****

**CS 30700**

**Design Document**

**Team 13**

* Adil Abdugaliyev
* Izzy Stookey
* Brian Yang
* Ryan Gamble
* Justin Hay
* Nathan Stern

**Index**

* **Purpose** 3
  + Functional Requirements 4
  + Non Functional Requirements 6
* **Design Outline** 7
  + High Level Overview 7
  + Sequence of Events Overview 8
* **Design Issues** 11
  + Functional Issues 11
  + Non Functional Issues 13
* **Design Details** 15
  + Class Design 15
  + Sequence Diagrams 20
  + Navigation Flow Map 34
  + UI Mockup 35

**Purpose**Traveling alone while abroad is fairly standard, but it can be unsafe to do so. Because of this, many solo travelers have taken to finding people to travel with wherever they may be in their journey. However, there are many challenges that come with doing this successfully. First, you have to find people who want to go to the same location and are within a similar budget as you. Next, if you’re going on a trip with a group of people, you need to know that you will enjoy spending time with those people for a prolonged period. Lastly, without proper tools for organization, it can be difficult to coordinate travel details between a group of strangers like where to meet up and how to split costs for things like hotels or travel expenses at the destination, etc.

This is where TogetherTrek comes in. TogetherTrek provides a web and mobile application that allows users to search for other people traveling around the world. Users can filter trips based on the destination, budget, age range of the group, etc. to find groups they feel would be a good fit for them. Or, users can just make a post describing themselves and the type of trip they would like to go on or a place they would like to go, and other users can search their post and invite them on a trip or just connect and potentially plan a trip down the line. If a user does start their own trip or join somebody else’s trip, everybody in the group can see trip details, upload photos throughout the trip, and see a breakdown of shared costs, making logistics clean and easy. TogetherTrek also allows users to search for nearby users who would like to meet up so that if a user is traveling they have an easy way to connect with the locals and have experiences they wouldn’t otherwise be able to have.

While there are many other services that allow users to find travel partners, like Gaffl, TogetherTrek has unique features that help it stand out from the crows. For one, it is one of, if not the only, one that offers real-time safety features such as quick access to local emergency contact info, safe meet-up place suggestions, and identity verification to ensure users are meeting up with who they expect to when they get there. On top of that, many similar services do not provide features like trip budgeting/cost splitting which TogetherTrek does. Lastly, many similar services specifically target groups of two whereas TogetherTrek targets all sizes of groups for large to small and makes it easy for those groups to coordinate.

**Functional Requirements**  
1. User Account

As a user,

1. I need to be able to register an account
2. I need to be able to login into my account
3. I need to be able to sign out of my account
4. I need to be able to reset my password
5. I need to be able to change my password
6. I need to be able to change the email address associated with my account
7. I need to be able to edit my profile
8. I would like to be able to login using third party authenticators like Google
9. I would like to connect social media accounts to the profile
10. I need to be able to verify my profile

2. System Features

As a user,

1. I would like to be able to block an unwanted user from viewing my profile
2. I would like to be able to report a user for inappropriate behavior
3. I would like to get emails about special offers
4. I would like to get reminders to my personal email
5. I would like to get reminders to my personal phone number

3. Posts

As a User,

1. I need to be able to post that I am looking to go on a trip to a location
2. I need to be able to edit the posts that I make
3. I need to be able to delete my own posts

4. Trips

As a user,

1. I need to be able to create trips
2. As a user, I need to be able to join trips
3. I need to be able to set an upper limit to the number of people in a group
4. I need to be able to add specific people to a group
5. I need to be able to remove people from a group
6. I need to be able to prevent more people from joining a group
7. I need to be able to mute notifications from a specific group
8. I would like to be able to see the total amount and my share of group costs
9. I would like to be able to search for trips with filters for location and planned budget
10. I would like to be able to have the trip budget broken down into categories
11. I would like to get a trip summary with highlights of the trip
12. I would like to share a trip summary on social media

5. Messaging

As a user,

1. I need to be able to send private messages
2. I need to be able to send group messages

6. Multimedia Handling

As a user,

1. I would like to be able to share photos/videos with my group
2. I would like to be able to share particular photos/videos on social media
3. I would like to be able to download uploaded photos

7. Meeting up with Users

As a user,

1. I need to be able to share my location
2. I would like to be able to see people in my area that I can meet up with
3. I would like to be able to quickly get emergency contact information for local authorities
4. I would like to have suggestions for safe meeting spots

8. Premium Users

As a premium user,

1. I would like to be able to upload more photos than a standard user
2. I would like to be able to control who sees me when searching
3. I would like to be able to change the theme of the chat

9. Admin Features

As an admin,

1. I need to be able to edit any user’s posts
2. I need to be able to delete any user’s posts
3. I need to be able to ban any user
4. I need to be able to see user’s posts

10. Core Infrastructure

As a developer,

1. I need a running database for data to be stored in
2. I need a performant database schema so that requests are quick
3. I need any stored data to be encrypted so that it is not vulnerable
4. I need a running RESTful API to service requests from the front end
5. I need the API to have any endpoints required for server requests
6. I need to be able to create an auto profile and post moderation bot
7. I need to be able to create an admin account
8. I need to have a way to store users’ photos and videos

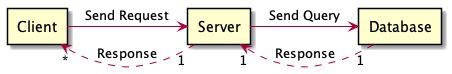
**Non Functional Requirements**

1. Performance
   1. I would like the application to run smoothly without crashing or lagging
   2. I would like backend requests to be serviced in less than 1 second
   3. I would like the backend to be able to handle at least 1000 concurrent requests
   4. I would like the automated moderation bot to verify images and text in less than 3 seconds
2. Frontend Applications
   1. I would like a web application built in ReactJS
   2. I would like Android and iOS applications built from Flutter
3. Backend API
   1. I would like an API built in NodeJS that follows RESTful principles
4. Database
   1. I would like to be able to store user data in a MongoDB database
5. Security
   1. I would like the API to restrict users from accessing data they don’t have the right to access
   2. I would like the database to be encrypted
   3. I would like a compromised frontend application to not be able to expose user data
   4. (If time allows) I would like the API to use tokens for user authentication
   5. (If time allows) I would like the API to use HTTPS for all requests
6. Usability
7. I would like the application to be easy to understand/easy to find what you’re looking for
8. I would like menus to not be overly nested
9. Appearance
   1. I would like a clean aesthetically pleasing design that is attractive to young adults

**Design Outline**

**High Level Overview**

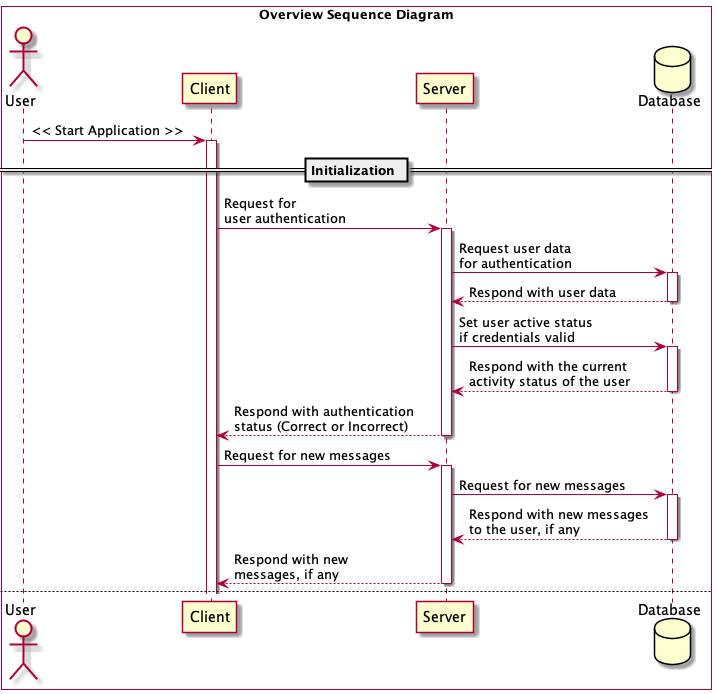
This project will be a web application that allows users to create (and post) trips, message other users, budget for trips, and share photos with other users that are found on the platform by browsing through suggested posts and searching based on specific criteria. The application will use the client-server model where one server handles requests from several users concurrently. The server will accept client requests, manipulate data in the database, and send responses back to the client(s) as needed.

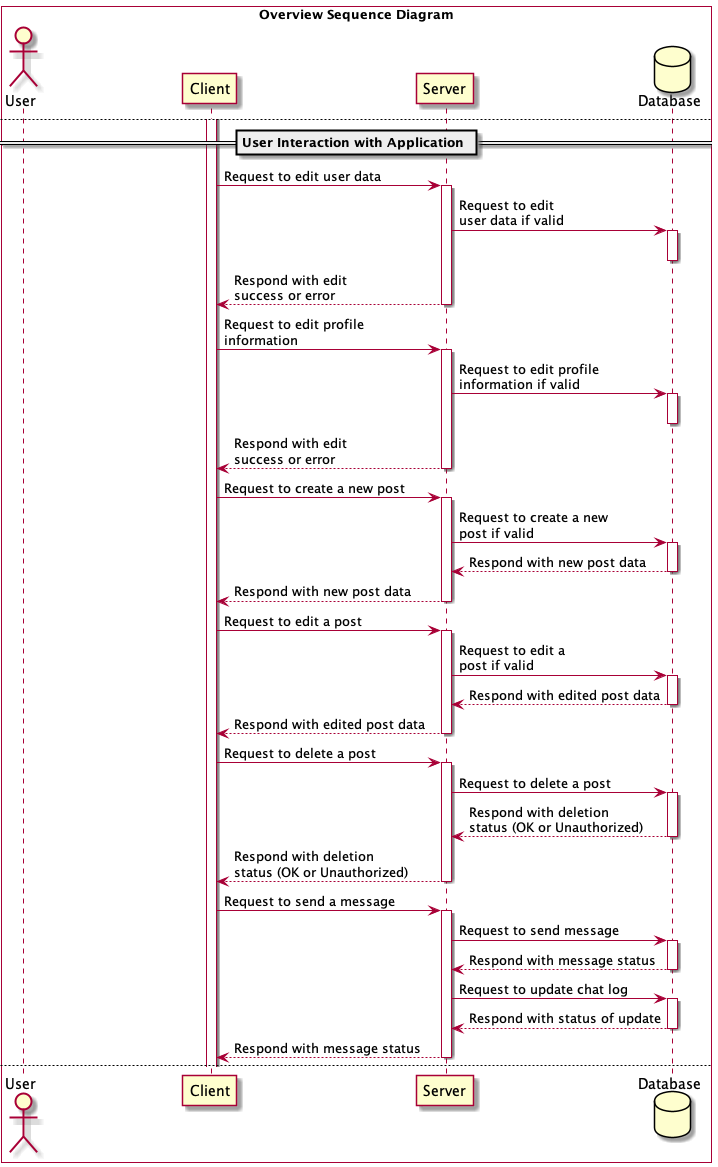


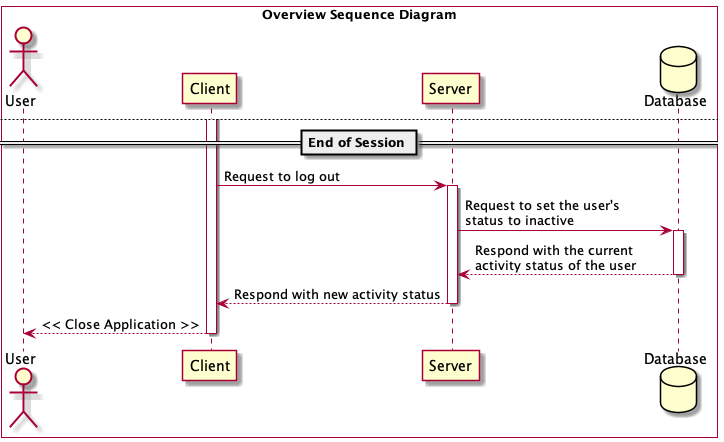
1. Client
   1. Client provides a user interface for interacting with the system.
   2. Client sends HTTP requests to the server.
   3. Client receives HTTP responses from the server and renders any necessary changes to the user interface
      1. Responses are returned from the server in JSON format
2. Server
   1. Server receives and handles HTTP requests from clients.
   2. Server validates requests and queries relevant information from the connected database.
   3. Server generates HTTP responses and sends them back to the originating client.
   4. Server conforms to a RESTful API protocol.
3. Database
   1. A NoSQL database that stores all data for the application such as user information, trip information, past user messages, etc.
   2. Database responds to queries from the server and responds with any data relevant data in JSON format

**Sequence of Events Overview**

The sequence diagram below shows the typical interaction between a client, the server, and the database. The sequence starts with the user starting the web application. When the user sends a request to log in, the client sends a request to the server. The server gets the relevant user login info from the database and validates the user’s password. Then, if successful, the server sends another request to set the user’s status to ‘active’ in the database and responds with the status of the user’s authentication to the client. After the user logs in, the client can send further requests to the server to update information, create new posts, delete old posts, or send messages. To respond, the server queries the database to get any data necessary to complete the client’s requests. After each operation, the database responds with the requested data or a status code. The server will then return any relevant results to the client to update the user interface.

****

****

****

**Design Issues**  
 **Functional Issues**:

1. What information do we need for signing up for an account?

* Option 1: Username, password, email address.
* Option 2: Username, password, email address, phone number, ID/Passport info as a safety measure
* Option 3: Username, password, email address for the view-only account. Full functionality requires additional verification.

Choice: Option 3  
 Justification: Username, password, and email address are absolutely essential for creating an account. However, we can’t accept option 1 since our service focuses a lot on personal security and it requires a verification system. We can add an additional level of security by adding ID/passport information as required at the registration stage. But this requirement may scare off some customers who just want to have a look at the app without sharing their personal data. We may solve this problem by creating a ‘basic’ account with the ‘view-only’ functionality. If the user is satisfied with the service, he can pass the verification process and engage with other users.

1. How do we ensure user’s safety during the trip?
   * Option 1: The user needs to update his status every few hours. If the user’s status isn’t updated on time, an alert is sent to his emergency contact.
   * Option 2: Button that shows the contacts of police, ambulance, etc in the country of travel

Choice: Option 2  
 Justification: Despite sounding somewhat nice, option 1 is too unpractical. The user may be too engaged in the dialogue or in the trip’s event and simply forget about updating his status on the app. Or the user may have no internet connection at the time, which will cause unnecessary stress to his family. We believe having a ‘panic’ button with important information is a better suit for trips. Prior to the trip, our app will fetch contacts of the emergency services in that particular country and save them on the user’s device.

3. Should we add premium users to the service?

* Option 1: Provide additional bonuses to premium users like prioritizing their trips over normal users’, restricting security features only for premium users, etc.
* Option 2: Have no premium user functionality
* Option 3: Both premium and regular accounts have the same functionality, but the premium accounts have a higher cloud storage size.

Choice: Option 3  
Justification: We want our service to appeal to a wider variety of users, so we don’t want to restrict the functionality of the regular users. However, since there may be users who will want to upload more photos and videos, we can provide them with additional cloud storage.

4. Should we add user ratings?

* Option 1: Add a rating system, which will sure how pleasant was the experience with a particular user
* Option 2: Do not add ratings since they’re discriminatory for the new users

Choice: Option 1

Justification: We believe that having a rating system will result in users having a better time because they won’t end up with someone unpleasant during the long trip. Moreover, the rating system also improves security since it is highly unlikely that a user with a very high rating will try to harm his group mates.

5. Should we have a reporting system?

* Option 1: Yes
* Option 1: Yes
* Option 2: No

Choice: Option 1

Justification: We believe that a reporting system will allow users to file complaints about the misuse of the platform by other users. This will also allow us, the developers, to know who to potentially remove from the service. This will work well in conjunction with our rating system.

**Non-Functional Issues:**

1. Where are we going to host our backend?

* Option 1: AWS
* Option 2: Heroku

Choice: Option 2

Justification: Heroku is a good choice for our smaller project since it satisfies our basic needs for hosting. As a container-based tool in the cloud, it doesn’t require complex infrastructure like AWS, allowing us to focus more on the developing app itself.

1. How are we going to build our frontend?

* Option 1: HTML + Javascript
* Option 2: Vue.js
* Option 3: React.js
* Option 4: Angular.js

Choice: Option 3

Justification: Writing front end in vanilla JavaScript and HTML is not an option nowadays since everything can be done way faster with the use of modern frameworks. There are many frontend frameworks that can suit our needs, but we decided to use React.js for our application. React is a very simple and lightweight library that most of our team is familiar with. Moreover, React recently moved to functional instead of class-based components, which makes the application less complex, thus more suitable for teamwork.

1. How are we going to build our backend?

* Option 1: Django(Python)
* Option 2: Node(Javascript)
* Option 3: Spring Boot(Java)
* Option 4: PHP

Choice: Option 2  
 Justification: The ability of Node to simultaneously handle multiple requests and provide prompt responses makes it an ideal solution for client-focused web app development. By implementing Node as the server technology, our team will be using the same language for both the frontend and backend. This will accelerate the development process as our team becomes more cross-functional and efficient.

1. What type of database should we use?

* Option 1: NoSQL(MongoDB)
* Option 2: SQL(PostgreSQL, MySQL,..)

Choice: Option 1

Justification: With NoSQL, we will not have to worry about hosting because MongoDB will host the database for us. We will also not have to worry about encrypting data because MongoDB does that by default.

1. Should we use HTTP or HTTPS?

* Option 1: HTTP
* Option 2: HTTPS

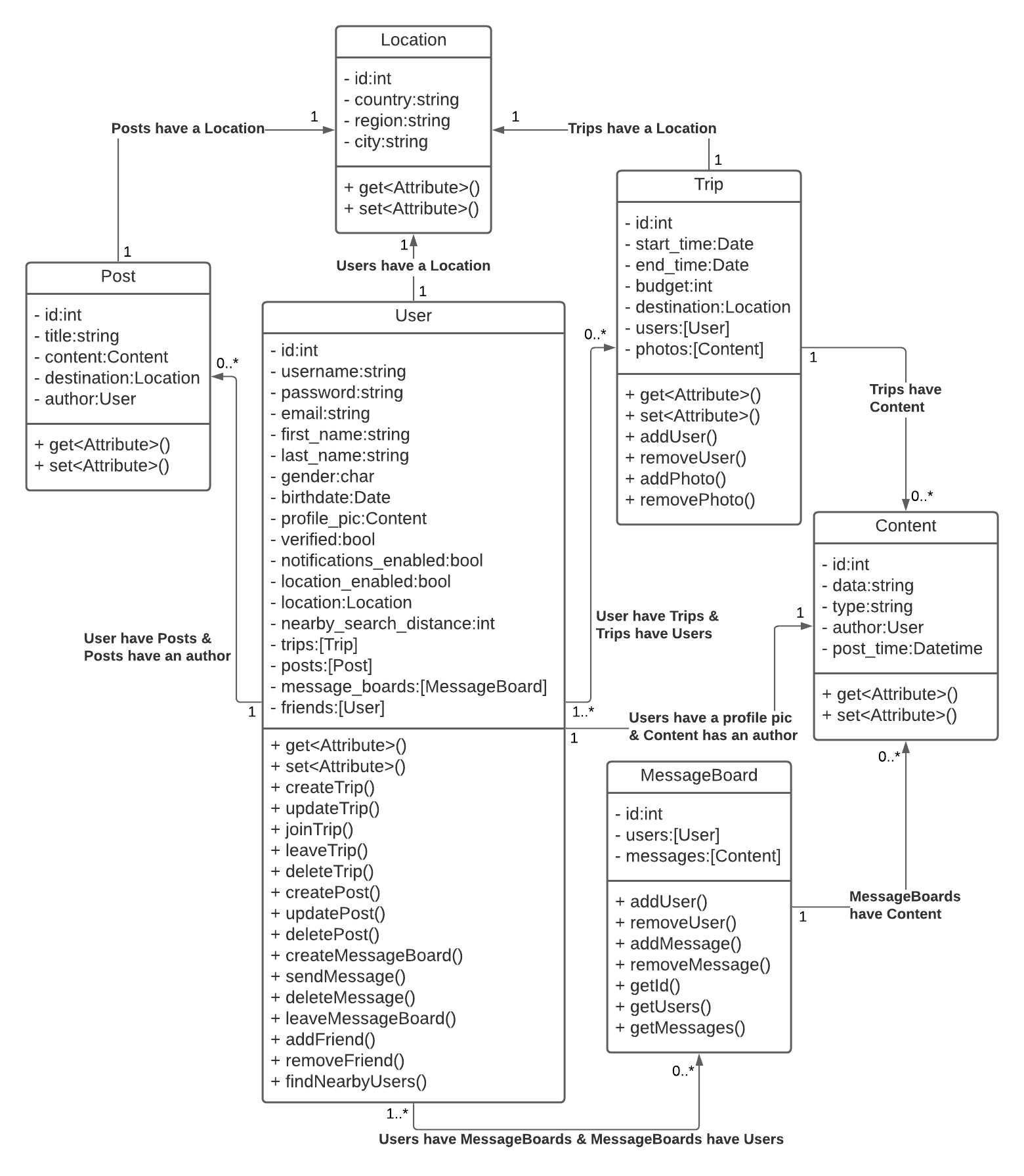
Choice: Option 2

Justification: HTTP stands for Hypertext Transfer Protocol, and it is a protocol used for transferring data over a network. Most information that is sent over the Internet, including website content and API calls, uses the HTTP protocol. HTTPS is a secure version of HTTP, where all the requests are encrypted. Thus, an attacker would see a bunch of seemingly random characters instead of valuable data.

**Design Details**

**Class Design**

**Frontend Data Classes**

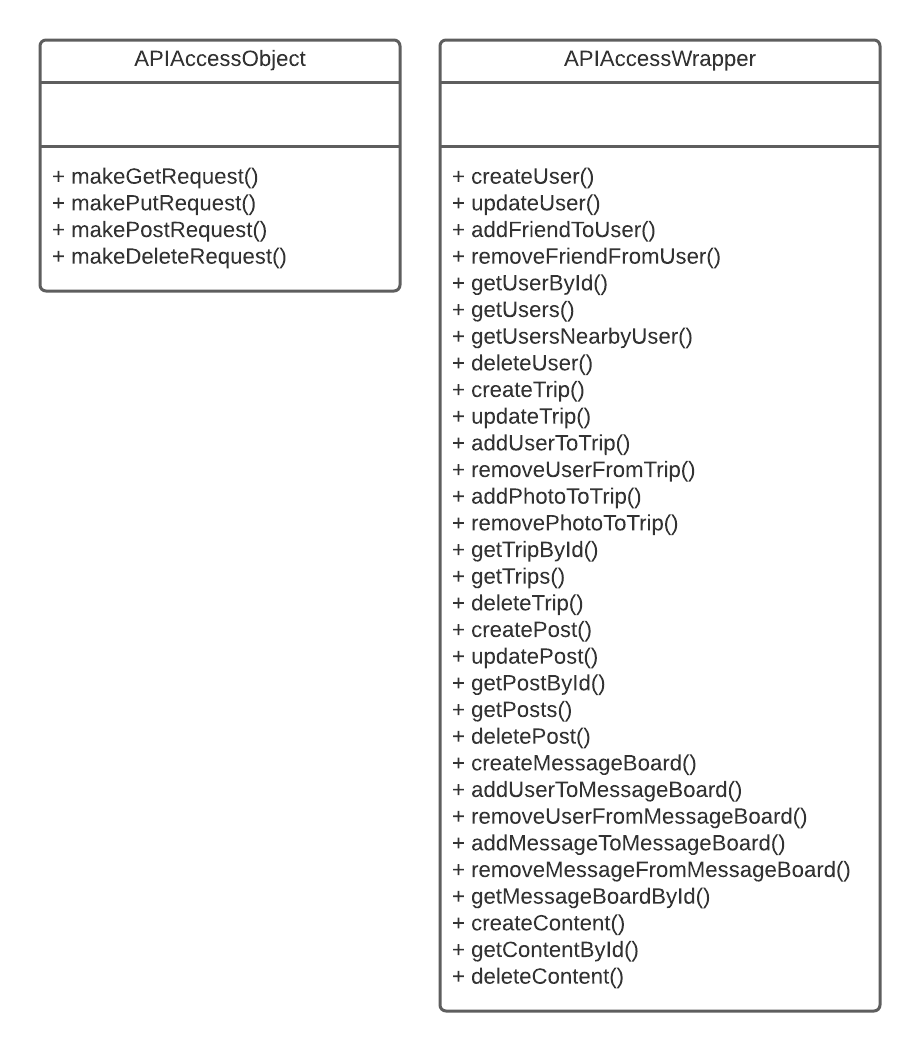
****

**Description of Frontend Data Classes and Interactions between Classes**

The classes are designed based on the objects that will be used in our front-end applications to keep track of data necessary for displaying all necessary information to the user. The values of class attributes are meant to mirror values stored in related tables in the database and all updates to class values would have an associated update to the database if the request is valid. Each class has an id attribute that corresponds to the id of the entry in the associated table of the database.

* **User**
  + User objects are created when a new user creates an account.
  + Each User has a username, email, and password for login purposes.
  + Each User also has a first name, last name, gender, birth date, and profile picture as part of their profile.
  + To control what services the user has access to, each user also has a verified, location\_enabled, and notification\_enabled attribute.
  + Users can have a list of Trips that they are either going on or have gone on which will show up on their profile. Trips that have not yet started will show up in search results by other Users to potentially join.
  + Users have a list of Posts they have made which will show up on their profile and in search results by other Users looking for fellow travelers.
  + Users have a list of MessageBoards they are a part of between 1 to many other Users.
  + Users have a list of other Users they are friends with. Friends can see more details on each other's profiles.
* **Location**
  + Locations have country, region, and city attributes used to identify the location
* **Post**
  + Posts have a title, author, content (message the author wrote describing what they would like to do, when, where, etc.), and a location where the author wants to go.
  + Posts are meant for people to advertise that they want to go somewhere but they don’t have a solid plan yet so they’re just seeing who else is interested in going to the same place.
* **Trip**
  + Trips have a location, start, and end date to let Users know when and where the trip is happening.
  + Trips also have a list of Users that have agreed to go on the trip together.
  + Trips also have a list of photos that Users from the trip have uploaded.
* **MessageBoard**
  + MessageBoards are for storing information for messages between users. If two users message each other, they create a MessageBoard. If a group chat is created, a MessageBoard is created.
  + MessageBoards have a list of Users that can see the associated Content (messages, photos).
  + MessagesBoards have a list of Content (messages, photos) that Users in the MessageBoard have sent and can see.
* **Content**
  + Content objects are either text or links to photos
  + Content objects have a data attribute (the text or link), a type (text or link), an author (the User who posted the Content), and a time that the Content was posted

**API Access Classes**

****

**Description of API Access Classes and Interactions**

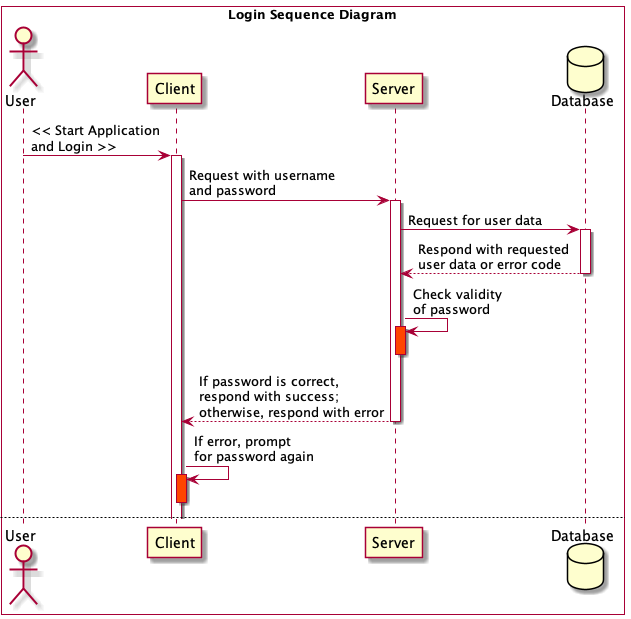
All interactions with the database must pass through the API so that we can control what users have access to what data and sensitive data is protected. To do this we will use two classes called APIAccessObject and APIAccessWrapper. Whenever the frontend needs data from the database, it will make a function call to APIAccessWraper which will make any necessary function calls to APIAccessObject which will make requests to the API directly.

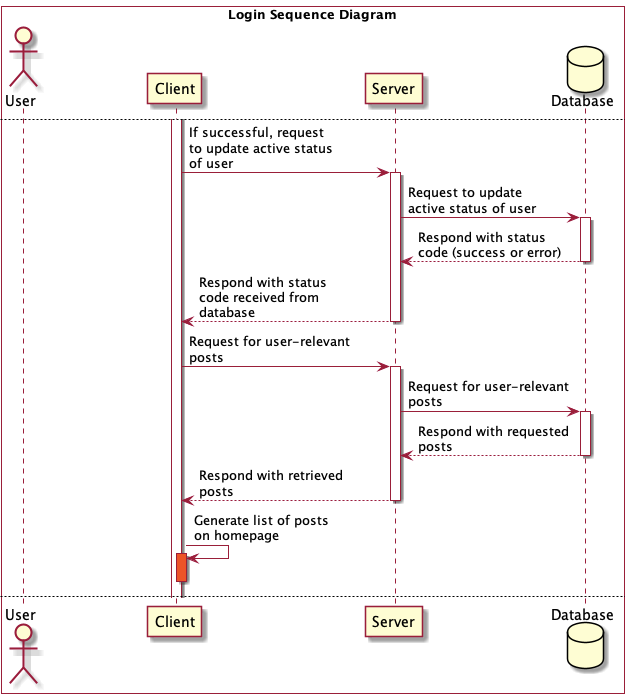
* **APIAccessObject**
  + The APIAccessObject is the class that will make all of the direct calls to our backend API. This class will be able to make generic GET, PUT, POST, and DELETE requests accepting arguments like endpoint and data.
  + Requests should be validated on the API but the APIAccessObject can also do some validation like ensuring only valid fields are passed however it should be kept in mind that since this is application code it could be tampered with so it is not the last line of defense.
* **APIAccessWrapper**
  + This class will have all the specific types of requests that can be made to the API. The internal classes of our frontend should not need to care about endpoint structure when requesting data from the database so this class will handle that abstraction.
  + This class will have straightforward functions like addUserToTrip() that the frontend classes can call and then inside the function APIAccessWrapper will make any necessary calls to APIAccessObject which will return the results of the HTTP requests and if the requests are successful, APIAccessWrapper will return any necessary data to the frontend classes in the format those classes expect rather than as a request result

**Sequence Diagrams**

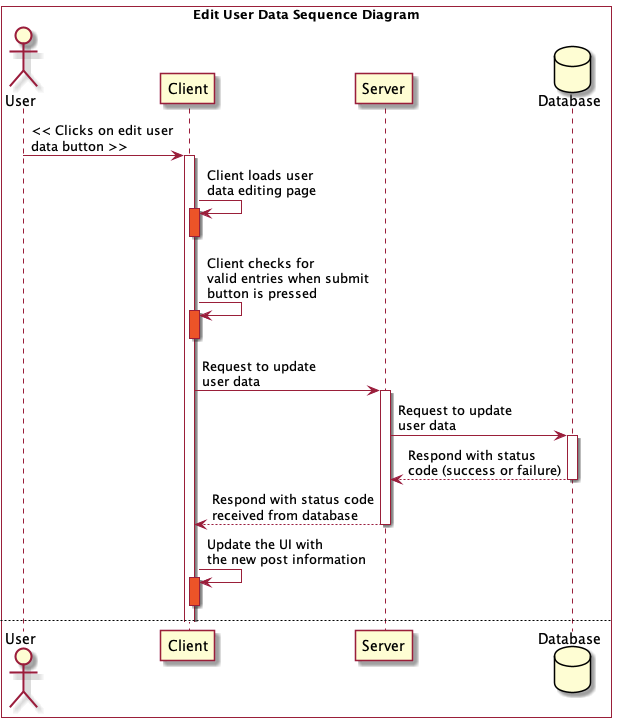
The following sequence diagrams show the sequence of events for all major events in this application. This includes user login, editing user data, editing profiles, creating and editing trips, creating and editing posts, filtering posts/trips, getting an emergency number, viewing a profile, sending a message, rating a user, and logging out of the application. When a user performs an action in the client, a request is sent to the server, which verifies the input, and a query is sent to the database to get any needed information. The server will return the requested data to the client, which will format the data and update the UI.

1. Sequence of events when the user logs into the application

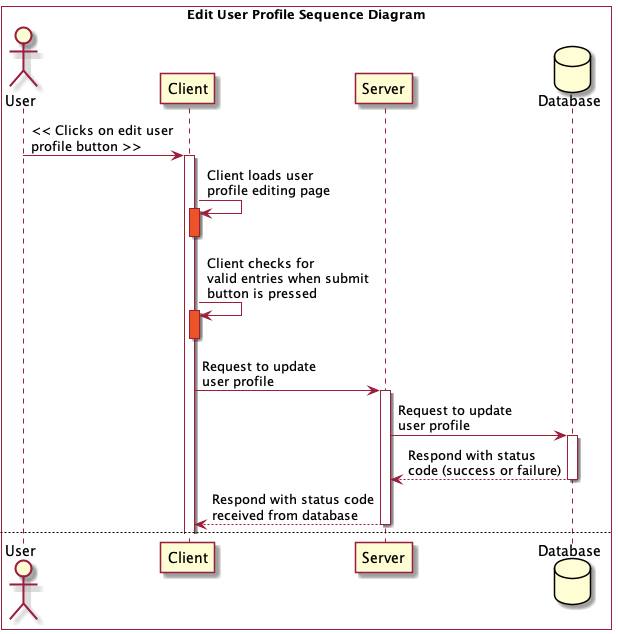




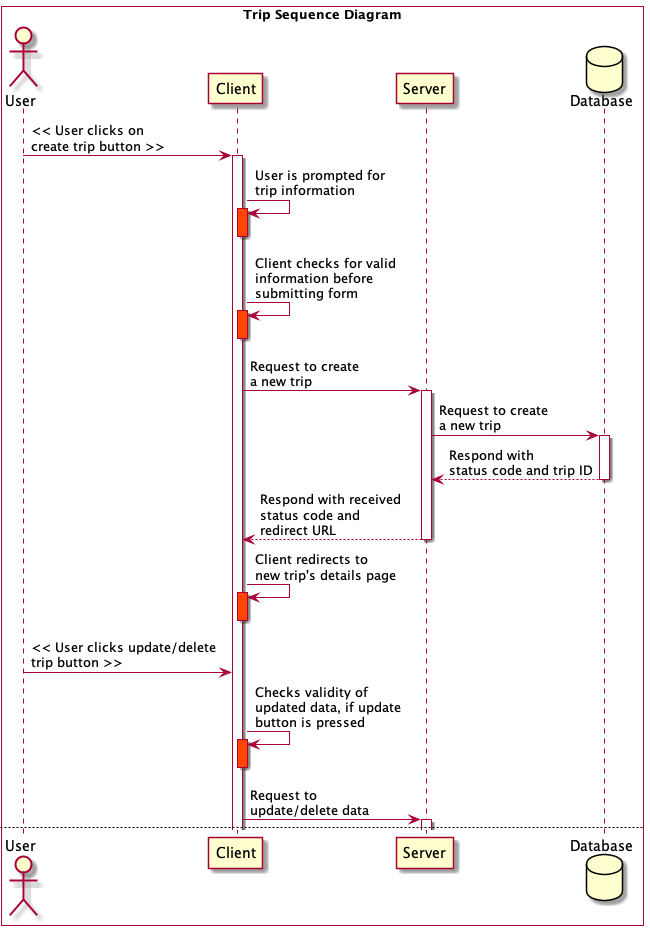
1. Sequence of events when the user edits their data

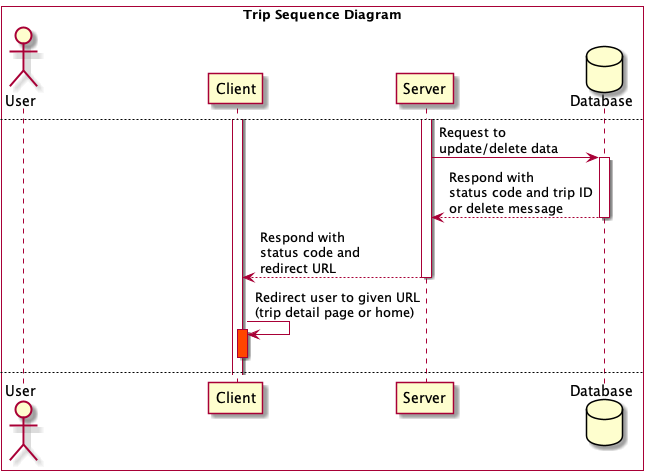


1. Sequence of events when the user edits their profile

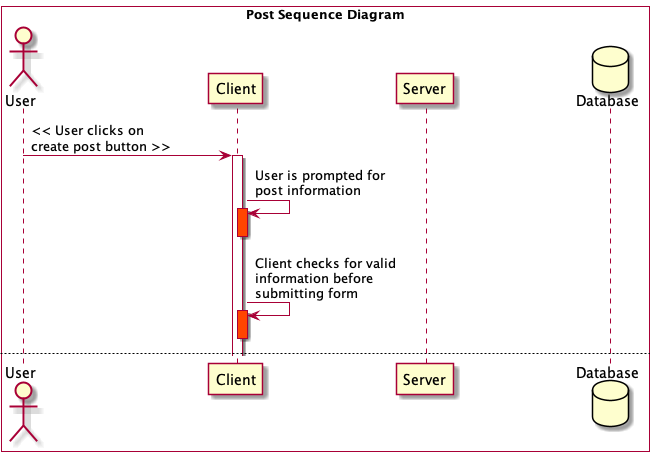


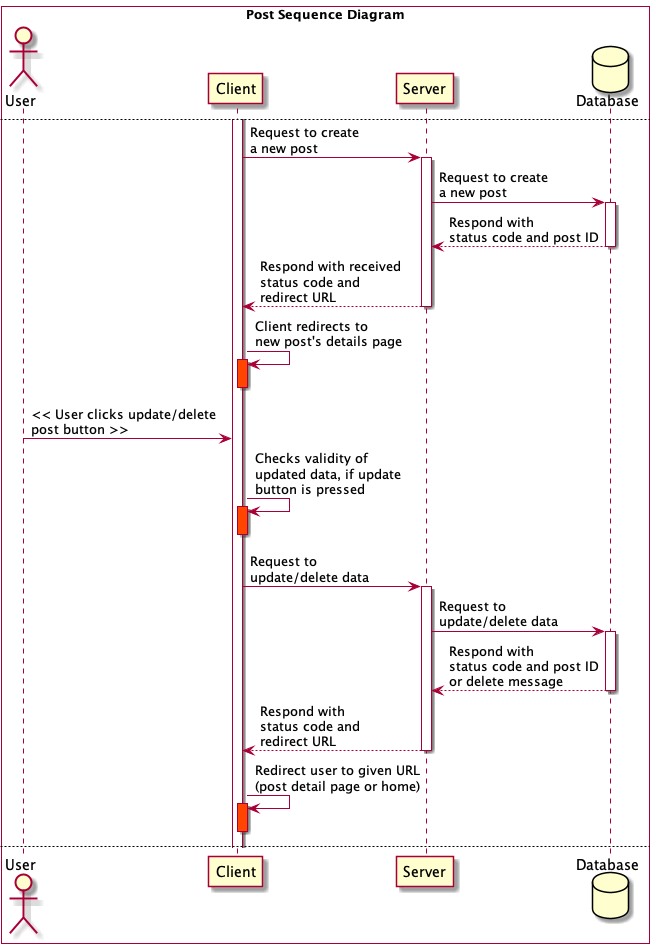
1. Sequence of events when the user creates, edits, or deletes a trip



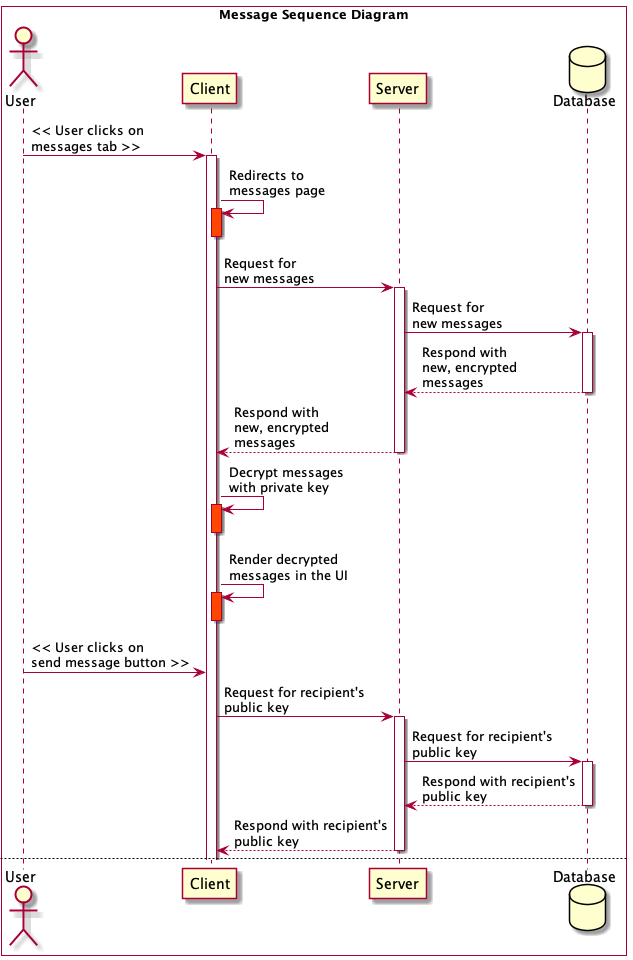


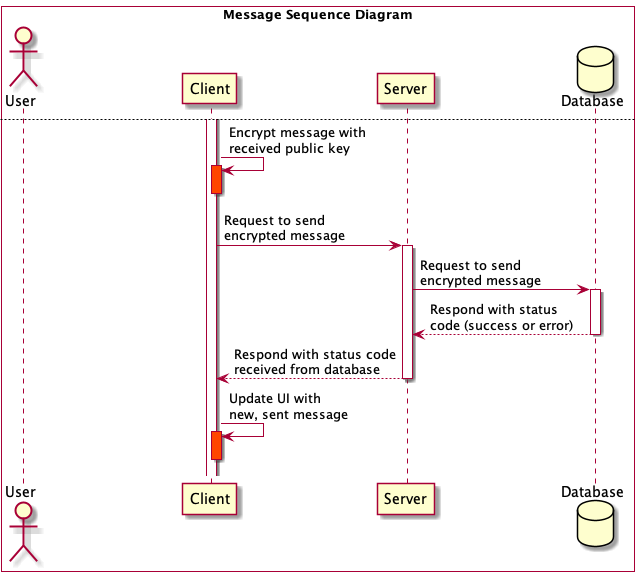
1. Sequence of events when the user creates, edits, or deletes a post



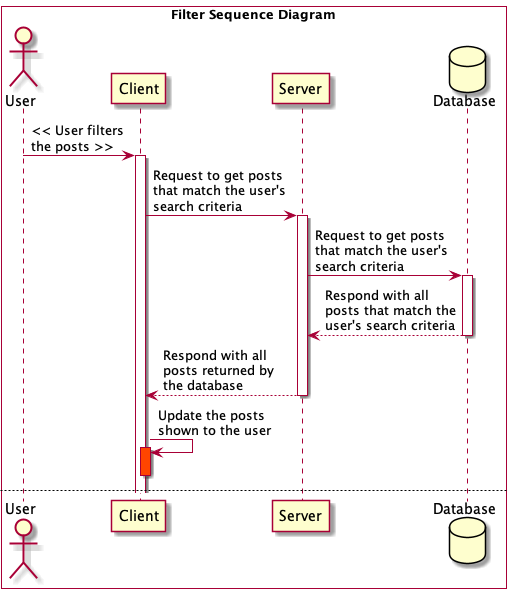


1. Sequence of events when the user sends a message

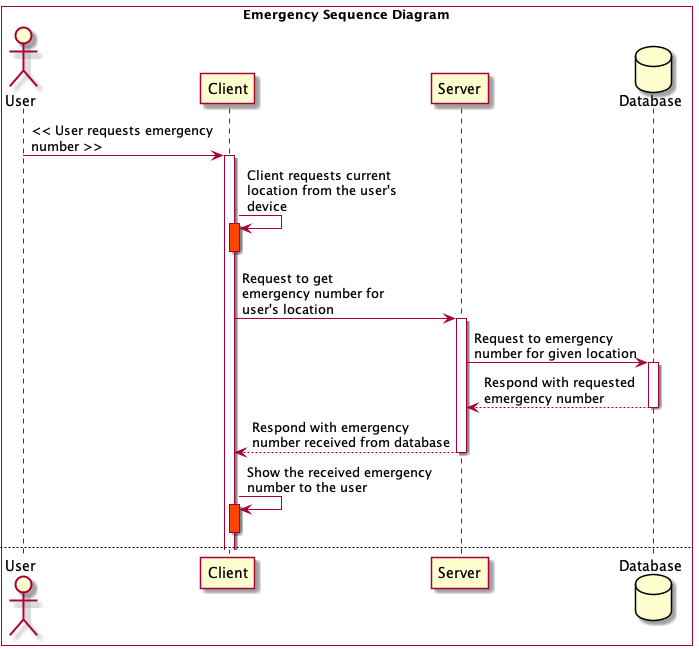




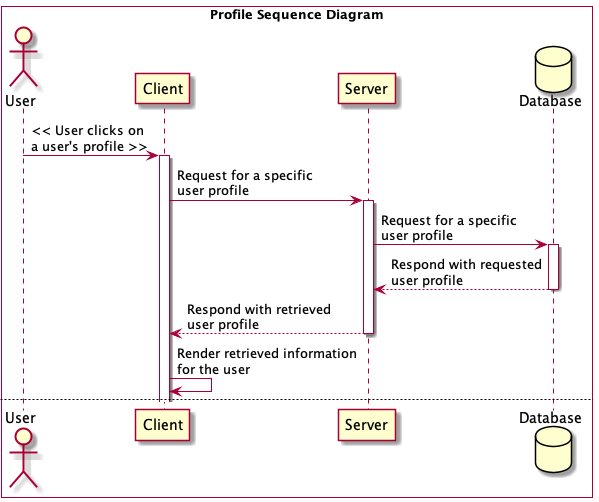
1. Sequence of events when the user filters all posts



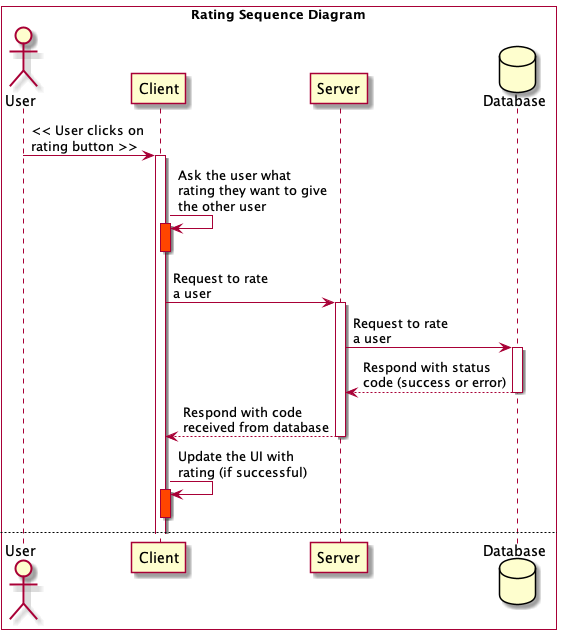
1. Sequence of events when the user requests a local emergency number



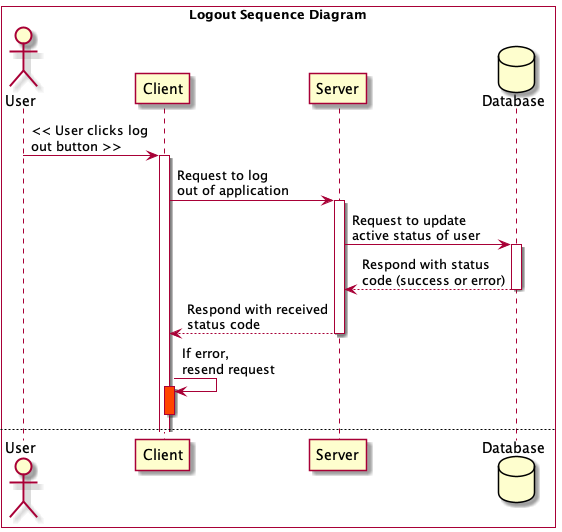
1. Sequence of events when the user views a profile



1. Sequence of events when the user rates another user

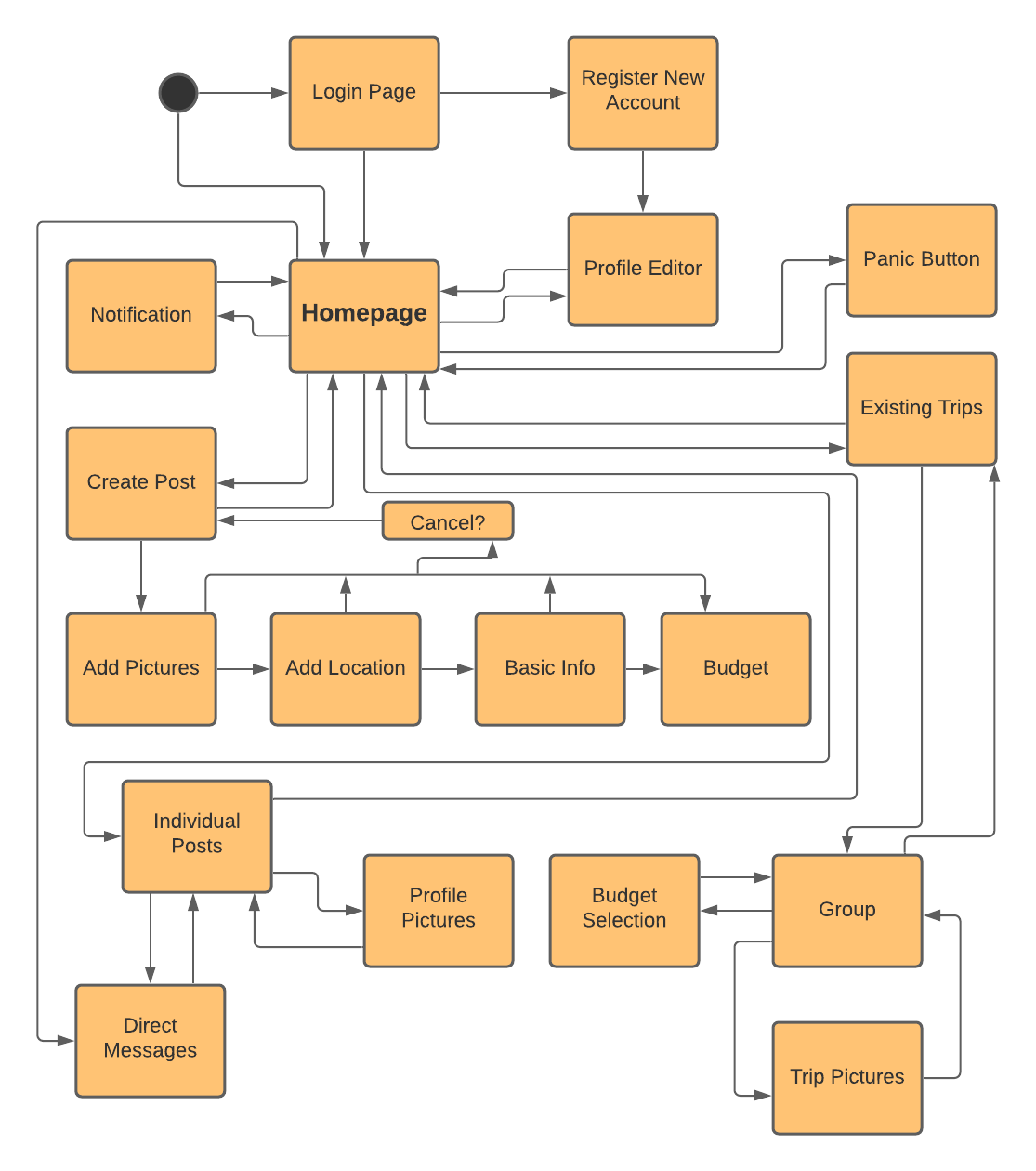


1. Sequence of events when the user logs out of the application



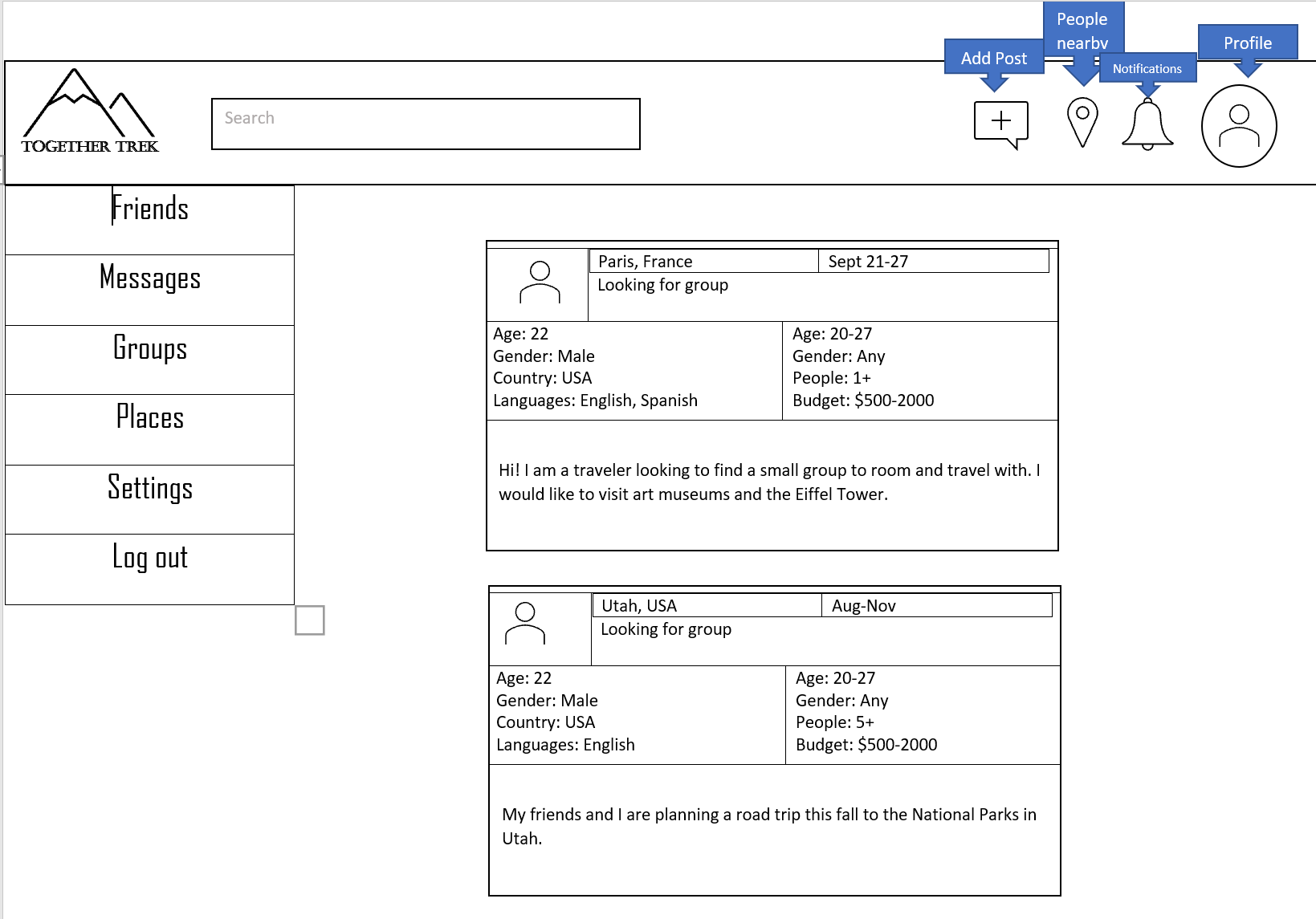
**Navigation Flow Map**

The main focus of this application is to help others find a travel partner. This is why so many of the functions lead back to the Homepage, where the users can utilize TogetherTrek’s filter system to match with people of similar interests. If the device has no past history of any user using TogetherTrek, it will require the user to log in or register a new account. A user can create a new event by adding pictures, a location, some basic information, and a budget to abide by. Any user can expand others’ posts to see their profile pictures or send a direct message to demonstrate their interest in joining the future trip. If the two sides agree to go on this trip together, a group can be created and the group can now split the budget and now have access to a cloud photo album.



**UI Mockup**

Logged in Home Screen:



Logged Out Home Screen:

