

HEALTH-N-CARE APP / RELEASE 1

Project Design Document

TEAM 3 Charlie

Zaher Asad <zsa4432@rit.edu>

Ethan Gapay <etg5588@rit.edu>


Allan Flores <arf7094@rit.edu>

Alfred Franco Salgado <af8857@rit.edu>

Alex Tedesco <act2076@rit.edu>

Nolan J Wira <njw1389@rit.edu>

Google Doc link to document with version history:

 Design Documentation.docx

1 Project Summary

The HealthNCare App is developed for a Wellness Consortium aimed at assisting users in monitoring and improving their daily lives. The app allows users to track their food intake, weight, and daily calorie goals. Users can add basic foods or recipes, log their daily food consumption, and view nutritional summaries. The app provides the user with a simple user interface and follows the MVC pattern, with an emphasis on the Composite pattern.

Key Features:

- The ability to add/define basic foods with nutrient information.
- The ability to add/define recipes composed of basic foods or other recipes.
- The ability to log a specific food intake with a specific quantity and date attached.
- The ability to view your total daily nutrition intake, along with basic recommendations.

2 Design Overview

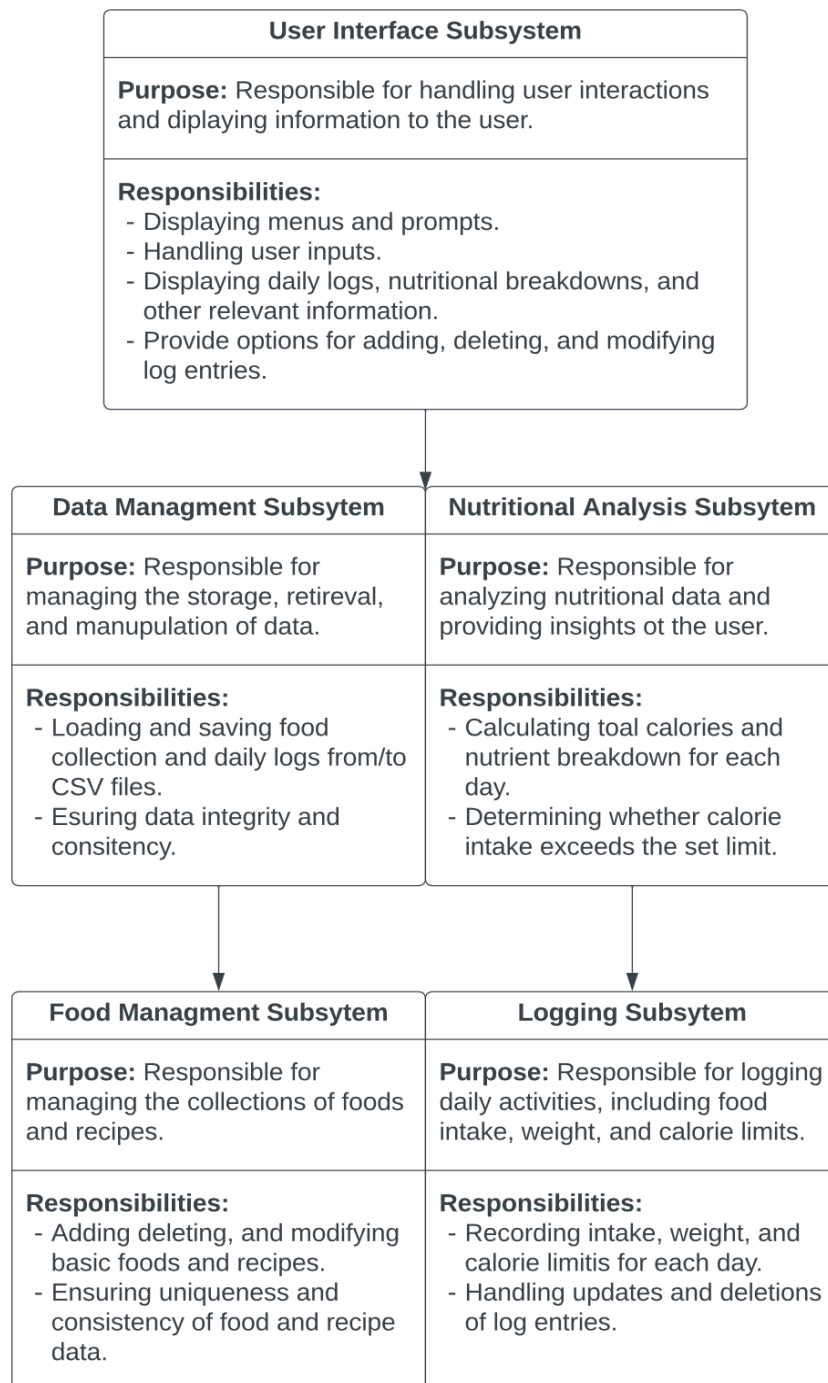
The development of an application designed to keep track of daily food intake must carefully consider the scalability and ease of use available to the user. The goal is to create a program that can be personalized to a person's general diet and remain usable on a daily basis. This stresses the importance of modular code and a reasonably structured back-end. Naturally, our ideas and implementations towards achieving this goal will change over time and this section is tasked with recording those changes and their rationales.

Working on the design sketch was the first instance of actual critical ideation into designing the implementation of our health app. With some guidance from the slides and descriptions of the app's inner workings, we extrapolated our first design. At the top level, we intended to implement a Model-View-Controller (MVC) architecture, which should function well for our needs as it separates the data from the behavior put upon it.

While contemplating our design, we identified the 3 main areas our system needed to be divided into: the UI, Food management (adding/tracking foods), and data management (saving and retrieving log data). We decided to divide the last 2 further to keep them focused and cohesive. In our current design implementation, the food subsystem is considered separate from the controller that manages it, along with the log sub-system is separate from the log controller. We chose to visualize it as such since this helps us support our separation of concerns.

With the knowledge that the user needed to be able to add foods with nutrient information at a base level and be able to use those foods to create recipes, which can then be used in other recipes, we decided to use the composite pattern as it fulfilled our scalability requirements. With this structure, we should be able to compose foods from other foods easily and treat them similarly when tracking them. We intend to use observers to update the UI, to reduce coupling between systems.

3 Subsystem Structure



4 Subsystems

4.1 User Interface Subsystem

Class UserInterface	
Responsibilities	Display Menus and prompts Handle user inputs Display daily logs and nutritional breakdowns

Class LogDisplay	
Responsibilities	Handles displaying logs
Collaborators (Uses)	UserInterface

Class FoodDisplay	
Responsibilities	Handles displaying food information
Collaborators (Uses)	UserInterface

4.2 Nutritional Log Subsystem

Class LogController	
Responsibilities	Load and save daily logs from/to CSV files. Ensure data integrity. Ensure data consistency.

4.3 Food Management Subsystem

Class FoodController	
Responsibilities	Load and save food collection from/to CSV files. Calculate total calories and nutrient breakdown for each day. Determine if calorie intake exceeds set limit.

4.4 Food Subsystem

Class Food (abstract)	
-----------------------	--

Responsibilities	Add, delete, and modify basic foods and recipes. Ensure uniqueness and consistency of food and recipe data.
-------------------------	--

Class BasicFood	
Responsibilities	Represents a food item. Includes calories, fat, carbs, protein, sodium.
Collaborators (extends)	Food

Class Recipe	
Responsibilities	Represents a recipe item. Can be composed of different BasicFood and Recipe
Collaborators (extends)	Food

4.5 Log Subsystem

Class Log	
Responsibilities	Entries of daily log

Class DailyLog	
Responsibilities	Record food intake, weight, and calorie limits for each day. Handle updates and deletions of log entries.
Collaborators (uses)	Log

Class FoodEntry	
Responsibilities	Record the user's food intake Record the number of servings per food
Collaborators (extends)	DailyLog

Class CalorieLimit	
Responsibilities	Record the current date Record daily calorie limits
Collaborators (extends)	DailyLog

Class Weight	
Responsibilities	Record the current date Record the user's weight
Collaborators (extends)	DailyLog

5 Sequence Diagrams

Scenario: The scenario described is reading in a food database with the following contents:

- Three basic foods
- A recipe containing two of those basic foods
- Another recipe that includes the first recipe plus the remaining basic food

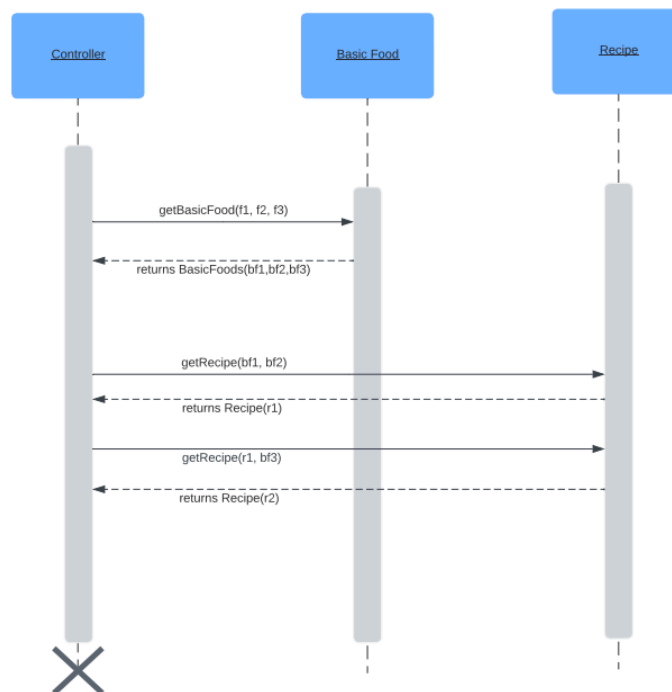
Features: The key features illustrated in the sequence diagram are:

1. The Controller makes a call to `getBasicFood(f1, f2, f3)` on the BasicFood component, which returns the basic food data.
2. The Controller then calls `getRecipe(bf1, bf2)` on the Recipe component to get the first recipe containing two basic foods.
3. Next, the Controller calls `getRecipe(r1, bf3)` to get the second recipe, which combines the first recipe (r1) with the remaining basic food (bf3).
4. Finally, the second recipe is returned to the Controller.

Operation: The overall operation being depicted is the Controller querying the food database to:

1. Retrieve the basic foods
2. Construct a recipe from two basic foods
3. Construct another recipe by combining the first recipe with the third basic food
4. Return the final composite recipe

So in summary, this sequence diagram illustrates the steps for the Controller to query a food database, retrieve basic foods, build recipes by combining those foods and other recipes, and ultimately return a final recipe that is a composition of foods and sub-recipes. The key operation is progressively building up more complex recipes from basic ingredients and simpler recipes.



Scenario: Adding two servings of a food to the log entry for the current date.

Features:

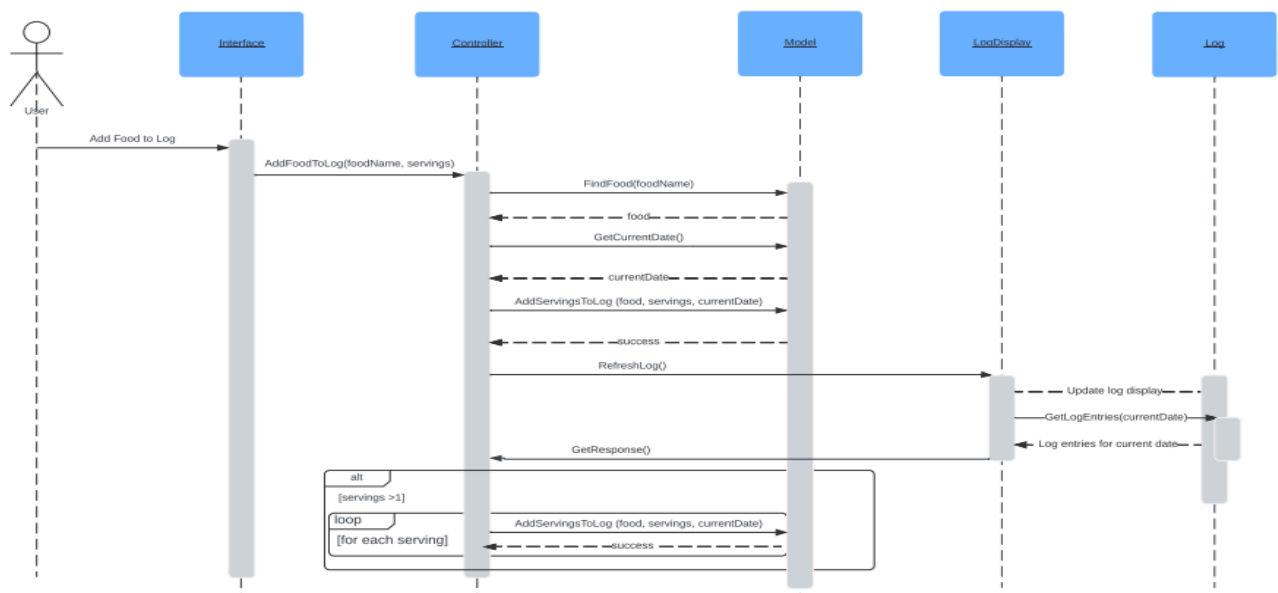
1. The User initiates the process by calling AddFoodToLog() on the Interface.
2. The Interface communicates with the Controller to addFoodForLogModel(name, servings).
3. The Controller interacts with the Model to:
 - FindFoodByName()
 - GetCurrentDate()
 - AddServings(food, servings, currentDate)
4. The Model updates the log entry for the current date with the added food servings.
5. The Model returns success to the Controller.
6. The Controller instructs the Interface to UpdateLogDisplay().
7. The Interface calls GetLogEntries(currentDate) on the LogDisplay.
8. The LogDisplay retrieves the updated log entries from the Model and displays them to the User.

Operation: The main operation is adding food servings to the log entry for the current date.

This involves:

1. User input (food name and number of servings)
2. Retrieving the food object based on the name
3. Getting the current date
4. Updating the log entry in the Model with the added food servings
5. Displaying the updated log to the user

The sequence diagram illustrates the interaction between components (Interface, Controller, Model, LogDisplay) to accomplish this operation, showing the flow of data and actions from the user's input to updating the display with the modified log entry.



Scenario: Computing the total number of calories for the current date, where the log contains a basic food and a recipe made up of two basic foods.

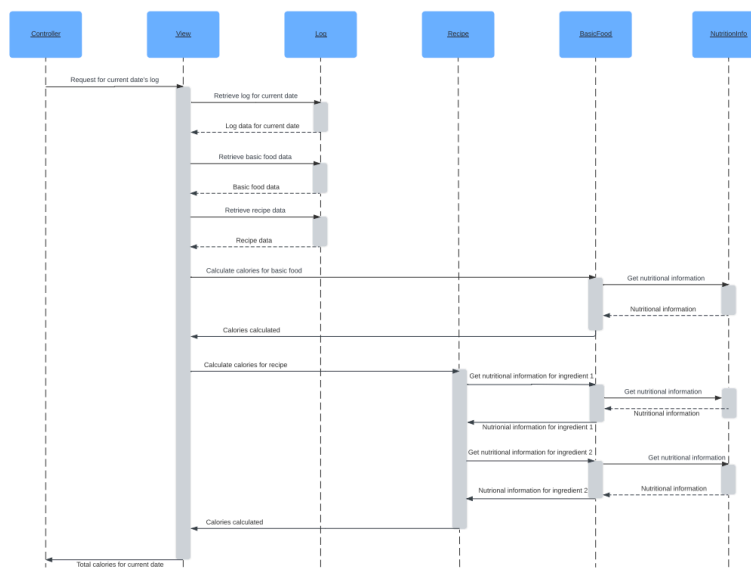
Features:

1. The Controller requests the current date's log from the View.
2. The View retrieves the log data for the current date from the Log component.
3. The Log returns the basic food data and recipe data for that date.
4. For each basic food item, the Controller:
 - Retrieves the food data from the BasicFood component
 - Calculates the calories by multiplying servings and calories per serving
5. For the recipe, the Controller:
 - Retrieves the recipe data from the Recipe component
 - Calculates the calories for each basic food in the recipe
 - Sums up the calories from the recipe's basic foods
6. The Controller calculates the total calories by summing the calories from basic foods and the recipe.
7. The total calories are returned to the View for display.

Main Operation: The primary operation is calculating the total calorie count for the current date's log entry, which includes both individual basic foods and a recipe. This involves:

1. Retrieving the log data for the specified date
2. Fetching the details for each basic food and recipe in the log
3. Computing the calories for each basic food based on servings and calories per serving
4. Calculating the calories for the recipe by summing its constituent basic foods' calories
5. Summing the calories from basic foods and the recipe to get the total calorie count
6. Displaying the total calories to the user

The sequence diagram illustrates the interaction between the Controller, View, Log, BasicFood, and Recipe components to achieve this calorie calculation operation, showing the flow of data and the steps involved in computing and aggregating the calorie information.



6 Pattern Usage

6.1 Pattern #1

Adapter Pattern	
Socket(s)	Log
Adapter(s)	DailyLog

6.1 Pattern #2

Factory Pattern	
Super-class(es)	Food
Sub-class(es)	BasicFood

6.1 Pattern #3

Observer Pattern	
Observer(s)	UserInterface
Observable(s)	FoodDisplay LogDisplay

6.1 Pattern #4

Composite Pattern	
Client	MVC
Component	Food
Composite	Recipe
Leaf	BasicFood

6.1 Pattern #5

Mediator Pattern	
Mediator	DailyLog
Components	Weight FoodEntry CalorieLimit

7 RATIONALE

Design Decisions:

Scenario 1,2,& 3 Sequence Diagrams:

The sequence diagrams demonstrate a Model-View-Controller architecture pattern where the controller would be interacting with a user to get basic foods and then gets the recipes from the basic foods upon the user's request. The Model-View Controller model also prevents the user from directly interacting with the data saved into the CSV, which would make the application more secure.

Pattern Usages:

- **Adaptor Pattern:** Selected for the food and weight log because it bridges both classes that will need to communicate with each other to get relevant information.
- **Factory Pattern:** Chosen to handle the different kinds of foods because it can distinguish between the choices while having them on the same interface.
Observer Pattern: Used to alert/update the interface when changes are made to the various displays.
- **Composite Pattern:** Chosen to assist Model-View Controller in distinguishing between different kinds of foods, and what operations should be performed.
Mediator Pattern: Organizes and coordinates all the tools and attributes of the daily log without causing them to directly interact with each other. This keeps the logic contained and encapsulated for ease of operations.

Subsystems:

- **User Interface:** This is what a person interacts with.
- **Nutritional Analysis:** Handles all nutritional functions.
- **Data Management:** Designates saving food into the CSV database to one class, it ensures integrity and consistency.
- **Food Management:** Contains the functions that provide the main aspect of the application such as loading food, and modifying basic food and ingredients.
- **Food Display:** displays information about food to a user.
- **Food:** Add/delete/modify food.
- **Logging:** Controls all aspects of logging.