**Candidate Name**:- SHARANYA K

**Superset ID**:-6419887

**WEEK – 3 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)**

**Module 5 - Spring Core and Maven**

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

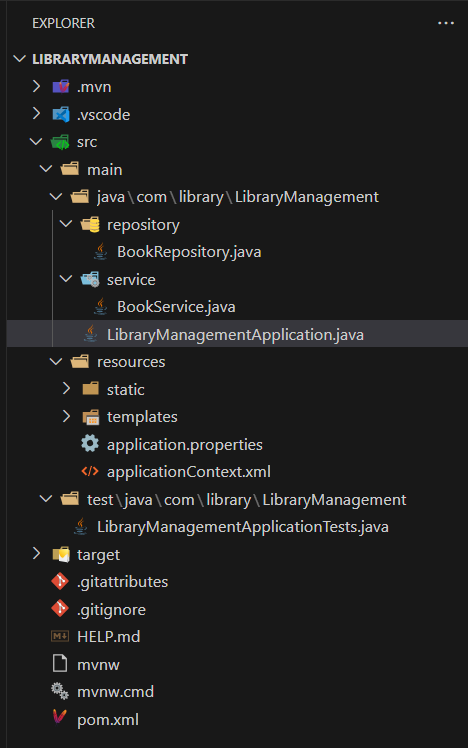
**Scenario:**

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

**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.

**SOLUTION:**



**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

https://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>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.1.5</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.11.0</version>

<configuration>

<source>17</source>

<target>17</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**BookRepository.java**

package com.library.LibraryManagement.repository;

public class BookRepository {

    public void saveBook(String title) {

        System.out.println("Saving book: " + title);

    }

}

**BookService.java**

package com.library.LibraryManagement.service;

import com.library.LibraryManagement.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

    }

    public void addBook(String title) {

        bookRepository.saveBook(title);

    }

}

**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

        https://www.springframework.org/schema/beans/spring-beans.xsd">

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

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

        <property name="bookRepository" ref="bookRepository"/>

    </bean>

</beans>

**LibraryManagementApplication.java**

package com.library.LibraryManagement;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.LibraryManagement.service.BookService;

public class LibraryManagementApplication {

    public static void main(String[] args) {

        System.out.println("Starting Spring XML App...");

        try (ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml")) {

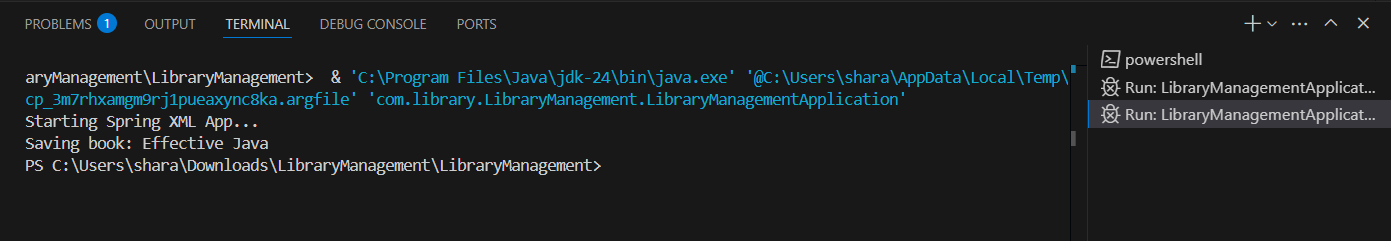
            BookService service = context.getBean("bookService", BookService.class);

            service.addBook("Effective Java");

        }

    }

}



**Exercise 2: Implementing Dependency Injection**

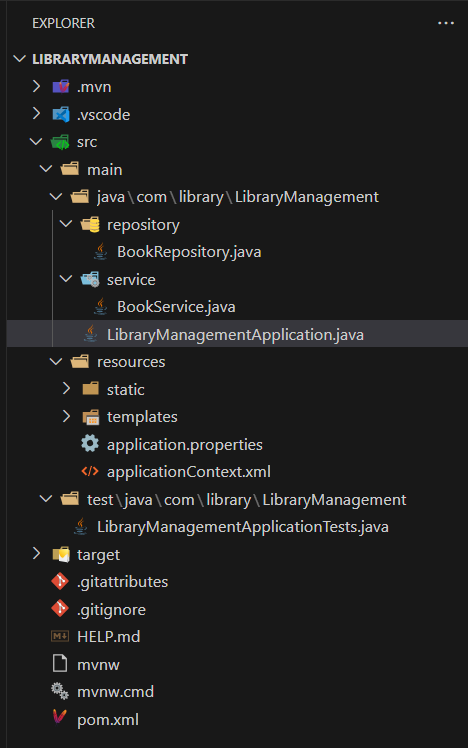
**Scenario:**

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.

**SOLUTION:**



**LibraryManagementApplication.java**

package com.library.LibraryManagement;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.LibraryManagement.service.BookService;

public class LibraryManagementApplication {

    public static void main(String[] args) {

        System.out.println("Starting Spring DI Application...");

        try (ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml")) {

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

            bookService.addBook("Clean Code");

        }

    }

}

**BookRepository.java**

package com.library.LibraryManagement.repository;

public class BookRepository {

    public void saveBook(String title) {

        System.out.println("Saving book: " + title);

    }

}

**BookService.java**

package com.library.LibraryManagement.service;

import com.library.LibraryManagement.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

    }

    public void addBook(String title) {

        bookRepository.saveBook(title);

    }

}

**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

        https://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Bean for BookRepository -->

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

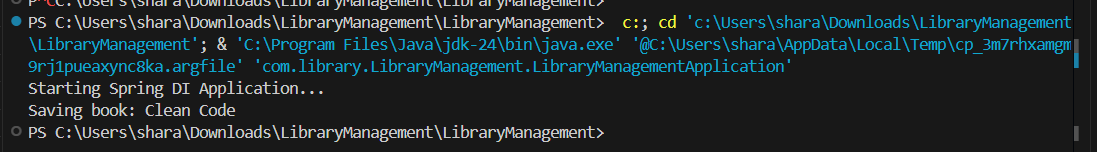
    <!-- Bean for BookService with setter injection -->

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

        <property name="bookRepository" ref="bookRepository"/>

    </bean>

</beans>



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

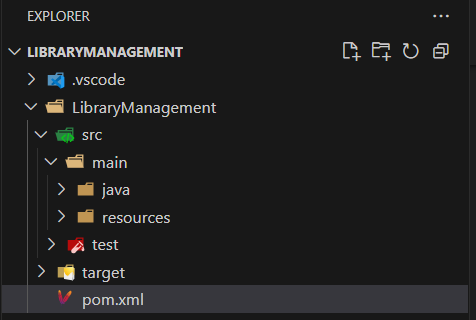
**Scenario:**

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.

**Solution:**

**Created the project LibraryManagement**

**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>com.library</groupId>

    <artifactId>management</artifactId>

    <version>1.0-SNAPSHOT</version>

    <properties>

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

    </properties>

    <dependencies>

        <!-- Spring Context -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>5.3.34</version>

        </dependency>

        <!-- Spring AOP -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-aop</artifactId>

            <version>5.3.34</version>

        </dependency>

        <!-- Spring WebMVC -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-webmvc</artifactId>

            <version>5.3.34</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

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

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

                <version>3.11.0</version>

                <configuration>

                    <source>1.8</source>

                    <target>1.8</target>

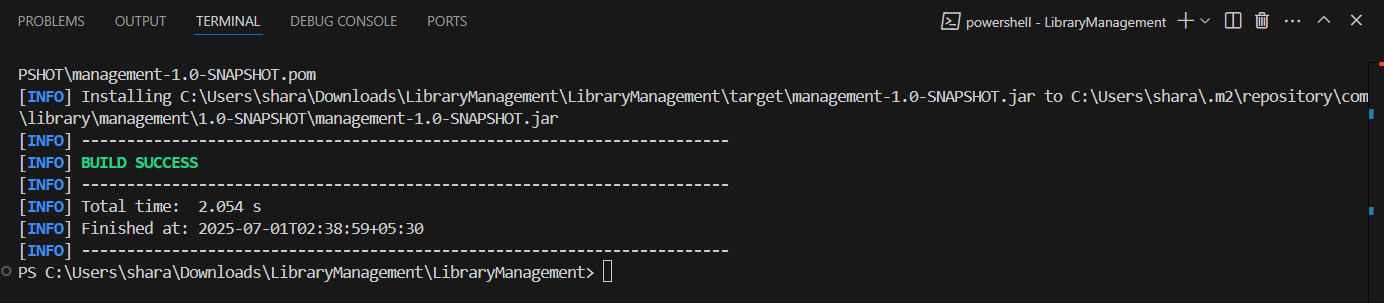
                </configuration>

            </plugin>

        </plugins>

    </build>

</project>



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

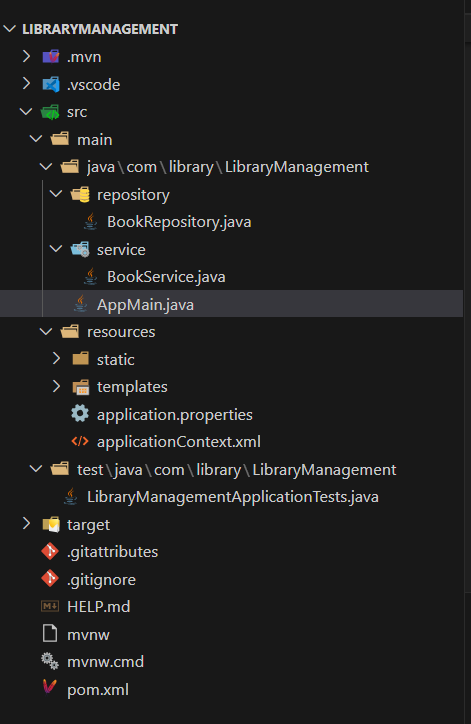
**Scenario:**

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.

**SOLUTION:**

****

**AppMain.java**

package com.library.LibraryManagement;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.LibraryManagement.service.BookService;

public class AppMain {

    public static void main(String[] args) {

        System.out.println("Starting Spring XML App...");

        try (ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml")) {

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

            bookService.addBook("Spring Framework Mastery");

        }

    }

}

**BookRepository.java**

package com.library.LibraryManagement.repository;

public class BookRepository {

    public void saveBook(String bookName) {

        System.out.println("Book saved: " + bookName);

    }

}

**BookService.java**

package com.library.LibraryManagement.service;

import com.library.LibraryManagement.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

    }

    public void addBook(String bookName) {

        bookRepository.saveBook(bookName);

    }

}

**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

           https://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Define BookRepository bean -->

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

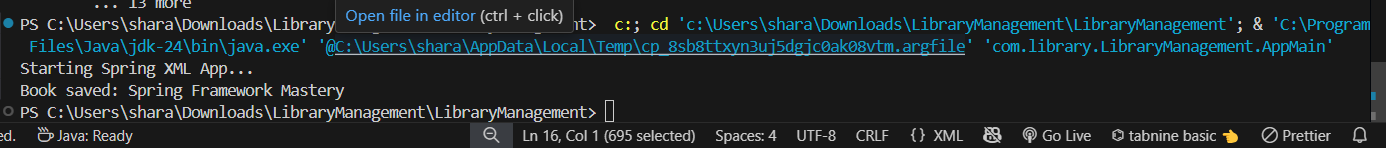
    <!-- Define BookService bean and inject BookRepository -->

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

        <property name="bookRepository" ref="bookRepository" />

    </bean>

</beans>



**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.

**SOLUTION:**

**AppMain.java**

package com.library.LibraryManagement;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.LibraryManagement.service.BookService;

public class AppMain {

public static void main(String[] args) {

System.out.println("Starting Spring XML App...");

try (ConfigurableApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml")) {

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

service.save("Clean Code");

}

}

}

**BookService.java**

package com.library.LibraryManagement.service;

import com.library.LibraryManagement.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    public BookService(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

        System.out.println("Constructor injection successful");

    }

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

        System.out.println("Setter injection successful");

    }

    public void save(String bookName) {

        bookRepository.saveBook(bookName);

    }

}

**BookRepository.java**

package com.library.LibraryManagement.repository;

public class BookRepository {

    public void saveBook(String bookName) {

        System.out.println("Book saved: " + bookName);

    }

}

**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

         https://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- BookRepository Bean -->

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

    <!-- BookService Bean with both constructor and setter injection -->

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

        <!-- Constructor Injection -->

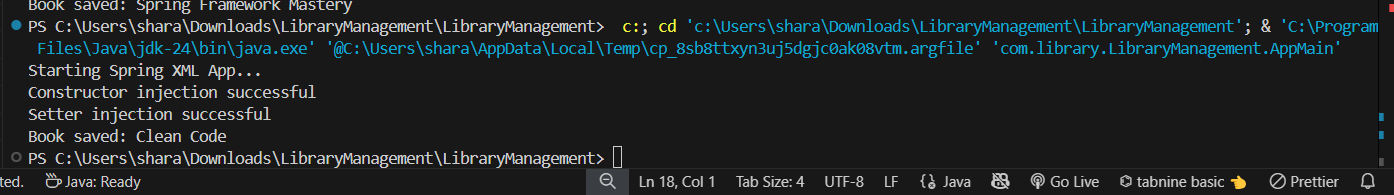
        <constructor-arg ref="bookRepository" />

        <!-- Setter Injection -->

        <property name="bookRepository" ref="bookRepository" />

    </bean>

</beans>



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

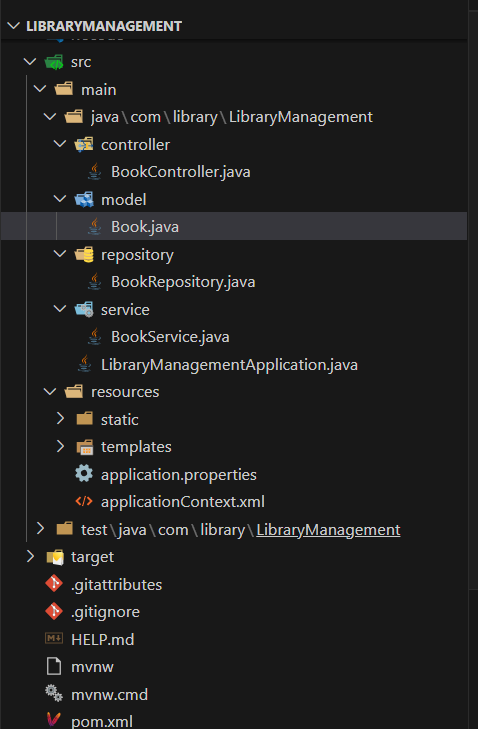
**Scenario:**

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.

**SOLUTION:**

****

**Application.properties**

# H2 Database settings

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA settings

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

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

# H2 Console (optional)

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

**application.content.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

         https://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- BookRepository Bean -->

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

    <!-- BookService Bean with both constructor and setter injection -->

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

        <!-- Constructor Injection -->

        <constructor-arg ref="bookRepository" />

        <!-- Setter Injection -->

        <property name="bookRepository" ref="bookRepository" />

    </bean>

</beans>

**Book.java**

package com.library.LibraryManagement.model;

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;

    // Getters & 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; }

}

**BookRepository.java**

package com.library.LibraryManagement.repository;

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

import com.library.LibraryManagement.model.Book;

public interface BookRepository extends JpaRepository<Book, Long> {

}

**BookService.java**

package com.library.LibraryManagement.service;

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

import org.springframework.stereotype.Service;

import com.library.LibraryManagement.model.Book;

import com.library.LibraryManagement.repository.BookRepository;

@Service

public class BookService {

    @Autowired

    private BookRepository bookRepository;

    public void saveBook(String title) {

        Book book = new Book();

        book.setTitle(title);

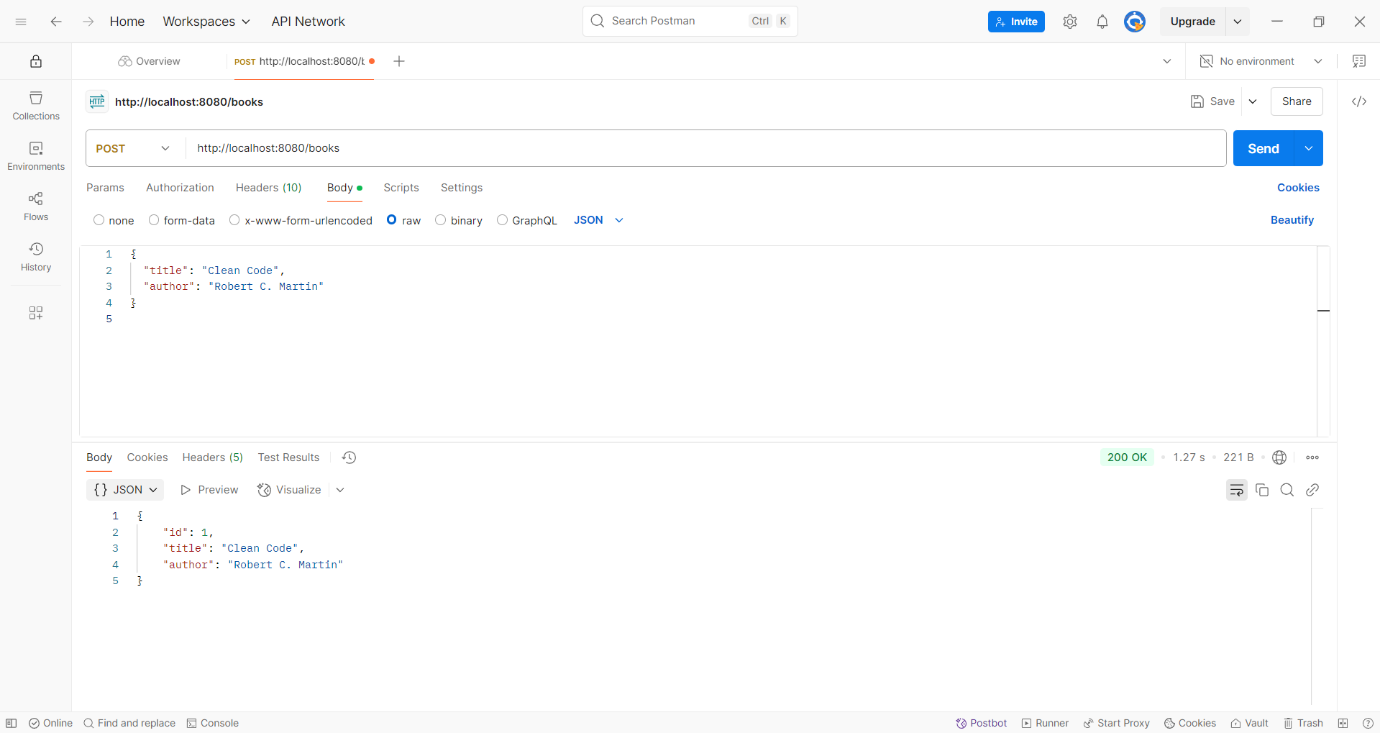
        bookRepository.save(book);

    }

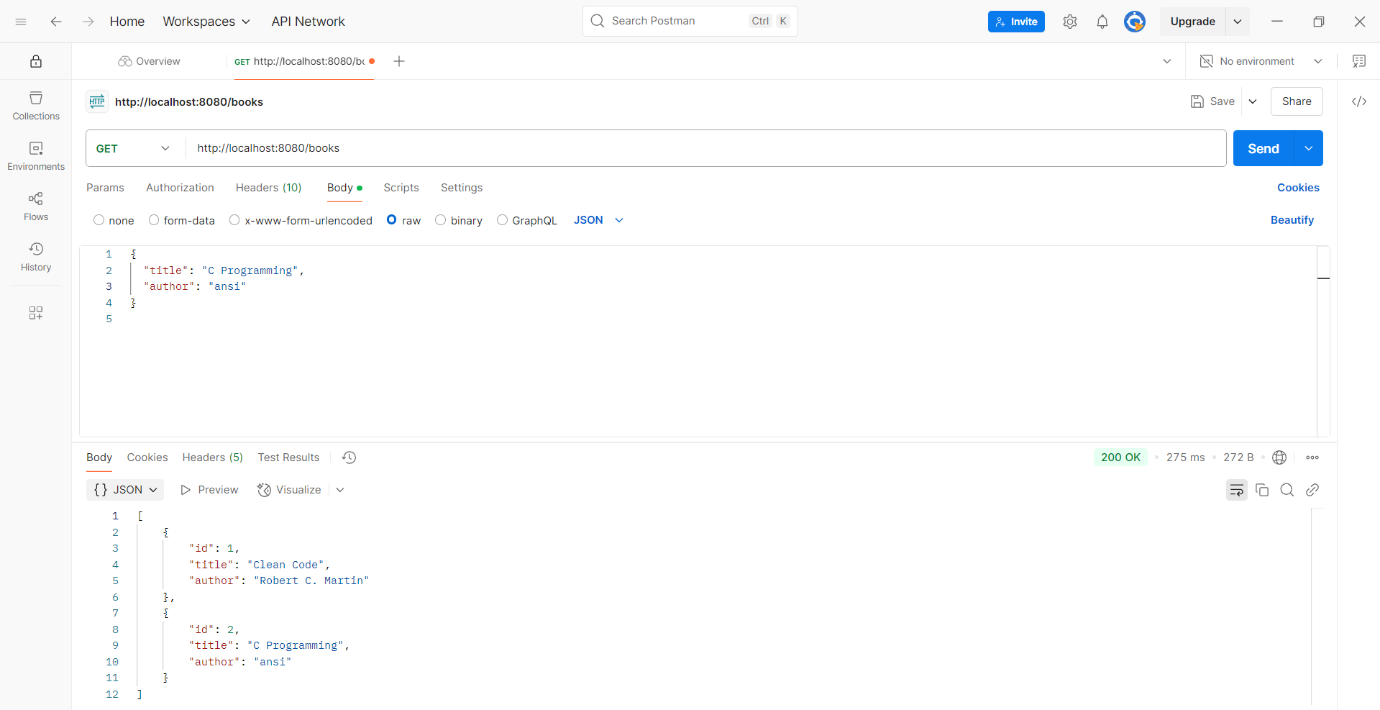
}

**CRUD OPERATION USING POSTMAN**

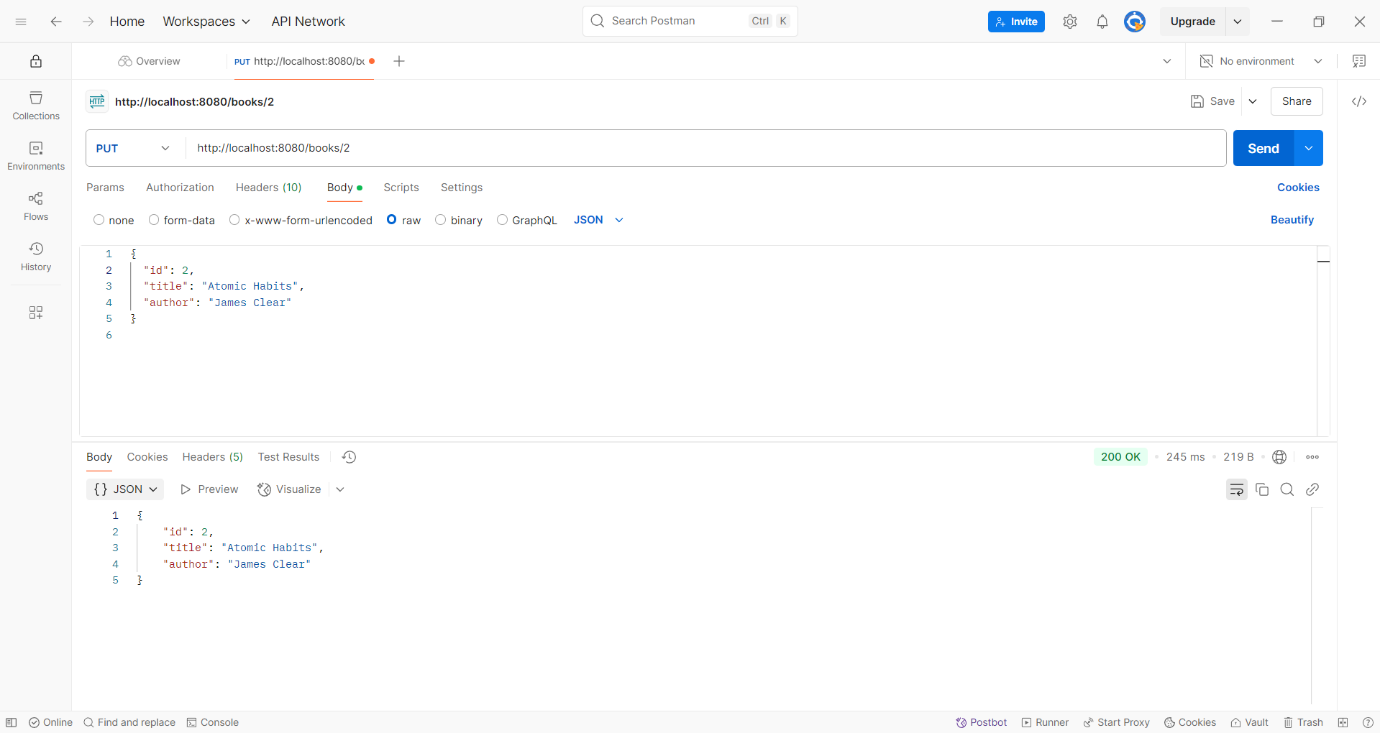
**POST OPERATION**



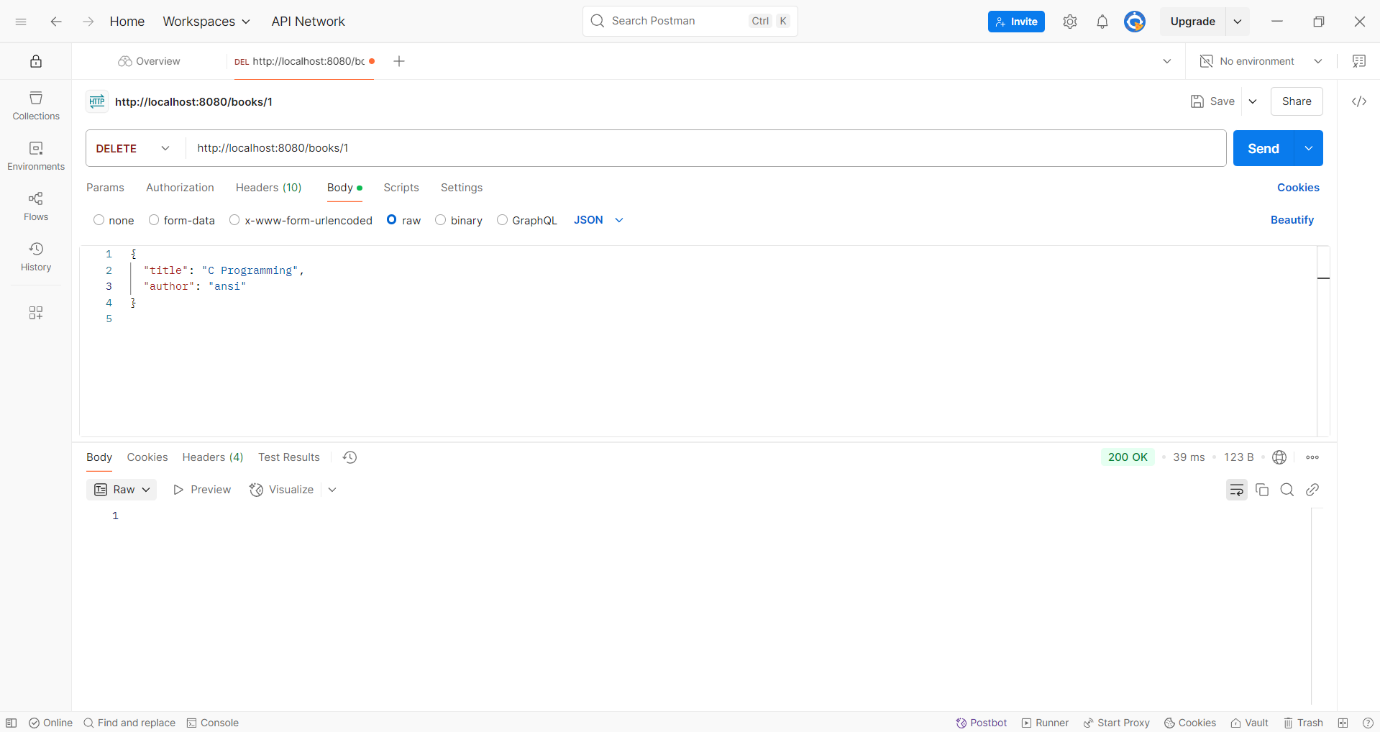
**GET OPERATION**

****

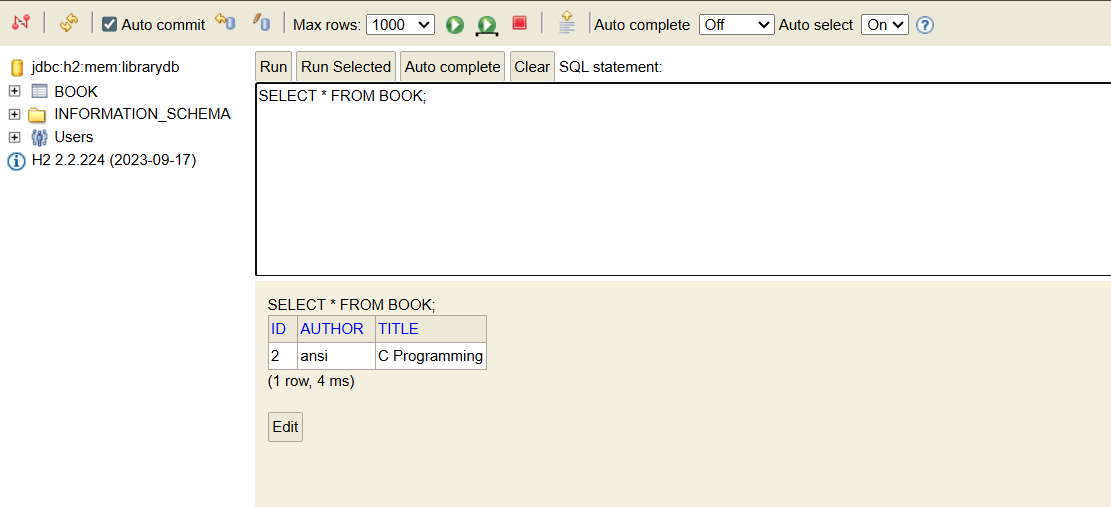
**PUT OPERATION**

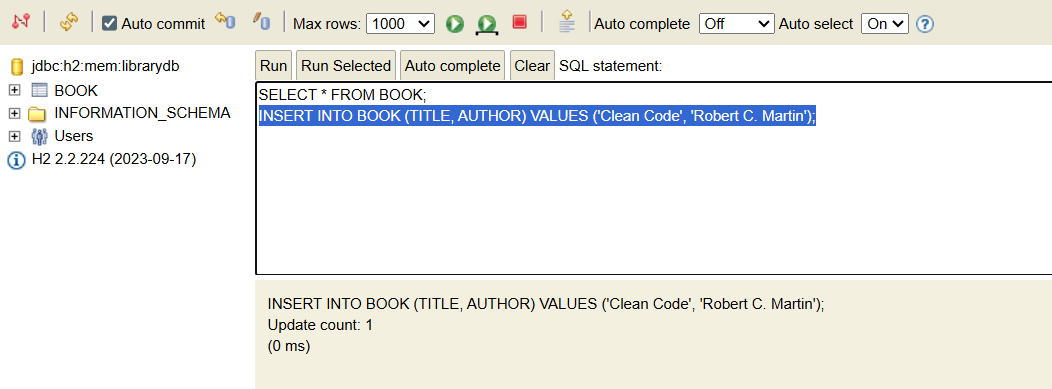
****

**DELETE OPERATION**

****

**Operation done using the H2 Console using SQL quiery**

****

****

