SOFTWARE REQUIREMENTS SPECIFICATION

RECIPE ROULETTE

Software Engineering

Bachelor's Degree in Bioinformatics

Course 2023-24

(Version 2.2)
(04-06-24)

Group member:

Alessandra Bonilla Salon
Maria Lopez Moriana
Carmen Samedi

Supervisor:

Daniel Soto Alvarez

ESCI upf.
School of International Studies

I. REVISION HISTORY

Name	Date	Sections Changed	Version
María/Alessandra/Carmen	29/05/2024	4.1. Architectural overview 4.2. Architecture diagram 4.3. Use case diagram 4.3. Use case diagram 4.5. Activity diagrams 4.7. Class diagram	2.0
María/Alessandra/Carmen	30/05/2024	4.7. Class diagram 4.2. Architecture diagram Overall organization of the document	2.1
María/Alessandra/Carmen	04/06/2024	4.3. Class diagram 3.1. Product perspective 3.2. System features 3.3. Operating environment 3.4. Design and implementation constraints	2.2

II. TABLE OF CONTENTS

1. PROJECT DOMAIN

1.1. PURPOSE

The purpose of the "Recipe Roulette" app is to provide a user-friendly platform that provides a personalized cooking assistant that helps users utilize their available ingredients to discover and prepare new recipes. By doing so, the app aims to reduce food waste and enhance the cooking experience for individuals of varying culinary skill levels, from beginner cook to culinary expert.

1.2. TARGET MARKET

This Software Requirements Specification (SRS) document outlines the functional and nonfunctional requirements for " Recipe Roulette ", it provides a detailed overview of the application's features, intended use, and constraints. It serves as a guideline for the development team to design and implement the software and as a contract between the stakeholder and the development team, ensuring that the final product meets the users' and stakeholders' expectations.

The intended audience would include:

Project Managers: Who organize and oversee the project progression.

Developers: Use this document as reference for feature implementation of the app.

Quality control team: Who ensure the app meets the outlined specifications.

1.3. PRODUCT SCOPE

Nearly a third of all the food (around 1.3 billion tonnes of food) produced each year is squandered or lost before it can be consumed. All this food produced but never eaten would be sufficient to feed two billion people. That's more than twice the number of undernourished people across the globe!

"Recipe Roulette" is a mobile and web application that aims to minimize food wastage and promote cooking at home by recommending recipes based on ingredients that users already have. It provides a holistic approach to cooking by facilitating recipe sharing, community building, and culinary education. It allows users to enter ingredients, apply filters for dietary preferences, and receive recipes that they can prepare. Additionally, users can contribute recipes, earn rewards for engagement, and partake in a community of cooking enthusiasts. It includes a comprehensive notification system for engaging users with new recipes and app updates.

1.4. REFERENCES

- https://www.wfp.org/stories/5-facts-about-food-waste-and-hunger
- Links to video recipes informing tips or visualizing the way to cook.
- IEEE Recommended practice for SRS
- User Interface Design Guidelines (SRS Template)

1.5. LINKS

Github:

- https://github.com/carmensat/RECIPE-ROULETTE/tree/main
- git clone git@github.com:carmensat/RECIPE-ROULETTE.git

Trello:

- https://trello.com/b/12HNlFAm/recipe-roulette

2. SPECIFIC REQUIREMENTS

In the following section, there are the detailed description and corresponding diagrams of the use cases and the requirements.

2.1. USE CASE LIST

The use case model serves as a key tool in understanding and documenting how users interact with the "Recipe Roulette" system. This model helps to identify and define the roles of various actors and the fundamental processes they perform within the application. It provides a framework for analyzing the functional requirements by showcasing the sequences of actions between the user and the system to achieve specific goals. Through use cases, we capture user interactions that are essential for the system to fulfill its functionalities, ensuring that all user expectations are met systematically.

UC1: Register Account
UC2: Search Recipes
UC3: Upload Recipe
UC4: Gain Points with Recipe Interactions
UC5: Add Recipe to Local Database
UC6: Delete Recipe from Local Database
UC7: Search In Local Database
UC8: Recipe Approval
UC9: Filter Recipe
UC10: User Verification Management
UC11: Software management
UC12: In-App messages
UC13: Check Reward Status
UC14: User LogIn
UC15: Delete profile
UC16: Delete Recipes

2.2. USE CASE DESCRIPTIONS

Each use case identified in the diagram above is described in detail below, outlining the system's response to user actions:

UC1: Register Account

Actor: Guest User
Description: Allows a guest user to register and create a personal account within the application.
Preconditions: The user must have valid personal information and access to the internet.
Postconditions: The user account is created and the user can log in to access personalized features.

Basic Flow:
1. The user selects the "Register" option.
2. The user fills in the required information (e.g., name, email, password).
3. The system validates the information and creates a new user account.
4. The system confirms account creation and prompts the user to log in.

Alternative Flows:
- If the user enters an email that is already in use, the system prompts to try a different email.
- If the user provides incomplete information, the system displays an error and requests complete details.

UC2: Search Recipes

Actor: User (Registered)
Description: Allows users to search for recipes based on various criteria (ingredients, cuisine, dietary restrictions).
Preconditions: None.
Postconditions: Recipes matching the criteria are displayed.

Basic Flow:
1. The user signs into their account.
2. The user goes to the search page.
3. The system displays the search page.
4. The user selects the search bar.
5. The user chooses to search by either dish name or ingredient.
6. The user types the dish name or ingredient and clicks the corresponding button.
7. The system checks if the ingredient exists in the database.
    - If yes, the system will search for exact matches in the ingredients of its database.
    - If no, the system will keep the entire word to find matches.
8. The user clicks the 'submit' button.
9. The system searches the recipe database and displays search results.
10. The user can choose to apply additional filters (e.g., time required, food type, dietary restrictions, difficulty level, allergies).
11. The system displays the filter menu.
12. The user selects and applies desired filters.
13. The system displays filtered results.

Alternative Flows:
- If no recipes match the criteria, the system displays a message indicating no results found and suggests altering search parameters.

UC3: Upload Recipe

Actor: Registered User
Description: Allows registered users to upload their own recipes into the database.
Preconditions: User must be logged in.
Postconditions: The recipe is added to the database and available for others to view.

Basic Flow:

1. The user navigates to the "Upload Recipe" section.
2. The user enters the details of the recipe (name, ingredients, steps) and submits.
3. The system validates the entered data and adds the recipe to the database.
4. The system confirms the successful addition of the recipe.

Alternative Flows:

- If mandatory fields are missing, the system prompts the user to complete all required fields before submission.

UC4: Gain Points with Recipe Interactions

Actor: Registered User
Description: Users earn points by engaging in app activities like uploading recipes, commenting, and interacting with posts (like and save recipes). These points increase the user's profile status, making them more visible to others. After gaining points, users can check their reward status to redeem earned points for rewards within the app.
Preconditions: User must be logged in, App is operational and tracks user interactions for point attribution.
Postconditions: User's point balance is updated, User's profile status is adjusted, User may check their reward status and redeem points for rewards.

Basic Flow:

1. User engages in app activities such as:
    - ❖ Uploading recipes
    - ❖ Commenting on recipes or posts
    - ❖ Liking other recipes
    - ❖ Saving recipes
2. The app tracks the user's interactions and attributes points accordingly.
3. The user's point balance is updated in real-time within their profile.
4. The user's profile status is adjusted based on the accumulated points, increasing visibility to other users.

Alternative Flows:

- **Activity Limit Reached:** User reaches limit for certain activities within a specified time frame: daily upload limit for the recipes.
- **Technical Issues:** The tracking system can be interrupted, so points may not be updated real-time, once solved the issues then the user’s points should be updated accordingly.
- **Redemption of Points:** The gained points can be exchanged by some rewards; e.g. possibility of more daily uploads.

UC5: Add Recipe to Local Database

Actor: Registered User
Description: This use case describes how a registered user can manage their local database of saved recipes. The user can save recipes to their local database, which allows for offline access.
Preconditions:

- The user must be logged into their account.
- The user must have internet access to initially download and save recipes to their local database.
- There must be sufficient storage space available in the user's device.

Postconditions:

- The local database is updated with the user's changes, whether it's adding or deleting recipes.
- The user can access their saved recipes offline**.**
- The local database is synced up with the original database in order to keep it updated.

Basic Flow:.

1. **The user made a search for a desired recipe:** Flow of the UC2
2. **User selects a recipe to save:** The user browses through recipes and selects one to save.
3. **System prompts for confirmation:** The system prompts (‘Cancel’ and ‘Confirm’) the user to confirm saving the recipe to the local database.
4. **User confirms saving the recipe.**
5. **A copy of the recipe is made from the primary database to its local database**
6. **User accesses the local database.**

Alternative Flows:

- **Internet Connection Issue:** If the user tries to save a recipe while offline, the system will notify the user that an internet connection is required. The user can retry once internet access is available.
- **Recipe Already Exists:** If the user tries to save a recipe that already exists in the local database, the system will notify the user and prompt the message ‘Recipe already saved in your profile !’
- **Local Storage Full:** If the user's device does not have enough storage space to save a new recipe, the system will notify the user. The user can choose to free up space or cancel the save operation.

UC6: Delete Recipe from Local Database

Actor: Registered User
Description: This use case describes how a registered user can manage their local database of saved recipes. Users can delete recipes from the local database.
Preconditions:

- The user must be logged into their account.
- There must be a copy of the recipe in the user’s local database.

Postconditions:

- Deleting the copy should not make any changes to the original database
- After deletion, the memory that the recipe occupied in the user’s local database is freed up.

Basic Flow:.

1. **User navigates to local database:** The user accesses the section of the app where locally saved recipes are stored.
2. **User selects a recipe to delete:** The user selects the recipe they wish to delete from their local database.
3. **System prompts for confirmation:** The system prompts the user to confirm the deletion of the recipe.
4. **User confirms deletion:** The user confirms the deletion action.
5. **System deletes the recipe:** The system deletes the recipe from the user's local database.
6. **System confirms success:** The system notifies the user that the recipe has been successfully deleted from the local database.

Alternative Flows:

- **Recipe Not Found:** If the system cannot find the recipe in the local database (e.g., it was already deleted or there was an error), the system notifies the user of the issue. The user is returned to the local database view without changes.

UC7: Search In Local Database

Actor: Registered User
Description: This use case describes how a registered user can search for recipes stored in their local database without needing an internet connection.
Preconditions:
- The user must be logged into their account.
- The user must have recipes saved in their local database.

Postconditions:
- The system displays the search results from the local database based on the user's query.
- The user can access the details of the recipes found in the local search.

Basic Flow:.
1. **User navigates through its profile page**: The user accesses the section of the app where they can perform a search in their local database.
2. **User selects 'Saved recipes**: The user selects this option in order to visualize all its local saved recipes.
3. **Users select the magnifying glass icon to make the search:** The user chooses the option to search within their local database.
4. **User enters search criteria**: The user types the name of the recipe or ingredients into the search bar.
5. **System performs local search**: The system searches the local database for recipes that match the entered criteria.
6. **System displays search results**: The system displays a list of recipes that match the search criteria from the local database.
7. **User selects a recipe from the search results**: The user clicks on a recipe from the search results to view its details.
8. **System displays recipe details**: The system displays the details of the selected recipe.

Alternative Flows:
- **No Recipes Found:** If no recipes match the search criteria, the system notifies the user that no matches were found. The user is prompted to refine their search criteria or try a different search.
- **User Cancels Search;** At any point, if the user decides to cancel the search operation, they can return to the previous page. No search results are displayed, and the user is returned to the local database view.

UC8: Recipe Approval

Actor: Content Administrator
Description: This use case describes how a Content Administrator reviews and approves recipes submitted by users. The administrator checks the recipes for compliance with guidelines and either approves them for public viewing or sends feedback for necessary modifications.
Preconditions:

- The Content Administrator must be logged into their administrative account.
- There must be pending recipe submissions awaiting review.

Postconditions:

- Approved recipes are made available in the public recipe database.
- If a recipe is not approved, feedback is sent to the user for modifications.

Basic Flow:.

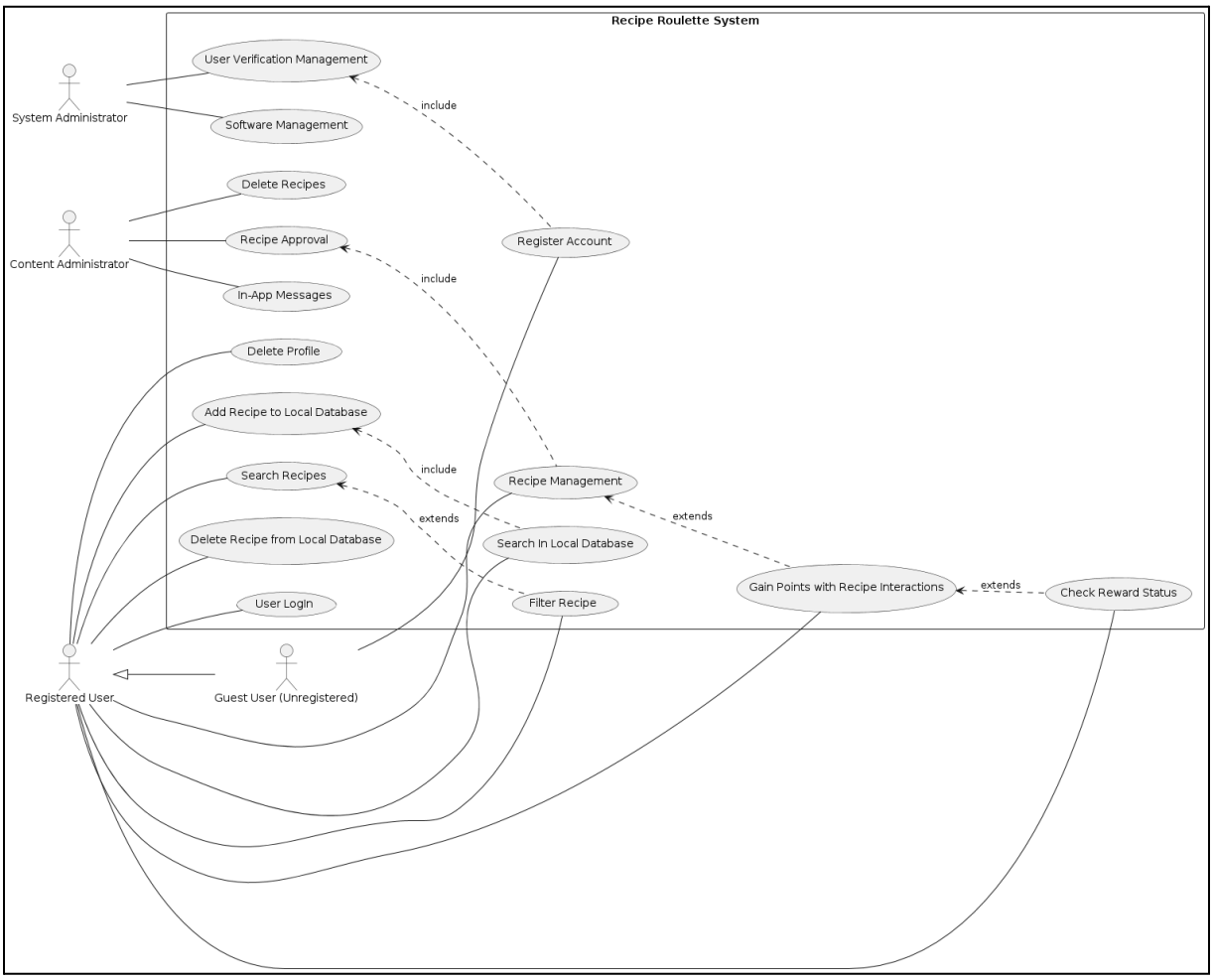
1. **Content Administrator logs in**: The Content Administrator accesses the administrative section of the app by logging into their account.
2. **Navigate to pending submissions**: The Content Administrator navigates to the section containing recipes pending approval.
3. **Select a recipe for review**: The Content Administrator selects a recipe from the list of pending submissions to review its details.
4. **Review recipe content**: The Content Administrator checks the recipe for compliance with the app's guidelines, including formatting, content quality, and adherence to dietary and safety standards.
5. **Approve or request modifications**: If the recipe meets all guidelines, the Content Administrator approves it, making it publicly available. If the recipe does not meet the guidelines, the Content Administrator sends a notification to the user with feedback on required modifications.

Alternative Flows:

- **Recipe Rejected for Modifications**:
  - If the recipe is not approved, the Content Administrator sends a notification to the user with specific feedback on what needs to be corrected.
  - The user receives the feedback and can resubmit the recipe after making the required changes.
- **Recipe Requires Minor Edits**:
  - If the recipe only needs minor edits, the Content Administrator might make these changes directly and then approve the recipe.
  - The Content Administrator documents the changes made and notifies the user about the approval.
- **Recipe Flagged for Further Review**:
  - If the recipe contains questionable content that requires a second opinion, the Content Administrator flags it for further review by another admin or a supervisor.
  - The flagged recipe is reviewed collaboratively before a final decision is made.

2.3. USE CASE DIAGRAM

Below is the detailed use case diagram for the "Recipe Roulette" application. This visual representation includes all user interactions and the relationships between different use cases.



The "Recipe Roulette System" use case diagram encapsulates a comprehensive framework for managing recipes and user interactions within a multi-faceted platform. The system is designed to accommodate a variety of users, each with distinct roles and associated functionalities. The diagram includes several actors (Registered User, Guest User, System Administrator and content administrator) and use cases for each of the actors.

Registered users have the capability to log in, manage their profiles, and interact with the system through actions such as searching and filtering recipes, adding or modifying recipes in the local database, and participating in a rewards system by earning points for their contributions. **Guest users**, while not able to contribute content or earn rewards and neither search recipes, can create an account and then have access to all the features of the app.

On the **administrative side**, system administrators oversee critical operational aspects like software management and user verification, ensuring the system maintains high standards of performance and content quality. **Content administrators** focus specifically on managing the flow and integrity of user-generated content, approving the uploaded recipes, sending feedback to users in case of any modification needed and in charge of notifications. This integrated approach not only enhances user engagement by providing tailored functionalities but also ensures robust system management and content oversight, thereby fostering a dynamic and interactive environment for all users.

Repository file path:

- PNG: DIAGRAMS/usecasediagram.png
- PUML: DIAGRAMS/usecasediagram.puml

2.4. SYSTEM REQUIREMENTS LIST

In the table below, it is displayed a general overview of the functional and nonfunctional requirements for the “Recipe Roulette" app. More detailed descriptions of each one of them can be found in the following sections.

Functional Requirements	Non-functional requirements
UFR-1: User registration	NFR-1: Data Encryption
UFR-2: User Login/Authentication	NFR-2: Fast Response Time
UFR-3: Delete Profile	NFR-3: Easy Code Maintainability
UFR-4: Search Recipes by Ingredients	NFR-4: Large User Load Handling
UFR-5: Search Recipes by Dish Name	NFR-5: Intuitive Interface
UFR-6: Filter Recipes	NFR-6: Regular Backups
UFR-7: Recipe Management	NFR-7: Constant Availability
UFR-8: Gain Points	NFR-8: Cross Platform Compatibility
UFR-9: Check Reward Status	NFR-9: Performance Optimization
UFR-10: Local Database Management	
UFR-11: Search In Local Database	
MFR-12: Management of Software	
MFR-13: User Verification Management	
CFR-14: Approve recipe	
CFR-15: Send Notification	
CFR-16: Delete recipe	
CFR-17: Revise Content	

2.5. FUNCTIONAL REQUIREMENTS

The functional requirements of the Cooking App define the core functionalities and features that the system must provide to meet the needs of its users. The following tables provide comprehensive descriptions of each functional requirement, highlighting their specific purpose, expected behavior, and the relationships with other system components.

UFR-1	User registration	Version: v2.2
Description: Users access registration, enter personal info including name, email, phone, address, gender, and age. Password creation and confirmation complete registration. First login verifies info for account creation		
Comments: If the person selects the role wrongly they can go back to select a different role. The password should follow the following requirements: - 8 to 10 characters - A combination of uppercase letters, lowercase letters, numbers, and symbols. If the information is invalid the user will return to the registration page where they have to enter their personal information to correct any errors.		
Relationships: UFR-2, UFR-3, MFR-12, CFR-15, CFR-14		

UFR-2	User Login/Authentication	Version: v2.2
Description: Users log in using their Username / email and Password		
Comments: If user fails to login after multiple attempts there could be two-factor authentication There could be a security measure so as to prevent hacks by blocking accounts after multiple failed login attempts.		
Relationships: UFR-1		

UFR-3	Delete Profile	Version: v2.2
Description: Users can delete their profile and associated data by: going to settings and Selecting 'Delete Profile' then confirming the decision twice to avoid mistakes After confirmation, the system deletes all data.		
Comments: To avoid permanently deleting all the information and progress of a user in the app by accident, we would add a warning as well as remind the user of the consequences of deleting the app accordingly.		
Relationships: UFR-2 (parent), UFR-1 (parent)		

UFR-4	Search Recipes by Ingredients	Version: v2.2
Description: Users search for recipes by entering available ingredients. They select ingredients from a list, press 'Search', and navigate through results that meet their requirements.		
Comments: Results should match with different names of ingredients (plural…) and not give back errors. If the ingredient is rare, the results should print a message that tell the user the that ingredient was not included in the results.App should be able to provide suggestions if the ingredient is rare. Users should be able to save recipes or download them to be able to access them off-line. We should monitor system performance to be able to search multiple ingredients at once. If no results are given the app could propose different related dishes or recipes.		
Relationships: UFR-1 (parent)		

UFR-5	Search Recipes by Dish Name	Version: v2.2
Description: Users search for recipes by entering the dish name. They then navigate results to find what they need.		
Comments: Users should be able to filter the recipes according to dietary restrictions or preferences. Users should be able to save recipes or download them to be able to access them off-line. As with recipes, results should be obtained based on different queries (different languages..) as well as take into account possible errors in the query. If no results are given the app could propose different related dishes.		
Relationships: URF-1 parent, UFR-6		

UFR-6	Filter Recipes	Version: v2.2
Description: Users can filter recipes based on → Time required, Food type (e.g., Indian, Mediterranean) ,Dietary restrictions (e.g., vegetarian, vegan), Difficulty level or Allergies (e.g., nuts, gluten)		
Comments: The user should be able to select from various different options.		
Relationships: URF-4, UFR-5, UFR-7, NF-3, NF-4, NF-5, NF-7, MFR-12		

UFR-7	Recipe Management	Version: v2.2
Description: Users can upload, modify, and delete recipes on the platform. They input: **Recipe Upload:** Users upload recipes and key ingredients for easy searchability. **Dietary Restrictions:** Users specify dietary restrictions the recipe can or cannot follow.		
Comments: The user should be able to delete a recipe if wanted, and to avoid problems it would need confirmation. If user violates possible dangerous information the recipe should be deleted		
Relationships: UFR-1 (parent), UFR-8, CFR-14, CFR-15		

UFR-8	Gain Points	Version: v2.2
Description: Users earn points by engaging in app activities, which can be redeemed for rewards and provides status to the user. Activities include: **Uploading Recipes:** Earn points for sharing recipes with the community. **Community Participation:** Get points for active engagement like commenting or liking. **Referrals:** Gain points by referring friends who sign up and use the app.		
Comments: The point system should be transparent, allowing users to understand how they earn points and what they can be redeemed for. The app should have a mechanism to prevent abuse or fraudulent behavior, such as spammy uploads or excessive referrals. Points should be trackable within the user profile, and users should have access to their point balance at all times. Points may have an expiration period to encourage continued engagement, with a notification system to alert users about upcoming expirations.		
Relationships: UFR-9, UFR-7, UFR-4, UFR-5, UFR-6, MFR-12		

UFR-9	Check Reward Status	Version: v2.2
Description: Users can track their points and view rewards in the app. The interface displays: **Point Balance:** Current points accumulated. **Rewards Catalog:** List of available rewards and their point costs. **Redemption:** Method to redeem points for rewards with clear instructions. **Point History:** Detailed transaction history, including earned, redeemed, and expired points.		
Comments: The rewards interface should be user-friendly and error-proof, with real-time balance updates and scalable features to accommodate growth and new rewards. Notifications can alert users about new rewards or expiring points.		
Relationships: UFR-8, UFR-7, MFR-12 , CFR-14		

UFR-10	Local Database Management	Version: v2.2
Description: This feature allows users to save their favorite recipes to their profile by clicking 'save'. A local copy of the recipe will be created on their device, enabling offline access and faster local searches. Users can initialize this feature when saving their first recipe, and can add or remove recipes as needed		
Comments: This functionality enhances user experience by providing offline access and quicker local searches. Users should ensure their local database is synced periodically to update any changes made while offline		
Relationships: UFR-8, UFR-11		

UFR-11	Search In Local Database	Version: v2.2
Description: This feature allows users to search for recipes stored in their local database. Since the recipes are stored locally, the search process is faster as it doesn't require network access or remote database queries. Users can quickly find and access their saved recipes offline.		
Comments: The local search is optimized for speed and efficiency, providing a seamless user experience even without internet connectivity. Ensure that the local database is regularly synced with the remote database to keep the saved recipes updated.		
Relationships: UFR-10		

MFR-12	Management of Software	Version: v2.2
Description: To ensure app performance and longevity, tasks include: Software Updates (Security and compliance),Performance Monitoring (Identifying and fixing issues), Error Handling (Swift response to errors), Scalability (Efficient user load management), Data Backups (Ensuring data integrity) and Security Measures (Protecting user data).		
Comments: The management of software should be proactive, focusing on preventing issues rather than just responding to them. The software management team should have clear roles and responsibilities, with processes in place for escalations and emergency response. Documentation should be maintained for all maintenance activities, including software updates, bug fixes, and performance reports.		
Relationships: UFR-1, UFR-3, CFR-14, CFR-15, CFR-16, UFR-9		

MFR-13	User Verification Management	Version: v2.2
Description: This requirement details the process of verifying newly registered users. Once a user completes the registration form, the system will automatically send a verification email to the user's provided email address. The email contains a verification link that the user must click to confirm their account. Additionally, an Administration Manager can manually verify users if necessary, especially in cases where users encounter issues with the automatic verification process.		
Comments: ● The verification email should include a unique link that expires after a certain period (e.g., 24 hours). ● If the user does not verify their account within the given time frame, they should be able to request a new verification email. ● The Administration Manager should have a dashboard to view unverified accounts and manually send verification emails or approve users if needed. ● Consider using third-party services like SendGrid or Amazon SES for reliable email delivery.		
Relationships: UFR-1, UFR-2, CFR-15		

CFR-14	Approve recipe	Version: v2.2
Description: The Content Administrator reviews submitted recipes to ensure they meet the app's quality standards, guidelines, and policies. This involves checking for complete and accurate information, compliance with community guidelines, and unique content. Approved recipes are marked for publication, while rejected ones receive feedback for improvement. Possible errors include incomplete information, guideline violations, duplicate content, and technical issues		
Comments: The approval process should be efficient to minimize delays in publishing user-submitted recipes. Content Administrators should have clear guidelines for recipe approval and consistent criteria for evaluating.		
Relationships: CFR-15, UFR-11, UFR-7		

CFR-15	Send Notification	Version: v2.2
Description: The Content Administrator sends notifications to users for different purposes, such as informing them about uploaded recipes that don't meet guidelines, requesting modifications, suggesting corrections, and notifying about rewards, points expiration, and app updates. This involves generating specific notifications and sending them to users via the app's messaging system.		
Comments: Notifications for rejected content should be clear, constructive, and personalized, with specific reasons for rejection and suggestions for improvement. Users should have a reasonable time frame to respond. The Content Administrator should have tools to track notification statuses and ensure follow-up actions.		
Relationships: CFR-14, UFR-1, MFR-12, CFR-16, UFR-9 , UFR-8, MFR-13		

CFR-16	Delete recipe	Version: v2.2
Description: The Content Administrator has the ability to delete recipes from the app, typically in response to violations of community guidelines, user requests, or inactivity. This function ensures the integrity and quality of content within the app. The process of deleting a recipe involves: - **Identify the recipe:** The Content Administrator identifies the recipe to be deleted, either from user reports, guideline violations, or system audits. - **Review and justify deletion:** Before deleting, the Content Administrator must review the reason for deletion to ensure it is justified and consistent with the app's policies. - **Delete the recipe:** Once the justification is confirmed, the recipe is deleted from the database. This removal is typically irreversible. - **Notify the user:** If the deletion is due to a guideline violation or user report, the Content Administrator sends a notification to the user explaining the reason for deletion.		
Comments: Deleting a recipe should be done with caution, ensuring that the reason for deletion is clear and justified. The Content Administrator should maintain a record of deleted recipes for auditing and accountability purposes. User notifications regarding deletions should be clear and provide guidance on next steps or appeal processes.		
Relationships: CFR-14, CFR-15, MFR-12, UFR-7		

CFR-17	Revise Content	Version: v2.2
Description: The Content Administrator is responsible for reviewing and editing existing content within the app to ensure accuracy and quality. This involves identifying content that needs revision based on user feedback, audits, or guideline changes. Once identified, the administrator revises and updates the content to correct errors, update information, or improve clarity. It's important to maintain a record of these revisions for auditing purposes. Additionally, if the revisions have a significant impact on users, the administrator communicates the changes, providing reasons and additional information for transparency.		
Comments: Revisions should be handled carefully to ensure they do not negatively impact user experience or introduce new errors.		
Relationships: CFR-15, CFR-16 , MFR-12, UFR-7		

2.6. NON-FUNCTIONAL REQUIREMENTS

The non-functional requirements of the Cooking App outline the essential criteria that determine the overall quality and performance of the system. These requirements address aspects such as performance, security, maintainability, and scalability, ensuring that the app operates reliably and efficiently under various conditions. The following sections detail each non-functional requirement, specifying the standards and guidelines that the system must adhere to in order to meet the expectations of both users and stakeholders.

NF-1	Data encryption	Version: v2.2
Type /Subtype: Security **Description:** All sensitive user data, including passwords, personal information, and recipe details, should be encrypted both during transit and at rest to prevent unauthorized access or data breaches.		
Comments: Utilize industry-standard encryption algorithms (e.g., AES) and protocols (e.g., TLS) to ensure the confidentiality and integrity of user data. Implement secure storage mechanisms and key management practices to protect encryption keys from unauthorized access. Regularly audit and update encryption protocols to address emerging security threats and vulnerabilities.		
Relationships: UFR-1, UFR-2		

NF-2	Fast response time	Version: v2.2
Type /Subtype: Performance Optimization **Description:** Ensure quick system responses to user actions, optimizing database queries, image loading, and network requests for a seamless user experience.		
Comments: Prioritize efficiency to maintain user engagement and satisfaction. Regularly monitor and update performance to accommodate increasing user loads and evolving system demands		
Relationships: UFR-4, UFR-5, UFR-7		

NF-3	Easy Code Maintainability	Version: v2.2
Type /Subtype: Maintainability **Description:** The codebase should be well-structured, documented, and adhere to coding best practices to facilitate easy maintenance by developers. Changes or updates to the code should be straightforward and require minimal effort.		
Comments: Implement modular design patterns and coding conventions to promote readability and ease of understanding. Document code comprehensively, including comments and documentation files, to aid future development efforts. Use version control systems (e.g., Git) and issue tracking tools to manage code changes and collaboration among development teams.		
Relationships: from UFR-1 to UFR-11, MFR-12, MFR-13, from CFR-14 to CFR-17, from NFR-1 to NFR-9		

NF-4	Large User Load Handling	Version: v2.2
Type /Subtype: Performance **Description:** The application should be capable of efficiently handling a large number of concurrent users without experiencing performance degradation or downtime.		
Comments: Implement scalable architecture and optimize server resources to accommodate spikes in user traffic. Utilize load balancing techniques, caching mechanisms, and horizontal scaling to distribute the load evenly across multiple servers. Conduct stress testing and performance tuning to identify and address bottlenecks before they impact user experience.		
Relationships: UFR-1, UFR-2, UFR-4, UFR-5, SF-4		

NF-5	Intuitive Interface	Version: v2.2
Type /Subtype: Usability **Description:** The user interface (UI) of the application should be intuitive and easy to navigate, allowing users of varying levels of technical expertise to interact with the app effortlessly.		
Comments: Conduct user experience (UX) research and usability testing to identify user preferences and pain points. Design the UI with clear navigation paths, intuitive layout, and consistent design elements. Provide informative feedback and guidance to users to help them accomplish tasks efficiently. Continuously gather user feedback and iterate on the interface design to improve usability over time.		
Relationships: UFR-1, UFR-4, UFR-5, SF-4		

NF-6	Regular Backups	Version: v2.2
Type /Subtype: Reliability **Description:** The application should perform regular backups of all critical data to prevent data loss in the event of system failures, disasters, or other unforeseen incidents.		
Comments: Establish automated backup processes to regularly archive user data, configuration settings, and application state. Store backups in secure and redundant storage locations to ensure data integrity and availability. Implement backup monitoring and alerting mechanisms to promptly address any backup failures or issues.		
Relationships: UFR-1, UFR-3, UFR-7		

NF-7	Constant Availability	Version: v2.2
Type /Subtype: Reliability **Description:** The app must ensure it is available to users at all times. This includes implementing redundant systems, regular backups, and robust disaster recovery plans to minimize downtime and ensure continuous operation.		
Comments: Continuous monitoring and alert systems should be in place to detect and address any issues promptly. Scheduled maintenance should be communicated to users in advance to minimize disruption.		
Relationships: UFR-1, UFR-2, UFR-4, UFR-6, UFR-7, SF-3		

NF-8	Cross-Platform Compatibility	Version: v2.2
Type /Subtype: Design Constraints **Description:** The app must be compatible with both iOS and Android platforms. It should provide a consistent user experience across different devices and screen sizes. This ensures that users can access the app seamlessly regardless of the device they are using.		
Comments: The app must be compatible with both iOS and Android platforms. It should provide a consistent user experience across different devices and screen sizes. This ensures that users can access the app seamlessly regardless of the device they are using.		
Relationships: UFR-1, UFR-4, UFR-5, SF-4		

NF-9	Performance Optimization	Version: v2.2
Type /Subtype: Design Objectives **Description:** The app should be optimized for performance, including efficient database queries, optimized image loading, and minimized network requests. Caching mechanisms should be used where appropriate to enhance speed and reduce latency. This optimization ensures a fast and responsive user experience, even under heavy load conditions		
Comments: Performance optimization should be an ongoing process, with regular monitoring and updates to ensure continued efficiency as the app scales and user numbers increase.		
Relationships: UFR-4, UFR-5, SF-4, SF-5		

2.7. EXTERNAL INTERFACE REQUIREMENTS

Description of the interfaces between the software and the world including user, hardware, software, and communication interfaces.

2.7.1. USER INTERFACES

The application should be compatible with both Android and iOS platforms, ensuring accessibility on a wide range of mobile devices. The interface should be user-friendly, easy to read, and accommodate diverse user needs, including options for different fonts. For the moment, we have in mind to start the development with only one language (English). Our future perspective is to increase the number of languages in the next releases.
Key features include intuitive navigation, responsive design, and accessibility support to enhance the user experience for all demographics.

2.7.2. HARDWARE INTERFACES

The app must support various hardware interfaces, including:

- **Mobile devices:** Smartphones and tablets running Android or iOS.
- **Local storage:** Capability to store data locally on user devices for offline access.

2.7.3. SOFTWARE INTERFACES

The application's backend will interact with several software components:

- **Database systems:** The app's database will store user information, recipes, and other relevant data. It must support CRUD operations (Create, Read, Update, Delete). The database can be managed using SQL or Oracle.
- **External notification services:** Use of external services for push notifications to inform users about updates, modifications in uploaded recipes, new recipes, and reminders.
- **Email verification services:** Integration with external email verification services to ensure secure and reliable user registration and account verification.

2.7.4. COMMUNICATIONS INTERFACES

- App notifications to follow up on progress for each user and propose new recipes
- Email for communication between app services and confirming accounts.
- Network communication for secure communication over the internet for data synchronization between the user devices and the server.

2.8. ASSUMPTIONS AND DEPENDENCIES

The success of the application depends on several assumptions and dependencies:

- **User base growth:** The app's success is dependent on active user growth and engagement. Marketing strategies and user retention plans are critical.
- **Recipe database:** Access to a comprehensive and updatable recipe database is essential to provide users with accurate and diverse recipe options.
- **Technology adoption:** It is assumed that users have access to modern mobile devices or computers with internet connectivity to use the application effectively.
- **Third-Party services:** The application may depend on third-party services (e.g. email providers, notification services) for full functionality. Reliable integration and service availability are crucial.
- **Regulatory compliance:** The application must comply with relevant data protection and privacy laws, such as GDPR or CCPA, which might affect how user data is stored and managed

3. SYSTEM ANALYSIS

3.1. PRODUCT PERSPECTIVE

"Recipe Roulette" is introduced as a new application in the culinary space, distinct from existing products. Its main feature is to recommend recipes based on user-inputted ingredients, facilitating meal planning and cooking for individuals. The app is standalone and does not serve as a continuation of or replacement for any other product.

At its inception, "Recipe Roulette" will operate independently. Future updates may, however, extend its functionality to include integration with online grocery services and smart kitchen devices to further aid the cooking process. Additionally, features to share content on social media may be added to foster a sense of community among users.

The application will source recipe information from an assortment of databases and APIs. It may also offer insights into user preferences and behaviors to those conducting market research or analytics, pending the activation of such features.

3.2. SYSTEM FEATURES

3.2.1. SF 1: RECIPE DISCOVERY ENGINE

This discovery engine is the core feature of the app, since it allows users to input the ingredients and obtain recipe suggestions that would be feasible for the ingredients in their fridges or pantries.

Functional Requirements:

- The engine must allow users to enter multiple ingredients as search criteria.
- The engine must provide recipe suggestions that utilize the inputted ingredients.
- The engine should offer the option to save favorite recipes for later reference.

3.2.2. SF 2: USER ACCOUNT MANAGEMENT

This management handles user registration, modifications and profile customization, and is in charge of data manipulation within the app.

Functional Requirements:

- Users must be able to create a new account using an email address.
- Users must be able to edit their profile information, preferences and allergies.
- The system must provide a secure method for users to delete their accounts and related data.

3.2.3. SF 3: DIETARY AND CUISINE FILTERING

This filter feature lets users apply specific filters to the recipe searches, catering to various dietary needs, like allergies or low-fat recipes, and cuisine preferences.

Functional Requirements:

- Users should be able to filter recipes by dietary restrictions such as vegetarian, vegan, gluten-free, etc.
- Users should be able to filter recipes based on dietary needs related to medical conditions such as allergies to a certain type of food or other diagnosis.
- Users should be able to filter recipes based on cuisine types; for example: Mediterranean, Asian, American, etc.
- The system must update the recipe suggestions in real-time as filters are applied.

3.2.4. SF 4: COMMUNITY ENGAGEMENT PLATFORM

The Community Engagement Platform encourages interaction among users through recipe sharing.

Functional Requirements:

- Users must be able to post and share their own recipes with the community.

3.2.5. SF 5: REWARDS AND INCENTIVES

The Rewards and Incentives feature aims to enhance user engagement by offering rewards for various in-app activities.

Functional Requirements:

- The system must track user activity points earned by sharing recipes or participating in community discussions.
- Users should be able to redeem points for in-app features or content.
- The system should display the user's rewards status and history.

3.2.6. SF 6: REAL-TIME NOTIFICATIONS

Real-Time Notifications keep users informed about updates, new recipes, and community activity.

Functional Requirements:

- Users must receive notifications for new recipes based on their preferences.
- The system must alert users about updates to the app or their community interactions.
- Users should have control over the types of notifications they receive.

3.3. OPERATING ENVIRONMENT

"Recipe Roulette" will be developed for iOS and Android platforms, ensuring wide accessibility.

3.4. DESIGN AND IMPLEMENTATION CONSTRAINTS

- **Device compatibility:** The app must be optimized for a wide range of device sizes and capabilities. This includes ensuring that the user interface is responsive and functions correctly on various screen sizes and resolutions, from small smartphones to large tablets.
- **Internet dependency:** Real-time features and recipe updates require a stable internet connection. While users will be able to save recipes locally for offline access, many functionalities, such as fetching new recipes, synchronizing user data, and receiving push notifications, will depend on an active internet connection. The app must handle scenarios where connectivity is intermittent or unavailable, providing appropriate user feedback and ensuring data integrity.
- **Content moderation:** A system for moderating user-submitted recipes to maintain quality. This includes implementing a moderation workflow where recipes submitted by users are reviewed by content administrators before being made publicly available. Automated tools may be used to flag inappropriate

content, but human oversight is essential to ensure quality and relevance. Additionally, user feedback mechanisms should be in place to report and review problematic content.

- **Scalability:** The cloud infrastructure must handle an increasing number of users and data volume. This involves designing the backend architecture to be scalable, with the ability to add more servers and resources as needed.
- **Regulatory compliance:** The app must comply with data protection and privacy laws. This includes adhering to regulations such as the General Data Protection Regulation (GDPR) in the European Union. The app must implement robust security measures to protect user data, including encryption of sensitive information, secure authentication mechanisms, and regular security audits. Additionally, users should be provided with clear privacy policies and options to manage their data, such as the ability to delete their accounts and export their data.

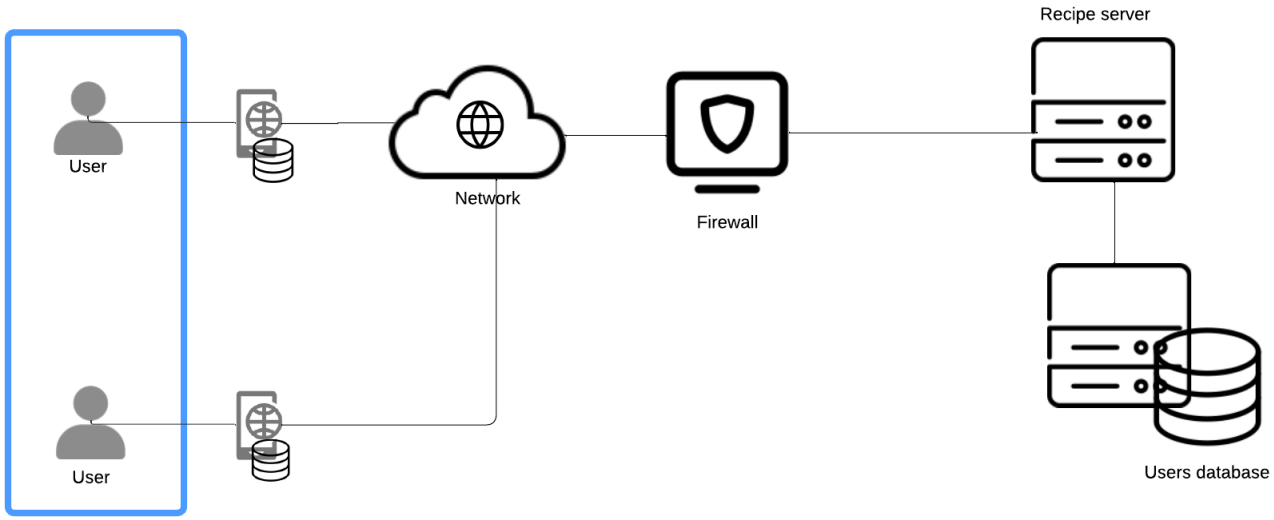
4. DESIGN ANALYSIS

4.1. ARCHITECTURAL OVERVIEW

The architectural design of the "Recipe Roulette" application follows a client-server model with a three-tier architecture comprising presentation, application, and data layers.

1. **Presentation Layer:** This layer consists of the user interface components of the application, the mobile app for iOS and Android platforms. The presentation layer is responsible for rendering the user interface elements, handling user interactions, and displaying recipe information.
2. **Application Layer:** The application layer serves as the business logic and processing engine of the system. It includes the recipe search engine, user account management, dietary and cuisine filtering, rewards and incentives system, and real-time notifications. This layer orchestrates the flow of data and logic between the presentation layer and the data layer, ensuring seamless interaction and functionality.
3. **Data Layer:** The data layer comprises the database and data storage components of the application. It stores user account information, recipe data, reward points, and other relevant data. The data layer is responsible for data retrieval, storage, and manipulation, providing the necessary data access interfaces for the application layer.

4.2. ARCHITECTURE DIAGRAM



Repository file path:

- PNG: DIAGRAMS/software_architecture_diag.png

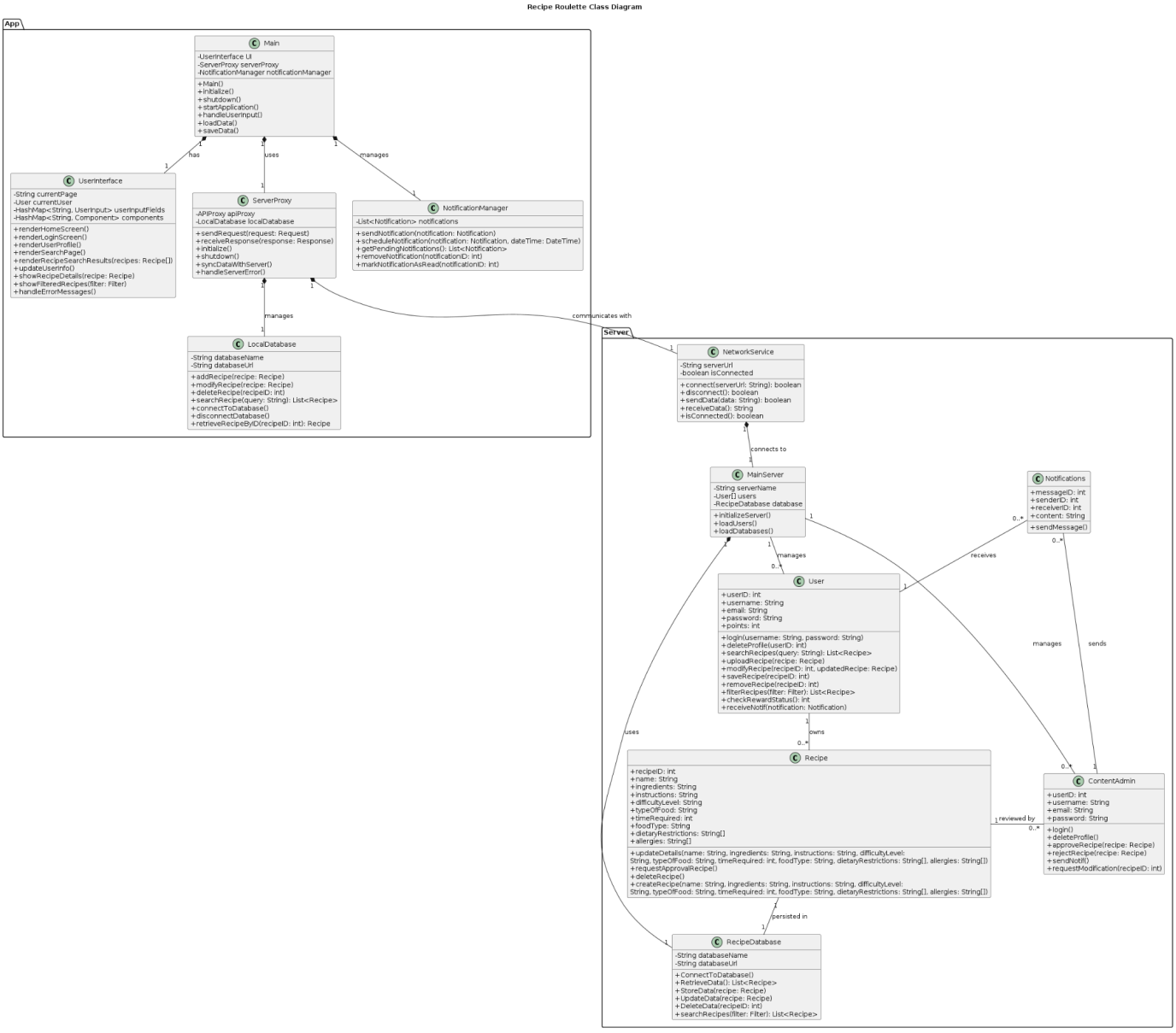
Description:

This diagram represents a client-server architecture for the “Recipe Roulette”application that allows users to search for recipes based on the ingredients they have. The architecture includes various components ensuring security, efficient data management, and smooth user interaction.

Components:

1. **Users**: These are the end-users who interact with the system via their mobile devices. They can search for recipes, upload their own recipes, and filter recipes based on their preferences.
2. **Mobile App**: The mobile application serves as the primary interface for users. It provides all necessary features such as searching, uploading, and filtering recipes. The app connects to the backend servers via the internet.
3. **Network**: The mobile app connects to the internet to communicate with the backend servers, facilitating the exchange of data between the client devices and the servers.
4. **Firewall**: A security system that monitors and controls incoming and outgoing network traffic based on predefined security rules. It provides a barrier between the secure internal network (where the servers are located) and the potentially less secure external network (where the users operate).
5. **Recipe Server**: This component is responsible for handling requests related to recipes. Interacts with the user devices via the network and firewall.
6. **Users Database**: Connected to the recipe server. Stores user information, including login credentials, profiles, and locally saved recipes. Ensures data persistence and retrieval for the application. Only autorized users (i.e. content administrators and managers) have access to it.

4.3. CLASS DIAGRAM



Repository file path:

- PNG: DIAGRAMS/class_diagram.png
- PUML: DIAGRAMS/class_diagram.puml

Description:

The class diagram for the "Recipe Roulette" system provides a detailed architectural blueprint of the software's structure, including classes, relationships, and their respective attributes and operations. This diagram is divided into two main components: Server and App, which together encapsulate the functionalities required for recipe management, user interaction, and system administration.

Server Component

- *MainServer:* This is the central class that initializes and manages the server operations. It holds the server's name, a list of users, and a recipe database. Methods include server initialization, database loading, and user sessions management.
- *User*: Represents the user entity with attributes such as userID, name, email, and password. It includes methods for managing user profiles, including updating profiles and managing saved recipes. Users can have a list of saved recipes.
- *Recipe*: Represents the recipe entity with detailed attributes including recipeID, title, ingredients, instructions, difficulty level, preparation time, serving size, food type, and dietary restrictions. Methods include creating, updating, and deleting recipes.
- *ContentAdmin*: A class for users with administrative rights to approve recipes, delete profiles, and handle content-related requests. It contains methods for managing the content on the platform, ensuring the quality and appropriateness of the recipes and user-generated content.
- *Notifications:* Handles the creation and management of notifications sent to users. Each notification includes a recipeID, message content, and a timestamp. Methods include sending notifications and managing notification preferences.
- *RecipeDatabase:* Manages the storage and retrieval of recipes from the database. Methods include adding, updating, deleting, and searching for recipes.
- *NetworkService:* Handles network communications between the client application and the server. It includes methods for starting and stopping the server, connecting and disconnecting clients, and handling data transmission.

App Component

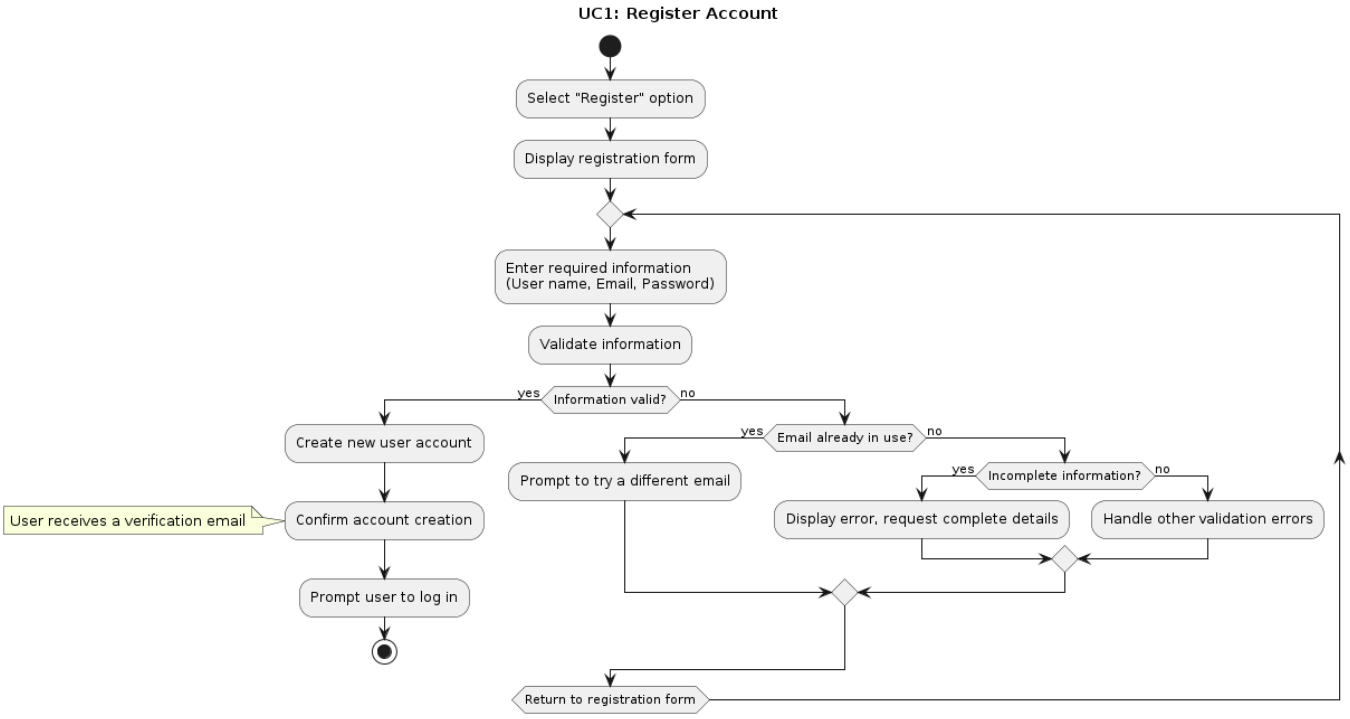
- *Main*: The main entry class for the app which initializes the user interface and server proxy, handling the application start-up and shutdown processes.
- *UserInterface*: Manages all user interface operations including rendering screens, updating user information, handling user inputs, and displaying error messages.
- ServerProxy: Acts as the mediator between the user interface and the server, managing requests and responses. This includes sending requests to fetch or submit recipes and synchronizing data with the server.
- *LocalDatabase:* Manages local data storage concerning recipes with methods to add, delete, and search recipes.
- *NotificationManager: Manages sending and receiving notifications to and from users.*

The system supports a variety of operations such as searching recipes, managing user profiles, administrating recipe content, and handling system messages.
This class diagram provides a comprehensive overview of the "Recipe Roulette" system's software architecture, delineating how different classes interact to support robust, scalable, and secure application functionality.

4.4. ACTIVITY DIAGRAMS

In the following section, it is provided a detailed description and visual representation of the flows of the different use cases.

4.4.1. UC1: Register Account



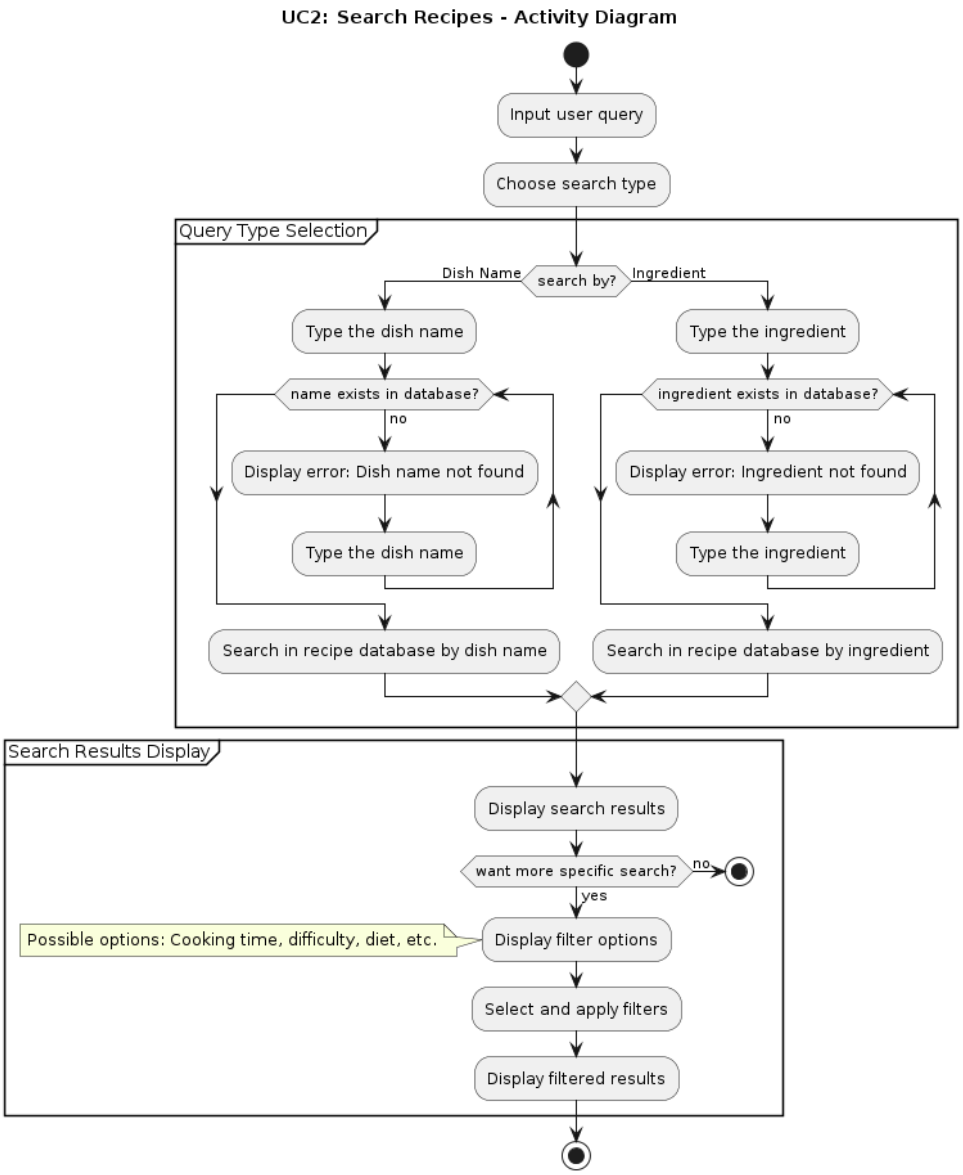
Repository file path:

- PNG: DIAGRAMS/UC1_ACTIVITY_register_account.png
- PUML: DIAGRAMS/UC1_ACTIVITY_register_account.puml

Description:

This activity diagram shows the process for new user account creation on the system. The process begins when a potential user selects the "Register" option, prompting the system to display the registration form. The user fills in their username, email, and password. The system validates the provided information, checking for completeness, uniqueness, and compliance with criteria. If the information is valid, a new user account is created and a confirmation is sent to the user. If there are validation issues, the system provides error messages and prompts the user to correct and resubmit their information.

4.4.2. UC2: Search Recipes

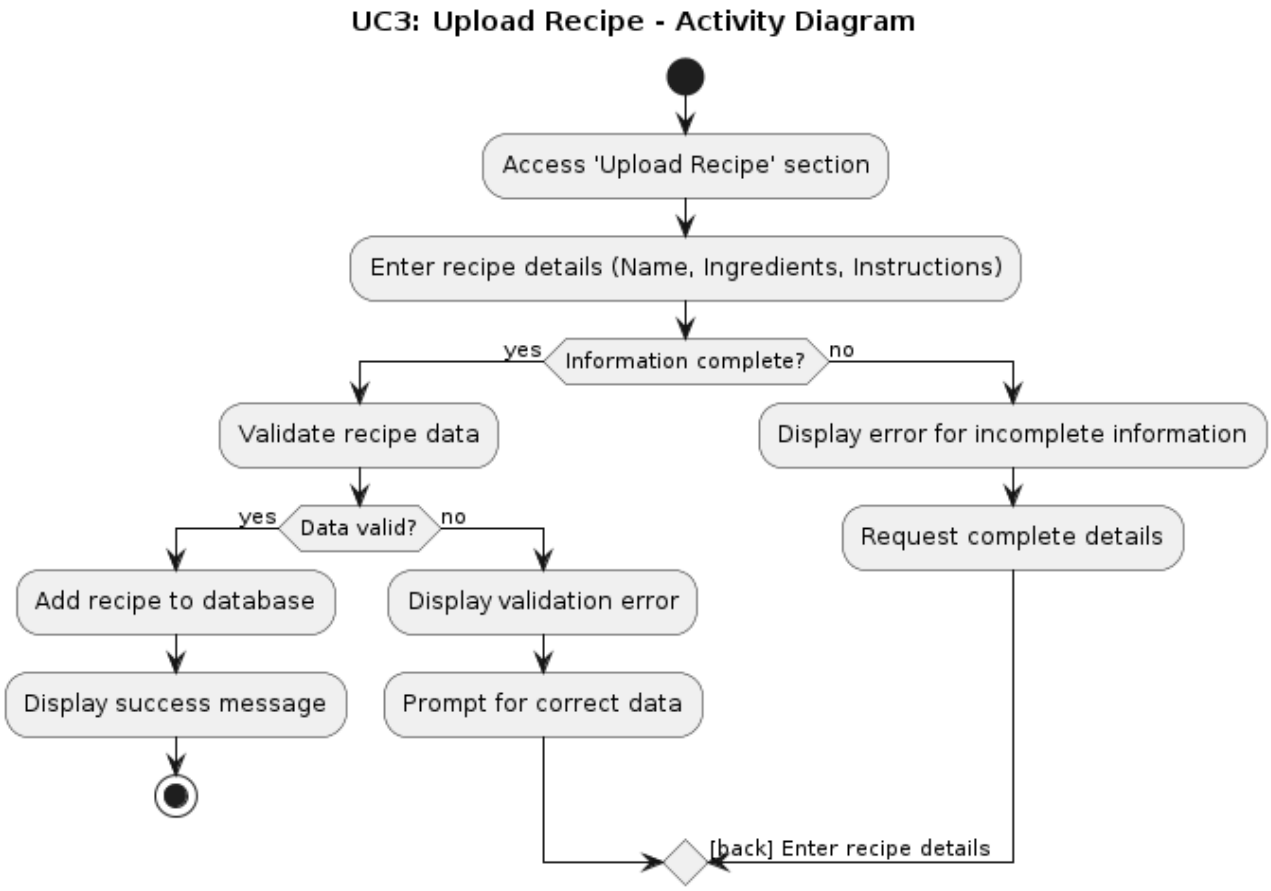


Repository file path:

- PNG: DIAGRAMS/UC2_ACTIVITY_search_recipe.png
- PUML: DIAGRAMS/UC2_ACTIVITY_search_recipe.puml

Description: The activity diagram "UC2: Search Recipes" shows how a user searches for recipes in the "Recipe Roulette" system. The user begins by selecting either to search by dish name or ingredient. If the search term is found, the system displays the results; if not, an error message prompts the user to try again. Users can then optionally apply filters to refine their search results. The process ends when search results are displayed, either filtered or unfiltered.

4.4.3. UC3: Upload Recipe



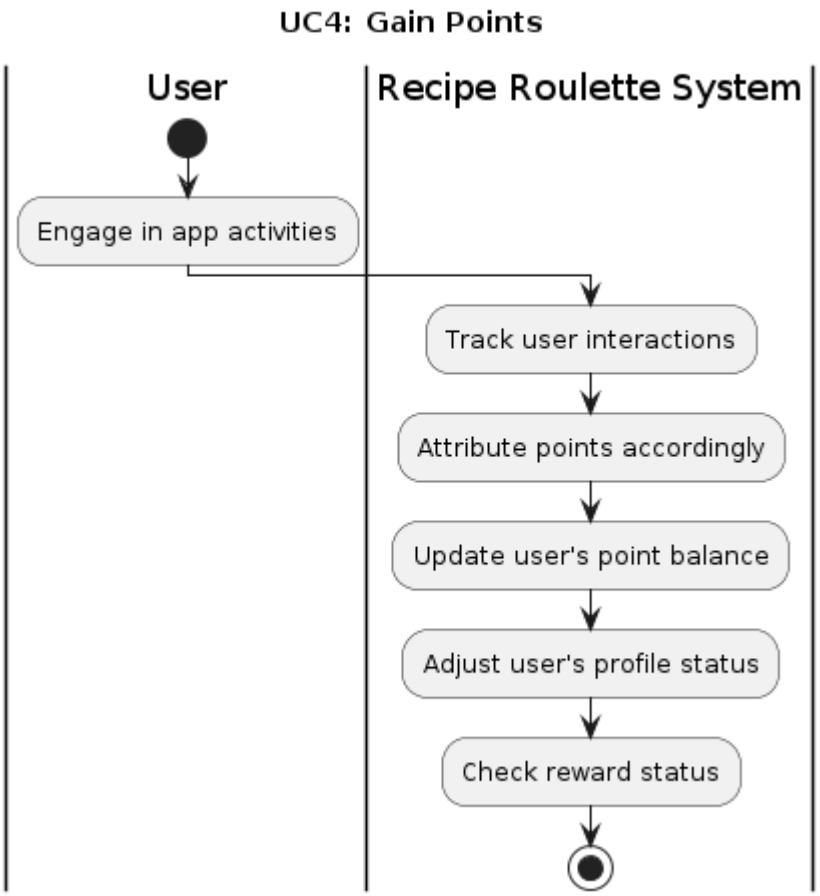
Repository file path:

- PNG DIAGRAMS/UC3_ACTIVITY_upload_recipe.png
- PUML: DIAGRAMS/UC3_ACTIVITY_upload_recipe.puml

Description:

The activity diagram maps out the steps involved in uploading a recipe within the Recipe Roulette application. It starts when a user selects the "Upload Recipe" option and enters essential details like name, ingredients, and cooking instructions. If any information is missing, an error message prompts the user to fill in the required fields. Once all data is inputted, it is validated for correctness. Successful validation leads to the recipe being saved in the database, and the user sees a confirmation message. If validation fails, the user is asked to correct any errors and resubmit the information.

4.4.4. UC4: Gain Points with Recipe Interactions



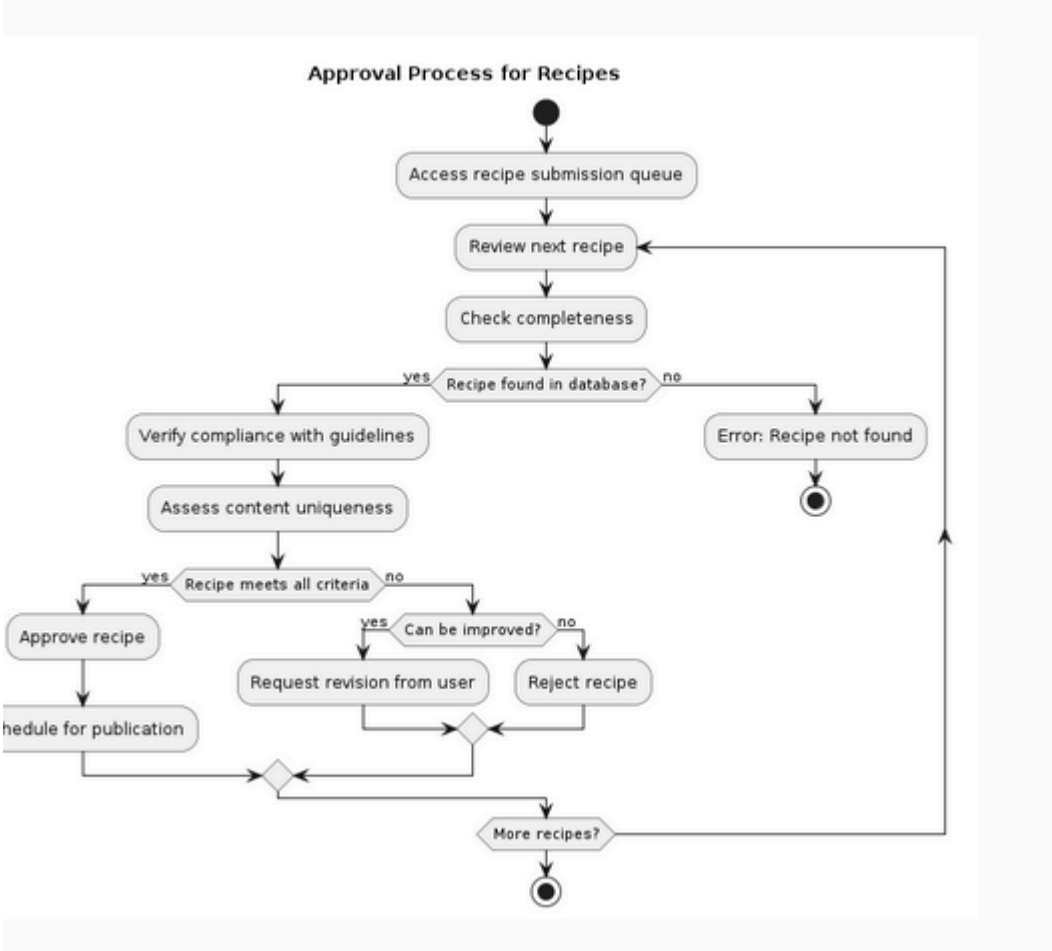
Repository file path:

- PNG: DIAGRAMS/UC4_ACTIVITY_gain_points.png
- PUML: DIAGRAMS/UC4_ACTIVITY_gain_points.puml

Description:

This sequence diagram for UC4: Gain Points depicts the interaction between the user and the Recipe Roulette System. The user engages in app activities, which the system tracks. Points are attributed based on user interactions, updating the user's point balance and adjusting their profile status accordingly. Finally, the system checks the user's reward status to determine eligibility for rewards based on accumulated points.

4.4.5. UC8: Recipe Approval



Repository file path:

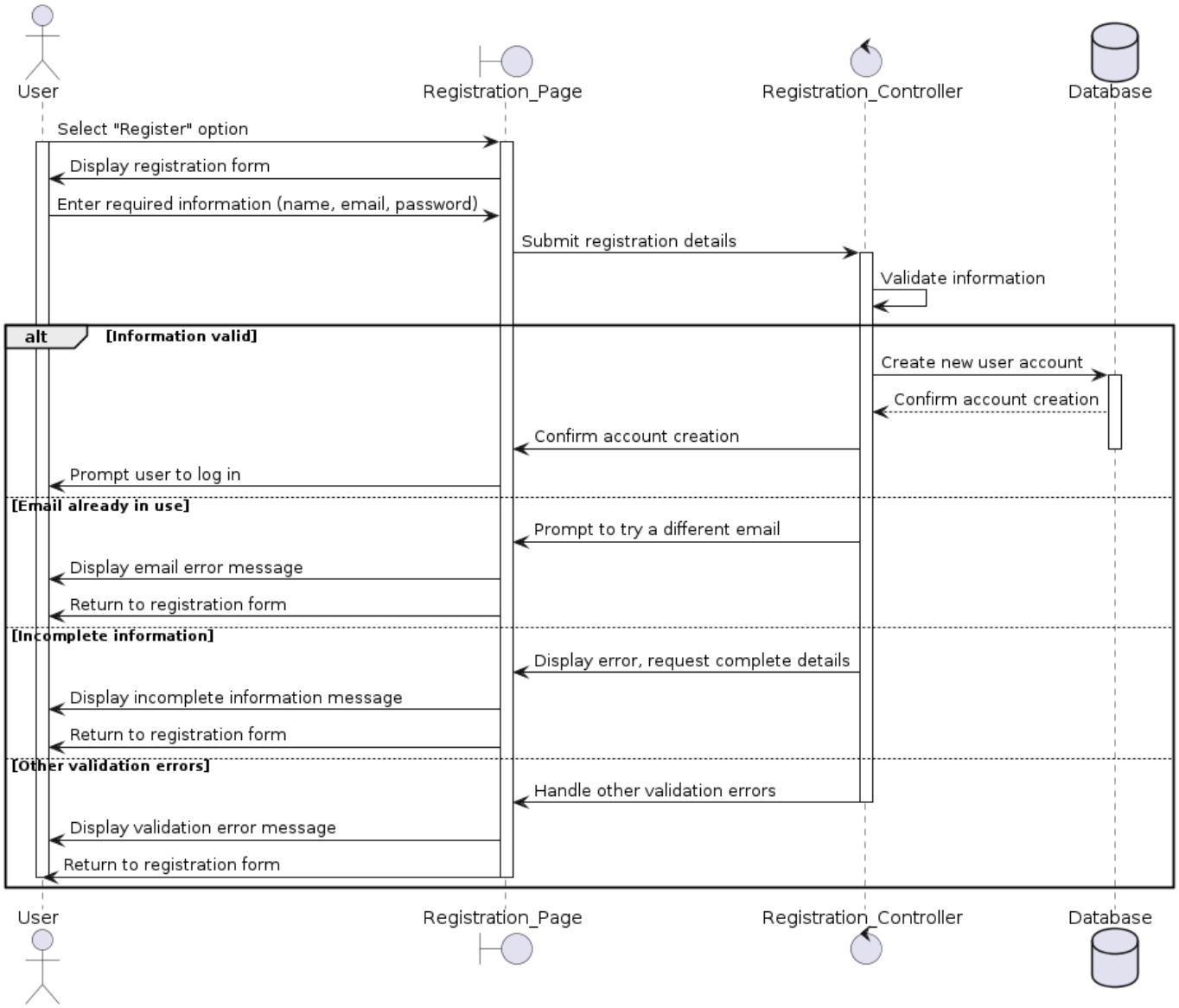
- PNG : DIAGRAMS/UC8_ACTIVITY_approve_recip.png
- PUML: DIAGRAMS/UC8_ACTIVITY_approve_recipe.puml

Description:

This activity diagram illustrates the approval process for recipes, starting from accessing the recipe submission queue. The process involves reviewing each recipe, checking completeness, and verifying if the recipe is in the database. Recipes meeting all criteria are approved and scheduled for publication, while those requiring improvement are sent back for revision, and non-compliant recipes are rejected. If a recipe is not found in the database, an error is displayed. The process continues until there are no more recipes to review.

4.5. SEQUENCE DIAGRAMS

4.5.1. UC1: Register Account



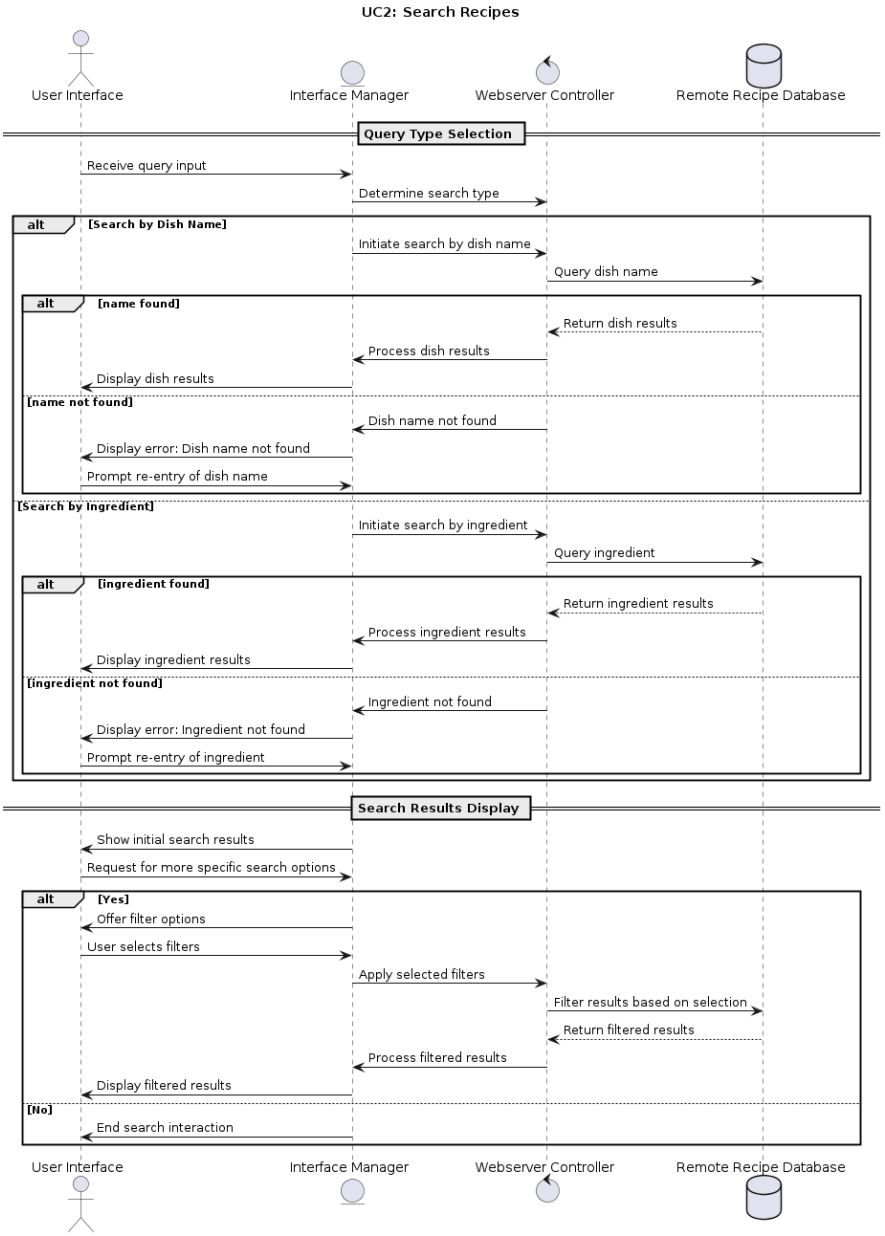
Repository file path:

- PNG: DIAGRAMS/UC1_SEQUENCE_register_account.png
- PUML: DIAGRAMS/UC1_SEQUENCE_register_account.puml

Description:

The sequence diagram depicts the registration process for new users in the Recipe Roulette application. A user initiates the process by selecting the "Register" option and entering their details on the registration page. This information is then submitted to a controller, which validates the input against the database. If the email is already in use or the data is incomplete, the system prompts for corrections. Upon successful validation, the database creates a new user account, and the user is prompted to log in, completing the registration process.

4.5.2. UC2: Search Recipes



Repository file path:

- PNG: DIAGRAMS/UC2_SEQUENCE_search_recipe.png
- PUML: DIAGRAMS/UC2_SEQUENCE_recipe_search.puml

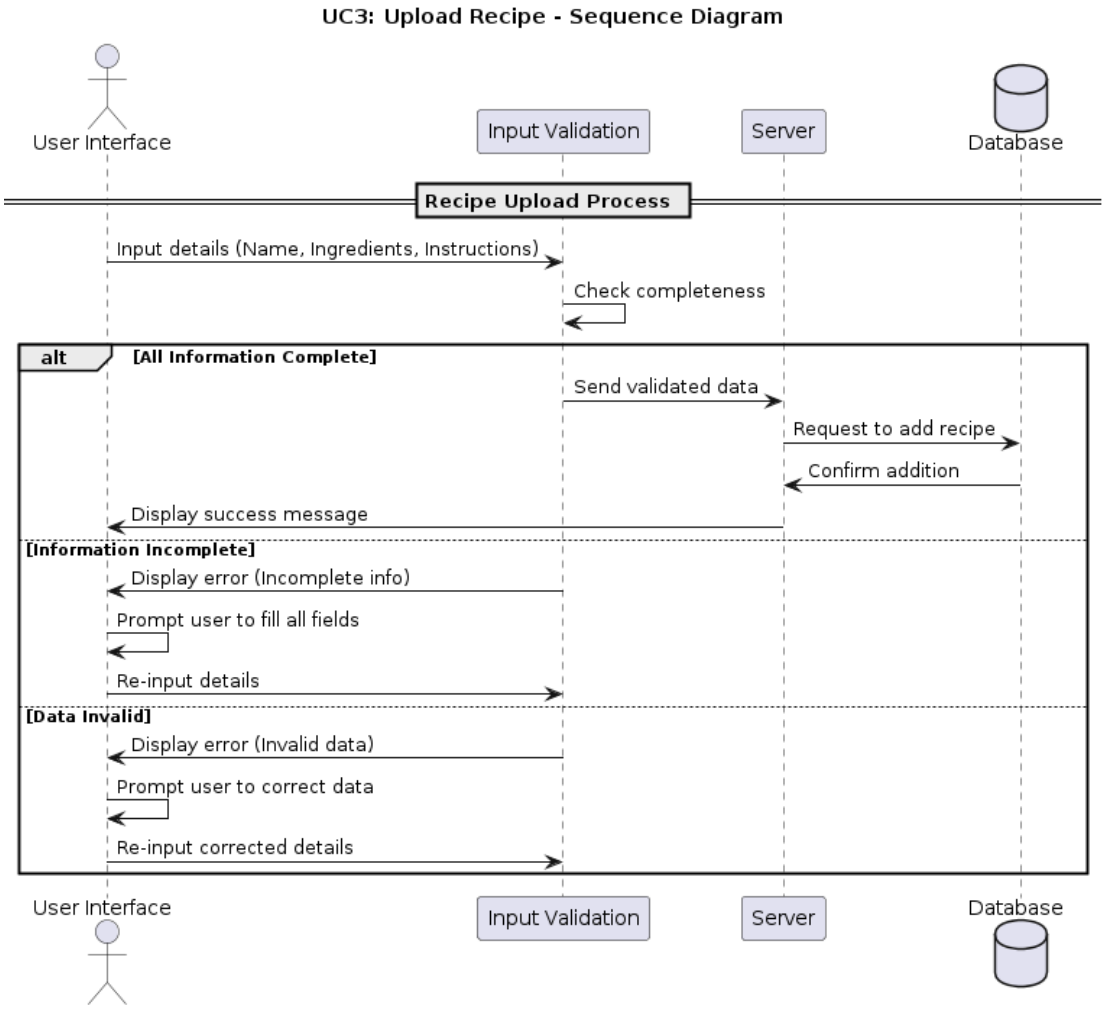
Description:

The sequence diagram outlines the process for searching recipes within the Recipe Roulette application. It begins with the user inputting a query through the user interface, which is then managed by the Interface Manager to determine the type of search: by dish name or ingredient. Based on the selected search type, the Webserver Controller either queries the dish name or the ingredient from a remote recipe database.

- **Search by Dish Name**: If the dish name is found, the database returns the results, which are then displayed to the user. If the dish name isn't found, an error message is displayed, and the user is prompted to re-enter the dish name.
- **Search by Ingredient**: Similar to the dish name search, if the ingredient is found, the results are processed and displayed. If not, an error prompts re-entry of the ingredient.

After displaying the initial search results, the user may request more specific options. The application then offers filter options such as cooking time, difficulty, or diet, which the user can select to refine the search. Once filters are applied, the database processes and returns the filtered results, which are then displayed to the user, completing the search interaction. This sequence effectively handles user queries, database interactions, and error management to enhance user experience in finding recipes.

4.5.3. UC3: Upload Recipe



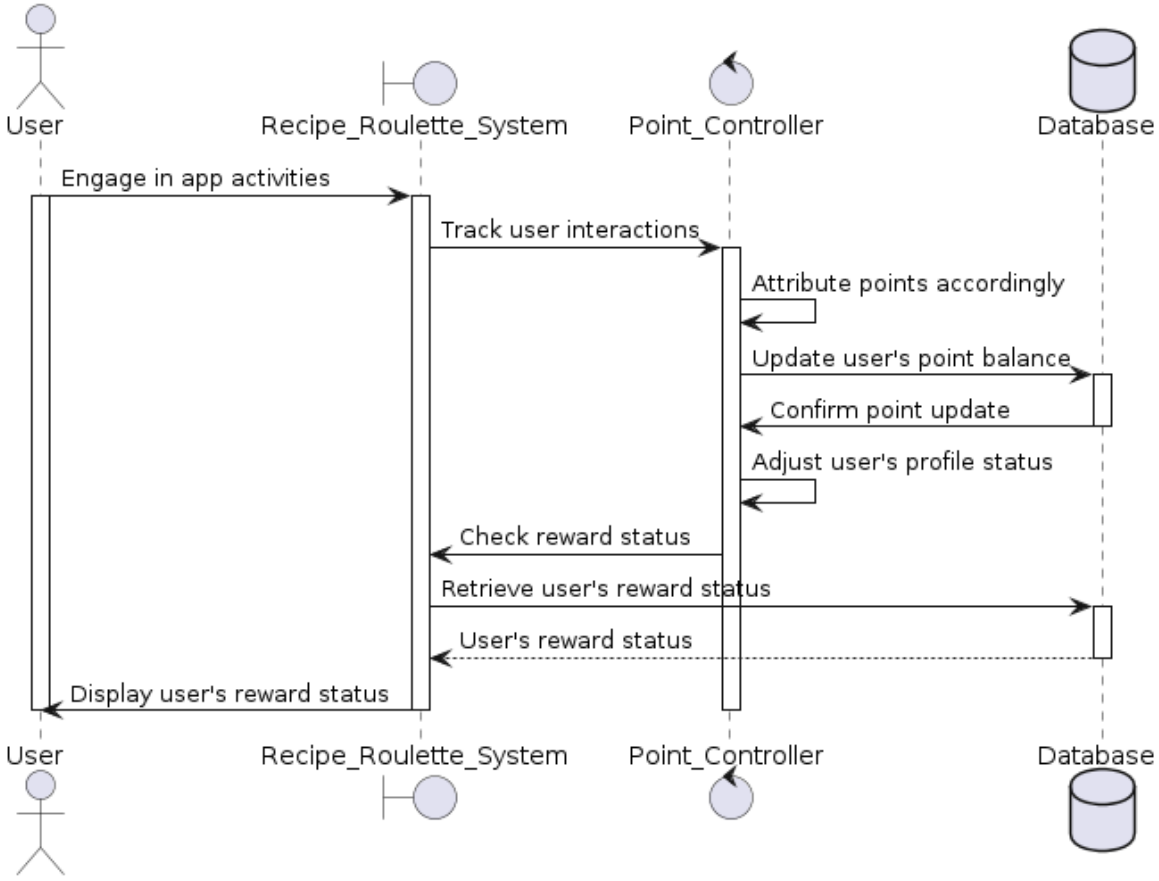
Repository:

- PNG: DIAGRAMS/UC3_SEQUENCE_upload_recipe.png
- PUML: DIAGRAMS/UC3_SEQUENCE_upload_recipe.puml

Description:

The sequence diagram for "Upload Recipe" in the Recipe Roulette app begins when a user inputs recipe details (name, ingredients, instructions) through the User Interface. This data undergoes a completeness check by the Input Validation component. If the information is complete and valid, it is sent to the server, which then makes a request to the database to add the recipe, and upon successful addition, a confirmation is sent back and displayed to the user. If the data is incomplete or invalid, error messages prompt the user for re-entry of the correct details.

4.5.4. UC4: Gain Points with Recipe Interactions



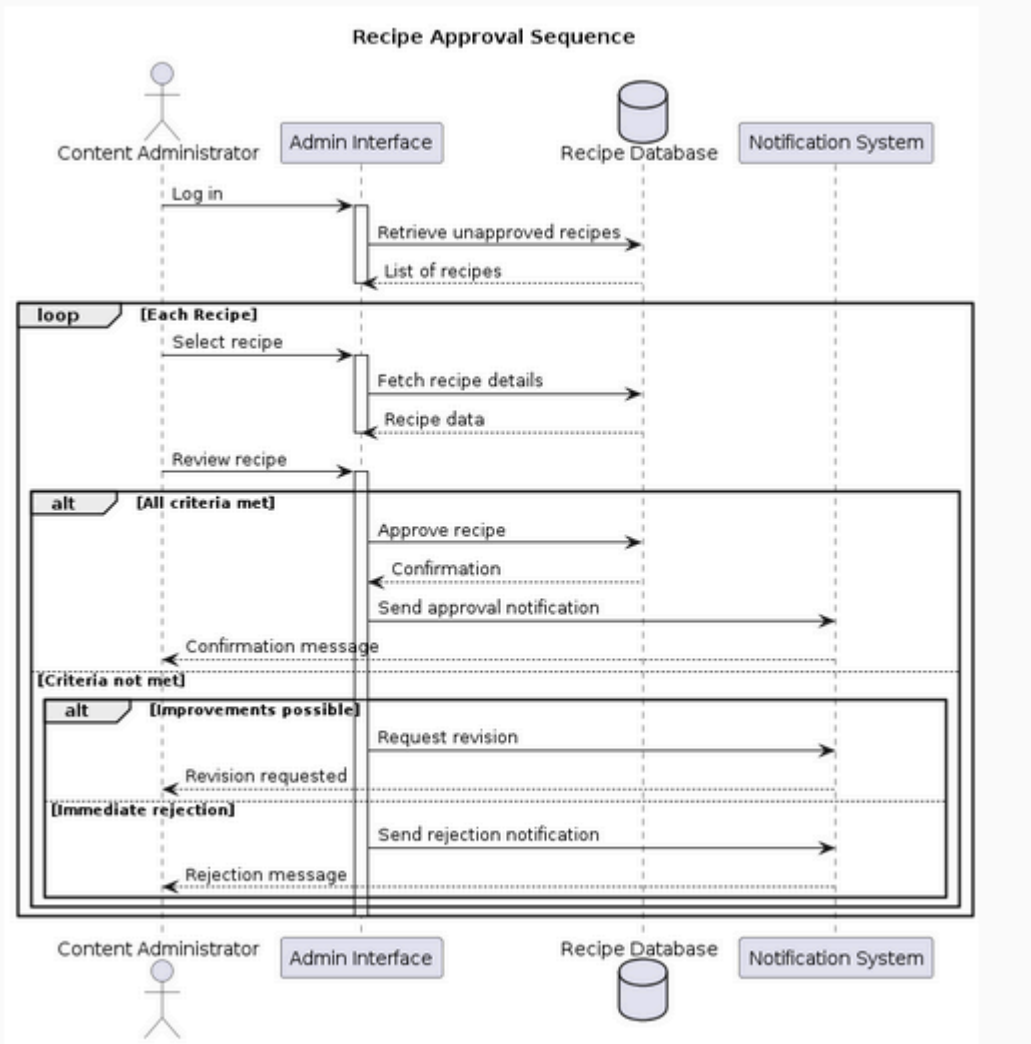
Repository:

- PNG: DIAGRAMS/UC4_SEQUENCE_gain_points.png
- PUML: DIAGRAMS/UC4_ACTIVITY_gain_points.puml

Description:

The sequence diagram illustrates the interaction process within the "Recipe Roulette" app, focusing on how users earn and track rewards. Users engage in various app activities that are monitored by the system's Point Controller, which then updates and manages their points balance in the database. Additionally, users can check their reward status at any time, prompting the system to retrieve and display the latest reward data from the database. This process ensures that users are consistently informed of their achievements and available rewards based on their interactions within the app.

4.5.5. UC7: Recipe Approval



Repository:

- PNG DIAGRAMS/UC8_SEQUENCE_approve_recip.png
- PUML DIAGRAMS/UC8_SEQUENCE_approve_recipe.pum

Description:

The sequence diagram for the "Recipe Approval Sequence" details the process of approving recipes. The Content Administrator logs in and retrieves unapproved recipes from the Recipe Database. For each recipe, the administrator reviews it, and if all criteria are met, the recipe is approved and a notification is sent. If the criteria are not met, the administrator can request revisions, which sends a revision notification. Immediate rejections trigger a rejection notification. The process repeats for each recipe until all are reviewed.