Software Requirements Specification

for

FlavorFind - A Recipe Finder & Personal Cookbook

Version 1.0

Prepared For:
Dr. Ahmed Antar

Prepared By:
Name: Mark William
Student ID: 120210348
Course: Data Engineering - CSE 428

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Mark William	24/4/2025	Initial Draft	1.0

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document describes the functional and nonfunctional requirements for the "FlavorFind" web application, Version 1.0. This application is being developed as the final project for the Data Engineering course (CSE 428). FlavorFind will allow users to search for recipes using an external API (TheMealDB) and manage (Create, Read, Update, Delete) their own personal recipes via a web interface connected to a MySQL database. The purpose of this document is to provide a detailed description of the system's intended behavior, features, constraints, and interfaces, serving as a guide for development and testing.

1.2 Document Conventions

- The key words "shall," "must," "should," "may," "required," "recommended," and "optional" in this document are to be interpreted as described in RFC 2119. Essentially, "shall" indicates a mandatory requirement.
- Functional requirements are uniquely identified with a prefix REQ- followed by a feature code (e.g., AUTH for Authentication) and a sequence number (e.g., REQ-AUTH-01).
- Placeholders for information yet to be determined are indicated by [TBD].
- References to external documents are indicated in Section 1.5.

1.3 Intended Audience and Reading Suggestions

This document is intended for the following audiences:

- **Developer (Mark):** To understand what needs to be built, including features, constraints, and interfaces.
- **Instructor (Dr. Ahmed Antar):** To evaluate the project's plan, scope, and adherence to requirements.
- **Testers (Implicitly, the Developer):** To define test cases based on the specified requirements.

Reading Suggestions:

- **Overview:** Start with Section 1 (Introduction) and Section 2 (Overall Description) for a high-level understanding of the project, its scope, users, and constraints.
- **Functional Details:** Refer to Section 3 (System Features) for detailed descriptions of what the system shall do.
- **Interfaces & Nonfunctional Needs:** Refer to Section 4 (External Interface Requirements) and Section 5 (Other Nonfunctional Requirements) for details on how the system interacts with external entities and its quality attributes (performance, security, etc.).
- **Definitions & Models:** Use Appendix A (Glossary) for definitions and Appendix B (Analysis Models) to visualize data structures (ERD) and potentially class structures.

1.4 Project Scope

FlavorFind is designed to be a web-based application accessible via standard web browsers. Its core functionalities include:

- **User Management:** Secure user registration and login.
- **External Recipe Discovery:** Searching recipes from TheMealDB API based on various criteria (e.g., name, ingredient) and viewing detailed recipe information.
- **Personal Cookbook Management:** Allowing authenticated users to perform CRUD operations (Create, Read, Update, Delete) on their own recipes stored in the application's database.

Relevant benefits and objectives:

- Provide a practical tool for users interested in finding and storing recipes.
- Demonstrate understanding and application of data engineering principles (database design, integration, backend/frontend development).
- Meet the requirements of the Data Engineering course final project, including potential bonus objectives like AJAX implementation and cloud deployment.

This SRS covers Version 1.0. Future versions might include features currently listed as out of scope (see Section 5 of the Project Proposal).

1.5 References

1. **Project Proposal: FlavorFind - A Recipe Finder & Personal Cookbook, Version 1.0**. Provides the initial overview, goals, and planned technology stack.
2. **TheMealDB API Documentation:** https://www.themealdb.com/api.php - Defines the interface for the external recipe data source.
3. **RFC 2119: Key words for use in RFCs to Indicate Requirement Levels.** (S. Bradner, March 1997) - Defines the interpretation of requirement keywords.
4. **Entity-Relationship Diagram (ERD) for FlavorFind, Version 1.0.** To be provided separately or embedded in Appendix B. Defines the database structure.
5. **Class Diagram for FlavorFind, Version 1.0.** May be provided separately or embedded in Appendix B. Illustrates key software classes.

2. Overall Description

2.1 Product Perspective

FlavorFind is a new, self-contained web application. It is not replacing an existing system nor is it a component of a larger software system beyond its interaction with the external TheMealDB API.

The system architecture comprises three main components:

1. **Web Frontend:** A browser-based user interface built with HTML, CSS, and Vanilla JavaScript, allowing users to interact with the system.

2. **Backend Application Server:** A Python/Flask application that handles user requests, implements business logic, interacts with the database via SQLAlchemy, communicates with the external TheMealDB API, and serves data (primarily as JSON) back to the frontend.
3. **Database:** A MySQL relational database storing user account information and user-created personal recipes.

The system interacts externally with:

- **Users:** Via the web frontend in a standard web browser.
- **TheMealDB API:** To fetch recipe data for the search functionality.

2.2 Product Features

FlavorFind will provide the following major features:

- **User Authentication:** Allows users to register for a new account and log in to access personalized features.
- **External Recipe Search:** Enables users (both anonymous and logged-in) to search for recipes from TheMealDB based on various criteria (e.g., name, ingredient) and view detailed results.
- **Personal Recipe Creation:** Allows authenticated users to add their own recipes to the system, including title, description, ingredients, and instructions.
- **Personal Recipe Viewing:** Enables authenticated users to view a list of their saved recipes and the details of any specific recipe they own.
- **Personal Recipe Modification:** Allows authenticated users to edit the details of recipes they have previously saved.
- **Personal Recipe Deletion:** Allows authenticated users to remove recipes from their personal collection.

Details of the functional requirements for each feature are provided in Section 3.

2.3 User Classes and Characteristics

There are two primary classes of users anticipated for FlavorFind:

1. **Anonymous User:**
    - **Characteristics:** Any individual accessing the application without logging in. Assumed to have basic web browsing skills.
    - **Allowed Functions:** Can browse publicly accessible parts of the site, primarily the external recipe search feature (UC-03) and viewing external recipe details (UC-04). Cannot manage personal recipes.
2. **Registered User:**
    - **Characteristics:** An individual who has created an account and logged in. Assumed to have basic web browsing skills and the ability to use web forms.
    - **Allowed Functions:** Can perform all functions available to Anonymous Users, plus manage their personal cookbook (Create, Read, Update, Delete personal recipes - UC-05, UC-06, UC-07, UC-08, UC-09).

The primary user class to satisfy is the Registered User, as their interactions involve the core database integration and CRUD functionality.

2.4 Operating Environment

- **Server-Side (Backend):**
  - The application backend (Flask) is expected to run in a standard server environment, typically Linux-based.
  - Requires Python 3.x runtime environment.
  - Requires access to a running MySQL database server (Version 5.7+ or 8.x recommended).
  - Requires installed Python libraries: Flask, SQLAlchemy, requests, Werkzeug, mysql-connector-python (or PyMySQL).
  - For production deployment, a WSGI server (like Gunicorn or Waitress) would typically be used in front of the Flask application.

- **Client-Side (Frontend):**
  - The application frontend shall be accessible and functional on modern desktop web browsers (e.g., latest versions of Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge) that support HTML5, CSS3, and JavaScript (ES6+). Responsiveness for mobile browsers is a desirable but secondary goal for V1.0.

- **External Dependencies:**
  - Requires reliable internet connectivity for the backend server to access TheMealDB API.

2.5 Design and Implementation Constraints

1. **Technology Stack:** The system shall be implemented using Python 3.x, the Flask web framework, MySQL database, SQLAlchemy ORM, and standard HTML/CSS/Vanilla JavaScript for the frontend.
2. **Database:** The primary data store shall be a MySQL relational database. The schema design (ERD) must support the required entities and relationships.
3. **External API:** The system shall utilize the public TheMealDB API (www.themealdb.com/api.php) for external recipe searching. Error handling must account for potential API unavailability or errors.
4. **Data Transfer Format:** Data exchanged between the Flask backend API endpoints and the JavaScript frontend shall primarily use the JSON format. This satisfies the project requirement for using XML or/and JSON.
5. **Security:** User passwords shall be securely hashed before storage using a recognized algorithm (e.g., via Werkzeug security helpers). Passwords shall never be stored in plain text. Input validation should be performed on data submitted via forms to mitigate common vulnerabilities. Parameterized queries (handled by SQLAlchemy) shall be used for database interactions to prevent SQL injection.
6. **Statelessness:** Backend API endpoints should be designed to be stateless where possible, relying on tokens or session management for user authentication state.
7. **Development Resources:** Development must be completed within the project timeline using available resources (primarily the single student developer).

2.6 User Documentation

The primary user documentation provided with Version 1.0 will be:

1. **README File:** A markdown file included with the source code detailing:
    - Project overview.
    - Technology stack used.
    - Step-by-step instructions for local setup (dependencies, database configuration, running the application).
    - Basic usage guide or description of features.
    - API endpoint descriptions (if applicable for clarity).
2. **Inline UI Guidance:** User interface elements (labels, placeholders, button text) will be designed to be intuitive. Error messages will guide the user in case of invalid input or system issues.

No separate formal user manual or online help system is planned for Version 1.0.

2.7 Assumptions and Dependencies

- **Assumptions:**
    - The TheMealDB API will remain publicly accessible, free to use, and maintain a stable interface throughout the project development and demonstration period.
    - Users possess basic computer literacy and familiarity with navigating and interacting with web applications (clicking links, filling forms, etc.).
    - The target web browsers correctly implement HTML5, CSS3, and JavaScript (ES6+) standards.
- **Dependencies:**
    - The backend application is dependent on a running Python 3.x interpreter.
    - The backend application is dependent on a properly configured and accessible MySQL database server.
    - The backend application depends on the availability and correct installation of required Python libraries (Flask, SQLAlchemy, requests, Werkzeug, MySQL driver).
    - The external recipe search functionality is dependent on network connectivity between the application server and TheMealDB API servers.
    - The frontend relies on the backend API being available and responsive.

3. System Features

This section details the functional requirements organized by major system features. Each feature corresponds roughly to one or more use cases identified in the proposal.

3.1 Feature: User Authentication

3.1.1 Description and Priority

Provides capabilities for user registration and login to secure access to personal recipe management features. Priority is high. Use Cases UC-01 and UC-02 are covered.

3.1.2 Stimulus/Response Sequences

- **Registration:** User navigates to registration page -> User enters username, email, password, confirms password -> User submits form -> System validates input -> System creates new user record (with hashed password) -> System redirects user to login page or dashboard with a success message.
- **Login:** User navigates to login page -> User enters username/email and password -> User submits form -> System validates credentials -> System establishes user session -> System redirects user to their dashboard/homepage.
- **Invalid Input/Credentials:** User submits form with invalid data (e.g., mismatched passwords, invalid email format, username taken) or incorrect login credentials -> System displays specific error message(s) on the form without clearing valid fields where appropriate.

3.1.3 Functional Requirements

- **REQ-AUTH-01:** The system **shall** provide a user registration interface where a user can enter a unique username, a valid email address, and a password.
- **REQ-AUTH-02:** The system **shall** require password confirmation during registration, and the two password fields must match.
- **REQ-AUTH-03:** The system **shall** validate that the chosen username is unique within the system during registration.
- **REQ-AUTH-04:** The system **shall** validate that the provided email address is unique within the system during registration.
- **REQ-AUTH-05:** The system **shall** validate that the provided email address follows a standard email format (e.g., user@domain.com).
- **REQ-AUTH-06:** The system **shall** securely hash user passwords using an appropriate algorithm (e.g., Werkzeug's implementation of PBKDF2 or bcrypt) before storing them in the database. Passwords **shall not** be stored in plain text.
- **REQ-AUTH-07:** Upon successful registration, the system **shall** store the user's details (username, email, hashed password) in the database.
- **REQ-AUTH-08:** The system **shall** provide a login interface where a registered user can enter their username (or email) and password.
- **REQ-AUTH-09:** The system **shall** authenticate the user by comparing the provided password (after hashing) against the stored hashed password associated with the username/email.
- **REQ-AUTH-10:** Upon successful authentication, the system **shall** establish a user session (e.g., using secure session cookies managed by Flask) to identify the user for subsequent requests.
- **REQ-AUTH-11:** The system **shall** provide clear error messages for failed registration or login attempts (e.g., "Username already taken," "Invalid username or password," "Passwords do not match") without revealing excessive information (e.g., distinguishing between non-existent username and incorrect password on login failure).
- **REQ-AUTH-12:** The system **shall** provide a mechanism for users to log out, which **shall** terminate their current session.

3.2 Feature: External Recipe Search & Viewing

3.2.1 Description and Priority

Allows users to search for recipes using TheMealDB API and view the details of the recipes found. Priority is High. Use Cases UC-03 and UC-04 are covered.

3.2.2 Stimulus/Response Sequences

- **Search:** User enters search term (e.g., ingredient, recipe name) into search bar -> User submits search -> Frontend (potentially via AJAX) sends request to backend -> Backend queries TheMealDB API -> Backend receives results -> Backend processes results and sends back to frontend -> Frontend dynamically displays list/grid of matching recipes (name, image thumbnail).
- **View Details:** User clicks on a recipe from the search results -> Frontend sends request to backend (possibly with recipe ID from API) OR directly queries API if ID is known -> Backend retrieves detailed recipe data from TheMealDB API -> Backend processes data and sends back to frontend -> Frontend displays detailed view (title, image, ingredients, instructions, etc.).
- **No Results:** User search yields no results from TheMealDB API -> System displays a "No recipes found" message.
- **API Error:** TheMealDB API request fails (timeout, error response) -> System displays a user-friendly error message (e.g., "Could not connect to recipe service").

3.2.3 Functional Requirements

- **REQ-SRCH-01:** The system **shall** provide an interface (e.g., a search bar) for users to enter search terms (keywords, ingredients, recipe names) to find external recipes.
- **REQ-SRCH-02:** The system **shall** query TheMealDB API based on the user's search input.
- **REQ-SRCH-03:** The system **shall** display a list of recipes matching the search criteria, including at least the recipe name and a thumbnail image, as returned by TheMealDB API.
- **REQ-SRCH-04:** If the search yields no results, the system **shall** display a clear message indicating that no matching recipes were found.
- **REQ-SRCH-05:** The system **shall** allow the user to select a recipe from the search results to view its details.
- **REQ-SRCH-06:** When a recipe is selected, the system **shall** retrieve and display detailed information for that recipe from TheMealDB API, including (where available):
  - Recipe Name/Title
  - Recipe Image
  - List of Ingredients and Measurements
  - Preparation Instructions
  - Category/Area (e.g., Cuisine type)
  - Link to original source (if provided by API)
- **REQ-SRCH-07:** The system **shall** handle potential errors during communication with TheMealDB API gracefully and display an appropriate message to the user.
- **REQ-SRCH-08 (Bonus - AJAX):** It is **recommended** that the recipe search results be loaded and displayed dynamically using AJAX calls, preventing a full page reload after submitting a search query.

3.3 Feature: Personal Recipe Creation

3.3.1 Description and Priority

Allows authenticated users to add their own recipes to their personal cookbook within the application. Priority is high**.** Use Case UC-05 is covered.

3.3.2 Stimulus/Response Sequences

- **Navigate to Create Form:** Authenticated user clicks "Add Recipe" link/button -> System displays the create recipe form.
- **Submit New Recipe:** User fills in recipe details (title, description, ingredients, instructions, optional image URL) -> User submits form -> System validates input -> System saves the new recipe to the database, associating it with the logged-in user -> System redirects user to their recipe list or the newly created recipe's detail page with a success message.
- **Invalid Input:** User submits form with missing required fields or invalid data -> System displays specific error message(s) on the form, retaining valid data entered by the user.
- **Unauthorized Access Attempt:** Unauthenticated user attempts to access the create recipe form/endpoint -> System redirects user to the login page.

3.3.3 Functional Requirements

- **REQ-PCR-01:** The system **shall** provide an interface (e.g., a web form) for authenticated users to create a new personal recipe.
- **REQ-PCR-02:** Access to the recipe creation interface **shall** be restricted to authenticated (logged-in) users.
- **REQ-PCR-03:** The recipe creation form **shall** allow the user to input at least the following details:
    - Recipe Title (Mandatory)
    - Description (Optional)
    - Ingredients (Mandatory, likely as a multi-line text area)
    - Instructions (Mandatory, likely as a multi-line text area)
    - Image URL (Optional, validated as a URL format if provided)
- **REQ-PCR-04:** The system **shall** validate that mandatory fields (Title, Ingredients, Instructions) are filled before saving the recipe.
- **REQ-PCR-05:** Upon successful submission and validation, the system **shall** save the new recipe details into the MySQL database.
- **REQ-PCR-06:** Each saved personal recipe **shall** be associated with the user ID of the authenticated user who created it.
- **REQ-PCR-07:** The system **shall** provide confirmation feedback (e.g., a success message) to the user upon successful creation of a recipe.
- **REQ-PCR-08:** The system **shall** provide clear error messages if recipe creation fails due to validation errors or other issues.

3.4 Feature: Personal Recipe Viewing

3.4.1 Description and Priority

Allows authenticated users to view the recipes they have personally created and saved. Priority is high. Use Cases UC-06 and UC-07 are covered.

3.4.2 Stimulus/Response Sequences

- **View Recipe List:** Authenticated user navigates to their "My Recipes" or "Cookbook" page -> System retrieves all recipes associated with that user ID from the database -> System displays a list of the user's recipes (e.g., titles, maybe thumbnails/descriptions).
- **View Recipe Details:** User clicks on a recipe title/link from their list -> System retrieves the full details of the selected recipe from the database -> System displays the detailed view of the personal recipe (title, description, ingredients, instructions, image if available).
- **Empty Cookbook:** Authenticated user navigates to "My Recipes" page but has not created any recipes -> System displays a message indicating that no recipes have been saved yet (e.g., "Your cookbook is empty. Add a recipe!").
- **Unauthorized Access Attempt:** User attempts to view the "My Recipes" list or a specific personal recipe detail page belonging to another user -> System shall deny access (e.g., show a "Not Found" or "Unauthorized" error, or redirect to login/dashboard).

3.4.3 Functional Requirements

- **REQ-PVR-01:** The system **shall** provide an interface for authenticated users to view a list of all personal recipes they have created.
- **REQ-PVR-02:** The displayed list **shall** only contain recipes created by the currently logged-in user.
- **REQ-PVR-03:** The recipe list **shall** display at least the title of each recipe, and each title should be a link to the detailed view of that recipe.
- **REQ-PVR-04:** If a user has no saved recipes, the system **shall** display an informative message instead of an empty list.
- **REQ-PVR-05:** The system **shall** allow the authenticated user to select a recipe from their list to view its full details.
- **REQ-PVR-06:** When viewing the details of a personal recipe, the system **shall** display all stored information for that recipe (Title, Description, Ingredients, Instructions, Image URL if present).
- **REQ-PVR-07:** Access to view the detailed information of a specific personal recipe **shall** be restricted to the user who created that recipe.

3.5 Feature: Personal Recipe Modification

3.5.1 Description and Priority

Allows authenticated users to edit the details of the recipes they have personally created. Priority is High. Use Case UC-08 is covered.

3.5.2 Stimulus/Response Sequences

- **Navigate to Edit Form:** Authenticated user views the details of their personal recipe -> User clicks an "Edit" link/button -> System retrieves the current recipe data -> System displays the edit recipe form, pre-filled with the existing recipe details.
- **Submit Updates:** User modifies recipe details in the form -> User submits the form -> System validates input -> System updates the corresponding recipe record in the database -> System redirects user back to the recipe's detail page or their recipe list with a success message.
- **Invalid Input:** User submits form with invalid data (e.g., clears a mandatory field) -> System displays specific error message(s) on the form, retaining other valid data.
- **Unauthorized Access Attempt:** User attempts to access the edit form/endpoint for a recipe they do not own -> System denies access (e.g., "Not Found" or "Unauthorized" error).
- **Cancel Edit:** User navigates to the edit form -> User clicks a "Cancel" button -> System discards changes and redirects the user back to the recipe detail view or recipe list.

3.5.3 Functional Requirements

- **REQ-PUP-01:** The system **shall** provide an interface for authenticated users to modify their existing personal recipes.
- **REQ-PUP-02:** Access to the recipe modification interface (edit form) for a specific recipe **shall** be restricted to the user who created that recipe.
- **REQ-PUP-03:** The edit form **shall** be pre-populated with the current saved details of the recipe being edited (Title, Description, Ingredients, Instructions, Image URL).
- **REQ-PUP-04:** The edit form **shall** allow the user to change all editable fields associated with the recipe (Title, Description, Ingredients, Instructions, Image URL).
- **REQ-PUP-05:** The system **shall** validate the modified data according to the same rules as recipe creation (e.g., mandatory fields cannot be emptied).
- **REQ-PUP-06:** Upon successful submission and validation, the system **shall** update the corresponding recipe record in the MySQL database with the new details.
- **REQ-PUP-07:** The system **shall** provide confirmation feedback (e.g., a success message) to the user upon successful update of a recipe.
- **REQ-PUP-08:** The system **shall** provide clear error messages if recipe update fails due to validation errors or other issues.
- **REQ-PUP-09:** The system **should** provide a way to cancel the editing process without saving changes.

3.6 Feature: Personal Recipe Deletion

3.6.1 Description and Priority

Allows authenticated users to permanently remove recipes they have created from their personal cookbook. Priority is medium (Slightly lower than C/R/U as accidental deletion is undesirable). Use Case UC-09 is covered.

3.6.2 Stimulus/Response Sequences

- **Initiate Deletion:** Authenticated user views their recipe list or a specific recipe detail page -> User clicks a "Delete" link/button associated with a recipe.
- **Confirm Deletion:** System prompts the user to confirm the deletion action (e.g., via a JavaScript confirmation dialog or a separate confirmation page).
- **Deletion Confirmed:** User confirms they want to delete -> System removes the corresponding recipe record from the database -> System redirects the user to their recipe list with a success message.
- **Deletion Cancelled:** User cancels the deletion action when prompted -> System takes no action on the recipe record -> User remains on the current page or is returned to the previous view.
- **Unauthorized Access Attempt:** User attempts to trigger the delete action/endpoint for a recipe they do not own -> System denies the action.

3.6.3 Functional Requirements

- **REQ-PDL-01:** The system **shall** provide a mechanism for authenticated users to delete their own personal recipes.
- **REQ-PDL-02:** The ability to initiate deletion for a specific recipe **shall** be restricted to the user who created that recipe.
- **REQ-PDL-03:** The system **shall** require user confirmation before permanently deleting a recipe (e.g., using a modal dialog: "Are you sure you want to delete '[Recipe Title]'? This action cannot be undone.").
- **REQ-PDL-04:** If the user confirms deletion, the system **shall** permanently remove the corresponding recipe record from the MySQL database.
- **REQ-PDL-05:** If the user cancels the deletion, the system **shall not** modify the recipe record.
- **REQ-PDL-06:** Upon successful deletion, the system **shall** provide confirmation feedback (e.g., a success message) to the user.

4. External Interface Requirements

4.1 User Interfaces

- **UI-01:** The application shall provide a web-based graphical user interface (GUI) accessible via standard desktop web browsers.
- **UI-02:** The user interface shall be built using HTML5, CSS3, and Vanilla JavaScript.
- **UI-03:** The interface design should be clean, intuitive, and provide clear navigation between major sections (e.g., Home/Search, My Recipes, Login, Register).
- **UI-04:** Key interface elements will include:
    - Navigation bar/menu for site-wide navigation.
    - Search input fields and buttons for finding external recipes.
    - Forms for user registration, login, and creating/editing personal recipes, utilizing standard HTML form controls (text inputs, password inputs, text areas, buttons).
    - Display areas for presenting lists of recipes (search results, personal recipes) and detailed recipe views.
    - Action buttons/links for operations like "Save", "Edit", "Delete", "View Details", "Login", "Logout", "Register".
- **UI-05:** The interface shall provide visual feedback for user actions (e.g., indicating successful save/update/delete, displaying loading indicators during AJAX requests).
- **UI-06:** Error messages resulting from user input validation or system errors shall be displayed clearly and contextually, ideally close to the input field or action that caused the error.
- **UI-07:** Consistency in layout, terminology, and visual style shall be maintained across different pages of the application.

4.2 Hardware Interfaces

No specific hardware interfaces are required for the FlavorFind application beyond standard computer hardware capable of running a modern web browser (for clients) and standard server hardware capable of running the backend application and MySQL database (for hosting). The application does not directly interact with any specialized hardware components.

4.3 Software Interfaces

- **SI-01: TheMealDB API**
    - **Interface:** The application backend shall interface with the public TheMealDB REST API via HTTP GET requests.
    - **Data Source:** Provides external recipe data for search and viewing.
    - **Key Endpoints (Planned):**
        - https://www.themealdb.com/api/json/v1/1/search.php?s={name} (Search by name)
        - https://www.themealdb.com/api/json/v1/1/filter.php?i={ingredient} (Search by ingredient)
        - https://www.themealdb.com/api/json/v1/1/lookup.php?i={id} (Lookup full details by ID)
        - (Other endpoints may be used if needed)
    - **Data Format:** The application expects to receive data from TheMealDB API in JSON format.

    - **Dependency:** The external recipe search functionality is dependent on the availability and responsiveness of this API. Error handling shall be implemented (Ref REQ-SRCH-07).
- **SI-02: MySQL Database**
    - **Interface:** The Flask backend application shall interface with the MySQL database server.
    - **Interaction:** Communication will be managed via the SQLAlchemy ORM library and a suitable Python MySQL driver (e.g., mysql-connector-python or PyMySQL).
    - **Purpose:** Persistent storage of user accounts and personal recipes.
    - **Configuration:** Database connection details (host, port, database name, username, password) shall be configurable via environment variables or a configuration file and not hardcoded in the source code.
- **SI-03: Web Browser**
    - **Interface:** The application frontend shall run within standard web browsers.
    - **Compatibility:** Requires browsers supporting HTML5, CSS3, and JavaScript (ECMAScript 6 or later). Target browsers include recent versions of Chrome, Firefox, Safari, and Edge.
    - **Functionality:** The browser renders the UI and executes client-side JavaScript for interactivity and communication with the backend API.
- **SI-04: Operating System**
    - **Server:** The backend application and database are expected to run on a standard server OS, preferably a Linux distribution (e.g., Ubuntu, CentOS).
    - **Client:** Users will access the application via browsers running on common desktop operating systems (Windows, macOS, Linux).

4.4 Communications Interfaces

- **CI-01:** Client-Server Communication (Browser <-> Backend)
    - **Protocol:** Communication between the user's web browser and the Flask backend server shall use the Hypertext Transfer Protocol Secure (HTTPS) for all interactions involving sensitive data (login, registration, personal recipe management) and should use HTTPS for all interactions if possible. Standard HTTP may be used for non-sensitive public content during local development only.
    - **API Style:** The backend shall expose API endpoints following RESTful principles where applicable (using appropriate HTTP verbs like GET, POST, PUT, DELETE for resource manipulation).
    - **Data Format:** Data exchanged between the backend API and the frontend JavaScript **shall** primarily use the **JSON (JavaScript Object Notation)** format. This fulfills the project requirement of using "XML or/and JSON".
- **CI-02: Server-External API Communication (Backend <-> TheMealDB)**
    - **Protocol:** Communication between the Flask backend server and TheMealDB API shall use HTTP GET requests over the public internet.
    - **Data Format:** Expects JSON responses from TheMealDB API.
- **CI-03: Server-Database Communication (Backend <-> MySQL)**
    - **Protocol:** Communication between the Flask backend server and the MySQL database server shall use the standard MySQL network protocol.
    - **Security:** Connection should be secured appropriately, especially in production (e.g., using SSL/TLS if configured, firewall rules, strong passwords). Connection details must be kept confidential.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- **PERF-01:** The web application pages (e.g., login, registration, recipe lists, recipe details) **should** load in the user's browser within 3 seconds under typical network conditions (e.g., broadband internet) and assuming nominal server load.
- **PERF-02:** Responses to user actions involving backend interaction (e.g., submitting login, saving a recipe, initiating a search) **should** generally be acknowledged by the UI (e.g., showing results, success/error message, loading indicator stops) within 2-4 seconds, excluding the network latency to the external TheMealDB API.
- **PERF-03:** Database queries retrieving user-specific recipe lists or details **shall** be optimized (e.g., using appropriate indexing on foreign keys like userId) to ensure responsiveness, especially as the number of users and recipes grows (though large scale is not expected for V1.0).
- **PERF-04:** The external recipe search via TheMealDB API is dependent on the external service's performance; however, the application frontend **should** provide visual feedback (e.g., a loading indicator) while waiting for the API response.

5.2 Safety Requirements

- **SAFE-01:** There are no specific safety requirements associated with the use of the FlavorFind application, as it does not control physical hardware or deal with safety-critical information or processes.

5.3 Security Requirements

- **SEC-01:** User authentication **shall** be required for all actions related to creating, updating, or deleting personal recipes (Ref REQ-PCR-02, REQ-PUP-02, REQ-PDL-02).
- **SEC-02:** The system **shall** prevent users from viewing, editing, or deleting personal recipes belonging to other users (Ref REQ-PVR-07, REQ-PUP-02, REQ-PDL-02). Authorization checks must be implemented on the backend for all relevant API endpoints.
- **SEC-03:** User passwords **shall** be securely hashed using an industry-recognized algorithm with a salt (e.g., via Werkzeug security helpers) before storage in the database (Ref REQ-AUTH-06).
- **SEC-04:** All communication involving sensitive data (passwords during login/registration, session identifiers) **shall** be transmitted over HTTPS in a production/deployment environment.
- **SEC-05:** The system **shall** implement input validation on all user-submitted data (forms, API requests) on the backend to prevent common web vulnerabilities such as Cross-Site Scripting (XSS) (e.g., by properly escaping output) and potentially SQL Injection (though largely mitigated by using SQLAlchemy ORM with parameterized queries).
- **SEC-06:** Session management **shall** be implemented securely (e.g., using Flask's secure session management with a strong secret key) to prevent session hijacking. Session cookies should be configured with appropriate flags (e.g., HttpOnly, Secure in production).
- **SEC-07:** Database credentials and application secret keys **shall not** be hardcoded in the source code repository and should be managed via environment variables or secure configuration files.

- **SEC-08:** The application **should** implement basic protection against Cross-Site Request Forgery (CSRF) for state-changing requests (e.g., saving/deleting recipes), for example, using Flask-WTF or a similar mechanism if forms are heavily used.

5.4 Software Quality Attributes

- **USAB-01 (Usability):** The user interface **shall** be intuitive and easy to navigate for users familiar with basic web applications. Key functions like search, add recipe, and view recipes should be easily discoverable.
- **MAIN-01 (Maintainability):** Backend code (Python/Flask) **should** be organized logically (e.g., using Blueprints for different application sections), well-commented where necessary, and follow standard Python style conventions (PEP 8) to facilitate understanding and future modifications.
- **MAIN-02 (Maintainability):** Frontend code (HTML/CSS/JS) **should** be well-structured and readable. CSS class names should be meaningful. JavaScript should be organized into functions or modules where appropriate.
- **REL-01 (Reliability):** The application **should** handle common errors (e.g., database connection issues, external API unavailability, invalid user input) gracefully without crashing, providing informative messages to the user where appropriate (Ref REQ-SRCH-07, REQ-AUTH-11, REQ-PCR-08, REQ-PUP-08).
- **PORT-01 (Portability):** The application **should** be reasonably portable to different hosting environments that support the chosen technology stack (Python/Flask, MySQL). Dependencies should be clearly listed (e.g., in a requirements.txt file).

6. Other Requirements

- **REQ-OTH-01 (Licensing):** All third-party libraries used (e.g., Flask, SQLAlchemy, Requests, MySQL driver) shall be compatible with the project's usage and distribution (if any). Primarily, open-source licenses (like MIT, BSD, Apache 2.0) are expected. Compliance with the terms of TheMealDB API usage is also required.
- **REQ-OTH-02 (Database Schema):** The design of the MySQL database schema, represented in the Entity-Relationship Diagram (ERD), is considered a core requirement deliverable for Phase 1. (See Appendix B or separate ERD document, Ref 1.5).
- **REQ-OTH-03 (Deployment - Bonus):** If pursued for bonus marks, the application **shall** be deployable to a public cloud platform (e.g., Render, PythonAnywhere) following standard deployment practices for Flask applications. Setup instructions for deployment should be included in the final README.
- **REQ-OTH-04 (AJAX - Bonus):** If pursued for bonus marks, the external recipe search functionality **shall** utilize AJAX for dynamic data loading (Ref REQ-SRCH-08).
- **REQ-OTH-05 (Real-world Data - Bonus):** The integration with TheMealDB API fulfills the requirement/bonus objective of using real-world data.

Appendix A: Glossary

Term	Definition
AJAX	Asynchronous JavaScript and XML. Technique for updating parts of a web page without a full page reload.
API	Application Programming Interface. A set of definitions and protocols for building and integrating software.
Backend	The server-side part of the application (Flask, Database) that handles logic and data management.
CRUD	Create, Read, Update, Delete. The four basic functions of persistent storage.
CSRF	Cross-Site Request Forgery. A type of web security vulnerability.
CSS	Cascading Style Sheets. Language used for describing the presentation of a web page.
Database	An organized collection of structured information, or data, typically stored electronically (MySQL here).
DOM	Document Object Model. A programming interface for HTML and XML documents. Used by JS to manipulate pages.
ERD	Entity-Relationship Diagram. A graphical representation of entities and their relationships in a database.
Flask	A micro web framework written in Python.
Frontend	The client-side part of the application (HTML, CSS, JS) that runs in the user's web browser.
GUI	Graphical User Interface.
HTML	HyperText Markup Language. The standard markup language for documents designed to be displayed in a browser.
HTTP	Hypertext Transfer Protocol. The foundation of data communication for the World Wide Web.
HTTPS	Hypertext Transfer Protocol Secure. Encrypted version of HTTP using SSL/TLS.
JSON	JavaScript Object Notation. A lightweight data-interchange format easy for humans/machines to read/write.
JavaScript (JS)	A programming language commonly used to create interactive effects within web browsers.
MySQL	An open-source relational database management system (RDBMS).
ORM	Object-Relational Mapping. Technique for converting data between incompatible type systems (e.g., Python objects and database tables).

RDBMS	Relational Database Management System.
REST	Representational State Transfer. An architectural style for designing networked applications (APIs).
SQL	Structured Query Language. Language used for managing and querying data in relational databases.
SQLAlchemy	A Python SQL toolkit and Object Relational Mapper.
SQLite	A C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine (Not used in final plan, but common term).
SRS	Software Requirements Specification. This document.
TheMealDB	The specific external online recipe database and API used by FlavorFind.
UI	User Interface.
Use Case (UC)	A description of sequences of actions that a system performs that yield an observable result of value to a particular actor.
Vanilla JS	Standard JavaScript without the use of any additional libraries or frameworks.
XSS	Cross-Site Scripting. A type of web security vulnerability involving injection of malicious scripts.
XML	Extensible Markup Language. A markup language that defines a set of rules for encoding documents.

Appendix B: Analysis Models

The primary analysis models developed during Phase 1 for the FlavorFind project are the Entity-Relationship Diagram (ERD) and the Enhanced Entity-Relationship Diagram (EERD).
Due to their graphical nature, these diagrams are provided as separate deliverables:

- **Entity-Relationship Diagram (ERD):** Please refer to the submitted file [FlavorFind_ERD.png]
- **Enhanced Entity-Relationship Diagram (EERD):** Please refer to the submitted file [FlavorFind_EERD.png]

Appendix C: Issues List

Issue ID	*Description*	*Status*	*Priority*	*Notes*
ISS-01	Specific pagination strategy for recipe lists (TBD)	Open	Low	How to handle large numbers of recipes?
ISS-02	Detailed error handling for all TheMealDB API calls	In Progress	Medium	Need to cover various failure modes.