**Spring Core 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.

**Steps:**

1. **Set Up a Spring Project:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.
2. **Configure the Application Context:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
3. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.
   * Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Project: LibraryManagement**

**Program:**

**Pom.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>org.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>LibraryManagement</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.2.7</version>  
 </dependency>  
 </dependencies>  
</project>

**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 definitions here -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
   
 <bean id="bookService" class="com.library.service.BookService"/>  
</beans>

**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 addBook() {  
 System.*out*.println("BookService: Adding a book...");  
 bookRepository.save();  
 }  
}

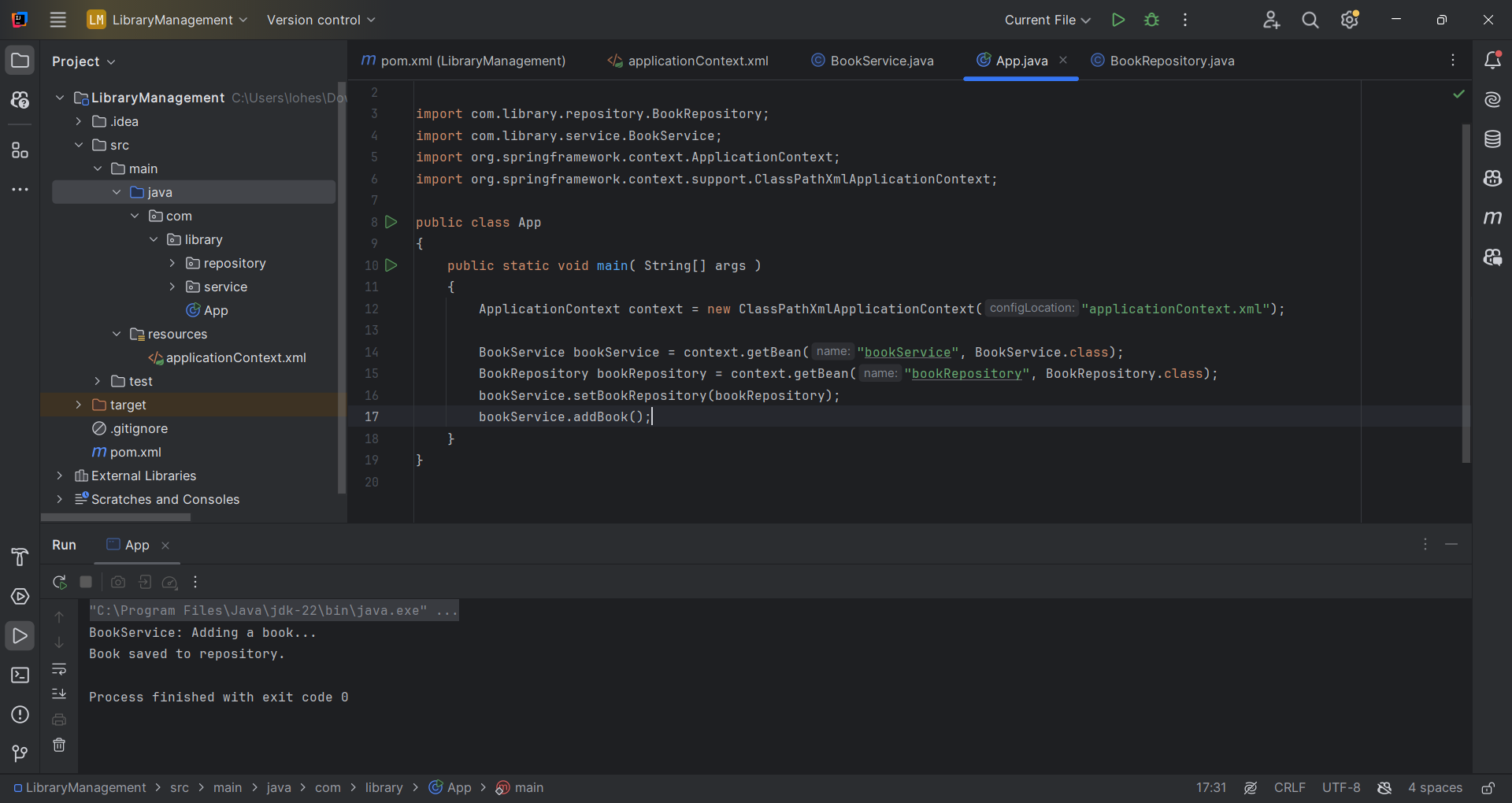
**BookRepository.java**

package com.library.repository;  
  
public class BookRepository{  
 public void save() {  
 System.*out*.println("Book saved to repository.");  
 }  
}

**App.java**

package com.library;  
  
import com.library.repository.BookRepository;  
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 = context.getBean("bookService", BookService.class);  
 BookRepository bookRepository = context.getBean("bookRepository", BookRepository.class);  
 bookService.setBookRepository(bookRepository);  
 bookService.addBook();  
 }  
}

**Output:**



**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.

**Steps:**

1. **Modify the XML Configuration:**
   * Update **applicationContext.xml** to wire **BookRepository** into **BookService**.
2. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.
3. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

**Program:**

**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 definitions here -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
   
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
  
</beans>

**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 addBook() {  
 System.*out*.println("BookService: Adding a book...");  
 bookRepository.save();  
 }  
}

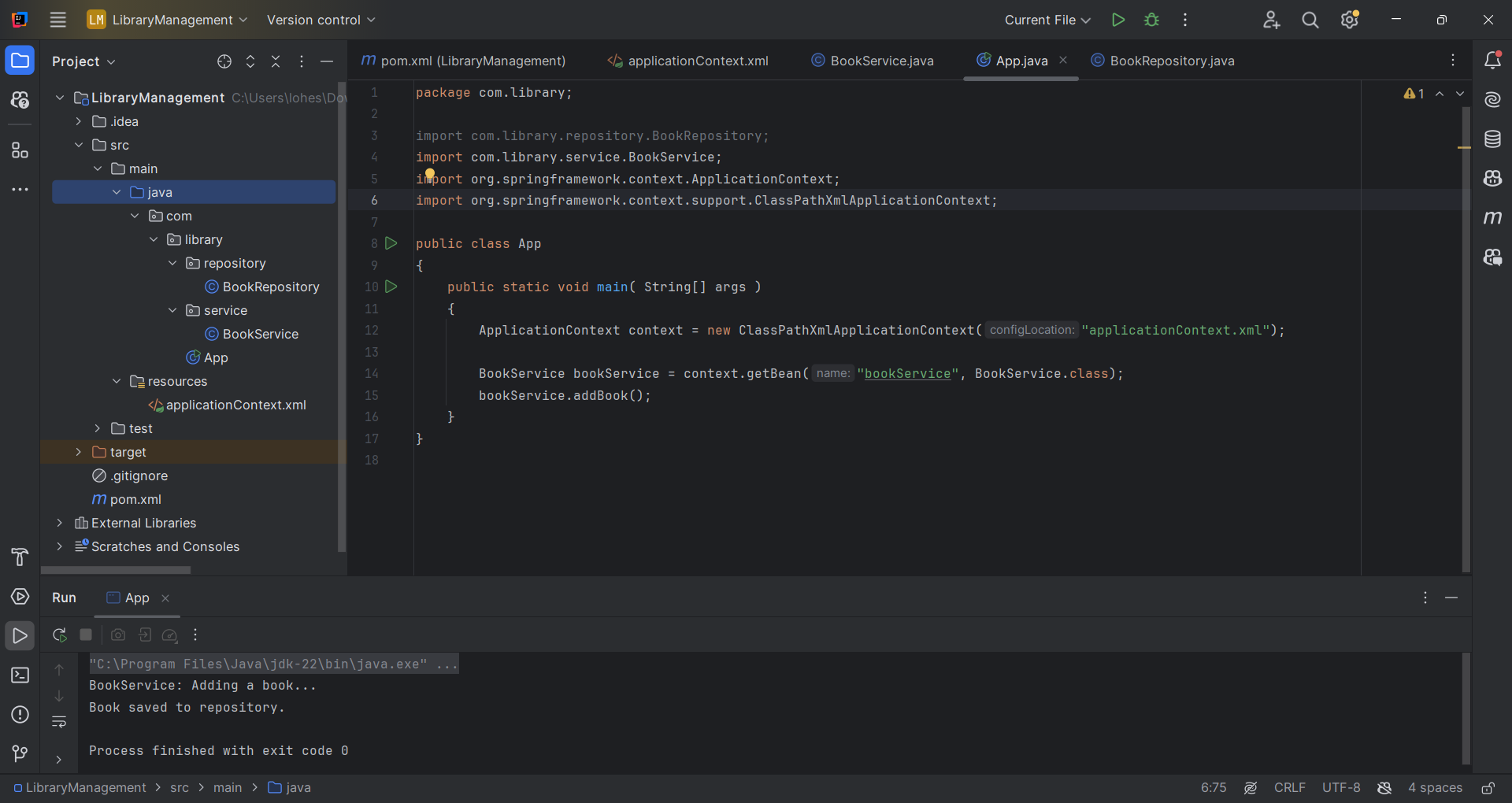
**BookRepository.java**

package com.library.repository;  
  
public class BookRepository{  
 public void save() {  
 System.*out*.println("Book saved to repository.");  
 }  
}

**App.java**

package com.library;  
  
import com.library.repository.BookRepository;  
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 = context.getBean("bookService", BookService.class);  
 bookService.addBook();  
 }  
}

**Output:**



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

**Scenario:**

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

**Steps:**

1. **Add Spring AOP Dependency:**
   * Update **pom.xml** to include Spring AOP dependency.
2. **Create an Aspect for Logging:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.
3. **Enable AspectJ Support:**
   * Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.
4. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

**Program:**

**Pom.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>org.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>LibraryManagement</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Core & Context -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.2.7</version>  
 </dependency>  
  
 <!-- Spring AOP -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>6.2.7</version>  
 </dependency>  
  
 <!-- AspectJ -->  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.9.21</version>  
 </dependency>  
 </dependencies>  
  
</project>

**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"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd">  
  
 <!-- Beans -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
  
 <bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>  
  
 <!-- AOP Config -->  
 <aop:config>  
 <aop:aspect ref="loggingAspect">  
 <aop:around method="logExecutionTime"  
 pointcut="execution(\* com.library.service.\*.\*(..))"  
 arg-names="joinPoint"/>  
 </aop:aspect>  
 </aop:config>  
</beans>

**BookRepository.java**

package com.library.repository;  
  
public class BookRepository {  
 public void save() {  
 System.*out*.println("BookRepository: Saving book to database...");  
 }  
}

**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 addBook() {  
 System.*out*.println("BookService: Adding book...");  
 bookRepository.save();  
 }  
}

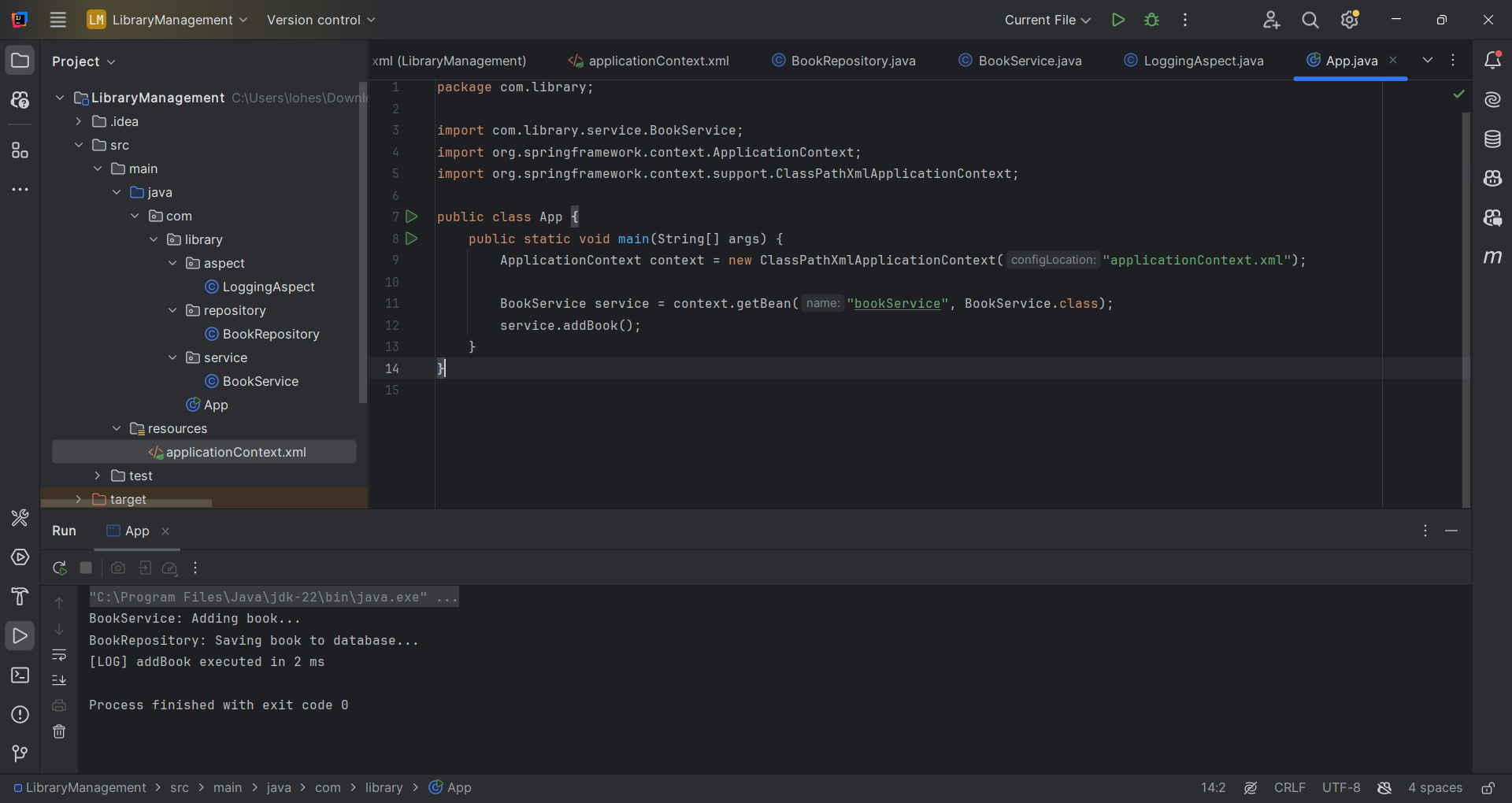
**LoggingAspect.java**

package com.library.aspect;  
  
import org.aspectj.lang.ProceedingJoinPoint;  
  
public class LoggingAspect {  
 public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {  
 long start = System.*currentTimeMillis*();  
  
 Object result = joinPoint.proceed();  
  
 long end = System.*currentTimeMillis*();  
 System.*out*.println("[LOG] " + joinPoint.getSignature().getName() + " executed in " + (end - start) + " ms");  
  
 return result;  
 }  
}

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

**Output:**



**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.

**Steps:**

1. **Create a New Maven Project:**
   * Create a new Maven project named **LibraryManagement**.
2. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Program:**

**Pom.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>org.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>LibraryManagement</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Core & Context -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.2.7</version>  
 </dependency>  
  
 <!-- Spring AOP -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>6.2.7</version>  
 </dependency>  
  
 <!-- AspectJ -->  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.9.21</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>6.2.8</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>

**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"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd">  
  
 <!-- Beans -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
  
 <bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>  
  
 <!-- AOP Config -->  
 <aop:config>  
 <aop:aspect ref="loggingAspect">  
 <aop:around method="logExecutionTime"  
 pointcut="execution(\* com.library.service.\*.\*(..))"  
 arg-names="joinPoint"/>  
 </aop:aspect>  
 </aop:config>  
</beans>

**BookRepository.java**

package com.library.repository;  
  
public class BookRepository {  
 public void save() {  
 System.*out*.println("BookRepository: Saving book to database...");  
 }  
}

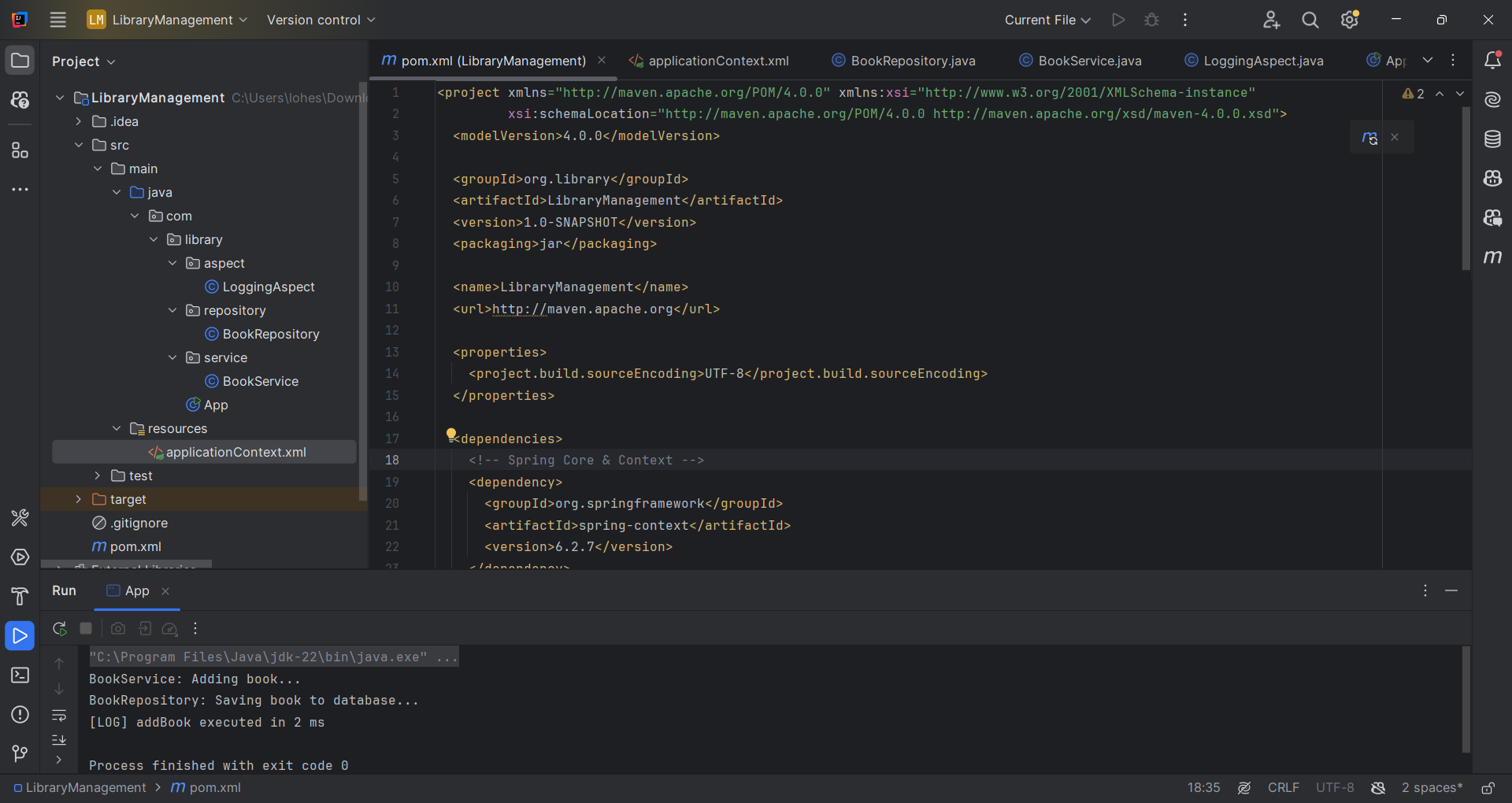
**BookService.java**

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 addBook() {  
 System.*out*.println("BookService: Adding book...");  
 bookRepository.save();  
 }  
}

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

**Output:**



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

**Scenario:**

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

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Project: LibraryManagement**

**Program:**

**Pom.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>org.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>LibraryManagement</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.2.7</version>

</dependency>

</dependencies>

</project>

**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 definitions here -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService"/>

</beans>

**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 addBook() {

System.out.println("BookService: Adding a book...");

bookRepository.save();

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository{

public void save() {

System.out.println("Book saved to repository.");

}

}

**App.java**

package com.library;

import com.library.repository.BookRepository;

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

BookRepository bookRepository = context.getBean("bookRepository", BookRepository.class);

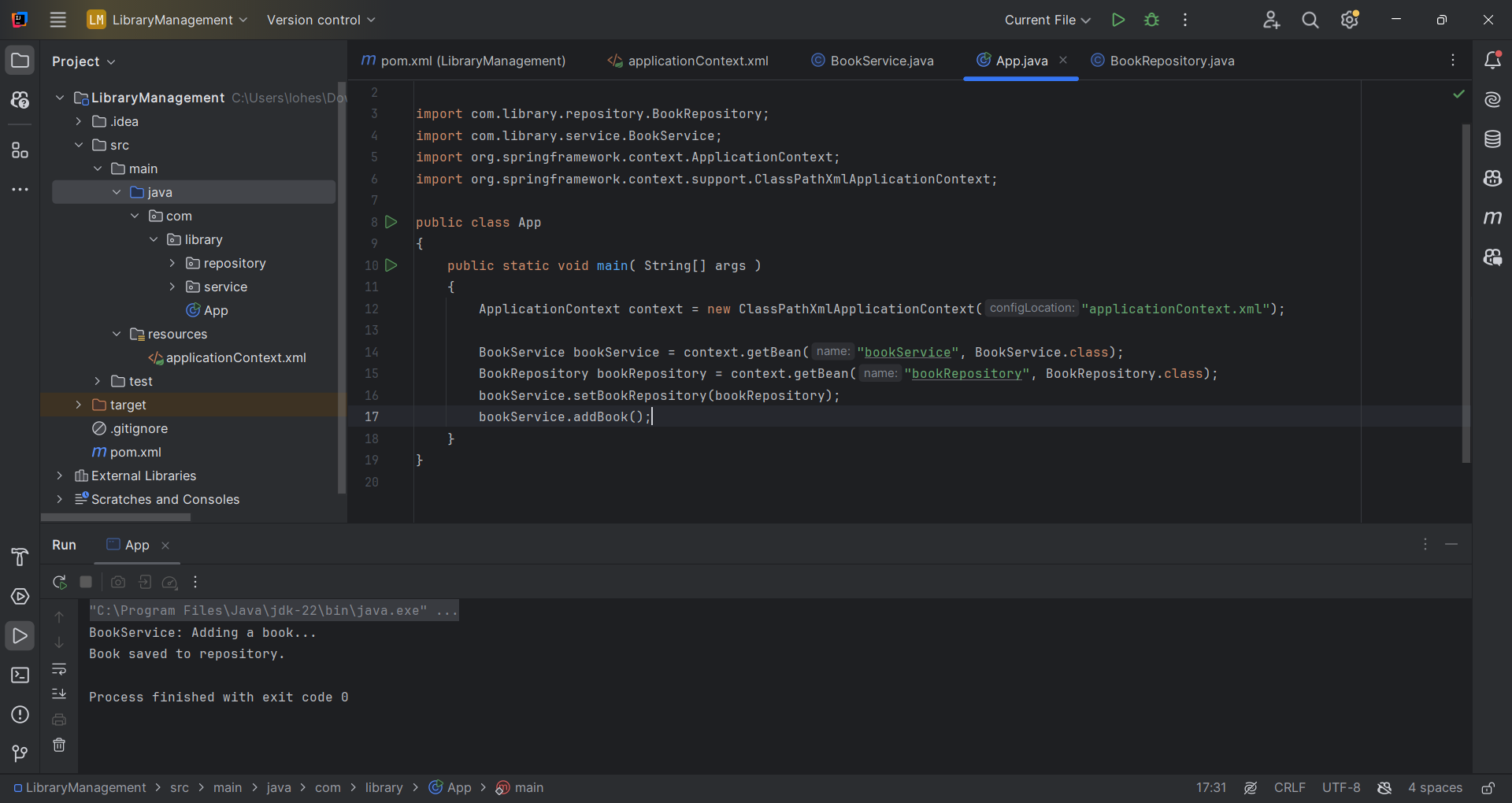
bookService.setBookRepository(bookRepository);

bookService.addBook();

}

}

**Output:**



**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

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

**Steps:**

1. **Enable Component Scanning:**
   * Update **applicationContext.xml** to include component scanning for the **com.library** package.
2. **Annotate Classes:**
   * Use **@Service** annotation for the **BookService** class.
   * Use **@Repository** annotation for the **BookRepository** class.
3. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

**Program**:

**Pom.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>org.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>LibraryManagement</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Core & Context -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.2.7</version>  
 </dependency>  
  
 <!-- Spring AOP -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>6.2.7</version>  
 </dependency>  
  
 <!-- AspectJ -->  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.9.21</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>6.2.8</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>

**ApplicationContext.java**

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd">  
  
 <context:component-scan base-package="com.library"/>  
</beans>

**BookRepository.java**

package com.library.repository;  
  
import org.springframework.stereotype.Repository;  
  
@Repository  
public class BookRepository {  
 public void save() {  
 System.*out*.println("BookRepository: Book saved.");  
 }  
}

**BookService.java**

package com.library.service;  
  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
@Service  
public class BookService {  
  
 private BookRepository bookRepository;  
  
 @Autowired  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook() {  
 System.*out*.println("BookService: Adding book...");  
 bookRepository.save();  
 }  
}

**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");

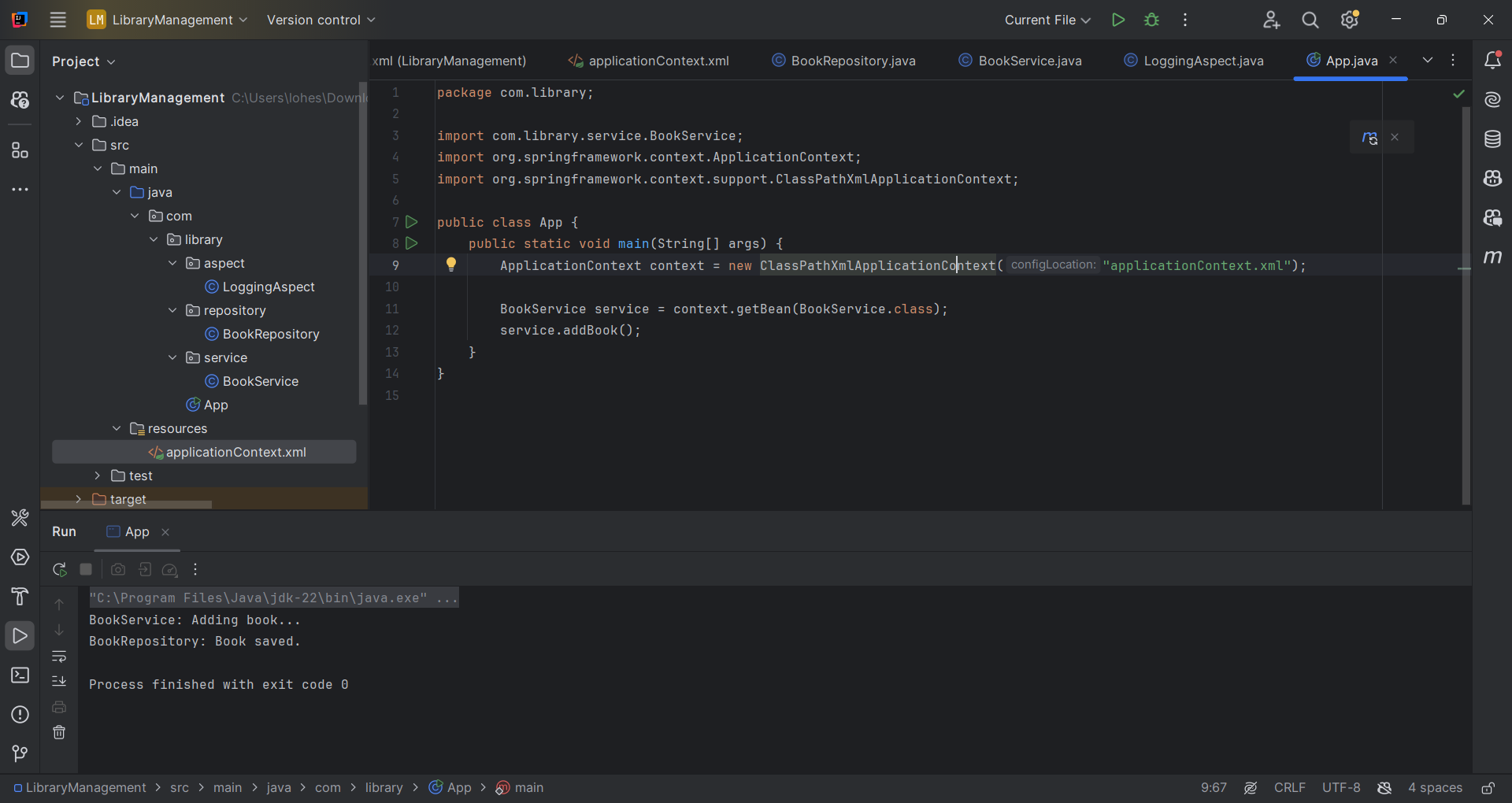
BookService service = context.getBean(BookService.class);

service.addBook();

}

}

**Output:**



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

**Scenario:**

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

**Steps:**

1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.
2. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**Program:**

**ApplicationContext.java**

<?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="LibraryService"/>  
  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

**BookRepository.java**

package com.library.repository;  
  
public class BookRepository {  
 public void save() {  
 System.*out*.println("BookRepository: Saving book to DB...");  
 }  
}

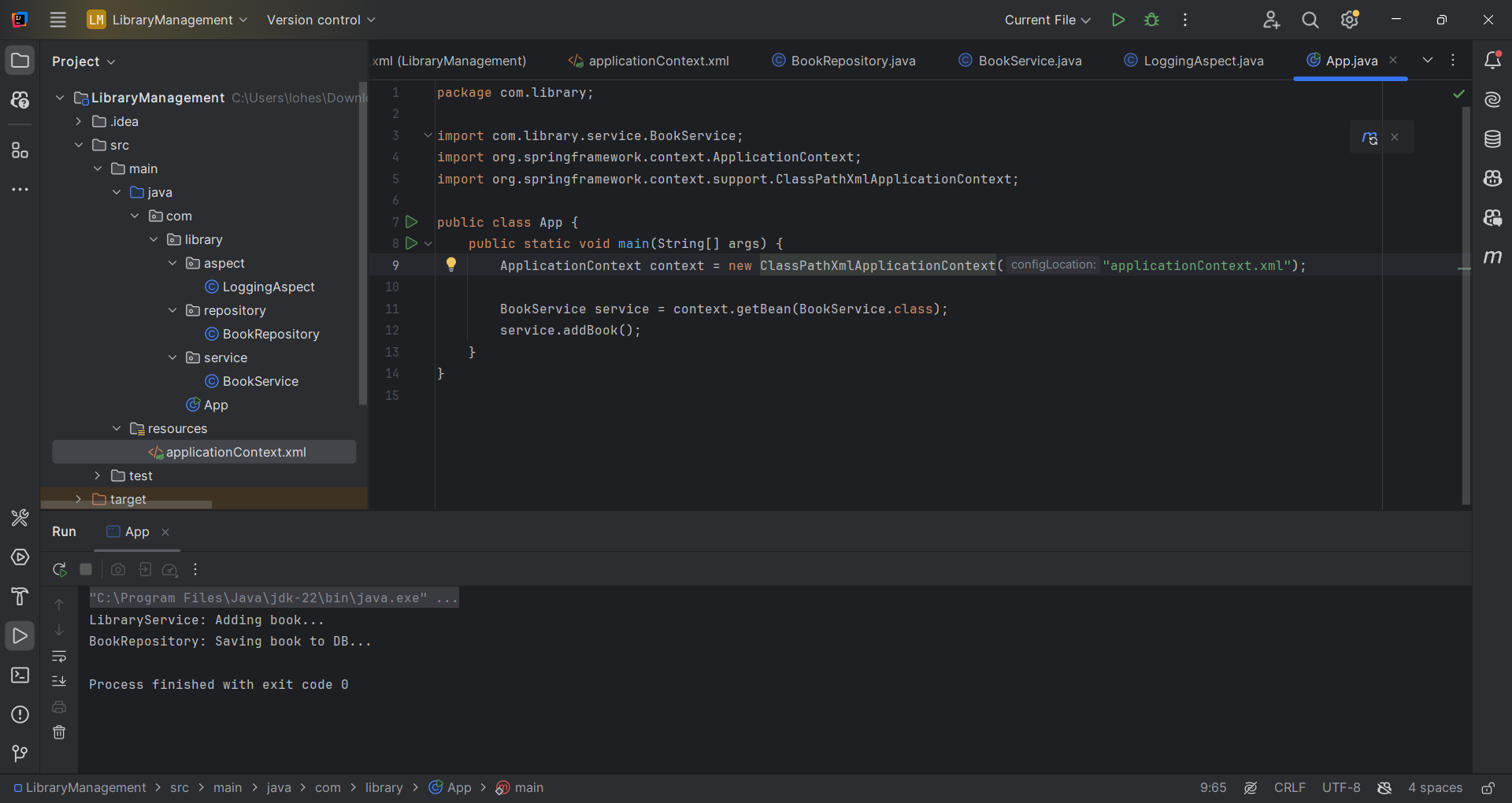
**BookService.java**

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private String serviceName;  
 private BookRepository bookRepository;  
  
 public BookService(String serviceName) {  
 this.serviceName = serviceName;  
 }  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook() {  
 System.*out*.println(serviceName + ": Adding book...");  
 bookRepository.save();  
 }  
}

**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 service = context.getBean(BookService.class);  
 service.addBook();  
 }  
}

**Output:**



**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.

**Steps:**

1. **Define an Aspect:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect**.
2. **Create Advice Methods:**
   * Define advice methods in **LoggingAspect** for logging before and after method execution.
3. **Configure the Aspect:**
   * Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.
4. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class to verify the AOP functionality.

**Program:**

**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"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd">  
  
 <!-- Enable AOP proxying -->  
 <aop:config>  
 <aop:aspect ref="loggingAspect">  
 <aop:pointcut id="allServiceMethods" expression="execution(\* com.library.service.\*.\*(..))"/>  
 <aop:before pointcut-ref="allServiceMethods" method="beforeAdvice"/>  
 <aop:after pointcut-ref="allServiceMethods" method="afterAdvice"/>  
 </aop:aspect>  
 </aop:config>  
  
 <!-- Beans -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
 <bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>  
</beans>

**BookRepository.java**

package com.library.repository;  
  
public class BookRepository {  
 public void save() {  
 System.*out*.println("BookRepository: Book saved to database.");  
 }  
}

**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 addBook() {  
 System.*out*.println("BookService: Adding a book...");  
 bookRepository.save();  
 }  
}

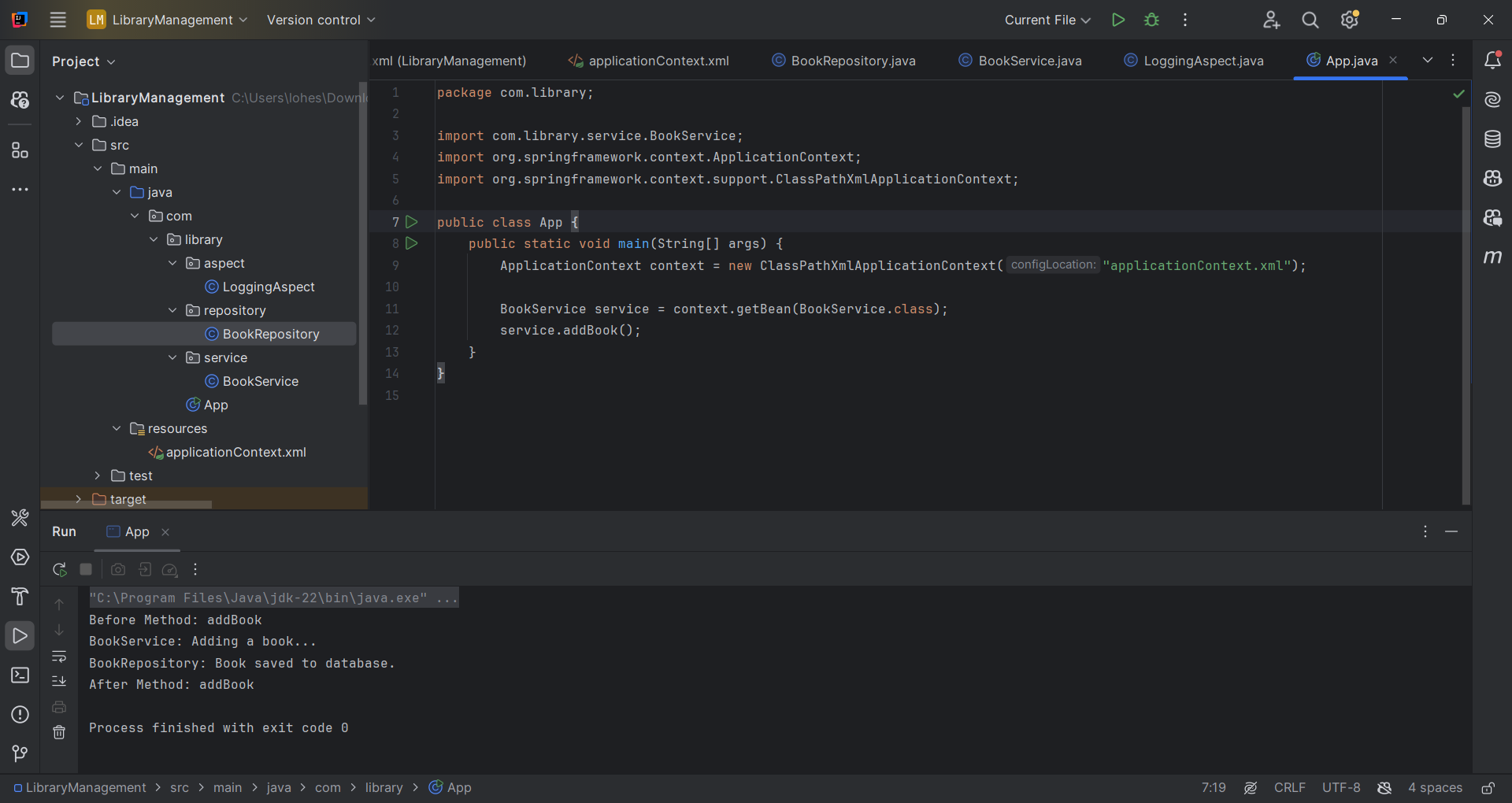
**LoggingAspect.java**

package com.library.aspect;  
  
import org.aspectj.lang.JoinPoint;  
  
public class LoggingAspect {  
  
 public void beforeAdvice(JoinPoint joinPoint) {  
 System.*out*.println("Before Method: " + joinPoint.getSignature().getName());  
 }  
  
 public void afterAdvice(JoinPoint joinPoint) {  
 System.*out*.println("After Method: " + joinPoint.getSignature().getName());  
 }  
}

**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 service = context.getBean(BookService.class);  
 service.addBook();  
 }  
}

**Output:**



**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**.
4. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

**Program:**

**Application.properties:**

spring.application.name=LibraryManagement  
  
  
spring.datasource.url=jdbc:h2:mem:librarydb  
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update  
  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console

**Book.java**

package com.library.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.Id;  
  
@Entity  
public class Book {  
 @Id  
 @GeneratedValue  
 private int id;  
 private String title;  
 private String author;  
  
 public Book() {  
 }  
  
 public Book(int id, String title, String author) {  
 this.id = id;  
 this.title = title;  
 this.author = author;  
 }  
  
 public long getId() {  
 return id;  
 }  
  
 public void setId(int 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.library.controller;  
  
import com.library.entity.Book;  
import com.library.service.BookService;  
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 BookService service;  
  
 @GetMapping  
 public List<Book> getAllBooks() {  
 return service.getAllBooks();  
 }  
  
 @GetMapping("/{id}")  
 public Book getBookById(@PathVariable int id) {  
 return service.getBookById(id);  
 }  
  
 @PostMapping  
 public Book addBook(@RequestBody Book book) {  
 return service.addBook(book);  
 }  
  
 @PutMapping  
 public Book updateBook(@RequestBody Book book) {  
 return service.updateBook(book);  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBook(@PathVariable int id) {  
 service.deleteBook(id);  
 }  
}

**BookService.java**

package com.library.service;  
  
import com.library.entity.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class BookService {  
  
 @Autowired  
 private BookRepository repository;  
  
 public List<Book> getAllBooks() {  
 return repository.findAll();  
 }  
  
 public Book getBookById(int id) {  
 return repository.findById(id).orElse(null);  
 }  
  
 public Book addBook(Book book) {  
 return repository.save(book);  
 }  
  
 public Book updateBook(Book book) {  
 return repository.save(book);  
 }  
  
 public void deleteBook(int id) {  
 repository.deleteById(id);  
 }  
}

**BookRepository.java**

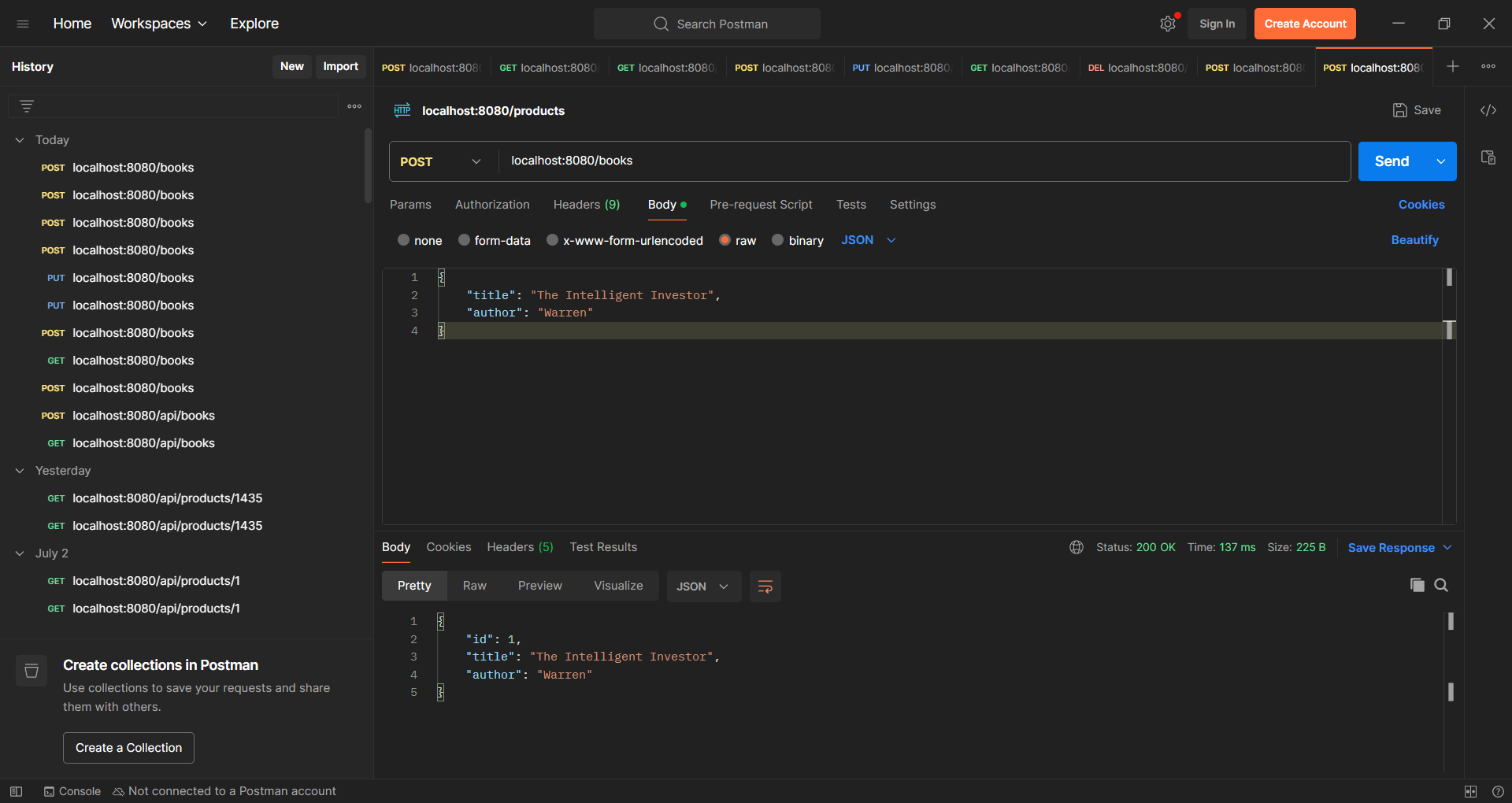
package com.library.repository;  
  
import com.library.entity.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface BookRepository extends JpaRepository<Book, Integer> {  
}

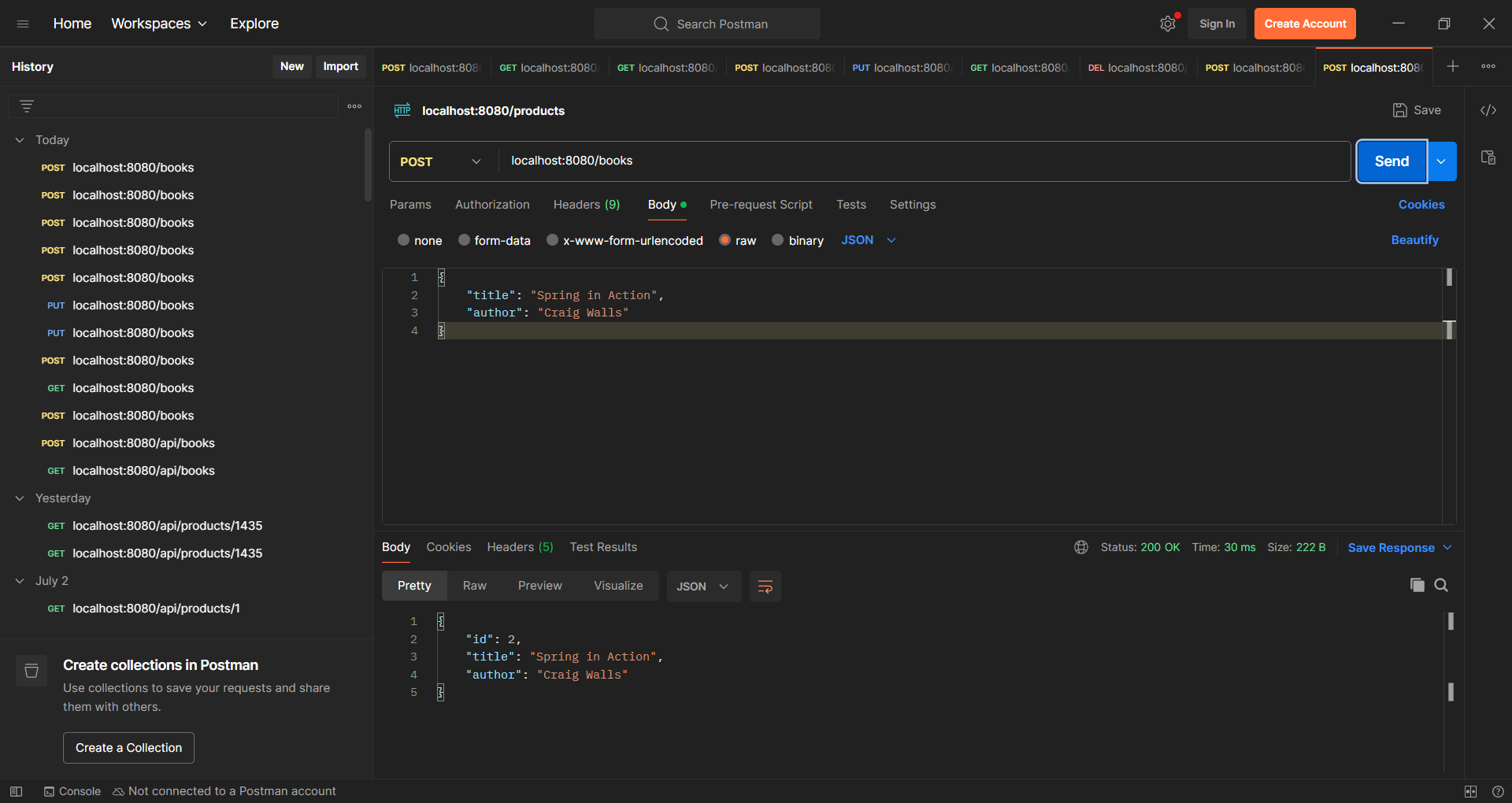
**LibraryManagementApplication.java**

package com.library;  
  
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);  
 }  
}

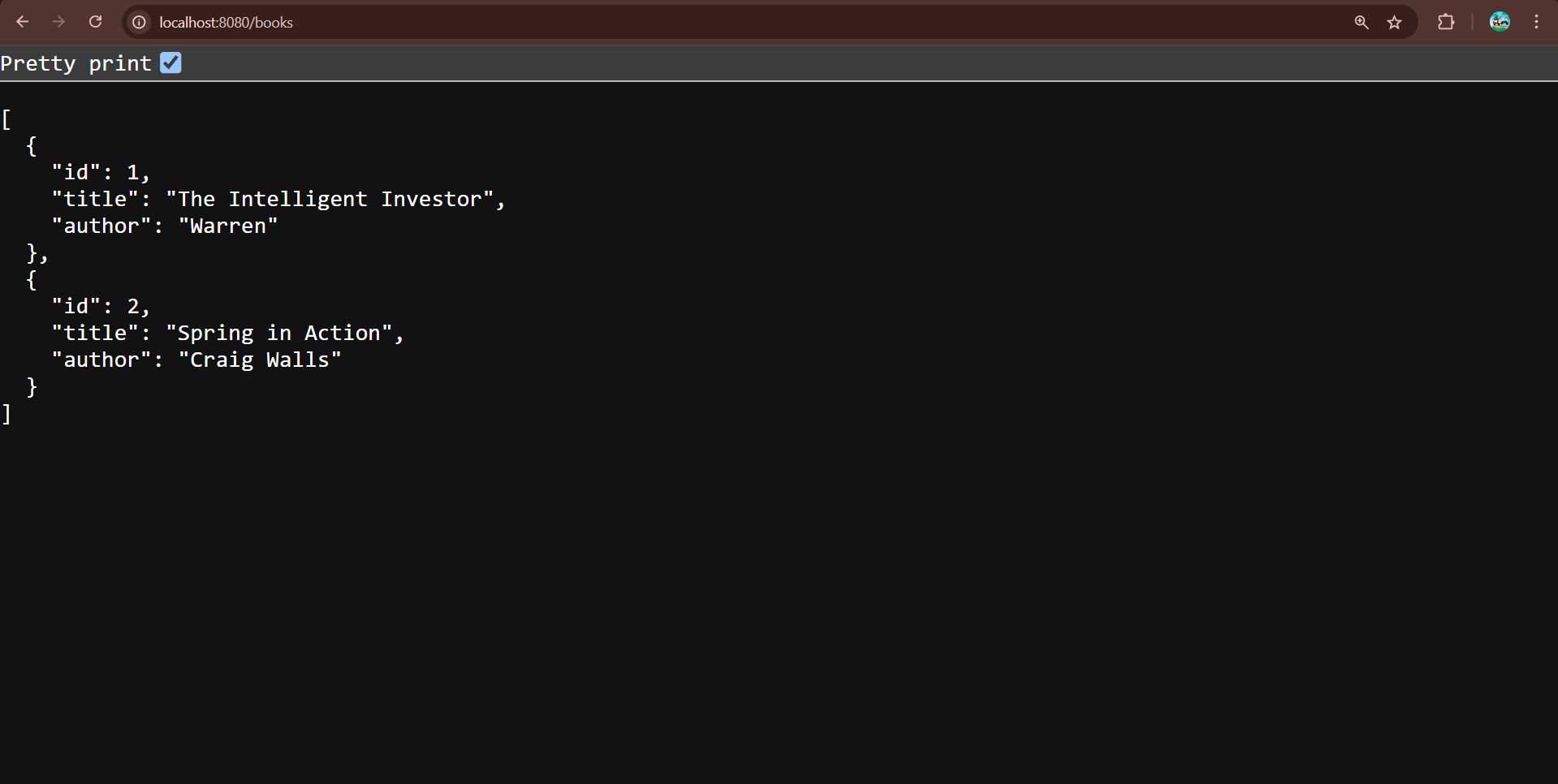
**Output:**

**Adding Books using POST method:**



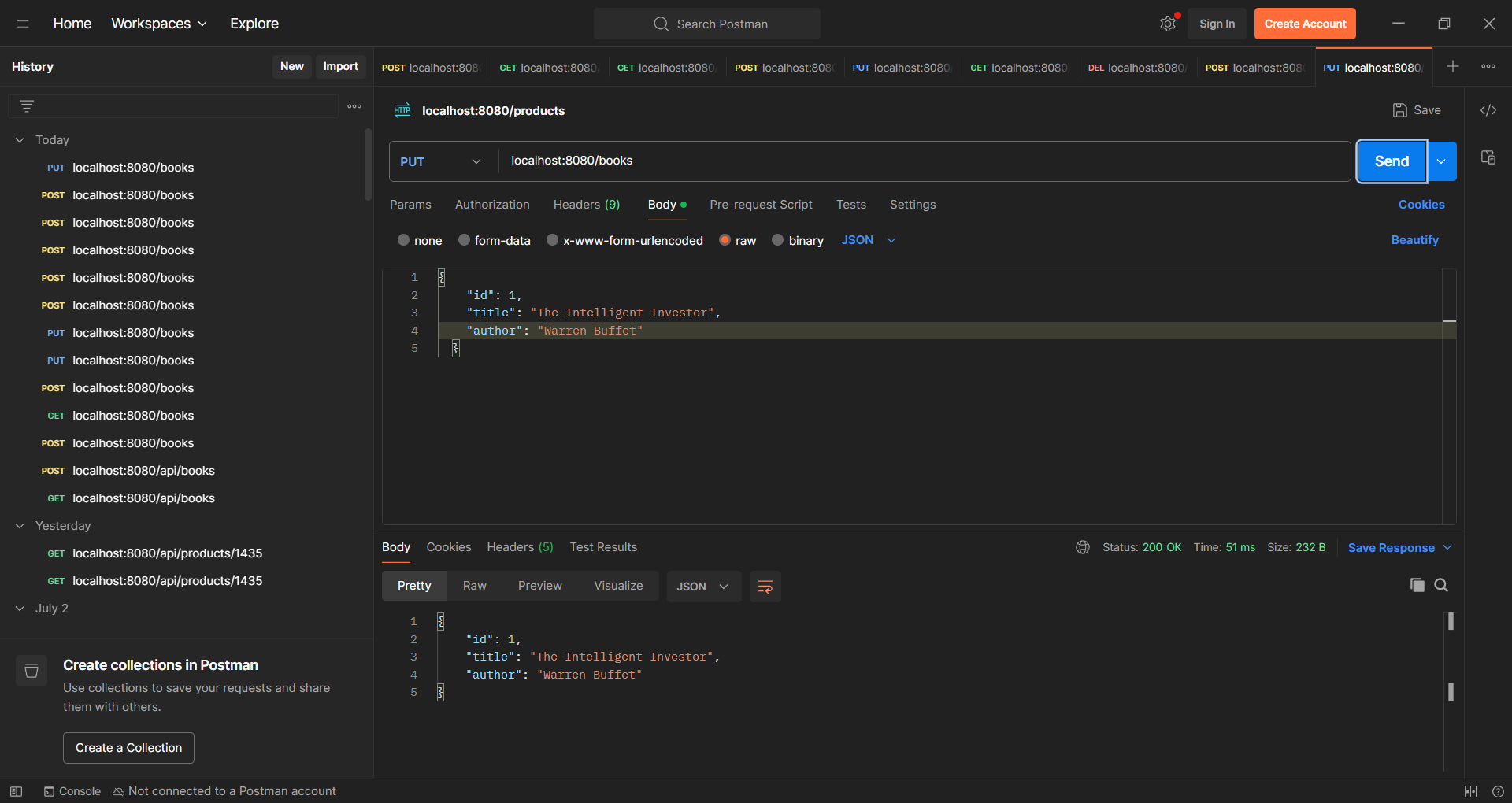


**Performing GET request:**

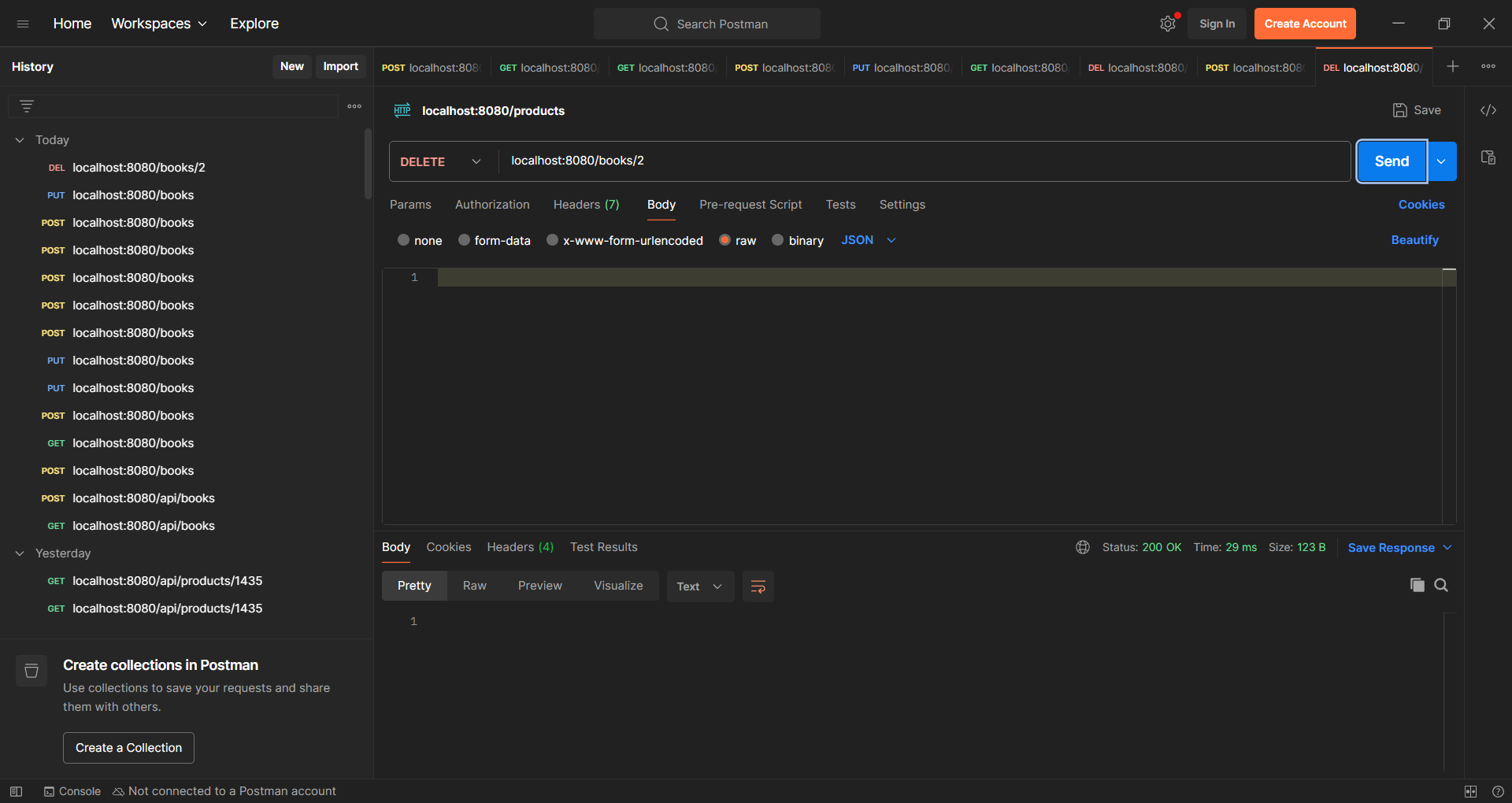


**Updating Books:**

Now let’s update the author of the book “The Intelligent Investor” from “Warren” to “Warren Buffet” using PostMan.



**Deleting Books:**



**After Deleting:**

