Chat App

Brandon Fabie

Overview

- Chat app is a mobile application that allows users to join the chat room and send messages to other users. Users can also send pictures and videos and share their own location. The app also features an offline mode, where although users are not able to send messages, users can still view previous messages.

Problem

- With the addition of the color palette, users can be creative with the way they are able to view their own interface instead of keeping a generic chat interface. Because of the offline mode feature, users may be able to view older messages. This is important because if there is essential information that needs to be accessed in the previous chat, users can still log in to see all chat history. For example, if a friend had sent you their address as you are on your way to their house but you lost signal on your phone, you are still able to access chat history to retrieve the address.

Process

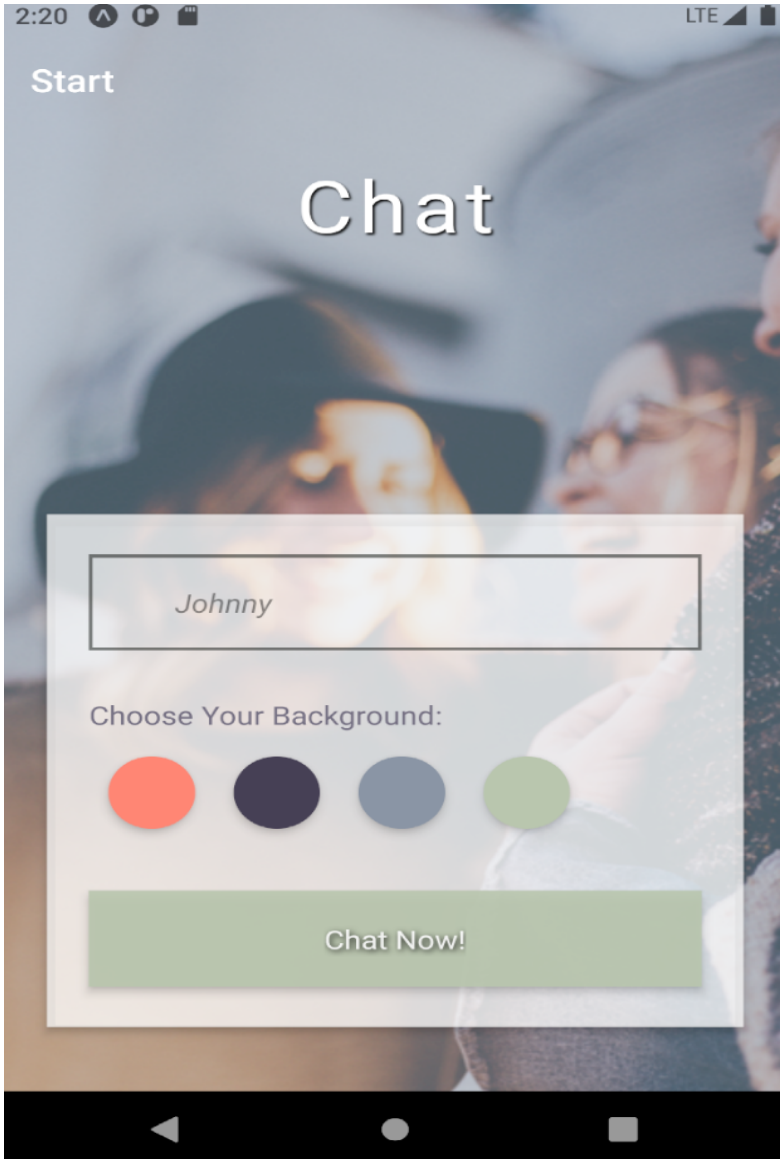
My first step in building a Chat App was to design and create the user interface and add the functionality of entering the chat room. There's a total of two pages the App has to maneuver between, the Start page and the Chat page.

Tools Used:

- React-Native
- React-Native-Gifted-Chat
- React-Native-Async-Storage
- Firebase/Firestore
- Expo permissions/ image picker/ locations/ media library/ camera

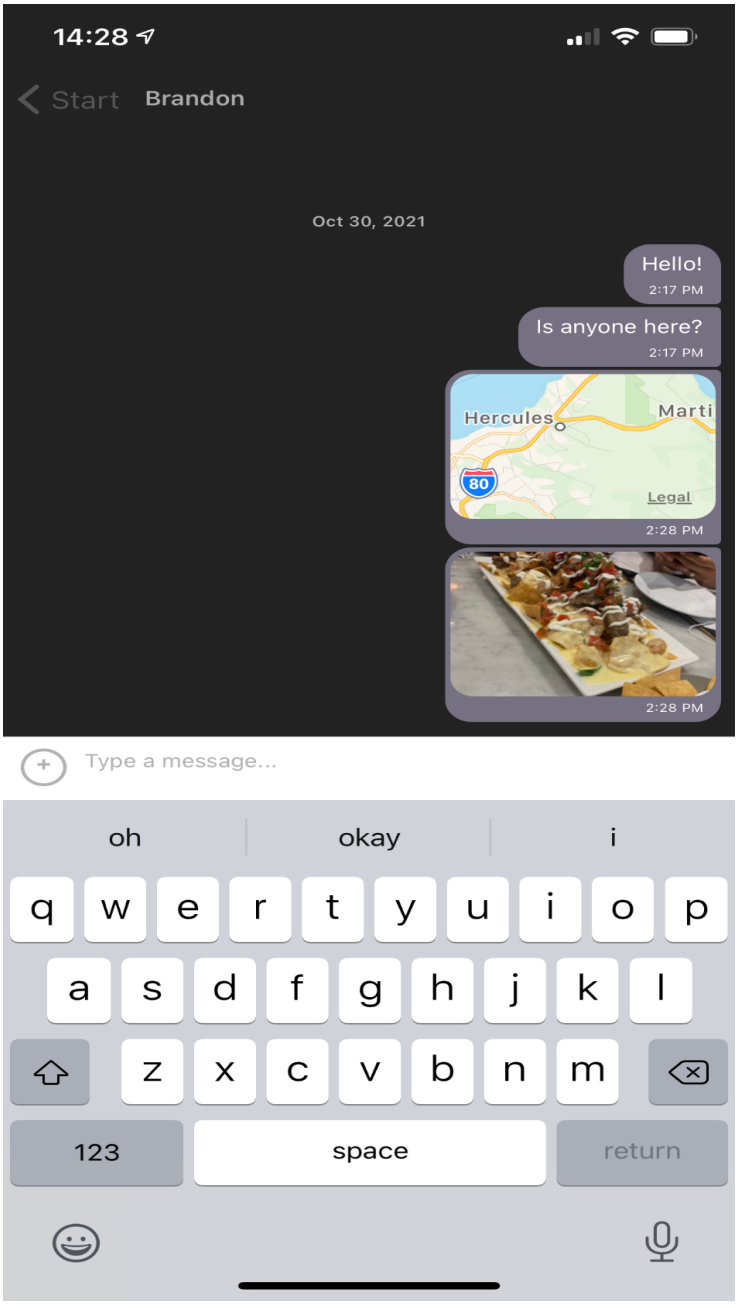
Start Page

This is where users can enter their own personal username and can choose the color of their text bubble from the choices presented.



Chat Page

This is the chat page where chatters can send messages, pictures, video, or share their location.



I utilized the tool “react-native-gifted-chat” to help build the Chat page interface. Gifted chat is a react-native library designed with the necessary tools to build a chat UI for cross-platform chat apps.

I also used expo permissions, image picker, camera, locations, and media library. My chat app always asks permission before accessing features such as the camera, location, or media library. Once given permission, users can share their location with others in the chat. Users can also choose a picture or video from their media library to send or take a new picture to send with their camera.

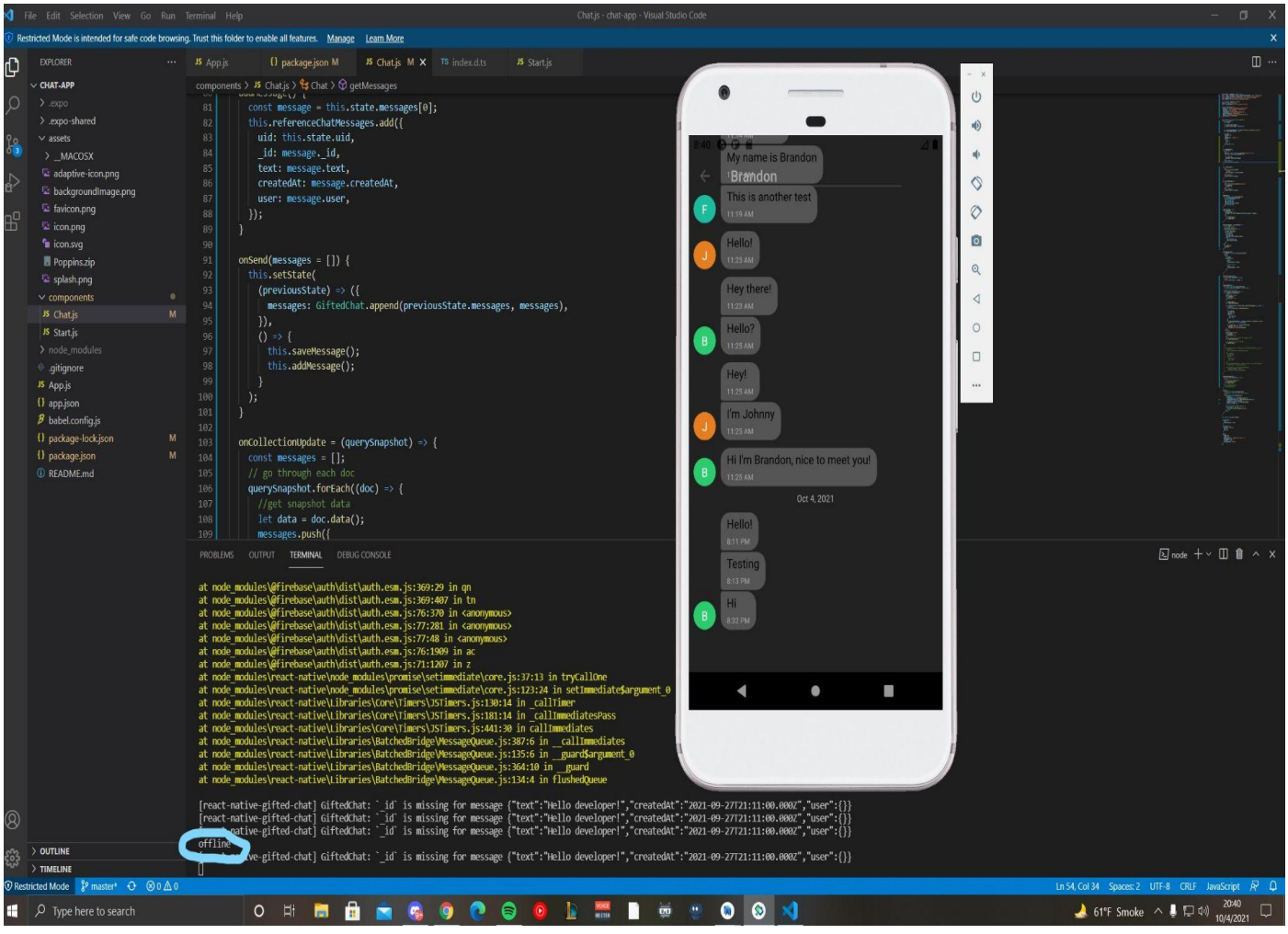
Once I was finished with the UI of the Start page and Chat page, the next step was to create a database to hold the messages and media that was sent from each user. For this, I used Firebase's Cloud Firestore and react-native's AsyncStorage library.

AsyncStorage is an asynchronous key-value storage system for React-Native and is the better option as opposed to LocalStorage.

Firebase's Cloud Firestore is a NoSQL database that allows you to store, sync, and query data for mobile and web apps.

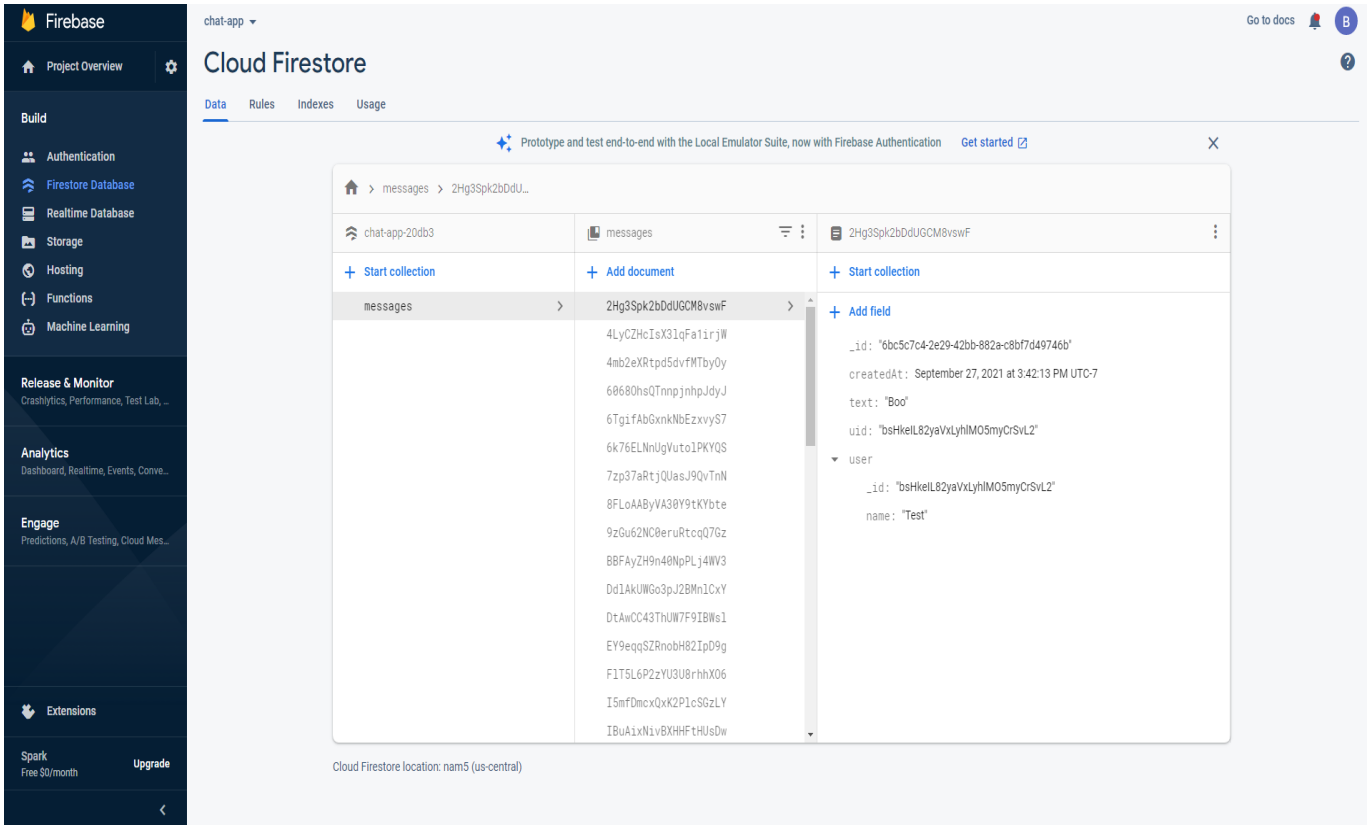
Offline Mode

You will be able to see from the screenshot below that according to the terminal, the application indeed is in “offline” mode. The user is still able to see chat history, but is not able to send any messages. The message bar and keyboard are no longer visible.



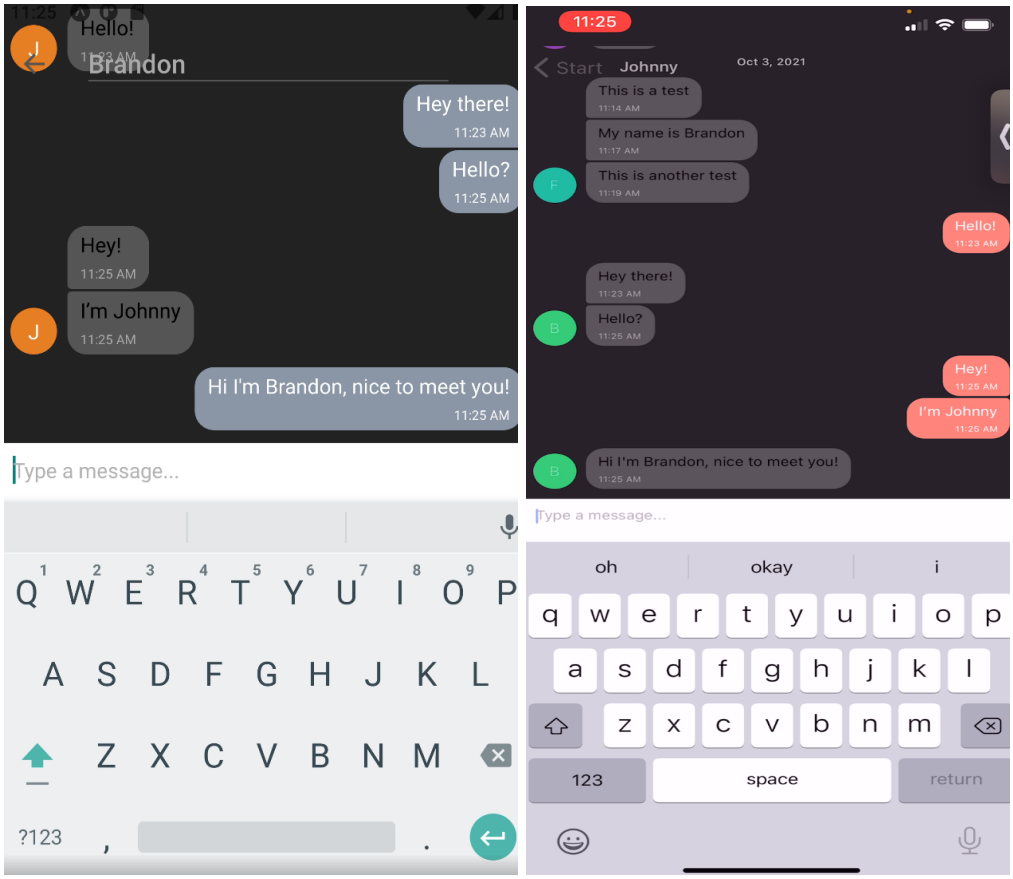
Cloud Database

This is where all of the messages are stored in order to view previous messages. In order to store data, I created a collection called “messages”. Every time a message is sent, a new document is created within the collection. Each document has its own unique ID with other keys being stored such as when it was created, the user and its ID, and the message itself.



Different Perspectives

Here are some perspectives from two different devices. On the left is user “Brandon” and on the right is user “Johnny”. You can follow along with their conversation.



Conclusion

Overall, I believe my Chat App was a success. I was successfully able to learn React-Native and how to create a cloud-based database with Cloud Firestore. This was my second time working with back-end coding, the first being another project that involved creating a movie database, my own API, and building the UI of the website.

The most challenging part of the project was implementing the back-end (Cloud Firestore) of the application. In my first try, I was able to store data into the collections. However, I failed to query the data and display it back onto the application successfully. Turns out, the issue was a simple typo with the back-end implementation. What I've learned with dealing with anything back-end is to ALWAYS make sure all of your endpoints, props, etc. are consistently named and called, avoiding all typos and confusion!

Next time...

I do want to improve on this application in the future. I plan to make this chat app more unique. There are already plenty of options for chat apps available, so finding something that makes mine stand out would be incredibly important. The first idea I have for that is to add a collaborative calendar for the application. We all know how difficult it can be to find a time to meet up with friends or family. I want to create a functionality to add your own personal calendar to the chat calendar while also being able to see everyone else's calendar at the same time. The app will then intuitively show a list of available days that everyone is free to meet up with, and then you can create a new event right then and there in the application!