**“Spring Core and Maven”**

**Exercise 1: Configuring a Basic Spring Application**

**Scenario:**

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**1.BookRepository.java**

package com.library.repository;

public class BookRepository {

public void printBooks() {

System.*out*.println("📚 BookRepository: List of books");

}

}

**2.BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

System.*out*.println("📖 BookService: Showing books using repository...");

bookRepository.printBooks();

}

}

**3.MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

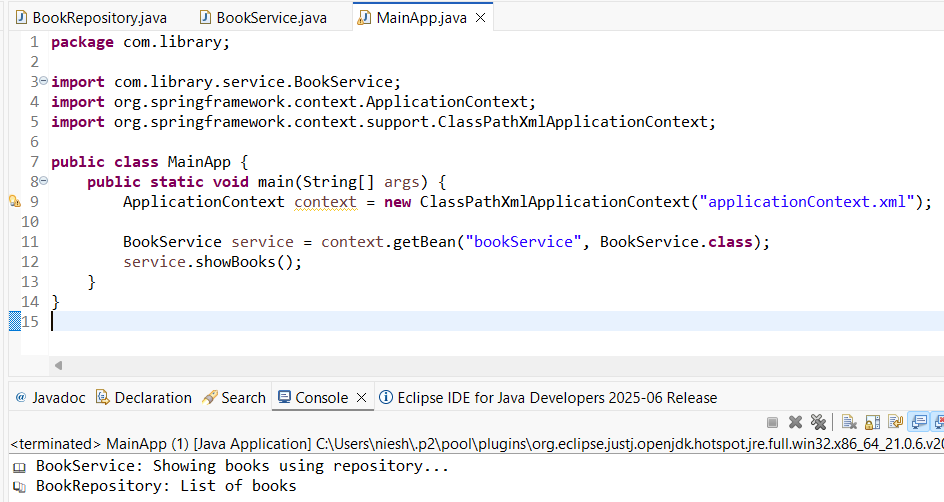
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean("bookService", BookService.class);

service.showBooks();

}

}



---------------------------------------------------------------------------------------------------------------------------

**Exercise 2: Implementing Dependency Injection**

**Scenario:**

In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**1.BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.*out*.println("Displaying books from the repository...");

}

}

**2.BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.*out*.println("BookRepository injected via setter");

}

public void listBooks() {

bookRepository.displayBooks();

}

}

**3.MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load the Spring IoC container

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

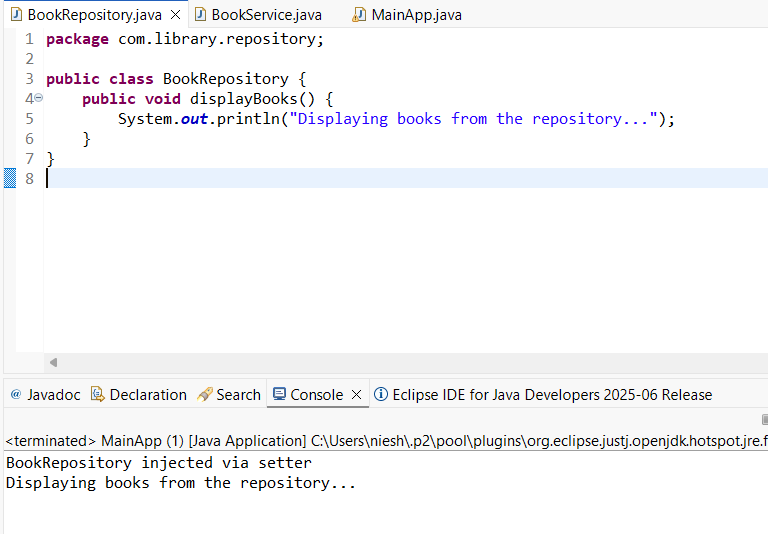
// Retrieve the bean and invoke method

BookService service = context.getBean("bookService", BookService.class);

service.listBooks();

}

}



---------------------------------------------------------------------------------------------------------------------------

**Exercise 3: Implementing Logging with Spring AOP**

**Scenario:**

The library management application requires logging capabilities to track method execution times.

**1.MainApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Spring automatically injects component-scanned beans

BookService bookService = context.getBean(BookService.class);

bookService.addBook();

bookService.deleteBook();

}

}

**2.BookService.java**

package com.library.service;

import org.springframework.stereotype.Component;

@Component

public class BookService {

public void addBook() {

System.*out*.println("Book added to the system.");

}

public void deleteBook() {

System.*out*.println("Book deleted from the system.");

}

}



---------------------------------------------------------------------------------------------------------------------------

**Exercise 4: Creating and Configuring a Maven Project**

**Scenario:**

You need to set up a new Maven project for the library management application and add Spring dependencies.

**1.MainApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

System.*out*.println("Library Management Application Started");

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

} }



---------------------------------------------------------------------------------------------------------------------------

**Exercise 5: Configuring the Spring IoC Container**

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**1.BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for dependency injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void listBooks() {

bookRepository.displayBooks();

}

}

**2.BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.*out*.println("Displaying list of books...");

}

}

**3.MainApp.java**  
package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load Spring configuration file

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Get BookService bean and call method

BookService service = context.getBean("bookService", BookService.class);

service.listBooks();

}

}



---------------------------------------------------------------------------------------------------------------------------

**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

You need to simplify the configuration of beans in the library management application using annotations.

**1.MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean(BookService.class);

service.listBooks();

}

}

**2.BookService.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean(BookService.class);

service.listBooks();

}

}

**3.BookRepository.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new

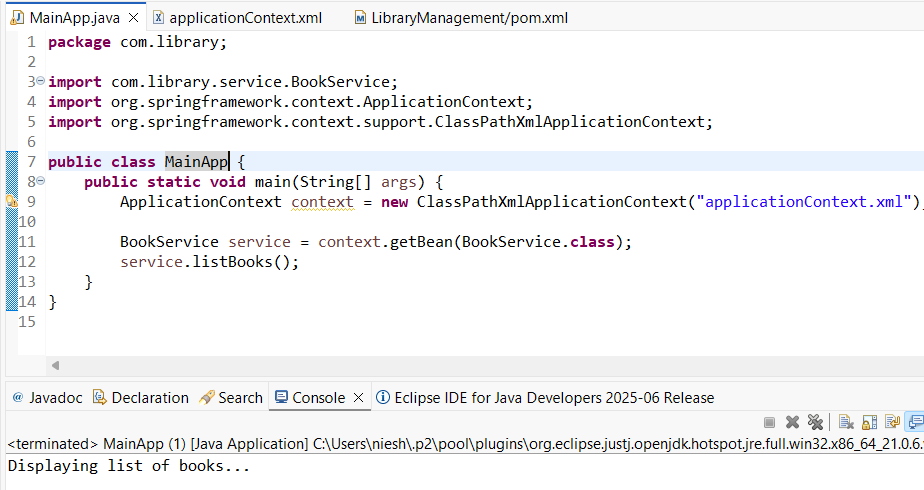
ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean(BookService.class);

service.listBooks();

}

}



---------------------------------------------------------------------------------------------------------------------------

**Exercise 7: Implementing Constructor and Setter Injection**

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**1.MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean("bookService", BookService.class);

service.listBooks();

}

}

**2.BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String serviceName;

// Constructor for constructor injection

public BookService(String serviceName) {

this.serviceName = serviceName;

System.*out*.println("Constructor Injection: serviceName = " + serviceName);

}

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.*out*.println("Setter Injection: BookRepository set");

}

public void listBooks() {

System.*out*.println("Service: " + serviceName);

bookRepository.displayBooks();

}

}

**3.BookRepository.java**

package com.library.repository;

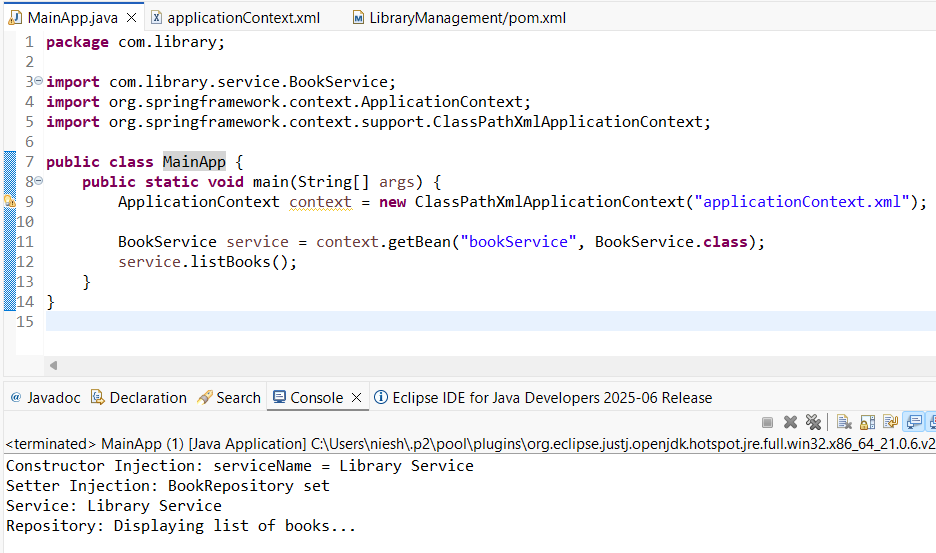
public class BookRepository {

public void displayBooks() {

System.*out*.println("Repository: Displaying list of books...");

}

}



---------------------------------------------------------------------------------------------------------------------------

**Exercise 8: Implementing Basic AOP with Spring**

**Scenario:**

The library management application requires basic AOP functionality to separate cross-cutting concerns like logging and transaction management.

**1,MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean("bookService", BookService.class);

service.listBooks();

}

}

**2.LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

@Aspect

public class LoggingAspect {

@Before("execution(\* com.library.service.\*.\*(..))")

public void logBefore(JoinPoint joinPoint) {

System.out.println("🔍 [Before] Executing method: " + joinPoint.getSignature().getName());

}

@After("execution(\* com.library.service.\*.\*(..))")

public void logAfter(JoinPoint joinPoint) {

System.out.println("✅ [After] Completed method: " + joinPoint.getSignature().getName());

}

}

**3.BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String serviceName;

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void listBooks() {

System.*out*.println("Service: " + "EXERCISE-8");

bookRepository.displayBooks();

}

}

**4.BookRepository.java**

package com.library.repository;

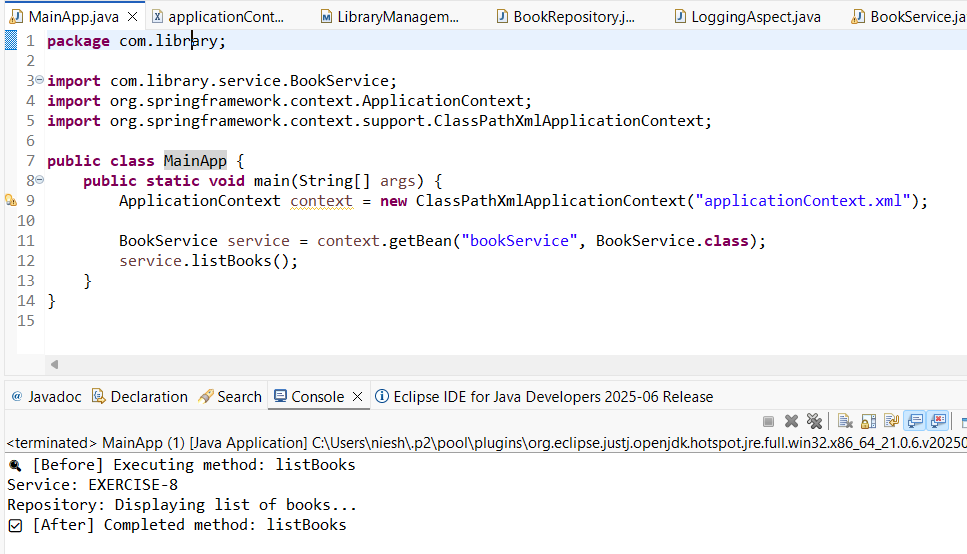
public class BookRepository {

public void displayBooks() {

System.*out*.println("Repository: Displaying list of books...");

}

}



---------------------------------------------------------------------------------------------------------------------------

**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.

**1.Book.java**

package com.library.entity;

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;

}

// Getters & setters

public Long getId() { return 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; }

}

**2.BookController.java**

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

Book existingBook = bookRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Book not found with ID: " + id));

existingBook.setTitle(bookDetails.getTitle());

existingBook.setAuthor(bookDetails.getAuthor());

return bookRepository.save(existingBook);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**3.BookRepository.java**

package com.library.repository;

import com.library.entity.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**4.LibraryManagementBootApplication.java**

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementBootApplication {

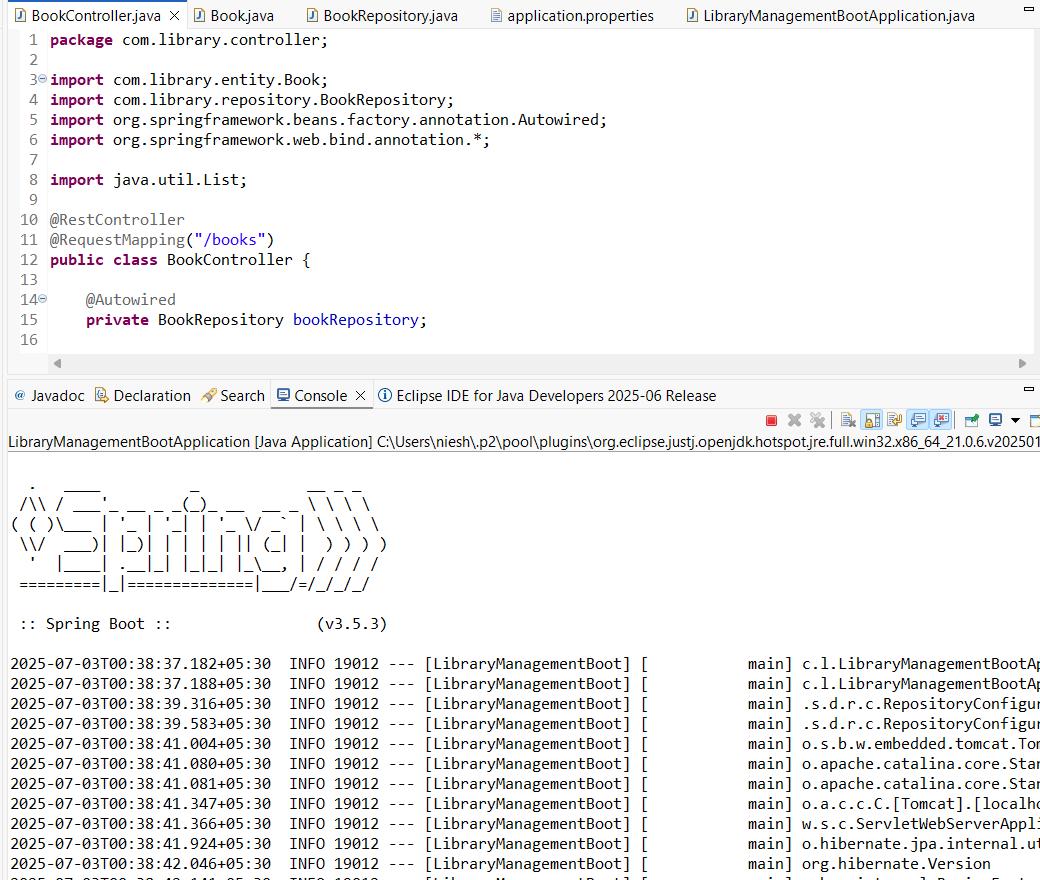
public static void main(String[] args) {

SpringApplication.*run*(LibraryManagementBootApplication.class, args);

}

}

**#OUPUT CONSOLE**



**5.application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

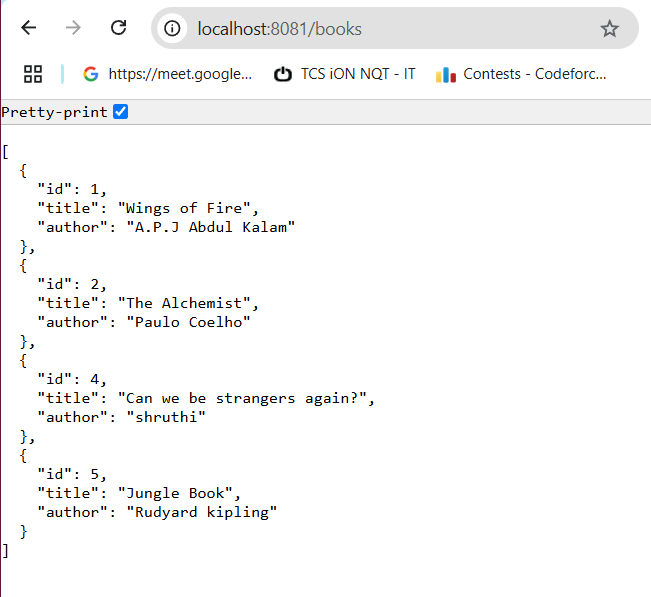
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

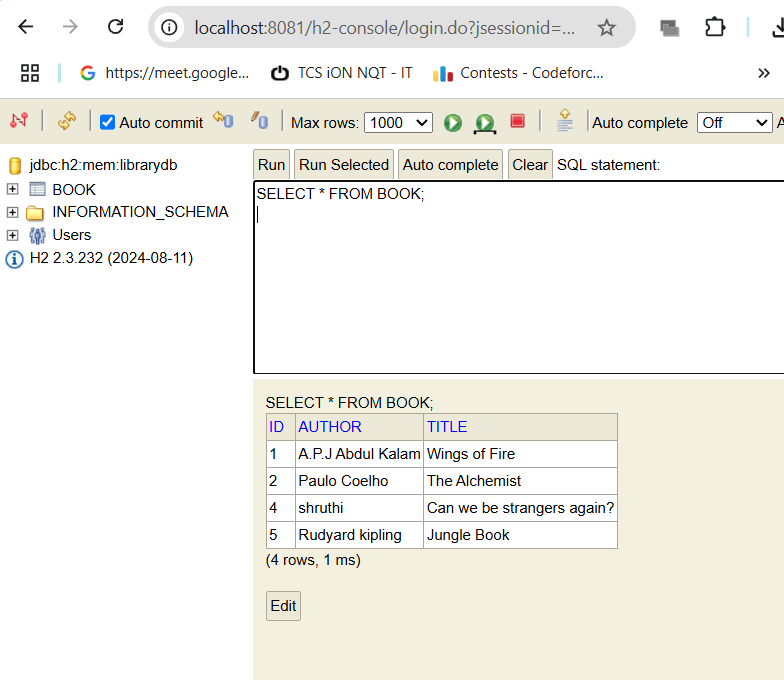
server.port=808

**#http://localhost:8081/books -> JSON**

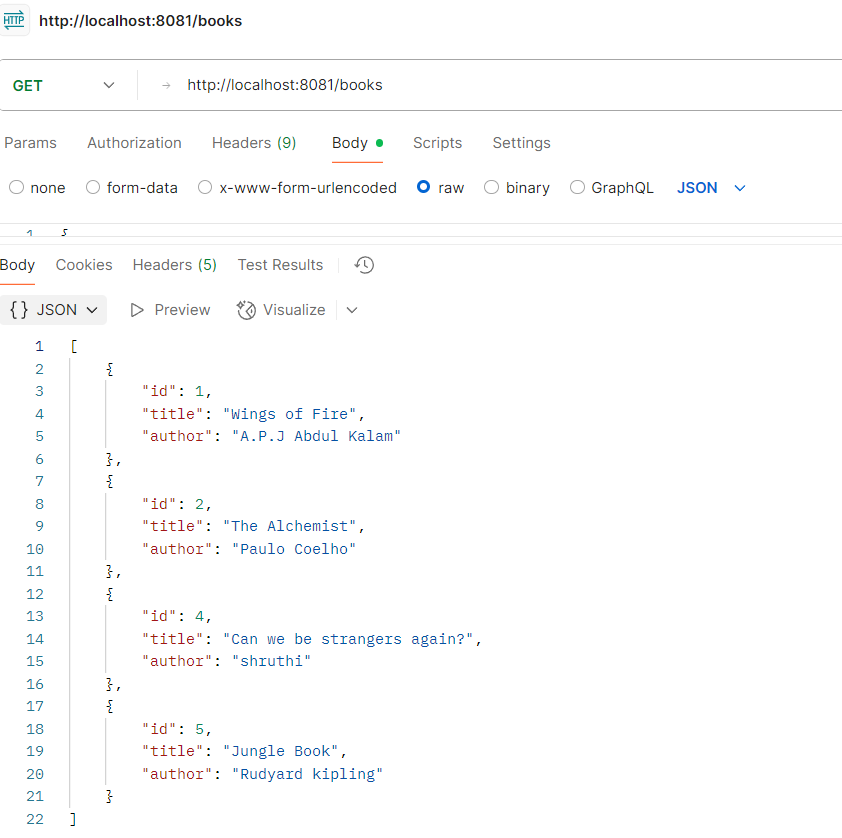


---------------------------------------------------------------------------------------------------------------------------

**#http://localhost:8081/h2-console -> SQL DB**

****

**#using postman commands- POST,GET,PUT,DELETE**

****

**---------------------------------------------------------------------------------------------------------------------------**