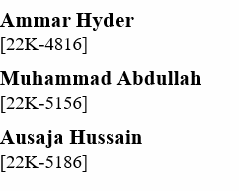


Library Management System

**By**

****

# 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 developed using ASP.NET MVC for the backend and MS SQL Server for data storage. The frontend is built with HTML and styled using Tailwind CSS to ensure a responsive and modern user interface. The system supports two roles—normal users and administrators—with role- based access control. Key features include user authentication, book reservation and cancellation, and profile management. Administrators have extended capabilities to manage users, reservations, and monitor overdue books. The system aims to deliver an intuitive and efficient platform for handling library operations and enhancing the overall user experience.

# Table Of Contents

[Acknowledgement 2](#_TOC_250045)

[Abstract 2](#_TOC_250044)

[Table Of Contents 3](#_TOC_250043)

[List of Figures 5](#_TOC_250042)

1. [Chapter 1: Introduction 6](#_TOC_250041)
   1. [Overview 6](#_TOC_250040)
   2. [Problem Statement 6](#_TOC_250039)
   3. [Database Management System 7](#_TOC_250038)
   4. [MS SQL Server 7](#_TOC_250037)
   5. [HTML and Razor Views 8](#_TOC_250036)
   6. [ASP.NET and Database Connectivity 8](#_TOC_250035)
2. [Chapter 2: Requirements Specification 9](#_TOC_250034)
   1. [Overall Description 9](#_TOC_250033)
   2. [Specific Requirements 9](#_TOC_250032)
      1. [Software Requirements 11](#_TOC_250031)
      2. [Hardware Requirements 12](#_TOC_250030)
      3. [Technology 12](#_TOC_250029)
3. [Chapter 3: Detailed Design 13](#_TOC_250028)
   1. [System Design 13](#_TOC_250027)
   2. [Entity Relationship Diagram 14](#_TOC_250026)
   3. [Relational Schema 15](#_TOC_250025)
   4. [Class Diagram 16](#_TOC_250024)
   5. [Data Flow Diagram 17](#_TOC_250023)
   6. [Use Case Diagram 18](#_TOC_250022)
   7. [Description of Tables 19](#_TOC_250021)
   8. [Result 23](#_TOC_250020)
4. [Chapter 4: Testing 24](#_TOC_250019)
   1. [Software Testing 24](#_TOC_250018)
   2. [Module Testing and Integration 24](#_TOC_250017)
   3. [Limitations 25](#_TOC_250016)
5. [Chapter 5: Snapshots 26](#_TOC_250015)
   1. [Login Page 26](#_TOC_250014)
   2. [Registration Page 26](#_TOC_250013)
   3. [Admin Home Page 27](#_TOC_250012)
   4. [Manage Books 28](#_TOC_250011)
   5. [View/Edit Books 29](#_TOC_250010)
   6. [Add New Book 29](#_TOC_250009)
   7. [Book Details 30](#_TOC_250008)
   8. [Book Details 30](#_TOC_250007)
   9. [User Home 31](#_TOC_250006)
   10. [View Profile & Edit Profile 31](#_TOC_250005)
   11. [Borrow Books 32](#_TOC_250004)
   12. [Return Books 32](#_TOC_250003)
   13. [Check/Pay Fine 33](#_TOC_250002)
6. [Chapter 6: Conclusion 34](#_TOC_250001)
7. [Chapter 7: Future Enhancements 35](#_TOC_250000)

# 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 |
| **3.3** | Class Diagram of Library Management System | 16 |
| **3.4** | Data Flow Diagram of Library Management System | 17 |
| **3.5** | Use Case Diagram of Library Management System | 18 |

# Chapter 1: Introduction

## Overview

This project aims to develop a web-based Library Management System focused on handling book reservations efficiently. The application enables users to create accounts, log in, reserve books, and manage their reservations seamlessly. Administrators are granted extended privileges to manage user accounts, review and update reservations, and handle overdue books.

The system is developed using **ASP.NET MVC** for the backend, with **MS SQL Server** as the relational database to store and manage data related to users, books, reservations, and genres. The frontend is designed using standard **HTML** and styled with **Tailwind CSS** to ensure a modern, responsive user interface. This project seeks to streamline the reservation process, reduce manual administrative tasks, and offer a user-friendly platform for managing library operations.

## 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) plays a vital role in storing, organizing, and retrieving large volumes of structured data efficiently. For this project, **Microsoft SQL Server** is utilized due to its reliability, scalability, and seamless integration with the ASP.NET MVC framework. The database stores all critical information, including user credentials, book details, reservations, and genres. MS SQL Server supports advanced features such as stored procedures, views, and triggers, which are used to enforce business rules, automate routine tasks, and maintain data integrity throughout the system.

## MS SQL Server

MS SQL Server is used as the underlying database management system for this project due to its reliability, performance, and strong support for handling relational data at scale. The database stores essential information such as user accounts, book reservations, book details, and genre classifications.

Its relational structure ensures efficient data organization, indexing, and querying—key factors for maintaining the performance and accuracy of the library reservation system. In this project, MS SQL Server is accessed using ADO.NET and Entity Framework, enabling seamless integration with the ASP.NET MVC application. The system also leverages stored procedures and triggers to automate tasks like updating reservation statuses, enforcing data validation rules, and maintaining system consistency.

## HTML and Razor Views

HTML serves as the foundational structure for the front-end of this web application, defining the layout and presentation of content to users. In this project, **ASP.NET Razor Views** are used to dynamically render HTML pages by integrating server-side logic directly into the markup. Razor syntax allows seamless embedding of C# code within HTML, enabling the efficient display of dynamic data and interactive content.

To enhance the user interface and ensure responsive design, **Tailwind CSS** is applied throughout the frontend. The combination of Razor and Tailwind allows for clean, maintainable templates that promote a clear separation of concerns between presentation and business logic. This approach ensures a scalable and user-friendly experience while simplifying the development and customization of front-end components.

## ASP.NET and Database Connectivity

In this project, **ASP.NET MVC** is responsible for managing the interaction between the application and the **MS SQL Server** database. The framework provides robust tools for data access and communication between the presentation, business, and data layers.

Database connectivity is handled using **ADO.NET** and **Entity Framework (EF)**. Entity Framework simplifies data access by allowing developers to interact with the database using C# objects, eliminating the need for most raw SQL queries. The connection to the database is configured through the web.config file, where parameters like the connection string, database name, and credentials are defined. EF handles operations such as

querying, inserting, updating, and deleting data, while also supporting LINQ queries for efficient data retrieval.

This structured approach ensures secure, maintainable, and scalable communication between the application and 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 option 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 Authentication**: Users can log in to the system using their credentials.
  + **User Profile Management**: Users can view and update their profile information and contact details.
  + **Book Search**: Users can search for books by title or author.

### Book Borrowing:

* + - Users can borrow available books.
    - Users can view the list of books they have borrowed.
    - Users can return borrowed books.

### Fine Calculation and Payment:

* + - Fines are automatically calculated based on late returns.
    - Users can view and pay outstanding fines.
  + **Book Review**: Users can write reviews and rate books
  + **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.

### 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:

* + - View all registered users.
    - Add, update, or delete user accounts.
    - Manage user roles and contact information.
    - View login history of users.
  + **Author Management**: Perform full CRUD (Create, Read, Update, Delete) operations on author records.

### Book Management:

* + - Perform full CRUD operations on books in the inventory.
    - Manage borrowed and available copies of each book.
  + **Book Review Management**: View and manage book reviews submitted by users.
  + **Genre Management**: Admins can add, update, delete, and view different book genres.
  + **Borrowing Record Management**: View and manage borrowing activity of all users.

### Fine Management:

* + - View, update, or delete fine records for users.
    - Monitor payment status.
  + **Publisher Management**: Manage publishers' information with full CRUD support.
  + **Role Management**: Manage different user roles (admin, member, etc.) within the system.
  + **Login History Monitoring**: Track user login/logout activity.

## Software Requirements

* + - * .NET Framework 4.7.2
      * ASP.NET MVC
      * Microsoft SQL Server 2017 or higher
      * Visual Studio 2019/2022
      * Tailwind CSS
      * Entity Framework

## Hardware Requirements

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

## Technology

* + - * **ASP.NET MVC:** For backend logic and routing.
      * **HTML & Razor Views:** For front-end structure and dynamic content rendering.
      * **Tailwind CSS:** For responsive and utility-first styling.
      * **MS SQL Server:** For structured data storage and management.

# Chapter 3: Detailed Design

## System Design

The system follows a modular, layered architecture, using **ASP.NET MVC** for the backend, **HTML** and **Razor Views** for the front-end, and **MS SQL Server** for data storage. The database layer interacts with the **Model** layer to persist and retrieve data.

### Controller Layer:

Handles incoming HTTP requests and determines the appropriate action. The controllers interact with models to retrieve or manipulate data and pass it to views for rendering.

### Model Layer:

The **Model** is responsible for representing and interacting with the data. **Entity Framework (EF)** is used for object-relational mapping (ORM), allowing the application to interact with the **MS SQL Server** database using C# objects.

### Database:

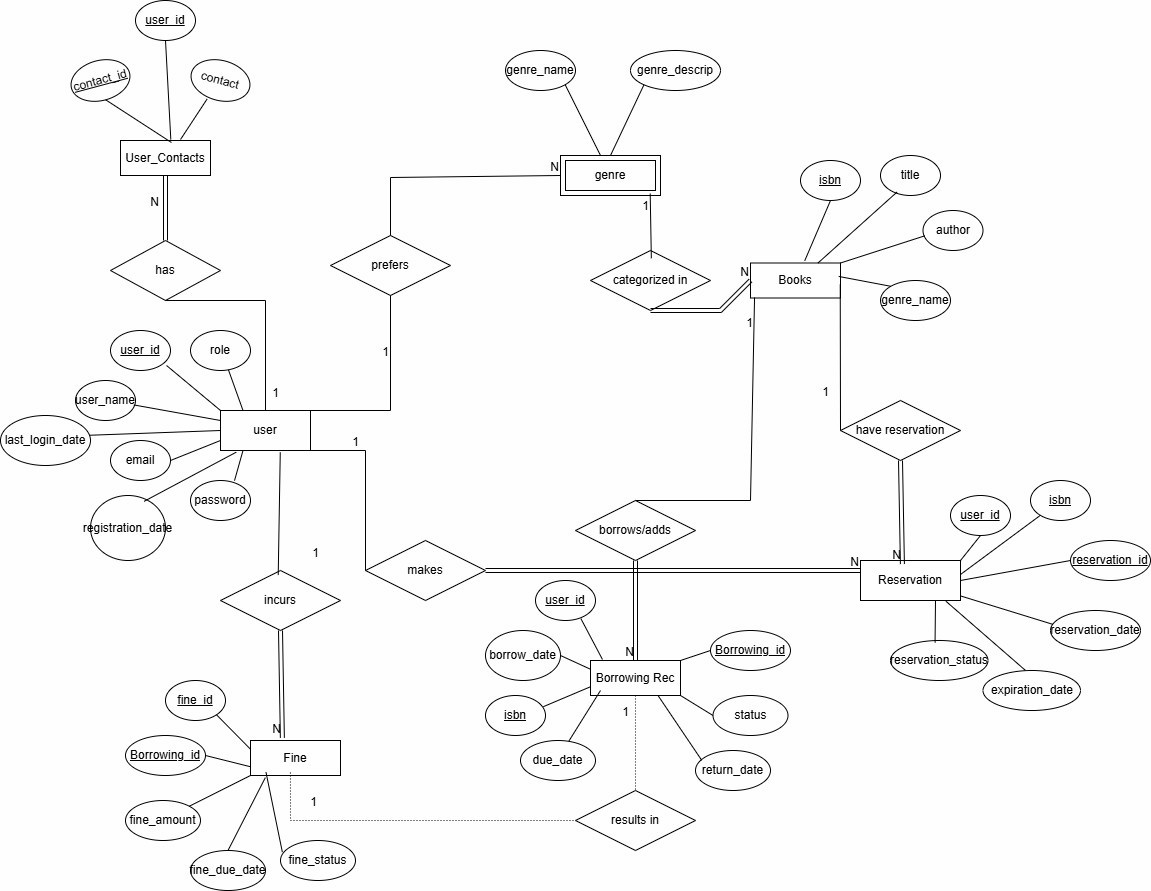
The **MS SQL Server** database is the central data store, designed with normalized tables to maintain data integrity and minimize redundancy. Relationships between tables are maintained using **foreign keys**, ensuring referential integrity.

### View Layer:

The **View** is responsible for rendering dynamic HTML content. **Razor Views** are used to generate the HTML pages, which are populated with data passed from controllers. **Tailwind CSS** is applied for responsive and clean design.

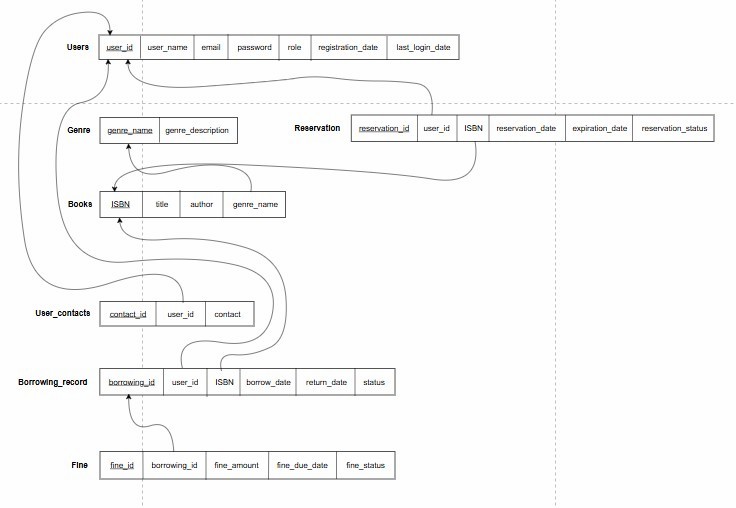
This design ensures separation of concerns, making the system easier to maintain and extend. Each layer performs a specific role, and their interaction is governed by well-defined interfaces, promoting modularity and scalability.

## Entity Relationship Diagram

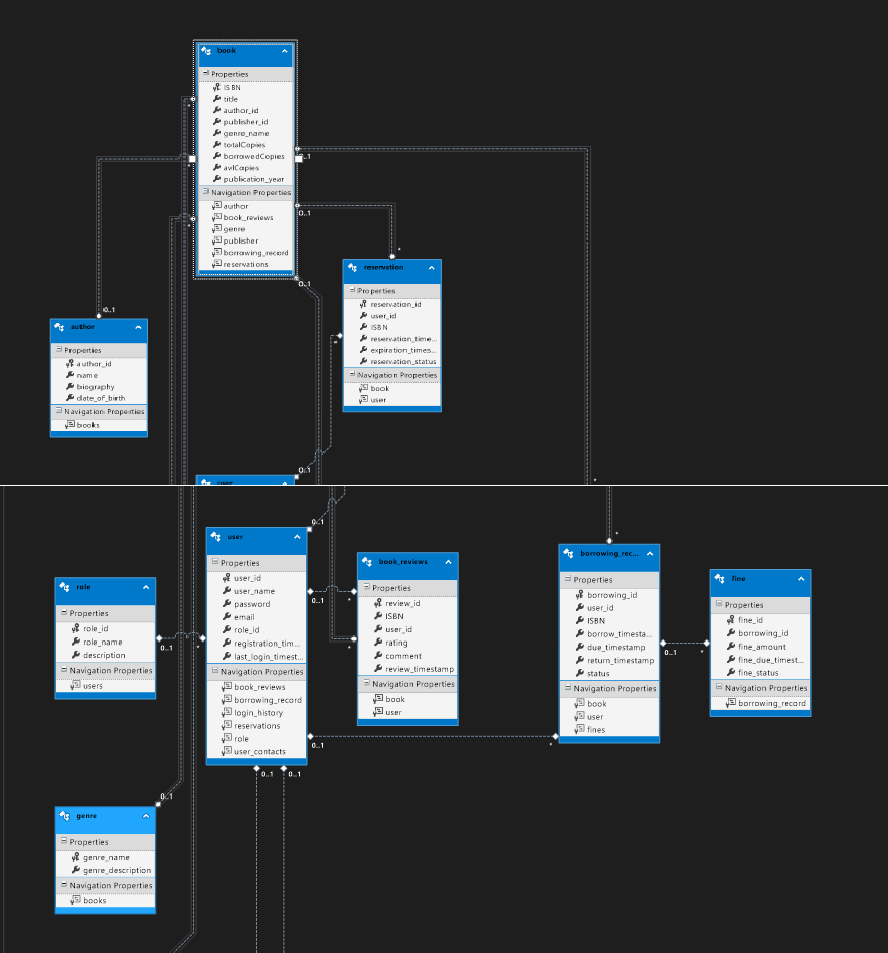
****

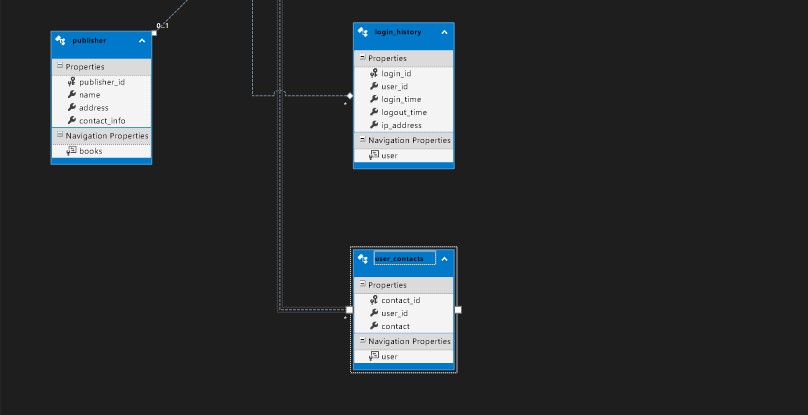
**Fig 3.1. ER Diagram of Library Management System**

## Relational Schema

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

## Class Diagram



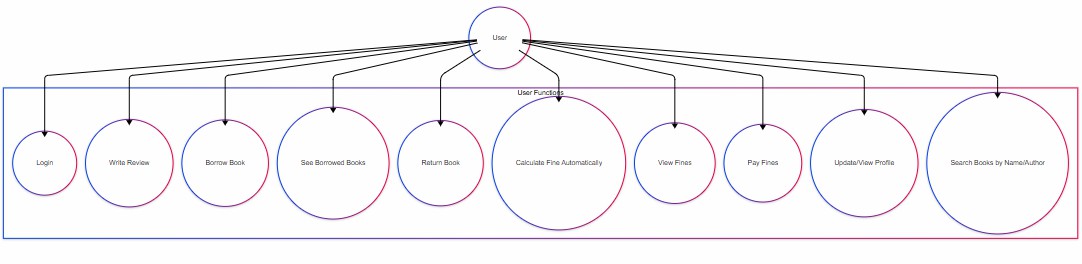


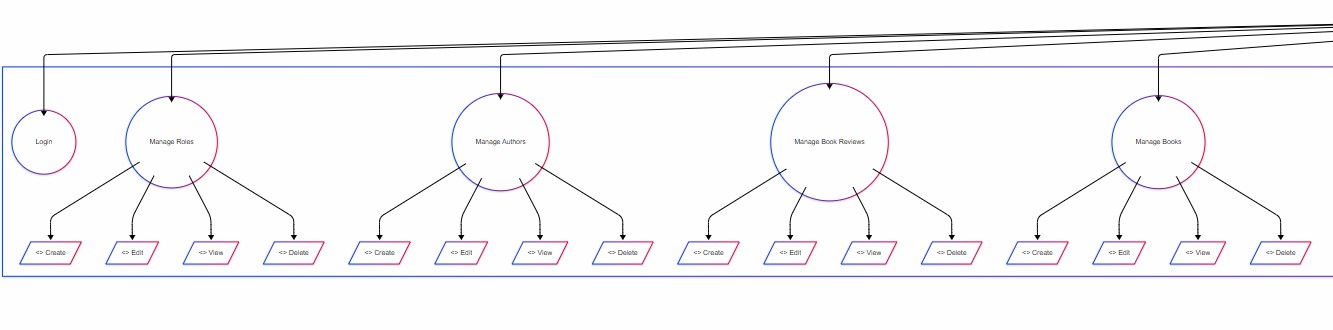
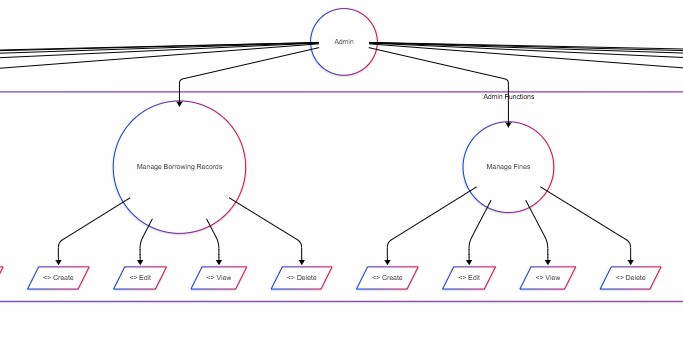
**Fig 3.3. Class Diagram of Library Management System**

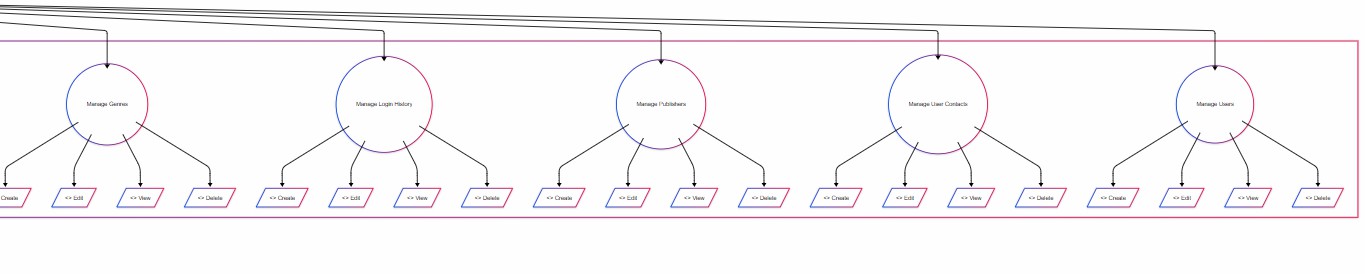
## Data Flow Diagram

**Fig 3.4. Data Flow Diagram of Library Management System**

## Use Case Diagram

****

****



**Fig 3.5. UseCase Diagram of Library Management System**

## Description of Tables

The database includes the following tables:

### Roles Table:

* + role\_id: Primary key, a unique identifier for each role.
  + role\_name: The name of the role (e.g., admin, member).
  + description: A description of the role's purpose.

### Users Table:

* + user\_id: Primary key, a unique identifier for each user.
  + user\_name: The name of the user.
  + password: The user's password for authentication.
  + email: The user's email address.
  + role\_id: Foreign key referencing the roles table, indicating the role of the user.
  + registration\_timestamp: The date and time the user registered in the system.
  + last\_login\_timestamp: The last time the user logged into the system.

### User Contacts Table:

* + contact\_id: Primary key, a unique identifier for each contact entry.
  + user\_id: Foreign key referencing the users table, indicating which user the contact belongs to.
  + contact: The contact details (e.g., phone number, alternate email).

### Login History Table:

* + login\_id: Primary key, a unique identifier for each login entry.
  + user\_id: Foreign key referencing the users table, indicating which user logged in.
  + login\_time: The date and time the user logged in.
  + logout\_time: The date and time the user logged out.
  + ip\_address: The IP address from which the user logged in.

### Genre Table:

* + genre\_name: Primary key, the unique name of the genre.
  + genre\_description: A brief description of the genre.

### Authors Table:

* + author\_id: Primary key, a unique identifier for each author.
  + name: The name of the author.
  + biography: A brief biography of the author.
  + date\_of\_birth: The author's date of birth.

### Publishers Table:

* + publisher\_id: Primary key, a unique identifier for each publisher.
  + name: The name of the publisher.
  + address: The address of the publisher.
  + contact\_info: Contact information for the publisher.

### Books Table:

* + ISBN: Primary key, a unique identifier for each book (International Standard Book Number).
  + title: The title of the book.
  + author\_id: Foreign key referencing the authors table, indicating the author of the book.
  + publisher\_id: Foreign key referencing the publishers table, indicating the publisher of the book.
  + genre\_name: Foreign key referencing the genre table, indicating the genre of the book.
  + totalCopies: The total number of copies available.
  + borrowedCopies: The number of copies currently borrowed.
  + avlCopies: A computed column that calculates the available copies (totalCopies - borrowedCopies).
  + publication\_year: The year the book was published.

### Borrowing Record Table:

* + borrowing\_id: Primary key, a unique identifier for each borrowing transaction.
  + user\_id: Foreign key referencing the users table, indicating the user borrowing the book.
  + ISBN: Foreign key referencing the books table, indicating the borrowed book.
  + borrow\_timestamp: The date and time the book was borrowed.
  + due\_timestamp: The date and time the book is due for return.
  + return\_timestamp: The actual date and time the book was returned.
  + status: The status of the borrowing (e.g., borrowed, returned, late).

### Fine Table:

* + fine\_id: Primary key, a unique identifier for each fine.
  + borrowing\_id: Foreign key referencing the borrowing\_record table, linking the fine to a specific borrowing.
  + fine\_amount: The amount of the fine imposed.
  + fine\_due\_timestamp: The date by which the fine should be paid.
  + fine\_status: The status of the fine (e.g., unpaid, paid).

### Book Reviews Table:

* + review\_id: Primary key, a unique identifier for each book review.
  + ISBN: Foreign key referencing the books table, indicating the book being reviewed.
  + user\_id: Foreign key referencing the users table, indicating the user who wrote the review.
  + rating: The rating given by the user (between 1 and 5).
  + comment: The review comment from the user.
  + review\_timestamp: The date and time the review was posted.

## 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 razor, ensuring responsive and interactive user interfaces.
6. All functionalities are backed by robust data handling through MS Sql 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 4: 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 Login Controller 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 borrowing a book using the Borrow Controller, integration tests verified that the database was updated correctly and the user’s borrow 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 books 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 Borrow modules was tested to confirm that a logged-in user could only access their own borrowing data. Similarly, the synchronization between the Admin and Borrowing modules was verified to ensure admins could view and manage all borrowings accurately.

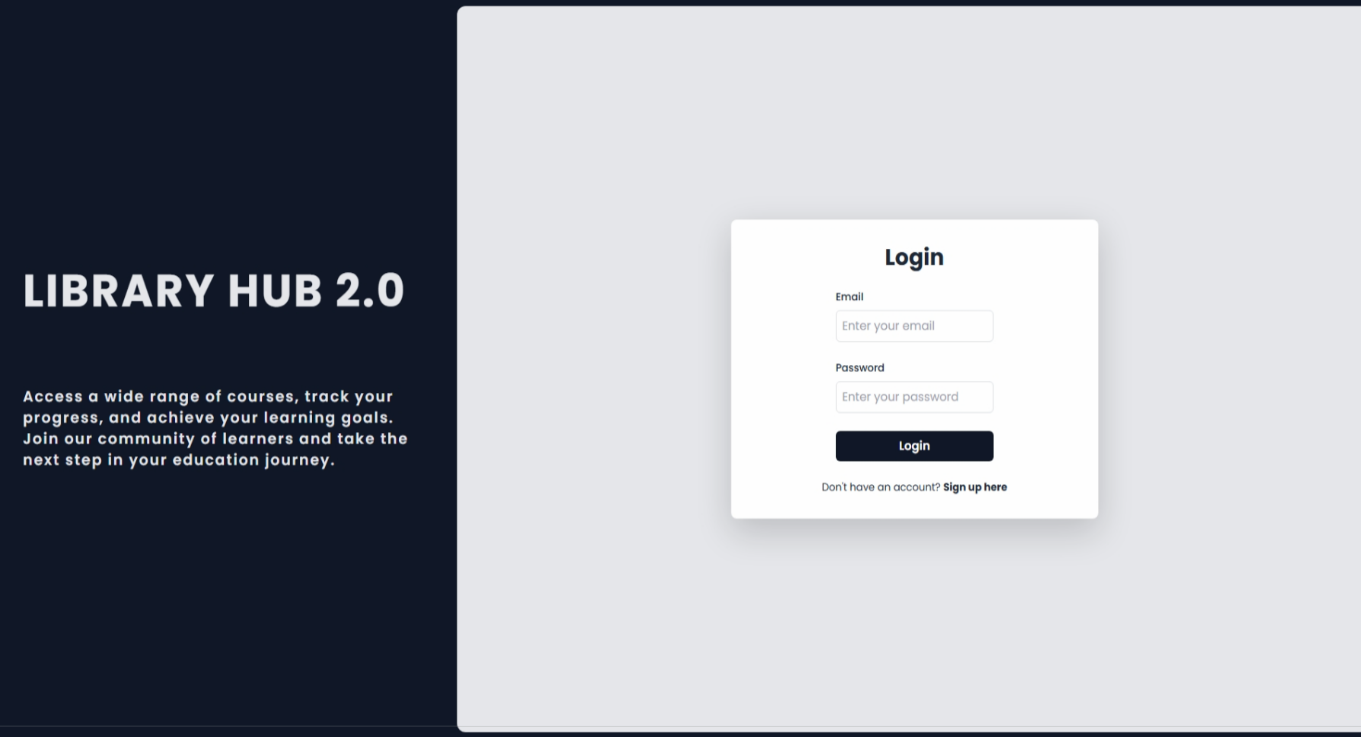
These integration tests helped identify and resolve minor issues such as data format mismatches or session inconsistencies, ensuring a robust and cohesive system.

## Limitations

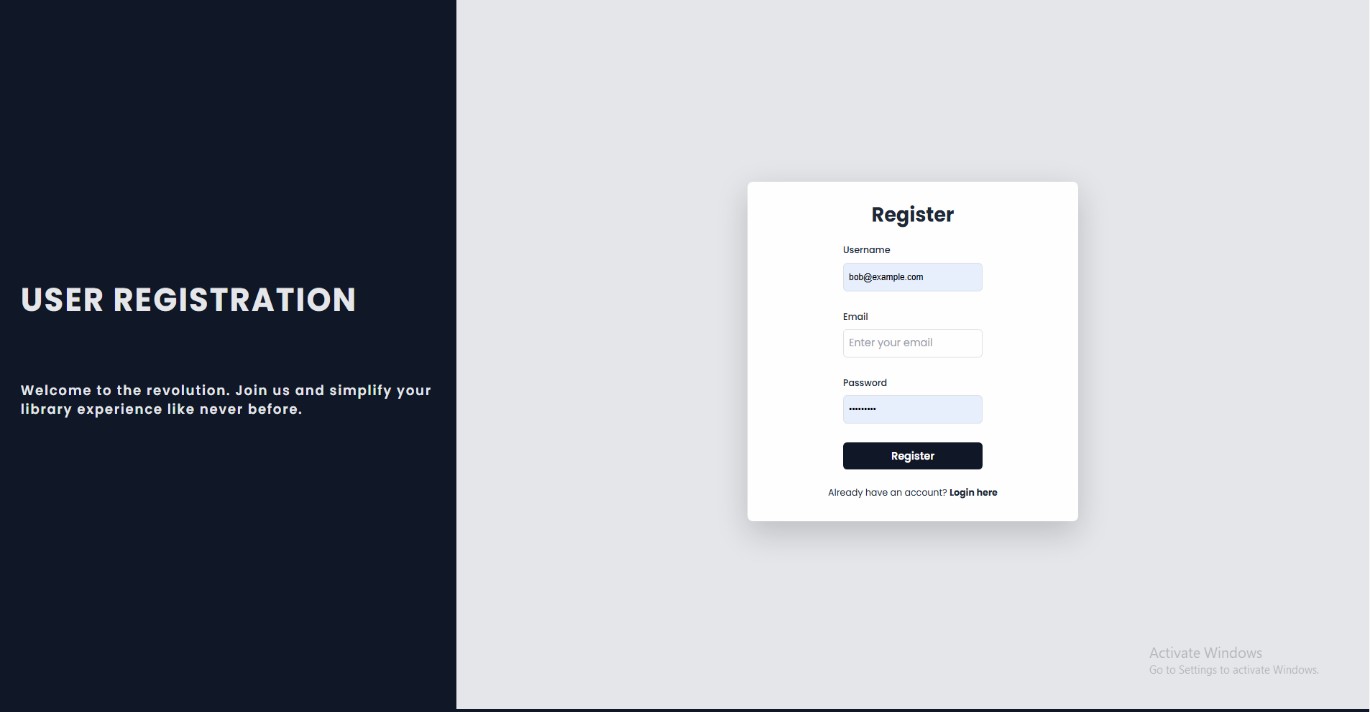
1. While functional, the current UI design could benefit from enhancements to improve aesthetic appeal and user-friendliness, especially on mobile devices.
2. The system may face challenges handling a significantly large number of users and reservations without performance optimization.
3. Features like 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 5: Snapshots

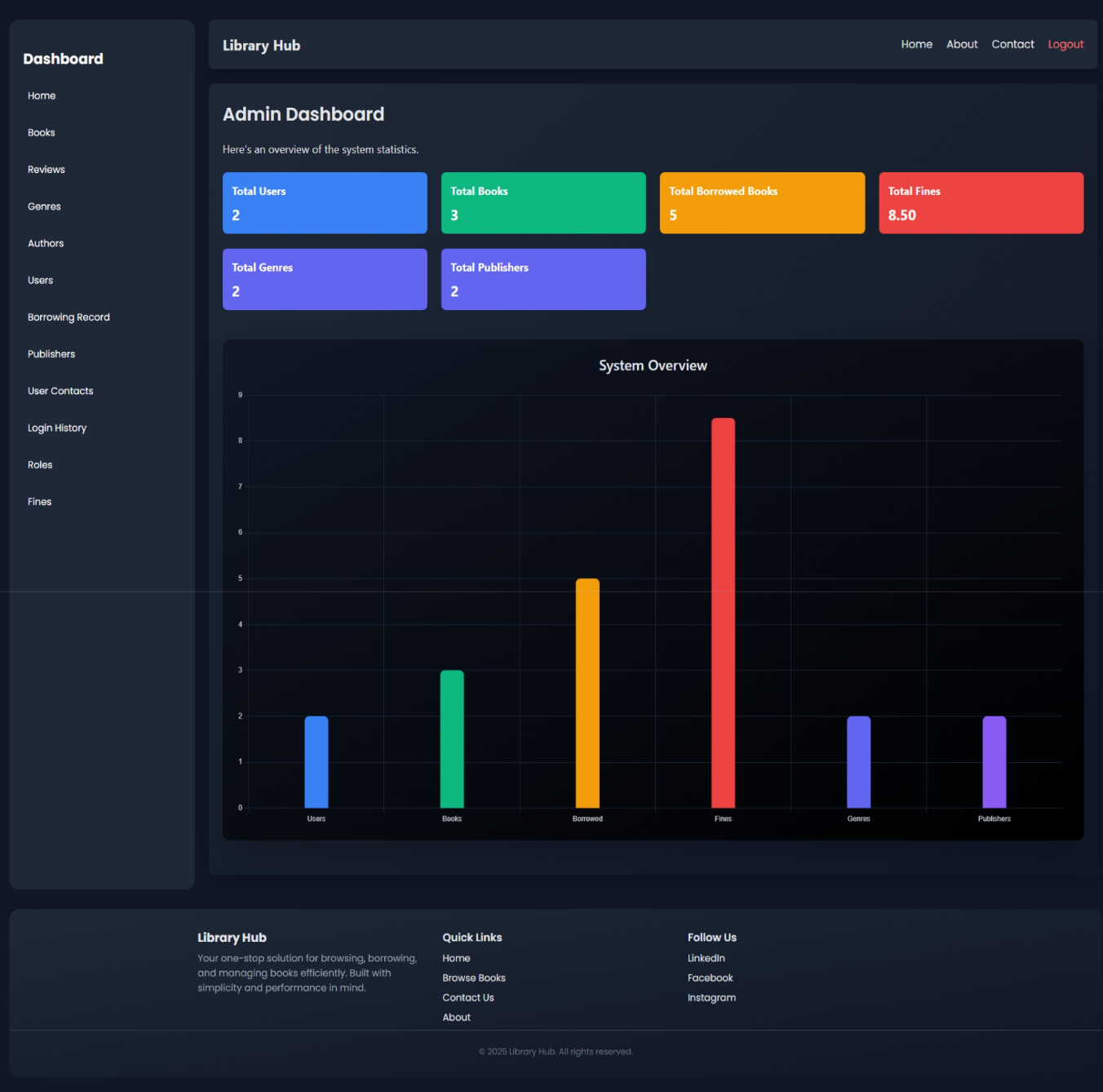
## Login Page



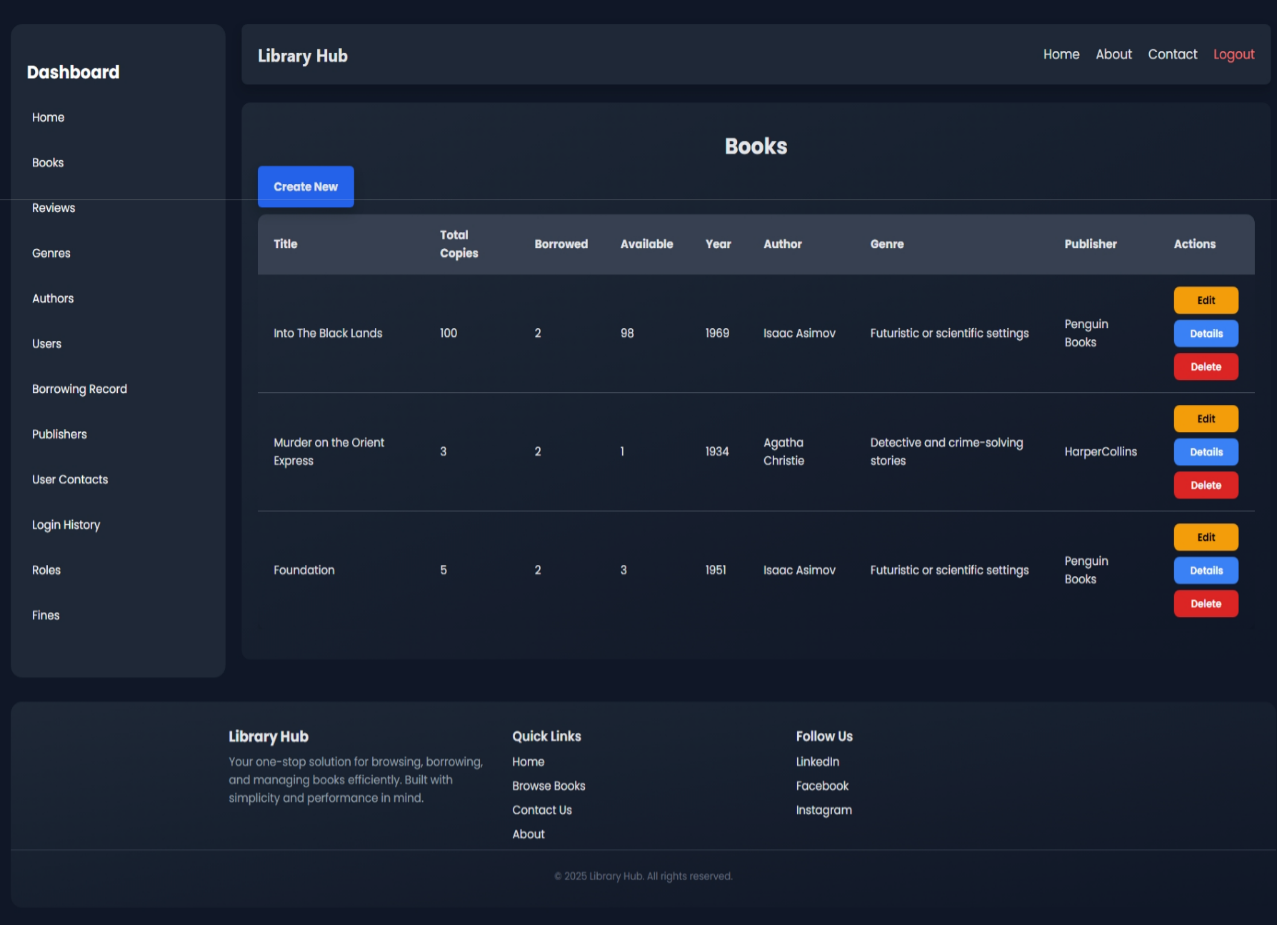
## Registration Page



## Admin Home Page



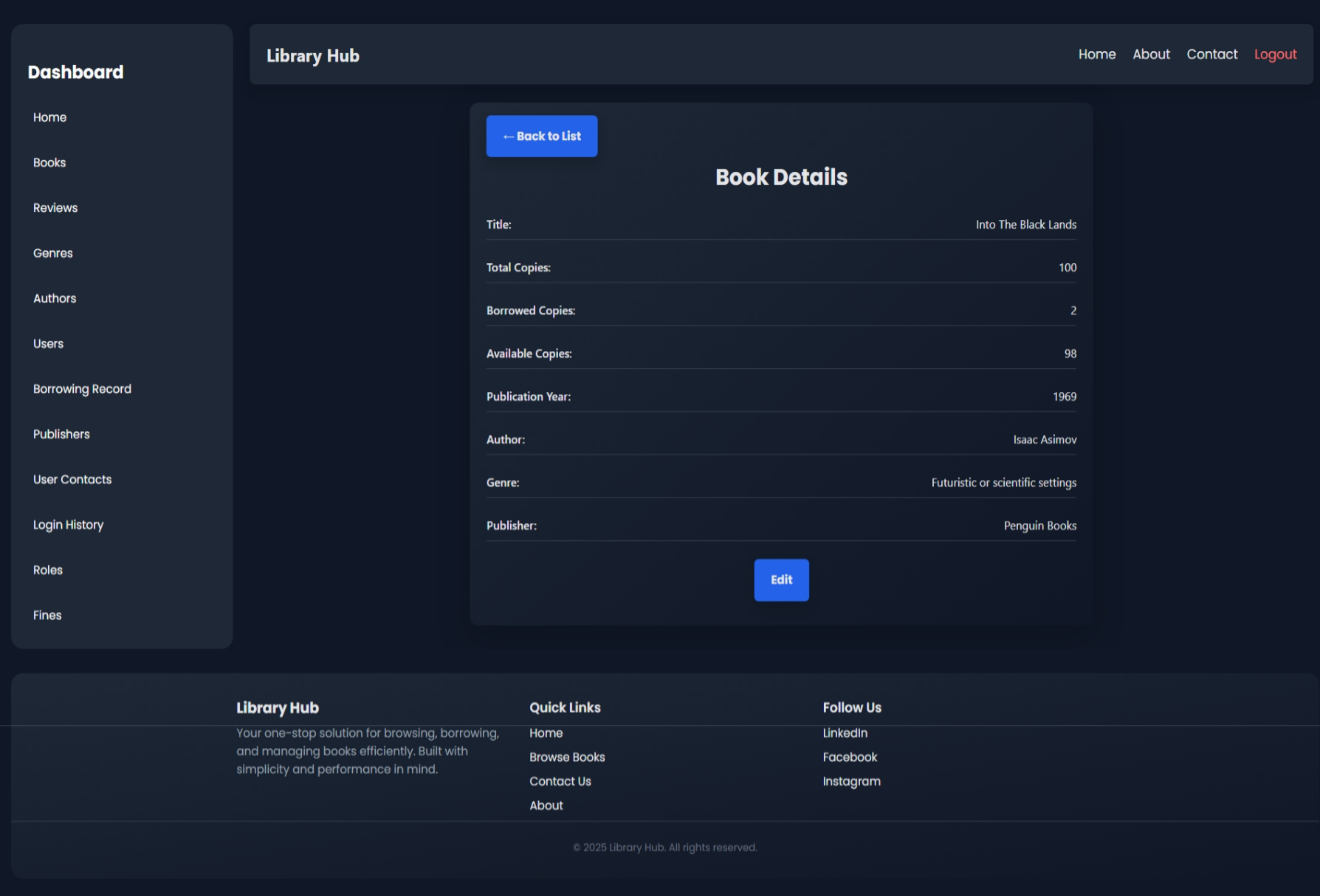
## Manage Books



## View/Edit Books

## Add New Book

## Book Details



## Book Details

For all page’s same format has been followed.

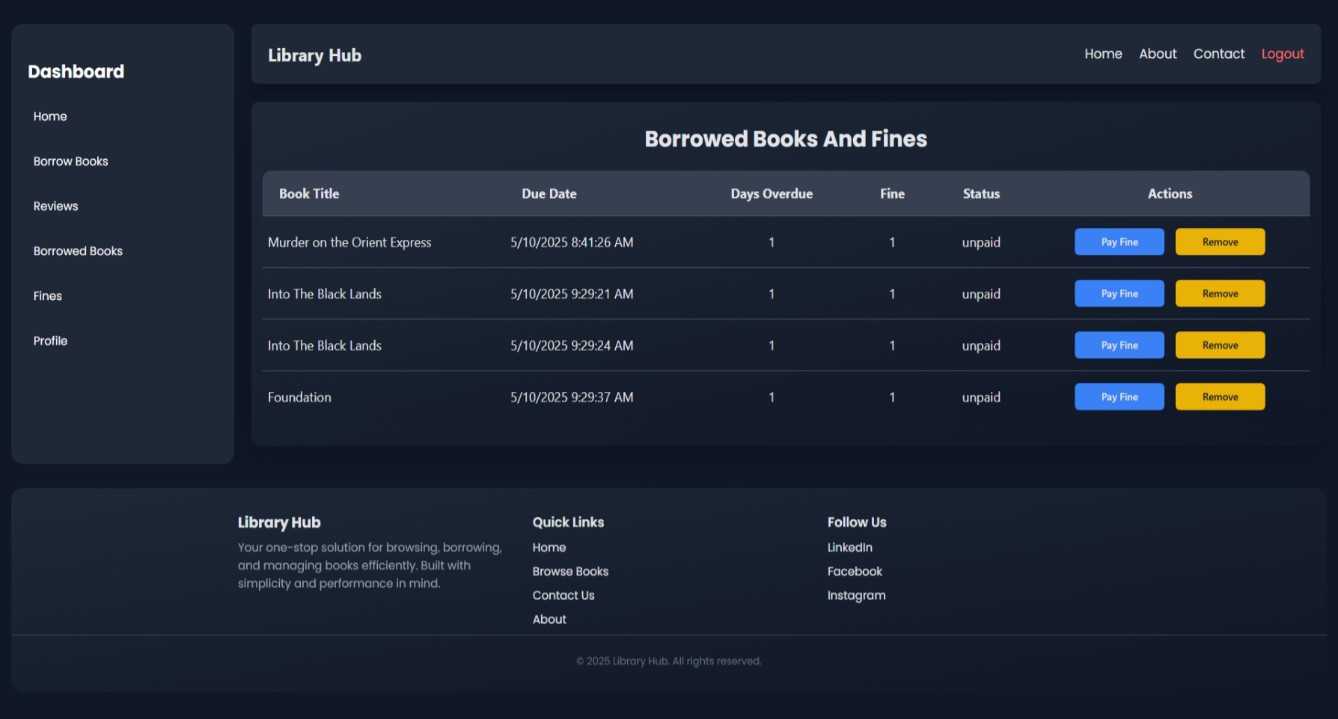
## User Home

## View Profile & Edit Profile

## Borrow Books

## Return Books

## Check/Pay Fine

****

# Chapter 6: Conclusion

The Library Management System developed as part of this project successfully meets its objective of providing an efficient, secure, and user-friendly platform for managing library operations. Utilizing ASP.NET MVC 4.7.2, Entity Framework (DB-First), and SQL Server, the system supports comprehensive functionality for both end users and administrators. Users can log in, search for books by name or author, borrow and reserve books, write reviews, manage their profiles, and

view or pay fines. Administrators have access to a full suite of tools to perform CRUD operations on users, books, genres, fines, roles, and reviews, as well as monitor system activities such as borrowing history and login records.

The system's clean architecture, role-based access control, and use of standard web development practices ensure maintainability, data integrity, and scalability. This project showcases a practical implementation of modern web technologies to streamline and digitize traditional library workflows, effectively addressing real- world library management needs.

# Chapter 7: 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. Provide language options to cater to a diverse user base, making the system accessible to non-native speakers and increasing its usability.
5. Introduce QR codes for book identification, allowing users to quickly scan and reserve books using their mobile devices.
6. Add options for paid services such as premium memberships, late fee payments, or access to exclusive book collections.