**7.Basic Spring Boot Application with Spring Data JPA**

**Aim:**To create the Basic Spring Boot Application with Spring Data JPA

**Description:**

This project demonstrates how to create a basic Spring Boot application integrated with Spring Data JPA. It provides a simple setup for building Java applications that interact with relational databases using JPA and Hibernate. The application includes:

* Spring Boot for rapid application development and simplified configuration.
* Spring Data JPA for database operations such as saving, updating, deleting, and retrieving records without writing boilerplate SQL queries.
* Entity classes to represent database tables.
* Repository interfaces for CRUD operations.
* REST controllers (optional) to expose endpoints for interacting with the database.

This serves as a foundational template for beginners to understand how to set up Spring Boot with JPA and can be extended for real-world applications.

**Program:**

**Student.java**

package com.example;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Student {

@Id

private int sno;

private String sname;

public Student() {}

public Student(int sno, String sname) {

super();

this.sno = sno;

this.sname = sname;

}

public int getSno() {return sno;}

public void setSno(int sno) {this.sno = sno;}

public String getSname() {return sname;}

public void setSname(String sname) {this.sname = sname;}

}

**StudentController.java**

package com.example;

import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.RequestMapping;

@RestController

//@RequestMapping("/students")

public class StudentController {

private final StudentRepository repo;

public StudentController(StudentRepository repo) {

this.repo = repo;

}

//@RequestMapping("/students")

//Add new student

@PostMapping

public Student addStudent(@RequestBody Student student) {

return repo.save(student);

}

//Get all students

@GetMapping

public List<Student> getAllStudents(){

return repo.findAll();

}

}

**StudentRepository.java**

package com.example;

import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student,Integer> {

}

**StudentU3Application.java**

package com.example;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

@SpringBootApplication

public class StudentU3Application {

public static void main(String[] args) {

SpringApplication.run(StudentU3Application.class, args);

}

@Bean

CommandLineRunner initDatabase(StudentRepository repo) {

return args ->{

repo.save(new Student(1,"Gayatri"));

repo.save(new Student(2,"Swarupa"));

repo.save(new Student(3,"Sridevi"));

System.out.println("Students inserted!");

};

}

}

**application.properties**

spring.application.name=StudentU3

server.port=0000

spring.datasource.url=jdbc:mysql://localhost:3306/mca

spring.datasource.username=root

spring.datasource.password=Gayatri

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com</groupId>

<artifactId>StudentU3Application</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>StudentU3</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jdbc</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

<version>3.5.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

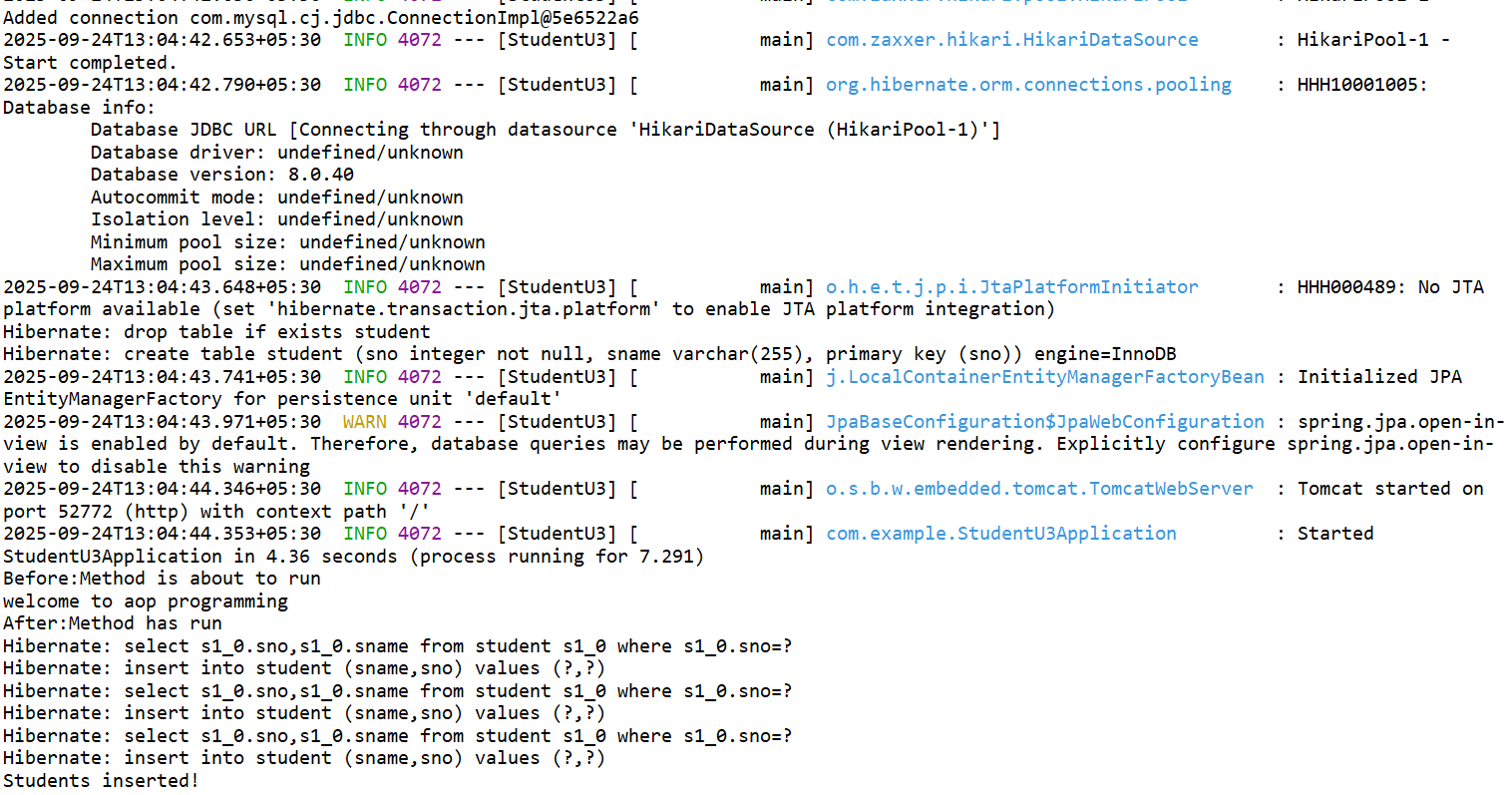
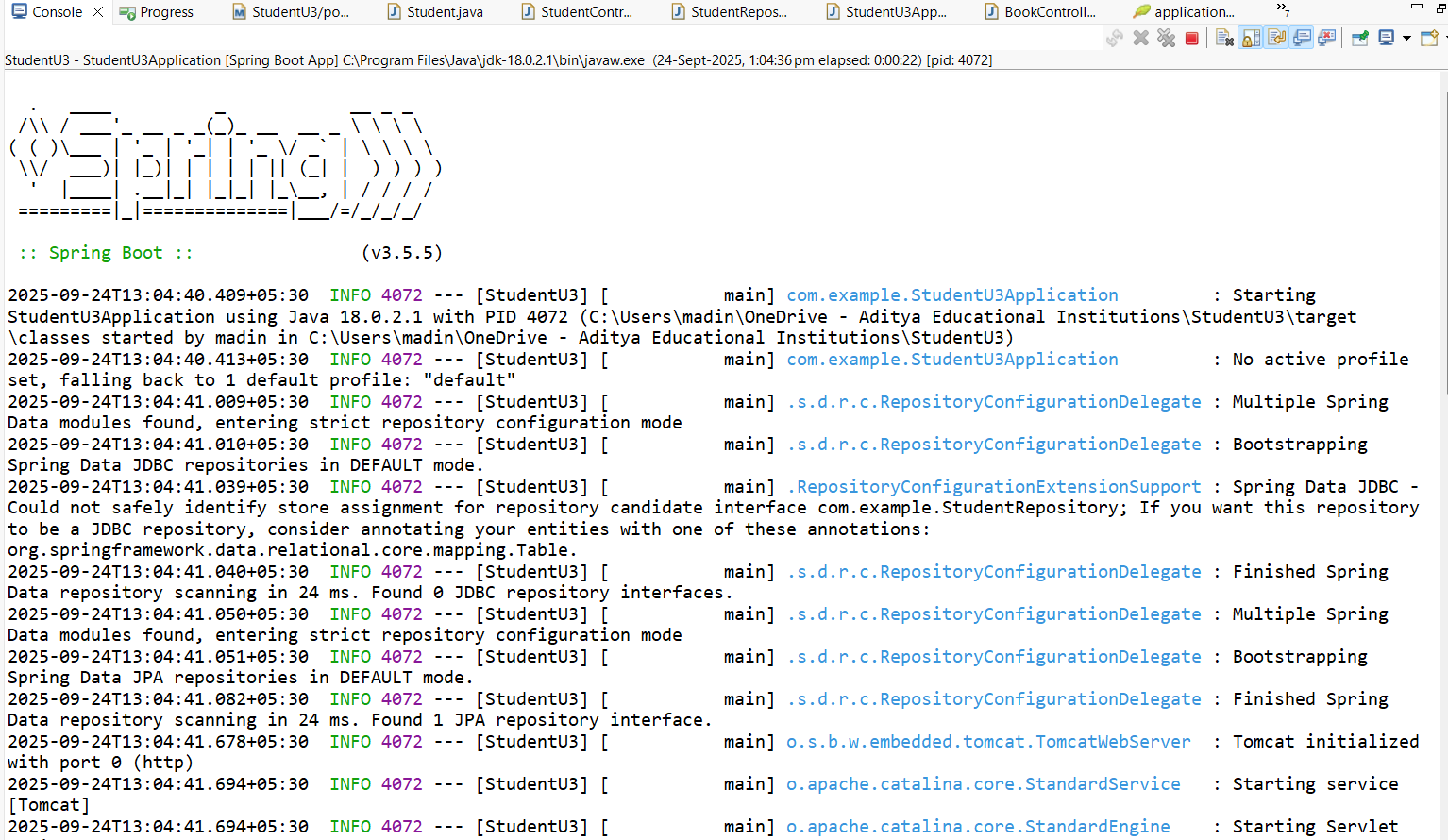
</plugin>

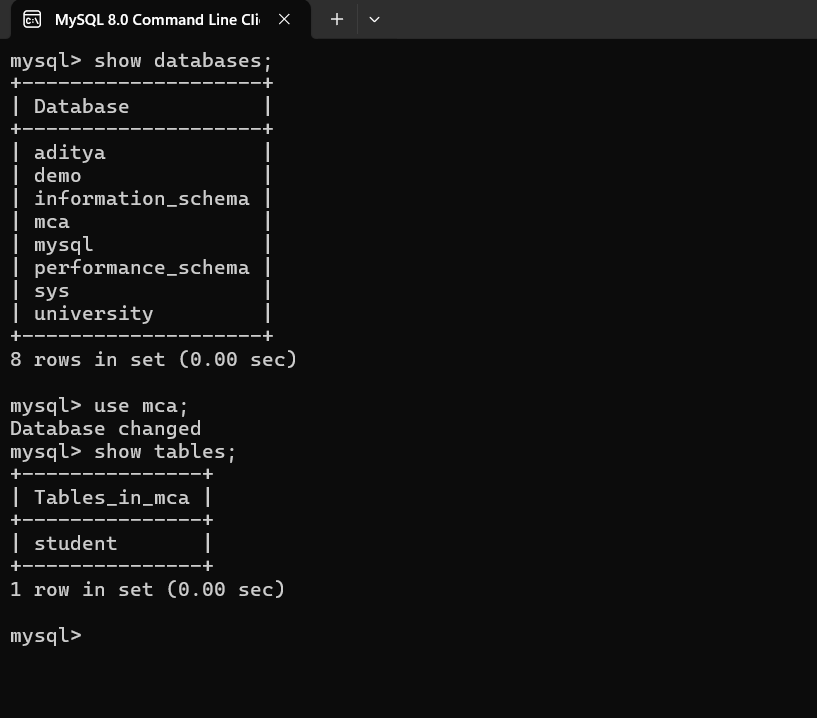
</plugins>

</build>

</project>

**Output:**





1. **Pagination and Sorting in Spring Data JPA**

**Aim:**To create the Pagination and Sorting in Spring Data JPA

**Description:**

Pagination and sorting are essential features when dealing with large sets of data in a database. Instead of retrieving all records at once, which can affect performance, Spring Data JPA provides built-in support to fetch data in chunks (pages) and in a specific order.

**Pagination:** It allows dividing the result set into smaller parts (pages). This helps reduce memory usage and speeds up response times. Spring Data JPA provides the Pageable interface and the PageRequest implementation to define the page number, size (number of records per page), and sorting details. The result is returned as a Page<T> or Slice<T> object.

**Sorting:** Sorting enables arranging the records in ascending or descending order based on one or more columns. Spring Data JPA supports sorting via the Sort class, which can be passed along with the Pageable object or used independently.

**Program:**

**Book.java**

package com.example;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private long id;

private String title;

private String author;

public Book() {}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

@Override

public String toString() {

return "Book [id=" + id + ", title=" + title + ", author=" + author + "]";

}

// getters and setters

public long getId() {return id;}

public void setId(long id) {this.id = id;}

public String getTitle() {return title;}

public void setTitle(String title) {this.title = title;}

public String getAuthor() {return author;}

public void setAuthor(String author) {this.author = author;}

}

**BookController.java**

package com.example;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/books")

public class BookController{

@Autowired

private BookRepository bookRepository;

//Insert sample data if empty

@GetMapping("/init")

public String initData() {

if(bookRepository.count() == 0) {

bookRepository.save(new Book("Spring Boot Basics","john"));

bookRepository.save(new Book("Java Programming","Alice"));

bookRepository.save(new Book("Hibernate in Action","Bob"));

bookRepository.save(new Book("MicroServices Guide","Carol"));

bookRepository.save(new Book("Data Structures","Davidraj"));

}

return "Sample books added!";

}

//Pagination + Sorting end point

@GetMapping

public Page<Book> getBooks(

@RequestParam(defaultValue = "0") int page,

@RequestParam(defaultValue = "3") int size,

@RequestParam(defaultValue = "title") String sortBy,

@RequestParam(defaultValue = "asc") String direction

){

Sort sort = direction.equalsIgnoreCase("asc") ?

Sort.by(sortBy).ascending():

Sort.by(sortBy).descending();

Pageable pageable = PageRequest.of(page, size, sort);

return bookRepository.findAll(pageable);

}

}

**BookRepository.java**

package com.example;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long>{

}

**U3BookApplication.java**

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class U3BookApplication {

public static void main(String[] args) {

SpringApplication.run(U3BookApplication.class, args);

}

}

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.6</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com</groupId>

<artifactId>U3StudentApplication</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>U3Book</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jdbc</artifactId>

</dependency>

<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.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

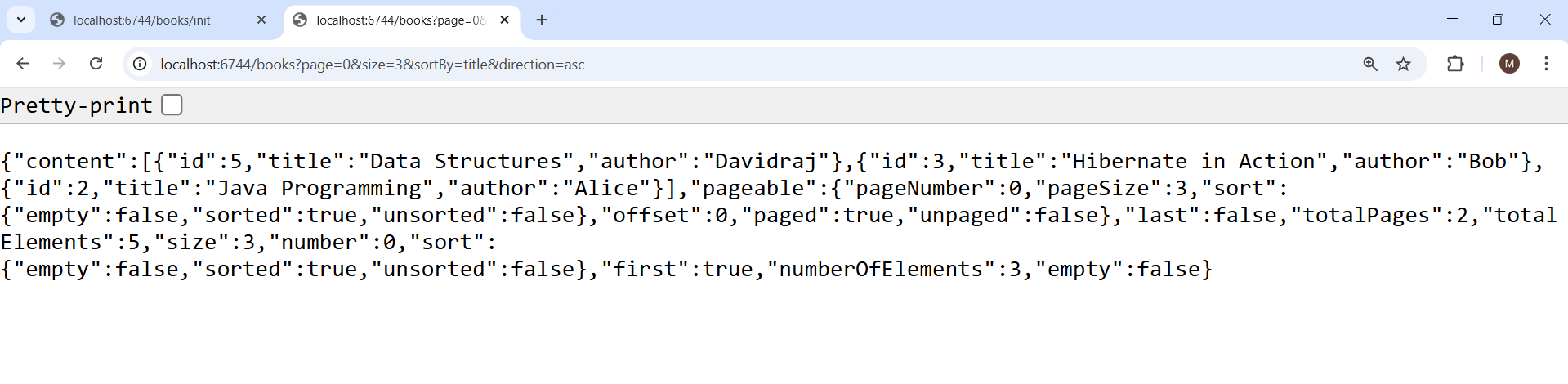
</plugins>

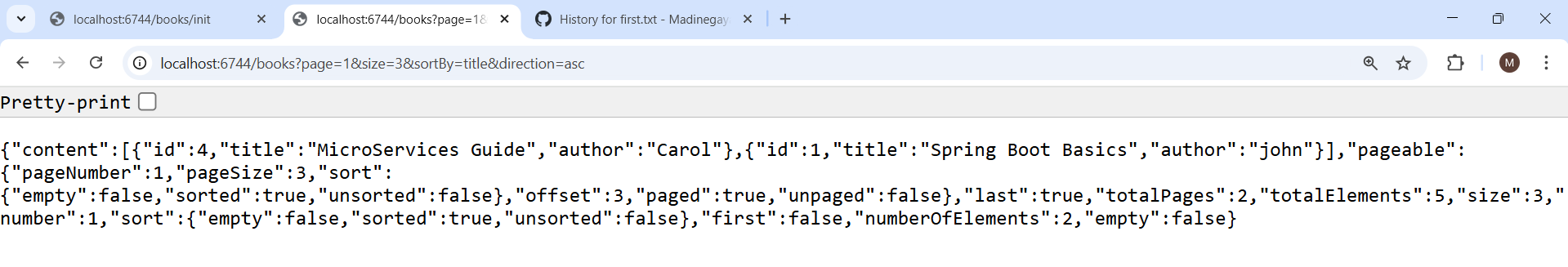
</build>

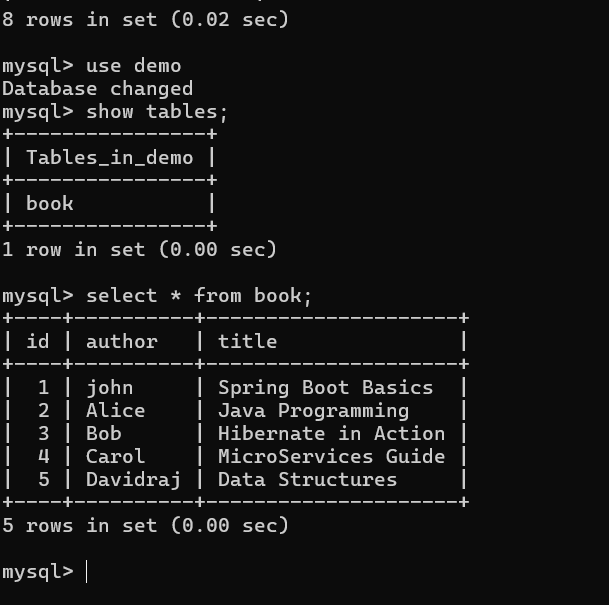
</project>

**Output:**

****

****

****

****

1. **Implementing AOP for Logging with Spring Data JPA**

**AIM:**To Implementing AOP for Logging with Spring Data JPA

**Description:**

Aspect-Oriented Programming (AOP) is a programming paradigm that allows separation of cross-cutting concerns, such as logging, from the core business logic of an application. In Spring applications, logging is an essential feature used to track system behavior, monitor method executions, capture inputs/outputs, and handle errors. Instead of embedding logging statements inside every repository or service method, AOP provides a clean and centralized way to implement logging.

When combined with Spring Data JPA, AOP can intercept calls to repository and service methods to automatically log important events. This includes logging method signatures, parameters, return values, and exceptions. By doing so, developers can achieve greater maintainability, reduce repetitive code, and improve debugging and monitoring.

Using AOP, we define aspects that contain advices such as before, after returning, and after throwing, which are applied to pointcuts that target specific repository or service methods. This way, all database interactions and business operations executed through Spring Data JPA can be consistently logged without altering the actual application logic.

In summary, implementing AOP for logging with Spring Data JPA ensures a clean separation of concerns, enhances transparency of system operations, simplifies debugging, and provides a unified logging mechanism across the entire data access and business layer.

**Program:**

**Product.java**

package com.example;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private long id;

private String name;

private double price;

public Product() {}

public Product(String name, double price) {

this.name = name;

this.price = price;

}

public long getId() {return id;}

public void setId(long id) {this.id = id;}

public String getName() {return name;}

public void setName(String name) {this.name = name;}

public double getPrice() {return price;}

public void setPrice(double price) {this.price = price;

}

**ProductController.java**

package com.example;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/products")

public class ProductController {

private final ProductService service;

public ProductController(ProductService service) {

this.service = service;

}

@PostMapping("/add")

public Product addProduct(@RequestBody Product product) {

return service.saveProduct(product);

}

@GetMapping("/all")

public List<Product> getAllProducts() {

return service.getAllProducts(); }

}

**Output:**

