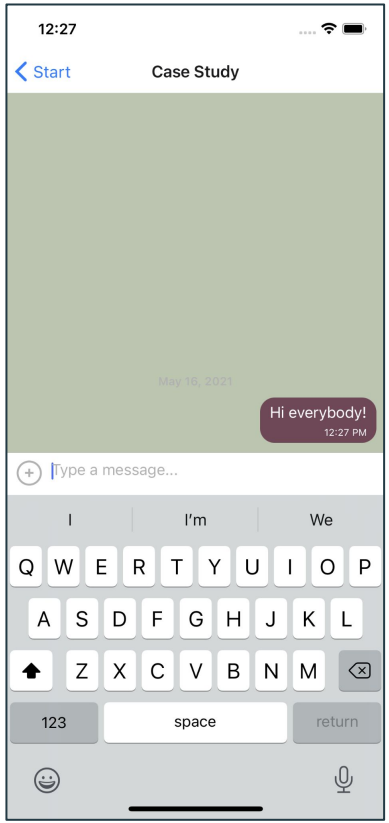
# Chat App - Case Study

Madison Bertis

# Overview of Project

This is a chat app for mobile devices built using React Native. The app provides users with a chat interface and options to share images and their location.

- *Check out my GitHub repository for this app here: https://github.com/mbertis/chat*




*Screenshot of chat app in use (taken using iOS simulator)*

# Purpose and Context

This chat app was a personal project I completed as part of my full-stack web development course with CareerFoundry to learn React Native development. This is the first native app I built and also introduced me to the Expo CLI.

```
//Allows access to camera to take photo
takePhoto = async () => {
  const { status } = await Permissions.askAsync(
    Permissions.CAMERA,
    Permissions.MEDIA_LIBRARY
  );

  if (status === "granted") {
    let result = await ImagePicker.launchCameraAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
    }).catch((error) => console.error(error));

    if (!result.cancelled) {
      const imageUrl = await this.uploadImageFetch(result.uri);
      this.props.onSend({ image: imageUrl, text: "" });
    }
  }
};
```

*A snippet of my code that allows a user to take a photo within the app and then send it in chat. All custom actions can be viewed here: https://github.com/mbertis/chat/blob/main/components/CustomActions.js*

# Objective

The goal of this project was to create a mobile app for both Android and iOS that would allow users to message each other easily. The problem I was trying to solve was to create a native application from scratch using an entirely new-to-me JavaScript framework, React Native.

# Duration

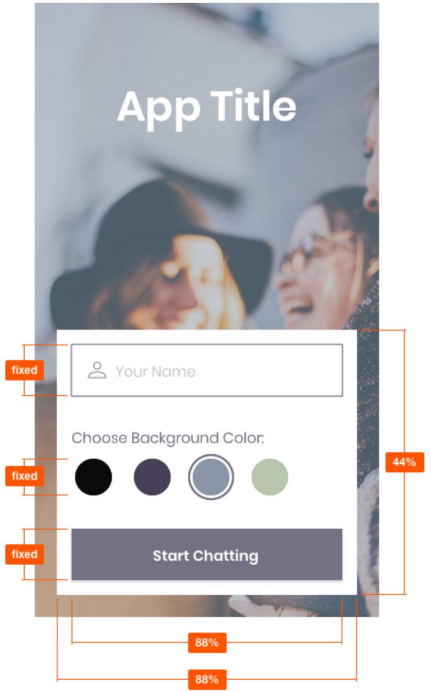
This project took me three weeks in total. I was able to finish the project within the allotted time frame while taking my time understanding and practicing the new concepts. I had some issues with images not appearing correctly in chats, but was able to remedy this by catching a small typo in my code.

# Approach

I began this project by setting up a new project using Expo CLI and creating the necessary components. Once this new project was established, I began implementing the UI as given in the project brief supplied by CareerFoundry. I really enjoyed having a target design I needed to replicate with my own UI.

*Want to see how I implemented this UI in my code? Click here: https://github.com/mbertis/chat/blob/main/components/Start.js*

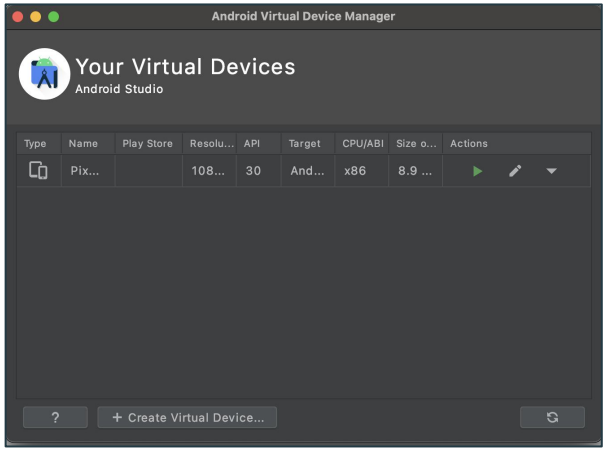
## Screen Design & Assets



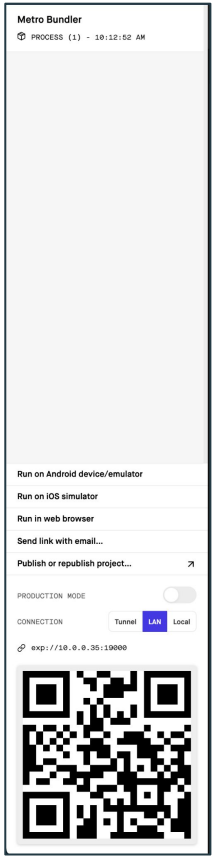

*Screen design given in CareerFoundry's project brief*

# Approach

I then set up an iOS simulator with XCode and an Android emulator with Android Studio. Setting up the iOS simulator was fairly straight-forward, while setting up the Android emulator was a bit more complex, as my system settings had to be modified to install the correct tools for the studio. Once these simulators and emulators were set up, I was able to view my app on these virtual devices as well as my own mobile phone.



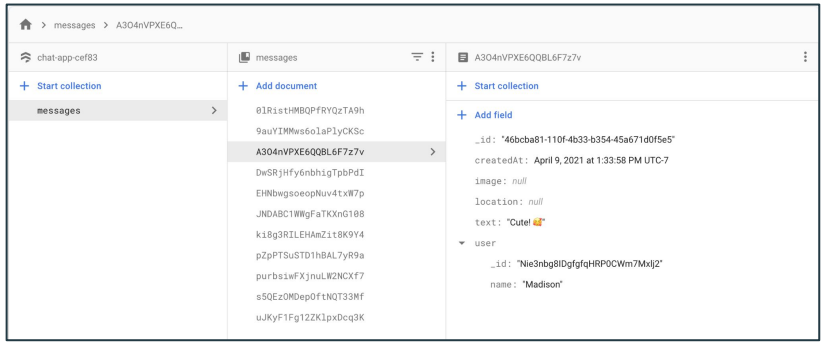

*Screenshot of Android Studio's virtual device manager*




*Screenshot of Expo's Metro Bundler, which allows me to view my app on an Android emulator, iOS simulator, or on my mobile phone using the QR code shown.*

# Approach

The next part of the project involved implementing the actual chat functionality of my app. This involved setting up a database with Firestore and implementing Gifted Chat, a popular chat UI library. Firestore allows for messages to be viewed by users even while offline, while Gifted Chat helped me to greatly improve my app's UI.




*Screenshot of Firestore "messages" database used for app*

```
// previousState references the component's state at the time when the change is applied. The append() function appends the new message to the messages object
onSend(messages = []) {
  this.setState(
    (previousState) => {{
      messages: GiftedChat.append(previousState.messages, messages),
    }),
    () => {
      this.addMessage();
      this.saveMessages();
    }
  );
}
```

*A snippet of my code that allows previously sent messages to be stored and viewed in the database*

# Approach

To finish up my app, I implemented the necessary communication features. This meant creating an Action Sheet that contains the code allowing users to take and send images as well as their location.

```
//Gets user's location
getLocation = async () => {
  const { status } = await Permissions.askAsync(Permissions.LOCATION);

  if (status === "granted") {
    let result = await Location.getCurrentPositionAsync({});

    const longitude = JSON.stringify(result.coords.longitude);
    const latitude = JSON.stringify(result.coords.latitude);

    if (result) {
      this.props.onSend({
        location: {
          longitude: longitude,
          latitude: latitude,
        },
        text: "",
      });
    }
  }
};
```

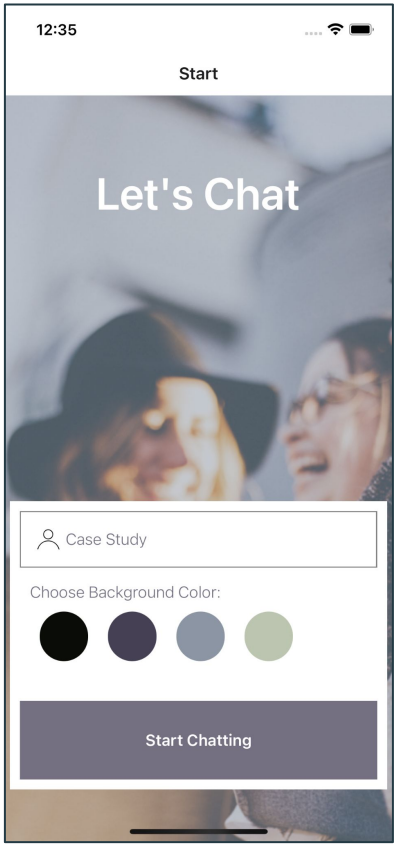
*A snippet of my code that gets the location of the app user, which they can then send in the app, as shown in the next image.*


*Screenshot of chat app in use (taken using iOS simulator)*

# Credits

**Lead Developer: Madison Bertis**
**Tutor: Gurpreet Kooner**
**Mentor: Sammy Khaleel**

*Curious to try the app yourself? Follow the instructions in my GitHub here: https://github.com/mbertis/chat/blob/main/README.md*




*Screenshot of start screen of chat app (taken on iOS simulator)*