CS 30700
Design Document

Team 16:

- Anirudh Seela
- Arnav Gupta
- Darsh Dalal
- Rohith Rajashekarbabu
- Shreyas Kharbanda
- Tejas Prakash

INDEX

Purpose:

Every year thousands of students leave the comfort of their homes behind and enter a complex educational ecosystem to chase their futures. Bombarded with a new environment, responsibilities, and socio-cultural landscape, it takes many students hours of clueless wandering and guessing that most often lead to misguided decisions and unwanted outcomes. We want to reduce this learning curve by creating a one-stop-shop for all things Purdue. Through a peer-based network, new students can learn about and understand the campus through the experiences of their peers.

Functional Requirements:

Account Management:

As a user,

1. I would like to register for an account.
2. I would like to login to my account.
3. I would like to continue being logged into my account unless I explicitly log out from the application.
4. I would like to delete my account and data.
5. I would like to change the password for my account in case I forget it or my account gets breached.
6. I would like to create a custom username.
7. I would like to edit my custom username.
8. I would like to have a way to recover my password through email.
9. As a user, I would like to have a variety of themes for the interface (light/dark mode).
10. (If time allows) I would like to be able to change my primary email for login and email purposes.

Application Navigation:

As a user,

1. I would like to view an onboarding screen that allows me to understand the features of the application.
2. I would like to easily navigate to all the different product screens.
3. I would like to download the application on either an iOS or an Android device without loss of functionality and user experience.
4. I would like access to a help/FAQ page if I have trouble with the app.

5. I would like to have access to a search engine functionality so that I am easily able to find a class.
6. I would like to choose to see more credible reviews first.
7. I would like to choose to see the recent reviews first.
8. I would like to choose to see the oldest reviews first.
9. I would like to choose to see the recent posts of first..
10. I would like to choose to see the oldest posts first.
11. I would like to search for reviews by the relevancy of my query.
12. I would like to search for posts by the relevancy of my query.
13. I would like to see community approved reviews at the top of the review feed for easier access.
14. (If time allows) I would like to have access to a search engine functionality so that I am easily able to find on-campus dining facilities.
15. (If time allows) I would like to have access to a search engine functionality so that I am easily able to find residence halls.
16. (If time allows) I would like to have access to a search engine functionality so that I am easily able to find a professor.

Credibility and Voting System:

As a user,

1. I would like to have a fair metric to assess the credibility of other users.
2. I would like to upvote or downvote a post or review.
3. I would like to see the credibility score of another user.
4. I would like to report users if they violate any of the platform policies.
5. I should be able to explain my position and have my credibility score taken into consideration before any decision is taken, if I am reported .
6. (If time allows) Sentiment analysis of reviews for a more comprehensive credibility score calculation.

Posts and Reviews System:

As a user,

1. I would like to read others' reviews for classes and professors.
2. I would like to read others' reviews for on-campus dining facilities and residence halls.
3. I would like to write my reviews for classes and professors.
4. I would like to write my reviews for on-campus dining facilities and residence halls.
5. I would like to edit my reviews for classes and professors.

6. I would like to edit my reviews for on-campus dining facilities and residence halls.
7. I would like to delete my reviews for classes and professors.
8. I would like to delete my reviews for on-campus dining facilities and residence halls.
9. I would like to write my posts for classes and professors.
10. I would like to write my posts for on-campus dining facilities and residence halls.
11. I would like to read others' posts for classes and professors.
12. I would like to read others' posts for on-campus dining facilities and residence halls.
13. I would like to edit my posts for classes and professors.
14. I would like to edit my posts for on-campus dining facilities and residence halls.
15. I would like to delete my posts for classes and professors.
16. I would like to delete my posts for on-campus dining facilities and residence halls.

Recommendations:

As a user,

1. I would like to see recommended dining court options.
2. I would like to see recommended roommate options.
3. I would like to create a profile and showcase my information.
4. I would like to see other's profiles in order to select my roommate.

Administrator Privileges:

As a user,

1. I would like to remove posts.
2. I would like to remove reviews.
3. I would like to uphold terms and conditions.
4. I would like to track engagement with the application.
5. I would like to have access to an emergency shutdown option.
6. I would like the ability to push updates to the application.
7. I would like to collect and review feedback from users.
8. I would like to authenticate users through proof of enrollment at Purdue University.
9. I would like the ability to ban accounts along with giving them reasoning through email.

Messaging:

As a user,

1. (If time allows) I would like to directly message other users.

2. (If time allows) I would like to have a privacy setting for message requests. Either can't be messaged, message after an accepted request, or any message.
3. (If time allows) I would like to block other users from messaging me.
4. (If time allows) I would like to receive notifications when I am messaged.
5. (If time allows) I would like to receive notifications when someone has responded to my post.

Non-Functional Requirements:

Architecture:

- We are going to follow the client-server architecture model.
- Our highest priority is to build a modular, highly integrable codebase for rapid iteration.
- High emphasis on separation of concerns and an easily navigable codebase for better developer productivity.
- The server will be built modeling RESTful (Representational State Transfer) Application Programming Interface, written in JavaScript using the popular Node.js framework.
- Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the Chrome V8 engine and executes JavaScript code outside a web browser.
- The Node Packet Manager (NPM), Node.js allows downloading third-party modules that enable faster development and re-usability for commonly required structural building blocks.
- For data persistence, we will be using the Firestore NoSQL database by Google Firebase. Our system will require heavy data flow to and from the database, we want to have a database that is quick and robust enough to handle the data storage and retrieval.
- As the prototyping and sprints will lead to new discoveries and understanding of what data we require and the order in which we need to fetch it, having a non-tabular format that can be easily modified will enable the developers to be productive and iterate quickly without any time-consuming overhead of sanitizing the codebase and restructuring the tables that generally comes with SQL databases.
- The client will be built using React Native. React Native is an open-source mobile application framework created by Facebook, Inc. It uses the React way of writing re-usable, stateful, and lightweight components that can be used to create a user interface. It uses a bridge, a way to compile applications written with JavaScript into native mobile applications. Thus, it allows the developer to maintain one codebase while deploying the application on all major platforms.

Performance and Scalability:

- We want to focus on performance and scalability throughout our development endeavor to create a seamless user experience.
- It is crucial for the application to be small in size (<50 MB) for easy download on the user's phone and cache only if direly required so that our application does not hoard space on the user's phone.
- By following the client-server model, we want to maximize performance by developing a thin client that delegates the computational work to the server to minimize crashes and have a secure place for business logic.
- Through using a non-blocking, asynchronous architecture on the server-side along with the lightweight architecture on the client-side, we can easily scale the application to accommodate as many users as required.
- Moreover, our database is an on-demand, cloud-based database that can be scaled up or down based on user traction. Such a structure gives us a great headstart when it comes to scalability and performance.
- We plan to scale this application for other colleges in the future.

Security:

- Security and privacy are important and of high-priority to foster an engaged and safeguarded community.
- We want to maximize security on our platform by minimizing sensitive data. Leveraging this approach, we delegate as much sensitive data storage to well-established, secure services.
- The main sensitive data that will most commonly flow through our system is the username and password.
- We delegate such data security by using Firebase by Google Cloud Platforms. Firebase takes charge of authenticating the user, generating a token, and checking for persistence when accessing private routes, making it a secure, efficient, and scalable option for authenticating users.

Usability:

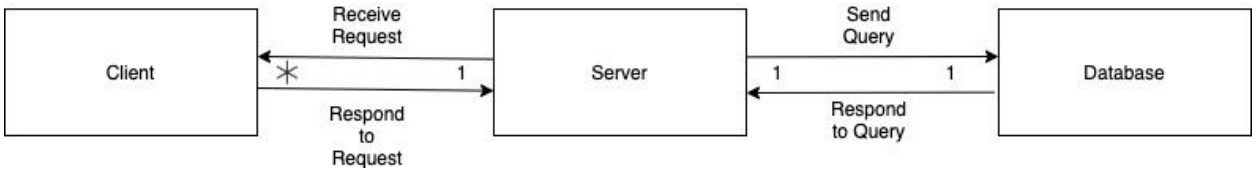
- It is crucial for a mobile application to facilitate information aggregation for students to be easily navigable and aesthetically pleasing.
- For user retention and engagement, the application needs to be easy to understand, intuitive, and devoid of jargon or any other specialized structural system.
- Moreover, we want our application to run on all different phones without degradation in user experience or crashes.

Hosting/Deployment:

- The React Native Application will be deployed using Expo to the App Store and the PlayStore.
- The Node.js backend will be deployed on Heroku or Digital Ocean based on economic and performance feasibility.
- The Firestore database would be deployed using the Google Cloud Platforms (GCP).

Design Outline

This project will be a mobile application that allows users to access a centralized repository of reviews about on-campus facilities, agency to post their own reviews and receive personalized suggestions for places to eat based on their preferences. This application will use the client-server model in which one server simultaneously handles access from a large number of clients using Node.js framework in JavaScript. The server will accept client requests, access or store data in the database, and feedback to the client(s) accordingly.



1. Client

a. Client provides the user agency to communicate with our system.
b. Client sends asynchronous requests to the server.
c. Client receives and interprets asynchronous responses from the server and renders changes in the virtual document object model that is subsequently displayed on the screen.

2. Server

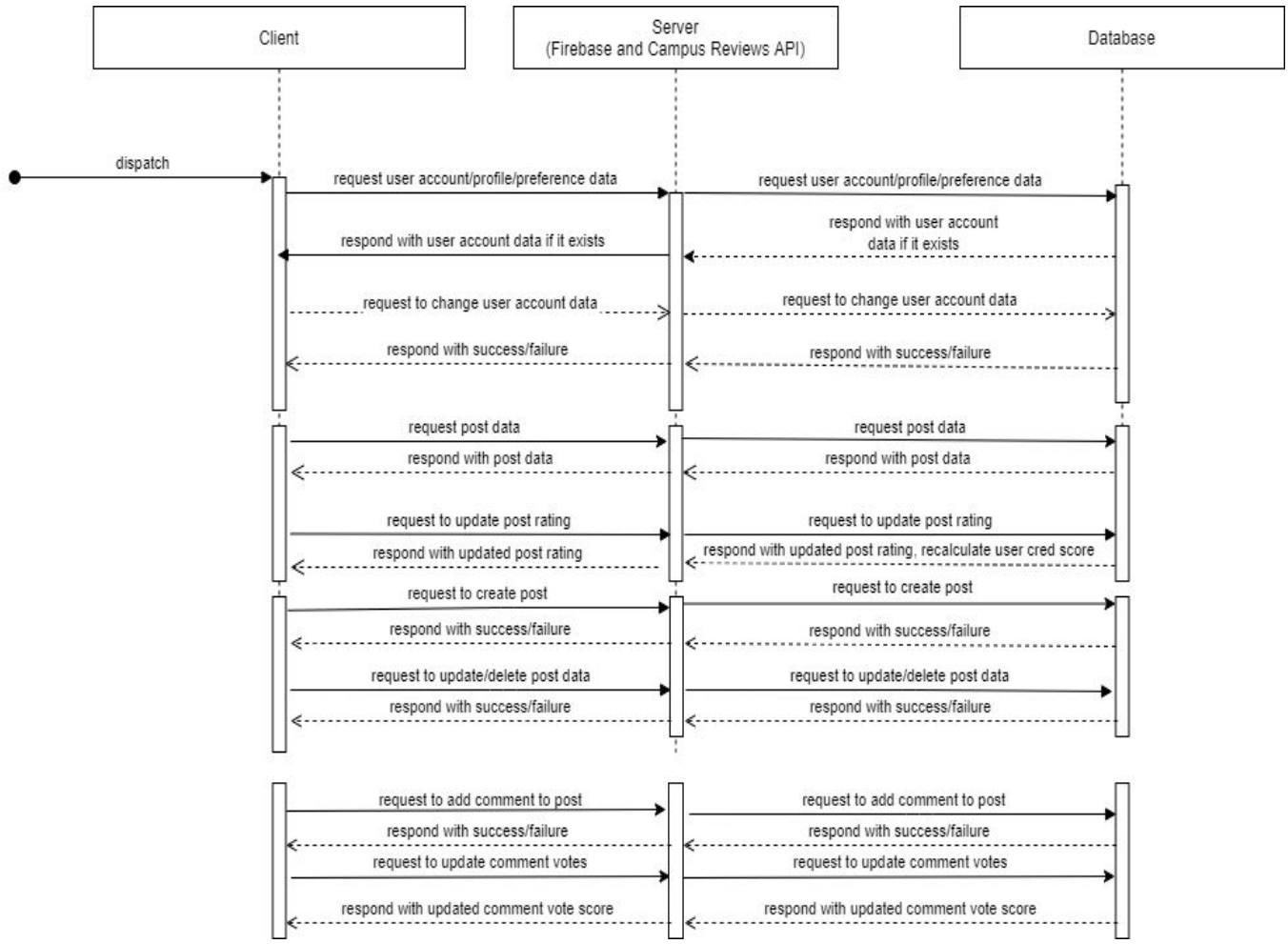
a. Server receives and handles asynchronous requests from the clients.
b. Server validates requests and sends queries to the database to create, read, update or delete data on demand.
c. Server generates appropriate responses and sends them to targeted clients.

3. Database

a. A database allows persistence of data, while allowing a smooth flow throughout the architecture.
b. A non-relational database stores all data used in the application as documents stored under collections.
c. Database responds to queries from the server and sends the extracted data back to the server.

Sequence Of Events Overview

The sequence diagram below shows the typical interaction among clients, server and database. The sequence is initiated by a user starting the mobile application. Upon logging in, the client sends a request to the firebase server, which authenticates the user and retrieves their data stored in the database and returns it to the client. After logging in, the client can further send requests to the server for other actions, like managing the profile data, creating, updating and accessing reviews, and giving votes (upvotes/downvotes) to certain posts. To respond to such requests, the server sends queries to the database to acquire data from the database.



Functional Issues:

1. What data do we need to register for an account?

a. Username and password
b. Purdue email and password only
c. Purdue email, username, and password

Choice: Purdue email, username, and password

Justification:
To register for an account, we want to be able to verify that you are an incoming freshmen, current Purdue students, or a Purdue alumni. We can do this quite easily by asking for the new user's unique Purdue email address and sending a verification email to the account. We want a password just for account security and data protection. Additionally, we are also giving the option to make and edit your own username so the user can remain anonymous, this should encourage more discussion and activity on the app.

2. How are we going to allow and promote interaction amongst users?

a. Upvote/Downvote for each review
b. Upvote/Downvote for each review and a comment section for further discussion
c. Upvote/Downvote for each review, a comment section for further discussion, and(if time allows) ability to privately message other users

Choice: Upvote/Downvote for each review, a comment section for further discussion, and a comprehensive credibility rating for each user based on their account history

Justification:
Interaction amongst users is integral for our app to succeed. We plan to promote this by allowing users to upvote/downvote on other reviews, this way they can see how other users feel about the review in hand. A comment section should help build a more interactive platform so people can discuss any issues they had with a post or what they agreed with in it. This will help with answering any further questions that people who view the review may have. We also plan to implement a private messaging option for the users(if time allows), this way they have the option to approach someone to answer their specific questions if a user has any.

3. How are we going to quantify the credibility of a user?

a. Upvote/Downvote for all user posts/reviews
b. Category specific Upvote/Downvote (Dining v.s. Class)

c. Category specific Upvote/Downvote + Sentiment Analysis of post content

Choice: Category specific Upvote/Downvote + Sentiment Analysis of post content

Justification:
Credible reviews is a major part of what separates us from our competition. As this is a peer to peer network, the primary factor in this quantification of credibility should be other user input. However, people may have more credible opinions in some topics while less credible ones in others. To correct this bias we want to add an individual credibility score between clearly defined aspects. In addition, people have different extremities of description in posts and reviews. To correct this, we are going to use sentiment analysis to standardize reviews in relation to each other.

We plan to enforce this by allowing users to upvote/downvote on other reviews, this way they can see how other users feel about the review in hand. A comment section should help build a more interactive platform so people can discuss any issues they had with a post or what they agreed with in it. This will also help with answering any further questions that people who view the review may have.

4. How are we going to ensure privacy for our users?

a. Secure data through firewall, encryption, etc.
b. Avoid storing sensitive data completely

Choice: Avoid storing sensitive data completely.

Justification:
The privacy of our users is pertinent to our product. We plan to guarantee this privacy to our users by not storing any of the data that is important and private to them. Rather than developing an encryption algorithm which may or may not be fool-proof, the choice of not storing data is the simplest way to avoid any data breaches or leaks of any notable data.

5. What level of anonymity should users be allowed?

a. No anonymity. Name and email accessible by all.
b. Complete anonymity. All posts have original posters whose names are anonymous
c. Up to the user. Allowing users to create and change their usernames.

Choice: Up to the user. Allowing users to create and change their usernames.

Justification:
User interaction, especially on a site that promotes the use of honesty, is usually dependent on users feeling safe and secure about how much information they are willing to put out into the open. As such, by allowing the user to choose their level of anonymity, we accomplish this ideal and optimize possible user interaction on the site.

6. What is our response to any malicious activity?

a. Not engage in any monitoring of the app.
b. Ban the account in question.
c. Ban the account in question and give the user the option to appeal the ban.

Choice: Ban the account in question and give the user the option to appeal the ban

Justification:
We want to maintain a safe space for our users and foster constructive feedback. We feel like we can achieve this by allowing users to report derogatory and gross reviews. If a user accumulates enough reports over a certain amount of review then the account in question will get banned. However, we want to give the user the option to appeal the ban by reaching out to us over our support email. We will take the user's reasoning and their credibility scores into consideration and come to the decision to revoke the ban and hand back the account.

Non-Functional Issues:

1. What web hosting service will we use?

a. AWS
b. Heroku
c. Google Cloud Platform
d. Azure

Choice: Heroku

Justification:
In light of the development time and expertise available, we do not want to spend too much energy on fine-tuning and configuring an enterprise cloud system. Rather, as we have all used Heroku before and understand the set-up and deployment procedure, it'll allow us to be more efficient in terms of debugging and reduce our time to market.

2. **What database service will we use?**

a. Firestore NoSQL

b. MongoDB
c. Azure SQL
d. PostgreSQL

Choice: Firestore NoSQL

Justification:
We chose Firestore NoSQL database by Google Firebase because we wanted a quick and robust way to handle the data storage and retrieval without dealing with rigid SQL databases like Azure. Also, Firebase authentication makes it easier for us to build a secure platform without dealing with multiple services, which is why the choice of Firestore is the most efficient for our use.

3. What backend language/framework should we use?

a. Node.js (JavaScript)
b. Django (Python)
c. Laravel (PHP)
d. Spring Boot (Java)

Choice: Node.js

Justification:
The server would be the primary engine for handling multiple requests from multiple users at the same time, thus, we want to focus on a non-blocking and asynchronous architecture that can handle I/O intensive requests. Node.js is a perfect choice for this as it is scalable, easy to use and lies in the expertise of our team. Through using Node.js, we can better optimize our server, and get started quickly.

4. What frontend language/framework should we use?

a. Vue (JavaScript)
b. React-Native (JavaScript)
c. Angular(JavaScript)
d. Ionic (JavaScript)

Choice: React-Native

Justification:
For an optimal user experience, we want to focus on compactness and simplicity, and therefore we want to build a thin client. To achieve this, we need a lightweight, stateful, and reusable

component-based library that can enable simple computations on the client, such as filtering and sorting, without disturbing the server. React-Native is a great choice for this as it acts as an extension of the React library, which is known for being lightweight, stateful, and reusable component-based. Moreover, as our team specializes in React, we will be easily able to port our knowledge to this framework.

5. How are we going to develop the mobile application?

a. Swift (iOS)
b. Kotlin (Android)
c. Ionic (JavaScript)
d. Expo (JavaScript)
e. Vanilla React-Native (CLI)

Choice: Expo

Justification:

Our team has always had the philosophy to have one codebase for cross-platform deployment. Expo is the best equipped framework that will allow us to carry out this philosophy. Moreover, expo comes with a bunch of pre-written modules for notifications, deployment and testing, which make the entire development process seamless and quicker.

6. What service should we use to secure the data of our users?

a. GCP Firebase
b. Auth0
c. Okta
d. Amazon Cognito

Choice: Firebase

Justification:
We chose Firebase to handle data security of our application, as it makes it easy to authenticate the user, generate a token, and secure private routes. As we have already decided to use Firestore, the most efficient choice is to use the same platform to secure our user data.

7. What sort of caching should we use?

a. Redis

b. Google Cloud Platform Caching
c. Async Storage
d. Epic

Choice: Async Storage

Justification:
Our goal is to consume the minimum amount of resources while still providing a smooth user experience. To achieve this, we do not want to add another dependency for caching mechanism. Async Storage comes built in with React-Native and allows us to have full control on the amount of caching we need to use, allowing us to scale it up and down easily. To ensure that there is no surprise bloatware that's getting attached or cached with our mobile application, we would be using Async Storage for our caching mechanisms.

8. How are we going to push notifications?

f. Expo Notifications
g. Google Cloud Platform Notifications
h. Native Notification Service
i. AWS Amplify Notifications.

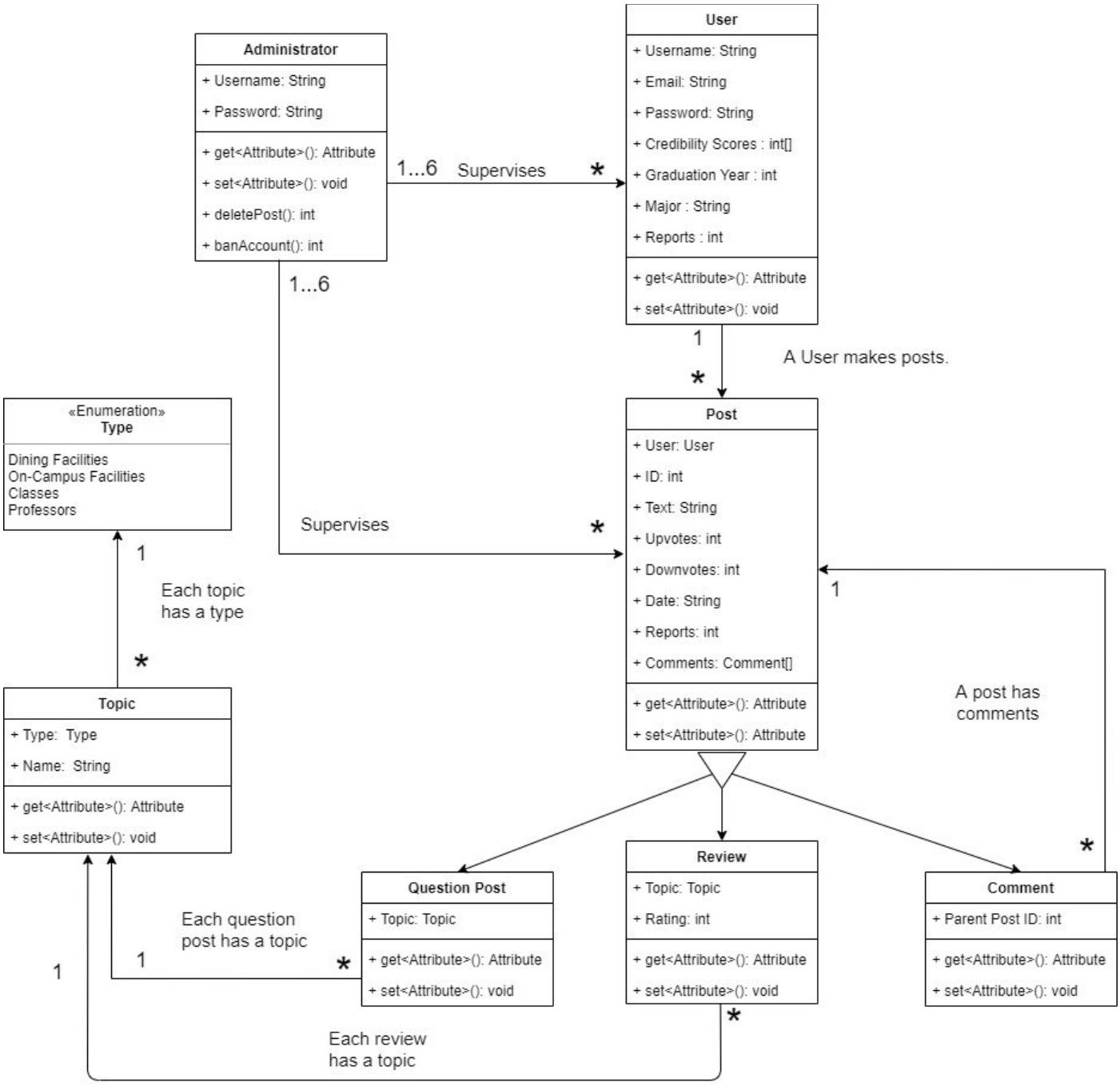
Choice: Expo Notifications

Justification:

Pushing notifications is a key characteristic of a mobile application. As we would be using Expo to develop a cross-platform application, we get access to its push notification module, which makes it very simple and quick to set up a platform for seamless push notifications.

Design Details

Class Diagram



Description of Data Classes and their Interactions

Our Classes are built based upon the unique objects and commonalities in our application. Each unique object has a set of necessary characteristics and methods.

Administrator

- Initially, we plan to have the 6 developers be administrators for the site.
- Administrators will have a unique login along with a unique username and password, however they don't need an email.
- An administrator will have access to functionality that will allow them to remove any other user's posts.
- An administrator will have access to functionality that will allow them to ban any other user's accounts.

User

- User object is created upon creation of account.
- User objects represent an individual profile in our application.
- A user object will contain a unique username, password, and email with a Purdue domain for login purposes.
- Each user will have a credibility score for all 4 topics (Classes, Dining courts, On Campus Facilities, Professors).
- User objects will also contain the graduation year of the user and major, for sake of time relevance and credibility.
- The object will have a running counter of the number of times the user has been reported, for the purpose of disciplinary action.
- The User object will contain various getters and setters to obtain the relevant data.

Post

- Post will be a superclass with the Question Post, Review, and Comment being subclasses of it.
- Each Post will have a unique id to identify it by and it will also store the username of the user who created the post.
- Additionally, a Post will also have text which will make up the body or the content of the post.
- We will also store other details of a post such as its number of upvotes and downvotes, date it was created, the number of reports, and an array of comments that pertain to the post.

Question Post

- A question post will be a subclass of Post.

- A question post will have a Topic, and will be displayed on a specific page accordingly.
- A question post is created when a user has a question regarding a particular Topic.

Review

- A review will be a subclass of Post.
- A review will have a Topic, and will be displayed on a specific page accordingly.
- A Review object will be created when the User would like to post a review about a certain Topic.
- A review will also have a rating field, where a user will rate the Topic they are reviewing out of 10.

Comment

- A comment will be a subclass of Post.
- A comment must have a parent Post, it cannot be a stand alone post.
- A Comment object will be created when the User would like to post a comment regarding any other Post object.

Topic

- A Topic specifies the various pages a question post or review will be posted to.
- A Topic will have a type to help narrow down the page.
- A Topic will offer more specificity to its Type.
- Example: CS307 will be a Topic of the Type Classes.

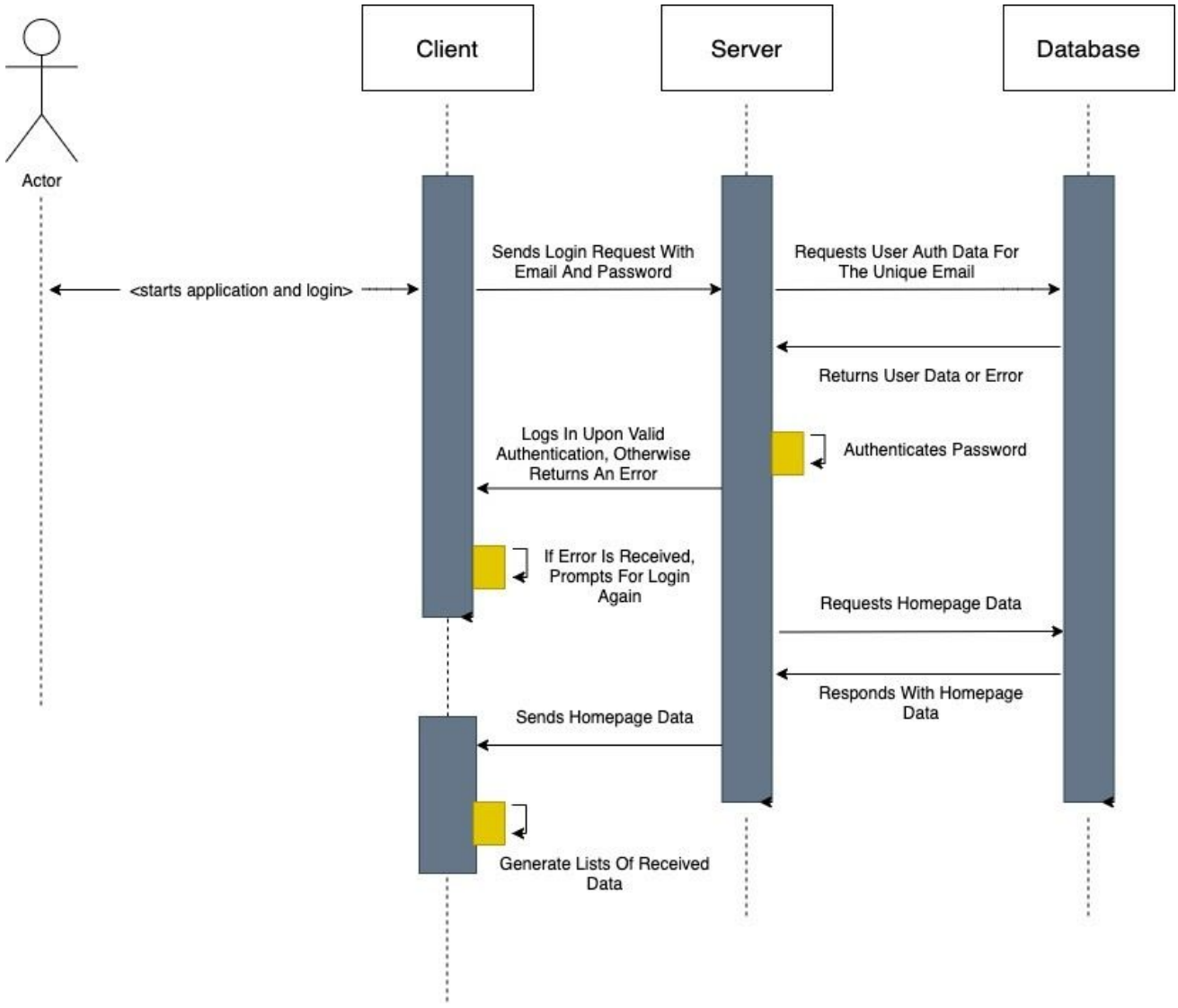
Type

- A Type will be an enum class with 4 possibilities: Dining Facilities, On-Campus Facilities, Classes, and Professors.

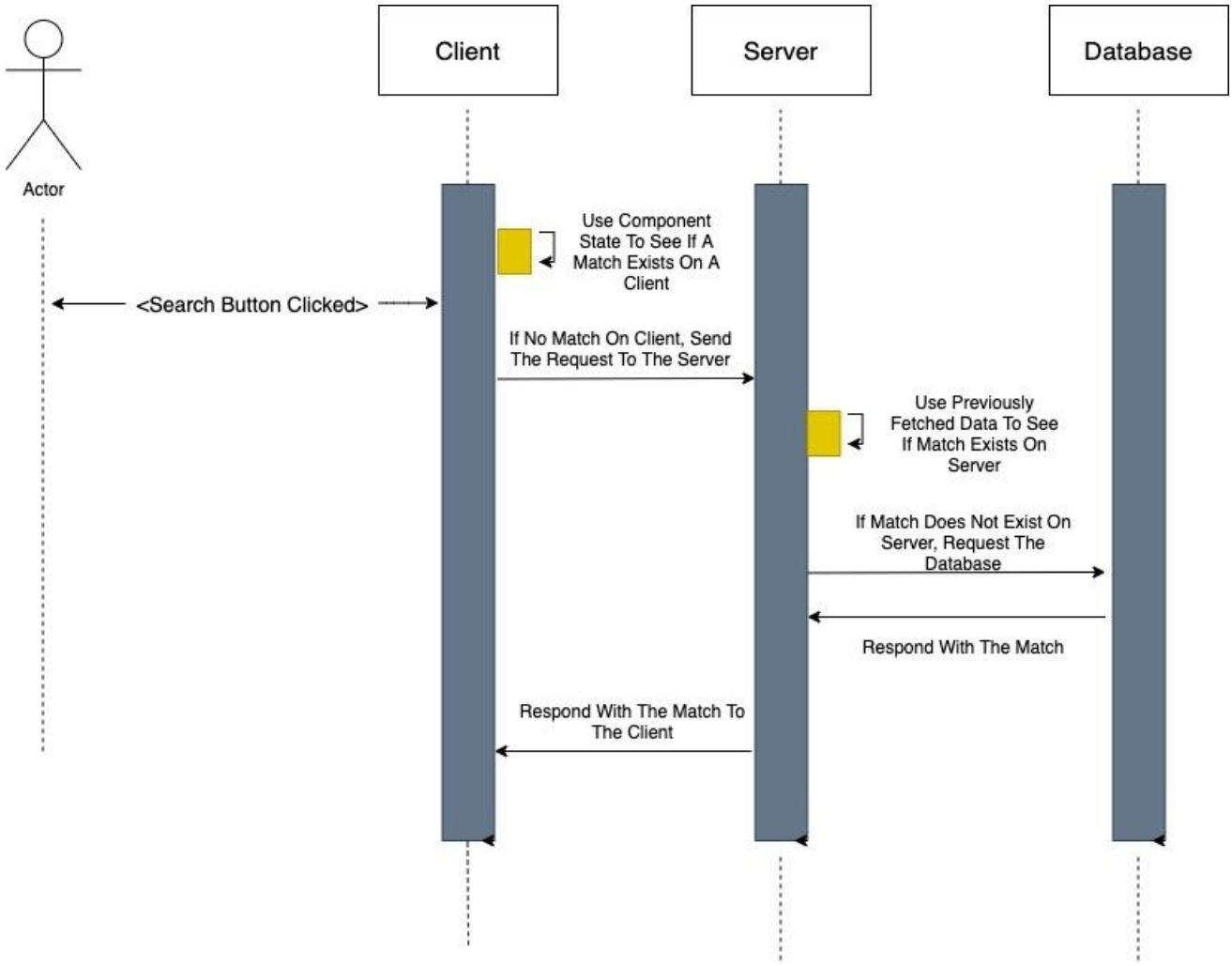
Sequence Diagrams

The following diagrams show the sequence of the necessary functioning events in this application, including user login, creating a post, commenting, upvoting/downvoting a post, and the recommendation engines (roommate matching and dining hall). The sequence shows how information exchanges in a client-server model. When a user performs an action on the frontend user interface, the client will send a request to the server and the server will send a query to the database to request the required information. The server may process or compute the data and send a response back to the client.

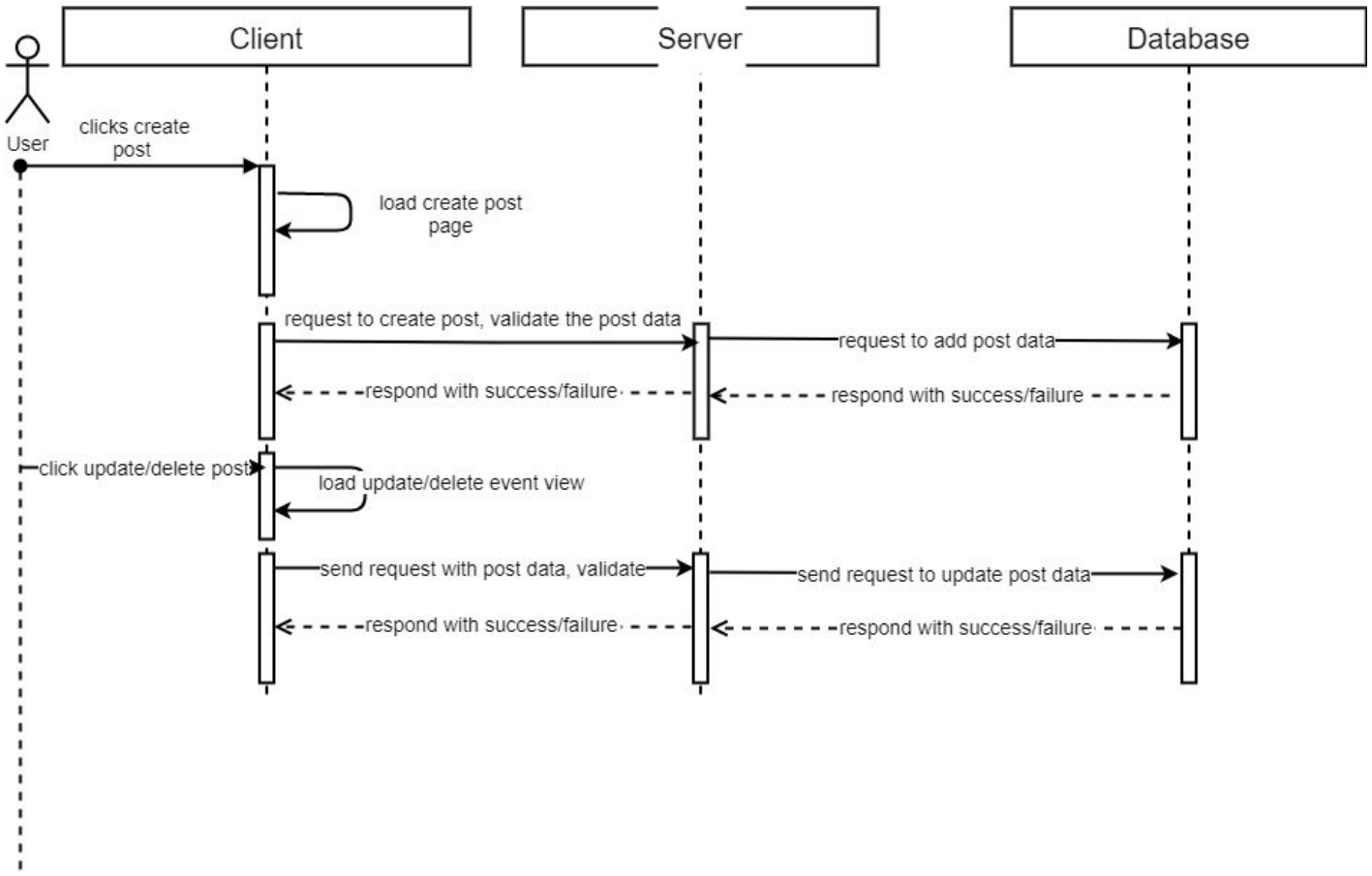
1. Sequence of events when user logs in



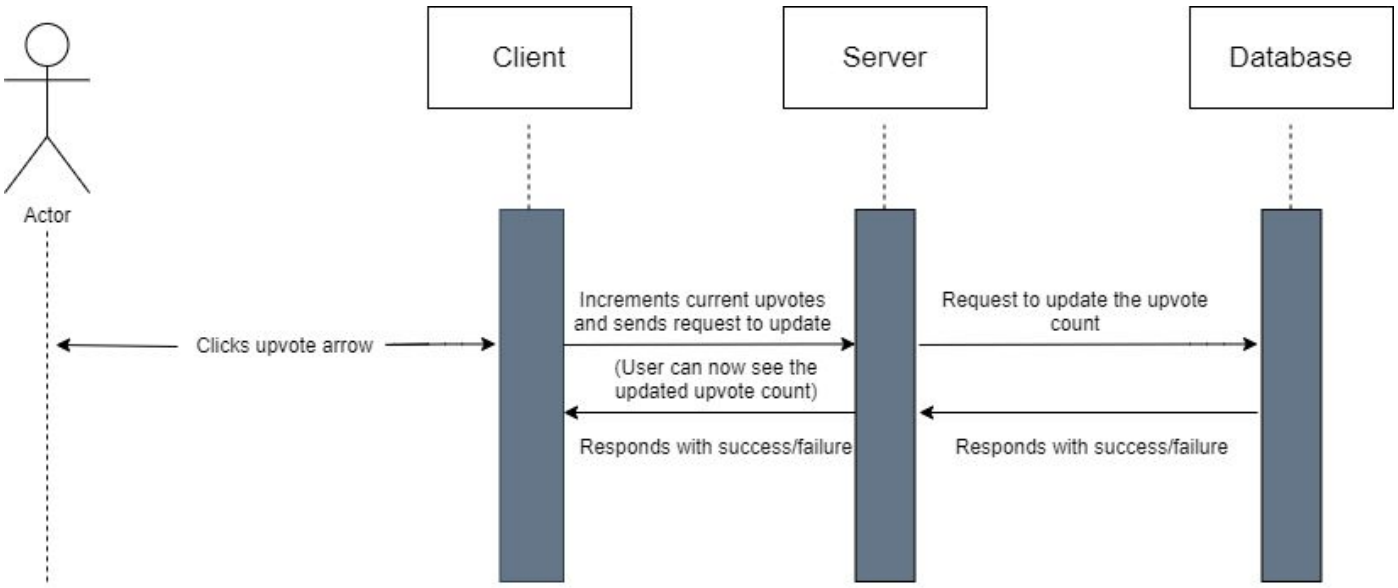
2. Sequence of events when user searches.



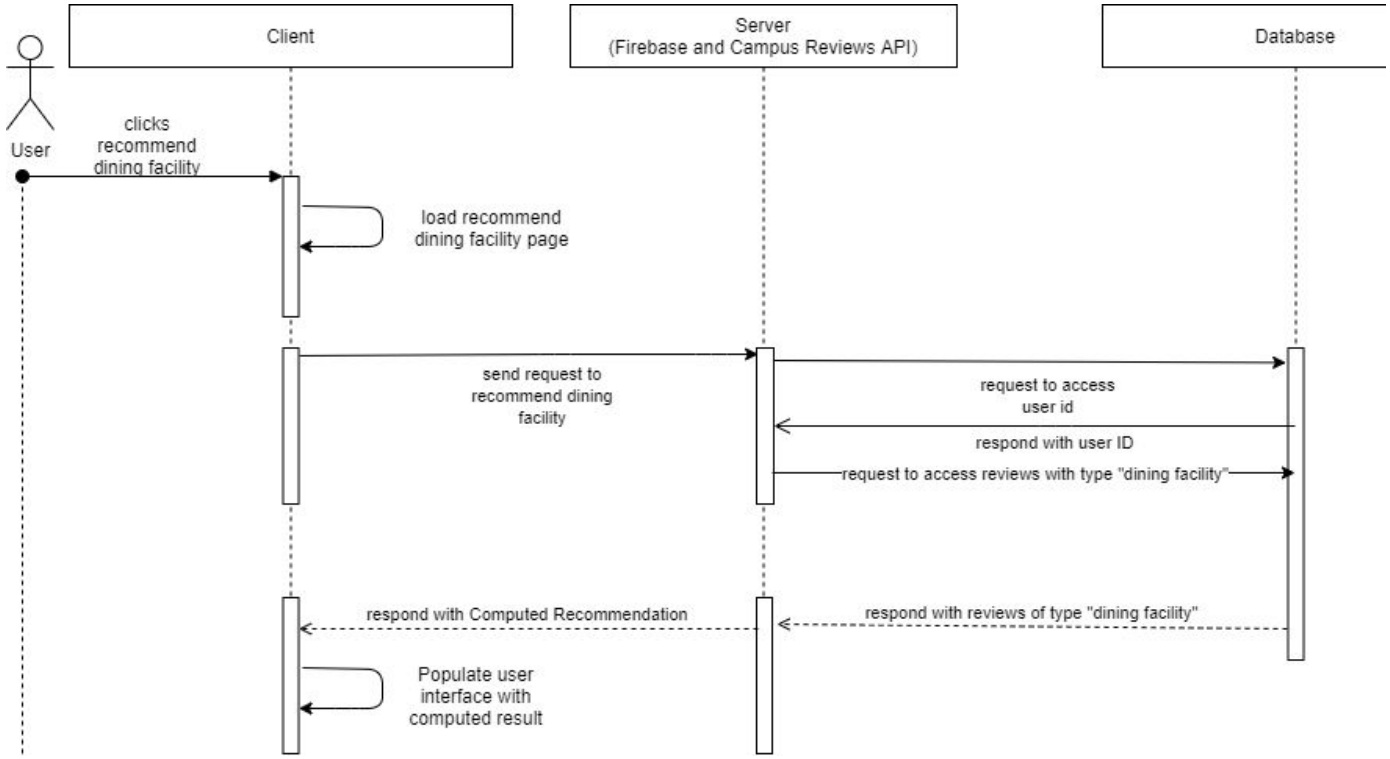
3. Sequence of events when user creates a post



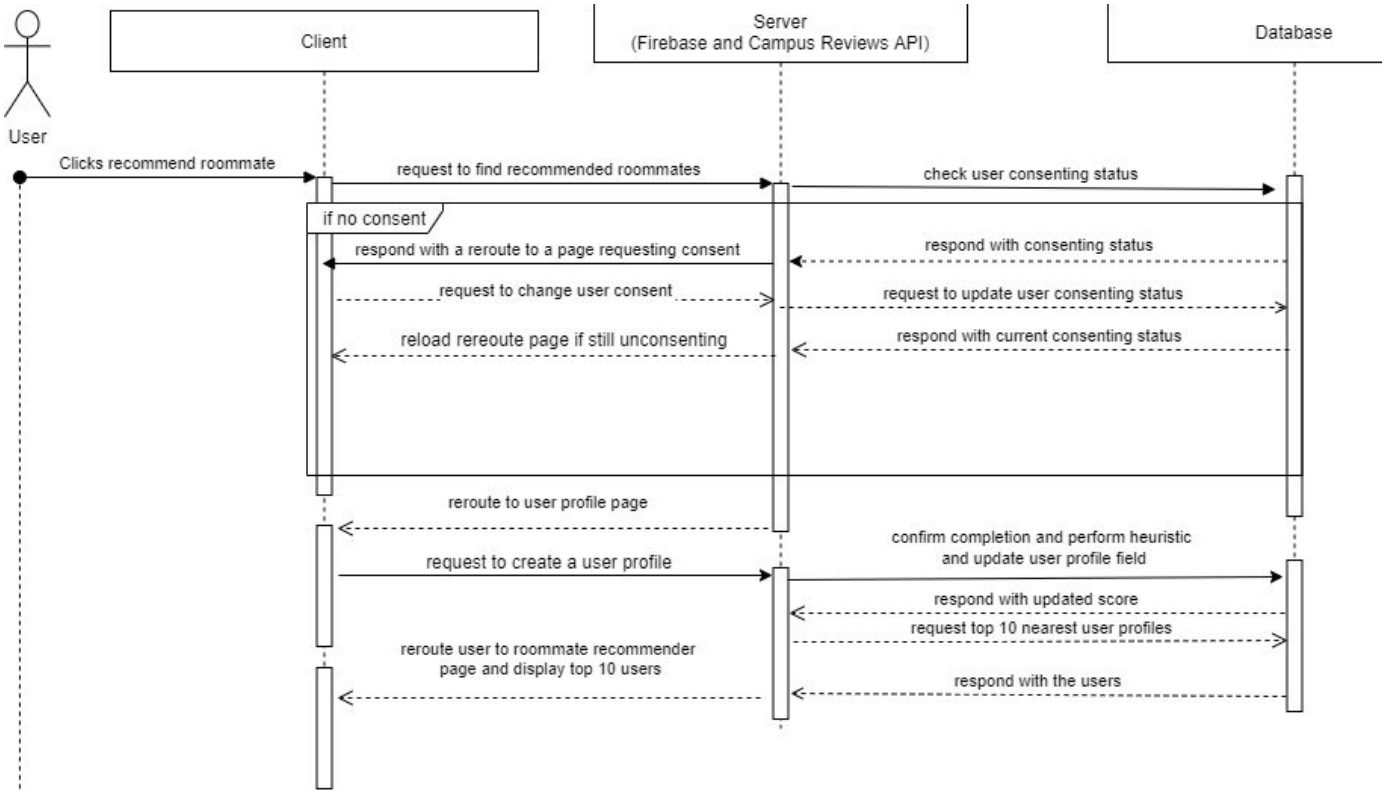
4. Sequence of events when a user upvotes/downvotes a post.



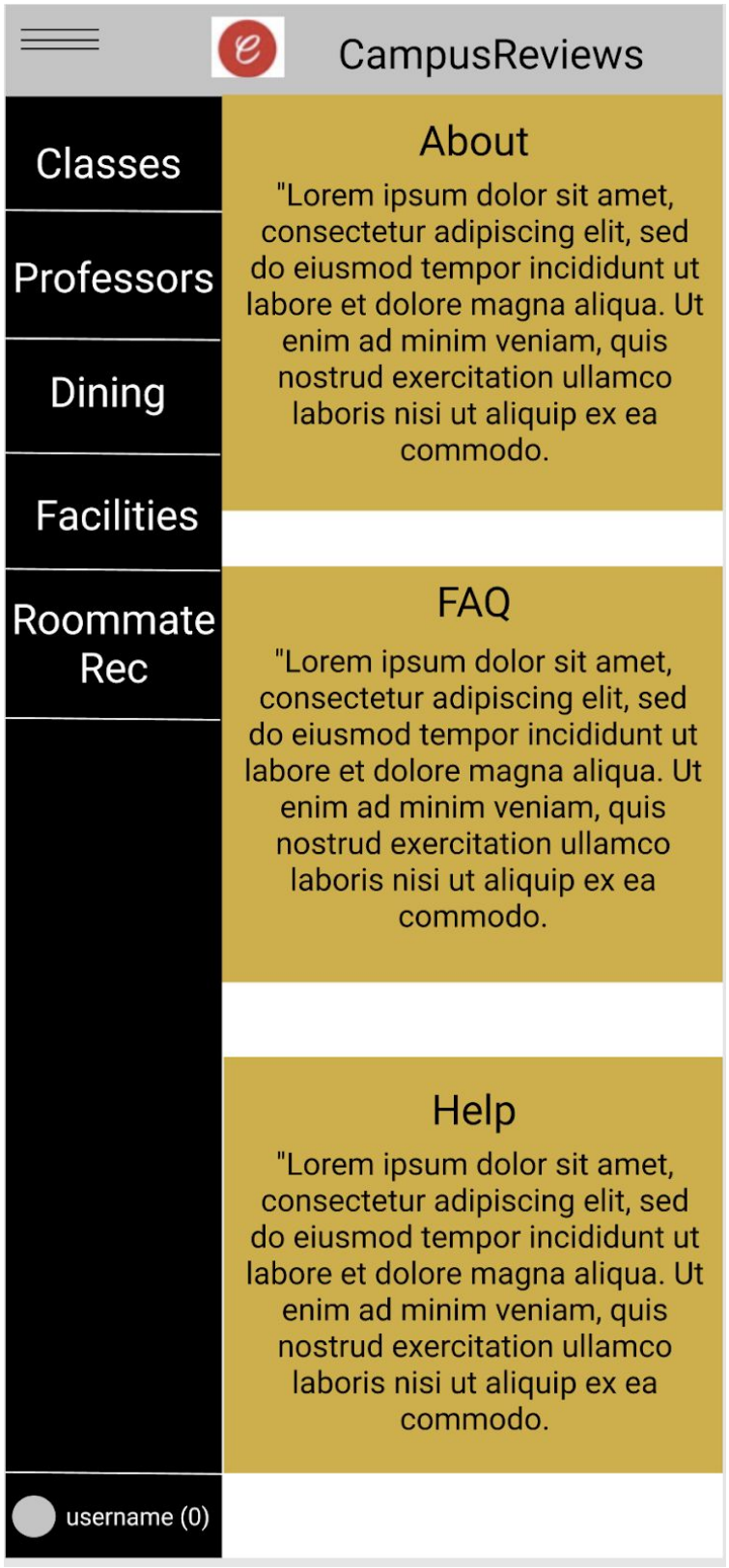
5. Sequence of events when a user requests dining recommendation.



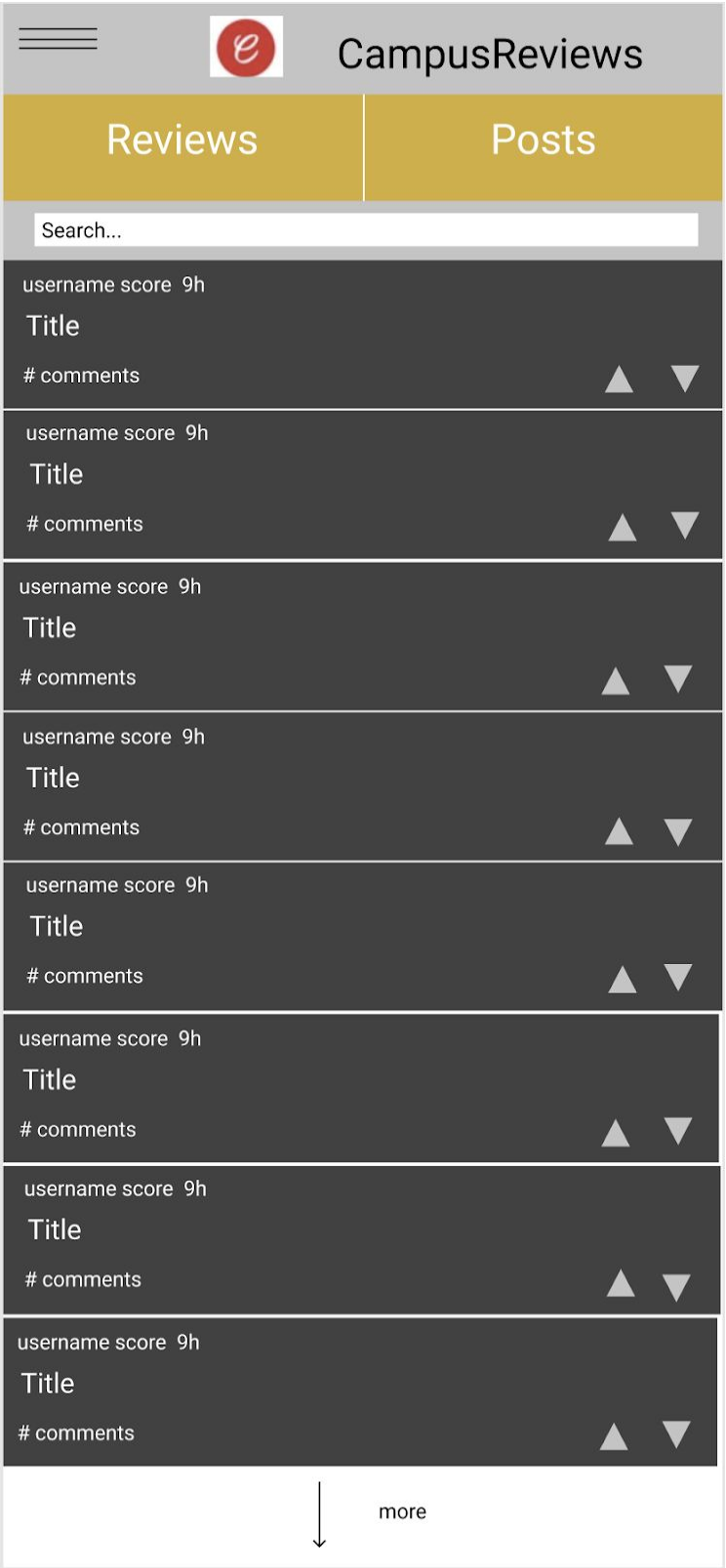
6. Sequence of events when a user requests roommate recommendation.



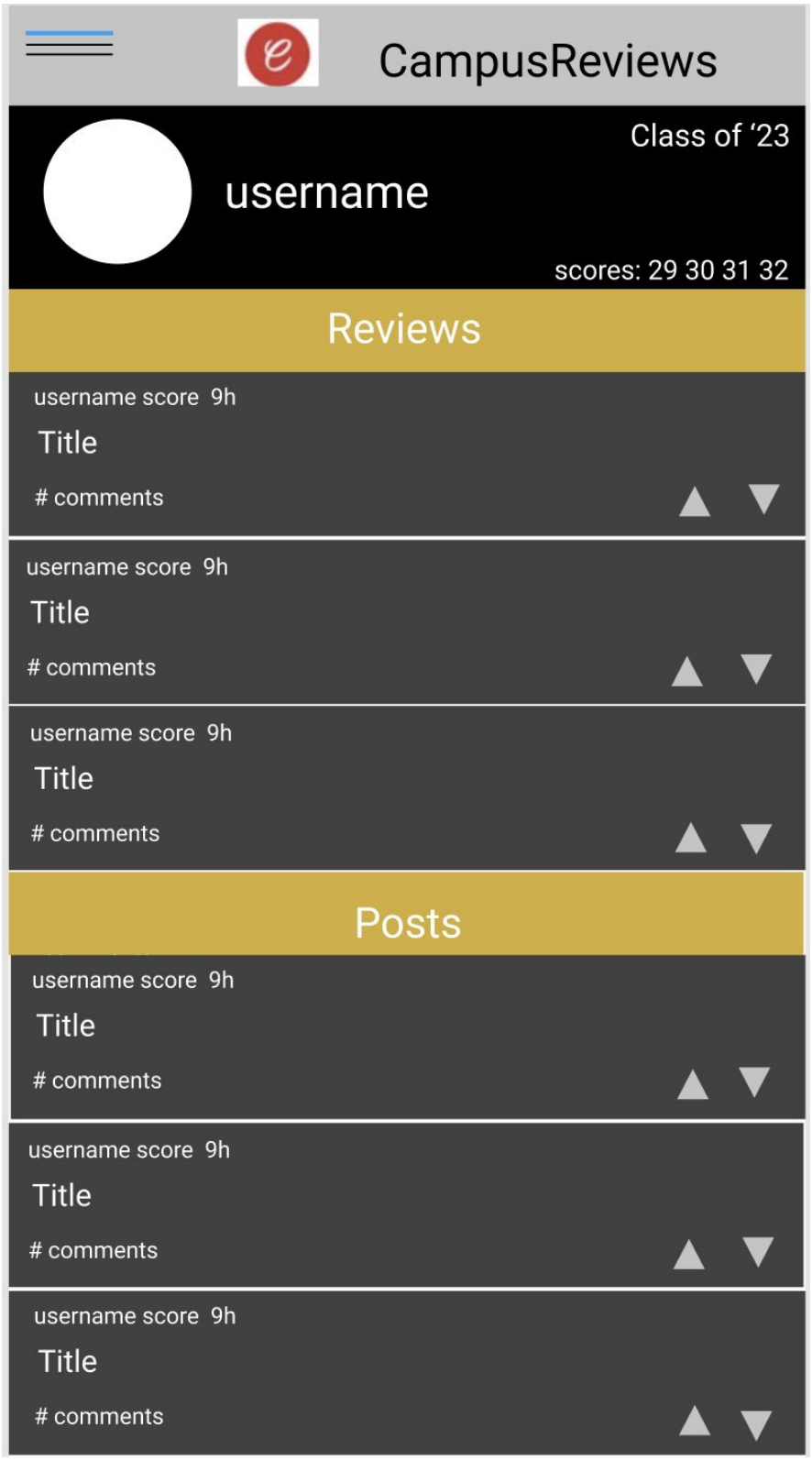
UI Mockup



Main screen with menu to navigate to different sections of the application.



Post and review screen template for all the screens that will contain user posts and reviews. Will be used at least 4 times in our application.



User profile screen. Will display their username, profile picture, graduation year, and credibility scores. It will also display their most recent posts and reviews.