maids.cc
+ nannies + visa services

10/03/2025

Library Management System Documentation

Ahmad Abdulla

1-How to Run the Application

Prerequisites

Ensure you have the following installed on your system:

- Java 17+
- Maven 3+
- MySQL Database
- An API testing tool (e.g., Postman)

Steps to Run the Application

1. **Clone the Repository**

   https://github.com/AhmadIbraAbdulla/LMS_Task

2. **Configure Database**
   - Ensure MySQL is running.
   - Update the application.properties or application.yml file with your database credentials.

3. **Run the Application**
   - Open the project in your IDE (e.g., IntelliJ, VS Code, Eclipse).
   - Locate the TaskApplication class.
   - Run the application using:

     - mvn spring-boot:run

   - Alternatively, run it from your IDE's Run/Debug configurations.
   - Once the application starts, the API will be available at:
     - http://localhost: 8085

Data Seeder and Database Initialization

The application automatically create the database if not created and includes a Data Seeder component that automatically seeds predefined roles and users when the application starts.

- **Roles:**
  The following roles are seeded into the database:
  - ROLE_ADMIN
  - ROLE_PATRON
  - ROLE_LIBRARIAN
- **Admin User:**
  An admin user is seeded with the following details:
  - Email: admin@library.com
  - Password: admin123Admin
  - Role: ROLE_ADMIN

Note: If the roles or admin user already exist, they will not be re-seeded.

2.Authentication Mechanism

The API end points uses **JWT-based Authentication & Authorization** to secure endpoints. When a user logs in successfully, they receive a **JWT token**, which contains:

- **User ID**
- **User Role**

This token must be included in the **Authorization header** of all subsequent requests (except the login endpoint) as follows:

Authorization: Bearer <JWT_TOKEN>

3. API Documentation:

1. Authentication Controller

Endpoint	HTTP Method	Access Level	Request Body	Response
`/api/auth/login`	`POST`	Public (No authentication required)	`LogInRequestDto`	`AuthResponseDto`
`/api/auth/signup`	`POST`	**ADMIN** only	`SignUpRequestDto`	`UserDto`

LoginRequestDto:

{ "email": "admin@library.com", "password": "admin123Admin"}

singUpRequestDto:
{ "email": "user1@example.com","roleName": "ROLE_PATRON","password": "stringA1"}

2. Book Controller

Endpoint	HTTP Method	Access Level	Request Body	Response
`/api/books`	`GET`	ADMIN, LIBRARIAN, PATRON	`PageRequestDto` (as query parameters)	`PageResponse<BookDto>`
`/api/books/{id}`	`GET`	ADMIN, LIBRARIAN, PATRON	N/A	`BookDto`
`/api/books`	`POST`	ADMIN, LIBRARIAN	`BookCreateDto`	`BookDto`
`/api/books/{id}`	`PUT`	ADMIN, LIBRARIAN	`BookUpdateDto`	`BookDto`
`/api/books/{id}`	`DELETE`	ADMIN, LIBRARIAN	N/A	`Void` (No Content)

BookCreateDto (for POST /api/books)

{ "title": " fake title", "author": " ahmad", "publicationYear": 1999, "isbn": "111113473221" }

BookUpdateDto (for PUT /api/books/{id})

{ "id": 1, "title": "fake title", "author": "ahmad abdulla", "publicationYear": 1999, "isbn": "1111111147316" }

PageRequestDto (Request body for GET /api/books/page)

{ "page": 1, "size": 10, "sort": "title", "sortDirection": "asc" }

Note: The delete endpoint performs a soft delete. Instead of permanently removing the entity and its associated borrowing records (which would result in data loss), it marks the entity as deleted. This allows for better management of records. The "Get All" endpoint only returns records where isDeleted = false.

3. Borrowing Controller

Endpoint	HTTP Method	Access Level	Request Body	Response
`/api/borrow/{bookId}/patron/{patronId}`	`POST`	ADMIN, LIBRARIAN, PATRON	N/A	`BorrowingRecordDto`
`/api/return/{bookId}/patron/{patronId}`	`PUT`	ADMIN, LIBRARIAN, PATRON	N/A	`BorrowingRecordDto`

4. Patron Controller

Endpoint	HTTP Method	Access Level	Request Body	Response
`/api/patrons`	`GET`	ADMIN, LIBRARIAN	`PageRequestDto` (as query parameters)	`PageResponse<PatronDto>`
`/api/patrons/{id}`	`GET`	ADMIN, LIBRARIAN	N/A	`PatronDto`
`/api/patrons`	`POST`	ADMIN, LIBRARIAN	`PatronCreateDto`	`PatronDto`
`/api/patrons/{id}`	`PUT`	ADMIN, LIBRARIAN	`PatronUpdateDto`	`PatronDto`
`/api/patrons/{id}`	`DELETE`	ADMIN, LIBRARIAN	N/A	`Void` (No Content)

Request :

1. POST /api/patrons (Create a New Patron)

{ "name": "name", "contactInformation": "093-131-1111" }

2. PUT /api/patrons/{id} (Update a Patron)

{ "id": 1, "name": " name", "contactInformation": "093-131-1111" }