Abstract

The goal of this project is to better understand how to use React Native and APIs to leverage the data available from Fitbit API so we can improve a given user’s ability to reach the heuristic goals they set for themselves. We will be using information available on the user’s account to better help them gauge their performance increase over time, using general trends in their profile to show positive or negative improvement. This project is made using React Native, a native application engine developed by Facebook, to build the general application architecture and Auth0, an online service that connects the current application to any number of social and database applications (ex: Fitbit). The application itself consists of several internal pages which either: allows the user to log into their Fitbit account, hosts profile information from the user, or allows the user to update the goals that they’ve set.

We managed to use Auth0 to authenticate the users Fitbit information and draw the relevant data points onto a profile page that is available on log-in. The final report describes in further detail the motivation behind the project, background on the technologies used, screenshots of the final result, and future steps to further enhance the final product.

Elevator Pitch/Narrative

Jeremiah decided to purchase a Fitbit to help himself train for a marathon that he has in 4 months. He is in decent shape but is nowhere near his goal of a 4hr 10minute mile. Originally, Jeremiah would only be able to track the number of steps he took, his heart rate during exercise, and other corollary statistics that he could see at the end of the day. Unfortunately, this doesn’t help Jeremiah unless he is a fitness professional trained to use this information to advance his skills.

Luckily, Jeremiah got the Fitbit Coach with his device. He now has a data-driven trainer that uses high-precision heart rate tracking, step measurements, and sleep analytics of both himself and others with a similar body profile to give recommendations on how to
improve. Even better, the trainer keeps track of these metrics throughout the day, along with user inputted information on food consumption, water consumption, and weight.

Now Jeremiah can get feedback on primary goals (daily activities) and on secondary goals (longer term activities) with extreme customization to his day-to-day life.

React Native

Background information on React Native and implementation specifics.

React Native lets you build mobile apps using only JavaScript and the enhanced JSX. Similar to React, it lets you compose a rich mobile interface from declarative components, which means that you no longer need to build mobile web apps, HTML5 apps, or hybrid apps to reach the native application customer base.

One fundamental part of the React Native architecture is that there is a single rendering entry point. This basically means that a render function is required within all React components and that the render function is key to what we see in the application. The rendering function is not written with DOM elements, it is written in JSX (a JavaScript syntax extension which mixes HTML/XML-like syntax directly into the JavaScript code). This is a useful feature to have because the only place to render components is inside of the render function which adds more organization to the application itself.

```jsx
import { Navigator } from 'react-native';

class FitCoach extends Component {
  render() {
    return (
      <Navigator
        style={styles.navigator}
        initialRoute={{ name: "Welcome" }}
        renderScene={this.renderScene}
        navigationBar={
          <Navigator.NavigationBar
            style={styles.nav}
            routeMapper={NavigationBarRouteMapper} />
        }
    );
  }
}
```

React also allows for properties (props) to help users build out the components themselves. In order to move data through the application, we must pass the property values directly to a new component, and that component will have access to the properties that you allocated towards it. From this description, we can see that the default data flow is in one direction, propagating in a tree-like structure where parents give to children. An example from my project shows how this is done using the native Navigator component:
The Navigator provides a JavaScript implementation of a navigation stack which will function on both iOS and Android. It can be adapted to render different components based on provided routes, and any props should be passed to the next component using passProp (as shown above). Implementing this Navigator is simple, it just requires that the developers define and compartmentalize their application into screen components.

```javascript
this.props.navigator.push({
  name: 'Profile',
  passProps: {
    profile: profile,
    token: token,
  }
});
```

The styling that is used in React Native Stylesheets is slightly different than normal CSS that web developers are used to. Specifically, Flexboxes and FlexDirections are commonly used to organize components within individual pages of React Native applications. To note differences in syntax and naming, please refer to this cheat sheet created by Github user vhpoet:

https://github.com/vhpoet/react-native-styling-cheat-sheet#text

Finally, in order to run a React Native application, the developer will need to run a virtual machine of the device they wish to test (this project had a focus on Android). Once the virtual machine is running, developers will need to run “react-native start” from one terminal, then “react-native run-android” or “react-native run-ios” from another terminal in order to properly compile the React Native code into the respective languages.

Auth0

AUTHENTICATION ENGINE AND THE KEY TO ACCESSING FITBIT INFORMATION FOR THIS PROJECT

In order to properly authenticate users, we decided to utilize a Single Sign On (SSO) login: basically, the scenario when a user logs into an application with a single set of credentials and is then automatically signed up into multiple applications. SSO logins work as follows:
The simple take-away concept is that there is one central domain through which authentication is performed, and then the session is shared with other domains in some secure way e.g., a signed JSON Web Token (JWT). With Auth0, SSO login is very simple as Auth0 provides out-of-the-box support for many different applications, including Fitbit.

In order to properly use Auth0 with the React Native application, developers will need to install `react-native-lock` so that the application will have access to the log-in screen that we will show soon. For more installation specific, please see this link which describes how to get your React Native environment set up for Auth0 integration. The most important part of the process is the `lock.show(...)` function which brings up the log-in screen and responds with profile information if the clientId and clientSecret both are approved.

Due to the fact that the user authenticates their identity with their Fitbit account, we not only authorize them to use our application but we gain access to their Fitbit information as a result of this. Moving forward, we will discuss the Fitbit API, its limitations, and its capabilities.

**Fitbit API**

*Main source of user information and the basis of goal improvement*

The Fitbit API is a plentiful resource that provides us with a diverse set of information, from activity logs, heart rate logs, and profile logs to food logs, sleep logs, and personal
achievement logs as well. The first challenge in accessing this information is completing
the OAuth 2.0 authentication which requires your application to obtain an Access Token
when the Fitbit user authorizes your app to access their data. The Access Token is used
for making the HTTP request to the Fitbit API. Luckily for us, this OAuth2.0 authentication
is already completed by the Auth0 service, so after implementing and incorporating
Auth0 into our application, we are able to complete OAuth 2.0 verification very easily.
When you complete this step through Auth0, you are provided with a verification page
and, when confirmed, a JSON file with the information from that specific user’s profile.

To better understand the what information comes from the JSON response, please reference this link which has more details on the return information. While engaging in
the authentication process, developers may run into some issues when dealing with the
JSON Web Tokens (JWTs). These are basically an open standard that defines a system of
transmitting information between parties as a JSON object. Many questions can be
answered by referencing this link.

While Auth0 manages to handle the initial authentication with the Fitbit server, in order
to access more information from Fitbit, the developer will need to use the fetch() function in order to request more information from the Fitbit API. I have attached
information on how to implement an API call, with more information (on this specific
type of call) available here:
More specifically, this is a JavaScript function `_onCallApi()` that gets called whenever a user hits the “Get Info” `TouchableHighlight` component (basically the same as an HTML Button). Example code for implementing that is also available below:

```javascript
_onCallApi: function() {
    fetch('https://api.fitbit.com/1/user/-/profile.json', {
        method: "POST",
        headers: new Headers({
            'Authorization': 'Bearer ' + this.props.token.idToken,
            'Content-Type': 'base64',
        }),
    })
    .then((response) => response.text())
    .then((responseText) => {
        console.log(responseText),
        Alert.alert(
            'Request Successful',
            'We got the secured data successfully',
            [
                {text: 'OK'},
            ]
        ))
    .catch((error) => {
        Alert.alert(
            'Request Failed',
            'Please download the API seed so that you can call it',
            [
                {text: 'OK'},
            ]
        );
    });
}
_onCallClick: function()

Utilizing this kind of strategy, the developer can update the information on the React Native application with continuous refreshes – just be careful that your tokens do not refresh/expire before you expect them to!

**Results**

What were the outcomes of the project?
While attempting to build out the original application, we ran into several hurdles which needed to be overcome. Initially, we (Ruzica and I) discussed the possibility of using the Fitbit API to collect anonymized information from a large set of users to help the application set goals on its own, using demographics and personal information of our user to decide whether they were reaching their goals at the right pace. Unfortunately, we were unable to access this anonymized data. This was disappointing because we felt that there could be many interesting insights that can be drawn by helping users create goals based on the performance of others who share similar physiques and health statistics.

Another hurdle was reached when we began to attempt to call Fitbit using the tokens generated with Auth0. We were able to collect general profile information and authenticate users using their Fitbit information, but for some reason, there was an invalid key response from the Fitbit API as we attempted to collect the user’s activity logs and all further logs from the API. A sample output with the token removed is shown below:

```
05-03 16:33:29.829  6916  6944 I ReactNativeJS: {
"errors":[{"errorType":"invalid_token","message":"Access token invalid: <show token here>. Visit https://dev.fitbit.com/docs/oauth2 for more information on the Fitbit Web API authorization process."}],"success":false}
```

Another challenge that we ran into was effectively saving information into the React Native component “AsyncStorage.” I intended to use AsyncStorage to hold information on the user at hand, including information on their goals and general profile information that we were able to collect through Fitbit. Unfortunately, when I attempted to utilize this feature, I would encounter the following error while trying to submit some data to the storage:

```
undefined is not a function (evaluating _this._appendMessage(AsyncStorage error: +error.message))
```

Moving forward, I would work to get AsyncStorage and the Fitbit tokens working because these two would make the application far more powerful and useful for the users. These are two key parts that would allow the user to really learn and enhance their exercise experience with the Fitbit.
Acknowledgement

I would like to give special thanks to Professor Ruzica Piskac for her insights in ideating and iteration on the project. She provided support during the process and let me explore an interesting new area of computer science which I very much appreciate.

References

1. React Native Stylesheet Cheat sheet
   https://github.com/vhpoet/react-native-styling-cheat-sheet#text
2. Facebook React Native Tutorial
   https://facebook.github.io/react-native/docs/tutorial.html
3. Auth0 QuickStart for React Native Applications
   https://manage.auth0.com/#/clients/KUgduJfoNWzcxT3XMSsmr068op5JnMK0/quickstart
4. Auth0 Single Sign On (SSO) Tutorial
   https://manage.auth0.com/#/clients/KUgduJfoNWzcxT3XMSsmr068op5JnMK0/quickstart
5. Auth0 User Profile Structure
   https://auth0.com/docs/user-profile/user-profile-structure
6. JSON Web Tokens
   https://auth0.com/docs/jwt
7. Understanding fetch()
   https://davidwalsh.name/fetch
8. Fitbit API Documentation
   https://dev.fitbit.com/docs/user/
9. TouchableHighlight React Native Component Documentation
   https://facebook.github.io/react-native/docs/touchablehighlight.html
10. React Native AsyncStorage
    https://facebook.github.io/react-native/docs/asyncstorage.html