Project title: Food Management System

Khoi Nguyen Mai, Do Phong Tran
University of Adelaide

1. Use Case Description

The food management system is an app designed to help busy individuals (especially students) manage their meal plans and cooking efficiently. The app will enable users to create a meal plan for a week/month, view and add different recipes to the database and create shopping lists containing different ingredients. Users can also search, add and remove ingredients as well as recipes from the database. The project goal is to help people with busy schedules to plan their meals quickly at any time, keep track of the ingredients, create a shopping list, providing a tool to aid in maintaining balanced and healthy eating habits.

The data from the system will be stored and manipulated in the database using SQLite and will contain all the information about different recipes and ingredients: name, quantity, expiration date, nutrition as well as a modified calendar to plan meals for different days. This will make it simple to add, and remove data after compilation, or store data permanently without having to write complex lines of code.

2. Class list

1. MainWindow: Represents a window for the food management system with attributes *ui, database, pointers to Ingredient, Recipe, MealPlan, Storage, OnlineShopping, and MarketPurchasing class: *ptrIngredient, *ptrRecipe, *ptrMealPlan, *ptrStorage, *ptrOnlineShopping, *ptrMarketPurchasing and associated functions:
   on_ingredientButton_clicked(), on_recipeButton_clicked(),
   on_mealPlanButton_clicked(), on_storageButton_clicked(),
   on_shoppingListButtonClicked().

2. Ingredient: Represents a dialog box that displays and prompts users to search for ingredients with pointers *ptrAddIngredient, *ptrRemoveIngredient, *ui and *model and associated functions: on_btnAdd_clicked(), on_btnSearch_clicked(), on_btnRemove_clicked(), loadAllIngredients().

3. AddIngredient: Represents a dialog box (can be accessed by clicking the add button in the Ingredient dialog box) that prompts users to add ingredients to the database by typing associated variables: IngredientName, IngredientQuantity, IngredientExpiryDate and IngredientDescription. Attributes ui*, database (used to access the database) and functions on_btnReset_clicked(), on_btnSave_clicked() are included.

4. RemoveIngredient: Represents a dialog box that prompts users to remove ingredients from the database by typing the ingredient's name. Attributes *ui, database and function: on_ButtonBox_accepted are included.

5. Recipe: Represents a dialog box that displays and prompts users to search for recipes with pointers *ui, *ptrAddRecipe, *ptrRemoveRecipe, *model and associated functions: on_btnAdd_clicked(), on_btnSearch_clicked(), on_btnRemove_clicked(), loadAllIngredients().

6. AddRecipe: Represents a dialog box that prompts users to add recipes to the database by typing associated variables: RecipeName, Ingredients, CookingTime, Nutrition, and Instruction. Attributes ui*, database and functions on_btnReset_clicked(), on_btnSave_clicked() are included.

7. RemoveRecipe: Represents a dialog box that prompts users to remove ingredients from the database by typing the recipe's name. The attribute ui*, database and function on_ButtonBox_accepted() are included.

8. MealPlan: Represents a split-screen widget with the left showing a month-long calendar and the right showing meal plan information when clicking on each day in the calendar. This class has attributes like *ui, database and *ptrEditMealPlan that point to the EditMealPlan class. The functions of the class are: on_btnLoad_clicked(), on_btnAdd_clicked(), on_btnDelete_clicked(), and on_btnEdit_clicked().

9. EditMealPlan: Represents a dialog box that allows the user to edit the meal plan of the currently selected day. This class has attributes like *ui, database, and function on_btnSave_clicked().

10. List: Represents a template list widget for the app with attributes *ui and virtual functions like on_btnAdd_clicked(), on_btnRemove_clicked(), on_list_itemClicked() and the function loadAllElements().

11. Storage: Represents a list of ingredients that the user has already had and inherits from the List class with additional attributes *ptrAddToStorage and *ptrRemoveFromStorage. The functions of the class are on_list_itemClicked(), on_btnAdd_clicked() and on_btnRemove_clicked().

12. AddToStorage: Represents a dialog box that can be accessed by clicking the Add button. This dialog box allows the user to choose ingredients and provide information to add to storage. The attributes of this class are *ui, database and the functions are on_btnLoad_clicked(), on_btnSave_clicked(), and on_btnReset_clicked().

13. RemoveFromStorage: Represents a dialog box that can be accessed by clicking the Remove button. This dialog box allows the user to choose the ingredient in storage and remove it. The attributes of this class are *ui, database and the functions are on_btnLoad_clicked() and on_ButtonBox_accepted().

14. ShoppingList: Represents a shopping list and inherits from the List class. It has additional attributes such as *ptrAddToShoppingList, *ptrRemoveFromShoppingList and functions like on_btnAddClicked(), on_btnRemoveClicked().

15. RemoveFromShoppingList: Represents a dialog box that can be accessed through the Remove button. This dialog box allows the user to choose the ingredient in the shopping list from a combo box and remove it. The attributes of this class are *ui, database and the functions are on_btnLoad_clicked() and on_ButtonBox_accepted().

16. AddToShoppingList: Represents a dialog box that can be accessed by clicking the Add button. This dialog box allows the user to choose ingredients and provide information to add to storage. The attributes of this class are *ui, database and the functions are on_btnLoad_clicked() and on_ButtonBox_accepted().

17. OnlineShopping: Represents an online shopping list and inherits from the shopping list. It has an additional attribute *ptrShoppingLink and function on_list_itemClicked().

18. ShoppingLink: Represents a dialog that provides shopping links when the user clicks on each ingredient in the online shopping list. This class has attributes like *ui, *model and the function displayLink().

19. MarketPurchasing: Represents a market purchasing list and inherits from the shopping list. It has an additional attribute *ptrMarketInfo and function on_list_itemClicked().

20. MarketInfo: Represents a dialog that provides market information when the user clicks on each ingredient in the MarketPurchasing list. This class has attributes like *ui, *model and the function showMarketInfo().

3. Data and function members

Note: All functions that have the form on_Button_clicked() are declared in private slots of each class so that it will not cause conflict if there are functions with the same name in other classes.

1. MainWindow

- Attributes:
    - *ui: Ui::MainWindow (a pointer points to the UI of the main window)
    - *ptrIngredient: Ingredient
    - *ptrRecipe: Recipe
    - database: QSqlDatabase
- Functions:
    - on_ingredientButton_clicked(): opens the Ingredient feature which allows user to add, remove and search for ingredient information.
    - on_recipeButton_clicked(): opens the Recipe feature which allows the user to add, remove, and search for recipe information.
    - on_mealPlanButton_clicked(): opens the MealPlan feature which allows user to set up their own meal plan
    - on_storageButton_clicked(): opens the Storage feature which allows user to
    - on_shoppingListButton_clicked(): opens the list feature which includes OnlineShopping list and MarketPurchasing list

2. Ingredient

- Attributes:
    - *ui: Ui::Ingredient
    - *ptrAddIngredient: AddIngredient
    - *ptrRemoveIngredient: RemoveIngredient
    - *model: QSqlQueryModel
- Functions:
    - loadAllIngredients(): load all the existing ingredients and populate the Ingredient dialog box with ingredient data.
    - on_btnAdd_clicked(): Function that associates with the add button on the Ingredient dialog box and will open the AddIngredient dialog.
    - on_btnSearch_clicked(): Find Ingredients by the name typed in the search bar
    - on_btnRemove_clicked(): open the RemoveIngredient dialog

3. AddIngredient

- Attributes:
    - *ui: Ui::AddIngredient
    - database: QSqlDatabase
- Functions:
    - on_btnSave_clicked(): Function that is responsible for adding ingredients (includes ingredient's name, quantity, expiry date and description) into the

database through a click button called “add” then calls on_btnReset_clicked() to reset the input to take new ingredients from the user.
    - on_btnReset_clicked(): Function that is responsible for resetting the input to take ingredients from the user.

4. RemoveIngredient

- Attributes:
    - *ui: Ui::RemoveIngredient
    - database: QSqlDatabase
- Functions:
    - on_ButtonBox_accepted(): Function that is responsible for confirming and removing ingredients from the database (The user will type the name of the ingredient) through a click button called “remove”.

5. Recipe

- Attributes:
    - *ui: Ui::Recipe
    - *ptrAddRecipe: AddRecipe
    - *ptrRemoveIngredient: RemoveRecipe
    - *model: QSqlQueryModel
- Functions:
    - loadAllRecipes(): Functions that load all the existing recipes and populate the Recipe dialog box with recipe data.
    - on_btnAdd_clicked(): open AddRecipe dialog
    - on_btnSearch_clicked(): Search recipes by the name inputted in the search bar
    - on_btnRemove_clicked(): open RemoveRecipe dialog

6. AddRecipe

- Attributes:
    - *ui: Ui::AddRecipes
    - database: QSqlDatabase
- Functions:
    - on_btnReset_clicked(): Function that is responsible for resetting the input to take recipes from the user.
    - on_btnSave_clicked(): Function that is responsible for adding recipes (includes recipe’s name, ingredients, nutrition, cooking time and description) into the database through a click button called “add” after that calls on_btnReset_clicked() to reset the input to take new recipes from the user.

7. RemoveRecipe

- Attributes:
    - *ui: Ui::RemoveRecipe

    - database: QSqlDatabase
- Functions:
    - on_ButtonBox_accepted(): Function that is responsible for removing recipes from the database (The user will type the name of the recipe) through a click button called “remove”.

8. MealPlan

- Attributes:
    - *ui: Ui::MealPlan
    - database: QSqlDatabase
    - *ptrEditMealPlan: EditMealPlan
- Functions:
    - on_btnLoad_clicked(): load all the recipes available in the database for the user to choose
    - on_btnAdd_clicked(): add the recipes into mealplan of the currently selected date
    - on_btnDelete_clicked(): delete the plan for the currently selected date
    - on_btnEdit_clicked(): open the EditMealPlan dialog which allows the user to edit meal plan in of the currently selected date

9. EditMealPlan

- Attributes:
    - *ui: Ui::EditMealPlan
    - database: QSqlDatabase
- Functions:
    - on_btnSave_clicked(): save the newly edited plan

10. List

- Attributes:
    - *ui: Ui::List
- Functions:
    - on_btnAddClicked(): a virtual function that can be overridden by the derived classes to provide different implementations of the function
    - on_btnRemoveClicked(): a virtual function that can be overridden by the derived classes to provide different implementations of the function
    - loadAllElements(): load all the ingredients added to the list
    - on_list_itemClicked(): a virtual function that can be overridden by the derived classes to provide different implementations of the function

11. Storage

- Attributes:
    - *ptrAddToStorage: AddToStorage
    - *ptrRemoveFromStorage: RemoveFromStorage
- Functions:

    - on_btnAddClicked(): open the AddToStorage dialog which allows the user to enter the ingredient information and add it to storage
    - on_btnRemoveClicked(): open the RemoveFromStorage dialog which allows the user to enter the name of the ingredient and remove it from the list
    - on_list_itemClicked(): show information of the currently selected ingredient

12. AddToStorage

- Attributes:
    - *ui: Ui::AddToStorage
    - database: QSqlDatabase
- Functions:
    - on_btnLoad_clicked(): load all the available ingredients in the database for the user to choose in the combo box
    - on_btnSave_clicked(): save the information of the ingredient and add to storage
    - on_btnReset_clicked(): reset the input to take new ingredient from the user

13. RemoveFromStorage

- Attributes:
    - *ui: Ui::RemoveFromStorage
    - database: QSqlDatabase
- Functions:
    - on_btnLoad_clicked(): load all the available ingredients in the storage for the user to choose in the combo box
    - on_ButtonBox_accepted(): remove the chosen ingredient from storage

14. ShoppingList

- Attributes:
    - *ptrAddToShoppingList: AddToShoppingList
    - *ptrRemoveFromShoppingList: RemoveFromShoppingList
- Functions:
    - on_btnAdd_clicked(): open the AddToShoppingList dialog which allows the user to enter the ingredient information and add it to the shopping list
    - on_btnRemove_clicked(): open the RemoveToShoppingList dialog which allows the user to enter ingredient name and remove it from the shopping list

15. AddToShoppingList

- Attributes:
    - *ui: Ui::AddToShoppingList
    - database: QSqlDatabase
- Functions:
    - on_btnLoad_clicked(): load all the recipe available in the database for the user to choose to add to the shopping list

    - on_ButtonBox_accepted(): confirm and add ingredients to shopping list

16. RemoveFromShoppingList
- Attributes:
    - *ui: Ui::RemoveFromShoppingList
    - database: QSqlDatabase
- Functions:
    - on_btnLoad_clicked(): load all the ingredients added into the shopping list for the user to choose
    - on_ButtonBox_accepted(): confirm and remove the chosen ingredient from the list

17. OnlineShopping
- Attributes:
    - *ptrShoppingLink: ShoppingLink
- Functions:
    - on_list_itemClicked(): open the ShoppingLink dialog that displays the online shopping link of the currently selected ingredient

18. ShoppingLink
- Attributes:
    - *model: QSqlQueryModel
- Functions:
    - displayLink(): show the online shopping link

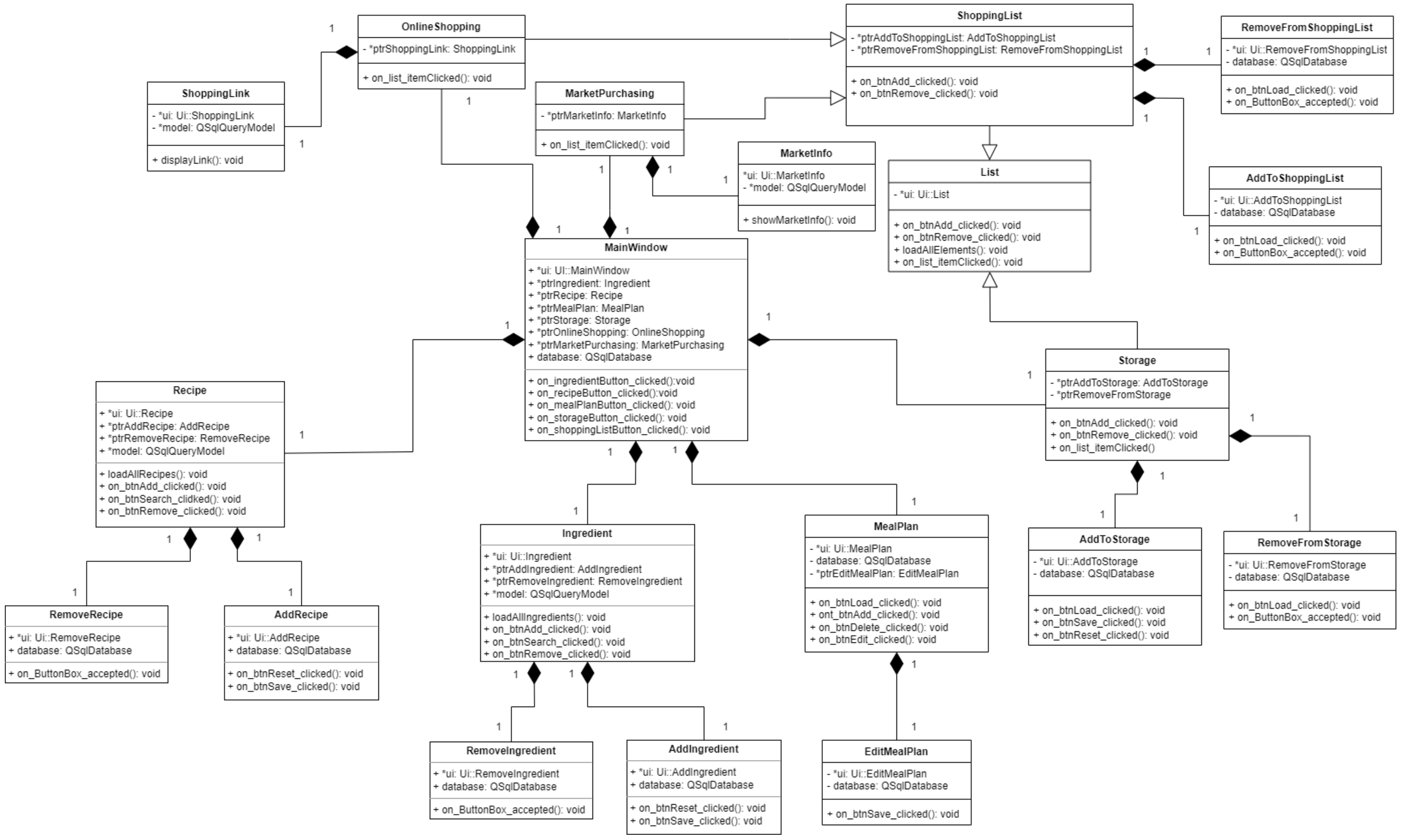
19. MarketPurchasing
- Attributes:
    - *ptrMarketInfo: MarketInfo
- Functions:
    - on_list_itemClicked(): open the MarketPurchasing dialog that displays the market information (stall name, phone, location) for the currently selected ingredient

20. MarketInfo
- Attributes:
    - *model: QSqlQueryModel
- Functions:
    - showMarketInfo(): show the market information

4. Relationship between classes

- MainWindow class is a separate class that displays the main window and all the interactive buttons: Ingredient, Recipe, MealPlan, ShoppingList and Storage for users to navigate and interact with the window.
- Ingredient class is the composition of MainWindow class.
- AddIngredient and RemoveIngredient are compositions of Ingredient class.

- Recipe class is the composition of MainWindow class.
- AddRecipe and RemoveRecipe are compositions of Recipe class.
- MealPlan class is the composition of MainWindow class.
- EditMealPlan class is the composition of MealPlan class, which displays a dialog box to edit the meal plan.
- List class is an interface (abstract class) that contains different functions for Storage class and Shopping List to inherit from and implement further.
- Storage class is the composition of MainWindow class and a derived class from List class.
- AddToStorage and RemoveFromStorage are compositions of Storage class.
- The ShoppingList class inherits the attribute and the functions of the List class
- The RemoveFromShoppingList and AddToShoppingList are compositions of ShoppingList
- The OnlineShopping and MarketPurchasing inherit attributes and functions from the ShoppingList class
- The ShoppingLink class is the composition of OnlineShopping class
- The MarketInfo class is the composition of MarketPurchasing class


OnlineShopping
- *ptrShoppingLink: ShoppingLink
+ on_list_itemClicked(): void
ShoppingLink
- *ui: Ui::ShoppingLink
- *model: QSqlQueryModel
+ displayLink(): void
MarketPurchasing
- *ptrMarketInfo: MarketInfo
+ on_list_itemClicked(): void
MarketInfo
*ui: Ui::MarketInfo
- *model: QSqlQueryModel
+ showMarketInfo(): void
ShoppingList
- *ptrAddToShoppingList: AddToShoppingList
- *ptrRemoveFromShoppingList: RemoveFromShoppingList
+ on_btnAdd_clicked(): void
+ on_btnRemove_clicked(): void
RemoveFromShoppingList
- *ui: Ui::RemoveFromShoppingList
- database: QSqlDatabase
+ on_btnLoad_clicked(): void
+ on_ButtonBox_accepted(): void
List
- *ui: Ui::List
+ on_btnAdd_clicked(): void
+ on_btnRemove_clicked(): void
+ loadAllElements(): void
+ on_list_itemClicked(): void
AddToShoppingList
- *ui: Ui::AddToShoppingList
- database: QSqlDatabase
+ on_btnLoad_clicked(): void
+ on_ButtonBox_accepted(): void
MainWindow
+ *ui: UI::MainWindow
+ *ptrIngredient: Ingredient
+ *ptrRecipe: Recipe
+ *ptrMealPlan: MealPlan
+ *ptrStorage: Storage
+ *ptrOnlineShopping: OnlineShopping
+ *ptrMarketPurchasing: MarketPurchasing
+ database: QSqlDatabase
+ on_ingredientButton_clicked():void
+ on_recipeButton_clicked():void
+ on_mealPlanButton_clicked(): void
+ on_storageButton_clicked(): void
+ on_shoppingListButton_clicked(): void
Recipe
+ *ui: Ui::Recipe
+ *ptrAddRecipe: AddRecipe
+ *ptrRemoveRecipe: RemoveRecipe
+ *model: QSqlQueryModel
+ loadAllRecipes(): void
+ on_btnAdd_clicked(): void
+ on_btnSearch_clidked(): void
+ on_btnRemove_clicked(): void
Storage
- *ptrAddToStorage: AddToStorage
- *ptrRemoveFromStorage
+ on_btnAdd_clicked(): void
+ on_btnRemove_clicked(): void
+ on_list_itemClicked()
RemoveRecipe
+ *ui: Ui::RemoveRecipe
+ database: QSqlDatabase
+ on_ButtonBox_accepted(): void
AddRecipe
+ *ui: Ui::AddRecipe
+ database: QSqlDatabase
+ on_btnReset_clicked(): void
+ on_btnSave_clicked(): void
Ingredient
+ *ui: Ui::Ingredient
+ *ptrAddIngredient: AddIngredient
+ *ptrRemoveIngredient: RemoveIngredient
+ *model: QSqlQueryModel
+ loadAllIngredients(): void
+ on_btnAdd_clicked(): void
+ on_btnSearch_clicked(): void
+ on_btnRemove_clicked(): void
MealPlan
- *ui: Ui::MealPlan
- database: QSqlDatabase
- *ptrEditMealPlan: EditMealPlan
+ on_btnLoad_clicked(): void
+ ont_btnAdd_clicked(): void
+ on_btnDelete_clicked(): void
+ on_btnEdit_clicked(): void
AddToStorage
- *ui: Ui::AddToStorage
- database: QSqlDatabase
+ on_btnLoad_clicked(): void
+ on_btnSave_clicked(): void
+ on_btnReset_clicked(): void
RemoveFromStorage
- *ui: Ui::RemoveFromStorage
- database: QSqlDatabase
+ on_btnLoad_clicked(): void
+ on_ButtonBox_accepted(): void
RemoveIngredient
+ *ui: Ui::RemoveIngredient
+ database: QSqlDatabase
+ on_ButtonBox_accepted(): void
AddIngredient
+ *ui: Ui::AddIngredient
+ database: QSqlDatabase
+ on_btnReset_clicked(): void
+ on_btnSave_clicked(): void
EditMealPlan
- *ui: Ui::EditMealPlan
- database: QSqlDatabase
+ on_btnSave_clicked(): void

5. User interaction and description

The user will interact with the food management system through a user-friendly interface (main window) which contains an interactive navigation bar with widgets: Ingredients, Recipes, Meal plan, Storage, and List.

+ Ingredients: When users push the Ingredients button, there will be a window that displays all the ingredients in the database. A search bar is available to find a particular ingredient. There are also two distinctive buttons for users to add and delete ingredients.
+ Recipes: Similar to Ingredients.
+ Meal plan: There will be a calendar that users can add and remove recipes for each day.
+ List: The user will be transported to another window that will be split in half. The left side is the online shopping list where the user can add, and remove ingredients. The links to websites to buy ingredients online will be displayed when clicking the ingredient's name. On the right side, ingredients that are purchased in-store will be displayed with a list of different stalls including names, phone numbers, and locations can be shown when clicking the ingredient's name.

The program will also use error messages and confirmation prompts to guide the user during the program execution. For example, the program will print out "Added successfully" when the user adds an ingredient or a recipe. When the user searches for a dish or an ingredient, the program will ask the user to enter again (error messages) if there are no corresponding dishes or ingredients in the database.

6. Timeline

Mid-semester break:

The main focus of the mid-semester break will be to construct a solid foundation for the code. The aim will be to construct clean, expandable and easy-to-navigate code structures that allow new features to be effectively implemented (Comments are included for readability). Qt framework will be used to develop the UI for the system and so does SQLite for manipulating the database.

- Both members will learn how to contribute to the same project on Git Hub and about version control. We will code a demo of the app in raw C++ to have a brief understanding of the idea.
- All members will learn how to use Qt framework for C++ application development and SQLite for databases during the first week of the mid-semester break.

- In the second week of the semester break, Nguyen and Phong will focus on finishing the documentation for the project and a class diagram to finish the structure of the project. Nguyen and Phong will both work on the UI design for the application.
- Nguyen will design and implement the MainWindow, Ingredient and Recipe features (which include the MainWindow class, Ingredient class, AddIngredient class, RemoveIngredient class, Recipe class, AddRecipe class and RemoveRecipe class). This will take 2 days to complete including UI design, class functionalities and data handling using SQLite.
- Phong will implement the raw C++ code for the 3-level inheritance hierarchy including the List class, ShoppingList class, Storage class, OnlineShopping class and MarketPurchasing class. This will take 2 days to complete. This will be the foundation to implement the final code for these classes in Qt.

Week 9:

The main focus for week 9 is to finish the first version of the application. During this time, any further implementation and new features will be considered to guarantee the progress of the project. This will allow the group to have enough time for debugging and testing in the next week.

- Nguyen will handle the MealPlan class and design the structure for the 3-level inheritance hierarchy in Qt. This will take 3-4 days to complete including research and implementation.
- Phong will implement the code for the List class, Storage class, AddToStorage class and RemoveFromStorage class. This will take 2 days for both rudimentary and refined versions.
- Phong will complete the ShoppingList class and all the derived and composition classes from it in 3-4 days under the supervision of Nguyen. Assistance from Nguyen (code implementation) may be included to guarantee the pace of the project.

Week 10:

This week will solely focus on unit testing and debugging. The first version of the project must be completed on Monday or Tuesday of this week. At the end of the week, the final version must be completed.

- All members will perform unit testing individually for the part that they are in charge of while developing the code. Assistance from others is compulsory if any member is not able to debug the code individually. After the first version is completed, Nguyen and Phong will test each other's code to make sure that all members fully understand the project code and identify any potential bugs.

- Phong will write unit testing and debugging for the MainWindow class, Recipe class, Ingredient class, MealPlan class and their compositions in 3 days.
- Nguyen will write unit testing and debugging for the 3-level inheritance hierarchy and any composition classes from the hierarchy in 3 days and final test for the entire project.

Week 11:

- The completed version of the project will be submitted in a zip file on MyUni on Sunday of week 10. Since Tuesday of week 11 is the deadline, no more implementation will be performed this week. The group will only check the zip file and download them from MyUni to make sure the final code will run normally before the practical on Thursday.

7. Unit testing and debugging plan

7.1 Testing

The program will be tested using a combination of unit testing and input/output testing.
The unit tests will cover all individual classes, functions, and components to guarantee that the project code will run as expected. All the interactive buttons will also be tested to make sure they can run normally and will not break during the presentation:

- Ingredient class:
  - Test that the UI of the Ingredient class is displayed correctly.
  - Test that data about ingredients are correctly stored in the database.
  - Test that previous data about ingredients still exists even if the application is closed.
  - Test that the Ingredient dialog box can retrieve data about ingredients correctly.

- Storage class:
  - Test that the UI of the Storage class is displayed correctly.
  - Test that the data about ingredients can be added or removed correctly from the database using the dialog box.
  - Test that the interactive buttons on the dialog box function correctly.
  - Test that the dialog box can retrieve the existing data about input ingredients correctly.
- etc.

The input/output test will verify that the application can take inputs from users and display data or output search results precisely without issues, especially when retrieving and manipulating data from the database using SQLite.

7.2 Debugging

- Use a debugger (built-in tool in Qt) to step through the code and identify where errors occur.
- Use logging to track the flow of the program and identify errors or bugs that occur during execution.
- Use error handling to catch and handle exceptions that occur during program execution.
- Test the program under different conditions, such as different input values or accessing and initializing a database using SQLite, to identify and fix errors that occur in specific situations.

Makefile and README files will be included in the project, which will provide clear instructions on how to compile, run, and test the program. The Makefile will handle dependencies, and the debugging and release build will be separated. The README file will explain how to use the program and will list any known issues or limitations of the program.