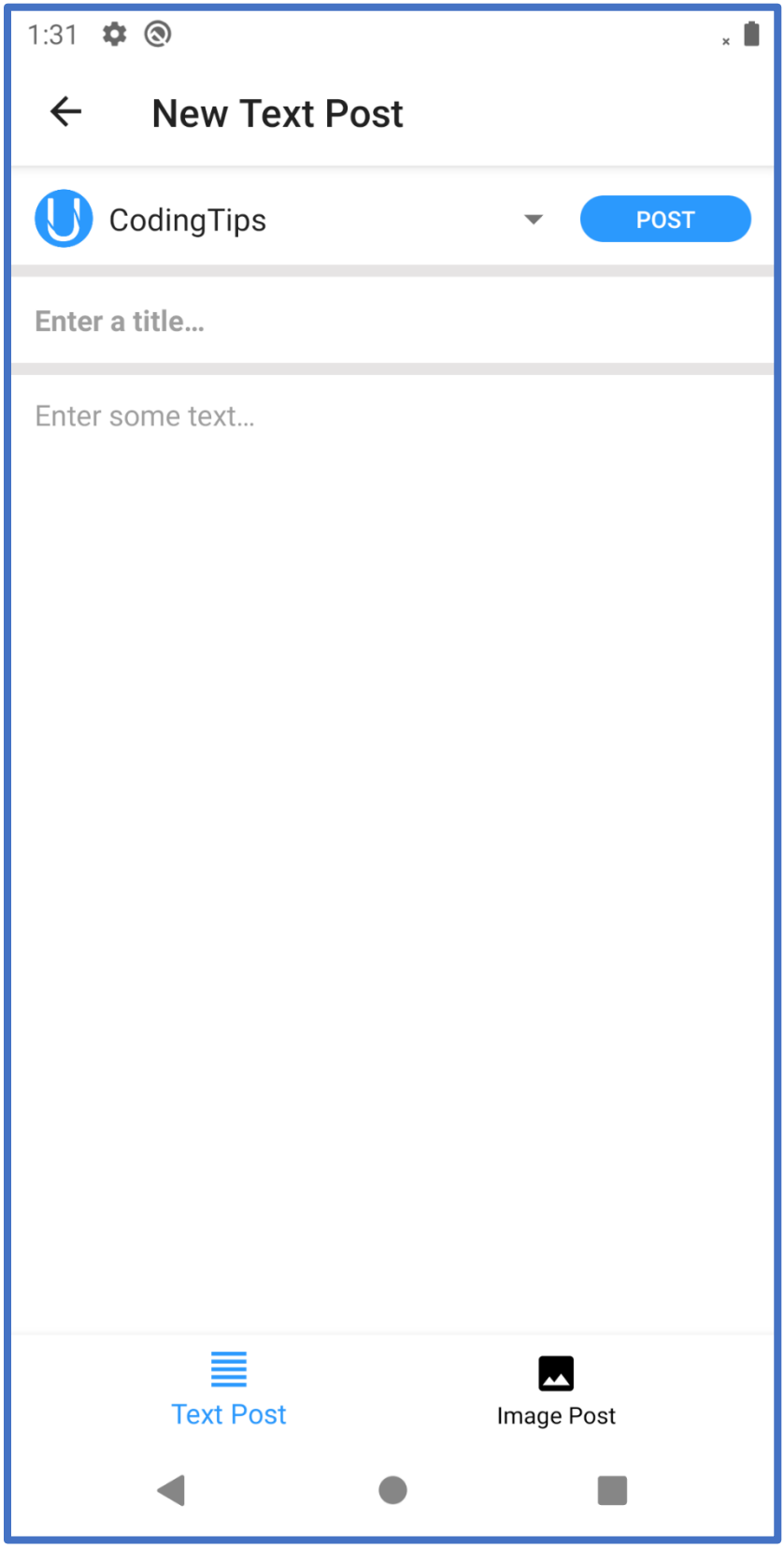


Figure 6: New Text Post

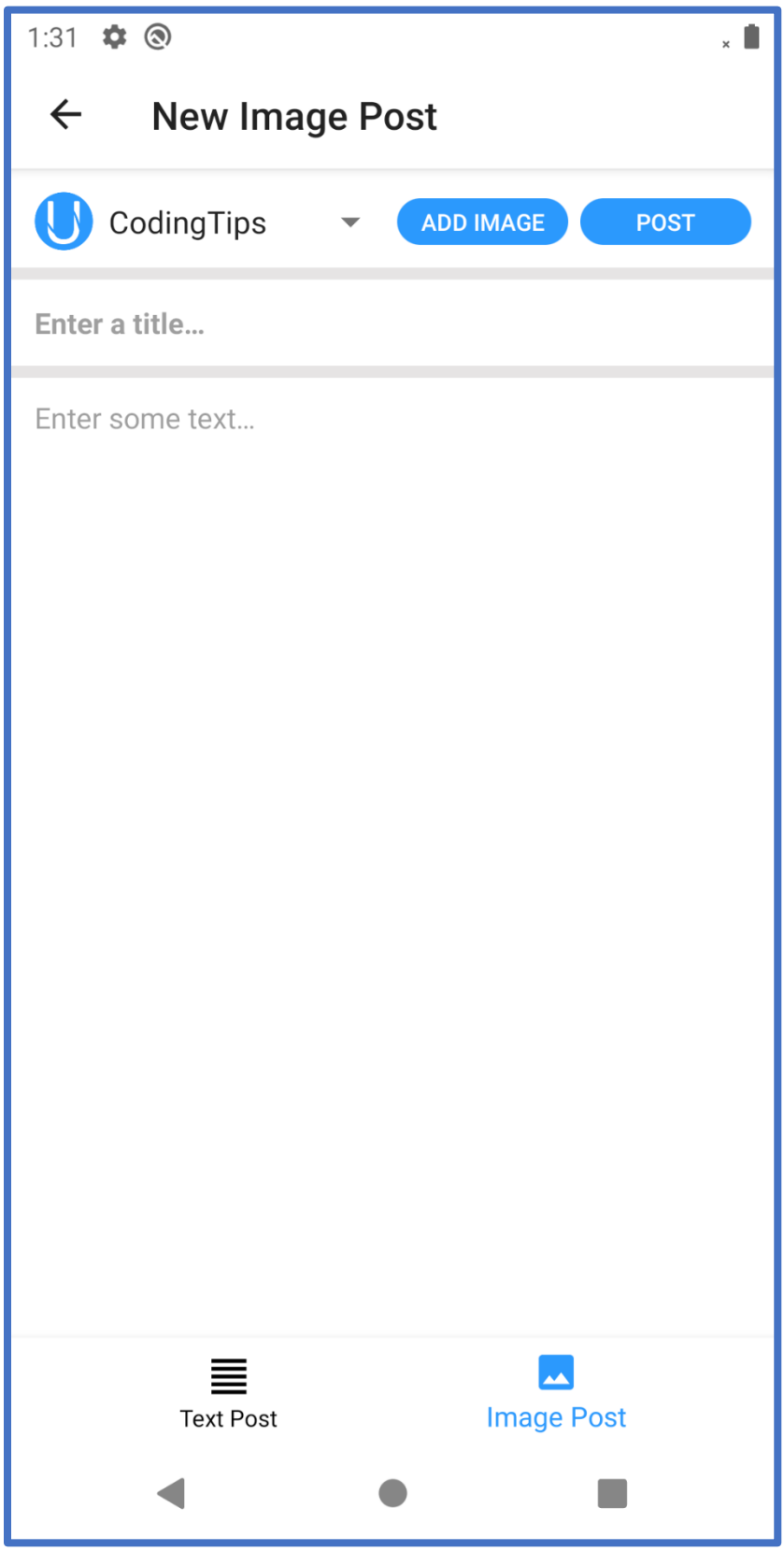


Figure 7: New Image Post

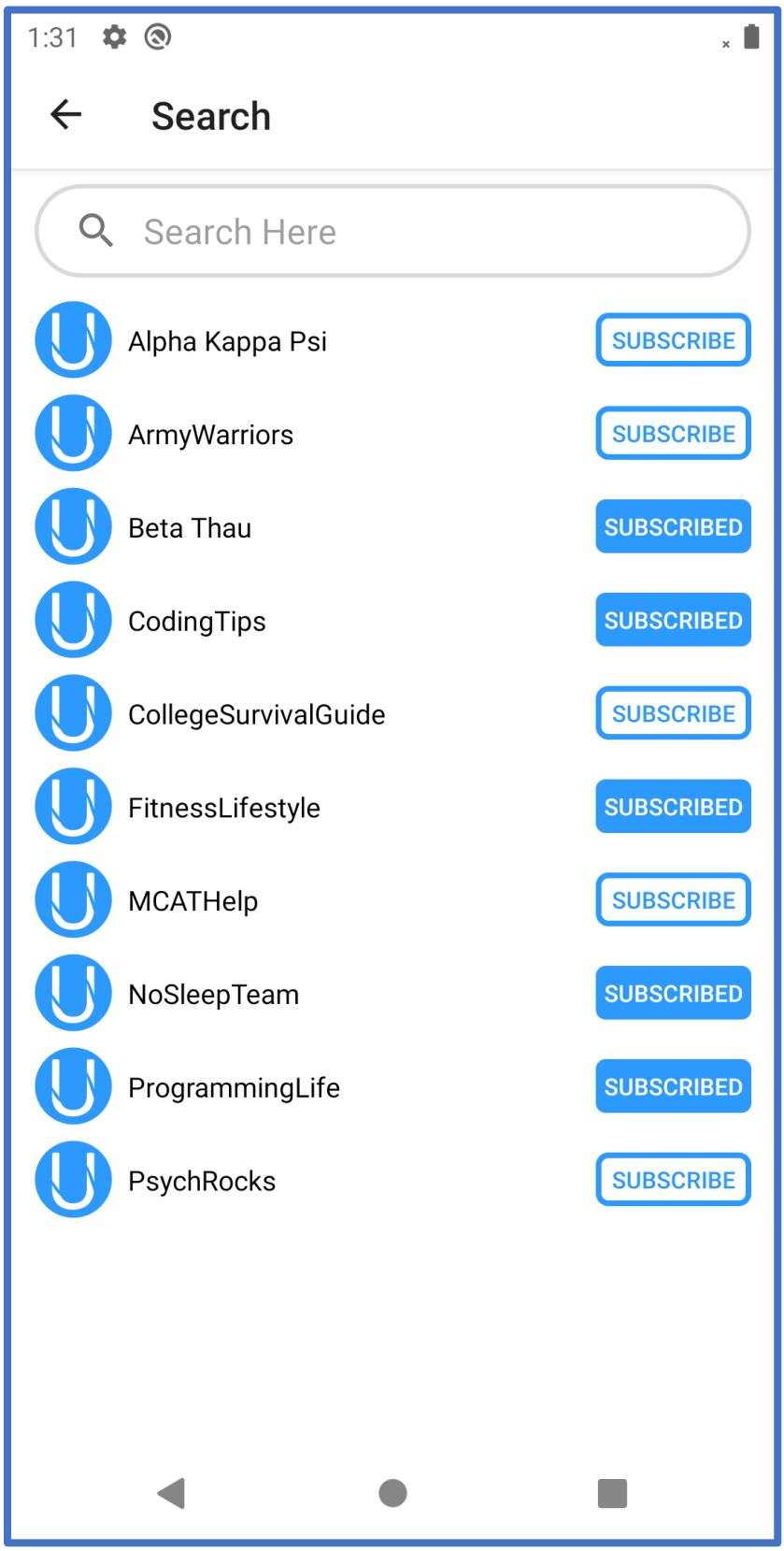


Figure 8: Class List

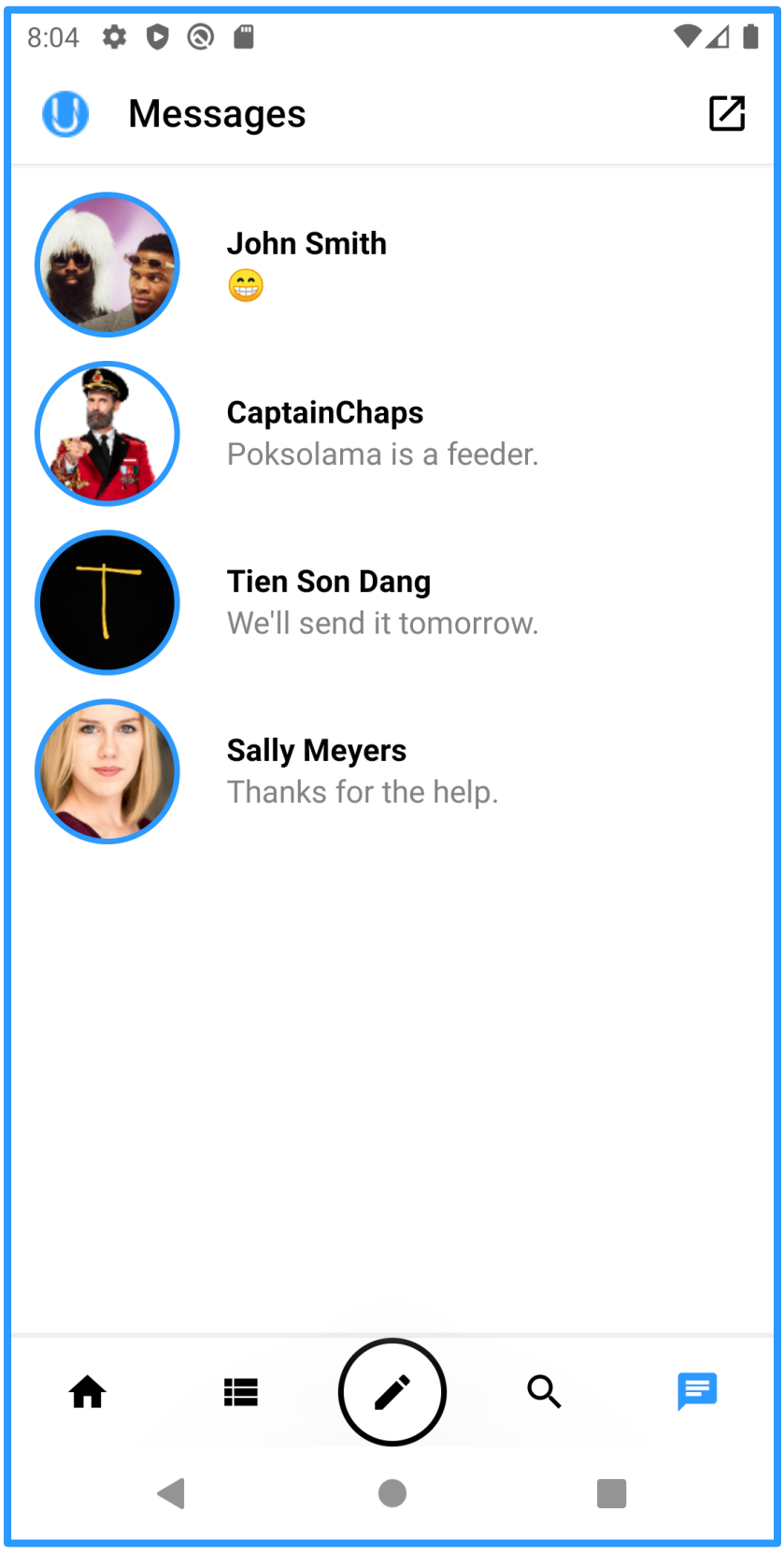


Figure 9: Recent Messages

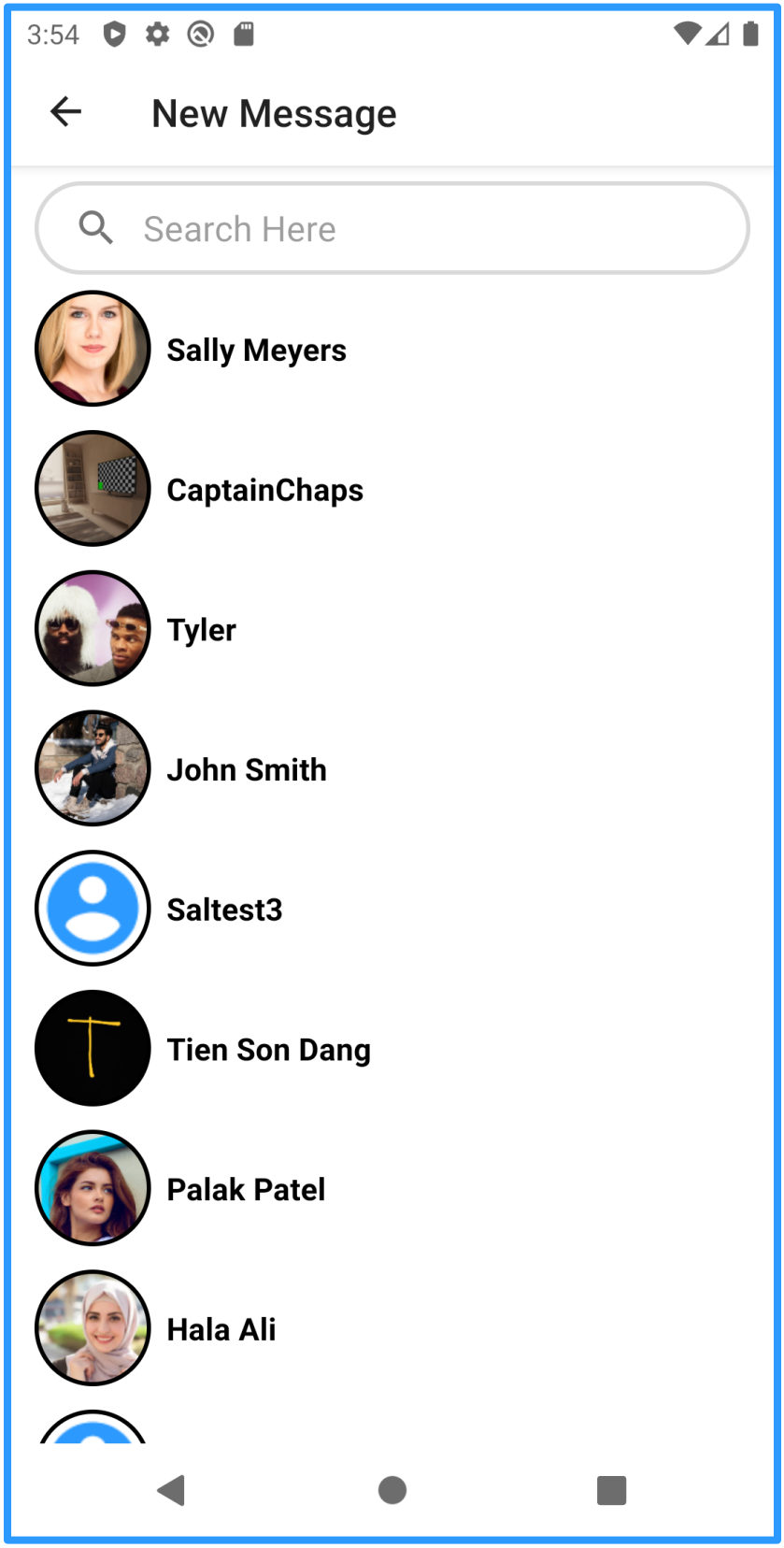


Figure 10: Send a New Message

Figure 11: Chatlog

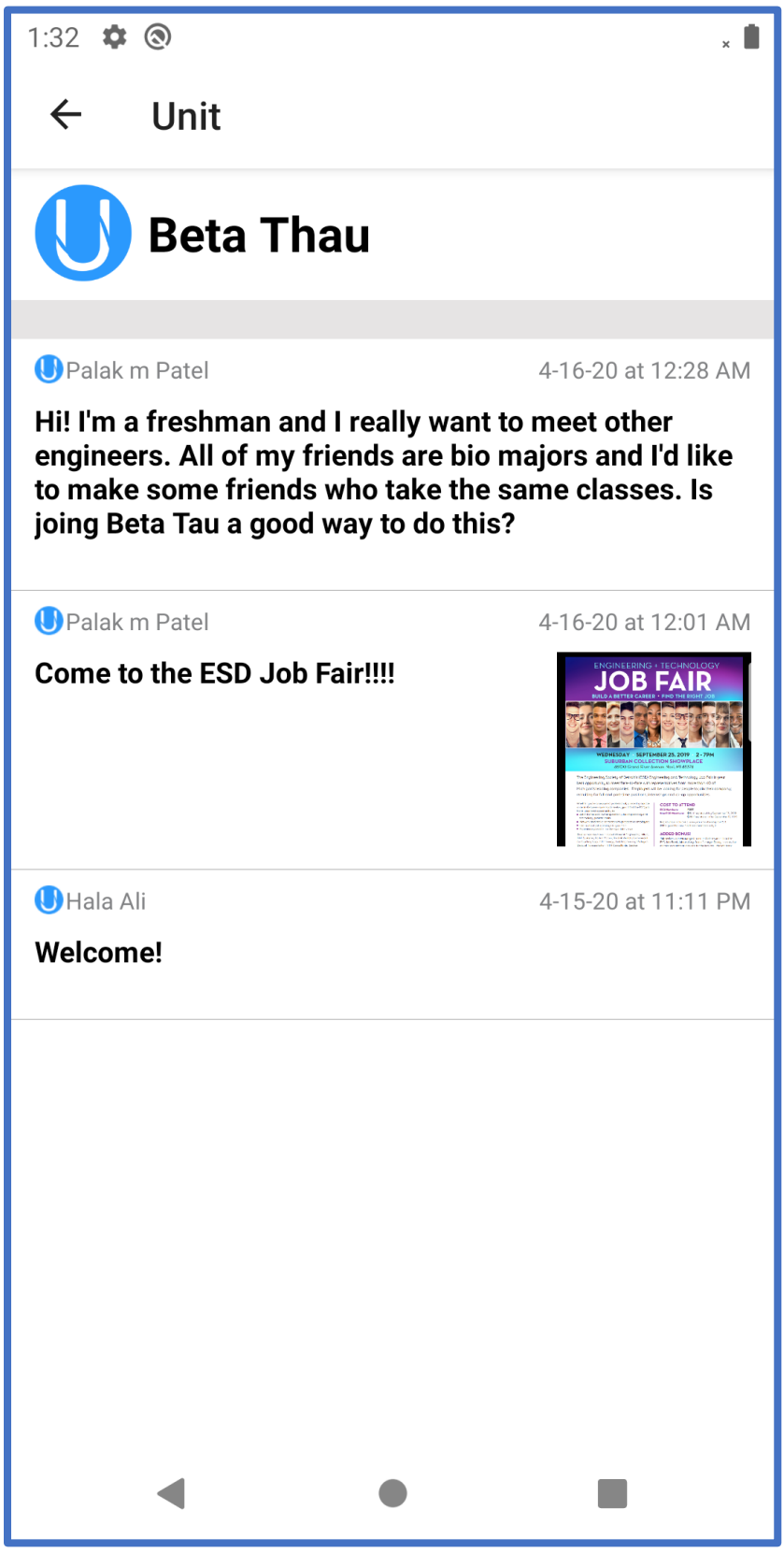


Figure 12: All posts for a class screen

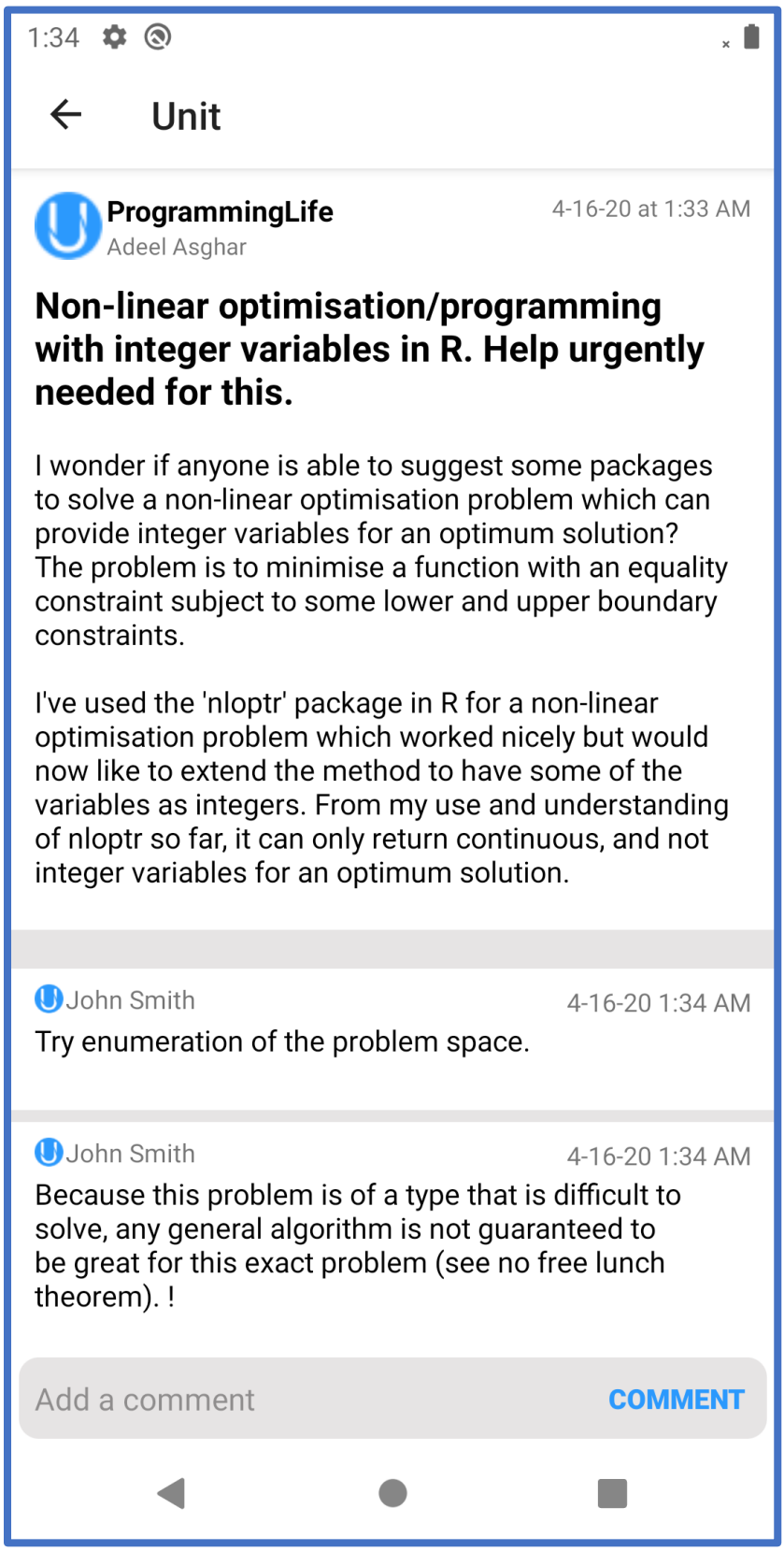


Figure 13: Clicked Post with Comments

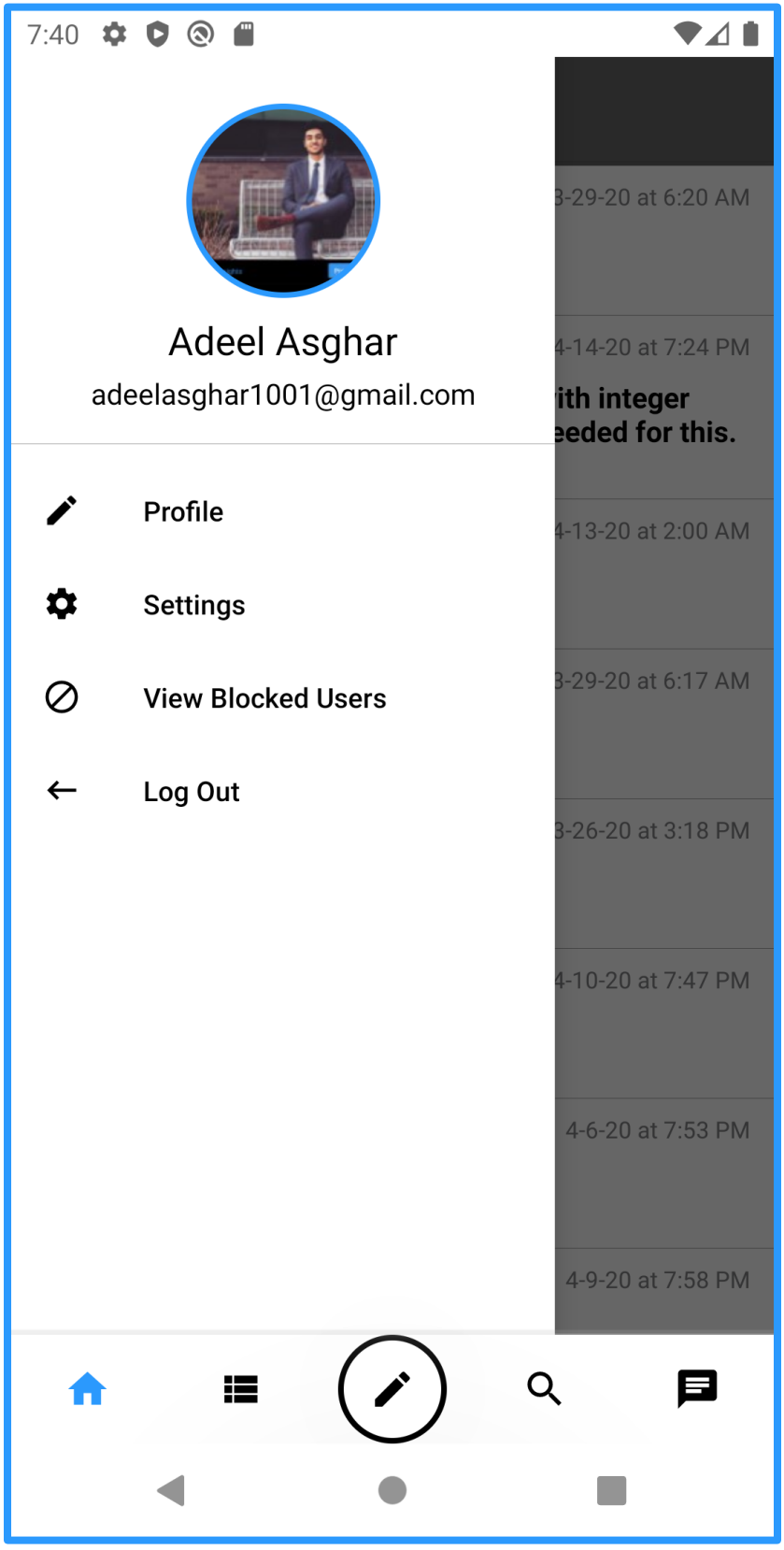


Figure 14: Navigation Drawer

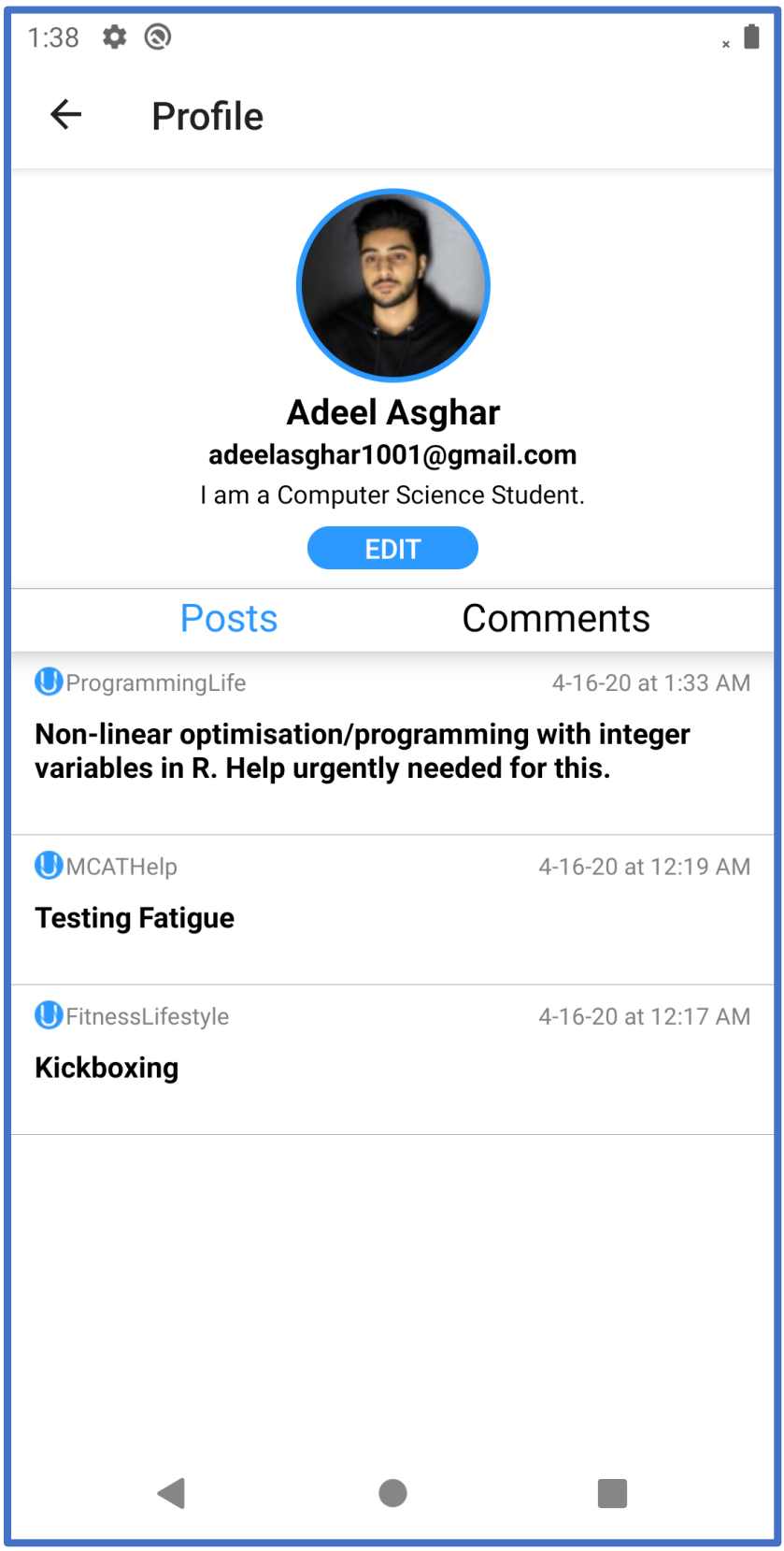


Figure 15: User Profile with User's Posts

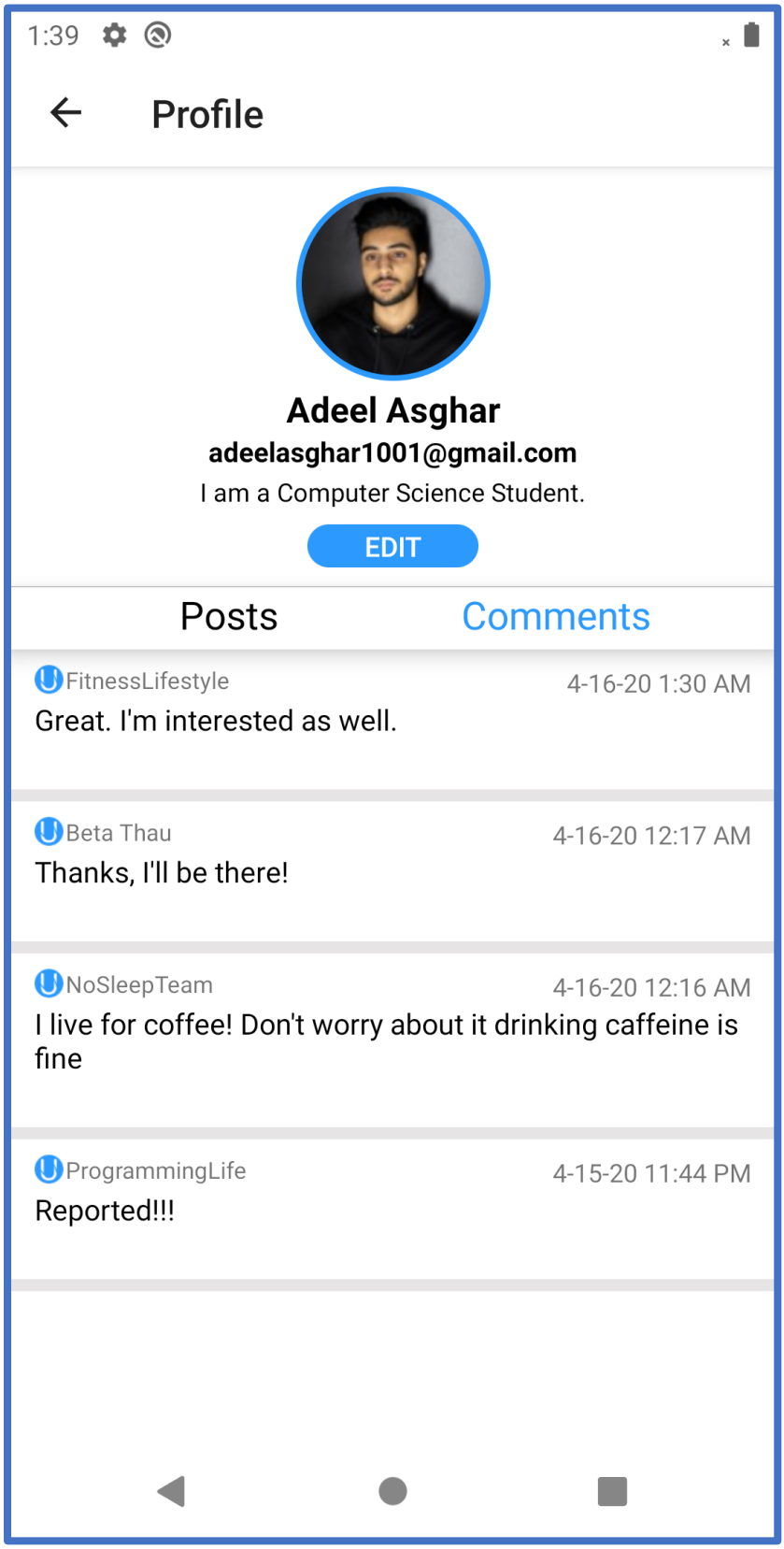


Figure 16: User Profile with User's Comments

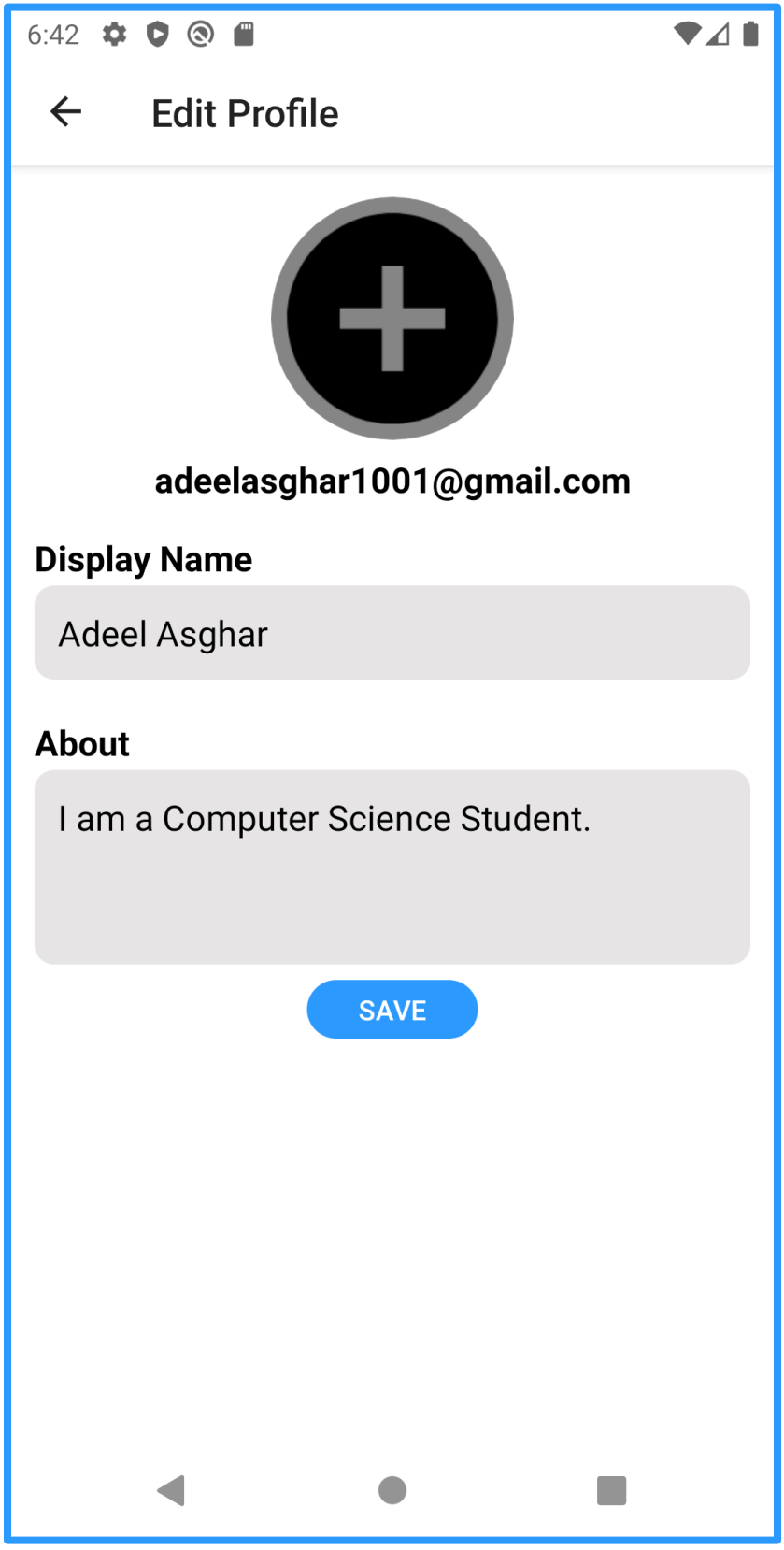


Figure 17: Edit User Profile

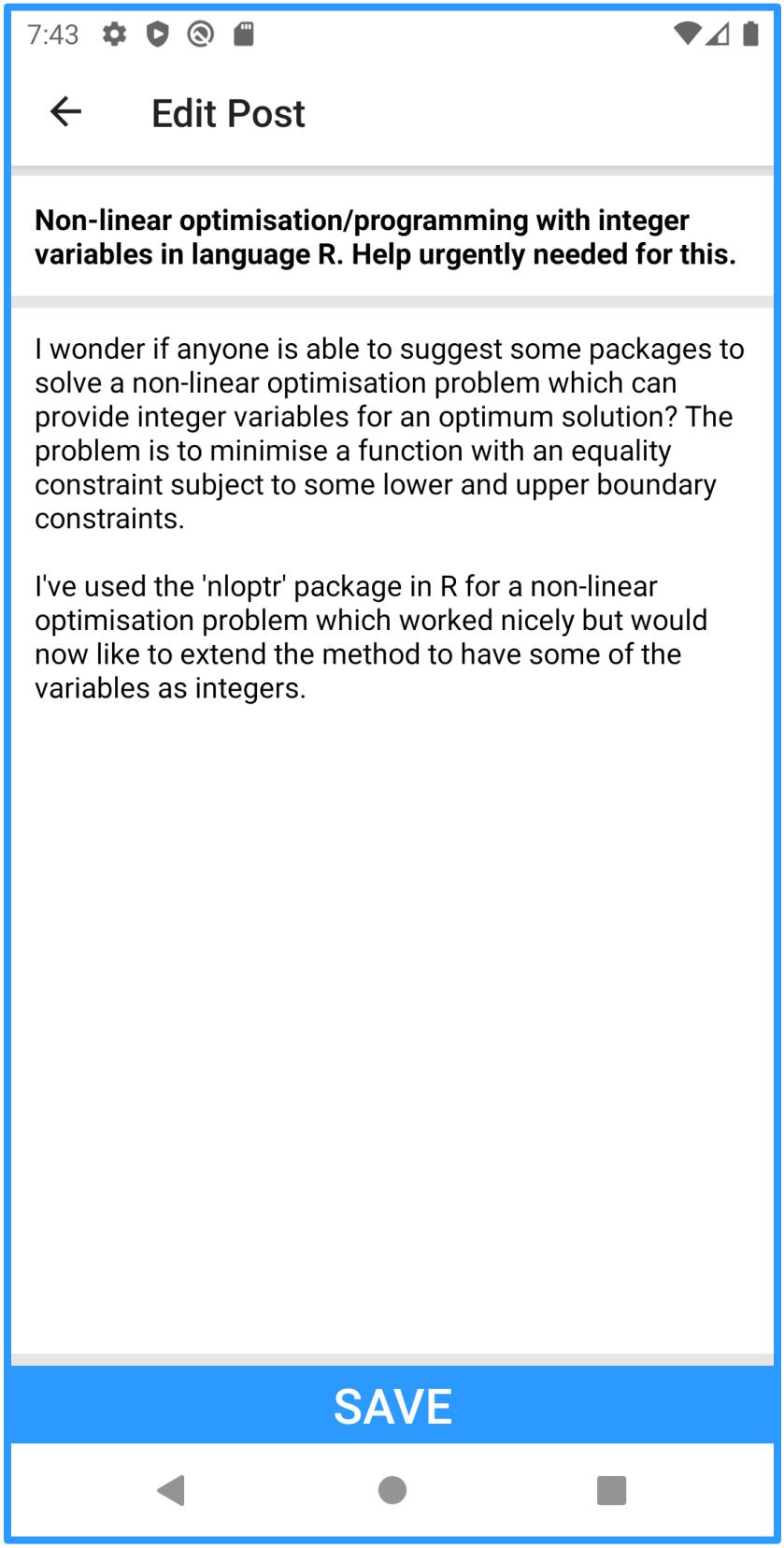


Figure 18: Editing a Post

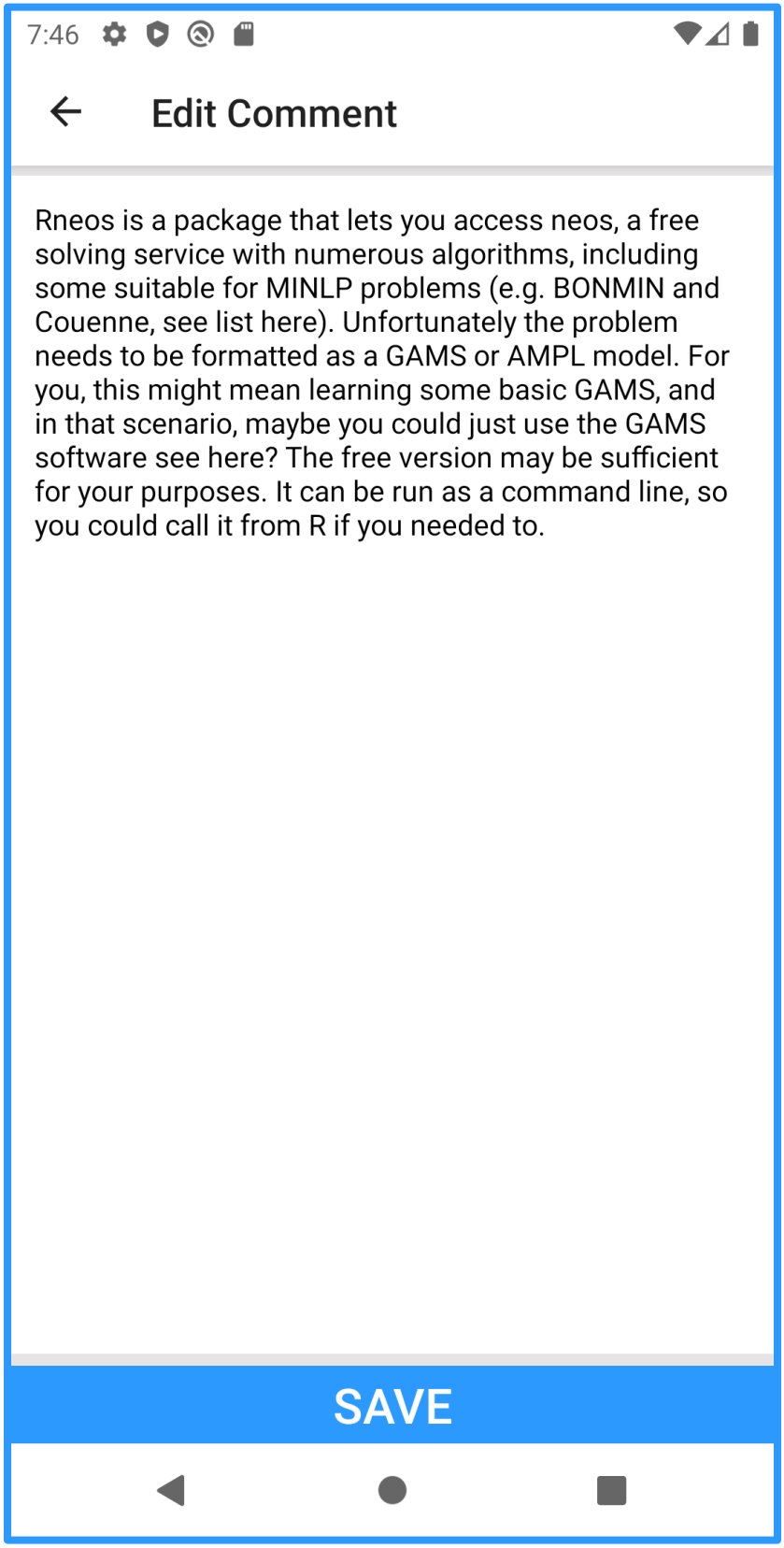


Figure 19: Editing a Comment

Figure 20: Blocked Users

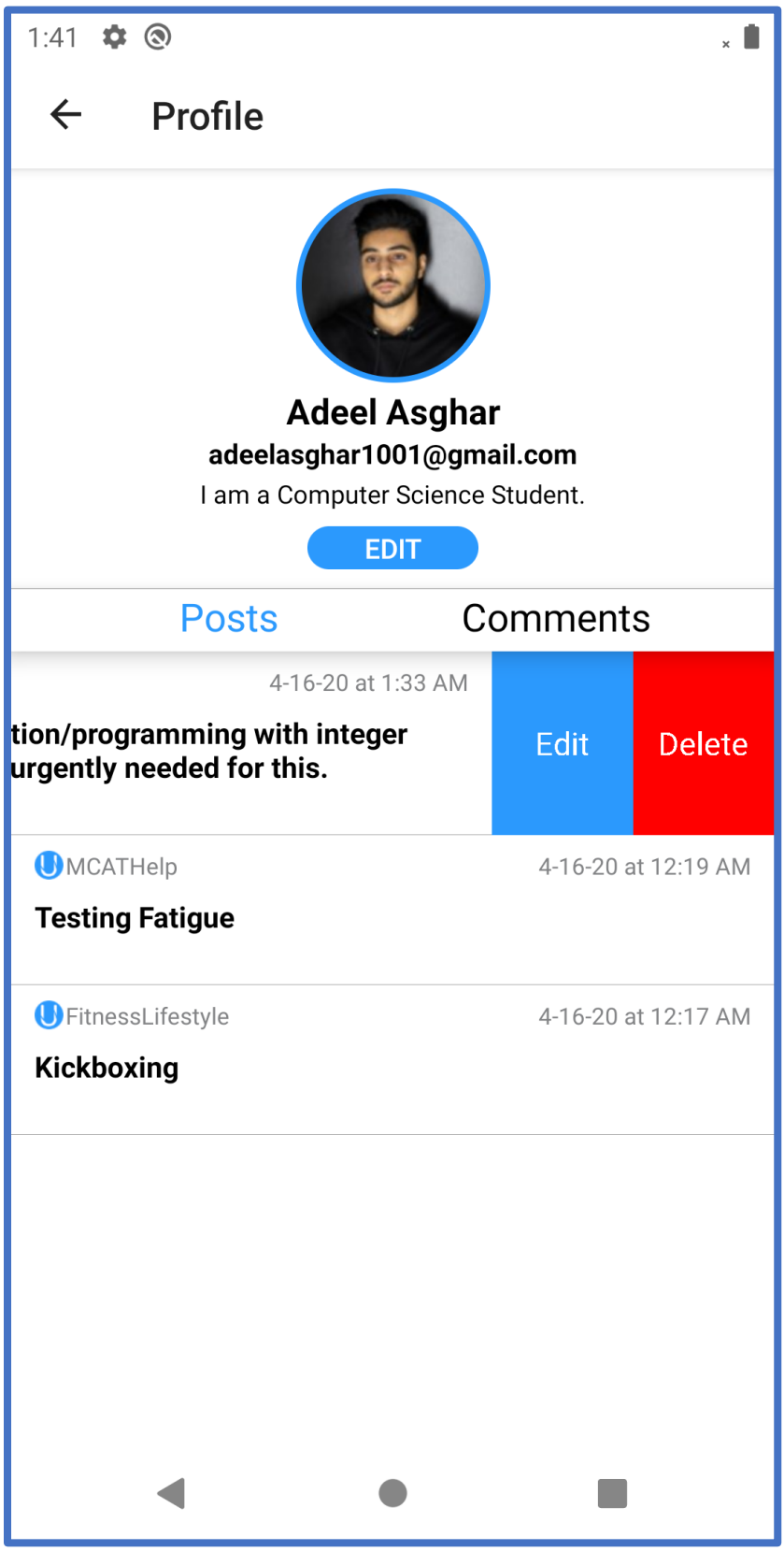


Figure 21: Editing and Deleting Posts

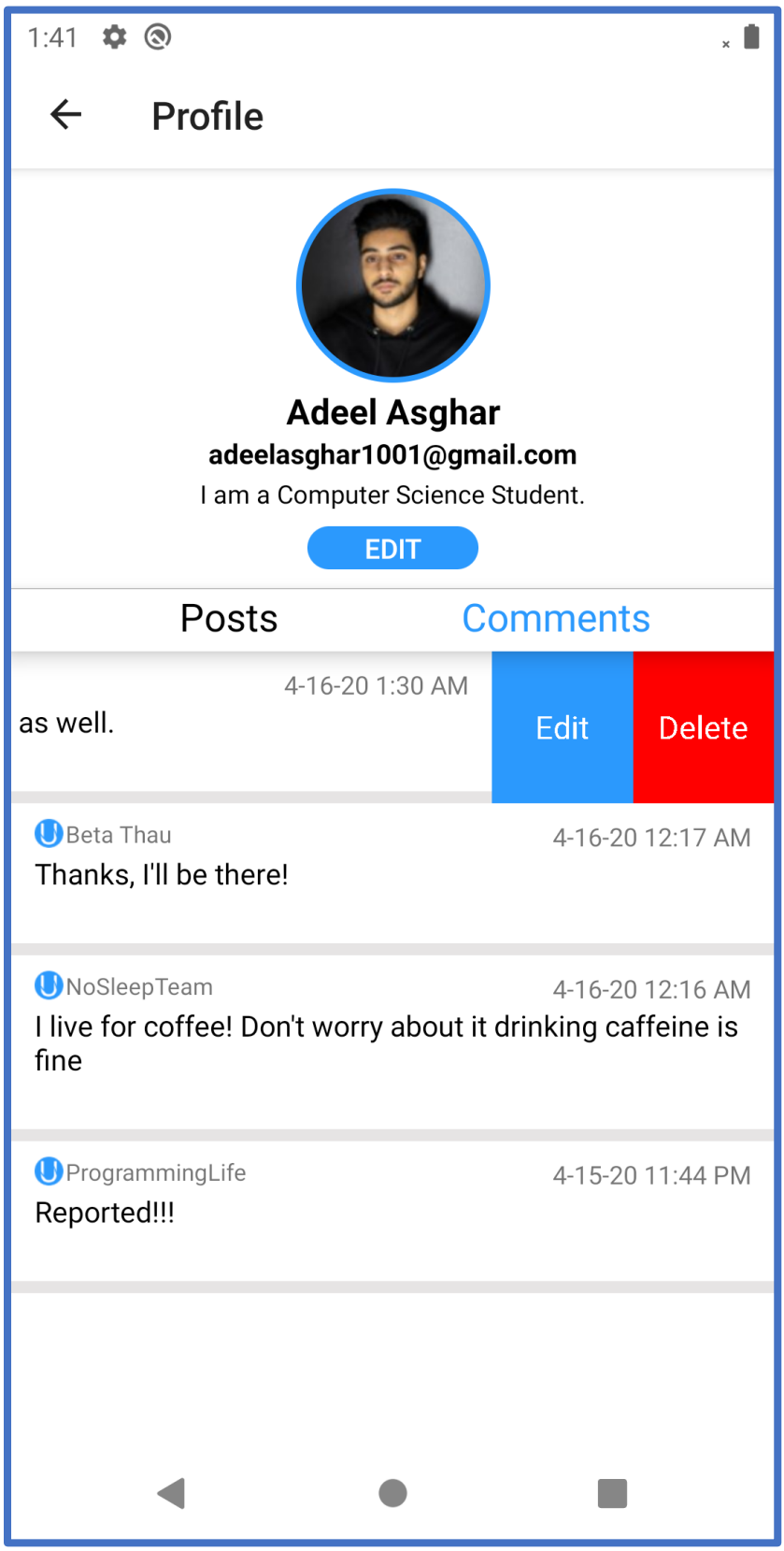


Figure 22: Editing and Deleting Comments

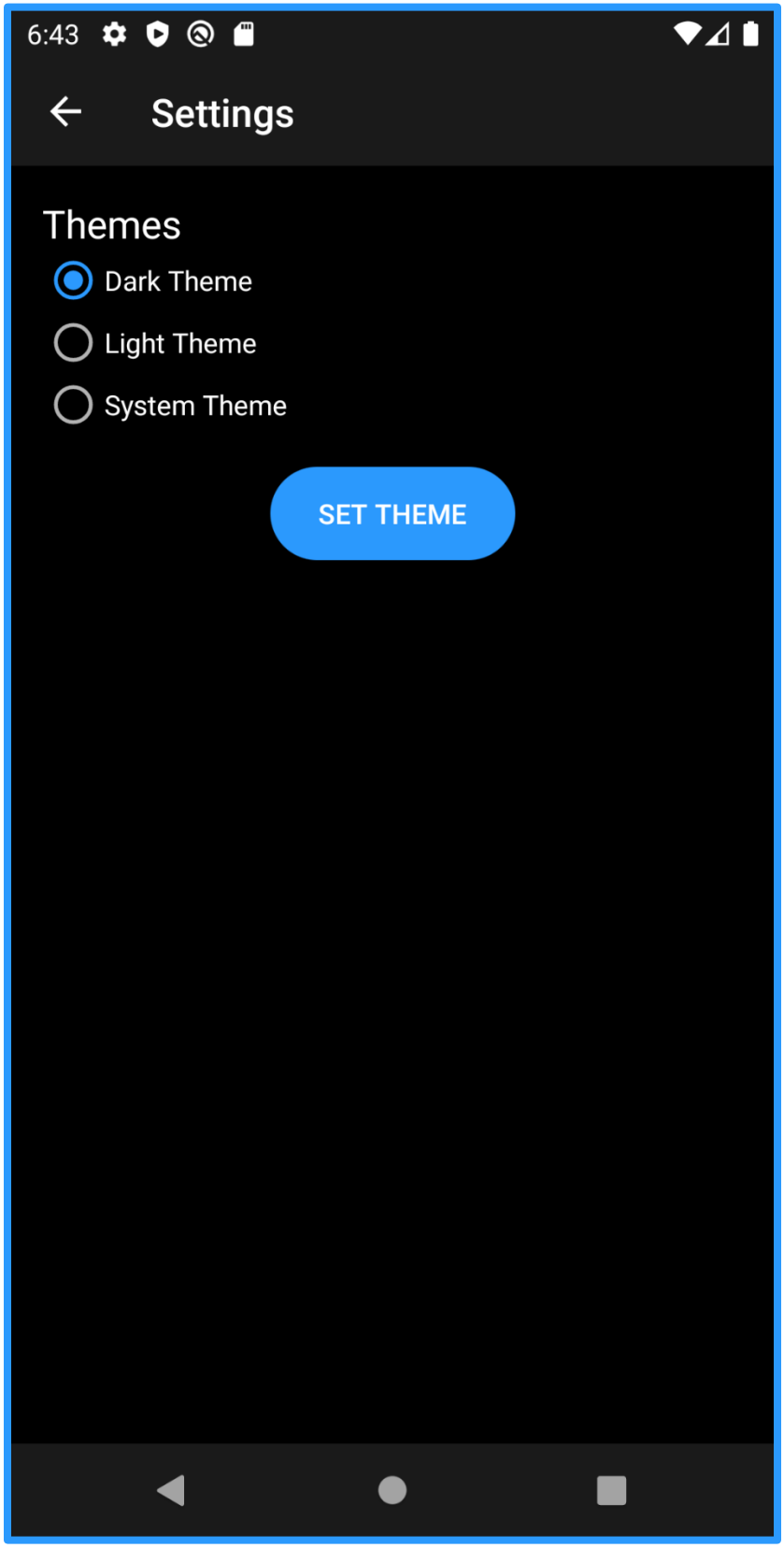


Figure 23: Enabling Night Mode

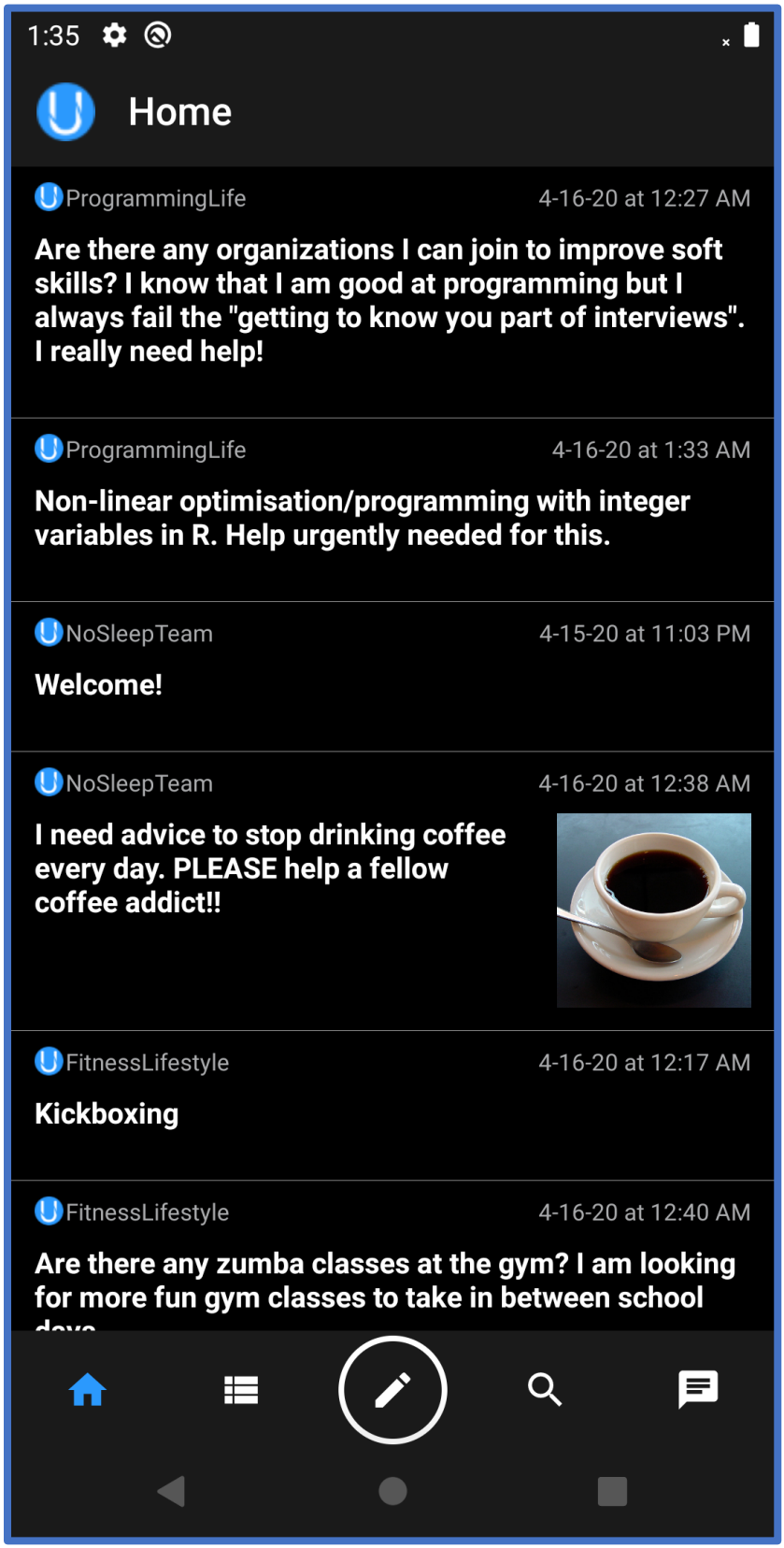


Figure 24: Home Page in Night Mode

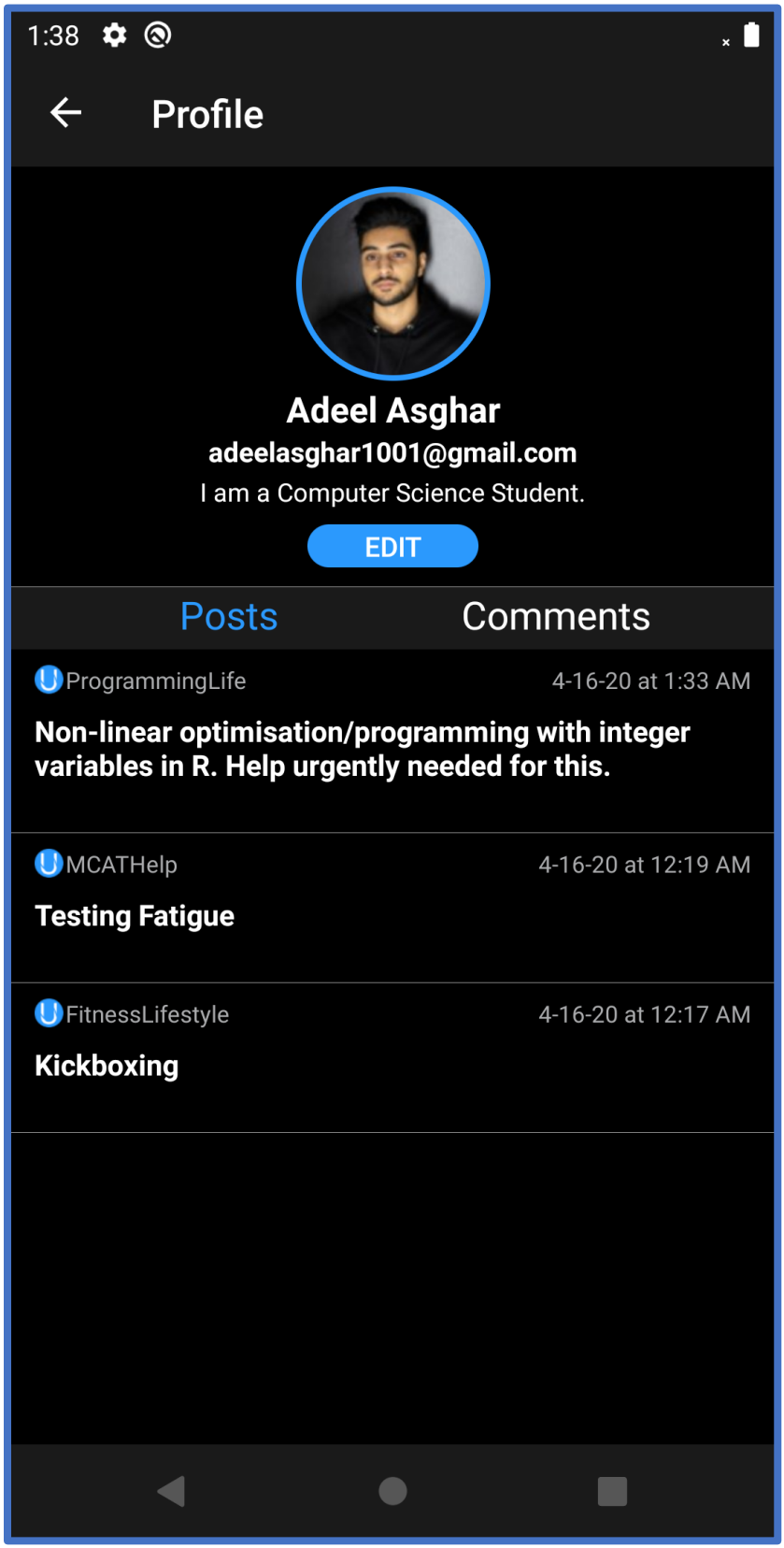


Figure 25: User Profile in Night Mode

Unit

3.1.2 Hardware Interfaces

Our application is designed for android mobile devices. It does not have designated hardware besides the android device that the application runs on.

3.1.3 Software Interfaces

Our application communicated with the Jenkins software interface for automating builds and tests. Jenkins is a management system that helps the development team track build stability and test reports on the software product to ensure smooth continuous integration and continuous delivery.

3.1.4 Communications Interfaces

All communication is achieved from underlying operations by the android device.

3.2 Functional Requirements

3.2.1 FR1: User Registration	
Introduction	This feature fulfills the initial user contact with the app. The user creates an account in which they will be able to complete further activities in the app and can login after they sign out.
Inputs	The user must input a username, their email address, and a password. When the user selects the button “Register”, the user will be taken to the main forum page.
Processing	The system shall store the username, email, and password into the NoSQL database for future reference. The register button will trigger a new activity screen that is the main homepage of the forum.
Outputs	An account is created for the user and the user can create a post.
Error Handling	When the register button is clicked, if the user is not taken to main homepage screen, then a message will appear to the user indicating that there is an error with their inputs. These messages could range from “You already have an account”, “that username or password is already taken”, or “One or more input fields are empty”. If the user already has an account, they can click on the text “Already have an account?” to access the login page and type in their email and password.
Dependencies	Forward Dependencies: FR3.2.2 Backward Dependencies: N/A
Priority	2

3.2.2 FR2: User Login (Use Case Scenario= FR9 Forgot/reset password)	
Introduction	This feature is responsible for registered users connecting to the app and gaining access to all the forums they are subscribed to.
Inputs	The user must input an email and password. When the user selects the button “Login”, they will be taken to the main homepage screen.
Processing	The system shall check the Firebase NoSQL database entity known as “user” in order to match the user input with the stored emails and password. If a match is found, the user will be connected to their homepage.
Outputs	The user is taken to the main homepage screen where they can view all the posts of the forums they have subscribed to.
Error Handling	After the user has entered their email and password and clicked “Login”, if the user is not taken to the main homepage, an error message will be presented to the user. That error message will be “the email and/or password entered is incorrect”. If the user does not have an account, they can click on the “Register for an account” text and complete the steps on that screen. However, if they forget their password, a screen will be taken so they can enter their email and click the button “reset password.” Then an email message will be sent to them with steps on how to reset their password.
Dependencies	Forward Dependencies: FR3.2.2, FR3.2.3, FR3.2.4, FR3.2.5, FR3.2.6, FR3.2.7, FR3.2.8, FR3.2.9 Backward Dependencies: FR3.2.1
Priority	1

3.2.3 FR3: Subscription to Community Forums	
Introduction	Once a user has entered the main homepage screen, they are able to choose a community forum to subscribe to. These ‘communities’ will be class titles set up by the developers. Once they select the class forum that they would like to join, a message will alert the user that they are now subscribed to that class. This means that they can view all the posts and comments on that forum.
Inputs	The user must select the class forum that they would like to be subscribed to.
Processing	The system shall first send a message to the user to let them know they are now subscribed to the class forum they clicked on. After that, all activity on that forum page will be displayed on the current screen.
Outputs	The user has access to all posts and comments on that forum.
Error Handling	If the user clicks on the class and is not able to join that community, an error message will appear first alerting the user if they are already members of that community. If not, then the error message will appear to the user that the community they are trying to subscribe to cannot be accessed.
Dependencies	Forward Dependencies: FR3.2.4, FR3.2.5, FR3.2.6 Backward Dependencies: FR3.2.2
Priority	5

3.2.4 FR4: Post to Subscribed Forum	
Introduction	The user can create a post in all forums that they are subscribed to.
Inputs	The user must include a title to their post in the title field and type their post in the text field. Once the user is finished, they click the button “Add post.”
Processing	The system shall observe the fields and wait for the user to click on “Add post.” Once the user has clicked it, the post will be saved to the Firebase database and the RecyclerView will read the data from the database and display it on the screen to the user.
Outputs	The post, along with its title, will be added to the forum and all subscribed members of that forum would be able to view the user’s post.
Error Handling	If the user’s post is not added after the “Add post” button is clicked, an error message will be presented to the user that the input field is empty if they have not typed anything or an error message will appear to user that the button does not work.
Dependencies	Forward Dependencies: N/A Backward Dependencies: FR3.2.2, FR3.2.3
Priority	4

3.2.5 FR5: Delete Post	
Introduction	The user can delete their own posts.
Inputs	To delete the post, the user would need to swipe on the Post and click the Delete button.
Processing	Once the user has clicked it, the post will be removed from the Firebase database and the RecyclerView.
Outputs	The post, along with its title, and any other comments connected to the user's post will be deleted in a cascading manner.
Error Handling	If the user's post is not deleted after the "Delete post" button is clicked, an error message will appear to the user that the button does not work.
Dependencies	Forward Dependencies: N/A Backward Dependencies: FR3.2.2, FR3.2.3
Priority	4

3.2.6 FR6: Add Comment to Subscribed Forum	
Introduction	The user can create comments on any post in a forum that they are subscribed to.
Inputs	The user must type their comment in the text field. Once the user is finished, they click the button “Add comment”.
Processing	The system shall observe the fields and wait for the user to click on “Add comment.” Once the user has clicked it, the comment will be saved to the Firebase database and the RecyclerView will display the data from the database on the screen to the user.
Outputs	The comment will be added to the related post on the forum and all subscribed members of that forum will be able to view the user’s comment.
Error Handling	If the user’s comment is not added after the “Add comment” button is clicked, an error message will be presented to the user that the text field is empty if they did not type anything or that the button does not work if there is text.
Dependencies	Forward Dependencies: N/A Backward Dependencies: FR3.2.2, FR3.2.3, FR3.2.4
Priority	6

3.2.7 FR7: Delete Comment	
Introduction	The user can delete their own comments any post.
Inputs	The user would need swipe on the comment and hit the delete button.
Processing	Once the user has deleted it, the comment will be removed from the Firebase database and the RecyclerView.
Outputs	The comment will be deleted from the related post and no longer be seen on the forum.
Error Handling	If the user's comment is not deleted after the delete button is clicked, then the user will see an error message that says the button does not work.
Dependencies	Forward Dependencies: N/A Backward Dependencies: FR3.2.2, FR3.2.3, FR3.2.4
Priority	6

3.2.8 FR8: Unsubscribe from Forum	
Introduction	The user can unsubscribe from a forum that they are currently subscribed to. This means that the user will no longer view activity on that forum or be able to participate in it by posting and commenting.
Inputs	The user will click on the "-" sign next to the class forum.
Processing	The change shall be caught by a listener on the view model that will trigger the Firebase to update that user's list of subscribed classes by removing the class they initially subscribed from. By doing so, changes to the database will be reflected to the user from the view model and all the activity pertaining to the now unsubscribed forum will be gone.
Outputs	After the user clicks "-" next to the class forum they would like to unsubscribe from, posts and comments on that forum page will no longer be seen on the user's main homepage screen.
Error Handling	If the user is still subscribed to the class forum after they click "unsubscribe," an error message will appear to the user saying that the button does not work.
Dependencies	Forward Dependencies: N/A Backward Dependencies: FR3.2.2, FR3.2.3
Priority	8

3.2.9 FR9: Search for Forum by Subject and Class Title	
Introduction	The user can search for the class forum they would like to subscribe to by selecting the subject first and then the class title.
Inputs	From the list of subjects offered at WSU, the user must choose the subject that they are interested in. The list of class forums pertaining to that subject will appear and the user must choose the specific class forum they would like to join.
Processing	The system shall organize the forums by the subject and display it on the screen to the user in a structured way, so it is easy for the user to choose a forum to join.
Outputs	The user chooses the class forum they would like to subscribe.
Error Handling	If the user spells the name of what they are searching for incorrectly or search for a forum that does not exists, we will handle this error by display a message stating that the forum does not exist and ask the user to try again.
Dependencies	Forward Dependencies: FR3.2.3, FR3.2.4, FR3.2.5 Backward Dependencies: FR3.2.2
Priority	7

3.2.10 FR10: Forgot/reset password: Extended Login Feature	
Introduction	This is an extension of the login feature that was created out of a use case model. The user will be able to reset their password.
Inputs	The user must enter their email address.
Processing	The system shall first check to see that the email address is listed in the database. Then, the Firebase Authentication will send an email to the user. The user will click on the link and will be instructed to input their new password. Once the user has completed that task and hit reset password, the new password will be updated in the database for that user. The user will be redirected to the Login screen where they will login with their email and new password.
Outputs	The user will receive an email with a link that will direct them to a text field where they can enter their new password.
Error Handling	If the user selects the reset password button and does not receive an email sent to their email address, then an error message will appear to the user that they have entered the wrong email address.
Dependencies	Forward Dependencies: N/A Backward Dependencies: 3.2.2
Priority	3

3.2.11 FR11: Edit Username	
Introduction	The user will be able to edit their username.
Inputs	The user must enter new username.
Processing	The username will be changed in the database in all places that it is saved. This includes the user's data as well as all posts and comments.
Outputs	The user's username will be changed on their profile page as well as everywhere their username can be viewed.
Error Handling	N/A
Dependencies	Forward Dependencies: N/A Backward Dependencies: 3.3.2
Priority	3

3.2.11 FR12: Edit Profile Image	
Introduction	The user will be able to edit their Profile.
Inputs	The user must enter new username.
Processing	The username will be changed in the database and saved to Firebase storage.
Outputs	The user's new profile image will be visible on their profile page.
Error Handling	N/A
Dependencies	Forward Dependencies: N/A Backward Dependencies: 3.3.2
Priority	3

3.2.13 FR13: Edit User Bio	
Introduction	The user will be able to edit their bio.
Inputs	The user must enter new bio.
Processing	The user's bio will be changed in the database.
Outputs	The user's bio will be changed on their profile page.
Error Handling	N/A
Dependencies	Forward Dependencies: N/A Backward Dependencies: 3.2.2
Priority	3

3.2.14 FR14: Messaging	
Introduction	The user can message any user that is using the application.
Inputs	The user must enter in a message and click the send button.
Processing	The message will be added to Firebase Database and added to RecyclerView data to display in the chatlog.
Outputs	The RecyclerView chatlog is updated with the new message.
Error Handling	If the user tries to send an empty message, they will not be able to and if there is no internet, then the messages will cache and send when internet is reconnected.
Dependencies	Forward Dependencies: N/A Backward Dependencies: FR3.2.2
Priority	1

3.3 Non-Functional Requirements

In this section, architecturally significant requirements of the product will be explained in a way that can be measured in quantifiable terms. The product will be evaluated through testing and verification by the following: performance, reliability, availability, security, maintainability, and portability.

3.3.1 Performance

The time it takes for the app to open once a user has clicked on it should be 200 milliseconds.

After a user registers their information correctly and hits register button, it should take anywhere between 200 to 500 milliseconds to redirect to the login page. Once the user has typed in their email and password and hits login button, it should take anywhere between 500 milliseconds to 1 second to direct user to main forum page.

The time it takes to connect the user to the class forum they would like to subscribe to after hitting the subscribe button should be anywhere between 500 milliseconds to a second. The performance time for unsubscribing from a forum should also be the same as subscribing to a forum.

The final performance requirement deals with post creation/deletion and comment creation/deletion. The performance time for all four functionalities should be anywhere between 200 to 500 milliseconds.

3.3.2 Reliability

The reliability of the system reflects on the communication between our Firebase Database backend and our UI front-end. The software product will return error messages to the user 99% of the time to alert them of any changes that resulted from user input or from software development updates.

3.3.3 Availability

The availability for the software product will be operating at 99% of the time annually. Once the product features have been built and tested, they will be pushed into the master version and very little support will be required afterwards. The 1% will be used to fix minor bugs found in the system.

3.3.4 Security

Security vulnerabilities is not a main concern of the client. The stretch goal for this mobile app is to use the Wayne State API search directory for people in order to prevent people not affiliated

with the university from gaining access to the app.

3.3.5 Maintainability

The maintainability of our app revolves around the modularity this team of software developers have employed. Modularity is the principle of creating independent modules and finding a way to combine them into the system of an app. Dagger dependency injection and the MVVM design architecture are the two main ways this app maintains its modularity. The app is designed with four main layers: a data access layer, a model, a view model, and a view. The purpose of the data access layer is to hold all our network calls. The model layer contains our data model objects such as posts, comments, reports, and classes. The view model is a staging area for the view that holds logic and the view involves code directly related to the UI. This separation of concerns is what makes the app maintainable and easy to test.

The dagger dependency injection proves itself to be useful in meeting the standards of having a modular app. The data access layer is injected into the repository, the repository is injected into the view model, and the view model is instantiated in the view using a generic view model factory. The advantage of dependency injection is that it makes the code more decoupled and the layers don't rely on one another as heavily as it would if the data access layer was directly called as an object in the repository layer. Dagger dependency injection also promotes code elasticity by allowing the view models to be multilinked through a module to one generic factory that uses a map of view model keys to initialize the factory for each view model. This is particularly helpful for future groups working on this app because they will not need to create a view model factory file each time a view model is needed to implement a new feature or activity.

3.3.6 Portability

This app can be portable across all Android devices that are running on an Android operating system. The software product can also be accessed completely independent from any hardware component constraints by using an Android device emulator found on Android Studio.

3.4 Design Constraints

The client has asked the design team to use the MVVM architectural design pattern within our application. This will affect dataflow design of the application as the data must flow through the UI, go straight to the view model, into the repository and then *finally* reach our database. The client also asked for all dependencies to be injected using dagger. This means that dependencies will be passed from the outside using the dagger dependency injection framework. Their reasoning for these constraints is that goal is managing a production level project and these constraints are common in production level programing. The only hardware limitation relevant to

this project is the fact that the client specified that the product should be an android application only.

3.5 Logical Database Requirements

The logical database section of the SRS document will talk about what type of database will be used for the software product and explain the functionalities as well as the tools that will be used. The database design for the system will be discussed along with the data structure chosen by the developers. The last part of the logical database requirements will talk about what steps are taken to ensure data integrity.

3.5.1 Database Selection

For the University Collaboration App, we have decided to use a Google Firebase Realtime Database as our backend database. A Firebase Realtime database is a NoSQL database that stores data in the form of JSON objects. A JSON Object can store Strings, Longs, Doubles, Booleans, Maps, and Lists of any of the types from the ones listed above into the database allowing for a flexible set of data types and values. A NoSQL database uses a tree structure to store the JSON objects in different levels, this database also has no relations between objects unlike a SQL database that have a set table structure and keys that identify values from other tables. The non-relational format structures the data in a way that fits the needs of the mobile application being developed.

3.5.2 Database Design

A major problem with a lot of non-relational database designs is nesting of related objects, since the designer cannot leverage keys in an object to point to a related value or row of a separate object, a common practice is to nest object related to each other within a parent object. The major problem with nesting objects is that as you continue to add objects and data to the tree it takes much longer to retrieve data causing a user to have to wait for a longer time to get the data relevant to them and causing frustration. To remedy this the data structure needs to be set up in a way that allows for easy data access while not overloading the database with multiple copies of the same dataset. For our database we have decided to design the database by nesting the post objects into classes so that we can keep relevant post together and comments will be nested inside the post object. There will also be a copy of the post in the user section of the database, so that we can easily find post pertaining to a specific user. The classes will be broken into subjects and stored as a list of class names so that a list of relevant class names can be found without having to search through a list of all the classes available.

3.5.3 Data Integrity

The Google Realtime database uses data synchronization to keep the dataset accurate as things are changed or added, that allows users data to stay relevant to changes without having to continually make a HTTP request to check the status of the database. The application is notified of any changes to the database through functions designed by the team to ensure when something does change it takes the correct actions to update the UI with the new information. Through the Firebase SDK the data also persist offline on the device disk that way when the device gets a connection it can use its local reference to decide whether a change has been made and if the device needs to change any data. Since a Realtime database is a NoSQL database the objects stored are user defined and do not have the restrictions of the tables of a SQL database, instead the rules are designed on the .rules file on the Firebase console. The ruleset is defined by the users and without it anyone in the system can read or write to anything in the dataset, but the rules can be written to keep the reads and writes access to users specific to that data. For instance, a common rule is that all users can read any users profile but only the user who created the data by storing the user ID in that data object and only letting a user with the same user ID to change anything. The Realtime database also includes support for IOS applications as well as web applications so in the future if we wanted to expand to IOS devices or a web application there would be no need to migrate the database to a new database to increase the number of devices used with this database.

3.6 Other Requirements

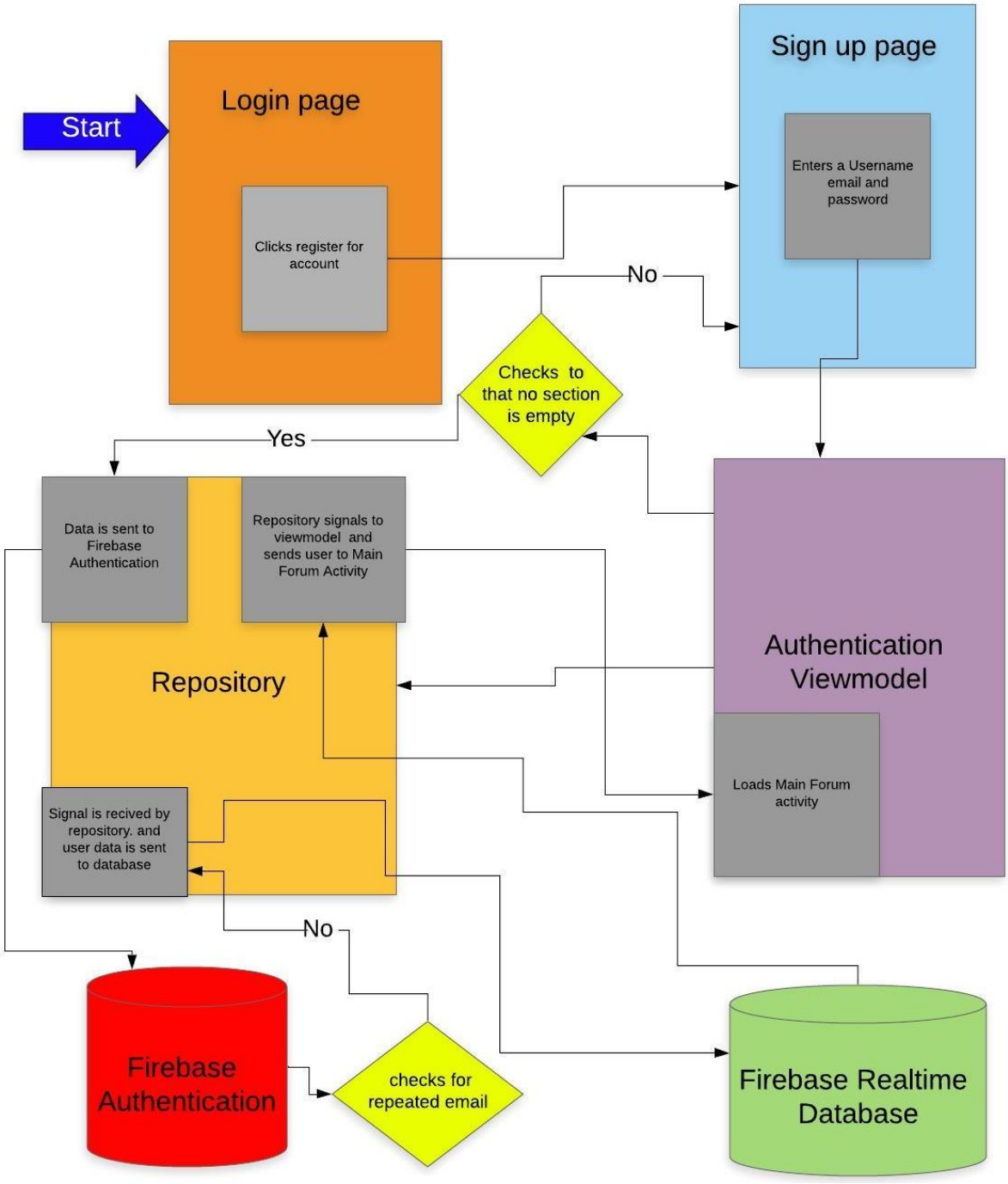
Another requirement not mentioned in the previous sections is that the user is able to have access to multiple forums; they are not just limited to one. Also, the user is not allowed to make a generic post outside of a class forum.

4. ANALYSIS MODELS

The Analysis Models section will consist of 4 data flow diagrams that will explain each of the complex requirements of the software product: user registration, user login, loading posts to the class forum, and creating new posts.

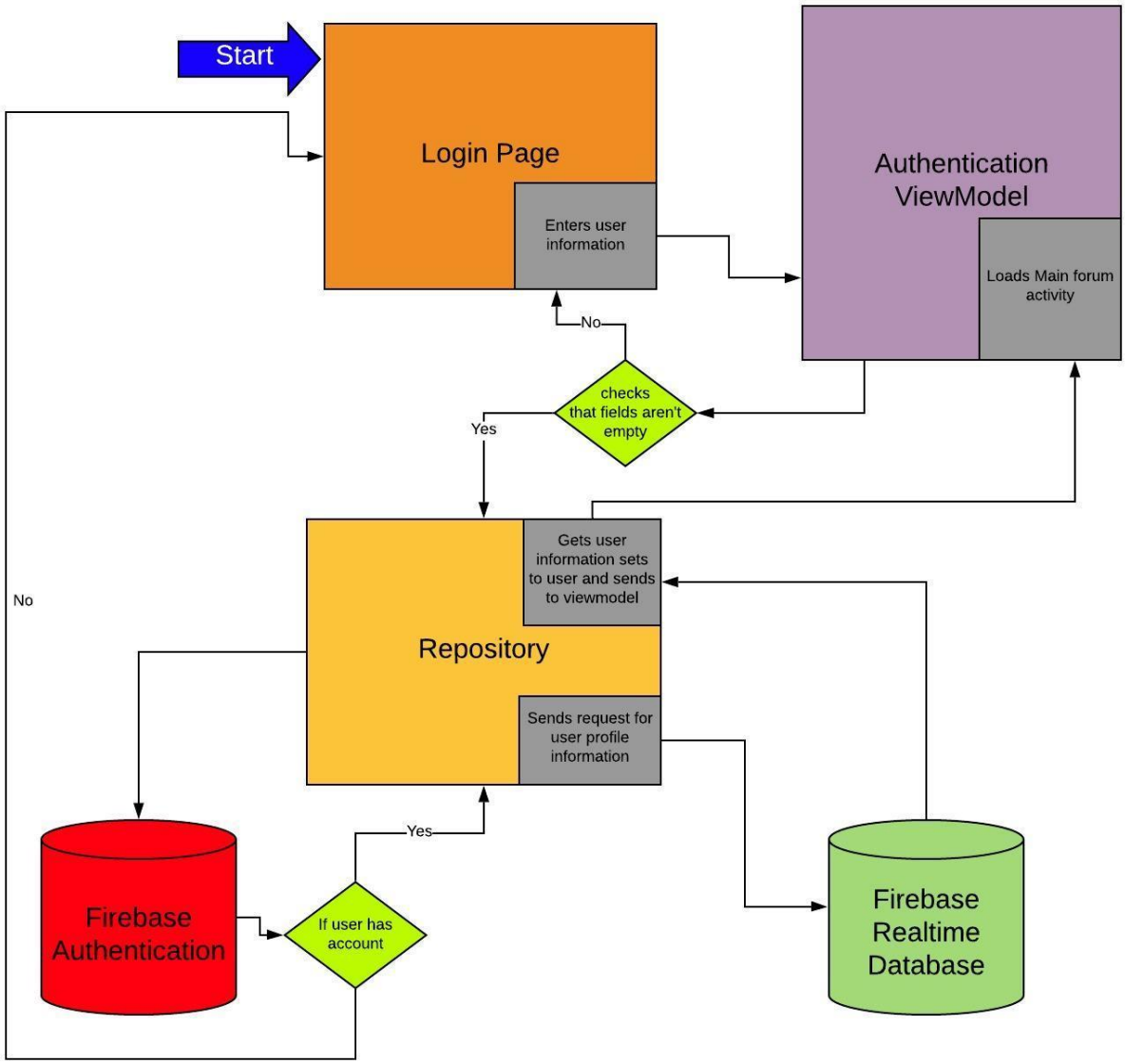
4.1 Registration Data Flow Diagram (DFD)

The DFD shows how the system processes the user registration. The registration process is shown interacting with the registration activity, the authentication view model, the repository class, the Firebase Authentication, and the Firebase Realtime Database.



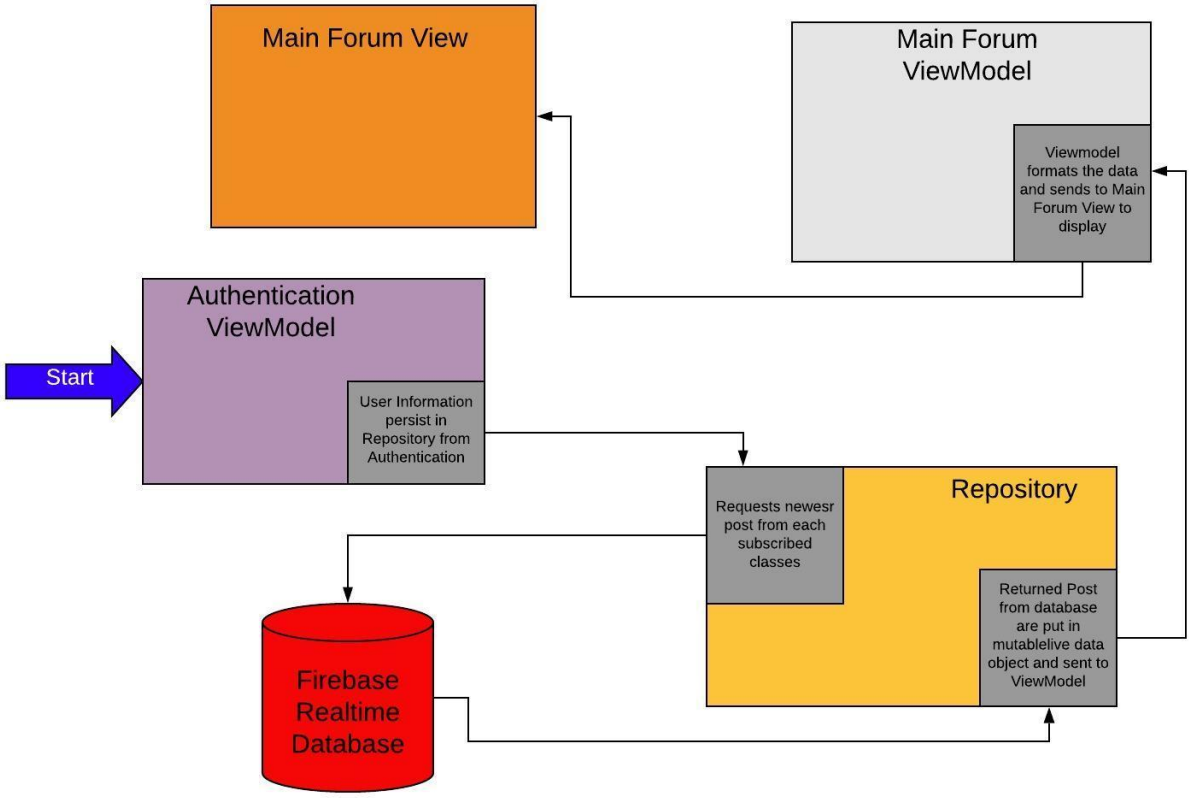
4.2 Log in Data Flow Diagram (DFD)

The DFD shows how the system processes the log in. The login process is shown interacting with the log in activity, the authentication view model, the repository class, the Firebase Authentication, and the Firebase Realtime Database.



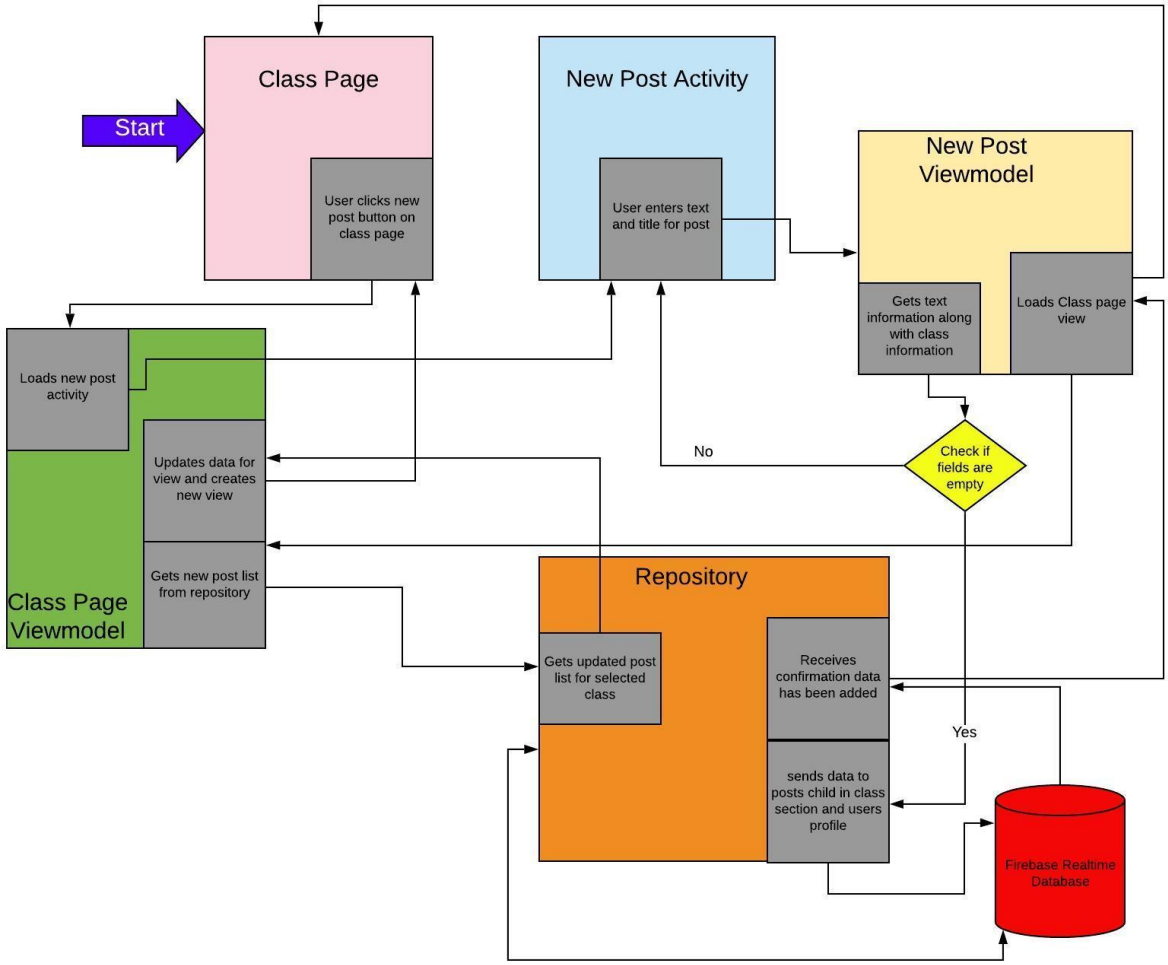
4.3 Load Posts Flow Diagram (DFD)

The DFD shows how the system processes loading posts for the user to see on the main forum. The load posts process is shown interacting with the new posts activity, the view model, the repository class, and the Firebase Realtime Database.



4.4 New Post Data Flow Diagram (DFD)

The DFD shows how the system processes the user creating a new post. The load posts process is shown interacting with the new posts activity, the view model, the repository class, and the Firebase Realtime Database.



A. APPENDICES

A.1 Meeting Minute Documents with Client

Attendees: Palak, Tyler, Me, Son GTA, Jahnu, Thomas

Core goal: study forum with classes
Stretch: assign flash cards for a certain class
2nd stretch: chat features if we have time

General, daily discussions

Assign flash cards to certain class

Break features into user stories
Team lead will delegate items from backlog. We complete test and return code to clients for approval

1 week sprints for work.
Standups: yes
Look at tutorial to get AWS set up. We will use AWS Amplify

Trello – task managing app. Jahnu made the trello. Backlog is in trello.
Start up Jenkins server and pipeline it with GitHub.

Book room with a tv next time.

Spike(investigation). Research technologies. Use the GitHub commercial account that Thomas sets up

Spike: research AWS.
Friday Noon: tech lead sends list to GTA of list of tasks for students.

Weekly report next week Saturday.
Icebox to backlog then backlog to members
MVVM. Set up Spike

Set up first prototype with SQLite then use network calls with AWS Amplify. MVVM plans as soon as first prototype work starts MVP(model view presenter)

Have login set
have functioning forum for first prototype goals
- -have list classes we can sort through
- -subscribing to that class
- -creating classes for the classes
- -post something to the class
  - -title
  - -body

Figure A.1.1 Meeting Minutes on 1/22/2020

Attendees: Hala, Palak, Tyler, Adeel, Tom, Tom's guest
Activity layer= binding to UI and context

GO one layer deeper in code implementation
Trigger network calls in the view model
When one person makes network call, everyone subscribed to that call gets updated
Cleaning up hanging threads. If someone is in a subreddit and they don't go back, kill that subscription to free memory
Any time you subscribe you create a disposable
If you need to unsubscribe, get rid of disposable use thing.clear()

For onpause you clear lifecycle subscription

Nothing but subscribers in a view model
Activity binding model in a ui layer
Fragments get view models… no lifecycle stuff in fragments.. but it happens

No event bus just write reusable functions that make intents
Don't comment your code.. make it self explanatory! Commenting in code is a bad practice!
Call function that makes a database query

1 point of entry that is used for testing. Can't call private functions in testing have one public point of entry. Test functions as you write them. Flow and individual function tests. 1 access point in your class. Adjust your flow

@Before before every single test mock these things

View model logic repository lo
1 repo per network

Eventbus= giant class that has context
Dagger map for this type of app
SharedPreferences = give key returns value persists thru if they log off
Need to Know type of shared preference

Figure A.1.2 Meeting Minutes on 1/29/2020

Attendees: Hala, Palak, Tyler, Adeel, Thomas, Jahnu

Added branch security rule on GitHub

Gilded Rose workshop: walk into hard-coded and fix it and refactor it

Next Time for the client meeting: One Unit Test. If PR gets approval from user, push into Master and have Jenkins test working.

Optimize database performance. Avoid duplicate calls in our app

Switch to cloud firestore

Look into Firebase functions.

Working state of master with small incremental changes.

Code Reviewers can look at pull request and confirm merge request

Git push origin head to do PR

Set flags in testing to flow from public methods to private

Finishing touches for prototype 3

Comment to a comment will be difficult
Only focusing on post to a comment

Putting comments in a post.
Test performance of time it takes to search for a post

Coroutines and testing next time with MockK

Adding function from xml to communicate with the view

Figure A.1.3 Meeting Minutes on 2/5/2020

A.2 Conceptual Documents on Project

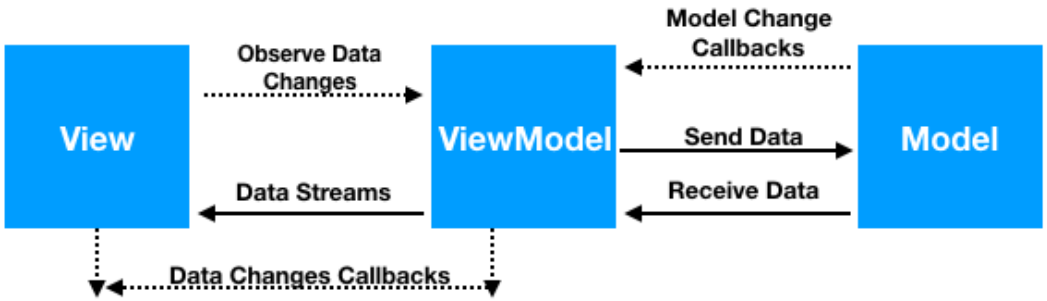


Figure A.2.1 Processing involved in MVVM architecture

Figure A.2.2 Accurate representation of database design in the form of a JSON object