ព្រះរាជាណាចក្រកម្ពុជា
ជាតិ សាសនា ព្រះមហាក្សត្រ

SOFTWARE ENGINEERING

Group I4-GIC-B

Topic: Library Borrowing and Book Tracking System

Name of Student	ID	Score
1. THOUNG SOKET	e20220586	……
2. SOPHAL MENGCHHIV	e20221053	……
3. THOU THEARO	e20220910	……
4. IM CHHEANGNGIM	e20221459	……
5. HUY CHANCHHINGHOIR	e20221469	……

Lecturer: Mr. ROEUN Pacharoth (TP)

Academic Year: 2025-2026

Table of Content

I. Introduction

1.1 Project overview

The Library Management System is a software application developed to automate and simplify the daily operations of a library. Traditional manual systems are time-consuming, error-prone, and difficult to manage. This project provides a digital solution to efficiently handle book records, user information, and borrowing transactions.

The system allows librarians to manage books, track availability, and maintain user records in a centralized database. It also helps users to search for books and view their borrowing history. By using modern technologies and structured system design, the application improves accuracy, reduces workload, and enhances overall library services.

1.2 Problem statement

In many libraries, book management and borrowing are still done manually or with simple systems. This can cause problems such as losing information, not knowing which books are available, and difficulty tracking borrowed books.

When records are not well organized, it becomes hard for librarians to manage books and users. They may not know who borrowed a book or when it should be returned. This makes library work slow and inefficient.

Because of these problems, a simple and organized Library Management System is needed to help manage books, users, and borrowing records more easily and accurately.

1.3 Objectives

The main objectives of this project are:

- To design and develop a basic Library Management System using Spring Boot
- To manage books, users, and borrowing records effectively
- To apply MVC architecture for better code organization
- To design a clear database structure with proper relationships
- To provide a simple and user-friendly interface using Thymeleaf
- To understand the full development process from planning to implementation

II. System Analysis

2.1 Functional Requirements

Functional requirements define the core features and operations that the system must support.

- The system shall allow administrators to manage user accounts, including creating, updating, and deleting users. It shall provide authentication and authorization features to ensure that only authorized users can access specific functionalities.

- The system shall allow librarians to add, update, delete, and search for books. Each book record shall contain essential information such as title, author, category, and availability status.

- The system shall support borrowing and returning of books. When a user borrows a book, the system shall record the transaction and update the book's availability. When a book is returned, the system shall update the borrowing record accordingly.

- The system shall store and manage all data in a centralized database. It shall retrieve accurate information about users, books, and borrowing history when requested.

2.2 Non-Functional Requirements

Non-functional requirements describe how the system performs its functions and define quality attributes.

- The system shall be user-friendly, providing a simple and intuitive interface for librarians and users. All operations should be easy to understand and navigate.

- The system shall be reliable and ensure data consistency at all times. Transactions such as borrowing and returning books must be processed correctly without data loss.

- The system shall be secure by implementing authentication and role-based access control. Sensitive data such as user credentials must be protected from unauthorized access.

- The system shall be efficient and capable of handling multiple user requests with minimal response time.

- The system shall be scalable, allowing new features such as online reservations, notifications, and reports to be added in the future.

III. System Design

3.1 System Architecture Overview

The Library Management System follows a layered architecture using the Model-View-Controller (MVC) design pattern. This structure helps separate responsibilities and makes the system easier to maintain and extend.

The Controller layer handles incoming requests and user actions. Controllers such as BookController, UserController, BorrowController, and AuthController manage operations like book management, user authentication, and borrowing processes.

The Service layer contains the business logic of the system. It processes data, applies rules, and coordinates between controllers and repositories.

The Repository layer interacts with the database using Spring Data JPA. It performs CRUD operations for entities such as Book, User, Role, and BorrowRecord.

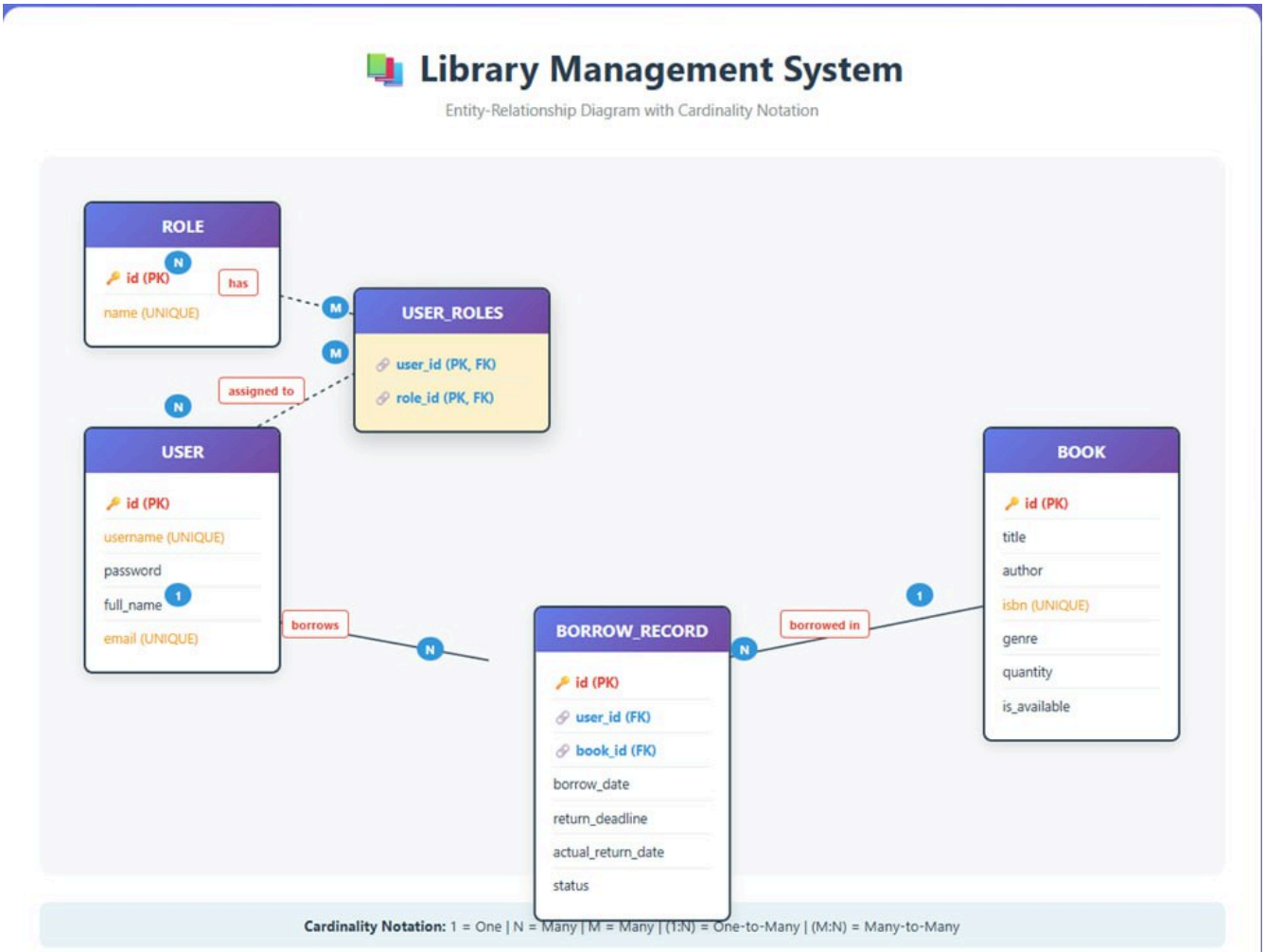
The Database layer stores all system data. Flyway migration scripts are used to manage and version database tables and default data.

Security is handled using Spring Security, which manages authentication and role-based access control.

This architecture ensures modularity, security, and reliable data management.

3.2 Database Design (ER Diagram / Tables)

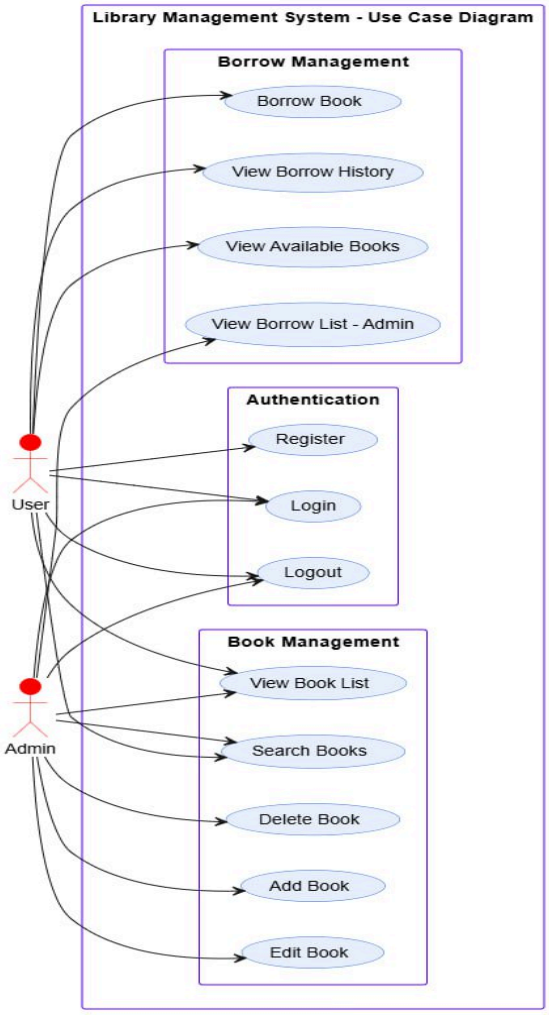
The database is designed using an Entity-Relationship (ER) diagram to clearly show data structure and relationships. The system contains five main tables: User, Role, User_Roles, Book, Borrow_Record



3.3 UML Design

Use case diagram

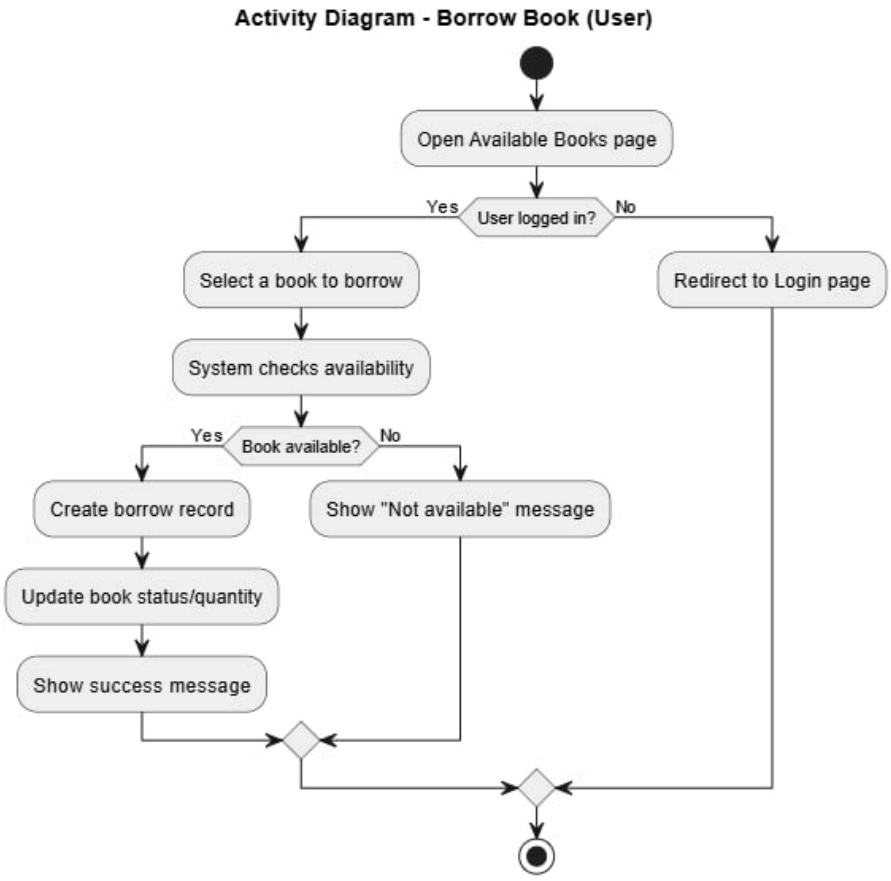
The use case diagram shows how users and administrators interact with the Library Management System. It presents the main functions of the system such as login, book management, and borrowing books. This diagram helps to understand what actions users and admins can perform in the system.



- Activity Diagram – Borrow Book (User)

The activity diagram for borrowing a book shows the step-by-step process followed by a user in the Library Management System. The process starts when the user opens the available books page. If the user is logged in, the system allows the user to select a book and checks whether the book is available.

If the book is available, the system creates a borrowing record and updates the book status. A success message is then displayed to the user. If the book is not available, the system informs the user. If the user is not logged in, the system redirects the user to the login page. This diagram helps to clearly understand the user borrowing workflow and decision points.



· Activity Diagram – Manage Books (Admin)

The activity diagram for managing books describes how an administrator performs book management tasks in the system. The process begins when the admin accesses the book management page. After successful login, the admin can choose to add, edit, or delete a book.

When adding a book, the admin fills in the book details and saves them to the database. For editing, the admin updates existing book information and saves the changes. For deleting, the admin removes a selected book from the system. After each action, the updated book list is displayed. This diagram shows how administrative actions are handled step by step in the system.

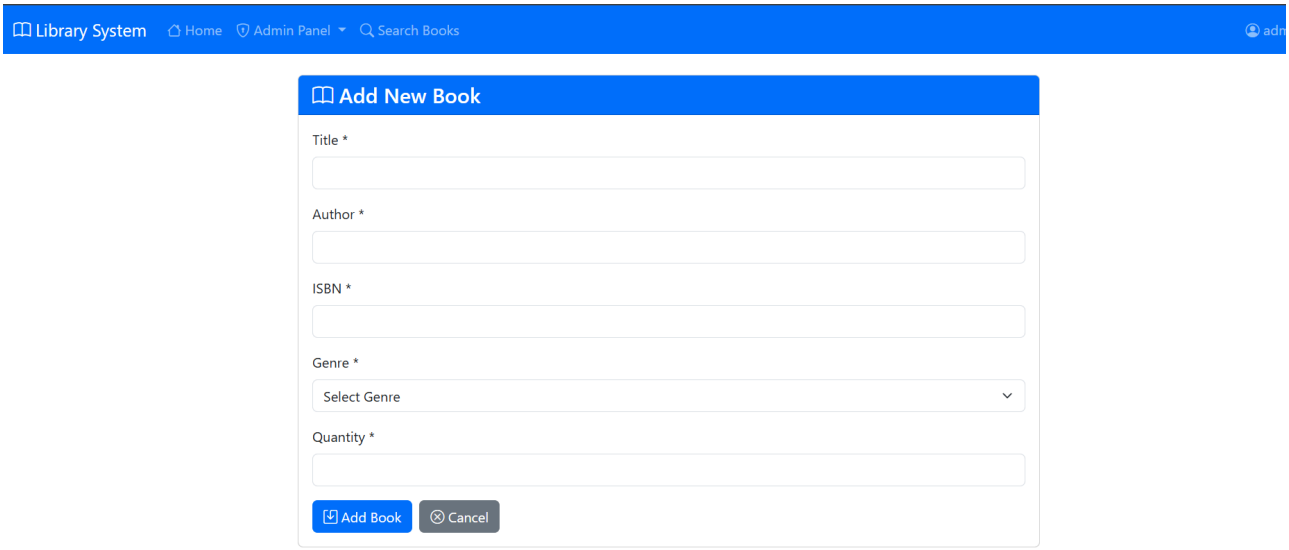
IV. Frontend Implementation (Thymeleaf Templates)

4.1 Books

4.1.1 Form

This page provides a form for administrators to add or edit book information.

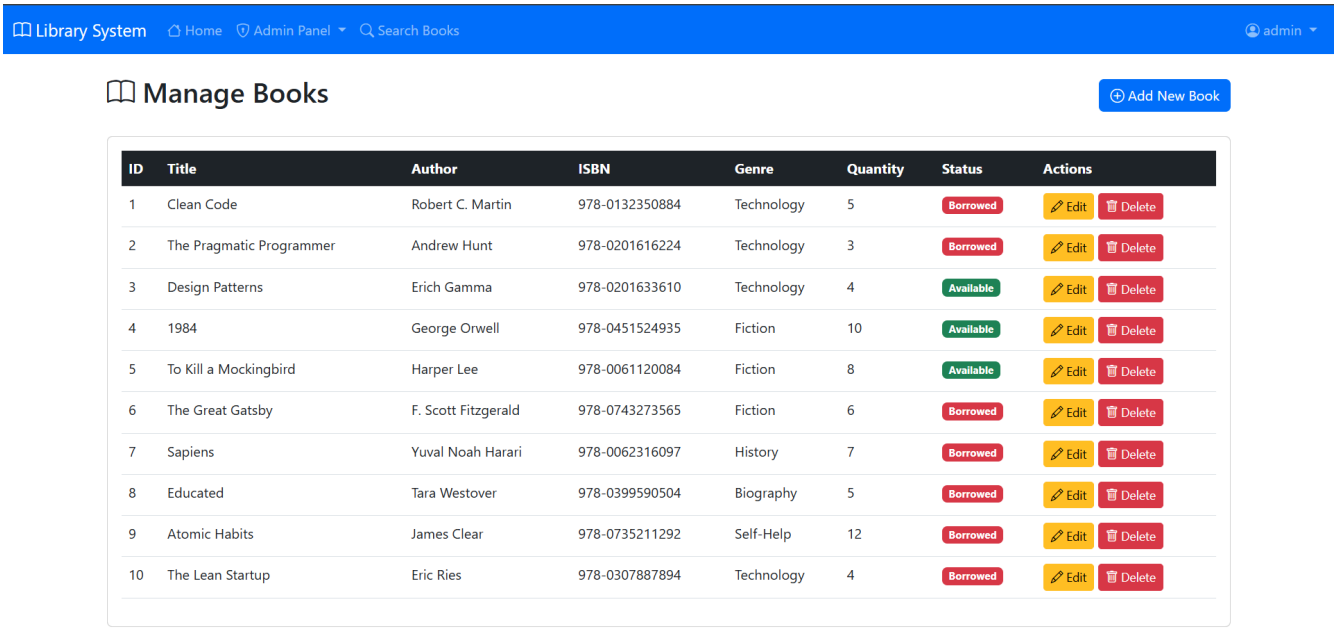
The form includes fields such as title, author, ISBN, genre, quantity, and availability status, and is implemented using Thymeleaf templates.



4.1.2 List

This page displays the list of books stored in the system in a table format.

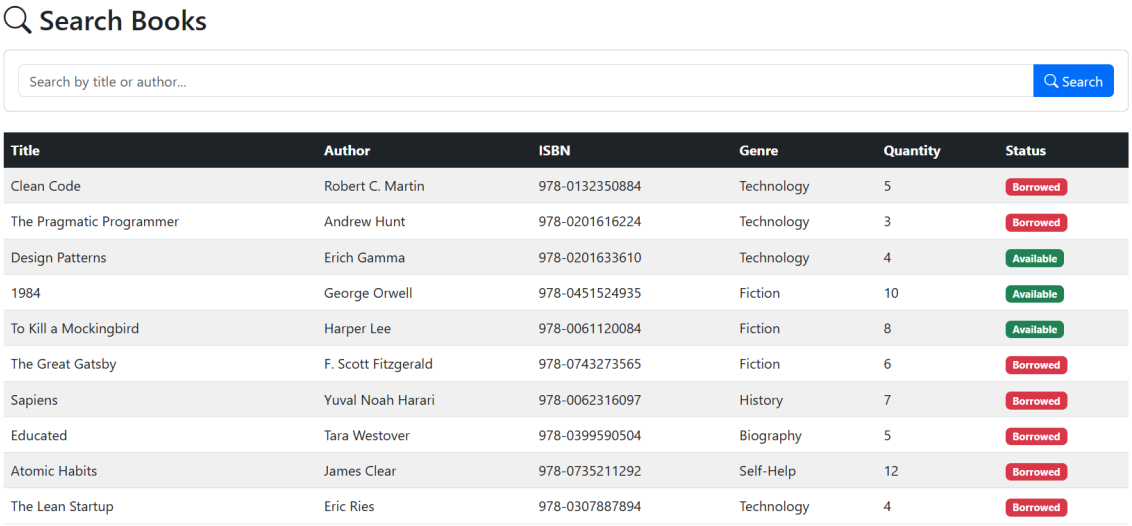
Administrators can view book details such as title, author, ISBN, quantity, status, and perform actions like edit or delete.



4.1.3 Search

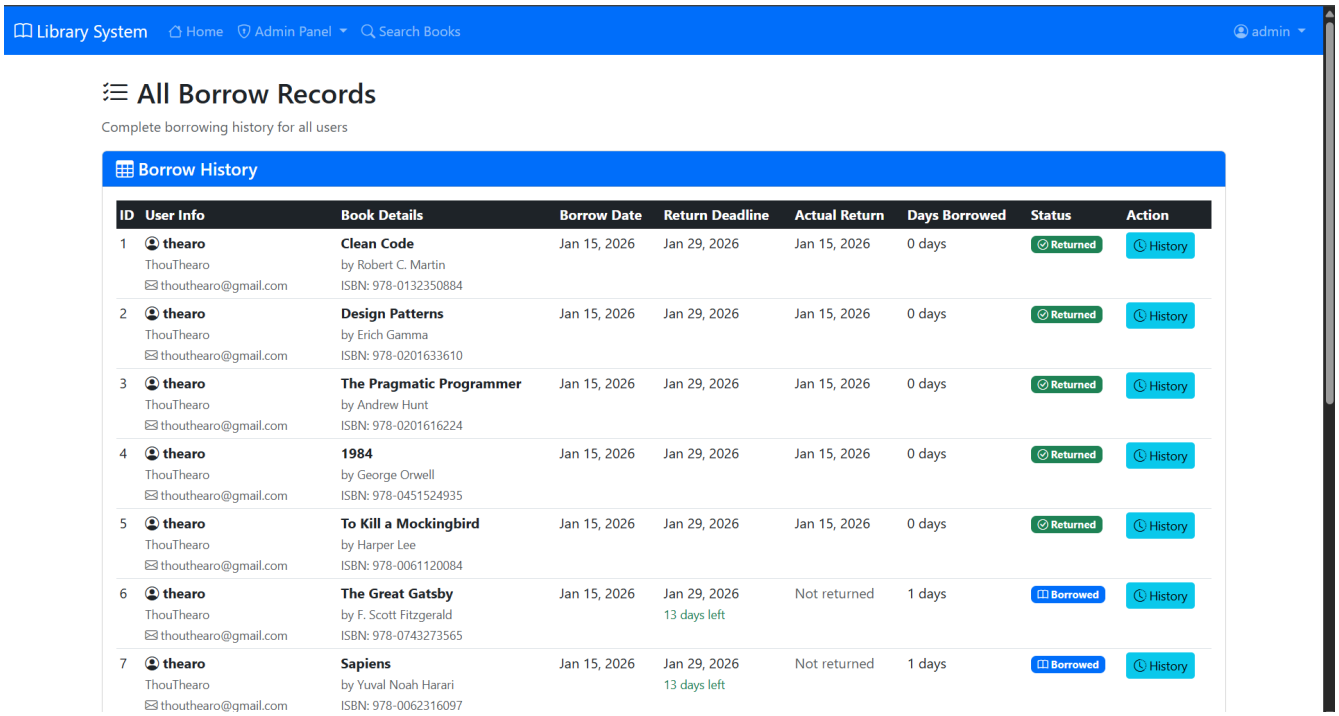
This page allows users to search for books by title or author.

The search results are displayed in a table showing book details and current availability status.

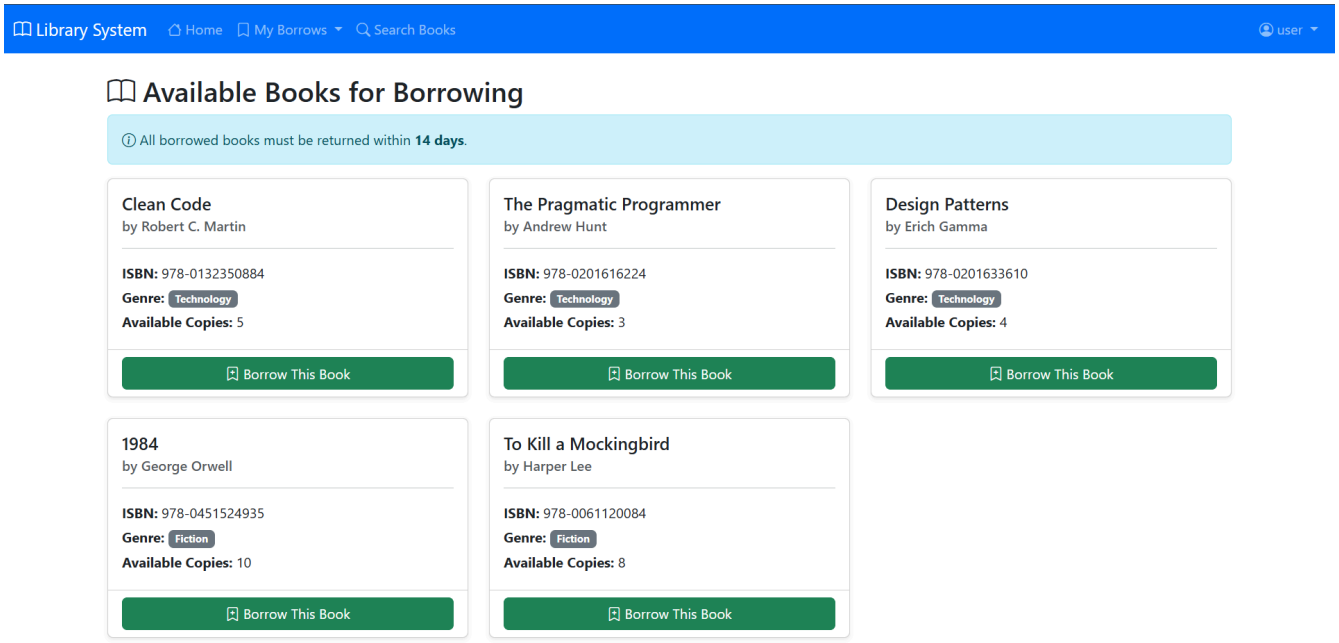


4.2 Borrows

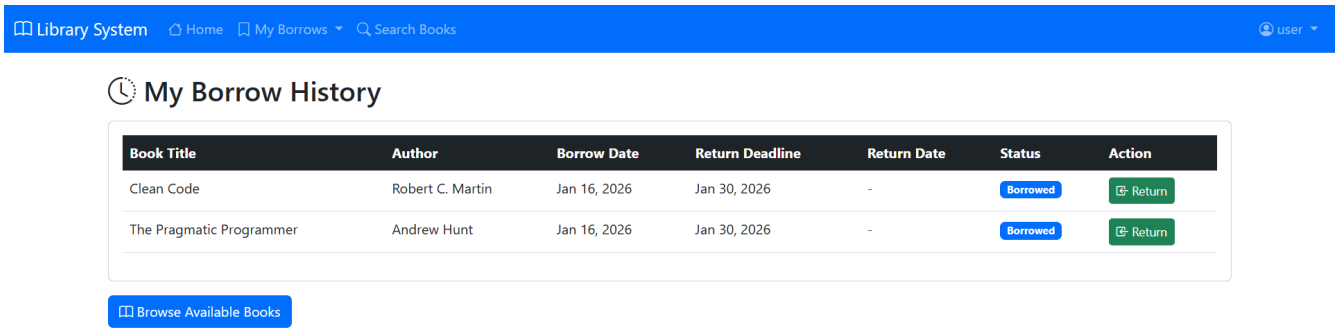
4.2.1 Admin List



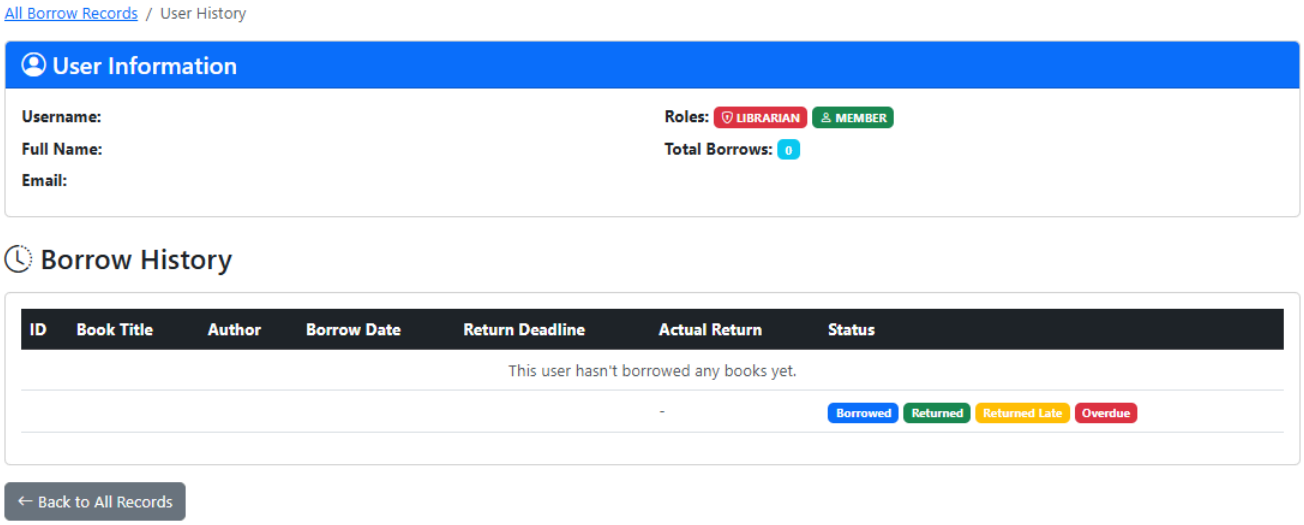
4.2.2 Available



4.2.3 History



4.2.4 User History

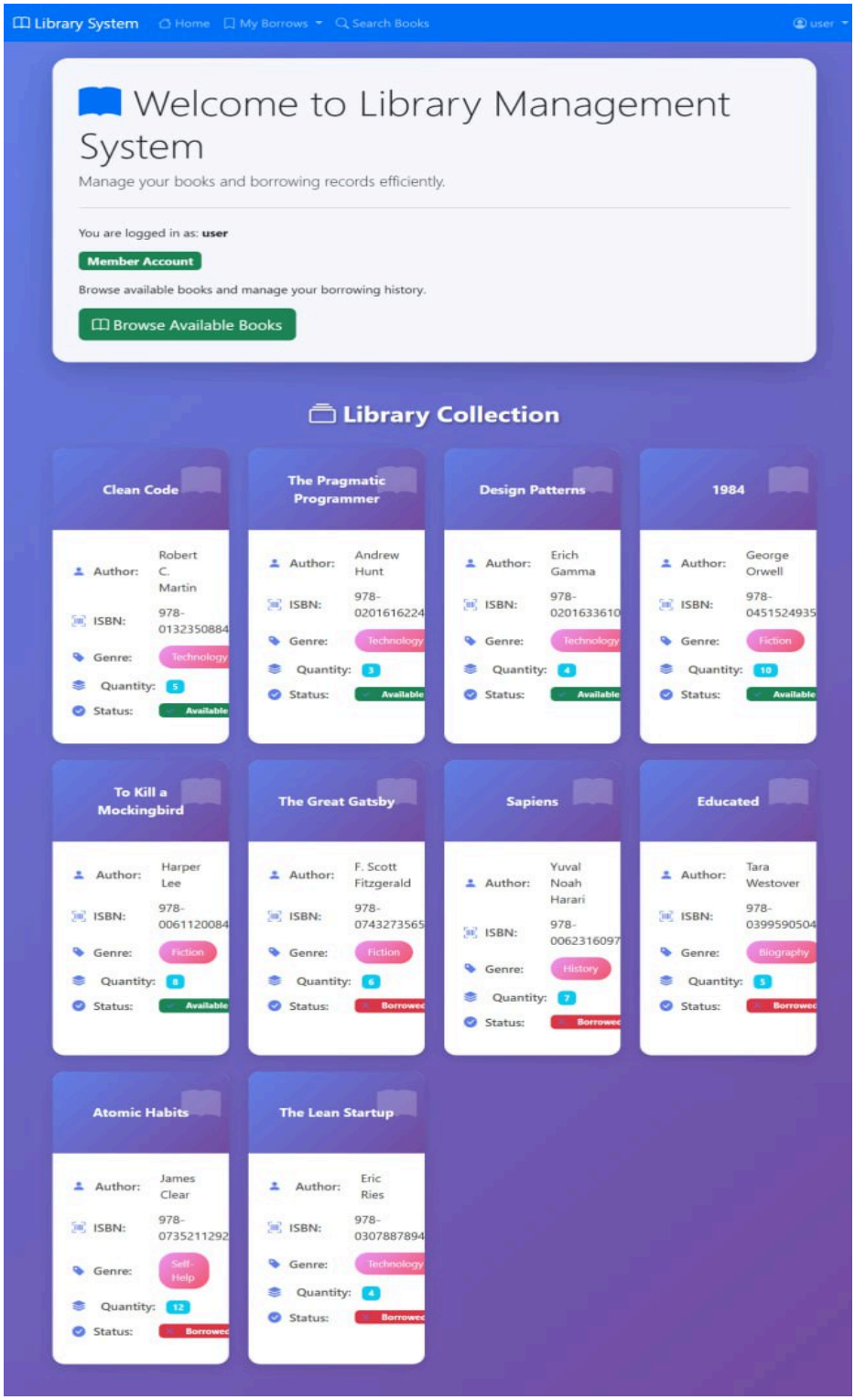


4.3 Layout and Navigation

Admin

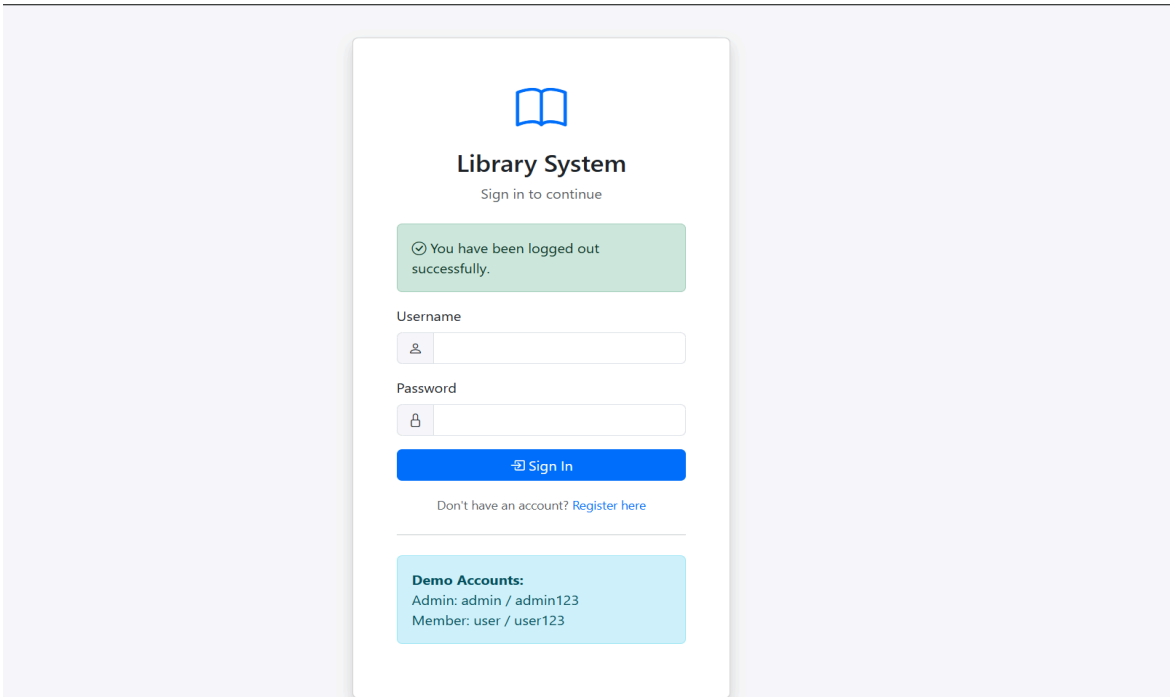
User

4.4 Index

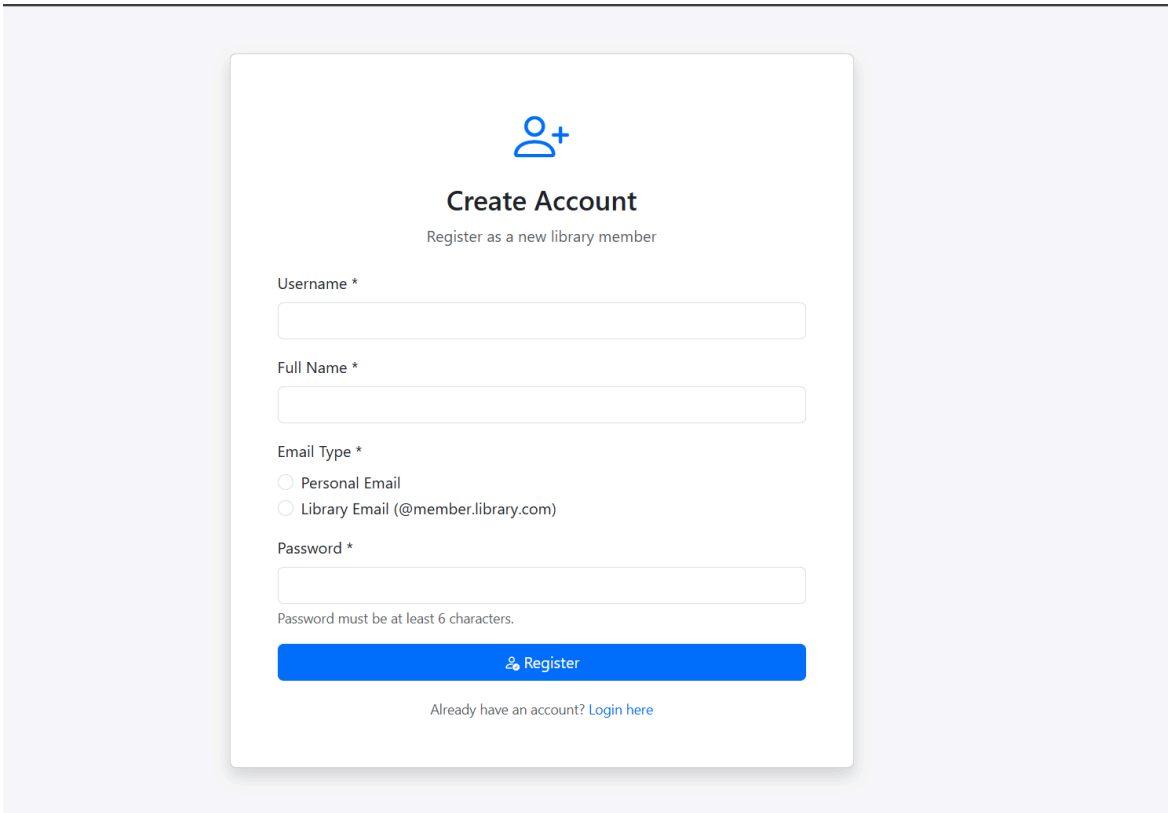

Library System
Home
My Borrows
Search Books
user
Welcome to Library Management System
Manage your books and borrowing records efficiently.
You are logged in as: user
Member Account
Browse available books and manage your borrowing history.
Browse Available Books
Library Collection
Clean Code
Author: Robert C. Martin
ISBN: 978-0132350884
Genre: Technology
Quantity: 5
Status: Available
The Pragmatic Programmer
Author: Andrew Hunt
ISBN: 978-0201616224
Genre: Technology
Quantity: 3
Status: Available
Design Patterns
Author: Erich Gamma
ISBN: 978-0201633610
Genre: Technology
Quantity: 4
Status: Available
1984
Author: George Orwell
ISBN: 978-0451524935
Genre: Fiction
Quantity: 10
Status: Available
To Kill a Mockingbird
Author: Harper Lee
ISBN: 978-0061120084
Genre: Fiction
Quantity: 8
Status: Available
The Great Gatsby
Author: F. Scott Fitzgerald
ISBN: 978-0743273565
Genre: Fiction
Quantity: 6
Status: Borrowed
Sapiens
Author: Yuval Noah Harari
ISBN: 978-0062316097
Genre: History
Quantity: 7
Status: Borrowed
Educated
Author: Tara Westover
ISBN: 978-0399590504
Genre: Biography
Quantity: 5
Status: Borrowed
Atomic Habits
Author: James Clear
ISBN: 978-0735211292
Genre: Self-Help
Quantity: 12
Status: Borrowed
The Lean Startup
Author: Eric Ries
ISBN: 978-0307887894
Genre: Technology
Quantity: 4
Status: Borrowed

4.5 Login and Register

4.5.1 Login



4.5.2 Register



V. Backend Implementation (Controllers, Dtos, Entities, Repositories , Securities, Services)

5.1 Controllers Layer

5.1.1 AuthController

The `AuthController` is responsible for managing user authentication and registration through both web-based views and RESTful API endpoints. It supports secure user onboarding, login, and token-based authentication using JWT.

Key Responsibilities:

Rendering login and registration pages for web users

Handling user registration with input validation and duplicate checks

Restricting registration based on email domain rules

Securely encoding user passwords before persistence

Assigning the default MEMBER role to newly registered users

Authenticating users and generating JWT tokens for API access

Providing structured error responses for authentication failures

The controller leverages Spring MVC annotations for request mapping, Spring Security for authentication and password encryption, and JWT for stateless API authentication. It interacts directly with the repository layer to manage user and role data while enforcing security and validation rules.

5.1.2 BookController

The `BookController` manages all book-related HTTP requests for both web-based views and RESTful API endpoints. It facilitates the display, creation, modification, deletion, and searching of books while supporting role-based access for administrative operations.

Key Responsibilities:

Displaying book listings on the public homepage and administrative pages

Providing forms for creating and editing book records

Handling CRUD operations for books with input validation and user feedback

Supporting keyword-based book search functionality

Exposing REST API endpoints for administrative book management

Managing success and error messages using redirect attributes

The controller uses Spring MVC annotations to map request endpoints and validate input data. All business logic related to book management is delegated to the `BookService`, ensuring a clear separation of concerns between request handling, validation, and application logic.

5.1.3 BorrowController

The `BorrowController` manages the borrowing and returning workflows for books in the library system.

Responsibilities:

- Displaying available books
- Allowing authenticated users to borrow and return books

- Displaying borrowing history

Borrowing rules and validations are enforced through the `BorrowService` to ensure consistency and data integrity.

5.1.4 UserController

The `UserController` provides administrative functionality related to user management within the system.

Responsibilities:

- Viewing individual user borrowing history
- Supporting administrative monitoring and management

This controller interacts with the service layer to retrieve users and borrow data securely.

5.2 Data Transfer Objects (DTOs)

The Data Transfer Object (DTO) layer is used to encapsulate and validate data exchanged between the client and the server. DTOs help decouple the presentation layer from the domain entities, ensuring controlled data flow, improved security, and centralized input validation.

All DTO classes use Lombok annotations to reduce boilerplate code and Jakarta Bean Validation to enforce data integrity.

5.2.1 BookRequest

The `BookRequest` DTO is used when creating or updating book records. It ensures that all required book attributes are provided and valid before processing.

Responsibilities:

- Transfer book information from client requests to the application
- Enforce mandatory fields and validation constraints
- Prevent invalid or incomplete book data from reaching the service layer

Validation Rules:

- `title`, `author`, `isbn`, and `genre` must not be blank
- `quantity` must be provided and must be at least 1

5.2.2 BorrowRequest

The `BorrowRequest` DTO is used to handle book borrowing operations. It contains only the necessary data required to identify the book being borrowed.

Responsibilities:

- Transfer the book identifier during borrow requests
- Ensure the presence of a valid book ID

Validation Rules:

- `bookId` must not be null

5.2.3 ErrorResponse

The `ErrorResponse` DTO provides a standardized structure for returning error information to API clients.

Responsibilities:

- Encapsulate error details in a consistent response format
- Improve API error readability and debugging

Fields:

- `message` – Detailed description of the error
- `error` – Short error title or category
- `status` – HTTP status code

5.2.4 JwtAuthResponse

The `JwtAuthResponse` DTO is used to return authentication results after successful login or registration.

Responsibilities:

- Deliver JWT tokens to authenticated users
- Provide authentication-related metadata

Fields:

- `token` – Generated JWT token
- `type` – Token type (`Bearer`)
- `username` – Authenticated user identifier
- `message` – Authentication status message

5.2.5 LoginRequest

The `LoginRequest` DTO captures user credentials during authentication.

Responsibilities:

- Transfer login credentials securely
- Validate authentication input before processing

Validation Rules:

- `username` must not be blank
- `password` must not be blank and contain at least 6 characters

5.2.6 RegisterRequest

The `RegisterRequest` DTO is used for new user registration and enforces strict validation rules to ensure data correctness.

Responsibilities:

- Transfer user registration data
- Enforce constraints for secure and valid user creation

Validation Rules:

- `username`: 3–50 characters
- `fullName`: 2–100 characters
- `email`: valid email format
- `password`: minimum 6 characters

5.3 Entities Layer

The Entities layer represents the data model of the system and defines how application objects are mapped to database tables. In this project, the Entities layer is implemented using the Java Persistence API (JPA) with Hibernate as the Object–Relational Mapping (ORM) framework. Each entity class is

annotated with `@Entity` and `@Table` to define the mapping between Java objects and relational database tables.

The entities encapsulate the core domain objects of the library management system, including users, roles, books, and borrowing records.

5.3.1 Book Entity

The `Book` entity represents a book available in the library system and is mapped to the `books` table in the database.

Key Attributes and Annotations:

- `@Id` and `@GeneratedValue` are used to define the primary key (`id`) with an auto-increment strategy
- Fields such as `title`, `author`, `isbn`, `genre`, and `quantity` are marked as `@Column(nullable = false)` to enforce data integrity
- The `isbn` field is defined with `unique = true` to prevent duplicate book entries
- The `isAvailable` attribute indicates whether a book can be borrowed and is initialized with a default value of `true`

Purpose:
This entity stores all essential information related to books and supports inventory management within the library system.

5.3.2 BorrowRecord Entity

The `BorrowRecord` entity represents the borrowing transaction between a user and a book and is mapped to the `borrow_records` table.

Key Attributes and Annotations:

- Each borrow record has a unique primary key (`id`)
- A `@ManyToOne` relationship is defined with both `User` and `Book`, allowing multiple borrow records to be associated with a single user or book
- Fetch type is set to `EAGER` to ensure related user and book data is loaded immediately

- Date fields (`borrowDate`, `returnDeadline`, and `actualReturnDate`) track borrowing and return timelines
- The `status` field indicates the borrowing state (e.g., BORROWED, RETURNED, OVERDUE)

Purpose:
This entity enables tracking of borrowing history, due dates, and return status, ensuring proper enforcement of library borrowing rules**.**

5.3.3 Role Entity

The `Role` entity defines user roles within the system and is mapped to the `roles` table.

Key Attributes and Annotations:

- The `name` field is unique and non-null, ensuring each role is distinct
- Roles such as `MEMBER` and `ADMIN` are used to control access permissions

Purpose:
This entity supports role-based access control, allowing the system to differentiate between regular users and administrators.

5.3.4 User Entity

The `User` entity represents registered users of the library system and is mapped to the `users` table.

Key Attributes and Annotations:

- Unique constraints are applied to `username` and `email` to prevent duplicate accounts
- The `password` field stores encrypted passwords for security
- A `@ManyToMany` relationship is established with the `Role` entity through a join table named `user_roles`
- The relationship is eagerly fetched to ensure user roles are readily available during authentication and authorization

Purpose:
This entity stores user account information and supports authentication, authorization, and user activity tracking within the system.

5.4 Repositories Layer

The Repositories layer is responsible for data access and persistence operations. It acts as an abstraction between the Service layer and the database. In this project, Spring Data JPA is used to simplify database interactions by extending the `JpaRepository` interface.

Each repository interface provides built-in CRUD operations while also defining custom query methods using Spring Data JPA's method naming conventions. Screenshots of repository implementations are included to demonstrate how database queries are defined and managed.

5.4.1 BookRepository

The `BookRepository` interface manages database operations related to the `Book` entity.

Custom query methods support:

- Searching books by ISBN
- Retrieving available books
- Searching books by title or author using case-insensitive matching

5.4.2 BorrowRepository

The `BorrowRepository` interface handles database operations related to borrowing transactions represented by the `BorrowRecord` entity.

Custom query methods allow:

- Retrieving all borrow records for a specific user
- Filtering borrow records by borrowing status
- Displaying borrowing history in descending order by borrow date

5.4.3 RoleRepository

The `RoleRepository` interface manages role-related data stored in the `roles` table.

It provides functionality to retrieve roles based on their names, which is essential during user registration and authorization processes.

5.4.4 UserRepository

The `UserRepository` interface manages persistence operations for the `User` entity.

Custom query methods include:

- Retrieving users by username
- Checking for existing usernames
- Checking for existing email addresses

5.5 Security and Authentication

The system implements JWT-based authentication using Spring Security to secure API requests and manage user identity. Authentication is handled in a stateless manner, ensuring scalability and secure access control.

5.5.1 JWT Authentication Filter

The `JwtAuthenticationFilter` is responsible for intercepting incoming HTTP requests and authenticating users based on the presence of a JWT token.

Responsibilities:

- Extract the JWT token from the `Authorization` request header
- Validate the token before authentication
- Retrieve the username from the token
- Load user details using `UserDetailsService`
- Set the authenticated user in the Spring Security context

This filter ensures that only requests with valid JWT tokens are authenticated before accessing protected resources.

5.5.2 JWT Token Provider

The `JwtTokenProvider` handles all operations related to JWT token management.

Responsibilities:

- Generate JWT tokens during login or registration
- Extract usernames from tokens
- Validate token integrity and expiration
- Sign tokens using a secret key and the HS512 algorithm

The provider ensures that only valid, signed, and unexpired tokens are accepted by the system.

5.6 Services Layer

The Services layer contains the core business logic of the application. It acts as an intermediary between the Controller layer and the Repository layer, ensuring that business rules and application workflows are properly enforced.

Spring's `@Service` annotation is used to define service components that coordinate data access operations through repository interfaces. Screenshots of service implementations are included to demonstrate how business logic is structured and executed within the system.

5.6.1 BookService

The `BookService` class manages business logic related to book management.

Responsibilities include:

- Retrieving all books and available books
- Adding new books to the system
- Updating existing book information
- Deleting book records
- Searching books by title or author

5.6.2 BorrowService

The `BorrowService` class handles the borrowing and returning processes of books.

Key responsibilities include:

- Validating book availability before borrowing
- Creating borrowing records with predefined return deadlines
- Updating borrowing status upon book return

- Managing borrowing history for users

5.6.3 CustomUserDetailsService

The `CustomUserDetailsService` class integrates Spring Security into the application by implementing the `UserDetailsService` interface.

It supports:

- Retrieving user details based on username
- Mapping user roles to Spring Security authorities
- Enabling role-based access control

VI. System Workflow and Data Flow

7.1 User Workflow

The general workflow of the system is as follows:

1. The user opens the web application in a browser.
2. The user logs in or registers through the login page.
3. The system verifies credentials using the authentication controller.
4. If authentication is successful, the user is redirected to the dashboard.
5. The user performs actions based on their role:
    - Admin: manage users, roles, and books
    - Librarian: manage books and borrowing records
    - Member: search books, borrow books, view history
6. Each action sends a request to the appropriate controller.
7. The controller calls the service layer.

8. The service layer communicates with the database through repositories.
9. The response is returned to the controller.
10. The controller sends the result to the UI (Thymeleaf page or JSON).

7.2 Login and Authentication Workflow

This workflow exists in your project through AuthController, JwtTokenProvider, and SecurityConfig.

Steps:

1. The user enters username and password on the login page.
2. The browser sends a POST request to the AuthController.
3. The controller validates the credentials.
4. If valid, a JWT token is generated.
5. The token is returned to the client.
6. The client includes the token in future requests.
7. Spring Security verifies the token before allowing access.

DATA FLOW:

User → Login Page → AuthController → Security Layer → Database → Token Generated → User Dashboard

7.3 Book Management Workflow

This workflow is handled by BookController, BookService, and BookRepository.

Steps:

1. User navigates to the book list page.
2. A GET request is sent to BookController.

3. BookController calls BookService.
4. BookService retrieves data using BookRepository.
5. Data is fetched from the database.
6. The list of books is sent back to the controller.
7. The controller passes the data to the Thymeleaf template.
8. The UI displays the book list.

DATA FLOW:

User → Book Page → BookController → BookService → BookRepository → Database

← Thymeleaf View ← Controller ← Service ← Repository ← Database

7.4 User and Role Management Workflow

This workflow is handled by UserController, UserService, and UserRepository.

Steps:

1. Admin accesses the user management page.
2. A request is sent to UserController.
3. UserService processes the logic.
4. UserRepository retrieves or updates data.
5. Changes are saved in the database.
6. The updated user list is displayed.

DATA FLOW:

Admin → User Page → UserController → UserService → UserRepository → Database

VII. Challenges and Lessons Learned

Challenges

- Designing the database structure and defining correct relationships between users, books, and borrowing records.
- Ensuring data consistency and proper connections between tables.
- Understanding the Spring Boot architecture.
- Learning how controllers, services, and repositories work together.
- Implementing security and managing user roles.

Lessons Learned

- The importance of proper system planning before implementation.
- How to design a modular system using MVC architecture.
- The value of testing during development.
- Improved understanding of backend development using Spring Boot.
- Gained experience in database management and secure system implementation.

VIII. Conclusion and Future Enhancements

Conclusion

- The Library Management System provides a digital solution for managing books, users, and borrowing records.
- The system replaces manual record-keeping with an automated process.
- It improves accuracy, reliability, and ease of management.
- The project uses Spring Boot, Spring Security, and a relational database.

- It demonstrates a real-world backend system with proper architecture and basic security.

Future Enhancements

- Add online book reservation functionality.
- Implement email notifications for borrowing and returning books.
- Add fine calculation for late book returns.
- Generate advanced reports for library management.
- Develop a full frontend interface to improve user interaction.
- Use Docker for deployment.