

MACAO POLYTECHNIC UNIVERSITY

Faculty of Applied Sciences

BSc in Computing

**COMP413-412** **ENTERPRISE SYSTEM AND APPLICATION DEVELOPMENT**

**Library Management System**

**Project Report**

Name：Rick GUOYUXUAN

Student No: P2211681

Class: B

Content

[1 Requirement Analysis 4](#_Toc216074790)

[1.1 Business needs 7](#_Toc216074791)

[1.2 User requirements 8](#_Toc216074792)

[1.3 System constraints 9](#_Toc216074793)

[2. System Architecture Design: 10](#_Toc216074794)

[2.1 Communication Flow 12](#_Toc216074795)

[2.2 Architectural Patterns Used 14](#_Toc216074796)

[3. Component Design: 15](#_Toc216074797)

[3.1 Identify Components 15](#_Toc216074798)

[3.2 Class Diagram 20](#_Toc216074799)

[4. Database Design: 22](#_Toc216074800)

[4.1 ER Diagram 22](#_Toc216074801)

[4.2 Schema Design 23](#_Toc216074802)

[5. Interface Design: 28](#_Toc216074803)

[Home Page 28](#_Toc216074804)

[Book list Page 28](#_Toc216074805)

[Login Page 29](#_Toc216074806)

[Register Page 29](#_Toc216074807)

[Set User Permission Page(XAMPP PHPmyAdmin) 30](#_Toc216074808)

[Admin Page 30](#_Toc216074809)

[Books detail page (end-user side) 34](#_Toc216074816)

[My borrowings page (end-user side) 34](#_Toc216074817)

[Search page (end-user side) 35](#_Toc216074818)

[Handle same username register (end-user side) 35](#_Toc216074819)

[6. Appendix: 36](#_Toc216074820)

# Requirement Analysis

**Figure1: Site Map**

**NAVIGATION HIERARCHY:**

**─────────────────────**

Level 0: Home (Landing Page)

Level 1: Main Sections (Books, My Borrowings, Admin)

Level 2: Sub-sections (Management Tabs, Modals)

Level 3: Actions (Forms, Details)

**USER ACCESS BY ROLE:**

**────────────────────**

GUEST:

└─ Home

└─ Books (Read-only)

└─ Book Details

└─ Login/Register

USER (Authenticated):

└─ Home

└─ Books (Can borrow)

└─ Book Details

└─ My Borrowings

└─ Active Borrowings

└─ Return Books

└─ Profile Settings

INSTRUCTOR:

└─ All User access

└─ Admin Dashboard

└─ Books Management (Issue books)

└─ Borrowings Management (View all)

└─ Overdue Books (Process returns)

ADMIN:

└─ All Instructor access

└─ Admin Dashboard

└─ Users Management

└─ Add/Edit/Delete Users

└─ Books Management

└─ Add/Edit/Delete Books

└─ Borrowings Management (Full control)

└─ Overdue Books (Full control)

└─ Statistics Dashboard

**Executive Summary**

The Library Management System is a comprehensive web-based application built using Jakarta EE 9.1, designed to streamline library operations including book inventory management, member registration, and lending processes. The system supports three distinct user roles (Admin, Instructor, and User) with role-based access control, providing a secure and efficient platform for managing library resources.

**Key Technologies:**

* Jakarta EE 9.1 (Servlets, JAX-RS, JPA)
* MySQL 8.0 Database
* Hibernate ORM
* BCrypt Password Encryption
* JWT Token Authentication
* HTML5, CSS3, JavaScript (Vanilla)
* Apache Tomcat 10
* Maven Build Tool

## 1.1 Business needs

The library management system is designed to meet several critical business needs: it provides automated book inventory management by tracking real-time availability, maintaining accurate records of total and available copies, supporting multiple copies of the same title, categorizing books by genre, and managing author information and relationships. It enables efficient member management through role-based registration, maintenance of member profiles and contact details, tracking of individual borrowing histories, and management of user statuses (Active, Inactive, or Suspended). The system streamlines the lending process with quick checkout and return functions, automatic due date calculation, overdue detection with fine computation, enforcement of borrowing limits (maximum of five books per user), and support for book reservations. Additionally, it offers robust administrative control, including comprehensive reporting and statistics, full user and book management capabilities, fine tracking and resolution, and system-wide monitoring to ensure smooth and secure library operations.

1. **Automated Book Inventory Management**
   * Track book availability in real-time
   * Maintain accurate records of total and available copies
   * Support multiple copies of the same book
   * Categorize books by genre/category
   * Manage author information and relationships
2. **Efficient Member Management**
   * Register new members with role-based access
   * Maintain member profiles and contact information
   * Track borrowing history for each member
   * Manage user status (Active, Inactive, Suspended)
3. **Streamlined Lending Process**
   * Quick book checkout and return
   * Automatic due date calculation
   * Overdue detection and fine calculation
   * Borrowing limit enforcement (max 5 books per user)
   * Support for book reservations
4. **Administrative Control**
   * User and book management capabilities
   * Fine management and tracking
   * System-wide monitoring and control

## 1.2 User requirements

The library management system serves three primary user groups with tiered access and functionality. **End Users (Library Members)** can browse and search the catalog, view detailed book information, borrow and return books online, track their personal borrowing history, check due dates and overdue status, receive automated notifications for overdue items, and write reviews or rate books. **Instructors (Library Staff)** have all the capabilities of end users, plus the ability to issue books on behalf of members, process returns, view all borrowing records, manage overdue books, and calculate and collect fines. **Administrators (System Managers)** possess full instructor privileges and additional system-level controls, including the ability to add, edit, or delete books; manage book categories and author information; create and manage user accounts; assign and modify user roles; generate detailed system reports; view comprehensive usage and performance statistics; and perform system configuration and maintenance to ensure optimal operation.

**End Users (Library Members)**

* Browse and search available books
* View detailed book information
* Borrow and return books online
* Track personal borrowing history
* View due dates and overdue status
* Receive notifications for overdue books
* Write reviews and rate books

**Instructors (Library Staff)**

* All user capabilities
* Issue books on behalf of members
* Process book returns
* View all borrowing records
* Manage overdue books
* Calculate and collect fines

**Administrators (System Managers)**

* All instructor capabilities
* Add, edit, and delete books
* Manage book categories and authors
* Create and manage user accounts
* Assign and modify user roles
* Generate system reports
* View comprehensive statistics
* System configuration and maintenance

## 1.3 System constraints

The Library Management System must operate within a defined set of technical, business, and user-related constraints to ensure alignment with institutional standards, regulatory compliance, and operational feasibility. Technically, the system is required to be built using **Jakarta EE 9.1 or higher**, deployed on **Apache Tomcat 10 or a compatible Jakarta EE-compliant application server**, and run on a **Java 11+ runtime environment**, with a **MySQL 8.0** relational database as the sole persistent data store. All external interactions must adhere to a **RESTful API architecture**, and the system must support secure, stateless authentication via **JWT tokens** with a 24-hour expiration. From a business policy perspective, the system must enforce a **maximum borrowing limit of 5 books per user**, a **standard loan period not exceeding 90 days** (with a default of 14 days), and a **fine rate of $1.00 per day for overdue items**, which must be configurable by administrators. Additionally, every book record must include a **unique ISBN**, and all user passwords must meet a **minimum length of 6 characters** (though stronger policies are recommended). User-related constraints mandate that all registered users must provide a **valid and verified email address**, and only users with an **ACTIVE status** are permitted to borrow books. Future enhancements will require that users **return all overdue books and settle outstanding fines before initiating new loans**, though this dependency is not enforced in the initial release. The system must also comply with core non-functional requirements, including **WCAG 2.1 AA accessibility standards**, **HTTPS encryption in production**, **protection against SQL injection and XSS attacks**, and support for **at least 100 concurrent users** with sub-second API response times. These constraints collectively ensure the system remains secure, maintainable, scalable, and fully aligned with both library operational policies and modern software engineering best practices.

# 2. System Architecture Design:

The Library Management System is architected as a multi-tier enterprise application, structured into five distinct layers—Client Tier, Web Tier, Business Tier, Data Access Tier, and Enterprise Information System Tier—each with clearly defined responsibilities and technologies.

The **Client Tier** serves as the user-facing layer, delivering a modern, responsive single-page application (SPA) experience. Built with **HTML5**, **CSS3**, and **ES6+ JavaScript**, it features semantic markup, a mobile-first responsive design using Flexbox, Grid, and media queries, and dynamic UI elements such as modal dialogs, toast notifications, and real-time search with debouncing. The frontend communicates with the backend exclusively via **HTTPS**, sending **RESTful API requests** with **JSON payloads** and including a **JWT token** in the Authorization header. Static assets like index.html and style.css are served directly, while client-side logic in app.js handles routing, DOM updates, form validation, and token persistence using browser **localStorage**.

The **Web Tier**, implemented using **Jakarta EE 9.1**, acts as the API gateway and request router. It exposes a comprehensive set of **JAX-RS REST resources**—including AuthResource, UserResource, BookResource, and BorrowingResource—that handle all HTTP endpoints for authentication, user management, book operations, and borrowing workflows. A dedicated **CORS filter** ensures secure cross-origin communication by managing preflight requests and setting appropriate headers. All incoming requests are deserialized from JSON, processed by the business layer, and returned as properly formatted JSON responses with correct HTTP status codes. Configuration is managed through web.xml, ApplicationConfig, and persistence.xml, ensuring seamless integration with the underlying tiers.

The **Business Tier** encapsulates all core business logic and enforces domain rules through a suite of stateless **service classes**: AuthService, UserService, BookService, and BorrowingService. These services implement critical policies such as limiting users to **5 active borrowings**, enforcing a **14-day default loan period**, calculating **$1.00/day overdue fines**, and validating that only **ACTIVE users** with available copies can borrow books. Security is handled via utility classes like PasswordUtil (using **BCrypt with 10 rounds**) and JWTUtil (for 24-hour token generation and validation). The tier also includes GsonUtil for robust JSON serialization with custom adapters for Java time types, and JPAUtil for managing JPA entity lifecycles, ensuring clean separation of concerns and transactional integrity.

The **Data Access Tier** abstracts database interactions using the **DAO (Data Access Object) pattern**, with a generic GenericDAO<T, ID> base class providing reusable CRUD and pagination methods. Specialized DAOs—such as UserDAO, BookDAO, BorrowingRecordDAO, and ReviewDAO—offer entity-specific queries optimized for performance, including search, aggregation, and statistical operations (e.g., "most borrowed books" or "pending fines"). All DAOs leverage **JPA annotations** for ORM mapping, with careful use of **lazy loading**, **bidirectional relationships**, and **cascade rules**. Transactions are managed declaratively via container-managed transactions, ensuring **ACID compliance** and automatic rollback on errors.

Finally, the **Enterprise Information System Tier** is powered by **MySQL 8.0**, housing a normalized relational schema with nine core tables (users, books, borrowing\_records, etc.) and supporting junction tables like book\_authors. The database enforces **data integrity** through primary/foreign keys, **unique constraints** (on username, email, and ISBN), **check constraints** (e.g., rating between 1–5, available copies ≥ 0), and **NOT NULL** rules. Performance is enhanced with strategic **indexes** on frequently queried columns (title, author, dates) and optimized views like available\_books\_view. **Stored procedures** encapsulate complex operations like borrowing and returning books, while **triggers** automatically maintain data consistency (e.g., updating overdue status). Connection management uses **JDBC pooling (10 connections)** with **READ\_COMMITTED** isolation, ensuring efficient, reliable, and scalable data persistence. Together, these tiers form a secure, maintainable, and high-performance system aligned with enterprise Java standards.

## 2.1 Communication Flow

**1. Client Tier (Browser)**

├─> User clicks "Borrow This Book" button

├─> JavaScript captures event

├─> Constructs JSON payload: {userId: 1, bookId: 5, days: 14}

└─> Sends POST request to /api/borrowing/borrow

└─> Includes JWT token in Authorization header

**2. Web Tier (Tomcat)**

├─> CorsFilter processes request

├─> JAX-RS matches route to BorrowingResource.borrowBook()

├─> Deserializes JSON to Java Map

└─> Calls borrowingService.borrowBook(userId, bookId, issuedById, days)

**3. Business Tier (Service Layer)**

├─> BorrowingService.borrowBook() validates request

├─> Checks user exists (calls userDAO.findById())

├─> Checks book availability (calls bookDAO.findById())

├─> Verifies borrowing limit (calls borrowingRecordDAO.countActiveBorrowingsByUser())

├─> Creates BorrowingRecord entity

├─> Calculates due date (current + 14 days)

└─> Calls borrowingRecordDAO.save(record)

**4. Data Access Tier (DAO Layer)**

├─> BorrowingRecordDAO.save() method

├─> Creates EntityManager

├─> Begins transaction

├─> Persists entity

├─> Commits transaction

├─> Updates book.availableCopies - 1

├─> Calls bookDAO.update(book)

└─> Returns saved entity

**5. EIS Tier (Database)**

├─> INSERT INTO borrowing\_records (...)

├─> UPDATE books SET available\_copies = available\_copies - 1

├─> Trigger checks constraints

├─> Commit transaction

└─> Returns success

**6. Response Flow (Bottom-Up)**

├─> DAO returns entity to Service

├─> Service constructs response Map

├─> Service returns to Web Tier

├─> JAX-RS serializes to JSON

├─> Sets HTTP 200 OK status

└─> Returns JSON to client

**7. Client Tier (Browser)**

├─> Receives JSON response

├─> JavaScript parses response

├─> Shows success toast notification

├─> Updates UI (decrements available copies)

└─> Refreshes "My Borrowings" list

## 2.2 Architectural Patterns Used

The system employs a robust set of well-established **architectural patterns** to ensure modularity, maintainability, and separation of concerns. It follows the **Model-View-Controller (MVC)** pattern, where **JPA entities** serve as the Model, **HTML/CSS/JavaScript** forms the View, and **JAX-RS resources** act as the Controller. Database interactions are abstracted using the **Data Access Object (DAO)** pattern, featuring a generic base class with common CRUD and pagination operations, extended by entity-specific DAOs that function as **Repositories**, providing a collection-like interface while hiding persistence complexity. Business logic is encapsulated in a dedicated **Service Layer**, which defines transaction boundaries and orchestrates multiple DAOs to enforce domain rules. **Dependency Injection (DI)**—primarily via constructor injection—promotes loose coupling and enhances testability. Additional patterns include the **Filter Pattern** (used for CORS and future authentication via a chain of responsibility), the **Builder Pattern** (for constructing Gson configurations and JWT tokens), and the **Singleton Pattern** (applied to shared resources like EntityManagerFactory and utility instances such as Gson). To support future growth, the architecture incorporates comprehensive **scalability considerations**: **horizontal scaling** is enabled through a stateless REST API, JWT-based authentication (eliminating server-side sessions), and the ability to deploy multiple Tomcat instances behind a load balancer; **vertical scaling** is supported via configurable JDBC connection pools, JVM heap tuning, and optimized database queries; and while caching is not implemented in the initial version, the design anticipates integration with **Redis** for session and query caching. Furthermore, the database layer is prepared for **scaling through read replicas, master-slave replication, and table partitioning** to handle large datasets and high read loads, ensuring the system can evolve to meet increasing demand without architectural overhaul.

# 3. Component Design:

## 3.1 Identify Components

The system is organized into distinct components following the layered architecture pattern. Each component has well-defined responsibilities and interfaces.

The Library Management System is structured around modern Jakarta EE–based components, eschewing legacy technologies like JSPs and EJBs in favor of a clean, layered RESTful architecture. At the web tier, **JAX-RS resource classes** (e.g., AuthResource, UserResource, BookResource, BorrowingResource) serve as stateless controllers—functionally replacing traditional servlets—by handling HTTP requests, processing JSON payloads, and returning API responses. A **CORS filter** acts as a standard Jakarta EE servlet filter to manage cross-origin security. The business logic is encapsulated in **stateless service classes** (AuthService, UserService, etc.), which operate like lightweight session beans by orchestrating data access, enforcing business rules, and defining transaction boundaries without relying on full EJB containers. The data access layer is built on **DAO (Data Access Object) components**—including a generic GenericDAO base class and entity-specific DAOs (UserDAO, BookDAO, etc.)—that abstract JPA operations and interact with the **MySQL 8.0 database** via JPAUtil-managed EntityManager instances. Frontend interaction is handled entirely by a **JavaScript-based single-page application** (HTML5/CSS3/ES6+) with no server-side rendering, while utility components like PasswordUtil and JWTUtil provide reusable, stateless security functions. Together, these components form a cohesive, modular system aligned with contemporary enterprise Java best practices.

Although the Library Management System is implemented using a modern, lightweight Jakarta EE 9.1 stack with a RESTful API and a JavaScript-based single-page application (SPA) frontend—thereby avoiding server-side rendering technologies like JSP—the system’s components can be conceptually mapped to traditional Java EE building blocks as required by the project specification. Specifically, the **JAX-RS resource classes** (e.g., AuthResource, UserResource) fulfill the role of **Servlets**, as they handle HTTP requests, manage request/response cycles, and route traffic—functionally equivalent to front controller servlets in a classic MVC web application. While **JSP pages are not used**, the index.html and dynamic JavaScript views serve the same purpose as JSPs in presenting data to the user, albeit in a decoupled, client-side manner. Similarly, the **stateless service classes** (AuthService, BorrowingService, etc.) implement business logic, transaction management, and orchestration of data access—mirroring the responsibilities of **Stateless Session Beans (EJBs)**—though they are implemented as plain Java beans to reduce container dependency and align with microservice-friendly practices. Finally, the **DAO layer** built on JPA with EntityManager provides the data persistence functionality analogous to what EJBs with container-managed persistence would offer. Thus, while the system adopts contemporary architectural patterns for scalability and maintainability, its components directly correspond to the core Java EE constructs (Servlets, JSP, EJBs) in terms of **responsibility and role**, satisfying the foundational principles of the Java EE component model.

**JPA Entity Classes** represent the domain model and map to database tables.

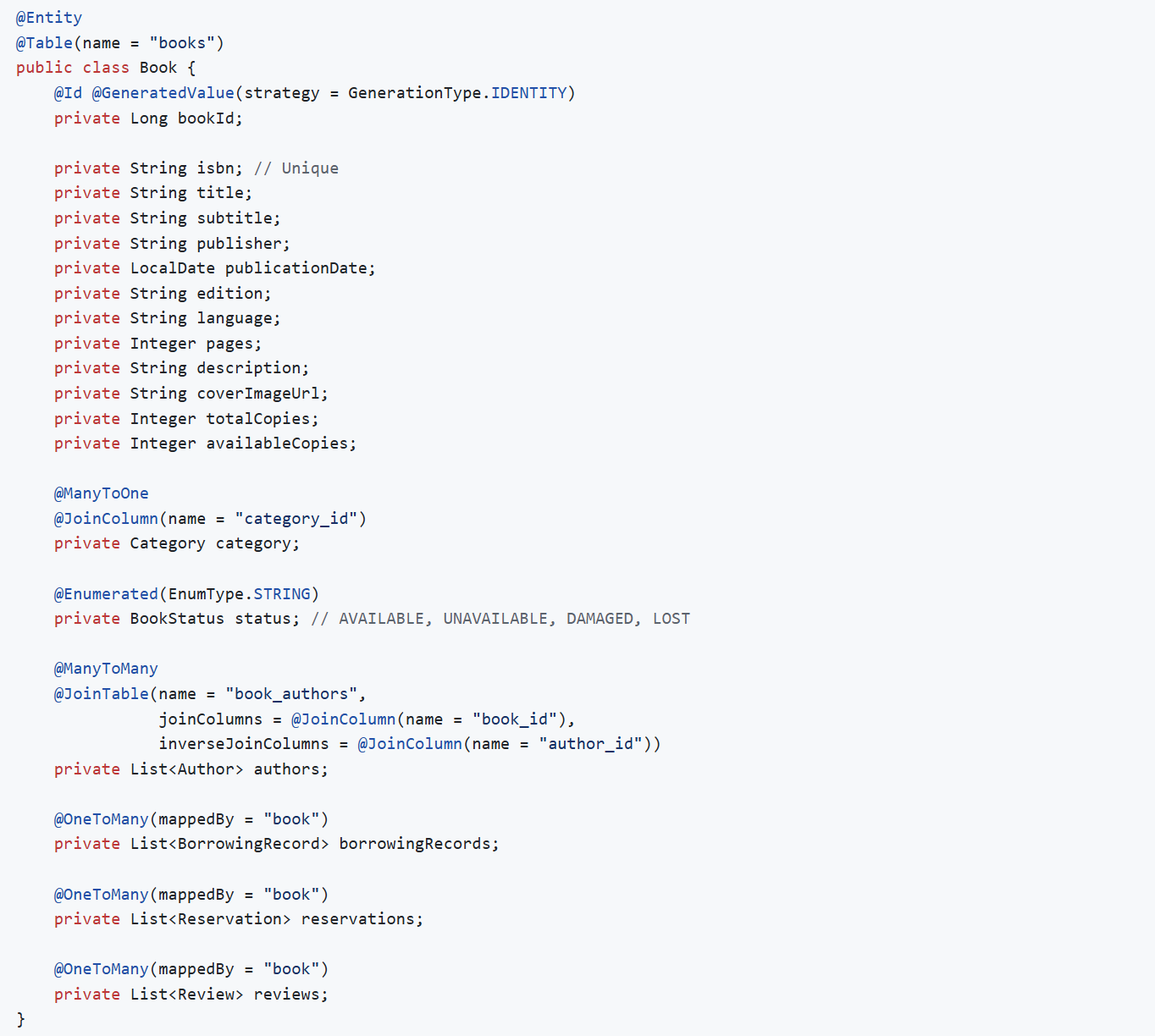
**User Entity**



**Relationships:**

* One User → Many BorrowingRecords
* One User → Many Reservations
* One User → Many Reviews

**Book Entity**



**Relationships:**

* Many Books → One Category
* Many Books → Many Authors (junction table: book\_authors)
* One Book → Many BorrowingRecords
* One Book → Many Reservations
* One Book → Many Reviews

**BorrowingRecord Entity**



**Relationships:**

* Many BorrowingRecords → One User (borrower)
* Many BorrowingRecords → One Book
* Many BorrowingRecords → One User (issued by)
* Many BorrowingRecords → One User (returned to)

**Other Entities:**

**Category Entity**

* categoryId (PK)
* categoryName (Unique)
* description
* One Category → Many Books

**Author Entity**

* authorId (PK)
* authorName
* biography
* birthDate
* nationality
* Many Authors → Many Books

**Reservation Entity**

* reservationId (PK)
* user (FK)
* book (FK)
* reservationDate
* expiryDate
* status (ACTIVE, FULFILLED, CANCELLED, EXPIRED)
* notified (Boolean)

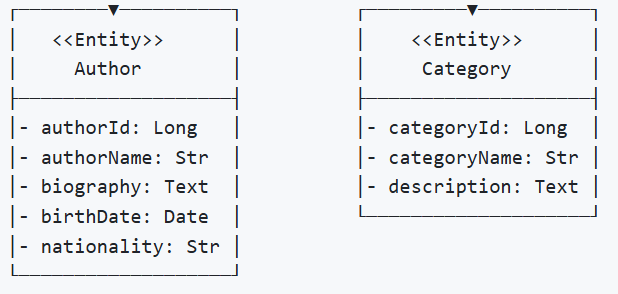
**Review Entity**

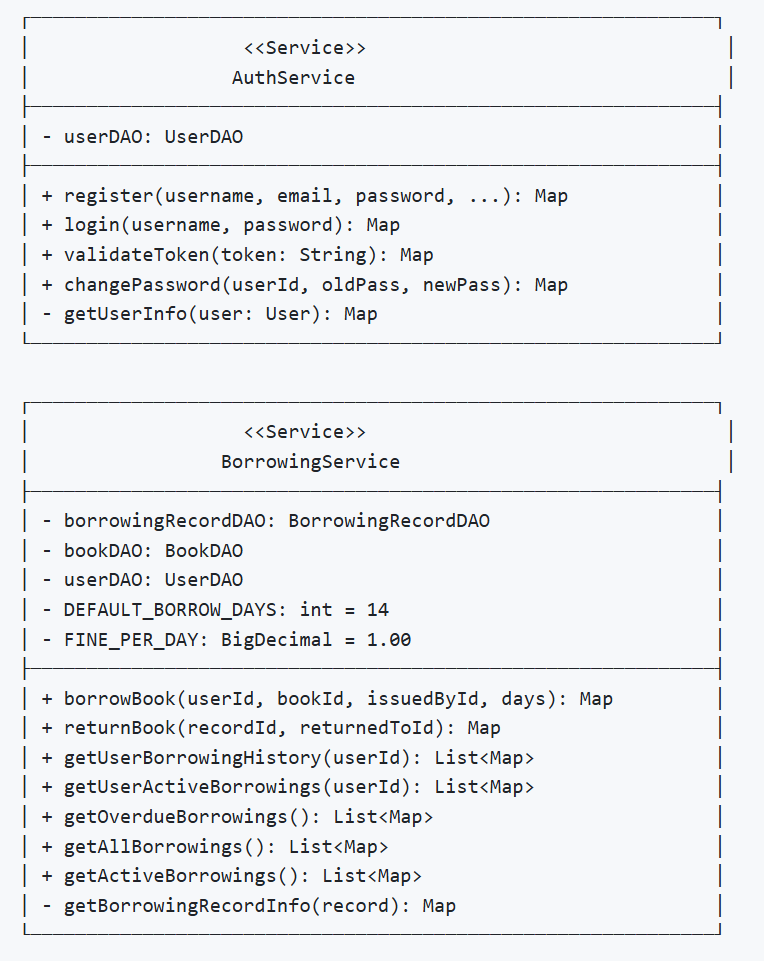
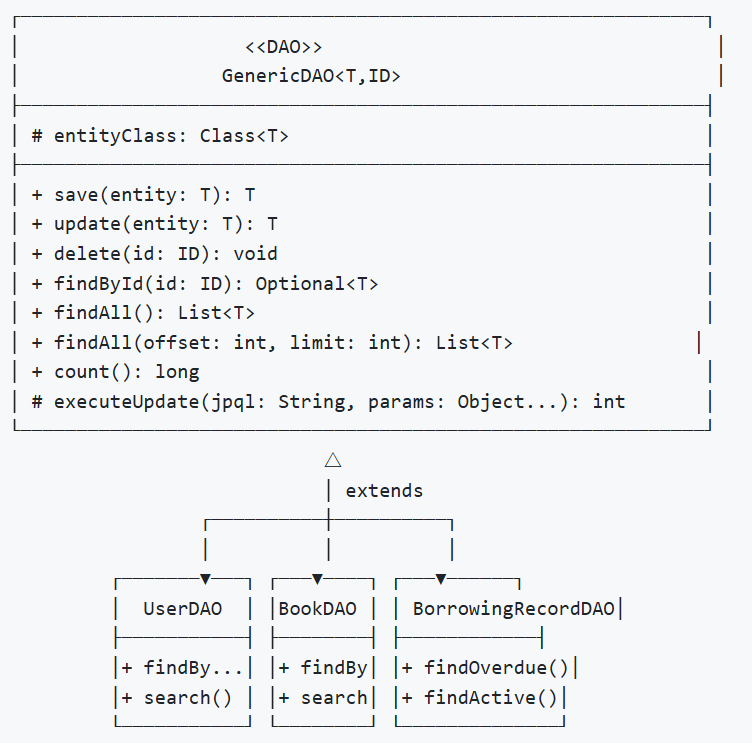
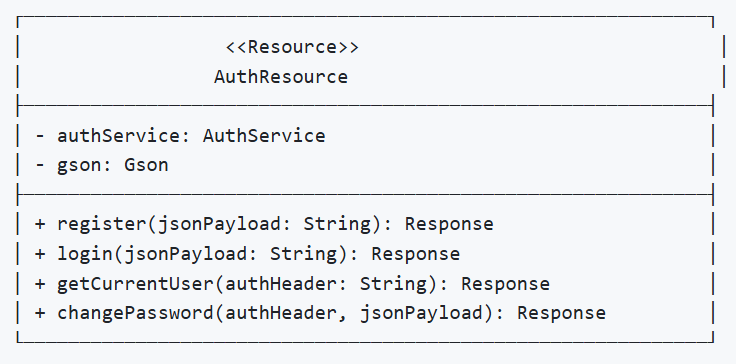
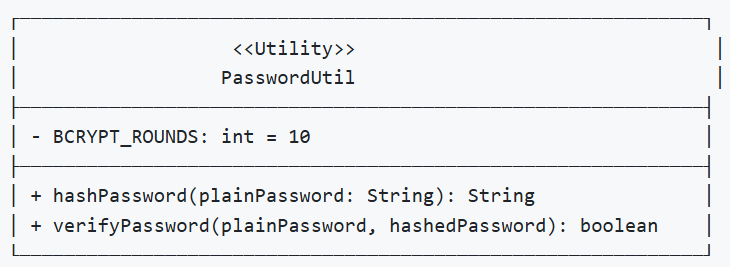
* reviewId (PK)
* user (FK)
* book (FK)
* rating (1-5)
* reviewText
* reviewDate
* Unique constraint: (user\_id, book\_id)

**Fine Entity**

* fineId (PK)
* borrowingRecord (FK)
* user (FK)
* fineAmount
* fineReason
* fineDate
* paymentDate
* status (PENDING, PAID, WAIVED)
* paymentMethod

## 3.2 Class Diagram





# 4. Database Design:

## 4.1 ER Diagram

## 4.2 Schema Design

**1. USERS**

* **Primary Key:** user\_id (BIGINT, AUTO\_INCREMENT)
* **Attributes:**
  + username (VARCHAR(50), UNIQUE, NOT NULL)
  + email (VARCHAR(100), UNIQUE, NOT NULL)
  + password\_hash (VARCHAR(255), NOT NULL)
  + first\_name (VARCHAR(50), NOT NULL)
  + last\_name (VARCHAR(50), NOT NULL)
  + phone (VARCHAR(20))
  + address (TEXT)
  + role (ENUM: 'ADMIN', 'INSTRUCTOR', 'USER')
  + status (ENUM: 'ACTIVE', 'INACTIVE', 'SUSPENDED')
  + registration\_date (TIMESTAMP)
  + last\_login (TIMESTAMP)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**2. BOOKS**

* **Primary Key:** book\_id (BIGINT, AUTO\_INCREMENT)
* **Foreign Key:** category\_id → categories(category\_id)
* **Attributes:**
  + isbn (VARCHAR(20), UNIQUE, NOT NULL)
  + title (VARCHAR(255), NOT NULL)
  + subtitle (VARCHAR(255))
  + publisher (VARCHAR(100))
  + publication\_date (DATE)
  + edition (VARCHAR(50))
  + language (VARCHAR(50), DEFAULT 'English')
  + pages (INT)
  + description (TEXT)
  + cover\_image\_url (VARCHAR(500))
  + total\_copies (INT, NOT NULL, DEFAULT 1)
  + available\_copies (INT, NOT NULL, DEFAULT 1)
  + status (ENUM: 'AVAILABLE', 'UNAVAILABLE', 'DAMAGED', 'LOST')
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**3. BORROWING\_RECORDS**

* **Primary Key:** record\_id (BIGINT, AUTO\_INCREMENT)
* **Foreign Keys:**
  + user\_id → users(user\_id)
  + book\_id → books(book\_id)
  + issued\_by → users(user\_id)
  + returned\_to → users(user\_id)
* **Attributes:**
  + borrow\_date (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)
  + due\_date (TIMESTAMP, NOT NULL)
  + return\_date (TIMESTAMP)
  + status (ENUM: 'BORROWED', 'RETURNED', 'OVERDUE', 'LOST')
  + fine\_amount (DECIMAL(10,2), DEFAULT 0.00)
  + notes (TEXT)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**4. CATEGORIES**

* **Primary Key:** category\_id (BIGINT, AUTO\_INCREMENT)
* **Attributes:**
  + category\_name (VARCHAR(100), UNIQUE, NOT NULL)
  + description (TEXT)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**5. AUTHORS**

* **Primary Key:** author\_id (BIGINT, AUTO\_INCREMENT)
* **Attributes:**
  + author\_name (VARCHAR(100), NOT NULL)
  + biography (TEXT)
  + birth\_date (DATE)
  + nationality (VARCHAR(50))
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**6. BOOK\_AUTHORS (Junction Table)**

* **Composite Primary Key:** (book\_id, author\_id)
* **Foreign Keys:**
  + book\_id → books(book\_id) ON DELETE CASCADE
  + author\_id → authors(author\_id) ON DELETE CASCADE

**7. RESERVATIONS**

* **Primary Key:** reservation\_id (BIGINT, AUTO\_INCREMENT)
* **Foreign Keys:**
  + user\_id → users(user\_id)
  + book\_id → books(book\_id)
* **Attributes:**
  + reservation\_date (TIMESTAMP)
  + expiry\_date (TIMESTAMP, NOT NULL)
  + status (ENUM: 'ACTIVE', 'FULFILLED', 'CANCELLED', 'EXPIRED')
  + notified (BOOLEAN, DEFAULT FALSE)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**8. REVIEWS**

* **Primary Key:** review\_id (BIGINT, AUTO\_INCREMENT)
* **Foreign Keys:**
  + user\_id → users(user\_id)
  + book\_id → books(book\_id)
* **Unique Constraint:** (user\_id, book\_id)
* **Attributes:**
  + rating (INT, CHECK: rating BETWEEN 1 AND 5)
  + review\_text (TEXT)
  + review\_date (TIMESTAMP)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**9. FINES**

* **Primary Key:** fine\_id (BIGINT, AUTO\_INCREMENT)
* **Foreign Keys:**
  + record\_id → borrowing\_records(record\_id)
  + user\_id → users(user\_id)
* **Attributes:**
  + fine\_amount (DECIMAL(10,2), NOT NULL)
  + fine\_reason (VARCHAR(255))
  + fine\_date (TIMESTAMP)
  + payment\_date (TIMESTAMP)
  + status (ENUM: 'PENDING', 'PAID', 'WAIVED')
  + payment\_method (VARCHAR(50))
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

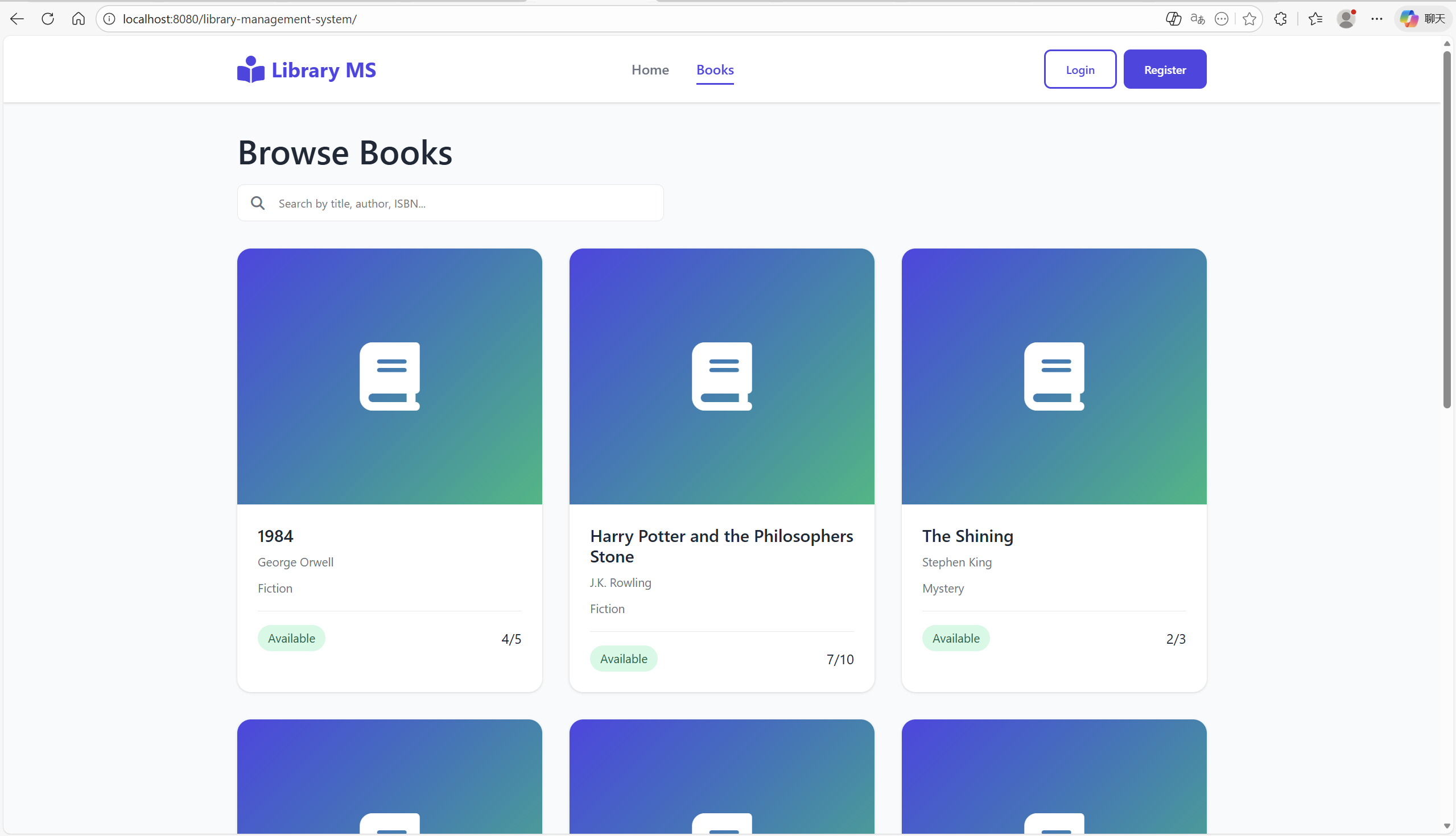
**Relationship Cardinalities**

| **Relationship** | **Entity 1** | **Cardinality** | **Entity 2** | **Type** |
| --- | --- | --- | --- | --- |
| R1 | User | 1..\* | BorrowingRecord | One-to-Many |
| R2 | Book | 1..\* | BorrowingRecord | One-to-Many |
| R3 | User | 1..\* | Reservation | One-to-Many |
| R4 | Book | 1..\* | Reservation | One-to-Many |
| R5 | User | 1..\* | Review | One-to-Many |
| R6 | Book | 1..\* | Review | One-to-Many |
| R7 | Category | 1..\* | Book | One-to-Many |
| R8 | Book | *..* | Author | Many-to-Many |
| R9 | BorrowingRecord | 1..1 | Fine | One-to-One |
| R10 | User | 1..\* | Fine | One-to-Many |

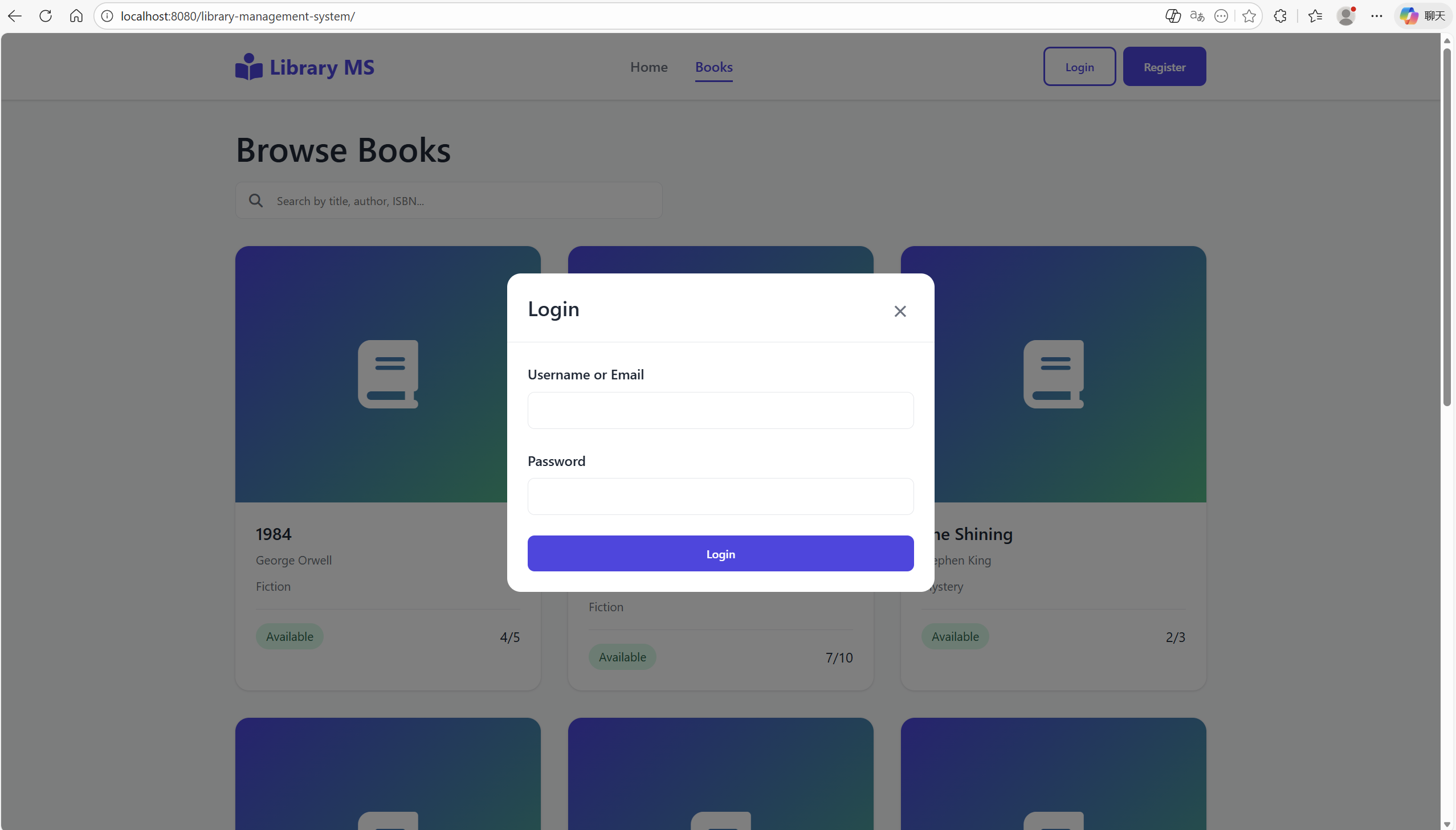
# 5. Interface Design:

## Home Page

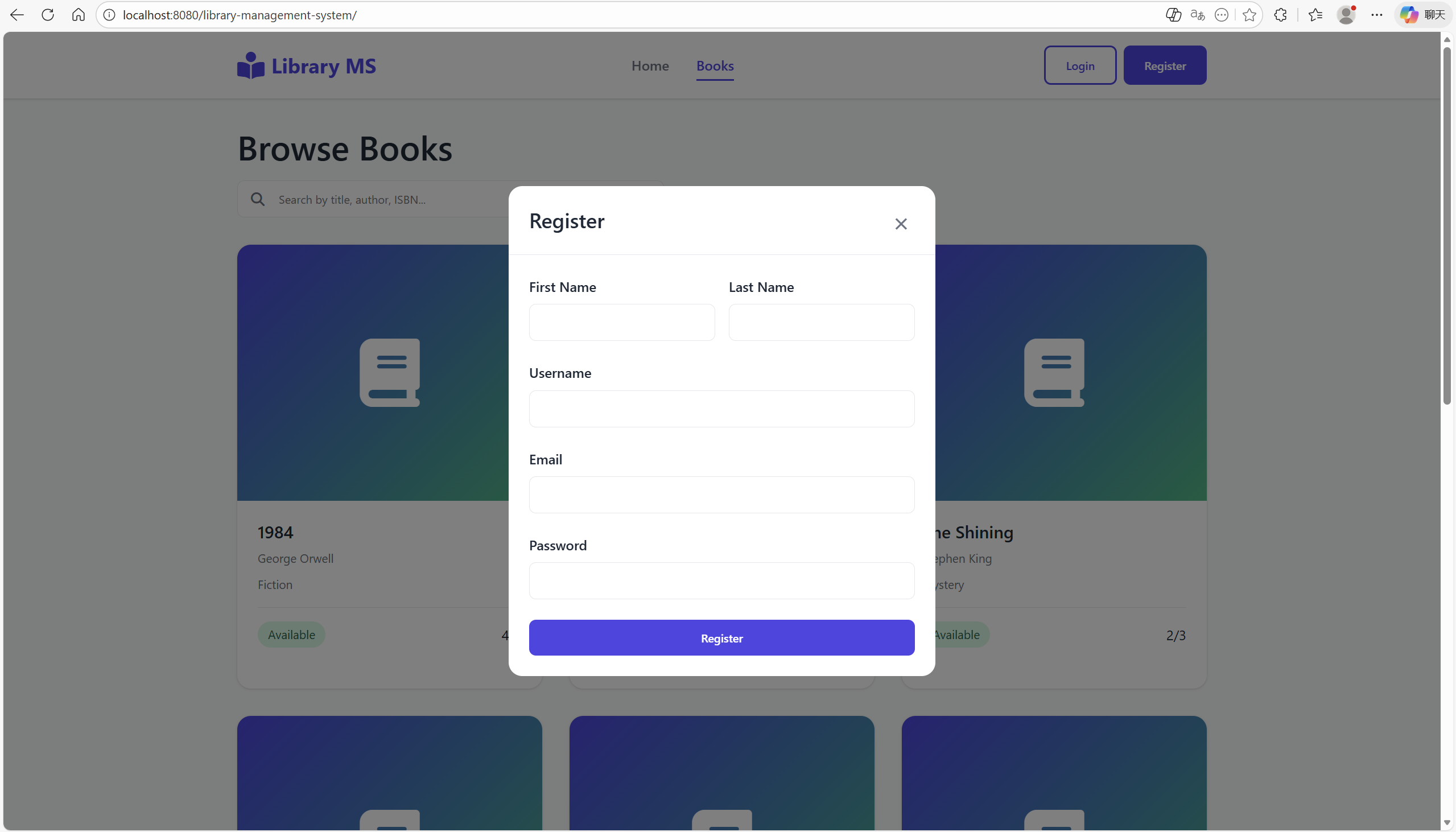
## Book list Page



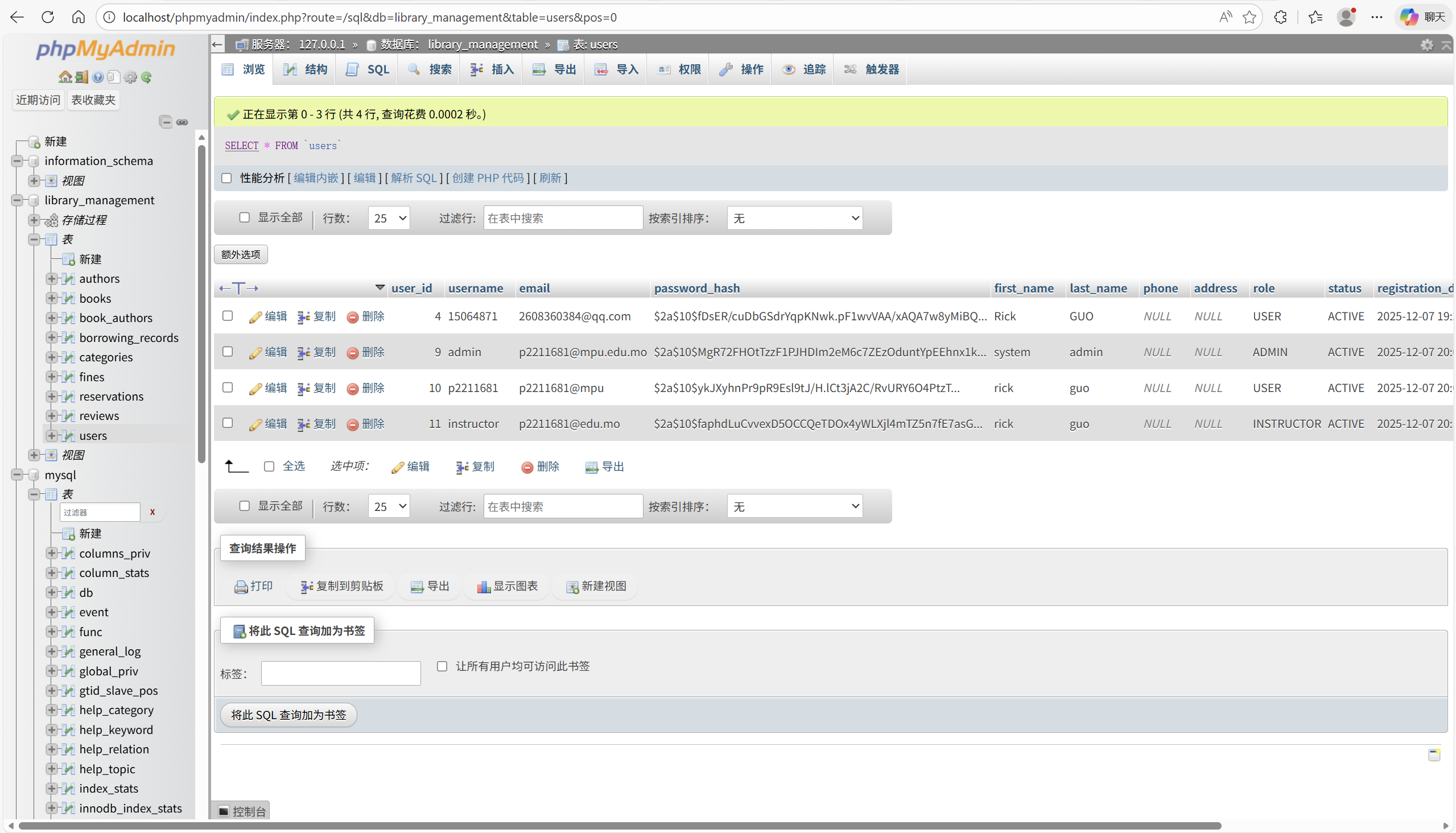
## Login Page



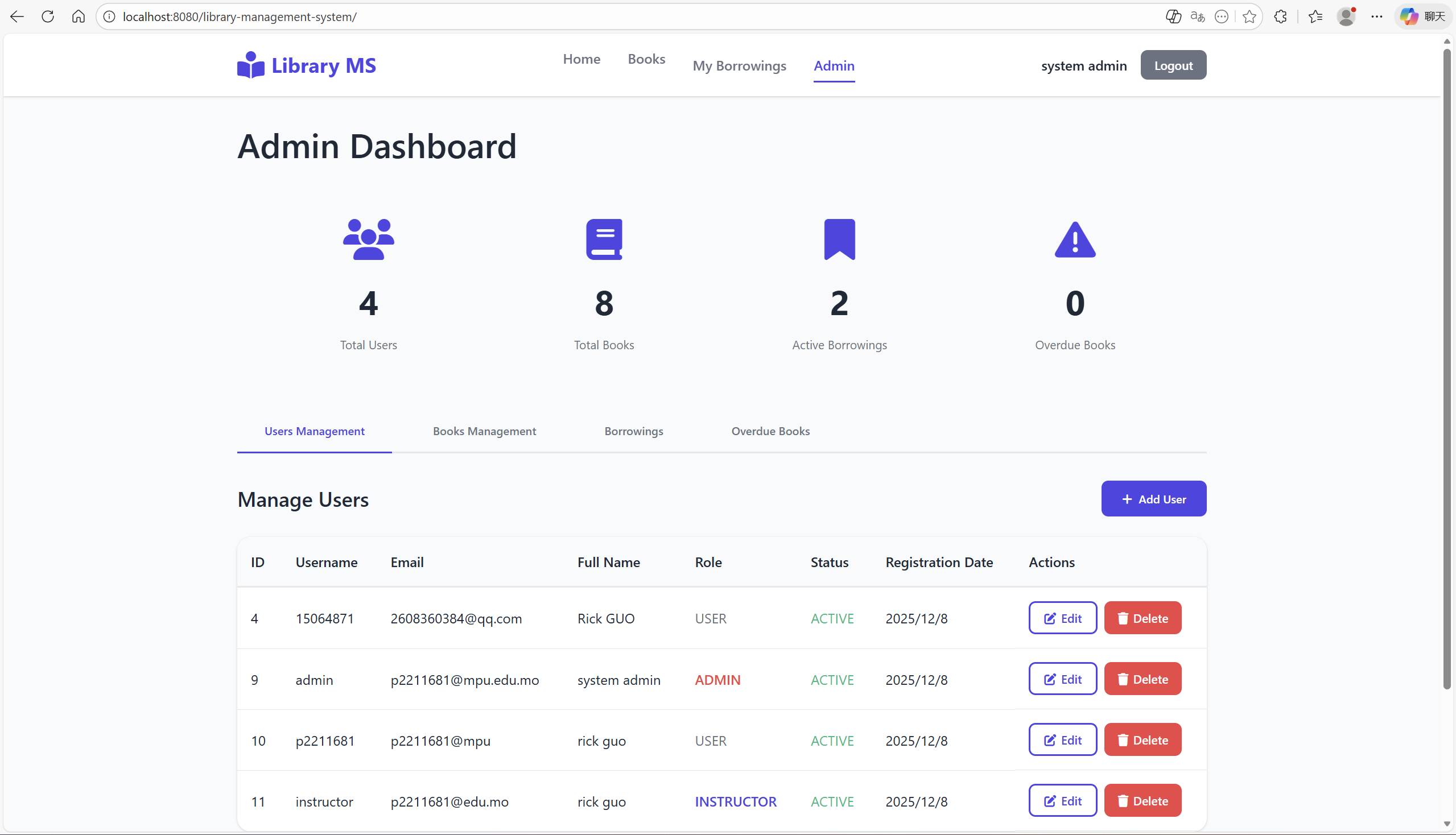
## Register Page



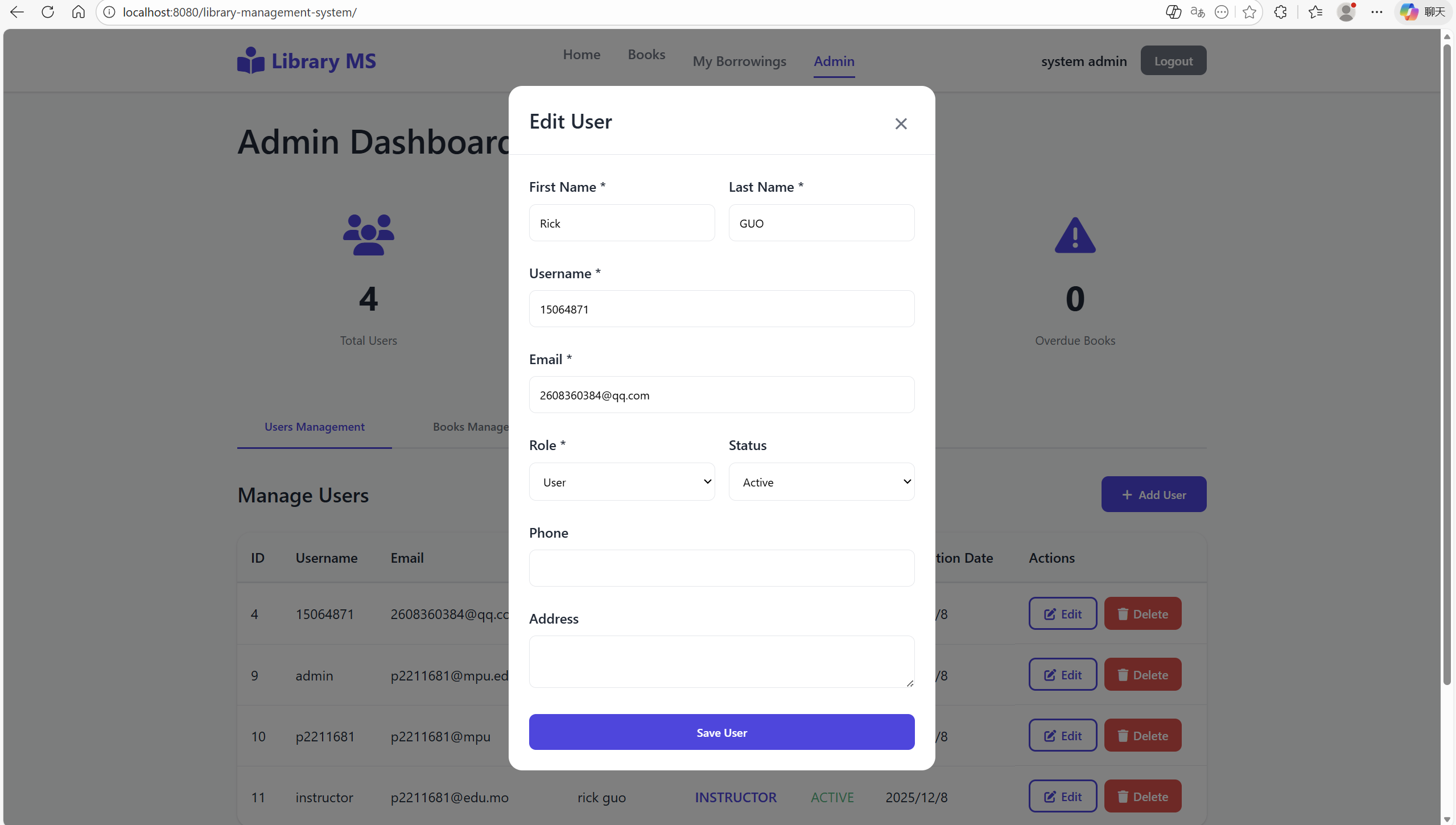
## Set User Permission Page(XAMPP PHPmyAdmin)



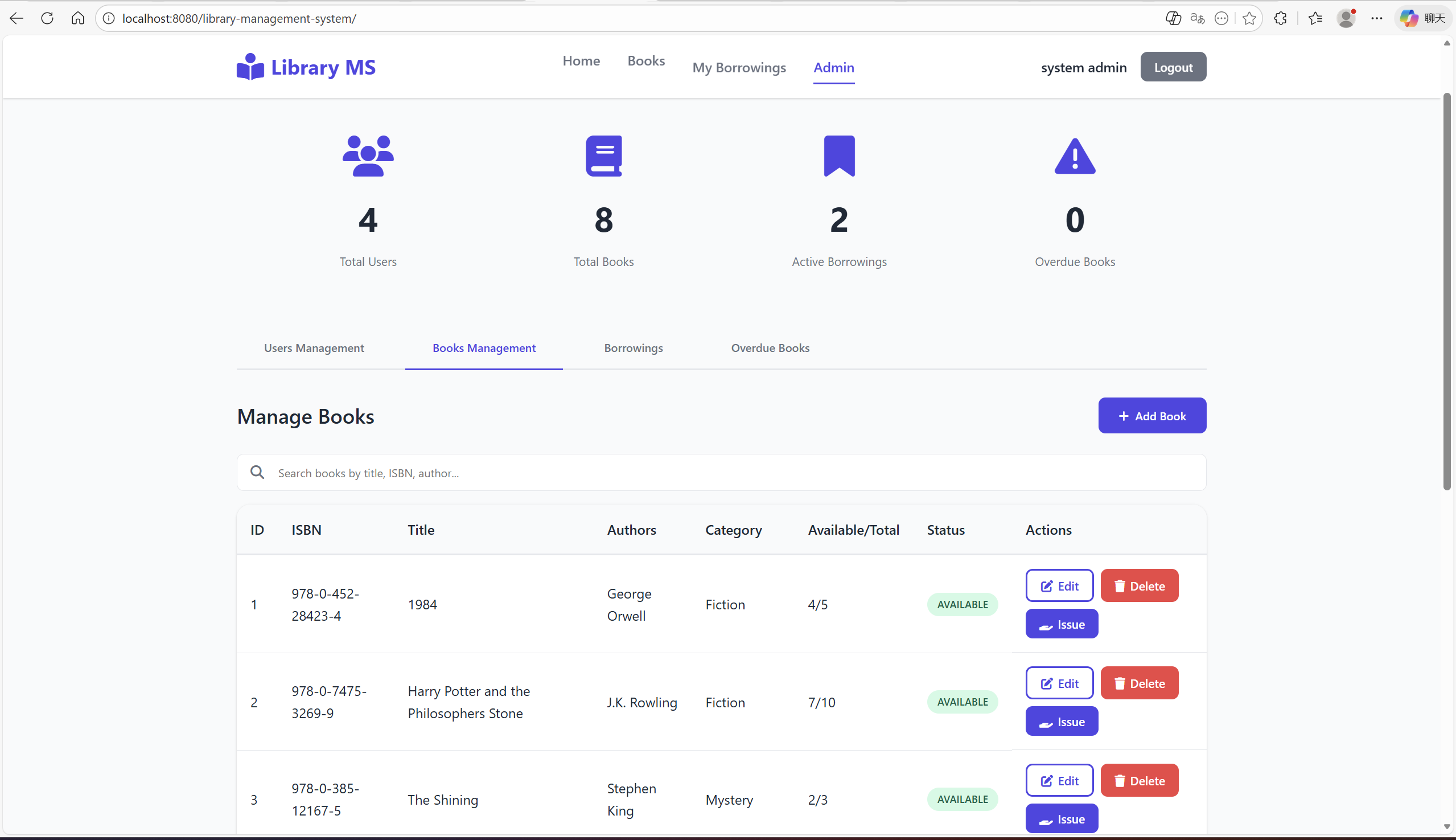
## Admin Page(account: admin password: admin123)



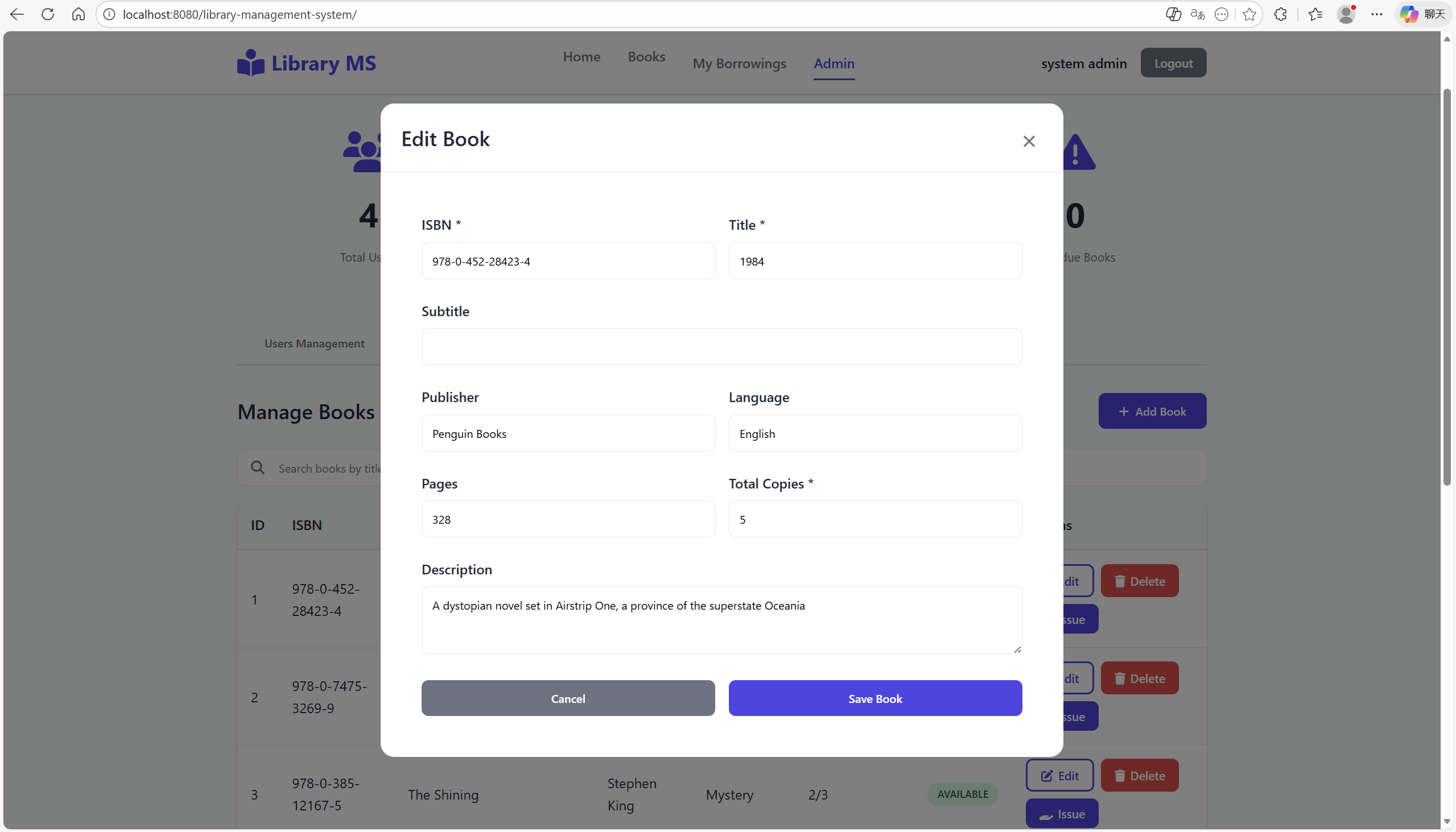
## Admin Page (User Info and Editing and deleting)



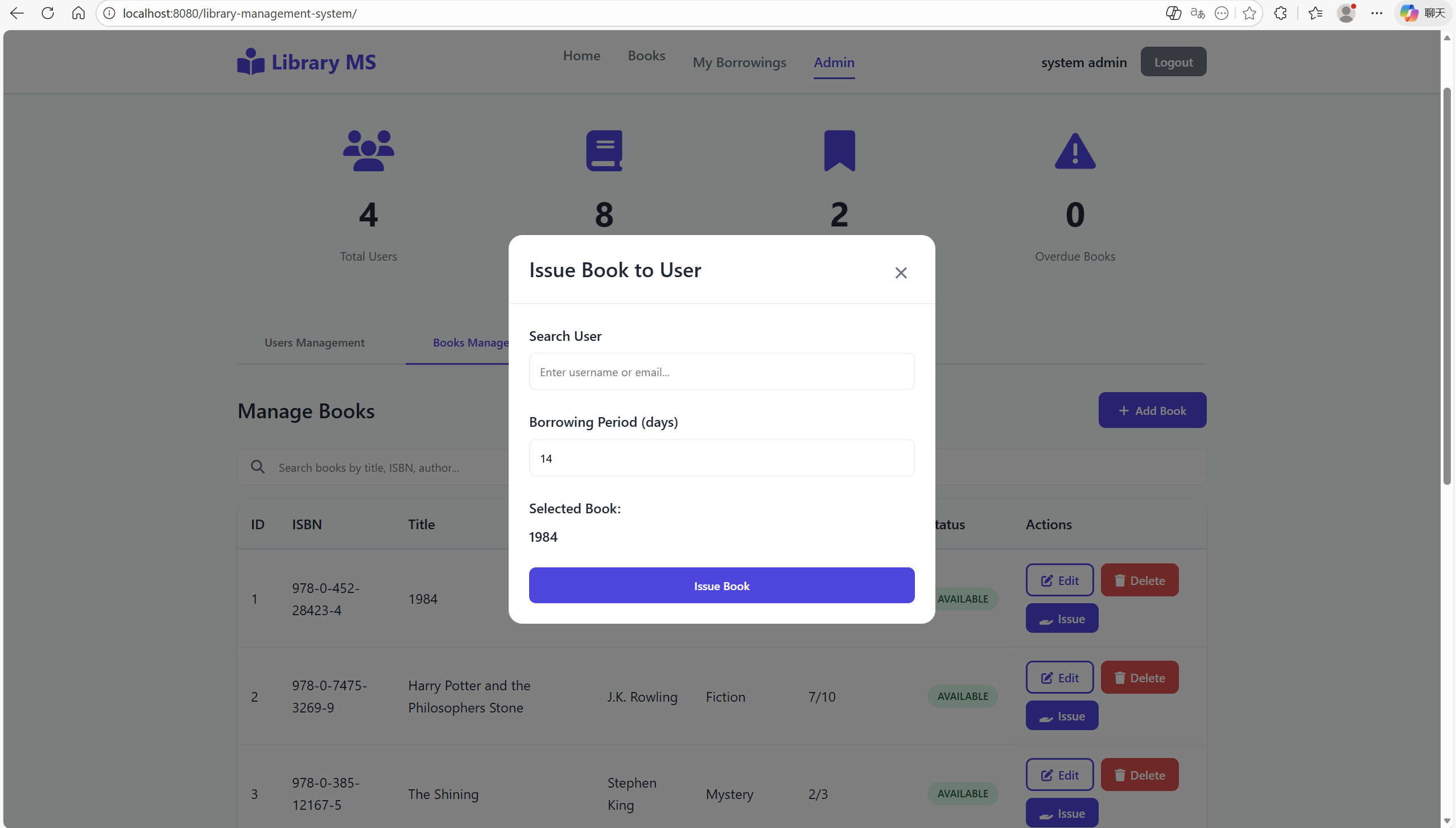
## Admin Page (Book Info page)



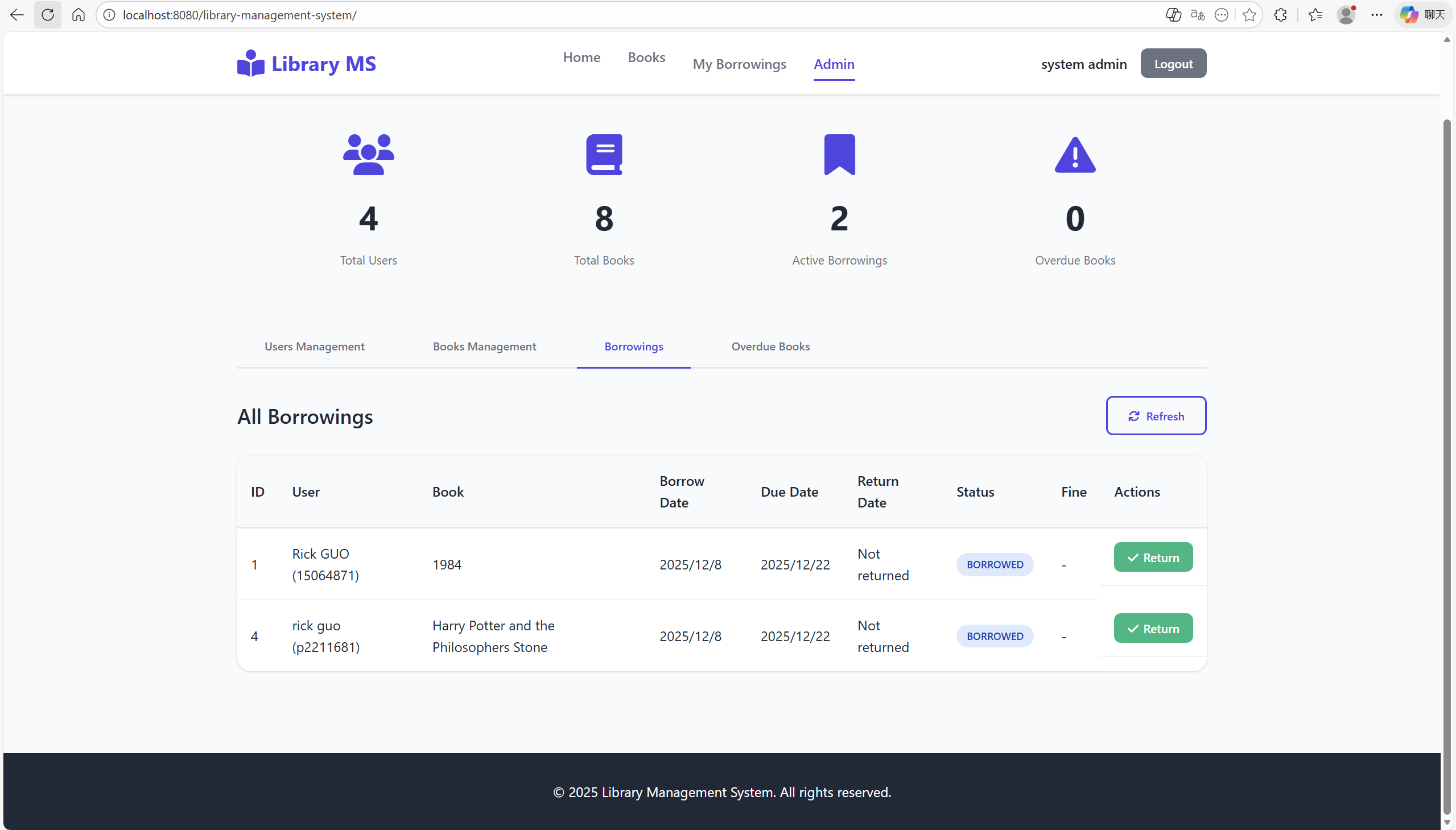
## Admin Page (Book edit page)



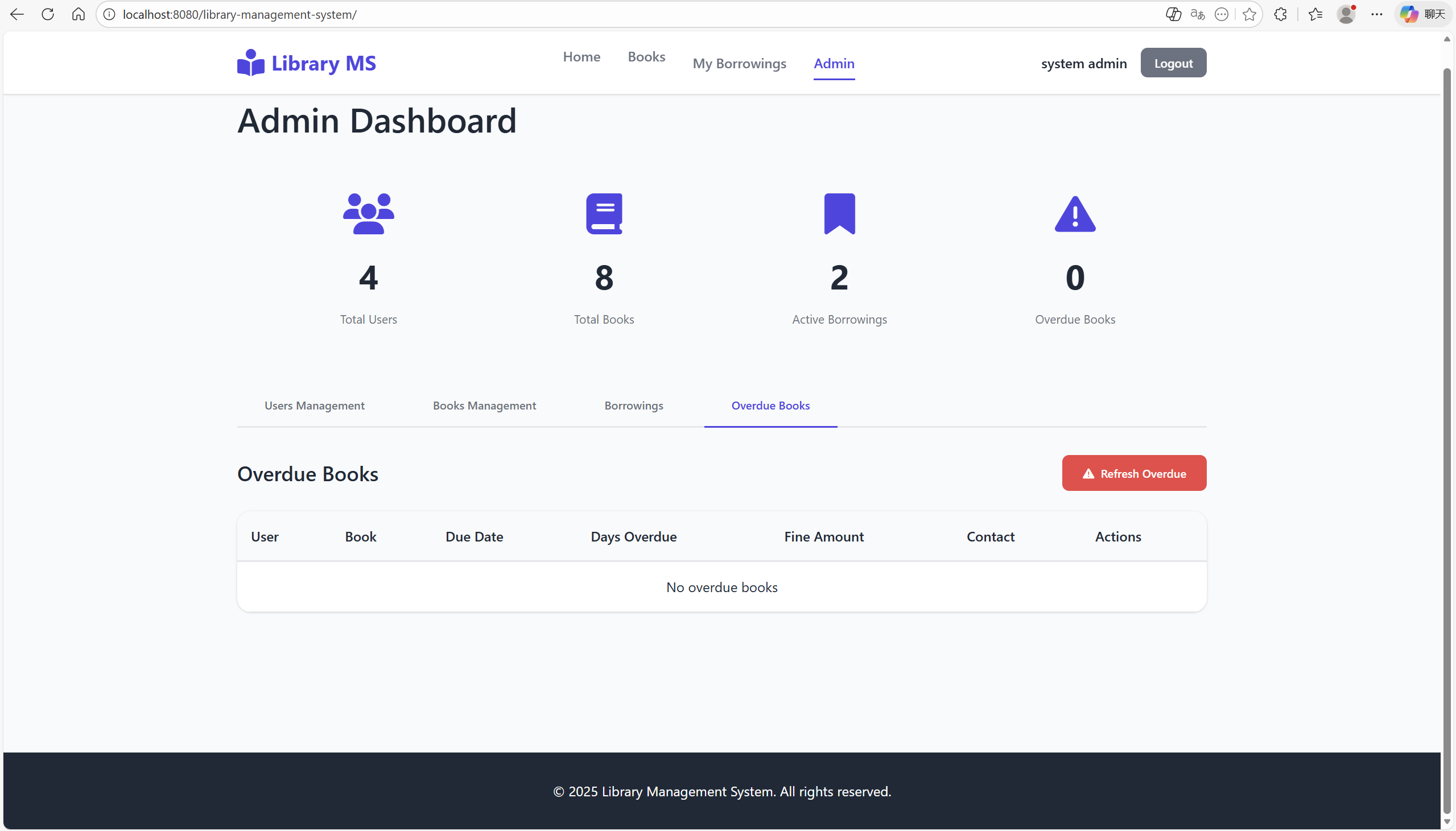
## Admin Page (Book issue page)



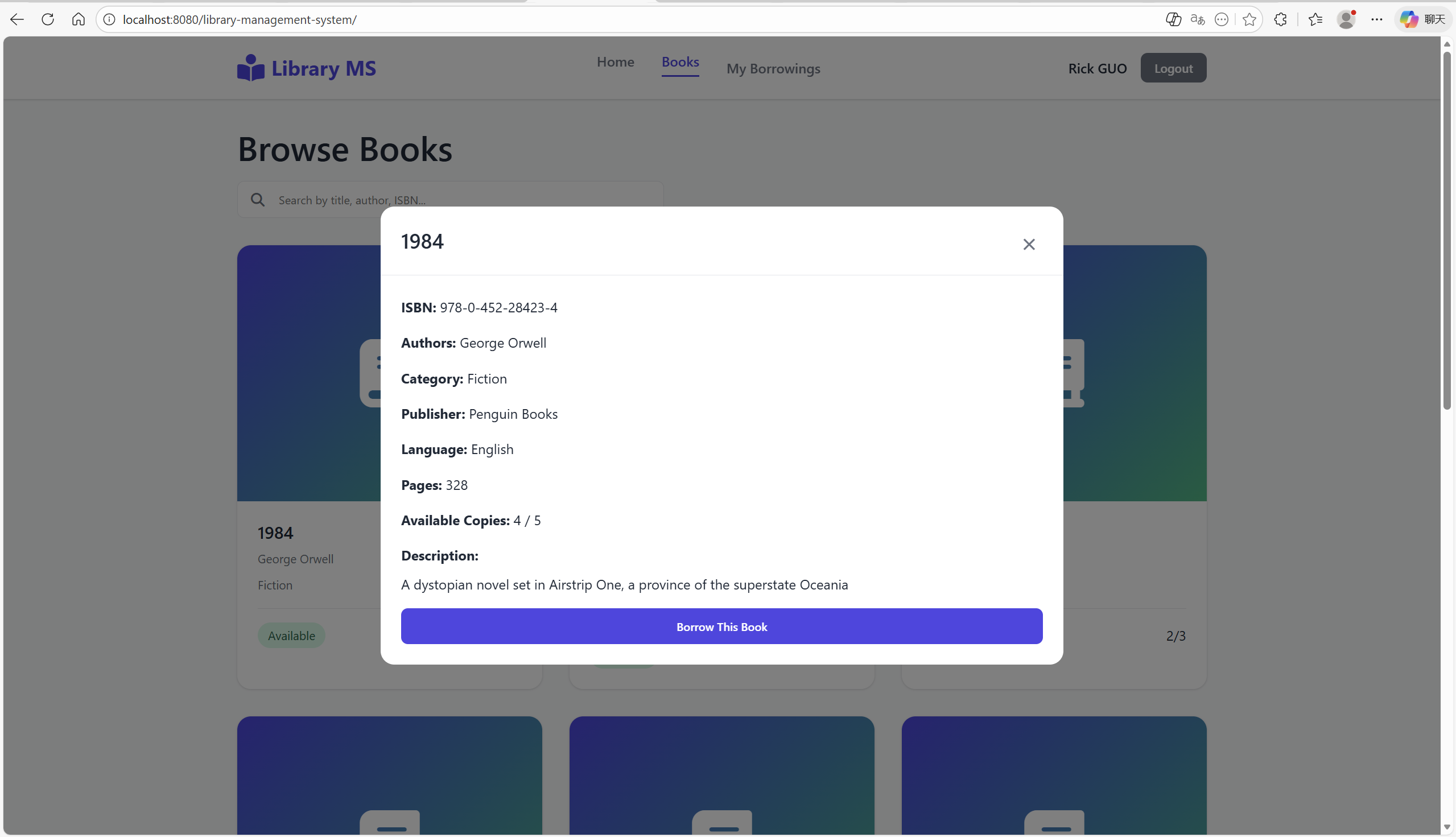
## Admin Page (Book Borrowings page)



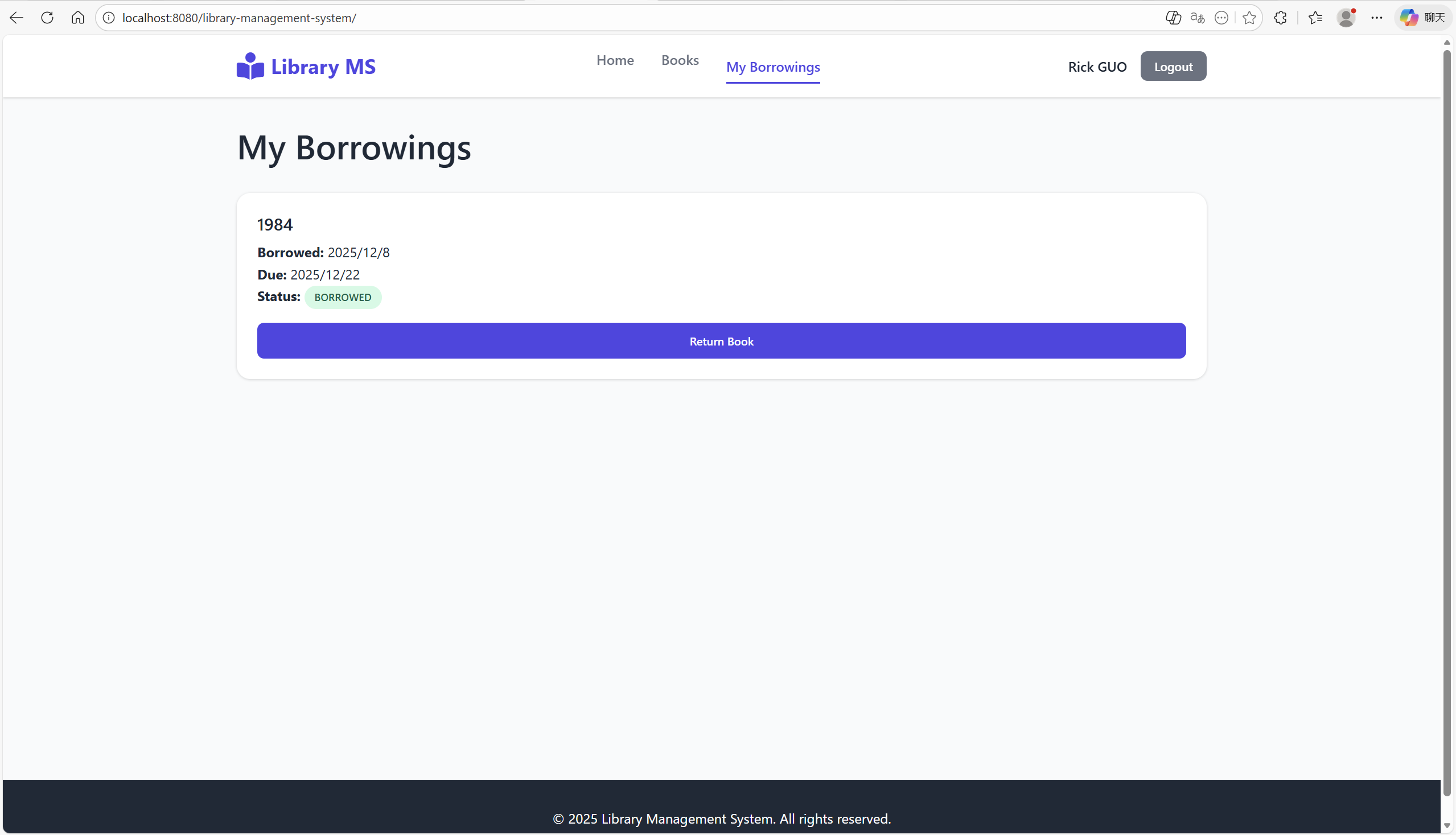
## Admin Page (override Book management page)



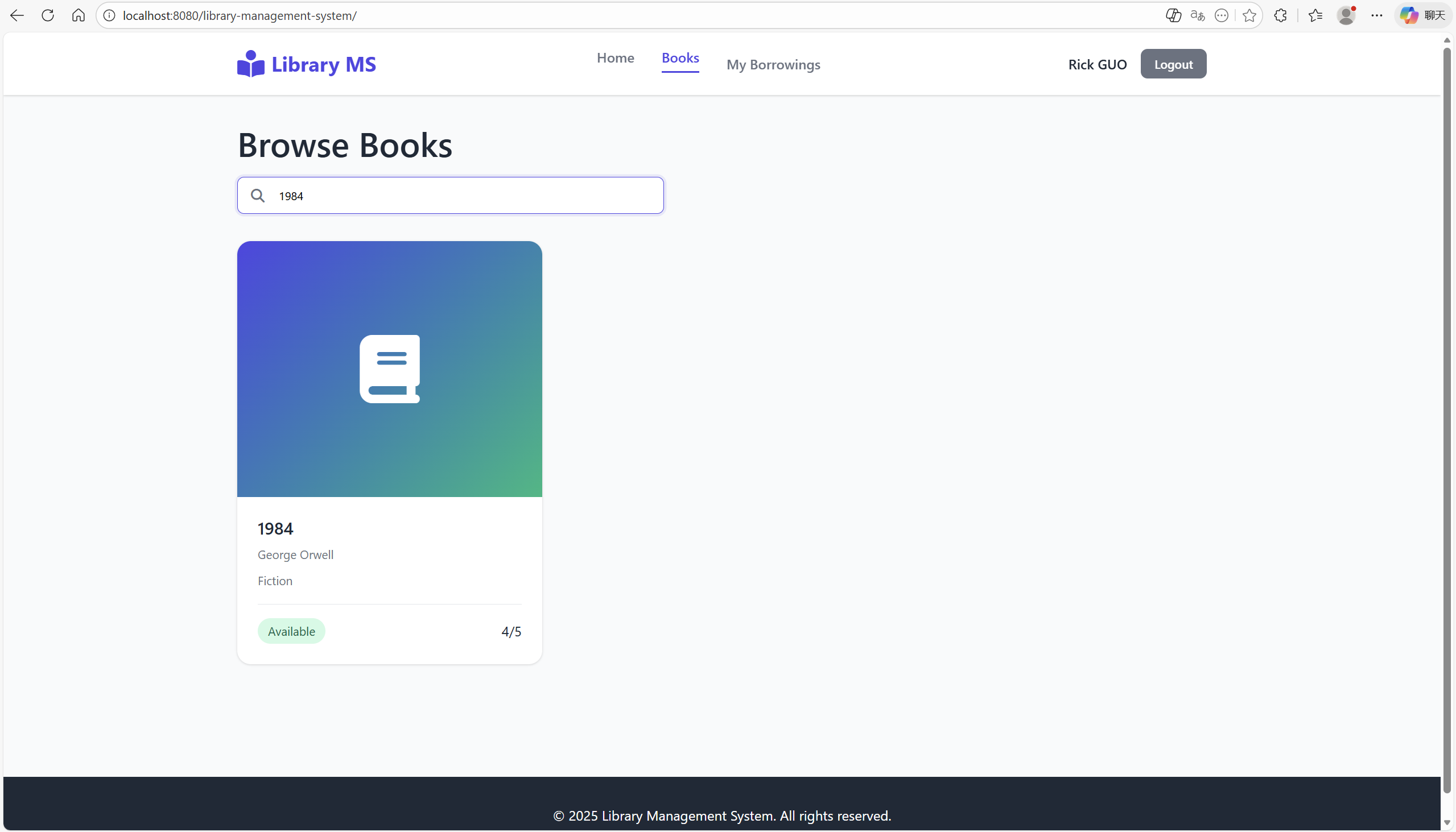
## Books detail page (end-user side)



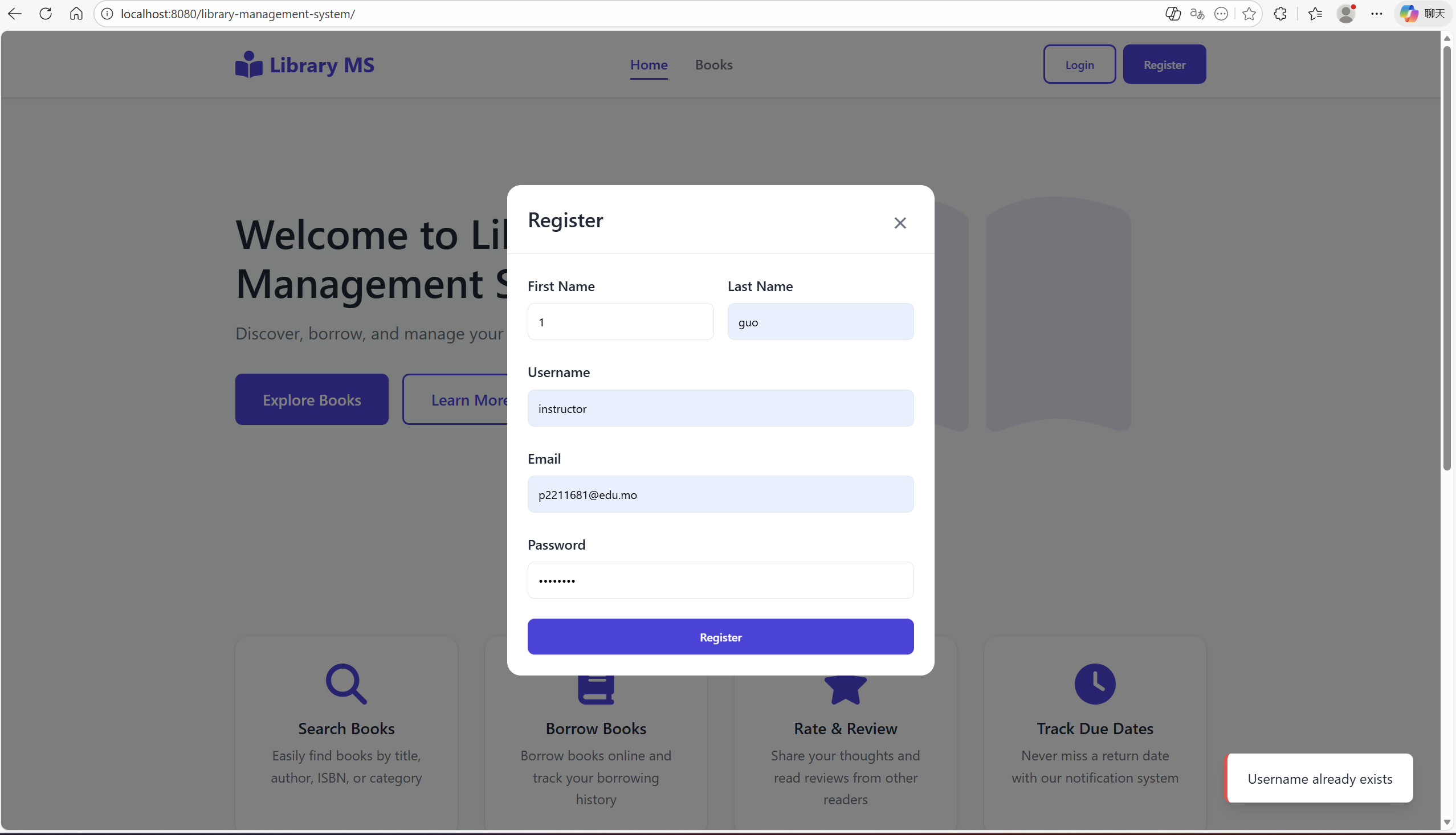
## My borrowings page (end-user side)



## Search page (end-user side)



## Handle same username register (end-user side)



# 6. Appendix:

**Appendix 1: Deployment method**

XAMPP: APACHE ——> active MySQL ——> active——>admin  
IDEA: maven tools ——> lifecycle ——>clean ——>deploy——>target dir.——>.war

.war——>D:\wildfly-38.0.1.Final\standalone\deployments

Click——>"D:\wildfly-38.0.1.Final\bin\standalone.bat" ——>visit [Library Management System](http://localhost:8080/library-management-system/)

**Appendix 2: Technologies Used**

| **Category** | **Technology** | **Version** | **Purpose** |
| --- | --- | --- | --- |
| Platform | Jakarta EE | 9.1 | Enterprise framework |
| ORM | Hibernate | 5.6.15 | Object-relational mapping |
| Database | MySQL | 8.0 | Data persistence |
| Server | Apache Tomcat | 10.x | Application server |
| Security | BCrypt | 0.4 | Password hashing |
| Security | JWT | 0.11.5 | Token authentication |
| JSON | Gson | 2.10.1 | JSON processing |
| Build | Maven | 3.6+ | Dependency management |
| Frontend | HTML5/CSS3/JS | - | User interface |