CM3050 Mobile Development Final Project
React Native Chat Application

By Garrison Jabs

Table of Contents

Introduction

With the busy lives most of us currently live, it is very easy to forget to engage with the people around us. It also creates a situation where even received communications can be forgotten in the noise, causing us not to respond, until it is too late, if at all. However, with the near perfect memory of technology, this provides an opportunity for solving this issue. Therefore, the proposed concept for this project is:

- A Messenger Application that would allow individuals to send instant messages, video chat, play games, and set reminder notifications to alert them of messages that had been received, but not yet responded to.

As the market is awash with similar apps such as “WhatsApp” and “Facebook Messenger” to name two of the biggest, it can be very difficult to stand out as a better application. However, a shortcoming of most apps is the inability to allow the user to set remind notifications of messages received. The apps typically only alert the user of a received message.

The main problem this solution is targeting was twofold:

- Engaging with friends, family, and coworkers over distance
- Reminding users of received messages that may need a response

Design/ Concept Development

Statement of Feasibility

The scope of this project is ambitious, as there are significant structural components involved with creating a messenger application alone, such as:

Backend Components

- A database for managing:
    - User information
    - Chats
    - Messages
- A media storage server for managing:
    - Images
    - Files
- REST API for managing:
    - User authentication
    - security

Frontend Components

- A Graphical User Interface to allow users to easily engage with the application
- Client Engine for managing the data requests involved with sending and receiving messages
- Application services

With the available production period being 10 weeks, not every portion of the application could be developed from scratch. Therefore, a canned solution of “Firebase” was selected to use as the Backend component. Firebase was chosen because it is a hosting service produced by google, that provides a cloud-based API connection to a:

- REST server
- NoSQL Realtime database
- Media storage

Furthermore, this service is provided at several access levels with the most fundamental being, a free subscription tier for development. With a solution for managing the server-side components sourced, the project falls within a reasonable scope that should be manageable within the available time frame.

Project Considerations/Plan of Action

Because two of main concerns when developing an application such as choosing a programing language, and/or development platform were dictated by the course material. The only other main concerns were:

- Designing a UI that is efficient and easy to use
- Correctly setting up the connection to the backend API to work with EXPO
- Revaluating if the scope was still too ambitious

Addressing the first concern should be simple. Because chat applications are common, this creates an opportunity to draw inspirations from common patterns of application flow. In turn, this makes it very easy for a new user to pick up and use the application as they likely have previous experience. The second concern should also be quite approachable. Firebase is a well-established service provider that has good documentation with an active community of users. This provides many resources for solving issues that may arise. The last concern is the hardest to know beforehand, whether it will be a problem or not. With that in mind, a development plan was created that would allow for a more agile approach. This resulted in a breakdown that established what was required for a Minimal Viable Product (MVP), and which components could be dropped, due to viable scope shrink. The following lays out the component priority list, with 1 being the most important and 4 being the first component to be dropped.

1. MVP (Messenger Service fundamental to the project)

- User management
    - A way of creating a new user
    - A way of validating a user to give access to the application
- Chat Management
    - A way of creating a chat
    - A way of joining a chat
    - Sending messages
    - Displaying messages

2. Notifications and Reminders (Second Priority)

- Message tracking
    - Data value indicating if a message has been read
    - System for updating the values
    - Push notifications
    - Data value for setting a reminder notification

3. Video Chat (Third Priority)

- Video management
    - Connecting the data streams from the two endpoints
    - Initiate call/end call

4. Play games (Fourth Priority)

- Game management
    - System for managing moves between players
    - Scoring
    - Game rules

With the three main development concerns addressed, the project began in earnest. An early design decision for the application was to control the availability of access to authorized users only. This is because most individuals place a high value on the privacy of their personal conversations. It was then required to think about user flow design, or rather how a potential user would interact with the application. This is an important consideration, as a poor flow that is over complicated can be a major impediment to the adoption of new applications. Furthermore, with so much availability of alternate solutions, even small pain points can be enough to cause users to abandon applications entirely.

Application Flow

The design decision to group users into two fields authorized and unauthorized, dictated that there would need to be user flow considerations for both. This then resulted in the Login/Signup flow design seen in figure 1 for an unauthorized user.

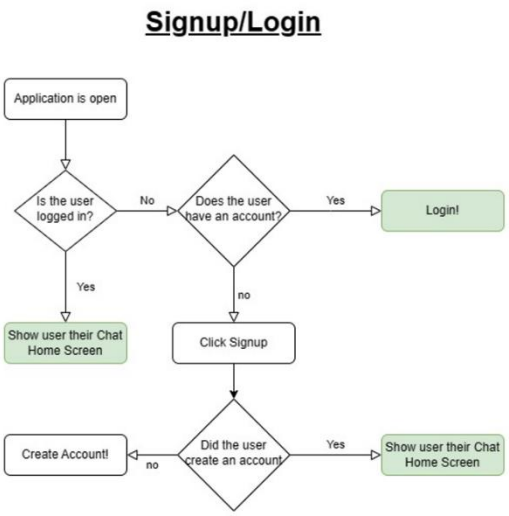

Figure 1: User flow pattern for Login and Signup

The assumed decision/action path of an unauthorized user indicated that a good user flow might use two screens to encapsulate the action of logging in a current user and signing up a new user. For authorized users, the flow considerations were more complicated as a user would have more available decision/action. The resulting flow based off of experience with other similar applications and other assumptions can be seen In Figure 2.

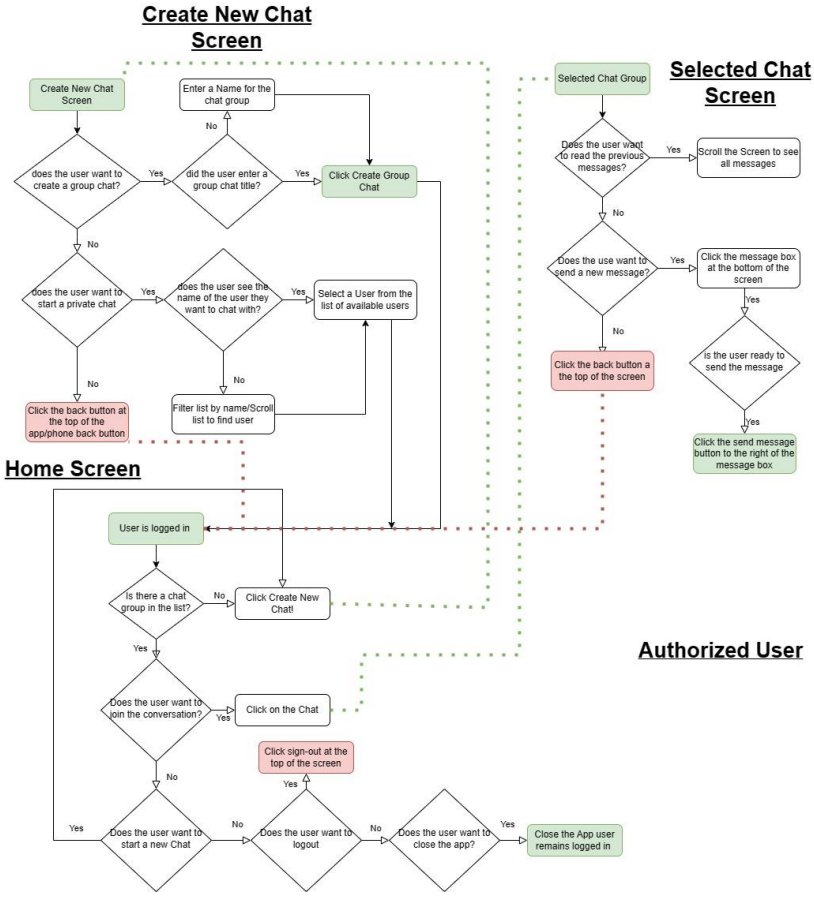

Figure 2: Authorized User flow pattern

By exploring the potential application flow, it became apparent that a series of 5 modules would be ideal to create the MVP. The resulting modules were as follows:

- Unauthorized User
    - Login Screen
    - Signup Screen
- Authorized User
    - Home Screen
    - Create Chat Screen
    - Chat Screen

Wireframing/Prototyping

At this point I began creating a mid-fidelity wireframe of the application. This was to begin testing my early assumptions about its ease of use, the flow of the application, and the design choices made. I also used this opportunity to gather some early feedback about the application.

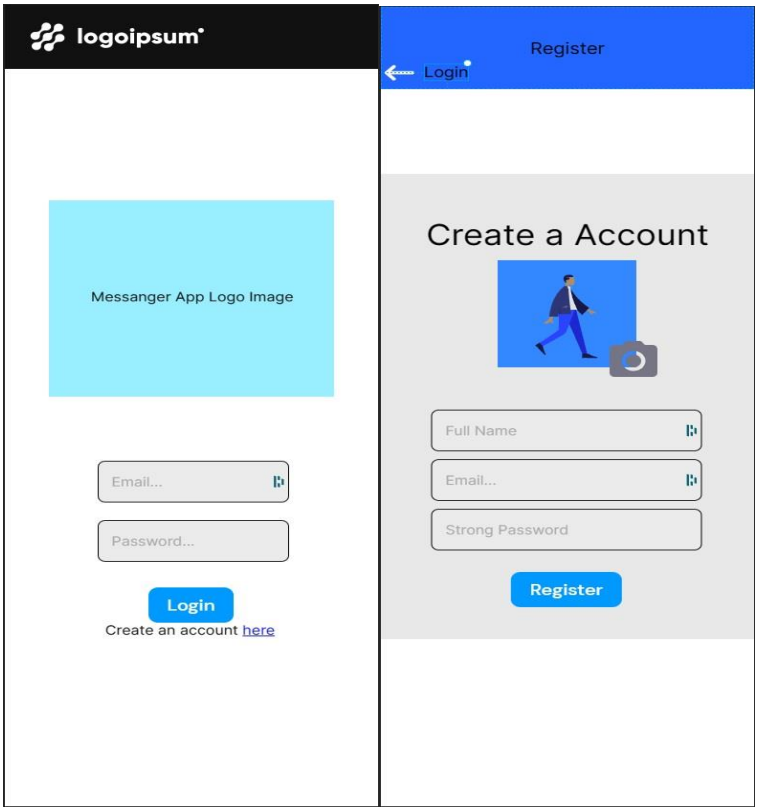

Figure 3: Mid-fidelity Wireframe of Login and Registration screens

Figure 3 shows the wireframe design of the Signup/Login screens and is a fairly standard model of the now ubiquitous screens used in most applications. All users asked to mock the experience of logging in or registering, had no trouble understanding the task and generally commented that they felt the format was pretty standard.

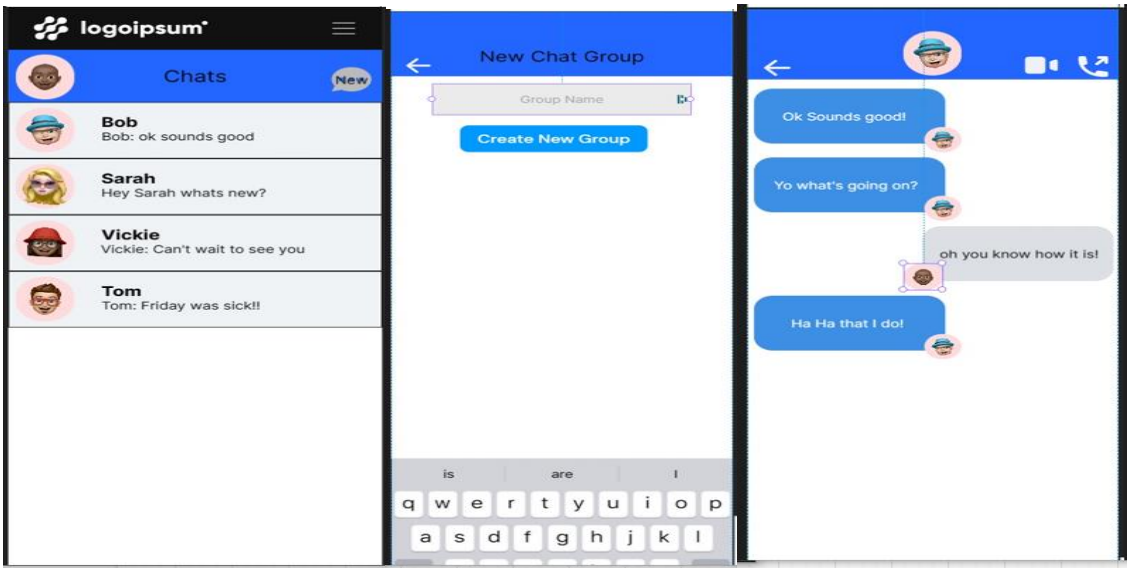

Figure 4: Mid-fidelity Wireframe of Home, New Chat, and Selected Chat screens

Figure 4 shows the wireframe design of the screens accessible to a logged in user. Again, when users were asked to mock the experience of using the application, most users intuitively understood the task as the design was similar to other applications that they had used. However, it was commented that the color scheme used in the wireframing was very similar to Facebook. As a result, a different color scheme will be used for the development itself. After testing the application flow and wire frames with users, it was felt that development could begin, as many of the assumptions were verified.

Development

Using Expo and React Native, an initial blank project was initiated to begin work on building the chat application. React navigation dependencies were then added to facilitate transitioning between different screens. Additionally, 4 JavaScript files were also added that would later become the other screens in the application. The JS files were then filled with some basic components such as <View> and <Button>, styling and navigation logic to build a very basic prototype of the application flow.

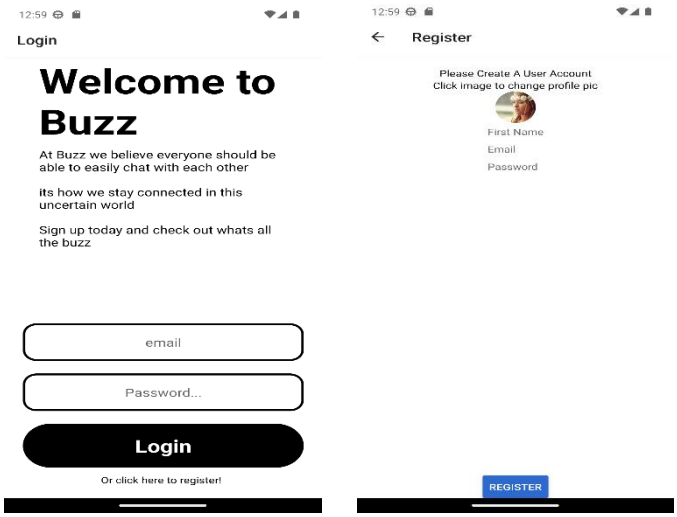

Figure 5: Initial Prototype of login and registration screens

None of the backend logic to connect and engage with firebase was created at this time, as the main thing being tested was the decision/action process through the application. Early tests proved that the flow functioned as expected, and so it was at this point that Firebase was added to the build. There are two versions of firebase that work with EXPO. However, EXPO documentation recommends using the JS SDK version when developing with EXPO GO, over React Native Firebase. This was the reasoning for adding Firebase JS SDK (9.15.0). As a side note, this did, however, end up causing a significant problem later on. The platform for hosting and emulating EXPO projects known as "EXPO Snacks", does not yet support versions of Firebase beyond "8.X.X", while EXPO GO does. Furthermore, the changes to the Firebase function calls have significantly changed between versions. This was only discovered after a significant amount of work had been completed, while trying to test the application with EXPO Snacks. That being said, the problem was not unsurmountable, as the active community had run into this problem before and offered guidance. The final version of Firebase eventually used was version "8.10.0". The needed firebase dependencies were installed, and a firebase account was created to gain access to the API keys. The connection to Firebase was established in a config file, which allowed creating/authenticating new users, accessing the real-time database and file storage as seen in Figure 6.

```
if (!firebase.apps.length)
{
  firebase.initializeApp(firebaseConfig);
}

export const auth = firebase.auth();
export const database = firebase.firestore();
export const storage = firebase.storage();
export {firebase}
```

Figure 6: Initialization of Firebase 8.10.0 and features

As it was now possible to identify authorized and unauthorized users, the logic to separate the navigation flows of both user types was built. An additional 4 JS files were added to the project, two files holding the different stack navigation logic. One for holding the logic for selecting the correct navigation stack based off login status, and another for separating the login screen from the main App.js file. This was done so that the "App" file was only rendering the needed page. The logic for selecting the correct navigation stack worked as such; when a user is signed into the app, Firebase returns a file containing information about that user. By checking if that file exists and storing that with a useState(), the selector logic returns the correct stack using a turnery statement, that updates with the change of user login status (as seen in Figure 7).

```
export default function RootNavigation() {
    const { user } = useAuthentication();
    return user ? <UserStack /> : <AuthStack/>;
}
```

Figure 7: Navigation Stack Selection Logic

With the navigation fully set up, it was now only a matter of attaching the “sign in” and “create new user” logic to their respective screens. This was fairly easy, as the Firebase API provided functions for doing this. Furthermore, I had chosen to authenticate with email and password over third party authentication, which meant only dealing with the firebase API. It was at this stage of construction where building the messenger logic began. The main chat page should show an instance of every chat room available to the user as seen in Figure 8.

Figure 8: Main Chat screen

This meant that when iterating through the available chats, there would be many repeated function calls. Due to this, I decided to create a custom list component that I could pass the chat properties. This built a uniform list where each instance was a clickable button, that navigated to a unique room. After testing the output with an array of static data, it was time to build the “New Chat” screen, which would allow a user to create new rooms. In creating this screen, I had made a poor assumption that each room would be private. This assumption was based off of the fact that Firebase creates unique ids for each document. However, I had forgotten to create specific unique IDs linked to the specific users to make that true. This resulted in every room being allowed to receive messages from all users. While this was unintended behavior, it ended up a case of fortunate misfortune, in that I had created a group chat feature. To solve the issue of private chats, I created a new id, by joining the two user's unique ids, as well as adding a reference to the stored list of available users. It was time to create the functionality that would allow someone to send and receive messages. Using a “useEffect()” function call and a Firebase listener, I was able to send and receive changes from

Firebase. The “useEffect” allowed the message component to re-render every time a change was detected. Which allowed messages to be displayed as soon as they were received. Furthermore, all a user had to do to send a message was to click send.

User Feedback

At this point the app had enough development to gather some user feedback. It was in this moment when uploading to EXPO Snacks, that I realized I had a problem with my Firebase dependencies. After solving those issues and gathering feedback, it was time to make some changes. While the feedback was generally good, it did indicate some significant problems. For instance, when signing up, a user was allowed to upload an avatar image to their profile. However, the application didn’t allow a user any means to change that image later. This was disappointing to the users. Another problem noted was that if the user list and chat room list grew too long, objects went off the screen. The final issue of note was users found it difficult to search for another user they wanted to chat with. This was because there was no search tool. Most other criticisms only involved style changes, such as adding space between components to reduce crowding on the screen. To address the concerns of objects growing out of view, a <ScrollView> component was used to encapsulate the objects, thus allowing a user to swipe through them. Additionally, a profile screen was added to the application to allow users to modify their avatar. Finally, a search function was added to the “create new chat” screen. This allowed a user to filter the list of participants by entered value aka “name”. After completing the changes, the application was run through another round of user testing, that included a prompt asking about features that were important to the users. The feedback gathered indicated that while the application was functionally sound, it was missing features that users wanted such as:

- The ability to send photos
- Message send dates
- The ability to search within a chat room for previous messages
- Seen Status indicator
- Online status indicator
- Dark Mode
- Notifications of new messages

However, with the unexpected issue caused by the Firebase dependency version, and the length of time it took to solve it. It was necessary to polish the current application into the deliverable, while making note of the additional features for future development.

Conclusion/Evaluation

I am proud of the application I have developed, even though it ended up not being the application I proposed. Originally, I had assumed that I understood what was needed for a chat application. This was an easy assumption for me to make because I used several in my life and didn't truly question my understanding. Due to this, I proposed a scope that was very ambitious, and this resulted in a large overall shrink of scope during development. Another assumption that affected my ability to develop this project was assuming that EXPO Go and EXPO Snacks were the same. If I were to start this project over, I would try to better understand the differences and shortcomings between the development environments I was required to use. This would be in order to avoid issues of incompatibility that significantly affected my development time. I would also try to gather more information on features users wanted within an application, to better understand the product I was trying to develop. When reflecting on the question "have I built a better application than what is already out there?". I have to admit that I have not. This is mainly due to the fact that the application by comparison is not fully fledged. In order for my application to be competitive, more nuanced features are required to be implemented. Features such as push notifications. However, given more development time I feel that the application I started could be, at minimum, a comparative product. With that being said, the application is a messenger application. Users can sign up, login, send new messages, read the history of previous messages and create both group and private chat rooms.