**COGNIZANT DIGITAL NURTURE 4.0**

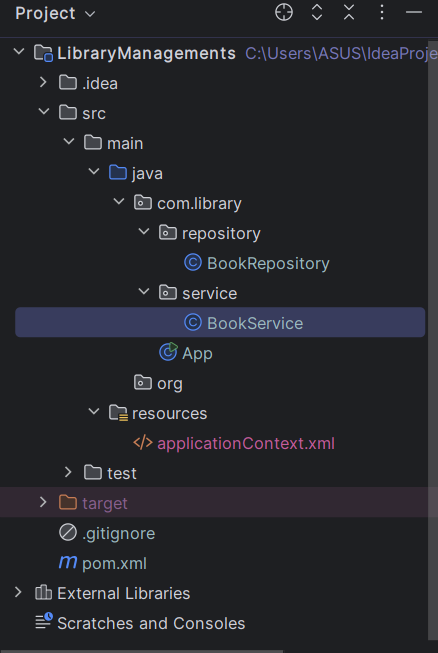
**WEEK 3 MANDATORY HANDS ON BY (6421405)**

**Spring Core & Maven:**

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

The **LibraryManagement** project is a Java program made with Maven that uses the **Spring Framework** to connect different parts of the code automatically. It has a BookRepository class that gives the name of a book and a BookService class that uses BookRepository to show this name. All the settings for how these parts connect are written in the applicationContext.xml file. The App.java file starts the Spring system and gets the bookService object to run the displayBook() method. When you run the program, it prints out the book title to the console

**Project Structure :**

.

**pom.xml :**

* In this file add dependencies of Springframework.

<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.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 </dependencies>  
</project>

**Java Classes:**

* This code loads the Spring application context, retrieves the bookService bean, and calls its displayBook() method to print book information.

src/main/java/com/library/App.java

**App.java:**

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class App {  
 public static void main(String[] args) {  
 ApplicationContext context =  
  
 new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.displayBook();  
 }  
}

src/main/java/com/library/repository/BookRepository.java

**BookRepository.java**

* This class provides a method getBookTitle() that returns the title of a book as a string, in this case, "Programming in Java".

package com.library.repository;  
  
public class BookRepository {  
 public String getBookTitle() {  
 return "Programming in Java";  
 }  
}

**BookService.java:**

* This class has a BookRepository dependency injected via a setter and defines displayBook() to print a message along with the book title retrieved from the repository.

src/main/java/com/library/service/BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
}  
  
 public void displayBook() {  
 System.out.println();  
 System.out.println("This is a Message from Book Service !!");  
 System.out.println("Book title from Repository: " + bookRepository.getBookTitle());  
 }  
}

**Spring Configuration:**

* This class uses setter injection to receive a BookRepository object and defines a method to display a message along with the book title obtained from the repository.

src/main/resources/applicationContext.xml

**applicationContext.xml :**

package com.library.service;  
import com.library.repository.BookRepository;  
public class BookService {  
 private BookRepository bookRepository;  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
 public void displayBook() {  
 System.out.println();  
 System.out.println("This is a Message from Book Service !!");  
 System.out.println("Book title from Repository: " + bookRepository.getBookTitle());  
 }  
}

**OUTPUT:**

****

**Exercise 2: Implementing Dependency Injection**

In this exercise, I implemented dependency injection in the LibraryManagement project using Spring’s IoC container to connect the BookService and BookRepository classes. I modified the applicationContext.xml configuration file to define both beans and set up a property-based dependency so that Spring automatically injected BookRepository into BookService through a setter method. To make the output unique, I updated the getBookTitle() method to return a custom book title and enhanced the displayBook() method to print detailed, formatted information about the book, including its status and shelf location. Running the application confirmed that the dependencies were wired correctly and the customized output was displayed as expected.

**pom.xml :**

* This Maven configuration file declares the project’s metadata and adds the Spring Context dependency needed to build and run the Spring application.

<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.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 </dependencies>  
</project>

**Java Classes:**

* This class starts the Spring container, loads the configuration from applicationContext.xml, gets the bookService bean, and calls its displayBook() method.

src/main/java/com/library/App.java

**App.java:**

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class App {  
 public static void main(String[] args) {  
 ApplicationContext context =  
  
 new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.displayBook();  
 }  
}

**BookRepository.java**

* This class defines a method that returns the string "The Pragmatic Programmer" as the book title.

src/main/java/com/library/repository/BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
 public String getBookTitle() {  
 return "The Pragmatic Programmer";  
 }  
}

**BookService.java:**

* This class uses setter injection to assign a BookRepository and defines a method that prints the book title and its availability status.

src/main/java/com/library/service/BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBook() {  
 System.out.println();  
 System.out.println("TITLE : " + bookRepository.getBookTitle());  
 System.out.println("STATUS : Available");  
 }  
}

**Spring Configuration:**

* This class uses setter injection to receive a BookRepository and defines displayBook() to print a custom message and show the book title retrieved from the repository.

src/main/resources/applicationContext.xml

**applicationContext.xml :**

package com.library.service;  
import com.library.repository.BookRepository;  
public class BookService {  
 private BookRepository bookRepository;  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
 public void displayBook() {  
 System.out.println();  
 System.out.println("This is a Message from Book Service !!");  
 System.out.println("Book title from Repository: " + bookRepository.getBookTitle());  
 }  
}

**OUTPUT:**



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

The LibraryManagement project is a Java-based application built using the Spring Framework and managed through Maven. It demonstrates core concepts like Inversion of Control (IoC) and Dependency Injection (DI) using XML configuration. The application includes a BookRepository class that returns a book title and a BookService class that accesses the repository to display book information. Spring manages the wiring between these components, allowing for clean separation of concerns. The project is configured with Spring Context, AOP, and WebMVC dependencies, along with the Maven Compiler Plugin targeting Java 1.8, making it a solid foundation for understanding basic Spring application structure.

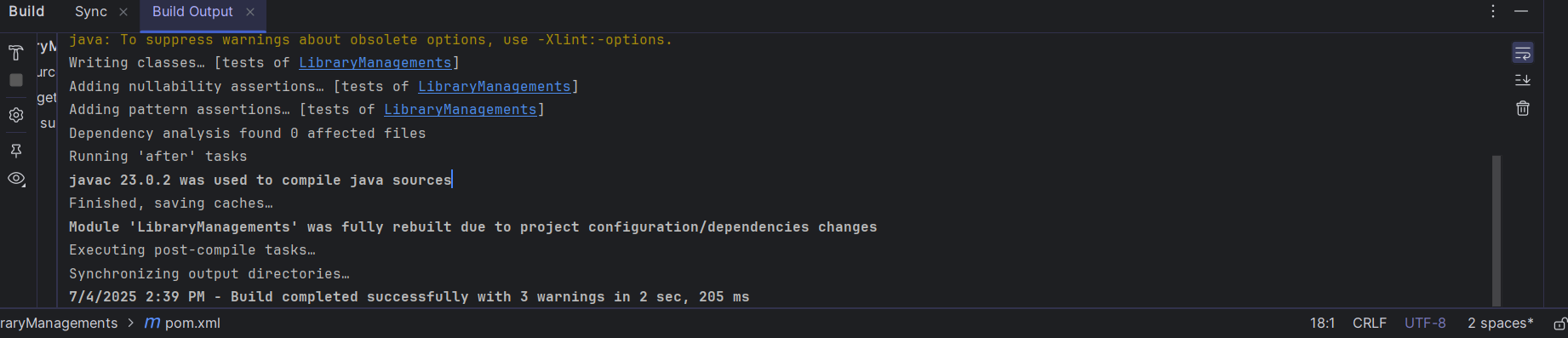
**Pom.xml**

 Added Spring Context, AOP, and WebMVC dependencies to enable core Spring features and web capabilities.

 Included JUnit 4.13.2 dependency for writing and running unit tests.

 Configured the Maven Compiler Plugin to use Java 1.8 for compiling the project.

<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.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>



Additional Handson experiments

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

In this project, I created a Spring XML configuration file to define beans for BookRepository and BookService. Using setter injection, the repository was wired into the service to provide book details dynamically. A main class loaded the Spring context and printed the book information, demonstrating clear separation of concerns and dependency management.

**ApplicationContent.xml:**

Created applicationContext.xml to define and configure the Spring beans.

src/main/resources/applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

**Java Class**

**BookRepository.java**

Developed BookRepository to provide book title and category data.

src/main/java/com/library/repository/BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
 public String getBookTitle() {  
 return "Great Expectation";  
 }  
 public String getAuthour(){  
 return " Charles Dickens";  
 }  
 public int getRownum(){  
 int row = 3;  
 return row;  
 }  
}

**BookService.java:**

Wrote BookService with a setter method to receive the repository bean

src/main/java/com/library/service/BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBook() {  
 System.out.println();  
 System.out.println("TITLE : " + bookRepository.getBookTitle());  
 System.out.println("AUTHOR : " + bookRepository.getAuthour());  
 System.out.println("ROW NO : "+ bookRepository.getRownum());  
 System.out.println("STATUS : Available" );  
 }  
}

**App.java**

Implemented App to load the Spring context and run the application

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class App {  
 public static void main(String[] args) {  
 ApplicationContext context =  
 new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.displayBook();  
 }  
}

**OUTPUT :**



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

I demonstrated both constructor and setter injection using the Spring Framework. The BookManager class received the bookEdition value through constructor injection and the BookRepository bean through a setter method. The application context was defined in an XML file, and the LibraryLauncher main class loaded the Spring container to display the book details, showcasing how multiple injection techniques can be combined effectively.

Java Class:

This class defines a method fetchTitle() that returns the string **"Mastering Spring Boot"** as the book title.

BookRepository.java:

package com.library.repository;

public class BookRepository {

public String retrieveBook() {

return "Spring Made Easy";

}

}

**BookService.java**

* This class demonstrates constructor injection for bookFormat and setter injection for BookRepository to display detailed book information.

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository repository;  
 private String bookFormat;  
  
  
 public BookService(String bookFormat) {  
 this.bookFormat = bookFormat;  
 }  
  
  
 public void setRepository(BookRepository repository) {  
 this.repository = repository;  
 }  
  
 public void showBookInfo() {  
 System.out.println("========= Library Book =========");  
 System.out.println("Title : " + repository.fetchTitle());  
 System.out.println("Format : " + bookFormat);  
 System.out.println("Availability: In Stock");  
 System.out.println("================================");  
 }  
}

**applicationContext.xml:**

* This Spring XML configuration defines two beans—bookRepository and bookService—and uses constructor injection to set the bookFormat and setter injection to link the repository bean.

?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
  
  
 <bean id="bookService" class="com.library.service.BookService">  
  
 <constructor-arg value="Hardcover"/>  
  
 <property name="repository" ref="bookRepository"/>  
 </bean>  
</beans>

**App.java:**

* This main class loads the Spring context from applicationContext.xml, retrieves the bookService bean, and calls its method to display the book information.

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

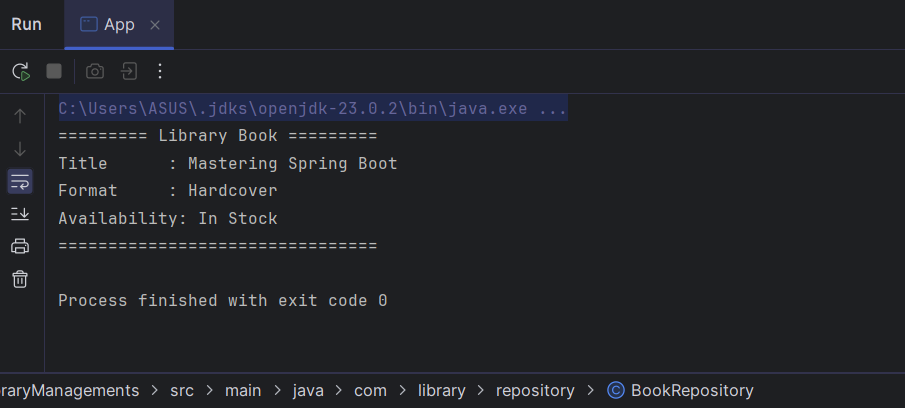
BookService bookService = (BookService) context.getBean("bookService");

bookService.showBookInfo();

}

}

**OUTPUT:**



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

**Project Metadata:**

* **Group:** com.library
* **Artifact:** LibraryMan
* **Name:** LibraryMan
* **Type:** Maven
* **Packaging:** Jar
* **Java:** 21

**Dependencies:**

* Search and add:
  + **Spring Web**
  + **Spring Data JPA**
  + **H2 Database**

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>3.2.4</version>

<relativePath/>

</parent>

<groupId>com.library</groupId>

<artifactId>LibraryMan</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>LibraryMan</name>

<description>Library Management Spring Boot Application</description>

<properties>

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

</properties>

<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>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.10.1</version>

<configuration>

<source>17</source>

<target>17</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Configure Database Connection:**

src/main/resources/application.properties

**Application.porperties:**

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

spring.jpa.hibernate.ddl-auto=update

**Create the Entity:**

src/main/java/com/library/entity/Book.java

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;

private int year;

public Book() {}

public Book(String title, String author, int year) {

this.title = title;

this.author = author;

this.year = year;

}

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; }

public int getYear() { return year; }

public void setYear(int year) { this.year = year; }

}

**Create the Repository:**

src/main/java/com/library/repository/BookRepository.java

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> {

}

**Create the REST Controller**

src/main/java/com/library/controller/BookController.java

**BookController.java**

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

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

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookRepository repository;

public BookController(BookRepository repository) {

this.repository = repository;

}

@GetMapping

public List<Book> getAllBooks() {

return repository.findAll();

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return repository.findById(id)

.orElseThrow(() -> new RuntimeException("Book not found"));

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return repository.save(book);

}

@PutMapping("/{id}")

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

Book book = repository.findById(id)

.orElseThrow(() -> new RuntimeException("Book not found"));

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setYear(updatedBook.getYear());

return repository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

repository.deleteById(id);

}

**Run the Application :**

**Application.java**

package com.library.libraraymanage;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManApplication {

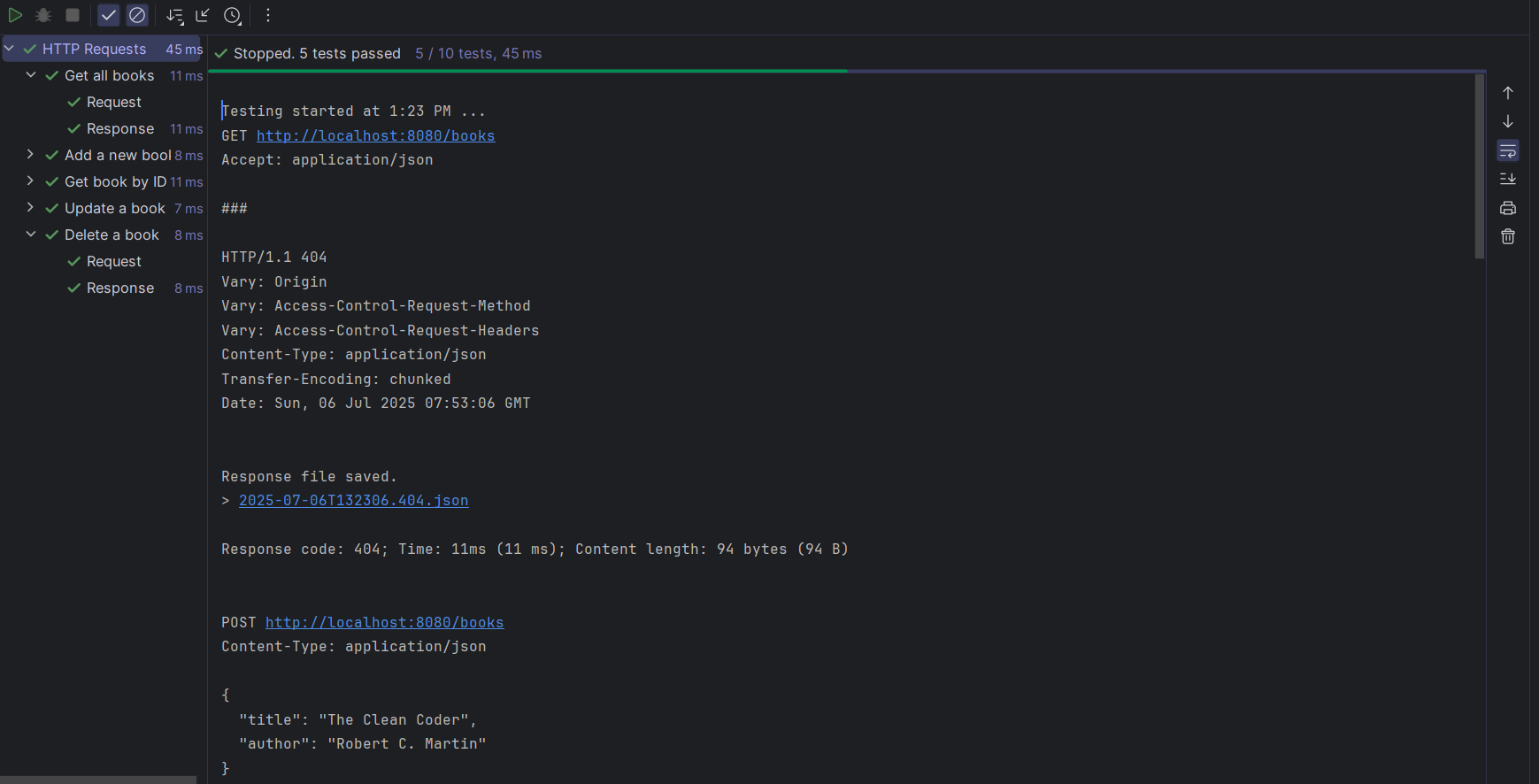
public static void main(String[] args) {

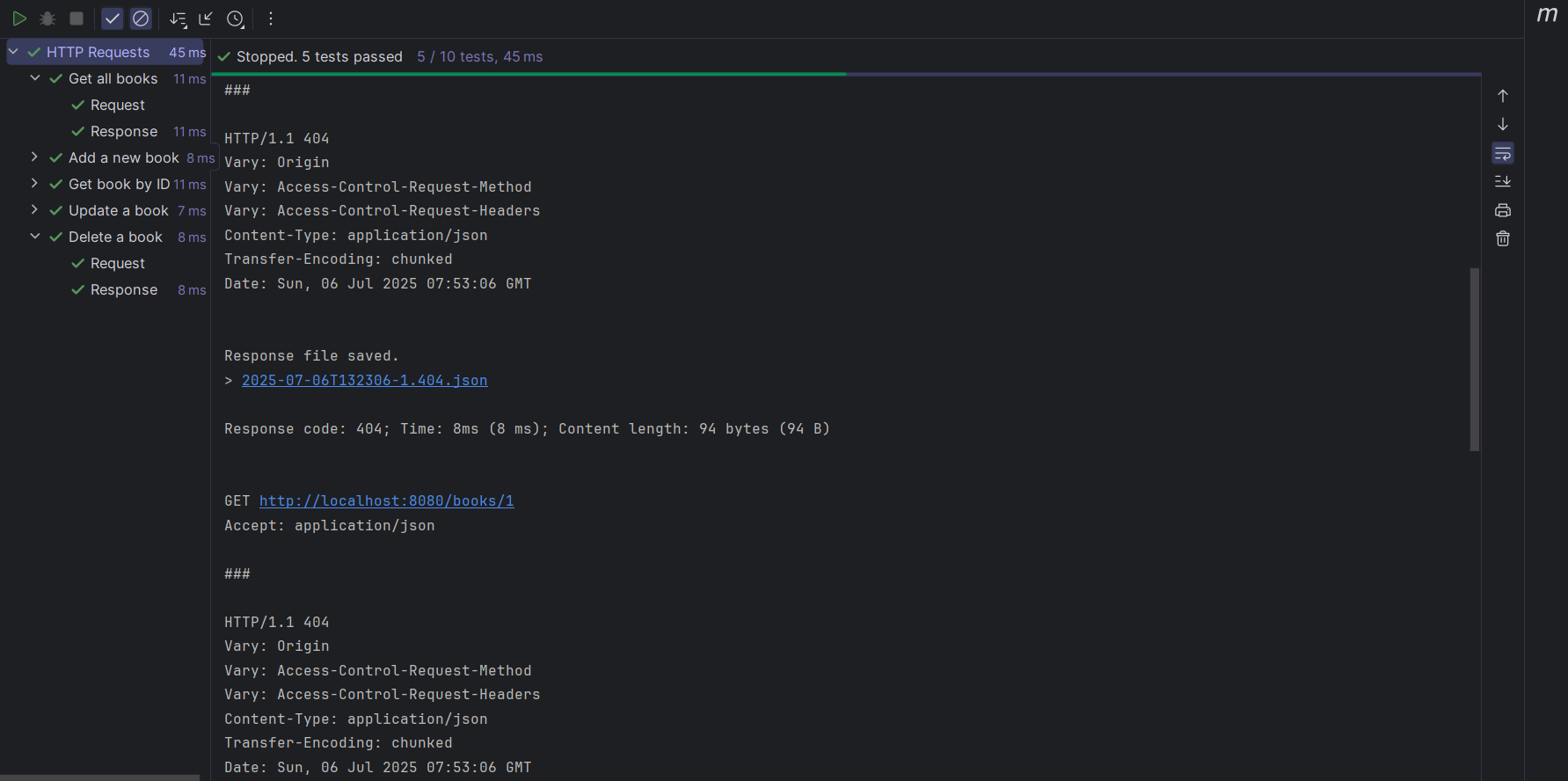
SpringApplication.run(LibraryManApplication.class, args);

}

}

**OUTPUT :**

****

****