React Native: « Calory Project »

Documentation

By IVANOFF Juan & MOUSSI Ouahib

I) Objectives of the project:

Here are the objectives of the React Native Project we are developing:

1. Develop a Calories Counter and Meal Planner application using React Native and Expo.

2. Allow users to set their health goals and track their caloric intake accordingly.

3. Implement user input handling to capture personal details and health goals.

4. Integrate with a nutrition API to provide users with food database and nutritional information.

5. Enable users to search for foods and view their calorie content and other nutritional facts.

6. Provide a user-friendly interface for meal planning, allowing users to select and add food items to their meal plan.

7. Calculate the daily caloric intake suggestion based on user's health goals, activity level, and other factors.

8. Display the calculated BMR (Basal Metabolic Rate) and adjusted caloric intake to the user.

9. Create a well-structured and organized codebase, following best practices and naming conventions.

10. Demonstrate a deep understanding of key concepts including state management, user input handling, API integration, and UI styling in React Native.

By achieving these objectives, we aim to deliver a comprehensive and functional Calories Counter and Meal Planner application that helps users track their caloric intake, plan their meals effectively, and work towards their health goals.

II) The application's architecture

The “Calory Project” application follows a modular architecture to scale with our progression and to ensure flexibility. The architecture is based on the React Native framework and utilizes Expo for both development and deployment. Here is an overview of the application's architecture:

Presentation Layer: This layer consists of the UI components responsible for rendering the user interface and interacting with the users. It includes screens, components, and navigation elements implemented using the React Native and Expo libraries. The UI components are designed to provide an intuitive and visually appealing experience to the users.

State Management: The application incorporates a state management approach to handle the application's data and state changes. Redux, a popular state management library, is employed to manage the global state of the application. It enables efficient data flow between components, ensuring consistent and synchronized data updates.

Data Services: The application integrates with external data sources to retrieve food database and nutritional information. This is accomplished through APIs provided by a nutrition service provider, here, we used Adamam for our needs. The retrieved data is processed and stored in the application's local state for efficient access.

Business Logic: The business logic layer contains the core functionality of the application. It encompasses the algorithms and rules responsible for calculating daily caloric intake suggestions, adjusting the intake based on user preferences, and generating meal plans. This layer also handles user input validation and performs necessary calculations to derive health-related metrics like BMR.

Data Persistence: The application employs a local data storage mechanism to store user preferences, meal plans, and other relevant data. According to documentation, we use AsyncStorage, a key-value storage system provided by React Native. We use it for storing and retrieving data persistently on the user's device. This ensures that the data remains accessible even when the application is closed and reopened. As this is used in the very last steps of development, we cannot guarantee it will be implemented.

Testing and Debugging: The application follows a comprehensive testing approach to ensure its stability and reliability. Through Versioning and sectioning through modules and steps within them, we ensure stability in our creation. Additionally, debugging tools and error logging mechanisms are utilized to identify and resolve any issues or bugs encountered during the development process.

By planning this architecture, the “Calory Project” application offers a robust and scalable foundation. It enabled our development, efficient data management, and easy progression, allowing for future enhancements and updates as required.

III) Successful features

Module 1: Setting up the project

- Created a new Expo project.
- Installed eslint and prettier.
- Installed React Navigation.
- Ran the app on your phone using Expo Go.
- Created 3 tabs using a Bottom Tab Navigator: Health goals, Food database, and Meal planning.

Module 2: Health goals

- Created a form for user input, including fields for age, gender, height, weight, activity level, and health goal.
- Handled user input using the useState hook and implemented form validation.
- Calculated BMR (Basal Metabolic Rate) using the Harris-Benedict equation.
- Adjusted BMR based on the user's activity level.
- Adjusted caloric intake based on the user's health goal.
- Displayed the calculated daily caloric intake to the user.

Module 3: Food Database

- Chose a nutrition API and obtained an API key.
- Created a search interface with a search bar using the TextInput component.
- Handled user input using state variables and the useState hook.
- Fetched data from the chosen API based on the user's search query.

- Displayed the search results, including food names, calorie content, and other nutritional facts.
- Allowed the user to select a food item for meal planning, specifying the quantity and meal type.

Module 4: Meal Planning

- Creating the Meal Planner Interface (screen and a view for each day of the week and sections for each meal (Breakfast, Lunch, Dinner, Snack) using the View and Text components.)

- Displaying the Meal Plan (food list to export on a particular day, renders items separated per meal and type)

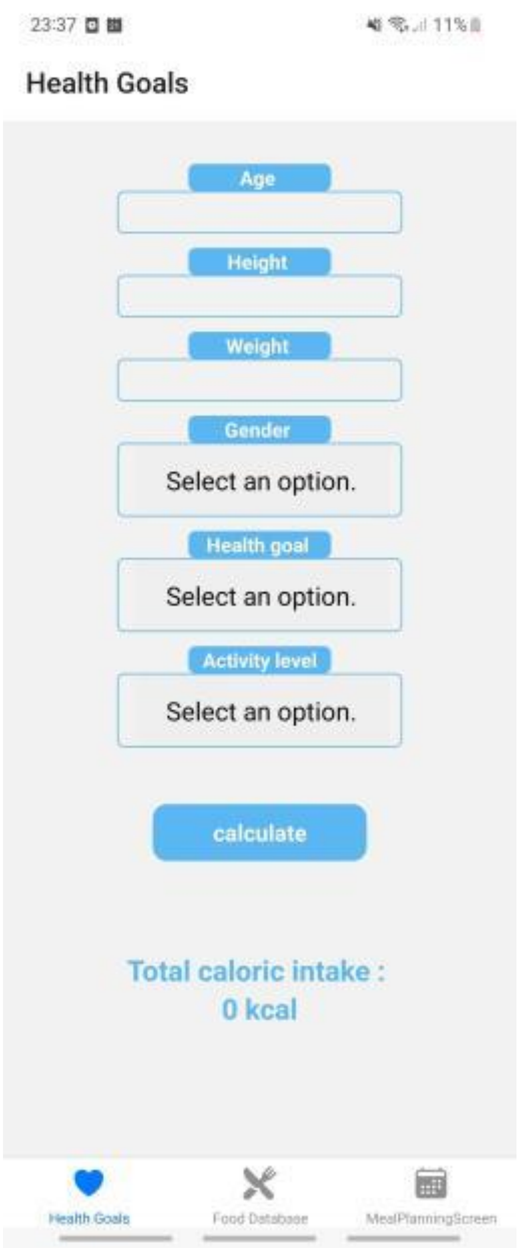
IV) Unsuccessful features

In the Meal Planning Module:

- We can't remove Items from the meal plan (though we can add items)
- The calory Counter per day and meal doesn't work correctly (remains at 0)
- The "export" feature from "Food Database" to "Meal Planning" doesn't work as the sections and calory counters remain blank.
- The Meal Plan doesn't save itself to remain when you reopen the app.

V) Roadmap and screenshots

- June 7: Creation of the Project
- June 12: Module 1
- June 15: Module 2



23:37 11%

Health Goals

Age

Height

Weight

Gender

Select an option.

Health goal

Select an option.

Activity level

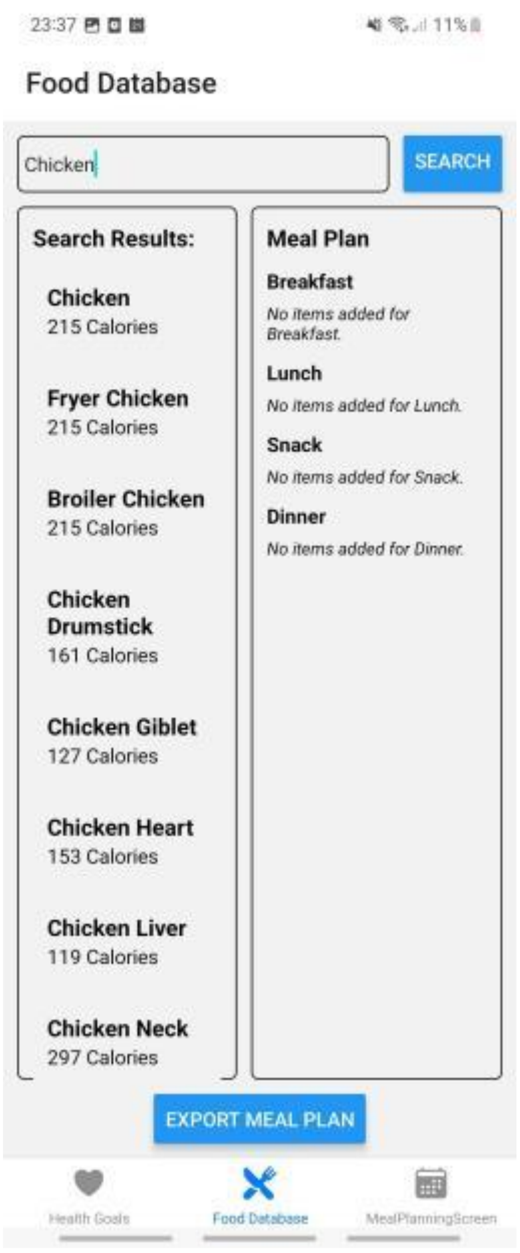
Select an option.

calculate

Total caloric intake :
0 kcal

Health Goals
Food Database
MealPlanningScreen

- June 17: Module 3


23:37

Food Database

Chicken

Search Results:

Chicken
215 Calories

Fryer Chicken
215 Calories

Broiler Chicken
215 Calories

Chicken Drumstick
161 Calories

Chicken Giblet
127 Calories

Chicken Heart
153 Calories

Chicken Liver
119 Calories

Chicken Neck
297 Calories

Meal Plan

Breakfast

No items added for Breakfast.

Lunch

No items added for Lunch.

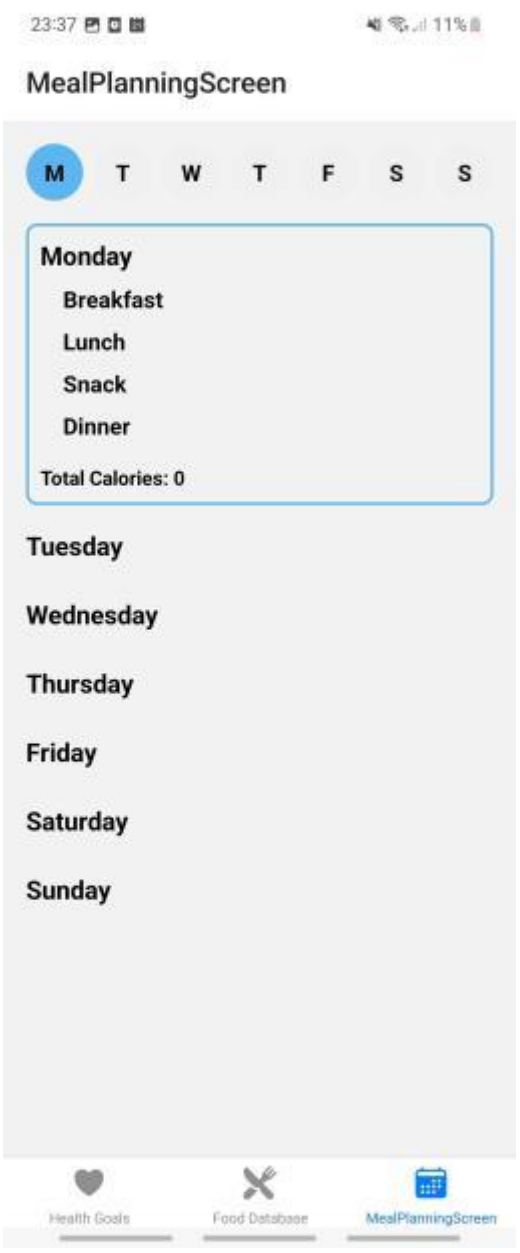
Snack

No items added for Snack.

Dinner

No items added for Dinner.

- June 18: Module 4 but The Export of Food Items doesn’t work



MealPlanningScreen

M T W T F S S

Monday
Breakfast
Lunch
Snack
Dinner
Total Calories: 0

Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

- June 21: “Finition”

VI) Difficulties faced while developing

<u>API Integration:</u> Integrating with a nutrition API was challenging. It required thorough understanding of the API documentation, which I had to reread several times and difficulties in handling API requests, authentication, and parsing the response data. Needless to say, to understand the difference between the API key and the Authentication Code took a while.

Complex State Management: Managing state in a meal planning app with multiple components and screens become complex easily. It lead to challenges in synchronizing and updating state variables effectively, especially when dealing with nested data structures.

UI Design and Styling: Designing an intuitive and visually appealing user interface was time-consuming, especially considering our lack of experience and the clashing of opinions between team members.

Handling Form Validation: Learning and managing to handle errors while and ensuring that user input is validated and is correctly interpreted is quite difficult in hindsight. It lead to delays in feature implementation, which lead to cut features in itself.

Time Constraints and Project Management: Balancing the development of multiple projects while coordinating with a partner, which is impossible to do on git was hard. We had to wait for the signal that the other had finished and pushed his part for the other to take the relay and not lose everything he made