Software Requirements Specification (SRS) for User Account CRUD Service

**Developer**

M. Shafraz Jiffry

**Date & Time**

17/08/2024 03.00 PM

# **Table of Contents**

[**1** **Introduction** 4](#_Toc174808714)

[**1.1** **Purpose** 4](#_Toc174808715)

[**1.2** **Scope** 4](#_Toc174808716)

[**1.3** **Definitions, Acronyms, and Abbreviations** 4](#_Toc174808717)

[**1.4** **References** 4](#_Toc174808718)

[**1.5** **Overview** 4](#_Toc174808719)

[**2** **Overall Description** 5](#_Toc174808720)

[**1.2** **Product Perspective** 5](#_Toc174808721)

[**1.3** **Product Functions** 5](#_Toc174808722)

[**1.4** **User Characteristics** 5](#_Toc174808723)

[**1.5** **Constraints** 5](#_Toc174808724)

[**1.6** **Assumptions and Dependencies** 5](#_Toc174808725)

[**3** **System Features** 6](#_Toc174808726)

[**3.1** **Create User** 6](#_Toc174808727)

[**3.1.1** **Description and Priority** 6](#_Toc174808728)

[**3.1.2** **Functional Requirements** 6](#_Toc174808729)

[**3.1.3** **Validations** 6](#_Toc174808730)

[**3.2** **Read Users** 6](#_Toc174808731)

[**3.2.1** **Description and Priority** 6](#_Toc174808732)

[**3.2.2** **Functional Requirements** 6](#_Toc174808733)

[**3.2.3** **Validations** 7](#_Toc174808734)

[**3.3** **Update User** 7](#_Toc174808735)

[**3.3.1** **Description and Priority** 7](#_Toc174808736)

[**3.3.2** **Functional Requirements** 7](#_Toc174808737)

[**3.3.3** **Validations** 7](#_Toc174808738)

[**3.4** **Delete User** 7](#_Toc174808739)

[**3.4.1** **Description and Priority** 7](#_Toc174808740)

[**3.4.2** **Functional Requirements** 8](#_Toc174808741)

[**3.4.3** **Validations** 8](#_Toc174808742)

[**4** **External Interface Requirements** 8](#_Toc174808743)

[**4.1** **User Interfaces** 8](#_Toc174808744)

[**4.2** **Hardware Interfaces** 8](#_Toc174808745)

[**4.3** **Software Interfaces** 8](#_Toc174808746)

[**4.4** **Communications Interfaces** 8](#_Toc174808747)

[**5** **System Architecture** 8](#_Toc174808748)

[**5.1** **Overview** 8](#_Toc174808749)

[**6** **System Design** 9](#_Toc174808750)

[**6.1** **Database Design** 9](#_Toc174808751)

[**6.2** **API Endpoints** 9](#_Toc174808752)

[**6.3** **Validations Implementation** 9](#_Toc174808753)

[**7** **Performance Requirements** 9](#_Toc174808754)

[**8** **Design Constraints** 9](#_Toc174808755)

[**9** **Security Requirements** 10](#_Toc174808756)

[**10** **Other Non-functional Requirements** 10](#_Toc174808757)

[**10.1** **Reliability** 10](#_Toc174808758)

[**10.2** **Scalability** 10](#_Toc174808759)

[**10.3** **Maintainability** 10](#_Toc174808760)

[**11** **Validation and Testing** 10](#_Toc174808761)

[**11.1** **Testing Criteria** 10](#_Toc174808762)

[**12** **Appendices** 10](#_Toc174808763)

[**12.1** **Glossary** 10](#_Toc174808764)

# **Introduction**

## **Purpose**

This document aims to present a comprehensive Software Requirements Specification (SRS) for the User Account CRUD (Create, Read, Update, Delete) Service. This service, built using Python's Flask framework and SQLite database, will handle user accounts by performing crucial actions such as creating, retrieving, updating, and deleting user records.

## **Scope**

The CRUD Service will handle user account management for an application. It will support:

* Creation of new user accounts.
* Retrieval of user accounts (all users or specific user).
* Updating existing user accounts.
* Deleting user accounts.

## **Definitions, Acronyms, and Abbreviations**

* **CRUD:** Create, Read, Update, Delete
* **API:** Application Programming Interface
* **SQLite:** A C library that provides a lightweight, disk-based database
* **Flask:** A micro web framework written in Python
* **JSON:** JavaScript Object Notation, a lightweight data-interchange format

## **References**

* Flask Documentation: https://flask.palletsprojects.com/
* SQLite Documentation: https://sqlite.org/docs.html
* JSON Documentation: https://www.json.org/json-en.html

## **Overview**

The SRS document provides a comprehensive overview of the functional and non-functional requirements, system architecture, interface requirements, and validation procedures for the User Account CRUD Service. Additionally, it outlines the functionalities that the service will facilitate and the necessary verifications for each functionality.

# **Overall Description**

## **Product Perspective**

The CRUD Service is an independent module that can be incorporated into a bigger system to manage user accounts. The application will be developed using the Flask framework and will utilize SQLite as its database management system.

## **Product Functions**

* **Create User**: Add a new user to the system.
* **Read Users**: Retrieve a list of all users or details of a specific user.
* **Update User**: Modify the details of an existing user.
* **Delete User**: Remove a user from the system.

## **User Characteristics**

The primary users will be administrators or automated systems needing user account management. Users should understand HTTP requests and JSON format.

## **Constraints**

* The service is built with Flask and SQLite.
* The service should adhere to RESTful API standards.
* It must handle up to 100,000 user records efficiently.

## **Assumptions and Dependencies**

* A working Python environment with Flask installed.
* The SQLite database is used for storage.
* The service will be hosted on a server compatible with Flask applications

# **System Features**

## **Create User**

### **Description and Priority**

This feature allows for creating new user accounts. It is high-priority as it is fundamental to user account management.

### **Functional Requirements**

* **Input:** A JSON object with username, password, and active.
* **Process:**
  + Validate input data.
  + Check for duplicate usernames.
  + Insert the user into the database if validations pass.
* **Output:** Return a confirmation message and the user ID of the newly created user.

### **Validations**

* **Username:**
  + Must be provided.
  + Length should be between 3 and 20 characters.
  + Must be unique in the database.
* **Password:**
  + Must be provided.
  + Length should be at least 8 characters.
  + Must include at least one letter, one number, and one special character.
* **Active:**
  + Must be provided.
  + Must be either 0 (inactive) or 1 (active).

## **Read Users**

### **Description and Priority**

This feature allows retrieving user data. It is high-priority for viewing user information.

### **Functional Requirements**

* **Input:** Optional id parameter.
* **Process:**
  + Retrieve user data from the database.
* **Output:** Return user data in JSON format.

### **Validations**

* **ID (for single user retrieval):**
  + If provided, must be a positive integer.

## **Update User**

### **Description and Priority**

This feature allows updating existing user accounts. It is medium-priority.

### **Functional Requirements**

* **Input:** User id and a JSON object with fields to update (‘username’, ‘password’, ‘active’).
* **Process:**
  + Validate input data.
  + Check for user existence.
  + Update user record in the database if validations pass.
* **Output:** Return a confirmation message.

### **Validations**

* **ID:**
  + Must be a positive integer.
* **User Existence:**
  + Check if user with the provided id exists.
* **Required Fields:**
  + At least one of username, password, or active must be provided.
* **Username (if updated):**
  + Must be between 3 and 20 characters.
  + Must be unique (except for the current user).
* **Password (if updated):**
  + Must be at least 8 characters long.
  + Must include at least one letter, one number, and one special character.
* **Active (if updated):**
  + Must be either 0 (inactive) or 1 (active).

## **Delete User**

### **Description and Priority**

This feature allows deleting user accounts. It is medium-priority.

### **Functional Requirements**

* **Input:** User ‘id’.
* **Process:**
  + Validate ‘id’.
  + Check if user exists.
  + Delete the user record from the database if validations pass.
* **Output:** Return a confirmation message.

### **Validations**

* **ID:**
  + Must be a positive integer.
* **User Existence:**
  + Check if user with the provided ‘id’ exists before deletion.

# **External Interface Requirements**

## **User Interfaces**

There are no direct user interfaces; the service will be accessed via API endpoints.

## **Hardware Interfaces**

No specific hardware interfaces are required.

## **Software Interfaces**

* The service will use Flask for the web framework.
* It will interact with an SQLite database.

## **Communications Interfaces**

* The service will communicate using HTTP/REST.
* Data will be transferred in JSON format.

# **System Architecture**

## **Overview**

* The CRUD Service will adhere to a client-server design in which the server manages API requests and interfaces with the SQLite database.

# **System Design**

## **Database Design**

* **Table**: ‘users’
  + **‘id’** (INTEGER, Primary Key, Auto-increment)
  + **‘username’** (TEXT, Not Null, Unique)
  + **‘Password’** (TEXT, Not Null)
  + **‘Active’** (INTEGER, Not Null, 0 for inactive, 1 for active)

## **API Endpoints**

* **POST /users:** Create a new user.
* **GET /users:** Retrieve all users.
* **GET /users/<id>:** Retrieve a specific user by id.
* **PUT /users/<id>:** Update a user by id.
* **DELETE /users/<id>:** Delete a user by id.

## **Validations Implementation**

* **Create User:** Validate **‘username’**, **‘password’**, and **‘active’** fields; check for duplicate usernames.
* **Read User:** Validate **‘id’** if specified.
* **Update User:** Validate **’id’**, check for user existence, and validate username, password, and active if provided.
* **Delete User:** Validate **‘id’** and check for user existence.

# **Performance Requirements**

* The system should respond to API requests within 1 second under normal load.
* The system should handle up to 100,000 user records efficiently.

# **Design Constraints**

* The service uses Flask and SQLite.
* It should be deployable on a standard Python environment.

# **Security Requirements**

* **Data Validation:** Ensure input data is validated to prevent SQL injection and other security risks.
* **Password Storage:** Passwords should be securely hashed.
* **Authentication/Authorization:** Further enhancements may include authentication and authorization.

# **Other Non-functional Requirements**

## **Reliability**

* The system should have 99.9% uptime.

## **Scalability**

* The system should support horizontal scaling if needed.

## **Maintainability**

* The code should follow best practices for maintainability and readability.

# **Validation and Testing**

## **Testing Criteria**

* Unit tests for each API endpoint.
* Integration tests for the entire system.
* Stress tests to evaluate performance under high load.

# **Appendices**

## **Glossary**

* **Flask:** A micro web framework for Python.
* **SQLite:** A lightweight, disk-based database.