Team Deliverable 5 – Project Phase III

Project: What-Can-We-Cook?

Team: Pseudocode Warriors

Course: ITIS 3300-002

1. Introduction

1.1 Purpose

This deliverable provides the final state of the Requirements Specification and implementation summary for *What-Can-We-Cook?* Phase III. It outlines the system's objectives, requirements, architecture, implemented features, known limitations, and testing approach. The goal is to provide a clear, concise record of what the system does, how it is structured, and how it can be operated.

1.2 Scope

What-Can-We-Cook? is a web application that helps users find recipes, manage a personal pantry, and see which recipes they can cook with the ingredients they already have. The system is implemented with a FastAPI backend, an SQLite database, and a static HTML/CSS/JavaScript frontend.

In Phase III, the project delivers the final version of the system by:

- Keeping recipe search by name and ingredient with detailed recipe views.
- Supporting user accounts and a personal pantry.
- Comparing recipe ingredients to the user's pantry and marking which items are available or missing. Users can then export the list of needed ingredients for shopping.
- Providing recipe categories and category-based browsing.
- Allowing authenticated users to submit new recipes with ingredients, categories, instructions, and optional media links.
- Adding a Cooking Mode layout that rearranges the recipe detail view for easier use while cooking.

2. Overall Description

2.1 System Overview

The system operates through a browser-based interface connected to a FastAPI backend. Through a web browser, users can search for recipes, view detailed recipe information, manage a personal pantry, and submit new recipes, while FastAPI handles backend logic and database communication with an SQLite database.

In Phase III, the same architecture from earlier phases is maintained, but the implementation is finalized and refined. The system fully supports recipe search, pantry management, pantry-to-recipe comparison, category browsing, recipe submission, and now includes a **Cooking Mode** layout that rearranges the recipe view to make it easier to follow while cooking.

Layer	**Technology**	**Role**
Frontend	HTML, CSS, JavaScript, Bootstrap (icons for styling consistency)	Displays recipes, pantry, and UI logic
Backend	Python + FastAPI	Handles API requests, authentication, business logic
Database	SQLite + SQLAlchemy	Persistent storage for users, ingredients, recipes
External	Embedded YouTube videos (iframe via video URLs)	Displays instructional videos on the recipe detail page when available

2.2 Subsystems

- **Recipe Management:** Handles recipe storage, searching, and retrieval of details including ingredients, steps, and optional videos.
- **Instructional Support:** Displays step-by-step cooking instructions and embeds YouTube videos when available.
- **Pantry:** Allows users to maintain an up-to-date list of available ingredients.
- **Pantry to Recipe Comparison:** When a logged-in user views a recipe, compares the recipe's ingredients to the user's pantry and visually marks which ingredients are available and which are missing. Users can then export the list of needed ingredients for grocery shopping.
- **Recipe Submission:** Allows authenticated users to submit new recipes.
- **Cooking Mode:** Provides an alternate layout for recipe viewing, emphasizing larger text and a simplified

3. Updated Requirements (Phase III)

In earlier phases, the system requirements focused on recipe search, detailed recipe views, pantry management, pantry-to-recipe comparison, category browsing, and recipe submission. Phase III keeps this scope and finalizes the behavior of these features while adding a Cooking Mode layout for easier use during cooking

Functional Requirements

- FR1.1: ***The system shall allow users to search recipes by name.***
    - When a user enters a recipe name in the search bar, the application retrieves matching results from the project's database and displays them on the browse page. The process is handled through the backends' search logic, which filters stored recipes by name and returns relevant matches for display.

- FR1.2: ***The system shall allow users to search for recipes by ingredient.***
    - The backend compares the provided ingredient keywords with stored recipe data and returns any recipes that include them. This feature helps users discover new dishes they can make with what they already have in their pantry.

- FR 2.1: ***The system shall display the recipe title.***
    - This is handled by retrieving the corresponding title field from the database and rendering it dynamically on the page. Ensuring users can quickly identify each recipe.

- FR 2.2: ***The system shall display the recipe description.***
    - These descriptions give users a quick idea of what the recipe involves before viewing the full details. The information is stored in the backend and passed to the user interface as part of the recipe data.

- FR 3: ***The system shall provide step-by-step cooking instructions.***
    - The instructions are stored as a structured list and displayed in sequential order on the recipe detail page, allowing users to follow the cooking process from start to finish without confusion.

- FR 4: ***The system shall embed an instructional video if a valid link is provided.***
    - This provides users with optional video guidance to accompany the written steps, enhancing accessibility for visual learners.

- FR 5.1: ***The system shall allow users to add items to their pantry.***
    - When an item is added, it is saved to the project's database under the user's account, ensuring the pantry remains consistent between sessions.

- FR 5.2: ***The system shall allow users to remove items in their pantry.***
    This keeps the stored ingredient list accurate and directly affects which recipe ingredients are marked as available or missing for the user.

- FR 6.1: ***The system shall compare recipe ingredients with pantry items***.
    - This comparison identifies which ingredients the user already owns, and which are missing.

- FR 6.2: ***The system shall identify and display missing ingredients.***
    - This helps users determine what items need to be purchased before cooking a particular recipe.

- FR 7: **The system shall allow users to browse recipes by category.**
    - Recipes are associated with one or more categories. Users can select a category and view recipes that belong to it.

- FR 8: **The system shall allow authenticated users to submit new recipes.**
    - Logged in users can create new recipes by providing title, description, servings, categories, ingredients, instructions, and optional media links. New recipes are stored and available in the normal recipe views.

- FR9: **The system shall provide a Cooking Mode layout for recipe viewing.**
    - From the recipe detail page, users can switch to a Cooking Mode view that rearranges the content into a simplified vertical layout with larger text, focusing on the video (if available), ingredients, and step-by-step instructions

Non-Functional Requirements

- NFR 1: ***The system should return search results and load recipe details within seconds under normal conditions***
    - The application retrieves data from the project's SQLite database and renders it in the browser within a few seconds. FastAPI's lightweight design and asynchronous handling allow smooth interactions when users search or load recipe details, ensuring minimal waiting time even with multiple active requests.

- NFR 2: ***User pantry data shall be restricted to that user and not accessible to others.***
    - Each user's pantry information is stored securely and linked to their individual account. Access is limited through authentication checks handled by the backend, preventing other users from viewing or altering private pantry data. Sensitive actions like adding or removing items are verified against the authenticated user session.

- NFR 3: ***The interface shall be intuitive, with simple navigation and responsive design to support both desktop and mobile browsers.***
    - The system interface follows a clean and consistent layout using HTML, CSS, and Bootstrap Icons. Buttons, forms, and navigation elements are clearly labeled, with visual feedback during actions such as searches or pantry updates. The design automatically adapts to different screen sizes to maintain readability and usability on both desktop and mobile devices.

- NFR 4: ***The system shall run on common web browsers across Windows, macOS, and Linux platforms without requiring installation.***
    - The application runs directly through any modern web browser without installation. It is tested for compatibility with Windows, macOS, and Linux, ensuring that users can access it seamlessly from any standard environment.

4. System Implementation Summary

By the end of Phase III, the implementation fully supports recipe search, detailed recipe views (including optional embedded videos), user-specific pantry management, pantry-to-recipe ingredient comparison, category-based browsing, and authenticated recipe submission. A Cooking Mode layout has been added to the recipe detail view to make instructions easier to follow while cooking. Features such as favorites and more advanced filtering remain unimplemented and are considered future enhancements rather than part of the current release.

5. Compile and Run Instructions

Compile and testing have been made easy and require only few steps:

1. Set up your Python environment by running `pip install -r requirements.txt`. This can be done globally or in a virtual environment (recommended).
2. Run `python run.py`. The backend will start at **http://127.0.0.1:8000** and the static site will be served at **http://127.0.0.1:8080**. These URLs are printed in the console when the script runs.

```
Serving static website at http://127.0.0.1:8080
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [27336] using WatchFiles
INFO:     Started server process [5676]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Running Automated Tests

To run the backend unit tests, use the following steps:

1. Open a terminal in the project root directory (Pseudocode-warriors-main).
2. Make sure the Python environment is activated, and dependencies are installed.
3. Run the pytest suite.

6. User Manual

- **Home Page:** Displays recent recipes and search bar.
- **Search:** Enter a recipe name or ingredient → browse results.
- **Recipe Page:** View title, description, instructions, and optional video as well as using the option Cooking Mode for focused viewing.
- **Login:** Enter demo credentials to enable pantry features.
- **Pantry:** Add or remove ingredients and compare against recipes.
- **Pantry Comparison:** Displays missing ingredients after comparison.
- **Submit Recipe:** Used to submit new recipes and instructions.

7. Test Cases

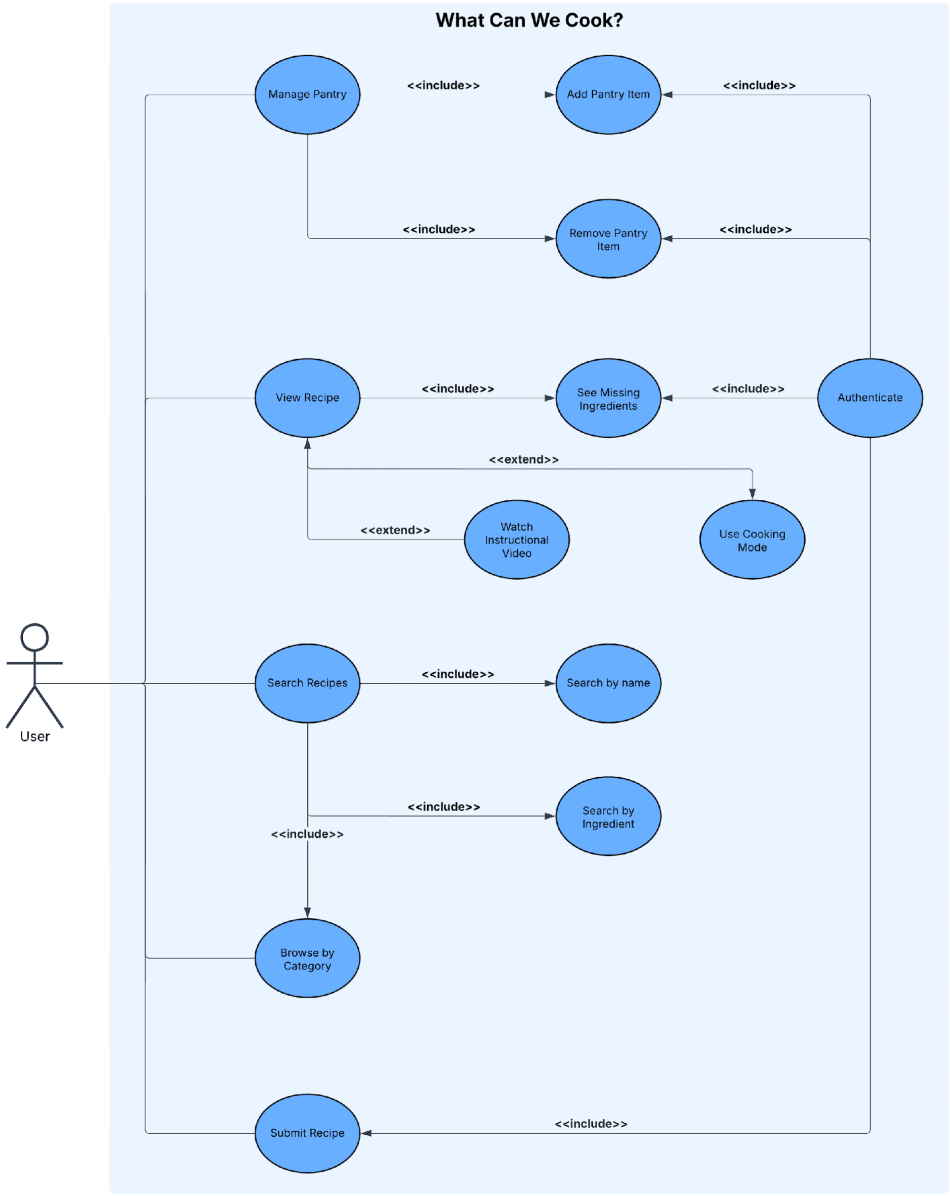
Functionality/Method	Input	Output	Pass/Fail
User registration (POST /auth/register)	JSON body with new username, email, password	200 OK; response JSON includes same username and email	PASS
Login with wrong password (POST /auth/login)	Form data with valid username but incorrect password	401 unauthorized	PASS
Get current user's pantry (GET /pantryingredient/pantry)	Request with valid authorization header	200 OK; JSON list of pantry items linked to that user	PASS
Login with valid credentials (POST /auth/login)	Form data with existing username and correct password	200 OK; JSON includes access_token	PASS
Get recipe details (GET /recipes/{id})	Valid recipe id for an existing recipe	200 OK; JSON includes id, title, description, instructions	PASS
Search recipes by keyword (GET /recipes/search)	Query parameter	200 OK; JSON list of recipes	PASS
Submit a new recipe (POST /recipes/)	JSON body with title, description, servings, ingredient id list, category id list, instructions, image url, and optional video url	200/201 OK; Response JSON includes new recipe with a generated id	PASS

8. Reflection

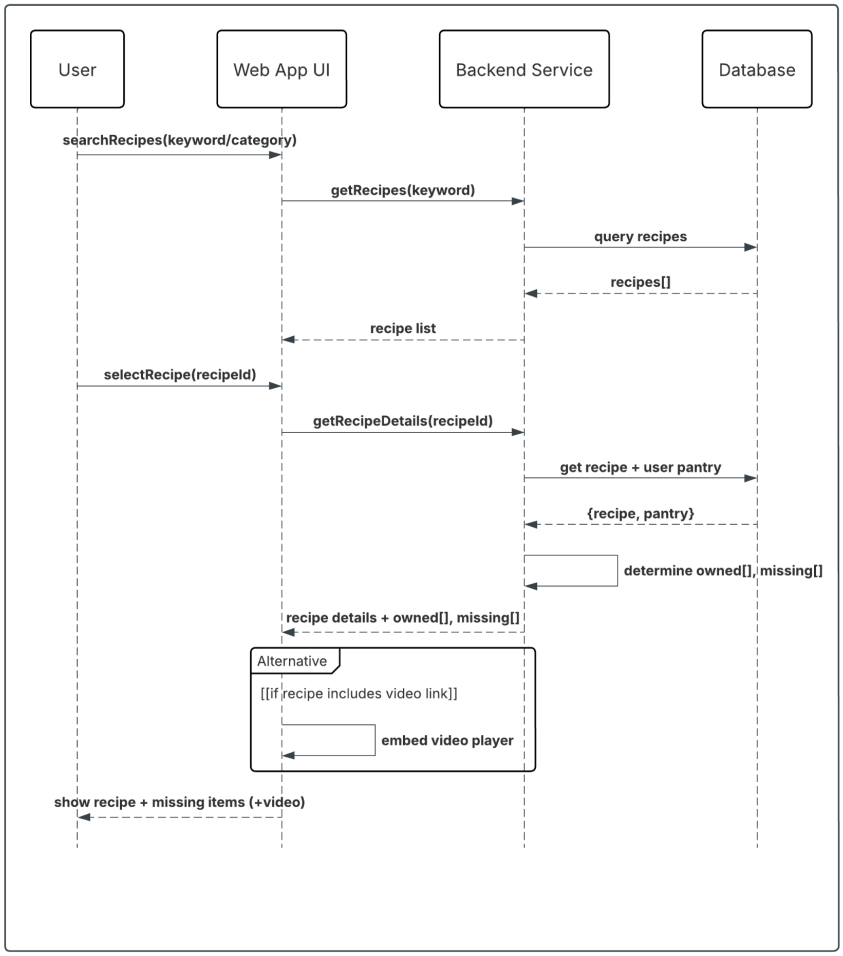
As a group, we are satisfied that we were able to deliver a working application that actually does what we set out to do. Building the system with Python and FastAPI let us get features running quickly and made it easy to experiment early on. However, as the project grew, we started to feel some pain from using Python for a larger codebase: keeping files organized, tracking data structures, and maintaining consistency across modules became harder over time, and the lack of strict typing made refactoring more risky. Going forward, we would keep the same general architecture but apply stricter project structure and type hints from the start, and we would seriously consider using a more opinionated framework or additional tooling to better support long-term growth.

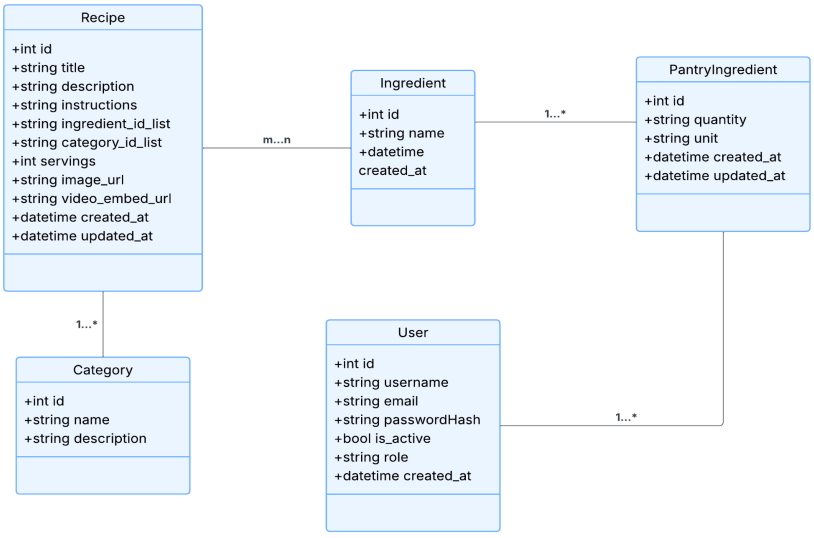
UML Diagrams

The following pages include the Class, Sequence, and Use Case diagrams for the project.



Sequence Diagram





Final Team Contributions

Member name	Roles / Contributions	Overall Contribution (%)
Ben Oz Elhadad	Designed and implemented the HTML/CSS pages and overall site visuals across all phases.	15%
Darell Isaac Sam	Implemented the Cooking Mode layout and exportable grocery list PDF feature.	15%
David Lucero	Documentation lead and coordination. Authored and updated project deliverables, diagrams, and the final report.	15%
Nicholas Smit	Lead backend developer; implemented the majority of FastAPI routes, database integration, and frontend JavaScript hooks, and resolved integration/formatting issues for the final release.	25%
Sebastian Lopez	Contributed to backend and implemented the Recipe Submission feature.	15%
Stanley Lu	Created and maintained database seed data used throughout development and testing.	15%