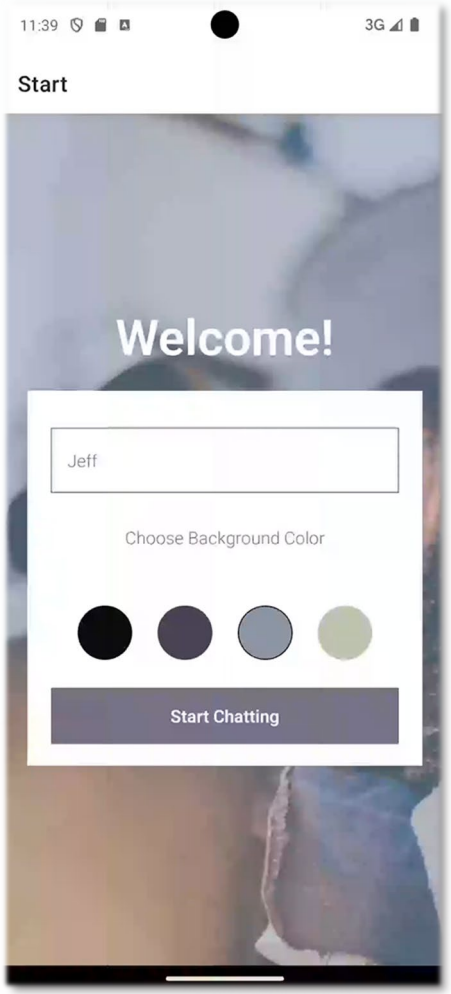
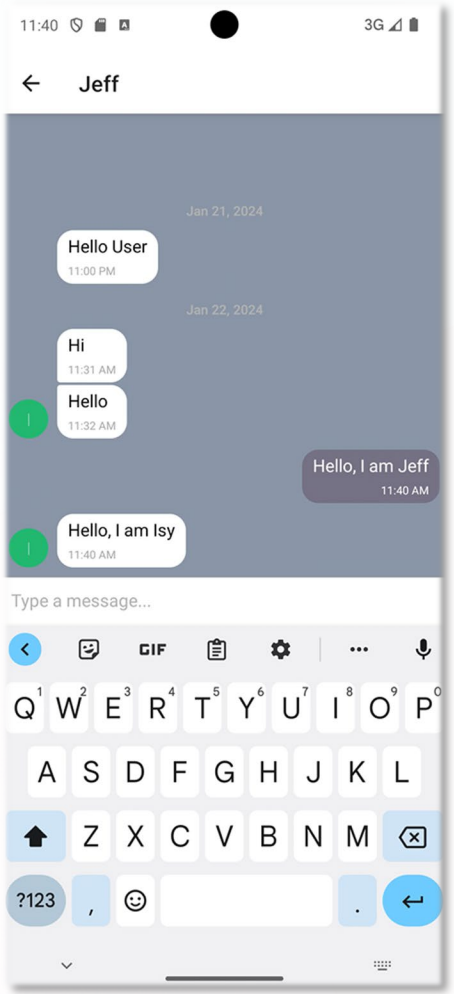




Case Study –

React Native Mobile Chat App

Isabel Matula

Overview

Objective

The aim of the project was to develop a mobile chat application using React Native, offering users a seamless chat interface along with functionalities to share images and location.

Purpose and Context

As reliance on mobile devices for daily tasks rises, so does the demand for native app development from companies. React Native allows developers to create high-quality apps for both iOS and Android using a single codebase, demonstrated in this project through the development of a chat app.

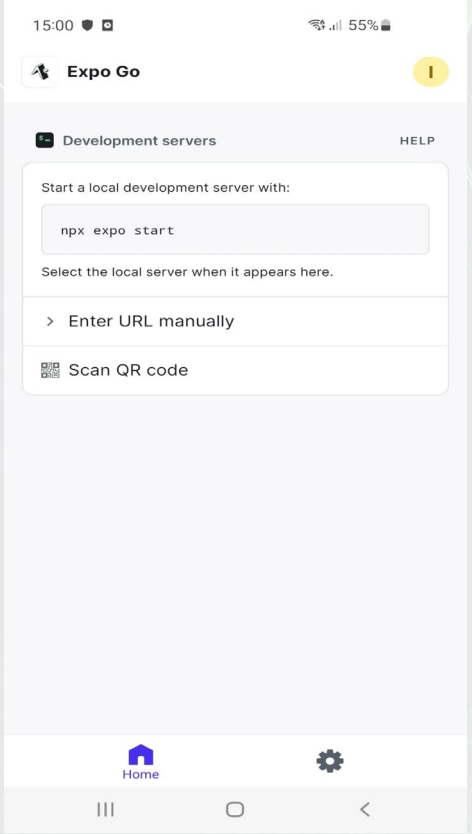
Tools

This project utilized React Native and Expo for mobile app development, integrating GiftedChat for chat interfaces, Google Firebase for real-time database and authentication, and various Expo APIs for accessing device features like image picking, media management, and location retrieval. Additionally, it incorporated react-native-maps for map display within the application.

Duration

The project was completed within a timeframe of 10 days.

Setting a React Native project using Expo



Expo Go app on phone.

Setting up Expo environment

Setting up the Expo environment was essential for seamless testing across various platforms, on both physical devices and emulated environments. This step involved configuring Expo Go for mobile testing and setting up an Android emulator for desktop simulation.

Setting up React Native project

The next step was to set up a React Native project using Expo and to establish the structure of the chat app project. Additionally, I familiarized myself with various React Native components, including TextInput, Button, Alert, TouchableOpacity, ScrollView, View, and Text.

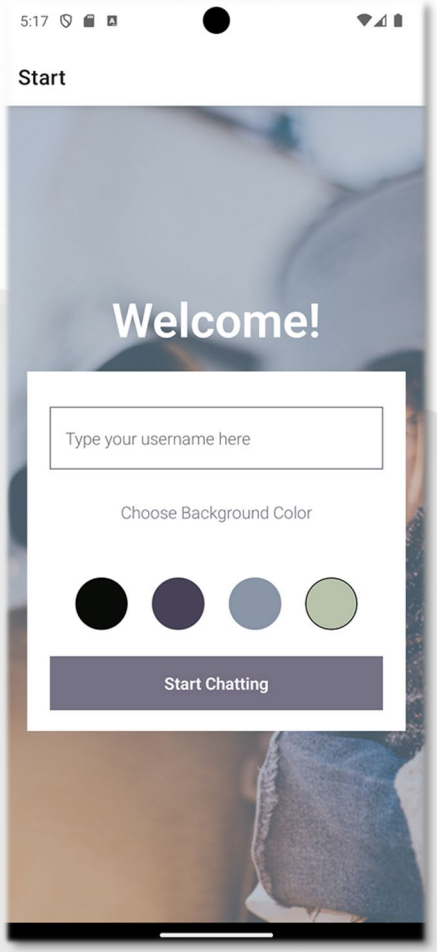
Application Layout

Navigating Between Screens

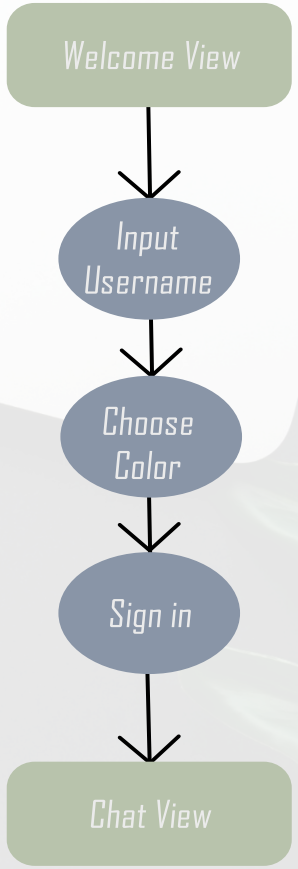
To be able to navigate between different screens within the application I imported the react-navigation library, using App.js as the root component. Then, I added navigation between the welcome and the chat screens.

Welcome Screen & Chat Screen

Inside the welcome screen the user can input their username and select the background color for the chat screen. The integration of gifted-chat library simplified the creation of a user-friendly chat interface and enabled the messaging functionality.

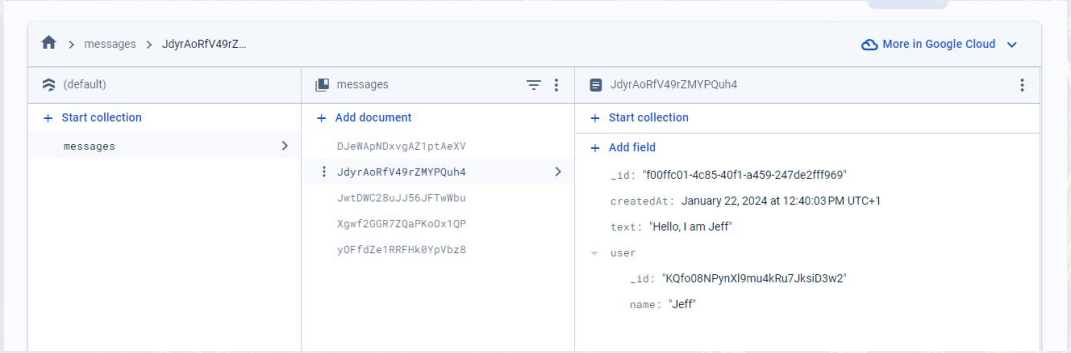


Welcome Screen.



User Flow

Google Firebase



Screenshot of my Firestore database.

```
const signInUser = () => {
    signInAnonymously(auth)
      .then(result => {
        navigation.navigate("Chat", { name: name, background: background, id: result.user.uid });
        Alert.alert("Signed in Successfully!");
      })
      .catch((error) => {
        Alert.alert("Unable to sign in, try later again.");
      })
}
```

Function to sign in user with user ID.

Storing Chat Data in Cloud Firestore

Firestore is a real-time database provided by Google Firebase. In this project it was used for storing chat data. Firestore was initialized in the root component App.js, enabling access across screens.

Using Firestore Authentication

In this project I used the Firestore anonymous authentication, which provides the user object with a unique ID that can be stored in the database for each user. Users are automatically logged in with their unique user ID whenever they open the app after clicking “Start chatting” on the welcome screen.

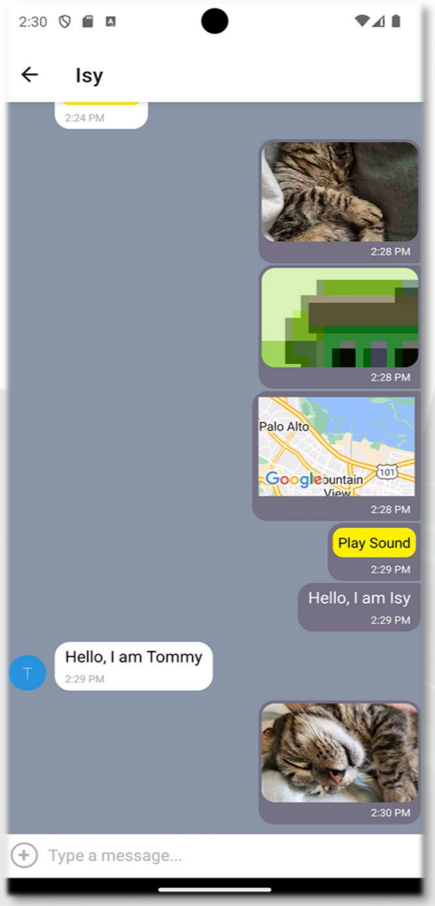
Offline Mode & Communication Features

Storing Data on Client Side

I used React Native AsyncStorage to store the messages on the client-side. Additionally, I imported NetInfo to detect network connections, enabling users to view old messages offline but preventing them from sending new ones without an internet connection.

Communication Features

Communication features were enhanced by allowing access to the media library and camera using expo-image-picker. Images were stored in Cloud Storage for Firebase. Additionally, geolocation access and sending were enabled using expo-location and react-native-maps. Audio messages could also be recorded and sent. All requiring permissions requested beforehand to respect user privacy.



Chat Screen showcasing different communication features.

Technical Lessons

1. Expo Environment Setup

Learn how to set up the Expo environment for seamless testing across various platforms, including physical devices and emulated environments.

2. React Native Project Setup

Understand the process of setting up a React Native project using Expo, including structuring the project and integrating necessary libraries and components.

3. Navigation Setup

Explore how to implement navigation between different screens within the application using the library react-navigation.

4. Implementing Communication Features

Explore the implementation of media sharing functionalities such as accessing the media library, recording and sending audio messages, and sharing geolocation using Expo APIs.

5. Client-Side Data Storage

Understand the use of React Native AsyncStorage for storing data on the client-side, enabling offline access to chat messages.

Conclusion

The original objective of building a mobile chat application using React Native was achieved, with the final result offering users a seamless chat interface along with functionalities to share images and location. Additionally, audio recording functionality was added, exceeding the initial objectives.

The most challenging aspect of the project was ensuring accurate data posting and retrieval from the Firestore database, ensuring that messages were consistently saved and visible upon reopening the app. I encountered the issue that sent messages from testing with the Android emulator were not appearing in the database. After checking my code, I discovered that I had not correctly passed the db prop from App.js to the Chat component. After fixing this, I was able to access the db prop variable in Chat.js, and the messages were saved in the Firestore database. From this experience, I learned the importance of accurately passing props between components in React Native applications and the need for thorough testing to ensure seamless communication between components.

Next time I would explore alternative authentication methods, such as Password-Based Accounts, to enable users to log in from multiple devices, enhancing user convenience and accessibility.