Exercise Tracker

Name: Ian Liu

I. Introduction

This application is a daily exercise tracker designed to help users monitor their exercise routines and maintain a summary of their exercise history. Users can set their desired daily calories goal and input their daily exercise activities. The application tracks the calories burned based on exercise type and duration, providing a comprehensive summary for each day. The user's data can be securely backed up to and restored from cloud storage.

In this application, a "session" refers to a day, and individual exercises are called "subsessions." The primary objective of this application is to track the calories burned in relation to the user's daily calories goal, providing valuable information such as calories burned, remaining calories to reach the goal, and estimated time required.

II. Problem Statement

The application is a useful tool designed to track your daily exercises and assist you in reaching your calorie goals. It can be challenging to keep track of the duration of our workouts and how much more we need to do. Additionally, knowing the amount of time spent exercising is not enough; we often desire a tool to convert those hours into calories, providing a better understanding of our progress. This application is specifically designed for individuals who are dedicated to achieving their goals, improving their health, and maintaining a fit physique. Moreover, it offers a history tab and backup function to ensure that your progress is saved and easily accessible.

III. Design and Implementation

The application offers various features to enhance user experience. It provides valuable information such as weather updates and motivational quotes. Additionally, users can store images and personal exercise data both locally and in the cloud storage for backup and easy access. The application boasts a user-friendly interface and includes convenient interactive functions, such as the ability to perform actions by simply shaking the device.

The application consists of three main pages:

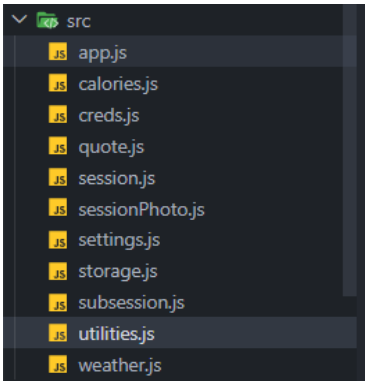
Home Page: This serves as the main page, displaying the current weather conditions and motivational quotes. It also shows the user's daily calories goal and provides a summary of the current day's exercise progress.

Session Page: Users can initiate a new exercise session on this page. They can choose the exercise type, start or end different exercises, and track the calories burned. The page displays a summary of the current day's exercise progress, allows users to take a picture to document their activities, and provides access to detailed exercise information. The application supports shake detection to facilitate starting or ending an exercise session.

History Page: This page enables users to review their past exercise sessions, including information and images from previous days. Users can also back up their exercise history to the cloud storage, restore previously saved data, and delete individual history entries. This functionality simulates the backup and restore features commonly found in modern applications.

The most commonly used native components in my application are Text, TextInput, View, FlatList, Animated, and several others. I built custom components on top of these and combined them to create the application. To facilitate the communication and functionality between components, I heavily relied on native hooks like useEffect and useRef, as well as native APIs such as Props.

In order to keep the code organized and maintainable, I separated the components into files based on their category. This allows for easier management and scalability of the codebase. Here is a screenshot of the file directory:



The application makes use of various external APIs to retrieve information and services from the internet. Here are the APIs utilized:

OpenWeather: This API provides weather information for the current location.
Quotable: This API is used to fetch motivational quotes.
Calories Burned API - API Ninjas: This API allows retrieving the calories burned per hour for specific activities.
AWS DynamoDB API: This API facilitates cloud storage operations.
In addition to external APIs, the application also utilizes internal APIs provided by certain packages. Here are some examples:

Axios: This package offers an HTTP client for making network requests.
Geolocation: It provides geolocation information for the current position.
Toast Message: This API displays toast messages on the screen.
Shake Detection: This API detects shake events from the user.
Image Picker: This package enables the functionality of capturing pictures.
Async Storage: It allows storing data in the local storage.
React Navigation: This package facilitates page/screen navigation within the application.
By leveraging these external and internal APIs, the application is able to fetch data, perform actions, and enhance the user experience.

In the development of the application, I faced several challenges and I manage to come up with solutions:

Challenge: Sorting the source code and avoiding writing everything in one file for the larger application.
Solution: To address this, I utilized JavaScript modules and separated the source code into multiple files. By doing so, I improved the organization and maintainability of the codebase. Additionally, I leveraged the props attribute of React Native components to pass data between different components.

Challenge: Ensuring smooth navigation between the three pages of the application.
Solution: To achieve seamless navigation, I integrated the react-navigation package into the application. This package provided a user-friendly solution for handling page transitions and ensuring a smooth user experience.

Challenge: Converting past codes into reusable components for the final project to avoid repetitive work.
Solution: To overcome this challenge, I reviewed previous assignments, identified useful code snippets, and transformed them into reusable components. I ensured the components could be integrated effectively and retained their functionality.

Challenge: Finding a free alternative API for motivational quotes since the original API was not free.
Solution: I discovered a free API called Quotable that could be utilized for the application. Quotable not only offered motivational quotes but also provided parameters to filter the quotes based on specific criteria.

Challenge: Storing data locally to prevent data loss when the application is closed.
Solution: I identified and implemented the async storage package, which allowed for easy and efficient local data storage within the application.

Challenge: Converting pictures taken from the camera into a string format for easier integration with AWS DynamoDB.
Solution: To avoid the need for additional services like AWS S3, I converted the pictures to base64 format and stored them as strings. Fortunately, the react-native-image-picker package offered a built-in option for converting photos to base64 format. The React Native Image component was then used to display the images by accepting base64 format strings.

Challenge: Limited support for activities in the Calories Burned API.
Solution: I manually selected a set of common activities and hard-coded them into the application. Care was taken to ensure that the chosen activities were distinct and not overly similar to one another.

Challenge: Performing calculations on activity duration and calories burned while ensuring accurate data display.
Solution: I dedicated time to verifying the accuracy of the calculations and ensuring proper data display. Special attention was given to correctly updating the data when the user modified the activity duration. Integration of the data with React Native components required careful handling to ensure appropriate updates.

Challenge: Compatibility issues with the react-native-sensors package in relation to the latest version of Gradle.
Solution: Instead of attempting to fix the outdated react-native-sensors package, I opted to use an alternative package called react-native-shake to detect shake events. This new package proved to be more reliable and easier to implement.

Challenge: Backing up data by converting it into a suitable format for AWS DynamoDB and designing the table format.
Solution: To address this challenge, I used the JSON stringify function to convert the data into a string format suitable for storage. The converted data was then stored in local storage and easily uploaded to DynamoDB as a string. When downloading the data, the string was stored back into local storage. Additionally, I categorized the data based on the components used to display it.

Challenge: Rerendering components after updating the state of the current component.

Solution: To enable communication between components and facilitate rerendering, I utilized the props attribute to pass data between components. Rerendering of components was achieved by utilizing the useEffect hook to update components when the state of the current component changed. To prevent unnecessary rerendering, the useCallback hook was also employed.

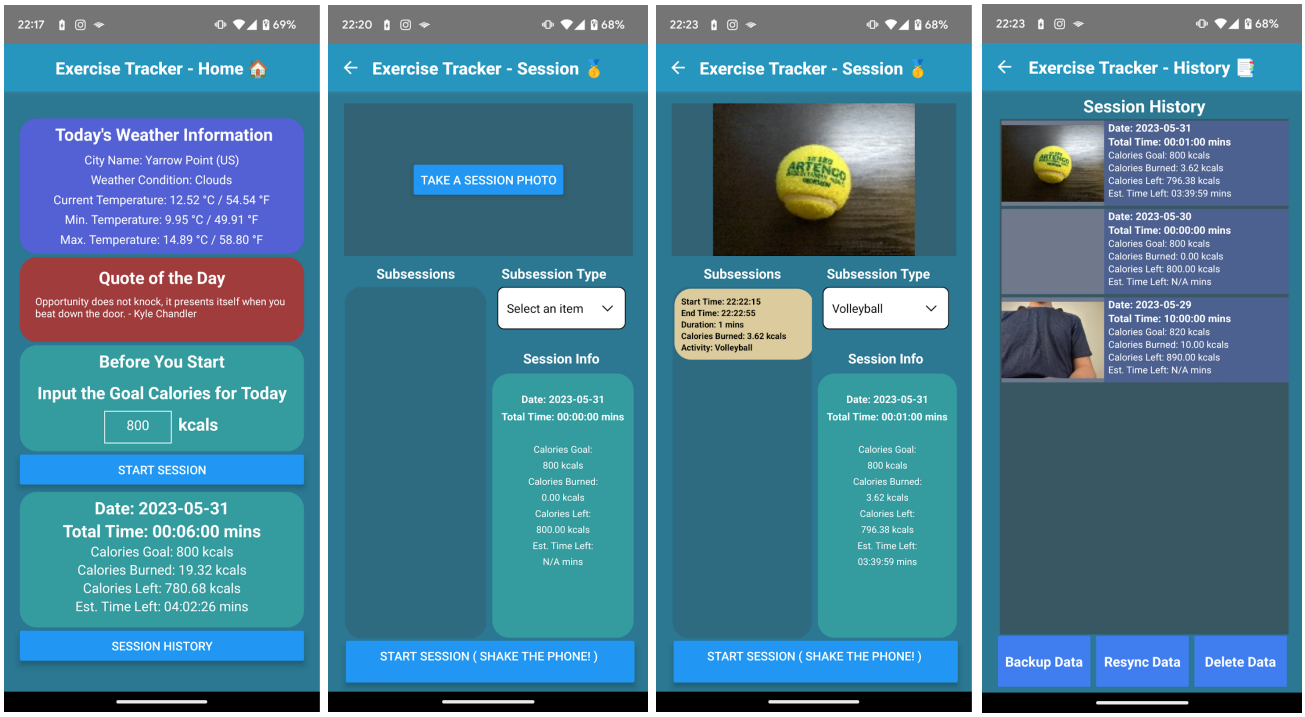
Challenge: Meeting the requirement of an interactive user experience with visual effects.
Solution:

1. To enhance interactivity, I focused on the arrangement of widgets and incorporated toast messages to provide feedback to the user. Additionally, I integrated shake detection, allowing users to trigger specific features by shaking their phone.
2. To provide visual effects, I implemented animations using the Animated package. Items in a ListView were given a fade-in effect upon loading or addition, resulting in a more engaging user experience.

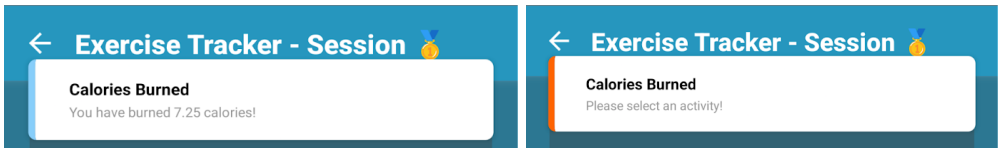
IV. Minimum UI Requirements

The application offers a wealth of information through its widgets, which include comprehensive statistics and external API data. Navigation is made simple by minimizing the number of buttons used in the user interface (UI). Additionally, I have incorporated a toast message function to display relevant information and error messages in a visually pleasing manner. The icons, fonts, and colors have been selected by me to create an attractive and user-friendly interface.

The UI of the application in test version:



Example of toast messages:



Example of a session that reached the goal calories:

The application is optimized for portrait mode, but its UI is adaptable to different screen sizes thanks to the use of percentage-based height and width settings in the stylesheet. Additionally, the application includes a feature to capture photos. To ensure the best display experience, the application prompts the user to retake a photo in landscape mode if it was initially taken in portrait mode. This ensures that the display area is maximized for landscape-oriented photos.

V. Additional Features

1. The application incorporated four external APIs, including a cloud storage service, to enhance its functionality.
2. The application implemented Animation on View that does fade-in effect.
3. The async storage package was leveraged to store data locally, a valuable feature for real-world applications.
4. The application integrated the camera, geolocation API, and sensors, such as shake detection, to provide additional functionality and showcase a way to combine the skills that were learned through the course.
5. The components were thoughtfully separated into widgets and organized into pages, improving the overall structure and maintainability of the application. The project serves as a demonstration of React Native skills and showcases the design of an expandable real-world application.
6. The source code utilized the Detox test suite, which was used in the class. The test cases are created to ensure some critical functions of the application.

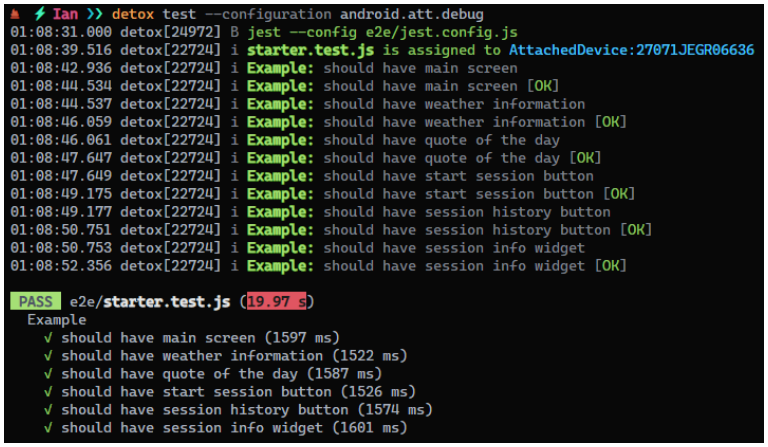
VI. Testing and Evaluation

The application testing process consists of several stages. To create a real-world testing environment, I utilized my Google Pixel 6a phone to directly run the application. I chose to use a physical device throughout the entire process to ensure that the application is tested under real-world conditions.

On the software side, the application was divided into three screens and various widgets. Initially, I developed and tested the individual widgets, as they are smaller components and relatively easier to set up. However, the screens posed a greater challenge due to their complexity, incorporating multiple interconnected widgets. Integrating the screens required more extensive testing and proved to be more difficult. Nonetheless, I successfully conducted thorough testing of the application to ensure its proper functionality.

During the application testing phase, I encountered numerous errors arising from a lack of synchronization between components. This allowed me to identify and address several bugs, enhancing the data connection between the components. Additionally, integration testing revealed that several UI components were not functioning as expected. Consequently, I returned to the widgets and resolved these issues. After resolving the bugs, I retested the application to confirm its proper operation.

In addition, I implemented tests using JEST and utilized Detox for application testing. Given the imminent deadline for the final project, my focus was primarily on verifying the presence and visibility of UI components on the main screen. Although the tests were relatively basic, they effectively showcased the capability of the testing suite package for testing React Native applications. Here are the results of the tests:



VII. Conclusion

The final project showcases my understanding of React Native and its capabilities through the development of a fitness application. This app enables users to track their daily activities and calories burned. Additionally, it leverages third-party APIs to provide motivational quotes and weather information, encouraging users to engage in regular exercise. To enhance functionality, I integrated geolocation and camera features, making use of the phone's built-in hardware. Shake detection is another notable feature that demonstrates the utilization of existing packages to expand the application's capabilities. Lastly, AWS DynamoDB was employed for data storage, showcasing the implementation of cloud services.

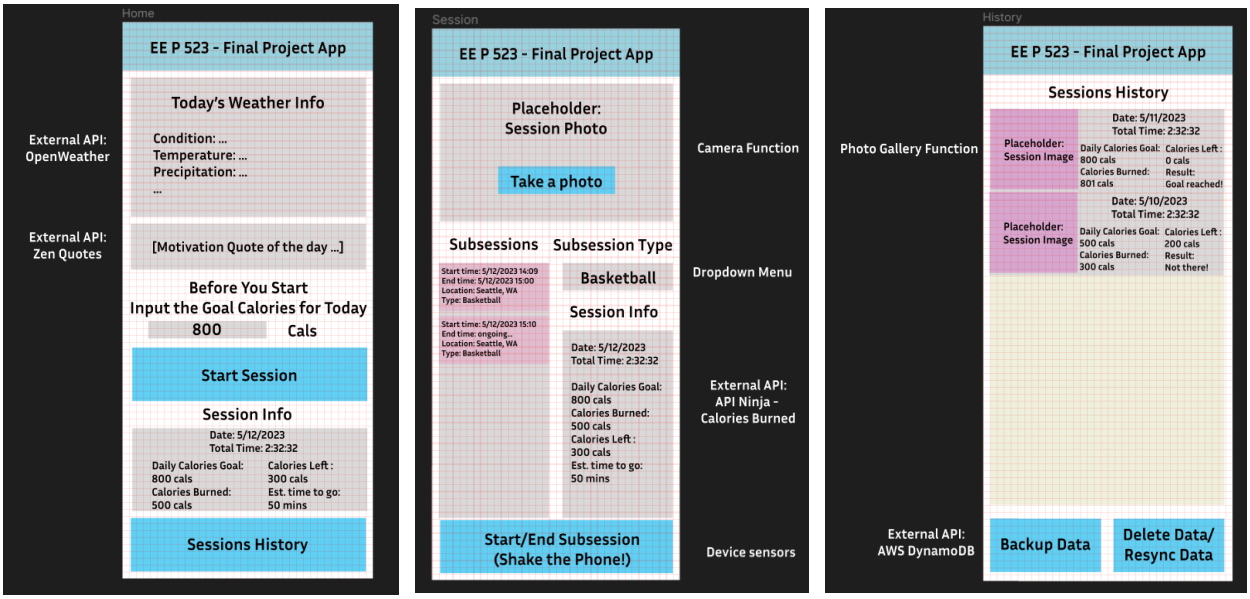
The application is designed with expandability in mind, allowing for easy modifications and the addition of new features. Major components are coded as reusable widgets that can be imported whenever needed. The code is well-organized into multiple files, enhancing readability and maintainability. Moreover, a comprehensive test suite is included to ensure code quality.

Throughout the project, I learned the power of React Native in creating cross-platform applications. By building the application with React Native, it can be deployed on both iOS and Android devices. Working with internal APIs on the hardware was an enjoyable experience, complementing my prior experience with external APIs. This opens up exciting possibilities for future mobile projects.

Moving forward, there are several areas for improvement in the application. Adding a workout planner feature that leverages more APIs could provide comprehensive suggestions to users. Implementing an authentication system would enable users to store their data in the cloud and access it from multiple devices. Currently, the app only supports one user, but with authentication integration, it can be expanded to accommodate multiple users.

VIII. Figma

https://www.figma.com/file/ohhIbaX0soSQKjQmQpeVot/Final-Project?type=design&node-id=0-1



X. References

Documentation

- React Native - Introduction - https://reactnative.dev/docs/getting-started
- React Native - Components - https://reactnative.dev/docs/components-and-apis
- React Native - APIs - https://reactnative.dev/docs/accessibilityinfo

Packages & Examples

- Axios - https://axios-http.com
- Axios - Minimal Example - https://axios-http.com/docs/example
- @react-native-community/geolocation - https://github.com/michalchudziak/react-native-geolocation
- React-native-toast-message - https://github.com/calintamas/react-native-toast-message
- React Native Sensors - https://github.com/react-native-sensors/react-native-sensors
- React Native Shake Event Detector - https://github.com/Doko-Demo-Doa/react-native-shake
- React-native-image-picker - https://github.com/react-native-image-picker/react-native-image-picker
- @react-native-async-storage/async-storage - https://github.com/react-native-async-storage/async-storage
- React Navigation - https://reactnavigation.org/docs/getting-started/

External APIs

- OpenWeather - https://openweathermap.org/api
- Zen Quotes - https://zenquotes.io/
- Quotable - https://github.com/lukePeavey/quotable
- Calories Burned API - API Ninjas - https://api-ninjas.com/api/caloriesburned
- AWS SDK for JavaScript v3 - https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html
- DynamoDB examples using SDK for JavaScript (v3) - https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/javascript_dynamodb_code_examples.html

XI. Appendices

Time spent:

- Application design and API research: 2 hours
- Application Figma design: 1 hour
- Development package and test suite setup: 1 hour
- Integrating async storage and moment library: 1 hour
- Main/Subsession/History components and screen options: 1 hour
- Weather widget: 1.5 hours
- Quote widget: 0.5 hour
- Calories Setter widget: 1 hour
- Subsession widget: 2 hours
- Session widget: 2.5 hours
- Sensor for shake detection: 1 hour
- AWS DynamoDB setup: 0.5 hour
- AWS Backup function: 1 hour
- Backup widget: 1 hour
- Session History widget: 1.5 hours
- UI/UX enhancement/Animation and transition: 1.5 hours
- Integration Testing: 2 hours
- Test code/Detox Testing: 1.5 hours
- Project README file: 1 hour
- Project report: 1.5 hours
- Project demo video: 1.5 hours

Total: 27 hours