Library Management System

Anusha Annengala

April 2, 2025

Contents

1 Introduction

Library Management System is a role-based RESTful application built with Java 21, Spring Boot 3, Spring Security, H2 (file-based), and Docker. The system allows librarians to manage books and view borrow histories, and members to borrow/return books and search/view books.

2 Functional Requirements

The Library Management System is designed to support the following key functionalities, based on user roles and use cases:

2.1 User Management

- The system shall allow the registration of new members and librarians.
- The system shall enforce unique usernames and securely store passwords using encoding.
- The system shall restrict actions based on user roles (Librarian, Member).

2.2 Book Management (Librarian Only)

- The system shall allow librarians to add new books to the catalog.
- The system shall prevent adding books with duplicate ISBNs.
- The system shall allow librarians to update book details.
- The system shall allow librarians to delete books.
- The system shall validate required fields such as title, author, and ISBN.

2.3 List Books and Search Books

- The system shall allow all users to list all books.
- The system shall allow all users to search for books by title or author.
- The system shall allow all users to view book details by ID.

2.4 Borrow Books and Return Books

- The system shall allow all users to borrow and return books.
- The system shall validate book availability before borrowing.
- The system shall update the book status to unavailable once borrowed and available once returned.

2.5 Borrow History (Librarian Only)

- The system shall allow librarians to view borrow history by member.
- The system shall allow librarians to view the history of borrowed books.

3 Design Overview

3.1 Architecture

The application is structured using a standard layered architecture:

- **Controller Layer:** REST API endpoints for interaction
- **Service Layer:** Business logic and rule enforcement
- **Repository Layer:** Interacts with the database using Spring Data JPA.
- **Model Layer:** Core domain models representing Users, Books, and Borrow Records

3.2 Entities

3.2.1 User

Represents a user of the system, who can either have a role Librarian or a role Member.

- Librarians can manage books, view borrowing history and all member priviliges
- Members can view books, search for books, borrow and return books

Key Fields:

- `username` (unique login ID)
- `password` (securely hashed)
- `role` (ROLE_MEMBER or ROLE_LIBRARIAN)
- `firstName`, `lastName`, `email`, `phoneNumber`

Relationships:

- **One-to-Many with BorrowRecord:** A user can have multiple borrowing records

3.2.2 Book

Represents a physical book in the library. Each book can be borrowed by a member, subject to availability.

Key Fields:

- `title`, `author`
- `isbn` (unique identifier)
- `publicationYear`
- `available` (boolean to indicate current availability)

Relationships:

- **One-to-Many with BorrowRecord:** A book can be borrowed multiple times

3.2.3 BorrowRecord

Logs each borrowing event, tying together a member, a book, and a librarian.

Key Fields:

- `member`: the user who borrows the book
- `librarian`: the librarian who issued the book
- `book`: the book being borrowed
- `borrowDate` and `returnDate`
- `returned`: flag to indicate if the book is returned

Relationships:

- **Many-to-One with User and Book:** Multiple borrow records can reference the same user and book

3.3 Security

- Spring Security with in-database authentication
- Roles: `ROLE_LIBRARIAN`, `ROLE_MEMBER`
- Passwords are hashed using BCrypt
- Endpoint access is controlled via roles

3.4 Design Choices

The development of the Library Management System involved a series of important design decisions.

3.4.1 Architecture

- **Layered Architecture:** The system follows a standard *Controller-Service-Repository* layered pattern to enforce separation of concerns and facilitate unit testing.
- **Spring Boot:** Chosen for its rapid development support, auto-configuration, and mature ecosystem for REST APIs and security.
- **Entity Modeling:** The domain is modeled with three primary entities: `User`, `Book`, and `BorrowRecord`. These reflect the core operations and real-world interactions within a library system.

3.4.2 Authentication and Roles

- **In-Database Authentication:** User credentials are stored in the database and authenticated via Spring Security. Passwords are securely hashed using `BCryptPasswordEncoder`.
- **Role-Based Access Control:** Two roles are defined:
  - `ROLE_LIBRARIAN` - Full access to manage books and borrowing history.
  - `ROLE_MEMBER` - Can view/search books and initiate borrow/return operations.

3.4.3 Single User Entity with Roles

The system uses a single `User` entity with a `role` attribute (`ROLE_MEMBER` or `ROLE_LIBRARIAN`) instead of separate `Member` and `Librarian` entities. This approach reduces redundancy, eases role-based access control using Spring Security, and keeps entity relationships like `BorrowRecord` straightforward, as both members and librarians can be referenced from the same user model.

4 Development

The Library Management System was developed using a modern Java-based tech stack with a focus on clean architecture, separation of concerns, and security. The system is designed to support both functional extensibility and ease of deployment.

4.1 Technology Stack

- **Backend:** Java 21 with Spring Boot 3.4.4
- **Build Tool:** Maven
- **Database:** H2 (file-based, persistent across Docker restarts using a named volume)

- **Authentication:** Spring Security with in-database authentication
- **Testing:** JUnit 5, Mockito, Spring MockMvc for integration testing
- **Deployment:** Docker, Docker Compose

4.2 Development Workflow

The development followed a modular and test-driven approach:

1. Initial setup with Spring Boot starter project and required dependencies.
2. Entity modeling for `User`, `Book`, and `BorrowRecord` using JPA annotations.
3. Implementation of CRUD operations and business logic in services and controllers.
4. Role-based access control was introduced via Spring Security.
5. Exception handling and validation added to improve robustness.
6. Postman collections used for API testing.
7. Dockerized setup using a `Dockerfile` and `docker-compose.yml` to ease environment setup.

4.3 API Overview

This section outlines the RESTful APIs exposed by the system. The APIs are grouped by domain: Users, Books, and Borrowing. Access to each endpoint is controlled via role-based security using Spring Security.

4.3.1 User Registration

- `POST /register/member`
  Registers a new member. Accessible by all users.
- `POST /register/librarian`
  Registers a new librarian. Accessible by all users.

4.3.2 Book Management

- `GET /api/books`
  Retrieve all books. Accessible by all users.
- `GET /api/books/{id}`
  Retrieve a book by ID. Accessible by all users.
- `GET /api/books/search?query={text}`
  Search books by title or author. Accessible by all users.

- `POST /api/books`
  Add a new book. LIBRARIAN only.
- `PUT /api/books/{id}`
  Update a book. LIBRARIAN only.
- `DELETE /api/books/{id}`
  Delete a book. LIBRARIAN only.

4.3.3 Borrowing APIs

- `POST /api/borrow`
  Borrow a book. Accessible by all users.
- `PUT /api/borrow/return/{recordId}`
  Return a book. Accessible by all users.
- `GET /api/borrow/history/member/{id}`
  View borrowing history for a member. LIBRARIAN only.
- `GET /api/borrow/history/book/{id}`
  View borrowing history for a book. LIBRARIAN only.

A full Postman Collection with all API requests, headers, and bodies is included in the project repository.

5 Future Improvements

- Switch to MySQL/PostgreSQL in production
- Add DTO layer for cleaner API responses
- Develop and implement a front-end component to interact with the application
- Extend the role system to support admin-level access or read-only roles.

6 Conclusion

The Library Management System provides a complete back-end solution with the design, development and deployment of the system.