

# National College of Ireland

Bachelor of Science (Honours) in Computing

Software Development

2022/2023

Ruby Lennon

X19128355

[x19128355@student.ncirl.ie](mailto:x19128355@student.ncirl.ie)

Food Planner App

Technical Report

## Contents

Executive Summary .....	2
1.0 Introduction .....	2
1.1. Background .....	2
1.2. Aims.....	3
1.3. Technology.....	4
1.4. Structure .....	6
2.0 System.....	7
2.1. Requirements.....	7
2.1.1. Functional Requirements.....	7
2.1.2 Data Requirements .....	25
2.1.3 User Requirements .....	25
2.1.4 Environmental Requirements .....	25
2.1.5 Usability Requirements.....	25
2.2 Design & Architecture .....	26
2.3 Implementation .....	30
2.3.1 User Management Feature (Login, Registration, Account Settings) .....	32
2.3.2 Ingredients List OCR Scanner Feature.....	36
2.3.3 Recipe Search Engine Feature.....	40
2.3.4 Recipe Manager Feature.....	42
2.3.5 Meal Planner & Shopping List Features .....	48
2.4 Graphical User Interface (GUI).....	51
2.5 Testing.....	65
2.5.1 Unit Testing.....	65
2.5.2 Integration Testing.....	69
2.5.3 System Testing .....	72
2.5.4 End-To-End Functionality Testing .....	76
2.6 Evaluation .....	77
3 Conclusions .....	81
4 Further Development or Research .....	82
5 References .....	83
6 Appendices.....	85
6.1 Project Proposal.....	85
6.2 Reflective Journals .....	95

## Executive Summary

The purpose of this report is to document the proposal, technology, requirements specification, design and architecture, implementation, testing, evaluation, and conclusions of the Android mobile software application, FoodPlannerApp. The FoodPlannerApp application aims to solve the following problems: food waste, obscure food product ingredients and unhealthy food habits due to hectic schedules. The project attempts to do this through providing users with the following features: User Account Management, a Food Ingredients Scanner, a Recipe Search Engine, a Recipe Manager, a Meal Planner, and a Food Shopping List Generator. Each feature corresponds with one of the six application functional requirements. The User Management feature allows users to create, manage and log in to their user account. The Ingredients Scanner feature allows users to scan images for ingredients text and receive the ingredients meaning. The Recipe Manager feature allows users to create, read, update, and delete recipes. The Meal Planner feature allows users to add recipes to their meal plan for a scheduled date which generates an ingredients shopping list as part of the Shopping List feature. Technologies, tools, and services such as Machine Learning, Optical Character Recognition, Google ML Kit Vision Text-Recognition API, Android Studio, Android Mobile Application Development, Java Programming, Versioning Control, Git, GitHub, Firebase Authentication, Firebase Realtime Database, and various Testing tools were all utilised in the implementation of this project. The project was planned and developed using an Agile Development approach. The projects correctness, security, performance, usability, accessibility, and internationalization were evaluated using lint scans, performance analysers, automated testing, and manual testing. The project was a success as all six functional requirements within the scope of the project were developed and implemented into a single working Android Mobile Application resulting in successful project completion.

## 1.0 Introduction

### 1.1. Background

The first problem that this project attempts to solve is food waste and its impact on the environment and on household expenditure. According to the EPA (The Environmental Protection Agency) Irish households threw away an estimated 221,000 tonnes of food in 2021. Food waste costs the average Irish household approximately €60 per month or €700 per year. It is estimated that food waste generates about 8% to 10% of global greenhouse gas emissions. (EPA, 2023) The second problem that this project attempts to solve is confusing obscure food product ingredients which causes difficulty for consumers to understand ingredients listed on food labels and their nutritional value. For example, manufacturers can avoid listing sugar and fats as the first ingredient by listing different forms of sugars and fats with different names. 'Dietary Free Sugars' are sugars added during food production or sugars naturally present in foods such as honey, syrup, and fruit juices. Higher intakes of free sugars have been linked to dental caries and overweight. (Dasgupta, et al., 2023) In a report published in the European Journal of Nutrition in 2019, it was found that 75% of Irish 3 year old children surveyed were exceeding the World Health Organisation (WHO) guidelines for free sugar intake. (Crowe, et al., 2019) The third problem that this project attempts to solve

is unhealthy food habits because of hectic schedules. In a report published by Bord Bia in 2020, due to a more time pressured lifestyle, only 30% of Irish adults surveyed prepared an evening meal using only fresh and raw ingredients every day or most days which was a 12% decrease from 2011. (Bord Bia, 2020) During the market research for the project many individual applications were found for meal planning, creating shopping lists, recipes finders, recipe managers, ingredients scanners and so on. What makes the 'Food Planner App' different than most food management applications is that it combines all these individual features into one all-encompassing food planner application. All the individual features work together to create a comprehensive solution for the user.

## 1.2. Aims

The 'Food Planner App' aims to solve the issue of food waste everywhere by supporting users in reducing their food waste and carbon footprint through providing them with the Recipe Search Engine feature which allows them to search for recipes containing ingredients they already own. Supporting users in reducing their food waste should also help them to save money. The 'Food Planner App' also aims to solve the issue of confusing food ingredients by providing the Ingredients Scanner Feature which allows users to upload a photo of a food product ingredients list and to receive a report of the ingredients and their meaning. The 'Food Planner App' aims to solve the problem with lack of meal planning due to busy schedules by allowing the user to organise their recipes, create a meal plan and view a food shopping list for each of their meals. The scope of the project is to develop a working Android Mobile Application which meet the following six functional requirements: Users must be able to create and manage a user account which will enable them to authenticate and use the application features using the User Management feature. Users must be able to create, read, update, and delete recipe records using the Recipe Manager feature. Users must be able to scan images of food product ingredients lists to receive the ingredients meaning using the Ingredients Scanner feature. Users must be able to search for public or owned recipes by ingredients, cuisine, suitability, or name using the Recipe Search feature. Users must be able to add recipes to their meal plan using the Meal Planner feature. Users must also be able to view and update a shopping list for each of their scheduled meals using the Shopping List feature. The project aims to develop the FoodPlannerApp making use of innovative technologies such as the Machine Learning process of Optical Character Recognition (OCR), Android Mobile Application Development and Backend Cloud computing services provided by Google's Firebase. The projects goals are to develop a working Mobile Application system with a high priority placed on technology, requirements specification, design and architecture, implementation, testing, evaluation, and maintainability.

### 1.3. Technology

#### **Android Studio / Android Mobile Application Development**

The project application was developed using Android Studio, the official Android App Development Integrated Development Environment (IDE). Android Studio uses a Gradle-based build system. The project was developed using Gradle Version 7.5. Android Studio provides and supports testing tools and frameworks such as Espresso Test Recorder, JUnit 4, and JUnit 5. These testing tools were used to automate the Unit tests, and some of the Integration and System tests. Android Studio provides a Lint tool which identifies problems with a projects structural code quality. The Lint tool checks a projects source files for potential bugs and suggests optimization improvements for correctness, security, performance, usability, accessibility, and internationalization. (Google For Developers, 2023) This Lint tool was used in the evaluation of the project. Android Studio offers a Performance Profiler tool to track an applications memory usage, energy usage and CPU usage. (Google For Developers, 2023) The Performance Profiler tool was used in the evaluation of the project. Android Studio can run Android applications virtually on a laptop using an Android Virtual Device (AVD) or by connecting a physical Android device in Developer Mode – both methods were used to run, test, and debug the application.

#### **Java**

The FoodPlannerApp Android application was developed in Android Studio using Gradle Java Development Kit Version 11. Java is a class-based and OOP (Object Oriented Programming) programming language designed to be simple, high-level, robust, secure, and object-oriented. (Oracle, no date)

#### **XML**

XML was used to define the User Interface (UI) layouts of the android application. XML (Extensible Markup Language) describes the UI layout of the application using a text-based document made up of XML tags, elements, and attributes. The XML files are then rendered by the application when run to display a Graphical User Interface (GUI) to the user.

#### **Google ML Kit Text Recognition API / Optical Character Recognition (OCR)**

Google's ML Kit is a mobile Software Development Kit (SDK) for providing Google's on-device Machine Learning (ML) services to mobile applications (Android and iOS). Google Provides Vision and Natural Language APIs. These APIs are based on Googles ML models. All ML Kit's APIs run on-device meaning that the functionality is available offline. (Google For Developers, 2023) Google's ML Kit Text Recognition v2 API recognises text from an input image. (Google For Developers, 2023). Google's ML Kit Text Recognition v2 API was used for the Ingredients Scanner feature of the application.

## **Firebase Authentication**

The app development platform, Firebase, provides the Firebase Authentication service used for creating and authenticating application users using SDK and premade UI libraries. The application uses Firebase Authentication as the application user authentication service. (Google For Developers, 2023)

## **Firebase Realtime Database**

Firebase provides the NoSQL cloud database, Firebase Realtime Database. Firebase Realtime Database is used as the application database. Firebase Realtime Database is a cloud-hosted database in which data is stored as JSON. Data in a Firebase Realtime Database is synced with all connected clients in real time through a single instance which results in users automatically receiving updated data in real time. (Google for Developers, 2023) The applications Recipe, Meals and Ingredient data are stored in the Firebase Project Realtime Database. The stored recipe data is used as part of the Recipe Manager and Recipe Search features, the ingredient data is used to populate the ingredients dialog in the Recipe Search feature and used to return ingredients meaning in the Ingredient Scanner feature. The meal data is used for the Meal Planner feature and the meal ingredient data is used for the Shopping List feature.

## **GitHub / Git**

GitHub is a software development and control platform which is built on the versioning control software Git. (GitHub, no date) GitHub was used as the main project repository for storing and managing the applications source code. When changes were made to the projects source code, a branch would first be checked out. Code changes were then pushed to the development branch using Git terminal commands. When the changes were ready to be merged to the main branch, a pull request was created and then subsequently merged.

## **Espresso / JUnit 4 / JUnit 5**

JUnit is a unit testing framework used in Java programming. JUnit 4 is a test framework used to write repeatable tests. The JUnit 4 test framework is an instance xUnit architecture for unit testing frameworks. (JUnit, 2021) JUnit 4 was used in conjunction with Android Espresso to run automated UI tests. JUnit 5 is the next generation of JUnit after JUnit 4. JUnit 5 is made up of the JUnit Platform, JUnit Jupiter, and JUnit Vintage. (Bechtold, et al., no date) JUnit 5 was used to create and run unit test.

## 1.4. Structure

Section	Overview
<b>Executive Summary</b>	This section addresses the key points of this technical report.
<b>1.0 Introduction</b>	This section addresses the Background, Aims and Technology of the project and the Technical Report Structure.
<b>1.1 Background</b>	This section addresses the reasons why the project was undertaken.
<b>1.2 Aims</b>	This section addresses the projects aims and objectives.
<b>1.3 Technology</b>	This section addresses which technologies were used, and how they were used, to achieve the projects aims.
<b>1.4 Structure</b>	This section provides an overview of the report sections.
<b>2.0 System</b>	This section addresses the project Systems requirements, design & architecture, implementation, graphical user interface, testing, and evaluation of the project.
<b>2.1 Requirements</b>	This section addresses the functional, data, user, environment, and usability requirements of the projects system.
<b>2.1.1 Functional Requirements</b>	This section of the report addresses all the functional requirements of the project system. For each functional requirement a use case diagram, description & priority, and use case has been included in this section.
<b>2.1.2 Data Requirements</b>	This section addresses the project systems requirements in relation to data.
<b>2.1.3 User Requirements</b>	This section addresses the project systems requirements in relation to users.
<b>2.1.4 Environment Requirements</b>	This section addresses the project systems requirements in relation to the system environment.
<b>2.1.5 Usability Requirements</b>	This section addresses the project systems requirements in relation to the system's usability.
<b>2.2 Design &amp; Architecture</b>	This section addresses the systems design, architecture and components used.
<b>2.3 Implementation</b>	This section addresses the systems implementation and describes the main code algorithms, classes and functions used.
<b>2.4 Graphical User Interface (GUI)</b>	This section includes wireframes and screenshots of the final key screens in the application.
<b>2.5 Testing</b>	This section addresses the testing tools, test cases, test plans, and types of testing used in the testing of the project.
<b>2.6 Evaluation</b>	This section addresses how the system was evaluated and the results.
<b>3.0 Conclusions</b>	This section addresses the projects advantages, disadvantages, results, strengths, and limitations.
<b>4.0 Further Development/Research</b>	This section addresses the direction the project would take with additional time and resources.
<b>5.0 References</b>	This section contains the bibliography for all references cited throughout the document.
<b>6.0 Appendices</b>	This section contains the Project Proposal document and Monthly Reflective Journals.

## 2.0 System

### 2.1. Requirements

#### 2.1.1. Functional Requirements

1. **FR1 – User Management:**

The software system must accomplish user authentication. This functional requirement will allow the End User to register for a user account, login to their user account, edit their account details and or delete their account. Google's Firebase User Authentication is the backend service that is going to be used for the application user authentication.

2. **FR2 - Ingredient List OCR Scanner:**

The software system must accomplish text recognition OCR functionality. This functional requirement will allow the End User to upload an image of a food ingredients list and receive a breakdown of the food ingredients and their meaning. Text from this image will then be extracted using Google's ML Kit Vision (Text Recognition) API. The ingredients definitions will be stored in and queried from an external Google Firebase Realtime Database.

3. **FR3 - Recipe Search Engine:**

The software system must accomplish recipe search functionality. This functional requirement will allow the End User to search for recipes by parameters such as name, ingredients, cuisine, and suitability. In the Project Proposal I noted that I may use web scraping to extract recipes from Google's SERP. However, I am considering an alternative solution where the recipe search engine queries user created recipes from the application primary Google Firebase Realtime Database instead. This will be decided once the development of this feature is started.

4. **FR4 - Recipe Manager:**

The software system must accomplish a recipe management system. This functional requirement will allow the End User to create, read, update, and delete private or public recipes. Public recipes will be visible to all users and retrievable using the Recipe Search Engine. The recipes will be stored externally using a Google Firebase Realtime Database.

5. **FR5 - Meal Plan Manager:**

The software system must accomplish a meal planner functionality. This functional requirement will allow the End User to add and remove recipes to and from their weekly meal plan. The meal plan information will be stored externally using a Google Firebase Realtime Database.

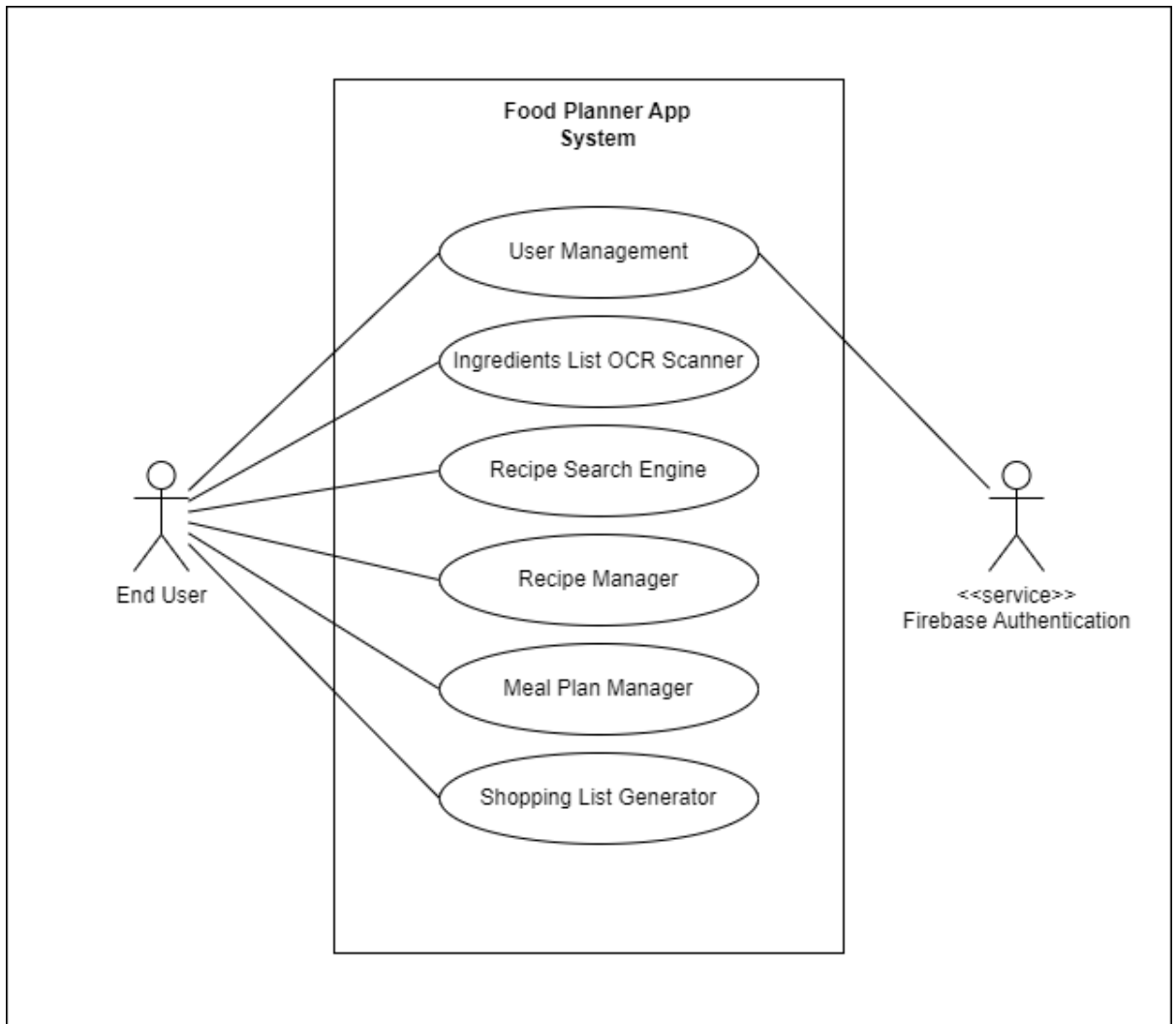
6. **FR6 - Shopping List Generator:**

The software system must accomplish generating an automated shopping list. This functional requirement will allow the End User to generate a shopping list containing all ingredients required for all recipes in their meal plan. The food shopping list information will be stored externally using a Google Firebase Realtime Database.

*Above are the system functional requirements listed in order of priority. The highest functional requirement for the Food Planner App is the user authentication system as all other functional requirements are dependent on the user being logged in to the application.*



### 2.1.1.1. Use Case Diagram



## 2.1.1.2. Requirement 1: User Management (FR1)

### 2.1.1.2.1. FR1 - Description & Priority

User Management is the highest priority requirement in the overall system. The End User must have a registered user account to log in to the application. All other application features require a user to be logged in to the application. In this Use Case the End User is the user using the application on their mobile device, and Firebase is the backend service used for the user authentication. This User Management use case covers user registration, user login and account deletion.

### 2.1.1.2.2. FR1 - Use Case

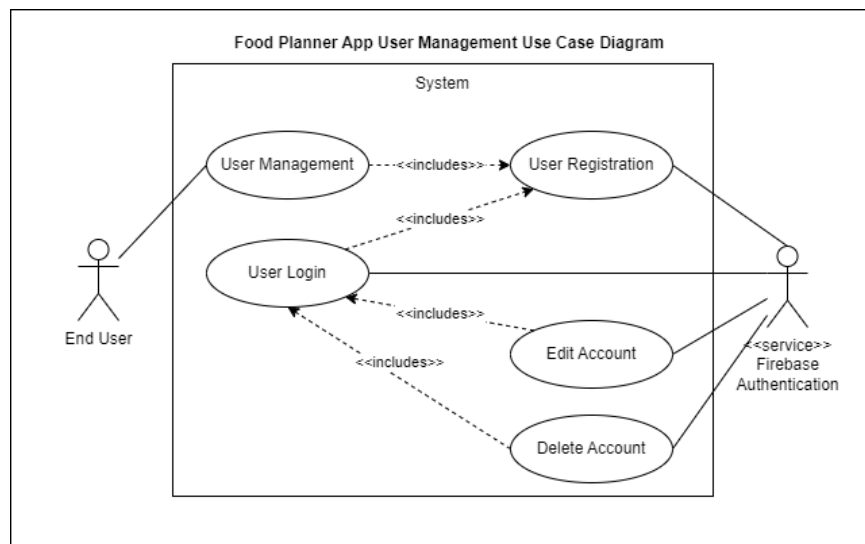
#### Scope

The scope of this use case is to allow the End User to register a new user account, login to their user account, edit their account settings and delete their user account in the Food Planner App system.

#### Description

This use case describes the user registration, user login, settings update, and account deletion processes. Using the application, the End User will be able to register a new user account, login to their existing user account, edit their account and or delete their account and associated data. A user requires a user account to use all main features of the application. The User Authentication service provider will be Firebase.

#### Use Case Diagram



#### Flow Description

The use case has three different basic flows:

- User Registration Task
- User Login Task
- User Edit Task
- User Deletion Task

## **Precondition**

- Food Planner App must be successfully installed on the End User's Android mobile phone device.
- The End User's Android mobile phone device mobile device must be connected to the internet.
- The Firebase Authentication backend service must be available.
- User Registration Task Flow – End User must have valid email address that is not associated with any existing user accounts on the system.
- User Login Task Flow – End User must have existing user account.
- User Deletion Task Flow – End User must have existing user account and be logged in to said account.

## **Activation**

### USER REGISTRATION TASK

This flow starts when a logged-out End User navigates to the User Login screen in the mobile application.

### USER LOGIN TASK

This flow starts when a logged-out End User navigates to the User Login screen in the mobile application.

### USER UPDATE TASK

This flow starts when a logged in End User navigates to the User Details screen in the mobile application.

### USER DELETION TASK

This flow starts when a logged in End User navigates to the User Details screen in the mobile application.

## **Main flow**

### USER REGISTRATION TASK

1. On the User Login screen, the user clicks the link to the User Registration screen.
2. The End User is redirected to the User Registration screen.
3. The End User enters text into the user registration form.
4. The End User clicks the Register button (See A1, A2, A3, E1, E2).
5. The End User new user account is successfully registered – a success message is displayed on screen.
6. The End User is successfully logged in to the application and redirected to the Home screen.

### USER LOGIN TASK

1. On the User Login screen, the End User enters text into the user login form.
2. The End User clicks the Login button (See E1, E2, E3, A4).
3. The End User is successfully logged in to the application - a success message is displayed on screen. The End User is redirected to the Home screen.

## USER UPDATE TASK

1. The logged in End User uses the application navigation menu to navigate to the User Settings screen.
2. On the User Settings screen, the End User adds a new password or email and clicks the Update Password or Update Email button.
3. A confirmation box appears with options to Confirm or Cancel the account update (See E4).
4. User clicks the Confirm button. (See E1, E2)
5. User account is successfully updated. User remains on the Edit Settings page.

## USER DELETION TASK

1. The logged in End User uses the application navigation menu to navigate to the User Settings screen.
2. On the User Settings screen, the End User clicks the 'Delete Account' button.
3. A confirmation box appears with options to Confirm or Cancel the account deletion (See E5).
4. User clicks the Confirm button. (See E1, E2)
5. User account is successfully deleted. User is logged out and redirected to the login page.

## Alternate flow

**A1** : Username is already associated with registered user.

1. An error notification is displayed on screen and the user registration is not successful.
2. The End User types a valid username into the username field that is not already associated with an existing user account.
3. The End User clicks the Register button again.
4. The use case continues at position 5 of the main flow.

**A2** : Password and password confirmation do not match.

1. An error notification is displayed on screen and the user registration is not successful.
2. The End User types a password and password confirmation into the form which match.
3. The End User clicks the Register button again.
4. The use case continues at position 5 of the main flow.

**A3** : Not all required registration form fields are completed.

1. An error notification is displayed on screen and the user registration is not successful.
2. The End User types a valid username, password and password confirmation into the form which match.
3. The End User clicks the Register button again.
4. The use case continues at position 5 of the main flow.

**A4** : Not all required login form fields are complete.

1. An error notification is displayed on screen and the user login is not successful.
2. The End User types a valid username and password into the form.
3. The End User clicks the Login button again.
4. The use case continues at position 3 of the main flow.

**A5** : The username is not valid.

1. An error notification is displayed on screen and the user login is not successful.

2. The End User types a valid username and password into the form.
3. The End User clicks the Login button again.
4. The use case continues at position 3 of the main flow.

**A6** : Incorrect password is entered.

1. An error notification is displayed on screen and the user login is not successful.
2. The End User types a valid username and correct password into the form.
3. The End User clicks the Login button again.
4. The use case continues at position 3 of the main flow.

### **Exceptional flow**

**E1** : The End User's device is disconnected from the internet.

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

**E2** : The Firebase Authentication backend service is unavailable.

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

**E3** : The username is not associated with an existing user account.

1. An error notification is displayed on screen and the user login is not successful.
2. User must complete User Registration Task.
3. The Use Case ends.

**E4** : The End User clicks the Cancel button.

1. The user account setting is not updated.
2. The Use Case ends.

**E5** : The End User clicks the Cancel button.

1. The user account is not deleted.
2. The Use Case ends.

### **Termination**

#### **USER REGISTRATION TASK**

The Use Case ends when the End User has successfully registered a new user account, are successfully logged in, and are redirected to the application Home screen.

#### **USER LOGIN TASK**

The Use Case ends when the End User has successfully logged in to their user account and are redirected to the application Home screen.

#### **USER UPDATE TASK**

The Use Case ends when the End User has successfully updated their account settings. The user remains on the Account Settings screen.

#### **USER DELETION TASK**

The Use Case ends when the End User has successfully deleted their user account and are redirected to the Login screen.

### **Post condition**

The system goes into a wait state.

### 2.1.1.3. Requirement 2: Ingredients List OCR Scanner (FR2)

#### 2.1.1.3.1. FR2 - Description & Priority

The Ingredient List OCR Scanner is the second highest functional requirement of the overall system. It is one of the core features of the application. This feature allows the End User to upload an image of a food Ingredients list to the application. Text from this image will then be extracted using Google's ML Kit Vision Text Recognition API. The End User will then receive a breakdown of the ingredients and their meaning.

#### 2.1.1.3.2. FR2 - Use Case

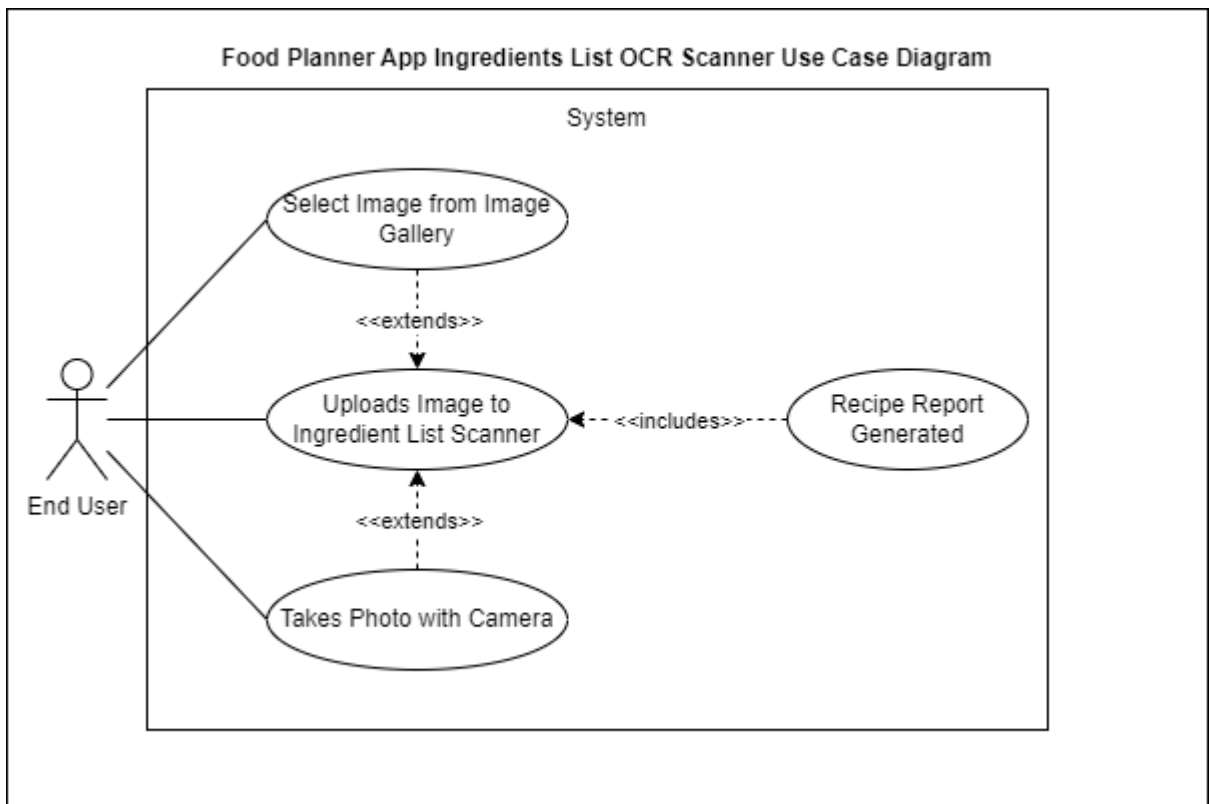
##### Scope

The scope of this use case is to allow the End User to use the Food Ingredients List OCR Scanner in the Food Planner App system.

##### Description

This use case describes the actions taken by the End User to successfully use the Food Planner App Food Ingredients List OCR Scanner. Using the application, the End User can upload an image of a food ingredients list and then receive a report containing a list of the ingredients and their meaning. Google's ML Kit Text Vision Recognition API is used to recognise and extract the ingredients text from the image.

##### Use Case Diagram



## Flow Description

### Precondition

- Application must be successfully installed on the End User's Android mobile phone device.
- The End User must have a registered user account and be logged in to the application.
- The End User's Android mobile phone device mobile device must be connected to the internet.
- ML Kit Text Recognition API must be compatible with End Users mobile phone device.

### Activation

This use case starts when an End User navigates to the Ingredients List Scanner screen.

### Main flow

1. The End User opens the application on their mobile device. (See A1)
2. The End User is located on the Home screen.
3. The End User clicks the Ingredients Scanner button.
4. The End User is redirected to the Ingredients List OCR Scanner screen.
5. The End User clicks the Select Image button (See A2, A3)
6. The End User clicks the Read Ingredients button (See E1, E2, E3).
7. The ingredients text is extracted from the image. The ingredients are looked up in the database. For each ingredient listed, if there is a description of the ingredient listed in the database, then it will be displayed to the user on the screen. (See E4)

### Alternate flow

**A1** : End User is logged out.

1. The End User is redirected to the User Login screen.
2. The End User enters a valid username and password into the user login form.
3. The End User clicks the Login button.
4. The End User is successfully logged in to the application.
5. The use case continues at position 2 of the main flow.

**A2** : End User selects Choose Existing Image from Library option.

1. End User's phone image gallery opens.
2. The End User select an existing image from their image gallery.
3. The use case continues at position 6 of the main flow.

**A3** : End User selects Take Photo Using Camera option.

1. End User's phone Camera app opens.
2. End User takes a photo using camera app.
3. The use case continues at position 6 of the main flow.

### Exceptional flow

**E1** : The End User's device is disconnected from the internet.

1. An error notification is displayed on screen and the text recognition is not executed.
2. The Use Case ends.

**E2** : Google ML Kit Text Recognition API is unavailable.

1. An error notification is displayed on screen and the text recognition is not executed.
2. The Use Case ends.

**E3** : No text is detected in selected image.

1. An error notification is displayed on screen and the user login is not successful. The End User is prompted to reupload a different image.
2. The End User uploads a valid image of a food ingredients list.
3. The End User clicks the Read Text button again.
4. The use case continues at position 6 of the main flow.

**E4** : Remote Firebase Cloud Database is unavailable.

1. An error notification is displayed on screen and the action is not executed.
2. The Use Case ends.

### **Termination**

The Use Case ends when the user has successfully uploaded an image of a food ingredients list and received a report of the ingredients and their meaning.

### **Post condition**

The system goes into a wait state.



#### 2.1.1.4. Requirement 3: Recipe Manager (FR3)

##### 2.1.1.4.1. FR3 - Description & Priority

The Recipe Manager is the third highest priority requirement in the overall system. The Recipe Manager feature allows the user to create, read, update, and delete custom recipes. In this use case the End User is the user using the application on their mobile device, Firebase is used as the database provider which will store the recipe records externally. The application Recipe Search feature is dependent on the recipe manager functionality.

##### 2.1.1.4.2. FR3 - Use Case

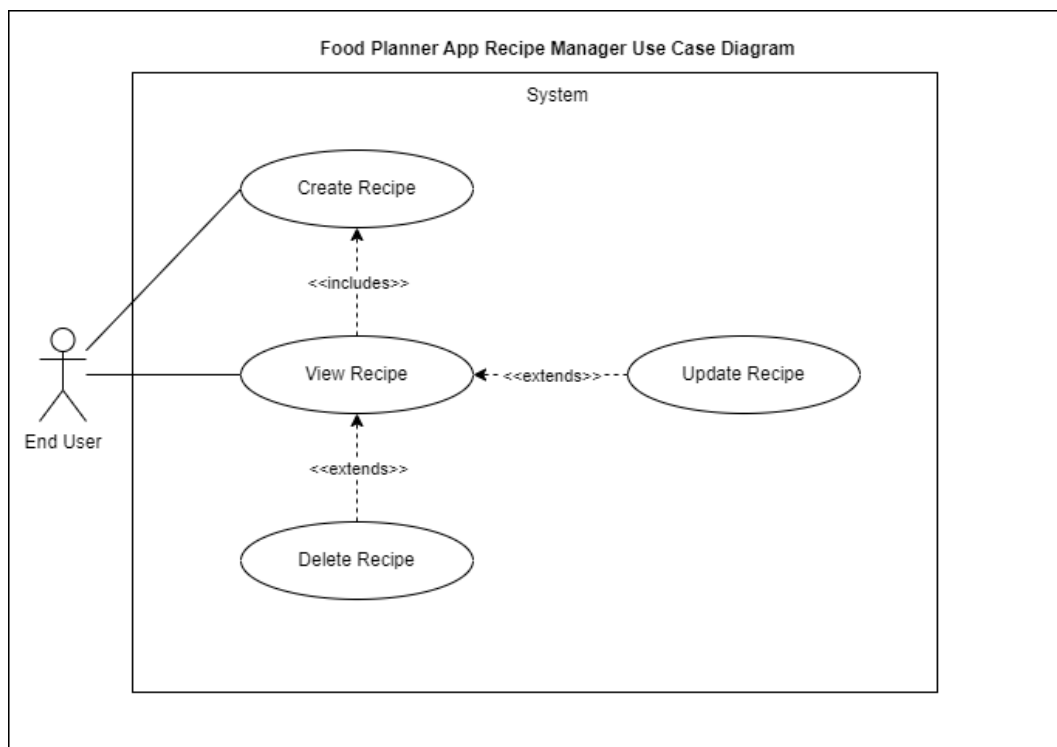
### Scope

The scope of this use case is to allow the End User to create, view, update and delete recipes using the Food Planner App system.

### Description

This use case describes the actions taken by the End User to create, view, update and delete custom recipes using the Recipe Manager functionality in the Food Planner App. The recipe records created will be stored externally in a Cloud Firebase remote database.

### Use Case Diagram



### Flow Description

The use case has three different basic flows:

- Create Recipe Task
- Update Recipe Task
- Delete Recipe Task

### **Precondition**

- The Food Planner Application must be successfully installed on the End User's Android mobile phone device.
- The End User must have a registered user account and be logged in to the application.
- The End User's Android mobile phone device must be connected to the internet.
- Remote Firebase Cloud Database service is available.
- Update Recipe Task – There must be at least 1 existing recipe which can be selected by the End User to be updated.
- Delete Recipe Task – There must be at least 1 existing recipe which can be selected by the End User to be deleted.

### **Activation**

This use case starts when a logged-in End User navigates to the Recipes List screen.

### **Main flow**

#### **CREATE RECIPE TASK**

1. On the Recipes List screen, the End User clicks the + icon which redirects the End User to the Recipe Creation page.
2. The end user fills out the form on the recipe creation page. (See A1, A2)
3. The End User clicks the Create Recipe button. (See E1, E2, A3)
4. A success message is displayed on screen and the recipe is successfully created. The user is redirected to the Recipe Details page of the recipe just created.

#### **UPDATE RECIPE TASK**

1. On the Recipes List screen, the End User clicks an existing recipe on screen and selects the Edit Recipe button.
2. The End User is redirected to the Update Recipe screen.
3. On the Update Recipe screen, the user fills out the new recipe information.
4. The End User clicks the Update Recipe button. (See A4, E1, E2)
5. The Recipe is successfully updated, the user is redirected to the Recipe List screen, the user clicks the recipe and clicks the View Details button. The user can view the updated details on the Recipe Details page.

#### **DELETE RECIPE TASK**

1. On the Recipes List screen, the End User clicks an existing recipe on screen and selects the Edit Details button. The End User is redirected to the recipe edit screen.
2. The End User clicks the Delete Recipe button.
3. A confirmation box appears with options to Confirm or Cancel the recipe deletion. (See E3)
4. User clicks the Confirm button. (See E1, E2)
5. Recipe is successfully deleted. User is redirected to the Recipe List screen and the recipe is no longer listed on this screen.

### **Alternate flow**

**A1** : User creates private recipe.

1. The end user sets the recipe as private.
2. Recipe will be created as private and only accessible to the owner.
3. The use case continues at position 3 of the main flow.

**A2** : User creates public recipe.

1. The end user sets the recipe as public.
2. Recipe will be created as public and can be accessed by anyone.
3. The use case continues at position 3 of the main flow.

**A3** : Not all required Create Recipe form fields are completed.

1. An error notification is displayed on screen and the recipe creation is not successful.
2. The End User completes all required fields.
3. The End User clicks the Create Recipe button again.
4. The use case continues at position 4 of the main flow.

**A4** : Not all required form Update Recipe form fields are completed.

1. An error notification is displayed on screen and the recipe update is not successful.
2. The End User completes all required fields.
3. The End User clicks the Update Recipe button again.
4. The use case continues at position 5 of the main flow.

### **Exceptional flow**

**E1** : The End User's device is disconnected from the internet.

1. An error notification is displayed on screen and the action is not executed.
2. The Use Case ends.

**E2** : Remote Firebase Cloud Database is unavailable.

1. An error notification is displayed on screen and the action is not executed.
2. The Use Case ends.

**E3** : The End User clicks the Cancel button.

1. The recipe is not deleted.
2. The Use Case ends.

### **Termination**

#### **CREATE RECIPE TASK**

The Use Case ends when the End User has successfully created a new recipe and is redirected to the Recipe List screen where the newly created recipe is listed.

#### **UPDATE RECIPE TASK**

The Use Case ends when the End User has successfully updated a recipe and is redirected to the Recipe List screen.

#### **DELETE RECIPE TASK**

The Use Case ends when the End User has successfully deleted a recipe and is redirected to the Recipe List screen and the deleted recipe is no longer listed.

### **Post condition**

The system goes into a wait state.

## 2.1.1.5. Requirement 4: Recipe Search (FR4)

### 2.1.1.5.1. FR4 - Description & Priority

The Recipe Search functionality is the fourth highest priority requirement in the overall system. The End User must be able to search for public Recipes using different search parameters. In this use case, the End User is the user using the application on their mobile device. The recipes will either be extracted from Google SERP using web scraping techniques or from the application primary Firebase Realtime Database. The decision between web scraping or extracting user recipes from a Firebase Realtime Database will be decided once the development begins. This use case outlines the Firebase Realtime Database solution where the application Recipe Search feature is dependent on the recipe manager functionality. The main objective of the Recipe Search feature is to allow the End User to search for recipes based on ingredients they already own to cut down on their food waste.

### 2.1.1.5.2. FR4 - Use Case

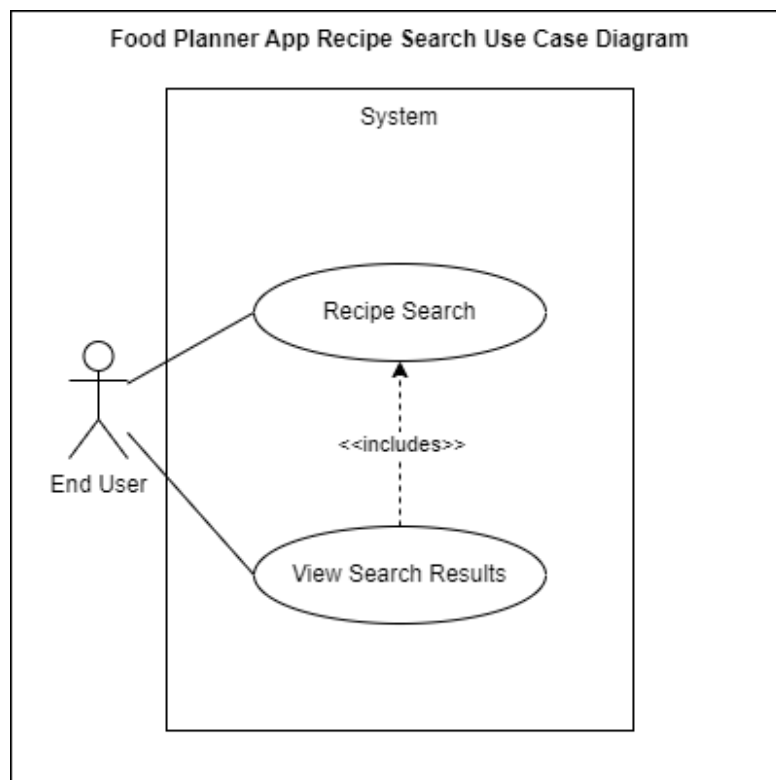
#### Scope

The scope of this use case is to allow the End User to search for existing public recipes by using different search parameters in the Food Planner App system.

#### Description

This use case describes the actions an End User must make to search for public recipes using the application recipe search functionality.

#### Use Case Diagram



## **Flow Description**

### **Precondition**

- Food Planner Application must be successfully installed on the End User's Android mobile phone device.
- The End User must have a registered user account and be logged in to the application.
- The End User's Android mobile phone device mobile device must be connected to the internet.
- Remote Firebase Cloud Database service is available.
- There must be existing public recipes that can be returned in the search results to the End User.

### **Activation**

This use case activates when the End User navigates to the Recipe Search screen.

### **Main flow**

1. One the Recipe Search screen, the End User enters search parameters.
2. The End User clicks the Search button. (See A1, E1, E2, E3)
3. Recipes matching the search queries entered are listed in the Search Results list.

### **Alternate flow**

**A1** : No Recipes match the search queries.

1. No recipes are returned in the Search Results list.
2. User refines or changes search parameters.
3. The use case continues at position 2 of the flow.

### **Exceptional flow**

**E1** : The End User's device is disconnected from the internet.

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

**E2** : Remote Firebase Cloud Database is unavailable.

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

**E3** : There are no public recipes on the system.

1. No search results are returned.
2. The Use Case ends.

### **Termination**

The Use Case ends when the End User receives a correct search result for their recipe search.

### **Post condition**

The system goes into a wait state.

## 2.1.1.6. Requirement 5: Meal Plan Manager (FR5)

### 2.1.1.6.1. FR5 - Description & Priority

The Meal Plan Manager is the fifth highest priority requirement in the overall system. Using the Meal Plan manager, a user will be able to add or remove recipes to and from their meal plan. In this use case the End User is the user using the application on their mobile device and Firebase is used as the database provider which will store the user meal plan records externally. The Meal Plan Manager feature is dependent on the recipe manager functionality.

### 2.1.1.6.2. FR5 - Use Case

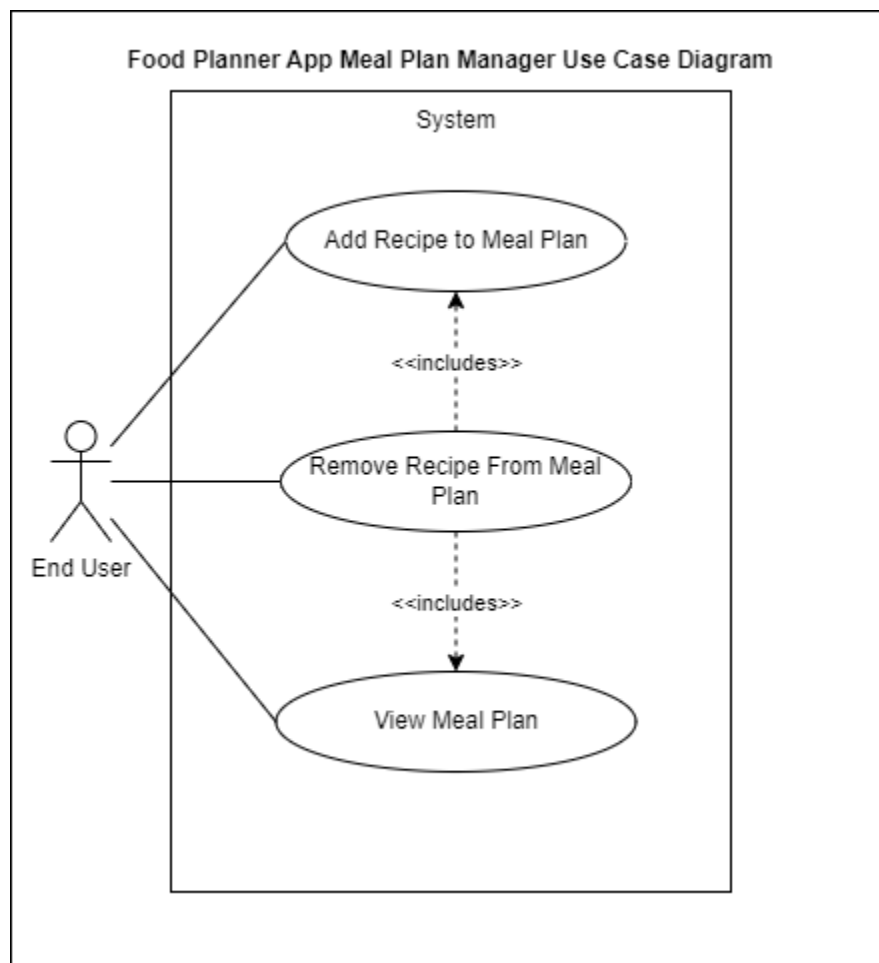
#### Scope

The scope of this use case is to allow the user to add and remove recipes to and from their meal plan on the Food Planner App system.

#### Description

This use case describes the steps a user must make to add or remove recipes from their weekly meal plan.

#### Use Case Diagram



## Flow Description

### Precondition

- Application must be successfully installed on the End User's Android mobile phone device.
- The End User must have a registered user account and be logged in to the application.
- The End User's Android mobile phone device mobile device must be connected to the internet.
- Remote Firebase Cloud Database service is available.
- There must be existing recipes that can be added to the End Users meal plan.

### Activation

This use case starts when an End User navigates to a Recipe Details screen.

### Main flow

1. On a Recipe Detail screen, the user clicks the Add to Meal Plan button.
2. The user will be redirected to the Weekly Meal Plan screen.
3. The user selects the day in which they would like to add the recipe to.
4. Once the day is selected, a confirmation prompt appears requesting for the user to Confirm or Cancel adding the recipe to the Meal Plan. (See E1)
5. The End User clicks confirm button. (See E2, E3)
6. The Recipe is successfully added to the Meal Plan
7. The End User is redirected to the Meal Plan screen.
8. The Selects the day that they added the Recipe to.
9. The End User highlights the meal just added and then Clicks the Remove from Meal plan button.
10. The recipe is successfully removed from the meal plan.

### Alternate flow

N/A

### Exceptional flow

**E1** : The End User clicks the Cancel button.

1. The recipe is not added.
2. The Use Case ends.

**E2** : The End User's device is disconnected from the internet.

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

**E3** : Remote Firebase Cloud Database is unavailable.

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

### Termination

This use cases ends when the End User successfully adds or removes a recipe to and from their weekly meal plan.

### Post condition

The system goes into a wait state.

### 2.1.1.7. Requirement 6: Food Shopping List Generator (FR6)

#### 2.1.1.7.1. FR6 - Description & Priority

The Shopping List generator is the lowest priority requirement in the overall system. In this Use Case the End User is the user generates a shopping list from their weekly meal plan. The purpose of this feature is that the user can create an automated list of items they need to buy to prepare the recipes in their meal plan.

#### 2.1.1.7.2. FR6 - Use Case

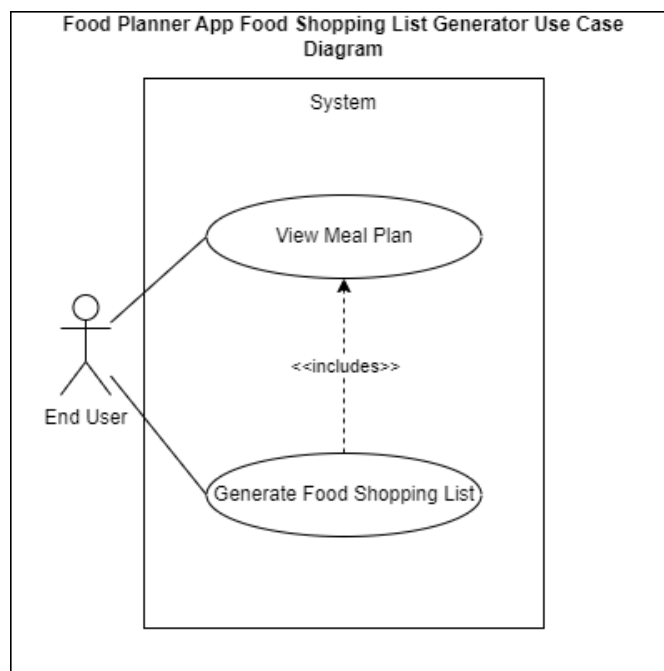
#### Scope

The scope of this use case is to allow the End User to generate a shopping list for the recipes on their weekly meal plan using the Food Planner App system.

#### Description

This use case describes the steps the End User must take to generate an automated shopping list from their weekly meal plan using the Food Planner Application.

#### Use Case Diagram



#### Flow Description

#### Precondition

- Food Planner Application must be successfully installed on the End User's Android mobile phone device.
- The End User must have a registered user account and be logged in to the application.
- The End User's Android mobile phone device mobile device must be connected to the internet.
- Remote Firebase Cloud Database service is available.
- There must be existing recipes added to the End Users meal plan.



### **Activation**

This use case is activated when the End User navigates to their Weekly Meal Plan screen.

### **Main flow**

1. On the Weekly Meal Plan screen, the user clicks the Create Shopping List button. (See E1)
2. The user is redirected to the Shopping List screen where all ingredients that are required to prepare the recipes on their weekly meal plan are listed. (See E2, E3)

### **Alternate flow**

N/A

### **Exceptional flow**

**E1** : No recipes added to Weekly Meal Plan

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

**E2** : The End User's device is disconnected from the internet.

3. An error notification is displayed on screen and the action is not successful.
4. The Use Case ends.

**E3** : Remote Firebase Cloud Database is unavailable.

1. An error notification is displayed on screen and the action is not successful.
2. The Use Case ends.

### **Termination**

The use case ends when the End User is successfully redirected to a screen containing the automatically generated Food Shopping list.

### **Post condition**

The system goes into a wait state.

### 2.1.2 Data Requirements

Registered user accounts will be created, stored and authenticated using Google's Firebase User Authentication backend service. This service must be available for a user to register for a new user account, to log in to an existing account, to update their user account details or to delete their account. Data relating to recipes, meal plans, and shopping lists will be stored externally using Google's Firebase Realtime Database. Database data should be able to be created, updated, and deleted without any errors. All errors from the manipulation of data in the application should be handled in the application code without effecting the user's experience. Security best practices should be implemented to protect users' data. The onus is on the user to control which data they submit to the application. Some prerequisites for registered users to access their user data are that their mobile device is connected to the internet, that they have the required version of Google Play Services installed on their device, that the Firebase User Authentication service is available, that they are logged in to the application, and that the Firebase Realtime Database service is available.

### 2.1.3 User Requirements

All users will be required to register for a user account and log in to access all core features of the application. By registering for a user account users accept the terms of service which require them to use the application responsibly. Using the application, users will be able to register for an account, log in to an account, log out of their account, delete their account, create a public or private custom recipe, update their recipes, delete their recipes, search for recipes, add recipes to their meal plan, remove recipes from their meal plan, and generate a food shopping list.

### 2.1.4 Environmental Requirements

The Food Planner application must be installed on an Android device which is running a Google supported Android OS version. The most up to date version of Google Play Services should be installed on the device. The application must be developed for an API level of 19 or higher as this is a requirement for all Google ML Kit APIs. The end user's mobile device will need to be connected to the internet to login or interact with the external Firebase Realtime Database. User authentication, external data storage, and external APIs are going to be used for integral parts of the application. Therefore, a lot of thought has been put into the chosen service providers. Google's products and services are renowned for their reliability, security, and availability. Google's products are well documented which will also support the development of this application.

### 2.1.5 Usability Requirements

During the development of this application, all efforts will be made to follow accessibility best practices. There will be a focus on the learnability, efficiency, memorability, and error handling of the application. The application UI and processes should be easy to use and intuitive. The application should be intuitively designed so that user training is not required in how to use it. User Experience (UX) will be a priority in the development of this application. There should be no accessibility warnings in the application code. The application should open on the user's device without crashing. Errors should be handled appropriately in the application code to prevent the application from crashing. There will be an appropriate use of easy-to-understand warnings, prompts, error messages, and confirmation messages throughout the application.

### System Architecture Diagram

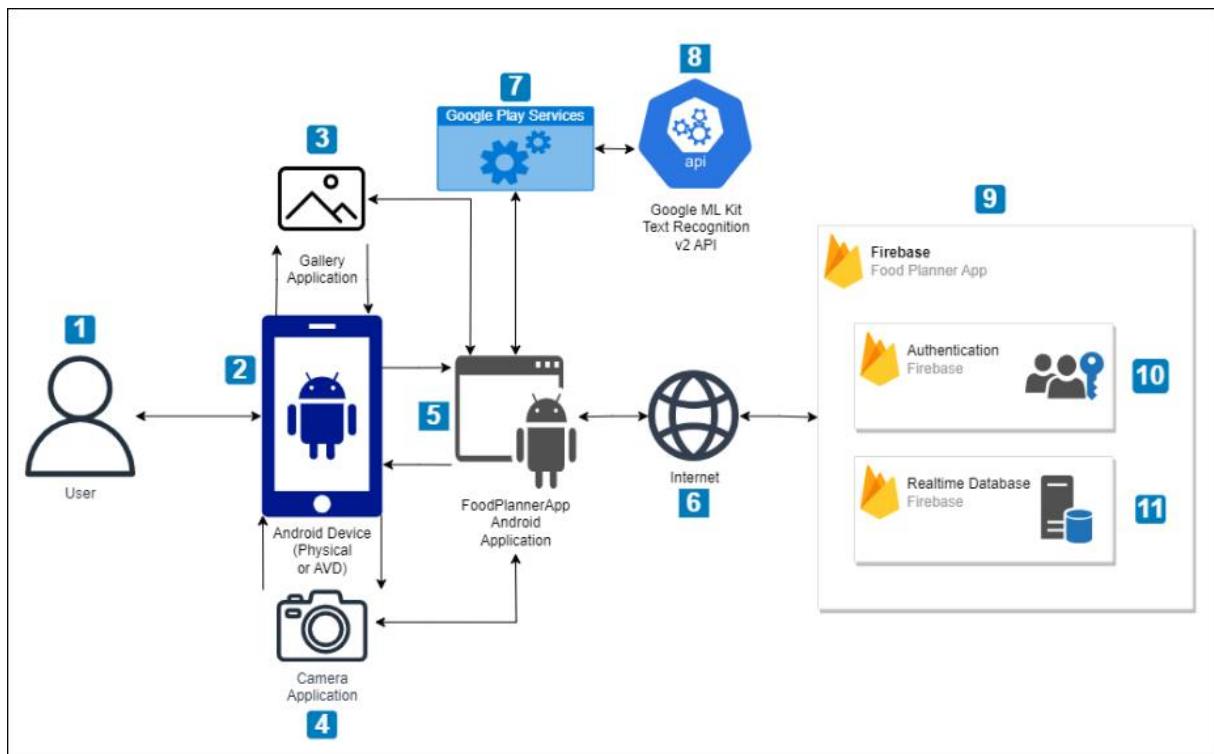


Figure 1 - FoodPlannerApp System Architecture Diagram (Development System)

1. **User** – The user running the FoodPlannerApp application on an Android device.
2. **Android Device** – This component represents the Android device (Physical) running the FoodPlannerApp application in Developer Mode or virtually using an AVD emulator via Android Studio IDE. The device must have an Android SDK version of 24 or higher installed to run the FoodPlannerApp application.
3. **Gallery Application** – This component represents the device Gallery application which the Ingredient Scanner feature is dependent on. The user must grant the FoodPlannerApp permission to access their gallery to use the full Ingredients Scanner feature functionality.
4. **Camera Application** – This component represents the device Camera application which the Ingredient Scanner feature is dependent on. The user must grant the FoodPlannerApp permission to access their Camera to use the full Ingredients Scanner feature functionality.
5. **FoodPlannerApp** – This component represents the FoodPlannerApp installed on the user’s device via Android Studio. If the device was released to production the FoodPlannerApp application would be installed via the Google Play Store. The FoodPlannerApp applications target Android SDK version is 33.
6. **Internet** – The device must be connected to the internet to connect to the Firebase web services. The Google ML Kit Text Recognition functionality is available offline as the ML Model is set to be downloaded to the device when the application is installed.

7. **Google Play Services** – Google Play Services is required to use the ML Kit Text Recognition model.
8. **Google ML Kit Text Recognition v2 API** – This ML Kit API requires an Android API level of 19 or greater. The Text Recognition model is dependent on Google Play Services however the ML model is automatically downloaded to the device once the application is installed.
9. **Firebase** – Firebase is used as the cloud app development platform.
10. **Firebase Authentication** – The application uses Firebase Authentication as the application user authentication service. The User Management feature is dependent on a successful connection to the Firebase Authentication service.
11. **Firebase Realtime Database** – Firebase Realtime Database is used as the application’s cloud-hosted NoSQL cloud database where the applications data is stored as JSON. The applications Recipe, Meals and Ingredient data are stored in a Firebase Project Realtime Database. The Recipe Manager, Recipe Search, Ingredient Scanner and Shopping List features are dependent on a successful connection to this database.

### MVC (Model-View-Controller) Architecture Pattern Diagram

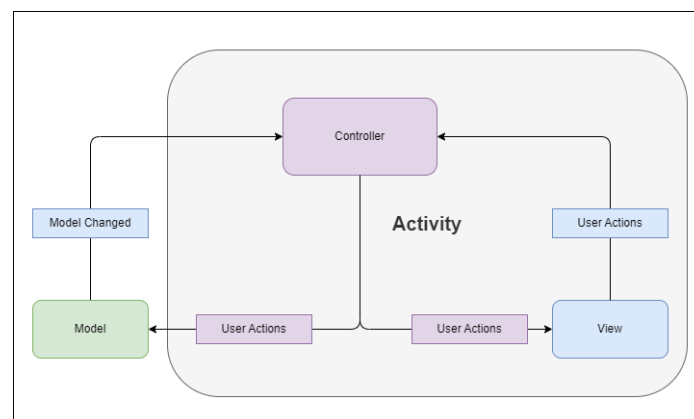


Figure 2 – MVC-style Architecture Pattern used in the FoodPlannerApp application. (Adilovic, 2021)

The software architecture design pattern that the FoodPlannerApp application implementation most resembles is MVC (Model-View-Controller). In the MVC design pattern there is a separation between the Model – used for structuring data, the Controller – used for handling user input, and the View – used to present data to the user. (Adilovic, 2021) For the most part, in the implementation of the FoodPlannerApp, the Activity class contains both the View (UI Presentation Logic) and the Controller (User Input Handling Logic). Therefore, this is not a strict implementation of the MVC pattern as there is no total separation of responsibility between the Controller and View. However, in some cases, RecyclerView adapter classes are used to control how sets of data are displayed in an Activity’s RecyclerView. The application also implements utility and interface classes that can be reused by multiple Activities.

## Simplified FoodPlannerApp Application UML Package/Class Diagram

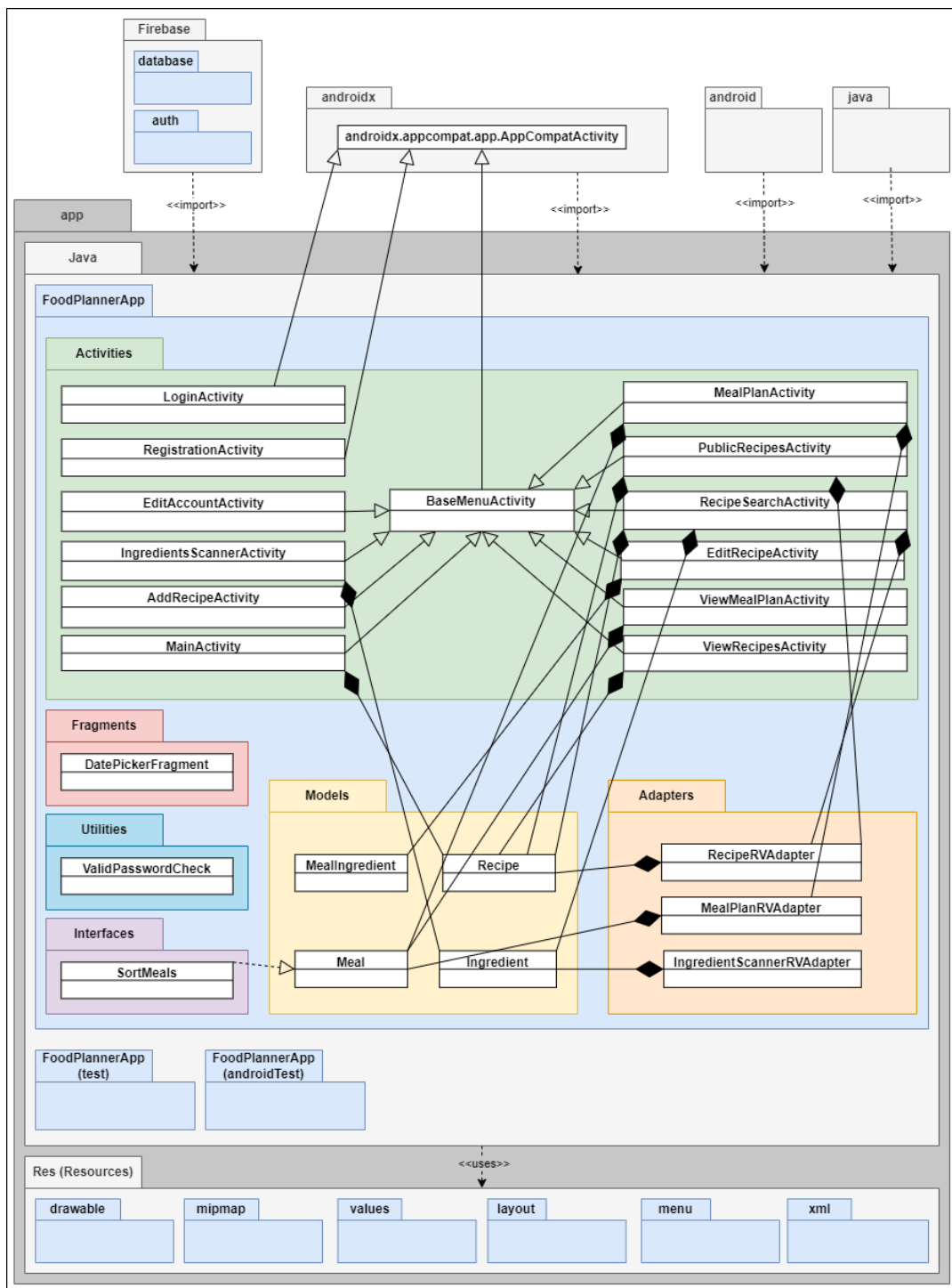


Figure 3 – FoodPlannerApp application simplified (class methods & variables excluded) UML Package/Class Diagram.

### Java > Fragments Package

The Fragment Package contains the DatePickerFragment class used to create a Calendar dialog to allow the user to select a date when adding a recipe to their meal plan.

### Java > Interfaces Package

The Interfaces Package contains the SortMeals comparator class used for sorting meals by date in ascending order in the Meal Planner. The Sort algorithm is used in this class.

## Java > Activities Package

The Activities Package contains all the projects Activity classes. Each activity class corresponds to one application UI screen and related functionality. Activity classes follow the below lifecycle flow – each methods in a rectangle in the below diagram represents a ‘Callback’ method to be executed when the Activity enters a certain state (Google for Developers, 2023):

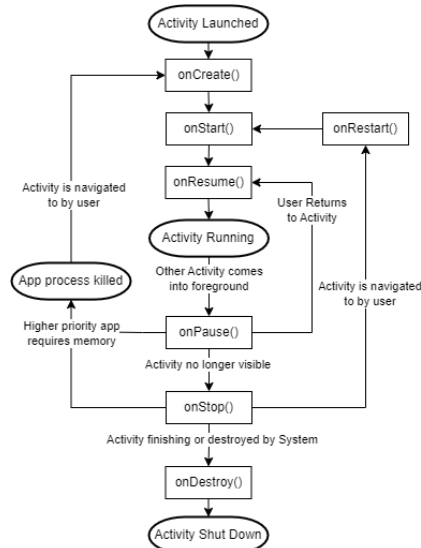


Figure 4 - Android Activity Class Lifecycle (Google for Developers, 2023)

## Java > Utilities Package

The Utilities Package contains the ValidPasswordCheck utility class used for verifying that a password meets the minimum password requirements using Regex and Pattern Matching.

## Java > Models Package

The Models Package contains all the classes for modelling the application data such as Meals, Meal Ingredients, Ingredients and Recipes.

## Java > Adapters Package

The Adapters Package contains the Adapter Classes used to control how sets of object ArrayList data are displayed in an Activity RecyclerView.

## Resource Folder

The application Res (Resource) folder contains a Drawable folder containing application vector and PNG images, the Layout folder containing the UI XML template files used by the Activity classes, the Menu folder containing the application UI Toolbar Menu XML template, the Mipmap folder containing the application logos and the Values folder containing reusable string/colour values and the application theme template.

## Firebase Packages (Imported)

The Activity classes import from Firebase ‘Auth’ and ‘Database’ packages to interact with Firebase Authentication and Firebase Realtime Database.

## Java / Android / Androidx Packages (Imported)

The Activity classes import various packages from the Java, Android and Androidx packages such as the ‘android.appcompat.app.AppCompatActivity’ which is used as an activity base class to support running new platform features on older devices.

## 2.3 Implementation

### BaseMenuActivity.java (Activity class in Activities package):

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/BaseMenuActivity.java*
- **Primary Resource Layout:** settings\_main.xml
- **Purpose:** The BaseMenuActivity class contains the logic for the Toolbar Settings Menu. The BaseMenuActivity class extends the AppCompatActivity Class, and all classes (apart from the Login and Registration Activities classes) extend the BaseMenuActivity class so that the settings menu can be accessed in those activities.

The onCreateOptionsMenu() method gets the current Firebase Authentication instance (required for the Log Out option) and inflates the setting menu UI:

```
// menu onCreateOptionsMenu
Ruby Lennon
public boolean onCreateOptionsMenu(Menu menu) {
    // assign current Firebase Authentication instance
    firebaseAuth = FirebaseAuth.getInstance();
    // inflate menu using settings_menu layout
    getMenuInflater().inflate(R.menu.settings_main, menu);
    return true;
}
```

The onOptionsItemSelected() method sets the functionality for each of the menu items when clicked though using a Switch statement:

```
public boolean onOptionsItemSelected(@NonNull MenuItem item){
    // get the selected menu item ID
    int id = item.getItemId();
    switch (id) { // case switch for defining menu item click actions
        case R.id.idAddRecipe: // Add Recipe option clicked
            Intent i1 = new Intent( packageContext: BaseMenuActivity.this, AddRecipeActivity.class);
            startActivity(i1); // start Add Recipe activity
            return true;
        case R.id.idMyRecipes: // My Recipes option clicked
            Intent i2 = new Intent( packageContext: BaseMenuActivity.this, MainActivity.class);
            startActivity(i2); // start Main (My Recipes) activity
            return true;
        case R.id.idPublicRecipes: // Public Recipes option clicked
            Intent i3 = new Intent( packageContext: BaseMenuActivity.this, PublicRecipesActivity.class);
            startActivity(i3); // start Public Recipe activity
            return true;
        case R.id.idScan: // Ingredients Scanner option clicked
            Intent i4 = new Intent( packageContext: BaseMenuActivity.this, IngredientsScannerActivity.class);
            startActivity(i4); // start Ingredients Scanner activity
            return true;
        case R.id.idSearch: // Recipe Search option clicked
            Intent i5 = new Intent( packageContext: BaseMenuActivity.this, RecipeSearchActivity.class);
            startActivity(i5); // start Recipe Search activity
            return true;
        case R.id.idMealPlan: // Meal Planner option clicked
            Intent i6 = new Intent( packageContext: BaseMenuActivity.this, MealPlanActivity.class);
            startActivity(i6); // start Meal Planner activity
            return true;
        case R.id.idEditAccount: // Edit Account option clicked
            Intent i7 = new Intent( packageContext: BaseMenuActivity.this, EditAccountActivity.class);
            startActivity(i7);
            return true;
        case R.id.idLogout: // Log Out option clicked
            Toast.makeText( context: this, text: "User Logged Out", Toast.LENGTH_SHORT).show();
            firebaseAuth.signOut(); // sign out currently logged in user
            Intent i8 = new Intent( packageContext: BaseMenuActivity.this, LoginActivity.class);
            startActivity(i8); // start Login activity
            this.finish(); // finish current activity
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

### **Ingredient.java (Model class in Models package):**

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/models/Ingredient.java*
- **Purpose:** Model used for receiving, storing, displaying, and processing ingredient objects retrieved from the Firebase Realtime Database.
- **Ingredient Attributes:** Name, Description.

### **Meal.java (Model class in Models package):**

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/models/Meal.java*
- **Purpose:** Model used for creating, receiving, storing, displaying, and processing meal objects sent to and from the Firebase Realtime Database.
- **Meal Attributes:** Name, cooking time, preparation time, servings, suitability, cuisine, image URL, source URL, description, method, ingredients, ID, user ID, recipe ID, date.

### **MealIngredients.java (Model class in Models package):**

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/models/MealIngredient.java*
- **Purpose:** Model used for creating, receiving, storing, displaying, and processing meal ingredient objects sent to and from the Firebase Realtime Database.
- **Meal Ingredient Attributes:** database key, ingredient, purchased status.

### **Recipe.java (Model class in Models package):**

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/models/Recipe.java*
- **Purpose:** Model used for creating, receiving, storing, displaying, and processing recipe objects sent to and from the Firebase Realtime Database.
- **Recipe Attributes:** Name, cooking time, preparation time, servings, suitability, cuisine, image URL, source URL, description, method, ingredients, user ID, recipe ID, visibility.

### **IngredientScannerRVAdapter.java (Recycler View Adapter class in Adapters package):**

- **Project File Location:**
  - *java/com/example/foodplannerapp/adapters/IngredientScannerRVAdapter.java*
- **Purpose:** Adapter class used to display Ingredient ArrayList data in a Recycler View. This class Inflates the Recycler View with one card layout item for every object in the ArrayList passed to the Adapter class. This class binds the ArrayList object data to the card layout element to display its data.
- **Activities Used By:** Ingredients Scanner Activity



### MealPlanRVAdapter.java (Recycler View Adapter class in Adapters package):

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/models/MealPlanRVAdapter.java*
- **Purpose:** Adapter class used to display Meal ArrayList data in a Recycler View. This class Inflates the Recycler View with one card layout item for every object in the ArrayList passed to the Adapter class. This class binds the ArrayList object data to the card layout element to display its data. This Adapter class also sets an on click interface for the recycler view items.
- **Activities Used By:** Meal Planner Activity

### RecipeRVAdapter.java (Recycler View Adapter class in Adapters package):

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/models/RecipeRVAdapter.java*
- **Purpose:** Adapter class used to display Recipe ArrayList data in a Recycler View. This class Inflates the Recycler View with one card layout item for every object in the ArrayList passed to the Adapter class. This class binds the ArrayList object data to the card layout element to display its data. This Adapter class also sets an on click interface for the recycler view items.
- **Activities used by:** Main Activity (My Recipes), Public Activity, Recipe Search Activity.

## 2.3.1 User Management Feature (Login, Registration, Account Settings)

### ValidPasswordCheck.java (Utility class in Utilities package):

- **Project File Location:**
  - *java/com/example/foodplannerapp/utilities/ValidPasswordCheck.java*
- **Purpose:** Utility class for checking if password meets minimum requirements
- **Activities used by:** Registration Activity, Edit Account Activity
- **Key imports:**
  - *java.util.regex.Matcher, Pattern*

```
// @Reference - https://www.geeksforgeeks.org/how-to-validate-a-password-using-regular-expressions-in-java/
// method for checking if user supplied password is valid
10 usages  ▸ Ruby Lennon *
public final class ValidPasswordCheck{
    ▸ Ruby Lennon
    private ValidPasswordCheck(){
    8 usages  ▸ Ruby Lennon *
    public static boolean isPasswordValid(String password) {
        // create regex pattern
        String regex = "^(?=.*\\d)"
            + "(?=.*[a-z])(?=.*[A-Z])"
            + "(?=.*[@#$%&+])"
            + "(?=\\S+$).{8,20}$";
        // regex compilation
        Pattern pattern = Pattern.compile(regex);
        // return false if password is empty
        if (password == null) {
            return true;
        }
        // use matcher method to check if password matches
        Matcher matcher = pattern.matcher(password);
        // Return true if password matches regex
        return matcher.matches();
    }
}
```

## LoginActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/LoginActivity.java*
- **Primary Resource Layout:** *activity\_login.xml*
- **Purpose:** The Login Activity is used to enable registered users to log in to the application. It also provides a link to the Registration Activity screen. If a user is already logged in, they will be directed to the Main Activity screen when the Login Activity is launched.
- **Key package imports:**
  - `com.google.firebase.auth.FirebaseAuth`, `FirebaseUser`

When the Login Activity is launched, an instance of the FirebaseAuth is obtained:

```
// retrieve and store the current firebase authentication instance
firebaseAuth = FirebaseAuth.getInstance();
```

The `firebase.auth.FirebaseAuth` package method `signInWithEmailAndPassword` is then called using the instance above and the user provided email and password is passed to it. If the login is successful using this method the user will be directed to the Main Activity (My Recipes) screen. If the login is unsuccessful then an error message will be displayed on screen:

```
// @Reference - https://www.geeksforgeeks.org/user-authentication-and-crud-operation-with-firebase-realtime-database-in-android/
// Reference description - tutorial on how to create a Login Activity for Firebase Authenticate
// else if the password and username value is provided then execute the following
// sign in the user to their account using the provided username and password
firebaseAuth.signInWithEmailAndPassword(userName, pwd).addOnCompleteListener(task -> {
    if(task.isSuccessful){// if the task is successful
        loadingPB.setVisibility(View.GONE);// hide the progress loading bar
        Toast.makeText(context LoginActivity.this, text "Login Successful",
            Toast.LENGTH_SHORT).show();// show toast
        // redirect the user to the main activity (My Recipes)
        Intent i = new Intent(context LoginActivity.this, MainActivity.class);
        startActivity(i);// start Main activity
        finish();// finish this activity
    } else {// if the task is not successful
        // if the login fails then execute the following
        loadingPB.setVisibility(View.GONE);// hide the progress loading bar
        Toast.makeText(context LoginActivity.this, text "Login Failed. Please try again..",
            Toast.LENGTH_SHORT).show();// show toast
    }
});
```

## RegistrationActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/RegistrationActivity.java*
- **Primary Resource Layout:** *activity\_registration.xml*
- **Purpose:** The Registration Activity is used to enable new users to register for an account. It also provides a link to the Login Activity screen.
- **Key package imports:**
  - `com.google.firebase.auth.FirebaseAuth`
  - `com.example.foodplannerapp.utilities.ValidPasswordCheck`

When the Registration Activity is launched, an instance of the FirebaseAuth is obtained. If the user provides all required data, and the user's provided password is valid (passes ValidPasswordCheck utility class check), then the firebase.auth.FirebaseAuth package method createUserWithEmailAndPassword is called using the FirebaseAuth instance and the user provided email and password is passed to it. If the registration is successful, then the user account will be created in Firebase Authentication and the user will be signed in and directed to the Main Activity (My Recipes) screen. If the registration fails an error message will be displayed on screen:

```

else{// if all required information is valid and provided execute the following
// @Reference - https://www.geeksforgeeks.org/user-authentication-and-crud-operation-with-firebase-realtime-database-in-android/
// Reference description - tutorial on how to create a Register Activity for Firebase Authenticate
// create a Firebase Authentication user
firebaseAuth.createUserWithEmailAndPassword(userName, pwd).addOnCompleteListener(task -> {
    if(task.isSuccessful()){ // if the task is successful then execute the following
        loadingPB.setVisibility(View.GONE);// hide the progress bar
        Toast.makeText( context: RegistrationActivity.this, text: "Registration Successful", Toast.LENGTH_SHORT).show();
        Intent i = new Intent( packageContext: RegistrationActivity.this, MainActivity.class);
        startActivity(i);// start Main activity
        finish();
    } else { // if the task fails execute the following
        loadingPB.setVisibility(View.GONE);// hide the loading progress bar
        // display error toast with failure reason
        if(Objects.requireNonNull(task.getException()).toString().contains("The email address is badly formatted.")){
            Toast.makeText( context: RegistrationActivity.this, text: "Registration failed. Please add valid email.",
                Toast.LENGTH_SHORT).show();
        } else if(Objects.requireNonNull(task.getException()).toString().contains("The given password is invalid. [ Password should be at least 6 characters ]")){
            Toast.makeText( context: RegistrationActivity.this, text: "Registration failed. Password should be at least 6 characters.",
                Toast.LENGTH_SHORT).show();
        } else if(Objects.requireNonNull(task.getException()).toString().contains("The email address is already in use by another account")){
            Toast.makeText( context: RegistrationActivity.this, text: "Registration failed. The email address is already in use by another account.",
                Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText( context: RegistrationActivity.this, text: "Registration failed. Please try again...", Toast.LENGTH_SHORT).show();
        }
    }
});

```

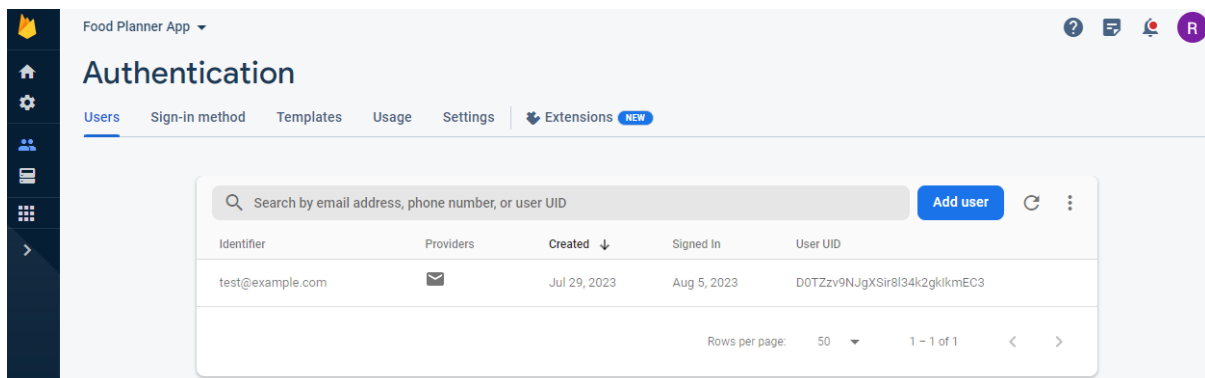


Figure 5 - Registered user listed in Firebase Authentication Users list.

### EditAccountActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - o `app/src/main/java/com/example/foodplannerapp/activities/EditAccountActivity.java`
- **Primary Resource Layout:** `activity_edit_account.xml`
- **Purpose:** The Edit Account Activity is used to enable existing logged in users to update their email or password, or to delete their account.
- **Key package imports:**
  - o `com.google.firebase.auth.FirebaseAuth, FirebaseAuth`
  - o `com.example.foodplannerapp.utilities.ValidPasswordCheck`

When the Edit Account Activity is launched, the currently logged in user (FirebaseUser) is obtained from the current Firebase Auth instance:

```
//get current user
final FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
```

This FirebaseUser reference is then used to call the updateEmail, updatePassword and delete FirebaseUser methods. Each of these three actions are time sensitive and require recent user authentication to be executed. If recent authentication is required, then an error message will be displayed on screen requesting for the user to sign in again before executing the action.

```
// @Reference - https://www.youtube.com/watch?v=FptELNWvngQ
// Ref Description - Tutorial on how to update Firebase Authentication users password/email
// and on how to delete users
// update email of signed in user using new email value
user.updateEmail(idEdtEmail.getText().toString().trim())
    .addOnCompleteListener(task -> {
        if(task.isSuccessful()){// if the task is successful
            Toast.makeText( context: EditAccountActivity.this, text: "Email updated", Toast.LENGTH_SHORT).show();
            idCurrentEmail.setText(newEmail);// update the current email textview value
            idPBLoading.setVisibility(View.GONE);// hide the progress bar
        }
    });
```

Figure 6 - updateEmail method

```
// @Reference - https://www.youtube.com/watch?v=FptELNWvngQ
// Ref Description - Tutorial on how to update Firebase Authentication users password/email
// and on how to delete users
//update the users password
user.updatePassword(pwd).addOnCompleteListener(task -> {
    if(task.isSuccessful()){// if the task is successful
        Toast.makeText( context: EditAccountActivity.this, text: "Password updated", Toast.LENGTH_SHORT).show();
        idPBLoading.setVisibility(View.GONE);// hide the progress bar
        pwdHelpText.setVisibility(View.GONE);// hide the password help text
    }else{// display different error message based on error received from Firebase Authenticate
    }
});
```

Figure 7 - updatePassword method

```
// @Reference - https://www.youtube.com/watch?v=FptELNWvngQ
// Ref Description - Tutorial on how to delete Firebase Authentication users
// and on how to delete users
user.delete().addOnCompleteListener(task -> {
    if(task.isSuccessful()){// if task is successful
        // display the following toast notification
        Toast.makeText( context: EditAccountActivity.this, text: "Account Successfully Deleted", Toast.LENGTH_SHORT).show();
        idPBLoading.setVisibility(View.GONE);// hide loading bar
        Intent i = new Intent( packageContext: EditAccountActivity.this, LoginActivity.class);
        startActivity(i);//navigate to login page
        finish();// finish the activity
    }
    // if the action is unsuccessful then execute the following
});
```

Figure 8 - delete method.

## 2.3.2 Ingredients List OCR Scanner Feature

### IngredientsScannerActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/IngredientsScannerActivity.java*
- **Primary Resource Layout:** *activity\_scan\_ingredients.xml*
- **Purpose:** The Ingredients Scanner Activity is used to enable logged in users to select an image from their device gallery, or to take an image using their device camera, and then scan the selected image to receive any scanned ingredients texts meaning. If there are any matching ingredients stored in the Firebase Realtime Database, then these ingredients and their meaning will be displayed to the user on screen.
- **Key package imports:**
  - *com.example.foodplannerapp.adapters.IngredientScannerRVAdapter*
  - *com.example.foodplannerapp.models.Ingredient*
  - *com.google.firebase.database.DataSnapshot, DatabaseReference, FirebaseDatabase*
  - *com.google.mlkit.vision.common.InputImage*
  - *com.google.mlkit.vision.text.Text, TextRecognition, TextRecognizer*

When the Ingredients Scanner Activity is launched, a Firebase Reference is created to the 'Ingredients' data stored in the database:

```
// set firebase database reference to 'Ingredients' data
ingredientsDBRef = FirebaseDatabase.getInstance().getReference().child("Ingredients");
```

This reference is used to retrieve and store all ingredient data in an Ingredient Model object ArrayList. This is the ArrayList of ingredients that will be queried for matching ingredients in a scanned image:

```
private ArrayList<Ingredient> ingredientsList;
```

```
ingredientsDBRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if(dataSnapshot.exists()){
            ingredientsList = new ArrayList<>();
            // store db ingredients to arraylist
            for(DataSnapshot ds : dataSnapshot.getChildren()){
                ingredientsList.add(ds.getValue(Ingredient.class));
            }
        }
    }
});
```

There are three buttons made available on the Ingredients Scanner screen, the Select Image button, the Capture Image button, and the Scan Ingredients button. When the Select Image button is clicked then the app's current image gallery permission is checked. If the user has not permitted access to their device, then the user will be prompted to permit access:

```
// @Reference - https://developer.huawei.com/consumer/en/doc/development/connectivity-Guides/adding-permissions-0000001064141094
// Reference description - Tutorial on how to request permissions
// request gallery permission if it is not already available
String[] permissionsStorage = {Manifest.permission.READ_MEDIA_IMAGES};
int requestExternalStorage = 2;
int externalStoragePermission = ActivityCompat.checkSelfPermission(context: this, Manifest.permission.READ_MEDIA_IMAGES);
if (externalStoragePermission != PackageManager.PERMISSION_GRANTED) { // if permission is not granted
    ActivityCompat.requestPermissions(activity: this, permissionsStorage, requestExternalStorage); // request permission
}
```

If the user grants permission to the app to access their photo gallery, then the Photo Gallery Image Selection activity will be launched using `ActivityResultLauncher`:

```
} else { // if permission is granted
    Intent intent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    selectImage.launch(intent); // launch select image activity
}
```

```
// @Reference - https://www.youtube.com/watch?v=f2odwvwrTKo
// Reference description - Tutorial on how to pick an image from a gallery using ActivityResultLauncher
// activity result handler for select image functionality
} usage
final ActivityResultLauncher<Intent> selectImage = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    result -> { // if the activity launch is successful
        if (result.getResultCode() == RESULT_OK && result.getData() != null) {
            Uri selectedImage = result.getData().getData();
            String[] filepath = {MediaStore.Images.Media.DATA};
            Cursor cursor = getContentResolver().query(selectedImage, filepath, selection: null, selectionArgs: null, sortOrder: null);
            cursor.moveToFirst();
            int columnIndex = cursor.getColumnIndex(filepath[0]);
            String picturePath = cursor.getString(columnIndex);
            cursor.close();

            // update the image view with the selected photo
            mImageView.setImageBitmap(BitmapFactory.decodeFile(picturePath));
            mImageView.buildDrawingCache();
            mSelectedImage = mImageView.getDrawingCache();

            // re-enable scan ingredients button
            mScanButton.setEnabled(true);

            // clear any currently listed ingredients
            clearIngredientsList();
        }
    }
);
```

The 'Capture Image' button functions similarly to the 'Select Image' button except that it allows the user to take an image using their device camera instead. The user must grant permission to their device camera and once the permission is granted then the `IngredientsScannerActivity` will start the `captureImage ActivityResultLauncher`.

Once an image is selected, it replaces the logo image on screen and the Scan Ingredients button is enabled. Once the Scan Ingredients button is clicked, the `runTextRecognition()` method will be run. Within this method, the user selected image is set as the ML Kit Vision `InputImage` and an instance of the Google ML Kit Vision `TextRecognizer` clients is created. The `Text Recognizer` client then processes the input image and if successful the text result is passed to the `processTextRecognitionResult()` method:

```

// @Reference - https://github.com/android/codelab-mlkit-android/blob/master/vision/final/app/src/main/java/com/google/codelab/mlkit/MainActivity.java
// Reference description - Code Lab sample application on how to implement Google ML Kit Vision Text Recognition
// method for running text recognition functionality
// usage - Ruby Lennon
private void runTextRecognition() {
    // create instance of input image using selected image bitmap
    InputImage image = InputImage.fromBitmap(mSelectedImage, 0);
    // create text recogniser
    TextRecognizer recognizer = TextRecognition.getClient();
    // disable scan ingredients button
    mScanButton.setEnabled(false);
    // pass the image to the process method
    recognizer.process(image)
        .addOnSuccessListener(
            texts -> {
                // enable scan ingredients button
                mScanButton.setEnabled(true);
                // run method for processing text and displaying results
                processTextRecognitionResult(texts);
            })
        .addOnFailureListener(
            e -> {
                // enable scan ingredients button
                mScanButton.setEnabled(true);
                // Task failed with an exception
                e.printStackTrace();
            });
}

```

In the processTextRecognitionResult() method, the text result is broken up into 'blocks' to check whether it is empty. If empty, a toast notification is displayed on screen to say that no text was detected. Otherwise, the text is stored to a single string using the getText() method called result\_text which is then further processed to remove certain characters. The processed text is then split by commas and stored in a string array. The scanned ingredients list is then stored in a string Array called scannedIngredients and this array is passed to a method called searchIngredients():

```

// @Reference - https://github.com/android/codelab-mlkit-android/blob/master/vision/final/app/src/main/java/com/google/codelab/mlkit/MainActivity.java
// Reference description - Code Lab sample application on how to implement Google ML Kit Vision Text Recognition
// method for processing text recognition result
// usage - Ruby Lennon
private void processTextRecognitionResult(Text texts) {
    // if no text is recognised execute the following
    List<Text.TextBlock> blocks = texts.getTextBlocks();
    if (blocks.size() == 0) {
        showToast();
        showNoIngredientsAlert();
        return;
    }

    // store the OCR text to variable
    String result_text = texts.getText();

    // update the stored OCR result text by formatting as below
    // remove new lines
    result_text = result_text.replaceAll( regex: "\\n", replacement: " ");
    // remove return carriages
    result_text = result_text.replaceAll( regex: "\\r", replacement: " ");
    // replace start bracket with comma
    result_text = result_text.replaceAll( regex: "\\[", replacement: "," );
    // remove end brackets
    result_text = result_text.replaceAll( regex: "\\]", replacement: "" );

    // split OCR result string by comma to create an Array of Ingredients
    String[] ingredients_list = result_text.split( regex: "\\s*,\\s*" );

    // add ingredients to string arraylist
    ArrayList<String> scannedIngredients = new ArrayList<>();
    Collections.addAll(scannedIngredients, ingredients_list);

    // pass the scanned ingredients to search method to find matching ingredients
    searchIngredients(scannedIngredients);
}

```



The `searchIngredients()` method creates a new `Ingredients Model ArrayList` called `allIngredientsList` which is updated with the ingredients stored in the original `ingredientList ArrayList`. Each ingredients in the `allIngredientsList ArrayList` is checked to see if it appears in the scanned ingredients `String Array list 'scannedIngredientsList'`. If the ingredient is present, then the ingredient is stored in a new `Ingredient Model Object ArrayList` called `matchingIngredientsList`. The `matchingIngredientsList ArrayList` is then used to populate the `Ingredients Scanner RecyclerView (ingredients card list)` using an instance of the `IngredientsScannerRVAdapter` class:

```
// method for searching for matching ingredients in database and displaying to user
// accepts text from scanned image
1 usage  Ruby Lennon
private void searchIngredients(ArrayList<String> scannedIngredientsList){
    // create arraylist for storing matching ingredients (scanned & db stored ingredients)
    ArrayList<Ingredient> matchingIngredientsList = new ArrayList<>();

    // add all ingredients from database to arraylist
    ArrayList<Ingredient> allIngredientsList = new ArrayList<>(ingredientsList);

    // compare ingredients list and if they match add ingredient to matching arraylist
    for(Ingredient object : allIngredientsList){
        if(scannedIngredientsList.toString().toLowerCase().contains(object.getIngredientName().toLowerCase())){
            matchingIngredientsList.add(object);
        }
    }

    // if there are no matching ingredients show a notification on screen
    if(matchingIngredientsList.isEmpty()){
        showNoIngredientsAlert();
    }else{ // if there are matching ingredients in the database then execute the following
        // update the ingredients recyclerview with matching ingredients
        IngredientScannerRVAdapter ingredientScannerRVAdapter = new IngredientScannerRVAdapter(matchingIngredientsList);
        ingredientsRV.setLayoutManager(new LinearLayoutManager(context, IngredientsScannerActivity.this));
        ingredientsRV.setAdapter(ingredientScannerRVAdapter);
        hideNoIngredientsAlert();
    }

    // hide loading progress bar
    loadingPB.setVisibility(View.GONE);
}
```

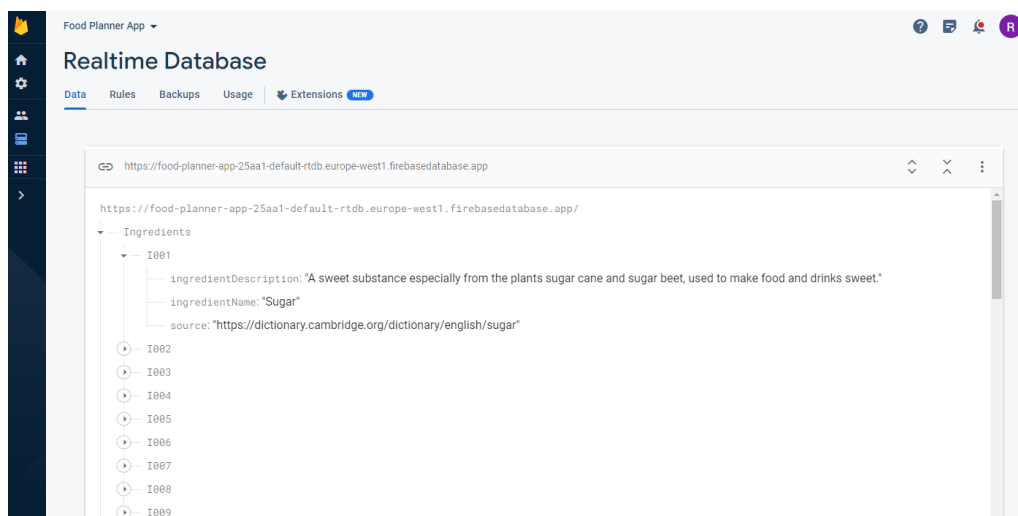


Figure 9 - Screenshot of Stored Ingredients in Firebase Realtime Database



### 2.3.3 Recipe Search Engine Feature

#### RecipeSearchActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/RecipeSearchActivity.java*
- **Primary Resource Layout:** *activity\_search\_recipes.xml*
- **Purpose:** The Recipe Search Activity enables users to search for public or owned recipes by Ingredients, Cuisine, Suitability and by Name.
- **Key package imports:**
  - *com.example.foodplannerapp.adapters.RecipeRVAdapter*
  - *com.example.foodplannerapp.models.Ingredient, Recipe*
  - *com.google.android.material.bottomsheet.BottomSheetDialog*
  - *com.google.firebase.auth.FirebaseAuth*
  - *com.google.firebase.database.ChildEventListener, DataSnapshot, DatabaseError, DatabaseReference, FirebaseDatabase, Query, ValueEventListener.*

The Recipe Search Activity first queries and stores recipes from the database that are public and or belong to the currently logged in user and stores them to a Recipe ArrayList. The user can then query the retrieved recipes by searching By Ingredients, Cuisine, Suitability or Name. When the 'Ingredients Search', 'Cuisine Search' and 'Suitability Search' buttons are clicked a corresponding multi-select alert dialog is displayed on screen. For example, below is the code for building the Ingredients Search dialog:

```
// show the ingredients search dialog with the ingredients retrieved from Firebase Realtime database
1 usage  ▸ Ruby Lennon
private void showIngredientsDialog(ArrayList<Ingredient> allIngredientsList){
    // create string arraylist to store all ingredients names
    ArrayList<String> ingredientsStringArrayList = new ArrayList<>();
    // store all ingredients objects to string arraylist
    for(Ingredient object : allIngredientsList){
        ingredientsStringArrayList.add(object.getIngredientName());
    }
    // store string ArrayList of string to object array
    Object[] objectsIngredientsArray = ingredientsStringArrayList.toArray();
    // convert array of ingredient objects to string array
    String[] stringIngredientsArray = Arrays.stream(objectsIngredientsArray).stream().map(Object::toString).toArray(String[]::new);
    // create arraylist to store selected ingredients in alert dialog
    final ArrayList<Object> selectedItems = new ArrayList<>();

    // @Reference - https://www.geeksforgeeks.org/how-to-implement-multiselect-dropdown-in-android/
    // Reference description - tutorial on how to implement a MultiSelect DropDown in Android
    // @Reference 2 - https://www.youtube.com/watch?v=46dbCl-47wE
    // Reference description - tutorial on how to implement a MultiSelect DropDown in Android
    // construct and configure alert dialog
    @SuppressWarnings("SetTextI18n") AlertDialog alertDialog = new AlertDialog.Builder(context, this)
        .setTitle("Search Recipes By Ingredients")// set the title
        .setMultiChoiceItems(stringIngredientsArray, checkedItems: null,
            (dialog, indexSelected, isChecked) -> {
                if (isChecked) {
                    selectedItems.add(indexSelected);// add the selected ingredient index
                } else if (selectedItems.contains(indexSelected)) {
                    selectedItems.remove(Integer.valueOf(indexSelected));// remove the selected ingredient index
                }
            })
        .setPositiveButton(text "Search", (dialog, id) -> { // if the search button is clicked
            // set the loading progress bar to visible
            loadingPB.setVisibility(View.VISIBLE);
            // run filterRecipes method and pass selected ingredients (indexes)
            filterRecipesByIngredients(selectedItems);
        })
        .setNegativeButton(text "Cancel", (dialog, id) -> {
        }).create();
    // show the alert dialog
    alertDialog.show();
}
```

When the user selects ingredients from the Ingredients Search dialog box and then clicks the 'Search' button, the filterRecipesByIngredients() method is called which queries the recipes list retrieved from the database for recipes with matching ingredients. The matching ingredients are stored in a new Recipe model ArrayList and the Recipe List (Recycler View) is updated to display the matching recipes:

```
// loop through the recipes and if they contain any of the selected ingredient names then
// store them to matchingRecipesList arraylist
for(Recipe recipe : originalRecipesList){
    for(String ingredient : filteredIngredients){
        // if the ingredient name is present in any of the stored recipes
        if(recipe.getRecipeIngredients().toLowerCase().contains(ingredient.toLowerCase())){
            matchingRecipesList.add(recipe);// then add the recipe to the matching recipes list
            break;
        }
    }
}

// recipe RV adapter config - display matching recipes in Recycler View
RecyclerView recipeRV = findViewById(R.id.idRVRecipes);
recipeRVAdapter = new RecipeRVAdapter(matchingRecipesList, context: this,
    recipeClickListener: this);
recipeRV.setLayoutManager(new LinearLayoutManager(context: this));
recipeRV.setAdapter(recipeRVAdapter);
```

The Cuisine and Suitability searches are implemented similarly. When the 'Cuisine Search' or the 'Suitability Search' button is clicked, the searchByCuisine() or searchBySuitability() methods are executed to build the multi-select alert dialog where users can select cuisines or suitability's to search by. Then when the dialog 'Search' button is clicked, the filterRecipesByCuisine() or filterRecipesBySuitability() methods are called to filter the original recipes list by the selected cuisine/suitability. If there are matching recipes, then the recipe list will be updated to display these recipes.

The Recipe Name search is implemented using a setOnQueryTextListener method for the Search View (Search Bar) element. When the Search View query text changes, the searchByName() method is called which filters the original recipe list by recipes with matching names.

The Clear Search button clears the currently applied search by executing the resetRecipesList() method.

The RecipeSearchActivity class implements the RecipeRVAdapter RecipeClickListener. When a Recipe is clicked in the Recycler View (Recipes List), a BottomSheetDialog is displayed on screen containing the clicked recipes details, and a 'View Details' button which when clicked directs to user to the Recipe details page.

## 2.3.4 Recipe Manager Feature

### AddRecipeActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/AddRecipeActivity.java*
- **Primary Resource Layout:** *activity\_add\_recipe.xml*
- **Purpose:** The Add Recipe Activity is used to enable logged in users to create private or public recipes. The recipes created in this Activity are stored in the Recipes section of the Firebase Realtime Database.
- **Key package imports:**
  - *com.example.foodplannerapp.models.Recipe*
  - *com.google.firebase.auth.FirebaseAuth*
  - *com.google.firebase.database.DataSnapshot, DatabaseReference, FirebaseDatabase*

When the Add Recipe Activity is launched, the `OnCreate()` method retrieves and assigns the currently logged in users ID from the current instance of Firebase Authenticate and assigns it to a local variable. The `OnCreate()` method also create a reference to the 'Recipes' object in the Firebase Realtime Database:

```
// get the currently signed in users ID from the current Firebase Auth instance
userID = Objects.requireNonNull(FirebaseAuth.getInstance().getCurrentUser()).getUid();
// get instance of Firebase realtime database
FirebaseDatabase firebaseDatabase = FirebaseDatabase.getInstance();
// create Firebase database 'Recipes' reference
databaseReference = firebaseDatabase.getReference("Recipes");
```

Dialog alerts are also built in the `OnCreate()` method for the recipe Suitability and Cuisine multi-select dialog alerts used for selecting the recipe cuisine/suitability.

Ingredients are added by clicking the 'Add Ingredient' button which opens an alert dialog where users can add a new ingredient. For every ingredient added, an ingredient 'card' view is added dynamically to the screen which contains the ingredient name and a delete button which when clicked deletes the ingredient. Once the Add Recipe button is clicked, if all recipe information has been provided, then a `StringBuilder` is used to combines all individual ingredients provided by the user into a single string value:

```
}else { // if all required fields are populated with values then execute the following
    // build a string with selected ingredients using string builder
    StringBuilder stringBuilder3 = new StringBuilder();
    // use a for loop
    for (int j = 0; j < ingredientArrayList.size(); j++) {
        // concatenate array values to string
        stringBuilder3.append(ingredientArrayList.get(j));
        // check condition
        if (j != ingredientArrayList.size() - 1) {
            // When j value not equal to ingredient list size - 1 add a comma
            stringBuilder3.append(", ");
        }
    }
    //store the string builder value to a string variable
    ingredientsSelectionString = stringBuilder3.toString();
}
```

The user input recipe information is extracted from the UI text input fields and used to create a new Recipe model object:

```
// @Reference - https://www.geeksforgeeks.org/user-authentication-and-crud-operation-with-firebase-realtime-database-in-android/
// Reference description - tutorial on how to add an object to the Firebase Realtime Database
// get user input text from fields
String recipeName = recipeNameEdt.getText().toString();
String recipeCookingTime = recipeCookingTimeEdt.getText().toString();
String recipePrepTime = recipePrepTimeEdt.getText().toString();
String recipeServings = recipeServingsEdt.getText().toString();
String recipeSuitedFor = recipeSuitabilityEdt.getText().toString();
String recipeCuisine = recipeCuisineEdt.getText().toString();
String recipeImg = recipeImgEdt.getText().toString();
String recipeLink = recipeLinkEdt.getText().toString();
String recipeDesc = recipeDescEdt.getText().toString();
String recipeMethod = recipeMethodEdt.getText().toString();
String recipeIngredients = ingredientsSelectionString;
Boolean recipePublic = recipePublicEdt.isChecked();
// get the database reference value for the new recipe object
recipeID = databaseReference.push().getKey();

// create a recipe object to store the new recipe values
Recipe recipe = new Recipe(recipeName, recipeCookingTime, recipePrepTime,
    recipeServings, recipeSuitedFor, recipeCuisine, recipeImg, recipeLink,
    recipeDesc, recipeMethod, recipeIngredients, recipePublic, recipeID, userID);
```

An Add Value Event listener is then created for the 'Recipes' database reference which is used to add the new recipe object to the Firebase Realtime Database:

```
// add the new recipe object to the Firebase Realtime database reference (Recipes)
RubyLennon
databaseReference.addValueEventListener(new ValueEventListener() {
    // execute the following if the add value action is successful
    RubyLennon
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        // hide the loading bar
        loadingPB.setVisibility(View.GONE);
        // add the recipe object to the database
        databaseReference.child(recipeID).setValue(recipe);
        // display a toast notification
        Toast.makeText(context, AddRecipeActivity.this, text: "Recipe Added", Toast.LENGTH_SHORT).show();
        // redirect the user to the MainActivity (My Recipes) screen
        startActivity(new Intent(context, AddRecipeActivity.this, MainActivity.class));
    }
    // execute the following if the add value action fails
    RubyLennon
    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Toast.makeText(context, AddRecipeActivity.this, text: "Error is " + error, Toast.LENGTH_SHORT).show();
    }
});
```

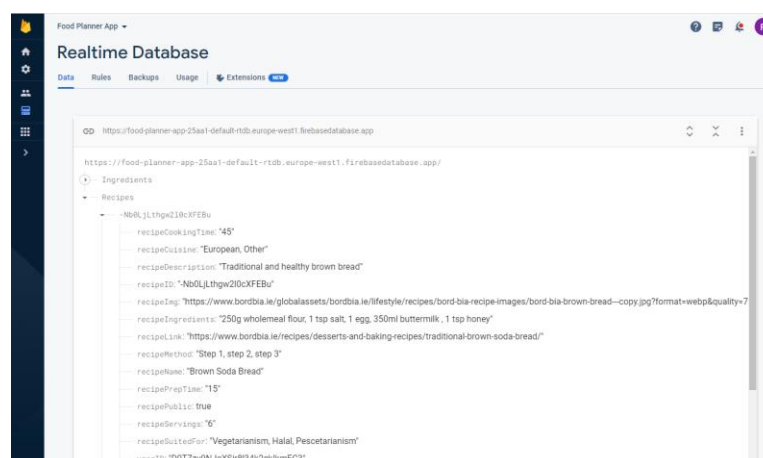


Figure 10 - Recipe created in the Firebase Realtime Database.

## EditRecipeActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - `app/src/main/java/com/example/foodplannerapp/activities/EditRecipeActivity.java`
- **Primary Resource Layout:** `activity_edit_recipe.xml`
- **Purpose:** The Edit Recipe Activity is used to enable logged in users to update or delete their own recipes.
- **Key package imports:**
  - `com.example.foodplannerapp.models.Recipe`
  - `com.google.firebase.database.DataSnapshot, DatabaseReference, FirebaseDatabase, ValueEventListener`

When the Edit Recipe Activity is launched, a Parcelable recipe object is retrieved which was passed to the Intent when the 'Edit Recipe' button was clicked on My Recipes screen:

```
// get the parcelable recipe object received when the edit recipe page was opened
Recipe recipe = getIntent().getParcelableExtra( name: "recipe");
```

This recipe information is then used to set the Edit Recipe screen input text fields with the recipe's current information:

```
// if the recipe object is not null update the input text fields with the recipe data
if (recipe != null){
    recipeNameEdt.setText(recipe.getRecipeName());
    recipeCookingTimeEdt.setText(recipe.getRecipeCookingTime());
    recipePrepTimeEdt.setText(recipe.getRecipePrepTime());
}
```

When the Update Recipe button is clicked, if all required information is provided, then a HashMap object is created using the updated recipe information:

```
// map the values to the object variables
Map<String, Object> map = new HashMap<>();
map.put("recipeName", recipeName);
map.put("recipeCookingTime", recipeCookingTime);
```

This mapped object is then used to update the recipe object in the Firebase Realtime Database using the database reference `.updateChildren()` method:

```
// database reference value event listener
Ruby Lennon
databaseReference.addValueEventListener(new ValueEventListener() {
    Ruby Lennon
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        // if the task is successful
        loadingPB.setVisibility(View.GONE); // hide the progress bar
        // update the recipe object using the mapped values defined above
        databaseReference.updateChildren(map);
        Toast.makeText(context: EditRecipeActivity.this, text: "Recipe Updated", Toast.LENGTH_SHORT).show();
        // redirect the user to the main activity screen (My Recipes)
        startActivity(new Intent( packageContext: EditRecipeActivity.this, MainActivity.class));
    }
});
```

The EditRecipeActivity also provides a Delete Recipe button which when clicked calls the `deleteRecipe()` method which deletes the recipe from the database:

```
// method for deleting recipe from Firebase Realtime database
Ruby Lennon
private void deleteRecipe(){
    databaseReference.removeValue(); // remove value from Recipes reference
    Toast.makeText(context: this, text: "Recipe Deleted", Toast.LENGTH_SHORT).show();
    // direct user to Main Activity (My Recipes)
    startActivity(new Intent( packageContext: EditRecipeActivity.this, MainActivity.class));
}
```

## MainActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - `app/src/main/java/com/example/foodplannerapp/activities/MainActivity.java`
- **Primary Resource Layout:** `activity_edit_recipe.xml`
- **Purpose:** The Main Activity is used to enable logged in users to view all owned recipes in a list. It also provides access to the Recipe Edit and Add Recipe screens.
- **Key package imports:**
  - `com.example.foodplannerapp.models.Recipe`
  - `com.example.foodplannerapp.adapters.RecipeRVAdapter`
  - `com.google.android.material.bottomsheet.BottomSheetDialog`
  - `com.google.firebase.database.DataSnapshot, DatabaseReference, FirebaseDatabase, ValueEventListener, Query, DatabaseError, ValueEventListener`

When the Main Activity is launched, a Firebase Database query is created to filter recipes returned from the Firebase Realtime database by the currently logged in users ID to ensure that the user can only view their own recipes:

```
// get the currently signed in users ID from the current Firebase Auth instance
String userID = Objects.requireNonNull(FirebaseAuth.getInstance().getCurrentUser()).getUid();
// get instance of Firebase realtime database
FirebaseDatabase firebaseDatabase = FirebaseDatabase.getInstance();
// create Firebase database 'Recipes' reference
DatabaseReference databaseReference = firebaseDatabase.getReference( path: "Recipes");
// set the query to filter the database reference to recipes that match the logged in users ID
query = databaseReference.orderByChild( path: "userID").equalTo(userID);
```

A ValueEventListener is created for the query which retrieves a snapshot of the recipes from the database belonging to the logged in user, stores them to a Recipe model ArrayList, and then uses the Recipe ArrayList to populate the Recipes List (Recycler View) on screen using the RecipesRVAdapter class:

```
// get all user recipe from Firebase Realtime Database
1 usage  ▸ Ruby Lennon
private void getAllUserRecipes() {
    recipeArrayList.clear();// clear the recipe arraylist
    // add value event listener to firebase query
    ▸ Ruby Lennon
    query.addListenerForSingleValueEvent(new ValueEventListener() {
        ▸ Ruby Lennon
        @SuppressWarnings("NotifyDataSetChanged")
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            loadingPB.setVisibility(View.GONE);// hide the loading bar
            if (dataSnapshot.exists()) { // if a data snapshot is retrieved from database using query
                for (DataSnapshot issue : dataSnapshot.getChildren()) {
                    // add the retrieved recipes to an ArrayList
                    recipeArrayList.add(issue.getValue(Recipe.class));
                    // notify the recycler view adapter the data has changed
                    recipeRVAdapter.notifyDataSetChanged();
                    // if there are no recipes returned then show the no recipes notice
                    if(recipeArrayList.isEmpty()){
                        showNoRecipesAlert();
                    }else{
                        hideNoRecipesAlert();
                    }
                }
            }
        }
    })
}
```

The MainActivity class implements the RecipeRVAdapter RecipeClickInterface. When a Recipe is clicked in the Recycler View (Recipes List), a BottomSheetDialog is displayed on screen containing the clicked recipes details, and two buttons which when clicked direct to the selected recipes Edit and View Details Screens:

```
// @Reference - https://www.geeksforgeeks.org/user-authentication-and-crud-operation-with-firebase-realtime-database-in-android/
// Reference description - tutorial on how to display a bottom sheet dialog for a clicked item in Recycler View
// if a recipe is clicked
! usage ▲ Ruby Lennon
@Override
public void onRecipeClick(int position) {
    // then display the bottom sheet dialog for that recipe
    displayBottomSheet(recipeArrayList.get(position));
}

// @Reference - https://www.geeksforgeeks.org/user-authentication-and-crud-operation-with-firebase-realtime-database-in-android/
// Reference description - tutorial on how to display a bottom sheet dialog for a clicked item in Recycler View
// bottom sheet dialog builder and functionality
! usage ▲ Ruby Lennon
@SuppressWarnings("SetTextI18n")
private void displayBottomSheet(Recipe recipe){
    // create a bottom sheet dialog object
    final BottomSheetDialog bottomSheetDialog = new BottomSheetDialog(context: this);
    // create a view object and set the layouts to be inflated
    View layout = LayoutInflater.from(context: this).inflate(R.layout.bottom_sheet_dialog_user_recipe,
        bottomSheetRL);
    bottomSheetDialog setContentView(layout);
    bottomSheetDialog.setCancelable(false);
    bottomSheetDialog.setCanceledOnTouchOutside(true);
    bottomSheetDialog.show(); // show the bottom sheet dialog
}
```

The MainActivity contains a floating action button of a recipe cookbook icon that when clicked directs the user to the Add Recipe screen:

```
// if the add floating actions button is clicked then direct user to add recipe page
addFAB.setOnClickListener(v -> startActivity(new Intent(packageContext: MainActivity.this,
    AddRecipeActivity.class)));
```

## ViewRecipesActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/ViewRecipeActivity.java*
- **Primary Resource Layout:** *activity\_view\_recipe.xml*
- **Purpose:** The View Recipe Activity is used to enable logged in users to view a recipe details screen. It also provides access to the ‘Add to Meal Plan’ and ‘View Recipe in Browser’ buttons.
- **Key package imports:**
  - *com.example.foodplannerapp.models.Recipe, Meal, MeallIngredient*
  - *com.example.foodplannerapp.fragments.DatePickerFragment*
  - *com.google.firebase.database.DatabaseReference, FirebaseDatabase*
  - *com.google.firebase.auth.FirebaseAuth*

Similarly, to the Edit Recipe page, a Parcelable recipe object is obtained from the instance when the View Recipe Activity is launched which is used to populate the recipe details on screen. The View Recipe Activity implements DatePickerDialog.OnDateSetListener. When the ‘Add to Meal Plan’ button is clicked, the DatePickerFragment is initiated, and a Calander dialog appears on screen. Once the user selects a calendar date and clicks OK, the onDateSet() method will be executed. In the onDateSet() method, a Meal Object is constructed using the



recipe data and the meal object is then added to the Firebase Realtime database. For every recipe ingredient, a child Ingredient object is added to the Meal object created in the database – this is required for the Shopping List feature:

```
// get the ID for the new meal
String mealPlanID = databaseReferenceMealPlan.push().getKey();

// create a meal object to store the new meal values
Meal meal = new Meal(currentDateStringShort, mealPlanID, recipeName, recipeCookingTime,
    recipePrepTime, recipeServings, recipeSuitedFor, recipeCuisine, recipeImg,
    recipeLink, recipeDesc, recipeMethod, recipeIngredients, recipePublic, recipeID,
    userID, currentDateStringLong);
assert mealPlanID != null;
// create new meal plan object
databaseReferenceMealPlan.child(mealPlanID).setValue(meal);

// store ingredients to ingredients child object of the new meal object
for (String ingredient : ingredientsArray) { // for every recipe ingredient
    // create a new meal ingredient object
    MealIngredient mealIngredient = new MealIngredient(ingredient.trim(), purchased: "false");
    // add the new meal ingredient object as a child of the new meal created
    databaseReferenceMealPlan.child(mealPlanID).child("ingredients").push().setValue(mealIngredient);
}
```

### PublicRecipesActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/PublicRecipeActivity.java*
- **Primary Resource Layout:** *activity\_public\_recipes.xml*
- **Purpose:** The Public Recipes Activity is used to enable logged in users to view all public recipes in a list. It also provides access to the Recipe View Screen for public recipe.
- **Key package imports:**
  - *com.example.foodplannerapp.models.Recipe*
  - *com.example.foodplannerapp.adapters.RecipeRVAdapter*
  - *com.google.android.material.bottomsheet.BottomSheetDialog*
  - *com.google.firebase.database.DataSnapshot, DatabaseReference, FirebaseDatabase, ValueEventListener, Query, DatabaseError, ValueEventListener*

### DatePickerFragment.java (Fragment Class in Fragments Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/fragments/DatePickerFragment.java*
- **Purpose:** Dialog fragment class for meal plan date selection on recipe view page
- **Key package imports:**
  - *android.app.DatePickerDialog, Dialog*
  - *androidx.fragment.app.DialogFragment*
  - *java.util.Calendar*



## 2.3.5 Meal Planner & Shopping List Features

### MealPlanActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - `app/src/main/java/com/example/foodplannerapp/activities/MealPlanActivity.java`
- **Primary Resource Layout:** `activity_meal_plan.xml`
- **Purpose:** The Meal Planner Activity is used to enable logged in users to view all their scheduled meals in a list. It also provides access to the 'Remove Meal from Plan' and 'View Meal Details and Shopping List' buttons.
- **Key package imports:**
  - `com.example.foodplannerapp.models.Recipe`
  - `com.example.foodplannerapp.adapters.RecipeRVAdapter`
  - `com.google.firebase.database.DataSnapshot, DatabaseReference, FirebaseDatabase, ValueEventListener, Query, DatabaseError, ValueEventListener`

Similarly, to the Main Activity, when the Meal Planner is launched, a Firebase Database query is created to filter meals returned from the Firebase Realtime database by the currently logged in users ID to ensure that the user can only view their own scheduled meals. A ValueEventListener is created for the query which retrieves a snapshot of the users' meals from the database, stores them to a Meal model ArrayList, and then uses the Meal ArrayList to populate the Meals List (Recycler View) on screen using the MealRVAdapter class. The MealPlanActivity class implements the MealPlanRVAdapter MealPlanClickListener. When a Meal is clicked in the Recycler View (Meal List), a BottomSheetDialog is displayed on screen containing the clicked meals details. The bottom sheet dialog contains a Remove Meal Plan from Plan button which when clicked enables the user to remove the meal from their plan. The bottom sheet dialog also contains a 'View Details & Shopping List' button which when clicked directs the user to the Meal details screen:

```
// if the bottom sheet dialog view details meal button is clicked then direct user to
// view meal page and pass selected meal object
viewDetailsBtn.setOnClickListener(v -> {
    Intent i = new Intent( packageContext, MealPlanActivity.this, ViewMealPlanActivity.class);
    i.putExtra( name: "MealPlan", meal);
    startActivity(i); // start View meal activity
});
```

### SortMeals.java (Interface Class in Interfaces Package)

- **Project File Location:**
  - `java/com/example/foodplannerapp/interfaces/SortMeals.java`
- **Purpose:** The purpose of this class is to sort the Meal ArrayList by date (Ascending order) to list Meals in the Meal Planner by date in ascending order.
- **Key package imports:**
  - `com.example.foodplannerapp.models.Meal`
  - `java.util.Comparator`

```

// @Reference - https://www.geeksforgeeks.org/java-program-to-sort-objects-in-arraylist-by-date/
// Reference description - tutorial on how to sort Objects in ArrayList by Date
4 usages Ruby Lennon *
public class SortMeals implements Comparator<Meal> {
    // comparison method
    Ruby Lennon *
    public int compare(Meal mealA, Meal mealB) {
        // return value after comparing the objects to sort meals in Ascending order
        return mealA.getDateShort().compareTo(mealB.getDateShort());
    }
}

```

## ViewMealPlanActivity.java (Activity Class in Activities Package)

- **Project File Location:**
  - *app/src/main/java/com/example/foodplannerapp/activities/ViewMealPlanActivity.java*
- **Primary Resource Layout:** *activity\_meal\_plan.xml*
- **Purpose:** The Meal Planner Activity is launched when a user clicks the ‘View Meal Details and Shopping List’ button on the Meal Planner Activity screen. This is where the Meal details and shopping list are located.
- **Key package imports:**
  - *com.example.foodplannerapp.models.Meal, Meal Ingredients*
  - *com.google.firebase.database.DataSnapshot, DatabaseReference, FirebaseDatabase, ValueEventListener, Query, DatabaseError*

Similarly, to the View Recipe Activity, a Parcelable meal object is obtained from the instance when the View Meal Activity is launched which is used to populate the meal details on screen. On this screen, the user can click the View Source recipe button which will launch their default device browser and navigate them to the Source Recipe URL. On this screen, the meal Shopping List is available. For every ingredient stored for the Meal, an ingredient card view is added to the shopping list layout dynamically using the `addIngredientCard()` method below:

```

// @Reference - https://codevedanam.blogspot.com/2021/04/dynamic-views-in-android.html
// Reference description - tutorial on how to add dynamic views in Android
// add ingredient cards to layout accepts (ingredient name, purchased value, key)
1 usage Ruby Lennon
private void addIngredientCard(String ingredient, String purchased, String key) {
    @SuppressWarnings("InflateParams") final View view = getLayoutInflater()
        .inflate(R.layout.shopping_list_item, root: null);

    // find the layout element by ID and assign to variables
    TextView nameView = view.findViewById(R.id.name);
    CheckBox checkBox = view.findViewById(R.id.checkBox);

    // set the ingredient card text to the ingredient name
    nameView.setText(ingredient);
}

```

The user can then check or uncheck ingredients listed in the Shopping List container to mark whether the item has been purchased or not. Checking and unchecking the ingredients checkbox updates the meal ingredients purchased value in the Firebase Realtime Database:

```
// ingredient card checkbox checking functionality for updating purchased value in the database
checkbox.setOnCheckedChangeListener((buttonView, isChecked) -> {
    if(checkbox.isChecked()){
        Toast.makeText( context: ViewMealPlanActivity.this, text: "Shopping List Updated",
            Toast.LENGTH_SHORT).show();
        databaseReferenceIngredients.child(key).child( pathString: "purchased")
            .setValue("true");// set the meal ingredients purchased value to true
    }else{// if the ingredient checkbox is checked then set the ingredients purchased value in the database
        // to false
        Toast.makeText( context: ViewMealPlanActivity.this, text: "Shopping List Updated",
            Toast.LENGTH_SHORT).show();
        databaseReferenceIngredients.child(key).child( pathString: "purchased")
            .setValue("false");// set the meal ingredients purchased value to false in the database
    }
});
```

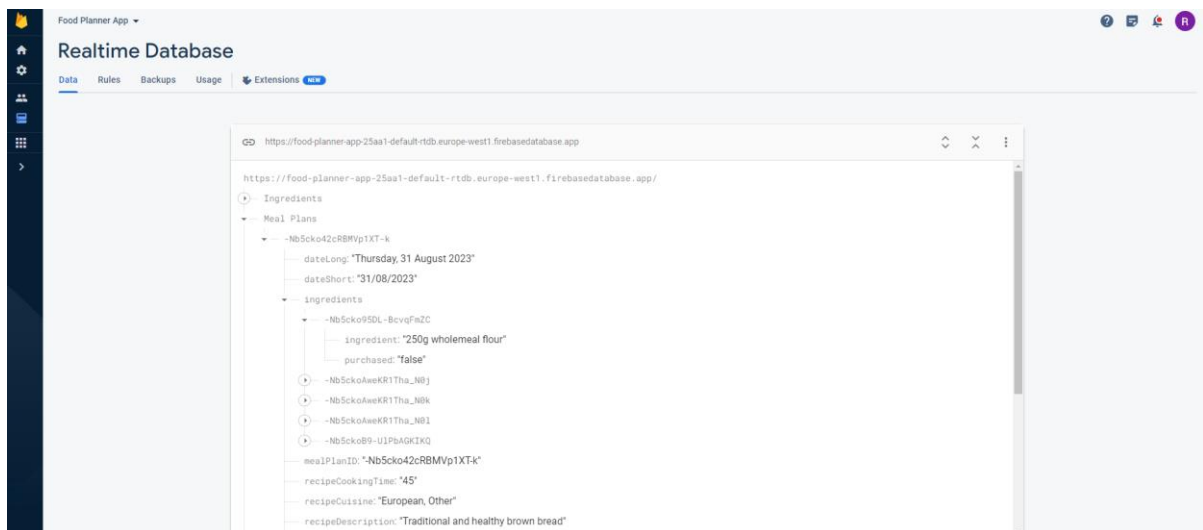


Figure 11 - Screenshot of Meal Object and Child Ingredients Object in Firebase Realtime Database

## 2.4 Graphical User Interface (GUI)

### User Login Screen

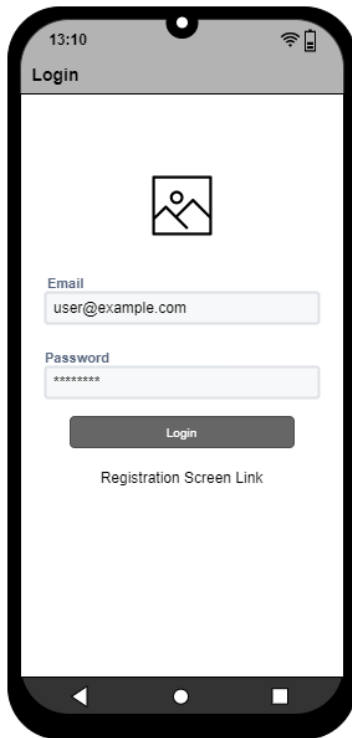


Figure 12 - Login Screen Wireframe.

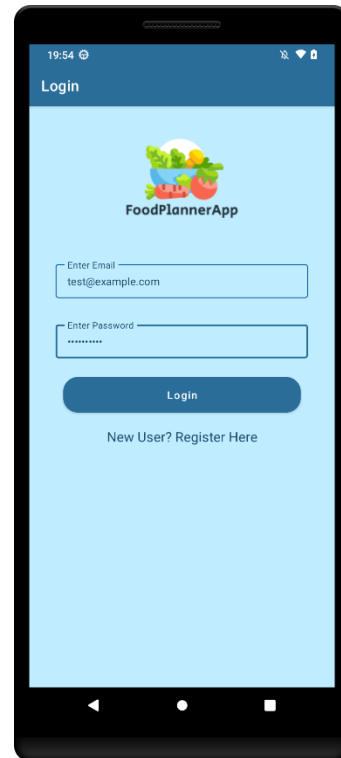


Figure 13 - Screenshot of Login Screen.

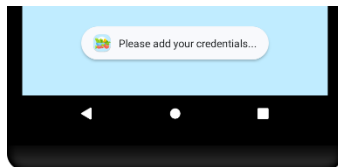


Figure 14 - Toast notification displayed when email and or password are not provided when the Login button is clicked.

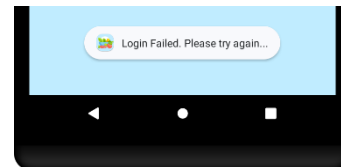


Figure 15 - Toast notification displayed when login fails.

The Login Screen is where the user can log into the application by providing their account email address and password. If the login is successful, they will be directed to the 'My Recipes' screen. If a user provides incorrect login credentials, then an error toast notification will appear on screen. The user is directed to the Login screen whenever they select the log out action. There is a link on the Login screen which directs the user to the Registration screen when clicked.

## User Registration Screen

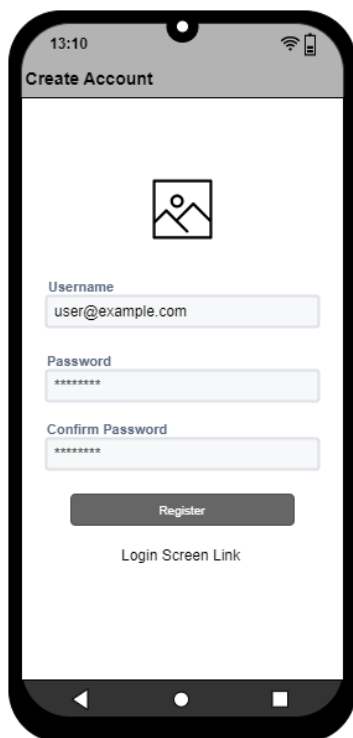


Figure 16 - Registration Screen Wireframe.

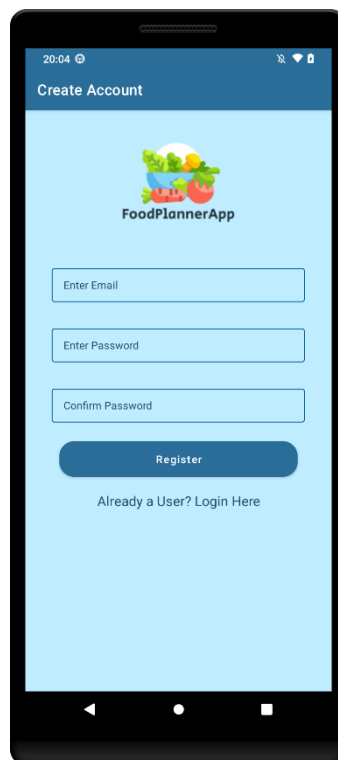


Figure 17 - Screenshot of Registration Screen.

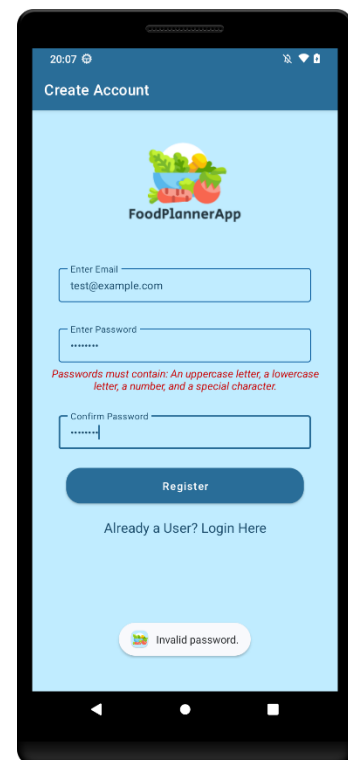


Figure 18 - Screenshot of Password Requirements alert.

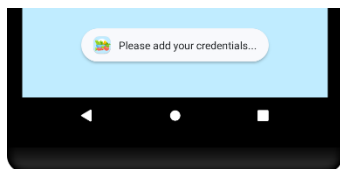


Figure 19 - Toast notification displayed when email and or passwords are not provided.

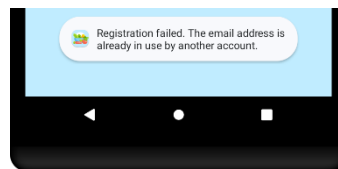


Figure 20 - Toast notification displayed when already in use email is provided.

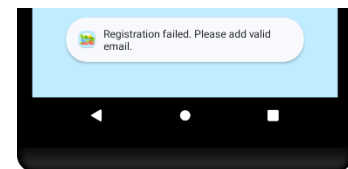


Figure 21 - Toast notification displayed when invalid email address is provided.

The Registration screen is where the user can register for a new account or navigate to the login screen. On this screen users must provide a valid email address, valid password, and a matching valid password. Users cannot register for a new account using an email address that is not in a correct email format or an email address already in use by another account. If an invalid email address is provided an error notification will be displayed on screen. The password provided by the user must contain at least one uppercase, lowercase, digit, and special character and must be at least 8 characters in length. If these requirements are not met, then an error toast alert will be displayed on screen. If a valid email address and matching valid passwords are provided an account will be created for the user when the Register button is clicked, and the user will be directed to the 'My Recipes' screen.

## Edit Account Screen

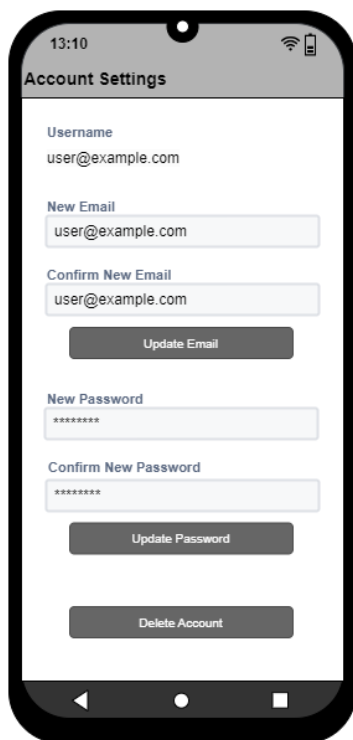


Figure 22 - Account Settings Screen Wireframe.

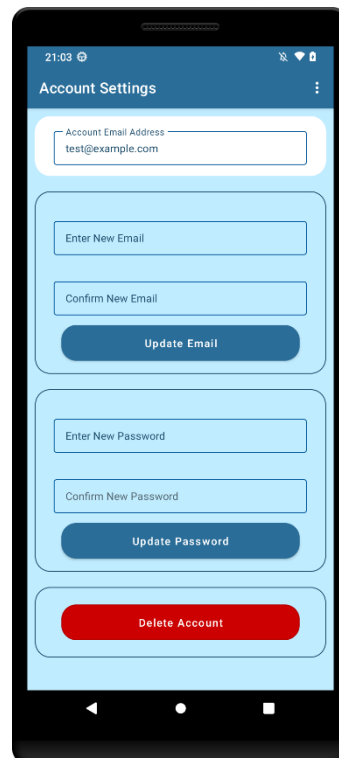


Figure 23 - Screenshot of Account Settings Screen.

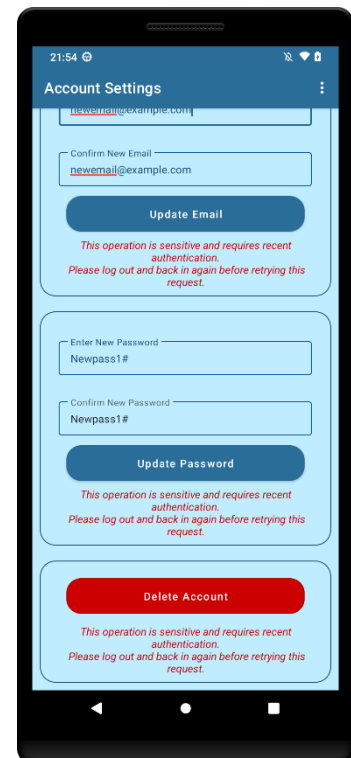


Figure 24 - Screenshot of Sign-in required alert for each Account Settings actions.

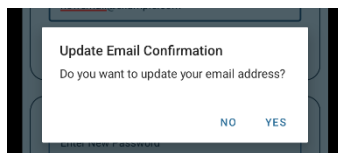


Figure 25 - Update Email Confirmation box.

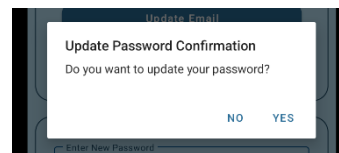


Figure 26 - Update Password Confirmation box.

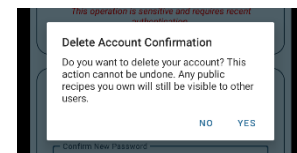


Figure 27 - Delete Account Confirmation box.

The Account Settings Screen is where users can update their account email address and password or delete their account. All actions on this screen are time sensitive in relation to when the user last logged in (5 minutes). If a user tries to execute one of the available actions after 5 minutes since they last logged in, then an error alert will be displayed on screen. If a valid email or password is not provided, an error alert will be displayed on screen and the password or email address will not be updated. The user must click 'Yes' in a confirmation dialog that appears when each of the actions is attempted to be executed on this page. The signed in user's current account email address is displayed at the top of this screen.

# Ingredients Scanner Screen

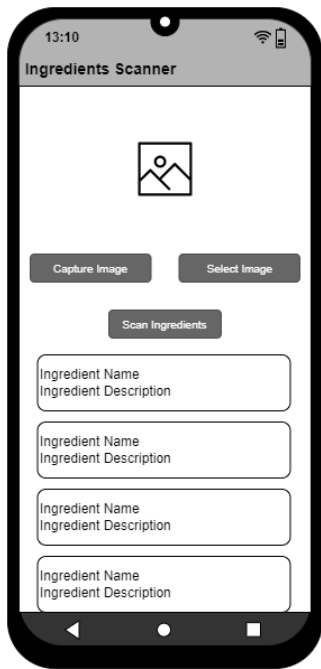


Figure 28 - Ingredients Scanner Screen Wireframe.

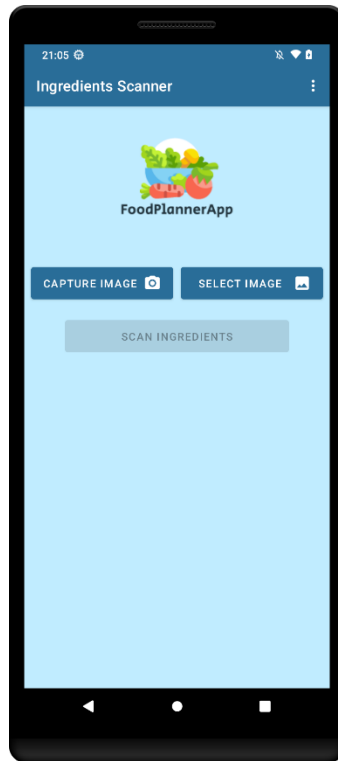


Figure 29 - Screenshot of Ingredient Scanner Default Screen.

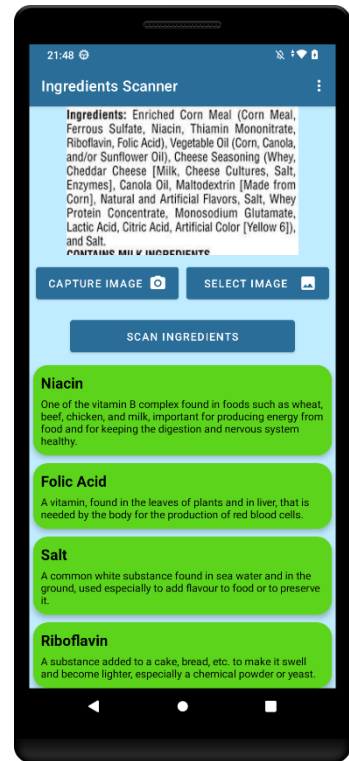


Figure 30 - Screenshot of Ingredient Scanner Screen displaying detected ingredients.

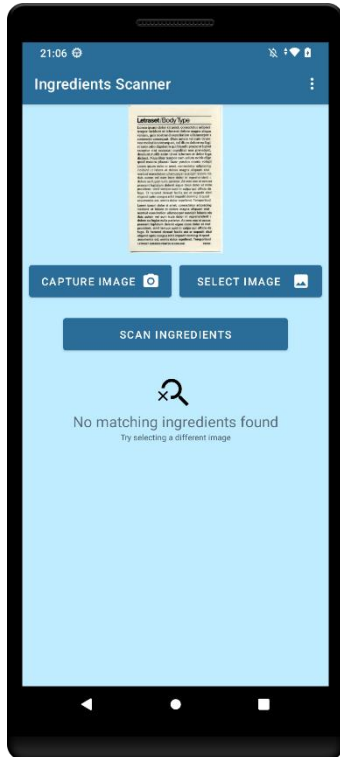


Figure 31 - Screenshot of Ingredient Scanner Screen displaying no matching results text result.



Figure 32 - Screenshot of Camera activated when Capture Image button is clicked.

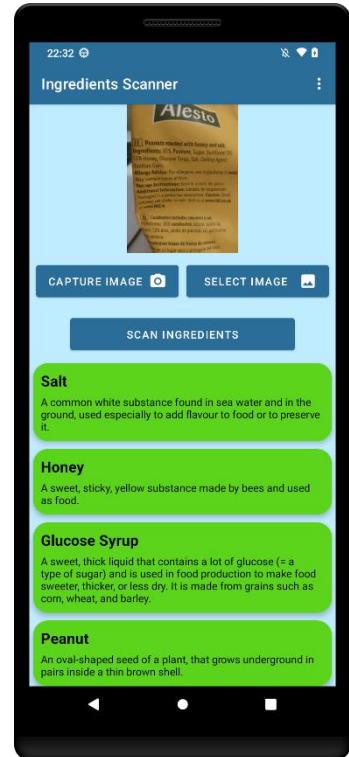


Figure 33 - Screenshot of Ingredient Scanner Screen displaying detected ingredients.

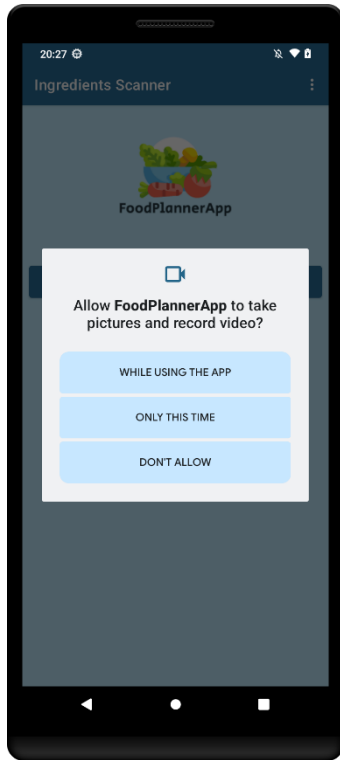


Figure 34 – Screenshot of request for user device camera permission required for Image Capture functionality.

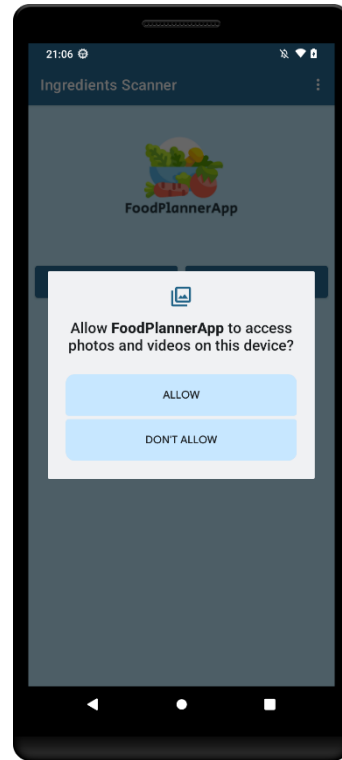


Figure 35 – Screenshot of request for user device image gallery permission required for Select Image functionality.

The Ingredients Scanner Screen is where the user can scan images of food ingredients lists to receive the ingredients meaning. The user can either select an existing image from their device image gallery by clicking the 'Select Image' button or take a photo using their device camera by clicking the 'Capture Image' button. The selected image will then be displayed on the screen. The 'Scan Ingredients' button scan the image for ingredient text and if there are any matching ingredients stored in the Google Firebase Realtime Database, they will be displayed on the screen. The matching ingredient name and description will be displayed in a single card listed in a Recycler View which the user will be able to scroll. If text is recognised in the image, but there are no matching ingredients in the database, then this will be displayed on screen. If the user is using the application for the first time, then they will need to grant access to their device image gallery and camera.



## 'Add Recipe' Screen

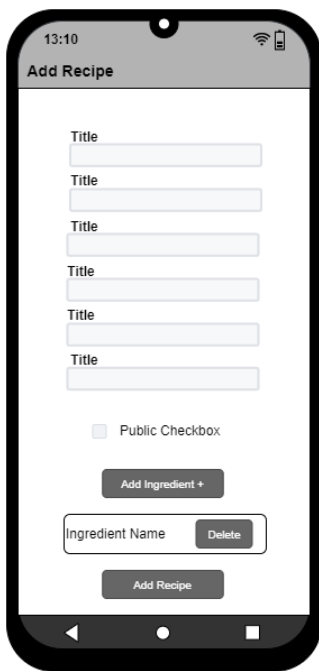


Figure 36 - Add Recipe Screen Wireframe.

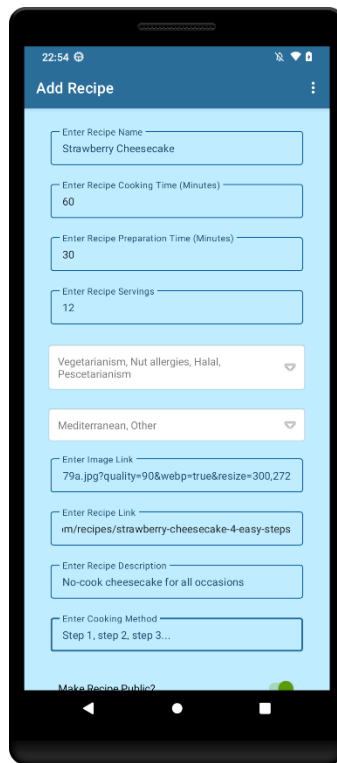


Figure 37 - Screenshot of Add Recipe screen (first half).

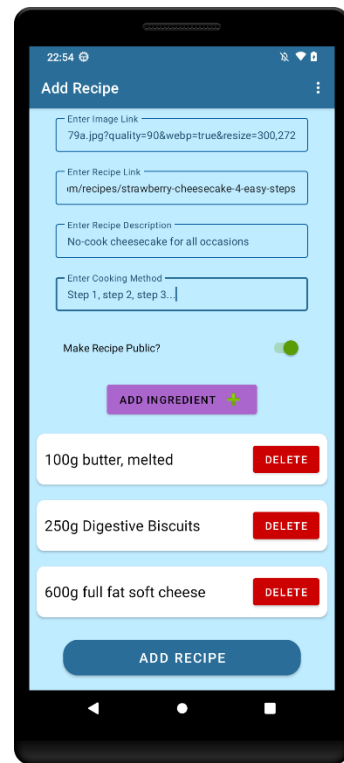


Figure 38 - Screenshot of Add Recipe screen (second half).

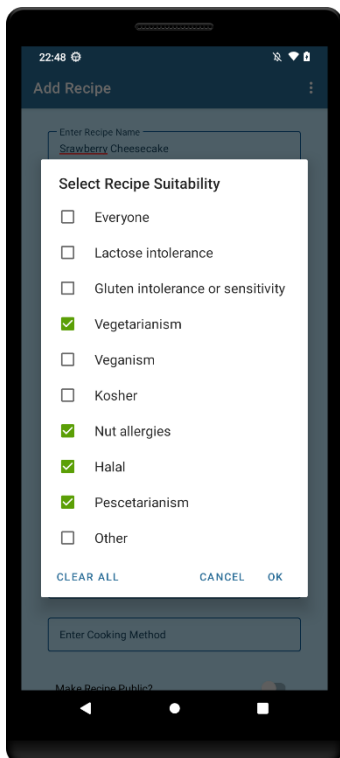


Figure 39 - Screenshot of Select Recipe Suitability multi-select alert dialog.

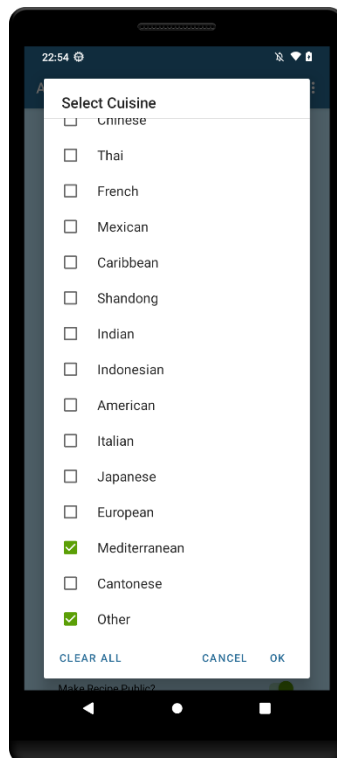


Figure 40 - Screenshot of Select Recipe Cuisine multi-select alert dialog.

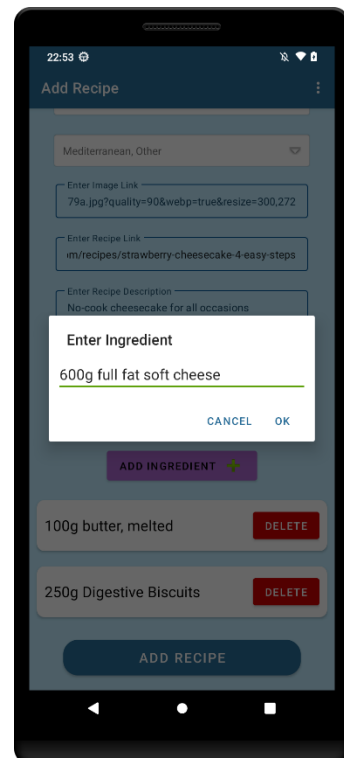


Figure 41 - Screenshot of Add Ingredients alert dialog.

The Add Recipe screen is where users can create a new recipe by adding all the following required data. If a mandatory field is not completed when the Add Recipe button is clicked, then an error toast alert is displayed on screen. When users select the Suitability or Cuisine fields a corresponding alert dialog appears allowing them to select multiple suitability's or cuisines. When the user selects the Add Ingredient button an alert dialog appears allowing the user to add an ingredient. Ingredients added using this alert dialog will be listed in a Recycler View on the screen. Added ingredients can then be deleted using the ingredient card 'Delete' button.

## 'Edit Recipe' Screen

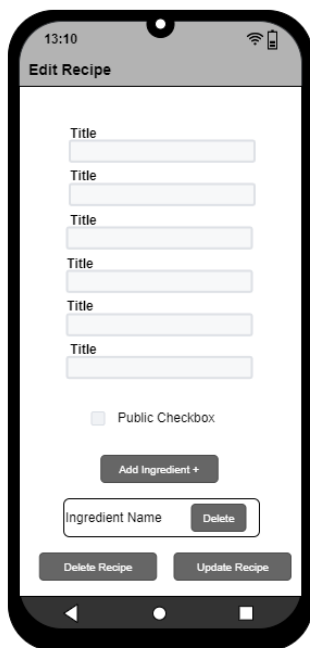


Figure 42 - Edit Recipe Screen Wireframe.

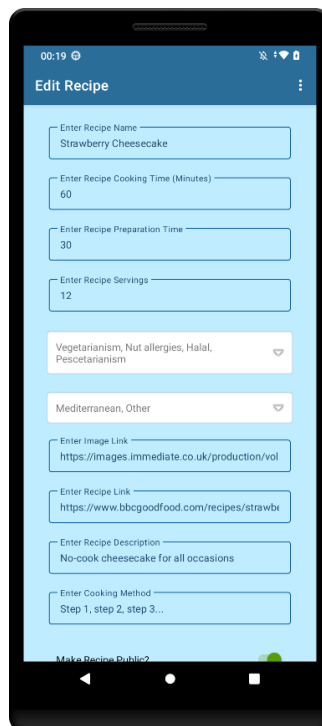


Figure 43 - Screenshot of Edit Recipe screen (first half).

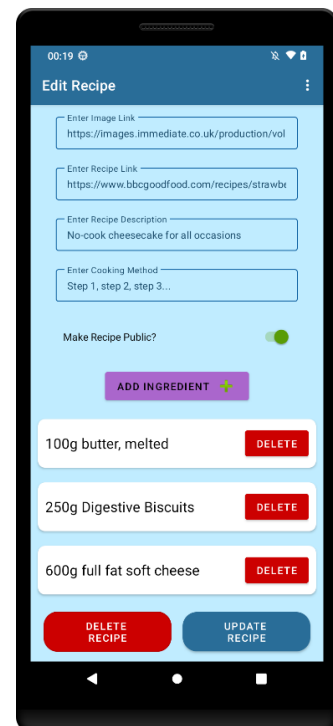


Figure 44 - Screenshot of Edit Recipe screen (second half).

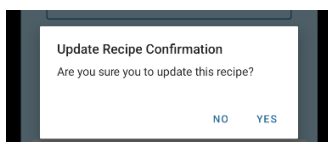


Figure 45 - Screenshot of confirmation modal for updating recipe.

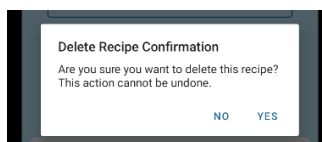


Figure 46 - Screenshot of confirmation modal for deleting recipe.

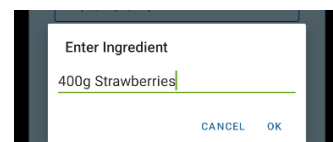


Figure 47 - Screenshot of Add Ingredient alert dialog.

The Edit Recipe screen is where users can update or delete their recipes. The user can update the existing recipe details by adding the updated details, clicking the Update Recipe button, and then clicking Yes in the Update Recipe Confirmation modal. The user can delete a recipe by clicking the Delete Recipe button and then clicking Yes in the Delete Recipe Confirmation modal.

## 'My Recipes' Screen

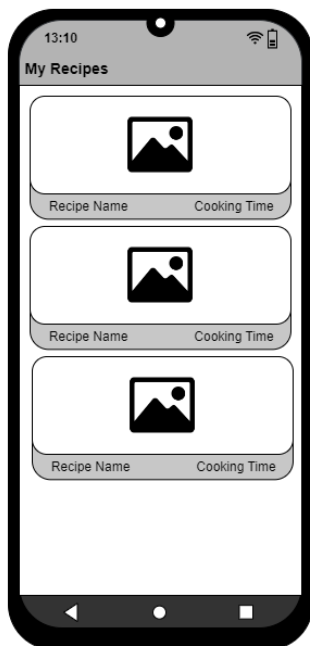


Figure 48 - My Recipes Screen Wireframe.



Figure 49 - My Recipes Screen Wireframe (bottom sheet dialog).



Figure 50 - Screenshot of the My Recipes screen with Add Recipe button on bottom right.

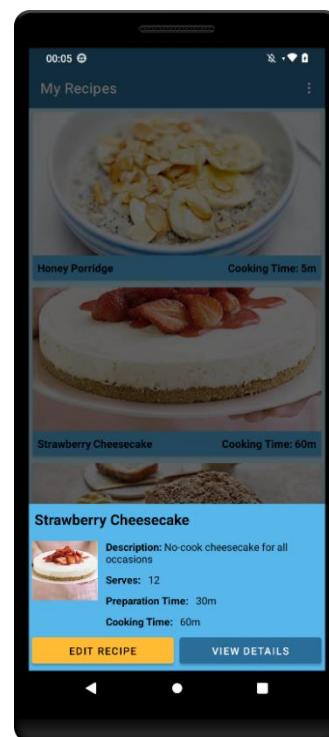


Figure 51 - Screenshot of My Recipes screen (with bottom sheet dialog open).

The My Recipes screen is where users can view their recipes. Once a recipe is clicked the recipe bottom sheet dialog is displayed, displaying recipe details, and buttons to the Edit Recipe or Recipe Details screens. When clicked, the cookbook icon on the bottom right of the screen directs the user to the Add Recipe screen.

## 'Public Recipes' Screen

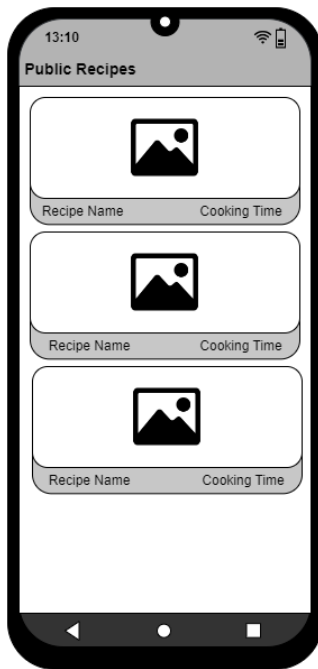


Figure 52 - Public Recipes Screen Wireframe.



Figure 53 - Public Recipes Screen Wireframe (bottom sheet dialog).



Figure 54 - Screenshot of the Public Recipes Screen.

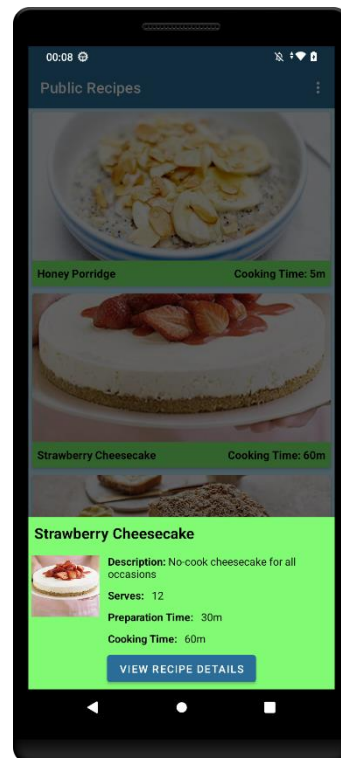


Figure 55 - Screenshot of the Public Recipes Screen (bottom sheet dialog).

The Public Recipes screen is where users can view public recipes. Once a recipe is clicked the recipe bottom sheet dialog is displayed, displaying recipe details, and a button to the Recipe Details screen.

## 'Recipe Details' Screen



Figure 56 - Recipe Details Screen Wireframe.

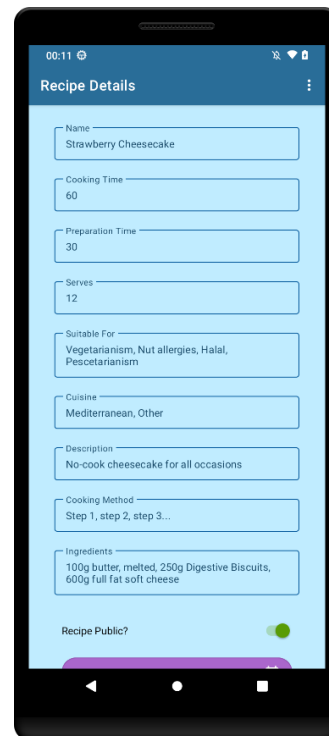


Figure 57 - Screenshot Recipe Details Screen (first half).

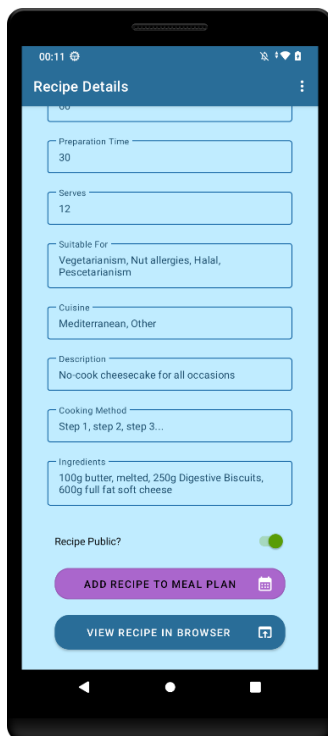


Figure 58 - Screenshot Recipe Details Screen (second half).

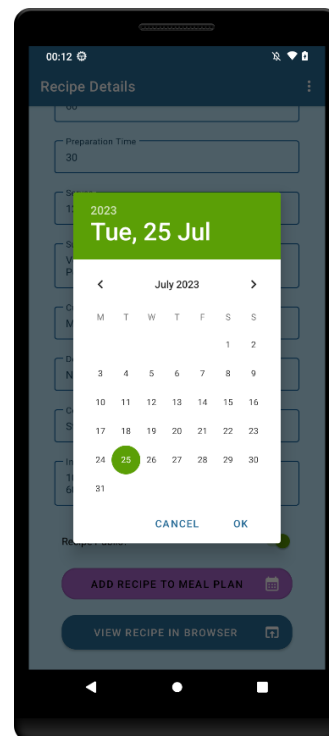


Figure 59 - Screenshot of Meal Plan date selection.

The Recipe Details screen is where users can view a recipe's details. The Add Recipe to Meal Plan button when click will display a calendar fragment on screen which allows the users to select a date which will add the recipe to their meal plan for the selected date and then direct the user to the Meal Planner screen. The View Recipe in Browser button will open the source recipe in their default browser.

## 'Meal Planner' Screen

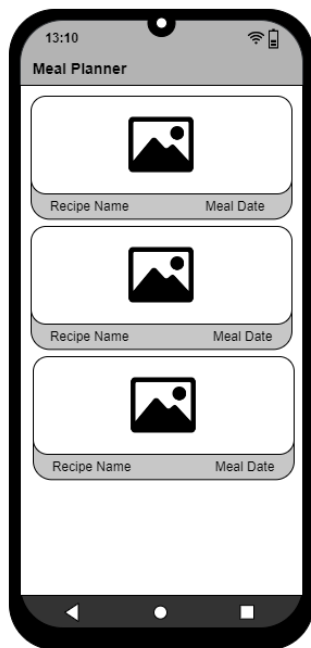


Figure 60 - Meal Planner Screen Wireframe.



Figure 61 - Meal Planner Screen Wireframe (bottom sheet dialog).

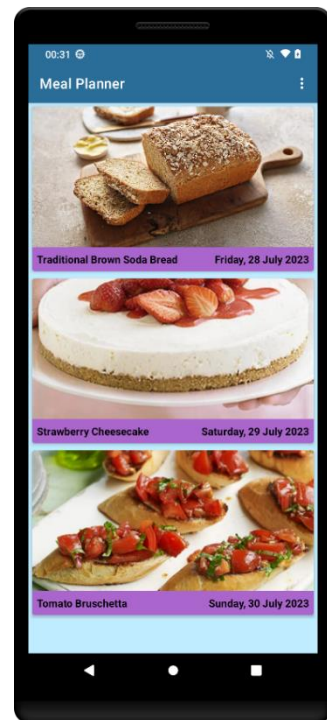


Figure 62 - Screenshot of Meal Planner Screen.

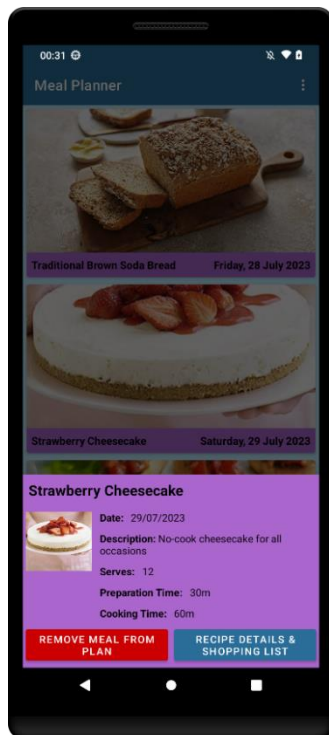


Figure 63 - Screenshot of Meal Planner Screen (bottom sheet dialog).

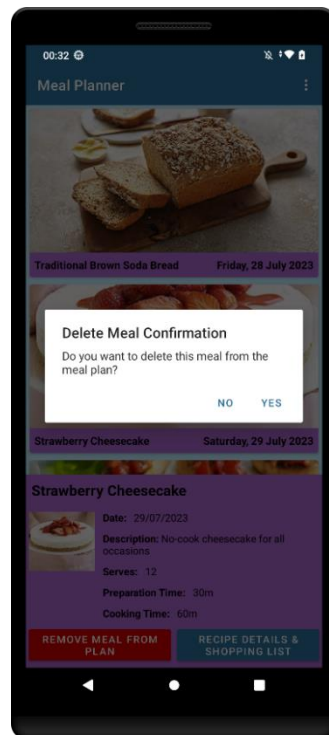


Figure 64 - Screenshot of Remove Meal from Plan confirmation modal.

The Meal Planner screen is where the user can view all Meals added to their Meal Plan. When a meal is selected from the list the meal bottom sheet dialog is displayed where the user can remove the meal from their plan or navigate to the Meal Detail and Shopping List screen.

## 'Meal Details' Screen

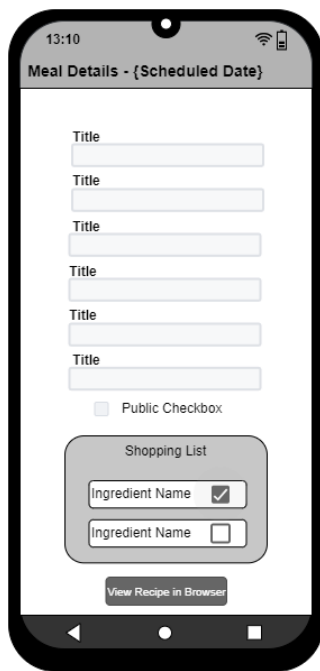


Figure 65 - Meal Details Screen Wireframe.

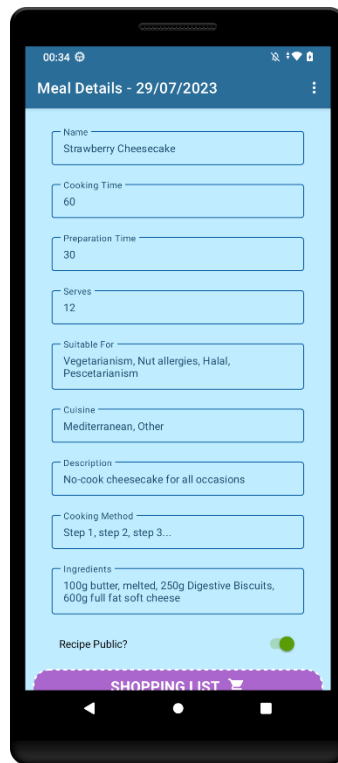


Figure 66 - Screenshot of Meal details screen (first half).

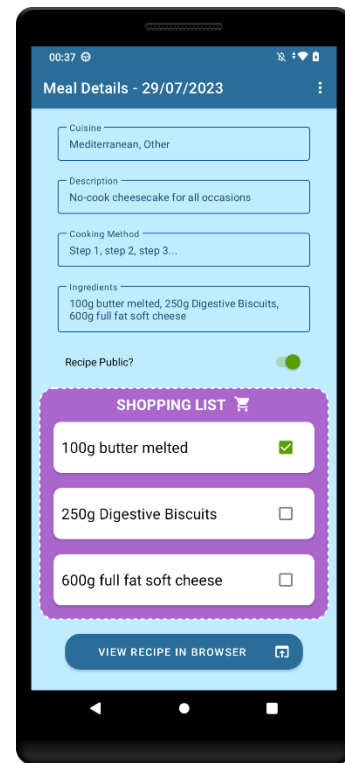


Figure 67 - Screenshot of Meal Details screen (second half).

The Meal Details screen allows the user to view the recipe details for the Meal, view their shopping list for the meal, and to view the recipe in their default browser by clicking the View Recipe in Browser button. The user can update their shopping list on this page. Checking an ingredient in this shopping list saves its purchased status.

# 'Recipe Search' Screen



Figure 68 - Recipe Search Screen Wireframe.

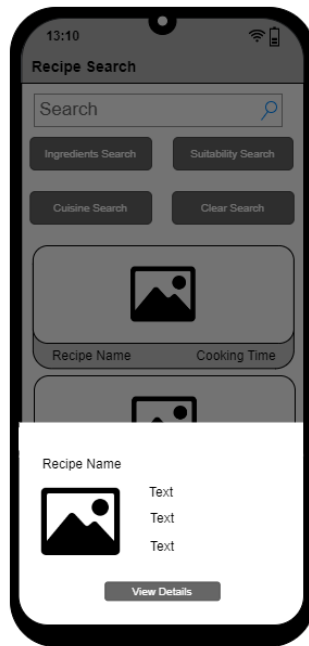


Figure 69 - Recipe Search Screen Wireframe (bottom sheet dialog).

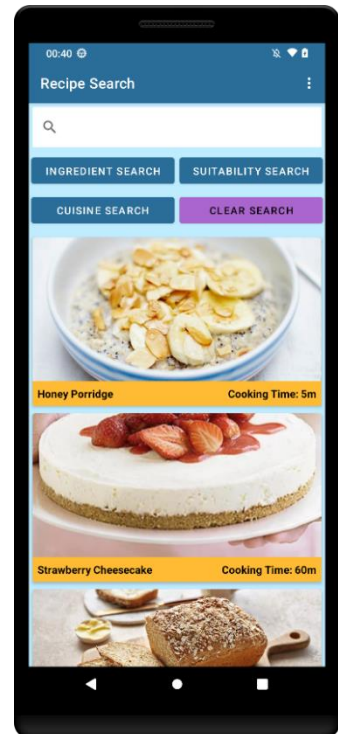


Figure 70 - Screenshot of Recipe Search screen.

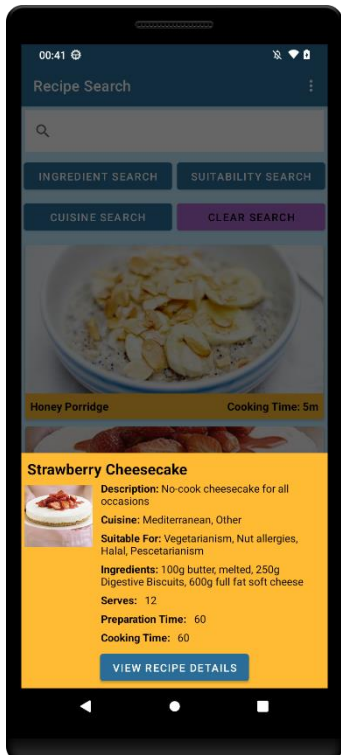


Figure 71 - Screenshot of Recipe Search Screen (bottom sheet dialog).

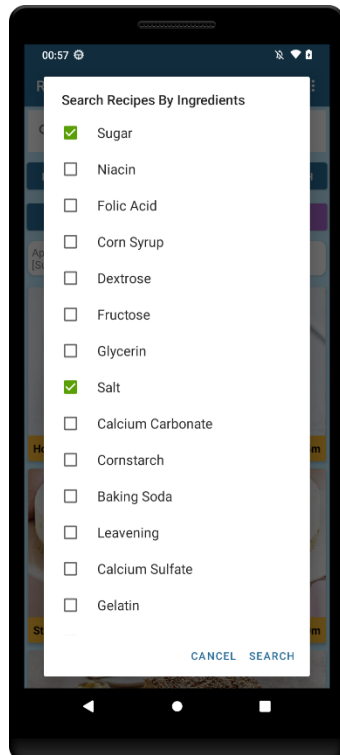


Figure 72 - Screenshot of Ingredients Search multi-select alert dialog.

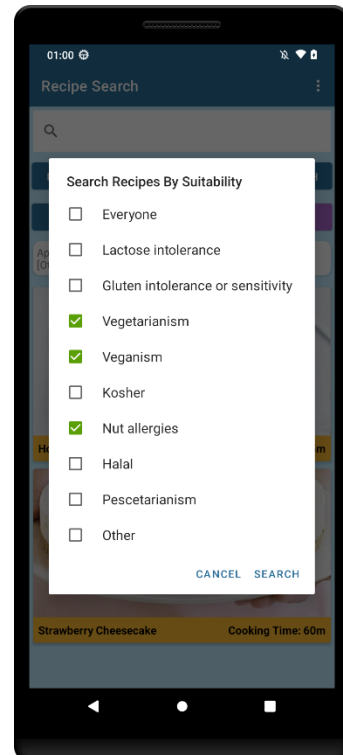


Figure 73 - Screenshot of Suitability Search multi-select alert dialog.



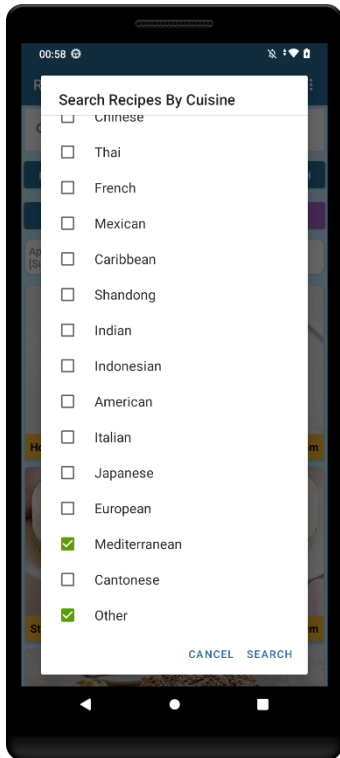


Figure 74 - Screenshot of Cuisine Search multi-select alert dialog.

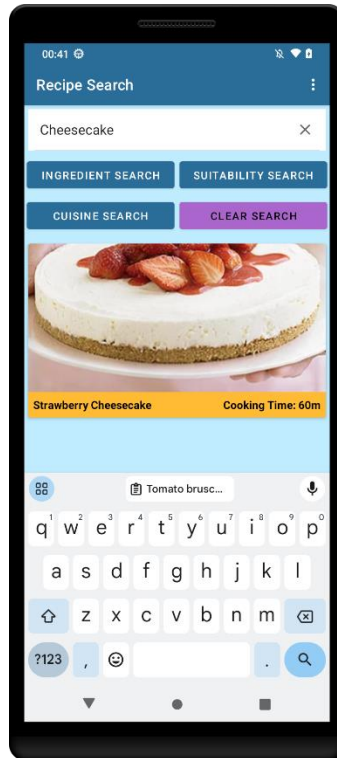


Figure 75 - Screenshot of Recipe Search by Name.

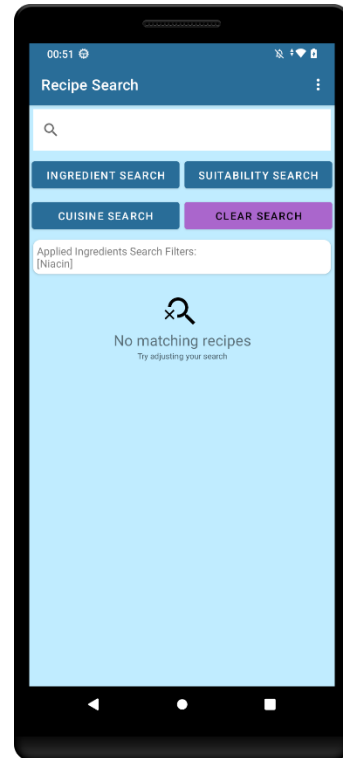


Figure 76 - Screenshot of Recipe Search (no results).

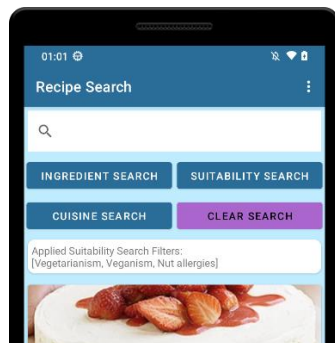


Figure 77 - Screenshot of Recipe Suitability Search filters applied.

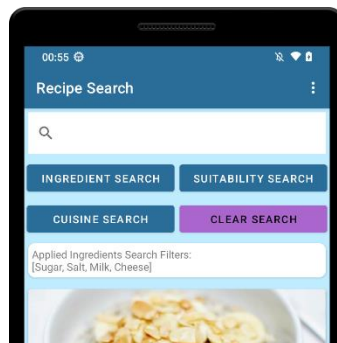


Figure 78 - Screenshot of Recipe Ingredients Search filters applied.

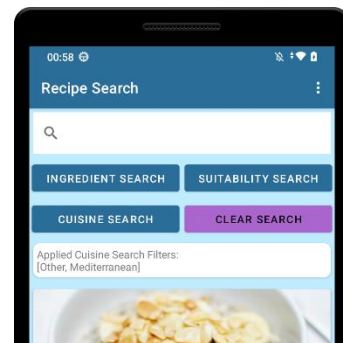


Figure 79 - Screenshot of Recipe Cuisine Search filters applied.

The Recipe Search screen is where users can search for recipes by Ingredients, Cuisine, Suitability or Name. The user can only view public recipes or owned recipes on this screen. The user can select a recipe from the results list which will open the recipe bottom sheet dialog which displays the recipe details and displays a button to the recipe details page. The user can search for recipes using the Search Bar at the top of the screen. The user can search for recipes by ingredients, cuisine, and suitability by selecting the corresponding search button which opens the corresponding multi-select alert dialog. The currently applied search will be displayed on screen to the user in a text view. If no recipes are returned by the search, then 'No Matching Recipes' will be displayed on screen. The currently applied search can be cleared using the Clear Search button.

## 2.5 Testing

### 2.5.1 Unit Testing

The project Unit Tests were created using the JUnit 5 test framework and are in the following project location: *app/src/test/java/com/example/foodplannerapp*. The following are the individual unit test cases and their results.

Test Case ID:	<b>TC01</b>
Test Case Title:	<b>Test Recipe Model</b>
Test Plan ID:	<b>1</b>
Test Type:	Unit
Test Case Description:	Test Recipe model creates recipe object & stores information correctly.
Project Test Location:	<i>java/com/example/foodplannerapp/models/RecipeTest.java (Test Package)</i>
Test Priority:	High
Pre-Conditions:	Recipe object is created using set values.
Executed Using:	IntelliJ Junit 5 Test Runner
Execution Steps:	<ol style="list-style-type: none"> <li>1. Confirm recipe name is as expected.</li> <li>2. Confirm cooking time is as expected.</li> <li>3. Confirm recipe preparation time is as expected.</li> <li>4. Confirm recipe servings is as expected.</li> <li>5. Confirm recipe suitability is as expected.</li> <li>6. Confirm recipe cuisine is as expected.</li> <li>7. Confirm recipe image URL is as expected.</li> <li>8. Confirm recipe URL is as expected.</li> <li>9. Confirm recipe description is as expected.</li> <li>10. Confirm recipe method is as expected.</li> <li>11. Confirm recipe ID is as expected.</li> <li>12. Confirm recipe user ID is as expected.</li> <li>13. Confirm recipe visibility is as expected.</li> </ol>
Post-condition:	System goes into wait state until next test is run.
Test Status:	Passed

Test Case ID:	<b>TC02</b>
Test Case Title:	<b>Test Ingredient Model</b>
Test Plan ID:	<b>1</b>
Test Type:	Unit
Test Case Description:	Test Ingredient model creates ingredient object and stores information correctly.
Project Test Location:	<i>java/com/example/foodplannerapp/models/IngredientTest.java (Test Package)</i>
Test Priority:	High
Executed Using:	IntelliJ Junit 5 Test Runner
Pre-Conditions:	Ingredient object is created using set values.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Confirm ingredient name is as expected.</li> <li>2. Confirm ingredient description is as expected.</li> </ol>
Post Conditions:	System goes into wait state until next test is run.
Test Status:	Passed

Test Case ID:	<b>TC03</b>
Test Case Title:	<b>Test Meal Model</b>
Test Plan ID:	<b>1</b>
Test Type:	Unit
Test Case Description:	Test Meal model creates meal object and stores information correctly.
Project Test Location:	<i>java/com/example/foodplannerapp/models/MealTest.java (Test Package)</i>
Test Priority:	High
Executed Using:	IntelliJ Junit 5 Test Runner
Pre-Conditions:	Meal object is created using set values.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Confirm meal name is as expected.</li> <li>2. Confirm cooking time is as expected.</li> <li>3. Confirm meal preparation time is as expected.</li> <li>4. Confirm meal servings is as expected.</li> <li>5. Confirm meal suitability is as expected.</li> <li>6. Confirm meal cuisine is as expected.</li> <li>7. Confirm meal image URL is as expected.</li> <li>8. Confirm meal URL is as expected.</li> <li>9. Confirm meal description is as expected.</li> <li>10. Confirm meal method is as expected.</li> <li>11. Confirm meal ID is as expected.</li> <li>12. Confirm meal user ID is as expected.</li> <li>13. Confirm meal date is as expected.</li> </ol>
Post Conditions:	System goes into wait state until next test is run.
Test Status:	Passed

Test Case ID:	<b>TC04</b>
Test Case Title:	<b>Test Meal Ingredient Model</b>
Test Plan ID:	<b>1</b>
Test Type:	Unit
Test Case Description:	Test Meal Ingredient model creates ingredient object and stores information correctly.
Project Test Location:	<i>java/com/example/foodplannerapp/models/MealIngredientTest.java (Test Package)</i>
Test Priority:	High
Executed Using:	IntelliJ Junit 5 Test Runner
Pre-Conditions:	Meal Ingredient object is created using set values.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Confirm meal ingredient name is as expected.</li> <li>2. Confirm meal ingredient purchase value is as expected.</li> </ol>
Post Conditions:	System goes into wait state.
Test Status:	Passed

Current scope: [all classes](#) | `com.example.firebasecrudapplication.models`

### Coverage Summary for Package: `com.example.firebasecrudapplication.models`

Package	Class, %	Method, %	Line, %
<code>com.example.firebasecrudapplication.models</code>	100% (6/6)	79.3% (46/58)	51.6% (81/157)

Class	Class, %	Method, %	Line, %
<code>Ingredient</code>	100% (1/1)	75% (3/4)	83.3% (5/6)
<code>Meal</code>	100% (2/2)	80.8% (21/26)	48.7% (38/78)
<code>MealIngredient</code>	100% (1/1)	80% (4/5)	85.7% (6/7)
<code>Recipe</code>	100% (2/2)	78.3% (18/23)	48.5% (32/66)

Figure 80 - Test Coverage Report for Models Package

### Package `com.example.foodplannerapp.models`

[all](#) > `com.example.foodplannerapp.models`

40 tests	0 failures	0 ignored	0.158s duration	<b>100%</b> successful
-------------	---------------	--------------	--------------------	---------------------------

#### Classes

Class	Tests	Failures	Ignored	Duration	Success rate
<code>IngredientTest</code>	2	0	0	0.090s	100%
<code>MealIngredientTest</code>	3	0	0	0.007s	100%
<code>MealTest</code>	19	0	0	0.031s	100%
<code>RecipeTest</code>	16	0	0	0.030s	100%

Generated by Gradle 7.5 at 29 Jul 2023, 14:33:20

Figure 81 - Test Summary Report for Models Package generated by Gradle.

Test Case ID:	<b>TC05</b>
Test Case Title:	<b>Test SortMeals Interface Class</b>
Test Plan ID:	-
Test Type:	Unit
Test Case Description:	Test SortMeals interface class successfully sorts Meal ArrayList items correctly by scheduled date (chronological order)
Project Test Location:	<code>java/com/example/foodplannerapp/interfaces/SortMealsTest.java</code> (Test Package)
Test Priority:	Low
Executed Using:	IntelliJ Junit 5 Test Runner
Pre-Conditions:	3 Meal objects are created using predefined values and added to ArrayList.
Test Data:	Meal A Date = 01/04/2023 Meal B Date = 01/04/2023 Meal C Date = 20/04/2023
Execution Steps:	<ol style="list-style-type: none"> <li>Assert Meal A date is same as Meal B date.</li> <li>Assert Meal B is sorted before Meal C.</li> <li>Assert Meal C is sorted after Meal A.</li> </ol>
Post Conditions:	System goes into wait state.
Test Status:	Passed

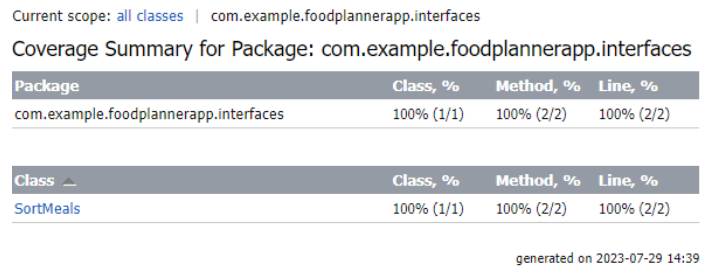


Figure 82 - Test Coverage Report for Interfaces Package

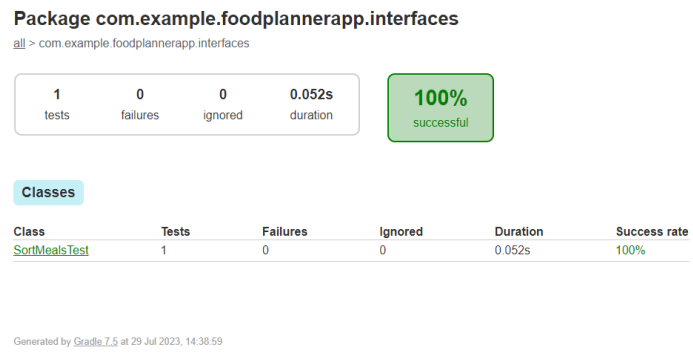


Figure 83 - Test Summary Report for Interfaces Package generated by Gradle.

Test Case ID:	<b>TC06</b>
Test Case Title:	<b>Test isValidPassword Method</b>
Test Plan ID:	-
Test Type:	Unit
Test Case Description:	Test ValidPasswordCheck isValidPassword utility method asserts password validity correctly.
Project Test Location:	<i>java/com/example/foodplannerapp/utilities/ValidPasswordCheckTest.java (Test Package)</i>
Test Priority:	Medium
Executed Using:	IntelliJ Junit 5 Test Runner
Pre-Conditions:	-
Execution Steps:	<ol style="list-style-type: none"> <li>1. Assert valid Password returns True value.</li> <li>2. Assert invalid Password returns False value.</li> <li>3. Assert invalid Password returns False value.</li> <li>4. Assert invalid Password returns False value.</li> <li>5. Assert invalid Password returns False value.</li> </ol>
Post Conditions:	System goes into wait state.
Test Status:	Passed

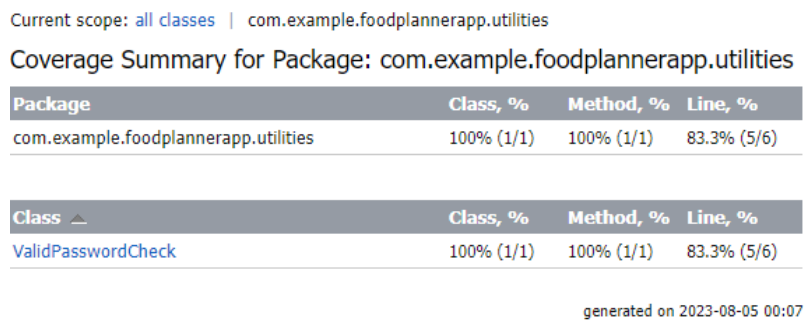


Figure 84 - Test Coverage Report for Utilities Package

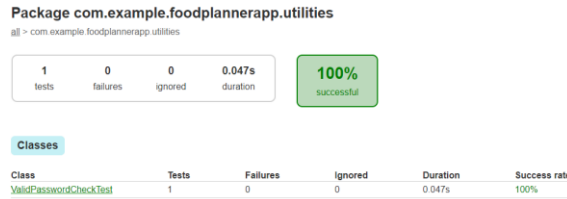


Figure 85 - Test Summary Report for isValidPasswordTest generated by Gradle.

## 2.5.2 Integration Testing

Integration tests were performed using Espresso, JUnit 4, and Manual Testing. The following are the individual integration test cases and their results.

Test Case ID:	<b>TC07</b>
Test Case Title:	Test Recipe Creation in Firebase Realtime Database
Test Plan ID:	-
Test Type:	Integration
Test Case Description:	Test that the Add Recipe Activity class successfully uses the Recipe model to create a recipe Json object in the Firebase database.
Project Test Location:	<i>java/com/example/foodplannerapp/AddRecipeIntegrationTest.java (androidTest Package)</i>
Test Priority:	High
Executed Using:	IntelliJ Espresso / JUnit 4 Test Runner & Manual Testing
Pre-Conditions:	Android AVD Device is available to run automated UI test.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Create Recipe Item</li> <li>2. View Recipe Details</li> <li>3. Confirm the recipe name is as expected.</li> <li>4. Confirm the cooking time is as expected.</li> <li>5. Confirm the recipe preparation time is as expected.</li> <li>6. Confirm the recipe servings is as expected.</li> <li>7. Confirm the recipe suitability is as expected.</li> <li>8. Confirm the recipe cuisine is as expected.</li> <li>9. Confirm the recipe image URL is as expected.</li> <li>10. Confirm the recipe URL is as expected.</li> <li>11. Confirm the recipe description is as expected.</li> <li>12. Confirm the recipe method is as expected.</li> <li>13. Assert the expected recipe ID (Manual Firebase Check)</li> <li>14. Assert the expected recipe user ID (Manual Firebase Check)</li> <li>15. Assert the expected recipe visibility (Manual Firebase Check)</li> <li>16. Delete Recipe</li> </ol>
Post Conditions:	System goes into wait state.
Test Status:	Passed

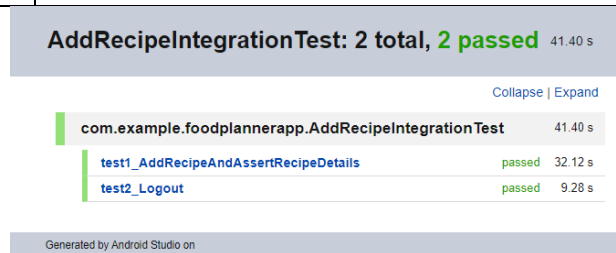


Figure 86 - Add Recipe Integration Test

Test Case ID:	<b>TC08</b>
Test Case Title:	Test Meal Creation in Firebase Realtime Database
Test Plan ID:	-
Test Type:	Integration
Test Case Description:	Test that the View Recipe Activity class successfully uses the Meal model to create a meal Json object and the MealIngredient mode to create meal ingredient object(s) in the Firebase database using a Recipe object's data.
Project Test Location:	<i>java/com/example/foodplannerapp/RecipeToMealItemIntegrationTest.java (androidTest Package)</i>
Test Priority:	Medium
Executed Using:	IntelliJ Espresso / JUnit 4 Test Runner & Manual Testing
Pre-Conditions:	Recipe item is created which is used to create Meal Plan item
Execution Steps:	<ol style="list-style-type: none"> <li>1. Create Recipe Item</li> <li>2. View Recipe Details</li> <li>3. Click Add to Meal Plan Button</li> <li>4. Select date from calendar date picker.</li> <li>5. Confirm the expected meal name is as expected.</li> <li>6. Confirm the cooking time is as expected.</li> <li>7. Confirm the meal preparation time is as expected.</li> <li>8. Confirm the meal servings is as expected.</li> <li>9. Confirm the meal suitability is as expected.</li> <li>10. Confirm the meal cuisine is as expected.</li> <li>11. Confirm the meal image URL is as expected.</li> <li>12. Confirm the meal URL is as expected.</li> <li>13. Confirm the meal description is as expected.</li> <li>14. Confirm the meal method is as expected.</li> <li>15. Confirm meal ID is as expected (Manual Firebase Check)</li> <li>16. Confirm meal user ID is as expected (Manual Firebase Check)</li> <li>17. Confirm meal date is as expected (Manual Firebase Check)</li> <li>18. Confirm meal ingredients is as expected (Manual Firebase Check)</li> <li>19. Delete Meal</li> <li>20. Delete Recipe</li> </ol>
Post Conditions:	System goes into wait state.
Test Status:	Passed

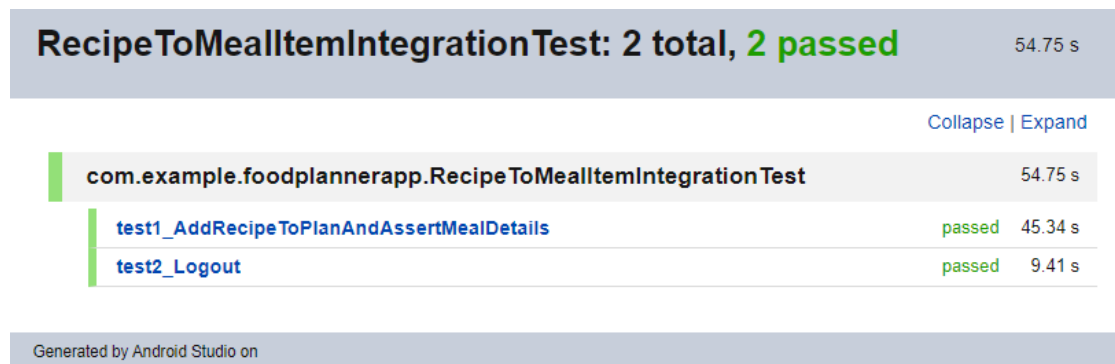


Figure 87 - Recipe to Meal Integration Test Results

Test Case ID:	<b>TC09</b>
Test Case Title:	Test MainActivity Class, Recipe Model and RecipeRVAdapter Class Integration.
Test Plan ID:	-
Test Type:	Integration
Test Case Description:	Test MainActivity Recipe RecyclerView is populated with recipes retrieved from Firebase using the Recipe model and RecipeRVAdapter class.
Project Test Location:	N/A
Test Priority:	Medium
Executed Using:	Manual Testing
Pre-Conditions:	Recipe item is created which is used to create Meal Plan item
Steps:	1. Assert Recipes are displayed in Main RecyclerView.
Post Conditions:	System goes into wait state.
Test Status:	Passed

Test Case ID:	<b>TC10</b>
Test Case Title:	Test IngredientsScannerActivity Class, Ingredient Model and IngredientsScannerRVAdapater Class Integration.
Test Plan ID:	-
Test Type:	Integration
Test Case Description:	Test IngredientsScannerActivity ingredient RecyclerView is populated with ingredients retrieved from Firebase using Ingredient model and IngredientsScannerRVAdapater class.
Project Test Location:	N/A
Test Priority:	Medium
Executed Using:	Manual Testing
Pre-Conditions:	Recipe item is created which is used to create Meal Plan item
Steps:	1. Scan image containing ingredients using ingredients scanner. 2. Assert matching ingredients are listed in ingredients RecyclerView.
Post Conditions:	System goes into wait state.
Test Status:	Passed

Test Case ID:	<b>TC11</b>
Test Case Title:	Test RecipeSearchActivity class, Recipe model and RecipeRVAdapter class Integration
Test Plan ID:	-
Test Case Description:	Test RecipeSearch recipe RecyclerView is populated with recipes retrieved from Firebase using the Recipe model and RecipeRVAdapter class.
Test Type:	Integration
Project Test Location:	N/A
Test Priority:	Medium
Executed Using:	Manual Testing
Pre-Conditions:	Recipe item is created which is used to create Meal Plan item
Execution Steps:	1. Assert Recipes are displayed in RecipeSearchActivity RecyclerView.
Post Conditions:	System goes into wait state.
Test Status:	Passed



### 2.5.3 System Testing

System tests were performed using Espresso, JUnit 4, and Manual Testing. The following are the individual system test cases and their results.

Test Case ID:	<b>TC12</b>
Test Case Title:	Test Firebase Authentication User Registration
Test Plan ID:	<b>3 (User Management)</b>
Test Type:	System
Test Case Description:	Test Registration Activity creates a new Firebase Authentication user.
Project Test Location:	<i>java/com/example/foodplannerapp/FirebaseAuthenticationTest.java (androidTest Package)</i>
Test Priority:	High
Executed Using:	IntelliJ Espresso / JUnit 4 Test Runner & Manual Testing
Pre-Conditions:	Firebase Authentication service is available
Execution Steps:	<ol style="list-style-type: none"> <li>1. Register new user.</li> <li>2. Assert correct email is printed on edit account screen.</li> <li>3. Assert new user is created with correct details. (Manual Firebase Check)</li> <li>4. Log out.</li> </ol>
Post Conditions:	System goes into wait state until next test in queue is run.
Test Status:	Passed

Test Case ID:	<b>TC13</b>
Test Case Title:	Test Firebase Authentication User Login
Test Plan ID:	<b>3 (User Management)</b>
Test Type:	System
Test Case Description:	Test Login Activity authenticates Firebase Authentication user.
Project Test Location:	<i>java/com/example/foodplannerapp/FirebaseAuthenticationTest.java (androidTest Package)</i>
Test Priority:	High
Executed Using:	IntelliJ Espresso / JUnit 4 Test Runner & Manual Testing
Pre-Conditions:	Firebase Authentication service is available
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. Assert correct email is printed on edit account screen.</li> <li>3. Assert user last login in Firebase Authentication (Manual Firebase Check)</li> <li>4. Log out.</li> </ol>
Post Conditions:	System goes into wait state until next test in queue is run.
Test Status:	Passed

Test Case ID:	<b>TC14</b>
Test Case Title:	Test Firebase Authentication User Deletion
Test Plan ID:	<b>3 (User Management)</b>
Test Type:	System
Test Case Description:	Test Edit Account Activity deletes Firebase Authentication user.
Project Test Location:	<i>java/com/example/foodplannerapp/FirebaseAuthenticationIntegrationTest.java (androidTest Package)</i>
Test Priority:	High
Executed Using:	IntelliJ Espresso / JUnit 4 Test Runner & Manual Testing

Pre-Conditions:	Firestore Authentication service is available
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. Delete new user on edit account screen.</li> <li>3. Assert user is deleted (Manual Firestore check)</li> <li>4. Log out.</li> <li>5. Assert user cannot log in.</li> </ol>
Post Conditions:	System goes into wait state.
Test Status:	Passed

**FirestoreAuthenticationTest: 3 total, 3 passed** 43.23 s

[Collapse](#) | [Expand](#)

<b>com.example.foodplannerapp.FirestoreAuthenticationTest</b>	43.23 s
test1_newUserTest	passed 18.26 s
test2_loginUserTest	passed 11.99 s
test3_deleteUserTest	passed 12.98 s

Figure 88 - FirestoreAuthenticationIntegrationTest Test Result

Test Case ID:	<b>TC15</b>
Test Case Title:	Test Firestore Authentication User Update
Test Type:	System
Test Plan ID:	<b>3 (User Management)</b>
Test Case Description:	Test Edit Account Activity deletes Firestore Authentication user.
Project Test Location:	N/A
Test Priority:	High
Executed Using:	Manual Testing
Pre-Conditions:	Firestore Authentication service is available.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. Navigate to the Edit Account Screen.</li> <li>3. Update the user email.</li> <li>4. Assert new email is displayed on screen.</li> <li>5. Update the user password.</li> <li>6. Log out.</li> <li>7. Log in using updated credentials.</li> <li>8. Assert user is logged in successfully.</li> </ol>
Post Conditions:	N/A
Test Status:	Passed

Test Case ID:	<b>TC16</b>
Test Case Title:	Test scanning ingredients using ingredients scanner
Test Plan ID:	-
Test Type:	System
Test Case Description:	Test the ingredients scanner successfully returns matching ingredients in scanned image
Project Test Location:	N/A
Test Priority:	High
Executed Using:	Manual Testing
Pre-Conditions:	The user has granted permission to the app to their camera and gallery.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. Navigate to the Ingredients Scanner Screen.</li> </ol>

	<ol style="list-style-type: none"> <li>3. Click the Select Image button.</li> <li>4. Select an image containing ingredients text.</li> <li>5. Click the Scan Ingredients button.</li> <li>6. Assert that any matching ingredients are listed on screen with their name and description.</li> <li>7. Click the Capture Image button and take a photo of ingredients text.</li> <li>8. Assert that any matching ingredients are listed on screen with their name and description.</li> </ol>
Post Conditions:	N/A
Test Status:	Passed

Test Case ID:	<b>TC17</b>
Test Case Title:	Test searching for recipes using recipes search
Test Plan ID:	-
Test Type:	System
Test Case Description:	Test the Recipe Search successfully returns recipes matching search criteria
Project Test Location:	N/A
Test Priority:	High
Executed Using:	Manual Testing
Pre-Conditions:	There are existing recipes available in the database.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. Navigate to the Recipe Search Screen.</li> <li>3. Assert public or user owned recipes are listed on screen.</li> <li>4. Click the Ingredients Search button.</li> <li>5. Select ingredients in the Search Ingredients modal and click Search.</li> <li>6. Assert recipes with matching ingredients are listed on screen.</li> <li>7. Click the Cuisine Search button.</li> <li>8. Select cuisine in the Search Cuisine modal and click Search.</li> <li>9. Assert recipes with matching cuisine are listed on screen.</li> <li>10. Click the Suitability Search button.</li> <li>11. Select suitability in the Search Suitability modal and click Search.</li> <li>12. Assert recipes with matching suitability are listed on screen.</li> <li>13. Click the Clear Search button.</li> <li>14. Assert that last search is cleared.</li> <li>15. Using the search bar search for a recipe by name.</li> <li>16. Assert that recipes with a matching name are returned.</li> </ol>
Post Conditions:	N/A
Test Status:	Passed

Test Case ID:	<b>TC18</b>
Test Case Title:	Test Recipe CRUD functionality
Test Plan ID:	-
Test Type:	System
Test Case Description:	Test that a recipe can be created, read, updated, and deleted.
Project Test Location:	N/A

Test Priority:	High
Executed Using:	Manual Testing
Pre-Conditions:	There are existing recipes available in the database.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. Click new recipe button.</li> <li>3. Add new recipe details and click Add Recipe button.</li> <li>4. Assert user is redirected to My Recipes screen and new recipe is listed.</li> <li>5. Click the recipe in the list and click the Edit Recipe button.</li> <li>6. One the recipe edit screen change the recipe name and then click the Update Recipe button.</li> <li>7. Assert user is redirected to the My Recipes screen and recipe displays updated name.</li> <li>8. Click the recipe button and click the View Recipe button.</li> <li>9. Assert details on recipe details screen are correct.</li> <li>10. Click the Delete Recipe button and click OK in the confirmation modal.</li> <li>11. Assert user is redirected to My Recipes screen and recipe is no longer listed on screen.</li> </ol>
Post Conditions:	N/A
Test Status:	Passed

Test Case ID:	<b>TC19</b>
Test Case Title:	Test adding and removing meals from plan
Test Plan ID:	-
Test Type:	System
Test Case Description:	Test that a meal can be added and removed from a user's meal plan.
Project Test Location:	N/A
Test Priority:	Medium
Executed Using:	Manual Testing
Pre-Conditions:	There are existing recipes available in the database.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. On the My Recipes screen click a recipe and then click the View Details button.</li> <li>3. On the recipe details page click the Add Recipe to Meal Plan button.</li> <li>4. Select a date from the Calander picker modal.</li> <li>5. Assert user is redirected to the Meal Planner screen and recipe is added to meal plan.</li> <li>6. Assert that the meals in the list are sorted by date (chronological order).</li> <li>7. Click the Meal from the List and then click the View Details and Shopping List button.</li> <li>8. Assert meal details are correct.</li> <li>9. Navigate back the previous screen.</li> <li>10.</li> </ol>
Post Condition:	N/A
Test Status:	Passed

Test Case ID:	<b>TC20</b>
Test Case Title:	Test updating ingredients in shopping list
Test Plan ID:	-
Test Type:	System
Test Case Description:	Test that an ingredient in a meal shopping list can be marked as purchased or not
Project Test Location:	N/A
Test Priority:	Medium
Executed Using:	Manual Testing
Pre-Conditions:	There are existing meals in the meal plan.
Execution Steps:	<ol style="list-style-type: none"> <li>1. Log in.</li> <li>2. On the Meal Planner page click a Meal from the List and then click the View Details and Shopping List button.</li> <li>3. Assert that there are ingredients items listed in the Shopping List section of the Meal details page.</li> <li>4. Check an unchecked ingredient checkbox in the shopping list.</li> <li>5. Navigate back to the Meal Planner screen.</li> <li>6. Click the same Meal from the List and then click the View Details and Shopping List button.</li> <li>7. Assert that the ingredient is still marked as checked in the Shopping List.</li> </ol>
Post Conditions:	N/A
Test Status:	Passed

#### 2.5.4 End-To-End Functionality Testing

As part of the application functionality testing. Manual End-To-End Testing was completed to ensure that the completed application met all functionality requirements and acceptance criteria.

Test Plan ID:	<b>4</b>
Test Plan Title:	Complete End-To-End functionality testing on completed application.
Test Type:	Functionality (End-To-End) / Acceptance Testing
Test Plan Description:	Test that the completed application meets all functionality requirements.
Project Test Location:	N/A
Test Priority:	High
Executed Using:	Manual Testing
Pre-condition:	Application development has completed.
Test Cases	<ol style="list-style-type: none"> <li>1. Execute Test Case IDs TC12, TC13, TC14, TC15, TC16, TC17, TC18, TC19, TC20</li> </ol>
Test Execution Status:	Passed
Post-condition:	N/A

### Android Studio IDE Lint Tool

A code scanning tool named ‘Lint’ is provided by the Android Studio IDE. This tool automatically runs lint checks for source files in an Android project to identify code structural quality problems and to detect potential bugs. The lint tool offers optimization improvement suggestions for improving an application code’s correctness, security, performance, accessibility, usability, and internalisation. Code structural issues can create issues with an applications reliability and efficiency which effects the maintainability of the application. (Google For Developers, 2023)

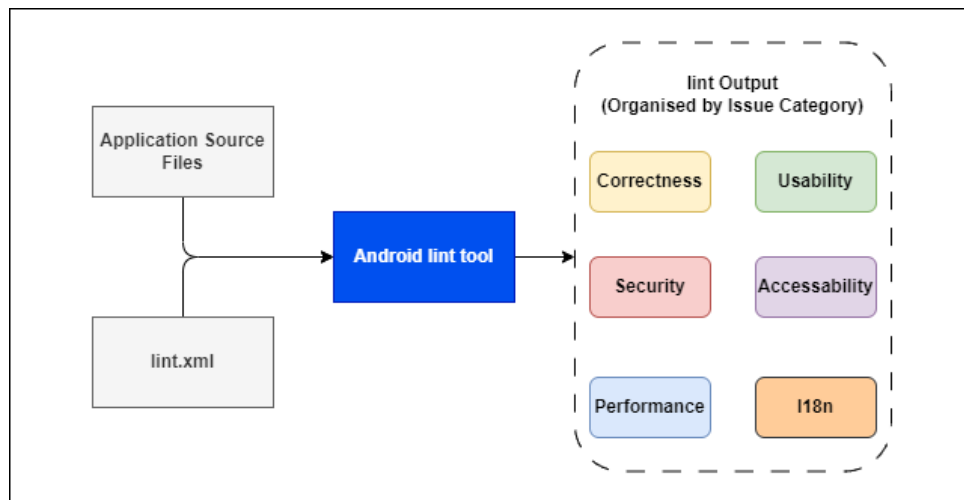


Figure 89 – Android Studio IDE Lint Tool Code Scan Workflow (Google For Developers, 2023)

One of the projects goals in improving the correctness, usability, security, accessibility, performance, and internalisation of the application source code was to attempt to resolve as many of the warnings reported by the Android lint tool as possible. To do this, a Git branch of the code was checked out and an attempt was made to resolve all reported lint issue. The following are the results of this effort.

#### Project Lint Scan Location One:

##### *app/src/main/java/com/example/foodplannerapp* (Main Java Package)

Before attempting to resolve any lint errors within the project Java package, the lint scan reported 322 Warnings and 17 Weak Warnings. Of these 399 reported warnings, all 339 (100%) warnings were then manually resolved. The first of the following two figures displays the lint scan before the warning were resolved. The second of the two figures display the Android Studio IDE notification result from the lint scan advising of no issues found in the *app/src/main/java/com/example/foodplannerapp* directory:

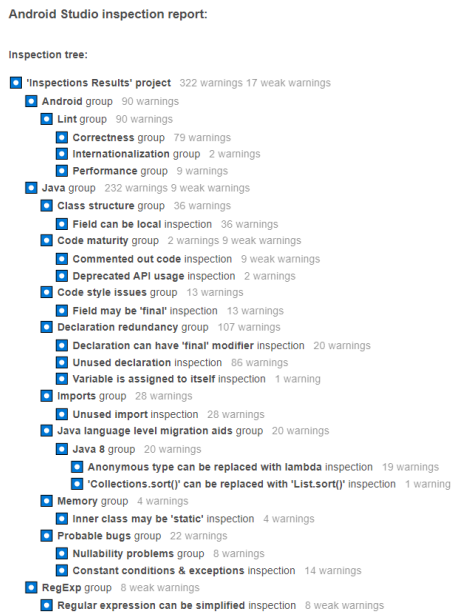


Figure 90 - Lint Report for Java Package before warnings were manually resolved (322 Warnings & 17 Weak Warnings Reported).

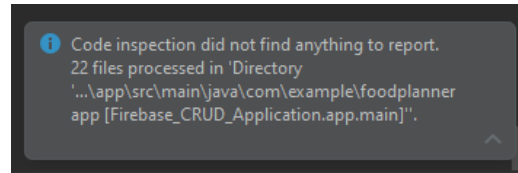


Figure 91 - Java Package Lint Scan notification stating that no problems were identified.

## Project Lint Scan Location Two:

### app/src/main/java/com/example/foodplannerapp (Main Resource Package)

Before attempting to resolve any lint errors within the project Resource package, the lint scan reported 183 Warnings and 12 Typos. Of these 194 reported warnings, 192 (97%) warnings were then manually resolved. The first of the following two figures displays the lint scan before the warnings were resolved. The second of the two figures display the lint scan after the warnings were resolved:



Figure 92 - Lint Report for Resource Package before warnings were manually resolved (183 Warnings / 12 Typos Reported).

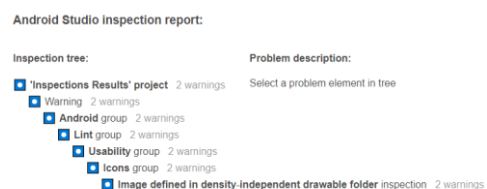


Figure 93 - Lint Report for Resource Package after warnings were manually resolved (2 Warnings and 0 Types reported)

## Android Profiler (Performance Analyse)

Android Studio IDE offers the following three profiling tools for measuring an applications performance: CPU Profiler, Memory Profiler, and Energy Profiler. The CPU profiler is used to identify runtime performance issues, the Memory Profiler is used to track memory allocations, and the Energy profiler is for tracking the application energy usage which is important for analysing the applications drainage effect on a device’s battery. (Google For Developers, 2023) The following image was captured using the Profiler tool in Android Studio. The profiling session was completed using complete data and does not represent the application’s performance in production as the profiling session was executed locally using a physical Android device connected to Android Studio running a development branch of the application. In the profiling session each of the application activities were launched to capture each of the activities’ CPU, Memory, and Energy usage:

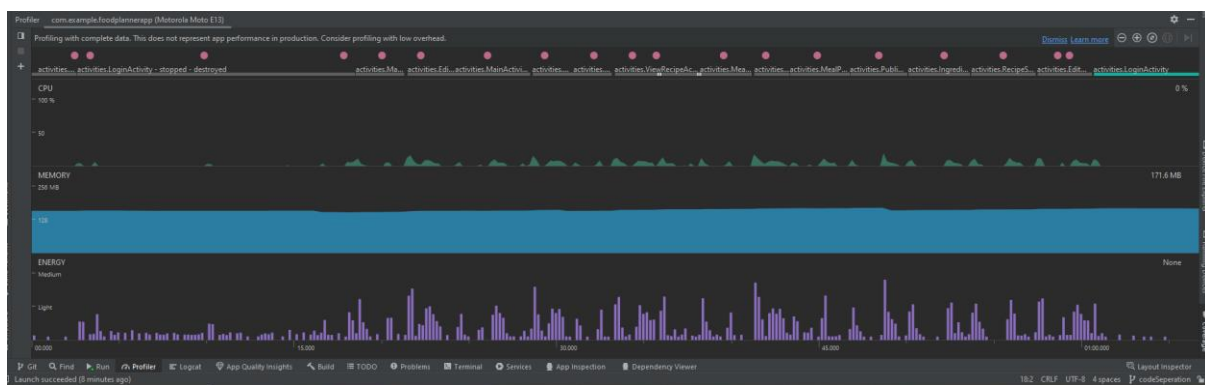


Figure 94 - Android Studio Performance Profiler Session in which all the application activity screen were opened and closed.

The following is a breakdown of the above performance profile session by Activity:

Activity	CPU (Highest Usage)	Memory (Highest Usage)	Energy (Grade)
LoginActivity (Login Screen)	8%	164MB	Light
MainActivity (My Recipes Screen)	12%	158MB	Medium
EditRecipeActivity (Edit Recipe Screen)	17%	163MB	Medium
AddRecipeActivity (Add Recipe Screen)	16%	166MB	Medium
ViewRecipeActivity (View Recipe Screen)	14%	168MB	Medium
MealPlanActivity (Meal Planner Screen)	13%	170MB	Medium
ViewMealPlanActivity (Meal Details Screen)	13%	172MB	Medium
PublicRecipesActivity (Public Recipes Screen)	20%	174MB	Medium
IngredientsScannerActivity (Ingredients Scanner Screen)	10%	166MB	Medium
RecipesSearchActivity (Recipe Search Screen)	13%	169MB	Medium
EditAccountActivity (Edit Account Screen)	13%	171MB	Medium
<b>Average:</b>	<b>14%</b>	<b>167MB</b>	-


Opening and closing each of the application activities (screens) uses 14% CPU on average and does not exceed 20% CPU usage. The activities have a highest memory usage of 167MB on average and none of the activities reported an energy usage grade of Heavy. All of which indicate good CPU, memory, and energy default usage.



## Testing Coverage

Below is a screenshot of the test coverage summary report for the interfaces, utilities, and model package classes. There is an average class coverage of 100%, an average method coverage of 93.1% and an average line coverage of 78.2 % for these three packages. The unit testing for the activities, adapters, and fragment classes were completed manually during the development of the project.

Coverage Breakdown

Package 	Class, %	Method, %	Line, %
com.example.foodplannerapp	0% (0/1)	0% (0/2)	0% (0/2)
com.example.foodplannerapp.interfaces	100% (1/1)	100% (2/2)	100% (2/2)
com.example.foodplannerapp.models	100% (6/6)	79.3% (46/58)	51.6% (81/157)
com.example.foodplannerapp.utilities	100% (1/1)	100% (1/1)	83.3% (5/6)

generated on 2023-08-05 00:26

Figure 95 - Unit Test Coverage Summary Report filtered by Models, Interface, and Utilities Packages

JUnit 5 was used to execute the above Unit Tests. A combination of automated UI Espresso Tests, JUnit 4 and Manual Testing was used to execute the integration and system testing in the project. End To End Manual testing was completed at the end of the projects development to ensure that all integrated system features were working successfully together and to ensure that all functional requirements were met. All functional requirements were confirmed as working during the End-to-End testing in the Acceptance Testing stage.

### 3 Conclusions

The Interfaces, Utilities, And Model package classes have an extensive unit test coverage. There is an average class test coverage of 100%, an average method test coverage of 93.1% and an average line test coverage of 78.2 % for these three packages. One limitation of creating unit tests for all packages within the application is the applications current architecture design pattern of MVC. With more time and resources, a Model-View-ViewModel (MVVM) or a Model-View-Presenter (MVP) architectural pattern design would be applied to the application in which the UI logic and Business logic would be separated completely to improve the testability of the application. This limitation was overcome through extensive unit, integration and system testing of the application using automated and manual testing. End-To-End functionality testing was completed at the end of the development of the project to ensure that the application met all functional requirements of the application and all tests passed.

During the development of the project a high priority was placed on minimising as many Android Lint tool warnings as possible. This attempt was a success as 339 (100%) warnings were manually resolved within the applications main Java package and 192 (97%) of warnings were manually resolved within the application Resource folder. This was to ensure the code structure quality of the application in areas such as correctness, security, performance, accessibility, usability, and internalisation. The results of the performance profiling of the application indicated good CPU, memory, and energy default usage.

The project was a success as all six functional requirements within the scope of the project were developed and implemented into a single working Android Mobile Application resulting in successful project completion. The following six functional requirements have been successfully implemented into the application; A User Management feature allowing users to create, manage and log in to their user account. An Ingredients Scanner feature allowing users to scan images for ingredients text and receive the ingredients meaning. A Recipe Manager feature allowing users to create, read, update, and delete recipes. A Recipe Search feature allowing users to search for public or owned recipes by ingredients, cuisine, suitability, or name. A Meal Planner feature allowing users to add or remove recipes to and from their meal plan. A Shopping List feature allowing users to manage a shopping list containing required ingredients for each of their scheduled meals in their meal plan. The application was successful in its aims to develop the FoodPlannerApp using innovative technologies such as Machine Learning models from Google's ML Kit Text Recognition API, Android Mobile Application Development and Backend Cloud computing services such as Firebase.

## 4 Further Development or Research

One issue with Android's MVC implementation is the violation of the main principle of the MVC pattern, the Single Responsibility Principle. Regardless, MVC is the architecture pattern that the FoodPlannerApp's architecture most resembles. The main goal in the development of the FoodPlannerApp was to produce a working application which delivers all functional requirements which was achieved. With more time and resources, the next evolution of the FoodPlannerApp's architecture design would be to separate the applications working logic into either a Model-View-ViewModel (MVVM) or a Model-View-Presenter (MVP) architectural pattern design.

Two drawbacks of using the Firebase Realtime Database are that relationships between the data cannot be created within the database and it can be difficult to query data from the database as the data is stored as a single JSON object. With more time and resources, the next step in the development of the FoodPlannerApp would be to possibly integrate a relational database, or to instead implement Firebase Firestore Database instead.

In the original proposal for this project, it was mentioned that the Recipe Search Engine feature may possibly use web scraping/crawling functionality to extract recipes containing specified ingredients from Google's Search Engine Results Pages (SERP). In the Requirements Specification creation stage of the project, it was stated that the Recipe Search feature functionality would either be achieved through web scraping or through querying user recipes stored in the application database. During the development of the Recipe Search Engine feature, it was decided that the search feature would query user created recipes from the database. With further development and research, the next evolution of the Recipe Search Engine would involve searching for recipes containing certain ingredients through extracting recipes from Google's SERP.

With further development and research, a Continuous Integration and Continuous delivery pipeline would be developed. This would involve setting up automated test runs/lint scans whenever a change is pushed to the project GitHub repository.

With further time and resources, the application functionality would be expanded upon by: Adding a sort to the ingredients listed in the ingredients multi-select search modal on the Recipe Search page so that the ingredients are sorted alphabetically. Adding a search bar to the ingredients multi-select search modal on the Recipe Search page so that users can search for an ingredient to select from within the modal. Adding image upload functionality to the Recipe Add/Edit pages so that users can upload their own images of recipes. Adding a forgotten password functionality to the login page. Adding an email verification system so that user must verify their email address after registering for a new account.

## 5 References

Adilovic, A., 2021. *A Guide to Choosing the Best Architecture Pattern for Android Apps*. [Online] Available at: <https://www.scalablepath.com/android/android-apps-architecture> [Accessed 3 August 2023].

Bechtold, S. et al., no date. *JUnit 5 User Guide*. [Online] Available at: <https://junit.org/junit5/docs/current/user-guide/> [Accessed 5 August 2023].

Bord Bia, 2020. *What Ireland Ate Last Night Report 2020*. [Online] Available at: <https://www.bordbia.ie/globalassets/bordbia2020/industry/insights/consumer-insights/what-ireland-ate-last-night-february-2020.pdf> [Accessed 23 July 2023].

Crowe, M., O'Sullivan, M., Casseti, O. & O'Sullivan, A., 2019. Estimation and consumption pattern of free sugar intake in 3-year-old. *European Journal of Nutrition*, 59(5), p. 2065–2074.

Dasgupta, K. et al., 2023. Associations of free sugars from solid and liquid sources with cardiovascular disease: a retrospective cohort analysis. *BMC Public Health*, 23(1).

EPA, 2023. *Food Waste Statistics*. [Online] Available at: <https://www.epa.ie/our-services/monitoring--assessment/waste/national-waste-statistics/food/> [Accessed 5 August 2023].

GitHub, no date. *About Git*. [Online] Available at: <https://docs.github.com/en/get-started/using-git/about-git> [Accessed 1 August 2023].

Google For Developers, 2023. *Firebase Authentication*. [Online] Available at: <https://firebase.google.com/docs/auth> [Accessed 1 August 2023].

Google for Developers, 2023. *Firebase Realtime Database*. [Online] Available at: <https://firebase.google.com/docs/database> [Accessed 1 August 2023].

Google For Developers, 2023. *Improve your code with lint checks*. [Online] Available at: <https://developer.android.com/studio/write/lint> [Accessed 1 August 2023].

Google For Developers, 2023. *Inspect your app's memory usage with Memory Profiler*. [Online] Available at: <https://developer.android.com/studio/profile/memory-profiler> [Accessed 1 August 2023].

Google For Developers, 2023. *Meet Android Studio*. [Online] Available at: <https://developer.android.com/studio/intro> [Accessed 1 August 2023].

Google For Developers, 2023. *ML Kit*. [Online]  
Available at: <https://developers.google.com/ml-kit/guides>  
[Accessed 1 August 2023].

Google For Developers, 2023. *Text recognition v2*. [Online]  
Available at: <https://developers.google.com/ml-kit/vision/text-recognition/v2>  
[Accessed 2 August 2023].

Google for Developers, 2023. *The activity lifecycle*. [Online]  
Available at: <https://developer.android.com/guide/components/activities/activity-lifecycle>  
[Accessed 3 August 2023].

JUnit, 2021. *JUnit 4 About*. [Online]  
Available at: <https://junit.org/junit4/>  
[Accessed 5 August 2023].

Oracle, no date. *The Java™ Tutorials*. [Online]  
Available at: <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>  
[Accessed 1 August 2023].



# National College of Ireland

## Project Proposal Food Planner App 21/12/2022

Bachelor of Science (Honours) in Computing

Software Development

2022/2023

Ruby Lennon

X19128355

x19128355@student.ncirl.ie

## Contents

1.0	Objectives .....	87
2.0	Background .....	87
3.0	State of the Art.....	88
4.0	Technical Approach.....	88
5.0	Technical Details .....	89
6.0	Special Resources Required .....	90
7.0	Project Plan .....	91
8.0	Testing.....	93
	References .....	94

## 1.0 Objectives

This project sets out to achieve creating a 'Food Planner' application. In this project I will aim to develop an Android mobile application that consists of a food ingredients scanner, a recipe search engine, a meal planner, a recipe manager, and a food shopping list generator. My aim in creating this application is to help lessen the impact of food waste on the environment, help users to save money by reducing their food waste, and to help users make better informed decisions on what food they choose to purchase. This mobile application is for anyone who wants to organise their eating schedule, make healthier food choices, and reduce food waste by only buying what they need to save money and lessen their impact on the environment. Using the mobile application users will be able to upload an image of an ingredients list on any food product. Through using OCR (Optical Character Recognition) technology, the application will then extract the ingredients text from the uploaded image. The application will then generate a report of all the listed ingredients and their meaning. Users will also be able to search for food recipes by ingredients, total preparation time, cuisine and or meal type etc. I currently envision the application doing this by executing a Google search using those search queries and crawling through the 'Recipes' section in Google SERP (Search Engine Results Page). Through web scraping, the key information in those recipe pages would then be extracted, formatted, and returned to the user in the Food Planner App. Users will also be able to manage their saved recipes, create meal plans and create food shopping lists.

## 2.0 Background

The first problem that this project attempts to solve is food waste and its impact on the environment and on household expenditure. According to the EPA (The Environmental Protection Agency) Irish households threw away an estimated 241,000 tonnes of food in 2020. Food waste costs the average Irish household about €60 per month or €700 per year. It is estimated that food waste generates about 8% to 10% of global greenhouse gas emissions. (EPA, 2022) The 'Food Planner App' aims to solve this by helping users to reduce their food waste and carbon footprint by searching for recipes containing ingredients they already own. Helping users to reduce their food waste should also help them to save money. The second problem that this project attempts to solve is confusing obscure food product ingredients. When it comes to selecting a food product in a shop, it can be difficult to understand the ingredients listed on food labels and whether they are good for you or not. Manufacturers can avoid listing sugar and fats as the first ingredient by using multiple forms of sugar and fat that go by other names. The 'Food Planner App' attempts to solve this by allowing users to upload a photo of a food products ingredients list and to receive a report of the ingredients and their general meaning. The third problem that this project attempts to solve is unhealthy food habits because of hectic schedules. The 'Food Planner App' attempts to solve this by allowing the user to organise their recipes, create a meal plan and generate a food shopping list. My solution to these problems is to develop an all-encompassing food planner application. Through clear goal setting, structured project planning, in depth research and supervisor guidance I believe I will be able to meet the project objectives.



### 3.0 State of the Art

A few examples of food ingredients scanners include 'Yuka - Food & Cosmetic Scan' (Yuka, 2022) and 'Infood app' (Infood Team, 2022) – both of which are mobile applications which allow the user to scan food product barcodes to find out more information regarding the products ingredients. One difference between these applications and the 'Food Planner App' is that the Food Planner App will use OCR technology to read the food ingredients from the text in an image to generate the report instead. An example of an application that allows a user to search for recipes by ingredients is 'Supercook' (SuperCook, 2022). One difference between 'Supercook' and the 'Food Planner App' is that 'Supercook' does not have an ingredients scanner functionality. An example of an application that generates a meal plan is 'Eat This Much' (Eat This Much Inc, 2022) – this application allows users to create and edit a meal plan and automatically generate a shopping list. The difference between 'Eat This Much' and the 'Food Planner App' is that 'Eat This Much' does not include an ingredients scanner and does not include a recipe search engine that allows the user to search for recipes containing owned ingredients. Two other applications to note that include one or more, but not all, of the 'Food Planner App' features are 'Whisk' (Whisk, 2022) and 'The CookBook App' (CookBook Co. Pty Ltd, 2022) - both of which allow users to share recipes, create meal plans, create shopping lists, and save recipes. While there are many individual applications available for meal planning, creating shopping lists, recipes finders, recipe managers, ingredients scanners etc., what makes the 'Food Planner App' different is that it combines all those individual features into one all-encompassing food planner application. All the individual features work together to create a comprehensive solution for the user. I was unable to identify an existing application that encompasses all those functionalities into a single mobile application.

### 4.0 Technical Approach

For this project, I plan to take an Agile Software Development approach. I am going to use Jira Software to support this approach. Each core feature of the application will be developed in sprints. Using Jira Software, each core feature will be represented by an Epic issue and contain User Stories. Each User Story will describe smaller functional requirements written from the perspective of the end user and will be tested individually. Each User Story will contain sub-tasks to be completed. Once all development tasks in an Epic are developed and tested, I will execute End-To-End testing for that feature. I plan to develop the following features in the following order of priority:

- 1. Basic App Functionality**
- 2. Ingredient List OCR Scanner**
- 3. Recipe Search Engine by Ingredients**
- 4. Recipe Manager**
- 5. Meal Planner**
- 6. Shopping List Generator**

I plan to define the application requirements by breaking the above features down into their functional and non-functional requirements. Each Epic will involve researching, functionality development, defining User Story acceptance criteria, integrating the developed functionality as a solution to the User Story, user interface design and implementation, Acceptance Criteria testing and End-To-End testing. Each of the project deliverables and features will represent a milestone in the project. Each milestone will be broken up into user stories and tasks to be completed. The final requirement specification will be confirmed on 05/03/2022 and submitted to Moodle. Once the final requirements specification has been confirmed, I will update the Jira Software project to reflect these requirements. Each requirement will be created as a User Story in the project. Each User Story will be broken up into sub-tasks and activities to complete. Each task will be given an estimate of how many days I expect to be able to complete the task. In my opinion the two most complex and difficult to develop features will be the Ingredients List OCR Scanner and the Recipe Search Engine. I believe that these are also the two most important and unique features of the application. This is the reason why I have added them to the top of the features priority list. Once I have developed and tested the Ingredients List OCR Scanner and the Recipe Search Engine, I plan to begin working on the dependent sub features such as the Recipe Manager, Meal Planner and Shopping List Generator. The project plan in section 7 of this project proposal provides more information regarding the milestones, tasks, and activities that I have defined so far for the development of this project.

## 5.0 Technical Details

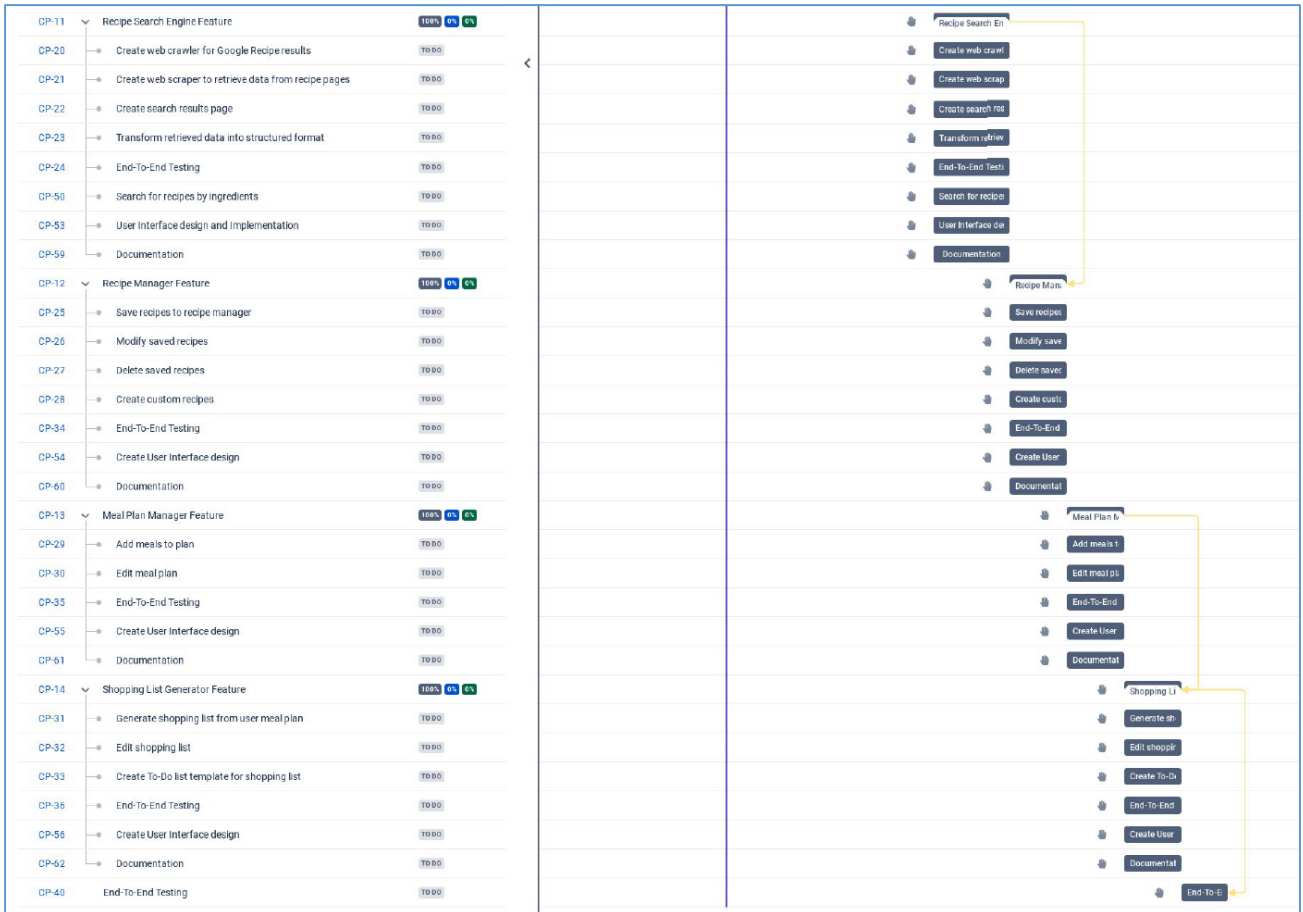
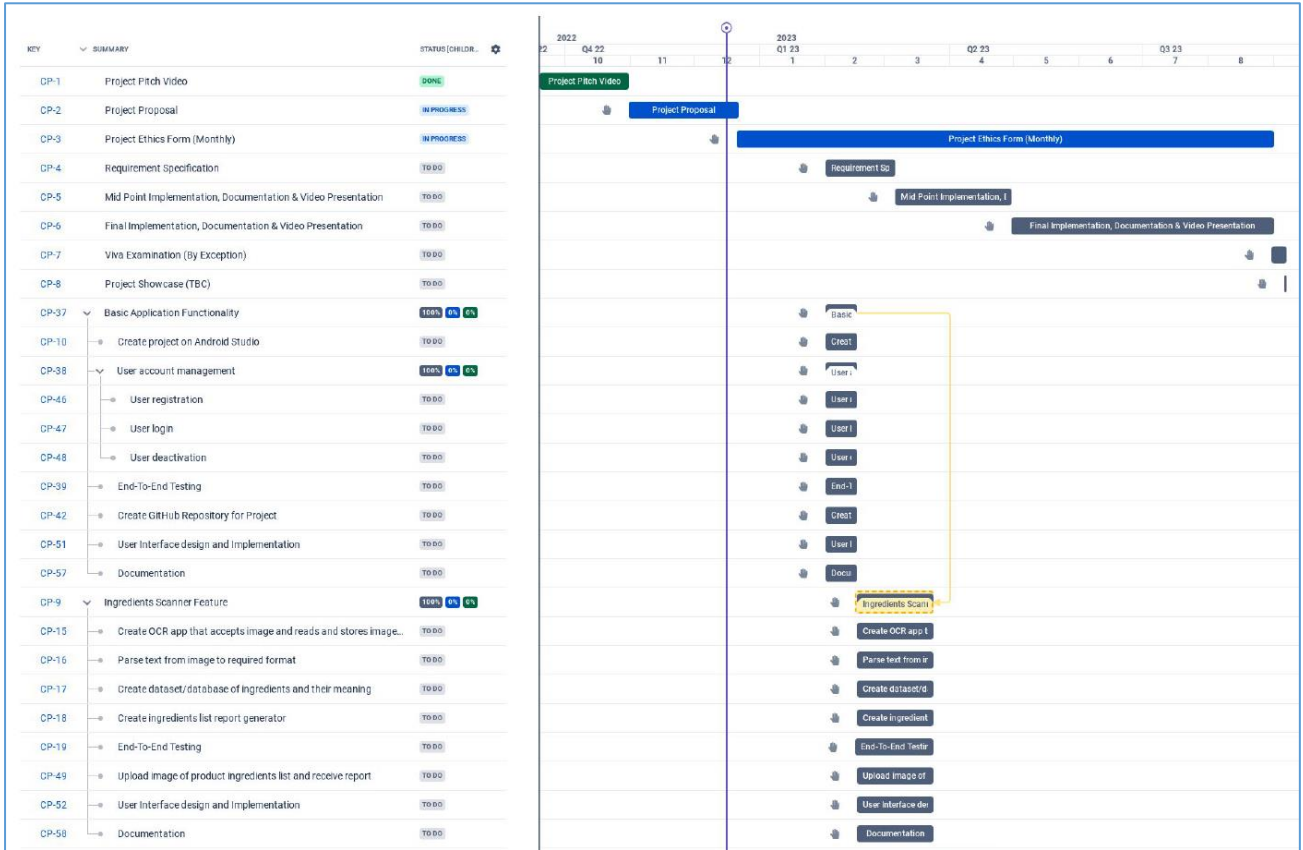
Language / Tool / Library / Method	Planned Use
Android studio	I plan to develop the 'Food Planner App' as an Android mobile application using Android Studio, the official IDE for Google's Android Operating System.
OCR (Optical Character Recognition)	I plan to use OCR technology to convert images of food product ingredient list text into machine-encoded text to create the ingredients report.
Web Scraping and Crawling	I plan to use web scraping in the recipe search engine to extract information from recipe websites that are returned in Google's SERP (Search Engine Results Page).
Java and or Kotlin	I plan to use Java and or Kotlin as the primary programming language to develop the application. I plan to finalise this decision once I begin developing the application in Android Studio.
Python	I may use Python for the web scraping/crawling functionality of the application.
Selenium	I may use Selenium for the web scraping/crawling functionality of the application.

Language / Tool / Library / Method	Planned Use
Google Play Services Libraries	I plan to use Google Play Services in the development of the mobile application.
Android Testing Support Library	I plan to use this testing framework as part of the application testing.
MongoDB / MySQL / PostgreSQL	I have not yet confirmed which database I am going to use for the mobile application development. I can see myself using one of the following three: MongoDB, MySQL, or PostgreSQL.
Espresso	I may use Espresso, the open-source Android UI Testing Framework, to write and automate Android UI tests
GitHub / Git	I plan to use GitHub as the primary project repository.
ML Kit Text Recognition API	I may use this API for capturing and extracting text from an image.
Android Image Cropper	I may use this Google Play Service for cropping images uploaded to the application.
jsoup	I may use this Java HTML parser for The Recipe Search Engine.
Search Algorithms	I plan to use a search algorithm in the Ingredients List OCR Scanner. I will require an appropriate search algorithm to quickly search for matching ingredients in the database. I will confirm which algorithm I use once I begin developing the feature.

## 6.0 Special Resources Required

No special resources will be required for the work completed in this project.

# 7.0 Project Plan



I created the project plan in Jira Software. I have created a Jira Software project to help manage the 'Food Planner App' project development. I created the above Gantt chart using the Jira Software app, 'BigGantt'. The Gantt chart depicts all current issues created in the Jira Software project and their start and end dates. The chart ranges from 19/09/2022 to 06/09/2022. I have created an Epic issue to track each of following project deliverables/milestones: the project pitch video, project proposal, project ethics forms, requirement specification, mid-point implementation/documentation/video presentation, final implementation/documentation/video presentation, viva examination and project showcase. I have added an estimated start date and due date to each of these issues. The start date is the date that I plan to begin working on that deliverable and the due date is the deliverable submission date. As I begin working on each of these Epics, I will add sub task issues to each. This will help me to keep track of all required tasks and their progress. The Gantt chart is synced to the project in Jira Software so all changes to the project will be reflected in the Gantt chart.

I have also added an Epic issue to represent each of the following features of the application: Basic Application Functionality, Ingredients Scanner, Recipe Search Engine, Recipe Manager, Meal Plan Manager and Shopping List Generator. I have added an estimated start date and end date for the development of each of these features as depicted in the project plan. These dates are subject to change based on my progression. The project plan also indicates the order in which I plan to develop each feature of the application. I have allocated more development time to the two key features of the application, the Ingredients List Scanner, and the Recipe Search Engine, compared to the other features. I have done this due to the complexity of those features and the estimated difficulty in developing them. Each feature epic will contain tasks, user stories and sub-tasks. Once I begin developing each feature, and once the final functional requirements have been defined, I will create further tasks and stories which I will then estimate in days. I have included some dependencies in the project plan. For example, the Recipe Manager feature is dependent on the Recipe Search Engine and the Shopping List Generator is dependent on the Meal Plan Manager. I hope to complete development and testing of all features by 09/08/2022.

## 8.0 Testing

Method / Tool / Framework	Planned Use
Unit Testing	Throughout the development of the project, I aim to execute local unit testing on individual components of the software. I may use Junit 5 as the testing framework for this.
Manual User Testing	I plan to complete manual user testing throughout all stages of the development process. I plan to manually test changes by executing test plans and test cases that are based on predefined acceptance criteria. All manual user testing will be completed from the perspective of an end user.
Xray Jira	I may use Xray, the test management tool for Jira, to manage all manual testing. This will facilitate creating test cases, test plans and test execution issues for each development task and user story. Using Xray I will be able to document the execution of the test plan within a test execution ticket. If I find any bugs while executing the test plan, I will be able to create a bug ticket to link to the development task. I will then be able to document the bugs resolution progress in the bug ticket.
Acceptance Criteria Testing	For each task I am going to define acceptance criteria that must be tested for, and met, for the testing to be marked as passed.
E2E (End-To-End) Testing	At the end of each feature development, I plan to complete E2E manual testing to ensure that all development work within the new feature work together and with other parts of the application. I will also complete E2E testing once all features have been developed to ensure that the application is functioning as expected.
Regression Testing	I plan to incorporate manual regression testing when testing each new feature of the application to ensure that the application still functions as expected and that the new changes have not introduced any bugs to previously developed and tested parts of the system.
Android Testing Support Library	I plan to make use of this framework to test the application. The APIs provided by this library will allow me to build and run test code for the application, some of which include Junit 4 and functional user interface (UI) tests.
Integration Testing	Each time I develop a new feature, I will integrate it into the application. I will then use the Top-down Integration Testing method where the higher-level modules are tested first and then the lower-level models are tested and integrated to test the applications functionality.
Espresso	I may use Espresso, the open-source Android UI Testing Framework, to write and automate Android UI tests.

## References

CookBook Co. Pty Ltd, 2022. *The ultimate pocket recipe manager*. [Online]

Available at: <https://thecookbookapp.com/>

[Accessed 21 December 2022].

Eat This Much Inc, 2022. *Put your diet on autopilot*. [Online]

Available at: <https://www.eatthismuch.com/>

[Accessed 21 December 2022].

EPA, 2022. *Food Waste Statistics*. [Online]

Available at: <https://www.epa.ie/our-services/monitoring--assessment/waste/national-waste-statistics/food/>

[Accessed 21 December 2022].

Infood Team, 2022. *Infood - Ingredients food scan*. [Online]

Available at: <https://play.google.com/store/apps/details?id=net.infood.app&gl=us&hl=en>

[Accessed 21 December 2022].

SuperCook, 2022. *SuperCook - Recipe Generator*. [Online]

Available at: [https://play.google.com/store/apps/details?id=com.supercook.app&hl=en\\_IE&gl=US](https://play.google.com/store/apps/details?id=com.supercook.app&hl=en_IE&gl=US)

[Accessed 21 December 2022].

Whisk, 2022. *The Ultimate Cooking App*. [Online]

Available at: <https://whisk.com/>

[Accessed 21 December 2022].

Yuka, 2022. *Yuka - The mobile app that scans your diet and cosmetics*. [Online]

Available at: <https://yuka.io/en/>

[Accessed 21 December 2022].

## 6.2 Reflective Journals

### Supervision & Reflection Template

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc (Honours) in Computing – Evening (BSHCSDE4)
<b>Supervisor</b>	Enda Stafford

**Month: November 2022**

<p><b>What?</b></p> <p>Reflect on what has happened in your project this month?</p> <ul style="list-style-type: none"> <li>– On 30/10/22 I submitted my Project Pitch Video. In this video I outlined the problems I will be attempting to solve in my project, why the project should be attempted, what the project will do, and why the project is challenging and how it is different from what has been done before.</li> <li>– On 11/11/22 I was notified that Enda Stafford is my project supervisor.</li> <li>– On 11/11/22 I received my project pitch feedback.</li> <li>– On 21/11/22 I had a meeting with Enda Stafford to discuss my project pitch feedback, project timeline, project proposal and upcoming deadlines and deliverables.</li> </ul>	
<p><b>So What?</b></p> <p>Consider what that meant for your project progress. What were your successes? What challenges still remain?</p> <ul style="list-style-type: none"> <li>– In terms of successes, on 11/11/22 I received confirmation that my project pitch idea was accepted and that no amendments were required.</li> <li>– The next challenge is to create my Project Proposal before the deadline on 17/12/22.</li> </ul>	
<p><b>Now What?</b></p> <p>What can you do to address outstanding challenges?</p> <ul style="list-style-type: none"> <li>– For creating the project proposal document – I am going to do the following: <ul style="list-style-type: none"> <li>○ I am going to complete more in-depth research and analysis of the technologies and methodologies that I am going to use to create my project.</li> <li>○ I am going to define in detail the project objectives and requirements.</li> <li>○ I am going to create a project plan.</li> <li>○ I am going to create a test plan.</li> <li>○ I am going to identify any special resources required in creating the project.</li> <li>○ I am going to conduct additional market research.</li> </ul> </li> </ul>	
<b>Student Signature</b>	Ruby Lennon 30/11/2022



## Supervision & Reflection Template

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc (Honours) in Computing – Evening (BSHCSDE4)
<b>Supervisor</b>	Enda Stafford

**Month: December 2022**

<p><b>What?</b></p> <p>Reflect on what has happened in your project this month?</p> <ul style="list-style-type: none"><li>– On 15/12/22 I had a call with Enda Stafford to discuss my Project Proposal and project ethics declaration requirements.</li><li>– On 21/12/22 I submitted my Project Proposal.</li></ul>	
<p><b>So What?</b></p> <p>Consider what that meant for your project progress. What were your successes? What challenges still remain?</p> <ul style="list-style-type: none"><li>– In terms of successes, through creating the Project Proposal, I completed the following:<ul style="list-style-type: none"><li>○ Defined the project objectives and background.</li><li>○ Conducted additional market research.</li><li>○ Created an initial project plan.</li><li>○ Worked out the initial project technical approach.</li><li>○ Completed further research regarding the application proposed technical details &amp; testing.</li></ul></li><li>– One of the next challenges is to create the project Requirement Specification before the deadline on 05/03/23.</li><li>– I plan to begin creating the basic application functionality by 01/02/23.</li></ul>	
<p><b>Now What?</b></p> <p>What can you do to address outstanding challenges?</p> <ul style="list-style-type: none"><li>– To create the Requirement Specification document – I am going to do the following:<ul style="list-style-type: none"><li>○ Look at each function/feature of the application and map out how the software will execute them in technical terms.</li><li>○ I believe creating the basic app functionality in Android Studio will support the process of creating the Requirement Specification document. I think it will provide an insight into the application structure and how to best develop the application features.</li></ul></li></ul>	
<b>Student Signature</b>	Ruby Lennon 05/01/23

**Supervision & Reflection Template**

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc (Honours) in Computing – Evening (BSHCSD4)
<b>Supervisor</b>	Enda Stafford

**Month: January 2023****What?**

Reflect on what has happened in your project this month?

- On 04/02/23 I had a call with Enda Stafford to discuss my Project Proposal feedback, my current progress with the project, the Requirements Specification, and questions I had regarding the project process.
- I downloaded Android Studio and started familiarising myself with how to use the IDE for Android mobile application development.

**So What?**

Consider what that meant for your project progress. What were your successes? What challenges still remain?

- One of the next challenges is to create the project Requirement Specification before the deadline on 05/03/23.
- One success is that I have begun familiarising myself with Android Studio so that I can begin working on the project code.

**Now What?**

What can you do to address outstanding challenges?

- I am going to create the functional and non-functional Requirement Specifications.
- I am going to continue following online tutorials on how to develop mobile applications using Android Studio so that I can begin developing the project features.

**Student Signature**

Ruby Lennon 05/02/23

**Supervision & Reflection Template**

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc (Honours) in Computing – Evening (BSHCSE4)
<b>Supervisor</b>	Enda Stafford

**Month: February 2023****What?**

Reflect on what has happened in your project this month?

- On 23/02/23 I had a call with Enda Stafford to discuss my Project.
- I completed the Requirement Specification document and submitted it on 05/03/23.

**So What?**

Consider what that meant for your project progress. What were your successes? What challenges still remain?

- One of the next challenges is complete the Mid-Point Implementation, Documentation & Video Presentation before the deadline on 29/04/23.
- Some successes are that I have implemented the basic app functionality such as user authentication using Firebase Authenticate, I have implemented basic recipe CRUD functionality using Firebase Realtime Database, and I have implemented basic OCR Text Recognition functionality using the Google ML Kit Vision API.

**Now What?**

What can you do to address outstanding challenges?

- I am going to complete implementing one or more of the functional requirements in the application.
- I am going to complete the Mid-Point Implementation, Documentation & Video Presentation before the deadline on 29/04/23.

**Student Signature**

Ruby Lennon 05/03/23

**Supervision & Reflection Template**

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc (Honours) in Computing – Evening (BSHCSDE4)
<b>Supervisor</b>	Enda Stafford

**Month: March 2023**

<b>What?</b>  Reflect on what has happened in your project this month? <ul style="list-style-type: none"><li>– On 25/03/23 I had a call with Enda Stafford to discuss my Project Requirements Specification feedback.</li></ul>	
<b>So What?</b>  Consider what that meant for your project progress. What were your successes? What challenges still remain? <ul style="list-style-type: none"><li>– One of the next challenges is complete the Mid-Point Implementation, Documentation &amp; Video Presentation before the deadline on 29/04/23.</li></ul>	
<b>Now What?</b>  What can you do to address outstanding challenges? <ul style="list-style-type: none"><li>– I am going to complete the Mid-Point Implementation, Documentation &amp; Video Presentation before the deadline on 29/04/23.</li></ul>	
<b>Student Signature</b>	Ruby Lennon 02/04/23

**Supervision & Reflection Template**

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc (Honours) in Computing – Evening (BSHCSDE4)
<b>Supervisor</b>	Enda Stafford

**Month: April 2023**

<b>What?</b>  Reflect on what has happened in your project this month? <ul style="list-style-type: none"><li>– On 15/04/23 I had a call with Enda Stafford to discuss the project Mid-Point Examination requirements</li><li>– On 01/05/23 I completed the Mid-Point Documentation, Demo and Presentation</li></ul>	
<b>So What?</b>  Consider what that meant for your project progress. What were your successes? What challenges still remain? <ul style="list-style-type: none"><li>– One of the next challenges is develop the OCR Ingredients List Scanner functionality before 27/05/23.</li><li>– I completed the development for the User Management and Recipe Manager features</li></ul>	
<b>Now What?</b>  What can you do to address outstanding challenges? <ul style="list-style-type: none"><li>– I am going to develop the OCR Ingredients List Scanner functionality and once working will implement into the main FoodPlannerApp app code.</li></ul>	
<b>Student Signature</b>	Ruby Lennon 01/05/23

**Supervision & Reflection Template**

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc in Computing
<b>Supervisor</b>	Enda Stafford

**Month: May 2023**

<b>What?</b>  Reflect on what has happened in your project this month? <ul style="list-style-type: none"><li>- Continued to develop Food Ingredients OCR Scanner feature.</li></ul>	
<b>So What?</b>  Consider what that meant for your project progress. What were your successes? What challenges still remain? <ul style="list-style-type: none"><li>- Challenge 1: Complete food ingredients scanner feature.</li><li>- Challenge 2: Begin developing recipe search feature.</li></ul>	
<b>Now What?</b>  What can you do to address outstanding challenges? <ul style="list-style-type: none"><li>- Prioritise completion of food ingredients scanner feature.</li><li>- Identify best approach for developing recipe search engine feature.</li></ul>	
<b>Student Signature</b>	Ruby Lennon

**Supervision & Reflection Template**

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc in Computing
<b>Supervisor</b>	Enda Stafford

**Month: June 2023**

<b>What?</b>  Reflect on what has happened in your project this month?  <ul style="list-style-type: none"><li>- Completed development of the Food Ingredients OCR Scanner feature and the Recipe Search feature.</li><li>- On 20/06/23 I had a call with Enda Stafford to receive feedback on my project Mid-Point Submission.</li></ul>	
<b>So What?</b>  Consider what that meant for your project progress. What were your successes? What challenges still remain?  <ul style="list-style-type: none"><li>- Success: Completing development of the Food Ingredients OCR Scanner feature and the Recipe Search feature.</li><li>- Challenge 1: Complete remaining two features - Meal Planner feature &amp; Shopping List feature.</li><li>- Challenge 2: Complete project final report, document testing, and the project poster.</li></ul>	
<b>Now What?</b>  What can you do to address outstanding challenges?  <ul style="list-style-type: none"><li>- Prioritise completion of Meal Planner feature &amp; Shopping List feature and then complete the project final report, document testing, and the project poster.</li></ul>	
<b>Student Signature</b>	Ruby Lennon 02/06/23

**Supervision & Reflection Template**

<b>Student Name</b>	Ruby Lennon
<b>Student Number</b>	X19128355
<b>Course</b>	BSc in Computing
<b>Supervisor</b>	Enda Stafford

**Month: July 2023**

<b>What?</b>  Reflect on what has happened in your project this month?  - Completed development of the FoodPlannerApp	
<b>So What?</b>  Consider what that meant for your project progress. What were your successes? What challenges still remain?  - Success – successfully developed all six functional requirements of the application.	
<b>Now What?</b>  What can you do to address outstanding challenges?  - Complete and submit final submission requirements.	
<b>Student Signature</b>	Ruby Lennon 06/08/23