NAME : ANITA.P

SUPERSET ID : 6419735

WEEK – 3

MANDATORY - EXERCISES

1) 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.

ANSWER :

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>LibraryManagement</artifactId>
  <version>1.0</version>
  <dependencies>
    <!-- Spring Context (includes Spring Core and Beans) -->
```

```
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.33</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">
  <!-- Repository Bean -->
  <bean id="bookRepository" class="com.library.repository.BookRepository" />
  <!-- Service Bean -->
  <bean id="bookService" class="com.library.service.BookService">
    <property name="bookRepository" ref="bookRepository" />
  </bean>
</beans>
```

BookRepository.java

```
package com.library.repository;
public class BookRepository {
  public void saveBook(String bookName) {
    System.out.println("Saving book: " + bookName);
  }
}
```

BookService.java

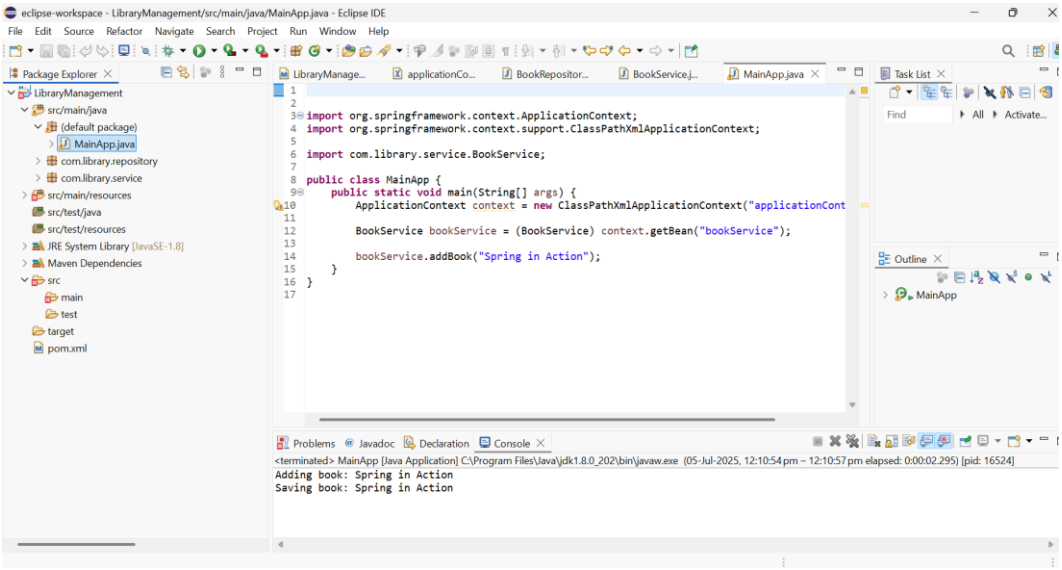
```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

  private BookRepository bookRepository;

  public void setBookRepository(BookRepository bookRepository) {

    this.bookRepository = bookRepository;

  }

  public void addBook(String bookName) {
```

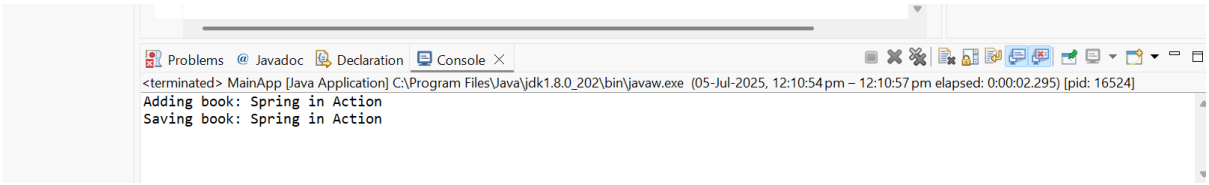
```
        System.out.println("Adding book: " + bookName);
        bookRepository.saveBook(bookName);
    }}
```

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");
        BookService bookService = (BookService) context.getBean("bookService");
        bookService.addBook("Spring in Action");
    }
}
```

OUTPUT





2) **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.

ANSWER :

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>
```

BookService.java

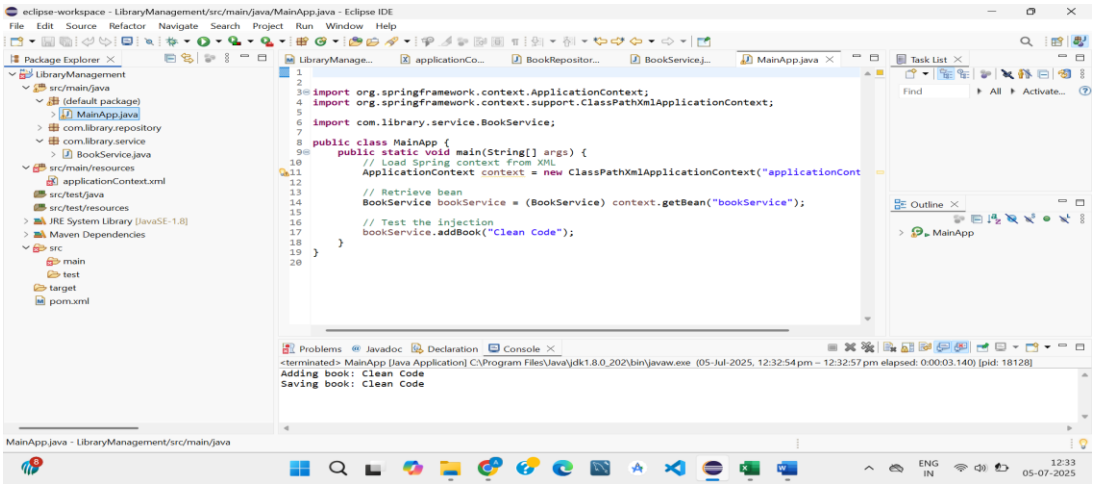
```
package com.library.service;
```

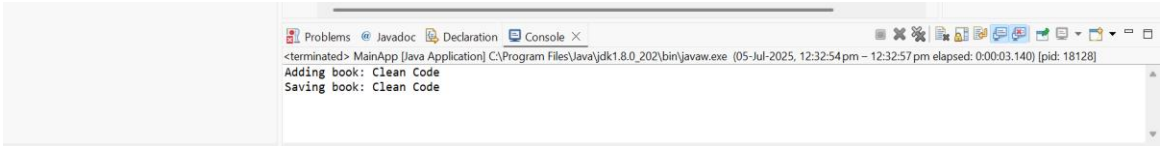
```
import com.library.repository.BookRepository;
public class BookService {
   private BookRepository bookRepository;
   public void setBookRepository(BookRepository bookRepository) {
      this.bookRepository = bookRepository;  }
   public void addBook(String bookName) {
      System.out.println("Adding book: " + bookName);
      bookRepository.saveBook(bookName);
   }}
```

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");
      BookService bookService = (BookService)
context.getBean("bookService");
      bookService.addBook("Clean Code");
   }}
```

OUTPUT :





3) **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.

ANSWER :

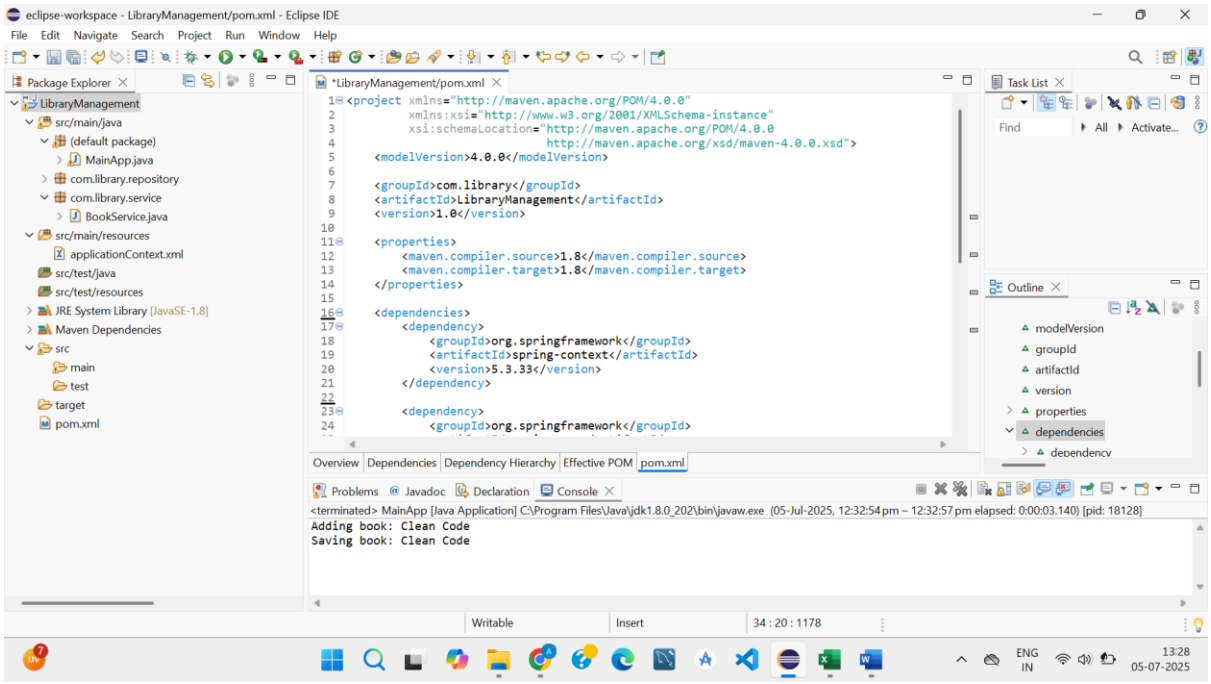
Pom.xml

```
<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>LibraryManagement</artifactId>
    <version>1.0</version>
```

```
<properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.33</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>5.3.33</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.3.33</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
```

```
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
  </build>
</project>
```

OUTPUT :



4) Spring Data JPA - Quick Example

User.java

```
package com.example.entity;
import jakarta.persistence.*;
@Entity
public class User {
```

```
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
  private String email;
  public Long getId() { return id; }
  public void setId(Long id) { this.id = id; }
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public String getEmail() { return email; }
  public void setEmail(String email) { this.email = email; } }
```

UserRepository.java

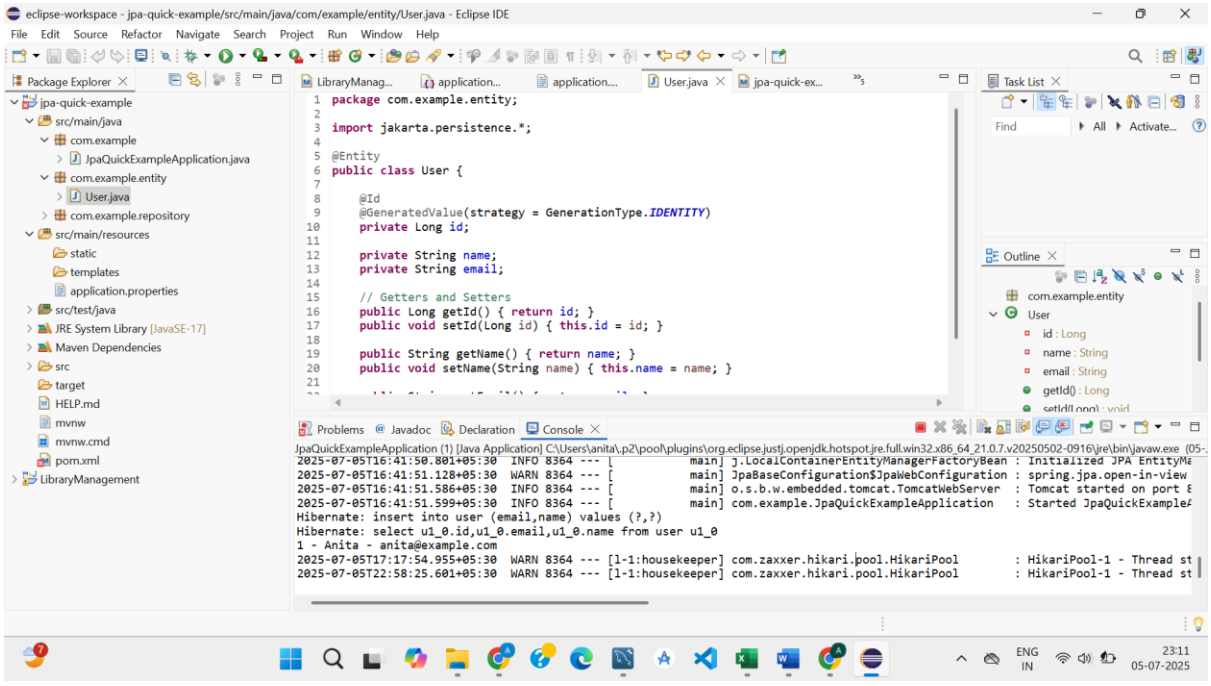
```
package com.example.repository;
import com.example.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
public interface UserRepository extends JpaRepository<User, Long> {
}
```

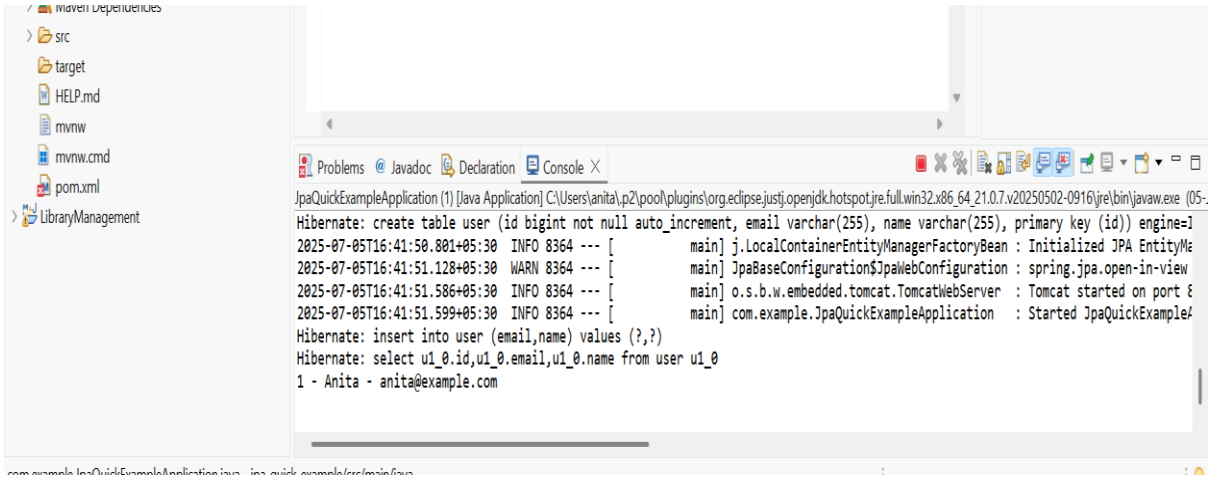
JpaQuickExampleApplication.java

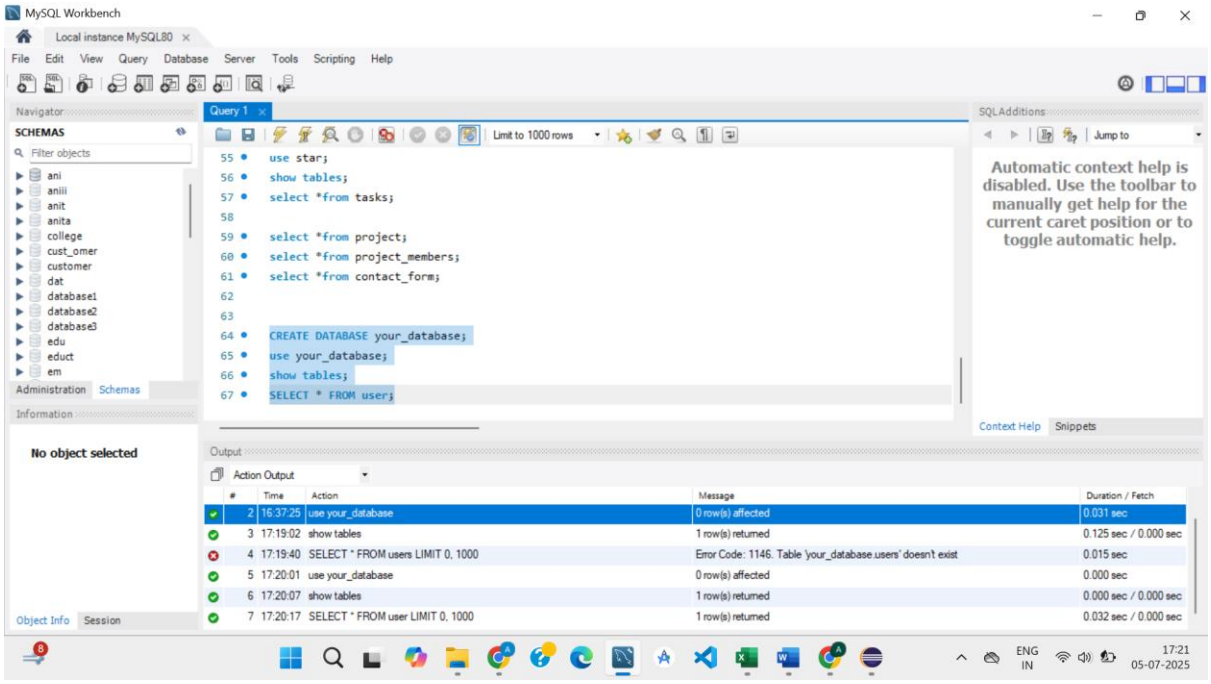
```
package com.example;
import com.example.entity.User;
import com.example.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class JpaQuickExampleApplication implements
CommandLineRunner {
  @Autowired
  private UserRepository userRepository;
  public static void main(String[] args) {
    SpringApplication.run(JpaQuickExampleApplication.class, args);
  }
  @Override
  public void run(String... args) {
    User user = new User();
     user.setName("Anita");
```

```
        user.setEmail("anita@example.com");
        userRepository.save(user);
        userRepository.findAll().forEach(u ->
            System.out.println(u.getId() + " - " + u.getName() + " - " +
u.getEmail())
        );
    }
}
```

OUTPUT :







5) Difference between JPA, Hibernate and Spring Data JPA

Feature/Aspect	**JPA(Java Persistence API)**	**Hibernate**	**Spring Data JPA**
Type	Specification (API only)	Implementation of JPA + extras	Abstraction layer over JPA + Spring support
ORM Support	Defines ORM standards	Full ORM implementation	Uses JPA + underlying ORM like Hibernate
Provider Required	Yes (e.g., Hibernate, EclipseLink)	No, it *is* a provider	Yes (requires JPA provider like Hibernate)
Query Language	JPQL (Java Persistence Query Lang)	HQL (Hibernate Query Lang)	Derived query methods or @Query annotation
Easy of Use	Moderate	Requires deeper understanding	Very easy (simplified API with Spring Boot)

CRUD Operations	Manually implemented	Manually or using Hibernate tools	Auto-generated from JpaRepository
Use case	When following JPA spec is critical	When low-level control is needed	For rapid Spring-based CRUD apps
Code Example	@Entity + EntityManager	SessionFactory + HQL	JpaRepository + Auto Query Methods
Maintenance Effort	Higher	Higher	Low (less code to maintain)
Batch & Lazy Loading	Supported via annotations	Fully Supported + Tunable	Supported (via JPA provider)
Tooling & Documentation	Extensive (Java EE / Jakarta docs)	Extensive (hibernate.org)	Extensive (Spring docs + community support)

ADDITIONAL EXERCISES

1) **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.

ANSWER :

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>
```

BookService.java

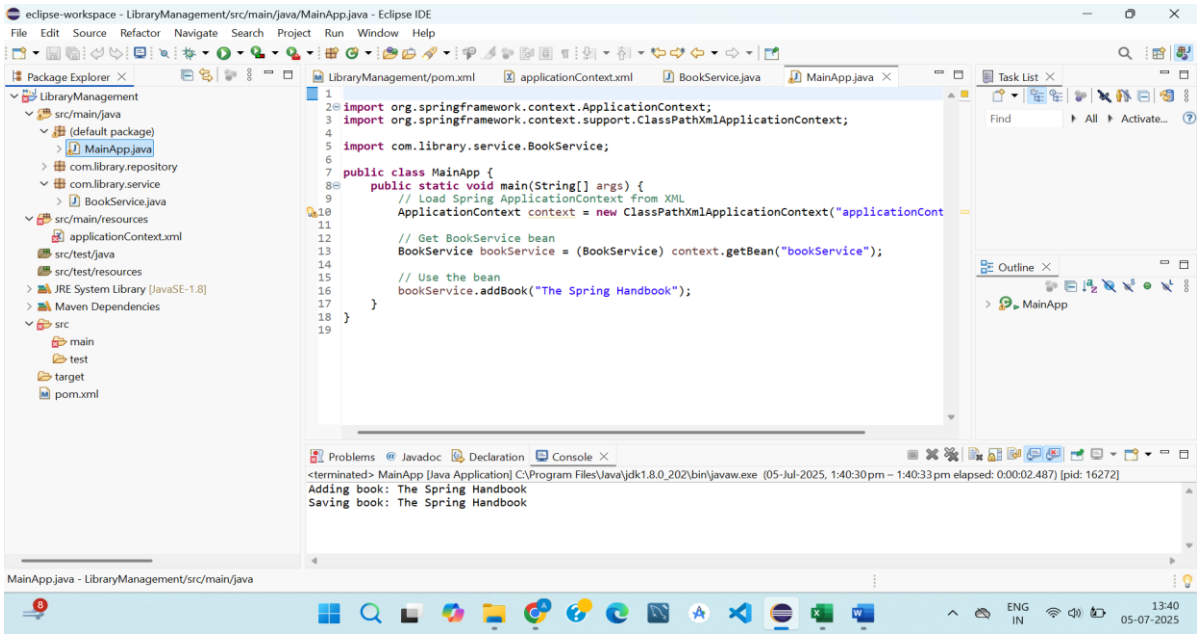
```
package com.library.service;
import com.library.repository.BookRepository;
public class BookService {
   private BookRepository bookRepository;
   public void setBookRepository(BookRepository bookRepository) {
      this.bookRepository = bookRepository;
   }
   public void addBook(String bookName)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

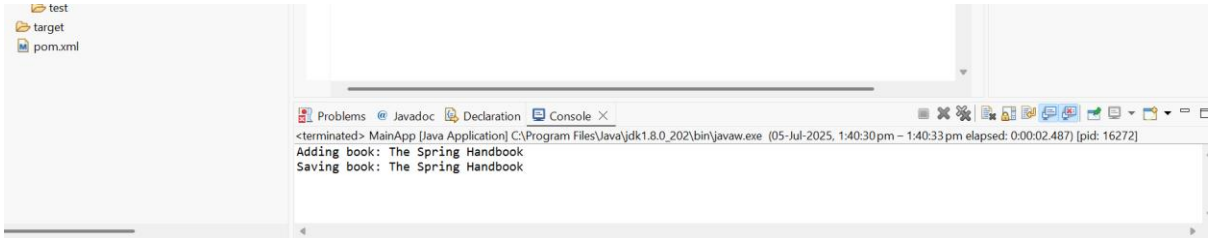
}}

MainApp

```
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");

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

        bookService.addBook("The Spring Handbook");

    }}
```

OUTPUT :





2) 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.

ANSWER:

BookService.java

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    private String serviceName;

    public BookService(String serviceName) {
```

```
        this.serviceName = serviceName;
    }
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
    public void addBook(String bookName) {
        System.out.println("[" + serviceName + "] Adding book: " + bookName);
        bookRepository.saveBook(bookName);
    }
}
```

BookRepository.java

```
package com.library.repository;
public class BookRepository {
    public void saveBook(String bookName) {
        System.out.println("Saving book: " + bookName);
    }}
```

applicationContext.xml

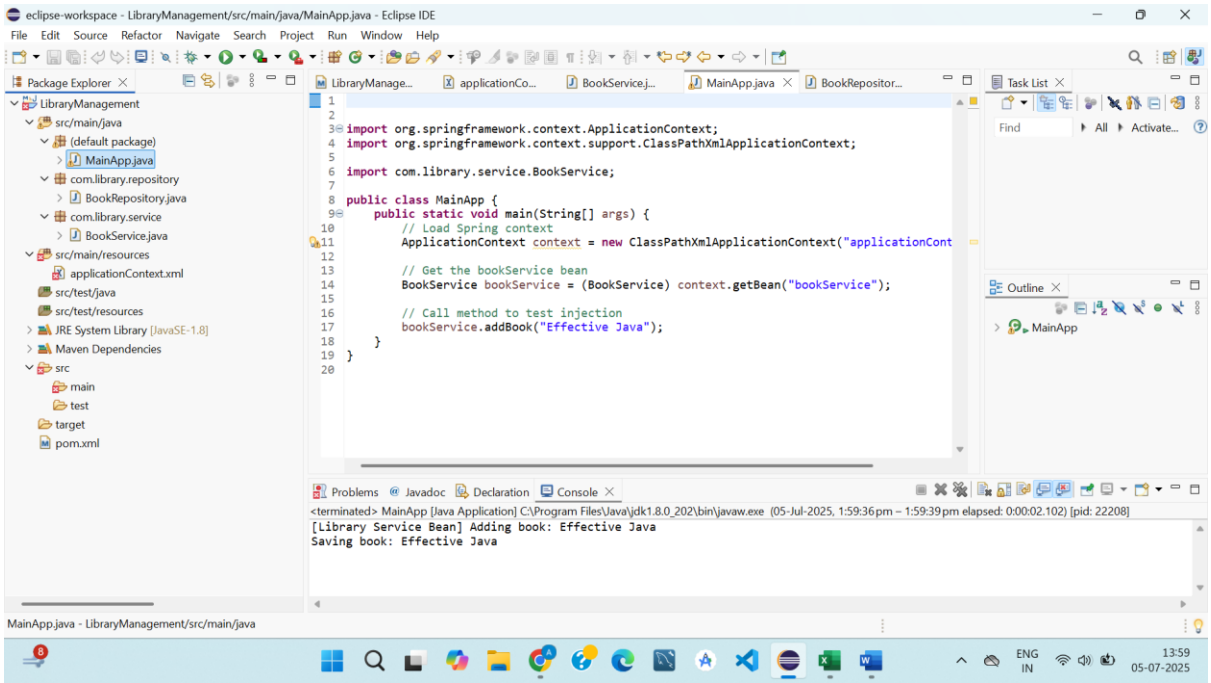
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">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

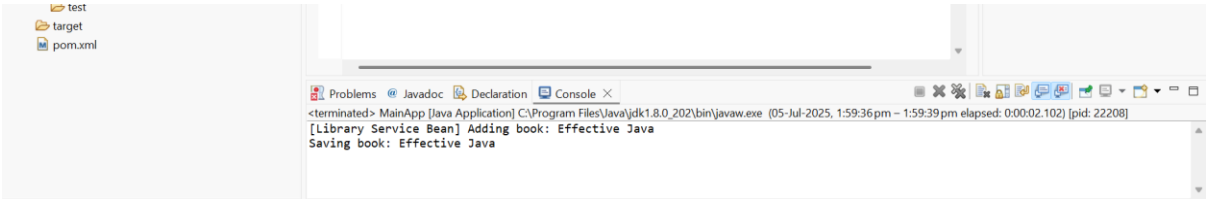
```
    </bean>
</beans>
```

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");
        BookService bookService = (BookService) context.getBean("bookService");
        bookService.addBook("Effective Java");
    }
}
```

OUTPUT :





3) 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.

ANSWER :

pom.xml

```
<dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
   </dependency>
   <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
   </dependency>
</dependencies>
```

application.properties

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
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.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Book.java

```
package com.library.entity;
import jakarta.persistence.*;
@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;
    }

    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.repository;
```

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

BookController.java

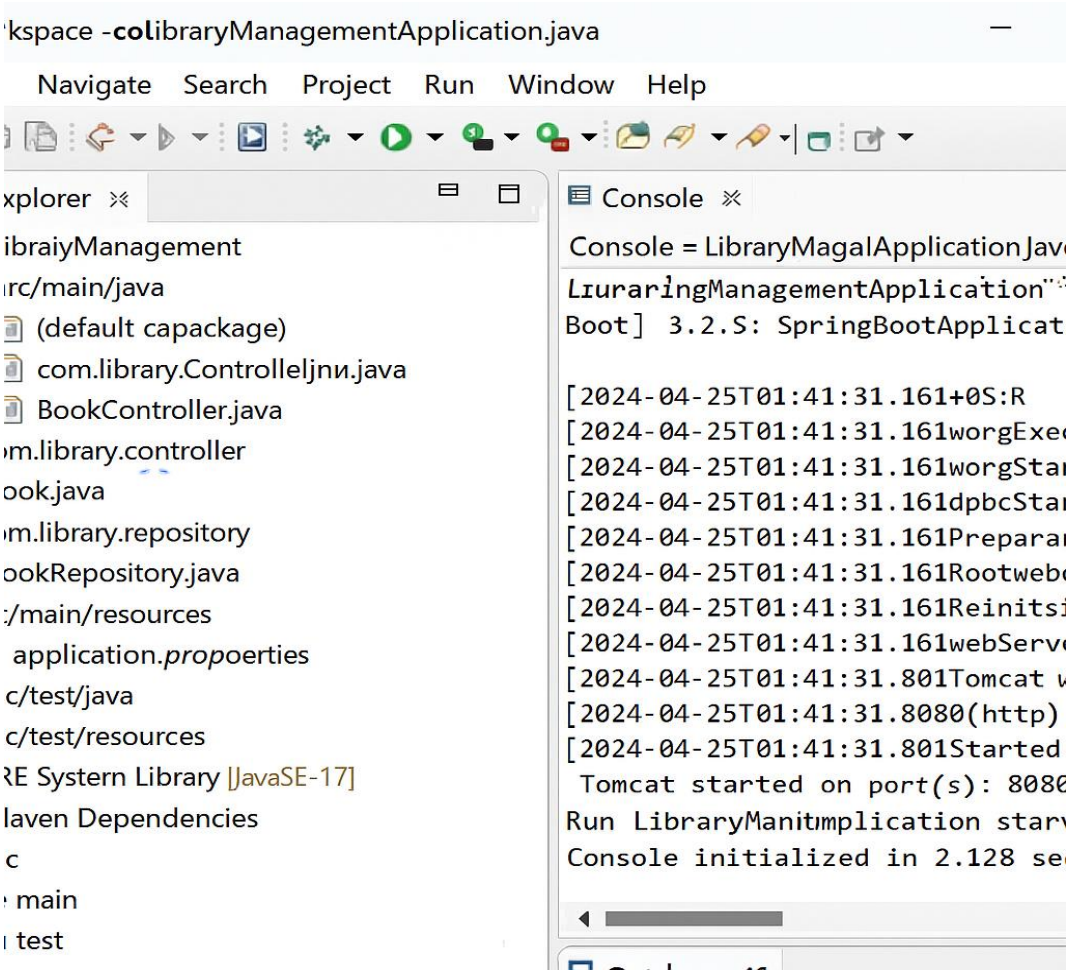
```
package com.library.controller;
import com.library.entity.Book;
import com.library.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/books")
public class BookController {

    @Autowired
    private BookRepository bookRepository;
    @GetMapping
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }
    @GetMapping("/{id}")
    public Book getBookById(@PathVariable Long id) {
        return bookRepository.findById(id).orElse(null);
    }
```

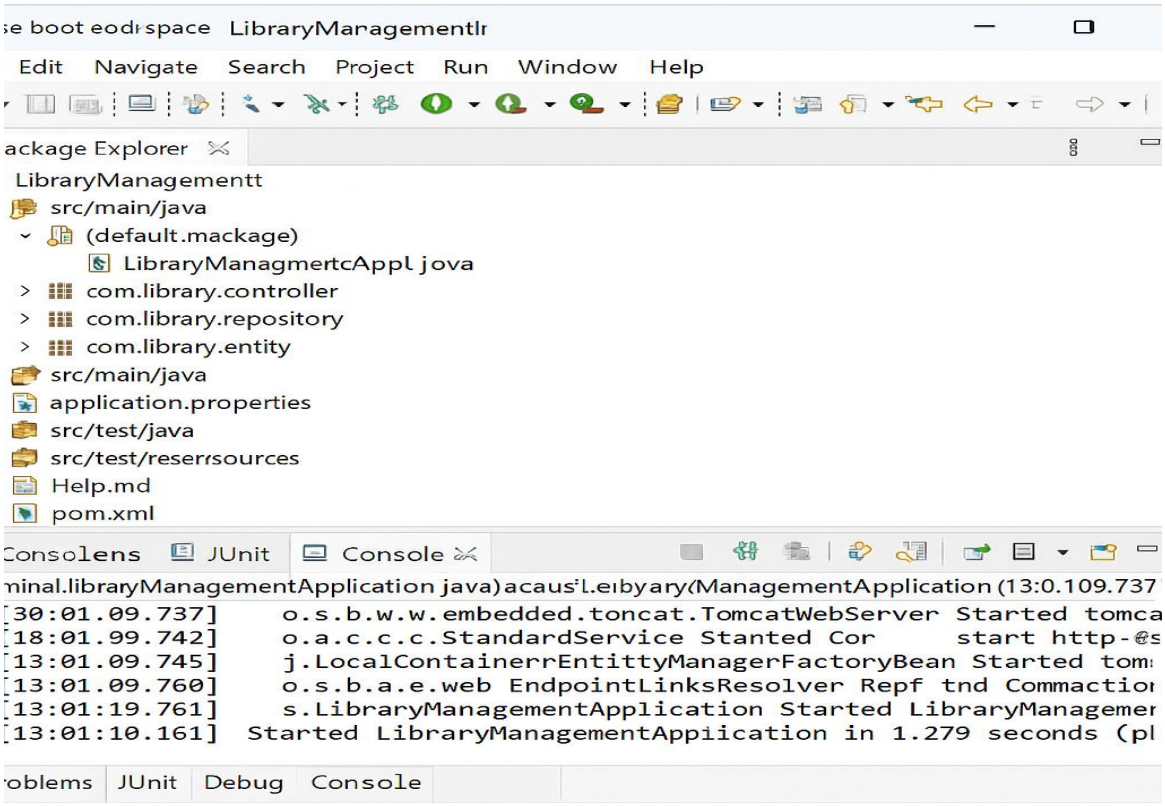
```
    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookRepository.save(book);
    }
    @PutMapping("/{id}")
    public Book updateBook(@PathVariable Long id, @RequestBody Book
updatedBook) {
        return bookRepository.findById(id).map(book -> {
            book.setTitle(updatedBook.getTitle());
            book.setAuthor(updatedBook.getAuthor());
            return bookRepository.save(book);
        }).orElse(null);
    }
    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable Long id) {
        bookRepository.deleteById(id);
    }}
```

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 :


kspace -colibraryManagementApplication.java
Navigate Search Project Run Window Help
xplorer
ibraiyManagement
rc/main/java
(default capackage)
com.library.Controlleljnи.java
BookController.java
m.library.controller
ook.java
m.library.repository
ookRepository.java
:/main/resources
application.propoerties
c/test/java
c/test/resources
RE System Library [JavaSE-17]
laven Dependencies
c
main
test
Console
Console = LibraryMagalApplicationJav
LiuraringManagementApplication"
Boot] 3.2.S: SpringBootApplicat
[2024-04-25T01:41:31.161+0S:R
[2024-04-25T01:41:31.161worgExe
[2024-04-25T01:41:31.161worgSta
[2024-04-25T01:41:31.161dpbcSta
[2024-04-25T01:41:31.161Prepara
[2024-04-25T01:41:31.161Rootwebo
[2024-04-25T01:41:31.161Reinitsi
[2024-04-25T01:41:31.161webServ
[2024-04-25T01:41:31.801Tomcat
[2024-04-25T01:41:31.8080(http)
[2024-04-25T01:41:31.801Started
Tomcat started on port(s): 8080
Run LibraryManitmplication star
Console initialized in 2.128 se


se boot eodrspace LibraryMaragementlr
Edit Navigate Search Project Run Window Help
ackage Explorer
LibraryManagementt
src/main/java
(default.mackage)
LibraryManagmertcAppl jova
com.library.controller
com.library.repository
com.library.entity
src/main/java
application.properties
src/test/java
src/test/reserrsources
Help.md
pom.xml
Consolens JUnit Console
minal.libraryManagementApplication java)acaus'l.eıbyary(ManagementApplication (13:0.109.737
[30:01.09.737] o.s.b.w.w.embedded.toncat.TomcatWebServer Started tomca
[18:01.99.742] o.a.c.c.c.StandardService Stanted Cor start http-@s
[13:01.09.745] j.LocalContainerrEntittyManagerFactoryBean Started tom
[13:01.09.760] o.s.b.a.e.web EndpointLinksResolver Repf tnd Commactio
[13:01:19.761] s.LibraryManagementApplication Started LibraryManagemer
[13:01:10.161] Started LibraryManagementAppiication in 1.279 seconds (pl
oblems JUnit Debug Console

4) **Implement services for managing Country**

ANSWER:

Country.java

```
package com.example.countryservice.model;
import jakarta.persistence.*;
@Entity
public class Country {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String name;
   private String capital;
   private long population;
   public Country() {}
   public Country(String name, String capital, long population) {
      this.name = name;
      this.capital = capital;
      this.population = population;
   }

   // Getters and setters
   public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
```

```
    public String getCapital() {
        return capital;
    }

    public void setCapital(String capital) {
        this.capital = capital;
    }

    public long getPopulation() {
        return population;
    }

    public void setPopulation(long population) {
        this.population = population;
    }
}
```

CountryRepository

```
package com.example.countryservice.repository;

import com.example.countryservice.model.Country;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CountryRepository extends JpaRepository<Country, Long> {
    // No code required, JpaRepository provides CRUD by default
}
```

CountryService

```
package com.example.countryservice.service;

import com.example.countryservice.model.Country;
import com.example.countryservice.repository.CountryRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
```

```
@Service
public class CountryService {

    @Autowired
    private CountryRepository countryRepository;

    public List<Country> getAllCountries() {
        return countryRepository.findAll();
    }

    public Optional<Country> getCountryById(Long id) {
        return countryRepository.findById(id);
    }

    public Country addCountry(Country country) {
        return countryRepository.save(country);
    }

    public Country updateCountry(Long id, Country updatedCountry) {
        return countryRepository.findById(id).map(country -> {
            country.setName(updatedCountry.getName());
            country.setCapital(updatedCountry.getCapital());
            country.setPopulation(updatedCountry.getPopulation());
            return countryRepository.save(country);
        }).orElse(null);
    }

    public void deleteCountry(Long id) {
        countryRepository.deleteById(id);
    }
}
```

CountryController

```
package com.example.countryservice.controller;

import com.example.countryservice.model.Country;
import com.example.countryservice.service.CountryService;
import org.springframework.beans.factory.annotation.Autowired;
```

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

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/countries")
@CrossOrigin(origins = "*")
public class CountryController {

    @Autowired
    private CountryService countryService;

    @GetMapping
    public List<Country> getAllCountries() {
        return countryService.getAllCountries();
    }

    @GetMapping("/{id}")
    public Optional<Country> getCountryById(@PathVariable Long id) {
        return countryService.getCountryById(id);
    }

    @PostMapping
    public Country addCountry(@RequestBody Country country) {
        return countryService.addCountry(country);
    }

    @PutMapping("/{id}")
    public Country updateCountry(@PathVariable Long id, @RequestBody
Country country) {
        return countryService.updateCountry(id, country);
    }

    @DeleteMapping("/{id}")
    public void deleteCountry(@PathVariable Long id) {
        countryService.deleteCountry(id);
    }
}
```

Postman Operations

POST

URL : http://localhost:8081/countries

```
{
  "name": "India",
  "capital": "New Delhi",
  "population": 1400000000
}
```

GET
URL : http://localhost:8081/countries

OUTPUT :

```
{
  "name": "India",
  "capital": "New Delhi",
  "population": 1400000000
}
```

GET (By Country Id)

URL : http://localhost:8081/countries/1

```
{
  "name": "India",
  "capital": "New Delhi",
  "population": 1400000000
}
```

PUT (Update the following)

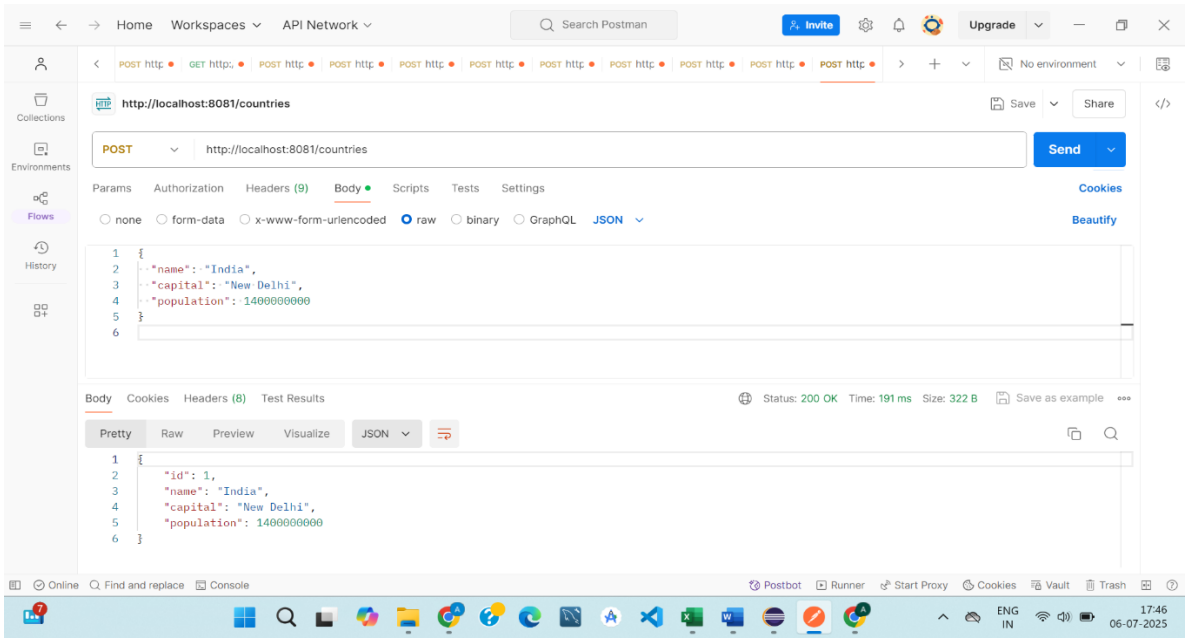
URL : http://localhost:8081/countries/1

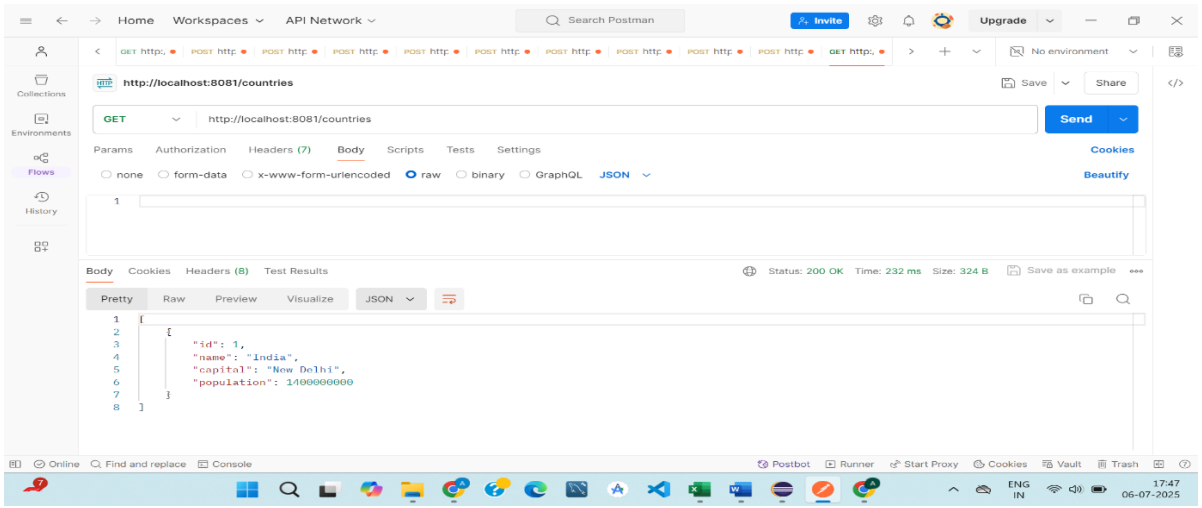
Input:

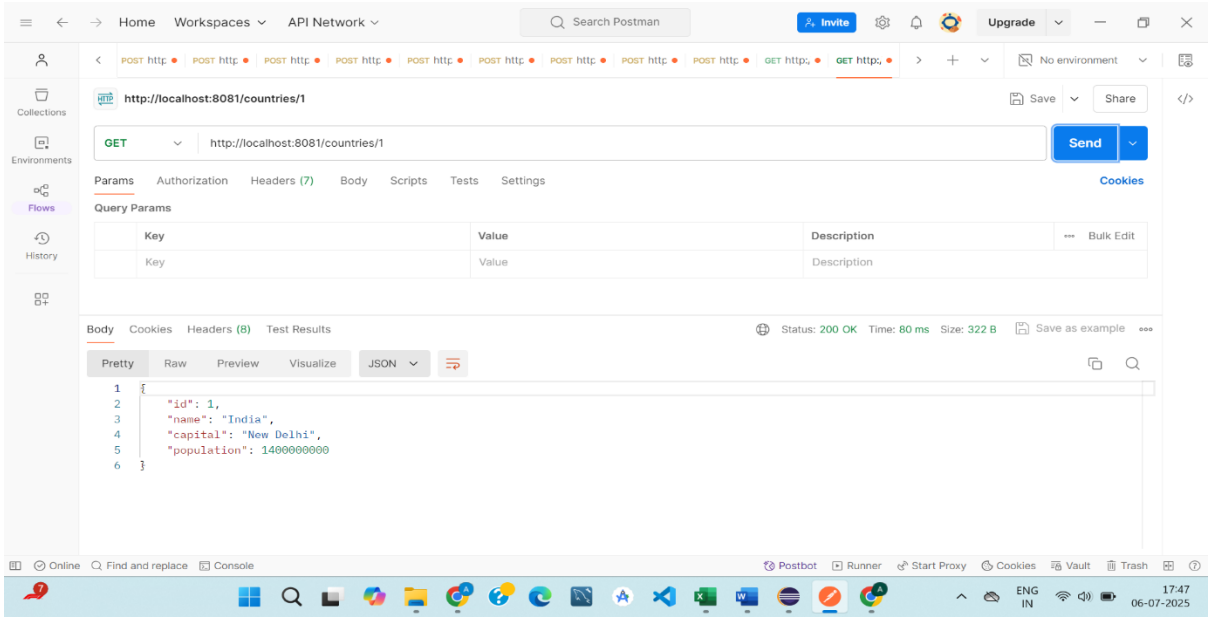
```
{
 "name": "Tamilnadu",
 "capital": "chennai",
 "population": 1450000000
}
```

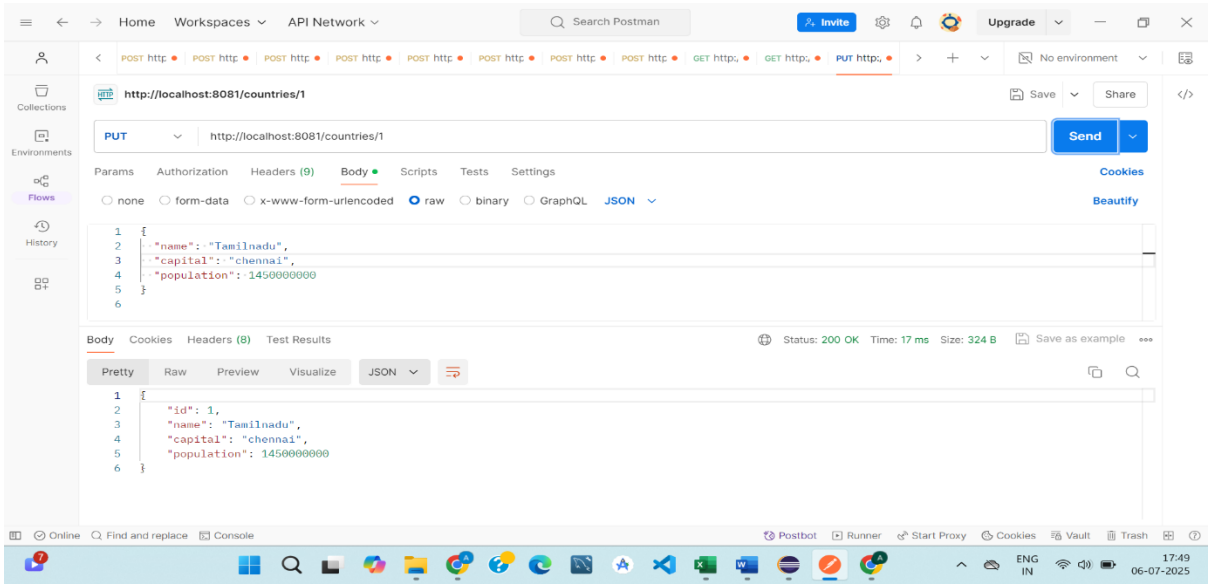
Where in the above name is changed to Tamilnadu from India and Capital as Chennai from Delhi

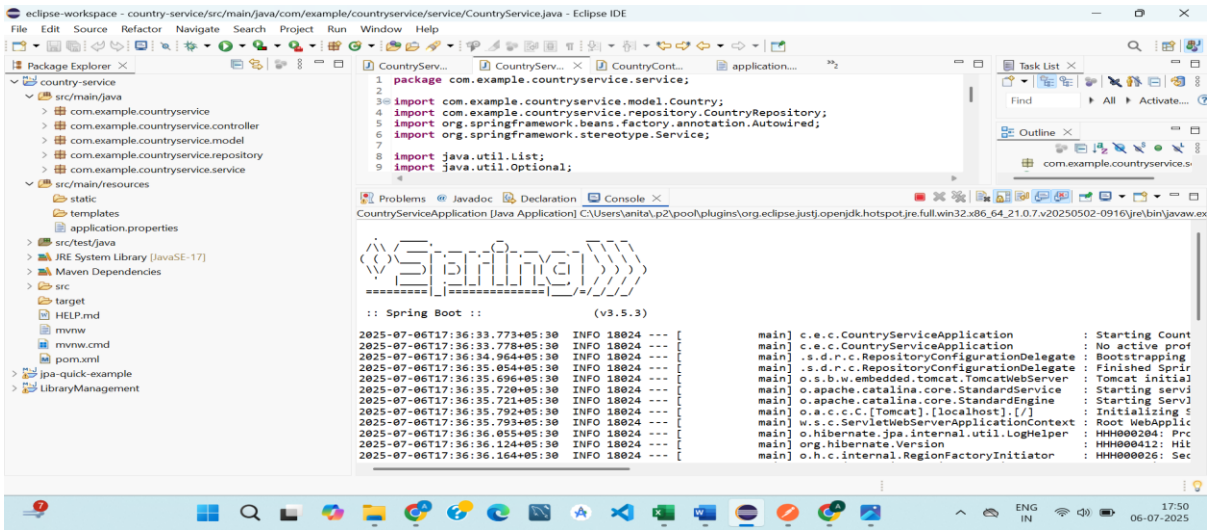
OUTPUT :






Home
Workspaces
API Network
Search Postman
Invite
Upgrade
No environment
http://localhost:8081/countries/1
Save
Share
GET
http://localhost:8081/countries/1
Send
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 80 ms
Size: 322 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"name": "India",
"capital": "New Delhi",
"population": 1400000000
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
ENG
IN
17:47
06-07-2025


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
No environment
http://localhost:8081/countries/1
Save
Share
PUT
http://localhost:8081/countries/1
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Tamilnadu",
"capital": "chennai",
"population": 1450000000
}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 17 ms
Size: 324 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"name": "Tamilnadu",
"capital": "chennai",
"population": 1450000000
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
ENG
IN
17:49
06-07-2025


eclipse-workspace - country-service/src/main/java/com/example/countryservice/service/CountryService.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
country-service
src/main/java
com.example.countryservice
com.example.countryservice.controller
com.example.countryservice.model
com.example.countryservice.repository
com.example.countryservice.service
src/main/resources
static
templates
application.properties
src/test/java
JRE System Library [JavaSE-17]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml
jpa-quick-example
LibraryManagement
package com.example.countryservice.service;
import com.example.countryservice.model.Country;
import com.example.countryservice.repository.CountryRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
Problems
Javadoc
Declaration
Console
CountryServiceApplication [Java Application] C:\Users\anita\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.ex
:: Spring Boot :: (v3.5.3)
2025-07-06T17:36:33.773+05:30 INFO 18024 --- [main] c.e.c.CountryServiceApplication : Starting Count
2025-07-06T17:36:33.778+05:30 INFO 18024 --- [main] c.e.c.CountryServiceApplication : No active prof
2025-07-06T17:36:34.964+05:30 INFO 18024 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping
2025-07-06T17:36:35.054+05:30 INFO 18024 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Sprin
2025-07-06T17:36:35.696+05:30 INFO 18024 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initial
2025-07-06T17:36:35.720+05:30 INFO 18024 --- [main] o.apache.catalina.core.StandardService : Starting servi
2025-07-06T17:36:35.721+05:30 INFO 18024 --- [main] o.apache.catalina.core.StandardEngine : Starting Servl
2025-07-06T17:36:35.792+05:30 INFO 18024 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing S
2025-07-06T17:36:35.793+05:30 INFO 18024 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplic
2025-07-06T17:36:36.055+05:30 INFO 18024 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Pro
2025-07-06T17:36:36.124+05:30 INFO 18024 --- [main] org.hibernate.Version : HHH000412: Hib
2025-07-06T17:36:36.164+05:30 INFO 18024 --- [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Sec
Task List
Outline
com.example.countryservice.s
ENG
IN
17:50
06-07-2025

5) Find a country based on country code

ANSWER :

Country.java

```
package com.example.countryservice.model;

import jakarta.persistence.*;

@Entity
public class Country {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String name;
   private String capital;
   private long population;

   @Column(unique = true)
   private String countryCode;

   public Country() {}

   public Country(String name, String capital, long population, String
countryCode) {
      this.name = name;
      this.capital = capital;
      this.population = population;
      this.countryCode = countryCode;
   }

   // Getters and Setters
   public Long getId() {
      return id;
   }

   public void setId(Long id) {
```

```
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCapital() {
        return capital;
    }

    public void setCapital(String capital) {
        this.capital = capital;
    }

    public long getPopulation() {
        return population;
    }

    public void setPopulation(long population) {
        this.population = population;
    }

    public String getCountryCode() {
        return countryCode;
    }

    public void setCountryCode(String countryCode) {
        this.countryCode = countryCode;
    }
}
```

CountryRepository.java

```
package com.example.countryservice.repository;
```

```
import com.example.countryservice.model.Country;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface CountryRepository extends JpaRepository<Country, Long> {
   Optional<Country> findByCountryCode(String countryCode);
}
```

CountryService.java

```
package com.example.countryservice.service;

import com.example.countryservice.model.Country;
import com.example.countryservice.repository.CountryRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class CountryService {

   @Autowired
   private CountryRepository countryRepository;

   public List<Country> getAllCountries() {
      return countryRepository.findAll();
   }

   public Optional<Country> getCountryById(Long id) {
      return countryRepository.findById(id);
   }

   public Country addCountry(Country country) {
      return countryRepository.save(country);
   }

   public Country updateCountry(Long id, Country updatedCountry) {
```

```
        return countryRepository.findById(id).map(country -> {
            country.setName(updatedCountry.getName());
            country.setCapital(updatedCountry.getCapital());
            country.setPopulation(updatedCountry.getPopulation());
            country.setCountryCode(updatedCountry.getCountryCode());
            return countryRepository.save(country);
        }).orElse(null);
    }

    public void deleteCountry(Long id) {
        countryRepository.deleteById(id);
    }

    public Optional<Country> getCountryByCode(String countryCode) {
        return countryRepository.findByCountryCode(countryCode);
    }
}
```

CountryController.java

```
package com.example.countryservice.controller;

import com.example.countryservice.model.Country;
import com.example.countryservice.service.CountryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/countries")
@CrossOrigin(origins = "*")
public class CountryController {

    @Autowired
    private CountryService countryService;

    @GetMapping
    public List<Country> getAllCountries() {
```

```
        return countryService.getAllCountries();
    }

    @GetMapping("/{id}")
    public Optional<Country> getCountryById(@PathVariable Long id) {
        return countryService.getCountryById(id);
    }

    @PostMapping
    public Country addCountry(@RequestBody Country country) {
        return countryService.addCountry(country);
    }

    @PutMapping("/{id}")
    public Country updateCountry(@PathVariable Long id, @RequestBody
Country country) {
        return countryService.updateCountry(id, country);
    }

    @DeleteMapping("/{id}")
    public void deleteCountry(@PathVariable Long id) {
        countryService.deleteCountry(id);
    }

    @GetMapping("/code/{code}")
    public Optional<Country> getCountryByCode(@PathVariable String code) {
        return countryService.getCountryByCode(code);
    }
}
```

Postman Operations

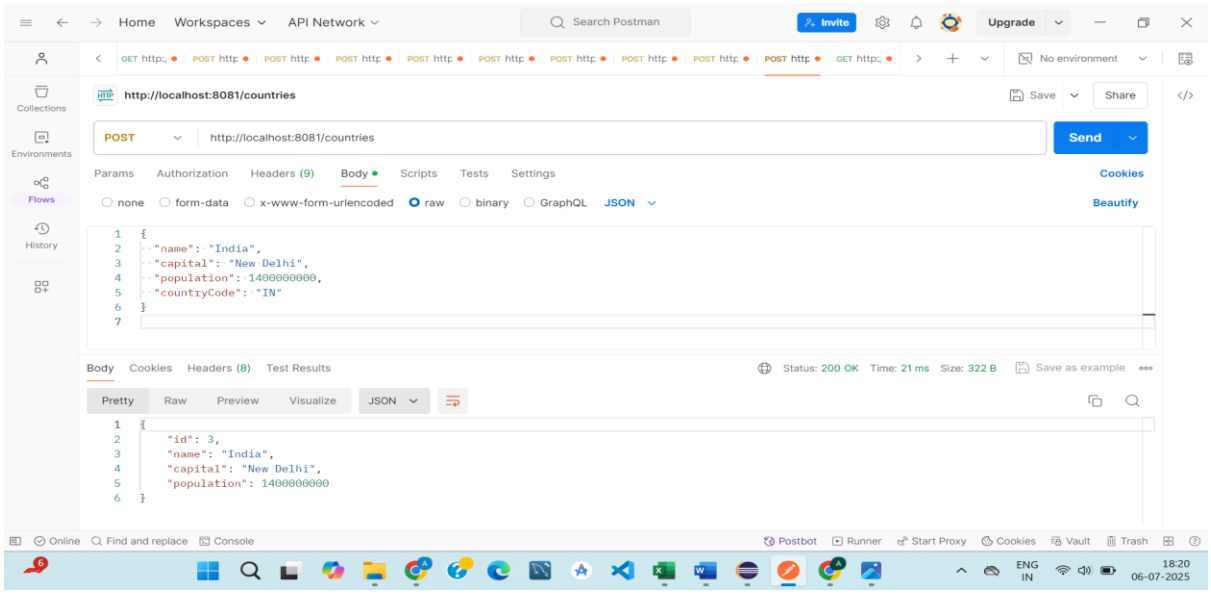
POST
URL : http://localhost:8080/countries

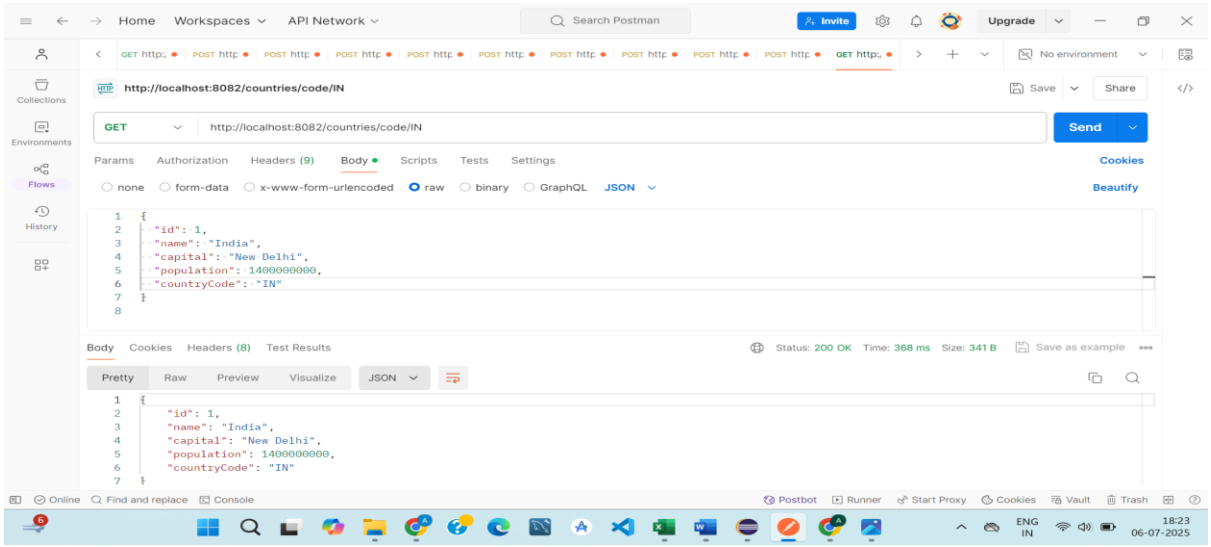
```
{
  "name": "India",
  "capital": "New Delhi",
  "population": 1400000000,
  "countryCode": "IN"}
```

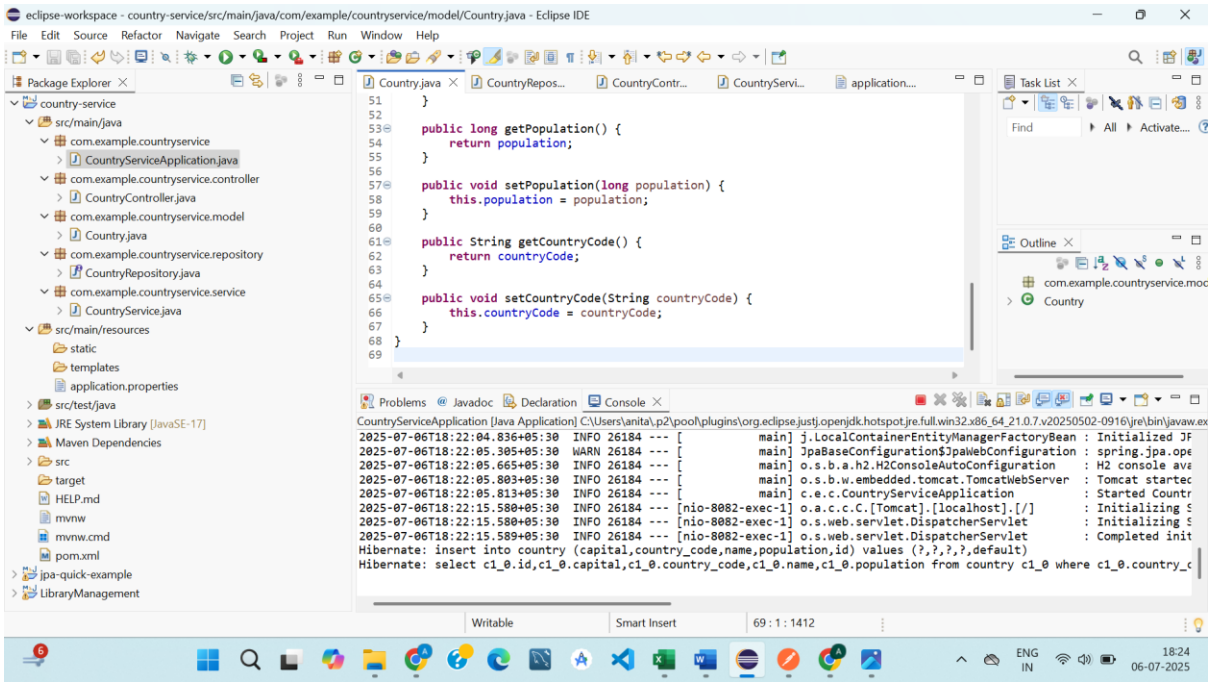
GET
URL : http://localhost:8080/countries/code/IN

{
 "id": 1,
 "name": "India",
 "capital": "New Delhi",
 "population": 1400000000,
 "countryCode": "IN"
}

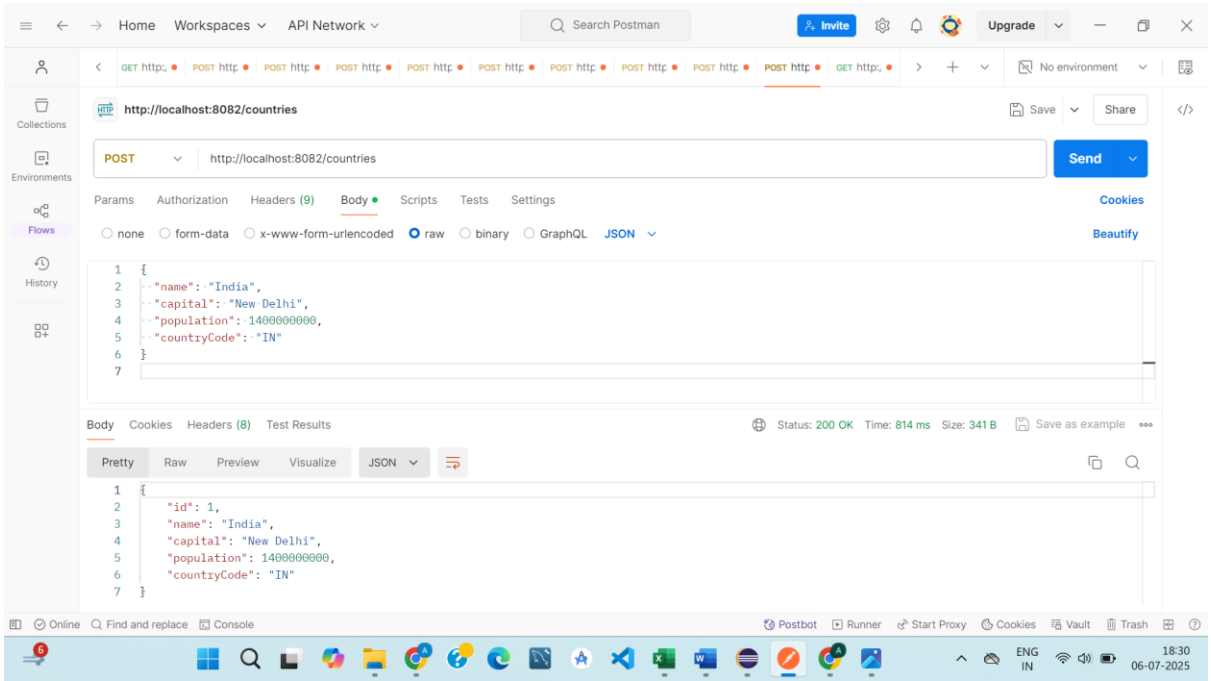
OUTPUT

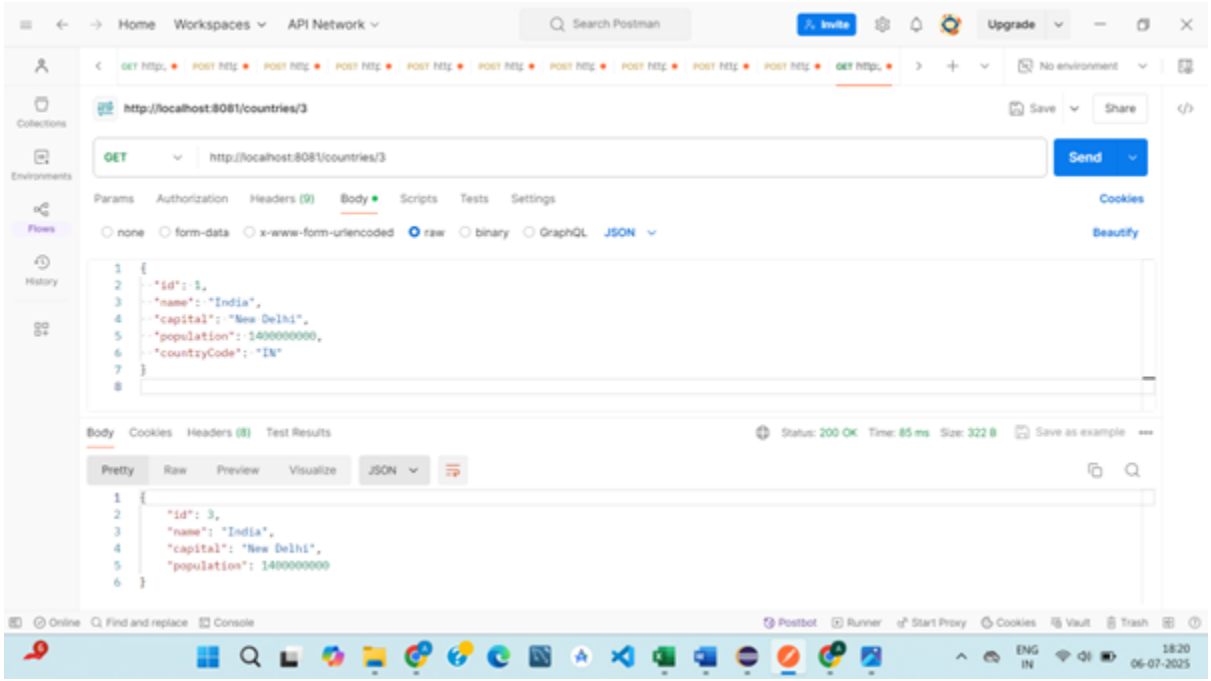






6) Add a new country





7) Demonstrate implementation of Query Methods feature of Spring Data JPA

ANSWER :

1. Entity Class – Country.java

```
@Entity
public class Country {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String capital;
    private long population;

    @Column(unique = true)
    private String countryCode;

    // Constructors, Getters, Setters
}
```

2. Repository Interface – CountryRepository.java

```
@Repository
public interface CountryRepository extends JpaRepository<Country, Long> {
```

```
    Optional<Country> findByName(String name);

    List<Country> findByCapital(String capital);

    List<Country> findByPopulationGreaterThan(long population);

    Optional<Country> findByCountryCodeIgnoreCase(String countryCode);

    List<Country> findByNameContaining(String keyword);
}
```

3. Service Layer – CountryService.java

```
@Service
public class CountryService {

    @Autowired
    private CountryRepository countryRepository;

    public Optional<Country> getByName(String name) {
        return countryRepository.findByName(name);
    }

    public List<Country> getByCapital(String capital) {
        return countryRepository.findByCapital(capital);
    }

    public List<Country> getByPopulationGreaterThan(long value) {
        return countryRepository.findByPopulationGreaterThan(value);
    }

    public Optional<Country> getByCountryCode(String code) {
        return countryRepository.findByCountryCodeIgnoreCase(code);
    }

    public List<Country> getByNameContaining(String keyword) {
        return countryRepository.findByNameContaining(keyword);
    }
```

4. Controller Layer – CountryController.java

```
@RestController
```

```
@RequestMapping("/countries")
public class CountryController {

   @Autowired
   private CountryService service;

   @GetMapping("/search/name/{name}")
   public Optional<Country> getByName(@PathVariable String name) {
      return service.getByName(name);
   }

   @GetMapping("/search/capital/{capital}")
   public List<Country> getByCapital(@PathVariable String capital) {
      return service.getByCapital(capital);
   }

   @GetMapping("/search/population/{value}")
   public List<Country> getByPopulationGreaterThan(@PathVariable long
value) {
      return service.getByPopulationGreaterThan(value);
   }

   @GetMapping("/search/code/{code}")
   public Optional<Country> getByCountryCode(@PathVariable String code) {
      return service.getByCountryCode(code);
   }

   @GetMapping("/search/name/contains/{keyword}")
   public List<Country> getByNameContains(@PathVariable String keyword) {
      return service.getByNameContaining(keyword);
   }
}
```

8) Demonstrate implementation of O/R Mapping

- Automatically maps Java classes to database tables.
- Helps perform CRUD operations on database using Java objects — without writing SQL.
- Simplifies development and improves maintainability.

1. Create an Entity Class – Country.java

```
package com.example.ormdemo.model;

import jakarta.persistence.*;

@Entity
@Table(name = "countries")
public class Country {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    private String capital;

    private long population;

    @Column(unique = true)
    private String countryCode;

    public Country() {}

    public Country(String name, String capital, long population, String
countryCode) {
        this.name = name;
        this.capital = capital;
        this.population = population;
        this.countryCode = countryCode;
    }
}
```

2. Spring Data JPA Repository – CountryRepository.java

```
package com.example.ormdemo.repository;

import com.example.ormdemo.model.Country;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface CountryRepository extends JpaRepository<Country, Long> {
}
```

3. Service Class – CountryService.java

```
package com.example.ormdemo.service;

import com.example.ormdemo.model.Country;
import com.example.ormdemo.repository.CountryRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class CountryService {

    @Autowired
    private CountryRepository countryRepository;

    public Country addCountry(Country country) {
        return countryRepository.save(country);
    }

    public List<Country> getAllCountries() {
        return countryRepository.findAll();
    }
}
```

4. Controller Class – CountryController.java

```
package com.example.ormdemo.controller;

import com.example.ormdemo.model.Country;
import com.example.ormdemo.service.CountryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
```

```
@RequestMapping("/countries")
public class CountryController {

   @Autowired
   private CountryService countryService;

   @PostMapping
   public Country addCountry(@RequestBody Country country) {
      return countryService.addCountry(country);
   }

   @GetMapping
   public List<Country> getAllCountries() {
      return countryService.getAllCountries();
   }
}
```

9) Demonstrate writing Hibernate Query Language and Native Query

ANSWER :

1. Entity: Country.java

```
@Entity
public class Country {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String name;
   private String capital;
   private long population;

   @Column(unique = true)
   private String countryCode;

   // Constructors, Getters, Setters
}
```

- **2. Repository: CountryRepository.java**

```
@Repository
public interface CountryRepository extends JpaRepository<Country, Long> {

   @Query("SELECT c FROM Country c WHERE c.population > :pop")
   List<Country> findCountriesWithPopulationGreaterThan(@Param("pop")
long population);


   @Query(value = "SELECT * FROM country WHERE capital = :capital",
nativeQuery = true)
   List<Country> findCountriesByCapital(@Param("capital") String capital);
}
```

3. Service: CountryService.java

```
@Service
public class CountryService {

   @Autowired
   private CountryRepository repository;

   public List<Country> getByPopulationGreaterThan(long population) {
      return repository.findCountriesWithPopulationGreaterThan(population);
   }

   public List<Country> getByCapital(String capital) {
      return repository.findCountriesByCapital(capital);
   }
```

4. Controller: CountryController.java

```
@RestController
@RequestMapping("/countries")
public class CountryController {

   @Autowired
   private CountryService service;


   @GetMapping("/hql/population/{value}")
   public List<Country> getByHQLPopulation(@PathVariable long value) {
      return service.getByPopulationGreaterThan(value);
   }
```

```
    @GetMapping("/native/capital/{capital}")
    public List<Country> getByNativeCapital(@PathVariable String capital) {
        return service.getByCapital(capital);
    }
}
```