**Software Requirements Specification (SRS) for Personal Task Manager**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the requirements for a Personal Task Manager application. The application will assist users in organizing, managing, and tracking their personal tasks effectively. It provides features such as task creation, categorization, prioritization, and user authentication while ensuring a simple, user-friendly interface.

**1.2 Scope**

The Personal Task Manager will:

* Allow users to manage tasks through operations like create, edit, delete, and search.
* Enable task categorization, prioritization, and deadline setting.
* Provide user authentication for secure access.
* Use Streamlit for the front-end and SQLite3 for data storage.

The application is designed for individual users and is not intended for multi-user or team task management scenarios.

**1.3 Definitions, Acronyms, and Abbreviations**

* **CRUD**: Create, Read, Update, Delete
* **SQLite3**: A lightweight, file-based relational database management system
* **Streamlit**: A Python-based web application framework for creating interactive applications
* **Task Status**: Indicates whether a task is complete or incomplete

**1.4 References**

* [Streamlit Documentation](https://docs.streamlit.io/)
* [SQLite Documentation](https://www.sqlite.org/docs.html)

**2. Functional Requirements**

**2.1 Task Management**

* Users can create, view, edit, and delete tasks.
* Each task includes the following fields: title, description, category, priority, deadline, and status.

**2.2 Categorization and Prioritization**

* Tasks can be categorized into user-defined categories (e.g., Work, Personal, Health).
* Tasks can be assigned priority levels: High, Medium, or Low.

**2.3 Status Tracking**

* Users can mark tasks as complete or incomplete.
* Status updates are displayed on the main dashboard.

**2.4 Search and Filter**

* Users can search tasks by title or description.
* Tasks can be filtered by category, priority, status, or due date.

**2.5 User Authentication**

* Secure login functionality using hashed passwords.
* Each user can manage their tasks independently.

**2.6 Data Backup and Export**

* Users can export their tasks to a CSV file for backup and offline use.

**3. Non-Functional Requirements**

**3.1 Performance**

* The application must respond to user actions (e.g., adding or editing a task) within 2 seconds.

**3.2 Usability**

* The interface must be intuitive, with minimal learning required.
* Provide meaningful error messages and user feedback.

**3.3 Scalability**

* The application should handle up to 10,000 tasks without performance degradation.

**3.4 Security**

* Passwords must be hashed using a secure algorithm (e.g., bcrypt).
* Prevent unauthorized access to user data.

**3.5 Portability**

* The application must run on any system with Python installed.

**3.6 Maintainability**

* The codebase should follow modular design principles for easy updates and feature additions.

**4. Use Cases**

**4.1 Use Case: Create a Task**

* **Actors**: User
* **Description**: Users can add tasks by providing details such as title, description, deadline, priority, and category.
* **Preconditions**: User is logged in.
* **Flow**:
  1. User clicks the "Add Task" button.
  2. A form appears for entering task details.
  3. Task is saved in the database upon submission.

**4.2 Use Case: Edit a Task**

* **Actors**: User
* **Description**: Users can update task details such as title, description, deadline, priority, or category.
* **Preconditions**: User is logged in.
* **Flow**:
  1. User selects a task to edit.
  2. The task details appear in an editable form.
  3. Updates are saved to the database upon submission.

**4.3 Use Case: Search Tasks**

* **Actors**: User
* **Description**: Users can search for tasks using keywords in the title or description.
* **Preconditions**: User is logged in.
* **Flow**:
  1. User enters a keyword in the search bar.
  2. The system displays matching tasks.

**4.4 Use Case: Export Tasks**

* **Actors**: User
* **Description**: Users can export their task list to a CSV file for backup.
* **Preconditions**: User is logged in.
* **Flow**:
  1. User clicks the "Export" button.
  2. The system generates a CSV file and provides a download link.

**5. User Stories**

* **As a user**, I want to create tasks with deadlines **so that** I can organize my work effectively.
* **As a user**, I want to categorize tasks **so that** I can manage different aspects of my life separately.
* **As a user**, I want to mark tasks as complete **so that** I can track my progress.
* **As a user**, I want to search and filter tasks **so that** I can quickly find specific tasks.

**6. System Architecture**

**6.1 Overview**

The application consists of the following components:

* **Frontend**: Streamlit for user interface
* **Backend**: Python scripts handling business logic
* **Database**: SQLite3 for storing task and user data

**6.2 Data Flow**

1. User interacts with the Streamlit UI.
2. Data is sent to the backend for processing.
3. Processed data is stored/retrieved from the SQLite database.
4. Results are displayed on the UI.

**7. Wireframes or Mockups**

* **Dashboard**: Displays tasks in a categorized list with search and filter options.
* **Task Form**: Used for creating or editing tasks.
* **Login Page**: User authentication interface.

**8. Data Requirements**

**8.1 Database Schema**

* **Users Table**:
  + id (Primary Key)
  + username (Unique)
  + hashed\_password
* **Tasks Table**:
  + id (Primary Key)
  + title
  + description
  + category
  + priority
  + due\_date
  + status (Complete/Incomplete)
  + user\_id (Foreign Key referencing Users)

**9. Risks and Mitigations**

**9.1 Risks**

* **Risk**: SQLite limitations for large-scale data.
  + **Mitigation**: Design schema for easy migration to PostgreSQL if needed.
* **Risk**: Unauthorized access to tasks.
  + **Mitigation**: Implement secure login and validate user ownership of tasks.

**10. Implementation Roadmap**

**Phase 1: Core Features**

* Task CRUD operations
* User authentication

**Phase 2: Advanced Features**

* Search, filter, and export functionality

**Phase 3: Optimization**

* Performance improvements
* UI enhancements

**11. Acceptance Criteria**

* Users can create, edit, delete, and search tasks.
* Each user’s tasks are private and accessible only after login.
* The system responds within 2 seconds for all user actions.

**12. Appendix**

* **Tools and Technologies**: Python, Streamlit, SQLite3
* **References**: Streamlit and SQLite documentation
* **Glossary**: Definitions of technical terms used in the document