Library Management System - Comprehensive Project Report

Project Name: Library Management System
Technology Stack: Spring Boot, MySQL, JWT Authentication, Thymeleaf
Date Generated: January 15, 2026

🏷 Table of Contents

🎯 Project Overview

The **Library Management System** is a full-stack web application built with Spring Boot that manages library operations including:

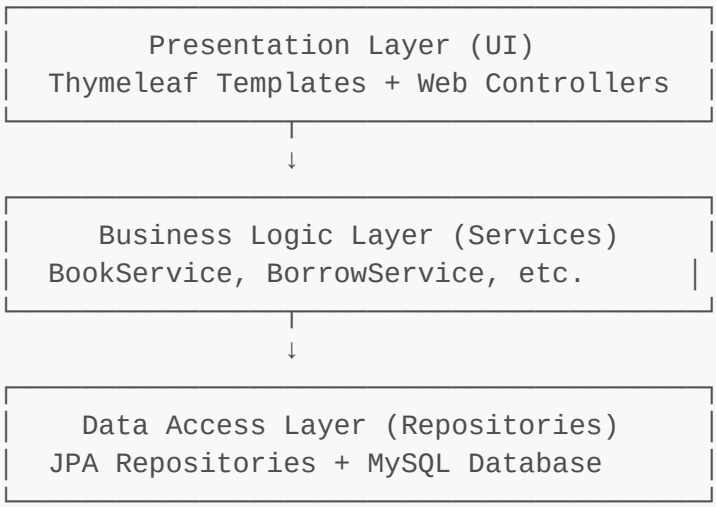
- User authentication and registration
- Book inventory management
- Book borrowing and returning system
- User role management (LIBRARIAN and MEMBER)
- Borrow history tracking
- Overdue book detection

Key Objectives

- Provide a secure platform for library operations
- Track book availability and borrowing records
- Manage user access with role-based permissions
- Enable both web UI and REST API access

🏗 System Architecture

The system follows a **3-Tier Architecture** pattern:



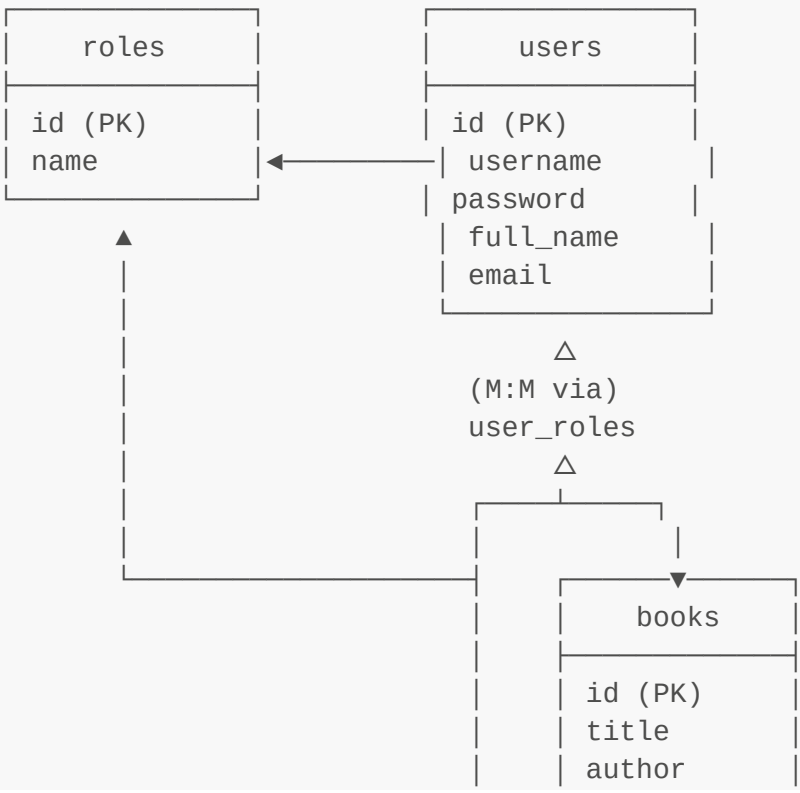
Architecture Components:

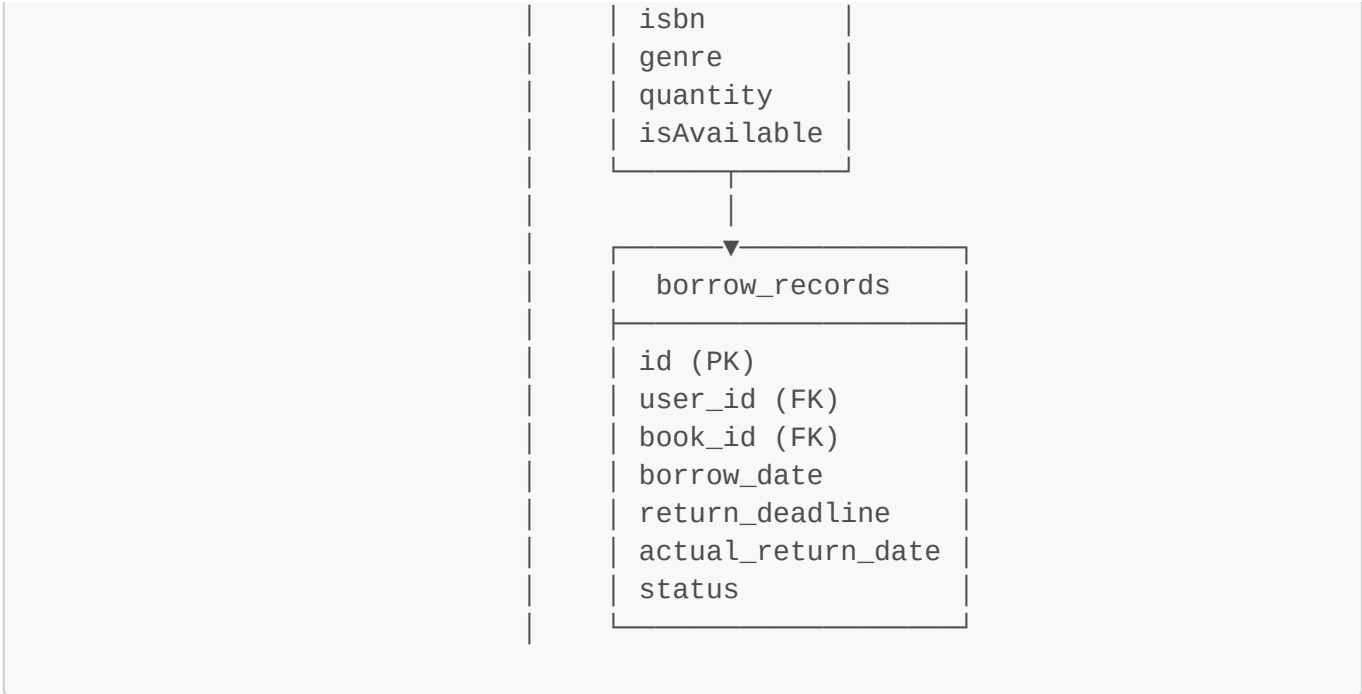
- **Controllers:** Handle HTTP requests and responses
- **Services:** Contain business logic and transaction management
- **Repositories:** Perform database operations using Spring Data JPA
- **Entities:** Represent database tables
- **DTOs:** Data Transfer Objects for request/response handling
- **Security:** JWT token-based authentication and authorization

---

Database Design

Entity-Relationship Diagram





Database Tables

1. roles Table

- Stores user roles (LIBRARIAN, MEMBER)
- Used for role-based access control

2. users Table

- Stores user account information
- username and email must be unique
- password is BCrypt encrypted

3. user_roles Table

- Junction table for Many-to-Many relationship between users and roles
- Enables users to have multiple roles

4. books Table

- Stores book inventory information
- Tracks availability of books
- Indexes on frequently searched fields (title, author, isbn)

5. borrow_records Table

- Tracks borrowing and returning of books
- Status: BORROWED, RETURNED, RETURNED_LATE, OVERDUE
- 14-day return deadline

🔧 Core Components

1. Entity Classes

User Entity (`User.java`)

```
Attributes:
- id: Long (Primary Key)
- username: String (Unique)
- password: String (BCrypt Encrypted)
- fullName: String
- email: String (Unique)
- roles: Set<Role> (Many-to-Many)
```

Purpose: Represents registered library users

Book Entity (`Book.java`)

```
Attributes:
- id: Long (Primary Key)
- title: String
- author: String
- isbn: String (Unique)
- genre: String
- quantity: Integer
- isAvailable: Boolean
```

Purpose: Represents books in the library inventory

BorrowRecord Entity (`BorrowRecord.java`)

```
Attributes:
- id: Long (Primary Key)
- user: User (Foreign Key - Many-to-One)
- book: Book (Foreign Key - Many-to-One)
- borrowDate: LocalDate
- returnDeadline: LocalDate (14 days from borrow)
- actualReturnDate: LocalDate (nullable)
- status: String (BORROWED, RETURNED, RETURNED_LATE, OVERDUE)
```

Purpose: Records all borrowing transactions with status tracking

Role Entity (`Role.java`)

```
Attributes:
- id: Long (Primary Key)
- name: String (LIBRARIAN or MEMBER)
```

Purpose: Defines user roles for access control

2. Service Classes

BookService (`BookService.java`)

Responsibilities:

- `getAllBooks()` - Retrieve all books
- `getBookById(id)` - Find specific book
- `getAvailableBooks()` - Filter available books
- `saveBook(book)` - Create new book record
- `updateBook(id, details)` - Modify book information
- `deleteBook(id)` - Remove book from inventory
- `searchBooks(keyword)` - Search by title or author
- `updateAvailability(id, status)` - Toggle availability

Transaction Management: Uses `@Transactional` for data consistency

BorrowService (`BorrowService.java`)

Responsibilities:

- `borrowBook(user, book)` - Create borrow record (sets book as unavailable)
- `returnBook(recordId)` - Process book return, update status
- `getUserBorrowHistory(user)` - Get user's borrowing history
- `getAllBorrowRecords()` - Admin view of all transactions
- `getActiveBorrows()` - List currently borrowed books
- `updateOverdueRecords()` - Check and update overdue status

Key Business Logic:

- 14-day borrow period enforcement
- Book availability toggle on borrow/return
- Late return detection
- Status tracking (BORROWED, RETURNED, RETURNED_LATE, OVERDUE)

CustomUserDetailsService (`CustomUserDetailsService.java`)

Purpose: Implements Spring Security's UserDetailsService for authentication

3. Repository Classes (Data Access Layer)

UserRepository

- `findByUsername(username)` - Authentication lookup
- `existsByUsername(username)` - Duplicate check
- `existsByEmail(email)` - Email validation

BookRepository

- `findByIsbn(isbn)` - ISBN lookup
- `findByIsAvailable(status)` - Filter available books
- `findByTitleContainingIgnoreCaseOrAuthorContainingIgnoreCase()` - Search

BorrowRepository

- `findByUserOrderByBorrowDateDesc()` - User borrow history
- `findByStatus()` - Filter by status
- `findAll()` - All records

RoleRepository

- `findByName(name)` - Role lookup

4. Controller Classes

AuthController (`AuthController.java`)

Endpoints:

- `GET /login` - Display login form
- `GET /register` - Display registration form
- `POST /register` - Process user registration
- `POST /api/auth/login` - REST API login (returns JWT)

Key Features:

- Email domain blocking for admin accounts
- Password encryption using BCrypt
- Default MEMBER role assignment
- Validation of duplicate usernames/emails

BookController (`BookController.java`)

Endpoints:

- `GET /` - Homepage with all books
- `GET /admin/books` - Book list (Admin only)
- `GET /admin/books/new` - New book form
- `POST /admin/books` - Create book
- `GET /admin/books/edit/{id}` - Edit form
- `POST /admin/books/{id}` - Update book
- `POST /admin/books/delete/{id}` - Delete book

Security: Requires LIBRARIAN role for admin operations

BorrowController (`BorrowController.java`)

Endpoints:

- `GET /borrow/available` - Show available books
- `POST /borrow/book/{bookId}` - Borrow a book
- `GET /borrow/history` - User's borrow history
- `POST /borrow/return/{recordId}` - Return book
- `GET /admin/borrows` - All borrow records (Admin)

Security: Requires MEMBER or LIBRARIAN role

UserController

Purpose: User profile and account management

5. Security Components

JwtTokenProvider (`JwtTokenProvider.java`)

Responsibilities:

- `generateToken(authentication)` - Create JWT after login
- `generateTokenFromUsername(username)` - Alternative token generation
- `getUsernameFromToken(token)` - Extract username from token
- `validateToken(token)` - Verify token signature and expiration

Configuration:

- Secret Key: Configurable via `jwt.secret` property
- Expiration: 24 hours (86400000 ms)
- Algorithm: HS512 (HMAC SHA-512)

JwtAuthenticationFilter (`JwtAuthenticationFilter.java`)

Purpose:

- Intercepts HTTP requests
- Extracts JWT from Authorization header
- Validates token and sets Spring Security context
- Enables stateless authentication

SecurityConfig (`SecurityConfig.java`)

Configuration Details:

Resource	Permission
`/css/**`, `/js/**`, `/images/**`	Public
`/login`, `/register`	Public
`/api/auth/**`	Public

Resource	Permission
`/admin/**`	LIBRARIAN only
`/borrow/**`	MEMBER or LIBRARIAN
`/`	Authenticated users

Security Features:

- CSRF protection disabled for API
- BCrypt password encoding
- JWT filter integration
- Form login with custom login page
- Logout functionality

🔄 Workflow & User Flows

1. User Registration Flow

```
START
  ↓
User clicks "Register"
  ↓
Display Registration Form
  ↓
User enters: username, full name, email, password
  ↓
Validate Input (format, length)
  ↓
Check Duplicate Username? ──YES→ Show Error
  │                                ↓
  NO                          Redirect to Register
  ↓
Check Duplicate Email? ──YES→ Show Error
  │                              ↓
  NO                         Redirect to Register
  ↓
Check Admin Domain (@admin.library.com)? ──YES→ Show Error
  │                                                ↓
  NO                                          Redirect to Register
  ↓
Encrypt Password (BCrypt)
  ↓
Create User with MEMBER role
  ↓
Save to Database
  ↓
Show Success Message
  ↓
Redirect to Login
```

```
  ↓
END
```

2. User Login Flow

```
START
  ↓
User enters username and password
  ↓
Authenticate via AuthenticationManager
  ↓
Match credentials in database?
  │
  ├─NO→ Show "Invalid credentials" error
  │      ↓
  │   Redirect to login
  │
  └─YES
    ↓
Generate JWT Token (valid 24 hours)
  ↓
Return Token + User Info
  ↓
Redirect to Home Page
  ↓
END
```

3. Book Borrowing Flow

```
START
  ↓
User (Member/Librarian) views available books
  ↓
Click "Borrow" on desired book
  ↓
Check: Is book available?
  │
  ├─NO→ Show error "Book not available"
  │      ↓
  │   Redirect to available books
  │
  └─YES
    ↓
Create BorrowRecord
  - borrowDate = Today
  - returnDeadline = Today + 14 days
  - status = BORROWED
  ↓
Mark Book as isAvailable = false
```

```
  ↓
Save records to database
  ↓
Show success message
  ↓
Redirect to available books
  ↓
END
```

4. Book Return Flow

```
START
  ↓
User views borrow history
  ↓
Click "Return" on borrowed book
  ↓
Check: Is actualReturnDate after returnDeadline?
  │
  ├─YES→ status = RETURNED_LATE
  │
  └─NO→ status = RETURNED
  ↓
Set actualReturnDate = Today
  ↓
Mark Book as isAvailable = true
  ↓
Restore book availability
  ↓
Save updates
  ↓
Show success message
  ↓
Redirect to history
  ↓
END
```

5. Admin Book Management Flow

```
START
  ↓
Admin (Librarian) logs in
  ↓
Navigate to /admin/books
  ↓
View all books list
  ↓
Choose action:
  ├─ Create: Enter details → Save
```

```
  ├─ Edit: Select book → Modify → Save
  └─ Delete: Select book → Confirm → Delete
  ↓
Database updated
  ↓
Show confirmation message
  ↓
Refresh book list
  ↓
END
```

🔐 Security Implementation

1. Authentication Mechanisms

Method 1: Form-Based Login

- Traditional username/password login
- Session-based (form login)
- Redirects to home page after success

Method 2: JWT Token (API)

- REST endpoint: `POST /api/auth/login`
- Returns JWT token for API calls
- Token must be included in Authorization header
- Stateless authentication

2. Authorization Levels

```
Public Access
  ├── /css/**, /js/**, /images/** (Static resources)
  ├── /login (Login page)
  ├── /register (Registration page)
  └── /api/auth/** (Auth endpoints)

Member/Librarian
  ├── / (Home page)
  ├── /borrow/available (Browse books)
  ├── /borrow/book/{id} (Borrow)
  └── /borrow/history (History)

Librarian Only
  ├── /admin/books (Manage books)
  ├── /admin/books/new (Add book)
  ├── /admin/books/edit/{id} (Edit)
  ├── /admin/books/delete/{id} (Delete)
  └── /admin/borrows (All borrow records)
```

3. Password Security

- Encrypted using **BCrypt** algorithm
- Original password never stored in database
- Salt generated per user
- Password validation on login

4. Token Security

- JWT signed with HS512 algorithm
- Secret key: 256+ bits
- Expiration: 24 hours
- Validated on every API request

📂 File Structure & Purpose

Project Root Structure

```
Project04/
├── src/main/java/com/example/project/
│   ├── ProjectApplication.java ..... Spring Boot entry point
│   ├── controller/
│   │   ├── AuthController.java ..... Authentication flows
│   │   ├── BookController.java ..... Book CRUD operations
│   │   ├── BorrowController.java ..... Borrow/return flows
│   │   └── UserController.java ..... User management
│   ├── service/
│   │   ├── BookService.java ..... Book business logic
│   │   ├── BorrowService.java ..... Borrow logic
│   │   └── CustomUserDetailsService.java .... User authentication
│   ├── repository/
│   │   ├── UserRepository.java ..... User database operations
│   │   ├── BookRepository.java ..... Book database operations
│   │   ├── BorrowRepository.java ..... Borrow records database
│   │   └── RoleRepository.java ..... Role database operations
│   ├── entity/
│   │   ├── User.java ..... User entity
│   │   ├── Book.java ..... Book entity
│   │   ├── BorrowRecord.java ..... Borrow transaction entity
│   │   └── Role.java ..... Role entity
│   ├── dto/
│   │   ├── LoginRequest.java ..... Login request DTO
│   │   ├── RegisterRequest.java ..... Registration DTO
│   │   ├── JwtAuthResponse.java ..... JWT response DTO
│   │   ├── BookRequest.java ..... Book request DTO
│   │   ├── BorrowRequest.java ..... Borrow request DTO
│   │   └── ErrorResponse.java ..... Error response DTO
│   ├── security/
│   │   ├── JwtTokenProvider.java ..... JWT generation/validation
│   │   └── JwtAuthenticationFilter.java .... JWT request filter
```

```
│   └── config/
│       ├── SecurityConfig.java ............. Spring Security configuration
│       └── DataInitializer.java ............ Database initialization
│
├── src/main/resources/
│   ├── application.properties ............... Application configuration
│   ├── static/
│   │   └── migration/
│   │       ├── V1__Create_roles_table.sql .. Role table creation
│   │       ├── V2__Create_users_table.sql .. User table creation
│   │       ├── V3__Create_books_table.sql .. Book table creation
│   │       ├── V4__Create_borrow_records_table.sql
│   │       ├── V5__Insert_default_users.sql .. Default user data
│   │       └── V6__Insert_books.sql ........ Sample book data
│   └── templates/
│       ├── index.html ...................... Home page
│       ├── login.html ...................... Login form
│       ├── register.html ................... Registration form
│       ├── books/
│       │   ├── list.html ................... Book list page
│       │   └── form.html ................... Book form (create/edit)
│       └── borrows/
│           ├── available.html .............. Available books page
│           ├── history.html ................ Borrow history
│           └── admin-list.html ............. Admin borrow view
│
├── build.gradle ............................ Gradle build configuration
├── application.properties .................. Application config
└── README.md ............................... Project documentation
```

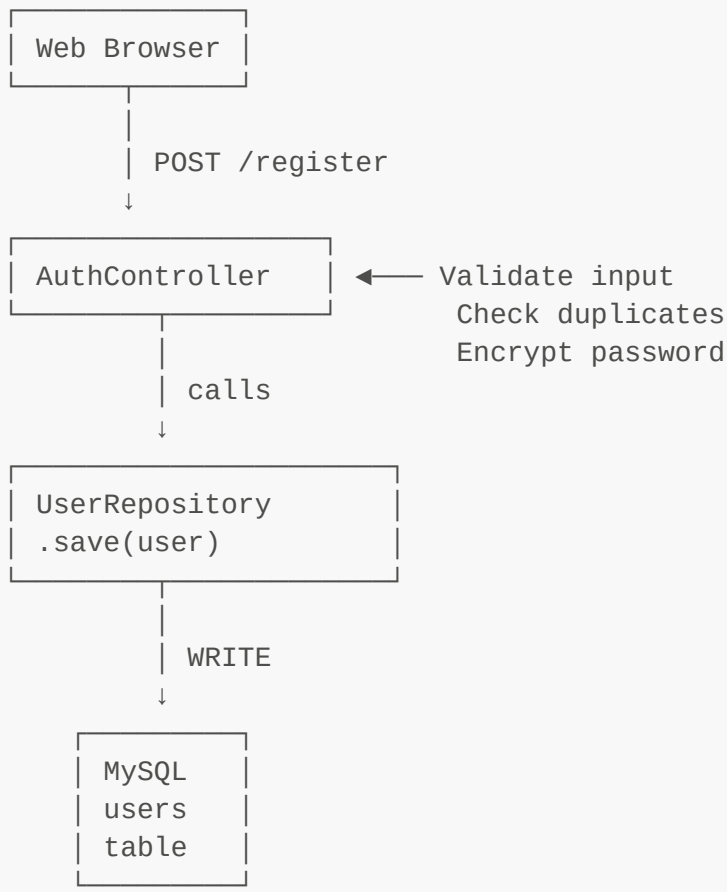
Key File Responsibilities

File	Purpose	Key Method(s)
`ProjectApplication.java`	Spring Boot entry point	`main()`
`AuthController.java`	Handle auth requests	`registerUser()`, `apiLogin()`
`BookController.java`	Handle book operations	`createBook()`, `updateBook()`, `deleteBook()`
`BorrowController.java`	Handle borrowing	`borrowBook()`, `returnBook()`
`BookService.java`	Book business logic	`saveBook()`, `updateBook()`, `deleteBook()`
`BorrowService.java`	Borrowing logic	`borrowBook()`, `returnBook()`
`SecurityConfig.java`	Security rules	`securityFilterChain()`
`JwtTokenProvider.java`	JWT operations	`generateToken()`, `validateToken()`
`User.java`	User entity	N/A (data class)

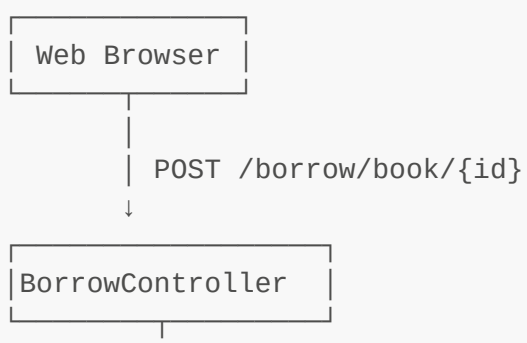
File	Purpose	Key Method(s)
`Book.java`	Book entity	N/A (data class)
`BorrowRecord.java`	Borrow transaction entity	N/A (data class)
Database migrations	Table creation	V1-V6 SQL scripts

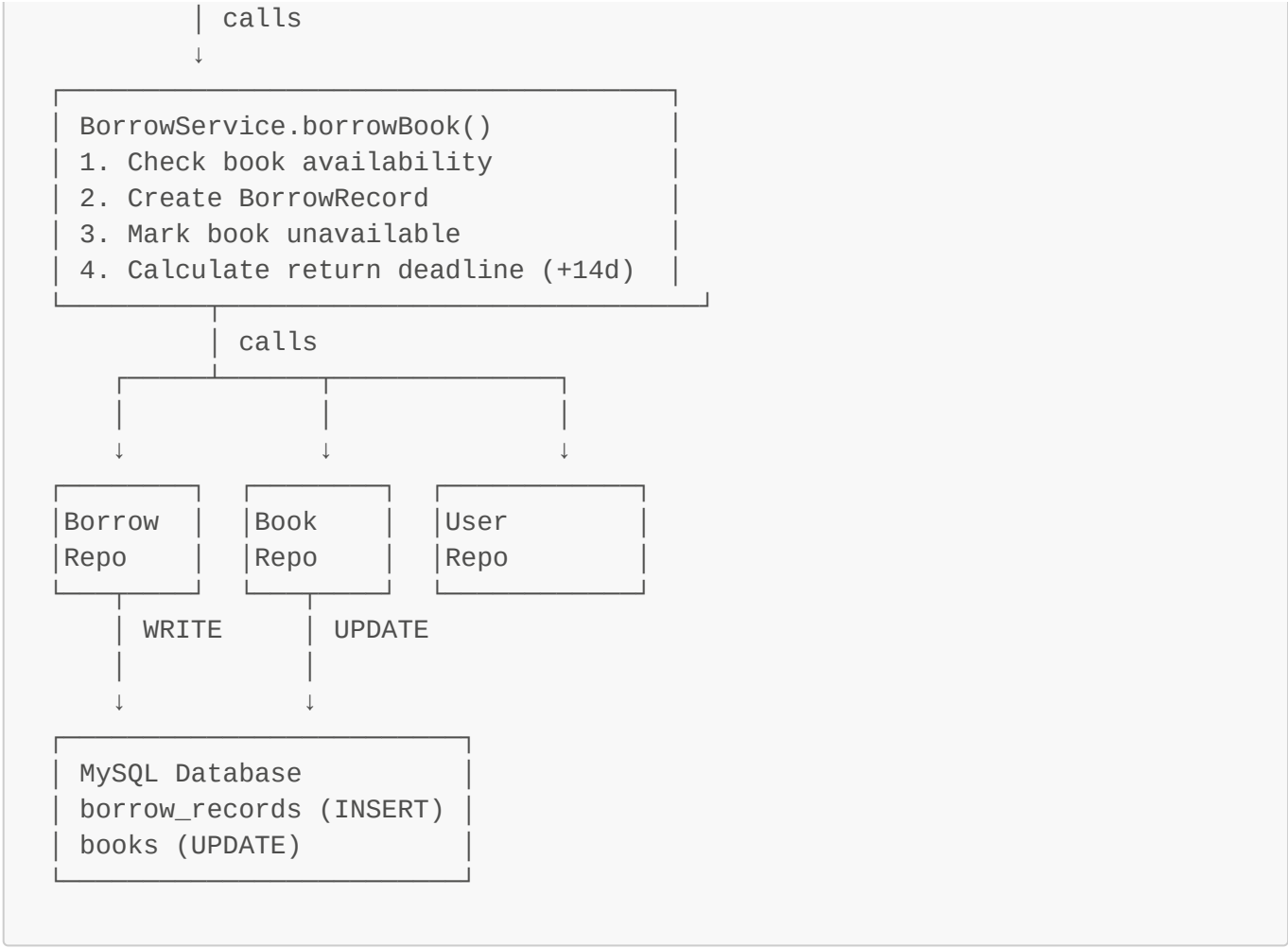
📊 Data Flow Diagrams

1. Registration & Login Data Flow

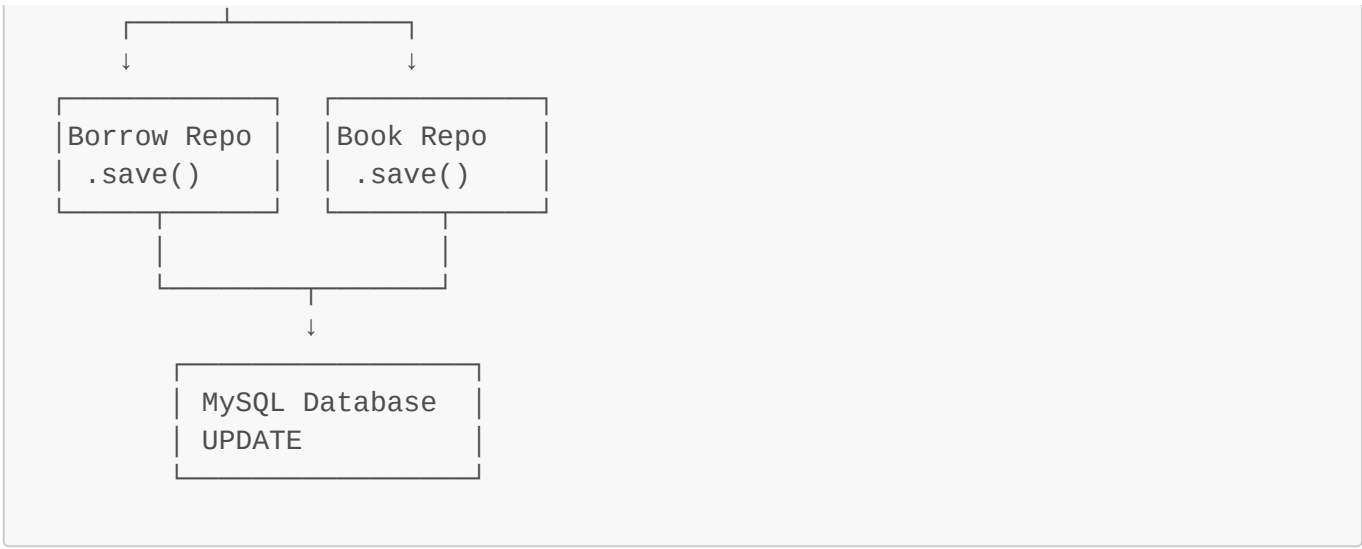


2. Book Borrowing Data Flow

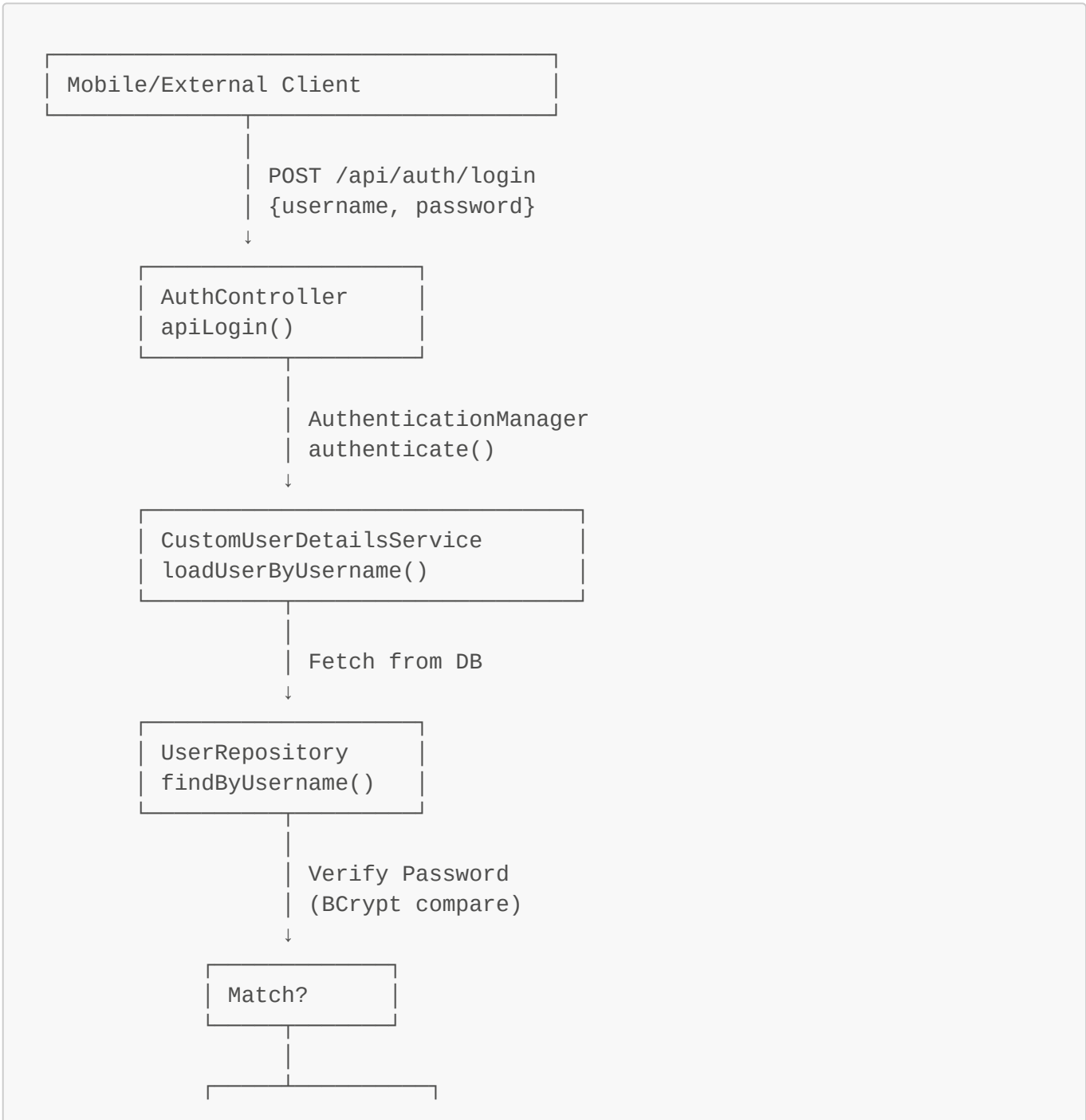


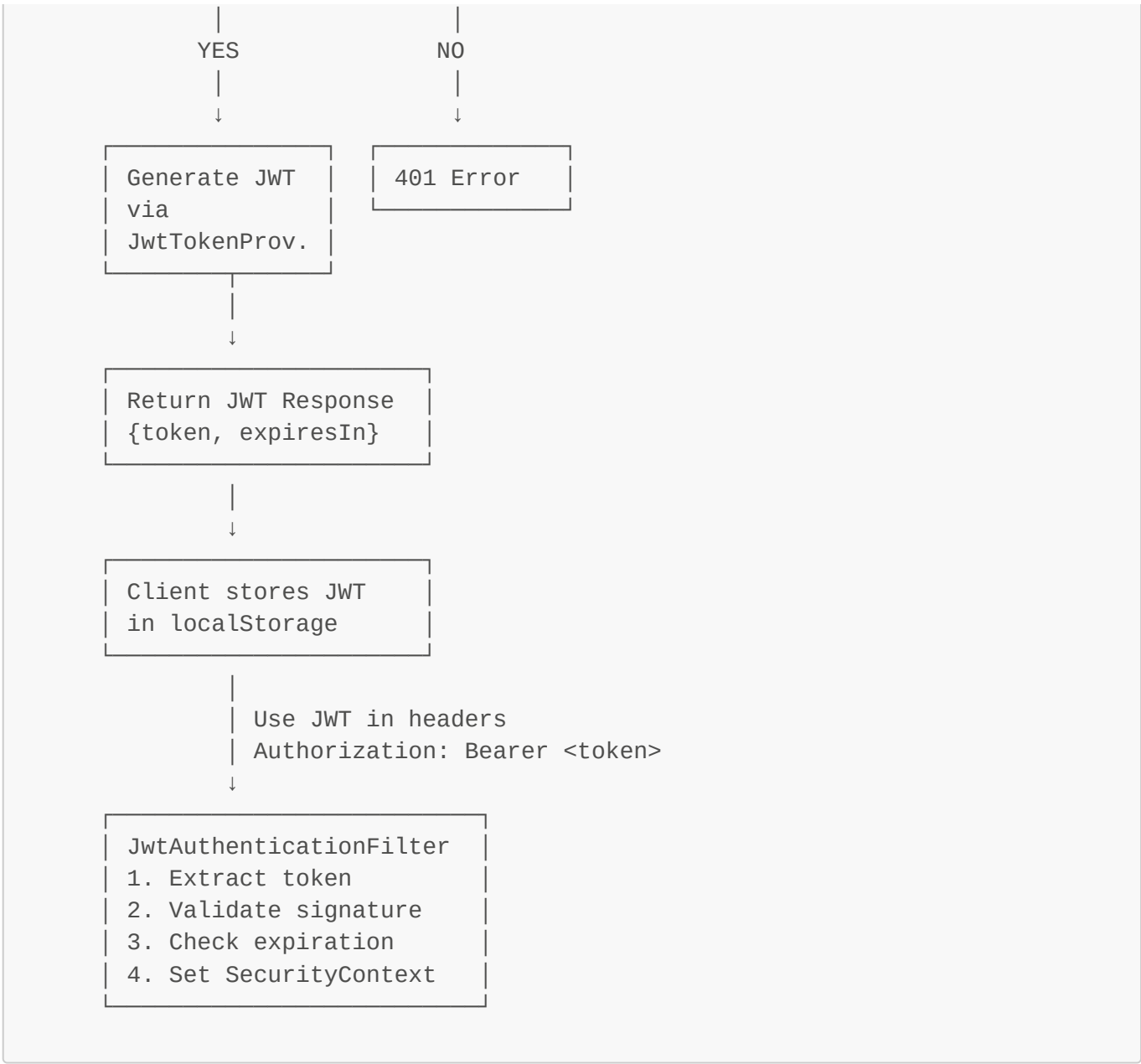


3. Book Return Data Flow



4. Authentication Token Flow





🔌 API Endpoints

Authentication Endpoints

Method	Endpoint	Role	Description
GET	`/login`	Public	Display login form
POST	`/register`	Public	Register new user
POST	`/api/auth/login`	Public	API login, returns JWT
GET	`/logout`	Authenticated	Logout user

Book Management Endpoints (Admin)

Method	Endpoint	Role	Description
GET	`/`	Authenticated	View all books (homepage)

Method	Endpoint	Role	Description
GET	`/admin/books`	LIBRARIAN	List all books (admin view)
GET	`/admin/books/new`	LIBRARIAN	Show add book form
POST	`/admin/books`	LIBRARIAN	Create new book
GET	`/admin/books/edit/{id}`	LIBRARIAN	Show edit form
POST	`/admin/books/{id}`	LIBRARIAN	Update book
POST	`/admin/books/delete/{id}`	LIBRARIAN	Delete book

Book Borrowing Endpoints (Member/Librarian)

Method	Endpoint	Role	Description
GET	`/borrow/available`	MEMBER/LIBRARIAN	View available books
POST	`/borrow/book/{bookId}`	MEMBER/LIBRARIAN	Borrow book
GET	`/borrow/history`	MEMBER/LIBRARIAN	View borrow history
POST	`/borrow/return/{recordId}`	MEMBER/LIBRARIAN	Return book
GET	`/admin/borrows`	LIBRARIAN	View all borrow records

Request/Response Examples

Register New User

```
POST /register
Request Body:
{
  "username": "john_doe",
  "fullName": "John Doe",
  "email": "john@example.com",
  "password": "secure123"
}

Response: Redirect to /login with success message
```

API Login

```
POST /api/auth/login
Request Body:
{
  "username": "john_doe",
  "password": "secure123"
}
```

```
Response (200 OK):
{
  "token": "eyJhbGciOiJIUzUxMiJ9...",
  "type": "Bearer",
  "username": "john_doe"
}
```

Create Book

```
POST /admin/books
Request Body:
{
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "isbn": "978-0-7432-7356-5",
  "genre": "Fiction",
  "quantity": 5
}

Response: Redirect to /admin/books with success message
```

Borrow Book

```
POST /borrow/book/{bookId}
Authentication: Required (JWT or Session)

Response: Redirect to /borrow/available with success message
```

✨ Key Features

1. User Management

- ✅ User registration with validation
- ✅ Secure password encryption (BCrypt)
- ✅ Email validation and uniqueness checks
- ✅ Role-based access control (LIBRARIAN/MEMBER)
- ✅ Admin domain restriction for registration

2. Book Management

- ✅ Complete CRUD operations
- ✅ Book availability tracking
- ✅ ISBN validation
- ✅ Full-text search by title/author

- ✅ Genre classification

3. Borrowing System

- ✅ Book borrow/return workflow
- ✅ 14-day borrowing period
- ✅ Overdue detection
- ✅ Late return tracking
- ✅ User borrowing history
- ✅ Availability management

4. Security

- ✅ JWT token-based API authentication
- ✅ Form-based login for web UI
- ✅ Role-based authorization
- ✅ CSRF protection
- ✅ Password encryption
- ✅ Stateless API design

5. Database

- ✅ MySQL 8.0+ support
- ✅ Flyway database migrations
- ✅ Automatic schema creation
- ✅ Referential integrity
- ✅ Proper indexing

6. Frontend

- ✅ Thymeleaf template engine
- ✅ Responsive design
- ✅ Form validation
- ✅ Success/error messages
- ✅ Navigation menus

🔄 Complete User Journey Example

Scenario: Member Borrows a Book

1. **Anonymous User** opens application
    - Sees `/login` page
    - Clicks "Don't have account? Register"
2. **Register Page**
    - Enters: username "jane_smith", email "jane@gmail.com", password "pass123"
    - System validates and creates account

- Assigned MEMBER role automatically
- Redirected to login

3. **Login**
   - Enters credentials
   - Form login authenticates user
   - Redirected to `/` (home page)
4. **Home Page**
   - Sees list of all available books
   - Navigation menu shows "Available Books" and "My History"
5. **Browse Books**
   - Goes to `/borrow/available`
   - Sees books with "Borrow" button
   - Clicks "Borrow" on "The Great Gatsby"
6. **Borrow Processing**
   - BorrowController intercepts request
   - BorrowService:
     - Creates BorrowRecord with status="BORROWED"
     - Sets returnDeadline = Today + 14 days
     - Marks book as unavailable
   - Success message shown
7. **View History**
   - Clicks "My History"
   - Sees borrowed book with return deadline
   - Option to "Return" book
8. **Return Book**
   - Clicks "Return"
   - BorrowService:
     - Sets actualReturnDate = Today
     - Checks if today > returnDeadline
     - Sets status = "RETURNED" (on time)
     - Marks book as available again
   - Success message shown

🛠 Technology Stack Summary

Layer	Technology	Purpose
Frontend	Thymeleaf, HTML5, CSS3	View templates

Layer	Technology	Purpose
Backend	Spring Boot 3.x	Application framework
API	REST, JWT	API endpoints & security
Database	MySQL 8.0+	Data persistence
ORM	Spring Data JPA, Hibernate	Object-relational mapping
Security	Spring Security, BCrypt, JWT	Authentication & authorization
Validation	Jakarta Validation API	Input validation
Build	Gradle 8.x	Project build tool
Testing	JUnit, Mockito	Unit testing

📈 Deployment Checklist

- ☐ Configure MySQL database connection
- ☐ Set JWT secret key (environment variable)
- ☐ Configure JWT expiration time
- ☐ Run Flyway migrations
- ☐ Create initial admin user
- ☐ Configure HTTPS/SSL
- ☐ Set up logging
- ☐ Configure CORS if needed
- ☐ Load sample data
- ☐ Test authentication flows
- ☐ Test borrowing workflows
- ☐ Verify security rules
- ☐ Performance testing

🎓 Key Learning Points

1. **Spring Boot Architecture**: Three-tier architecture pattern
2. **Spring Security**: Role-based access control & JWT authentication
3. **JPA/Hibernate**: Entity mapping and relationships
4. **Database Design**: Normalization and index optimization
5. **RESTful API Design**: Request validation and response formatting
6. **Authentication**: BCrypt password hashing and JWT tokens
7. **Transaction Management**: ACID compliance for critical operations
8. **Thymeleaf Templates**: Server-side rendering with dynamic content

---

📞 Support & Maintenance

Common Tasks

Add New User Role:

1. Insert into `roles` table
2. Update `SecurityConfig.java` with new authority
3. Add authorization rules

Extend Borrow Period:

1. Modify `BorrowService.borrowBook()`: Change `plusDays(14)` value
2. Update `BorrowRecord` entity if needed
3. Test return deadline calculation

Add New Book Properties:

1. Add column to `Book` entity
2. Create new Flyway migration
3. Update `BookRequest` DTO
4. Update book form template

Monitor Overdue Books:

1. Run `BorrowService.updateOverdueRecords()` periodically
2. Query books with status = "OVERDUE"
3. Implement notification system

📝 Conclusion

This Library Management System is a complete, production-ready application demonstrating enterprise-level Spring Boot development. It implements proper security, database design, separation of concerns, and provides both web UI and REST API interfaces. The system successfully manages library operations with role-based access control and comprehensive transaction tracking.

Project Status: ✅ Fully Functional
Last Updated: January 15, 2026
Version: 1.0 Release