Software Design Specification (SDS)

Project Name: WriteShelf

Prepared By:

Seif Eldin 202200973,

Ahmed Mostafa 202201114

Draft 3

Date: 11/9/2024

---

1. Introduction

1.1 Purpose

This document describes the design, architecture, and technical specifications of WriteShelf. It outlines the system’s functionalities, components, and key design decisions to guide its development and implementation.

1.2 Scope

The WriteShelf project is a web application that provides:

- A digital library for browsing, searching, and reviewing books.
- A platform for writers to publish and manage content.
- Social interaction features, such as following authors and tracking activities.

---

2. System Overview

WriteShelf is built using a client-server architecture. The key components include:

- **Frontend**: HTML, CSS, and JavaScript deliver the user interface, enabling book browsing, story creation, and user interaction.
- **Backend**: Flask handles server-side processing, API endpoints, and business logic.
- **Database**: MongoDB stores persistent data, including users, books, reviews, and activity logs.
- **Session Management**: User sessions are managed using Flask sessions with a secret key for security.
- **Third-Party Integrations**: Integration with external services for authentication and notifications (if applicable).

---

3. System Architecture

3.1 Architectural Design

WriteShelf follows a client-server model:

- The **frontend** communicates with the backend via RESTful APIs.

- The **backend** processes requests, interacts with the database, and returns responses to the frontend.
- The **database** uses collections such as `users`, `books`, `reviews`, and `activities` to store data.

3.2 Data Flow

1. **User Interaction**: Users interact with the web interface to perform actions like searching for books, writing reviews, or following authors.
2. **Request Processing**: The frontend sends HTTP requests to the backend with relevant data.
3. **Data Handling**:
   - The backend processes the request and interacts with MongoDB for data retrieval or updates.
   - Example: A book search queries the `books` collection, while a new review updates the `reviews` field of a book.
4. **Response**: The backend sends HTTP responses containing data or status updates to the frontend.

---

4. Database Design

The database uses MongoDB with the following schema:

4.1 Collections and Fields

1. `users` Collection

- **Fields**:
  - `username`: Unique identifier for each user.
  - `password`: Hashed password for secure login.
  - `email`: User’s email address.
  - `bio`: Short description of the user.
  - `photo`: Profile picture URL (optional).
  - `preferences`: User’s genre preferences.
  - `stats`: A nested object for tracking user activities (e.g., followers, books authored).
  - `created_at`: Timestamp of account creation.

2. `books` Collection

- **Fields**:
  - `title`: Title of the book.
  - `author`: Author's username.
  - `description`: Brief summary of the book.
  - `genres`: List of genres associated with the book.
  - `reviews`: List of review objects (each containing reviewer, rating, and review text).
  - `rating`: Average rating (calculated).
  - `cover_image`: URL of the book's cover image.
  - `created_at`: Timestamp of book addition.

3. `reviews` Collection

- **Fields**:
  - `book_id`: Foreign key linking to the reviewed book.
  - `reviewer`: Username of the reviewer.
  - `rating`: Integer rating from 1 to 5.
  - `review`: Text of the review.
  - `created_at`: Timestamp of review submission.

4. `activities` Collection

- **Fields**:
  - `username`: User performing the activity.
  - `type`: Type of activity (e.g., view, review, publish).
  - `details`: Additional data about the activity (e.g., book ID, title).
  - `timestamp`: Time of activity occurrence.

---

5. Technology Stack

- **Frontend**: HTML, CSS, JavaScript.
- **Backend**: Flask with Flask-CORS.
- **Database**: MongoDB.
- **Security**: bcrypt for password hashing.
- **Configuration**: dotenv for environment variable management.

---

6. Testing Plan

6.1 Unit Testing

- **Frontend**: Verify components like book search, review submission, and profile updates.
- **Backend**: Test API endpoints (e.g., login, signup, book retrieval) and business logic functions.
- **Database**: Test CRUD operations on MongoDB collections to ensure data integrity.

6.2 Integration Testing

- Ensure seamless interaction between the frontend and backend.
- Verify backend communication with the database for complex queries.

6.3 User Acceptance Testing (UAT)

- Scenario-based tests to simulate common user workflows (e.g., registering, browsing books, adding reviews).
- Role-specific tests to confirm proper access control for different user roles.

6.4 Performance Testing

- **Load Testing**: Measure system performance under typical and peak loads.
- **Stress Testing**: Evaluate behavior under extreme conditions.
- **Database Optimization**: Monitor and optimize query performance.
