Library Management System: Project Report

Team Members
Leonardo Kildani, 016147950, leonardo.kildani@sjsu.edu
Asher Alacar, 016247868, alacar@sjsu.edu
Muyi Lin, 016165201, muyi.lin@sjsu.edu
Jiaxin Zhao, 016475966, jiaxin.zhao@sjsu.edu
Shivansh Hedaoo, 015872116, shivansh.hedaoo@sjsu.edu

Goals & Overview

The goal of this project is a library management system built to handle the functions involved in borrowing books and managing library inventory. This system will aim to digitize the traditional library's manual processes and will provide a dashboard interface for members. Our library management system offers the ability to track borrowed books, manage due dates, and store member information. Key components involve user authentication, book catalog management, borrowing and return processes, fee calculations, and reporting tools. The system will use a relational database for effective data storage and management, and a web interface to enable user engagement and make the user experience more comfortable.

Functional Requirements

1. User Registration and Management: The system will allow users to register as library members. It will store user information, including name, contact email, and membership status.

2. Book Cataloging: The system will catalog books, storing information such as title, author, ISBN, and checked-out status. This will allow for easy searching and inventory management.

3. Borrowing Process: Members will be able to check out books through an intuitive interface. The system will track the borrowing date, calculate the due date, and update the checked-out status.

4. Return and Renewal: The system will facilitate the return process, automatically updating book statuses and managing renewals as per member requests.

5. Late Fees Calculation: The system will automatically calculate late fees based on the overdue period and notify members of any outstanding charges.

6. Search and Recommendation: Members will be able to search the library catalog and receive recommendations based on their borrowing history and preferences.

7. Reporting: Library staff will have access to reports on book circulation, member activity, and overdue items, aiding in efficient library management.

Non-Functional Requirements

1. Usability: The system will be designed with a focus on user-friendly interfaces for both library staff and members, ensuring ease of navigation and operation.

2. Reliability: The system will ensure data accuracy and consistency, with robust error handling and validation mechanisms.

3. Performance: The system will be optimized for fast response times, even under heavy load, ensuring a smooth user experience.

4. Security: User data and library information will be protected through secure authentication mechanisms and data encryption, safeguarding against unauthorized access.

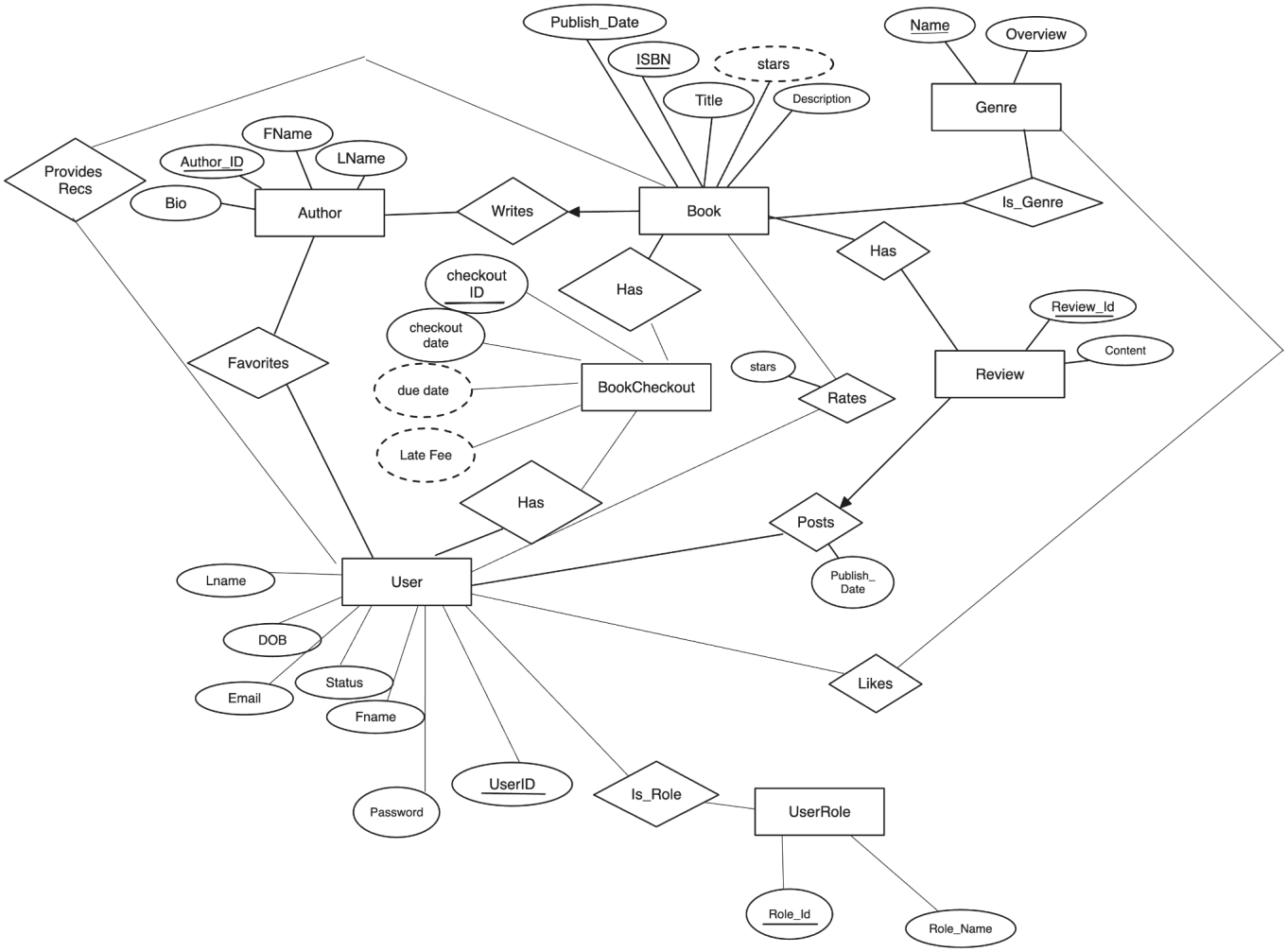
5. Scalability: The system will be built to easily accommodate an increasing number of users and books without degradation in performance.

Architecture

- Front-End
    - HTML, CSS, Bootstrap Framework
- Back-end
    - Spring Boot
    - Thymeleaf Templating Engine
- Database
    - MySQL
    - Spring Data JPA and JDBC for database interaction

The frontend of our system is designed using HTML, CSS, and the bootstrap framework to create a responsive and somewhat friendly looking library interface. These tools allowed us to build a dashboard for library members to search for books, check out books, view their history, and much more. The backend utilizes the java framework SpringBoot. The thymeleaf templating engine is also integrated for rendering the HTML dynamically, allowing server-side processing before the page reaches a client, which helps in handling interactions with the system. The database uses mySQL, a RDBMS to store, manage, and retrieve all data related to the library. This includes member data, information about its books, borrowing transactions, fees, and due dates.

ER Diagram



Database Schema

- Genre(Name, Overview)
    - Name -> Overview - Based on the right hand rule all possible ck is (name) - Name -> Name, Overview
    - 1st normal as all attributes are defined as atomic

    - 2nd normal as all non candidate keys (overview) fully functionally dependent on a candidate key (name)
    - 3rd normal as no non candidate key attribute is transitively dependent on a candidate key
    - BCNF as every functional dependency contains a candidate key
- Book(ISBN, Title, Publish_Date, Description)
    - ISBN -> Title, Publish_Date, Description - Based on the right hand rule all possible ck is (ISBN) - ISBN -> Title, Publish_Date, Description
    - 1st normal as all attributes are defined as atomic
    - 2nd normal as all non candidate keys (Title, Publish_Date, Description) fully functionally dependent on a candidate key (ISBN)
    - 3rd normal as no non candidate key attribute is transitively dependent on a candidate key
    - BCNF as every functional dependency contains a candidate key
- Author(Author_ID, FName, LName, Bio)
    - Author_ID -> FName, LName, Bio - Based on the right hand rule all possible ck is (Author_ID) - Author_ID -> FName, LName, Bio
    - 1st normal as all attributes are defined as atomic
    - 2nd normal as all non candidate keys (FName, LName, Bio) fully functionally dependent on a candidate key (Author_ID)
    - 3rd normal as no non candidate key attribute is transitively dependent on a candidate key
    - BCNF as every functional dependency contains a candidate key
- User(User_Id, Fname, Lname, DOB, Status, password, email)
    - User_ID -> Fname, Lname, DOB, Status, password, email - Based on the right hand rule all possible ck is (User_ID) - User_ID -> Fname, Lname, DOB, Status, password, email
    - 1st normal as all attributes are defined as atomic
    - 2nd normal as all non candidate keys (Fname, Lname, DOB, Status, password, email) fully functionally dependent on a candidate key (User_ID)
    - 3rd normal as no non candidate key attribute is transitively dependent on a candidate key
    - BCNF as every functional dependency contains a candidate key
- UserRole(roleID, Role_name)
    - RoleID -> Role_name - Based on the right hand rule all possible ck is (roleID) - roleID -> Role_name
    - 1st normal as all attributes are defined as atomic
    - 2nd normal as all non candidate keys (Role_name) fully functionally dependent on a candidate key (roleID)
    - 3rd normal as no non candidate key attribute is transitively dependent on a candidate key
    - BCNF as every functional dependency contains a candidate key
- Review(Review_Id,Content)

    - Review_ID -> Content - Based on the right hand rule all possible ck is (review_ID) - review_ID -> Content
    - 1st normal as all attributes are defined as atomic
    - 2nd normal as all non candidate keys (Content) fully functionally dependent on a candidate key (review_ID)
    - 3rd normal as no non candidate key attribute is transitively dependent on a candidate key
    - BCNF as every functional dependency contains a candidate key
- BookCheckout(CheckoutId, CheckoutDate)
    - CheckoutID -> CheckoutDate - Based on the right hand rule all possible ck is (CheckoutID) - CheckoutID -> CheckoutDate
    - 1st normal as all attributes are defined as atomic
    - 2nd normal as all non candidate keys (CheckoutDate) fully functionally dependent on a candidate key (CheckoutID)
    - 3rd normal as no non candidate key attribute is transitively dependent on a candidate key
    - BCNF as every functional dependency contains a candidate key

Major Design Decisions

- Our group decided to go with a Model-View-Service-Controller Design for our project
    - The Model-View-Service-Controller (MVSC) design pattern is a variation of the popular Model-View-Controller (MVC) design pattern, tailored for Java web applications interacting with a MySQL database. In this architecture, the Model comprises two components: Entities (Data Transfer Objects) and Repositories (Data Access Objects). The Entity folder contains Java classes that encapsulate the data stored in the MySQL database, serving as objects that facilitate data transfer between the application and the database. The Repository folder houses classes with CRUD (Create, Read, Update, Delete) methods for each entity and its attributes, providing an abstraction layer for data access operations.
    - The View layer utilizes the Bootstrap framework and HTML for designing the application's user interfaces, while Thymeleaf, a server-side Java template engine, is employed to receive data from and send data to the controllers.
    - The Service layer consists of Java classes that encapsulate the Data Access Objects (DAOs) and apply business logic to ensure data integrity and validity before passing it to and from the database.
    - Finally, the Controller layer utilizes the Service methods to process and send data to or receive data from the View layer. It acts as an intermediary between the View and the Service layers, managing the flow of data and handling user interactions.
- We decided to work with Maven and Spring Boot in this project
    - We opted to leverage Maven and Spring Boot (Java Framework) to streamline the development and deployment process of your Java web application. Maven, a powerful build automation tool, was chosen to efficiently manage all the project dependencies, including Spring Boot, Lombok, and Thymeleaf. This approach

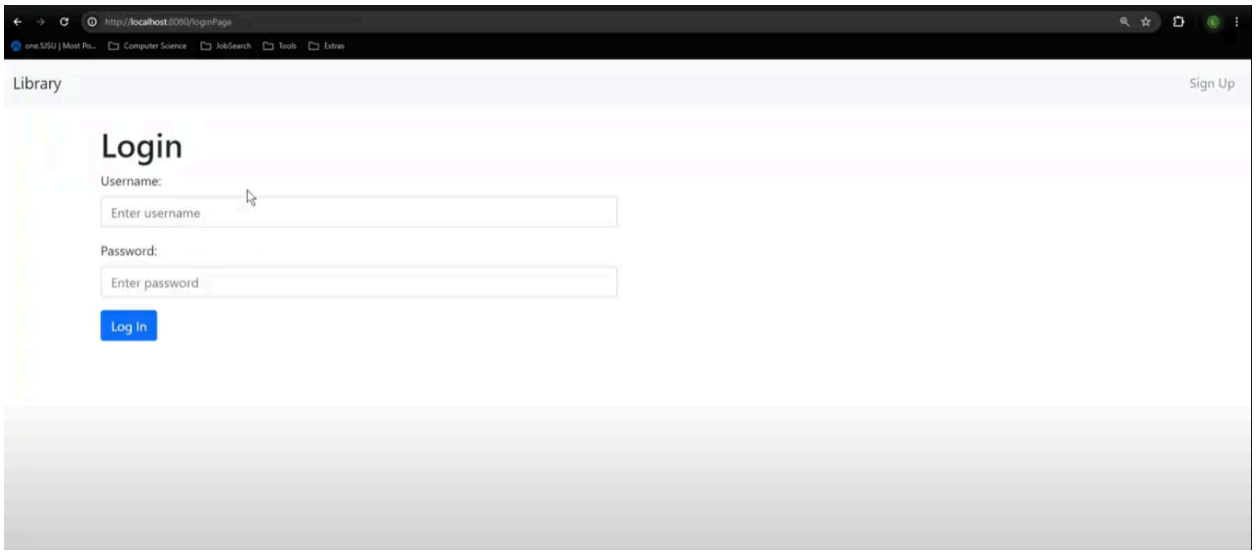
ensures a consistent and reproducible build process, making it easier to integrate and update external libraries as needed. Additionally, we embraced Spring Boot for its opinionated approach to application development, which simplifies the configuration process and enables rapid application setup. Specifically, Spring Boot's integration with Spring Data JPA provides a straightforward and intuitive library and API for establishing database connectivity and handling persistence operations. Furthermore, Spring Beans and its dependency injection capabilities through annotations allow for loosely coupled components, promoting modularity and testability within our codebase.

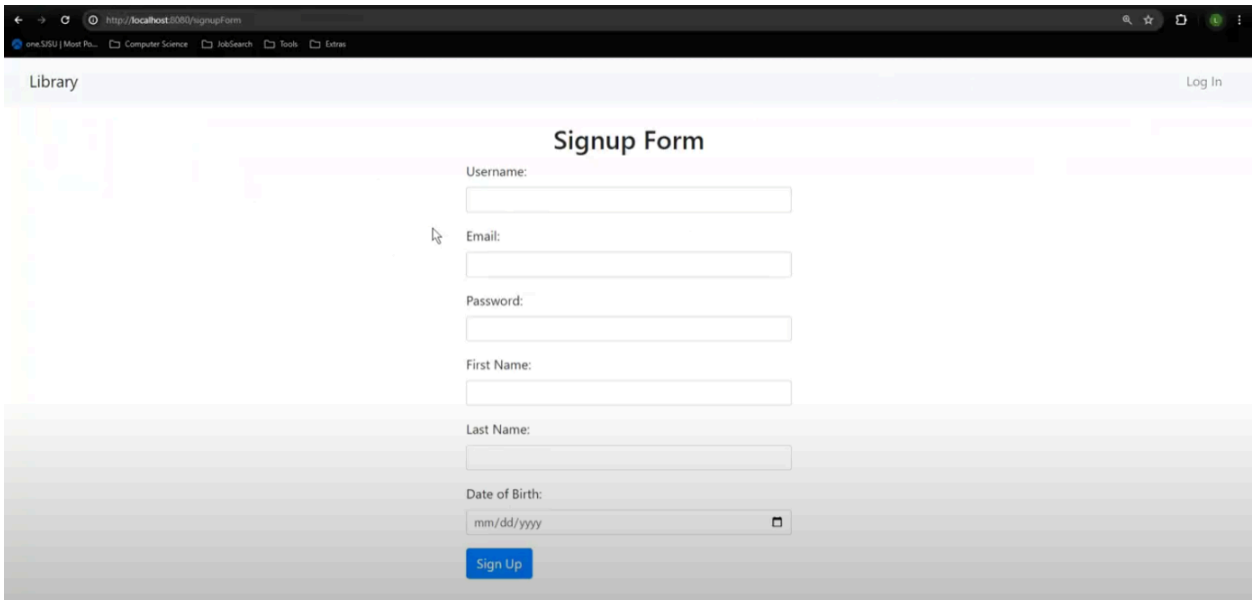
Implementation Details

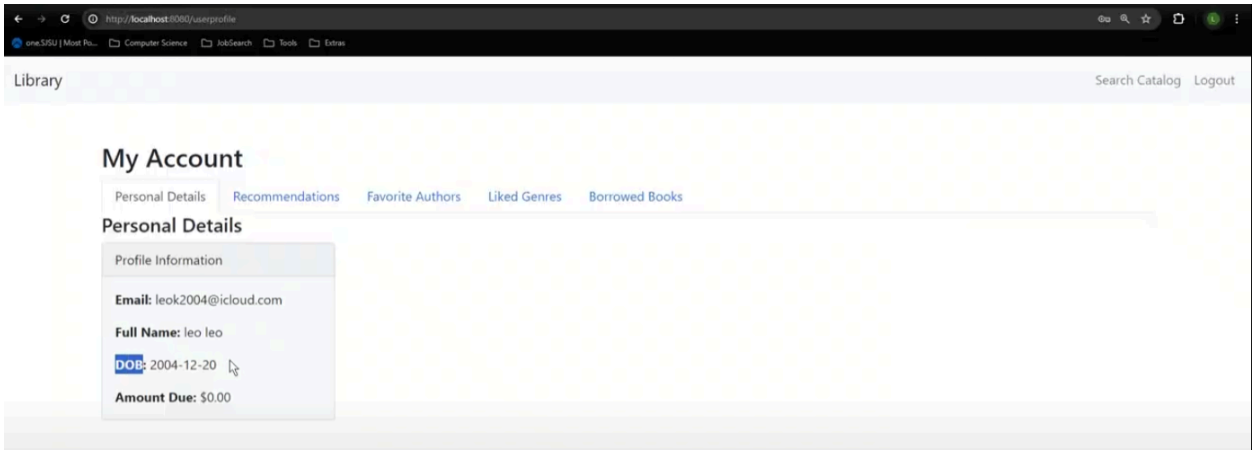
- Entity Directory
    - We make an entity class for each entity in our relational schema (we don't make one for relationship tables) and label them as entities, provided their table names, and the column names that their attributes map to.
    - This allows Spring Data JPA to know how to map the results of an SQL query to an entity class
    - Use the "@Data" annotation from Lombok to reduce boilerplate code like getters/setters and constructors
- Repository Directory
    - Each entity class has its own repository interface. The User and Book entities require more query methods than the basic CRUD operations, therefore they have their own interface implementations, the other repositories extend Spring Data JPA's JpaRepository class, which provides implemented basic CRUD functionality and nothing else. In the case we need a specific query from a JpaRepository, we declare a new method in the interface and annotate it with "@Query" and provide our native query string as a parameter to the annotation
    - Each custom query is done in native SQL and does nothing but return the query results
- Service Directory
    - The service directory encapsulates every Repository class into one large usable interface. Although this does break the Interface Segregation Principle, and couples all the repository interfaces more tightly, we decided to do it for this project due to the smaller size of each interface
    - The methods in the service interface take the results of the queries and apply business logic, like checking for null values or the size of a query result, etc.
- Controller Directory

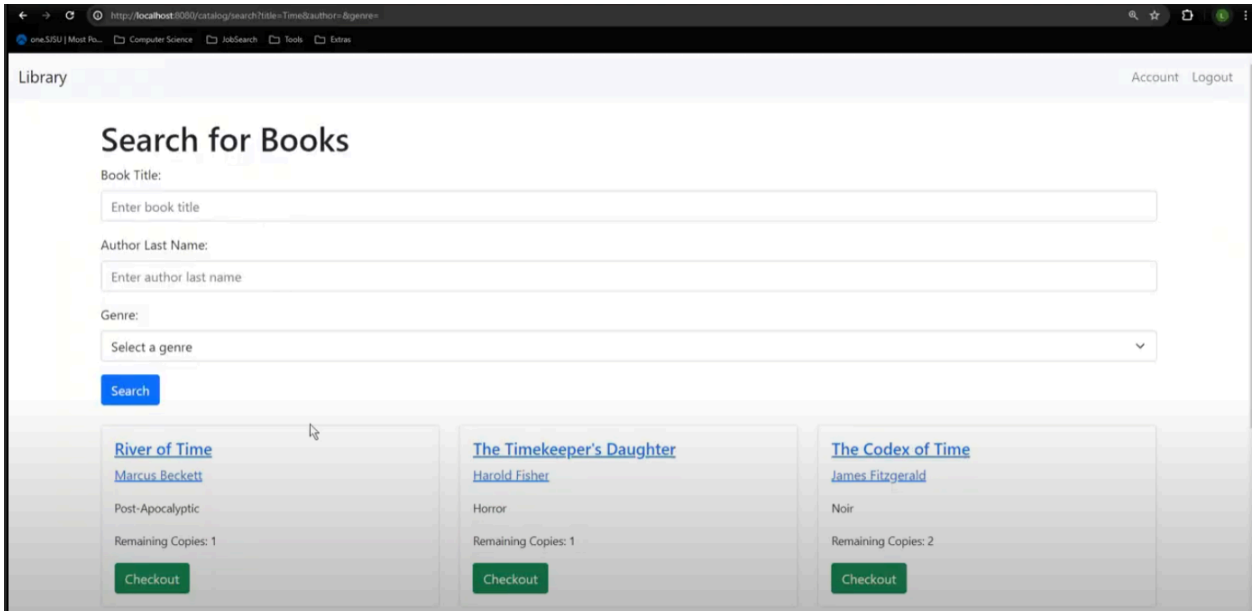
- The controller classes act as an intermediary between our View (html files) and our service. Each html page gets its own controller. The controller maps methods to web endpoints, therefore when users are redirected to those endpoints, the controller method mapped to that endpoint then interacts with the necessary services (which encapsulate the business logic) to perform operations, such as fetching data from a database or processing user input.
- This is all made simple with Spring MVC, which is a submodule of Spring Boot. Using the "@Controller" annotation along with "@GetMapping" and "@PostMapping", we are easily able to map methods to endpoints.

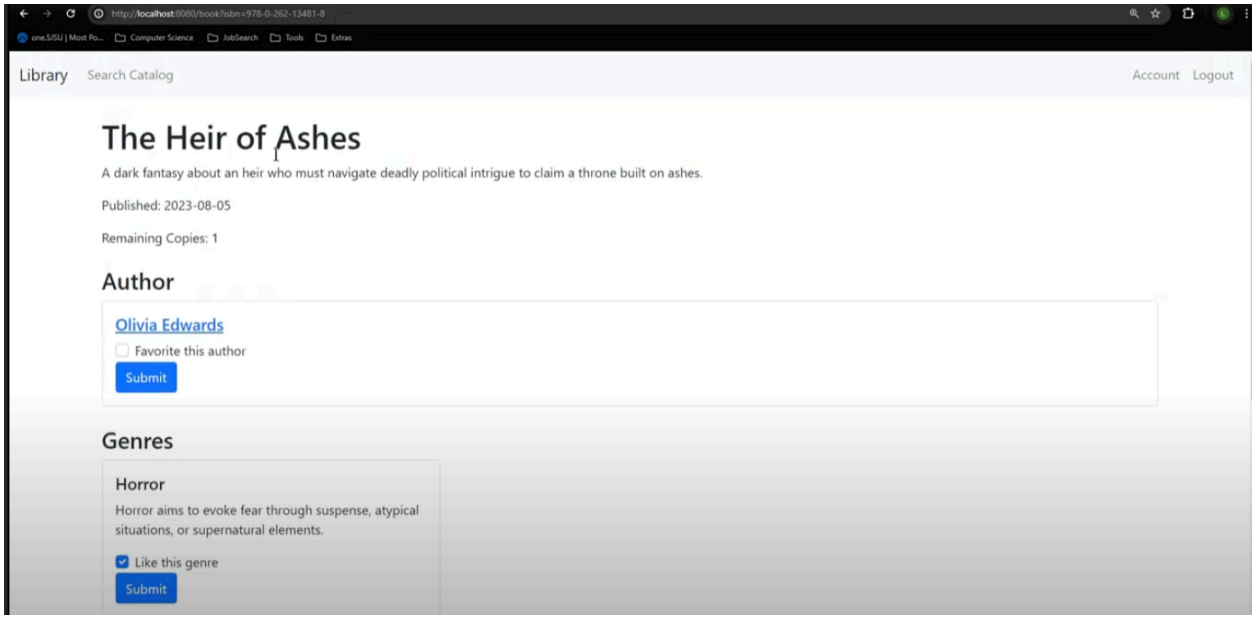
Demonstration






Library
Search Catalog
Logout
My Account
Personal Details
Recommendations
Favorite Authors
Liked Genres
Borrowed Books
Personal Details
Profile Information
Email: leok2004@icloud.com
Full Name: leo leo
DOB: 2004-12-20
Amount Due: $0.00


Library
Account
Logout
Search for Books
Book Title:
Enter book title
Author Last Name:
Enter author last name
Genre:
Select a genre
Search
River of Time
Marcus Beckett
Post-Apocalyptic
Remaining Copies: 1
Checkout
The Timekeeper's Daughter
Harold Fisher
Horror
Remaining Copies: 1
Checkout
The Codex of Time
James Fitzgerald
Noir
Remaining Copies: 2
Checkout


Library
Search Catalog
Account
Logout
The Heir of Ashes
A dark fantasy about an heir who must navigate deadly political intrigue to claim a throne built on ashes.
Published: 2023-08-05
Remaining Copies: 1
Author
Olivia Edwards
Favorite this author
Submit
Genres
Horror
Horror aims to evoke fear through suspense, atypical situations, or supernatural elements.
Like this genre
Submit

Conclusion / Lessons Learned

- Throughout the development of this project, our team gained invaluable experience and skills that will undoubtedly benefit us in future endeavors. We learned the importance of following best practices in version control, such as never pushing directly to the main branch, and instead creating and deleting branches for specific features and designs. This approach allowed us to work efficiently in a rapidly changing environment, minimizing conflicts and ensuring a smooth development process.
- Furthermore, we acquired hands-on experience with Spring Data JPA and native querying in Java, enabling us to interact with the MySQL database at a low level. We learned how to wrap data in objects, enhancing code organization and maintainability.
- Perhaps most importantly, we recognized the significance of effective team collaboration. We understood the value of trust among team members, allowing each individual to contribute their skills without unnecessary criticism or degradation. Instead, we embraced constructive criticism and discipline as the path to success. Maintaining a positive morale throughout the team was crucial, fostering a productive and supportive environment where everyone could thrive.
- As we reflect on our journey, we are grateful for the challenges we faced and the lessons we learned. These experiences have not only strengthened our technical abilities but also our interpersonal and collaboration skills, preparing us for future endeavors in the ever-evolving world of software development.

Future Plans

In the future, our library management system plans to implement robust staff and admin login functionalities to ensure secure and efficient management of the platform. This feature will allow administrators to monitor user activities closely and manage library assets, including the ability to add or remove books, authors, and genres, ensuring the library's catalog remains up-to-date and relevant. Additionally, we aim to enhance the visual appeal of our digital interface by incorporating more color and sophisticated design elements into our HTML pages, making the interface more inviting and intuitive for users. High-quality images of book covers will also be added to the catalog, facilitating quick recognition and enriching the browsing experience. These planned enhancements are designed to improve both the functionality and aesthetics of the system, thereby enriching user interaction and increasing administrative efficiency.