Technical Guide

💡 This technical guide can be useful for the Talent Insight Team.

Team GLGS is proposing the development of a mobile application named "Athlete Insight" that aligns with the goals of the Talent Insight Solution. This innovative solution aims to support the enhancement of athletes' performance and the development of insights. The application will serve as a tool to facilitate collaborative relationships, fostering cooperation among athletes, coaches, clubs, and parents. Through this, we intend to introduce a competitive "Athlete Insight" application to the industry.

> **Project Name :** Athlete Insight
> **Client Name:** Telent Insight Solution
> **Team: GLGS**
> **Team Member :** Gilseon Kim / Grace Hwang / Subin Choo / Lincoln Holmans
> **Date:** Oct 27, 2023

TABLE OF CONTENTS

- Data Security and Permissions
- Test

- Collaboration work guide

Environment Setting and Preferences

1. Download Visual Studio Code
    a. Visual Studio Code is a free lightweight code editor and integrated development environment (IDE). You can download and install the version that corresponds to your operating system from here. With a variety of available extensions, Visual Studio Code is a valuable tool for app development."
2. **Install Node.js**
    a. First, you need to install Node.js, which provides the JavaScript environment required for React Native and Expo development. You can download and install Node.js from the official Node.js website.
3. **Install Expo CLI & Simulator**
    a. Expo CLI is essential for creating and managing Expo projects. Open your terminal or command prompt and run the following command to install Expo CLI globally.
        i. - npm install -g expo-cli
    b. Installing an Android emulator (using Android Studio) or an iOS simulator (using Xcode) is optional.
4. **Create a Project Directory**
    a. Create a project directory in the location of your choice by running the following command. you can replace "athlete-insight-app" with your preferred project name.
        - expo init athlete-insight-app
5. Importing Athlete Insight development files
    a. Developers retrieve app files that have already been worked on. Typically, these files consist of source code and resources located within the project directory.
6. **Navigate to the Project Directory**
    a. Change your working directory to the newly created project directory using the terminal.
        - cd my-mobile-app
7. **Install Dependencies**
    a. Inside the project directory, install the required dependencies by running
        - npm install
8. **Run the App**

To run the mobile app, use one of the following processes.

a. **Run the server.**

i. **Open the "TIS_Server_side" folder on the new window of Visual Studio.**

ii. **Open the "new terminal" from the menu bar that the top of the window.**

1. node index.js
2. npm start

a. Run the Client.

i. Open the "TIS_Client_side" folder on another new window of Visual Studio.

ii. Open the "new terminal" from the menu bar that the top of the window.

1. npm start

Project Architecture / Client

Components

Comment.jsx

GoalList.jsx

GoalTodo.jsx

ImageSelcetion.jsx

MindsetHistory.jsx

MindsetNewPost.jsx

MindsetPost.jsx

MindsetScroll.jsx

MindsetTopBar.jsx

NoteNewBtn.jsx

PerformanceNewPost.jsx

PerformancePost.jsx

SearchBar.jsx

Image

All image files of the logo, icons, and image files for application development.

Navigation

RegirationStack.jsx

LoginStack.jsx

AthleteHomeStack.jsx

CoachHomeStack.jsx

Pages

- Athlete
  - FinxS
    - FinxS_Pages
      - FinxS_GetPDF.jsx
      - FinxS_Main.jsx
      - FinxS_SingInAuth.jsx
      - smtpConfig.jsx
    - FinxS_Password
      - CheckMultiplePassword.jsx
      - CheckPassword.jsx
      - GeneratePassword.jsx
      - UploadPassword.jsx
  - AthleteCalendarScreen.jsx
  - AtheleteGoalScreen.jsx
  - AthleteHomeScreen.jsx
  - AthleteNoteScreen.jsx
  - AthletePerformance.jsx
  - AthleteProfile.jsx
  - AthleteSettingScreen.jsx
  - AthleteSignUp.jsx
  - AthleteSocialScreen.jsx
  - AthleteMindset.jsx.
  - AthleteWellbeing.jsx
  - AthleteWellbeingUpdate.jsx
- Club
  - ClubHomeScreen.jsx
  - ClubSignUp.jsx
- Coach
  - CoachCalendarScreen.jsx
  - CoachHomeScreen.jsx
  - CoachMindsetScreen.jsx
  - CoachPerformanceSceen.jsx
  - CoachProfile.jsx
  - CoachSettingScreen.jsx
  - CoachSignUp.jsx
  - CoachSocialScreen.jsx

CoachTeam.jsx
CoachWellbeingHistory.jsx
CoachWellbeingScreen.jsx
Parent
ParentHomeScreen.jsx
ParentSignUp.jsx
LoginScreen.jsx
RegisterationScreen.jsx
LoadingPage.jsx
App.jsx

Endpoints / Server

Our goal is to develop and integrate functionality within the Fins X online platform that allows sports athletes to conduct sport-related assessments using the API provided by the Extended DISC Behavioral Analysis Platform offered by Team Talent Insight Solution. This will enable athletes to perform effective behavioural characteristic assessments through the platform and leverage the resulting data to enhance their individual performance and improve their athletic abilities.

Endpoint

1. **Athlete(GET):**
    - api/Performance: Coach-created performance posts are sent through the API's POST endpoint and stored in the database. Subsequently, the athlete can retrieve the corresponding post information via the GET request
    - api/Mindset: This endpoint searches for all emotion posts by referring to the "Mindset" collection. The athlete uses the GET request to request all emotional posts

Athlete(POST):

- api/athlete: During athlete registration, the API POST endpoint stores personal data in both the "Athlete" and "Users" collections. The "userType" is set to indicate the user's athlete status
- api/login: This endpoint handles user(athlete) logins. The athlete submits the username and password using the POST request.
- api/Notes: This endpoint handles user(athlete) Notes posts. The athlete submits the contents of posts using the POST request

Athlete(Delete):

- api/Notes/:id: This endpoint deletes the note information. When the athlete provides a note's unique Id as a DELETE request, the server deletes the note from the database and returns a success or failure message.

2. **Coach(POST):**
    - api/coach: This endpoint is responsible for coach registration. The user(coach) sends a POST request with the required information, and the server registers the new coach in the database after duplicate checks and returns a successful result
    - api/login: This endpoint handles user(coach) logins. The athlete submits the username and password using the POST request
    - api/Performance: This endpoint stores grade and performance information. The client sends a POST request with the required information. The server validates the request, stores the information in the database, and returns a unique identifier(posted) and a success message.
    - api/Mindset: This endpoint stores uploaded posts based on athletes' emotional status and mental state. The coach sends a POST request with the required information, and the server stores the information in the database and returns a successful result.

Project Tech Stack and Library

In the development of the Athlete Insight app, the following technology stack and libraries are used:

1. **Expo (simulator):**
    - Role: Expo simplifies the development and testing of the Athlete Insight app by providing an emulator. It allows for efficient testing and debugging of the application.
    - Reason: Using Expo's simulator streamlines the development process, enabling developers to save time and resources in the creation of the app.
2. **React Native:**
    - Role: React Native enables the development of a cross-platform mobile application that can run on both Android and iOS devices. This reduces development efforts and costs.
    - Reason: Athlete Insight aims to reach a broad user base, and React Native ensures that the app is accessible to users on various devices.
3. **React Navigation:**
    - Role: React Navigation facilitates the creation of an intuitive and user-friendly navigation structure within the Athlete Insight app, enhancing the user experience.
    - Reason: Effective navigation is crucial for ensuring users can access and use all the features of the app seamlessly.
4. **AsyncStorage:**
    - Role: AsyncStorage is used to store and manage user preferences and data relevant to the Athlete Insight app. It provides a means to persist information between app sessions.

- Reason: To provide a personalized and efficient user experience, the app needs to store and retrieve user-specific data.

5. **DocumentPicker:**
   - Role: DocumentPicker is employed to select and manage documents related to athletes' well-being and insights, streamlining document handling within the app.
   - Reason: Athlete Insight users may need to work with documents and records, making DocumentPicker essential for a seamless user experience.
6. **ImagePicker:**
   - Role: ImagePicker allows users to upload and manage images related to their athletic activities and insights, enhancing the visual aspect of the app.
   - Reason: Incorporating ImagePicker supports user engagement by enabling users to visually document their progress and insights.
7. **JWT (Json Web Token):**
   - Role: JWT is used for secure user authentication and data integrity. It ensures that only authorized individuals can access the app's features and data.
   - Reason: Athlete Insight deals with sensitive information, and JWT helps maintain data security and user access control.
8. **Crypto:**
   - Role: The Crypto library is essential for encrypting sensitive data and ensuring the confidentiality and security of athletes' insights and personal information.
   - Reason: Athlete Insight requires a robust encryption method to protect the privacy of user data and maintain the integrity of insights.

Key Developments

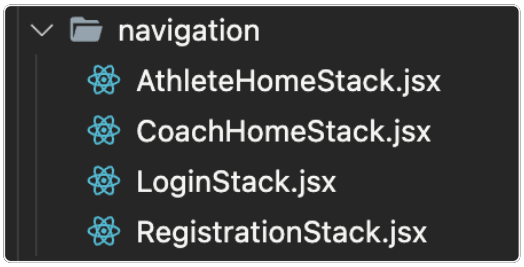
Screen Navigation

For the navigation function for the mobile application "createNativeStackNavigator" and "createBottomTabNavigator" were imported.

```
1  import { createNativeStackNavigator } from "@react-
   navigation/native-stack";
2  import { createBottomTabNavigator } from "@react-
   navigation/bottom-tabs";
```

There are all the navigation of the athlete insight application. they include the most related screens for the stack and tab. Registration and login stack are for all users. However, the limitation of the stack development is that they only have athlete and coach users' screen stacks. It should be developed more for 'parent and 'club' users.

- **Navigation folder**



navigation folder

  - **RegistrationStack.jsx**

```
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import RegisterationScreen from '../pages/RegisterationScreen';
import ParentSignUp from '../pages/Parent/ParentSignUp';
import ClubSignUp from '../pages/Club/ClubSignUp';
import LoginStack from './LoginStack';
import AthleteSignUp from '../pages/Athlete/AthleteSignUp';
import CoachSignUp from '../pages/Coach/CoachSignUp';

const Stack = createNativeStackNavigator();

function RegistrationStack() {
  return (
    <Stack.Navigator initialRouteName="RegistrationScreen">
      <Stack.Screen name="RegistrationScreen" component={RegisterationScreen} options={{ headerShown: false }} />
      <Stack.Screen name="AthleteSignUp" component={AthleteSignUp} options={{ headerShown: false }} />
      <Stack.Screen name="CoachSignUp" component={CoachSignUp} options={{ headerShown: false }}/>
      <Stack.Screen name="ParentSignUp" component={ParentSignUp} options={{ headerShown: false }} />
      <Stack.Screen name="ClubSignUp" component={ClubSignUp} options={{ headerShown: false }}  />
      <Stack.Screen name="LoginScreen" component={LoginStack} options={{ headerShown: false }} />
    </Stack.Navigator>
  );
}
```

```
export default RegistrationStack;
```

- **LoginStack.jsx**

```
import React from "react";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import LoginScreen from "../pages/LoginScreen";
import AthleteHomeStack from "./AthleteHomeStack";
import CoachHomeStack from "./CoachHomeStack";

const Stack = createNativeStackNavigator();
function LoginStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="LoginScreen" component={LoginScreen} options={{ headerShown: false }} />
      <Stack.Screen name="AthleteHomeStack" component={AthleteHomeStack} options={{ headerShown: false }} />
      <Stack.Screen name="CoachHomeStack" component={CoachHomeStack} options={{ headerShown: false }} />
    </Stack.Navigator>
  );
}
export default LoginStack;
```

- **AthleteHomeStack.jsx**
  - The Athlete user screen utilizes both tab navigation and stack navigation on a single screen. "Stack.Screen" is employed for screen alignment within the navigate stack navigator, while "Tab.Screen" functions are used to define the tabs at the bottom of the navigator.

```
import * as React from "react";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import AthleteHomeScreen from "../pages/Athlete/AthleteHomeScreen";
import AthleteMindset from "../pages/Athlete/AthletetMindset"
import AthleteWellbeing from "../pages/Athlete/AthleteWellbeing"
```

```
import AthleteWellbeingUpdate from "../pages/Athlete/AthleteWellbeingUpdate";
import AthletePerformance from "../pages/Athlete/AthletePerformance";
import FinxS_Main from "../pages/Athlete/FinxS/FinxS_Pages/FinxS_Main"
import FinxS_SignInAuth from "../pages/Athlete/FinxS/FinxS_Pages/FinxS_SignInAuth"
import CheckPassword from "../pages/Athlete/FinxS/FinxS_Password/CheckPassword";
import CheckMultiplePassword from "../pages/Athlete/FinxS/FinxS_Password/CheckMultiplePassword";
import AthleteSettingScreen from '../pages/Athlete/AthleteSettingScreen';
import AthleteProfile from '../pages/Athlete/AthleteProfile';
import AthleteSocialScreen from '../pages/Athlete/AthleteSocialScreen';
import { Image } from "react-native";
const Stack = createNativeStackNavigator();
const Tab = createBottomTabNavigator();

function AthleteHomeStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="AthleteHomeScreen"
        component={AthleteHomeBottomStack}
        options={{ headerShown: false }}
      />
      <Stack.Screen
        name="FinxS_SignInAuth"
        component={FinxS_SignInAuth}
        options={{
          tabBarButton: () => null,
          tabBarVisible: false,
          headerTitle: 'Athlete Insight',
        }} />
      <Stack.Screen
        name="FinxS_Main"
        component={FinxS_Main}
        options={{
```

```
            tabBarButton: () => null,
            tabBarVisible: false,
            headerTitle: 'Form',
          }} />
        <Stack.Screen
          name="CheckPassword"
          component={CheckPassword}
          options={{
            tabBarButton: () => null,
            tabBarVisible: false,
            headerShown: false,
          }} />
        <Stack.Screen
          name="CheckMultiplePassword"
          component={CheckMultiplePassword}
          options={{
            tabBarButton: () => null,
            tabBarVisible: false,
            headerShown: false,
          }} />
        <Stack.Screen name="AthleteMindset" component={AthleteMindset} options={{ headerTitle: "Mindset" }} />
        <Stack.Screen name="Wellbeing" component={AthleteWellbeing} options={{ headerTitle: "Wellbeing" }} />
        <Stack.Screen name="Performance" component={AthletePerformance} options={{ headerTitle: "Performance" }} />
        <Stack.Screen name="AthleteWellbeingUpdate" component={AthleteWellbeingUpdate} options={{ headerTitle: "Update Wellbeing" }} />
        <Stack.Screen name="AthleteSettingScreen" component={AthleteSettingScreen} options={{ headerTitle: "Setting" }} />
        <Stack.Screen name="AthleteProfile" component={AthleteProfile} options={{ headerShown: false }} />
      </Stack.Navigator>

    );
}
export default AthleteHomeStack;
function AthleteHomeBottomStack() {
    return (
```

```
      <Tab.Navigator
        initialRouteName="Home"
        screenOptions={{
          tabBarStyle: {
            backgroundColor: "#023B64",
            height: 100,
          },
          tabBarActiveTintColor: "#fff",
          tabBarInactiveTintColor: "#023B64",
        }}
      >
        <Tab.Screen
          name="Goal"
          component={AthleteGoalScreen}
          options={{
            tabBarIcon: ({ focused }) => (
              <Image
                source={
                  focused
                    ? require("../images/Tab_Goal.png")
                    : require("../images/Tab_Goal.png")
                }
                style={{ width: 30, height: 28 }}
              />
            ),
            headerShown: false, tabBarVisible: true,
          }}
        />
        <Tab.Screen
          name="Note"
          component={AthleteNoteScreen}
          options={{
            tabBarIcon: ({ focused }) => (
              <Image
                source={
                  focused
                    ? require("../images/Tab_Note.png")
                    : require("../images/Tab_Note.png")
                }
                style={{ width: 30, height: 28 }}
```

```
            />
          ),
          headerShown: false, tabBarVisible: true,
        }}
      />
      <Tab.Screen
        name="Home"
        component={AthleteHomeScreen}
        options={{
          tabBarIcon: ({ focused }) => (
            <Image
              source={
                focused
                  ? require("../images/Tab_Home.png")
                  : require("../images/Tab_Home.png")
              }
              style={{ width: 30, height: 35 }}
            />
          ),
          headerShown: false, tabBarVisible: true,
        }}
      />
      <Tab.Screen
        name="Social"
        component={AthleteSocialScreen}
        options={{
          tabBarIcon: ({ focused }) => (
            <Image
              source={
                focused
                  ? require("../images/Tab_Social.png")
                  : require("../images/Tab_Social.png")
              }
              style={{ width: 30, height: 28 }}
            />
          ),
          headerShown: false, tabBarVisible: true,
        }}
      />
      <Tab.Screen
```

```
          name="Calendar"
          component={AthleteCalendarScreen}
          options={{
            tabBarIcon: ({ focused }) => (
              <Image
                source={
                  focused
                    ? require("../images/Tab_Calendar.png")
                    : require("../images/Tab_Calendar.png")
                }
                style={{ width: 30, height: 28 }}
              />
            ),
            headerShown: false, tabBarVisible: true,
          }}
        />
      </Tab.Navigator>
    );
  }
  import { createNavigationContainerRef } from '@react-
  navigation/native';
  import AthleteGoalScreen from
  "../pages/Athlete/AthleteGoalScreen";
  import AthleteNoteScreen from
  "../pages/Athlete/AthleteNoteScreen";
  import AthleteCalendarScreen from
  '../pages/Athlete/AthleteCalendarScreen';

  export const navigationRef = createNavigationContainerRef();
  export const navigate = (name, params) => {
    if (navigationRef.isReady()) {
      navigationRef.navigate(name, params);
    }
  };
  export const AppNavigator = () => {
    return (
      <NavigationContainer ref={navigationRef}>
        <Stack.Navigator initialRouteName="AthleteHomeStack"
  screenOptions={{ headerShown: false }}>
```

```
        <Stack.Screen name="AthleteHomeStack" component={AthleteHomeStack} />
        <Stack.Screen name="AthleteHomeBottomStack" component={AthleteHomeBottomStack} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

```

- **CoachHomeStack.jsx**

```
import React from "react";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import CoachHomeScreen from "../pages/Coach/CoachHomeScreen";
import CoachTeam from "../pages/Coach/CoachTeam";
import CoachPerformanceScreen from '../pages/Coach/CoachPerformanceScreen';
import CoachMindsetScreen from '../pages/Coach/CoachMindsetScreen';
import CoachCalendarScreen from '../pages/Coach/CoachCalendarScreen';
import CoachSocialScreen from '../pages/Coach/CoachSocialScreen';
import CoachWellbeingScreen from '../pages/Coach/CoachWellbeingScreen';
import CoachWellbeingHistory from '../pages/Coach/CoachWellbeingHistory';
import CoachProfile from '../pages/Coach/CoachProfile';
import CoachSettingsScreen from '../pages/Coach/CoachSettingScreen';

const Stack = createNativeStackNavigator();
export default function CoachHomeStack() {
  return (
    <Stack.Navigator initialRouteName="CoachHomeScreen">
      <Stack.Screen name="CoachHomeScreen" component={CoachHomeScreen} options={{ headerShown: false }} />
      <Stack.Screen name="CoachTeam" component={CoachTeam} options={{ headerShown: false }} />
```

```
20        <Stack.Screen name="CoachProfile" component={CoachProfile}
    options={{ headerShown: false }} />
21        <Stack.Screen name="CoachCalendarScreen" component=
    {CoachCalendarScreen} options={{ headerTitle: "Calendar" }} />
22        <Stack.Screen name="CoachSocialScreen" component=
    {CoachSocialScreen} options={{ headerTitle: "Social" }} />
23        <Stack.Screen name="CoachPerformanceScreen" component=
    {CoachPerformanceScreen} options={{headerTitle:'Performance'} }/>
24        <Stack.Screen name="CoachWellbeingScreen" component=
    {CoachWellbeingScreen} options={{ headerShown: true ,headerTitle:
    'Wellbeing'}} />
25        <Stack.Screen name="CoachWellbeingHistory" component=
    {CoachWellbeingHistory} options={{ headerShown: true
    ,headerTitle: 'History'}} />
26        <Stack.Screen name="CoachMindsetScreen" component=
    {CoachMindsetScreen} options={{ headerShown: true ,headerTitle:
    'Mindset'}} />
27        <Stack.Screen name="CoachSettingsScreen" component=
    {CoachSettingsScreen} options={{ headerShown: true ,headerTitle:
    'Setting'}} />
28
29      </Stack.Navigator>
30
31    );
32  }
```

FinxS

- Check passwords (CheckPassword.jsx / CheckMultiplePassword.jsx)

  Access code is set which has authentication as Capstone - GLGS : **AUS-SportsApp** it needs to be input code after handover and users can put themselves when they get access code via email

- Generate passwords (GeneratePassword.jsx)

  The passwords that could give users authentication to do the assessment on FinxS page, According to this code, it generates 20 numbers of passwords but it's the administrator's role so we included it as GeneratePassword.jsx in the code, so if there is a case to add the administrator privilege function in the future, the code can be applied.

```
1   import React, { useState, useEffect } from 'react';
2   import { View, Button, Text, Alert } from 'react-native';
```

```
import axios from 'axios';
import AsyncStorage from '@react-native-async-storage/async-storage';

const API_BASE_URL = 'https://finxs.com';
const GeneratePassword = () => {
  const [passwords, setPasswords] = useState([]);
  const [authToken, setAuthToken] = useState(null);
  //load token to access to athlete insight page
  useEffect(() => {
    loadToken();
  }, []);

  const loadToken = async () => {
    try {
      const storedToken = await AsyncStorage.getItem('authToken');
      if (storedToken !== null) {
        setAuthToken(storedToken);
        console.log('Stored Token:', storedToken);
      }
    } catch (error) {
      console.error('Error:', error);
    }
  };

  //generate 20 numbers of the password
  const handleGeneratePasswords = async () => {
    try {
      const response = await axios.post(`${API_BASE_URL}/api/values/passwords/generate`, {
        auth_token: authToken,
        access_code: 'AUS-SportsApp',
        number_of_passwords: 20,
      });

      const responseData = response.data;
      console.log(responseData);
      if (responseData.success) {
        const generatedPasswords = responseData.passwords;
```

```
        setPasswords(generatedPasswords);
        console.log('Generated Passwords:', generatedPasswords);
      } else {
        console.error('Password generation failed:', responseData.message);
        Alert.alert('Password Generation Failed', responseData.message);
      }
    } catch (error) {
      console.error('Error during password generation:', error.message);
      Alert.alert('Error', 'Something went wrong with password generation.');
    }
  };
  return (
    <View>
      <Button title="Generate Passwords" onPress={handleGeneratePasswords} />
      {passwords.length > 0 && (
        <View>
          <Text>Generated Passwords:</Text>
          {passwords.map((password, index) => (
            <Text key={index}>{`Password ${index + 1}: ${password.value}, Link: ${password.link}`}</Text>
          ))}
        </View>
      )}
    </View>
  );
};
export default GeneratePassword;
```

- Upload Password (UploadPassword.jsx)

  The passwords that could be uploaded the passwords in the FinxS page can be used as new passwords. Same as generating passwords, if there is a case to add the administrator privilege function in the future, the code can be applied.

```
import React, { useState, useEffect } from 'react';
```

```
import { View, Button, Text, Alert, TextInput } from 'react-native';
import axios from 'axios';
import AsyncStorage from '@react-native-async-storage/async-storage';

const API_BASE_URL = 'https://finxs.com';
const UploadPassword = () => {
  const [authToken, setAuthToken] = useState('');
  const [accessCode, setAccessCode] = useState('AUS-SportsApp');
  const [passwords, setPasswords] = useState([]);
//load token to access athlete insight page
  useEffect(() => {
    loadToken();
  }, []);

  const loadToken = async () => {
    try {
      const storedToken = await AsyncStorage.getItem('authToken');
      if (storedToken !== null) {
        setAuthToken(storedToken);
        console.log('Stored Token:', storedToken);
      }
    } catch (error) {
      console.error('Error:', error);
    }
  };

  //upoad the passwords (new passwords)
  const handleUploadPasswords = async () => {
    try {
      const response = await axios.post(`${API_BASE_URL}/api/values/passwords/upload`, {
        auth_token: authToken,
        access_code: accessCode,
        passwords: passwords.map(password => password.value),
      });

      const responseData = response.data;
```

```
      if (responseData.success) {
        console.log('Password upload successful.');
        Alert.alert('Password Upload Successful', 'New passwords have been uploaded.');
      } else {
        console.error('Password upload failed:', responseData.message);
        Alert.alert('Password Upload Failed', responseData.message);
      }
    } catch (error) {
      console.error('Error during password upload:', error.message);
      Alert.alert('Error', 'Something went wrong with password upload.');
    }
  };

  return (
    <View>
      <TextInput
        placeholder="Enter passwords, separated by commas"
        onChangeText={text => {
          const passwordArray = text.split(',').map(value => ({ value: value.trim() }));
          setPasswords(passwordArray);
        }}
      />
      <Button title="Upload Passwords" onPress={handleUploadPasswords} />
    </View>
  );
};

export default UploadPassword;
```

- **Email through the form (limitation)**

  Due to the server address limitation, we cannot send an email to 'reports@talentinsightsolutions.com.au', below code is sending email code with available smtp settings.

the endpoint we set : /sendEmail

```
const { smtpServer, tls, starttlsPort, sslPort } = smtpConfig;
    const emailData = {
      firstName,
      lastName,
      email,
      comment,
    };
    fetch( `${serverAddress}/sendEmail`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(emailData),
    })
      .then((response) => response.json())
      .then((data) => {
        if (data.success) {
          console.log('Email sent successfully.');
          alert('Email is sent to TIS');
        } else {
          console.log('Email could not be sent.');
          alert('Try again')
          navigation.navigate('FinxSMain');
        }
      })
      .catch((error) => {
        console.error('Error:', error);
      });
```

JsonWebToken(JWT)

In the given code below, JSON Web Tokens (JWT) is a key element for managing user authentication and authorization. JWT securely communicates and stores user login information to handle authentication and authorization. This allows users to identify themselves and gain access to specific tasks or resources. JWT also uses signatures and encryption to protect the integrity of data and prevent sensitive information exposure.

```
const jwt = require("jsonwebtoken");

// GET Request Handler: Query user information based on tokens
app.get("/api/athletes", async (req, res) => {
  // Extract token from request
  const token = req.token;
  console.log(token);
  // Returns a 401 Unauthorized error if the token does not exist
  if (!token) {
    return res.status(401).json({ message: "Authorization token is required" });
  }
  try {
    // Decrypt the JWT token to extract the username
    const decoded = jwt.verify(token, secretKey);
    const username = decoded.username;
    // Creating MongoDB clients and establishing connections
    const client = new MongoClient(url, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    await client.connect();

    // Database and Collection Settings
    const database = client.db(dbName);
    const collection = database.collection(collectionNameAthlete);
    // Search user information based on username
    const user = await collection.findOne({ username });
    if (user) {
      // Returns 200 OK response when user information is found
      res.status(200).json(user);
    } else {
      // Returns the 404 Not Found error if user information is not found
      res.status(404).json({ message: "User not found" });
    }
  } catch (error) {
    // Output error messages to console and return 500 Internal Server Errors if an error occurs
```

```
      console.error("Error:", error);
      res.status(500).json({
        error: true,
        message: "An error occurred",
      });
    } finally {
      // Close MongoDB client connection
      await client.close();
    }
  });

  // POST Request Handler: User Login
  app.post("/api/login", async (req, res) => {
    // Extract required data from request body
    const { username, password } = req.body;
    // Returns 400 Bad Request error if required field is missing
    if (!username || !password) {
      return res.status(400).json({
        error: true,
        message: "Username and password are required",
      });
    }
    // Creating MongoDB clients and establishing connections
    const client = new MongoClient(url, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    try {
      //Connecting to MongoDB
      await client.connect();

      // Database and Collection Settings
      const database = client.db(dbName);
      const collection = database.collection(collectionNameUser);
      // Search for user information based on username
      const user = await collection.findOne({ username });
      if (user) {
        // If find a user, check for a password match
        if (password) {
          // JWT token creation and return (1 hour valid)
```

```
77          const token = jwt.sign({ username: user.username },
    secretKey, {
78            expiresIn: "1h",
79          });
80          // 200 OK response and token and user data return
81          res.status(200).json({ token, userData: user });
82        } else {
83          // Return 401 Unauthorized Error in Password Mismatch
84          res.status(401).json({ message: "Invalid username or
    password" });
85        }
86      } else {
87        // Returns a 401 Unauthorized error if a user is found
88        res.status(401).json({ message: "Invalid username or
    password" });
89      }
90    } catch (error) {
91      // Output error messages to console and return 500 Internal
    Server Errors if an error occurs
92      console.error("Error:", error);
93      res.status(500).json({
94        error: true,
95        message: "An error occurred",
96      });
97    } finally {
98      // Close MongoDB client connection
99      await client.close();
100   }
101 });
```

Managing Data

The strategy of managing data. How to set the MongoDB to connect and us?

Data Security and Permissions

Guide the data security and permissions for Athlete insight developers. how to store and secure data and how to get the permissions, who gets the permissions.

Test

There are written test codes for the server side. It is based on the Jest and Supertest library used to test the Express.js application. The purpose and method of performing each test suite(group) and individual test are described below.

1. Run the test code.
2. Open the new terminal on the server-side directory.
    a. Type the command "npm install" to update the npm library.
    b. Type the command "npm test" to run the test code.
3. Test code list
    - **POST /api/athlete test:**
        - **Purpose:** Test endpoints (POST /api/athlete) creating new athlete users
        - **How to do it:**

          1.1 Defines an athlete object consisting of valid data

          1.2 Use Supertest to send POST requests to the /api/athlete endpoint

          1.3 Check the response to see if there is a 201 status code (successful generation) and a message ("User created")

          1.4 Retest the field with an incomplete player object that is missing and check for 400 status codes (missing required fields) and messages ("All fields are required")

          1.5 Retest using player objects that conflict with existing user information, and verify that there are 409 status codes (crashes) and messages ("User ready exits")
    - **GET /api/athletes test:**
        - **Purpose:** Test endpoints (GET /api/athletes) that return user information when a valid token is provided
        - **How to do it:**

          2.1 Create a token and send a GET request to the /api/athletes endpoint with valid user information

          2.2 Check your response to ensure that the 200 status code and the correct user name are returned

          2.3 Verify the 401 status code (requires an authentication token) by sending a request to /api/athletes without a token

          2.4 Create a token with incorrect user information and retest, check for 404 status code (user not found) and message ("User not found")
    - **POST /api/coach test:**
        - **Purpose:** Test endpoints (POST /api/coach) that create new coach users
        - **How to do it:**

          3.1 Defines a coach user object that consists of valid data

          3.2 Use Supertest to send POST requests to the /api/coach endpoint

3.3 Check the response to see if there is a 201 status code and message ("User created")

3.4 Retest using Coach user objects that conflict with existing user information, and check for 409 status codes and messages ("User ready exits")

- **POST /api/login test:**
  - **Purpose:** Perform user authentication and test endpoints (POST /api/login) that return tokens and user data
  - **How to do it:**

    4.1 Send POST requests to the /api/login endpoint with valid user credentials

    4.2 Check the response to see if 200 status codes, tokens, and user data are returned

    4.3 Send a request with invalid user credentials to verify that you have a 401 status code (an invalid username or password) and a message ("Invalid username or password")
- **POST /api/Notes test:**
  - **Purpose:** Test the endpoints (POST /api/Notes) that create new notes
  - **How to do it:**

    5.1 Sends POST requests to the /api/Notes endpoint with valid note data

    5.2 Check the response to see the 201 status code and message ("Note created")

    5.3 Retest the required fields with incomplete note data that is missing, and check for 400 status codes (missing required fields) and messages ("All fields are required")
- **DELETE /api/Notes/:id test:**
  - **Purpose:** Test the endpoint (DELETE /api/Notes/:id) that deletes notes using the given Id
  - **How to do it:**

    6.1 Send the DELETE request to the /api/Notes/:id endpoint using the note ID existing in the physical database

    6.2 Check your response to see if you have 200 status codes and messages ("Note deleted")

    6.3 Retest with a note ID that does not exist, and verify that you have a 404 status code (note not found) and a message ("Note not found")

    6.4 If you miss an ID, make sure you have a 400 status code (Note ID required) and a message ("Note ID is required")
- **POST /api/Mindset test:**
  - **Purpose:** Test endpoints (POST /api/Mindset) that create new mindset posts
  - **How to do it:**

7.1 Send POST requests to /api/Mindset endpoints with valid mindset post data

7.2 Check the response to see if there is a 201 status code and message ("Mindset post created"). Also, verify that the ID of the new post is returned

7.3 Retest with incomplete mindset post data missing required fields, and make sure you have 400 status codes (missing required fields) and messages ("All fields (emotion, date, mindset content) are required")

- **GET /api/Mindset test:**
  - **Purpose:** Test endpoints (GET /api/Mindset) that return a list of mindset posts
  - **How to do it:**

    8.1 Send a GET request to the /api/Mindset endpoint

    8.2 Check the response to see if data in the form of 200 status codes and arrangements is returned
- **GET /api/Performance test:**
  - **Purpose:** Test endpoints (GET /api/Performance) that return a list of performance posts
  - **How to do it:**

    9.1 Send the GET request to the /api/Performance endpoint

    9.2 Check the response to see if data in the form of 200 status codes and arrangements is returned
- **POST /api/Performance test:**
  - **Purpose:** Test the endpoints (POST /api/Performance) that create performance posts
  - **How to do it:**

    10.1 Send POST requests to the /api/Performance endpoint with valid performance post data

    10.2 Check your response to see if you have 201 status codes and messages ("Performance post created"). Also, check if the ID of the new post has been returned

    10.3 Retest with incomplete performance post data missing required fields, and check for 400 status codes (missing required fields) and messages ("All fields are required")

These tests are used to verify the operation of multiple endpoints and the Express.js application and to verify that requests and responses are as expected. Each test identifies the different functions of the application and is tested for exceptions. Supertest makes it easy to simulate HTTP requests and responses, helping to ensure code reliability and accuracy.

Collaboration work guide

1. File name : "User type" + "Feature" (example: "AthleteHome)
2. Adding related screens in a single stack with navigation on one screen.
3. To efficiently define and apply styles for managing and reusing common styles in React Native, StyleSheet is used.
4. To enhance the quality of the development project and promote transparency in collaboration, it's important to communicate and share information with team members regarding changes that arise during the process of code modifications, enhancements, or development. Depending on the needs, using a platform like GitHub can facilitate collaboration and streamline source code management.