COMP5541 - Requirements Documentation

Amirali Ashraf, Bo Jin, Nicholas LaMothe, Kyle Taylor Lange, Pavlo Ruban

January 2020 - April 2020

1 Project Overview

The client, Dr. Nora Houri, has asked our team to construct a Dietary Manager Application. The requirements for the project are simple and fairly specific, although there are some ambiguities that will need to be addressed.

We have been asked to build an application that will distinguish between meals that are eaten at home and meals that are eaten when not at home. These have been specified as *Indining* and *Outdining* , respectively.

For the Indining foods consumed, we have been asked to construct these modules in terms of Name, Time, Serving, and Type.

For the Outdining foods consumer, we have been asked to construct these modules in terms of Retailer, Time, Meal, and Group.

2 Requirements: Increment I - Roles

The majority of the current functionality has been implemented by Parham (Amirali) and Paul, and this includes the majority of the work done for the view (GUI) portion of the application.

Bo Jin has been involved in working on the graphical user interface, as well as some code editing, specifically regarding input validation.

Nicholas and Kyle have taken on the roles of producing documentation and testing the code, although it was Paul who produced the class diagram.

3 Requirements: Increment I - Basics

For the first increment, we have the task of adding some specific functionalities.

The program must allow the user to view all of the consumed foods in the form of a list. The order here is to be arbitrary.

It must allow the user to add or remove a particular food from the list. Here, she has used both the term 'foodstuff' and 'food serving', which, until further notice, we will consider as interchangeable.

In order to facilitate the view, the application must use a Java Swing User Interface.

Furthermore, we must plan for the future addition of food types, which I will take to mean 'general' classes of food (dairy, meat, vegetables, etc.). We must also plan for extending support into multiple user interfaces, as well as varied ways of presenting the dietary list (e.g. graphically as a pie chart)

For the purposes of having more functionality in planning the diet, we have collectively decided to have *all* of the following properties be considered separately, and will all factor into what is added from the *Diet Plan* (UI) to the *List of Consumed Foods* (UI) - again, note that this is a high-level discussion of user functionality just to offer an overall view:

- Food Name
- Meal Type
- Amount
- Calories
- Location
- Food Group
- Date (including Time)

The following section will begin with a brief explanation for the attribution of *Location* to each food item that is to be incorporated into the diet plan.

4 Requirements: Increment I - Planned Functionality

A difficult decision for our team was how we were to approach the issue of the separation of the *Indining* and *Outdining* portions of the program in a reasonable fashion. The decision to include Location factors into this, and will ultimately allow for more flexibility with future increments.

In particular, this decision will allow us to change the *List of Consumed Foods* (called **EatenMealPanel**) into **two** separate user interfaces, one of which will be for *Indining* meals and the other of which will be for *Outdining* meals. The Foods will be added to either panel according to two simple conditions:

- *If* the Location is set to Home, *then* the Food will be transferred to the Indining Panel
- *If* the Location is **not** set to Home (i.e. is anything else), *then* the Food will be transferred to the Outdining Panel

This above modification will take place within **Increment 2**. Additionally, we will add an option that allows to check any food as *consumed* or *unconsumed*, although this will require a re-structuring of the general display of these panels, as there is no purpose in referring to them as *consumed food lists* if we're to provide the option to check them as *consumed* or *unconsumed*.

It will also be necessary to provide the user the option to hide or un-hide the consumed diet. In brief, it will be an option that will allow the removal or re-addition of consumed foods within either *Indining* or *Outdining* panels.

Finally, **Increment 3** will require that the team design an adequate database schema to map *Indining* and *Outdining* diets. **This will need to be documented with database schemata and class diagrams**. We will implement a class that permits the opening and closing of a database connection (some of which is perhaps already implemented), and we will have to design it such that, at program start-up, it will load diets that have been saved in the database. Furthermore, the user should have the option of being able to **save** his/her diet at any point while using the program.

Although not required explicitly, we will eventually provide an additional panel that will provide basic instructions to the user.

5 Requirements: Increment I - Current State

As stated previously, Paul and Parham are credited with creating the majority of the functionality in the model, view, and database portions of the program.

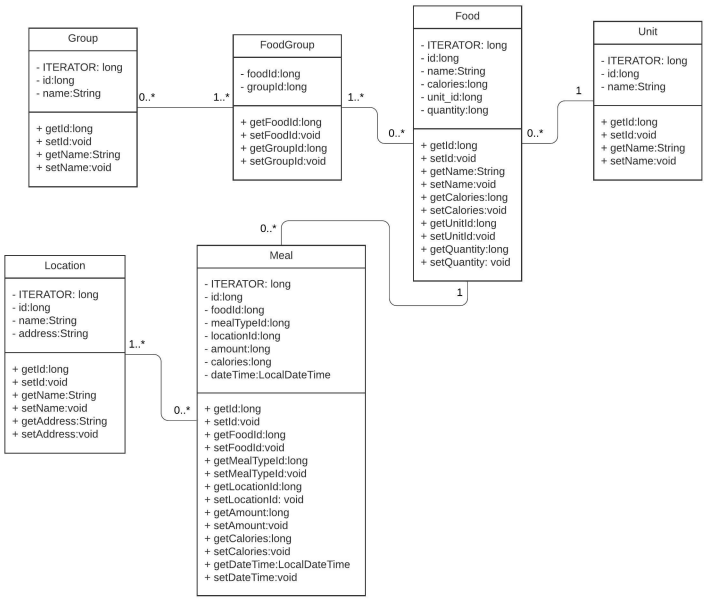
There were some original hurdles regarding whether we were to add various intermediary classes that would create 'relations' between some of the main classes (e.g. Food.java and Group.java), but ultimately these were removed or otherwise consolidated.

MODEL: The basic heart of the model of our program goes as follows. We

take for granted that all of the classes contain the appropriate accessors and mutators:

- A *Food* class, for which there are the following attributes:
  - *id, name, calories, unit_id, quantity*
- A *FoodGroup* class, for which there are the following attributes:
  - *foodId, groupId*
- A *Group* class, for which there are the following attributes:
  - *id, name*
- A *Unit* class, for which there are the following attributes:
  - *id, name*
- A *Meal* class, for which there are the following attributes:
  - *id, foodId, mealTypeId, locationId, amount, calories, dateTime*
- A *Location* class, for which there are the following attributes:
  - *id, name, address*

Below is a Class Diagram that details the functionality of our program:



VIEW: The program at this stage has a decent amount of functionality:

- A **Food** panel (FoodPanel), where the attributes *Food Name*, *Quantity*. *Unit*, and *Calories* can be added. The user has the option of deciding which **Groups** may be added, and may include categories such as *Dairy*, *Vegetables*, etc.
- A **Food Groups** panel (GroupPanel), where the user has the option of adding or removing categories of food. This can be useful for users whose diets are restricted to specific groups.
- A **Locations** panel (LocationPanel), where the user has the option of adding or removing specific locations (*i.e. !Home*) where food may be consumed.
- A **Units** panel (UnitPanel), where the user has the option of adding or removing specific units of measurement. This will unlikely be useful, but it can come in handy, depending on how much flexibility the client wishes to have.
- A **Diet Plan** panel (MealPanel), where the user can add:
  - *Meal Type* (e.g. *Breakfast*, *Lunch*, *Dinner*)
  - *Food Name* (which will automatically add foods that have been inserted in the **Food** panel)
  - *Location* (which will automatically add locations that have been inserted in the **Locations** panel)
  - *Number of Servings*, which will simply determine the number of servings of whatever given Food has been added
  - *Date*, which will specify the date on which the food was consumed
  - *Time*, which will specify the time at which the food was consumed
  - *Add to Consumed Food List*, which will take the given item from the **Diet Plan** and add it to **List of Consumed Foods** (EatenMealPanel)
- A **List of Consumed Foods** panel (EatenMealPanel), where the user can view the properties of the foods that have been consumed, and there is the option of removing a given food item
- A **Report** panel (ReportPanel), where the use can view food groups that have been automatically added or removed, as is applicable given the actual foods that have been consumed

  NOTE: EatenMealPanel *should be re-named*

Again, there have been some minor deviations from the requirements as specified, but the team believes that they will increase both the functionality of the application and the user experience. Specifically, the required specifications of *Indining* and *Outdining* modules have instead been consolidated into the 'Location' functionality, which allows the user to specify any number of locations that can then be added to the *Consumed Foods* list, which is displayed in the top right hand portion of the user interface.

As mentioned earlier in Section 3, this inclusion of Location in FoodPanel (and thus in the individual Foods taken and stored within EatenMealPanel) will still enable us to later have Foods placed into two separate panels for *Indining* and *Outdining*, respectively, so there should be no issue with that going forward.

6 Requirements: Increment I - Use Cases

Title: Insert new unit into the Units list.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Units* tab.
- The user enters a string into the *Units* box and clicks *Insert*.
- *Failure:* the *Units* box has no input. The program displays an error message to the user.
- *Success:* the unit is assigned a unique ID and added to the list of units displayed below the *Insert* button.

Title: Insert new location into the Locations list.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Locations* tab.
- The user enters a string into the *Location Name* and *Location Address* boxes and clicks *Insert*.
- *Failure:* any of the boxes has no input. The program displays an error message to the user.

- *Success:* the location is assigned a unique ID and added to the list of locations displayed below the *Insert* button.

Title: Insert new food group into the Food Groups list.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Food Groups* tab.
- The user enters a string into the *Enter New Food Group* box and clicks *Insert*.
- *Failure:* any of the boxes has no input. The program displays an error message to the user.
- *Success:* the food group is assigned a unique ID and added to the list of food groups displayed below the *Insert* button.

Title: Insert new food into the Foods list.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Foods* tab.
- The user enters a string into the *Food Names* box, a number into the *Quantity* box, selects an option from the *Unit* box, and enters a number in the *Calories* box.
- The user checks the number of food groups that the item belongs to and clicks *Insert*.
- *Failure:* either the Quantity or Unit boxes contain non-number strings, or any box has no input. The program displays an error message to the user.
- *Success:* the food is assigned a unique ID and added to the list of food items displayed below the *Insert* button.

Title: Add food to the List of Consumed Food.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Diet Plan* tab.

- The user selects an appropriate meal type from the *Meal Type* drop-down box, a food name from the *Food Name* drop-down box, and a location from the *Location* drop-down box.
- The user types a number into the *Number of Servings* box, selects a date from the *Date* box, enters a time in the *Time* boxes, and clicks *Add to Consumed Food List*.
- *Failure:* a date was not selected or a number was not input into the Number of Servings or Time box. The program displays an error message to the user.
- *Success:* the food item is added to the List of Consumed Food panel. The calories consumed are calculated and displayed next to the food.
- The Report Panel updates to display the total number of calories consumed in the day, and the food groups eaten and not eaten are updated to reflect which (if any) food groups have been eaten.

Title: Delete unit from Units tab.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Units* tab.
- The user selects the unit they wish to delete from the list.
- The user clicks the *Delete* button.
- *Success:* the unit is deleted and the user is notified via popup message.

Title: Delete location from Locations tab.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Locations* tab.
- The user selects the location they wish to delete from the list.
- The user clicks the *Delete* button.
- *Success:* the location is deleted and the user is notified via popup message.

Title: Delete food group from Food Groups tab.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Food Groups* tab.
- The user selects the food group they wish to delete from the list.
- The user clicks the *Delete* button.
- *Success:* the food group is deleted and the user is notified via popup message.

Title: Delete food from Food tab.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the *Foods* tab.
- The user selects the food they wish to delete from the list.
- The user clicks the *Delete* button.
- *Success:* the food is deleted and the user is notified via popup message.

Title: Delete a consumed food from the List of Consumed Food.
Primary Actor: User.
Scope: Dietary Manager Application.
Level: !
Story:

- The user clicks on the consumed food entry they wish to delete.
- The user clicks on the *Remove from Consumed List* button at the bottom of the panel.
- *Success:* the consumed food is deleted from the List of Consumed food. The calories update to reflect the removal of the food item, and any food groups under the eaten list will move to the not eaten list if the food item was the only remaining food item from that group.

7 Requirements: Increment II - Roles

In continuation of the previous increment, Parham will continue to implement a good portion of the main code, including the GUI, and in particular will provide an early implementation of most or all of the database functionality.

Paul will provide the *Design Document* for this increment, as well as notable changes to the user interface. Nicholas will continue to work on the *Requirements Document*, and will spend some time working on the user interface. Kyle will work on the GUI and do all of the unit testing. Bo Jin will work on input validation and assist with the user interface design.

8 Requirements: Increment II - Current State

In this increment, several important aspects of functionality were added, which will both reflect our goals for both Increment II as well as Increment III.

First, we collectively decided against separating **EatenMealPanel** into two separate panels that would each reflect foods that were either *Indining* or *Outdining*. Instead, Kyle started the process of adding *Filters* above the panel itself which would allow the user to toggle whether a food is set to either given meal setting. The Foods added from *Diet Plan* (**MealPanel**) to *List of Consumed Foods* (**EatenMealPanel**) would be added based upon whether they were set to *Home* or else anywhere that is **not** *Home*, and this follows our original plan of action.

Some of the naming conventions have yet to be changed, and this has been somewhat problematic, and it is difficult to know if, at this stage, changing them would make it more problematic going forward. Thus, this is an open question, although it is something that theoretically should be addressed, likely during Increment III.

As discussed, the **Food** model has been updated to reflect nutrient additions for the constructor, allowing for the addition of the following properties to any given food item:

- Fat
- Carbohydrate
- Salt
- Protein

This approach of adding the nutrients seemed to be the simplest and most elegant way of implementing them. There were discussions of whether these

properties should be considered as a separate class, but this would involve unnecessary coupling.

Parham, Paul, and Kyle made updates to the GUI that allow for the following user engagements:

- Filtering Foods by Indining, Outdining, or All
- Filtering Foods (Indining, Outdining, or All) by Time Period
- Removing Food from Diet List
- Marking Food as Consumed
- Marking Food as Not Consumed
- Hiding Consumed Food
- Hiding Not Consumed Food

The original GUI changes worked well, but not for all members of the project. Paul's updates worked more consistently for all users, as not all members in **FoodPanel** were fitting consistently for all users. Specifically, the addition of a scroll panel to the Foods tab allowed for better usability across differing users' configurations. That said, there are still quirks to work out.

The *Design Document* has been completed by Paul, which includes the necessary diagrams and information flows.

Parham has made substantial updates to **ReportPanel** that allow for a display of all of the caloric and nutritional properties attributed to the Foods displayed within the *List of Consumed Food*, and which are displayed correctly with respect to the given filters being applied. This panel, named *Calories and ingredients*, was a beneficial addition to the panels that display *Eaten food* and *Not Eaten food*. This panel displays the following information:

- Calories
- Fat
- Salt
- Protein
- Carbohydrate

Finally, all of the database functionality has been implemented by Parham, which allows for the current state of the program to be saved to the program before exiting. And changes to the database file will be appropriately re-loaded

upon the re-starting of the application.

Kyle has conducted the JUnit testing. It is important to note here that these tests will fail if all of the classes are run at once, but they will pass if they are run separately, one-by-one. Apparently the reason for this is that SQLite doesn't permit multiple connections to be run simultaneously.

9 Requirements: Increment II - Planned Functionality

Much of the functionality that is required for Increment III has already been implemented. Thus, in proceeding with the next increment, the focus will mainly be corrective or otherwise cosmetic.

As such, we need to accordingly plan what would be the best course of action in finalizing the project.

Crucially, there are several things that need to be worked out, clarified, and cleaned up:

- Naming **must** be standardized and clarified and agreed upon. This includes Class names, variable names, and written displays of UI functionality. This kind of refactoring is long overdue.
- Ideally, important aspects of the functionality should be commented to increase the clarity of the code. This is *especially* true with regard to the important dependencies that may be less obvious in the event that new developers should need to untangle the causal chain. Moreover, it simply improves the traceability/maintainability of applications over time, quite irrespective of whoever has produced the majority of the code itself.
- Unused code segments should be cleaned up, removed, or re-purposed if necessary. If they are unused and may potentially be useful in the future, then **clear** inline documentation should say *what the code is for*.
- The team ought to do as much as possible to ensure that the display works fluidly and consistently across all users and configurations. This is a final concern for the usability and consistency of our application - quite aside from whatever functionality is left to debug.

These concerns aside, the application very functions well, and goes above and beyond the requirements as specified. The .jar executable will be provided to the client, as well as instructions for execution and navigation to be provided within a README file. The *Requirements Document* and *Design Document* will also be included within a folder titled 'docs'.