-FOO MY FOOD-

CS 30700
Design Document

Team 29
Xiuwen “Charlie” Fang, Xinyi Guan, Songyu “Hathaway” He, Xinyi Wu

INDEX

Purpose

In modern society, individuals living alone often encounter substantial difficulties when it comes to meal planning and food management. These challenges extend beyond simply deciding what to eat or how to cook; they also involve keeping track of what has been purchased, what remains in the pantry or refrigerator, and how to effectively use these ingredients before they spoil. Despite the availability of numerous applications that address some of these concerns, a significant gap persists in terms of tracking and reminding users about the freshness and expiration dates of the food they already own. This oversight can lead to unnecessary food waste, additional financial burden, and, potentially, health risks associated with consuming expired products.

Our team aims to address this gap by developing a comprehensive application designed specifically to help users efficiently manage their food inventory. The application will allow users to easily record the details of ingredients they have purchased, including purchase dates and expiration dates. By maintaining an up-to-date digital pantry, the app will send timely notifications to users about upcoming expiration dates, enabling them to plan meals and use ingredients before they go bad.

Moreover, the application will offer personalized recipe suggestions based on the ingredients the user already has, reducing the need for frequent grocery trips and promoting the use of what is on hand. Users can select specific ingredients, and the app will recommend diverse recipes tailored to those items. This feature not only helps in meal planning but also encourages culinary creativity and variety, even for those living alone.

In addition to traditional food-tracking and recipe recommendations, the application will feature an innovative recipe creation and sharing platform. Users can easily develop and share their own unique recipes, tailored to the ingredients they have on hand. This personalized approach not only encourages creativity in the kitchen but also allows users to explore new dishes and culinary techniques. By choosing their food intake and using their own recipes, users gain a deeper understanding of their dietary habits and have complete control over the nutritional content of their meals.

Overall, this application will serve as a comprehensive food management tool, addressing the unique challenges faced by individuals living alone. It aims to reduce food waste, simplify meal planning, and promote healthier eating habits through an intuitive and user-friendly platform.

Functional Requirements

A. Account System

As a user:

1. I would like to sign up for a FOO MY FOOD account via email with a unique username on both iOS and Android platform, so that I can easily distinguish my account and access the app through a traditional method.
2. I would like to be able to log in for my FOO MY FOOD account via email address.
3. I would like to be able to log out of my FOO MY FOOD account at any time to ensure the security of my account and to protect personal information.
4. I would like to be able to reset my FOO MY FOOD account's password through a confirmation email with the rest link if I forget it in order to regain access to my account.
5. I would like to set a new password for my account by answering a security question, without waiting for an email.
6. I would like to be able to set my profile information (e.g., username, email address, avatar) for my FOO MY FOOD account, in order to display my basic information.
7. I would like to be able to change my profile to keep my information up to date.
8. I would like to be able to select an avatar from the gallery or just take an avatar photo in order to update my profile in real time.
9. I would like to be able to set my food and cooking style preferences (e.g. dietary restrictions, preferred cuisine) in my FOO MY FOOD account so that the app can recommend personalized recipes and meal plans based on my preferences.
10. I would like to be able to update my food and cooking style preferences at any time (e.g. adjusting my dietary restrictions or adding new preferred dishes) so that the app can continue to recommend personalized recipes and meal plans based on my latest needs.

B. Expiration Tracking and Notification

As a user:

11. I would like to add new items into my inventory by selecting preset ingredient options for quick updates.
12. I would like to be able to manually enter ingredients to add to inventory in order to enter ingredients that are not in the preset options.
13. I would like to be able to see the current inventory of ingredients to have a better idea of what is in my home.
14. I would like to be able to change the names of the ingredients in my inventory in order to update incorrect or outdated entries.

15. I would like to be able to modify the quantity of each ingredient in my inventory to accurately reflect the amount of ingredients available.
16. I would like to be able to delete items from my current inventory shelf for recording the immediate consumption.
17. I would like to set custom expiration dates for my ingredients so that I can take into account personal preferences or storage methods.
18. I would like to set food preservation methods (e.g., freezer, pantry) for ingredients.
19. I would like the system to suggest appropriate saving methods and saving times when adding or modifying ingredient information.
20. I would like to receive an automatically updated food status from the app, calculated with the provided preservation methods and starting datesation dates with my Google calendar so that I can receive notification on my device.
21. I would like to synchronize ingredient expiration information with my Google Calendar so that I can receive notification on my device.
22. I would like to synchronize ingredient expiration dates with my Apple calendar so that I can receive notification on my device.
23. I would like to receive an in-app notice of the expiration information.
24. I would like to receive a notice from the calendar of the expiration information.
25. I would like to receive a preview notice of the expiration information.
26. I would like to receive expiration notifications for ingredients either at a preset or customized number of days before expiration, and at a preset or customized time.
27. I want to be able to filter ingredients based on different categories to make it easier to manage and view specific ingredient categories in my inventory.
28. I want to be able to receive an automatically categorized inventory shelf after each modification.
29. I would like to be able to change the categorization of an ingredient at any time so that I can manually adjust the categorization if the automatic categorization is inaccurate.
30. I would like to be able to filter and sort current inventory by preset or customized “expiration times”.

C. Smart Menu

As a user:

31. I would like to get menu suggestions based on what I have at home so I can plan meals without having to buy additional ingredients.
32. I would like to get menu suggestions based on what ingredients are near the expiration dates so I can avoid food wastes.
33. I would like to record the times of cooking a specific dish with the cooking date.

34. I would like to receive a reminder when checking a dish recipe that has been cooked too often, either within a preset or customized period of time.
35. I want recipe recommendations and meal plans tailored to my current ingredient preferences
36. I want to categorize recipes with labels like ‘Breakfast’, ‘Vegetarian’, etc.
37. I want to be able to create my recipes via text, image or video links and keep accurate records.
38. I would like to be able to edit and delete recipes that I have previously created.
39. I would like to see a calculated nutritional report after selecting specific dish recipes.

D. Shopping List

As a user:

40. I would like to have an in-app list function to track what I am going to buy soon.
41. As a user, I would like to have a categorized search feature in the list to help me keep track of my inventory while shopping.
42. As a user, I would like to be able to get a pop-up notification when adding an existing ingredient on the shelf into the shopping list.

E. User Clustering

As a user:

43. I would like to be able to build up a group chat with other users.
44. I would like to be able to add friends through searching email or by searching the username.
45. I would like to be able to upload my created recipe into a group chat.
46. I would like to be able to hit “like” buttons on recipes posted from others publicly.

F. Other Functions

As a user:

47. I want to track the nutritional value of each item on the shelf and see detailed nutritional info, such as calories, to make healthier meal choices.
48. I would like to have a personal main page showing my public collection folders and other information.
49. I would like to be able to change the theme color from the provided options.

G. Advanced Features (If Time Available)

50. As a user, I want the app to automatically recognize the newly purchased items via scanning receipts.

51. I want receipt scanning to generate default shelf life values for ingredients and paired with the added items in my inventory shelf.
52. I would like to be able to modify the scanned items' names to correct the errors created through scanning.
53. I would like to be able to manually edit the shelf life of each ingredient to ensure accuracy and avoid missing important information when scanning.
54. I would like recurring purchases to automatically adjust their expiration dates based on historical data.
55. I would like to save my favorite recipes as pictures into my device gallery.
56. I would like to be able to collect favorite recipes posted from others publicly.
57. I would like to be able to share the publicly uploaded recipes from others with friends.
58. I want to generate a shareable link for my inventory so friends can view what I have in stock.
59. I want to be able to share links from other websites to share recipes or cooking tutorials.
60. I want to control permissions on shared links, making them either view-only or editable.
61. I would like to be able to set my "like" and "collected" recipe folders into public or private.
62. I would like to be able to set an expiration date for my shared link so that access to my inventory or recipe is automatically revoked after a certain period.
63. I would like to receive an email or in-app confirmation when my link has been successfully shared so that I know the sharing process was completed.
64. I would like to be able to set customized tags for my recipe or in the collection folder.
65. I would like to be able to customize my preferred language in the app.
66. I want to get suggestions for food chosen based on the ingredients I use frequently in order to make healthier choices in my meal planning.

Non-functional Requirements

A. Architecture and Performance

As a developer:

1. I would like _FOO MY FOOD_ to have a fully separated frontend and backend architecture to reduce compatibility issues and improve efficiency.
2. I would like _FOO MY FOOD_ to use Spring Boot for the backend, enabling scalability with features like database caching and ORM support.
3. I would like the frontend to be developed using Flutter, allowing a cross-platform solution for both iOS and Android.

B. Security

As a developer:

4. I would like _FOO MY FOOD_ to implement two-factor authentication (2FA) to enhance security for user accounts during login and other sensitive actions.
5. I would like sensitive data stored in the SQLite database to be encrypted using SQLCipher, ensuring data at rest is secure.
6. I would like to implement role-based access control to limit access to sensitive data to only authorized users.

C. Response Time

As a developer:

7. I would like _FOO MY FOOD_ to complete user actions, such as adding ingredients or searching recipes, within 500 milliseconds to maintain a smooth experience.
8. I would like server-side API calls to return data within 300 milliseconds for simple queries, ensuring quick load times.
9. I would like complex operations to be handled asynchronously, with a response time of no more than 1 second, so the UI remains responsive.
10. I would like the application to scale to handle up to 500 simultaneous requests during peak times while maintaining performance.

D. Usability

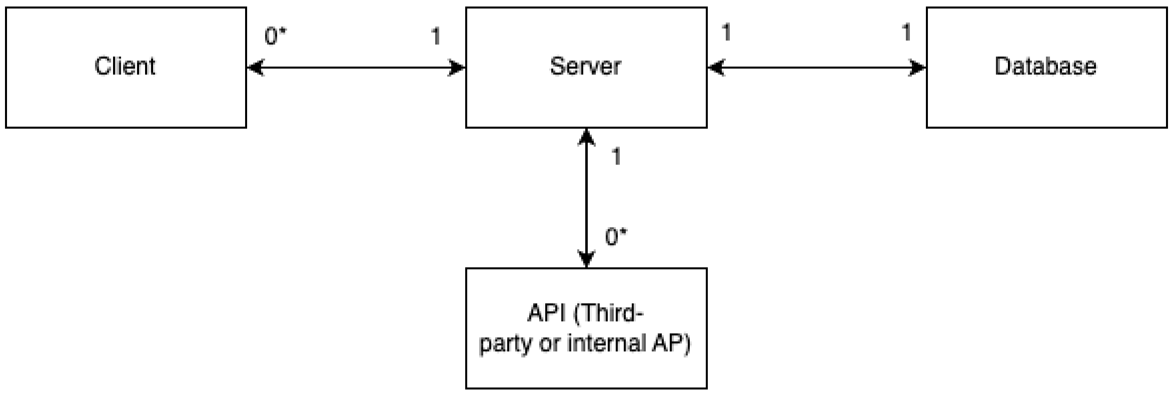
As a developer:

11. I would like to create a visually appealing and functional interface that is easy to navigate for users.
12. I would like the design to balance aesthetics and functionality, making key tasks like searching for recipes or adding ingredients intuitive.
13. I would like smooth transitions and responsive UI elements that enhance the user experience on both Android and iOS.

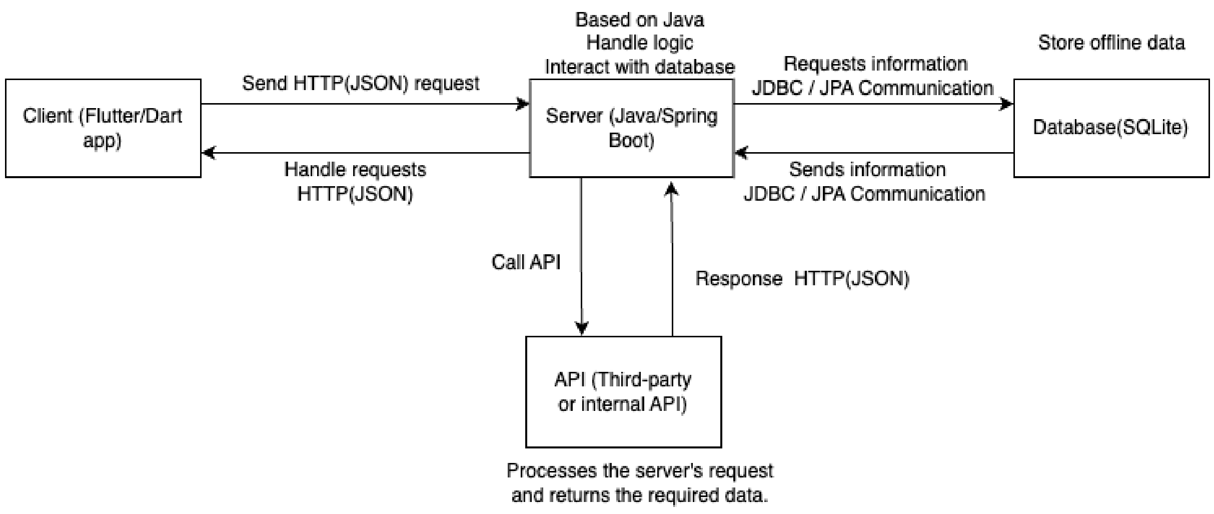
Design Outline

High-Level Overview

This project is a mobile application that helps users manage their food inventory, track ingredient expiration dates, and recommend recipes based on the available ingredients. The system also provides meal planning suggestions, nutritional data, and shopping list management. The application will utilize the client-server model, where the backend server is developed using Spring Boot (Java), and the frontend uses Flutter. The server handles client requests, interacts with the database to store and retrieve data, and provides responses back to the client, ensuring real-time updates on food expiration and personalized meal suggestions.



Detailed Overview



<u>**Clients**</u>:

1. The client provides users with a mobile interface for interacting with their food inventory, expiration notifications, and recipe recommendations.
2. The client sends HTTP requests (via API calls) to the server to fetch or send data related to ingredients, recipes, and meal plans.
3. The client interprets responses from the server and updates the user interface in real time, such as displaying expiration alerts, recipe recommendations, inventory management and personal/account information editing.

Server:

1. The server receives and processes client requests related to food inventory, expiration dates, recipes, shopping lists and account information editing.
2. The server interacts with the database, retrieving, adding, or modifying records based on the client's requests, including managing user preferences and syncing with external calendars for expiration alerts.
3. The server sends the processed data (such as updated inventory, recipe suggestions, or expiration alerts) back to the client in a structured response.

Database:

1. The database stores all user-related data, including food inventory, expiration dates, recipes, user preferences, nutritional data, shopping lists and account information.
2. The database responds to queries from the server, delivering requested data(e.g., upcoming expiration dates, recommended recipes) or updating the database when the user modifies inventory or adds new ingredients.

API:

1. In this project, the API played the role of a bridge between the client and the server, helping the client send requests to the server and get data. At the same time, it also ensured data synchronization and interaction between the client and the database.
    a. user management API:
    The user management API is mainly used to handle user account operations, including user registration, login, update information, and reset password, etc. Through this API, users can use email to create an account, log in, and manage their own information, such as setting dietary preferences Or upload a head image. These operations interact with the server and database to store and verify the user's personal information and security data. Through the security design of the API (such as JWT verification), the user's privacy and account security are guaranteed.
    b. Foodstuff API:
    Food inventory API helps users manage their food inventory information. Through this API, users can add new ingredients to the system, modify the amount of ingredients, set the food storage method and expiration date. Through this API, the server interacts with the database to store and update these Information, ensure that the user's inventory information is updated in real time, and can send reminders to the user based on the expiration date. This API also supports the user to view the current inventory status, ensuring that the user always knows the available ingredients.
    c. Recipe API:

The recipe recommendation API provides users with a menu recommendation based on the ingredients in their inventory. The purpose of the API is to recommend a suitable menu based on the ingredients that are about to expire or the user's preferences, to avoid food waste. The API sends inventory data to the server through the client, and the server based on this data searches for matching recipes in the database, and the recommended results are returned to the client for the user to view and select. This feature ensures a personalized user experience and helps users better plan their meals.

d. Overdue notification and synchronization API:
   This API synchronizes with the user's calendar, and reminds the user of the expiration date of the ingredients. 。The purpose of this API is to help users avoid expired ingredients, and at the same time provide a convenient reminder mechanism, so that users do not have to actively check inventory, and can also receive timely reminders about expired ingredients.
e. shopping listAPI:
   Shopping list API allows users to create and manage their shopping list. Users can add ingredients they need to buy to the shopping list through the client, and use them when actually shopping. This API also helps users avoid duplicate purchases already in stock. When the user adds ingredients to the shopping list, the server will check whether the ingredients are already in stock and pop up a notification to remind the user to prevent repeat purchases.

Design Issues

Functional Issues

1. How should users register?
    a. Email registration
    b. Phone number registration
    c. Third-party platform login (e.g., Google, Apple, Facebook)

    Choice: Email registration

    Justification: We have chosen to use email registration, which allows users to register their accounts with an email address and a unique username, ensuring that the account is recognizable and easy to access. Email registration is a traditional and effective way for users who are used to managing their accounts with their email address. In addition, email registration can support features such as password recovery and account security notifications, which meets the needs and preferences of most users. Email registration is less dependent on external services and ensures that users retain full control over the account creation and login process without being subject to third-party platforms that may not be available in all regions or may face outages. Unlike phone registration, email registration also avoids issues related to lost phone numbers or changes in phone plans.

2. How should users' uploaded information be stored?
    a. Manually inputted by users.
    b. Google Cloud Vision API
    c. Cloud Database (e.g., Firebase or AWS DynamoDB)

    Choice: all of them

    Justification: We choose to utilize all three options for storing users' uploaded information to provide flexibility and optimize user experience. Manual input allows users to directly enter and verify their data, ensuring accuracy. The Google Cloud Vision API enables automated extraction of information from images, reducing user effort and errors. Finally, using a cloud database like Firebase or AWS DynamoDB offers secure and scalable storage for both manually inputted and API-extracted data, with real-time synchronization across devices. This hybrid approach ensures a balance between user control, automation efficiency, and reliable data management, catering to diverse user needs and improving overall application functionality.

3. How should users be notified about expiring ingredients?
    a. Push notifications
    b. Email alerts
    c. Calendar integration (Google/Apple)

    Choice: Push notifications and Calendar integration (Google/Apple)

    Justification: Push notifications can alert users in a timely manner and send them directly to their devices, ensuring that they don't miss important expired reminders. And calendar integration helps users sync reminders to their schedules, making it easy for them to schedule shopping and cooking. While email reminders are useful, push notifications and calendar integration are more direct and efficient, so we chose both.

4. How do you recommend recipes based on available ingredients?
    a. User manually selects ingredients
    b. Automatically recommend recipes based on expiring ingredients
    c. Machine learning-based recipe recommendations based on past user preferences

    Choice: User manually selects ingredients and automatically recommends recipes based on expiring ingredients

    Justification: We chose to recommend recipes by letting the user manually select ingredients and by automatically recommending ingredients based on those that are about to expire. The manual selection of ingredients by the user provides a great deal of flexibility, allowing the user to decide which ingredients to use based on their needs and preferences. At the same time, automatically recommending recipes based on ingredients that are about to expire helps users minimize ingredient waste. While machine learning recommendations based on user's past preferences have the advantage of personalization, they are complex to implement and may lead to a lack of transparency in the recommendations, so we currently prefer to provide flexible and practical manual selection and smart reminders to ensure a simple and efficient user experience.

Nonfunctional Issues

1. What frontend framework should we use?
    a. React Native
    b. Flutter

    Choice: Flutter

    Justification: We chose Flutter as our development framework primarily because of its cross-platform capabilities, allowing us to build applications for both iOS and Android using a single codebase, which significantly reduces development and maintenance costs. Additionally, Flutter offers near-native performance due to its Dart language and direct compilation to native code, ensuring a smooth user experience. It also provides a rich set of pre-designed widgets and highly customizable UI tools, making it easy to create high-quality visuals and complex animations. Therefore, Flutter is the ideal choice for our project in terms of development efficiency and user experience.

2. Which backend framework should we use?
    a. Node.js
    b. Django
    c. Spring Boot

    Choice: Spring Boot

Justification: We chose Spring Boot as the back-end framework because the back-end of the project will be written in Java, and Spring Boot provides an environment that seamlessly integrates with Java.Spring Boot simplifies the back-end development process by providing auto-configuration, embedded servers, robust security mechanisms, and good integration with databases, and is particularly well suited for building scalable, enterprise-class applications. At the same time , Spring Boot has a rich ecosystem , can easily implement microservice architecture , which is very favorable for future expansion and maintenance . Therefore, Spring Boot is the best choice, in line with our project's technology stack and long-term planning.

3. Which database should we use?
    a. Firebase
    b. SQLite

Choice: SQLite

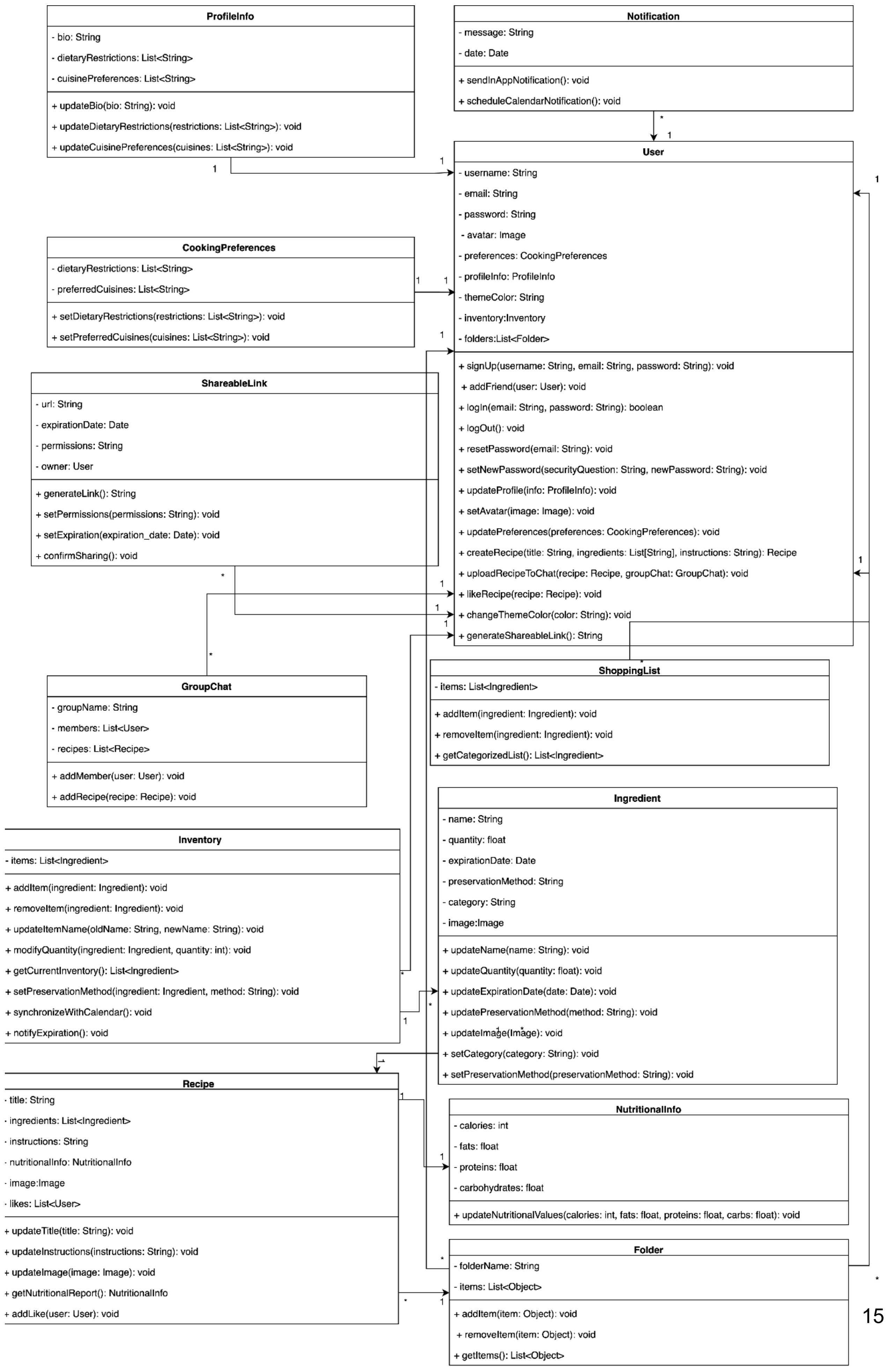
Justification: We chose SQLite as our database solution because it is lightweight, embedded and easy to maintain, which is ideal for the local data storage needs of mobile applications.SQLite does not require complex server configurations and supports cross-platform usage, enabling effective management of users' ingredients, recipes and preference information. In addition, SQLite's superior performance provides fast and reliable access to data, especially in scenarios where real-time cloud synchronization is not required. Therefore, SQLite fits well with our project's need for local data storage, balancing performance and simplicity.

4. How should we design and manage APIs?
    a. RESTful API
    b. GraphQL
    c. gRPC

Choice: RESTful API

Justification: We chose RESTful API as our main API design approach because it is simple, intuitive and widely used, suitable for handling various requests and resource management needs in our projects.RESTful API is based on the HTTP protocol, which has good compatibility and is easy to be integrated with front-ends (e.g. Flutter). Its layered architecture and statelessness make the system more flexible and easy to scale. Although GraphQL provides a more flexible data query mechanism and gRPC has higher performance, RESTful API is the most suitable choice considering that the main needs of the project are cross-platform support and simplicity. It provides a clear interface design and integrates well with Spring Boot, making back-end development more efficient.

Class Design


ProfileInfo
- bio: String
- dietaryRestrictions: List<String>
- cuisinePreferences: List<String>
+ updateBio(bio: String): void
+ updateDietaryRestrictions(restrictions: List<String>): void
+ updateCuisinePreferences(cuisines: List<String>): void
Notification
- message: String
- date: Date
+ sendInAppNotification(): void
+ scheduleCalendarNotification(): void
User
- username: String
- email: String
- password: String
- avatar: Image
- preferences: CookingPreferences
- profileInfo: ProfileInfo
- themeColor: String
- inventory:Inventory
- folders:List<Folder>
+ signUp(username: String, email: String, password: String): void
+ addFriend(user: User): void
+ logIn(email: String, password: String): boolean
+ logOut(): void
+ resetPassword(email: String): void
+ setNewPassword(securityQuestion: String, newPassword: String): void
+ updateProfile(info: ProfileInfo): void
+ setAvatar(image: Image): void
+ updatePreferences(preferences: CookingPreferences): void
+ createRecipe(title: String, ingredients: List[String], instructions: String): Recipe
+ uploadRecipeToChat(recipe: Recipe, groupChat: GroupChat): void
+ likeRecipe(recipe: Recipe): void
+ changeThemeColor(color: String): void
+ generateShareableLink(): String
CookingPreferences
- dietaryRestrictions: List<String>
- preferredCuisines: List<String>
+ setDietaryRestrictions(restrictions: List<String>): void
+ setPreferredCuisines(cuisines: List<String>): void
ShareableLink
- url: String
- expirationDate: Date
- permissions: String
- owner: User
+ generateLink(): String
+ setPermissions(permissions: String): void
+ setExpiration(expiration_date: Date): void
+ confirmSharing(): void
GroupChat
- groupName: String
- members: List<User>
- recipes: List<Recipe>
+ addMember(user: User): void
+ addRecipe(recipe: Recipe): void
ShoppingList
- items: List<Ingredient>
+ addItem(ingredient: Ingredient): void
+ removeItem(ingredient: Ingredient): void
+ getCategorizedList(): List<Ingredient>
Inventory
- items: List<Ingredient>
+ addItem(ingredient: Ingredient): void
+ removeItem(ingredient: Ingredient): void
+ updateItemName(oldName: String, newName: String): void
+ modifyQuantity(ingredient: Ingredient, quantity: int): void
+ getCurrentInventory(): List<Ingredient>
+ setPreservationMethod(ingredient: Ingredient, method: String): void
+ synchronizeWithCalendar(): void
+ notifyExpiration(): void
Ingredient
- name: String
- quantity: float
- expirationDate: Date
- preservationMethod: String
- category: String
- image:Image
+ updateName(name: String): void
+ updateQuantity(quantity: float): void
+ updateExpirationDate(date: Date): void
+ updatePreservationMethod(method: String): void
+ updateImage(Image): void
+ setCategory(category: String): void
+ setPreservationMethod(preservationMethod: String): void
Recipe
- title: String
- ingredients: List<Ingredient>
- instructions: String
- nutritionalInfo: NutritionalInfo
- image:Image
- likes: List<User>
+ updateTitle(title: String): void
+ updateInstructions(instructions: String): void
+ updateImage(image: Image): void
+ getNutritionalReport(): NutritionalInfo
+ addLike(user: User): void
NutritionalInfo
- calories: int
- fats: float
- proteins: float
- carbohydrates: float
+ updateNutritionalValues(calories: int, fats: float, proteins: float, carbs: float): void
Folder
- folderName: String
- items: List<Object>
+ addItem(item: Object): void
+ removeItem(item: Object): void
+ getItems(): List<Object>
1
*

Descriptions of Classes

User

- Attributes:
    - username: Unique identifier for the user.
    - email: User's email address.
    - password: User's password.
    - avatar: User's profile image.
    - preferences: Instance of CookingPreferences.
    - profileInfo: Instance of ProfileInfo, containing additional user information.
    - themeColor: User's preferred theme color.
- Methods:
    - Manages user registration, login, logout, password management, and profile updates.
    - Can create recipes and upload them to group chats.
    - Likes recipes and changes the app's theme color.

ProfileInfo

- Attributes:
    - bio: A brief description of the user.
    - dietaryRestrictions: List of dietary restrictions.
    - cuisinePreferences: List of preferred cuisines.
- Methods:
    - Updates user bio, dietary restrictions, and cuisine preferences.

CookingPreferences

- Attributes:
    - dietaryRestrictions: List of dietary restrictions.
    - preferredCuisines: List of preferred cuisines.
- Methods:
    - Sets dietary restrictions and preferred cuisines for the user.

Inventory

- Attributes:
    - items: List of Ingredient instances.
- Methods:
    - Manages adding, removing, and updating inventory items.
    - Can synchronize inventory with a calendar and notify users of item expirations.

Ingredient

- Attributes:
    - name: Name of the ingredient.
    - quantity: Amount of the ingredient.
    - expirationDate: Expiration date of the ingredient.

    - preservationMethod: How the ingredient should be stored.
    - category: Category (e.g., dairy, vegetable).
    - image: Image of the ingredient.
- Methods:
    - Updates the ingredient's details, including name, quantity, expiration date, preservation method, and image.

Recipe
- Attributes:
    - title: Title of the recipe.
    - ingredients: List of Ingredient instances used in the recipe.
    - instructions: Instructions for preparing the recipe.
    - nutritionalInfo: Instance of NutritionalInfo.
    - image: Image of the recipe.
    - likes: List of User instances who liked the recipe.
- Methods:
    - Updates the recipe's title, instructions, and image.
    - Retrieves nutritional information and allows users to like the recipe.

Folder
- Attributes:
    - folderName: Name of the folder.
    - items: List of objects (could be recipes, ingredients, etc.).
- Methods:
    - Manages adding and removing items from the folder and retrieving the list of items.

NutritionalInfo
- Attributes:
    - calories: Total calories in the recipe or ingredient.
    - fats: Amount of fats.
    - proteins: Amount of proteins.
    - carbohydrates: Amount of carbohydrates.
- Methods:
    - Updates the nutritional values.

ShoppingList
- Attributes:
    - items: List of Ingredient instances.
- Methods:
    - Manages adding and removing items from the shopping list and categorizing them.

Notification
- Attributes:

- message: Notification message.
- date: Date of the notification.
- Methods:
    - Sends in-app notifications and schedules calendar notifications.

GroupChat

- Attributes:
    - group_name: Name of the group chat.
    - members: List of User instances in the group.
    - recipes: List of Recipe instances shared in the chat.
- Methods:
    - Adds members to the chat and allows recipes to be shared within the group.

ShareableLink

- Attributes:
    - url: Generated URL for sharing.
    - expirationDate: Date when the link will expire.
    - permissions: Defines access permissions (e.g., view-only, editable).
    - owner: The user who created the link.
- Methods:
    - Generates a shareable link, sets permissions and expiration date, and confirms sharing.

Interactions Between Classes

1. User and ProfileInfo/CookingPreferences:
    - A User has ProfileInfo and CookingPreferences attributes, allowing them to manage their personal details and dietary preferences.
2. User and Recipe:
    - Users can create recipes, which are instances of the Recipe class. They can also like recipes and upload them to GroupChat instances.
3. User and Inventory:
    - Each User has an associated Inventory that contains Ingredient items. Users can add or modify ingredients in their inventory.
4. Inventory and Ingredient:
    - The Inventory class manages Ingredient objects, allowing for operations such as updating quantities and expiration dates.
5. Recipe and NutritionalInfo:
    - Each recipe contains an instance of NutritionalInfo, which provides detailed nutritional data about the recipe.
6. Folder:
    - Users can organize their recipes and ingredients into Folder objects, allowing for better management and retrieval.
7. Notification:

- The Notification class can be used to alert users about important events, such as item expirations in their inventory or updates to recipes they follow.

8. GroupChat and User/Recipe:
    - GroupChat facilitates communication among users and allows for recipe sharing among members.
9. ShareableLink:
    - Users can generate ShareableLink instances to share their inventory or recipes with others, specifying permissions and expiration dates.

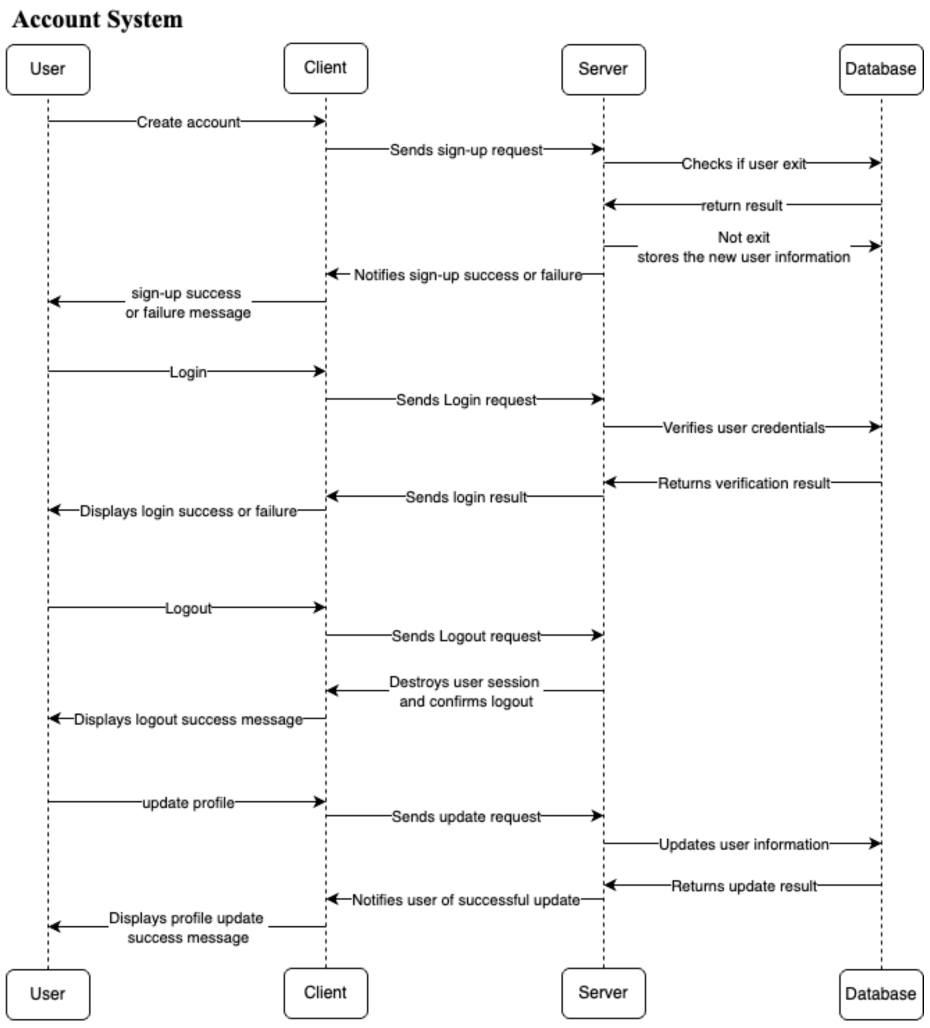
Sequence of Events Overview

In these sequence diagrams, we show the interaction between the user, the client, the server, and the database. The user's actions begin with the creation of a FOO MY FOOD account or logging into an existing account via e-mail; account information is first passed to the client, which then sends the information to the server, which processes the request and queries or stores the user's information from the database.

After creating an account, the user can perform the following actions: add or modify ingredients, upload or edit recipes, participate in group chats, create shopping lists, set and modify food and cooking preferences, and view ingredient inventory. Each operation is initiated by the client and sent as a request to the server. The server is responsible for creating query requests to store or retrieve data from the database. Once the request is successfully processed by the database, the result of the operation (e.g., a success message) is sent back to the client via the server, ultimately notifying the user that the operation is complete.

Sequence diagram for Account System

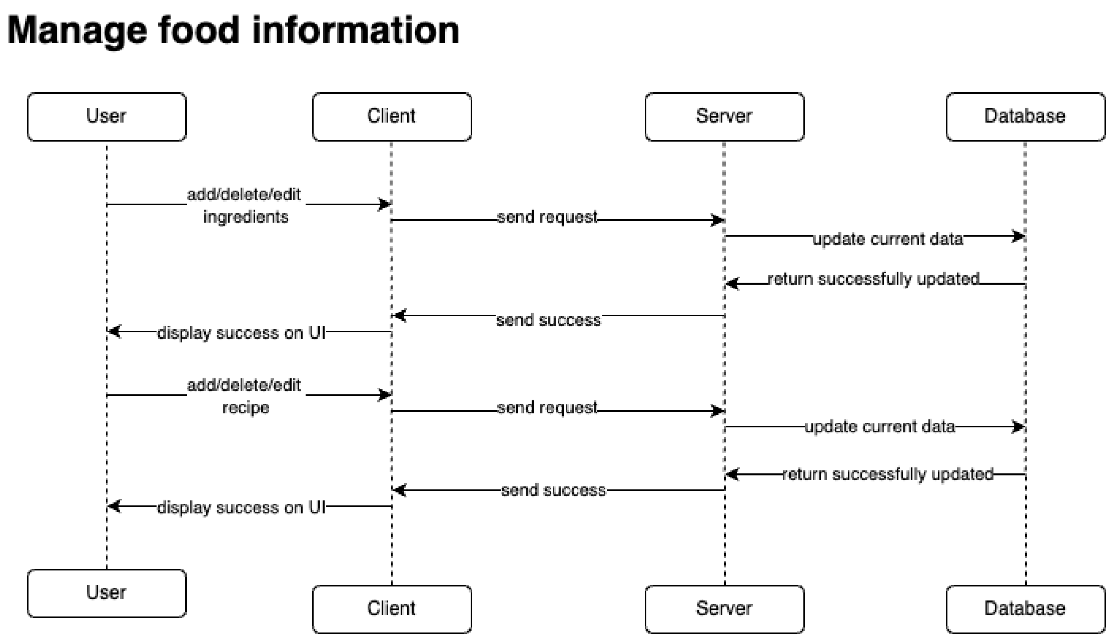
When a user installs the FOO MY FOOD application, the user has the option to create an account or log in via email. If the user chooses to create a new account, the client sends the request to the server, which saves the user details and stores the data in the database. Once the account is created, the user can also update their profile information (e.g. username, email, avatar, etc.). When the user updates this information, the client sends the request to the server, which updates the user's details in the database and returns a success message to the client informing the user that the operation was successful.



Sequence Diagram for Food Information Management

In the function of managing food information, users can add, delete or edit ingredients through the client. The client sends the user's request to the server, and the server stores the updated information in the database. After the database is updated successfully, the server returns a success message, and the client displays the successful operation information on the user interface.

Similarly, users can add, delete or edit recipes through the client. The client sends these requests to the server, and the server also updates the information in the database and returns a success message to the client, and the client displays the successful operation information on the user interface.



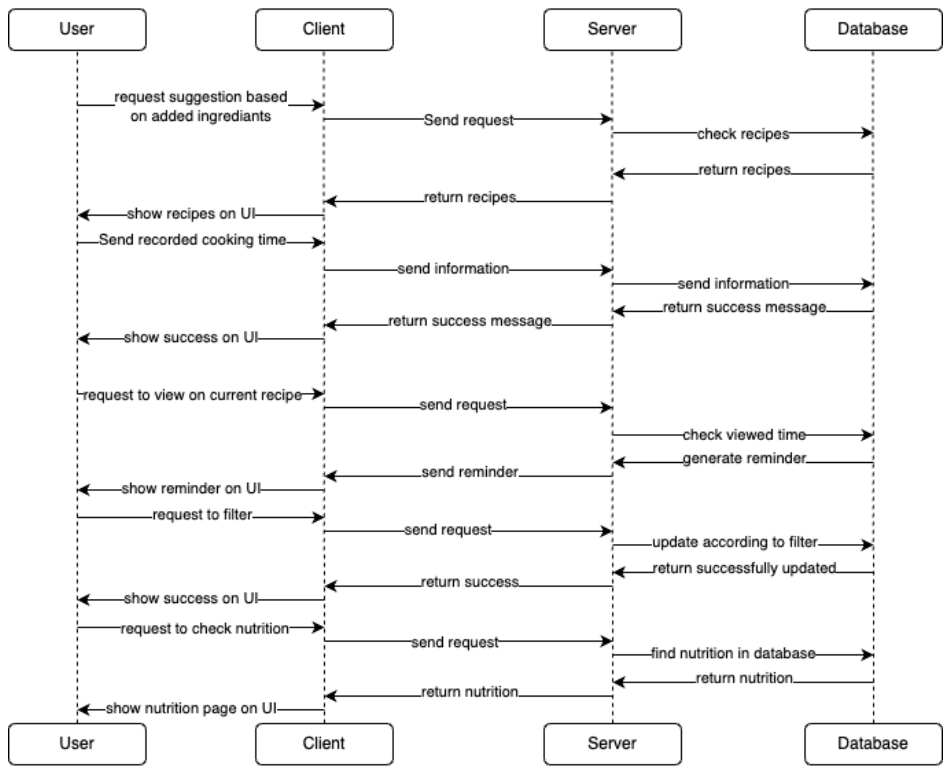
Sequence Diagram for Smart Menu

In the smart menu function, users can request recipe suggestions based on the currently added ingredients through the client. The client sends the request to the server, which checks the recipes in the database and returns the relevant recipe information to the client, which displays the recipes on the user interface.

Users can also record cooking time or view the current recipe through the client, and the client sends these requests to the server. The server records the relevant cooking information and recipes in the database and returns a success message to the client, which displays the success message on the user interface.

Users can also request reminders or filter recipe information through filters. The server generates reminders or filter results based on the user's request and returns them to the client, which displays a prompt of successful operation on the user interface.

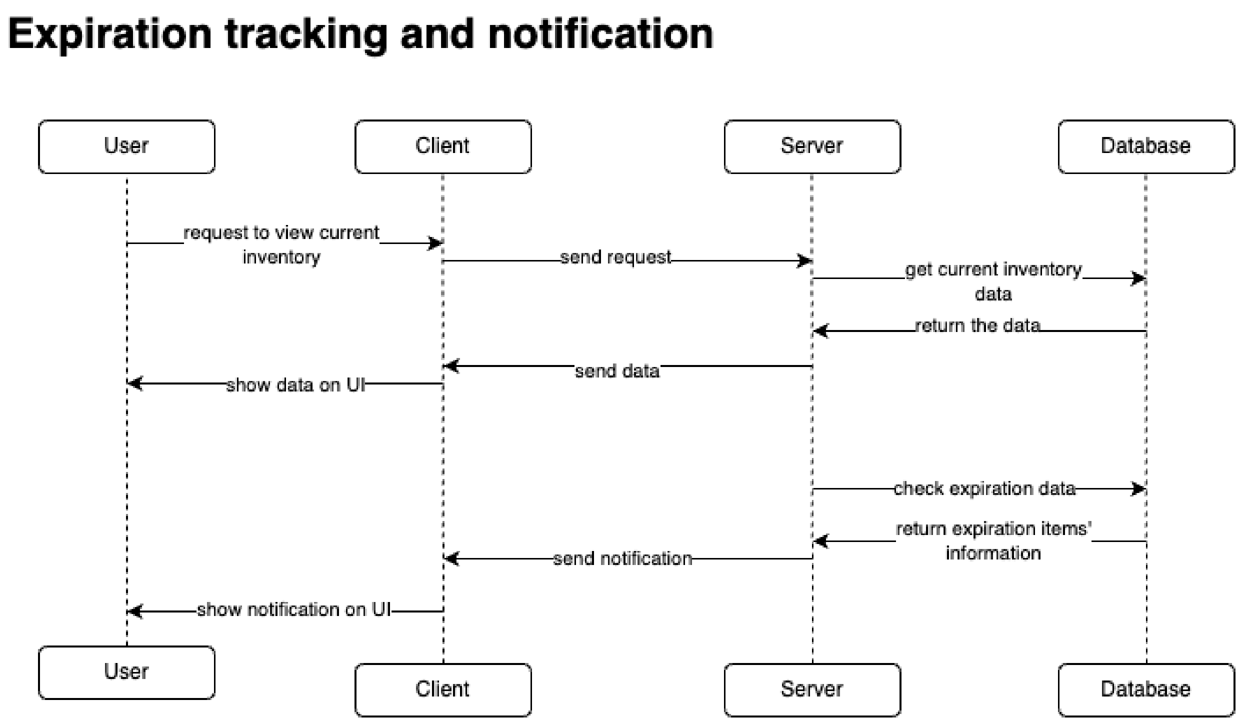
Smart Menu



Sequence Diagram for Expiration Tracking and Notification

In the expiration tracking and notification feature of the application, users can view the current inventory through the client request. The client sends the request to the server, and the server obtains the current inventory data from the database and returns it to the client. The client then displays the inventory data on the user interface.

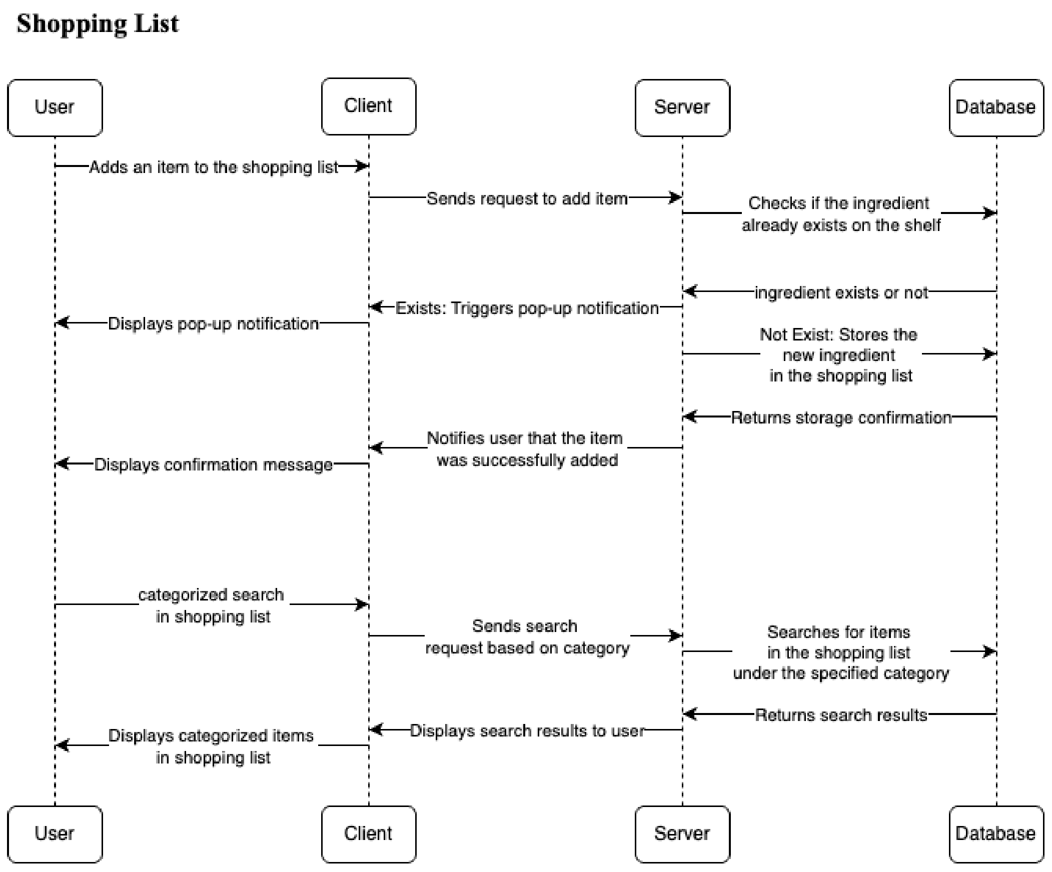
The server also periodically checks the inventory in the database to see if there are items that are about to expire. If so, the server sends a notification of the expired item to the client, and the client displays the relevant notification to the user on the user interface.



Sequence Diagram for Shopping List

The user adds the item to the shopping list The user adds the item to the shopping list through the client, which sends the request to the server. The server checks if the item already exists in the database. If the item exists, the server will trigger a pop-up notification to remind the user; if the item does not exist, the server will store the new item in the database. After the database confirms successful storage, the client notifies the user that the item has been successfully added and displays a corresponding confirmation message.

The user can also perform a category search in the shopping list. The client sends the category search request to the server, which passes the request to the database, which searches the items in the shopping list according to the specified categories and returns the search results. The client displays the results to the user.

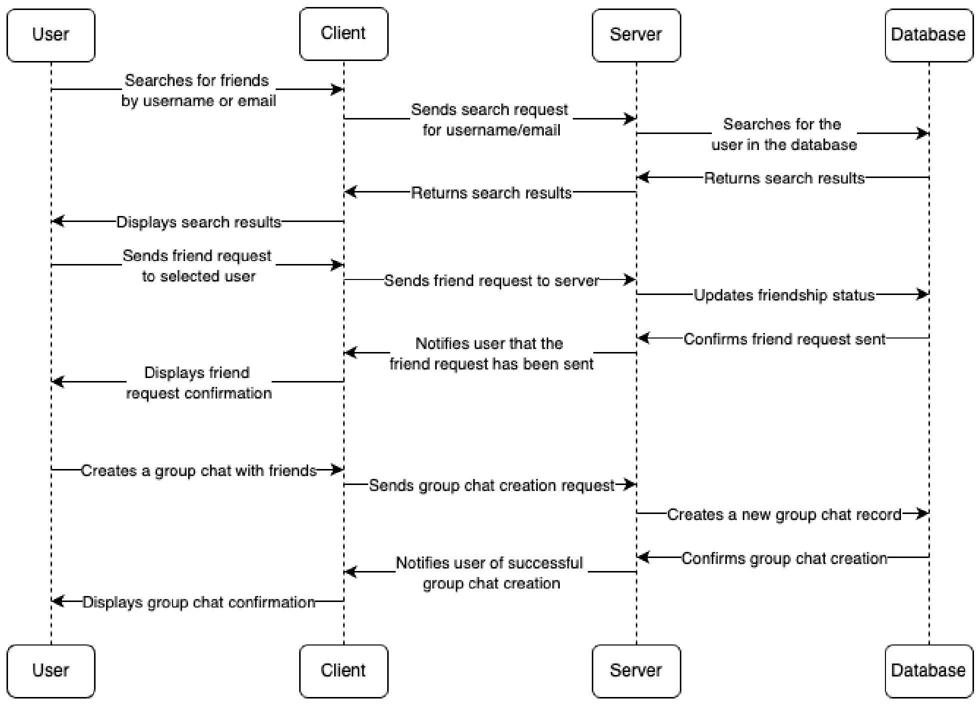

Shopping List
User
Client
Server
Database
Adds an item to the shopping list
Sends request to add item
Checks if the ingredient already exists on the shelf
ingredient exists or not
Exists: Triggers pop-up notification
Displays pop-up notification
Not Exist: Stores the new ingredient in the shopping list
Returns storage confirmation
Notifies user that the item was successfully added
Displays confirmation message
categorized search in shopping list
Sends search request based on category
Searches for items in the shopping list under the specified category
Returns search results
Displays search results to user
Displays categorized items in shopping list
User
Client
Server
Database

Sequence Diagram for Friend Management and Group Chat Features

When a user searches for a friend, the client sends the request to the server, which queries the database to see if a matching username or email exists. Once a matching user is found, the client displays the search results. The user can send a friend request to the selected user. The client sends the request to the server, which updates the status of the friend relationship in the database and returns a confirmation message informing the user that the friend request was successfully sent.

After confirming the friend relationship, users can create a group chat with their friends. The client sends the request to create a group chat to the server, which processes the request and creates a new group chat record in the database. After the server confirms that the group chat has been created successfully, the client notifies the user that the group chat has been created.

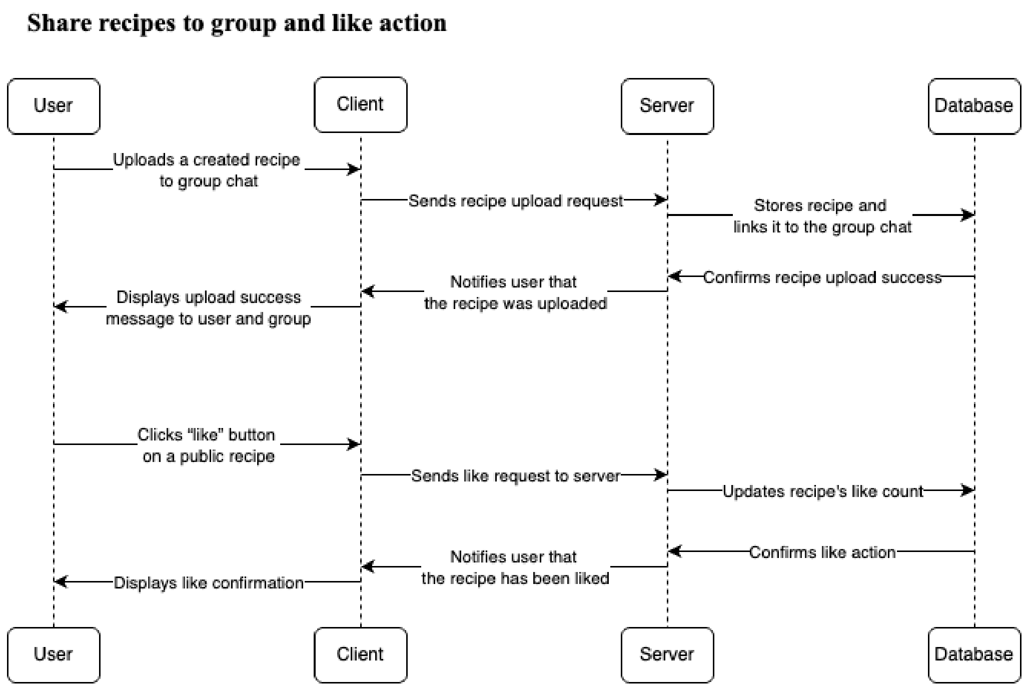
Friend and group Management


User
Client
Server
Database
Searches for friends by username or email
Sends search request for username/email
Searches for the user in the database
Returns search results
Returns search results
Displays search results
Sends friend request to selected user
Sends friend request to server
Updates friendship status
Confirms friend request sent
Notifies user that the friend request has been sent
Displays friend request confirmation
Creates a group chat with friends
Sends group chat creation request
Creates a new group chat record
Confirms group chat creation
Notifies user of successful group chat creation
Displays group chat confirmation
User
Client
Server
Database

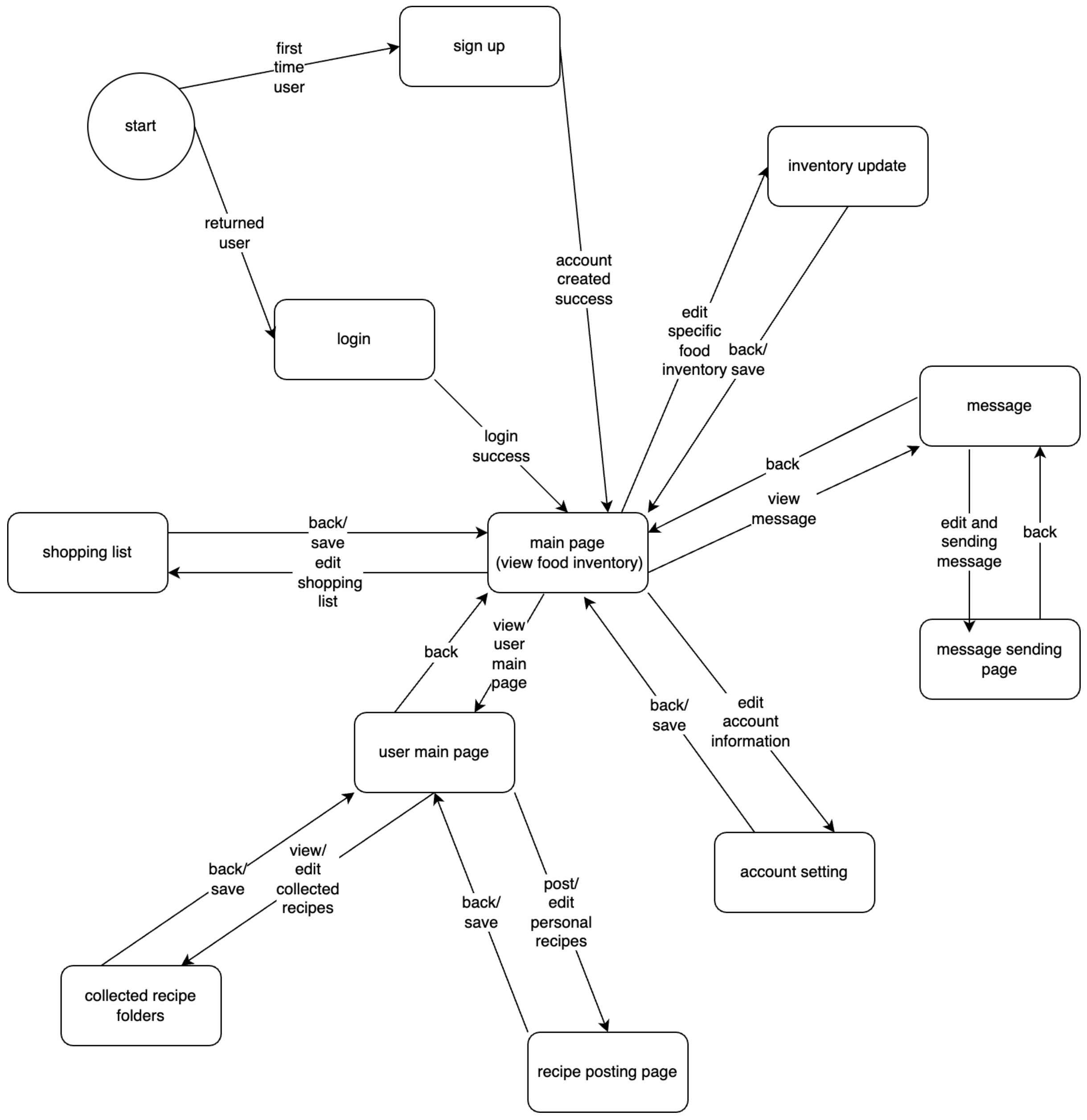
Sequence Diagram for Sharing Recipe to group and Liking a Public Post

When a user creates and shares a recipe in the FOO MY FOOD app, the client sends a request to the server to share the recipe to a specified group chat. The server passes the request to the database, which stores the recipe and associates it with the group chat. When the operation is complete, the database returns an acknowledgement message, the server returns the result to the client, and the client notifies the user that the sharing was successful and displays the information in the group chat.

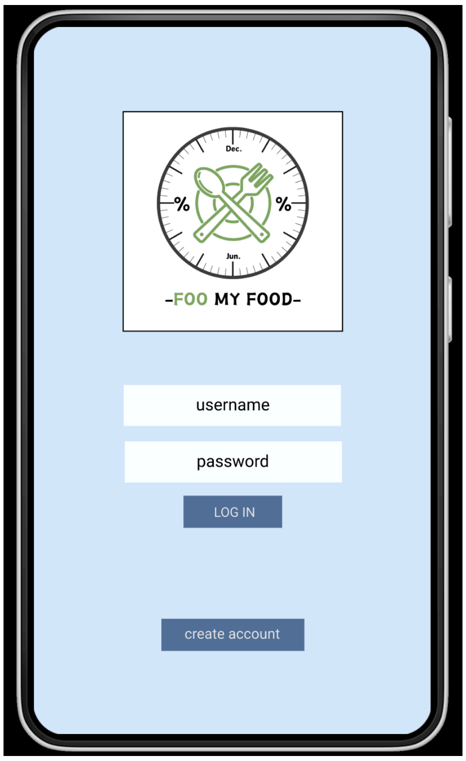
When a user clicks the “Like” button on a public recipe, the client sends the like request to the server. The server processes the request and passes it to the database, which updates the number of likes for the recipe and confirms that the operation was successful. The confirmation message is returned to the client through the server and the client notifies the user that the like has been successful.

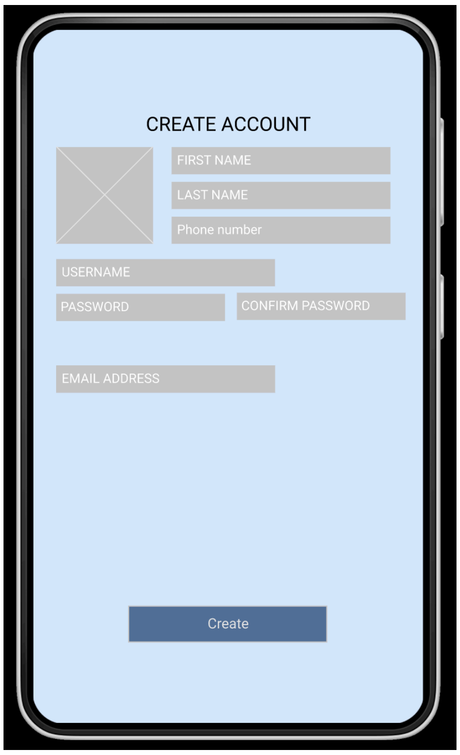

Share recipes to group and like action
User
Client
Server
Database
Uploads a created recipe to group chat
Sends recipe upload request
Stores recipe and links it to the group chat
Confirms recipe upload success
Notifies user that the recipe was uploaded
Displays upload success message to user and group
Clicks "like" button on a public recipe
Sends like request to server
Updates recipe's like count
Confirms like action
Notifies user that the recipe has been liked
Displays like confirmation
User
Client
Server
Database

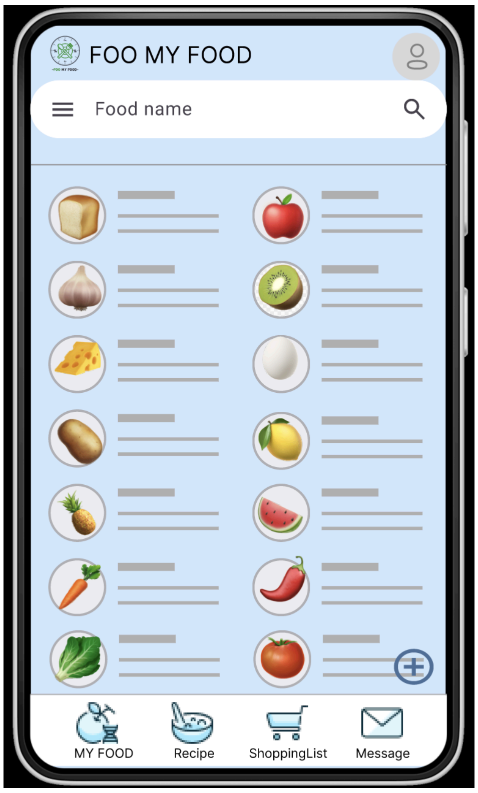
Navigation Flow Map

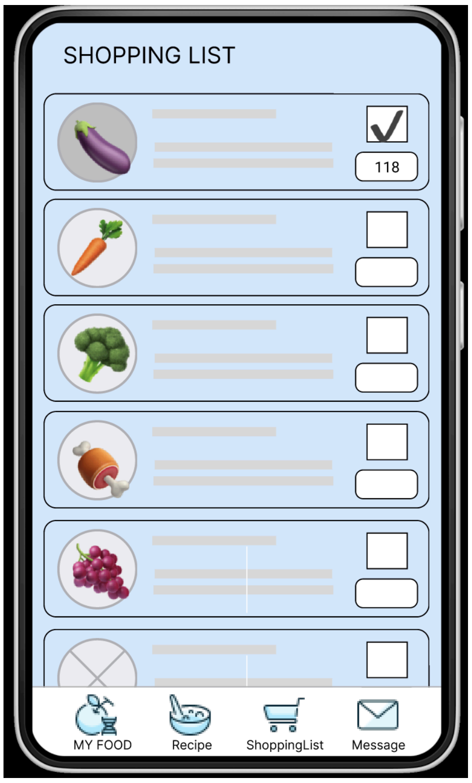


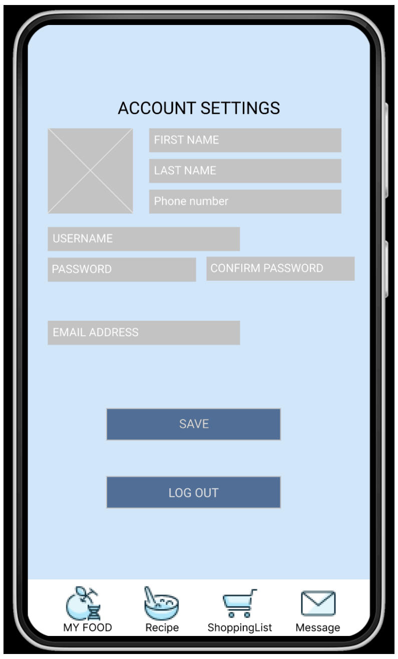
UI Mockups

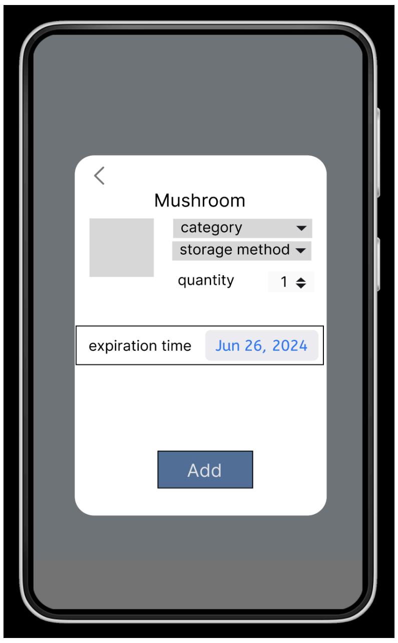

Dec.
%
%
Jun.
-FOO MY FOOD-
username
password
LOG IN
create account

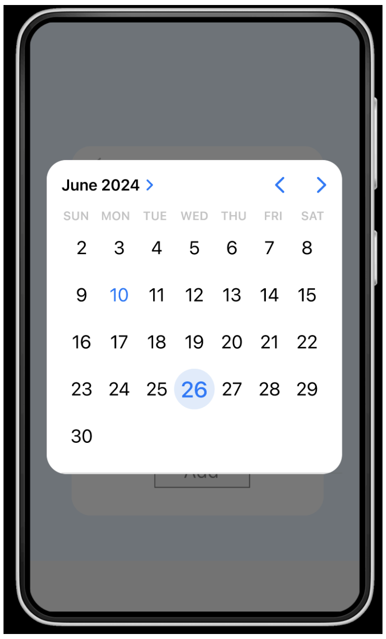

CREATE ACCOUNT
FIRST NAME
LAST NAME
Phone number
USERNAME
PASSWORD
CONFIRM PASSWORD
EMAIL ADDRESS
Create

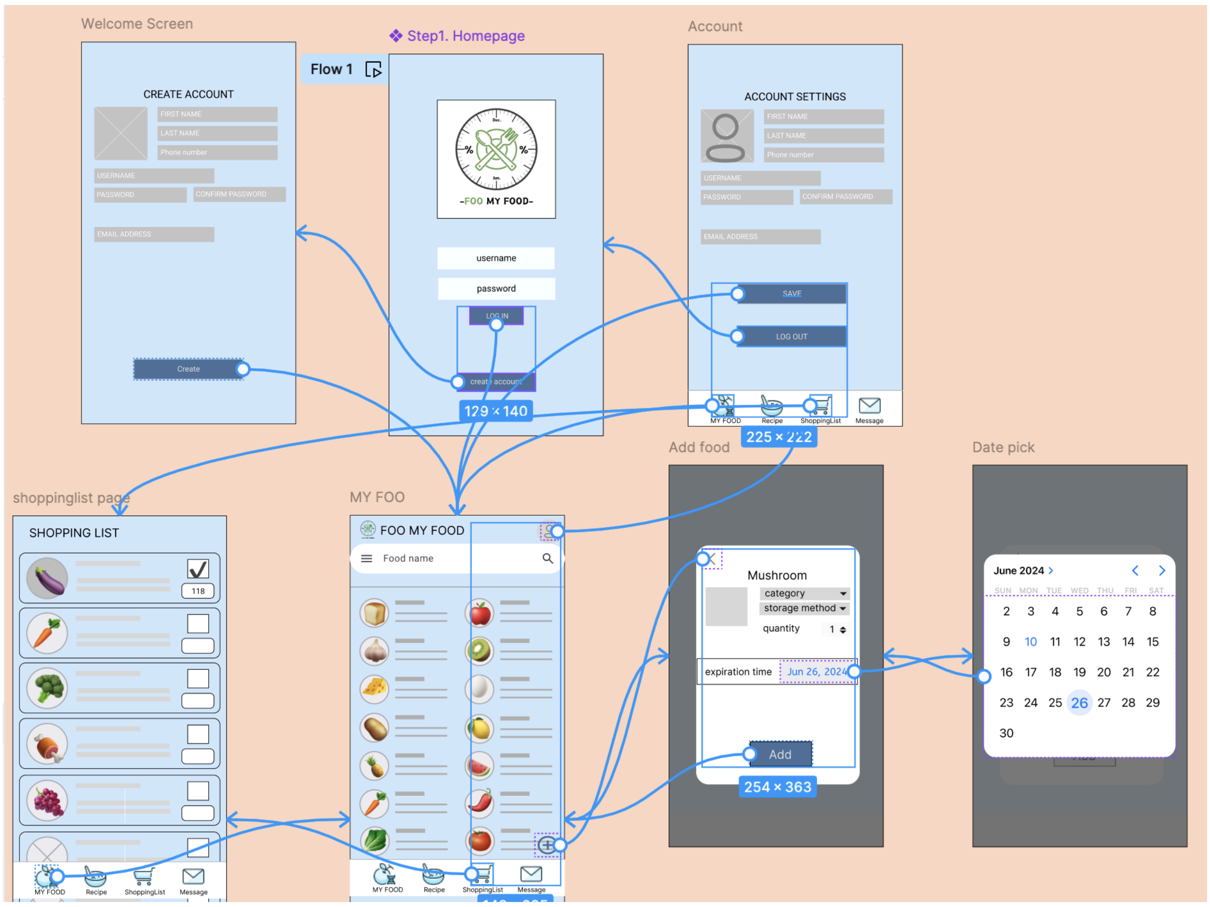

FOO MY FOOD
Food name
MY FOOD
Recipe
ShoppingList
Message


SHOPPING LIST
118
MY FOOD
Recipe
ShoppingList
Message


ACCOUNT SETTINGS
FIRST NAME
LAST NAME
Phone number
USERNAME
PASSWORD
CONFIRM PASSWORD
EMAIL ADDRESS
SAVE
LOG OUT
MY FOOD
Recipe
ShoppingList
Message


Mushroom
category
storage method
quantity
1
expiration time
Jun 26, 2024
Add


June 2024
SUN MON TUE WED THU FRI SAT
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30


Welcome Screen
CREATE ACCOUNT
Create
Flow 1
Step1. Homepage
FOO MY FOOD
username
password
LOG IN
create account
129 × 140
Account
ACCOUNT SETTINGS
SAVE
LOG OUT
225 × 222
Add food
Date pick
shoppinglist page
SHOPPING LIST
MY FOO
FOO MY FOOD
Food name
Mushroom
expiration time
Jun 26, 2024
Add
254 × 363
June 2024
MY FOOD
Recipe
ShoppingList
Message