Shelf Smart

Software Requirements Specifications

Version 1.1

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| <26/2/24> | <1.0> | Initial ideas, brainstorming requirements, and definitions. | Ansuman Sharma, Yadhunath Tharakeswaran, Shyaan Mohamad, Kundhana Pragnya. |
| <24/03/24> | <1.1> | Finalizing ideas, Declaring functions, Started with backend, initial codings. | Ansuman Sharma, Yadhunath Tharakeswaran, Shyaan Mohamad, Kundhana Pragnya. |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Overall Description 5

2.1 Product perspective 5

2.1.1 System Interfaces 5

2.1.2 User Interfaces 5

2.1.3 Hardware Interfaces 5

2.1.4 Software Interfaces 5

2.1.5 Communication Interfaces 5

2.1.6 Memory Constraints 5

2.1.7 Operations 5

2.2 Product functions 5

2.3 User characteristics 5

2.4 Constraints 5

2.5 Assumptions and dependencies 5

2.6 Requirements subsets 5

3. Specific Requirements 5

3.1 Functionality 5

3.1.1 <Functional Requirement One> 6

3.2 Use-Case Specifications 6

3.3 Supplementary Requirements 6

4. Classification of Functional Requirements 6

5. Appendices [6](#_heading=h.qsh70q)

Software Requirements Specifications

# Introduction

The Software Requirements Specification (SRS) serves as the foundational document outlining the intricate details necessary for the development of Shelf-Smart. In response to the evolving landscape of pantry management and recipe recommendations, this SRS encapsulates a comprehensive set of requirements essential for the successful implementation of the web application. With a focus on enhancing user experience and efficiency, this document delineates not only the functional aspects but also the nuanced non-functional requirements, design constraints, and other pivotal factors critical for a holistic understanding of the software's demands. By providing a structured framework, the SRS aims to align the development process with user expectations and project objectives, ensuring the creation of a robust and intuitive platform poised to revolutionize culinary resource management.

## Purpose

The purpose of this SRS is to delineate the complete set of requirements for the Shelf-Smart web application. It aims to define the external behavior of the application, encompassing functional and non-functional requirements, design constraints, and other factors crucial for a thorough understanding of the software requirements.

## Scope

This SRS applies to the Shelf-Smart web application, focusing on features related to pantry management, recipe recommendations, and user interaction. It is associated with the use-case model depicting various scenarios of user interaction with the system. Additionally, this document influences the development process, ensuring alignment with user expectations and project objectives.

## Definitions, Acronyms, and Abbreviations

## Ingredient: A commodity added to the user's pantry.

## Spice (child class of Ingredient): Resembles an ingredient but doesn't require an expiry date.

## Recipe: Consists of the dish name and a brief description.

## User: Any individual utilizing the Shelf-Smart application for pantry management.

## References

* IEEE 830-1998: Provides guidelines and standards for Software Requirements Specifications.
* Project Glossary: Contains definitions of terms, acronyms, and abbreviations relevant to the project.

## Overview

The SRS comprises a package containing detailed use cases, supplementary specifications, and supporting information. It delineates the external behavior of the Shelf-Smart application, addressing functional requirements, non-functional requirements, and design constraints. The organization of the document facilitates a comprehensive understanding of the software requirements, aiding in the development process and ensuring adherence to project objectives

# Overall Description

*[This section of the* ***SRS*** *describes the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3, and makes them easier to understand. Include such items as:*

## Product perspective

### **System Interfaces**:

### An interface would be created for the user to enter the ingredients in their pantry, their food preferences, and the quantity.

### **User Interfaces:** It would be the same as the system interface.

### **Software Interfaces:** This website is compatible on all Windows, Mac, and Linux devices.

### **Communication Interfaces**:

### We will be using HTTP for the client-server communication, a RESTful API would be built to maintain a seamless interaction between the front-end and back-end, and JSON and XML would be the major medium for the data interaction between the front-end and back-end.

### **Memory Constraints**:

### The recipes would be produced only based on the ingredients and recipes available on the database. Therefore, the user would be able to access the recipes based on the current database. At the same time, this website also contains a caching mechanism to reduce the data overload.

### **Operations**: Some of the key operations are:

### Adding and removing items from the pantry.

### Classifying each ingredient as carbs, protein, fruits, vegetables.

### Guide the user to input their data on the right spot.

### Produce warnings during any technical errors, or exceptions.

### Produce relevant recipes based on the given ingredients to the user.

### And allowing the user to save the recipes for future use.

## Product functions

Lets users keep track of items in their pantry and also recommend dishes to them based on their pantry items.

## User characteristics

User is someone who wishes to keep track of their pantry items and utilize them efficiently.

## Constraints

Must be implemented mainly using C++ and JavaScript (for API utilization); should use SQL to store and update pantry items.

## Assumptions and dependencies

* User enters a text input.
* Collaboration between teammates to create a functional application with enhanced features and robust user interface.
* Ensure that every deliverable is finished within deadlines.

## Requirements subsets

* User input: Specify the type of ingredient the user wants to add and then accordingly decide if an expiration date is needed.
* Recommendation logic: There needs to be a minimum number of items in the pantry before a recommendation can be given to the user.
* Execution: The tool serves as a book-keeping place for the user to keep track of their groceries and at the same time be given creative ideas on using their groceries for taste delights. It is mostly a static tool apart from the recommendation features.

# Specific Requirements

## Functionality

**Input Processing**

* Shelf-Smart will take user input for food products, amounts, and expiry dates, but excluding spices.
* Users will be able to define the order of operations by entering ingredients separated by parenthesis.
* The system will evaluate user input to ensure that it is in a valid format for processing.
* In the event of invalid input, Shelf-Smart will display appropriate error messages directing the user to fix the input.

**Classification of ingredients**

* Shelf-Smart will classify the input ingredients into categories such as meat, grain, spice, and more.
* The technology will effortlessly manage the cupboard inventory depending on ingredient classification.

**Output Generation**

* Shelf-Smart will create 3-5 dish recommendations based on the ingredients entered by the user.
* Each dish recommendation will be accompanied by a brief 1-2 line description that entices users with the food's characteristics.
* Shelf-Smart can optionally provide calorie statistics for each recommended dish.

**User Interface**

* Shelf-Smart will have a user-friendly interface for both input and output, accessible via the command line.
* The interface will present a curated selection of foods, along with descriptions and optional calorie information.
* Users will be given the opportunity to start new operations or depart the Shelf-Smart system easily.

**Error Handling**

* Shelf-Smart will handle errors smoothly throwing exceptions during situations of incorrect input or unsupported operations.
* Appropriate error messages will be shown to help users correct problems discovered during input or processing.

## Use-Case Specifications

* Input food items, quantities, and expiration dates: Shelf-Smart prompts the user to enter food items(except spices), amounts, and expiration dates.
* Recognize categories of ingredients: organize input items into categories such as meat, grain, spice, and more.
* Build a curated selection of dishes: generates a menu of 3-5 recipes based on the items entered by the user.
* Provide dish descriptions: offers 1-2 line tantalizing descriptions for each meal.
* Optional Calorie Information: optionally displays calorie information for each dish in the curated menu.
* Showcase Pantry Inventory: provides a complete representation of the user's pantry inventory.
* Use random pantry items for recommendations: Users can also choose random items from the existing pantry to generate personalized cooking recommendations.

## Supplementary Requirements

**Programming Language and Data Structure**

* Shelf-Smart must be implemented with C++.
* The system will use a data structure(ex: Stack) to efficiently process input expressions.

**Version Control and Collaboration**

* The source code for Shelf-Smart will be hosted on GitHub.
* The project repository will be shared and accessible via GitHub for collaboration and version control purposes.

**Object-Oriented Programming Principles.**

* Shelf-Smart will use object-oriented programming concepts for code structuring and design.
* Classes, objects, inheritance, and polymorphism need to be used well to improve code maintainability and extensibility.

**Documentation and Comments**

* The code will contain detailed comments and documentation to explain the application's logic and capabilities.
* Comments have to be included in the source code to clarify the meaning of classes, functions, and algorithms.

**Unit Testing**

* Unit tests will be created for verifying the validity and reliability of Shelf-Smart's features.
* Test cases have to include a variety of circumstances, including valid input expressions, invalid input expressions, and edge cases.

**Error Handling**

* Shelf-Smart will display clear and helpful error messages to users in the event of invalid input or mistakes during expression evaluation.
* Error messages should assist users in discovering and correcting input errors, hence improving the user experience and usability of the program.

# Classification of Functional Requirements

| **Functionality** | **Type** |
| --- | --- |
| Lets user add a commodity to their pantry | Essential |
| Lets user remove a commodity from their pantry | Essential |
| Lets user view the items currently in the pantry | Essential |
| Classifies commodities into fruits, vegetables, grains, or spices | Essential |
| Lets user get three recommended dish options from their pantry items | Essential |
| Save a certain recipe for future use. | Essential |
| Display calorie count and serving count for each recommendation | Optional |
| Displays pictures of some common ingredients after they’re added to pantry | Optional |

# Appendices

At the current stage of the project, appendices are not applicable. For further information regarding the scope and definition of the project, please refer to document 001 - Project Plan. Any additional details pertinent to the project's requirements, scope, or definitions will be incorporated into subsequent revisions of the Software Requirements Specification as deemed necessary during the development process.