



**Module Code & Module Title**

**CS6PO5NT-Final Year Project**

**Assessment Weightage & Type**

**Final Year Project Final Report (40%)**

**Year and Semester**

**2023-24 Autumn**

**Student Name: Aasmee Rai**

**London Met ID: 22072035**

**College ID: NP05CP4A220006**

**Assignment Submission Date: August 01, 2025**

**Internal Supervisor: Sonam Rai**

**External Supervisor: Utsav Dhungana**

**Title: Ingreedy (Smart Recipe App)**

**GitHub link: <https://github.com/Aasmee/FYP-new-.git>**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded*

## **Abstract**

The "Ingreedy" project proposes an innovative solution to the challenges of meal preparation and pantry management through the development of a Smart Recipe Finder application. Leveraging advanced technologies such as artificial intelligence (AI) for ingredient recognition and personalized recipe recommendations, the app addresses issues like food waste, inefficient grocery planning, and dietary accommodations. Key features include AI-powered ingredient recognition via OCR and barcode scanning, automated pantry management with expiry tracking and inventory alerts, and a personalized recipe generator that aligns with user preferences and available ingredients.

The app also fosters community engagement through a recipe-sharing platform. Designed with user-friendliness, sustainability, and inclusiveness in mind, "Ingreedy" enhances the cooking experience while promoting efficient ingredient usage. By incorporating a Scrum-based agile methodology, the project ensures iterative development and stakeholder feedback to achieve a functional and user-centric application. This project aims to redefine cooking and pantry management, making meal preparation an efficient, enjoyable, and resourceful process

## Table of Contents

Abstract.....	i
Chapter 1: Introduction.....	1
1.1. Introduction to the topic.....	1
1.2. Problem Statement.....	1
1.3. Project as a Solution.....	2
1.4. Aims and Objectives .....	3
Chapter 2: Background.....	4
2.1. About client/end user .....	4
2.2. Understanding the solution .....	4
2.3. Similar projects .....	4
2.4. Comparisons between similarity of the systems .....	6
Chapter 3: Development.....	7
3.1. Approach / Methodology Considerations .....	7
3.2. Stages of chosen Approach / Methodology .....	8
3.3. Survey Results .....	10
3.3.1. Pre-Survey Results.....	10
3.3.2. Post-Survey Results .....	10
3.4. Requirement Analysis .....	11
3.4.1. SRS .....	11
3.5. Designs.....	13
3.5.1. Work Breakdown Structure.....	13
3.5.2. Entity Relation Diagram .....	14
3.5.3. Data Flow Diagram (DFD) .....	19
3.5.4. Use Case Diagrams .....	19

3.5.5.	High-level Expanded Use Case Diagram .....	20
3.5.6.	Flowchart .....	28
3.6.	Implementation .....	29
3.6.1.	Database .....	29
3.6.2.	Code Implementation.....	39
Chapter 4:	Testing and Analysis .....	47
4.1.	Test Plan.....	47
4.2.	System Testing .....	47
4.2.1.	Authentication Testing .....	48
4.2.2.	Pantry Testing .....	1
4.2.3.	Post Testing.....	1
4.2.4.	Shopping List Testing .....	1
4.2.5.	Recipe Testing.....	1
4.3.	API testing .....	2
4.3.1.	Register .....	3
4.3.2.	Login.....	4
4.3.3.	Make post.....	5
4.3.4.	Get post .....	6
4.3.5.	Delete post .....	7
4.3.6.	Update posts.....	8
4.3.7.	Like post.....	9
4.3.8.	Bookmark posts .....	10
4.3.9.	View bookmarked posts.....	11
4.3.10.	Add comment.....	12
4.3.11.	Get comment.....	13

4.3.12.	Delete comment .....	14
4.3.13.	Edit comment.....	15
4.3.14.	Get notification .....	16
4.3.15.	Read notifications .....	17
4.3.16.	Add items to Shopping List .....	18
4.3.17.	Get shopping list .....	19
4.3.18.	Update list .....	20
4.3.19.	Delete list item.....	21
4.3.20.	Get all recipes .....	22
4.3.21.	Get a single recipe.....	23
4.3.22.	Getting pantry items.....	24
4.3.23.	Add pantry items.....	25
4.3.24.	Delete Pantry items .....	26
4.3.25.	Save search tags .....	27
4.3.26.	Get profile .....	28
4.3.27.	Recommendations.....	29
Chapter 5:	Conclusion .....	30
5.1.	Advantages.....	30
5.2.	Limitations .....	30
Chapter 6:	Future Work .....	31
Chapter 7:	References.....	32
Chapter 8:	Appendix.....	34
8.1.	Appendix A: Pre-Survey .....	34
8.1.1.	Pre-Survey Form.....	34
8.1.2.	Sample of Filled Pre-Survey Forms.....	39

8.1.3.	Pre-Survey Result .....	43
8.2.	Appendix B: Post-Survey .....	48
8.2.1.	Post-Survey Form .....	48
8.2.2.	Sample of Filled Post-Survey Forms .....	51
8.2.3.	Post-Survey Result.....	54
8.3.	Appendix C: Sample Codes .....	58
8.3.1.	Sample Code of The UI .....	58
8.4.	Appendix D: Designs.....	84
8.4.1.	Gantt Chart.....	84
8.4.2.	Work Breakdown Structure.....	85
8.4.3.	Flowcharts.....	85
8.4.4.	Data Flow Diagrams .....	85
8.4.5.	Use Case.....	86
8.4.6.	Wireframe .....	87
8.5.	Appendix E: Screenshots of The System.....	98

## Table of Figures

Figure 1: Scrum Methodology .....	9
Figure 2: Total responses accumulated in pre-survey form.....	10
Figure 3: Entity Relationship Diagram .....	14
Figure 4: Collaboration diagram of “Manage Pantry”.....	21
Figure 5: Sequence diagram of “Manage Pantry” .....	21
Figure 6: Collaboration diagram of “Search Recipe” .....	22
Figure 7: Sequence diagram for “Search Recipe” .....	23
Figure 8: Collaboration of “Make Shopping List” .....	24
Figure 9: Sequence diagram for “Make Shopping List”.....	25
Figure 10: Collaboration of “Make Post” .....	26
Figure 11: Sequence diagram for “Make Post” .....	27
Figure 12: Database Structure.....	29
Figure 13: Registering new user .....	48
Figure 14: Login page.....	48
Figure 15: Entering login credentials.....	1
Figure 16: Home page.....	1
Figure 17: Adding item to pantry .....	1
Figure 18: Succession message.....	1
Figure 19: Creating a new post .....	1
Figure 20: New post on home page .....	1
Figure 21: Comments box.....	1
Figure 22: New comment added .....	1
Figure 23: Post before liking.....	1
Figure 24: Post after like.....	1
Figure 25: Bookmarking post .....	1
Figure 26: Bookmarked post in saved section .....	1
Figure 27: Adding item to list .....	1
Figure 28: List with added item .....	1
Figure 29: Shopping list before deleting item.....	1
Figure 30: Shopping list after deleting the item.....	1

Figure 31: Register API testing.....	3
Figure 32: Login API testing.....	4
Figure 33: Post creation API testing .....	5
Figure 34: Get post API testing.....	6
Figure 35: Deleting post API testing.....	7
Figure 36: Updating posts API testing .....	8
Figure 37: Liking post API testing.....	9
Figure 38: Bookmarking posts API testing.....	10
Figure 39: View bookmarked posts API testing .....	11
Figure 40: Adding comment API testing .....	12
Figure 41: Getting all comments on a specific post.....	13
Figure 42: Deleting comment API testing .....	14
Figure 43: Editing comment API testing .....	15
Figure 44: Getting notification API testing.....	16
Figure 45: Reading notification API testing .....	17
Figure 46: Adding items to list API testing.....	18
Figure 47: Get shopping list of the active user API testing .....	19
Figure 48: Update shopping list item API testing.....	20
Figure 49: Deleting list item API testing .....	21
Figure 50: Get all recipes API testing .....	22
Figure 51: Get a single recipe by id API testing .....	23
Figure 52: Getting pantry items API testing .....	24
Figure 53: Adding items to pantry API testing .....	25
Figure 54: Deleting items from pantry API testing.....	26
Figure 55: Save tags from the user's search API testing .....	27
Figure 56: Get user profile API testing .....	28
Figure 57: Recommendation system API testing.....	29
Figure 58: Pre-Survey Form pt.1 .....	34
Figure 59: Pre-Survey Form pt.2 .....	35
Figure 60: Pre-Survey Form pt.3 .....	36
Figure 61: Pre-Survey Form pt.4 .....	37

Figure 62: Pre-Survey Form pt.5 .....	38
Figure 63: Filled Pre-Survey Form Sample pt.1 .....	39
Figure 64: Filled Pre-Survey Form Sample pt.2 .....	40
Figure 65: Filled Pre-Survey Form Sample pt.3 .....	40
Figure 66: Filled Pre-Survey Form Sample pt.4 .....	41
Figure 67: Filled Pre-Survey Form Sample pt.5 .....	42
Figure 68: Filled Pre-Survey Form Sample pt.6 .....	42
Figure 69: Pre-Survey Q1 .....	43
Figure 70: Pre-Surgery Q2.....	43
Figure 71: Pre-Survey Q3 .....	44
Figure 72: Pre-Survey Q4 .....	44
Figure 73:Pre-Survey Q5 .....	45
Figure 74: Pre-Survey Q6 .....	45
Figure 75: Pre-Survey Q7 .....	46
Figure 76: Pre-Survey Q8 .....	46
Figure 77: Pre-Survey Q9 .....	46
Figure 78: Pre-Survey Q10 .....	47
Figure 79: Pre-Survey Q11 .....	47
Figure 80: Post Survey Form pt.1 .....	48
Figure 81: Post-Survey Form pt.2.....	49
Figure 82: Post-Survey Form pt.3.....	50
Figure 83: Post-Survey Form pt.4.....	50
Figure 84: Post-Survey Smple Form pt.1.....	51
Figure 85: Post-Survey Sample Form pt.2.....	52
Figure 86: Post-Survey Sample Form pt.3.....	52
Figure 87: Post-Survey Sample Form pt.4.....	53
Figure 88:Post-Survey Q1.....	54
Figure 89: Post-Survey Q2.....	54
Figure 90: Post-Survey Q3.....	55
Figure 91: Post-Survey Q4.....	55
Figure 92: Post-Survey Q5.....	56

Figure 93: Post-Survey Q6.....	56
Figure 94: PosT-Survey Q7.....	57
Figure 95: Gantt Chart .....	84
Figure 96: Work Break Structure .....	85
Figure 97: Data Flow Diagram .....	85
Figure 98: Use case diagram.....	86
Figure 99: Register Page Wireframe.....	87
Figure 100: Login Page Wireframe.....	88
Figure 101: Home Page Wireframe .....	89
Figure 102: Recipe Page Wireframe .....	90
Figure 103: Scan result wireframe .....	91
Figure 104: Recipe details wireframe .....	92
Figure 105: Create Post Wireframe.....	93
Figure 106: Pantry Page Wireframe.....	94
Figure 107: Add ingredient wireframe.....	95
Figure 108: Saved recipes screen wireframe .....	96
Figure 109: Profile Page Wireframe .....	97
Figure 110: Login page screenshot .....	98
Figure 111: Register page screenshot.....	99
Figure 112: Home page screenshot.....	100
Figure 113: List page screenshot .....	101
Figure 114: Recipes page screenshot .....	102
Figure 115: Create Post page screenshot .....	103
Figure 116: Pantry page screenshot .....	104
Figure 117: Profile page screenshot.....	105

## Table of Tables

Table 1: Similarity comparison.....	6
Table 2: Registration test case.....	48
Table 3: Login test case.....	1
Table 4: Adding pantry item test case .....	1
Table 5: Creating post test case.....	1
Table 6: Making comments test case .....	1
Table 7: Liking post test case.....	1
Table 8: Bookmarking posts test case .....	1
Table 9: Adding item to list test case .....	1
Table 10: Deleting list item test case .....	1
Table 11: Filtering recipe list by tag test case .....	1

## **Table of Abbreviations**

<b>Abbreviations</b>	<b>Full Form</b>
AI	Artificial Intelligence
OCR	Optical Character Recognition
DFD	Data Flow Diagram
ERD	Entity Relationship Diagram
SRS	Software Requirement Specification
WBS	Work Breakdown Structure
UI	User Interface

# **Chapter 1: Introduction**

## **1.1. Introduction to the topic**

Ingreedy (Smart Recipe Finder) transforms meal preparation with innovative features like AI-powered ingredient recognition, personalized recipe suggestions, pantry tracking, and automatic shopping list generation. The app helps users make the most of their ingredients, explore recipes tailored to their dietary preferences, and plan meals efficiently.

Beyond its practical benefits, Ingreedy fosters creativity by encouraging users to try new recipes and reduce food waste. Its interactive community forum allows users to share tips, post recipes, and connect with like-minded cooking enthusiasts. Designed for convenience, sustainability, and inclusivity, Ingreedy ensures cooking is an enjoyable and resourceful experience for users of all skill levels.

## **1.2. Problem Statement**

In today's fast-paced lifestyle, meal preparation often becomes a challenging task for individuals, families, and even professional cooks. A common problem is the inefficient use of available ingredients, leading to food waste and missed opportunities to create meals from what's already in the pantry. (Porter & Reay, 2015) People often find it time-consuming and tedious to plan meals, especially when trying to accommodate dietary restrictions, nutritional goals, or personal preferences. (Huff, 2022)

Another significant issue is the lack of tools to manage pantry inventory effectively. Ingredients may expire or run out unnoticed, causing disruptions in meal planning and unnecessary trips to the grocery store. (THE OWL, 2024) The abundance of online recipes can overwhelm users, making it hard to find ones that suit their tastes or needs. Additionally, there's a growing demand for community and inspiration in cooking, highlighting a gap for an all-in-one kitchen management and recipe discovery tool.

### **1.3. Project as a Solution**

The Ingreedy (Smart Recipe Finder) simplifies cooking and pantry management with a user-friendly platform powered by AI. By using photos or barcode scans, users can quickly digitize their pantry contents, enabling the app to suggest recipes based on the ingredients they already have. This feature helps reduce food waste, save time, and make meal preparation more efficient.

In addition to recipe suggestions, Ingreedy tracks ingredient quantities, monitors expiry dates, and sends timely alerts for low-stock or near-expiry items, helping users stay organized and plan meals effectively. The app also generates shopping lists for missing ingredients and allows users to customize recommendations based on dietary preferences or nutritional goals, ensuring a personalized experience.

Beyond practicality, Ingreedy fosters a sense of community through a recipe-sharing forum where users can exchange ideas, discover new techniques, and explore diverse cuisines. With multilingual support, the app is accessible to a global audience, transforming cooking into a creative, efficient, and enjoyable activity for users around the world.

## **1.4. Aims and Objectives**

### **Aim:**

To reduce food waste by enabling users to utilize existing pantry ingredients through intelligent recipe suggestions.

### **Objectives:**

- Develop a pantry management module that allows users to track ingredient quantities and receive alerts for items nearing expiration.
- Enable personalized recipe filtering.
- Create shopping lists.
- Design an intuitive user interface that encourages frequent use of the pantry feature and simplifies meal planning.
- Provide a “Surprise Me” feature that uses AI to suggest creative recipes using leftover or rarely used ingredients.
- Community forums for users to interact.

## **Chapter 2: Background**

### **2.1. About client/end user**

The end users for the "Ingreedy" app are individuals or families looking to simplify their cooking and pantry management processes. This includes users with dietary restrictions, culinary enthusiasts, or people aiming to reduce food waste. The app is also designed for a global audience with multilingual support to ensure inclusivity.

### **2.2. Understanding the solution**

The app leverages AI-powered OCR for ingredient recognition, personalized recipe recommendations, and efficient pantry management. It integrates advanced technology to automate inventory tracking, provide dietary customization, and generate automated shopping lists. Additionally, the app fosters a community environment where users can share recipes and engage with others.

### **2.3. Similar projects**

Several projects and applications like Ingreedy have been developed in the past, each addressing various aspects of meal preparation and pantry management. Below is a concise review of some noteworthy examples.

#### **i. Yummly**

- The Yummly app is a smart cooking companion, offering personalized recipe recommendations tailored to your tastes, dietary preferences, and allergies. With features like ingredient-based searches to reduce food waste, quick recipe filters, a digital cookbook for saving favorites, and step-by-step guided cooking, Yummly makes mealtime effortless. Its smart shopping list and meal planning tools streamline your grocery trips and daily schedules, while Yummly Premium unlocks guided videos, nutritional insights, and advanced planning options. Whether you're exploring new cuisines or optimizing leftovers, Yummly simplifies cooking for every lifestyle. (Aptoide, 2025)

### **ii. Tasty**

- Tasty is a comprehensive cooking platform offering over 10,000 recipes, combining detailed instructions with video content and personalized recommendations. Its AI assistant, Botatouille, uses user preferences to suggest recipes, encouraging exploration of diverse culinary options. The platform integrates with Walmart, allowing users to shop for ingredients and arrange delivery or pickup directly through the app. Tasty also fosters a community-driven experience, enabling users to share recipes, gain inspiration, and exchange cooking knowledge. Additional features include custom recipe collections, an ingredient-based search function, adjustable serving sizes, and insights from creators and community feedback, making it a versatile tool for home cooks. (Sensor Tower, 2025)

### **iii. Cookpad**

- Cookpad is a global platform for recipes and meal planning, showcasing home-cook-created dishes with clear, step-by-step instructions. Users can search for recipes by ingredients to reduce food waste, organize them into custom collections, and plan weekly menus. The app also features a vibrant cooking community where users can share recipes, exchange tips, and post photos of their creations. Designed for all skill levels and dietary needs, Cookpad provides inspiration for everything from quick breakfasts to intricate dinners and international cuisine. (Aptoide, 2025)

## 2.4. Comparisons between similarity of the systems

*Table 1: Similarity comparison*

Features	Yummly	Tasty	Cookpad	Ingreedy
AI-powered ingredient recognition	No	No	No	Yes
Pantry Management	No	No	No	Yes
Recipe generation based on pantry items	No	No	No	Yes
Profile management	Yes	Minimal	Yes	Yes
Community engagement	Yes	No	Yes	Yes
Automatic shopping list generation	Yes	No	No	Yes

## **Chapter 3: Development**

### **3.1. Approach / Methodology Considerations**

For the development of the Smart Recipe Finder App, several development methodologies were carefully evaluated to determine the most suitable approach for the project's needs and objectives. Each methodology presented unique strengths and weaknesses when applied to the dynamic and feature-rich nature of this project.

- **Waterfall**

- Waterfall provided a structured and linear framework, offering clear stages for planning, design, development, and testing. However, its rigid structure proved unsuitable for accommodating evolving requirements and iterative refinements, which are critical for a project of this scope and complexity.

- **Kanban**

- Kanban excelled in tracking workflow and ensuring visibility across tasks, promoting a smooth and efficient process. Despite its strengths in workflow management, it lacked time-boxed iterations, which are essential for managing the complex and interdependent features of the app, such as AI-based ingredient recognition and pantry management.

- **Lean**

- Lean emphasized minimizing waste and maximizing efficiency, aligning well with the goal of delivering value-driven outcomes. However, it fell short in supporting the frequent collaboration, adaptive planning, and iterative refinement necessary for such a dynamic and user-centric project.

By understanding the limitations and benefits of each approach, the team was able to make an informed decision to adopt a methodology that best aligns with the app's goals and ensures successful delivery.

### **3.2. Stages of chosen Approach / Methodology**

After evaluating the Waterfall, Kanban, and Lean methodologies, Scrum was chosen as the most suitable methodology for the development of the "Ingreedy" Smart Recipe Finder app. Scrum's emphasis on flexibility, collaboration, and iterative progress makes it ideal for handling the app's dynamic requirements and complex feature set. Below is a brief description of the key phases in Scrum:

- Product Backlog Creation
  - The team identifies and prioritizes all the tasks and features required for the project. These are documented in the Product Backlog, serving as a dynamic to-do list.
- Sprint Planning
  - Before each sprint (a time-boxed iteration), the team selects high-priority tasks from the Product Backlog and defines clear goals for that sprint.
- Sprint Execution
  - During the sprint, the team works collaboratively to develop the selected features. Daily Scrum meetings are held to discuss progress, address obstacles, and ensure alignment.
- Sprint Review
  - At the end of the sprint, the team demonstrates the completed work to stakeholders, gathers feedback, and identifies areas for improvement.
- Sprint Retrospective
  - The team reflects on the sprint process to identify successes, challenges, and opportunities for improvement in future iterations.
- Increment Delivery
  - Each sprint delivers a functional increment of the app, contributing to the project's overall progress and ensuring continuous value delivery.

This iterative process ensures that the app evolves based on stakeholder feedback, allowing the team to adapt to changing requirements while maintaining a focus on delivering high-quality features.

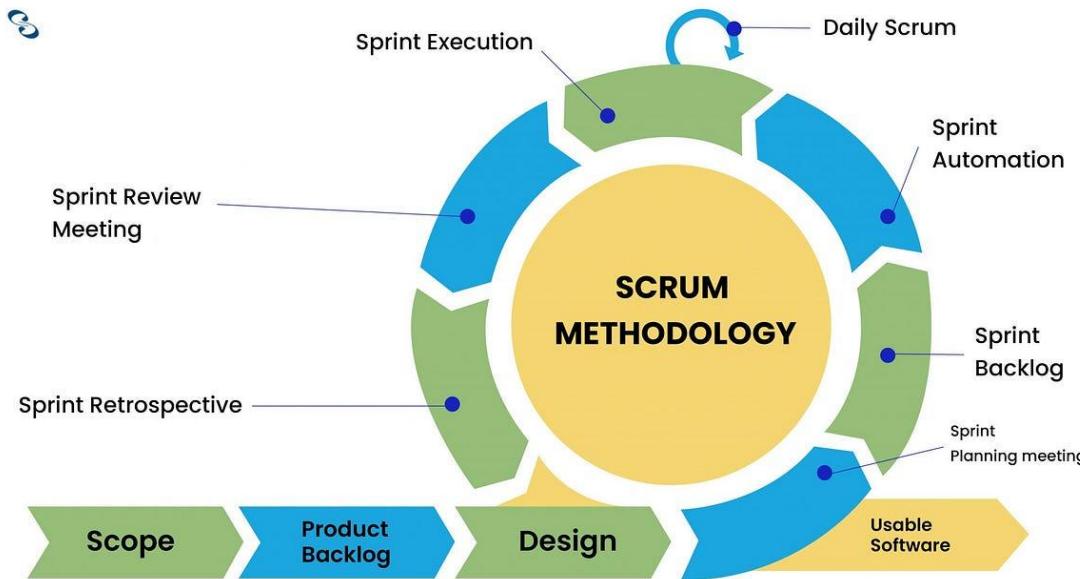


Figure 1: Scrum Methodology

### 3.3. Survey Results

#### 3.3.1. Pre-Survey Results

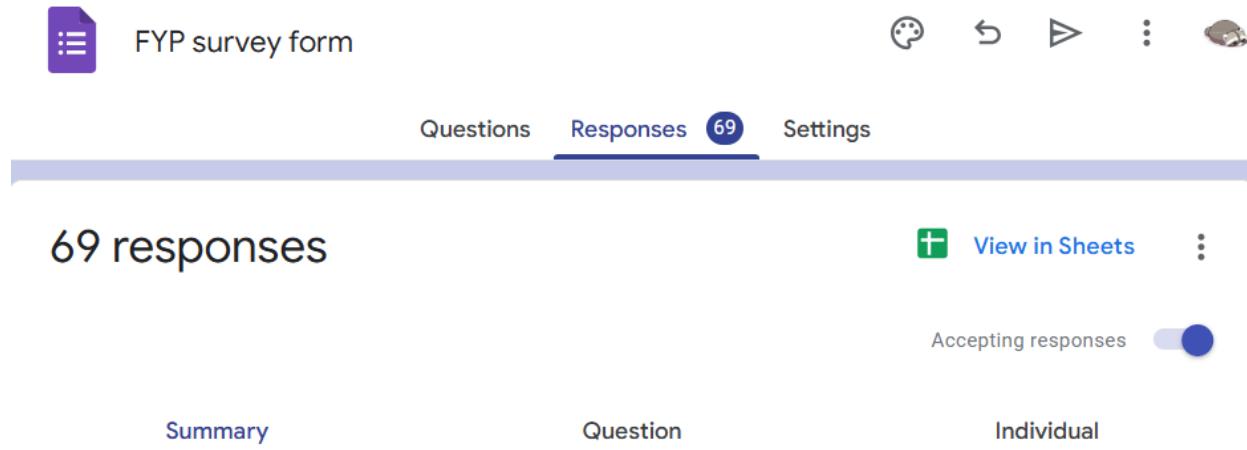


Figure 2: Total responses accumulated in pre-survey form

A comprehensive survey was conducted to identify and understand the key challenges and user needs that the project aims to address. The primary objective was to gather insights into common issues faced during meal preparation, pantry management, and recipe selection, ensuring that the final solution effectively responds to real user concerns and expectations.

The findings of the pre-survey can be found in the appendix section.

#### 3.3.2. Post-Survey Results

The survey was conducted to evaluate users' experiences with the app, focusing on how intuitive, functional, and satisfying they found it during their interaction with its features. It aimed to gather feedback on usability, performance, and overall satisfaction to identify strengths and areas for improvement. The responses help guide future development and ensure the app better meets user needs and expectations.

The findings of the post-survey can be found in the appendix section.

## **3.4. Requirement Analysis**

### **3.4.1. SRS**

#### **Functional requirements**

##### **Add Ingredients:**

Users can add ingredients manually or via an OCR scanner, and the system will store details such as expiry dates and stock levels. Notifications will be sent for items nearing expiration or running low.

##### **Search Recipe:**

Users can search for recipes based on available ingredients and apply filters like dietary preferences. The system will display a personalized list of recipe suggestions.

##### **Make Shopping List:**

Users can generate a shopping list by selecting ingredients that are low in stock or needed for recipes.

##### **Make Post:**

Users can create and publish posts in the community forum after content validation by the system.

## **Non-functional requirements**

### **i. Design and Implementation constraints**

The "Ingreedy" application is designed with scalability, security, and usability in mind. The system must handle a growing user base without compromising performance, ensuring smooth functionality even with large datasets of recipes and ingredients. Security protocols like data encryption safeguard user information and pantry details. Additionally, the technology stack integrates AI for OCR and barcode scanning, ensuring seamless ingredient recognition and inventory management.

### **ii. External interfaces required:**

#### **User interfaces**

- The app provides an intuitive, mobile-friendly interface that allows users to easily add ingredients, search recipes, manage their pantry, and engage with the community. The design prioritizes accessibility and ease of navigation to cater to users of all skill levels.

#### **Hardware interfaces**

- Integration with smartphone cameras enables users to scan barcodes and capture images of ingredients for OCR-based recognition. This feature ensures efficient digitization of pantry data, making the app accessible to a wide range of devices.

#### **Software interfaces**

- The application interacts with APIs for fetching recipe data, cloud storage for managing pantry information, and third-party libraries for OCR and barcode functionality. This modular approach ensures the app's robustness and maintainability.

#### **Communication interfaces**

- The app requires a stable internet connection to access cloud-based functionalities like inventory synchronization, personalized recipe suggestions, and community forum interactions. It supports real-time data transfer to keep user information up to date across devices.

## **3.5. Designs**

### **3.5.1. Work Breakdown Structure**

A Work Breakdown Structure (WBS) is a hierarchical framework that breaks down a project into manageable components, providing a clear overview of the project scope and deliverables. It organizes tasks and subtasks in a structured manner, allowing project teams to better plan, allocate resources, and manage risks effectively.

[The WBS for this project can be found in the appendix area.](#)

### 3.5.2. Entity Relation Diagram

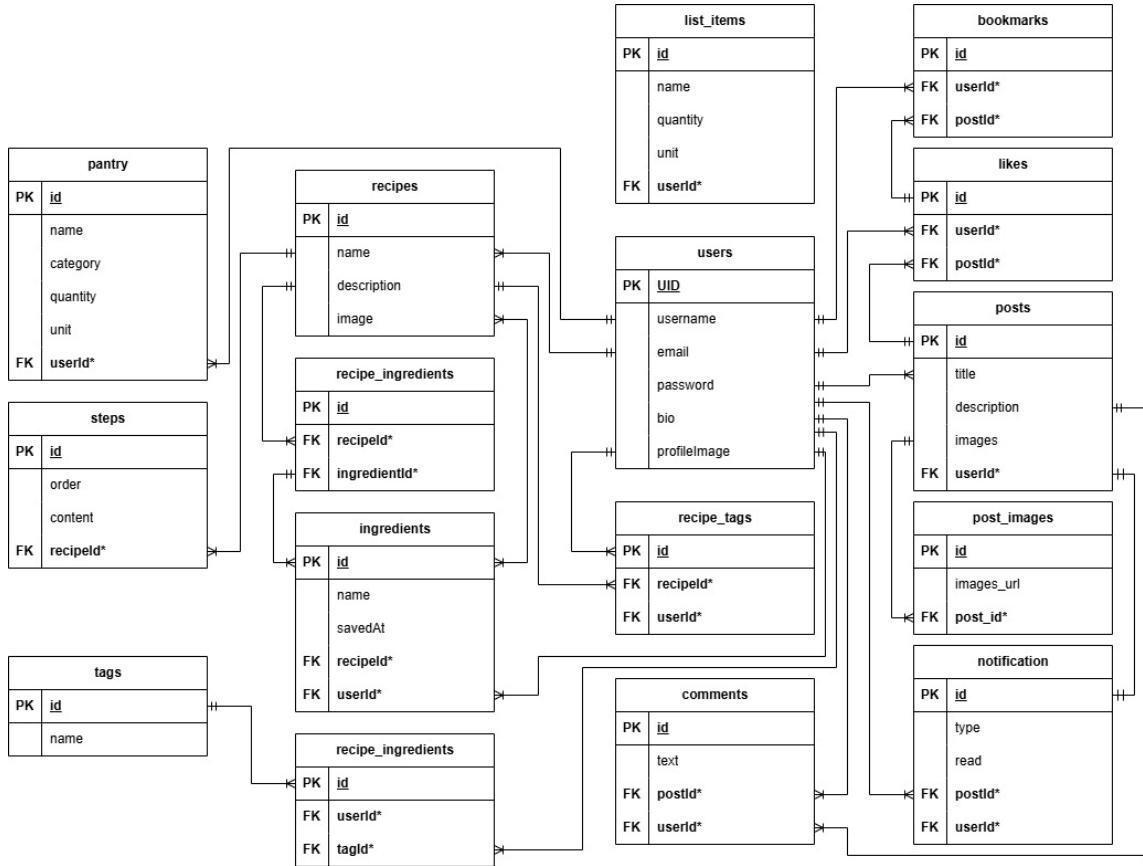


Figure 3: Entity Relationship Diagram

### Entity and their attributes:

#### users

- id (Primary Key)
- username
- email
- password
- bio
- profileImage

## **posts**

- id (Primary Key)
- userId\* (Foreign Key)
- title
- description
- images

## **comments**

- id (Primary Key)
- postId\* (Foreign Key)
- userId\* (Foreign Key)
- text

## **likes**

- id (Primary Key)
- userId\* (Foreign Key)
- postId\* (Foreign Key)

## **bookmarks**

- id (Primary Key)
- userId\* (Foreign Key)
- postId\* (Foreign Key)

## **pantry**

- id (Primary Key)
- userId\* (Foreign Key)
- name
- category
- quantity
- unit

## **recipes**

- id (Primary Key)
- name
- description
- image

## **ingredients**

- id (Primary Key)
- name

## **recipe\_ingredients**

- id (Primary Key)
- recipeId\* (Foreign Key)
- ingredientId\* (Foreign Key)

## **steps**

- id (Primary Key)
- order
- content
- recipeId\* (Foreign Key)

## **search\_history**

- id (Primary Key)
- userId\* (Foreign Key)
- tagId (Foreign Key)

## **tags**

- id (Primary Key)
- name

## **recipe\_tags**

- id (Primary Key)
- recipeId\* (Foreign Key)
- tagId\* (Foreign Key)

## **list\_items**

- id (Primary Key)
- userID\* (Foreign Key)
- name
- quantity
- unit

## **notifications**

- id (Primary Key)
- type
- read
- userId\* (Foreign Key)
- postId\* (Foreign Key)

## **Relationship:**

- A user can have many posts, likes, comments, bookmarks, notifications, list items, pantry items, and search history entries.
- A post belongs to one user and can have many likes, comments, bookmarks, and notifications.
- Comments can have replies (self-relation).
- Notifications link senders and recipients (both users) and optionally relate to a post.
- Recipes have many tags, ingredients, and steps.
- Tags and ingredients can belong to many recipes via join tables.

### **3.5.3. Data Flow Diagram (DFD)**

A Data Flow Diagram (DFD) is a graphical representation of how data flows through a system. It illustrates the processes, data stores, external entities, and data flows within a system. DFDs are used to analyze and design systems by showing how information is input, processed, and output. They are particularly useful for understanding system functionality and identifying inefficiencies. (Valcheva, 2025)

The DFD for this project can be found in the appendix area.

### **3.5.4. Use Case Diagrams**

Use Case Diagrams are a type of behavioral diagram in UML (Unified Modeling Language) that visually represent the interactions between actors (users or external systems) and a system. They focus on the functionality or behavior of the system from the user's perspective, capturing the functional requirements. (creately, 2022)

These diagrams consist of four main components:

**Actors:** Entities that interact with the system, such as users or external systems.

**Use Cases:** Functions or actions within the system, depicted as ovals.

**System Boundary:** A rectangle that defines the scope of the system.

**Relationships:** Connections between actors and use cases, showing interactions. (Fakhroudinov, 2025)

Use Case Diagrams are valuable for understanding system requirements, identifying roles, and clarifying interactions. They are often used in the early stages of system design to communicate with stakeholders. (creately, 2022)

The use case diagram for this project can be found in the appendix area.

### **3.5.5. High-level Expanded Use Case Diagram**

#### **Add Ingredients**

**Use case:** Manage Pantry

**Actor:** User

**Description:** Users add ingredients to the system, which stores their details, including expiry dates. It actively monitors the ingredients and tracks their status. Notifications are sent for items nearing expiration.

#### **Typical Course of Actions:**

<b>Actor Action</b>	<b>System Response</b>
1. Users add ingredients, manually or via OCR scanner.  6. Users receive and review the notifications	2. System stores ingredient data in the database.  3. System tracks expiry dates and stock levels of ingredients using stored data.  4. System detects ingredients nearing expiry or running low.  5. System sends notifications about expiry and low stock.

#### **Domain Class:**

- a) Ingredients
- b) Pantry

## Collaboration diagram:

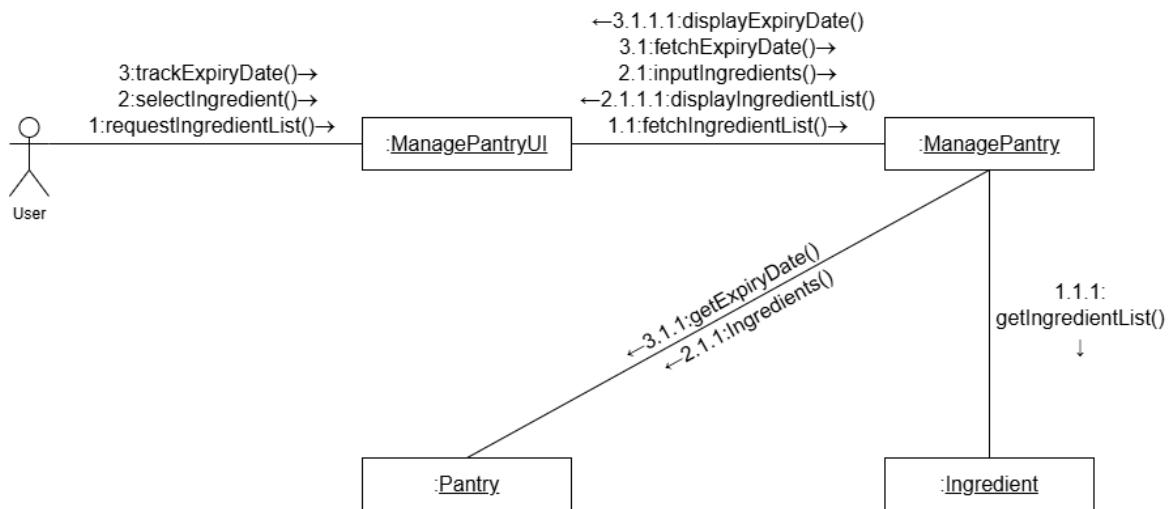


Figure 4: Collaboration diagram of “Manage Pantry”

## Sequence diagram:

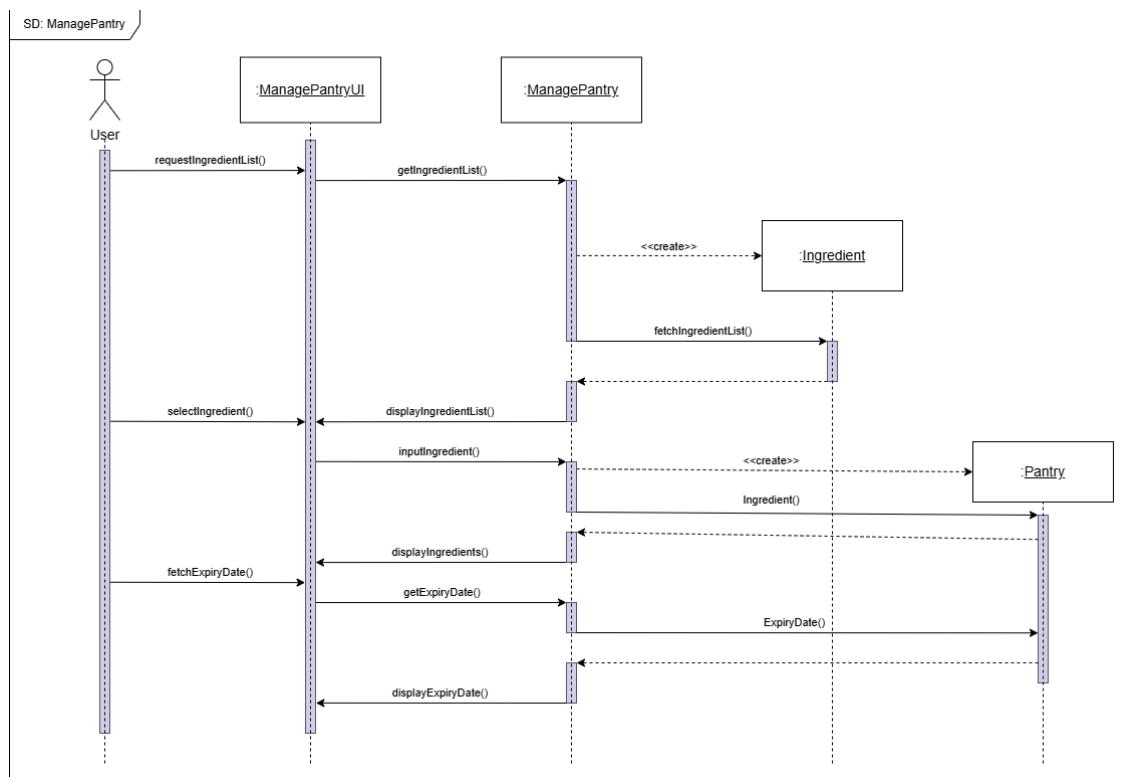


Figure 5: Sequence diagram of “Manage Pantry”

## Search Recipe

**Use case:** Search Recipe

**Actor:** User

**Description:** Users search for recipes, and the system generates a list of results. Users apply filters to narrow down the list. The system updates the list based on the selected filters.

### Typical Course of Actions:

Actor Action	System Response
1. Users navigate to recipe page.  3. Users filter the recipe list based on their needs.	2. System displays recipe list from the database.  4. System displays the filtered recipe list.

### Domain Class:

- a) Recipe

### Collaboration diagram:

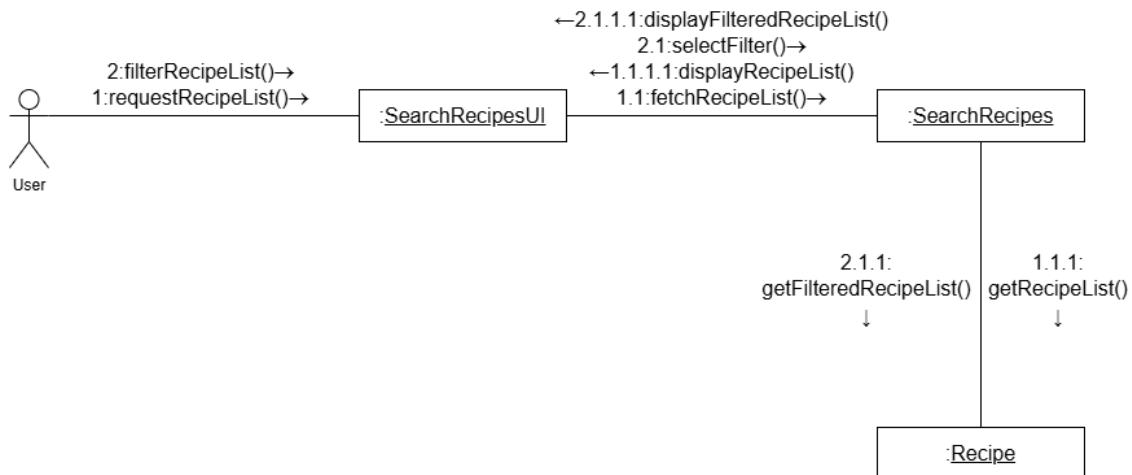


Figure 6: Collaboration diagram of “Search Recipe”

## Sequence diagram:

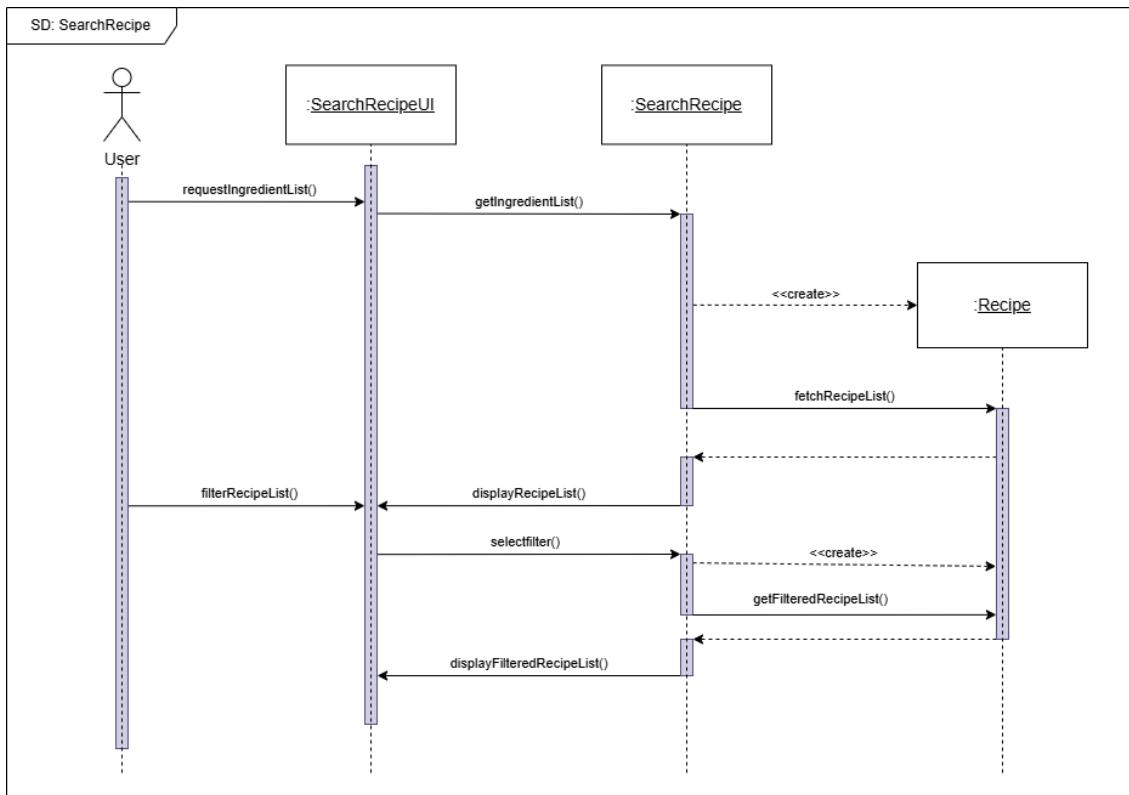


Figure 7: Sequence diagram for “Search Recipe”

## Make Shopping List

**Use case:** Make Shopping List

**Actor:** User

**Description:** Users select ingredients that are low in stock or need to be added for a recipe or that was finished. The system provides an ingredient list for users to select from. The shopping list is then generated.

### Typical Course of Actions:

<b>Actor Action</b>	<b>System Response</b>
1. Users navigate to shopping list page.	2. System displays the user's already existing list.
2. Users add ingredients to the list.	4. System add the newly selected ingredients to the list.

### Domain Class:

- a) Ingredients
- b) Shopping List

### Collaboration diagram:

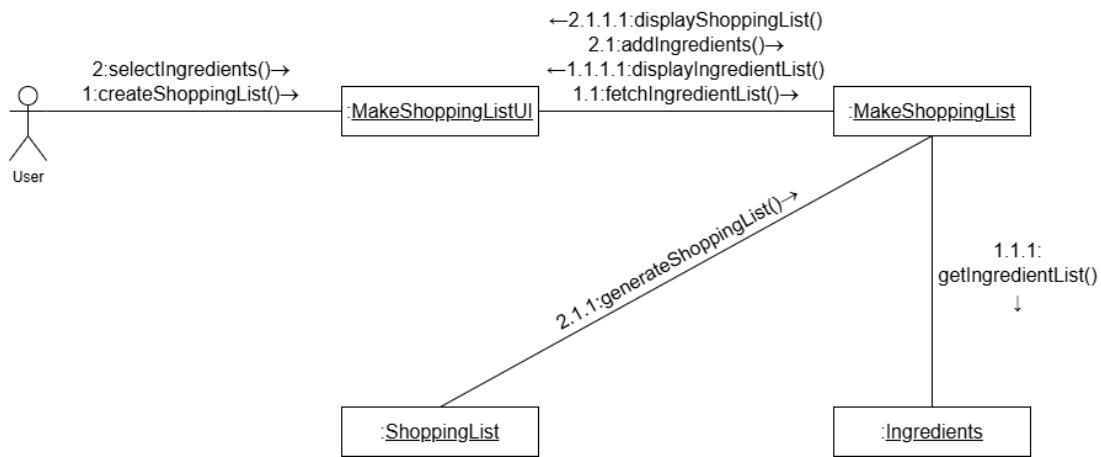


Figure 8: Collaboration of “Make Shopping List”

## Sequence diagram:

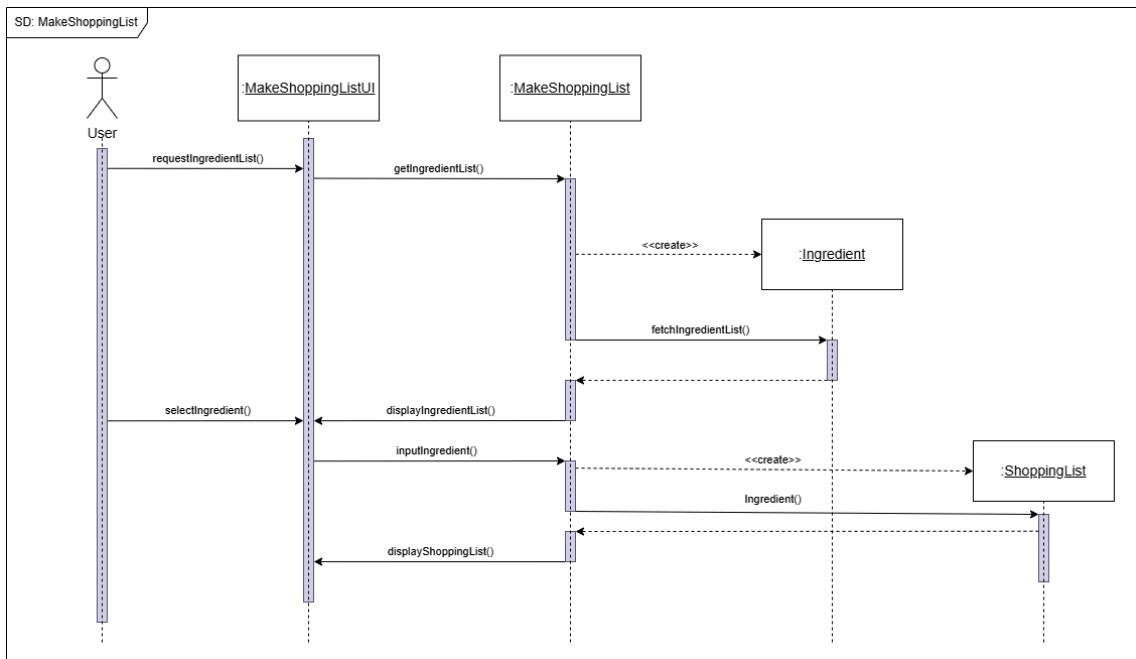


Figure 9: Sequence diagram for “Make Shopping List”

## Make Post

**Use case:** Make Post

**Actor:** User

**Description:** Users input the content for a post, and the system validates it. Once validated, users submit the post. The system then publishes it.

### Typical Course of Actions:

Actor Action	System Response
1. Users inputs the content of the post.	2. System validates the content.
3. Users submit the post.	4. System creates new post and stores it.
	5. System confirms that the post was created.

### Domain Class:

a) Post

### Collaboration diagram:

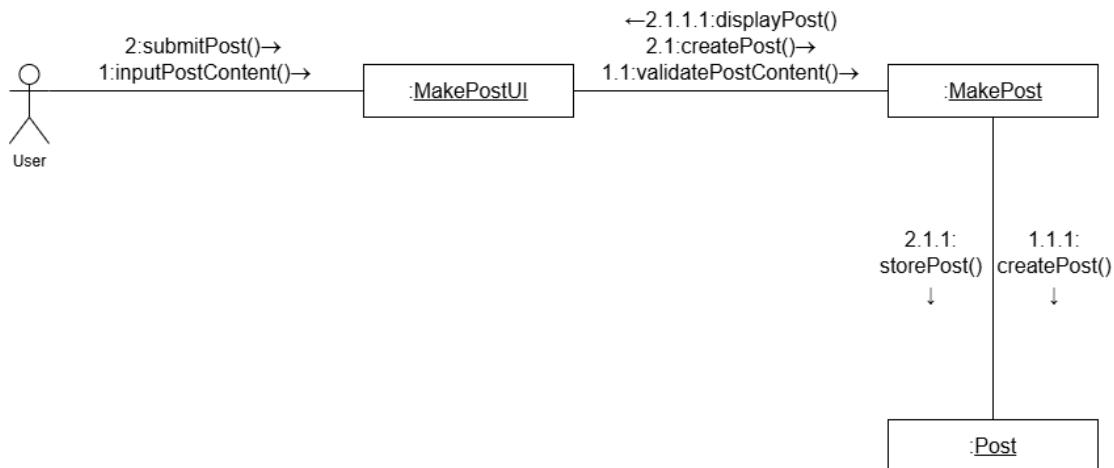


Figure 10: Collaboration of “Make Post”

### Sequence diagram:

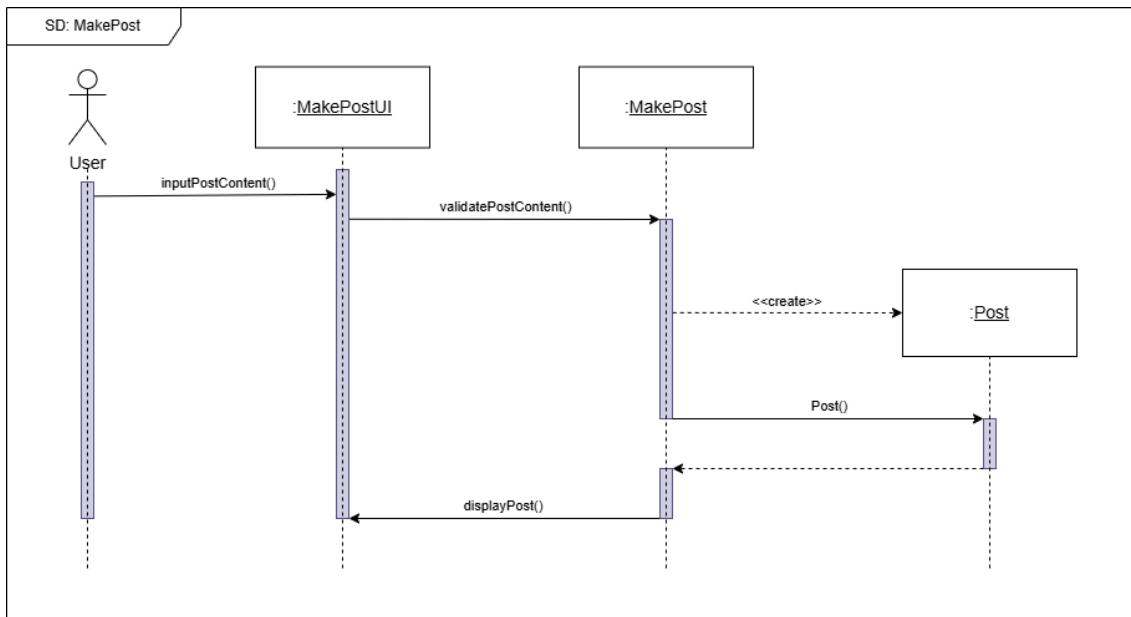


Figure 11: Sequence diagram for “Make Post”

### **3.5.6. Flowchart**

A flowchart is a graphical representation of a process or workflow, using standardized symbols to depict steps, decisions, and the flow of tasks. It is widely used to visualize algorithms, workflows, or processes, making complex systems easier to understand. Flowcharts typically include shapes like rectangles (processes), diamonds (decisions), and arrows (flow direction). (Visual Paradigm, 2025)

[The flowchart diagram for this project can be found in the appendix area.](#)

## 3.6. Implementation

### 3.6.1. Database

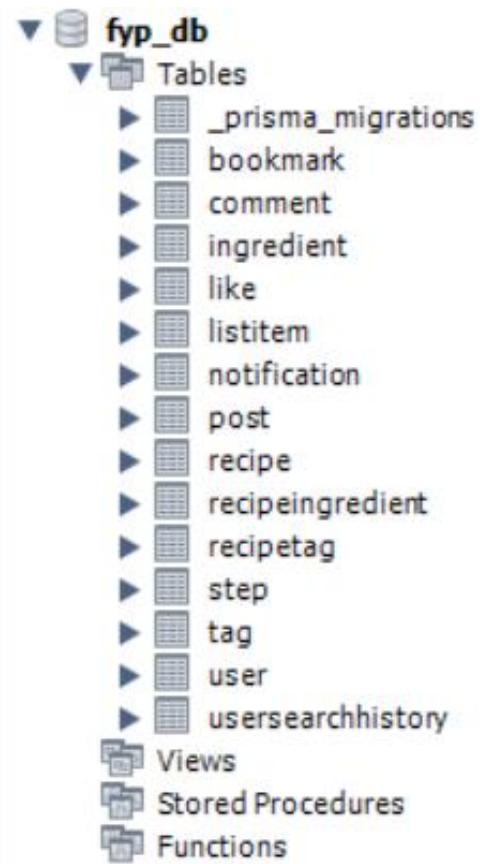


Figure 12: Database Structure

**Database Connection:**

```
generator client {  
  provider = "prisma-client-js"  
  
}  
  
datasource db {  
  provider = "mysql"  
  url    = env("DATABASE_URL")  
  
}
```

```
enum NotificationType {
```

```
  LIKE
```

```
  COMMENT
```

```
  BOOKMARK
```

```
  MENTION
```

```
  SHARE
```

```
}
```

```
model User {
```

```
  id      Int    @id @default(autoincrement())
```

```
  username String @unique
```

```
  email   String @unique
```

```
  password String
```

```

bio          String?      @db.VarChar(500)

profileImage  String?

isPrivate    Boolean     @default(false)

isVerified   Boolean     @default(false)

resetToken   String?

resetTokenExpiry  DateTime?

createdAt    DateTime  @default(now())

updatedAt    DateTime  @updatedAt

verificationCode  String?

codeExpiry   DateTime?

posts        Post[]

likes        Like[]

comments    Comment[]

bookmarks   Bookmark[]

searchHistories UserSearchHistory[]

notifications Notification[] @relation("Notification_recipient")

sentNotifications Notification[] @relation("Notification_sender")

// Relation with shopping list items

listItems    ListItem[]

```

```
}
```

```
model Post {  
    id      Int    @id @default(autoincrement())  
    title   String  @db.VarChar(255)  
    description String @db.VarChar(255)  
    imagePaths Json  
    videoPaths Json?  
    createdAt DateTime @default(now())  
    updatedAt  DateTime @updatedAt  
    userId     Int  
    user       User    @relation(fields: [userId], references: [id])  
    likes      Like[]  
    comments   Comment[]  
    bookmarks  Bookmark[]  
    notifications Notification[]  
}
```

```
model Like {
```

```
    id      Int    @id @default(autoincrement())  
    createdAt  DateTime @default(now())  
    updatedAt  DateTime @updatedAt
```

```

userId    Int
postId    Int
user      User  @relation(fields: [userId], references: [id])
post      Post   @relation(fields: [postId], references: [id], onDelete: Cascade)

@@unique([userId, postId], name: "userId_postId")
}

model Comment {
    id      Int    @id @default(autoincrement())
    text    String
    userId  Int
    postId  Int
    parentId Int?
    user    User   @relation(fields: [userId], references: [id])
    post    Post   @relation(fields: [postId], references: [id], onDelete: Cascade)
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt
}

// Relation fields
replies  Comment[] @relation("CommentReplies") // Child comments
parent   Comment? @relation("CommentReplies", fields: [parentId], references: [id])
}

```

```

model Bookmark {

    id      Int  @id @default(autoincrement())

    userId  Int

    postId  Int

    createdAt  DateTime @default(now())

    updatedAt  DateTime @updatedAt

    user    User  @relation(fields: [userId], references: [id])

    post    Post  @relation(fields: [postId], references: [id], onDelete: Cascade)

    @@unique([userId, postId], name: "unique_bookmark")

}

model Notification {

    id      Int      @id @default(autoincrement())

    type   NotificationType

    read   Boolean   @default(false)

    createdAt  DateTime   @default(now())

    updatedAt  DateTime   @updatedAt

}

// Relationships

```

```

recipient User      @relation("Notification_recipient", fields: [recipientId], references: [id])

recipientId Int

sender User      @relation("Notification_sender", fields: [senderId], references: [id])

senderId Int

post Post?      @relation(fields: [postId], references: [id])

postId Int?

}

```

```

model Recipe {

    id String      @id @default(cuid())

    name String

    imageUrl String

    description String

    ingredients RecipeIngredient[]

    tags RecipeTag[]

    steps Step[]

}

```

```

model Tag {

    id Int      @id @default(autoincrement())

    name String  @unique

```

```
    recipes      RecipeTag[]
    searchHistories  UserSearchHistory[]
}


```

```
model Ingredient {
    id      Int      @id @default(autoincrement())
    name    String   @unique
    recipes  RecipeIngredient[]
}


```

```
model RecipeTag {
    id      Int      @id @default(autoincrement())
    recipe  Recipe  @relation(fields: [recipeId], references: [id])
    recipeId String
    tag     Tag     @relation(fields: [tagId], references: [id])
    tagId   Int
}


```

```
model RecipeIngredient {
    id      Int      @id @default(autoincrement())
    recipe  Recipe  @relation(fields: [recipeId], references: [id])
    recipeId String
    ingredient  Ingredient @relation(fields: [ingredientId], references: [id])
}


```

```

ingredientId Int

}

model Step {
    id      Int  @id @default(autoincrement())
    order   Int
    content String
    recipeId String
    recipe  Recipe @relation(fields: [recipeId], references: [id])
}

model ListItem {
    id      Int  @id @default(autoincrement())
    name   String
    quantity Float
    unit    String
    isChecked Boolean @default(false)
    createdAt DateTime @default(now())
}

// Relation to User (Each user has their own list)

userId Int
user   User @relation(fields: [userId], references: [id])
}

```

```
model UserSearchHistory {  
    id      String  @id @default(cuid())  
    userId  Int  
    tagId   Int  
    createdAt DateTime @default(now())  
  
    user    User @relation(fields: [userId], references: [id])  
    tag     Tag  @relation(fields: [tagId], references: [id])  
}
```

### 3.6.2. Code Implementation

#### User Login

```
export const login = async (req, res) => {
  try {
    const { email, password } = req.body;

    const user = await prisma.user.findFirst({
      where: { email },
    });

    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).json({ message: "Invalid credentials" });
    }

    // ⚡ Create a token
    const token = jwt.sign(
      { id: user.id, email: user.email },
      process.env.JWT_SECRET,
      { expiresIn: '1d' }
    );

    res.status(200).json({
      message: "Login successful",
      token, // Your token is here
      user: {
        id: user.id,
        email: user.email,
        username: user.username,
      }
    });
  } catch (error) {
    console.error("Login error:", error);
    res.status(500).json({ message: "Login failed" });
  }
};
```

## User Registration

```
export const register = async (req, res)=>{
  console.log("✉ Received registration request:", req.body);
  const{ username, email, password, confirmPassword }=req.body;

  if (!email || !username || !password || !confirmPassword) {
    return res.status(400).json({ error: "All fields are required." });
  }
  if (password !== confirmPassword) {
    return res.status(400).json({ error: "Passwords do not match." });
  }
  if (password.length < 6) {
    return res.status(400).json({ error: "Password must be at least 6 characters long." });
  }
  try {
    const existingUser = await prisma.user.findUnique({
      where: { email: email.toLowerCase() },
    });
    if (existingUser) {
      return res.status(400).json({ error: "Email is already in use." });
    }

    const hashedPassword = await bcrypt.hash(password, 10);

    const newUser = await prisma.user.create({
      data: {
        username: username.toLowerCase(),
        email: email.toLowerCase(),
        password: hashedPassword,
      },
    });

    console.log("User registered successfully:", newUser);

    const { password: _, ...userWithoutPassword } = newUser;
    return res.status(201).json({ message: "User registered successfully", user: userWithoutPassword });
  } catch (error) {
    console.error("Registration error:", JSON.stringify(error));
    return res.status(500).json({ error: "Internal Server Error" });
  }
};
```

## Get All Recipe

```
export async function getAllRecipes(req, res) {
  try {
    const recipes = await prisma.recipe.findMany({
      include: {
        tags: { include: { tag: true } },
        ingredients: { include: { ingredient: true } },
        steps: { orderBy: { order: "asc" } },
      },
    });

    const formatted = recipes.map((r) => ({
      id: r.id,
      name: r.name,
      imageUrl: r.imageUrl,
      description: r.description,
      tags: r.tags.map((t) => t.tag.name),
      ingredients: r.ingredients.map((i) => i.ingredient.name),
      steps: r.steps.map((s) => ({ order: s.order, content: s.content })),
    }));
    res.json(formatted);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Failed to fetch recipes" });
  }
}
```

## Scanner

```
Future<void> _pickAndProcessImage() async {
    final pickedFile = await ImagePicker().pickImage(
        source: ImageSource.camera,
    );
    if (pickedFile != null) {
        _image = File(pickedFile.path);
        await _processImage(_image!);
    }
}

Future<void> _processImage(File image) async {
    final inputImage = InputImage.fromFile(image);
    final textRecognizer = TextRecognizer();
    final RecognizedText recognizedText = await textRecognizer.processImage(
        inputImage,
    );
    textRecognizer.close();

    final extractedText =
        recognizedText.text.isNotEmpty ? recognizedText.text : "No text found.";

    if (mounted) {
        Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) => ScanResultsPage(extractedText: extractedText),
            ),
        );
    }
}
```

## Add item to Pantry

```
export const addPantryItem = async (req, res) => {
  const userId = req.user.id;
  const { name, quantity, unit, category } = req.body;

  if (!name || !quantity || !unit) {
    return res.status(400).json({ message: "Missing required fields" });
  }

  try {
    const newItem = await prisma.pantry.create({
      data: {
        name,
        quantity: parseFloat(quantity),
        unit,
        category: category?.trim(), // Optional – Prisma will use default if undefined
        userId,
      },
    });
    res.status(201).json(newItem);
  } catch (error) {
    console.error("Error adding pantry item:", error);
    res.status(500).json({ message: "Failed to add pantry item", error: error.message });
  }
};
```

## Create and store posts

```
export const addPost = async (req, res) => {
  try {
    const { title, description } = req.body;
    const files = req.files;
    const userId = req.user?.id;

    console.log("User:", req.user); // ⚡ Add this
    console.log("Token User ID:", userId); // ⚡ Add this
    console.log("Request Body:", req.body);

    if (!userId) {
      return res.status(401).json({ message: "User ID is not authenticated" });
    }

    const imagePaths = files?.['image']
      ? files['image'].map(file => `/uploads/postImages/${file.filename}`)
      : [];
    const videoPaths = files?.['video']
      ? files['video'].map(file => `/uploads/postVideos/${file.filename}`)
      : [];

    const newPost = await prisma.post.create({
      data: {
        title: title.trim(),
        description: description.trim(),
        imagePaths,
        videoPaths,
        userId: parseInt(userId),
      },
    });

    res.status(201).json({
      message: 'Post added successfully',
      post: newPost
    });
  } catch (error) {
    console.error("Add post error:", error);
    res.status(500).json({ message: 'Something went wrong' });
  }
};
```

## User Profile

```
export const getProfile = async (req, res) => {
  try {
    const userId = req.user.id;

    // Fetch user with own posts & bookmarks
    const user = await prisma.user.findUnique({
      where: { id: userId },
      include: {
        posts: {
          include: {
            user: { select: { id: true, username: true, profileImage: true } },
            likes: { select: { id: true } },
            _count: { select: { likes: true, comments: true } },
          },
          orderBy: { createdAt: "desc" },
        },
        bookmarks: {
          include: {
            post: {
              include: {
                user: { select: { id: true, username: true, profileImage: true } }
              },
              likes: { select: { id: true } },
              _count: { select: { likes: true, comments: true } },
            },
          },
        },
        orderBy: { createdAt: "desc" },
      },
    });
  }

  if (!user) {
    return res.status(404).json({ message: "User not found" });
  }

  // Format own posts
  const posts = user.posts.map((p) => ({
    id: p.id,
    user: p.user,
    description: p.description,
    imagePaths: p.imagePaths || [],
    videoPaths: p.videoPaths || [],
  })
}
```

```

    mediaPaths: [...(p.imagePaths || []), ...(p.videoPaths || [])],
    createdAt: p.createdAt,
    likes: p._count.likes,
    comments: p._count.comments,
    hasLiked: p.likes.some((like) => like.id === userId),
    isBookmarked: false,
  });

  // Format saved posts
  const savedPosts = user.bookmarks.map((b) => ({
    id: b.post.id,
    user: b.post.user,
    description: b.post.description,
    imagePaths: b.post.imagePaths || [],
    videoPaths: b.post.videoPaths || [],
    mediaPaths: [...(b.post.imagePaths || []), ...(b.post.videoPaths || [])],
    createdAt: b.post.createdAt,
    likes: b.post._count.likes,
    comments: b.post._count.comments,
    hasLiked: b.post.likes.some((like) => like.id === userId),
    isBookmarked: true,
  }));

  res.json({
    id: user.id,
    username: user.username,
    profileImage: user.profileImage,
    posts,
    savedPosts,
  });
} catch (error) {
  console.error("Profile fetch error:", error);
  res.status(500).json({ message: "Internal server error" });
}
};

```

## **Chapter 4: Testing and Analysis**

### **4.1. Test Plan**

The testing approach for the Ingreedy (Smart Recipe Finder) application was focused primarily on two areas: System Testing and API Testing. These testing methods ensured both the application's functional reliability and backend communication accuracy.

### **4.2. System Testing**

System testing was performed to validate the integrated application as a whole, ensuring all modules work together as intended on Android devices.

#### **Objectives:**

- Verify that all features (pantry management, recipe search, shopping list generation, community posting) work as expected.
- Ensure compatibility across devices and screen sizes.
- Validate user interface (UI) flow and overall user experience

#### **Result Summary:**

- **Pass Rate:** 95% of test cases passed successfully.
- **Issues Found:** Minor UI alignment issues on smaller screens and occasional OCR delays.
- **Actions Taken:** Adjusted UI scaling and added image preprocessing for OCR.

#### 4.2.1.Authentication Testing

Table 2: Registration test case

Objective	Verify that a new user can successfully register and is redirected to the login page upon completion.
Action	Fill in the required fields (username, email, and password), then click the Register button.
Expected result	Upon successful registration, the user should get a success message.
Actual result	After registration, the user successfully received a successful message.
Conclusion	The test is successful.

**Proof:**

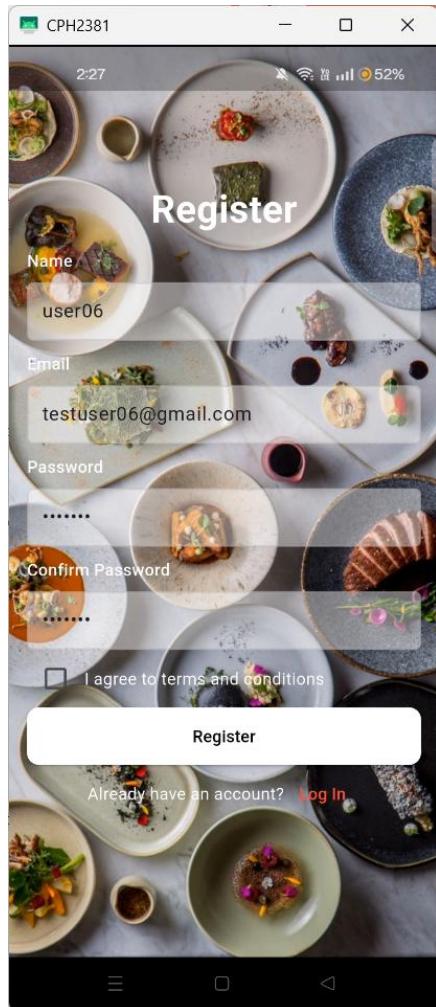


Figure 13: Registering new user

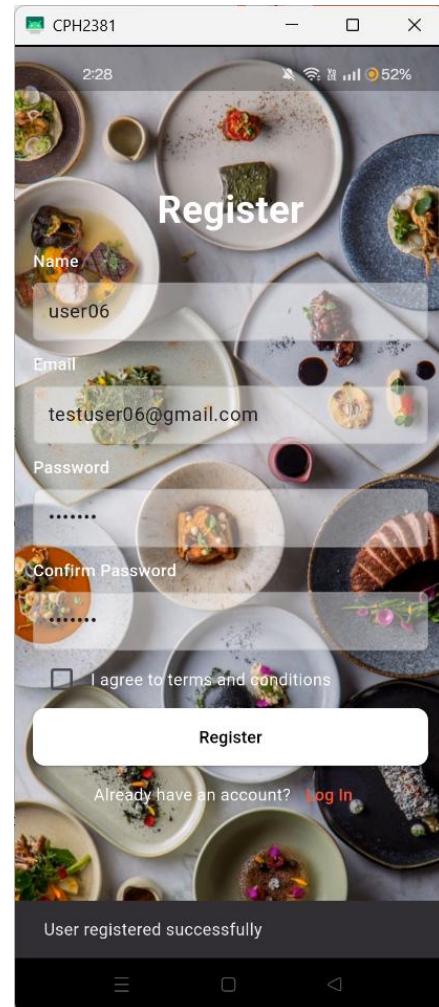


Figure 14: Login page

Table 3: Login test case

Objective	Confirm that a registered user can log in and gain access to the application.
Action	Enter valid login credentials (email and password), then click the Login button.
Expected result	Upon successful authentication, the user should be redirected to the home page.
Actual result	The user was successfully authenticated and redirected to the home page.
Conclusion	The test is successful.

**Proof:**

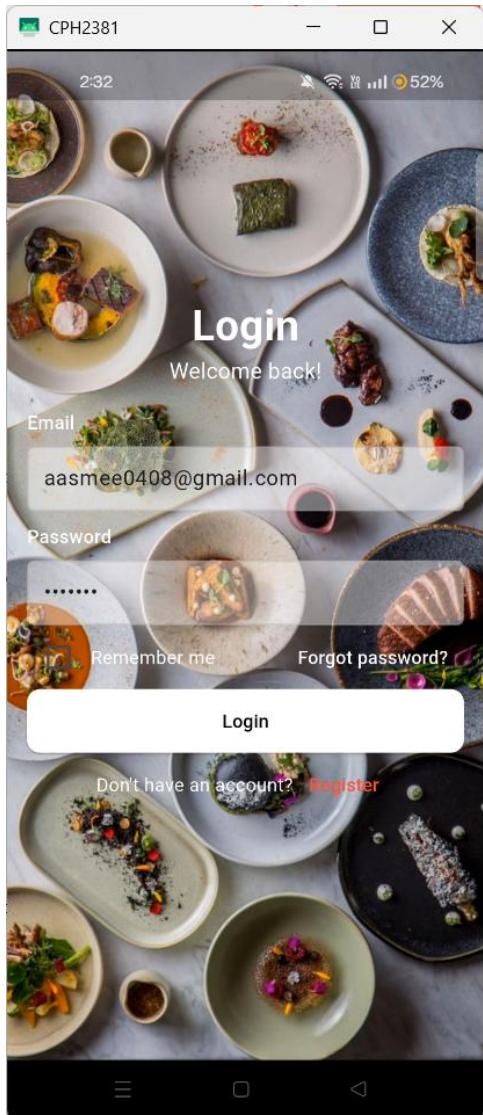


Figure 15: Entering login credentials

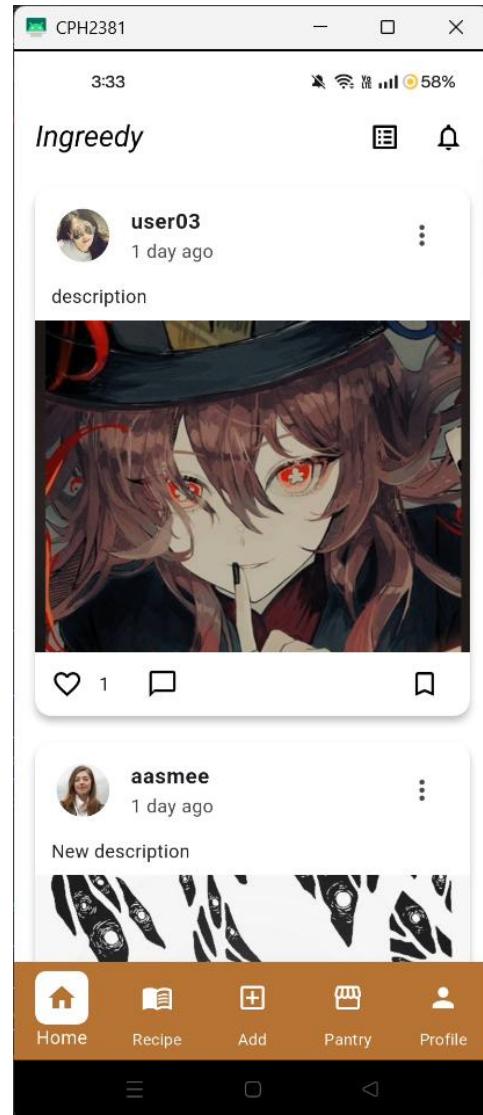


Figure 16: Home page

#### 4.2.2.Pantry Testing

Table 4: Adding pantry item test case

Objective	To verify the functionality of adding a new ingredient to the pantry.
Action	Navigate to the Pantry page, click the "Add Ingredients" button, enter the required ingredient details, and confirm the addition.
Expected result	The newly added ingredient should appear on the Pantry page, correctly categorized under its respective section
Actual result	The ingredient appeared on the Pantry page as expected, under the correct category.
Conclusion	The test is successful.

**Proof:**

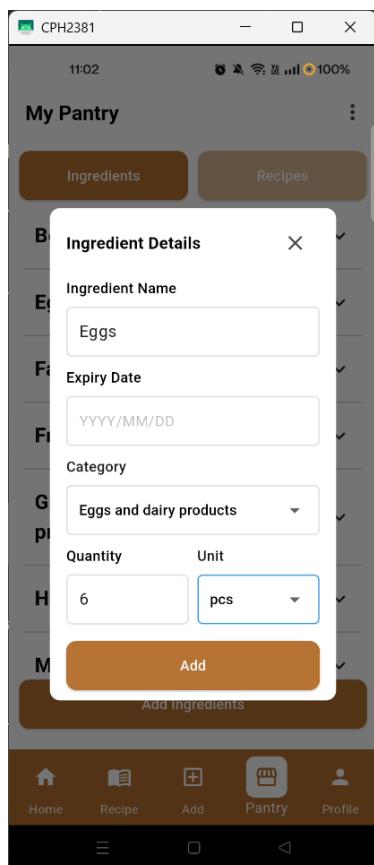


Figure 17: Adding item to pantry

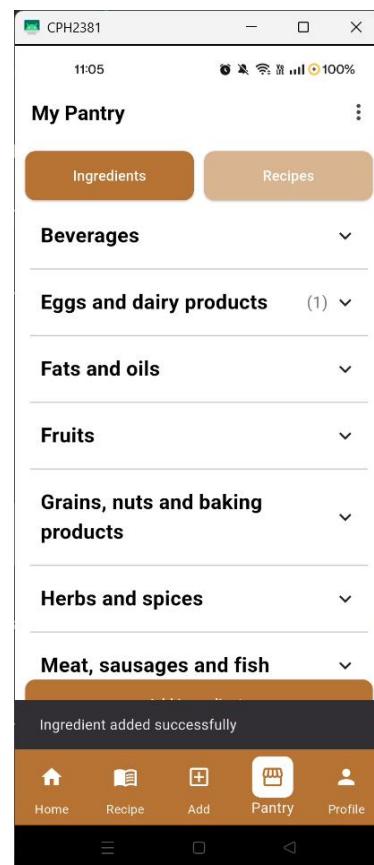


Figure 18: Succession message

#### 4.2.3.Post Testing

Table 5: Creating post test case

Objective	To verify that a new post can be successfully created, stored in the database, and retrieved for display on the homepage.
Action	Select preferred images, write a caption, click the “POST” button, and then navigate to the homepage.
Expected result	The new post should be saved in the database and correctly displayed on the homepage.
Actual result	The post was successfully created, stored in the database, and displayed on the homepage.
Conclusion	The test is successful.

**Proof:**

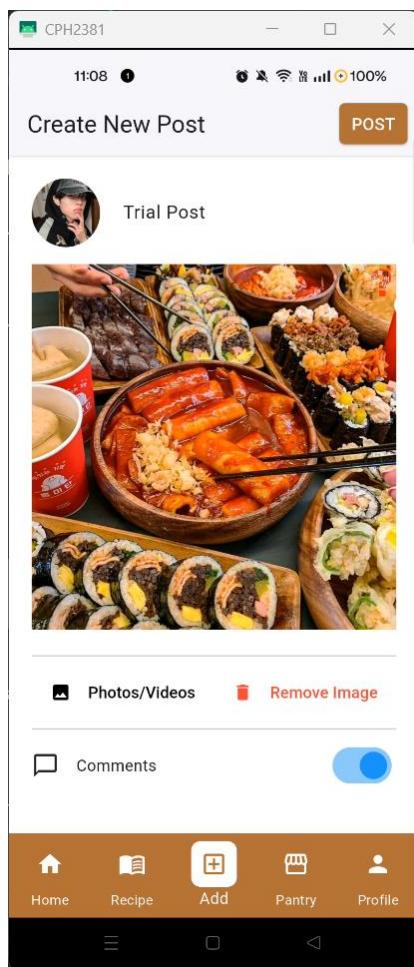


Figure 19: Creating a new post

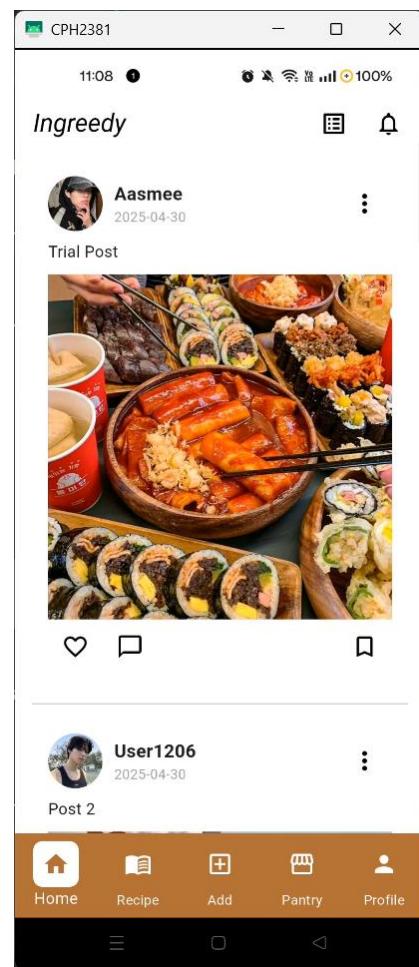


Figure 20: New post on home page

Table 6: Making comments test case

Objective	To verify that users can successfully post a comment and have it displayed correctly.
Action	Tap the chat button, enter a comment, and submit it by clicking “Post”.
Expected result	The newly submitted comment should appear at the top of the comment section.
Actual result	The new comment appeared at the top of the comment section as expected.
Conclusion	The test is successful.

**Proof:**

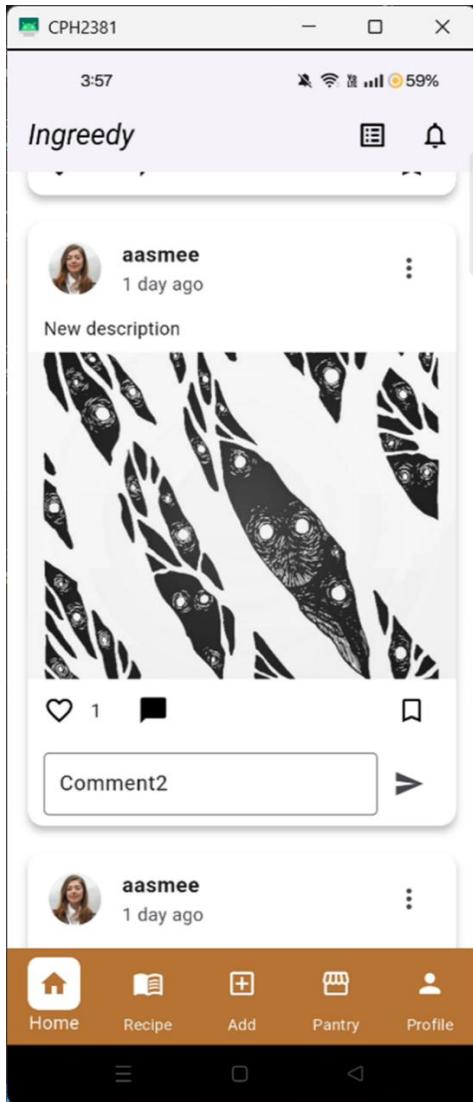


Figure 21: Comments box

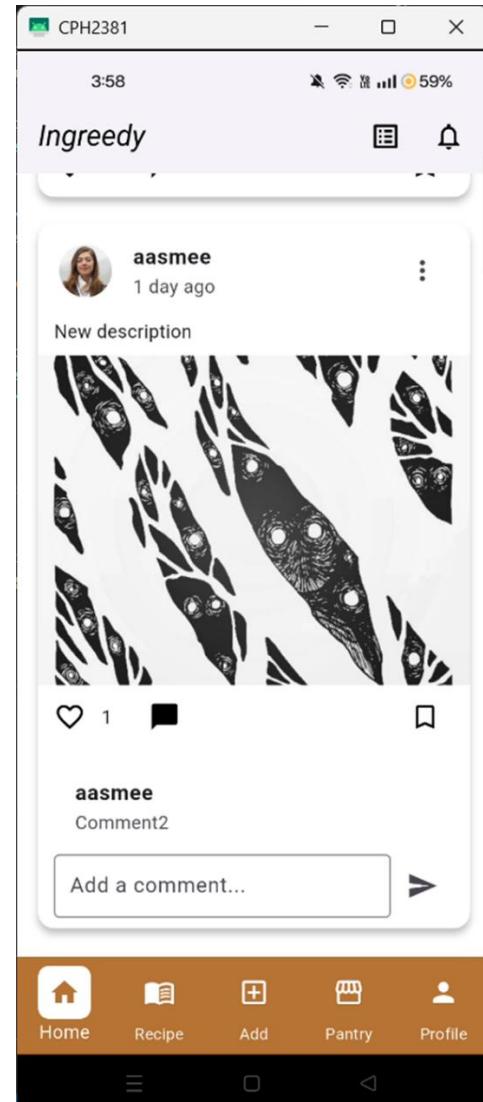


Figure 22: New comment added

Table 7: Liking post test case

Objective	To verify that users can successfully like a post
Action	Tap the “Like” button on the desired post.
Expected result	The heart icon should change to a red filled heart, and the like count should increase by one.
Actual result	The heart icon changed to a red filled heart, and the like count increased by one.
Conclusion	The test is successful.

### Proof:

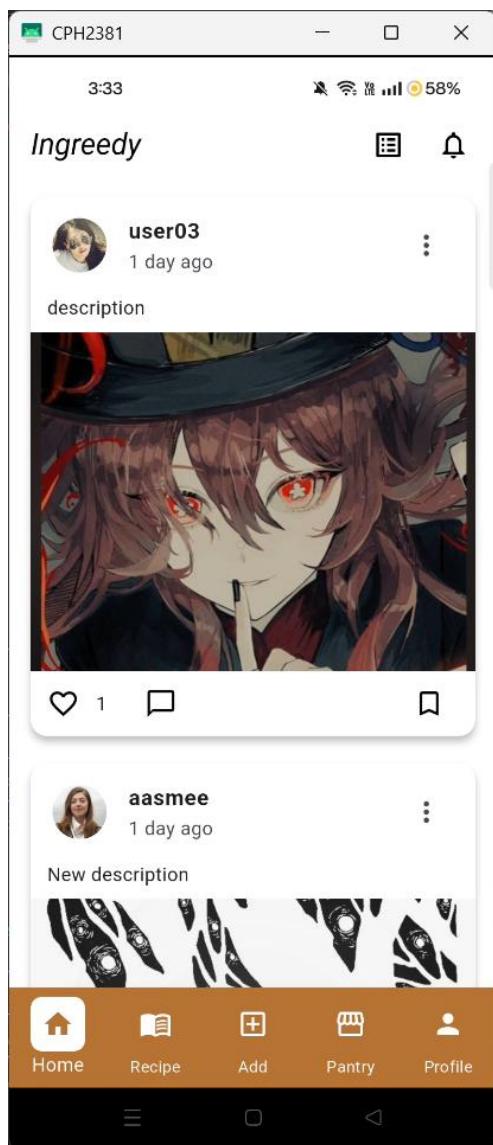


Figure 23: Post before liking

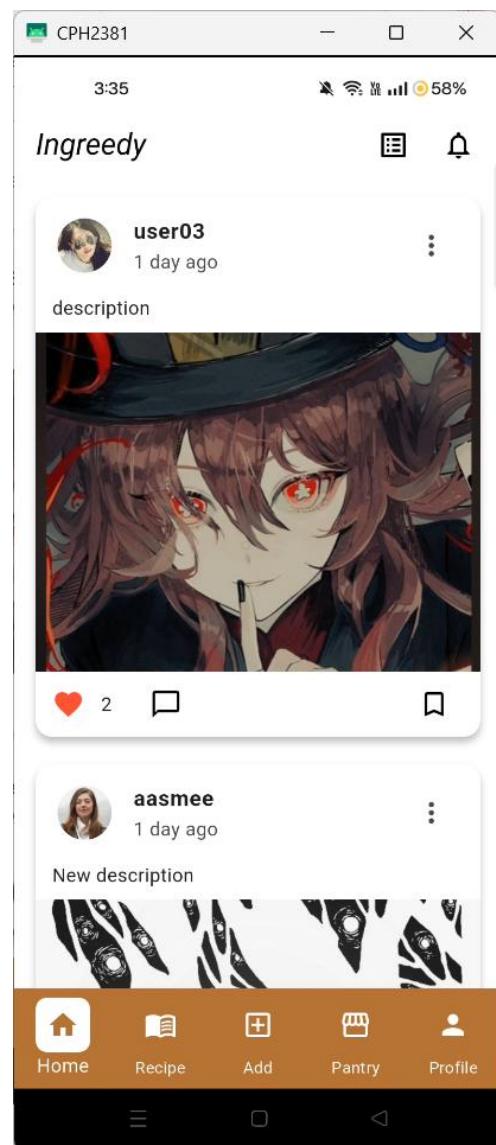


Figure 24: Post after like

Table 8: Bookmarking posts test case

Objective	To verify that users can successfully bookmark a post.
Action	Tap the “Bookmark” button on the desired post, then navigate to the profile page and select the “Saved” section to verify the bookmark.
Expected result	The bookmarked post should appear in the “Saved” section of the profile.
Actual result	The bookmarked post appeared correctly in the “Saved” section of the profile.
Conclusion	The test is successful.

**Proof:**

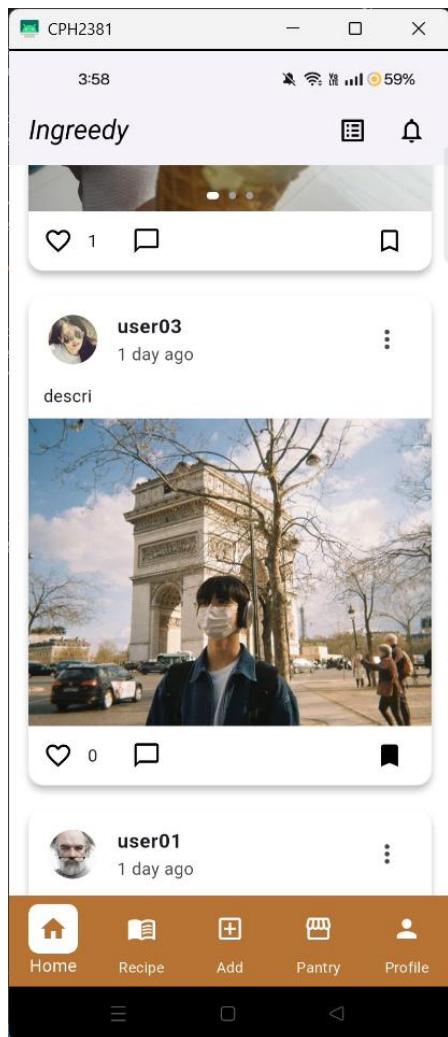


Figure 25: Bookmarking post

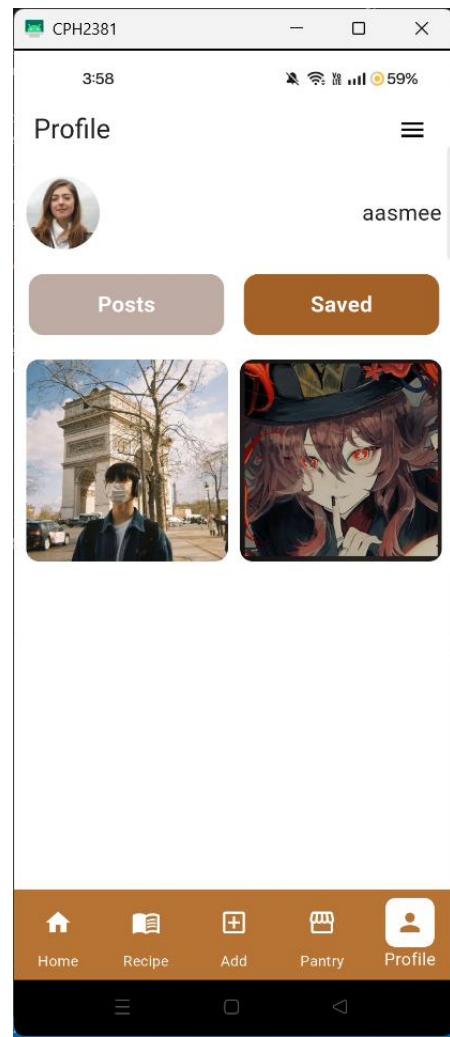


Figure 26: Bookmarked post in saved section

#### 4.2.4.Shopping List Testing

Table 9: Adding item to list test case

Objective	To verify that users can successfully create a new list item.
Action	Tap the “+ Add to List” button and enter the required information.
Expected result	The newly added item should appear in the list below.
Actual result	The newly added item appeared correctly in the list below.
Conclusion	The test is successful.

**Proof:**

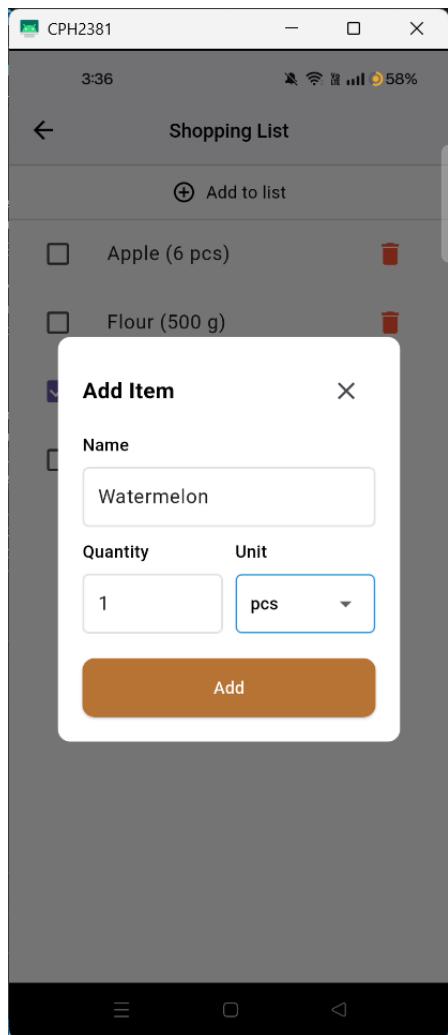


Figure 27: Adding item to list

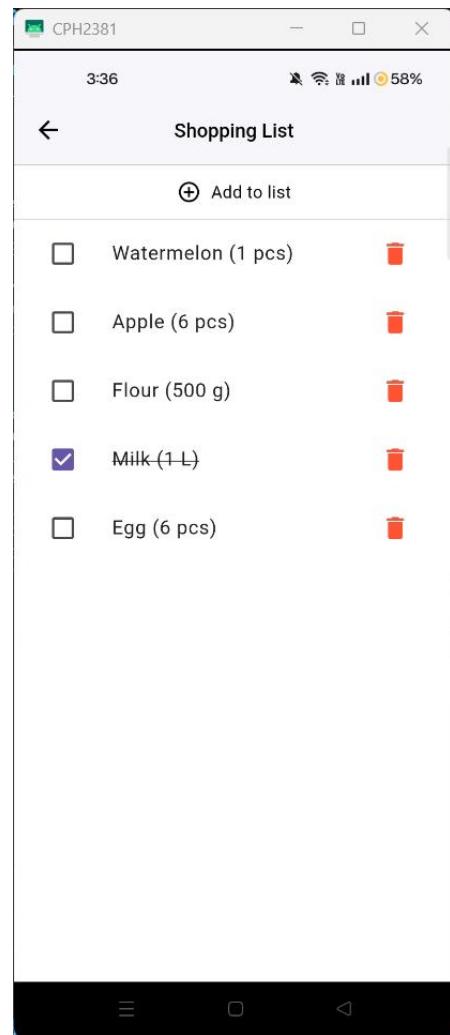


Figure 28: List with added item

Table 10: Deleting list item test case

Objective	To verify that users can successfully delete an item from the list.
Action	Tap the delete icon corresponding to the desired item.
Expected result	The selected item should be removed and no longer appear in the list.
Actual result	The selected item was removed and no longer appeared in the list.
Conclusion	The test is successful.

**Proof:**

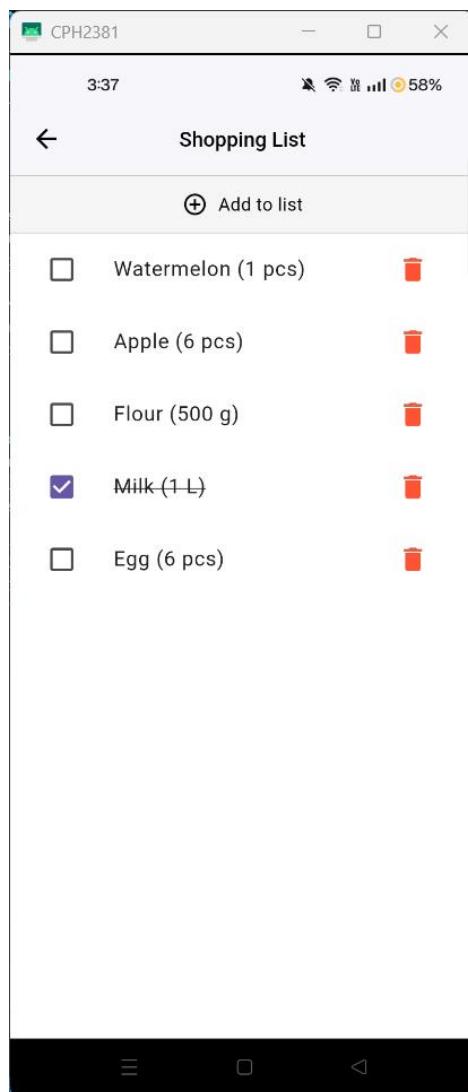


Figure 29: Shopping list before deleting item

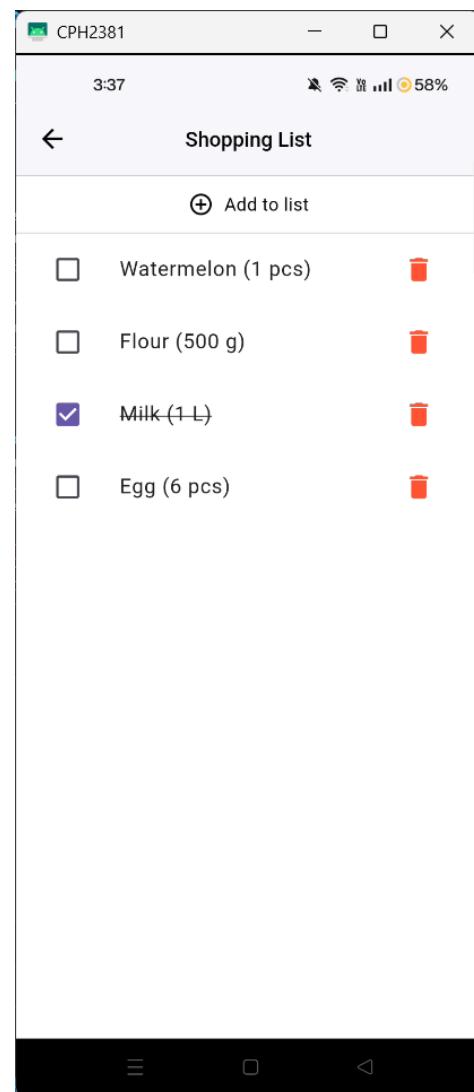


Figure 30: Shopping list after deleting the item

#### 4.2.5.Recipe Testing

*Table 11: Filtering recipe list by tag test case*

Objective	To verify that users can filter the recipe list based on selected tags.
Action	Tap on any recipe tag to apply the filter.
Expected result	A filtered list displaying only recipes associated with the selected tag should be generated.
Actual result	The filtered list displaying recipes with the selected tag was successfully generated.
Conclusion	The test is successful.

### **4.3. API testing**

API testing was conducted to ensure that backend services work correctly and return the expected responses. Postman was used for testing all API endpoints.

#### **Objectives:**

- Validate endpoints for ingredient management, recipe fetching, shopping list creation, and user authentication.
- Ensure response times meet performance requirements.
- Verify error handling and response codes.

#### **Scope:**

- POST /user/login – User authentication.
- GET /recipe – Fetching all recipes.
- POST /pantry – Adding ingredients.
- GET /bookmark – Retrieving saved recipes.
- PATCH /notifications/:id/read – Updating notification status.

#### **Result Summary:**

- **Pass Rate:** 98% of API endpoints passed functional tests.
- **Issues Found:** Occasional timeout when fetching large recipe data sets on unstable networks.
- **Action Taken:** Implemented request optimizations and caching for frequent API calls.

#### 4.3.1. Register

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, user profile, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar shows several recent requests: POST Pos, POST Log, GET http://, POST Reg, POST http, and GET. A message 'No environment' is displayed. The main workspace is titled 'Re-FYP / Register'. A POST request is selected, with the URL 'http://localhost:3001/user/register' entered. The 'Body' tab is active, showing JSON input:

```
1 {  
2   "username": "user05",  
3   "email": "testuser05@gmail.com",  
4   "password": "pass123",  
5   "confirmPassword": "pass123"  
6 }
```

Below the body, the response is shown under the 'Body' tab, indicating a '201 Created' status with a response time of 431 ms and a response size of 699 B. The response JSON is:

```
1 {  
2   "message": "User registered successfully",  
3   "user": {  
4     "id": 6,  
5     "username": "user05",  
6     "email": "testuser05@gmail.com",  
7     "bio": null,  
8     "profileImage": null,  
9     "isPrivate": false,  
10    "isVerified": false,  
11    "resetToken": null,  
12    "resetTokenExpiry": null,  
13    "createdAt": "2025-07-31T14:13:02.129Z",  
14    "updatedAt": "2025-07-31T14:13:02.129Z",  
15    "verificationCode": null,  
16  }  
17}
```

At the bottom, there are buttons for Console, Postbot, Runner, and Vault.

Figure 31: Register API testing

#### 4.3.2. Login

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, add, settings, and upgrade. A red 'Upgrade' button is prominent. Below the header, a navigation bar shows a lock icon, a back arrow, and several recent requests: POST Pos (red), POST Log (red), GET http:// (red), POST Reg (red), POST http (red), and GET (green). To the right of these is a dropdown for 'No environment' and a close button.

The main workspace displays a POST request to 'Re-FYP / Login'. The URL is set to 'http://localhost:3001/user/login'. The method is 'POST' and the status is 'Send'. Below the URL, there are tabs for Params, Auth, Headers (9), Body (selected), Scripts, and Settings. The Body tab shows the JSON payload:

```
1 {  
2   "email": "aasmee0408@gmail.com",  
3   "password": "pass123"  
4 }
```

Below the request details, the response section shows a green '200 OK' status with a response time of 437 ms and a size of 642 B. The response body is displayed in JSON format:

```
{ } JSON ▾ ▶ Preview ⚡ Visualize ▾
```

```
1 {  
2   "message": "Login successful",  
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpZCI6MSwiZW1haWwiOiJhYXNtZWUwNDA4QGdtYWlsLmVbSIsImhdCI6MTc1Mzk2MjI4MCwiZX  
hwIjoxNzU0MDQ4NjgwfQ.0WDkyCzDWLybjfaXD6bWmX08Bya6n2X7MuqTigew8m0",  
4   "user": {  
5     "id": 1,  
6     "email": "aasmee0408@gmail.com",  
7     "username": "aasmee"  
8   }  
9 }
```

At the bottom of the interface, there are buttons for Console, Postbot, Runner, Vault, and a help icon.

Figure 32: Login API testing

### 4.3.3. Make post

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, add, settings, and upgrade. The main header shows a POST request to 'Re-FYP / Posts' with the URL `http://localhost:3001/post/create`. Below the header, there are sections for Params, Auth, Headers (9), Body (selected), Scripts, and Settings. The Body section is set to 'form-data' and contains three fields: 'title' (Text, value: 'post9'), 'description' (Text, value: 'description'), and 'image' (File, value: '2023-02-27.png'). To the right of the body fields is a table for 'Bulk Edit'. Below the body section, the response status is shown as '201 Created' with a timestamp of '483 ms' and a size of '617 B'. The response body is displayed as JSON:

```
1 {  
2   "message": "Post added successfully",  
3   "post": {  
4     "id": 9,  
5     "title": "post9",  
6     "description": "description",  
7     "imagePaths": [  
8       "/uploads/postImages/1753971485400.png"  
9     ],  
10    "videoPaths": [],  
11    "createdAt": "2025-07-31T14:18:05.490Z",  
12    "updatedAt": "2025-07-31T14:18:05.490Z",  
13    "userId": 4  
14  }  
15 }
```

At the bottom of the interface, there are buttons for Console, Postbot, Runner, Vault, and other tools.

Figure 33: Post creation API testing

#### 4.3.4. Get post

The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, API Network, and a search bar labeled 'Search Postman'. On the right side of the header, there are buttons for 'Invite', 'Upgrade', and environment management.

The main workspace displays a collection named 'http://localhost:3001/post/get'. A GET request is selected with the URL 'http://localhost:3001/post/get'. The 'Params' tab is active, showing a single query parameter 'Key' with the value 'Value'. The 'Headers' tab shows six headers: 'Content-Type: application/json', 'Accept: application/json', 'User-Agent: PostmanRuntime/7.31.0', 'Host: localhost:3001', 'Connection: keep-alive', and 'Postman-Token: 6333e3f3-1a2d-4a2c-8a20-1a2a2a2a2a2a'. The 'Body' tab is selected, showing a JSON response with 10 items. The 'Comments' tab is also visible.

	<b>id</b>	<b>title</b>	<b>description</b>	<b>imagePaths</b>	<b>videoPaths</b>	<b>createdAt</b>	<b>likes</b>	<b>hasLiked</b>	<b>isBookmarked</b>	<b>user</b>	<b>comments</b>
0	9	post9	description	0 /uploads/postImages/1753971485400.png	[ ]	2025-07-31T14:18:05.490Z	0	false	false	id: 4 username: user03	[ ]
1	8	post8	desc	0 /uploads/postImages/1753960608459.jpg	[ ]	2025-07-31T11:16:48.484Z	0	false	false	id: 1 username: aasmee	[ ]
2	7	post7	des	0 /uploads/postImages/1753960309422.jpg 1 /uploads/postImages/1753960310040.jpg 2 /uploads/postImages/1753960310105.jpg	[ ] [ ] [ ]	2025-07-31T11:15:13Z	0	false	false	id: 1 username: aasmee	[ ]
3	6	post6	descript	0 /uploads/postImages/1753945927699.jpg 1 /uploads/postImages/1753945927700.jpeg	[ ] [ ]	2025-07-31T07:12:07.721Z	0	false	false	id: 4 username: user03	[ ]
4	5	post5	descri	0 /uploads/postImages/1753945872559.jpeg	[ ]	2025-07-31T07:11:12.613Z	0	false	false	id: 4	[ ]

At the bottom of the interface, there are various status indicators and links: 'Online' (green), 'Find and replace' (blue), 'Console' (grey), 'Postbot' (blue), 'Runner' (grey), 'Start Proxy' (blue), 'Cookies' (blue), 'Vault' (blue), 'Trash' (blue), and a help icon (grey).

Figure 34: Get post API testing

#### 4.3.5. Delete post

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, refresh, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar shows recent requests: 'Save' (GET Recip), 'GET Profil', 'POST Lis', and 'DEL http://'. The current request is 'http://localhost:3001/post/delete/6', indicated by a red underline.

The main area shows a 'DELETE' request being sent to the same URL. The 'Auth' tab is selected, showing 'Bearer Token' as the type and a token value displayed in a yellow-bordered box. Other tabs include Headers (9), Body, Scripts, and Settings. A note below the auth section states: 'The authorization header will be automatically generated when you send the request. Learn'.

At the bottom, the response is shown as a 200 OK status with a response time of 107 ms and a size of 392 B. The response body is displayed in JSON format:

```
1 {  
2   "message": "Post deleted successfully"  
3 }
```

Figure 35: Deleting post API testing

#### 4.3.6. Update posts

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, refresh, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar shows recent requests: POST Log, POST Save, GET Recip, and PUT http://. The current request is a PUT to <http://localhost:3001/post/update/8>. The request details panel shows the method as PUT and the URL as http://localhost:3001/post/update/8. The 'Body' tab is selected, showing the JSON payload:

```
1 {
2   "description": "New description"
3 }
```

The response panel shows a green '200 OK' status with a response time of 195 ms and a response size of 1 KB. The response body is displayed in JSON format:1 {
2 "message": "Post updated successfully",
3 "post": {
4 "id": 8,
5 "title": "post8",
6 "description": "New description",
7 "imagePaths": [
8 "/uploads/postImages/1753960608459.jpg"
9 ],
10 "videoPaths": [],
11 "createdAt": "2025-07-31T11:16:48.484Z",
12 "updatedAt": "2025-07-31T15:25:28.381Z",
13 "userId": 1,
14 "user": {
15 "id": 1,
16 "username": "aasmee",
17 "email": "aasmee0408@gmail.com",
18 "password": "\$2b\$10\$V1XZJ04NV1H0672Xey69VukQvUt3RVlg0fhzDFHiS3FjqI.K3Ce20",
19 "bio": null,
}

At the bottom, there are buttons for Postbot, Runner, Vault, and help.

Figure 36: Updating posts API testing

#### 4.3.7. Like post

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, refresh, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar shows several recent requests: 'Pos' (red), 'POST Log' (red), 'POST Save' (green), 'GET Recip' (red), and 'POST http' (red). The current request is highlighted with a red border and shows the URL 'http://localhost:3001/post/8/like'. To the right of the URL are 'Save' and 'Share' buttons, and a copy icon.

The main workspace contains a POST request to 'http://localhost:3001/post/8/like'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "success": true,  
3   "liked": true,  
4   "likeCount": 1  
5 }
```

Below the request, the response details are shown: '200 OK', '115 ms', '396 B', and a globe icon indicating the response is local. The response body is identical to the request body, showing the JSON object with 'success', 'liked', and 'likeCount' fields.

At the bottom of the interface, there are icons for 'Console', 'Postbot', 'Runner', and other vault-related functions.

Figure 37: Liking post API testing

#### 4.3.8. Bookmark posts

The screenshot shows the Postman application interface. At the top, there are navigation links for 'Workspaces' and 'More', along with various status indicators and a 'Upgrade' button. Below the header, a search bar and a toolbar with icons for copy, paste, settings, and notifications are visible. The main workspace displays a list of recent requests: 'Pos' (status red), 'POST Log' (status red), 'POST Save' (status green), 'GET Recip' (status red), 'POST http' (status red), and the current request, 'POST http://localhost:3001/post/8/bookmark' (status red). To the right of the list are buttons for 'Save' (with a dropdown arrow), 'Share', and a link icon.

The request details panel shows the method as 'POST' and the URL as 'http://localhost:3001/post/8/bookmark'. Below this, tabs for 'Params', 'Auth', 'Headers (9)', 'Body', 'Scripts', and 'Settings' are present, with 'Body' being the active tab. Under 'Body', the content type is set to 'raw' and then switched to 'JSON'. The JSON payload is defined as:

```
1 {  
2   "bookmarked": true  
3 }
```

On the right side of the body panel, there are 'Beautify' and 'Copy' buttons. Below the body panel, the response section shows a green status bar indicating '201 Created' with a timestamp of '53 ms' and a size of '405 B'. The response body is displayed in JSON format:

```
{ } JSON ▾ D Preview ⚡ Visualize | v
```

```
1 {  
2   "bookmarked": true,  
3   "message": "Post bookmarked"  
4 }
```

At the bottom of the interface, there are icons for 'Console', 'Postbot', 'Runner', 'Vault', and other utility tools.

Figure 38: Bookmarking posts API testing

#### 4.3.9. View bookmarked posts

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various status indicators and a search bar. The main area displays a collection of API requests. One request is highlighted with a red border: "GET http://localhost:3001/post/bookmarked-posts". Below this, the request details are shown: method "GET", URL "http://localhost:3001/post/bookmarked-posts", and a "Send" button. The "Params" tab is selected, showing a table with one row: "Key" (Body) and "Value" ({}). The "Headers" tab shows "(9)" entries, and the "Body" tab shows "JSON" content. The response section shows a green "200 OK" status with a response time of "60 ms" and a size of "1.02 KB". The response body is displayed as JSON:

```
1 {  
2   "username": "user01",  
3   "profileImage": null,  
4   "posts": [  
5     {  
6       "id": 4,  
7       "title": "post4",  
8       "description": "desc",  
9       "imagePaths": [  
10         "/uploads/postImages/1753797024725.jpeg"  
11       ],  
12       "videoPaths": [],  
13       "createdAt": "2025-07-29T13:50:24.732Z",  
14       "updatedAt": "2025-07-29T13:50:24.732Z",  
15       "userId": 2,  
16       "user": {  
17         "id": 2,  
18         "username": "user01",  
19         "profileImage": null  
20       },  
21       "_count": {  
22         "likes": 0,  
23         "comments": 0  
24     }  
25   }  
26 ]  
27 }  
28 }
```

Figure 39: View bookmarked posts API testing

#### 4.3.10. Add comment

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces, More, and several recent requests. The current request is titled "Re-FYP / Comment". The method is set to POST, and the URL is http://localhost:3001/comment/. The "Body" tab is selected, showing a JSON payload:

```
1 {
2   "postId": 1,
3   "text": "This is a comment"
4 }
```

Below the body, the response is displayed under the "Body" tab. The status code is 201 Created, and the response time is 697 ms. The response body is a JSON object:

```
1 {
2   "message": "Comment added",
3   "comment": {
4     "id": 3,
5     "text": "This is a comment",
6     "userId": 1,
7     "postId": 1,
8     "parentId": null,
9     "createdAt": "2025-07-31T23:53:44.221Z",
10    "updatedAt": "2025-07-31T23:53:44.221Z",
11    "user": {
12      "username": "aasmee"
13    },
14    "parent": null,
15    "replies": []
16  }
17 }
```

At the bottom, there are buttons for Postbot, Runner, Vault, and other tools.

Figure 40: Adding comment API testing

#### 4.3.11. Get comment

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces, More, and several recent requests. The current request is a GET operation to `http://localhost:3001/comment/1`. The request details section includes tabs for Params, Auth, Headers (9), Body, Scripts, and Settings. The Headers tab is selected, showing a single header entry: `Content-Type: application/json`. The Body tab is also selected, displaying the JSON response. The response status is `200 OK`, with a timestamp of `30 ms` and a size of `582 B`. The response body is a JSON object:

```
1 {  
2   "postId": "1",  
3   "comments": [  
4     {  
5       "id": 3,  
6       "text": "This is a comment",  
7       "userId": 1,  
8       "postId": 1,  
9       "parentId": null,  
10      "createdAt": "2025-07-31T23:53:44.221Z",  
11      "updatedAt": "2025-07-31T23:53:44.221Z",  
12      "user": {  
13        "id": 1,  
14        "username": "aasmee"  
15      },  
16      "replies": []  
17    }  
18  ]  
19 }
```

At the bottom of the interface, there are buttons for Postbot, Runner, Vault, and other tools.

Figure 41: Getting all comments on a specific post

#### 4.3.12. Delete comment

The screenshot shows the Postman application interface. At the top, there are tabs for 'Workspaces' and 'More'. On the right side, there are various icons for search, refresh, settings, notifications, and account management, along with an 'Upgrade' button.

In the main workspace, there is a list of recent requests: 'POST Log', 'POST Save', 'GET Recip', 'DEL Comr' (which is highlighted in red), and 'GET Http'. Below this, the current request is listed as 'HTTP Re-FYP / Comment'.

The request details show a 'DELETE' method selected, with the URL 'http://localhost:3001/comment/3'. There are buttons for 'Save', 'Share', and 'Send' (which is highlighted in blue).

Below the request details, there are sections for 'Params', 'Auth' (marked with a green dot), 'Headers (9)', 'Body' (marked with a green dot), 'Scripts', and 'Settings'. The 'Cookies' section is also visible.

The 'Query Params' section contains a table with columns 'Key', 'Value', 'Description', and 'Bulk Edit'. It has one row with the key 'Key' and value 'Value'.

At the bottom, the response section shows a status of '200 OK' with a response time of '55 ms' and a size of '383 B'. The response body is displayed as JSON:

```
1 {  
2   "message": "Comment deleted!"  
3 }
```

The bottom navigation bar includes icons for 'Console', 'Postbot', 'Runner', 'Vault', and other tools.

Figure 42: Deleting comment API testing

#### 4.3.13. Edit comment

The screenshot shows the Postman application interface for testing a comment editing API. The top navigation bar includes 'Workspaces', 'More', a search icon, and an 'Upgrade' button. The left sidebar has icons for 'Log', 'Save', 'Recip.', 'Comi', 'Http', 'No environment', and a 'Share' button. The main workspace title is 'HTTP Re-FYP / Comment'. A 'PUT' request is selected with the URL 'http://localhost:3001/comment/2'. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "text": "Edited comment"  
3 }
```

The response section shows a successful '200 OK' status with a response time of 455 ms and a response size of 563 B. The response body is displayed in JSON format:

```
1 {  
2   "message": "Comment updated successfully",  
3   "updatedComment": {  
4     "id": 2,  
5     "text": "Edited comment",  
6     "userId": 1,  
7     "postId": 8,  
8     "parentId": null,  
9     "createdAt": "2025-07-31T22:12:58.073Z",  
10    "updatedAt": "2025-08-01T02:26:59.587Z"  
11  }  
12 }
```

At the bottom, there are links for 'Console', 'Postbot', 'Runner', 'Vault', and help icons.

Figure 43: Editing comment API testing

#### 4.3.14. Get notification

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various status icons and an Upgrade button. Below the header, a search bar and a toolbar with icons for Save, Delete, and Get are visible. The URL bar shows the endpoint `http://localhost:3001/notifications/`. The main workspace contains a request card with a GET method and the same URL. Below the card, the "Auth" tab is selected, showing a Bearer Token type and a token input field containing a redacted string. The "Headers" tab is also visible. The "Body" tab is selected, showing an empty JSON object. The response section shows a green "200 OK" status with a response time of 488 ms and a size of 1.41 KB. The response body is displayed as a JSON array with one element, representing a notification object. The JSON content is as follows:

```
1 [  
2   {  
3     "id": 5,  
4     "type": "LIKE",  
5     "read": false,  
6     "createdAt": "2025-07-31T21:32:29.689Z",  
7     "updatedAt": "2025-07-31T21:32:29.689Z",  
8     "recipientId": 1,  
9     "senderId": 2,  
10    "postId": 7,  
11    "sender": {  
12      "id": 2,  
13      "username": "user01",  
14      "profileImage": null  
15    },  
16    "post": {  
17      "id": 7,  
18      "imagePaths": [  
19        "/uploads/postImages/1753960309422.jpg",  
20        "/uploads/postImages/1753960310040.jpg",  
21        "/uploads/postImages/1753960310105.jpg"  
22      ],  
23    }  
24  ]
```

Figure 44: Getting notification API testing

#### 4.3.15. Read notifications

The screenshot shows the Postman application interface. At the top, there are navigation links for 'Workspaces' and 'More', along with various status icons and a prominent orange 'Upgrade' button. Below the header, a list of recent requests includes 'Save', 'GET Recip...', 'DEL Comr...', 'GET Recor...', and the current request, 'PATCH http://localhost:3001/notifications/1/read'. The main workspace displays a 'PATCH' request to 'http://localhost:3001/notifications/1/read'. The 'Auth' tab is selected, showing 'Bearer Token' as the type and a token value displayed in a yellow-highlighted input field. The 'Headers' tab indicates 8 headers. The 'Body' tab shows an empty JSON object. The response section shows a green '200 OK' status with a response time of 358 ms and a size of 369 B. The response body is a JSON object:

```
1 {  
2   "success": true  
3 }
```

Figure 45: Reading notification API testing

#### 4.3.16. Add items to Shopping List

The screenshot shows the Postman application interface. At the top, there are navigation menus for Workspaces and More, along with various status icons. The main workspace title is "Re-FYP / List". A POST request is selected, and the URL is set to "http://localhost:3001/list/". The "Body" tab is active, displaying a JSON payload:

```
1 {  
2   "name": "Apple",  
3   "quantity": 6,  
4   "unit": "pcs"  
5 }
```

Below the request, the response details are shown: "201 Created" with a timestamp of "32 ms" and a size of "476 B". The response body is displayed in JSON format:

```
1 {  
2   "id": 5,  
3   "name": "Apple",  
4   "quantity": 6,  
5   "unit": "pcs",  
6   "isChecked": false,  
7   "createdAt": "2025-07-31T15:08:09.209Z",  
8   "userId": 1  
9 }
```

At the bottom, there are links for Postbot, Runner, Vault, and other Postman features.

Figure 46: Adding items to list API testing

#### 4.3.17. Get shopping list

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, refresh, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar shows a POST request to 'Pos' and a GET request to 'http://localhost:3001/list'. The main workspace displays a GET request to 'http://localhost:3001/list'. The 'Auth' tab is selected, showing 'Bearer Token' as the type and a token value. The 'Body' tab is also selected, showing an empty JSON object. The response status is '200 OK' with a duration of 78 ms and a size of 822 B. The response body is a JSON array containing three items, each representing a shopping item with fields: id, name, quantity, unit, isChecked, createdAt, and userId.

```
1 [  
2 {  
3   "id": 4,  
4   "name": "Rice",  
5   "quantity": 100,  
6   "unit": "g",  
7   "isChecked": false,  
8   "createdAt": "2025-07-29T09:43:23.792Z",  
9   "userId": 1  
10 },  
11 {  
12   "id": 3,  
13   "name": "Flour",  
14   "quantity": 500,  
15   "unit": "g",  
16   "isChecked": false,  
17   "createdAt": "2025-07-29T09:42:37.644Z",  
18   "userId": 1  
19 },  
20 {  
21   "id": 2,  
22   "name": "Milk",  
23   "quantity": 1,  
-   -   -   -   -
```

Figure 47: Get shopping list of the active user API testing

#### 4.3.18. Update list

The screenshot shows the Postman application interface for API testing. The top navigation bar includes 'Workspaces', 'More', a search icon, a settings gear, a bell notification icon, and an 'Upgrade' button. Below the header, a list of recent requests is shown: 'http' (red), 'GET Recip' (red), 'GET http://' (red), 'GET http://' (red), and 'PATCH http://' (red). The current request is selected and shows the URL 'http://localhost:3001/list/2'. The method is set to 'PATCH'. The 'Send' button is highlighted in blue. Below the URL, the request body is defined as 'Body' (underlined) and 'JSON' (selected). The JSON payload is:

```
1 {  
2   "isChecked": true,  
3   "name": "Milk",  
4   "quantity": 1,  
5   "unit": "L"  
6 }
```

At the bottom of the request panel, the response status is '200 OK', execution time is '186 ms', and size is '467 B'. The response body is displayed in JSON format:

```
1 {  
2   "id": 2,  
3   "name": "Milk",  
4   "quantity": 1,  
5   "unit": "L",  
6   "isChecked": true,  
7   "createdAt": "2025-07-29T09:23:47.775Z",  
8   "userId": 1  
9 }
```

The bottom navigation bar includes icons for 'Console', 'Postbot', 'Runner', 'Vault', and other application features.

Figure 48: Update shopping list item API testing

#### 4.3.19. Delete list item

The screenshot shows the Postman application interface during an API test. The top navigation bar includes 'Workspaces', 'More', a search icon, a settings gear, a bell notification icon, and an 'Upgrade' button. The main workspace shows a sequence of requests: 'GET Recip', 'GET http://', 'GET http://', and 'DEL List'. The current request is 'DEL List' with the URL `http://localhost:3001/list/4`. The method dropdown shows 'DELETE'. Below the URL, there are tabs for 'Params', 'Auth', 'Headers (9)', 'Body', 'Scripts', and 'Settings'. The 'Body' tab is selected, showing 'Query Params' with a table:

Key	Value	Description	Bulk Edit
Key	Value	Description	

Below the table, the response section displays a green '200 OK' status with a timestamp of '27 ms' and a size of '392 B'. The response body is shown in JSON format:

```
1 {  
2   "message": "Item deleted successfully"  
3 }
```

At the bottom of the interface, there are icons for 'Console', 'Postbot', 'Runner', 'Vault', and other tools.

Figure 49: Deleting list item API testing

#### 4.3.20. Get all recipes

The screenshot shows the Postman application interface. The left sidebar has sections for Collections, Environments, Flows, and History. The main area shows a collection named "Re-FYP / Recipe". A GET request is selected with the URL "http://localhost:3001/recipes/". The response status is 200 OK, with a duration of 112 ms and a size of 54.83 KB. The response body is displayed as JSON:

```
[{"id": 1, "name": "Spaghetti Bolognese", "imageUrl": "https://umamiology.com/wp-content/uploads/2024/04/Umamiology-Spaghetti-Bolognese-RecipeCard16.jpg", "description": "A classic Italian pasta dish with rich tomato sauce.", "tags": [{"id": 0, "label": "savory"}, {"id": 1, "label": "rich"}, {"id": 2, "label": "meaty"}, {"id": 3, "label": "italian"}], {"id": 10, "name": "Egg Tart", "imageUrl": "https://assets.bonappetit.com/photos/59b807037f7c9a1ef926e5f6/1/w_2560%2Cc_limit/portuguese-egg-custard-tarts.jpg", "description": "A sweet and creamy custard-filled pastry.", "tags": [{"id": 0, "label": "sweet"}, {"id": 1, "label": "creamy"}, {"id": 2, "label": "flaky"}]}
```

Figure 50: Get all recipes API testing

#### 4.3.21. Get a single recipe

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various status indicators and a central search bar. Below the header, a navigation bar includes a lock icon, a back arrow, a forward arrow, and a dropdown for environments, currently set to 'No environment'. A prominent orange 'Upgrade' button is visible. The main workspace displays a collection named 'Re-FYP / Recipe'. A specific GET request is selected, with the URL 'http://localhost:3001/recipes/17' entered in the request field. To the right of the URL are 'Save' and 'Share' buttons. A large blue 'Send' button is positioned below the URL. The request details panel shows the method 'GET' and the URL. Below this, tabs for Params, Auth, Headers (6), Body, Scripts, and Settings are visible, with 'Headers (6)' currently selected. The 'Body' tab is expanded, showing a JSON response with line numbers from 1 to 26. The JSON data represents a recipe for 'Jambalaya' with fields for id, name, imageUrl, description, tags, ingredients, and steps. The response status is '200 OK' with a duration of 347 ms and a size of 929 B. There are options to save the response and view it in various formats. At the bottom of the interface, there are tabs for Console, Postbot, Runner, Vault, and other tools.

```
1 {  
2   "id": "17",  
3   "name": "Jambalaya",  
4   "imageUrl": "https://kolbykash.com/wp-content/uploads/2023/07/  
        Jambalaya-Finished--scaled.jpg",  
5   "description": "A spicy Creole rice dish with chicken, sausage, and shrimp.",  
6   "tags": [  
7     "spicy",  
8     "savory",  
9     "Creole"  
10    ],  
11   "ingredients": [  
12     "egg",  
13     "parsley",  
14     "mushroom"  
15    ],  
16   "steps": [  
17     {  
18       "order": 1,  
19       "content": "Sauté sausage, chicken, onions, and bell peppers."  
20     },  
21     {  
22       "order": 2,  
23       "content": "Add tomatoes, rice, and Cajun seasoning."  
24     },  
25     {  
26       "order": 3,
```

Figure 51: Get a single recipe by id API testing

#### 4.3.22. Getting pantry items

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, refresh, settings, and notifications. An orange 'Upgrade' button is visible. The main area shows a list of recent requests: 'Pos' (red), 'POST Log' (red), 'POST Save' (green), 'GET Recip' (red), and 'GET http://...' (red). Below this is a search bar and a 'No environment' dropdown.

The current request is 'http://localhost:3001/pantry/' via HTTP. The method is set to 'GET'. The 'Auth' tab is selected, showing 'Bearer Token' as the type and a token field containing a redacted string. Other tabs include Headers (9), Body, Scripts, and Settings. A note explains that the authorization header will be automatically generated when the request is sent.

In the 'Body' section, the response is shown as JSON. The status is '200 OK' with a response time of 44 ms and a body size of 514 B. The JSON data is:

```
1 [  
2   {  
3     "id": 1,  
4     "name": "Salt",  
5     "quantity": 200,  
6     "unit": "gm",  
7     "category": "Others",  
8     "createdAt": "2025-08-01T03:38:41.560Z",  
9     "updatedAt": "2025-08-01T03:38:41.560Z",  
10    "userId": 1  
11  }  
12 ]
```

At the bottom, there are buttons for Postbot, Runner, Vault, and other tools. A 'Console' tab is also present.

Figure 52: Getting pantry items API testing

#### 4.3.23. Add pantry items

The screenshot shows the Postman application interface for testing an API. The URL is set to `http://localhost:3001/pantry`. A POST request is selected. The Body tab is active, showing the following JSON payload:

```
1 {  
2   "name": "Salt",  
3   "quantity": 200,  
4   "unit": "gm"  
5 }  
6
```

The response status is `201 Created`, with a response time of `141 ms` and a response size of `517 B`. The response body is displayed as:

```
1 {  
2   "id": 1,  
3   "name": "Salt",  
4   "quantity": 200,  
5   "unit": "gm",  
6   "category": "Others",  
7   "createdAt": "2025-08-01T03:38:41.560Z",  
8   "updatedAt": "2025-08-01T03:38:41.560Z",  
9   "userId": 1  
10 }
```

Figure 53: Adding items to pantry API testing

#### 4.3.24. Delete Pantry items

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, refresh, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar shows recent requests: 'Pos' (red), 'POST Log' (red), 'POST Save' (green), 'GET Recip' (red), and 'DEL http://...' (red). The current request is 'http://localhost:3001/pantry/2'. To the right of the URL are 'Save' and 'Share' buttons, and a copy icon.

The main area shows a 'DELETE' request to 'http://localhost:3001/pantry/2'. The 'Auth' tab is selected, showing 'Bearer Token' as the type and a token value displayed in a yellow-bordered input field. Other tabs include 'Params', 'Headers (9)', 'Body', 'Scripts', and 'Settings'. A note on the left explains that the authorization header will be automatically generated when sending the request, with a link to learn more about Bearer Token authorization.

Below the request details, the response status is '200 OK' with a response time of '126 ms' and a size of '392 B'. The response body is shown in JSON format:

```
1 {  
2   "message": "Item deleted successfully"  
3 }
```

At the bottom of the interface, there are buttons for 'Console', 'Postbot', 'Runner', 'Vault', and other tools. A help icon is also present.

Figure 54: Deleting items from pantry API testing

#### 4.3.25. Save search tags

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various status icons and an Upgrade button. Below the header, a navigation bar includes a lock icon, a back arrow, a forward arrow, a plus sign, and a dropdown for environments, currently set to 'No environment'. The main workspace displays a POST request to 'http://localhost:3001/recipes/save-tags'. The request method is selected as 'POST' from a dropdown, and the URL is shown in the input field. To the right of the URL are 'Save', 'Share', and 'Copy' buttons. Below the URL, the 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "userId": 2,  
3   "tagId": [3]  
4 }  
5
```

Below the body, the response section shows a successful '200 OK' status with a response time of 17 ms and a response size of 404 B. The response body is displayed as JSON:

```
{ } JSON ▾
```

```
1 {  
2   "success": true,  
3   "message": "Tag saved successfully"  
4 }
```

At the bottom of the interface, there are several utility buttons: Console, Postbot, Runner, Vault, and others.

Figure 55: Save tags from the user's search API testing

#### 4.3.26. Get profile

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, user, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar shows a sequence of requests: POST Reg, POST http, GET Recip, GET https, and GET https. To the right of the navigation is a dropdown for 'No environment'. The main workspace shows a request to 'http://localhost:3001/user/profile' via 'GET'. The 'Auth' tab is selected, showing 'Bearer Token' as the type. A token value is displayed in a yellow-bordered input field. The 'Body' tab shows a JSON response with 15 lines of code. The response status is '200 OK' with a duration of 76 ms and a size of 1.34 KB. The JSON response is as follows:

```
1 {  
2   "id": 1,  
3   "username": "aasmee",  
4   "email": "aasmee0408@gmail.com",  
5   "bio": null,  
6   "profileImage": null,  
7   "createdAt": "2025-07-23T07:57:41.522Z",  
8   "posts": [  
9     {  
10       "id": 8,  
11       "title": "post8",  
12       "description": "desc",  
13       "imagePaths": [  
14         "/uploads/postImages/1753960608459.jpg"  
15       ],  
16     }  
17   ]  
18 }  
19  
20 }
```

Figure 56: Get user profile API testing

#### 4.3.27. Recommendations

The screenshot shows the Postman application interface. At the top, there are tabs for Workspaces and More, along with various icons for search, refresh, settings, and notifications. An orange 'Upgrade' button is visible. Below the header, a navigation bar includes a lock icon, a log entry, a POST save, a GET Recip entry, a POST Con entry, a GET http:// entry, and a 'No environment' dropdown.

The main workspace displays a request to `http://localhost:3001/recommendation/recommendations`. The method is set to GET, and the URL is `http://localhost:3001/recommendation/recommendations`. There are buttons for Save, Share, and a copy icon. Below the URL, there are tabs for Params, Auth (selected), Headers (7), Body, Scripts, and Settings. The Auth tab shows 'Bearer Token' selected, and a token field containing a redacted string.

On the right side, there are sections for Cookies and a preview of the response. The response status is 200 OK, with a duration of 145 ms and a size of 3.93 KB. The response body is shown in JSON format:

```
{
  "id": 11,
  "name": "Bibimbap",
  "imageUrl": "https://carlsbadcravings.com/wp-content/uploads/2023/09/Bibimbap-9.jpg",
  "description": "Korean rice bowl with assorted vegetables, egg, and spicy sauce.",
  "tags": [
    {
      "id": 0,
      "value": "38"
    },
    {
      "id": 1,
      "value": "39"
    },
    {
      "id": 2,
      "value": "40"
    },
    {
      "id": 3,
      "value": "41"
    }
  ]
}
```

Below this, another item is listed:

```
{
  "id": 21,
  "name": "Ramen",
  "imageUrl": "https://www.craftycookbook.com/wp-content/uploads/2023/04/tonkotsu.jpg",
  "description": "A Japanese noodle soup with broth, meat, and vegetables.",
  "tags": [
    {
      "id": 0,
      "value": "73"
    },
    {
      "id": 1,
      "value": "74"
    },
    {
      "id": 2,
      "value": "75"
    },
    {
      "id": 3,
      "value": "76"
    }
  ]
}
```

At the bottom, there is a 'Console' tab and several footer icons: Postbot, Runner, Vault, and others.

Figure 57: Recommendation system API testing

## Chapter 5: Conclusion

### 5.1. Advantages

- **Reduced Food Waste:** The application helps users efficiently utilize existing pantry items, minimizing food wastage.
- **Time Efficiency:** Recipe suggestions and shopping list generation streamline meal planning and grocery management.
- **User Engagement:** The integrated community forum allows users to share recipes, tips, and cooking experiences, fostering collaboration.
- **Convenience:** Features like pantry management, expiry tracking, and shopping list creation make meal preparation easier.
- **Personalized Experience:** Recipe filters and dietary customization ensure recommendations suit individual user preferences.

### 5.2. Limitations

- Most of the app's advanced features are not yet fully automated and require further development.
- OCR accuracy may vary depending on image quality.
- Requires an active internet connection for recipe suggestions and community features.
- Community moderation features are limited in the current version.

## **Chapter 6: Future Work**

### **1. Enhanced Recommendation System:**

Improve the recipe suggestion feature for more personalized and context-based results.

### **2. Improved User Interaction:** Introduce direct messaging, interactive community events, and better user engagement features.

### **3. Image-Based Recipe Suggestions:**

Allow users to scan or upload ingredient images to receive complete meal options.

### **4. Chatbot-Based Cooking Assistance:**

Implement a virtual assistant to guide users step by step during cooking.

### **5. Offline Mode:**

Provide offline functionality for accessing saved recipes and pantry data.

### **6. Smart Kitchen Integration:**

Enable future compatibility with smart kitchen devices for automated pantry synchronization and inventory tracking

## Chapter 7: References

- i. Aptoide, 2025. *Cookpad recipes, homemade food* -. [Online] Available at: <https://cookpad.en.aptoide.com/app> [Accessed 06 January 2025].
- ii. Aptoide, 2025. *Yummly Recipes & Cooking Tools.* [Online] Available at: <https://yummly.en.aptoide.com/app> [Accessed 06 January 2025].
- iii. creately, 2022. *Use Case Diagram Tutorial (Guide with Examples).* [Online] Available at: <https://creately.com/guides/use-case-diagram-tutorial/> [Accessed 25 4 2025].
- iv. Fakhroutdinov, K., 2025. *UML use case diagrams graphical notation reference.* [Online] Available at: <https://www.uml-diagrams.org/use-case-reference.html> [Accessed 27 4 2025].
- v. Huff, B., 2022. *'What shall we have for dinner?' Choice overload is a real problem, but these tips will make your life easier.* [Online] Available at: <https://theconversation.com/what-shall-we-have-for-dinner-choice-overload-is-a-real-problem-but-these-tips-will-make-your-life-easier-193317> [Accessed 1 December 2023].
- vi. Porter, S. D. & Reay, D. S., 2015. *Addressing food supply chain and consumption inefficiencies: potential for climate change mitigation.* [Online] Available at: <https://link.springer.com/article/10.1007/s10113-015-0783-4> [Accessed 1 December 2023].
- vii. Sensor Tower, 2025. *Tasty: Recipes, Cooking Videos - Overview.* [Online] Available at: <https://app.sensortower.com/overview/1217456898?country=US> [Accessed 06 January 2025].
- viii. THE OWL, 2024. *What is Inventory Expiration and How Can Food Manufacturers Prevent It?.* [Online]

Available at: <https://theowlssolutions.com/inventory-expiration-and-how-to-prevent-it/> [Accessed 1 December 2024].

- ix. Valcheva, S., 2025. *Data Flow Diagram*. [Online] Available at: <https://www.intellspot.com/data-flow-diagram-examples/> [Accessed 28 4 2025].
- x. Visual Paradigm, 2025. *Flowchart Tutorial (with Symbols, Guid and Examples)*. [Online] Available at: <https://www.visual-paradigm.com/tutorials/flowchart-tutorial/> [Accessed 28 04 2025].

## Chapter 8: Appendix

### 8.1. Appendix A: Pre-Survey

#### 8.1.1. Pre-Survey Form

**FYP survey form**

We are conducting a short survey to understand your cooking habits, pantry management practices, and preferences for a smart recipe app. Your feedback will help us create a user-friendly and innovative solution. The survey will take only a few minutes to complete.

Thank you for your valuable input!

np05cp4a220006@iic.edu.np Switch account 

 Not shared

\* Indicates required question

Your Name \*

Your answer

1. How often do you cook at home?

Daily

A few times a week

Once a week

Rarely

Never

Figure 58: Pre-Survey Form pt.1

2. How do you currently find recipes to cook?

- Online search engines (e.g., Google, YouTube)
- Recipe apps/websites
- Cookbooks
- Family/friends
- Other: \_\_\_\_\_

3. Do you create shopping lists for groceries?

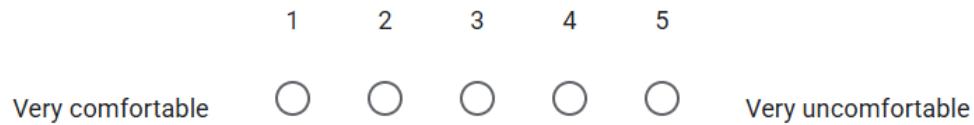
- Yes, always
- Sometimes
- Rarely
- No

4. Have you ever used a recipe or pantry management app before? \*

- Yes
- No

Figure 59: Pre-Survey Form pt.2

5. How comfortable are you using mobile apps for kitchen-related tasks? \*



6. Which features would you find most helpful in a recipe app? (Choose all that apply) \*

- AI-based ingredient recognition
- Personalized recipe suggestions
- Pantry inventory tracking
- Expiry notifications for ingredients
- Shopping list generation
- Community recipe sharing
- Other: \_\_\_\_\_

Figure 60: Pre-Survey Form pt.3

7. What dietary preferences or restrictions do you wish for the app to consider?

- Vegetarian
- Vegan
- Gluten-free
- Low-carb/Keto
- None
- Other: \_\_\_\_\_

8. How often do you throw away unused or expired ingredients? \*

- Often (weekly)
- Occasionally (monthly)
- Rarely (a few times a year)
- Never

9. Would you prefer notifications about low-stock or near-expiry items?

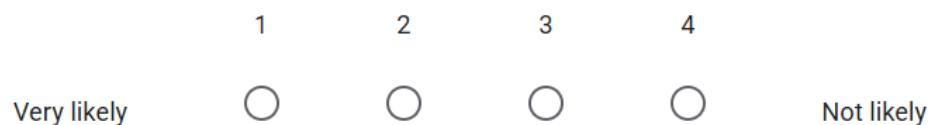
- Yes, very useful
- Maybe, if they're not too frequent
- No, I wouldn't use this feature

Figure 61: Pre-Survey Form pt.4

10. What's your biggest challenge in cooking at home?

- Deciding what to cook
- Matching recipes with available ingredients
- Considering dietary restrictions/preferences
- Other: \_\_\_\_\_

11. How likely are you to recommend the app to friends or family?



12. Do you have any suggestions for improving the platform?

Your answer  
\_\_\_\_\_

**Submit**

[Clear form](#)

Figure 62: Pre-Survey Form pt.5

### 8.1.2. Sample of Filled Pre-Survey Forms

Responses cannot be edited

## FYP survey form

We are conducting a short survey to understand your cooking habits, pantry management practices, and preferences for a smart recipe app. Your feedback will help us create a user-friendly and innovative solution. The survey will take only a few minutes to complete. Thank you for your valuable input!

\* Indicates required question

Your Name \*

Sadiksha Rai

1. How often do you cook at home?

Daily

A few times a week

Once a week

Rarely

Never

Figure 63: Filled Pre-Survey Form Sample pt.1

2. How do you currently find recipes to cook?

Online search engines (e.g., Google, YouTube)

Recipe apps/websites

Cookbooks

Family/friends

Other: \_\_\_\_\_

3. Do you create shopping lists for groceries?

Yes, always

Sometimes

Rarely

No

Figure 64: Filled Pre-Survey Form Sample pt.2

4. Have you ever used a recipe or pantry management app before? \*

Yes

No

5. How comfortable are you using mobile apps for kitchen-related tasks? \*

1      2      3      4      5

Very comfortable

Very uncomfortable

Figure 65: Filled Pre-Survey Form Sample pt.3

6. Which features would you find most helpful in a recipe app? (Choose all that apply) \*

- AI-based ingredient recognition
- Personalized recipe suggestions
- Pantry inventory tracking
- Expiry notifications for ingredients
- Shopping list generation
- Community recipe sharing
- Other: .....

7. What dietary preferences or restrictions do you wish for the app to consider?

- Vegetarian
- Vegan
- Gluten-free
- Low-carb/Keto
- None
- Other: .....

Figure 66: Filled Pre-Survey Form Sample pt.4

8. How often do you throw away unused or expired ingredients? \*

- Often (weekly)
- Occasionally (monthly)
- Rarely (a few times a year)
- Never

9. Would you prefer notifications about low-stock or near-expiry items?

- Yes, very useful
- Maybe, if they're not too frequent
- No, I wouldn't use this feature

Figure 67: Filled Pre-Survey Form Sample pt.5

10. What's your biggest challenge in cooking at home?

- Deciding what to cook
- Matching recipes with available ingredients
- Considering dietary restrictions/preferences
- Other: \_\_\_\_\_

11. How likely are you to recommend the app to friends or family?

1

2

3

4

Very likely



Not likely

12. Do you have any suggestions for improving the platform?

Figure 68: Filled Pre-Survey Form Sample pt.6

### 8.1.3. Pre-Survey Result

1. How often do you cook at home?

69 responses

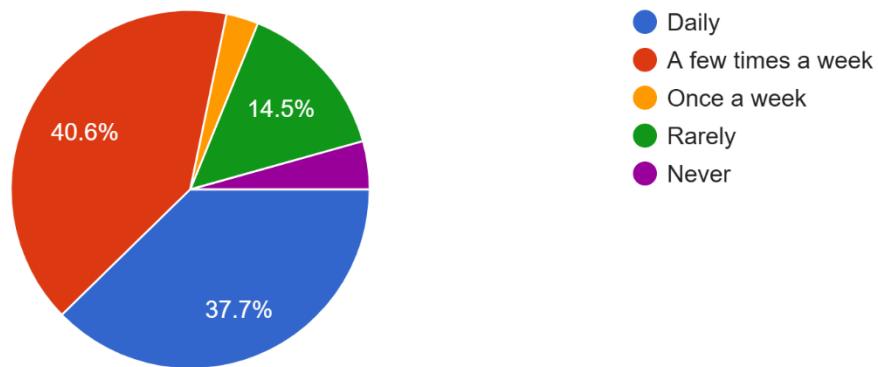


Figure 69: Pre-Survey Q1

2. How do you currently find recipes to cook?

68 responses

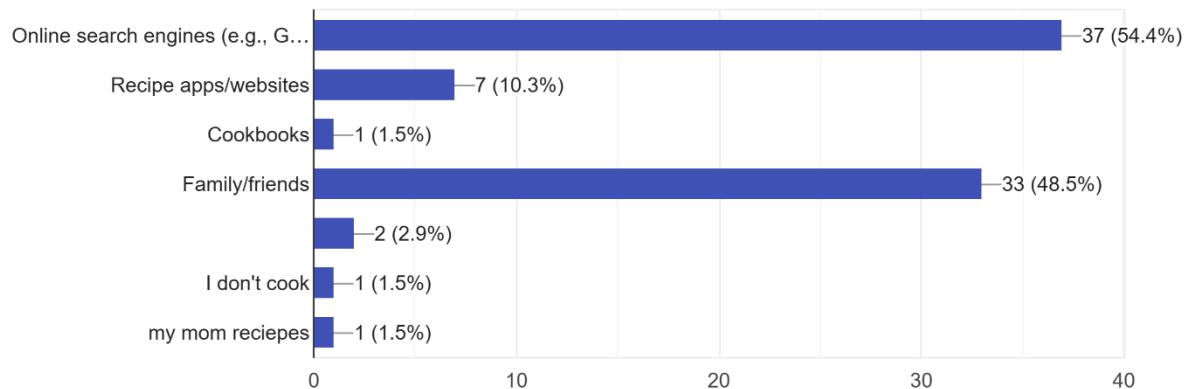


Figure 70: Pre-Surgery Q2

3. Do you create shopping lists for groceries?

68 responses

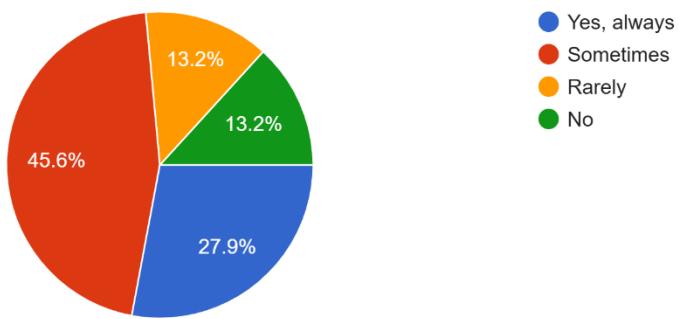


Figure 71: Pre-Survey Q3

4. Have you ever used a recipe or pantry management app before?

69 responses

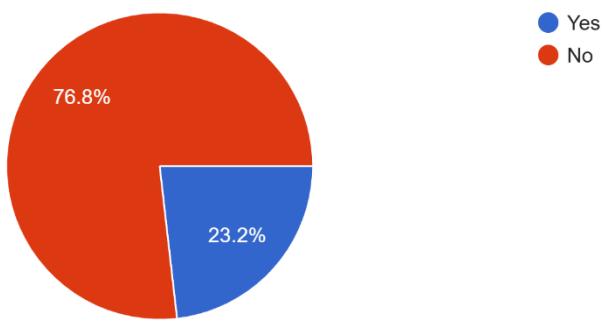


Figure 72: Pre-Survey Q4

5. How comfortable are you using mobile apps for kitchen-related tasks?

69 responses

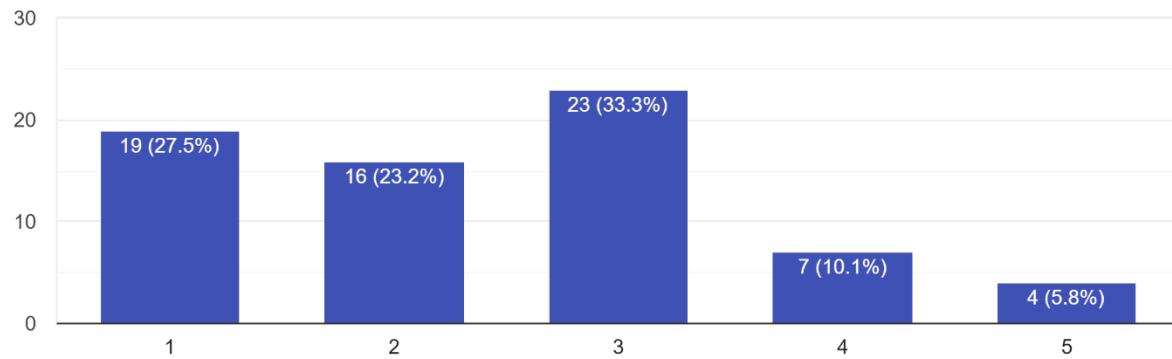


Figure 73: Pre-Survey Q5

6. Which features would you find most helpful in a recipe app? (Choose all that apply)

69 responses

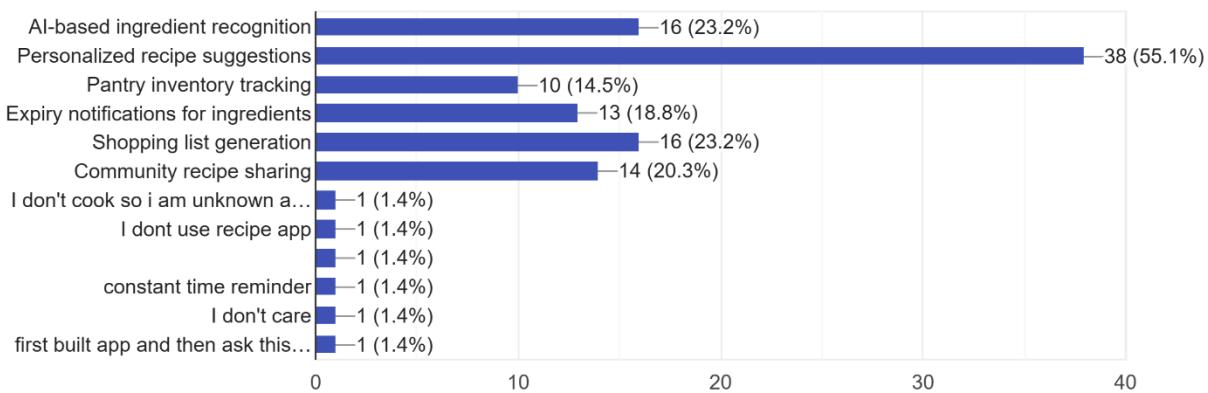


Figure 74: Pre-Survey Q6

7. What dietary preferences or restrictions do you wish for the app to consider?

68 responses

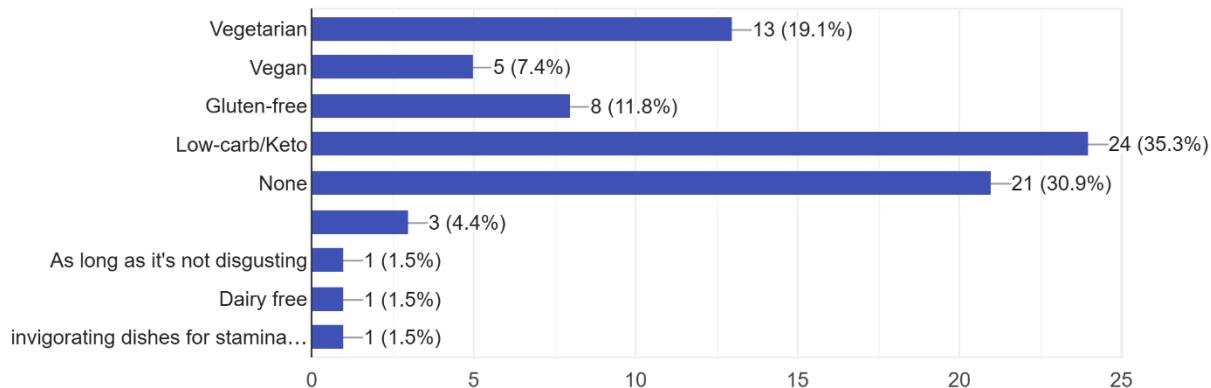


Figure 75: Pre-Survey Q7

8. How often do you throw away unused or expired ingredients?

69 responses

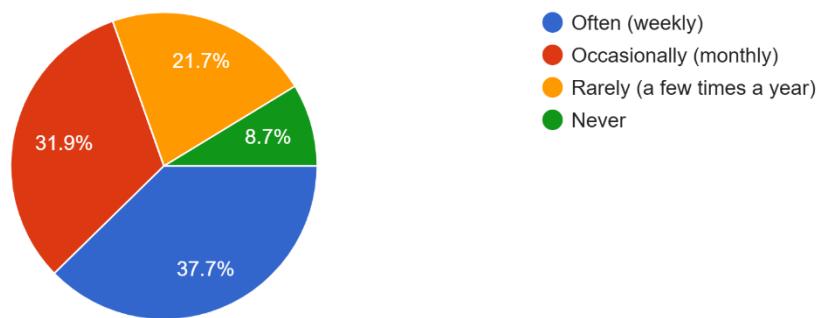


Figure 76: Pre-Survey Q8

9. Would you prefer notifications about low-stock or near-expiry items?

68 responses

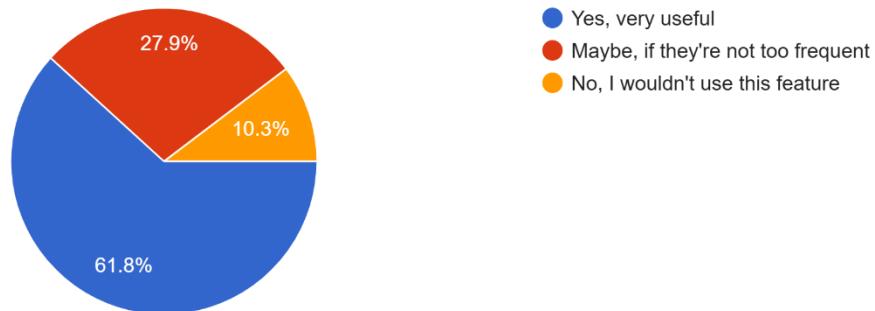


Figure 77: Pre-Survey Q9

10. What's your biggest challenge in cooking at home?

68 responses

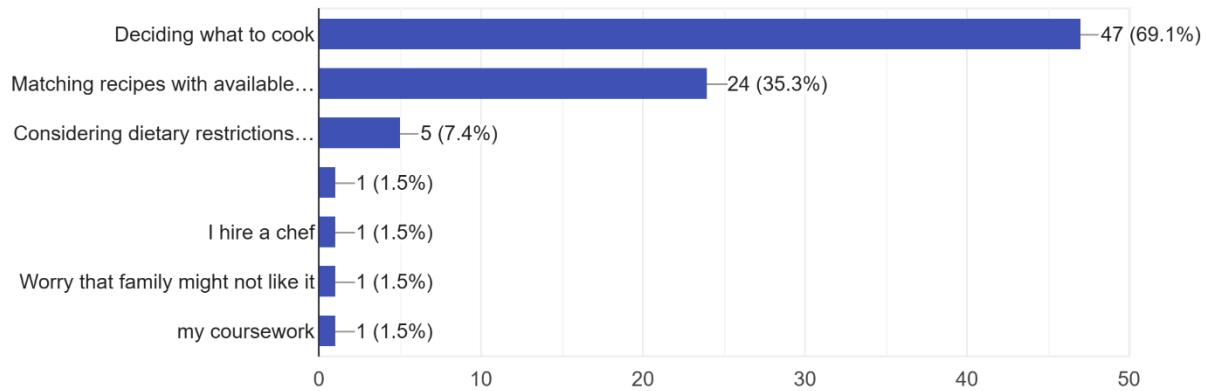


Figure 78: Pre-Survey Q10

11. How likely are you to recommend the app to friends or family?

69 responses

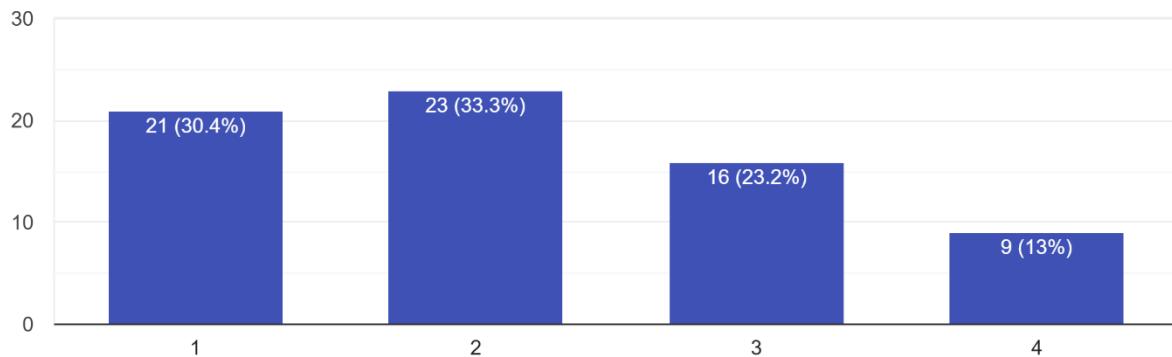


Figure 79: Pre-Survey Q11

## 8.2. Appendix B: Post-Survey

### 8.2.1. Post-Survey Form

**FYP Post-Survey Form**

\* Indicates required question

Your Name \*

Your answer

How easy was it to navigate through the app?

1      2      3      4      5

Very easy                                    Very difficult

Were you able to understand how to use the features without instructions?

Yes  
 No  
 Somewhat

Figure 80: Post Survey Form pt.1

How would you rate the overall design and layout of the app?

1      2      3      4      5

Excellent                                    Very Poor

Was the ingredient recognition accurate and helpful?

Always  
 Often  
 Sometimes  
 Rarely  
 Never

Which feature did you find most useful? \*

Pantry Management  
 Search Recipe  
 Shopping List  
 Community Forum  
 Other: \_\_\_\_\_

Figure 81: Post-Survey Form pt.2

How likely are you to use this app regularly for meal planning?

1      2      3      4      5

Very likely                                    Very unlikely

What did you like the most about the app?

Your answer

Would you like to recommend this app to other friends and families?

Yes  
 No  
 Maybe

What improvements or additional features would you suggest?

Your answer

Figure 82: Post-Survey Form pt.3

Any other comments or feedback?

Your answer

**Submit**      **Clear form**

Figure 83: Post-Survey Form pt.4

### 8.2.2. Sample of Filled Post-Survey Forms

Responses cannot be edited

## FYP Post-Survey Form

\* Indicates required question

Your Name \*

Sumi Rai

How easy was it to navigate through the app?

1      2      3      4      5

Very easy                              Very difficult

Were you able to understand how to use the features without instructions?

Yes  
 No  
 Somewhat

Figure 84: Post-Survey Smaple Form pt.1

How would you rate the overall design and layout of the app?

1	2	3	4	5		
Excellent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Very Poor

Was the ingredient recognition accurate and helpful?

- Always
- Often
- Sometimes
- Rarely
- Never

Figure 85: Post-Survey Sample Form pt.2

Which feature did you find most useful? \*

- Pantry Management
- Search Recipe
- Shopping List
- Community Forum
- Other: \_\_\_\_\_

How likely are you to use this app regularly for meal planning?

1	2	3	4	5		
Very likely	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very unlikely

What did you like the most about the app?

Easy for searching recipes as well as create a shopping list

Figure 86: Post-Survey Sample Form pt.3

Would you like to recommend this app to other friends and families?

Yes

No

Maybe

What improvements or additional features would you suggest?

---

Any other comments or feedback?

---

Figure 87: Post-Survey Sample Form pt.4

### 8.2.3. Post-Survey Result

How easy was it to navigate through the app?

28 responses

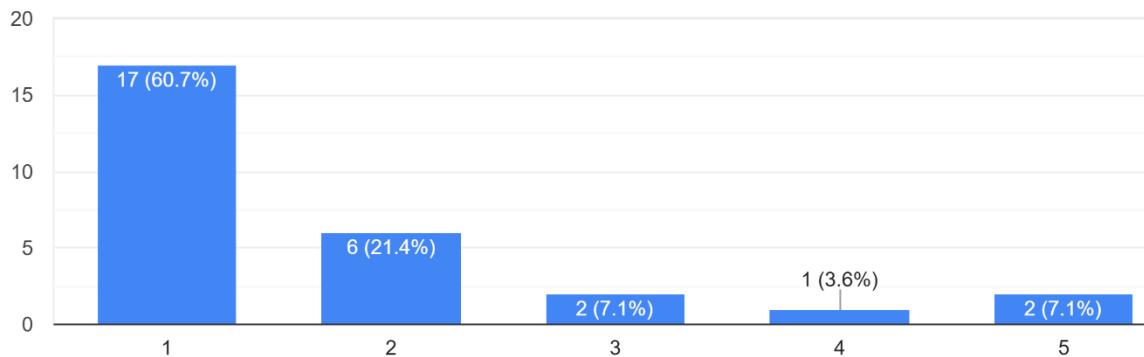


Figure 88: Post-Survey Q1

Were you able to understand how to use the features without instructions?

28 responses

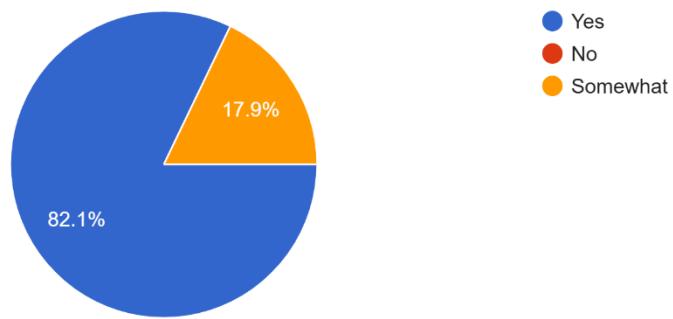


Figure 89: Post-Survey Q2

How would you rate the overall design and layout of the app?

28 responses

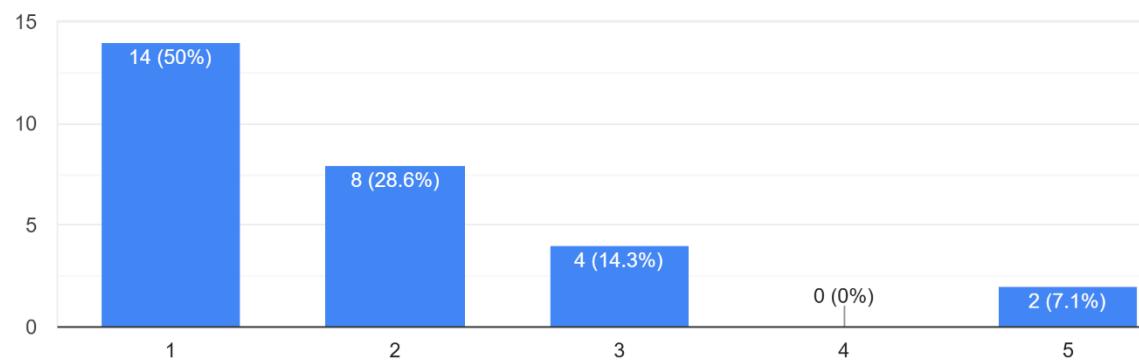


Figure 90: Post-Survey Q3

Was the ingredient recognition accurate and helpful?

28 responses

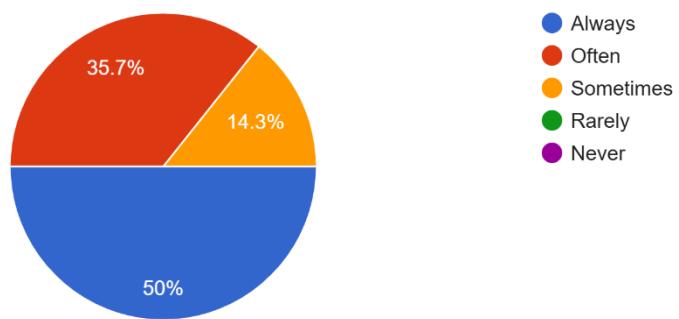


Figure 91: Post-Survey Q4

Which feature did you find most useful?

28 responses

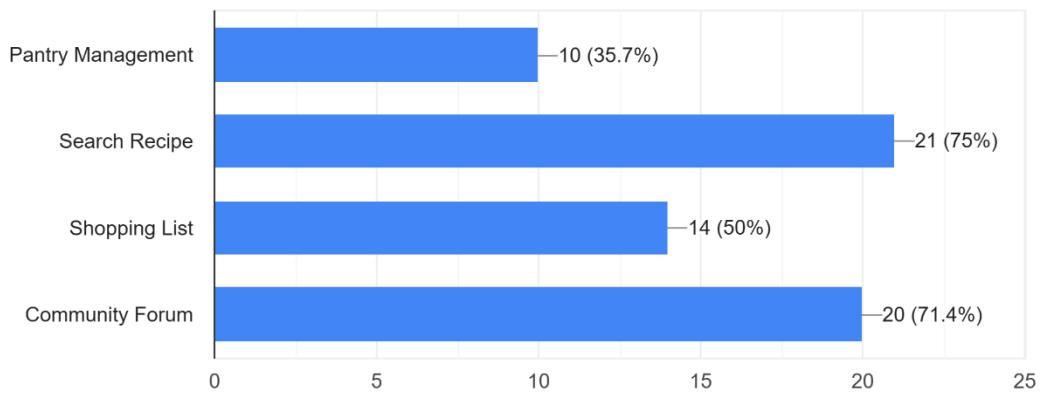


Figure 92: Post-Survey Q5

How likely are you to use this app regularly for meal planning?

28 responses

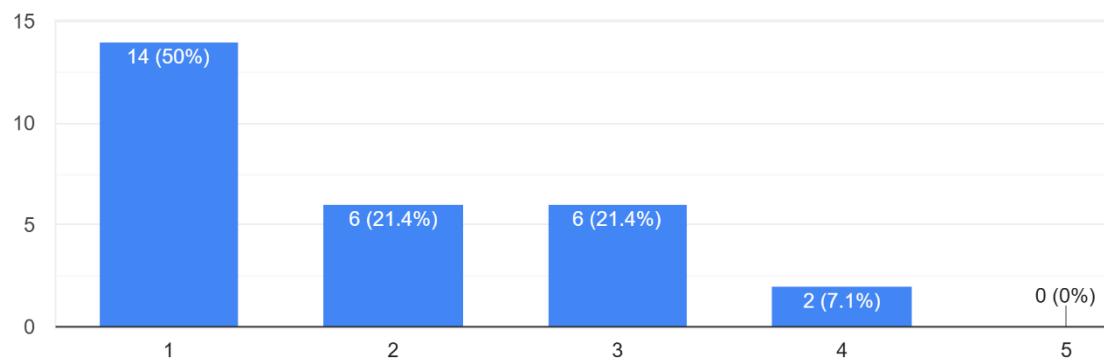


Figure 93: Post-Survey Q6

Would you like to recommend this app to other friends and families?  
28 responses

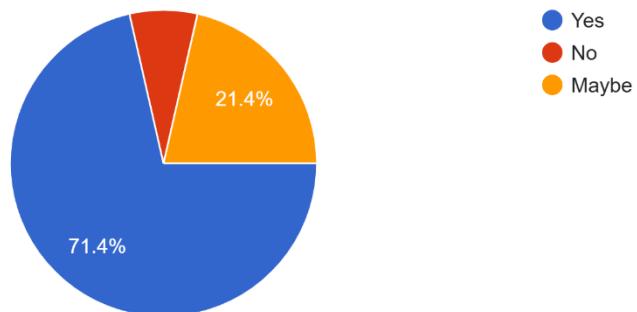


Figure 94: Post-Survey Q7

## 8.3. Appendix C: Sample Codes

### 8.3.1. Sample Code of The UI

#### Login Page

```
import 'package:flutter/material.dart';
import 'package:frontend/services/authServices.dart';
import 'package:frontend/widgets/button.dart';
import 'package:frontend/widgets/txtfield.dart';

class LoginPage extends StatefulWidget {
  @override
  _LoginPageState createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final emailController = TextEditingController();
  final passwordController = TextEditingController();
  bool rememberMe = false;

  void login() async {
    final response = await AuthService.login(
      emailController.text,
      passwordController.text,
    );

    if (response['token'] != null) {
      Navigator.pushReplacementNamed(context, '/navbar');
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(response['message'] ?? 'Login failed')),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: BoxDecoration(
          image: DecorationImage(
            image: AssetImage('images/login.jpeg'),
            fit: BoxFit.cover,
          ),
        ),
      ),
    );
  }
}
```

```

child: Center(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 15),
    child: SingleChildScrollView(
      child: Column(
        children: [
          const Text(
            "Login",
            style: TextStyle(
              color: Colors.white,
              fontSize: 32,
              fontWeight: FontWeight.bold,
            ),
          ),
          const Text(
            "Welcome back! ",
            style: TextStyle(color: Colors.white, fontSize: 16.5),
          ),
          SizedBox(height: 16),
          Textfield(
            controller: emailController,
            hintText: "Email",
            label: "Email",
            labelColor: Colors.white,
            keyboardType: TextInputType.emailAddress,
          ),
          const SizedBox(height: 9),
          Textfield(
            controller: passwordController,
            hintText: "Password",
            label: "Password",
            labelColor: Colors.white,
            isPassword: true,
          ),
          Row(
            children: [
              Checkbox(
                value: rememberMe,
                onChanged: (val) => setState(() => rememberMe = val!),
              ),
              Text(
                "Remember me",
                style: TextStyle(color: Colors.white),
              ),
              Spacer(),
            ],
          ),
        ],
      ),
    ),
  ),
);

```

```

        TextButton(
            onPressed: () {},
            child: Text(
                "Forgot password?",
                style: TextStyle(color: Colors.white),
            ),
        ),
    ],
),
Button(
    text: "Login",
    color: Colors.white,
    txtColor: Colors.black,
    onPressed: login,
),
Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
        const Text(
            "Don't have an account?",
            style: TextStyle(color: Colors.white, fontSize: 14),
        ),
        TextButton(
            onPressed:
                () => Navigator.pushNamed(context, '/register'),
            child: const Text(
                "Register",
                style: TextStyle(color: Colors.red, fontSize: 14),
            ),
        ),
    ],
),
),
),
),
),
),
),
);
},
);
}
}

```

## Register Page

```
import 'package:flutter/material.dart';
import 'package:frontend/services/authServices.dart';
import 'package:frontend/widgets/button.dart';
import 'package:frontend/widgets/txtfield.dart';

class RegisterPage extends StatefulWidget {
  @override
  _RegisterPageState createState() => _RegisterPageState();
}

class _RegisterPageState extends State<RegisterPage> {
  final nameController = TextEditingController();
  final emailController = TextEditingController();
  final passwordController = TextEditingController();
  final confirmPasswordController = TextEditingController();
  bool agreeTerms = false;

  void register() async {
    final response = await AuthService.register(
      nameController.text,
      emailController.text,
      passwordController.text,
      confirmPasswordController.text,
    );
    if (response['token'] != null) {
      Navigator.pushReplacementNamed(context, '/login');
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(response['message'] ?? 'Login failed')),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        decoration: BoxDecoration(
          image: DecorationImage(
            image: AssetImage('images/login.jpeg'),
            fit: BoxFit.cover,
          ),
        ),
      ),
    );
  }
}
```

```
child: Padding(
  padding: const EdgeInsets.symmetric(horizontal: 15),
  child: Center(
    child: SingleChildScrollView(
      child: Column(
        children: [
          Text(
            "Register",
            style: TextStyle(
              fontSize: 32,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
          ),
          SizedBox(height: 10),
          Textfield(
            controller: nameController,
            hintText: "Name",
            label: "Name",
            labelColor: Colors.white,
          ),
          const SizedBox(height: 9),
          Textfield(
            controller: emailController,
            hintText: "Email",
            label: "Email",
            labelColor: Colors.white,
            keyboardType: TextInputType.emailAddress,
          ),
          const SizedBox(height: 9),
          Textfield(
            controller: passwordController,
            hintText: "Password",
            label: "Password",
            labelColor: Colors.white,
            isPassword: true,
          ),
          const SizedBox(height: 9),
          Textfield(
            controller: confirmPasswordController,
            hintText: "Confirm Password",
            label: "Confirm Password",
            labelColor: Colors.white,
            isPassword: true,
          ),
        ],
      ),
    ),
  ),
)
```



## Home Page

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:frontend/constants.dart';
import 'package:frontend/screens/Home/list.dart';
import 'package:frontend/screens/Home/notification.dart';
import 'package:frontend/screens/Home/widgets/post_card.dart';
import 'package:http/http.dart' as http;

class Home extends StatefulWidget {
  const Home({super.key});

  @override
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  List<dynamic> posts = [];
  bool isLoading = true;
  String errorMessage = '';
  static final String _baseUrl = "${ApiConfig.baseUrl}/post";

  @override
  void initState() {
    super.initState();
    fetchPosts();
  }

  Future<void> fetchPosts() async {
    try {
      final response = await http.get(Uri.parse('$_baseUrl/get'));
      if (response.statusCode == 200) {
        final List<dynamic> data = json.decode(response.body);
        setState(() {
          posts = data;
          isLoading = false;
        });
      } else {
        setState(() {
          errorMessage = 'Failed to fetch posts: ${response.statusCode}';
          isLoading = false;
        });
      }
    } catch (error) {
```

```
        setState(() {
            errorMessage = 'Error fetching posts: $error';
            isLoading = false;
        });
    }
}

@Override
Widget build(BuildContext context) {
    if (isLoading) {
        return const Center(child: CircularProgressIndicator());
    }

    if (errorMessage.isNotEmpty) {
        return Center(child: Text(errorMessage));
    }

    if (posts.isEmpty) {
        return const Center(child: Text("No posts found"));
    }

    return Scaffold(
        backgroundColor: Colors.white,
        appBar: AppBar(
            backgroundColor: Colors.white,
            title: const Text(
                "Ingreedy",
                style: TextStyle(color: Colors.black, fontStyle: FontStyle.italic),
            ),
            actions: [
                IconButton(
                    icon: const Icon(Icons.list_alt, color: Colors.black),
                    onPressed: () {
                        Navigator.of(
                            context,
                        ).push(MaterialPageRoute(builder: (context) => const ListPage()));
                    },
                ),
                IconButton(
                    icon: const Icon(Icons.notifications_outlined, color: Colors.black),
                    onPressed: () {
                        Navigator.of(context).push(
                            MaterialPageRoute(
                                builder: (context) => const NotificationPage(),
                            ),
                        );
                    },
                ),
            ],
        ),
    );
}
```

```

        );
    },
),
],
),
body: SingleChildScrollView(
    child: Column(
        children: posts.map((post) => PostWidget(post: post)).toList(),
    ),
),
);
}
}

```

## Recipe Page

```

import 'package:flutter/material.dart';
import 'package:frontend/models/recipe.model.dart';
import 'package:frontend/screens/Recipe/recipe_tile.dart';
import 'package:frontend/services/recipe.service.dart';
import 'package:frontend/widgets/scan_icon.dart';

class RecipeListScreen extends StatefulWidget {
    const RecipeListScreen({super.key});

    @override
    State<RecipeListScreen> createState() => _RecipeListScreenState();
}

class _RecipeListScreenState extends State<RecipeListScreen> {
    List<Recipe> allRecipes = [];
    List<Recipe> filteredRecipes = [];
    final TextEditingController searchController = TextEditingController();
    String selectedTag = "";
    bool isLoading = true;

    @override
    void initState() {
        super.initState();
        _loadRecipes();
    }
}

```

```

Future<void> _loadRecipes() async {
  try {
    final recipes = await RecipeService.fetchRecipes();
    setState(() {
      allRecipes = recipes;
      filteredRecipes = recipes;
      isLoading = false;
    });
  } catch (e) {
    setState(() => isLoading = false);
  }
}

void _applyFilters() {
  String query = searchController.text.toLowerCase();
  setState(() {
    filteredRecipes =
      allRecipes.where((recipe) {
        final matchesQuery =
          recipe.name.toLowerCase().contains(query) ||
          recipe.tags.any((tag) => tag.toLowerCase().contains(query));
        final matchesTag =
          selectedTag.isEmpty ||
          recipe.tags
            .map((e) => e.toLowerCase())
            .contains(selectedTag.toLowerCase());
        return matchesQuery && matchesTag;
      }).toList();
  });
}

void _onTagSelected(String tag) {
  selectedTag = tag;
  _applyFilters();
}

void _clearFilter() {
  setState(() {
    selectedTag = "";
    searchController.clear();
  });
  _applyFilters();
}

```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white,
    appBar: AppBar(
      backgroundColor: Colors.white,
      title: const Text(
        "Recipes",
        style: TextStyle(
          color: Colors.black,
          fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      actions: [IconButton(onPressed: () {}, icon: ScanFrameIcon())],
    ),
    body: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: TextField(
            controller: searchController,
            onChanged: (_) => _applyFilters(),
            decoration: InputDecoration(
              hintText: 'Search recipes...',
              prefixIcon: const Icon(Icons.search),
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(8),
              ),
            ),
          ),
        ),
        Padding(
          padding: EdgeInsets.symmetric(horizontal: 10),
          child: SizedBox(
            width: double.infinity,
            child: ElevatedButton(
              onPressed: () {
                // Add your suggested action here
                print("Suggested button clicked");
              },
              style: ElevatedButton.styleFrom(
                backgroundColor: Color(0xFFAF7036),
                shape: RoundedRectangleBorder(

```



```
        );
    }
}
```

## Create Post Page

```
import 'dart:io';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:frontend/constants.dart';
import 'package:frontend/nav.dart';
import 'package:image_picker/image_picker.dart';
import 'package:http/http.dart' as http;
import 'package:shared_preferences/shared_preferences.dart';
import 'package:mime/mime.dart';
import 'package:http_parser/http_parser.dart';

class CreatePostScreen extends StatefulWidget {
  const CreatePostScreen({Key? key}) : super(key: key);

  @override
  _CreatePostScreenState createState() => _CreatePostScreenState();
}

class _CreatePostScreenState extends State<CreatePostScreen> {
  static final String _baseUrl = "${ApiConfig.baseUrl}/post";
  final TextEditingController _titleController = TextEditingController();
  final TextEditingController _descController = TextEditingController();
  final ImagePicker _picker = ImagePicker();

  List<File> _images = [];
  File? _video;
  bool _isLoading = false;

  Future<void> _pickImage() async {
    final pickedFile = await _picker.pickMultiImage();
    if (pickedFile.isNotEmpty) {
      setState(() {
        _images = pickedFile.map((e) => File(e.path)).toList();
      });
    }
  }
}
```

```

}

Future<void> _pickVideo() async {
    final pickedVideo = await _picker.pickVideo(source: ImageSource.gallery);
    if (pickedVideo != null) {
        setState(() {
            _video = File(pickedVideo.path);
        });
    }
}

Future<String?> _getToken() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('token');
}

Future<void> _createPost() async {
    final token = await _getToken();
    if (token == null) {
        ScaffoldMessenger.of(
            context,
        ).showSnackBar(const SnackBar(content: Text('User not authenticated')));
        return;
    }

    setState(() => _isLoading = true);

    var uri = Uri.parse('${_baseUrl}/create');
    var request =
        http.MultipartRequest('POST', uri)
            ..headers['Authorization'] = 'Bearer $token'
            ..fields['title'] = _titleController.text
            ..fields['description'] = _descController.text;

    for (var img in _images) {
        final mimeType = lookupMimeType(img.path);
        final mediaType =
            mimeType != null
                ? MediaType.parse(mimeType)
                : MediaType('image', 'jpeg');
        request.files.add(
            await http.MultipartFile.fromPath(
                'image',
                img.path,
                contentType: mediaType,
            )
        );
    }
}

```

```

        ),
    );
}

if (_video != null) {
    final videoMimeType = lookupMimeType(_video!.path);
    final videoMediaType =
        videoMimeType != null
            ? MediaType.parse(videoMimeType)
            : MediaType('video', 'mp4');
    request.files.add(
        await http.MultipartFile.fromPath(
            'video',
            _video!.path,
            contentType: videoMediaType,
        ),
    );
}

var response = await request.send();
final responseData = await http.Response.fromStream(response);

setState(() => _isLoading = false);

if (response.statusCode == 201) {
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Post created successfully')),
    );
    Navigator.pushAndRemoveUntil(
        context,
        MaterialPageRoute(builder: (context) => const NavBar()),
        (route) => false,
    );
} else {
    if (kDebugMode) {
        print('Backend response: ${responseData.body}');
    }
    ScaffoldMessenger.of(
        context,
    ).showSnackBar(SnackBar(content: Text('Failed: ${response.statusCode}')));
}
}

@Override
Widget build(BuildContext context) {

```

```
return Scaffold(
  backgroundColor: Colors.white,
  appBar: AppBar(
    backgroundColor: Colors.white,
    title: const Text(
      'Create New Post',
      style: TextStyle(
        color: Colors.black,
        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
  actions: [
    ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: const Color(0xFFAF7036),
        foregroundColor: Colors.white,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(8),
        ),
      ),
      onPressed: _isLoading ? null : _createPost,
      child: const Text(
        "POST",
        style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
      ),
    ),
  ],
),
body: Padding(
  padding: const EdgeInsets.all(16),
  child: SingleChildScrollView(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        TextField(
          controller: _titleController,
          decoration: const InputDecoration(
            hintText: "What's on your mind?",
          ),
        ),
        const SizedBox(height: 10),
        TextField(
          controller: _descController,
          maxLines: 3,
```

```

        decoration: const InputDecoration(hintText: "Description"),
    ),
    const SizedBox(height: 20),
    Wrap(
        spacing: 8,
        children:
            _images
                .map(
                    (e) => Image.file(
                        e,
                        width: 80,
                        height: 80,
                        fit: BoxFit.cover,
                    ),
                )
                .toList(),
),
if (_video != null)
    Padding(
        padding: const EdgeInsets.only(top: 10),
        child: Text(
            'Video Selected: ${_video!.path.split('/').last}',
        ),
),
Row(
    children: [
        IconButton(
            icon: const Icon(Icons.image),
            onPressed: _pickImage,
        ),
        IconButton(
            icon: const Icon(Icons.videocam),
            onPressed: _pickVideo,
        ),
    ],
),
if (_isLoading) const Center(child: CircularProgressIndicator()),
],
),
),
);
}
}

```

## Pantry Page

```
import 'package:flutter/material.dart';
import 'package:frontend/screens/Pantry/widgets/add_inge.dart';
import 'package:frontend/services/pantry.service.dart';
import 'package:frontend/widgets/button.dart';

class Pantry extends StatefulWidget {
  const Pantry({super.key, this.title});

  final String? title;

  @override
  PantryState createState() => PantryState();
}

class PantryState extends State<Pantry> {
  bool isIngredientsActive = true;

  final TextEditingController nameController = TextEditingController();
  final TextEditingController quantityController = TextEditingController();
  final TextEditingController unitController = TextEditingController();
  final TextEditingController expiryDateController = TextEditingController();
  final TextEditingController categoryController = TextEditingController();

  List<dynamic> pantryItems = [];
  bool isLoading = true;

  @override
  void initState() {
    super.initState();
    fetchPantry();
  }

  Future<void> fetchPantry() async {
    try {
      final items = await PantryService.getPantryItems();
      setState(() {
        pantryItems = items;
        isLoading = false;
      });
    } catch (e) {
      print(e);
    }
  }
}
```

```

    } catch (e) {
        setState(() => isLoading = false);
        ScaffoldMessenger.of(
            context,
        ).showSnackBar(SnackBar(content: Text('Error loading pantry items: $e')));
    }
}

void _toggleActiveTab(bool isIngredients) {
    setState(() {
        isIngredientsActive = isIngredients;
    });
}

void _showPopupBox() async {
    final result = await showDialog(
        context: context,
        barrierDismissible: false,
        builder: (context) {
            return AddIngredient(
                onClose: () => Navigator.of(context).pop(false),
                nameController: nameController,
                quantityController: quantityController,
                expiryDateController: expiryDateController,
                categoryController: categoryController,
                unitController: unitController,
            );
        },
    );
}

if (result == true) {
    fetchPantry();
}
}

@Override
Widget build(BuildContext context) {
    return Scaffold(
        backgroundColor: Colors.white,
        appBar: AppBar(
            backgroundColor: Colors.white,
            title: const Text(
                'My Pantry',
            style: TextStyle(
                color: Colors.black,

```

```
        fontSize: 20,
        fontWeight: FontWeight.bold,
    ),
),
actions: [
    IconButton(onPressed: () {}, icon: const Icon(Icons.more_vert)),
],
),
body: Column(
    children: [
        Container(
            padding: const EdgeInsets.all(10),
            child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                    Expanded(
                        child: Button(
                            text: 'Ingredients',
                            onPressed: () => _toggleActiveTab(true),
                            color:
                                isIngredientsActive
                                    ? const Color(0xFFAF7036)
                                    : const Color(0xFFD4B293),
                            txtColor: Colors.white,
                        ),
                    ),
                    const SizedBox(width: 10),
                    Expanded(
                        child: Button(
                            text: 'Recipes',
                            onPressed: () => _toggleActiveTab(false),
                            color:
                                isIngredientsActive
                                    ? const Color(0xFFD4B293)
                                    : const Color(0xFFAF7036),
                            txtColor: Colors.white,
                        ),
                    ),
                ],
            ),
        ),
        Expanded(
            child:
                isIngredientsActive
                    ? ListView(

```

```

        children: [
          Container(
            padding: const EdgeInsets.all(15),
            child: Column(
              children:
                isLoading
                  ? [
                      const Center(
                        child: CircularProgressIndicator(),
                      ),
                    ]
                  : pantryItems.map((item) {
                      return ListTile(
                        title: Text(item['name']),
                        subtitle: Text(
                          '${item['quantity']} ${item['unit']}',
                        ),
                        trailing: IconButton(
                          icon: const Icon(Icons.delete),
                          onPressed: () async {
                            await PantryService.deletePantryItem(
                              item['id'],
                            );
                            fetchPantry();
                          },
                        ),
                      );
                    });
            ),
          ],
        )
      : ListView(
        children: const [
          // Add recipe widgets here later
        ],
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(bottom: 20, left: 10, right: 10),
    child: Button(
      text: 'Add Ingredients',
      onPressed: _showPopupBox,
      color: const Color(0xFFAF7036),
      txtColor: Colors.white,
    ),
  ),

```

```
        ),
        ),
        ],
        ),
    );
}
}
```

## Profile Page

```
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:frontend/constants.dart';
import 'package:frontend/screens/Profile/post_detail.dart';
import 'package:frontend/screens/Profile/settings.dart';
import 'package:frontend/services/authServices.dart';
import 'package:frontend/services/saved_posts_notifier.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

class ProfilePage extends StatefulWidget {
    const ProfilePage({super.key});

    @override
    State<ProfilePage> createState() => _ProfilePageState();
}

class _ProfilePageState extends State<ProfilePage> {
    bool showPosts = true;
    Map<String, dynamic>? profileData;
    static final String _baseUrl = ApiConfig.baseUrl;

    @override
    void initState() {
        super.initState();
        loadProfile();
    }

    Future<void> loadProfile() async {
        final token = await AuthService.getToken();
        final response = await http.get(

```

```

        Uri.parse('${_baseUrl}/user/profile'),
        headers: {'Authorization': 'Bearer $token'},
    );

    if (response.statusCode == 200) {
        final data = json.decode(response.body);
        debugPrint('Profile data: $data');
        savedPostsNotifier.setPosts(
            List<Map<String, dynamic>>.from(data['savedPosts']),
        );
        setState(() {
            profileData = data;
        });
    }
}

@Override
Widget build(BuildContext context) {
    if (profileData == null) {
        return const Center(child: CircularProgressIndicator());
    }

    final posts = List<Map<String, dynamic>>.from(profileData!['posts']);

    return Scaffold(
        backgroundColor: Colors.white,
        appBar: AppBar(
            backgroundColor: Colors.white,
            title: const Text("Profile"),
            actions: [
                Padding(
                    padding: EdgeInsets.only(right: 10),
                    child: IconButton(
                        icon: const Icon(Icons.menu, color: Colors.black),
                        onPressed: () {
                            Navigator.of(
                                context,
                            ).push(MaterialPageRoute(builder: (context) => Settings()));
                        },
                    ),
                ),
            ],
        ),
        body: Column(
            children: [

```

```

const SizedBox(height: 10),
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 10),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      CircleAvatar(
        radius: 30,
        backgroundImage: NetworkImage(
          profileData!['profileImage'] ??
          'https://i.pravatar.cc/150?u=${profileData!['username']}'
        ),
      ),
      Text(
        profileData!['username'],
        style: const TextStyle(fontSize: 18),
      ),
    ],
  ),
),
const SizedBox(height: 20),

// Toggle buttons
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    _buildToggleButton("Posts", showPosts, () {
      setState(() => showPosts = true);
    }),
    _buildToggleButton("Saved", !showPosts, () {
      setState(() => showPosts = false);
    }),
  ],
),
const SizedBox(height: 10),

Expanded(
  child:
  showPosts
    ? _buildPostsGrid(posts)
    : ValueListenableBuilder<List<Map<String, dynamic>>>(
      valueListenable: savedPostsNotifier,
      builder: (context, savedPosts, _) {
        if (savedPosts.isEmpty) {

```

```

                return const Center(child: Text("No saved posts"));
            }
            return _buildPostsGrid(savedPosts);
        },
        ),
        ),
        ],
        ),
    );
}
}

Widget _buildPostsGrid(List<Map<String, dynamic>> posts) {
    return GridView.builder(
        padding: const EdgeInsets.all(10),
        itemCount: posts.length,
        gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 2,
            mainAxisSpacing: 10,
            crossAxisSpacing: 10,
        ),
        itemBuilder: (context, index) {
            final post = posts[index];
            final imagePaths = post['imagePaths'];

            if (imagePaths is List &&
                imagePaths.isNotEmpty &&
                imagePaths[0] != null) {
                final imageUrl = '${_baseUrl}${imagePaths[0]}';

                return GestureDetector(
                    onTap: () {
                        Navigator.push(
                            context,
                            MaterialPageRoute(builder: (_) => PostDetailScreen(post: post)),
                        );
                    },
                    child: ClipRRect(
                        borderRadius: BorderRadius.circular(10),
                        child: Image.network(imageUrl, fit: BoxFit.cover),
                    ),
                );
            } else {
                if (kDebugMode) {
                    print(

```

```
        'Skipping post at index $index because imagePaths is invalid:  
$imagePaths',  
        );  
    }  
    return const SizedBox(); // Empty box if no image  
}  
},  
);  
}  
  
Widget _buildToggleButton(String label, bool isActive, VoidCallback onTap) {  
    return GestureDetector(  
        onTap: onTap,  
        child: Container(  
            width: 160,  
            height: 50,  
            margin: const EdgeInsets.symmetric(horizontal: 8),  
            decoration: BoxDecoration(  
                color: isActive ? const Color(0xFF9C5F2B) : Colors.brown[200],  
                borderRadius: BorderRadius.circular(12),  
            ),  
            alignment: Alignment.center,  
            child: Text(  
                label,  
                style: const TextStyle(  
                    color: Colors.white,  
                    fontSize: 18,  
                    fontWeight: FontWeight.bold,  
                ),  
            ),  
        ),  
    );  
}  
}
```

## 8.4. Appendix D: Designs

### 8.4.1. Gantt Chart

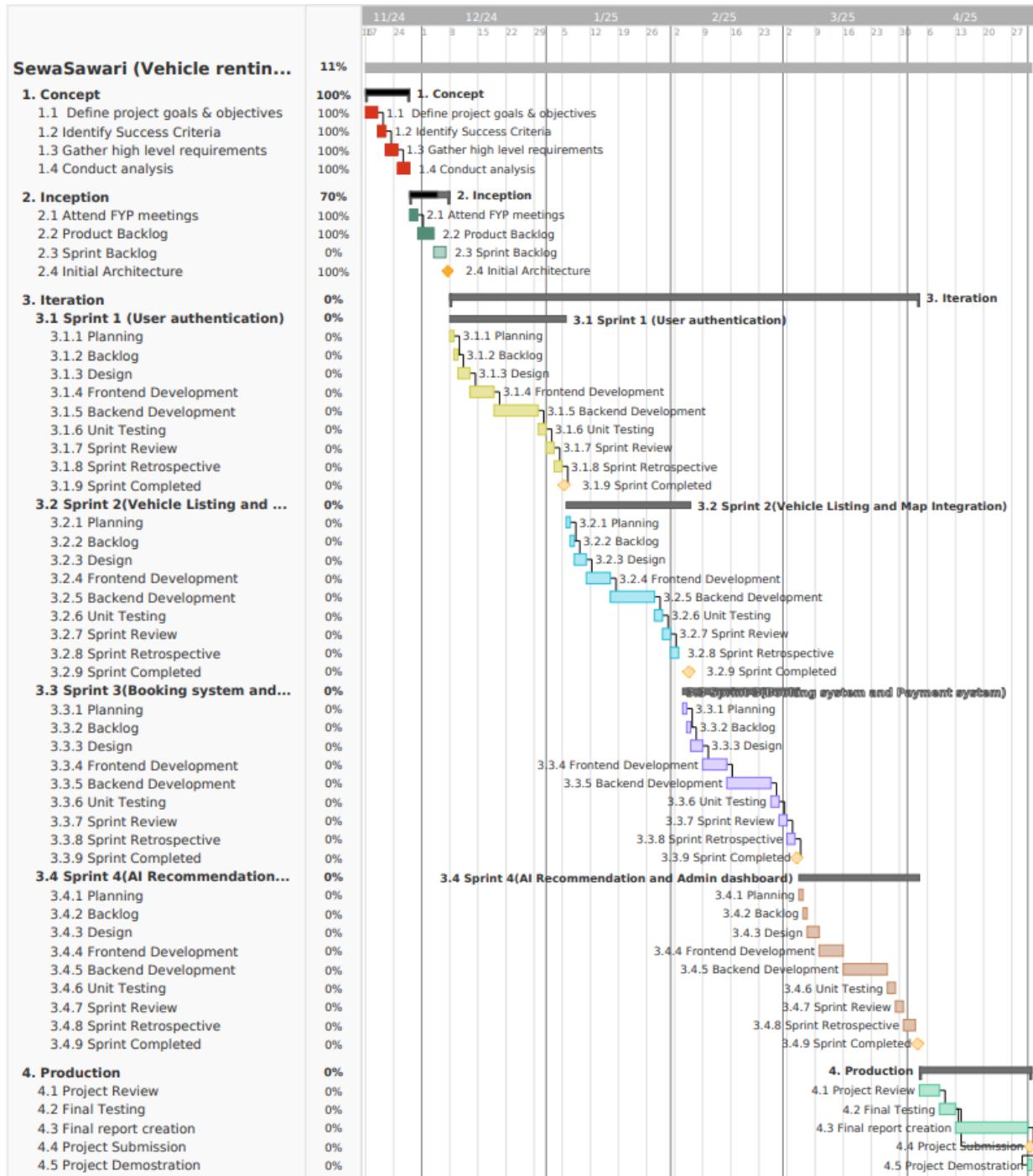


Figure 95: Gantt Chart

### 8.4.2. Work Breakdown Structure

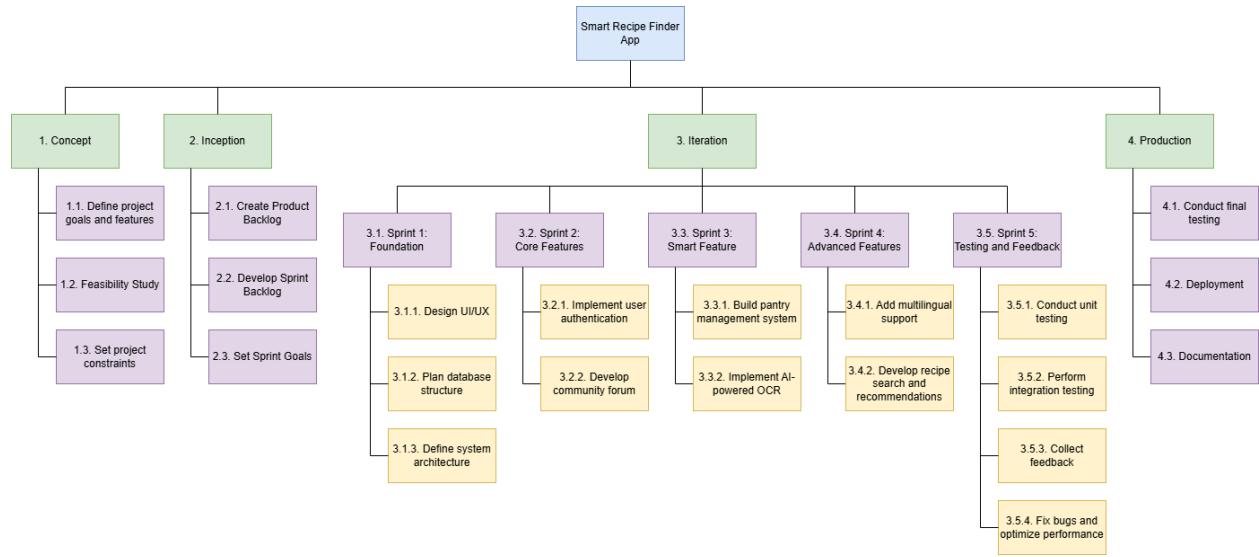


Figure 96: Work Break Structure

### 8.4.3. Flowcharts

### 8.4.4. Data Flow Diagrams

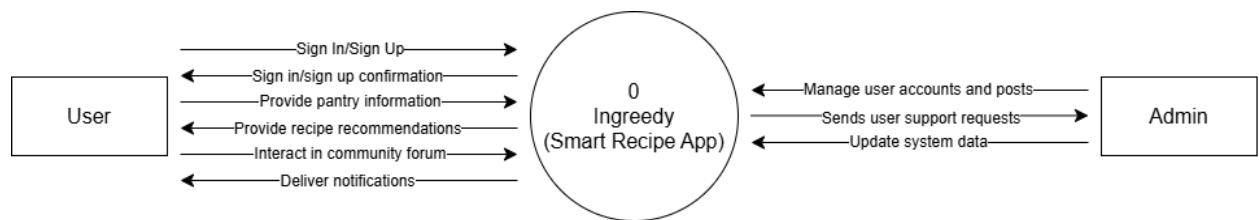


Figure 97: Data Flow Diagram

#### 8.4.5. Use Case



Figure 98: Use case diagram

#### 8.4.6. Wireframe

The wireframe depicts a registration form within a large rectangular container. At the top center, the word "Register" is displayed in a large, bold, black font. Below it, a sub-instruction reads "Create a new account to access this application". The form consists of three input fields: "Name", "Email", and "Password", each preceded by a label and followed by a light gray rectangular input box. A prominent "Register" button is located below these fields. At the bottom left, a link "Already have an account? [Log in](#)" is provided.

**Register**

Create a new account to access this application

Name

Email

Password

Register

Already have an account? [Log in](#)

Figure 99: Register Page Wireframe

The wireframe shows a large rectangular container with a thin black border. Inside, at the top center, is the word "Login" in a large, bold, black font. Below it is the text "Welcome back!" in a smaller, regular black font. To the left of the first input field is the label "Name". Below the "Name" label is a light gray rectangular input field with rounded corners. To the left of the second input field is the label "Password". Below the "Password" label is another light gray rectangular input field with rounded corners. At the bottom center is a white rectangular button with a black border and the word "Login" in black text. Below this button is the text "Don't have an account? [Register here](#)" in black font.

Figure 100: Login Page Wireframe

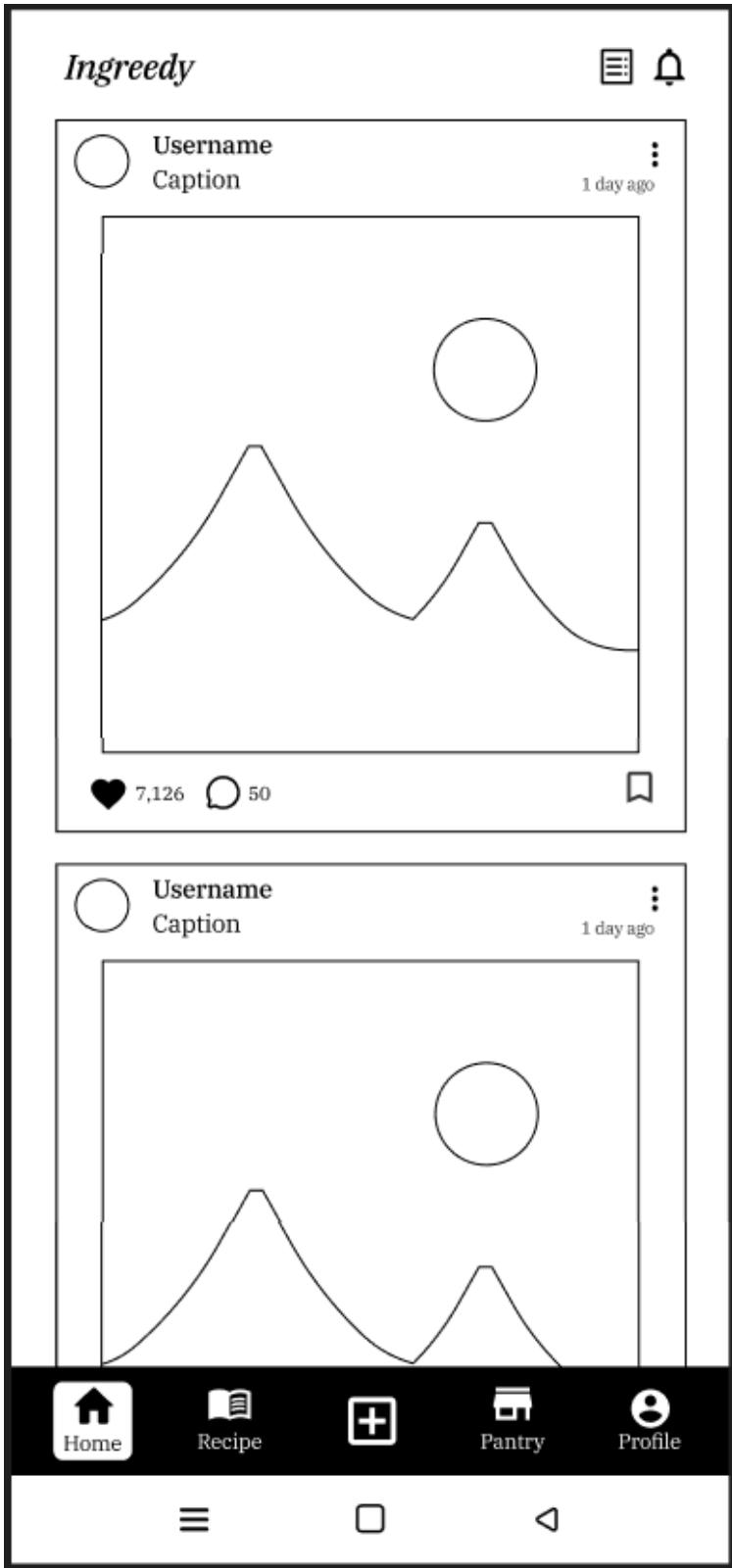


Figure 101: Home Page Wireframe

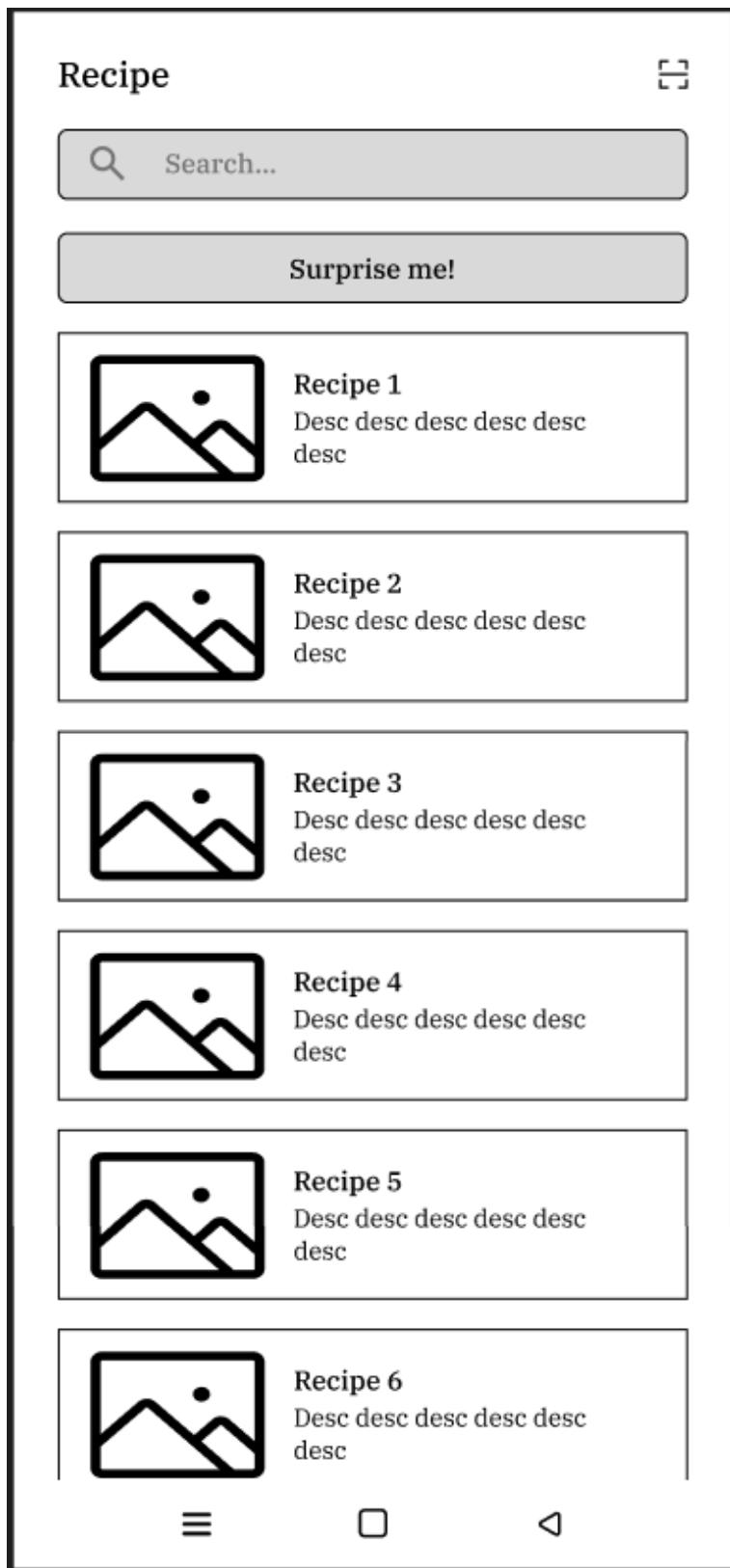


Figure 102: Recipe Page Wireframe

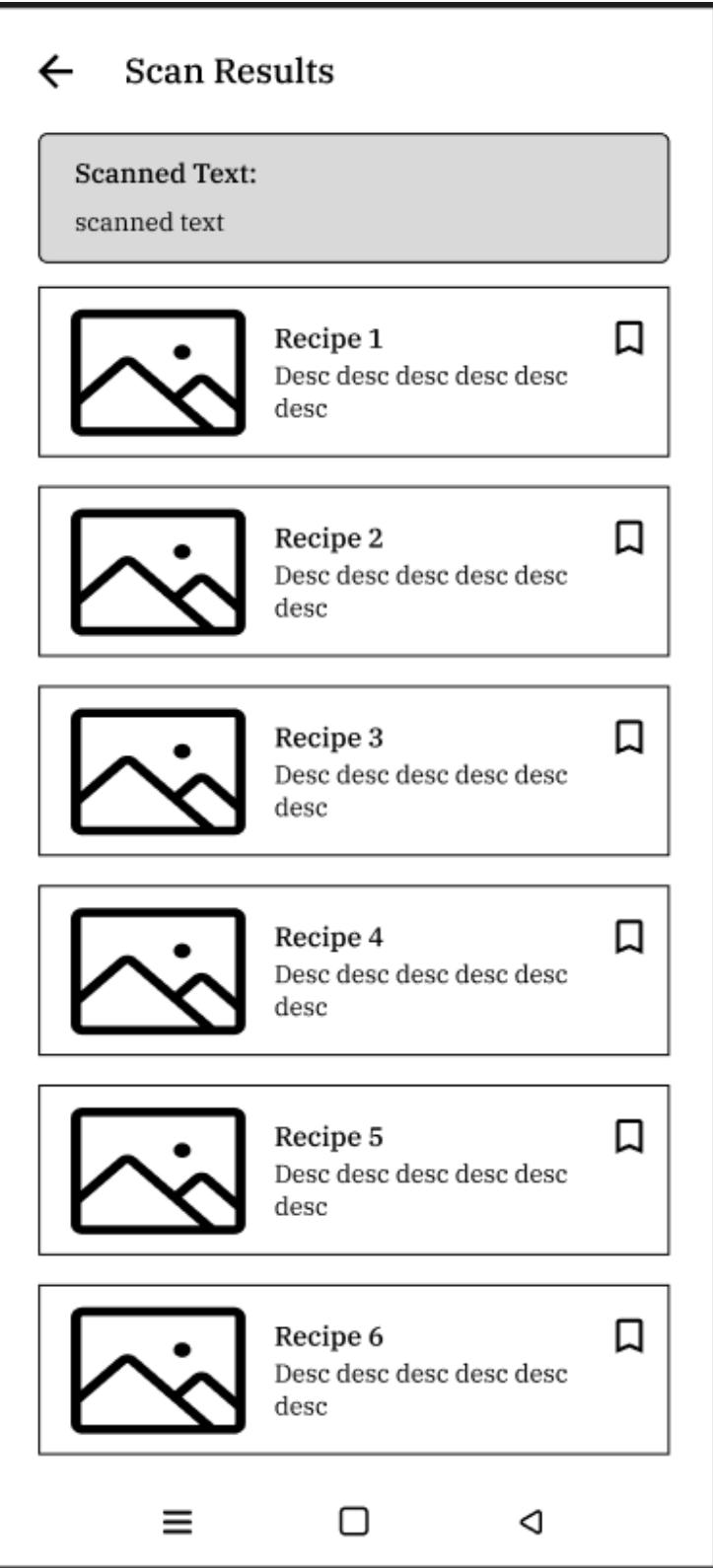


Figure 103: Scan result wireframe



Figure 104: Recipe details wireframe

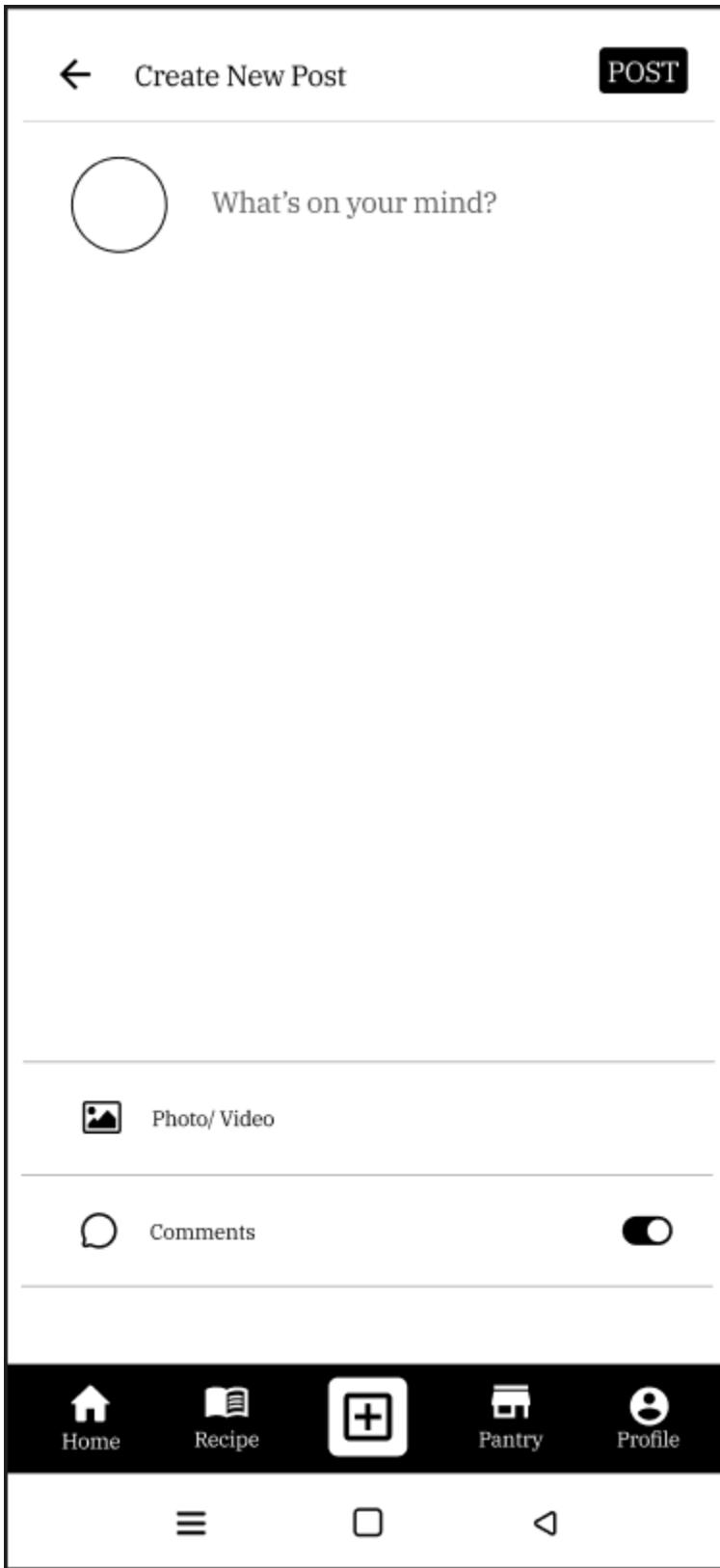


Figure 105: Create Post Wireframe



Figure 106: Pantry Page Wireframe

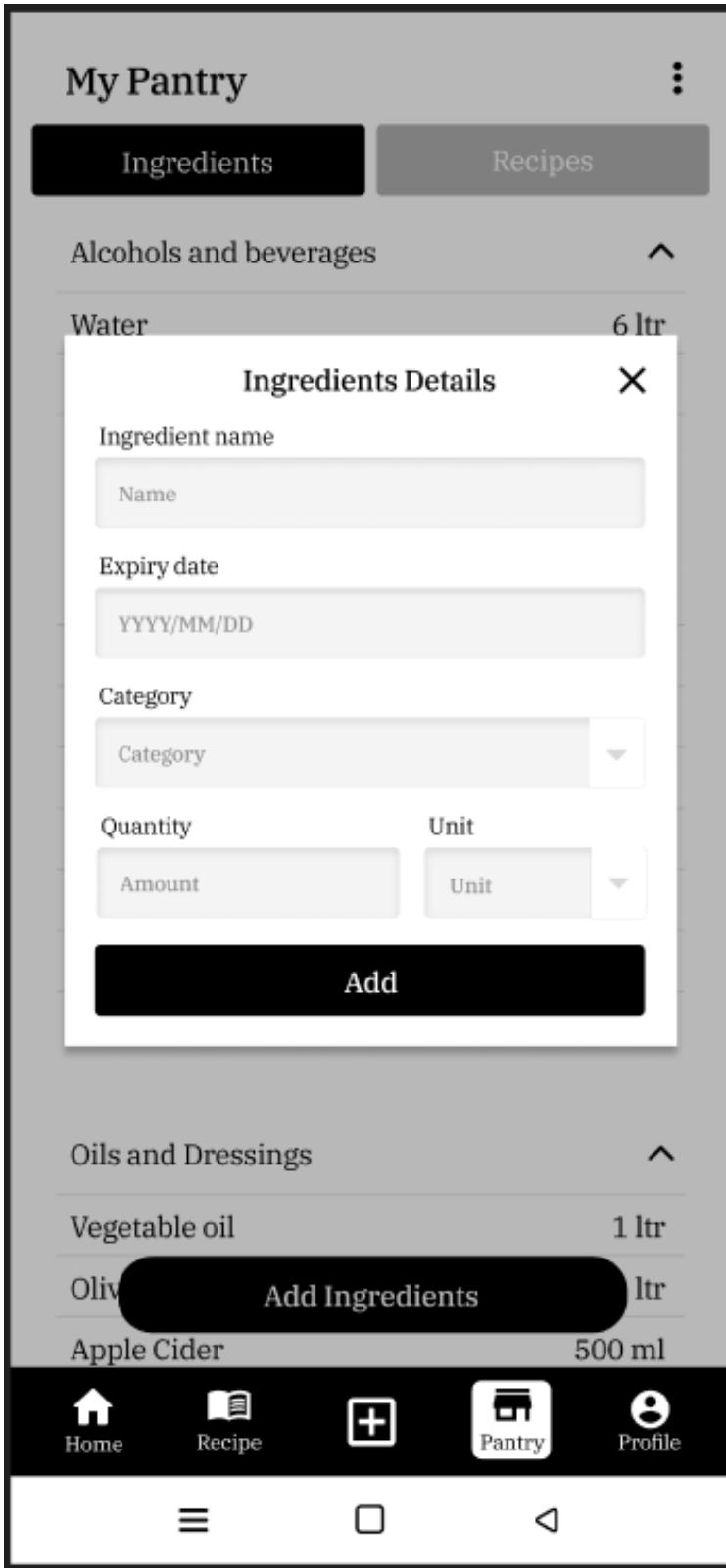


Figure 107: Add ingredient wireframe

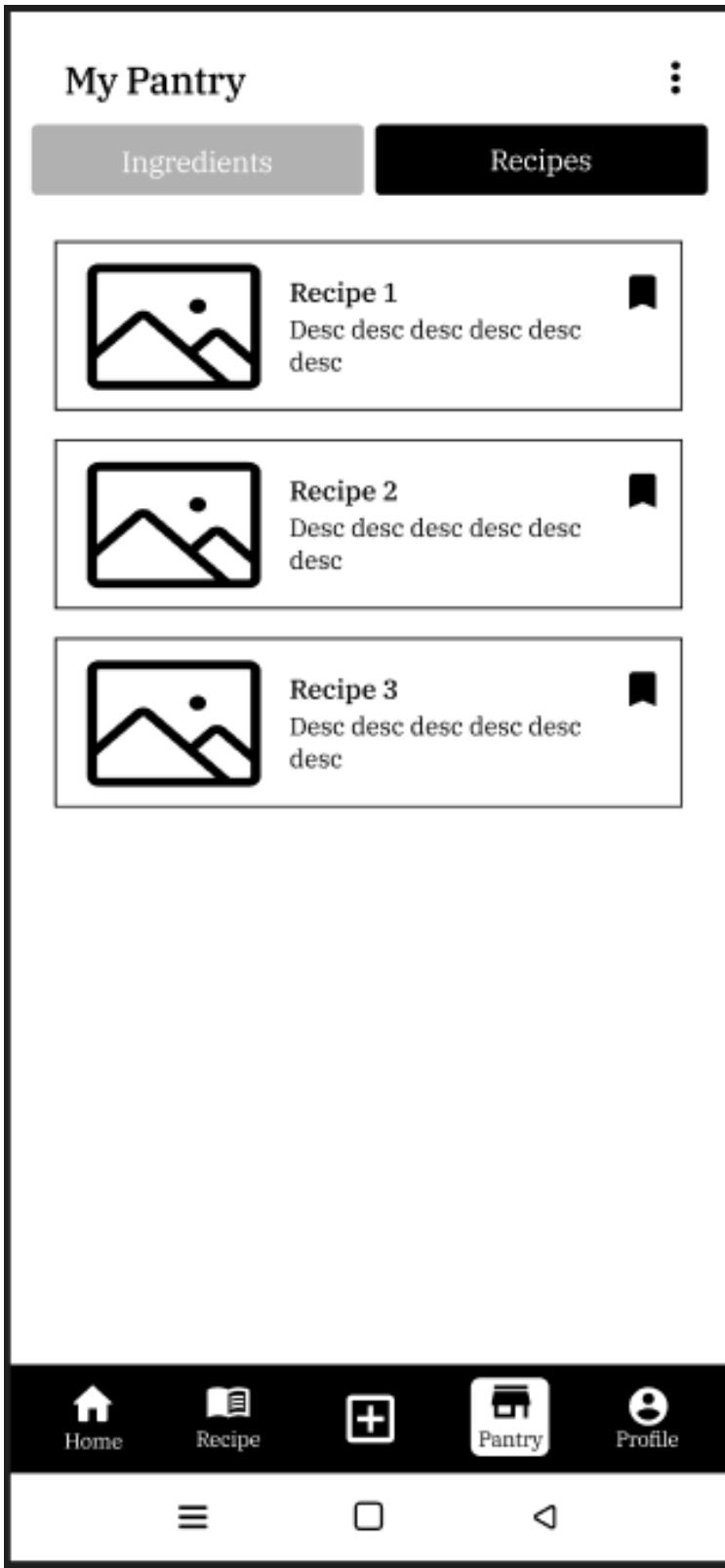


Figure 108: Saved recipes screen wireframe

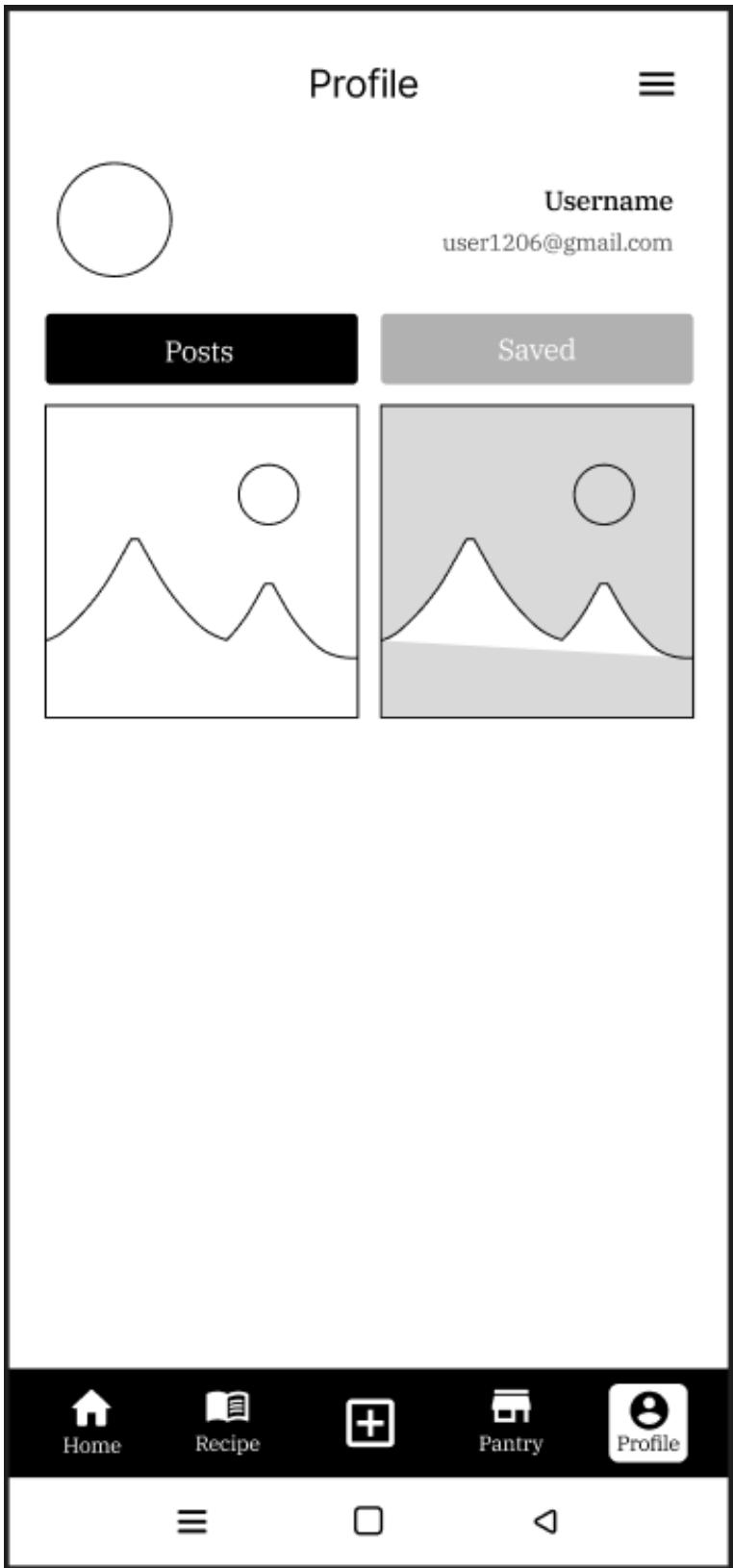


Figure 109: Profile Page Wireframe

## 8.5. Appendix E: Screenshots of The System

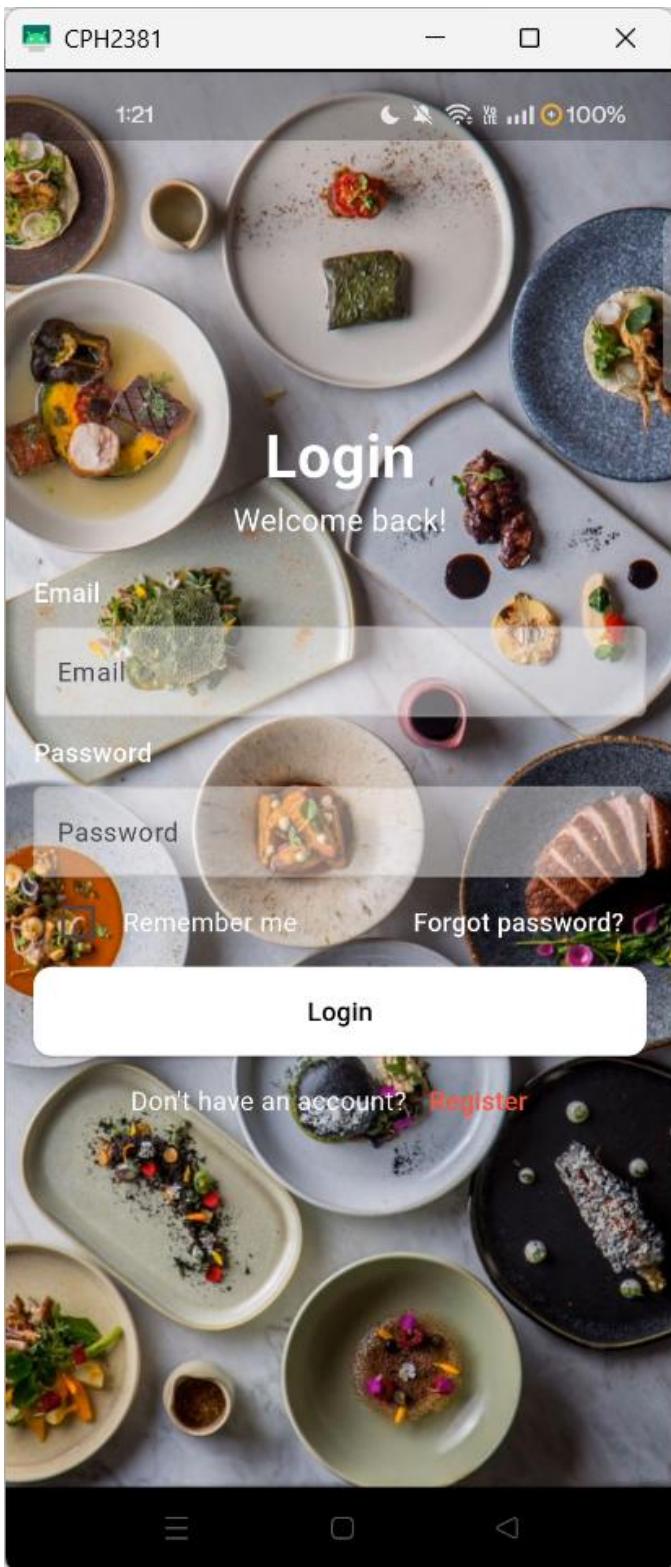


Figure 110: Login page screenshot

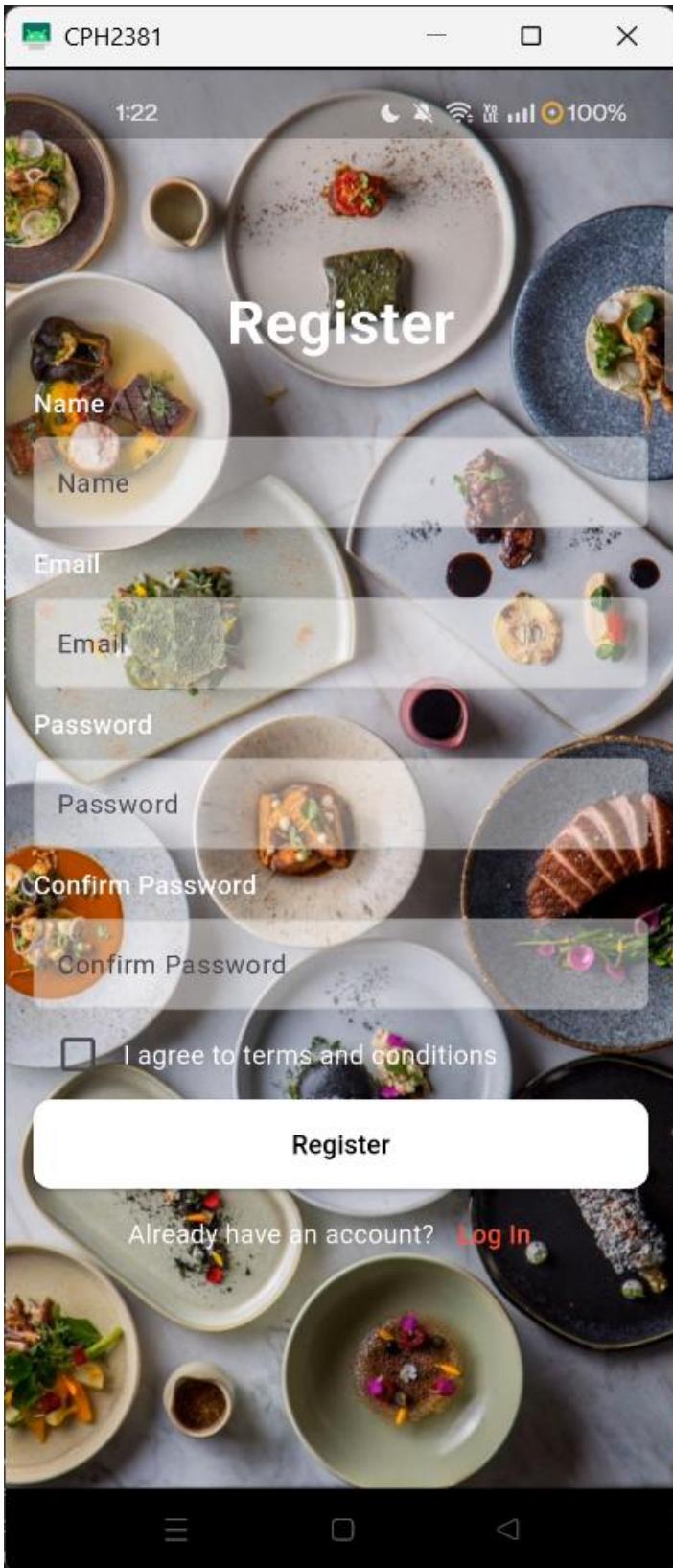


Figure III: Register page screenshot

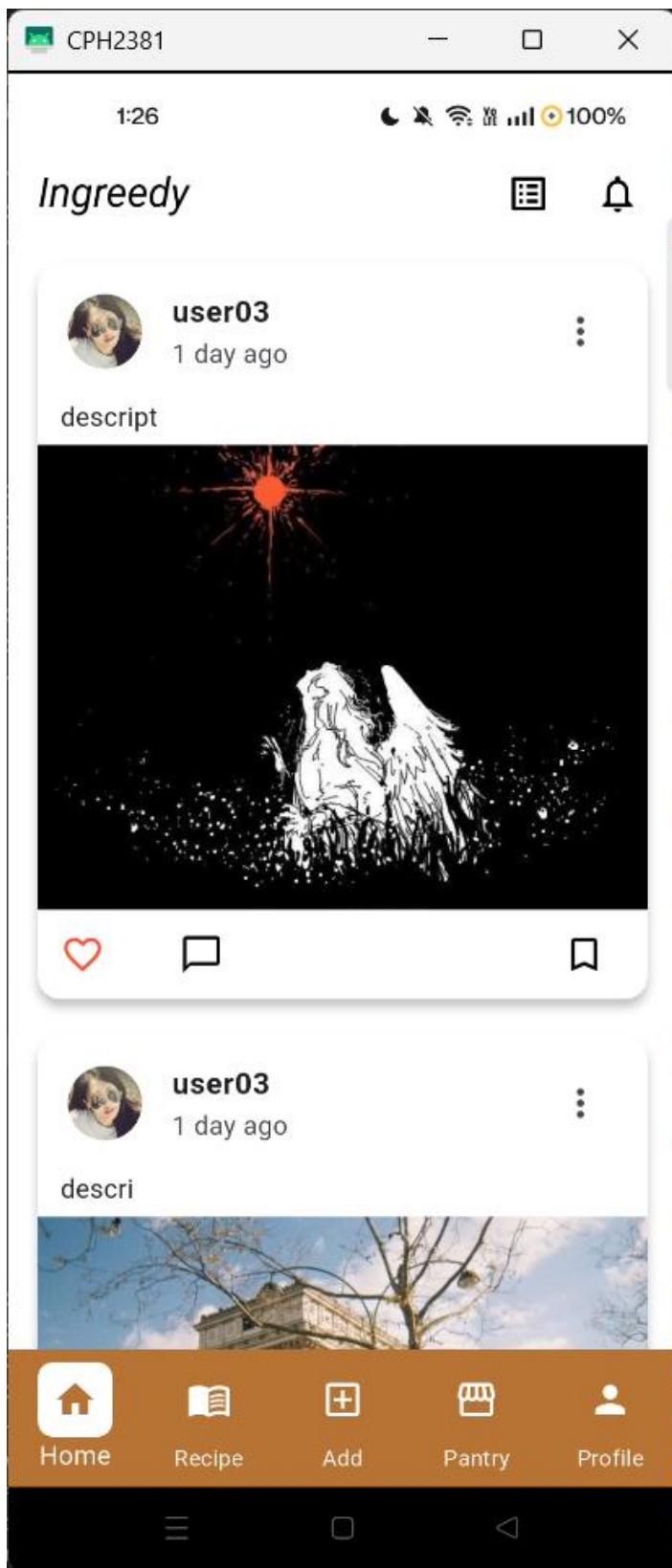


Figure 112: Home page screenshot

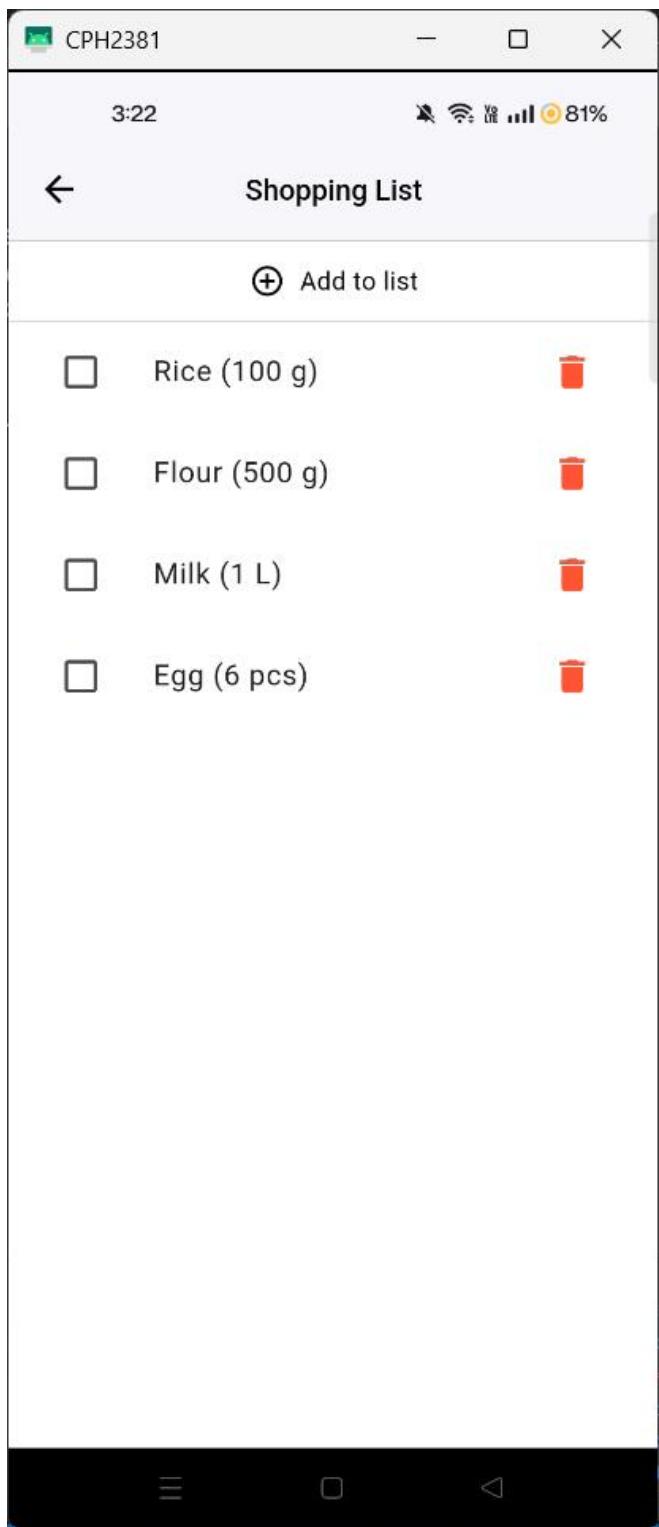


Figure 113: List page screenshot

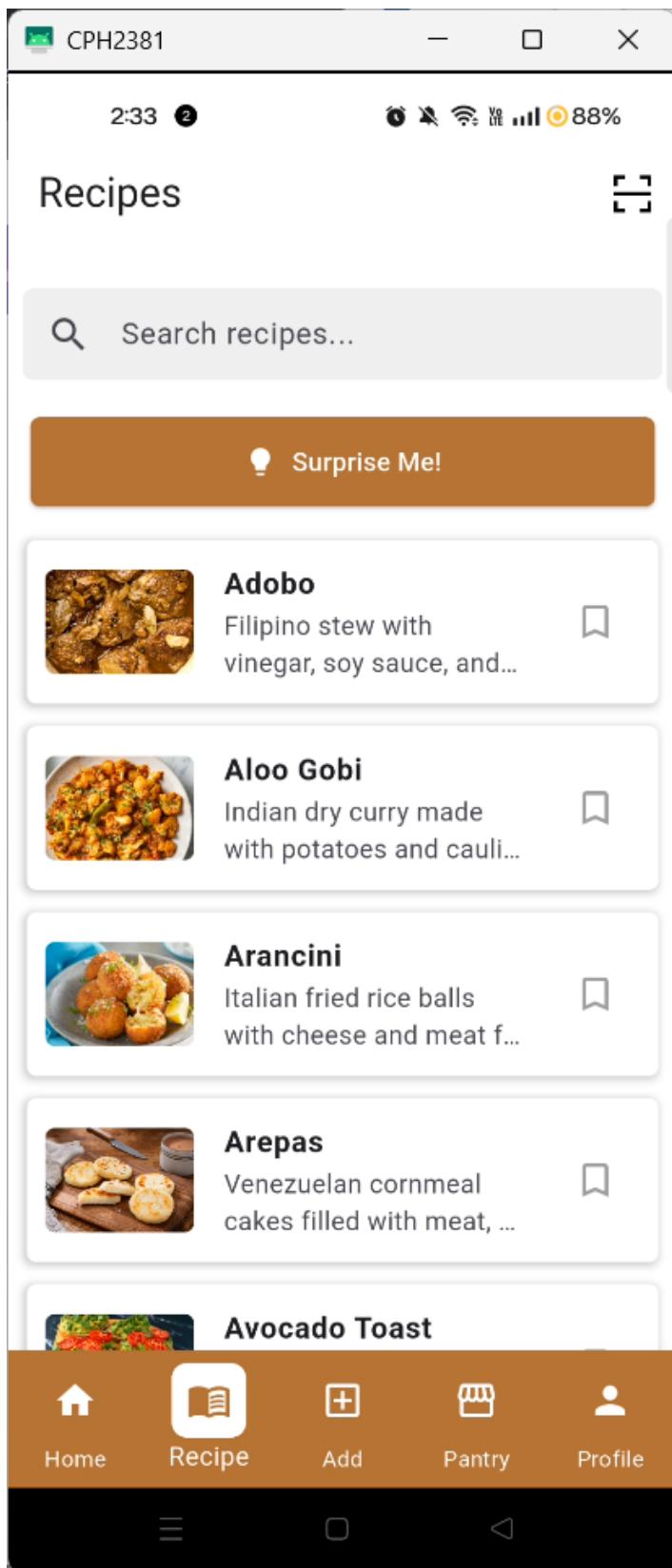


Figure 114: Recipes page screenshot

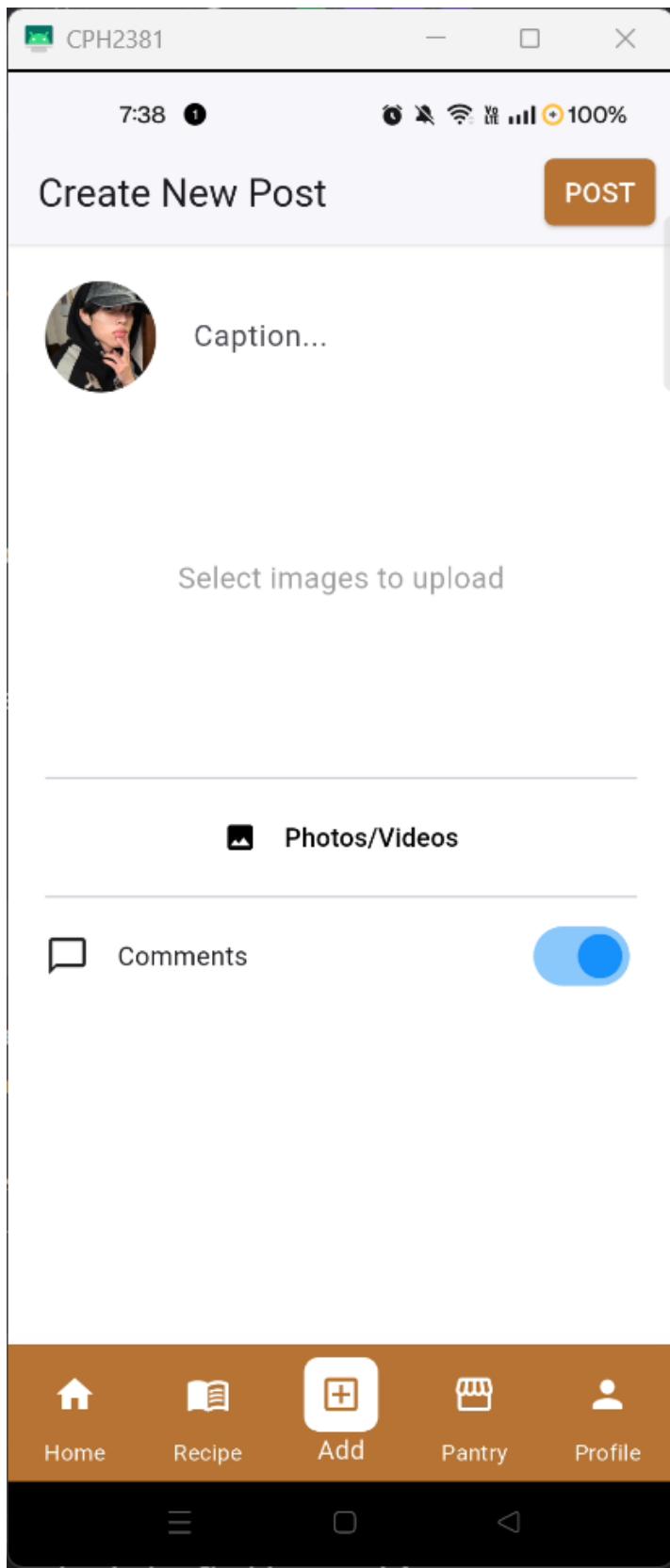


Figure 115: Create Post page screenshot

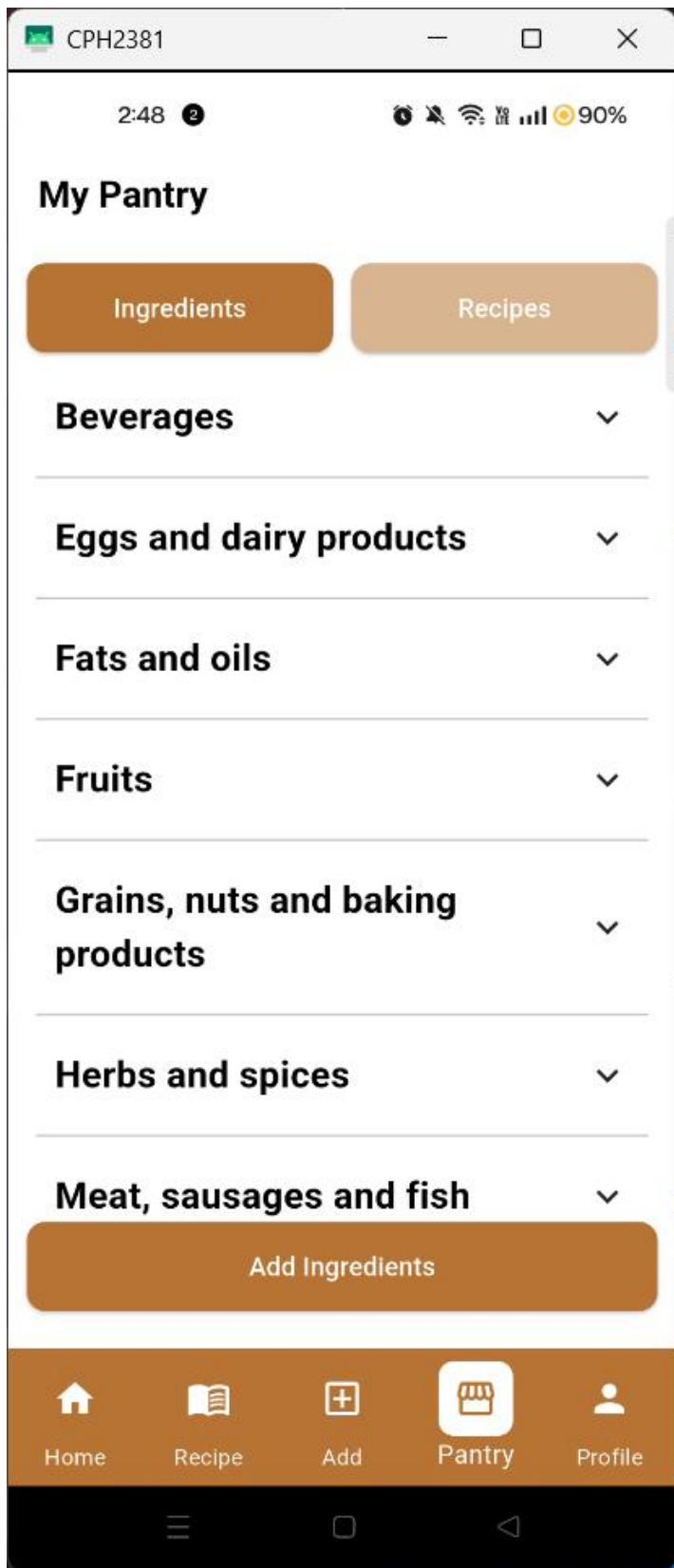


Figure 116: Pantry page screenshot

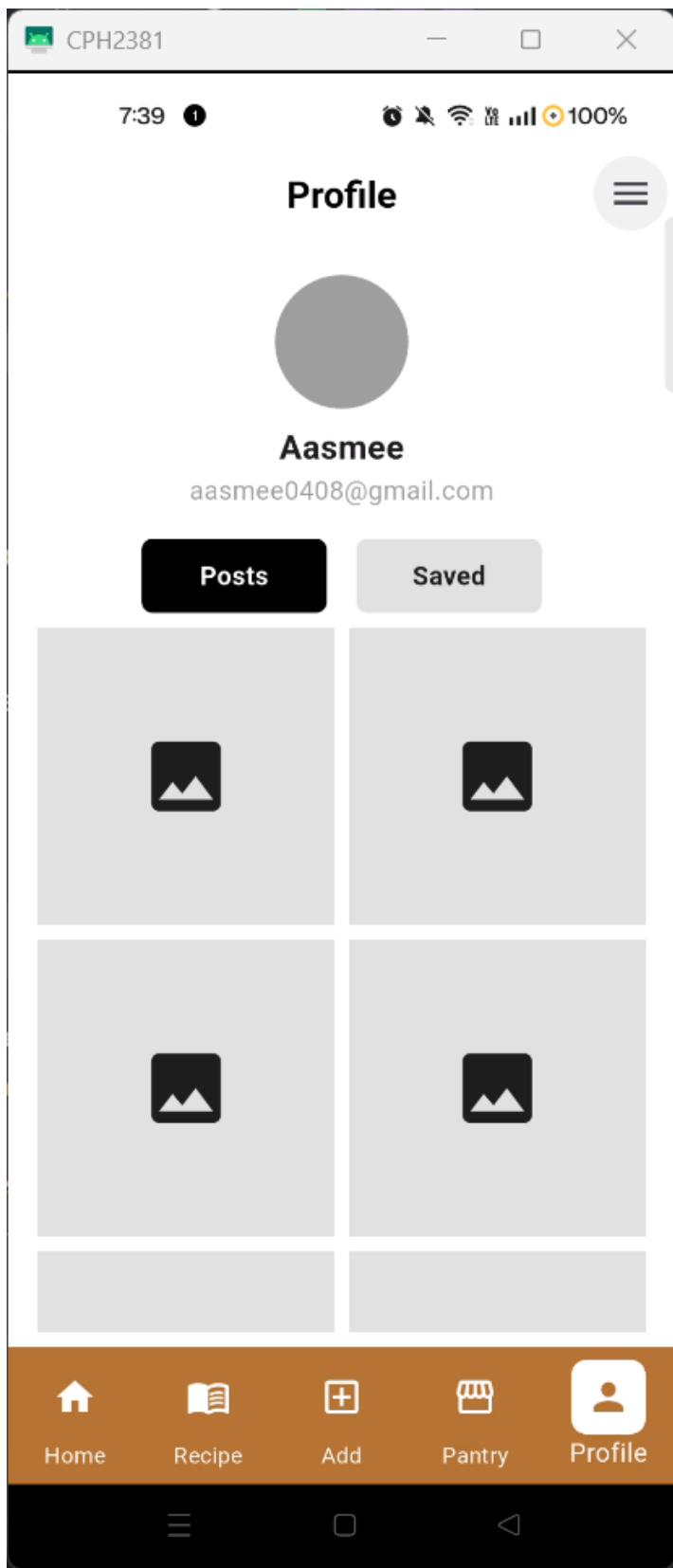


Figure 117: Profile page screenshot