Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

The **Library Management System (LMS)** is a web-based application designed to automate the management of books, users, and borrowing processes within a library. The system enhances efficiency by providing a centralized platform for book tracking, lending management, and overdue notifications.

1.2 Document Conventions

- Code: Follows **CamelCase** for variables and **PascalCase** for classes.
- API endpoints: Use RESTful naming conventions (e.g., `/api/books`, `/api/users`).
- Database: Table names follow **snake_case** (e.g., `users`, `books`).

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- Developers implementing the system.
- Project managers tracking progress.
- Testers validating system functionality.

1.4 Scope

The **Library Management System** provides functionality for:

- User authentication and role-based access control.
- Managing books, including adding, updating, deleting, and searching.
- Borrowing and returning books with due date tracking.
- Notifications for overdue books and automatic fine calculation.
- Ensuring data security with authentication and role management.

1.5 References

- **Level 2 - Intermediate Tier Requirements**
- **BRD for LMS**
- **Project Plan & Milestones**
- **Coding Style Guide**

2. Overall Description

2.1 Product Perspective

The system follows a **three-tier architecture**:

1. **Frontend:** React.js with React Router for UI.
2. **Backend:** Spring Boot RESTful APIs.
3. **Database:** MySQL for structured data storage.

2.2 Product Functions

- **User Management**: Registration, authentication, password reset.
- **Role-Based Access**: Admin (Librarian) and Member.
- **Book Management**: CRUD operations, search, filter.
- **Borrowing System**: Borrow/return books, track due dates.
- **Notifications**: Alerts for overdue books, fine calculation.

2.3 User Classes and Characteristics

- **Librarian (Admin)**: Manages users, books, lending.
- **Member**: Searches for books, borrows/returns books.

2.4 Operating Environment

- **Frontend**: React.js, Bootstrap/Material-UI
- **Backend**: Spring Boot, Spring Security, JPA/Hibernate
- **Database**: MySQL
- **OS**: Windows/Linux/MacOS
- **Browser Compatibility**: Chrome, Firefox, Edge

2.5 Design and Implementation Constraints

- JWT-based authentication for security.
- API requests must be validated before processing.
- Database queries must be optimized for efficiency.

2.6 Assumptions and Dependencies

- Users must have internet access.
- Libraries have a structured cataloging system.

3. Specific Requirements

3.1 Functional Requirements

User Management

- Users must be able to **register** and **log in**.
- Role-based access control (Admin/Member) should be enforced.
- Users should be able to **reset passwords** securely.

Book Management

- Admin can **add, update, delete, and view** books.
- Books should have fields: **ISBN, Title, Author, Category, Year, Status**.
- Books can be searched by **title, author, category**.

Borrowing and Return System

- Members can **borrow and return books**.
- Borrowed books must have a **due date** (14 days from borrow date).
- Members can view **borrowing history**.
- Books must update their **status** when borrowed/returned.

Overdue Notifications & Fines

- Scheduled tasks should check overdue books.
- Overdue books trigger **email/UI notifications**.
- Fines calculated as **$0.50 per day, max $20 per book**.

Security Requirements

- **JWT-based authentication** for API access.
- **Password encryption** before storage.
- **Role-based access** enforced on endpoints.

3.2 Non-Functional Requirements

Performance

- System should handle **100 concurrent users** efficiently.
- Database queries should be optimized for response time **<500ms**.

Reliability & Availability

- System uptime must be **99.5%**.
- Error handling should provide **meaningful messages**.

Maintainability & Scalability

- Code should follow best **modular programming** practices.
- System should support adding **new user roles** in the future.

Usability

- UI should be intuitive and responsive across devices.
- Search results should be paginated for large datasets.

4. External Interface Requirements

4.1 User Interfaces

- **Admin Dashboard**: Book and member management.
- **Member Portal**: Book search, borrowing history.

4.2 API Interfaces

Endpoint	**Method**	**Description**
`/api/auth/register`	POST	Register a new user
`/api/auth/login`	POST	Authenticate user
`/api/books`	GET	Get all books
`/api/books/{id}`	GET	Get book details
`/api/books`	POST	Add new book (Admin)
`/api/books/{id}`	PUT	Update book (Admin)
`/api/books/{id}`	DELETE	Delete book (Admin)
`/api/borrow/{id}`	POST	Borrow book
`/api/return/{id}`	POST	Return book

`/api/notifications/overdue`	GET	Get overdue books

4.3 Hardware Interfaces

- Server requires **8GB RAM, 4-core CPU, 50GB storage**.

4.4 Software Interfaces

- **Backend:** Java 17+, Spring Boot, Spring Security, Hibernate.
- **Frontend:** React.js, Axios.
- **Database:** MySQL, Docker (optional).

5. Other Requirements

- Full test coverage for unit and integration tests.
- Logging and monitoring tools should be implemented.
- System should be **GDPR-compliant** for user data privacy.

6. Appendices

- [BRD for LMS]
- [Project Plan & Milestones]
- [Coding Style Guide]