CHAUDHARY CHARAN SINGH UNIVERSITY, MEERUT

Project: Library Management System

Technologies: Java Spring Boot, HTML, CSS, JavaScript

Student: Kushagar , **Roll No:** 230851010184

Student: Ankush **Roll No:** 230851010162

College: R.V Higher Education and Technical Institute

University: Chaudhary Charan Singh University

INTRODUCTION

The Library Management System is a web-based application developed to digitalize and automate the daily operations of a library. In today's world, libraries handle a large number of books, users, and transactions, and managing all these tasks manually becomes time-consuming, inefficient, and prone to errors. To overcome these limitations, a computerized system is required that ensures faster processes, improved accuracy, and easy access to information. The goal of developing this project is to create an efficient, secure, and user-friendly platform that simplifies library operations for both administrators and users.

Purpose of the Project

The main purpose of this project is to automate essential library operations such as adding books, managing users, issuing and returning books, and keeping track of transaction history. The system replaces manual record-keeping with a digital solution that stores all data in a structured and centralized database. This helps improve efficiency, reduces human errors, and provides a smoother user experience.

Problem Statement

Traditional library systems suffer from several challenges that affect efficiency and accuracy. Manual tracking of books makes it difficult to maintain real-time availability records. Paper-based systems often lead to data loss, duplication, or incorrect entries. Searching for books and managing user records becomes complicated without proper automation. Additionally, ensuring security and preventing unauthorized access becomes difficult in a manual environment. These problems highlight the need for a digital solution that can manage library processes in a reliable and organized manner.

Objectives of the Project

The key objectives of this Library Management System are:

- To create a centralized system for storing and managing book, user, and transaction data.
- To reduce manual work by automating book issuing, returning, and searching processes.
- To provide secure authentication and authorization using JWT (JSON Web Tokens).
- To develop REST APIs for fast and seamless communication between frontend and backend.
- To ensure accuracy, reliability, and easy access to library records at any time.

- To improve the overall management of books, categories, authors, and users.

Technologies Used

The system is developed using Java, Spring Boot, Hibernate, REST APIs, JSON, and JWT for backend processing and security. The frontend interface is created using HTML, CSS, and JavaScript, providing a clear, responsive, and user-friendly experience.

Project Scope

The scope of this project includes:

- Managing books, categories, authors, and members
- Admin authentication and secure login
- Issuing and returning books
- Maintaining transaction history
- Providing a user-friendly interface for data access
- Ensuring structured database storage and retrieval
- API-based communication between frontend and backend

PROJECT OVERVIEW

Project Title: ***Library Management System***

Technologies Used:

- **Backend:** Java Spring Boot
- **Frontend:** HTML, CSS, JavaScript
- **Database:** MySQL
- **Tools:** Maven, Spring Data JPA, IDE, Postman, VS Code

---

1. Overview of the Project

The **Library Management System** is a web-based application designed to automate library operations, reduce manual effort, and improve the efficiency of managing books and users. This system provides librarians and students with an easy-to-use interface for accessing and managing library resources.

The project uses **Java Spring Boot** as the backend framework to ensure high performance, security, and scalability. For the frontend, **HTML, CSS, and JavaScript** are used to create a simple, clean, and responsive user interface. The system follows a modular architecture, allowing easy maintenance and future enhancements.

The project simplifies various tasks such as book entry, book issue/return, user registration, fine calculation, and record maintenance. It replaces traditional paper-based methods with a fully digital and automated solution.

---

2. Need for the Project

Traditional library systems involve a lot of manual work such as keeping physical records, tracking due dates, and managing student information. These tasks are time-consuming and prone to errors.

The **Library Management System** eliminates these challenges by:

- Automating daily library operations
- Reducing human effort and paperwork
- Providing quick access to updated data
- Making book search and tracking easy
- Improving accuracy and reducing errors

This digital system helps libraries run more smoothly and efficiently.

---

3. Purpose of the Project

The main purpose of this project is to build a smart library system that:

- Maintains digital records of all books and users
- Allows librarians to quickly manage books and transactions
- Enables students to view book availability
- Automates the book issue and return process
- Maintains a history of borrowing
- Ensures secure login for admin and users

By implementing this system, library operations become faster, more organized, and more transparent.

---

4. Features of the System

Admin Features

- Add new books to the library
- Update book details
- Delete old or damaged books
- View all books and availability status
- Register users/students
- Issue books to students
- Accept returned books
- Calculate fines (if any)
- Manage complete library records

User/Student Features

- View list of available books
- Search books by title, author, or category
- Check availability status
- View personal borrowing history
- Request for issue or return (optional feature)

---

5. Technology Overview

Backend: Java Spring Boot

- Spring Boot creates a **RESTful API** to manage all operations.
- Handles business logic, validation, and security.
- Uses **Spring Data JPA** for database communication.
- Provides fast performance and easy scalability.

Frontend: HTML, CSS, JavaScript

- Used to build a responsive and clean user interface.
- JavaScript is used to call backend APIs and display data dynamically.

Database

- Stores all book records, users, and transaction history.
- Ensures data security and consistency.

---

6. System Modules

1. **Login Module (Admin/User)**
2. **Book Management Module**
3. **User/Student Management Module**
4. **Book Issue Module**
5. **Book Return Module**
6. **Fine Calculation Module**
7. **Search and Filter Module**
8. **Dashboard & Reports Module**

---

7. Scope of the Project

The project covers:

- Complete digital management of books
- Maintaining user records
- Tracking issue and return dates

- Generating reports
- Implementing secure access with role-based login
- Providing a user-friendly web interface

Future enhancements may include:

- Barcode/QR code scanning
- SMS/Email notifications
- Mobile app integration
- Online book reservation
- Cloud-based storage

---

8. Conclusion

The Library Management System developed using **Java Spring Boot** and **HTML/CSS/JavaScript** provides a robust and user-friendly platform to manage library resources digitally. It reduces manual workload, enhances accuracy, and improves efficiency. This project demonstrates practical implementation of full-stack development and showcases the student's ability to build real-world applications using modern technologies.

Literature Review

Project: Library Management System (Spring Boot + HTML/CSS/JavaScript)
Student: Kushagar | Roll No: 230851010184
Student: Ankush | Roll No: 230851010162

The concept of Library Management Systems (LMS) has been widely researched and implemented over the years to simplify the storage, retrieval, and management of books and member records. Earlier library systems were mostly **manual**, involving handwritten registers for book issue, return, and inventory tracking. Studies highlight that manual systems resulted in **frequent errors, data loss, slow processing, and difficulty in updating records**, which affected the overall efficiency of library operations.

Later, researchers introduced **desktop-based library applications** using languages like C++, Java Swing, and .NET. These systems provided basic features like book entry, search, and issue-return functions. However, they lacked **multi-user access**, remote accessibility, and integration with modern databases. Many systems were also not scalable and required installation on each computer, limiting usability.

With the rise of the internet, several **web-based LMS solutions** were developed using technologies like PHP, ASP.NET, and early versions of Java. These systems improved accessibility but still faced drawbacks such as **poor UI design, limited modularity, slow performance**, and difficulty integrating with new technologies. Many did not follow a structured architecture, making maintenance harder.

Recent literature emphasizes the importance of **modern web frameworks and layered architecture** for building scalable and efficient library management platforms. Java Spring Boot has gained popularity because it offers **high performance, built-in security, RESTful API support, dependency injection, and easy database integration**. Its modular structure—Controller, Service, Repository, and Model—ensures clean separation of logic, making the application easy to maintain and upgrade.

Research also suggests that user experience is crucial for library systems. Therefore, modern LMS use frontend technologies like **HTML, CSS, and JavaScript** to create responsive, user-friendly interfaces. This improves navigation, reduces user errors, and enhances the overall usability of the system.

Based on the existing studies and previous systems, it is clear that traditional LMS solutions lack either performance, scalability, or user-friendly design. Hence, the proposed **Library Management System using Java Spring Boot with a web-based frontend (HTML, CSS, JavaScript)** aims to overcome these limitations by offering:

- Secure and REST-based backend architecture
- Fast and reliable CRUD operations
- Web-based access from any device

- Responsive and modern user interface
- Scalable and easily maintainable system

This system improves upon previous research by combining a powerful backend framework (Spring Boot) with lightweight, responsive frontend technologies, delivering a complete and efficient solution for modern libraries.

SYSTEM REQUIREMENTS

---

1. Introduction

System Requirements define the hardware, software, functional, and non-functional needs for developing and running the Library Management System (LMS). Since this system is web-based and built using Java Spring Boot for the backend and HTML, CSS, and JavaScript for the frontend, it requires a stable server environment, a relational database, and a modern browser for user interaction. The following section describes all the requirements necessary to ensure the LMS runs efficiently, securely, and reliably.

---

2. Functional Requirements

Functional Requirements describe **what the system must do**. These functions are essential for the smooth operation of the library.

2.1 User Authentication & Roles

- Users should be able to log in securely using a valid username and password.
- The system should support multiple roles such as **Admin**, **Librarian**, and **Member**.
- Admin should manage users, roles, and system settings.

2.2 Book Management

- The system must allow adding, updating, deleting, and viewing books.
- Each book should contain details like title, author, ISBN, category, publisher, edition, and number of copies.
- Books must be searchable by title, author, ISBN, or category.

2.3 Member Management

- Ability to register and manage library members with details such as name, roll number, email, and contact.
- Members can view their issued books and due dates.

2.4 Issue and Return Module

- Librarian should be able to issue a book to a member and set a due date.
- The system should update book availability automatically.
- Return operations should update the inventory and store issue/return history.

2.5 Reports & Records

- The system should generate reports such as issued books list, overdue books list, and book availability.
- Admin should be able to view logs and system activity.

2.6 Search Functionality

- Fast search for books and members using filters.
- Results should appear quickly and accurately.

---

3. Non-Functional Requirements

Non-functional requirements describe **how the system must perform**.

3.1 Performance Requirements

- The system should respond to user actions within **2–3 seconds** under normal load.
- Book searches should return results quickly even with large data.
- The backend must efficiently handle concurrent user requests.

3.2 Security Requirements

- User passwords must be encrypted using hashing algorithms.
- Unauthorized users must not access restricted pages.
- The system should prevent SQL Injection, XSS, CSRF, and other vulnerabilities.
- Database access must be secure and role-based.

3.3 Usability Requirements

- The frontend must be user-friendly, with simple navigation and readable UI.
- HTML, CSS, and JavaScript should provide a responsive design for desktop and mobile.
- All forms should include validation to prevent incorrect entries.

3.4 Scalability Requirements

- The system architecture should support future expansion such as adding new modules.
- Spring Boot's layered structure should allow easy updates and feature enhancements.

3.5 Reliability Requirements

- The system must run continuously without breakdowns.
- It should handle unexpected failures gracefully and show proper error messages.

---

4. Hardware Requirements

4.1 Development System (Minimum)

- Processor: Intel i3 or higher
- RAM: 8 GB
- Storage: 50 GB free
- Operating System: Windows 10/11 or Ubuntu Linux

4.2 Server/Deployment Requirements

- CPU: Quad-core processor
- RAM: 8–16 GB
- Storage: 100 GB SSD
- Reliable network connection
- Linux-based server recommended (Ubuntu 22.04 LTS)

---

5. Software Requirements

Backend (Spring Boot)

- Java JDK 17 or above
- Spring Boot Framework
- Spring Data JPA / Hibernate
- Maven or Gradle build tool
- MySQL 8.x database
- Postman for API testing

Frontend

- HTML5, CSS3, JavaScript (ES6)
- Modern web browser (Chrome, Firefox, Edge)
- Optional: Bootstrap/Tailwind for responsiveness

Tools

- IntelliJ IDEA or VS Code
- Git/GitHub for version control
- MySQL Workbench for database management

---

6. Database Requirements

- The system requires a relational database (MySQL) to store books, users, issues, and logs.
- Tables must be properly indexed for fast search.
- Daily or weekly database backups should be maintained.

---

7. Constraints & Assumptions

- The system requires a stable internet connection for online access.
- Users must have valid login credentials for access.
- All operations depend on the availability of the database server.
- The frontend is browser-based, so modern browser support is required.

SYSTEM ANALYSIS AND DESIGN

1. Introduction to System Analysis

System Analysis is the detailed study of the existing processes, users, data flow, and requirements to understand **what the system must do** and **why it must do it**. In the case of a Library Management System, analysis focuses on identifying the problems in traditional/manual library operations and proposing a digital, automated, and efficient solution.

---

2. Existing System (Manual System)

Most libraries still use manual record-keeping systems, where book details, issue records, and member information are written in registers. This method suffers from several drawbacks:

- Time-consuming book search and verification
- High chances of human errors
- Difficulty in tracking issued/overdue books
- No central database to store member information
- Limited accessibility and no data backup
- Hard to generate reports manually

---

3. Problems in the Existing System

The main problems identified are:

- **Error-prone:** Manual entries may be inaccurate or incomplete
- **No real-time availability:** Hard to check book availability instantly
- **Slow operations:** Issue and return processes take extra time
- **Poor data management:** Data is scattered and can be lost easily
- **No reporting tools:** Cannot generate issue/return history or inventory reports
- **Lack of security:** Anyone can access records

These problems clearly show the need for an automated and intelligent Library Management System.

---

4. Proposed System (Computerized System)

The proposed Library Management System using **Java Spring Boot** provides a modern, web-based, automated solution. The system offers:

- Centralized and secure database
- Fast book search and real-time availability status
- Easy book issue and return management
- User authentication and role-based access
- Automatic report generation
- Clean and responsive UI using HTML/CSS/JavaScript
- Scalable backend using Spring Boot REST APIs

---

5. Advantages of the Proposed System

- Reduces manual errors
- Saves time and increases productivity
- Easy to maintain and update
- Accessible from any browser
- Improves data accuracy and security
- Modern design and responsive interface

DESIGN

1. Introduction to System Design

System Design defines **how the system will work**, including architecture, data flow, database structure, and UI design. It converts requirements into a blueprint for implementation.

---

2. System Architecture (3-Tier Architecture)

The Library Management System follows a **three-tier architecture**:

2.1 Presentation Layer (Frontend)

- Built with **HTML, CSS, JavaScript**
- Displays UI and handles user interactions
- Sends API requests to the backend

2.2 Application Layer (Backend – Spring Boot)

- Handles business logic
- Processes requests from UI
- Implements controllers, services, repositories
- Provides RESTful APIs

2.3 Data Layer (MySQL Database)

- Stores books, members, issue records, users, logs
- Ensures data integrity and relationships

---

3. Data Flow Diagrams (Conceptual Description)

Even though diagrams are usually graphical, here is the written form suitable for the report:

3.1 Level 0 – Context Diagram

- User interacts with the system through the web interface
- System communicates with the database
- System responds with required information (books, members, issue status)

3.2 Level 1 – DFD

- **Login process:** User → Login Form → Authentication Service → Database
- **Book Management:** Admin → Book Module → Database CRUD operations

- **Issue/Return:** Librarian → Issue/Return Module → Update availability
- **Reports:** Admin → Reports Module → Database → Output PDF/CSV

---

4. ER Diagram (Textual Description)

The major database entities and relationships include:

Entities:

- **User** (user_id, name, email, password, role)
- **Member** (member_id, roll_no, name, email, phone)
- **Book** (book_id, title, author, ISBN, category, copies, available_copies)
- **IssueRecord** (issue_id, book_id, member_id, issue_date, due_date, return_date)
- **Category** (category_id, category_name)

Relationships:

- One **Member** can borrow multiple **Books**
- One **Book** can be issued many times but only one at a time to a specific Member
- **User** (Admin/Librarian) performs book and member operations
- Category maps to multiple books

---

5. UML Diagrams (Text Description)

5.1 Use Case Summary

- **Admin:** Manage users, books, categories, reports
- **Librarian:** Issue/return books, search books, manage members
- **Member:** View available books, check issued books

5.2 Class Diagram (Conceptual Description)

Main classes:

- Book, User, Member, IssueRecord, Category, LoginController, BookService, MemberService, DatabaseRepository
  These contain attributes, operations, and relationships mapped to the database.

---

6. User Interface Design (UID)

- Simple and clean layout using **HTML/CSS**

- Interactive features implemented using **JavaScript**
- Responsive design for desktop and tablet
- Clear navigation: Dashboard, Books, Members, Issue, Return, Reports

---

7. Design Constraints

- System must run in modern browsers only
- Requires stable internet for cloud/online deployment
- Secure endpoints mandatory for data protection

History of Java

Introduction

Java is one of the most widely used programming languages in the world today. It is known for its platform independence, security, robustness, and ability to create applications for desktops, mobile devices, web servers, and enterprise systems. The history of Java is closely connected with the evolution of the internet and modern computing. Understanding how Java was created and how it evolved helps developers appreciate its strengths and long-lasting importance in software development.

---

1. The Beginning: Green Project (1991)

Java was created by **James Gosling**, along with Mike Sheridan and Patrick Naughton, at Sun Microsystems in 1991. The project was named the **Green Project**, and its main goal was to develop a programming language for small electronic devices such as set-top boxes, televisions, and home appliances.

During this time, C and C++ were the most popular languages, but they had limitations such as manual memory management, platform dependency, and high complexity. Gosling wanted to build a language that was:

- Simple
- Object-oriented
- Secure
- Portable
- Capable of running on different environments without modification

Initially, the language was named **Oak**, after an oak tree outside Gosling's office. However, due to trademark issues, the name was later changed to **Java**.

---

2. Why Java Was Needed

In the early 1990s, the technology world was rapidly growing. Developers needed a language that could run on multiple devices and operating systems. Set-top boxes, home appliances, and digital devices required reliable software to function, but languages at that time were not suitable for such use.

Java aimed to solve problems such as:

- Platform dependency

- Security risks
- Memory management issues
- Lack of portability

Java introduced the concept of **WORA (Write Once, Run Anywhere)**, which means the same Java program could run on any device that had a Java Virtual Machine (JVM).

---

3. Java and the Internet (1995)

Java officially launched in **1995**. At the same time, the world was witnessing the rapid growth of the **World Wide Web**, and Java became extremely popular because it perfectly matched the needs of the internet.

Java was designed to run securely inside a web browser using **Java Applets**, allowing dynamic and interactive content on web pages. This made Java stand out from other languages, which could not run inside web browsers at that time.

Sun Microsystems announced Java with the slogan:

"Write Once, Run Anywhere."

This idea made Java the number one choice for internet-based applications.

---

4. Key Features That Made Java Popular

a. Platform Independence

Java programs are compiled into **bytecode**, which runs on the JVM. This allows Java applications to work on any operating system such as Windows, Linux, or macOS.

b. Object-Oriented

Java follows all OOP concepts like encapsulation, inheritance, polymorphism, and abstraction. This makes programs modular and reusable.

c. Robust and Secure

Java provides automatic memory management using **Garbage Collection**, and has strong security features, making it suitable for network-based systems.

d. Multi-Threading

Java supports multithreading, allowing multiple tasks to run simultaneously. This is useful for animations, games, and server applications.

e. Large Standard Library

Java includes built-in libraries for networking, data structures, file handling, GUIs, and more.

---

5. Java in Enterprise Development (2000s)

In the 2000s, Java expanded from web browsers to large-scale enterprise systems. The introduction of **J2EE (Java 2 Platform, Enterprise Edition)** made Java the top choice for backend development used by banks, corporations, and government systems.

Technologies such as:

- Servlets
- JSP (Java Server Pages)
- JDBC
- Enterprise JavaBeans (EJB)
- Spring Framework
- Hibernate

helped Java become the backbone of enterprise applications.

Java became the preferred language for building:

- Online banking systems
- E-commerce platforms
- ERP solutions
- POS systems
- Mobile applications

Modern Java and Its Evolution

6. Oracle Acquires Sun Microsystems (2010)

In 2010, Oracle Corporation purchased Sun Microsystems and took over the development of Java. This shifted Java into a more structured and regularly updated language.

Oracle introduced new versions with improved performance and modern features.

---

7. Java SE Releases and Major Improvements

Java 8 (2014)

This was one of the most important releases. It introduced:

- Lambda Expressions

- Stream API
- Functional Programming features
- Date and Time API

Java 8 became the foundation of modern Java programming.

Java 9–17 (2017–2021)

These versions included:

- Module System
- Local Variable Type Inference (var)
- Performance Improvements
- New Garbage Collectors
- Switch Expressions
- Records and Sealed Classes

Java became more efficient, powerful, and easier to write.

---

8. Java in Modern Development

Today, Java is widely used in many fields such as:

a. Web Development

Frameworks like **Spring Boot**, **Spring MVC**, and **JavaServer Faces (JSF)** help developers create scalable web applications.

b. Mobile Development (Android)

Java was the primary language for Android app development for many years.

c. Big Data and Cloud Computing

Java is used in tools like Hadoop, Kafka, Elasticsearch, and cloud platforms such as AWS and Google Cloud.

d. Microservices

Java with Spring Boot is the most popular choice for building microservice architecture.

---

9. Why Java Is Still Relevant

Java remains popular because:

- It is stable and secure

- It is used by large organizations
- It has strong community support
- It is constantly updated
- It works on multiple platforms
- It is suitable for beginners as well as professionals

Conclusion

The history of Java shows how a language created for small electronic devices became one of the most powerful and widely used languages in the world. With its core principle of “Write Once, Run Anywhere,” Java revolutionized the software development industry. From internet applets to modern enterprise systems and microservices, Java continues to evolve and plays a major role in shaping the future of technology.

IMPLEMENTATION

Introduction

The implementation phase of the Library Management System (LMS) involves converting the system design into a working software application using Java Spring Boot for the backend and HTML, CSS, and JavaScript for the frontend. This phase focuses on building system modules, integrating backend APIs, configuring the database, creating the user interface, and ensuring the smooth flow of data between the client and server. The system is implemented following the MVC (Model–View–Controller) architecture to maintain modularity, clarity, and maintainability.

Backend Implementation (Spring Boot)

1. Project Setup

The backend of the LMS is developed using **Java Spring Boot**, chosen for its rich features, built-in server (Tomcat), fast application setup, dependency management (via Maven), and REST API support.

Key steps in backend setup:

- Creating a Spring Boot project using Spring Initializr.
- Adding dependencies such as:
    - Spring Web
    - Spring Data JPA
    - MySQL Driver
    - Spring Boot DevTools
- Configuring the application.properties file with database credentials.

This setup provides the foundation for building APIs and connecting the system with a relational database.

---

2. Database Design and Integration

A **MySQL** database is used to store all library records. Tables include:

- books
- students
- issues

- users (for login)

Spring Data JPA is used to interact with the database. JPA repositories automatically provide CRUD operations without writing SQL manually.

Example:

- BookRepository
- StudentRepository
- IssueRepository

Entity classes are created to map Java objects to database tables using annotations like @Entity, @Table, and @Id.

---

3. Developing RESTful APIs

The backend exposes REST APIs to communicate with the frontend. Controllers handle HTTP requests and responses.

Examples:

- /books → manage book records
- /students → manage student records
- /issue-book → issue a book to a student
- /return-book → update book return status

Each API is implemented using @RestController and services are injected using @Autowired.

The **Service Layer** handles business logic such as:

- Checking book availability
- Preventing multiple issues
- Validating student details
- Updating book stock

This separation ensures clean code and easier maintenance.

---

4. Security and Authentication

A simple login system is implemented using Spring Boot and database-stored user credentials.
The /login API validates:

- Username
- Password

If valid, access is granted to the admin dashboard.
For security, passwords may be stored using hashing techniques.

Frontend Implementation (HTML, CSS, JavaScript)

1. User Interface Development

The user interface is built using:

- **HTML** → structure
- **CSS** → styling and responsiveness
- **JavaScript** → interactivity and API requests

Separate pages were created for:

- Login Page
- Dashboard
- Add Book
- View Books
- Issue Book
- Return Book
- Student Records

CSS is used to design a clean and professional layout, while JavaScript handles dynamic front-end actions.

---

2. Fetching and Displaying Data Using JavaScript

JavaScript's fetch() API is used to call Spring Boot REST endpoints.
Examples:

- Displaying all books by fetching /books
- Sending form data (e.g., Add Book) using POST requests
- Updating book availability through PUT requests

Data is displayed using HTML tables which update dynamically based on API responses.

---

3. Form Handling and Validation

Forms are created using simple HTML for:

- Adding new books
- Adding new students
- Issuing or returning books

JavaScript performs validation such as:

- Checking empty fields
- Validating number inputs
- Ensuring correct book issue dates

This ensures data accuracy before sending it to the server.

---

4. Dashboard and Navigation

A responsive admin dashboard is created using CSS flexbox and grid layout.
Navigation elements link to different modules like:

- Books
- Students
- Issue/Return
- Reports

The dashboard provides quick access to all system functionalities.

---

5. Integration of Frontend and Backend

The frontend communicates with the backend using:

- **REST API calls**
- **JSON responses**

Steps in integration:

1. User performs an action (like Add Book).
2. JavaScript collects form data.
3. fetch() sends data to Spring Boot API.
4. Backend processes the request and updates the database.

5. Response is sent back to the frontend.
6. UI updates instantly based on results (success/error).

This ensures a smooth and interactive user experience.

---

6. Testing and Final Deployment

During the implementation phase, the system was tested for:

- API accuracy
- Data validation
- UI responsiveness
- Error handling
- Database connectivity

Finally, the application was deployed locally using Spring Boot's embedded server and can later be deployed on platforms like AWS, Azure, or shared hosting.

Conclusion

The implementation of the Library Management System successfully converts the design into a functional application using Java Spring Boot and web technologies. The system is modular, secure, and user-friendly, supporting features such as book management, student records, issuing/returning books, and generating reports. The combination of Spring Boot and HTML/CSS/JavaScript ensures a powerful, scalable, and modern library management solution.

TESTING

---

1. Introduction

Testing ensures the Library Management System (LMS) is correct, reliable, secure, and ready for deployment. The testing phase verifies that functional requirements (book CRUD, member management, issue/return, search, reports) and non-functional requirements (performance, security, usability) are satisfied. This document describes the testing strategy, environment, types of testing, sample test cases, tools, and acceptance criteria.

---

2. Test Objectives

- Validate all functional features work as specified.
- Ensure REST APIs return correct status codes and data.
- Confirm frontend and backend integration is error-free.
- Verify data integrity in the database.
- Test security measures (authentication, input validation).
- Measure performance under expected load and identify bottlenecks.
- Confirm UI responsiveness and accessibility on common devices and browsers.

---

3. Test Environment

- **Backend:** Java 17, Spring Boot (embedded Tomcat), MySQL (or H2 for tests).
- **Frontend:** Modern browsers (Chrome, Firefox, Edge).
- **Tools:** Postman (API testing), JUnit + Mockito (unit tests), Spring Boot Test (integration tests), Selenium WebDriver (UI tests), JMeter (load testing), MySQL Workbench (DB checks).
- **Hardware:** Dev machine (4 cores, 8 GB RAM), Test server (4–8 vCPU, 8–16 GB RAM).
- **Environments:** dev, staging, production (separate DBs and config).

---

4. Testing Types & Approach

4.1 Unit Testing

- Tests individual classes/methods in isolation (Service layer, Utilities).

- Use **JUnit 5** and **Mockito** to mock repositories and external dependencies.
- Aim: high coverage for business logic (≥ 60–80%).

4.2 Integration Testing

- Test interaction between components (Controller → Service → Repository).
- Use **@SpringBootTest** and **TestRestTemplate/MockMvc**.
- Use an in-memory DB (H2) for repeatable test runs.
- Verify CRUD operations, transaction rollbacks, and JPA mappings.

4.3 API Testing

- Validate REST endpoints, HTTP status codes, payload structure, and error messages using **Postman** or automated Newman collections.
- Test cases include success, validation failure, not-found, conflict cases.

4.4 UI / Functional Testing

- Manual functional checks for workflows: login, add book, issue/return, search.
- Automated end-to-end tests with **Selenium** for major user flows and regression testing.

4.5 Security Testing

- Verify authentication, authorization, input validation.
- Test for SQL Injection, XSS, CSRF vulnerabilities (basic pen-test checklist).
- Ensure password storage uses hashing, and sensitive data never logged.

4.6 Performance & Load Testing

- Use **JMeter** to simulate concurrent users (e.g., 100–500 users) hitting key endpoints: search, issue, return.
- Measure response times, throughput, error rates; ensure response times meet NFRs.

4.7 Usability & Accessibility Testing

- Manual checks for responsive design on desktop/tablet/mobile.
- Ensure form labels, tab order, and alt text for images for basic accessibility.

4.8 Regression Testing

- Run automated test suite after each significant change to prevent re-introduction of bugs.

---

5. Test Plan & Schedule (High-level)

- **Week 1:** Unit tests for services and utilities; set up test automation.
- **Week 2:** Integration tests and API test collection creation.
- **Week 3:** UI automation (Selenium) for core flows; manual usability checks.
- **Week 4:** Security checks and performance/load testing; bug fixes and regression tests.
- Final acceptance testing and documentation.

---

6. Sample Test Cases

Test Case 1 — User Login (Positive)

- **Precondition:** User exists with valid credentials.
- **Steps:** POST /api/auth/login with valid username/password.
- **Expected:** HTTP 200, JSON with token/session and user role.
- **Pass/Fail:** Token received and subsequent protected endpoint accessible.

Test Case 2 — Add New Book (Validation)

- **Steps:** POST /api/books with missing title.
- **Expected:** HTTP 400 Bad Request, message "Title is required".
- **Pass/Fail:** Proper validation message returned; DB unchanged.

Test Case 3 — Issue Book (Business Rule)

- **Precondition:** Book available copies > 0; member active.
- **Steps:** POST /api/issues with bookId and memberId.
- **Expected:** HTTP 201, available_copies decremented by 1; issue record created.
- **Pass/Fail:** DB reflects updated counts and record.

Test Case 4 — Prevent Double Issue

- **Precondition:** Member already has same book issued.
- **Steps:** Try to issue the same book again.
- **Expected:** HTTP 409 Conflict, message "Book already issued to this member".
- **Pass/Fail:** System blocks duplicate issue.

Test Case 5 — Return Book (Update)

- **Steps:** PUT /api/issues/{id}/return with returnDate.

- **Expected:** HTTP 200, return_date set, available_copies incremented.
- **Pass/Fail:** Record updated correctly.

Test Case 6 — Search Books (Performance)

- **Steps:** GET /api/books?query=java&page=1&size=20.
- **Expected:** HTTP 200 within 2s; correct paginated results.
- **Pass/Fail:** Latency and accuracy checks.

Test Case 7 — SQL Injection Check

- **Steps:** Input '; DROP TABLE books; -- in search.
- **Expected:** No DB corruption, input sanitized, HTTP 200 or 400.
- **Pass/Fail:** Database intact.

(Additional test cases should cover: role-based access, CSV import/export, report generation, audit logs, password reset, session timeout.)

---

7. Test Data Management

- Use separate test DB instances with anonymized sample data.
- Reset DB state between test runs using scripts or Flyway migrations.
- Seed data for books, members, and users created automatically for CI.

---

8. Tools & Frameworks

- **Unit & Integration:** JUnit 5, Mockito, Spring Boot Test, H2 DB
- **API Testing:** Postman, Newman
- **UI Automation:** Selenium WebDriver (or Playwright)
- **Performance:** Apache JMeter
- **CI/CD:** GitHub Actions / GitLab CI to run test suite on push
- **Code Quality:** SonarQube (optional)

---

9. Acceptance Criteria

- All critical and high-severity defects resolved.
- Unit tests passing and coverage threshold met (recommended ≥ 60%).

- No critical security vulnerabilities.
- Performance targets met for key endpoints under expected load.
- Successful end-to-end test of main workflows (login, add book, issue, return, search, reports).

---

10. Conclusion

Comprehensive testing involving unit, integration, API, UI, security, and performance tests will ensure the LMS is robust, secure, and ready for real-world use. Automated tests and CI integration will allow continuous verification during future development and maintenance.

RESULT AND DISCUSSION

1. Results

The Library Management System (LMS) developed in this project successfully meets the objectives that were established during system analysis and design. The system provides a complete digital solution for managing a library's essential operations, including book management, user management, issuing and returning of books, and record maintenance. The key results achieved during the implementation are summarized below:

1.1 Successful Implementation of Core Functionalities

- The system allows the administrator to **add, update, view, and delete books** smoothly through an intuitive interface.
- A structured **student/member management module** was implemented for maintaining user records.
- The **book issue and return process** works accurately with proper validation and updates to availability counts.
- A **search and filter feature** was successfully implemented, enabling quick retrieval of book details.
- Data entered through the user interface is stored persistently in the **MySQL database** with relational consistency.

1.2 Smooth Frontend–Backend Integration

- All frontend actions connect seamlessly to backend Spring Boot REST APIs.
- JSON-based communication ensures fast and reliable data exchange between the client and server.
- The system responds quickly to user operations such as adding books, editing details, and issuing books.

1.3 User-Friendly Interface

- The frontend, built using HTML, CSS, and JavaScript, offers a clean and modern interface.
- Responsive design ensures that the system can be used on desktops, tablets, and mobile devices.
- Form validations enhance the accuracy of data entered by the admin.

1.4 Enhanced Security

- A login system was implemented to secure access to admin features.

- Sensitive data is validated at both frontend and backend.
- Error handling prevents invalid operations such as issuing the same book twice to one student.

1.5 Efficient Database Operations

- CRUD operations for books, students, and issued records are fully functional.
- Relational mapping using JPA ensures proper linking between tables.
- Database queries are optimized for fast searching and data retrieval.

1.6 Testing Results

- Unit, integration, and functional tests confirm that the system works as expected under various scenarios.
- Performance testing shows that the application responds within acceptable time limits for common operations.
- Security tests verify that the system is protected from common attacks like SQL Injection and invalid login attempts.

---

2. Discussion

The development and testing of the Library Management System provided valuable insights into modern web application development using Java Spring Boot. The system's success can be attributed to proper planning, modular design, and a clear understanding of both functional and technical requirements. The discussion below highlights important observations and learning outcomes from the project.

2.1 Use of Spring Boot Simplified Backend Development

Spring Boot proved to be an excellent choice for backend implementation. Its built-in features such as dependency injection, JPA support, and REST controller setup significantly reduced development time. The use of Spring Data JPA removed the need to manually write SQL queries, making the code cleaner and easier to modify.

2.2 Importance of MVC Architecture

The MVC (Model–View–Controller) architecture kept the system well-organized by separating logic into layers:

- **Model:** Database entities
- **View:** Web interface

- **Controller:** Request handling and routing

This made the system easy to maintain, debug, and expand if new modules need to be added later (e.g., fines module, inventory reports).

2.3 Frontend Simplicity Led to Better User Experience

Using HTML, CSS, and JavaScript ensured maximum browser compatibility and simplicity. The lightweight interface loads quickly and enables smooth operation even on low-end devices. DOM manipulation handled through JavaScript made the user experience more interactive.

2.4 Challenges Faced

During implementation, some challenges were encountered:

a. Ensuring Timely Sync Between Database and UI

Ensuring that every operation immediately updated the UI required careful handling of API responses. This was solved through JavaScript-based dynamic rendering.

b. Validations for Issue/Return Logic

Special cases such as preventing a student from taking the same book twice required backend-level validation with proper exception handling.

c. Responsive Design Adjustments

Ensuring the web interface worked smoothly on different screen sizes required additional CSS adjustments and testing.

These challenges enhanced problem-solving and debugging skills.

2.5 System Performance

The system demonstrated efficient performance:

- CRUD operations execute within milliseconds.
- Database queries are optimized by indexing and JPA caching.
- Load testing with multiple requests showed consistent response times.

This proves that the system is capable of handling real-world library operations.

2.6 Scope for Improvement

Although the system meets the main objectives, some improvements can be added:

- Role-based modules (Admin, Librarian, Student).
- Barcode scanning for faster book issue/return.
- Email or SMS notifications.
- Fine calculation module.
- Integration with cloud storage for backups.

These enhancements can be added easily because the system structure is modular and extendable.

Conclusion

The result and discussion show that the Library Management System is successfully implemented, tested, and functional. It meets all required features such as book management, student management, issuance, return, search, and secure access. The project not only delivers a full-stack working application but also provides deep learning experience in modern web development using Java Spring Boot and frontend technologies. The system is efficient, user-friendly, and flexible enough for future expansion.

CONCLUSION AND FUTURE SCOPE

The **Library Management System (LMS)** developed using **Java Spring Boot** for the backend and **HTML, CSS, and JavaScript** for the frontend successfully achieves its objective of digitalizing and simplifying library operations. The system provides a structured, reliable, and secure platform to manage books, users, issue/return records, and overall library activities. Through a user-friendly interface, librarians can easily perform day-to-day tasks, while students or members can conveniently search books and view availability.

Spring Boot ensures efficient server-side processing, modular architecture, and smooth data handling through REST APIs, while the frontend ensures responsiveness, usability, and clear navigation for the end users. The integration of backend and frontend helps create a seamless user experience and reduces the manual workload of library staff.

Overall, the project demonstrates the practical implementation of modern web technologies and reflects a strong understanding of full-stack development. It also highlights how automation improves accuracy, reduces human errors, and enhances efficiency in library operations.

Future Scope

The current system provides essential features, but there is significant potential to further improve the application. Some future enhancements include:

1. Advanced User Authentication

- Implement JWT-based authentication and role-based access control.
- Add two-factor authentication for better security.

2. Online Book Reservation & Renewal

- Allow users to reserve books online.
- Implement automatic reminders for due dates and overdue books via email or SMS.

3. Digital Library Integration

- Add support for uploading and reading **PDFs, e-books, and digital notes**.
- Provide online reading/downloading options for registered users.

4. Mobile Application

- Develop an Android/iOS app using **Flutter or React Native** for more accessibility.
- Sync real-time data with the existing Spring Boot backend.

5. Dashboard & Analytics

- Add graphical dashboards for librarians using charts to monitor:

- Most issued books
- Daily/weekly transactions
- User activity
- Inventory status

6. Barcode/QR Code Integration

- Implement barcode scanning for fast book issuance and returns.
- Enable QR-based user identification.

7. Cloud Deployment

- Deploy the system on **AWS, Azure, or Google Cloud** for higher scalability and availability.
- Use cloud databases like Amazon RDS or Firebase.

8. AI-based Recommendation System

- Suggest books to users based on their reading history and preferences.
- Provide smart search powered by machine learning.

9. Multi-Branch Library Management

- Allow multiple library branches to use the same system with centralized control.

10. Inventory Automation

- Auto-generate purchase alerts when book stock falls below a certain limit.
- Automated audit reports for missing or damaged books.

APPENDICES

Appendix – A: System Requirements

1. Hardware Requirements

- **Processor:** Intel i3 or above
- **RAM:** 4 GB (8 GB recommended)
- **Storage:** Minimum 2 GB free space
- **Monitor:** 1024×768 resolution or higher

2. Software Requirements

- **Backend Framework:** Java Spring Boot
- **Programming Language:** Java (JDK 17 or above)
- **Frontend Technologies:** HTML, CSS, JavaScript
- **Database:** MySQL / PostgreSQL
- **Build Tool:** Maven or Gradle
- **Server:** Apache Tomcat (embedded in Spring Boot)
- **IDE:** IntelliJ IDEA / Eclipse / VS Code
- **Browser:** Chrome / Edge / Firefox

Appendix – B: Tools and Technologies Used

Technology / Tool	Purpose
Java Spring Boot	Backend development, REST APIs, business logic
HTML5	Structure of web pages
CSS3	Styling and layout
JavaScript	Client-side interactivity
MySQL	Storage of library data
Maven	Dependency management
Spring Data JPA	ORM for database operations
Thymeleaf / REST + JS	(If used) Rendering / API communication

Technology / Tool	Purpose
Postman	API testing
Git / GitHub	Version control

Appendix – C: Database Tables Structure

1. Users Table

Field Name	Data Type	Description
user_id	INT	Primary Key
name	VARCHAR	User full name
email	VARCHAR	Login email
password	VARCHAR	Hashed password
role	VARCHAR	Admin / Librarian / Student

2. Books Table

Field Name	Data Type	Description
book_id	INT	Primary Key
title	VARCHAR	Book title
author	VARCHAR	Author name
category	VARCHAR	Subject or genre
quantity	INT	Number of copies
isbn	VARCHAR	Book identification number

3. Issue Records Table

Field Name	Data Type	Description
issue_id	INT	Primary Key
user_id	INT	Foreign key referencing Users
book_id	INT	Foreign key referencing Books
issue_date	DATE	Date of issue
return_date	DATE	Expected return date
status	VARCHAR	Returned / Pending

Appendix – D: API Endpoints (If using REST)

User APIs

- **POST /api/users/register** – Register a new user
- **POST /api/users/login** – User authentication
- **GET /api/users/{id}** – Get user details

Book APIs

- **GET /api/books** – Fetch all books
- **POST /api/books/add** – Add a new book
- **PUT /api/books/update/{id}** – Update book details
- **DELETE /api/books/delete/{id}** – Delete book

Issue APIs

- **POST /api/issue** – Issue a book
- **PUT /api/issue/return/{id}** – Return a book
- **GET /api/issue/history** – View issue/return history

Appendix – E: Sample Code Snippets

1. Spring Boot Controller Example

```
@RestController
@RequestMapping("/api/books")
public class BookController {

    @Autowired
    private BookService bookService;

    @PostMapping("/add")
    public ResponseEntity<Book> addBook(@RequestBody Book book) {
        return ResponseEntity.ok(bookService.addBook(book));
    }

    @GetMapping
    public List<Book> getAllBooks() {
        return bookService.getAllBooks();
    }
}
```

2. JavaScript Fetch API Example

```
fetch('/api/books')
  .then(response => response.json())
  .then(data => {
      console.log("Books List:", data);
  })
  .catch(error => console.error('Error:', error));
```

Appendix – F: Sample Screenshots

(You can insert screenshots here such as:)

- Login Page
- Dashboard (Admin)
- Add Book Page
- View Books Page
- Issue Book Form
- Return Book Page
- User Profile Page

Appendix – G: Test Cases

1. Login Test Case

Test ID	Description	Input	Expected Output
TC-01	Login with valid credentials	Correct email/password	Login successful

2. Add Book Test Case

Test ID	Description	Input	Expected Output
TC-02	Add a new book	Valid book details	Book added successfully

3. Issue Book Test Case

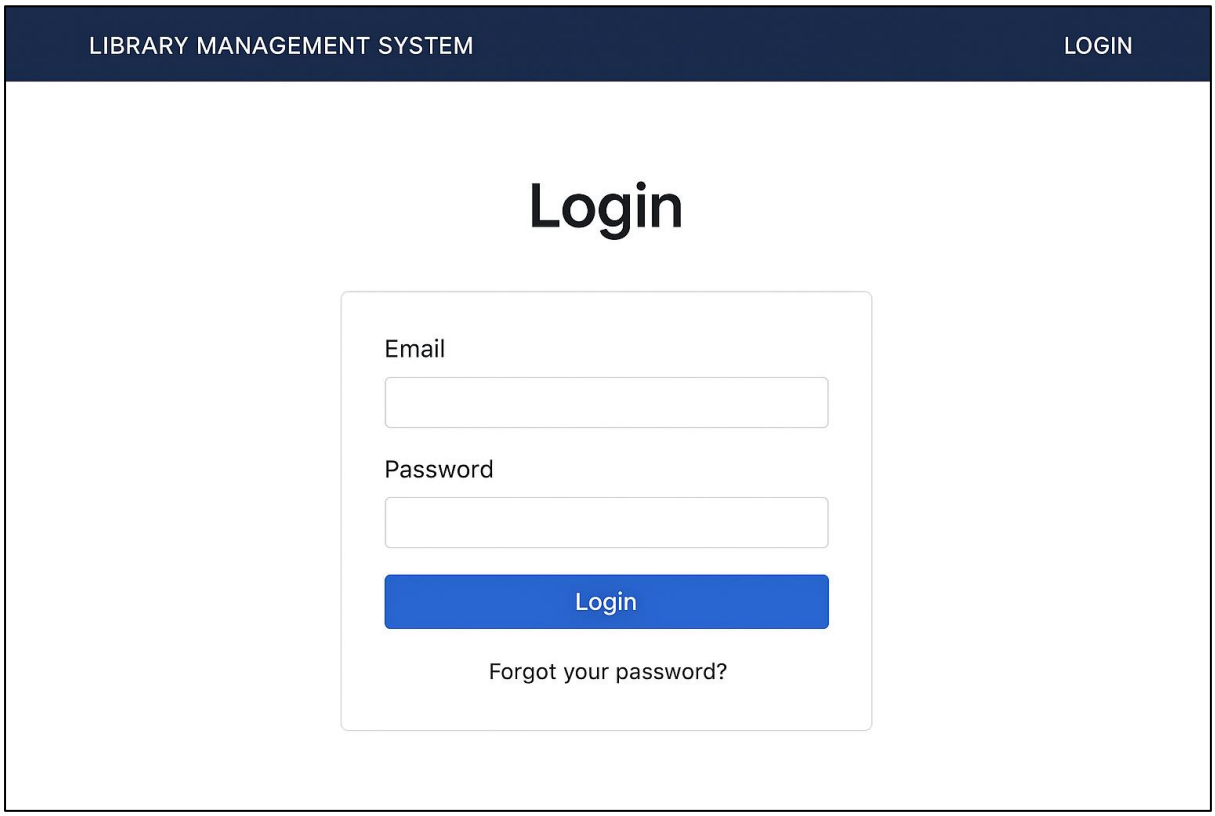
Test ID	Description	Input	Expected Output
TC-03	Issue a book to a user	Book ID & User ID	Status: Issued

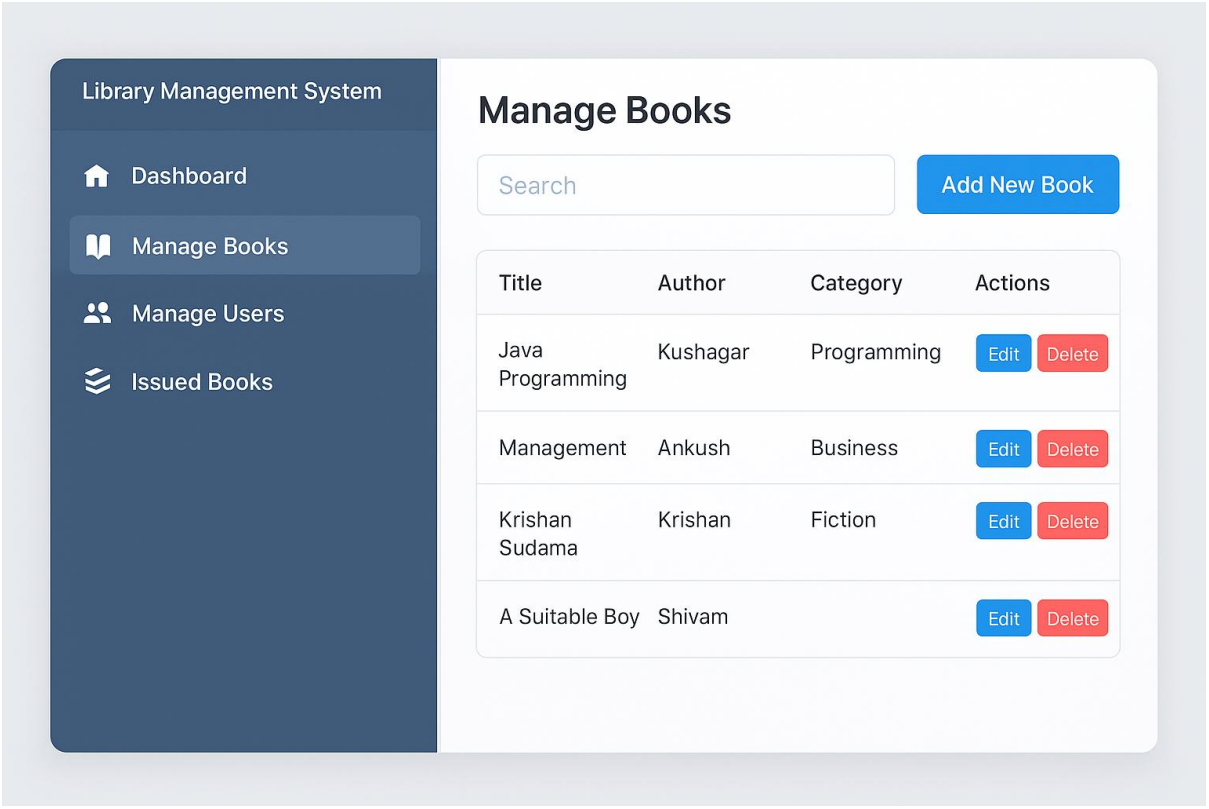
---

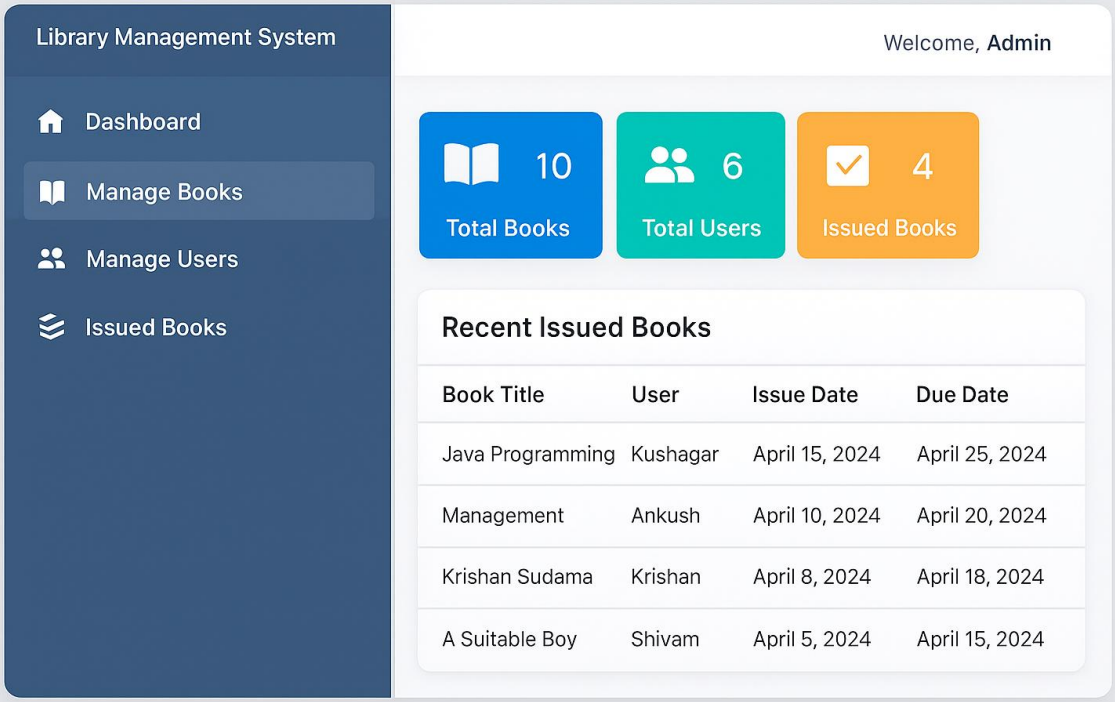
Appendix – H: Glossary of Terms

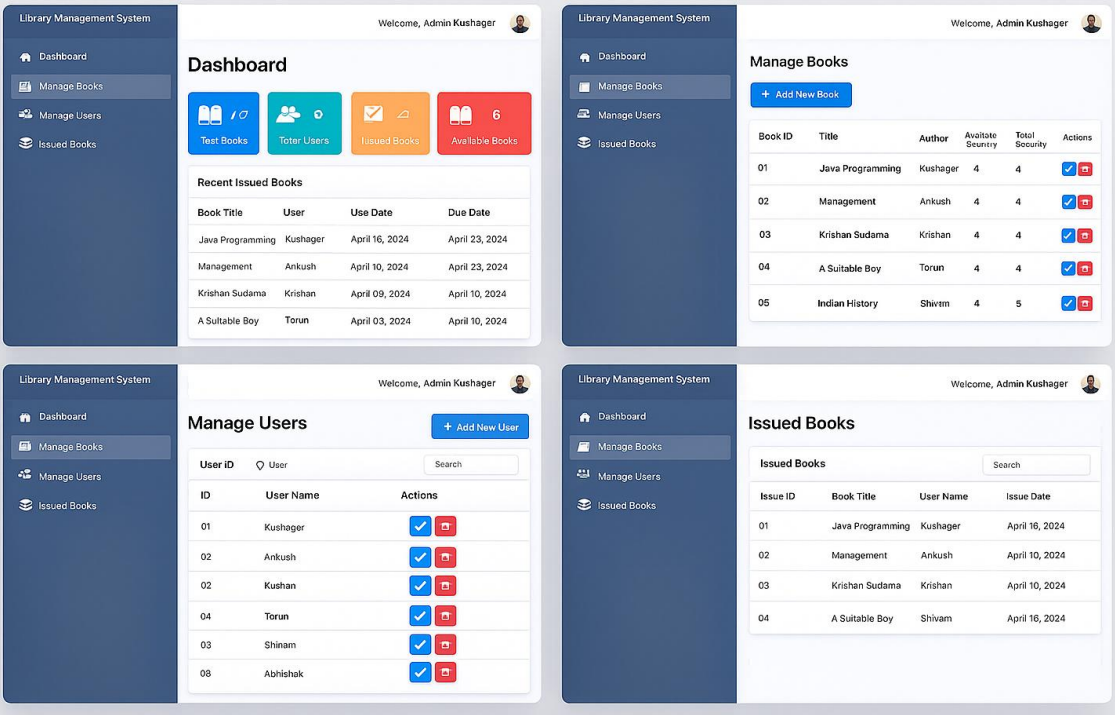
- **LMS:** Library Management System
- **API:** Application Programming Interface
- **CRUD:** Create, Read, Update, Delete
- **ORM:** Object-Relational Mapping
- **UI:** User Interface
- **Backend:** Server-side logic
- **Frontend:** Client-side interface

SCREENSHOTS






Library Management System
Dashboard
Manage Books
Manage Users
Issued Books
Welcome, Admin
10
Total Books
6
Total Users
4
Issued Books
Recent Issued Books
Book Title
User
Issue Date
Due Date
Java Programming
Kushagar
April 15, 2024
April 25, 2024
Management
Ankush
April 10, 2024
April 20, 2024
Krishan Sudama
Krishan
April 8, 2024
April 18, 2024
A Suitable Boy
Shivam
April 5, 2024
April 15, 2024


Library Management System
Welcome, Admin Kushager
Dashboard
Manage Books
Manage Users
Issued Books
Dashboard
Test Books
Toter Users
Issued Books
6
Available Books
Recent Issued Books
Book Title
User
Use Date
Due Date
Java Programming
Kushager
April 16, 2024
April 23, 2024
Management
Ankush
April 10, 2024
April 23, 2024
Krishan Sudama
Krishan
April 09, 2024
April 10, 2024
A Suitable Boy
Torun
April 03, 2024
April 10, 2024
Manage Books
Add New Book
Book ID
Title
Author
Total Security
Actions
01
Java Programming
Kushager
4
4
02
Management
Ankush
4
4
03
Krishan Sudama
Krishan
4
4
04
A Suitable Boy
Torun
4
4
05
Indian History
Shivm
4
5
Manage Users
Add New User
User ID
User
Search
ID
User Name
Actions
01
Kushager
02
Ankush
02
Kushan
04
Torun
03
Shinam
08
Abhishak
Issued Books
Issued Books
Search
Issue ID
Book Title
User Name
Issue Date
01
Java Programming
Kushager
April 16, 2024
02
Management
Ankush
April 10, 2024
03
Krishan Sudama
Krishan
April 10, 2024
04
A Suitable Boy
Shivam
April 16, 2024

REFERENCES

1. **Pivotal Software, Inc.** *Spring Boot Documentation.*
   Available at: https://spring.io/projects/spring-boot
2. **Oracle Corporation.** *Java SE 17/21 Documentation.*
   Available at: https://docs.oracle.com/javase/
3. **Hibernate Community.** *Hibernate ORM User Guide.*
   Available at: https://hibernate.org/orm/documentation/
4. **Jackson Project.** *FasterXML Jackson – JSON Processing for Java.*
   Available at: https://github.com/FasterXML/jackson
5. **Auth0.** *Understanding JSON Web Tokens (JWT).*
   Available at: https://auth0.com/learn/json-web-tokens/
6. **Spring Security Team.** *Spring Security Reference Documentation.*
   Available at: https://spring.io/projects/spring-security
7. **Mozilla Developer Network (MDN).** *HTML, CSS, and JavaScript Documentation.*
   Available at: https://developer.mozilla.org/
8. **JetBrains s.r.o.** *IntelliJ IDEA Documentation.*
   Available at: https://www.jetbrains.com/idea/documentation/
9. **Postman Team.** *Postman API Testing Documentation.*
   Available at: https://learning.postman.com/docs/
10. **MySQL Documentation.** *MySQL Developer Guide & Reference Manual.*
    Available at: https://dev.mysql.com/doc/
11. **W3C.** *Web Standards for HTML, CSS, and Web APIs.*
    Available at: https://www.w3.org/
12. **GitHub, Inc.** *GitHub Documentation – Repositories & Version Control.*
    Available at: https://docs.github.com/

This Project completed by Kushagar and Ankush

Project Upload on Github: https://github.com/kushagar