PROJECT / RELEASE

Project Design Document

Royal Dabblers

Alex Leute <[acl2809@rit.edu](mailto:acl2809@rit.edu)>

Erich Snell <[eds5997@rit.edu](mailto:eds5997@rit.edu)>

Griffin Moose <[gsm1298@rit.edu](mailto:gsm1298@rit.edu)>

Julia Sarun <[jns2613@rit.edu](mailto:jns2613@rit.edu)>

Edward<[ess6446@rit.edu](mailto:ess6446@rit.edu)>

# Project Summary

This project is a wellness management program that lets users leave daily logs of what they are eating. The program will have some basic foods available, and those basic foods can also be combined to create recipes. Users can also create their own recipes and foods. These recipes and foods will store nutritional information, such as calories and carbs.

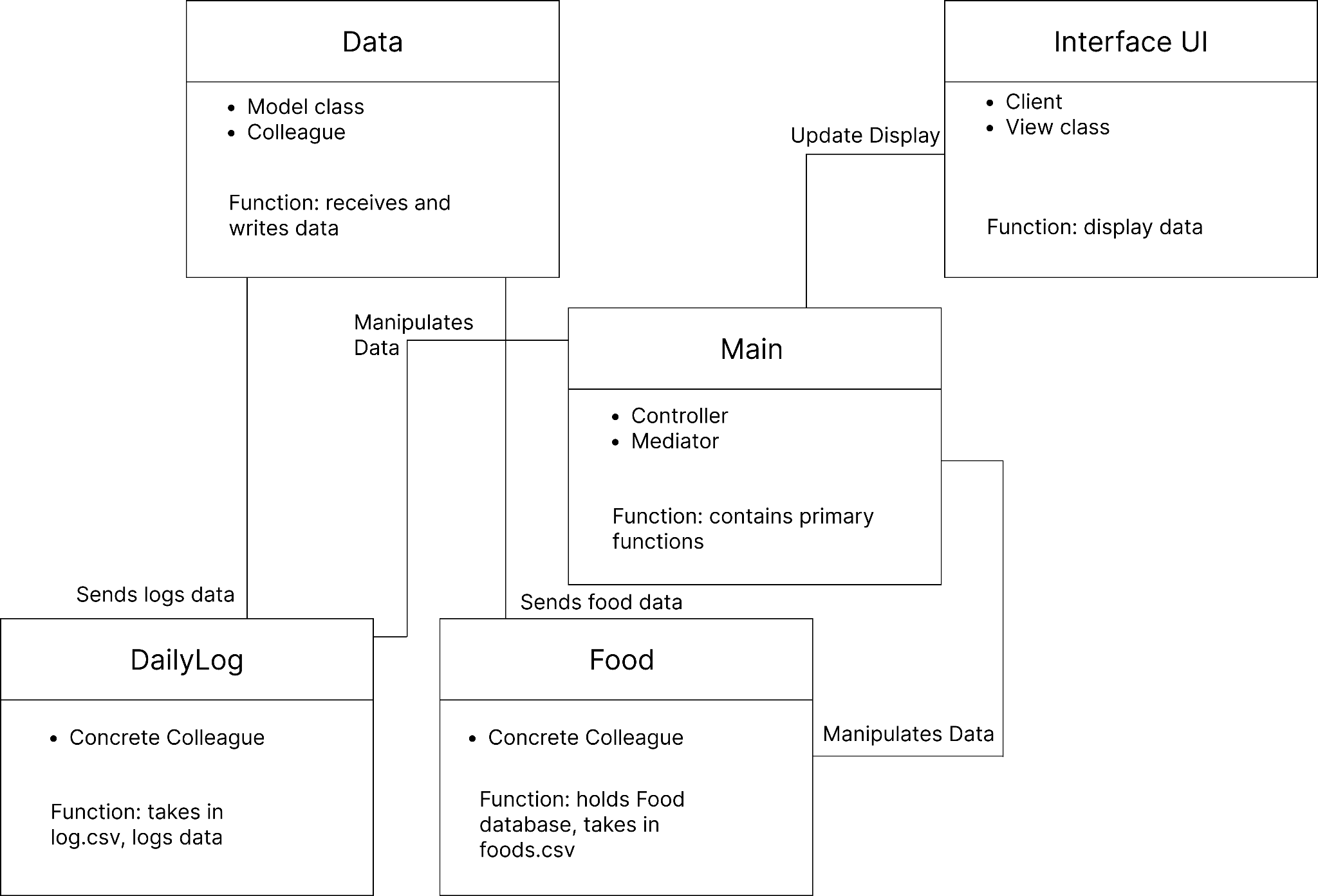
After users log what they are eating throughout the day, the software will help users track their intake and how well they are following their goals. It will also display nutrients per day, and graph distributions of what nutrients they are consuming daily. This will help users track their weight over time.

# Design Overview

Before we did any design sketches, subsystem diagrams, or UML, we sat down and combed through the assignment document and recorded all of the classes we believed we would need to complete the task. In brainstorming these classes, we consider that we wanted to limit coupling as much as possible while promoting high levels of cohesion. Furthermore, after completing the list of classes, we compiled another list of potential methods we would use. At this point, we began making rough sketches of a UML class diagram detailing the relationships between all of our different classes.

Using the rough sketches we could make subsystems based on many of the classes and further break them down into a more cohesive and extensible form. The UI subsystem has an interface that would allow for different UIs to be later added without changing any existing code. We did this under the assumption that a more user friendly UI would need to be implemented later. The Food subsystem uses the composite pattern to allow easy searching of both the recipes and basic foods. DailyLog has been left as is but may change later. Lastly we made data use an interface to allow for different data files to be used later with no changes to the code. We did this under the assumption that users may want different data files to be able to be used in saving and loading. The Data subsystem is also only linked to the DailLog and Food subsystems to reduce the coupling to Main and keeping unnecessary methods out of main increasing cohesion.

# Subsystem Structure

******

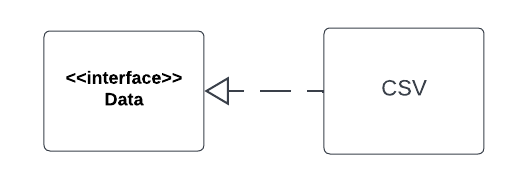
# Subsystems

## **Subsystem Data**

| **Class** Data (interface) | |
| --- | --- |
| **Responsibilities** | Provide a generic interface to all classes that interact with a data file.  Can receive a formatted data file (Load).  Can write a formatted data file (Save). |

## 

| **Class** CSV | |
| --- | --- |
| **Responsibilities** | Can receive a formatted CSV file (Load).  Can write a formatted CSV file (Save). |
| **Collaborators (inheritance)** | Data |



## **Subsystem Food**

| **Class** FoodCollection | |
| --- | --- |
| **Responsibilities** | Support access to the foods or recipes in the food collection.  Add, find, delete an existing food or recipe in the collection.  Save/load to and from a data file |
| **Collaborators**  **(uses)** | Food- the basic type for all different foods or recipes in the collection.  [Data](#_heading=h.2et92p0)- the basic methods needed to interact with a data file |

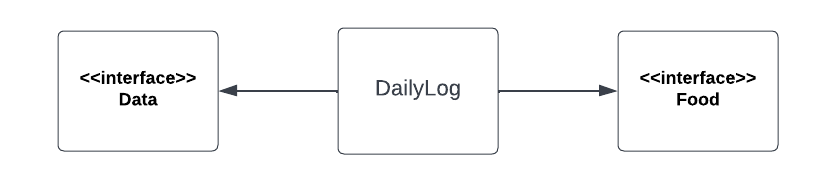
| **Class** Food (interface) | |
| --- | --- |
| **Responsibilities** | Provide a generic interface to all food and recipe types.  Includes unique name of food or recipe. |

| **Class** BasicFood | |
| --- | --- |
| **Responsibilities** | Represents a basic food in the collection.  Provides access to calories, fat, carb, and protein in a serving of the food |
| **Collaborators (inheritance)** | Food |
| **Class** Recipe | |
| **Responsibilities** | Represents a recipe in the collection.  Provides access to all basic foods and their servings in the recipe  Provides access to the calories, fat, carb, and protein in terms of the foods stated serving amounts in the recipe. |
| **Collaborators (inheritance)** | Food |

## 

## **Subsystem DailyLog**

| **Class** DailyLog | |
| --- | --- |
| **Responsibilities** | Represents a daily log  Provides access to calorie limit, foods eaten, and servings consumed on a date.  Save/load to and from a data file |
| **Collaborators (uses)** | [Data](#_heading=h.2et92p0)  [Food](#_heading=h.wdsjzgewah7g) |



## **Subsystem UI**

| **Class** UI (interface) | |
| --- | --- |
| **Responsibilities** | Provide a generic interface to all UI types.  Allows users to add, find, delete an existing food or recipe in the collection using the UI.  Save/load to and from a data file using the UI. |

| **Class** CLI | |
| --- | --- |
| **Responsibilities** | Represents a basic CLI UI.  Allows users to add, find, delete an existing food or recipe in the collection using commands.  Save/load to and from a data file using commands. |
| **Collaborators (inheritance)** | UI |

## 

## **Subsystem Main**

| **Class** Main | |
| --- | --- |
| **Responsibilities** | Represents the main controller.  Allows UI to interact with other classes.  Use Food class functions  Use DailyLog class functions |
| **Collaborators (uses)** | [Food](https://docs.google.com/document/d/1_vzMzLyPqrMy6d1huPeKitsN_d5xXSlY/edit#heading=h.wdsjzgewah7g)  [DailyLog](https://docs.google.com/document/d/1_vzMzLyPqrMy6d1huPeKitsN_d5xXSlY/edit#heading=h.lt6lbdwb1st) |

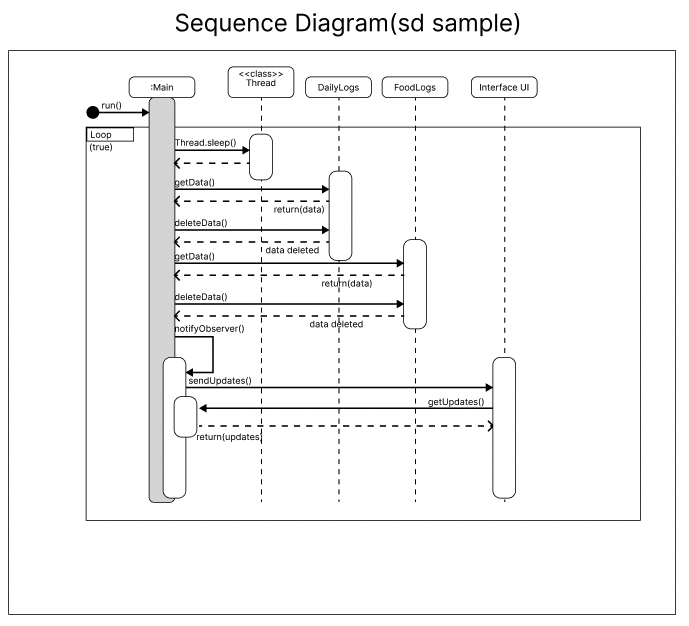
## 

# Sequence Diagrams

This sequence diagram (The initializer) details the creation of the observer, the observable, and all of the classes. Furthermore, it also illustrates the creation of the thread which will start the actual sequence of classes and methods.

# 

This next sequence diagram (the sample) shows the typical sequence of the classes. when the program is ran, the main class receives data from the CSV interface, and also has the ability to delete data. After receiving the data, it sends updates to the Interface UI so that the user can see the data displayed in a text UI format.



# Pattern Usage

## **Composite Pattern**

| **Composite Pattern** | |
| --- | --- |
| **Component** | Food |
| **Leaf** | BasicFood |
| **Composite** | Recipe |

## **MVC Pattern**

| **MVC Pattern** | |
| --- | --- |
| **Model** | Food  DailyLog  Data |
| **View** | UI |
| **Controller** | Main |

## 

**RATIONALE**

We decided to use the MVC pattern to decrease coupling, and to separate the concerns from the UI and the Model.

We are using the composite pattern for Food because Recipes are made up of BasicFoods, but have all of the same properties as BasicFoods