EatThis: Your Smart Nutrition Tracker

Software Design Document

Document Change Control

Version	Date	Authors	Summary of Changes
0.1.0	23/06/2025		Deliverable 1
		Philips Rhoguns	

Contents

1. Introduction

1.1. Purpose

This application is designed to allow the user to track their daily nutrient intake with detailed breakdowns that help the user understand their eating habits. The food swap feature is the core functionality that uses this data and allows the user to achieve their personal dietary goals or to improve their nutrient intake profile per Canada Food Guide recommendations.

Software Requirements Specification

1. Profile creation
    a. Basic user information
        i. Name
        ii. Sex
        iii. Date of birth
        iv. Height
        v. Weight
    b. System preferences
        i. Metric vs Imperial
    c. Profile selection in the splash screen
    d. Main profile UI has options to edit profile or settings
    e. Store profile data in the database instead of files
    f. Profile or settings changes should be immediately reflected in the UI
2. Diet data logging
    a. Date of meal, type (breakfast, lunch, dinner, snack)
        i. Snacks have no logging limit, one per day for other types
    b. Ability to log basic ingredients and their quantities in cooked amounts
    c. Application should extract information from the meal and break down the micro- and macronutrient profile
    d. Journal view only displays calories, but when a meal is selecte,d the full breakdown is displayed
3. Smart food swaps
    a. Options with diet goals, including intensity of change
    b. Swaps should allow for 1-2 goals to be achieved together
    c. Application queries the database for reasonable swaps (other nutrients should remain within 5-10% of the original value)
    d. Swaps should come from the same food group whenever possible
    e. Swaps should replace 1-2 food items at most from one meal
4. Compare nutrient intake pre- and post-swaps
    a. Side-by-side ingredients list of both meals with highlighted swaps
    b. Tooltip on highlighted ingredients with nutrient changes
    c. Separate view with nutrient breakdown for both meals, with visual indicators for changes
5. Save swaps to apply to past meals and record cumulative changes
6. Provide visual daily nutrient intake information in percentages over a selected time period, and notification or visualisation for how close to recommended daily intakes
7. Provide user diet comparison to Canada Food Guide recommendations
    a. Create visualisation of the user's average meal in percentage of represented food groups and compare to the recommended amounts given by CFG
        i. Food group percentages determined by cooked ingredient weights
    b. Approximate CFG recommendations for a healthy plate (meal)
        i. 50% vegetables and fruits
        ii. 25% whole grain foods
        iii. 25% proteins (preferably plant proteins)
8. Provide visualisation of the effect of food swaps
    a. Multiple options for visualisations for:
        i. Change in specific nutrient intake over specific time periods
        ii. Comparison of nutrient intake before and after swaps
        iii. Comparison of nutrient intake to CFG guidelines before and after swaps

1.2. Overview

This application provides users with a simple and interactive tool that removes the tedium of tracking and analysing your diet to make healthy changes. EatThis draws nutritional information from the Canada Nutrient Profile to create food swaps that facilitate these changes. The user is presented with different visualisation options to view the effects of these changes on their nutrient profile in different contexts, as well as their adherence to recommendations given by the Canada Food Guide.

1.3. Resources

[1] Canada Nutrient File 2015. Retrieved from: https://www.canada.ca/en/health-canada/services/food-nutrition/healthy-eating/nutrient-data/canadian-nutrient-file-2015-download-files.html

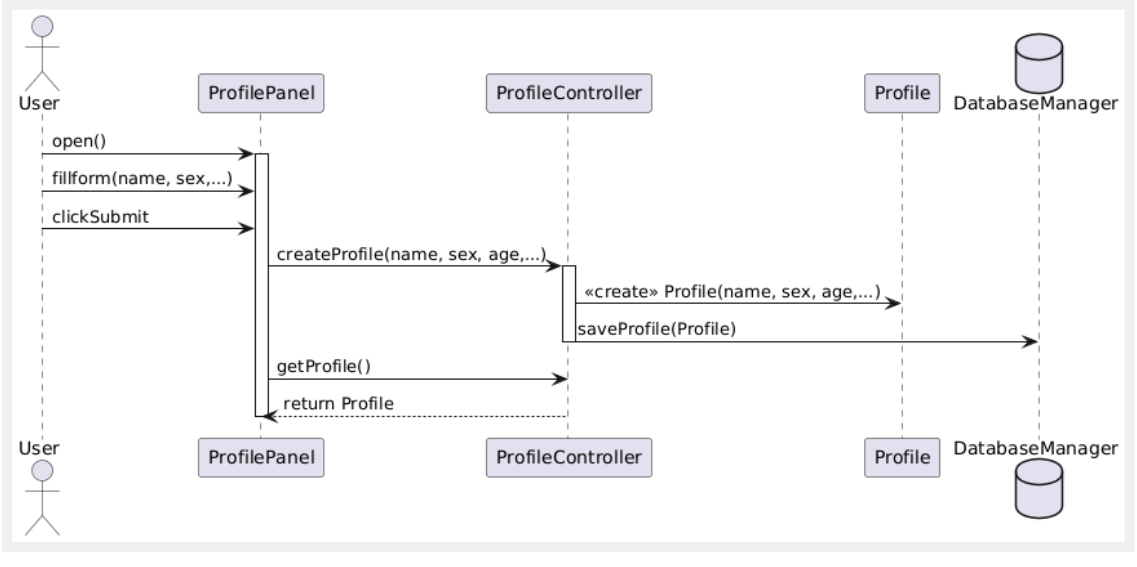
[2] Canada's Food Guide 2007. Retrieved from: https://publications.gc.ca/collections/Collection/H164-38-1-2007E.pdf

[3] Canada's Food Guide 2019. Accessed via: https://food-guide.canada.ca/en/

2. Sequence Diagram

2.1. UC1: Creating a user profile

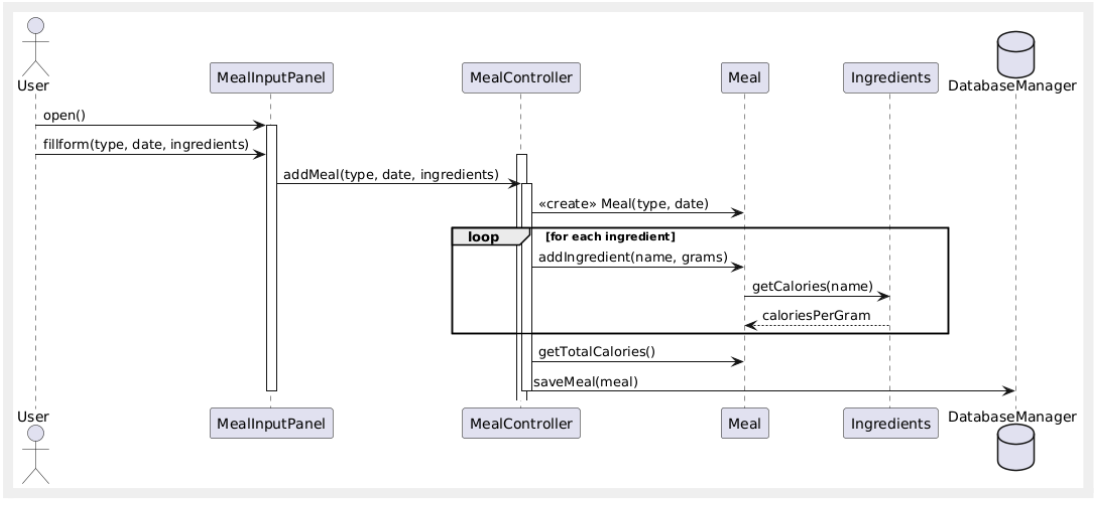
Use case summary: This use case allows a user to create and save their profile by entering personal data through the GUI. The profile is created and stored in the system.



Sequence Diagram explanation: This diagram illustrates how a user creates a profile through the ProfilePanel. The panel collects the input and forwards it to the ProfileController, which creates the Profile object and saves it using the DatabaseManager.

2.2. UC2: Logging a meal

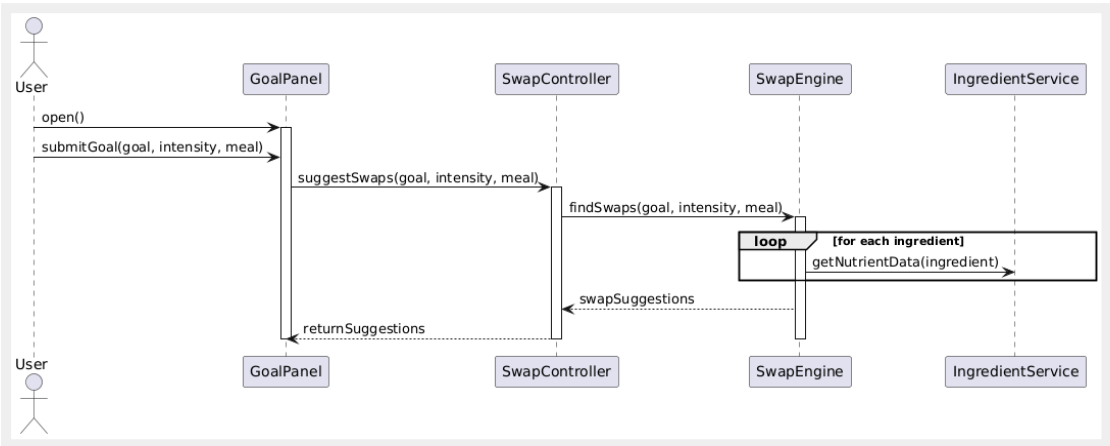
Use case summary: A user can record their meal in this use case. The total nutritional values are then computed, shown, and stored by the system.



Sequence Diagram explanation: The meal logging procedure is described in this sequence diagram. In the MealInputPanel, the user enters a meal and its components. The MealController generates a Meal object, uses Ingredients to determine how many calories it contains, and then uses DatabaseManager to store it.

UC3: Requesting food swaps to meet nutritional goals

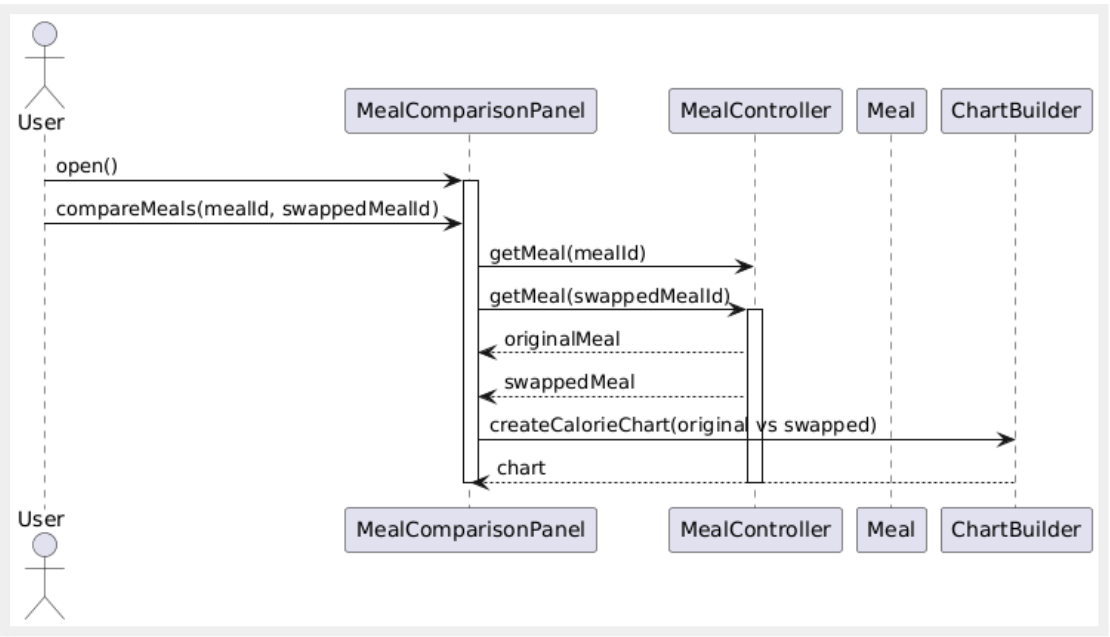
Use case summary: In this use case, the user can choose a goal (like increasing fibre) and the system will recommend foods that will help them reach that goal.



Sequence Diagram explanation: The way the application compares an original meal with a swapped meal is shown in this sequence diagram. The MealComparisonPanel uses ChartBuilder to create a visual comparison of nutritional differences after requesting both meal objects from the MealController.

2.3. UC4: Comparing swapped meals

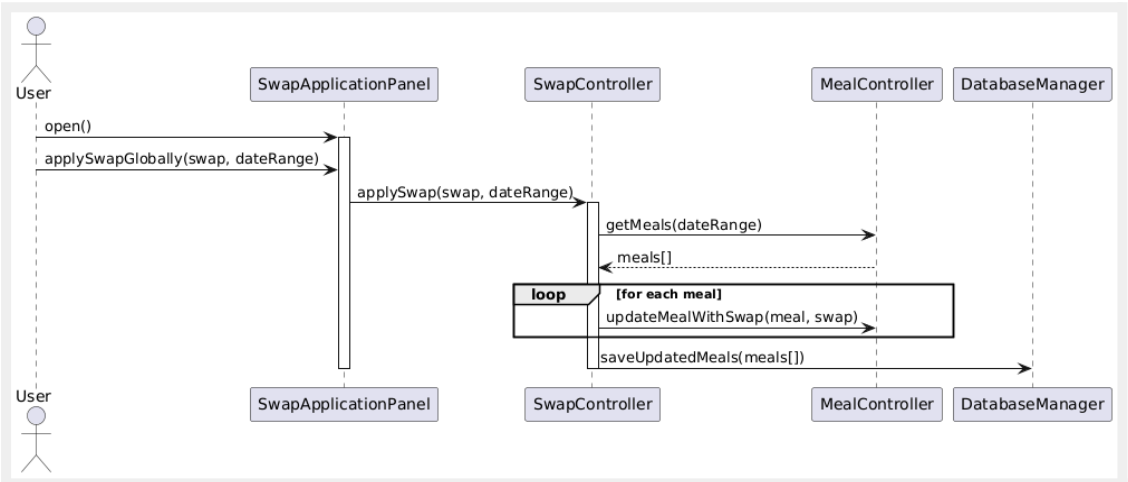
Use case summary: This use case highlights nutrient changes and swapped items by comparing an original meal and a suggested substitute meal side by side.



Sequence Diagram explanation: The way the application compares an original meal with a swapped meal is shown in this sequence diagram. The MealComparisonPanel uses ChartBuilder to create a visual comparison of nutritional differences after requesting both meal objects from the MealController.

2.4. UC5: Applying substitution over time

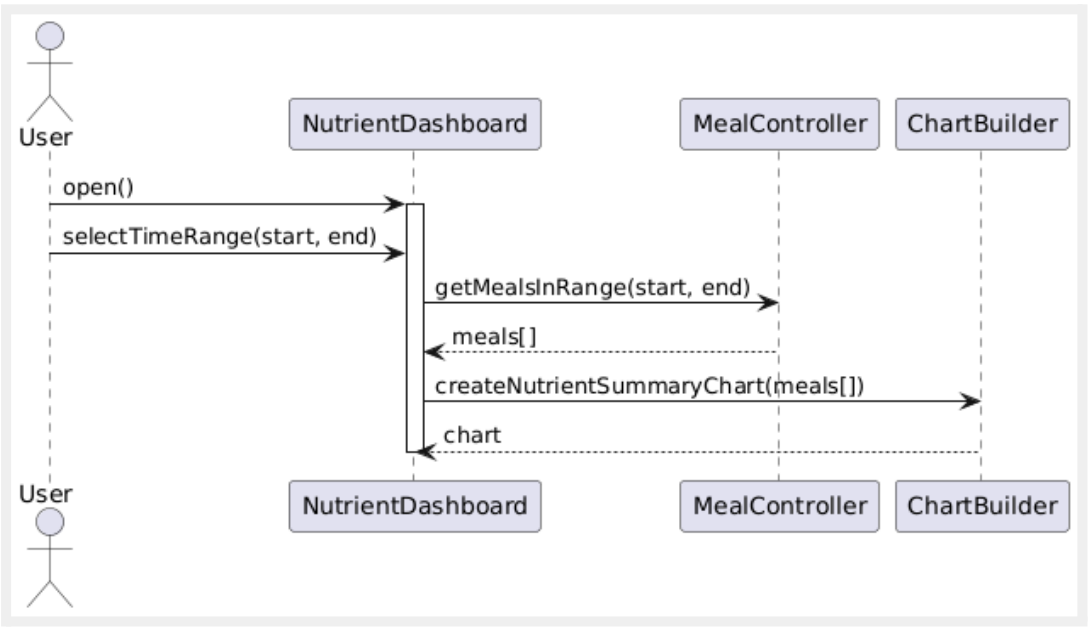
Use case summary: This use case enables a user to observe the long-term effects of applying a recognised food swap across meals within a chosen time frame on nutrient intake.



Sequence Diagram explanation: This sequence diagram illustrates how the user applies a chosen food substitution to several meals over a predetermined amount of time. The impacted meals are updated by the system, and the updated versions are stored in the database.

2.5. UC6: Visualising daily nutrient intake

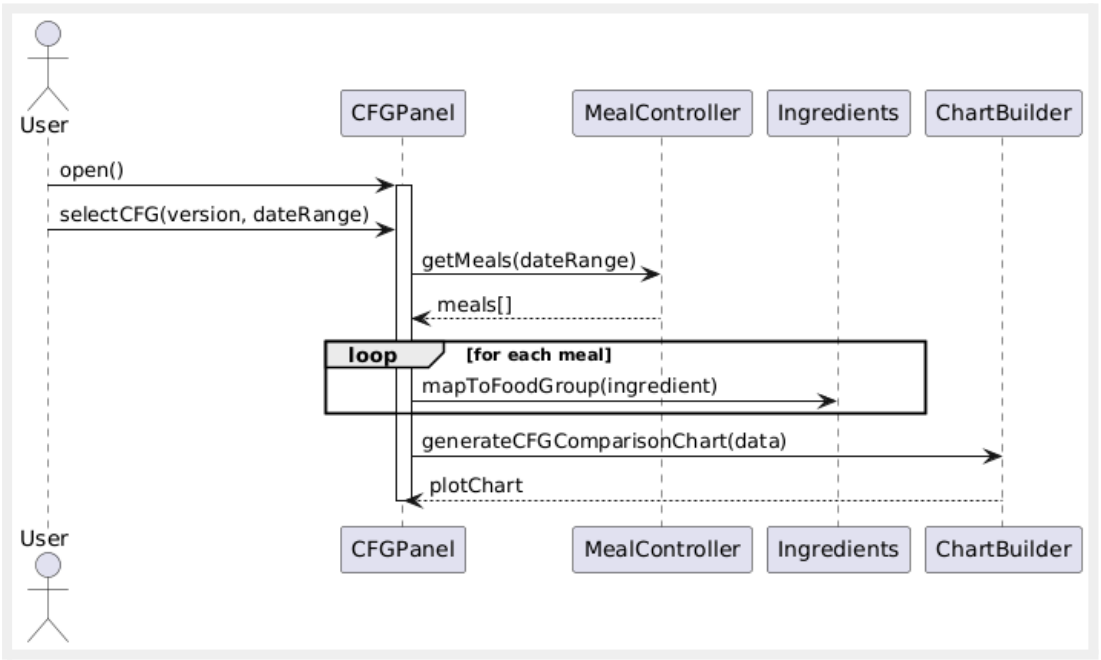
Use case summary: This use case highlights the most important nutrients and compares them to suggested values, displaying the average daily intake of nutrients over a selected time period.



Sequence Diagram explanation: This sequence diagram illustrates how a user can choose a time frame and view a chart that shows the average daily nutrient intake. A nutrient summary chart is produced by the system after compiling the pertinent meals.

2.6. UC7: Checking alignment with CFG

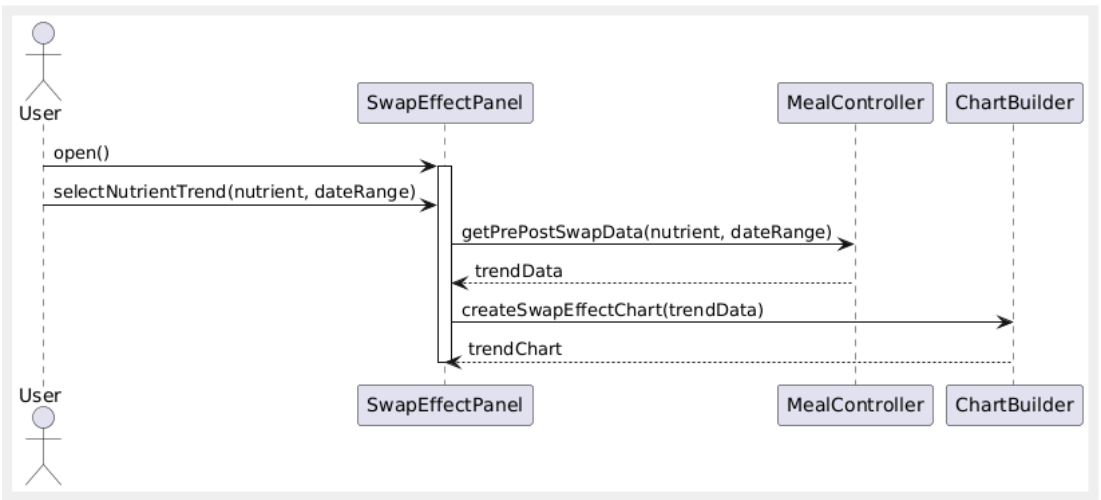
Use case summary: The user can evaluate how closely their diet complies with the Canada Food Guide's recommendations using this use case.



Sequence Diagram explanation: This diagram illustrates how the user verifies that their diet complies with the Canada Food Guide. The system visually compares dietary intake to CFG guidelines by mapping it to food groups.

2.7. UC8: Visualising the effects of food swaps

Use case summary: This use case illustrates how food substitutions affect nutrient trends over time and compliance with the Canada Food Guide.



Sequence Diagram explanation: This sequence diagram illustrates how a user perceives the long-term impacts of food substitutions. The system creates a comparative chart for visualisation by retrieving nutrient trends prior to and following swaps.

3. Major Design Decisions

Our priority with the design is to create an implementation that is simple to use and work with, from the perspective of users and developers. Since the project is limited in functionality to its given use cases, we want to focus our attention on the aspects we have the most control over. This means a focus on the user experience—we want to accurately assess what factors go into a satisfactory experience, which should then be fine-tuned with feedback from real-world evaluations from test users once a prototype is ready. For us, the design implication is a user-focused implementation of the UI (and any viewer-related parts of the application).

As for the implementation of the core functionalities outlined in the use cases, we want to create a system that follows some of the best practices outlined in the major frameworks of design principles. Generally speaking, we want the code to be clear in its purpose in any given section. This means creating the necessary modules that create a clear separation of responsibilities and a clear hierarchy. Our preliminary modules are as follows:

- ProfileModule: manages user profile creation and retrieval
- MealModule: handles meal logging and retrieval
- SwapEngine: suggests and applies food swaps
- Analysis Module: analyses nutrient data and Canada Food Guide alignment
- DatabaseManager: stores and retrieves all persistent data
- ChartBuilder: builds charts for nutrient breakdown and comparison

We believe this is a streamlined decomposition of the system from an initial assessment, but we are certainly ready to remove, add, or split modules as necessary throughout the development process. Per the goals of achieving high cohesion and low coupling, modules were chosen so that they operate in the domain of a clearly defined single responsibility, while also appropriately broad enough in scope to house all required functionalities. The initial decomposition should also allow for any necessary abstraction and encapsulation as we develop the system.

Many of these design decisions are reflected in the design pattern choices and their justification, which are discussed in greater detail later in this document.

4. Architecture

This section explains how the "EatThis: Your Smart Nutrition Tracker" system was first broken down into interconnected modules, as well as the features, interfaces, and functions of each module. The design places a high priority on user and developer ease of use, emphasising a hierarchical structure and a clear division of duties.

4.1. System Decomposition and Modules

In this section, we will go over how the project is organised. The goal is to make sure each part is easy to understand and develop. The entire system is divided into smaller sections so that every section has its own job and duty. This subsequently makes it easier to maintain and add new features as it grows.

- **User Interface Module (UI):** The user interface (UI) module is in charge of gathering user input and presenting it to the user.
    - Functionality: Displays meal logging forms, user profiles, food swap options, charts, and comments regarding compliance with the Canada Food Guide.
    - Some Operations:
        - displayProfile(profileData): To display user profile details.
        - showMealLoggingForm(): Form to log meals.
        - displaySwapSuggestions(suggestions): Prints a list of suggested food swaps.
        - renderChart(chartData): To visualise data in the form of charts.

- **ProfileModule:** Manages the development, archiving, and retrieval of user profiles, which contain dietary preferences and objectives in addition to personal data.

    - Functionality: Manages secure storage, updates profile information, retrieves existing profiles, and creates new user profiles.
    - Exposed Interfaces: IProfileManager
    - Some Operations:
        - createProfile(userData): creates a new user profile after filling in the form
        - getProfile(userId): Fetches the user profile by unique ID.
        - updateProfile(userId, newProfileData): To alter the existing profile.
        - deleteProfile(userId): To remove a profile from the system

- **Meal Module:** Handles every aspect of meal logging, including processing food, receiving meal input, and communicating with other modules to store information and compute nutrients.

    - Functionality: Manages meal data, logs new meals, retrieves previous meals, and starts calculating the nutrients in logged meals.
    - Exposed Interfaces: IMealLogger
    - Operations:
        - logMeal(mealEntryData): Logs a meal entry with details such as name and quantity.
        - getMealsByDate(date): Fetches all meals logged on a specific date.
        - getMealById(mealId): Fetches a meal entry by unique ID.
        - updateMeal(mealId, updatedMealEntryData): Modifies an existing meal entry.

- **SwapEngine:** Responsible for developing and carrying out suggestions for food substitutions based on dietary goals specified by the user. It interacts with the AnalysisModule to understand nutrient profiles and the DatabaseManager for food data.

    - Functionality: Suggests options for selected food items, compares original and swapped meals.
    - Exposed Interfaces: ISwapSuggester, IMealComparer
    - Operations:
        - suggestSwaps(originalMeal, nutritionalGoal): Generates a list of options for the selected food item and goal
        - CompareMeals (meal1, meal2): Generates a detailed comparison between the selected food and its swapped food item.
        - applySwap(originalMeal, swapSuggestion): If swap is applied, creates a new meal object.

- **AnalysisModule:** Evaluates how well a diet fits the Canada Food Guide (CFG) by analysing meal nutritional data. It calculates the intake of calories, nutrients, and food groups.

    - Functionality: Evaluates diet in relation to CFG recommendations, calculates complete nutritional data for meals, and aggregates nutrient intake over time.
    - Exposed Interfaces: INutrientAnalyzer, ICFGAnalyzer
    - Operations:
        - analyzeNutrients(foodItems): Calculates total cals, macro and micro nutrients.
        - getDailyNutrientSummary(dateRange): Provides an average daily nutrient intake for a specific period of time.
        - assessCFGAlignment(mealData): Determines if the user's diet aligns with CFG recommendations.

- **DatabaseManager:** Used for storing and retrieving data.

    - Functionality: Provides access to the Canadian Nutrient File (CNF) data and keeps track of user profiles, meal logs, and food swap histories.
    - Exposed Interfaces: IDataStore
    - Operations:
        - saveProfile(profile): Stores the user profile.
        - loadProfile(userId): Fetches a user profile.
        - saveMeal(meal): Saves logged meals.
        - loadMeals(query): Fetches meal entry information based on selected criteria (Date, type, etc).
        - getFoodNutrientData(foodName): Fetches detailed nutrient information for a specific food item from the CNF.

- **ChartBuilder:** Uses libraries like JFreeChart to visualise stored, retrieved data.

    - Functionality: Creates different types of charts based on saved nutrient information by the user.
    - Exposed Interfaces: IChartGenerator
    - Operations:
        - buildNutrientBreakdownChart(nutrientData): Generates a chart to visualise the user's nutritional data.
        - buildComparisonChart(dataBefore, dataAfter): Generates a chart to visualise two different sets of nutrition data.
        - buildCFGAlignmentChart(alignmentData): Generates a chart of nutritional data stored by CFG.

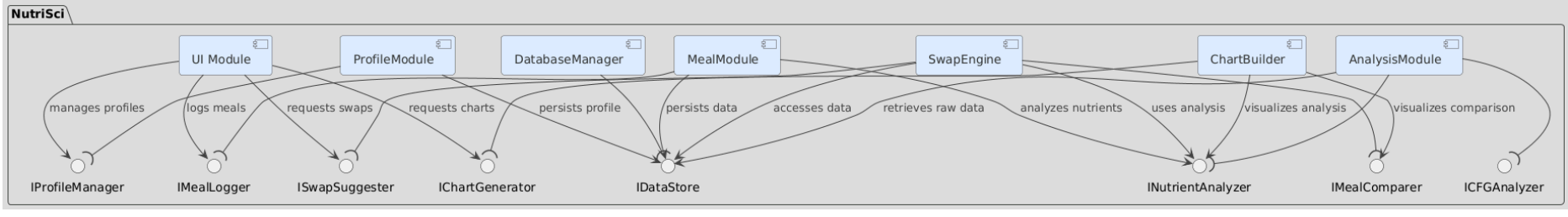
4.2. Module Interaction

Together, the modules enable the system to meet the following use cases:

- To display data and send user commands, the UI Module—the primary user interface—interacts with the ChartBuilder, SwapEngine, ProfileModule, and MealModule.
- The AnalysisModule calculates nutrients for logged meals on behalf of the MealModule, while the DatabaseManager stores and retrieves meal data.
- The SwapEngine compares meals, queries the AnalysisModule to get nutritional data for food items, and interacts with the DatabaseManager to retrieve food data for swap recommendations.
- The raw nutrient information for foods—specifically, the CNF data—is obtained from the DatabaseManager by the AnalysisModule.
- The ChartBuilder generates visualisations using processed data from the AnalysisModule (daily nutrient summaries, for example) and potentially the SwapEngine (for meal comparisons).
- The ProfileModule, MealModule, AnalysisModule, and SwapEngine all receive and store data from the DatabaseManager, which serves as the persistence layer.

4.3. Architectural Diagram

Below is a conceptual representation of the system's architecture, illustrating the major modules and their primary interaction:

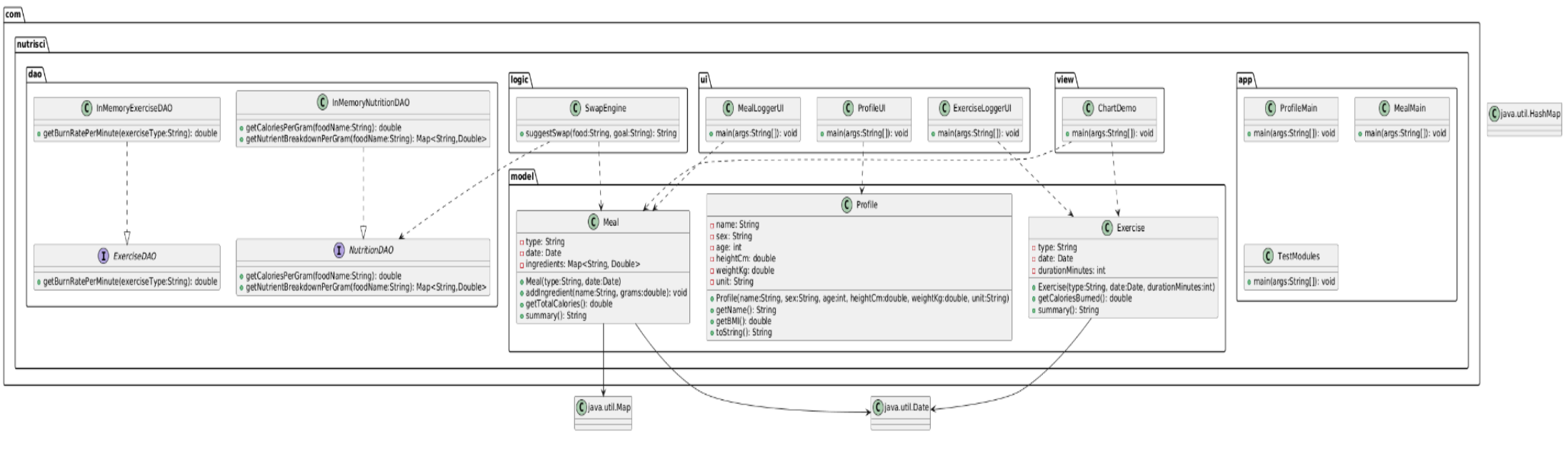


The overall control or data dependency flow is depicted by the diagram's arrows. For example, other modules receive requests from the UI Module and may return data to the UI for display. The DatabaseManager provides the storage and retrieval of data for almost all other functional modules. The AnalysisModule's core data processing features (nutrient analysis, CFG alignment) are used by other modules like MealModule and SwapEngine.ChartBuilder uses processed data from the AnalysisModule and SwapEngine to create visual outputs that are displayed by the UI Module. The clear separation of responsibilities in this architecture makes it easier to develop, test, and maintain the system.

5. Detailed Class Diagram

5.1. UML Class Diagrams

Class diagram here



Below are detailed class diagrams for each class that was created or modified as part of the extension.

Class Name	Attribute/Method Signature	Description
Profile	– `name: String`	The user’s name.
	– `sex: String`	Sex/gender of the user.
	– `age: int`	Age in years.
	– `heightCm: double`	Height in centimeters.
	– `weightKg: double`	Weight in kilograms.
	– `unit: String`	Units of measurement ("metric" or "imperial").
	+ `Profile(name:String, sex:String, age:int, heightCm:double, weightKg:double, unit:String)`	Constructor that initializes all profile fields.
	+ `getName(): String`	Returns the profile’s `name`.
	+ `getBMI(): double`	Calculates BMI = weightKg / (heightCm/100)².
	+ `toString(): String`	Returns a formatted string summarizing all fields including BMI.
Meal	– `type: String`	Meal type ("Breakfast", "Lunch", "Dinner", or "Snack").
	– `date: Date`	Date/time when the meal was logged.
	– `ingredients: Map<String, Double>`	Maps ingredient names → quantity in grams.
	+ `Meal(type:String, date:Date)`	Constructor: sets meal type and date.
	+ `addIngredient(name:String, grams:double): void`	Adds or updates an ingredient name with its quantity (in grams).
	+ `getTotalCalories(): double`	Sums calories by looking up each ingredient’s calories-per-gram.

Class Name	Attribute/Method Signature	Description
	+ `summary(): String`	Returns a multiline summary (type, date, each ingredient with quantity, and total calories).
Exercise	– `type: String`	Exercise type (e.g., "Running", "Cycling", "Yoga").
	– `date: Date`	Date/time when the exercise was performed.
	– `durationMinutes: int`	Duration in whole minutes.
	+ `Exercise(type:String, date:Date, durationMinutes:int)`	Constructor: sets exercise type, date, and duration.
	+ `getCaloriesBurned(): double`	Calculates calories burned based on type and duration (simple stub/hardcoded rates for now).
	+ `summary(): String`	Returns a multiline summary (type, date, duration, calories burned).
SwapEngine	+ `suggestSwap(food:String, goal:String): String`	Given a current food name and a goal (e.g., "reduce calories", "increase fiber"), returns a single best-swap suggestion as a String.
ChartDemo	+ `main(args:String[]): void`	Launches a `JFreeChart` window displaying a static pie chart of nutrients).
ProfileMain	+ `main(args:String[]): void`	Instantiates a sample `Profile` and prints `toString()` to the console.
MealMain	+ `main(args:String[]): void`	Creates a sample `Meal`, adds a few ingredients, then prints `summary()` to the console.
TestModules	+ `main(args:String[]): void`	Runs basic test routines.
NutritionDAO	`+ getCaloriesPerGram(foodName:String): double`` + getNutrientBreakdownPerGram(foodName:String): Map<String,Double>`	Interface defining methods to fetch calorie-per-gram and full nutrient breakdown for a food item.
InMemoryNutritionDAO	`+ getCaloriesPerGram(foodName:String): double`` + getNutrientBreakdownPerGram(foodName:String): Map<String,Double>`	In-memory stub implementation of `NutritionDAO`, using a hardcoded lookup map.
ExerciseDAO	`+ getBurnRatePerMinute(exerciseType:String): double`	Interface defining method to fetch calorie burn rate (kcal/minute) for a given exercise type.
InMemoryExerciseDAO	`+ getBurnRatePerMinute(exerciseType:String): double`	In-memory stub implementation of `ExerciseDAO`, using a hardcoded lookup map.
ProfileUI	`+ main(args:String[]): void`	Swing GUI entry point for creating/editing a profile; collects user inputs and displays the saved profile.
MealLoggerUI	`+ main(args:String[]): void`	Swing GUI for logging meals; lets the user add ingredients and displays a running list and total calories.
ExerciseLoggerUI	`+ main(args:String[]): void`	Swing GUI for logging exercises; collects type and duration, then displays calories burned.

Link to implementation: https://github.com/Rhoguns/3311-nutrisci

6. Use of Design Patterns

Creational: Builder

There are several functions of the app that might benefit from a builder design pattern. The three that initially come to mind are object creation for profiles, meals, and charts. This allows more flexible object creation, including ease of handling optional parameters, while avoiding the confusion of constructor parameters which may become bloated and hard to track. This also makes it easier to futureproof additional parameters.

In the case of a potential interface ChartBuilder, it encapsulates all the logic behind different configurations and visualizations that are available to the user. Data representation is a large part of the functionality of this app, so there will likely be multiple functionalities that can apply this design pattern.

This builder design also matches the UI flow well in the case where users “build” an object by inputting values one by one, reflecting the builder structure. For example, when logging a meal, users might input values for date, ingredients, amounts, etc. which get built into a meal object.

Structural: Decorator

The decorator pattern could be useful in creating a wrapper for relevant objects that require additional functionalities instead of bloating the meal interface with all possible actions. One application of this design could be to encapsulate the additional properties of a meal with swaps applied, since we might not apply swaps to all meals.

Structural: Façade

Using the façade design pattern, we can create a centralized access point that cleanly separates the UI from the modules. The UI layer now only communicates with the façade and is no longer involved with the lower-level logic of the system, which achieves reduced coupling.

Behavioral: Strategy

Given the multipurpose use of the swap function in our system, it is likely best practice to implement some level of abstraction for the logic behind these swaps. This encapsulates the swap logic behind different goals given by the user, which can be called through a common swap interface.

Behavioral: Observer

The observer design pattern addresses the need for the system to update data in the current UI upon profile or settings changes. In this system, our subject would be the active profile, and any related data or preferences would be its observers, representing a one-to-many dependency where a profile state change notifies all its dependents to update the UI.

Behavioral: Singleton

The above responsibility could be handled by some class ProfileManager that controls the active profile and updates the UI appropriately. This class itself could possibly employ the Singleton design to ensure only one profile is active and to simplify calls to the active profile.

7. Activity Plan

7.1. Project Backlog and Sprint Backlog

Deliverable-Specific Breakdown

PB = Product Backlog

Priority Items

- PB-004: User Profile Creation
- PB-007: Meal Logging Interface
- PB-010: Exercise Logging Interface
- PB-012: Basic JFreeChart Visualization
- PB-011: BMR Calculation Implementation
- PB-001: Basic Database Setup
- PB-008: Basic Nutritional Analysis

Expected Deliverables

- Working profile creation module
- Functional meal logging with basic nutrient lookup
- Exercise logging with calorie burn calculation
- At least one chart visualization
- Initial test cases (10-12 tests)

Case 1: System Infrastructure

- **PB-001**: Database Setup and Configuration
    - Set up MySQL database
    - Create database schema for CNF data
    - Implement database connection management
- **PB-002**: CNF Data Loading System
    - Parse CSV files from Canadian Nutrient File
    - Load data into database tables
    - Create data validation mechanisms
- **PB-003**: Application Architecture Setup
    - Implement modular system architecture
    - Set up Java Swing GUI framework
    - Integrate JFreeChart library

Case 2: User Management

- **PB-004**: User Profile Creation
    - Create user registration system
    - Store basic information (sex, DOB, height, weight)
    - Implement profile settings (metric/imperial units)
- **PB-005**: Profile Management
    - Edit profile functionality
    - Profile selection from splash screen
    - Settings update with real-time data refresh
- **PB-006**: Profile Data Persistence
    - Store profile data in database
    - Implement profile loading/saving mechanisms

Case 3: Diet Tracking System

- **PB-007**: Meal Logging Interface
    - Create structured UI for meal entry
    - Date and meal type selection (breakfast, lunch, dinner, snacks)
    - Ingredient and quantity input system
- **PB-008**: Nutritional Analysis Engine
    - Extract nutrient information from CNF database

    - Calculate meal nutritional values (calories, proteins, carbs, vitamins)
    - Generate meal nutritional breakdown
- **PB-009**: Diet Journal System
    - Display daily calorie summary
    - Detailed nutrient breakdown view
    - Meal history management

Case 4: Exercise Tracking System

- **PB-010**: Exercise Logging Interface
    - Date and time input for exercises
    - Exercise type selection (walking, running, biking, swimming, etc.)
    - Duration and intensity input (low, medium, high, very high)
- **PB-011**: Calorie Burn Calculation
    - Implement BMR calculation algorithms
    - Calculate calories burned based on exercise data
    - Total Daily Energy Expenditure (TDEE) calculation

Epic 5: Data Visualization and Analytics

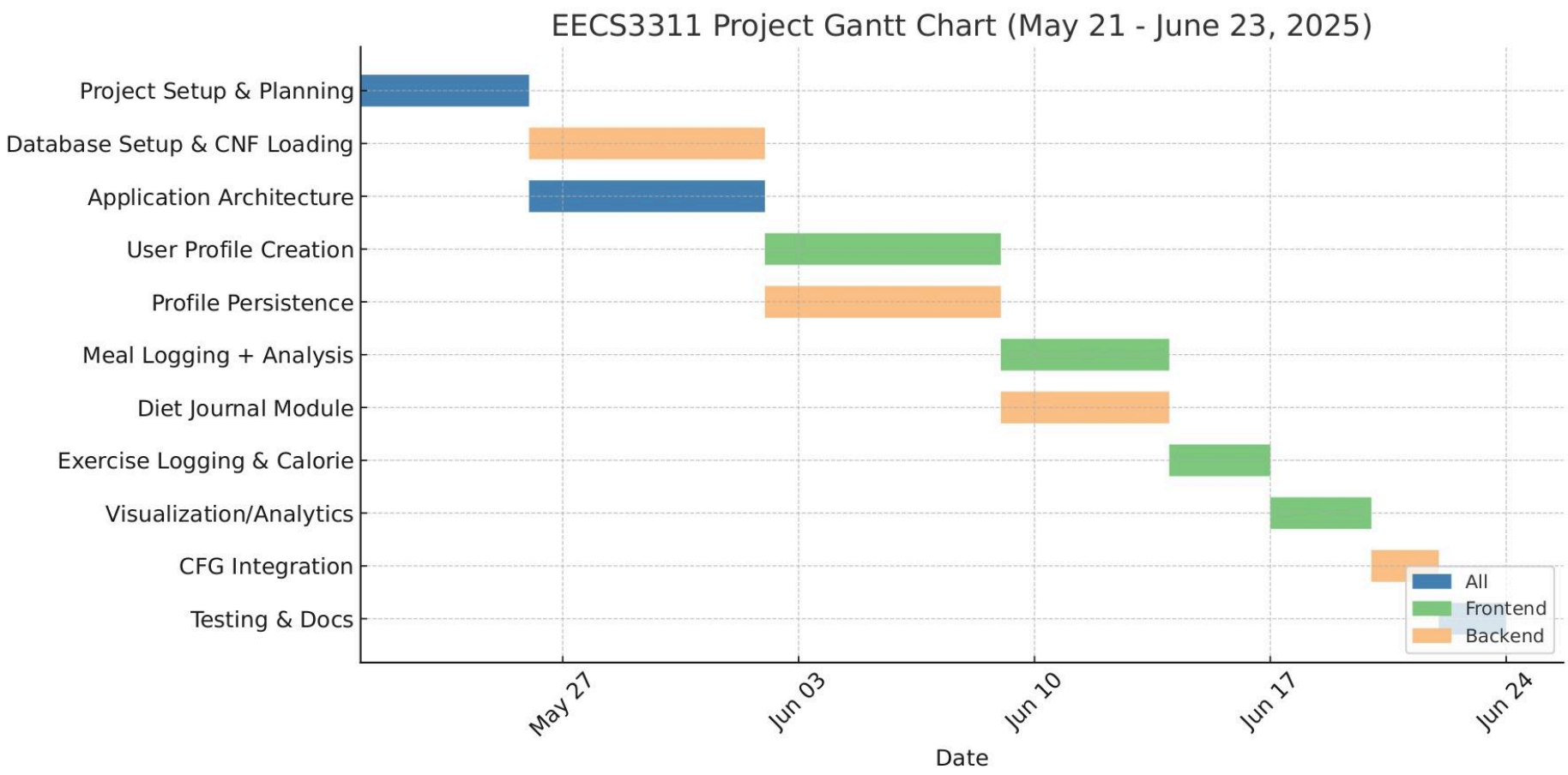
- **PB-012**: Calorie Intake vs Exercise Visualization
    - Time period selection interface
    - Daily calorie intake plotting
    - Total daily energy expenditure visualization
- **PB-013**: Nutrient Analysis Visualization
    - Average daily nutrient intake calculations
    - Top 5-10 nutrients pie chart visualization
    - "Other nutrients" grouping functionality
- **PB-014**: Recommended Daily Portions Comparison
    - Visual notifications for nutrition recommendations
    - Progress indicators for daily goals
- **PB-015**: Weight Loss Prediction System
    - Future date selection interface
    - Fat loss calculation (1kg fat = 7,700kcal)
    - Prediction visualization based on current patterns

Case 6: Canada Food Guide Integration

- **PB-016**: Food Group Classification
    - Map CNF data to Canada Food Guide categories
    - Calculate food group percentages in meals
- **PB-017**: CFG Compliance Analysis
    - Average plate visualization
    - CFG recommendation comparison
    - Compliance scoring and feedback

Case 7: System Quality and Testing

- **PB-018**: Comprehensive Test Suite
    - Unit tests for all core functionalities
    - Integration tests for database operations
    - GUI testing for user interactions
- **PB-019**: Error Handling and Validation
    - Input validation for all user entries
    - Database error handling
    - User-friendly error messages
- **PB-020**: Performance Optimization
    - Database query optimization
    - GUI responsiveness improvements
    - Large dataset handling



8. Group Meeting Logs

Meeting #1: Project Kickoff

- **Date:** May 29, 2025
- **Attendees:** Miguel, Sungwon, Sanya, Philips
- **Topics Discussed:**
    - Review of course project requirements and deliverable 1
    - Discussed overall team goals and expectations
    - Agreed on weekly meeting schedule and communication (Discord, Google Drive)
    - Initial division of assignment task (Profile, Meal, Exercise, DB, GUI, Testing)
    - Set up GitHub repository and invited members
- **Decisions Made:**
    - Use Google Docs for collaborative editing
    - Project architecture will be modular (Profile, Meal, Exercise, DB, GUI)
    - Each member responsible for documentation of their assigned task
- **Task Assignments:**
    - Miguel: Draft initial backlog and critical timeline
    - Sanya: Begin MySQL schema design and sequence diagram setup
    - Sung: Research GUI frameworks (Java Swing/JFreeChart)
    - Philips: Start preparing test plan and read TDD requirements

---

Meeting #2: Profile & Database Planning

- **Date:** June 3, 2025
- **Attendees:** Philips, Sanya, Miguel, Sungwon
- **Topics Discussed:**
    - Progress update: Philips completed initial DB schema; Sanya drafted backlog
    - Detailed design for profile creation (required fields, validation)
    - Database structure for user profiles and meals
    - Integration plan between frontend and backend modules

- **Decisions Made:**
    - Profile includes height, weight, DOB, sex, units
      All DB access through DatabaseManager class
    - Architecture has to follow the sequence diagram
- **Task Assignments:**
    - Miguel: Finalize backlog and update in Google Doc and work on architecture
    - Philips: Implement DB schema and connection classes
    - Sanya: Design profile input form and validation logic
    - Sung: Research design patterns and relate to deliverable

---

Meeting #3: Meal Logging, Integration, and Visualization

- **Date:** June 8, 2025
- **Attendees:** Philips, Miguel Sanya, Sungwon
- **Topics Discussed:**
    - Implementation status: Profile GUI built; DB connection tested
    - Meal logging UI requirements and user workflow
    - JFreeChart library demo and first chart prototype
    - Planning first round of integration testing
    - Review draft of 10 initial test cases
- **Decisions Made:**
    - Meals logged by user, date, meal type, ingredients, and quantity
    - Nutritional analysis to be displayed after meal is logged
    - Use JFreeChart for nutrient breakdown visualization
    - Test cases will cover profile, meal, and chart functions
- **Task Assignments:**
    - Philips: Integrate profile and meal UI, update documentation
    - Sung: Implement nutrient lookup in backend
    - Sanya: Add JFreeChart to meal logging panel
      Philips: Write unit tests for profile and meal modules

Meeting #4: Pre-Submission Review and Testing

- **Date:** June 18, 2025
- **Attendees:** Sanya, Sungwon, Miguel, Philips
- **Topics Discussed:**
    - Completed integration testing and bug fixes
    - Documentation review and final editing
    - Verification of at least 10 passing test cases
    - Checklist for final submission on eClass/GitHub
- **Decisions Made:**
    - Freeze code for deliverable 1 (only bug fixes allowed)
    - Double-check all features against backlog and deliverable list
    - Each member to submit a project contribution statement
- **Task Assignments:**
    - Sanya: Finalize and submit documentation, ensure project archive ready
    - Philips: Database backup and clean code comments
    - Miguel: Prepare screenshot evidence for visualization/chart
    - Sung: Complete and submit test results

9. Test Driven Development

The following table outlines the developed test cases for the extension, detailing the test ID, covered requirements, initial conditions, procedures, and expected outcomes. Each entry ensures the feature behaves as intended under specified conditions.

Test ID	Category	Requirements Coverage	Initial Condition	Procedure	Expected Outcome	Notes
TC01	Profile Creation	UC1-CreateProfile	Application is running; no profile exists	1. The user selects “Create Profile.” 2. Enters valid name, sex, age=25, height=180 cm, weight=75 kg, units=metric. 3. Clicks “Save.”	Profile is saved in memory/database; confirmation message appears; ProfileMain displays correct BMI.	Validates basic profile fields; tests success path.
TC02	Profile Creation (Validation)	UC1-CreateProfile	Application is running; no profile exists	1. The user selects “Create Profile.” 2. Leaves “name” blank; fills other fields. 3. Clicks “Save.”	Error message “Name is required” appears; profile is not saved.	Tests input validation (missing required field).
TC03	Profile Editing	UC1-EditProfile	A profile already exists (e.g., name=“TestUser”, age=30, etc.)	1. User selects existing “TestUser” profile. 2. Changes weight from 65 kg to 60 kg. 3. Clicks “Save.”	Profile is updated; new BMI is recalculated; updated fields are shown in ProfileMain.	Checks that editing persists and recalculates derived data.
TC04	Meal Logging	UC2-LogMeal	Application is running; “TestUser” profile active; CNF database stubbed	1. The user navigates to “Log Meal.” 2. Selects type=“Lunch”, date=today. 3. Adds ingredient “Egg”=100 g, “Bread”=50 g. 4. Clicks “Save Meal.”	Meal stored in memory; getTotalCalories() returns 100 g1.55 + 50 g2.5 = 155 + 125 = 280 kcal; summary prints “Total Calories: 280 kcal.”	Verifies normal meal-logging path with stubbed nutrient lookup.
TC05	Meal Logging (Missing Data)	UC2-LogMeal	Application is running; “TestUser” profile active	1. The user navigates to “Log Meal.” 2. Selects type=“Dinner”, date=blank. 3. Adds “Tomato”=50 g. 4. Clicks “Save Meal.”	Error message “Date is required” appears; meal is not stored.	Validates required “date” field.
TC06	Meal Logging (Unknown Food)	UC2-LogMeal	Application is running; “TestUser” profile active; CNF lookup stub returns no entry for “Dragonfruit”	1. The user navigates to “Log Meal.” 2. Selects type=“Snack”, date=today. 3. Adds “Dragonfruit”=50 g. 4. Clicks “Save Meal.”	Error message “Ingredient not found: Dragonfruit” appears; meal is not saved.	Ensures invalid ingredient is rejected.

TC07	Exercise Logging	UC2-LogMeal (Exercise)	Application is running; “TestUser” profile active	1. User navigates to “Log Exercise.” 2. Selects type=“Running”, date=today, duration=30 minutes. 3. Clicks “Save Exercise.”	Exercise stored; getCaloriesBurned() returns 30 * 5 = 150 kcal; summary prints “Calories burned: 150 kcal.”	Tests basic exercise logging with stubbed burn rate.
TC08	Swap Recommendation	UC3-RequestSwap	“TestUser” has logged at least one meal containing “Bread”	1. User selects “Suggest Swap.” 2. Choosing goal=“reduce calories.” 3. Choosing food=“Bread.” 4. Clicks “Get Swap.”	SwapEngine returns “lettuce wrap”; suggestion displayed in UI.	Verifies correct suggestion for reduce-calories scenario.
TC09	Swap Recommendation (Multi)	UC3-RequestSwap	“TestUser” has logged meals containing both “White Rice” and “White Bread”	1. User selects “Suggest Swap.” 2. Choosing goal=“increase fiber.” 3. Chooses first food=“White Rice”→“Get Swap”. 4. Chooses second food=“White Bread”→“Get Swap.”	First swap→“brown rice”; second→“whole wheat bread”; both suggestions display.	Tests multiple swap suggestions under one goal.
TC10	Before/After Comparison	UC4-CompareSwaps	User has an original meal and a suggested swap for “Bread”	1. User selects a logged meal (Egg, Bread). 2. User applies suggested swap (Bread→Lettuce wrap). 3. System displays two side-by-side summaries (original vs. swapped meal).	Original “Total Calories: 280 kcal”; swapped “Total Calories: 100 g Egg*1.55* + *Lettuce wrap*1.0=155 + 50 =205 kcal.”Swap highlights “Bread→Lettuce wrap.”	Checks that comparison view shows correct nutrient deltas.
TC11	Chart Generation	UC6-VisualizeIntake	User has logged multiple meals over 3 days	1. User selects “Visualize Daily Intake.” 2. Chooses 3-day date range. 3. Clicks “Generate Chart.”	Pie chart window appears showing percentages of Protein, Carbs, Fats, Other based on logged meals.	Verifies that ChartDemo (or equivalent) can render with sample data.
TC12	CFG Adherence Calculation	UC7-CompareToFoodGuide	UC7-CompareToFood Guide	1. User selects “Compare to Canada Food Guide.” 2. Chooses a date range containing that meal. 3. Clicks “Compare.”	Textual/visual indicator shows percentage of each food group vs. CFG—e.g., Veggies 30% vs. CFG 40%, Fruits 10% vs. CFG 10%.	Ensures basic CFG adherence logic (stub or static thresholds).

EECS3311 Project

Contribution Attestation

This note is to confirm that the contribution of each group member for this proposal was approximately equal.

• Group Members (print)	Group Members (sign)	Date
Sungwon Chung	**Sungwon Chung**	**23/06/2025**
Franco Miguel Dela Cruz		**23/06/2025**
Philips Rhoguns		**23/06/2025**
Sanya Malhotra		**23/06/2025**