**Library Management System**

**By:**

**Ammar Hyder (22K-4816)**

**Ausaja Hussain (22K-5186)**

# Acknowledgement

I would like to express my sincere gratitude to my instructors who guided me throughout my semester, providing necessary information which was crucial for the development of this project. I also appreciate the support from my peers, who helped me in incorporating various modules/functionalities. Finally, I am grateful to the online resources and documentation that assisted me in developing this project.

# Abstract

This Library Management System is a web-based application for managing library operations for both normal users and admin. It is designed using Spring Boot for the back-end development, Thymeleaf for the front-end, and is integrated with an Oracle database for data persistence and incorporates a rich set of features including user authentication, book reservation management, and admin functionalities. The system enables users to reserve books, manage their profiles, and perform various tasks like canceling or viewing reservations. Admin users have additional capabilities to manage all users, reservations, and view overdue items. The project aims to provide a simple yet efficient platform for managing book reservations, ensuring smooth user interaction and data management.

# Table Of Contents

[**Acknowledgement**](#_m3koft65bdhc) **2**

[**Abstract**](#_ue0om965fg9i) **3**

[**Table Of Contents**](#_867sptg6qjhk) **4**

[**List of Figures**](#_k5tpo832rtu3) **6**

[**1. Chapter 1: Introduction**](#_8hxxfpkyvn29) **7**

[1.1. Overview](#_vuwfgpkw7hvx) 7

[1.2. Problem Statement](#_4gw5pr6rvoay) 7

[1.3. Database Management System](#_4ylgrfp5d86a) 8

[1.4. Oracle](#_acs9f34n40or) 8

[1.5. HTML/Thymeleaf](#_2zb9xo7pb6da) 8

[1.6. JAVA Connections](#_mfwf3ct15yp4) 9

[**2. Chapter 2: Requirements Specification**](#_rsd95zd2ibet) **10**

[2.1. Overall Description](#_9iqblcit63dw) 10

[2.2. Specific Requirements](#_uvsd4voboe15) 10

[2.2.1. Software Requirements](#_wb4apybyvzue) 11

[2.2.2. Hardware Requirements](#_20dotx4o6ful) 12

[2.2.3. Technology](#_vq03b3c1zzse) 12

[**3. Chapter 3: Detailed Design**](#_r036v5dadew3) **13**

[3.1. System Design](#_pqeghoiuwjl4) 13

[3.2. Entity Relationship Diagram](#_t1yuljggvll6) 14

[3.3. Relational Schema](#_wd9gyz1qvgtn) 14

[3.4. Description of Tables](#_g26pehy8k5e0) 15

[**4. Chapter 4: Implementation**](#_a3hve4ko3ir8) **18**

[4.1. Modules and their Roles](#_brxc3fpweuxu) 18

[4.2. Triggers and Stored Procedures](#_orku3g5ebrns) 39

[4.3. Result](#_383o79irvbpu) 42

[**5. Chapter 5: Testing**](#_p0hk8e2risw) **44**

[5.1. Software Testing](#_hax0c66974yf) 44

[5.2. Module Testing and Integration](#_1d1ujat5ya1g) 44

[5.3. Limitations](#_iji3k89m6bjm) 45

[**6. Chapter 6: Snapshots**](#_u7ryq9w13fc5) **46**

[6.1. Login Page](#_jut77l3pqup1) 46

[6.2. Registration Page](#_rd3rebkamf1t) 46

[6.3. Admin Home Page](#_tyzu4viqlngm) 47

[6.4. Manage Users](#_x6fagwcdryuy) 47

[6.5. Manage Books](#_tgc54qup0j6d) 48

[6.6. View/Edit Books](#_umrv4ljrw5sf) 48

[6.7. Edit User](#_kpsdgjhbi6r6) 49

[6.8. View Reservation](#_9coi9cxcz3w0) 49

[6.9. OverDue](#_y2cdm8lxy0e3) 50

[6.10. Manage Fine](#_xxrmk961id7) 50

[6.11. User Home](#_4pfb1b1yxxmy) 51

[6.12. View Profile](#_idzhslb547wq) 51

[6.13. Edit Profile](#_juuj67qw14tk) 52

[6.14. Borrow Books](#_xxh5c2gclaym) 52

[6.15. Reserved Books](#_rl9z3xgutm3g) 53

[6.16. Reserve Books](#_xkeyuzcd3dyt) 53

[6.17. Return Books](#_t3mg6xlvj4n1) 54

[6.18. Check/Pay Fine](#_p3s3034q9m59) 54

[**7. Chapter 7: Conclusion**](#_ndco054la28s) **55**

[**8. Chapter 8: Future Enhancements**](#_u1xyne7uqz5t) **56**

[**References**](#_5cqn7ox6djwq) **57**

# List of Figures

| **Figure No.** | **Figure Name** | **Page No.** |
| --- | --- | --- |
| **3.1** | ER Diagram of Library Management System | 14 |
| **3.2** | Relational Schema of Library Management System | 15 |

# Chapter 1: Introduction

## Overview

This project is designed to develop a web-based application for managing book reservations in a library. The application allows users to create accounts, log in, reserve books, and manage their reservations. Administrators have additional privileges to manage users, view and modify reservations, and handle overdue items.

The system is built using **Spring Boot** for the back-end and **Thymeleaf** for the front-end. It integrates with an **Oracle Database** to manage and persistent data related to users, books, reservations, and genres. This project aims to streamline the reservation process, reduce administrative overhead, and provide users with an intuitive and efficient platform for managing their library activities.

## Problem Statement

Managing book reservations in a library setting can be a complex and error-prone process when handled manually, especially as the number of users and reservations grows. Traditional methods of tracking availability, ensuring timely cancellations, and handling user requests can lead to miscommunication, inefficiencies, and delays. This project aims to address these issues by developing a web-based system that automates the entire reservation process. By digitizing the booking, cancellation, and management workflows, the system provides a seamless experience for users and administrators alike, reducing human error and improving overall efficiency. The goal is to create a reliable, user-friendly platform that ensures easy access to reservation data while offering enhanced control and visibility for library staff.

## Database Management System

A Database Management System (DBMS) is crucial for efficiently managing and organizing large volumes of data. In this project, Oracle Database has been chosen as the DBMS due to its robustness, scalability, and support for complex queries and transaction management. The database stores essential information such as user credentials, reservations, books, and genres. The DBMS also supports the use of stored procedures and triggers, which automate certain database operations and ensure that business rules are enforced consistently.

## Oracle

Oracle Database is used as the underlying database management system for this project due to its proven reliability, scalability, and performance in handling large volumes of data. In this project, Oracle is utilized to store critical information, including user data, book reservations, and genres.

The database's relational nature allows for efficient data organization and querying, which is crucial for the reservation system's functionality. Using Oracle’s JDBC (Java Database Connectivity) API, the application can easily connect to the database, perform queries, and retrieve data as needed. Additionally, Oracle supports stored procedures and triggers, which are implemented in this project to automate certain tasks like reservation updates and status changes.

## HTML/Thymeleaf

HTML forms the backbone of the front-end structure, providing the foundation for displaying content to the user. In this project, Thymeleaf is utilized as the template engine to dynamically render HTML content. Thymeleaf allows the integration of server-side logic with HTML, enabling the display of dynamic data and interactive web pages.

Thymeleaf's seamless integration with Spring Boot simplifies the development process by automatically binding HTML elements with server-side objects, ensuring that the application can efficiently serve dynamic content to users. By using Thymeleaf with Spring Boot, the project achieves a clean separation of concerns between the presentation layer and business logic, enhancing maintainability and scalability.

## JAVA Connections

Java is primarily used for establishing connections between the application and the Oracle database, as well as facilitating communication between different layers of the application. Java provides robust support for database interaction via JDBC (Java Database Connectivity), enabling the application to perform operations like querying and updating the database.

JDBC is used to manage the connections between the Java application and the Oracle database. The DataSource object is configured to establish a connection to the Oracle database using the connection URL, username, and password, as defined in the application.properties file. The JdbcTemplate provided by Spring is then used to simplify interaction with the database.

# Chapter 2: Requirements Specification

## Overall Description

The project aims to develop a web-based application that streamlines the process of managing book reservations within a library or similar environment. It allows users to create accounts, log in, make reservations, and manage their profile and reservations. Additionally, admin has access to a set options that enable them to oversee user accounts, reservations, and other library data. The application is designed to provide an efficient, user-friendly interface where both regular users and admin can perform tasks easily.

The system integrates with an Oracle Database to manage the data, ensuring reliability and security. The application employs Spring Boot for the back-end, providing a robust framework for handling requests and processing business logic, while Thymeleaf is used for rendering dynamic web pages, ensuring a smooth and responsive user experience. The overall design is intended to be simple yet powerful, providing a fully functional library reservation system.

## Specific Requirements

The specific requirements of Library Management System are as follows:

* **For Users:**
  + **User Registration and Authentication:** users can register/create accounts, log on to the system, and log out of the system.
  + **User Profile Management:** users can view and update their profile information.
  + **Search Functionality:** users can search books according to book title, author etc. for borrowing/reservation.
  + **Book Reservation:** users can reserve a specific book to pick-up later (but reservation gets cancel if user does not pick up the book in a certain period of time)
  + **Reservation Cancellation:** users can cancel their reservations in a specific time frame (before their reservation expires)
  + **Fine Payment:** users can pay fine for returning a book late.
* **For Admin:**
  + **Admin Authentication:** admins can log on to the system, where it is authenticated whether or not admin credentials are valid or not.
  + **User Management:** admins can manage accounts of each user (add new user, update user info. of current user, delete current user, view data of all current users).
  + **Book Management:** admins can manage all the books in inventory (add new book, update information of a book in inventory, delete book currently in inventory, view data of all books in inventory).
  + **Genre Management:** admins can manage all types of genres (add new genres, update info of current genres in system, delete genres, view all kinds of genres).
  + **Fine Management:** admins can view all fines imposed on all users (if there are any).
  + **Reservation Management:** admins can view all the reservations made by all users, and on which specific book.

### Software Requirements

* Java 11+
* Spring Boot 2.x
* Thymeleaf
* Oracle Database 18c+
* Maven

### Hardware Requirements

* Processor: Intel i3 or equivalent
* RAM: 4GB minimum
* Disk Space: 2GB for application and database storage

### Technology

* **Spring Boot**: For back-end development.
* **Thymeleaf**: For front-end template rendering.
* **Oracle Database**: For data storage.
* **Java**: For core application logic.

# Chapter 3: Detailed Design

## System Design

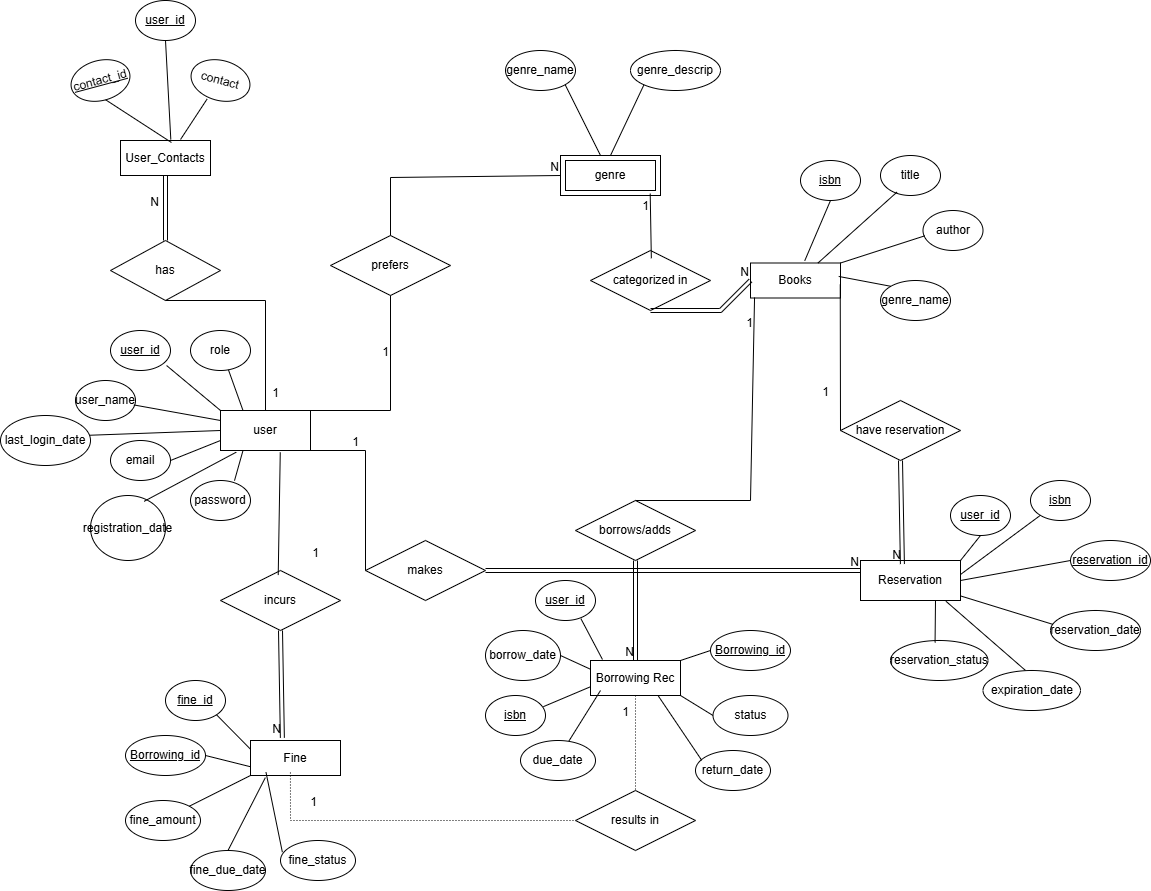
The system design follows a modular and layered architecture, leveraging Spring Boot as the back-end framework and Thymeleaf for front-end rendering. The application consists of several components that interact with each other to fulfill the required functionality:

1. **Controller Layer**: This layer handles HTTP requests and maps them to appropriate services. Some important controllers include:
   * **UserController**:
   * **FineController**:
   * **ReservationController**:
2. **Service Layer**: This layer contains the business logic of the application. It acts as a bridge between the controllers and the data access layer, ensuring that all operations adhere to the defined business rules.
3. **Data Access Layer**: Using Spring's JdbcTemplate, this layer interacts with the Oracle database to perform CRUD operations. The data source configuration is managed in the DatabaseConfig class.
4. **Database**: The Oracle database is designed with normalized tables to ensure data integrity and reduce redundancy. The tables include Users, Books, Reservations, Genres etc., which are linked through foreign key relationships.
5. **View Layer**: The front-end is built with Thymeleaf templates, providing dynamic HTML pages that adapt based on user roles and actions.

This design ensures separation of concerns, making the system easier to maintain and extend. Each layer performs a specific role, and their interaction is governed by well-defined interfaces, promoting modularity and scalability.

## Entity Relationship Diagram

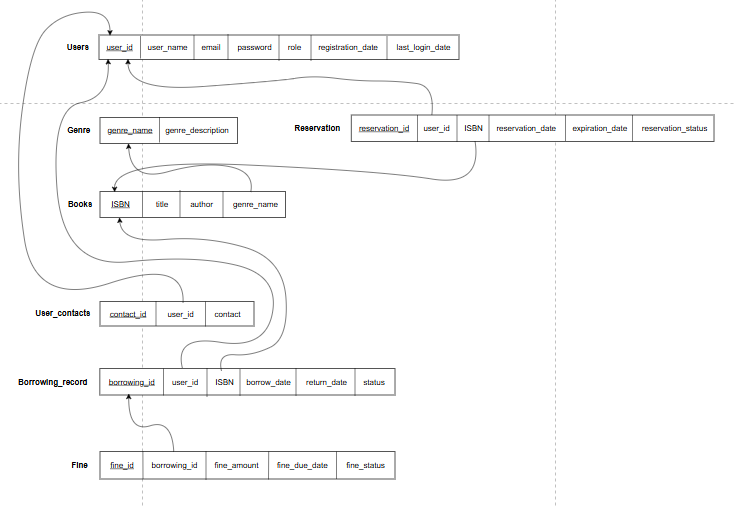
The database includes entities such as Users, User\_Contacts, Fine, Borrowing\_Record, Reservations, Books, and Genres, with relationships modeled using foreign keys to link them.



**Fig 3.1. ER Diagram of Library Management System**

## Relational Schema

The schema includes tables for recording Users, User\_Contacts, Fine, Borrowing\_Record, Reservations, Books, and Genres with it’s necessary attributes.



**Fig 3.2. Relational Schema of Library Management System**

## Description of Tables

The database includes the following tables:

1. **Users Table:**
   1. **user\_id:** Primary key, a unique identifier for each user.
   2. **user\_name:** The name of the user.
   3. **password:** The user's password for authentication.
   4. **email:** The user's email address.
   5. **role:** The role of the user (e.g., admin, user).
   6. **registration\_date:** The date the user registered in the system.
   7. **last\_login\_date:** The last time the user logged into the system.
2. **Genre Table:**
   1. **genre\_name:** Primary key, the unique name of the genre.
   2. **genre\_description:** A brief description of the genre.
3. **Books:**
   1. **ISBN:** Primary key, a unique identifier for each book (International Standard Book Number).
   2. **title:** The title of the book.
   3. **author:** The name of the book's author.
   4. **genre\_name:** Foreign key referencing the genre table, indicating the genre of the book.
4. **User\_Contacts Table:**
   1. **contact\_id:** Primary key, a unique identifier for each contact entry.
   2. **user\_id:** Foreign key referencing the users table, indicating which user the contact belongs to.
   3. **contact:** The contact details (e.g., phone number, alternate email).
5. **Borrowing\_Record Table:**
   1. **borrowing\_id:** Primary key, a unique identifier for each borrowing transaction.
   2. **user\_id:** Foreign key referencing the users table, indicating the user borrowing the book.
   3. **ISBN:** Foreign key referencing the books table, indicating the borrowed book.
   4. **borrow\_date:** The date the book was borrowed.
   5. **due\_date:** The date the book is due for return.
   6. **return\_date:** The actual date the book was returned.
   7. **status:** The status of the borrowing (e.g., active, returned, overdue).
6. **Fine Table:**
   1. **fine\_id:** Primary key, a unique identifier for each fine.
   2. **borrowing\_id:** Foreign key referencing the borrowing\_record table, linking the fine to a specific borrowing.
   3. **fine\_amount:** The amount of the fine imposed.
   4. **fine\_due\_date:** The date by which the fine should be paid.
   5. **fine\_status:** The status of the fine (e.g., unpaid, paid).
7. **Reservation Table:**
   1. **reservation\_id**: Primary key, a unique identifier for each reservation.
   2. **user\_id**: Foreign key referencing the users table, indicating the user making the reservation.
   3. **ISBN**: Foreign key referencing the books table, indicating the reserved book.
   4. **reservation\_date**: The date the reservation was made.
   5. **expiration\_date**: The date the reservation expires.
   6. **reservation\_status**: The status of the reservation (e.g., active, expired, fulfilled).

# Chapter 4: Implementation

## Modules and their Roles

* + 1. **BookService.java:**

@Service

public class BookService {

private final JdbcTemplate jdbcTemplate;

@Autowired

public BookService(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public boolean returnBook(String isbn, int userId) {

// Check if the user has borrowed the book

String checkBorrowingRecordSql = "SELECT borrowing\_id, borrow\_date, due\_date FROM borrowing\_record WHERE user\_id = ? AND ISBN = ? AND status = 'BORROWED'";

Map<String, Object> record = jdbcTemplate.queryForMap(checkBorrowingRecordSql, userId, isbn);

if (record == null || record.isEmpty()) {

return false;

}

Timestamp borrowTimestamp = (Timestamp) record.get("borrow\_date");

Timestamp dueTimestamp = (Timestamp) record.get("due\_date");

LocalDate borrowDate = borrowTimestamp.toLocalDateTime().toLocalDate();

LocalDate dueDate = dueTimestamp.toLocalDateTime().toLocalDate();

LocalDate returnDate = LocalDate.now();

long daysLate = ChronoUnit.DAYS.between(dueDate, returnDate);

double fine = 0.0;

if (daysLate > 0) {

fine = daysLate \* 1.0; // You can change this rate as needed

}

// Update the borrowing record status to 'RETURNED' and add return date

String updateBorrowingRecordSql = "UPDATE borrowing\_record SET status = 'RETURNED', return\_date = ? WHERE user\_id = ? AND ISBN = ? AND status = 'BORROWED'";

int rowsUpdated = jdbcTemplate.update(updateBorrowingRecordSql, Date.valueOf(returnDate), userId, isbn);

if (rowsUpdated > 0) {

// Insert a record into the fine table if a fine is applicable

if (fine > -1) {

// Get the next fine\_id

String getNextFineIdSql = "SELECT NVL(MAX(fine\_id), 0) + 1 AS next\_fine\_id FROM fine";

Integer nextFineId = jdbcTemplate.queryForObject(getNextFineIdSql, Integer.class);

// Insert the fine record

String insertFineSql = "INSERT INTO fine (fine\_id, borrowing\_id, fine\_amount, fine\_due\_date, fine\_status) VALUES (?, ?, ?, ?, ?)";

LocalDate fineDueDate = returnDate.plusDays(7); // Set fine due date (7 days after return date)

jdbcTemplate.update(insertFineSql, nextFineId, record.get("borrowing\_id"), fine, Date.valueOf(fineDueDate), "PENDING");

}

// Increase the available count of the book

String updateBookCountSql = "UPDATE bookcount SET available\_count = available\_count + 1 WHERE ISBN = ?";

jdbcTemplate.update(updateBookCountSql, isbn);

return true;

}

return false;

}

public List<Map<String, Object>> getBorrowedBooksByUser(int userId) {

String sql = "SELECT b.ISBN, b.title, b.author, br.borrow\_date, br.due\_date, br.status " +

"FROM books b " +

"JOIN borrowing\_record br ON b.ISBN = br.ISBN " +

"WHERE br.user\_id = ? AND br.status = 'BORROWED'";

return jdbcTemplate.queryForList(sql, userId);

}

public boolean borrowBook(String isbn, int userId) {

String checkBorrowingRecordSql = "SELECT COUNT(\*) FROM borrowing\_record WHERE user\_id = ? AND ISBN = ? AND status = 'BORROWED'";

int count = jdbcTemplate.queryForObject(checkBorrowingRecordSql, new Object[]{userId, isbn}, Integer.class);

if (count > 0) {

// The user already borrowed the book, can't borrow again

return false;

}

String fetchMaxBorrowingIdSql = "SELECT MAX(borrowing\_id) FROM borrowing\_record";

Integer lastBorrowingId = jdbcTemplate.queryForObject(fetchMaxBorrowingIdSql, Integer.class);

// Generate the next borrowing ID (last + 1)

int nextBorrowingId = (lastBorrowingId == null) ? 1 : lastBorrowingId + 1;

LocalDate borrowDate = LocalDate.now();

LocalDate dueDate = borrowDate.plusDays(14);

// Insert new borrowing record into the borrowing\_record table

String insertBorrowingRecordSql = "INSERT INTO borrowing\_record (borrowing\_id, user\_id, ISBN, borrow\_date, due\_date, status) VALUES (?, ?, ?, ?, ?, ?)";

int rowsInserted = jdbcTemplate.update(insertBorrowingRecordSql, nextBorrowingId, userId, isbn, Date.valueOf(borrowDate), Date.valueOf(dueDate), "BORROWED");

if (rowsInserted > 0) {

String updateBookCountSql = "UPDATE bookcount SET available\_count = available\_count - 1 WHERE ISBN = ?";

jdbcTemplate.update(updateBookCountSql, isbn);

return true;

}

return false;

}

public List<dBook> getAllBooks() {

String sql = "SELECT b.ISBN, b.title, b.author, b.genre\_name, bc.available\_count " +

"FROM books b JOIN bookcount bc ON b.ISBN = bc.ISBN";

return jdbcTemplate.query(sql, new RowMapper<dBook>() {

@Override

public dBook mapRow(ResultSet rs, int rowNum) throws SQLException {

dBook book = new dBook();

book.setISBN(rs.getString("ISBN"));

book.setTitle(rs.getString("title"));

book.setAuthor(rs.getString("author"));

book.setGenre\_name(rs.getString("genre\_name"));

book.setAvailable\_count(rs.getInt("available\_count"));

return book;

}

});

}

public List<dBook> getAlldBooks() {

// SQL query to join Book and BookCount on ISBN

String sql = "SELECT b.ISBN, b.title, b.author, b.genre\_name, bc.total\_count, bc.available\_count " +

"FROM books b " +

"JOIN bookcount bc ON b.ISBN = bc.ISBN";

// RowMapper to map each row of the result set to a dBook object

RowMapper<dBook> rowMapper = (rs, rowNum) -> {

dBook book = new dBook();

book.setISBN(rs.getString("ISBN"));

book.setTitle(rs.getString("title"));

book.setAuthor(rs.getString("author"));

book.setGenre\_name(rs.getString("genre\_name"));

book.setTotal\_count(rs.getInt("total\_count"));

book.setAvailable\_count(rs.getInt("available\_count"));

return book;

};

return jdbcTemplate.query(sql, rowMapper);

}

public boolean bookExists(String isbn) {

String sql = "SELECT COUNT(\*) FROM books WHERE ISBN = ?";

Integer count = jdbcTemplate.queryForObject(sql, Integer.class, isbn);

return count != null && count > 0;

}

public boolean addBookCount(BookCount bookCount) {

String checkSql = "SELECT COUNT(\*) FROM bookcount WHERE ISBN = ?";

Integer count = jdbcTemplate.queryForObject(checkSql, Integer.class, bookCount.getISBN());

if (count != null && count > 0) {

return false;

}

// SQL to insert a new book count

String insertSql = "INSERT INTO bookcount (ISBN, total\_count, available\_count) VALUES (?, ?, ?)";

return jdbcTemplate.update(insertSql, bookCount.getISBN(), bookCount.getTotal\_count(), bookCount.getAvailable\_count()) > 0;

}

public List<Map<String, Object>> getAllBookCount() {

String sql = "SELECT bc.ISBN, bc.total\_count, bc.available\_count " +

"FROM bookcount bc";

return jdbcTemplate.queryForList(sql);

}

public boolean genreExists(String genreName) {

String sql = "SELECT COUNT(\*) FROM genre WHERE genre\_name = ?";

Integer count = jdbcTemplate.queryForObject(sql, Integer.class, genreName);

System.out.println(count);

return count != null && count > 0;

}

public boolean addBook(Book book) {

String sql = "INSERT INTO books (ISBN, title, author, genre\_name) VALUES (?, ?, ?, ?)";

int rowsAffected = jdbcTemplate.update(sql, book.getISBN(), book.getTitle(), book.getAuthor(), book.getGenre\_name());

return rowsAffected > 0;

}

public Book getBookByIsbn(String isbn) {

String sql = "SELECT \* FROM books WHERE ISBN = ?";

try {

return jdbcTemplate.queryForObject(sql, (rs, rowNum) -> {

Book book = new Book();

book.setISBN(rs.getString("ISBN"));

book.setTitle(rs.getString("title"));

book.setAuthor(rs.getString("author"));

book.setGenre\_name(rs.getString("genre\_name"));

return book;

}, isbn);

} catch (EmptyResultDataAccessException e) {

System.out.println("No book found with ISBN: " + isbn);

return null;

} catch (Exception e) {

System.out.println("Error fetching book by ISBN: " + e.getMessage());

return null;

}

}

public boolean updateBook(String isbn, Book book) {

System.out.println(book.getGenre\_name());

String sql = "UPDATE books SET title = ?, author = ?, genre\_name = ? WHERE ISBN = ?";

int rowsAffected = jdbcTemplate.update(sql, book.getTitle(), book.getAuthor(), book.getGenre\_name(), isbn);

System.out.println(rowsAffected);

return rowsAffected > 0;

}

public boolean deleteBook(String isbn) {

String sql = "DELETE FROM books WHERE ISBN = ?";

int rowsAffected = jdbcTemplate.update(sql, isbn);

return rowsAffected > 0;

}

}

* + 1. **FineService.java:**

@Service

public class FineService {

private final JdbcTemplate jdbcTemplate;

@Autowired

public FineService(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public List<Fine> getFineByUserId(int userId) {

String sql = "SELECT f.fine\_id, f.borrowing\_id, f.fine\_amount, f.fine\_due\_date, f.fine\_status " +

"FROM fine f " +

"JOIN borrowing\_record br ON f.borrowing\_id = br.borrowing\_id " +

"WHERE br.user\_id = ?";

return jdbcTemplate.query(sql, this::mapRowToFine, userId);

}

public List<Fine> getAllFines() {

String sql = "SELECT f.fine\_id, f.borrowing\_id, f.fine\_amount, f.fine\_due\_date, f.fine\_status, br.user\_id, br.isbn " +

"FROM fine f " +

"JOIN borrowing\_record br ON f.borrowing\_id = br.borrowing\_id";

return jdbcTemplate.query(sql, this::mapRowToFine);

}

public boolean payFineAdmin(int fineId) {

String sql = "UPDATE fine SET fine\_status = 'PAID' WHERE fine\_id = ?";

return jdbcTemplate.update(sql, fineId) > 0;

}

public void deleteFine(int fineId) {

String sql = "DELETE FROM fine WHERE fine\_id = ?";

jdbcTemplate.update(sql, fineId);

}

private Fine mapRowToFine(ResultSet rs, int rowNum) throws SQLException {

Fine fine = new Fine();

fine.setFineId(rs.getInt("fine\_id"));

fine.setBorrowing\_Id(rs.getInt("borrowing\_id"));

fine.setFineAmount(rs.getDouble("fine\_amount"));

fine.setFineDueDate(rs.getDate("fine\_due\_date").toLocalDate());

fine.setFineStatus(rs.getString("fine\_status"));

return fine;

}

}

* + 1. **GenreService.java:**

@Service

public class GenreService {

private final JdbcTemplate jdbcTemplate;

@Autowired

public GenreService(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public List<Map<String, Object>> getAllGenres() {

String sql = "SELECT genre\_name, genre\_description FROM genre";

List<Map<String, Object>> genres = jdbcTemplate.queryForList(sql);

return genres;

}

public boolean genreExists(String genreName) {

String sql = "SELECT COUNT(\*) FROM genre WHERE genre\_name = ?";

Integer count = jdbcTemplate.queryForObject(sql, Integer.class, genreName);

return count != null && count > 0;

}

public boolean addGenre(Genre genre) {

// Check if genre already exists

if (genreExists(genre.getGenre\_Name())) {

return false; // Genre exists, cannot be added

}

// Insert the genre if it doesn't exist

String sql = "INSERT INTO genre (genre\_name, genre\_description) VALUES (?, ?)";

int rowsAffected = jdbcTemplate.update(sql, genre.getGenre\_Name(), genre.getGenre\_Description());

return rowsAffected > 0; // Return true if the genre was added successfully

}

public Genre getGenreByName(String genreName) {

String sql = "SELECT genre\_name, genre\_description FROM genre WHERE genre\_name = ?";

return jdbcTemplate.queryForObject(sql, (rs, rowNum) -> {

Genre genre = new Genre();

genre.setGenre\_Name(rs.getString("genre\_name"));

genre.setGenre\_Description(rs.getString("genre\_description"));

return genre;

}, genreName);

}

}

* + 1. **ReservationService.java:**

@Service

public class ReservationService {

@Autowired

private JdbcTemplate jdbcTemplate;

public List<dBook> getAllBooks() {

String sql = "SELECT b.ISBN, b.title, b.author, g.genre\_name, bc.available\_count " +

"FROM books b " +

"JOIN genre g ON b.genre\_name = g.genre\_name " +

"JOIN bookcount bc ON b.ISBN = bc.ISBN";

return jdbcTemplate.query(sql, (rs, rowNum) -> {

dBook book = new dBook();

book.setISBN(rs.getString("ISBN"));

book.setTitle(rs.getString("title"));

book.setAuthor(rs.getString("author"));

book.setGenre\_name(rs.getString("genre\_name"));

book.setAvailable\_count(rs.getInt("available\_count"));

return book;

});

}

public boolean reserveBook(int userId, String isbn) {

// Check if the book is already reserved by the user

String checkExistingReservationSql = "SELECT COUNT(\*) FROM reservation WHERE user\_id = ? AND ISBN = ? AND reservation\_status = 'PENDING'";

Integer existingReservationCount = jdbcTemplate.queryForObject(checkExistingReservationSql, new Object[]{userId, isbn}, Integer.class);

if (existingReservationCount != null && existingReservationCount > 0) {

return false;

}

// Check if the book is available

String checkAvailabilitySql = "SELECT available\_count FROM bookcount WHERE ISBN = ?";

Integer availableCount = jdbcTemplate.queryForObject(checkAvailabilitySql, new Object[]{isbn}, Integer.class);

if (availableCount == null || availableCount <= 0) {

return false;

}

// Determine the next reservation\_id

String getMaxReservationIdSql = "SELECT COALESCE(MAX(reservation\_id), 0) FROM reservation";

Integer maxReservationId = jdbcTemplate.queryForObject(getMaxReservationIdSql, Integer.class);

int nextReservationId = (maxReservationId != null ? maxReservationId : 0) + 1;

// Insert reservation record

String insertReservationSql = "INSERT INTO reservation (reservation\_id, user\_id, ISBN, reservation\_date, expiration\_date, reservation\_status) " +

"VALUES (?, ?, ?, ?, ?, ?)";

int rowsInserted = jdbcTemplate.update(insertReservationSql,

nextReservationId, // The new reservation\_id

userId,

isbn,

LocalDate.now(),

LocalDate.now().plusDays(7), // Reservation valid for 7 days

"PENDING");

if (rowsInserted > 0) {

String updateBookCountSql = "UPDATE bookcount SET available\_count = available\_count - 1 WHERE ISBN = ?";

jdbcTemplate.update(updateBookCountSql, isbn);

return true;

}

return false;

}

public List<Reservation> getUserReservations(int userId) {

String sql = "SELECT reservation\_id, ISBN, reservation\_date, expiration\_date, reservation\_status " +

"FROM reservation WHERE user\_id = ?";

return jdbcTemplate.query(sql, new Object[]{userId}, this::mapReservation);

}

public void cancelReservation(int reservationId, int userId) {

// Check if the reservation belongs to the user and is still active

String checkReservationSql = "SELECT ISBN FROM reservation WHERE reservation\_id = ? AND user\_id = ? AND reservation\_status = 'PENDING'";

List<String> isbnList = jdbcTemplate.query(checkReservationSql, new Object[]{reservationId, userId},

(rs, rowNum) -> rs.getString("ISBN"));

if (isbnList.isEmpty()) {

throw new IllegalArgumentException("Reservation not found or cannot be canceled.");

}

String isbn = isbnList.get(0);

// Update the reservation status to CANCELED

String cancelReservationSql = "UPDATE reservation SET reservation\_status = 'CANCELED' WHERE reservation\_id = ?";

jdbcTemplate.update(cancelReservationSql, reservationId);

// Increment the book count

String incrementBookCountSql = "UPDATE bookcount SET available\_count = available\_count + 1 WHERE ISBN = ?";

jdbcTemplate.update(incrementBookCountSql, isbn);

}

private Reservation mapReservation(ResultSet rs, int rowNum) throws SQLException {

Reservation reservation = new Reservation();

reservation.setReservation\_id(rs.getInt("reservation\_id"));

reservation.setIsbn(rs.getString("ISBN"));

reservation.setReservation\_date(rs.getDate("reservation\_date").toLocalDate());

reservation.setExpiration\_date(rs.getDate("expiration\_date").toLocalDate());

reservation.setReservation\_status(rs.getString("reservation\_status"));

return reservation;

}

public List<Reservation> getAllReservations() {

String sql = "SELECT reservation\_id, user\_id, ISBN, reservation\_date, expiration\_date, reservation\_status " +

"FROM reservation";

return jdbcTemplate.query(sql, this::mapReservation);

}

public void cancelReservationAsAdmin(int reservationId) {

String checkReservationSql = "SELECT ISBN FROM reservation WHERE reservation\_id = ? AND reservation\_status = 'PENDING'";

List<String> isbnList = jdbcTemplate.query(checkReservationSql, new Object[]{reservationId},

(rs, rowNum) -> rs.getString("ISBN"));

if (isbnList.isEmpty()) {

throw new IllegalArgumentException("Reservation not found or already canceled.");

}

String isbn = isbnList.get(0);

// Update the reservation status to CANCELED

String cancelReservationSql = "UPDATE reservation SET reservation\_status = 'CANCELED' WHERE reservation\_id = ?";

jdbcTemplate.update(cancelReservationSql, reservationId);

// Increment the book count

String incrementBookCountSql = "UPDATE bookcount SET available\_count = available\_count + 1 WHERE ISBN = ?";

jdbcTemplate.update(incrementBookCountSql, isbn);

}

public List<Reservation> getOverdueReservations() {

String sql = "SELECT reservation\_id, user\_id, ISBN, reservation\_date, expiration\_date, reservation\_status " +

"FROM reservation " +

"WHERE expiration\_date < ? AND reservation\_status = 'PENDING'";

return jdbcTemplate.query(sql, new Object[]{LocalDate.now()}, this::mapReservation);

}

}

* + 1. **UserService.java:**

@Service

public class UserService {

private final JdbcTemplate jdbcTemplate;

@Autowired

public UserService(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public User authenticateUser(String username, String password) {

try {

String sql = "SELECT \* FROM users WHERE user\_name = ? AND password = ?";

// Query database for user with matching username and password

return jdbcTemplate.queryForObject(sql, new Object[]{username, password}, (rs, rowNum) -> {

User user = new User();

user.setUserID(rs.getInt("user\_id"));

user.setUsername(rs.getString("user\_name"));

user.setPassword(rs.getString("password"));

user.setEmail(rs.getString("email"));

user.setRole(rs.getString("role"));

user.setRegistrationDate(String.valueOf(rs.getDate("registration\_date")));

user.setLastLoginDate(String.valueOf(rs.getDate("last\_login\_date")));

return user;

});

} catch (Exception e) {

return null;

}

}

public boolean userExists(String username, String email) {

String sql = "SELECT COUNT(\*) FROM users WHERE user\_name = ? OR email = ?";

Integer count = jdbcTemplate.queryForObject(sql, Integer.class, username, email);

return count != null && count > 0;

}

public boolean registerUser(User user) {

try {

if (userExists(user.getUsername(), user.getEmail())) {

return false;

}

// Fetch the last user\_id from the database and increment it

String fetchIdSql = "SELECT COALESCE(MAX(user\_id), 0) FROM users";

Integer lastUserId = jdbcTemplate.queryForObject(fetchIdSql, Integer.class);

int newUserId = (lastUserId != null) ? lastUserId + 1 : 1;

// Insert the new user into the database

String insertSql = "INSERT INTO users (user\_id, user\_name, email, password, role, registration\_date) VALUES (?, ?, ?, ?, ?, SYSDATE)";

int rowsAffected = jdbcTemplate.update(insertSql, newUserId, user.getUsername(), user.getEmail(), user.getPassword(), user.getRole());

return rowsAffected > 0;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

public List<User> getAllUsers() {

String sql = "SELECT user\_id as userID, user\_name as username , password, email, role, registration\_date as registrationDate, last\_login\_date as lastLoginDate FROM users";

return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(User.class));

}

public User getUserById(int userId) {

String sql = "SELECT user\_id as userID, user\_name as username , password, email, role, registration\_date as registrationDate, last\_login\_date as lastLoginDate FROM users WHERE user\_id = ?";

try {

return jdbcTemplate.queryForObject(sql, new Object[]{userId}, new BeanPropertyRowMapper<>(User.class));

} catch (Exception e) {

return null;

}

}

public boolean addUser(User user) {

String sql = "INSERT INTO users (user\_name, email, password, role, registration\_date) VALUES (?, ?, ?, ?, SYSDATE)";

int rowsAffected = jdbcTemplate.update(sql, user.getUsername(), user.getEmail(), user.getPassword(), user.getRole());

return rowsAffected > 0;

}

// Update an existing user

public boolean updateUser(int userId, User user) {

String sql = "UPDATE users SET user\_name = ?, email = ?, password = ?, role = ? WHERE user\_id = ?";

int rowsAffected = jdbcTemplate.update(sql, user.getUsername(), user.getEmail(), user.getPassword(), user.getRole(), userId);

return rowsAffected > 0;

}

// Delete a user

public boolean deleteUser(int userId) {

String sql = "DELETE FROM users WHERE user\_id = ?";

int rowsAffected = jdbcTemplate.update(sql, userId);

return rowsAffected > 0;

}

public User getUserProfile(int userId) {

String sql = "SELECT user\_id AS userID, user\_name AS username, password, email, role, " +

"registration\_date AS registrationDate, last\_login\_date AS lastLoginDate " +

"FROM users WHERE user\_id = ?";

return jdbcTemplate.queryForObject(sql, new Object[]{userId}, (rs, rowNum) -> {

User user = new User();

user.setUserID(rs.getInt("userID"));

user.setUsername(rs.getString("username"));

user.setPassword(rs.getString("password")); // Store password only if necessary

user.setEmail(rs.getString("email"));

user.setRole(rs.getString("role"));

return user;

});

}

public void updateUserProfile(User user) {

String sql = "UPDATE users SET user\_name = ?, email = ? WHERE user\_id = ?";

jdbcTemplate.update(sql, user.getUsername(), user.getEmail(), user.getUserID());

}

}

## Triggers and Stored Procedures

**Trigger 1**

CREATE OR REPLACE TRIGGER maintain\_books\_backup

AFTER INSERT OR UPDATE OR DELETE

ON books

FOR EACH ROW

BEGIN

IF INSERTING THEN

INSERT INTO books\_backup (ISBN, title, author, genre\_name)

VALUES (:NEW.ISBN, :NEW.title, :NEW.author, :NEW.genre\_name);

ELSIF UPDATING THEN

UPDATE books\_backup

SET title = :NEW.title,

author = :NEW.author,

genre\_name = :NEW.genre\_name

WHERE ISBN = :OLD.ISBN;

ELSIF DELETING THEN

DELETE FROM books\_backup WHERE ISBN = :OLD.ISBN;

END IF;

END;

**Trigger 2**

CREATE OR REPLACE TRIGGER restrict\_genre\_update

BEFORE UPDATE OF genre\_name

ON books

FOR EACH ROW

BEGIN

RAISE\_APPLICATION\_ERROR(-20001, 'Genre name updates are not allowed.');

END;

**Trigger 3**

CREATE OR REPLACE TRIGGER maintain\_users\_backup

AFTER INSERT OR UPDATE OR DELETE

ON users

FOR EACH ROW

BEGIN

IF INSERTING THEN

INSERT INTO users\_backup (user\_id, user\_name, password, email, role, registration\_date, last\_login\_date)

VALUES (:NEW.user\_id, :NEW.user\_name, :NEW.password, :NEW.email, :NEW.role, :NEW.registration\_date, :NEW.last\_login\_date);

ELSIF UPDATING THEN

UPDATE users\_backup

SET user\_name = :NEW.user\_name,

password = :NEW.password,

email = :NEW.email,

role = :NEW.role,

registration\_date = :NEW.registration\_date,

last\_login\_date = :NEW.last\_login\_date

WHERE user\_id = :OLD.user\_id;

ELSIF DELETING THEN

DELETE FROM users\_backup WHERE user\_id = :OLD.user\_id;

END IF;

END;

## Result

The following results were achieved upon completing the implementation and testing phases of the project:

1. Users can successfully register and log in to the system with secure password storage and validation.
2. Users can create, view, and cancel reservations seamlessly through the user-friendly interface. Reservation statuses are updated in real time in the database.
3. Admins can manage users and reservations efficiently, with options to view, add, and delete records as needed.
4. Admin dashboards provide an overview of system activities and overdue reservations.
5. The system generates dynamic HTML pages using Thymeleaf, ensuring responsive and interactive user interfaces.
6. All functionalities are backed by robust data handling through Oracle Database, ensuring data integrity and consistency.
7. The application gracefully handles common errors like invalid inputs, authentication failures, and database constraints.
8. The system demonstrates stable performance under typical loads, with quick response times for database operations.
9. The modular design allows for easy addition of new features or integration with other systems in the future.
10. The web application works seamlessly across modern web browsers and devices, ensuring broad accessibility.

# Chapter 5: Testing

## Software Testing

The testing phase of the project focused on ensuring the reliability, functionality, and usability of the application. Software testing was conducted in two primary stages: unit testing and integration testing.

* **Unit Testing:** Each individual module was tested in isolation to ensure that it performed as expected. For example, the UserController was tested to validate correct behavior for login, registration, and profile management. These tests helped identify and fix issues in the core logic of the application.
* **Integration Testing:** This phase ensured that the modules interacted seamlessly. For instance, after successfully reserving a book using the ReservationController, integration tests verified that the database was updated correctly and the user’s reservation history was displayed accurately in the front-end.

The testing strategy included using mock data to simulate user interactions and database transactions. Various test cases were executed, such as verifying successful user registration and login, testing admin functionalities for managing users and reservations etc.

Overall, testing confirmed that the application met its functional requirements, had no major errors, and provided a user-friendly experience.

## Module Testing and Integration

Each module of the system was tested independently to ensure that its functionality aligned with the defined requirements. After individual module testing, integration testing was performed to ensure smooth communication between modules. For instance, the integration between the User and Reservation modules was tested to confirm that a logged-in user could only access their own reservation data. Similarly, the synchronization between the Admin and Reservation modules was verified to ensure admins could view and manage all reservations accurately.

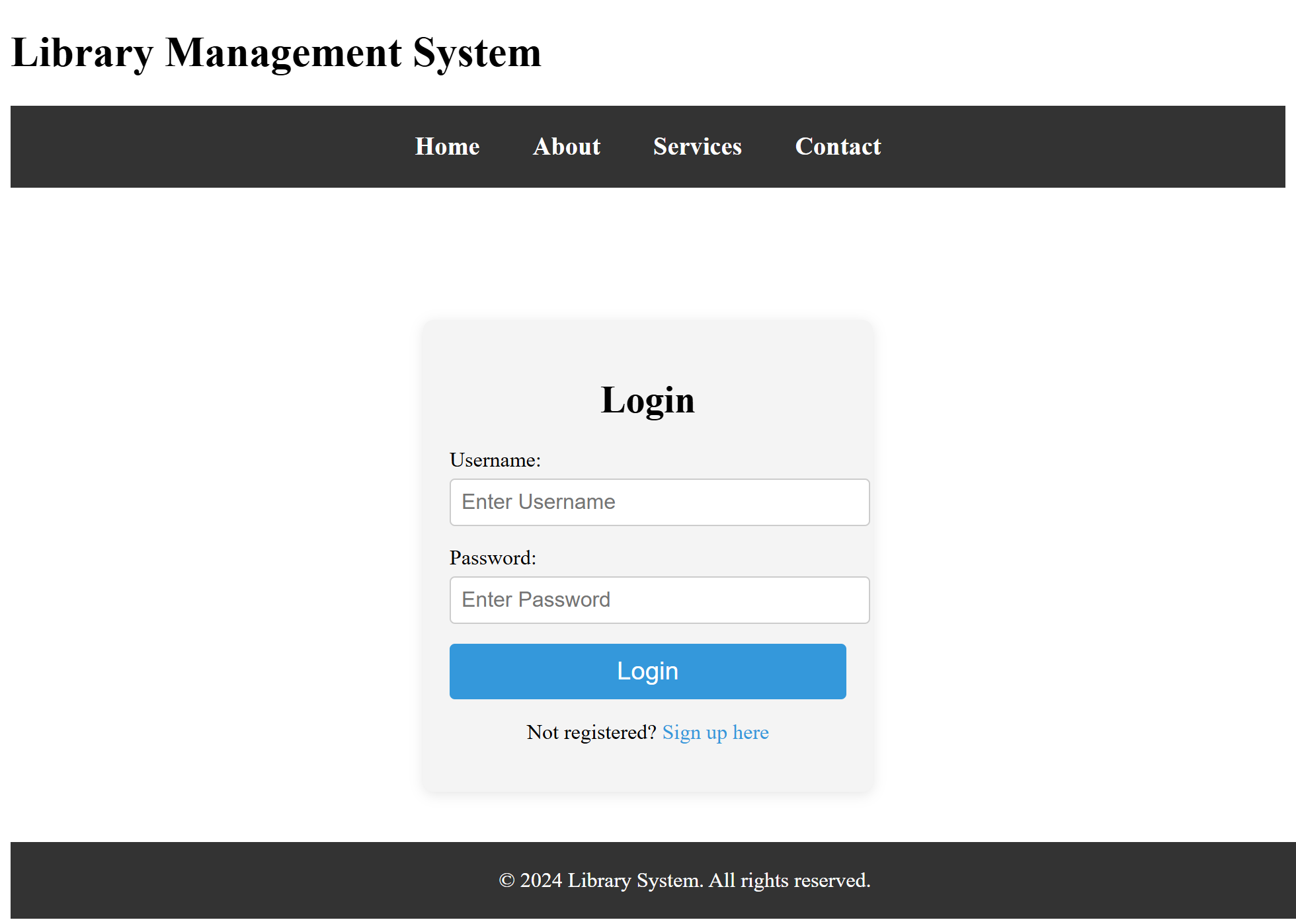
These integration tests helped identify and resolve minor issues such as data format mismatches or session inconsistencies, ensuring a robust and cohesive system.

## Limitations

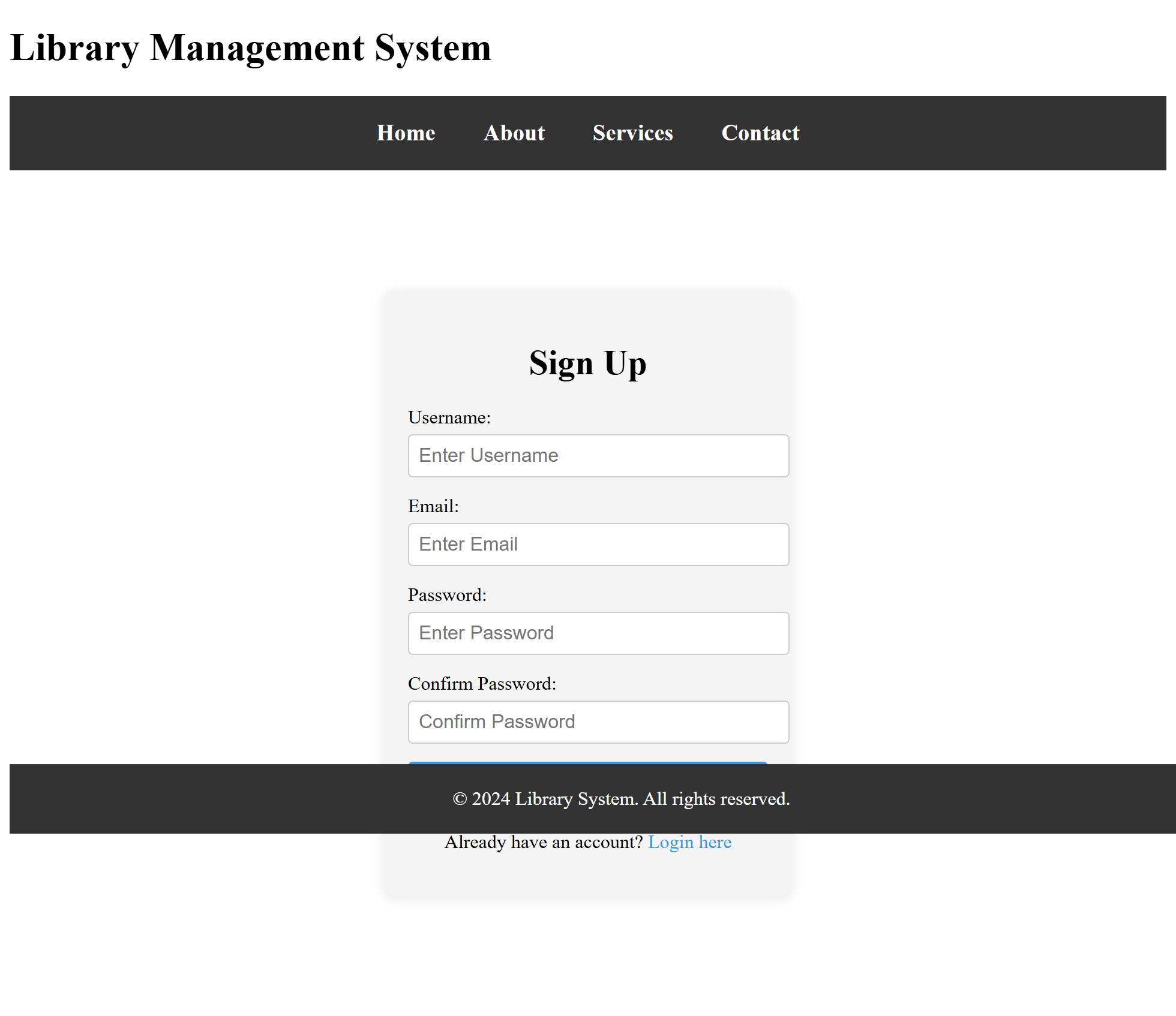
1. While functional, the current UI design could benefit from enhancements to improve aesthetic appeal and user-friendliness, especially on mobile devices.
2. The system may face challenges handling a significantly large number of users and reservations without performance optimization.
3. Features like advanced search filters, bulk operations, and detailed analytics are not yet implemented, which may limit the system's usability in larger libraries.
4. Basic error handling is implemented, but more robust mechanisms, such as comprehensive logging and user-friendly error messages, are needed for production-level deployment.
5. The system lacks complete mobile optimization, which could limit its accessibility for users relying on smaller screens.

# Chapter 6: Snapshots

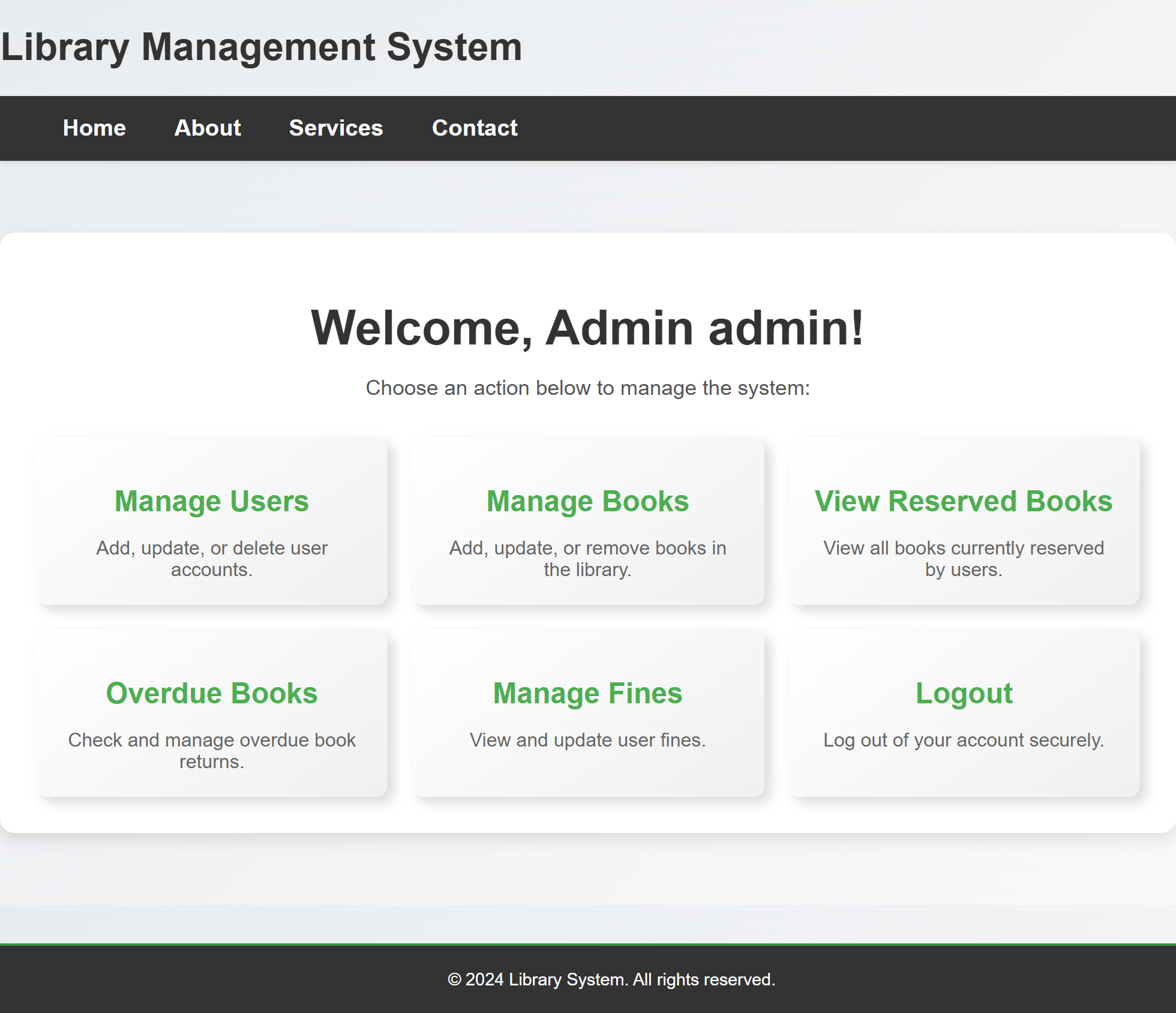
## Login Page

****

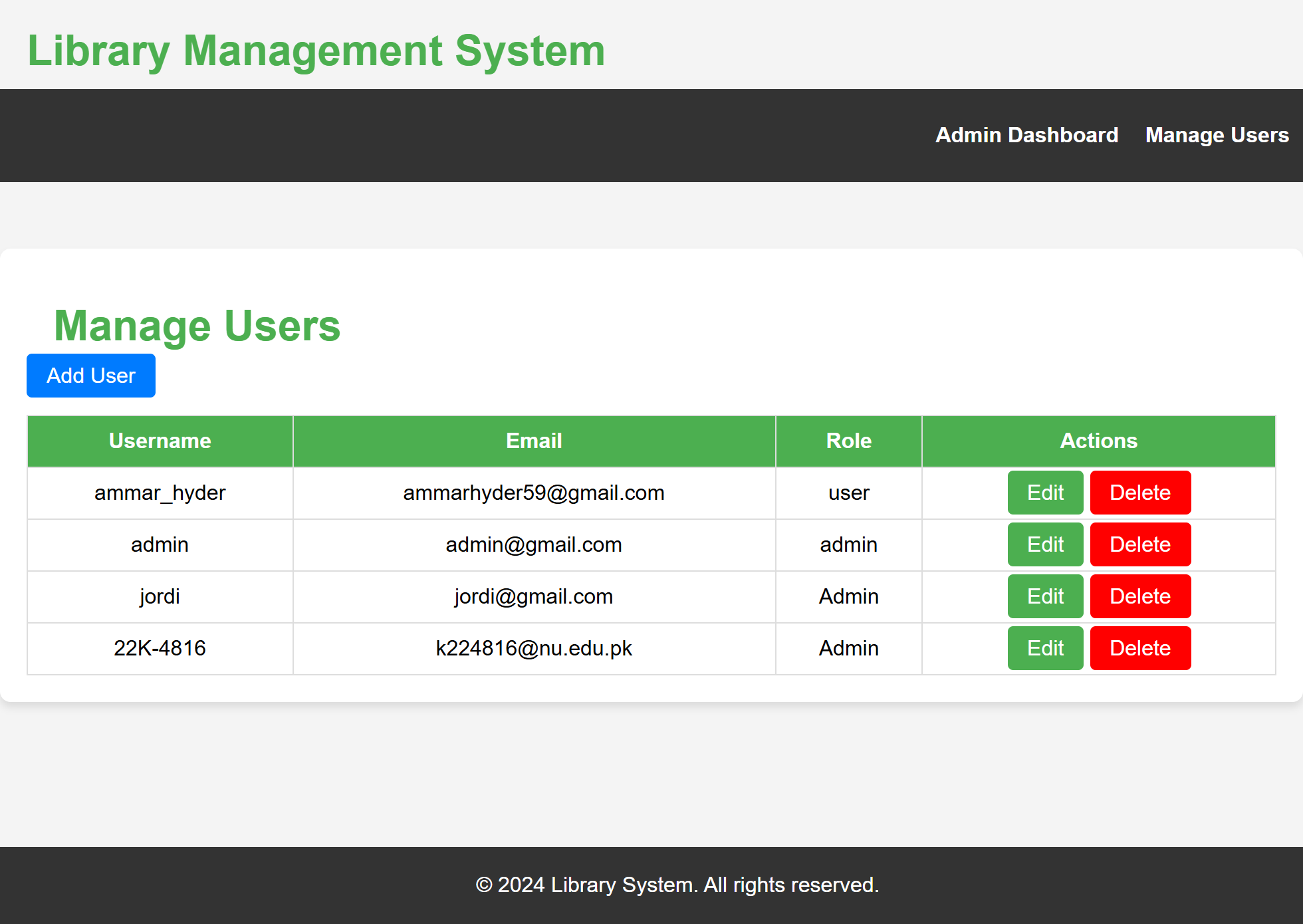
## Registration Page

****

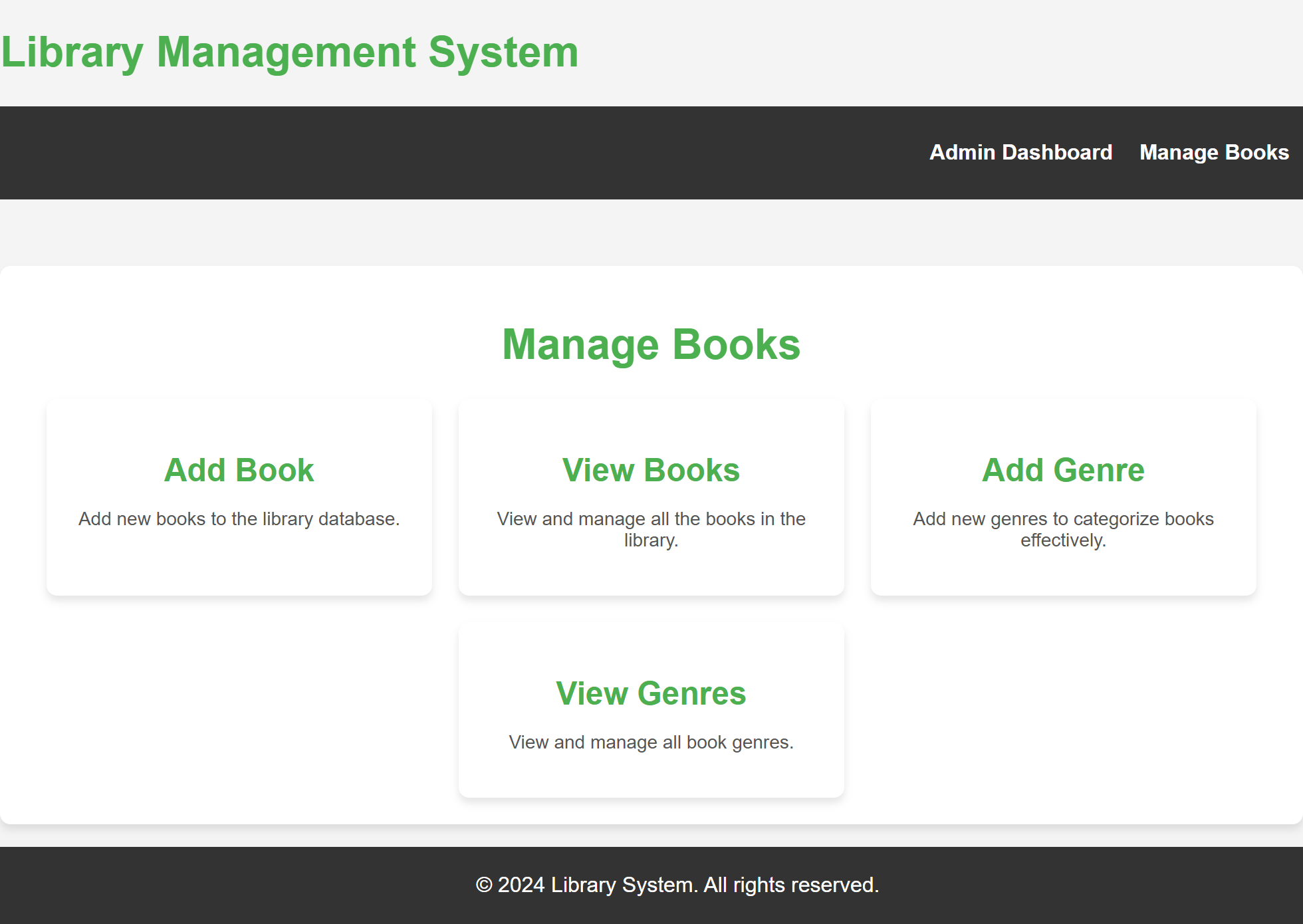
## Admin Home Page

****

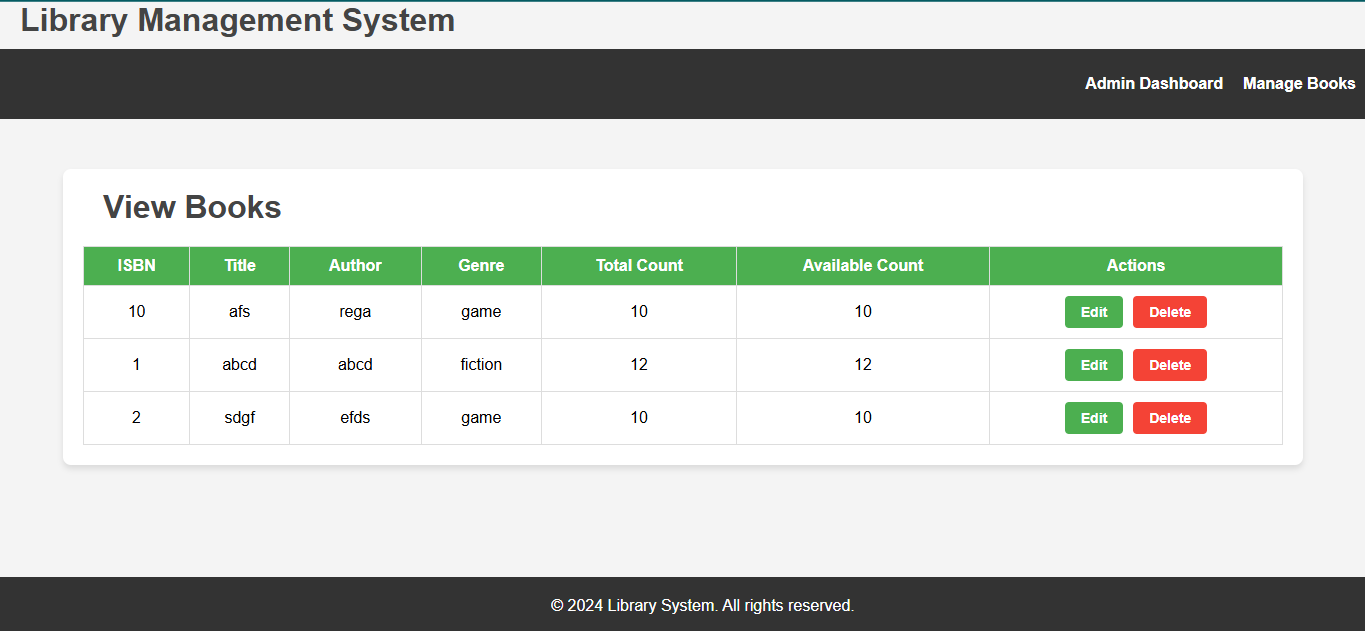
## Manage Users

****

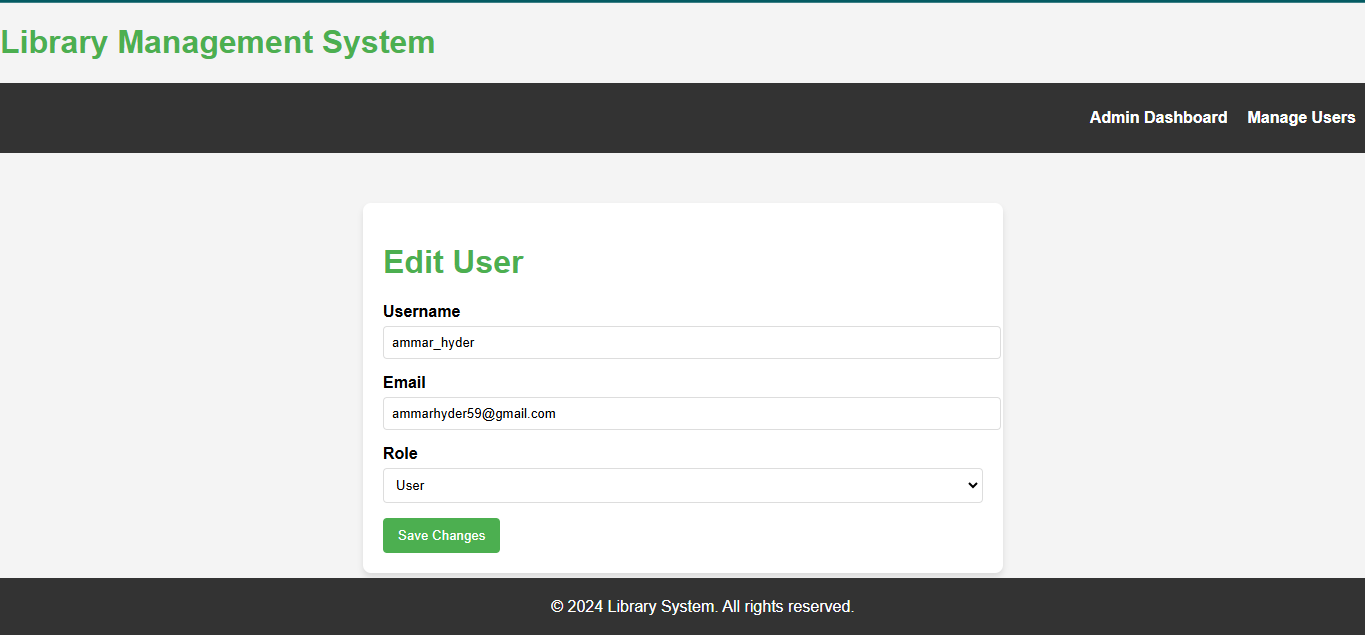
## Manage Books

****

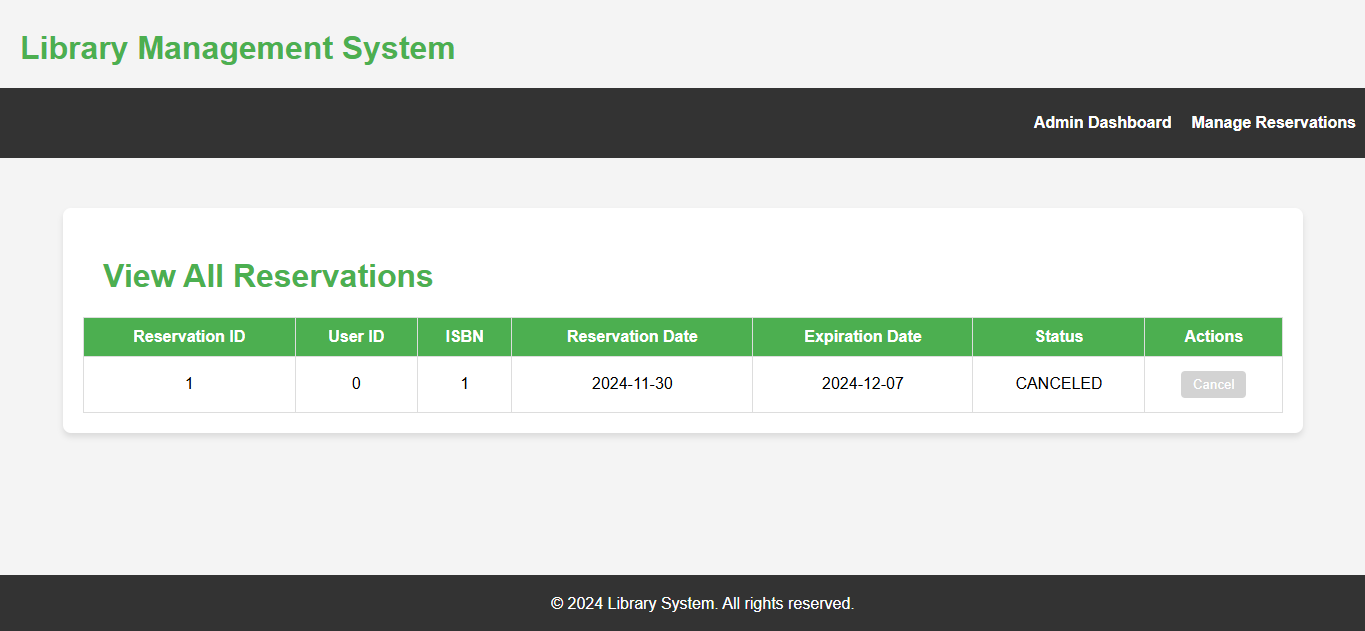
## View/Edit Books

****

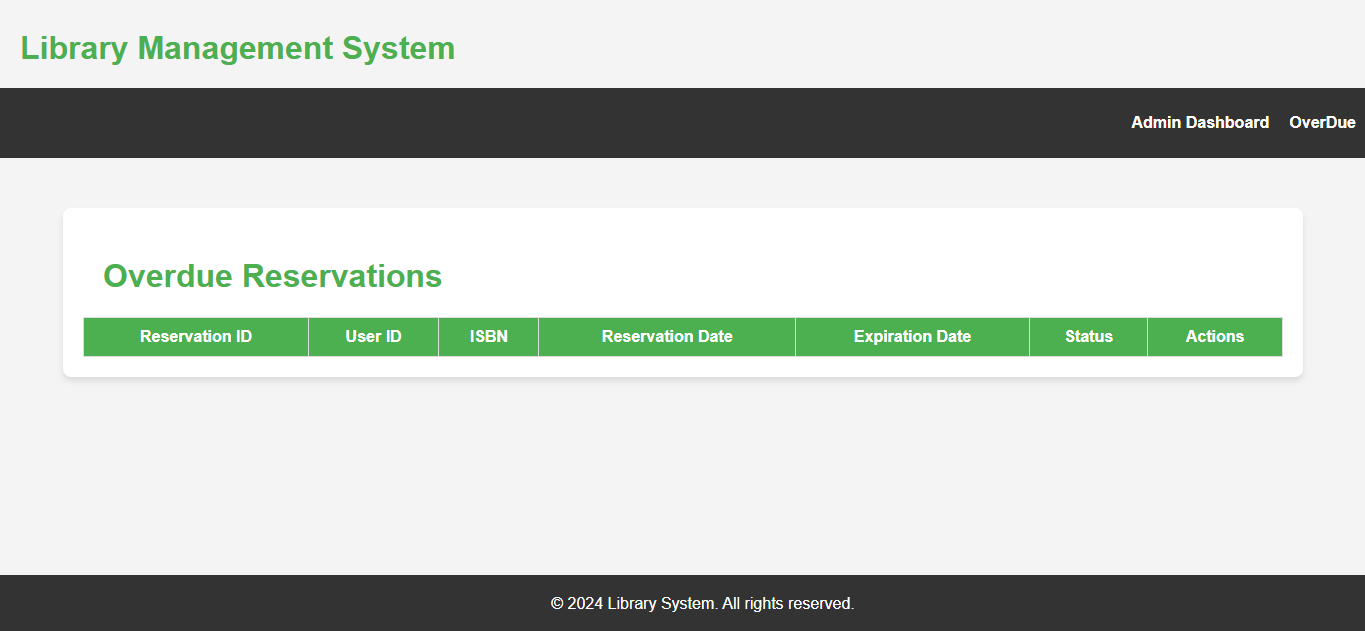
## Edit User

****

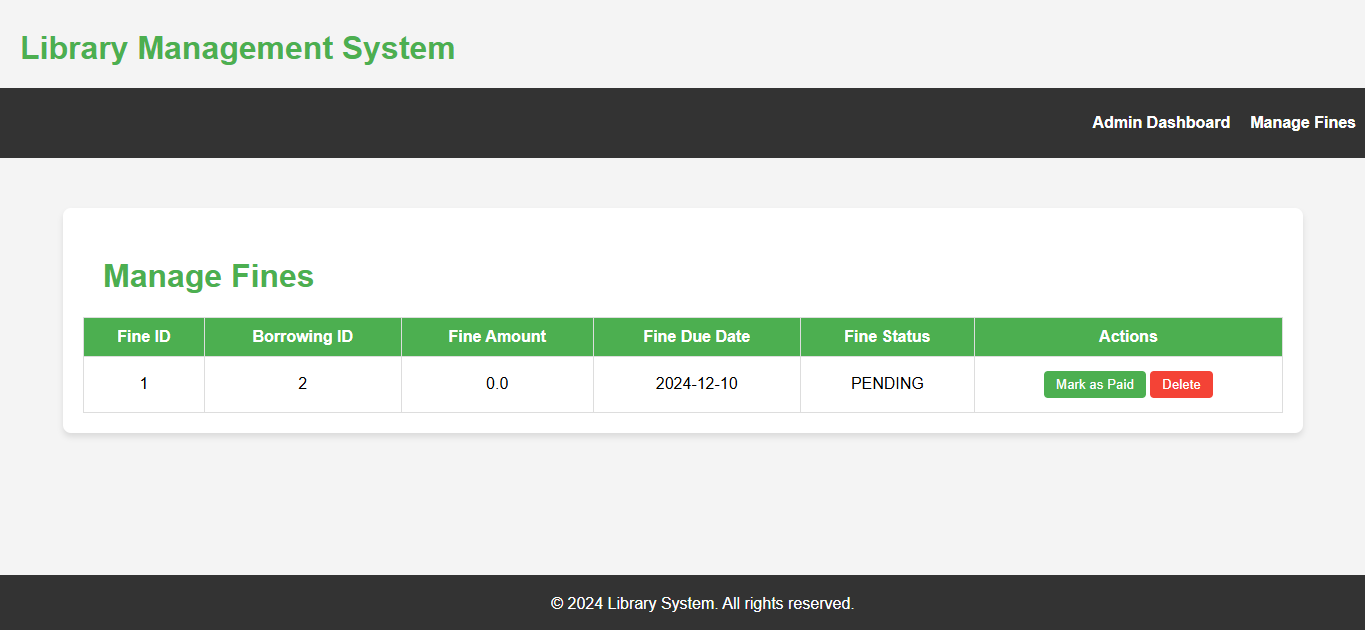
## View Reservation

****

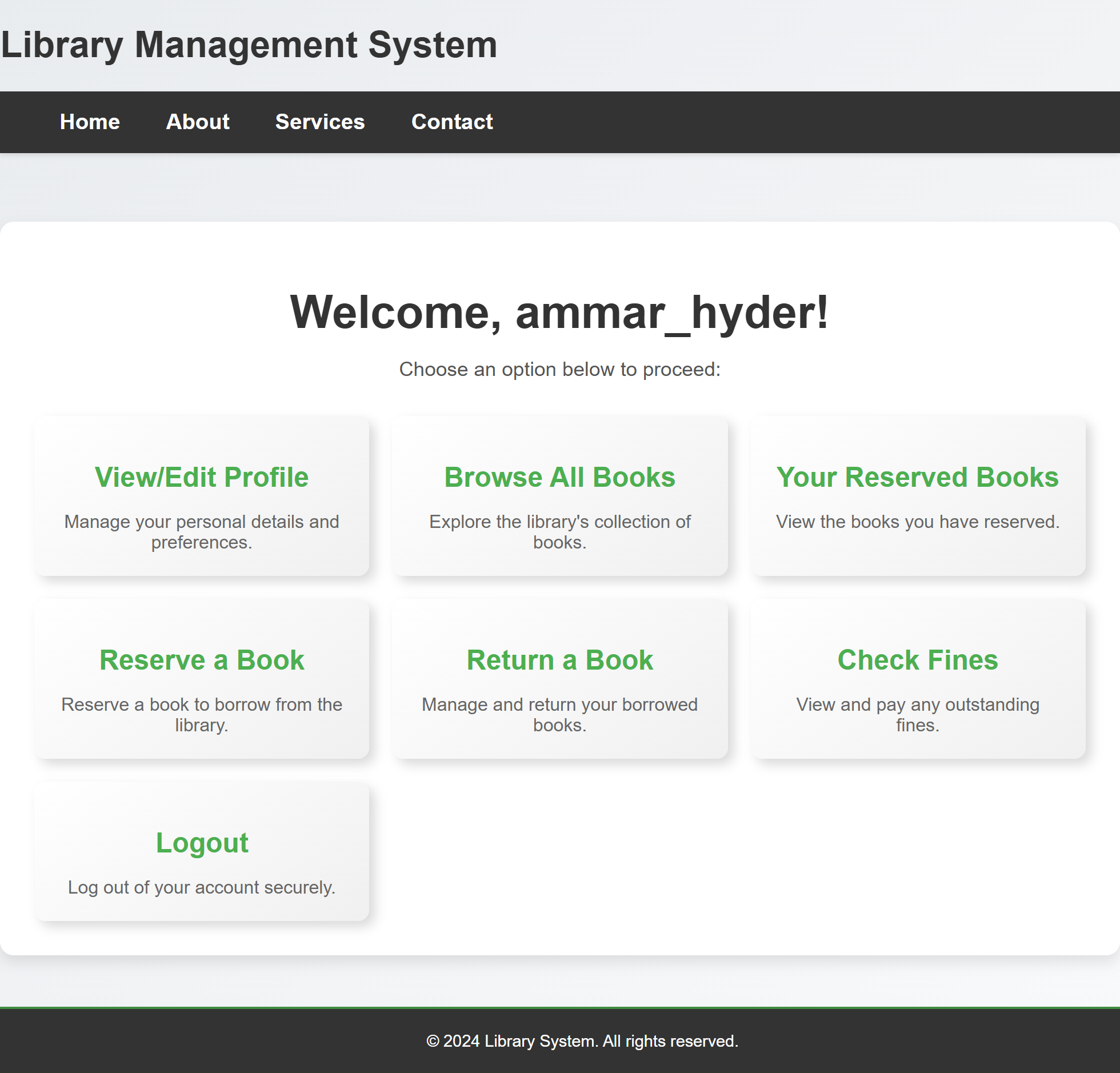
## OverDue

****

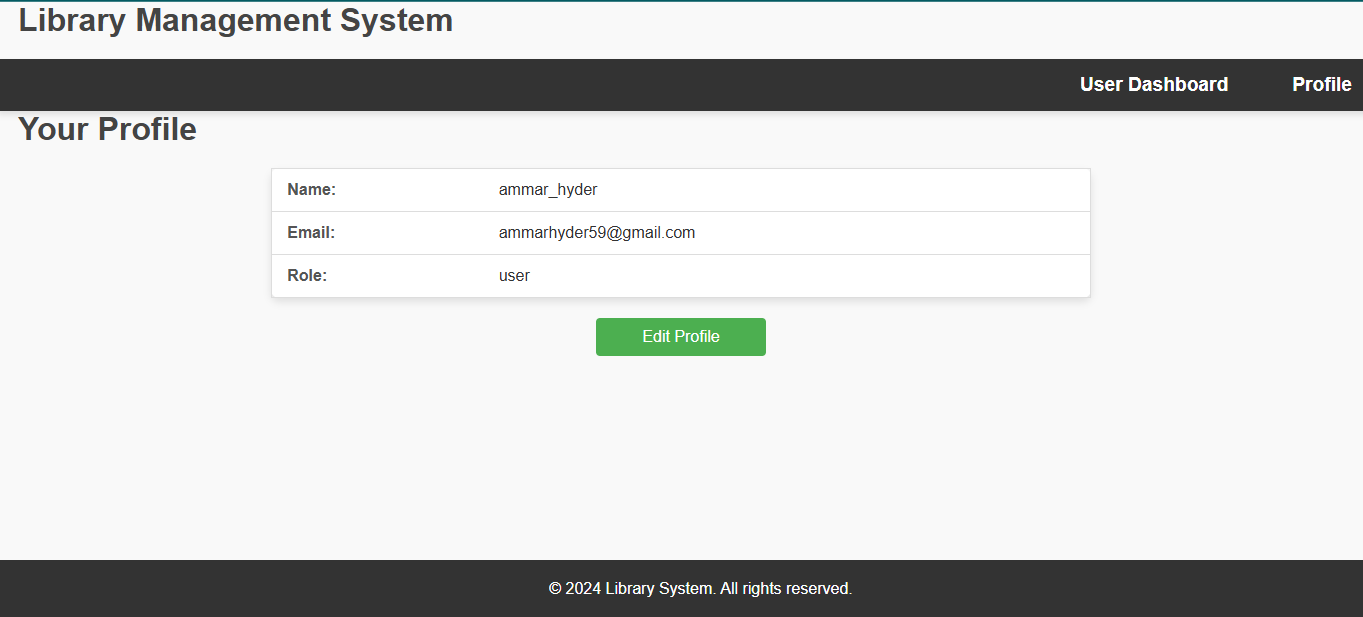
## Manage Fine

****

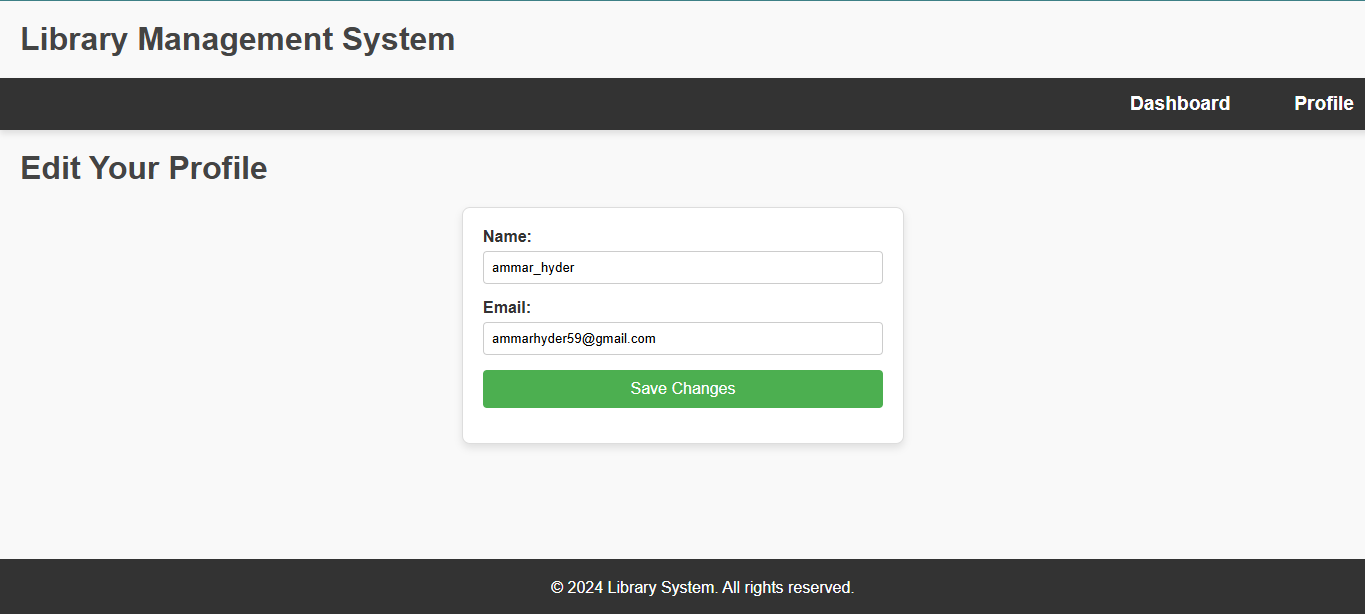
## User Home

****

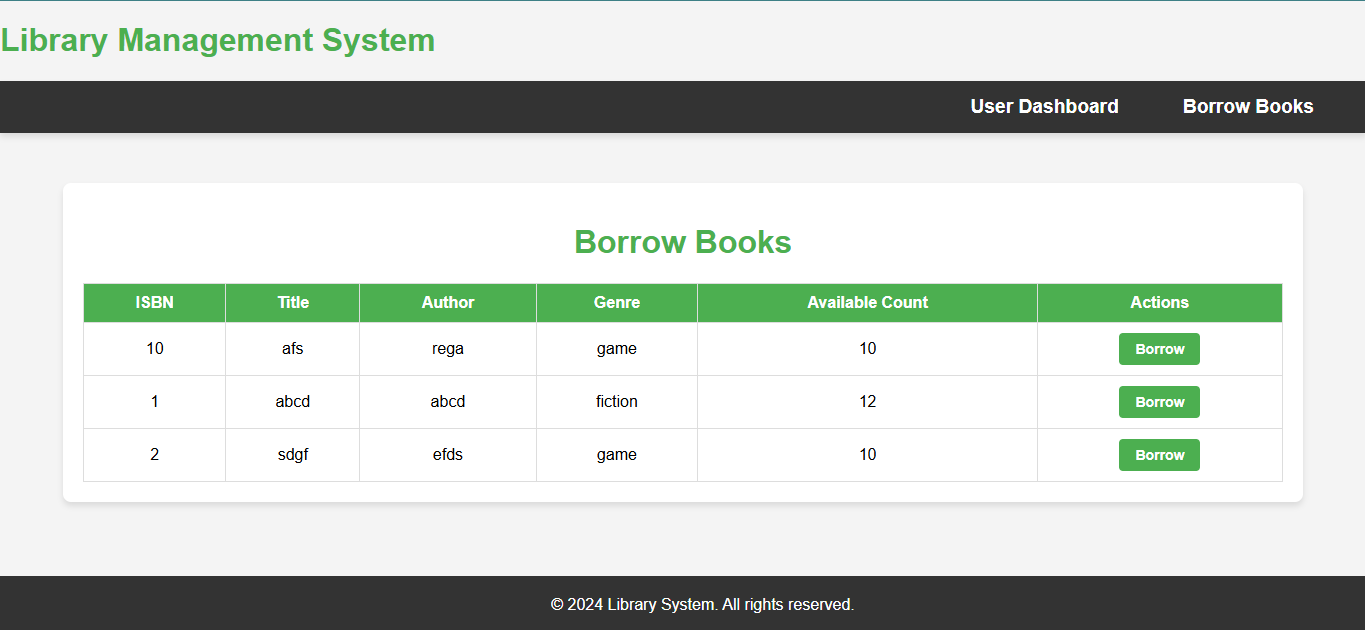
## View Profile

****

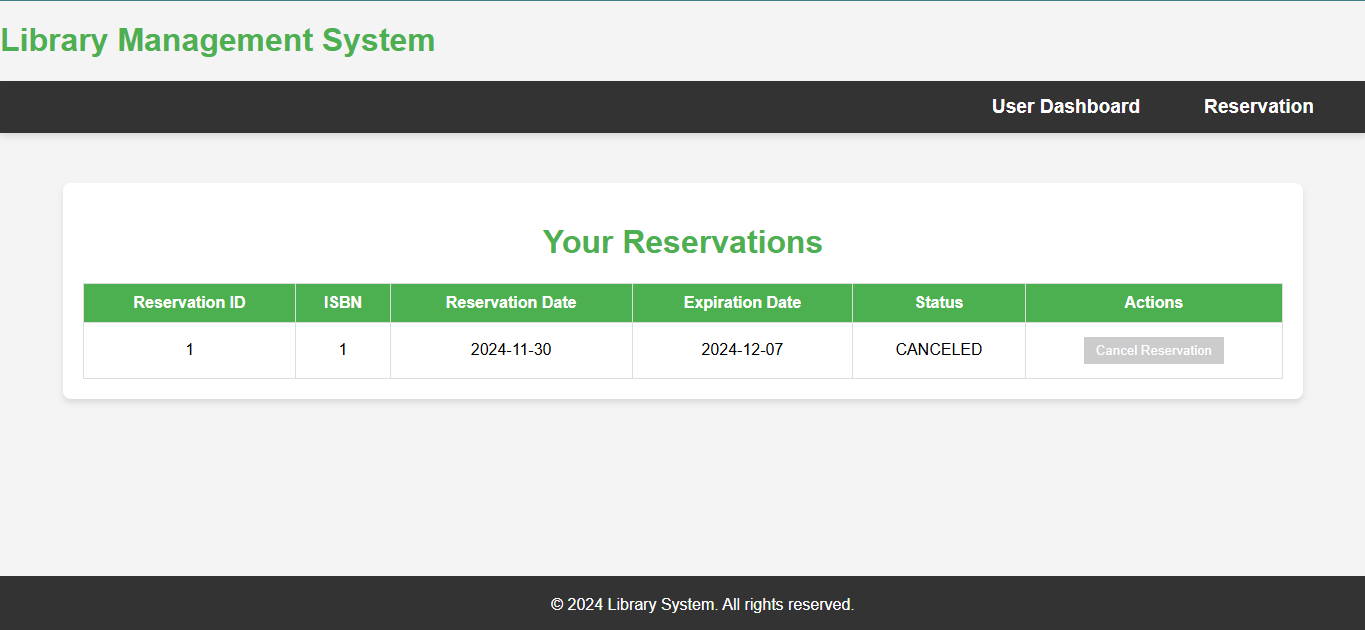
## Edit Profile

****

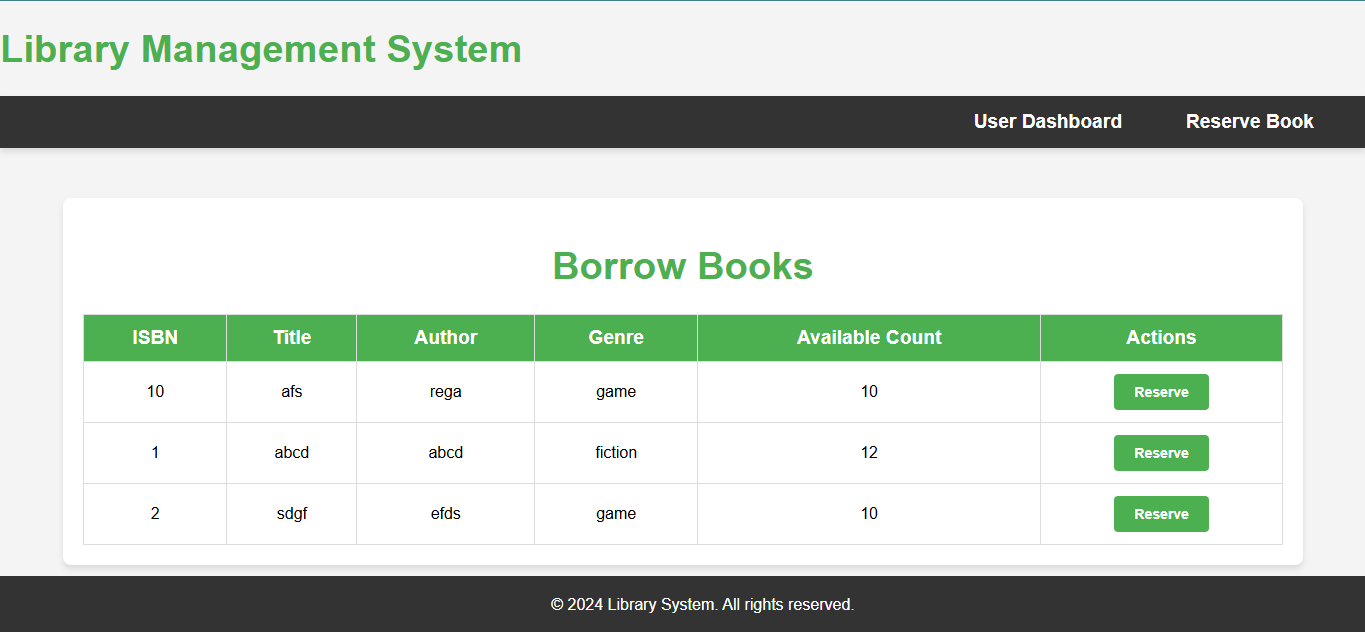
## Borrow Books

****

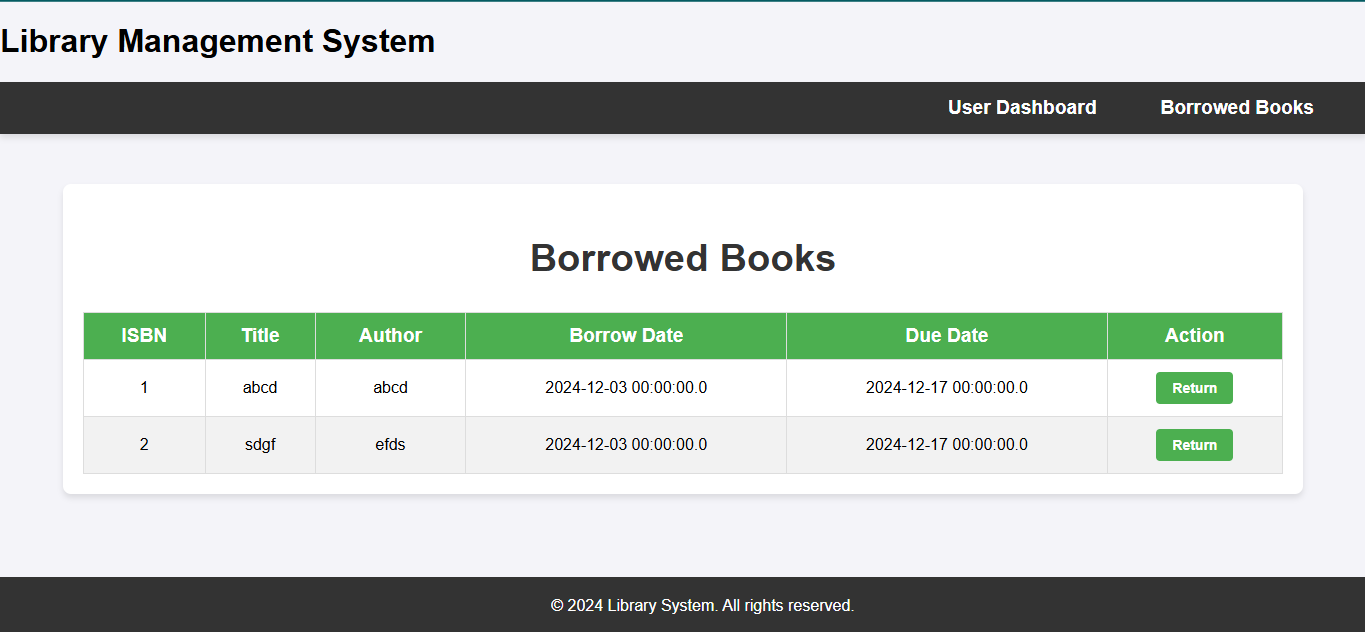
## Reserved Books

****

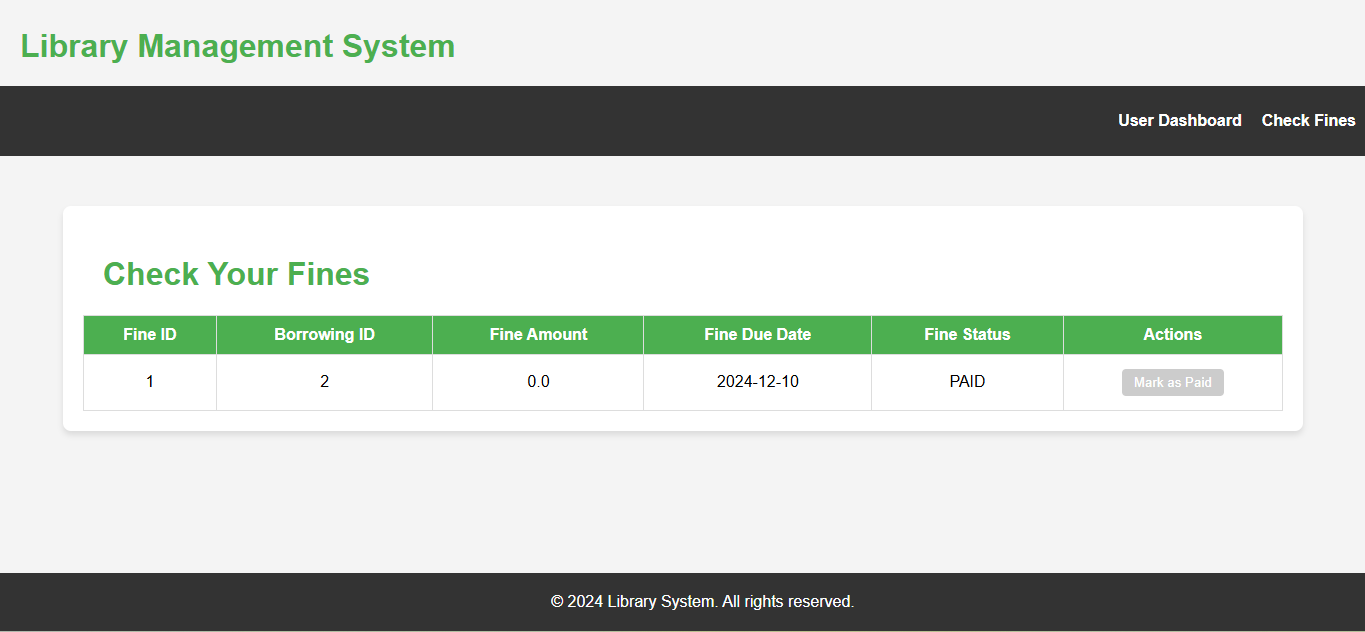
## Reserve Books

****

## Return Books

****

## Check/Pay Fine

****

# Chapter 7: Conclusion

The library reservation system developed as part of this project fulfills its intended purpose of providing an efficient and user-friendly platform for managing book reservations. By leveraging the power of Spring Boot, Thymeleaf, and Oracle Database, the system offers seamless functionality for both users and administrators. Users can easily register, log in, and manage their reservations, while administrators are equipped with tools to oversee users and reservations effectively. The modular architecture ensures maintainability and scalability, making the system adaptable to future requirements. Overall, the project demonstrates a practical application of modern web development technologies to solve real-world problems in library management.

# Chapter 8: Future Enhancements

Future enhancements which may be made include:

1. Expand the system to include a mobile application, enabling users to manage reservations, browse genres, and access their profiles on-the-go.
2. Introduce machine learning algorithms to provide personalized book recommendations based on users’ borrowing history, preferences, and ratings.
3. Implement a notification system (sent via email, SMS, or push notifications) that alerts users about overdue books, upcoming reservation deadlines, new book arrivals etc.
4. Add an analytics feature for admin-users to track system usage, monitor overdue books, and generate reports on user activity and book trends.
5. Expand the user profile functionality to include features like reading history, wish lists, and reviews.
6. Provide language options to cater to a diverse user base, making the system accessible to non-native speakers and increasing its usability.
7. Introduce QR codes for book identification, allowing users to quickly scan and reserve books using their mobile devices.
8. Add options for paid services such as premium memberships, late fee payments, or access to exclusive book collections.

# References

1. Spring Boot Documentation: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
2. Thymeleaf Documentation: <https://www.thymeleaf.org/documentation.html>
3. Oracle Database Documentation:<https://docs.oracle.com/en/database/>
4. "Spring in Action" by Craig Walls