ClapBack

CS 30700
Design Document

Team 1:

Aidan Chen
Dawon Jeong
Ethan Buck
Ilhoon Lee
Luke Lawson
Sooha Park

Index

Purpose

Academia is arguably the most common way people make friends throughout their lives. By being constantly surrounded by relatable peers in school and having sufficient free time, people can easily feel fulfilled in their social lives as a student. However, once someone has graduated, this forced interaction through classes and recesses are gone, which makes building and maintaining friendships significantly more difficult. Friends drift apart once they go off to work in industry, and with no incentives to maintain friendships, people can start to feel incredibly lonely.

Our mobile application, ClapBack, remedies this issue by maintaining a list of a user's registered friends, and randomly selects a friend for the user to connect with for the day. For the whole day, this user is only permitted to talk with the randomly selected friend, and after 24 hours a new friend is chosen. The purpose of this app is to make maintaining relationships less strenuous and time-consuming while offering extra incentives for people to stay connected and bonds such as in-app customization and activities. There are many messaging apps competing for phone screen time, but most focus on maintaining contact with current friends. ClapBack focuses on all friends both old and new, helping users stay connected no matter where they go and no matter how long they haven't seen their friends.

Functional Requirements

1. User accounts
    a. As a user, I would like to register for a ClapBack account and log-in using my email address and a password.
    b. As a user, I would like to set a unique username, profile picture, and description for other users to see.
    c. As a user, I would like to edit my name, description, profile picture, password, and email on my account or delete it.
    d. As a user, I would like to recover my account should I forget my password.
    e. As a user, I would like to link other social media accounts where I can be contacted.
    f. As a user, I would like to import my contact list.
2. Friends list and requests
    a. As a user, I would like to be able to scroll through a list of my friends and see how many friends I have.
    b. As a user, I would like to be able to search for other user accounts by username and send them friend requests.
    c. As a user, I would like to be able to see another user's profile picture, username, and description.
    d. As a user, I would like to be able to set a nickname for a friend only visible to me.
    e. As a user, I would like to receive friend requests and decide to either accept or deny them.

f. As a user, I would like other accounts based on mutual friends to be recommended to me as friends.

3. Messaging
    a. As a user, I would like to have a standard and seamless messaging experience with my chosen friend.
    b. As a user, I would like to have chat features such as read receipts, reactions, replies, showing when the other person is typing, and support for sending images and videos.
    c. As a user, I would like to be able to be given a prompt when I have trouble finding a topic to talk about with my friend.
    d. As a user, I would like to see chat histories with friends I have messaged previously.
    e. As a user, I would like to have a prompt to answer with my friend if we are having difficulty starting a conversation.
4. Friend-choosing system (placeholder name)
    a. As a user, I would like to be able to click on an icon on the front page of the app to navigate to my chat with my selected friend for the day.
    b. As a user, I would like to have a friend randomly chosen to message at the same time as all other users during the day.
    c. As a user, I would like to see how much time is left until a new friend is chosen.
    d. As a user, I would like to choose if I want to receive a notification when my friend is chosen and if they message me.

e. As a user, I would like to reveal my other contact information on my profile if I wish to continue the conversation after the day is over.

5. Gamification system
    a. As a user, I would like to keep track of how many consecutive days I have used the app.
    b. As a user, I would like to be rewarded for continued interaction with friends, including:
        i. In-app customization (i.e. chat room or main page themes)
        ii. Extra chat features
    c. As a user, I would like to engage in fun chat activities with my friend, including:
        i. “Would you rather” questions generated by the system
        ii. “20 questions” word guessing game

Non-Functional Requirements

1. Platform and Performance
    a. I want the application to work as an Android mobile application.
    b. I want the application to be responsive and handle several concurrent users.
2. Architecture
    a. I want the application to use Flask/Firebase as the backend service to facilitate the development process and increase performance.
    b. I want the application to communicate through Flask with the real-time database in Firebase to create the main chat service and authentication system.

c. I want the front-end of the application to be written in Kotlin to target not only Android but other platforms such as iOS and Java desktop applications.

3. Usability
    a. I want the application to be intuitive for users and easily navigable so that a user can contact their friend.
    b. I want the application to have a short tutorial of the app so that users can quickly learn its features.
    c. I want the action of messaging a friend for the day to be as easy as possible.
    d. I want the application's GUI to be very simple and not cluttered.
4. Security
    a. I want the application to encrypt and hash user passwords in a database with a password salt.
    b. I want to store the number of messages sent between users to implement gamification, but not store the content of the messages.
    c. I want the application to recover my forgotten password if I correctly answer preset security questions.
    d. I want the application to allow me to edit my password.

Design Outline

High Level Overview

This project will be an application that allows users to message with only one friend randomly chosen from the user's list of friends. This application follows the standard Client-Server model with Flask for our Server and Firebase for our Database communication. Through Firebase, we can handle a large number of users at once and store their profile and conversations in a database.

I. Client

A. The client is where the user will be able to interact with our program

B. The client communicates with the server through AJAX requests

C. The client will be able to receive AJAX responses from the server and display the requested information

II. Server

A. The server will be able to communicate with the client and the database through Flask

B. The server will be able to receive requests from clients, grab/edit the required information from the database, and return the data to the requesting client

C. The server will also be able to retrieve data from client and request information to be updated in the database

III. Database

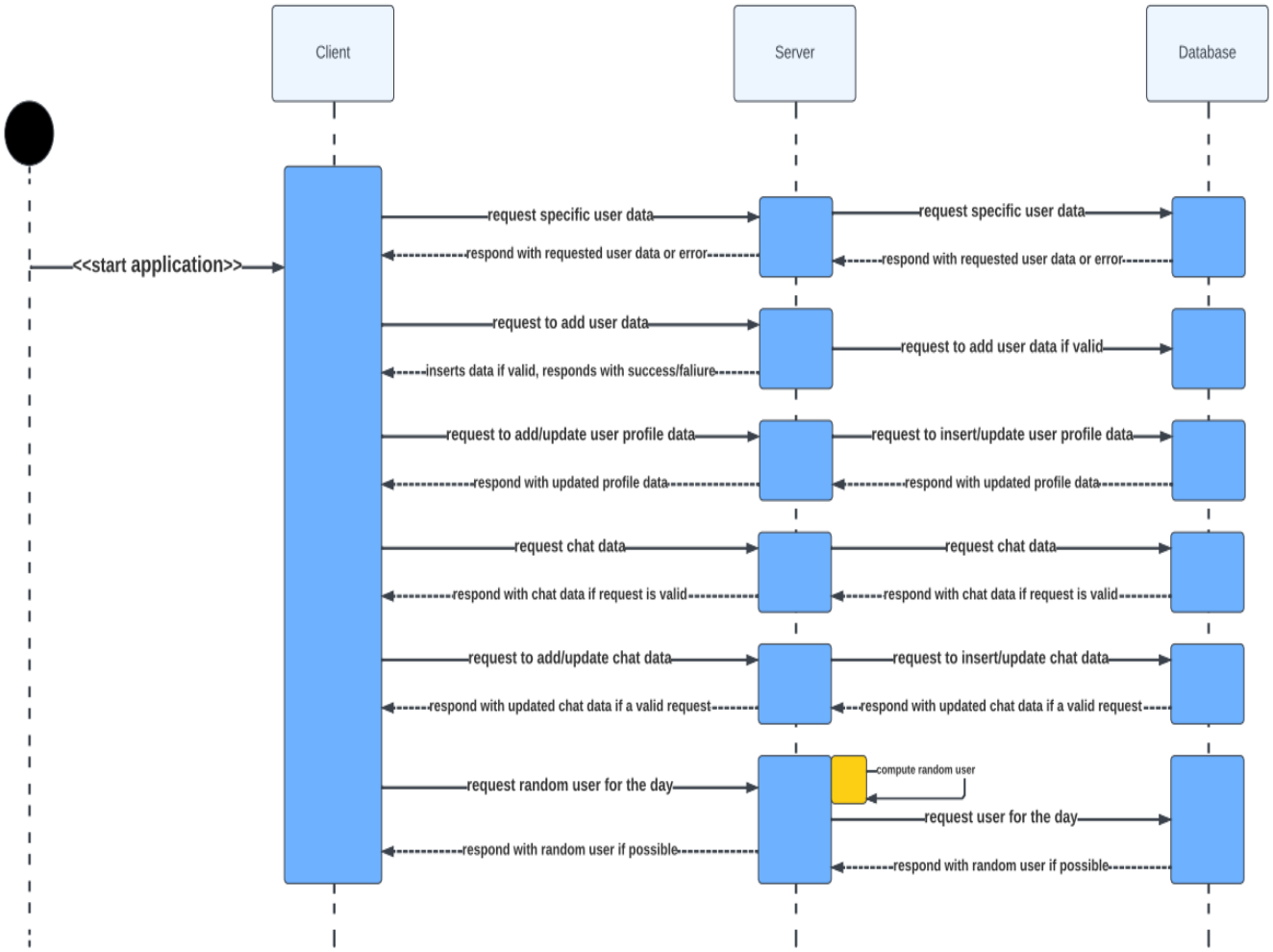
A. The database will be able to receive requests from the server and return or modify the requested data

B. The database will store all user information such as login credentials, profile details, and friends

C. The database will also store conversation information such as previous messages, timestamps, and read receipts

Sequence of Events Overview

The displayed sequence diagram outlines the interaction between the client and server/database. ClapBack uses Flask and Firebase, a micro web framework that communicates to a cloud-hosted NoSQL realtime database. The sequence begins with a user starting the phone application. The user's data will be sent to the server as a request to log in, and the server will respond with the appropriate data if given a valid query. Once the user is logged in, they will have the ability to make requests editing their profile data, adding new data, grab their chat data, and update or add to chat data. The server/database will then update the requested data and send the results back to the client. When the user first logs in per 24-hour increments, a request will automatically be sent to the server/database for their chosen user for the day. The server will algorithmically calculate a user for the client and respond with said user if the request was valid.


Client
Server
Database
<<start application>>
request specific user data
respond with requested user data or error
request specific user data
respond with requested user data or error
request to add user data
inserts data if valid, responds with success/faliure
request to add user data if valid
request to add/update user profile data
respond with updated profile data
request to insert/update user profile data
respond with updated profile data
request chat data
respond with chat data if request is valid
request chat data
respond with chat data if request is valid
request to add/update chat data
respond with updated chat data if a valid request
request to insert/update chat data
respond with updated chat data if a valid request
request random user for the day
respond with random user if possible
compute random user
request user for the day
respond with random user if possible

Design Issues

Functional Issues

1. What information should we ask for when signing up for an account?

- Option 1: Username and password only
- Option 2: Username, password, and email address
- Option 3: Username, password, email address, and dual factor authentication

Choice: Option 2

Justification: To set up an account a username and password is essential to identify a user. An email account also allows for extra security and allows a recovery system in the case that the user forgets their username or password. Dual factor authentication adds another layer of security for a user account. This makes it very difficult for attackers to gain access to user accounts. However, for a messaging application, this level of security may not be necessary. Since our app will not be storing sensitive user data, having a username, password, and email address should be enough for our project.

2. How do we access user storage and contacts?

- Option 1: Require it in permissions for the application to run
- Option 2: Only request when feature is needed

Choice: Option 2

Justification: We need access to the user's storage whenever they want to upload an image or video file from their phone to send to others or store as their profile image. We decided to only request this when that feature is specifically needed to respect the privacy of our users. Similarly, we want to access the contacts of the user to automatically sync any users already using

ClapBack. We also wanted to make sure this would only be a feature if the user wanted it to protect privacy.

3. How often do we show walkthrough of application

- Option 1: Everytime the application is opened
- Option 2: Every couple months
- Option 3: Only the first time the application is opened
- Option 4: When user manually requests the walkthrough

Choice: Option 3 & 4

Justification: The walkthrough of the application is meant to be a short feature that helps users understand what the main features of the application are. Because of this, having the mini-tutorial pop up everytime in unnecessary. We thought it would be sufficient to just have it come up the first time a user opened the application. But in case users wanted to get the walkthrough again, we decided to have another option where the user can request the walkthrough while the app is running at any point to get a refresher on the app's features.

4. What gamification features should ClapBack utilize?

- Option 1: Multiple minigames (i.e. minesweeper, tic tac toe, etc.)
- Option 2: No gamification
- Option 3: Simple gaming features (i.e. would you rather, etc.)

Choice: Option 3

Justification: We wanted a gamification system to add another element to the chatting experience. Yet we felt creating multiple minigames would detract users from the original

purpose of the application as well as for us developers, as more resources would be needed to create separate minigames, taking away from development of the main application.

5. What do we count as an ‘interaction’ for the day?

- Option 1: Just opening the application
- Option 2: Interacting with randomized friend for the day

Choice: Option 2

Justification: We want our gamification system to have an impact on the amount of days a user uses the app consecutively. To do this we felt it best that an “interaction” be counted if the user actually engaged in the functions of the app and not count simply opening the application for the purpose of keeping the ‘streak’.

Non-Functional Issues

1. What frontend language/framework should we use?

- Option 1: React Native
- Option 2: Angular
- Option 3: Android Studio with Kotlin
- Option 4: Flutter

Choice: Option 3

Justification: Android Studio has a strong SDK that supports libraries that allow developers to customize applications into anything they need. Moreover, it has the widely preferred IDE that is backed by IntelliJ. Such IDE will facilitate our process in development, debugging, and testing as it provides all the tools for them. Kotlin is easy to learn and has a big community that will help us navigate our ways when we get stuck. It also requires less code and increases readability.

2. What backend framework/technology should we use?

- Option 1: Spring Boot (Java)
- Option 2: Cloud Functions for Firebase (Javascript)
- Option 3: Flask (Python)
- Option 4: Node.js (Javascript)

Choice: Option 3

Justification: Flask is one of the most popular backend frameworks that is being used in the industry and it allows developers to add additional functionality as they're needed. Not only does Flask come with the largest package installer called PIP, which provides us with millions of packages we can use on our application, the core is kept simple so that it is easy to learn for

beginners. Since we will be using Firebase as our database, it will be convenient to use Flask as it has access to most of the Firebase features, unlike other backend frameworks. To implement our main chat feature, we can also utilize Flask-SocketIO that is strongly compatible with Flask.

3. What database should we use?

- Option 1: AWS
- Option 2: Azure
- Option 3: Firebase
- Option 4: Heroku

Choice: Option 3

Justification: We chose Firebase because it is a BaaS(Backend as a Service) that is prestigious among developers for providing useful cloud services. Most importantly, Firebase is highly compatible with Android applications that the Firebase Assistant tool is located within Android Studio. It uses a NoSQL-type database like MongoDB that is scalable and flexible in design. This web service provides an excellent dashboard UI to manage the database and will help us focus on the application itself to produce high-quality features in the long run.

4. What web service should we use?

- Option 1: AWS
- Option 2: Azure
- Option 3: Google Cloud
- Option 4: Heroku

Choice: Option 4

Justification: Heroku supports deployment of Flask applications. For a semester-long project like ours, it will be perfect to use Heroku as it is scalable and free. We can also create a pipeline between our Github Repository and Heroku Git Repository to frequently deploy by committing and pushing our code. Because Heroku takes care of configuring infrastructures, we will be able to focus on building our app.

5. What password-hashing function should we use?

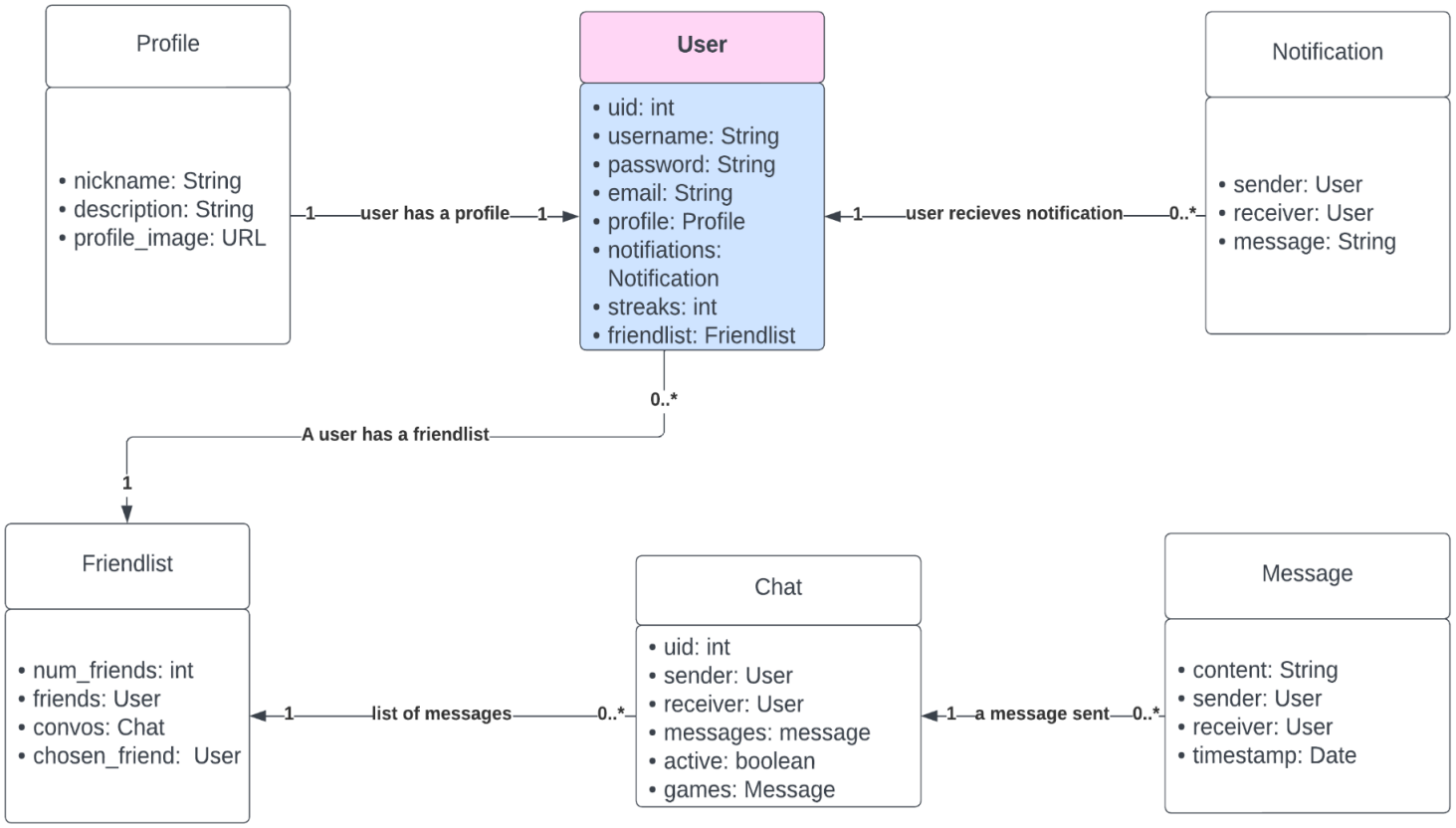
- Option 1: bcrypt
- Option 2: scrypt
- Option 3: SHA-1
- Option 4: MD5

Choice: Option 1

Justification: bcrypt is a password-hashing function based on Blowfish. Besides incorporating salts to protect against rainbow table attacks, bcrypt is an adaptive function that can avoid brute-force search attacks by increasing the iteration count to make it slower. This is a huge advantage due to the fact that even powerful computers will have a hard time hacking the password.

Design Details

Class Design



Description of Classes and their Interactions

The classes are structured based on an understanding of the types of objects in our application. Each class has its own list of attributes.

User:

- User object is created when someone signs up for our application.
- Users will have a unique id.
- Users will have a username, password, and email for login uses.
- Users can manage their profile settings.
- Users receive notifications when they receive messages.
- Users can maintain streaks based on how long they have used the app on consecutive days.
- Users will have a friend list that will keep track of User's friends.

Profile:

- Profile object is created when the User object is created.
- Each profile will have a nickname and the user can change it.
- Each profile will have a description where the user can add their favorite media or introduce themselves.
- Each profile will have a profile image that the user can upload an image.

Notification:

- Notification object is created when the user receives a message.

- There will be a User who will be sending the message and a User that is receiving the message.
- The notification will have the content of the message.

Friendlist:

- Friendlist object is created when the User object is created.
- Each Friendlist contains a number of friends the user has.
- Each Friendlist contains a conversation with friends which can be opened to see chat histories.
- Each Friendlist also contains a chosen friend who is the friend that is allowed to talk to that day.
- Each Friendlist will have a list of User objects

Chat:

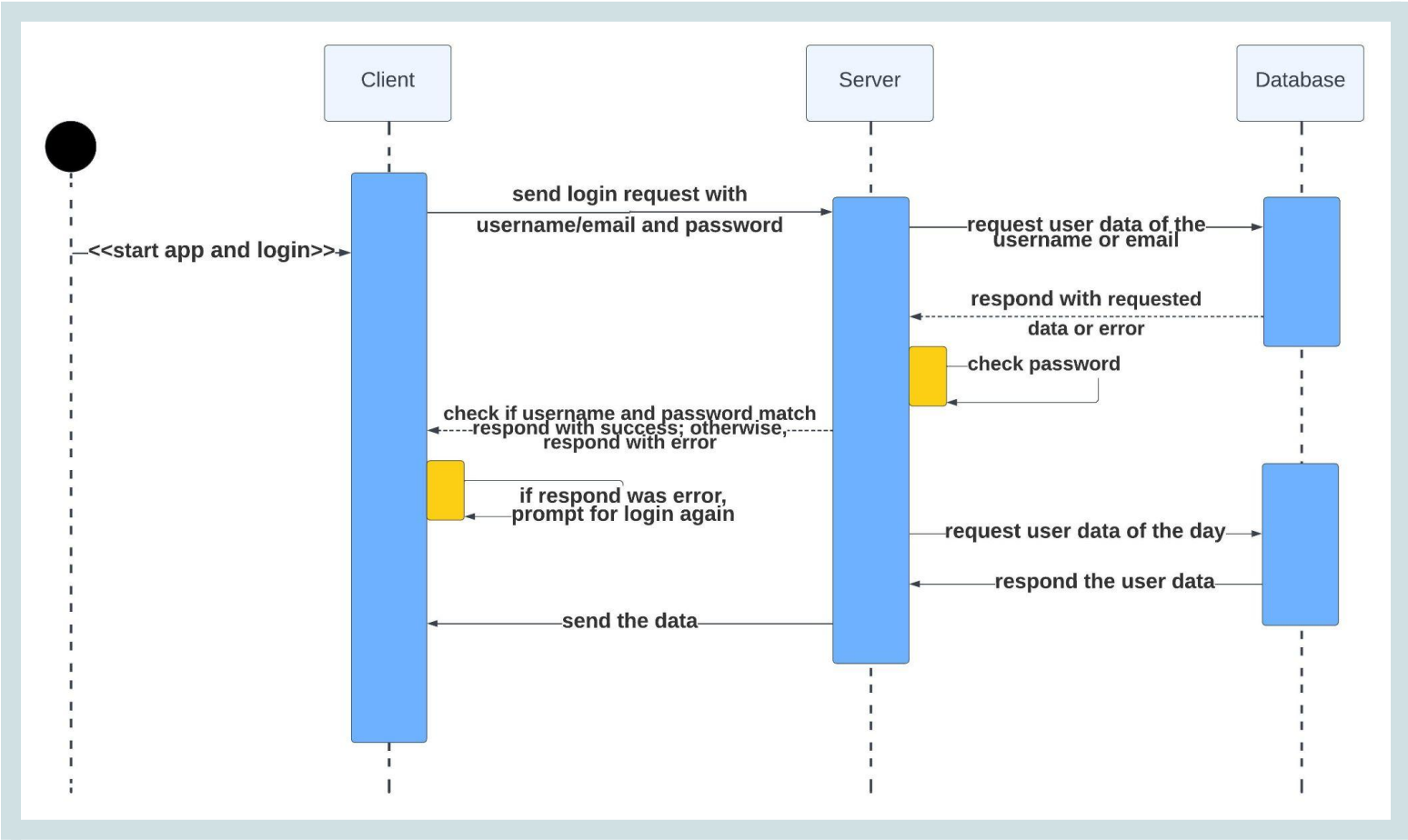
- Chat object is created when the User object is created.
- Chat will have a unique id in order to keep track of its location.
- Each chat will have a sender and a receiver to keep track of messages.
- Each chat will contain messages that have been sent and are going to send.
- Each chat will also be active or not. If the receiver of the chat is not the chosen friend, then the sender cannot send a message.
- When a chat object is created, users can choose to play in-chat-based games or not.

Message:

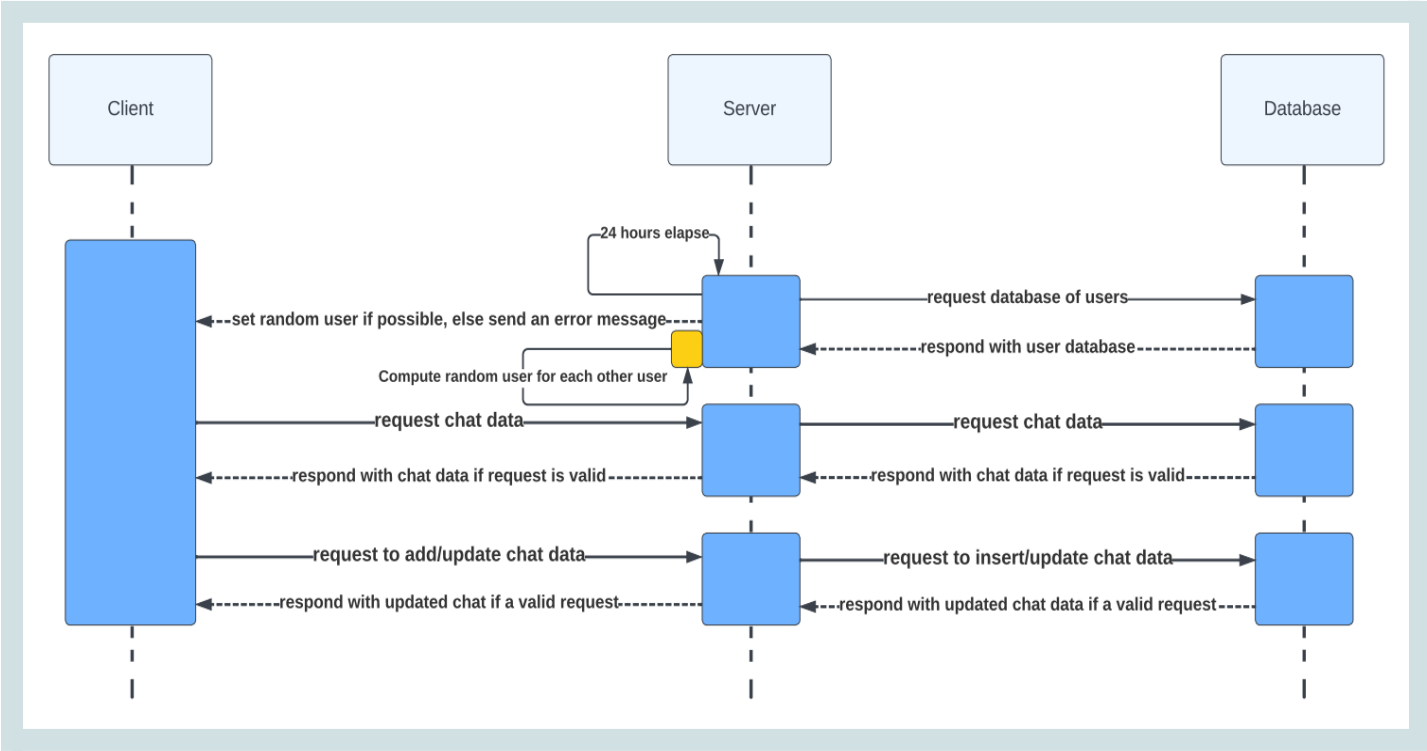
- Message object is created when the User sends or receives a message.
- Each message will have the content of the message.
- Each message will have a sender and receiver of the message.
- Message will have a timestamp that keeps track of when the message was sent and received.

Sequence Diagram

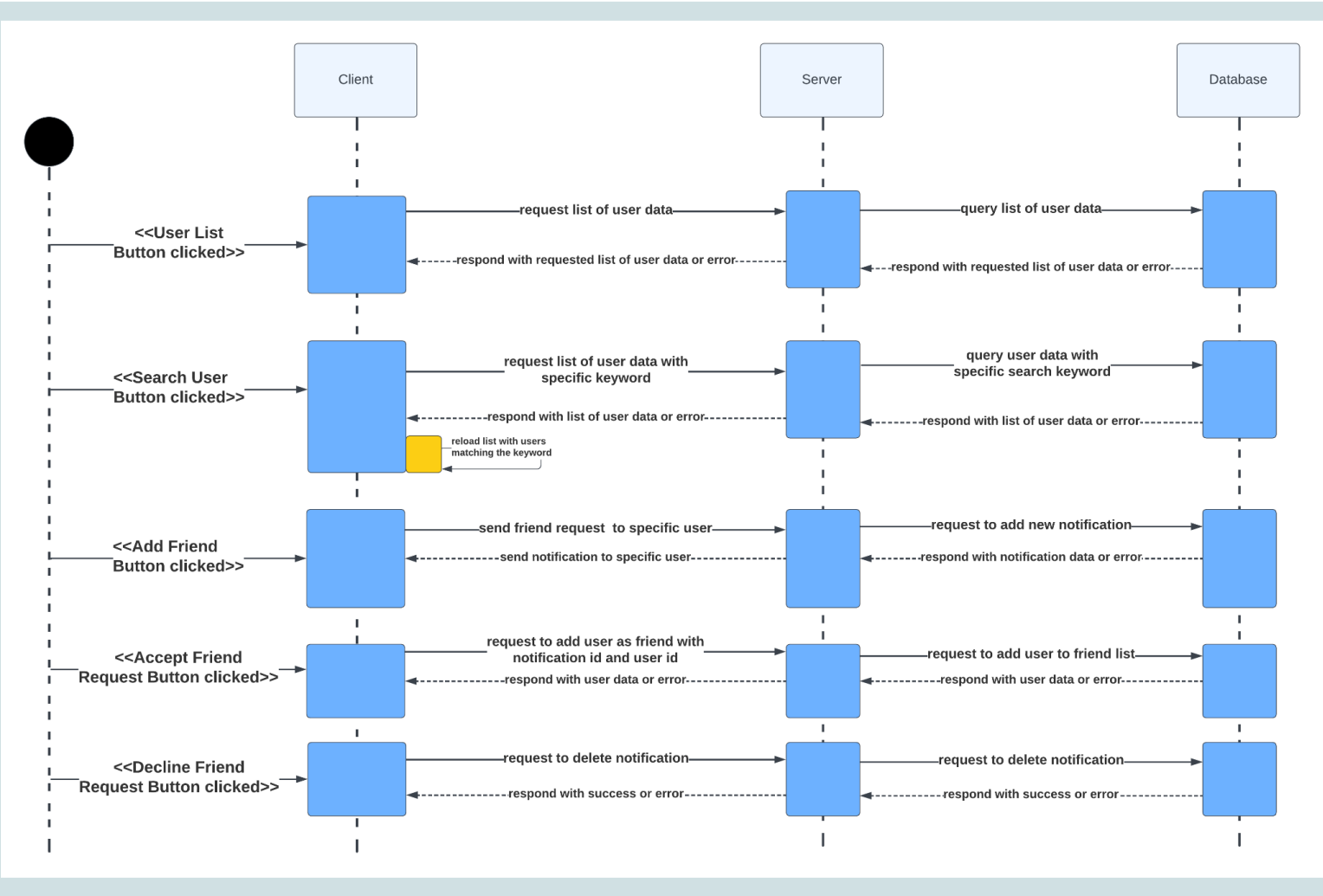
(1) When a user logs in



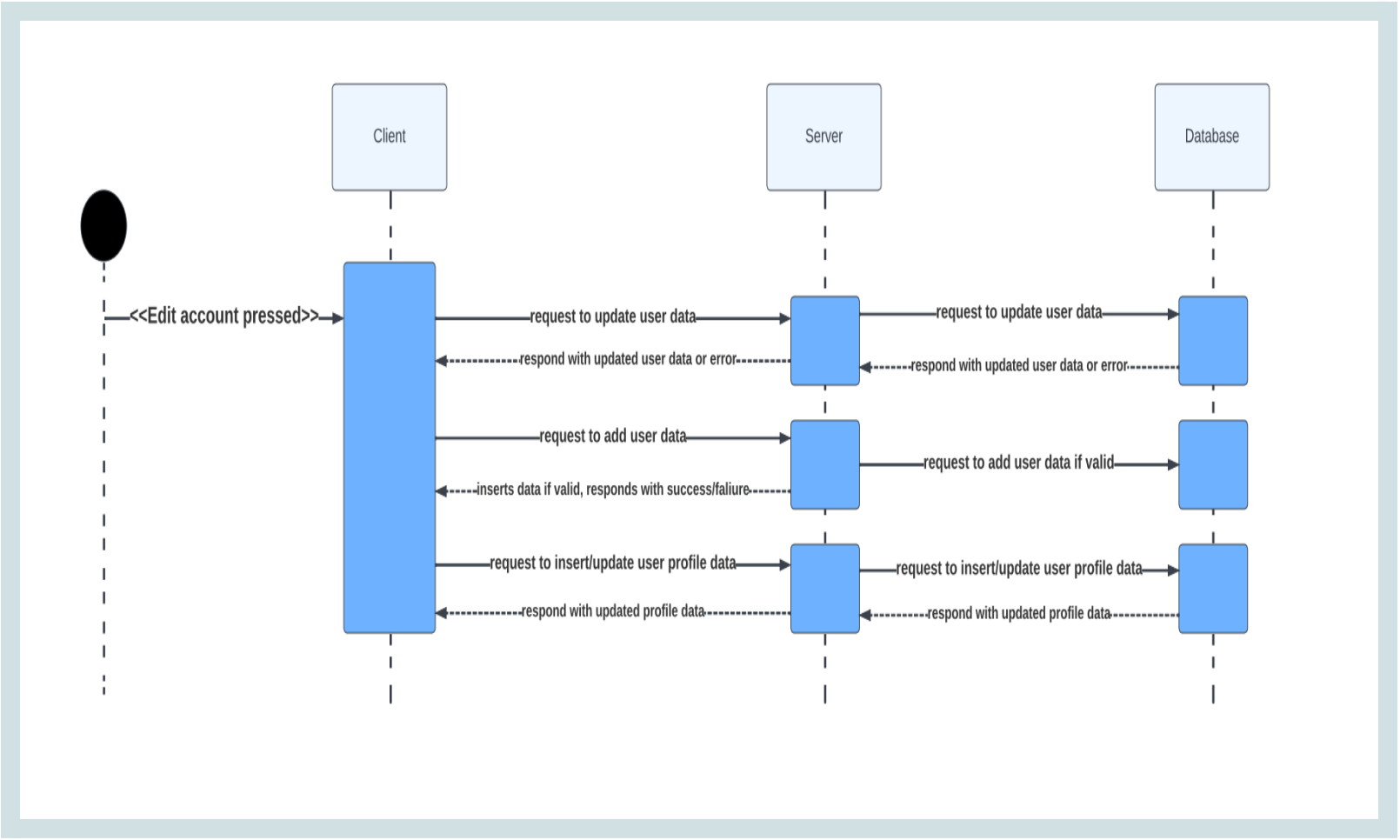
(2) When a user is generated and the user chats



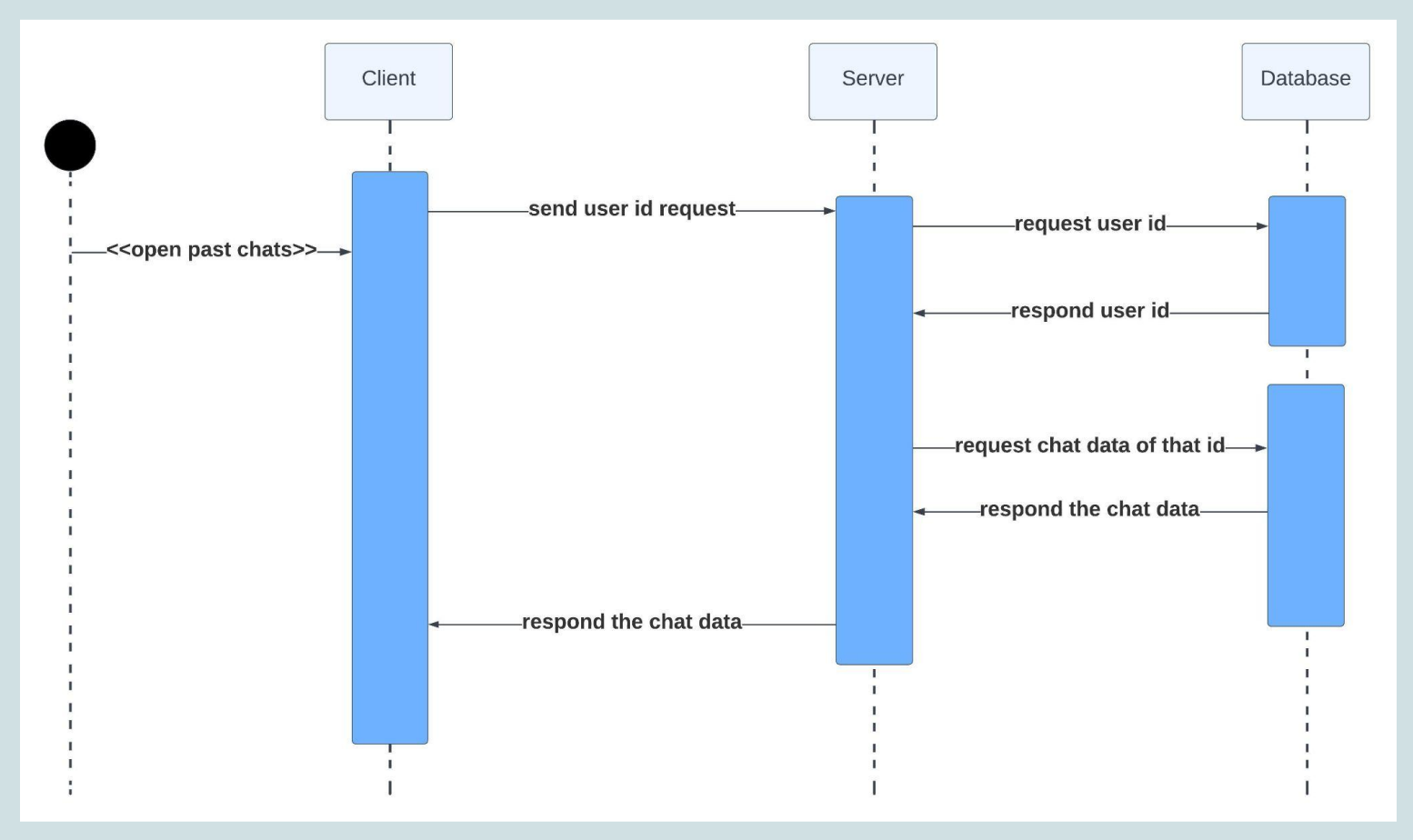
(3) When user adds a friend



(4) When a user edits their account

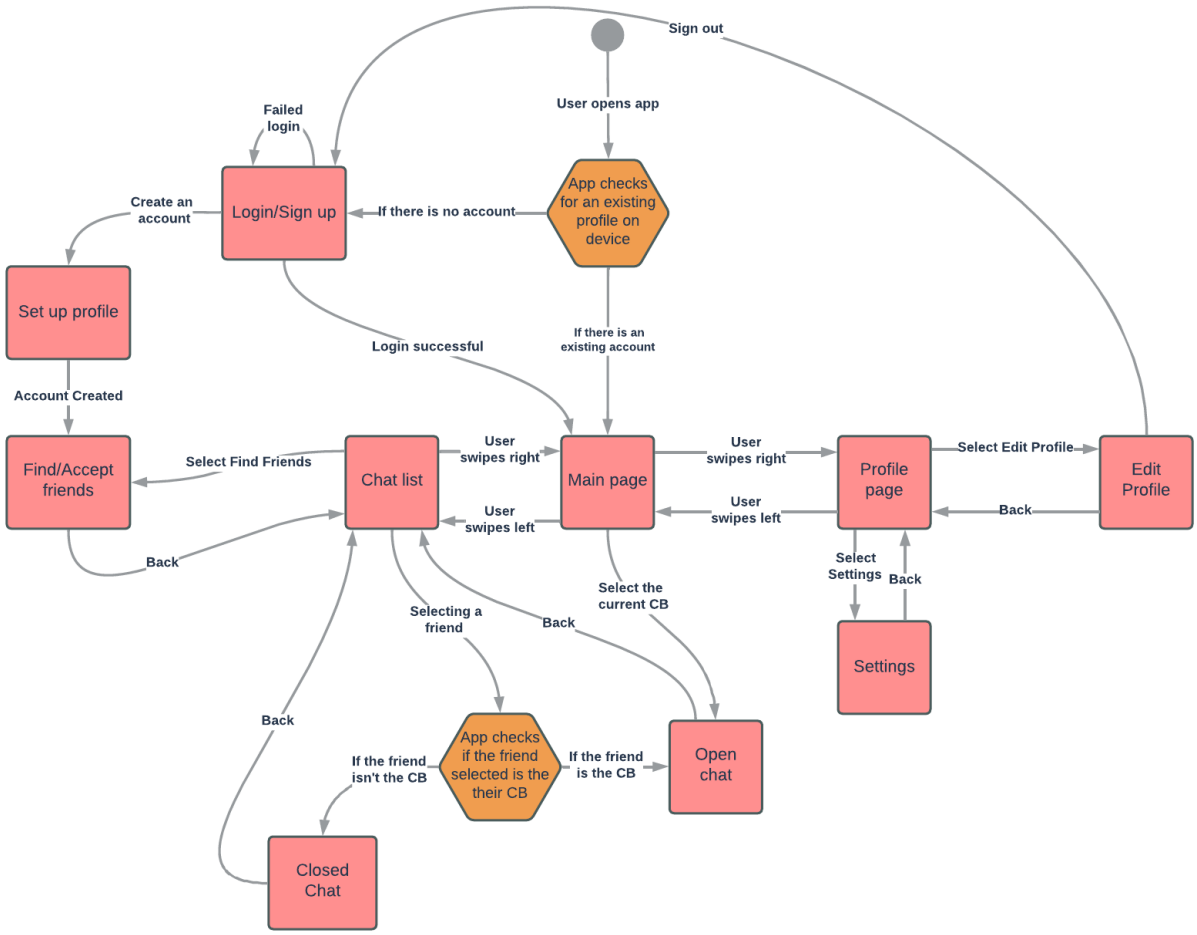


(5) When the user looks at past chats

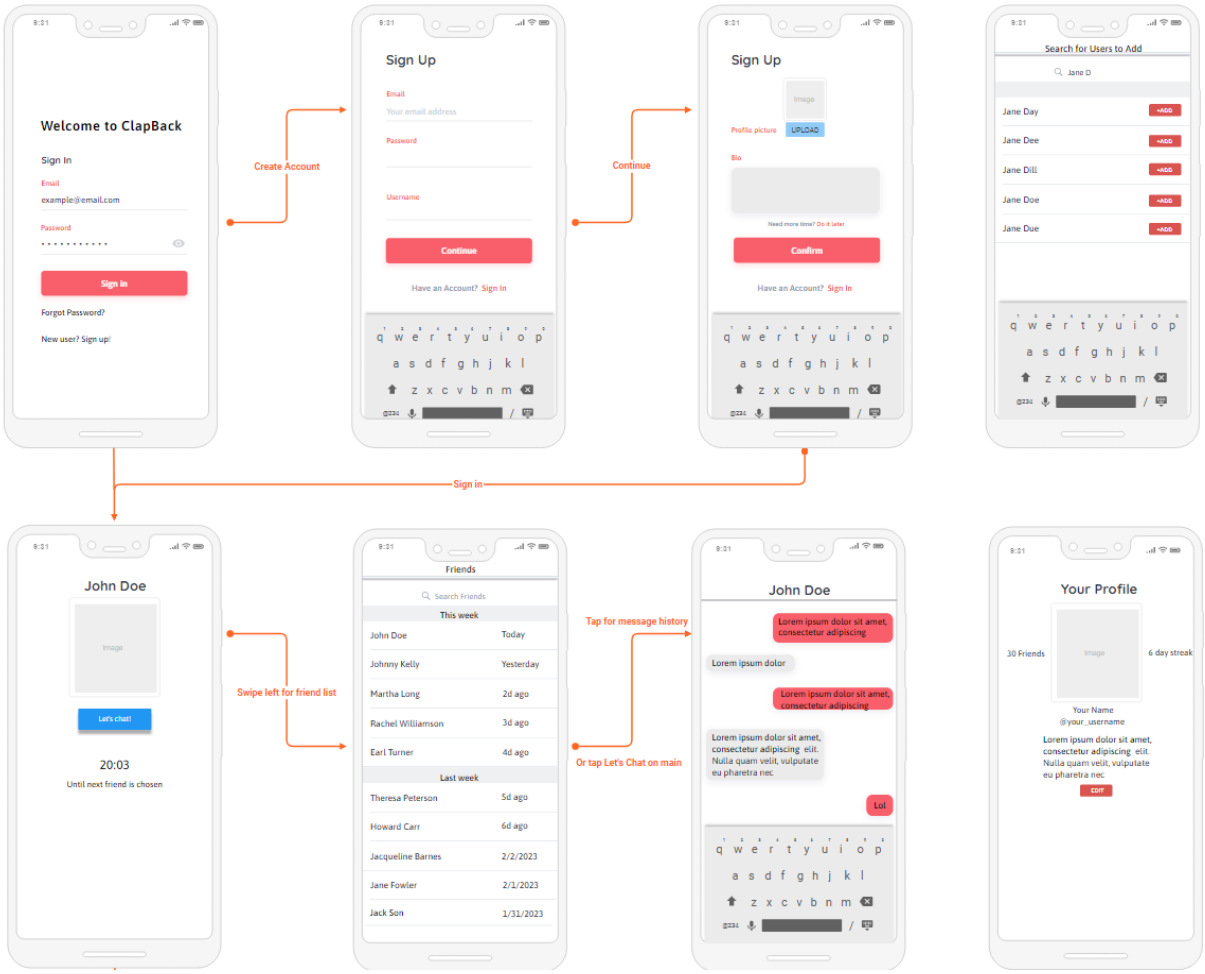


Page Navigation Flowchart

This is the general page flow a user will experience when using our app. At any page, a user can close the app, and reopening the app will start the flow over again. Our design is centered around getting the user to message their CB. A user's "CB" is their one friend they can message that day. When a user creates an account, they are immediately prompted to add friends so they can be given a CB the next day. A closed chat allows a user to see past messages, but not send any new ones. An open chat allows users to send messages and is reserved for the user's CB for the day.



UI Mockup



https://app.moqups.com/B5zKgR6XFotB2H3wFeIz0xTh33UxE09r/view/page/a880590a1

This is our general idea of what the app will look like. We want it to be very simple and intuitive, with as few pages as possible. The user will start at the sign in page, and has the option to sign in or create an account. If they create an account, they will be prompted to enter what information they want to display as well as a password and email for logging in. Once they fill out their information, they will be sent to a page to search for users to add as friends. This process is shown in the top four pages. The bottom four represent what a signed in user sees,

with the leftmost being the main page. From the main page, you can swipe left to see your friends list or swipe right to see your own profile. Also shown is what a chat room would look like in the app.