PROJECT / RELEASE

Project Design Document

Team A

Brianna Buttaccio <bab7607@rit.edu>
Caitlyn Daly <cnd9351@rit.edu>
Tiffany Ellis <tae7612@rit.edu>
Roy Tran <rxt7649@rit.edu>
Kevin Reynolds <kmr1188@rit.edu>
Fu Quan Li <fxl2328@rit.edu>

Project Summary

The project's goal is to create an application that allows a user to keep track of personal dietary information. This is important to assist those whom are concerned with calorie intake, whether they are trying to maintain a certain calorie count to gain weight, or stay under a calorie count to lose weight. This application will be able to allow users to add basic foods and recipes in order to accurately keep track of all nutritional information. Basic foods consist of single food items such a banana or an apple that may be consumed by itself or used as an ingredient for a combination of foods. This combination of foods is called a recipe, which have many different kinds of basic foods and recipes used together. The user can add basic foods and recipes to the food log in order to reuse and add the food items to their food consumption log at a later time. The food log also tracks the calories, number of grams of fat, carbohydrates, and proteins per serving for each food item.

In addition to tracking, recording, and viewing nutritional information from foods, users will able be able to record and view personal information such as daily logs of what they consumed, their calorie goal, and their weight. Users will be able to record their daily weight in order to view their progress. Daily calorie limits can be chosen by the user in order to assist them with accidental overconsumption of calories. Every log recorded will be stored for a specific day, so the user can view food consumption, weight, and calorie intake on a given day.

In order for the user to be able to interact with the application, a graphical interface will display the data to the user. This interface will display the data to the user to view data in list and graphical views. This will be the most user friendly way for the application to interact with any user. The graphical interface will be separated through tabs, allowing the user to access any part of the application in order to enter or view that information in an easy to read way.

Design Overview

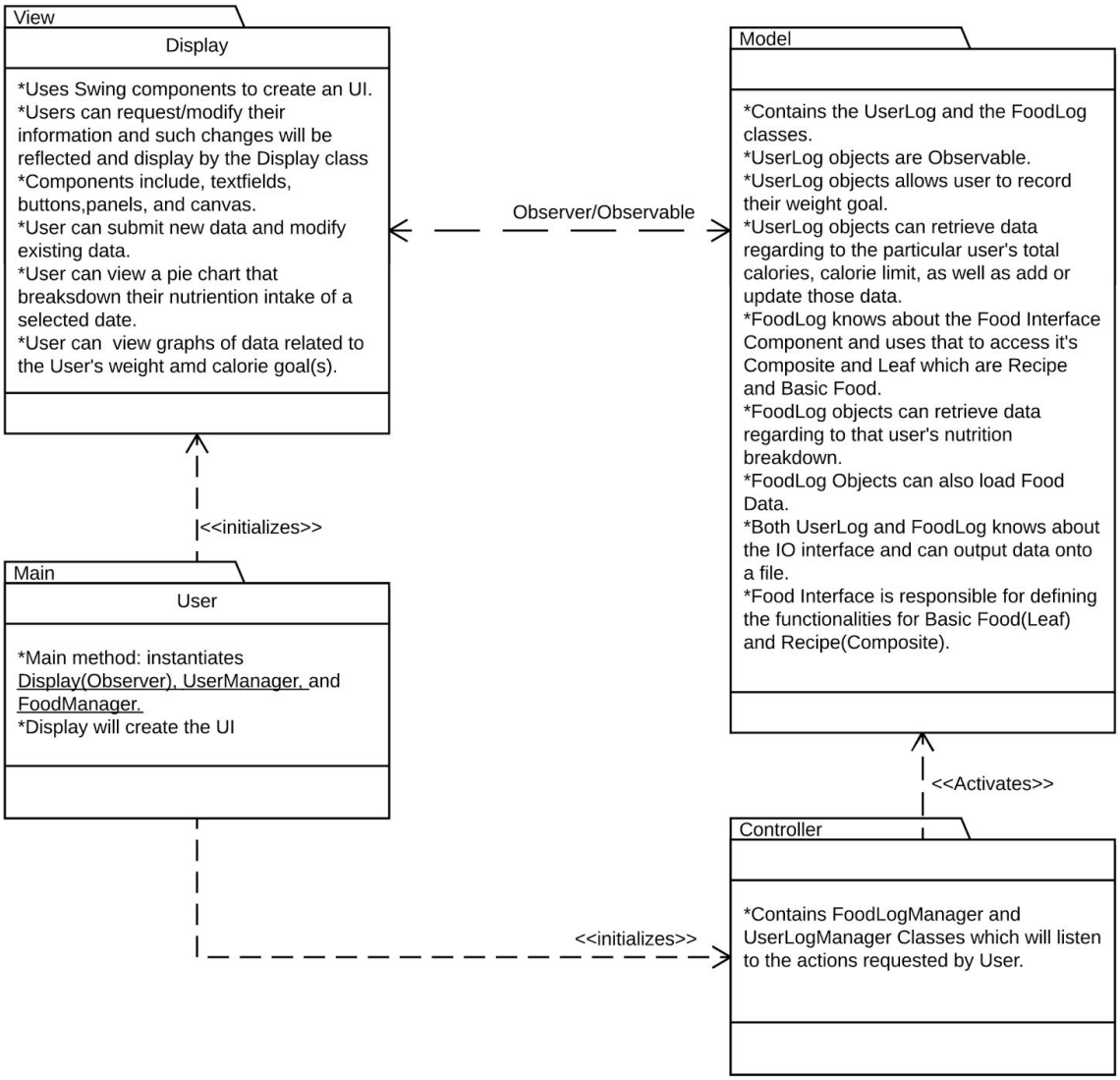
For the design of our project we decided to have the user be the main class. Our project follows the MVC pattern, so the user can see and knows about the view and controller classes which are the Display, User Log Manager, and Food Manager. The user just initializes the classes. The Display will function as our view. It will know about and be able to call on other classes that will act as widgets that can be reused in different parts of the program. These widgets will include a line graph, food consumption list, food and recipe addition and deletion, and a delete button. We wanted to keep all of these widgets separate so that we have good separation of concerns, high cohesion and low coupling. We will also be able to re-use specific widgets on whatever page we need them, and they will change depending on our needs. All of the widgets that include changing data such as food lists and graphs act as observers. All widgets that trigger an action will trigger functions in the controllers.

The User Log Manager and Food Log Manager are classes that act as our controllers. They will handle all of the logical functionality that comes with the user's nutritional, daily log data and food and recipe data respectively. The controller will use ActionListeners to run functions when the user clicks on buttons and enter data in the view. Some of the classes that the

controller will call upon are the food log, user log, and our io interface. The io interface will allow the food log and user log to have access to our csv io which is in charge of reading and writing information to our csv file to store data. The reason that we decided to have an io interface and a csv io class is so that is the datatype (csv file) ever changes a new class can be written to parse that data specifically without having to change the rest of the files. Originally we had the io class writing and reading to the csv file directly, however by separating the classes we were able to create better abstraction in our program.

User Log, Food Log, Food, Basic Food, Recipe, CSVIO, IO Interface are our models. Each class is in charge of recording their respective data. The model uses the Observable pattern so that the view can update the display when data changes in the model. All of these files are use the controller to handle the logic and business rules of changing data. The model can see the IO interface and the CSV files so they can store the data permanently in files. Food is also part of the model. Food represents a basic food or recipe object so that the user can create a food once and reuse it. Food acts as our component in the composite pattern. The component is the Recipe and the basic food is the leaf.

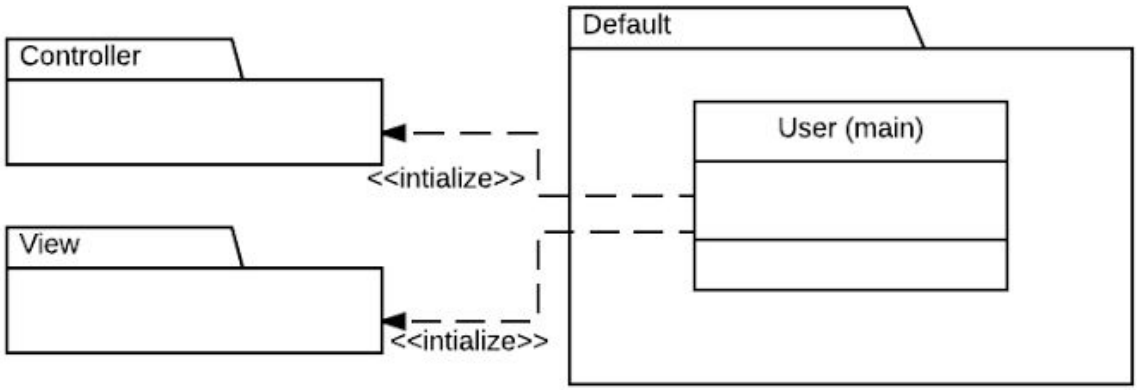
Subsystem Structure


View
Display
*Uses Swing components to create an UI.
*Users can request/modify their information and such changes will be reflected and display by the Display class
*Components include, textfields, buttons,panels, and canvas.
*User can submit new data and modify existing data.
*User can view a pie chart that breaksdown their nutriention intake of a selected date.
*User can view graphs of data related to the User's weight amd calorie goal(s).
Observer/Observable
Model
*Contains the UserLog and the FoodLog classes.
*UserLog objects are Observable.
*UserLog objects allows user to record their weight goal.
*UserLog objects can retrieve data regarding to the particular user's total calories, calorie limit, as well as add or update those data.
*FoodLog knows about the Food Interface Component and uses that to access it's Composite and Leaf which are Recipe and Basic Food.
*FoodLog objects can retrieve data regarding to that user's nutrition breakdown.
*FoodLog Objects can also load Food Data.
*Both UserLog and FoodLog knows about the IO interface and can output data onto a file.
*Food Interface is responsible for defining the functionalities for Basic Food(Leaf) and Recipe(Composite).
<<initializes>>
Main
User
*Main method: instantiates Display(Observer), UserManager, and FoodManager.
*Display will create the UI
<<Activates>>
Controller
*Contains FoodLogManager and UserLogManager Classes which will listen to the actions requested by User.
<<initializes>>

Subsystems

Default

User	
Responsibilities	Initializes Food Log Manager and User Log Manager classes in the Controller subsystem. Initializes Display class in the View subsystem.
Collaborators (uses)	controller.FoodLogManager controller.UserLogManager view.Display



Model

Food Log	
Responsibilities	Interfaces with IO interface to read and save Food. Observed by the display.
Collaborators (uses)	Model.IOInterface View.Display

User Log	
Responsibilities	Interfaces with IO interface to read and save user food stats. Observed by the display.
Collaborators (uses)	Model.IOInterface View.Display

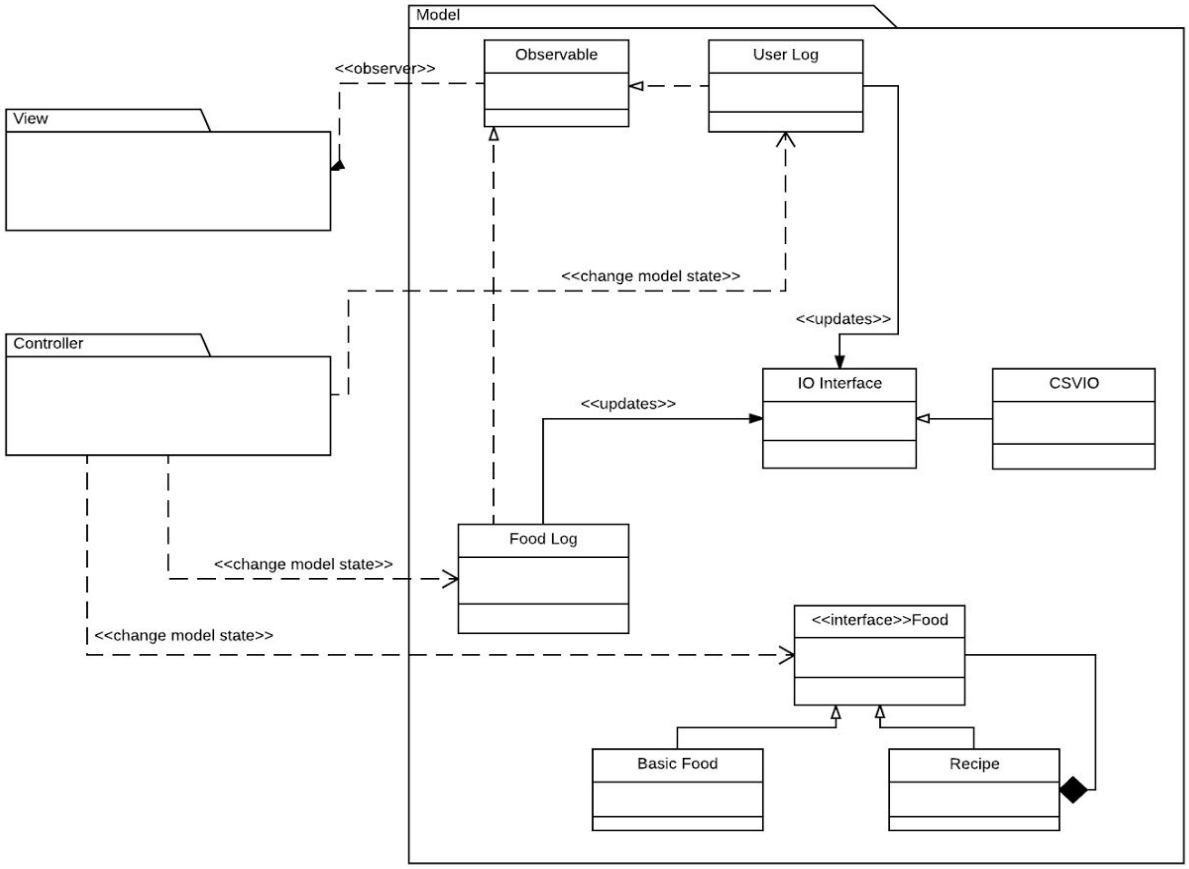
IO Interface	
Responsibilities	Recieves data to load or save from either User or Food Log classes.
Collaborators (uses)	Model.UserLog Model.FoodLog Model.CSVIO

CSVIO	
Responsibilities	Saves and loads data from the csv log files.
Collaborators (uses)	Model.IOInterface

Food	
Responsibilities	Keeps track of the different foods (basic foods and recipes). Recieves data from Food Log Manager.
Collaborators (uses)	Controller.FoodLogManager

Basic Food	
Responsibilities	Holds a basic food object
Collaborators (uses)	Model.Food

Recipe	
Responsibilities	Holds a recipe object created from basic food objects
Collaborators (uses)	Model.Food



View

Create Recipe Text View Panel	
Responsibilities	Creates a panel for displaying recipes
Collaborators (uses)	view.submitRecipeJButton view.addNewFood/RecipeJFrame

Submit Recipe JButton	
Responsibilities	Submits new recipes to the food log
Collaborators (uses)	view.CreateRecipeTextViewPanel

Add New Food/Recipe JFrame	
Responsibilities	Creates a JFrame for recipes or basic food
Collaborators (uses)	view.createRecipeTextViewPanel view.createBasicFoodViewPanel

Create Recipe Text View Panel	
Responsibilities	Creates a panel for displaying basic foods
Collaborators (uses)	view.submitBasicFoodJButton view.addNewFood/RecipeJFrame

Submit BasicFood JButton	
Responsibilities	Submits new recipes to the food log
Collaborators (uses)	view.CreateBasicFoodTextViewPanel

Foods Consumed Text View Panel	
Responsibilities	Creates a panel to display which foods have been consumed
Collaborators (uses)	view.dashboardJFrame

Calorie Intake and Calorie Limit Text View Panel	
Responsibilities	Creates a panel to display calorie intake and limits
Collaborators (uses)	view.dashboardJFrame

Over/Under Calorie Limit Text View Panel	
Responsibilities	Displays whether or not the user is over or under their calorie limit
Collaborators (uses)	view.dashboardJFrame

Dashboard JFrame	
Responsibilities	Holds all of the calorie and food panels
Collaborators (uses)	view.foodsConsumed view.calorieIntake view.overUnderCalorie view.JDatePicker view.CalorieLimit

JDatePicker	
Responsibilities	Allows the user to pick a date
Collaborators (uses)	view.dashboardJFrame

Calorie Limit Text Field Panel	
Responsibilities	Accepts user input for daily calorie limit
Collaborators (uses)	view.dashboardJFrame view.submitCalorieLimit

Submit Calorie Limit JButton	
Responsibilities	Button for setting the user calorie limit
Collaborators (uses)	view.CalorieLimit

Weight GraphCanvas	
Responsibilities	Graph of weight over time
Collaborators (uses)	view.weightJFrame

Weight JFrame	
Responsibilities	A JFrame to hold the weight items
Collaborators (uses)	view.weightGraphCanvas view.WeightTextFieldPanel

Weight Text Field Panel	
Responsibilities	Accepts the user's current weight
Collaborators (uses)	view.weightJFrame view.submitWeightJButton

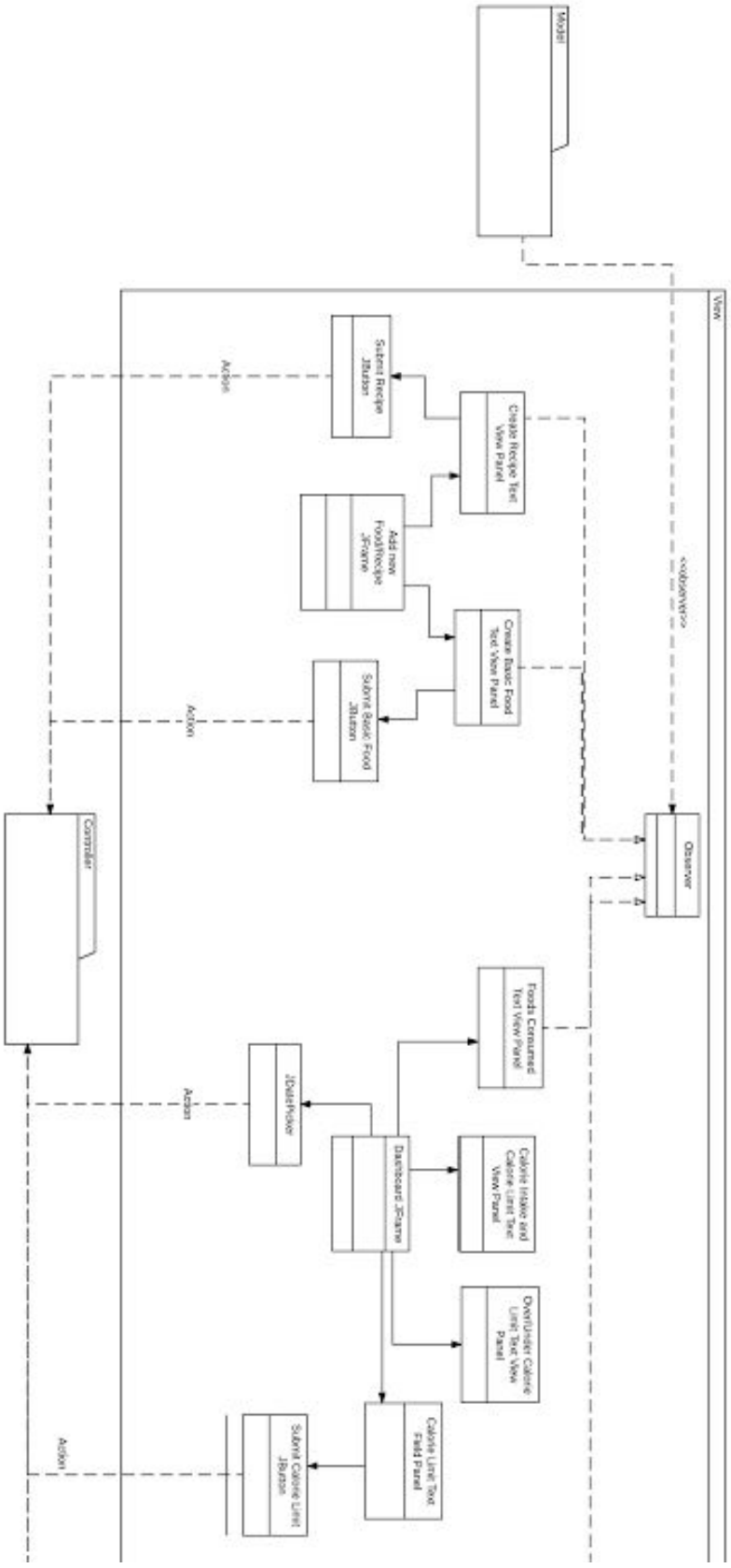
Submit Weight JButton	
Responsibilities	Submits the current weight in the weight text field
Collaborators (uses)	view.WeightTextFieldPanel

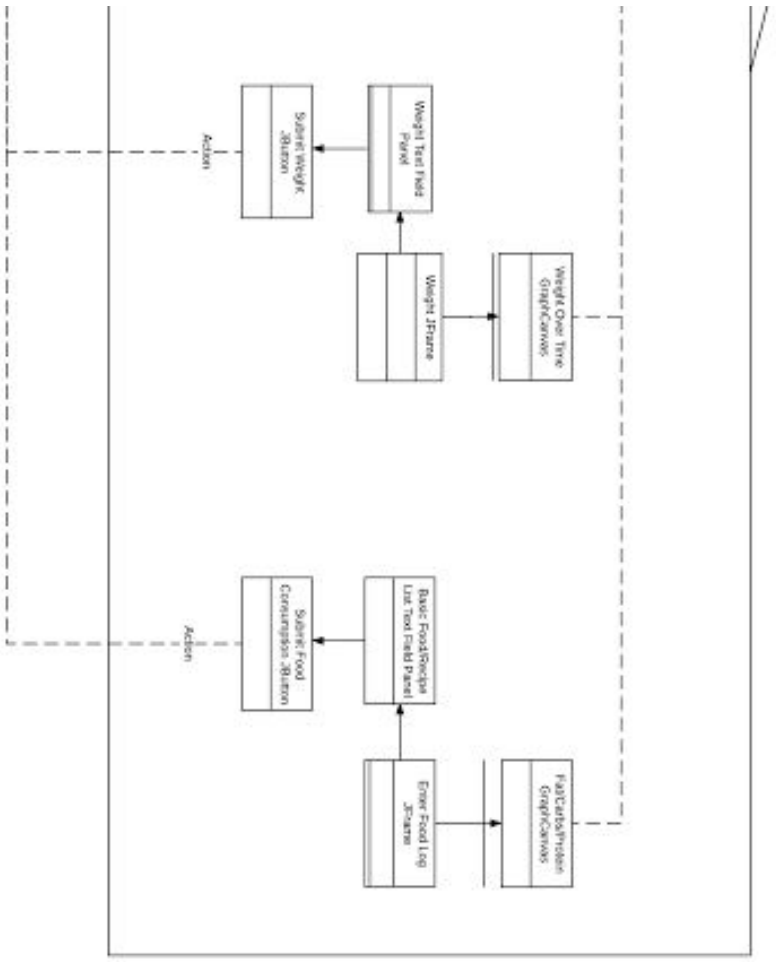
Fat/Carbs/Protein GraphCanvas	
Responsibilities	Graph of the percent fat carbs and protein for the day
Collaborators (uses)	view.foodLogJFrame

FoodLog JFrame	
Responsibilities	A JFrame to hold the food items
Collaborators (uses)	view.FatCarbsProteinGraphCanvas view.FoodRecipeTextFieldPanel

Basic Food/Recipe Text Field Panel	
Responsibilities	Accepts a basic food or recipe to be added to user log
Collaborators (uses)	view.foodLogJFrame view.submitFoodJButton

Submit Food JButton	
Responsibilities	Submits the current food/recipe in the food/recipe text field
Collaborators (uses)	view.FoodRecipeTextFieldPanel

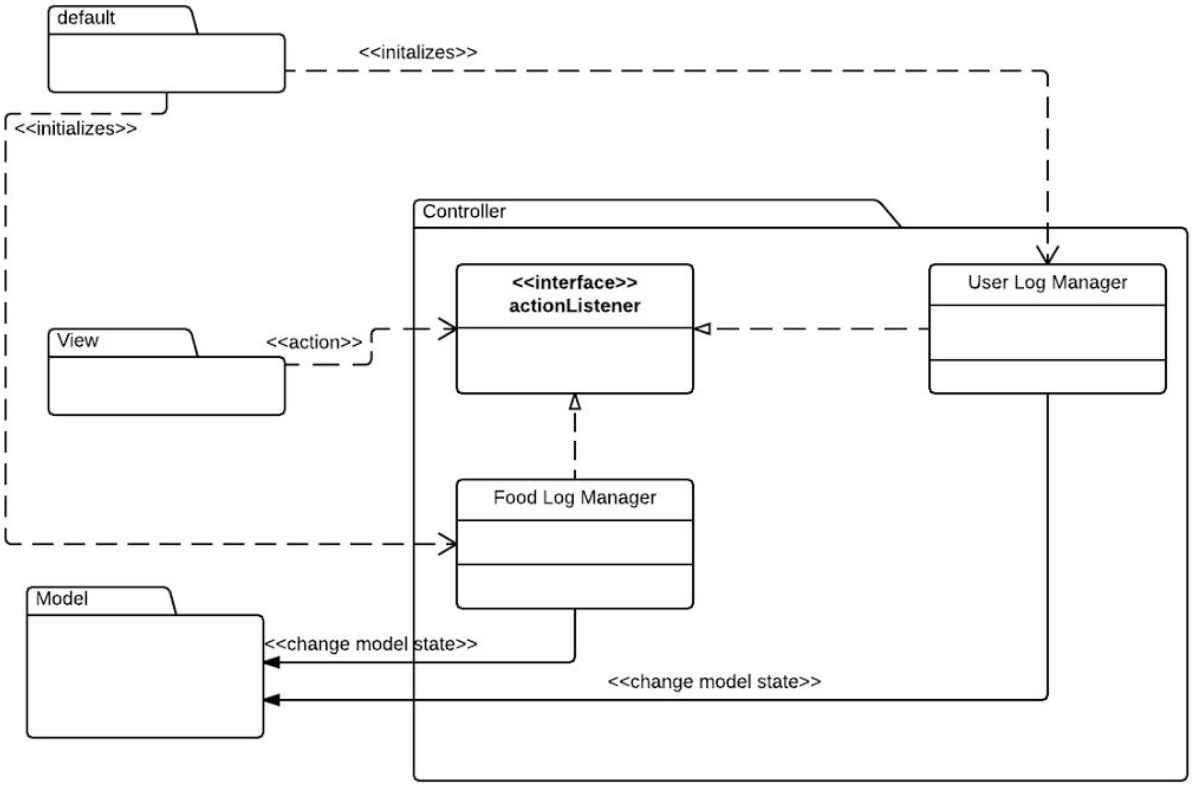

Model
View
<<observes>>
Submit Recipe JButton
Create Recipe Text View Panel
Add new FoodRecipe JFrame
Create Basic Food Text View Panel
Submit Basic Food JButton
Action
Action
Action
Observer
Controller
Foods Consumed Text View Panel
JDatePicker
Action
Dashboard JFrame
Calorie Intake and Calorie Limit Text View Panel
Overall/Set Calorie Limit Text View Panel
Calorie Limit Text Field Panel
Submit Calorie Limit JButton
Action


Action
Submit Weight JButton
Weight Text Field Panel
Weight JFrame
Weight Over Time GraphCanvas
Action
Submit Food Consumption JButton
Basic Food/Recipe List Text Field Panel
Enter Food Log JFrame
Fat/Carbs/Protein GraphCanvas

Controller

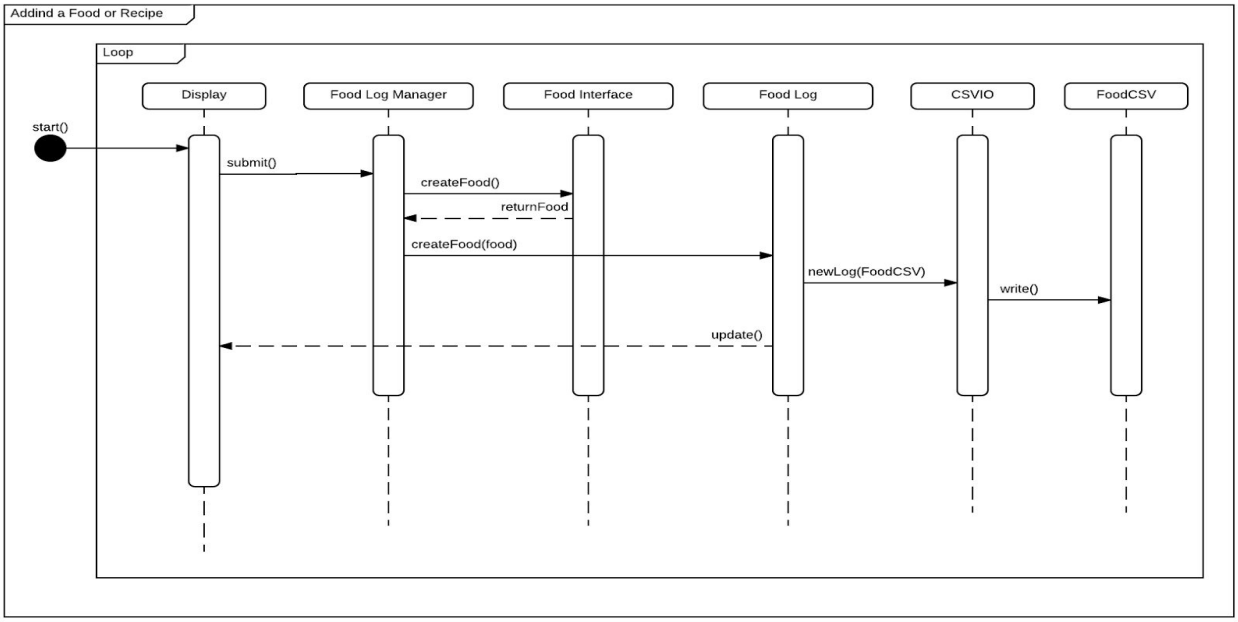
User Log Manager	
Responsibilities	Initializes and updates User Log class from user input in the Display class
Collaborators (uses)	model.UserLog actionListener

Food Log Manager	
Responsibilities	Initializes and updates Food Log class from user input in the Display class. Initializes and creates/updates Food components.
Collaborators (uses)	model.FoodLog model.Food actionListener

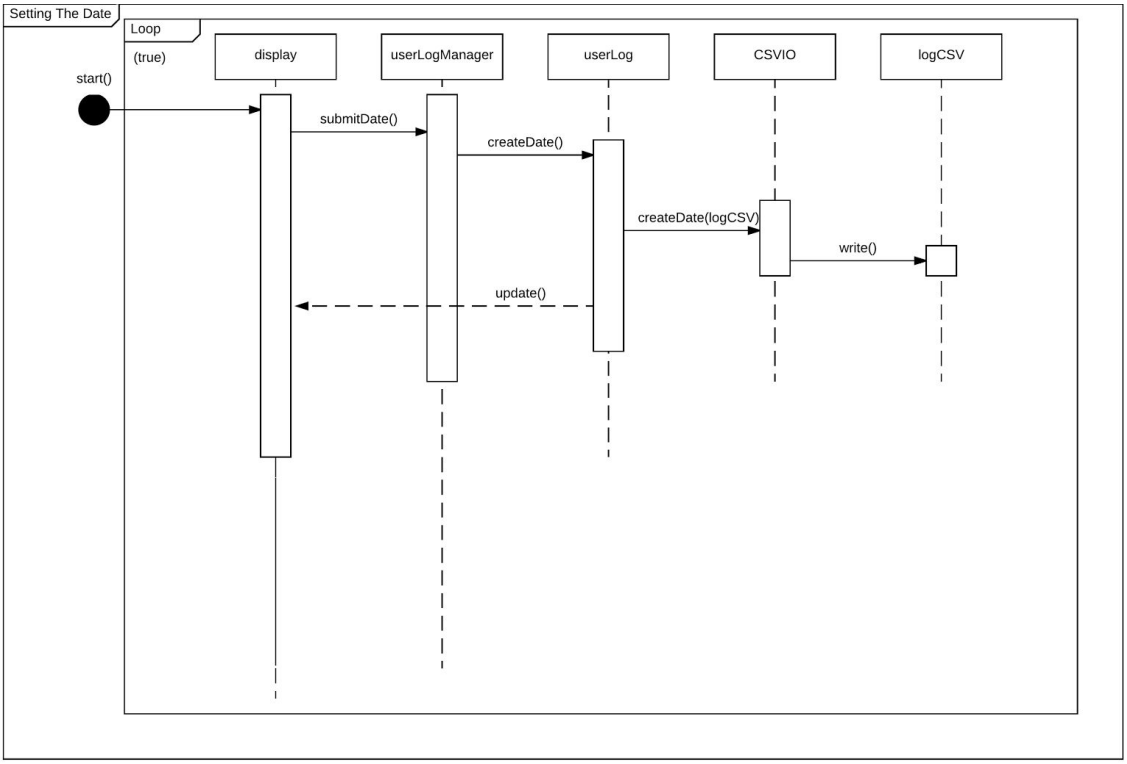

default
<<initalizes>>
<<initializes>>
Controller
<<interface>>
actionListener
User Log Manager
View
<<action>>
Food Log Manager
Model
<<change model state>>
<<change model state>>

Sequence Diagrams

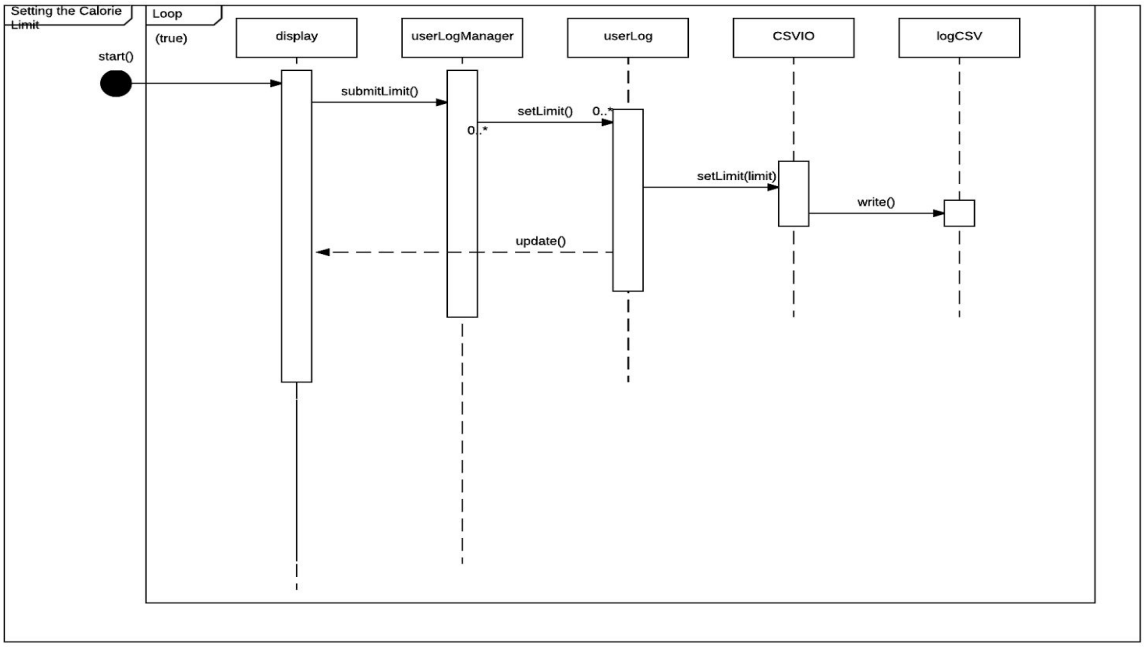
Adding a new basic food or recipe to the food log



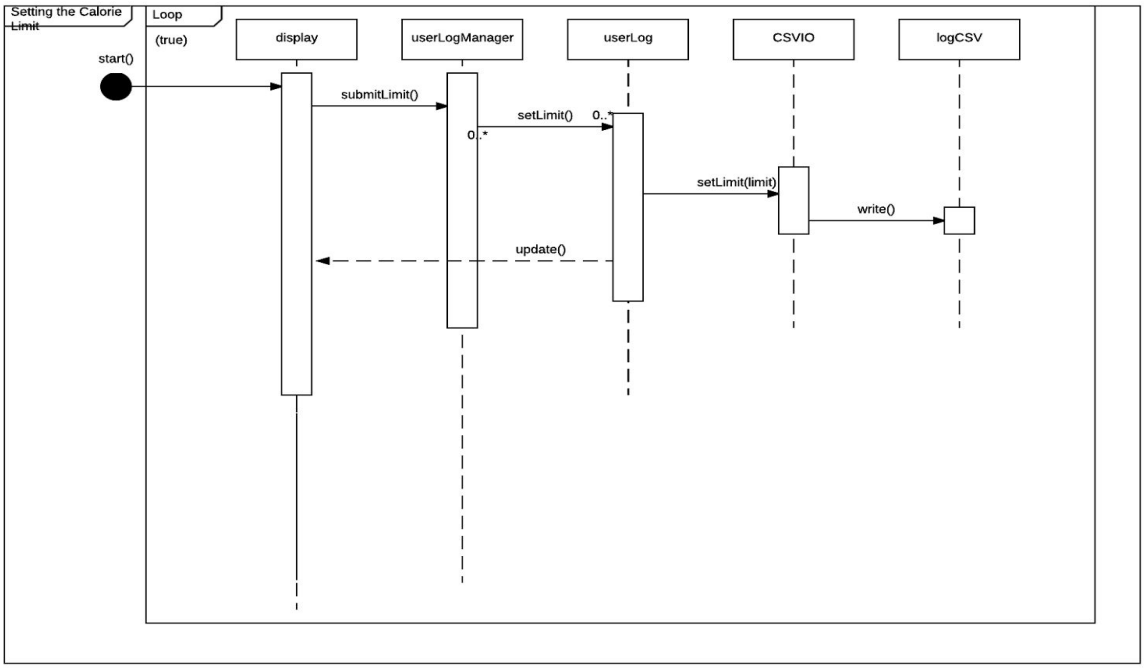
Setting the date



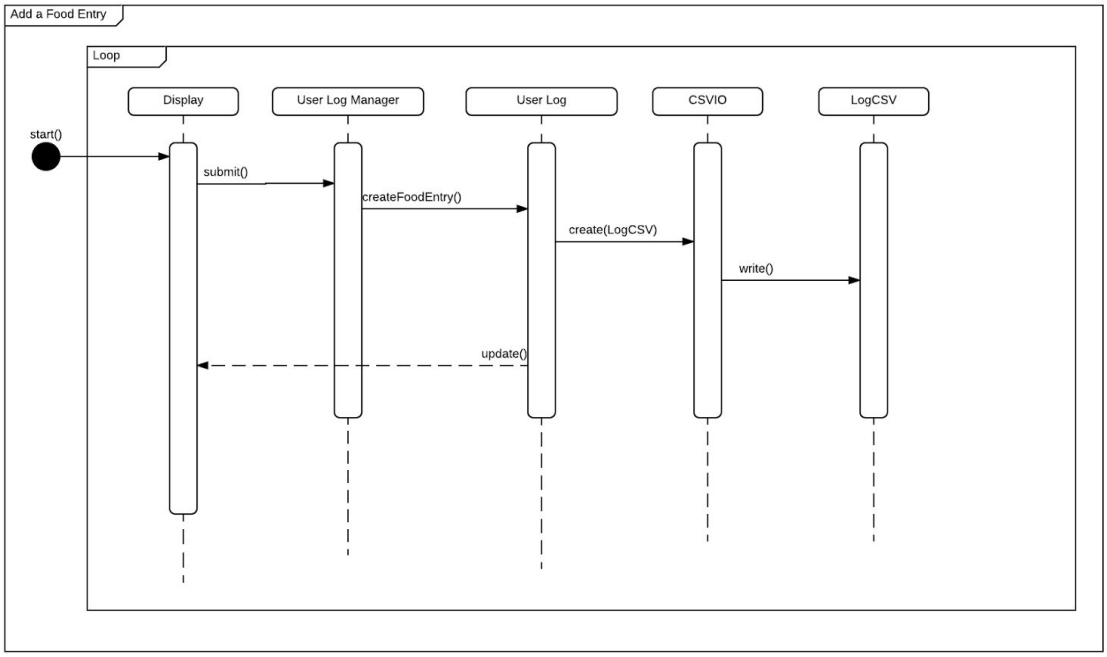
Setting the calorie limit



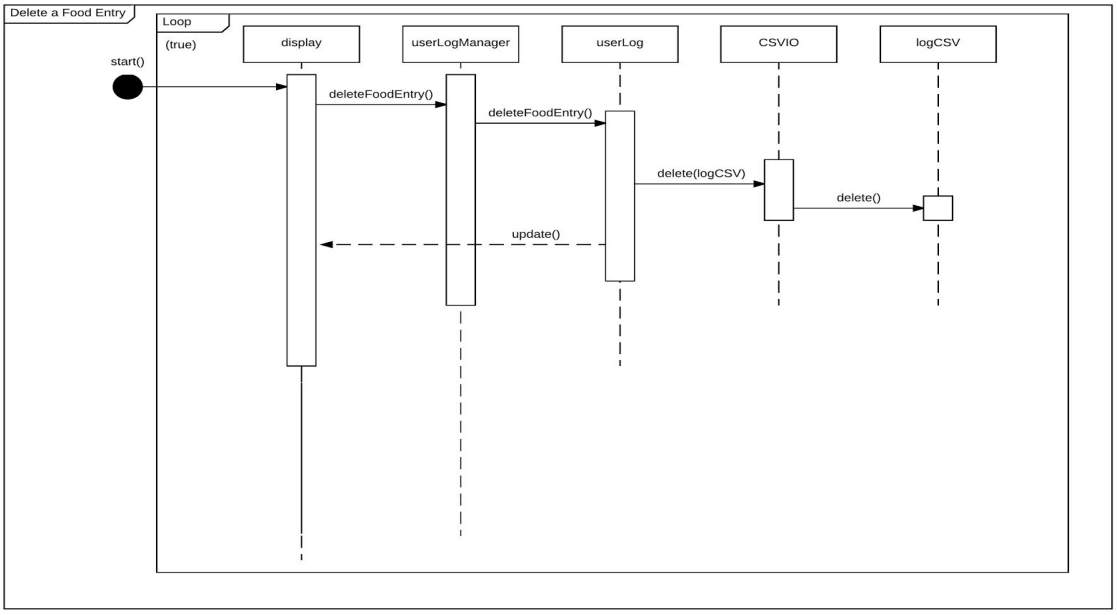
Setting the weight



Add food intake to user log



Deleting food entry from the user log



Pattern Usage

Observer Pattern

Observer Pattern	
Observer(s)	Display
Observable(s)	User Log, Food Log

Composite Pattern

Composite Pattern	
Component	Food
Leaf	Basic Food
Composite	Recipe

MVC Pattern

MVC Pattern	
Model	User Log, Food Log, Food, Basic Food, Recipe, CSVIO, IO Interface
View	Display
Controller	User Log Manager, Food Log Manager