**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.

**application.properties🡪**

spring.application.name=LibraryManagement

# H2 Database Configuration

spring.datasource.url=jdbc:h2:mem:librarymanagement

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

# JPA Configuration

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

server.port=8185

1. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.

**Book.java 🡪**

package com.library.api.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table

public class Book {

// primary key

@Id

// auto generate the value of primary key

@GeneratedValue(strategy = GenerationType.AUTO)

// created member variables

private int bookid;

private String bookname;

private String authorname;

private double price;

private String isbn;

// default constructor initializing the default value to member variables

public Book() {

this.bookid = 0;

this.bookname = null;

this.authorname = null;

this.price = 0;

this.isbn = null;

}

public Book(String bookname, String authorname, double price, String isbn) {

super();

this.bookname = bookname;

this.authorname = authorname;

this.price = price;

this.isbn = isbn;

}

//generate getter and setter

public int getBookid() {

return bookid;

}

public void setBookid(int bookid) {

this.bookid = bookid;

}

public String getBookname() {

return bookname;

}

public void setBookname(String bookname) {

this.bookname = bookname;

}

public String getAuthorname() {

return authorname;

}

public void setAuthorname(String authorname) {

this.authorname = authorname;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public String getIsbn() {

return isbn;

}

public void setIsbn(String isbn) {

this.isbn = isbn;

}

}

**BookRepository.java 🡪**

package com.library.api.repository;

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

import com.library.api.entity.Book;

public interface BookRepository extends JpaRepository<Book, Integer>{

}

1. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.

**BookController.java🡪**

import java.util.List;

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

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

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

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

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

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

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

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

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

import com.library.api.entity.Book;

import com.library.api.repository.BookRepository;

@RestController

@RequestMapping("api/books")

public class BookController {

//add the dependency BookRepository interface

@Autowired

BookRepository bookRepository;

@GetMapping("/test")

public String test()

{

return "Welcome to Book API";

}

//creating a route map for adding the new book

@PostMapping("/add")

public Book addBook(@RequestBody Book book)

{

//calling the repository method save

//if id=0 then create new record

//if >0 update the record

return bookRepository.save(book);

}

//route mapping for edit

@PutMapping("/edit")

public Book updateBook(@RequestBody Book book)

{

//update the record with passing the object with id to update

//it will update the records

return bookRepository.save(book);

}

//this root mapping

@GetMapping("/")

public List<Book> getAllBooks()

{

//returns all the exiting records

List<Book> books=bookRepository.findAll();

return books;

}

//mapping to get the records by id

@GetMapping("/{id}")

public Book getBook(@PathVariable int id)

{

//returns only one object if found returns null if not found

Book book=bookRepository.findById(id).get();

return book;

}

//route mapping to delete the record by id

@DeleteMapping("/delete/{id}")

public String getBookDelete(@PathVariable int id)

{

Book book=bookRepository.findById(id).get();

if(book!=null)

{

bookRepository.deleteById(id);

return "Book with "+id+" is deleted successfully";

}

return "Book with "+id+" not found";

}

}

1. **Run the Application:**

Run the Spring Boot application and test the REST endpoints.

**LibraryManagementApplication.java 🡪**

package com.library.api;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}