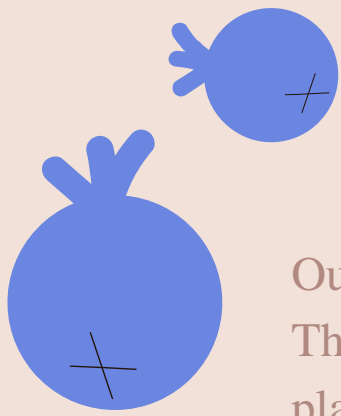


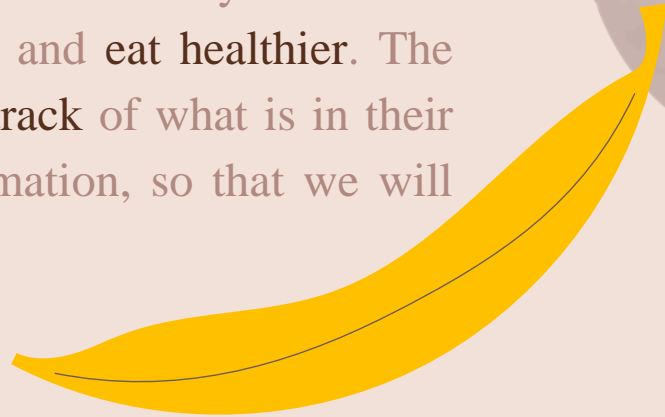


YumCycle: Revolutionizing Food Management



Introduction To YumCycle

Our project, *YumCycle*, is a fridge contents management application. This mobile app is designed around helping people of all ages and places in managing their fridge leftovers. The idea has started in our own dorm room, where every week, we buy a lot of unnecessary groceries and always end up throwing some of them away. This is how this project was born, we realized we could optimize the way we handle our aliments and also find new recipes to cook and eat healthier. The main focus of this app is to help the user keep track of what is in their fridge and generate recipes based on this information, so that we will minimize waste.



What do we want to achieve?

O1

Efficient Inventory Management

YumCycle enables users to efficiently manage their fridge and pantry contents by providing tools to track what they have on hand.

O2

Reducing Food Waste

The application is designed with a focus on reducing food waste, allowing users to receive suggestions recipes that utilize ingredients that might otherwise go to waste.

O3

Recipe Discovery and Sharing

YumCycle enhances the user experience by offering personalized recipe suggestions based on the ingredients users already have, and share their own recipes.

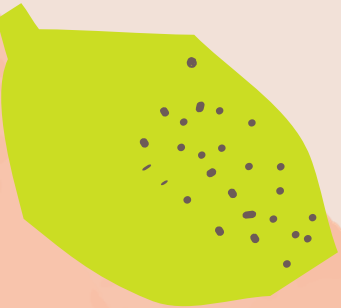


KEY FEATURES

● Barcode Scanning

One of the standout features of YumCycle is its barcode scanning capability. Users can quickly add items to their virtual fridge simply by scanning product barcodes, making inventory updates fast and efficient, while eliminating manual entry errors.

● Manual Entry Options



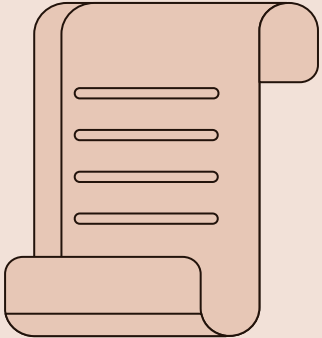
For products without barcodes, YumCycle provides a manual entry option. This ensures that users can still log all their pantry and fridge items, regardless of packaging, thus maintaining a comprehensive inventory for better meal planning.

● Recipe Suggestions

Based on the current inventory, YumCycle offers personalized recipe suggestions that help users utilize their existing ingredients. This feature not only encourages creative cooking but also helps in meal preparation while minimizing food waste.

"Throwing away food is like stealing from the table of those who are poor and hungry."

— POPE FRANCIS



Backend Development with Java and Spring Boot

◎ Java as a Programming Language

Java is a robust and scalable programming language that is widely used for building mobile and backend applications.

Its security features and cross-platform capabilities make it an excellent choice for developers. Java's compatibility with Android also allows for seamless mobile application development, making it a versatile tool in the tech stack of YumCycle.

◎ Spring Boot Framework

Spring Boot is a revolutionary framework that simplifies the development of Java applications.

By offering pre-configured settings, Spring Boot allows developers to focus on building features rather than dealing with complex configurations. It supports the creation of RESTful APIs, enhances security protocols, and efficiently manages database interactions, playing a crucial role in the architecture of YumCycle.

◎ MySQL Database Management

MySQL serves as the relational database management system for YumCycle, where it efficiently stores critical data such as user accounts, inventory items, recipes, and user reviews.

Its robust querying capabilities and reliability ensure that the application can handle large volumes of data while maintaining performance and integrity.

SECURITY FEATURES



YumCycle implements various security measures to protect user data and application integrity. Utilizing Spring Boot's built-in security features, the application safeguards sensitive information, implements authentication processes, and enforces authorization protocols, ensuring a secure environment for users.

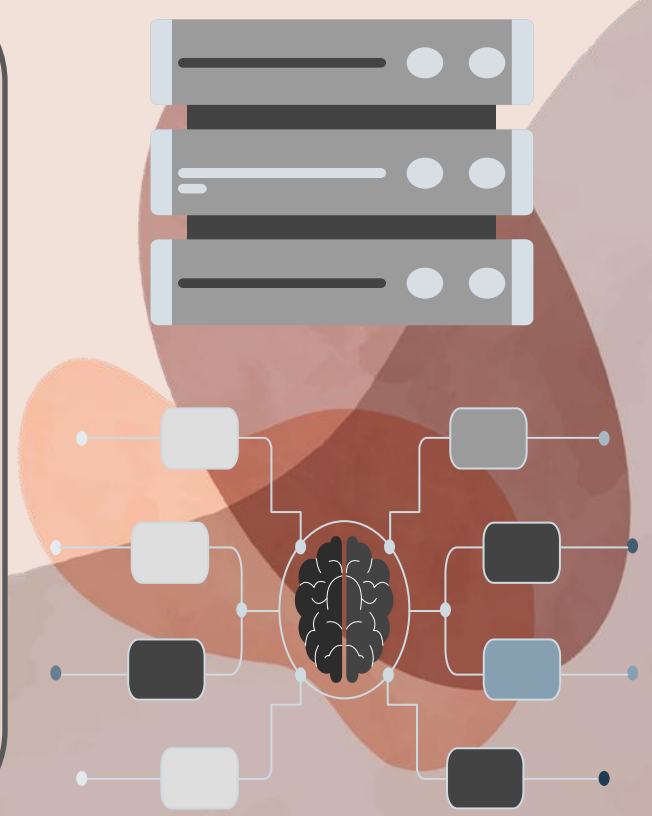
API INTEGRATION

Barcode API Integration

YumCycle leverages a Barcode API to simplify the addition of items to the virtual fridge. Users can scan product barcodes, and the API retrieves product details such as name, brand, and packaging. This reduces manual entry, speeds up inventory management, and improves accuracy in item tracking. You'll never forget to buy milk.

Recipe API Integration

YumCycle integrates a Recipe API to dynamically fetch recipes based on users' inventory and preferences. The API matches available ingredients with a diverse recipe database, offering tailored suggestions that reduce food waste. This integration enriches the recipe discovery experience while maintaining efficiency and reliability.



USER EXPERIENCE AND INTERACTION

Because user experience is so important to us,
we will ask you to be the first ones to make a
decision related to the user interface:

Please scan this QR
code and help us!



OPTION A



OPTION B



User Registration and Authentication

- Secure User Registration:

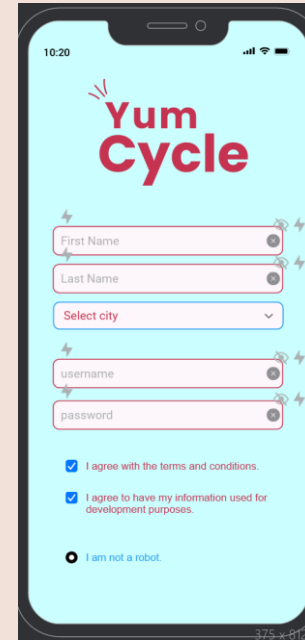
Enables users to create accounts with strong passwords, ensuring the protection of personal information.

- Efficient Authentication Process:

Provides quick and seamless login options, including password recovery and 'Remember Me' functionality.

- User Profile Management:

Allows users to update personal details, change passwords, and customize notification preferences for a tailored experience.



10:20

Yum Cycle

First Name

Last Name

Select city

username

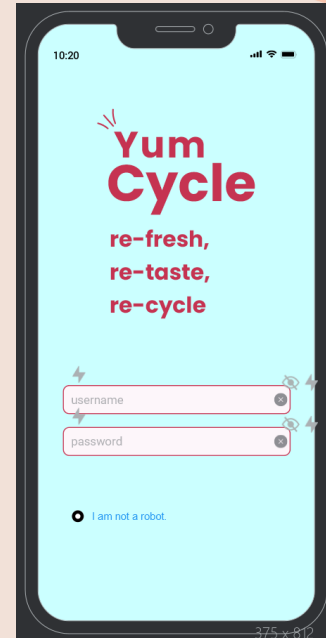
password

☒ I agree with the terms and conditions.

☒ I agree to have my information used for development purposes.

☐ I am not a robot.

375 x 812



10:20

Yum Cycle

re-fresh,
re-taste,
re-cycle

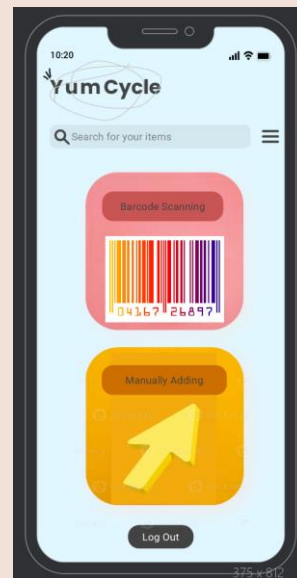
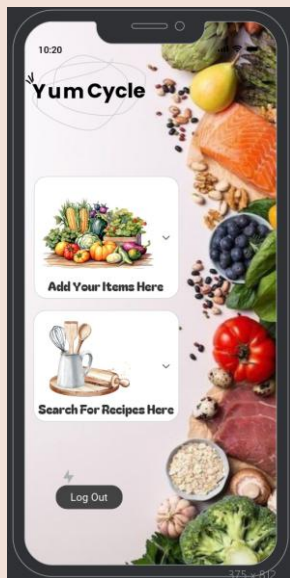
username

password

☐ I am not a robot.

375 x 812

Inventory Management And Recipe Discovery



● Personalized Recipes:

YumCycle enhances the cooking experience by offering personalized recipe suggestions based on the ingredients available in the user's inventory and application allows users to submit their own recipes, fostering a community of sharing and creativity.

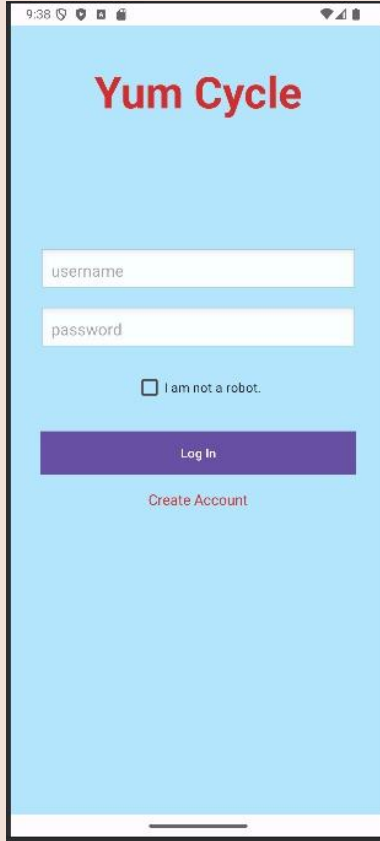
● Barcode Scanning Feature

Users can quickly add items to their fridge and pantry by scanning barcodes.

● Manual Entry Options

For items without barcodes, YumCycle provides a manual entry option. Users can easily input product names and quantities.

The Implemented User Interface



9:38

Yum Cycle

username

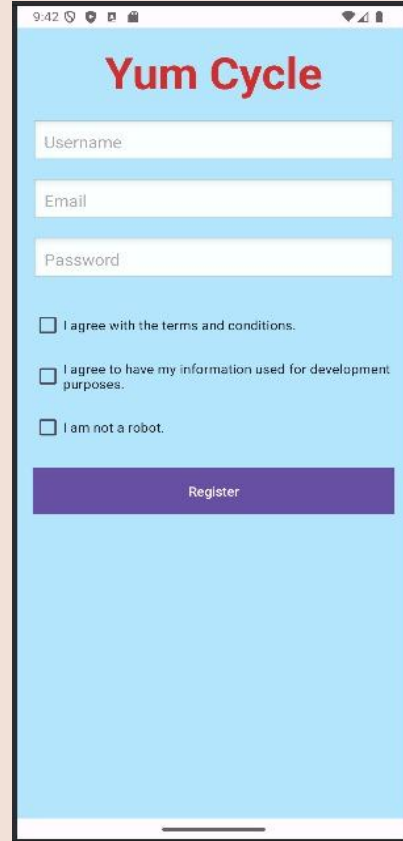
password

☐ I am not a robot.

Log In

[Create Account](#)

This is a mobile app login screen for 'Yum Cycle'. It features a light blue background. At the top, the status bar shows the time as 9:38. The app title 'Yum Cycle' is in red. Below it are two white input fields for 'username' and 'password'. A checkbox labeled 'I am not a robot.' is positioned below the password field. A purple 'Log In' button is centered below the checkbox, and a red 'Create Account' link is at the bottom.



9:42

Yum Cycle

Username

Email

Password

☐ I agree with the terms and conditions.

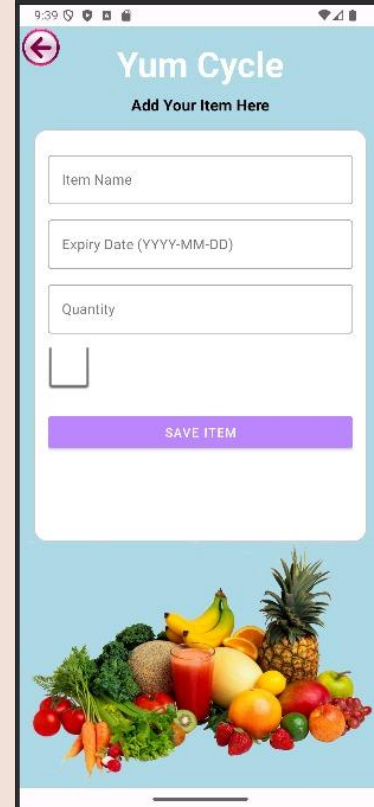
☐ I agree to have my information used for development purposes.

☐ I am not a robot.

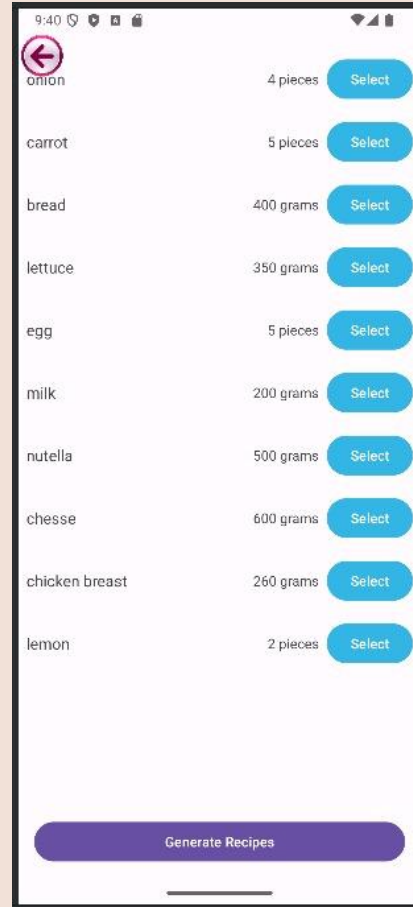
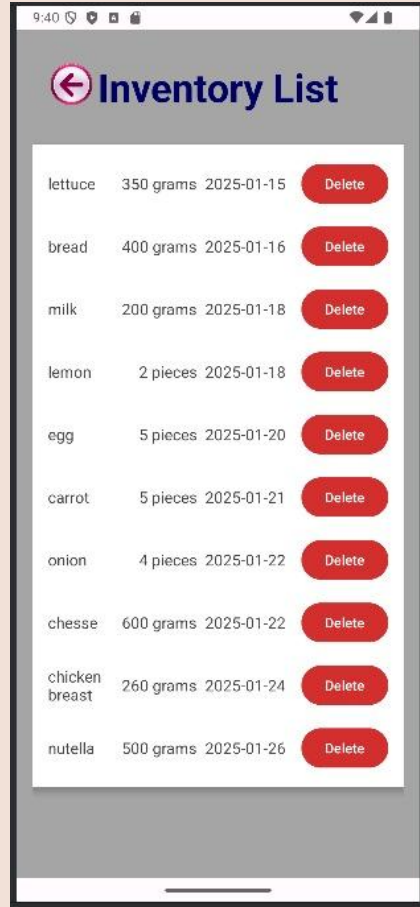
Register

This is a mobile app registration screen for 'Yum Cycle'. It features a light blue background. At the top, the status bar shows the time as 9:42. The app title 'Yum Cycle' is in red. Below it are three white input fields for 'Username', 'Email', and 'Password'. Three checkboxes follow: 'I agree with the terms and conditions.', 'I agree to have my information used for development purposes.', and 'I am not a robot.'. A purple 'Register' button is centered below the checkboxes.

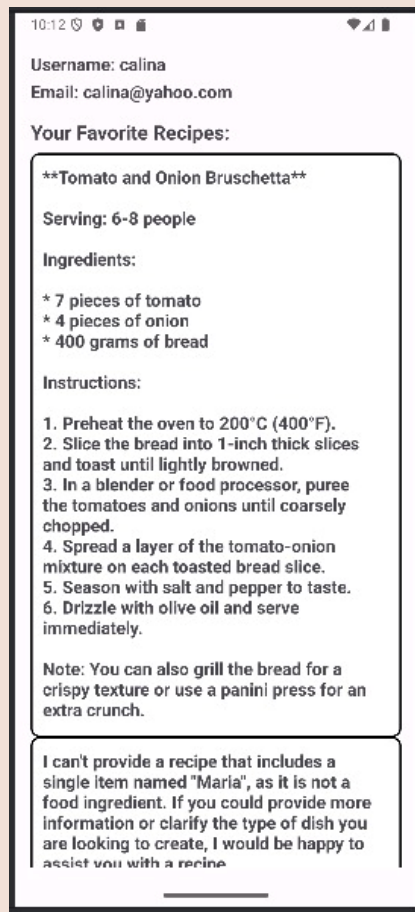
The Implemented User Interface



The Implemented User Interface



The Implemented User Interface



Testing and Validation

Bugs List and Fixes

Nr.	Description	Iden. date	Fix date	Open Period	Final Status
1.	Data Type Mismatch in Foreign Keys Issue	18.12.2024	28.12.2024	10 days	Closed
2.	Conflicting Entity Mappings Between Backend and Frontend	22.12.2024	07.01.2025	16 days	Closed
3.	Inconsistent Data Types Between Frontend Models and Backend Entities Issue	05.01.2025	10.01.2025	5 days	Closed
4.	Lack of Data Transfer Objects (DTOs) Issue	08.01.2025	11.01.2025	3 days	Closed
5.	Backend Controller Not Properly Handling Entity Relationships Issue	03.01.2025	05.01.2025	2 days	Closed
6.	Database Schema Not Fully Aligned with Backend Entities Issue	19.12.2024	28.12.2024	8 days	Closed
7.	Frontend Fragment Not Using Updated Models and DTO Issue	08.01.2025	10.01.2025	2 days	Closed
8.	Inconsistent Naming Conventions and Field Definitions Issue	15.12.2024	18.12.2024	3 days	Closed
9.	Lack of Comprehensive Logging and Error Handling Issue	15.12.2024	11.01.2025	27 days	Closed
10.	Frontend-Backend Integration Testing Oversights	12.12.2024	11.01.2025	30 days	Closed

Testing and Validation

Bugs List and Fixes

1. Data Type Mismatch in Foreign Keys

Ex: The favorites table in the database had columns (user_id and recipe_id) defined as INT, while the corresponding id fields in the users and recipes tables were of type BIGINT. This mismatch prevented the establishment of foreign key constraints.

Solution:

Changed the user_id and recipe_id columns in the favorites table from INT to BIGINT to match the referenced id fields. Dropped existing foreign key constraints and re-added them after ensuring the data types were compatible.

2. Conflicting Entity Mappings Between Backend and Frontend

Ex: In the backend, both a @ManyToMany relationship in the Recipe entity and a separate Favorites entity were managing the same favorites table. This dual mapping caused Hibernate to misinterpret entity relationships, leading to errors when persisting data.

Solution:

Removed the @ManyToMany annotation from the Recipe entity to eliminate conflicts. Utilized the separate Favorites entity exclusively to manage the relationship between User and Recipe.

3. Inconsistent Data Types Between Frontend Models and Backend Entities

Ex: The frontend Favorites class used an int for userId, whereas the backend expected a Long. Additionally, field names like name vs. recipeName caused serialization mismatches.

Solution:

Updated the frontend Favorites class to use Long for userId to align with the backend. Renamed frontend fields (e.g., from name to recipeName) to match backend entity field names, ensuring proper serialization and deserialization.

Testing and Validation

Bugs List and Fixes

4. Lack of Data Transfer Objects (DTOs)

Ex: The frontend was directly sending Favorites entities to the backend, which expected a simplified FavoritesDTO containing only userId and recipeId. This led to incomplete or mismatched data being processed by the backend.

Solution:

Created separate FavoritesDTO classes in both frontend and backend to handle data transfer, ensuring that only the necessary fields (userId and recipeId) were communicated. Modified Retrofit service interfaces and backend controllers to utilize FavoritesDTO for adding favorites, ensuring accurate data mapping.

5. Backend Controller Not Properly Handling Entity Relationships

Ex: When the frontend sent a FavoritesDTO, the backend controller did not correctly fetch and set the associated User and Recipe entities in the Favorites entity before saving. This resulted in null or transient references, causing Hibernate to throw PropertyValueException.

Solution:

Updated the FavoritesController to fetch User and Recipe entities based on the IDs provided in FavoritesDTO. Set these entities in the Favorites entity before persisting, ensuring all required fields were properly populated.

6. Database Schema Not Fully Aligned with Backend Entities

Ex: Even after updating entity mappings, discrepancies in the database schema (such as incorrect data types or missing foreign key constraints) persisted, leading to further persistence errors.

Solution:

Verified and modified the database schema to ensure that all columns and constraints matched the backend entity definitions. This included confirming that user_id and recipe_id were correctly typed and that foreign key relationships were properly established.

Testing and Validation

Bugs List and Fixes

7. Frontend Fragment Not Using Updated Models and DTOs

Ex: The RecipeDisplayFragment in the Android frontend was still creating and sending Favorites objects with outdated data types and missing fields, leading to improper communication with the backend.

Solution:

Updated the fragment to create and send FavoritesDTO objects instead of Favorites entities. Ensured that all IDs used were of type Long and matched the backend expectations, facilitating correct data transfer and persistence.

8. Inconsistent Naming Conventions and Field Definitions

Mismatches in field names and data types between frontend models and backend entities led to serialization issues, where fields expected by the backend were either missing or incorrectly named.

Solution:

Ensured that all field names in frontend models precisely matched those in backend entities. Used consistent data types across both ends to prevent serialization and mapping errors.

9. Lack of Comprehensive Logging and Error Handling

Insufficient logging made it difficult to trace the flow of data and identify where exactly failures were occurring during the add-to-favorites process.

Solution:

Implemented detailed logging in both frontend and backend to monitor incoming requests, data being sent, and any errors that occurred. This facilitated easier debugging and quicker identification of issues. Added global exception handlers in the backend to manage and respond to errors gracefully, providing meaningful feedback to the frontend.

Testing and Validation

Bugs List and Fixes

10. Frontend-Backend Integration Testing Oversights

Initial lack of thorough testing between the frontend and backend components led to undetected issues persisting through development phases.

Solution:

Utilized tools like Postman to test backend endpoints independently before integrating with the frontend. Conducted end-to-end testing within the Android app to ensure seamless communication and data persistence between the frontend and backend.

By addressing these issues systematically, we ensured that our frontend and backend were perfectly synchronized in terms of data structures, entity relationships, and data types. This comprehensive approach eliminated the `PropertyValueException` and other related errors, leading to a more stable and reliable implementation of the favorites feature in your application.

COMPETING SOFTWARE

Streamlined User Interface

YumCycle offers a user-friendly interface that prioritizes essential functionalities, making it easy for users to navigate. Unlike competitors, the design focuses on simplicity, allowing users to quickly access inventory management and recipe discovery.

Community Engagement

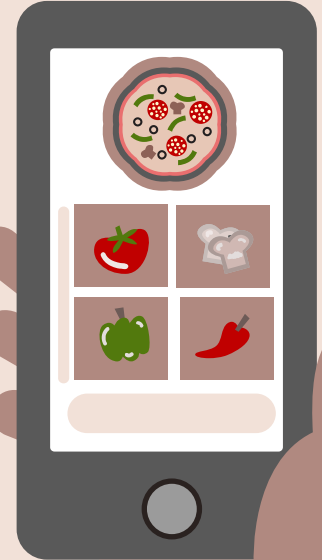
YumCycle fosters a community-driven approach, enabling users to share their personal recipes. This feature not only enhances user engagement but also enriches the recipe database, providing a diverse range of options for all users.

No Subscription Required

One of the standout features of YumCycle is that it does not require users to subscribe. This model sets it apart from competitors like Kitchen Pal, which may include subscription fees, thus making YumCycle a more accessible option for budget-conscious users.

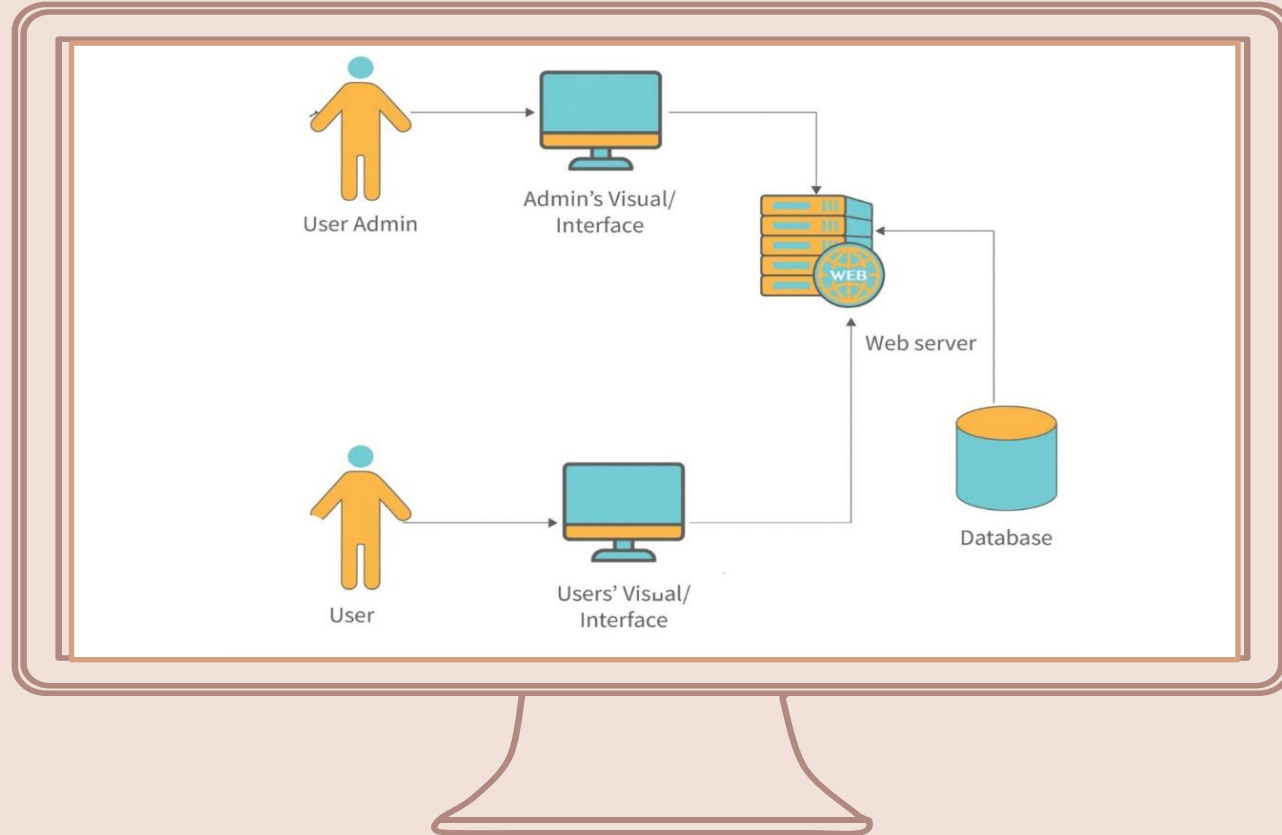
Focus on Essential Functions

While Kitchen Pal provides a plethora of features, YumCycle emphasizes a streamlined approach by honing on essential functions that help minimize food waste and grocery expenses, making it a practical choice for many households.



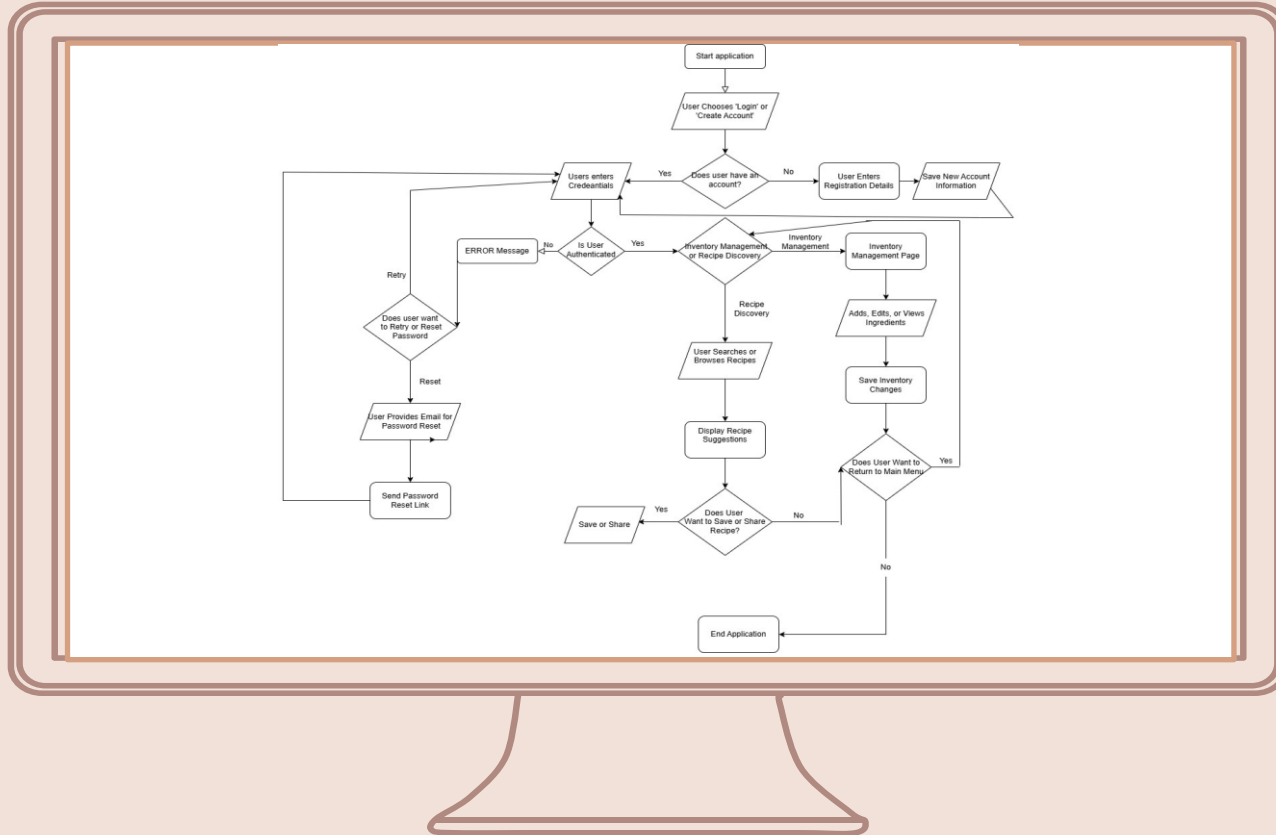
DIAGRAMS

1. General Architecture Diagram



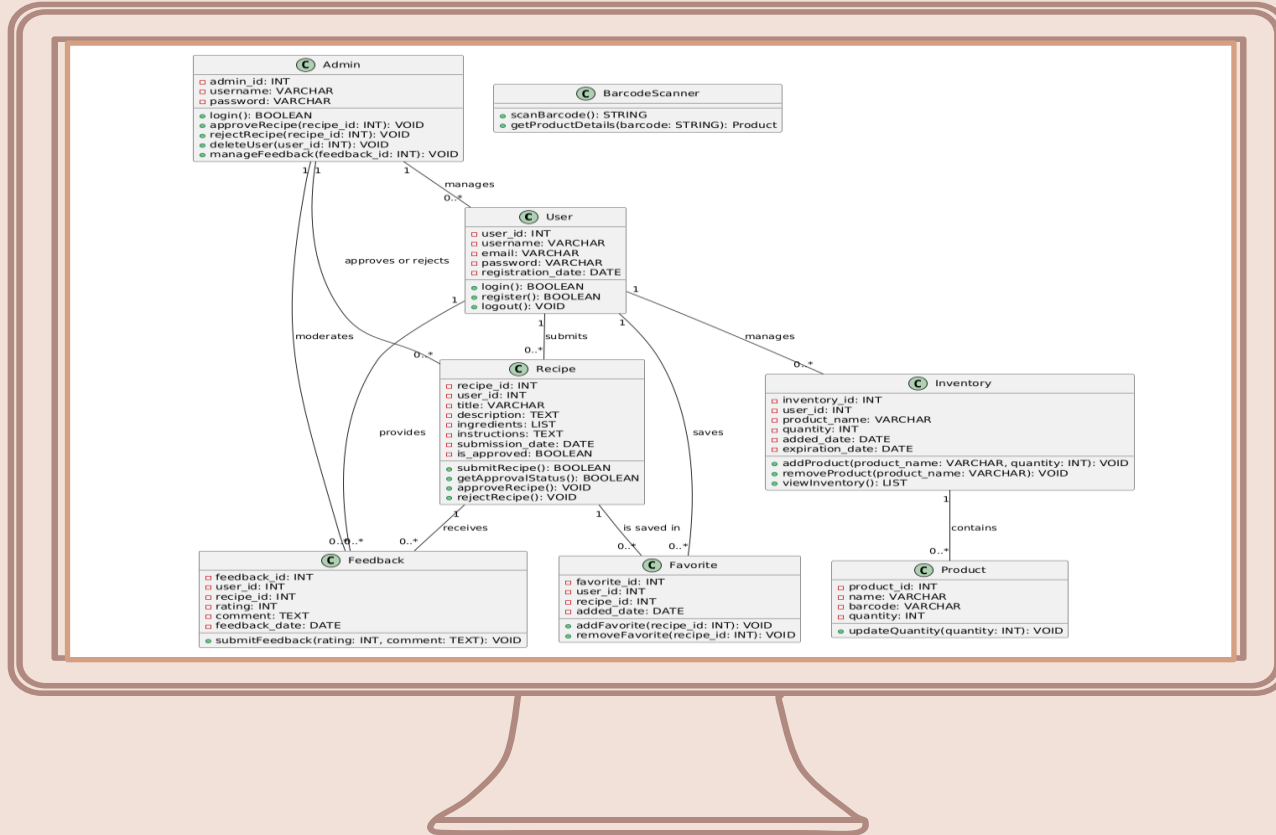
DIAGRAMS

2. Flowchart Diagram



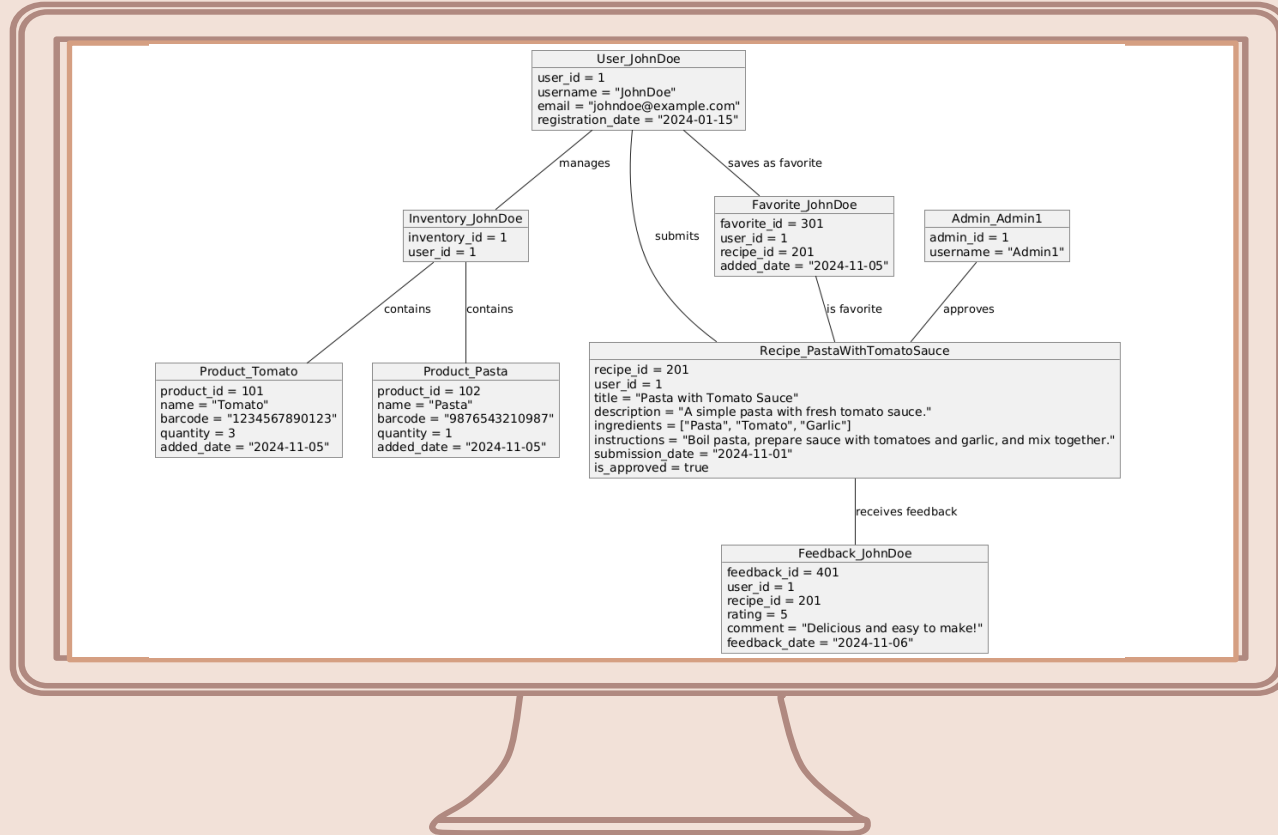
DIAGRAMS

3. Class Diagram



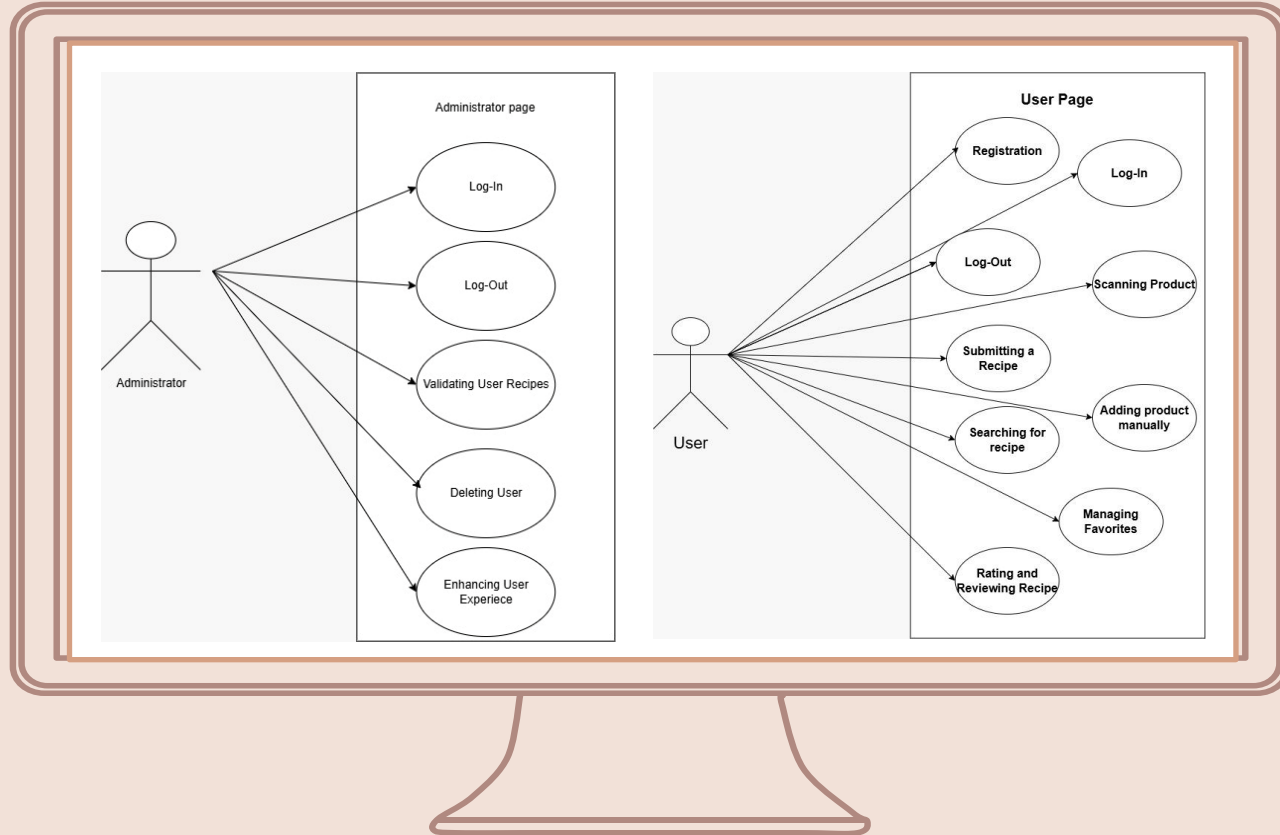
DIAGRAMS

4. Object Diagram



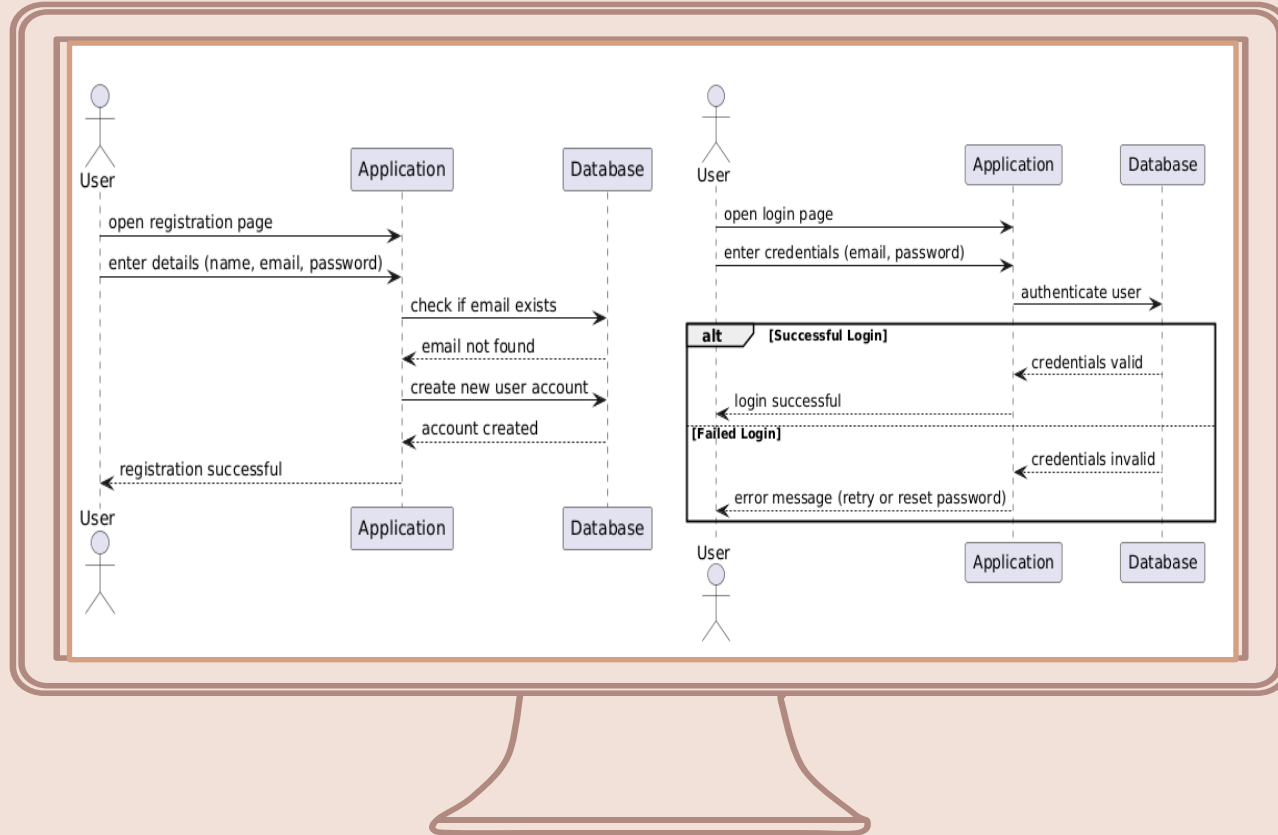
DIAGRAMS

5. Use Case Diagrams



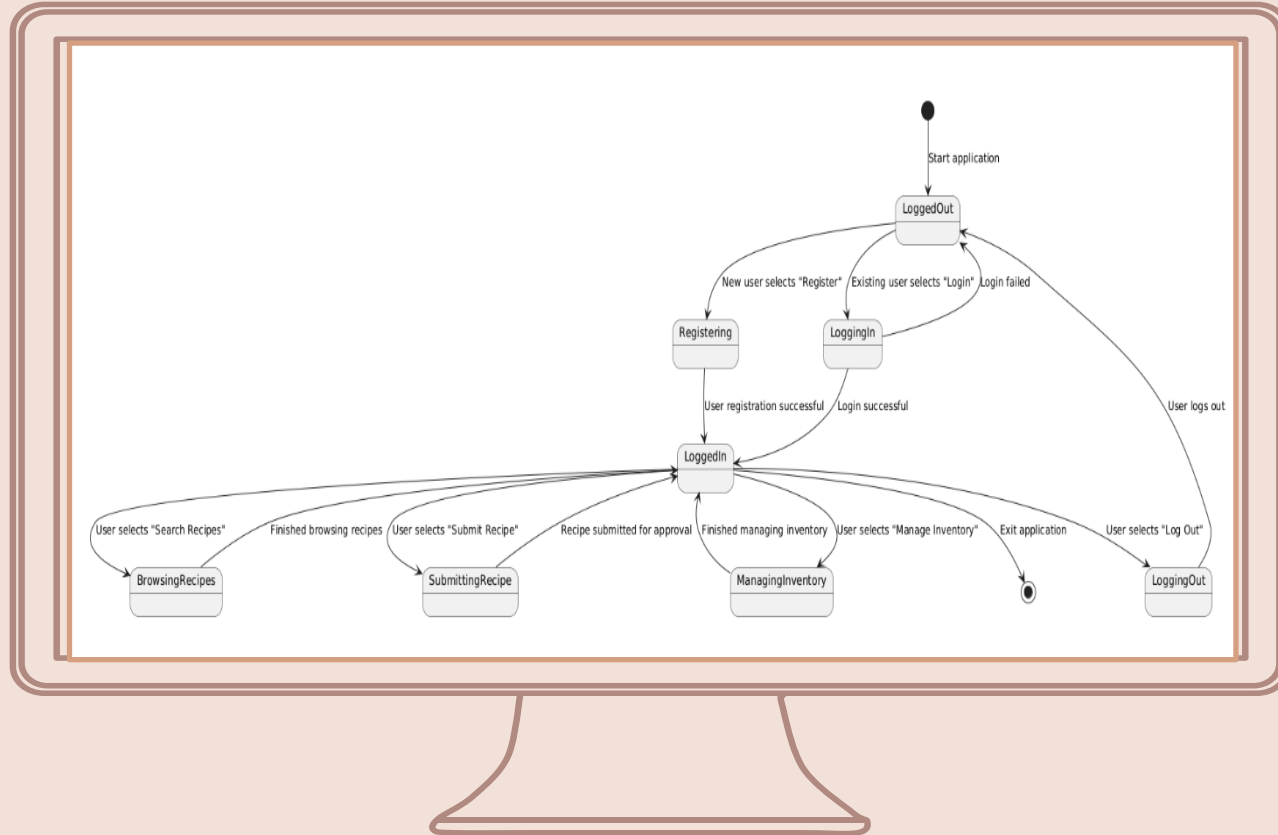
DIAGRAMS

6. Sequence Diagrams



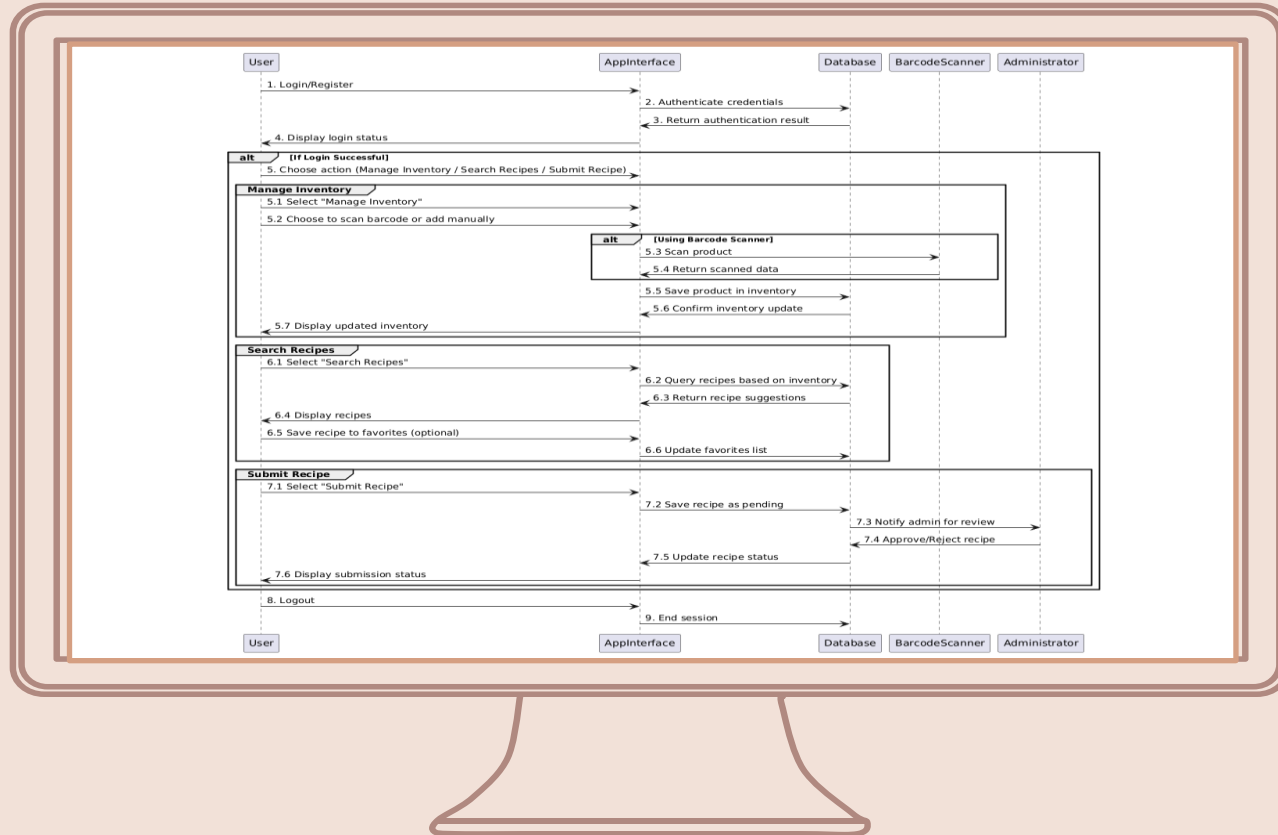
DIAGRAMS

8. State Transition Diagram



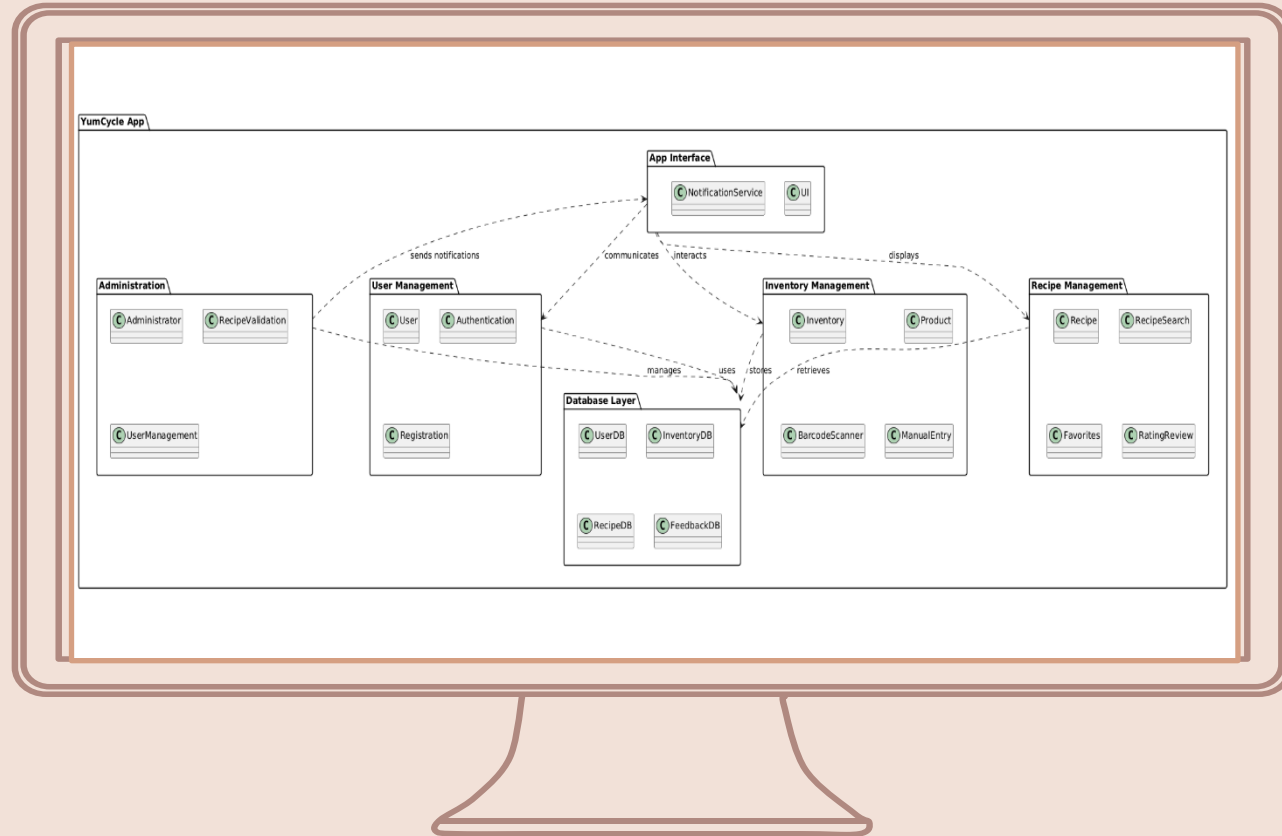
DIAGRAMS

9. Communication Diagram



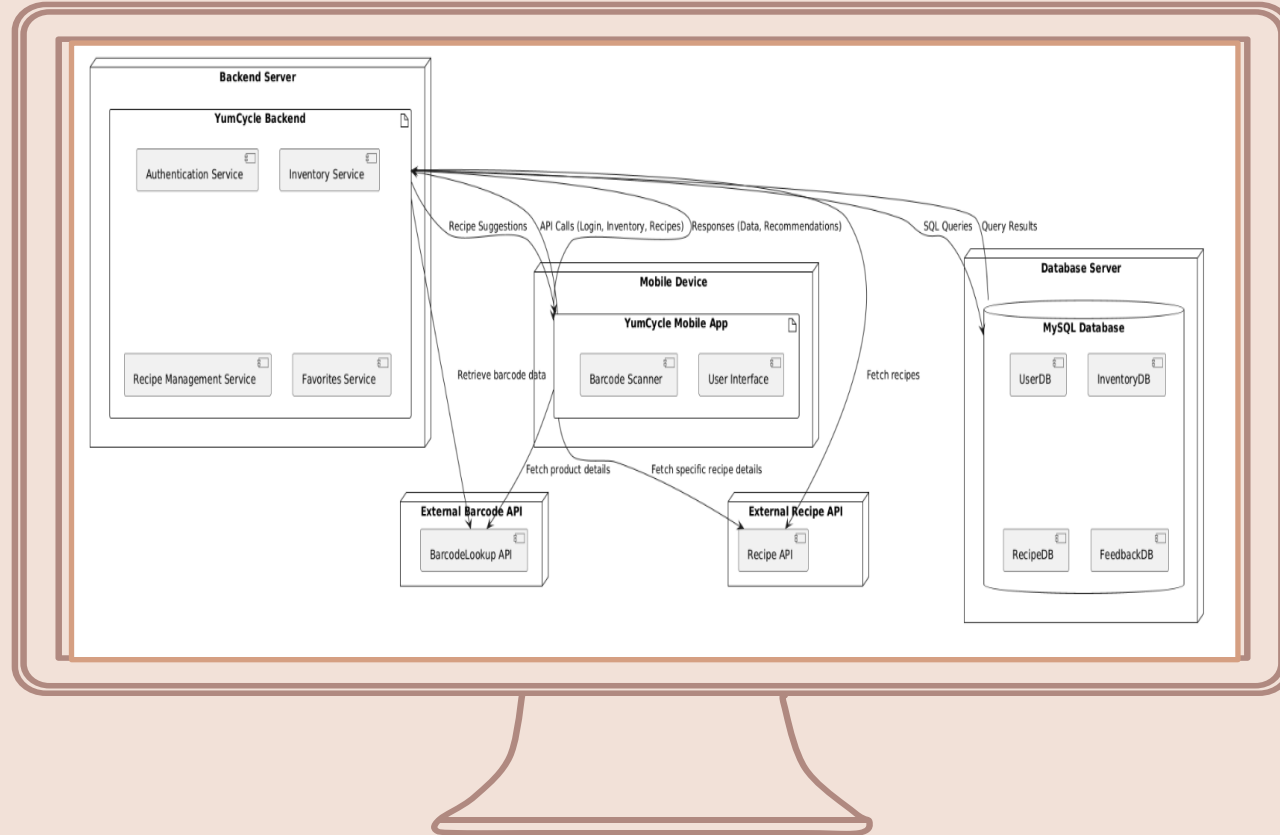
DIAGRAMS

10. Package Diagram



DIAGRAMS

11. Deployment Diagram



DIAGRAMS

12. Database Diagram

