PROJECT / RELEASE 2

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 health information. This information is important to users who wish to maintain a healthy lifestyle. Users will able to add basic foods and recipes in order to accurately keep track of all nutritional information such as calorie intake, which can be influential to maintaining a healthy lifestyle. 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.

Exercising is very important to maintain especially when trying to control or keep track of one's health. Different kinds of exercises can be added by the user. Once added the user can select from a list of exercises to be included in their daily log. The user will be able to see various interactions between calories for exercising. Each exercise has a value of calories that is expended doing the exercise .This is be displayed along with the the net calories that is calculated by subtracting calories consumed by calories expended. The difference of calorie goal and net calorie goal will then be displayed. This is all useful information to users who are trying to keep a strict calorie limit.

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

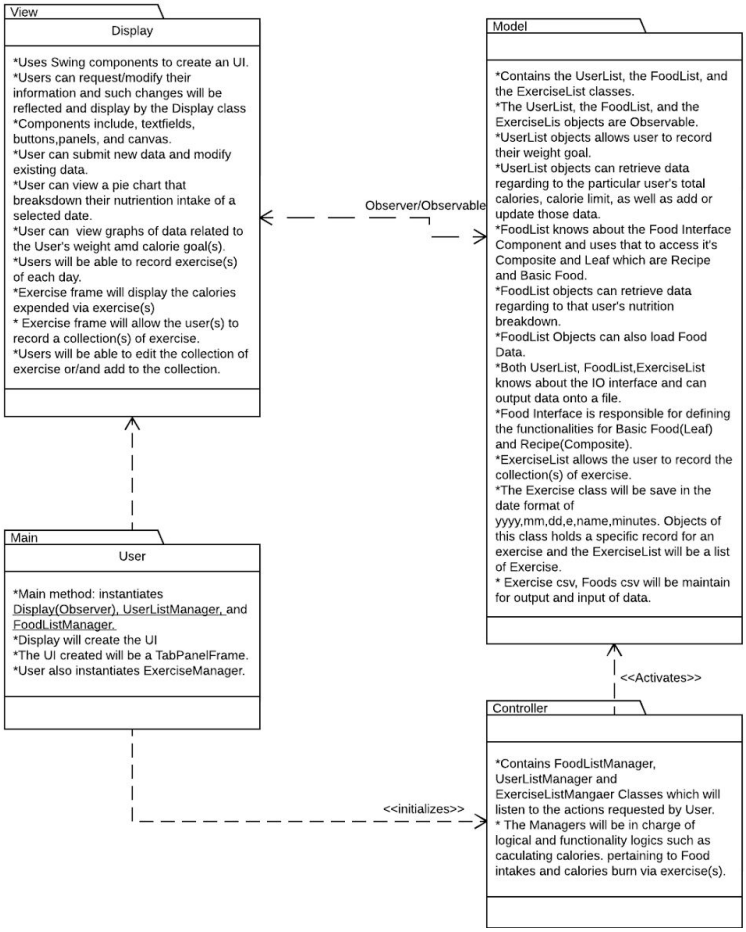
For the design of our project we decided to have the user be the main class. Our project follows the MVC pattern, so the user can see and knows about the view and controller classes which are the Display, User Log Manager, ExerciseListManager, and FoodListManager. The user just initializes the classes. 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 UserLogManager, ExerciseListManager, and FoodListManager 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.

UserLog, FoodList, ExerciseList, 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

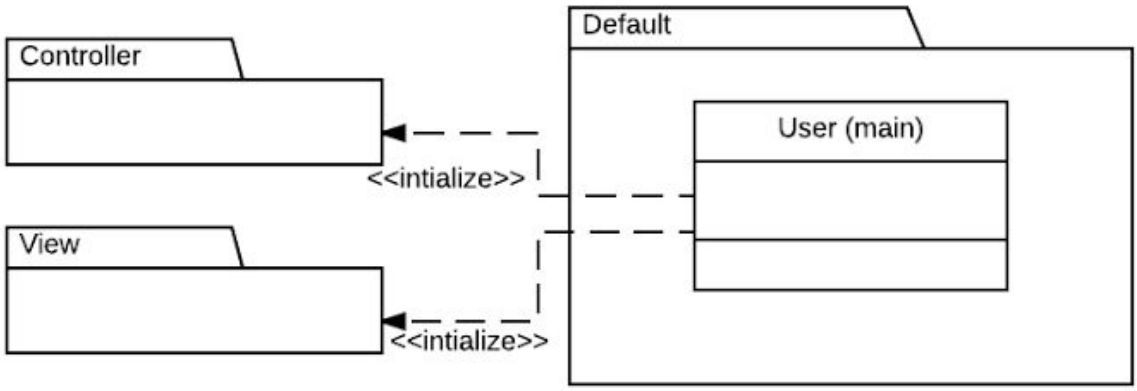


Subsystems

Default

User	
Responsibilities	Initializes FoodList Manager, User Log Manager,ExerciseManager classes in the Controller subsystem. Initializes Display class in the View subsystem.
Collaborators (uses)	controller.FoodListManager controller.UserLogManager

	controller.ExerciseManager view.DisplayAll



Model

FoodList	
Responsibilities	Interfaces with IO interface to read and save Food. As well as read and save Exercise. 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

Daily Log

Responsibilities	Logs the information of the users for a specific day.
Collaborators (uses)	Model.Food View.Display

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

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

Exercise	
Responsibilities	An entry for a specific instance of exercise recorded in a specific date format.
Collaborators (uses)	Controller.ExerciseManager

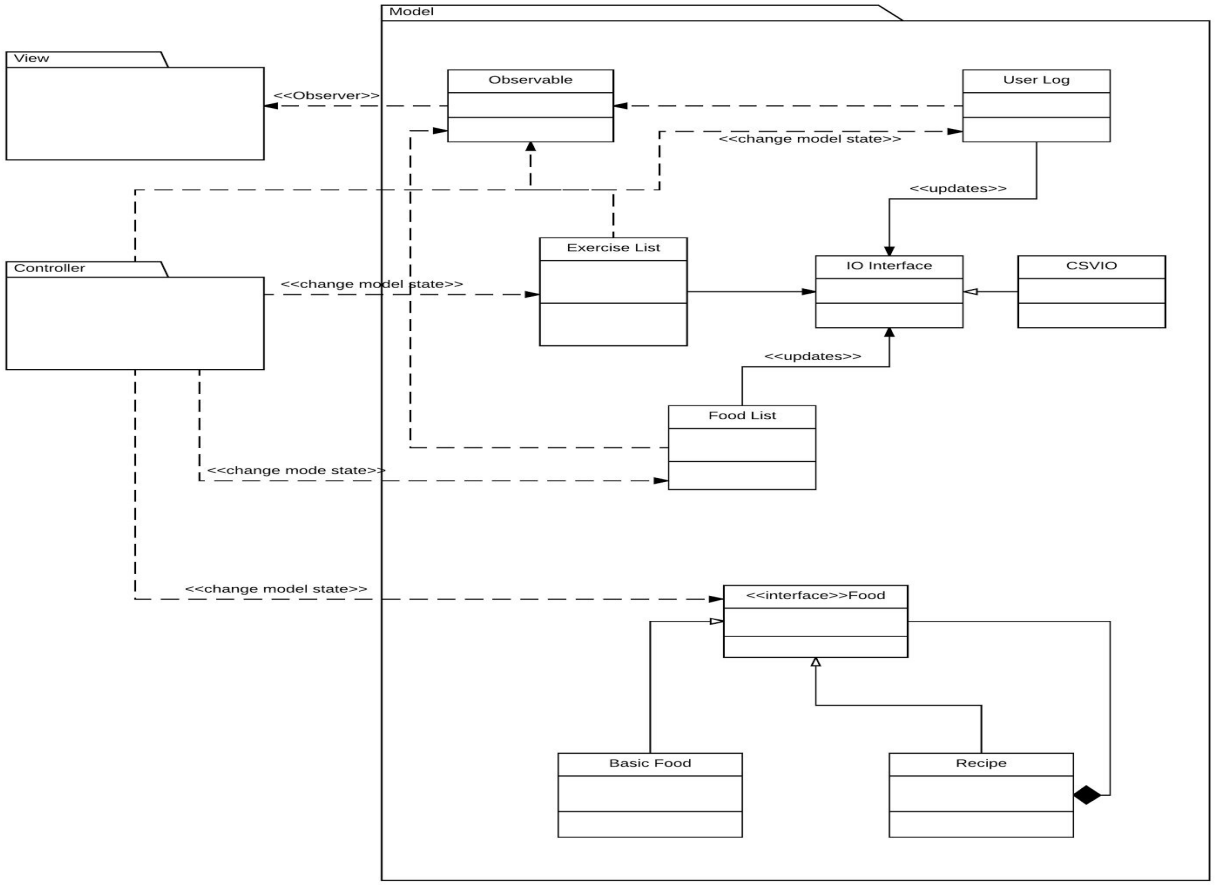
ExerciseList	
Responsibilities	A collection/list of Exercise. Maintains and load/save the collection to the Exercise CSV.
Collaborators (uses)	Controller.ExerciseManager Model.Exercise

Food	
Responsibilities	Keeps track of the different foods (basic foods and recipes). Recieves data from FoodList 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

Frame Interface	
Responsibilities	A general interface that will be extended by other frame classes.
Collaborators	create.JFrame

(uses)	

Display	
Responsibilities	Create objects from all the other frame classes.
Collaborators (uses)	create.WeightFrame create.AddFoodFrame create.DashBoardFrame create.FoodFrame create.ExerciseFrame

DashBoard Frame	
Responsibilities	The landing page frame for all users after the program launches.
Collaborators (uses)	extend..Frame create.JButton, JTextField, JLabel, JPanel

Exercise Frame	
Responsibilities	The Exercise frame for users to add and edit their collection of Exercise
Collaborators (uses)	extend..Frame create.JButton, JTextField, JLabel, JPanel

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

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

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

	view.FoodFrame

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

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

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

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

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

FoodFrame Frame	
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.dashboardFrame

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

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

Weight Text Field Panel

Responsibilities	Accepts the user's current weight
Collaborators (uses)	view.weightFrame 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.foodFrame

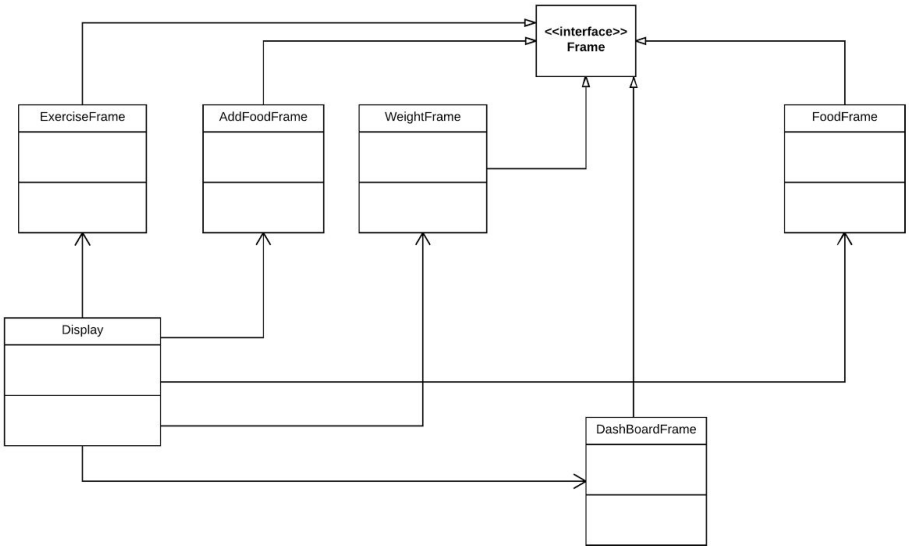
Food Frame	
Responsibilities	A JFrame to hold the food items
Collaborators (uses)	view.FatCarbsProteinGraphCanvas view.FoodRecipeTextFieldPanel

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

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

Submit Exercise JButton	
Responsibilities	Submits the current exercise in the exercise text field
Collaborators (uses)	view.ExerciseTextFieldPanel

Exercise Frame	
Responsibilities	JFrame to hold the Exercise items
Collaborators (uses)	view.CaloriesExpendedTextField view.MinutesExerciesedTextField

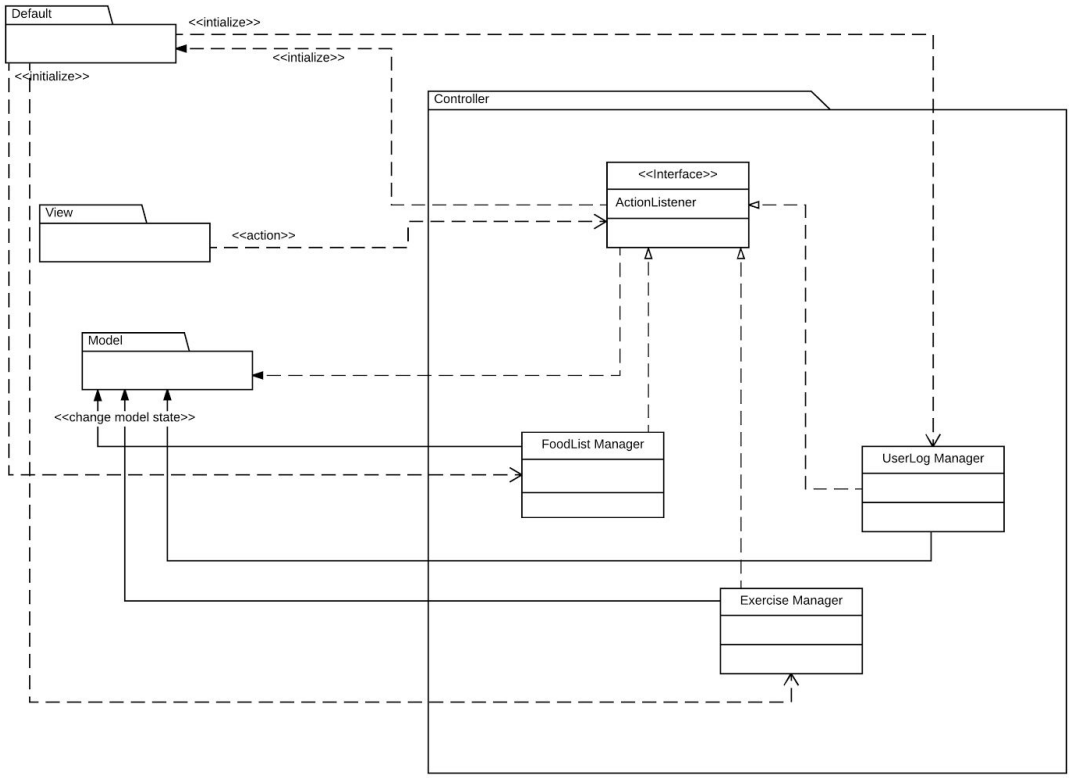


Controller

User Log Manager	
Responsibilities	Initializes and updates User Log class from user input in the Display class. Calculates the net calories from the ExerciseList and the FoodList Managers.
Collaborators (uses)	model.UserLog actionListener

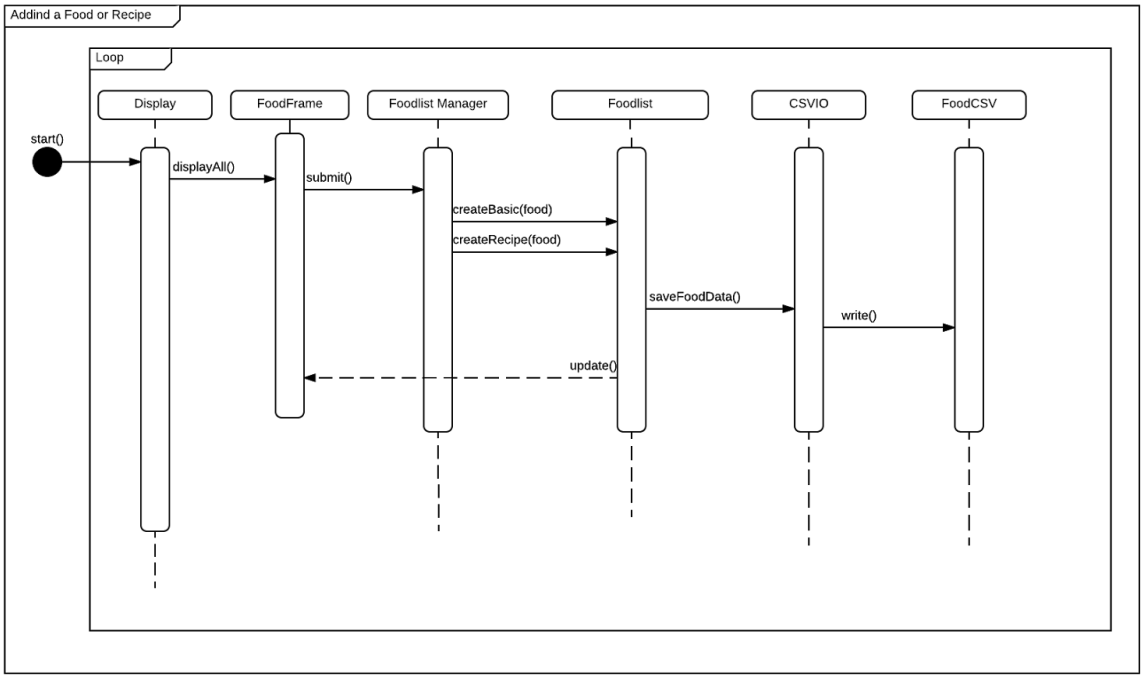
ExerciseList Manager	
Responsibilities	Initializes and updates ExerciseList class from user input in the Display class. Initializes and creates/updates Exercise components. Calculates all the calories expended from exercises.
Collaborators (uses)	model.ExerciseList model.Exercise actionListener

FoodList Manager	
Responsibilities	Initializes and updates FoodList class from user input in the Display class. Initializes and creates/updates Food components. Calculates all the calories consumed from food.
Collaborators (uses)	model.FoodList model.Food actionListener

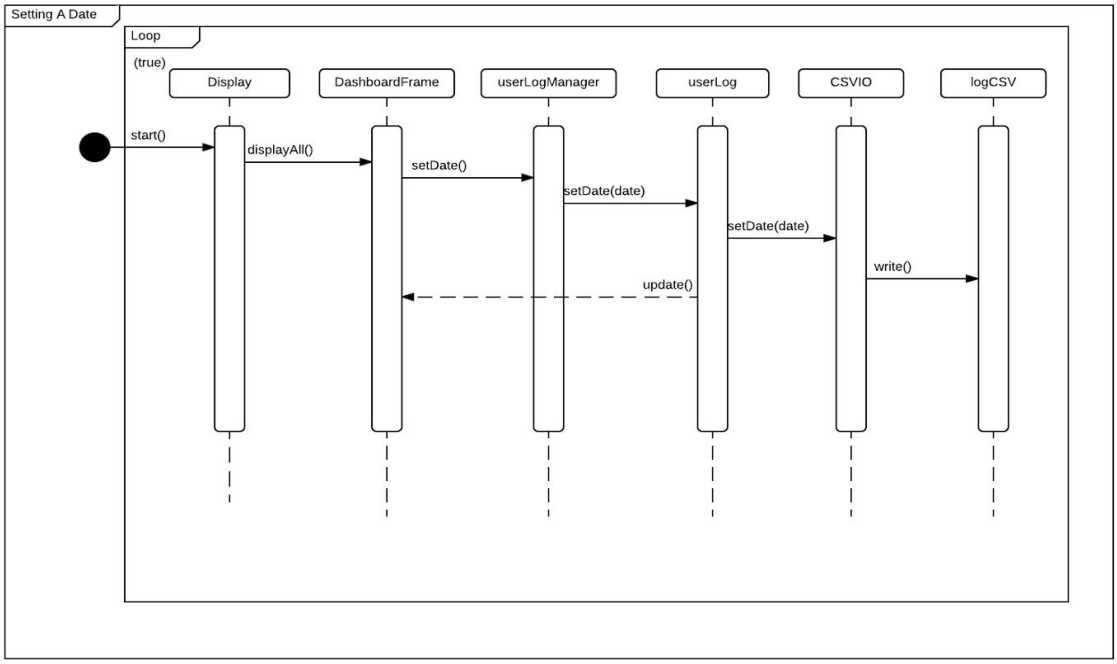

Default
<<intialize>>
<<intialize>>
<<initialize>>
Controller
<<Interface>>
ActionListener
View
<<action>>
Model
<<change model state>>
FoodList Manager
UserLog Manager
Exercise Manager

Sequence Diagrams

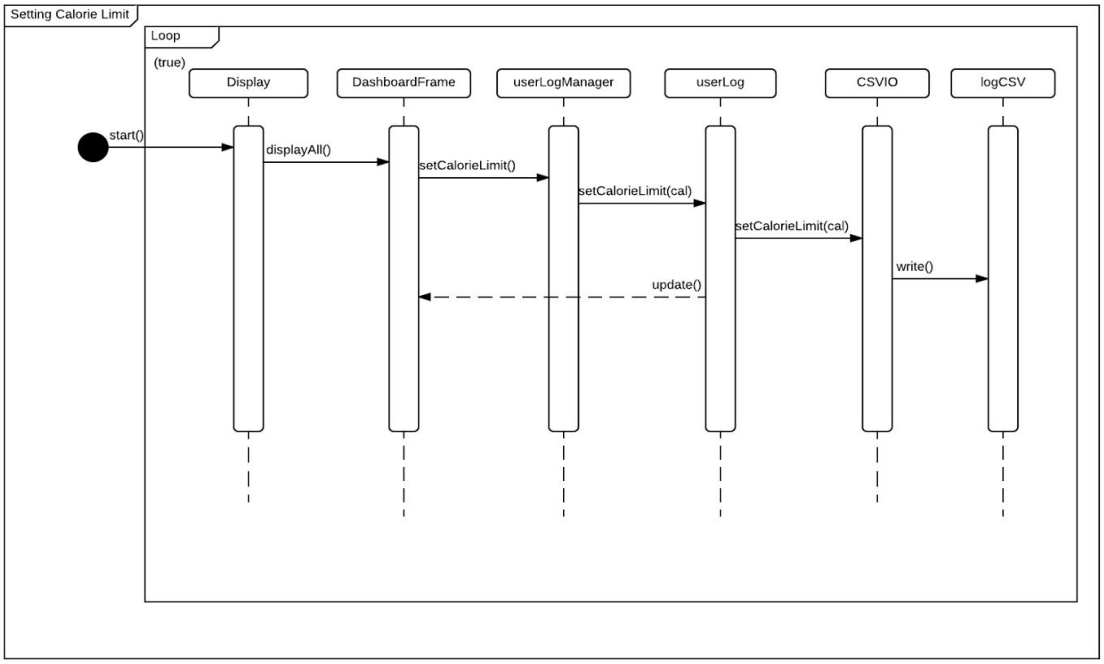
Adding a new basic food or recipe to the foodlist



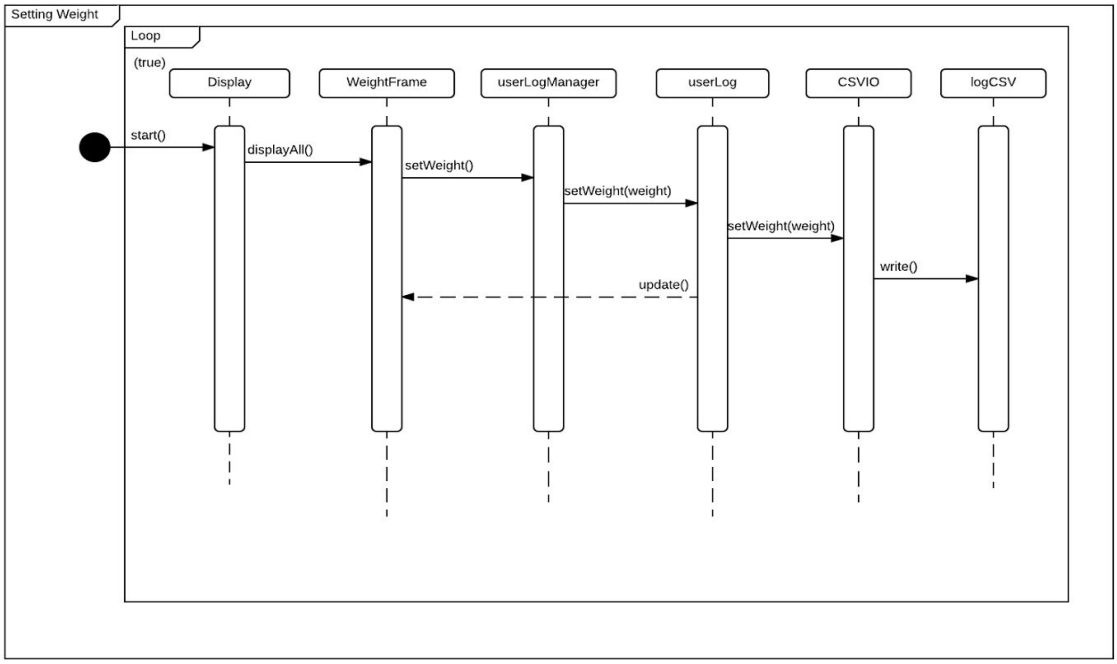
Setting the date



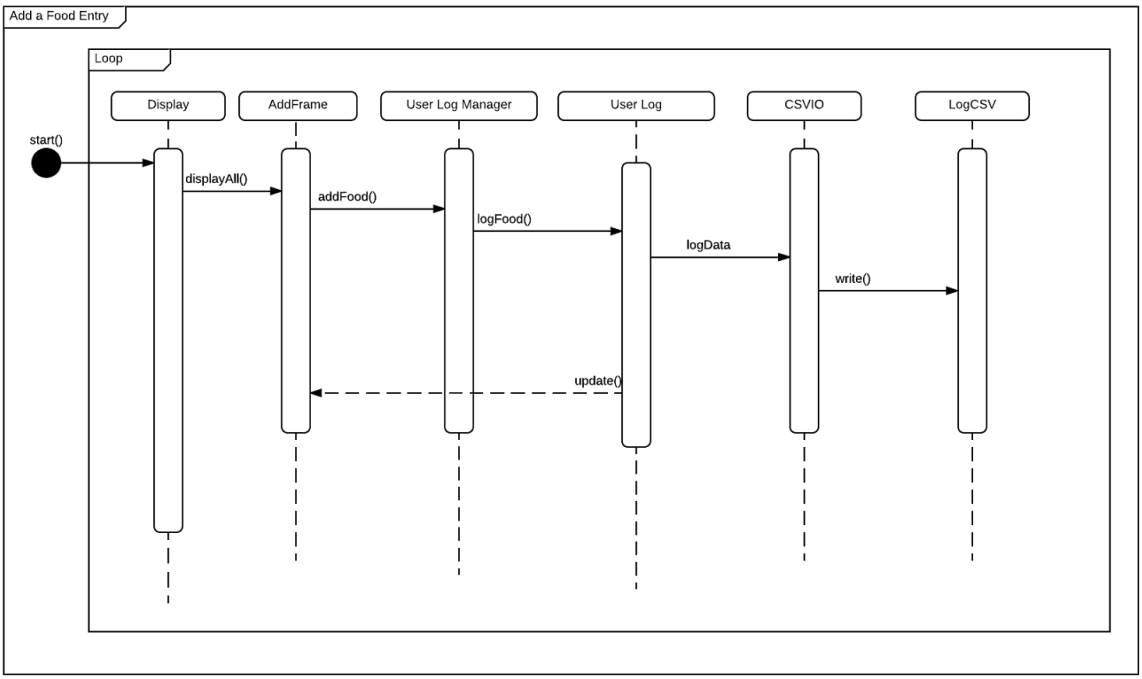
Setting the calorie limit



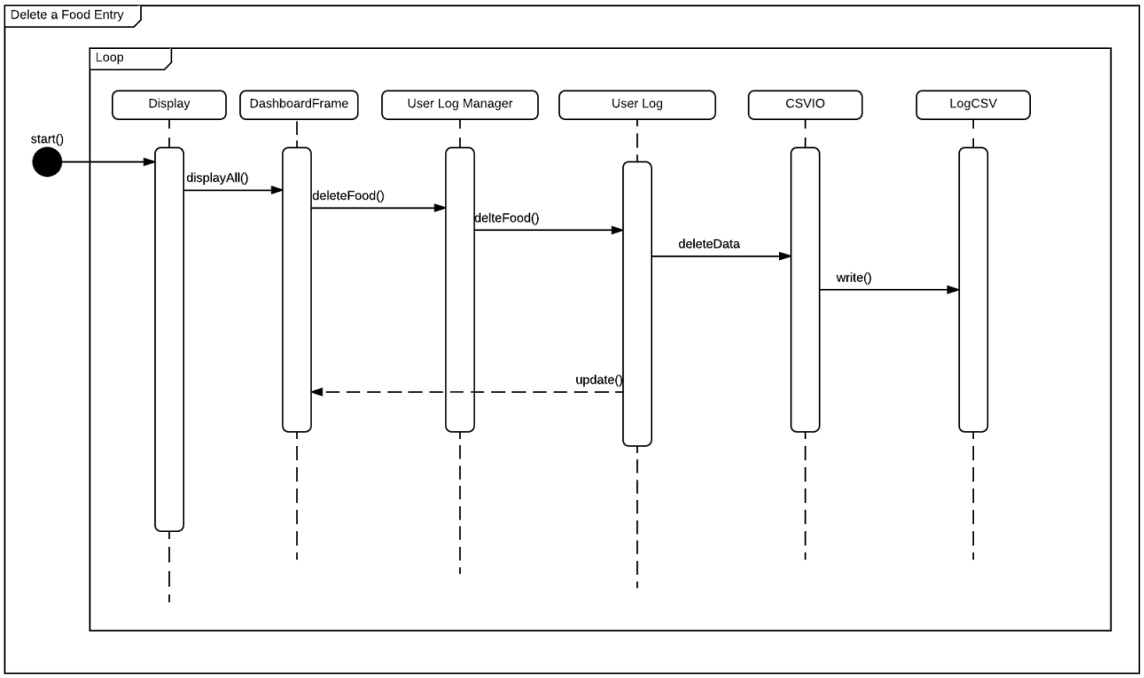
Setting the weight



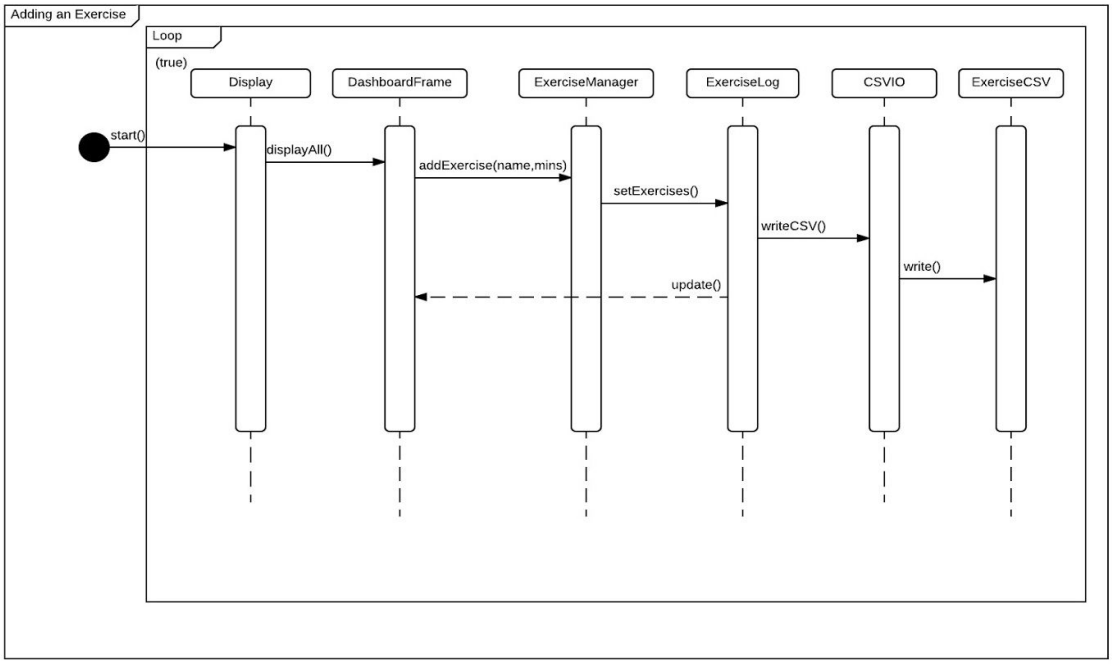
Add food intake to user log



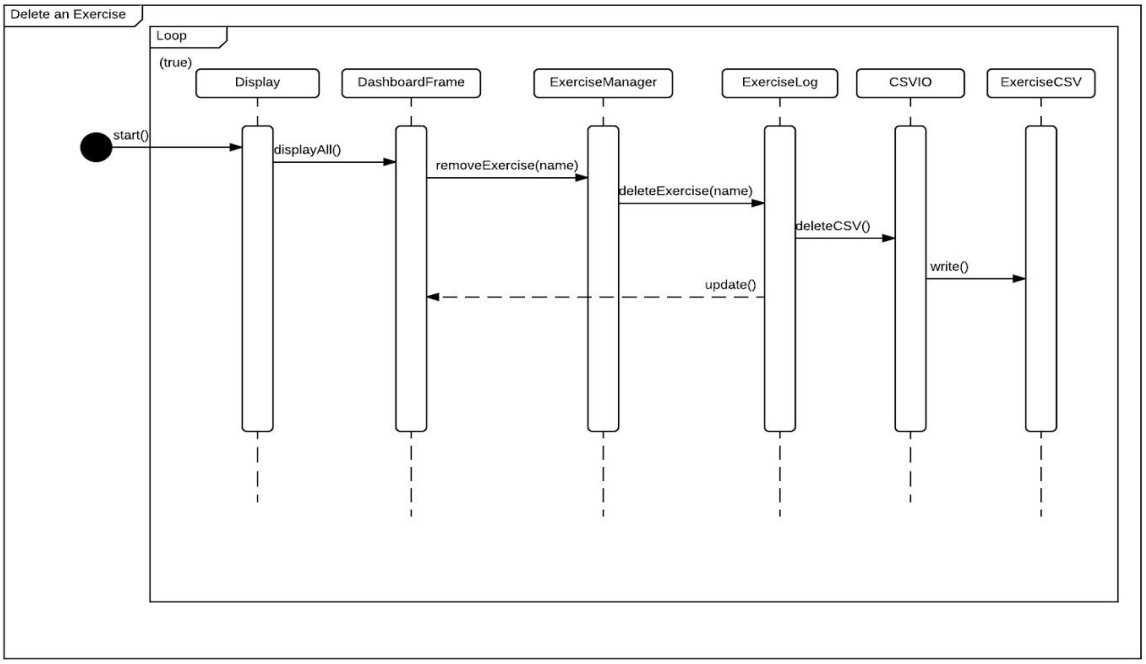
Deleting food entry from the user log



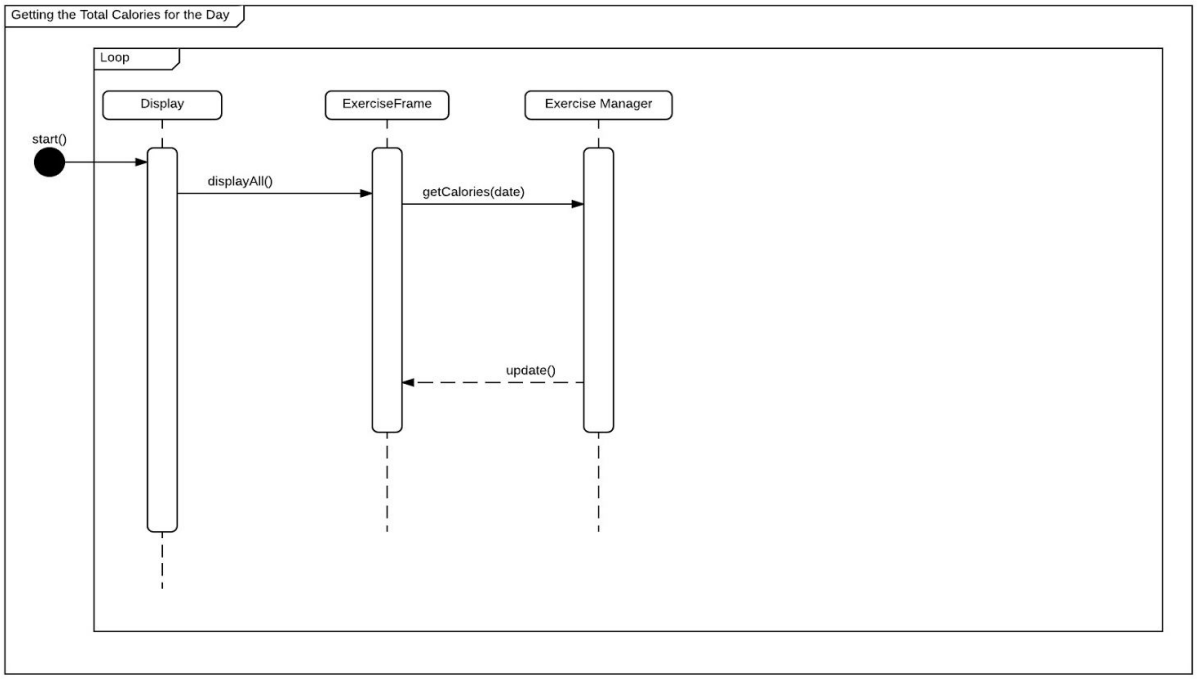
Add Exercise



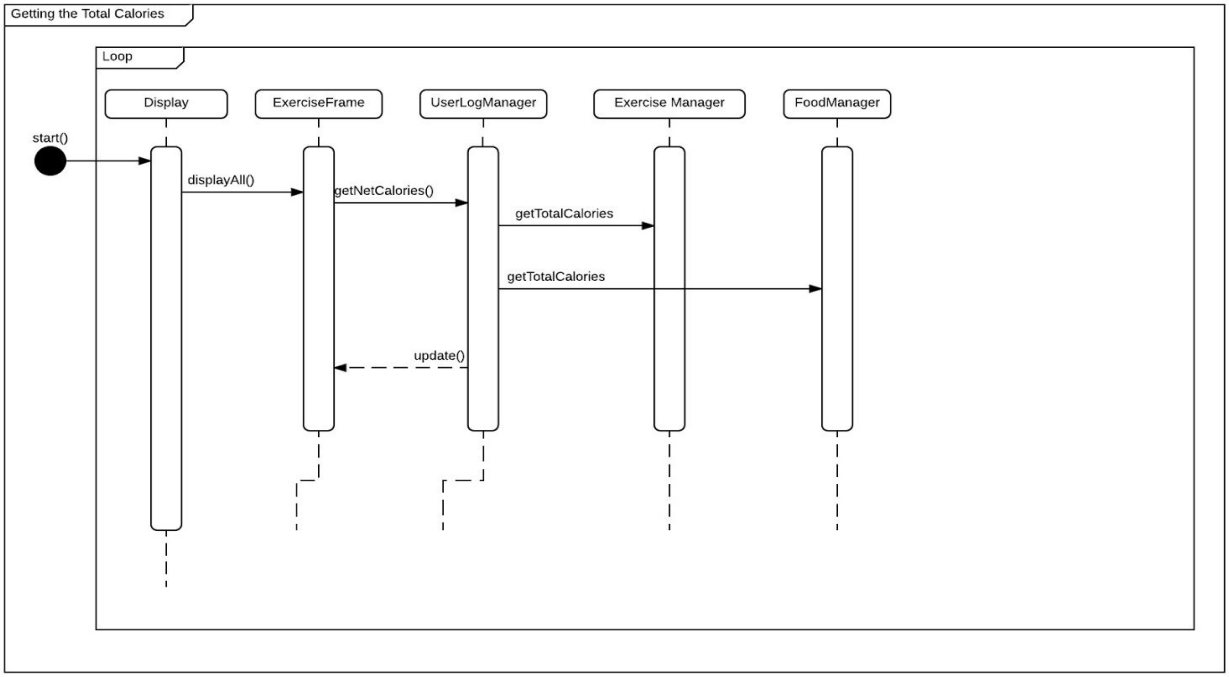
Delete Exercise



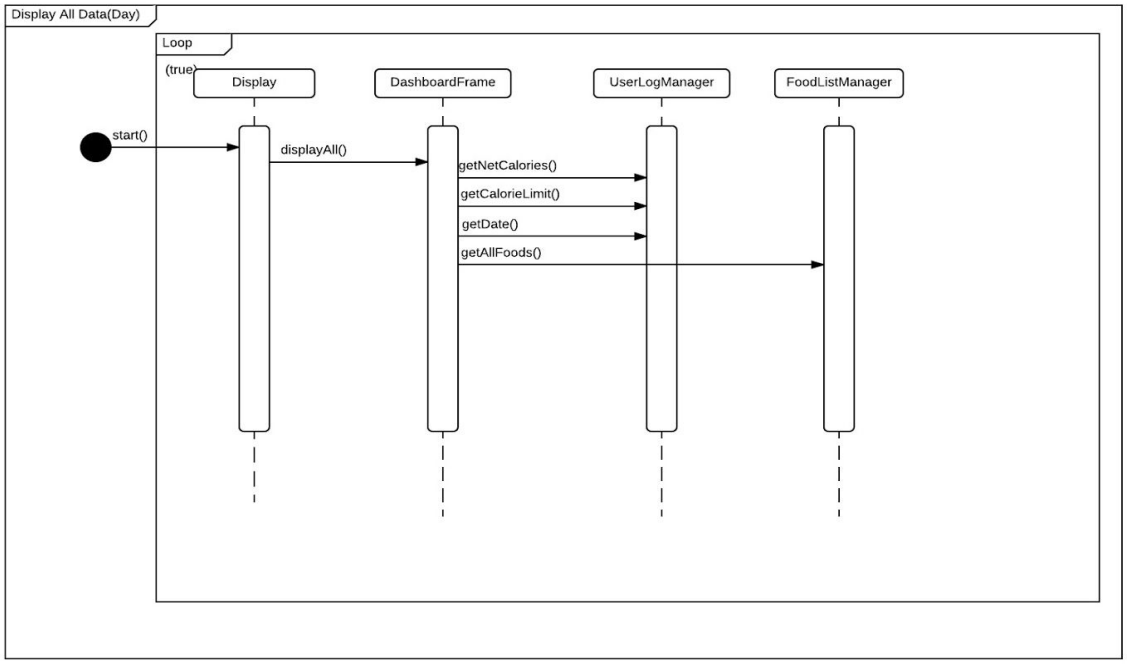
Getting Total Calories for the Day



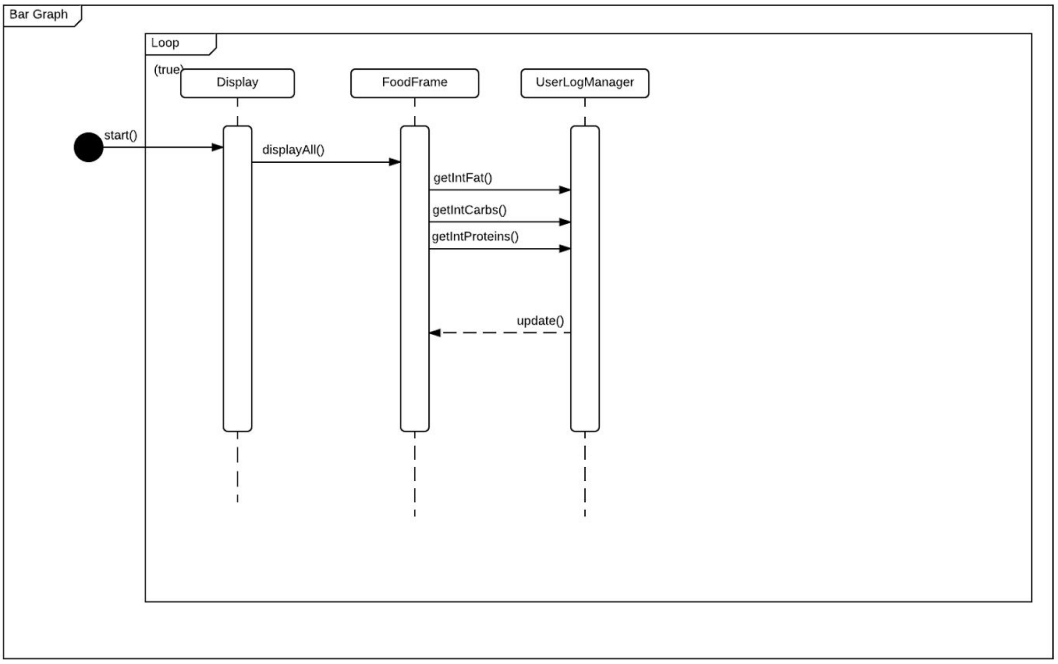
Getting the Total Net Calories



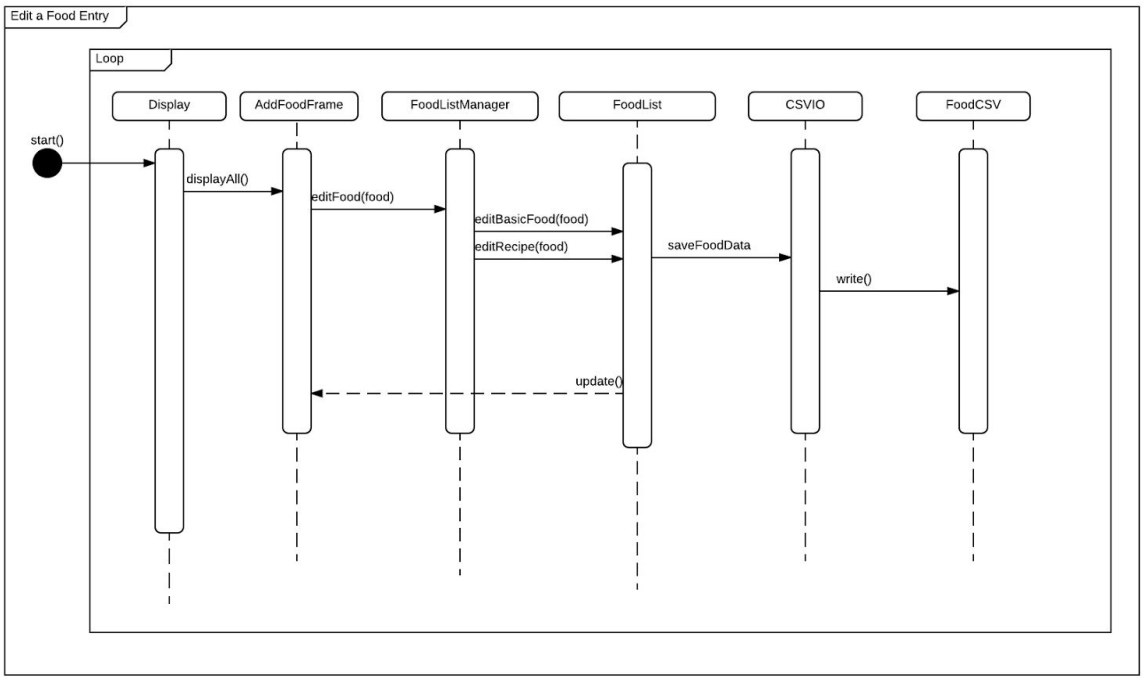
Display All Data Related for the Day



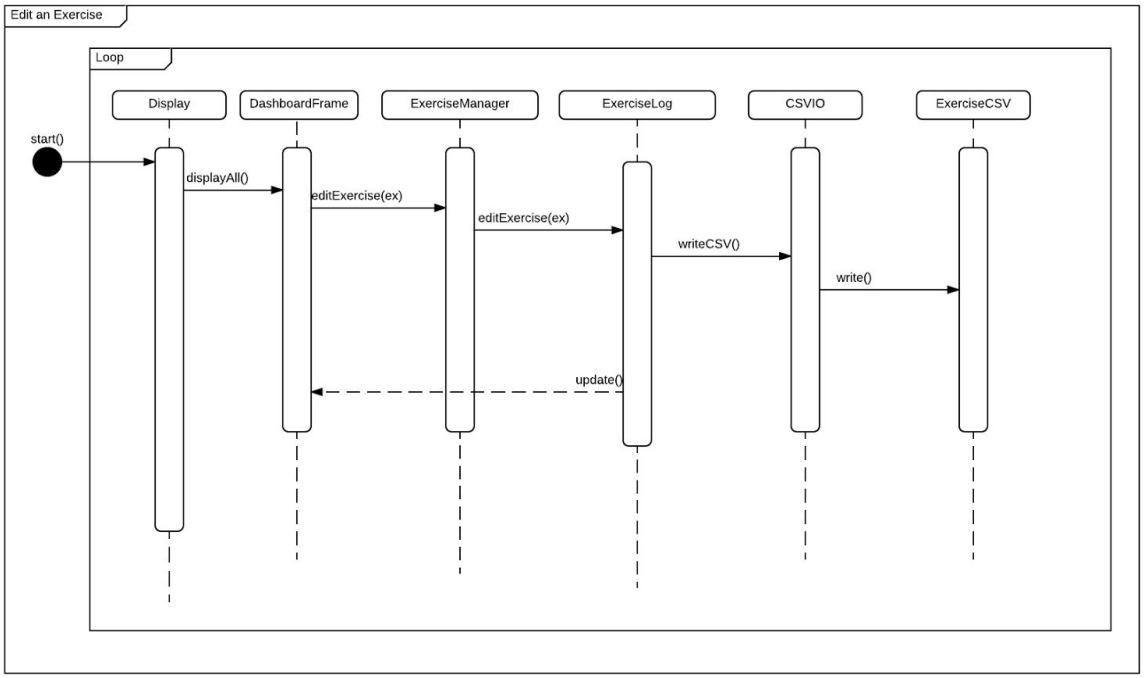
Bar Graph



Edit a Food



Edit an Exercise



Pattern Usage

Observer Pattern

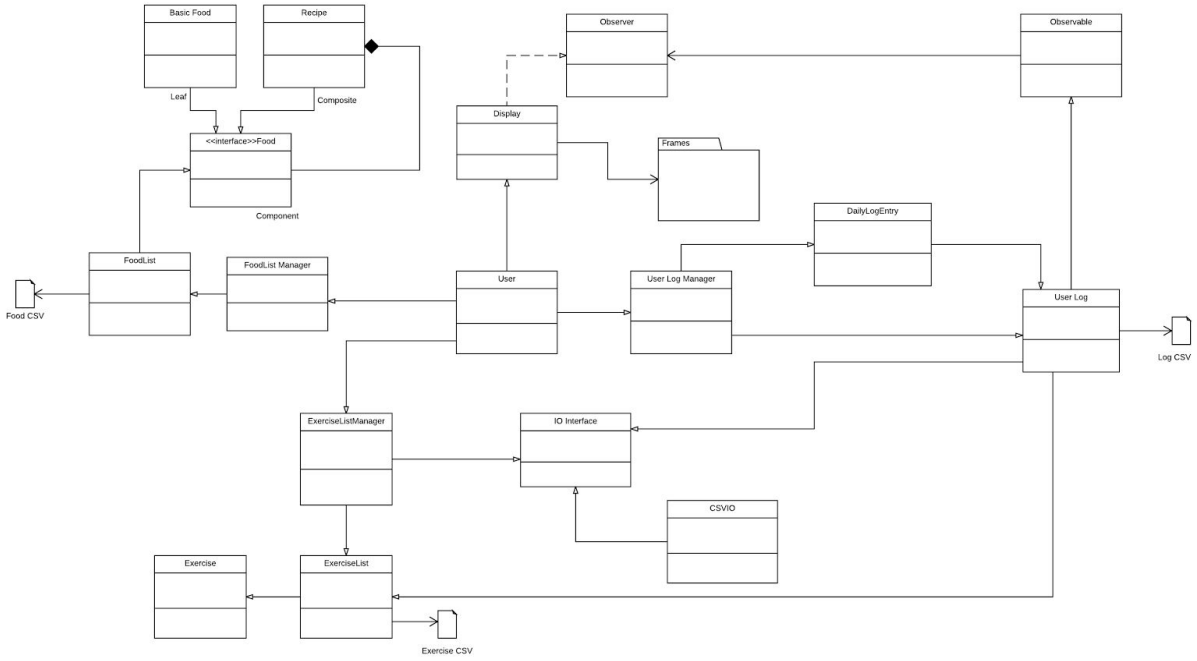
Observer Pattern	
Observer(s)	Display
Observable(s)	User Log, FoodList, ExerciseList

Composite Pattern

Composite Pattern	
Component	Food
Leaf	Basic Food
Composite	Recipe

MVC Pattern

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


Basic Food
Recipe
Leaf
Composite
<<interface>>Food
Component
Observer
Observable
Display
Frames
DailyLogEntry
FoodList
FoodList Manager
User
User Log Manager
User Log
Food CSV
Log CSV
ExerciseListManager
IO Interface
CSVIO
Exercise
ExerciseList
Exercise CSV