Yet Another Diet Manager

April 1st, 2025

Team Members: Harshith Sai Seera (2023101064) and Disha Pant (2023101085)

Overview

This is a command-line interface application that is designed to help users monitor their daily food intake and manage nutritional goals. Through this system, users can log their food consumption, track calories, and compare it with their own targets and goals.

Key features are as follows:

- Food database supporting basic and composite food items
- Keyword-based food search
- Daily food logging with serving sizes
- User profile management
- Multiple calorie calculation methods
- Undo functionality while logging
- Viewing historical and daily log data

The diet manager can be extended to incorporate more methods to calculate target calories fairly easily.

- The UserProfile class has an attribute for representing the method currently being used for calculating target calories.
- Each calorie calculation method (e.g., Harris-Benedict, Mifflin St. Jeor, Katch McArdle) is encapsulated as a separate private function in the UserProfile class.
- This makes it easier to add new methods without modifying the logic of existing methods.
- The method can be used by simply adding its respective function and having its name listed in one of the options for picking a method.

To extend this design to handle additional websites as data sources, without changes rippling through the system:

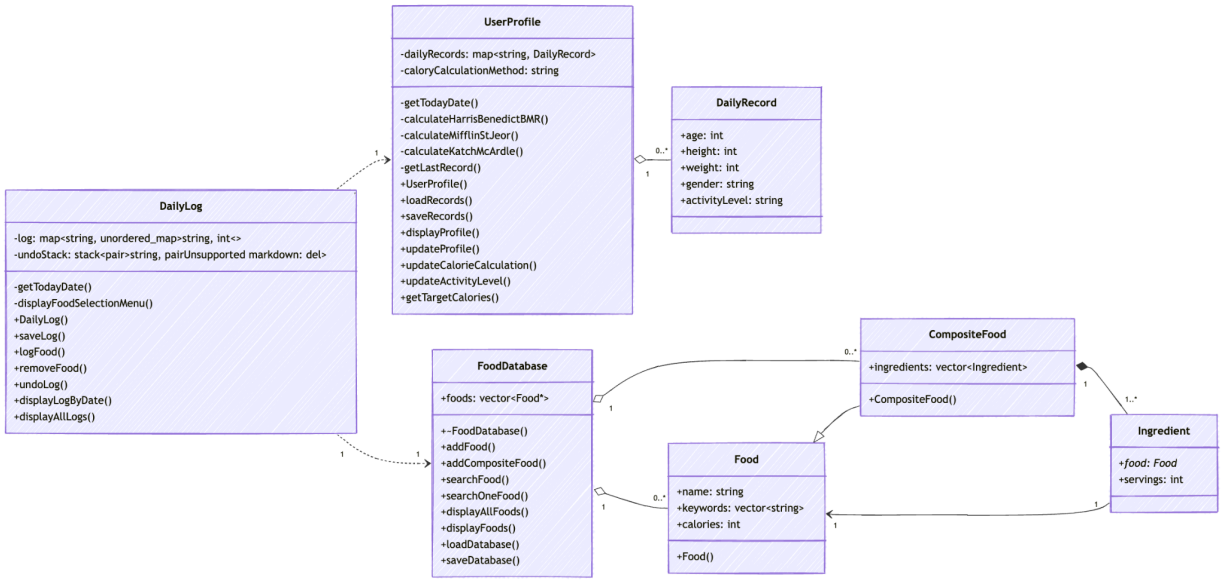
- We can create a generic FoodSourceAdapter class which would have a main method to actually get the food from the sources
- We can then create specific adapters for each food source using this base class which parse data from the respective source and convert it into a format like our Food class
- We can also have a SourceManager class to maintain a list of all possible sources
- Now, in FoodDatabase, we can have a method called importFoods(FoodSourceAdapter, query) which queries the respective adapter to get the foods required. It then adds these to the database.
- Lastly, we just need to update the UI to show possible sources, and let the user import foods from these.

To eliminate redundancy in the daily log's storage format, we have followed the following aspects:

- Date Mentioned Only Once: Each log entry starts with a date (31/03/2025|) with all items grouped under it
- Food Names Not Repeated: Each food appears once per day with aggregated servings (Peanut Butter,10)
- Compact Format: Uses minimal syntax (FoodName,Servings) with semicolons separating items
- Simplified Structure: Eliminates redundant field names and metadata for maximum efficiency

The design is segregated into classes, so adding any additional attributes relating to Food, or any other class is simple and can be done by adding onto the class definition header.

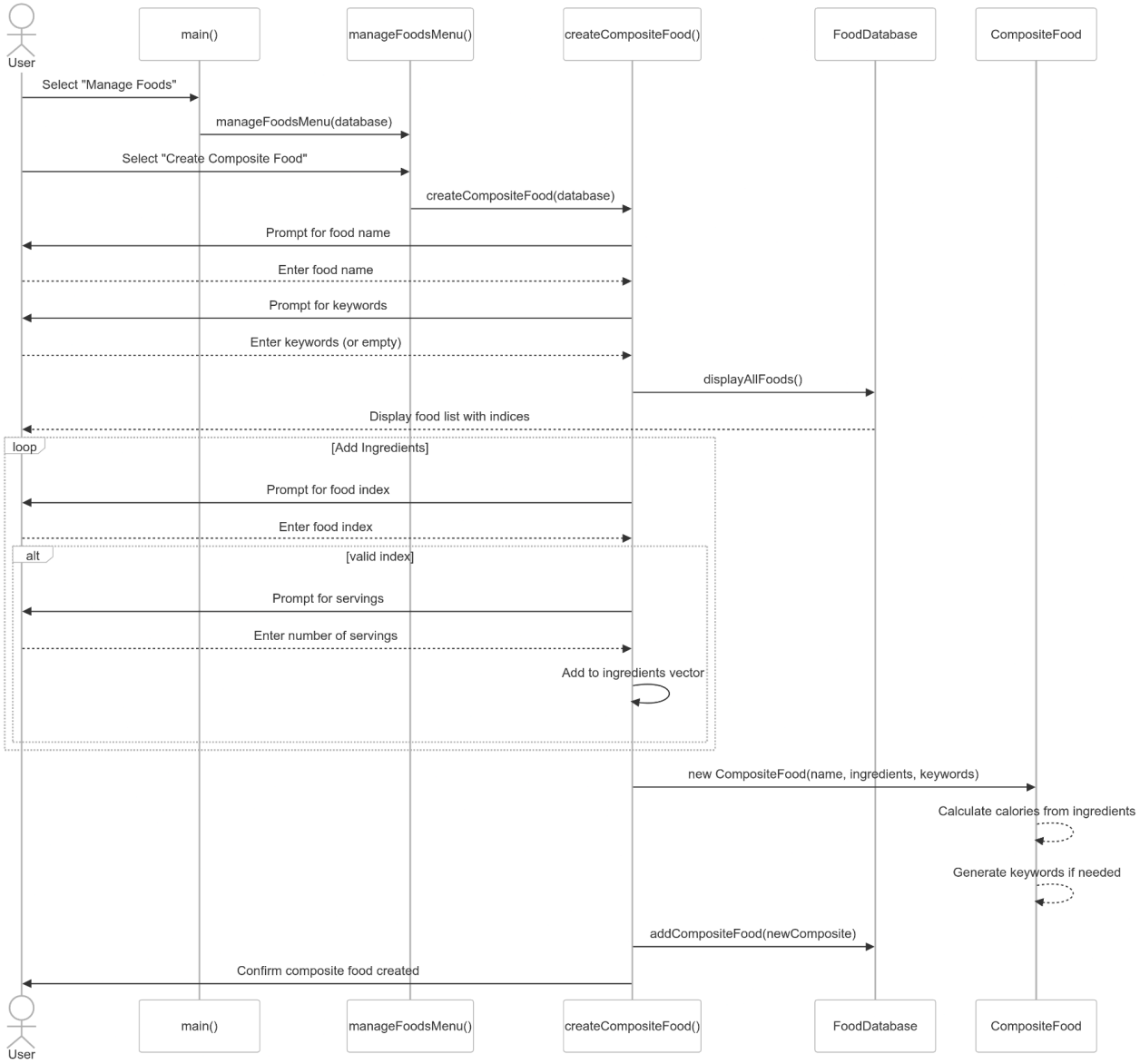
UML Class Diagram



https://drive.google.com/file/d/1XTcBu9JSR7-4GWdhBrWY8nyraO8_viru/view?usp=sharing

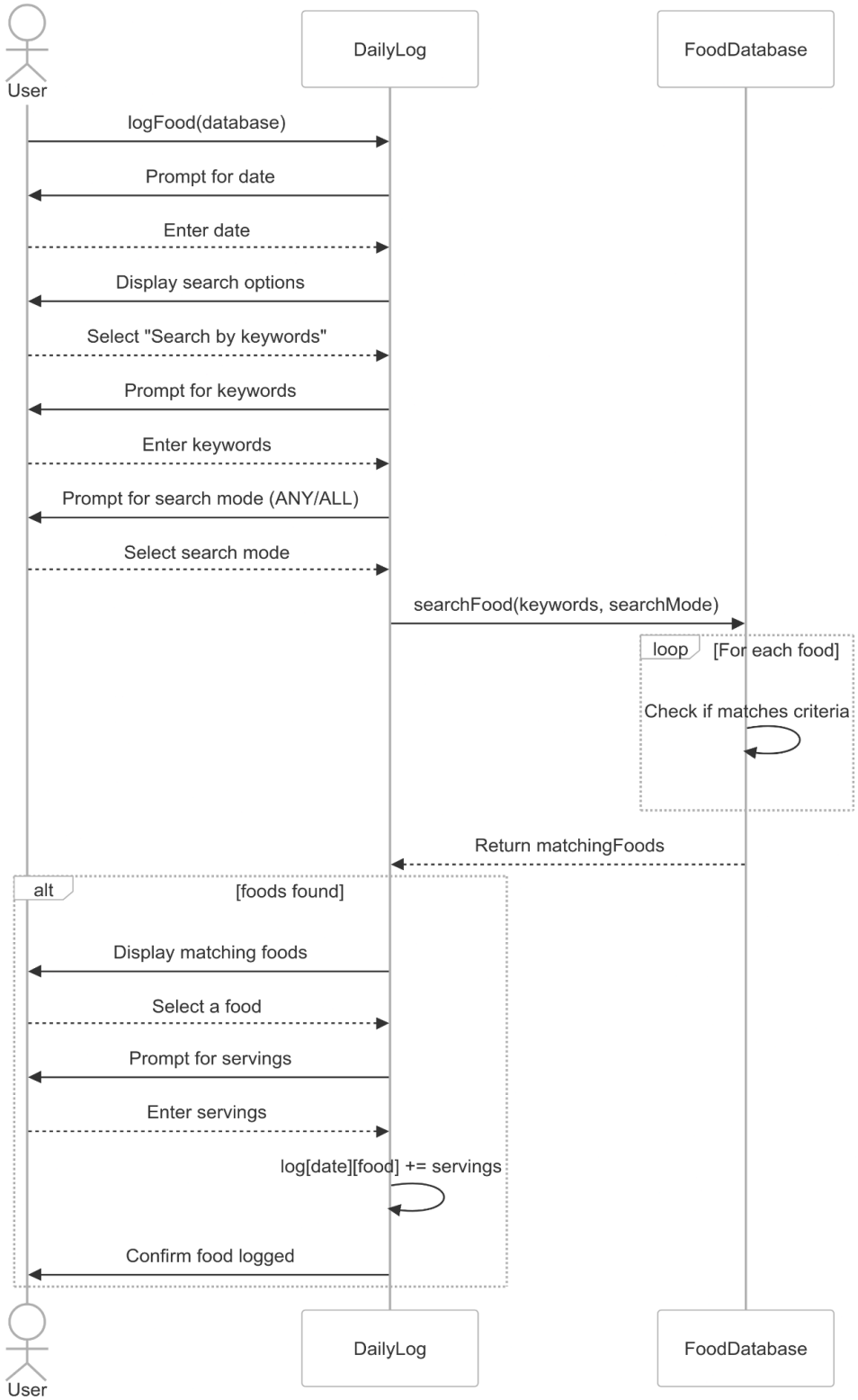
Sequence Diagrams

1. Adding a Composite Food



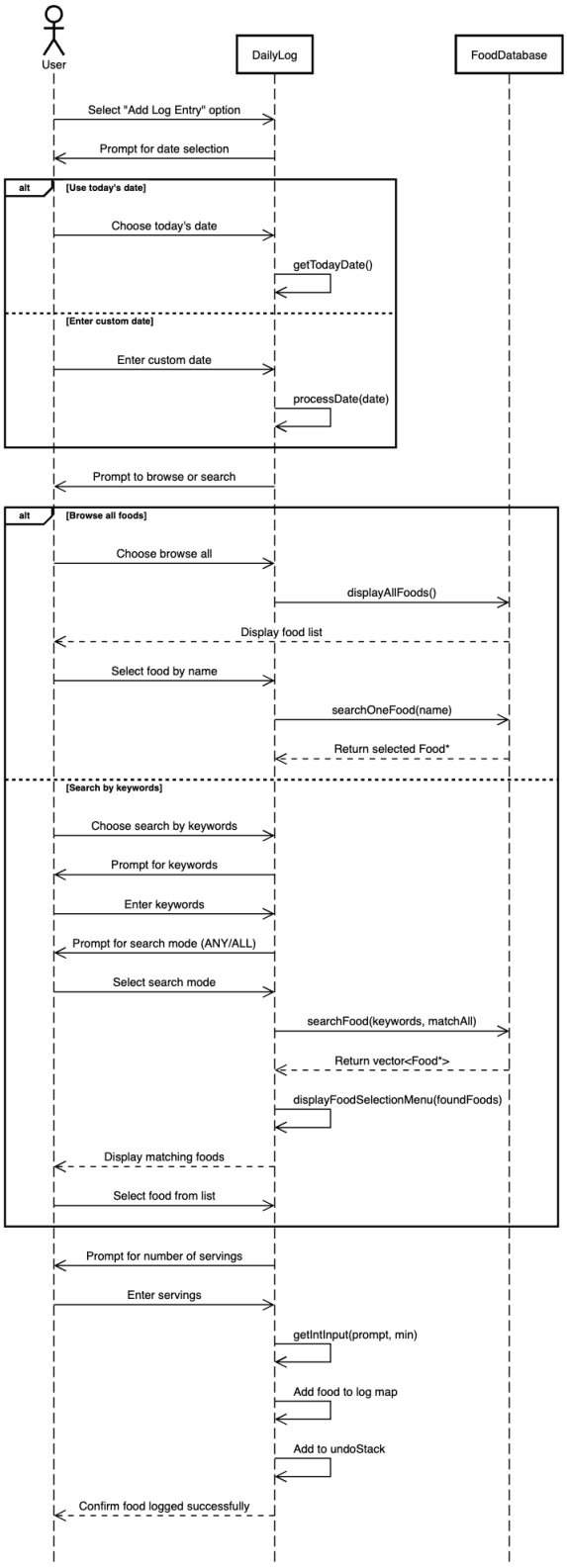
https://drive.google.com/file/d/1MCT1jUZRvpXGlLfNbqknLRFl6wKHXJpj/view?usp=sharing

2. Searching for a food by keyword



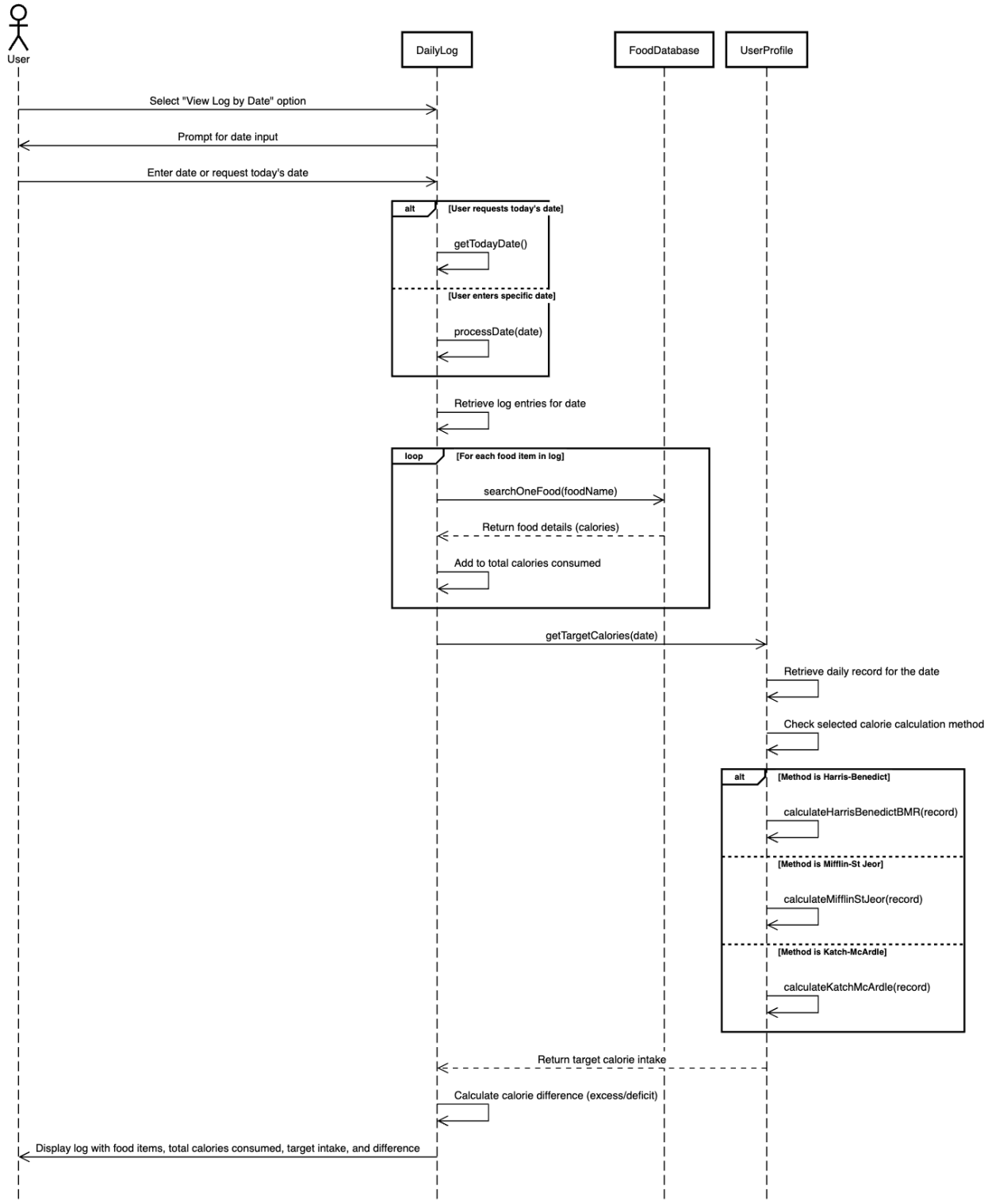
https://drive.google.com/file/d/1mfwentRyrqIQhLSLloKVpfNfjwMxOxkE/view?usp=sharing

3. Add Food to Log



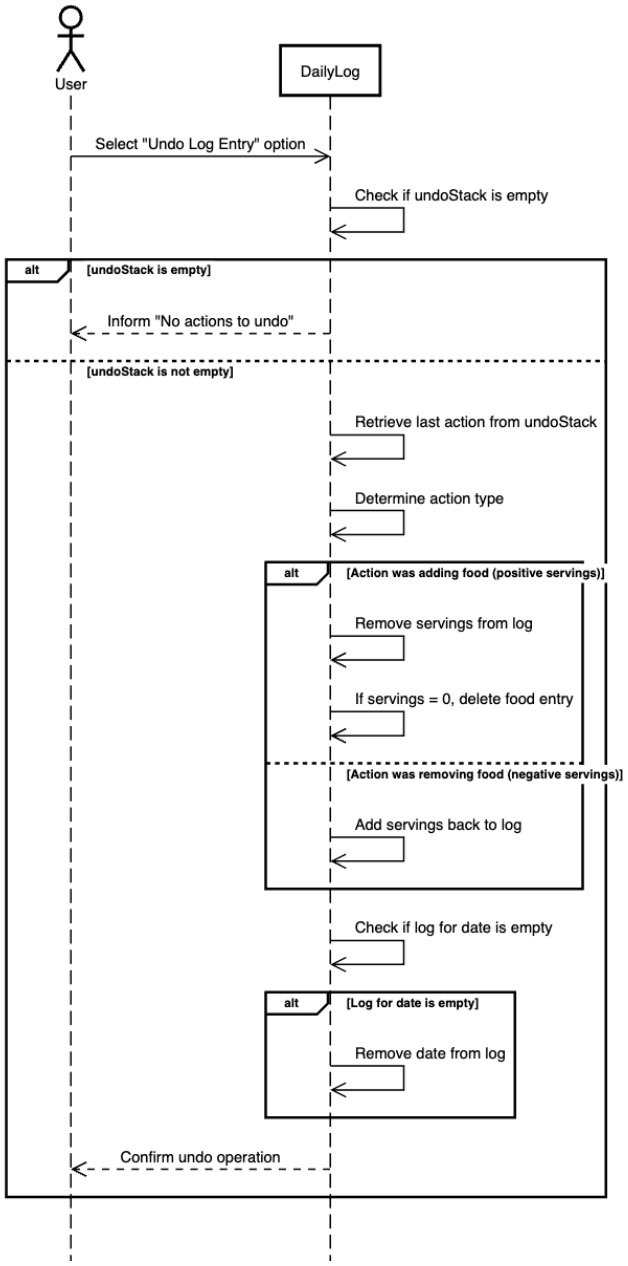
https://drive.google.com/file/d/1T2JBEVVGXTRypQivHFQzBaw3oMMq9G0q/view?usp=sharing

4. Get Log by Date



https://drive.google.com/file/d/1xba-169IRuEPp_hrytCuwZomg0ZjWe0R/view?usp=sharing

5. Undo Log Entry



https://drive.google.com/file/d/1u7J9yx5zzl67frREwt5dSPwh594iqvPc/view?usp=sharing

Design Narrative

The design ensures low coupling by clearly separating major components like Food, FoodDatabase, DailyLog, UserProfile. Utils.h has reusable utilities that are useful across files. We are passing any dependent values through parameters and not using global variables.

High cohesion and separation of concerns is ensured by having each class have its own distinct responsibility and not intersecting with other classes. **Food** manages individual food items. **FoodDatabase** handles food storage and retrieval. **DailyLog** handles managing records for the log file. **UserProfile** handles user data and calories. We have abstracted out common functions to the file **Utils.h** which handles things like input validation.

Information hiding is implemented by having private helper methods in classes like **DailyLog** and **UserProfile**. Using **Utils.h** also helps with this, by hiding away the implementation details of centralized utility functions.

The design mostly adheres to the Law of Demeter aka the principle of least knowledge. Methods largely interact with their own class's data, and don't expose their internal structure to other objects in most cases. The main function interacts with all of these via their public interfaces.

Reflection

Strengths of our Design

1. **Composite Pattern Implementation:** We have effectively implemented the Composite pattern for food items. The **CompositeFood** class allows creation of complex food items that comprise other foods. The calculations for calories are automatically adjusted to account for ingredient's servings. This makes our system able to effectively handle both simple and complex foods.
2. **Hierarchical Menu Structure:** There is a clear, hierarchical structure for the menu system which makes the system intuitive and easy to navigate. The menu is clearly divided into separate areas of different functions, namely logging, food management and profile management. We have also ensured input is constantly validated.

Weaknesses of our Design

1. **Mixing of UI and Logic**: In the DailyLog class, both data storage and user interaction functionalities are being mixed in some cases, which could be segregated out for better cohesion.
2. **Missing information hiding**: In the Food class, information hiding has not been implemented. Making some properties private and adding accessor methods can help improve information hiding.