PROJECT / RELEASE

Project Design Document

Group 1

Amra Rrahi <ar9393@rit.edu>

Arian Vucelić <av7189@rit.edu>

Ejup Elezi <ee1641@rit.edu>

Janko Leskovac <jl8592@rit.edu>

Roko Hafner <rh9922@rit.edu>

# Project Summary

The Diet Manager Application is a software tool that is designed to assist users in managing their dietary habits effectively. It integrates various features including a vast collection of foods, recipes, daily intake tracking, and weight monitoring, all aimed at promoting healthier lifestyles and facilitating users’ progress toward their fitness goals.

Key features of our application include a food database that offers nutritional information such as calorie count and, a recipe repository that makes it easier for users to plan means that align with their fitness goals. Daily intake tracking is important since users can log their meals throughout the day, enabling them to monitor their calorie intake in real-time.

The benefit of using our Diet Manager is that it provides personalized recommendations based on users’ dietary preferences and health goals. Information given to the user will help them be more informed about their diet and how to get in shape as soon as possible.

# Design Overview

The project is done with a Model-View-Controller (MVC) design pattern in which we divide our application into three interconnected components, a model that manages the application’s state and responds to requests for information about the data, the view that represents the presentation layer of our application, and controller that is an intermediary between model and view. We chose this design since we separate the application logic which makes it easier to manage and modify each component independently. Additionally, it is easier to maintain the application since everything is divided into logical parts, therefore, we can add and remove new features more easily.

## **Directory Structure**

Firstly, we created a clear directory structure that will help us navigate through the project and make it easier to know what each directory contains.

## **View**

The view was implemented right after the directory structure since it represents the user interface of our application. We have decided to have four buttons and two text areas. Buttons can load the data, add food, add recipes, and add food to the logs. The first text area located on the left has foods and recipes in it, while the log is on the right side and represents what person ate at what time.

## **Model**

In the model, we have created an abstract class, csvModel, that consists of the constructor, and abstract methods that are used in the Foods and Logs classes. Foods class manages food data and provides methods for reading and writing. It interacts with the FileHandler class which is used for file input and output operations. The logs class manages log data, provides reading and writing methods and also interacts with the FileHandler class. Log class contains accessors and mutators for attributes that are used for their respective attributes. It also contains the toString method which prints out information about logs. Food class is an abstract class that is used in BasicFood class, and it contains abstract methods which will be used in BasicFood. Additionally, there is BasicFood class which extends Food class and it contains accessors and mutators for attributes, as well as, inherited methods from Food class.

## **Controller**

The controller class is responsible for handling user input, updating the model accordingly, and manipulating the view to reflect any changes in the model. Our constructor acts as an intermediary between the view class and foods model and the logs model. It loads the data from log CSV file within the array list and in that way, it adds it to the log area as well. From foods CSV file it reads basic foods and recipes which are also displayed in the area in the application.

**Factory**

The Factory subsystem is tasked with the creation of Food objects. It utilizes the createFood method to generate a Food object based on the data extracted from the foods.csv file. This method constructs a Food composite object and distinguishes between BasicFood and Recipe objects based on the attributes read from the file. These objects serve as composed component elements within the composite structure.

# Overall System Structure

In this class diagram, there are three main subsystems, model, view, and controller. The view is responsible for the graphical user interface and creates it. The model consists of seven classes out of which one is an abstract class. CsvModel is an abstract class with constructor and abstract methods. Foods class have the responsibility of accessing and modifying the food, as well as, reading and writing the foods into files. Similarly with log and logs classes where one has accessors and mutators, while the other has reading and writing methods. Abstract class Food has its abstract methods that are inherited into BasicFood class which has constructor for making basic foods and accessors and mutators for them. Additionally, read, and write methods are part of the csvModel abstract class that is inherited into Foods and Logs classes. The recipe class consists of ingredients and how many ingredients are needed for the recipe. The last class in the model is FileHandler which has two methods for getting a reader and writer. Additionally, there are handle methods for all buttons which make them functional and manipulate data.

Finally, the DietManagerRunner class is used for running the application which uses façade pattern to run DietManagerApplication.

# Subsystems

## **Subsystem View**

## 

The view subsystem creates the user interface for the application. It interacts with the controller class and handler classes since the controller needs a view to function. It uses model classes since it needs to display data from foods and logs. And when the data is changed in the Foods or Logs classes, it needs to be visible in the user interface. Data is changed through the user’s click on a button which functions through the handle methods that are the part of the controller.

## **Subsystem Model**

## A diagram of a computer program Description automatically generated with medium confidence

The model subsystem manages the application’s state and responds to requests for information about the data. In our application, it reads values from the CSV file and along with the view, the food and recipe are presented to the user. It also interacts with the controller since it needs a model to function. Methods from the abstract class, csvModel, can be inherited which is in the Foods and Logs classes. Logs class uses the Log class to access the logs. The Logs are visible in the text areas which are part of View and therefore need to interact with them for our application to be accurate. Abstract class Food is used in BasicFood class that inherits methods from Food class and creates new instance of food object.

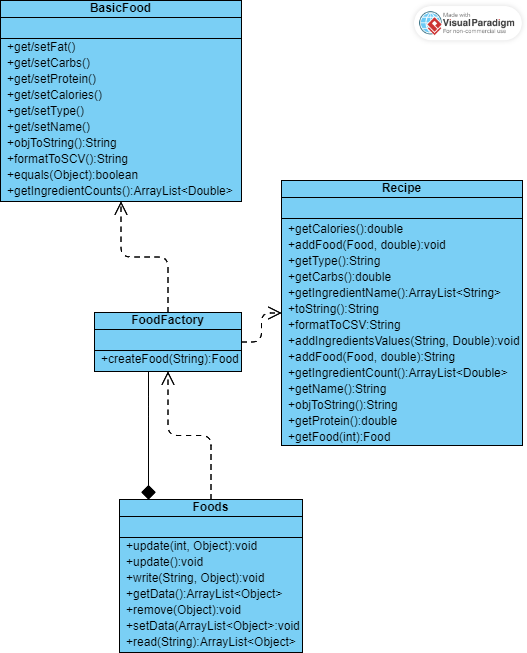
**Subsystem Controller**

**A diagram of a computer

Description automatically generated**

The controller subsystem consists of Controller class and all handle classes that make the buttons from view work. View and csvModel are used to create the Controller object which is used as an intermediary for model and view. It cannot exist without those classes. It is responsible for handling user input, updating the model accordingly, and manipulating the view to reflect any changes in the model. Handle classes use View’s buttons and csvModel to make buttons work and manipulate data displayed in text area.

**Subsystem Factory**

****

The Factory subsystem is responsible for creating Food, it creates a Food object based on what is read from the foods.csv file using the createFood method, the method creates a Food which is a composite and then based on the attributes that are read from the file it differentiates BasicFood and Recipe objects which are composed component objects.

# Sequence Diagrams

**Sequence Diagram 1 – Loading date for 1 basic food (Pizza Slice) and 1 recipe (PB+J sandwich)**

Once we run the app the DietManagerApplication will be called which has inside start method to start the app.

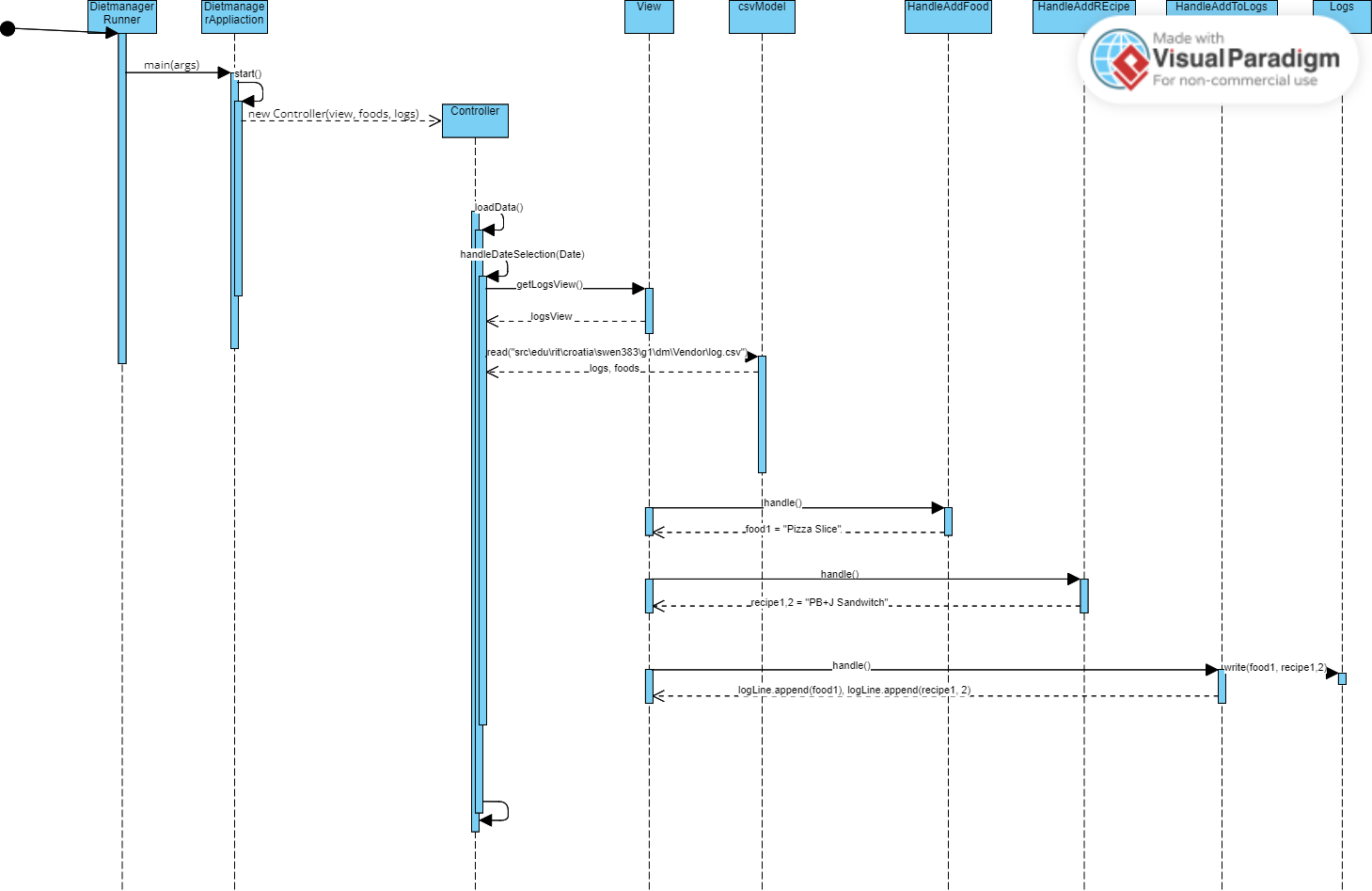
Inside the start method, the controller is initialized and inside the controller, the data will be loaded. In the controller is the load data method which is responsible for loading data. Inside this method, the read method from Foods(subclass of csvModel) will be called to read each specific line in the food.csv file. For each line read, the food factory will have to deal with two cases(create Recipe or Basic food) based on how data is structured in the file. In the case when the line loaded is a recipe then it goes through each food loaded and adds specific foods like ingredients in the specific recipe. Then all data loaded will be stored in a list and display to the app.

## 

**Sequence Diagram 2 – Add one serving of the Pizza Slice & 2 servings of the PB+J Sandwich to the log entry for the current date**

Once we run the app the DietManagerApplication will be called which has an inside start method to start the app.

Inside the start method, the controller is initialized and inside the controller, the data will be loaded. In the controller is the load data method which is responsible for loading data. The data was loaded inside handleDateSelection where it was loaded for a certain date. Then through handle method from handle method of HandleAddFood class, the user gets food1 which is “Pizza slice”. After, in handle method of HandleAddRecipe, user gets recipe 1 and recipe 2 which are two servings of PB+J sandwich. After all that, through handle method of HandleAddToLogs class it appends the log and finally with write method it writes those servings into Logs.



**Sequence Diagram 3 – Compute the total number of calories for the current date**

Once we run the app the DietManagerApplication will be called which has an inside start method to start the app.

Inside the start method, the controller is initialized and inside the controller, the data will be loaded. In the controller is the load data method which is responsible for loading data. The data was loaded inside handleDateSelection where it was loaded for a certain date. Then in the constructor, it computes the total number of calories for certain date by calling calculateTotalNutrientForDate method which calls get method from View and reads CSV model to get logs and foods from the file path given as the method parameter. Then the controller gets the date, food name, carbs calories, fat, and protein from Log class, and it gets name and servings from Food class. After getting all the data, nutrients are calculated, and sets a text to the text fields that are retrieved from the View.

A screenshot of a computer program

Description automatically generated