**Software Requirements Specification (SRS) Document**

**Library Management System**

**1. Introduction**

**1.1 Purpose**

This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements for the **Library Management System**. This system is designed to help users manage the books in a library by allowing them to add, edit, delete, search, and view book details. The system will also allow data persistence through CSV file storage.

**1.2 Scope**

The system will include functionalities for:

* Managing book details (add, edit, delete)
* Searching books by title or author
* Storing the library data in a CSV file
* User interface with search, add, and edit book features

**1.3 Definitions, Acronyms, and Abbreviations**

* **Book**: A collection of data, including a title, author, and the number of copies.
* **LibraryApp**: The core application that manages book-related tasks.
* **CSV**: Comma Separated Values - the format used for storing data.

**2. Overall Description**

**2.1 Product Perspective**

The system is a desktop application with a graphical user interface (GUI) built using **Tkinter** in Python. It will interact with a CSV file for storing book data, allowing for persistent data management. The system will be primarily used by a librarian (admin).

**2.2 Product Functions**

* **User Registration and Login**: Users can register and log in.
* **Book Management**: Users can add, edit, delete, and search for books.
* **CSV File Storage**: The system will read from and write to a CSV file to store library data.
* **Search Functionality**: Users can search for books by title or author.

**2.3 User Classes and Characteristics**

* **End Users**: They can add, delete, and search for books.
* **Admin Users**: They have the same functionalities as end users with additional rights to manage the library data.

**2.4 Operating Environment**

* The system will run on desktop operating systems (Windows, macOS, Linux).
* The application will use the Python language with the **Tkinter** library for the GUI.

**3. Specific Requirements**

**3.1 Functional Requirements**

1. **Add Book**:
   * Users should be able to add a book by entering its title, author, and number of copies.
   * The system will validate input and show an error if any field is empty or invalid.
2. **Edit Book**:
   * Users can select an existing book and edit its title, author, or number of copies.
   * The updated details will be reflected in the CSV file.
3. **Delete Book**:
   * Users can select a book to delete. The system will prompt for confirmation before deletion.
4. **Search Book**:
   * Users can search for books by title or author.
   * The search results will be displayed in the book table.
5. **CSV File Storage**:
   * The library data will be saved in a Library.csv file. The system will automatically load and save data when needed.

**3.2 Non-Functional Requirements**

* **Performance**: The system should be responsive and capable of handling at least 100 books without significant performance degradation.
* **Security**: The application should ensure that user data (if any) is securely stored and handled.
* **Usability**: The system will have a simple, user-friendly interface suitable for librarians.

**4. Use Cases**

**4.1 Use Case 1: Add Book**

* **Actors**: Admin, User
* **Pre-condition**: User is logged into the system.
* **Post-condition**: The book is added to the library, and the system is updated.
* **Flow of Events**:
  1. The user clicks on "Add Book".
  2. A popup window appears asking for the book’s title, author, and copies.
  3. The user fills out the form and clicks "Add".
  4. The book is added to the list and saved to the CSV file.

**4.2 Use Case 2: Edit Book**

* **Actors**: Admin
* **Pre-condition**: A book exists in the system.
* **Post-condition**: The book’s details are updated.
* **Flow of Events**:
  1. The user selects a book to edit.
  2. A popup window displays the current details of the book.
  3. The user updates the information and clicks "Update".
  4. The book’s details are updated in the list and saved to the CSV file.

**4.3 Use Case 3: Delete Book**

* **Actors**: Admin
* **Pre-condition**: A book exists in the system.
* **Post-condition**: The book is deleted from the system and CSV file.
* **Flow of Events**:
  1. The user selects a book to delete.
  2. The system asks for confirmation before deletion.
  3. If confirmed, the book is removed from the list and CSV file.

**4.4 Use Case 4: Search Book**

* **Actors**: User
* **Pre-condition**: The library has books stored.
* **Post-condition**: Search results are displayed based on user input.
* **Flow of Events**:
  1. The user enters a search term.
  2. The system filters books by title or author.
  3. The system displays the search results.

**5. Assumptions and Dependencies**

* The system assumes that all users have basic knowledge of operating desktop applications.
* The system relies on Python and Tkinter for GUI development.
* The system is dependent on the existence and availability of the Library.csv file for storing book data.

**6. External Interface Requirements**

**6.1 User Interface**

* The system will use a graphical user interface with buttons, entry fields, and a treeview table.

**6.2 Hardware Interfaces**

* The system will require a device capable of running a Python program (e.g., a computer with an operating system that supports Python).

**6.3 Software Interfaces**

* The system uses the **Tkinter** library for the GUI and **CSV** for data storage.

**7. System Architecture**

The architecture is based on a simple model where the user interacts with the GUI, which in turn manipulates data through the **LibraryApp** class and stores the data in the CSV file.

A diagram of a graph

Description automatically generated