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 and Food classes both contain accessors and mutators for attributes that are used for their respective methods. They also contain the toString method which prints out information about food and logs.

## **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. In the handle method, it checks the source of the event and then performs actions accordingly.

# Overall System Structure

A screenshot of a computer

Description automatically generatedIn 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. Food and foods classes 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. 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. Finally, the DietManagerRunner class is used for running the application.

# Subsystems

## **Subsystem View**

## **A screenshot of a computer Description automatically generated**

The view subsystem creates the user interface for the application. It interacts with the controller class 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.

## **Subsystem Model**

A screenshot of a computer

Description automatically generatedThe 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. The foods class uses the Food class to access the food, and the Logs class uses the Log class to access the logs. The Logs and Foods are visible in the text areas which are part of View and therefore need to interact with them for our application to be accurate.

A screenshot of a computer program

Description automatically generated**Subsystem Controller**

The controller subsystem consists of only three classes, which are Controller, View, and csvModel since 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. In its handle method, it checks the source of the event and then performs actions accordingly.

# Sequence Diagrams

## **Sequence Diagram 1 – Read a food database consisting of three basic food**

## **A diagram of a food system Description automatically generated**

The user interacts with the view by clicking the specific button to load the data from the food database. Once the button is clicked then the controller will take care of action to read data from the food.csv file(model) which has inside different information about various foods. Also, some data can be read from the log.csv file if needed. Food data are stored in an ArrayList of objects. After the data are read then the controller will return to view the specific data requested and the user will be able to see them in the UI.

## **Sequence Diagram 2 – Add two servings of basic food to the log entry for the current date**

A diagram of a food entry

Description automatically generated

In this case, the user will interact with the view by clicking a specific button to add data to the log file. The controller will check if the object is food and then will write specific data to the log file(model) and then the updated data will be read and shown to the view where the user can see them.

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

## **A diagram of a diagram Description automatically generated**

In this case, the user will interact with the view by clicking a specific button to calculate total calories. The controller firstly will read data from the log.csv file and then calculate the total number of calories based on the data read. After finishing the calculation the controller will return to view the specific value calculated which will be shown to the user.