**Hands On 1 : Create a Eclipse Project using Spring Initializr**

**Country.java**

package com.cognizant.orm\_learn.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Country {

@Id

private String code;

private String name;

// Getters and Setters

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**CountryRepository.java**

package com.cognizant.orm\_learn.repository;

import com.cognizant.orm\_learn.model.Country;

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

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java**

package com.cognizant.orm\_learn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**Application.properties:**

spring.application.name=orm-learn

# Logging

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger**{25}** %25M %4L %m%n

# DB Config

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=Poorani123@

# Hibernate

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**OrmLearnApplication.java**

package com.cognizant.orm\_learn;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.service.CountryService;

@SpringBootApplication

public class OrmLearnApplication {

private static CountryService *countryService*;

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);

public static void main(String[] args) {

ApplicationContext context = SpringApplication.*run*(OrmLearnApplication.class, args);

*LOGGER*.info("Inside main");

*countryService* = context.getBean(CountryService.class);

*testGetAllCountries*();

}

private static void testGetAllCountries() {

*LOGGER*.info("Start");

List<Country> countries = *countryService*.getAllCountries();

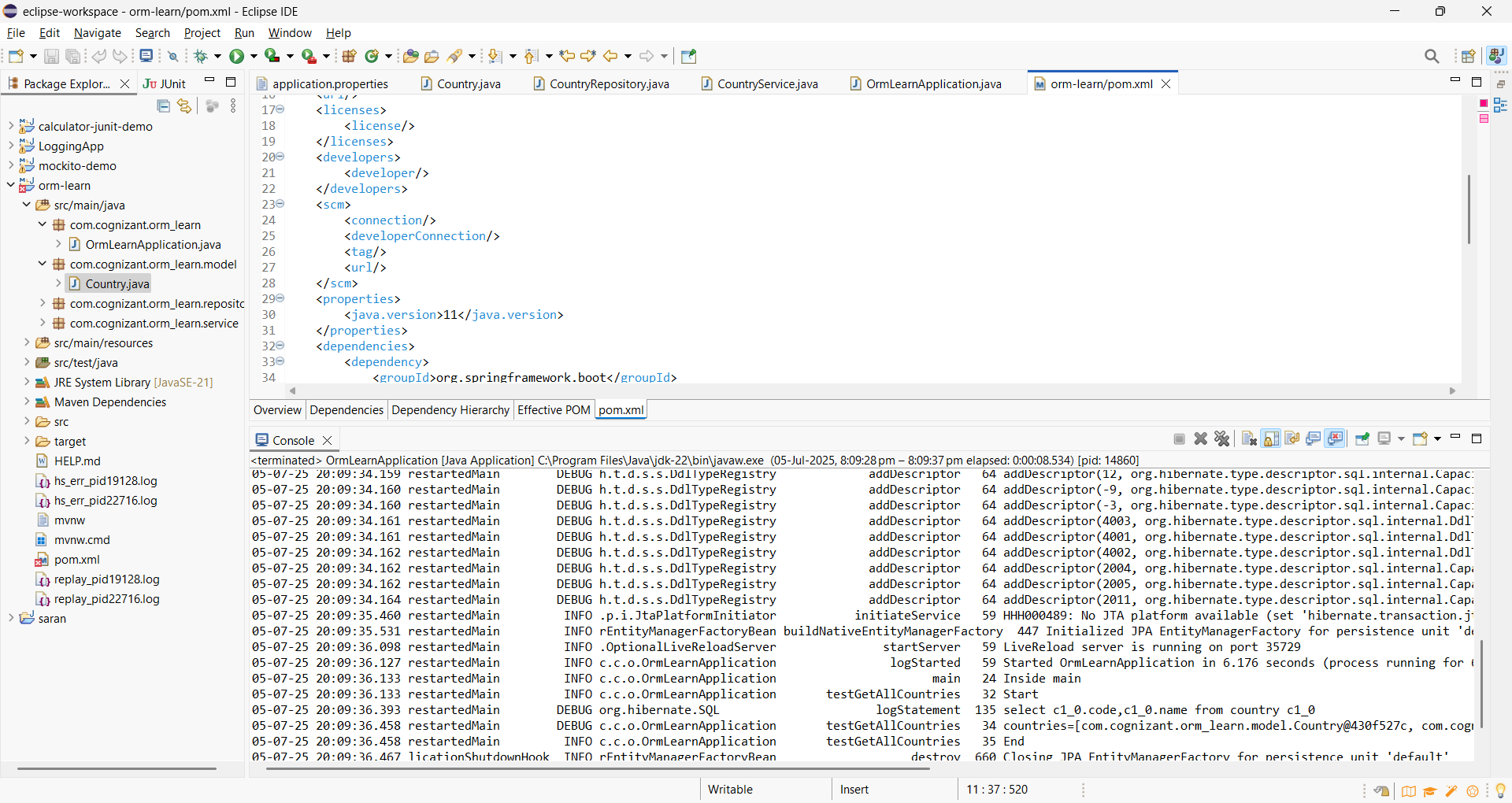
*LOGGER*.debug("countries={}", countries);

*LOGGER*.info("End");

}

}

**Output:**



**Hands on 2 : Hibernate XML Config implementation walk through**

1. **Explain how object to relational database mapping done in hibernate xml configuration file**

* Hibernate uses XML files (like hibernate.cfg.xml and mapping files .hbm.xml) to map Java classes (objects) to database tables (relational).
* hibernate.cfg.xml configures connection details (DB URL, username, password, dialect, driver).
* Mapping files (\*.hbm.xml) describe how Java class properties map to database table columns.

**Example of Mapping XML (Country.hbm.xml):**

<class name="com.example.Country" table="COUNTRY">

<id name="code" column="COUNTRY\_CODE"/>

<property name="name" column="COUNTRY\_NAME"/>

</class>

1. **Explain about following aspects of implementing the end to end operations in Hibernate:**
   * **SessionFactory**

* It is a factory for Session objects.
* Created once during application startup.
* It is thread-safe and heavyweight, so you should only have one per database.
* It is created using the Configuration object by reading hibernate.cfg.xml.

Example:

SessionFactory factory = new Configuration().configure().buildSessionFactory();

* + **Session**
* A Session represents a **single-threaded unit of work** with the database.
* It is used to create, read, update, and delete persistent objects.
* Sessions are **not thread-safe** and should be closed after use.

Example:

Session session = factory.openSession();

* + **Transaction**
* A transaction represents a **unit of work** in the database.
* Helps ensure **data consistency** by allowing commit or rollback.
* It is used to **group multiple operations** into a single atomic unit.

Example:

Transaction tx = session.beginTransaction();

* + **beginTransaction()**
* Begins a new database transaction.
* Should always be called before performing any write operation (save, update, delete).

Example:

Transaction tx = session.beginTransaction();

* + **commit()**
* Commits the current transaction to the database.
* Saves all changes made during the session.

Example:

tx.commit();

* + **rollback()**
* Rolls back the current transaction if any error occurs.
* Ensures that no partial or invalid data is stored.

Example:

tx.rollback();

* + **session.save()**
* Saves a Java object (entity) to the database.
* The object is inserted as a new row in the mapped table.

Example:

Country country = new Country("IN", "India");

session.save(country);

* + **session.createQuery().list()**
* Executes a **Hibernate Query Language (HQL)** query and returns a list of results.
* It is used for **select operations** involving multiple records.

Example:

List<Country> countries = session.createQuery("from Country").list();

* + **session.get()**
* Retrieves an object from the database using its primary key.
* Returns null if no matching row is found.

Example:

Country country = session.get(Country.class, "IN");

* + **session.delete()**
* Deletes a record (row) from the database that corresponds to the given object.

Example:

session.delete(country);

**Hands on 3 : Hibernate Annotation Config implementation walk through**

* **Explain how object to relational database mapping done in persistence class file Employee**

In the annotation-based approach, the Employee class uses **JPA annotations** to define mappings between class members and database table columns.

**Example:**

@Entity

@Table(name = "EMPLOYEE")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private int id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "salary")

private int salary;

// Getters and Setters

}

This class tells Hibernate how to persist Employee objects in the EMPLOYEE table.

* **Explain about following aspects of implementing the end to end operations in Hibernate:**
  + **@Entity**
* Marks the class as a **persistent entity** (a table will be mapped to this class).
* Required for Hibernate to recognize the class as a database-mapped object.

**Example:**

@Entity

* + **@Table**
* Specifies the name of the **table** in the database that this entity maps to.
* If omitted, the table name defaults to the class name.

**Example:**

@Table(name = "EMPLOYEE")

* + **@Id**
* Denotes the **primary key** of the entity.
* Required for every entity to uniquely identify rows.

**Example:**

@Id

* + **@GeneratedValue**
* Specifies the **auto-generation strategy** for the primary key.
* Common strategies: AUTO, IDENTITY, SEQUENCE, TABLE.

**Example:**

@GeneratedValue(strategy = GenerationType.IDENTITY)

* + **@Column**
* Maps a Java field to a specific **column** in the table.
* You can define name, length, nullable, unique, etc.

**Example:**

@Column(name = "first\_name")

* + **Hibernate Configuration (hibernate.cfg.xml)**

The hibernate.cfg.xml is used to configure Hibernate runtime settings like database connection, dialect, and mapping resources.

**Example:**

<hibernate-configuration>

<session-factory>

<!-- JDBC Configuration -->

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/your\_db</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">password</property>

<!-- Hibernate Properties -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="show\_sql">true</property>

<!-- Mapping Class -->

<mapping class="com.example.Employee"/>

</session-factory>

</hibernate-configuration>

* + - **Dialect**
* Specifies the SQL dialect Hibernate should use.
* Example: org.hibernate.dialect.MySQLDialect, OracleDialect, etc.
* This helps Hibernate generate database-specific SQL.

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

* + - **Driver**
* The fully qualified Java class name of the **JDBC driver**.

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

* + - **Connection URL**
* JDBC URL to connect to the database.

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/your\_db</property>

* + - **Username**
* The **username** for the database.

<property name="hibernate.connection.username">root</property>

* + - **Password**
* The password for the database user.

<property name="hibernate.connection.password">password</property>

Annotation-based configuration in Hibernate simplifies mapping and reduces the need for separate XML files. The @Entity, @Table, @Id, and other annotations directly link Java classes and fields to their corresponding database tables and columns. Together with hibernate.cfg.xml, this allows seamless object-relational integration for modern Java applications.

**Hands on 4 : Difference between JPA, Hibernate and Spring Data JPA**   
**Java Persistence API (JPA)**

1. Specification for object-relational mapping (ORM) in Java.
2. Defines interfaces, not implementations (e.g., EntityManager, @Entity, etc.).
3. Hibernate is one of its most popular implementations.

**Hibernate**

1. A concrete implementation of the JPA specification.
2. Also provides additional features beyond JPA (like caching, lazy loading strategies, etc.).
3. Manages DB operations via Session, Transaction, etc.

**Example (Hibernate):**

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

session.save(employee);

tx.commit();

session.close();

**Spring Data JPA**

1. Built on top of JPA/Hibernate to reduce boilerplate code.
2. Offers CRUD operations, custom queries, and pagination out-of-the-box.
3. Uses interfaces like JpaRepository<Employee, Integer> and annotations like @Transactional.

**Example (Spring Data JPA):**

@Autowired

EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee emp) {

employeeRepository.save(emp);

}

**Hands on 5 : Implement services for managing Country**   
  
An application requires for features to be implemented with regards to country. These features needs to be supported by implementing them as service using Spring Data JPA.

* Find a country based on country code
* Add new country
* Update country
* Delete country
* Find list of countries matching a partial country name

**Program:**

**Application.properties**

spring.application.name=countryapp

spring.datasource.url=jdbc:mysql://localhost:3306/countrydb

spring.datasource.username=root

spring.datasource.password=Poorani123@

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

server.port=8081

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**Country.java**

package com.example.countryapp;

import jakarta.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String coCode;

@Column(name = "co\_name")

private String coName;

// No-argument constructor (required by JPA)

public Country() {

}

// Parameterized constructor (optional, useful)

public Country(String coCode, String coName) {

this.coCode = coCode;

this.coName = coName;

}

// Getter and Setter for coCode

public String getCoCode() {

return coCode;

}

public void setCoCode(String coCode) {

this.coCode = coCode;

}

// Getter and Setter for coName

public String getCoName() {

return coName;

}

public void setCoName(String coName) {

this.coName = coName;

}

}

**CountryController.java**

package com.example.countryapp;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/countries")

public class CountryController {

@Autowired

private CountryRepository countryRepo;

@GetMapping

public List<Country> getAllCountries() {

return countryRepo.findAll();

}

@PostMapping

public Country createCountry(@RequestBody Country country) {

return countryRepo.save(country);

}

@GetMapping("/{code}")

public ResponseEntity<Country> getCountry(@PathVariable String code) {

return countryRepo.findById(code)

.map(ResponseEntity::*ok*)

.orElse(ResponseEntity.*notFound*().build());

}

@DeleteMapping("/{code}")

public ResponseEntity<Void> deleteCountry(@PathVariable String code) {

if (countryRepo.existsById(code)) {

countryRepo.deleteById(code);

return ResponseEntity.*noContent*().build();

} else {

return ResponseEntity.*notFound*().build();

}

}

@PutMapping("/{code}")

public ResponseEntity<Country> updateCountry(@PathVariable String code, @RequestBody Country updatedCountry) {

return countryRepo.findById(code)

.map(country -> {

country.setCoName(updatedCountry.getCoName()); // Use getter/setter consistent with entity

return ResponseEntity.*ok*(countryRepo.save(country));

})

.orElse(ResponseEntity.*notFound*().build());

}

}

**CountryRepository.java**

package com.example.countryapp;

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

import java.util.List;

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByCoNameContainingIgnoreCase(String name);

}

**CountryService.java**

package com.example.countryapp;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository repo;

public Country getByCode(String code) {

return repo.findById(code).orElse(null);

}

public Country addCountry(Country country) {

return repo.save(country);

}

public Country updateCountry(Country country) {

return repo.save(country);

}

public void deleteCountry(String code) {

repo.deleteById(code);

}

public List<Country> searchByName(String name) {

return repo.findByCoNameContainingIgnoreCase(name);

}

}

**CountryappApplication.java**

package com.example.countryapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class CountryappApplication {

public static void main(String[] args) {

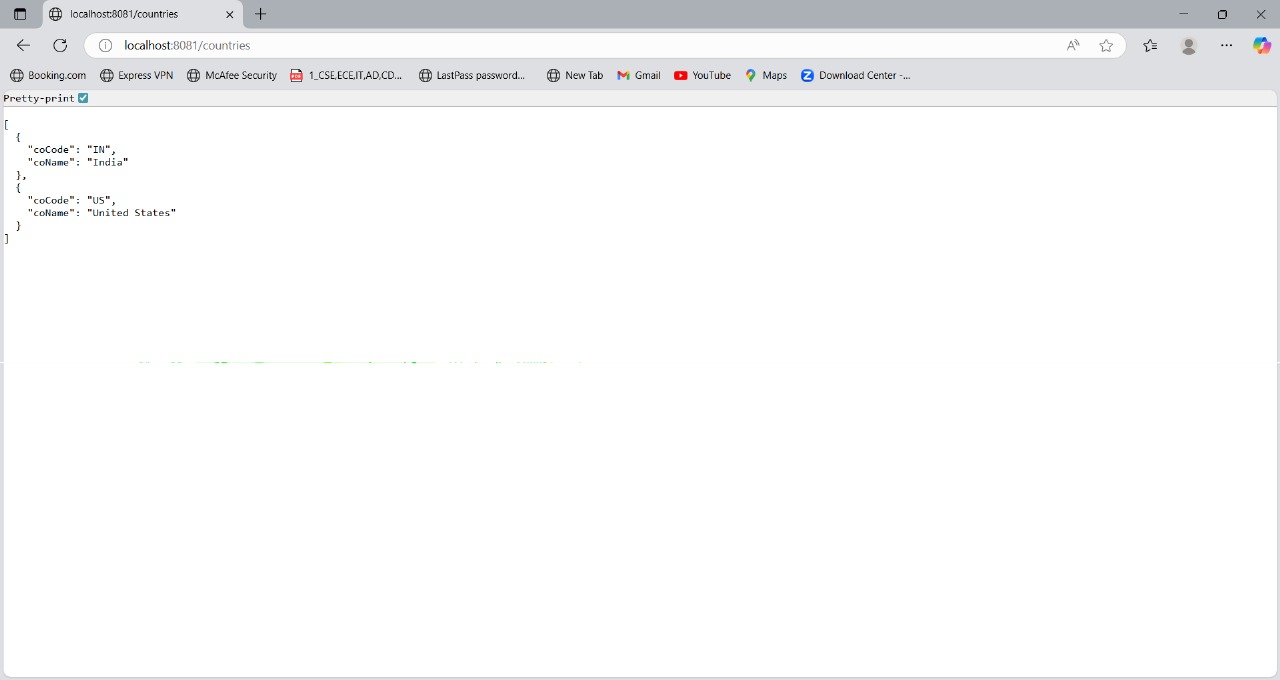
SpringApplication.*run*(CountryappApplication.class, args);

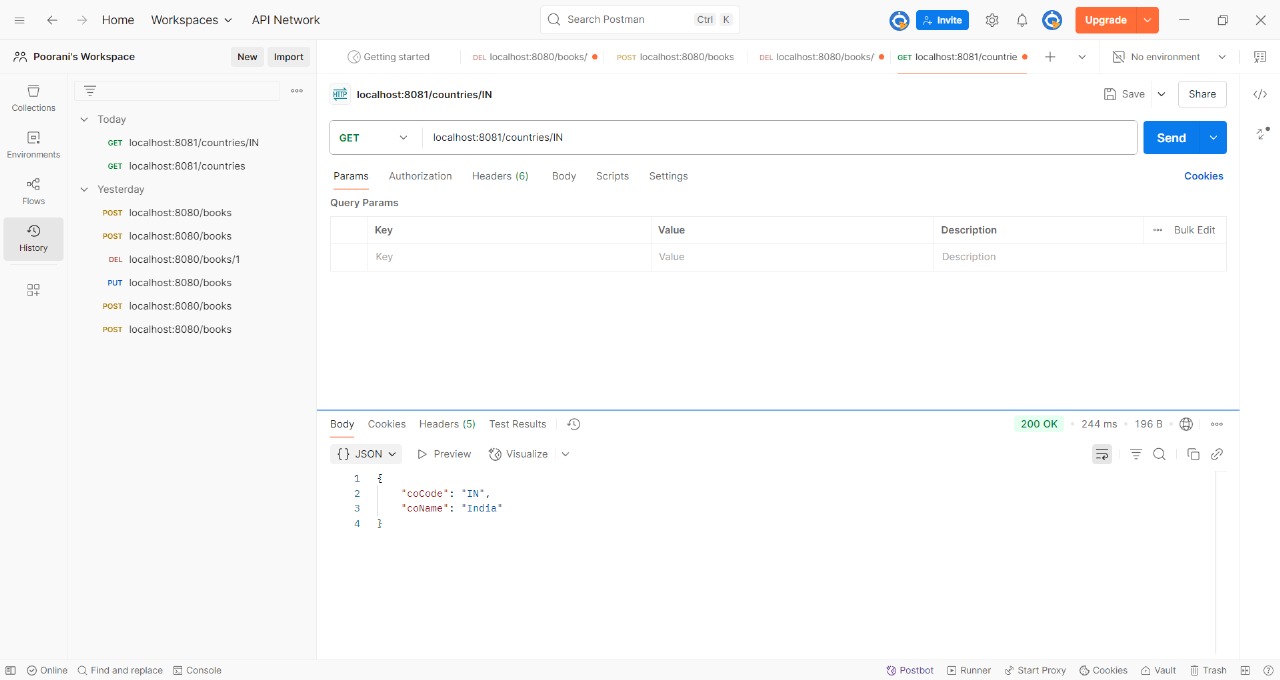
}

}

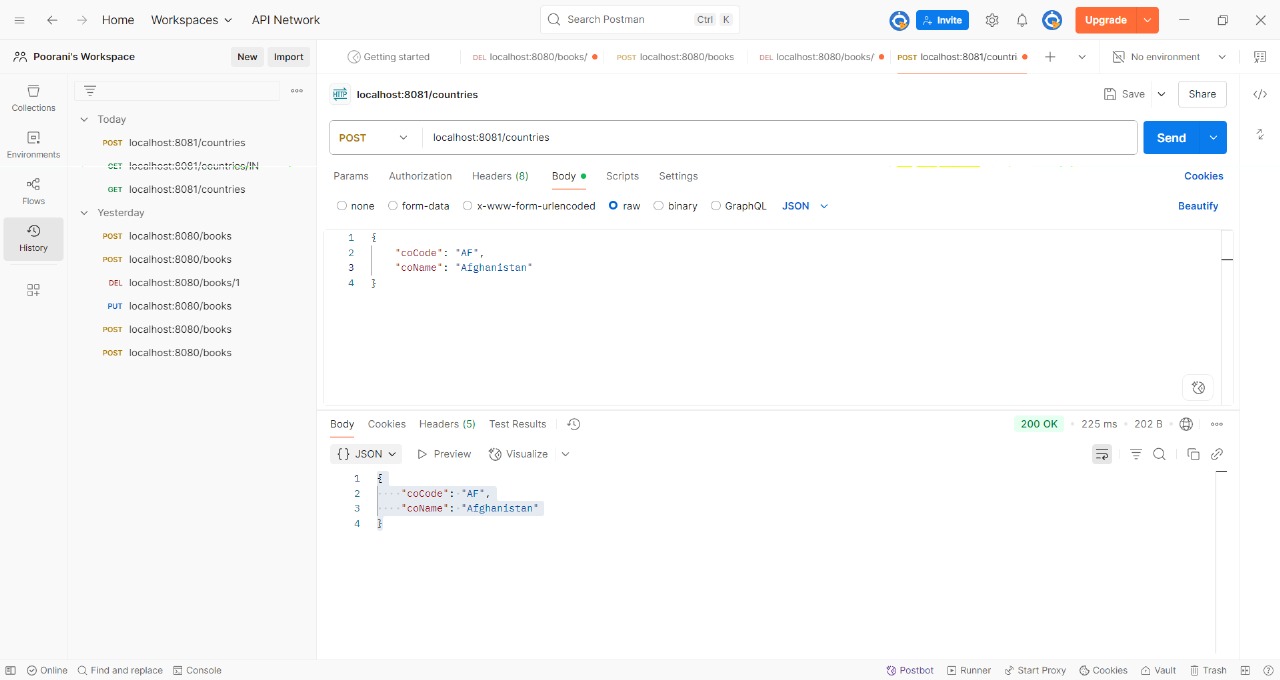
**Output:**

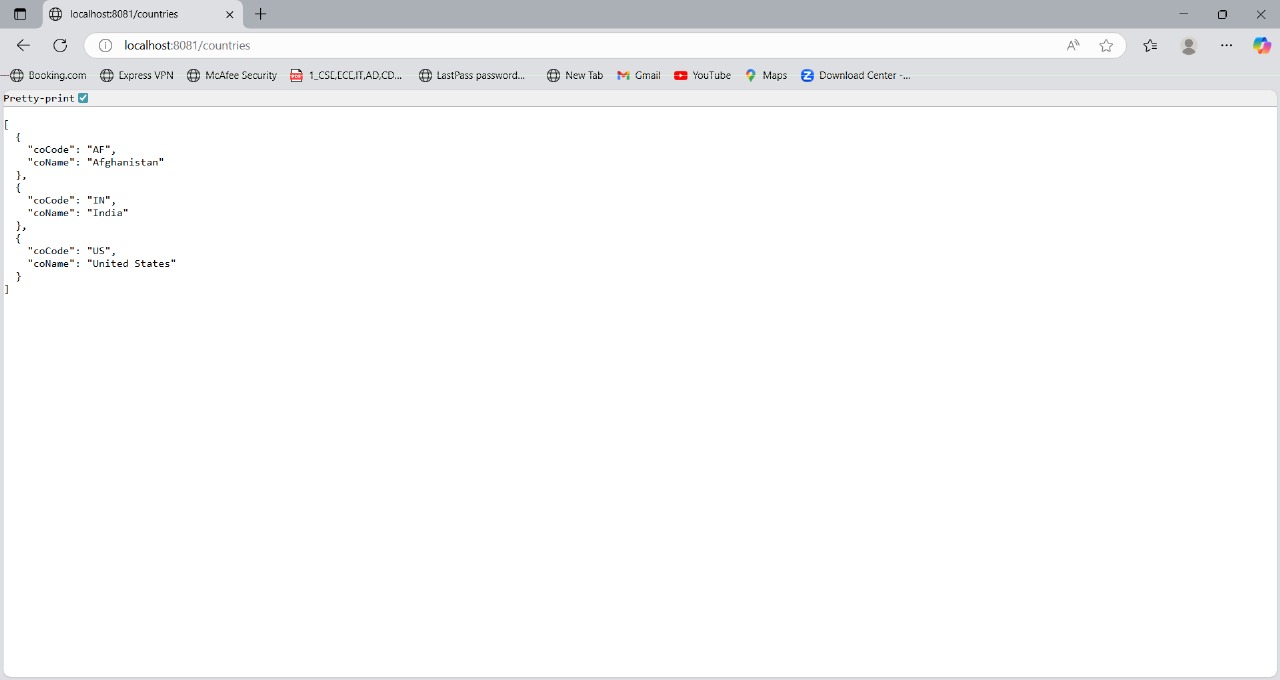
**By using GET method finding a country by country code**



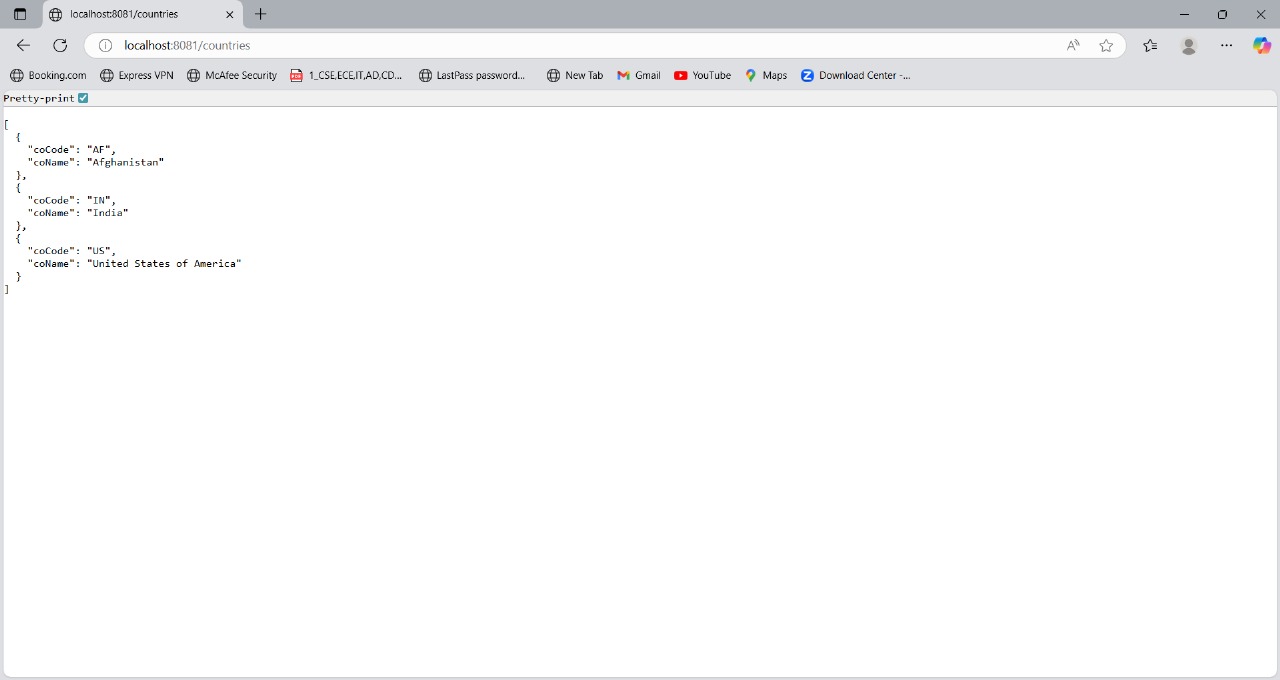


**By using POST method adding a country**

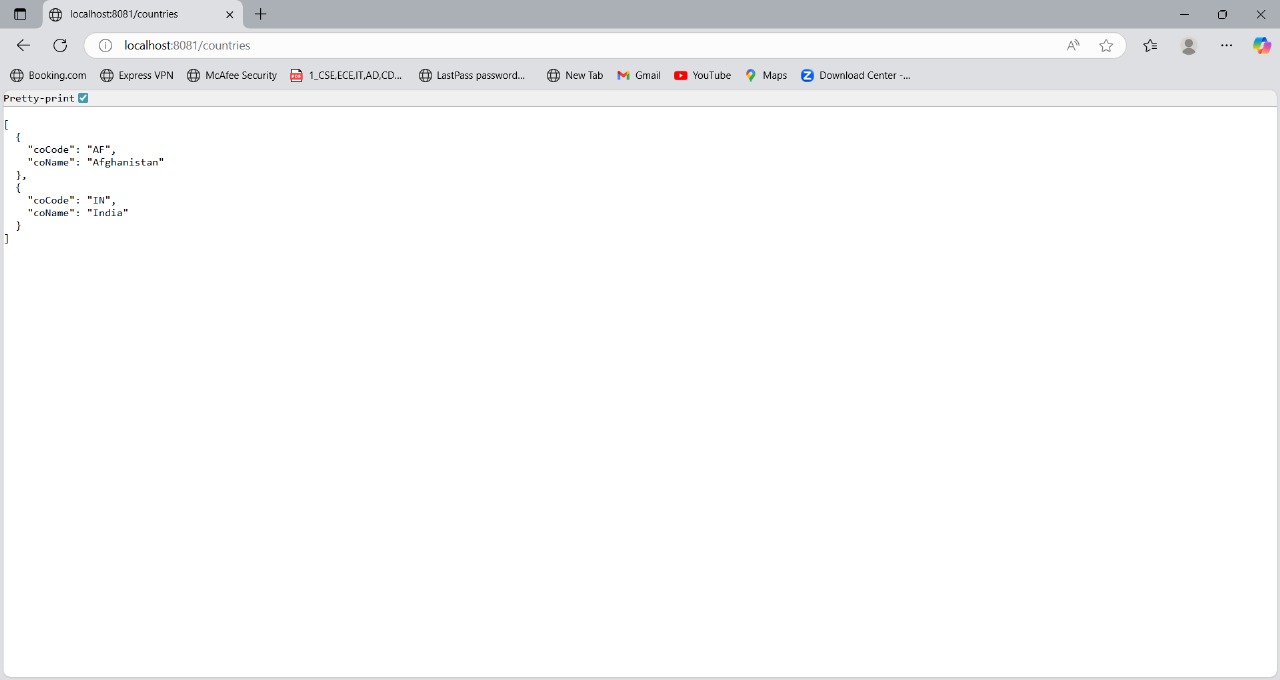




**By using PUT method updating a country US**



**By using DELETE method deleting a country US**



**Hands on 6 : Find a country based on country code**

**Program:**

**Application.properties**

spring.application.name=springlearn

spring.datasource.url=jdbc:mysql://localhost:3306/countrydb

spring.datasource.username=root

spring.datasource.password=Poorani123@

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

logging.level.com.cognizant.springlearn=DEBUG

server.port=8081

**CountryController.java**

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/countries")

public class CountryController {

@Autowired

private CountryService countryService;

@GetMapping("/{code}")

public ResponseEntity<Country> getCountry(@PathVariable String code) throws CountryNotFoundException {

Country country = countryService.findCountryByCode(code);

return ResponseEntity.*ok*(country);

}

@PostMapping

public ResponseEntity<Country> addCountry(@RequestBody Country country) {

Country saved = countryService.addCountry(country);

return ResponseEntity.*ok*(saved);

}

@PutMapping("/{code}")

public ResponseEntity<Country> updateCountry(@PathVariable String code, @RequestBody Country updatedCountry)

throws CountryNotFoundException {

Country updated = countryService.updateCountry(code, updatedCountry);

return ResponseEntity.*ok*(updated);

}

@DeleteMapping("/{code}")

public ResponseEntity<Void> deleteCountry(@PathVariable String code) throws CountryNotFoundException {

countryService.deleteCountry(code);

return ResponseEntity.*noContent*().build();

}

@GetMapping("/search")

public ResponseEntity<List<Country>> searchCountries(@RequestParam String name) {

List<Country> countries = countryService.searchCountriesByName(name);

return ResponseEntity.*ok*(countries);

}

}

**CountryService.java**

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.repository.CountryRepository;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import jakarta.transaction.Transactional;

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;

@Transactional

public Country findCountryByCode(String code) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(code);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + code + " not found");

}

return result.get();

}

@Transactional

public Country addCountry(Country country) {

return countryRepository.save(country);

}

@Transactional

public Country updateCountry(String code, Country updatedCountry) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(code);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + code + " not found");

}

Country country = result.get();

country.setCoName(updatedCountry.getCoName());

return countryRepository.save(country);

}

@Transactional

public void deleteCountry(String code) throws CountryNotFoundException {

if (!countryRepository.existsById(code)) {

throw new CountryNotFoundException("Country with code " + code + " not found");

}

countryRepository.deleteById(code);

}

public List<Country> searchCountriesByName(String name) {

return countryRepository.findByCoNameContainingIgnoreCase(name);

}

}

**CountryRepository.java**

package com.cognizant.springlearn.repository;

import com.cognizant.springlearn.model.Country;

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

import java.util.List;

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByCoNameContainingIgnoreCase(String name);

}

**SpringlearnApplication.java**

package com.cognizant.springlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

@SpringBootApplication

public class SpringlearnApplication {

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(SpringlearnApplication.class);

private static CountryService *countryService*;

public static void main(String[] args) throws CountryNotFoundException {

ApplicationContext context = SpringApplication.*run*(SpringlearnApplication.class, args);

*countryService* = context.getBean(CountryService.class);

*getAllCountriesTest*();

}

private static void getAllCountriesTest() throws CountryNotFoundException {

*LOGGER*.info("Start");

Country country = *countryService*.findCountryByCode("IN");

*LOGGER*.debug("Country: {}", country);

*LOGGER*.info("End");

}

}

**CountryNotFoundException.java**

package com.cognizant.springlearn.service.exception;

public class CountryNotFoundException extends Exception {

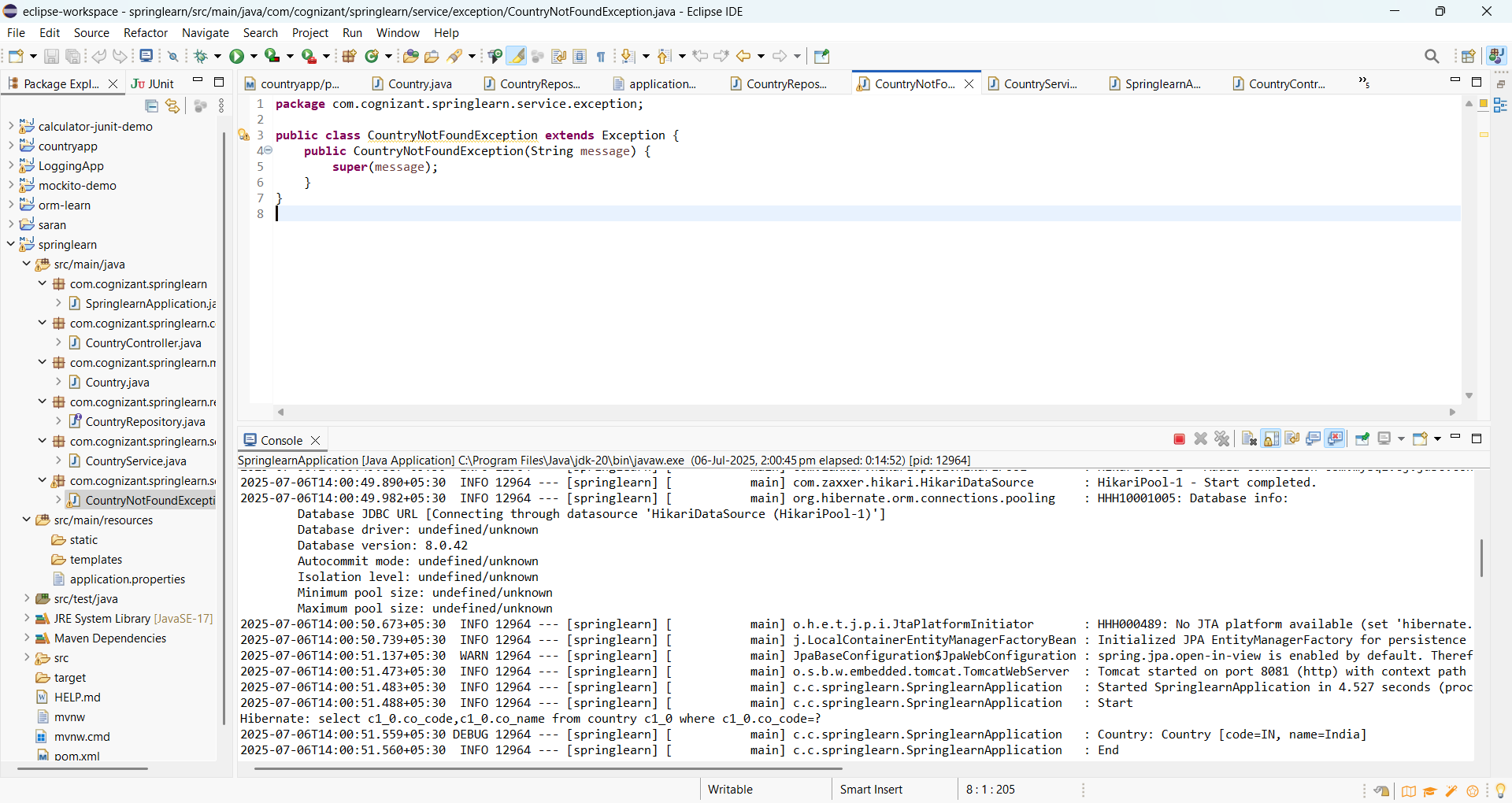
public CountryNotFoundException(String message) {

super(message);

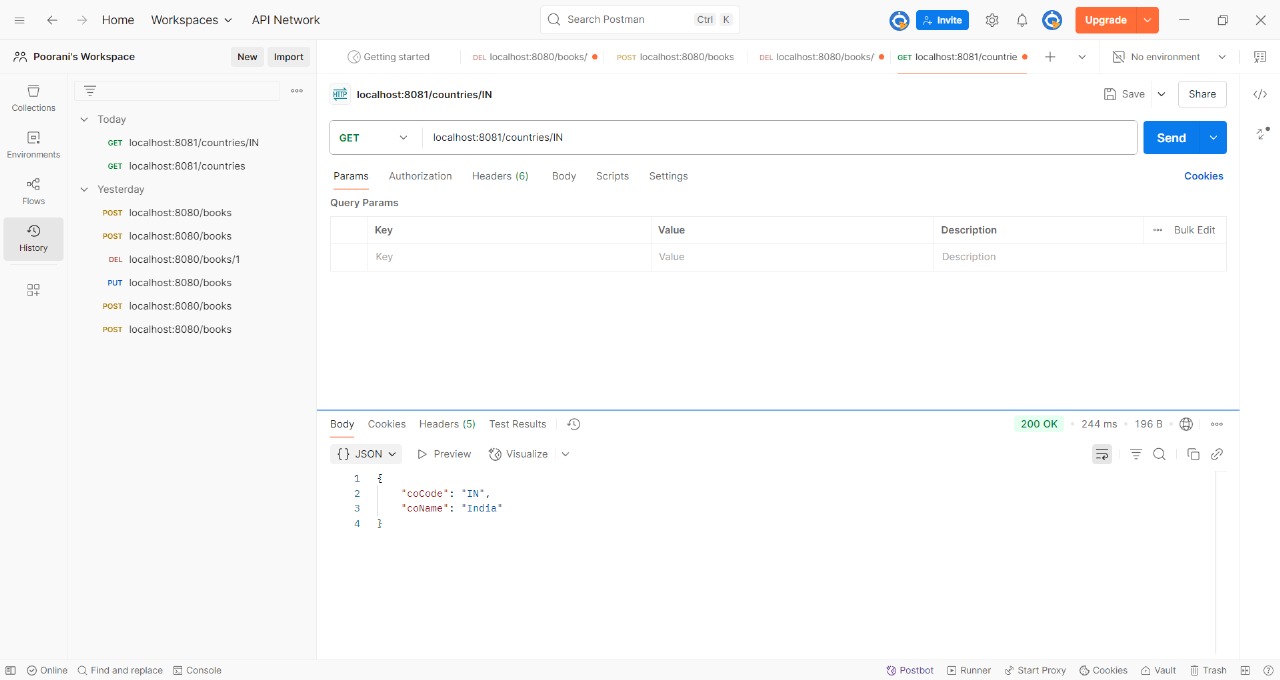
}

}

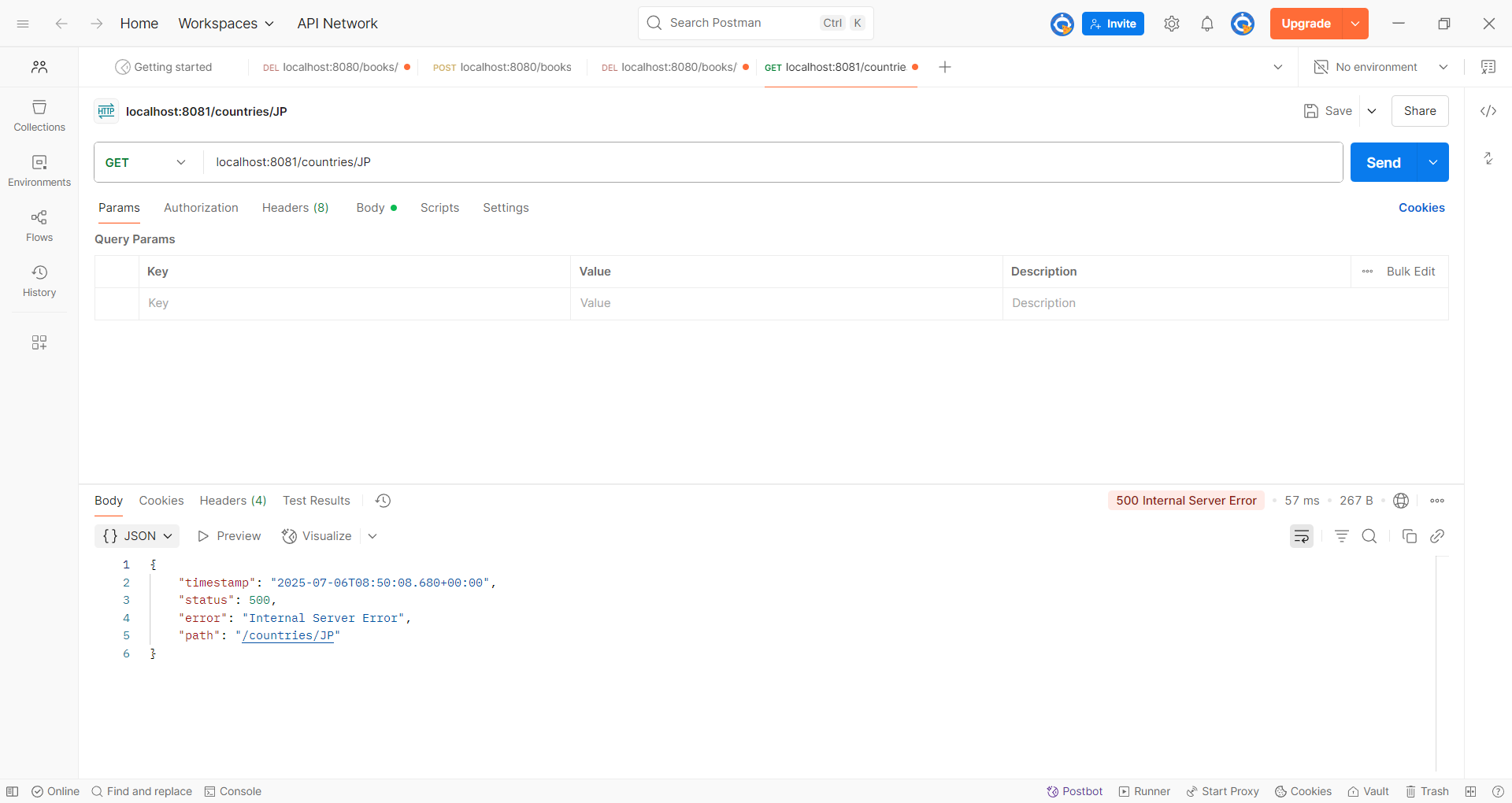
**Output:**

****

**By using GET method finding country by id**



**If the country is not found then the CountryNotFoundException will be thrown**

****

**Hands on 7 : Add a new country**

**Create new method in CountryService**

**CountryService.java**

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.repository.CountryRepository;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import jakarta.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found with code: " + countryCode);

}

return result.get();

}

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

}

**Include new testAddCountry() method in OrmLearnApplication. Perform steps below:**

**OrmlearnApplication.java**

package com.cognizant.springlearn;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import jakarta.annotation.PostConstruct;

@SpringBootApplication

public class OrmlearnApplication {

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmlearnApplication.class);

@Autowired

private CountryService countryService;

public static void main(String[] args) {

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

}

@PostConstruct

public void testAddCountry() throws CountryNotFoundException {

*LOGGER*.info("Start");

// Step 1: Create a new Country object

Country country = new Country();

country.setCoCode("JP");

country.setCoName("Japan");

// Step 2: Add the country

countryService.addCountry(country);

// Step 3: Fetch it to verify

Country addedCountry = countryService.findCountryByCode("JP");

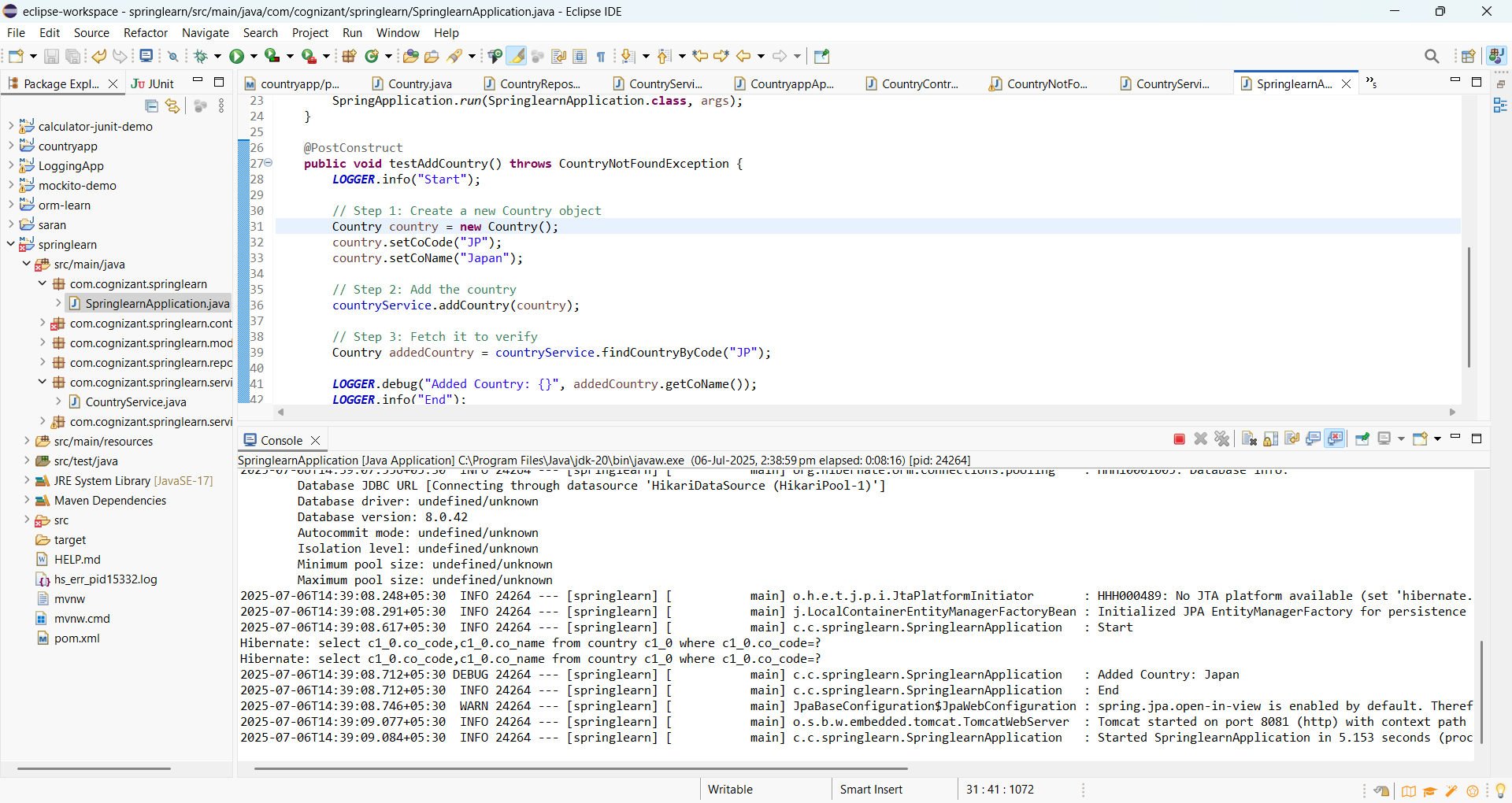
*LOGGER*.debug("Added Country: {}", addedCountry.getCoName());

*LOGGER*.info("End");

}

}

**Output:**

****

**Hands on 8 : Update a country based on code**

**Create a new method updateCountry() in CountryService with parameters code and name. Annotate this method with @Transactional.**

**CountryService.java**

package com.cognizant.springlearn.service;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

import com.cognizant.orm\_learn.service.exception.CountryNotFoundException;

import jakarta.transaction.Transactional;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

*@Transactional*

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + countryCode + " not found");

}

return result.get();

}

*@Transactional*

public void addCountry(Country country) {

countryRepository.save(country);

}

*@Transactional*

public void updateCountry(String code, String name) throws CountryNotFoundException {

Optional<Country> optionalCountry = countryRepository.findById(code);

if (!optionalCountry.isPresent()) {

throw new CountryNotFoundException("Country with code " + code + " not found.");

}

Country country = optionalCountry.get();

country.setName(name);

countryRepository.save(country);

}

}

**Include new test method in OrmLearnApplication, which invokes updateCountry() method in CountryService passing a country's code and different name for the country.**

**OrmlearnApplication**

package com.cognizant.springlearn;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import jakarta.annotation.PostConstruct;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OrmlearnApplication {

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmlearnApplication.class);

@Autowired

private CountryService countryService;

public static void main(String[] args) {

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

}

@PostConstruct

public void runAllTests() throws CountryNotFoundException {

testUpdateCountry();

}

public void testUpdateCountry() throws CountryNotFoundException {

*LOGGER*.info("Start");

// Update existing country with code "IN"

countryService.updateCountry("IN", "Bharat");

// Verify update

Country updatedCountry = countryService.findCountryByCode("IN");

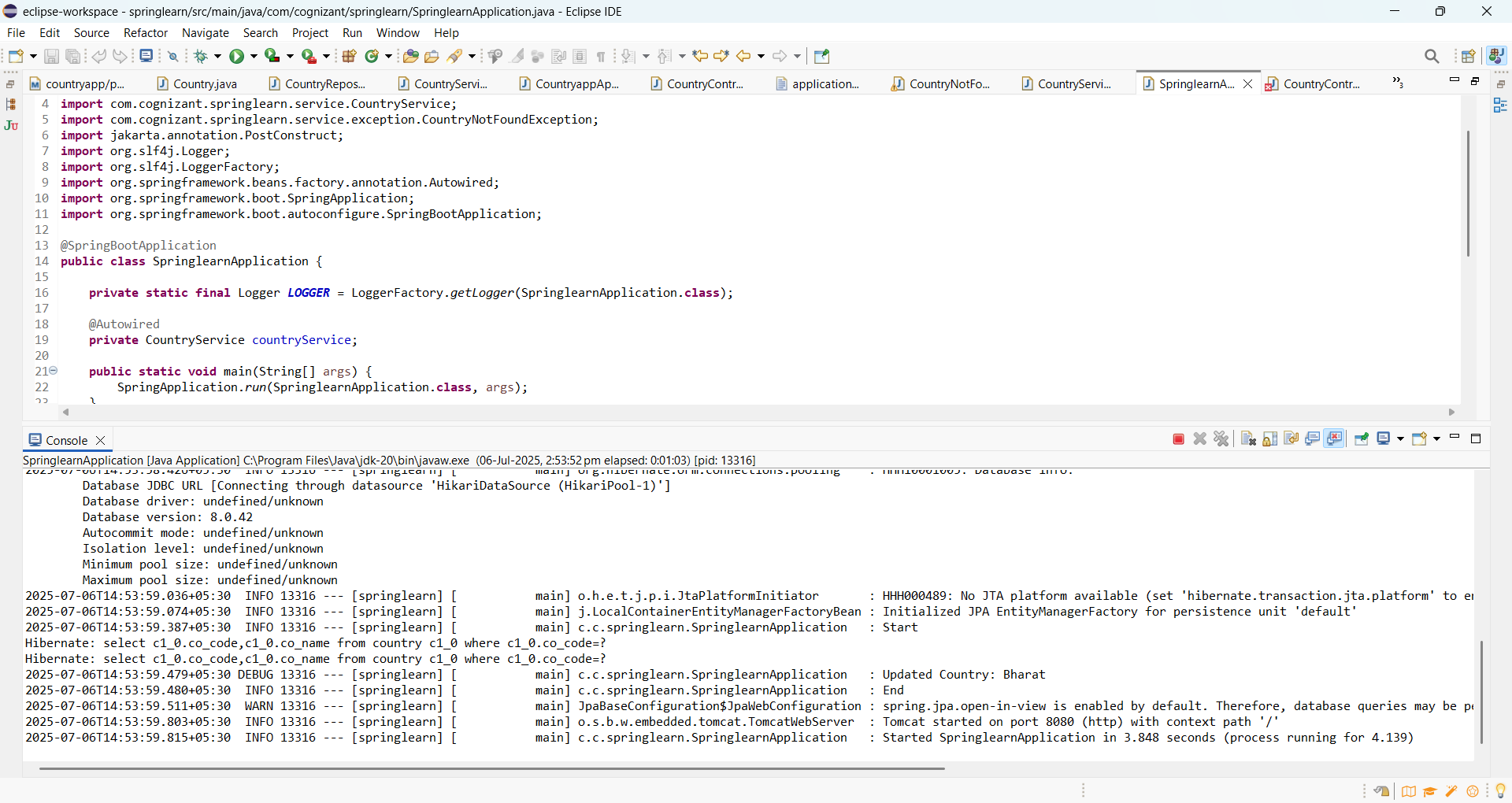
*LOGGER*.debug("Updated Country: {}", updatedCountry.getCoName());

*LOGGER*.info("End");

}

}

**Output:**

****

**Hands on 9 : Delete a country based on code**

**Create new method deleteCountry() in CountryService. Annotate this method with @Transactional.**

**CountryService.java**

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.repository.CountryRepository;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import jakarta.transaction.Transactional;

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;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + countryCode + " not found");

}

return result.get();

}

@Transactional

public Country addCountry(Country country) {

return countryRepository.save(country);

}

@Transactional

public Country updateCountry(String code, Country updatedCountry) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(code);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + code + " not found");

}

Country country = result.get();

country.setCoName(updatedCountry.getCoName());

return countryRepository.save(country);

}

@Transactional

public void deleteCountry(String code) throws CountryNotFoundException {

if (!countryRepository.existsById(code)) {

throw new CountryNotFoundException("Country with code " + code + " not found");

}

countryRepository.deleteById(code);

}

public List<Country> searchCountriesByName(String name) {

return countryRepository.findByCoNameContainingIgnoreCase(name);

}

public void deleteCountry(Country country) {

// TODO Auto-generated method stub

}

}

**Include new test method in OrmLearnApplication with following steps**

**OrmlearnApplication.java**

package com.cognizant.springlearn;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class OrmlearnApplication {

private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmlearnApplication.class);

private static CountryService *countryService*;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.*run*(OrmlearnApplication.class, args);

*countryService* = context.getBean(CountryService.class);

*getCountryByCodeTest*();

*testAddCountry*();

*testUpdateCountry*();

*testDeleteCountry*();

}

private static void getCountryByCodeTest() {

*LOGGER*.info("Start - getCountryByCodeTest");

try {

Country country = *countryService*.findCountryByCode("IN");

*LOGGER*.debug("Country: {}", country);

} catch (CountryNotFoundException e) {

*LOGGER*.error("Error: {}", e.getMessage());

}

*LOGGER*.info("End - getCountryByCodeTest");

}

private static void testAddCountry() {

*LOGGER*.info("Start - testAddCountry");

Country country = new Country();

country.setCoCode("JP");

country.setCoName("Japan");

*countryService*.deleteCountry(country);

*LOGGER*.info("Added country: {}", country);

*LOGGER*.info("End - testAddCountry");

}

private static void testUpdateCountry() {

*LOGGER*.info("Start - testUpdateCountry");

try {

Country updated = new Country();

updated.setCoName("Republic of Japan");

Country result = *countryService*.updateCountry("JP", updated);

*LOGGER*.info("Updated Country: {}", result);

} catch (CountryNotFoundException e) {

*LOGGER*.error("Error: {}", e.getMessage());

}

*LOGGER*.info("End - testUpdateCountry");

}

private static void testDeleteCountry() {

*LOGGER*.info("Start - testDeleteCountry");

try {

*countryService*.deleteCountry("JP");

*LOGGER*.info("Deleted country with code JP");

} catch (CountryNotFoundException e) {

*LOGGER*.error("Error: {}", e.getMessage());

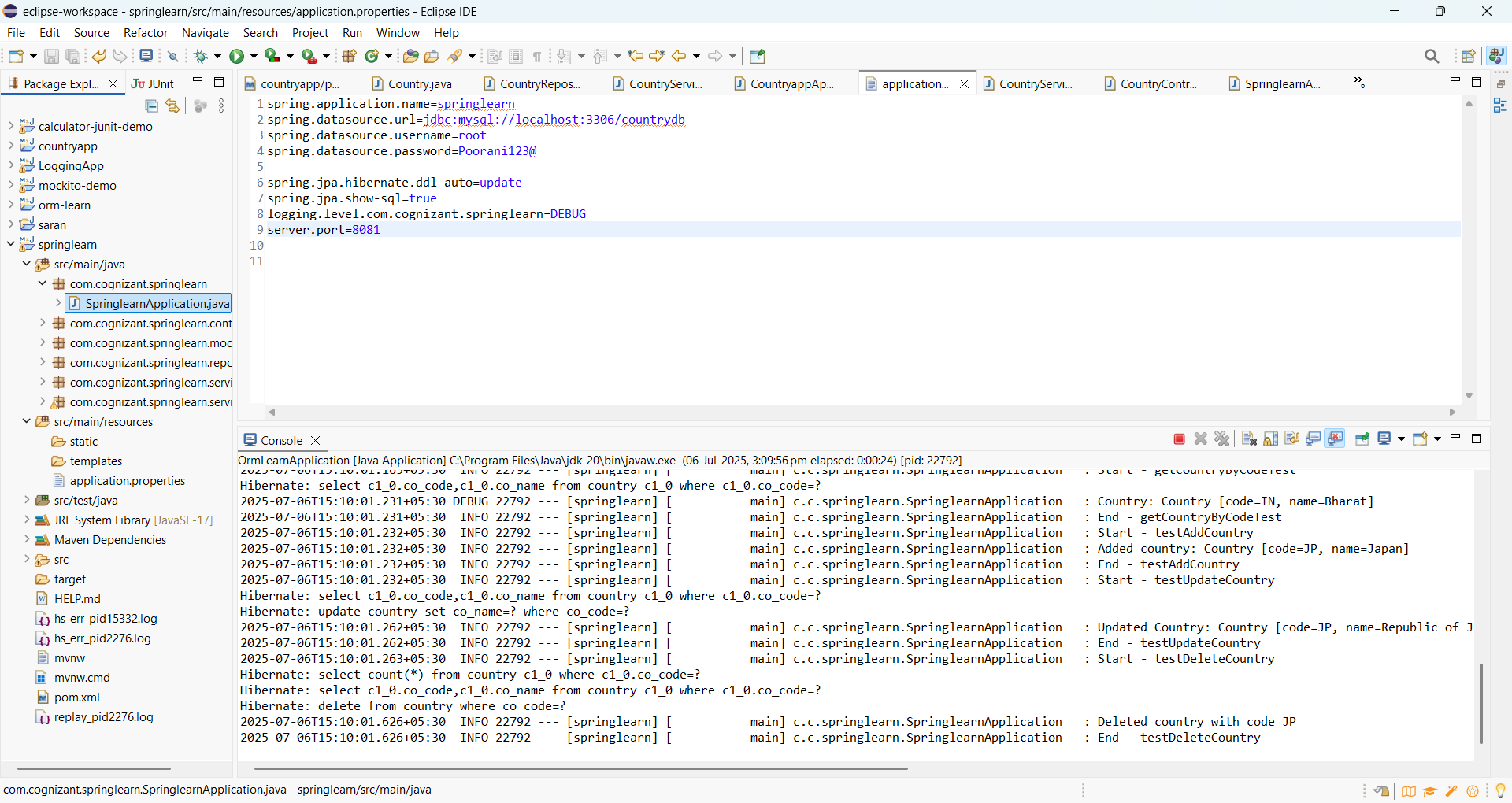
}

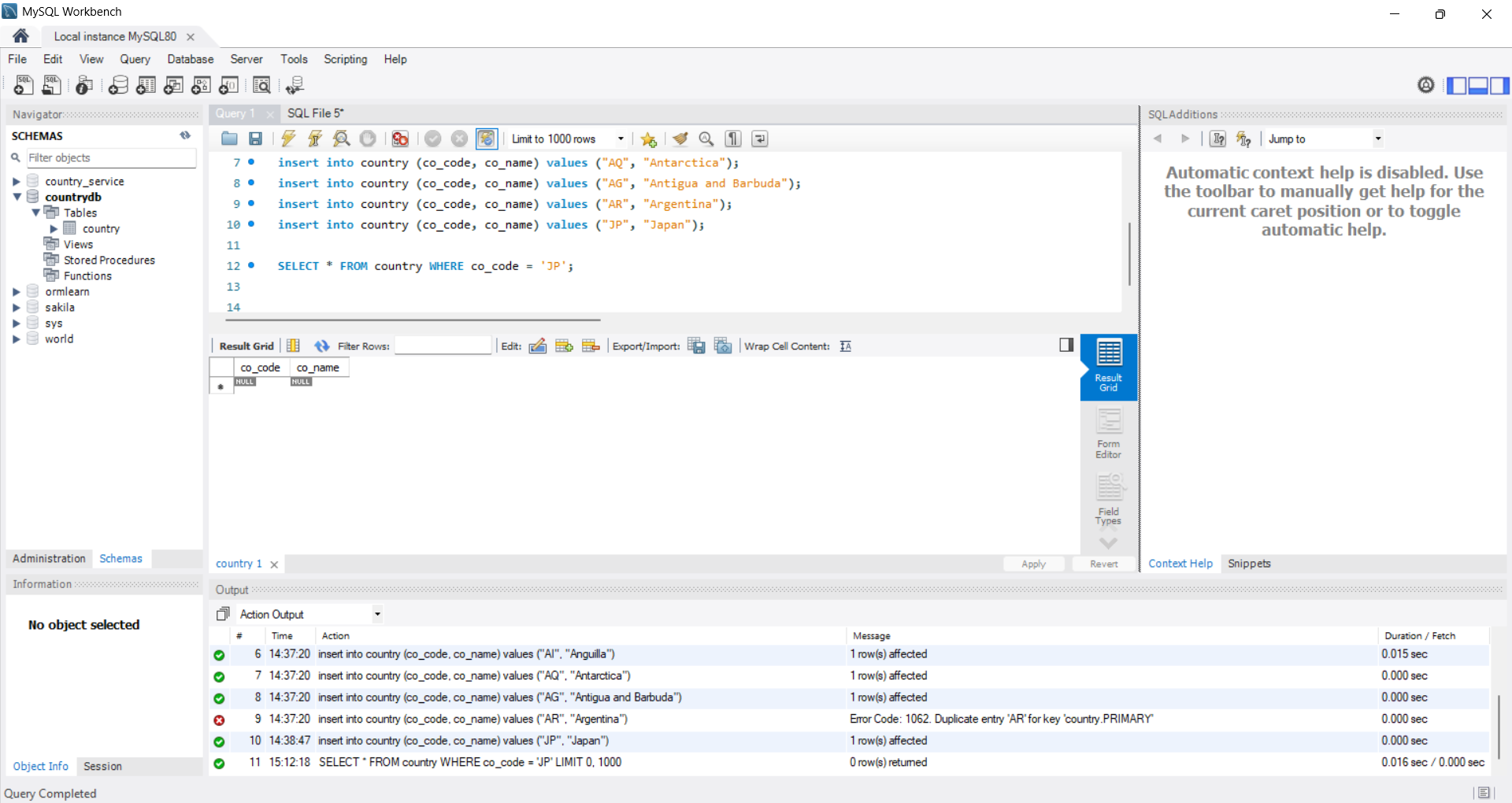
*LOGGER*.info("End - testDeleteCountry");

}

}

**Output:**

****

****