Versions:

Date	Version	Autor	Comments
04.11.2024	1	Borzan Călina-Annemary Gozman-Pop Maria-Eliza	

Analysis Design for <YumCycle >

Contents:

Recommended: 20.11.2017	**Approved:**
Company/University Name: Address: Universitatea Tenhica Cluj-Napoca Romania	

1. General Presentation

YumCycle is a mobile application designed to help users efficiently manage the contents of their fridge and pantry, reduce food waste, and promote healthier eating habits by suggesting recipes based on ingredients users already have. Developed by a team inspired by personal experiences with food waste, YumCycle combines inventory management with recipe discovery, simplifying food tracking and planning while reducing unnecessary purchases. The primary features of YumCycle include a barcode-scanning feature that allows users to quickly add items to their virtual fridge by scanning product barcodes, and a manual entry option for items without barcodes. Users can also receive personalized recipe suggestions based on their current inventory. Additionally, YumCycle has a recipe-sharing component where users can submit their own recipes, which are then validated by an administrator before being added to a shared recipe database. With these tools, YumCycle empowers users to make the most of their food inventory, save money, and reduce environmental impact.

2. Theoretical Fundamentals

The development of YumCycle incorporates several fundamental principles in software design and user experience:

- **User-Centric Design**: YumCycle's features are centred around the user's needs, enabling them to manage inventory with minimal input and maximize the utility of their existing food items. Features like recipe suggestions create a seamless experience that directly benefits users' daily lives.
- **Inventory Management**: The concept of digital inventory management is central to YumCycle, allowing users to efficiently track food items. This management feature is designed to be user-friendly and leverages a database backend to store details about each item, including name and quantity with barcode scanning to make inventory updates easier and faster.
- **Recipe Discovery and Sharing**: The app's recipe component utilizes an algorithm that matches users' ingredients with available recipes, supporting customized and relevant recipe suggestions. This feature encourages engagement by allowing users to submit and share their favourite recipes, contributing to a broader user-driven database of recipe options.
- **Moderation and Feedback Mechanisms**: An essential part of the YumCycle ecosystem is moderation. The administrator role involves validating recipes to ensure quality and prevent inappropriate content. This system promotes reliability in the recipe database while offering feedback and improvement opportunities through user ratings and reviews.

3. IT Technology

Given that YumCycle is built using Java, Spring Boot, and a MySQL database, the following components of the application architecture can be outlined:

- **Backend Development with Java and Spring Boot**:
  - Java: A robust and scalable programming language, Java is widely used for building mobile and backend applications due to its security, cross-platform capabilities, and compatibility with Android, which is the target platform for YumCycle.
  - Spring Boot: This Java-based framework simplifies application development by providing out-of-the-box configurations for commonly used features like RESTful APIs, security, and database management. For YumCycle, Spring Boot is ideal for handling backend operations such as user authentication, recipe validation, and inventory management, allowing for a modular, scalable application structure.
  - RESTful API Development: Spring Boot's REST API support facilitates seamless data exchange between the frontend and backend. For example, when a user scans a barcode or submits a recipe, these interactions are handled through RESTful API calls, making the backend highly responsive and efficient.
- **Database Management with MySQL**:
  - MySQL Database: The relational database stores critical information such as user accounts, inventory items, recipes, and reviews. MySQL is known for its reliability, scalability, and ease of integration with Java-based applications, making it a suitable choice for managing YumCycle's data-intensive operations.
  - Entity-Relationship Modeling: The data model for YumCycle would likely consist of several tables representing entities like User, InventoryItem, Recipe, Feedback, and Favorites, among others. Relationships between these tables allow for complex querying, such as finding recipes based on ingredients, tracking user preferences, and managing administrative actions.
  - Data Access Layer (DAO): In Spring Boot, the Data Access Object (DAO) layer manages database interactions. Through the Spring Data JPA module, developers can use Java classes to interact with database tables, making data management straightforward and minimizing the need for manual SQL queries.

- **Frontend and User Interface**:
  - Mobile-First Design: YumCycle's interface is optimized for mobile devices, focusing on intuitive navigation, large touch targets, and streamlined flows for tasks like adding items, viewing recipes, and managing inventory.
  - Barcode Scanning Integration: Barcode scanning in the app can be implemented using libraries compatible with Android, such as ZXing or ML Kit, which work with the device's camera to identify and translate barcode data into actionable information.

4. Functionalities

The YumCycle application includes a variety of functionalities to help users manage their fridge and pantry contents, reduce food waste, and find relevant recipes. Here's a breakdown of its main functionalities:

- **User Registration and Authentication**:
  - Sign Up / Registration: New users can create accounts by entering personal details (e.g., name, email, password).
  - Log In / Log Out: Existing users can securely log in to access their saved inventory and recipes, clearing session data to prevent unauthorized access.

- **Inventory Management**:
  - Barcode Scanning: Users can add items to their inventory by scanning product barcodes, which automatically retrieves details like name from a connected database.
  - Manual Entry: For items without barcodes, users can manually enter details, including product name, and quantity.
  - Viewing and Organizing Inventory: Users can view a list of items in their inventory, sorted by categories like product type, or custom tags.

- **Recipe Management**:
  - Recipe Discovery: Users can search for recipes based on ingredients in their inventory, dietary preferences, or keywords. The app suggests recipes that use ingredients the user already has to minimize additional shopping.
  - Recipe Submission: Users can submit their own recipes to the app's recipe database, which are reviewed by the administrator for approval before becoming available to the community.
  - Favourites: Users can save favourite recipes for easy access in the future, as well as manage their preferred recipes list.

- **Feedback and Ratings**:
  - Rating and Reviewing Recipes: After trying a recipe, users can rate and review it to help other users determine recipe quality and reliability. Ratings and reviews contribute to the recipe's overall ranking in the app.
  - User Feedback for App Improvement: Users can provide feedback on their experience with the app itself, which administrators can use to enhance functionality and user experience.

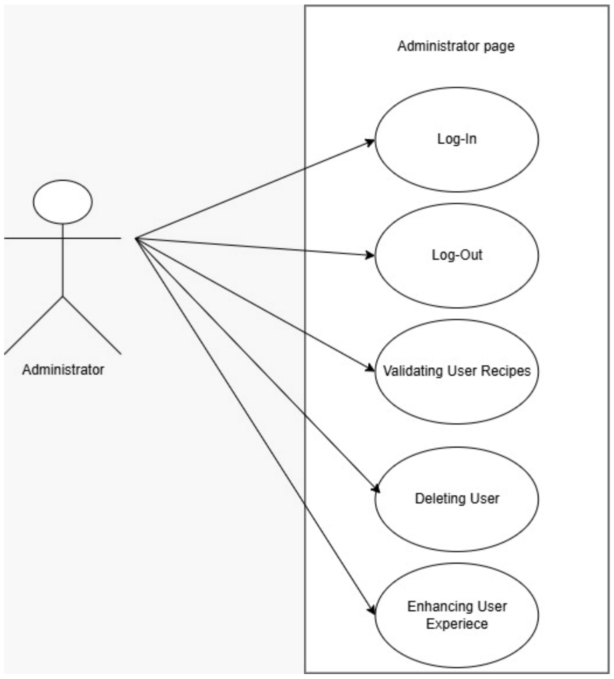
- **Administrator Functionalities**:
  - Recipe Validation: The administrator reviews user-submitted recipes to ensure they meet the app's quality and content guidelines. Approved recipes are published in the database, while unapproved recipes are returned with feedback.
  - User Management: Administrators can delete user accounts if necessary due to misuse, spam, or violations of app guidelines.
  - App Improvement Management: The administrator reviews user feedback and app analytics to plan and implement improvements that enhance the overall user experience.

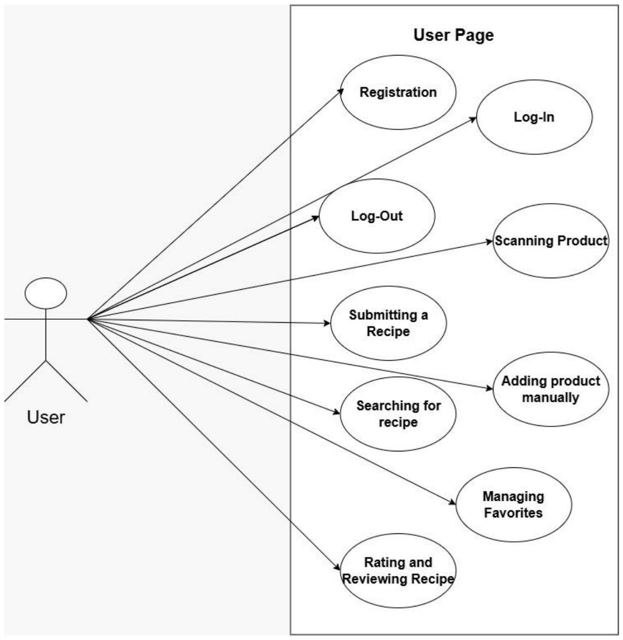
5. Actors and related access rights

- **Administrator**
  Access Rights:
    - Full Access: Administrators have complete access to all app features, including those restricted from general users, like user and recipe management.
    - Recipe Moderation: Administrators can review, approve, or reject recipes submitted by users. They can provide feedback on rejected recipes for potential improvement.
    - User Management: Administrators have access to delete user accounts if necessary. This includes full control over the user's data and inventory management privileges.
    - App Improvement: Administrators can access and analyse user feedback and app usage data to make decisions about updates and improvements.
- **User**
  Access Rights:
    - Inventory Management: Users have full access to manage their personal inventory, including adding, viewing, and deleting items via barcode scanning or manual entry.
    - Recipe Management: Users can search for, view, rate, and review recipes in the database. They can also submit their own recipes, though these submissions require administrator approval before publication.
    - Personal Preferences: Users can customize settings such as notification preferences for expiring items, dietary restrictions, and recipe discovery preferences.
    - Feedback and Review: Users can rate and review recipes they have tried, contribute feedback about the app, and manage their list of favourite recipes and saved items.

These access rights are designed to ensure that users can personalize and engage with the app while the administrator maintains oversight and quality control.

6. Use Case Diagrams





7. System Architecture

The YumCycle application follows a multi-layered architecture, typically structured as a three-tier architecture to separate concerns and ensure modularity, scalability, and maintainability. The system architecture is organized into the following layers:

- Presentation Layer (Frontend)
- Application Layer (Backend)
- Data Layer (Database)

7.1. Presentation Layer (Frontend)

The presentation layer serves as the user interface for the YumCycle mobile application. It is responsible for displaying the application's functionality, receiving user inputs, and sending requests to the backend. It also handles the barcode scanning feature and provides a responsive and intuitive user experience.
Components:

- Mobile Application Interface: The app is designed primarily for mobile devices, with a user-friendly interface that enables users to easily navigate features like inventory management, recipe discovery.
- Barcode Scanning: The app uses a barcode scanning library (e.g., ZXing or Google ML Kit for Android) to enable users to scan products. This component interacts with the backend to retrieve product details based on barcode data.

Technologies:

- Android for mobile app development (Java/Kotlin).
- Barcode Scanning SDK such as ZXing or ML Kit for barcode scanning capabilities.

7.2. Application Layer (Backend)

The application layer is the core of the YumCycle system, developed using Java and Spring Boot to manage the application's business logic, RESTful APIs, and data processing. This layer acts as a bridge between the frontend and the database, handling data retrieval, validation, and processing before returning results to the frontend.
Components:

- RESTful API: Spring Boot serves as a RESTful service layer, allowing the frontend to communicate with the backend through HTTP requests. This API manages tasks like user authentication, inventory updates, and recipe searches.

- User Authentication and Authorization: Handles secure logins, session management, and access control for both user and administrator roles.
- Inventory Management Services: Manages the addition and retrieval of items in the user's inventory.
- Recipe Management and Approval Workflow: Includes services for users to submit recipes and for administrators to review, approve, or reject these submissions.
- Data Validation and Error Handling: Ensures that all inputs (e.g., barcode scans, manual entries, recipe submissions) are validated to prevent errors or inconsistencies in the application.

Technologies:

- Java for the backend logic.
- Spring Boot for building the REST API and managing dependency injection, data processing, and business logic.
- Spring Security for authentication and access control.
- JSON for data format in API requests and responses.

---

2. Data Layer (Database)

The data layer is responsible for persistent storage, housing user data, inventory records, recipes, and user feedback. MySQL is used as the primary relational database, chosen for its reliability, ease of integration with Java-based applications, and efficient handling of structured data.

Components:

- User Data Management: Stores information such as user profiles, login credentials, preferences, and inventory settings.
- Inventory Records: Tracks items added by each user, including product names, quantities, and associated barcodes.
- Recipe Database: Houses all recipes, including user-submitted and approved recipes, along with ratings, reviews, and any categorization tags.
- Feedback and Reviews: Manages user reviews, recipe ratings, and general app feedback, enabling administrators to monitor user satisfaction.
- Relationships and Constraints: Enforces referential integrity and relationship mappings to maintain data consistency, such as linking inventory items to users and recipes to reviews.

Technologies:
- MySQL for relational data storage and management.
- Spring Data JPA as an ORM (Object-Relational Mapping) layer, allowing Java classes to interact directly with database tables.
- ER Diagrams to model relationships between tables, such as users, inventory items, recipes, and reviews.

Inter-Layer Communication and Workflow

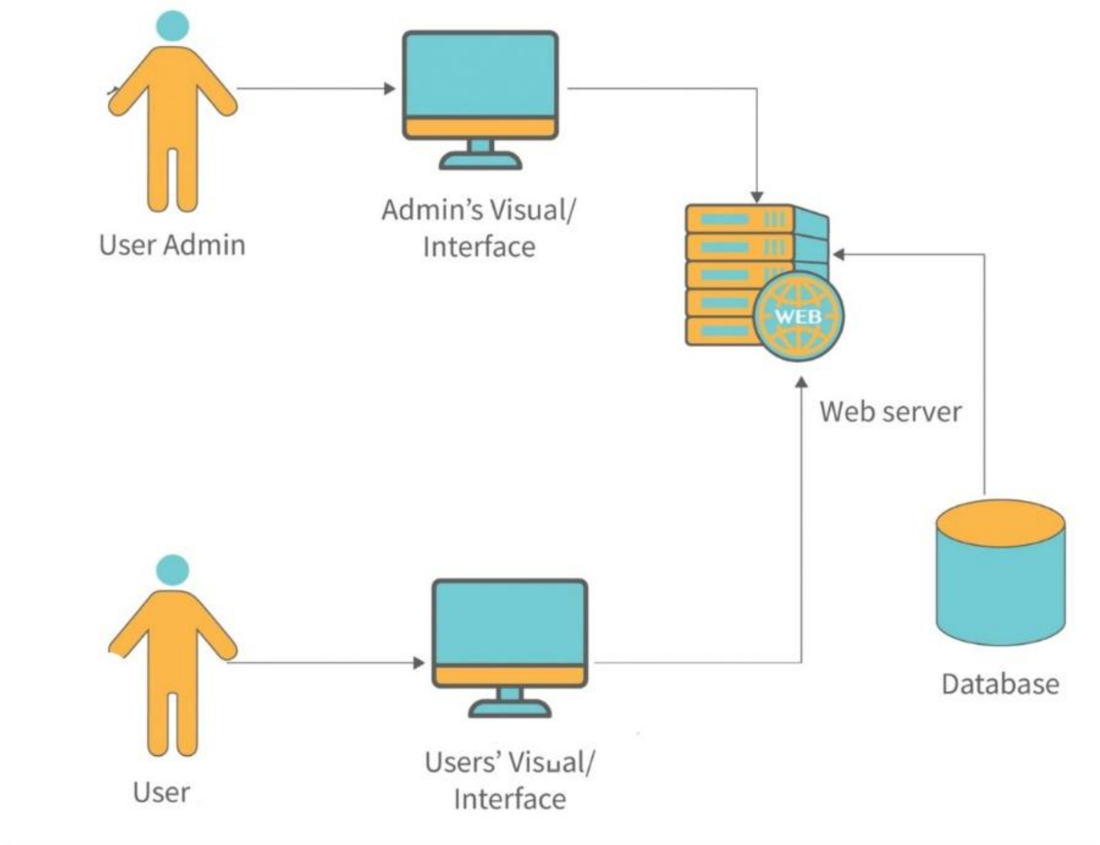
- User Action Initiation: A user initiates an action (e.g., scanning a barcode, adding an item) on the frontend.
- API Call to Backend: The frontend sends a REST API request to the backend to perform the desired operation. For example, if the user scans a barcode, the backend queries the barcode database for product information.
- Data Processing in Backend: The backend processes the request, interacts with the MySQL database if necessary, and applies business logic (e.g., validating recipes).
- Response to Frontend: The backend sends a response to the frontend with the results, such as the item details retrieved or a confirmation message.

Additional Considerations

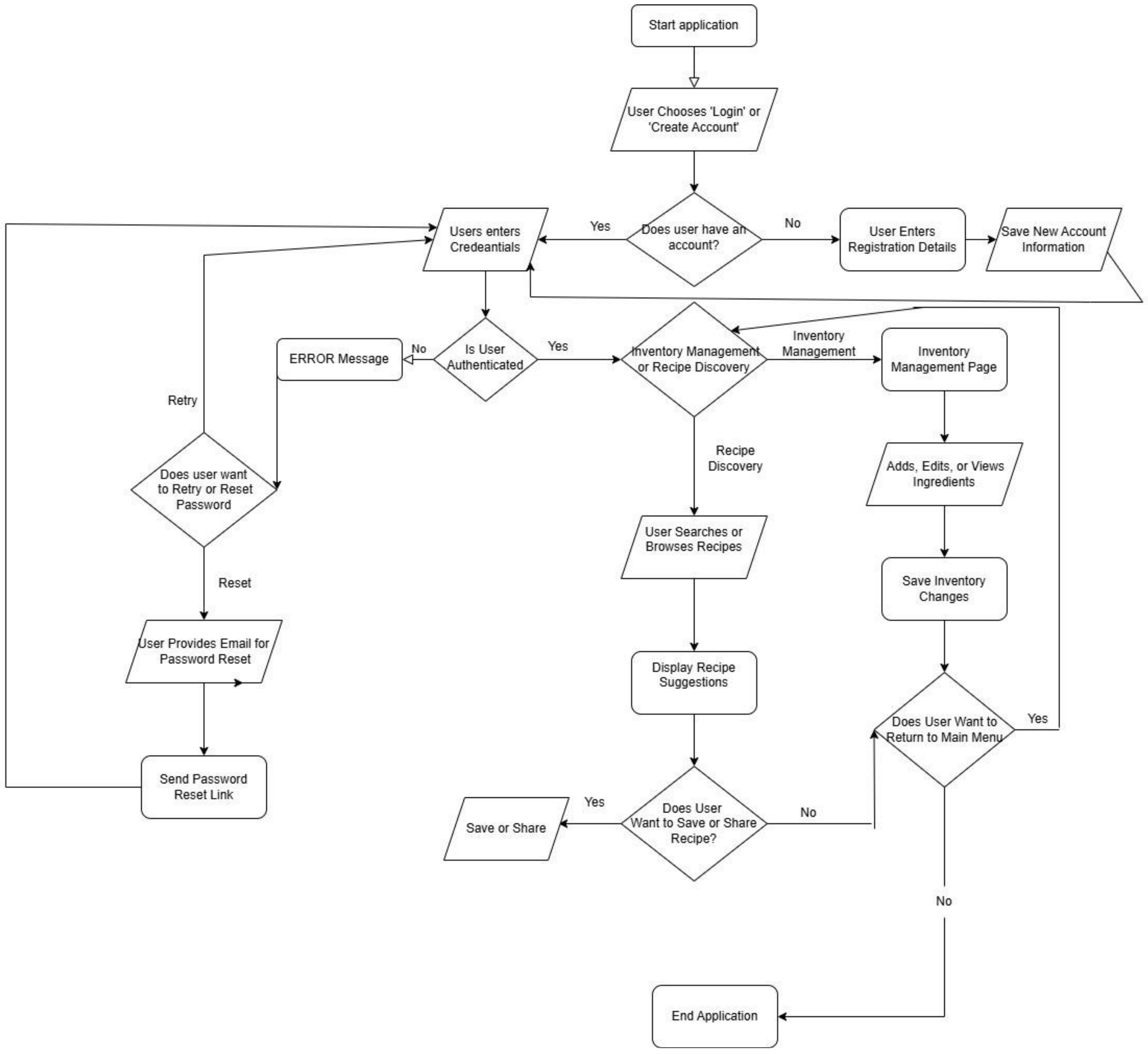
- Scalability: The architecture can be scaled by deploying the application on cloud platforms like AWS or Google Cloud, using load balancers and microservices to handle high user loads.
- Security: Spring Security and JWT (JSON Web Tokens) for secure user authentication, role-based access control, and data encryption ensure that user data is protected.
- Future Expansion: The architecture is designed to allow the integration of additional services, such as machine learning for personalized recipe recommendations or cloud storage for images in recipe submissions.

8. Design (approx. 12 UML Diagrams!!!)

8.1. General Architecture Diagram

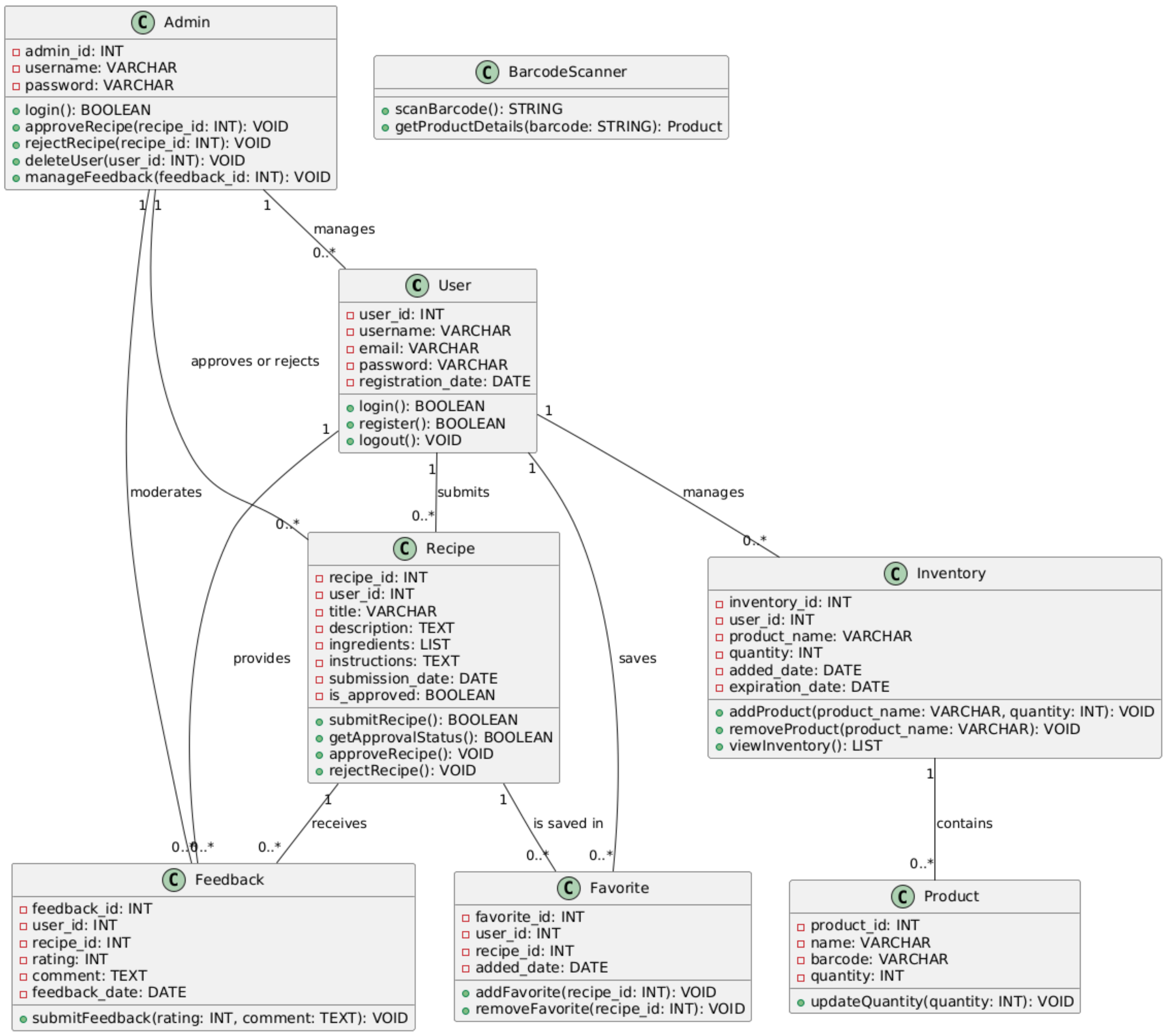


8.2 Flowchart



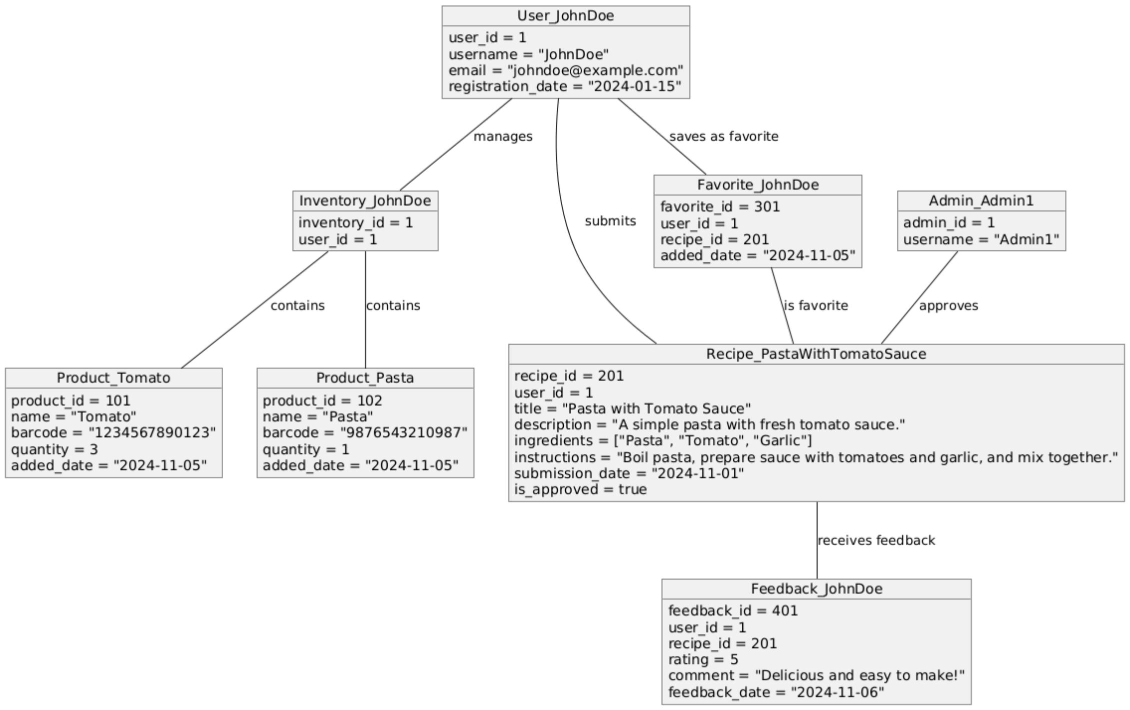
Start application
User Chooses 'Login' or 'Create Account'
Does user have an account?
Yes
No
Users enters Credeantials
User Enters Registration Details
Save New Account Information
Is User Authenticated
No
Yes
ERROR Message
Inventory Management or Recipe Discovery
Inventory Management
Inventory Management Page
Retry
Does user want to Retry or Reset Password
Reset
User Provides Email for Password Reset
Send Password Reset Link
Recipe Discovery
User Searches or Browses Recipes
Display Recipe Suggestions
Adds, Edits, or Views Ingredients
Save Inventory Changes
Does User Want to Return to Main Menu
Yes
Does User Want to Save or Share Recipe?
Yes
No
Save or Share
No
End Application

8.3 Class Diagram



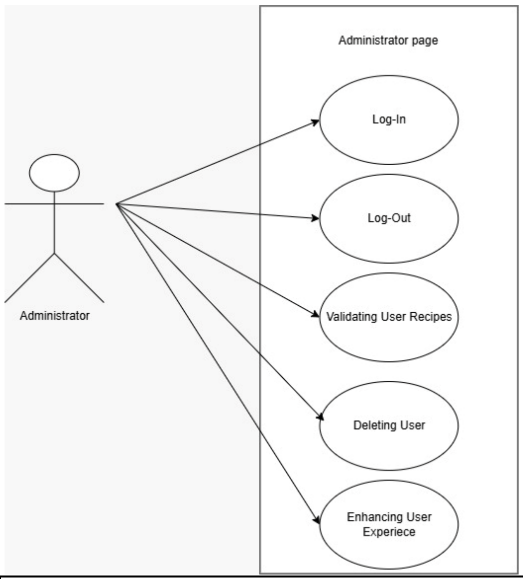
Recommended: 20.11.2017	**Approved:**
Company/University Name: Address: Universitatea Tenhica Cluj-Napoca Romania	

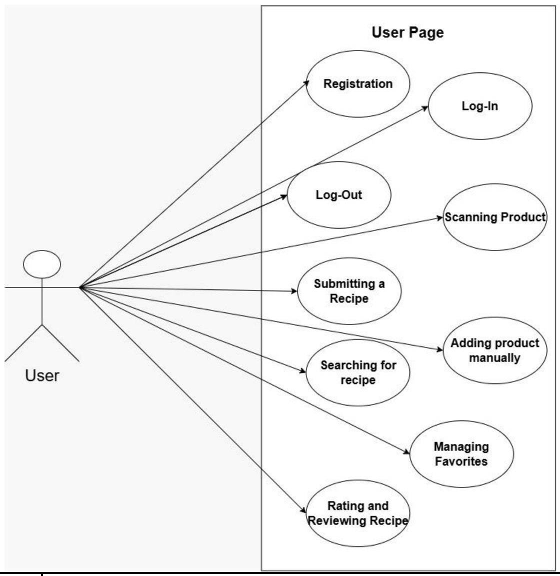
8.4 Object Diagram



User_JohnDoe
user_id = 1
username = "JohnDoe"
email = "johndoe@example.com"
registration_date = "2024-01-15"
manages
submits
saves as favorite
Inventory_JohnDoe
inventory_id = 1
user_id = 1
Favorite_JohnDoe
favorite_id = 301
user_id = 1
recipe_id = 201
added_date = "2024-11-05"
Admin_Admin1
admin_id = 1
username = "Admin1"
contains
contains
is favorite
approves
Product_Tomato
product_id = 101
name = "Tomato"
barcode = "1234567890123"
quantity = 3
added_date = "2024-11-05"
Product_Pasta
product_id = 102
name = "Pasta"
barcode = "9876543210987"
quantity = 1
added_date = "2024-11-05"
Recipe_PastaWithTomatoSauce
recipe_id = 201
user_id = 1
title = "Pasta with Tomato Sauce"
description = "A simple pasta with fresh tomato sauce."
ingredients = ["Pasta", "Tomato", "Garlic"]
instructions = "Boil pasta, prepare sauce with tomatoes and garlic, and mix together."
submission_date = "2024-11-01"
is_approved = true
receives feedback
Feedback_JohnDoe
feedback_id = 401
user_id = 1
recipe_id = 201
rating = 5
comment = "Delicious and easy to make!"
feedback_date = "2024-11-06"

8.5 Use Cases

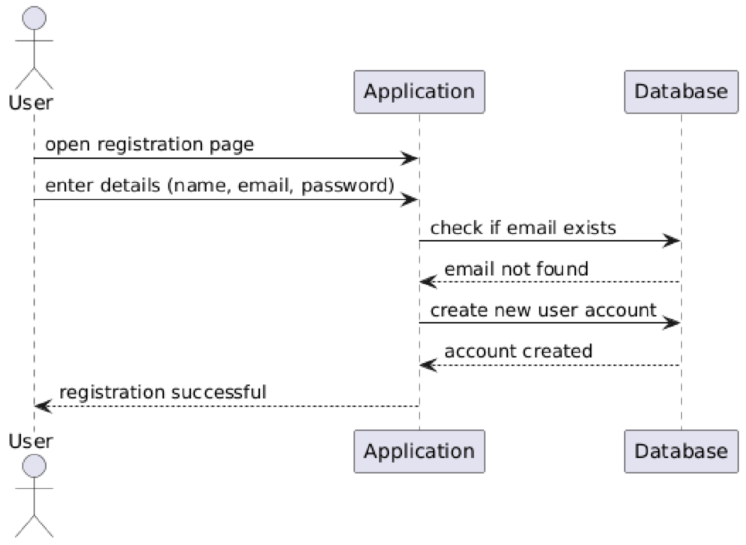




8.6 Sequence Diagram

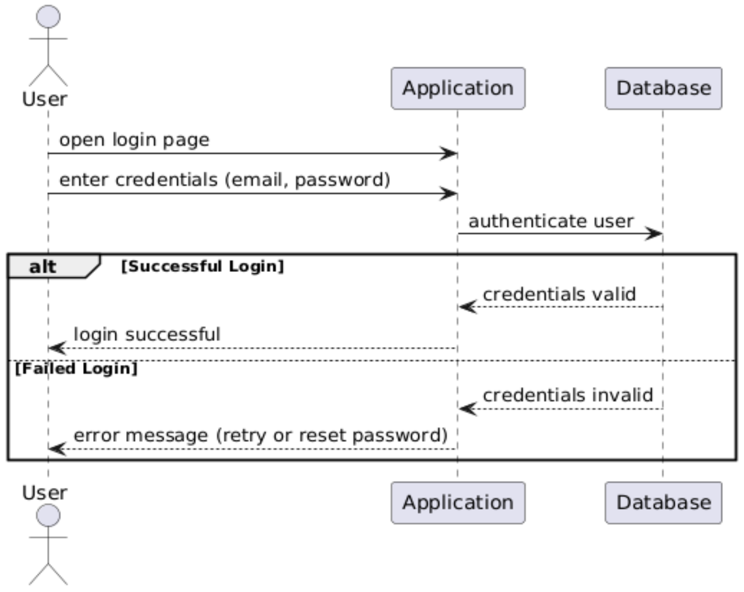
8.6.1. User Registration Sequence Diagram

This diagram shows the flow when a new user registers an account.



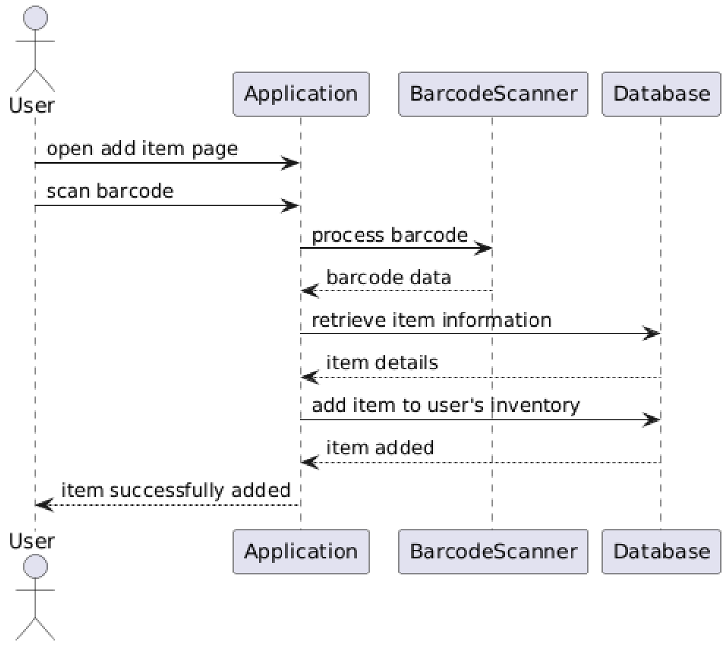
8.6.2 User Login Sequence Diagram

This diagram demonstrates the process of a user logging into the app.



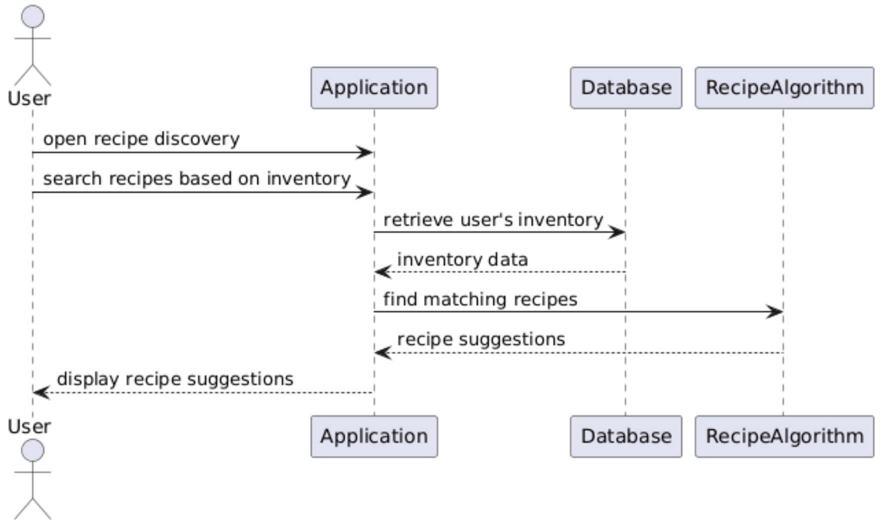
8.6.3 Inventory Management - Adding an Item with Barcode

This sequence shows a user adding an item to their inventory by scanning a barcode.

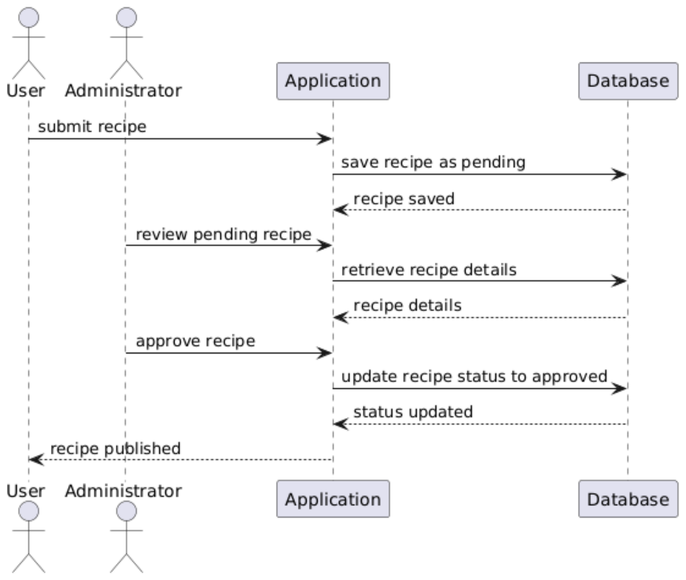


8.6.4 Recipe Discovery Sequence Diagram

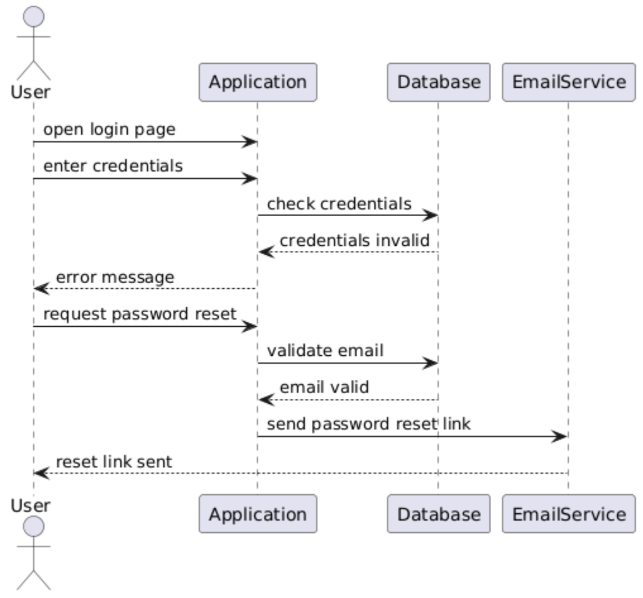
This diagram shows the interaction flow for a user searching for recipes based on their current inventory.



8.6.5 **Recipe Submission and Validation Sequence Diagram**
This sequence demonstrates a user submitting a recipe, which is then reviewed and approved by an administrator.

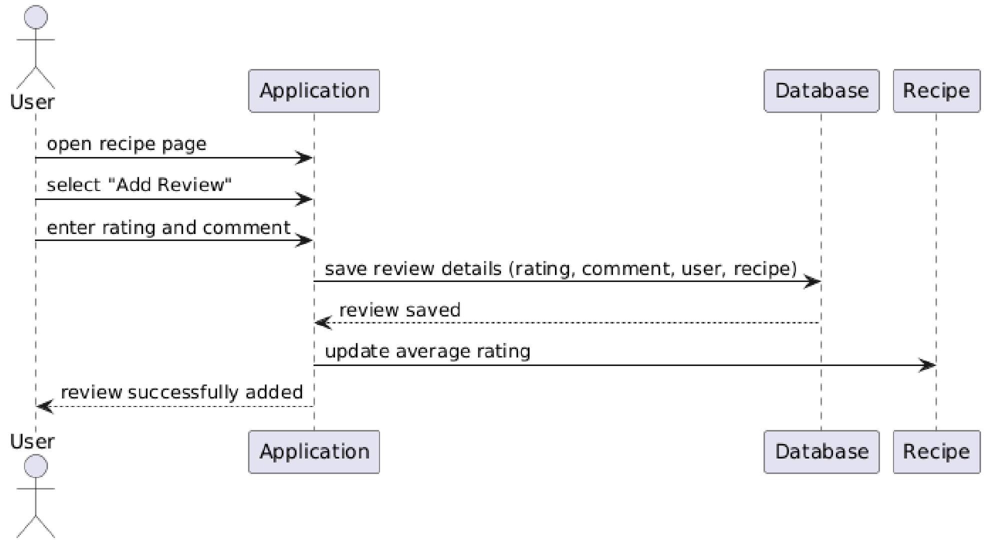


8.6.6 **Password Reset Sequence Diagram**
This diagram shows the process of resetting a password after a failed login attempt.



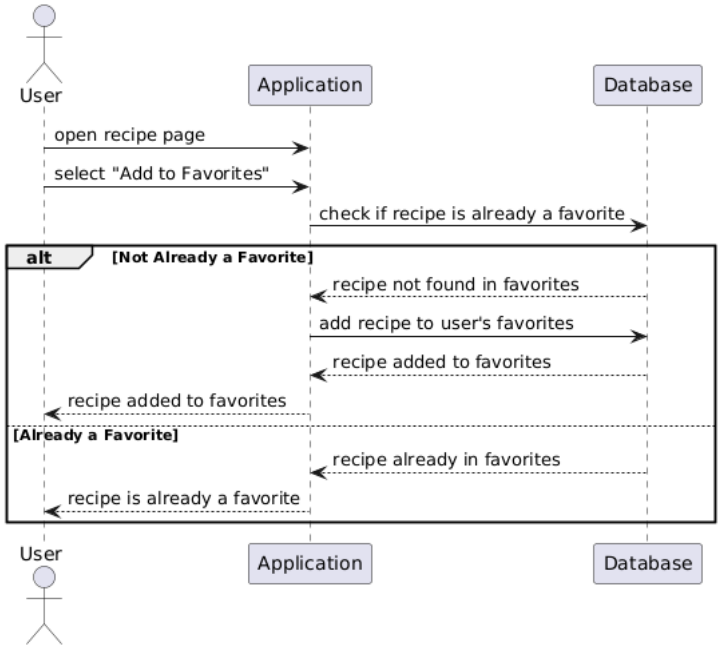
8.6.7 Adding a Review Sequence Diagram

This sequence diagram shows the process of a user adding a review for a recipe they have tried.

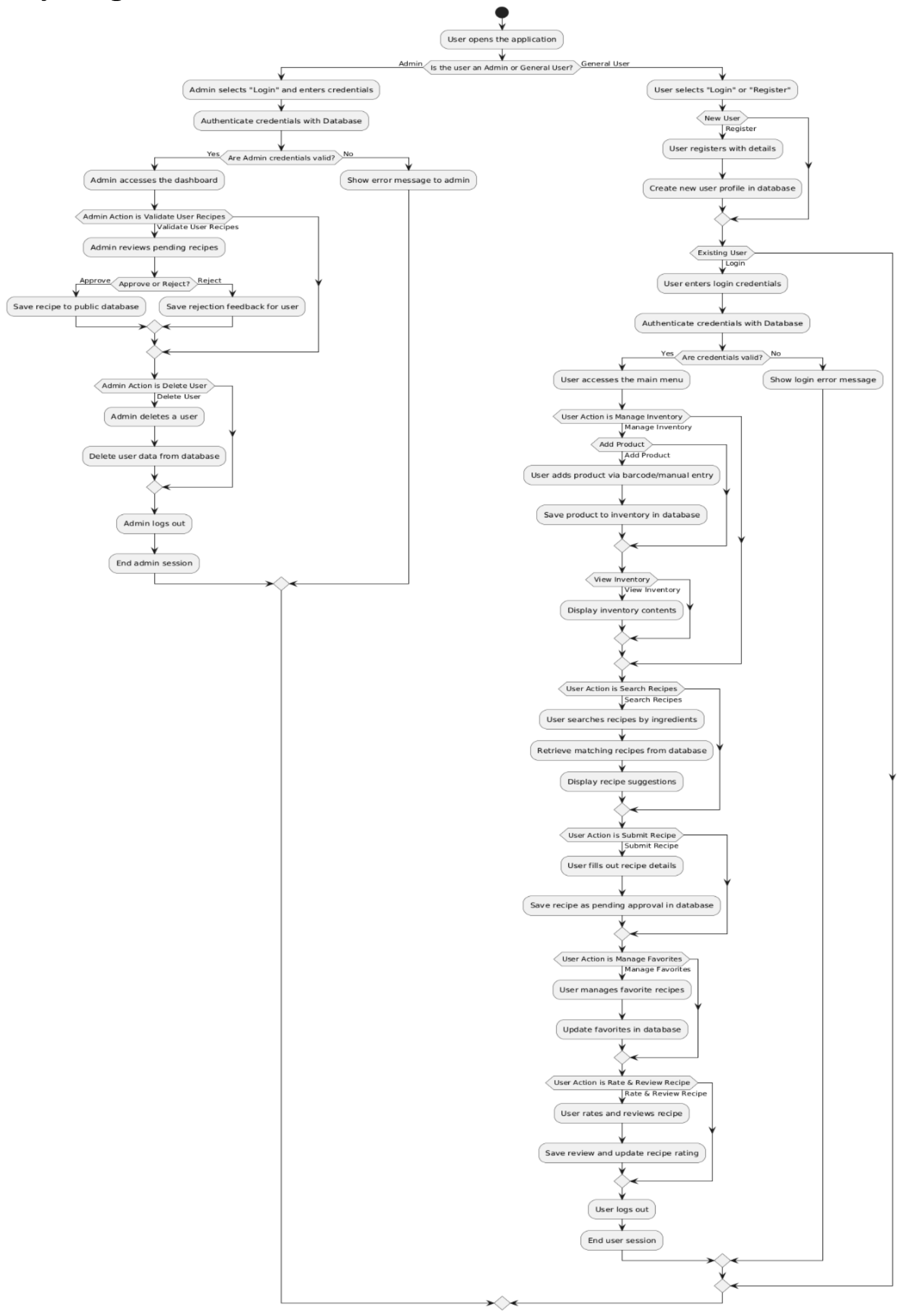


8.6.8. Adding a Recipe as Favorite Sequence Diagram

This sequence diagram illustrates a user marking a recipe as a favorite for easy access later.


User
Application
Database
open recipe page
select "Add to Favorites"
check if recipe is already a favorite
alt [Not Already a Favorite]
recipe not found in favorites
add recipe to user's favorites
recipe added to favorites
recipe added to favorites
[Already a Favorite]
recipe already in favorites
recipe is already a favorite
User
Application
Database

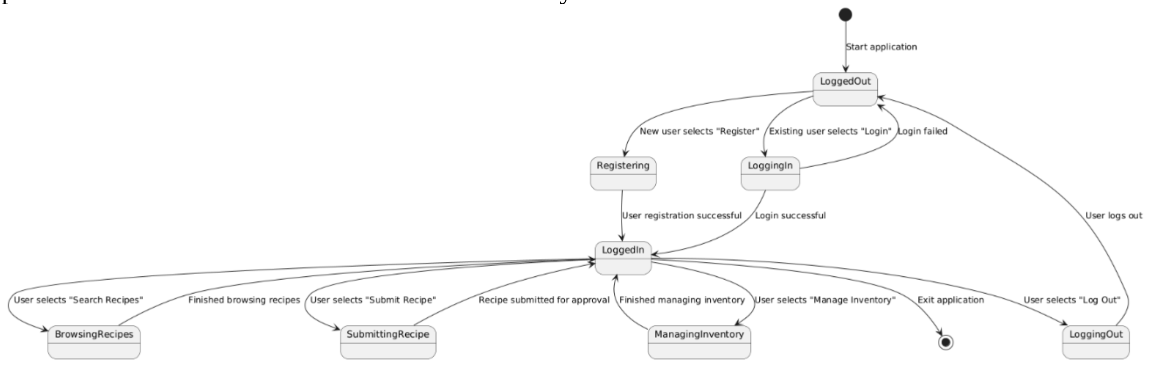
8.7 Activity Diagram



User opens the application
Admin
Is the user an Admin or General User?
General User
Admin selects "Login" and enters credentials
Authenticate credentials with Database
Yes
Are Admin credentials valid?
No
Admin accesses the dashboard
Show error message to admin
Admin Action is Validate User Recipes
Validate User Recipes
Admin reviews pending recipes
Approve
Approve or Reject?
Reject
Save recipe to public database
Save rejection feedback for user
Admin Action is Delete User
Delete User
Admin deletes a user
Delete user data from database
Admin logs out
End admin session
User selects "Login" or "Register"
New User
Register
User registers with details
Create new user profile in database
Existing User
Login
User enters login credentials
Authenticate credentials with Database
Yes
Are credentials valid?
No
User accesses the main menu
Show login error message
User Action is Manage Inventory
Manage Inventory
Add Product
Add Product
User adds product via barcode/manual entry
Save product to inventory in database
View Inventory
View Inventory
Display inventory contents
User Action is Search Recipes
Search Recipes
User searches recipes by ingredients
Retrieve matching recipes from database
Display recipe suggestions
User Action is Submit Recipe
Submit Recipe
User fills out recipe details
Save recipe as pending approval in database
User Action is Manage Favorites
Manage Favorites
User manages favorite recipes
Update favorites in database
User Action is Rate & Review Recipe
Rate & Review Recipe
User rates and reviews recipe
Save review and update recipe rating
User logs out
End user session

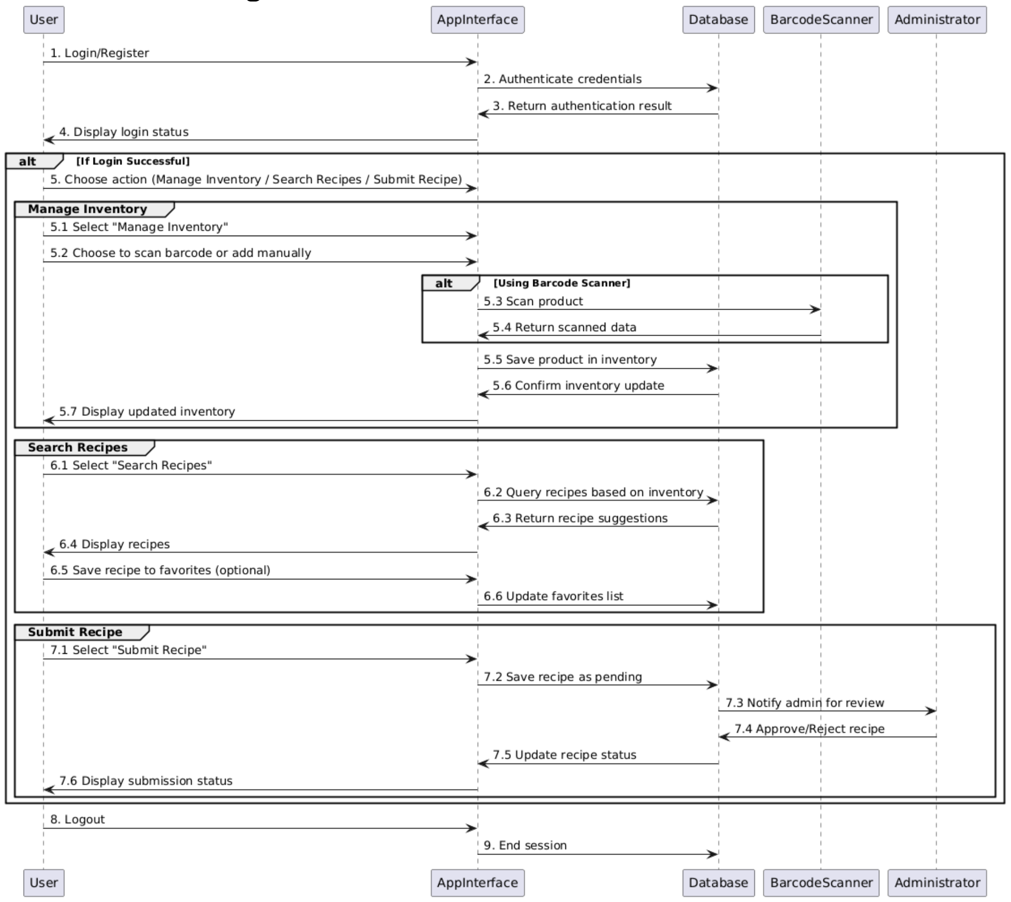
8.8 State Transition Diagram

In the case of my application, this type of diagram helps visualize the user's flow through various parts of the app and how their actions affect their state within the system.



Start application
LoggedOut
New user selects "Register"
Existing user selects "Login"
Login failed
Registering
LoggingIn
User registration successful
Login successful
User logs out
LoggedIn
User selects "Search Recipes"
Finished browsing recipes
User selects "Submit Recipe"
Recipe submitted for approval
Finished managing inventory
User selects "Manage Inventory"
Exit application
User selects "Log Out"
BrowsingRecipes
SubmittingRecipe
ManagingInventory
LoggingOut

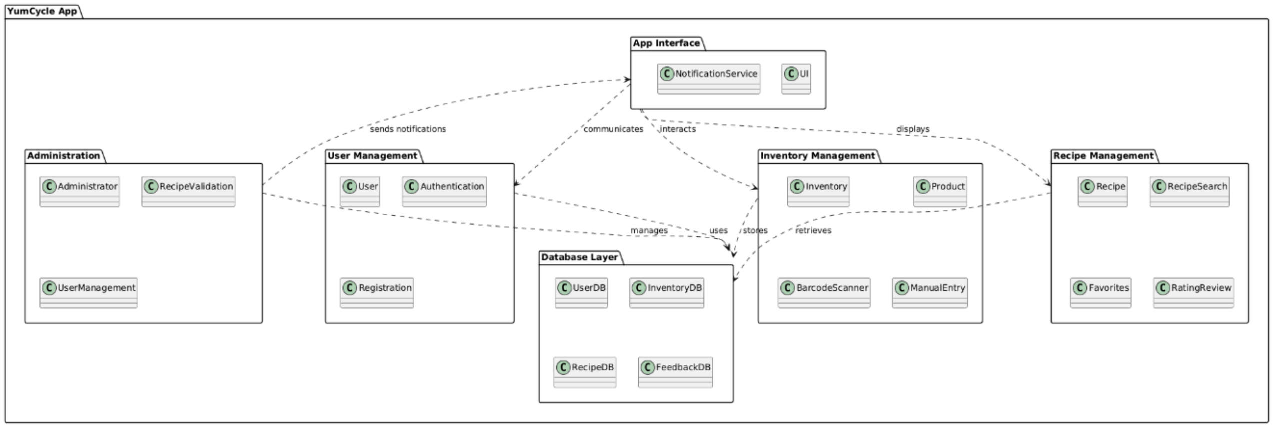
8.9 Communication diagram



User
AppInterface
Database
BarcodeScanner
Administrator
1. Login/Register
2. Authenticate credentials
3. Return authentication result
4. Display login status
alt [If Login Successful]
5. Choose action (Manage Inventory / Search Recipes / Submit Recipe)
Manage Inventory
5.1 Select "Manage Inventory"
5.2 Choose to scan barcode or add manually
alt [Using Barcode Scanner]
5.3 Scan product
5.4 Return scanned data
5.5 Save product in inventory
5.6 Confirm inventory update
5.7 Display updated inventory
Search Recipes
6.1 Select "Search Recipes"
6.2 Query recipes based on inventory
6.3 Return recipe suggestions
6.4 Display recipes
6.5 Save recipe to favorites (optional)
6.6 Update favorites list
Submit Recipe
7.1 Select "Submit Recipe"
7.2 Save recipe as pending
7.3 Notify admin for review
7.4 Approve/Reject recipe
7.5 Update recipe status
7.6 Display submission status
8. Logout
9. End session

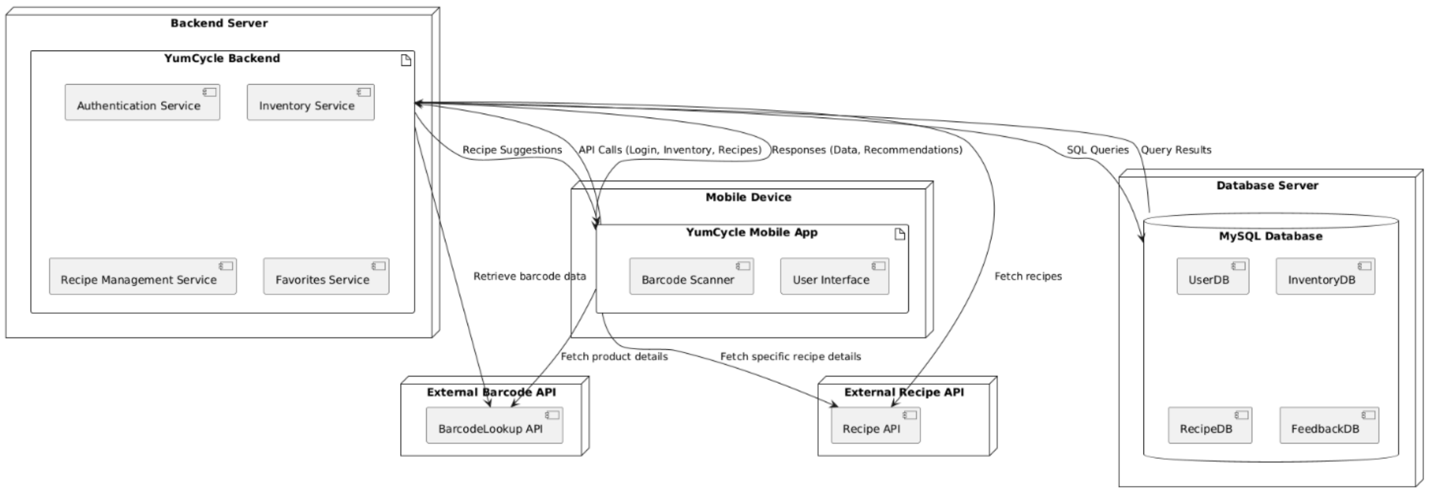
8.10 Package Diagram

This package diagram organizes the different components of the YumCycle application into logical groups, making it easier to understand the overall structure. This is useful, as it helps break down the system into manageable sections, showing dependencies between them, including packages for major parts of the system, such as User Management, Inventory Management, Recipe Management, and Administration.



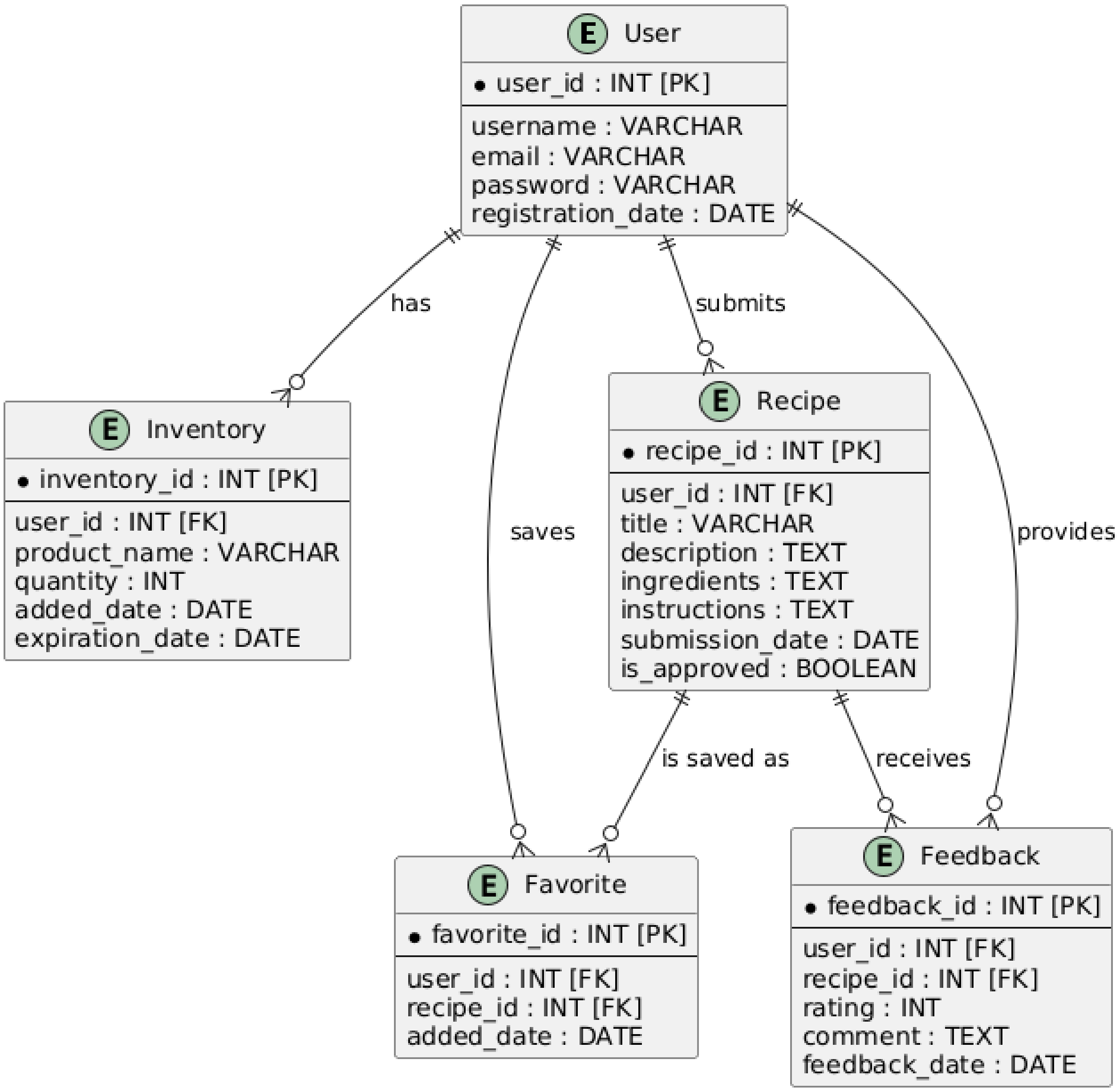
8.11 Deployment diagram

A deployment diagram represents the physical architecture of the system, showing how software components are deployed on hardware nodes and how different parts of the system communicate in the actual environment. For the *YumCycle* app, a deployment diagram would show how the mobile application, backend server, database, and other components are deployed in a real-world setup.



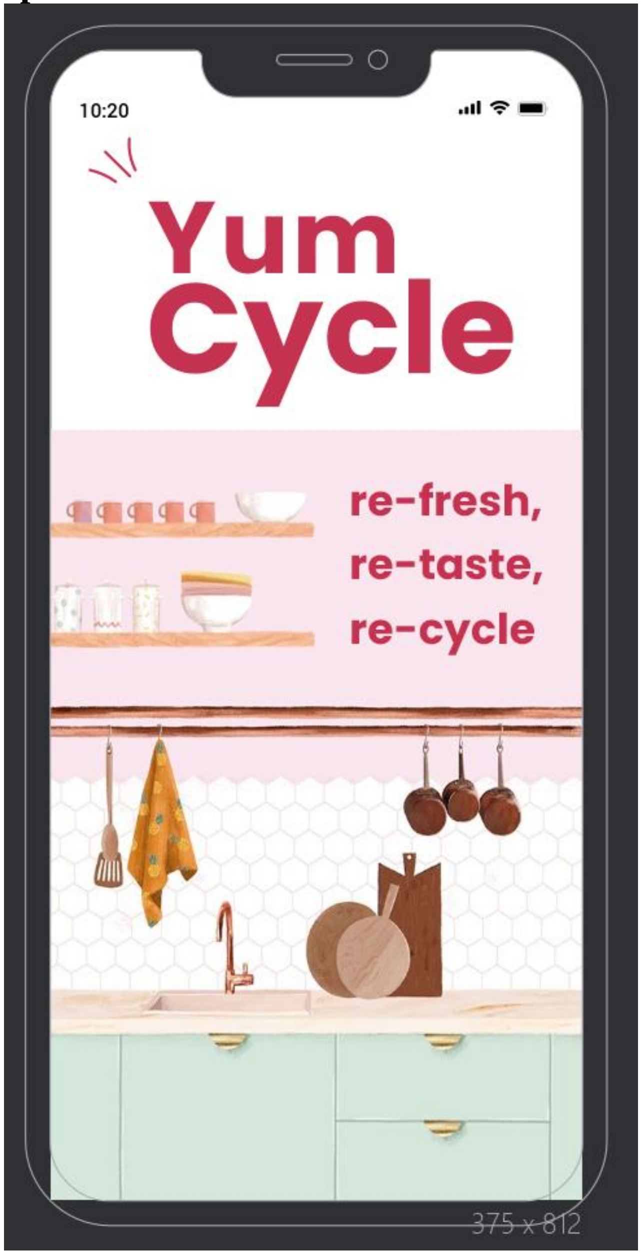
8.12 Data Base Diagram

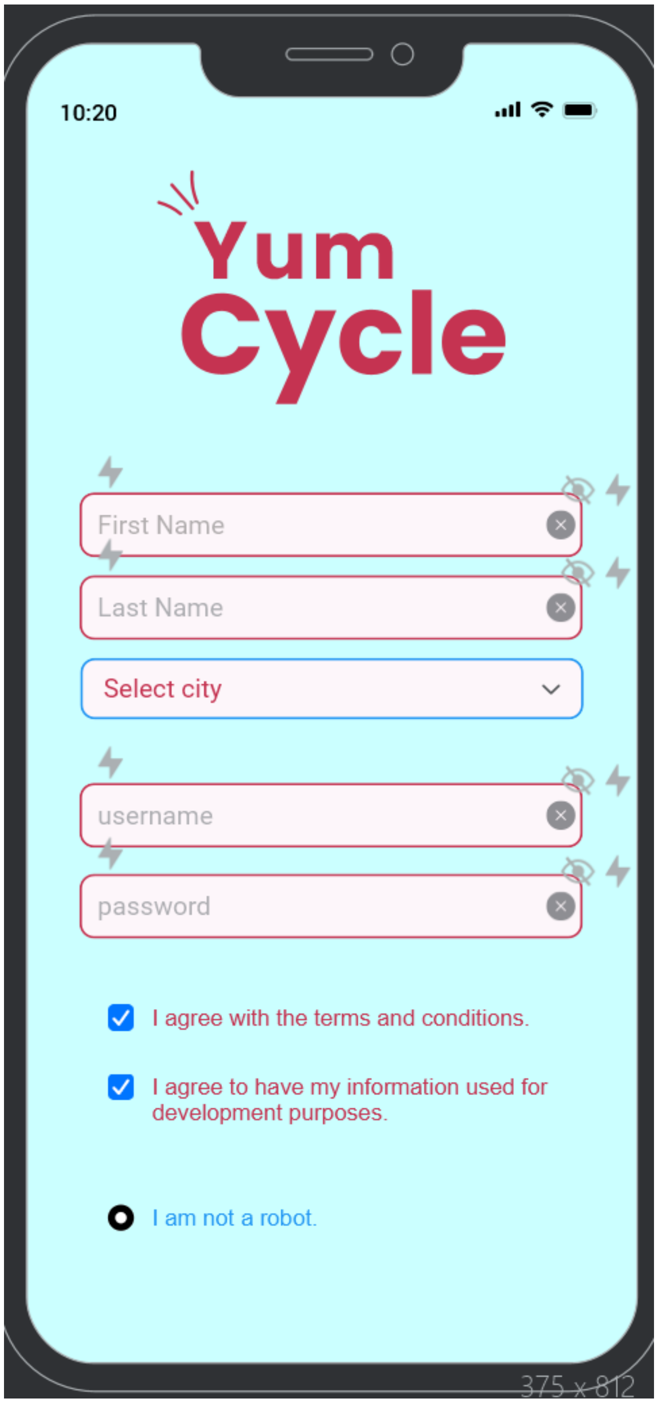
This is the database schema diagram for the YumCycle app, which includes tables for Users, Inventory, Recipes, Favorites, and Feedback. This schema reflects the main entities and their relationships in the app's data model.

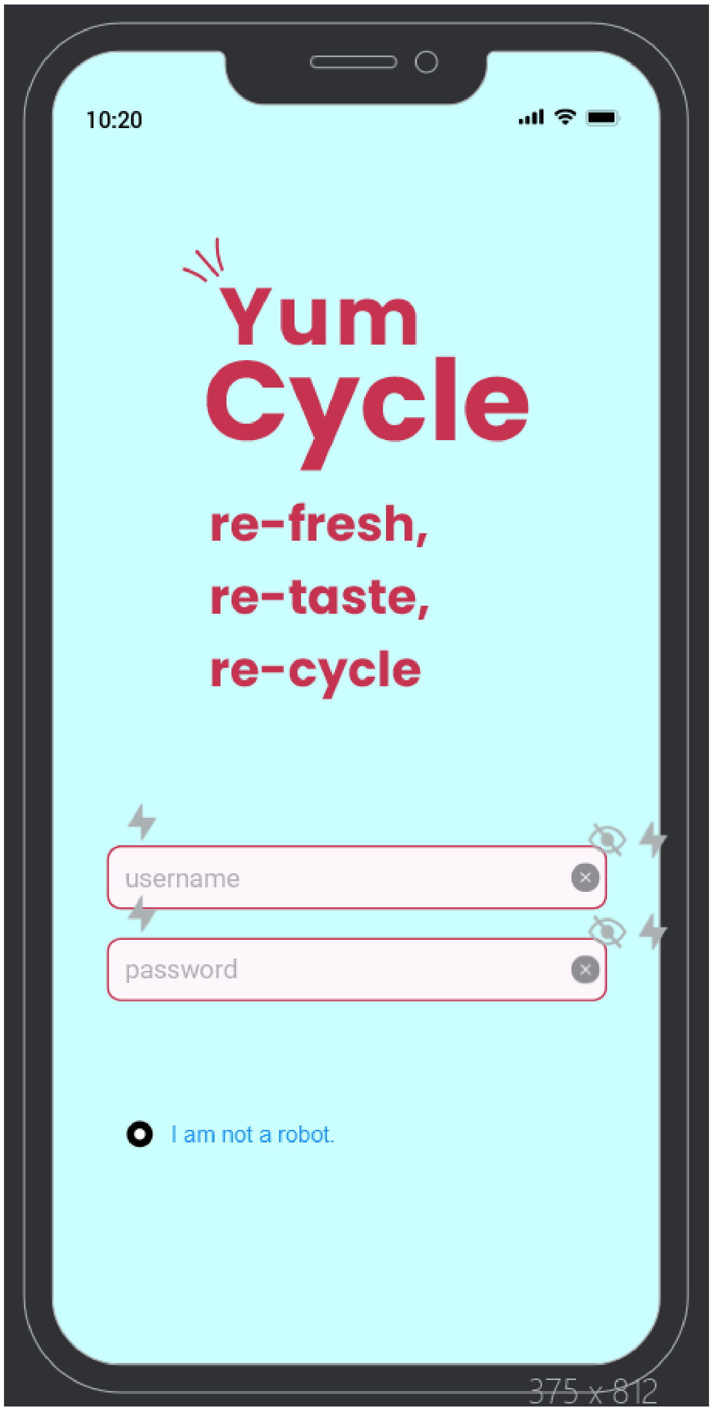


9. Operation Mode (Interactivity presentation) + screen shots (UI design)

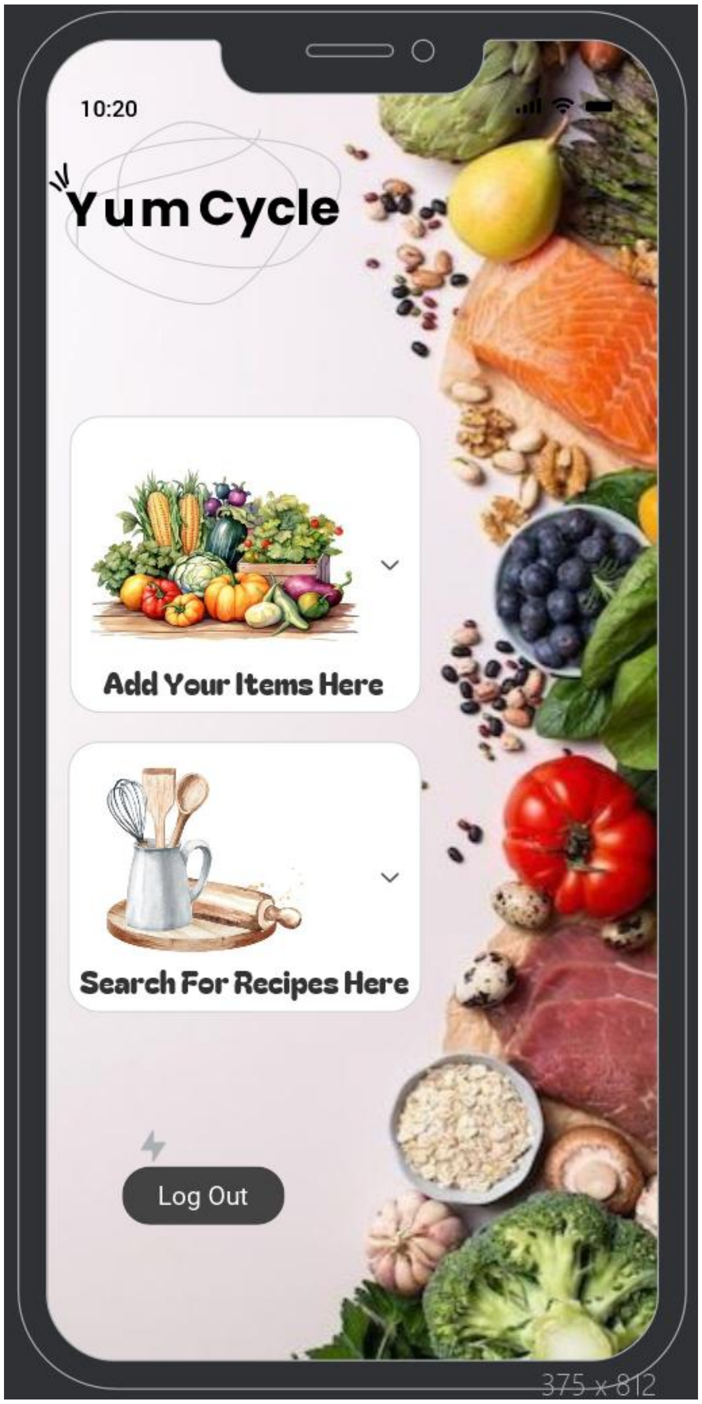
- **Mock-Ups**

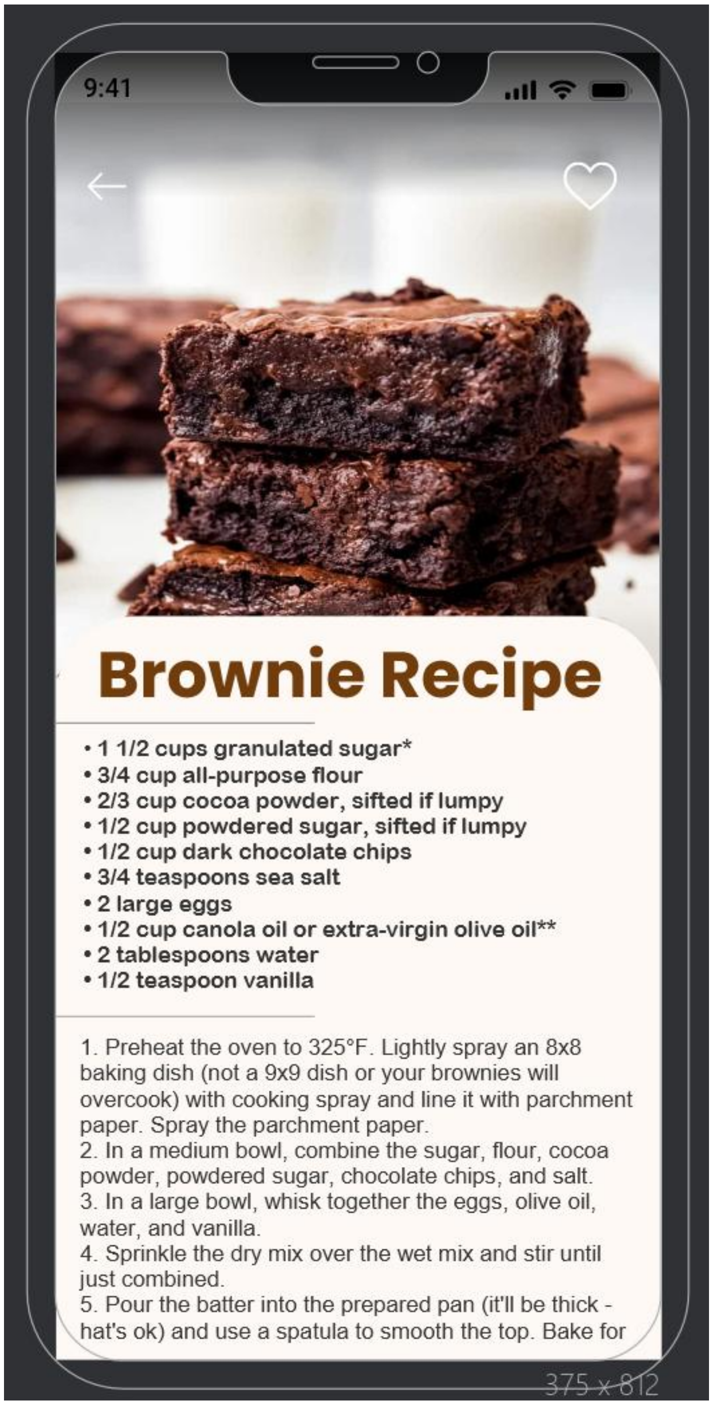



10:20
Yum
Cycle
First Name
Last Name
Select city
username
password
I agree with the terms and conditions.
I agree to have my information used for development purposes.
I am not a robot.
375 x 812



10:20
Yum Cycle
re-fresh,
re-taste,
re-cycle
username
password
I am not a robot.
375 x 812

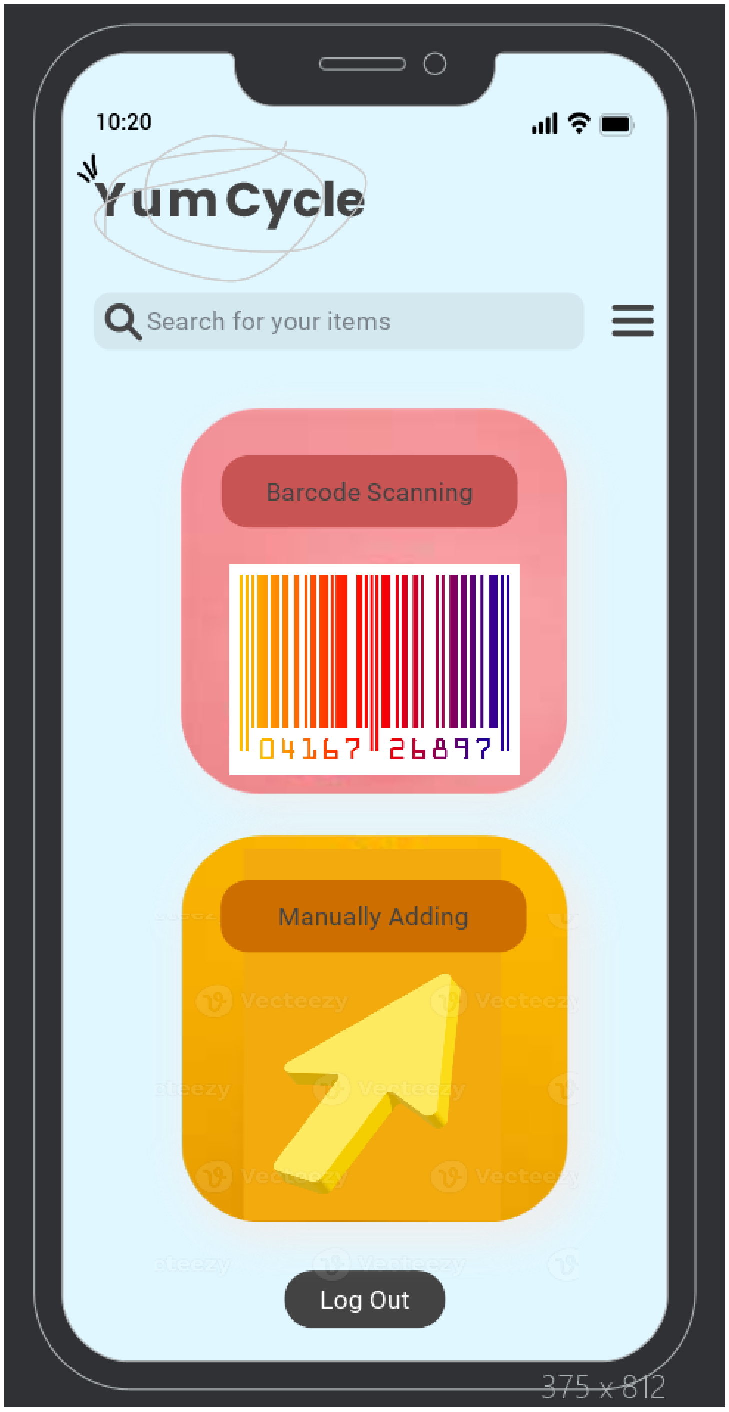

10:20
YumCycle
Add Your Items Here
Search For Recipes Here
Log Out
375 x 812



Brownie Recipe

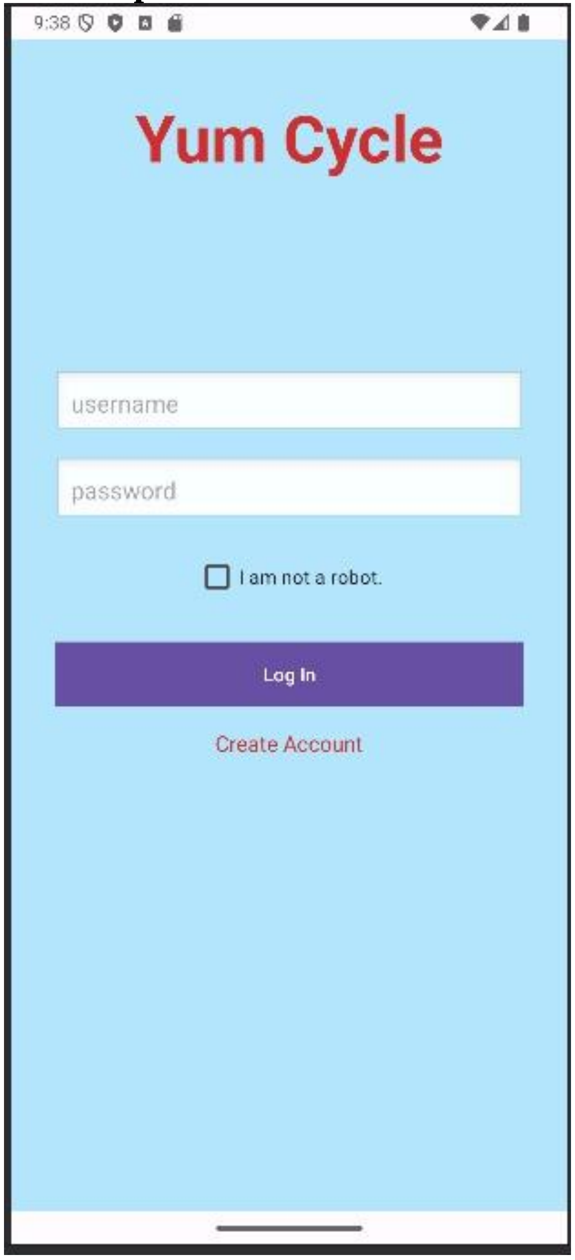
- 1 1/2 cups granulated sugar*
- 3/4 cup all-purpose flour
- 2/3 cup cocoa powder, sifted if lumpy
- 1/2 cup powdered sugar, sifted if lumpy
- 1/2 cup dark chocolate chips
- 3/4 teaspoons sea salt
- 2 large eggs
- 1/2 cup canola oil or extra-virgin olive oil**
- 2 tablespoons water
- 1/2 teaspoon vanilla

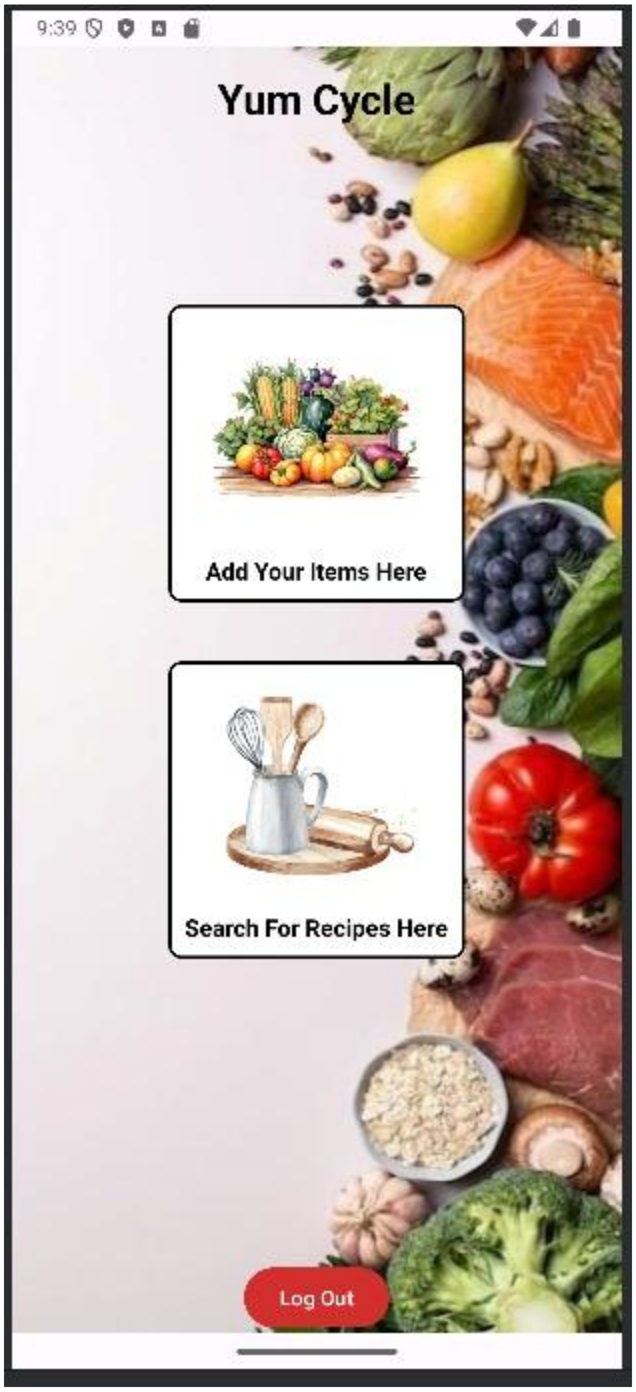
1. Preheat the oven to 325°F. Lightly spray an 8x8 baking dish (not a 9x9 dish or your brownies will overcook) with cooking spray and line it with parchment paper. Spray the parchment paper.
2. In a medium bowl, combine the sugar, flour, cocoa powder, powdered sugar, chocolate chips, and salt.
3. In a large bowl, whisk together the eggs, olive oil, water, and vanilla.
4. Sprinkle the dry mix over the wet mix and stir until just combined.
5. Pour the batter into the prepared pan (it'll be thick - hat's ok) and use a spatula to smooth the top. Bake for

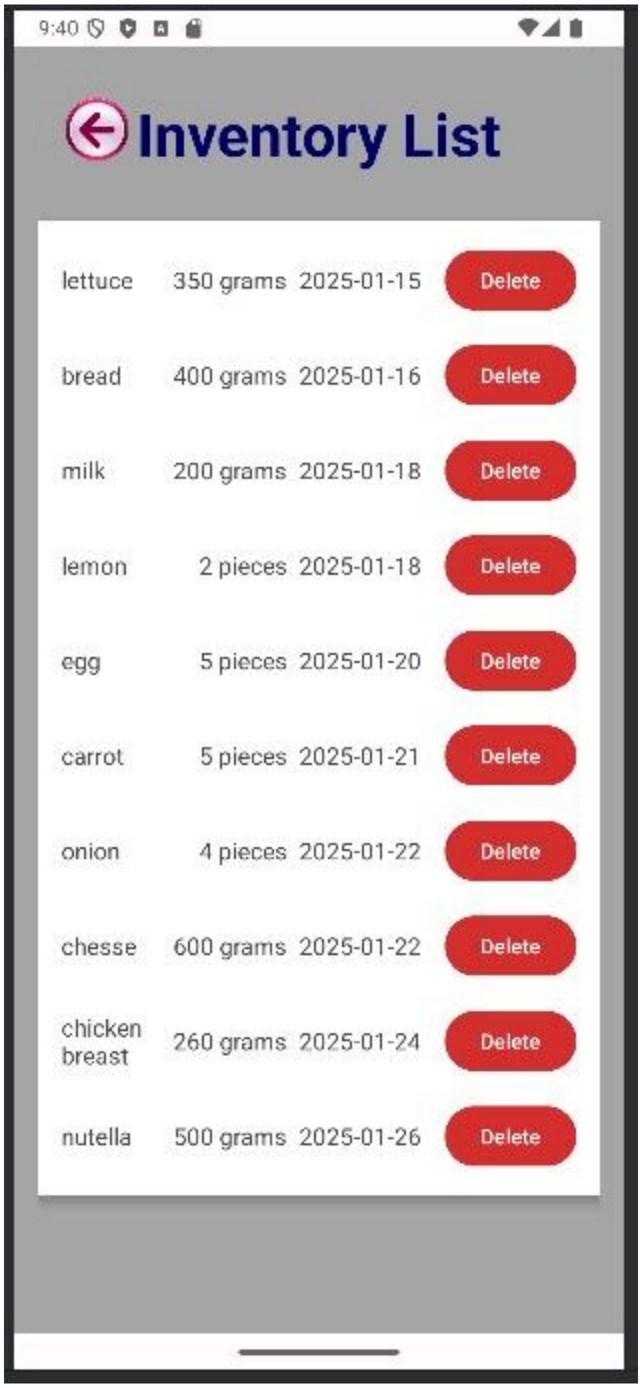


10:20
YumCycle
Search for your items
Barcode Scanning
04167 26897
Manually Adding
Log Out
375 x 812

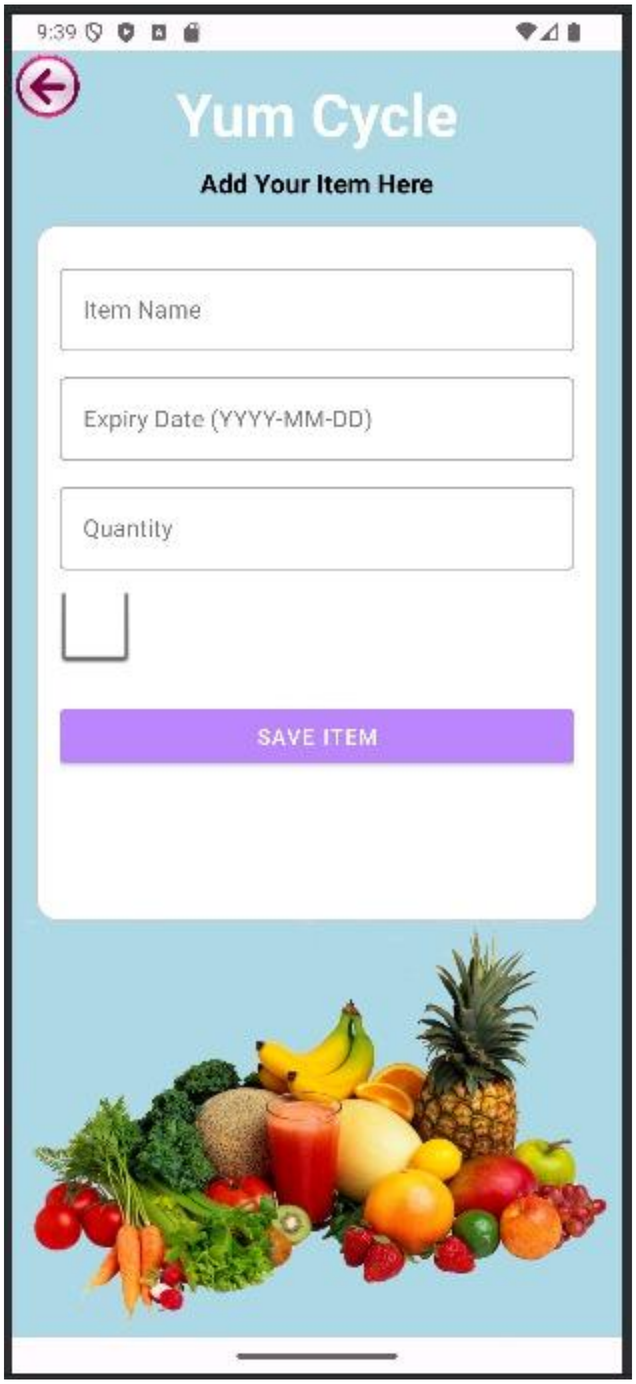
- **The Implemented User Interface :**

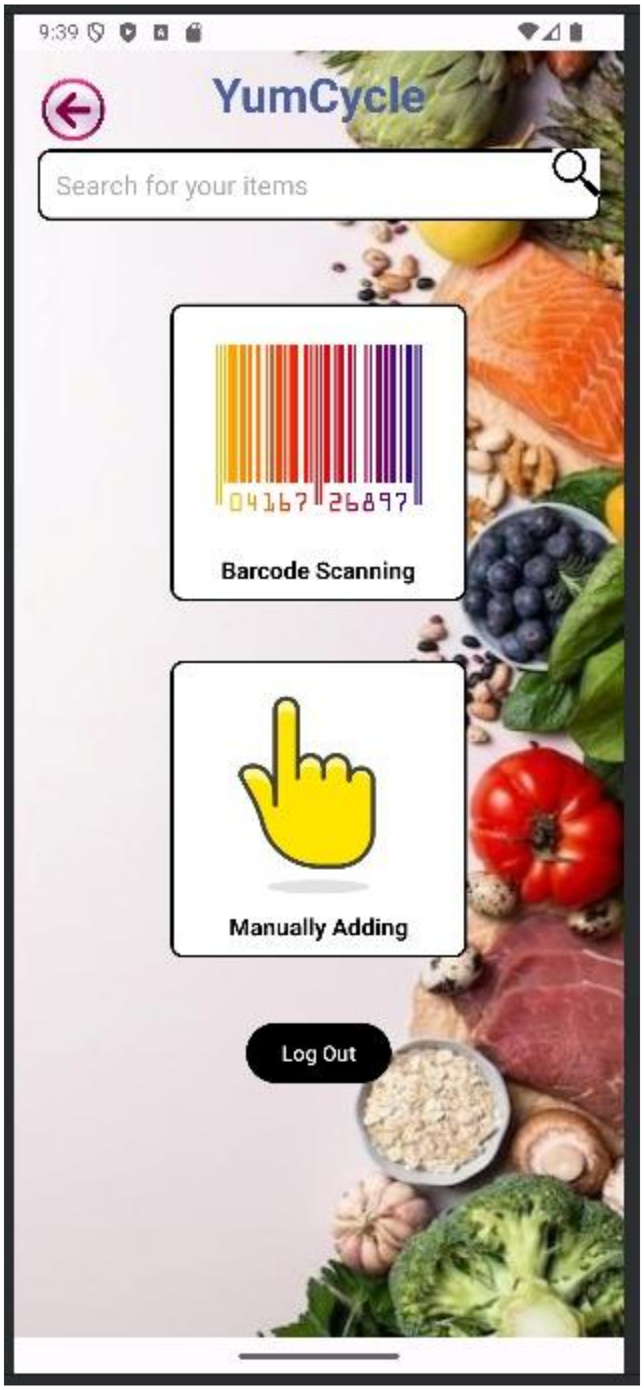



9:39
Yum Cycle
Add Your Items Here
Search For Recipes Here
Log Out

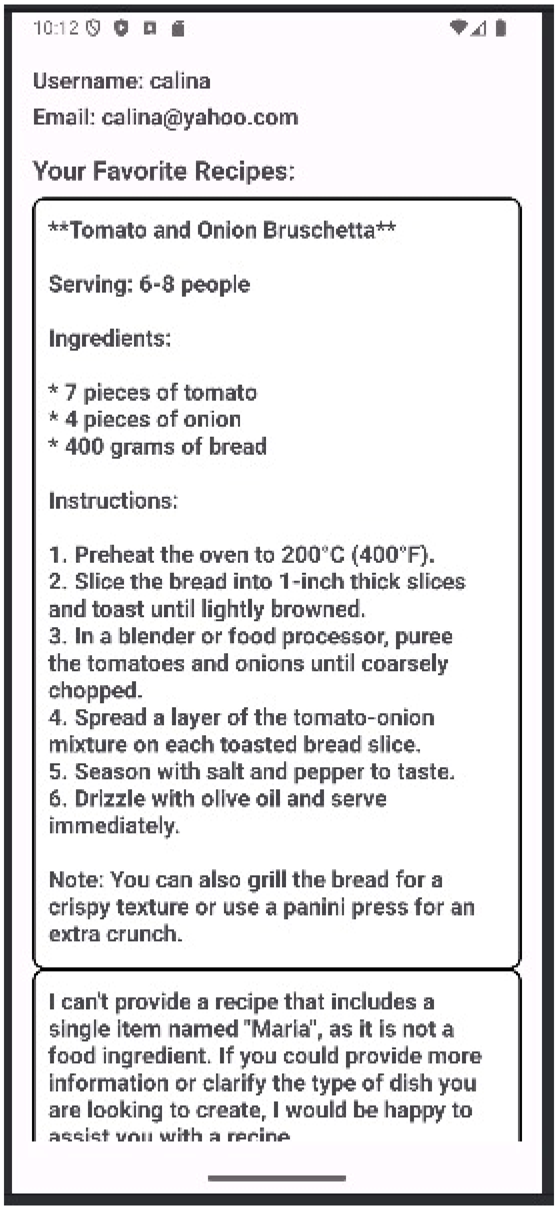


lettuce	350 grams	2025-01-15	Delete
bread	400 grams	2025-01-16	Delete
milk	200 grams	2025-01-18	Delete
lemon	2 pieces	2025-01-18	Delete
egg	5 pieces	2025-01-20	Delete
carrot	5 pieces	2025-01-21	Delete
onion	4 pieces	2025-01-22	Delete
chesse	600 grams	2025-01-22	Delete
chicken breast	260 grams	2025-01-24	Delete
nutella	500 grams	2025-01-26	Delete


9:39
Yum Cycle
Add Your Item Here
Item Name
Expiry Date (YYYY-MM-DD)
Quantity
SAVE ITEM



9:39
YumCycle
Search for your items
04167 26897
Barcode Scanning
Manually Adding
Log Out



10:12

Username: calina

Email: calina@yahoo.com

Your Favorite Recipes:

Tomato and Onion Bruschetta

Serving: 6-8 people

Ingredients:

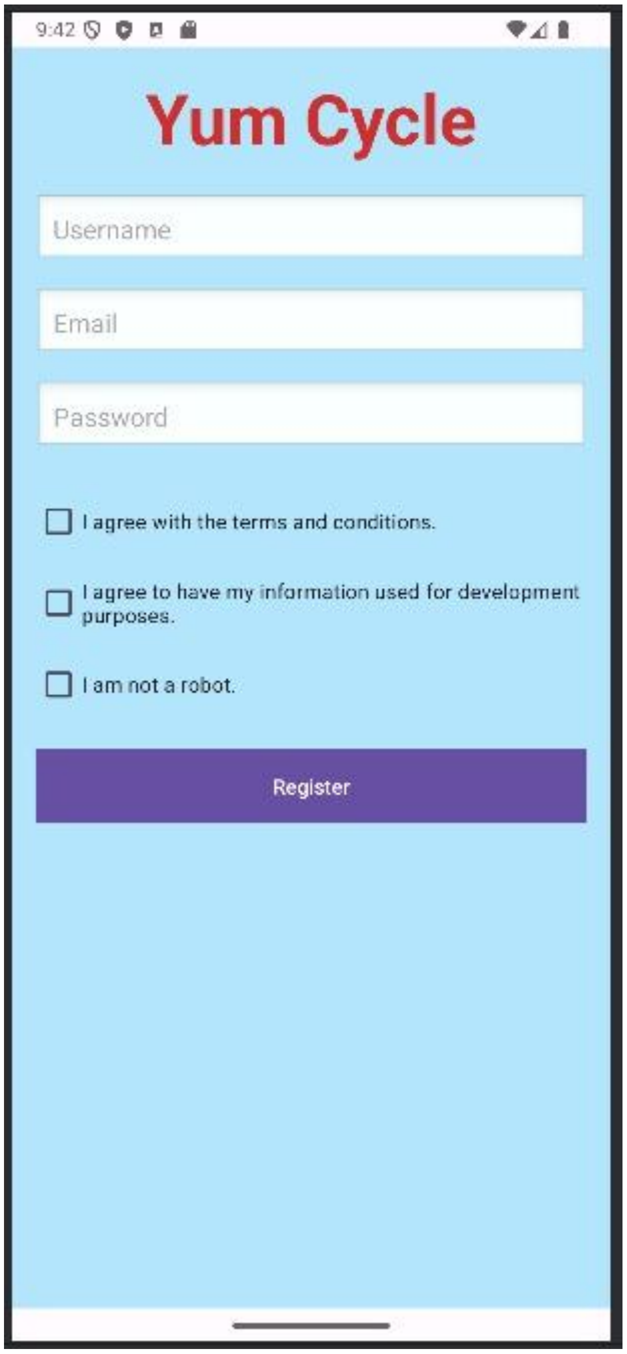
* 7 pieces of tomato
* 4 pieces of onion
* 400 grams of bread

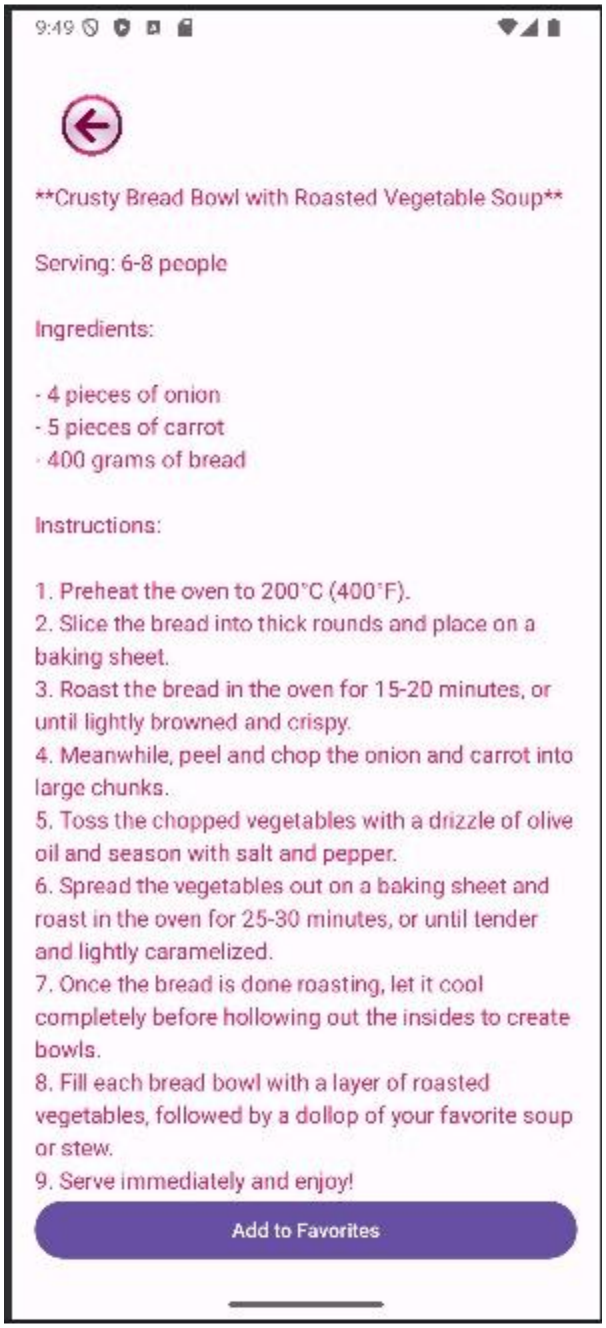
Instructions:

1. Preheat the oven to 200°C (400°F).
2. Slice the bread into 1-inch thick slices and toast until lightly browned.
3. In a blender or food processor, puree the tomatoes and onions until coarsely chopped.
4. Spread a layer of the tomato-onion mixture on each toasted bread slice.
5. Season with salt and pepper to taste.
6. Drizzle with olive oil and serve immediately.

Note: You can also grill the bread for a crispy texture or use a panini press for an extra crunch.

I can't provide a recipe that includes a single item named "Maria", as it is not a food ingredient. If you could provide more information or clarify the type of dish you are looking to create, I would be happy to assist you with a recipe


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



Crusty Bread Bowl with Roasted Vegetable Soup

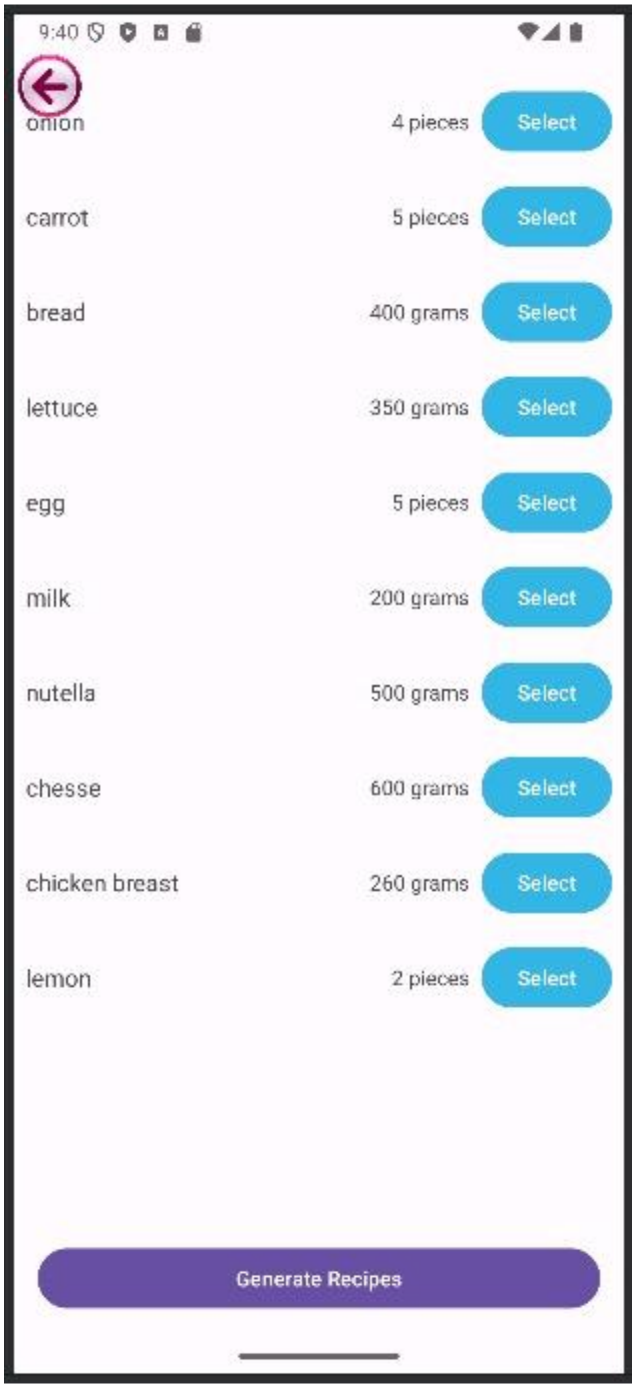
Serving: 6-8 people

Ingredients:

- 4 pieces of onion
- 5 pieces of carrot
- 400 grams of bread

Instructions:

1. Preheat the oven to 200°C (400°F).
2. Slice the bread into thick rounds and place on a baking sheet.
3. Roast the bread in the oven for 15-20 minutes, or until lightly browned and crispy.
4. Meanwhile, peel and chop the onion and carrot into large chunks.
5. Toss the chopped vegetables with a drizzle of olive oil and season with salt and pepper.
6. Spread the vegetables out on a baking sheet and roast in the oven for 25-30 minutes, or until tender and lightly caramelized.
7. Once the bread is done roasting, let it cool completely before hollowing out the insides to create bowls.
8. Fill each bread bowl with a layer of roasted vegetables, followed by a dollop of your favorite soup or stew.
9. Serve immediately and enjoy!



onion	4 pieces	Select
carrot	5 pieces	Select
bread	400 grams	Select
lettuce	350 grams	Select
egg	5 pieces	Select
milk	200 grams	Select
nutella	500 grams	Select
chesse	600 grams	Select
chicken breast	260 grams	Select
lemon	2 pieces	Select

10.Portability

The YumCycle application is designed to be highly portable, allowing for broad compatibility and easy deployment across different environments.

1. Platform Compatibility:
   - Android Operating System: The YumCycle app is developed for Android devices, using Kotlin or Java as the programming language. This ensures compatibility with a wide range of Android devices, from smartphones to tablets, provided they meet the minimum Android version requirements.
   - Backend Server Compatibility: The backend, built with Spring Boot, is platform-agnostic and can run on any server environment that supports Java, making it compatible with various operating systems, such as Windows, Linux, and macOS.
   - Database Portability: MySQL, the application's database, is widely compatible and can be hosted on different database servers or cloud services (e.g., Amazon RDS, Google Cloud SQL), making the app database easily portable across different environments.
2. Cloud Deployment:
   - Cloud-Ready Architecture: The backend and database can be hosted on cloud platforms like AWS, Google Cloud, or Microsoft Azure, allowing for scalability and redundancy. Cloud deployment enables YumCycle to serve a larger user base and maintain performance under varying loads.
   - Containerization with Docker: The backend and database can be containerized using Docker, enhancing portability by enabling easy deployment across any Docker-supported platform, including cloud services and local environments.
3. Device Portability:
   - Data Synchronization Across Devices: YumCycle's data synchronization allows users to log in from multiple Android devices and access the same inventory and recipe data. User data is stored centrally, enabling portability across different devices with seamless access to stored information.
   - API-First Approach: The app's RESTful API design allows easy integration with other platforms or frontend applications, should YumCycle expand to other operating systems (e.g., iOS or web apps) in the future.
4. Future Expansion to Other Platforms:
   - Cross-Platform Framework: If the app is to be expanded to other mobile operating systems (such as iOS), frameworks like Flutter or React Native could be used to rebuild the application for both Android and iOS.

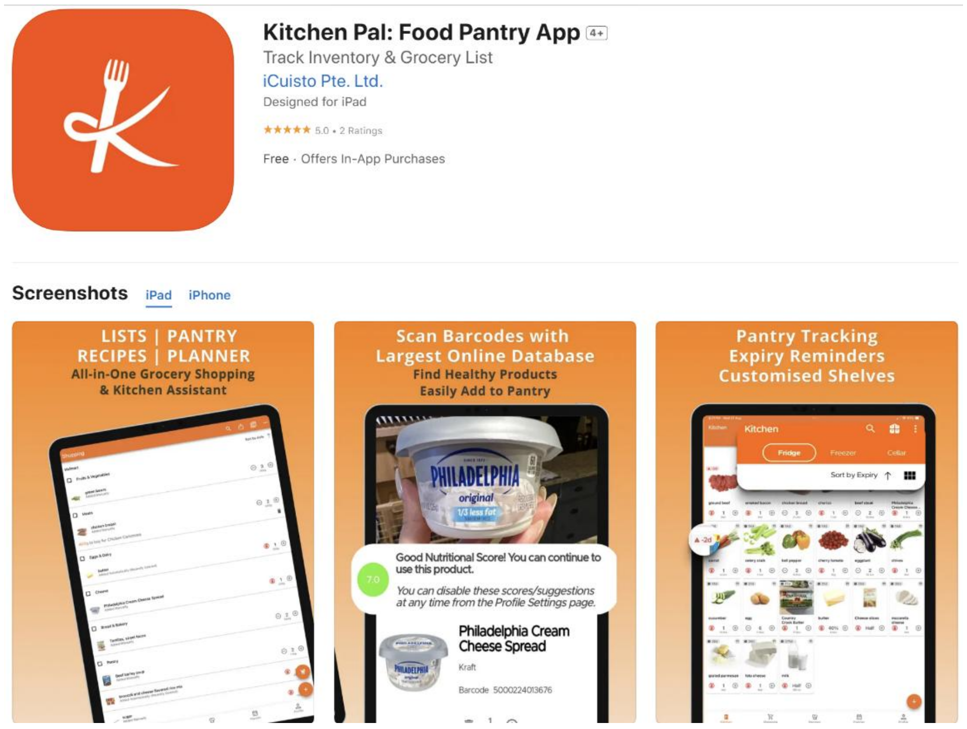
- Web Version: Due to its API-driven backend, YumCycle could easily offer a web-based application, extending accessibility and making it available across different device types and operating systems.

The YumCycle app's portability and adaptable design ensure it can be deployed, expanded, and maintained across various platforms and environments as user needs and technology evolve.

11. Competing software

- **Kitchen Pal: Food Pantry App**

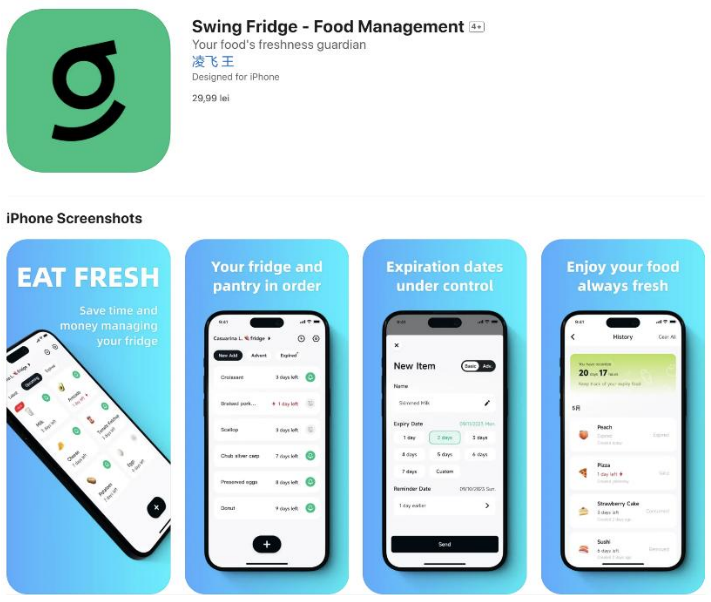
"KitchenPal (iCuisto) is the only app that works as a pantry manager, grocery list, product comparison, meal planner, family organizer, and recipe ideas app - all rolled into one. The app learns and provides suggestions the more you use it, like a smart assistant."


Kitchen Pal: Food Pantry App 4+
Track Inventory & Grocery List
iCuisto Pte. Ltd.
Designed for iPad
★★★★★ 5.0 • 2 Ratings
Free · Offers In-App Purchases
Screenshots
iPad
iPhone
LISTS | PANTRY
RECIPES | PLANNER
All-in-One Grocery Shopping & Kitchen Assistant
Scan Barcodes with Largest Online Database
Find Healthy Products
Easily Add to Pantry
Good Nutritional Score! You can continue to use this product.
You can disable these scores/suggestions at any time from the Profile Settings page.
Philadelphia Cream Cheese Spread
Kraft
Barcode 5000224013676
Pantry Tracking
Expiry Reminders
Customised Shelves
Kitchen
Fridge
Freezer
Cellar
Sort by Expiry

While KitchenPal offers a broad range of features, YumCycle stands out by providing a more streamlined, user-friendly experience focused on essential functionalities without requiring a subscription. YumCycle's barcode scanning feature connects directly with the inventory to suggest recipes based on ingredients users already have, making it easier to minimize food waste and grocery expenses. Unlike KitchenPal, YumCycle allows users to submit personal recipes for community sharing, fostering engagement and enriching the recipe database with unique, user-approved options. Furthermore, YumCycle is designed to be less overwhelming, prioritizing core features like pantry tracking and recipe generation, for a straightforward, accessible experience. Future plans for dietary filters, family syncing, and smart shopping lists will make YumCycle a highly personalized yet simple solution for managing household food effectively, offering an appealing alternative to KitchenPal's extensive, premium-locked features.

- **Swing Fridge**

"Swing Fridge is a straightforward, user-friendly tool for tracking your refrigerator's contents. We provide various efficient methods for recording, catering to diverse food requirements. Enhanced with charming emojis and entertaining notifications, you'll discover an entirely novel way to engage with your food. Let's commence with minor changes, curbing food wastage, and elevating our sense of accomplishment."

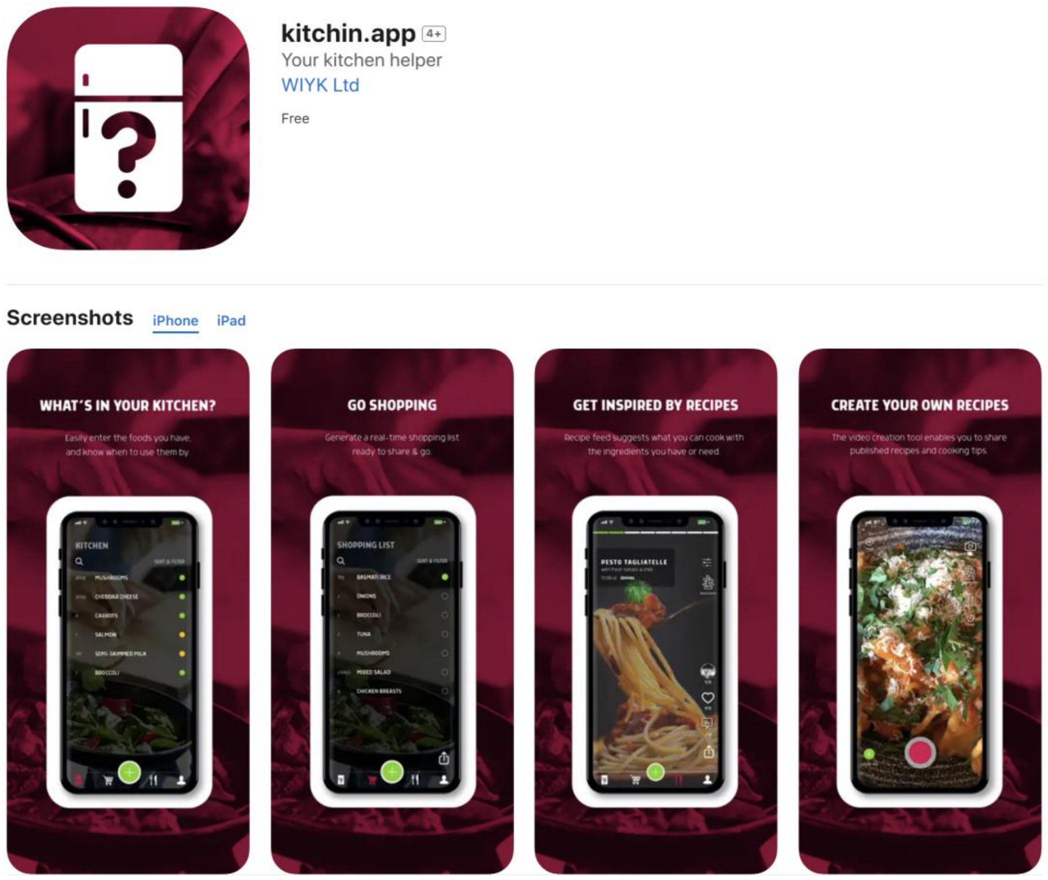

Swing Fridge - Food Management 4+
Your food's freshness guardian
凌飞王
Designed for iPhone
29,99 lei
iPhone Screenshots
EAT FRESH
Save time and money managing your fridge
Your fridge and pantry in order
Expiration dates under control
Enjoy your food always fresh

Recommended: 20.11.2017	**Approved:**
Company/University Name: Address: Universitatea Tenhica Cluj-Napoca Romania	

YumCycle outshines Swing Fridge by offering more comprehensive functionality without sacrificing ease of use. While Swing Fridge emphasizes simplicity and a fun approach with emojis and notifications, YumCycle goes beyond basic fridge tracking by integrating recipe generation based on existing ingredients and barcode scanning to help users make the most of their pantry while reducing waste.

- **kitchin.app**

Kitchin helps you to optimize the running of the most important part of the home, the kitchen… it suggests what you can cook using the ingredients you have, compiles a shopping list for ingredients you need, creates and publishes your own favorite recipes, and reduces waste by helping you to make the most out of food.

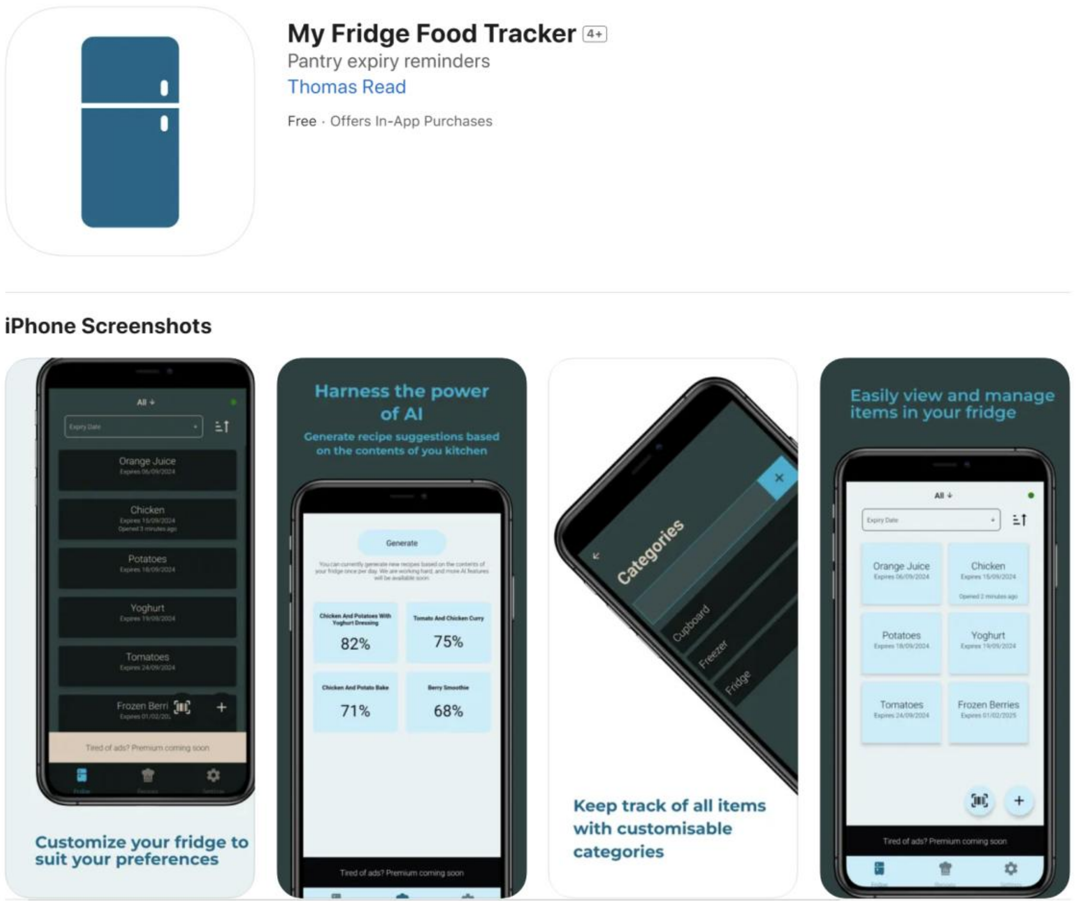

kitchin.app 4+
Your kitchen helper
WIYK Ltd
Free
Screenshots
iPhone
iPad
WHAT'S IN YOUR KITCHEN?
Easily enter the foods you have, and know when to use them by.
GO SHOPPING
Generate a real-time shopping list ready to share & go.
GET INSPIRED BY RECIPES
Recipe feed suggests what you can cook with the ingredients you have or need.
CREATE YOUR OWN RECIPES
The video creation tool enables you to share published recipes and cooking tips.

Recommended: 20.11.2017	**Approved:**
Company/University Name: Address: Universitatea Tenhica Cluj-Napoca Romania	

YumCycle surpasses Kitchin in offering a more accessible and community-oriented approach to kitchen management. While Kitchin covers essential functions like ingredient-based recipe suggestions, shopping list creation, and waste reduction, YumCycle provides a more robust experience by allowing users to directly scan barcodes to add items to their inventory, which simplifies and speeds up pantry management.

- **My Fridge Food Tracker**

Harness the power of AI to generate recipe suggestions based on the contents of your kitchen - and never worry about what to eat again! Introducing MyFridge - the modern fridge management app that makes managing your fridge inventory a breeze! You can now generate recipe suggestions based on the contents of your kitchen using AI for free!


My Fridge Food Tracker 4+
Pantry expiry reminders
Thomas Read
Free · Offers In-App Purchases
iPhone Screenshots
Customize your fridge to suit your preferences
Harness the power of AI
Generate recipe suggestions based on the contents of you kitchen
Keep track of all items with customisable categories
Easily view and manage items in your fridge

YumCycle offers a richer, more community-driven experience compared to MyFridge by integrating both pantry management and meal inspiration in a seamless, user-friendly app. While MyFridge focuses on tracking fridge inventory, and AI-generated recipe suggestions, YumCycle goes further by providing a community space where users can share and access unique recipes, reviewed by administrators for quality. This feature allows YumCycle to deliver personalized meal ideas that go beyond simple ingredient matching.

12. Testing and Validation

a. Testing Strategy

- **API Testing with Postman**

Objective: Manually test backend API endpoints to ensure they respond correctly to various requests.
Approach:

- Endpoint Verification: Created a collection of API requests in Postman to test all backend endpoints, including:
  User Endpoints: Creating users, fetching user details.
  Recipe Endpoints: Adding recipes, retrieving recipes.
  Favorites Endpoints: Adding favorites, fetching user favorites, deleting favorites.
- Test Scenarios:
  Valid Requests: Sending well-formed requests with correct data to verify successful operations.
  Invalid Requests: Sending malformed requests, missing required fields, or incorrect data types to ensure the backend handles errors gracefully.
  Edge Cases: Testing boundary conditions, such as adding a favorite with non-existent userId or recipeId.

Tools: Postman, for crafting and executing API requests, as well as automating tests with scripts.

- **Manual Testing Through the Android App**

Objective: Interact with the application as an end-user to identify functional and usability issues.
Approach:

- Feature Testing

Add to Favorites: Used the app to add recipes to favorites, ensuring that the action reflects correctly in both the frontend and backend.

Navigation: Tested navigation flows (e.g., moving from RecipeDisplayFragment to SearchRecipeFragment) to ensure smooth transitions.

Data Display: Verified that data fetched from the backend (e.g., list of favorite recipes) displays correctly in the UI.

- Error Handling:

Network Failures: Simulated network issues to check how the app responds (e.g., displaying error messages, retry options).

Invalid Inputs: Entered invalid or incomplete data to ensure the app handles input validation appropriately.

- Performance Testing:

Load Testing: Observed the app's performance when handling a large number of favorites or recipes to ensure responsiveness.

Tools:

Android Emulator/Real Devices: For running and interacting with the app.

Logcat: To monitor logs from the Android app for debugging purposes.

- **Database Verification**

Objective: Ensure that data is correctly stored, relationships are maintained, and constraints are enforced in the database.

Approach:

- Direct Inspection:

SQL Queries: Ran SQL queries to inspect the contents of tables (users, recipes, favorites) to verify that records are inserted, updated, and deleted as expected.

Foreign Key Constraints: Checked that foreign key relationships are correctly established and that cascading deletes function properly.

- Data Consistency:

Referential Integrity: Ensured that user_id and recipe_id in the favorites table correctly reference existing records in users and recipes tables.

Data Types: Confirmed that column data types (BIGINT for IDs) align with backend entity definitions to prevent mismatches.

Tools: Database Client (e.g., DBeaver, pgAdmin): For executing SQL queries and visually inspecting database schemas and data.

a. Bug 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

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.

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.

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.

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.