Team Deliverable 3 – Phase I Implementation

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) for *What-Can-We-Cook?* Phase I. It expands upon the previous deliverable by refining requirements and describing the subset of features implemented in this first development phase.

1.2 Scope

What-Can-We-Cook? is a web application that enables users to manage their pantry, browse and search for recipes, view cooking instructions, and generate grocery lists for missing ingredients. Phase I delivers these core capabilities using a FastAPI backend and static HTML/CSS/JS frontend. Future phases will introduce advanced features such as recipe submission, exporting grocery lists, and dietary filters. At this stage, three of the four planned core features are functional and testable in the Phase I prototype.

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

Layer	**Technology**	**Role**
Frontend	HTML, CSS, JavaScript, Bootstrap (icons for styling consistency)	Displays recipes, pantry, and UI logic

Layer	Technology	Role
Backend	Python + FastAPI	Handles API requests, authentication, business logic
Database	SQLite + SQLAlchemy	Persistent storage for users, ingredients, recipes
External	YouTube API	Embeds instructional videos 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.
- **Smart Grocery List:** Compares selected recipe requirements to the pantry contents and generates a list of missing items for shopping. (Not implemented yet)

3. Updated Requirements (Phase 1)

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 the recipe recommendation and grocery list features.

- 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.

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

Phase I implements all core system modules with authenticated pantry management and recipe browsing. Search and recipe detail features are fully operational, while grocery list comparison remains in progress for the next phase. Testing includes endpoint unit tests for authentication, ingredient retrieval, and CRUD operations.

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.
```

3. Test cases can be run by navigating to `src/api/tests` and running pytest.

6. User Manual

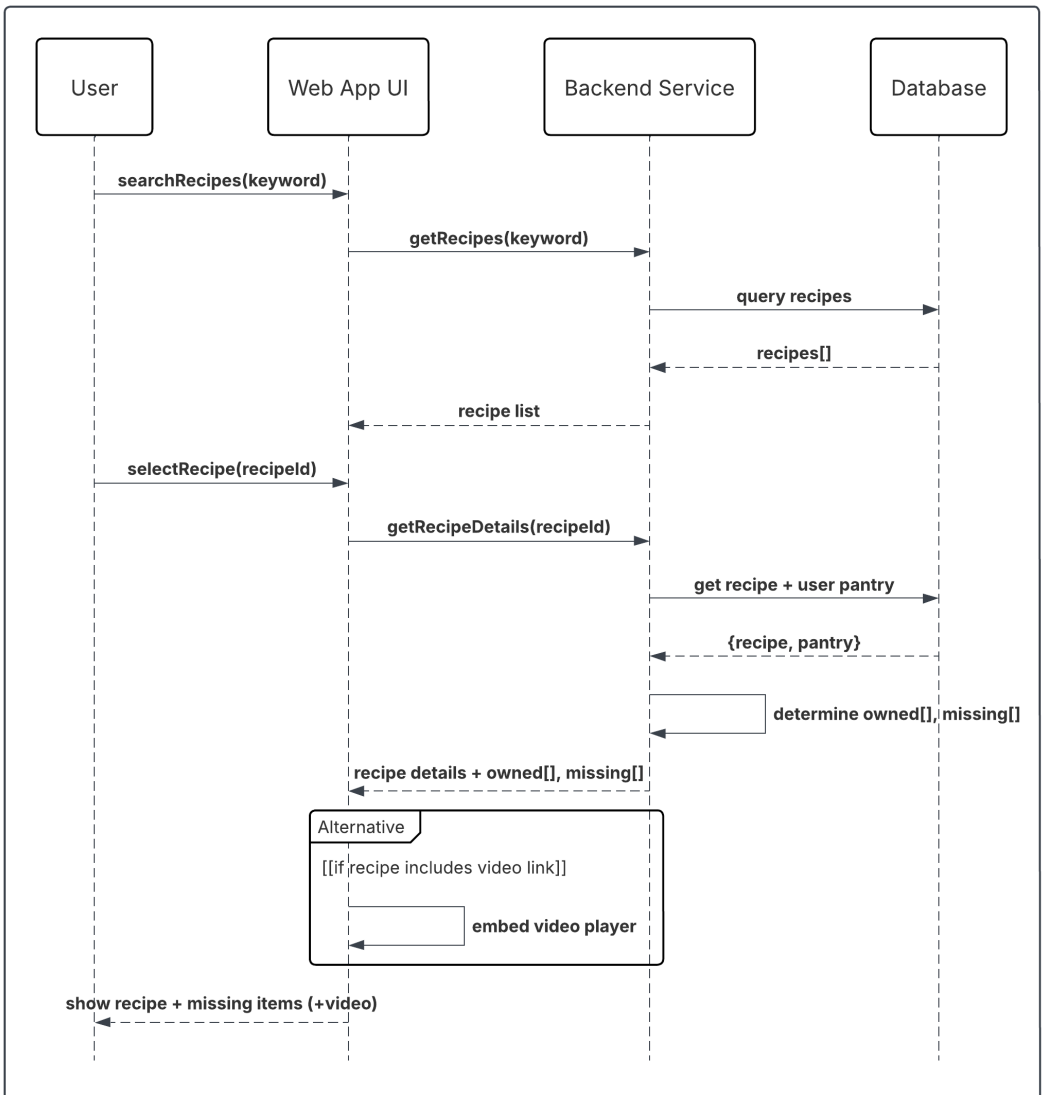
- **Home Page:** Displays recent recipes and search bar.
- **Search:** Enter a recipe name or ingredient → browse results.
- **Recipe Page:** View title, description, instructions, and optional video.
- **Login:** Enter demo credentials to enable pantry features.
- **Pantry:** Add or remove ingredients and compare against recipes.
- **Grocery List:** Displays missing ingredients after comparison.

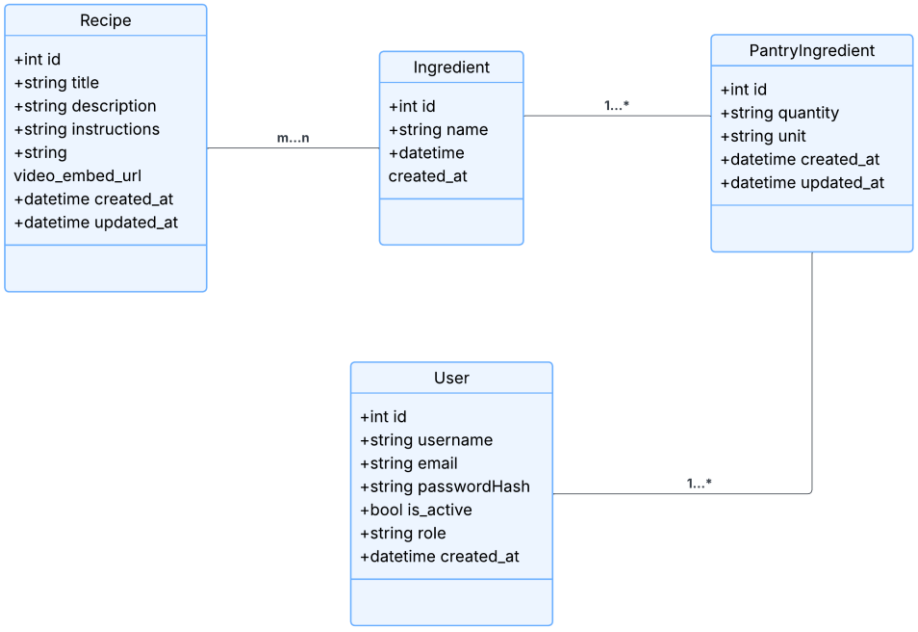
7. Reflection

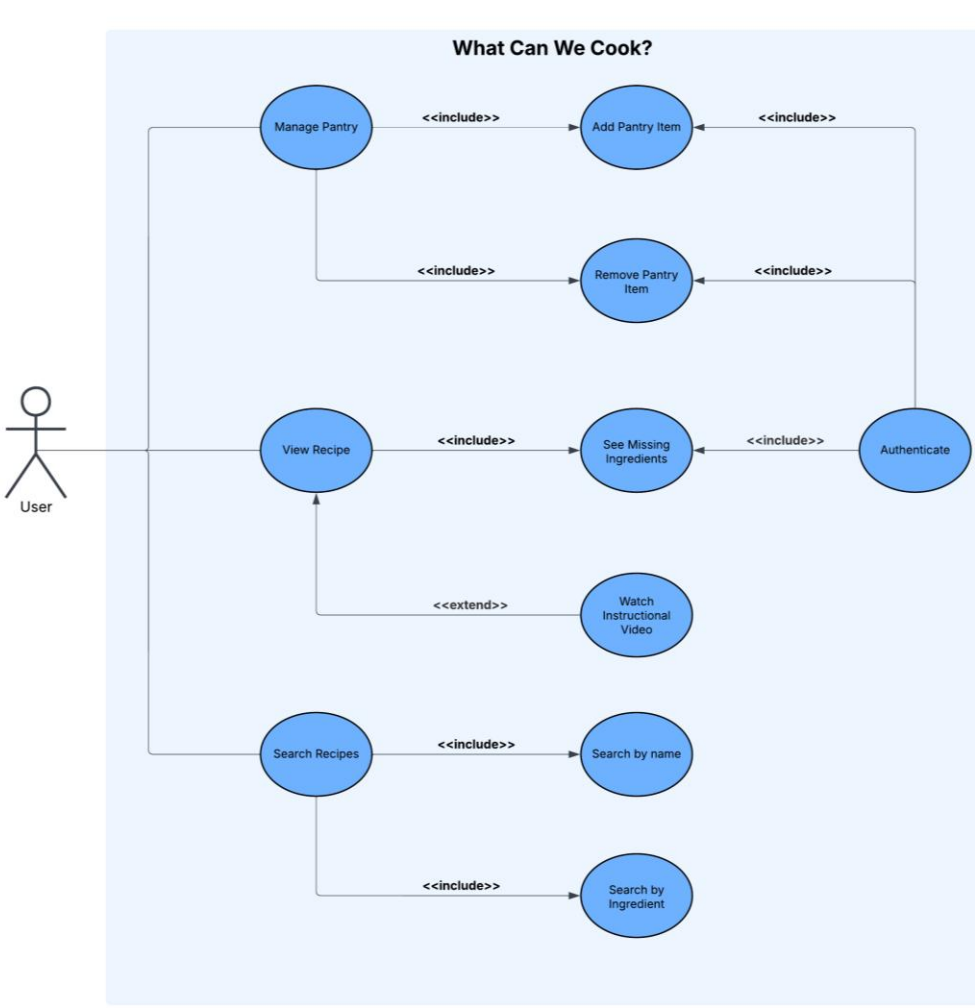
Phase I delivered a functional prototype demonstrating recipe search, viewing, and pantry management. Development was occasionally slowed by Python's asynchronous behavior and lack of strict typing, which complicated debugging and large-scale structure. These challenges highlighted both the flexibility and the limitations of using a lightweight language for a growing project. The team adapted effectively through consistent communication and integration of all major system components into a working implementation. While some members noted that a stricter language such as Java or C# might offer better maintainability, the project will continue using Python for the remaining phases.

UML Diagrams

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




Recipe
+int id
+string title
+string description
+string instructions
+string video_embed_url
+datetime created_at
+datetime updated_at
m...n
Ingredient
+int id
+string name
+datetime created_at
1...*
PantryIngredient
+int id
+string quantity
+string unit
+datetime created_at
+datetime updated_at
User
+int id
+string username
+string email
+string passwordHash
+bool is_active
+string role
+datetime created_at
1...*


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

Team Contributions

Member name	Roles / Contributions	Overall Contribution (%)
Ben Oz Elhadad	Designed and implemented HTML/CSS pages, handled site visuals,	21%
Darell Isaac Sam	Updated meeting minutes	5%
David Lucero	Documentation lead and coordination. Authored Deliverables and Diagrams.	10%
Nicholas Smit	Lead backend developer. Implemented majority of FastAPI routes, database integration, and frontend JavaScript hooks.	30%
Sebastian Lopez	Contributed to backend controllers and testing.	17%
Stanley Lu	Created and managed seed data.	17%