HealthNCare App / 1

Project Design Document

#4 Team4AndCo.LLC.INC
Izzy Pedrizco-Carranza <ip7431@rit.edu>
Derek Kasmark <djk2529@rit.edu>
Colin Chomas <cwc6640@rit.edu>
Adhel Siddique <as8758@rit.edu>
Philip Snyder <pjs7207@rit.edu>

Google Doc link to document with version history:

https://docs.google.com/document/d/1ixyqrHVhnMYQA3XHvu1HFkSmE65J_aWUPPTi8YTze80/edit?usp=sharing

1 Project Summary

An application we are developing is a health care app for users who want to log in their daily intake of foods and store food recipes. Additionally, this allows users to monitor and use data from the app to improve their daily lives. The application provides a list of basic foods which users can combine into recipes as well as adding new basic ingredients or recipes to the system. The information they must provide is the name of the food, calories, grams of fat, grams of proteins, and many more.

Moreover, the application has a feature of storing log entries for daily intake of consumption daily. The purpose of this is users can look at what food they have eaten and how much they need to adjust to their daily intake to improve their health.

Overall, users will be able to use the health food app that can improve their daily lives as well as carefully analyze what food they can add and make their daily intake. They can log their daily intake to get specific intake they need such as proteins, carbs, and many more. In addition, they can edit or delete some of their log entries based on their preferences.

2 Design Overview

The design of our healthcare application evolved from initial sketches to its final implementation with a focus on modularity, scalability, and maintainability. We structured the system into four main subsystems:

1. **User Management** – Handles authentication and profile settings.

2. **Food Database** – Stores predefined food items and user-added entries.

3. **Recipe Management** – Enables users to create and save meal combinations.

4. **Daily Log System** – Records user food consumption for health tracking.

Key Design Principles Applied

- Separation of Concerns: Each subsystem operates independently, reducing unnecessary dependencies.
- High Cohesion & Low Coupling: Components interact via well-defined interfaces, making the system easier to modify and extend.
- Dependency Inversion: Abstractions (interfaces) allow for future feature expansions without affecting existing implementations.

Design Decisions & Adjustments

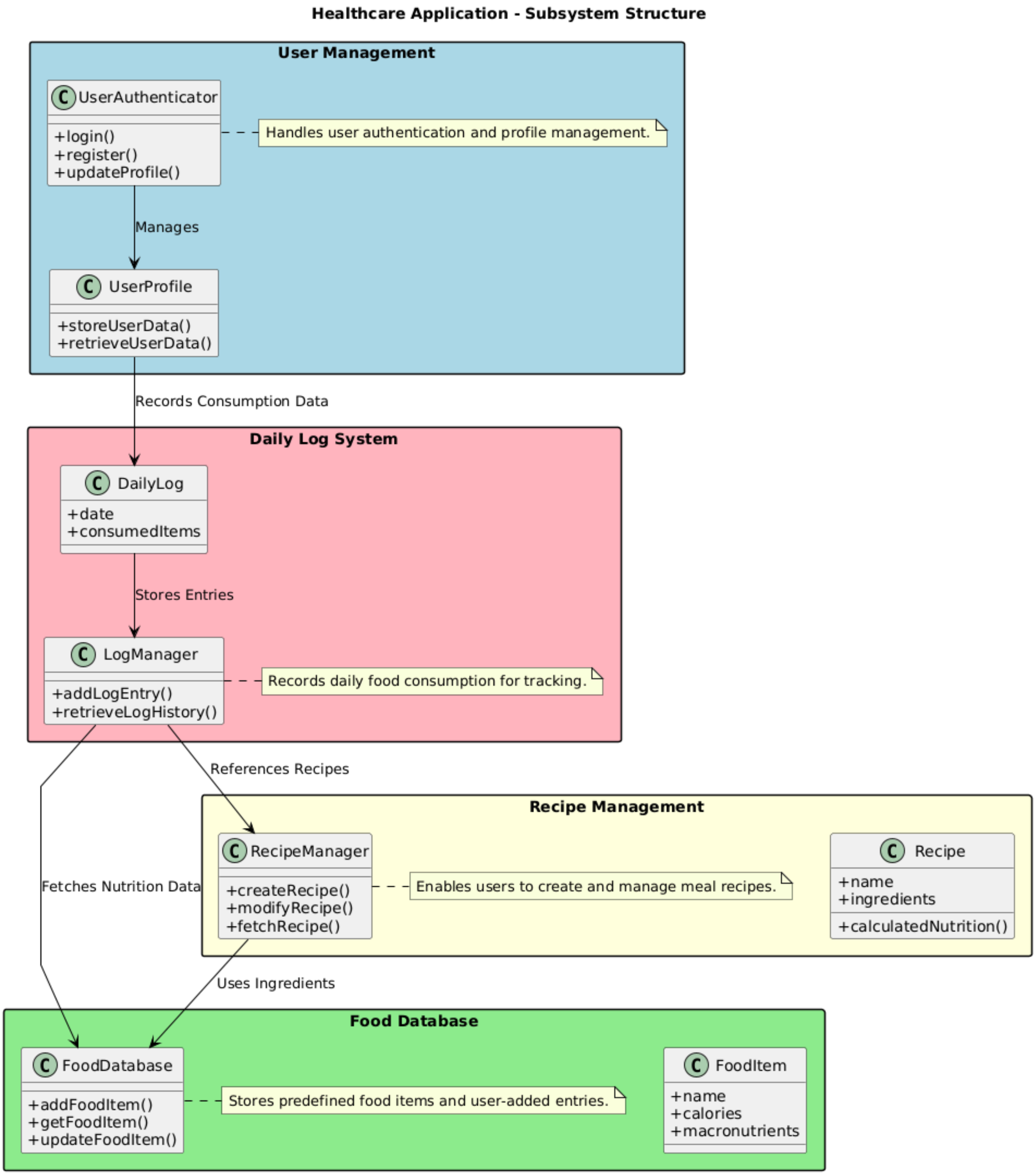
- Initially, a monolithic database structure was considered but later revised to a modular approach for improved efficiency.
- The original manual food entry system was enhanced with auto-suggestions and predefined food categories for better usability.

Assumptions & Future Considerations

- Users are expected to input accurate nutritional data when adding new food items.
- The system assumes internet connectivity for cloud-based storage.
- Future versions may incorporate machine learning for personalized health recommendations.

3 Subsystem Structure

The system is divided into four key subsystems, each handling a specific functionality while maintaining low coupling and high cohesion. Below is a graphical representation of the subsystem relationships, showing how data flows between different components in the application.



4 Subsystems

CRC Tables for Each Subsystem

1. User Management Subsystem

Class	Responsibilities	Collaborators
UserAuthenticator	Handles user login, registration, and profile authentication.	UserProfile
UserProfile	Stores and retrieves user-related data.	DailyLog

2. Food Database Subsystem

Class	Responsibilities	Collaborators
FoodItem	Represents a food entry with name, calories, and macronutrients.	FoodDatabase
FoodDatabase	Stores and manages food items, allowing add/update/retrieve operations.	LogManager, RecipeManager

3. Recipe Management Subsystem

Class	Responsibilities	Collaborators
Recipe	Represents a user-created recipe containing multiple food items.	RecipeManager, FoodDatabase
RecipeManager	Allows users to create, modify, and fetch recipes.	LogManager, FoodDatabase

4. Daily Log System Subsystem

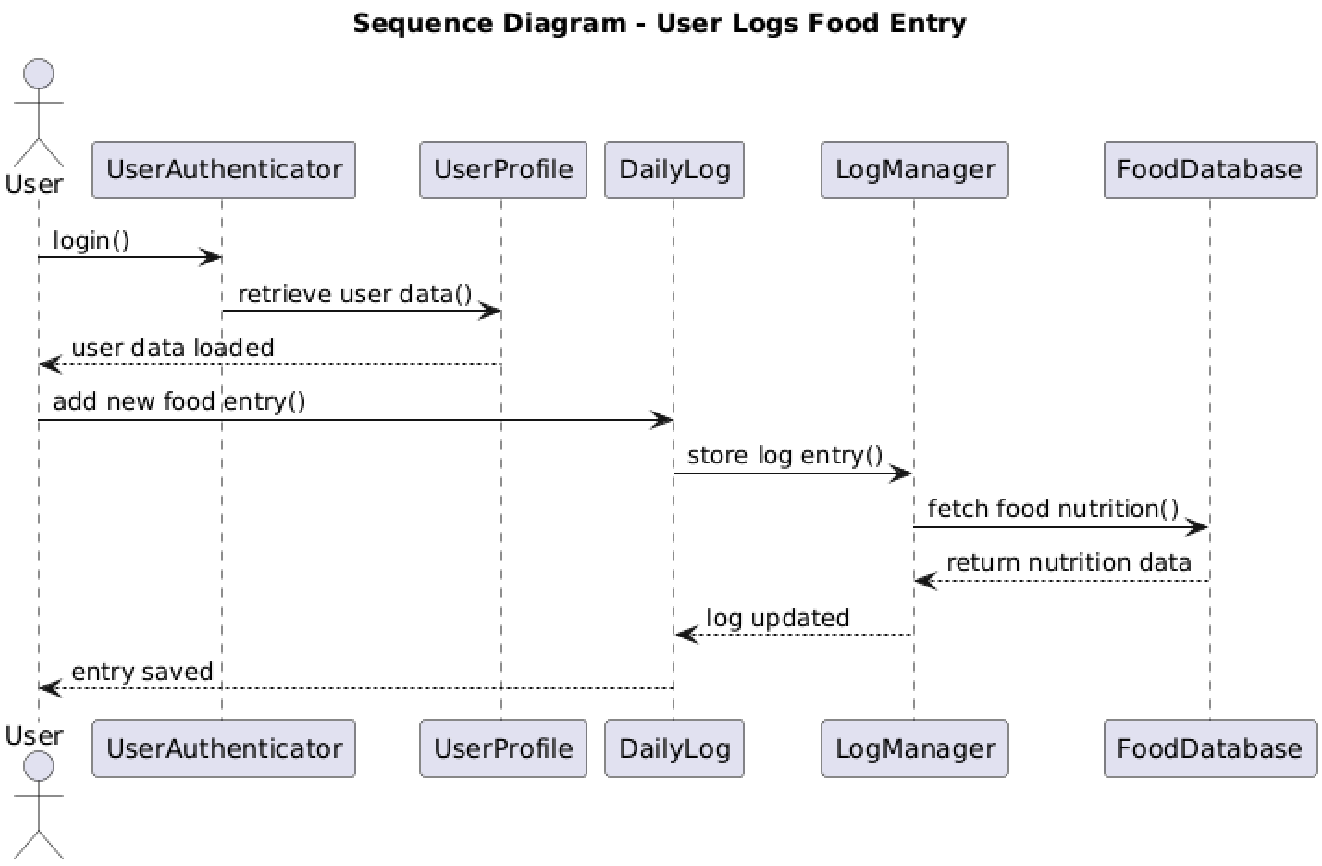
Class	Responsibilities	Collaborators
DailyLog	Stores daily food intake entries for users.	LogManager, UserProfile
LogManager	Handles log entries and retrieves consumption history.	DailyLog, FoodDatabase, RecipeManager

Reference to Subsystem Diagram

The class relationships for each subsystem are already shown in the Subsystem Structure section. Please refer to that diagram for a clear view of how the classes interact and depend on each other.

5 Sequence Diagrams

Sequence Diagram #1: User Logs Food Entry



Description

This sequence diagram illustrates the process of a user logging a food entry.

1. The user logs into the system.

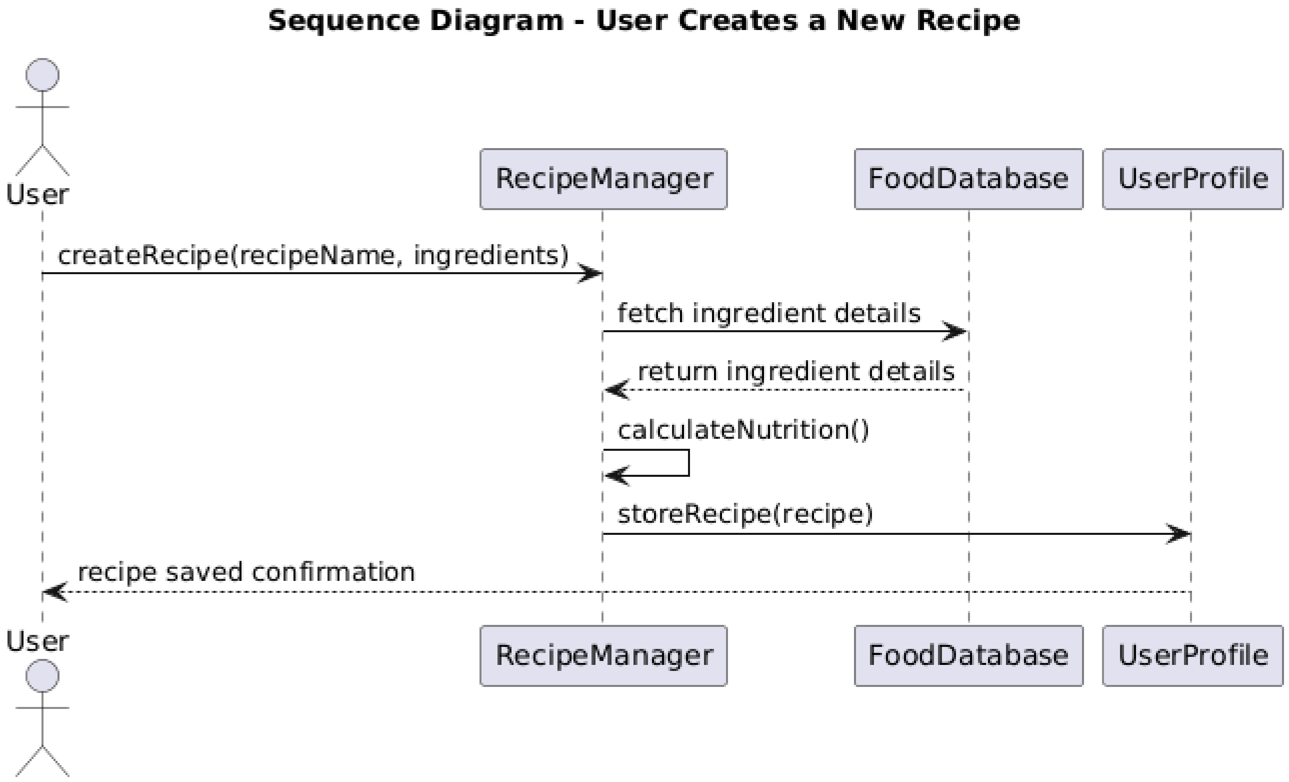
2. The UserAuthenticator retrieves their profile data.

3. The user enters a food log, which gets stored in DailyLog.

4. LogManager fetches nutritional data from FoodDatabase.

5. The updated log is saved, and the user receives confirmation.

Sequence Diagram #2: User Creates a New Recipe



Description

This sequence diagram illustrates the process of a user creating a new recipe.

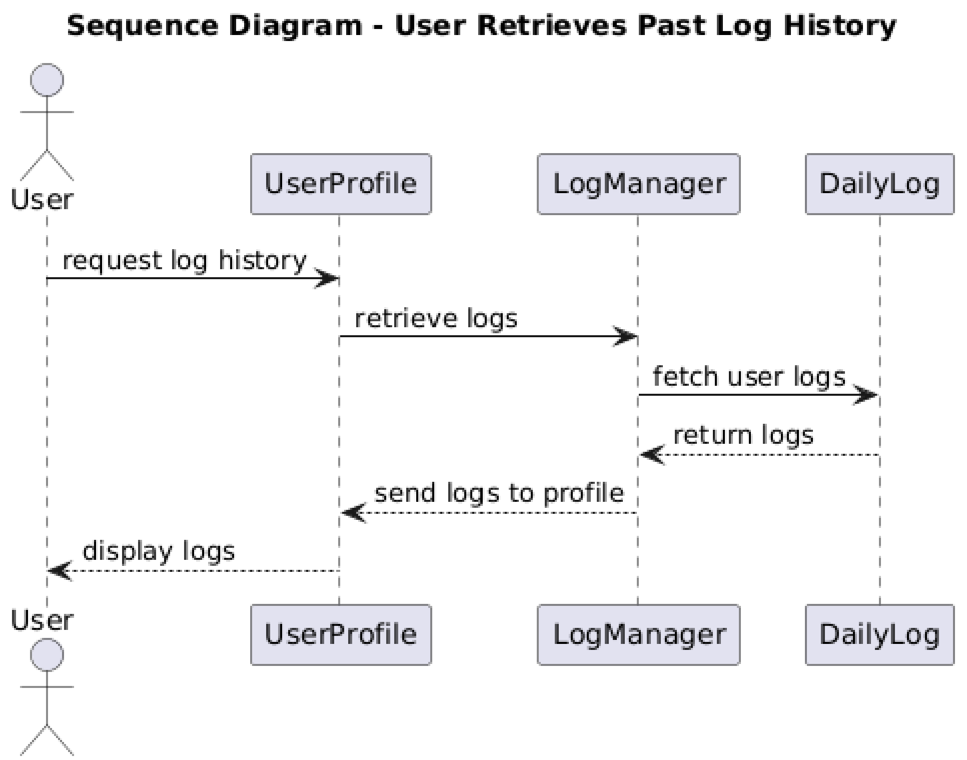
1. The user provides a recipe name and ingredients.

2. RecipeManager retrieves ingredient details from FoodDatabase.

3. It calculates nutrition values and saves the recipe in UserProfile.

4. The user receives a confirmation message that the recipe has been saved.

Sequence Diagram #3: User Retrieves Past Log History



Description

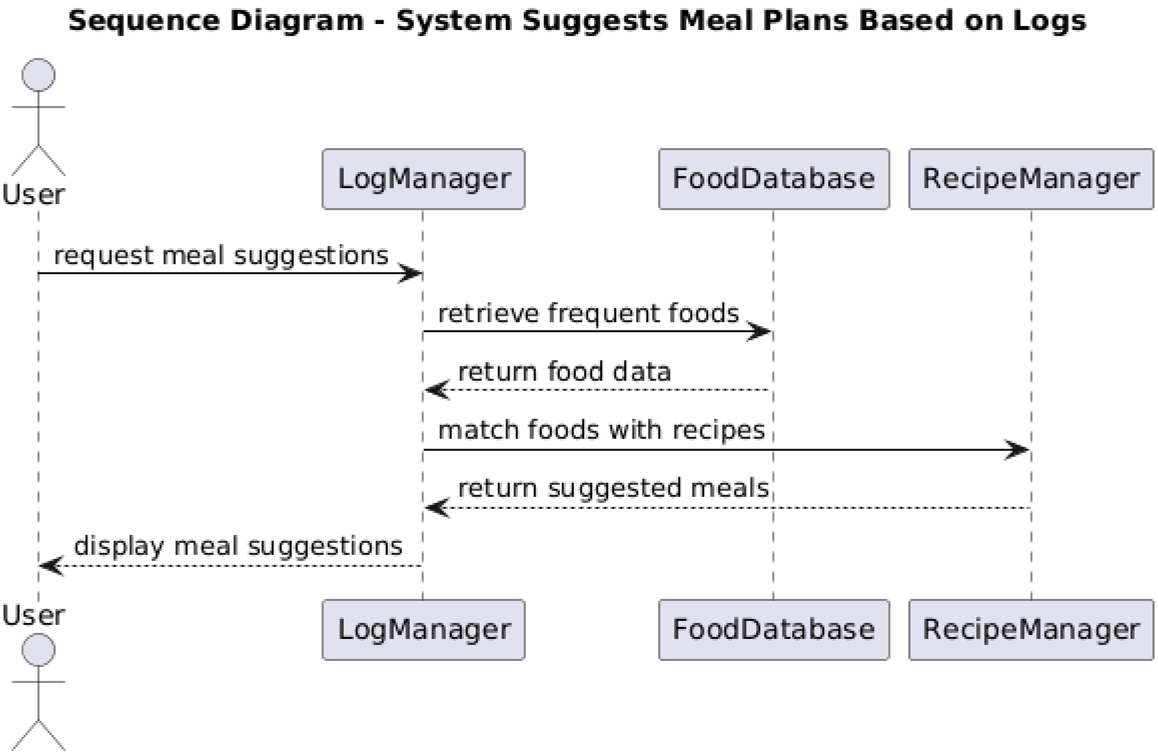
This sequence diagram illustrates how a user retrieves past food log entries.

1. The user requests their log history.

2. UserProfile contacts LogManager, which fetches logs from DailyLog.

3. The log data is retrieved and displayed to the user.

Sequence Diagram #4: System Suggests Meal Plans Based on Logs



Description

This sequence diagram illustrates how the system suggests meal plans based on the user’s food log history.

1. The user requests meal suggestions.

2. LogManager analyzes past food logs and queries FoodDatabase for commonly logged foods.

3. RecipeManager suggests recipes that match the user’s food history.

4. The system displays personalized meal recommendations to the user.

6 Pattern Usage

6.1 Composite

Composite Pattern	
Composite	Recipe
Leaf	FoodItem

6.2 Model-View-Controller

MVC Pattern	
Model	FoodItem Recipe DailyLog UserProfile
View	LogManager
Controller	UserAuthenticator RecipeManager

7 RATIONALE

This section outlines the major design and architectural decisions made throughout the project and the reasons behind them. These choices were made to ensure modularity, scalability, and maintainability while adhering to best software design principles.

Key Design Decisions & Justifications

1. Subsystem-Based Architecture

- Decision: We divided the system into four main subsystems:
- User Management (Handles authentication and profile management)
- Food Database (Stores food data and retrieves nutritional information)
- Recipe Management (Manages user-created recipes)
- Daily Log System (Logs user food consumption history)
- Rationale: This modular approach follows separation of concerns, ensuring that each subsystem operates independently while maintaining low coupling and high cohesion.

2. Use of Interfaces for Abstraction

- Decision: Interfaces (IRecipeManager, ILogManager, etc.) were introduced to abstract implementations.
- Rationale: This follows the Dependency Inversion Principle, allowing for future extensibility without modifying core system logic.

3. Choice of Composite Pattern in Recipe Management

- Decision: We applied the Composite Pattern for managing recipes and ingredients.
- Rationale: This allows a recipe to be treated as a collection of individual food items, making the system more flexible in handling nested structures.

4. Adoption of Model-View-Controller (MVC) Pattern

- Decision: The system follows an MVC architecture.

- Rationale:

- Model: Handles data operations (FoodItem, DailyLog, UserProfile)

- View: Displays data (LogManager)

- Controller: Manages user interactions (UserAuthenticator, RecipeManager)

- This approach ensures a clear separation of concerns and makes future modifications easier.

5. Transition from a Monolithic Database to a Modular Structure

- Decision: Initially, a single-table database was considered, but later, separate collections/tables were used for each major entity.

- Rationale:

- Reduces redundancy and improves query efficiency.

- Enhances scalability for future expansion.

- Easier maintenance since each table handles a specific function.

Summary of Design Changes Over Time

Date	Decision	Reasoning
2025-03-10	Decided to use MVC architecture	To improve maintainability and follow industry best practices.
2025-03-12	Transitioned from a monolithic database to a modular database	To improve scalability and optimize data retrieval.
2025-03-15	Applied the Composite Pattern in Recipe Management	To allow nested structures for recipes containing multiple ingredients.
2025-03-17	Added interfaces for dependency inversion	To support future system extensions without modifying core logic.