Spring Boot CRUD Operations : Library Management System

Name: Pawar Rohit S.

Roll.no.: 115

1. Introduction

This documentation provides a comprehensive guide to building a Library Management System using Spring Boot. The system performs CRUD (Create, Read, Update, Delete) operations on library resources such as books and members using RESTful APIs and stores data in an H2 or MySQL database. This project showcases the use of Spring Boot, Spring Data JPA, and REST controllers to create a modular and scalable backend service.

2. Objectives

- To understand and implement CRUD operations using Spring Boot.
- To build REST APIs for managing library entities like books and users.
- To connect with databases using Spring Data JPA.
- To manage dependencies using Maven.
- To implement a layered architecture including Controller, Service, and Repository layers.

3. Technologies Used

- Spring Boot
- Spring Data JPA
- Spring Web
- Maven
- Java 11 or higher
- H2 or MySQL Database
- IDE: IntelliJ IDEA / Eclipse

4. Project Structure

```
LibraryManagementSystem/
├── src/
│   ├── main/
│   │   ├── java/com/example/library/
│   │   │   ├── controller/
│   │   │   ├── model/
│   │   │   ├── repository/
│   │   │   └── service/
│   └── resources/
│       └── application.properties
├── pom.xml
```

5. Entity Class (Book.java)

```java
package com.example.library.model;

import jakarta.persistence.*;

@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String author;
    private String isbn;
    private int quantity;

    // Getters and Setters
}
```

6. Repository Interface (BookRepository.java)

```java
package com.example.library.repository;

import com.example.library.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
}
```

7. Service Class (BookService.java)

```java
package com.example.library.service;

import com.example.library.model.Book;
import com.example.library.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class BookService {
    @Autowired
    private BookRepository repository;

    public Book save(Book book) {
        return repository.save(book);
    }

    public List<Book> getAll() {
        return repository.findAll();
    }

    public Optional<Book> getById(Long id) {
        return repository.findById(id);
    }

    public void delete(Long id) {
        repository.deleteById(id);
    }
}
```

8. Controller Class (BookController.java)

```
package com.example.library.controller;

import com.example.library.model.Book;
import com.example.library.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

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

    @Autowired
    private BookService service;

    @PostMapping
    public Book create(@RequestBody Book book) {
        return service.save(book);
    }

    @GetMapping
```

```java
    public List<Book> getAll() {
        return service.getAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Book> getById(@PathVariable Long id) {
        return service.getById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PutMapping("/{id}")
    public ResponseEntity<Book> update(@PathVariable Long id, @RequestBody Book book) {
        return service.getById(id).map(existing -> {
            existing.setTitle(book.getTitle());
            existing.setAuthor(book.getAuthor());
            existing.setIsbn(book.getIsbn());
            existing.setQuantity(book.getQuantity());
            return ResponseEntity.ok(service.save(existing));
        }).orElse(ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable Long id) {
        service.delete(id);
        return ResponseEntity.noContent().build();
    }
}
```

9. application.properties

```
# For H2 database
spring.datasource.url=jdbc:h2:mem:librarydb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

10. Sample pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>LibraryManagementSystem</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

11. Testing the API

- POST /api/books – Add a new book
- GET /api/books – Get all books
- GET /api/books/{id} – Get book by ID
- PUT /api/books/{id} – Update book
- DELETE /api/books/{id} – Delete book

12. Conclusion

The Library Management System illustrates how to create a robust backend system using Spring Boot. It follows a layered architecture and clean coding principles, and serves as a starting point for full-stack applications in educational and institutional environments.

13. References

- https://spring.io/projects/spring-boot
- https://docs.spring.io/spring-boot/docs/current/reference/html/
- https://www.baeldung.com/spring-boot-crud-thymeleaf
- https://www.geeksforgeeks.org/spring-boot-crud-operations/

- https://www.javatpoint.com/spring-boot-crud-example