

Software Requirements Specification (SRS)

1. Introduction:

What Can I Cook? is a web application that helps users discover meal ideas based on user-provided ingredients. It features AI-generated recipe suggestions. Alongside recipe suggestions, the app provides an estimated calorie breakdown using a built-in calorie lookup system, and a clean interface. The app aims to promote healthier eating, reduce food waste, and simplify meal planning through a fast and user-friendly experience

1.2 Document Conventions:

- Programming language: Python 3.x (Flask framework)
- Database: SQLite3
- API: Spoonacular API
- Templates follow Flask's Jinja2 syntax (`{{ }}`, `{% %}`)
- "User" refers to the end user of the system, "System" refers to the Smart Recipes app

1.3 Intended Audience and Reading Suggestions:

1. Developers: Focus on sections 2 and 3 for implementation details.
2. Testers: Review sections 3, 4, and 5 to validate functionality and performance.
3. Project Managers: Read sections 1 and 2 for project scope and constraints.
4. End Users/Clients: Refer to section 2 for high-level features overview.

1.4 Project Scope:

Smart Recipes helps users discover and manage recipes in an interactive way. Its core objectives are:

- Provide personalized recipe search using keywords.
- Allow user registration and secure login with email verification.
- Enable saving favorite recipes for later use.
- Provide detailed nutritional breakdown (calories, macros).
- Offer a simple, user-friendly web interface accessible from browsers.
-

1.5 References:









1. [Spoonacular API](#) for recipe data
2. [Flask](#) web framework
3. [Bootstrap](#) for responsive design
4. SQLite Documentation: <https://www.sqlite.org/docs.html>
5. Templates follow Flask's **Jinja2** syntax (`{{ }}`, `{% %}`)
6. "User" refers to the end user of the system, "System" refers to the Smart Recipes app

2. Overall Description:

2.1 Product Perspective

The platform is a standalone web application developed using (2.4 *Operating Environment*) It serves as a web application that helps users discover meal ideas based on user-provided ingredients.

2.2 Product Features:

-  Recipe Discovery: Find recipes based on available ingredients using Spoonacular API
-  User Authentication: Secure registration with email verification and login system
-  Favorites System: Save and manage favorite recipes
-  Nutritional Info: View calories, protein, and carbohydrates for recipes
-  Dietary Filters: Support for vegetarian, vegan, gluten-free, and dairy-free options
-  Security Features: Rate limiting, CSRF protection, and secure sessions
-  Search History: Track recent ingredient searches
-  Responsive Design: Mobile-friendly interface

2.3 User Classes and Characteristics:

- **User:**(soon to be chefs)

2.4 Operating Environment

- Backend: Flask (Python)
- Database: SQLite with proper migrations
- Frontend: HTML, CSS, JavaScript with Bootstrap
- Email: Flask-Mail with SMTP integration
- API: Spoonacular Recipe API
- Caching: Flask-Caching for improved performance
- Security: Rate limiting, password hashing, session management

2.5 Design and Implementation Constraints:

- Pending registrations stored in memory (use Redis for production)
- Limited error recovery for API failures
- No automated testing suite
- Must use Spoonacular API (API key required).

2.6 User Documentation:

- [README.md](#)
- User guide (basic login, search, save favorites).
- API documentation (internal routes).
- Setup guide for developers.

2.7 Assumptions and Dependencies

- Users have access to a stable internet connection.
- Users must have a valid email for verification.
Requires internet access for API calls.
- Spoonacular API availability and limitations may affect features.

3. System Features

3.1 User Registration & Login

- Description: Users can create accounts and log in securely. Email verification ensures account validity.
Priority: High
- Stimulus/Response:
 - User registers → System sends verification email → User confirms → Account activated.
- Functional Requirements:
 - FR1: The system must validate unique emails.
 - FR2: The system must hash and store passwords securely.
 - FR3: The system must allow login/logout with sessions.

3.2 Recipe Search

- Description: Users can search recipes using keywords.
- Priority: High
- Stimulus/Response:
 - User enters keyword → System fetches recipe list from Spoonacular → Displays recipes with title, image, and link.
- Functional Requirements:
 - FR4: The system must integrate with Spoonacular API.
 - FR5: The system must display top search results with images and titles.

3.3 Recipe Details & Nutrition

- Description: View full recipe details with calorie and macro breakdown.
- Priority: Medium
- Functional Requirements:
 - FR6: The system must fetch ingredients, instructions, and nutrition details for selected recipes.

3.4 Favorites System

- Description: Users can save recipes for future reference.
 - Priority: Medium
 - Functional Requirements:
 - FR7: The system must allow saving/deleting favorites.
 - FR8: The system must store favorites in the database.
 - FR9: The system must display saved recipes for each user.
-

4. External Interface Requirements

4.1 User Interfaces

- Web-based interface with search bar, recipe cards, and user dashboard.
- Responsive design (desktop-first, mobile compatible).

4.2 Hardware Interfaces

- No special hardware requirements beyond a standard PC or server.

4.3 Software Interfaces

- Flask (backend) ↔ Spoonacular API
- Flask ↔ SQLite3 (database operations)

4.4 Communications Interfaces

- SMTP (for email verification).
-

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- Response time for search results ≤ 3 seconds (depends on API).
- Support up to 50 concurrent users in free hosting environment.

5.2 Safety Requirements

- Ensure no sensitive data (passwords) is stored in plaintext.

5.3 Security Requirements

- Passwords must be hashed.
- Sessions must expire on logout.
- Email verification required for new accounts.

5.4 Software Quality Attributes

- Maintainability: Modular Flask routes and functions.
 - Usability: Simple, clean UI.
 - Portability: Can run locally or in cloud.
-

6. Other Requirements

- Future expansion may include meal planning, grocery list generation, and user-uploaded recipes.
-

Appendices

A. Glossary

- API: Application Programming Interface
- SQLite: Lightweight database engine

B. Analysis Models

- [Future placeholder: Use case diagrams, sequence diagrams, ER diagram]

C. Issues List

- Free Spoonacular API key may have request limits.