FACULTY OF ENGINEERING AND TECHNOLOGY

FINAL ASSESSMENT FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; BACHELOR of SOFTWARE ENGINEERING (HONS)YEAR 2

ACADEMIC SESSION 2025; SEMESTER 3

PRG2104: OBJECT ORIENTED PROGRAMMING

Project **DEADLINE: Week 14**

INSTRUCTIONS TO CANDIDATES

- This assignment will contribute 50% to your final grade.
- This is an individual assignment.

IMPORTANT

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

- Coursework submitted after the deadline will be awarded 0 marks unless extensions are granted.

Lecturer's Remark (Use additional sheet if required)

I, Lee Ming Hui Isaac (Name), 22057301 (Student identification number), received the assignment and read the comments.
leeminghuiisaac, 24 August 2025 (Signature/Date)

Academic Honesty Acknowledgement

"I Lee Ming Hui Isaac (Student's name), verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties *(refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme)* for any kind of copying or collaboration on any assignment."

leeminghuiisaac, 24 August 2025 (Signature/Date)

Table of Contents

Important Links

GitHub account username: https://github.com/LeeIsaac1201

GitHub final project repository link: https://github.com/sunwaydcis/final-project-LeeIsaac1201/tree/main

YouTube video presentation link: https://youtu.be/tfiztrMctaA

Microsoft OneDrive video presentation link: https://imailsunwayedu-my.sharepoint.com/:v:/g/personal/22057301_imail_sunway_edu_my/EcHf4cIWtxpEgiftz5qtDXcBcdZBxxkKCi6NmoG-WUPw_g?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=Ew0U1o

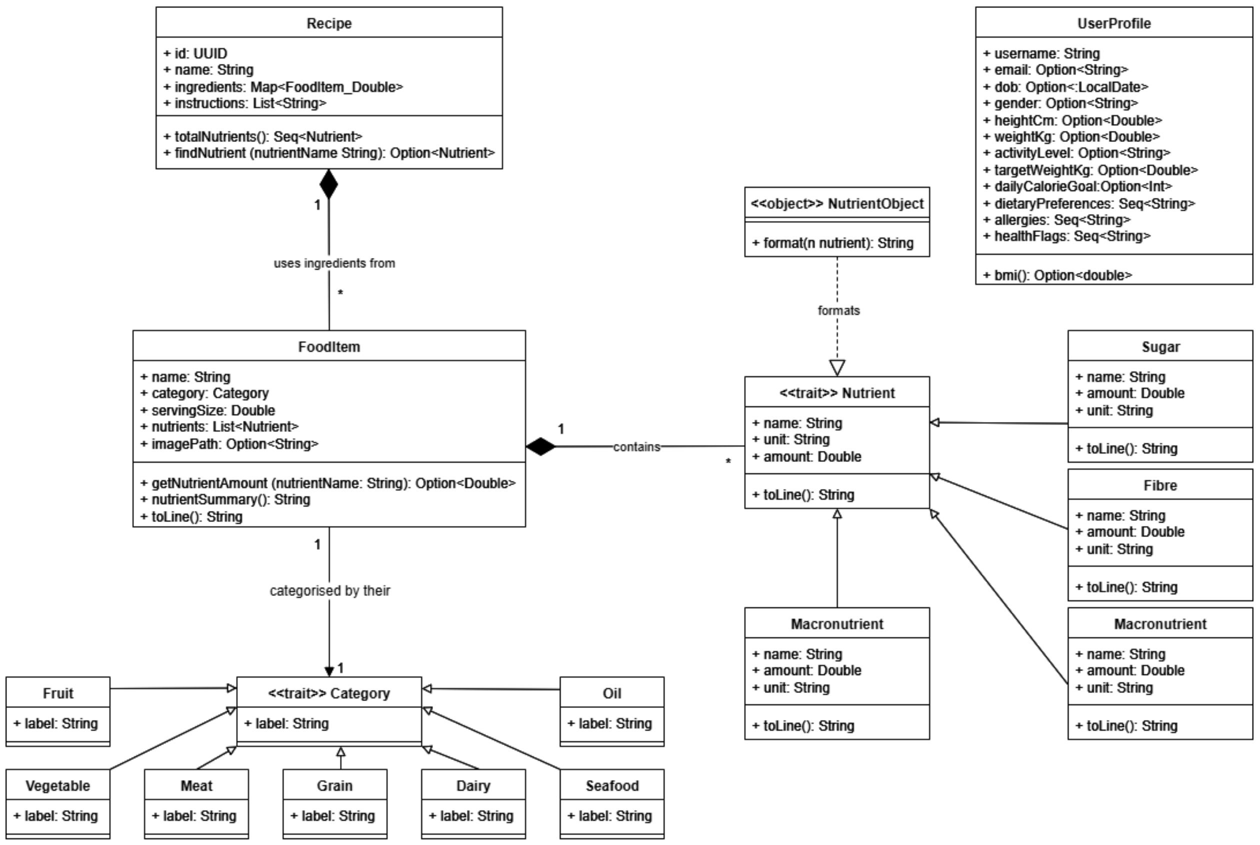
Introduction

NutriTrack is an integrated nutrition information database system that combines a comprehensive food-item database with a persistent storage to support healthy and sustainable meal planning. Built with the United Nations' Sustainable Development Goals (SDGs), specifically SDG 2: End hunger, achieve food security and improved nutrition and promote sustainable agriculture (*Goal 2 | Department of Economic and Social Affairs*, n.d.) in mind, NutriTrack helps users and relevant stakeholders translate nutrition data into practical actions, such as creating balanced recipes, tracking nutrient content, and preserving culturally appropriate food knowledge. In the Malaysian context, where rapid urbanisation and changing dietary patterns coexist with pockets of under- and over-nutrition, NutriTrack can serve schools, community health programs, dietitians, and local farmers by making nutrition information accessible, actionable, and locally relevant.

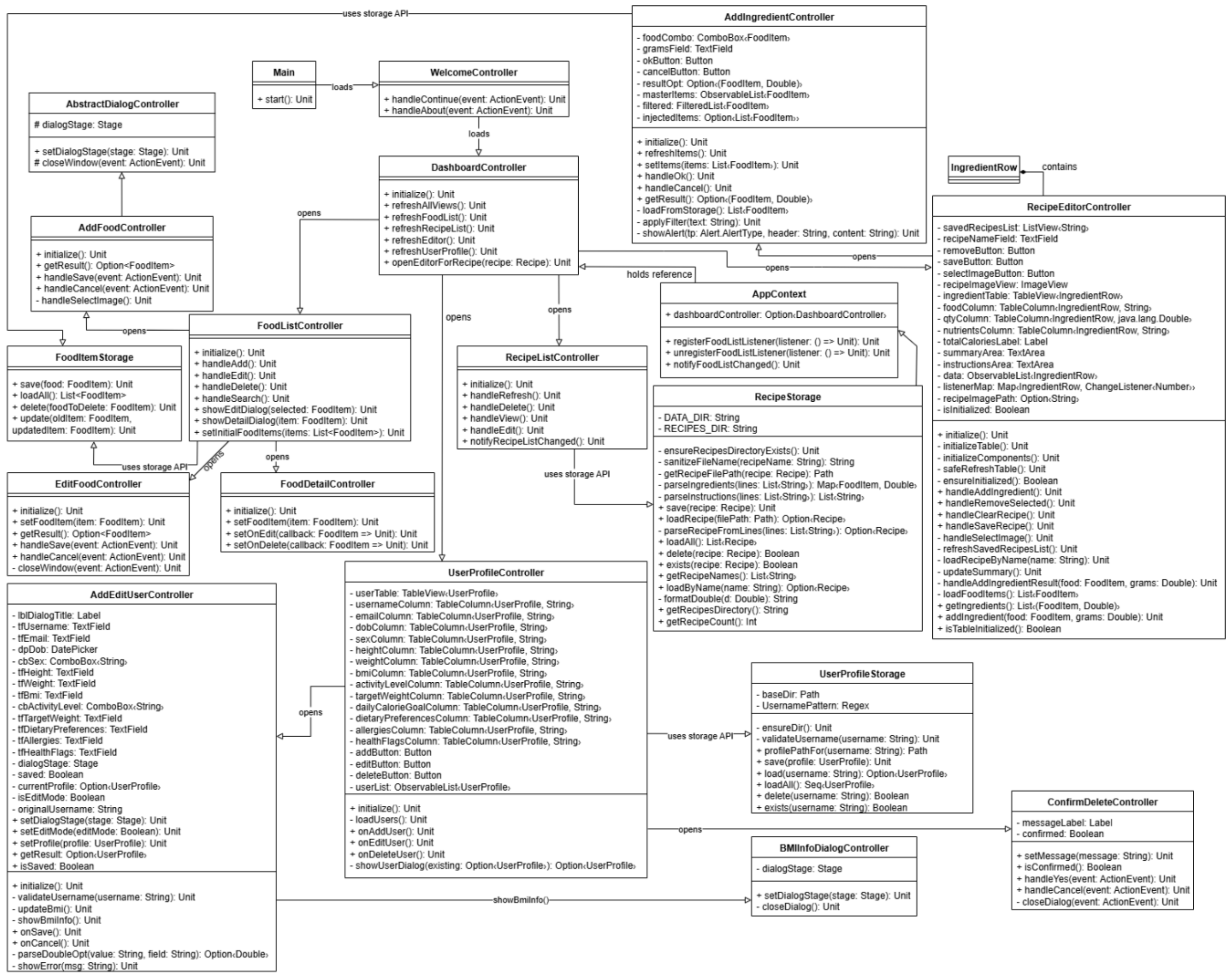
The main motivation behind the development of this project is to target Malaysia's pressing nutrition-related health problems and complications through three complementary actions. First and foremost is to help reduce overweight, obesity, and diabetes by giving users clear nutrition breakdowns so that everyday meal choices cut excess calories, added sugars, and unhealthy fats. Nationally, about 20.1 percent of Malaysian adults are considered obese, alongside having additional health issues, such as diabetes, hypertension, elevated blood pressure, and many more (Chong et al., 2022, p. 789). Second, to address childhood undernutrition and micronutrient shortfalls by producing age-appropriate nutrient-balanced recipes and school or community meal plans that ensure children and other vulnerable groups receive adequate energy and essential vitamins and minerals. The National Health and Morbidity Survey (NHMS) 2019 reported that the prevalence of underweight among children under five rose from 11.6% in 2011 to 14.1% in 2019, while stunting increased from 16.6% to 21.8% over the same period (Lee et al., 2022, p. 2). Third, to support healthier eating habits within the constraints of rising living costs in Malaysia by helping families and individuals make cost-effective food choices without sacrificing nutrition. Practical strategies such as sticking to a grocery budget, planning weekly meals, and taking advantage of discounts can be facilitated through NutriTrack+'s integrated features, like its recipe builder and food-item database, allowing users to plan meals around versatile, affordable ingredients and compare nutritional value across alternatives (Balaratnam, 2024).

Unified Modelling Language (UML) Diagrams

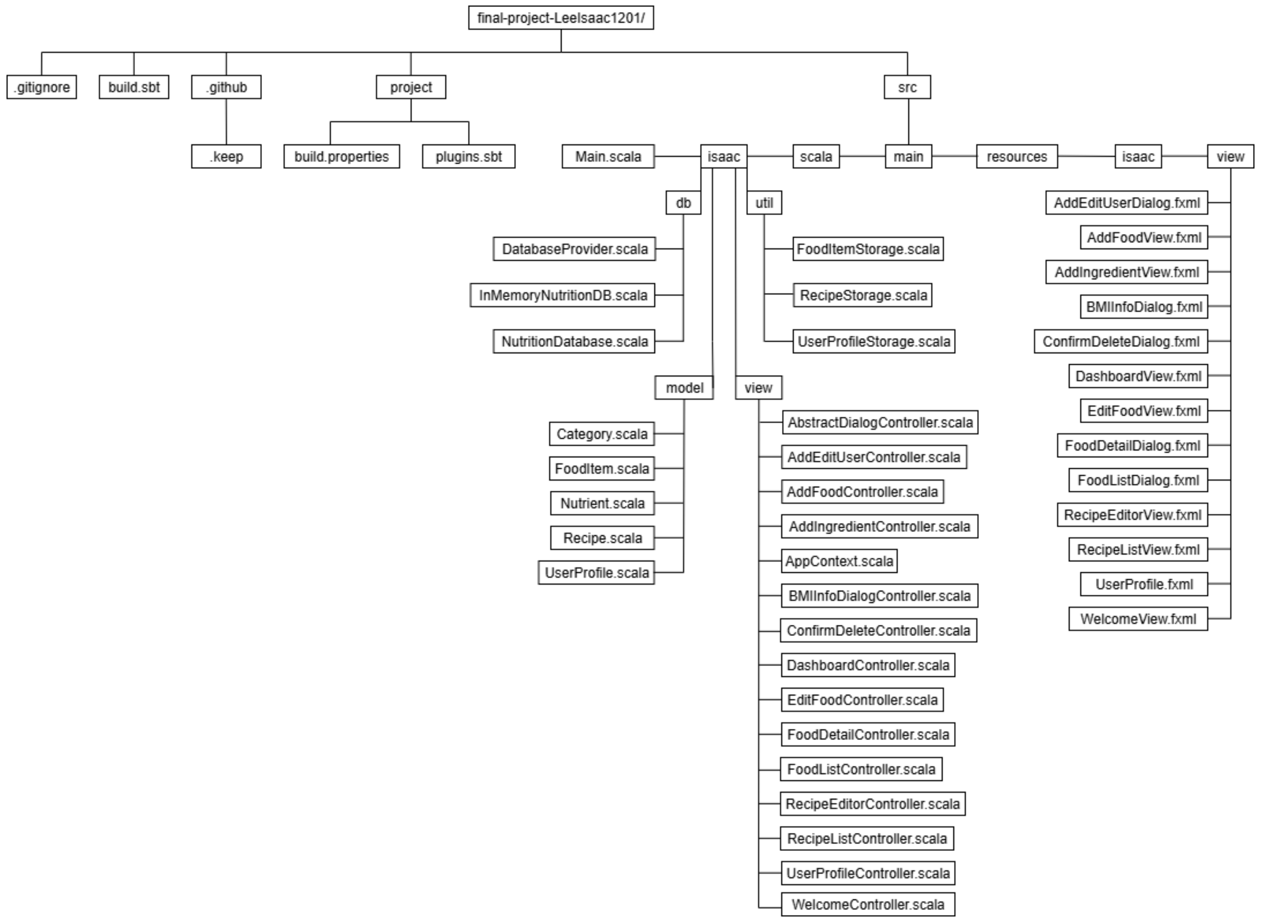
Model Classes UML Diagram



Controller Classes UML Diagram


uses storage API
Main
+ start(): Unit
loads
WelcomeController
+ handleContinue(event: ActionEvent): Unit
+ handleAbout(event: ActionEvent): Unit
loads
AbstractDialogController
# dialogStage: Stage
+ setDialogStage(stage: Stage): Unit
# closeWindow(event: ActionEvent): Unit
DashboardController
+ initialize(): Unit
+ refreshAllViews(): Unit
+ refreshFoodList(): Unit
+ refreshRecipeList(): Unit
+ refreshEditor(): Unit
+ refreshUserProfile(): Unit
+ openEditorForRecipe(recipe: Recipe): Unit
opens
AddFoodController
+ initialize(): Unit
+ getResult(): Option<FoodItem>
+ handleSave(event: ActionEvent): Unit
+ handleCancel(event: ActionEvent): Unit
- handleSelectImage(): Unit
opens
FoodListController
+ initialize(): Unit
+ handleAdd(): Unit
+ handleEdit(): Unit
+ handleDelete(): Unit
+ handleSearch(): Unit
+ showEditDialog(selected: FoodItem): Unit
+ showDetailDialog(item: FoodItem): Unit
+ setInitialFoodItems(items: List<FoodItem>): Unit
FoodItemStorage
+ save(food: FoodItem): Unit
+ loadAll(): List<FoodItem>
+ delete(foodToDelete: FoodItem): Unit
+ update(oldItem: FoodItem, updatedItem: FoodItem): Unit
uses storage API
opens
opens
EditFoodController
+ initialize(): Unit
+ setFoodItem(item: FoodItem): Unit
+ getResult(): Option<FoodItem>
+ handleSave(event: ActionEvent): Unit
+ handleCancel(event: ActionEvent): Unit
- closeWindow(event: ActionEvent): Unit
FoodDetailController
+ initialize(): Unit
+ setFoodItem(item: FoodItem): Unit
+ setOnEdit(callback: FoodItem => Unit): Unit
+ setOnDelete(callback: FoodItem => Unit): Unit
opens
opens
RecipeListController
+ initialize(): Unit
+ handleRefresh(): Unit
+ handleDelete(): Unit
+ handleView(): Unit
+ handleEdit(): Unit
+ notifyRecipeListChanged(): Unit
uses storage API
AddIngredientController
- foodCombo: ComboBox‹FoodItem›
- gramsField: TextField
- okButton: Button
- cancelButton: Button
- resultOpt: Option‹(FoodItem, Double)›
- masterItems: ObservableList‹FoodItem›
- filtered: FilteredList‹FoodItem›
- injectedItems: Option‹List‹FoodItem››
+ initialize(): Unit
+ refreshItems(): Unit
+ setItems(items: List‹FoodItem›): Unit
+ handleOk(): Unit
+ handleCancel(): Unit
+ getResult(): Option‹(FoodItem, Double)›
- loadFromStorage(): List‹FoodItem›
- applyFilter(text: String): Unit
- showAlert(tp: Alert.AlertType, header: String, content: String): Unit
opens
opens
holds reference
IngredientRow
contains
AppContext
+ dashboardController: Option‹DashboardController›
+ registerFoodListListener(listener: () => Unit): Unit
+ unregisterFoodListListener(listener: () => Unit): Unit
+ notifyFoodListChanged(): Unit
RecipeStorage
- DATA_DIR: String
- RECIPES_DIR: String
- ensureRecipesDirectoryExists(): Unit
- sanitizeFileName(recipeName: String): String
- getRecipeFilePath(recipe: Recipe): Path
- parseIngredients(lines: List‹String›): Map‹FoodItem, Double›
- parseInstructions(lines: List‹String›): List‹String›
+ save(recipe: Recipe): Unit
+ loadRecipe(filePath: Path): Option‹Recipe›
- parseRecipeFromLines(lines: List‹String›): Option‹Recipe›
+ loadAll(): List‹Recipe›
+ delete(recipe: Recipe): Boolean
+ exists(recipe: Recipe): Boolean
+ getRecipeNames(): List‹String›
+ loadByName(name: String): Option‹Recipe›
- formatDouble(d: Double): String
+ getRecipesDirectory(): String
+ getRecipeCount(): Int
RecipeEditorController
- savedRecipesList: ListView‹String›
- recipeNameField: TextField
- removeButton: Button
- saveButton: Button
- selectImageButton: Button
- recipeImageView: ImageView
- ingredientTable: TableView‹IngredientRow›
- foodColumn: TableColumn‹IngredientRow, String›
- qtyColumn: TableColumn‹IngredientRow, java.lang.Double›
- nutrientsColumn: TableColumn‹IngredientRow, String›
- totalCaloriesLabel: Label
- summaryArea: TextArea
- instructionsArea: TextArea
- data: ObservableList‹IngredientRow›
- listenerMap: Map‹IngredientRow, ChangeListener‹Number››
- recipeImagePath: Option‹String›
- isInitialized: Boolean
+ initialize(): Unit
- initializeTable(): Unit
- initializeComponents(): Unit
- safeRefreshTable(): Unit
- ensureInitialized(): Boolean
+ handleAddIngredient(): Unit
+ handleRemoveSelected(): Unit
+ handleClearRecipe(): Unit
+ handleSaveRecipe(): Unit
- handleSelectImage(): Unit
- refreshSavedRecipesList(): Unit
- loadRecipeByName(name: String): Unit
- updateSummary(): Unit
- handleAddIngredientResult(food: FoodItem, grams: Double): Unit
- loadFoodItems(): List‹FoodItem›
+ getIngredients(): List‹(FoodItem, Double)›
+ addIngredient(food: FoodItem, grams: Double): Unit
+ isTableInitialized(): Boolean
AddEditUserController
- lblDialogTitle: Label
- tfUsername: TextField
- tfEmail: TextField
- dpDob: DatePicker
- cbSex: ComboBox‹String›
- tfHeight: TextField
- tfWeight: TextField
- tfBmi: TextField
- cbActivityLevel: ComboBox‹String›
- tfTargetWeight: TextField
- tfDietaryPreferences: TextField
- tfAllergies: TextField
- tfHealthFlags: TextField
- dialogStage: Stage
- saved: Boolean
- currentProfile: Option‹UserProfile›
- isEditMode: Boolean
- originalUsername: String
+ setDialogStage(stage: Stage): Unit
+ setEditMode(editMode: Boolean): Unit
+ setProfile(profile: UserProfile): Unit
+ getResult: Option‹UserProfile›
+ isSaved: Boolean
+ initialize(): Unit
- validateUsername(username: String): Unit
- updateBmi(): Unit
- showBmiInfo(): Unit
+ onSave(): Unit
+ onCancel(): Unit
- parseDoubleOpt(value: String, field: String): Option‹Double›
- showError(msg: String): Unit
opens
UserProfileController
- userTable: TableView‹UserProfile›
- usernameColumn: TableColumn‹UserProfile, String›
- emailColumn: TableColumn‹UserProfile, String›
- dobColumn: TableColumn‹UserProfile, String›
- sexColumn: TableColumn‹UserProfile, String›
- heightColumn: TableColumn‹UserProfile, String›
- weightColumn: TableColumn‹UserProfile, String›
- bmiColumn: TableColumn‹UserProfile, String›
- activityLevelColumn: TableColumn‹UserProfile, String›
- targetWeightColumn: TableColumn‹UserProfile, String›
- dailyCalorieGoalColumn: TableColumn‹UserProfile, String›
- dietaryPreferencesColumn: TableColumn‹UserProfile, String›
- allergiesColumn: TableColumn‹UserProfile, String›
- healthFlagsColumn: TableColumn‹UserProfile, String›
- addButton: Button
- editButton: Button
- deleteButton: Button
- userList: ObservableList‹UserProfile›
+ initialize(): Unit
- loadUsers(): Unit
+ onAddUser(): Unit
+ onEditUser(): Unit
+ onDeleteUser(): Unit
- showUserDialog(existing: Option‹UserProfile›): Option‹UserProfile›
uses storage API
UserProfileStorage
- baseDir: Path
- UsernamePattern: Regex
- ensureDir(): Unit
- validateUsername(username: String): Unit
+ profilePathFor(username: String): Path
+ save(profile: UserProfile): Unit
+ load(username: String): Option‹UserProfile›
+ loadAll(): Seq‹UserProfile›
+ delete(username: String): Boolean
+ exists(username: String): Boolean
opens
ConfirmDeleteController
- messageLabel: Label
- confirmed: Boolean
+ setMessage(message: String): Unit
+ isConfirmed(): Boolean
+ handleYes(event: ActionEvent): Unit
+ handleCancel(event: ActionEvent): Unit
- closeDialog(event: ActionEvent): Unit
showBmiInfo()
BMIInfoDialogController
- dialogStage: Stage
+ setDialogStage(stage: Stage): Unit
- closeDialog(): Unit

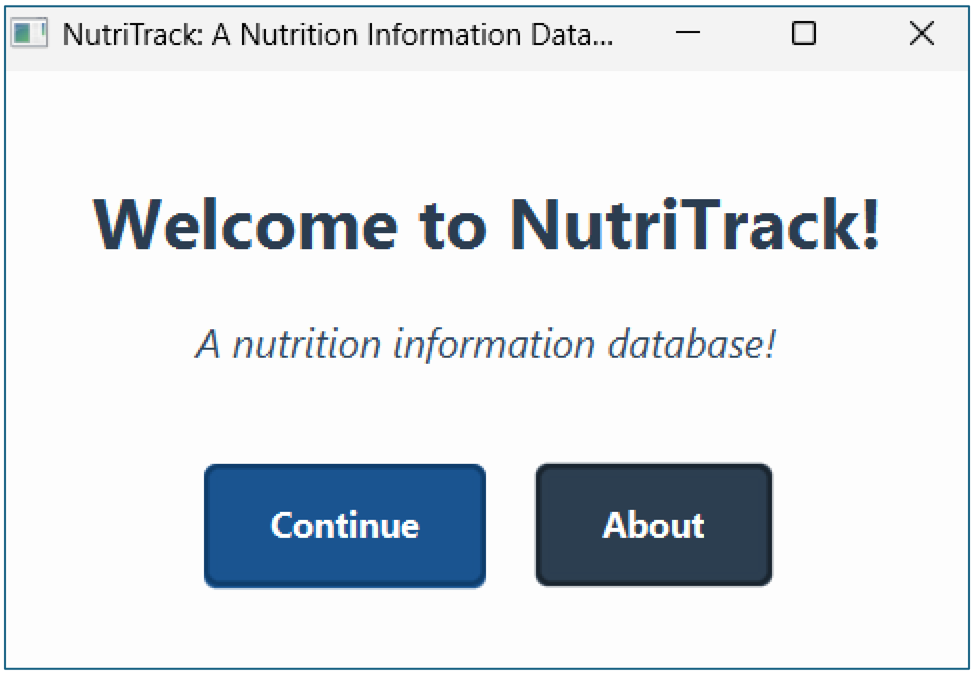
Code Structure Overview



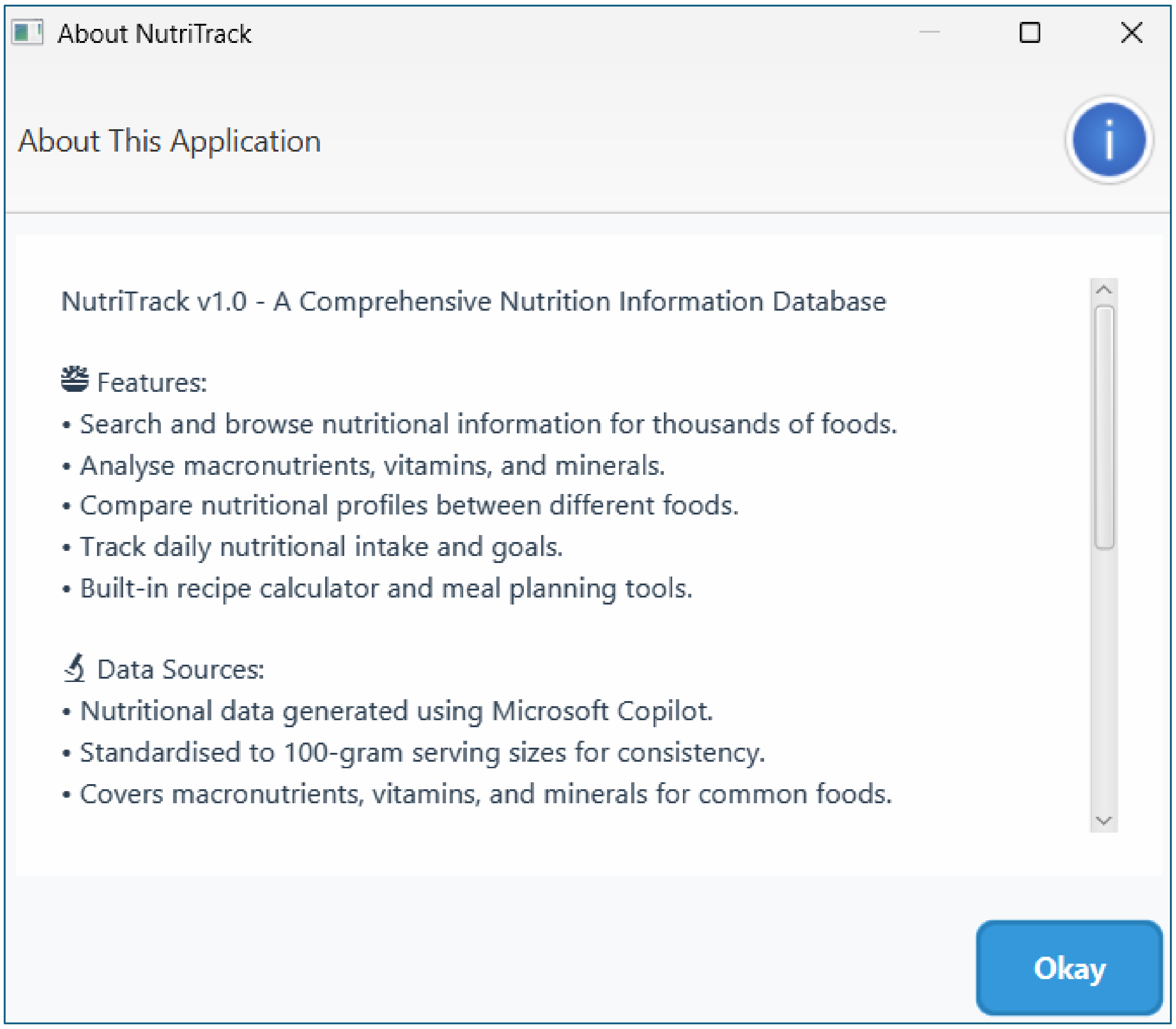
System Functionalities

Welcome Dashboard

This functionality first appears when the user runs Main.scala. It shows the welcome page with two buttons, Continue or About. The Continue button directs the user to the dashboard where the Food List tab is automatically loaded with its contents. The About button displays this project's purpose and motives.


NutriTrack: A Nutrition Information Data...
Welcome to NutriTrack!
A nutrition information database!
Continue
About

(The screenshot above displays the welcome page when users load the application. Two buttons are displayed for the user to proceed to the dashboard or to view this project's relevant information.)


About NutriTrack

About This Application

NutriTrack v1.0 - A Comprehensive Nutrition Information Database

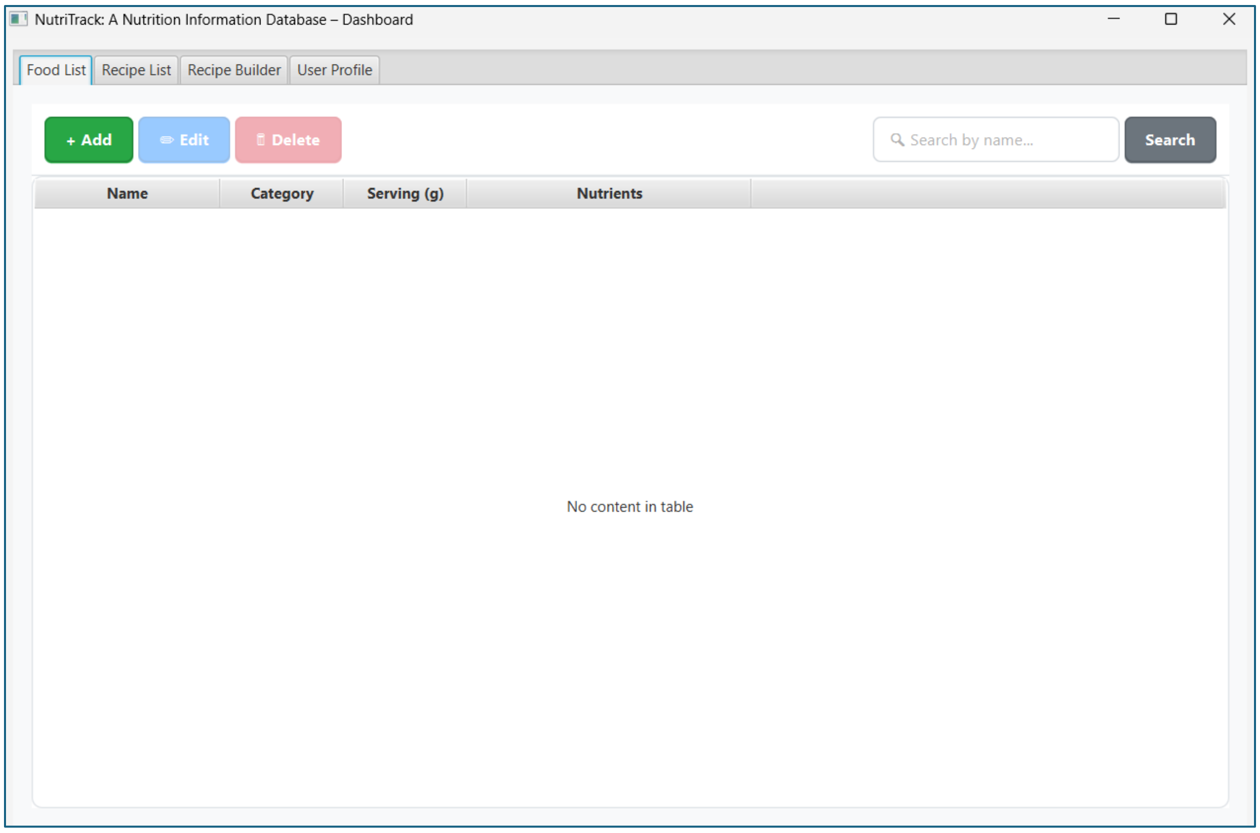
Features:
• Search and browse nutritional information for thousands of foods.
• Analyse macronutrients, vitamins, and minerals.
• Compare nutritional profiles between different foods.
• Track daily nutritional intake and goals.
• Built-in recipe calculator and meal planning tools.

Data Sources:
• Nutritional data generated using Microsoft Copilot.
• Standardised to 100-gram serving sizes for consistency.
• Covers macronutrients, vitamins, and minerals for common foods.

(The screenshot above displays the About pop-up dialog appears when the user clicks on the "About" button previously mentioned. The user can view this project's features, data sources, contact information, and other relevant information.)

Food List Management

This functionality lets users to maintain the catalogue of food items by performing create, read, and delete operations. All food items persisted in a text file named food_items.txt in the application's data directory, with one serialised record per line. This Food List display will be displayed after the "Continue" in the Welcome dashboard is clicked:

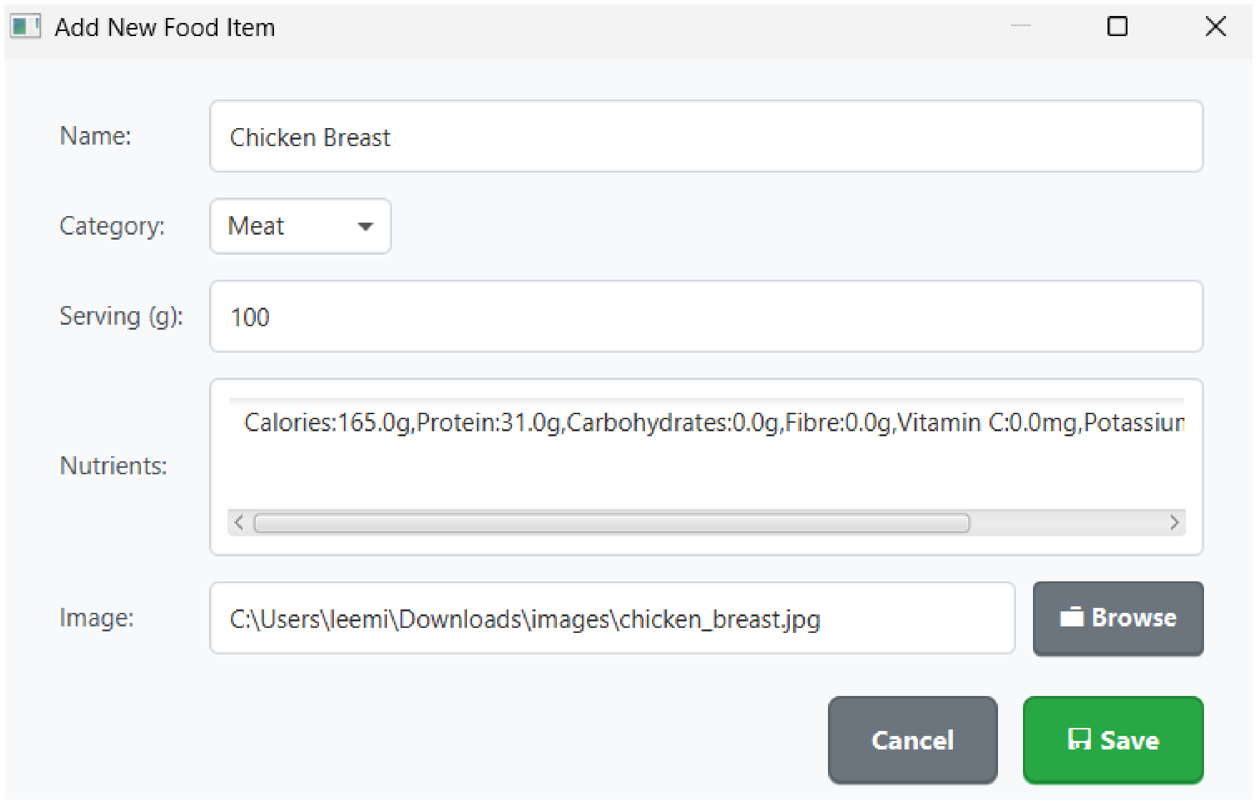


Name	Category	Serving (g)	Nutrients

No content in table

(A screenshot of the Food List dashboard showing the Food List dashboard, which is automatically loaded after the "Continue" button is clicked in the Welcome dashboard. No food items have been created yet.)

When the user wants to add a food item, the “Add” button is clicked and this pop-up dialog will appear:


Add New Food Item

Name: Chicken Breast

Category: Meat

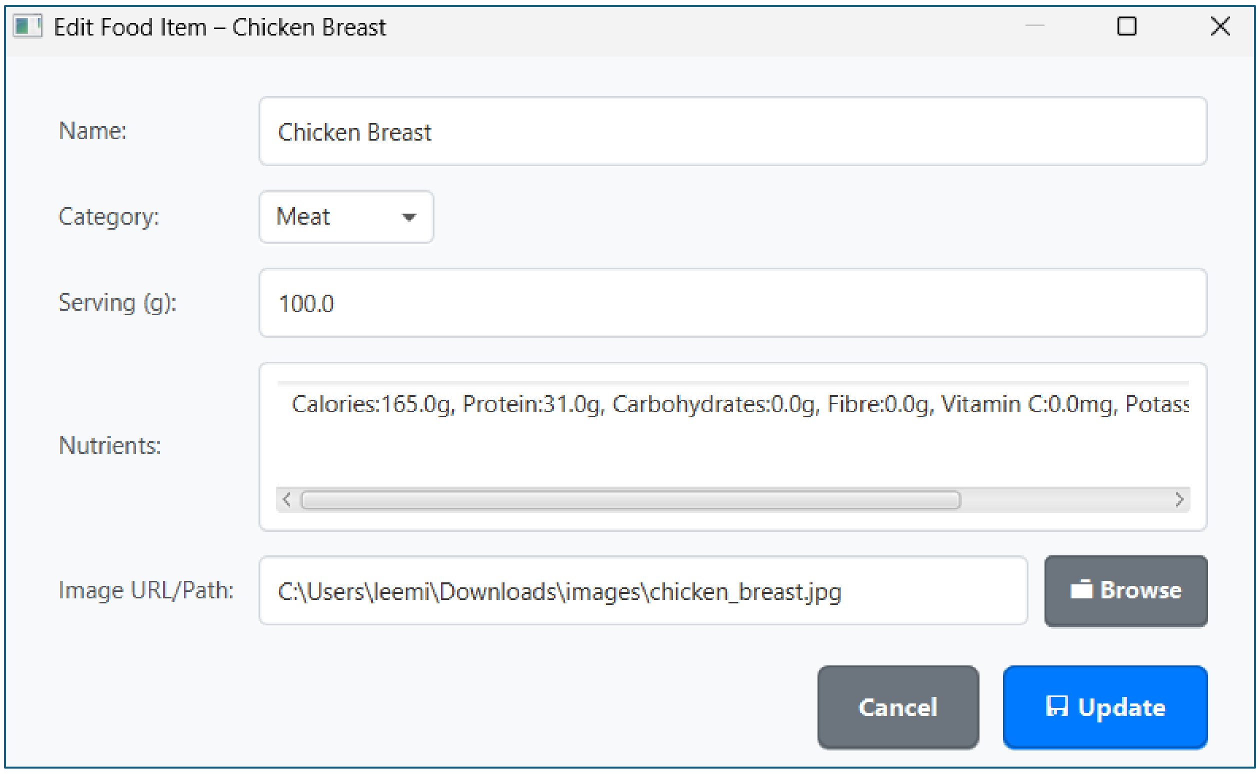
Serving (g): 100

Nutrients: Calories:165.0g,Protein:31.0g,Carbohydrates:0.0g,Fibre:0.0g,Vitamin C:0.0mg,Potassiur

Image: C:\Users\leemi\Downloads\images\chicken_breast.jpg

(A screenshot displaying the Add New Food Item pop-up dialog. Users can insert the food item’s name, serving (in grams), nutrients in a specific format, the image URL/path, and select the food item’s category.)

When the user wants to edit the existing information of a food item, the user will click on the “Edit” button on the Food List dashboard, and the Edit Food Item pop-up dialog will appear with the existing information pre-filled for viewing and editing convenience:


Edit Food Item – Chicken Breast

Name: Chicken Breast

Category: Meat

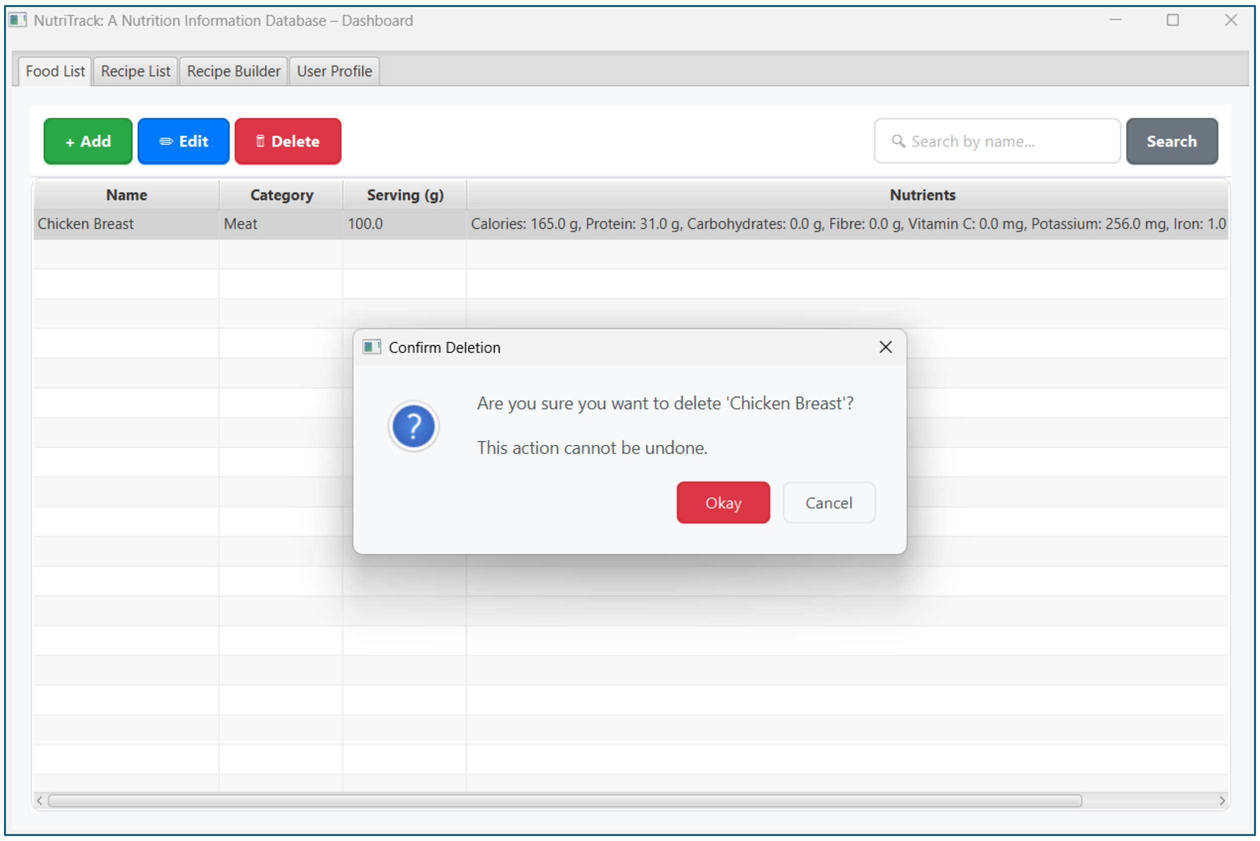
Serving (g): 100.0

Nutrients: Calories:165.0g, Protein:31.0g, Carbohydrates:0.0g, Fibre:0.0g, Vitamin C:0.0mg, Potass

Image URL/Path: C:\Users\leemi\Downloads\images\chicken_breast.jpg

(A screenshot exhibiting the Edit Food Item pop-up dialog, where the name of the food item being edited is also displayed alongside pre-filled existing information.)

When the user wants to delete an existing food item, the user will first select the food item to delete, the click on the "Delete" button. The Confirm Deletion pop-up dialog will appear to seek further confirmation before deleting the food item.



NutriTrack: A Nutrition Information Database – Dashboard

Name	Category	Serving (g)	Nutrients
Chicken Breast	Meat	100.0	Calories: 165.0 g, Protein: 31.0 g, Carbohydrates: 0.0 g, Fibre: 0.0 g, Vitamin C: 0.0 mg, Potassium: 256.0 mg, Iron: 1.0

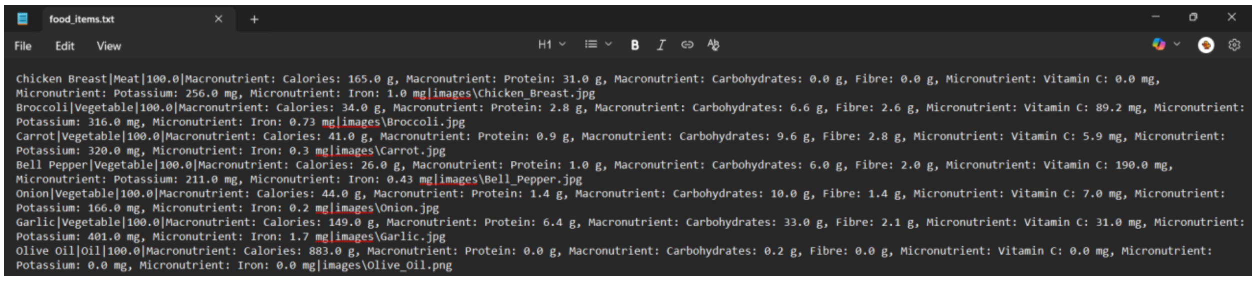
Confirm Deletion

Are you sure you want to delete 'Chicken Breast'?

This action cannot be undone.

(A screenshot of the selected food item being processed for deletion. Before that, a Confirm Deletion pop-up dialog will appear to seek confirmation before deleting the food item, as the food item's information cannot be restored after deletion.)

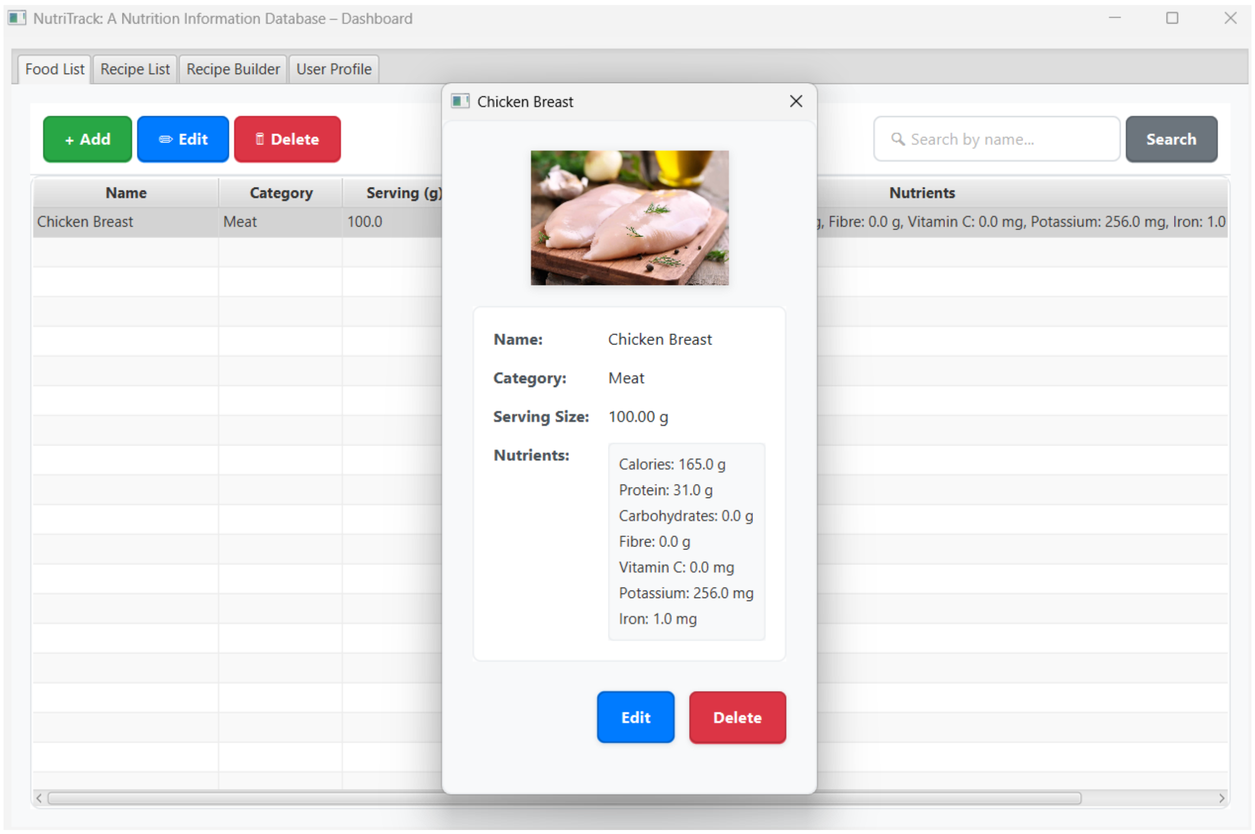
After the user chooses an operation (add, edit, or delete) for a food item, the food_items.txt file storing the list of food items will be updated as well. If a food item is added, the food item will appear at the bottom of the list; if a food item's information is edited, then the food item's information will be updated to the latest version as well; if a food item is deleted, the food item will no longer appear in the .txt file.



```
Chicken Breast|Meat|100.0|Macronutrient: Calories: 165.0 g, Macronutrient: Protein: 31.0 g, Macronutrient: Carbohydrates: 0.0 g, Fibre: 0.0 g, Micronutrient: Vitamin C: 0.0 mg,
Micronutrient: Potassium: 256.0 mg, Micronutrient: Iron: 1.0 mg|images\Chicken_Breast.jpg
Broccoli|Vegetable|100.0|Macronutrient: Calories: 34.0 g, Macronutrient: Protein: 2.8 g, Macronutrient: Carbohydrates: 6.6 g, Fibre: 2.6 g, Micronutrient: Vitamin C: 89.2 mg, Micronutrient:
Potassium: 316.0 mg, Micronutrient: Iron: 0.73 mg|images\Broccoli.jpg
Carrot|Vegetable|100.0|Macronutrient: Calories: 41.0 g, Macronutrient: Protein: 0.9 g, Macronutrient: Carbohydrates: 9.6 g, Fibre: 2.8 g, Micronutrient: Vitamin C: 5.9 mg, Micronutrient:
Potassium: 320.0 mg, Micronutrient: Iron: 0.3 mg|images\Carrot.jpg
Bell Pepper|Vegetable|100.0|Macronutrient: Calories: 26.0 g, Macronutrient: Protein: 1.0 g, Macronutrient: Carbohydrates: 6.0 g, Fibre: 2.0 g, Micronutrient: Vitamin C: 190.0 mg,
Micronutrient: Potassium: 211.0 mg, Micronutrient: Iron: 0.43 mg|images\Bell_Pepper.jpg
Onion|Vegetable|100.0|Macronutrient: Calories: 44.0 g, Macronutrient: Protein: 1.4 g, Macronutrient: Carbohydrates: 10.0 g, Fibre: 1.4 g, Micronutrient: Vitamin C: 7.0 mg, Micronutrient:
Potassium: 166.0 mg, Micronutrient: Iron: 0.2 mg|images\Onion.jpg
Garlic|Vegetable|100.0|Macronutrient: Calories: 149.0 g, Macronutrient: Protein: 6.4 g, Macronutrient: Carbohydrates: 33.0 g, Fibre: 2.1 g, Micronutrient: Vitamin C: 31.0 mg, Micronutrient:
Potassium: 401.0 mg, Micronutrient: Iron: 1.7 mg|images\Garlic.jpg
Olive Oil|Oil|100.0|Macronutrient: Calories: 883.0 g, Macronutrient: Protein: 0.0 g, Macronutrient: Carbohydrates: 0.2 g, Fibre: 0.0 g, Micronutrient: Vitamin C: 0.0 mg, Micronutrient:
Potassium: 0.0 mg, Micronutrient: Iron: 0.0 mg|images\Olive_Oil.png
```

(A screenshot showing how the list of food items is stored in the .txt file. The relevant information is stored separately for each content, where the symbol | is used as a divider.)

When the user wants to view the information of a food item, the user just has to double click on a selected food item, and a pop-up dialog appears displaying the food item's image and the relevant information.



Name	Category	Serving (g)	Nutrients
Chicken Breast	Meat	100.0	, Fibre: 0.0 g, Vitamin C: 0.0 mg, Potassium: 256.0 mg, Iron: 1.0

Name: Chicken Breast

Category: Meat

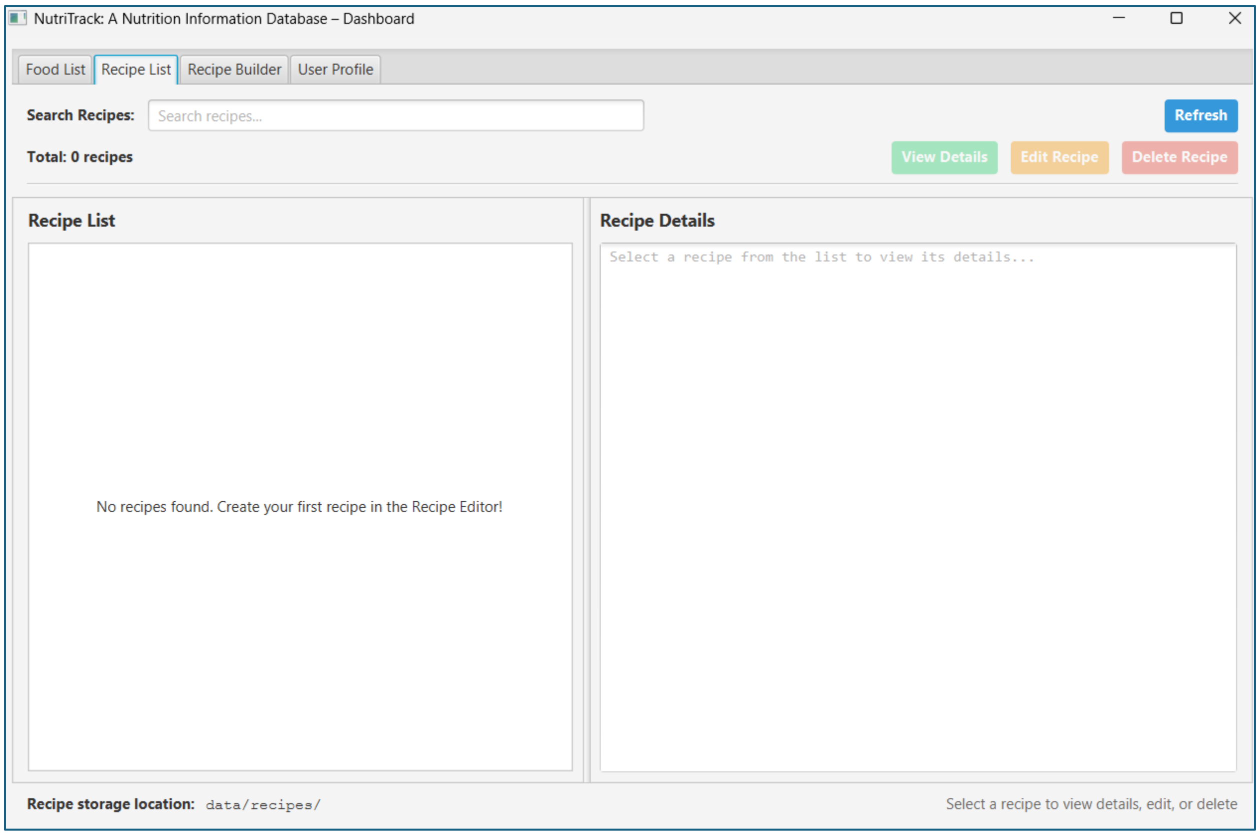
Serving Size: 100.00 g

Nutrients:
Calories: 165.0 g
Protein: 31.0 g
Carbohydrates: 0.0 g
Fibre: 0.0 g
Vitamin C: 0.0 mg
Potassium: 256.0 mg
Iron: 1.0 mg

(This screenshot of the pop-up dialog appears after a user double-clicks on a food item. The food item's picture, name, category, serving size (in grams), and nutrients will be displayed. The buttons to edit or delete the food item are also displayed for the user to initiate the relevant operations.)

<u>Recipe List Management</u>

This functionality lets users maintain the catalogue of recipes by performing create, edit, and delete operations. All recipes stored as individual text files in the data/recipes directory, with each file named <RecipeName>.txt and containing the serialised recipe content and cooking instructions.


NutriTrack: A Nutrition Information Database – Dashboard

Food List | Recipe List | Recipe Builder | User Profile

Search Recipes: Search recipes...

Refresh

Total: 0 recipes

View Details | Edit Recipe | Delete Recipe

Recipe List

No recipes found. Create your first recipe in the Recipe Editor!

Recipe Details

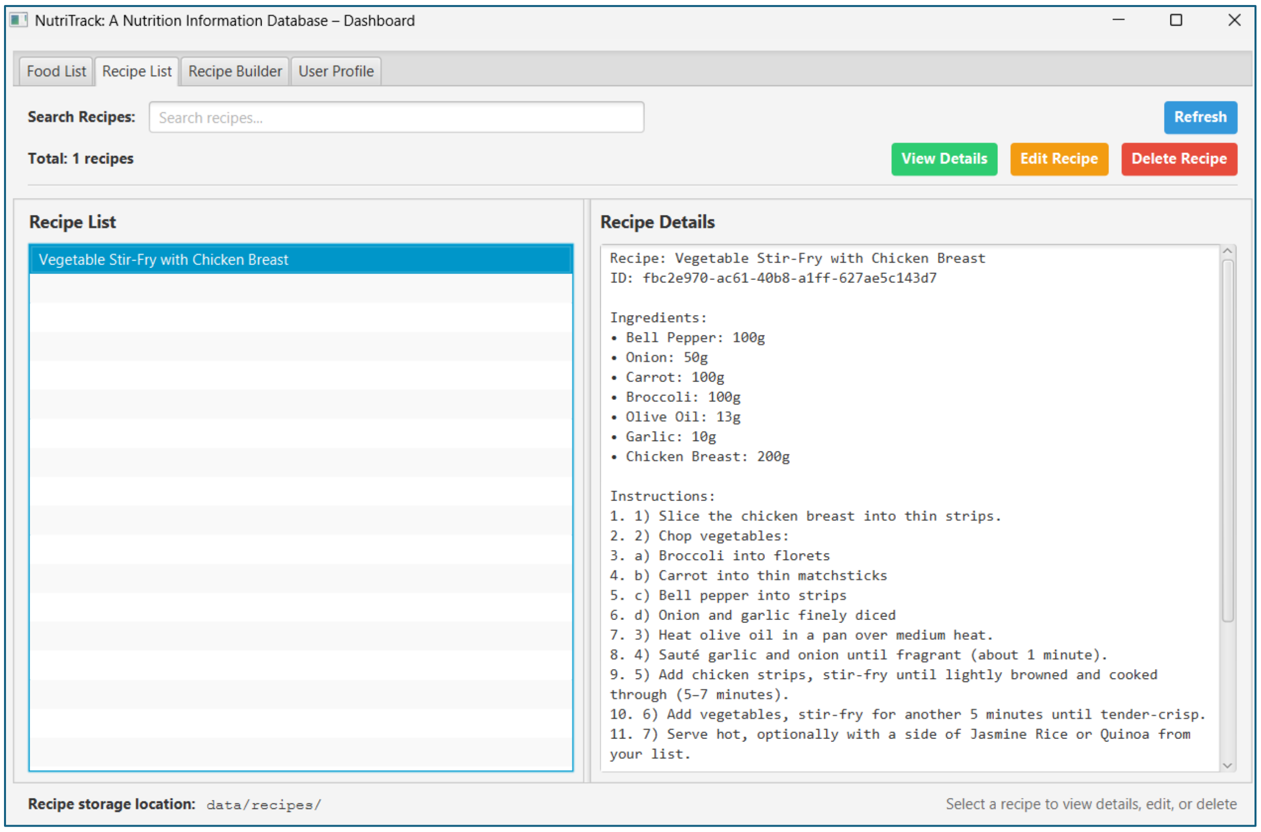
```
Select a recipe from the list to view its details...
```

Recipe storage location: `data/recipes/`

Select a recipe to view details, edit, or delete

(This screenshot displays the Recipe List dashboard, where the recipe list (currently empty) and its details can be viewed. Since there are no recipes, the View Details, Edit Recipe, and Delete Recipe buttons are inaccessible until a recipe has been added and selected.)

After a recipe has been created, the user can click on the refresh button to refresh the dashboard and the list of recipes stored in the recipes folder is loaded.



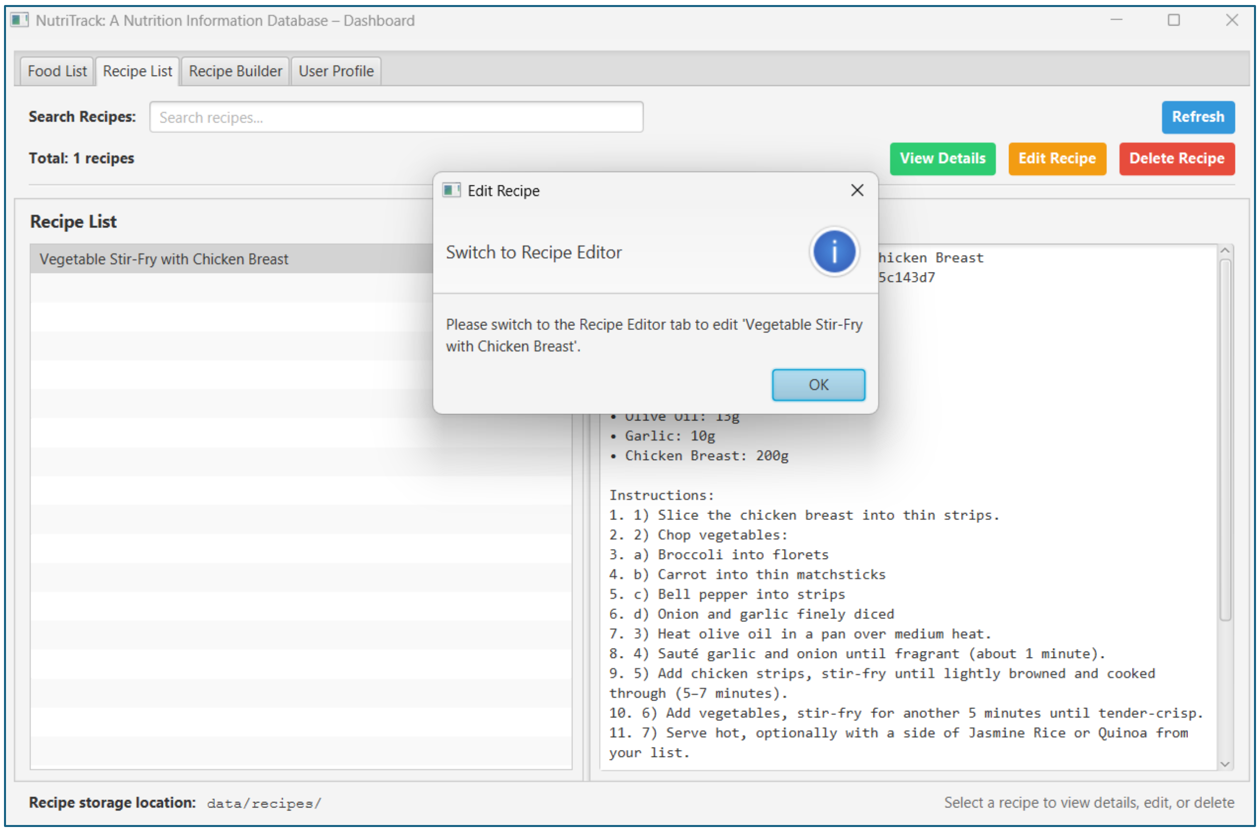
```
Recipe: Vegetable Stir-Fry with Chicken Breast
ID: fbc2e970-ac61-40b8-a1ff-627ae5c143d7

Ingredients:
• Bell Pepper: 100g
• Onion: 50g
• Carrot: 100g
• Broccoli: 100g
• Olive Oil: 13g
• Garlic: 10g
• Chicken Breast: 200g

Instructions:
1. 1) Slice the chicken breast into thin strips.
2. 2) Chop vegetables:
3. a) Broccoli into florets
4. b) Carrot into thin matchsticks
5. c) Bell pepper into strips
6. d) Onion and garlic finely diced
7. 3) Heat olive oil in a pan over medium heat.
8. 4) Sauté garlic and onion until fragrant (about 1 minute).
9. 5) Add chicken strips, stir-fry until lightly browned and cooked
through (5–7 minutes).
10. 6) Add vegetables, stir-fry for another 5 minutes until tender-crisp.
11. 7) Serve hot, optionally with a side of Jasmine Rice or Quinoa from
your list.
```

Recipe storage location: data/recipes/

Select a recipe to view details, edit, or delete

(The screenshot above illustrates the recipe and its information being displayed when selected. Now, the "View Details," "Edit Recipe," and "Delete Recipe" buttons are accessible for the user to click.)

However, when the "Edit Recipe" button is clicked, a pop-up dialog appears telling the user to switch to the Recipe Builder tab for editing. The reason behind this is that the Recipe Builder tab is equipped with the functionalities to edit the existing contents of a recipe, while keeping it visually appealing and be easily navigated.

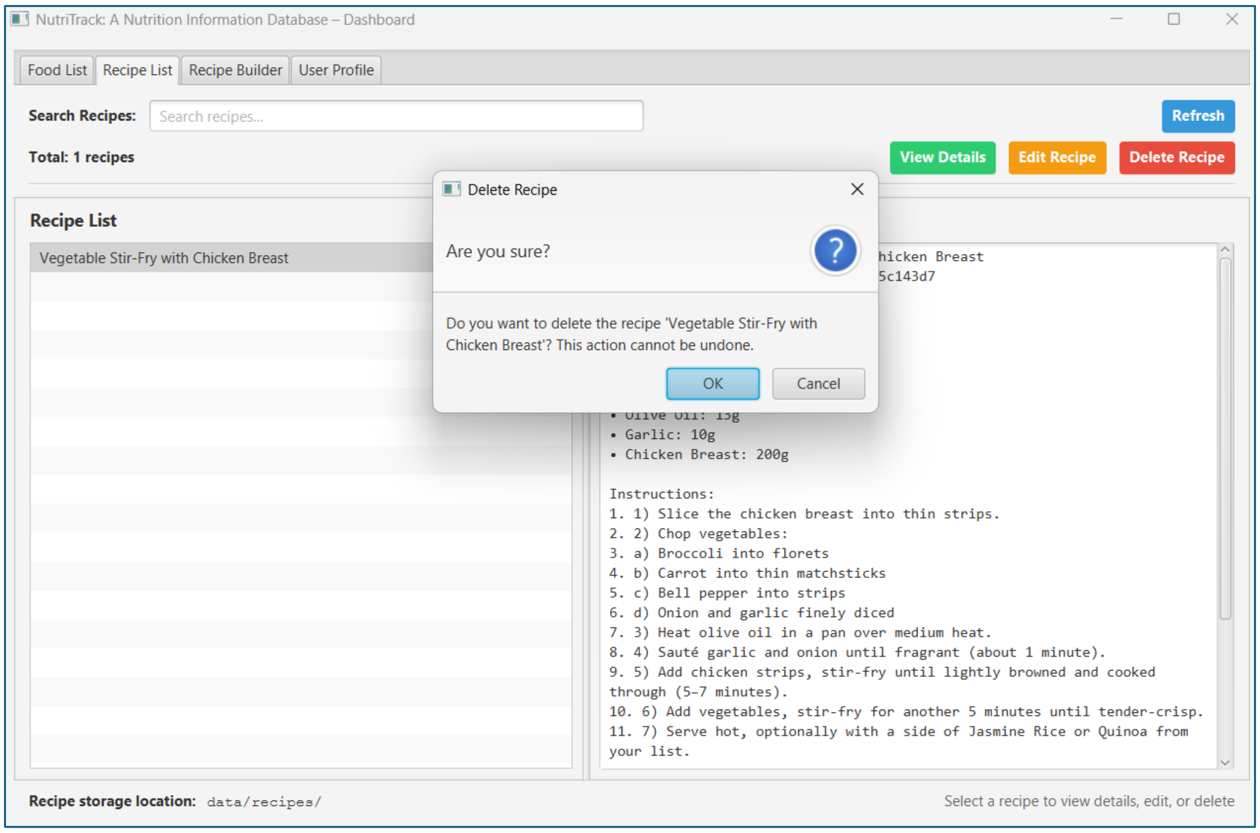


Please switch to the Recipe Editor tab to edit 'Vegetable Stir-Fry with Chicken Breast'.

```
hicken Breast
5c143d7

• Olive Oil: 15g
• Garlic: 10g
• Chicken Breast: 200g

Instructions:
1. 1) Slice the chicken breast into thin strips.
2. 2) Chop vegetables:
3. a) Broccoli into florets
4. b) Carrot into thin matchsticks
5. c) Bell pepper into strips
6. d) Onion and garlic finely diced
7. 3) Heat olive oil in a pan over medium heat.
8. 4) Sauté garlic and onion until fragrant (about 1 minute).
9. 5) Add chicken strips, stir-fry until lightly browned and cooked
through (5–7 minutes).
10. 6) Add vegetables, stir-fry for another 5 minutes until tender-crisp.
11. 7) Serve hot, optionally with a side of Jasmine Rice or Quinoa from
your list.
```

(The screenshot above displays an Edit Recipe pop-up dialog, asking the user to switch to the Recipe Builder tab for editing the recipe.)

For the Delete Recipe button, when it is clicked, a pop-up dialog asks the user to reconfirm before deleting the selected recipe:



Total: 1 recipes

Vegetable Stir-Fry with Chicken Breast

Delete Recipe

Are you sure?

Do you want to delete the recipe 'Vegetable Stir-Fry with Chicken Breast'? This action cannot be undone.

OK Cancel

```
hicken Breast
5c143d7

• Olive Oil: 15g
• Garlic: 10g
• Chicken Breast: 200g

Instructions:
1. 1) Slice the chicken breast into thin strips.
2. 2) Chop vegetables:
3. a) Broccoli into florets
4. b) Carrot into thin matchsticks
5. c) Bell pepper into strips
6. d) Onion and garlic finely diced
7. 3) Heat olive oil in a pan over medium heat.
8. 4) Sauté garlic and onion until fragrant (about 1 minute).
9. 5) Add chicken strips, stir-fry until lightly browned and cooked
through (5–7 minutes).
10. 6) Add vegetables, stir-fry for another 5 minutes until tender-crisp.
11. 7) Serve hot, optionally with a side of Jasmine Rice or Quinoa from
your list.
```

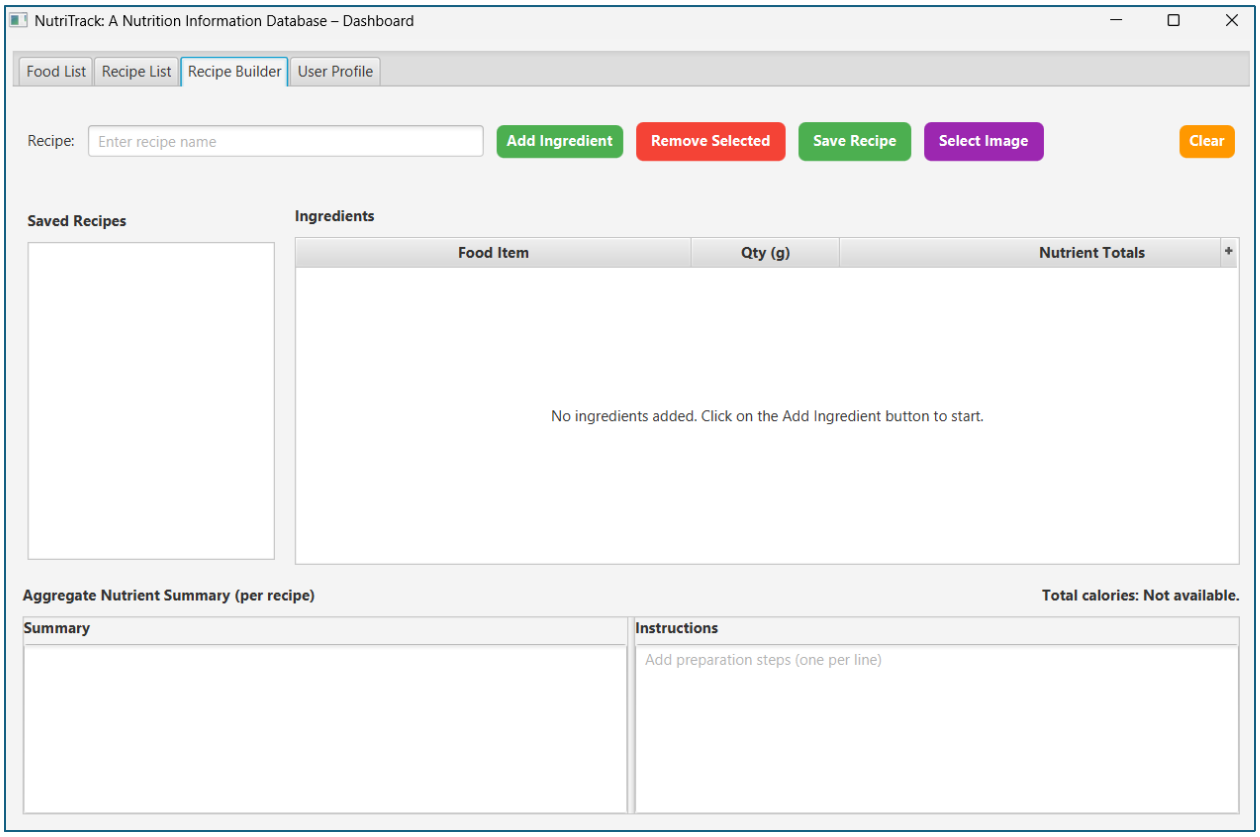
Recipe storage location: `data/recipes/`

Select a recipe to view details, edit, or delete

(The screenshot above is showing the pop-up Delete Recipe dialog for the user to confirm before deleting the selected recipe.)

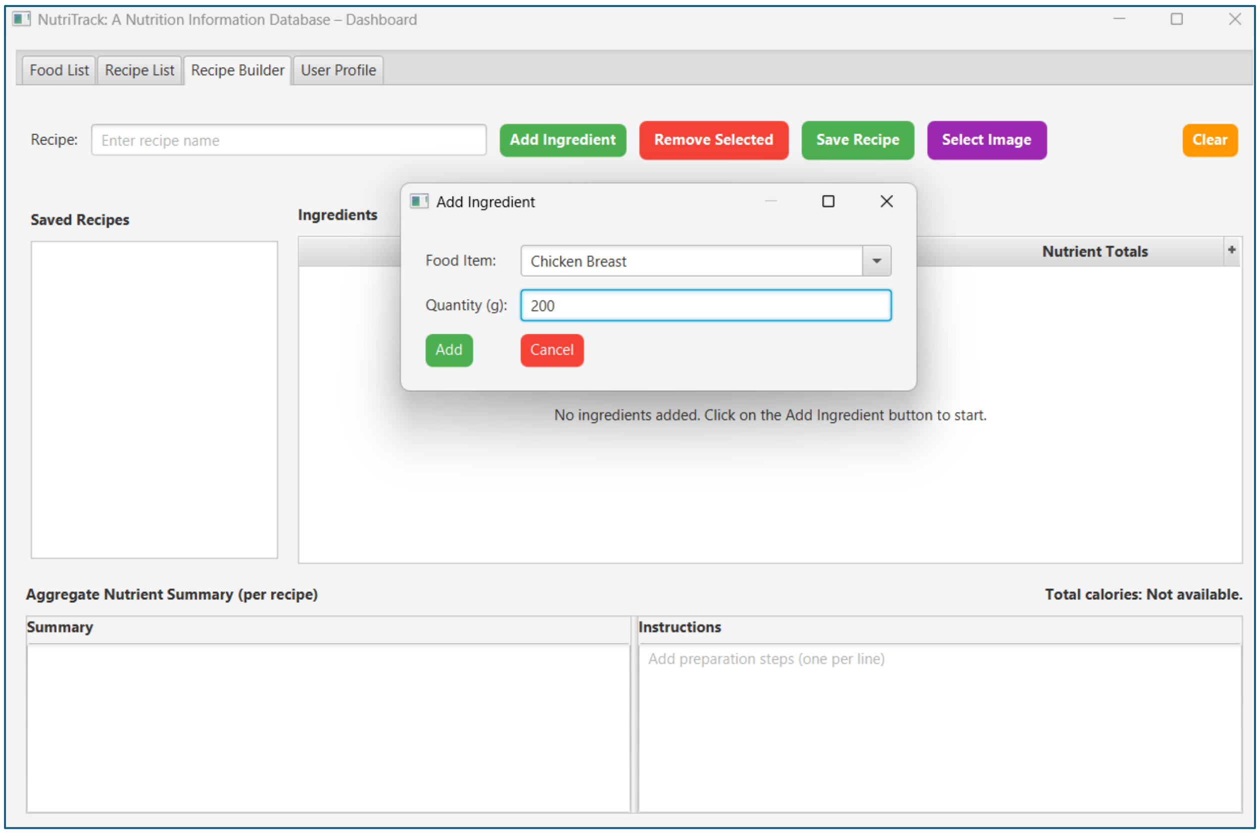
<u>Recipe Builder Management</u>

The Recipe Builder tab is responsible for generating new recipes from the user. Users can select the food items to be inserted into the recipe, set its quantity to be added, insert the relevant recipe image, view the total calories and aggregate nutrient summary per recipe, as well as inputting the cooking instructions.


NutriTrack: A Nutrition Information Database – Dashboard
Food List
Recipe List
Recipe Builder
User Profile
Recipe:
Enter recipe name
Add Ingredient
Remove Selected
Save Recipe
Select Image
Clear
Saved Recipes
Ingredients
Food Item
Qty (g)
Nutrient Totals
No ingredients added. Click on the Add Ingredient button to start.
Aggregate Nutrient Summary (per recipe)
Total calories: Not available.
Summary
Instructions
Add preparation steps (one per line)

(This screenshot displays the Recipe Builder tab where users can create their own recipes by choosing ingredients, uploading the relevant recipe image, view the aggregate nutrient summary, and customising the cooking instructions.)

To create a recipe, the user should click on the "Add Ingredient" button to select the necessary ingredients, let's say I were to create a recipe named "Mediterranean Spinach & Tomato Salad with Garlic-Olive Oil Dressing." I would insert the relevant food items through this pop-up dialog named "Add Ingredient:"



Food Item: Chicken Breast

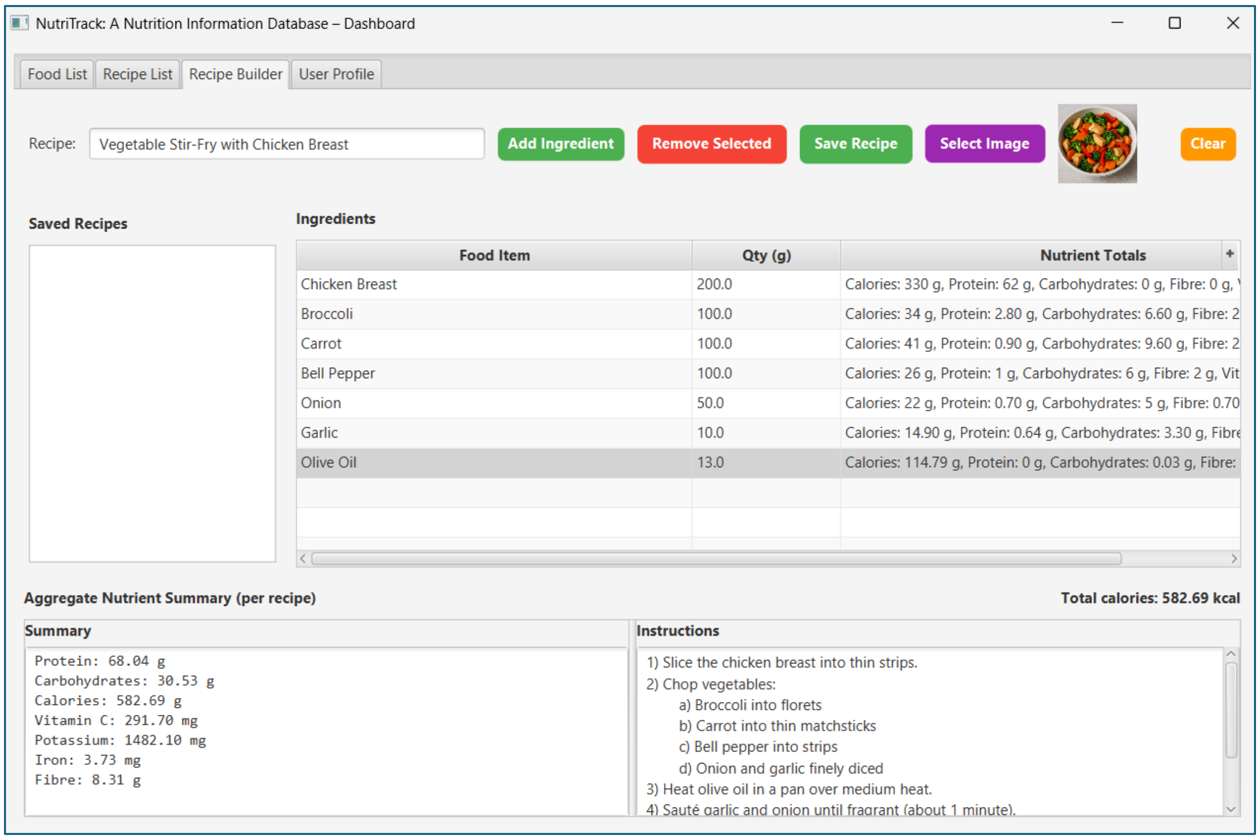
Quantity (g): 200

No ingredients added. Click on the Add Ingredient button to start.

Total calories: Not available.

(The screenshot above displays the Add Ingredient pop-up dialog, where I chose chicken breast as a food item through the pull-down menu/typing it, and indicate the appropriate quantity.)

After the necessary information (e.g., recipe name, ingredients, recipe image, and cooking instructions) have been inserted into the respective input boxes and ingredients chosen, the user will have to click on the “Save Recipe” button for the recipe to be successfully logged.



Recipe: Vegetable Stir-Fry with Chicken Breast

Food Item	Qty (g)	Nutrient Totals
Chicken Breast	200.0	Calories: 330 g, Protein: 62 g, Carbohydrates: 0 g, Fibre: 0 g, V
Broccoli	100.0	Calories: 34 g, Protein: 2.80 g, Carbohydrates: 6.60 g, Fibre: 2
Carrot	100.0	Calories: 41 g, Protein: 0.90 g, Carbohydrates: 9.60 g, Fibre: 2
Bell Pepper	100.0	Calories: 26 g, Protein: 1 g, Carbohydrates: 6 g, Fibre: 2 g, Vit
Onion	50.0	Calories: 22 g, Protein: 0.70 g, Carbohydrates: 5 g, Fibre: 0.70
Garlic	10.0	Calories: 14.90 g, Protein: 0.64 g, Carbohydrates: 3.30 g, Fibre
Olive Oil	13.0	Calories: 114.79 g, Protein: 0 g, Carbohydrates: 0.03 g, Fibre:

Aggregate Nutrient Summary (per recipe) Total calories: 582.69 kcal

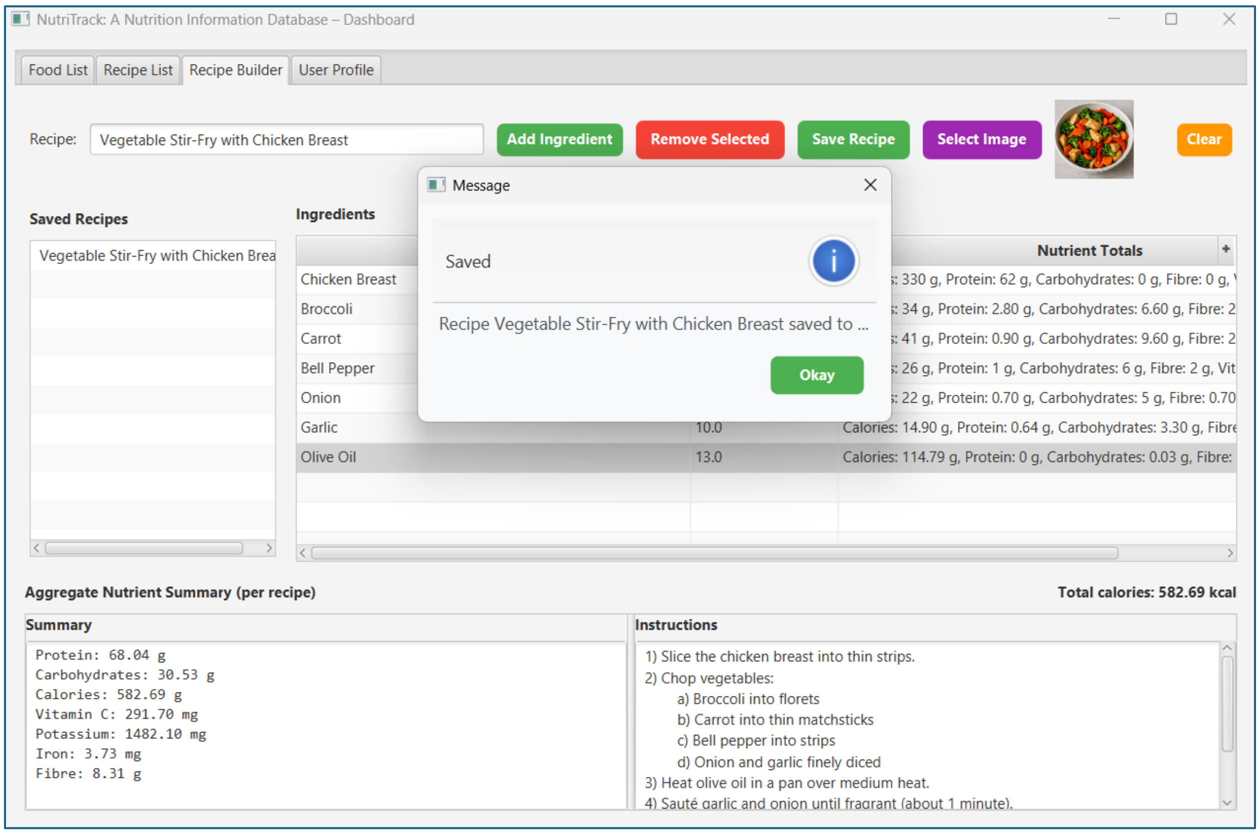
Summary

```
Protein: 68.04 g
Carbohydrates: 30.53 g
Calories: 582.69 g
Vitamin C: 291.70 mg
Potassium: 1482.10 mg
Iron: 3.73 mg
Fibre: 8.31 g
```

Instructions

1) Slice the chicken breast into thin strips.
2) Chop vegetables:
   a) Broccoli into florets
   b) Carrot into thin matchsticks
   c) Bell pepper into strips
   d) Onion and garlic finely diced
3) Heat olive oil in a pan over medium heat.
4) Sauté garlic and onion until fragrant (about 1 minute).

(The screenshot above displays the inserted information for the recipe before clicking on the "Save Recipe" button. As you can see, the summary of nutrients for the entire recipe and the total calories is automatically calculated and displayed.)



Recipe: Vegetable Stir-Fry with Chicken Breast

Saved Recipes: Vegetable Stir-Fry with Chicken Brea

Saved

Recipe Vegetable Stir-Fry with Chicken Breast saved to ...

Food Item	Qty (g)	Nutrient Totals
Chicken Breast		: 330 g, Protein: 62 g, Carbohydrates: 0 g, Fibre: 0 g, V
Broccoli		: 34 g, Protein: 2.80 g, Carbohydrates: 6.60 g, Fibre: 2
Carrot		: 41 g, Protein: 0.90 g, Carbohydrates: 9.60 g, Fibre: 2
Bell Pepper		: 26 g, Protein: 1 g, Carbohydrates: 6 g, Fibre: 2 g, Vit
Onion		: 22 g, Protein: 0.70 g, Carbohydrates: 5 g, Fibre: 0.70
Garlic	10.0	Calories: 14.90 g, Protein: 0.64 g, Carbohydrates: 3.30 g, Fibre
Olive Oil	13.0	Calories: 114.79 g, Protein: 0 g, Carbohydrates: 0.03 g, Fibre:

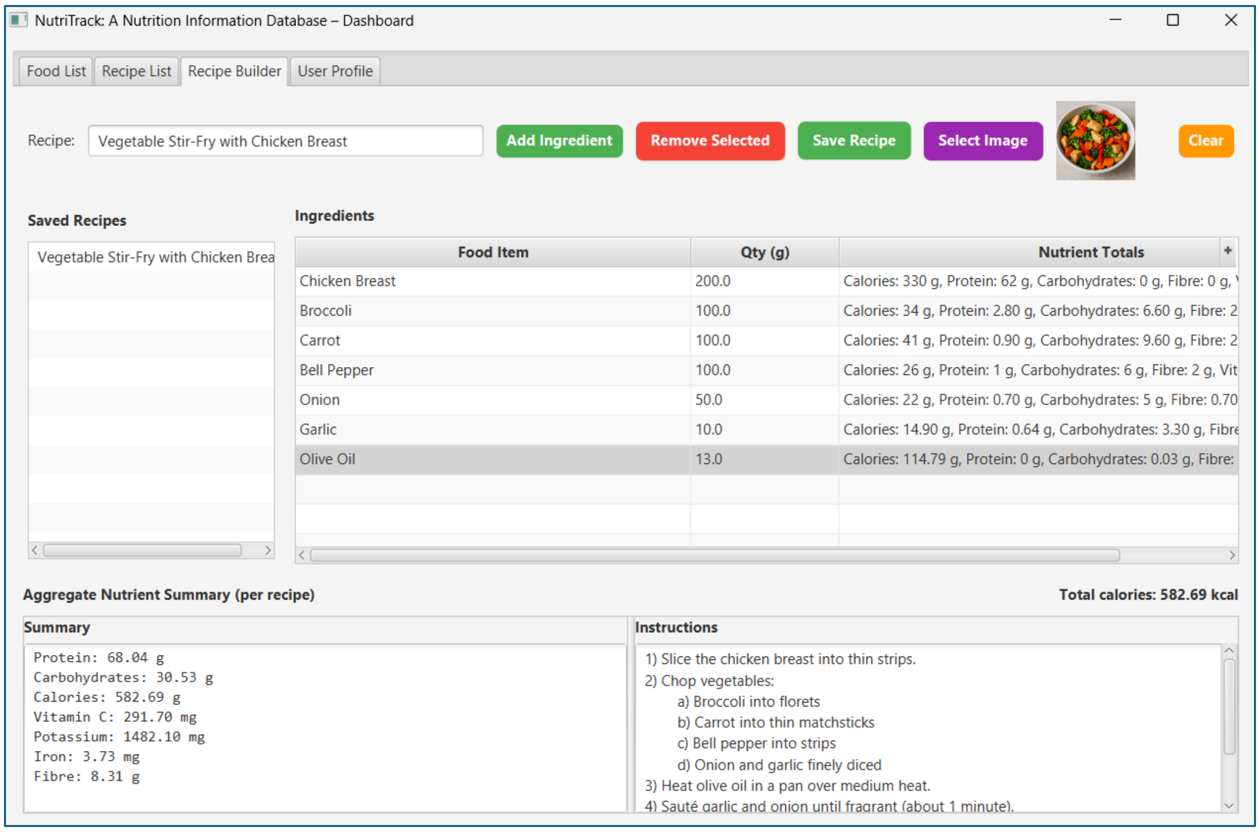
Aggregate Nutrient Summary (per recipe) Total calories: 582.69 kcal

Summary

```
Protein: 68.04 g
Carbohydrates: 30.53 g
Calories: 582.69 g
Vitamin C: 291.70 mg
Potassium: 1482.10 mg
Iron: 3.73 mg
Fibre: 8.31 g
```

Instructions

1) Slice the chicken breast into thin strips.
2) Chop vegetables:
   a) Broccoli into florets
   b) Carrot into thin matchsticks
   c) Bell pepper into strips
   d) Onion and garlic finely diced
3) Heat olive oil in a pan over medium heat.
4) Sauté garlic and onion until fragrant (about 1 minute).



Recipe: Vegetable Stir-Fry with Chicken Breast

Saved Recipes

Vegetable Stir-Fry with Chicken Brea

Ingredients

Food Item	Qty (g)	Nutrient Totals
Chicken Breast	200.0	Calories: 330 g, Protein: 62 g, Carbohydrates: 0 g, Fibre: 0 g, V
Broccoli	100.0	Calories: 34 g, Protein: 2.80 g, Carbohydrates: 6.60 g, Fibre: 2
Carrot	100.0	Calories: 41 g, Protein: 0.90 g, Carbohydrates: 9.60 g, Fibre: 2
Bell Pepper	100.0	Calories: 26 g, Protein: 1 g, Carbohydrates: 6 g, Fibre: 2 g, Vit
Onion	50.0	Calories: 22 g, Protein: 0.70 g, Carbohydrates: 5 g, Fibre: 0.70
Garlic	10.0	Calories: 14.90 g, Protein: 0.64 g, Carbohydrates: 3.30 g, Fibre
Olive Oil	13.0	Calories: 114.79 g, Protein: 0 g, Carbohydrates: 0.03 g, Fibre:

Aggregate Nutrient Summary (per recipe) **Total calories: 582.69 kcal**

Summary

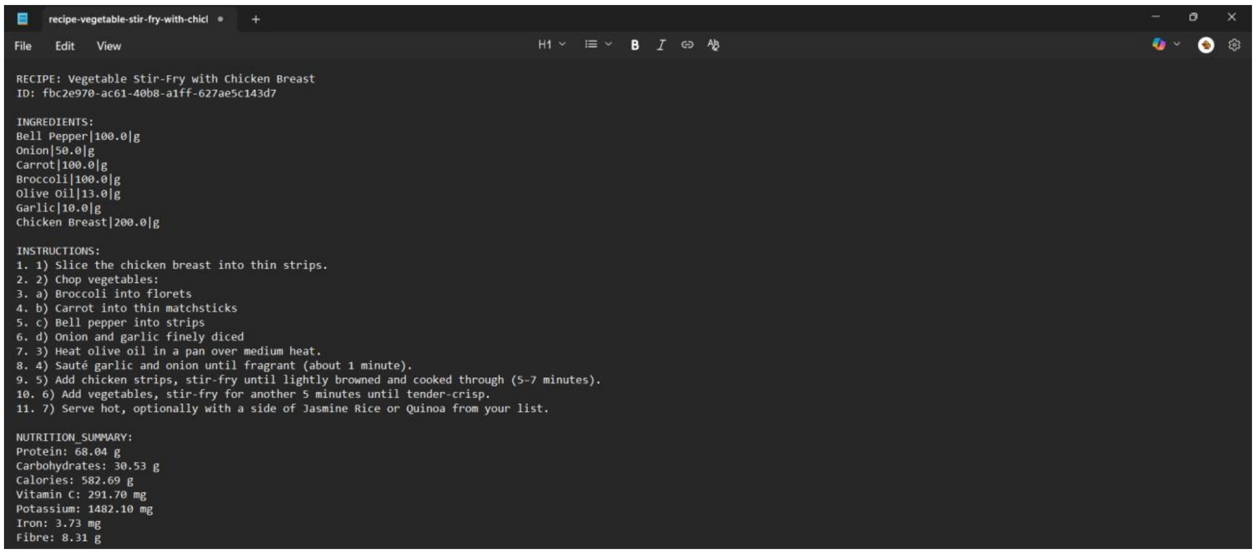
```
Protein: 68.04 g
Carbohydrates: 30.53 g
Calories: 582.69 g
Vitamin C: 291.70 mg
Potassium: 1482.10 mg
Iron: 3.73 mg
Fibre: 8.31 g
```

Instructions

1) Slice the chicken breast into thin strips.
2) Chop vegetables:
   a) Broccoli into florets
   b) Carrot into thin matchsticks
   c) Bell pepper into strips
   d) Onion and garlic finely diced
3) Heat olive oil in a pan over medium heat.
4) Sauté garlic and onion until fragrant (about 1 minute).

(The two screenshots above display the pop-up message dialog to inform the user that the recipe has been successfully created and stored in the recipes folder in a .txt file format. In the second screenshot, you can see the saved recipe at the left side of the dashboard. Users can select from the list of existing recipes to update its contents.)

In the recipes folder (automatically created if not available), a .txt file named recipe-mediterranean-spinach-tomato-salad-with-garlic-olive-oil-dressing.txt is created with the information obtained from the user when inputting them:

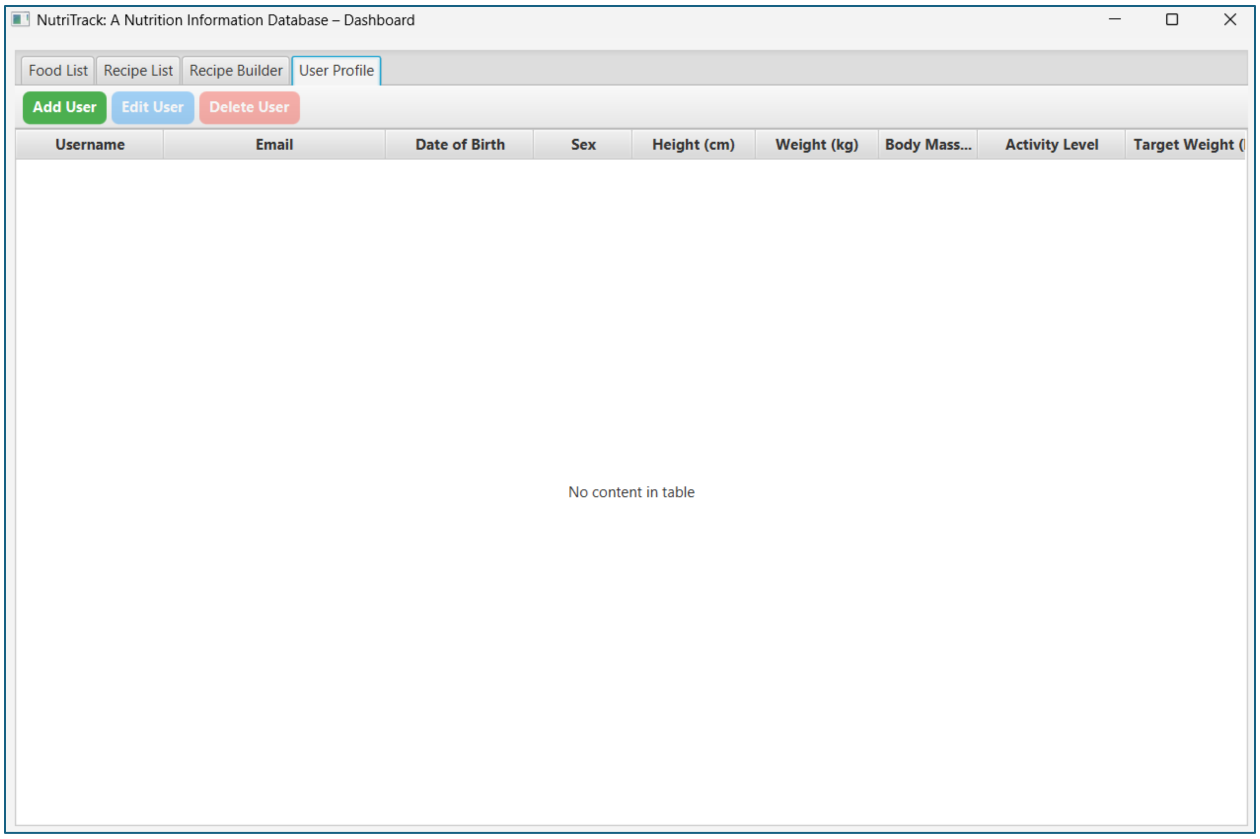


```
RECIPE: Vegetable Stir-Fry with Chicken Breast
ID: fbc2e970-ac61-40b8-a1ff-627ae5c143d7

INGREDIENTS:
Bell Pepper|100.0|g
Onion|50.0|g
Carrot|100.0|g
Broccoli|100.0|g
Olive Oil|13.0|g
Garlic|10.0|g
Chicken Breast|200.0|g

INSTRUCTIONS:
1. 1) Slice the chicken breast into thin strips.
2. 2) Chop vegetables:
3. a) Broccoli into florets
4. b) Carrot into thin matchsticks
5. c) Bell pepper into strips
6. d) Onion and garlic finely diced
7. 3) Heat olive oil in a pan over medium heat.
8. 4) Sauté garlic and onion until fragrant (about 1 minute).
9. 5) Add chicken strips, stir-fry until lightly browned and cooked through (5-7 minutes).
10. 6) Add vegetables, stir-fry for another 5 minutes until tender-crisp.
11. 7) Serve hot, optionally with a side of Jasmine Rice or Quinoa from your list.

NUTRITION_SUMMARY:
Protein: 68.04 g
Carbohydrates: 30.53 g
Calories: 582.69 g
Vitamin C: 291.70 mg
Potassium: 1482.10 mg
Iron: 3.73 mg
Fibre: 8.31 g
```

(The screenshot above displays how the recipe information is stored in the .txt file. An additional identification (ID) field is generated automatically for the recipe to be uniquely identified.)

<u>User Profile Management</u>

This functionality lets users maintain their personal profiles by performing create, read, update, and delete operations. Each profile is stored as its own text file named <username>.txt inside the data/users directory, which the system will automatically create if it does not exist. Each file contains the serialised credentials, personal details, and dietary preferences for that user.

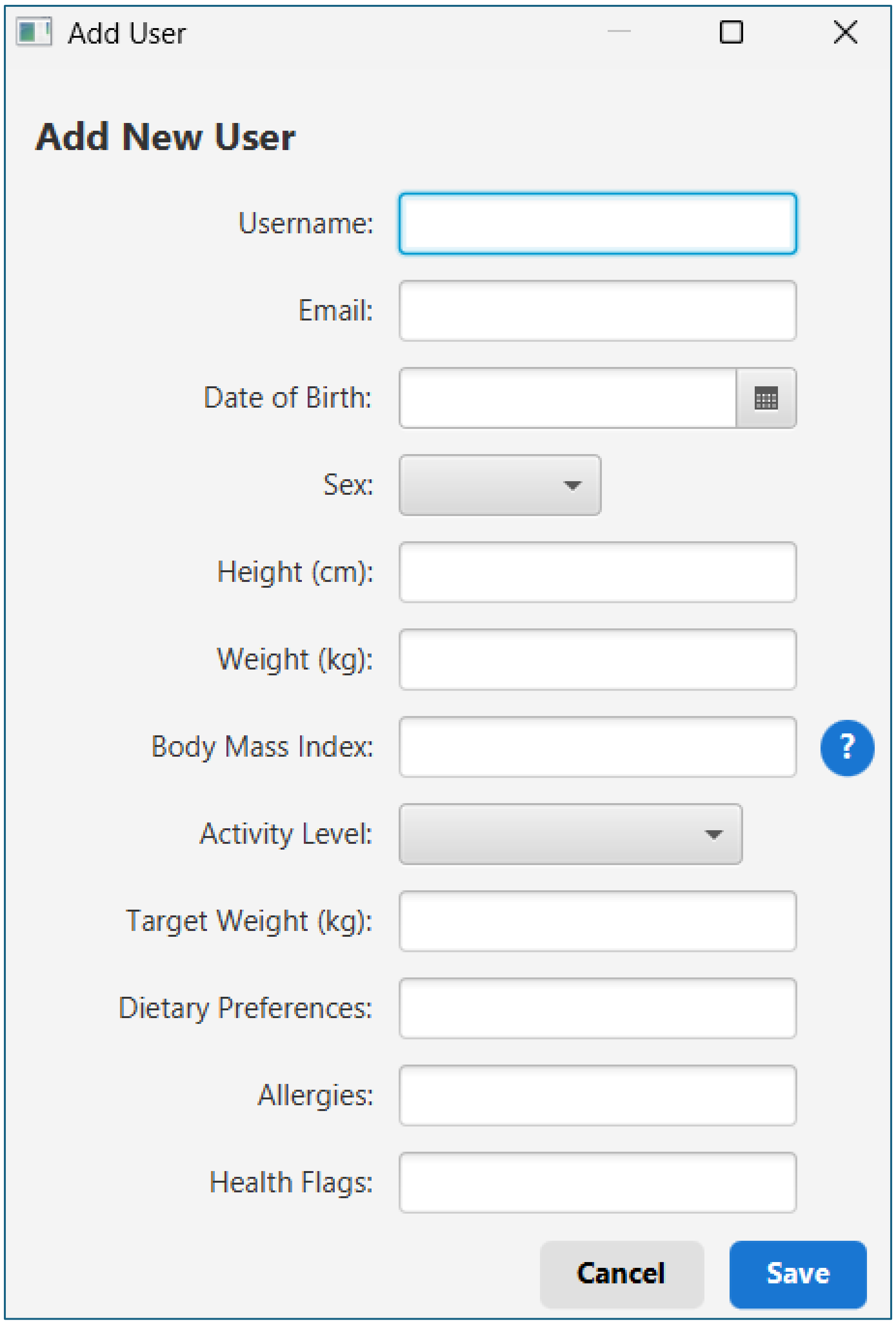


Username	Email	Date of Birth	Sex	Height (cm)	Weight (kg)	Body Mass...	Activity Level	Target Weight (

No content in table

(The screenshot above displays the User Profile tab dashboard, where users can insert their personal profile and relevant information, which will be stored in a .txt file for future usage when the application's functionality expands.)

To add a user, the user must click on the "Add User" button, which will display the Add User pop-up dialog box. The user will input their username, email address, date of birth, sex, height (in centimetres), weight (in kilograms), target weight (in kilograms), dietary preferences, allergies, health flags, and choose their activity level ranging from sedentary to extra active.

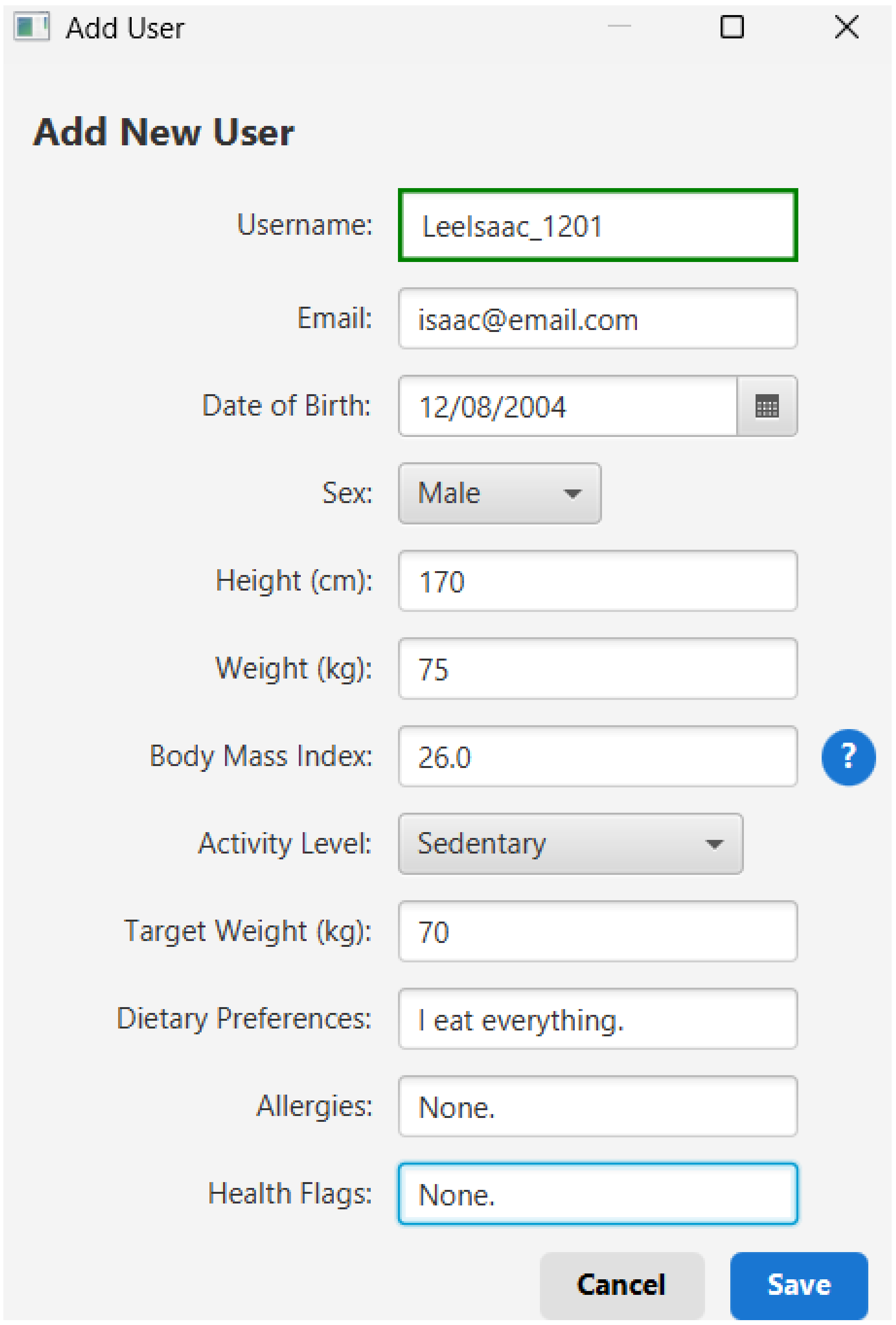

Add User

Add New User

Username:	
Email:	
Date of Birth:	
Sex:	
Height (cm):	
Weight (kg):	
Body Mass Index:	
Activity Level:	
Target Weight (kg):	
Dietary Preferences:	
Allergies:	
Health Flags:	

(The screenshot above shows the Add User pop-up dialog box for new users to be added. The new user must insert all personal information.)

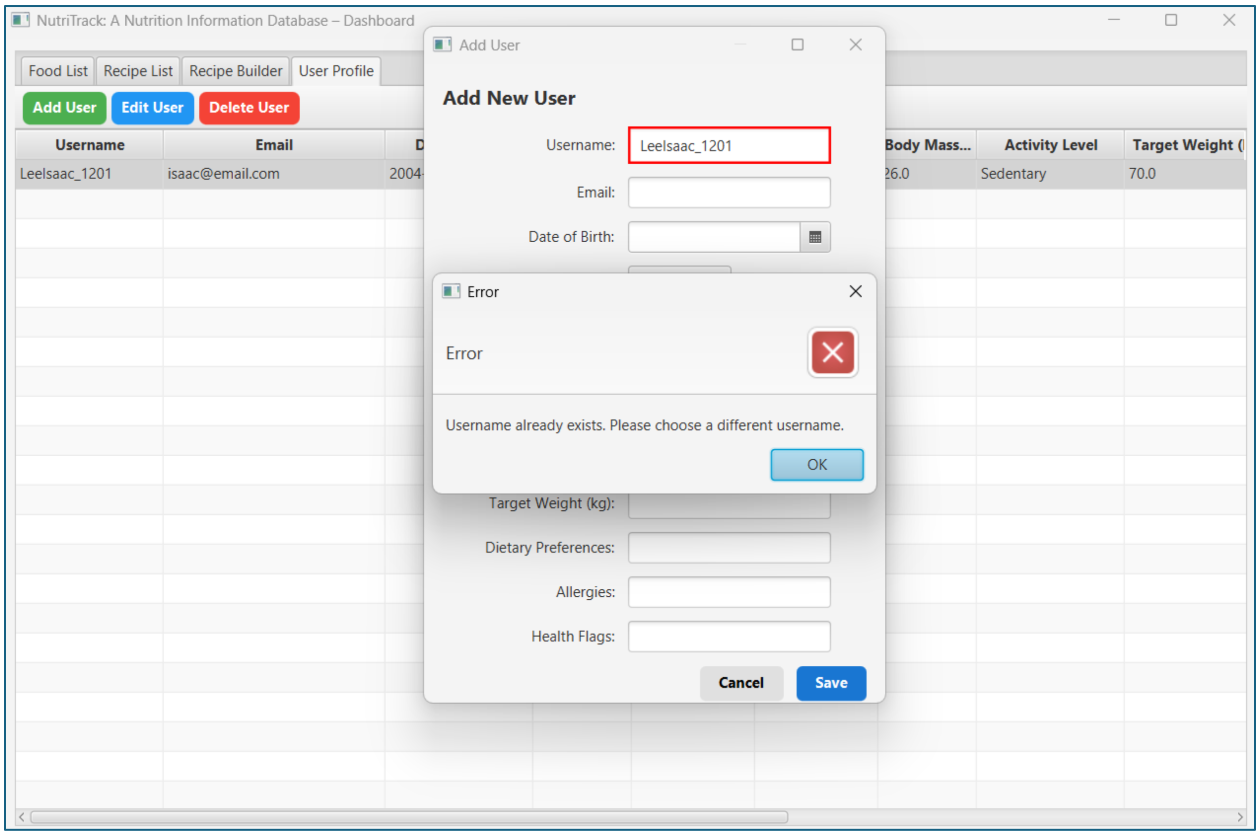
After inserting the following personal information, the user will notice that there's a green border in the "Username" field. This is the system's way to check for duplicate usernames. If a duplicate username is present, then the border box will turn red and would not allow the user to create a profile with said username and must choose an alternative:


Add User

Add New User

Username:	LeeIsaac_1201
Email:	isaac@email.com
Date of Birth:	12/08/2004
Sex:	Male
Height (cm):	170
Weight (kg):	75
Body Mass Index:	26.0
Activity Level:	Sedentary
Target Weight (kg):	70
Dietary Preferences:	I eat everything.
Allergies:	None.
Health Flags:	None.

(This Username field above has a green border box, allowing the user to create their profile using the inserted username.)


NutriTrack: A Nutrition Information Database – Dashboard

Username	Email	D	Body Mass...	Activity Level	Target Weight (
Leelsaac_1201	isaac@email.com	2004	26.0	Sedentary	70.0

Add User

Add New User

Username:	Leelsaac_1201
Email:	
Date of Birth:	
Target Weight (kg):	
Dietary Preferences:	
Allergies:	
Health Flags:	

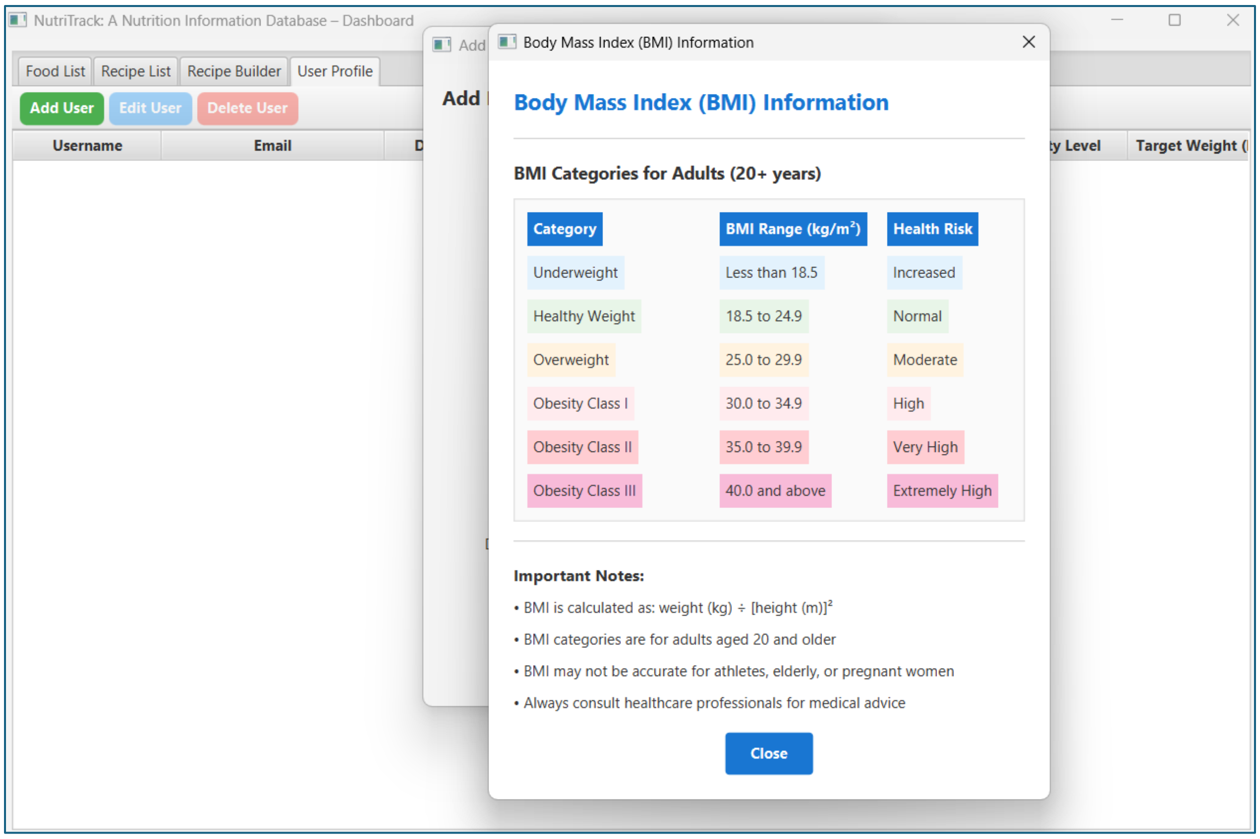
Error

Error

Username already exists. Please choose a different username.

(When a user wants to choose a username that has been taken, the border box will turn red colour and a pop-up dialog box will appear informing the user that he/she must choose a different username.)

Now, you may also notice that there's a white question mark symbol in a blue circle, located next to the body mass index (BMI) field. The BMI is not accessible to users to insert the numbers, as it is automatically calculated from the height and weight numbers inputted by the user. The blue circle with a white question mark beside is field is actually a button for a pop-up dialog box. When it is clicked, information related to a person's BMI is displayed to indicate the user's health condition.


NutriTrack: A Nutrition Information Database – Dashboard
Food List
Recipe List
Recipe Builder
User Profile
Add User
Edit User
Delete User
Body Mass Index (BMI) Information
Close

Body Mass Index (BMI) Information

BMI Categories for Adults (20+ years)

Category	BMI Range (kg/m²)	Health Risk
Underweight	Less than 18.5	Increased
Healthy Weight	18.5 to 24.9	Normal
Overweight	25.0 to 29.9	Moderate
Obesity Class I	30.0 to 34.9	High
Obesity Class II	35.0 to 39.9	Very High
Obesity Class III	40.0 and above	Extremely High

Important Notes:

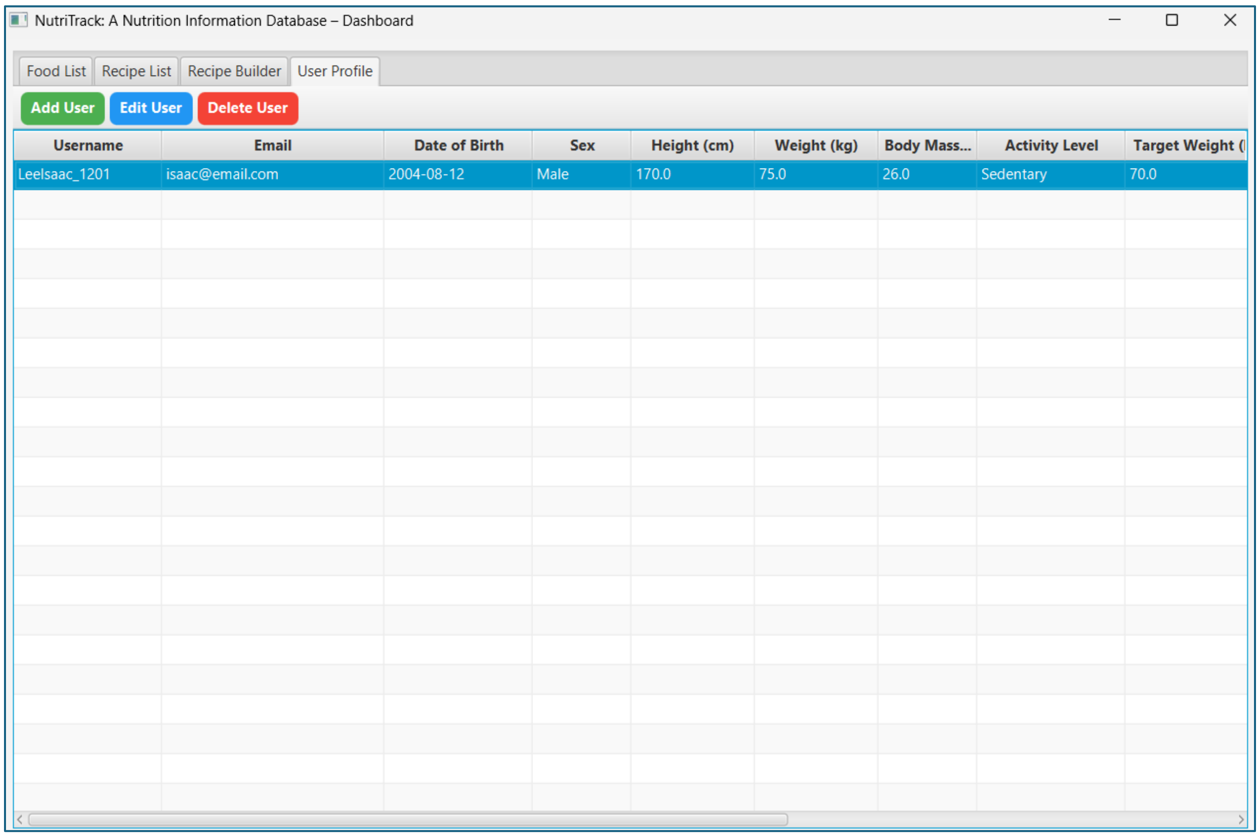
• BMI is calculated as: weight (kg) ÷ [height (m)]²

• BMI categories are for adults aged 20 and older

• BMI may not be accurate for athletes, elderly, or pregnant women

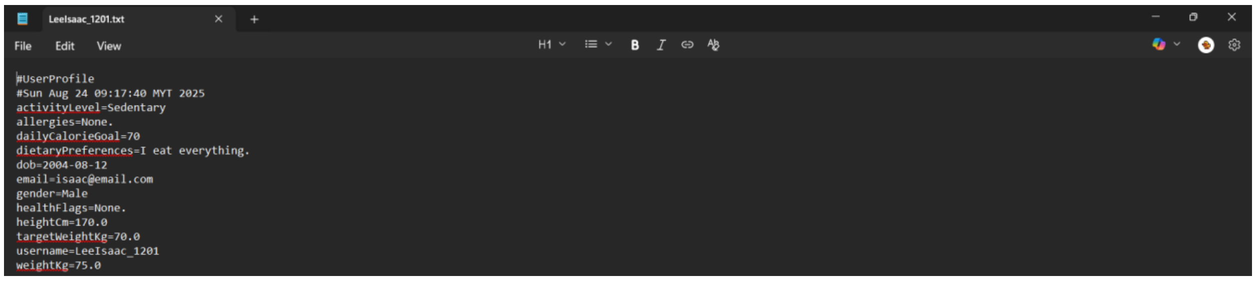
• Always consult healthcare professionals for medical advice

(The screenshot above displays the BMI information after clicking on the circular blue button with a white question mark in it. Users can compare their automatically generated BMI against this table to assess their health performance.)



Username	Email	Date of Birth	Sex	Height (cm)	Weight (kg)	Body Mass...	Activity Level	Target Weight (
LeeIsaac_1201	isaac@email.com	2004-08-12	Male	170.0	75.0	26.0	Sedentary	70.0

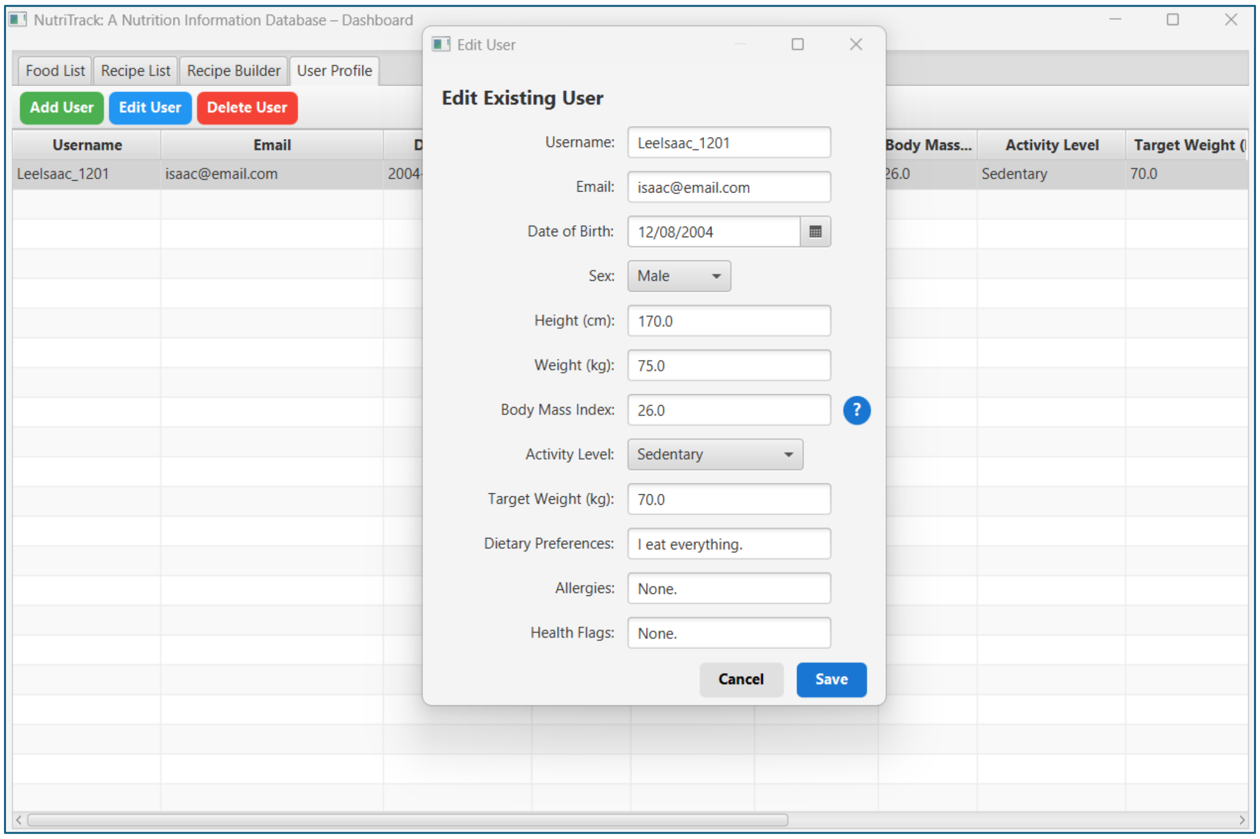
(The screenshot above displays the aftermath when a user has successfully created their profile. The information is also stored in a .txt file titled with the user's username.)



```
#UserProfile
#Sun Aug 24 09:17:40 MYT 2025
activityLevel=Sedentary
allergies=None.
dailyCalorieGoal=70
dietaryPreferences=I eat everything.
dob=2004-08-12
email=isaac@email.com
gender=Male
healthFlags=None.
heightCm=170.0
targetWeightKg=70.0
username=LeeIsaac_1201
weightKg=75.0
```

(The screenshot above displays the list of information entered by the user and is stored in a .txt file with the username as the title.)

To edit existing user details, the user will have to select on a user to edit and click on the "Edit User" button, then the Edit User pop-up dialog will appear with pre-filled existing user information:



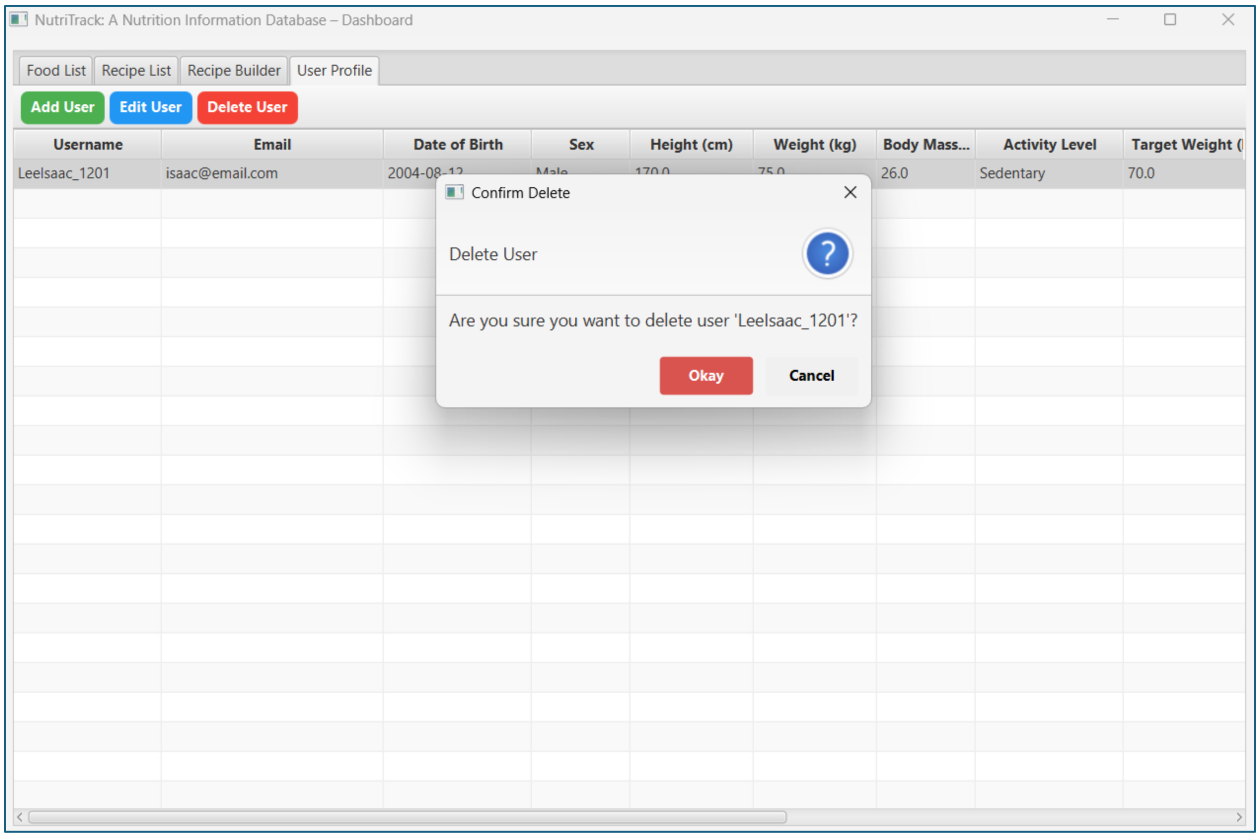
Username	Email	D	Body Mass...	Activity Level	Target Weight (
LeeIsaac_1201	isaac@email.com	2004	26.0	Sedentary	70.0

Edit Existing User

Username:	LeeIsaac_1201
Email:	isaac@email.com
Date of Birth:	12/08/2004
Sex:	Male
Height (cm):	170.0
Weight (kg):	75.0
Body Mass Index:	26.0
Activity Level:	Sedentary
Target Weight (kg):	70.0
Dietary Preferences:	I eat everything.
Allergies:	None.
Health Flags:	None.

(The screenshot above displays the Edit User pop-up dialog. Existing user information is pre-filled for convenience and ease of viewing. After editing the user information and the "Save" button is clicked, the information on the User Profile dashboard and the relevant .txt file will be updated.)

And finally, the option to delete a user profile. The user profile must be selected before clicking on the "Delete User" button. This is to prevent a user's profile from being accidentally deleted, as it cannot be restored and the .txt file related to the user will be destroyed.



Username	Email	Date of Birth	Sex	Height (cm)	Weight (kg)	Body Mass...	Activity Level	Target Weight (
LeeIsaac_1201	isaac@email.com	2004-08-12	Male	170.0	75.0	26.0	Sedentary	70.0

Confirm Delete

Delete User

Are you sure you want to delete user 'LeeIsaac_1201'?

(A screenshot displaying the Confirmation pop-up dialog before the user profile can be deleted.)

Applications of Object-Oriented Programming Concepts

Inheritance

Inheritance in Scala is a foundational concept of object-oriented programming that allows the subclass to acquire fields and methods of the superclass. By using the "extends" keyword, a subclass reuses and builds upon the implementation of its parent class, which promotes code reusability and reducing duplication. The superclass acts as the base template whose features are inherited, while the subclass can introduce its own fields and methods to specialise behaviour. As such, this mechanism not only streamlines the creation of related classes but also establishes a clear hierarchical organisation of types (GeeksforGeeks, 2022).

Inheritance is found in the codebase named Category.scala, where sealed trait Category defines an abstract supertype and each case object (e.g., Fruit, Vegetable, Meat) inherits from it, giving closed, type-safe values and exhaustiveness checking in pattern matches. Another trait-based hierarchy exists for nutrients (the controller constructs Macronutrient, Micronutrient, Fibre, Sugar instances that act as concrete Nutrient subtypes), and FoodItem.scala composes those by storing a List[Nutrient]. Additionally, FoodItem itself is a case class, which implicitly inherits useful behaviour from Scala's Product/Serializable/AnyRef (automatic equals/hashCode/toString/copy) even it does not extend a domain trait. The controllers interact with JavaFX's class hierarchies (e.g., TextField, ChoiceBox, Stage, Alert) by programming to supertype APIs while concrete implementations supplied by the framework provide runtime behaviour. As a result, these inheritance relationships enable subtype polymorphism, safer pattern matching where sealed is used, and convenient data semantics via case classes.

Polymorphism

Polymorphism is defined as the capability of a single entity to take on and operate in multiple forms. This versatility is achieved in Scala through virtual (overridable) methods, overloaded functions, and overloaded operators. Polymorphism exists when a variable declared as a superclass type holds a subclass instance, enabling identical calls to dispatch to different implementations at runtime (GeeksforGeeks, 2023). There are two types of polymorphism that exists in this project, namely subtype (inclusion/runtime) polymorphism and parametric (generic) polymorphism. Both types can be found in AddEditUserController.scala code file.

The first type is subtype (inclusion/runtime) polymorphism, which appears where the code works with values through a common supertype while the concrete subtype, that is chosen at runtime provides the actual behaviour. In AddEditUserController.scala, that pattern shows up in two places: the JavaFX UI fields are declared with general component types (e.g., Label, TextField, ComboBox[String], DatePicker, Stage, Scene, Alert), and the controller simply calls methods on those types (like lblDialogTitle.setText(...) or dialogStage.close()). At runtime, the JavaFX framework supplies concrete objects (actual subclasses/implementations) that perform the real work, so method dispatch happens against those concrete instances. Moreover, the same concept applies to Option[UserProfile]: Option is a common supertype with the concrete subtypes Some and None, as the code builds and inspects Some(profile) and uses None-like behaviour (if (saved) currentProfile else None), threreby relying on the Option subtype hierarchy so the program can treat presence and absence of a value uniformly while different runtime instances (Some vs None) represent the two cases.

The second type of polymorphism is parametric (generic) polymorphism, which can be seen as types are written with parameters so the that same code or data structure can operate over many different concrete types while preserving static type safety. There are two examples viewed from the file. First, generic containers and controls, like Option[UserProfile], ComboBox[String], and methods like .map / .filter operate over parameterised container types, where currentProfile: Option[UserProfile] uses the same Option container type that could hold any T, but in AddEditUserController.scala it is specialised to UserProfile, giving null-safe presence/absence semantics with compile-time type checking. Second, the generic application programming interface (API) usage and typed loaders are generic method calls that let the API return a value already typed to the expected concrete UI class or controller type. For example, loader.load[javafx.scene.layout.VBox]() and loader.getController[BMIInfoDialogController]. Consequently, these generics let the same implementation work for many types while keeping the code type-safe and avoiding casts.

Abstract Class

Abstraction in Scala refers to the practice of exposing only an object's essential functionality while concealing its internal implementation details. It is achieved by abstract classes, which are declared with the abstract keyword and may contain both abstract methods, defined without a method body, and concrete methods with full implementations. Unlike traits, abstract classes cannot be multiplied inherited, so any Scala class may extend at most one abstract class using the extends keyword. Abstract classes provide a clear separation between interface and implementation, promote code reuse, and establish a structured foundation for more specialised subclasses, by defining shared behaviour and forcing subclasses to supply specific implementations for abstract methods (using def methodName() without a body) (GeeksforGeeks, 2025).

Abstraction is demonstrated in the dialog controller hierarchy where the abstract class AbstractDialogController.scala defines common behaviour for all dialog windows, such as storing a reference to the dialog's Stage and providing a reusable closeWindow method but not prescribe the concrete details of any single dialog. AddFoodController.scala extends this base class, inheriting the generic lifecycle support while layering on its own specialised logic for parsing user input, constructing FoodItem instances, and saving them to storage. This separation means that dialog-specific controllers can focus on domain concerns (like handling nutrients and categories) without duplicating boilerplate window-management code while JavaFX's FXML injection mechanism relies on abstract method contracts (@FXML handlers and fields) that decouple UI definitions from implementation. This approach of abstraction keeps the design modular where the reusable base ensures consistent window control, while each concrete subclass provides just the details needed for its specific dialog.

<u>Personal Reflection</u>

Throughout this semester, taking this PRG2104: Object-Oriented Programming subjects has been a yet meaningful continuation of my learning in programming. Having first taken PRG1203: Object-Oriented Programming Fundamentals, I built my foundation in object-oriented concepts such as classes, objects, and encapsulation using the Java language. This subject revisited many of these same concepts, but this time in greater depth and with a new programming language: Scala. The transition from Java to Scala was not easy as a lot of information was foreign to me, but it gave me the opportunity to see how object-oriented principles remain consistent across languages, even though their syntax and application can differ. Compared to PRG1203: Object-Oriented Programming Fundamentals, this subject pushed me to think more critically about abstraction, inheritance, polymorphism, and traits, and to apply them in more complex ways. I also gained practical experience with graphical user interface (GUI) programming using ScalaFX, which helped me understand how these principles are applied in developing interactive and user-friendly applications. This shift from learning fundamentals to applying them in advanced contexts was a big step forward in my learning.

The assignments and final project provided me with opportunities to put theory into practice. For example, in earlier assignments, I was able to demonstrate the use of inheritance, polymorphism, and abstraction in structured problem-solving. Later, the final project required me to integrate these concepts into a complete application with a functional GUI. This project not only tested my technical skills but also encouraged me to consider design decisions such as modularity, reusability, and maintainability of code. In completing it, I gained valuable experience in applying object-oriented principles in a practical, real-world context which will help me in my future endeavours, both in building my personal projects and professionally to gain valuable experience.

Although the subject was academically enriching, my learning journey was not without its difficulties. My tutorial lecturer, Mr Kisheen Rao a/l Gsangaya, was frequently absent throughout the semester, which made it difficult for my class to receive consistent guidance during tutorials. At times, this was frustrating, as I felt uncertain about certain topics that could have been clarified more quickly in direct discussion. However, I am grateful that Dr Chin Teck Min stepped in to provide replacement and substitution classes, ensuring that we were not left entirely unsupported. In addition, when I faced challenges completing the final project on time, Dr Chin approved my request for an extension until 24 August, which gave me the necessary time to refine and complete my work to a higher standard. On a personal level, these experiences forced me to persevere, rely on self-study through lecture materials, and engage in peer discussions. While the circumstances were less than ideal, they ultimately strengthened my independence as a learner and reinforced the importance of resilience when faced with obstacles.

Overall, PRG2104: Object-Oriented Programming has been a demanding yet rewarding subject that significantly advanced both my technical and personal growth. While PRG1203: Object-Oriented Programming Fundamentals introduced me to the fundamentals of object-oriented programming in Java, PRG2104: Object-Oriented Programming required me to apply those same principles in more advanced ways using Scala, particularly in the development of GUI applications. The subject has deepened my understanding of abstraction, modularity, and polymorphism, while also sharpening my problem-solving skills and adaptability, despite the challenges caused by inconsistent tutorial support.

Challenges & Solutions

Ensuring Reliable File and Directory Management

The first challenge I faced when developing this application was keeping my .fxml user interface files and their respective .scala controller files in sync. This is because I would often rename, delete a text field, and updating an event handler in the .scala controller file without updating the matching fx:id or onAction in the .fxml file, only to hit a NullPointerException or see an empty pane at runtime. Identifying each discrepancy required a detailed examination of each XML tags and Scala annotations individually, and this issue recurred repeatedly, wasting more valuable time as the number of screens increased while the application grew in size. To address this issue, I established a standard for each view–controller pair, ensuring that my .fxml file consistently corresponds to the relevant .scala controller file while implementing a Gradle build task that loads every FXML in a headless manner. As a result, if any @FXML field or handler method is absent or incorrectly named, the build process fails instantly which allows me to detect these errors in advance before moving on to other parts of the application.

Race Conditions on Concurrent File Operations

The second challenge I encountered when expanding the size of this application was that simultaneous file operations on the single food_items.txt storage quickly resulted in elusive race conditions. Whenever two threads tried to save, delete, or update at nearly the same moment, their writes would overlap or overwrite each other, leading to missing lines, incomplete entries, and obscure data corruption. Under light load, everything seemed fine, but as soon as background saves and user-driven updates occurred in parallel (e.g., during bulk imports or rapid UI interactions), the file would become unsynchronised and reproducing the exact interleaving bugs while increased my frustration in examining raw file differences and cluttered logs. To overcome this challenge, I implemented a JVM-level ReentrantReadWriteLock in the FoodItemStorage object. All write operations (save, delete, update) now obtain the exclusive write lock, ensuring that only one thread can modify the file at any given time, while loadAll acquires the shared read lock, allowing multiple reads to proceed concurrently, only waiting if a write is in progress.

Dual Mode Add/Edit User Dialog

The third challenge I had to deal with while increasing the complexity of this application involved the reuse of a single .fxml file and .scala controller class file for both the "Add User" and "Edit User" processes, all while avoiding the duplication of files or logic. The reason behind this was utilising two distinct controllers would have necessitated the maintenance of nearly identical code in parallel, which introduces the risk of discrepancies in validation rules, dialog layout, and field bindings as the form progressed. I required the same dialog to present "Add New User" when establishing a profile, transition smoothly to "Edit Existing User" when altering one, pre-filling all fields, bypassing the uniqueness verification for the original username, and subsequently save or cancel without the need to instantiate a second controller class. To resolve this problem, I implemented an isEditMode flag along with two auxiliary methods in AddEditUserController:

- setEditMode(editMode: Boolean) toggles the lblDialogTitle text and window title between add/edit modes and adjusts validation behavior based on whether the username can change.
- setProfile(profile: UserProfile) flips the controller into edit mode, stores the original username for save-time comparisons, populates every TextField, ComboBox, and DatePicker with the existing profile’s values, and calls updateBmi() to display the current BMI.

In my view code’s showUserDialog(existing: Option[UserProfile]), I provided None for additions or Some(selectedProfile) for modifications. Consequently, the same controller dynamically adjusts its title, form data, and validation criteria, thereby keeping the code, FXML, and the dialog’s functionality clear.

Areas of Improvement

User Experience (UX)/User Interface (UI): Making the Application Friendly and Efficient

Since testing experiences led to encountering inconsistent controls, ambiguous fields, and visual clutter that leads to fatigue, the interface is to be improved to ensure that common tasks are both clear and efficient. For example, having consistent button placement, unambiguous labels, inline validation messages, and informative tooltips for units, such as grams or kcal. Moreover, implementing logical keyboard behaviours (such as using Enter to submit, Esc to cancel, and double-clicking to access details), along with convenient shortcuts for frequently performed actions and responsive layouts, will contribute to a smoother UX. In addition, implementing a cohesive colour palette, legible typography, and subtle transitions will foster a more calming and professional aesthetic in terms of UI visual design.

Maintainability and Reliability: Strengthening the Codebase for Stability and Growth

Due to the fact that the codebase is plagued with testing challenges that is compounded by compiler warnings, ad-hoc input/output (I/O), and mixed responsibilities leading to frequent bugs and risky fixes, the maintainability and reliability must be constantly upheld to prevent errors from happening. Business logics should be separated into dedicated services (for instance, FoodService, UserProfileService) to ensure that controllers remain streamlined and amenable to unit testing. Furthermore, unnecessary 'var's are to be substituted with 'val's or immutable structures, refactor duplicated code into helper functions, and implement structured logging in place of simple print statements. At the same time, unit tests for parsing and storage should be established using temporary files, and more secure persistence strategies to guarantee long-term codebase stability.

Data Accuracy and Security: Ensuring Trustworthy and Safe Information Handling

Given that the application manages personal health information and nutritional data, the accuracy and security of information must be securely safeguarded to prevent unnecessary data leaks and upholding user trust. By strengthening data validation, inputs such as weight, height, and calorie counts remain within realistic limits, which avoids errors that could distort calculations or reports. At the same time, introducing backup and recovery systems protects against data loss or corruption, adding an additional layer of backup and recovery in case of system failures. The application will be able to offer users dependable insights while protecting their personal information, as implementing audit trails and clear error reporting will help track inconsistencies and make debugging easier.

References

Balaratnam, I. (2024, September 3). 5 tips for healthy eating on a budget. *Free Malaysia Today | FMT*. https://www.freemalaysiatoday.com/category/leisure/2024/09/03/5-tips-for-healthy-eating-on-a-budget

Chong, C. T., Lai, W. K., Zainuddin, A. A., Pardi, M., Sallehuddin, S. M., & Ganapathy, S. S. (2022). Prevalence of obesity and its associated factors among Malaysian adults: finding from the National Health and Morbidity Survey 2019. *Asia Pacific Journal of Public Health*, *34*(8), 786–792. https://doi.org/10.1177/10105395221129113

GeeksforGeeks. (2022, January 12). *Inheritance in scala*. GeeksforGeeks. https://www.geeksforgeeks.org/scala/inheritance-in-scala/

GeeksforGeeks. (2023, April 11). *Scala | Polymorphism*. GeeksforGeeks. https://www.geeksforgeeks.org/scala/scala-polymorphism/

GeeksforGeeks. (2025, July 11). *Abstract classes in scala*. GeeksforGeeks. https://www.geeksforgeeks.org/scala/abstract-classes-in-scala/

Goal 2 | Department of Economic and Social Affairs. (n.d.). https://sdgs.un.org/goals/goal2

Lee, W. S., Jalaludin, M. Y., Khoh, K. M., Kok, J. L., Nadarajaw, T., Soosai, A. P., Mukhtar, F., Fadzil, Y. J., Zaini, A. A., Mohd-Taib, S. H., Rosly, R. M., Khoo, A. J., & Cheang, H. K. (2022). Prevalence of undernutrition and associated factors in young children in Malaysia: A nationwide survey. *Frontiers in Pediatrics*, *10*. https://doi.org/10.3389/fped.2022.913850