COMP5541 - Test Documentation

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

January 2020 - April 2020

1 Introduction

The main purpose of the project is to develop a dietary manager application based on requirements provided by Dr. Nora Houari, further referred as the Client. The original requirements have been modified and extra requirements have been added. The development is carried out in three steps, of which the third and the final step requires a Test Document. This document will a short summary of a test plan along with test results. The test plan contains a list of test cases that will be performed and the results documented.

1.1 Purpose

The Test Document is a requirement of the Project initiated by the Client. The objectives of the Test Plan are to provide a description of the major phases of testing, identify objectives for the testing, relate test cases to the requirements of the software, outline the testing schedule and specify the format of the test results.

1.2 Scope

This document is composed during the third and final increment of the Project and it reflects the tests that have been done to ensure the quality of the software, such as correctness, performance, friendliness, maintainability verifiability. Unit testing is carried out using JUnit and the Test Document has been created with OverLeaf, an online LaTex editor.

1.3 Definitions and Abbreviations

1.3.1 Definitions

Term	Definition
Software Testing	An activity aimed at providing the software stakeholders with the quantitative and qualitative information on the software product quality under a test.
White-Box Testing	Software testing method that is targeting testing the internal software structures. The tester in this case is familiar with the system and its behaviour.
White-Box Testing	Software testing method that is targeting the external functionality, i.e. desired output on an input provided. The tester has no knowledge of the internal mechanism of the system.
Grey-Box Testing	Combination of White-Box testing and Black-box testing.

1.3.2 Abbreviations

Abbreviation	Term
UI	User Interface
DAO	Data Access Object
DB	Database
QA	Quality Assurance

1.4 Overview

In the following sections the document will contain the Test Plan that includes Test Cases reflecting the Use Cases from the Software Requirement Documentation. The results of the testing will also be presented along with testing methodology. A database schema will also be added to the document at this stage, since the

central requirement of this increment is integrating a database into the software. Conclusion will be wrap up the document and summarize the results of the increment.

2 Test Plan

2.1 Test Items

In this section the items that will endure testing are indicated. Unlike, the full-scale project this project is limited in time and requirements. Therefore, testing will be carried out to address the qualities that are in line with the software requirements for this project. These include Unit Testing and UI Testing.

2.1.1 Unit Testing

Unit tests serve the purpose of testing each component (a class and its methods) in isolation from the rest of the system. Unit testing will be performed as White Box Testing. Ideally every class and each method within the class would have to be tested. However, being limited by the time constraints of a term, we will be presenting the unit tests carried out with JUnit for the following items:

- FoodController and Food DAO
- GroupController and Group DAO
- LocationController and Location DAO
- MealController and Meal DAO
- UnitController and Unit DAO

2.1.2 UI Testing

UI testing is carried out to ensure usability and friendliness of the system. The testing is performed manually and it is considered Grey-Box testing, since the tester is familiar with the internal behaviour of the system. Test cases in this section aim at matching the expected functionality of the UI with its actual behaviour. The the following items will be tested at this stage:

- User Panel
- Diet Panel
- Report Panel

3 Unit Test

Unit testing has been performed using JUnit and the test code is in the unitTest folder of the project. The description and the results of the tests are presented in this section.

3.1 FoodController and Food DAO Testing

Testing FoodController also implies testing Food DAO. These tests can be traced back to the requirements: add a new foodstuff and remove a food item, as well delete a food item.

create()	
Test Case 1	Create a new Food item
Test Case Description	This test creates a new food item and adds it to the database. The result should an increased number of Food table entries and the created food must not be null and it should be equal to the food, added to the database.
Expected Result	The count of Food table should increase by one and when queried by id, the result should return the food added.
Actual Result	The count of Food table has increased by one and when queried by id, the result returns the food that has been added for the test.
Conclusion	Test Passed.

getAll()	
Test Case 2	Query all the entries in Food Table
Test Case Description	This test verifies that the Food Controller performs correctly the query for selecting all the items in the Food table.
Expected Result	The count of the Food items retrieved by calling FoodDao must be equal to the count of Food table entries.
Actual Result	The count of the Food items retrieved by calling FoodDao is equal to the count of Food table entries.
Conclusion	Test Passed.

delete()	
Test Case 3	Create and Delete a Food Item by ID
Test Case Description	This test deletes a Food item from the database, so the count of the Food table entries should be decreased by one and if the deleted food is queried by id, the result must be null.
Expected Result	The count of the Food table should decrease by one after an item deletion and when queried by id, the result should return null.
Actual Result	The count of the Food table has decreased by one after an item deletion and when queried by id, the result returns null.
Conclusion	Test Passed.

3.2 GroupController and Group DAO Testing

Testing GroupController also implies testing Group DAO. These tests can be traced back to the additional requirements of the system imposed by the team. Manipulating the group table allows the user to add more groups to the database. Marking Food group as eaten or not is an original requirement and by satisfying it, the system should be able to retrieve all the food groups and match the ones that are present in the diet against the ones that are not (eaten and not eaten).

create()	
Test Case 4	Create a new Food Group
Test Case Description	This test creates a new food group item and adds it to the database. The result should an increased number of FoodGroup table entries and the created group must not be null and it should be equal to the group, added to the database.
Expected Result	The count of Group table should increase by one and when searched by name, the result should match the name of the group added.
Actual Result	The count of Group table has increased by one and when searched by name, the result matches the name of the group added.
Conclusion	Test Passed.

getAll()	
Test Case 5	Query all the entries in Group Table
Test Case Description	This test verifies that the Group Controller performs correctly the query for selecting all the items in the Group table.
Expected Result	The count of the Group items retrieved by calling GroupDao must be equal to the count of Group table entries.
Actual Result	The count of the Group items retrieved by calling GroupDao is equal to the count of Group table entries.
Conclusion	Test Passed.

getGroupNames()	
Test Case 6	Create and Delete a Food Item by ID
Test Case Description	This test creates three groups and adds them to the database. Then it queries the groups to retrieve the list of group names.
Expected Result	The result should be a sting listing groups, separated by commas.
Actual Result	The result is a sting listing groups, separated by commas.
Conclusion	Test Passed.

3.3 LocationController and Location DAO Testing

Testing LocationController also implies testing Location DAO. These tests can be traced back to the additional requirements of the system imposed by the team. Manipulating the Location table allows the user to add more Locations to the database. Keeping track of the location of each meal also allows to separate Indining from Outdining diets. Adding new Locations will allow the user to be specific in identifying Outdining meals in their diet list.

create()	
Test Case 7	Create a new Location and return the Result
Test Case Description	This test creates a new Location item and adds it to the database. The result should an increased number of Location table entries and the created location must not be null. IN addition, it should be equal to the location, added to the database.
Expected Result	The count of Location table should increase by one and when searched by id, the result should match the location added.
Actual Result	The count of Location table has increased by one and when searched by id, the result matches the location added.
Conclusion	Test Passed.

getAllLocations()	
Test Case 8	Query all the entries in Location Table
Test Case Description	This test verifies that the Location Controller performs correctly the query for selecting all the items in the Location table.
Expected Result	The count of the Location items retrieved by calling LocationDao must be equal to the count of Location table entries.
Actual Result	The count of the Location items retrieved by calling LocationDao is equal to the count of Location table entries.
Conclusion	Test Passed.

delete()	
Test Case 9	Delete a Location Item by ID
Test Case Description	This test deletes a Location item from the database, so the count of the Location table entries should be decreased by one and if the deleted Location is queried by id, the result must be null.
Expected Result	The count of the Location table should decrease by one after an item deletion and when queried by id, the result should return null.
Actual Result	The count of the Location table has decreased by one after an item deletion and when queried by id, the result returns null.
Conclusion	Test Passed.

3.4 UnitController and Unit DAO Testing

Testing UnitController also implies testing Unit DAO. These tests can be traced back to the additional requirements of the system imposed by the team. Manipulating the Unit table allows the user to add more Units of measurement for foods to the database.

create()	
Test Case 10	Create a new Unit
Test Case Description	This test creates a new Unit item and adds it to the database. The result should an increased number of Unit table entries and the created Unit must not be null. In addition, it should be equal to the unit, added to the database.
Expected Result	The count of Unit table should increase by one and when searched by id, the result should match the unit added.
Actual Result	The count of Unit table has increased by one and when searched by id, the result matches the unit added.
Conclusion	Test Passed.

getAllUnits()	
Test Case 11	Query all the entries in Unit Table
Test Case Description	This test verifies that the Unit Controller performs correctly the query for selecting all the items in the Unit table.
Expected Result	The count of the Unit items retrieved by calling UnitDao must be equal to the count of Unit table entries.
Actual Result	The count of the Unit items retrieved by calling UnitDao is equal to the count of Location table entries.
Conclusion	Test Passed.

delete()	
Test Case 12	Delete a Unit Item by ID
Test Case Description	This test deletes a Unit item from the database, so the count of the Location table entries should be decreased by one and if the deleted Unit is queried by id, the result must be null.
Expected Result	The count of the Unit table should decrease by one after an item deletion and when queried by id, the result should return null.
Actual Result	The count of the Unit table has decreased by one after an item deletion and when queried by id, the result returns null.
Conclusion	Test Passed.

3.5 MealController and Meal DAO Testing

Testing MealController also implies testing Meal DAO. Testing these units is central to the functionality of the whole system as they incorporate numerous dependencies and successful testing of the unit signifies correct functionality of the related classes. The requirements to be considered while testing are the following

- View all the consumed food list (in arbitrary order).
- View diet with their hierarchy in case of composite element (i.e. Serving, Time, and food Group.
- Add any Indining or Outdining diet to the diet list.

create()	
Test Case 13	Create a new Meal (Diet)
Test Case Description	This test creates a new Meal item and adds it to the database. The result should an increased number of Meal table entries and the created Meal must not be null. In addition, it should be equal to the Meal, added to the database.
Expected Result	The count of Meal table should increase by one and when searched by id, the result should match the Meal added.
Actual Result	The count of Meal table has increased by one and when searched by id, the result matches the Meal added.
Conclusion	Test Passed.

getAllMeals()	
Test Case 14	Query all diets
Test Case Description	This test verifies that the Meal Controller performs correctly the query for selecting all the items in the Meal table. So a meal is added to the database to initiate the test.
Expected Result	The expected count of the Meals retrieved by calling MealDao must increase by one.
Actual Result	The actual count of the Meals retrieved by calling MealDao has increased by one.
Conclusion	Test Passed.

getInDiningAllMeals()	
Test Case 15	Query Indining diets
Test Case Description	This test verifies that the Meal Controller performs correctly the query for selecting items that belong to Indining from the Meal table.
Expected Result	The expected count of the Indining Meals retrieved by calling MealDao must be zero, since the Meal that has been added for the test purpose is of the Category Outdining.
Actual Result	The actual count of the Indining Meals retrieved by calling MealDao is zero.
Conclusion	Test Passed.

getOutDinigAllMeals()	
Test Case 15	Query Outdining diets
Test Case Description	This test verifies that the Meal Controller performs correctly the query for selecting items that belong to Outdining from the Meal table.
Expected Result	The expected count of the Outdining Meals retrieved by calling MealDao must be one, since the Meal that has been added for the test purpose is of the Category Outdining.
Actual Result	The actual count of the Indining Meals retrieved by calling MealDao is one.
Conclusion	Test Passed.

findById()	
Test Case 16	Find a Meal
Test Case Description	This test verifies that the Meal Controller performs correctly the query for searching a Meal from the Meal table. By retrieving a previously created Meal from the database, we verify that it indeed the one that we have added. This is one by matching the attributes of the Meal against the initial ones.
Expected Result	The Meal attributes mush be equal to the ones that were used to create it.
Actual Result	The Meal attributes are equal to the ones that were used to create it.
Conclusion	Test Passed.

delete()	
Test Case 17	Delete a Meal from diet
Test Case Description	This test verifies that the Meal Controller performs correctly the deletion of selecting item from the Meal table.
Expected Result	The expected count of the Meals retrieved by calling MealDao must be zero.
Actual Result	The actual count of the Meals retrieved by calling MealDao is zero.
Conclusion	Test Passed.

4 UI Test

The whole User Interface can be divided into 3 parts (Fig. 1). Part 1 (labeled as UI-1 in Fig. 1) is the "USER PANEL" where the user inputs almost all his food information; part 2 (labeled as UI-2 in Fig. 1) is the "DIET PLAN" where all the food items planed in a specific day is displayed, it can be filtered based on certain conditions, and the user can mark if certain food is consumed or not; part 3 (labeled as UI-3 in Fig. 1) is the "REPORT PANEL" where the total amount of each ingredient of the food listed in part 2 will be displayed in a table, and the consumed and unconsumed food groups will also be shown in two separate tables. To be able to further illustrate the test content, each of those 3 parts is subdivided into subcategory interfaces, each subcategory interface is represented by a number following its parent interface label. For example, UI-1-1 (Fig. 2) represents the "Food Groups" subcategory UI of the "USER PANEL" UI.

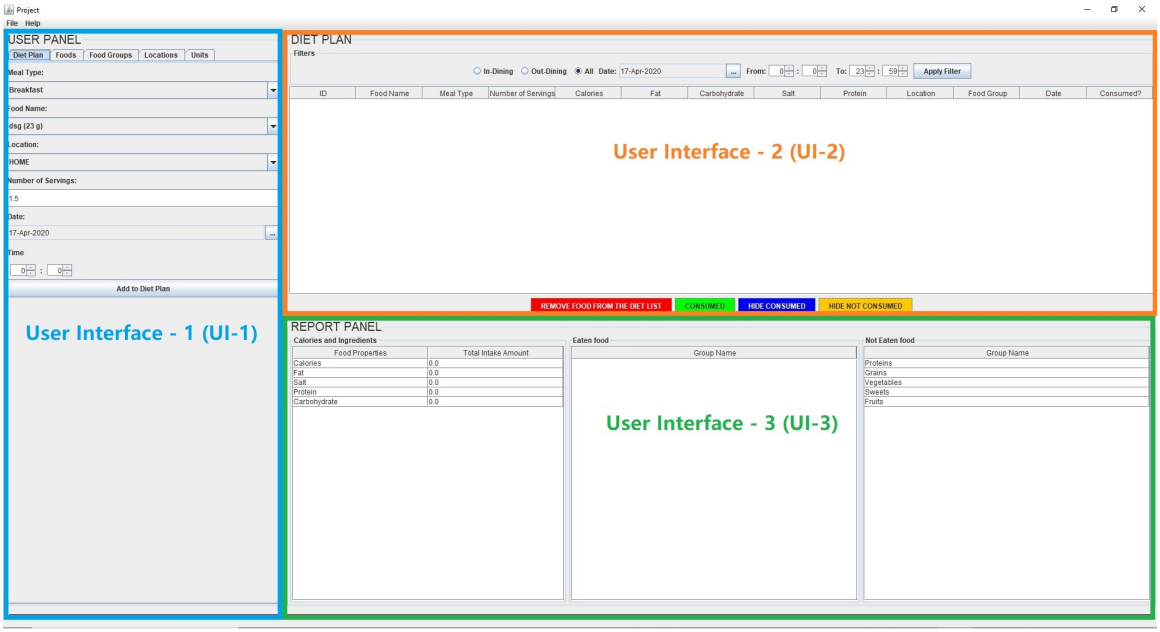

Fig. 1 - Overview of UI

Interface	Food Groups UI of USER PANEL, UI-1-1, Fig. 2
What is tested?	1. Test if the input validation method can screen out empty string input; 2. After insert a food group name, test if an integer ID is assigned to this new name; 3. After insert a food group name, or delete a selected food group name, test if the GroupObserver class can update the food group list in all the related display fields in the UI.
Inputs	Input the following names respectively, followed by clicking “Insert” icon after each input name: “”(empty string), Proteins, Grains, Vegetables, Sweets, Fruits, Oil. Highlight “Oil” in the food group table and click the red “DELETE” icon.
Expected result	“”: after click the “Insert” icon, the validation method should pop out a window to remind the user to re-input a new food group name; Other input names: after click the “Insert” icon following each input, the input new group food name with the assigned integer ID should be shown in the food group table below the input text field, they should also be shown in the “Not Eaten food” table in the “REPORT PANEL”, and the checkboxes with the corresponding name should be shown under the “Groups” category of the “Foods” tab of the “USER PANEL”. After highlight a food group name, e.g. Oil, in the food group table below the input text field and click the red “DELETE” icon, the name along with its ID should be deleted from the table, it should also be deleted from the “Not Eaten food” table, and the checkbox named after it should also be deleted.
Effective result	“”: pop out a reminder window with message “Please input the new food group name in the ” New Group” field!”. Click “OK” to close the pop out window and re-input the food group name. Other input names: the input names with the assigned integer IDs (1, Protein; 2, Grains; 3, Vegetables; 4, Sweets; 5, Fruits; 6, Oil) are listed in the food group table below the text field; the 6 food group names (Proteins, Grains, Vegetables, Sweets, Fruits, Oil) are also listed in the “Not Eaten food” table in the “REPORT PANEL”; 6 checkboxes with the corresponding food group names are added under the “Groups” category of the “Foods” tab of the “USER PANEL”. After highlight the food group name “Oil” in the food group table and click the red “DELETE” icon, the name “Oil” along with its ID “6” are deleted from the table, “Oil” is also deleted from the “Not Eaten food” table, and the checkbox named “Oil” is also deleted from the “Foods” tab.

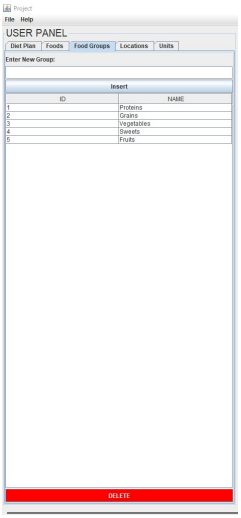

Fig. 2 - Food Groups tab in USER PANEL

Interface	Locations UI of USER PANEL, UI-1-2, Fig. 3
What is tested?	1. Test if the input validation method can screen out empty string input; 2. After insert a dining location information, test if an integer ID is assigned to this new dining location; 3. After insert a dining location information, or delete a selected dining location information, test if the LocationObserver class can update the dining place information in all the related display fields in the UI.
Inputs	Input the following pairs of location name and location address respectively, followed by clicking "Insert" icon after each input pair: ""(empty string), ML; Home, ""(empty string); Tim Hortons, ML; Starbucks, ML; Highlight "Starbucks, ML" in the location table and click the red "DELETE" icon.
Expected result	""(empty string), ML: after click the "Insert" icon, the validation method should pop out a window to remind the user to re-input a new food group name; Home, ""(empty string): should have the same expected result as above; Other input pairs: after click the "Insert" icon following each input pair, the input new location name and location address with the assigned integer ID should be shown in the location table below the input text field, the location name should also be shown in the "Location" dropdown in the "Diet Plan" tab of "USER PANEL". After highlight "Starbucks, ML" in the location table and click the red "DELETE" icon, this item along with its ID should be deleted from the table, it should also be deleted from the "Location" dropdown in the "Diet Plan" tab of "USER PANEL".
Effective result	""(empty string), ML: pop out a reminder window with message "Please input the location name in the "Location Name" field!". Click "OK" to close the pop out window and re-input the food group name. Home, ""(empty string): pop out a reminder window with message "Please input the location address in the "Location Address" field!". Click "OK" to close the pop out window and re-input the food group name. Other input pairs: the input pairs of location name and location address with the assigned integer IDs (0, Home, ML; 1, Tim Hortons, ML; 2, Starbucks, ML) are listed in the location table below the text field; the 3 location names (Home, Tim Hortons, Starbucks) are also shown in the "Location" dropdown in the "Diet Plan" tab of "USER PANEL". After highlight the "Starbucks, ML" in the location table and click the red "DELETE" icon, the item along with its ID "2" are deleted from the table, "Starbucks" is also deleted from the "Location" dropdown in the "Diet Plan" tab of "USER PANEL".

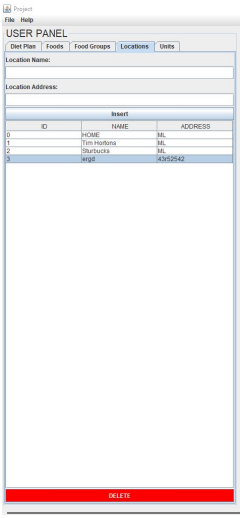

Fig. 3 - Locations tab in USER PANEL

Interface	Units UI of USER PANEL, UI-1-3, Fig. 4
What is tested?	1. Test if the input validation method can screen out empty string input; 2. After insert a unit name, test if an integer ID is assigned to this new dining unit name; 3. After insert a unit name, or delete a selected unit name, test if the UnitObserver class can update the unit name list in all the related display fields in the UI.
Inputs	Input the following unit names respectively, followed by clicking “Insert” icon after each input name: “”(empty string), g, ml, l, cup, piece. Highlight “piece” in the units table and click the red “DELETE” icon.
Expected result	“”(empty string): after click the “Insert” icon, the validation method should pop out a window to remind the user to re-input a new food group name; Other input unit names: after click the “Insert” icon following each input unit name, the input new unit name with the assigned integer ID should be shown in the units table below the input text field, the unit name should also be shown in the “Unit” dropdown in the “Foods” tab of “USER PANEL”. After highlight “piece” in the units table and click the red “DELETE” icon, this item along with its ID should be deleted from the table, it should also be deleted from the “Unit” dropdown in the “Foods” tab of “USER PANEL”.
Effective result	“”(empty string): pop out a reminder window with message “Please input the unit name in the ”Unit Name” field!”. Click “OK” to close the pop out window and re-input the food group name. Other input unit names: the input unit names with the assigned integer IDs (1, g; 2, ml; 3, l; 4, cup; 5, piece) are listed in the units table below the text field; the 5 unit names (g, ml, l, cup, piece) are also shown in the “Unit” dropdown in the “Foods” tab of “USER PANEL”. After highlight “piece” in the units table and click the red “DELETE” icon, the item along with its ID “5” are deleted from the table, “piece” is also deleted from the “Unit” dropdown in the “Foods” tab of “USER PANEL”.

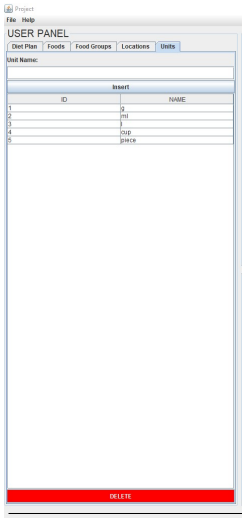

Fig. 4 - Units tab in USER PANEL

Interface	Foods UI of USER PANEL, UI-1-4, Fig. 5
What is tested?	1. Test if the input validation method can screen out empty string input or negative integer; 2. After insert a food item with all its information, test if an integer ID is assigned to this new food; 3. After insert a unit name, or delete a selected unit name, test if the FoodObserver class can update the food list in all the related display fields in the UI.
Inputs	Input the following sets of values corresponding to "Food Name", "Quantity", "Unit", "Calories", "Fat", "Carbohydrate", "Salt", "Protein", and select an item from "Group" category respectively, followed by clicking "Insert" icon after finishing input each set: ""(empty string), 10, g, 30, 10, 20, 2, 5, Grains(checkbox); Pasta, ""(empty string)(or str, 0, 1.5, -3), g, 30, 10, 20, 2, 5, Grains(checkbox); Pasta, 10, g, ""(empty string)(or str, 1.5, -3), 10, 20, 2, 5, Grains(checkbox); Pasta, 10, g, 30, ""(empty string)(or str, 1.5, -3), 20, 2, 5, Grains(checkbox); Pasta, 10, g, 30, 10, ""(empty string)(or str, 1.5, -3), 2, 5, Grains(checkbox); Pasta, 10, g, 30, 10, 20, ""(empty string)(or str, 1.5, -3), 5, Grains(checkbox); Pasta, 10, g, 30, 10, 20, 2, ""(empty string)(or str, 1.5, -3) , Grains(checkbox); Pasta, 10, g, 30, 10, 20, 2, 5, don't select any checkbox in "Group" category; Pasta, 10, g, 30, 10, 20, 2, 5, Grains and Vegetables(2 checkboxes); Bread, 2, g, 100, 0, 100, 3, 5, Grains(checkbox); Chicken, 700, g, 1000, 50, 100, 5, 300, Protein(checkbox); Highlight "chicken" in the foods table and click the red "DELETE" icon.
Expected result	""(empty string), 10, g, 30, 10, 20, 2, 5, Grains(checkbox): Pasta, ""(empty string)(or str, 0, 1.5, -3), g, 30, 10, 20, 2, 5, Grains(checkbox): Pasta, 10, g, ""(empty string)(or str, 1.5, -3), 10, 20, 2, 5, Grains(checkbox): Pasta, 10, g, 30, ""(empty string)(or str, 1.5, -3), 20, 2, 5, Grains(checkbox): Pasta, 10, g, 30, 10, ""(empty string)(or str, 1.5, -3), 2, 5, Grains(checkbox): Pasta, 10, g, 30, 10, 20, ""(empty string)(or str, 1.5, -3), 5, Grains(checkbox:); Pasta, 10, g, 30, 10, 20, 2, ""(empty string)(or str, 1.5, -3) , Grains(checkbox): Pasta, 10, g, 30, 10, 20, 2, 5, don't select any checkbox in "Group" category: For all above inputs, after click the "Insert" icon, the validation method should pop out a window to remind the user to re-input a legal value; Pasta, 10, g, 30, 10, 20, 2, 5, Grains and Vegetables (2 checkboxes): Bread, 2, g, 100, 0, 100, 3, 5, Grains (checkbox): Chicken, 1, g, 1000, 50, 100, 5, 300, Protein (checkbox): Input the above 3 sets values: after click the "Insert" icon following inputting each set of values, the new food item with the assigned integer ID should be shown in the foods table below the input text field, the food name should also be shown in the "Food Name" dropdown in the "Diet Plan" tab of "USER PANEL". After highlight "chicken" in the foods table and click the red "DELETE" icon, this food item along with its ID should be deleted from the table, it should also be deleted from the "Food Name" dropdown in the "Diet Plan" tab of "USER PANEL".

Effective result	“”(empty string), 10, g, 30, 10, 20, 2, 5, Grains(checkbox): Pop out a reminder window with message “Please input the food name in the ”Food Name” field!”. Click “OK” to close the pop out window and re-input the food name. Pasta, g, 30, 10, 20, 2, 5, Grains(checkbox): Pop out a reminder window with message “Please input a positive integer in the ”Quantity” field!”. Click “OK” to close the pop out window and re-input the quantity value. Pasta, 100, g, “”(empty string)(or str, 1.5, -3), 10, 20, 2, 5, Grains(checkbox): Pop out a reminder window with message “Please input zero or a positive integer in the ”Calories” field!”. Click “OK” to close the pop out window and re-input the calories value. Pasta, 100, g, 30, “”(empty string)(or str, 1.5, -3), 20, 2, 5, Grains(checkbox): Pop out a reminder window with message “Please input zero or a positive integer in the ”Fat (g)” field!”. Click “OK” to close the pop out window and re-input the fat value. Pasta, 100, g, 30, 10, “”(empty string)(or str, 1.5, -3), 2, 5, Grains(checkbox): Pop out a reminder window with message “Please input zero or a positive integer in the ”Carbohydrate (g)” field!”. Click “OK” to close the pop out window and re-input the carbohydrate value. Pasta, 100, g, 30, 10, 20, “”(empty string)(or str, 1.5, -3), 5, Grains(checkbox:): Pop out a reminder window with message “Please input zero or a positive integer in the ”Salt (g)” field!”. Click “OK” to close the pop out window and re-input the salt value. Pasta, 100, g, 30, 10, 20, 2, “”(empty string)(or str, 1.5, -3) , Grains(checkbox): Pop out a reminder window with message “Please input zero or a positive integer in the ”Protein (g)” field!”. Click “OK” to close the pop out window and re-input the protein value. Pasta, 100, g, 30, 10, 20, 2, 5, don’t select any checkbox in “Group” category: Pop out a reminder window with message “Please select at least one food group from the ”Food Groups” field!”. Click “OK” to close the pop out window and select at least one checkbox in the “Group” category. Pasta, 100, g, 30, 10, 20, 2, 5, Grains and Vegetables (2 checkboxes): Bread, 100, g, 100, 0, 100, 3, 5, Grains (checkbox): Chicken, 500, g, 1000, 50, 100, 5, 300, Protein (checkbox): Input the above 3 sets values: after click the “Insert” icon following inputting each set of values, the input new food items with assigned IDs (1, pasta; 2, bread; 3, chicken) are shown in the foods table below the input text field, the food names are also shown in the “Food Name” dropdown in the “Diet Plan” tab of “USER PANEL”. After highlight “chicken” in the foods table and click the red “DELETE” icon, this food item along with its ID is deleted from the table, it is also be deleted from the “Food Name” dropdown in the “Diet Plan” tab of “USER PANEL”.

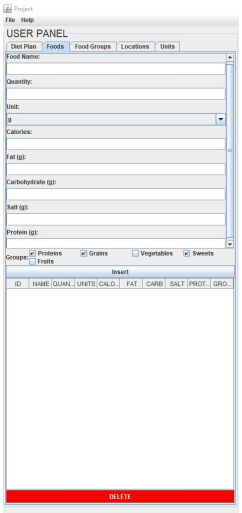

Fig. 5 - Foods tab in USER PANEL

Interface	Diet Plan UI of USER PANEL, UI-1-5, Fig. 6
What is tested?	1. Test if the input validation method can screen out non positive integer input or empty input; 2. After insert a unit name, test if an integer ID is assigned to this new diet; 3. After insert a meal, test if the MealObserver class can update the diet list in all the related display fields in the UI. 4. After insert a meal, check if all the information about this meal is correctly shown in the diet plan table.
Inputs	Input the following sets of values corresponding to “Meal Type”, “Food Name”, “Location”, “Number of Serving”, “Date”, and “Time”, followed by clicking “Add to Diet Plan” icon after finishing input each set: Breakfast, bread (100 g), Home, “”(empty string)(or str, 0, 1.5, -3), 19-Apr-2020, 7:00; Breakfast, bread (100 g), Home, 1, don’t select date, 7:00; Breakfast, bread (100 g), Home, 1, 19-Apr-2020, 7:00; Lunch, pasta (100 g), Starbucks, 1, 19-Apr-2020, 12:00; Dinner, chicken (500 g), Tim Hortons, 1, 19-Apr-2020, 19:00;
Expected result	Breakfast, bread (100 g), Home, “”(empty string)(or str, 0, 1.5, -3), 19-Apr-2020, 7:00; Breakfast, bread (100 g), Home, 1, don’t select date, 7:00; For all above inputs, after clicking the “Add to Diet Plan” icon, the validation method should pop out a window to remind the user to re-input a legal value; Breakfast, bread (100 g), Home, 1, 19-Apr-2020, 7:00; Lunch, pasta (100 g), Starbucks, 1, 19-Apr-2020, 12:00; Dinner, chicken (500 g), Tim Hortons, 1, 19-Apr-2020, 19:00; Input the above 3 sets values: after click the “Add to Diet Plan” icon following inputting each set of values, the new diet item with the assigned integer ID should be shown in the diet plan table below in the “DIET PLAN” UI. All the information about each diet should be the same as their values input in the USER PANEL UI.
Effective result	Breakfast, bread (100 g), Home, “”(empty string)(or str, 0, 1.5, -3), 19-Apr-2020, 7:00: Pop out a reminder window with message “Please input a positive integer in the ”Number of Servings” field!”. Click “OK” to close the pop out window and re-input the number of serving. Breakfast, bread (100 g), Home, 1, don’t select date, 7:00; Pop out a reminder window with message “Please input the date in the ”Date” field!”. Click “OK” to close the pop out window and select the date. Breakfast, bread (100 g), Home, 1, 19-Apr-2020, 7:00; Lunch, pasta (100 g), Starbucks, 1, 19-Apr-2020, 12:00; Dinner, chicken (500 g), Tim Hortons, 1, 19-Apr-2020, 19:00; Input the above 3 sets values: after click the “Add to Diet Plan” icon following inputting each set of values, the new diet items with assigned IDs (1, bread; 2, pasta; 3, chicken) are shown in the diet table in the “DIET PLAN” UI, and after check the values of each diet item, they are the same as those input in the “USER PANEL” UI.

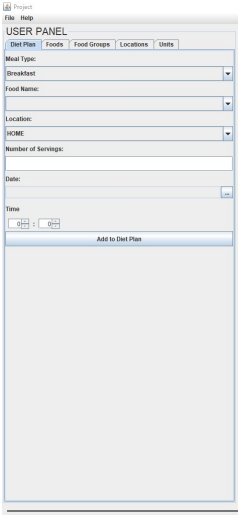

Fig. 6 - Diet Plan tab in USER PANEL

Interface	DIET PLAN UI, UI- 2, Fig. 7; REPORT PANEL UI, UI - 3, Fig. 8
What is tested?	In DIET PLAN UI: 1. Test if the radio button "In-Dining", "Out-Dining", and "All" can screen out the in-dining diet, out-ding diets, and show both in-dining and out-dining diet items in the diet plan table; 2. Test if the "CONSUMED /NOT CONSUMED" icon can mark the selected diet item in diet plan table as 'consumed" or "not consumed"; 3. Test if the blue color "HIDE CONSUMED/SHOW CONSUMED" icon can hide all diet items labeled as consumed in the diet plan table, and then resume hidden diet items; 4. Test if the orange color "HIDE NOT CONSUMED/SHOW NOT CONSUMED" icon can hide all diet items labeled as unconsumed in the diet plan table, and then resume hidden diet items; 5. Test if the red color "REMOVE FOOD FROM THE DIET LIST" icon can delete the selected diet items from the diet plan table; In PREPORT PANEL UI: 6. Check if the "Calories and Ingredients" table can show the total amount of each ingredient of all the diet items in the diet plan table; 7. Check if all the food groups contained by all the diet items in the diet plan table are shown in the "Eaten food" table, and if those food groups not contained by those diet items are shown in the "Not Eaten food" table.
Inputs	In "Diet Plan" tab of "USER PANEL", using "Add to Diet Plan" icon to add the following diet items to the diet plan table in "DIET PLAN": - Breakfast, bread (100 g), Home, 2, 19-Apr-2020, 7:00; - Lunch, pasta (100 g), Starbucks, 2, 19-Apr-2020, 12:00; - Dinner, chicken (500 g), Tim Hortons, 1, 19-Apr-2020, 19:00; In DIET PLAN UI: Click "In-Dining" radio button, then click "Apply Filter" icon; Click "Out-Dining" radio button, then click "Apply Filter" icon; Click "All" radio button, then click "Apply Filter" icon; Highlight "bread" diet item, click "CONSUMED" icon, and now the label of the icon changed to "NOT CONSUMED"; highlight "bread" diet item again, click "NOT CONSUMED" icon, now the label of the icon changed back to "CONSUMED"; highlight "bread" diet item again, click "CONSUMED"; Click "HIDE CONSUMED" icon, and now the label of the icon changed to "SHOW CONSUMED"; click "SHOW CONSUMED" icon; Click "HIDE NOT CONSUMED" icon, and now the label of the icon changed to "SHOW NOT CONSUMED"; click "SHOW NOT CONSUMED"; Highlight "chicken" diet item, click "REMVOE FOOD FROM THE DIET LIST";
Expected result	Click "In-Dining" radio button, then click "Apply Filter" icon: Only "bread. . . Home" diet item should be shown in the diet plan table. Check the ingredient values listed in the "Calories and Ingredients" table in REPORT PANEL, they should be the same as those values in "bread" diet item; check the food group listed in the "Eaten food" table in REPORT PANEL, it should contain all food groups contained by the "bread" diet item; check the food group listed in the "Not Eaten food" table, it should contain all the food groups not contained by the "bread" diet item. Click "Out-Dining" radio button, then click "Apply Filter" icon: Only "pasta. . . Starbucks" and "chicken,. . . Tim Hortons" should be shown in the diet plan table; Check the ingredient values listed in the "Calories and Ingredients" table in REPORT PANEL, they should be the sum of those values in "pasta" and "chicken" diet items; check the food group listed in the "Eaten food" table in REPORT PANEL, it should contain all food groups contained in the "pasta" and "chicken" diet items; check the food group listed in the "Not Eaten food" table, it should contain all the food groups not contained by the "pasta" and "chicken" diet items. Click "All" radio button, then click "Apply Filter" icon:

Expected result continued	All the three diet items should be shown in the diet plan table; Check the ingredient values listed in the "Calories and Ingredients" table in REPORT PANEL, they should be the sum of those values in "bread", "pasta", and "chicken" diet items; check the food group listed in the "Eaten food" table in REPORT PANEL, it should contain all food groups contained in the "bread", "pasta", and "chicken" diet items; check the food group listed in the "Not Eaten food" table, it should contain all the food groups not contained by the "bread", "pasta", and "chicken" diet items. Highlight "bread" diet item, click "CONSUMED" icon, and now the label of the icon changed to "NOT CONSUMED": The value in the "Consumed?" column from "bread" diet item should be changed from "No" to "Yes"; highlight "bread" diet item again, click "NOT CONSUMED" icon, now the label of the icon changed back to "CONSUMED": The value in the "Consumed?" column from "bread" diet item should be changed from "Yes" to "No"; highlight "bread" diet item again, click "CONSUMED": The value in the "Consumed?" column from "bread" diet item should be changed from "No" to "Yes"; Click "HIDE CONSUMED" icon, and now the label of the icon changed to "SHOW CONSUMED": The "bread" diet item should disappear from the diet plan table, only "pasta" and "chicken" diet items left; click "SHOW CONSUMED" icon: The "bread" diet item should appear again in the diet plan table. Click "HIDE NOT CONSUMED" icon, and now the label of the icon changed to "SHOW NOT CONSUMED": The "pasta" and "chicken" diet items should disappear from the diet plan table, only "bread" diet item left; click "SHOW NOT CONSUMED"; The "pasta" and "chicken" diet items should appear again in the diet plan table. Highlight "chicken" diet item, click "REMVOE FOOD FROM THE DIET LIST": The "chicken" diet item should be deleted from the diet plan table.
Effective result	Click "In-Dining" radio button, then click "Apply Filter" icon: Only "bread... Home" diet item is shown in the diet plan table, "pasta... Starbucks" and "chicken... Tim Hortons" diet items are disappeared. Check the ingredient values listed in the "Calories and Ingredients" table in REPORT PANEL, they are all the same as those values in "bread" diet item; check the food group listed in the "Eaten food" table in REPORT PANEL, it contains "Protein" and "Grains" food groups which are also contained by the "bread" diet item; check the food group listed in the "Not Eaten food" table, it contains "Vegetable", "Sweets", and "Fruits" food groups which are not contained by the "bread" diet item. Click "Out-Dining" radio button, then click "Apply Filter" icon: Only "pasta,... Starbucks" and "chicken,... Tim Hortons" are shown in the diet plan table, "bread... Home" diet item disappeared; Check the ingredient values listed in the "Calories and Ingredients" table in REPORT PANEL, they are the sum of those values in "pasta" and "chicken" diet items; check the food group listed in the "Eaten food" table in REPORT PANEL, it contains "Grains" and "Proteins" food groups which are contained in the "pasta" and "chicken" diet items; check the food group listed in the "Not Eaten food" table, it contains "Vegetables", "Sweets", and "Fruits" food groups which are not contained by the "pasta" and "chicken" diet items. Click "All" radio button, then click "Apply Filter" icon: All the three diet items are shown in the diet plan table; Check the ingredient values listed in the "Calories and Ingredients" table in REPORT PANEL, they are the sum of those values in "bread", "pasta", and "chicken" diet items; check the food group listed in the "Eaten food" table in REPORT PANEL, it contains "Proteins" and "Grains" food groups which are contained by the "bread", "pasta", and "chicken" diet items; check the food group listed in the "Not Eaten food" table, it contains "Vegetables", "Sweets", and "Fruits" food groups which are not contained by the "bread", "pasta", and "chicken" diet items. Highlight "bread" diet item, click "CONSUMED" icon, and now the label of the icon changed to "NOT CONSUMED":

Effective result continued	The value in the “Consumed?” column from “bread” diet item is changed from “No” to “Yes”; highlight “bread” diet item again, click “NOT CONSUMED” icon, now the label of the icon changed back to “CONSUMED”: The value in the “Consumed?” column from “bread” diet item is changed from “Yes” to “No”; highlight “bread” diet item again, click “CONSUMED”: The value in the “Consumed?” column from “bread” diet item is changed from “No” to “Yes”; Click “HIDE CONSUMED” icon, and now the label of the icon changed to “SHOW CONSUMED”: The “bread” diet item is disappeared from the diet plan table, only “pasta” and “chicken” diet items left; click “SHOW CONSUMED” icon: The “bread” diet item appears again in the diet plan table. Click “HIDE NOT CONSUMED” icon, and now the label of the icon changed to “SHOW NOT CONSUMED”: The “pasta” and “chicken” diet items are disappeared from the diet plan table, only “bread” diet item left; click “SHOW NOT CONSUMED”: The “pasta” and “chicken” diet items appear again in the diet plan table. Highlight “chicken” diet item, click “REMVOE FOOD FROM THE DIET LIST”: The “chicken” diet item is deleted from the diet plan table.

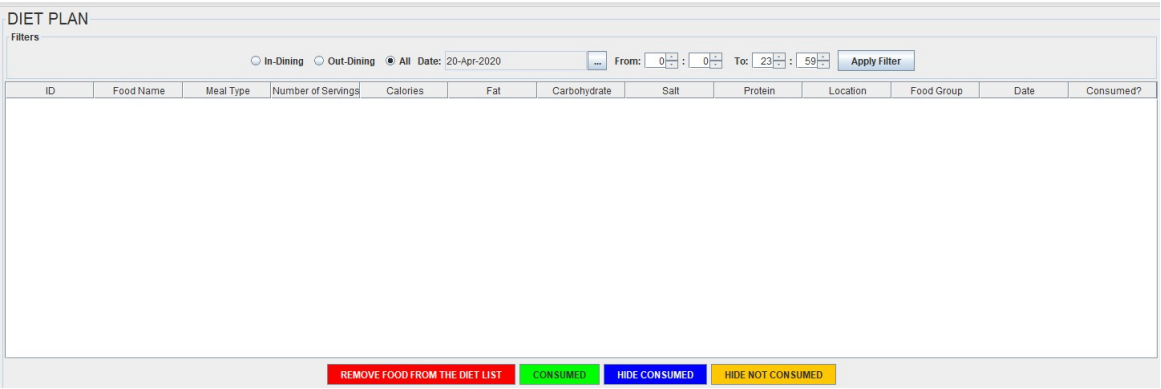

Fig. 7 - DIET PLAN UI

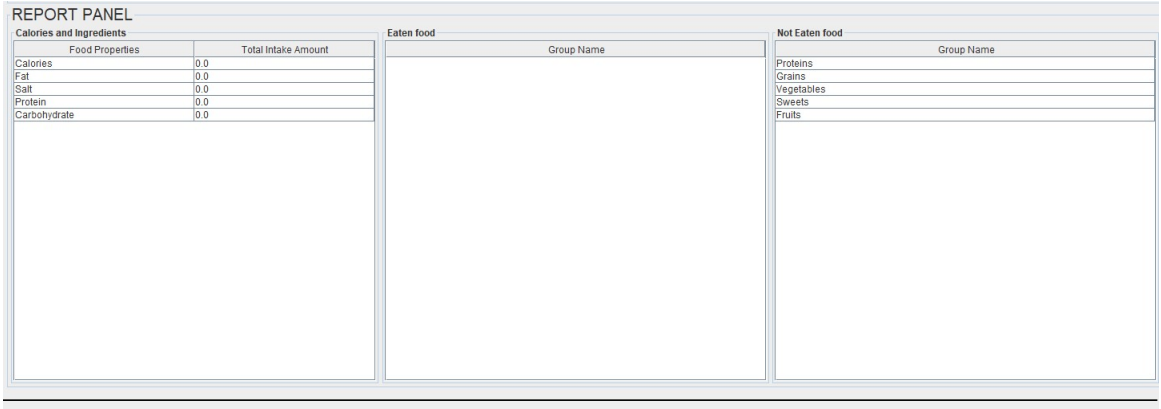

Fig. 8 - REPORT PANEL UI

5 Database Integration

One of the central requirements of this increment was implementation of database in the software. Although this has been done in the preceding increment, the documentation of the database was not included in the Design Document. Therefore, in this section the DB schema will be provided along with the explanation of the database integration in the project.

Among the reasons for choosing SQLite database for the project are the following:

- Small size.
- Portable.
- Serverless.
- No need for setup or administration.
- Popularity.

DBeaver utility has been used to create the database and fill the tables. The graph below shows the ER-diagram generated by DBeaver that represents the db schema.

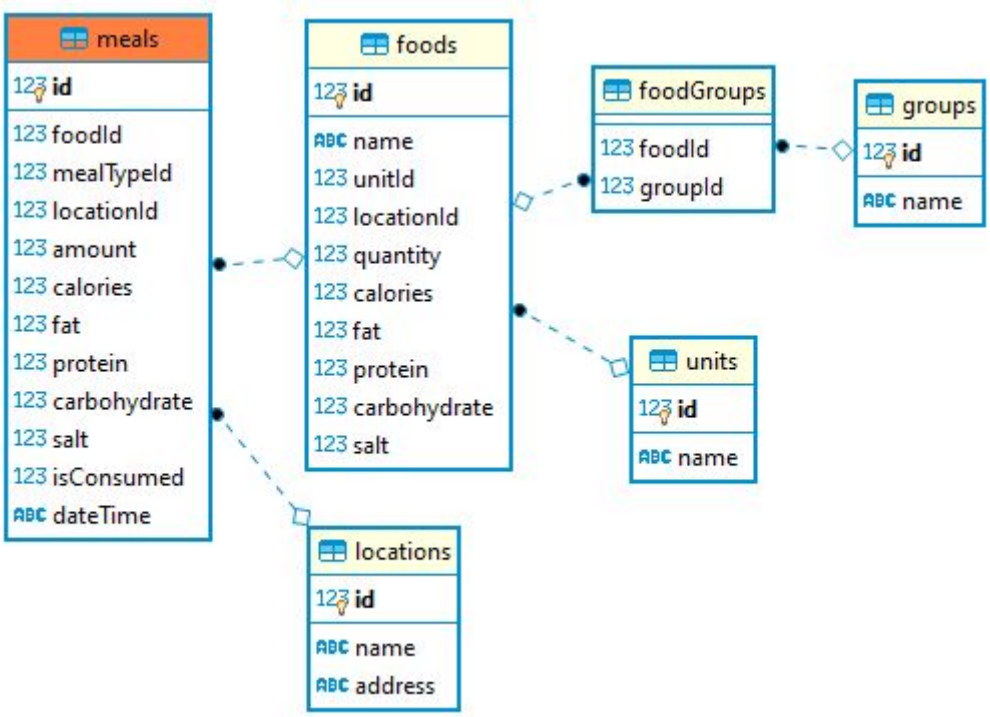

Fig. 9 - ER-Diagram of the database

With the Java library java.sql it is possible to pass SQL queries to the SQLite database and manipulate it. Communication with the database is done with Data Access Objects (daos) and controllers.

6 Conclusion

Over the course of the term our team has been developing a personal dietary application. The development has been conducted in three increments, each bearing the results in the form of a working prototype and corresponding documentation. The application expanded functionality and complexity with each increment.

At the end of the increment 3, the software successfully addresses all the requirements provided by the client. Extra requirements have been added to expand the functionality of the software. The documentation provides a brief summary of different stages of the Software Engineering process: requirement gathering is reflected in the SRD, design and implementation - in Design Document and Verification - in Test Document.