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

Table of Figures

Table of Tables

Table of Abbreviations

Abbreviations	Full Form
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.

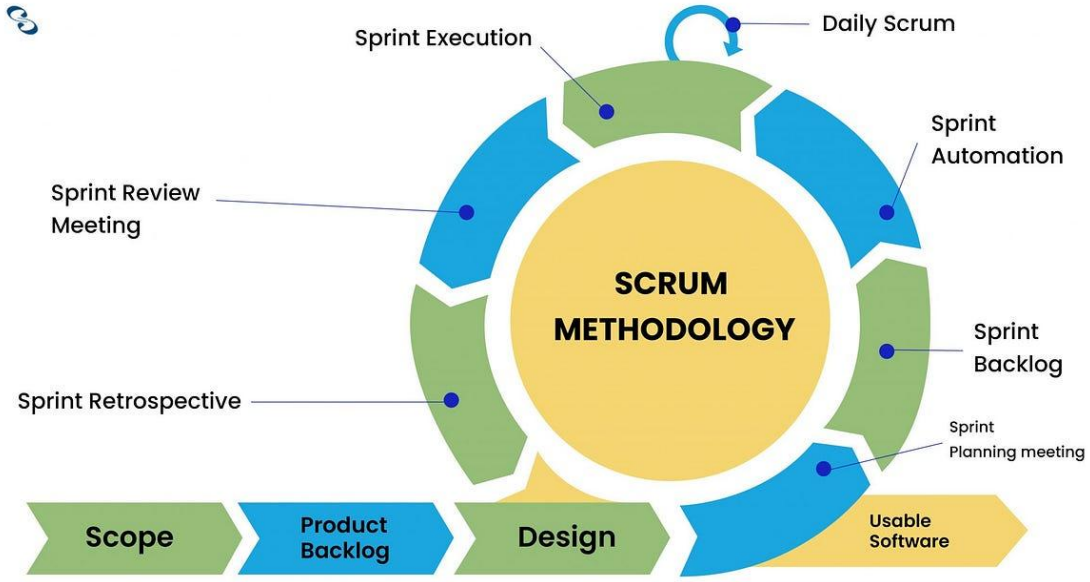


Figure 1: Scrum Methodology

3.3. Survey Results

3.3.1. Pre-Survey Results

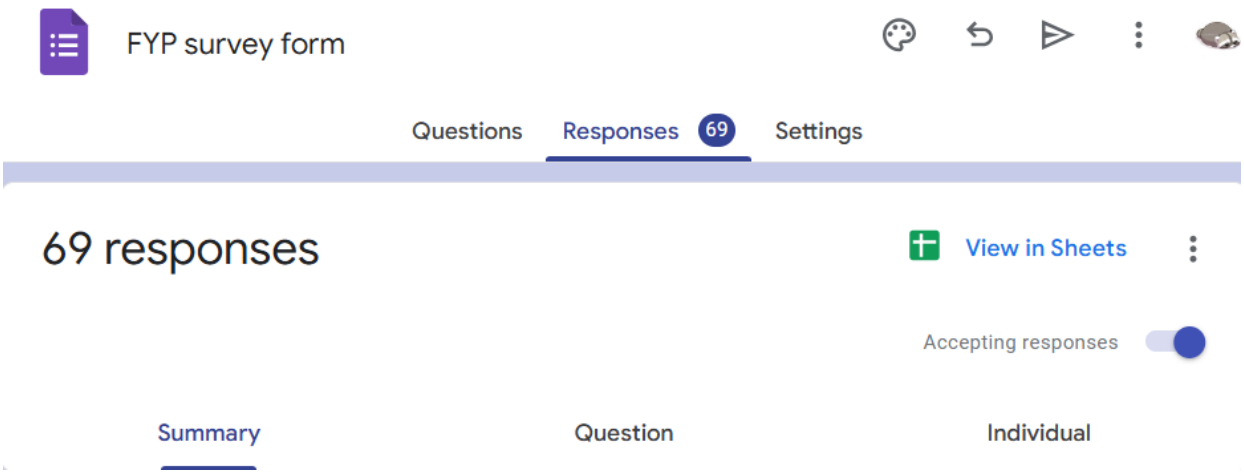


Figure 2: Total reponses accumulated in pre-survey form

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

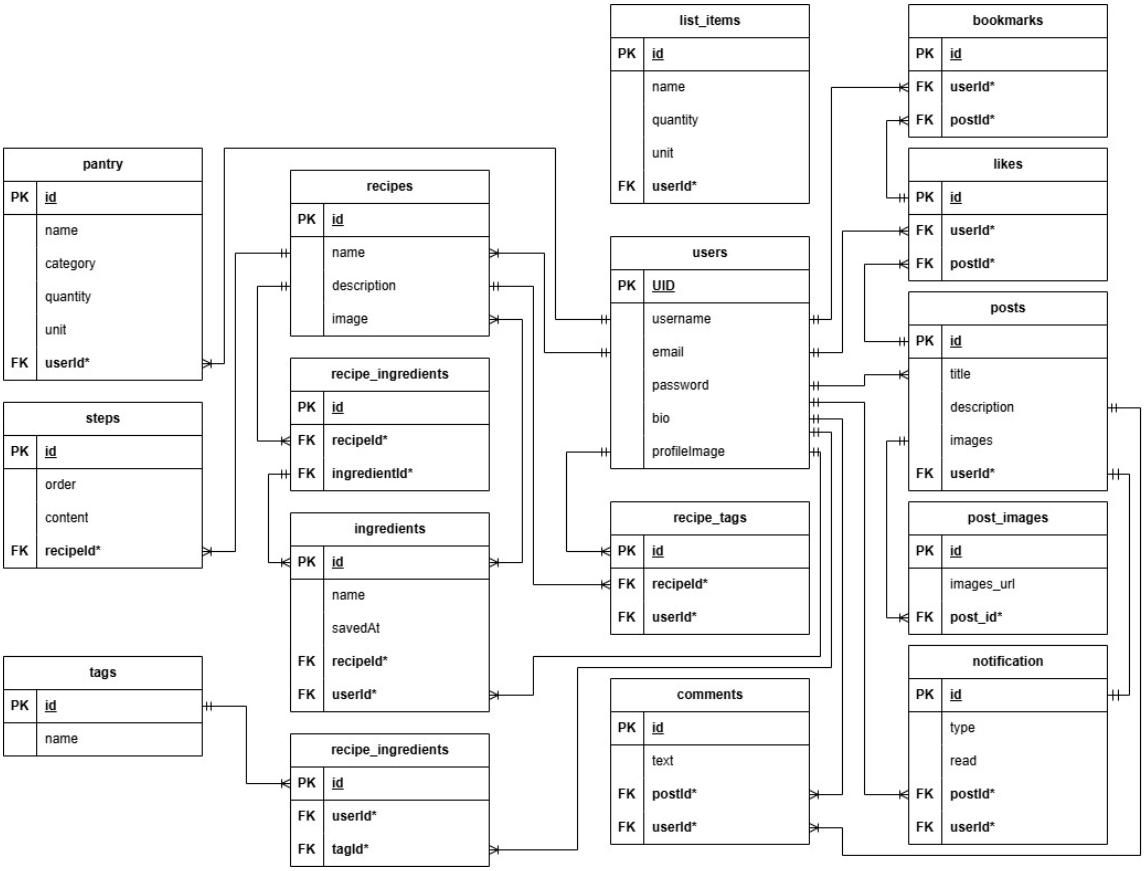


Figure 3: Entity Relationship Diagram

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. (Fakhroutdinov, 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:

Actor Action	System Response
1. Users add ingredients, manually or via OCR scanner.	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.
6. Users receive and review the notifications	5. System sends notifications about expiry and low stock.

Domain Class:

a) Ingredients
b) Pantry

Collaboration diagram:

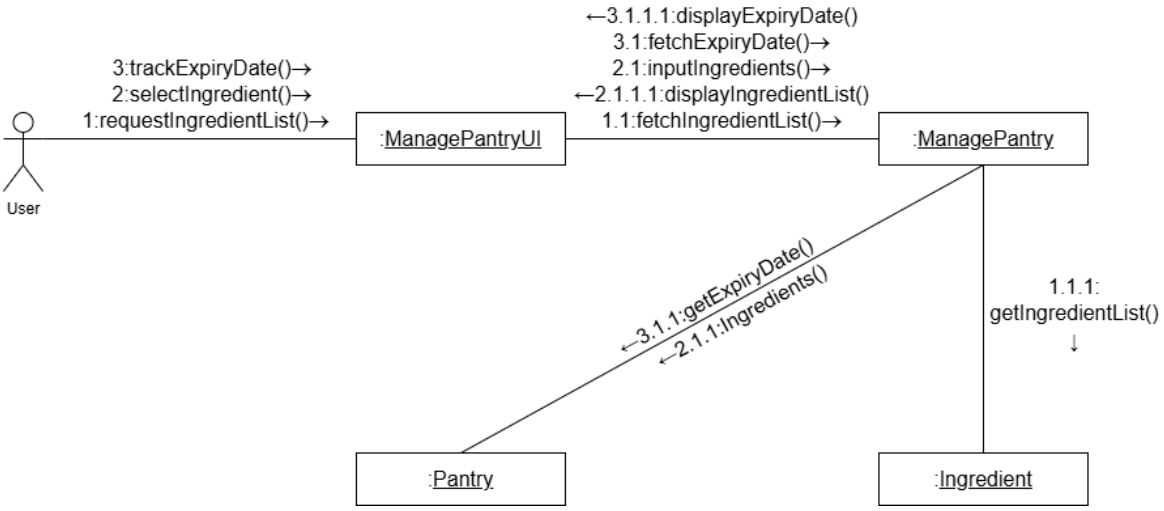


Figure 4: Collaboration diagram of "Manage Pantry"

Sequence diagram:

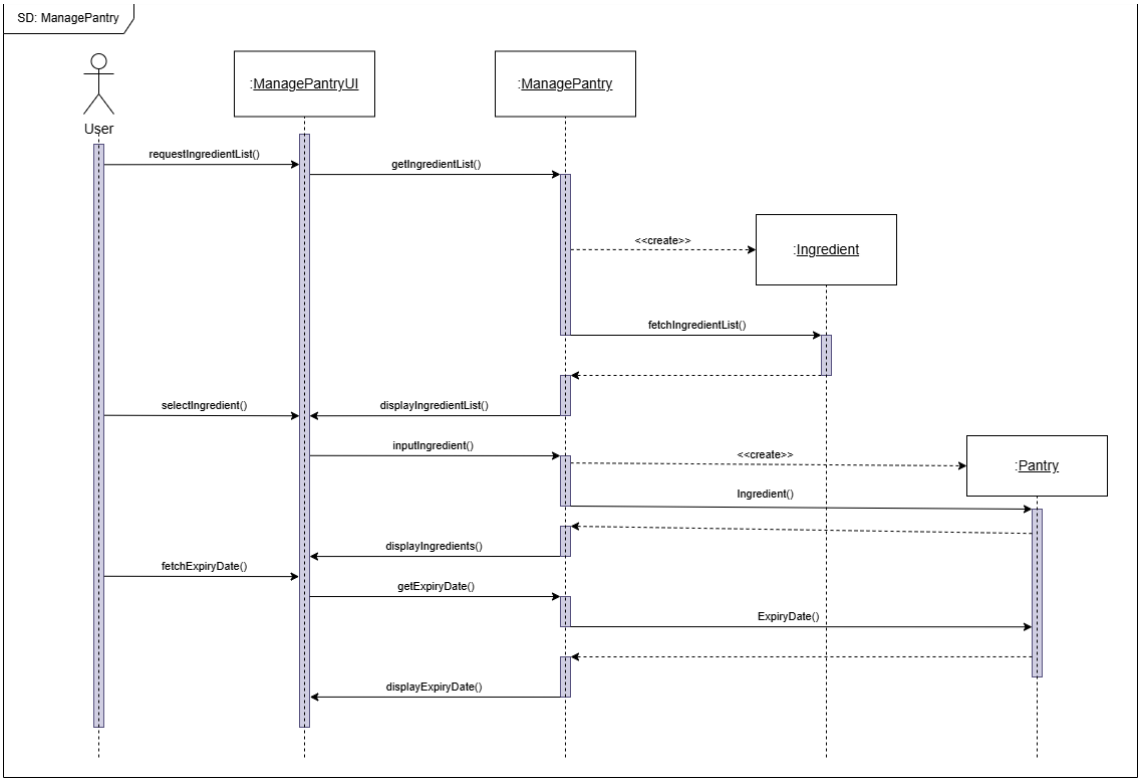


Figure 5: Sequence diagram of "Manage Pantry"

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:

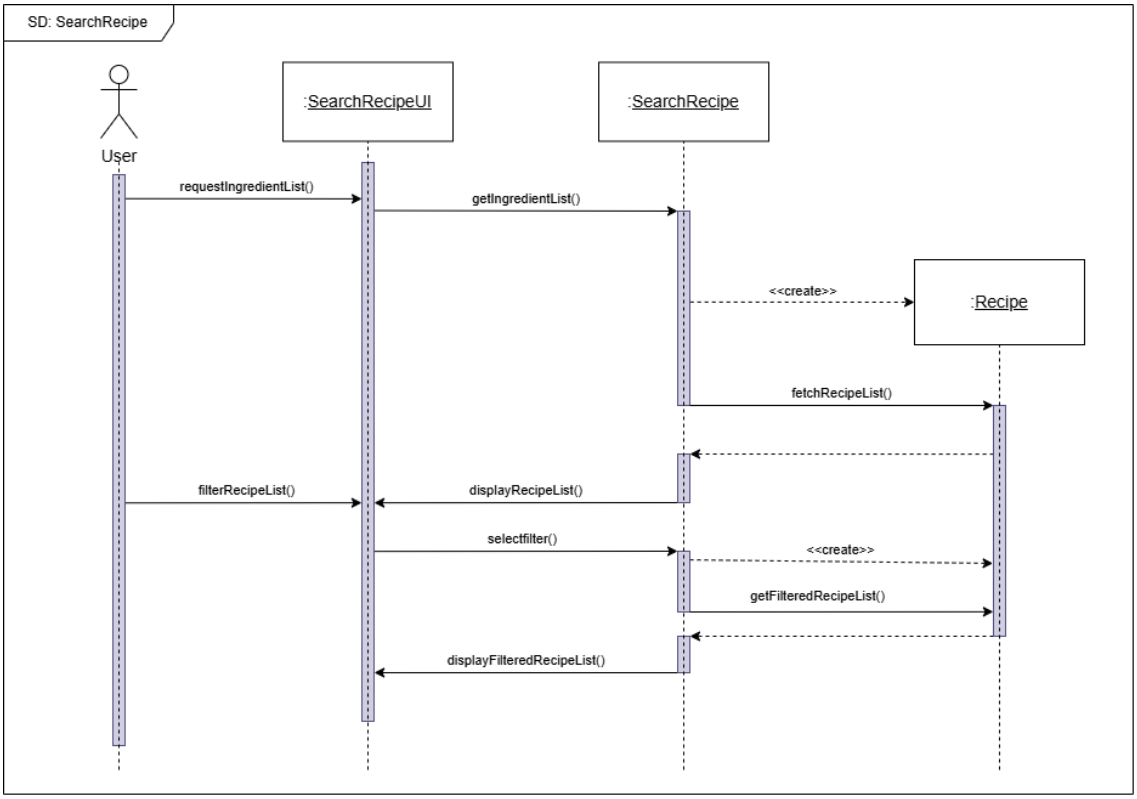


Figure 7: Sequence diagram for "Search Recipe"

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:

Actor Action	System Response
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:

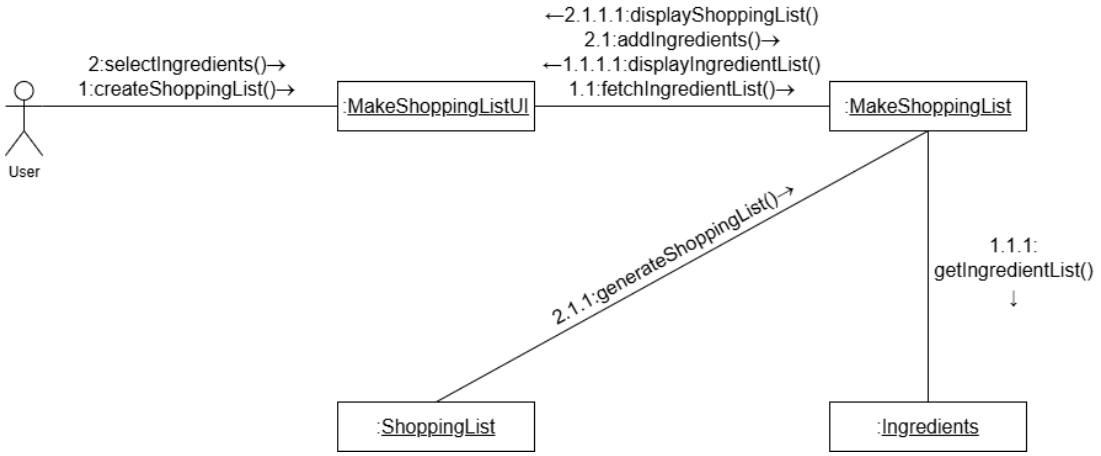


Figure 8: Collaboration of "Make Shopping List"

Sequence diagram:

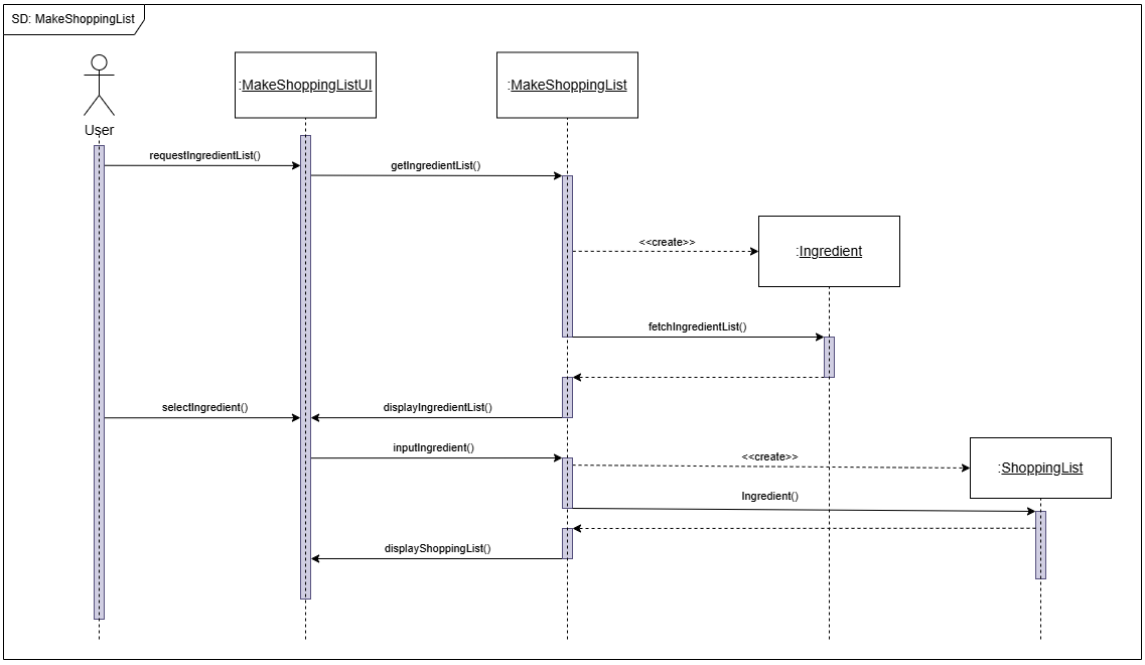


Figure 9: Sequence diagram for "Make Shopping List"

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:

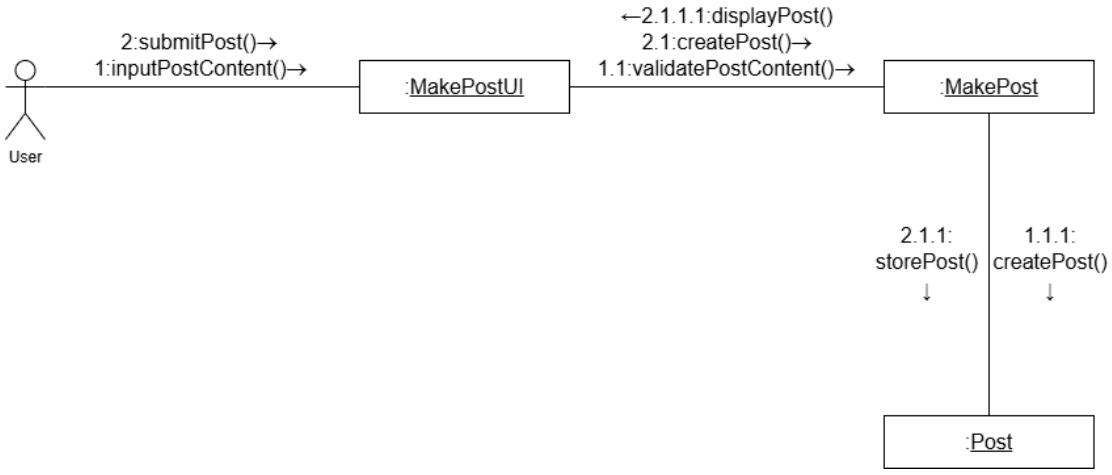


Figure 10: Collaboration of "Make Post"

Sequence diagram:

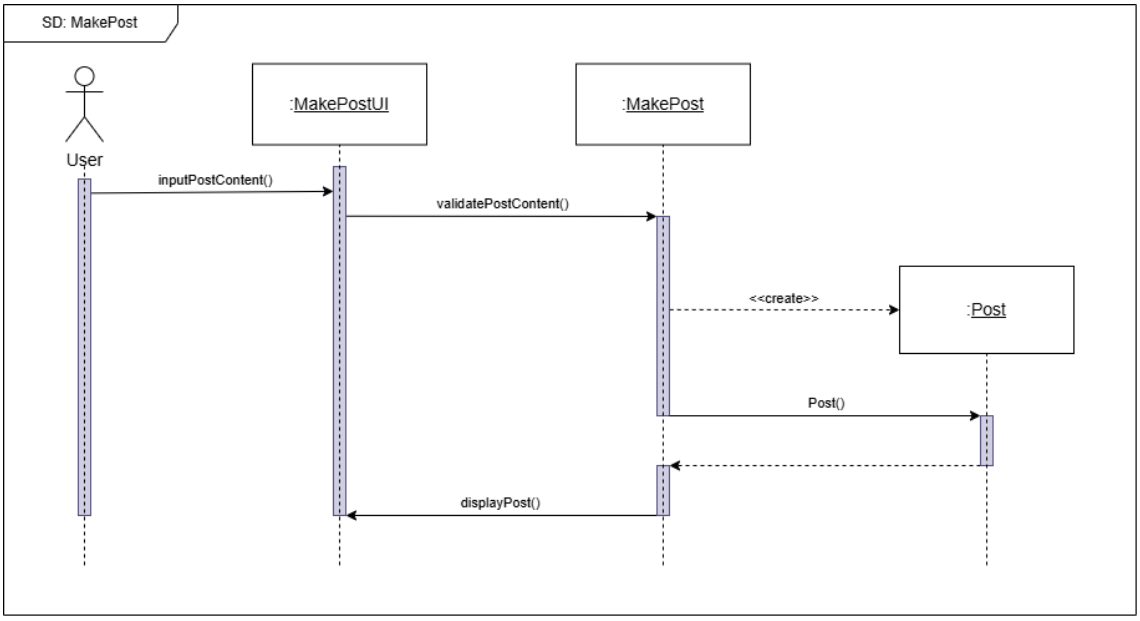


Figure 11: Sequence diagram for “Make Post”

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

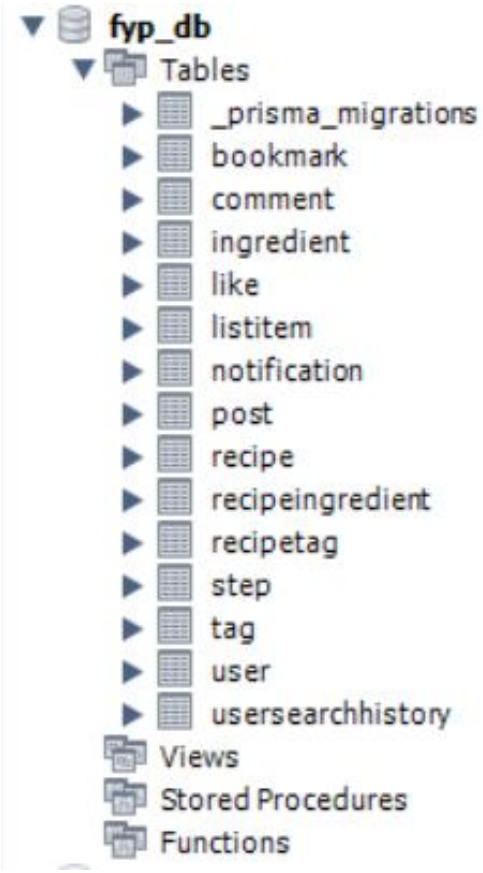


Figure 12: Database Structure

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:

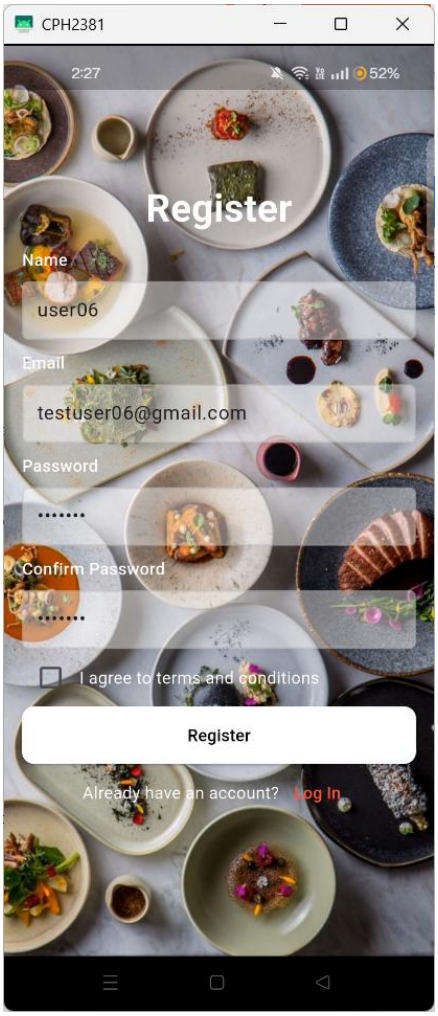


Figure 13: Registering new user

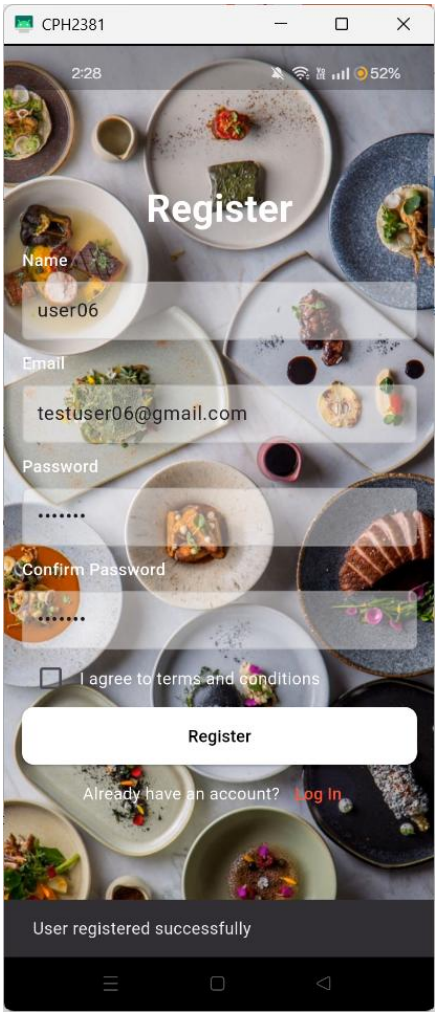


Figure 14: Login page

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:

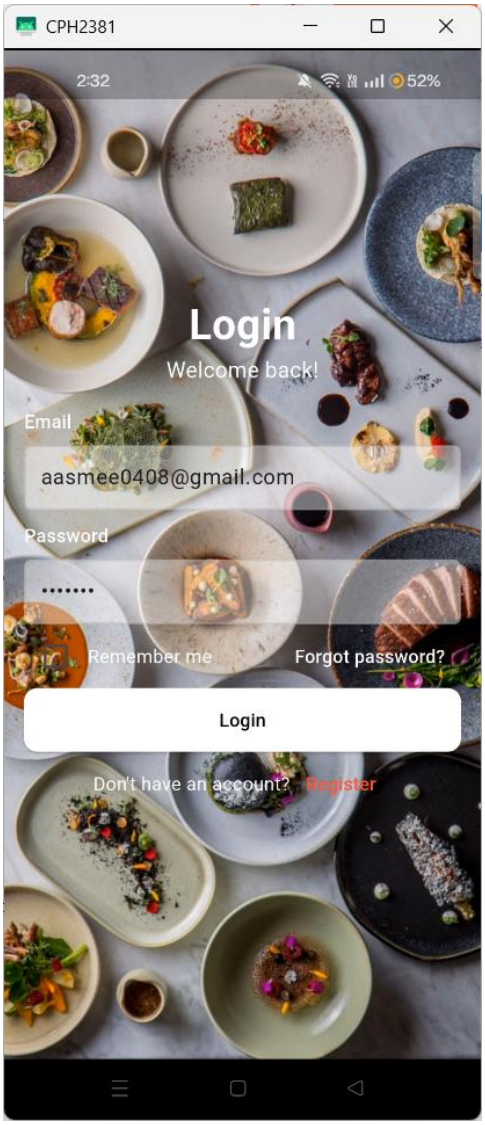


Figure 15: Entering login credentials

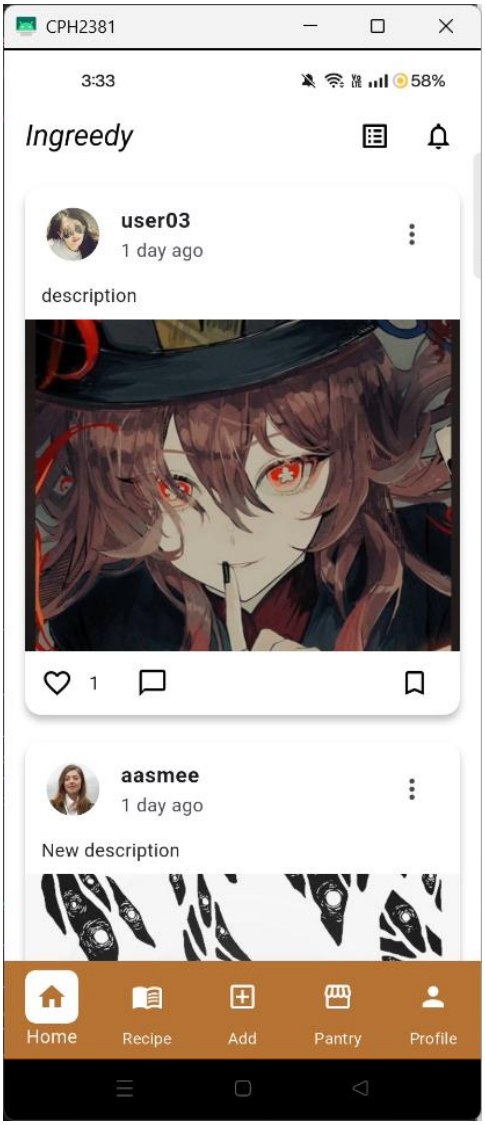


Figure 16: Home page

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:

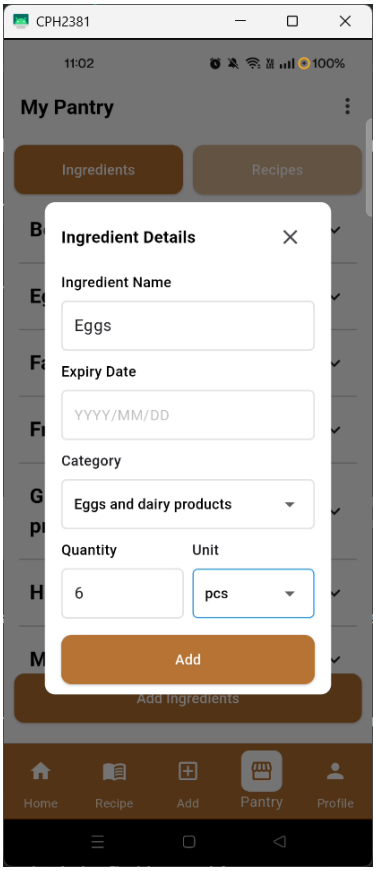


Figure 17: Adding item to pantry

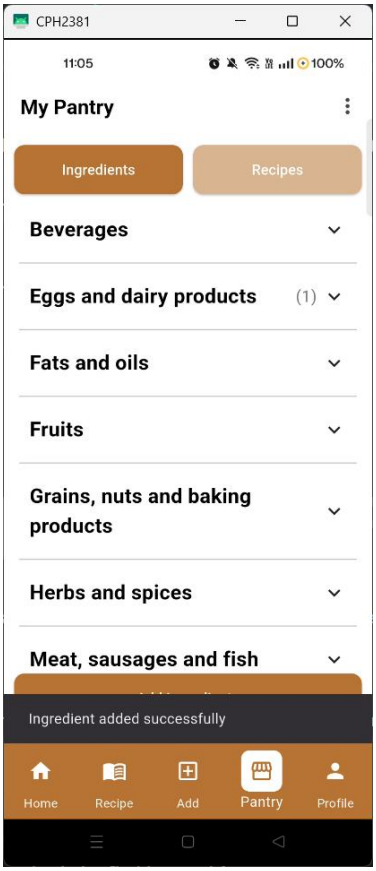


Figure 18: Succession message

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:

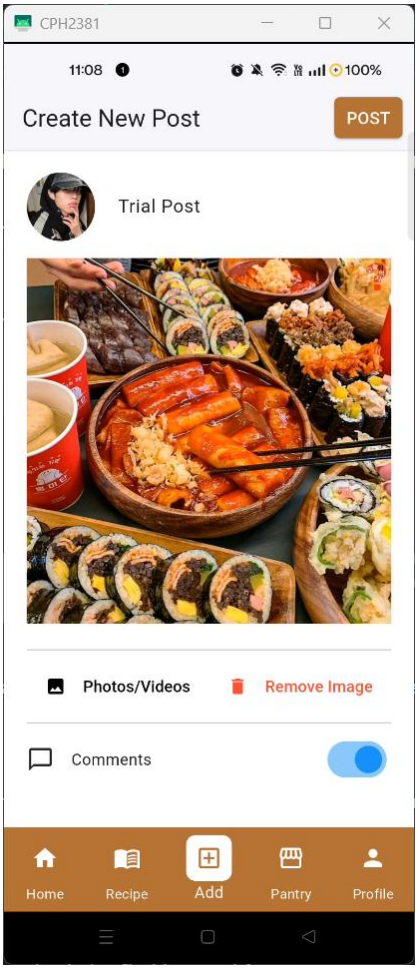


Figure 19: Creating a new post

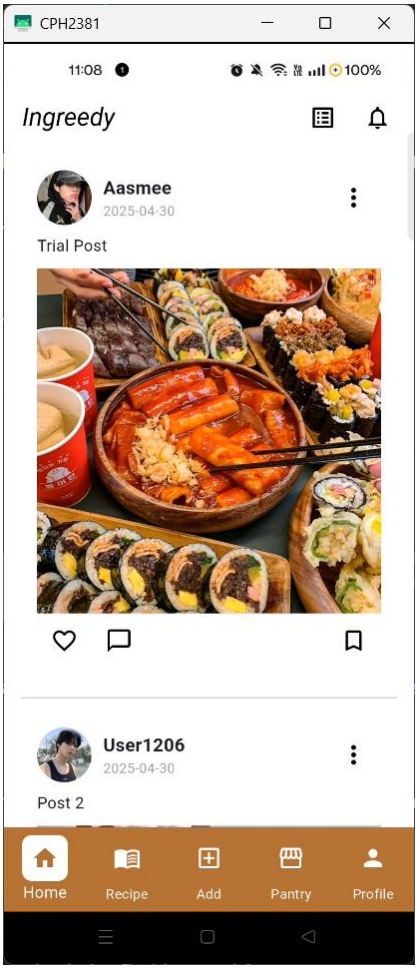


Figure 20: New post on home page

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

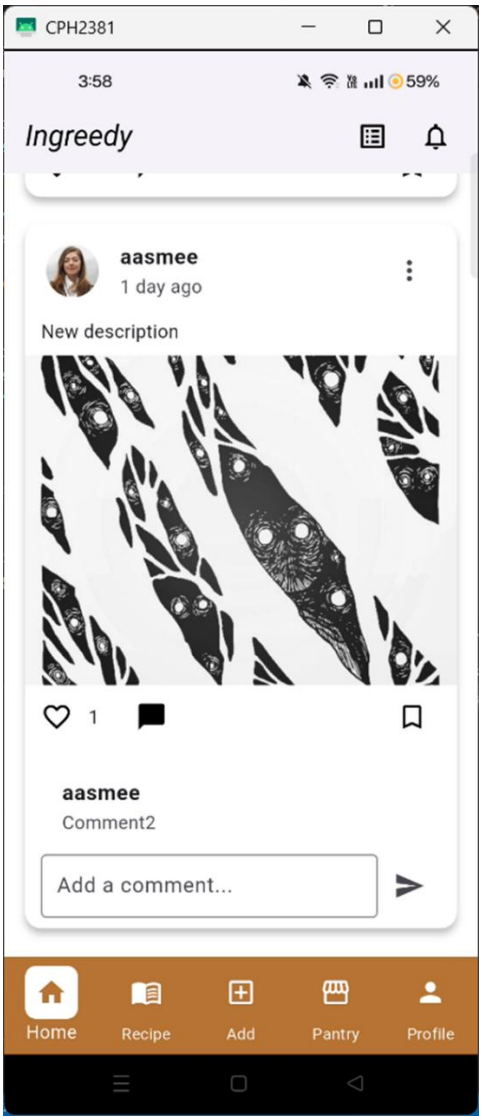


Figure 22: New comment added

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:

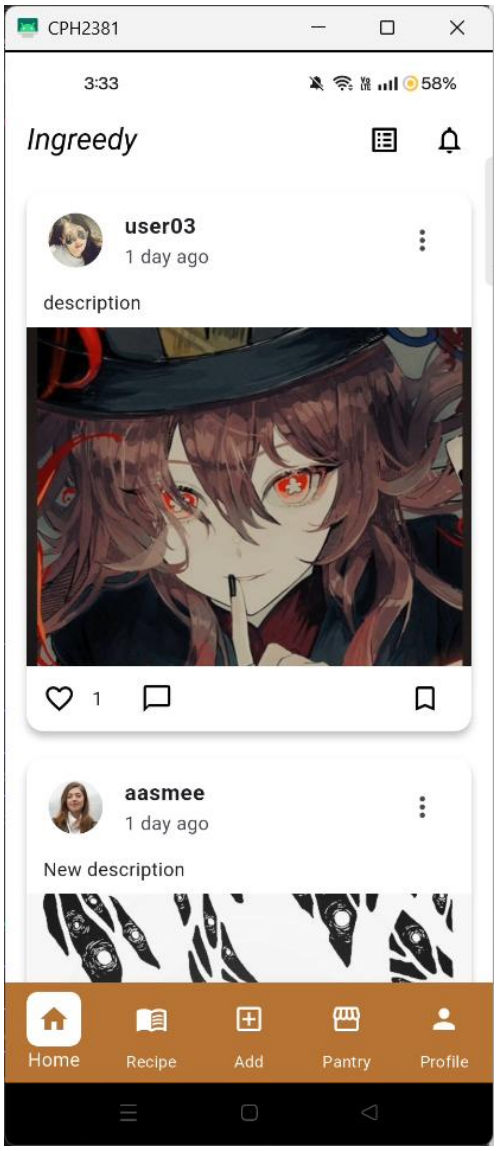


Figure 23: Post before liking

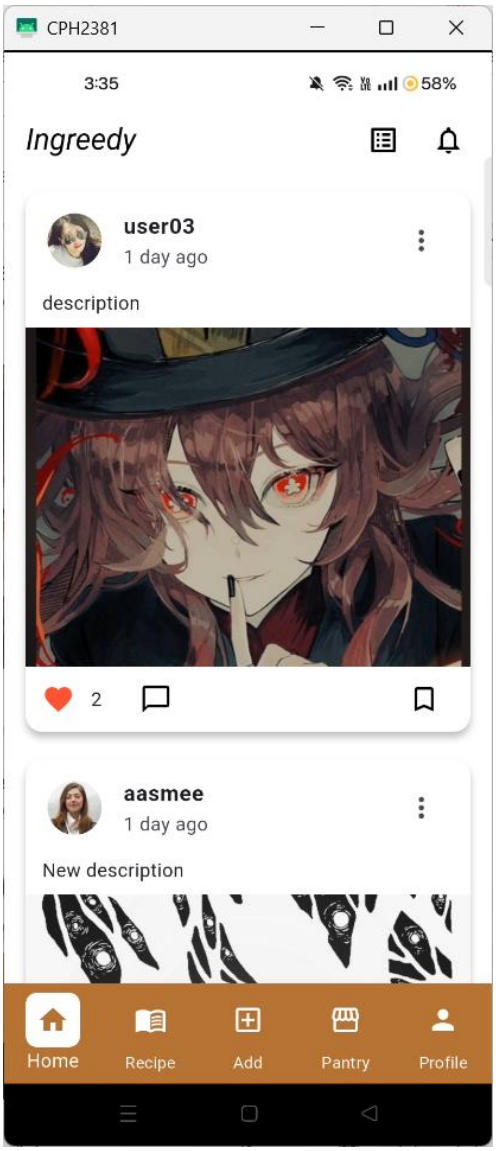


Figure 24: Post after like

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:

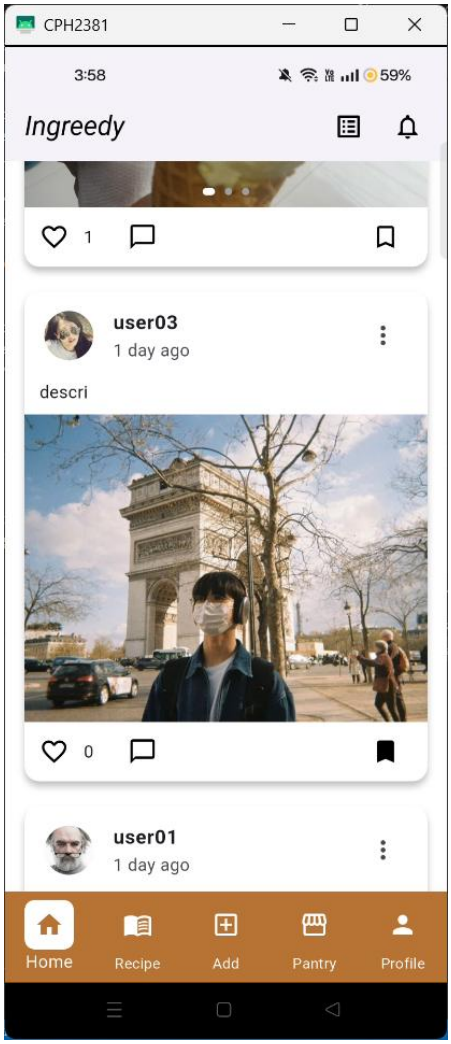


Figure 25: Bookmarking post

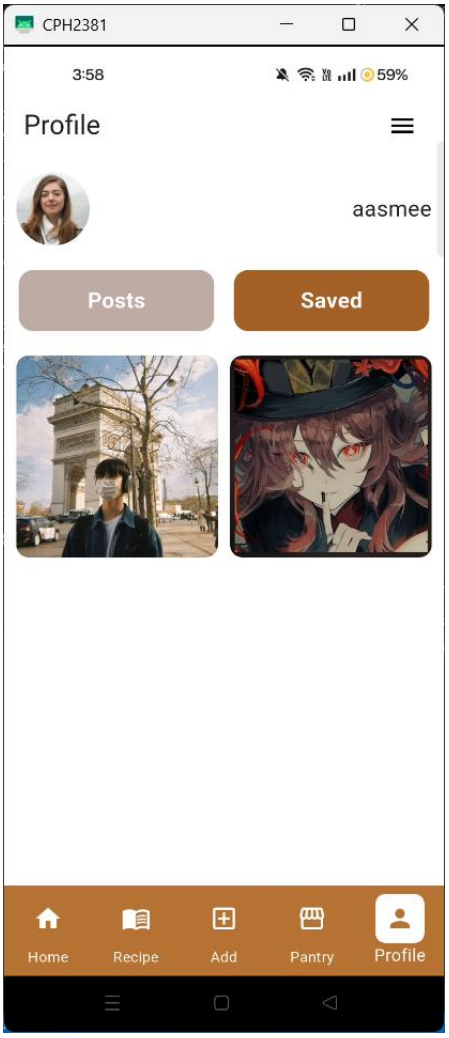


Figure 26: Bookmarked post in saved section

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:

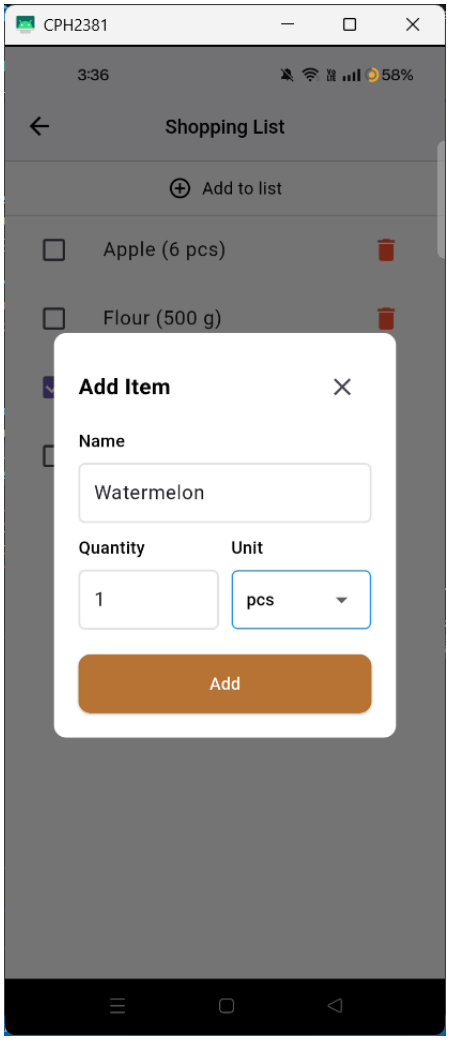


Figure 27: Adding item to list

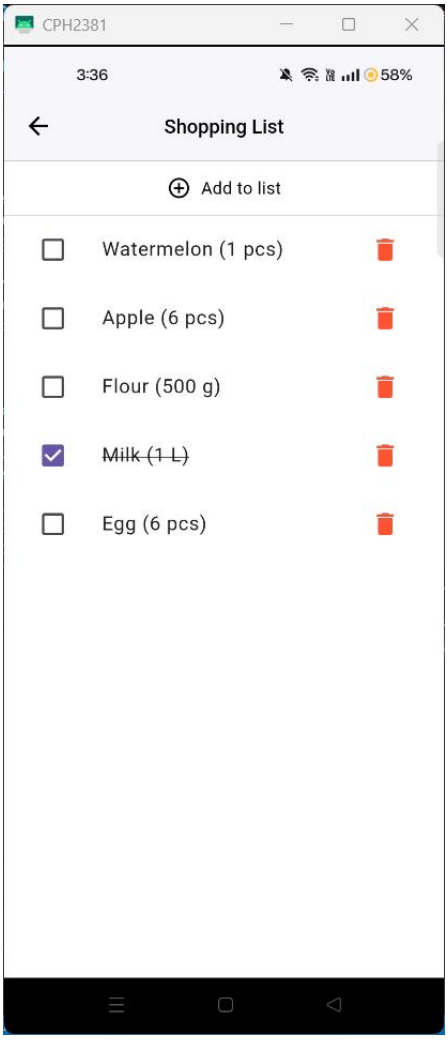


Figure 28: List with added item

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:

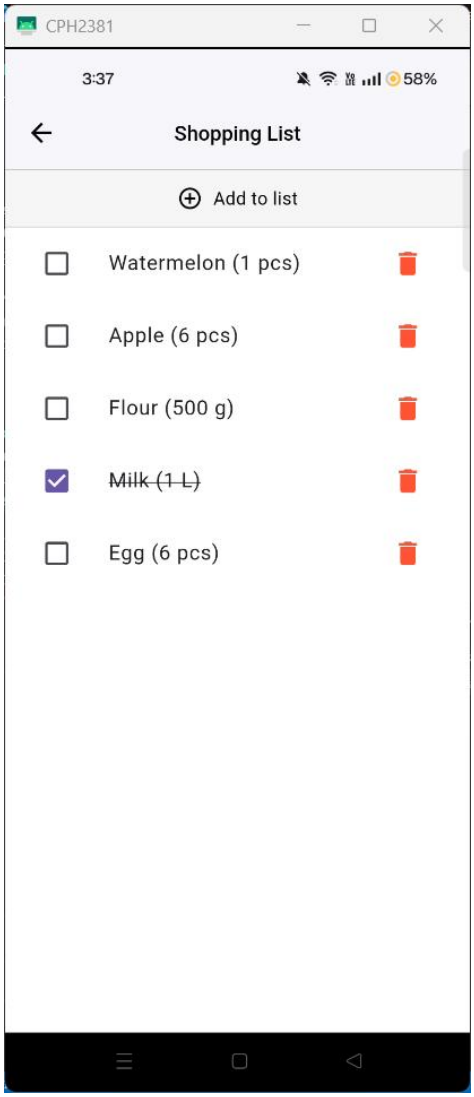


Figure 29: Shopping list before deleting item

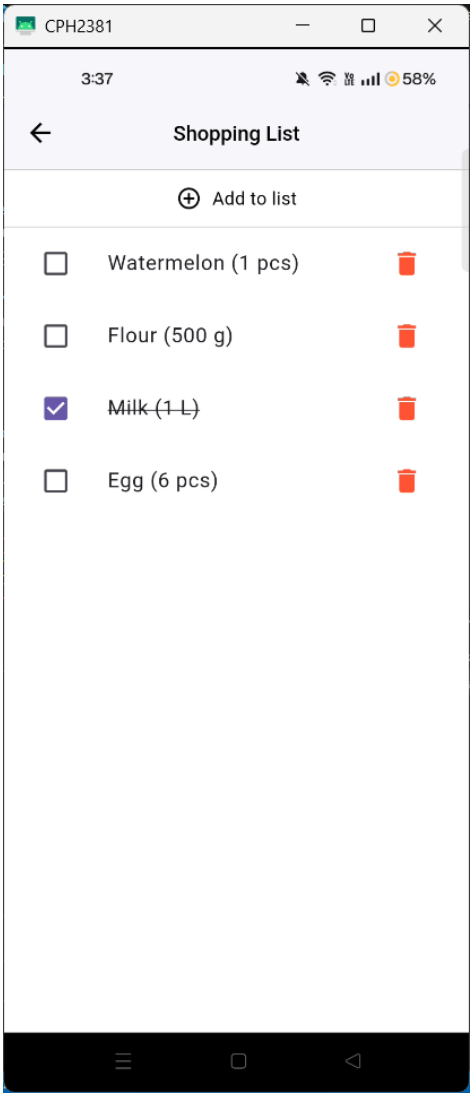


Figure 30: Shopping list after deleting the item

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

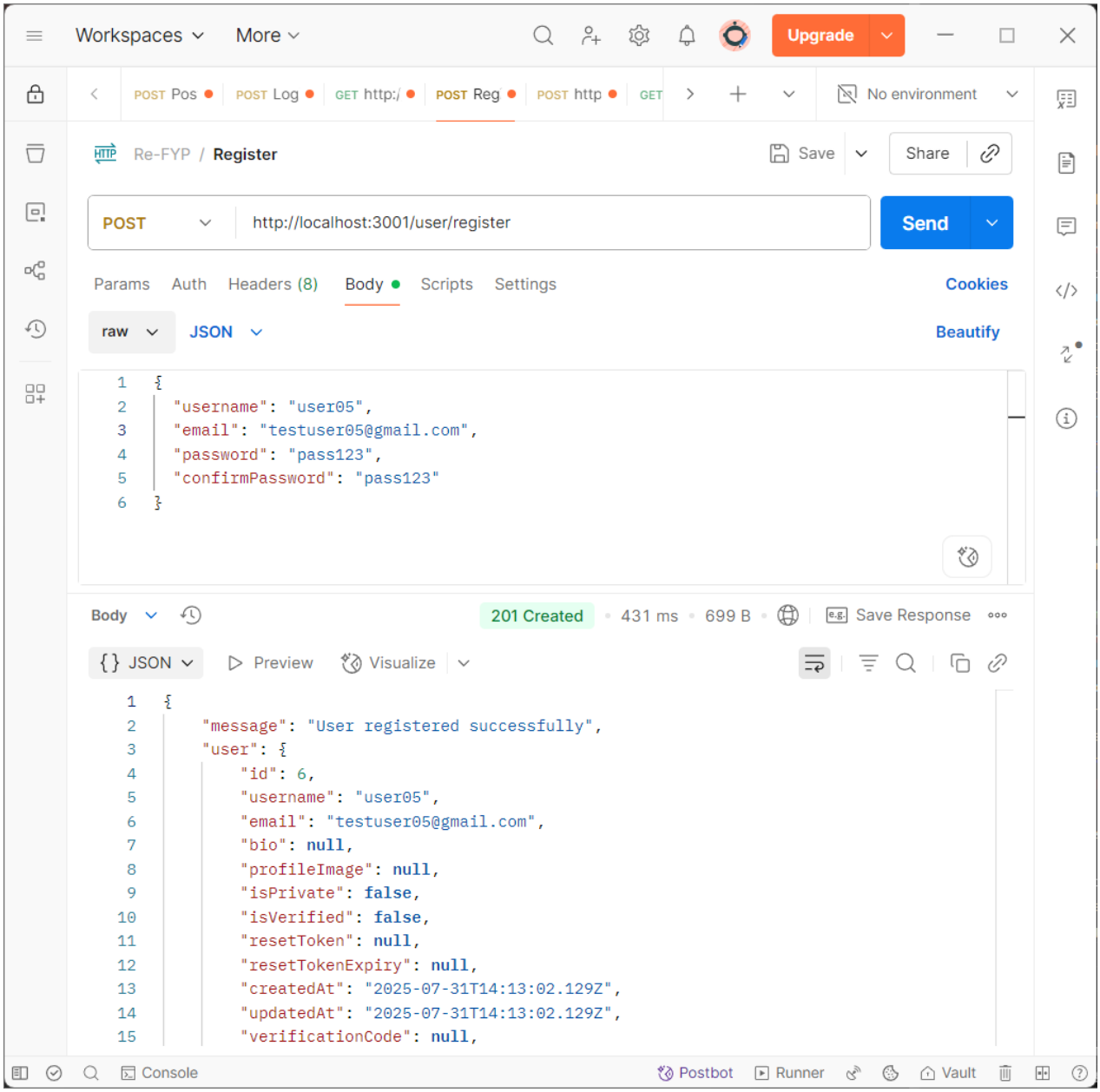


Figure 31: Register API testing

4.3.2. Login

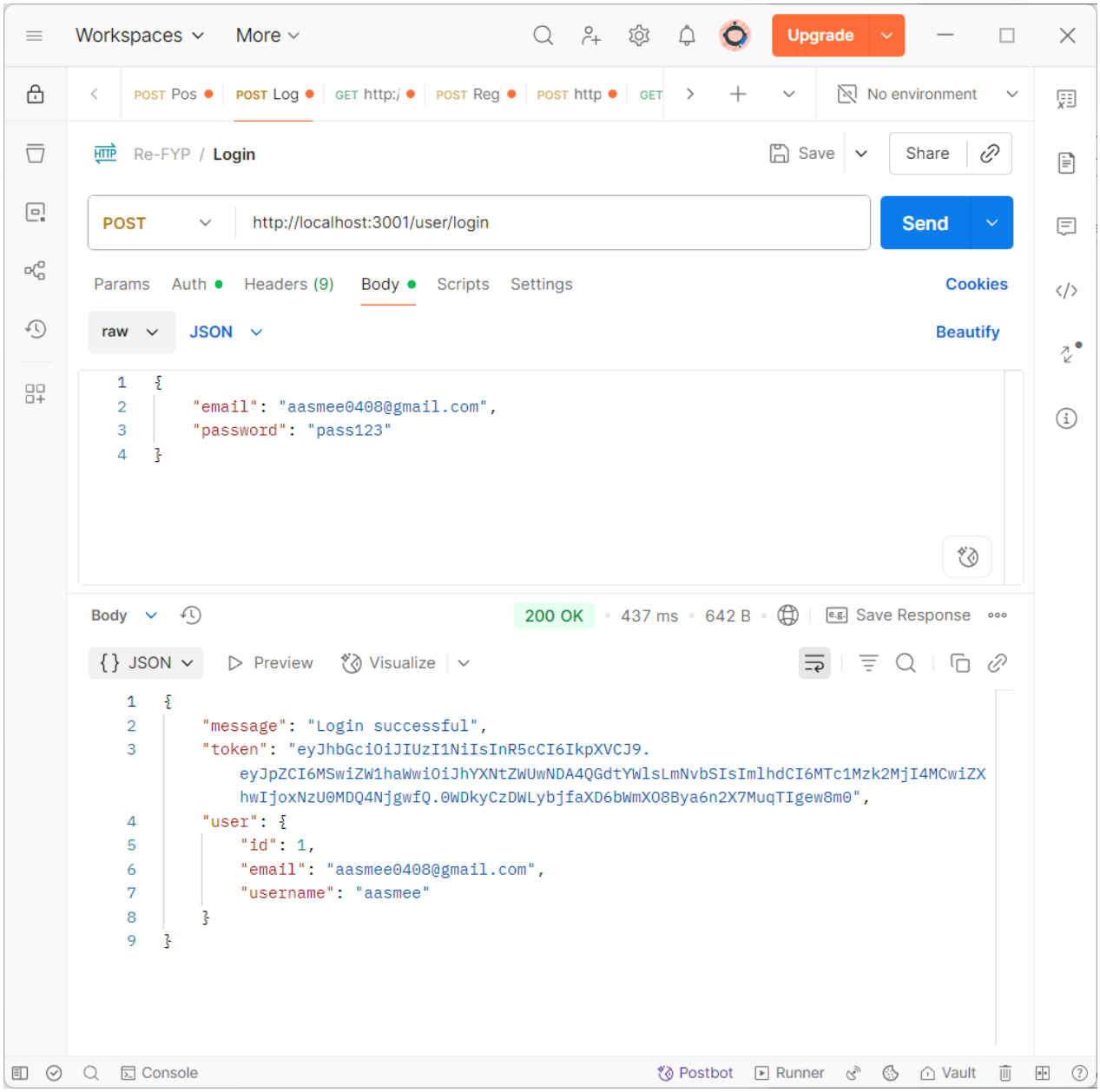


Figure 32: Login API testing

4.3.3. Make post

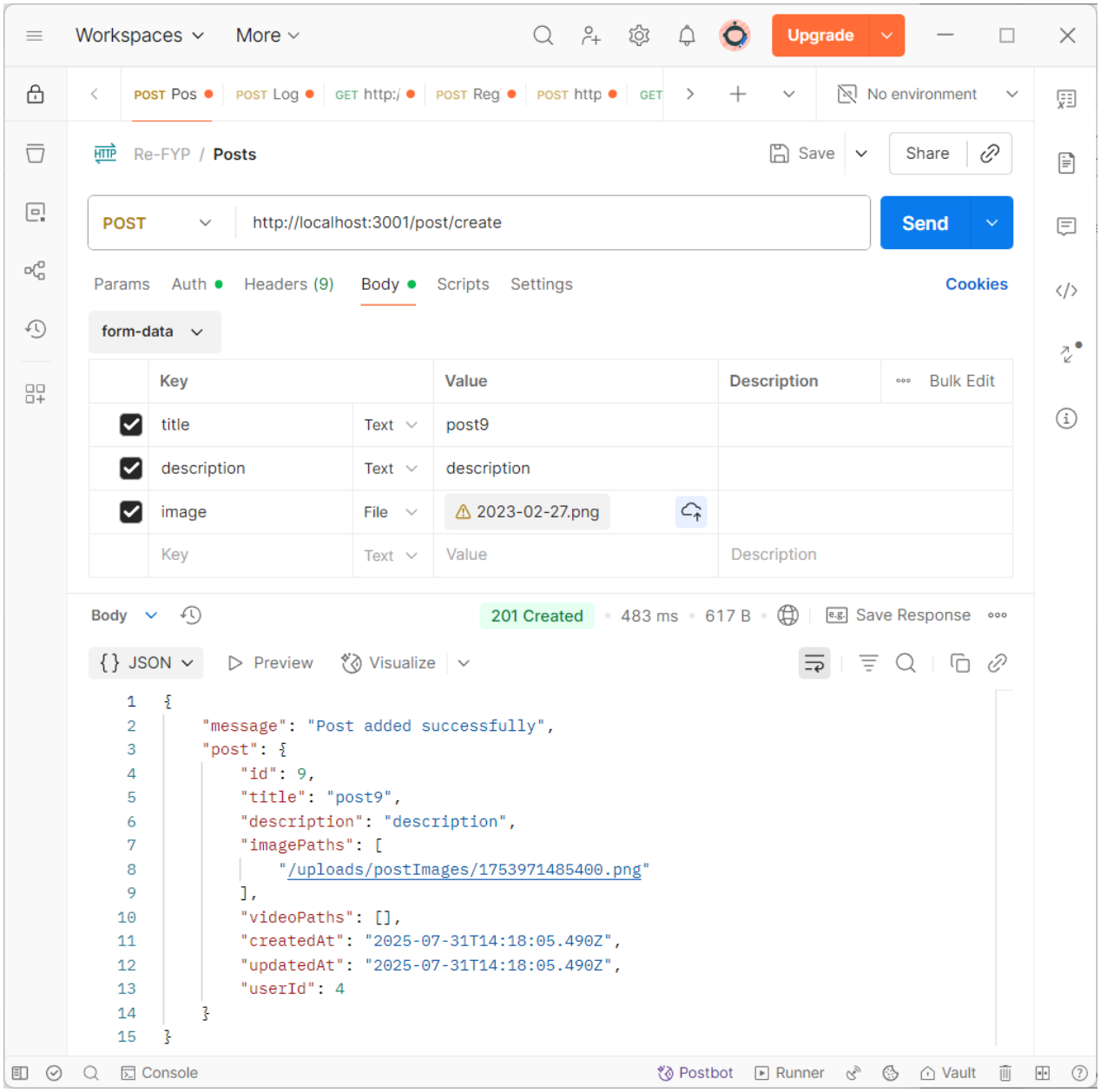


Figure 33: Post creation API testing

4.3.4. Get post

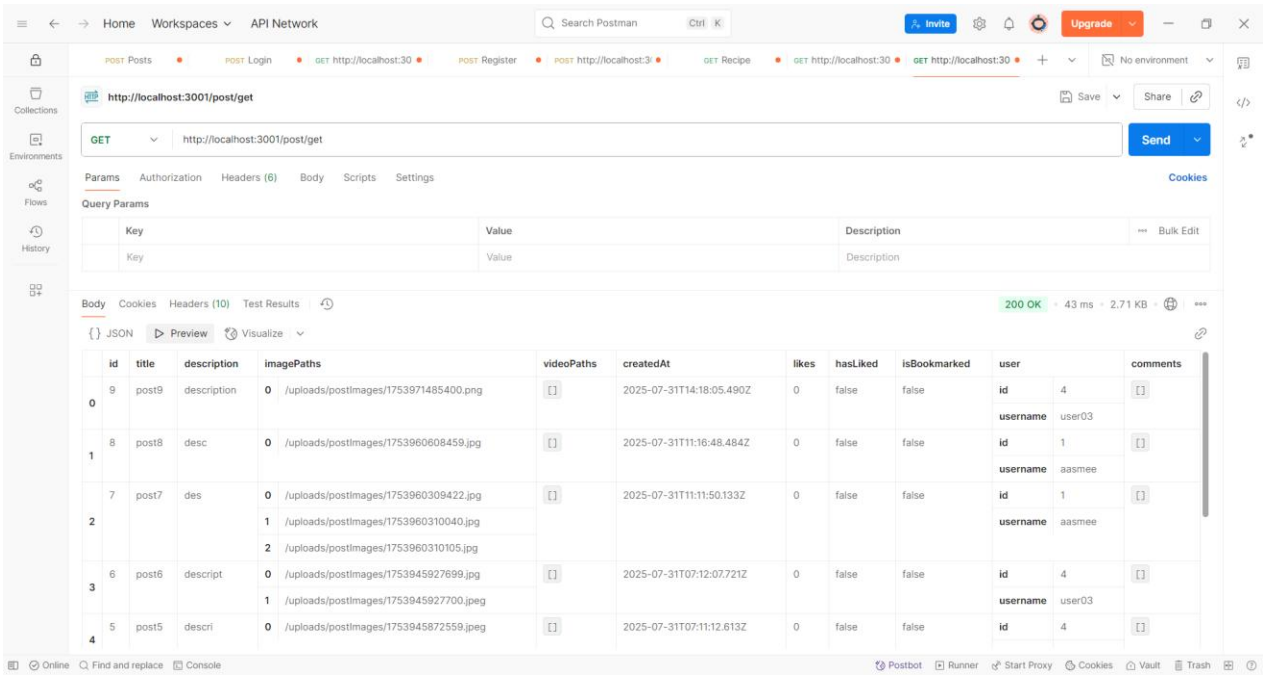


Figure 34: Get post API testing

4.3.5. Delete post

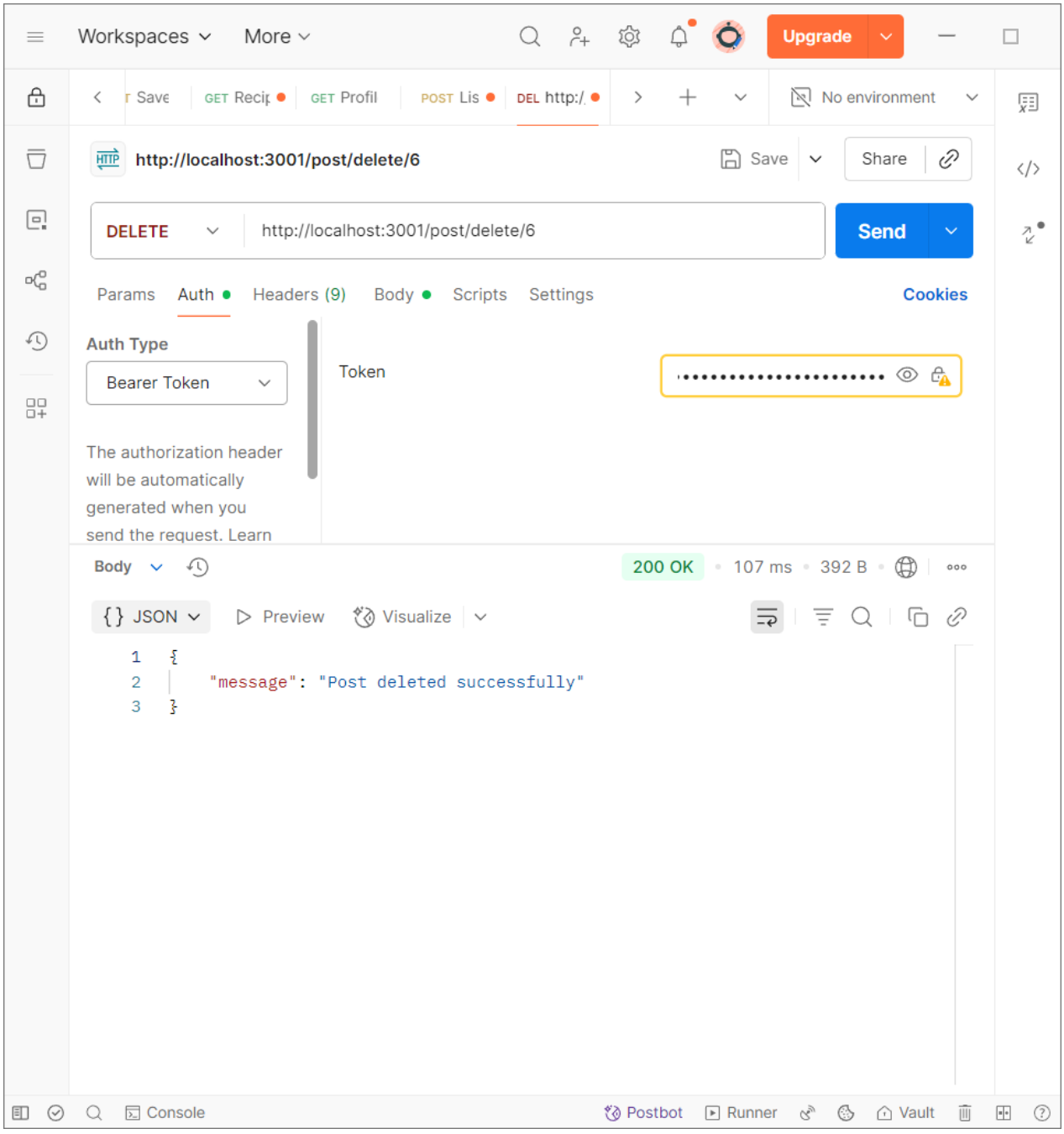


Figure 35: Deleting post API testing

4.3.6. Update posts

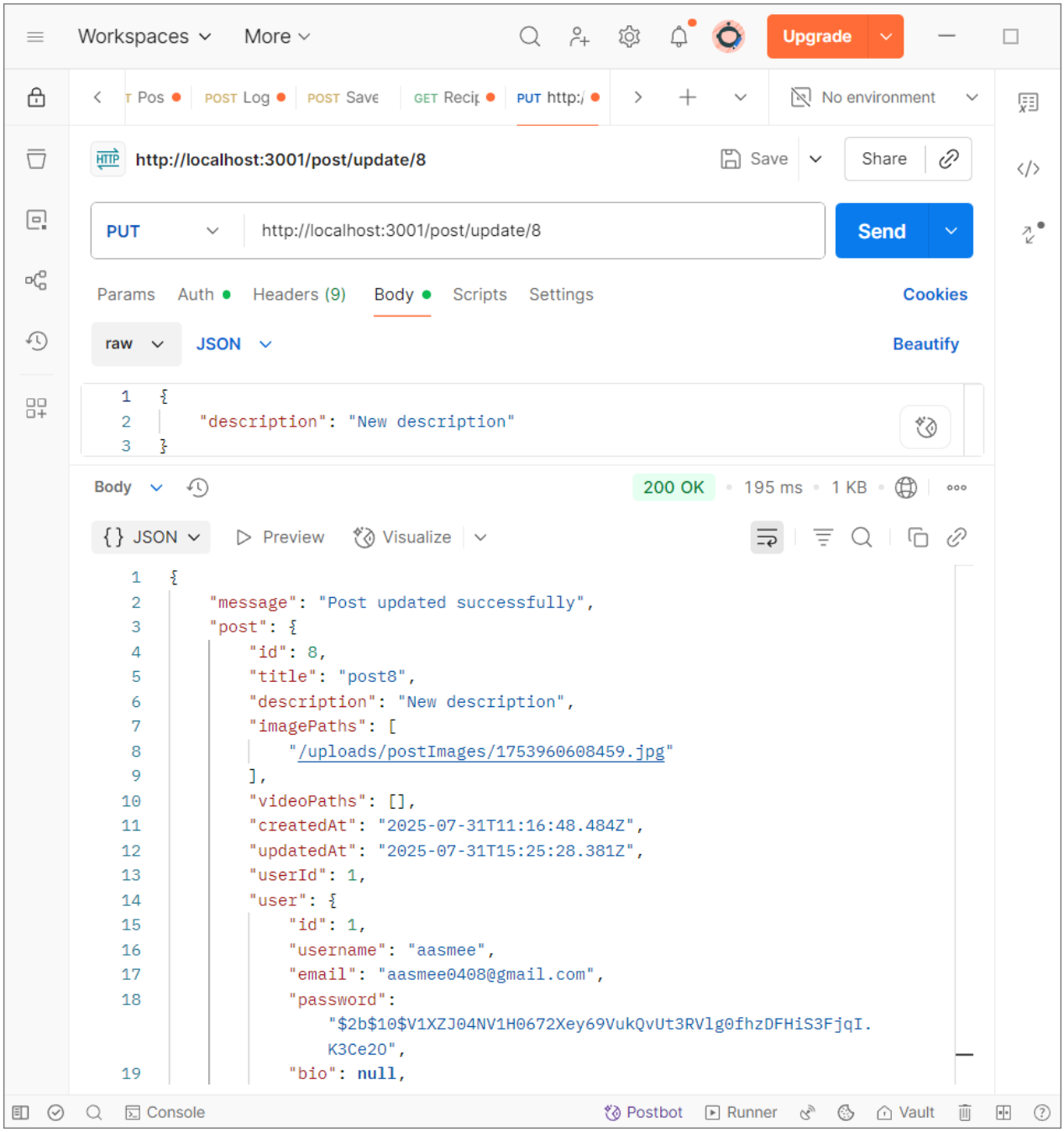


Figure 36: Updating posts API testing

4.3.7. Like post

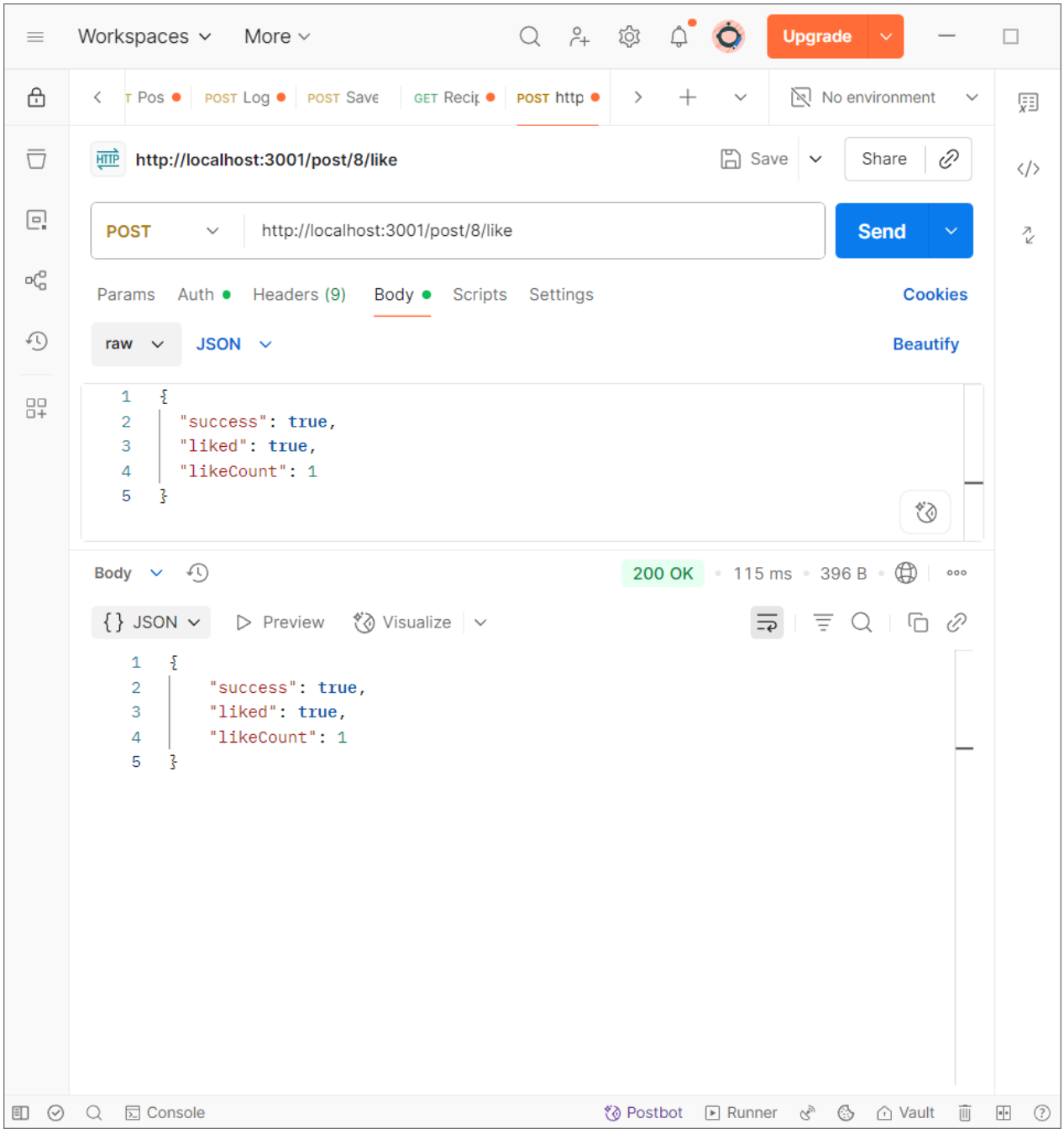


Figure 37: Liking post API testing

4.3.8. Bookmark posts

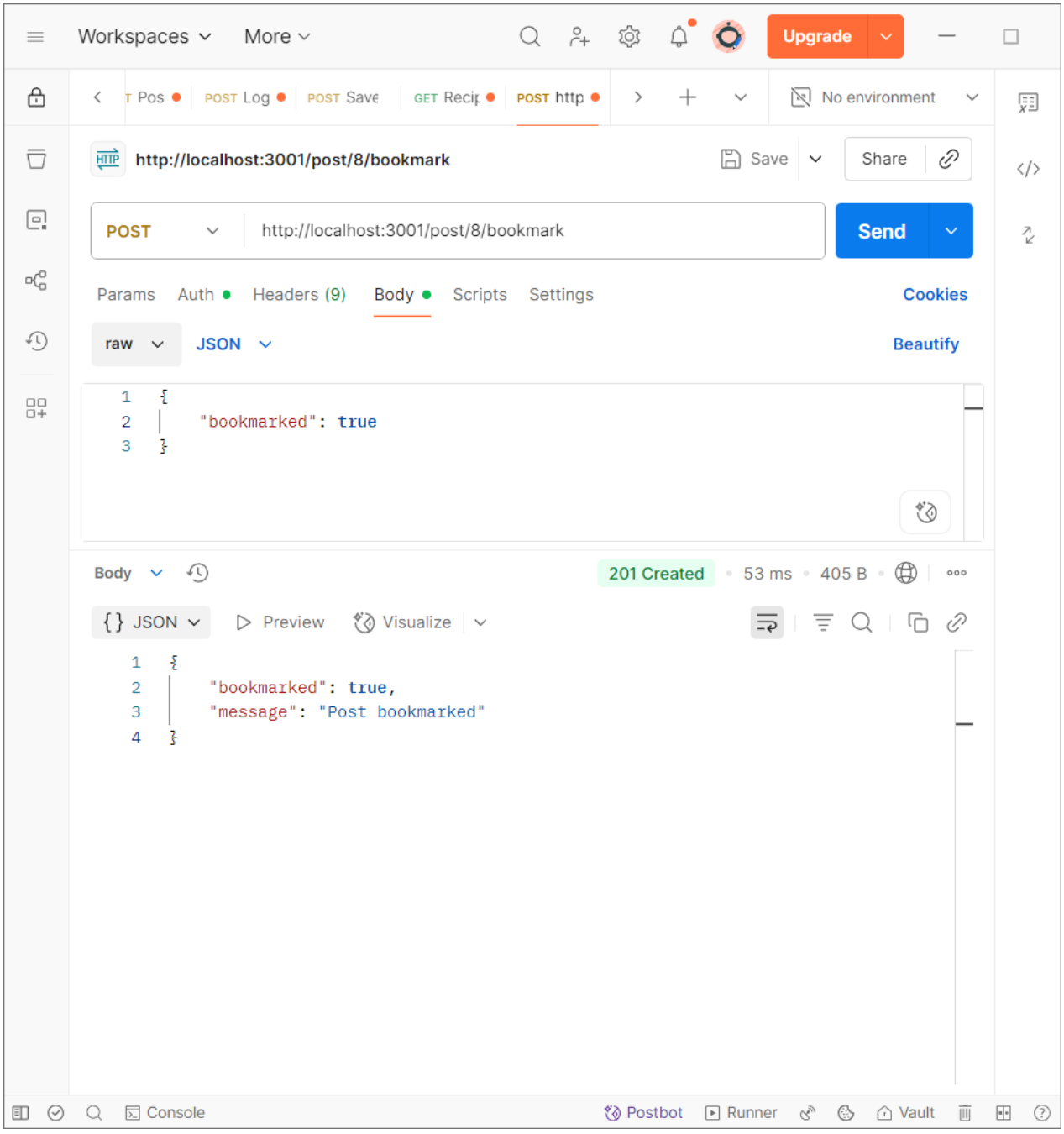


Figure 38: Bookmarking posts API testing

4.3.9. View bookmarked posts

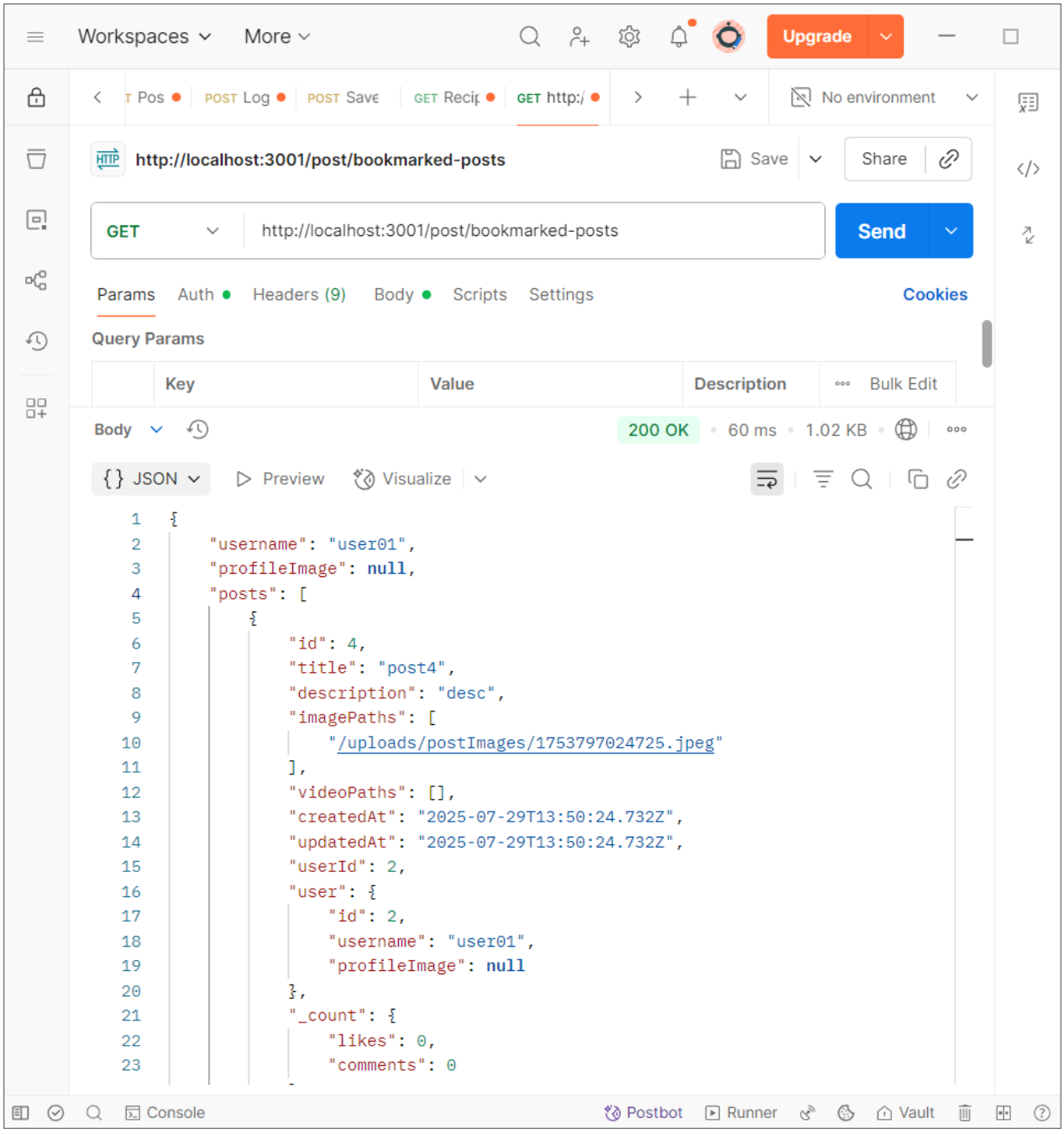


Figure 39: View bookmarked posts API testing

4.3.10. Add comment

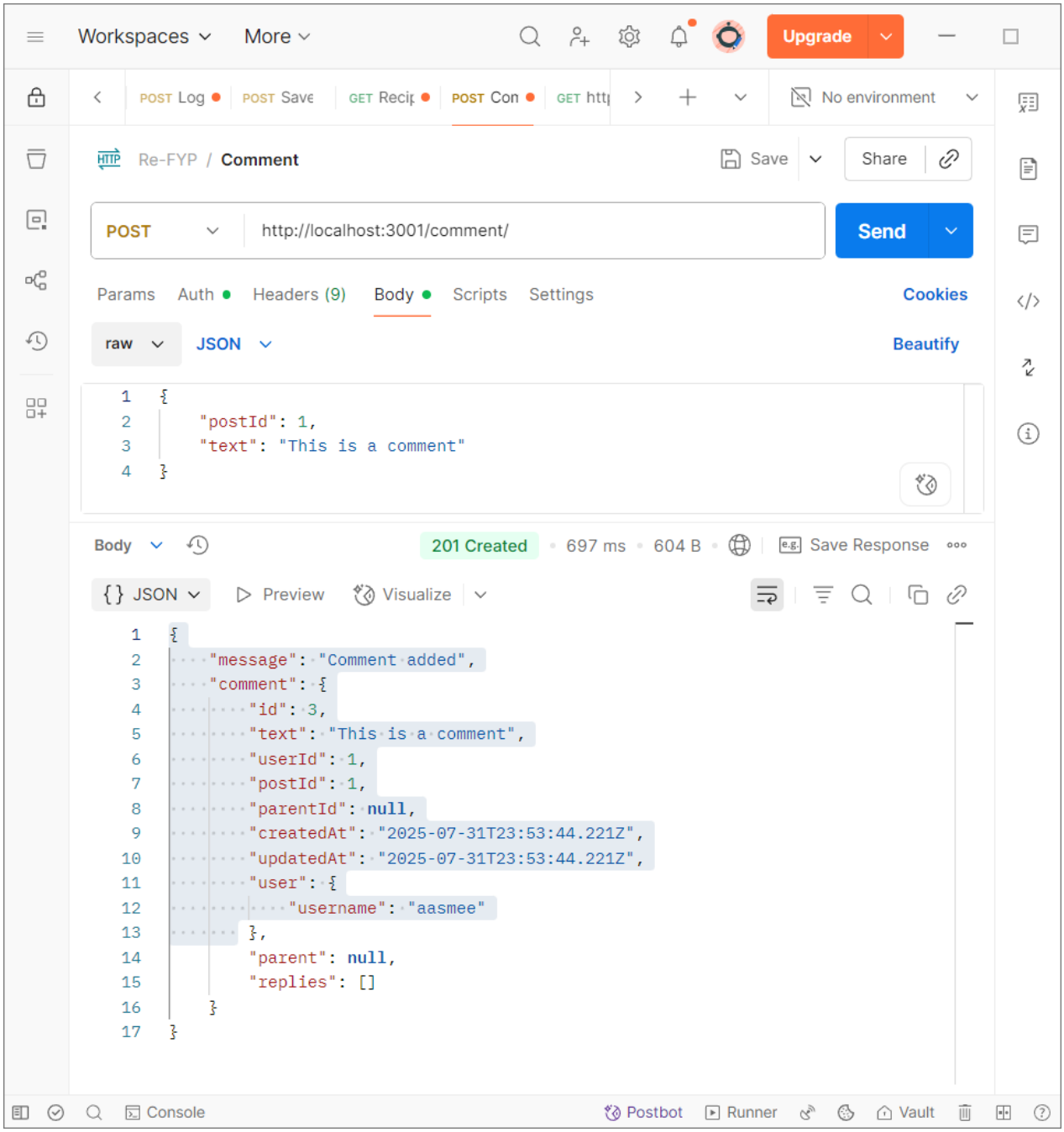


Figure 40: Adding comment API testing

4.3.11. Get comment

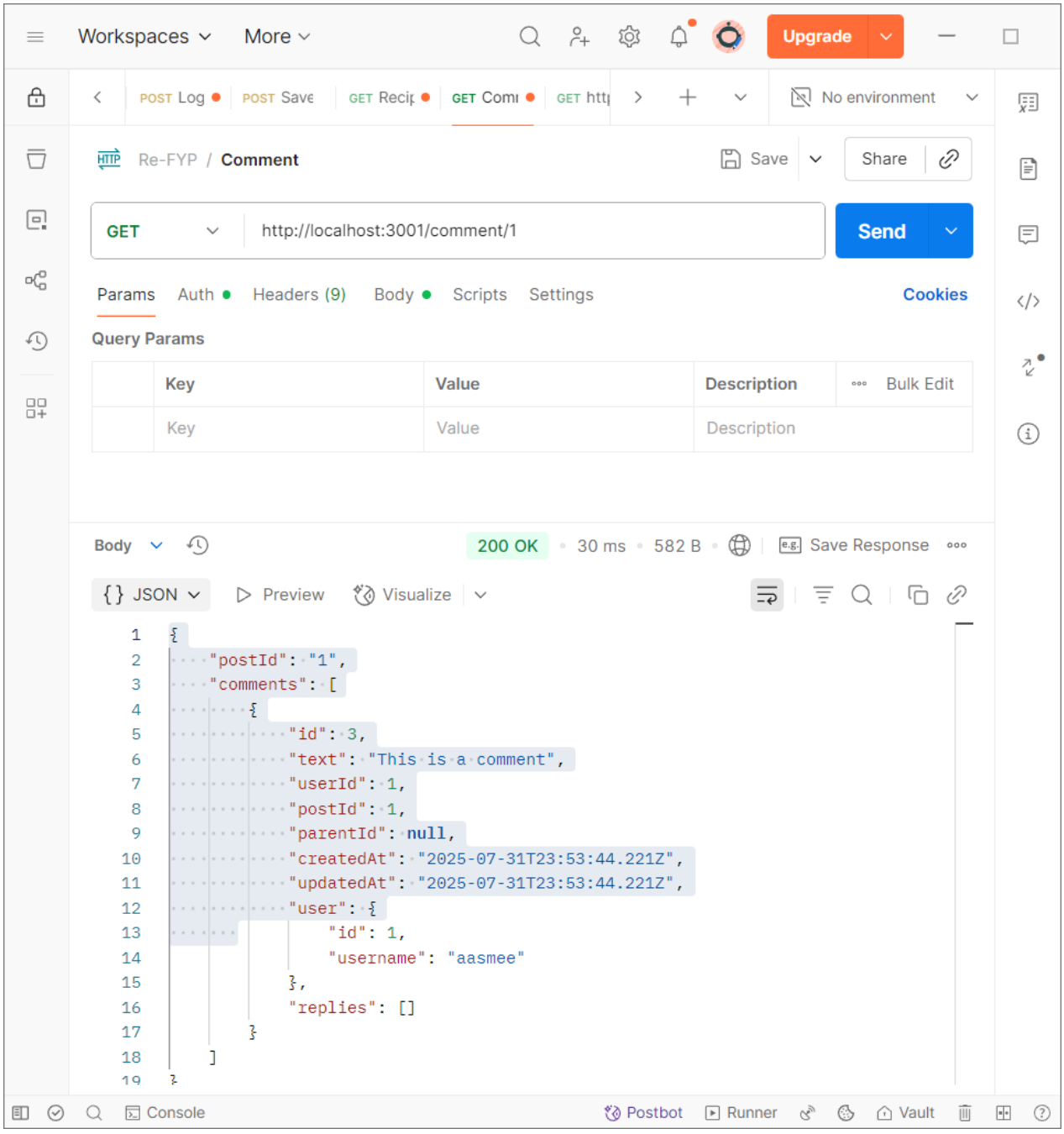


Figure 41: Getting all comments on a specific post

4.3.12. Delete comment

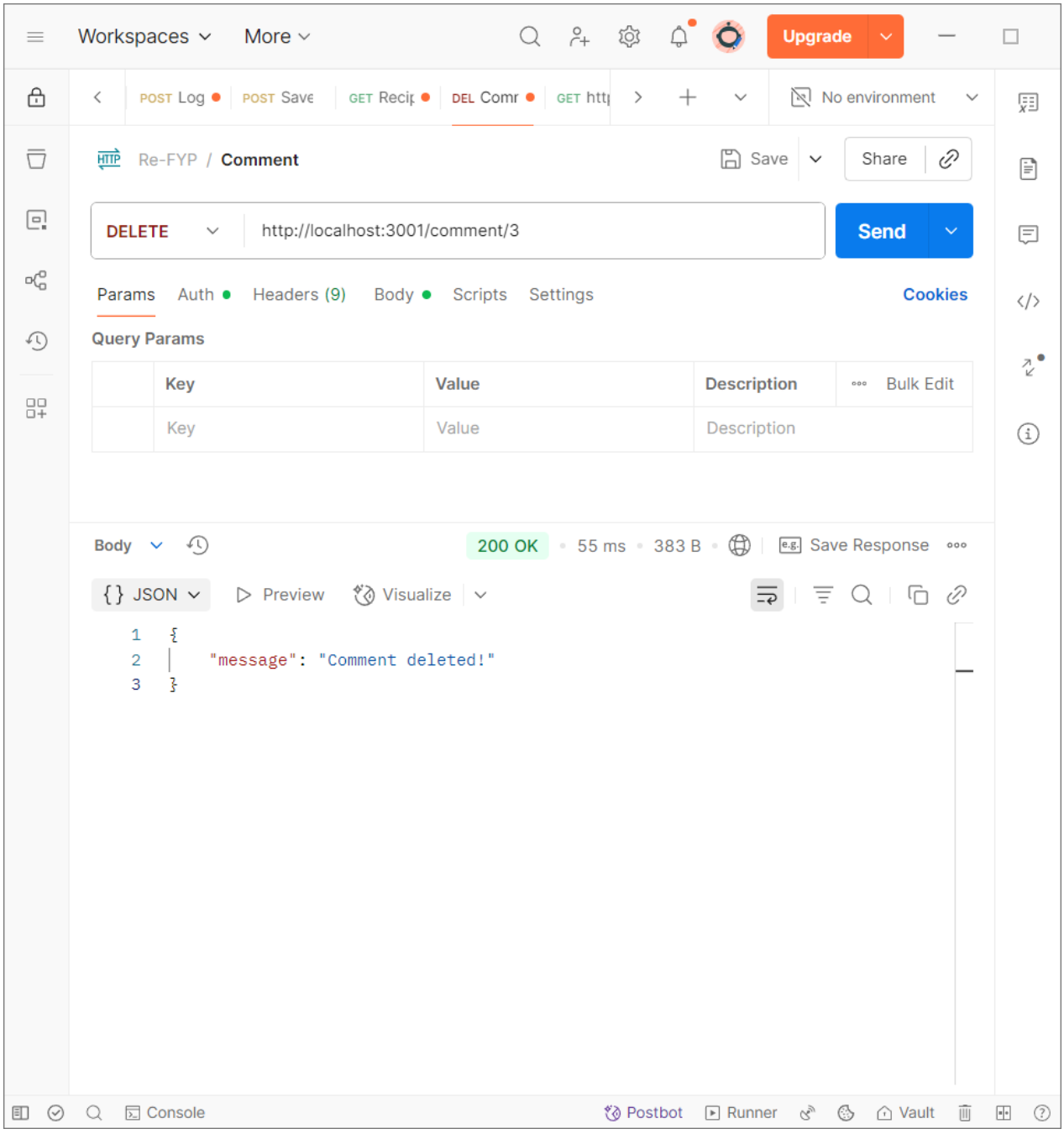


Figure 42: Deleting comment API testing

4.3.13. Edit comment

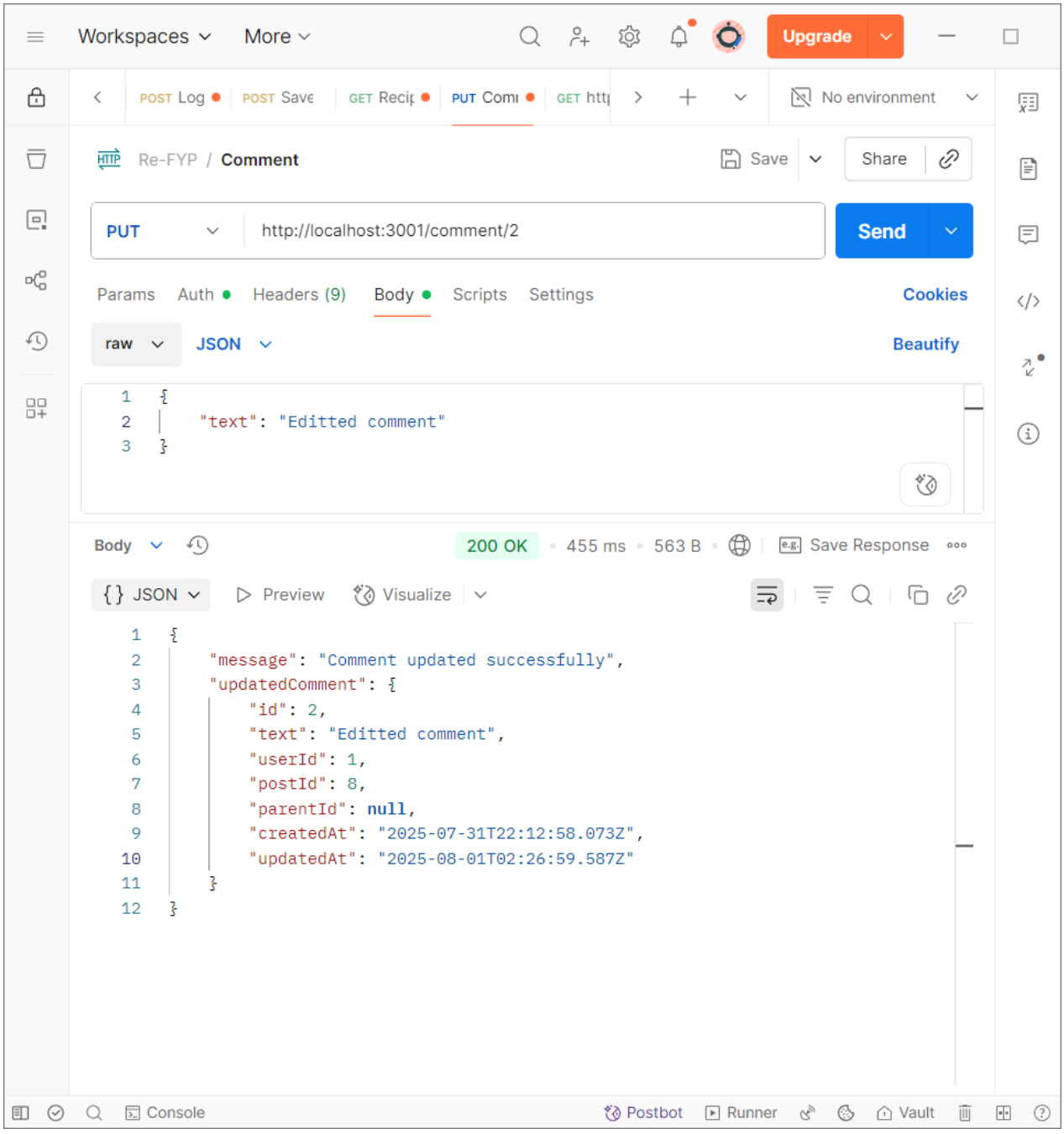


Figure 43: Editing comment API testing

4.3.14. Get notification

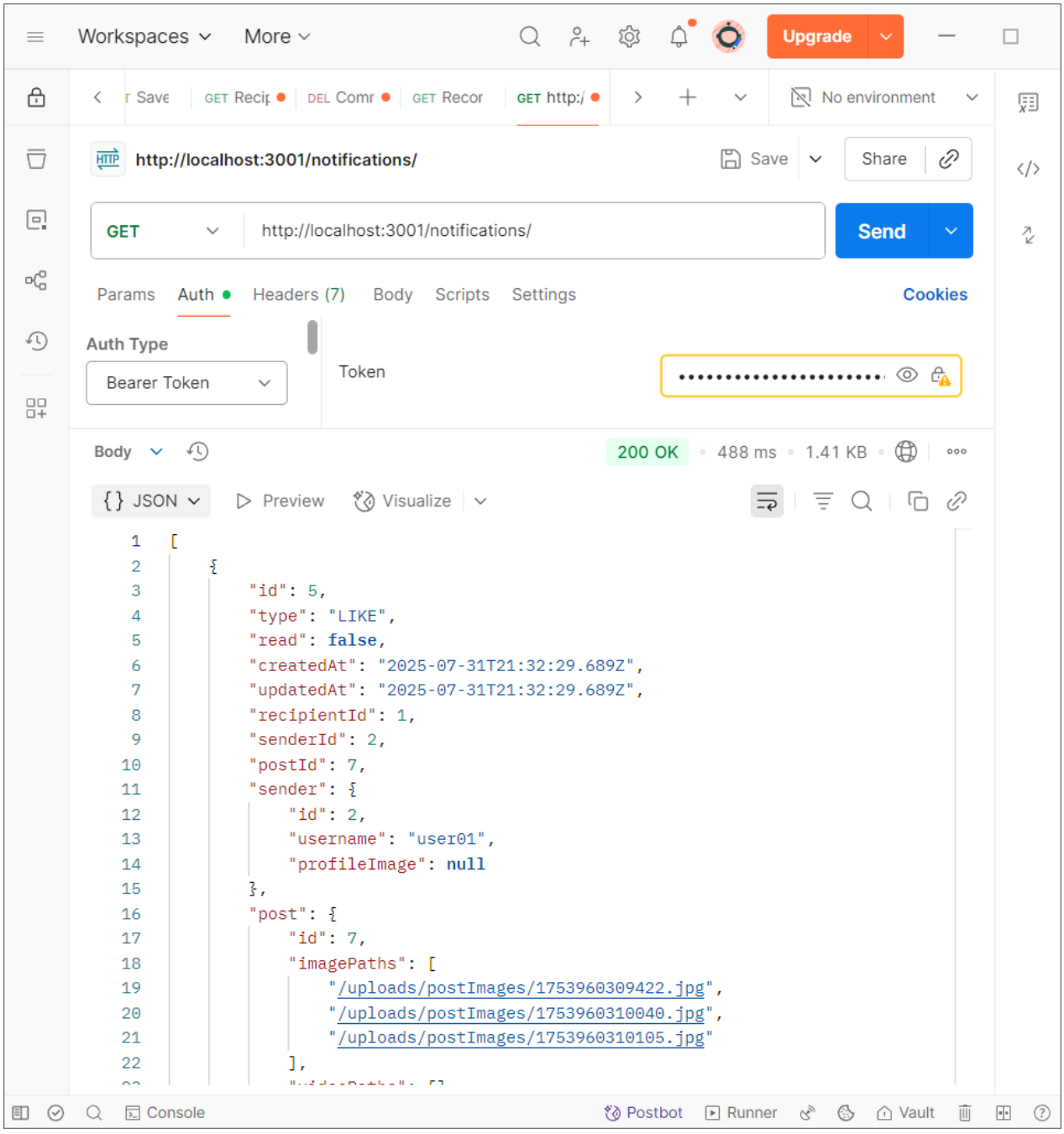


Figure 44: Getting notification API testing

4.3.15. Read notifications

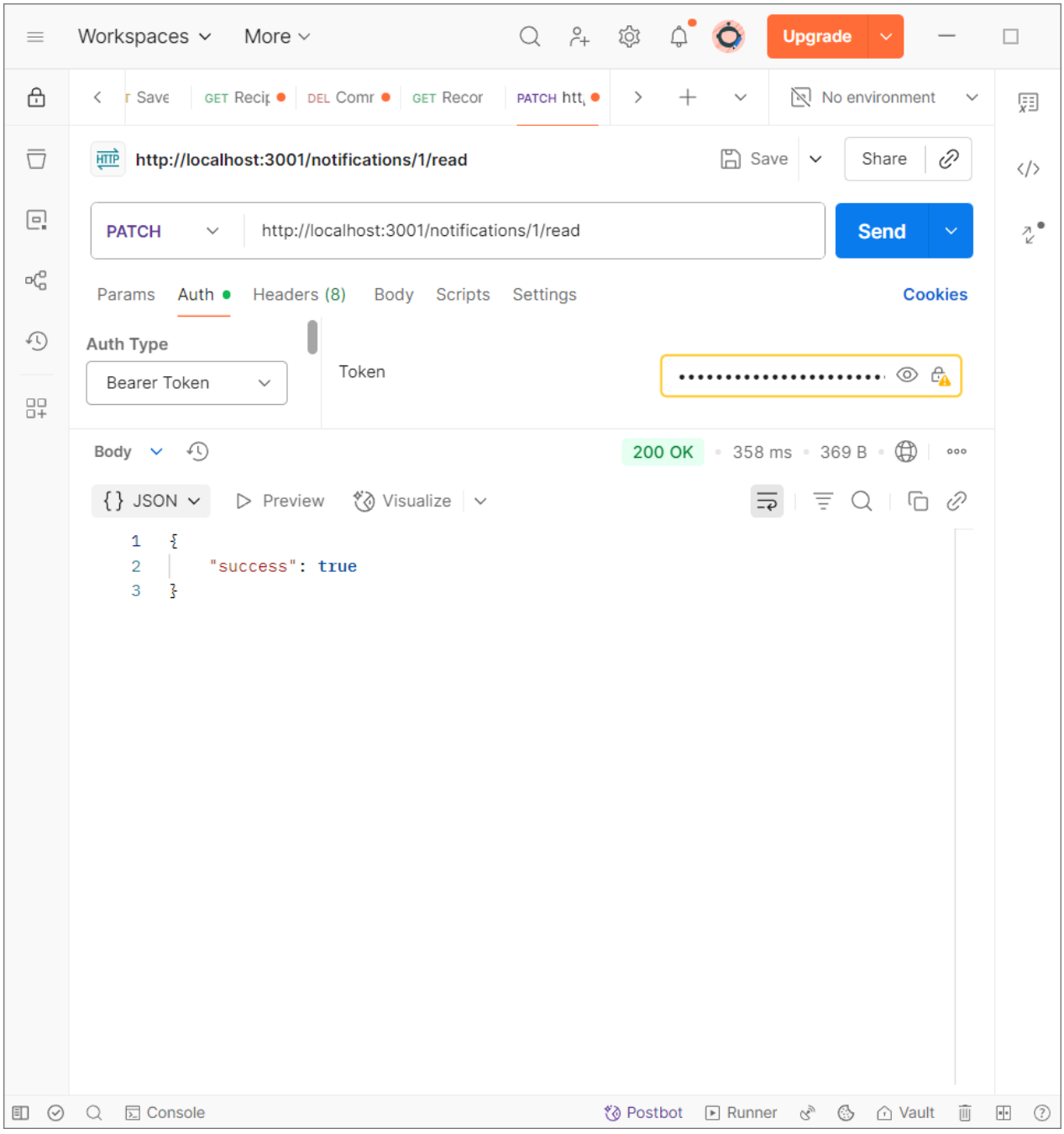


Figure 45: Reading notification API testing

4.3.16. Add items to Shopping List

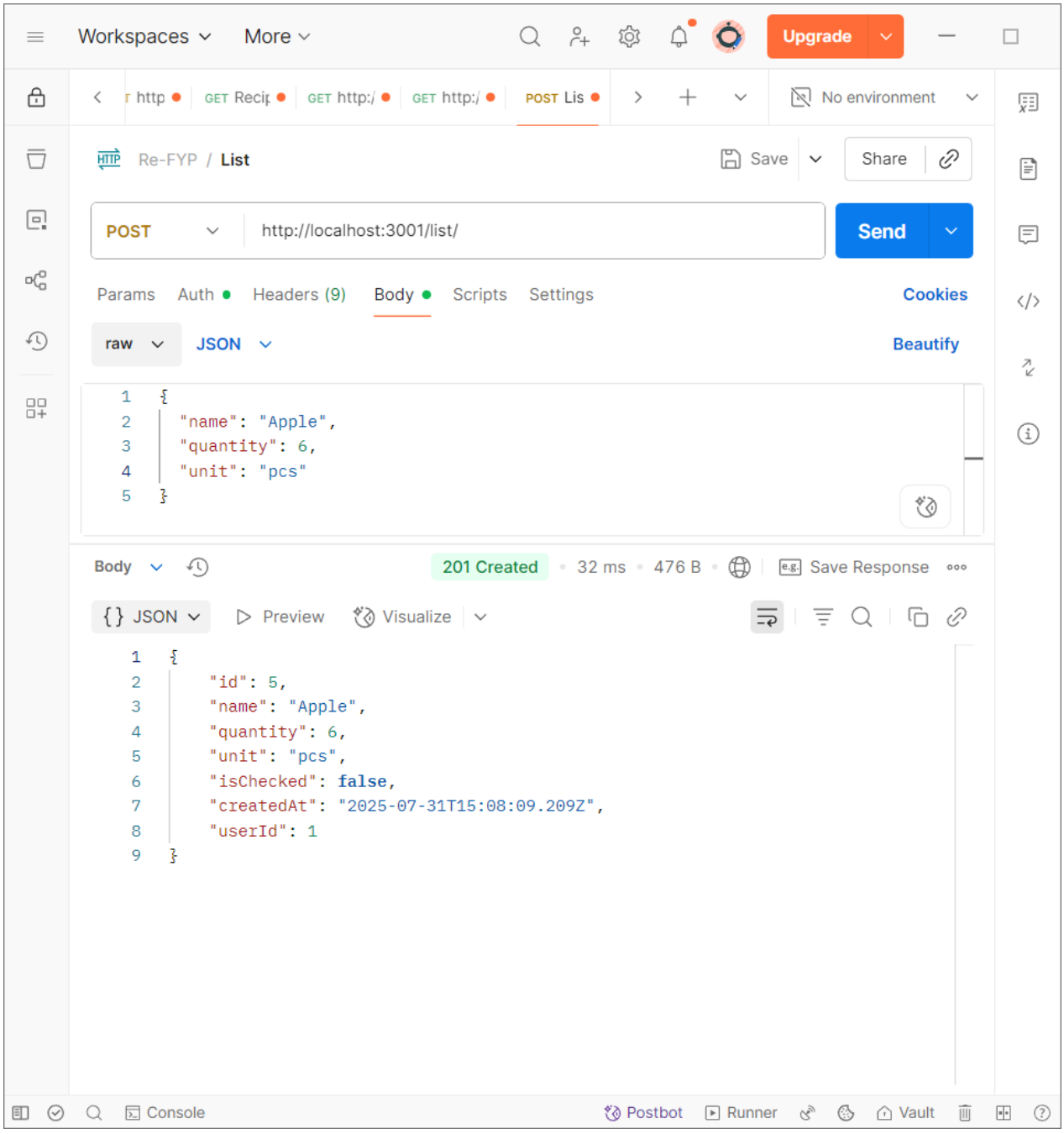


Figure 46: Adding items to list API testing

4.3.17. Get shopping list

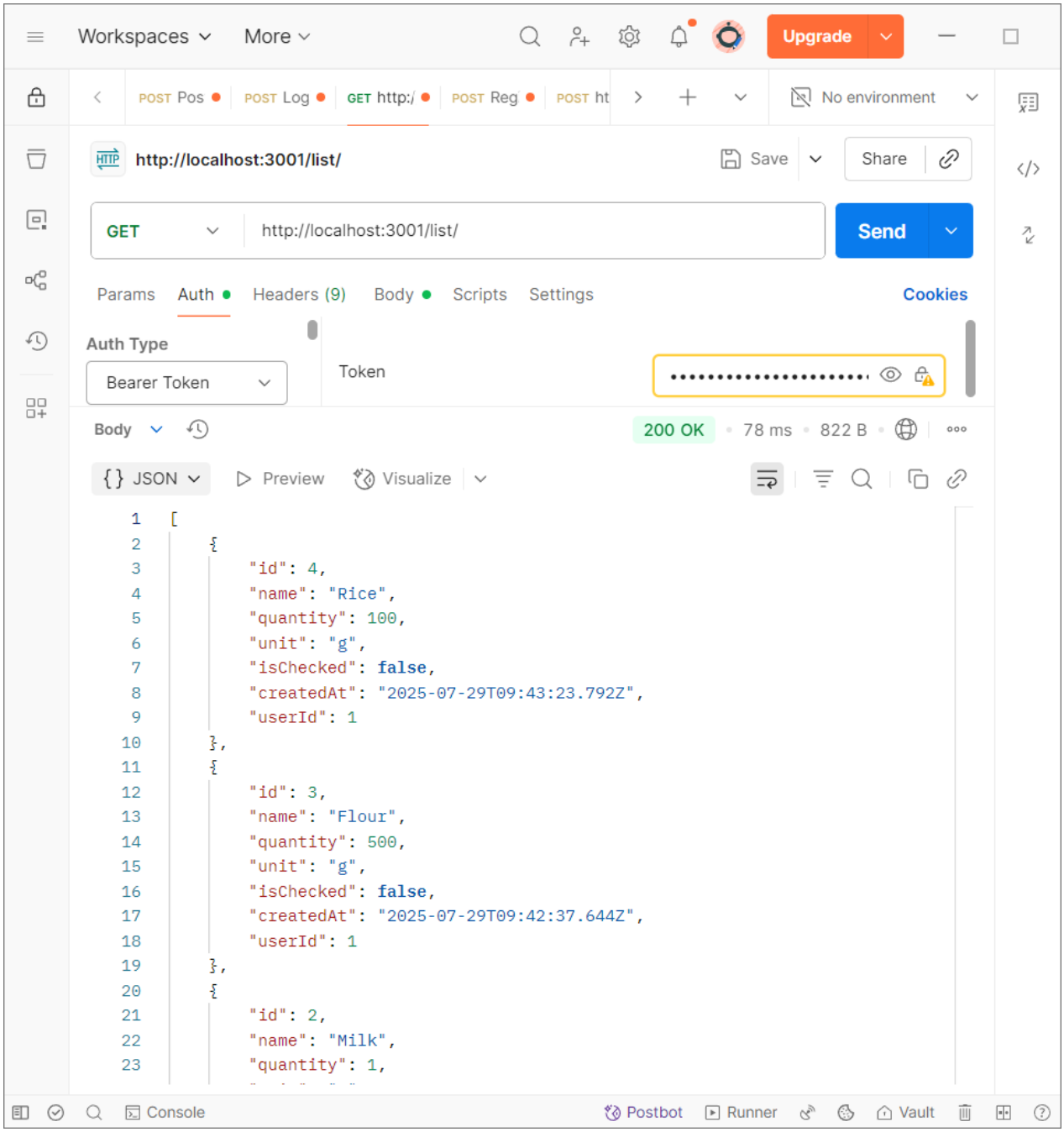


Figure 47: Get shopping list of the active user API testing

4.3.18. Update list

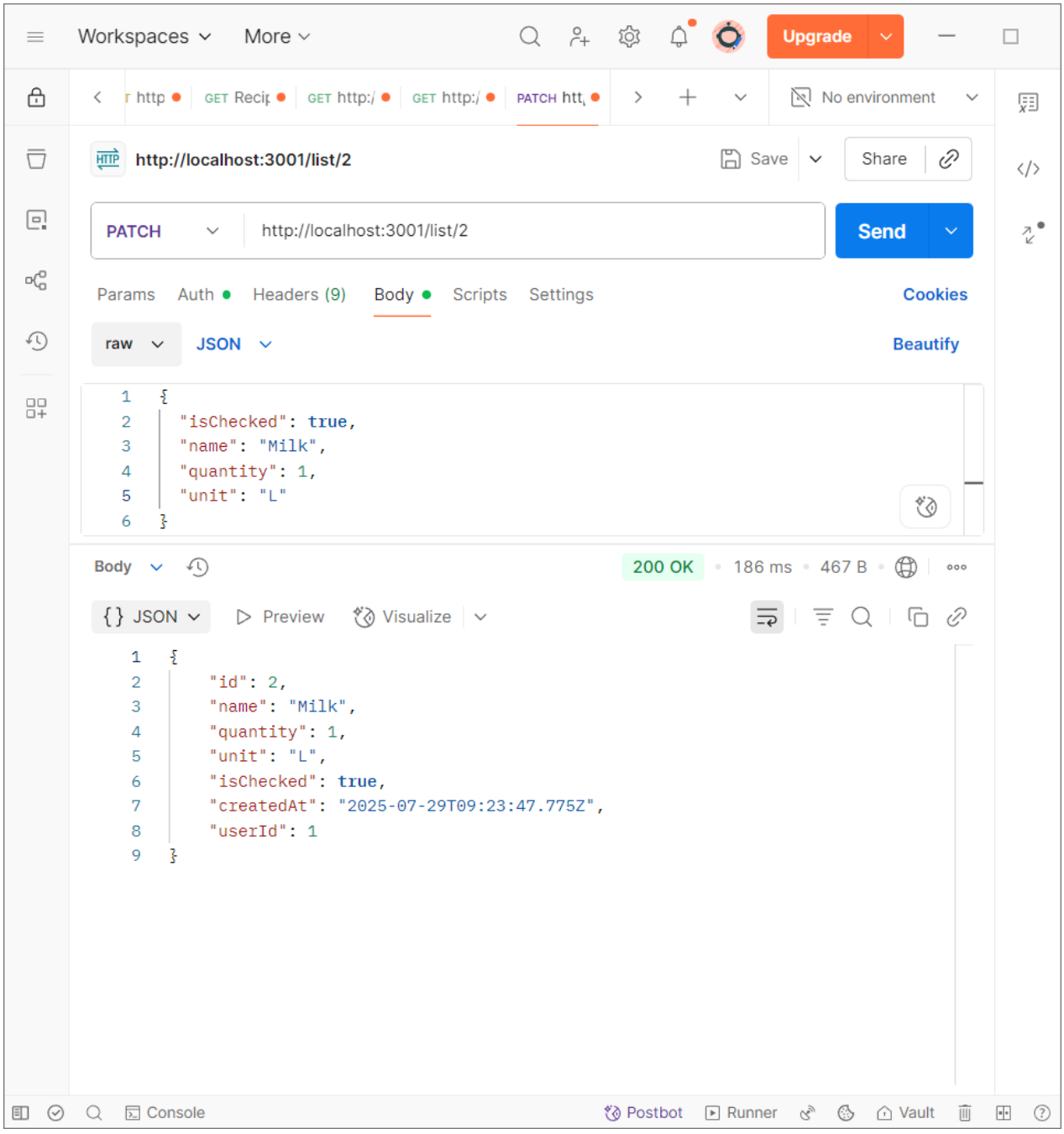


Figure 48: Update shopping list item API testing

4.3.19. Delete list item

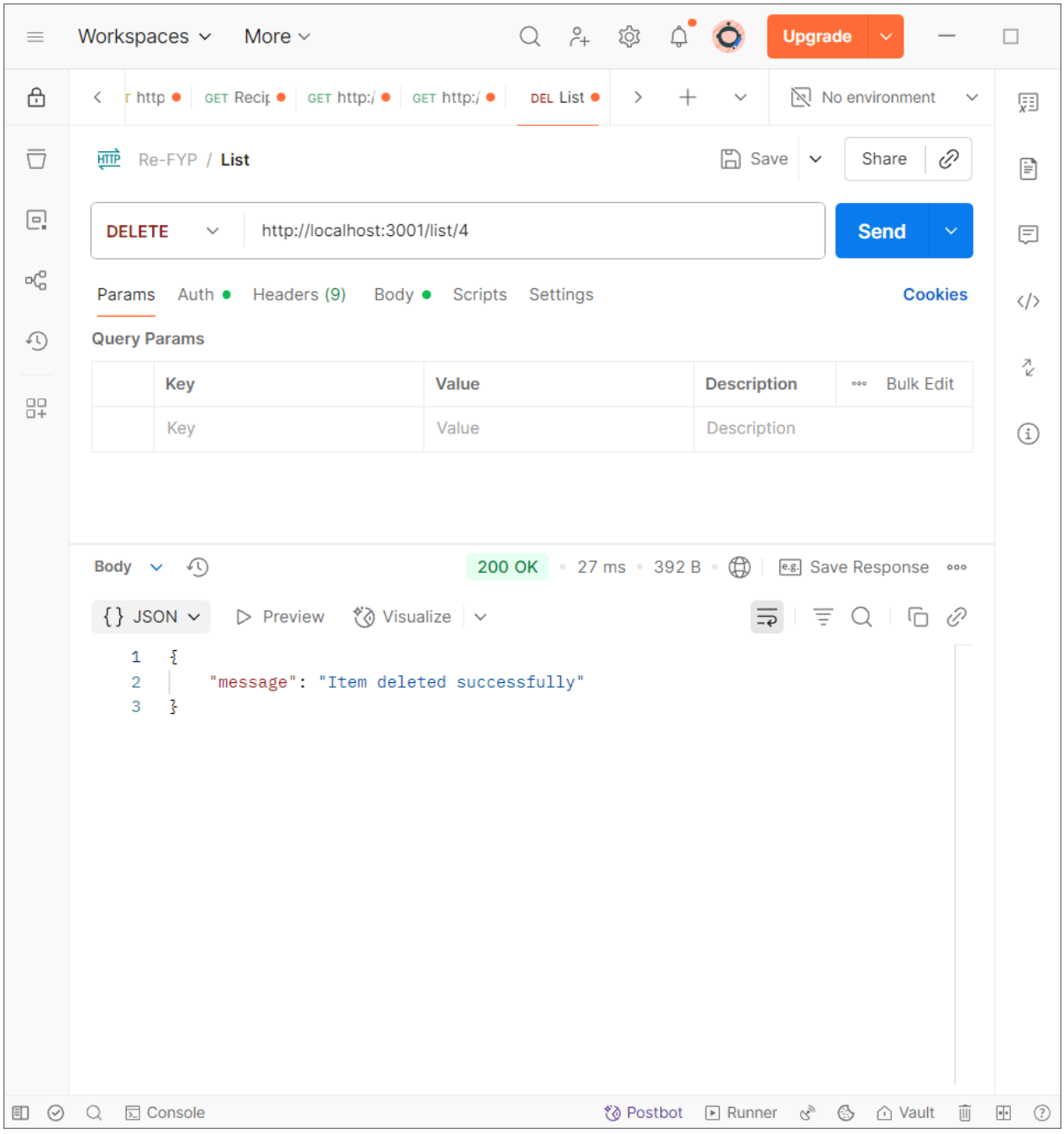


Figure 49: Deleting list item API testing

4.3.20. Get all recipes

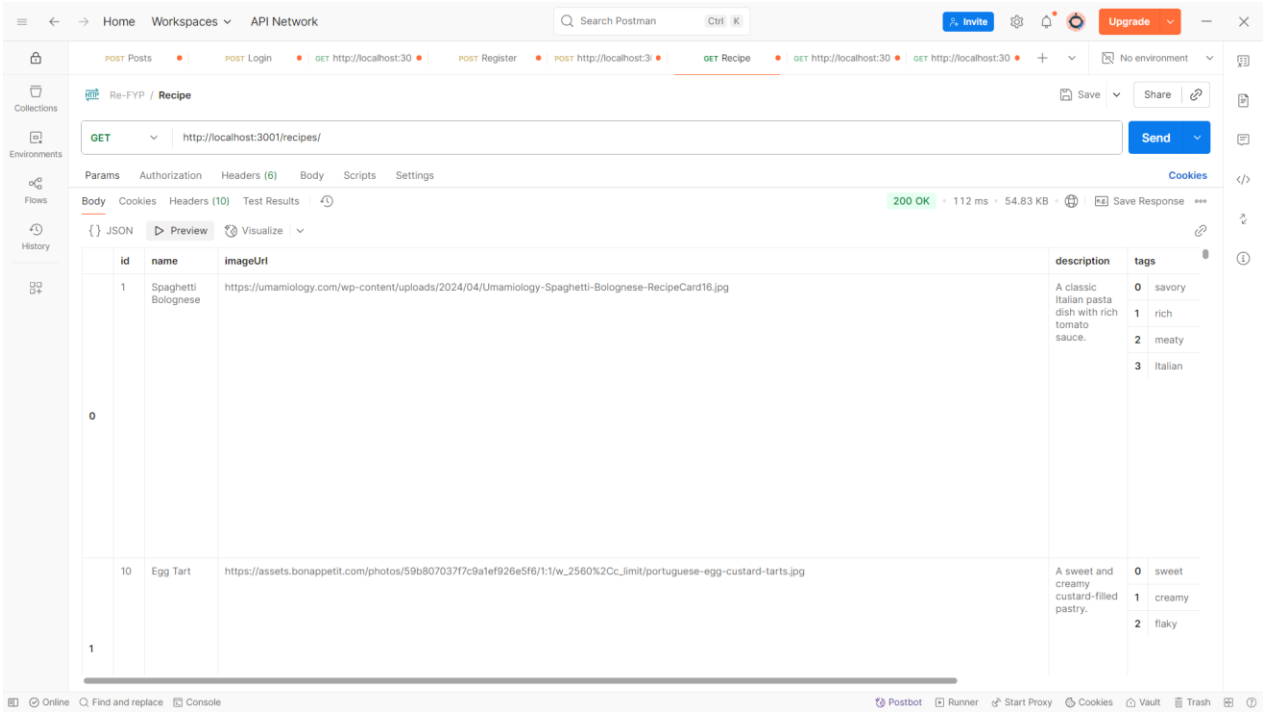


Figure 50: Get all recipes API testing

4.3.21. Get a single recipe

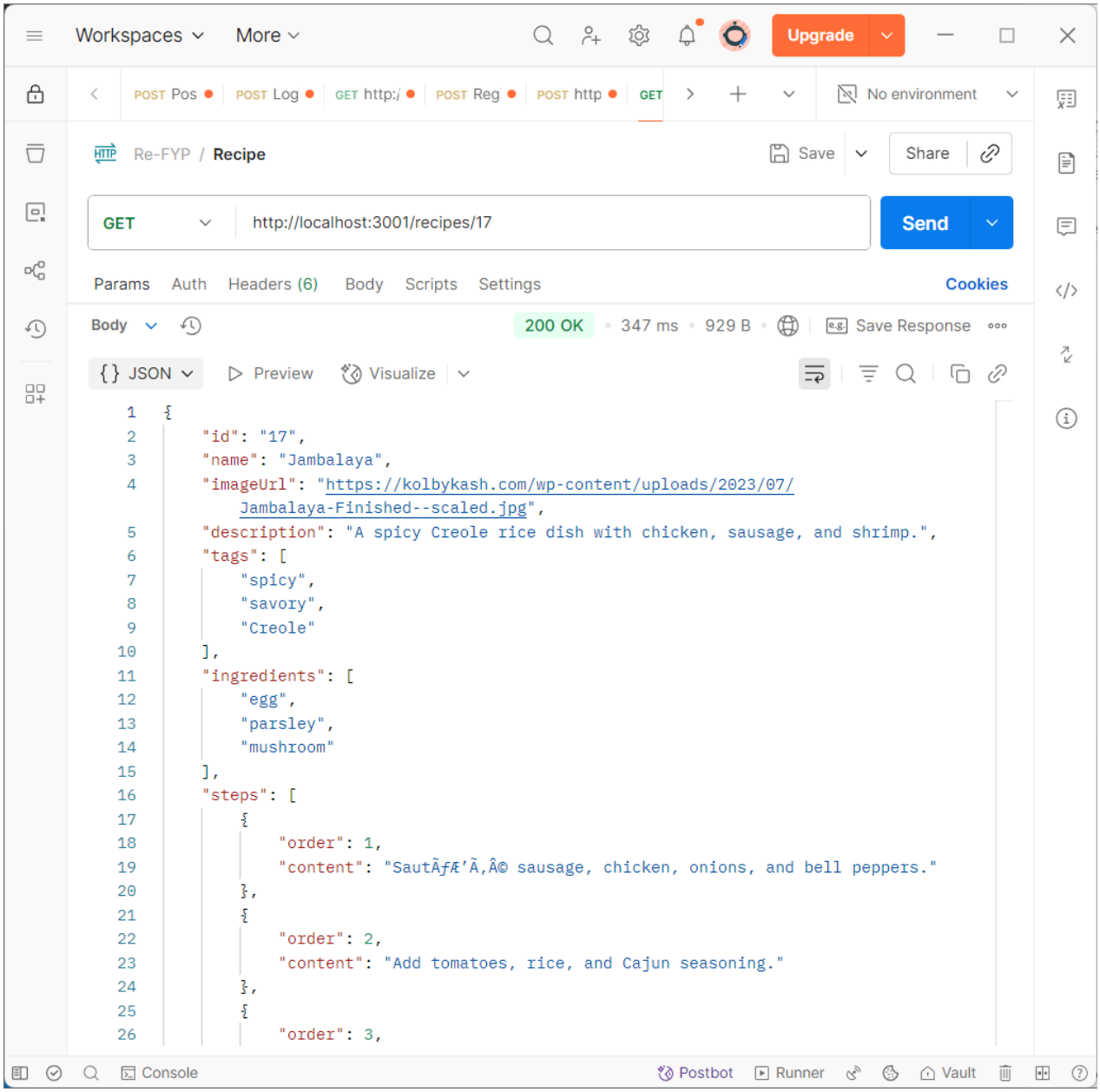


Figure 51: Get a single recipe by id API testing

4.3.22. Getting pantry items

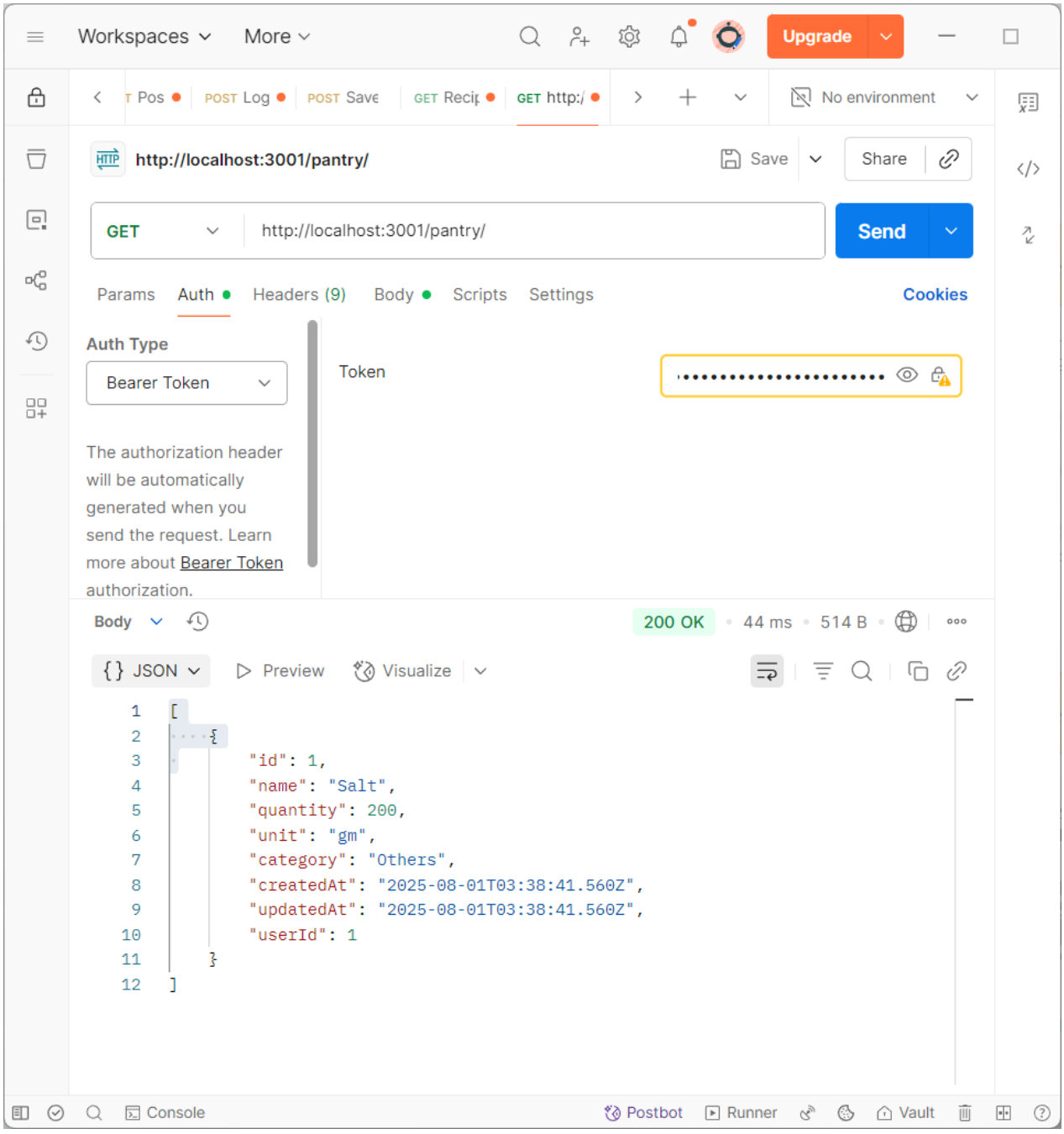


Figure 52: Getting pantry items API testing

4.3.23. Add pantry items

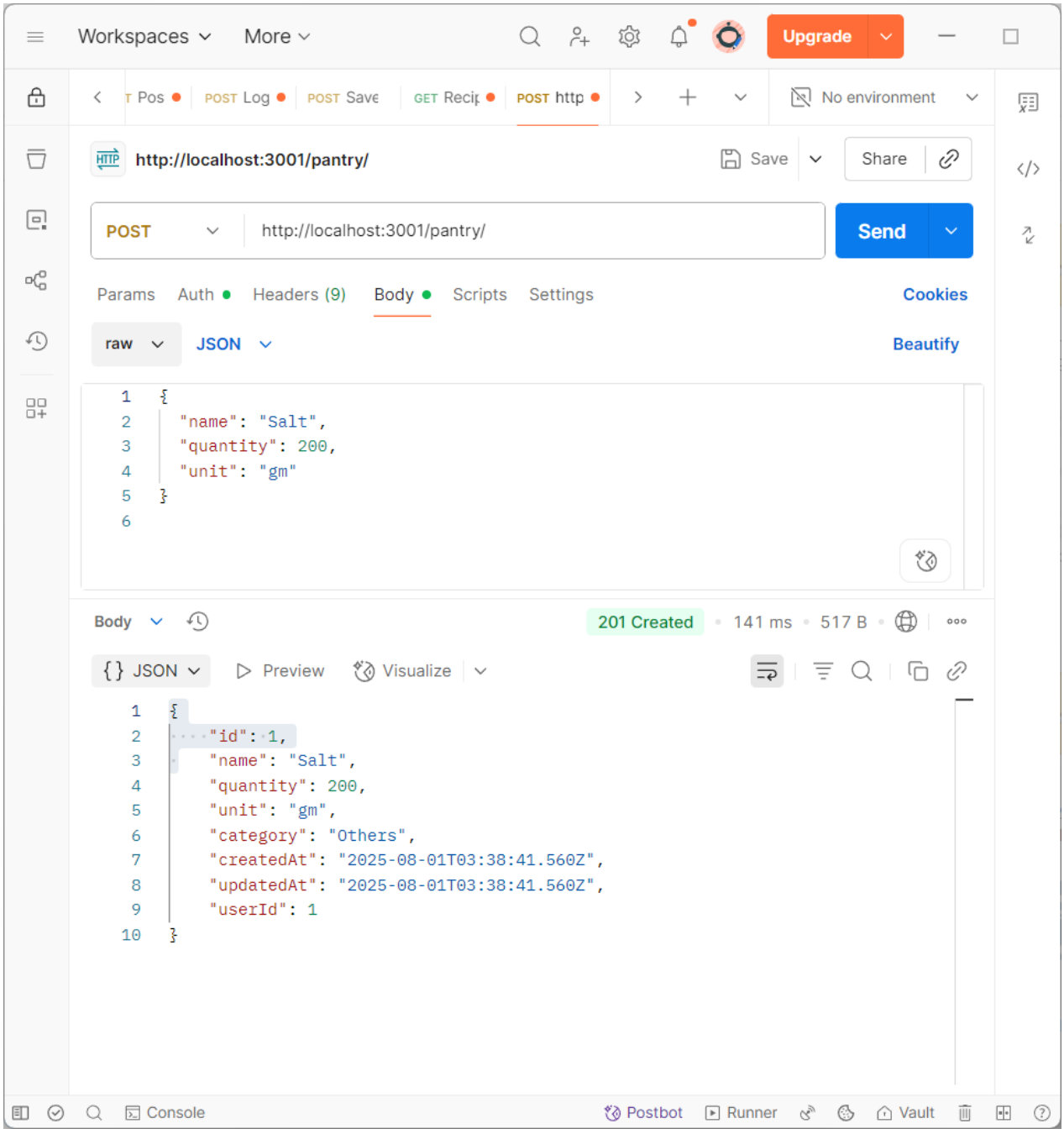


Figure 53: Adding items to pantry API testing

4.3.24. Delete Pantry items

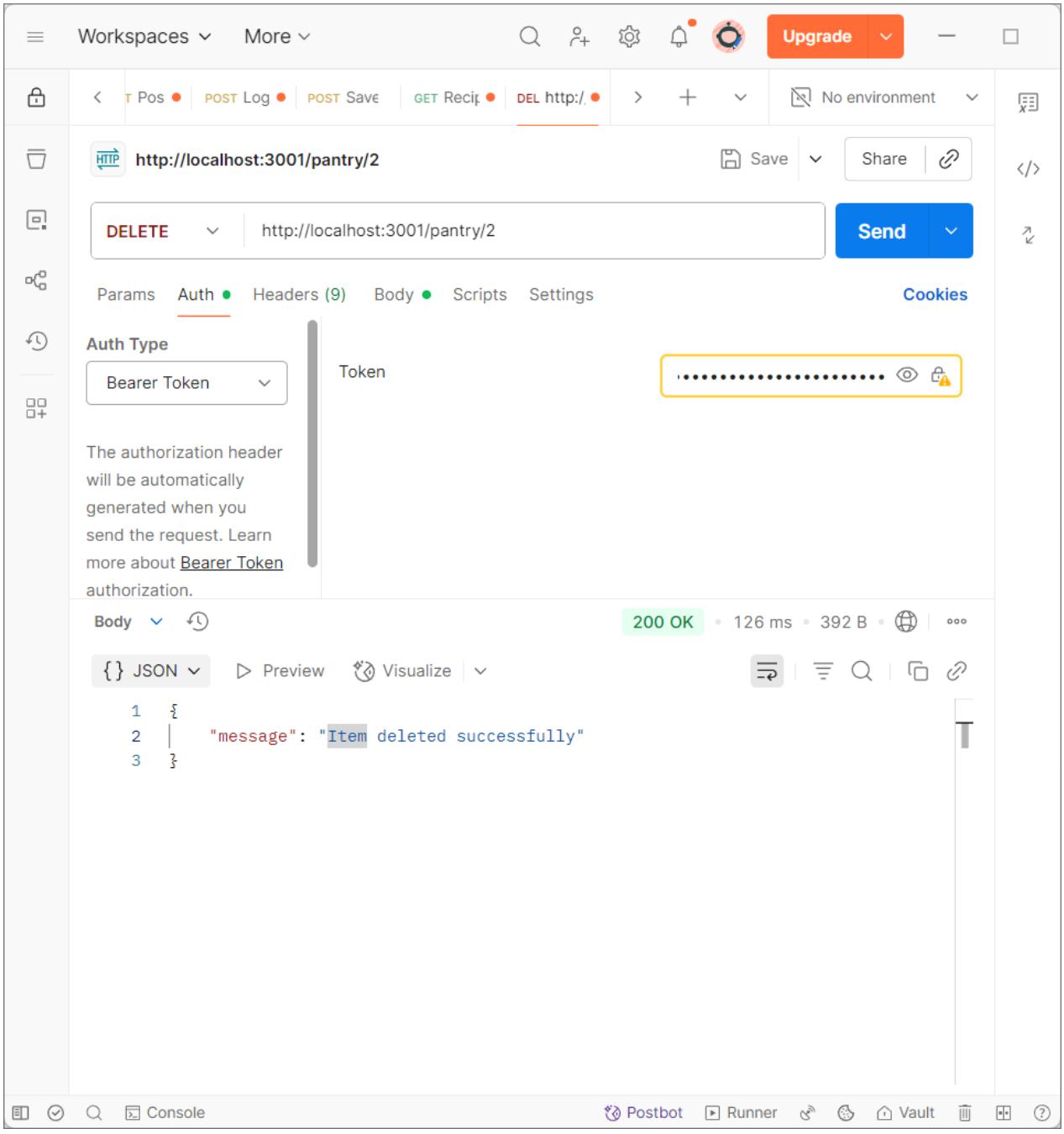


Figure 54: Deleting items from pantry API testing

4.3.25. Save search tags

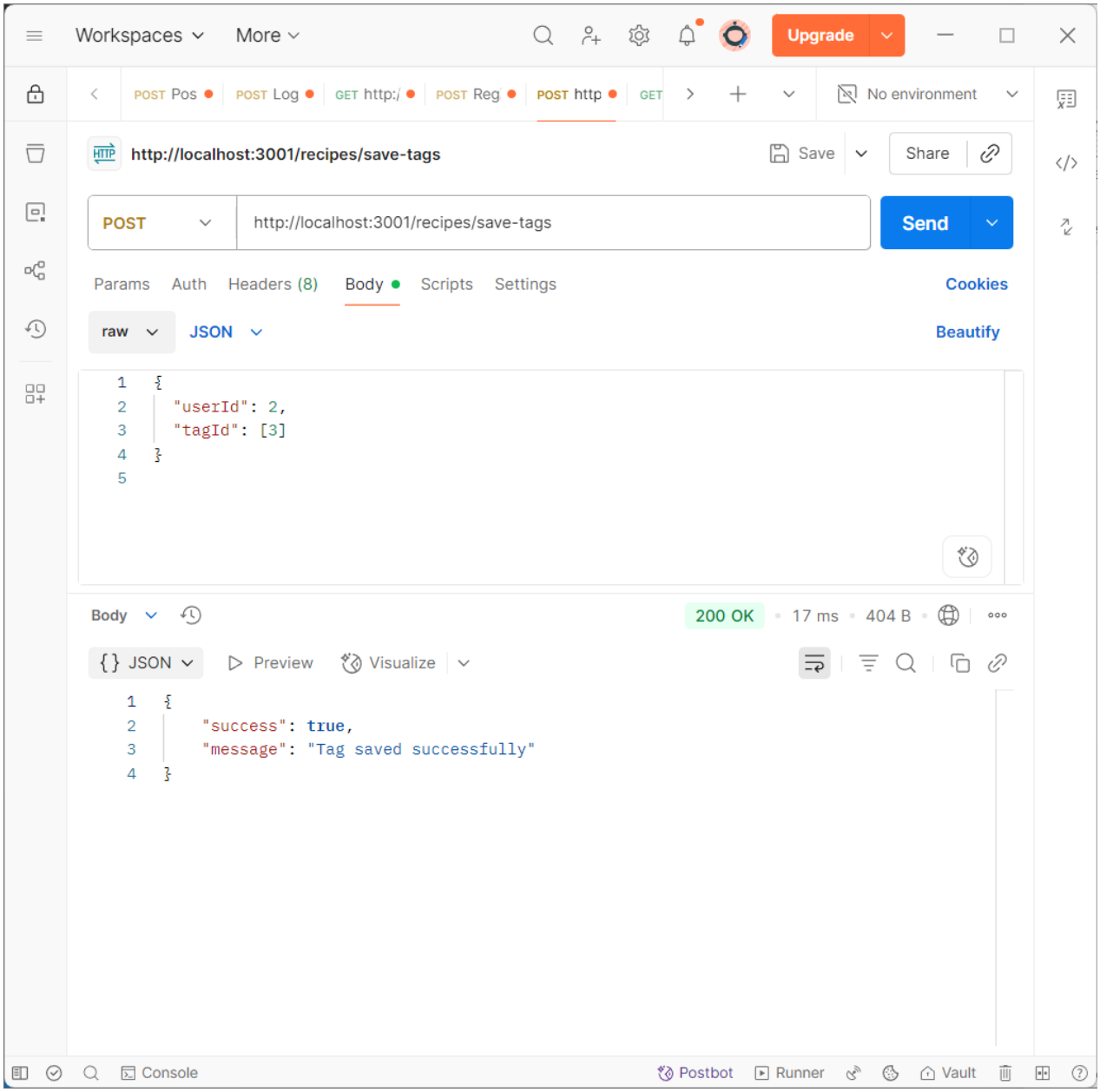


Figure 55: Save tags from the user's search API testing

4.3.26. Get profile

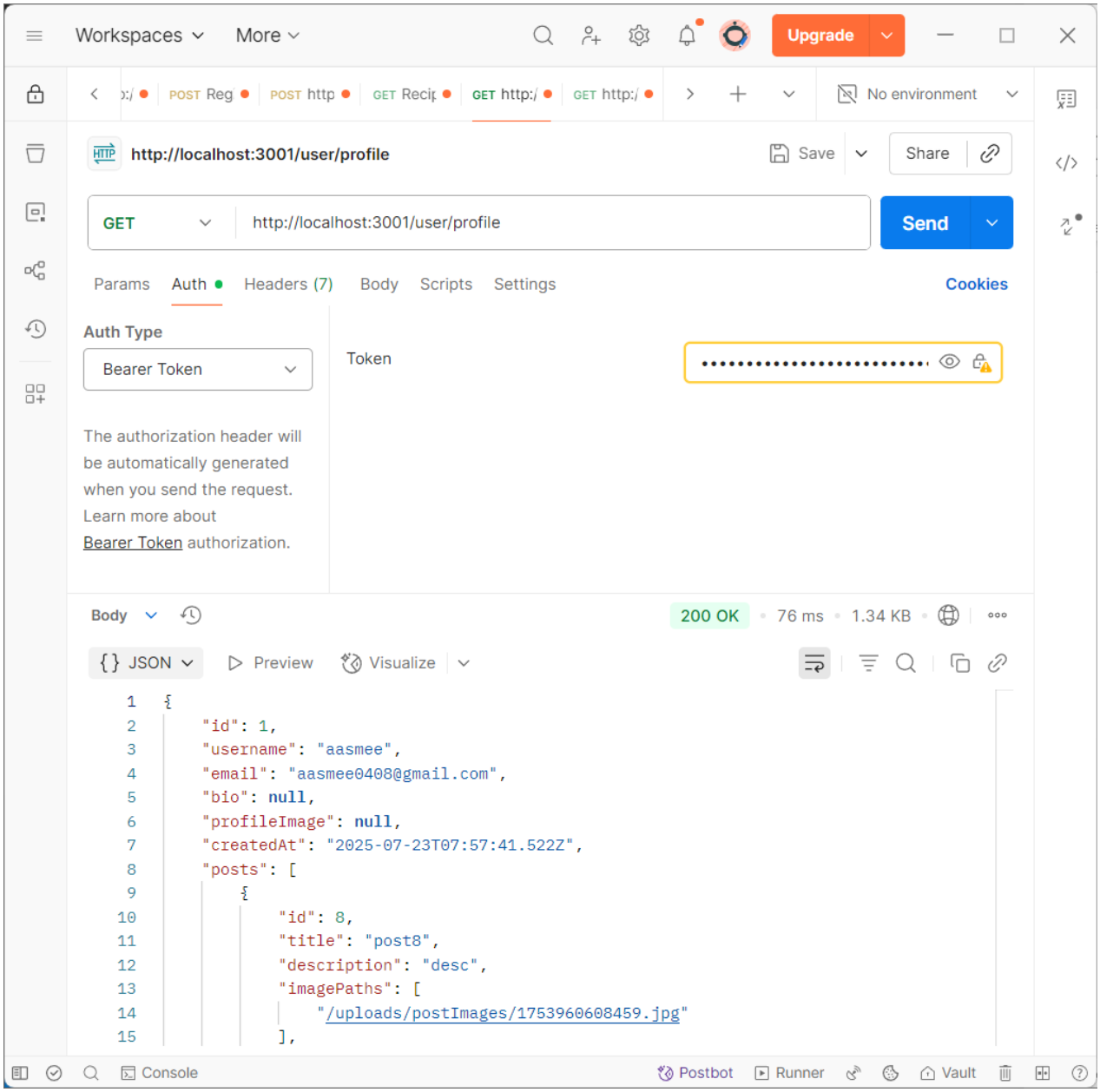


Figure 56: Get user profile API testing

4.3.27. Recommendations

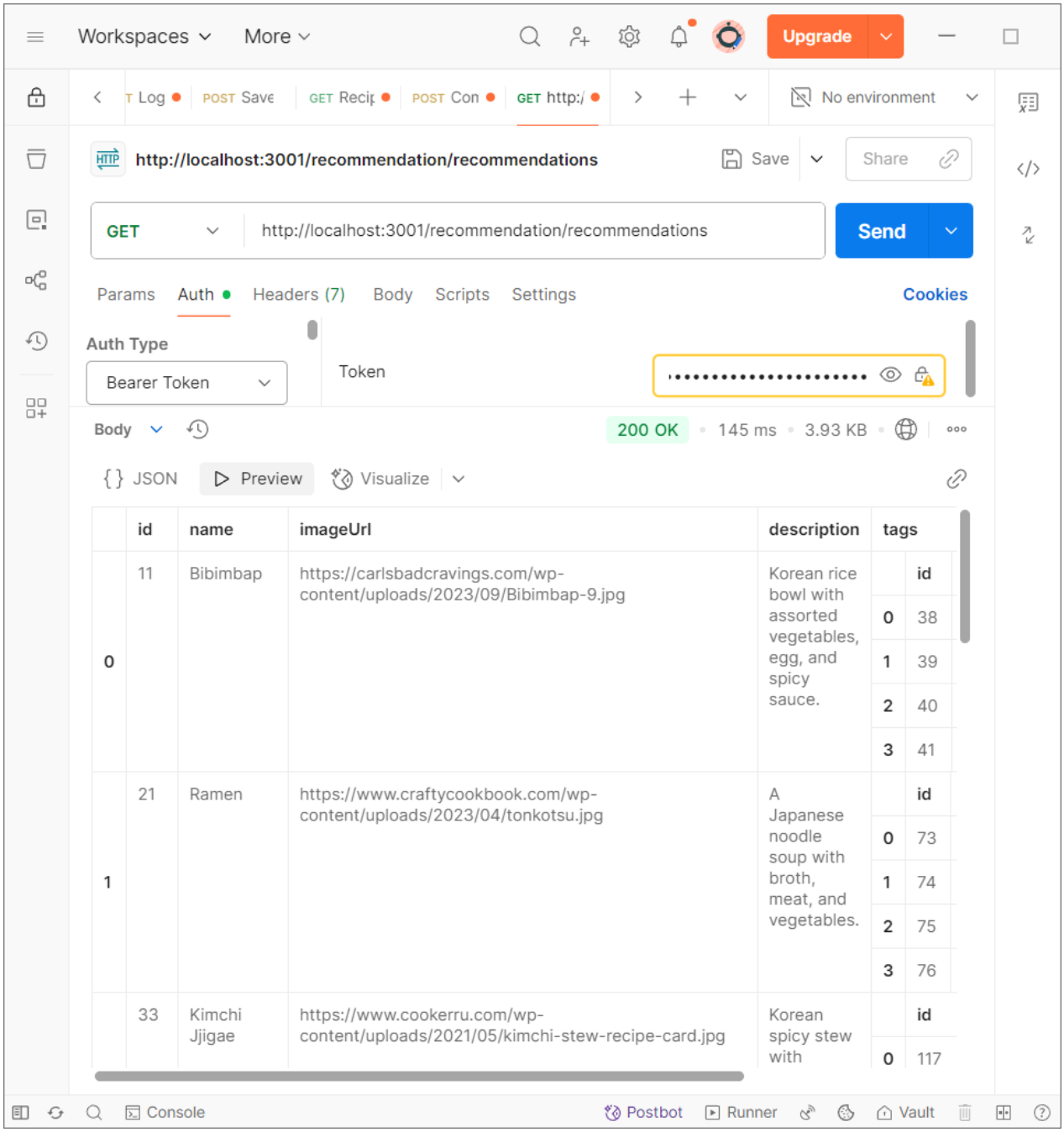


Figure 57: Recommendation system API testing

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 Digram 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://theowlsolutions.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

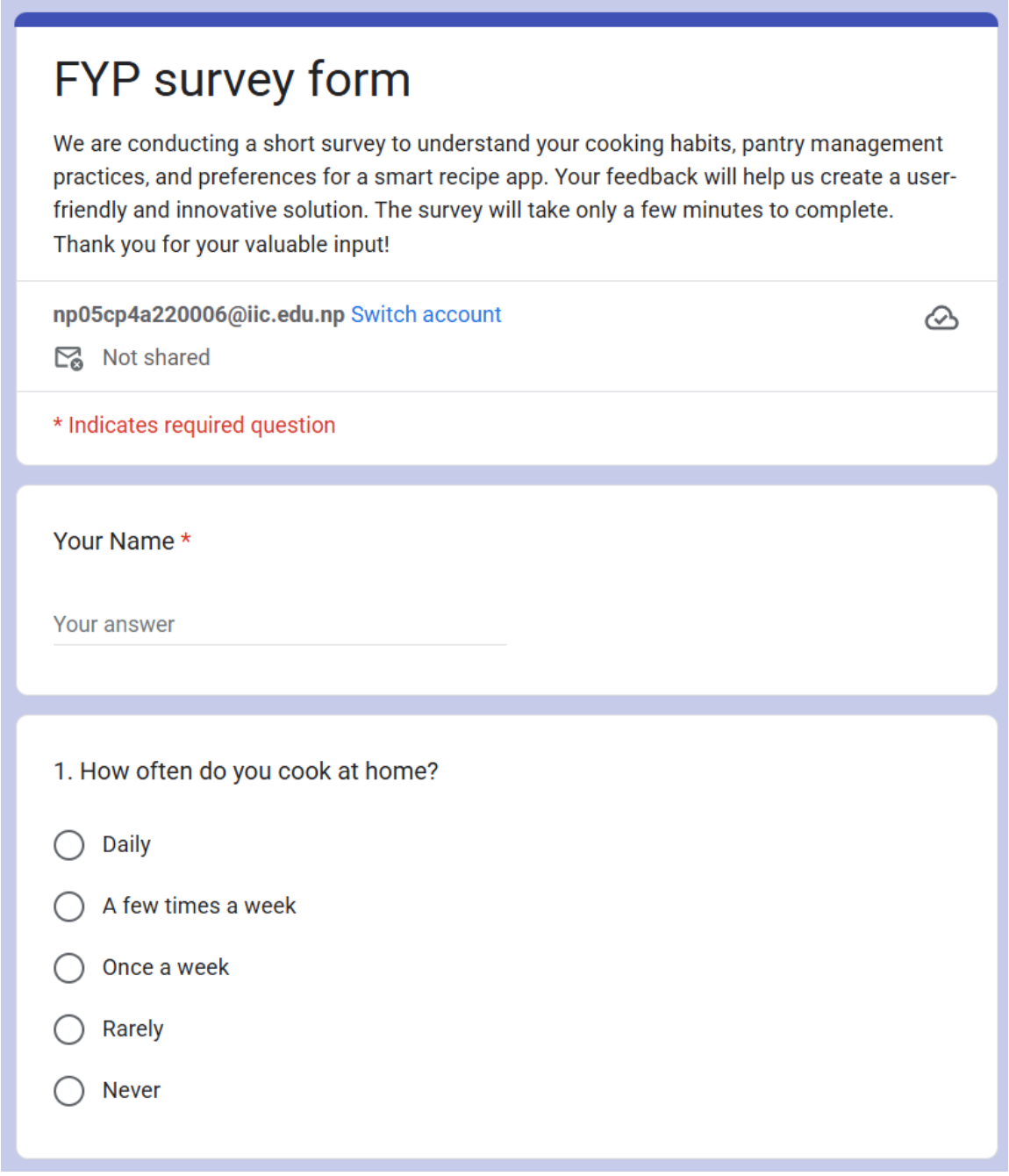


Figure 58: Pre-Survey Form pt.1

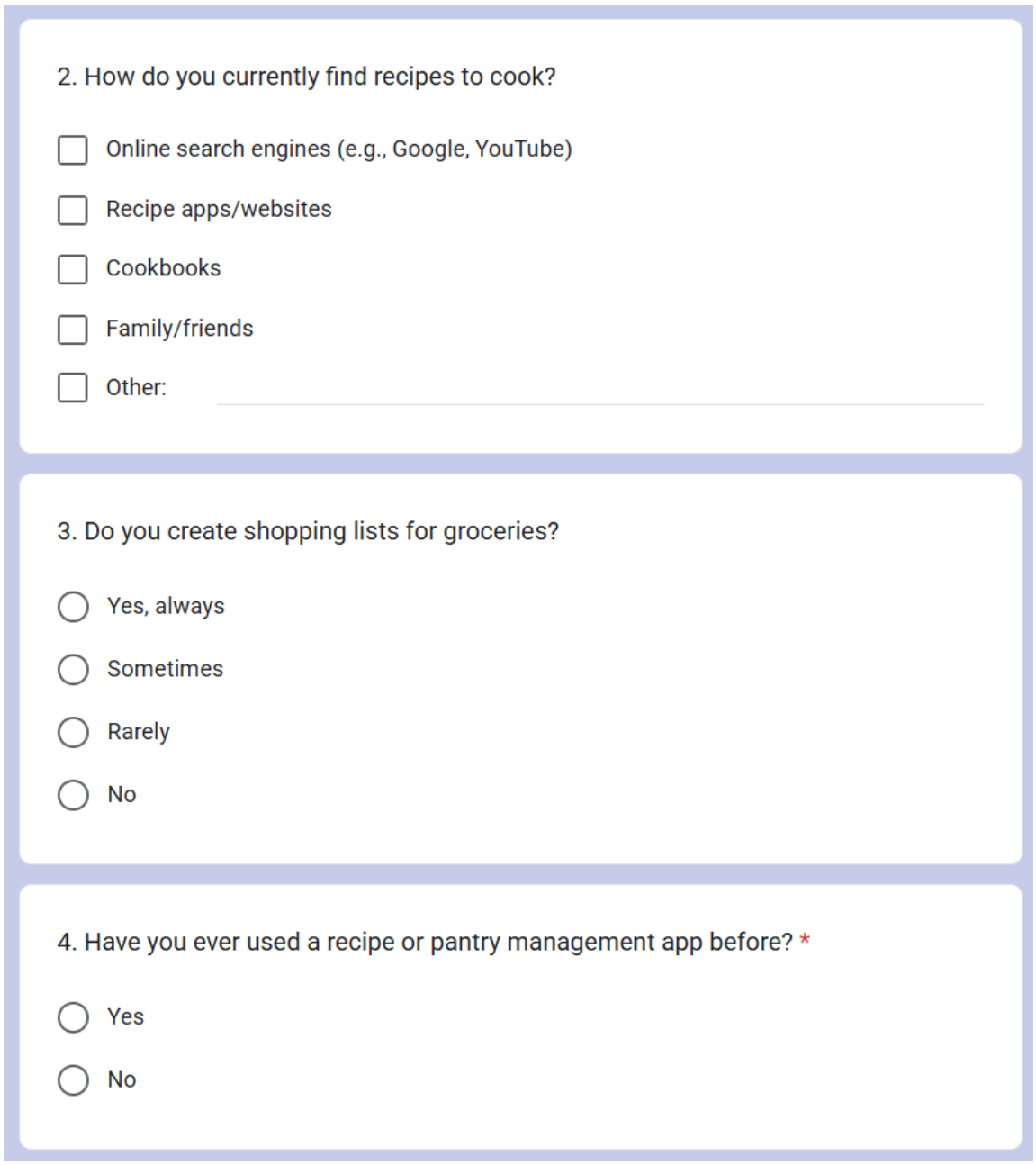


Figure 59: Pre-Survey Form pt.2

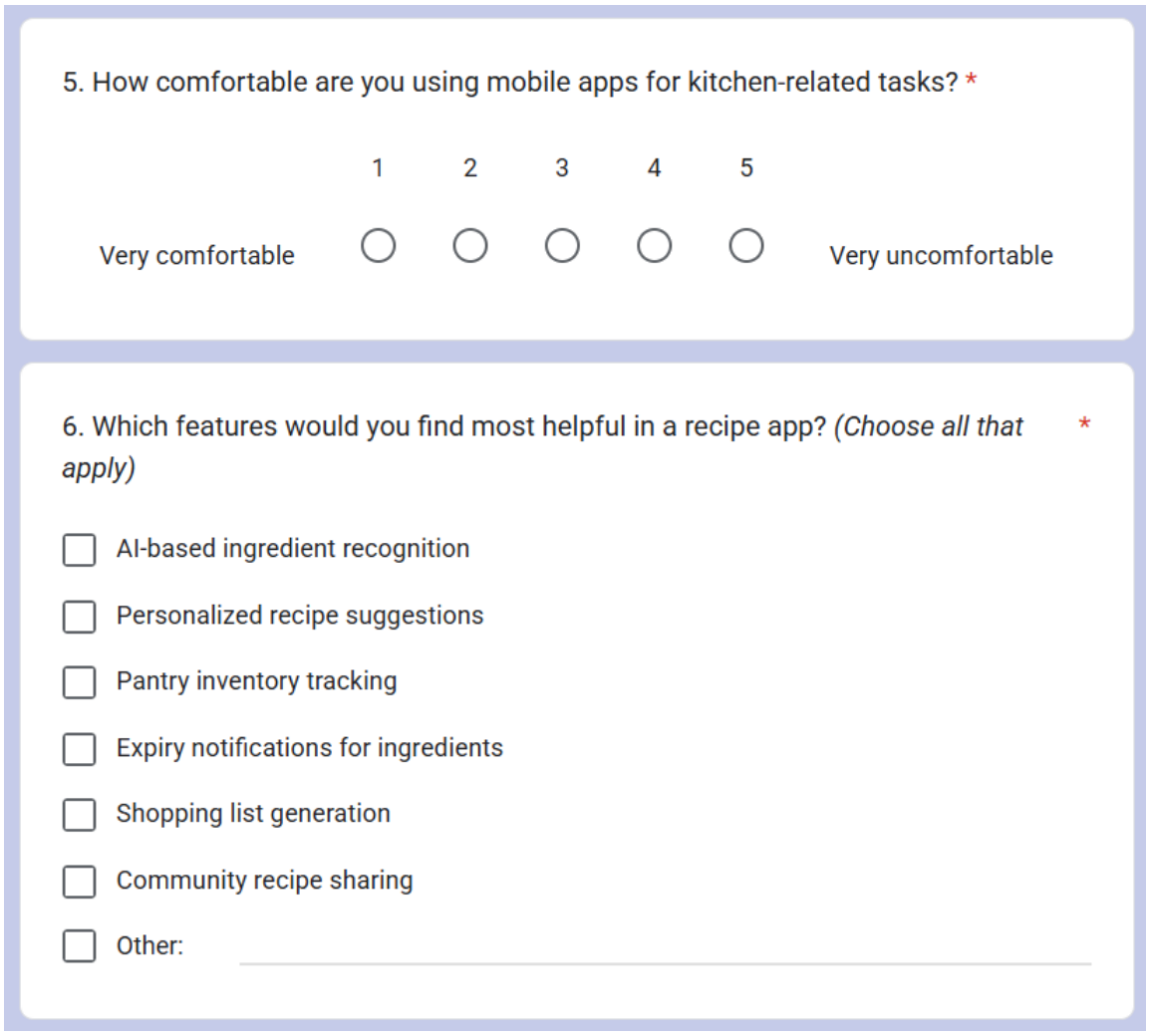


Figure 60: Pre-Survey Form pt.3

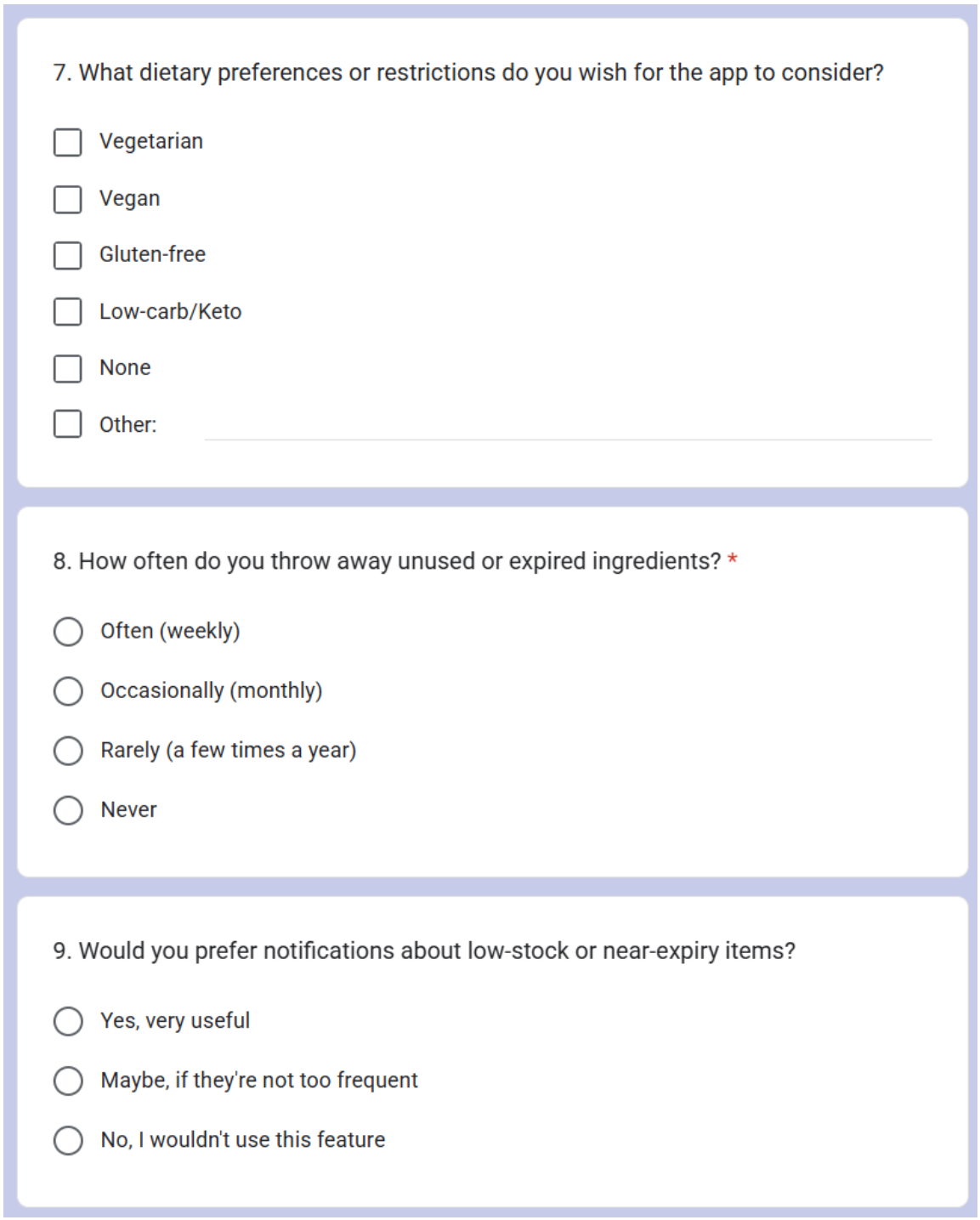


Figure 61: Pre-Survey Form pt.4

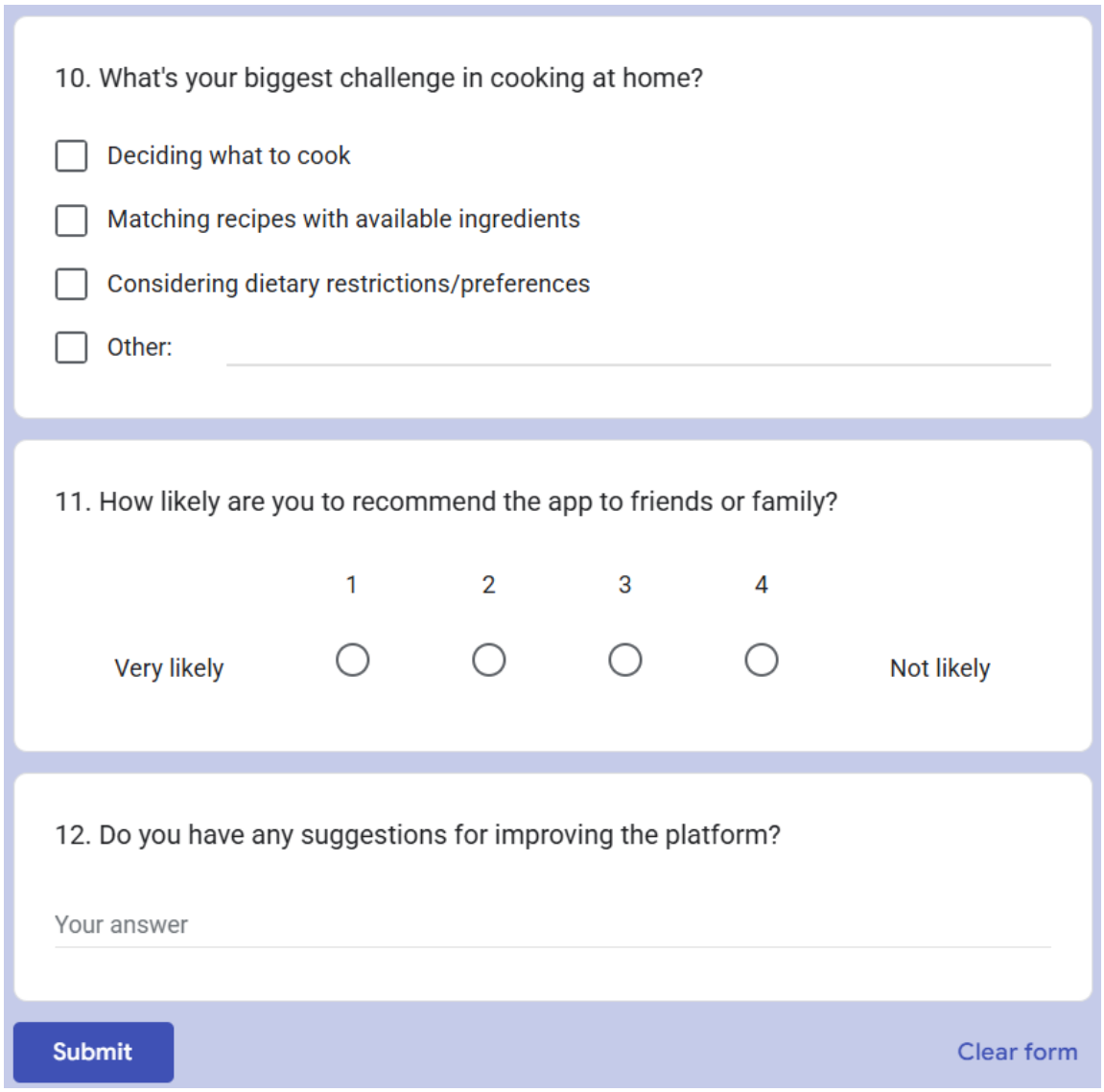
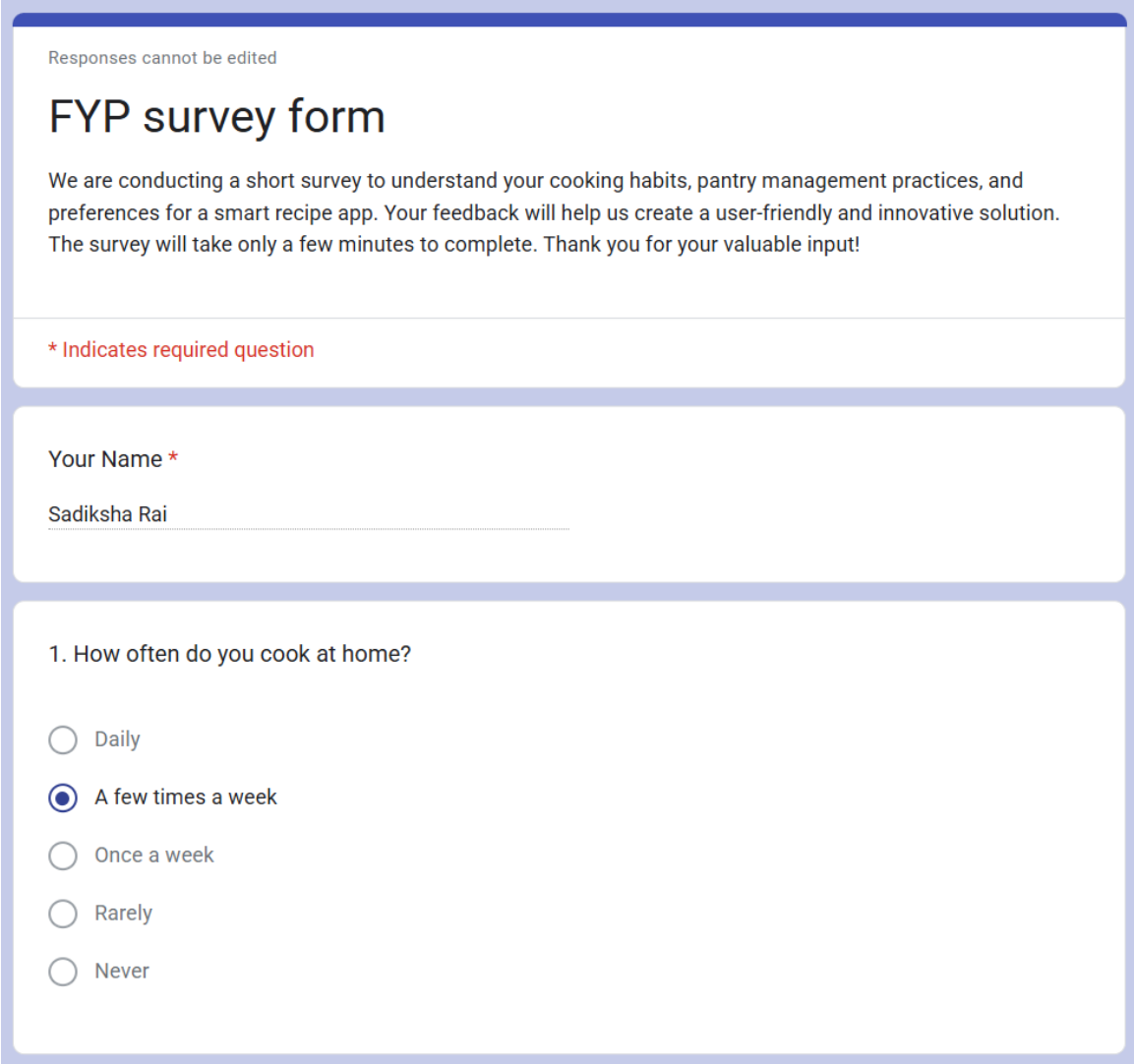


Figure 62: Pre-Survey Form pt.5

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

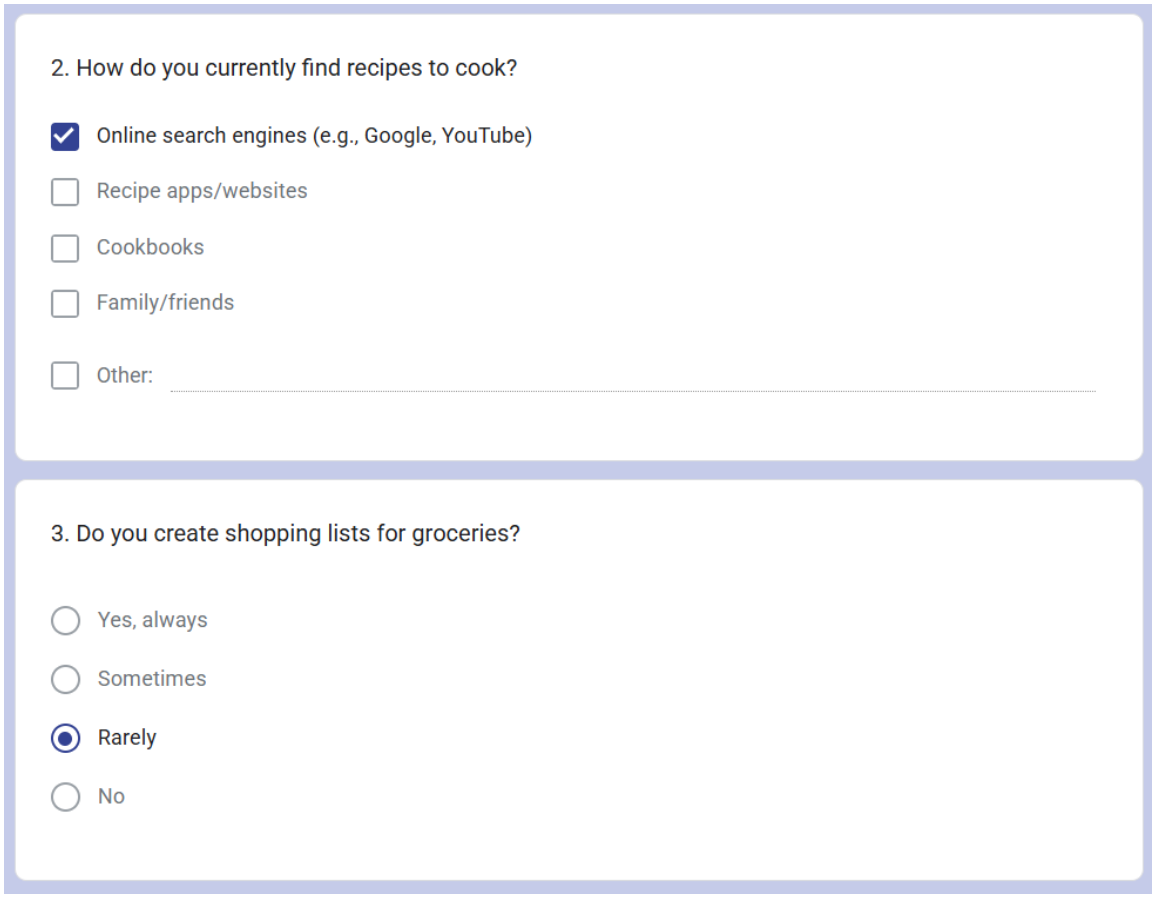


Figure 64: Filled Pre-Survey Form Sample pt.2

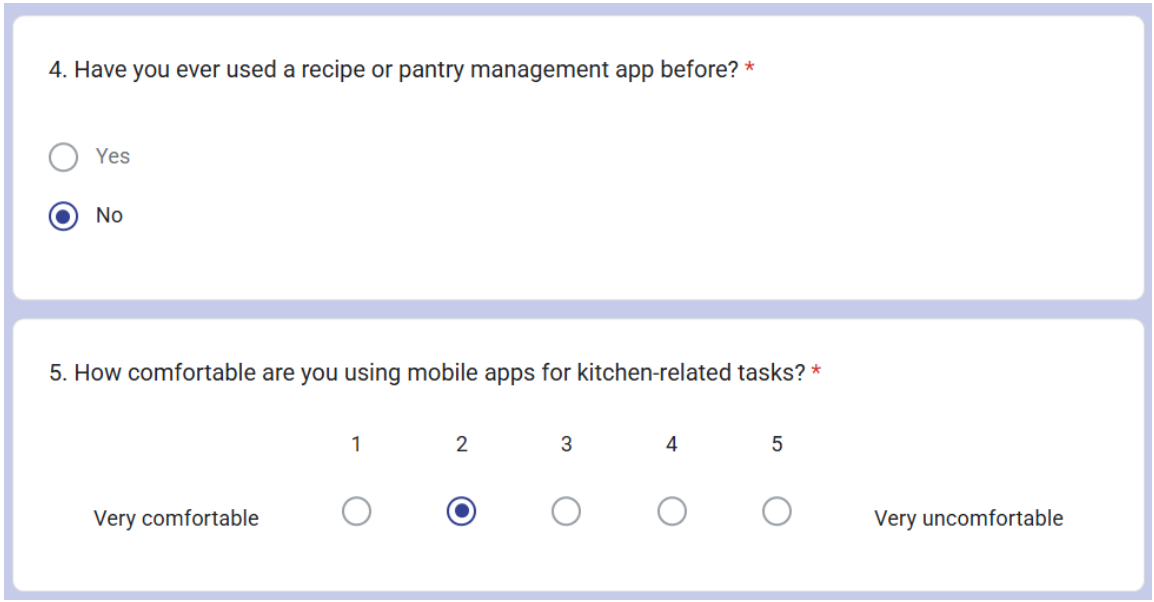


Figure 65: Filled Pre-Survey Form Sample pt.3

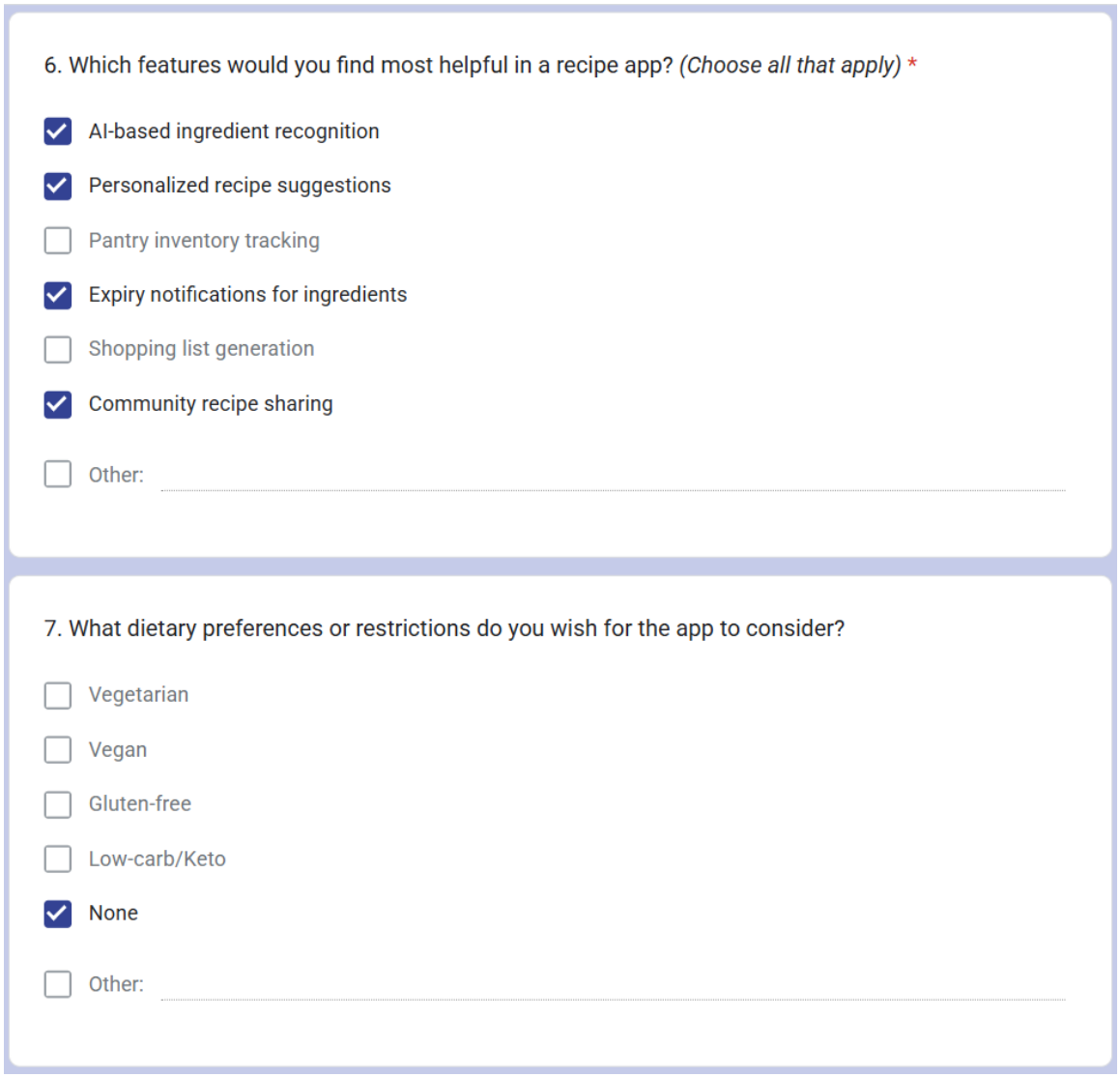


Figure 66: Filled Pre-Survey Form Sample pt.4

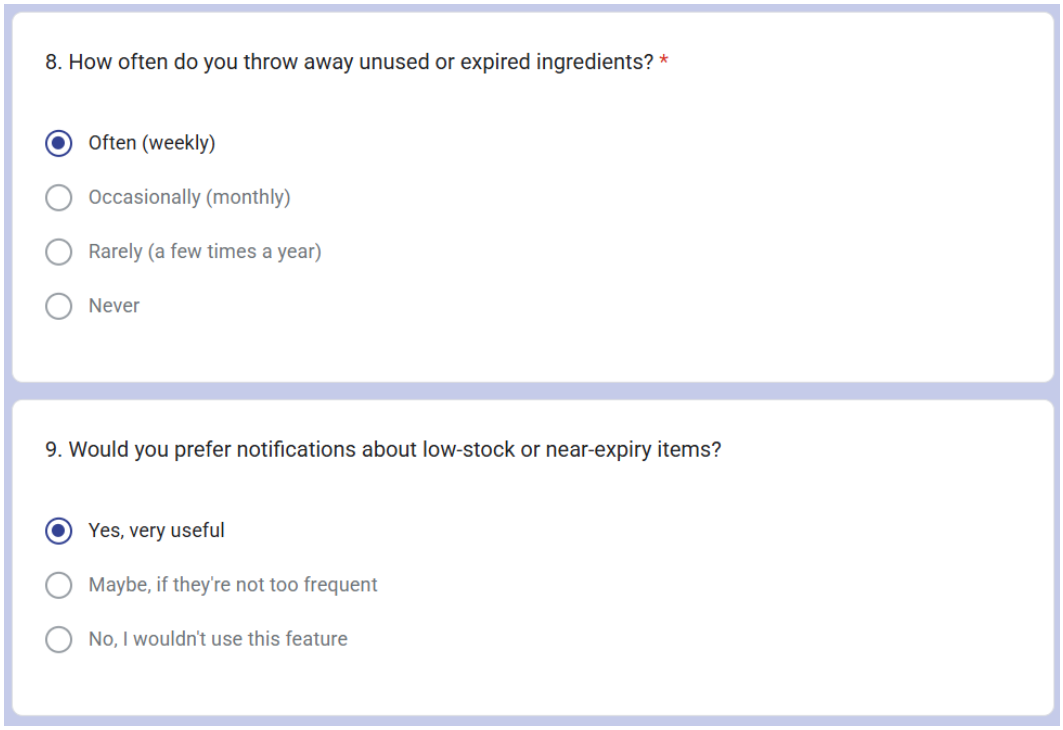


Figure 67: Filled Pre-Survey Form Sample pt.5

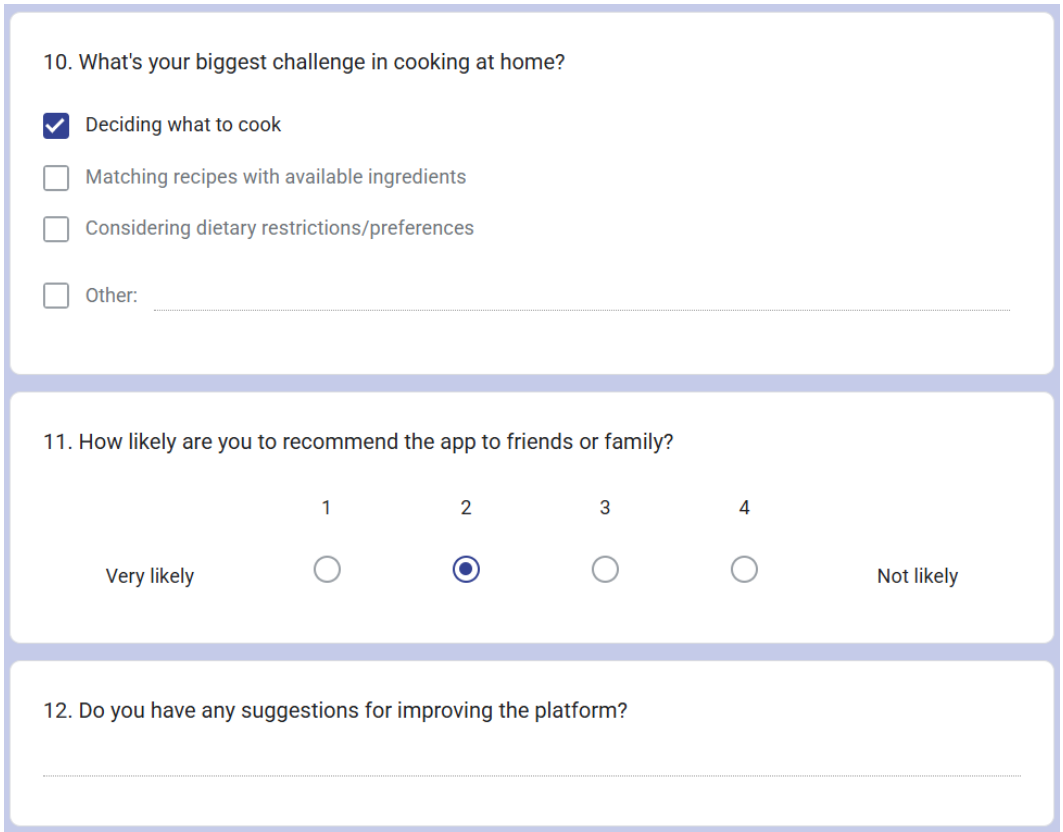


Figure 68: Filled Pre-Survey Form Sample pt.6

8.1.3. Pre-Survey Result

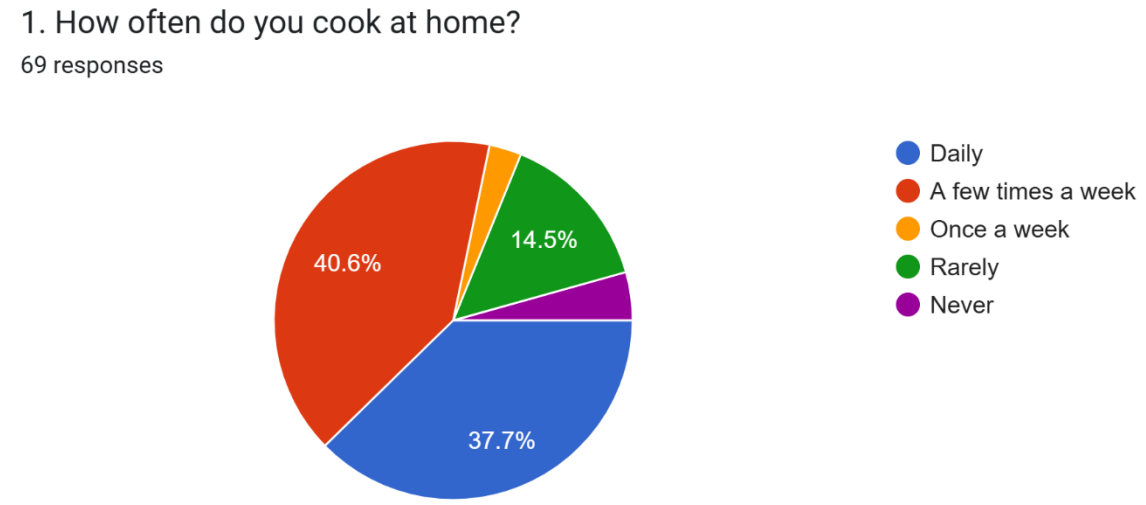


Figure 69: Pre-Survey Q1

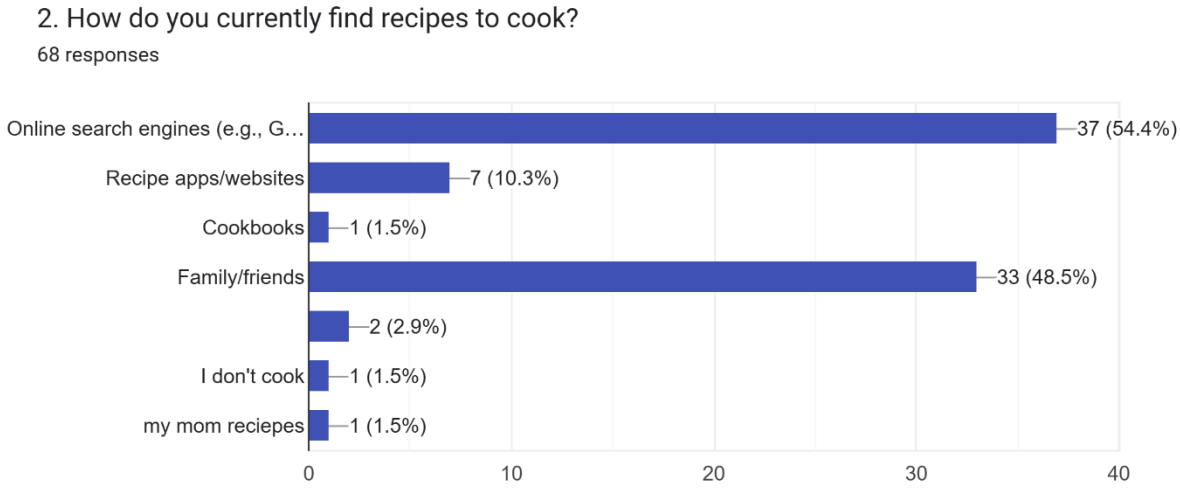


Figure 70: Pre-Surgery Q2

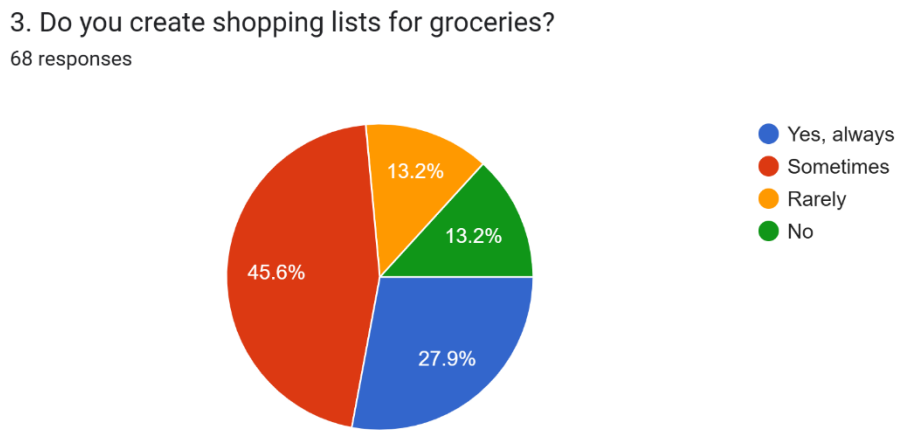


Figure 71: Pre-Survey Q3

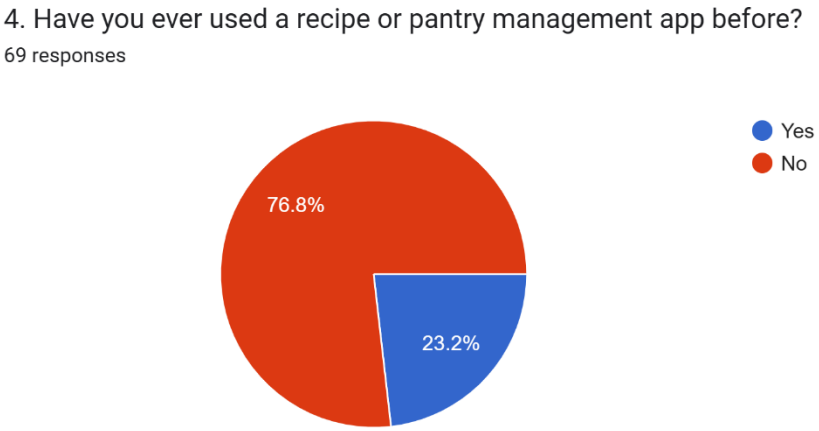


Figure 72: Pre-Survey Q4

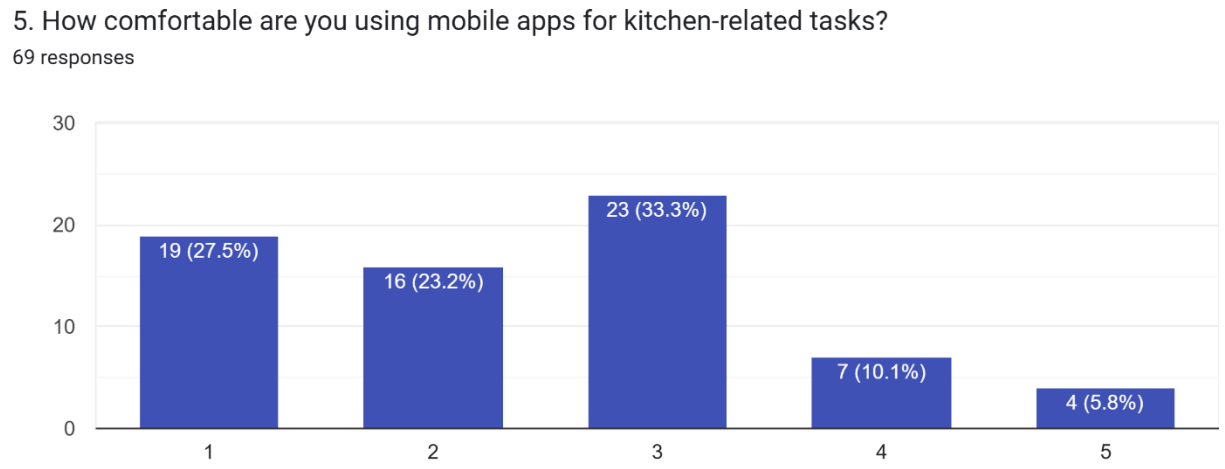


Figure 73:Pre-Survey Q5

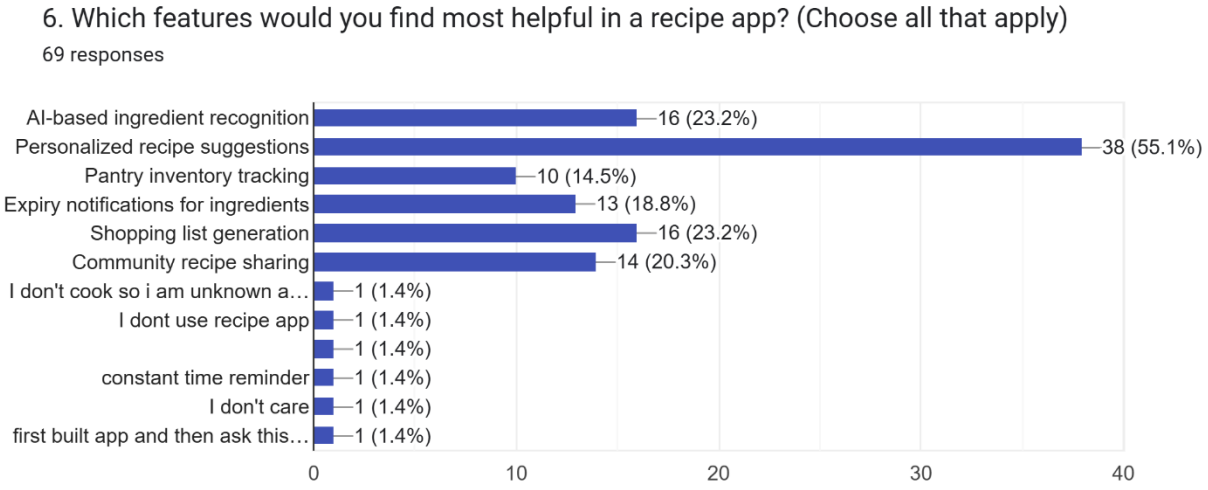


Figure 74: Pre-Survey Q6

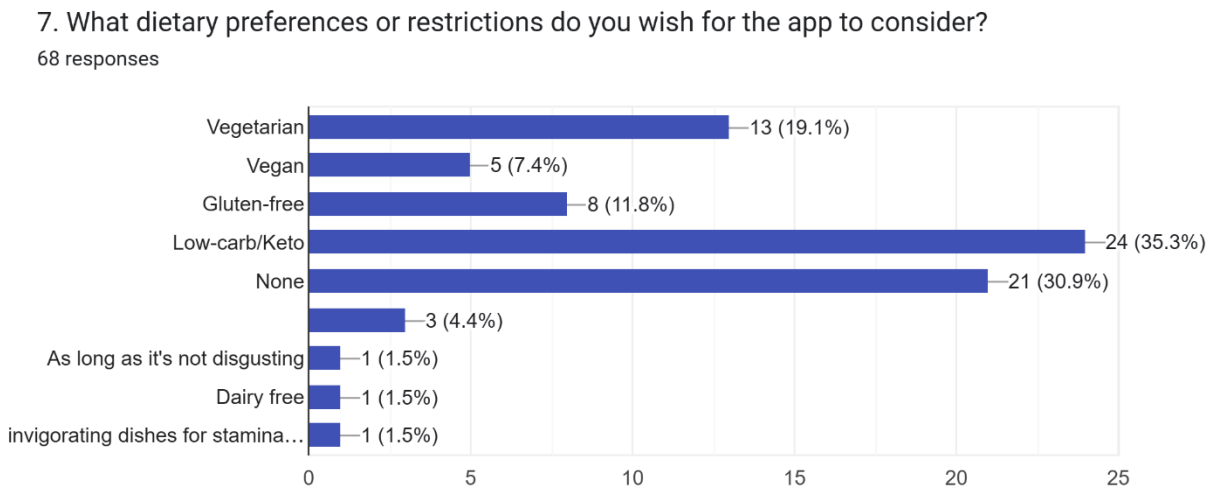


Figure 75: Pre-Survey Q7

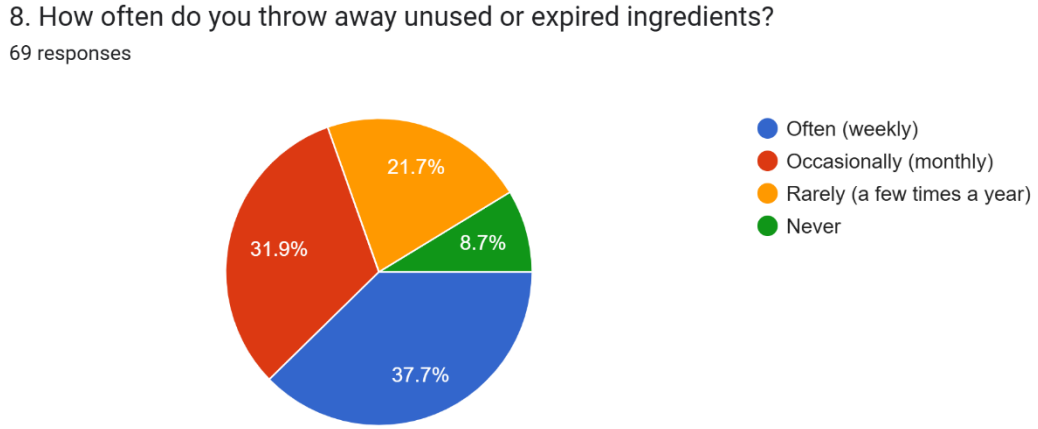


Figure 76: Pre-Survey Q8

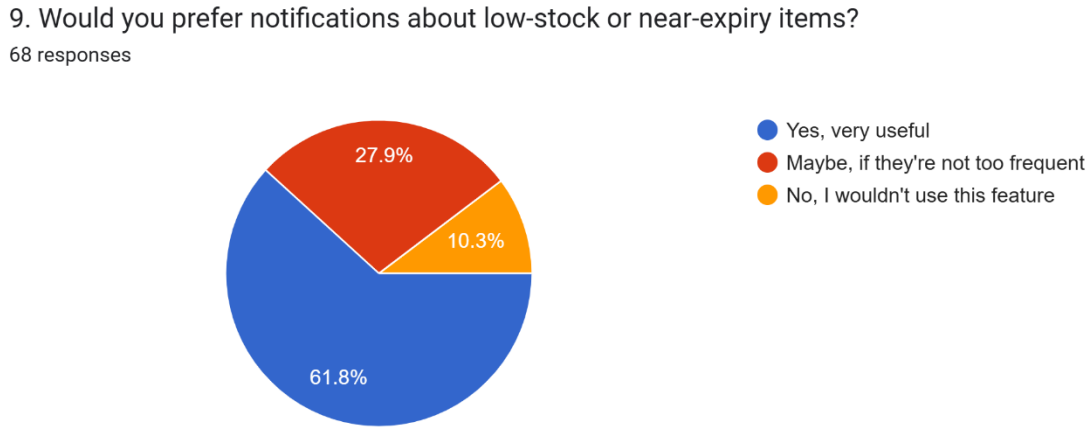


Figure 77: Pre-Survey Q9

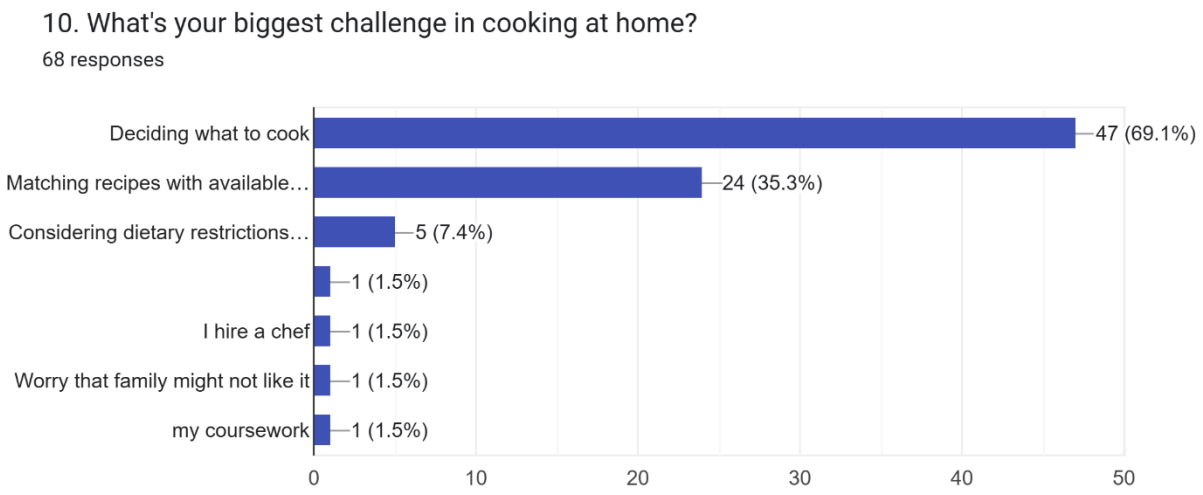


Figure 78: Pre-Survey Q10

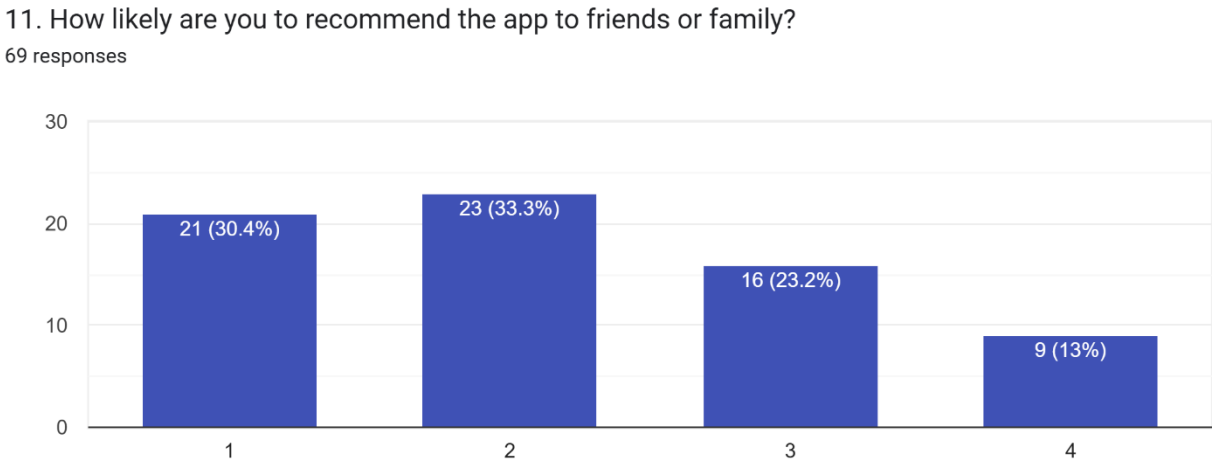


Figure 79: Pre-Survey Q11

8.2. Appendix B: Post-Survey

8.2.1. Post-Survey Form

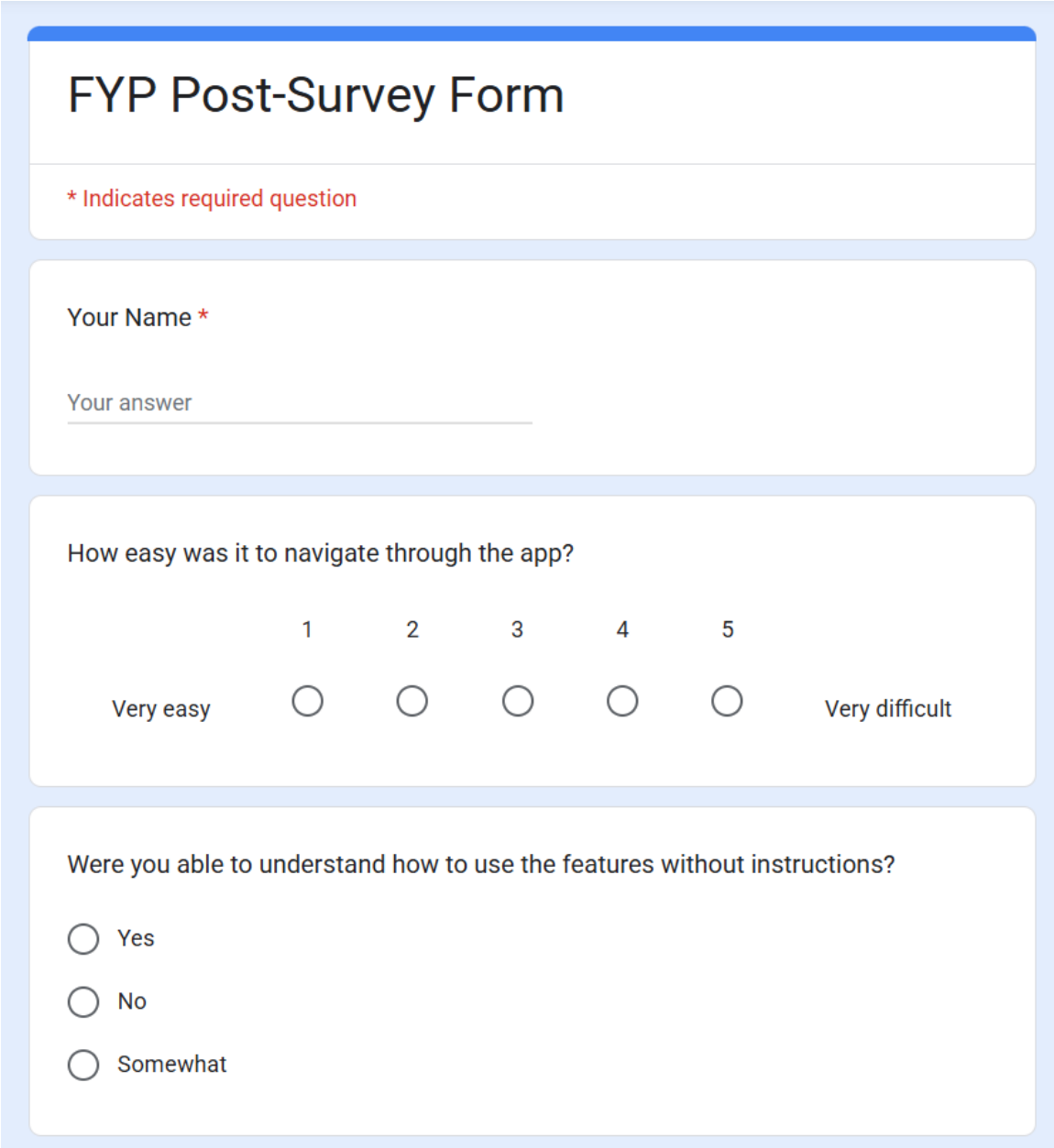


Figure 80: Post Survey Form pt.1

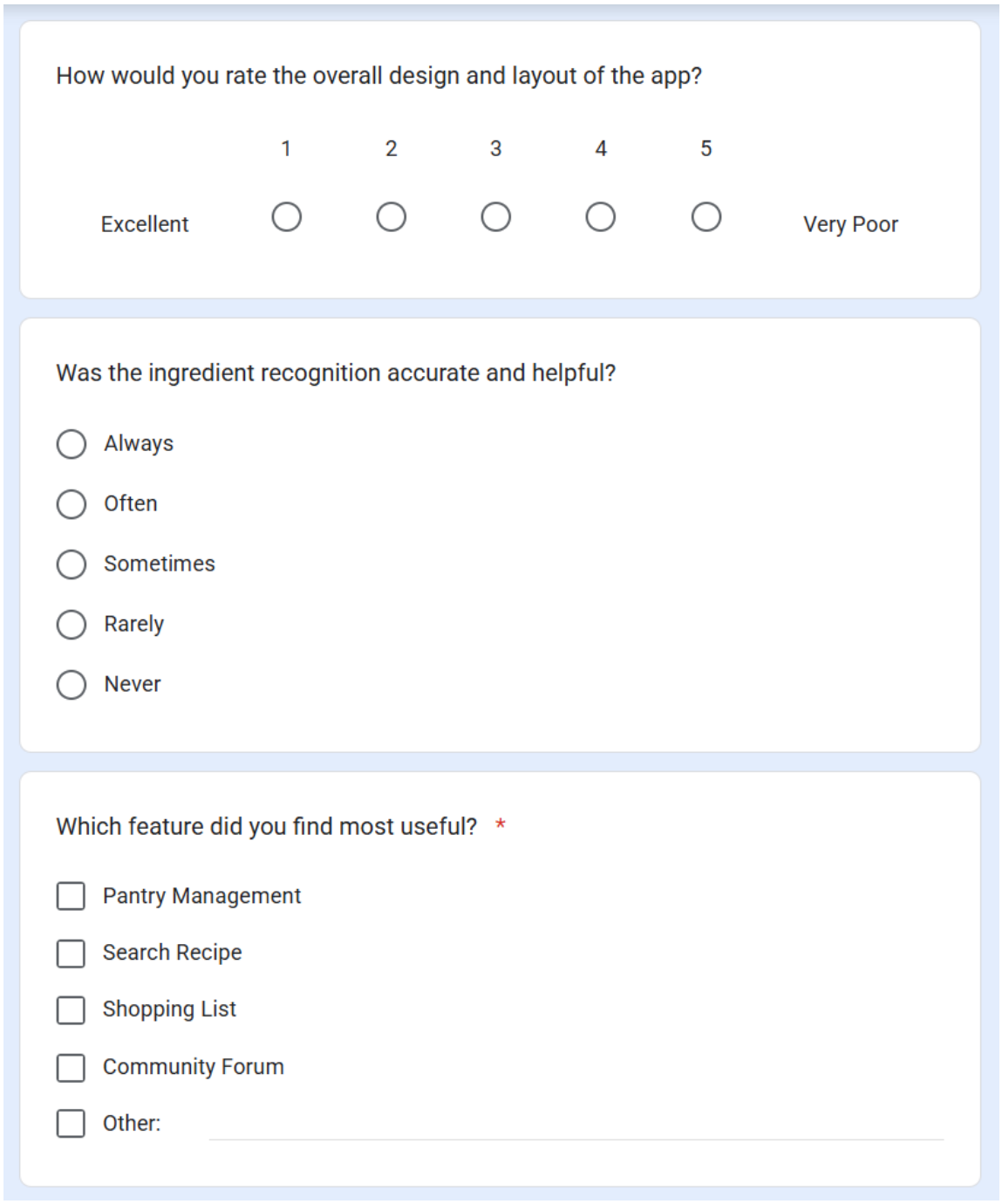


Figure 81: Post-Survey Form pt.2

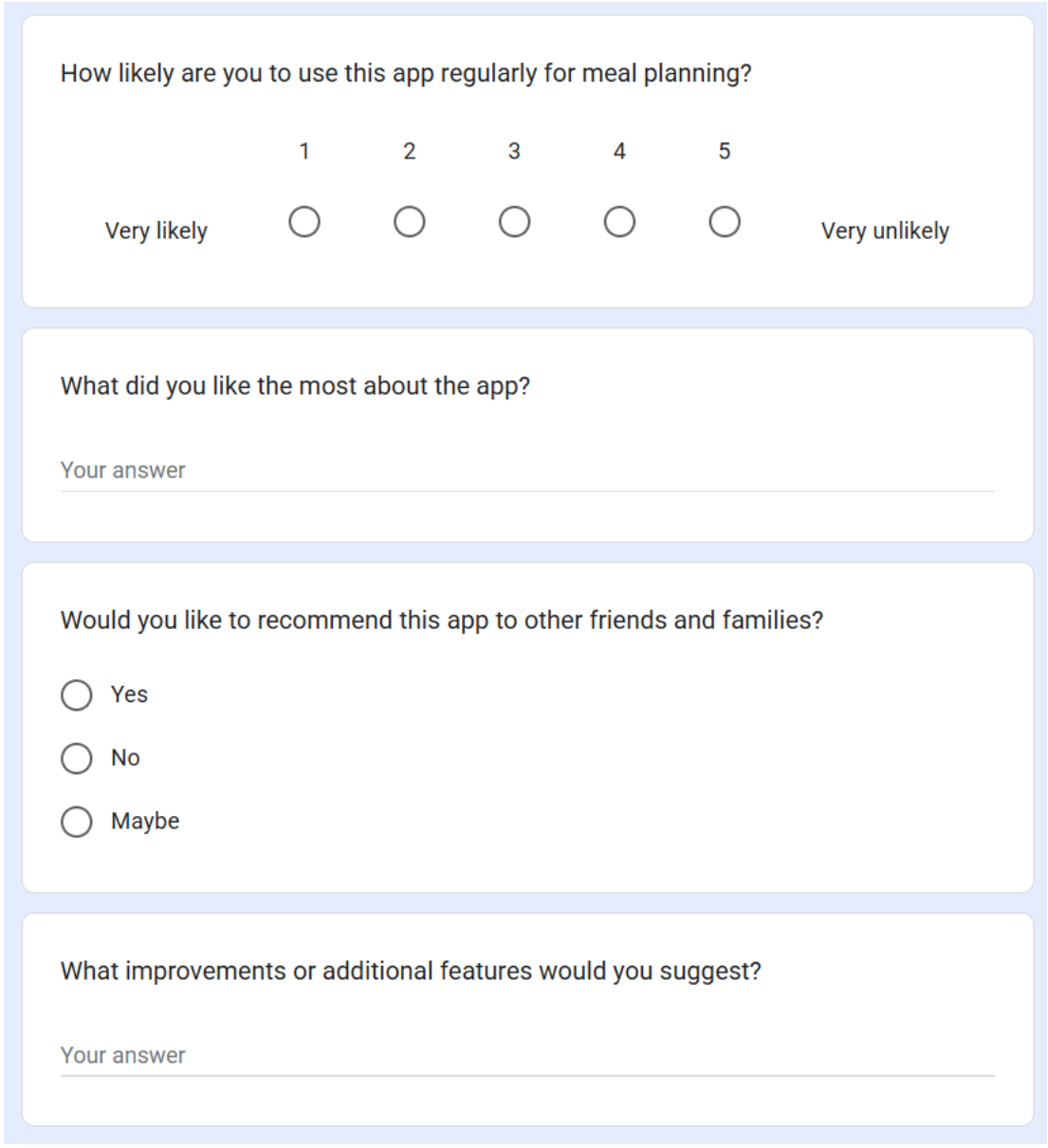


Figure 82: Post-Survey Form pt.3

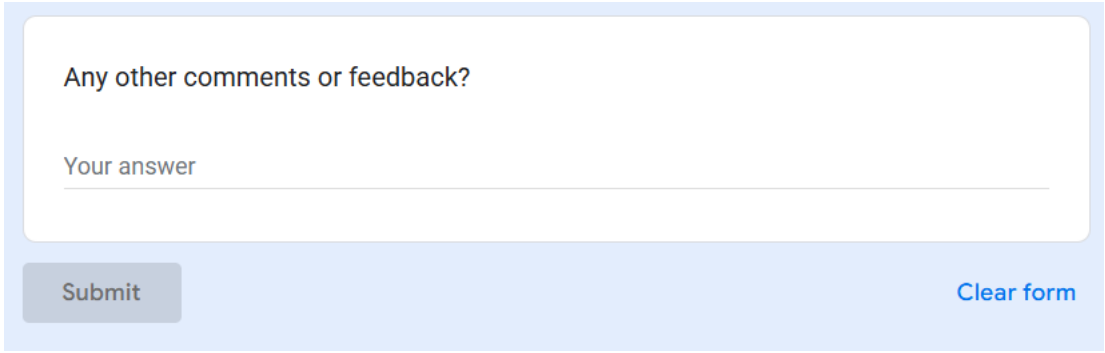


Figure 83: Post-Survey Form pt.4

8.2.2. Sample of Filled Post-Survey Forms

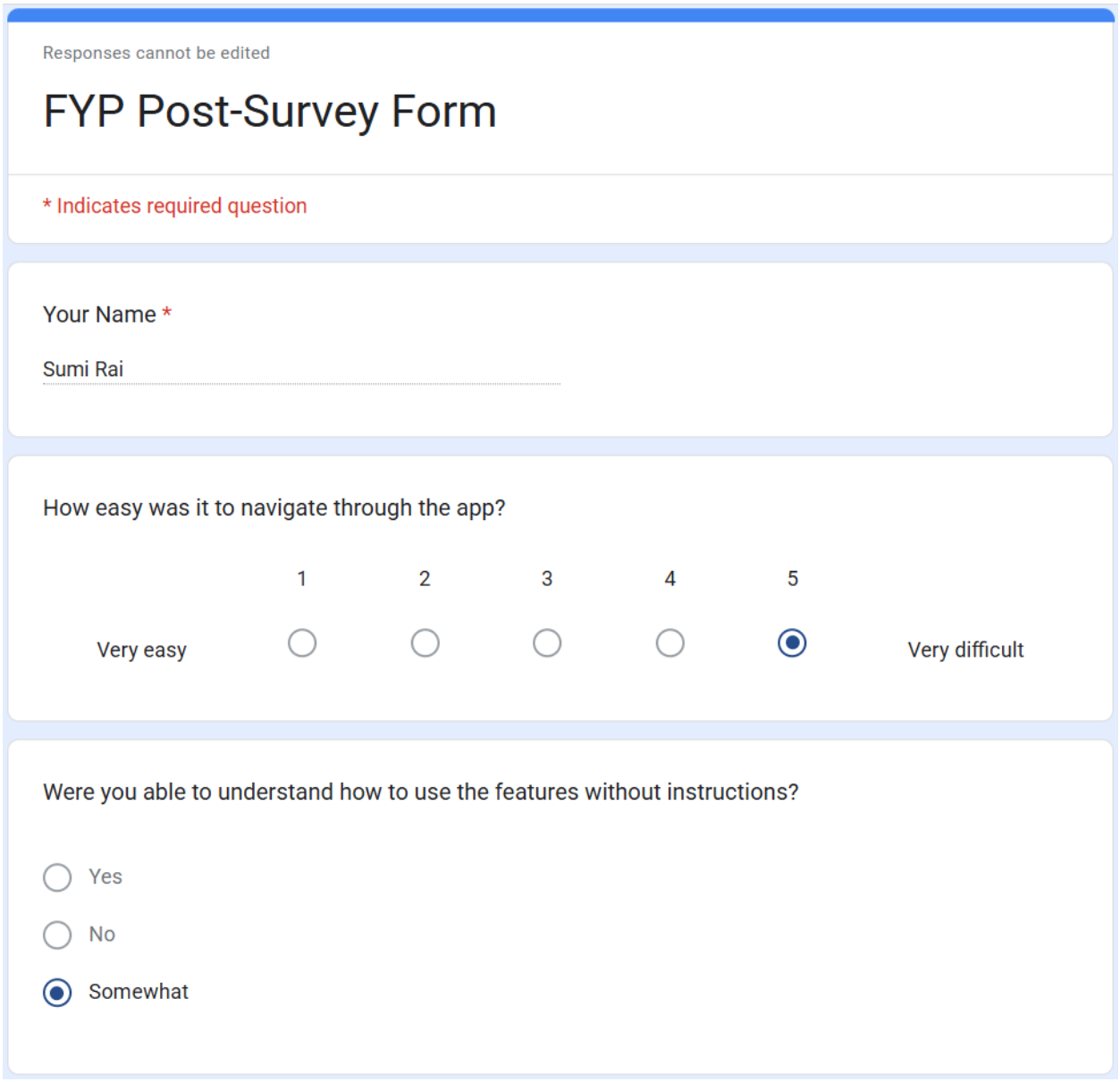


Figure 84: Post-Survey Smaple Form pt.1

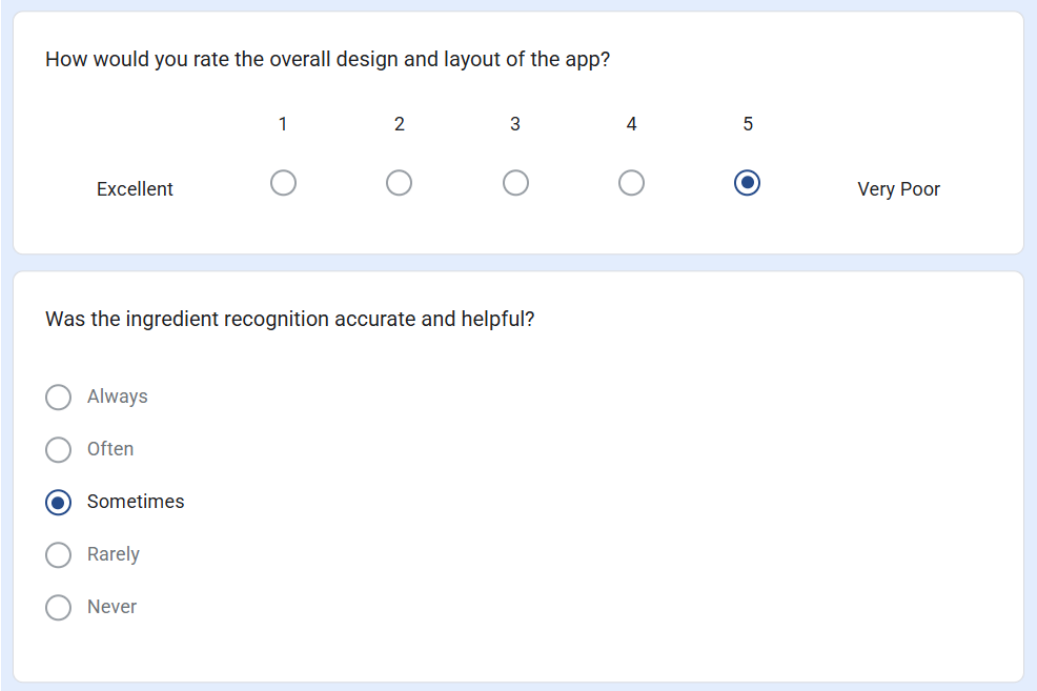


Figure 85: Post-Survey Sample Form pt.2

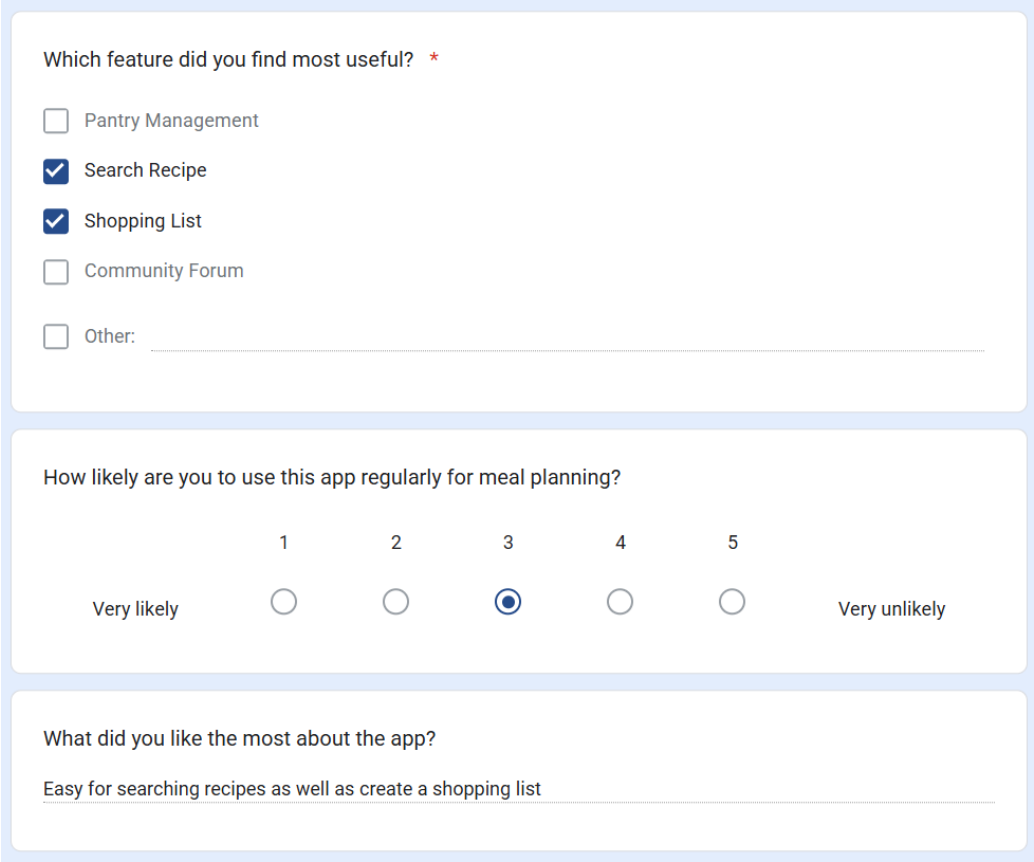


Figure 86: Post-Survey Sample Form pt.3

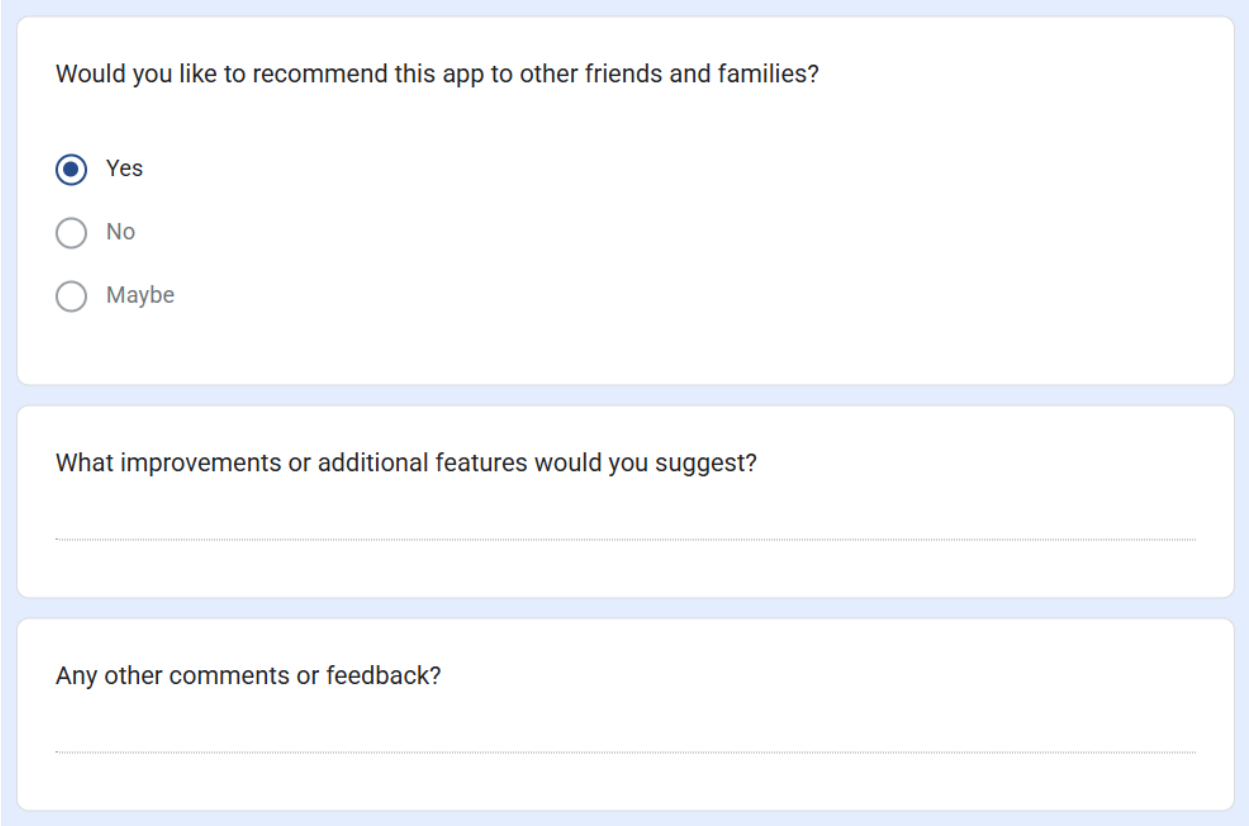


Figure 87: Post-Survey Sample Form pt.4

8.2.3. Post-Survey Result

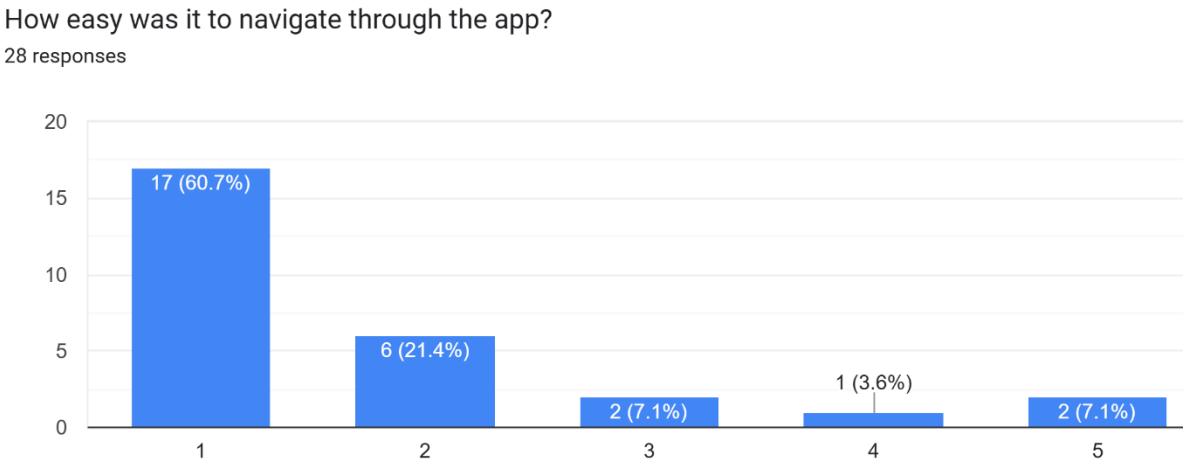


Figure 88:Post-Survey Q1

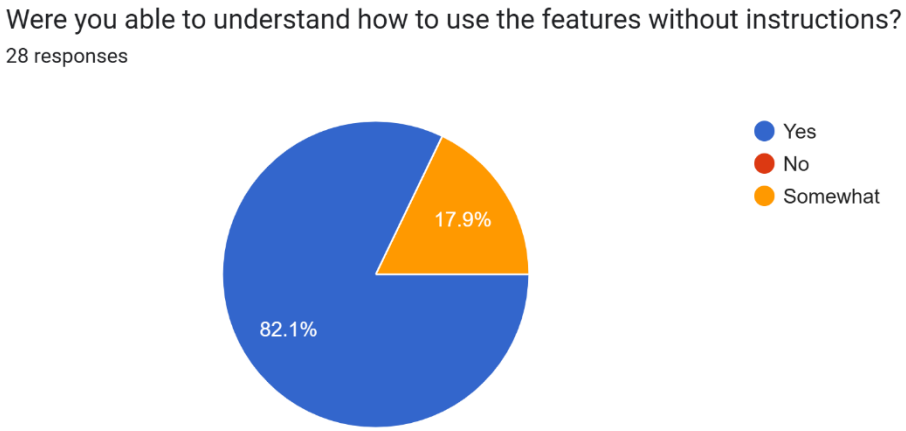


Figure 89: Post-Survey Q2

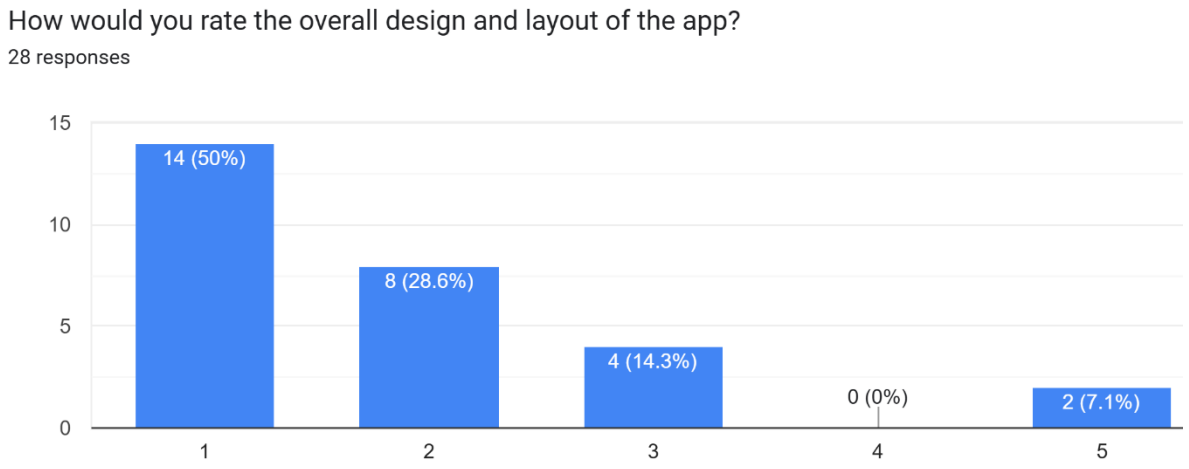


Figure 90: Post-Survey Q3

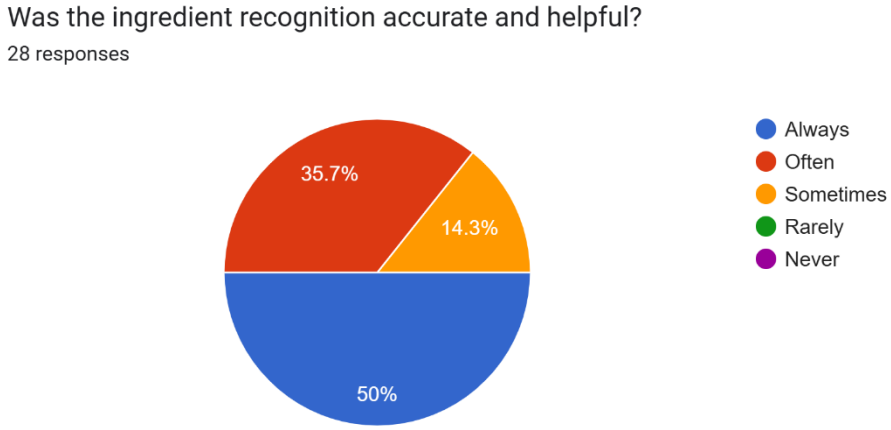


Figure 91: Post-Survey Q4

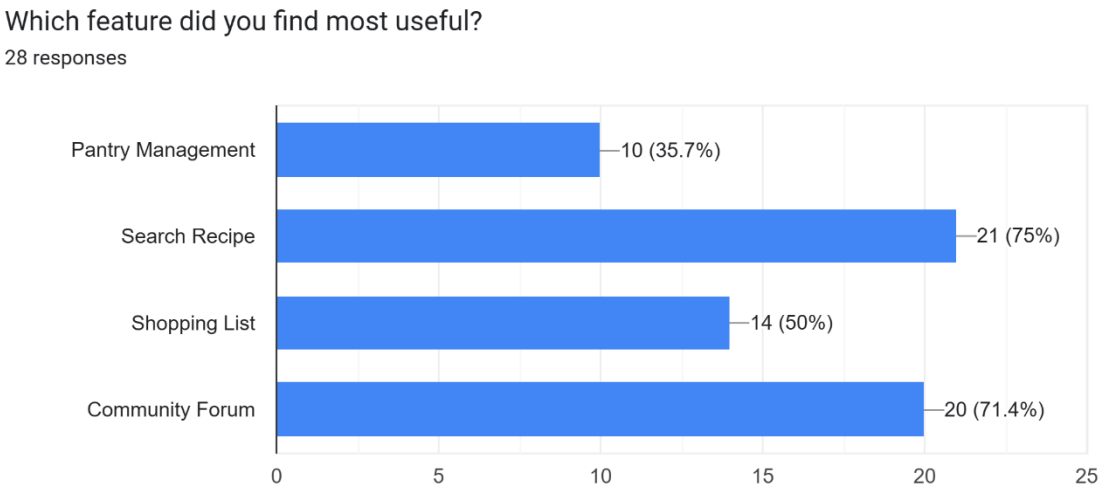


Figure 92: Post-Survey Q5

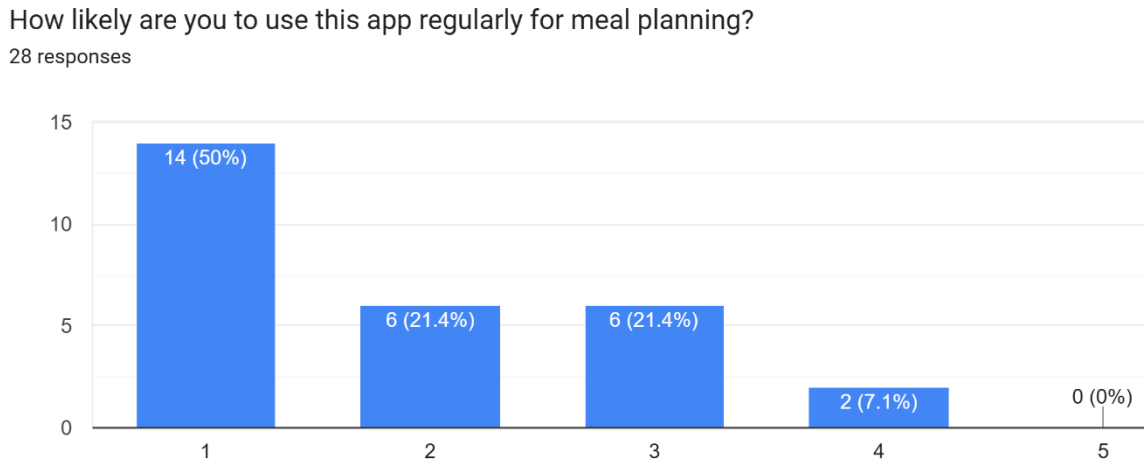


Figure 93: Post-Survey Q6

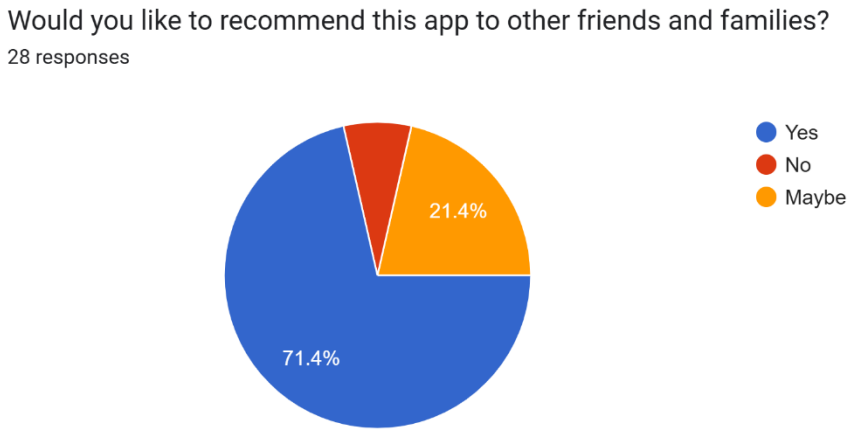


Figure 94: PosT-Survey Q7

8.3. Appendix C: Sample Codes

8.3.1. Sample Code of The UI

Login Page

```dart
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

```dart
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,
                  ),
```

```
                Row(
                  children: [
                    Checkbox(
                      value: agreeTerms,
                      onChanged: (val) => setState(() => agreeTerms = val!),
                    ),
                    Text(
                      "I agree to terms and conditions",
                      style: TextStyle(color: Colors.white),
                    ),
                  ],
                ),
                Button(
                  text: "Register",
                  color: Colors.white,
                  txtColor: Colors.black,
                  onPressed: register,
                ),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Text(
                      "Already have an account?",
                      style: TextStyle(color: Colors.white, fontSize: 14),
                    ),
                    TextButton(
                      onPressed: () => Navigator.pushNamed(context, '/login'),
                      child: const Text(
                        "Log In",
                        style: TextStyle(color: Colors.red, fontSize: 14),
                      ),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

Home Page

```dart
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(
```

```
                borderRadius: BorderRadius.circular(8),
              ),
            ),
            child: const Text(
              "Suggested",
              style: TextStyle(color: Colors.white, fontSize: 18),
            ),
          ),
        ),
      ),

      if (selectedTag.isNotEmpty)
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 8.0),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Text(
                "Filtered by tag: $selectedTag",
                style: const TextStyle(fontWeight: FontWeight.bold),
              ),
              TextButton.icon(
                onPressed: _clearFilter,
                icon: const Icon(Icons.clear),
                label: const Text("Clear Filter"),
              ),
            ],
          ),
        ),
      Expanded(
        child:
            isLoading
                ? const Center(child: CircularProgressIndicator())
                : ListView.builder(
                  itemCount: filteredRecipes.length,
                  itemBuilder:
                      (_, i) => RecipeTile(
                        recipe: filteredRecipes[i],
                        onTagSelected: _onTagSelected,
                        selectedTag: selectedTag,
                      ),
                ),
      ),
    ],
  ),
```

```
    );
  }
}
```

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

```dart
import 'package:flutter/material.dart';
import 'package:frontend/screens/Pantry/widgets/add_ingre.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) {
      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();
                                    },
                                  ),
                                );
                              }).toList(),
                    ),
                  ),
                ],
              )
              : 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

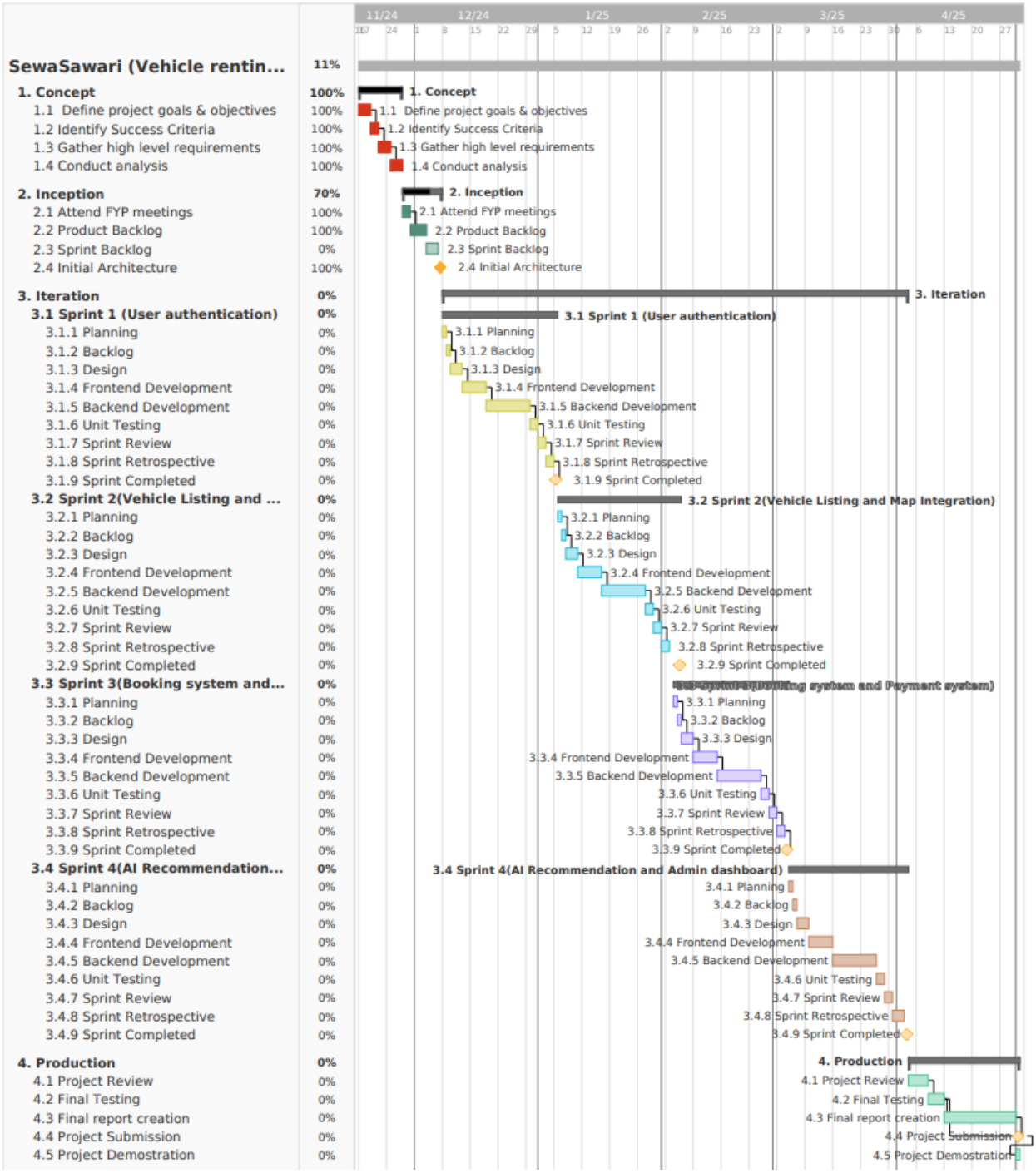


Figure 95: Gantt Chart

8.4.2. Work Breakdown Structure

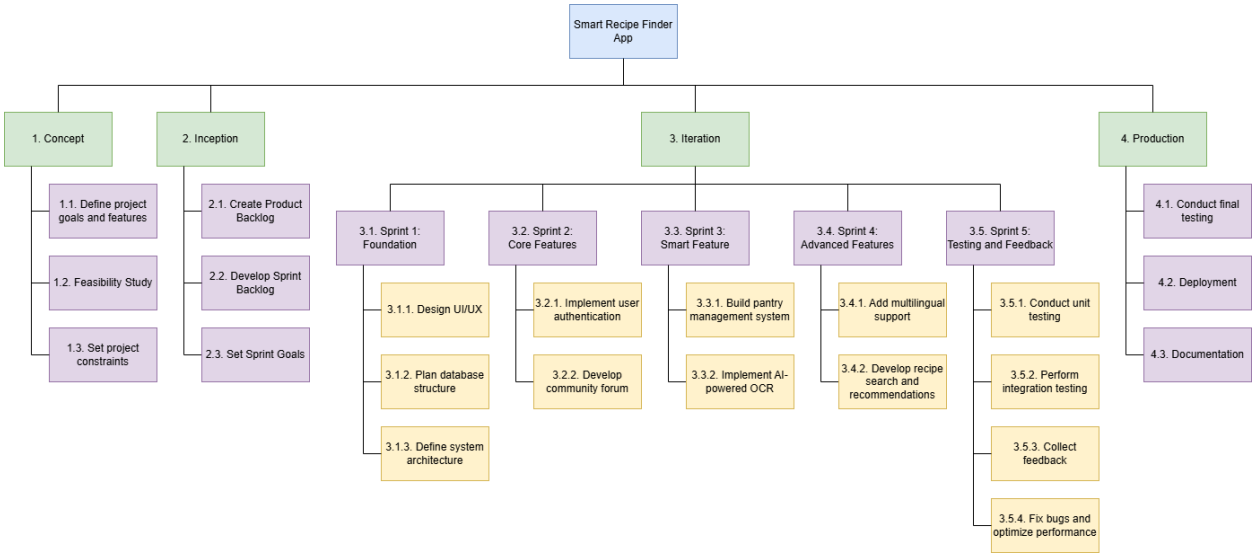


Figure 96: Work Break Structure

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

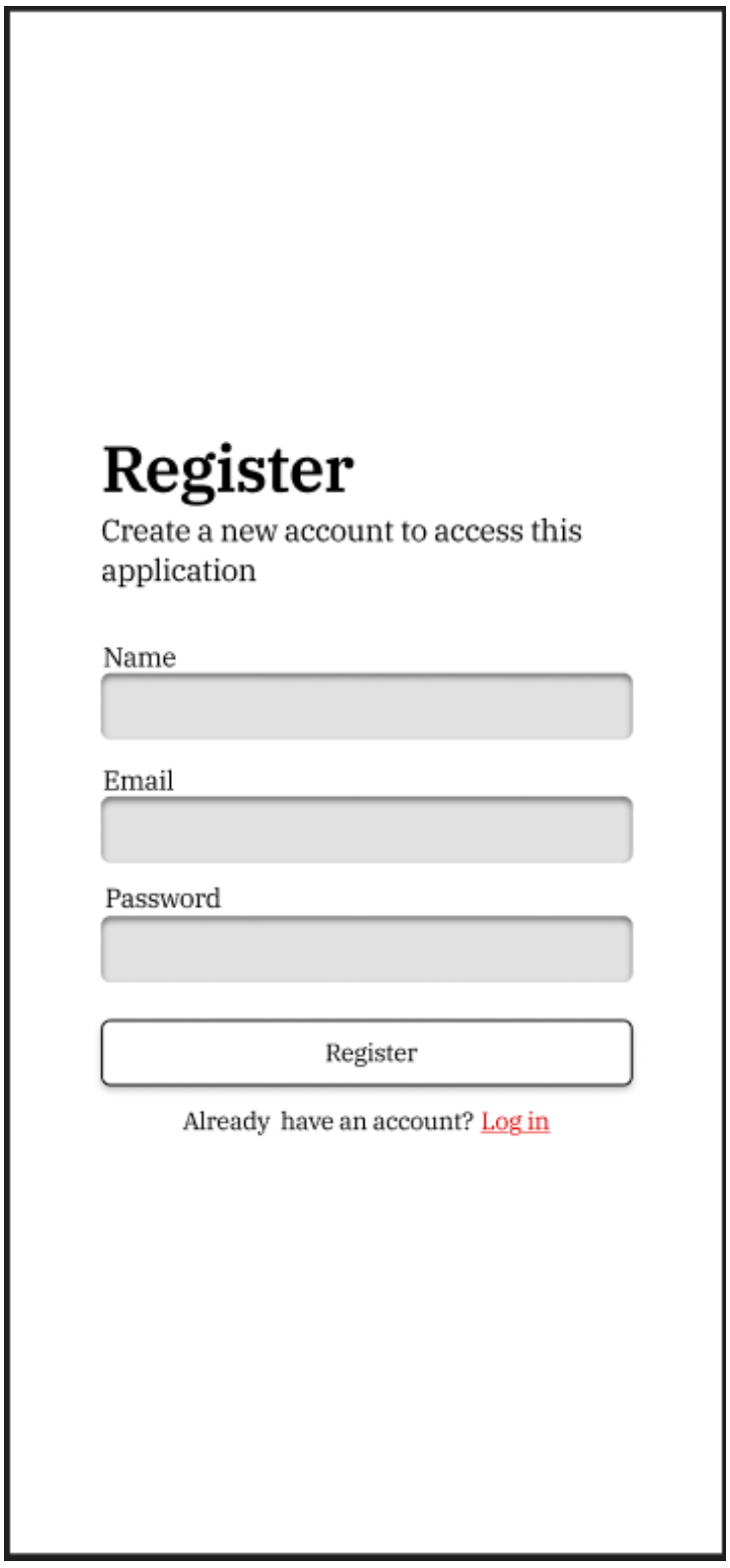


Figure 99: Register Page Wireframe

Figure 100: Login Page Wireframe

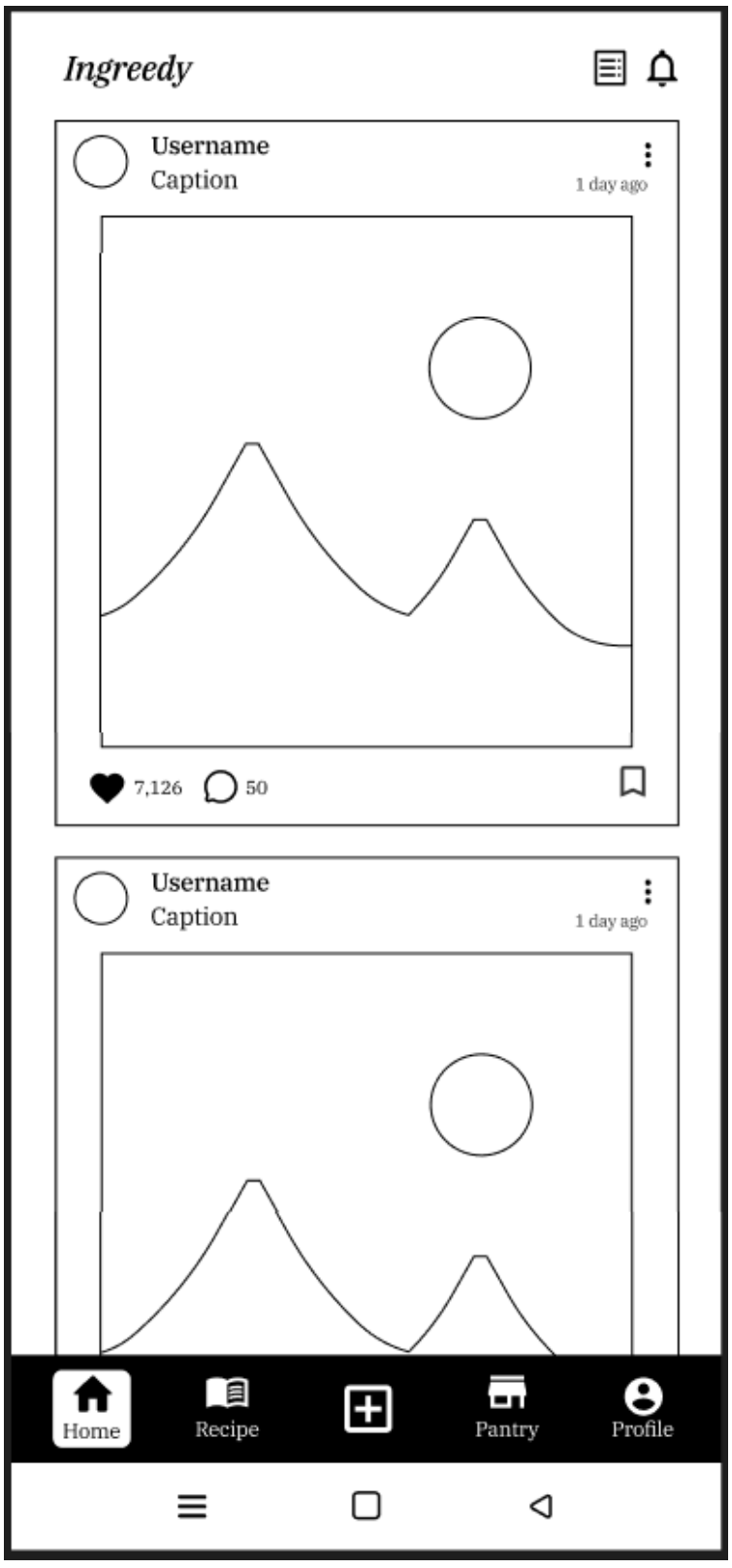


Figure 101: Home Page Wireframe

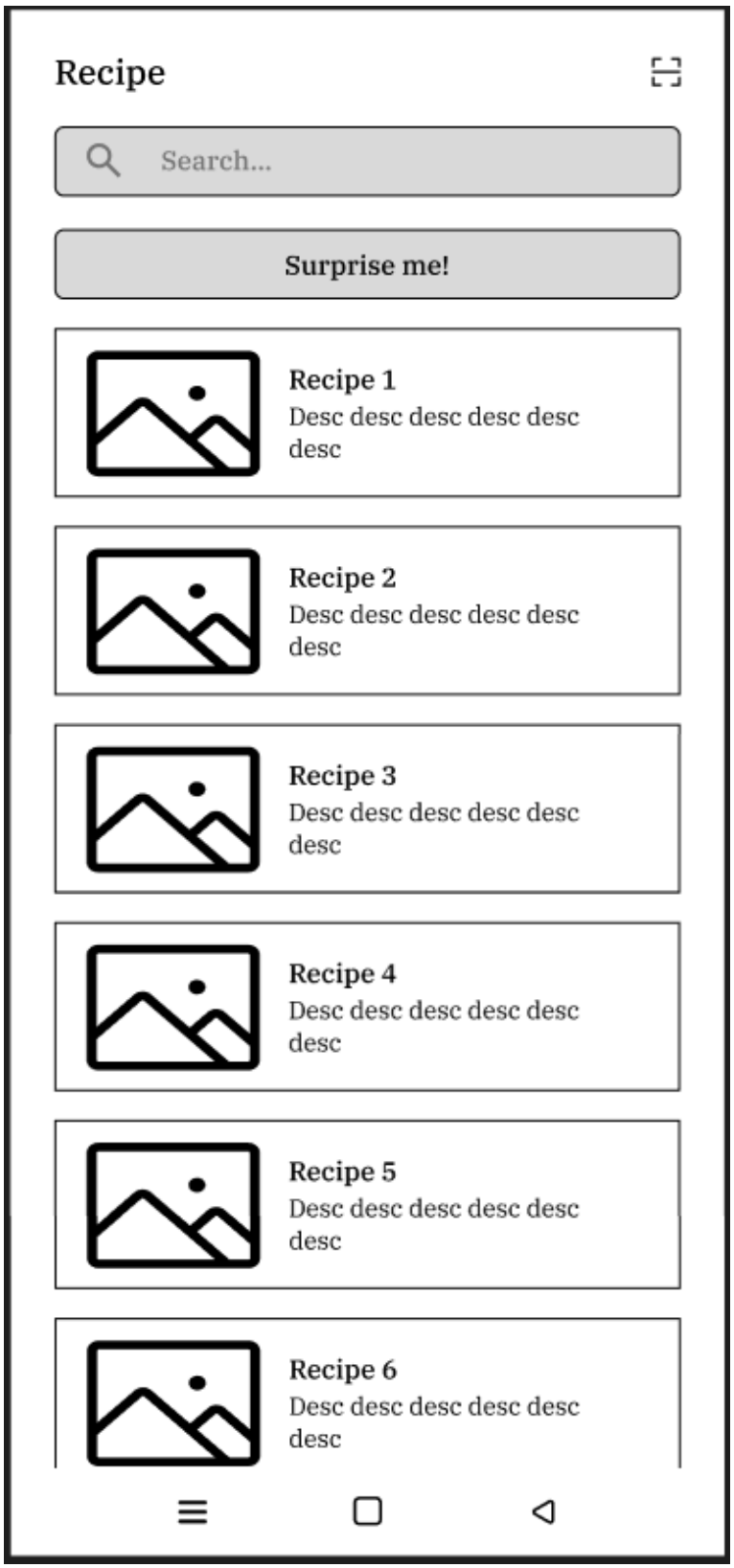


Figure 102: Recipe Page Wireframe

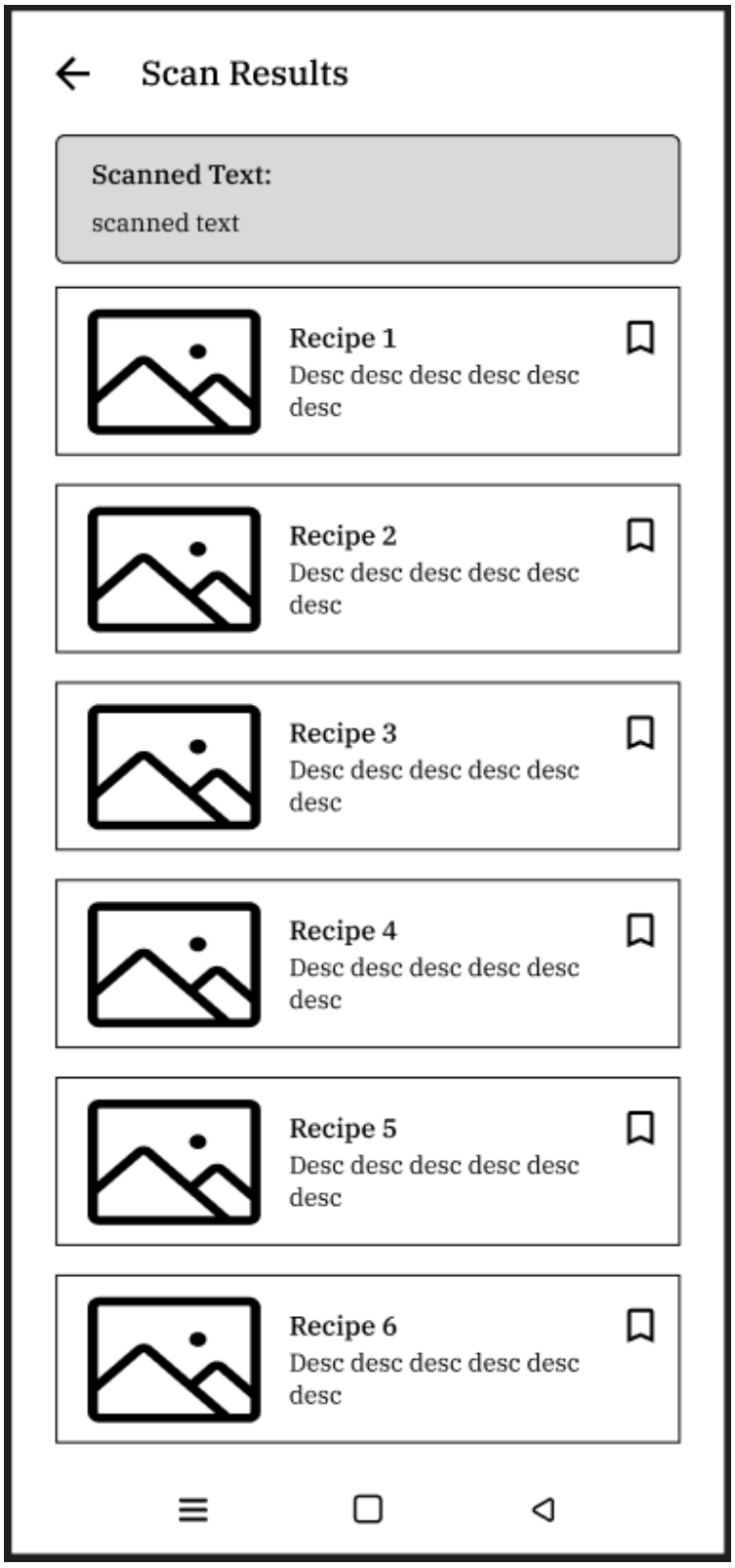


Figure 103: Scan result wireframe

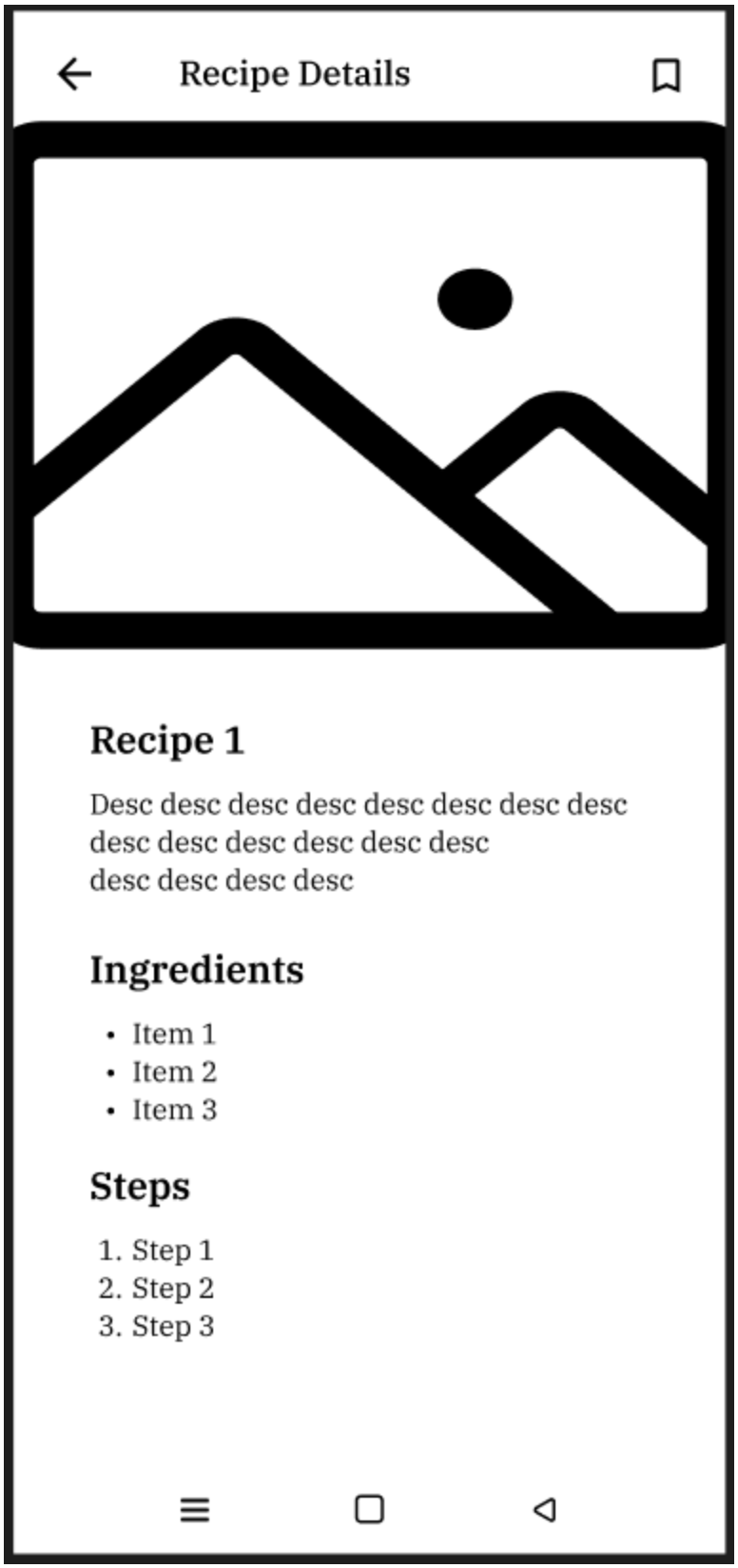


Figure 104: Recipe details wireframe

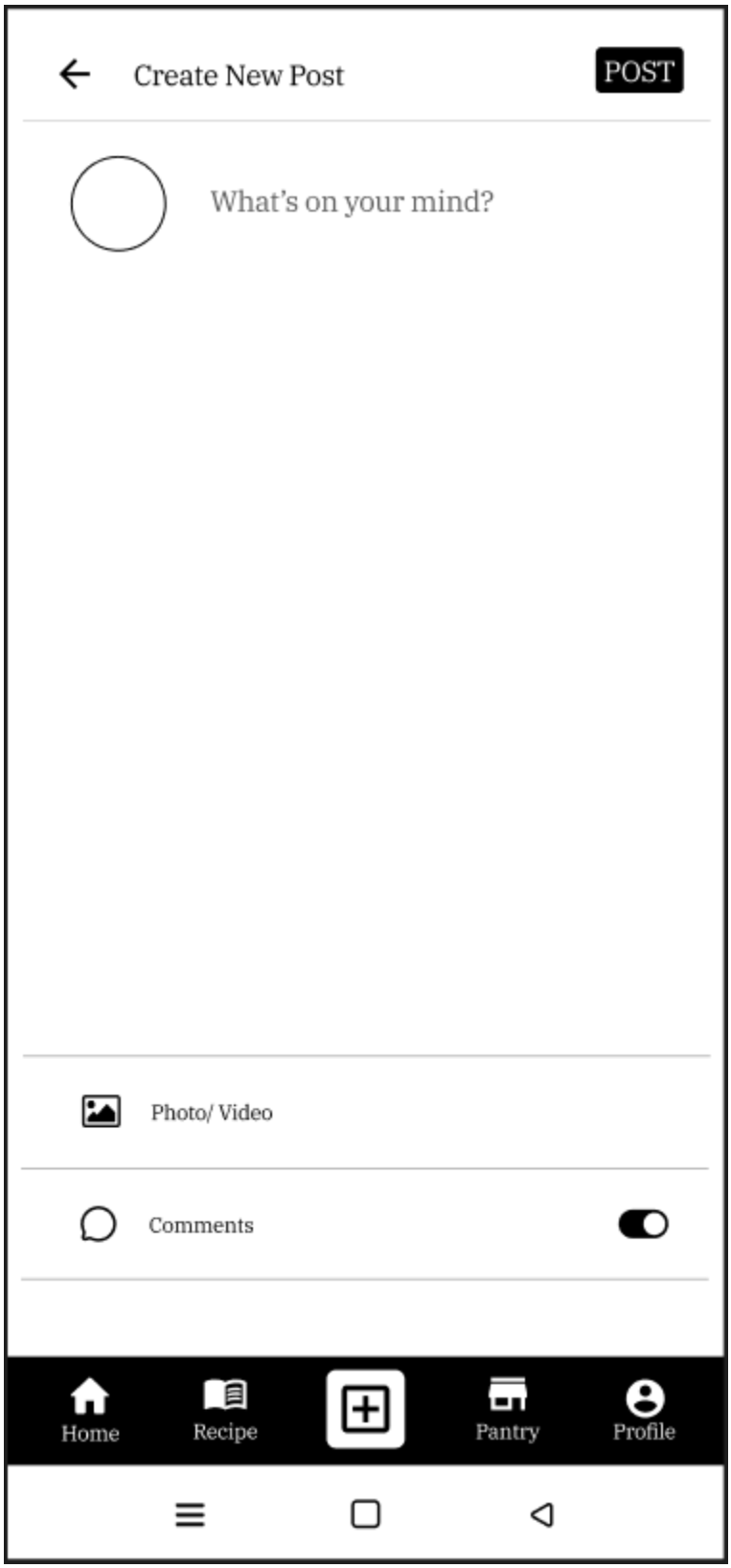


Figure 105: Create Post Wireframe

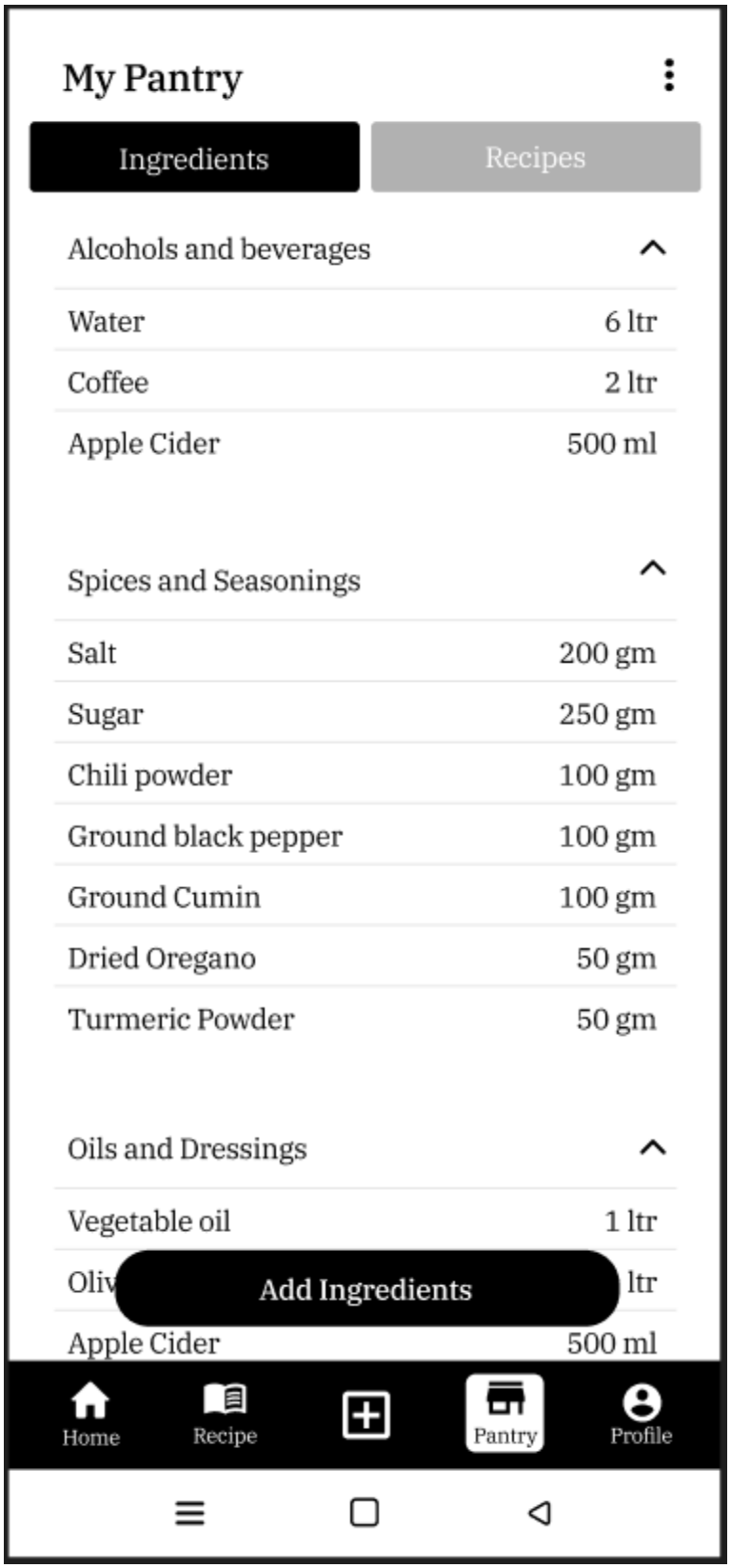


Figure 106: Pantry Page Wireframe

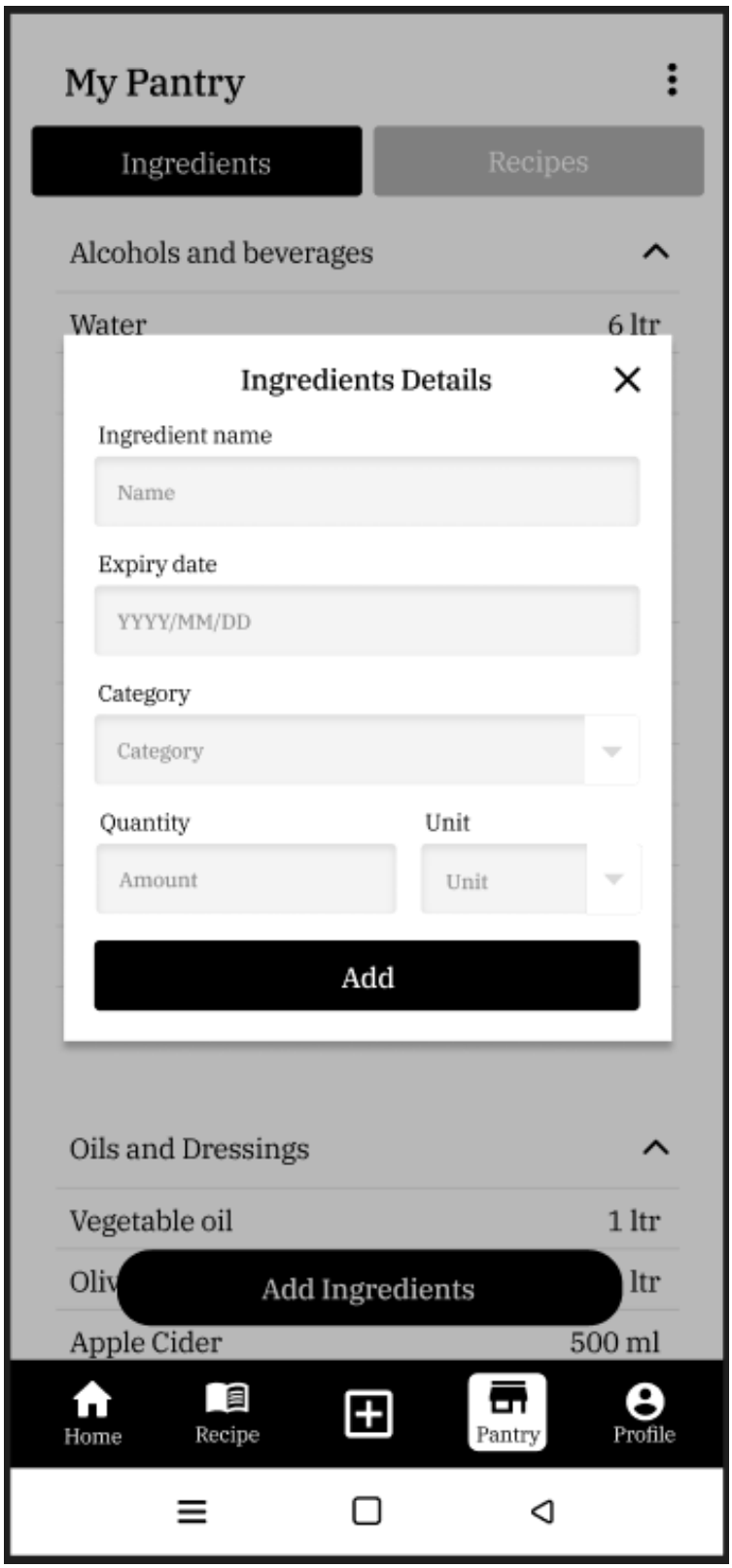


Figure 107: Add ingredient wireframe

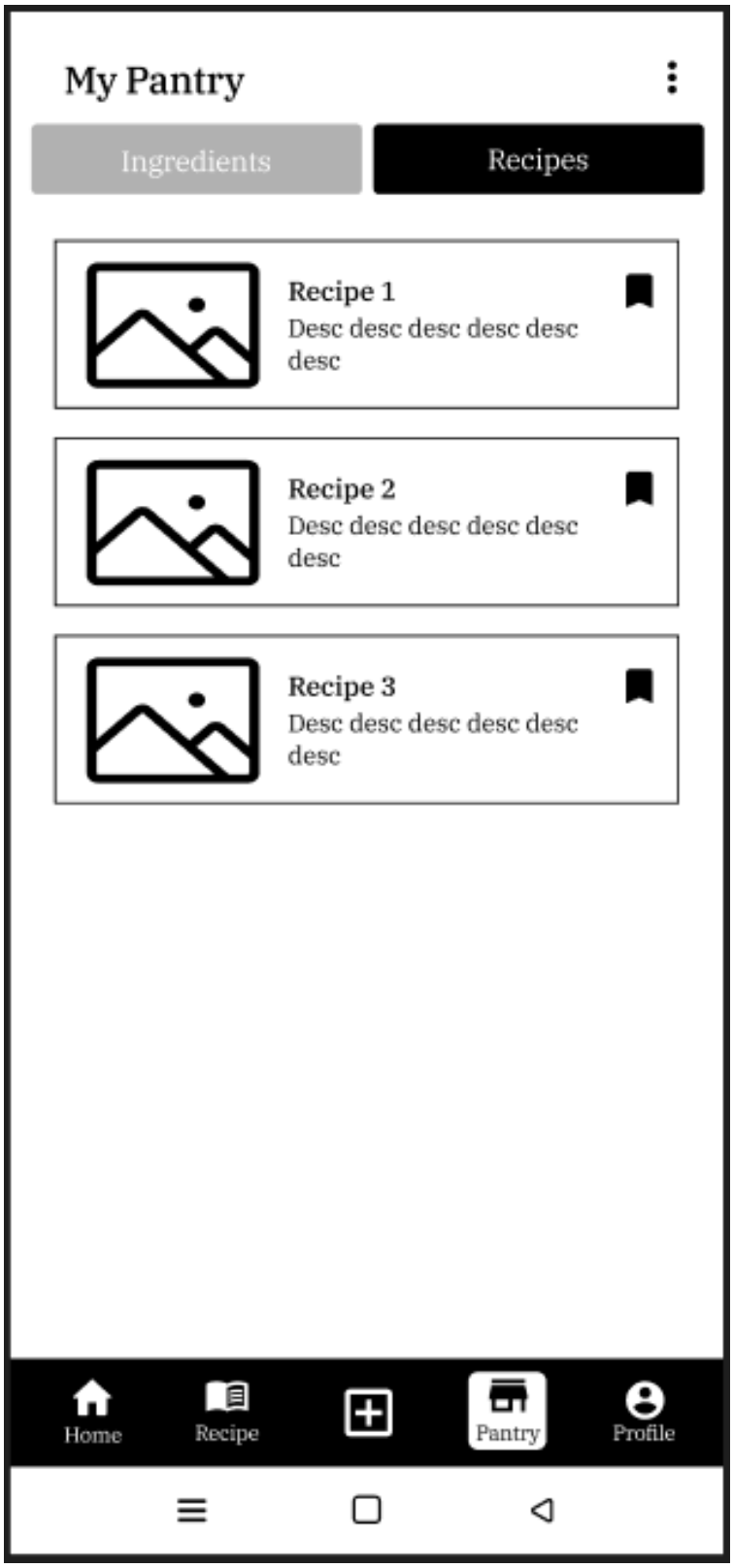


Figure 108: Saved recipes screen wireframe

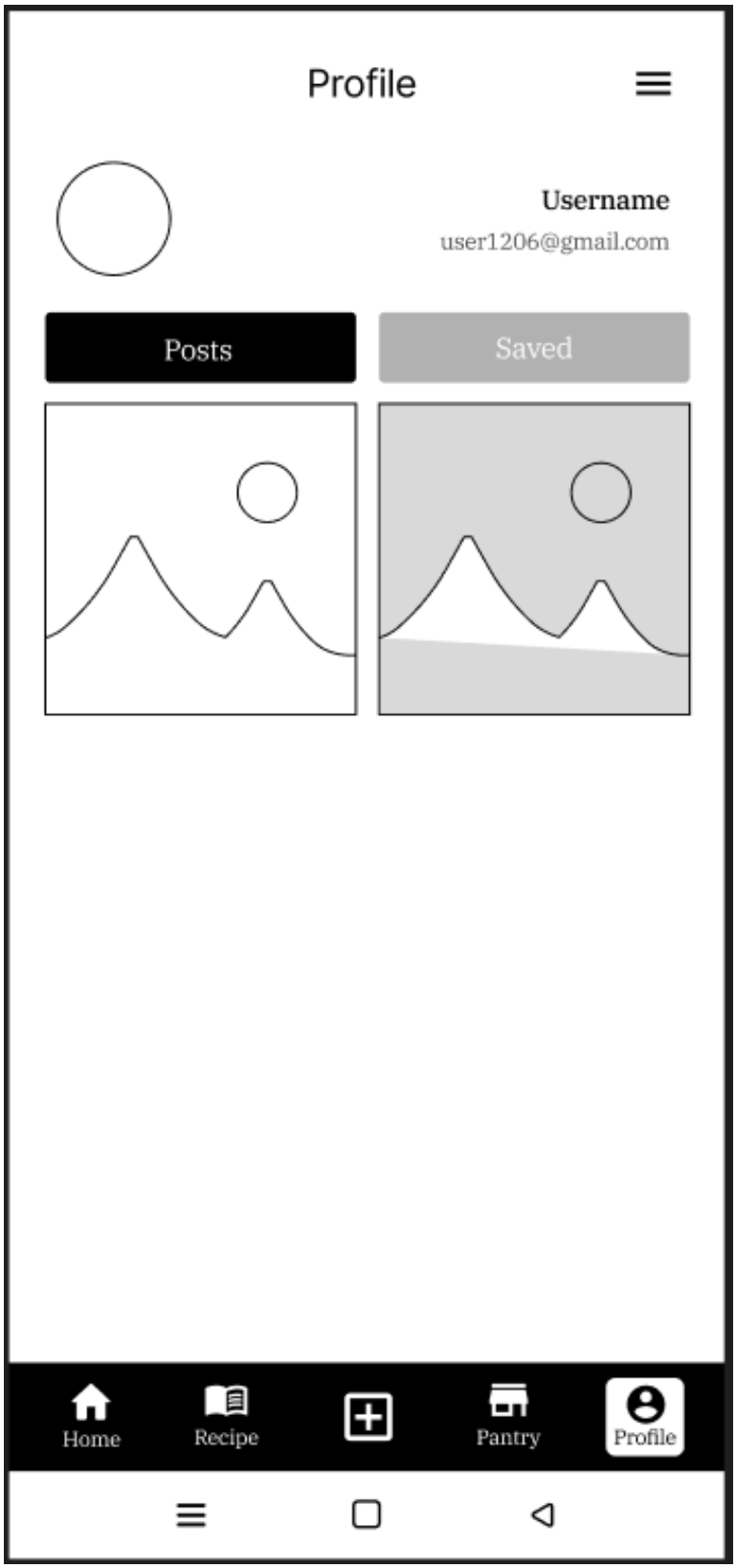


Figure 109: Profile Page Wireframe

8.5. Appendix E: Screenshots of The System

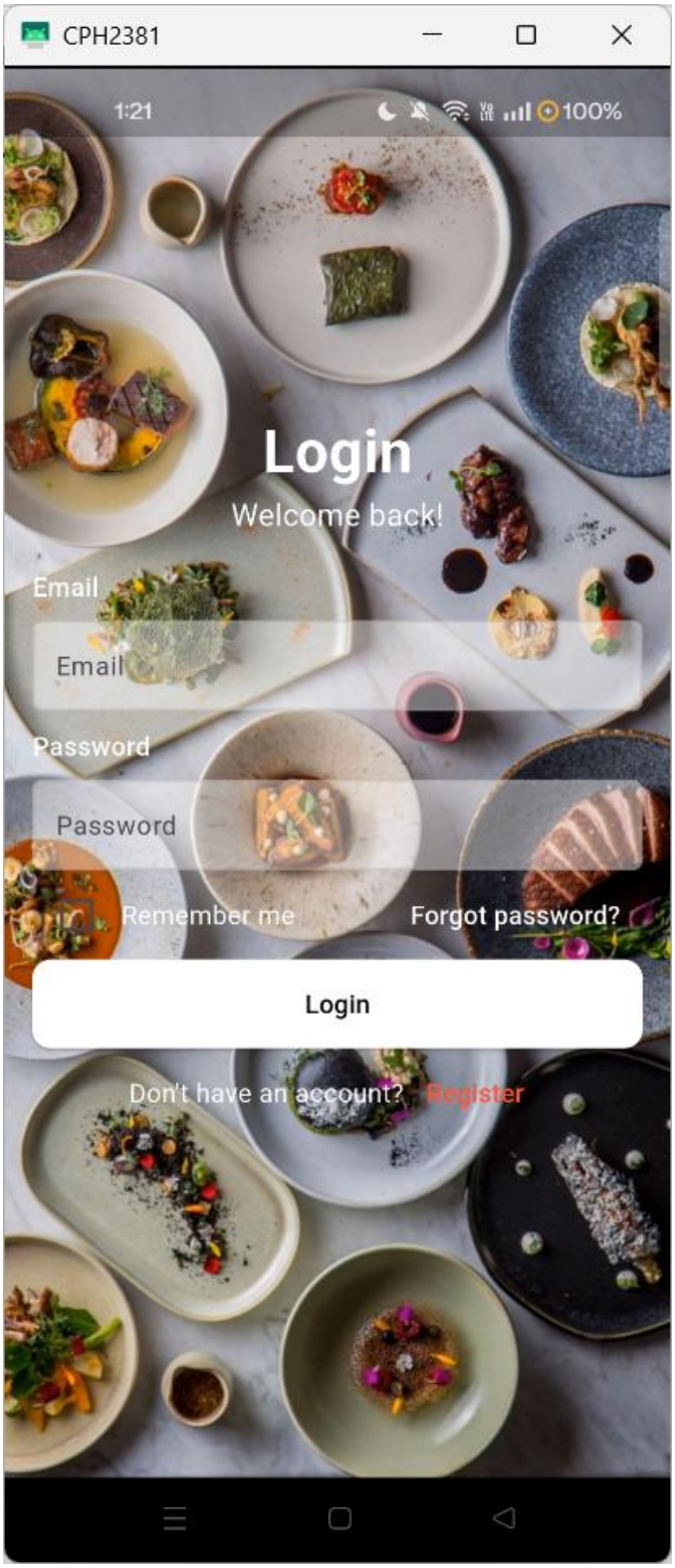


Figure 110: Login page screenshot

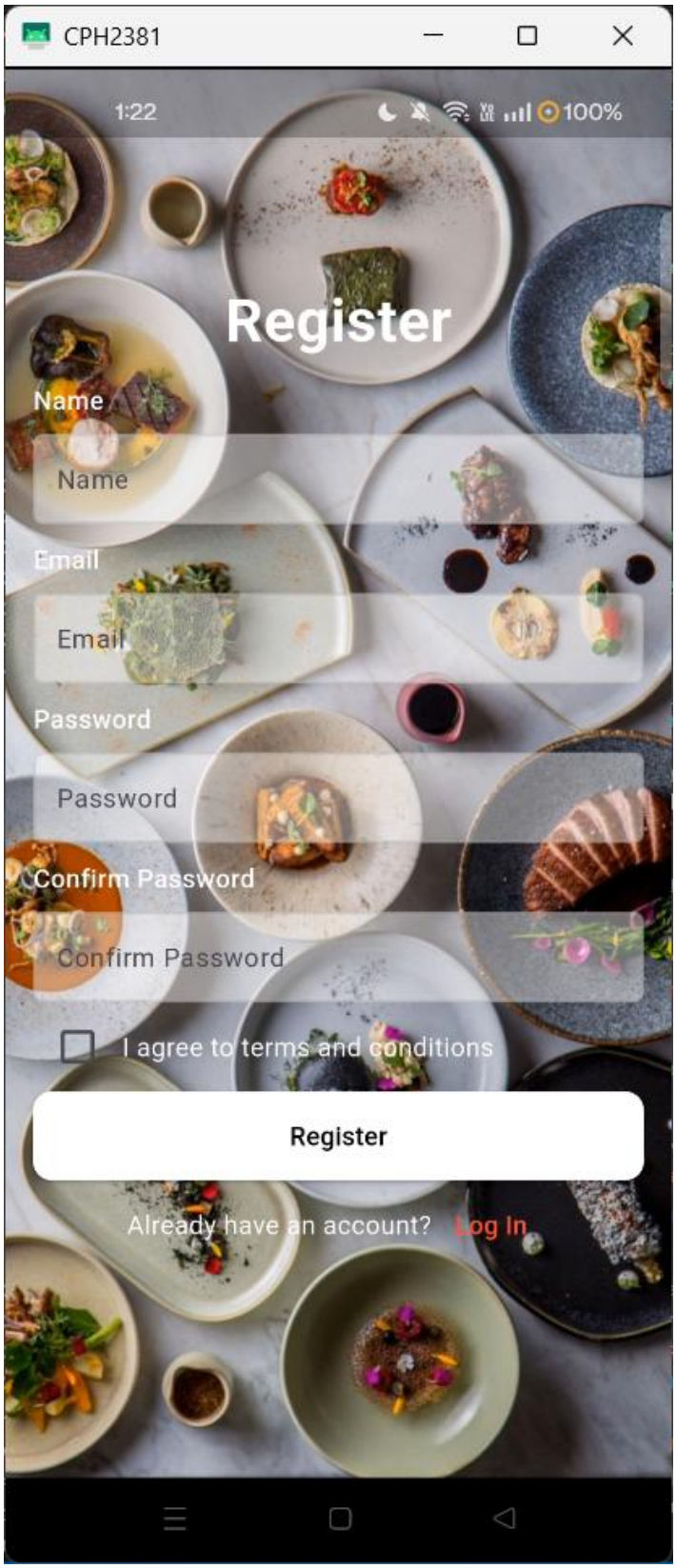


Figure 111: Register page screenshot

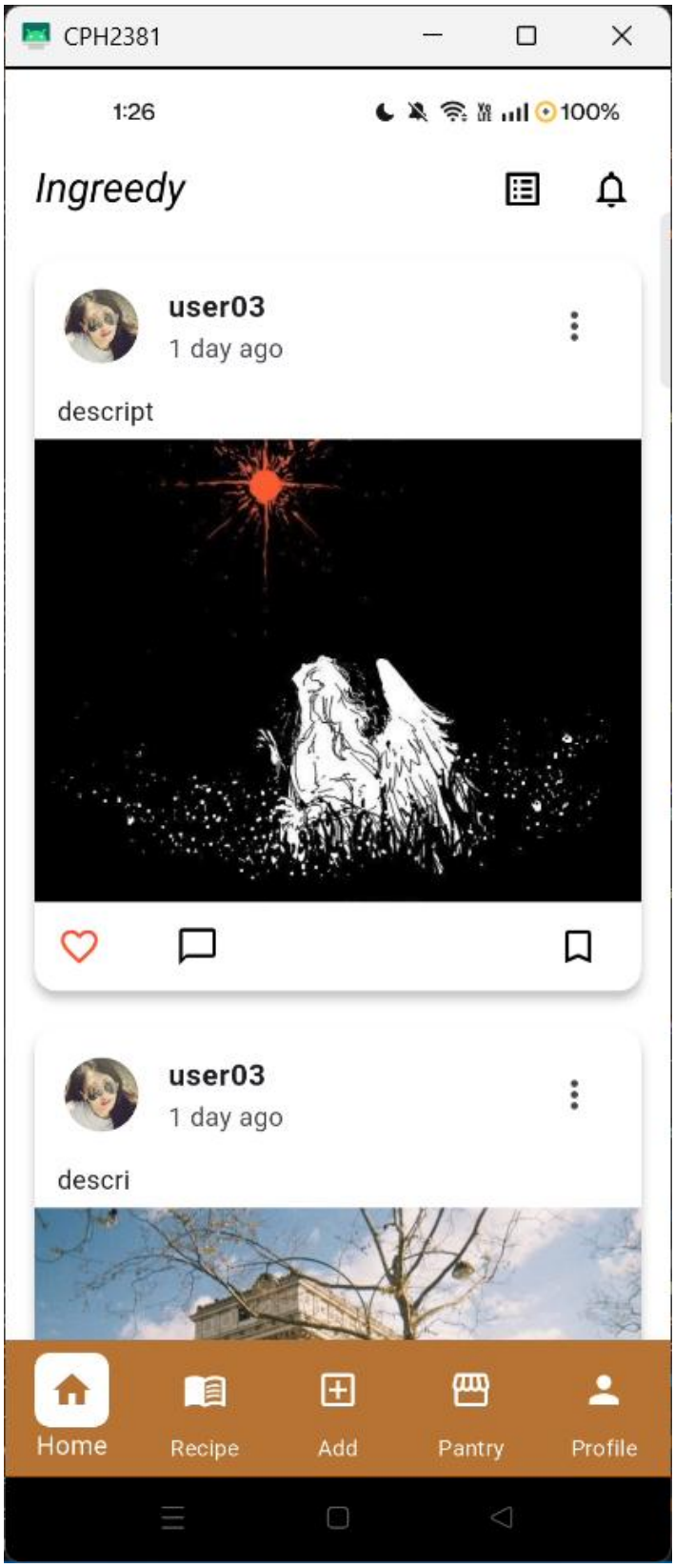


Figure 112: Home page screenshot

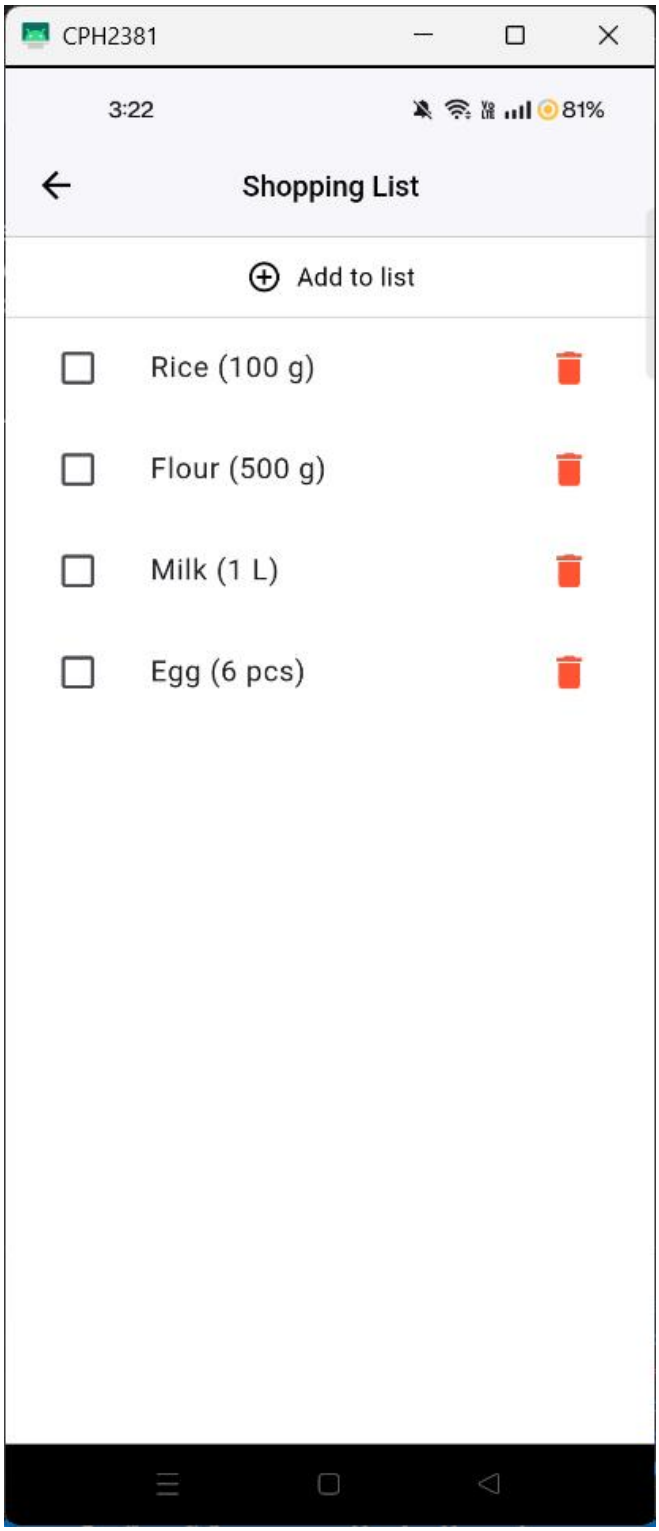


Figure 113: List page screenshot

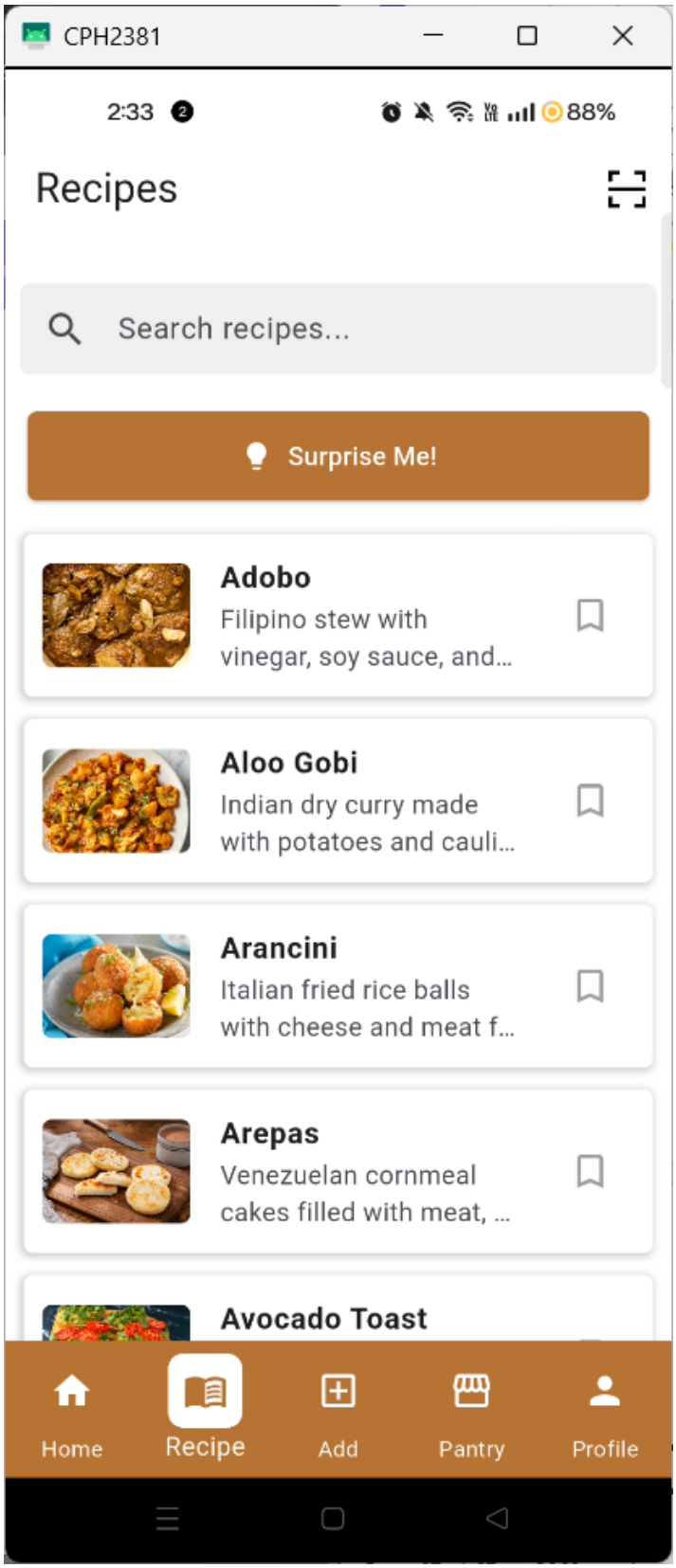


Figure 114: Recipes page screenshot

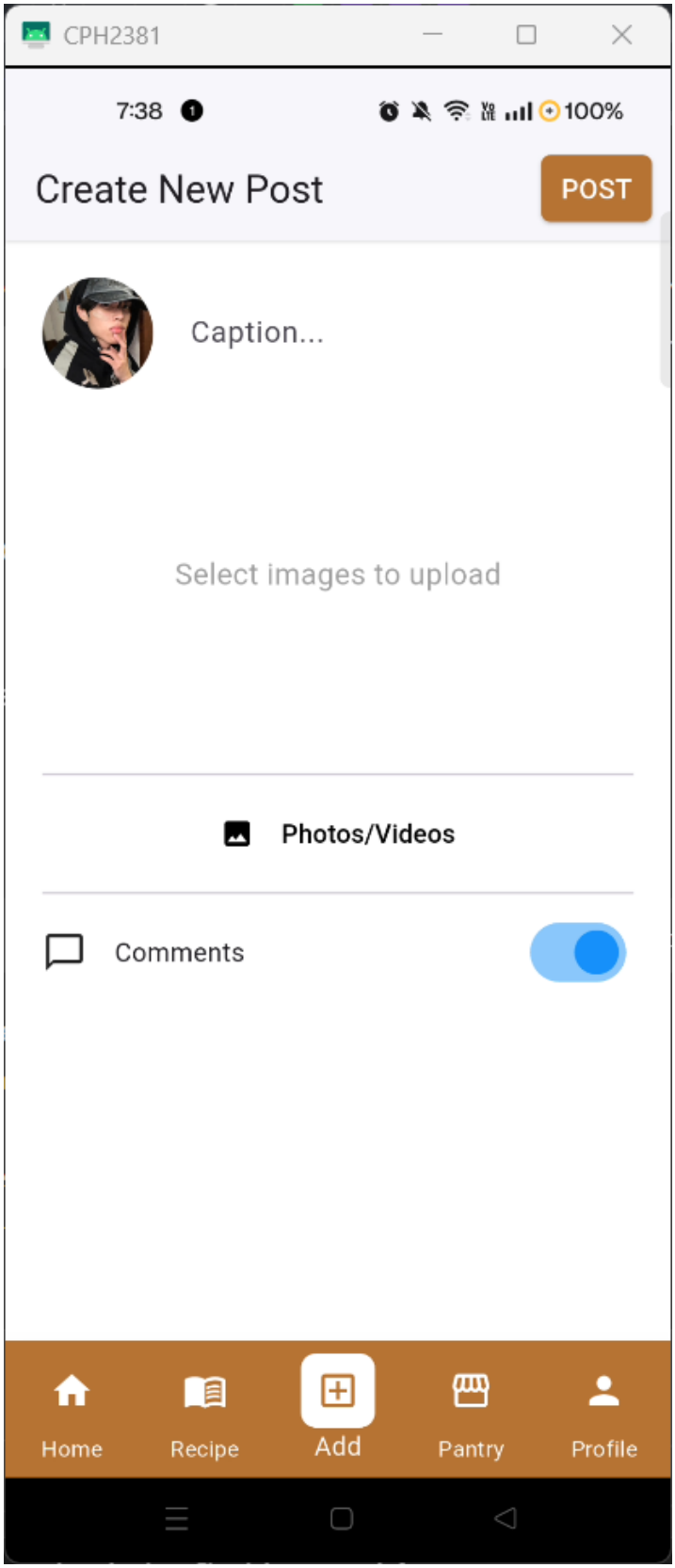


Figure 115: Create Post page screenshot

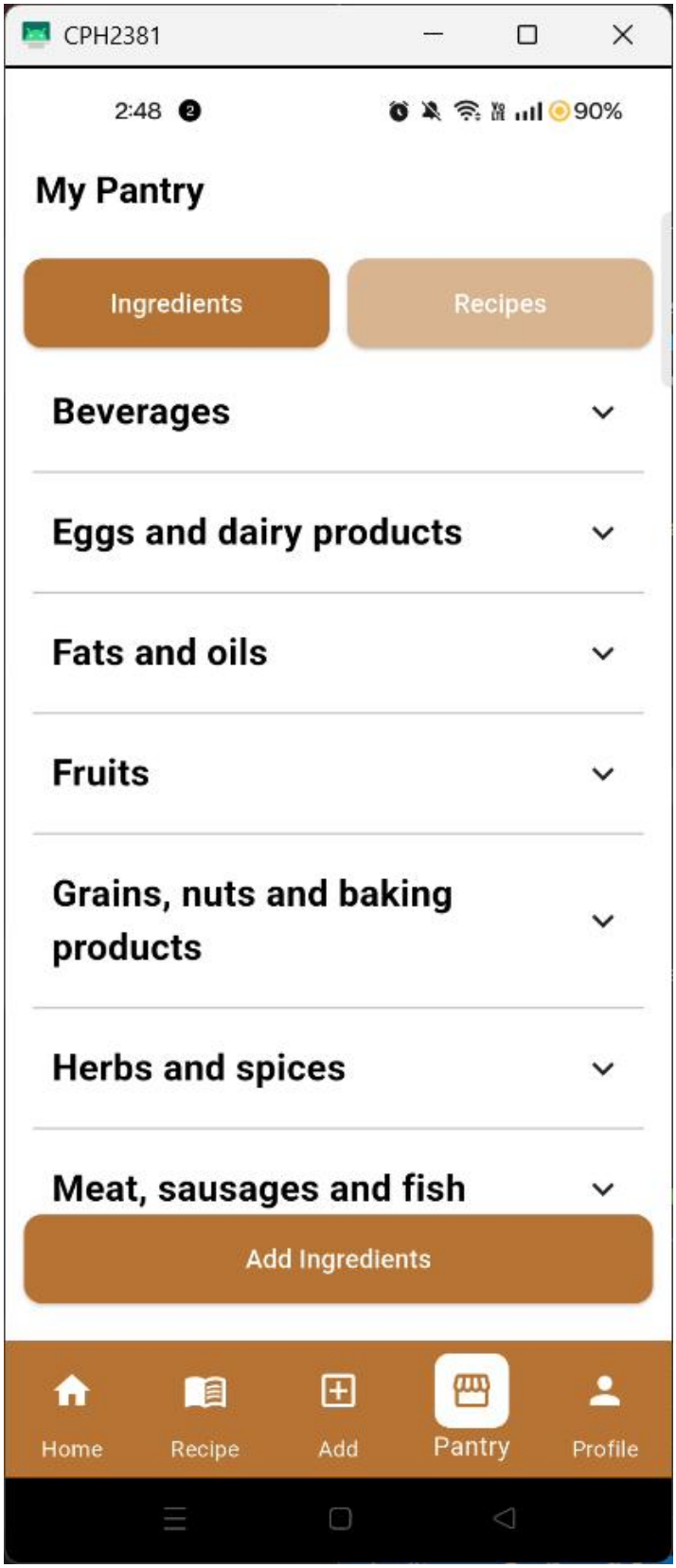


Figure 116: Pantry page screenshot

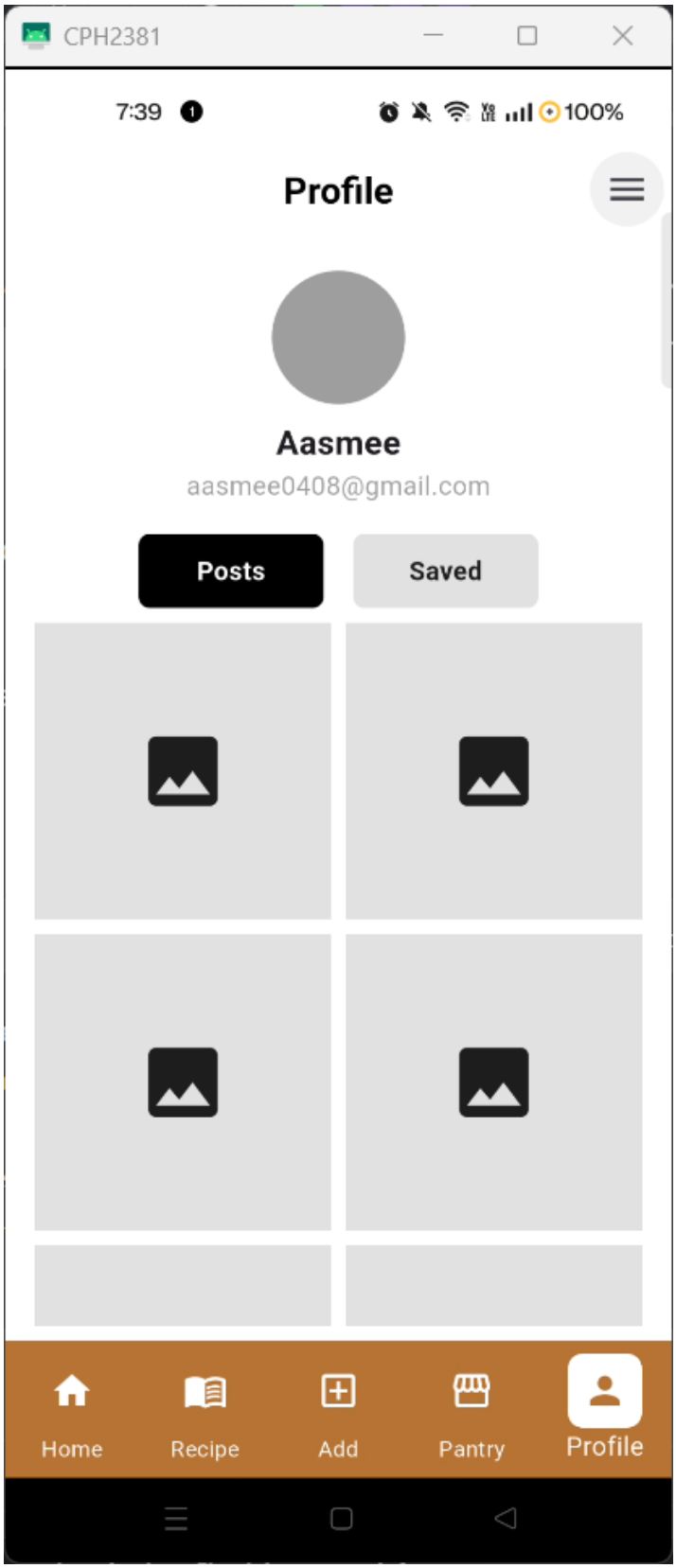


Figure 117: Profile page screenshot