**Health at Hand - Front End Documentation**

Author: Vishal Karande and Josh Koshy

Because of the limited features requested for this project, we’ve utilized React Native for development. We also decided to use the Expo library for key development tools. React Native Redux is also required in order to understand the state management behind this application.

**To run:**

Install Node.js

Move to the root directory (HealthAtHand/React Native Application)

Run “npm install -g expo-cli” (If this version of the Expo-cli does not work, we were using expo-cli@2.6.0)

Run “npm install” to install all project dependencies listed in package.json

Run “expo start”

This will start Expo and will enable you to download and run the application through the given IP address that will be displayed.

Important note: The IP addresses utilized in the pages listed in the Actions folder should all point to wherever you decide to host our server code. Failure to change this will make the application not function properly.

For the current development of this project, we were hosting the server on our local public IP address, meaning that ran the Flask server locally. To do so:

Move to the root directory of the server code (HealthAtHand/Flask Server)

Run the flask server through your public IP address:

Run “export FLASK\_APP=healthathand.py”

Run “flask run --host 0.0.0.0”

The IP address in the pages under the Actions directory should be changed to point to your public IP address. This will allow the application to properly communicate to the server through your network.

**Actions (src/actions):**

The following actions are tied to the functionality of react-native-redux and is a standalone state management library. These actions are referred to and help to transfer data around the application.

**Name:** AuthActions.js

**Description:** Action used to authenticate user credentials

**Name:** EmailAction.js

**Description:** Stores email in terms of email body and subject in database

**Name:** ExerciseNotesActions.js

**Description:** Logs users daily exercise notes to the database

**Name:** FoodActions.js

**Description:** Action used to get and store all user meals.

**Name:** FoodNotesActions.js

**Description:** Logs users daily food notes to the database

**Name:** ReportActions.js

**Description:** Action used to get and post user report from the database

**Name:** SearchAction.js

**Description:** Action used to get results from nutrition IX according to the search term provided.

**Name:** types.js

**Description:** Maps all actions to the corresponding reducer.

**Components (src/components/common):**

These are standalone components within the app comprised on react-native-element components and are utilized for ease of development.

**Name:** HaH\_Header.js

**Description:** This component is displayed at the top of every page. You can attach an icon to the left and right of this component, and you can change the title that will be displayed in the middle.

**Name:** HaH\_NavBar.js

**Description:** This component is displayed at the bottom of every page. There are five icons that are to be displayed, each redirecting to Home.js, MealLog.js, ExerciseLog.js, Report.js, and Messenger.js, respectively from left to right. When adding this component to any page, you can edit the “selected” field to change whichever icon you wish to display a gold color instead, to indicate that is the page that is currently selected.

**Pages (src/pages):**

These pages are the ones targeted for rendering. The Router.js page will navigate and redirect the user to these pages. All of these pages below utilize the Redux state management system to pass information to one another as they transition. Some many dynamically change depending on the flags set when entering the page.

**Name:** Email.js

**Description:** This page will either display an email or let the user create a new email depending on how the page was reached by the user from the Messenger.js page. There are two fields that are designated for the subject and body of the message associated with the page. If there is information loaded from an existing email to display, it will fill these fields for the user. There is an additional confirmation button at the bottom of this page that will only display if the user creates a new email. If pressed, it will redirect to the user’s own emailing app on their phone with the message’s subject, body, and the health coach’s email (the only person the user should be emailing through the app).

**Name:** ExerciseCard.js

**Description:** This page is the main display page for any exercise item that the user specifically wants to add or edit. If the user reaches this page through the ExerciseSearch.js page, the page will be properly formatted to show that the user is adding the exercise. If the user reaches this page through the ExerciseLog.js page, it will be properly formatted to show that the user is editing the exercise. There is a main title card for the food item, as well as the food items corresponding information. The duration and the intensity of the exercise can be set by the user. This confirmation button will redirect the user back to the exercise’s corresponding ExerciseLog.js page with the exercise information pushed onto the exercise object array stored there.

**Name:** ExerciseLog.js

**Description:** All exercises available for the listed day will be displayed here. There is a TouchableOpacity displayed at the top that initially displays the current day. The user can change the date through this TouchableOpacity and display the exercises attached to that day. There is an icon displayed on the top right that redirects the user to an ExerciseSearch.js page. Any existing exericses that were logged previously will be loaded through the loadData function and displayed through a FlatList of TouchableOpacitys, each tied to a specific ExerciseCard.js page with individual data. When pressing any of these TouchableOpacitys, the data tied to each exercise is passed through. There is also a TouchableOpacity for ExerciseNotes.js that displays at the bottom of the page if the user is still on the current day. It will disappear if the user redirects to an exercise log of a different day.

**Name:** ExerciseNotes.js

**Description:** This page is to log any meal notes for the user’s exercises for that current day. The majority of the screen is filled with a TextInput which allows the user to input their note. If a note already exists for that day, it will be loaded into the page and the title and confirmation button’s text will change accordingly.

**Name:** FoodCard.js

**Description:** This page is the main display page for any food item that the user specifically wants to add or edit. If the user reaches this page through the FoodSearch.js page, the page will be properly formatted to show that the user is adding the food. If the user reaches this page through the MealCard.js page, it will be properly formatted to show that the user is editing the food. There is a main title card for the food item, as well as the food items corresponding information. The serving size is a fixed variable set by NutritionIX. The servings input field can be changed by the user and should reflect how many servings of the food item the user had. This servings value will also change how many total calories the user is receiving and will be reflected at the bottom of the page, above the confirmation button. This confirmation button will redirect the user back to the food card’s corresponding MealCard.js page with the food information pushed onto the meal object array stored there.

**Name:** FoodNotes.js

**Description:** This page is to log any meal notes for the user’s meal log for that current day. The majority of the screen is filled with a TextInput which allows the user to input their note. If a note already exists for that day, it will be loaded into the page and the title and confirmation button’s text will change accordingly.

**Name:** MealCard.js

**Description:** The main page for a specific meal is formatted here. This will only be initialized with every new meal from the MealLog.js page. After the initializing props received from FoodActions, all data will be loaded in through “this.props.mealObj” if any data is added to this specific meal. If there is any existing data for the meal when entering this page later on, all data will be loaded in through the “data” variable to be formatted. When the user accesses this page with a different date set, “this.props.viewOnly” will be set to true, forbidding any changes to the page as well as removing buttons. The buttons

**Name:** MealLog.js

**Description:** All meals available for the listed day will be displayed here. There is a TouchableOpacity displayed at the top that initially displays the current day. The user can change the date through this TouchableOpacity and display the meals attached to that day. There is an icon displayed on the top right that redirects the user to a newly initialized MealCard page with no meal data. Any existing Meal Log data will be loaded through the loadData function and displayed through a FlatList of TouchableOpacitys, each tied to a specific Meal Card page with individual data. When pressing any of these TouchableOpacitys, the data tied to each Meal Card is passed through. There is also a TouchableOpacity for MealNotes that displays at the bottom of the page if the user is still on the current day. It will disappear if the user redirects to a Meal Log of a different day.

**Name:** Messenger.js

**Description:** An inbox for the user’s incoming emails from the health coach and study will be displayed here. There is a FlatList of TouchableOpacitys in order of recent emails. Any unread emails should be displayed. It shows the subject and the date of each email in the TouchableOpacity. There is also a TouchableOpacity at the bottom of the page for the user to create a new message. All of these TouchableOpacitys redirect to the Email.js page with different flags set, depending on whether it is an existing email from the inbox or whether the user wants to create a new message.

**Name:** Report.js

**Description:** This is a page used for the user to input their weight for the current day and see a graphical representation of their weights overtime as they are being logged. All weights for the user are requested and loaded through a JSON object. The graph is formed through the Victory-Native library for displaying graphical data. The x-axis uses a scale of time while the y-axis uses the user’s minimum and maximum weights to form a range of values. At the top, there is a view that initially displays the most recent inputted weight and the date that the weight was logged by the user. If the user presses their finger down on the graph at any point, the closest date on the x-axis to the point of the user’s finger is displayed on the view at the top, along with the corresponding weight logged on that date. Below the graph are three buttons that will restrict the range of dates displayed on the graph. “1W” will restrict the dates shown to being the previous 7 days. “1D” will restrict the dates shown to being the previous month. “1M” will restrict the dates shown to being the previous year. Below these group of buttons, there is an input box for the user to input their current weight for the day. If the date hasn’t been added, clicking the confirm button will push the new weight and date to the existing data and initialize a POST call. Setting a new weight for the day will initialize a PUT call.

**Name:** ExerciseSearch.js

**Description:** This is where the user will search the NutritionIX database for an exercise. The SearchBar at the top will let the user input any text to be searched. Every change to the SearchBar text is sent through as a request. Data is retrieved from this request and displayed as a FlatList of TouchableOpacitys below. Each TouchableOpacity redirects to the ExerciseCard.js page and passes the information about the specific exercise chosen.

**Name:** FoodSearch.js

**Description:** This is where the user will search the NutritionIX database for a food item. The SearchBar at the top will let the user input any text to be searched. Every change to the SearchBar text is sent through as a request. Data is retrieved from this request and displayed as a FlatList of TouchableOpacitys below. Each TouchableOpacity redirects to the FoodCard.js page and passes the information about the specific food chosen. There is also two buttons to filter the search results received to display either the common food items only or the branded food items only. This design decision was chosen because of how NutritionIX decided to format their information sent through requests. When either of these buttons are pressed, we also change how the information is being passed through to the FoodCard.js page, because a common food item and a branded food item will have different information and variables to use within their data.

**Extra notes:**

1. We utilized a website called Figma to create mock-ups of the pages. The link below is access to the file we used.

<https://www.figma.com/file/H3TmIrs9ef5E5GRheoTW1FIp/Health-At-Hand?node-id=0%3A1>

1. The email page uses static information to fill the incoming messages box. To create a full-functioning email application, the health-coach view feature will need to be completed.