

ReadRealm

Harshitha Batta hab213@pitt.edu

Neha Navarkar nen28@pitt.edu

INFSCI 2710 - Database Management Systems

Instructor: Dimitriy Babichenko

04/22/2024

Introduction:

ReadRealm is an innovative literary platform tailored to enhance readers' interaction with books and fellow enthusiasts. It acts as a virtual bookshelf where individuals can meticulously monitor their reading endeavors, manage personal reading lists, and access an array of books spanning various genres. The platform stands out by offering an intuitive environment for readers to share insights, leave reviews, and rate books, which collectively contributes to a dynamic and well-informed reading community.

Target Audience:

The primary audience for ReadRealm consists of avid readers, ranging from casual book lovers to ardent bibliophiles seeking a structured approach to track and share their reading experiences. It also caters to reading clubs, authors seeking feedback, and literature students looking to engage with a broader community. Additionally, ReadRealm provides a valuable tool for educators and researchers interested in literary trends and reading habits.

Purpose and Benefits:

The purpose of ReadRealm's database is to streamline the process of cataloging and sharing reading experiences, thereby simplifying the way users discover, engage with, and discuss literature. Its benefits are manifold:

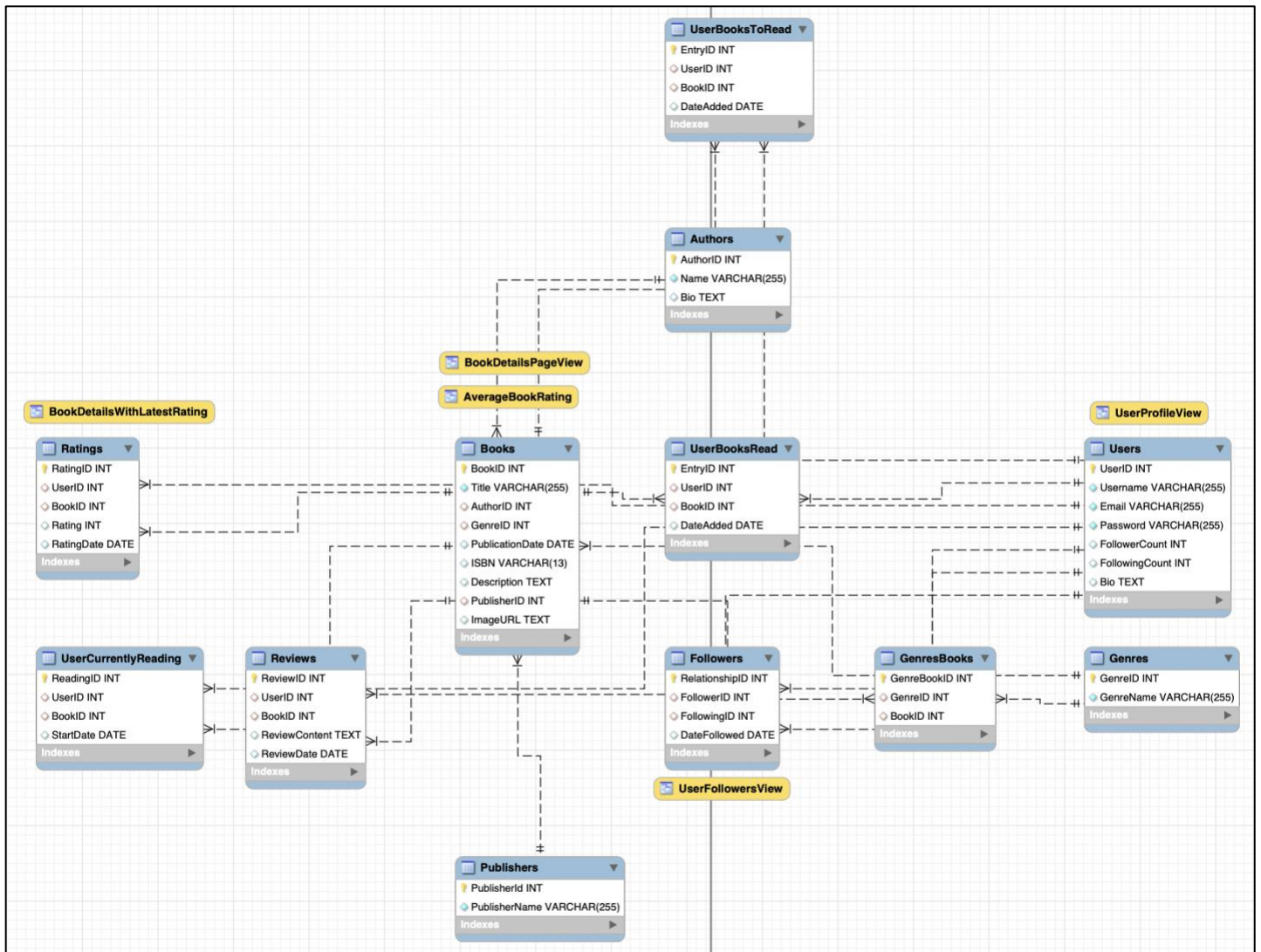
Organizational Tool: Users can meticulously catalog their current reads, completed books, and future reading aspirations, fostering a methodical approach to reading.

Discovery Platform: By exploring community-curated lists and reviews, users can uncover new titles and authors that align with their reading preferences.

Community Building: The platform empowers users to connect with like-minded individuals, participate in literary discussions, and contribute to a culture of shared literary appreciation.

In essence, ReadRealm's database is designed to transform the solitary act of reading into a shared, enriching journey, elevating the overall literary experience for each member of its community.

ER Diagram



Entity 1	Entity 2	Cardinality on Entity 1 side	Cardinality on Entity 2 side	Business Rule(s)
Users	Books	UserBooksRead	UserBooksRead	A user can read many books, and a book can be read by many users.
Users	Books	UserBooksToRead	UserBooksToRead	A user can have many books they intend to read, and a book can be on the 'to-read' list of many users.
Users	Books	UserCurrentlyReading	UserCurrentlyReading	

				A user can be currently reading many books, and a book can be currently read by many users.
Users	Followers			A user can follow many users, and a user can have many followers.
Books	Authors		1	A book has one author, but an author can write many books.
Books	Genre	GenresBooks	GenresBooks	A book can belong to many genres, and a genre can include many books. This is managed through a junction table GenresBooks.
Books	Publishers	1		A book is published by one publisher, but a publisher can publish many books.
User	Reviews			A user can write many reviews, and a book can have many reviews written about it.
User	Ratings			A user can rate many books, and a book can be rated by many users.

SYSTEM REQUIREMENTS

Web Server:

- Flask web framework for Python, utilizing Werkzeug as the WSGI server.
- Deployment on a web server capable of handling Flask applications (e.g., Gunicorn, uWSGI).
- Consideration for reverse proxy setup (e.g., Nginx, Apache) to enhance security and performance.

Database Management System:

- MySQL 8.0 for efficient data storage and retrieval.
- Flask-SQLAlchemy for simplified database interactions within the Flask application.
- Have database schema “readrealm” with tables for storing books, authors, publishers, ratings, genres, and reviews. Ensure you have necessary permissions to create and modify tables.
- Configuration management using a YAML file to store database login details securely.

Programming Languages:

- Python 3.7 or later for server-side scripting using Flask.

- HTML5, CSS3, and JavaScript for the front-end user interface.
- Jinja2 templating engine for seamless integration of Python with HTML templates.
- Bootstrap for styling the webpage.

Bootstrap CDN Links:

- The template includes links to Bootstrap CSS and JavaScript files from a CDN (Content Delivery Network). These links ensure that the Bootstrap styles and functionalities are available to the page.

User Authentication and Security:

- Flask-Security for user authentication and authorization.
- pymysql is a Python package that provides a way to connect to MySQL databases from Python.
- Install pymysql using pip by running `pip install pymysql` in your command line.

User Interface Compatibility:

- The frontend of the application uses HTML, CSS, and JavaScript to render web pages and provide optimal user experience across devices.
- JavaScript for client-side interactivity and dynamic content.
- These are standard web technologies supported by web browsers.

Backend Functionality:

- Utilization of Flask's jsonify function for serializing data into JSON format before returning responses to client requests.
- This ensures that data sent from the server to the client is formatted correctly and can be easily consumed by frontend components.

User Management:

- Flask-Login for user session management.
- User registration and profile management using Flask forms.

Compatibility Testing:

- Cross-browser compatibility testing for major browsers (Chrome, Firefox, Safari).
- Responsive design testing on various devices and screen resolutions.

FRONT-END AND BACK-END INTERACTION OVERVIEW

Front-end Components:

1. HTML5, CSS3, and JavaScript:

These technologies are utilized to create the front-end user interface (UI), handling the rendering of web pages and enabling interactive features for user interactions.

2. Bootstrap CSS Framework:

Bootstrap is incorporated to streamline the styling of front-end components, ensuring a visually appealing and responsive design across different devices and screen sizes.

3. Font Awesome Icons:

Font Awesome icons are integrated to enhance the visual elements of the UI, providing intuitive symbols for various actions and features, improving user experience.

Back-end Server:

1. Python 3.7 and Flask Web Framework:

Python, along with Flask, powers the server-side scripting, handling HTTP requests, and generating dynamic content for the front-end.

2. Flask Routing System:

Flask's routing system is employed to define endpoints that handle specific client requests, such as fetching book details or adding comments, ensuring efficient request handling.

3. Jsonify Function:

The jsonify function from Flask is utilized to serialize data into JSON format before sending responses to the client, facilitating seamless communication between the front-end and back-end components.

Database Interaction:

1. MySQL:

MySQL serves as the relational database management system (RDBMS) for storing book details, user information, and comments, ensuring efficient data storage and retrieval.

2. Pymysql Python Package:

The pymysql Python package facilitates the connection between the Flask application and the MySQL database, enabling database operations such as querying and data insertion.

3. SQL Queries:

SQL queries are executed within Flask routes to retrieve or manipulate data stored in the MySQL database, ensuring seamless interaction between the back-end server and the database.

Asynchronous Communication:

1. JavaScript's XMLHttpRequest (XHR):

XHR is employed to facilitate asynchronous communication between the front-end and back-end components, enabling dynamic updates without page reloads.

2. AJAX (Asynchronous JavaScript and XML) Requests:

AJAX requests are utilized to fetch data from the server asynchronously and update specific sections of the UI without refreshing the entire page, enhancing user experience.

3. WebSocket Functionality (Flask-SocketIO):

WebSocket functionality, implemented using Flask-SocketIO, enables bidirectional communication between the client and server, supporting real-time notifications and updates, improving interactivity.

Password Hashing with Flask-Bcrypt (flask_bcrypt):

1. flask_bcrypt is imported to enhance security by securely hashing passwords before storing them in the database.
2. The Bcrypt module from flask_bcrypt is utilized to hash passwords using the bcrypt hashing algorithm.
3. Hashing passwords before storage protects user data in case of a data breach, as plain-text passwords are not stored.

TEST CASES:

1. Verify login functionality:

Test Case: Attempt to log in with valid username and password.

Expected Result: User should be successfully authenticated and redirected to the dashboard or profile page.

Example: Enter valid username "user123" and password "password123" into the login form, click the login button.

Validation: Ensure the dashboard or profile page loads without errors and user-specific information is displayed.

2. Verify Password Encryption:

Test Case: User's original password stored in database should be hashed.

Expected Result: Password are hashed making it unable to cipher.

Validation: When a user registers, the persons password will be saved in bycrpted and saved to database in User table.

2. Test profile page access:

Test Case: Navigate to the user's profile page after logging in.

Expected Result: User should be able to access their profile page.

Example: Click on the user's profile icon or username displayed after logging in.

Validation: Verify that the profile page loads with the user's information, including username, profile picture, and other relevant details.

3. Validate book details page navigation:

Test Case: Navigate to the details page of a specific book.

Expected Result: User should be redirected to the book details page.

Example: Click on the title or image of a book displayed on the dashboard or search results.

Validation: Confirm that the book details page loads with information such as title, author, description, and user reviews.

4. Test adding books to reading lists:

Test Case: Add a book to the currently reading, read, or want to read lists.

Expected Result: Book should be successfully added to the selected reading list.

Example: Click on the "Add to Currently Reading" button next to a book.

Validation: Check that the book is added to the appropriate reading list and the user's profile or dashboard reflects the change.

5. Test adding review comments:

Test Case: Add a comment or review to a book.

Expected Result: Comment should be successfully added to the book's review section.

Example: Type a comment in the designated text field on the book details page and click "Submit".

Validation: Ensure the comment appears in the book's review section and is attributed to the correct user.

6. Test follower management:

Test Case: Add or remove followers from a user's profile.

Expected Result: Followers should be added or removed accordingly.

Example: Click on the "Follow" button next to another user's profile, then click "Unfollow" to remove them.

Validation: Check that the follower count updates accordingly on both the follower's and following user's profiles.

7. Verify GET requests:

Test Case: Retrieve data from the server using GET requests.

Expected Result: Data should be successfully fetched and displayed on the webpage.

Example: Perform a search for books by genre or author and verify the results.

Validation: Confirm that the requested data is displayed correctly on the webpage without errors.

8. Verify POST requests:

Test Case: Send data to the server using POST requests.

Expected Result: Data should be successfully submitted and processed by the server.

Example: Submit a new comment or update user preferences and verify the changes.

Validation: Check that the submitted data is processed correctly by the server and reflected in the application's interface.

A DATABASE-DRIVEN APPLICATION BUILT ON TOP OF A MYSQL DATABASE

Registration Page:

Register | ReadRealm


Register for ReadRealm

Username:

Email:

Password:

Bio (optional):

 Register


Login | ReadRealm


Welcome to ReadRealm

Login to continue

Username

Password



 Login


If you don't have an account, please [Register](#)

ReadRealm


CatalogProfileLogout

Welcome to ReadRealm

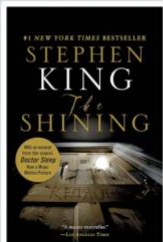
Where Books Reign and Readers Rule!



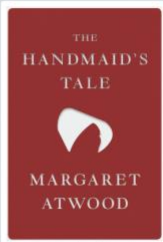
Murder on the Orient Express



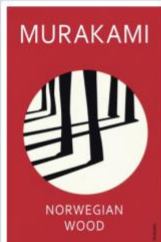
The Hobbit



The Shining




The Handmaid's Tale



Norwegian Wood

ReadRealm

CatalogProfileLogout



Author: Agatha Christie

Publisher: None

Publication Date: 1934-01-01

Summary: A classic mystery novel by Agatha Christie.

Genre: Thriller

Average Rating: ★★★★★

Want to Read

Read It

Currently Reading

Rate the Book: ★★★★★

Reviews

Loved It!

Nice!

Nice!

Great Book!

It was okay...

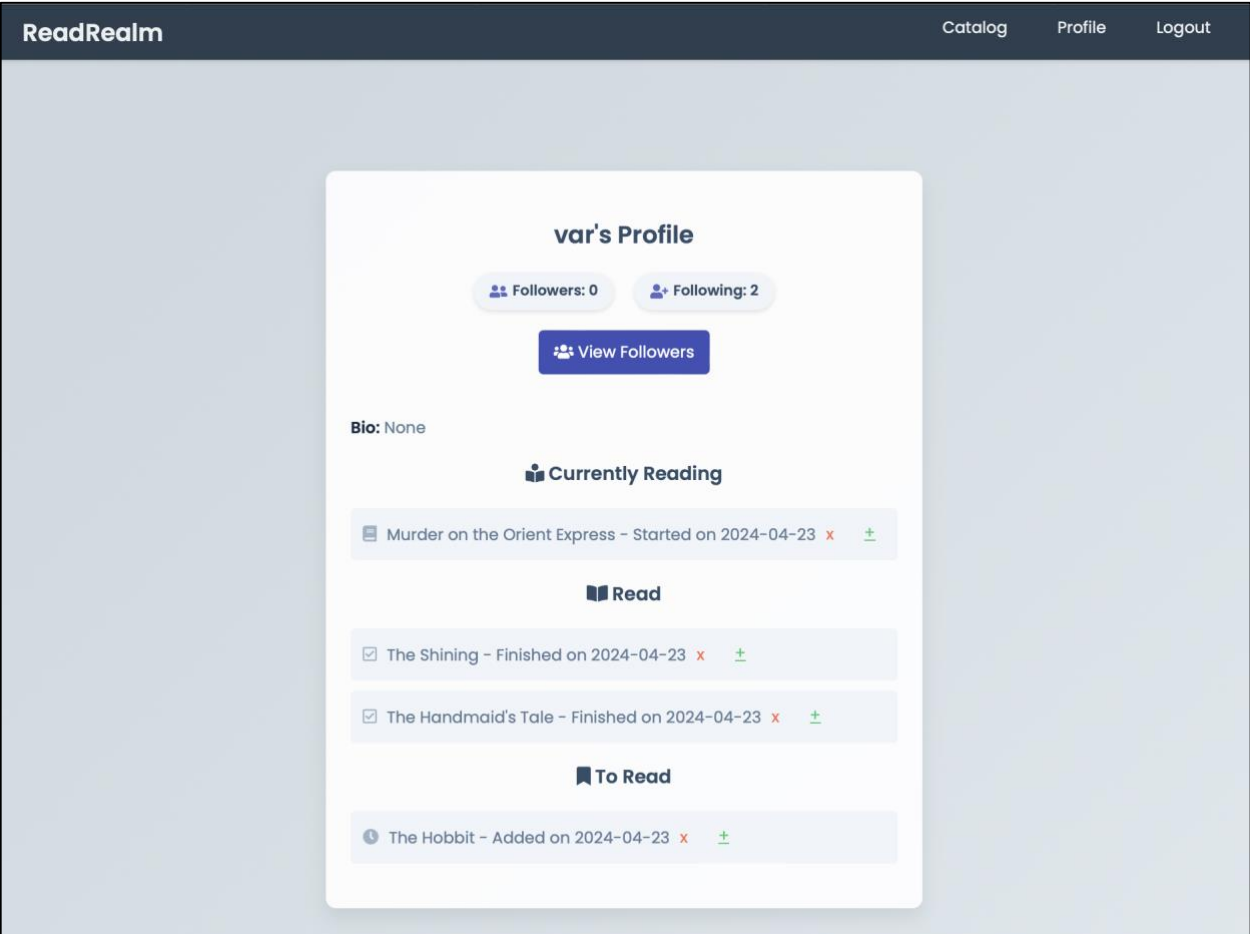
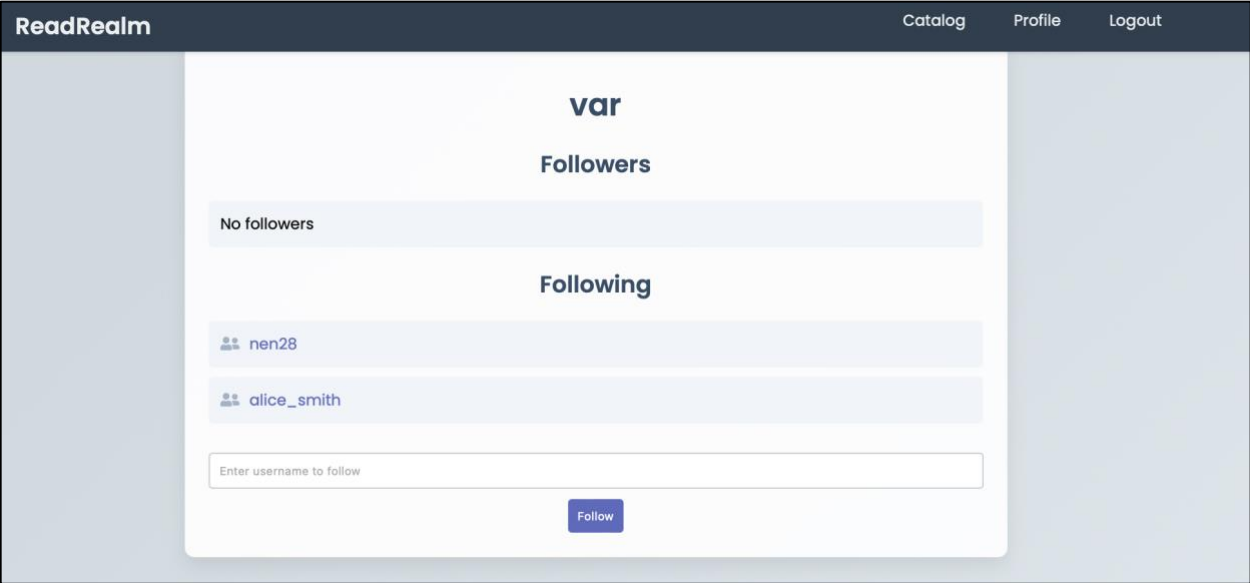
Have read better books

Awesome!!!

Its okay

Add a comment...

Submit



LIMITATIONS:

1. Scalability Concerns:

While the current architecture supports moderate user traffic, scalability may become a concern as the user base grows significantly. The reliance on traditional web servers like Gunicorn or uWSGI may impose limitations on horizontal scalability, requiring additional optimizations or migration to cloud-based solutions.

2. Performance Overhead:

The use of Flask-SQLAlchemy for database interactions may introduce performance overhead, particularly when handling complex queries or large datasets. As the application scales, optimizing database queries and considering alternative ORM frameworks may be necessary to maintain optimal performance.

3. Security Vulnerabilities:

Despite utilizing Flask-Security and pymysql for user authentication and database connections, the system may still be susceptible to security vulnerabilities such as SQL injection or cross-site scripting (XSS) attacks. Regular security audits and updates are essential to mitigate these risks and ensure data integrity and confidentiality.

4. Front-end Complexity:

While HTML5, CSS3, JavaScript, and Bootstrap provide a solid foundation for the front-end user interface, managing the complexity of client-side scripting and ensuring cross-browser compatibility can be challenging. Future enhancements may involve adopting front-end frameworks like React or Vue.js to simplify development and enhance user experience.

FUTURE WORK:

1. Microservices Architecture:

Transitioning to a microservices architecture could address scalability concerns by decoupling components and enabling independent scaling of services. This approach would involve breaking down the monolithic Flask application into smaller, specialized services, enhancing flexibility and scalability.

2. Containerization and Orchestration:

Implementing containerization with Docker and orchestration with Kubernetes could streamline deployment and management of application components. Containerization would facilitate consistency across development, testing, and production environments, while Kubernetes would automate deployment, scaling, and monitoring.

3. Improved Real-time Communication:

Enhancing real-time communication capabilities through advanced WebSocket libraries or integrating with third-party messaging services could enrich user interactions. Features such

as real-time chat, collaborative reading, or live notifications could foster a more engaging and interactive user experience.

4. Machine Learning Integration:

Leveraging machine learning algorithms to personalize recommendations, analyze user behavior, or extract insights from user-generated content could enhance the platform's functionality. Implementing features like personalized book recommendations, sentiment analysis of reviews, or trend prediction could add value to users and publishers.

5. Enhanced Security Measures:

Implementing robust security measures such as two-factor authentication, encryption of sensitive data, and regular security audits could further strengthen the platform's security posture. Additionally, incorporating security headers, CSRF protection, and input validation techniques can mitigate common web vulnerabilities.

6. Accessibility and Internationalization:

Ensuring accessibility standards compliance and supporting multiple languages through internationalization (i18n) and localization (l10n) could broaden the platform's reach and accessibility. Enhancing support for assistive technologies, improving keyboard navigation, and providing language options would enhance inclusivity and user satisfaction.