# Software Architecture and Design Document

**for**

**FoWRA**

**Prepared by**


| Name | Matric Number |
|---|---|
| AMYLIA NATASYA BINTI MOHD FAIZAL | 214887 |
| NURIN ADNI BINTI MOHD ZARONIZAM | 214691 |
| NUR AKMAL BIN ABDUL HAMDI | 214399 |
| MUHAMMAD ARIQ ULWAN BIN ANUAR | 215746 |
| SAFAN MUHAMMAD JAY | 213258 |


**Instructor : Prof. Madya Dr. Sa'adah Binti Hassan**

**Course : SSE3302-1 : Software Architecture and Design**

**Date : 18 January 2024**

# Table of Contents

# 1.0 Introduction

## 1.1 Purpose

The primary objective of the document is to serve as an in-depth guide reflecting every aspect of the design considerations, detailed design, and architecture that drive FoWRA . As a result of this, it aims to meet the needs of the development team as well as other stakeholders and other entities involved in the complex process of creating, designing, and executing the FoWRA product. It aims to ensure that everyone has a clear understanding of the design principles, objectives, and methodologies to be employed in the development of FoWRA.

## 1.2 Intended Audience & Reading Suggestions

- Development Team: Delve into the technical intricacies, algorithms powering waste reduction, and the seamless integration of donation features.
- Product Managers: Explore features that enhance user experience and align with sustainability and donation objectives.
- Stakeholders: Understand the potential societal impact of FoWRa in reducing food waste and fostering community engagement.
- End Users: Discover the user-centric features, benefits, and guidelines to embrace a sustainable lifestyle through waste reduction and donation facilitation

## 1.3 Product Scope

Food waste is recognized to be a huge problem worldwide and is particularly severe in developed countries. Therefore this inspires us to create a software product called FoWRA: Food Waste Reduction App that is almost similar like other food waste app such as OLIO and Kitche but with additional features.This app have inventory management features where user can track the food items they have at home and receive notifications about items that are approaching their expiration date.The app also can suggest recipes based on the ingredients that the user have in their inventory to help them use up their food before it goes bad. It can scan food products barcodes and use the information to alert user about the expiration date, so they can plan to use or donate the food item before it goes bad.Tips and tricks for reducing food waste, such as proper storage techniques, meal planning, and composting also provided by the app.User can connect with other app user in their community to share tips, recipes, and ideas for reducing food waste.

FoWRA aims to help user reduce the amount of food they waste.This helps the user to save money and encourage a more sustainable lifestyle.User can make more informed choices about their food consumption which will educate them about the environmental impact of food waste. This will undoubtedly improve food safety which helps to prevent foodborne illnesses.

## 1.4 Product Perspective

The FoWRa food waste app is designed to simplify food management processes and provide an intuitive application for users looking to contribute to reducing food waste. This application is built on a relational database, featuring functions for efficient food management

and reservations. Our primary goal is to deliver a user-friendly experience, coupled with fair and competitive practices for food donation and waste reduction efforts.

More specifically, the system is tailored to enable seamless coordination among users, including contributors, reviewers, and administrators, in the process of reducing food waste and sharing valuable insights with the community. The application fosters communication through integrated messaging features, ensuring efficient interactions via the platform. The app offers configurable settings for easy access. Additionally, FoWRA makes use of an effective relational database to store essential information on users, food products, contributions, and donations. It enables for the creation of a full system for the reduction of food waste and engagement with the community.

## 1.5 Definition, Acronyms & Abbreviations

Definitions of all terms, acronyms and abbreviation used are to be defined here.

| Term | Description |
|---|---|
| Application | A software program that runs on your computer |
| Food Waste | Any edible food that is discarded or goes uneaten, including both perishable and non-perishable |
| Expiration Date | The date specified on a food product by manufacturer or producer indicating the recommended last day of consumption for optimal quality and safety |
| Inventory Management | The practice of tracking and controlling the quantity and flow of food items in stock, including purchasing, storage, and monitoring for waste reduction |
| Recipe Suggestions | Recommendations or ideas provided by the app to utilise leftover ingredients or items nearing their expiration to minimise food waste. |
| Food Tracking | The act of recording or logging food purchases, consumption and waste within the app to gain insights and make more informed decisions |
| Push Notifications | Messages or alerts sent by the app to users' devices to provide reminders, updates, or personalised recommendations. |

| Acronym | Description |
|---|---|
| SRS | Software Requirement Specification |
| SQL | Structured Query Language |
| SSL | Secure Socket Layer |
| TLS | Transport Layer Security |
| MQTT | Message Queuing Telemetry Transport |
| FoWRA | Food Waste Reduction App |
| MTTR | Mean Time To Repair |
| MTBF | Mean Time Between Failure |
| GDPR | General Data Protection Regulation |
| SMS | Short Message Service |

## 1.6 References

1. Bennett, S., Mcrobb, S., & Farmer, R. (2014). *Object-oriented systems analysis and design : using UML*. Mtm.

2. Bass, L., Clements, P., & Kazman, R. (2013). *Software Architecture in Practice* (3rd ed.). Addison-Wesley.

3. "IEEE Recommended Practice for Software Design Descriptions," in IEEE Std 1016-1998 , vol., no., pp.1-23, 4 Dec. 1998, doi: 10.1109/IEEESTD.1998.88828

4. GeeksforGeeks. (2024, January 5). Package diagram: Introduction, elements, use cases and benefits. GeeksforGeeks. https://www.geeksforgeeks.org/package-diagram-introduction-elements-use-cases-and-benefits

5. Athuraliya, A. (2023, January 5). Component diagram tutorial: Complete guide with examples. Creately Blog. https://creately.com/blog/software-teams/component-diagram-tutorial/

6. Athuraliya, A. (2023, January 5). Component diagram tutorial: Complete guide with examples. Creately Blog. https://creately.com/blog/software-teams/component-diagram-tutorial/

7. S. J. Lincke, T. H. Knautz and M. D. Lowery, "Designing System Security with UML Misuse Deployment Diagrams," 2012 IEEE Sixth International Conference on Software Security and Reliability Companion, Gaithersburg, MD, USA, 2012, pp. 57-61, doi: 10.1109/SERE-C.2012.12.

8. Stangherlin, I. D. C., & De Barcellos, M. D. (2018). Drivers and barriers to food waste reduction. *British Food Journal*, *120*(10), 2364-2387.

# 2.0 Design Consideration

## 2.1 Operating Environment

- Android

## 2.2 Design & Implementation Constraints

The design and implementation constraints for FoWRA application is as listed as below :

**DI1: Operating System Compatibility:**
The app must be compatible with various versions of Android operating systems to cater to a broad user base.

**DI2: Screen Resolutions:**
The app's user interface (UI) design should be responsive and adaptable to different screen sizes and resolutions on both iOS and Android devices, ensuring a seamless user experience.

**DI3: User Authentication:**
Implement secure user authentication mechanisms to ensure that only authorised users can access FoWRA features. User authentication should be robust and may include email verification.

**DI4: Data Protection:**
Ensure that user data, including inventory details and personal information, is stored securely. Comply with relevant data protection regulations, such as GDPR to safeguard user privacy.

**DI5: Secure Communication:**
The app should use secure communication protocols, such as HTTPS, to encrypt data transmission between the app and the server. This helps prevent unauthorised access and ensures data integrity.

**DI6: Responsiveness:**
The app provides users with a smooth and efficient experience. Minimise loading times and delays when retrieving data, suggesting recipes, or performing other actions within the app.

**DI7: Existing Systems Integration:**
The app needs to integrate with existing systems, such as barcode databases for product information or community forums for user interactions. Proper coordination and data exchange protocols should be established for seamless integration.

**DI8: API Integration:**
The app integrates with external services or APIs as needed. This could include barcode scanning APIs for product information or geolocation services for community features. Adhere to the respective API documentation and standards to ensure compatibility and reliability.

# 3.0 Detailed Design

## 3.1 Class and Object Design

For our class and object design, we took back the class diagram from the requirement specifications and we added some improvement in terms of details and design for the class which is the attributes, operations, interfaces, associations and integrity constraints on the design.

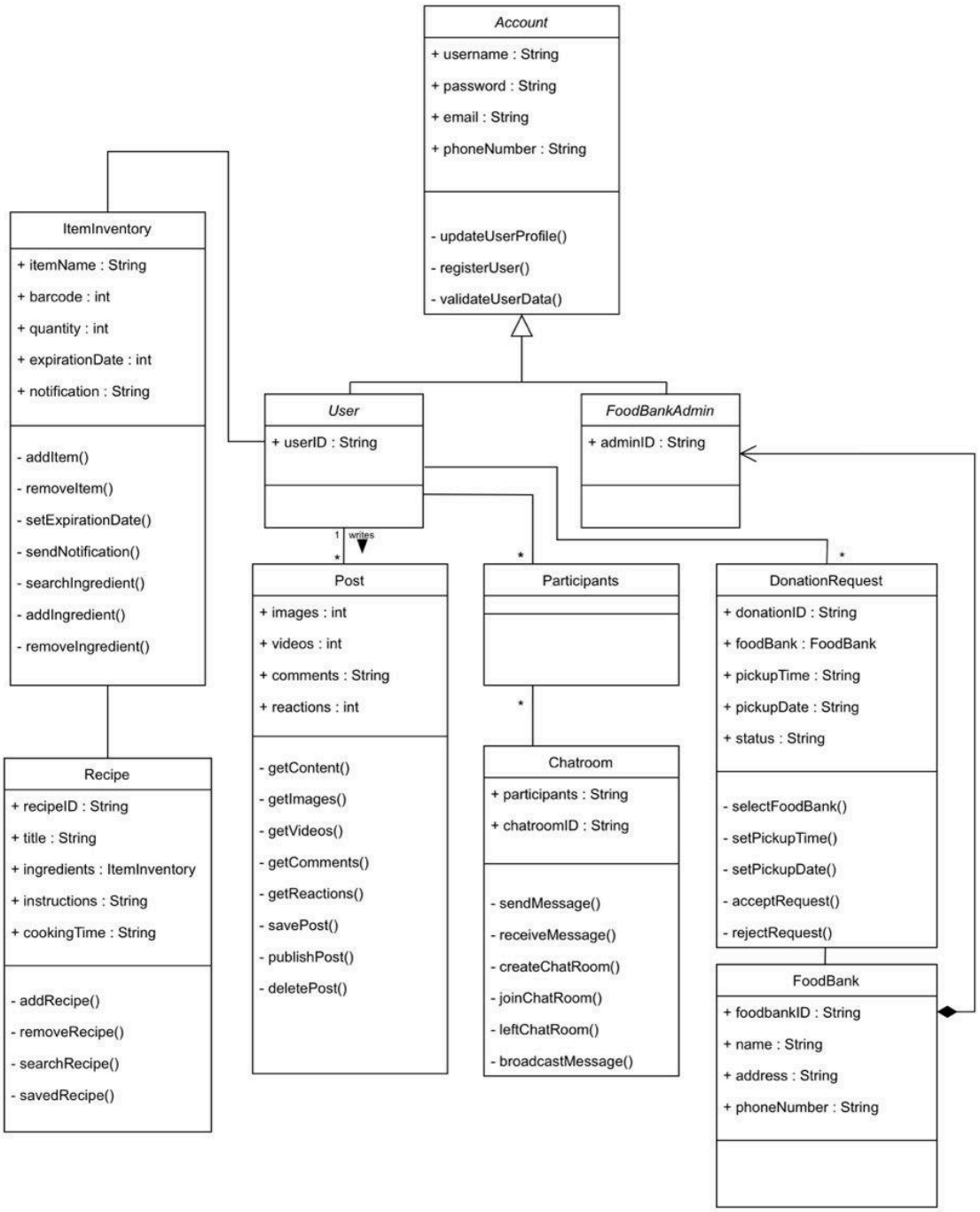



Figure 3.1.1 Class Diagram

**Account Class**

***Classification:***
Class

***Definition:***
Account class is used to encapsulate the essential data and behaviour associated with a user or admin account providing a structured way to manage user information, authentication, authorization, and related operations.

***Responsibilities:***
Data Storage:

Account class acts as the data storage to store common account information which specifically are username, password, email, and phone number.

Authentication:

Account class is used to verify user credentials which is to check the validity of provided usernames and passwords during login attempts including comparing entered credentials with stored data to ensure authenticity

Authorization:

Account class is used for managing account permissions, determining which actions a user can perform based on their assigned role.This enforces access control rules to protect sensitive information and functionality.

Account Management:

Account class contains the method to register new users, update user profile, and validate user data as the main class for account.

***Constraints:***
Unique Usernames and Emails - Each account must have distinct username to identify users correctly and unique email address for reliable communication and account recovery

Secure authentication - Implement robust security measures for login and password reset to protect accounts from unauthorised access.

Compatible Inheritance - Ensure subclasses like User and FoodbankAdmin inherit attributes and methods correctly while maintaining compatibility with the parent Account class.

**User Class**

***Classification:***
Class

***Definition:***
User class represents a specific type of account within the system, designed to manage the data and actions associated with individual users who are not food bank administrators. It

inherits common attributes and methods from the parent Account class while introducing additional features tailored to the needs of general users.

***Responsibilities:***

Personalising User Experience:

User accounts are used to tailoring content and features based on individual preferences and needs.

Managing Food Inventory:

User accounts allow users to add, remove, and manage food items in a personal inventory ,tracking expiration dates to prevent food waste, receive notifications about expiring items and searching for recipes based on available ingredients

Donation Requests:

User account used to request donations of excess food items to food banks and coordinating pickup time and dates

Community Engagement

User accounts used to interact with other users through chat rooms and posts to share experiences, recipes and tips.

***Constraints:***

Unique Constraints - Each account must have distinct username to identify users correctly and unique email address for reliable communication and account recovery

Security - Sensitive information like passwords must be stored securely using hashing and encryption techniques.Privacy settings for user data must be respected and enforced.

Authorization - Users should only be able to access and modify their own data and perform actions permitted by their role.

Validation - Food inventory items should have valid expiration dates and quantities. Donation requests must adhere to food bank policies and logistics constraints.

**Food Bank Admin Class**

***Classification:***

Class

***Definition:***

The FoodbankAdmin class is designed to manage the functionalities and data associated with foodbank administrators. It inherits common attributes and methods from the parent account class while introducing additional features tailored to the needs of foodbank management.

***Responsibilities:***

Managing Food Bank Inventory:

Foodbank admin can Add, remove, and track food items within the associated foodbank.

Reviewing Donation Requests:

Foodbank admin will review user requests for food donations and make approval decisions.

Scheduling Pickups:

Foodbank admin can coordinate the logistics of collecting donated food from users.

Communicating with Users:

Foodbank admin can provide updates, instructions, and information to users regarding food donations and services.

***Constraints:***

Authorization - Access control mechanisms, restrict specific features and data based on the FoodbankAdmin's permission level.

Foodbank Association: Each FoodbankAdmin is related to a specific foodbank and cannot manage inventory or resources of other food banks.

Data Security: Sensitive information like food bank locations and donor details require appropriate security measures.

**Donation Request Class**

***Classification:***

Class

***Definition:***

The DonationRequest class represents a user's request to donate food items to a foodbank within the application. It captures the essential information about the donation and facilitates the communication and logistics involved in the process.

***Responsibilities:***

Capturing Donation Information:

Donation Request collects details about the offered food items, preferred pickup date/time, and any additional comments.

Facilitating Communication:

Donation Request provides a channel for communication between the user and the foodbank admin regarding the request status and logistics.

Managing Request Status:

Donation Request tracks the progress of the request through different stages (pending, approved, declined, completed).

Coordinating Pickup Logistics:

Donation Request helps schedule and confirm the food pickup details based on mutual agreement between the user and foodbank admin.

***Constraints:***
Valid Food Items - The requested food items should be suitable for donation and comply with the food bank's guidelines.

Pickup Availability - Preferred pickup date and time should be within the food bank's operational hours and capabilities.

Data Security - User information and donation details should be securely stored and handled.

**Foodbank Class**

***Classification:***
Class

***Definition:***
The Foodbank class represents a specific food bank within the application's ecosystem. It captures the key information about the food bank and facilitates its operations while managing its interactions with users and admin accounts.

***Responsibilities:***
Managing Food Inventory:

Foodbank can track available food items, quantities, and expiration dates.

Processing Donation Requests:

Foodbank will Review user requests, approving suitable donations, and coordinating pickup logistics.

Report Generation:

Foodbank will Provide various reports to track food bank activity, donations, and volunteer contributions.

User Communication:

Foodbank keeps users informed about food bank services, donation status, and pickup arrangements.

***Constraints:***
Inventory Management - The system should prevent storing unsafe or expired food items.

Pickup Scheduling - Pickup dates and times should align with the food bank's operational hours and volunteer availability.

Data Security - Food bank information and user data should be securely stored and accessed.

**Chatroom Class**

***Classification:***
Class

***Definition:***
The chatroom class represents a virtual space within the application where users can interact and exchange messages in real-time.

***Responsibilities:***
Facilitate Communication:

Chatroom will create a platform for users to interact and exchange messages within the chatroom.

Manage Participants:

Track users who are currently engaged in the chatroom and potentially handle user access and permissions.

Store and Display Messages:

Provide a secure and efficient way to store, retrieve, and display chatroom messages for participants.

Enable Moderation:

Allow designated users to moderate the chatroom by managing messages, user access, and ensuring a safe and positive environment.

Maintain Chatroom History:

Persist chat room messages for potential review or record-keeping purposes.

Notify Users:

Provide optional notifications to participants about new messages or relevant events within the chatroom.

***Constraints:***
Access Control - Define and enforce access rules for joining, messaging, and moderation based on user roles and chatroom type.

Content Moderation - Implement mechanisms to prevent offensive or harmful messages and maintain a safe environment.

Message History Management - Determine how much chat history to store and implement methods for archiving or purging old messages.

User Privacy - Protect user privacy by ensuring secure message storage and access control.

**Post Class**

***Classification:***
Class

***Definition:***

The Post Class represents a specific message or content item shared within the chatroom. It encapsulates the data and functionalities associated with a post, allowing users to share information, engage in discussions, and interact with the content.

***Responsibilities:***

Content Management:

Storing and retrieving the post's text, images, and videos (if applicable).
Providing methods to access different aspects of the post content

Engagement Management:

Tracking and storing comments and reactions received on the post (getcomments, getreactions).
Facilitating user interaction with comments and reactions.

Post Management:

Allowing users to save or publish a post within the chatroom (savePost, publishPost).
Enabling users to delete their own posts (deletePost).

Visibility and Permissions:

Defining and enforcing access rules for viewing, commenting, and reacting to posts based on user roles or chatroom settings.

Chronology:

Maintaining the order of posts within the chatroom timeline.

Optional Features:

Time stamping posts for reference.
Supporting post editing or revision in certain cases.
Implementing voting or poll functionalities associated with posts.

***Constraints:***

Content Moderation - Mechanisms need to be in place to prevent inappropriate or harmful content in posts.

Spam Prevention - Implement measures to prevent flooding the chatroom with unwanted posts.

Storage and Performance: - Optimise data storage and retrieval to ensure efficient access and performance.

Scalability: - The Post Class should be able to handle increased chatroom activity and volume of posts without degrading performance.

User Privacy: - Protect user privacy by managing sensitive information in post content and comments.

**Item Inventory Class**

***Classification:***
Class

***Definition:***
The Item Inventory class represents a collection of food items within a user's personal inventory within your application. It tracks item details, manages inventory levels, and facilitates various actions related to item management and utilisation.

***Responsibilities:***
Item Management:

Item Inventory can add and remove items from the inventory,update item details and tracking current inventory levels for each item.

Expiration Tracking:

Item Inventory set and store expiration dates for items. It also generates notifications or alerts for items nearing expiration (based on user settings).

Searching and Filtering:

Item Inventory allows users to search for items by name or other criteria and filter items based on expiration dates, quantity, or other relevant factors.

Recipe Integration:

Item Inventory link items in the inventory to recipes and suggests recipes based on available items.It also tracks item usage in recipes and updates inventory levels accordingly.

Notification Management:

Item Inventory manage users preferences for expiration notifications including sending timely notifications to prevent food waste.

***Constraints:***
Data Validation - Implement mechanisms to ensure data integrity and prevent errors in item information (e.g., valid names, quantities, expiration dates).

Inventory Limits - Consider setting maximum inventory capacities based on storage constraints or user preferences.

Data Storage - Choose appropriate storage methods to ensure efficient item retrieval and updates.

Performance - Optimise inventory operations for smooth performance, especially with large item lists.

User Privacy - Protect user privacy by securely storing and managing inventory data.

**Recipe Class**

***Classification:***
Class

***Definition:***
The Recipe class represents a set of instructions for preparing a particular dish within the application. It encapsulates the recipe's ingredients, instructions, nutritional information, and other relevant details, enabling users to discover, plan, and execute recipes.

***Responsibilities:***
Recipe Information:

It should store the recipe's name, description, cooking time and calories. Recipe also should manage a list of ingredients with quantities and optional preparation notes to providing clear and concise cooking instructions, potentially with step-by-step guidance or multimedia elements.

Ingredient Integration:

Recipe will link recipe ingredients to the ItemInventory class which allows users to check inventory availability for recipes. Item Inventory will track ingredient usage and update inventory levels when recipes are prepared.

Search and Discovery:

Enabling users to search for recipes by name, ingredients, cuisine type, dietary restrictions, or other relevant criteria.

Suggesting recipes based on user preferences, inventory contents, or past cooking history.

User Interaction:

Allowing users to save favourite recipes, create personal recipe collections, and share recipes with others and provide features for rating and reviewing recipes.

***Constraints:`***
Data Accuracy - Ensure the accuracy of recipe information, including ingredient quantities, cooking times, and instructions.

Accessibility - Make recipes accessible to users with visual impairments or other disabilities through appropriate formatting and features.

Cultural Sensitivity - Respect cultural and dietary sensitivities when curating or suggesting recipes.

Content Attribution - Properly attribute recipes to their creators or sources to protect intellectual property rights.

Image and Video Storage - Consider storage and optimization strategies for visual elements associated with recipes.

## 3.2 Database Design

1. User

| userID | username | password | email | phoneNumber |
|---|---|---|---|---|

2. FoodBankAdmin

| adminID | username | password | email | phoneNumber |
|---|---|---|---|---|

3. ItemInventory

| barcode | itemname | quantity | expirationDate | notification |
|---|---|---|---|---|

4. Chatroom

| chatroomID | participants |
|---|---|

5. Participants

| userID | chatroomID |
|---|---|

6. Post

| userID | images | videos | comments | reaction |
|---|---|---|---|---|

7. DonationRequest

| donationID | userID | foodbank | pickUpTime | pickUpDate | status |
|---|---|---|---|---|---|

8. Recipe

| recipeID | title | ingredients | instructions | cookingTime |
|---|---|---|---|---|

9. FoodBank

| foodbankID | adminID | name | street | city | postcode | country | phoneNumber |
|---|---|---|---|---|---|---|---|

Figure 3.2.1 List of Relational Tables

**Mapping Tables Explained:**

***1. User Table***
This table stores information about all users in the system, including their userID as primary key, username, password, email address, and phone number. Think of it as your digital address in the food bank system.

***2. FoodBankAdmin Table***
Similar to the User table, this stores information about food bank administrators, but with potentially additional admin-specific privileges. It's like a special user account for managing the food bank side of things.

***3. ItemInventory Table***
This table keeps track of all food items available in the system. Each item has a unique barcode as a primary key, name, quantity, expiration date, and maybe even a notification flag for nearing expiry. It's your grocery list for the food bank, with all the important details.

***4. Chat room Table***
This table manages chat rooms for communication within the system. Each chat room has a chatroomID that works as the primary key and participants. The users can join multiple chat rooms. Think of it as a virtual room where people can talk about food donations, recipes, or anything else relevant.

***5. Participants Table***
This "bridge" table connects users to the chatrooms they participate in. It contains userID and chatroomID. It also simply records which user is in which chatroom, avoiding the need to repeat user information within each chatroom entry.

***6. Post Table***
This table stores all the informative posts shared within the system. Each post is linked to the user who created it, and it can contain userID, images, videos, comments, and reactions. Think of it as a social media board for sharing tips, recipes, or donation requests.

***7. DonationRequest Table***
This table tracks all donation requests made by users. Each request has its own donationID, links to the requesting userID, the chosen food bank, the preferred pick-up time and date, and its current status. It's like a record of requests to help keep things organised.

***8. Recipe Table***
This table stores all the delicious recipes shared within the system. Each recipe has a recipeID, title, list of ingredients, instructions, and even the estimated cooking time. It's a cookbook within the food bank system, encouraging healthy eating for everyone.

***9. FoodBank Table***
This table details all the registered food banks participating in the system. Each food bank has a foodbankID, links to its adminID, name, location details (street, city, postcode and country) and phone number. It's like the contact information for each food bank.

**Relationships between the mapping tables :**

***One-to-One Relationships:***

1. FoodBankAdmin and User: A food bank admin is a specific type of user, so they have a one-to-one relationship. Each admin has a corresponding user record.
2. FoodBank and FoodBankAdmin: Each food bank is managed by a single admin, and each admin is responsible for only one food bank, forming a one-to-one relationship.
3. DonationRequest and FoodBank: A donation request is associated with a specific food bank, indicating where the donor wants to pick up the items.
4. ItemInventory and User: Each user has their own item inventory that is connected to the application.
5. ItemInventory and Recipe: Each recipe contains a lot of ingredients which the user can search for the suitable recipe based on the items(ingredients) that they have in their inventory.

***One-to-Many Relationships:***

1. User and Post: A user can create multiple posts, but a post belongs to only one user.
2. User and DonationRequest: A user can make multiple donation requests, but each request is associated with only one user.

***Many-to-Many Relationships:***

1. User, Participants and Chatroom: A user can participate in multiple chatrooms, and a chatroom can have multiple users.

***Foreign Keys:***

1. The DonationRequest table has a foreign key userID that references the primary key userID in the User table.
2. The FoodBank table has a foreign key adminID that references the primary key adminID in the FoodBankAdmin table.

## 3.3 UI Design

Food Waste Reduction App (FoWRA) incorporates a vibrant and visually appealing colour scheme, with pink as the primary colour to signify freshness. The colour scheme is consistent throughout the app, including background elements, icons, buttons, and text. It creates a cohesive and recognizable visual identity, resonating with users and reinforcing the app's mission.

- Main fonts will be **Playfair Display & ArchivoBlack**.
- The system can be displayed in both portrait and landscape orientation.
- The system supports all types of screen resolution.
- Colour in this system mainly uses:
    - ❖ Floral White (#FFFDF6)
    - ❖ Light Pink (#FFA9F9)
    - ❖ Light Yellow (#FFF7AD)

### 3.3.1 Register User

Use Case Description :

| | |
|---|---|
| **ID** | UC01 |
| **Use Case** | Register User |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | The user has not yet registered an account in the app |
| **Post-Conditions** | The user has a registered account in the app, which allows them to log in and access its features |
| **Flow Of Events** | 1. User click “Sign Up” button<br>2. The app presents a registration form, prompting the user to enter their personal details such as name, email address and password.<br>3. The user fills in the required information in the registration form.<br>4. The app validates the entered information, checking for any missing or any invalid data.<br>5. The app stores the user’s account information securely |
| **Alternative Flow Of Events** | 4. If there are any errors, the app displays appropriate error messages and prompts the user to correct the information |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

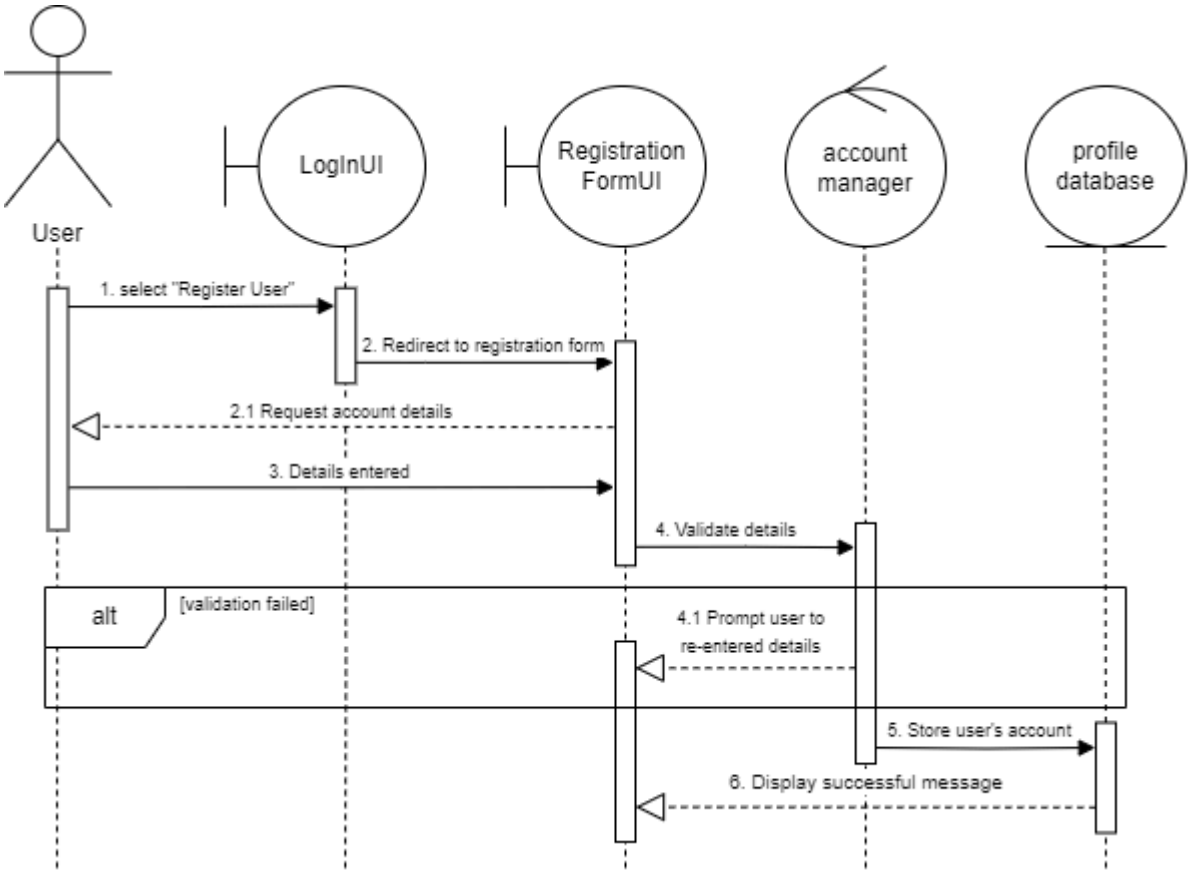
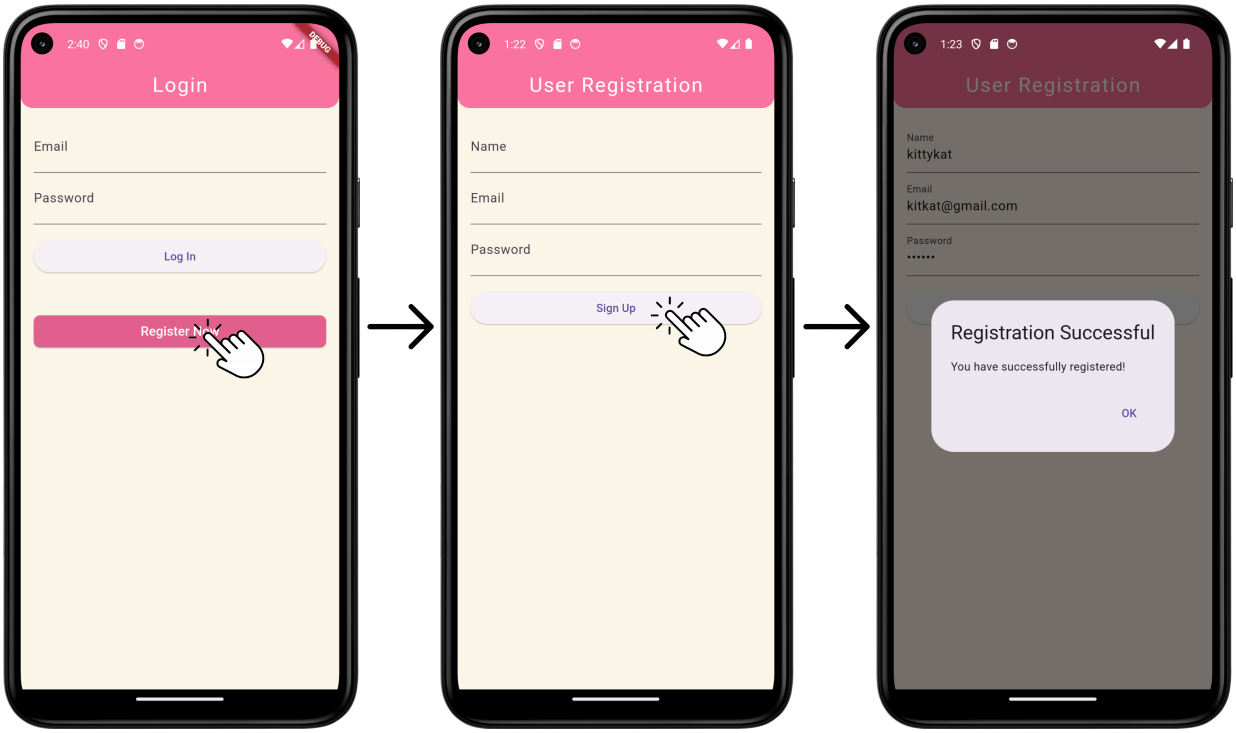



Figure 3.3.1 Interaction Diagram of Register User

The login interface serves as the primary screen upon initiating any action within the app. Upon clicking the "Register User" button, users are seamlessly directed to a user-friendly registration form interface. This interface features an intuitive layout with three input dialog boxes, facilitating the entry of their name, email, and password when creating a new account. Following successful validation, the system securely registers the user, storing their account information in the database. In the event of validation errors, the app promptly guides the user to rectify the details. A reassuring success message promptly appears, confirming the seamless completion of the registration process.



### 3.3.2 Update profile

Use Case Description :

| | |
|---|---|
| **ID** | UC02 |
| **Use Case** | Update Profile |
| **Priority** | Medium |
| **Actors** | User |
| **Pre-Conditions** | The user is logged into their account in the app |
| **Post-Conditions** | The user's profile information is updated and reflected in the app |
| **Flow Of Events** | 1. The user navigates to the "Profile" section within the app<br>2. The app displays the user's current profile information, including name, email address, and other relevant details<br>3. The user selects the option to edit their profile information and the app presents will interface that allows the user to modify their profile information<br>4. The user makes the desired changes, such as updating their name, email address, or password<br>5. The app validates the entered information, checking for any missing or invalid data<br>6. The app updates the user's profile with the new details<br>7. The app stores the updated profile information securely<br>8. The app confirms successful profile update and displays a notification or message to the user<br>9. The user's profile information is now updated in the app |
| **Alternative Flow Of Events** | 3. The user cancels or goes back to the previous screen without modifying their profile information.<br>5. If there are any errors, the app displays the appropriate error messages and prompts the user to correct the information |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

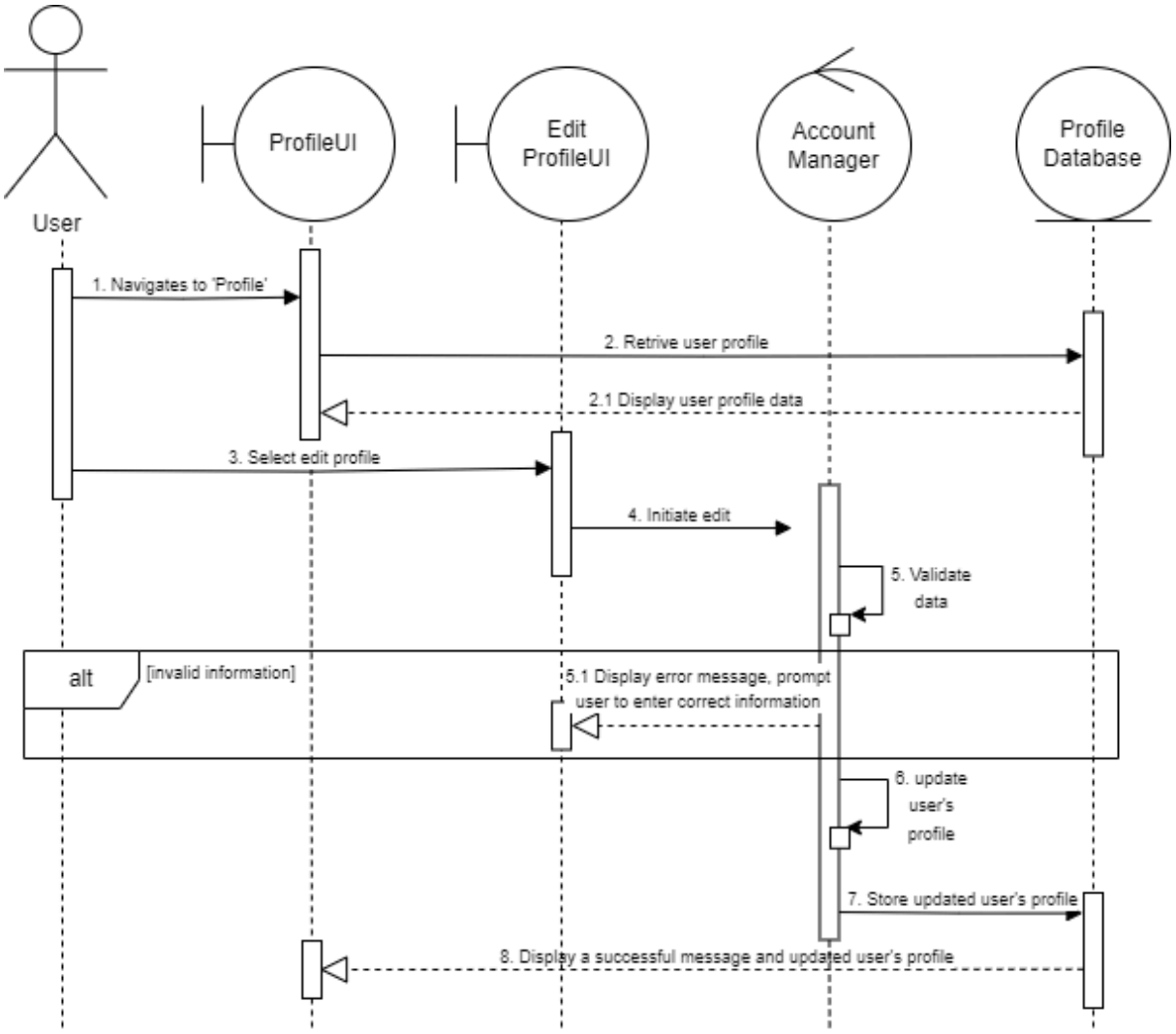
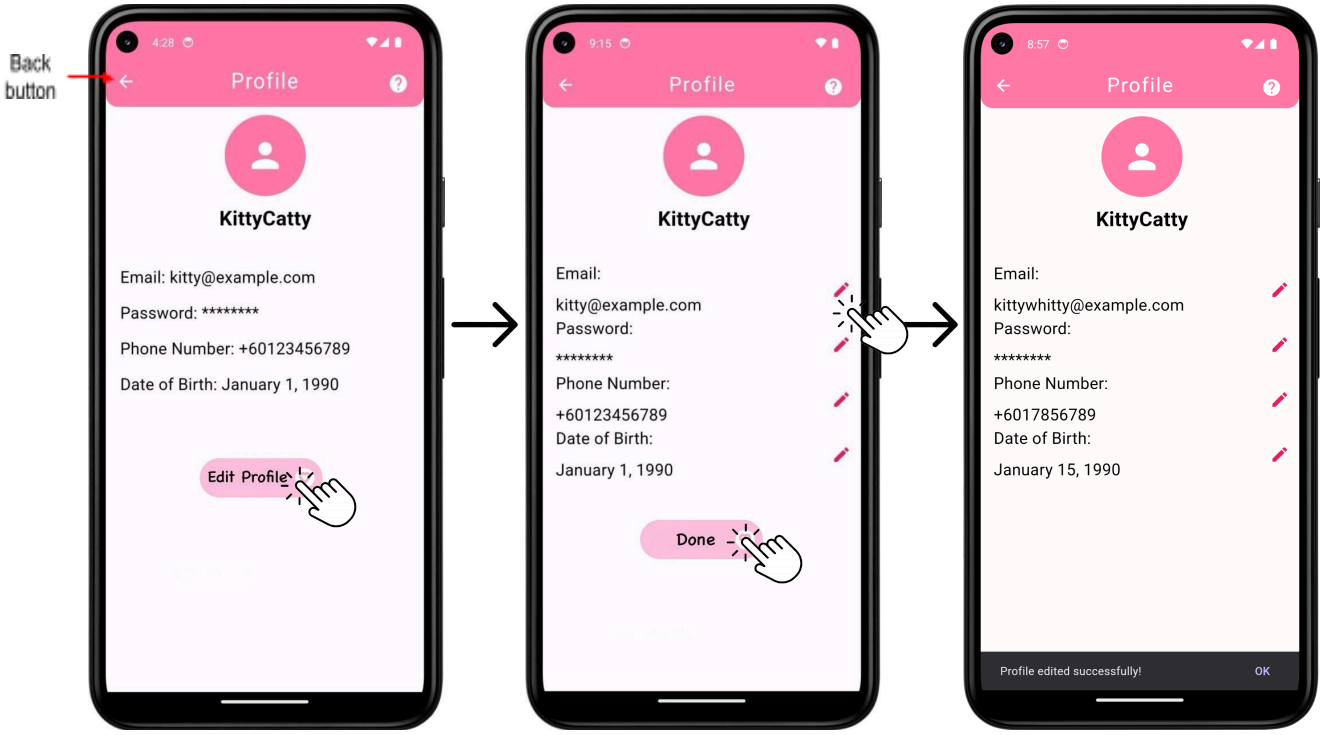
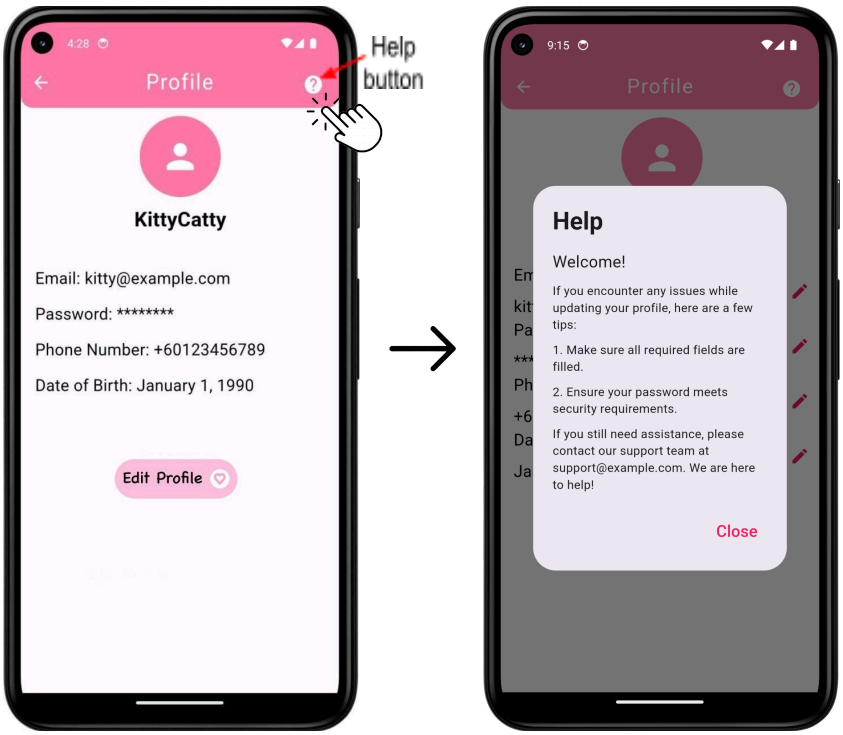



Figure 3.3.2 Interaction Diagram of Update User

In the profile interface, users can edit their details with the "Edit Profile" button. After making changes and clicking "Done," the app validates them. If successful, a success message appears; if not, users are prompted to review and re-enter information. A back arrow simplifies navigation to the profile page, and a "Help" button offers assistance by providing relevant information and tips.



### 3.3.3 Add item to Inventory

Use Case Description :

| | |
|---|---|
| **ID** | UC03 |
| **Use Case** | Add item to the inventory |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | User chooses the "Add Item" option in the inventory tab from the main menu |
| **Post-Conditions** | The item is successfully added to the user's inventory |
| **Flow Of Events** | 1. User selects the method either to scan the barcode or manually enter the item information.<br>2. If the user chooses to manually enter the item information, they can fill the information needed for the item.<br>3. The system asks for a picture of a new item.<br>4. User submits the picture by snap using a phone camera or chooses a photo from the gallery.<br>5. The new item and details will be validated.<br>6. The system adds the new item to the inventory list with the item’s details, including the quantity available.<br>7. Inventory list then list all the items that have been added.<br>8. The system will present the new list of items in the inventory. |
| **Alternative Flow Of Events** | 2. If the user chooses to scan the barcode, they can scan the barcode by their phone camera or choose a photo from the gallery. |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

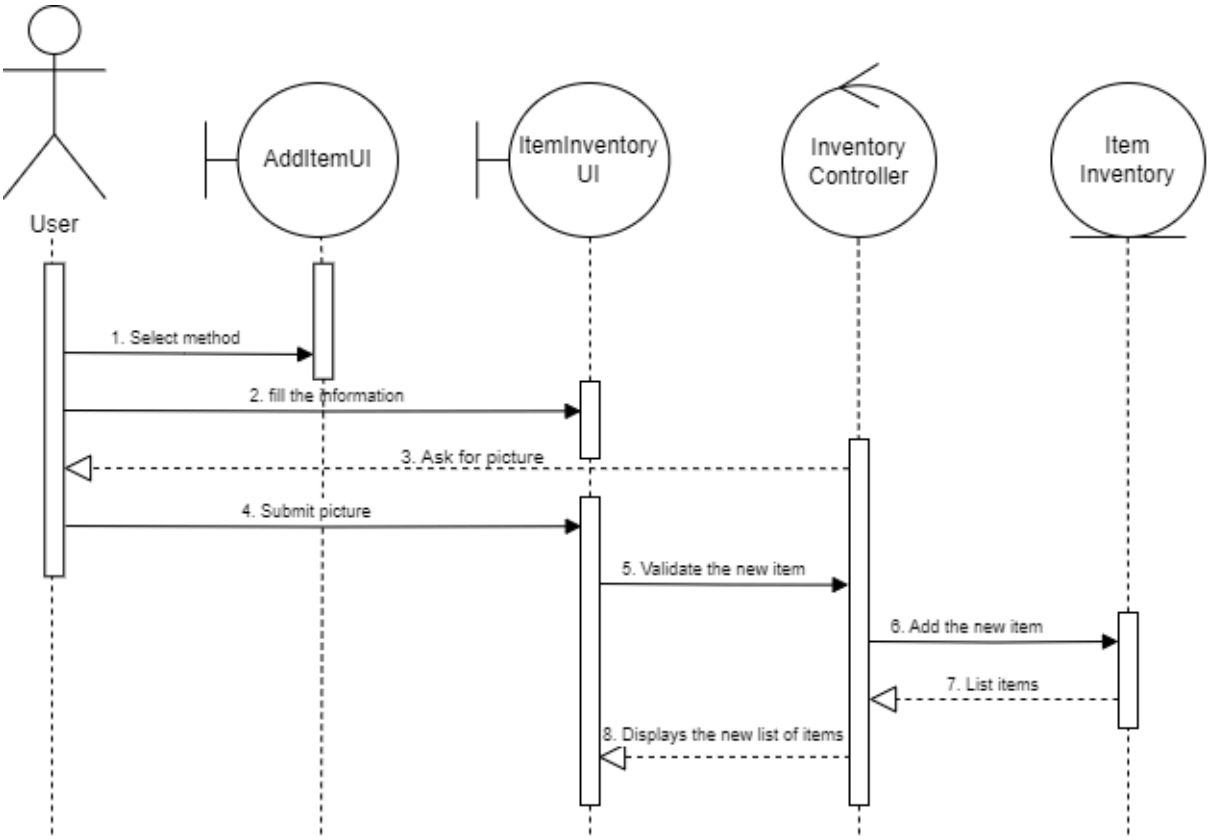
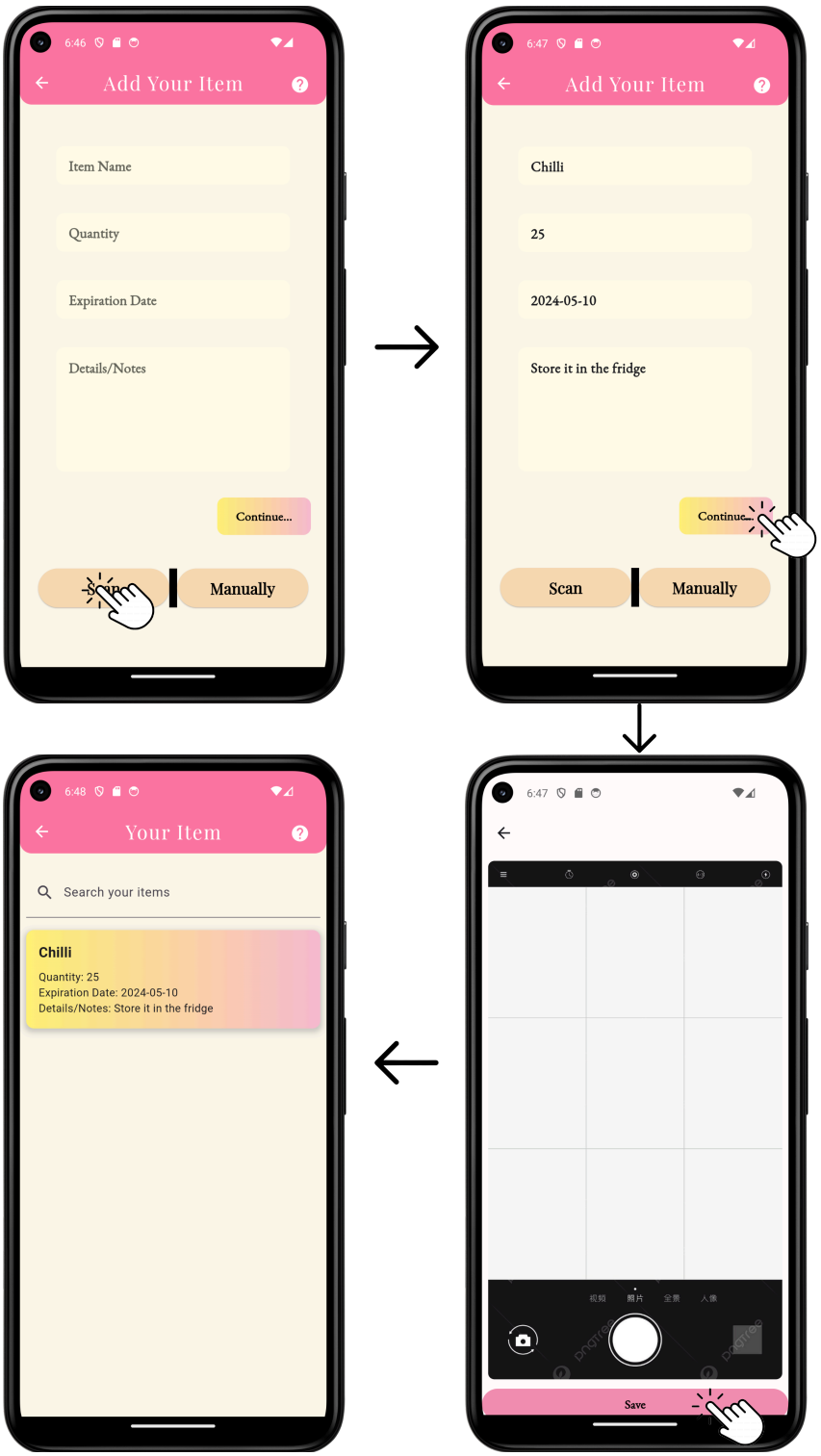
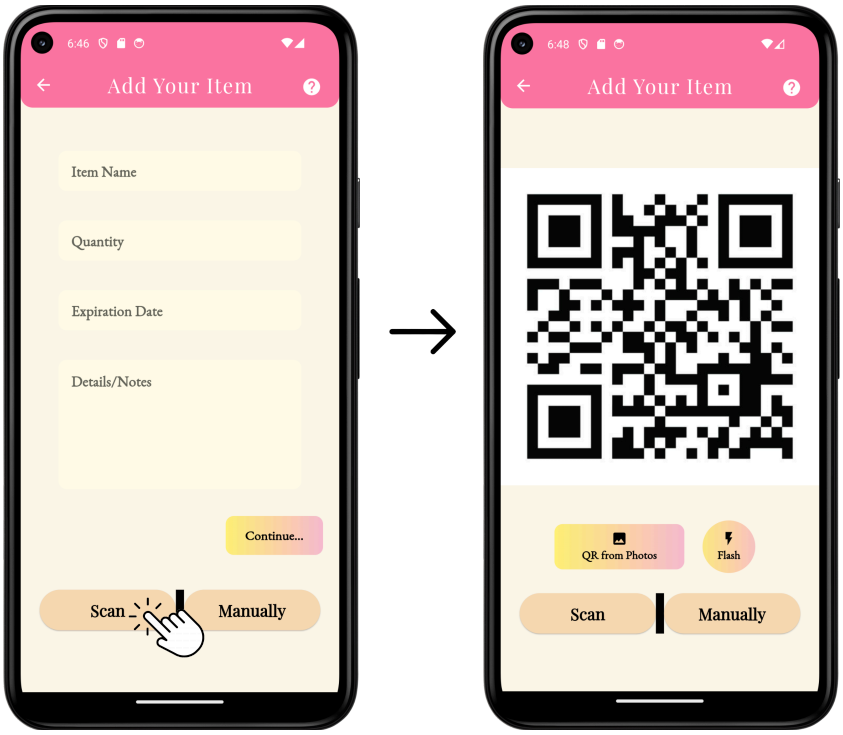



Figure 3.3.3 Interaction Diagram of Add Item to Inventory

Users have the flexibility to choose their preferred method for adding items to the inventory: either manually inputting the details or streamlining the process by scanning the item's barcode. If the manual option is selected, the system prompts users to input essential information about the item. Additionally, users are encouraged to provide a snapshot of the item, which can be conveniently captured in real-time using their phone camera or selected from their gallery. Once this information is supplied, the system seamlessly incorporates the new item into the inventory list, presenting users with an updated and comprehensive list for their convenience.



When the "Scan" option is chosen, users have the ability to effortlessly scan the barcode of their item using the dedicated scan function.



### 3.3.4 Remove item in Inventory

Use Case Description :

| | |
|---|---|
| **ID** | UC04 |
| **Use Case** | Remove item in the inventory |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | User chooses the "Remove Item" option in the inventory tab from the main menu |
| **Post-Conditions** | The item is successfully removed from the user's inventory |
| **Flow Of Events** | 1. System display list of item in the inventory<br>2. The user can choose to remove the items based on the quantity of items that they want.<br>3. The user saves the new inventory list.<br>4. The system will present the new list of items in the inventory. |
| **Alternative Flow Of Events** | 3. The user can click the “undo” option if there is any problem with the quantity of items that has been removed. |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

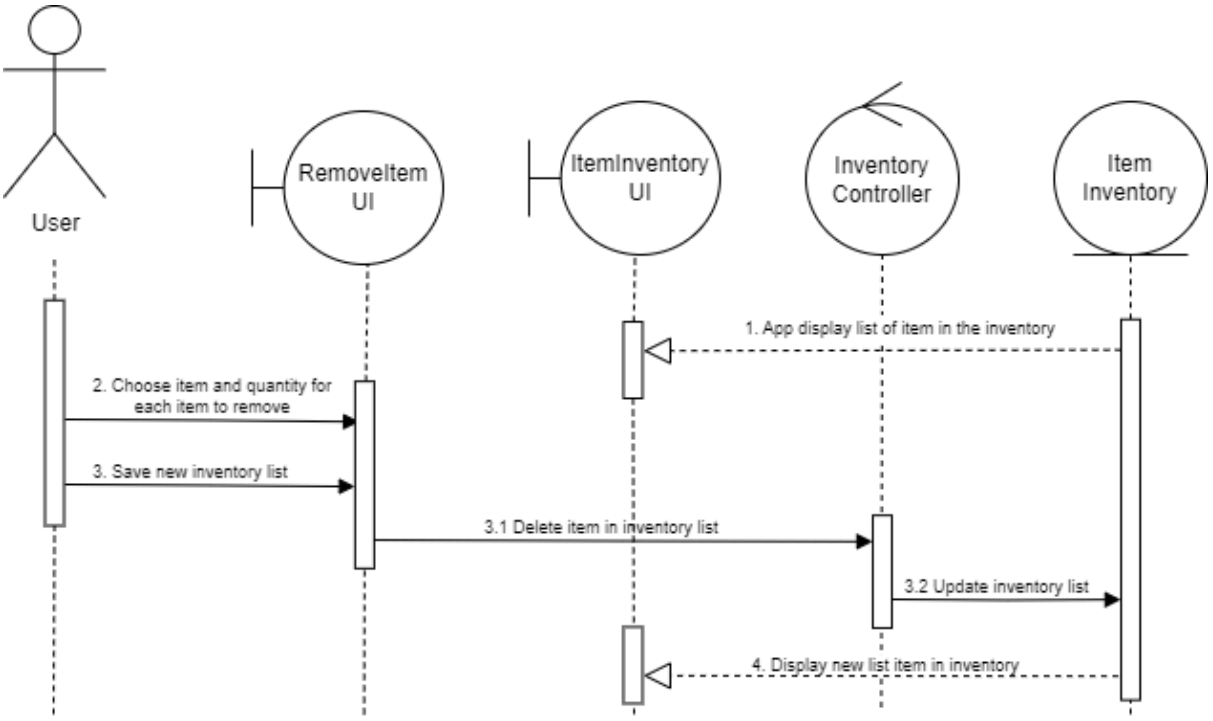
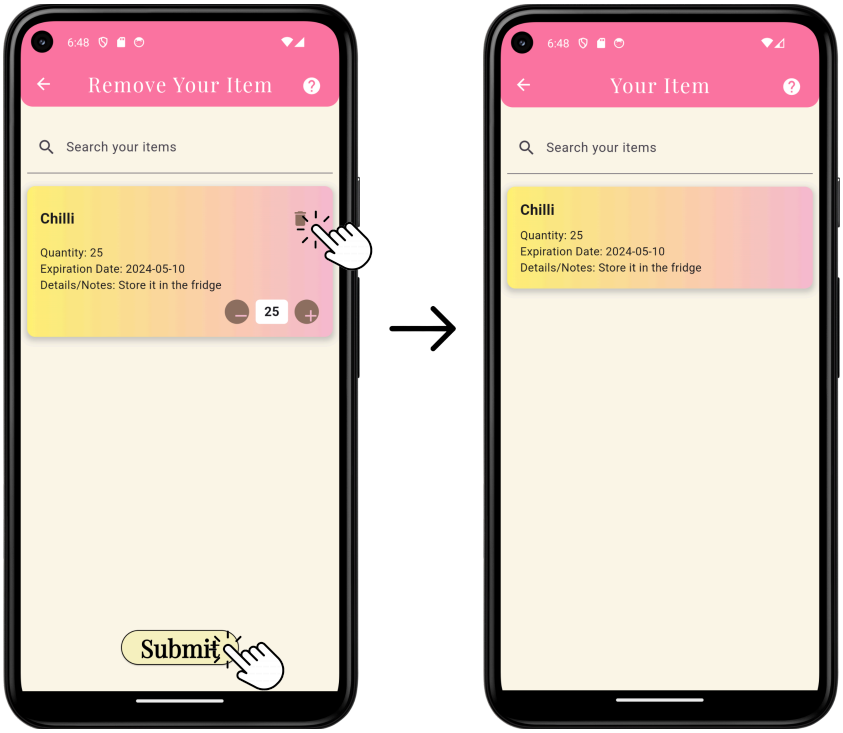



Figure 3.3.4 Interaction Diagram of Remove Item in Inventory

The system will initially present the user with a comprehensive list of items currently available in their inventory. Then, the user can make selections by choosing specific items and specifying the desired quantity for removal. Upon confirmation, the system will proceed to update and save the revised inventory list. Finally, the user will be presented with the updated inventory, reflecting the changes made.



### 3.3.5 Food Expiration Tracking

Use Case Description :

| | |
|---|---|
| **ID** | UC05 |
| **Use Case** | Remove item in the inventory |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | User set the expiration date for the food that has been added into the inventory. |
| **Post-Conditions** | User receives notifications about upcoming or expired food items |
| **Flow Of Events** | 1. The user to set up the expiration date tracking manually by entering the date of expiration for the food.<br>2. The system asks the user to set the notification and reminder for the expiration date automatically or manually.<br>3. The system display notification settings.<br>4. The user can choose to receive notifications through various channels such as email, SMS or mobile app notifications.<br>5. The system will display confirmation for expiration date tracking setup |
| **Alternative Flow Of Events** | 1. The system automatically tracks the expiration date for the food that has been put in the inventory.<br>3. The system will set the notification and reminder for the expiration date automatically |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

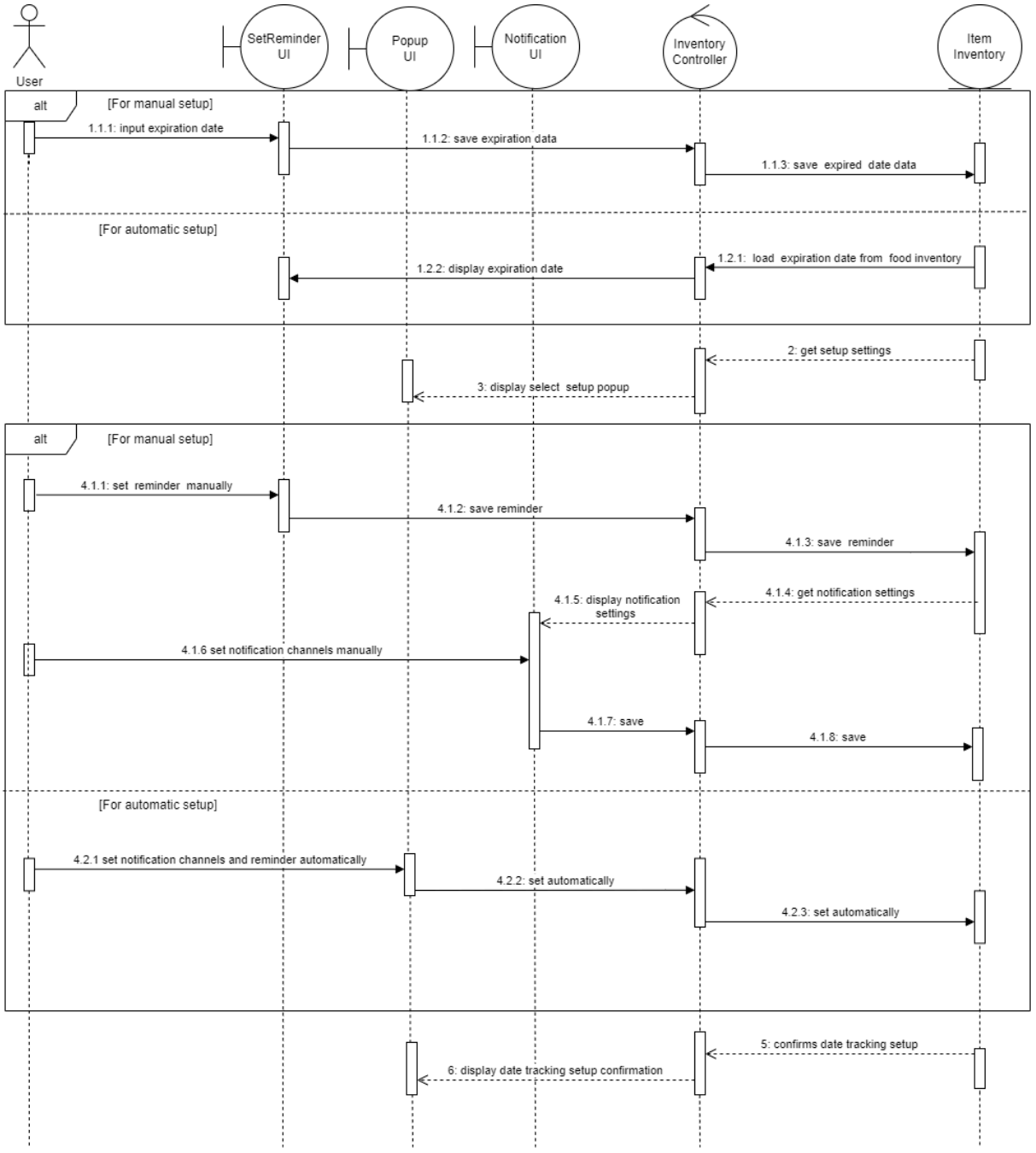
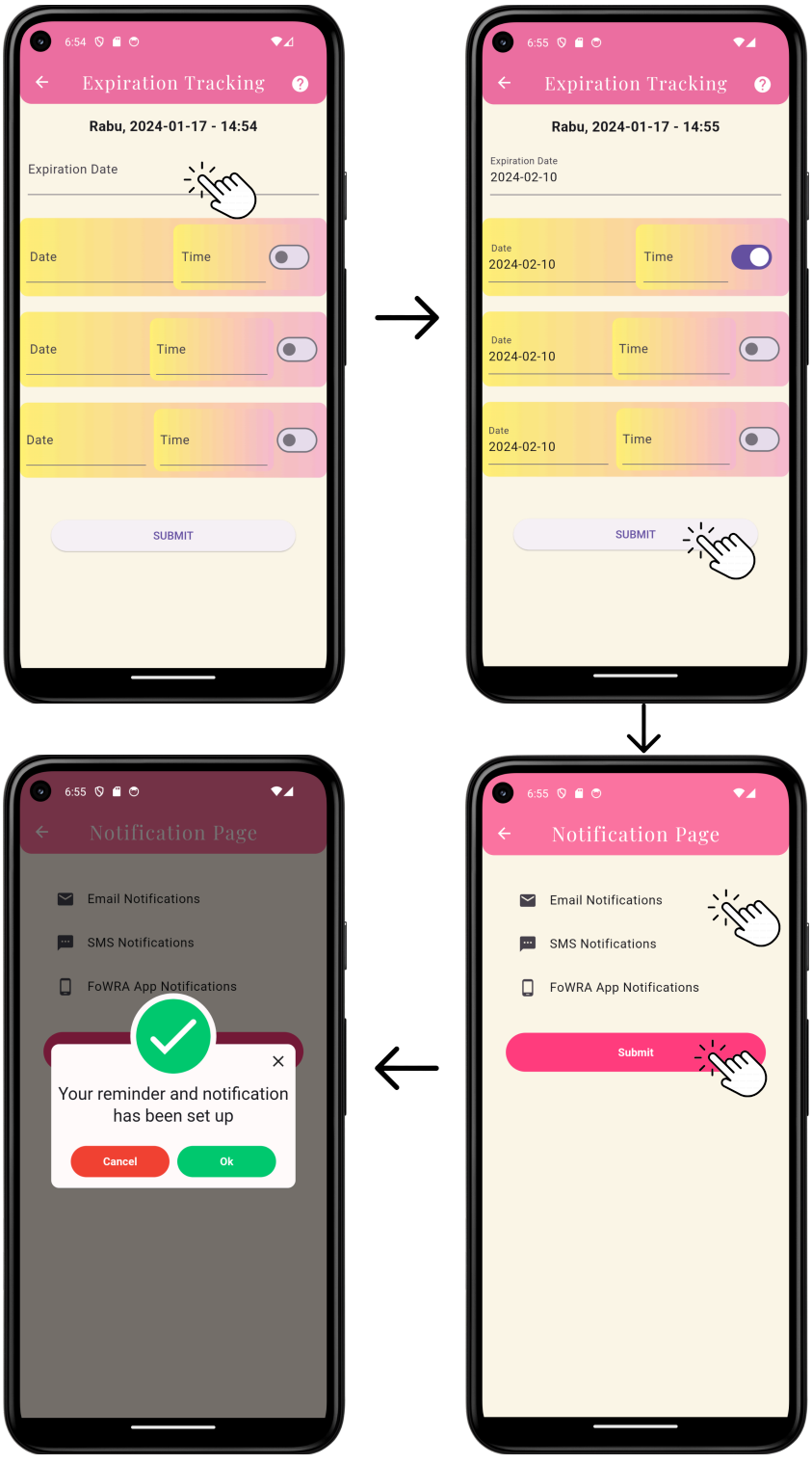



Figure 3.3.5 Interaction Diagram of Food Expiration Tracking

Users can choose to set expiration dates manually or opt for automatic configuration. In automatic setup, the system retrieves expiration dates from the inventory. In manual setup, users input the desired expiration date. Users can also manage notifications manually by setting reminders and choosing notification channels (email, SMS, or phone notification bar) in their phone settings. With automatic setup, reminders are generated automatically based on specified expiration dates when adding items to the inventory.



### 3.3.6 Recipe Suggestions

Use Case Description :

| | |
|---|---|
| **ID** | UC06 |
| **Use Case** | Recipe Suggestions |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | User inputted the ingredients they have on hand |
| **Post-Conditions** | The user can view a selected recipe and its details and save the recipe to their favourites |
| **Flow Of Events** | 1. User inputs the ingredients into the system.<br>2. The system retrieves its recipe database for recipes based on the existence of the ingredients.<br>3. The system displays a list of recipe suggestions.<br>4. The recipe page displays the ingredients, steps, and cooking time for the selected recipe.<br>5. User can save the recipe to their favourites, or return to the recipe suggestions page. |
| **Alternative Flow Of Events** | 3. If the system can’t find any recipe suggestions based on the user's inputted ingredients, it will prompt the user to try a different combination of ingredients ( return to step 2 ) |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

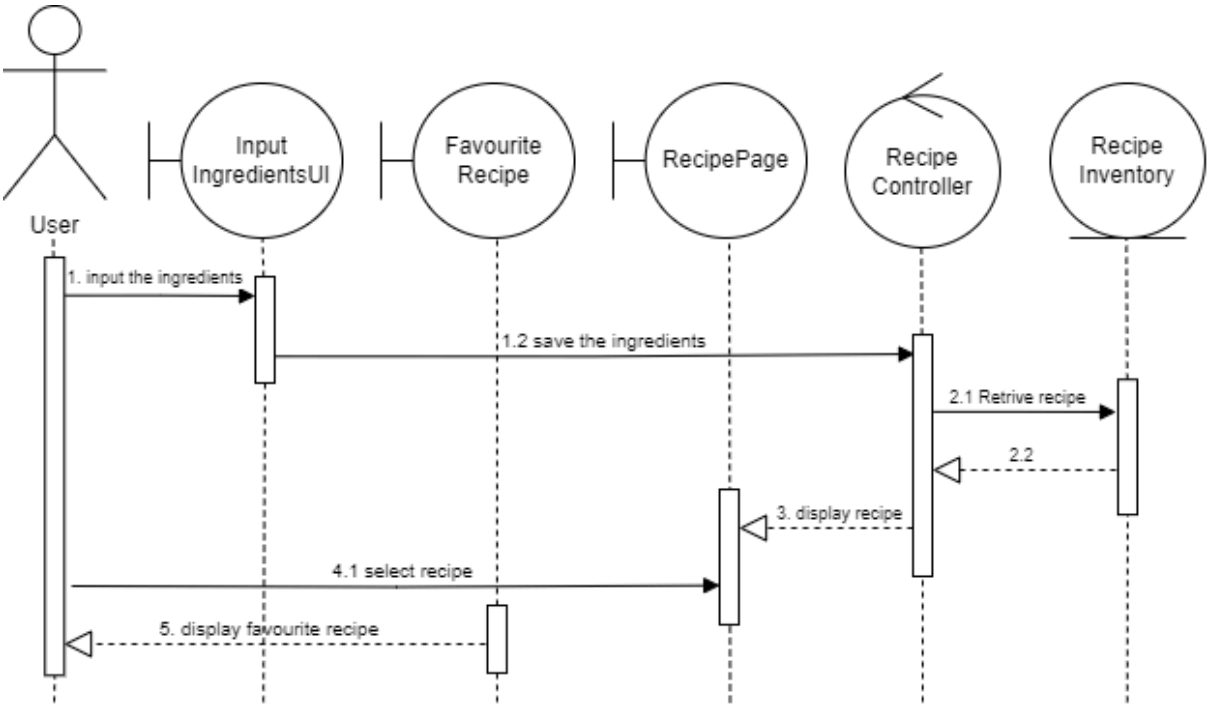
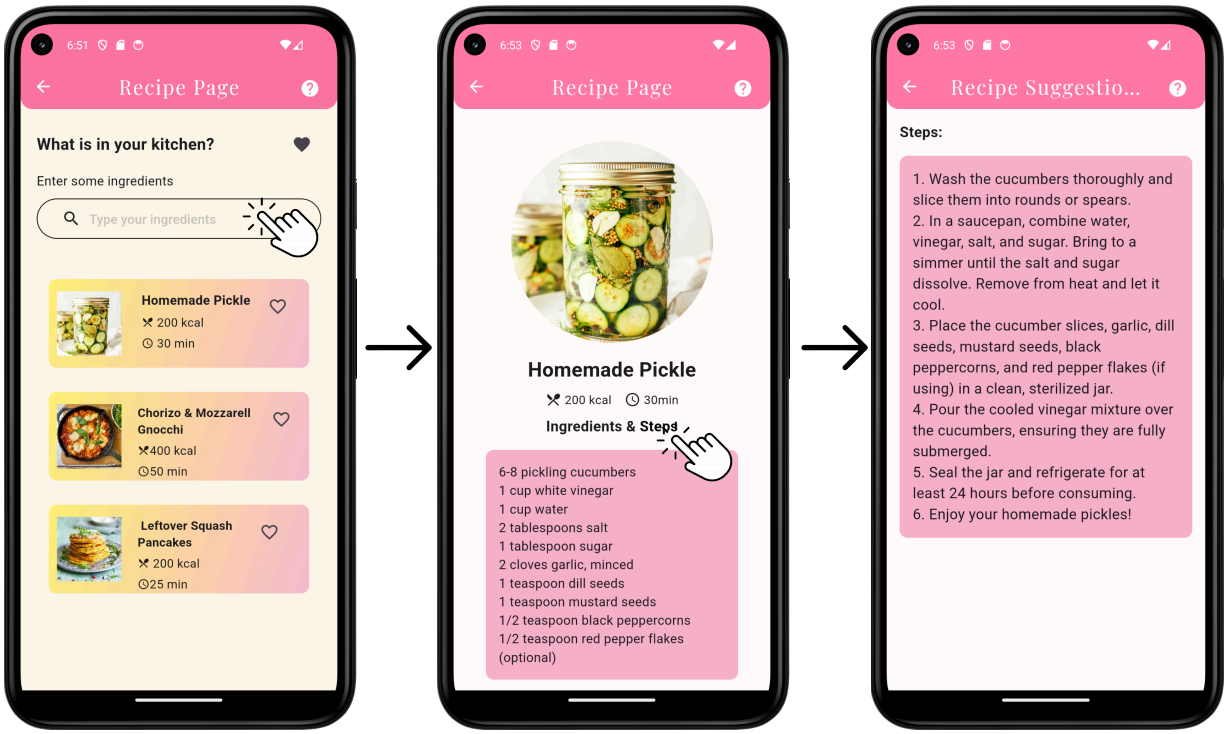
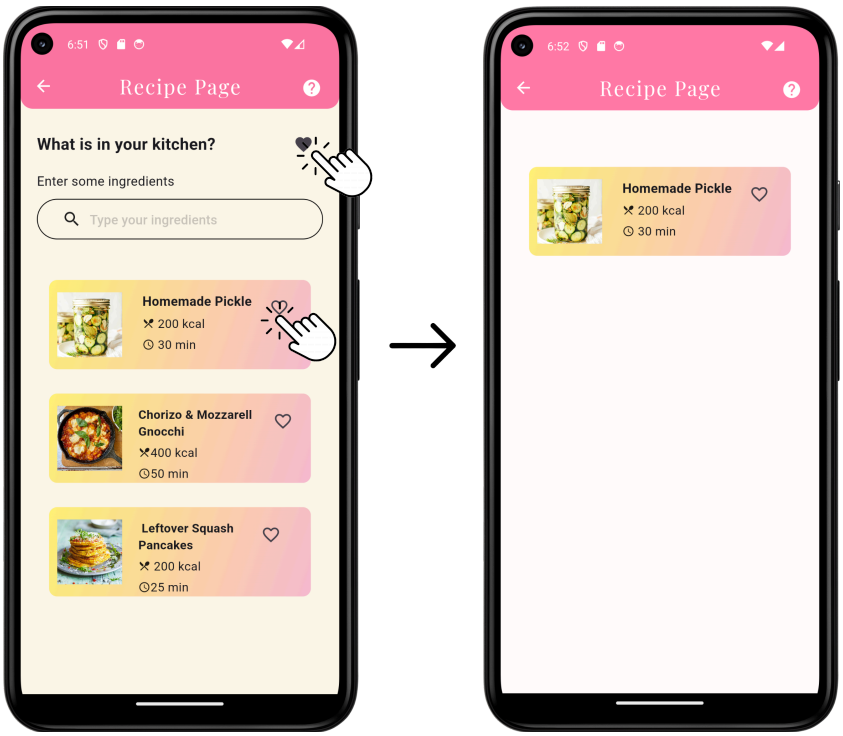



Figure 3.3.6 Interaction Diagram of Recipe Suggestions

The user begins by inputting the available ingredients into the system. Subsequently, the system generates a curated list of recipe suggestions tailored to the user's input and the ingredients present in their inventory. Once the user selects a preferred recipe, the system provides comprehensive details, including the required ingredients, step-by-step instructions, and estimated cooking time for the chosen recipe. Additionally, users have the option to save their favourite recipes to a dedicated favourites page for future reference.



### 3.3.7 Write a Post

Use Case Description :

| | |
|---|---|
| **ID** | UC07 |
| **Use Case** | Write a Post |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | User select the “Community” tab from the main menu to write a post |
| **Post-Conditions** | User’s posts are published successfully to the platform and can be viewed by the other user |
| **Flow Of Events** | 1. User selects the “Create a post” option.<br>2. User writes the post content and attaches any relevant images and videos.<br>3. System will display the preview of the post and make any necessary edits.<br>4. User publish the post<br>5. System saves and posts the contents to the community.<br>6. System will display the finalised posts |
| **Alternative Flow Of Events** | 4. User wants to continue editing the post and save it as draft. User also confirm cancellation, and will be redirected back to the community section without posting the content. |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

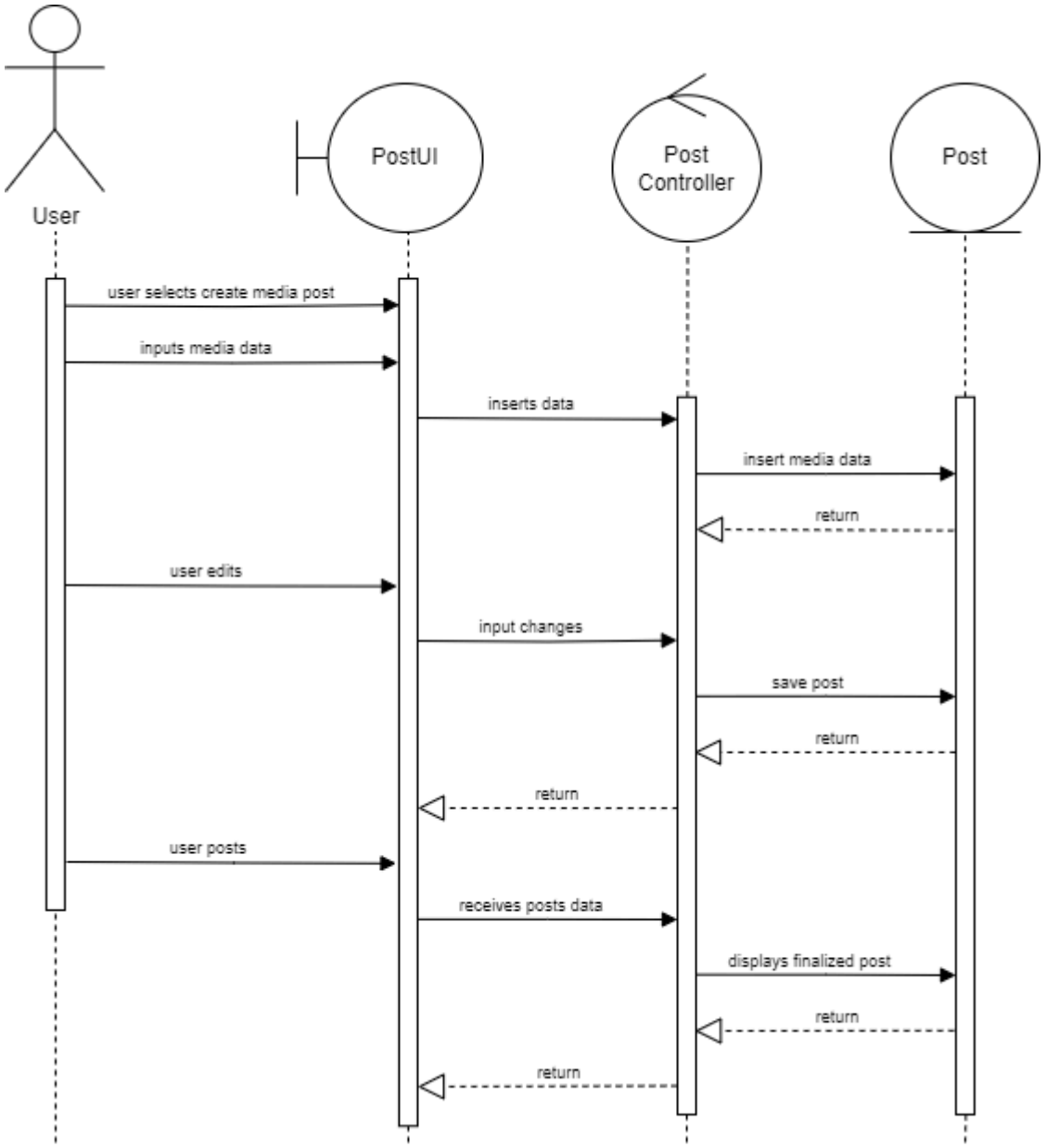
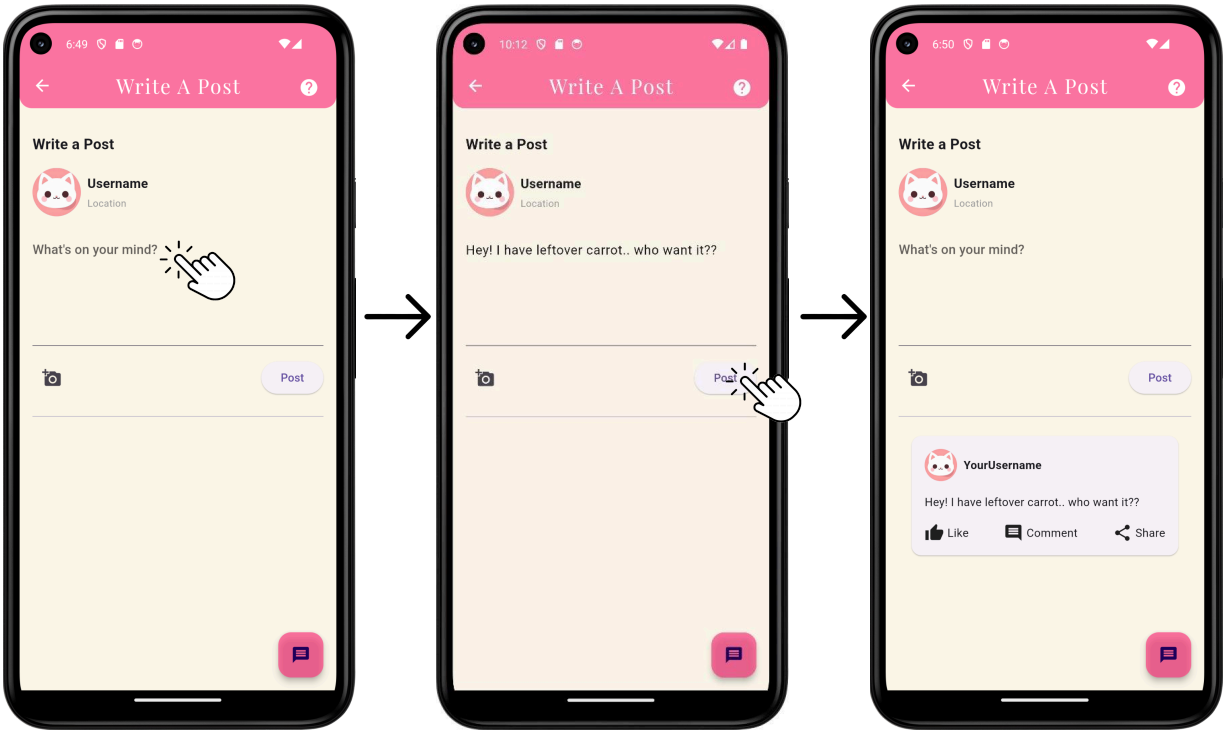



Figure 3.3.7 Interaction Diagram of Write a Post

Users can seamlessly initiate their posts by effortlessly typing in the designated field. Additionally, they have the convenience of attaching relevant images and videos to enhance their content. Upon clicking the post button, the application promptly saves and shares the submitted content with the community, presenting it in an engaging and visually appealing display.



### 3.3.8 Public Chat

Use Case Description :

| | |
|---|---|
| **ID** | UC08 |
| **Use Case** | Public Chat |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | The use case starts when the user has access to the public chat platform that is available in the application. |
| **Post-Conditions** | User successfully joined the public chat in the application. |
| **Flow Of Events** | 1. User joins the chat room.<br>2. The user sends messages.<br>3. The message database retrieve all the messages<br>4. The system broadcasts the messages to all the connected users in the system. |
| **Alternative Flow Of Events** | 1. User can leave the chat room when desired.<br>2. User can also choose to send private messages to other users. |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

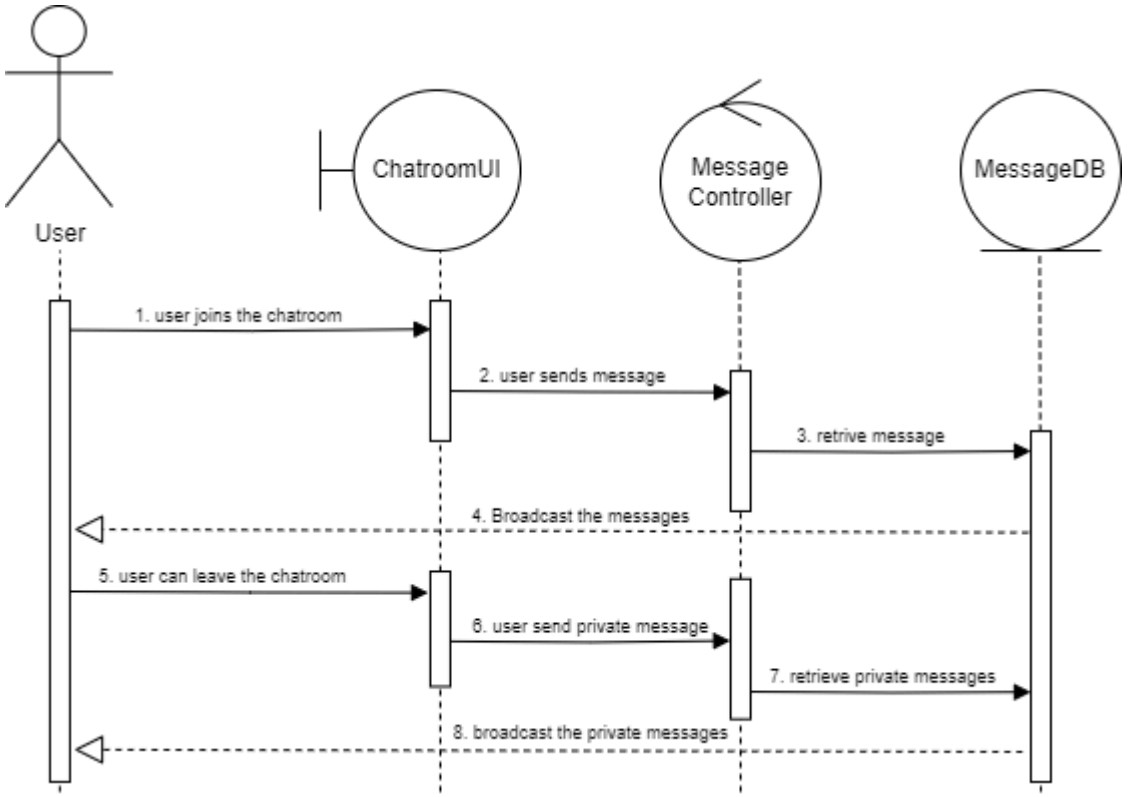
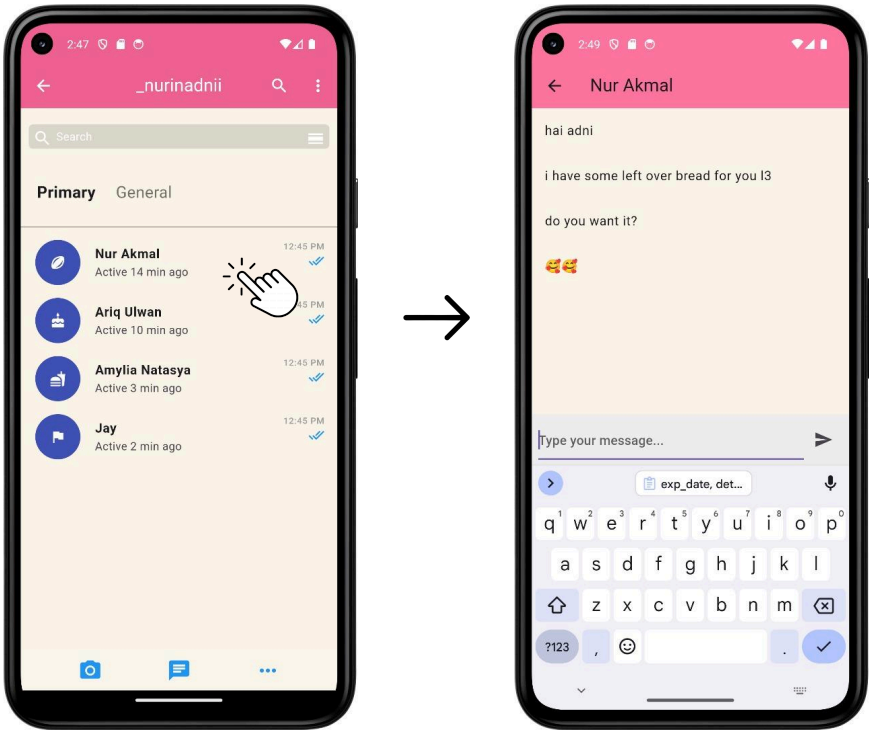



Figure 3.3.7 Interaction Diagram of Public Chat

Upon entering the chat platform, users can seamlessly navigate to their preferred chat room or initiate a conversation with a specific user. Once selected, users can craft and dispatch messages that will be broadcasted to the chosen chat room or recipient.



### 3.3.9 Request Donations

Use Case Description :

| | |
|---|---|
| **ID** | UC09 |
| **Use Case** | Request Donations |
| **Priority** | High |
| **Actors** | User |
| **Pre-Conditions** | User select the donation option from the home page |
| **Post-Conditions** | User receives a confirmation of the successful donation |
| **Flow Of Events** | 1. The system prompt user to insert donation information<br>2. User select a nearby food bank that is participating in the program and also available pickup date and times<br>3. Donation request will be validated<br>4. Donation request created<br>5. Donation request send to the food bank admin to be confirm<br>6. User can see their donations history and details at “Your donations” option |
| **Alternative Flow Of Events** | 3. System displays an error message stating that there are error while attempting to create the donations request. Prompt user to re-enter donations details. |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

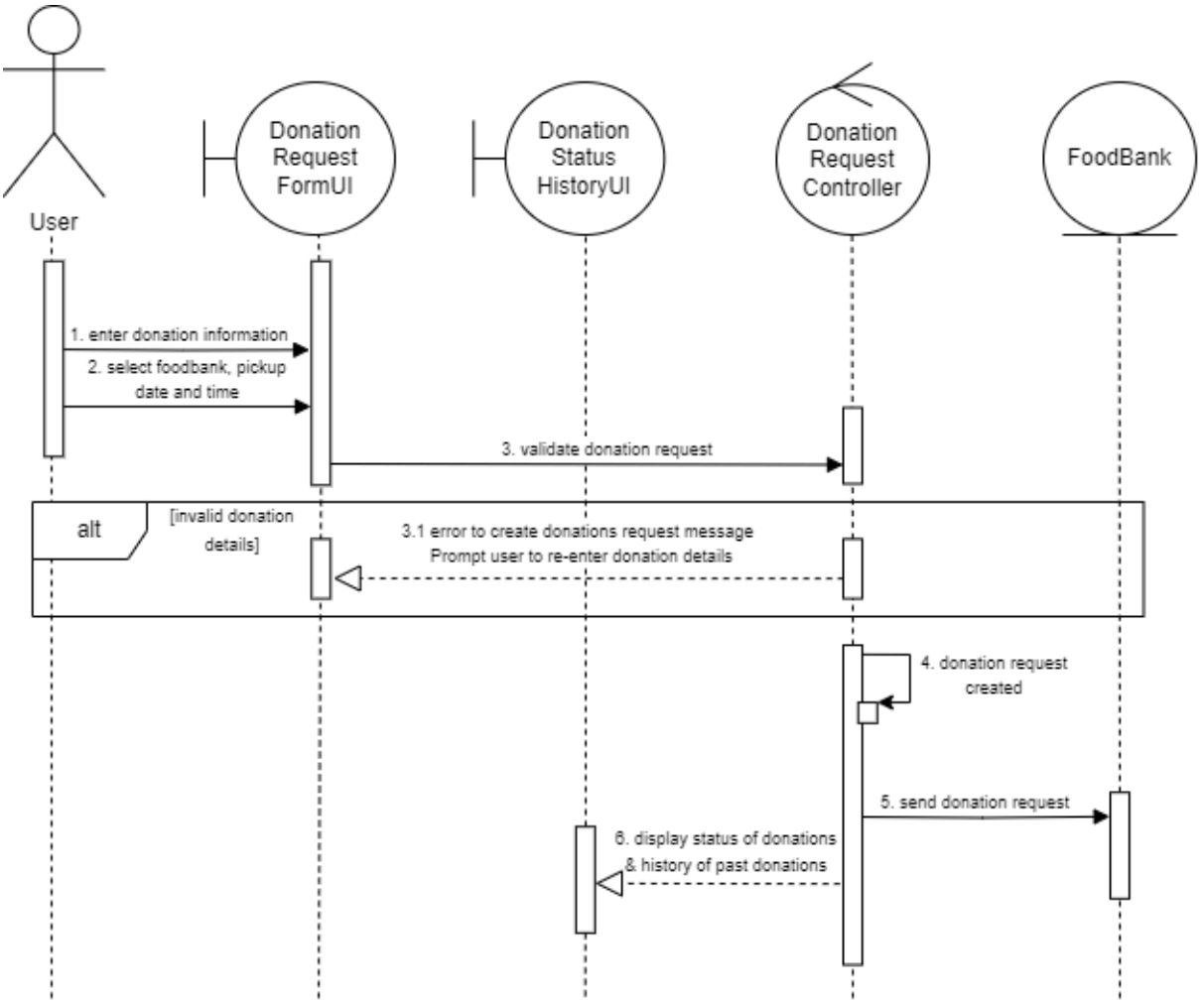
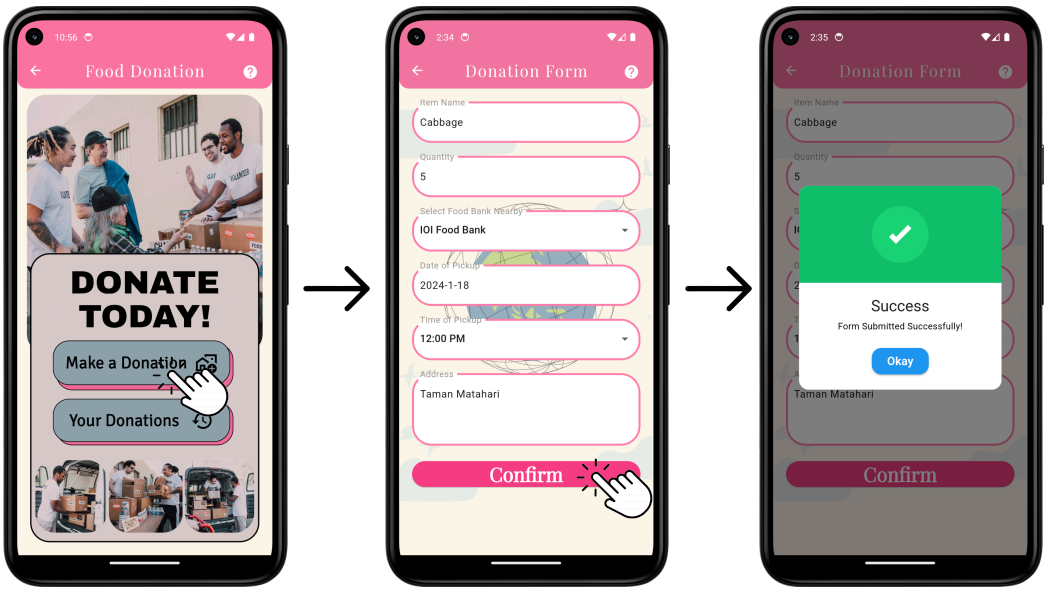
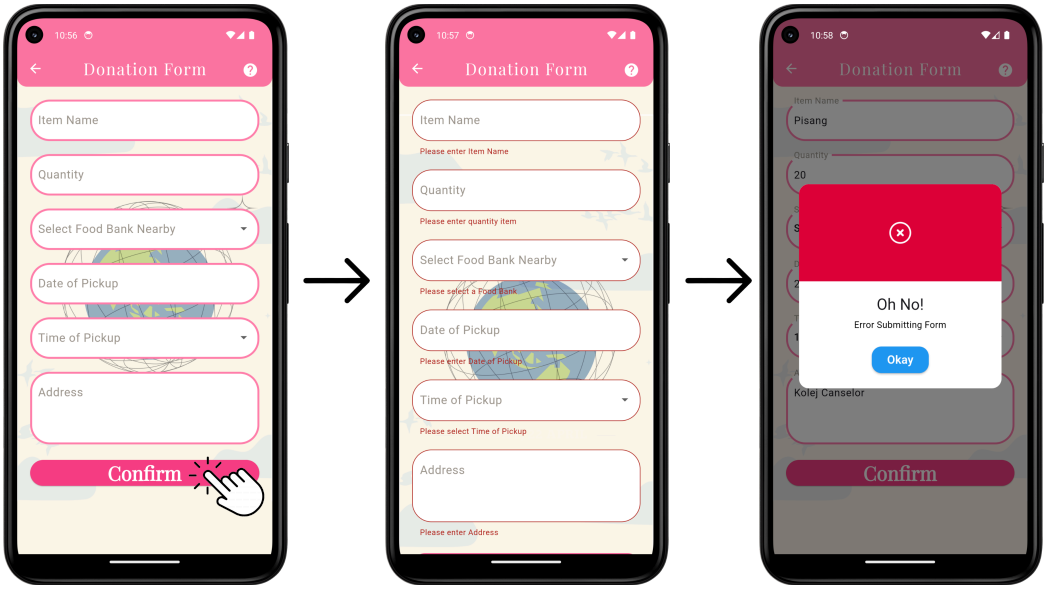



Figure 3.3.7 Interaction Diagram of Request Donations

The application initiates by prompting users to input their donation information. Then, users can choose a nearby food bank participating in the program, along with a convenient pickup date and time. Upon completing these selections, users can confidently click the "Confirm" button to submit the form. A successfully submitted form will display a successful message. In the event of any incomplete information or errors, the application will display an error message, prompting the users to re-entering their donation details. Users can view their live and history of their donation request under the “Your Donations” option.


Food Donation
DONATE TODAY!
Make a Donation
Your Donations
Donation Form
Item Name
Cabbage
Quantity
5
Select Food Bank Nearby
IOI Food Bank
Date of Pickup
2024-1-18
Time of Pickup
12:00 PM
Address
Taman Matahari
Confirm
Success
Form Submitted Successfully!
Okay


Donation Form
Item Name
Quantity
Select Food Bank Nearby
Date of Pickup
Time of Pickup
Address
Confirm
Please enter Item Name
Please enter quantity item
Please select a Food Bank
Please enter Date of Pickup
Please select Time of Pickup
Please enter Address
Pisang
20
Kolej Canselor
Oh No!
Error Submitting Form
Okay


Food Donation
DONATE TODAY!
Make a Donation
Your Donations
Your Donations
Live Donations
Food Bank Malaysia
Quantity: 5
Time of Pickup: 5:00PM
Date: 15/1/2024
Status : Pending
Your Donations History
Yayasan Food Bank
Quantity: 10
Time of Pickup: 12:00PM
Date: 11/1/2024
Status : Completed
Your Food Bank
Quantity: 20
Time of Pickup: 8:00AM
Date: 01/1/2024
Status : Completed
Cambrige Food Bank
Quantity: 8
Time of Pickup: 9:00PM
Date: 30/12/2023
Status : Rejected
Donation Details
Food Bank: Food Bank Malaysia
Quantity: 5
Pickup Time: 5:00PM
Date: 15/1/2024
Status: Pending
Close

### 3.3.10 Confirm Donations

Use Case Description :

| | |
|---|---|
| **ID** | UC10 |
| **Use Case** | Confirm Donations |
| **Priority** | High |
| **Actors** | Food Bank Admin |
| **Pre-Conditions** | Admin receive donation request |
| **Post-Conditions** | Admin confirmed or reject donation request |
| **Flow Of Events** | 1. Admin either confirms or rejects the donation request<br>2. Update status of the donations<br>3. Admin can view the history of donation requests under the "Request History" option |
| **Alternative Flow Of Events** | None |
| **Includes** | None |
| **Notes/Issues** | None |

Interaction Diagram :

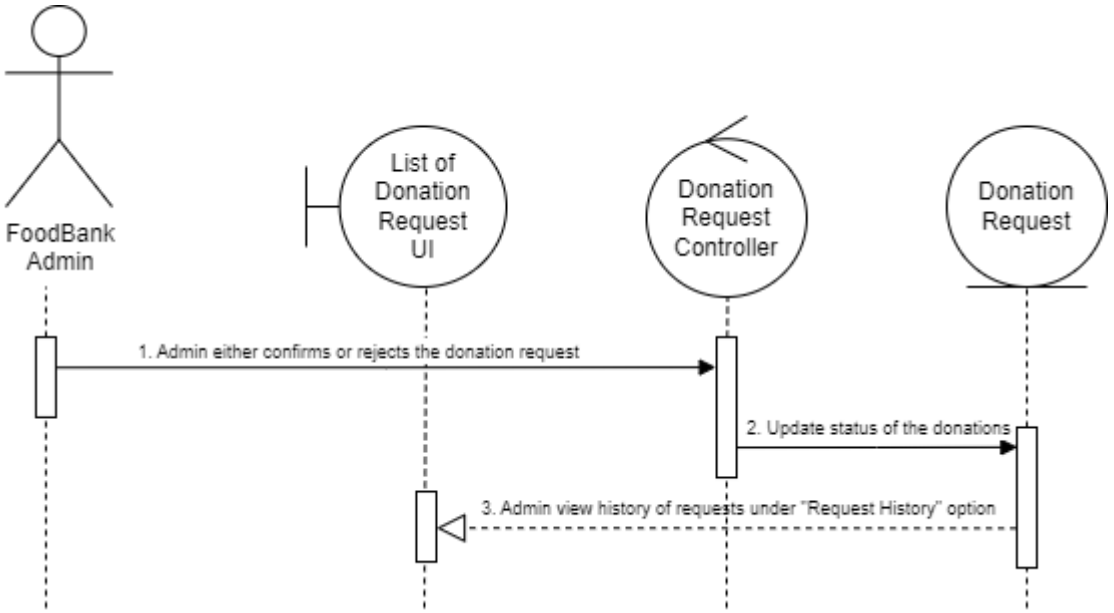



Figure 3.3.7 Interaction Diagram of Confirm Donations

The Food Bank admin will be presented with a comprehensive list of pending donation requests awaiting approval. The administrator can then seamlessly navigate through the options to either approve or reject each donation request. Additionally, the admin has the convenience of accessing and reviewing the complete history of donation requests through the user-friendly interface under the dedicated "Request History" option.


10:52
Admin
DONATION REQUEST
Pending Approval
Request History
10:53
Donation Request
gvrsrdhninsn_
Quantity: 5
Time of Pickup: 5:00PM
Date: 15/1/2024
Approve
Reject
danishabahni
Quantity: 10
Time of Pickup: 12:00PM
Date: 11/1/2024
Approve
Reject
dear._sunflower
Quantity: 20
Time of Pickup: 8:00aM
Date: 01/1/2024
Approve
Reject
10:56
Food Donation
DONATE TODAY!
Make a Donation
Your Donations
10:54
Request History
gvrsrdhninsn_
Quantity: 5
Time of Pickup: 5:00PM
Date: 15/1/2024
Status : Confirmed
danishabahni
Quantity: 10
Time of Pickup: 12:00PM
Date: 11/1/2024
Status : Confirmed
dear._sunflower
Quantity: 20
Time of Pickup: 8:00AM
Date: 01/1/2024
Status : Rejected
bummyydummyy
Quantity: 8
Time of Pickup: 9:00PM
Date: 30/12/2023
Status : Rejected
11:00
Request History
Donation Details
Username: gvrsrdhninsn_
Quantity: 5
Pickup Time: 5:00PM
Date: 15/1/2024
Status: Confirmed
Address: Taman Bukit Utara
Close

# 4.0 Architecture Design

## 4.1 Architecture Requirement

| Quality Attributes | Architecture Requirement |
|---|---|
| Performance | The system should be accessed within less than 5 seconds. |
| Availability | The system should have an uptime of at least 99.9% per month, allowing for scheduled maintenance windows. |
| Accessibility | The system provides a “Help” button to help users navigate the app more effectively and understand its features by offering a centralised location for users to seek assistance and connect users to useful resources. |
| Security | Only the authorised users can log into the system and there is a login attempt limit of 3 for each user. |
| | The system ensures secure communication and data transmission by implementing encryption and authentication measures. |
| Correctness | The system provides all the required functionality desired by the user with a correctness percentage of 99%. |
| Reliability | The system should be performed without having any failure and it should not break down more than 3 times in a week. |
| | The system should have a mean time between failures (MTBF) of at least 1,000 hours, with a mean time to repair (MTTR) of less than 1 hour. |
| Scalability | The system should handle a minimum of 10,000 concurrent users without significant performance degradation and not reach its bandwidth limit. |
| Robustness | The system can handle any unexpected errors during operating, such as user inputs, hardware failures, or external disruptions, and continue to operate reliably. |
| Modifiability | The system should be capable of accepting changes made to the system to 5 years of extension. |
| Compatibility | The system should be accessed through 1 supported mobile application. |
| Consistency | There is no conflict between any set of requirements at the consistency percentage of 95%. |

| | |
|---|---|
| Usability | The system is not using any unfamiliar symbols and using a clear language with the instruction that can easily be understandable by users under 10 minutes of learning. |
| Flexibility | Ability to work uniformly across multiple regions and cultures as it provides 1 universal language which is English. |
| Reliability | Copyright laws and licence agreements must be respected for any third party software used in the creation of this system. |
| Efficiency | The estimated time of arrival functionality's accuracy should not be more than 2 minutes. |
| | The system's load time should not be more than 1 minute. |
| Manageable | The visual representation of the location should be as accurate as possible within a 500-metre radius of the actual location. |

| **Constraint** | **Architecture Requirement** |
|---|---|
| Development | The application uses technology from the existing GPS system to track and locate the nearest Food Bank from user location. |
| | The database management system (DBMS) used to store data is MySQL. |
| | Flutter is used to develop the system. |
| Operating environment | The supported operating system is AndroidOS and iOS. |
| Design and Implementation | The information of all user's items in inventory has to be stored in a database which can be accessed by the system. |

## 4.2 Architecture Styles and Patterns

Briefly, Architecture Styles provide high-level principles for structuring and organising a system, influencing its overall behaviour and characteristics. Examples include monolithic architecture, microservices architecture, layered architecture, and event-driven architecture. Meanwhile, Patterns offer reusable solutions to common design problems at various levels of granularity, from small-scale coding challenges to high-level architectural decisions. These patterns encapsulate best practices, promote modularity, and provide a common language for communication among developers.

### 4.2.1 Client-Server Architecture Pattern

This architecture facilitates the communication between the client and the server, which may or may not be under the same network. A client requests specific resources to be fetched from the server, which might be in the form of data, content, services, files and more. The server identifies the requests made and responds to the client appropriately by sending over the requested resources. The Client-Server pattern allows for centralised security measures. Access control, authentication, and encryption can be implemented on the server side, reducing the complexity of securing individual client applications. Furthermore, software updates, patches, and maintenance can be performed centrally on the server. Clients only need to update their software when there are changes on the server side, simplifying the maintenance process.

FoWRA application adopts a Client-Server Architecture Pattern to efficiently manage its features and functionalities. In this architecture, the client, which is the mobile application installed on users' devices, communicates with a central server. The server is responsible for handling various tasks such as managing user data, processing inventory information, generating recipe suggestions, and facilitating communication between users. When a user scans a food product barcode or updates their inventory, the client sends requests to the server, which processes the information and responds accordingly. The server, in turn, maintains the database of user profiles, food items, and community interactions. This architecture allows for seamless integration of features like inventory management, expiration date alerts, and recipe suggestions, ensuring that users can access the app's functionalities while keeping their data and interactions centralised for efficient processing and coordination. Additionally, the client-server pattern facilitates the donation process, as user-submitted forms are processed centrally, and designated food bank centres are informed to pick up donations, enhancing the overall effectiveness and responsiveness of the FoWRA application.

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

This architecture pattern separates into three distinct layers:

1. **Model.** The model layer is responsible for the application's data logic and storing and retrieving data from back-end data stores. The model layer might also include mechanisms for validating data and carrying out other data-related tasks. This layer is responsible for maintaining all aspects of the data and ensuring its integrity and accessibility.

2. **View.** The view layer provides the UI necessary to interact with the application. It includes components needed to display the data and enables users to interact with that data. For example, the view layer might include buttons, links, tables, drop-down lists or text boxes.

3. **Controller.** The controller layer contains the application logic necessary to facilitate communications across the application, acting as an interface between the view and model layers. The controller is sometimes viewed as the brains of the application, keeping everything moving and in sync.

MVC architecture pattern and style provides several advantages such as maintainability where the separation of concerns and modularity in MVC makes it easier to maintain and update the FoWRA application. This makes it simpler to add new features or modify existing ones. Besides, MVC supports scalability by allowing different components to scale independently.

The FoWRA application employs the Model-View-Controller (MVC) Architecture Pattern to organise and streamline its structure. In the MVC pattern, the Model represents the application's data and business logic, including the inventory management system, expiration date tracking, and user profiles. The View handles the presentation layer, displaying information to users, such as the inventory, recipe suggestions, and community interactions. The Controller acts as the intermediary that manages user inputs, processes them, and updates both the Model and the View accordingly. For example, when a user scans a barcode, the Controller triggers the Model to update the inventory and informs the view to display relevant expiration alerts. This architectural pattern ensures a clear separation of concerns, making FoWRA modular and scalable. It allows for easy modification of one component without affecting the others, promoting maintainability and flexibility. The MVC pattern is particularly beneficial for FoWRA, as it accommodates the diverse functionalities, from inventory management to community interaction, creating a well-structured and efficient foundation for the development of the food waste reduction app.

### 4.2.3 Publish-Subscribe Architecture Pattern

The Publish-Subscribe architectural pattern, also known as the Pub-Sub pattern, is a messaging pattern where components (subscribers) express interest in certain events or messages, and publishers send those messages without knowing the specific subscribers. This pattern promotes loose coupling between components in a system and is commonly used to implement event handling systems, message queues, and communication between distributed systems. The pattern provides flexibility in terms of adding or removing subscribers without affecting other components. Subscribers can dynamically subscribe or unsubscribe from topics based on their requirements.

The Publish-Subscribe Architecture Pattern is aptly suited for the FoWRA application, providing an effective means of communication and data distribution among its various components. In this architecture, FoWRA acts as a publisher that generates events or notifications, such as approaching expiration dates or newly added inventory items. Subscribers, representing different modules or features within the app, express interest in specific types of events. For instance, users subscribing to inventory notifications or recipe

suggestions will receive updates relevant to their preferences. This pattern facilitates seamless communication between components without direct dependencies, ensuring a flexible and scalable design for FoWRA. For example, when a user scans a barcode to update their inventory, the inventory management module can publish an event that triggers notifications or recipe suggestions. Likewise, when a user donates food to a nearby food bank, the donation module can publish an event, notifying relevant users or updating community activity feeds. This approach enhances modularity, simplifies maintenance, and allows FoWRA to dynamically adapt to changing requirements while efficiently delivering personalised information to users based on their specific interests and actions within the app.

### 4.2.4 Security Architecture Pattern

Security Architecture Patterns are structured frameworks designed to provide a comprehensive and cohesive approach to safeguarding information systems and data. These patterns define a set of principles, best practices, and reusable components that organisations can adopt to create a robust security posture. Typically, they address key aspects such as authentication, authorization, encryption, monitoring, and incident response. By employing Security Architecture Patterns, organisations can systematically integrate security measures into their IT infrastructure, applications, and processes, ensuring a consistent and effective defence against cyber threats. These patterns help establish a foundation for building secure systems by guiding the design and implementation of security controls, thereby reducing vulnerabilities and enhancing overall resilience in the face of evolving cybersecurity challenges.

In the context of FoWRA, implementing robust security architecture patterns is essential to ensure the confidentiality, integrity, and availability of user data. The application must employ authentication and authorization mechanisms to secure user accounts and ensure that only authorised individuals can access sensitive information such as inventory data and personal details. Secure communication protocols, such as HTTPS, should be implemented to protect data in transit between the mobile app and the server. Additionally, incorporating encryption techniques for stored data, especially in the case of user profiles and inventory details, adds an extra layer of protection. Given the nature of FoWRA, where users may share information within their community, a role-based access control system can be established to manage permissions effectively. Regular security audits and monitoring mechanisms should be in place to detect and mitigate potential threats. Moreover, barcode scanning and food donation features need to be carefully designed and tested to prevent malicious activities or unauthorised access. By integrating these security architecture patterns, FoWRA can offer a trustworthy and secure platform for users to manage their food inventory, share information, and contribute to the reduction of food waste without compromising their data security.

## 4.3 Architecture Views

### 4.3.1 Package Diagram

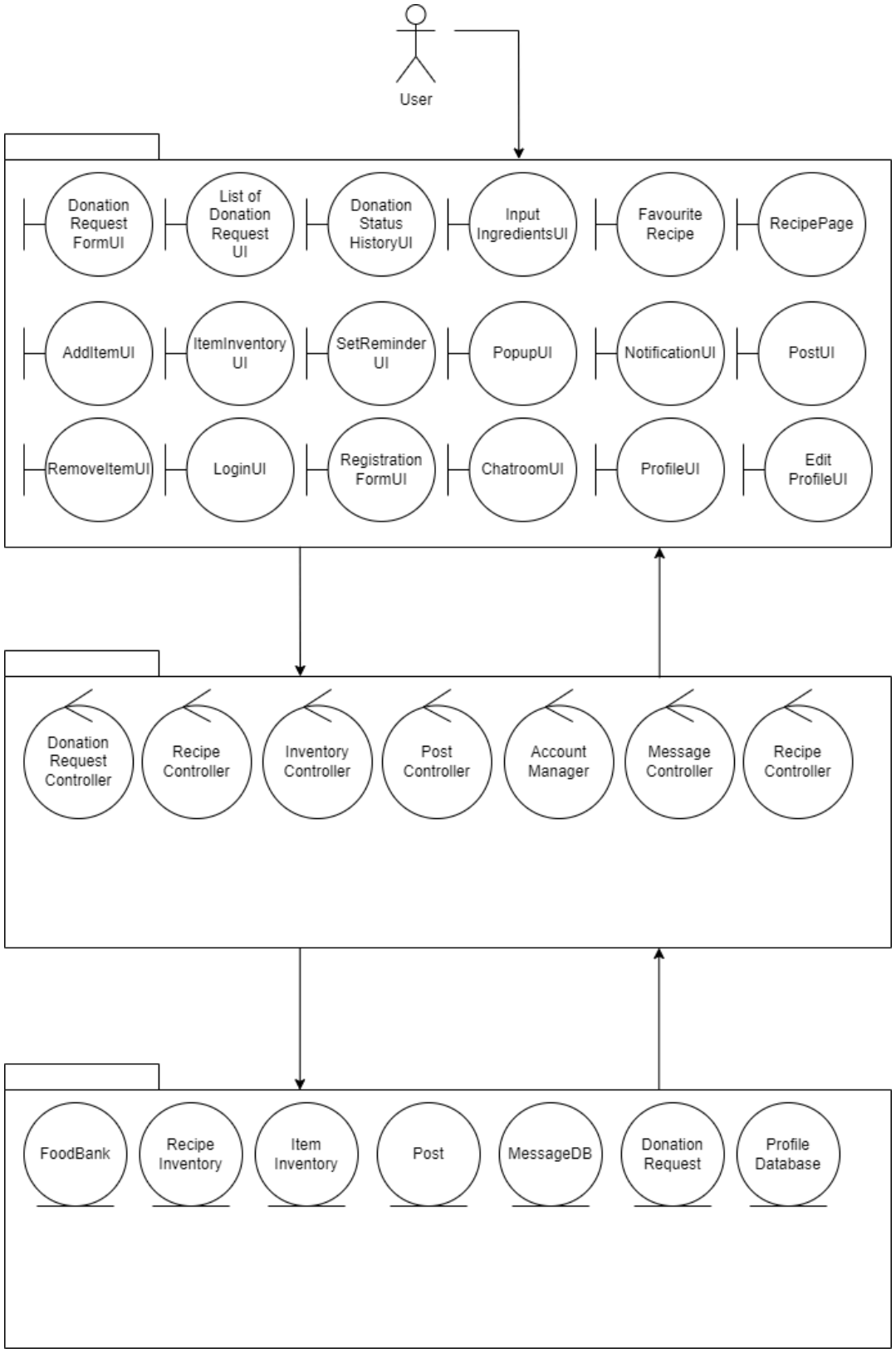



Figure 4.3.1 Package Diagram

Figure above shows the top-down representation illustrates the organisational structure of the system, showcasing how various packages encapsulate and group related components. In the package diagram, it shows the interactions between the packages are depicted through dependencies and associations, providing a comprehensive view of the system's architecture and the relationships between its high-level modules.

***Notation Explanation***

Each overarching system or major functionality is represented by a UML notation. The FoWRA system adopts a three-layered architecture. The presentation layer, represented by boundary objects, facilitates user interaction and information display. The control layer acts as an intermediary, managing workflow and responding to presentation layer requests through control objects that coordinate communication between system objects. These control objects encapsulate specific use case behaviours, ensuring separation from shared behaviours in entity classes. The domain layer focuses on implementing domain rules, featuring entity objects that embody information and functionality within the application domain. Communication and organisation among packages are conveyed through dependencies, associations, and aggregations, showcased in the package diagram. Dependencies on external packages are depicted with arrows, effectively communicating the structural relationships and collaborative dependencies within the system. This design prioritises clarity, encapsulation, and systematic communication to create a well-structured system architecture for FoWRA.

***Element Catalog***

The diagram shows a “UML Notation” view that is connected by 2 types of arrows of three related layers - View Layer, Control Layer and Model Layer.

***View Layer (Presentation)***

It contains eighteen boundary objects related to every FoWRA use case - Donations Request Form UI, List of Donation Request UI, Donation Status History UI, Input Ingredients UI, Favourite Recipe, Recipe Page UI, Add Item UI, Item Inventory UI, Set Reminder UI, Popup UI, Notification UI, Post UI, Remove Item UI, Login UI, Registration Form UI, Chatroom UI, Profile UI and Edit Profile UI.

- Donation Request Form UI allows the users to fill the donation form and all the data will be displayed on Donation Status History UI.
- List of Donation Request UI is basically an interface for admin to view the history of all donation requests.
- Inputs Ingredients UI interact with Favourite Recipe UI where both UI gets and saves the information the ingredients from the user.
- Recipe Page UI will display all the recipes from the databases.
- Add Item UI gives two options for the user to add their new item to the inventory then will be saved to Item Inventory UI.
- Set Reminder UI ensures that the user makes an input for the expiration date then the user will get the notification from the Notification UI and Popup UI.
- Post UI is basically for the user to create and input their media post and data.
- Remove Item UI is used by the user to adjust the quantity or directly remove the item in the inventory.
- Login UI will give the permission for new users to sign up in the FoWRA application.
- Profile UI allows users to enter the profile page and looks for profile data.

- Edit Profile UI represents all the user profile data that can be edited.

***Control Layer (Application Logic)***

It contains seven control objects - Donation Request Controller, Recipe Controller, Inventory Controller, Post Controller, Account Manager, Message Controller and Profile Control.

- User's donation request will be validated by the Donation Request Controller.
- The Recipe Controller will save the ingredients and display it when asked.
- Inventory Controller have various types of use. Basically, it controls the whole system of the item inventory for this application.
- User's posts are conducted by the Post Controller where all the data will be inserted, changed and received.
- Account manager works to validate the details from the user when registration. aa
- User's message will be sent through the Message Controller.
- Profile control will display and update the latest user's profile.

***Model Layer (Domain)***

It provides seven entity objects; FoodBank, Recipe Inventory, Item Inventory, Post, Message DB, Donation Request and Profile Database. Here all the details about the Model Layer(Domain):

- FoodBank object is a database that is directly connected to the admin or company that handles it.
- Recipe Inventory object stored and retrieved all the recipes from all users.
- ItemInventory object will list all the added items that can be adjusted anytime by the user.
- Post object is where all the details about the user's post will be posted and saved.
- All the messages and private messages will be retrieved in the Message DB object.
- The donation's status will be updated through the Donation Request object.
- Profile Database object is to retrieve and store all the user's profile data.

***Relations***

***View Layer (Presentation) with Control Layer (Application Logic)***

The View Layer (Presentation) and Control Layer (Application Logic) in the system are closely connected for seamless user interaction. The View Layer includes user interface components. The Control Layer, in turn, manages these interactions through seven control objects, each handling specific functions. When users interact with the interface, the View Layer triggers actions that are processed by the Control Layer, ensuring the application's smooth operation and maintaining a clear separation of user interface and application logic.

***Control Layer (Application Logic) with Model Layer (Domain)***

The Control Layer (Application Logic) interacts closely with the Model Layer (Domain) to manage and execute the system's functionalities. Seven control objects handle specific application logic, interfacing with seven entity objects. This interaction ensures seamless coordination between user actions, business rules, and data management, creating a well-integrated system where the Control Layer bridges user interface interactions with underlying domain entities.

***Design Rationale***

The FoWRA system utilises a Client-iArchitecture. It showed in the diagrams layered structure, with distinct View (Presentation), Control (Application Logic), and Model (Domain) layers, strongly suggests a client-server approach. Client-server architecture fosters modularity and maintainability by decoupling the user interface (client-side) from the core application logic and data storage (server-side). The scalability also enables independent scaling of the client and server components to accommodate varying workloads and user demands. Next, based on Model-View-Controller(MVC) pattern, the explicit division of the system into View, Control, and Model layers directly aligns with the MVC pattern. It provides flexibility on the system where It allows for independent modification of the UI, application logic, or data model without affecting other components, fostering adaptability. Each component within each layer can also potentially be reused across different functionalities, promoting code efficiency.

### 4.3.2 Component Diagram

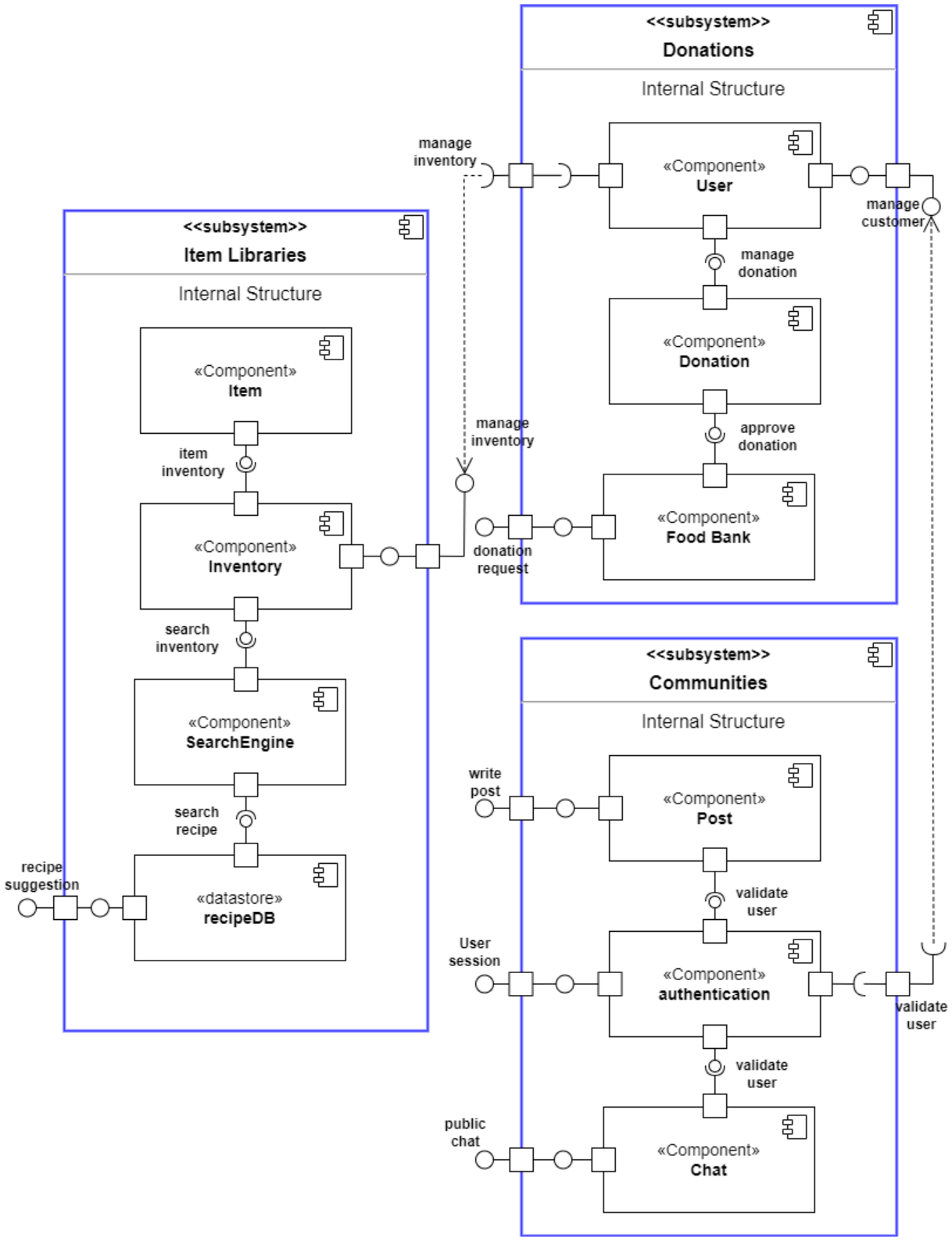



Figure 4.3.2 Component Diagram

Figure above shows a top down description of how FoWRA is expected to work and how components will interact with one another. In the component diagram, It shows the interactions between the components of the system via interfaces.

***Notation Explanation***

Each subsystem is represented by a long rectangle with a blue border. Inside each subsystem, a component is illustrated as a rectangle marked with the keyword <<component>>, signifying its role within that specific subsystem. It's important to note that the included components are not an exhaustive list of entities but rather a representation of the key components within each subsystem. A component defines its behaviour through provided (denoted by a circle and solid line) and required (depicted with a semi-circle and solid line) interfaces, highlighting the dynamic where one component offers a service that another component needs. Ports, symbolised by a small square at the end of a required or provided interface, come into play when a component delegates interfaces to an internal class. While the ball-and-socket notation can offer more detailed insights into the relationship between two components, a dependency arrow suffices to convey the connection effectively.

***Element Catalog***

The diagram shows a “white-box” view of the internal structure of three related subsystems - Item Libraries, Donations and Communities.

***Item Libraries Subsystem***

It contains four components related to FoWRA - Search Engine, Item, Inventory and RecipeDB. Inventory will be provided with the list of items for the item inventory and it relies on a search engine to locate specific items within the inventory. Additionally, the recipeDB requires a search engine to find items within the inventory, facilitating seamless suggestions for recipes.

***Donations Subsystem***

It provides three interfaces - User, Donation and Food Bank. Users will be provided with a donation interface to seamlessly manage their donation. Food Bank admin will be presented with an interface to approve pending requests. This utilises a client-server pattern with user interfaces interacting with a central server managing donations and approvals.

***Communities Subsystem***

It provides three interfaces Post, Chat and Authentication used by other subsystems. Post and chat will require authentication components to validate its content.

***Relations***

***Item Libraries subsystem with Donations Subsystem***

Users in the donations subsystem can manage inventory within item libraries. Inventory depends on the user.

***Donations Subsystem with Communities Subsystem***

Users in the donations subsystem need to be validated by the authentication component. This demonstrates a security pattern to validate users in both the Donations and Communities Subsystems.

### *Design Rationale*

The design rationale for the illustrated subsystems and their interrelationships is rooted in a commitment to modularity, interface clarity, and effective communication patterns. The notation, featuring <<component>> symbols and detailed interface definitions, ensures a nuanced understanding of the internal structure. The Item Libraries Subsystem, with components like Search Engine and RecipeDB, demonstrates a dependency on efficient interfaces, notably the reliance of Inventory on the Search Engine for item location. The Donations Subsystem employs a client-server pattern, enhancing scalability, and features clearly defined User, Donation, and Food Bank interfaces. In the Communities Subsystem, the Post and Chat interfaces integrate with Authentication components, reflecting a security pattern for user validation across both Donations and Communities Subsystems. This design approach prioritises a clear, modular, and secure system, fostering seamless interactions and overall robustness.

### 4.3.3 Deployment Diagram

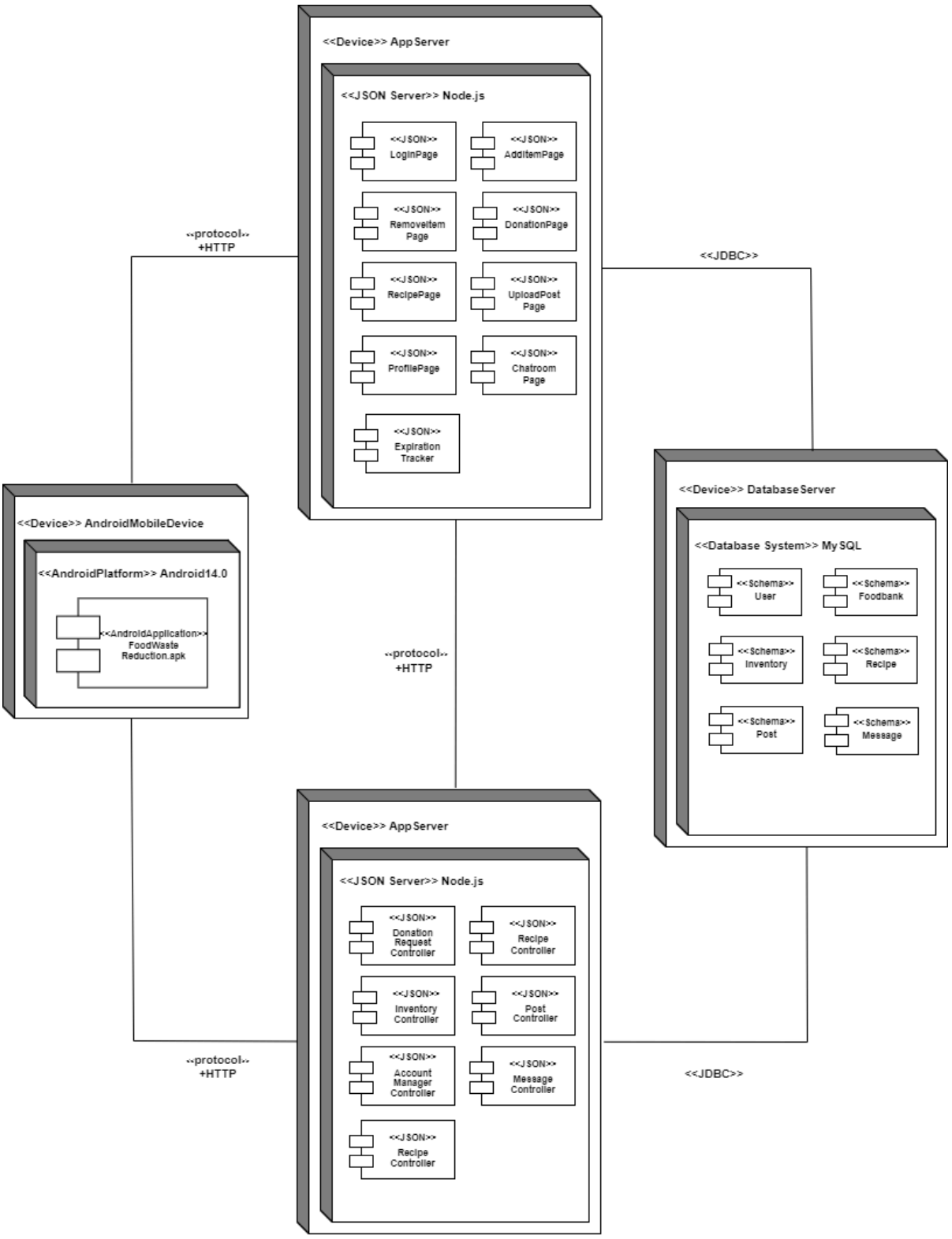



Figure 4.3.3 Deployment Diagram

Figure above shows the execution architecture of a FoWRA application system. In the deployment diagram, It shows the interactions application through the mobile phone, application and the database.

***Notation Explanation***

There are four interconnected nodes: AndroidMobileDevice, 2 App Server, and Database Server. Each node is connected by the communication association. All the nodes have a nested node that represents the details about that specific node. The AndroidMobileDevice node represents a physical device running the Android OS. The App Server node signifies a higher-level entity, an application server, encapsulating its deployment environment. The Database Server node represents a dedicated server for database activities. This notation provides a clear overview of the interconnected hardware and software components, offering insights into their configurations and relationships within the deployment architecture.

***Element Catalog***

The diagram shows four “nodes” that are related with each other through the communication associations - AndroidMobileDevice, App Server and Database Server.

***AndroidMobileDevice node***

It suggests the representation of a physical device running the Android operating system. Within this "AndroidMobileDevice" node, there is a nested node labelled "Android 14.0." This nested node likely signifies the specific version or instance of the Android operating system installed on the mobile device.

***App Server node***

The presence of two nodes named "App Server" signifies a higher-level entity representing an application server within the system architecture. The first node is representing all the user interface that will be provided in the FoWRA’s application. The next one will provide all the controllers that will control the movement of data from user to the application itself. Both of these nodes encapsulate the deployment environment for the application server. Inside the "App Server" nodes, there is a nested node labelled '<<JSON Server>> Node.js'. This nested node likely indicates the specific configuration of the application server, specifying that Node.js is utilised and configured as a JSON server within this deployment environment.

***Database Server node***

The node implies the representation of a dedicated server responsible for managing and storing database-related activities within the system architecture. Within the "Database Server" node, There is a nested node labelled '<<*database system*>> MySQL.' This nested node indicates the specific database system employed within this deployment environment, specifying that MySQL is the chosen database management system.

***Relations***

***AndroidMobileDevice node with App Server node***

The "AndroidMobileDevice" node is connected to the "App Server" node through a communication association labelled <<*protocol*>> + HTTP. This association signifies that the Android mobile device communicates with the application server using the HTTP protocol. This connection indicates that when the Android mobile device interacts with the application hosted on the "App Server" node, it utilises the HTTP protocol for data exchange. This could involve

sending HTTP requests from the Android device to the application server and receiving HTTP responses, facilitating seamless communication between the client-side (Android mobile device) and server-side (App Server) components of the system.

***App Server node with Database Server node***

The "App Server" node and the "Database Server" node, connected by the <<*JDBC*>> stereotype, signifies the interaction between the application server and the database server using the Java Database Connectivity (JDBC) protocol. JDBC is a Java-based API that allows Java applications, such as those running on the application server, to interact with relational databases like MySQL.

***Design Rationale***

In the FoWRA's deployment architecture, the AndroidMobileDevice node functions as the client in a Client-Server model, initiating HTTP requests for data exchange, hosting the View layer (UI) in an MVC paradigm, and potentially subscribing to server-side events for real-time updates. It also implements essential security measures for user authentication and data encryption during communication with the App Server. The App Server node, serving as the central server, processes client requests, manages application logic, and interacts with the Database Server. It likely houses the Controller layer, employs a possible internal publish-subscribe pattern for efficient communication, and implements robust security measures to protect against unauthorised access. The Database Server, operating as a dedicated server for database management, represents the Model layer in the MVC architecture, storing and managing application data while implementing access controls and encryption for data protection. It is crucial to note that the deployment diagram offers a high-level view, and component-level diagrams would provide more in-depth insights into specific interactions and security patterns. Additionally, security patterns are essential for all nodes, with implementation details tailored to specific requirements and threat models.

## 4.4 Implementation

For the FoWRA system, the backend server Node.js is chosen for the development of the mobile application. It processes requests, performs computation, and manages the overall functionality of the system. In the FoWRA mobile application, the relationship between JSON and Node.js is integral to achieving efficient data exchange and server-side functionality. Node.js, serving as the backend technology, facilitates the creation of RESTful APIs for seamless communication between the mobile app and the server. JSON, being a lightweight and human-readable data format, is employed to structure the data exchanged between the FoWRA app and the server. The app utilises JSON for transmitting inventory information, recipe suggestions, barcode-scanned data, and user interactions within the community. Node.js, with its asynchronous capabilities, handles the processing of requests, database interactions, and the formulation of JSON-formatted responses. This combination ensures a robust and scalable architecture, enabling features such as inventory management, expiration date alerts, recipe suggestions, community interactions, and food donation functionalities, thereby contributing to the overall success of FoWRA in reducing food waste and promoting sustainable practices.

Flutter, as the chosen framework for the FoWRA (Food Waste Reduction App) project, plays a pivotal role in achieving cross-platform development efficiency. Leveraging Dart as its programming language, Flutter enables the creation of a single codebase for iOS, Android, web, and desktop platforms. The widget-based architecture of Flutter facilitates the development of a visually cohesive and responsive user interface, crucial for FoWRA's features like inventory management, barcode scanning, and recipe suggestions. The framework's hot reload feature expedites development iterations, enhancing the efficiency of implementing functionalities such as expiration date notifications and community interactions. With Flutter's commitment to native performance and a rich ecosystem, FoWRA benefits from a unified and expressive development environment, empowering the creation of a feature-rich, visually appealing, and cross-platform food waste reduction application.

For the database server, MySQL is chosen to store and manage all the information. The reason is that mySQL is an open-source relational database management system (RDBMS), which means it is freely available for use, modification, and distribution. This open nature encourages a large community of developers to contribute and improve the software. Furthermore, MySQL provides various security features, including access controls, user authentication, and encryption, to protect data from unauthorised access. Regular security updates are released to address potential vulnerabilities.

# 5.0 Architecture Analysis

Understanding how our app behaves under different conditions is crucial for ensuring its resilience, reliability, and user satisfaction. This part outlines a comprehensive scenario analysis, exploring potential user interactions, performance fluctuations, and security vulnerabilities.This document is structured to present various scenarios, their potential errors and risks, and mitigation strategies. By following this roadmap, we can develop a comprehensive understanding of the app's potential challenges and ensure its successful implementation.

## 5.1 Scenario Analysis

**Register User Scenario**

The registration process is a critical entry point for users, making it essential to anticipate and address potential errors to ensure a smooth and secure experience. Here's breakdown of common errors and mitigation strategies:

1. **Missing Fields in Registration Form:**
   *Error*: This happens because Users may inadvertently omit required information.
   *Mitigation*: client-side validation was implemented to prevent form submission until all fields are complete.The app will display clear error messages that identify missing fields.

2. **Invalid Email Format:**
   *Error*: Users may enter an email address that doesn't adhere to valid syntax.
   *Mitigation*: Validate email format using regular expressions or built-in functions.The app will provide specific error messages indicating invalid email syntax.

3. **Weak Password:**
   *Error*: Users may choose easily guessable or compromised passwords.
   *Mitigation*: Enforce strong password policies requiring a minimum length, complexity and no common patterns.The app displays password strength indicators to guide users. We also consider implementing password expiration and reuse prevention.

4. **Database Connection Failure:**
   *Error*: Communication with the database may be interrupted due to technical issues.
   *Mitigation*: Implement retry logic with exponential backoff to handle transient errors.The app will provide informative error messages to users, indicating temporary unavailability.

5. **Duplicate Email Account:**
   *Error*: Users may attempt to register with an already existing email address.
   *Mitigation*: The database system will check for existing accounts during registration. The app will display clear messages indicating the email is already in use and will offer options for password recovery or account merging if appropriate.

6. **Network Issues:**
   *Error*: Connectivity problems may disrupt communication between app and server.
   *Mitigation*: Implement robust error handling for network failures.The app will display messages indicating network issues and suggest actions like retrying or checking connections.

**Update Profile Scenario:**
Updating user profiles offers flexibility and personalization, but also introduces potential errors that can impact user experience and data integrity. Here's a breakdown of some key aspects:

1. **Invalid Data Input:**
   *Error*: Users may accidentally enter incorrect information like name, email addresses, and password.
   *Mitigation*: Implement data validation on both client and server sides to ensure correct data formats and types.The app will display clear error messages indicating invalid input and suggesting corrections.

2. **Data Synchronisation Errors:**
   *Error*: Changes made on one device may not be reflected immediately on other devices.
   *Mitigation*: Implement real-time or near-real-time data synchronisation mechanisms. The app will inform users about potential delays in data updates across devices.

3. **Privacy Settings Issues:**
   *Error*: Users may unintentionally make their profile data publicly visible or encounter difficulty managing privacy settings.
   *Mitigation*: Design intuitive privacy settings interfaces.The UI will provide clear explanations of different privacy options. The user will have granular control over what information is visible to others.

4. **Database Errors:**
   Error: Technical issues may prevent profile updates from being saved or retrieved correctly.
   Mitigation: Implement robust database error handling to handle database failures gracefully. The app will display informative error messages to users indicating temporary unavailability.

5. **Security Risks:**
   *Error*: Malicious actors might attempt to exploit vulnerabilities during profile updates to access sensitive data.
   *Mitigation*: Secure API endpoints and user data access. The system is Implemented with strong authentication and authorization mechanisms. The security also will Perform regular security audits and vulnerability assessments.

**Add and Remove Item Scenario:**
Updating user profiles offers flexibility and personalization, but also introduces potential errors that can impact user experience and data integrity. Here's a breakdown of some key aspects**:**

1. **Invalid Item Information:**
   *Error*: Users may accidentally enter incorrect information like name quantities, or expiration dates.
   *Mitigation*: Implement robust input validation to ensure consistency and accuracy. The Ui will provide clear guidance on required fields and acceptable data formats.

2. **Quantity Tracking Errors:**
   *Error*: Inaccurate tracking of item quantities could lead to miscalculations and potential food waste.
   *Mitigation*: We implemented clear and intuitive user interfaces for quantity input and adjustment.The UI will provide visual indicators of item quantities, such as progress bars or charts. The system also gives alerts or notifications for low-stock items.

3. **Integration with Other Features:**
   *Error*: Adding/removing items might impact other functionalities, such as recipe suggestions or expiration notifications.
   *Mitigation*: Ensure robust communication and data exchange between different app components. We also test for potential inconsistencies and update related features accordingly.

**Food Expiration Tracking Scenario:**
Accurately predicting and tracking food expiration is at the heart of your food waste reduction app. However, this critical functionality comes with unique challenges and potential errors that could undermine the app's effectiveness. Here's a breakdown of key concerns:

1. **Incorrect Expiration Date Calculation:**
   *Error*: Algorithms might misinterpret user input, scan dates incorrectly, or apply inaccurate expiration logic.
   *Mitigation*: Implement robust date parsing and validation mechanisms. The app will utilise reliable and well-tested algorithms for expiration calculations. This allows users to manually override or adjust pre-calculated expiration dates for specific items.

2. **Missing Expiration Date Information:**
   *Error*: Users might forget to enter expiration dates for some items, leaving the app with insufficient information.
   *Mitigation*: Make entering expiration dates mandatory for perishable items and Implement default expiration dates based on food category or user-defined preferences (optional).

3. **Notification Failure:**
   *Error*: Users might not receive expiration notifications due to technical issues or device settings.
   *Mitigation*: Employ multiple notification channels like push notifications, in-app alerts, and email reminders.

4. **User Input Errors:**
   *Error*: Users might inadvertently enter incorrect or inconsistent information about expiration dates or quantities.

*Mitigation*: Design user-friendly interfaces for date input and selection. The system will Implement data validation to catch invalid or incompatible values. The UI will allow users to easily review and edit their entries.

5. **Integration with Other Features:**
   *Error*: Expiration dates might not be properly considered in other functionalities like recipe suggestions or inventory syncing.
   *Mitigation*: Account for expiration dates when performing calculations or providing recommendations to ensure seamless data exchange and coordination between different app components.
6. **Algorithm Bias:**
   Error: Expiration prediction algorithms could be biassed depending on training data or assumptions.
   Mitigation: Utilise diverse and comprehensive datasets for algorithm training. Admin will monitor and continuously improve algorithm performance based on user feedback and real-world data.

**Recipe Suggestion Scenario:**
Recipes play a key role in the app, encouraging users to utilise their existing ingredients before they expire. However, inaccurate or irrelevant suggestions can frustrate users and undermine the app's effectiveness. Let's explore potential errors and mitigation strategies for this crucial feature:

1. **Inaccurate Ingredient Recognition:**
   *Error*: The app misidentifies or misinterprets user-entered ingredients, leading to incorrect recipe suggestions.
   *Mitigation*: Implement user input parsing algorithms and allow users to manually edit or confirm identified ingredients. The database system will also have common substitutes for potential recognition errors.

2. **Irrelevant or Incompatible Recipe Suggestions:**
   *Error*: The app suggests recipes that don't fit users' preferences, dietary restrictions, or available cooking equipment.
   *Mitigation*: Allow users to filter or prioritise recipe suggestions based on their preferences.

3. **Algorithm Bias:**
   *Error*: The algorithm might favour certain types of recipes or ingredients, leading to limited options for users.
   *Mitigation*: Utilise diverse and comprehensive recipe datasets for algorithm training. The app will Incorporate user feedback and engagement data to refine the algorithm over time.The system will offer a variety of recipe filtering options and personalization mechanisms.

4. **Technical Issues:**
   *Error*: Server errors or connectivity problems might prevent the app from displaying or generating recipes correctly.

*Mitigation*: Implement robust error handling and retry logic for server-side issues.Provide offline access to curated recipe lists or allow users to save favourite recipes for later viewing. The app will Inform users about temporary connectivity issues and offer retry options.

**Public Chat Scenario:**
Public chat facilitates user interaction and knowledge sharing, but also presents unique challenges related to content moderation, user safety, and technical issues. Let's analyse potential errors and mitigation strategies:

1. **Inappropriate or Offensive Content:**
   *Error*: Users may post hate speech, offensive language, or harmful content.
   *Mitigation*: Implement keyword filters and automated spam detection mechanisms. The app also allows users to report inappropriate content. We also Have a clear moderation policy and enforce it consistently.
2. **Spam or Abuse:**
   *Error*: Malicious users may flood the chat with irrelevant messages or engage in abusive behaviour.
   *Mitigation*: Implement rate limiting and anti-spam measures.We also allow users to block other users.we also provide mechanisms for reporting abusive behaviour. The Admin will monitor chat activity and intervene quickly when necessary.

3. **Privacy Violations:**
   *Error*: Users may unintentionally share personal information or violate others' privacy.
   *Mitigation*: Allow users to control what information they share publicly. The system will Implement data security measures to protect user data.

4. **Technical Issues:**
   *Error*: Server outages or connectivity problems may disrupt chat functionality.
   *Mitigation*: The system will be Implemented with robust infrastructure and error handling mechanisms.The app will Inform users about temporary outages and offer retry options.

**Write a Post Scenario:**
Writing a post allows users to share information and engage with the community, but also introduces potential errors related to content relevance, formatting, and technical issues. Let's explore mitigation strategies:

1. **Content Moderation:**
   *Error*: Similar to public chat, posts may contain inappropriate or offensive content.
   *Mitigation*: Implement content review mechanisms before publishing posts. This app allows users to report inappropriate content. The app enforces community guidelines and post removal policies.

2. **Formatting Errors:**
   *Error*: Users may encounter issues with formatting their posts, leading to poor readability or visual inconsistencies.

*Mitigation*: Provide user-friendly tools and options for text formatting.The app also offers pre-configured formatting templates for common post types. The app will Implement automatic formatting correction where feasible.

3. **Technical Issues:**
   *Error*: Similar to public chat, technical issues can prevent users from writing or publishing posts.
   *Mitigation*: Ensure reliable server infrastructure and error handling mechanisms. The app will Inform users about temporary outages and offer retry options. The Admin will Monitor post creation performance and address technical issues promptly.

4. **Spam or Irrelevant Content:**
   *Error*: Users may write overly promotional or irrelevant posts, diluting the value of the community.
   *Mitigation*: Enforce community guidelines against spam and irrelevant content. We will allow users to downvote or flag irrelevant posts to encourage high-quality, community-relevant content creation.

**Donation Request Scenario:**
Facilitating donations is a noble endeavour, but ensuring a smooth and secure experience for both donors and your organisation requires acknowledging potential errors and planning mitigation strategies. Here's a breakdown:

1. **Food Item Selection Issues:**
   *Error*: Users might be unable to find or select the specific food item they want to donate.
   *Mitigation*: Offer a comprehensive and searchable list of food categories and specific items. We also allow users to add custom items if the desired option isn't available.

2. **Foodbank Selection Errors:**
   *Error*: Users might face difficulties choosing the desired food bank or encounter incorrect information.
   *Mitigation*: Display an interactive map or list of available food banks with easily accessible contact information. The app will ensure accurate and updated information about each food bank's location, operating hours, and accepted donation types. The app also allows users to filter food banks based on location, specific needs, or personal preferences.

3. **Donation Quantity Errors:**
   *Error*: Users might enter invalid quantities or encounter limitations with the quantity selection system.
   *Mitigation*: Implement input validation to prevent invalid quantities and provide clear error messages. We also considered setting minimum and maximum donation quantities based on food bank needs and logistical constraints. The app will offer suggestions for appropriate quantities based on chosen food items and foodbank preferences.

4. **Availability and Scheduling Issues:**
   *Error*: Users might try to donate unavailable items or face scheduling conflicts with the chosen foodbank.

*Mitigation*: We will display real-time or near real-time foodbank inventory information. Allow users to schedule donation pickup or drop-off within available time slots. If inventory or schedule conflicts occur, the app will suggest alternative food banks or encourage users to try donating later.

5. **Communication and Coordination Issues:**
   *Error*: Miscommunication or lack of coordination between users and food banks could lead to confusion or missed connections.
   *Mitigation*: The app will provide clear instructions for the donation process, including pickup or drop-off procedures. It also will facilitate communication channels between users and food banks (e.g., phone numbers, emails, chat functionalities). The system will send confirmation notifications for received donation requests and updates on their status.

6. **Technical Issues:**
   *Error*: Server outages or technical glitches could interrupt the donation request process.
   *Mitigation*: Implement robust infrastructure and error handling mechanisms.System will Inform users about temporary outages and suggest retrying later. The app will monitor system performance and address technical issues promptly

## 5.2 Risks

Every app carries potential risks, and the FoWRA app is no exception. While it helps users do good for the planet, it's crucial to anticipate potential bumps in the road to ensure everyone has a safe, positive experience.After analysing every scenario possible, there are some risks related with all of the use cases in the FoWRA app which are categorised as below:

**Security:**
**Data breaches:** User data breaches, including personal information and food preferences, could be compromised by hackers or internal threats.
**Unauthorised access:** Malicious actors might gain access to user accounts or app functionalities, causing harm or disruption.

**Privacy:**
**Misuse of user data**: User data collected for app functionality could be misused for marketing purposes or sold to third parties without proper consent.
**Lack of transparency:** Users might not be fully informed about how their data is being collected, used, and stored.
**Privacy violations:** Inappropriate content in public features like chat could expose personal information or infringe on user privacy.

**Functionality:**
**Errors and bugs:** Unexpected errors or bugs could prevent users from completing tasks or interacting with the app properly, leading to frustration and churn.
**Integration issues:** Different app features might not function well together, causing inconsistencies or inaccurate information.
**Scalability**: The app might not be able to handle large volumes of users or data, leading to performance issues and crashes.

**User Experience:**
**Confusing interface:** A complex or unintuitive interface could discourage users from engaging with the app and hinder its effectiveness.
**Accessibility issues:** The app might not be accessible to users with disabilities, resulting in exclusion and discrimination.
**Lack of engagement:** If the app doesn't offer enough value or incentives, users might lose interest and abandon it.

**Reputational:**
**Negative user reviews:** Poor user experiences or ethical concerns could lead to negative feedback and damage the app's reputation.
**Media attention:** Negative media coverage due to privacy breaches or security vulnerabilities could severely harm the app's public image.
**Loss of trust**: Users might lose trust in the app if its functionalities or data practices are perceived as unreliable or unsafe.

# 6.0 Prototype

```
http: ^1.1.2
```

Figure 6.0.1: Import package

Dependencies need to be imported in the pubspec.yaml file which serves as the configuration file for Flutter. For this case http: ^1.1.2 are imported as the external package FoWRA relies on.

```
ElevatedButton(
  onPressed: () async {
    if (_formKey.currentState!.validate()) {
      // Form data
      var formData = {
        'itemName': itemNameController.text,
        'quantity': quantityController.text,
        'foodBank': foodBankController.text,
        'date': dateController.text,
        'time': selectedTime,
        'address': addressController.text,
      };

      // Send form data to the server
      var response = await http.post(
        Uri.parse('http://10.0.2.2:3000/api/submitForm'),
        body: formData,
      );
```

Figure 6.0.2: Send data to server

The data needs to be sent to the database according to the input entered by the user. Figure above shows an example of the data entered will be passed to its own parameter as the data for the database in form data. Then the form data will be sent to the server according to the specific API endpoint given.

```
// MySQL Database Configuration
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    database: 'fowra',
});
```

Figure 6.0.3: MySQL Database Configuration

Create a connection with MySQL database using the createConnection method where the host, user and database name is specified.These can be configured according to the database in MySQL

## 6.1 Register user

The initial screen of the application functions as the main gateway, presenting users with a login interface. Upon selecting the "Register User" button, individuals are smoothly navigated to a user-friendly registration form. This form incorporates a user-intuitive design, featuring three input dialog boxes for entering name, email, and password details when establishing a new account. After successfully verifying the provided information, the system securely records the user's account data in the database. In cases of validation errors, the application promptly guides users to address and correct the details. A reassuring success message promptly appears, affirming the seamless completion of the registration process.

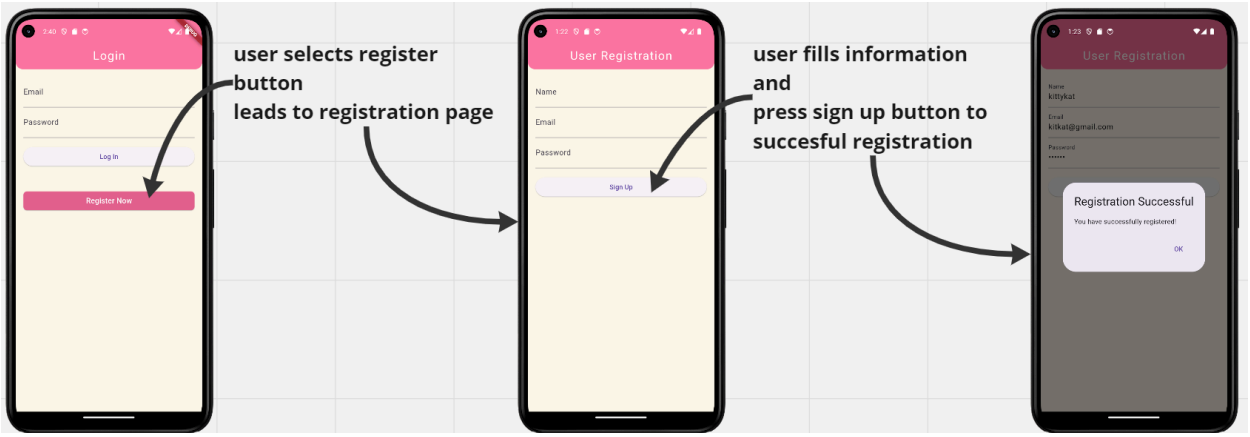



Figure 6.1.1: Register user

```
// API Endpoint for Register User
app.post('/api/registerUser', (req, res) => {
    const { userID, username, password, email, phoneNumber } = req.body;

    const registerSql = `
        INSERT INTO register_user (userID, username, password, email, phoneNumber)
        VALUES (?, ?, ?, ?, ?)
    `;
```

Figure 6.1.2: Register user to the database

The API Endpoint Register User, will extract form data that will have the userID, username, password, email and phoneNumber values. The form data will be inserted into the register_user table  via the SQL query.

## 6.2 Update Profile

Navigation is simplified through the use of a back arrow, allowing users to effortlessly return to the profile page. Additionally, a "Help" button is accessible, offering support by providing information and user guidance.

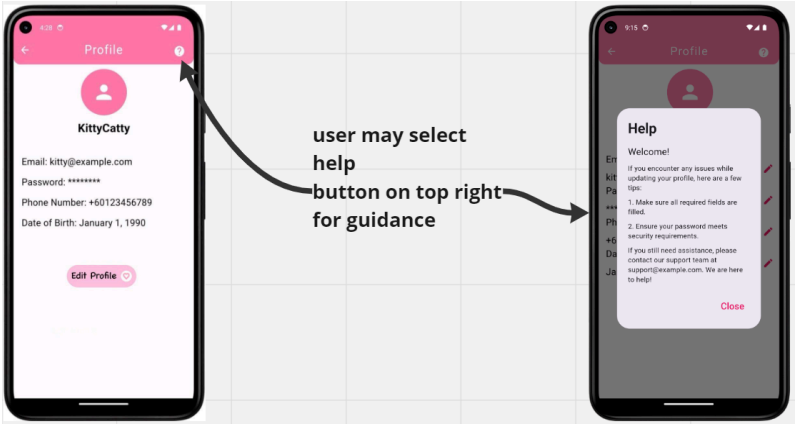



Figure 6.2.1: Update Profile

In the profile section, users have the option to update their details with the "Edit Profile" button. After making adjustments and confirming with the "Done" button, the app undergoes a validation process. A success message promptly appears if the validation is successful, ensuring users of the accurate to their changes.

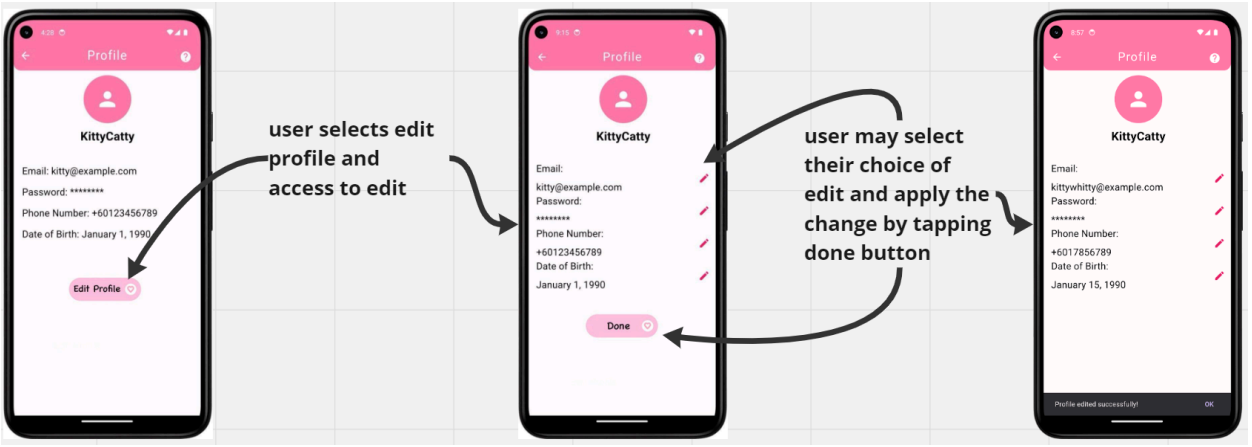



Figure 6.2.2: Update Profile

```
// API Endpoint for update user profile
app.post('/api/updateUserData', (req, res) => {
    const { userId, username, password, email, phoneNumber } = req.body;

    // Update data in MySQL database (replace 'update_user' with your actual table name)
    const updateSql = `
        UPDATE register_user
        SET username = ?, password = ?, email = ?, phoneNumber = ?
        WHERE userId = ?
    `;
```

Figure 6.2.3: Update user profile at database

The API Endpoint Update User Data will extract form data that will have the userID, username, password, email and phoneNumber values. The SQL query will update the register_user table with the new form data.

## 6.3 Home Page

The homepage is designed to be a user-friendly hub for food waste reduction, featuring access to inventory management system, personalised recipe suggestions based on available ingredients, alerts for food expiration, and a community section for interacting and insights on sustainable practices.


Figure 6.3.1: Home Page

## 6.4 Item Management

Users are given the freedom to select their preferred approach when adding items to the inventory: either by manually entering details or opting for a streamlined process through barcode scanning.

### Add item manually

From the main home page, users can navigate to the item management section on the app. Within this section, they can access a dedicated page where they can choose to add a new item by clicking on the "Add your Item" button. If users choose the manual input option, the system guides them to furnish essential information about the item. Furthermore, users are prompted to include a snapshot of the item, which can be conveniently taken in real-time using their phone camera or selected from their gallery. After providing this information, the system smoothly integrates the new item into the inventory list, presenting users with an updated and comprehensive list for their convenience.

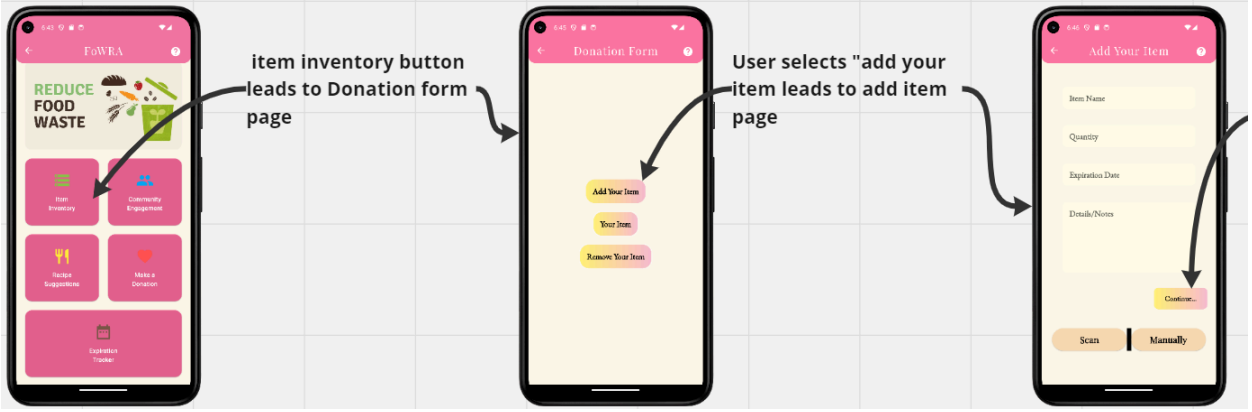



Figure 6.4.1: add item (manually)

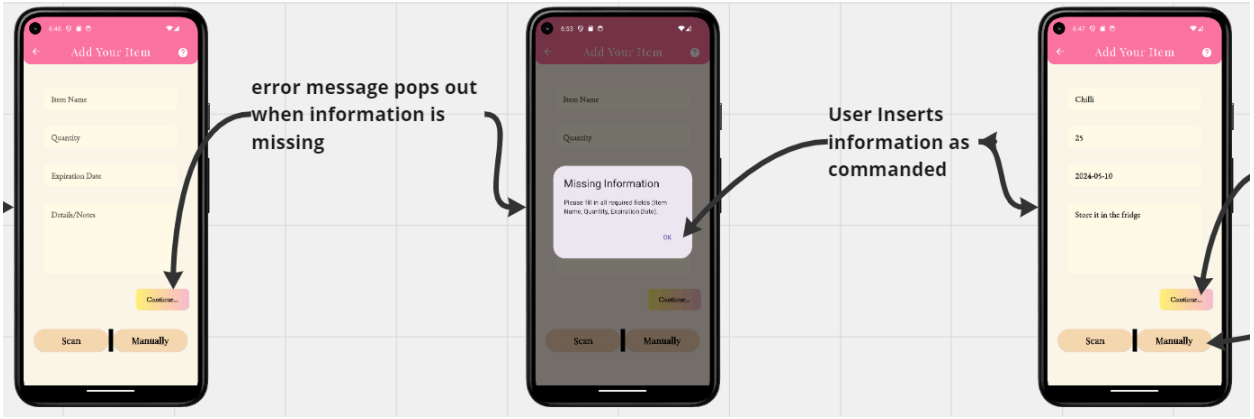



Figure 6.4.2: add item (manually)

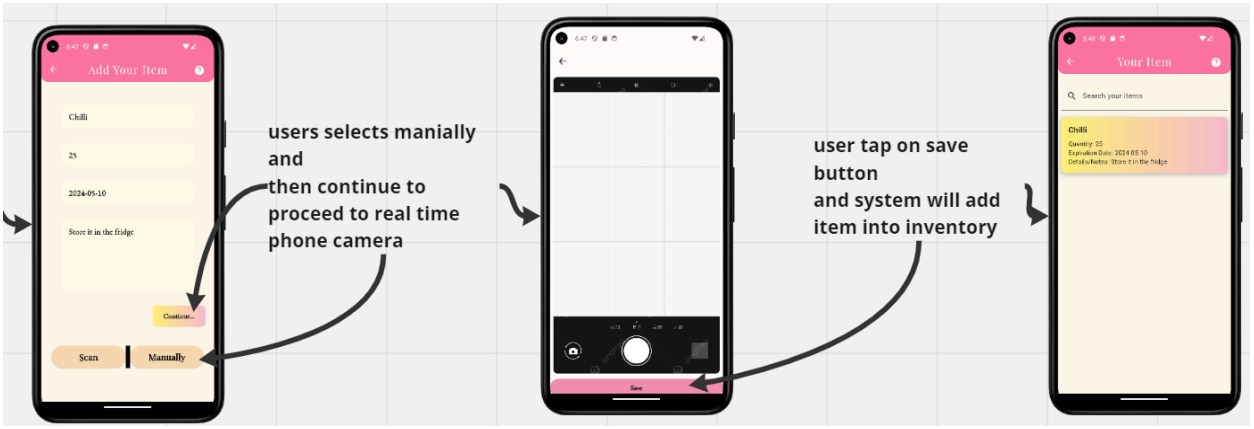



Figure 6.4.3 add item (manually)

## Add item by scan item's barcode

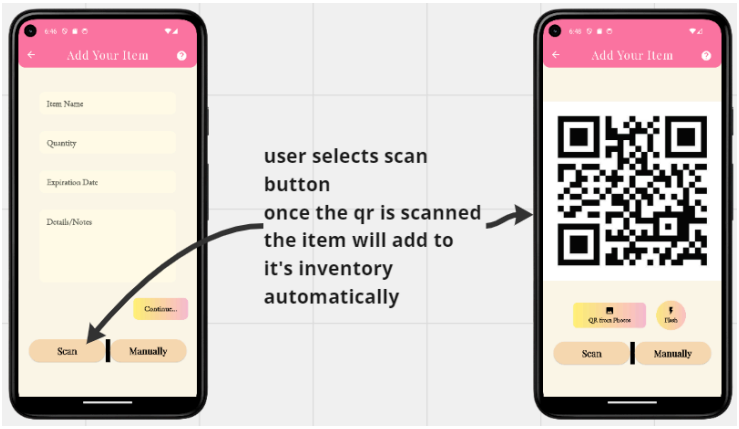



Figure 6.4.4 add item (scan)

```
// API Endpoint for Add Item to Inventory
app.post('/api/addItem', (req, res) => {
    const { itemName, quantity, exp_date, details } = req.body;

    const cameraSql = `
        INSERT INTO itemInventory (itemName, quantity,details)
        VALUES (?, ?, ?)
    `;
```

Figure 6.4.5: Add Item in inventory to database

The API Endpoint Add Item will extract form data which contains itemName, quantity, exp_date and details values. The form data will be inserted into the itemInventory table via the SQL query.

### Check Inventory

When users select the "Scan" option, they can effortlessly employ the dedicated scanning function to capture the barcode information of their item. This streamlined process enables users to swiftly and efficiently incorporate the item into the inventory with ease.

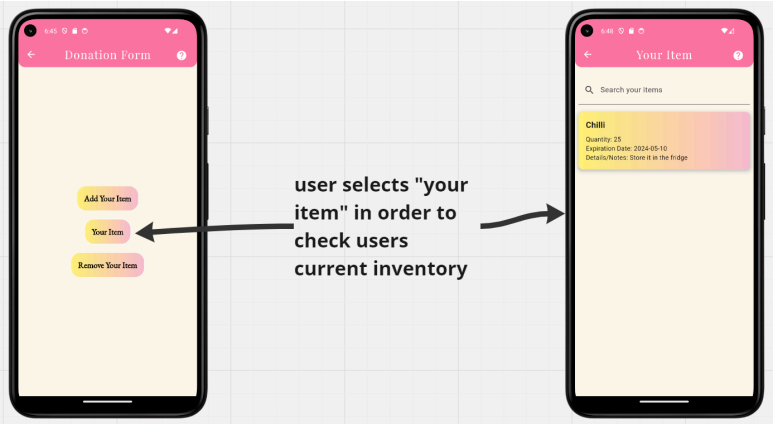



Figure 6.4.6: Inventory List

## Remove item in inventory

The system begins by providing users with a detailed list of the current inventory items. Users can then make selections by choosing specific items and indicating the desired quantity for removal. After confirmation, the system proceeds to update and save the modified inventory list.

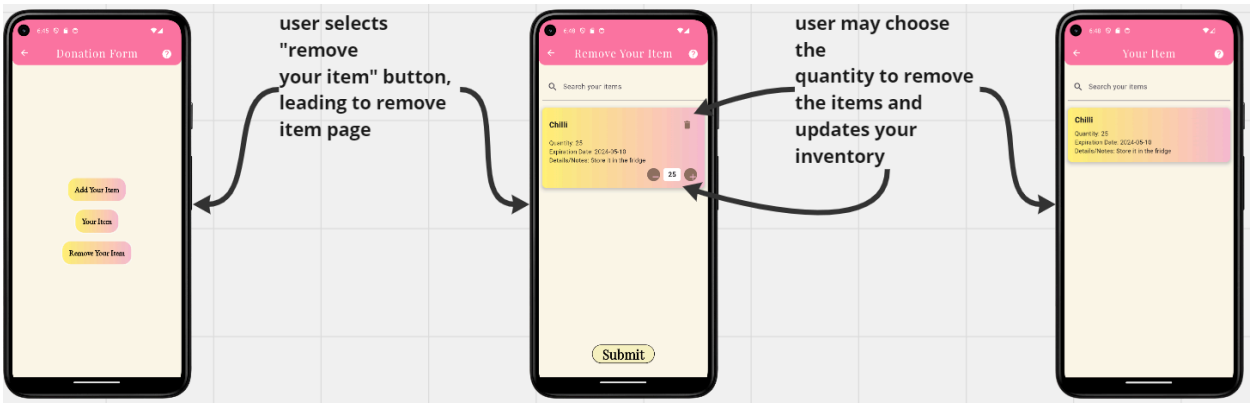



Figure 6.4.7: Remove item in inventory list

```
// API Endpoint for Remove Item to Inventory
app.post('/api/removeItem', (req, res) => {
    const { itemId } = req.body;

    const deleteSql = `
        DELETE FROM itemInventory
        WHERE itemId = ?
    `;
```

Figure 6.4.8: Remove item in inventory to database

The API Endpoint Remove Item will extract form data which contains only itemId value. The SQL query will delete the item from the itemInventory table to remove it from the database.

## 6.5 Food Expiration Tracking

Users are provided with the flexibility to establish expiration dates through manual input or opt for an automated configuration. In the automatic setup, the system extracts expiration dates directly from the inventory. Conversely, in the manual setup, users have the option to input their preferred expiration date. Additionally, users retain control over notifications, managing reminders manually by setting preferences and selecting notification channels (email, SMS, or phone notification bar) in their phone settings. In the case of automatic setup, reminders are automatically generated based on specified expiration dates during the addition of items to the inventory.

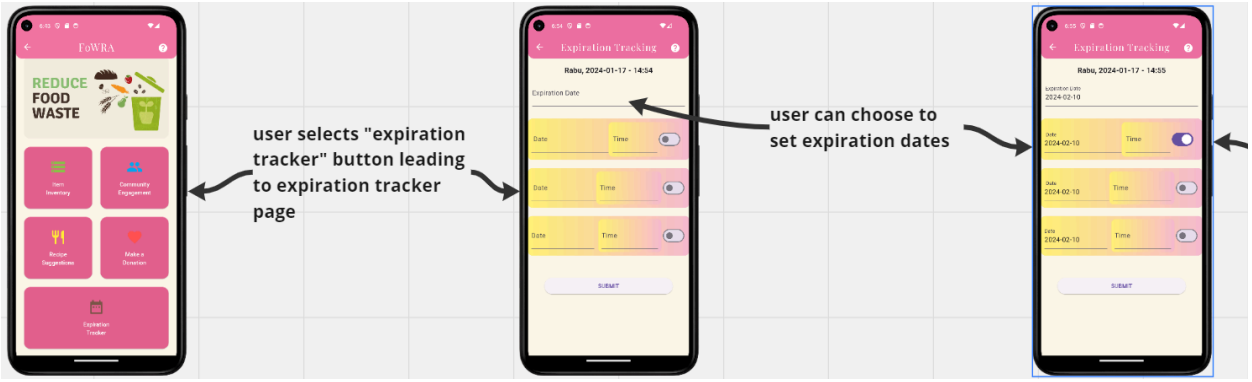



Figure 6.5.1: Set Expiration Tracking

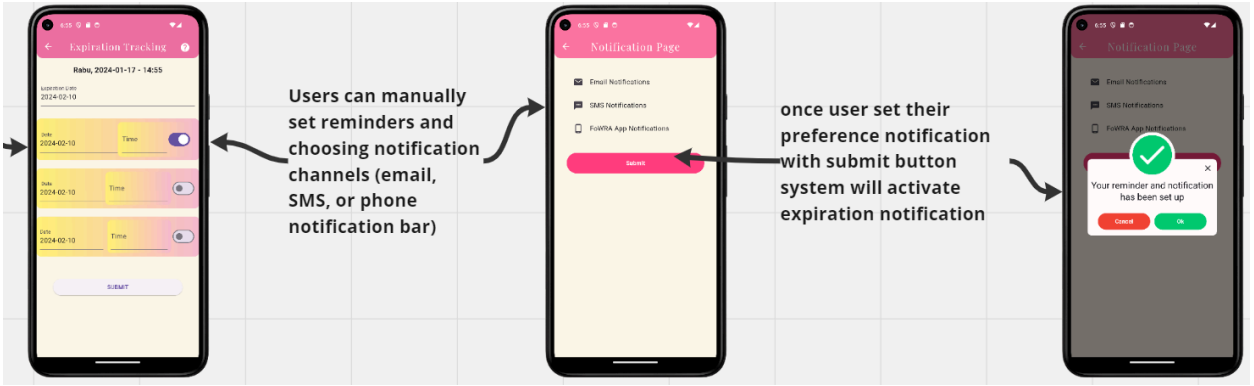



Figure 6.5.2: Notification Options

```
// API Endpoint for set expiration date
app.post('/api/saveData', (req, res) => {
    const { itemName, quantity, exp_date, details } = req.body;

    const insertSql = `
        INSERT INTO itemInventory (itemName, quantity, exp_date, details)
        VALUES (?, ?, ?, ?)
    `;
```

Figure 6.9: Set expiration date in database

The API Endpoint Save Data to Inventory will extract form data which contains itemName, quantity, exp_date and details values. The SQL query will insert these values inside the itemInventory table to save the expiration date into the database.

## 6.6 Recipe Suggestion

Users initiate the process by entering the available ingredients into the system. Following this, the system generates a personalised list of recipe suggestions based on the user's input and the ingredients stored in their inventory. Upon selecting a preferred recipe, the system presents detailed information, ensuring the necessary ingredients, step-by-step instructions, and an estimated cooking time for the chosen recipe.

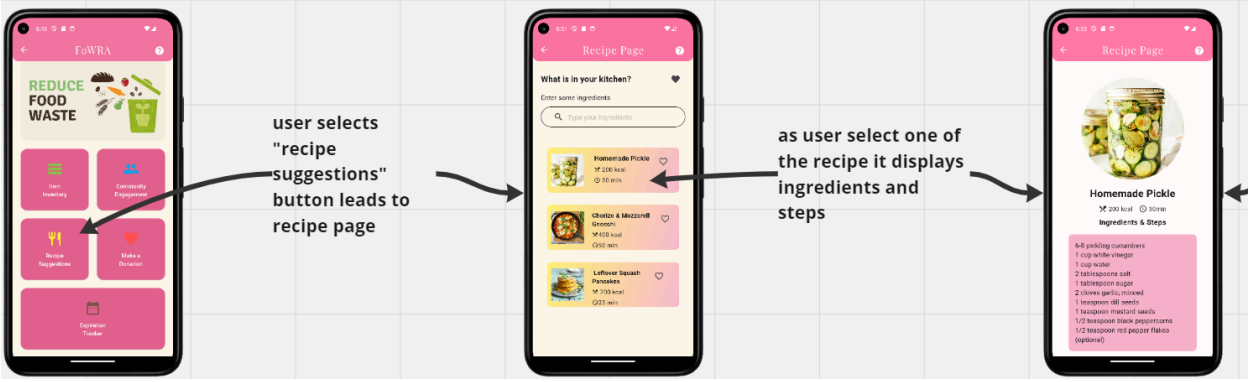



Figure 6.6.1: recipe suggestion

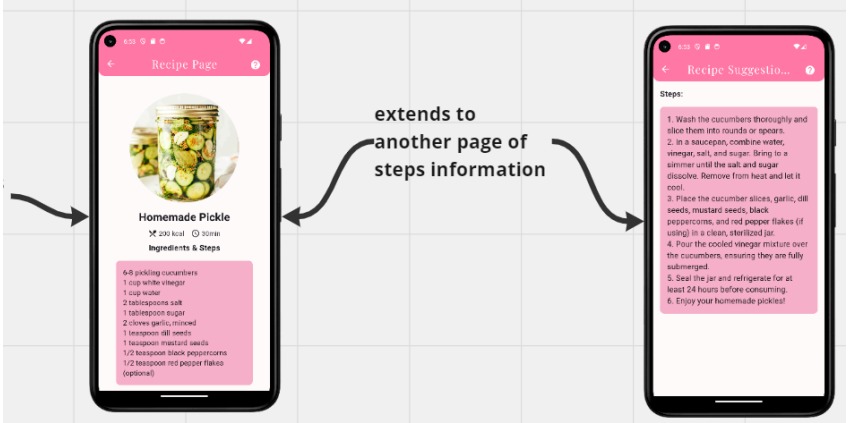



Figure 6.6.2 recipe suggestion

```
// API Endpoint for recipe suggestons
app.post('/api/getRecipeSuggestions', (req, res) => {
    const { itemName, quantity, exp_date, details } = req.body;

    const recipeSql = `
        SELECT recipeName, ingredients, instructions
        FROM recipeDB
        WHERE ingredients LIKE ?
    `;

    const searchQuery = `%${itemName}%`;
```

Figure 6.6.3: Find recipe in the database

The API Endpoint Get Recipe Suggestion will extract form data which contains itemName, quantity, exp_date and details values.The SQL query will then find the recipe related to the ingredients in the database.

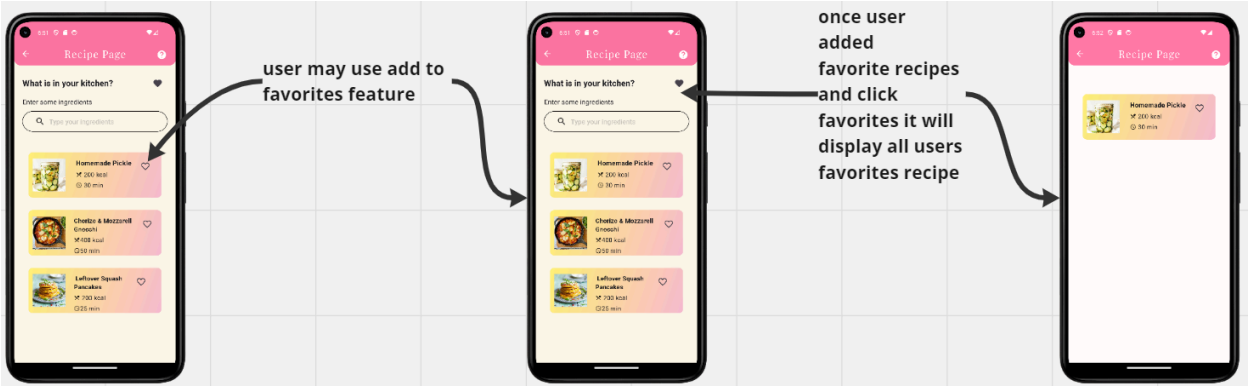



Figure 6.6.4: Save recipe to favourite

Furthermore, users can choose to bookmark their favourite recipes, saving them to a designated favourites page for convenient access in the future.

## 6.7 Community Engagement

### Write a post

Users can initiate their posts seamlessly by entering text into the dedicated field. Additionally, they have the convenience of enriching their content by attaching relevant images and videos. Upon clicking the post button, the application promptly saves and shares the submitted content with the community, displaying it in an engaging format.

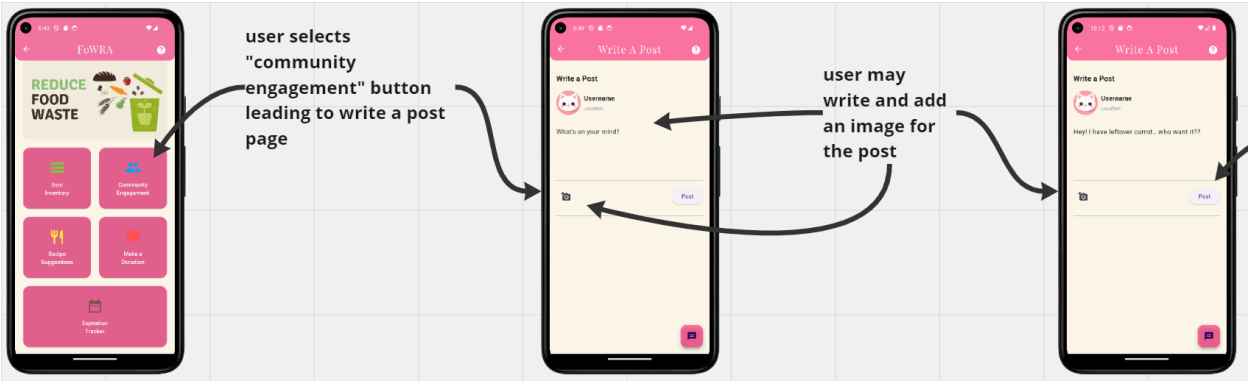



Figure 6.7.1: Write post

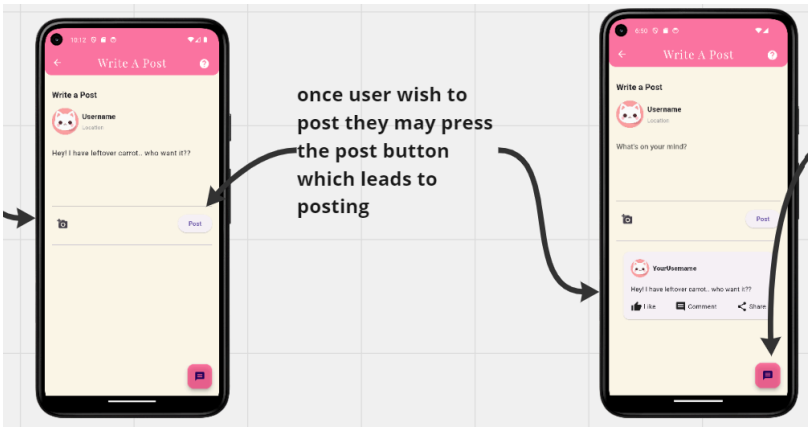



Figure 6.7.2: Write post

```
// API Endpoint for write post
app.post('/api/savePost', (req, res) => {
    const { userID, images, videos, comments, reaction } = req.body;

    const postSql = `
        INSERT INTO Post (userID, images, videos, comments, reaction)
        VALUES (?, ?, ?, ?, ?)
    `;
```

Figure 6.7.3: Save post data to database

The API Endpoint Get Save Post will extract form data which contains userID, images, videos, comments and reaction value. The SQL query will insert these values into the post table to store it in the database.

### Public chat

entering the chat platform, users can smoothly navigate to their chosen chat room or initiate a conversation with a specific user. After making a selection, users have the ability to compose and send messages that will be broadcasted to the chosen chat room

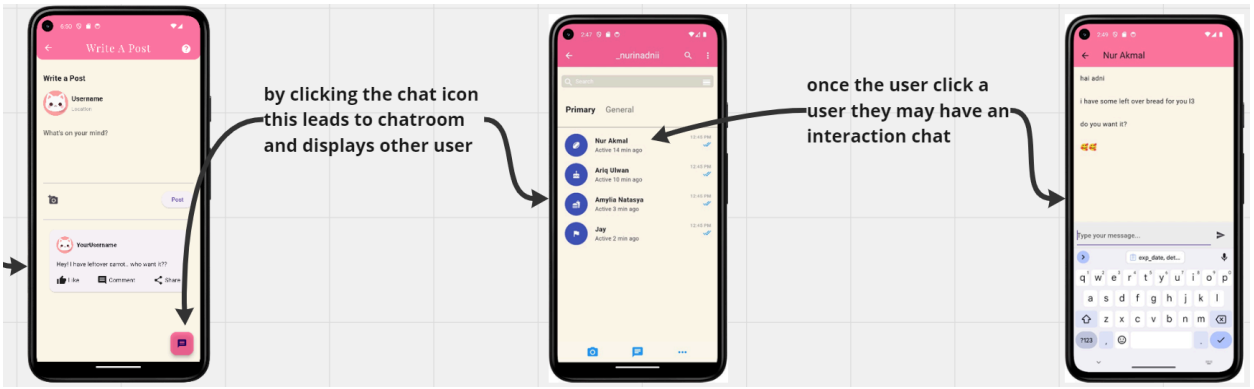



Figure 6.7.4: Public chat

```
// API Endpoint for public chat
app.post('/api/saveChatMessage', (req, res) => {
    const { chatroomID, participants } = req.body;

    const chatSql = `
        INSERT INTO Chatroom (chatroomID, participants)
        VALUES (?, ?)
    `;
```

Figure 6.12: Save chat to database

The API Endpoint Save Chat Message  will extract form data which contains chatroomID and participants values. The MySQL query will insert the values into the Chatroom table to save the chat in the database.

## 6.8 Donation

### Request donation (user)

The application begins by inviting users to input their donation information. Subsequently, users have the option to select a nearby food bank participating in the program, specifying a convenient pickup date and time. Once these choices are made, users can confidently proceed by clicking the "Confirm" button to submit the form. A successfully submitted form triggers the display of a success message.

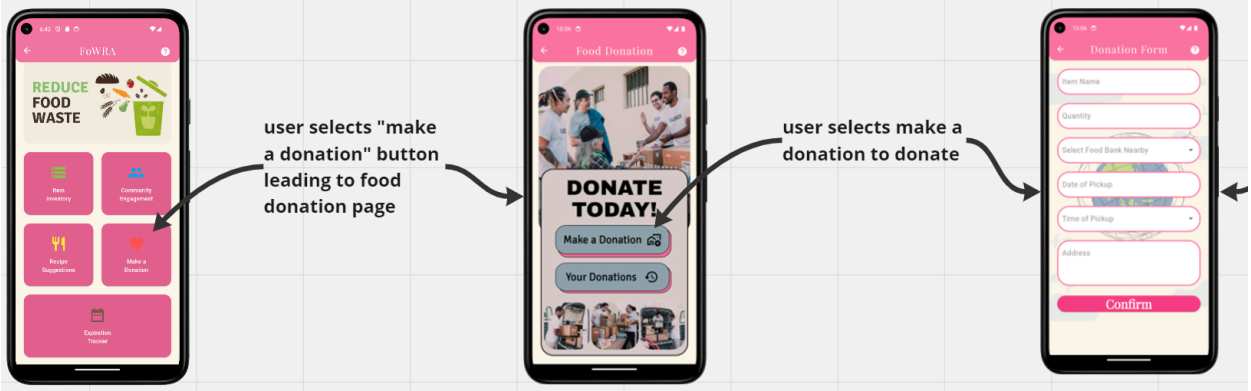



Figure 6.8.1: Donation Request


Figure 6.8.2: Donation Request Successful

In the case of incomplete information or errors, the application responds with an error message, guiding users to re-enter their donation details.

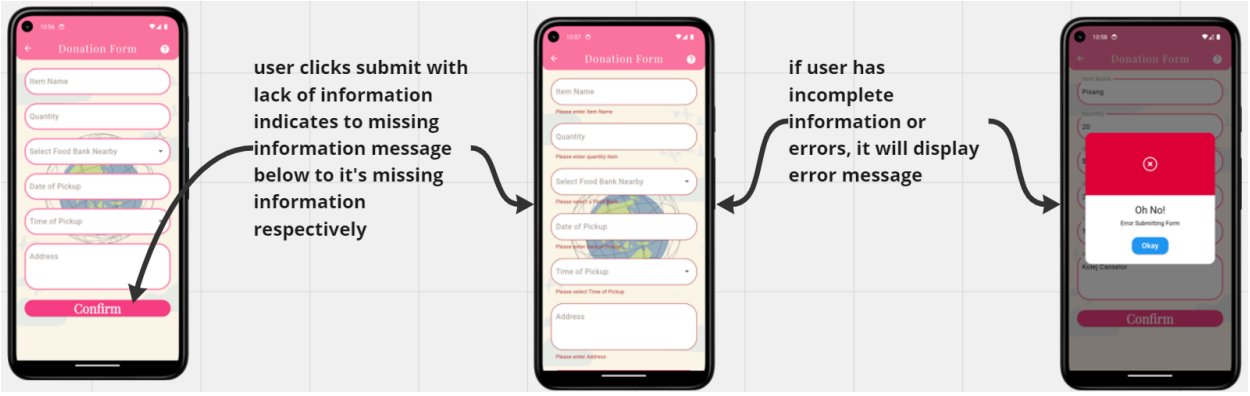



Figure 6.8.3: Donation Request Error

```
// API Endpoint for Form Submission
app.post('/api/submitForm', (req, res) => {
    const { itemName, quantity, foodBank, date, time, address } = req.body;

    // Insert form data into MySQL database
    const sql = `
    INSERT INTO DonationRequest (itemName, quantity, foodBank, pickUpDate, pickUpTime, address)
    VALUES (?, ?, ?, ?, ?, ?)
  `;

    db.query(sql, [itemName, quantity, foodBank, date, time, address], (err, result) => {
        if (err) {
            console.error('Error inserting data into database:', err);
            res.status(500).json({ error: 'Internal Server Error' });
        } else {
            console.log('Form data inserted successfully:', result);
            res.status(200).json({ message: 'Form submitted successfully' });
        }
    });
});
```

Figure 6.8.4: Save donation request to the database

The API Endpoint Submit Form will extract form data which contains itemName, quantity, foodBank, date, time and address values. The form data will be inserted into the DonationRequest table via SQL query.The callback function handles the result of the database query. If there's an error, it logs the error and sends a 500 Internal Server Error response. If the query is successful, it logs the success and sends a 200 OK response.

## Donation History (user)

On the food donation page, users can click the "Your Donations" button to view a list of pending donations and their donation history. Additionally, users have the option to select a specific food bank to display detailed information about its donation history.

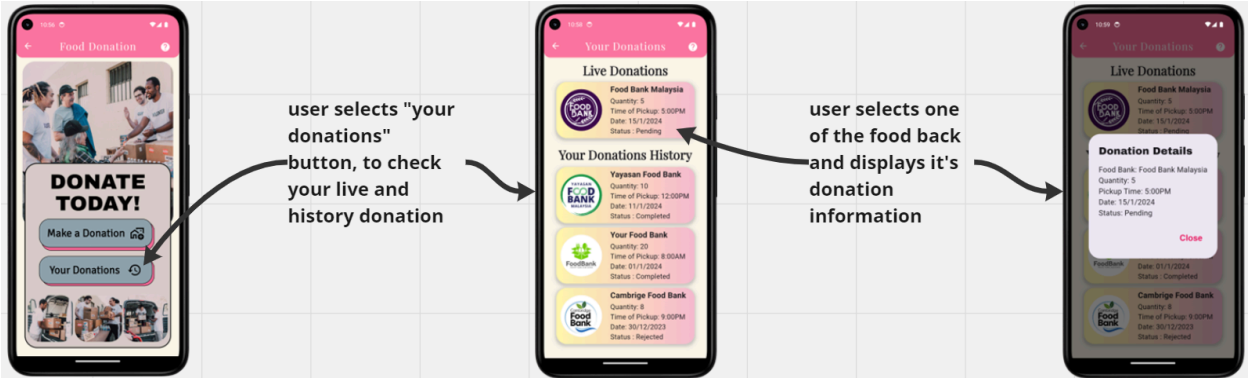



Figure 6.8.5: Donation history

```
// API Endpoint to display donation history
app.get('/api/donationHistory', (req, res) => {
    const query = 'SELECT * FROM donation_forms';
    connection.query(query, (error, results) => {
        if (error) {
            res.status(500).send(error.message);
        } else {
            res.json(results);
        }
    });
});
```

Figure 6.8.6: Display donation history from the database

The API Endpoint Submit Form will execute the SQL query and it will get all the data of donation history from the database.If there's an error, it logs the error and sends 500 Internal Server Error response. If the query is successful, it will display the results.

### Confirm Donation (admin)

The Food Bank admin is presented with a detailed list of pending donation requests awaiting approval. The administrator can effortlessly navigate through the options to either approve or reject each donation request.

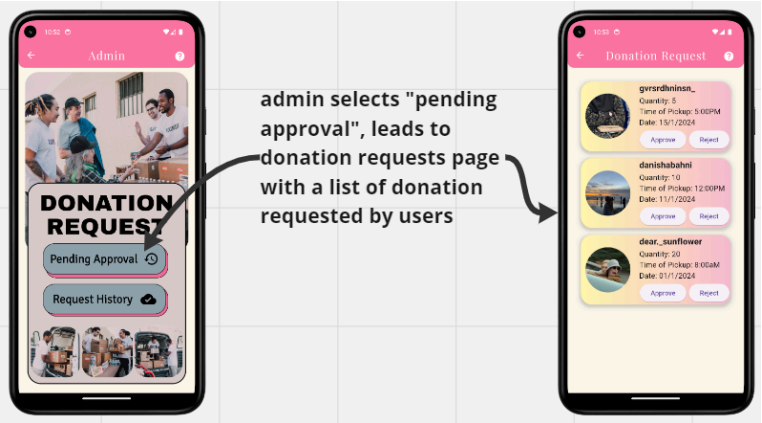



Figure 6.8.7:Confirm donation

Furthermore, the admin has the convenience of accessing and reviewing the complete history of donation requests through the user-friendly interface under the dedicated "Request History" option.

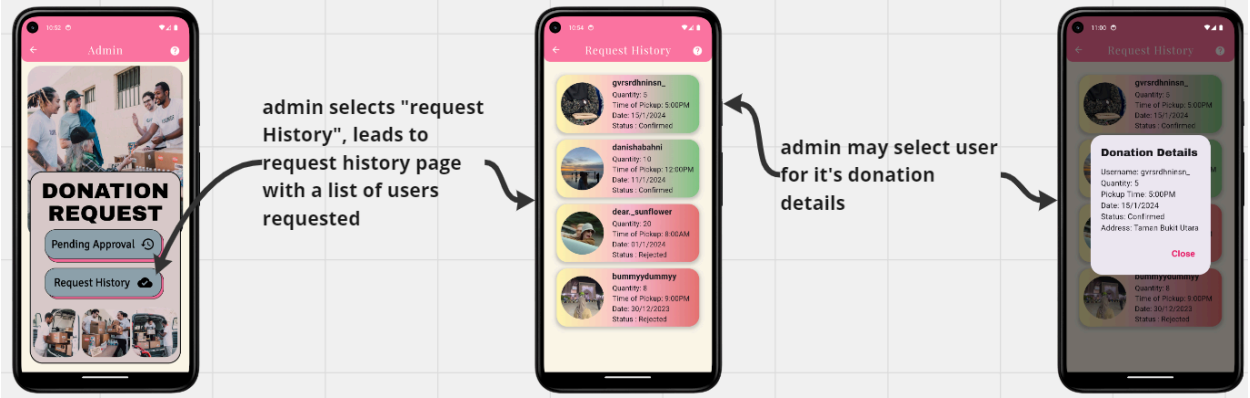



Figure 6.8.8:Confirm donation

```
// API Endpoint to Change Donation Status
app.put('/api/changeStatus/:donationId', (req, res) => {
    const donationId = req.params.donationId;
    const newStatus = req.body.newStatus;
    // Update donation status in MySQL database
    const sql = `
    UPDATE donation_forms
    SET status = ?
    WHERE id = ?
  `;
```

Figure 6.8.9: Update request donation status in the Database

The API Endpoint Change Status will extract form data which contains donationId and newStatus values.The SQL query will update the donation_forms and change the status of the donation in the database.

# Appendix A : Team Contribution Table

| | Tasks | Responsibility | Status |
|---|---|---|---|
| 1. | Class Design | Ariq | Completed |
| 2. | Database Design | Ariq | Completed |
| 3. | Boundary Class | Akmal | Completed |
| 4. | User Interface Design | Akmal | Completed |
| 5. | Architecture Design | Adni | Completed |
| 6. | Prototype | Jay | Completed |
| 7. | Divide task for Software Architecture and Design Document | Amylia | Completed |
| 8. | Documentation | All Member | Completed |
| 9. | Creating prototype using flutter | All member | Completed |
| 10. | Review and Finalise the documentation | Amylia | Completed |