Team Deliverable 4 – Project Phase II

Project: What-Can-We-Cook?

Team: Pseudocode Warriors

Course: ITIS 3300-002

1. Introduction

1.1 Purpose

This document provides the updated Software Requirements Specification (SRS) and implementation summary for *What-Can-We-Cook?* Phase II. It builds on the previous deliverables by refining the original requirements, documenting newly implemented functionality and describing the subset of features that are fully implemented and testable at the end of this development phase.

1.2 Scope

What-Can-We-Cook? is a web application that helps users find recipes and track what ingredients they have on hand. It uses a FastAPI backend with a SQLite database and a static HTML/CSS/JavaScript frontend.

In Phase II, the project extends the Phase I prototype by:

- Keeping recipe search by name and ingredient and returning detailed recipe views.
- Letting logged-in users manage a personal pantry (add, edit, remove ingredients).
- Comparing recipe ingredients to the user's pantry and marking which items are available or missing.
- Adding recipe categories and category-based browsing.
- Allowing authenticated users to submit new recipes with ingredients, categories, instructions, and optional media links.

Planned enhancements such as exporting grocery lists, favorites, and more advanced filtering are not included in this phase and are deferred to future work or possibly dropped.

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 details, and manage their pantry. The frontend uses HTML, CSS, and JavaScript with Bootstrap Icons for a consistent UI style, while FastAPI handles backend logic and database communication

In Phase II, the same architecture is kept from Phase I, but the data model and endpoints are extended to support recipe categories, user-submitted recipes, and pantry-aware recipe views that show which ingredients a logged-in user already has, and which are missing.

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.
- **Recipe Submission (Phase II implementation):** Allows authenticated users to submit new recipes.

3. Updated Requirements (Phase II)

In Deliverable 2, Phase II was originally planned to focus on enhancements such as grocery list export, a recipe submission and approval workflow, tags/filters/favorites, and broader UI improvements. During implementation, the team adjusted the scope to prioritize stabilizing the core system and delivering features that directly build on the existing Phase I work.

Specifically, we:

- Implemented recipe submission without an approval queue and added recipe categories with category-based browsing.
- Implemented pantry aware recipe views that compare required ingredients against the user's pantry and visually indicate available vs. missing ingredients.
- Dropped grocery list export, favorites, advanced filters, and other stretch enhancements to a future phase due to time constraints.

The functional requirements listed below reflect this updated Phase II scope and align with the features that are implemented and testable in the current prototype.

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.

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. 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.

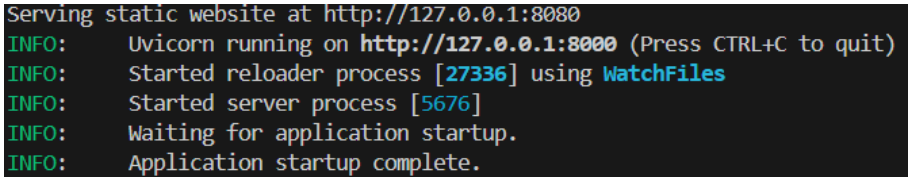
4. System Implementation Summary

In Phase II, we kept the core system from Phase I and added several new features on top of it. The backend now includes a Category model and related API endpoints so recipes can be tagged and browsed by category. We also added a recipe creation endpoint so authenticated users can submit new recipes with ingredients, categories, instructions, and optional media links. The recipe detail page is now pantry-aware: when a logged-in user views a recipe, each ingredient is checked against their pantry and visually marked as available or missing. The My Pantry page was improved with ingredient autocomplete, a searchable table, and clearer feedback messages. Finally, the backend test suite was expanded and organized by feature using pytest, making it easier to verify that the main API endpoints continue to work.

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.



Running Automated Tests

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

1. Open a terminal in the project root directory and navigate to src/api/tests.
2. Make sure the Python environment is activated, and dependencies are installed.
3. Run the test suite via pytest. Results are printed to the console.

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.
- **Login:** Enter demo credentials to enable pantry features.
- **Pantry:** Add or remove ingredients and compare against recipes.
- **Pantry Comparison:** When viewing a recipe while logged in, ingredients you have are marked as available and missing ingredients are highlighted.
- **Submit Recipe:** Used to submit new recipes and instructions.

7. Test Cases

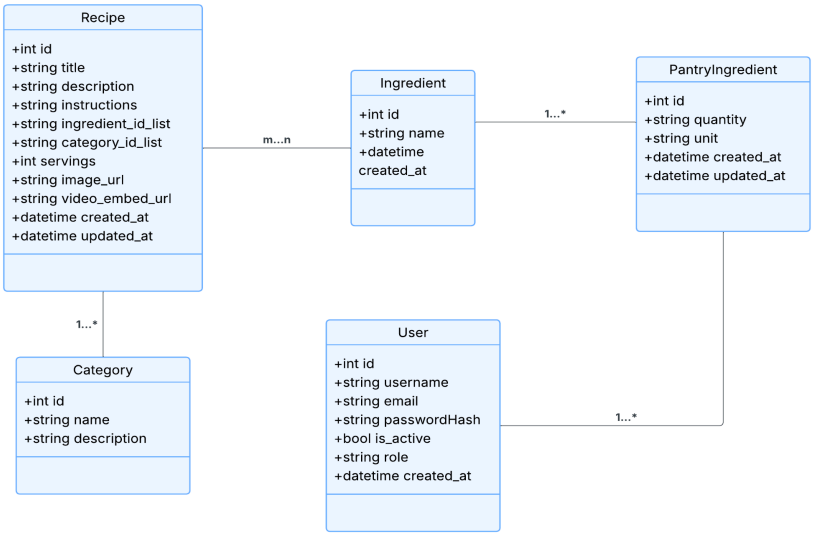
Functionality/Method	Input	Output	Pass/Fail
User registration (POST /auth/register)	JSON body with new username, email, password	200 OK; response JSON includes same username and email	PASS
Login with wrong password (POST /auth/login)	Form data with valid username but incorrect password	401 unauthorized	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

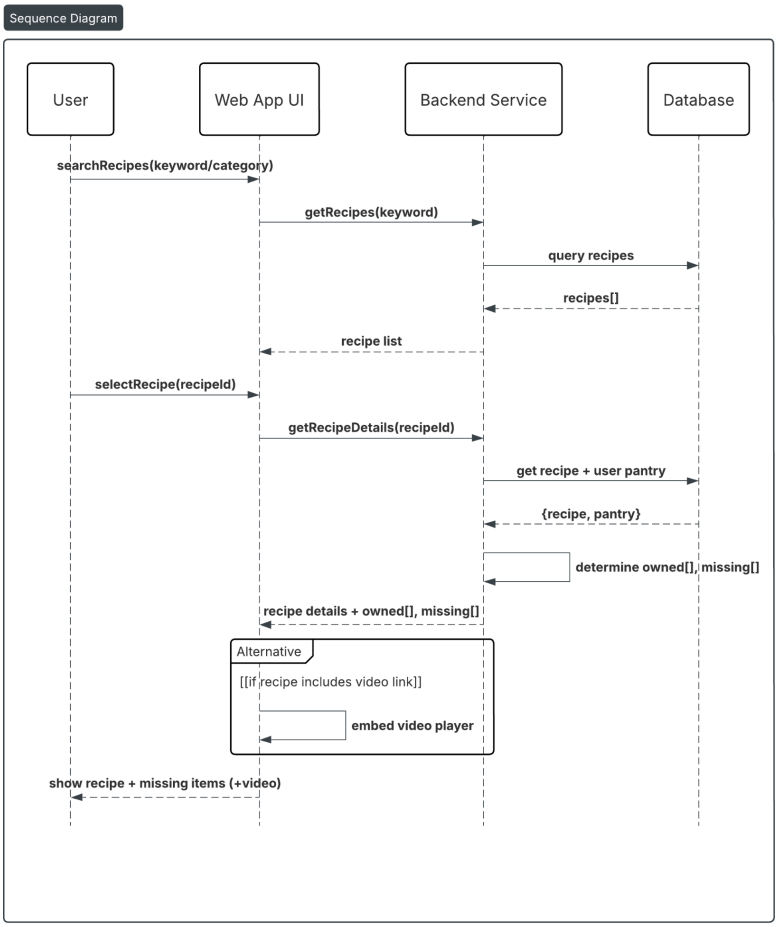
8. Reflection

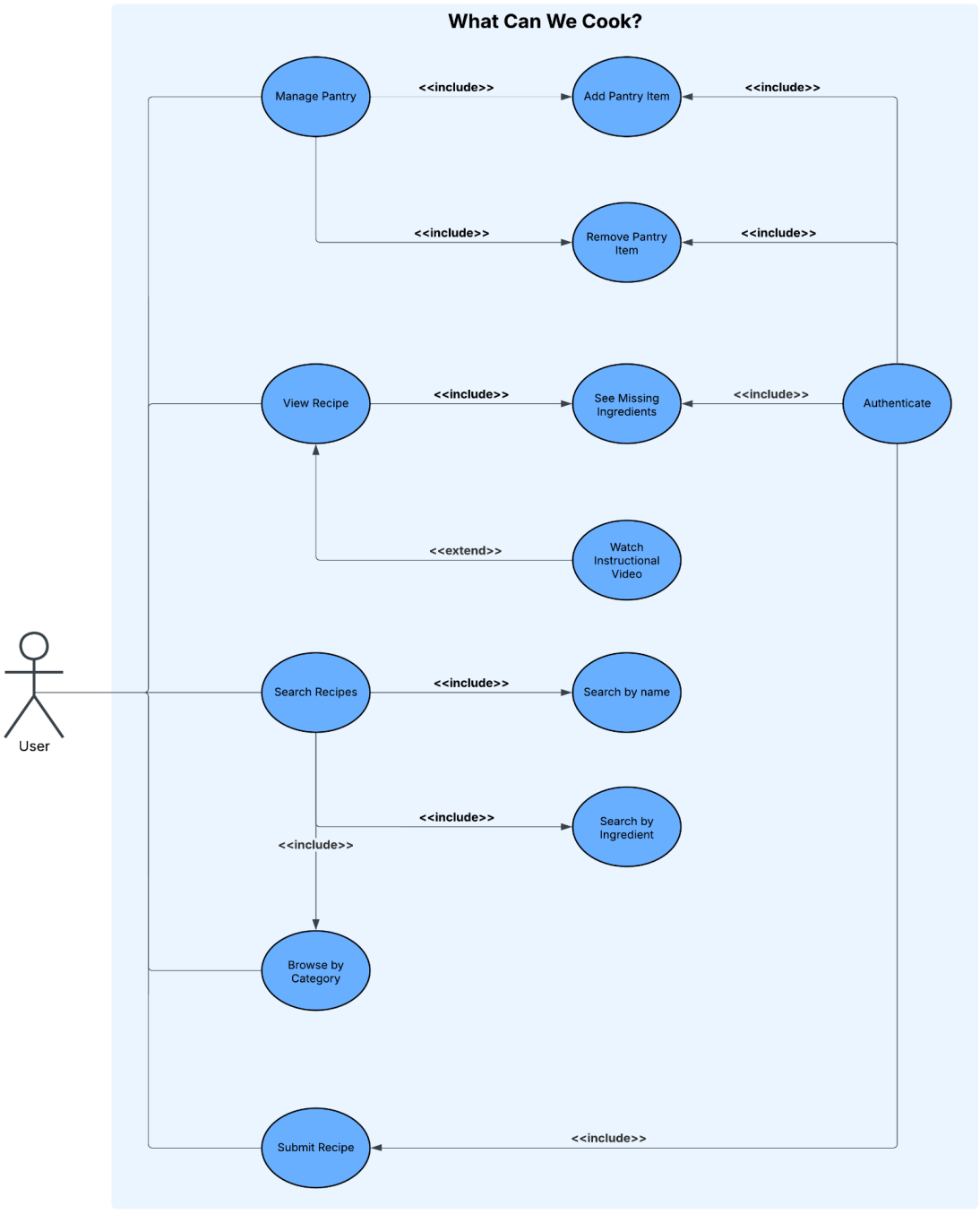
In this phase, a lot of the difficulty was less about code and more about general "admin" work: splitting tasks fairly, managing time, and making sure everyone actually delivered their part so the project came together on schedule. On the technical side, wiring the JavaScript frontend to the Python/FastAPI backend was one of the most frustrating parts, especially handling API calls, JSON responses, and keeping the UI in sync with the data. We also felt some of the usual pain of using Python on a larger project as more modules and files were added, although this was less intense than in the previous deliverable since there was less new backend work this time and more focus on tying the pieces together.

UML Diagrams

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






What Can We Cook?
Manage Pantry
<<include>>
Add Pantry Item
<<include>>
<<include>>
Remove Pantry Item
<<include>>
View Recipe
<<include>>
See Missing Ingredients
<<include>>
Authenticate
<<extend>>
Watch Instructional Video
User
Search Recipes
<<include>>
Search by name
<<include>>
Search by Ingredient
<<include>>
Browse by Category
Submit Recipe
<<include>>

Team Contributions

Member name	Roles / Contributions	Overall Contribution (%)
Ben Oz Elhadad	Frontend tweaks, UI help, contributed to demo slides	15%
Darell Isaac Sam	No contribution reported for this phase.	0%
David Lucero	Documentation, coordination, deliverable editing, test case write-up.	10%
Nicholas Smit	Lead backend developer. Implemented most API routes, models, and tests and fixed integration issues.	40%
Sebastian Lopez	Implemented recipe submission feature and related frontend/backend integration; prepared demo content.	20%
Stanley Lu	Added new recipes/seed data and contributed to demo slides.	15%