Department of Computer Science & Engineering, University of Nevada, Reno

# EmotiFit

Team #24: Pamella Nipay, Johnathon Franco Sosa, & Daniel Guzman

Instructors: David Feil-Seifer, Devrin Lee, Sara Davis, Vinh Le, Zach Estreito

Advisor: Sean Montgomery - Connected Future Labs

Tuesday, May 7th, 2024

## Table of Contents

## 2. Final Deliverables

### 2.1: Project Summary

Table 1 delineates the implemented functionalities, encompassing fully developed and operational components, alongside unfulfilled level 1 and 2 requirements expounded upon, each accompanied by a concise list rationales for non-completion.

**Table 1: Overview of Implemented and Unimplemented Functionality with Explanations**

| **Completed Functional Requirements** | | | | |
|---|---|---|---|---|
| Completed by | Level | Name | ID | Description |
| Daniel | 1 | **User Registration** | FR 1.1 | Users should be able to create accounts by providing essential details like email and password.<br><br>User data should be securely stored in the system. |
| Daniel | 1 | **User Login** | FR 1.2 | Returning users should be able to securely log in to their accounts.<br>Account authentication should be robust to ensure data security. |
| Johnathon | 1 | **Engaging with Chatbot (OpenAI API):** | FR 1.3 | Users should be able to interact with the chatbot seamlessly for real-time assistance.<br><br>Chatbot responses should be accurate and relevant to user queries |
| Johnathon and Daniel | 1 | **Receiving Recommendations from the Chatbot:** | FR 1.4 | The chatbot should provide personalized fitness recommendations based on user inputs and preferences.<br><br>Recommendations should cover workout tips, dietary advice, and exercise routines. |
| Pamella and Daniel | 1 | **Heart Data Visualizer Screen:** | FR 1.5 | The system should include a visualizer screen that changes background colors and sound based on the user's heart rate exceeding a set threshold. |

| | | | | |
|---|---|---|---|---|
| | | | | Visual feedback should be intuitive and engaging for users during workouts. |
| Johnathon and Pamella | 1 | **Integration with Spotify API:** | FR 1.6 | The application should integrate with the Spotify API for the application to curate music playlists based off of the user's music.<br><br>Users should be able to access and modify their Spotify playlists within the EmotiFit app. |
| Pamella | 2 | **Calculating Average Heart Rate:** | FR 2.1 | The system should compute and display the user's average heart rate based on EmotiBit data. |
| **Completed Non Functional Requirements** | | | | |
| Johnathon, Pamella, Daniel | 1 | **Security:** | NR 1.1 | All user data should be encrypted and stored securely to prevent unauthorized access.<br>Communication between the application and external APIs should be encrypted to ensure data privacy |
| Johnathon, Pamella, Daniel | 1 | **Performance:** | NR 1.2 | The application should be responsive and scalable to accommodate varying user loads and concurrent interactions. |
| Johnathon, Pamella, Daniel | 1 | **Usability:** | NR 1.3 | Response times for chatbot interactions and data processing should be optimized for a smooth user experience. |
| Johnathon, Pamella, Daniel | 2 | **Reliability:** | NR 2.1 | The user interface should be intuitive and user-friendly, catering to users of different technical backgrounds.<br>Navigation within the application should be seamless, with clear prompts and instructions provided where necessary. |
| **Completed Level 3 Requirements** | | | | |
| Pamella and Daniel | 3 | **Viewing Historical EmotiBit Data:** | FR 3.1 | Users should be able to access and review their historical heart rate data to monitor progress and analyze fitness trends.<br><br>Users should be able upload CSV files with past Emotibit Data<br><br>Historical data visualization should be intuitive and informative for users. |

| Incomplete Level 1 and 2 Requirements | | | | |
|---|---|---|---|---|
| Johnathon and Pamella | 2 | **Customizing Music Tempo:** | FR 2.2 | Users should have the ability to manually adjust the tempo of their music playlists to synchronize with their workout pace.<br>Tempo customization should enhance motivation and performance during exercise sessions<br>Average heart rate calculations should inform dynamic adjustments in the heart rate data visualizer in music tempo.<br>**Reason:** Unfortunately, this feature was not completed due to time constraints and the challenges faced in integrating the heart rate data visualizer screen. The complexity of implementing and debugging the visualizer, which provides real-time feedback by changing background colors based on heart rate thresholds, required significant development focus, diverting resources away from the tempo customization functionality. |
| **Incomplete Level 3 Requirements** | | | | |
| | 3 | **Connecting to Other Heart Rate Monitors or Watches:** | FR 3.2 | The application should support connectivity with other heart rate monitors or wearable devices to broaden compatibility.<br><br>Users should have the option to use alternative heart rate monitoring devices. |
| | 3 | **Connecting Different Music Apps (e.g., Apple Music):** | FR 3.3 | The system should extend integration capabilities to other music streaming platforms like Apple Music, providing users with more options for personalized playlists. |
| | 3 | **Tailoring Music to Non-Stationary Workouts (e.g., Cycling vs. Weightlifting):** | FR 3.4 | The application should adapt music playlists based on the type of workout or activity, such as cycling, weightlifting, or running.<br><br>Music selection should align with the intensity and rhythm of different exercises to enhance user experience and motivation. |

| | | | | |
|---|---|---|---|---|
| | 3 | **Spotify Web Player** | FR 3.5 | **Reason:** EmotiFit's proposed use case involves displaying the Spotify Web Player within the app, intended as a Level 3 feature for advanced user interaction. However, this functionality was not implemented due to Spotify's security restrictions, which prevent the use of WebViews for displaying content in external applications, including React Native platforms. Instead, users can access Spotify music through a web browser, ensuring secure and compliant usage while interacting with EmotiFit's personalized workout features. |
| | 3 | **Compatibility:** | R3.1 | **Reason:** The implementation of the BrainFlow API in our EmotiFit project was not completed due to several technical and compatibility challenges. Although we had the opportunity to collaborate with the founder of BrainFlow and another developer experienced with BrainFlow and React Native, our attempts to integrate the API using TypeScript did not succeed. A significant hurdle was that the EmotiBit device, which the BrainFlow API was intended to connect with, is primarily rooted in Python and C++ and thus incompatible directly with mobile environments. Consequently, we had to adapt by using the EmotiBit oscilloscope for data streaming as a temporary solution and pass in UDP data. Moving forward, we remain hopeful about eventually integrating the BrainFlow API to directly stream data with React Native, enhancing our app's functionality and user experience. |
| | 3 | **Scalability:** | R3.2 | The system architecture should support future growth and expansion, allowing for seamless integration of new features and functionalities.<br><br>Scalability testing should be performed to evaluate the system's ability to handle increasing user demands over time. |

## Contributions of Team Members

Johnathon spent about 15 hours on this project part but worked consistently on the entire project 8 months in total. Johnathon focused on section 2, 4 and contributed to getting the Spotify API connection on the React-Native bare workflow to interact with the user's specific information such as their Spotify account statistics, the Chatbot to recommend songs to users if they inputted a specific BPM.

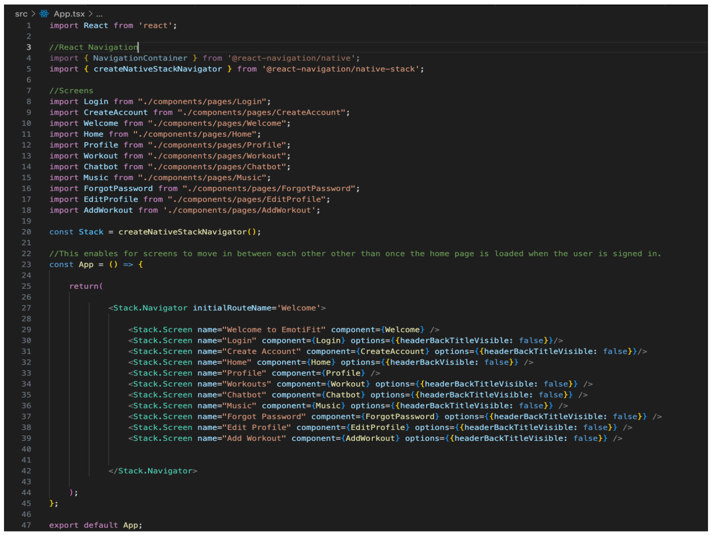
Pam spent about 15 hours on this project part but worked consistently on the entire project 8 months in total. Pam focused on section 1, 2, 3 and contributed to getting the Emotibit's UDP connection on the React-Native bare workflow with Daniel, the formula to calculate heart rate data based on anaerobic exercise for the data visualizer screen, collaborating with Johnathon to have the user input their bpm and implementing a way to import past historical emotibit data onto the EmotiFit App.

Daniel spent about 15 hours on this project part but worked consistently on the entire project 8 months in total. Daniel contributed on the overall structure of the code to import the Emotibit's UDP Data, Spotify API, Chatbot, and FireBase Data Base onto one React-Native bare workflow, connect the React Native Application to the OpenAI API, the spearheaded the implementation of the login functionalities on the application, and focused on section 2.

## 2.2: Software Overview

With the software overview of our project,

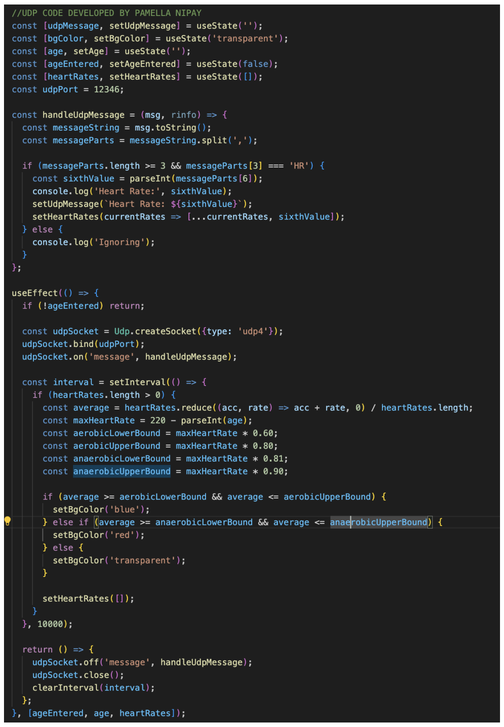
Stack Navigator from React Navigation: This portion of the code is a stack navigator, which is how the application can move in between screens, and . This code was found in documentation from React Navigation, but was developed by Daniel. The documentation had a skeleton version, and we adapted it for use with EmotiFit.



src > App.tsx > ...

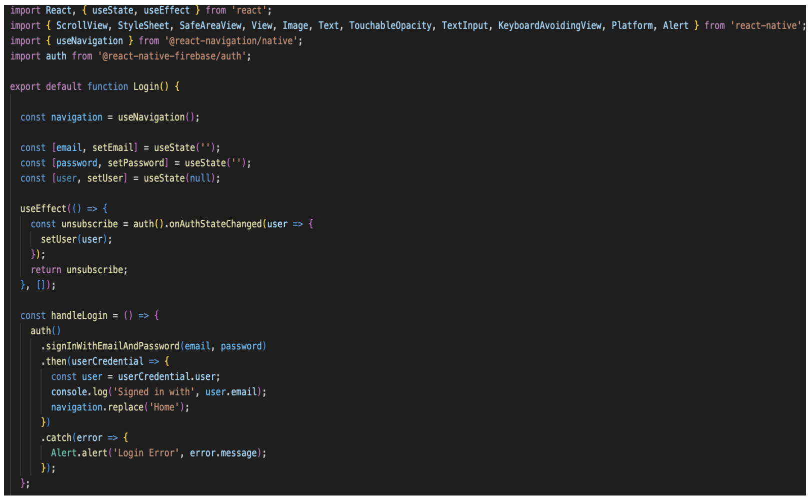
```
import React from 'react';

//React Navigation
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

//Screens
import Login from "./components/pages/Login";
import CreateAccount from "./components/pages/CreateAccount";
import Welcome from "./components/pages/Welcome";
import Home from "./components/pages/Home";
import Profile from "./components/pages/Profile";
import Workout from "./components/pages/Workout";
import Chatbot from "./components/pages/Chatbot";
import Music from "./components/pages/Music";
import ForgotPassword from "./components/pages/ForgotPassword";
import EditProfile from "./components/pages/EditProfile";
import AddWorkout from './components/pages/AddWorkout';

const Stack = createNativeStackNavigator();

//This enables for screens to move in between each other other than once the home page is loaded when the user is signed in.
const App = () => {

    return(

            <Stack.Navigator initialRouteName='Welcome'>

                <Stack.Screen name="Welcome to EmotiFit" component={Welcome} />
                <Stack.Screen name="Login" component={Login} options={{headerBackTitleVisible: false}}/>
                <Stack.Screen name="Create Account" component={CreateAccount} options={{headerBackTitleVisible: false}}/>
                <Stack.Screen name="Home" component={Home} options={{headerBackVisible: false}} />
                <Stack.Screen name="Profile" component={Profile} />
                <Stack.Screen name="Workouts" component={Workout} options={{headerBackTitleVisible: false}} />
                <Stack.Screen name="Chatbot" component={Chatbot} options={{headerBackTitleVisible: false}} />
                <Stack.Screen name="Music" component={Music} options={{headerBackTitleVisible: false}} />
                <Stack.Screen name="Forgot Password" component={ForgotPassword} options={{headerBackTitleVisible: false}} />
                <Stack.Screen name="Edit Profile" component={EditProfile} options={{headerBackTitleVisible: false}} />
                <Stack.Screen name="Add Workout" component={AddWorkout} options={{headerBackTitleVisible: false}} />


            </Stack.Navigator>

    );
};

export default App;
```

Another component that makes up our software is the UDP data streaming in from EmotiBit to React Native. With this, the code listens for UDP data on the network, and once it is found, we parse the data to select certain parts of that data streaming in, such as the heart rate identifier ‘HR’ and the actual heart rate number.

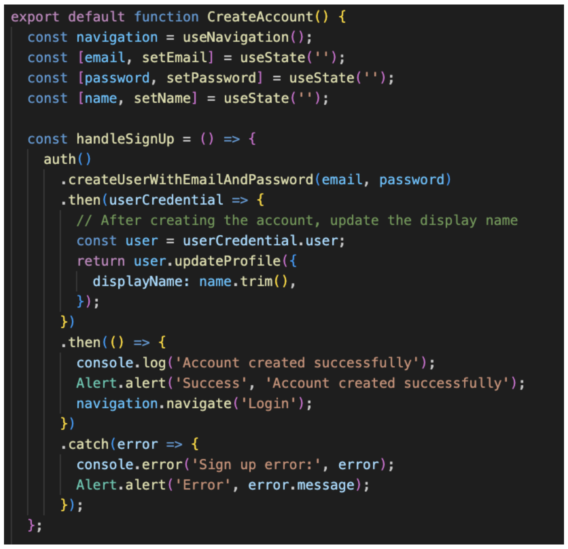


```javascript
//UDP CODE DEVELOPED BY PAMELLA NIPAY
const [udpMessage, setUdpMessage] = useState('');
const [bgColor, setBgColor] = useState('transparent');
const [age, setAge] = useState('');
const [ageEntered, setAgeEntered] = useState(false);
const [heartRates, setHeartRates] = useState([]);
const udpPort = 12346;

const handleUdpMessage = (msg, rinfo) => {
  const messageString = msg.toString();
  const messageParts = messageString.split(',');

  if (messageParts.length >= 3 && messageParts[3] === 'HR') {
    const sixthValue = parseInt(messageParts[6]);
    console.log('Heart Rate:', sixthValue);
    setUdpMessage(`Heart Rate: ${sixthValue}`);
    setHeartRates(currentRates => [...currentRates, sixthValue]);
  } else {
    console.log('Ignoring');
  }
};

useEffect(() => {
  if (!ageEntered) return;

  const udpSocket = Udp.createSocket({type: 'udp4'});
  udpSocket.bind(udpPort);
  udpSocket.on('message', handleUdpMessage);

  const interval = setInterval(() => {
    if (heartRates.length > 0) {
      const average = heartRates.reduce((acc, rate) => acc + rate, 0) / heartRates.length;
      const maxHeartRate = 220 - parseInt(age);
      const aerobicLowerBound = maxHeartRate * 0.60;
      const aerobicUpperBound = maxHeartRate * 0.80;
      const anaerobicLowerBound = maxHeartRate * 0.81;
      const anaerobicUpperBound = maxHeartRate * 0.90;

      if (average >= aerobicLowerBound && average <= aerobicUpperBound) {
        setBgColor('blue');
      } else if (average >= anaerobicLowerBound && average <= anaerobicUpperBound) {
        setBgColor('red');
      } else {
        setBgColor('transparent');
      }

      setHeartRates([]);
    }
  }, 10000);

  return () => {
    udpSocket.off('message', handleUdpMessage);
    udpSocket.close();
    clearInterval(interval);
  };
}, [ageEntered, age, heartRates]);
```

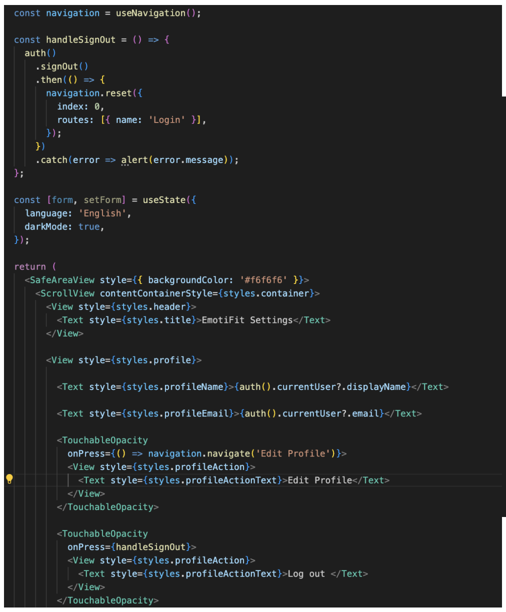
Another component that makes up our software is the Firebase implementation for our app. This allows users to login, create accounts and manage their profiles such as changing their display name or accessing the Firestore database for their workouts. As seen below, these components were from Firebase documentation, with slight tweaks to adjust for React Native to read it in Javascript.



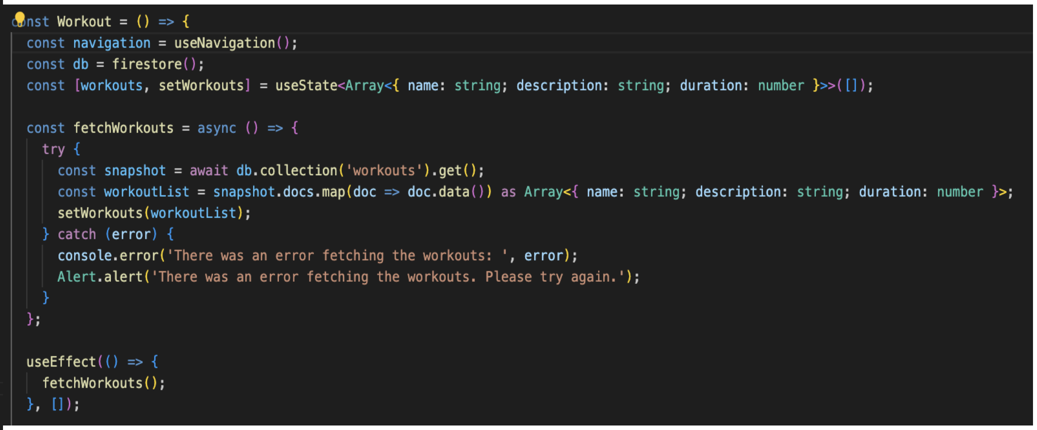
```
import React, { useState, useEffect } from 'react';
import { ScrollView, StyleSheet, SafeAreaView, View, Image, Text, TouchableOpacity, TextInput, KeyboardAvoidingView, Platform, Alert } from 'react-native';
import { useNavigation } from '@react-navigation/native';
import auth from '@react-native-firebase/auth';

export default function Login() {

  const navigation = useNavigation();

  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [user, setUser] = useState(null);

  useEffect(() => {
    const unsubscribe = auth().onAuthStateChanged(user => {
      setUser(user);
    });
    return unsubscribe;
  }, []);

  const handleLogin = () => {
    auth()
      .signInWithEmailAndPassword(email, password)
      .then(userCredential => {
        const user = userCredential.user;
        console.log('Signed in with', user.email);
        navigation.replace('Home');
      })
      .catch(error => {
        Alert.alert('Login Error', error.message);
      });
  };
```



```
export default function CreateAccount() {
  const navigation = useNavigation();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [name, setName] = useState('');

  const handleSignUp = () => {
    auth()
      .createUserWithEmailAndPassword(email, password)
      .then(userCredential => {
        // After creating the account, update the display name
        const user = userCredential.user;
        return user.updateProfile({
          displayName: name.trim(),
        });
      })
      .then(() => {
        console.log('Account created successfully');
        Alert.alert('Success', 'Account created successfully');
        navigation.navigate('Login');
      })
      .catch(error => {
        console.error('Sign up error:', error);
        Alert.alert('Error', error.message);
      });
  };
```

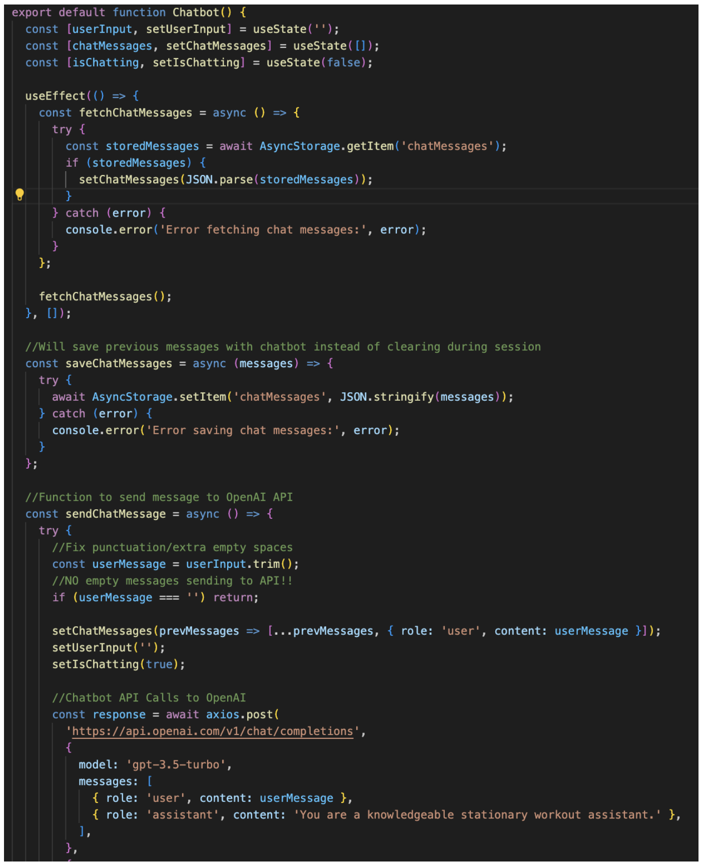


```
const navigation = useNavigation();

const handleSignOut = () => {
  auth()
    .signOut()
    .then(() => {
      navigation.reset({
        index: 0,
        routes: [{ name: 'Login' }],
      });
    })
    .catch(error => alert(error.message));
};

const [form, setForm] = useState({
  language: 'English',
  darkMode: true,
});

return (
  <SafeAreaView style={{ backgroundColor: '#f6f6f6' }}>
    <ScrollView contentContainerStyle={styles.container}>
      <View style={styles.header}>
        <Text style={styles.title}>EmotiFit Settings</Text>
      </View>

      <View style={styles.profile}>

        <Text style={styles.profileName}>{auth().currentUser?.displayName}</Text>

        <Text style={styles.profileEmail}>{auth().currentUser?.email}</Text>

        <TouchableOpacity
          onPress={() => navigation.navigate('Edit Profile')}>
          <View style={styles.profileAction}>
            <Text style={styles.profileActionText}>Edit Profile</Text>
          </View>
        </TouchableOpacity>

        <TouchableOpacity
          onPress={handleSignOut}>
          <View style={styles.profileAction}>
            <Text style={styles.profileActionText}>Log out </Text>
          </View>
        </TouchableOpacity>
```

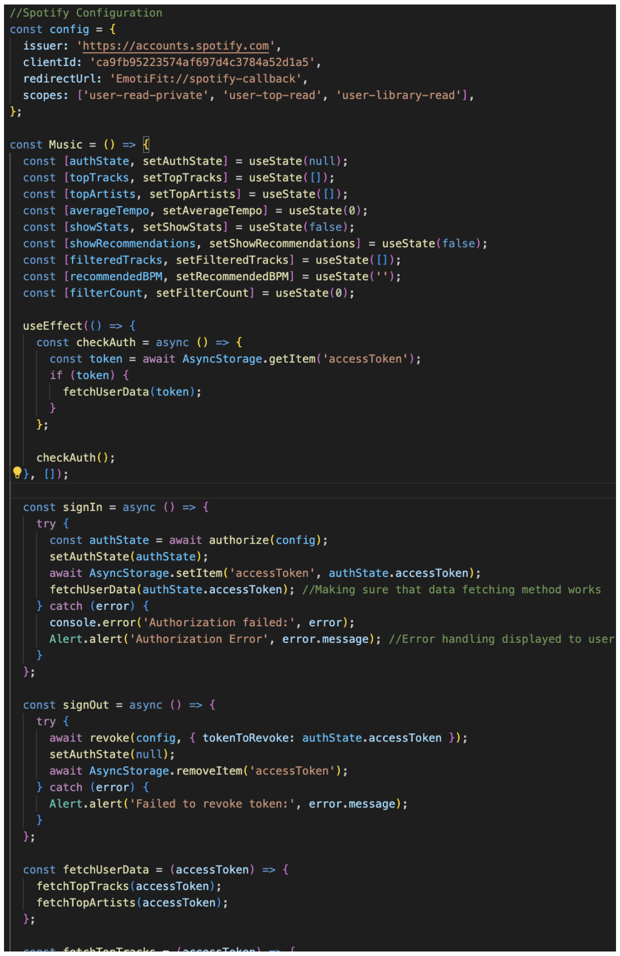


```
const Workout = () => {
  const navigation = useNavigation();
  const db = firestore();
  const [workouts, setWorkouts] = useState<Array<{ name: string; description: string; duration: number }>>([]);

  const fetchWorkouts = async () => {
    try {
      const snapshot = await db.collection('workouts').get();
      const workoutList = snapshot.docs.map(doc => doc.data()) as Array<{ name: string; description: string; duration: number }>;
      setWorkouts(workoutList);
    } catch (error) {
      console.error('There was an error fetching the workouts: ', error);
      Alert.alert('There was an error fetching the workouts. Please try again.');
    }
  };

  useEffect(() => {
    fetchWorkouts();
  }, []);
```

Two last components that make up our software is both the Spotify API & OpenAI API integration. With the Spotify API (the left photo), it handles the user log in with a pop up browser in React Native, and will get the information to display the users top tracks, top artists, and make recommendations based on a manual input of BPM. With the OpenAI API (the right photo), it uses the API provided based on our configurations with OpenAI. This allows for users to interact with the chatbot to get recommendations based on the workout wanting to be achieved, and also saves the conversation using the 'AsyncStorage' library from React Native.



```
export default function Chatbot() {
  const [userInput, setUserInput] = useState('');
  const [chatMessages, setChatMessages] = useState([]);
  const [isChatting, setIsChatting] = useState(false);

  useEffect(() => {
    const fetchChatMessages = async () => {
      try {
        const storedMessages = await AsyncStorage.getItem('chatMessages');
        if (storedMessages) {
          setChatMessages(JSON.parse(storedMessages));
        }
      } catch (error) {
        console.error('Error fetching chat messages:', error);
      }
    };

    fetchChatMessages();
  }, []);

  //Will save previous messages with chatbot instead of clearing during session
  const saveChatMessages = async (messages) => {
    try {
      await AsyncStorage.setItem('chatMessages', JSON.stringify(messages));
    } catch (error) {
      console.error('Error saving chat messages:', error);
    }
  };

  //Function to send message to OpenAI API
  const sendChatMessage = async () => {
    try {
      //Fix punctuation/extra empty spaces
      const userMessage = userInput.trim();
      //NO empty messages sending to API!!
      if (userMessage === '') return;

      setChatMessages(prevMessages => [...prevMessages, { role: 'user', content: userMessage }]);
      setUserInput('');
      setIsChatting(true);

      //Chatbot API Calls to OpenAI
      const response = await axios.post(
        'https://api.openai.com/v1/chat/completions',
        {
          model: 'gpt-3.5-turbo',
          messages: [
            { role: 'user', content: userMessage },
            { role: 'assistant', content: 'You are a knowledgeable stationary workout assistant.' },
          ],
        },
```



```
//Spotify Configuration
const config = {
  issuer: 'https://accounts.spotify.com',
  clientId: 'ca9fb95223574af697d4c3784a52d1a5',
  redirectUrl: 'EmotiFit://spotify-callback',
  scopes: ['user-read-private', 'user-top-read', 'user-library-read'],
};

const Music = () => {
  const [authState, setAuthState] = useState(null);
  const [topTracks, setTopTracks] = useState([]);
  const [topArtists, setTopArtists] = useState([]);
  const [averageTempo, setAverageTempo] = useState(0);
  const [showStats, setShowStats] = useState(false);
  const [showRecommendations, setShowRecommendations] = useState(false);
  const [filteredTracks, setFilteredTracks] = useState([]);
  const [recommendedBPM, setRecommendedBPM] = useState('');
  const [filterCount, setFilterCount] = useState(0);

  useEffect(() => {
    const checkAuth = async () => {
      const token = await AsyncStorage.getItem('accessToken');
      if (token) {
        fetchUserData(token);
      }
    };

    checkAuth();
  }, []);

  const signIn = async () => {
    try {
      const authState = await authorize(config);
      setAuthState(authState);
      await AsyncStorage.setItem('accessToken', authState.accessToken);
      fetchUserData(authState.accessToken); //Making sure that data fetching method works
    } catch (error) {
      console.error('Authorization failed:', error);
      Alert.alert('Authorization Error', error.message); //Error handling displayed to user
    }
  };

  const signOut = async () => {
    try {
      await revoke(config, { tokenToRevoke: authState.accessToken });
      setAuthState(null);
      await AsyncStorage.removeItem('accessToken');
    } catch (error) {
      Alert.alert('Failed to revoke token:', error.message);
    }
  };

  const fetchUserData = (accessToken) => {
    fetchTopTracks(accessToken);
    fetchTopArtists(accessToken);
  };

  const fetchTopTracks = (accessToken) => {
```

## 2.3: Source Code

The source code is attached in the submission of this assignment.

## 3. Engineering Design Assessment

In assessing our senior project's engineering design, we've meticulously considered various critical factors that underpin its societal impact and functionality in the following ways. From addressing public health and safety concerns to accommodating global, cultural, social, environmental, and economic considerations, our project embodies a comprehensive approach to engineering design, ensuring it meets diverse needs and aligns with broader societal values.

Public Health: EmotiFit promotes public health by encouraging regular exercise through personalized workout experiences, which can contribute to improved cardiovascular health and overall well-being.

Public Safety: Safety measures embedded in EmotiFit, such as reliable heart rate monitoring and adherence to safety standards for wearable fitness technology, ensure users can engage in workouts safely without compromising their health.

Public Welfare: By offering a user-friendly and motivating exercise environment, EmotiFit contributes to public welfare by promoting physical activity and fostering healthier lifestyles among individuals.

Global Factors: EmotiFit's accessibility across different platforms and its potential to reach users worldwide align with global factors, addressing the universal need for personalized fitness solutions regardless of geographical location.

Cultural Factors: EmotiFit acknowledges cultural diversity by providing customizable features and music playlists, allowing users to incorporate their cultural preferences into their workout routines, thus enhancing their overall exercise experience.

Social Factors: The integration of a chatbot powered by the Spotify API encourages social interaction and community engagement within the EmotiFit platform, fostering a sense of camaraderie and support among users pursuing their fitness goals.

Environmental Factors: EmotiFit promotes environmentally friendly exercise options by encouraging stationary workouts, which reduce the carbon footprint associated with transportation to fitness facilities, contributing to environmental sustainability.

Economic Factors: EmotiFit offers cost-effective fitness solutions by utilizing affordable hardware components like the EmotiBit device and providing a comprehensive fitness experience without the need for expensive gym memberships or equipment, thereby enhancing accessibility to fitness for users across different economic backgrounds.

## 4. Engineering Standards Compliance

The UDP library is critical in EmotiFit for establishing a real-time data flow between the application and the EmotiBit heart rate device. The protocol's full-duplex communication capabilities allow efficient two-way communication, which is key for the real-time synchronization of heart rate data. This ensures that users experience minimal latency in receiving feedback, enabling them to see heart rate changes instantly and adjust workouts accordingly. EmotiFit has fully complied with this standard by incorporating UDP library into every aspect where real-time data transmission is essential, such as in updating the heart data visualizer. As a result, users can receive immediate, personalized recommendations.

- FR1 (Real-Time Heart Rate Monitoring): UDP library enables consistent transmission of heart rate data to update the heart data.
- FR2 (Interactive Adjustments): Users can adjust their workouts based on immediate feedback, leveraging the UDP library channel's low-latency communication.
- UC1 (Monitoring and Adjusting Workout Regime): Users receive live feedback and can adapt their exercises, improving engagement and personalization.

EmotiFit uses a unified authentication system combining Firebase and Auth0 to ensure secure access to both the app and Spotify's API. This system integrates Firebase Authentication to securely manage user login credentials for the app, providing flexibility through email/password logins. Meanwhile, Auth0 handles the OAuth 2.0-based login for Spotify, enabling seamless playlist management and recommendations without compromising user security. Together, these technologies offer robust user authentication and maintain compliance with data privacy and user consent requirements.

- FR3 (Secure User Login): Firebase Authentication handles secure user login credentials, offering flexibility with email/password.
- FR4 (Secure Spotify Integration): Auth0 implements the OAuth 2.0 standard, ensuring that Spotify data remains protected while delivering personalized recommendations.
- NFR1 (Data Security and Privacy): Firebase and Auth0 manage sensitive user credentials with high security, meeting data privacy standards.
- UC2 (User Logs In to EmotiFit): Firebase Authentication ensures users can log in and access the app securely.
- UC3 (Spotify Integration): Users grant permissions through Auth0 to sync their Spotify playlists with workout intensity.