Diet Manager 1.0

Project Design Document

TEAM E

Vignesh Kumar, vk4534@rit.edu

Vincent Cheng, vkc9448@rit.edu

Efmajackson Rosario, exr4361@rit.edu

Ryan Wren, rdw6297@rit.edu

Project Summary

The Diet Manager application will be a useful tool for people to record important nutrition information over a long period of time. The app will be easy for the user to use with a simple interface and a quick simple user input for regular use. The user will be able to log their diet by recording the food they eat each day by selecting a food from the collection or adding a new one to the collection. The user can also retrieve the calculated information for past days from the log file. Users can store information like their weight and set daily calorie limits.

Food collection/hub contains information on fat content, carbs, protein, and calories. The user can retrieve the consumption information of food eaten by day. Calculations will be shown to provide the user with their daily nutrition information as well. All of this will be presented in a minimalistic easy to read format for the user.

The daily logs will store information like the date, weight calorie limit, and food items eaten. The user can set their weight each day or if they do not it will use the last recorded weight. If a user does not specify the calorie limit for a day then 2,000 will be used. Users can also delete food entries from the log for a specific day.

Diet Manager will work to help users easily keep track of their food intake with as little hassle as possible.

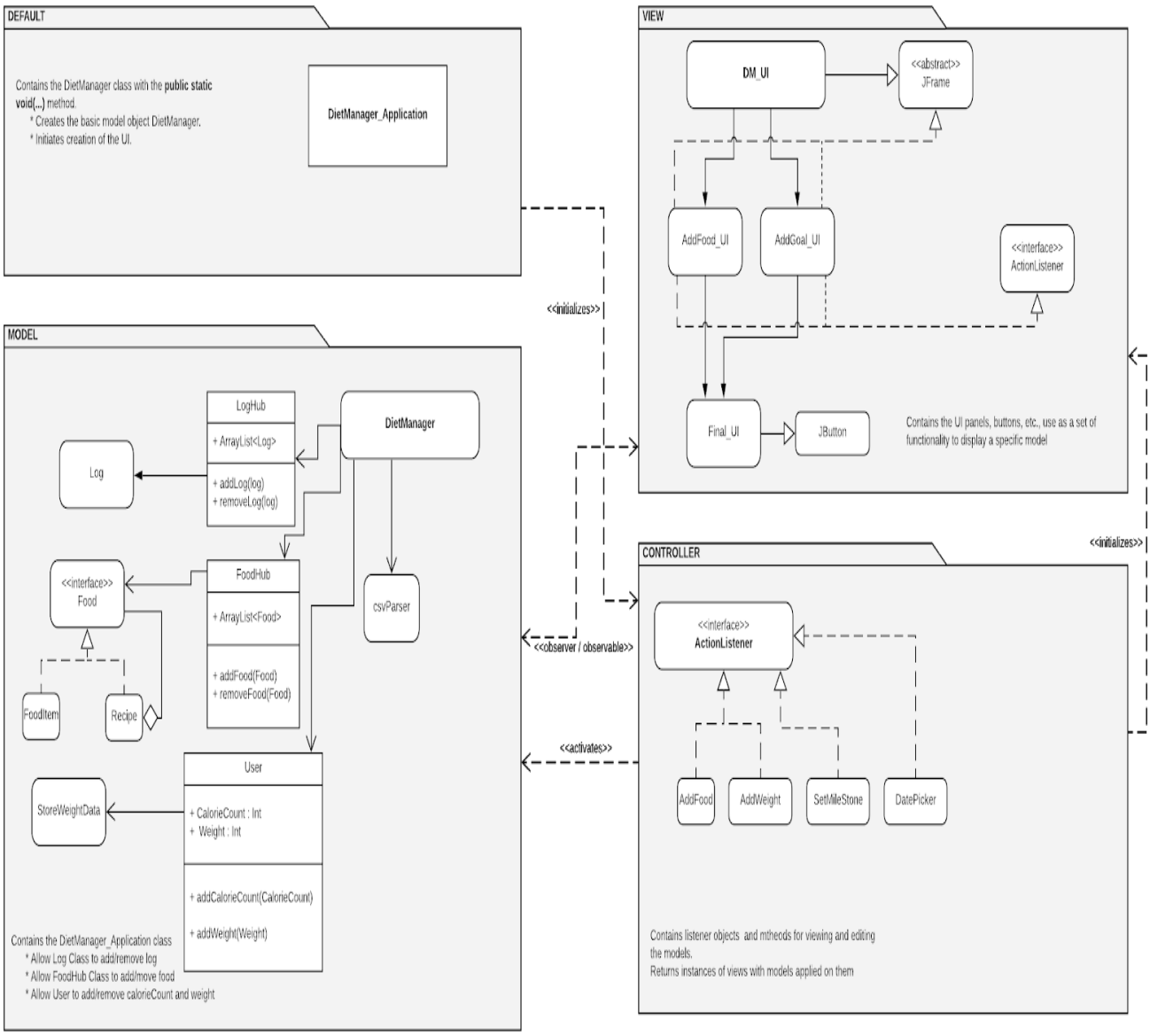
Design Overview

Starting off, we kept it broad to capture the main objective of what we wanted our design to achieve featuring a controller that gets notified from the model and then passes on the information for display to the user. Once we had the concrete idea that we wanted the user to input an entry of the food eaten along with a timestamp we moved forward in breaking every functionality down into its own pieces and going more depth into the process of the design once the user initiates the diet manager application.

Moving forward we progressed more into the detailed methods of each class to make sure that we all have an understanding of what each subsystem is in charge of initiating as it moves along the process. We started with the MVC pattern by making sure we had the model as the backbone with all the main functions, the view to display the output and user interface of the results, and controller to communicate with the model to set entries from the user request and send them back to the view with the outcome.

Once we had the MVC pattern, we had to implement the Composite pattern so we broke down the interface food which is the one class required for the system to function properly into having its own subsystem consisting of the composite which in our case is the recipe class which is skeleton of the class food and food items where food items is the leaf of food, which depends on the receipt to deliver the user an accurate result towards their goal and set milestone.

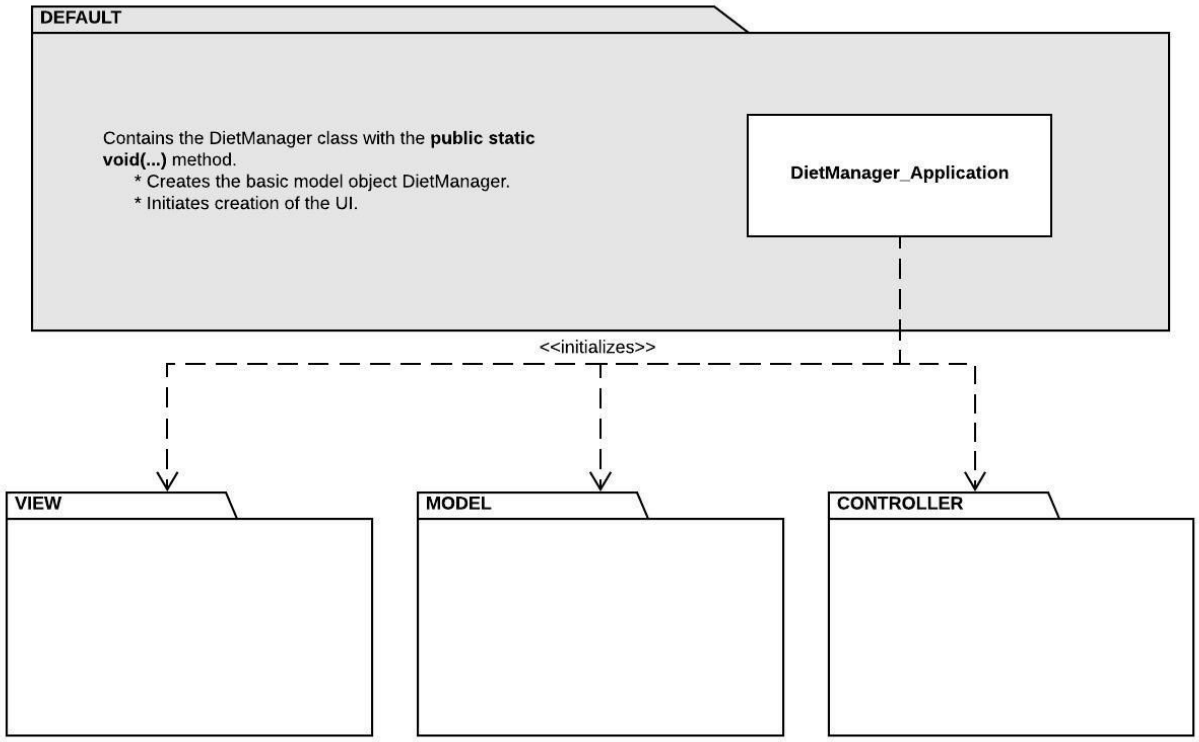
Subsystem Structure



Subsystems

Subsystem Default

Class DietManager_Application	
Responsibilities	It initializes the controller which in turn initializes the view for the user to start interacting with the GUI
Collaborators (uses)	model.DietManager_Application



Subsystem Model

Class DietManager	
Responsibilities	Notify User interface of all the changes.
Collaborators (uses)	

Class csvParser	
Responsibilities	Enables the entries to be read/written to a CSV file
Collaborators	java.io.*-> To use printwriter to write to csv file. java.util.* ->To use the date to pick and write the date. java.io.*-> To use File or Buffered Readers to read from a csv file. java.util.* ->To use the date to pick and read the date.

Class Log	
Responsibilities	It acts as an entry/record for the Write class. It stores the user's weight and his max calorie count.
Collaborators	Model.StoreFoodData Mode.StoreWeightData

Class StoreWeightData	
Responsibilities	Stores the weight of the user along with a timestamp. The date and the weight are stored in an object.
Collaborators	java.util.* ->To use the date for the timestamp.

Class User	
Responsibilities	Helps recording and changing the weight of the user and the user's calorie count.
Collaborators	java.io.*-> To use File or Buffered Readers to read from a csv file. java.util.* ->To use the date to pick and read the date.

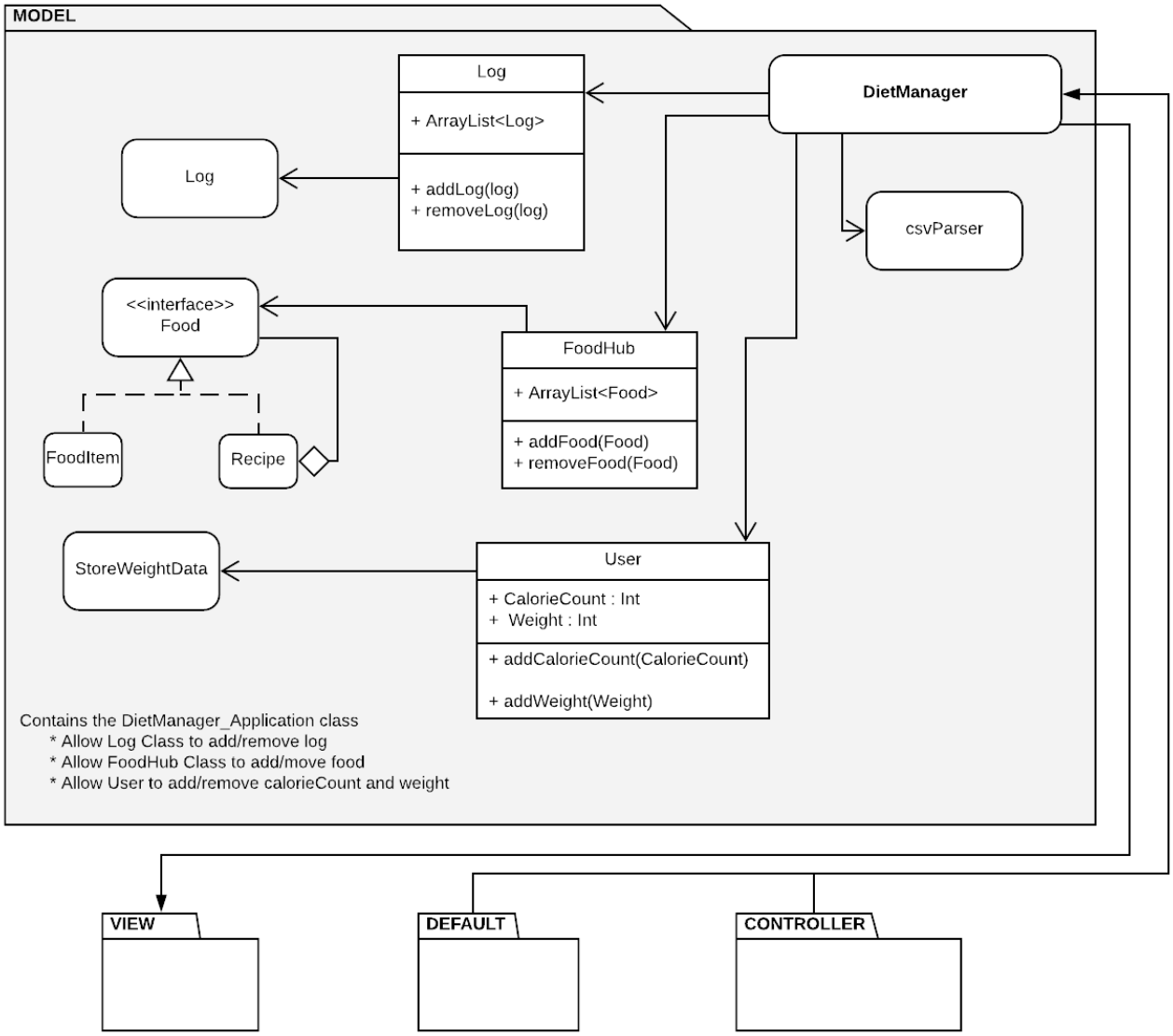
Class StoreFoodData	
Responsibilities	Stores the food eaten by the user along with a timestamp. The date and the consumed food are stored in an object.
Collaborators	java.util.* ->To use the date for the timestamp.

Class Food <<Interface>>	
Responsibilities	Provides interface for all. (food items and recipe) getDiet() - Returns information related to nutrition of a diet. getFoodName() -Returns the name of the food item or the recipe.

Class FoodItem	
Responsibilities	Attributes: Name, Calories, fat, carb and proteins. Stores the name of the food and number of calories it contains. Also sores number of grams of fat, carb and protein in one of the food. Methods: getDiet() - Returns information related to nutrition of a diet. getFoodName() -Returns the name of the food item.
Collaborators	Model.Food

Class Recipe	
Responsibilities	It is a collection of food items and recipes. It consists of name of the recipe, name of the sub recipe/ food items that make together the main recipe since it is a composition. It also consists of number of servings of each food item/sub-recipe. Methods: getDiet() - Returns information related to nutrition of a diet. getFoodName() -Returns the name of the recipe.

Collaborators	Model.Food



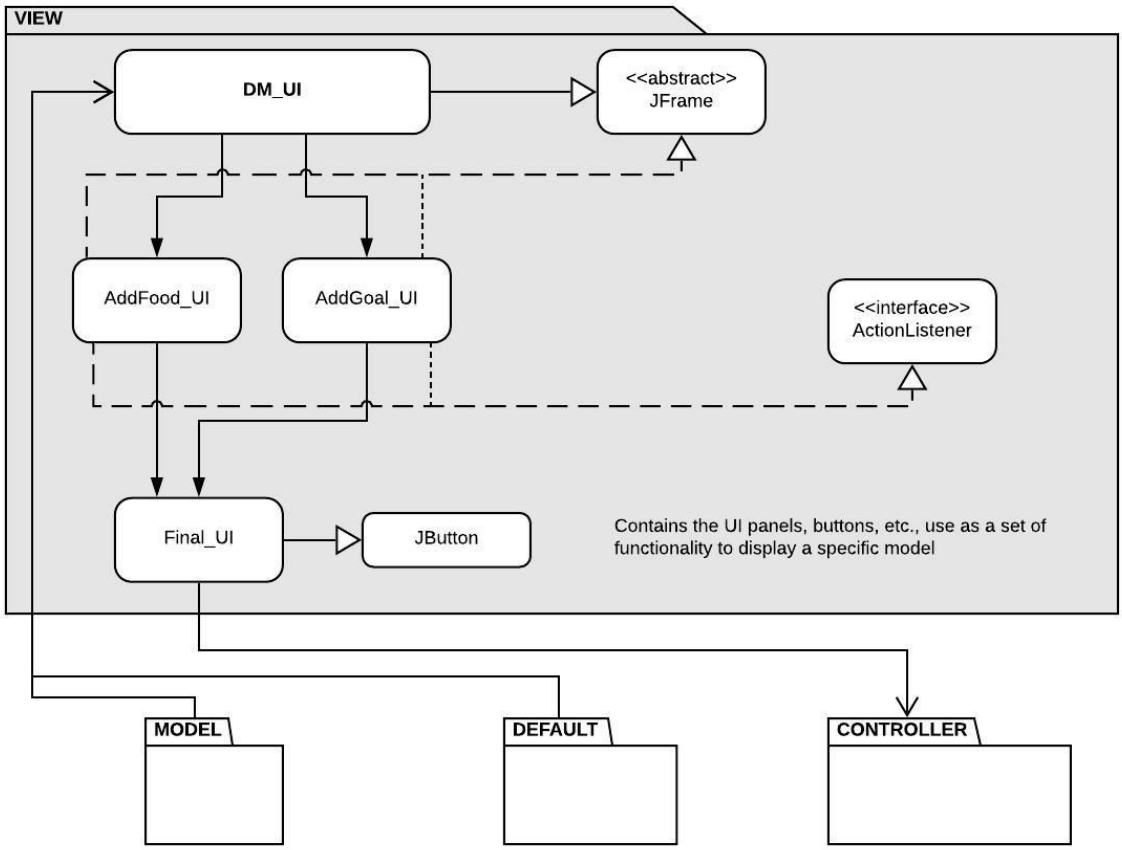
Subsystem View

Class DM_UI	
Responsibilities	The User Interface of the DietManager that helps the user to add food, update weight and also to set a calorie intake limit.
Collaborators (uses)	Implements ActionListener. All of the classes in View Subsystem.

Class AddFood_UI	
Responsibilities	It consists of the necessary UI components to enable the user to add food items or recipe and also add the food eaten to the log.
Collaborators	Controller.AddFood

Class AddGoal_UI	
Responsibilities	It consists of the necessary UI components to enable the user to add desired caloric intake and their weight.
Collaborators	Controller.AddWeight Controller.SetMilestone

Class Final_UI	
Responsibilities	It consists of necessary UI components to display all the nutritional information pertaining to the user
Collaborators (uses)	



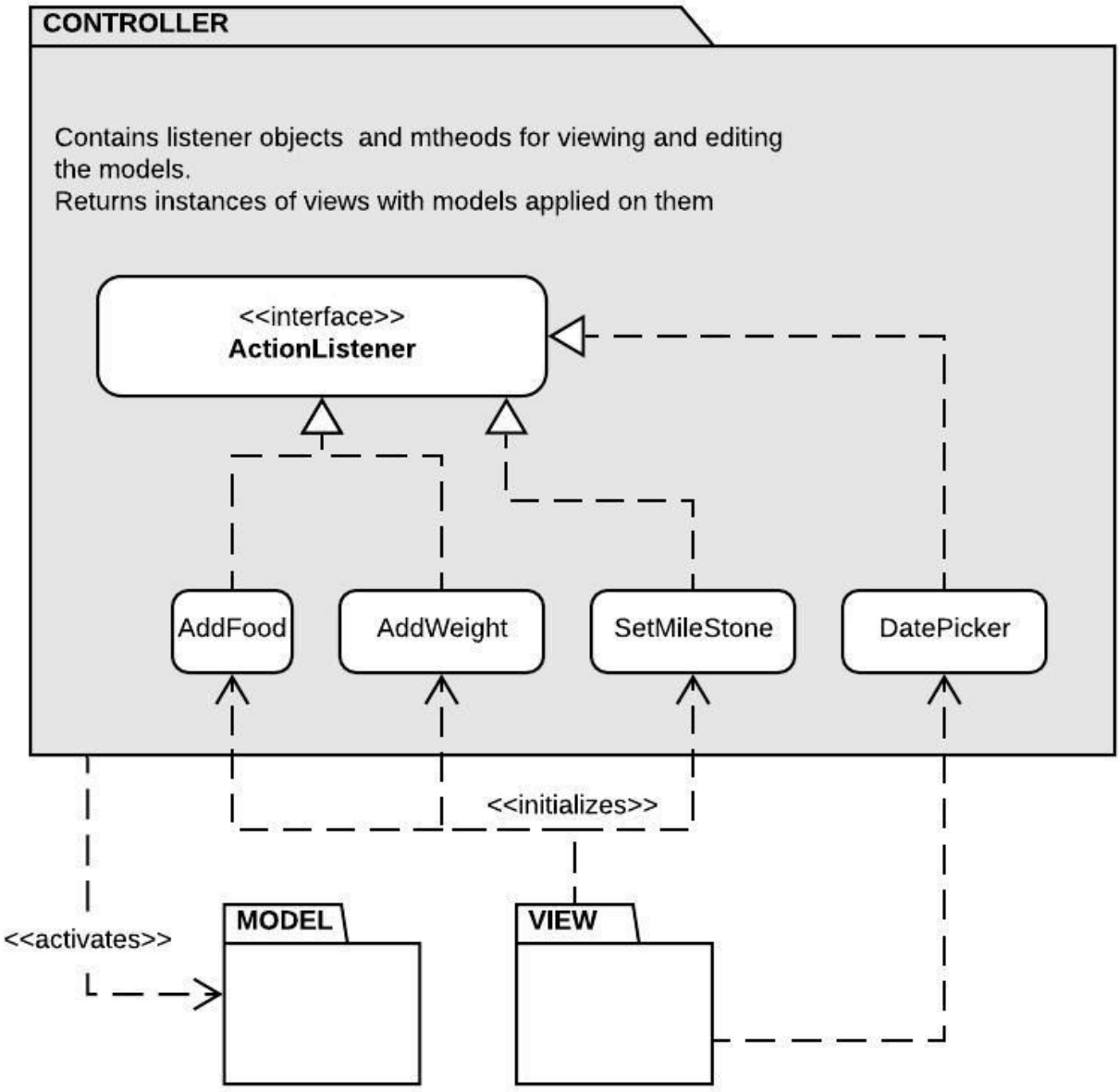
Subsystem Controller

Class AddFood	
Responsibilities	Creates an object which will be added to the log and food collection.
Collaborators (uses)	Model.Food Model.Write Model.DietManager_Application

Class DatePicker	
Responsibilities	Searches the log using the date picked by the user.
Collaborators	Model.DietManager_Application

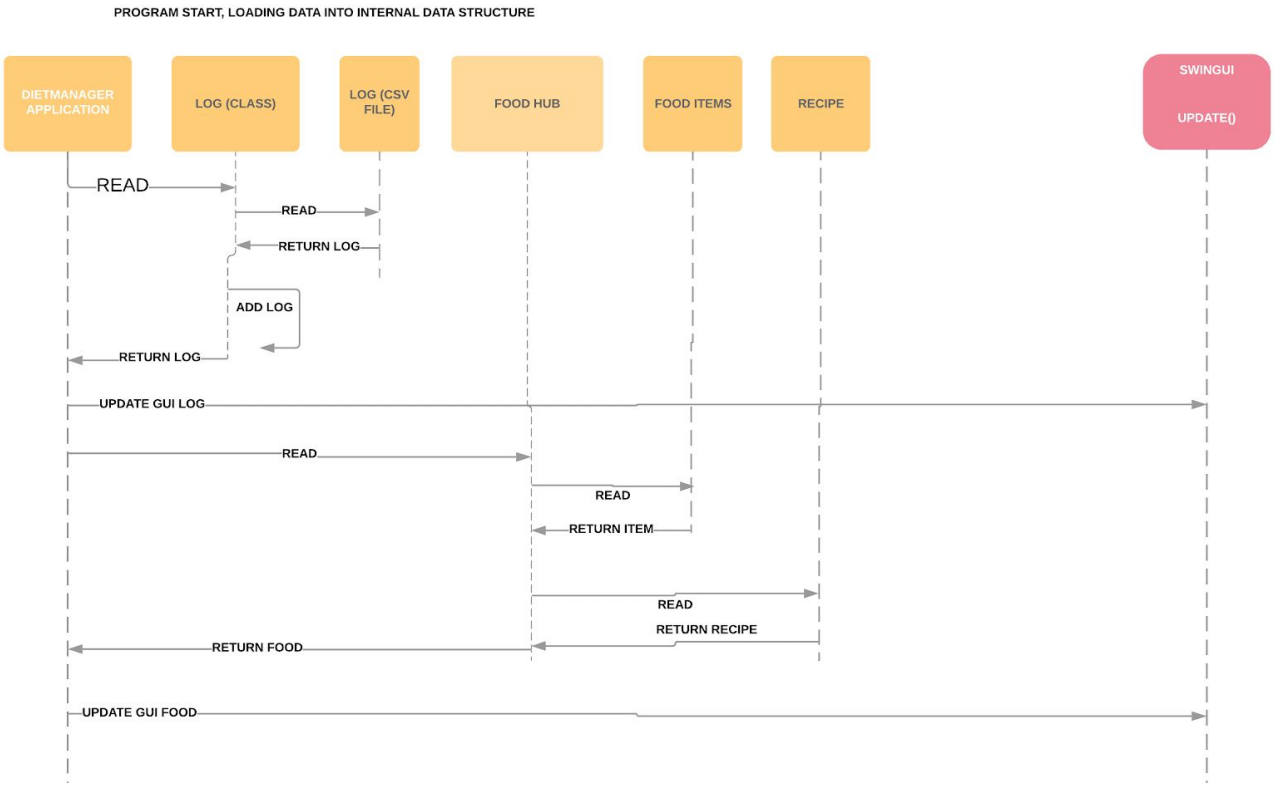
Class AddWeight	
Responsibilities	Adds/updates the user's weight
Collaborators	Model.User Model.DietManager_Application

Class SetMileStone	
Responsibilities	Updates the user's calorie intake limit and desired weight to be attained.
Collaborators	Model.User Model.DietManager_Application

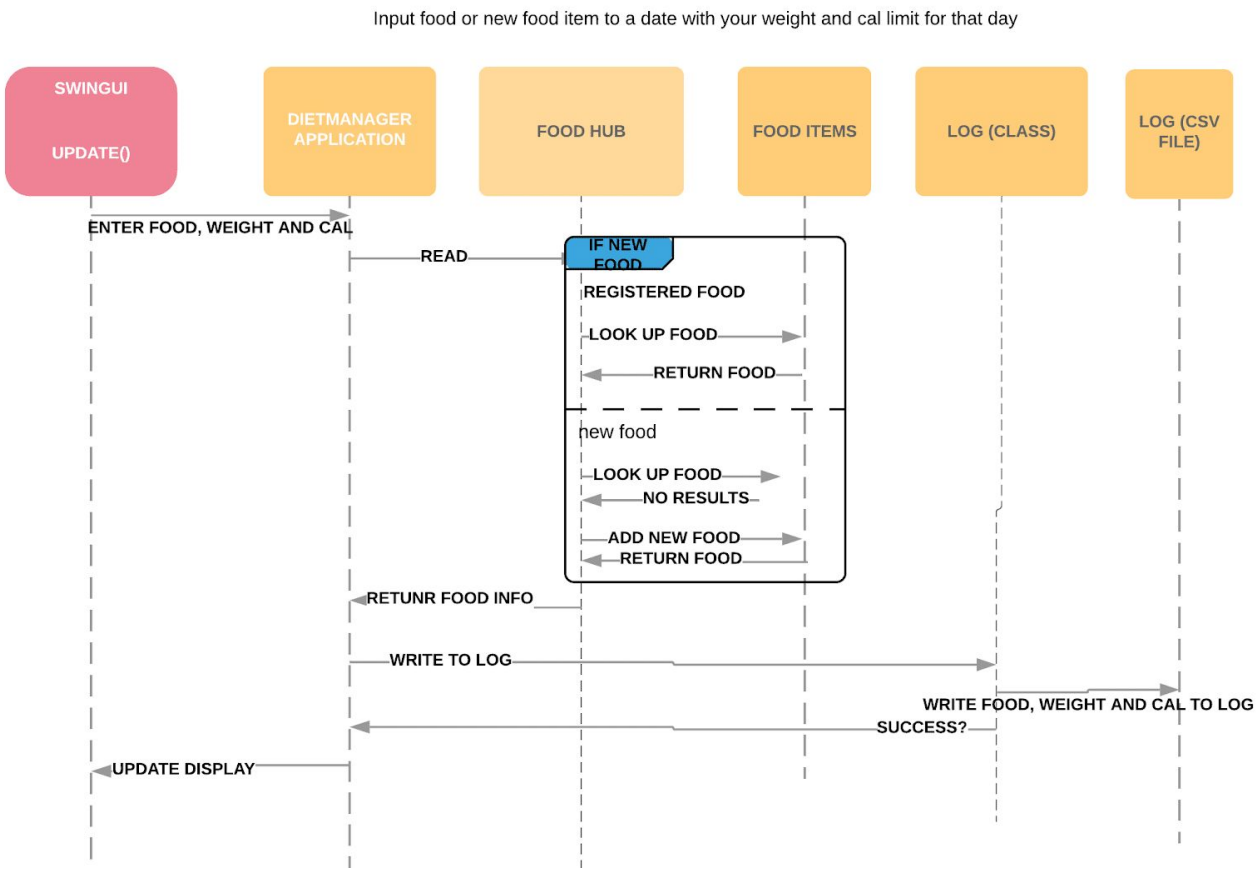

CONTROLLER
Contains listener objects and mtheods for viewing and editing the models.
Returns instances of views with models applied on them
<<interface>>
ActionListener
AddFood
AddWeight
SetMileStone
DatePicker
<<initializes>>
<<activates>>
MODEL
VIEW

Sequence Diagrams

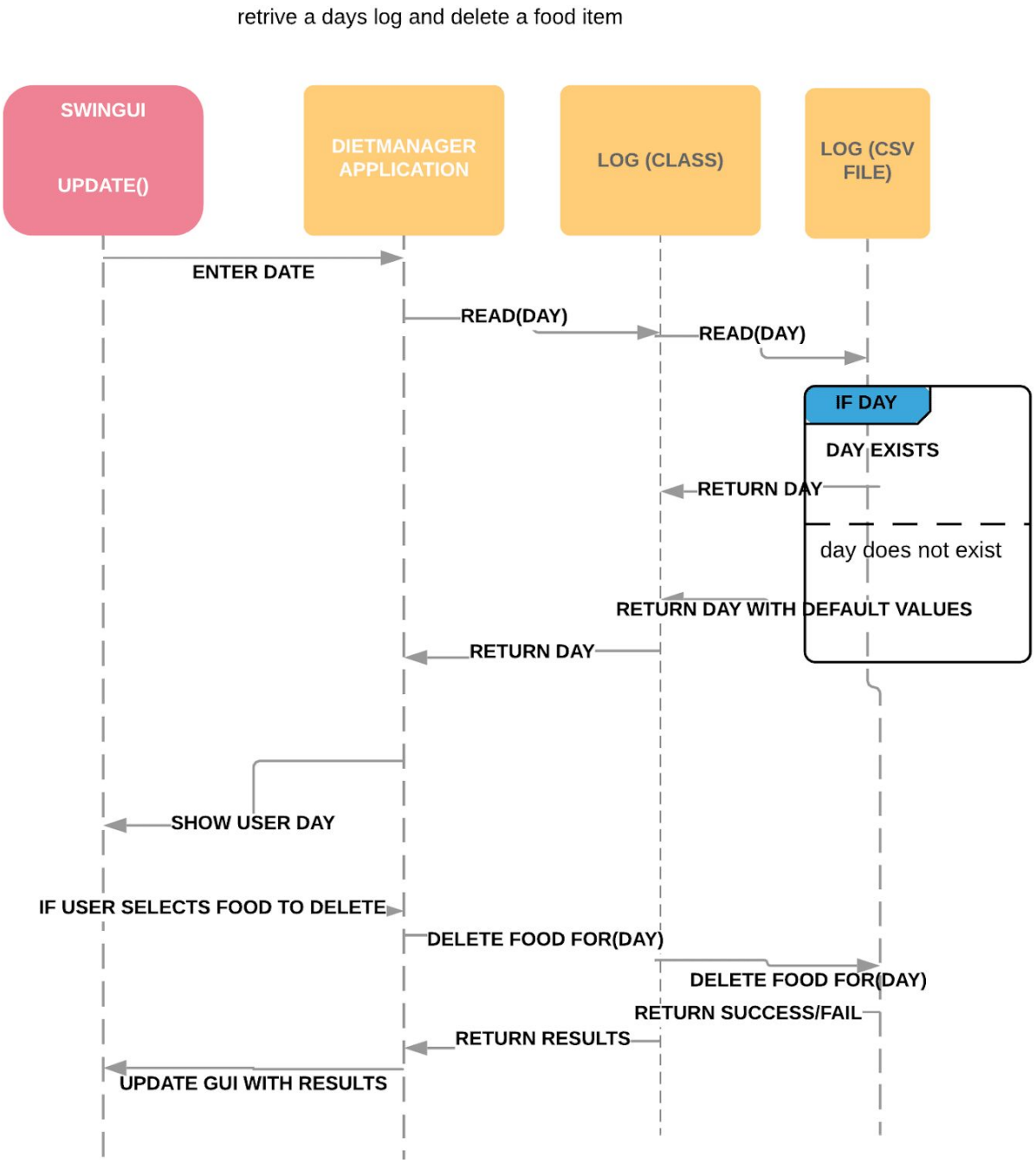
Start up load food and log into internal data structures



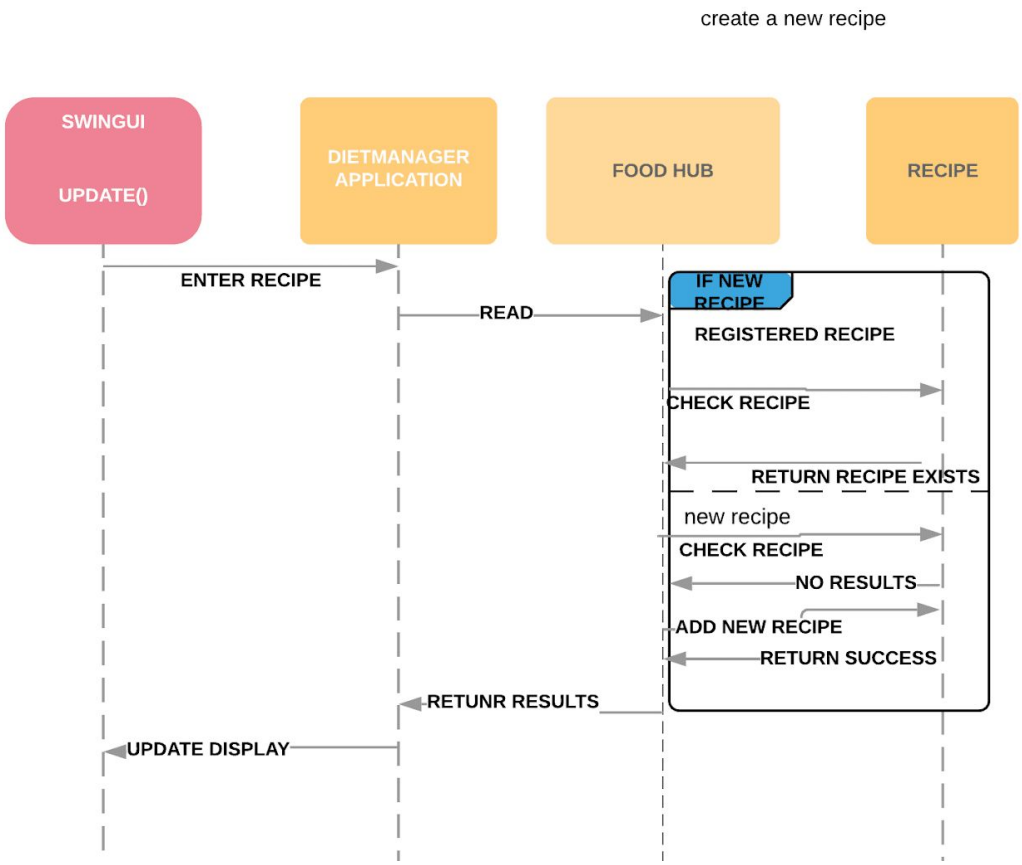
Input food or new food item to a date with your weight and cal limit for that day



Retrieve info for a day and delete a food entry



Create a recipe from foods



Pattern Usage

Pattern #1 Composite

The Composite pattern ensures high cohesion between recipe and FoodItems and less coupling overall.

Composite Pattern	
Composite	Recipe
Leaf	FoodItems
Component	Food

Pattern #2 Model View Control (MVC)

The Application also uses the MVC pattern for a better organization. This pattern helps in reducing the coupling and increase the overall cohesion of the system.

MVC Pattern	
Model	DietProgram_Application, Write, Read, User, Log, StoreWeightData, StoreFoodData, FoodItems, Recipe
View	DM_UI, AddFood_UI, AddGoal_UI, Final_UI
Controller	SetMileStone, AddGoal, Datepicker, AddWeight

Once the user interacts with the view subsystem, it sends a request to controller to get a specific data. The controller then sends this request to model to get the required data. The acquired data is sent from model to the view via the controller