Team P: Oussama Rahmouni, Seul Eric Kim, Audrey Dockendorf
COMP 523.001

Application Architecture

How should we handle chats?

1. **Summary**
    a. In order to allow users to join and send messages in group chats, we will use the Stream Software Development Kit for messages in React Native.
2. **Problem**
    a. One important piece of client "desirada" is that app users (usually students) should be able to communicate with other users. We want to allow the creation of group chats, naming the chat, adding people by username to a group chat, and sending messages. Users should also be able to send a "direct message" to another individual. On the chat homepage, users should see all of their active chats. Because we don't have the time this semester to build a group chat app, we need to use some sort of chat API to help us out.
3. **Constraints**
    a. Needs to integrate with React Native
    b. Needs to be free
    c. Needs to allow for group chats and direct messages
    d. Must allow multiple chats per user
    e. Needs to have a customizable UI to adhere to clients color scheme
4. **Options**
    a. Stream SDK
        i. Pros: #1 Chat API (Widely used and trusted), offers React Native Software Development Kit, follows widely accepted Chat UI/UX designs including message previews, image uploading, chat page with all chat names, @mentions in chat, etc. Highly customizable, many tutorials available online, detailed documentation, claims to take only a few minutes to set up, multi-region support.
        ii. Cons: Pricing on website is unclear
    b. Firebase: storing each message and group details in DB
        i. Pros: We are already using firebase for data storage, so our architecture would be less complicated, Database is RealTime and can handle chats
        ii. Cons: More Work: Would need to design UI from scratch, would need to consider data model for storing messages, group chats, users, etc. More possibilities for things to go wrong, would be less efficient than leveraging a professionally-designed API. Firestore

Free Version is limited to about 1MB per document, or under 1000 chat messages.

c. Talk JS
    i. Pros: easily integrates with React Native, Claims to be “up and running within 10 minutes”, allows for 1-on-1 chats, group chats, and chat rooms, customizable design
    ii. Cons: out of our budget (billed monthly)

5. **Rationale**
    a. Our project will be using the Stream Software Development Kit for React Native. The group last semester who started the app made this choice, but we fully support their decision. Stream provides all the functionality that our app requires and integrates via an easy-to-use REST API. Based on the detailed documentation, debugging guide, contact information, and tutorials the Stream website provides, it appears that this option will be easy to get started with, and also get help with should we run into an issue. Our client provided us with specific UI designs and color palettes that we must follow when making our app, so the customizability of the Stream chat is very appealing to us as well.

<u>What should we use for the backend?</u>

1. **Summary**
    a. In order to store and retrieve data dynamically, we decided to use Google Firebase.
2. **Problem**
    a. Each user on our application will require their own storage and a lot of retrieval of that storage during use. Some of this data will change as time goes on, either periodically or manually. This means that this application needs a storage system that is capable of retrieving and changing data quickly and very, very often. The system also needs to be able to grow with the application's needs (more users mean more storage), while not being expensive.
3. **Constraints**
    a. The client wants this app to be able to scale to large numbers. We require a database that can scale with this demand of storage and data retrieval. A constraint for this portion of the project is cost. Data storage and retrieval can be very expensive, but our client's budget is relatively small. Another constraint is our specific use of the data. It will be manipulated many times by many different users, and it will also need to be able to deliver data at a moment's notice. A backend that will work with both iOS and Android is absolutely necessary.
4. **Options**
    a. Amazon Web Services
        i. Pros:
            1. User Authentication included
            2. Good customer service
            3. Can scale with demand
            4. Free tier
        ii. Cons:
            1. Less user-friendly website
            2. A smaller amount of free data
    b. Google Firebase
        i. Pros:
            1. User Authentication included
            2. Flexible free tier
            3. Well-documented
            4. Can scale with demand
        ii. Cons:
            1. Difficult to debug Promises
5. **Rationale**

a. We decided on Google Firebase after exploring its documentation, and comparing it with Amazon Web Services (AWS) scaling. Firebase documents the differences between its offered databases more clearly than AWS. Google Firebase is more user-friendly, which is comforting to our clients. Lastly, Google Firebase's higher tiers and prices are more appealing to our application use. AWS would have been an adequate choice as the two are very similar NoSQL databases.

What should we use for the frontend?

1. **Summary**
   a. In order to accommodate both iOS and Android applications, we decided to use React Native.
2. **Problem**
   a. We want each user to be able to use our application regardless of their mobile device platform. Android and iOS applications are developed by using either Java or Swift. It is time consuming to learn both languages and develop them accordingly and have the same functionalities.
3. **Constraints**
   a. One of the constraints for this project is the application needs to be compatible with a large number of versions of both iOS and Android. There are various operating system versions still on different devices plus many more versions to come.
4. **Options**
   a. React Native
      i. Pros:
         1. Open Source
         2. Javascript that all of the team members know
         3. Rapid Approach
      ii. Cons:
         1. Some libraries may get deprecated
   b. Flutter
      i. Pros:
         1. UI are customizable
         2. It supports its own visual when native components lacks
         3. Well documented
      ii. Cons:
         1. Framework is still pretty young
         2. Project files are larger compared to other tools
5. **Rationale**
   a. We decided on React Native after realizing everyone knew how to program in Javascript. There might be features and libraries that may get deprecated in the future, it may be preventable with little bit more research and consideration. Flutter is fairly newer and may not have all the features we want and need.