Final Design Document

Diet Manager – Version 2.0

Desa Brnada, Luka Dragicevic, Niko Madesko

Dubrovnik, April 2018.

Introduction

1.1 Version 1.0 Overview

Provided by the RIT myCourses SWEN 383 Diet Manager V2.0 Specification

There is a wide-variety of programs available to help people monitor their diets. Such programs provide a collection of basic foods, along with their dietary information (calories, and grams of fat, carbohydrates, and protein). This basic collection can be extended by users, who can add their own basic foods, as well as recipes consisting of basic foods and sub-recipes.

Each day the user selects the foods in the collection that he or she consumed during the day, including the number of servings of each food. From this information the program can display the totals for each of the distinct pieces of dietary information, as well as various graphics showing the composition of the day's diet in terms of fat, carbs, and protein.

Users can also periodically record their weight, and the program can use this to show the weight change over time. Users can also set a desired caloric intake, and the program will indicate the amount by which the user is under or over this goal.

The program, of course, must save the collection of foods, both basic and extended, as well as the information on daily consumption, calorie targets, and recorded weights.

1.2 Version 2.0 Overview

There is a wide-variety of programs available to help people monitor their diets. Such programs provide a collection of basic foods, along with their dietary information (calories, and grams of fat, carbohydrates, and protein). This basic collection can be extended by users, who can add their own basic foods, as well as recipes consisting of basic foods and sub-recipes.

Typically, these programs also provide a way to record exercises, and the calories expended on exercise are deducted from the calories consumed to compute net calories. The user can add to or edit the collection of exercises is provided.

Each day the user selects the foods in the collection that he or she consumed during the day, including the number of servings of each food, as well as the exercises performed. From this information the program can display the totals for each of the distinct pieces of dietary information, as well as various graphics showing the composition of the day's diet in terms of fat, carbs, and protein.

Users can also periodically record their weight, and the program can use this to show the weight change over time. Users can also set a desired caloric intake. The program will display the calorie goal, the calories consumed, the calories expended via exercise, the net calories, and the difference between the goal and net calories.

The program, of course, must be able to save the collection of foods, both basic and extended, the collection of exercises, and the information on daily consumption, daily exercises, calorie targets, and recorded weights.

1.3 Resources

To create this document and application the following resources were used

1. SWEN 383 Lectures at RIT myCourses
2. https://docs.oracle.com/javase/8/docs/
3. https://www.jetbrains.com/idea/- IntelliJ IDEA IDE
4. https://www.formdev.com/jformdesigner/doc/ides/intellij-idea/ - JformDesigner6

1.4 Final Design Document Overview

To provide as much useful information as possible the document is organized into parts so that it is easy to read and find valuable information. Each segment covers an important part of description of the project, such as some background information, resources used, problems solved, functionalities that were implemented etc.

2. Project Requirements Table

Basic Requirement	Description
File: foods.csv	Maintain a food file named exactly foods.csv with the food collection in CSV (Comma Separated Values) format.
b,*name*,*calories*,*fat*,*carb*,*protein*	For a basic food, the CSV line shall be formatted like this
r,*name*,*f1name*,*f1count*,*f2name*,	For a recipe, the CSV line shall be formatted like this
Unique names for food names	
Appropriate internal data structure	The files shall be loaded into an appropriate internal data structure when the program executes and saved on exit or when the user explicitly requests the information be saved.
No duplicate information	each basic food or recipe occurs exactly once
File: log.csv	The program shall maintain a CSV format log file named exactly log.csv of the food intake, weight, and desired calorie limit daily.
yyyy,*mm*,*dd*,**w**,*weight*	Date format the Weight is recorded on a line
yyyy,*mm*,*dd*,**c**,*calories*	The way calorie limit is recorded on a line
yyyy,*mm*,*dd*,**f**,*name*,*count*	The way each food item consumed on a given day is recorded on a line
yyyy,*mm*,*dd*,**e**,*name*,*minutes*	The format Each exercise performed on a given day is recorded
GUI	The program shall employ a simple user interface sufficient to achieve the functionality
Start Requirements	When started, the program shall load the food and log data into their internal data structures.

2.1 Design Constraints

The project is required to be built using the following patterns:

1. *Composite Pattern*

The pattern is used to develop the Foods and Recipes requirements. Since they are logically interconnected it was the best possible part to implement the pattern.

The pattern is used trough the Food Interface, which is used by the BasicFood and Recipe classes. It was implemented so that the Recipe class can use objects from BasicFood class and the Recipe class itself.

The user isn't bothered with the hierarchy of the classes, so he can ignore the differences between single and composite objects. It functions so that he can add a single BasicFood object (i.e. Hot Dog Bun) or a composite recipe. The composite pattern works greatly here because the user won't add the complete list of basic foods when adding improved recipe, he simply adds a recipe with added basic foods – i.e. basic Hotdog recipe and Hotdog with added onion.

In conclusion BasicFood acts as part *objects* while the Recipe acts as *whole objects.* Or in other words, trough the *Food Component interface* a hierarchy between BasicFoods as *Leaf objects* and Recipes as Composite *objects* is implemented.

2. *MVC – Model, View, Controller Pattern*

The project requires implementation of the Model, View, Controller structure.

2.3 Model Description

Part of the application that hold the business logic and the state of the application – the state of the data the application uses. It also defines the structure and contains the data. It can update the data and provides the data if the Controller calls for the data.

We structured the model so that it performs three basic functions. Simplified, it needs to get the data form the provided CSV files. The model will update the controller if the data in the files is changed. The model also writes the data.

Next step is to separate the concerns, so that logically similar functionalities Foods and Recipes are connected using the composite pattern while the Exercises have a different logic.

LogDay is the base of the Model. It holds the FoodCount, ExerciseCount collections, Date objects and all other important data.

FoodCount is essentially a replacement for a HashMap or a payer object. It holds a Food object (BasicFood/Recipe) and how many portions of that food is consumed. ExerciseCount is very similar, holding the Exercise object and a double indicating minutes exercised.

Also, the model updates the controller with changes in the data, as well it writes the data into CSV with the CSVWriter class.

2.4 Controller Description

Since we used the JformDesigne v.6 the controller is tied to the view. Still it operates and has all the logic as a separate Conntroler. Essentially it is the Listener. The point was to not overcomplicate the application since this is a single user application.

If there was a requirement need to serve more users a controller separate from the view would be better choice. For example, if more than one user connects to use the app at the same time, or if it used by different user profiles, or if it is used by various levels of users – like administrators and basic users.

Since we are serving just one user at the time, we simplified the design.

Controller manages all interaction between the View and the model. It handles all the method calls provided by the user over the view and proceeds the to the model. Model responds with updating the controller with necessary data or saving the provided data to the file.

As is required for reusability, the View doesn't know about the Model. The View doesn't hold a reference to the controller either. However, the View sends button actions to the controller, so the button must be given a reference to the controller.

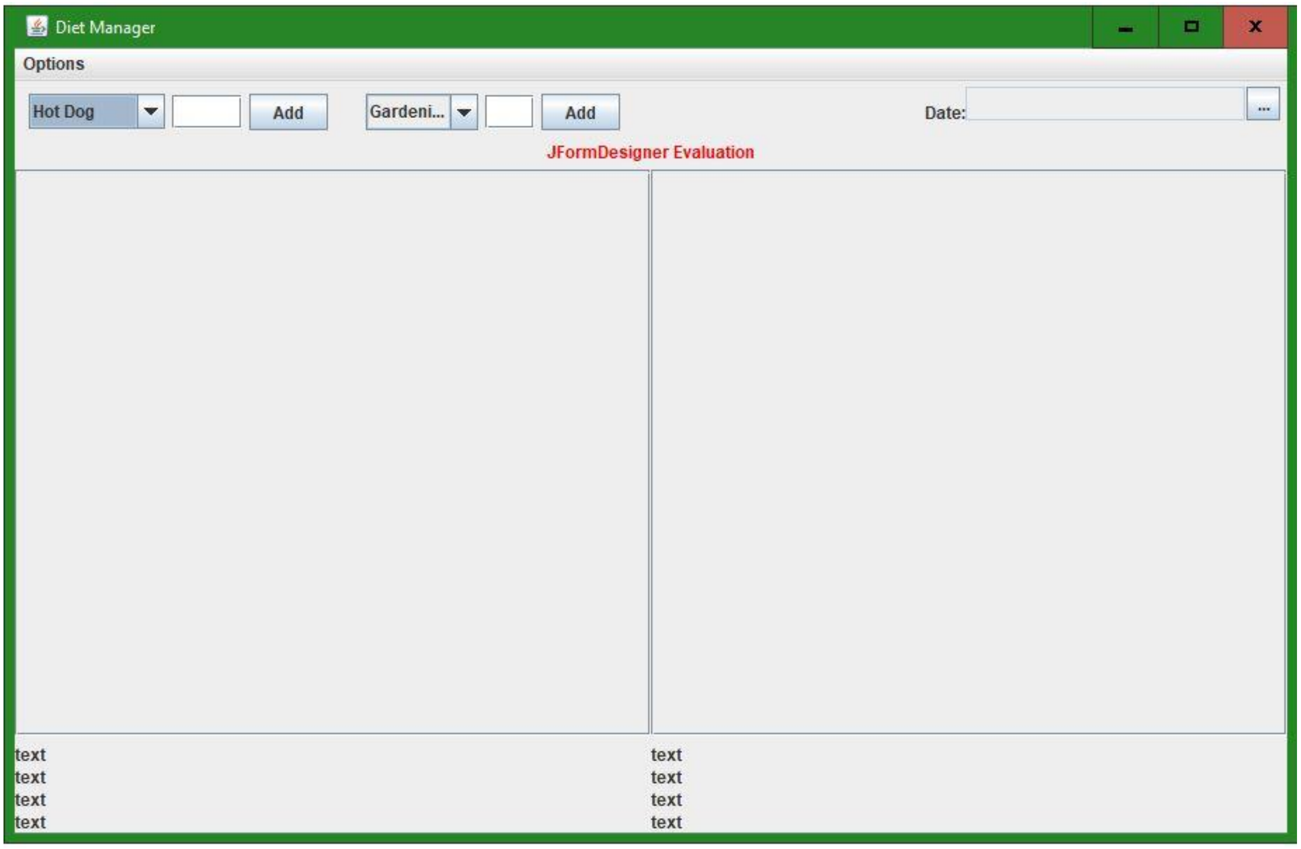
Controller has the following routine functionality:

- listeners
- methods to initialize the Model.
- methods to get references to Model and View which are run by the class ObjectGetter which instantiates the classes and passes references to the classes which need them. It then runs the initialization code.

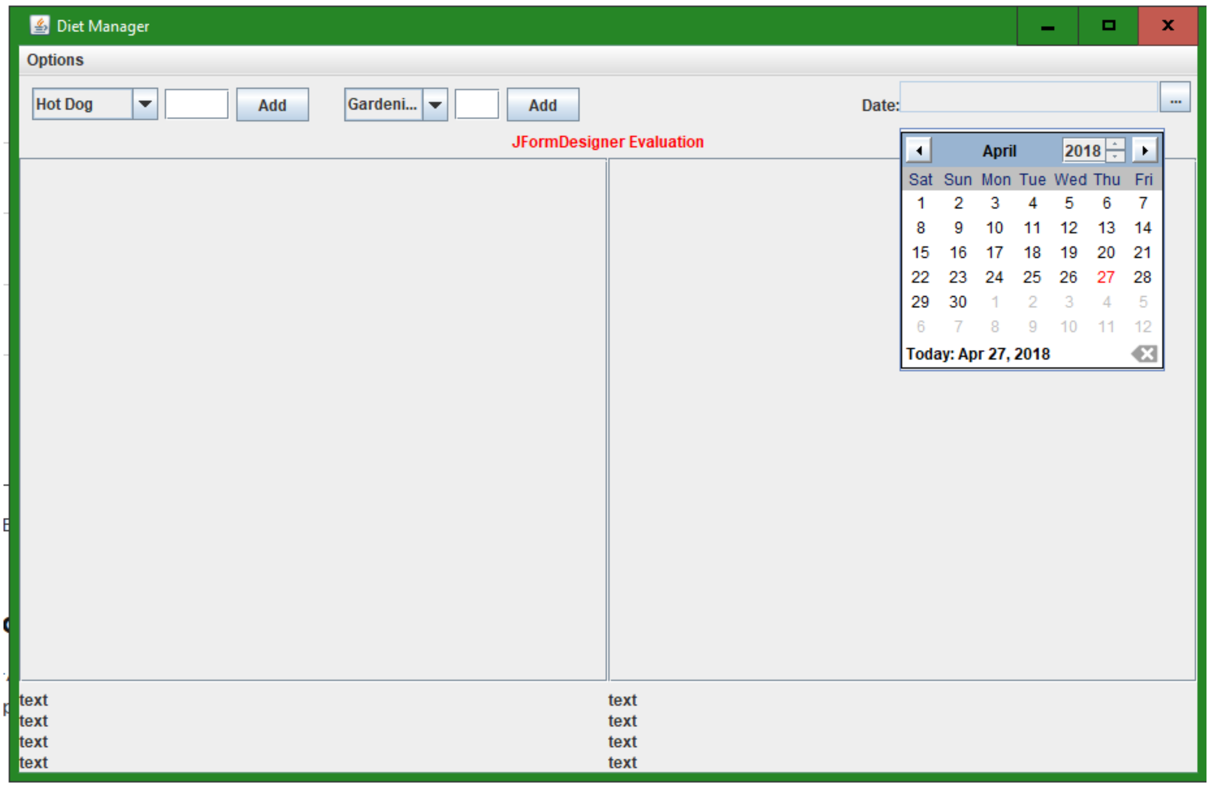
2.5 View Description

Our View is the main GUI for the Application. It was designed using JformDesigner.

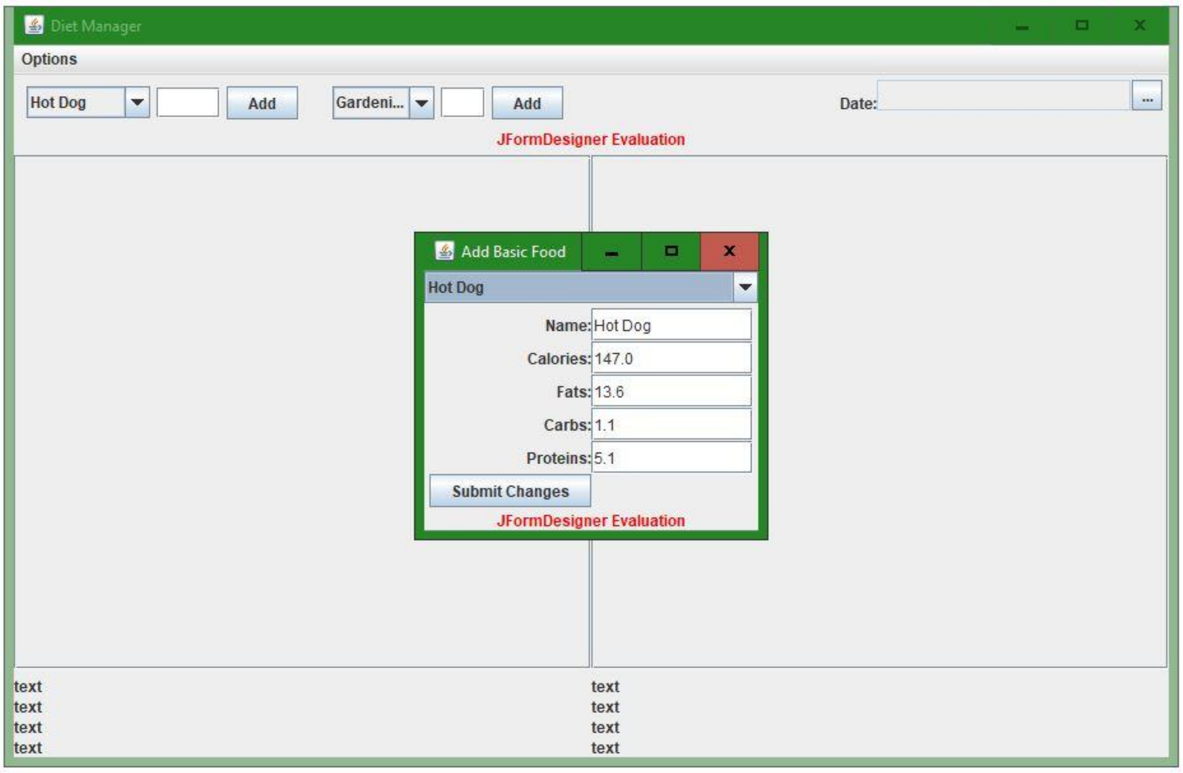
1. Start of the application:



2. Date Picker

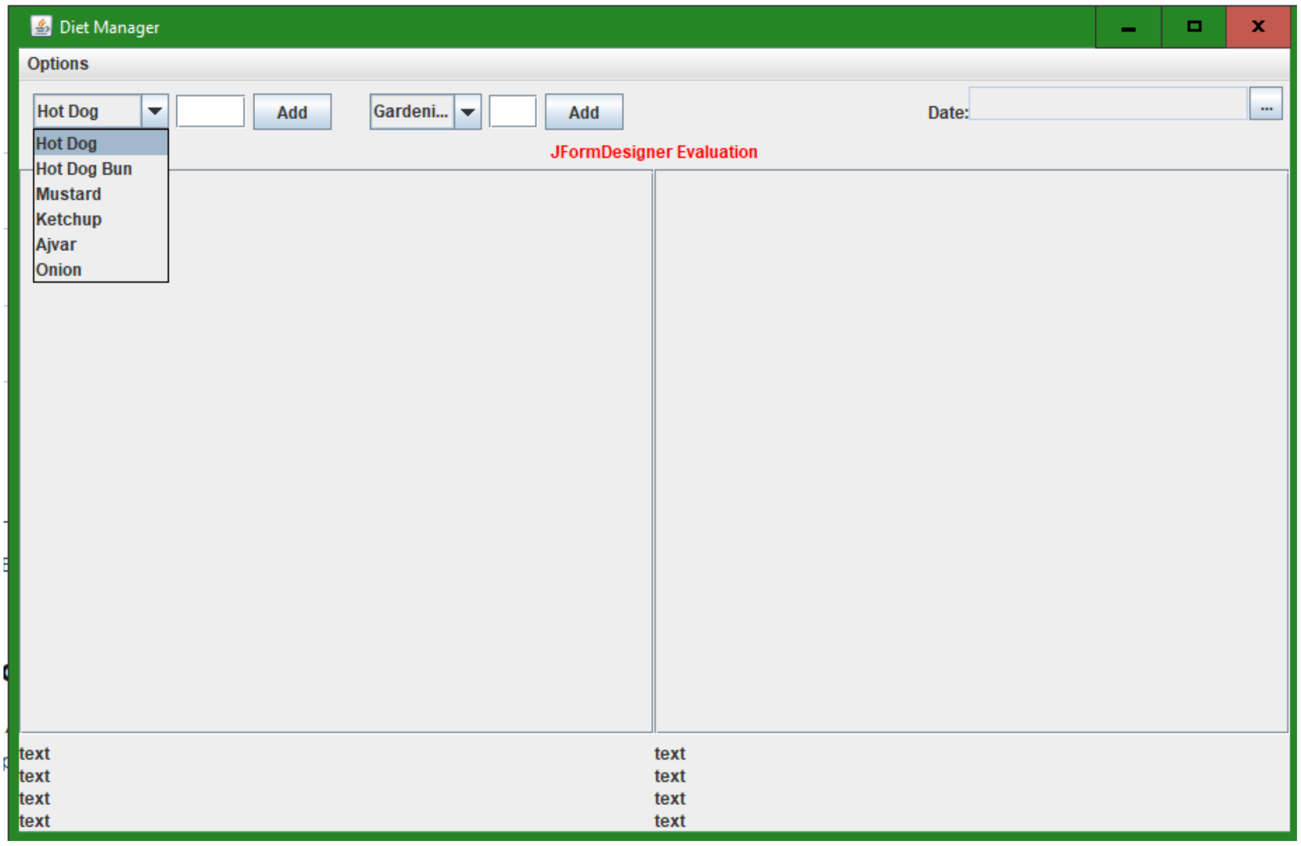


3. Edit Food

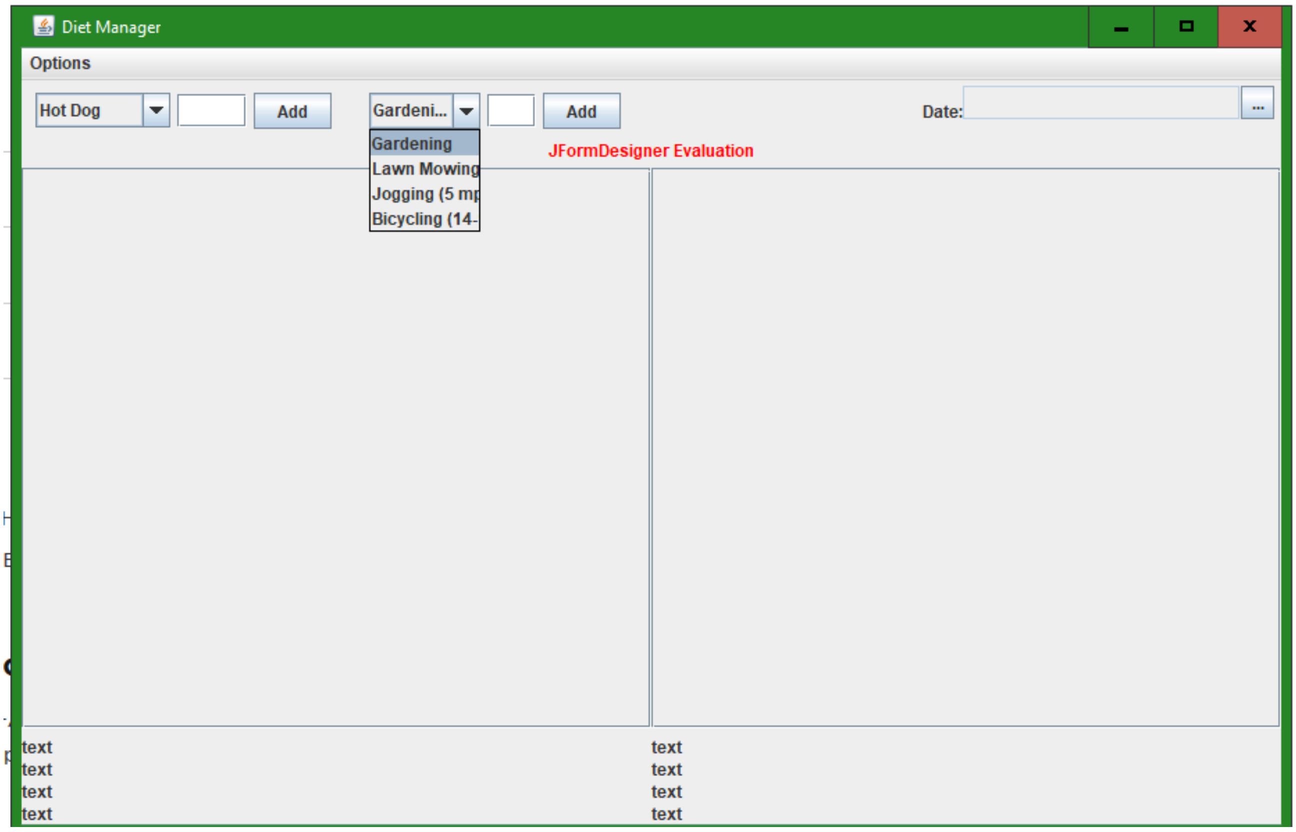


Name: Hot Dog
Calories: 147.0
Fats: 13.6
Carbs: 1.1
Proteins: 5.1

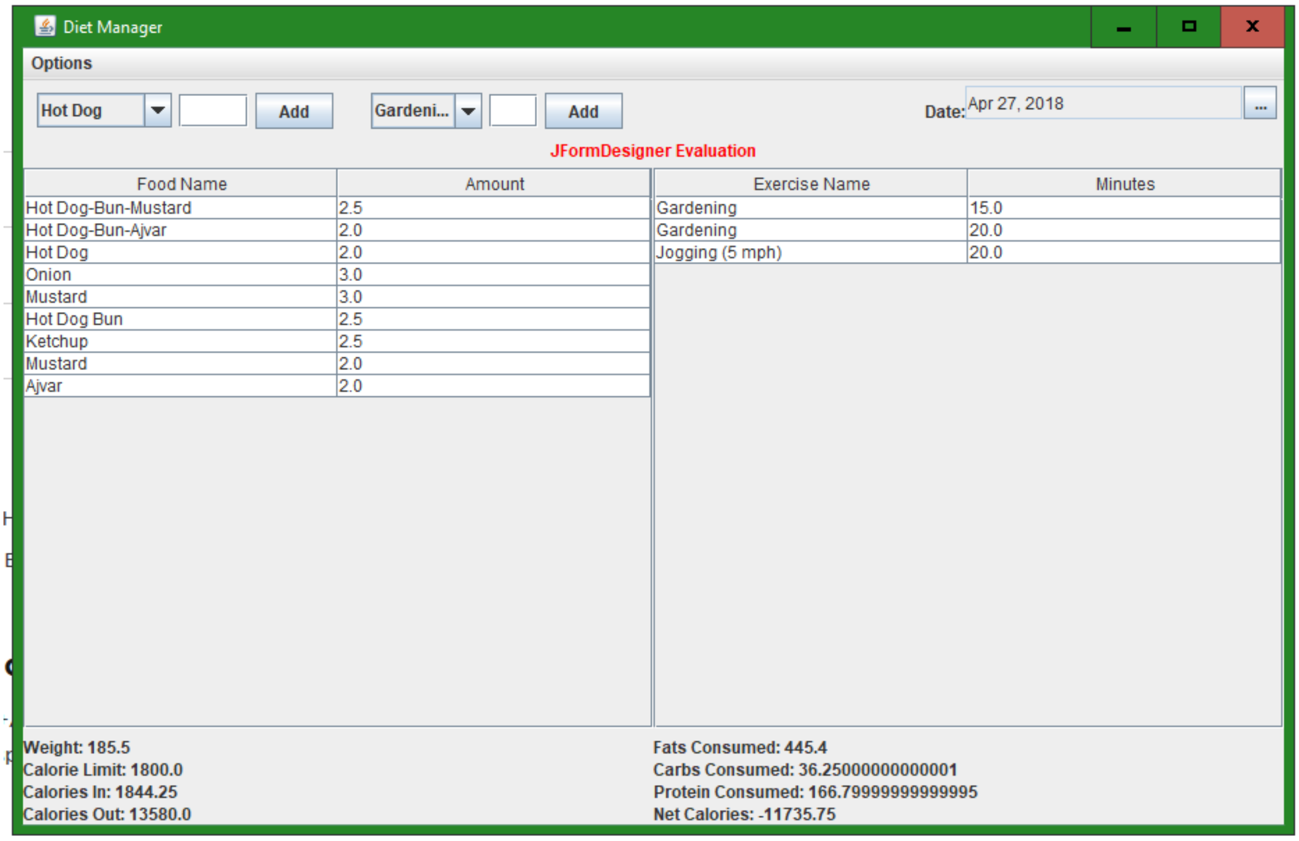
4. Choose Food



5. Choose Exercise



6. Main Application Window with all information



Diet Manager

Options

Hot Dog | Add | Gardeni... | Add | Date: Apr 27, 2018

JFormDesigner Evaluation

Food Name	Amount
Hot Dog-Bun-Mustard	2.5
Hot Dog-Bun-Ajvar	2.0
Hot Dog	2.0
Onion	3.0
Mustard	3.0
Hot Dog Bun	2.5
Ketchup	2.5
Mustard	2.0
Ajvar	2.0

Exercise Name	Minutes
Gardening	15.0
Gardening	20.0
Jogging (5 mph)	20.0

Weight: 185.5
Calorie Limit: 1800.0
Calories In: 1844.25
Calories Out: 13580.0

Fats Consumed: 445.4
Carbs Consumed: 36.25000000000001
Protein Consumed: 166.79999999999995
Net Calories: -11735.75

3. Program Operation Table

Operation	Implemented (Y/N)	Comments
The program shall employ a simple user interface sufficient to achieve the functionality given	Y	Implemented
When started, the program shall load the food, exercise, and log data into their internal data structures.	Y	Implemented
The program shall allow the user to select a specific date on which to log activities (the default on start-up is the current date).	Y	Implemented
The food intake is empty, and no exercises are recorded. The calorie limit and weight are determined using the rules from the Daily Log section.	Y	Implemented
The program shall allow the user to change the daily calorie limit and weight for the selected date	Y	Implemented
The program shall support adding of new basic foods and recipes in a manner consistent with the specifications	Y	Implemented
The program shall support adding of new exercises in a manner consistent with the specifications	Y	Implemented

Each food item consumed, along with the number of servings and total calories.	Y	Implemented
The total calories consumed for the day.	Y	Implemented
The total calories expended in exercise for the day.	Y	Implemented
The net calories (consumed - expended) for the day (this may be negative).	Y	Implemented
The weight for the day (the changing of which requires recalculation of exercise calories).	Y	Implemented
Bar Chart	N	Time Constraint
The program shall allow the user to add a food item and the number of servings to the daily log on the currently selected date.	Y	Implemented
The program shall allow the user to add an exercise and number of minutes to the log for the currently selected date.	Y	Implemented
The program shall allow the user to delete food items from the daily log. The user shall be able to unambiguously specify which food to delete even if several entries have the same food name.	N	Not implemented due to bad management of .CSV files

The program shall allow the user to delete an exercise item from the daily log. The user shall be able to unambiguously specify which exercise to delete even if several entries have the same name.	N	Not implemented due to bad management of .CSV files
All displays of information about the selected date shall immediately reflect the results of additions, changes, and deletions to foods, weight, or calorie limit. This includes the Bar Graph.	Partially	No Bar Graph Implemented
The program shall allow the user to edit the number of calories per hour per 100 pounds for any exercise (the effects of such changes are immediately displayed if the exercise is in the daily log).	N	Not implemented
The program shall provide a means for the user to save the current system state (the food collection, exercise collection, and the daily log). The saved food collection must be in a form consistent with the no forward references constraint in the Food section.	Y	Implemented

Final Remarks and Conclusion

During the development phase we had huge issues with the requirement that all data should be handled with the CSV files. There were some issues with the documentation since some technical solutions and possible upgrades/functionalities are harder to describe than to implement.

User interface was left with just the basic design from the JformDesigner. There was a plan to implement ModernUI design, again since CSV data is hard to handle (more complicated that any database) this was put aside.

Another functionality that was scraped was the multiple user functionality. We planned to have a registration form, but it was scrapped since handling multiple users without a database is a hassle and a security issue. Also, in that format it would seem like totally unprofessional coding practice. For that reason, implementing more patterns was omitted too.

Due to the other engagements the team had – all team members work full time; the development plan was scheduled carefully so that the main requirements are properly addressed. Minor functionalities were left for and when the bulk of work was done. Some things were left unimplemented due to other obligations and issues with handling the .CSV files.

In conclusion, although there is plenty of space for updates and additional functionalities, we think we achieved satisfactory level of product completeness.