Group 3: Jahlil Owens, Trishelle Leal, and Landon Strappazon

Written By Trishelle Leal

Dr. Ruth Lamprecht

CMSCI 349 A

September 20, 2024

Milestone 1b

**Data engineer**

When designing the Nutrition Tracking App, it is essential to consider the functionalities that we will offer within the app, and how these functionalities breakdown onto individual components. The main features the app offers are daily food intake tracking, nutrition goal managing, physical activity monitoring and health metric logging. These are all reflected within the data architecture in the following UML Class Diagram. The following is an explanation of each component and how they interact with one another.

**User**

The user class stores the details of the individual using the app. It includes personal information of the user, such as userID, name, email, password, age, gender, height, and weight. This information is used to offer personalized recommendations for the user.

**Relationships:**

* Each user will have multiple DailyLog entries, representing the user’s daily activity and food consumption records.
* The User also has a relationship with HealthMetrics, tracking the changes within weight and body fat over time.
* The User can define multiple goals, to define an objective in their health and fitness journey.

**GoalMetrics:**

This class consists of the user’s desired metrics, stored to motivate the user by offering personalized notifications in celebration of hitting daily or set goals.

**Relationships:**

* This class works hand in hand with the HealthMetrics class, to offer a more personalized app for the user. It also tracks whenever one of the HealthMetrics attributes has been set as lower than the goal set for the given metric, meaning that the goal has been achieved.
* This class is used to compare the daily intake with the intake the user desires to have, this helping them become motivated through graphs and notifications.
* The User can have multiple entries, pertaining to when the user decides to change their goals.

**HealthMetrics**

This class stores the user’s health and fitness metrics, with the purpose of identifying trends and facilitating the tracking of the journey. The user can have multiple entries; to represent the changes within the body across the time the app has been in use.

**DailyLog**

The DailyLog class represents a daily record of the user’s food intake and physical activity. This class holds the following attributes.

* logID: log identification.
* Date: log date.
* meals[]: each meal the user logged within the day and their individual information.
* activities[]: stores the user’s exercising activity logs.
* totalCalories\_consumed: total amount of the calories consumed across all the meals logged.
* totalCalories\_burned: total amount of the calories burned through activities and the steps taken within the day.
* Macros[]: stores the macro intake for the day.

**Relationships:**

* The DailyLog class can have multiple meal and activity entries, depending on the consumption and exercising done for the day.
* The DailyLog class will be often used to create trends and compare the user’s consumption for the day against other DailyLogs.
* The DailyLog objects will be displayed against the goals, helping the user visualize what they need for the logs.

**Meal**

The Meal class is a subclass of the DailyLog class. It stores a collection of FoodEntry items consumed within a singular meal and the specific values pertaining to the totality of the meal. It contains:

* mealID: meal identification.
* mealType: identifies the type of meal logged (Breakfast, lunch, dinner, snacks).
* Date: date of the meal.
* totalCalories: calories for the totality of the meal.
* totalNutrients[]: total nutrient intake for the meal.
* foodEntries[]: stores every food input for the meal.
* totalMacros[]: stores the sum of every macro.

**Relationships:**

* Each meal includes one or more FoodEntry objects, each representing an individual food item.

**FoodEntry**

The FoodEntry class is a subclass of the meal class, and tracks the food items the user consumes at a time. It includes:

* EntryID: food entry identification.
* name: name of the food item.
* servingSize: amount of food intaken (used to calculate the item’s calories).
* calories: calories intaken in the food item.
* Macros[]: Macros registered in the item.
* Nutrients[]: Nutrients in food item.

**Relationship:**

* A singular FoodEntry object contains one or more Nutrient entries, this provides detailed information about the nutrients within the item.
* A singular FoodEntry will contain 3 Macro object, each giving the quantities per macro for the food item.

**Nutrient and Macros**

The Nutrient class and Macro are both a subclass of the FoodEntry class, providing the nutrient and macros value for a singular food item. Both include the following attributes:

* nutrientID / MacroID: nutrient identification / Macro identification.
* name: names the nutrient.
* amount: quantity of the nutrient in each food item.
* unit: unit of measurement (mg, g, IU).

**Activity**

The activity class functions as a subclass of the DailyLog class, and stores the exercise the user has done in a given day. It includes the following attributes.

* activityID: activity identification.
* activityType: identifies the kind of exercise done.
* duration: minutes the activity lasted.
* caloriesBurned: number of calories burned within the activity.
* date: date the activity was done in.
* steps: steps taken within the day.

A screenshot of a computer

Description automatically generated

**Deployment Engineer**

As a deployment engineer, I have set up an initial repository on Git to store all group documentation and source code. Folder and branch structure has been initially set, but as the project grows, it will change or add branches or folders. To be able to have contributions from different members of the group, we have opted to use GitHub. The repository is private, to ensure that nobody but the group members and the professor have access to said repository.

Merging code can be difficult, so we have opted to create a process to be able to ensure the code that is committed is correct and to facilitate the identification of possible issues within the project. We have decided that our branching strategy will work using the following methods:

* The main branch will only be used for stable, production-ready code.
* Those working on an individual feature will create separate branches for their work. This is to recognize files per feature and the features that are still not ready to be deployed.
* The develop branch will function as a pre-stage for the main branch, where integrating features can be tested prior to merging.
* The developer will submit a pull request to be able to merge their code into the develop branch. Each pull request will require a code review and approval.

As the person in charge of deployment, I will leverage the merging between branches, making sure that the group members’ code is correct and able to work with what is already integrated into the branches. On another note, I will manage the Integration environment. Once code is merged into the develop branch, it will be deployed to the integration environment for testing. All team members will be set as responsible for ensuring that their code works within this environment before requesting a merge onto the main branch.

**Git capabilities**

Capabilities are essential for a collaborative environment, and to ensure code integrity, avoid conflict and promote the creation of good, efficient code.

* **Branching:**  The group members will work within isolated branches to avoid interfering with each other’s work. This allows for parallel development and editing without creating any conflicts.
* **Push:** Each will push their code to their feature branches and open pull requests when needed. This ensures that only reviewed and approved code is integrated.
* **Merge:** This is a capability only I will have. After code reviews, I will merge into develop and later onto main.

PyCharm’s built-in Git functionality will be integrated with GitHub for easy-version control. Using this IDE, the group members can easily use Git capabilities within the interface. Additionally, PyCharm’s diff and merge tools can be used when resolving conflicts or reviewing changes in the code. This will ensure that the project that is delivered towards the end of the semester is of great quality and that the group members can create it seamlessly, without conflict.