PBL 4: Software Project Final Report

Project Information

- **Course:** 53340:PBL 4: Team-based Creative Design (G1) § 53341:PBL: Team-based Creative Design (G1)
- **Project Title:** Foood: Recipe Manager
- **Team Members:**
  - **[2600240467-1] -** [Jordan Keiwein Lay]
  - **[2600240001-3] -** [Maydebura Yaroslav]
  - **[2600240283-0] -** [Kanato Nishiura]
  - **[2600240464-7] -** [Islam Md Refadul]
  - **[Student ID] -** [Student Name]
- **Date:** [January 6th, 2025]

---

1. Project Overview

1.1 System Description

Foood: Recipe Manager is a web-based application designed to address common problems in everyday meal preparation, such as recipe disorganization, inefficient meal planning, forgotten ingredients, and food waste. Many users rely on scattered notes, screenshots, or multiple applications to manage recipes, plan meals, and shop for groceries, leading to unnecessary frustration and inefficiency. This system integrates recipe storage, weekly meal planning, and automatic shopping list generation into a single platform, allowing users to organize their cooking workflow from recipe creation to grocery shopping. By centralizing these functions and providing a structured, user-specific experience through secure authentication, the system reduces cognitive load, saves time, and simplifies meal management for home cooks, families, and students.

1.2 Motivation

This system is needed to address common problems such as frustration and confusion during grocery runs, forgotten ingredients, food waste, recipe memorization, and the daily challenge of deciding what to cook. Modern households face several persistent challenges in meal management such as **Recipe Disorganization**, **Time-Consuming Meal Planning, Inefficient Shopping, Food Waste,** and **Recipe Memorization.**

By linking recipe management directly with meal planning and ingredient aggregation, the Recipe Manager streamlines the process from thinking up recipes to planning meals to grocery runs. Recipe Manager was developed to address these pain points by providing an integrated solution that reduces cognitive load, saves time and frustration for users, and promotes better resource utilization.

1.3 Objectives

The main objectives of the Foood: Recipe Manager system are as follows:

- **Centralized Recipe Storage:** Provide a structured database for storing recipes with ingredients, instructions, images, and categorization tags
- **Intelligent Meal Planning:** Enable users to plan weekly meals with smart randomization features that respect meal type categories
- **Automated Shopping List Generation:** Automatically aggregate ingredients from planned meals into a categorized, organized shopping list
- **Multi-Device Accessibility:** Support network hosting to allow access from multiple devices on the same local network
- **Recipe Sharing:** Enable users to share recipes via URL for easy distribution and import

1.4 Scope

Current Implementation:

- Complete user authentication system with registration, login, and session persistence

- Full CRUD operations for recipes with image upload support
- Weekly meal planner with 4 meal slots per day (Breakfast, Lunch, Dinner, Snacks)
- Smart and random meal plan generation algorithms
- Automatic shopping list generation with category-based organization
- Recipe search, filtering, and sorting capabilities
- Recipe sharing via base64-encoded URLs
- User profile management with settings
- Mobile-responsive UI with touch-optimized controls

Future Work:

- Nutritional information approximation
- Ingredient substitution suggestions
- Integration with external recipe APIs
- Mobile native applications (iOS/Android)
- Advanced meal plan optimization (budget, nutrition, dietary restrictions)
- Cloud deployment for public access
- Shopping list ingredient conversion according to standard grocery sizes

2. Requirements and Analysis

2.1 Functional and Non-Functional Requirements

Functional Requirements

FR1: User Management

- FR 1.1: The system shall support user registration, login, and session persistence using JWT-based authentication.
- FR 1.2: Users shall be able to update profile information and configure the meal planner randomization mode (Smart or Random).

FR2: Recipe Management

- FR 2.1 Users shall create, view, edit, and delete recipes containing ingredients, instructions, tags, and optional images.
- FR 2.2: The system shall support recipe search, filtering by tags, sorting, and paginated display.

FR3: Meal Planning

- FR 3.1: Users shall plan meals for a 7-day week with four meal slots per day.
- FR 3.2: Recipes shall be assignable, removable, auto-filled using Smart Match or Random modes, and persisted automatically.

FR4: Shopping List Management

- FR 4.1: The system shall generate a categorized shopping list based on planned meals.
- FR 4.2: Users shall manage shopping items through check, uncheck, bulk actions, and clearing completed items.

FR5: Recipe Sharing

- FR 5.1: Users shall share and import recipes via encoded URLs.
- FR 5.2: The system shall support native sharing on mobile and clipboard copying on desktop, with import confirmation.

Non-Functional Requirements

NFR1: Performance - The system shall meet responsive performance targets for API responses, frontend rendering, and database scalability.

NFR2: Usability - The UI shall be responsive across devices, easy to navigate, and provide clear feedback and validation.

NFR3: Security - The system shall securely handle authentication, password storage, token expiration, and access control.

NFR4: Reliability - The system shall maintain data consistency, support concurrent users, and handle failures gracefully.
NFR5: Maintainability - The codebase shall follow standard style conventions, RESTful design principles, and modular architecture.

2.2 Use Case Modeling

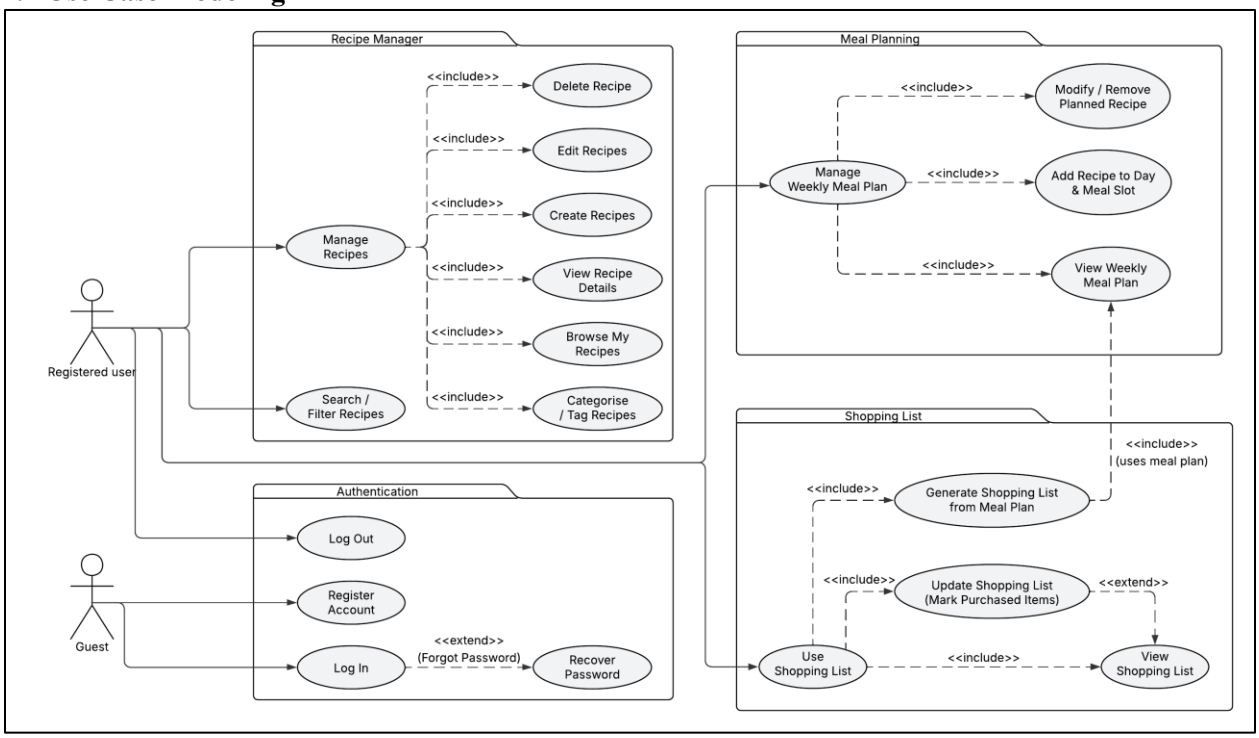

Figure 2.2: Use Case Diagram

Use Case 1: Create Recipe
Primary Actor: Registered User

Preconditions:
- User is authenticated
- User is on "My Recipes" tab

Basic Flow:
1. User clicks "+ Add Recipe" button
2. System displays recipe creation form modal
3. User enters recipe title (required)
4. User enters ingredients as line-separated list (required)
5. User enters cooking instructions (required)
6. User enters tags as comma-separated list (e.g., "Breakfast, Italian")
7. User optionally uploads recipe image from device or provides image URL
8. User clicks "Create Recipe" button
9. System validates all required fields (title, ingredients, instructions)
10. System uploads image to server (if file uploaded)
11. System creates recipe in database linked to user
12. System closes modal and refreshes recipe list

Alternative Flow 9.1: Missing required field

- 9.1.1. System displays error message specifying missing field
- 9.1.2. System prevents form submission
- 9.1.3. Resume at step 3

Alternative Flow 10.1: Image upload fails
- 10.1.1. System displays error notification
- 10.1.2. System saves recipe without image
- 10.1.3. Resume at step 11

Postconditions:
- New recipe exists in database
- Recipe appears in user's recipe list
- Image file stored on server (if uploaded)

Use Case 2: Plan Weekly Meals with Smart Randomizer
Primary Actor: Registered User

Preconditions:
- User is authenticated
- User has created at least one recipe with specific tags (Breakfast, Lunch, Dinner, Dessert/Snack)
- User is on "Weekly Meal Planner" tab

Basic Flow:
1. User views empty or partially filled 7×4 meal grid
2. User clicks "Randomize" button
3. System checks user's randomizer mode setting (defaults to Smart Match)
4. System retrieves all user's recipes with tags
5. For each of 28 meal slots:
   - 5.a. System identifies slot meal type (Breakfast, Lunch, Dinner, Dessert/Snacking)
   - 5.b. System filters recipes matching slot meal type tag
   - 5.c. System randomly selects one matching recipe
   - 5.d. System assigns recipe to slot
6. System persists entire meal plan to database
7. System updates UI to display all assigned recipes
8. User reviews generated meal plan

Alternative Flow 3.1: Full Random mode selected
- 3.1.1. System ignores tag filtering
- 3.1.2. System randomly assigns any recipe to any slot
- 3.1.3. Resume at step 6

Alternative Flow 5.1: No recipes match meal type
- 5.1.1. System leaves slot empty
- 5.1.2. Resume at step 5 for next slot

Alternative Flow 8.1: User dislikes assignment
- 8.1.1. User clicks X button on specific slot
- 8.1.2. System removes recipe from slot
- 8.1.3. User manually selects different recipe from dropdown
- 8.1.4. System saves updated meal plan

Postconditions:
- Meal plan populated with recipes
- All assignments saved to database
- Shopping list automatically updated with new ingredients

2.3 Behavioral Analysis

2.3.1 Activity Diagram

The shopping list generation process, shown in Figure 2.3.1, begins when the user opens the Shopping List tab. The system first checks if the user is authenticated; if not, they are redirected to the login page. Once authenticated, the system fetches the user's current meal plan. If no meal plan exists, an empty state message is displayed, suggesting the user plan meals first.

When a meal plan exists, the system retrieves all recipe IDs from the meal plan entries and queries the database for recipe details. The system then extracts all ingredients from the recipes and performs two parallel operations: aggregating duplicate ingredients and categorizing ingredients by type (Produce, Dairy, Meat, Grains, Pantry, Other). After sorting ingredients within categories, the system generates and displays the categorized shopping list.

Users can then interact with the shopping list by checking or unchecking items, with each action updating the item state in the database. If the user clicks "Clear Checked," all checked items are removed from the list, the database is updated, and the shopping list display is refreshed.

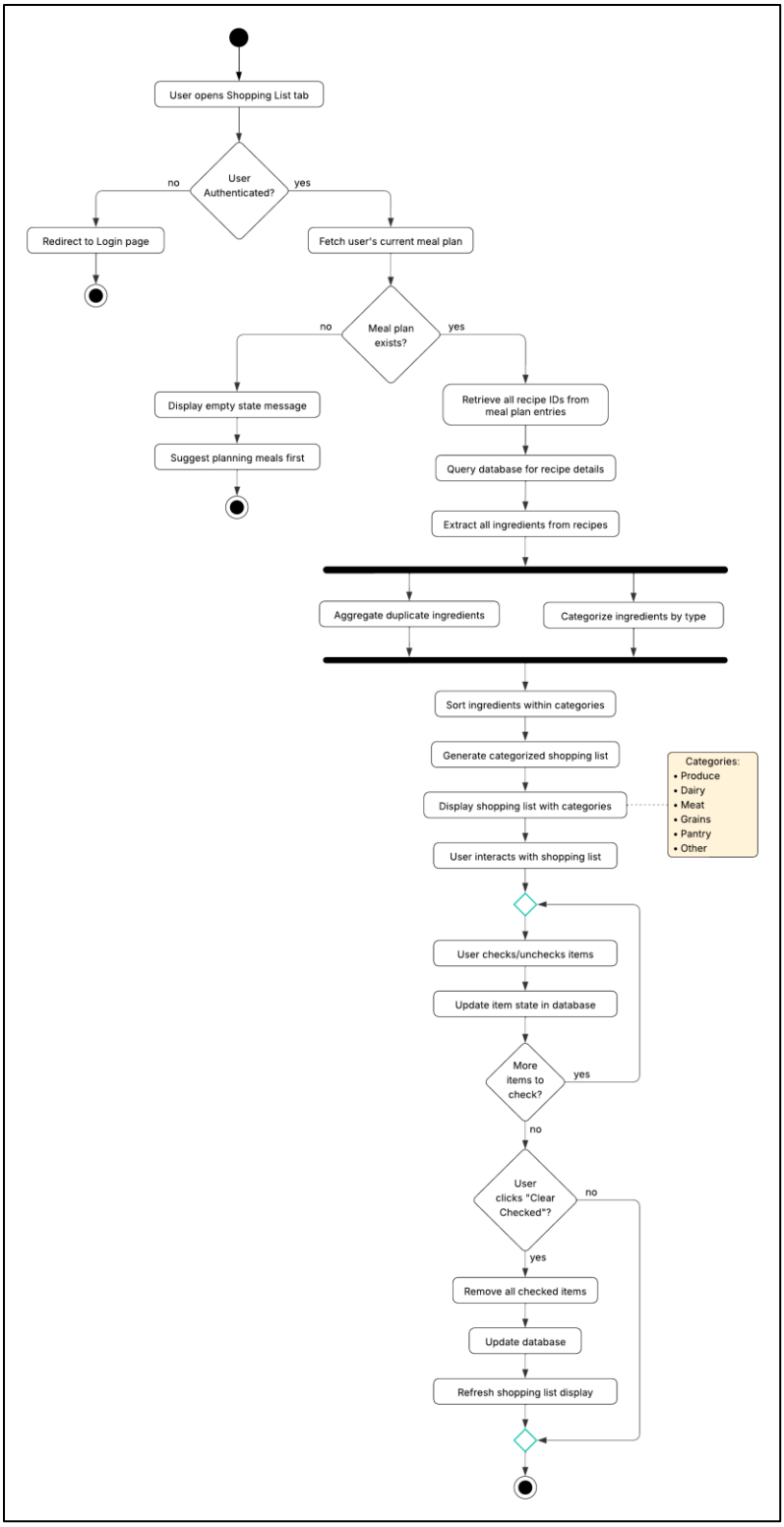

Figure 2.3.1: Activity Diagram for Shopping List

2.3.2 State Machine Diagram

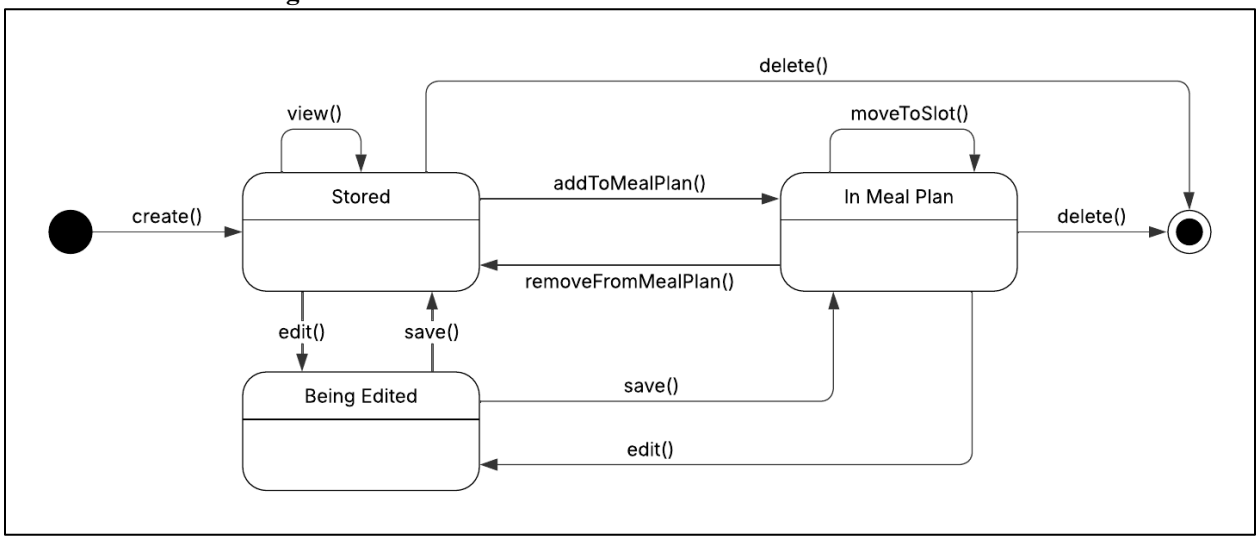

Figure 2.3.2: State Machine Diagram for Recipe Lifecycle

The Recipe state machine shown in Figure 2.3.2 models the lifecycle from creation to deletion. When a user successfully creates a recipe through the creation form (after client-side validation), it enters the **Stored** state and is persisted to the database.

From Stored, recipes can transition to **In Meal Plan** when added to the weekly meal planner via dropdown selection, making them eligible for shopping list generation. Recipes in the meal plan can move between different meal slots or be removed entirely, returning to Stored.

Users can transition a recipe from either Stored or In Meal Plan to **Being Edited** by clicking the edit button. After editing, saving returns the recipe to its previous state (Stored or In Meal Plan depending on whether it was in the meal plan before editing).

Recipes can be deleted from any state (Stored, In Meal Plan, or Being Edited), permanently removing the recipe and all associated data (ingredients, tags, meal plan entries) from the database.

3. System Design

3.1 Architectural Design

The Recipe Manager implements a 3-tier client-server architecture separating presentation, application logic, and data layers. The Frontend (React SPA) communicates with the Backend API (FastAPI) via REST HTTP, while the backend manages data through SQLAlchemy ORM connecting to a SQLite database. JWT tokens stored in browser localStorage maintain stateless authentication.

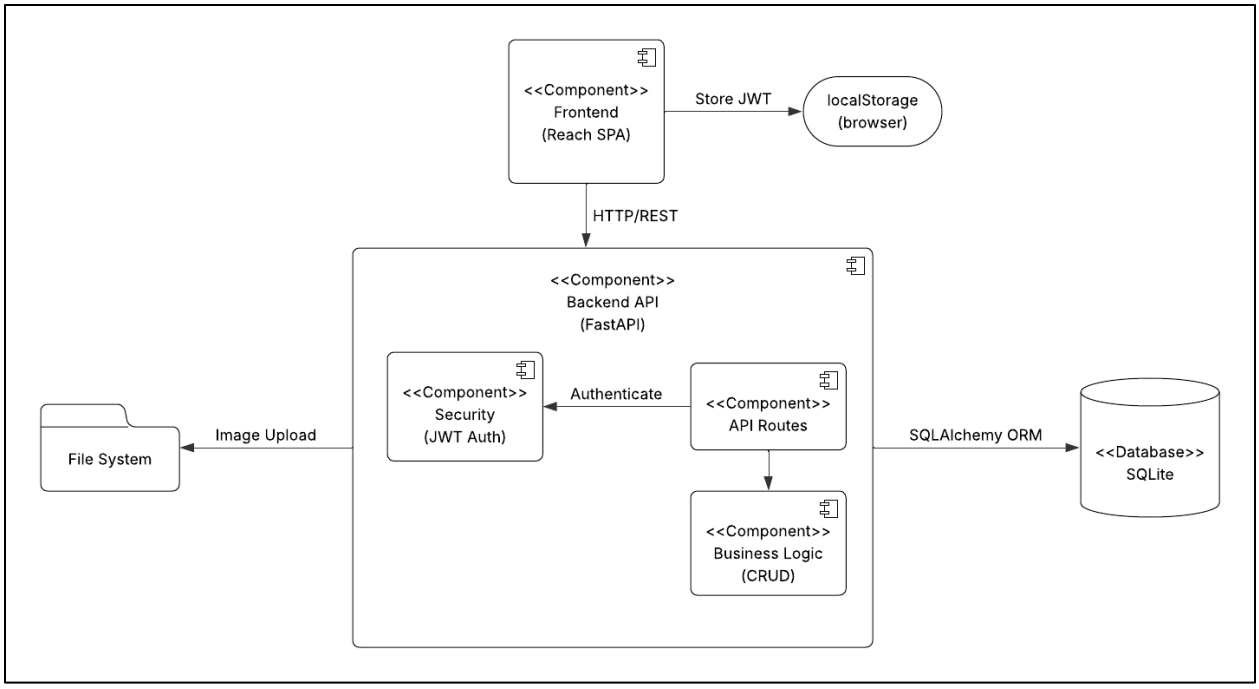

Figure 3.1: Component Diagram

The system, shown in Figure 3.1, consists of five major components. The **Frontend** (React SPA) handles user interactions and sends HTTP/REST requests to the **Backend API** (FastAPI), which contains three subcomponents: **API Routes** process incoming requests, **Security** (JWT Auth) validates authentication, and **Business Logic** (CRUD) executes database operations. The **Database** (SQLite) stores users, recipes, meal plans, and shopping lists. **localStorage** (Browser) persists JWT tokens for session management. The **File System** stores recipe images. Data flows from Frontend to Backend to Business Logic to Database, with Security validating requests at the API layer.

3.2 Database Design

The database schema consists of nine tables: four core entities (users, recipes, ingredients, tags), three planning tables (meal_plan_entries, shopping_lists, shopping_list_items), and two junction tables (recipe_ingredients, recipe_tags). All primary keys use auto-incrementing integers, with foreign keys enforcing referential integrity through SQLAlchemy ORM.

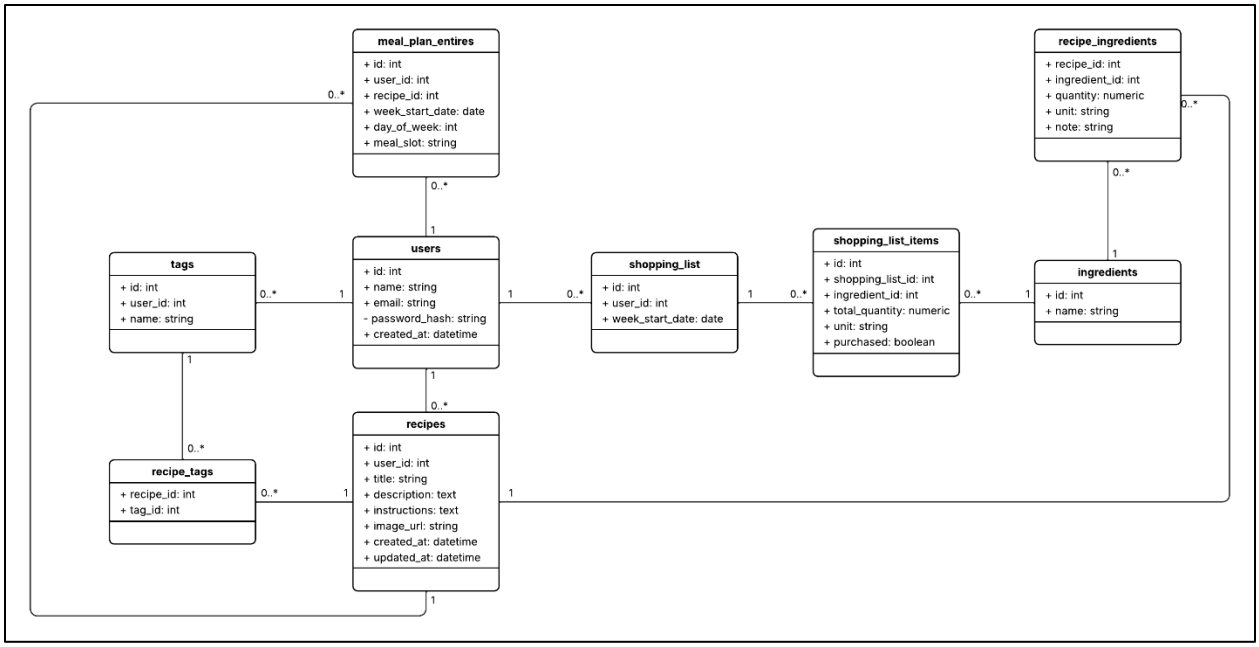

Figure 3.2: Database Class Diagram

Core entities:

- **users** stores authentication data
- **recipes** links to users and stores title, instructions, image_url, and timestamps
- **ingredients** contains unique ingredient names
- **tags** links to users for recipe categorization

Planning tables:

- **meal_plan_entries** schedules recipes with week_start_date, day_of_week (0-6), and meal_slot (breakfast/lunch/dinner/snack), linking users to recipes
- **shopping_lists** groups items by user and week_start_date
- **shopping_list_items** links shopping lists to ingredients with total_quantity, unit, and purchased status

Junction tables:

- **recipe_ingredients** implements the recipes-to-ingredients many-to-many relationship, storing quantity, unit, and note.
- **recipe_tags** implements the recipes-to-tags many-to-many relationship. All other relationships use standard one-to-many foreign key constraints.

3.3 Detailed Design

3.3.1 Class Diagram

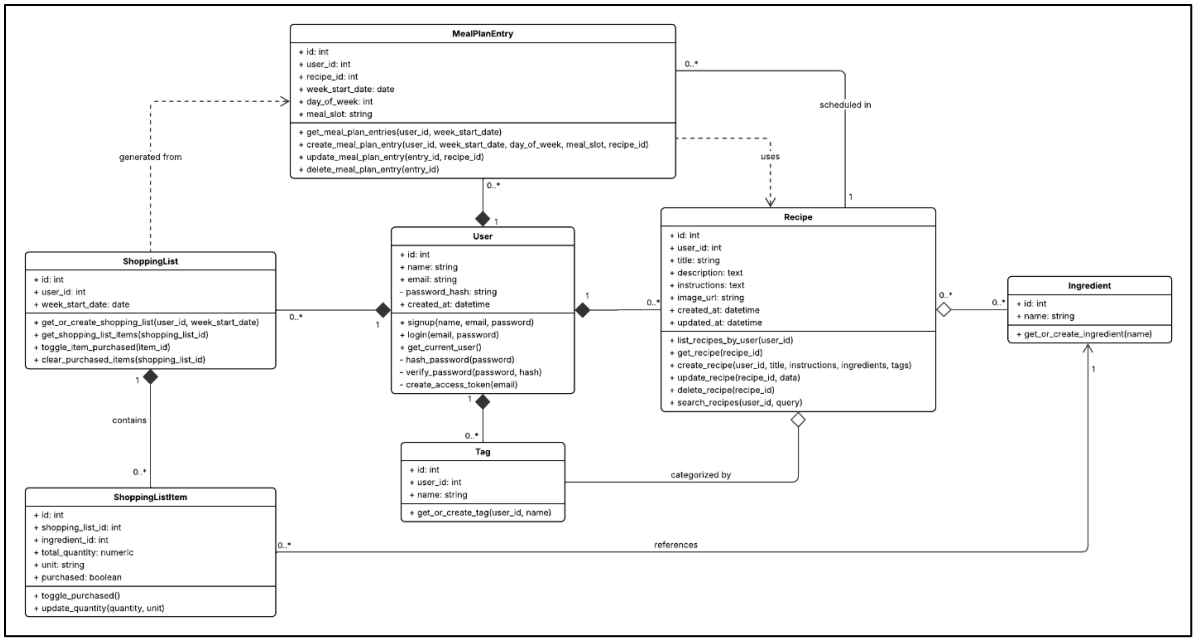

Figure 3.3.1: Class Diagram

The class diagram, shown in Figure 3.3.1, illustrates the main domain classes with their attributes and methods. The main classes are as follows:

- **User** manages authentication with password hashing and JWT token generation.
- **Recipe** handles CRUD operations including list, get, create, update, delete, and search functionality.
- **MealPlanEntry** schedules recipes by week with day_of_week (0-6) and meal_slot attributes.
- **ShoppingList** generates from meal plans and manages shopping items with purchase tracking.
- **ShoppingListItem** tracks individual ingredients with quantity, unit, and purchased status. Ingredient and Tag provide categorization for recipes.
- **Ingredient** and **Tag** provide categorization for recipes.

3.3.2 Sequence Diagram

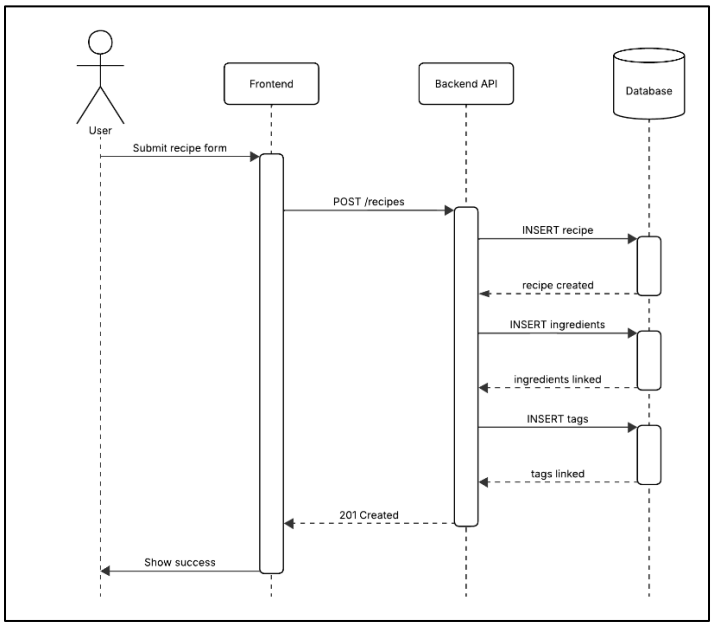

Figure 3.3.2: Sequence Diagram

The sequence diagram, shown in Figure 3.3.2, illustrates the create recipe use case. User submits recipe form through Frontend, which sends POST request to Backend API. The API executes three database operations: inserting the recipe record, linking ingredients, and linking tags. Upon success, API returns 201 Created status and Frontend displays success message to user.

3.4 User Interface Design

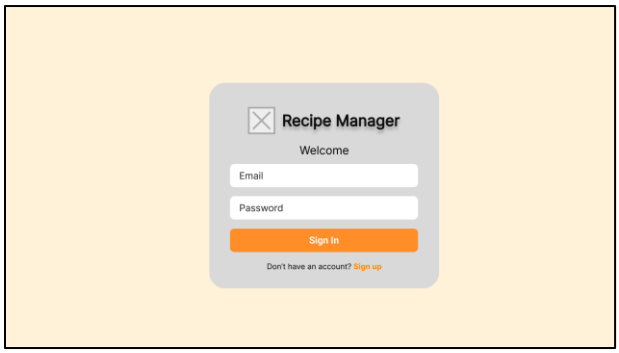

Figure 3.4.1: Login Screen

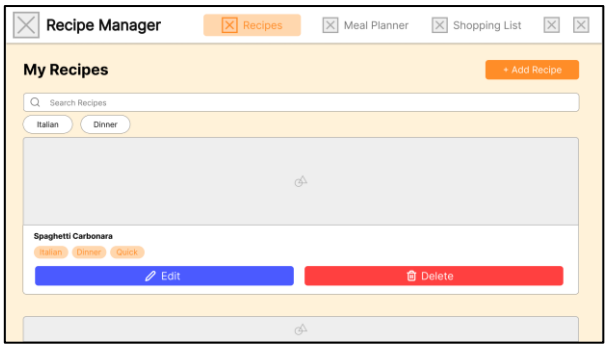

Figure 3.4.2: Recipe Screen

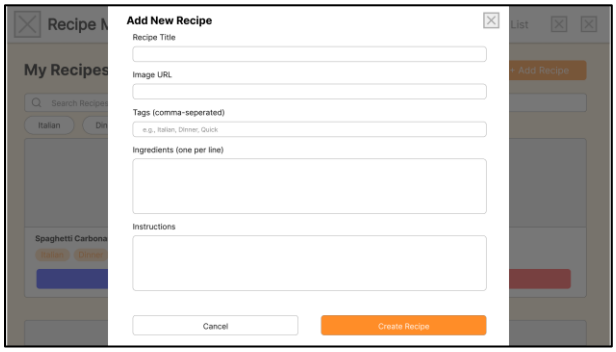

Figure 3.4.3: Recipes Popup – Add New Recipe

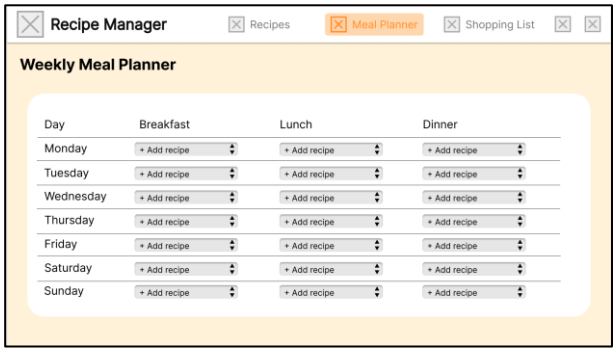

Figure 3.4.4: Meal Planner Screen

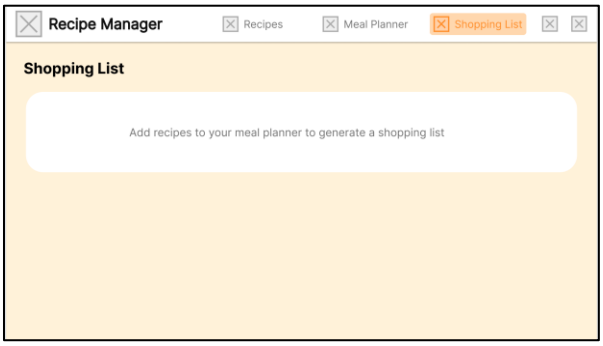

Figure 3.4.5: Meal Planner Screen

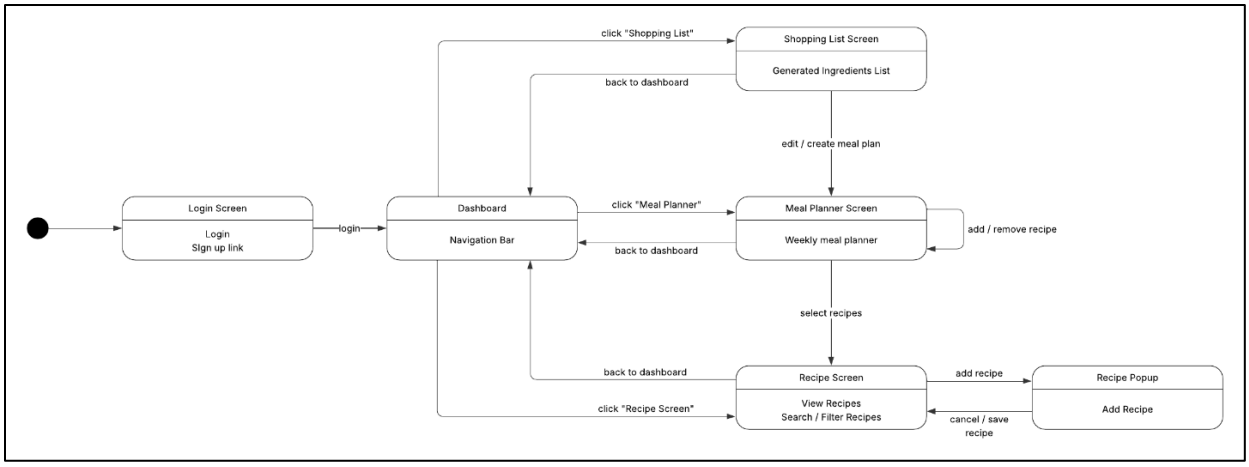

Figure 3.4.6 : Wireframe State Diagram

The Wireframe State Diagram, shown in Figure 3.4.6, illustrates the navigation flow between the main user interface screens of the Recipe Manager system. Each state represents a UI wireframe screen, and transitions show how users move between these screens through actions such as logging in, selecting navigation tabs, viewing recipes, creating new recipes, planning weekly meals, and viewing the shopping list. The diagram shows the relationship between each of the wireframes.

4. Implementation and Technologies

4.1 Technology Stack

Frontend: React 19.2 with Vite build tool provides fast development. Lucide React allows for consistent icons. CSS provides easy styling.

Backend: FastAPI enables fast API development with automatic OpenAPI documentation. SQLAlchemy ORM provides database abstraction with type-safe queries and relationship management. Python-Jose handles JWT token generation and validation.

Database: SQLite offers zero-configuration embedded database for easy development and small deployments.

Security: Hashing via Passlib provides secure password storage. JWT tokens enable stateless authentication with 60-minute expiration.

Tools: Git for version control

4.2 Implementation Highlights

Recipe Sharing: Users can share recipes via generated URLs. GET /recipes/share/{share_code} endpoint retrieves recipes without authentication, enabling public access. Share codes use base64-encoded recipe IDs. Recipients can view recipe details including ingredients, instructions, and images, with option to copy recipe to their own account.

Shopping List Generation: Users generate shopping lists from weekly meal plans. The ShoppingList class diagram shows dependency on MealPlanEntry, implemented via generate_shopping_list_from_meal_plan CRUD operation. Backend aggregates all recipe ingredients from meal plan entries for specified week, sums quantities by ingredient, and creates shopping_list_items with total_quantity and unit. Frontend displays categorized list with checkboxes for purchase tracking.

4.3 Integration and Deployment

Component Communication: Frontend communicates with Backend API via HTTP REST. All requests include JWT token in Authorization header for authentication. Backend validates tokens before processing requests. Backend uses SQLAlchemy ORM to communicate with SQLite database via SQL queries.

Architecture Consistency: Deployment matches component diagram architecture. Frontend (React SPA) runs on 0.0.0.0:5173 (accessible via localhost and network IP), Backend API (FastAPI) runs on 127.0.0.1:8000, Database (SQLite file) persists to local filesystem at `backend/recipe_manager.db`, localStorage stores JWT tokens in browser.

Deployment Steps:
1. Backend: Install Python dependencies, start server
2. Frontend: Install Node dependencies, start development server
3. Access application at http://localhost:5173 or http://[network-ip]:5173

The system runs entirely on local machine for development and testing purposes.

5. Evaluation and Testing

5.1 Testing Strategy

The application was tested on both Windows and macOS systems, with the web interface accessed from Windows, macOS, and Android devices.

Backend Testing: API endpoints validated using curl commands against the backend server running on 127.0.0.1:8000. Response codes and JSON were verified to ensure correct data formats.

Frontend Testing: User interface tested on Chrome and Edge browsers across multiple operating systems to verify responsive design and functionality. The frontend server configured with host 0.0.0.0:5173 enabled network access for cross-device testing.

Integration Testing: Browser console and backend terminal logs monitored to verify successful REST API communication and JWT authentication flow. SQLite database file directly inspected to confirm data persistence.

User Testing: Tested features across different devices and platforms to identify bugs and validate user workflows.

5.2 Test Cases

Test Case	Description	Steps	Expected Result	Status
01	Register new user account	Enter name, email, password and submit	User account created and redirected to recipe page	Pass
02	Log in with valid credentials	Enter email and password, click login	JWT token received and user authenticated	Pass
03	Verify session persistence	Log in and reload browser page	User remains logged in after page reload	Pass
04	Create new recipe	Fill form with title, instructions, ingredients, tags and submit	Recipe saved with ingredients and tags	Pass
05	View recipe details	Click on recipe card	Recipe information displayed correctly	Pass
06	Edit existing recipe	Click edit, modify fields, save changes	Recipe updates saved and reflected in UI	Pass

07	Delete recipe	Click delete button and confirm	Recipe and associated data removed	Pass
08	Share recipe via URL	Click share button on recipe	Public URL generated and accessible	Pass
09	Search recipes by keyword	Enter search term in search box	Matching recipes displayed in results	Pass
10	Filter recipes by tag	Click on tag chip	Only tagged recipes shown	Pass
11	Add recipe to meal plan	Select recipe and assign to day/meal slot	Recipe assigned to calendar slot	Pass
12	View weekly meal plan	Navigate to meal plan tab	All planned meals displayed in calendar	Pass
13	Modify meal plan entry	Click remove or update meal plan entry	Meal plan updated or entry removed	Pass
14	Generate shopping list	Click generate button in shopping list	Ingredients aggregated by category	Pass
15	Mark items as purchased	Click checkbox next to item	Item status updated with visual feedback	Pass
16	Clear purchased items	Click clear purchased button	Checked items removed from list	Pass

5.3 Validation Summary

All main functional requirements identified in the use case diagram were implemented and validated through testing. Database constraints enforce data integrity. Application performs fine for standard usage with little lag. Some of the limitations include lack of automated tests and pagination for large recipe lists due to slow loads.

6. Discussion and Lessons Learned

6.1 Key Challenges

The project presented challenges across design, implementation, and teamwork. During design, the shopping list generation algorithm needed to aggregate ingredients across meal plans while preserving quantities and units. Implementation challenges included managing synchronization between frontend and backend for real-time updates, implementing proper user authentication with data isolation, and handling ingredient deduplication when the same ingredient appeared across recipes with different units. From a teamwork perspective, merging code from different branches has led to issues, and the lack of participation of some teammates caused a lag in development and more workload on the rest.

6.2 Role of UML in Development

UML diagrams aided in the development. The use case diagram helped identify all required features early. The database class diagram was essential for understanding table relationships. The sequence diagram illustrated the flow of operations and helped identify where authentication checks and error handling were needed in API calls.

6.3 Possible Improvements and Extensions

Future improvements to the Foood: Recipe Manager include enhancing the shopping list generation to better reflect how ingredients are typically sold in grocery stores, such as converting small measurements (e.g., teaspoons or tablespoons) into standard package sizes or common retail units. Additionally, since all ingredients and quantities are already stored, the system could approximate calorie and basic nutritional values for each recipe and meal plan using external nutrition reference data. Ingredient substitution recommendations could also be introduced to help users adapt recipes based on availability, dietary needs, or preferences. These enhancements would improve practicality and usability in real-world meal planning scenarios.

7. Conclusion

This project delivered a functional full-stack web application that integrates recipe management, weekly meal planning, and automated shopping list generation into a single platform. While the system successfully implements core features such as user authentication, recipe CRUD operations, smart meal plan randomization, and recipe sharing, the overall scope was reduced from the initial objectives due to multiple setbacks encountered throughout development. Despite these limitations, the final system addresses key meal-planning problems and demonstrates a coherent and usable solution.

Through this project, I gained experience and knowledge in full-stack software design and development. Future work could focus on improving shopping list realism by normalizing ingredient quantities to standard grocery package sizes, approximating nutritional and calorie information from stored ingredients, and introducing ingredient substitution and advanced meal plan optimization features to enhance real-world usability.

8. References

FastAPI. (2024). *FastAPI documentation*. https://fastapi.tiangolo.com/

Internet Engineering Task Force. (2015). *JSON Web Token (JWT)* (RFC 7519). https://datatracker.ietf.org/doc/html/rfc7519

Lucide. (2024). *Lucide React documentation*. https://lucide.dev/guide/packages/lucide-react

Mozilla Developer Network. (2024). *Clipboard API*. https://developer.mozilla.org/en-US/docs/Web/API/Clipboard_API

Mozilla Developer Network. (2024). *Web Share API*. https://developer.mozilla.org/en-US/docs/Web/API/Web_Share_API

Passlib. (2024). *Passlib documentation*. https://passlib.readthedocs.io/

Pydantic. (2024). *Pydantic documentation*. https://docs.pydantic.dev/

Python-Jose. (2024). **Python-Jose documentation**. https://python-jose.readthedocs.io/

React. (2024). *React documentation*. https://react.dev/

SQLAlchemy. (2024). *SQLAlchemy documentation*. https://docs.sqlalchemy.org/

Tailwind CSS. (2024). *Tailwind CSS documentation*. https://tailwindcss.com/docs

Uvicorn. (2024). *Uvicorn documentation*. https://www.uvicorn.org/

Vite. (2024). *Vite documentation*. https://vitejs.dev/

9. Appendices

GitHub repository: https://github.com/PBL4-2025/PBL_GroupG_Foood