



# CHAT APP CASE STUDY


Helia Tutueanu



# OBJECTIVE AND TOOLS

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, voice messages and their location.

This project utilized React Native and Expo for the 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.



# SETTING UP EXPO AND THE CHAT APP

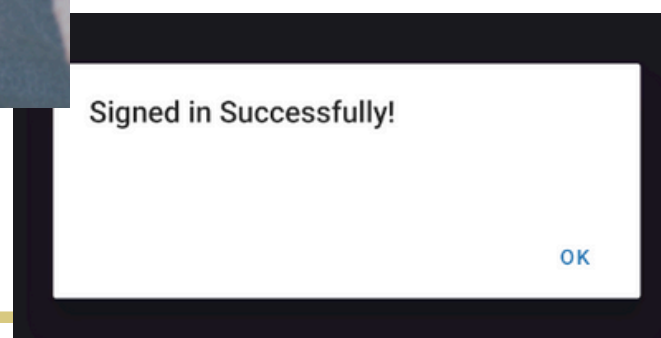
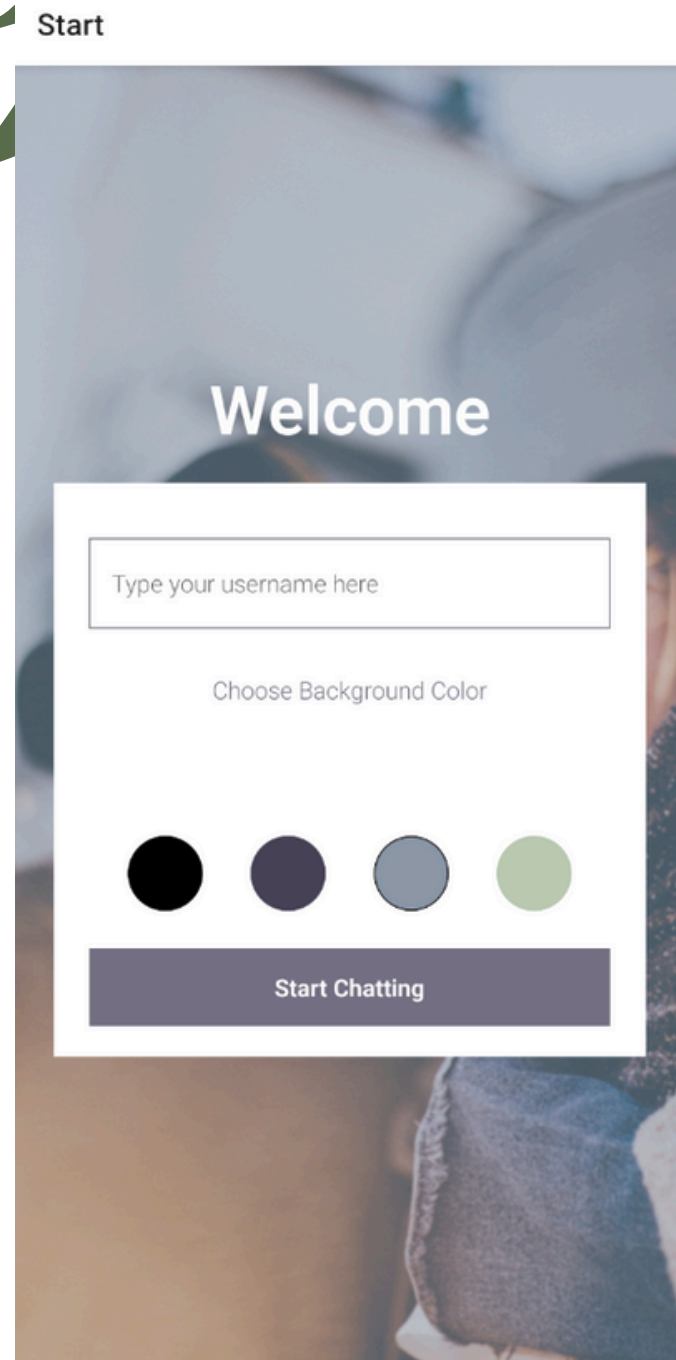
The first step involved configuring Expo Go for mobile testing and setting up an Android emulator for desktop simulation. Afterwards, I set up the Chat app using Expo and established the structure of the project.

Here, I familiarized myself with various React Native components, including TextInput, Button, Alert, TouchableOpacity, ScrollView, View, and Text.

# APPLICATION LAYOUT

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.

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



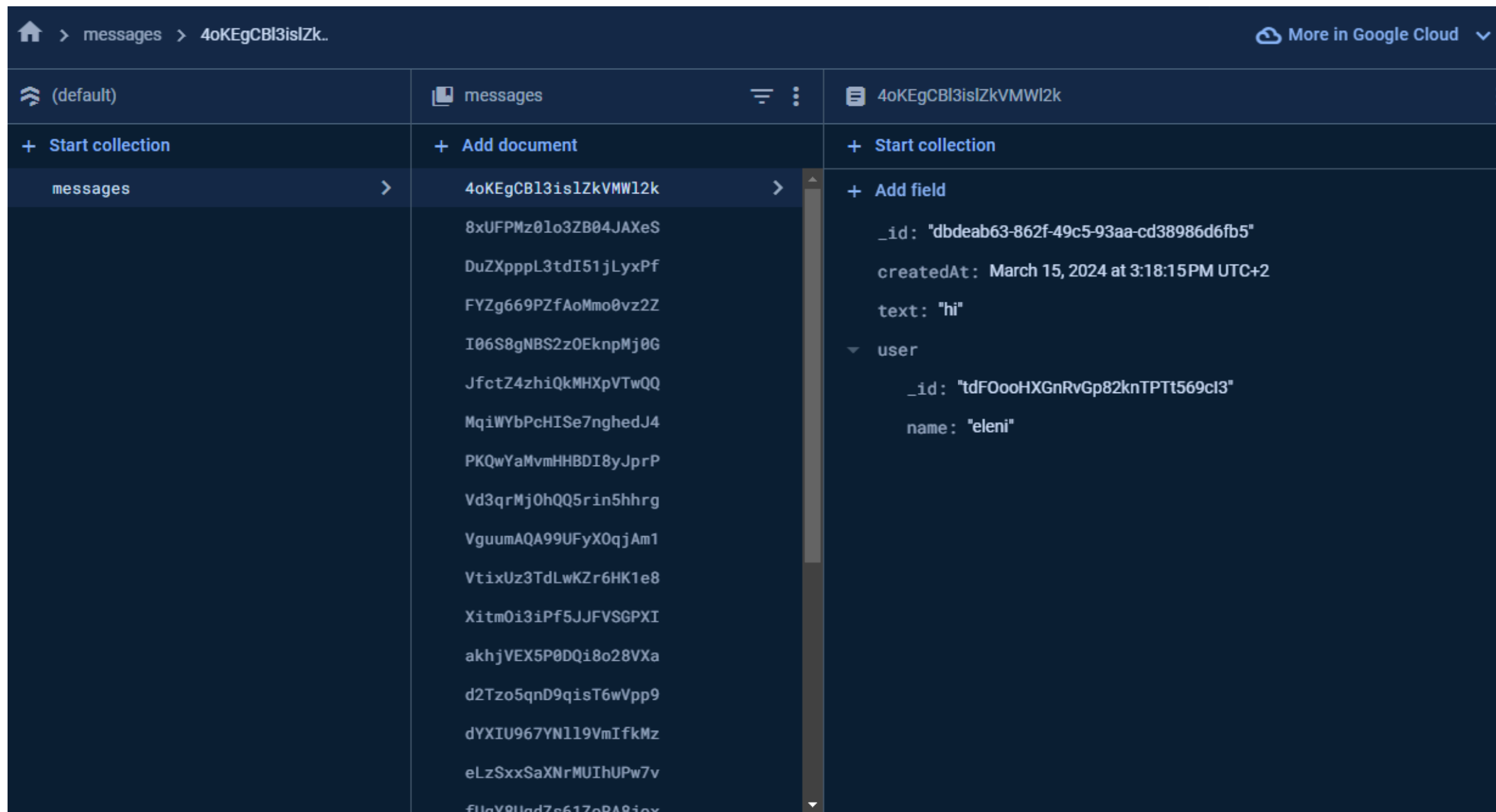


# GOOGLE FIREBASE

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

For this to happen, the Firestore anonymous authentication provides the user object with a unique ID that can be stored in the database for each user. Users are also automatically logged in with their unique user ID whenever they open the app, upon clicking “Start chatting” on the welcome screen.

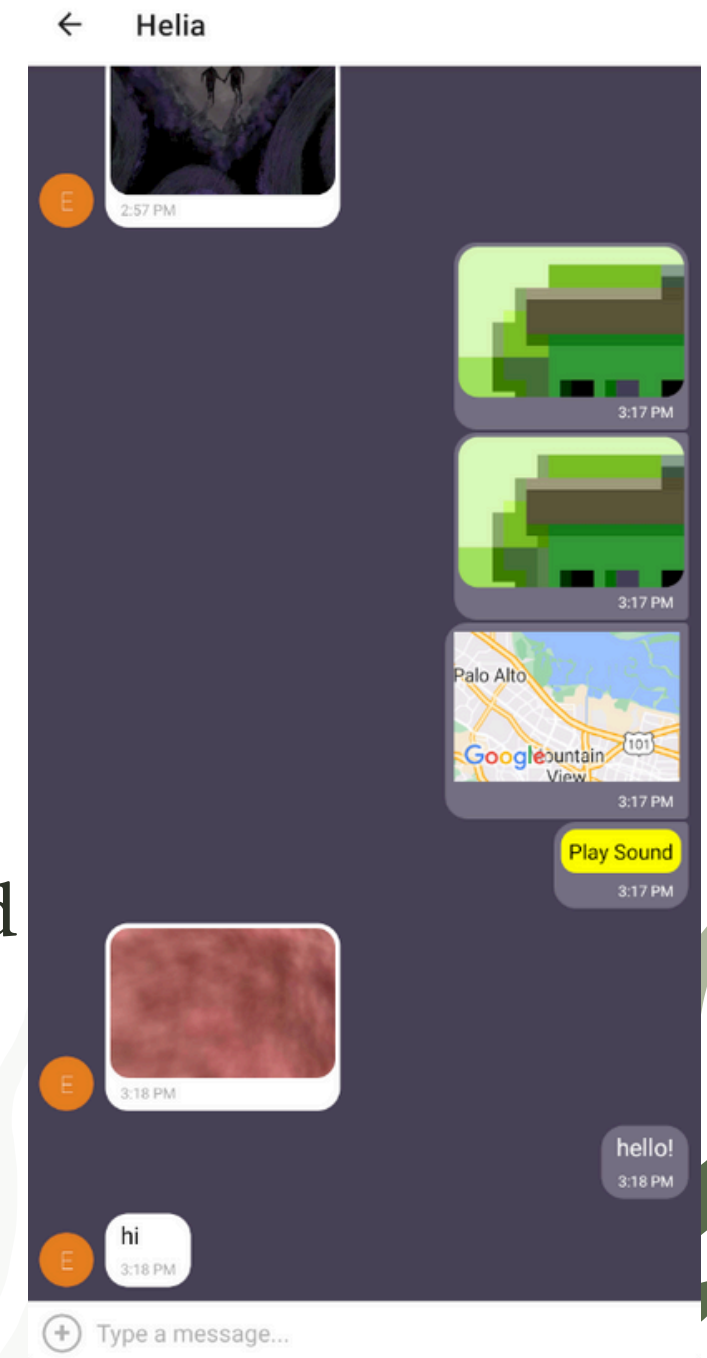
# CLOUD FIRESTORE



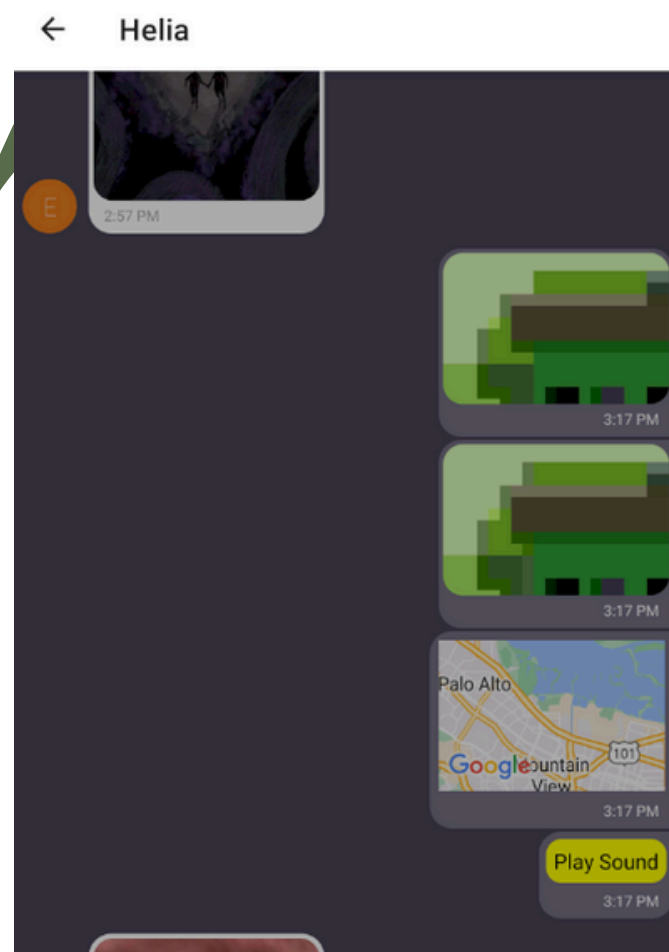
# OFFLINE MODE & COMMUNICATION FEATURES

To store messages, I used React Native AsyncStorage and imported NetInfo to detect network connections, enabling users to view old messages offline but preventing them from sending new ones without an internet connection.

Using Expo Image Picker, users are free to use the media library and camera. All sent images are stored in the Cloud Storage for Firebase. Using Expo Location and React Native Maps, geolocation was also enabled. Audio messages may also be recorded and sent. All communication features require permission before usage.



# COMMUNICATION FEATURES



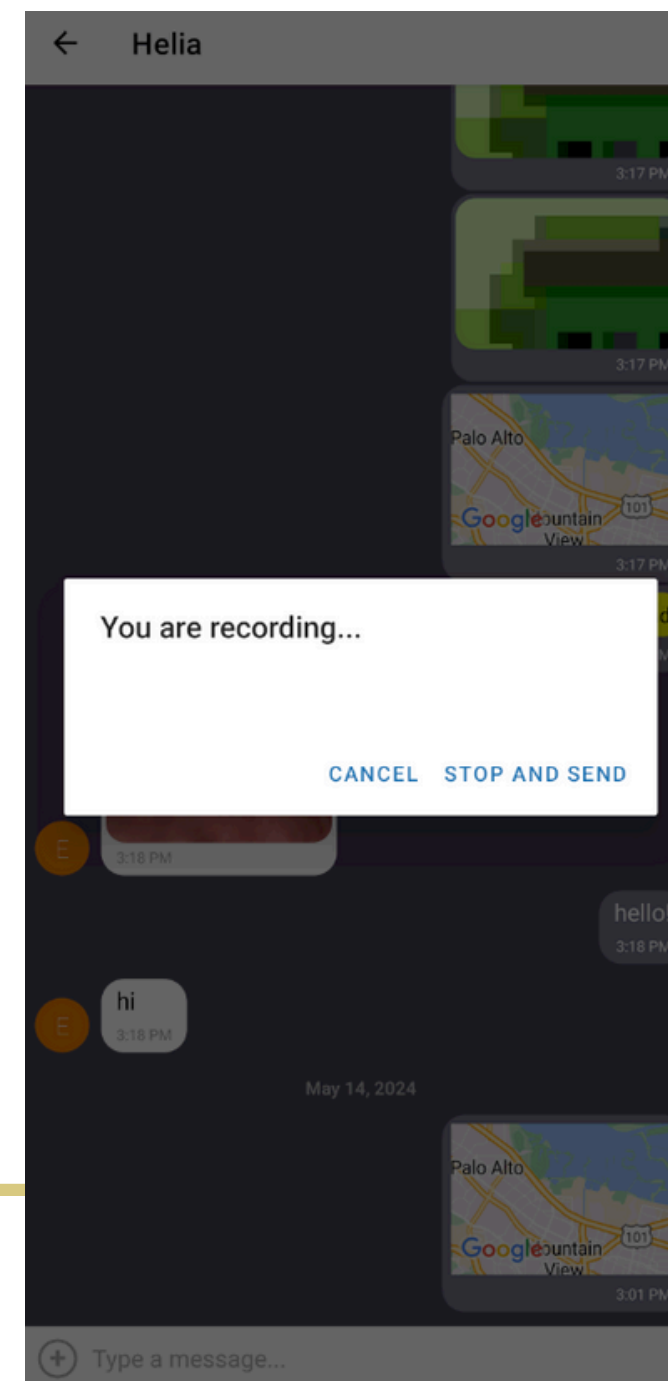
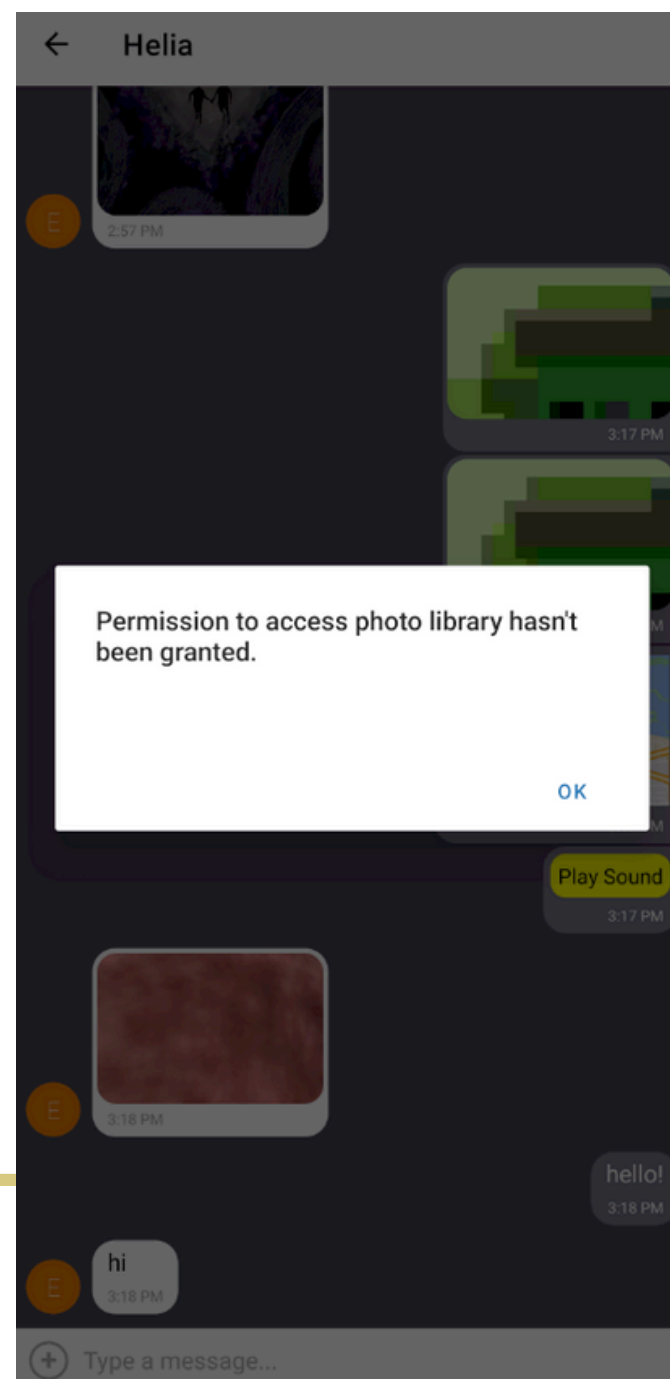
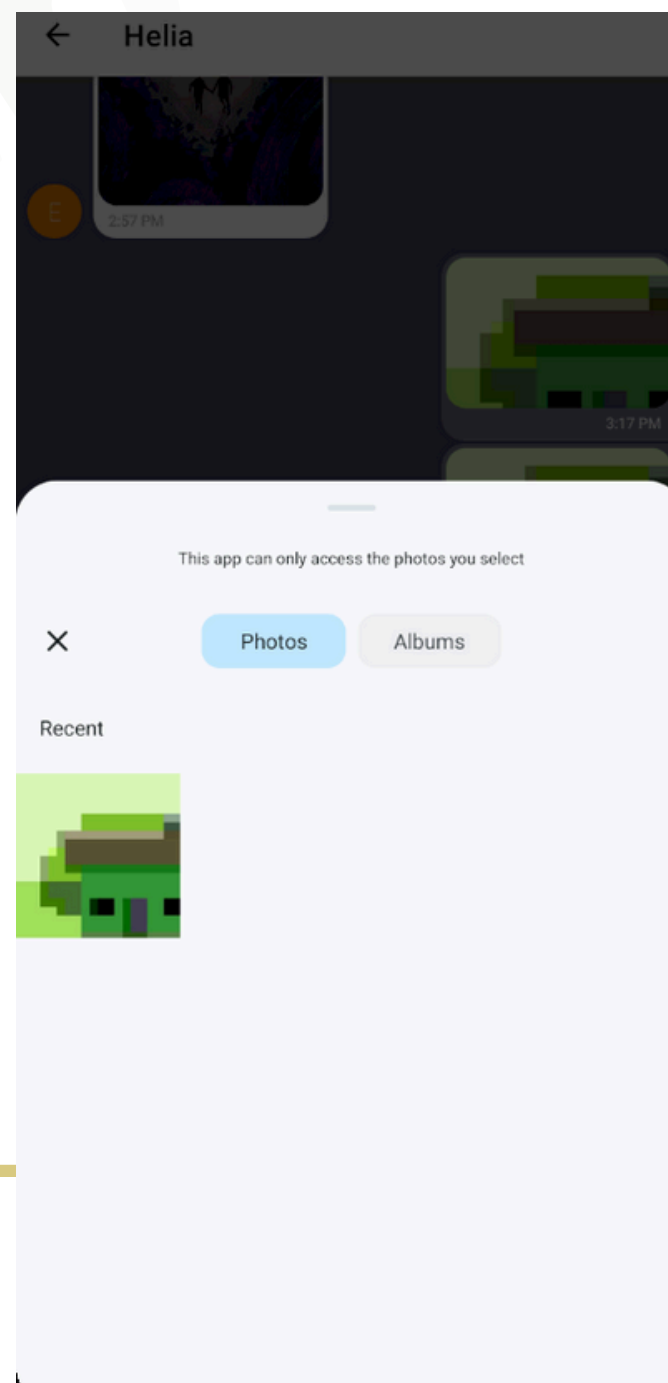
Choose From Library

Take Picture

Send Location

Record Sound

Cancel





# TECHNICAL LESSONS

- Learned how to set up the Expo environment for seamless testing across various platforms, including physical devices and emulators.
- Set up a React Native project using Expo, including structuring the project and integrating necessary libraries and components.
- Explored how to implement navigation between different screens within the application using the library react-navigation.
- Explored the implementation of media sharing functionalities such as accessing the media library, recording and sending audio messages, and sharing geolocation using Expo APIs.
- Learned the use of React Native AsyncStorage for storing data on the client-side, enabling offline access to chat messages.

# CONCLUSION

I achieved my original objective of building a mobile chat app using React Native. The most challenging aspect of the project was ensuring accurate data posting and retrieval from the Firestore database. The main issue I encountered was that the messages sent from the Android emulator were not appearing in the database. After checking my code, I discovered that I hadn't 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. Next time, I would love to explore alternative authentication methods, such as PasswordBased Accounts, which I may use to enable users to log in from multiple devices, enhancing overall user convenience and accessibility.