Software Design Specification (SDS)

Project Name: WriteShelf
Prepared By: Seif Eldin 202200973 Ahmed Mostafa 202201114
Date: 11/9/2024

---

1. Introduction

1.1 Purpose

The purpose of this document is to describe the design, architecture, and technical specifications of the **WriteShelf**. It outlines the functionality, system components, and design decisions to be followed during the development process.

1.2 Scope

This SDS covers the design and implementation details of **WriteShelf**. The software will perform the following major tasks:

- **Task 1**: Serve as a digital library where users can browse and search for books, filter by genre, theme, and author, and view book details.
- **Task 2**: Provide an interactive space for writers to create, publish, and manage written works while engaging with readers through comments and feedback.
- **Task 3**: Enable social interaction features, such as following authors, creating personalized feeds, and viewing activity history.

---

2. System Overview

The WriteShelf system is a **standalone system** designed to operate independently. It interacts with various **external services** for third-party authentication (such as Google and Facebook) and email notifications for user communication It consists of:

- **Frontend**: allowing users to browse the library, write and publish stories, view profiles, and receive notifications.
- **Backend**: The server-side processing and business logic,and user authentication, as well as the handling of interactions between the frontend and the database.
- **Database**: The storage for all persistent data,including user profiles, books, tutorials, comments, and interaction history.

---

3. System Architecture

3.1 Architectural Design

This project follows the client-server architecture, where:

- **Frontend** communicates with the backend using Flask.
- **Backend Handles** the main business logic and manages interactions with the database to process user requests.
- **Database**:.MongoDB serves as the data storage solution, where all persistent information

3.2 Data Flow

1. **User Interaction**: The user interacts with the UI to perform an action, such as creating a new entry, searching for books, rating content, or following a creator.

2. **Request Processing**:Upon interaction, the frontend sends an HTTP request to the backend server, including the necessary data and request details.
3. **Data Handling**:The backend receives and processes the request. Depending on the nature of the request, the server interacts with the MongoDB database to fetch, update, or store data. For instance:
    - When a user searches for a book, the backend queries the database to find matching records.
    - When a user submits a new story, the backend saves the content and updates relevant data fields.
4. **Response**: Once the backend completes data handling, it sends back an HTTP response to the frontend. This response includes any requested data, confirmation of actions taken, or error messages if issues arise.

---

4. Database Design

4.1 Database Schema

The system will store data in MongoDB with the following entities and relationships:

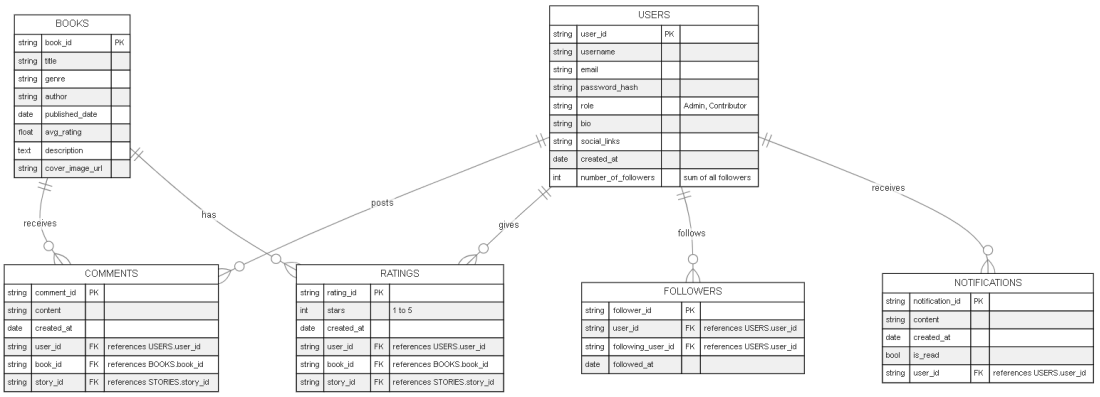

Table 1: USERS

- **user_id**: Unique identifier for each user.
- **username**: User's chosen display name.
- **email**: User's email address (also used for login).
- **password_hash**: Hashed password for secure login.
- **role**: Defines user type (Admin, Contributor, Reader).
- **bio**: Short user bio or description.
- **social_links**: Links to user's social media profiles.
- **created_at**: Timestamp of account creation.

Table 2: BOOKS

- **book_id**: Unique identifier for each book.
- **title**: Title of the book.
- **genre**: Genre classification of the book (e.g., Fantasy, Romance).
- **author**: Author's name (can be a string if one author per book).
- **published_date**: Date the book was published.
- **avg_rating**: Average rating calculated from user ratings.
- **description**: A brief summary of the book's content.
- **cover_image_url**: URL for the book's cover image.

Table 3: COMMENTS

- **comment_id**: Unique identifier for each comment.
- **content**: Text content of the comment.
- **created_at**: Timestamp of when the comment was created.
- **user_id**: Foreign key linking the comment to the user who made it.
- **book_id**: Foreign key linking the comment to the specific book (nullable if it's a story comment).
- **story_id**: Foreign key linking the comment to the specific story (nullable if it's a book comment).

Table 4: RATINGS

- **rating_id**: Unique identifier for each rating.
- **stars**: Star rating from 1 to 5.
- **created_at**: Timestamp when the rating was given.
- **user_id**: Foreign key linking the rating to the user who submitted it.
- **book_id**: Foreign key linking the rating to the specific book (nullable if it's for a story).
- **story_id**: Foreign key linking the rating to the specific story (nullable if it's for a book).

Table 5: FOLLOWERS

- **follower_id**: Unique identifier for each follow relationship.
- **user_id**: Foreign key linking the follower to the user following someone.
- **following_user_id**: Foreign key linking the user being followed.
- **followed_at**: Timestamp for when the follow action occurred.

Table 6: NOTIFICATIONS

- **notification_id**: Unique identifier for each notification.
- **content**: Text content describing the notification.
- **created_at**: Timestamp for when the notification was generated.
- **is_read**: Boolean indicating if the user has read the notification.
- **user_id**: Foreign key linking the notification to the user receiving it

5. Technology Stack

- **Frontend**: HTML, CSS, JS
- **Backend**: Flask.
- **Database**: MongoDB.
- **Hosting**: our computers or if we can afford a VPS.

---

6. Testing Plan

6.1 Unit Testing

Each individual module, function, and component of the WriteShelf system will undergo unit testing. This includes:

- **Frontend Tests**: Validating individual components like book search, comment submission, and profile updates to confirm that each feature functions in isolation.
- **Backend Tests**: Testing individual business logic functions such asfollowUser, addComment, addRating, and data handling methods for creating, updating, and retrieving information.
- **Database Tests**: Verifying database operations, including CRUD operations for users, books, comments, and ratings, to ensure data integrity and correct interactions with MongoDB.

6.2 Integration Testing

Frontend and Backend Integration: ensure that the http responses from the frontend to the backend and vise versa.

Backend and Database Integration: testing that the backend successfully interacts with MongoDB to process complex queries and data updates.

6.3 User Acceptance Testing (UAT)

End users will be involved in testing the system to verify that it meets their requirements and expectations,UAT will involve:
Scenario-Based Testing: as the user will engage in a common interaction with the wep-application
Role-Specific Testing: Test that every role can access their functionalities correctly.

6.4 Performance Testing

Stress and load testing will be conducted to ensure the system can handle the required number of users and operations without degradation in performance.which will include : Load Testing,Stress Testing,Database Optimization: Monitoring.