Library Management System

Objective

The **Library Management System** provides a simple and efficient way to manage library operations.
It enables borrowers to:

- **Register as borrowers** with the library
- **Borrow books** from the library collection
- **Return borrowed books** to the library

This system is designed to streamline borrowing and returning processes while maintaining accurate records of borrowers and available books.

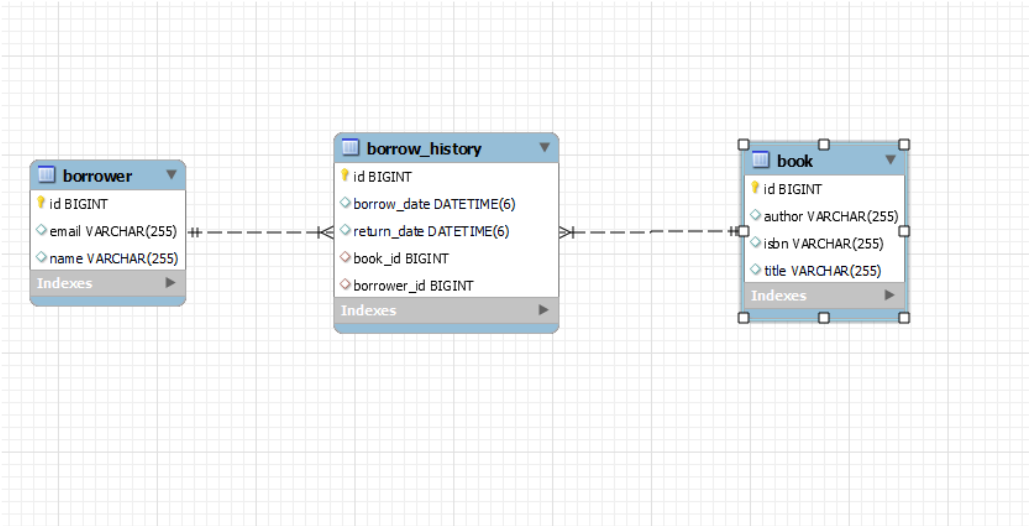
Terminology

An **ISBN (International Standard Book Number)** is a unique code that acts as a book's fingerprint.

- **What it does:** It identifies a specific edition and format of a book (e.g., hardcover, paperback, or ebook).
- **Its purpose:** It's the global standard used by publishers, libraries, and bookstores to efficiently order, catalog, and manage inventory.

- **Where to find it:** It's the 10 or 13-digit number found on the book's back cover, typically with the barcode.

Database Design



Process Flow

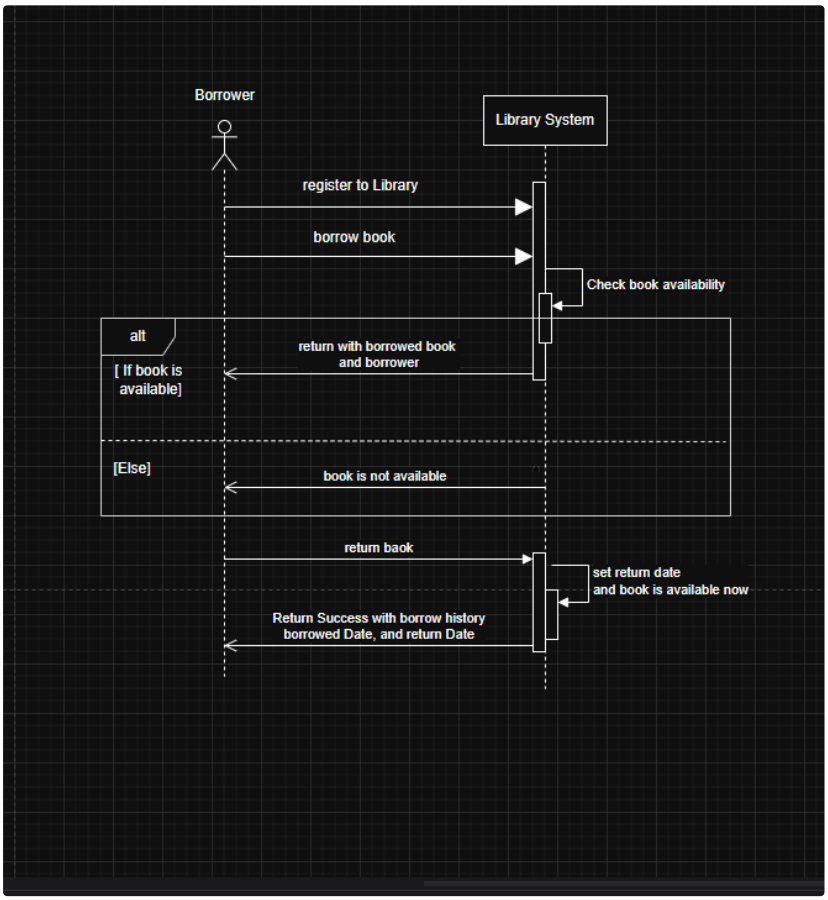
To Borrow a Book

1. **Registration**
   - A borrower must be registered in the library system before borrowing.
2. **Borrow Request**
   - The borrower requests to borrow a book.
   - The library system checks if the book is currently available (i.e., not borrowed by someone else).
3. **Availability Check**
   - **If available:**
     - The borrower is allowed to borrow the book.
     - A borrowing history record is created with the **borrow date**.
   - **If not available:**
     - The borrower is informed that the book is currently unavailable.

To Return a Book

1. **Return Request**
   - The borrower returns the borrowed book to the library.
2. **Update History**
   - The library updates the borrowing history record with the **return date**.

3. **Make Book Available**
   - The returned book is marked as **available** for future borrowing.



API Design

/api/library/books POST - To Add a new book to Library System

Request Body

```
{
  "isbn": "string",   // Must follow the standard ISBN format
  "title": "string",  // Cannot be empty
  "author": "string"  // Cannot be empty
}
```

Response Body - 201 Created

```
{
  "id": 1,
  "isbn": "string",
  "title": "string",
  "author": "string"
}
```

Error Response - 400 Bad Request or 500 Internal Server Error

```
{
  "message": "Invalid ISBN!",
  "httpStatus": "BAD_REQUEST",
```

```
  "time": "2025-08-22T17:11:53.652976800Z"
}
```

/api/library/books GET - To get the book paginated list

Request Parameter

```
?page=0&size=10
```

Response Body - 200 OK

```
{
  "totalElements": 5,
  "books": [
    {
      "id": 1,
      "isbn": "9799707827625",
      "title": "React Basic",
      "author": "Darwin"
    },
    {
      "id": 2,
      "isbn": "9799707827625",
      "title": "React Basic",
      "author": "Darwin"
    },
    ...
  ]
}
```

Error Response - 500 Internal Server Error

/api/library/borrowers POST - To register borrower to library system

Request Body

```
{
  "name": "string",
  "email": "user@example.com"
}
```

Response Body - 201 Created

```
{
  "id": 1,
  "name": "string",
  "email": "user@example.com"
}
```

Error Response - 400 Bad Request or 500 Internal Server Error

```
{
  "message": "Name is required",
  "httpStatus": "BAD_REQUEST",
  "time": "2025-08-22T17:19:17.900307800Z"
}

```

/api/library/borrows POST - To borrow the book by bookId

Request body

```
{
  "bookId": 1,
  "borrowerId": 1
}
```

Response Body - 200 Ok

```
{
  "id": 8,
  "borrower": {
    "id": 1,
    "name": "User One",
    "email": "user1@example.com"
  },
  "book": {
    "id": 1,
    "isbn": "9799707827625",
    "title": "React Basic",
    "author": "Darwin"
  },
  "borrowDate": "2025-08-22T18:34:21.250363",
  "returnDate": null
}
```

Error Response - 400 Bad Request or 500 Internal Server Error

```
{
  "message": "This book already borrowed!",
  "httpStatus": "BAD_REQUEST",
  "time": "2025-08-22T17:28:56.563443200Z"
}
```

/api/library/returns POST - To return the borrowed book to library

Request Body

```
{
  "bookId": 1,
  "borrowerId": 1
}
```

Response Body - 200 Ok

```
{
  "id": 8,
  "borrower": {
    "id": 1,
    "name": "User One",
    "email": "user1@example.com"
  },
  "book": {
    "id": 1,
    "isbn": "9799707827625",
```

```
    "title": "React Basic",
    "author": "Darwin"
  },
  "borrowDate": "2025-08-22T18:34:21.250363",
  "returnDate": "2025-08-23T00:20:57.5154199"
}
```

Error Response - 400 Bad Request or 500 Internal Server Error

```
{
  "message": "No active borrow record found!",
  "httpStatus": "BAD_REQUEST",
  "time": "2025-08-22T17:25:15.930385100Z"
}
```

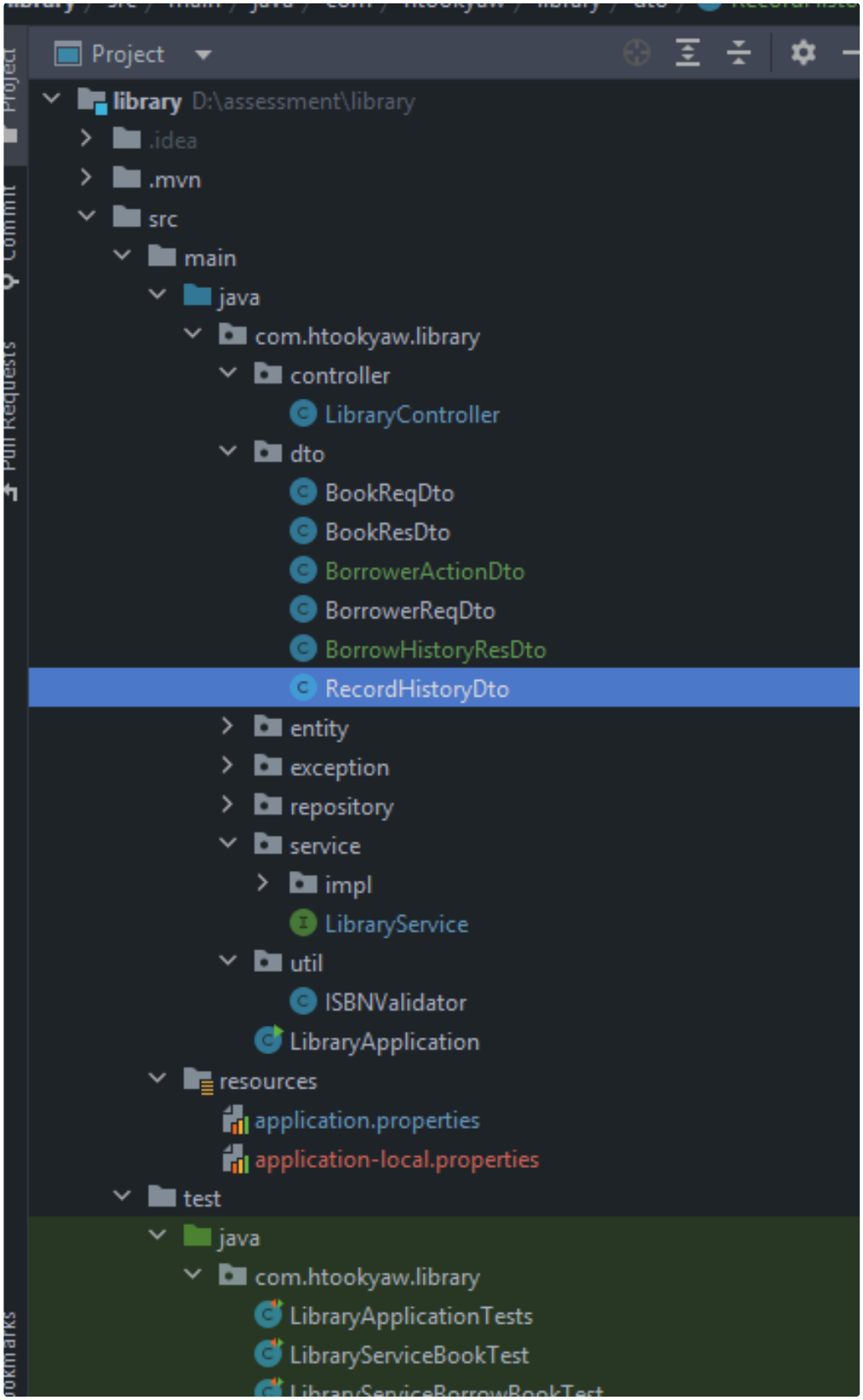
/api/library/borrower/history/{borrowerId} GET - To get borrowing history data of borrowerId

Path Variable - {borrowerId} as path variable

Response Body

```
{
  "borrower": {
    "id": 1,
    "name": "User One",
    "email": "user1@example.com"
  },
  "borrowHistory": [
    {
      "book": {
        "id": 1,
        "isbn": "9799707827625",
        "title": "React Basic",
        "author": "Darwin"
      },
      "borrowDate": "2025-08-22T18:34:21.250363",
      "returnDate": "2025-08-23T00:20:57.51542"
    },
    {
      "book": {
        "id": 1,
        "isbn": "9799707827625",
        "title": "React Basic",
        "author": "Darwin"
      },
      "borrowDate": "2025-08-23T00:28:32.853731",
      "returnDate": null
    }
  ]
}
```

Code Structure



The project follows a standard layered architecture to ensure a clean separation of concerns. Each package has a distinct responsibility:

- **controller:** Handles incoming HTTP requests and API endpoints.
- **dto (Data Transfer Object):** Defines the shape of data sent to and from the API.
- **entity:** Represents the database table structures as Java objects.
- **exception:** Contains custom exception classes for handling application-specific errors.
- **repository:** Manages all database operations and queries using Spring Data JPA.
- **service:** Implements the core business logic of the application.
- **util:** Provides utility classes and helper functions.

Setting Up In Local

Prerequisites

Before running the **Library Management System**, ensure you have the following installed on your system:

- ✅ Git CLI
- ✅ **Docker**

Clone the Repository

```
git clone https://github.com/htookyaw223/library-mgmt-system.git
cd library-mgmt-system
```

Database Configuration & Environment Setup

src/main/resources/application.properties

```
spring.application.name=library
# Pick active profile (overridden by ENV variable in Docker)
spring.profiles.active=${SPRING_PROFILES_ACTIVE:local}
#jpa database configuration
spring.datasource.url=${SPRING_DATASOURCE_URL}
spring.datasource.username=${SPRING_DATASOURCE_USERNAME}
spring.datasource.password=${SPRING_DATASOURCE_PASSWORD}

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
logging.level.org.springframework=INFO
logging.level.your.package.name=DEBUG
spring.main.allow-bean-definition-overriding=true
```

src/main/resources/application-local.properties

ℹ This local properties file is to run by Eclipse or Intellij or Executable Jar file

```
spring.application.name=library
server.port=8089

#jpa database configuration
spring.datasource.url=jdbc:mysql://localhost:3306/library_management_db
spring.datasource.password=root
spring.datasource.username=root

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
logging.level.org.springframework=INFO
logging.level.your.package.name=DEBUG
spring.main.allow-bean-definition-overriding=true
```

Env files

These .env file will be used when we run the application by docker.

.env.dev

This is for Dev environment

```
SPRING_PROFILES_ACTIVE=dev
SPRING_DATASOURCE_URL=jdbc:mysql://db:3306/dev_db?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
SPRING_DATASOURCE_USERNAME=dev_user
SPRING_DATASOURCE_PASSWORD=dev_password
MYSQL_DATABASE=dev_db
MYSQL_USER=dev_user
MYSQL_PASSWORD=dev_password
MYSQL_ROOT_PASSWORD=rootpass

```

.env.test

This is for Test environment

```
SPRING_PROFILES_ACTIVE=dev
SPRING_DATASOURCE_URL=jdbc:mysql://db:3306/dev_db?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
SPRING_DATASOURCE_USERNAME=dev_user
SPRING_DATASOURCE_PASSWORD=dev_password
MYSQL_DATABASE=dev_db
MYSQL_USER=dev_user
MYSQL_PASSWORD=dev_password
MYSQL_ROOT_PASSWORD=rootpass
```

.env.prod

This is for Production environment

```
SPRING_PROFILES_ACTIVE=dev
```

```
SPRING_DATASOURCE_URL=jdbc:mysql://db:3306/dev_db?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
SPRING_DATASOURCE_USERNAME=dev_user
SPRING_DATASOURCE_PASSWORD=dev_password
MYSQL_DATABASE=dev_db
MYSQL_USER=dev_user
MYSQL_PASSWORD=dev_password
MYSQL_ROOT_PASSWORD=rootpass
```

Build & Run the Application

Build & Run the Application By Docker Container

Dockerfile

```
# Build stage
FROM eclipse-temurin:17-jdk AS build
WORKDIR /app

# Copy Maven wrapper and prepare dependencies
COPY mvnw ./
COPY .mvn .mvn
COPY pom.xml ./
RUN ./mvnw dependency:go-offline

# Copy source and build the jar
COPY src ./src
RUN ./mvnw package -DskipTests

# Runtime stage
FROM eclipse-temurin:17-jdk
WORKDIR /app
COPY --from=build /app/target/library-management-system.jar library-management-system.jar

# Default run (can override with SPRING_PROFILES_ACTIVE)
ENTRYPOINT ["java", "-jar", "library-management-system.jar"]

```

docker-compose.yml

```
version: "3.8"

services:
  app:
    image: library-management-app:1.0.0
    container_name: ${APP_CONTAINER_NAME:-libraryContainer}
    ports:
      - "${APP_PORT:-8080}:8080"
    environment:
      SPRING_PROFILES_ACTIVE: ${SPRING_PROFILES_ACTIVE}
      SPRING_DATASOURCE_URL: ${SPRING_DATASOURCE_URL}
      SPRING_DATASOURCE_USERNAME: ${SPRING_DATASOURCE_USERNAME}
      SPRING_DATASOURCE_PASSWORD: ${SPRING_DATASOURCE_PASSWORD}
    depends_on:
      db:
        condition: service_healthy

  db:
    image: mysql:8.0
```

```yaml
    container_name: ${DB_CONTAINER_NAME:-mysql}
    restart: always
    environment:
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
    ports:
      - "${MYSQL_HOST_PORT:-3307}:3306"  # Map container 3306 to host 3307 by default
    volumes:
      - mysql_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "127.0.0.1", "-u", "${MYSQL_USER}", "-p${MYSQL_PASSWORD}"]
      interval: 10s
      timeout: 5s
      retries: 10

volumes:
  mysql_data:

```

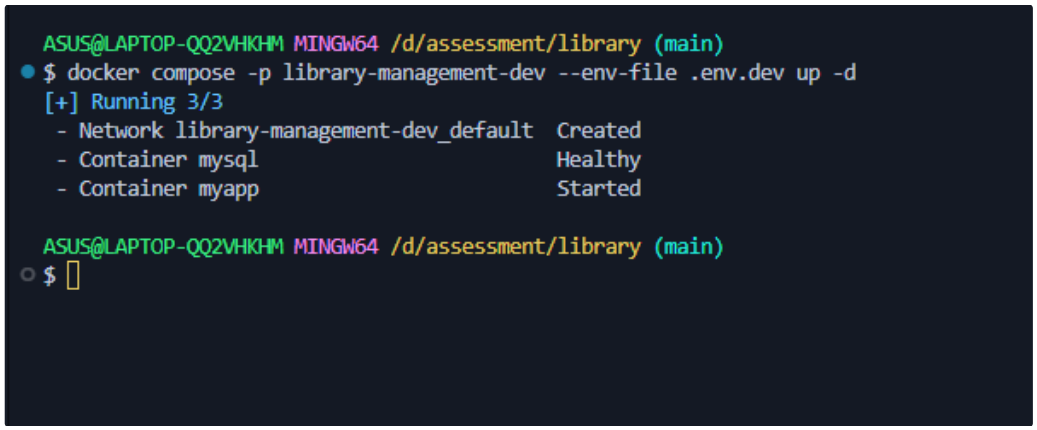
Open a terminal, navigate to the project root directory, and run the following command to build the Docker image:

```
docker build -t library-management-app:1.0.0 .
```

To run the application, use **Docker Compose** with the appropriate environment file. For example:

> You can dynamically specify the environment file (`.env.dev` , `.env.test` , `.env.prod` , etc.) depending on which environment you want to run.

```
docker compose -p library-management-dev --env-file .env.dev up -d
```



Build & Run the Application By Executable Jar

Prerequisites

Before running the **Library Management System**, ensure you have the following installed on your system:

- ✅**Java 17** (JDK)
- ✅ **MySQL** (database)
- ✅ **IntelliJ IDEA** or **Eclipse** (IDE for development)

1. Generate Executable JAR

Open a terminal, navigate to the project root directory, and run:

Windows (PowerShell / CMD):

```
mvnw package
```

Linux / macOS:

```
./mvnw package
```

This will build the project and generate a JAR file under:

```
/target/library-management-system.jar
```

---

2. Run the Application

Navigate to the `target` directory and start the application:

```
java -jar library-management-system.jar
```

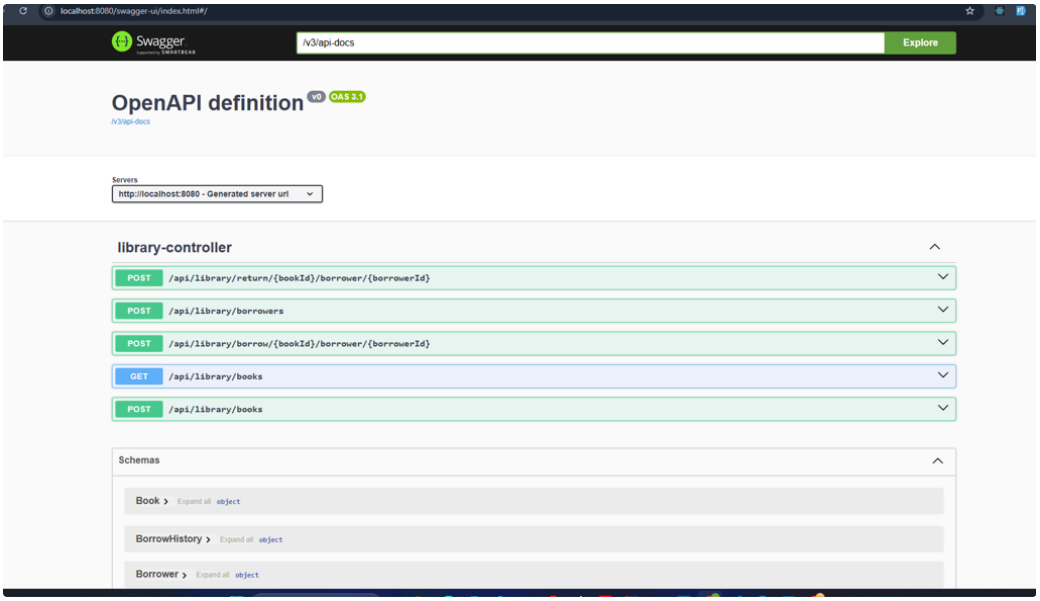
Access API Documentation (Swagger UI)

Once the application is running successfully, open your browser and go to:

```
http://localhost:8080/swagger-ui/index.html#/
```

Here you can:

- Explore the API documentation interactively
- Execute and test API requests directly from Swagger UI


localhost:8080/swagger-ui/index.html#/
Swagger
/v3/api-docs
Explore
OpenAPI definition
v0
OAS 3.1
/v3/api-docs
Servers
http://localhost:8080 - Generated server url
library-controller
POST /api/library/return/{bookId}/borrower/{borrowerId}
POST /api/library/borrowers
POST /api/library/borrow/{bookId}/borrower/{borrowerId}
GET /api/library/books
POST /api/library/books
Schemas
Book
Expand all
object
BorrowHistory
Expand all
object
Borrower
Expand all
object