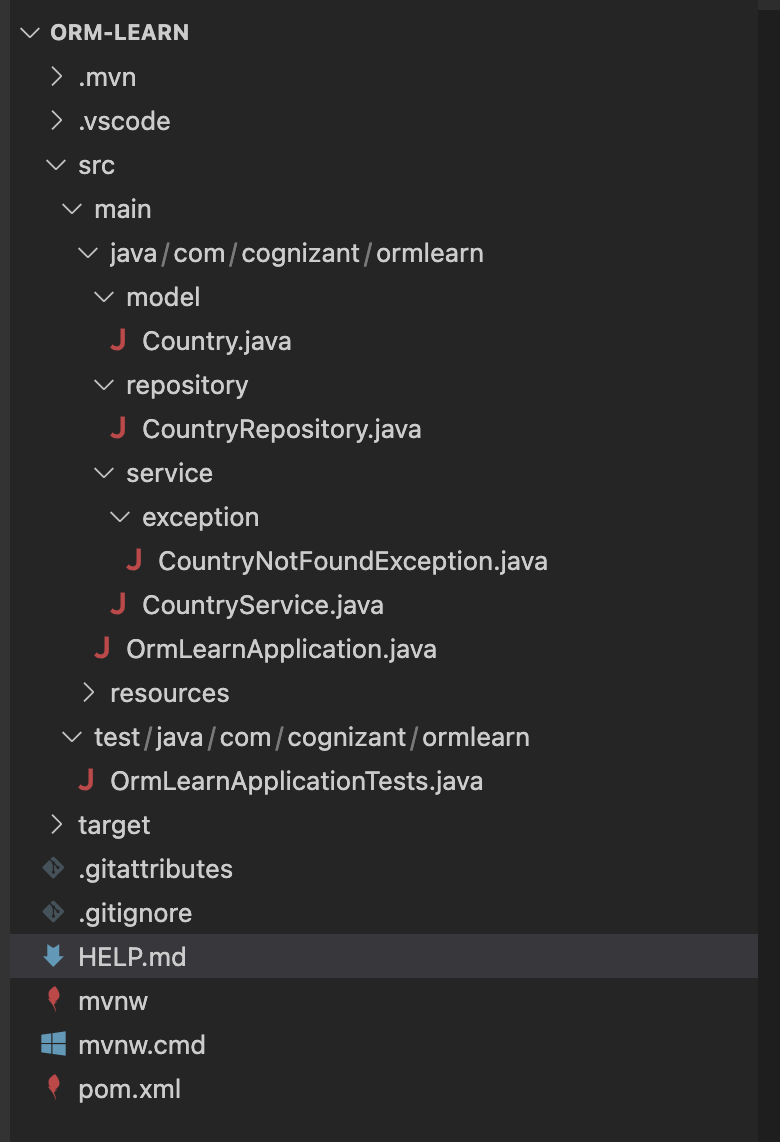
**Folder Structure on the whole**



**Hands-On 1: Spring Data JPA - Quick Example**

**Application.properties:**

**# Logging config**

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

**# Database 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=root

**# Hibernate config**

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

**Country.java:**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "name")

private String name;

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;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**CountryRepository.java**

package com.cognizant.ormlearn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java:**

package com.cognizant.ormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import jakarta.transaction.Transactional;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java:**

package com.cognizant.ormlearn;

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.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@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);

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");

}

}

**import.sql:**

CREATE DATABASE ormlearn;

USE ormlearn;

CREATE TABLE country (

code VARCHAR(2) PRIMARY KEY,

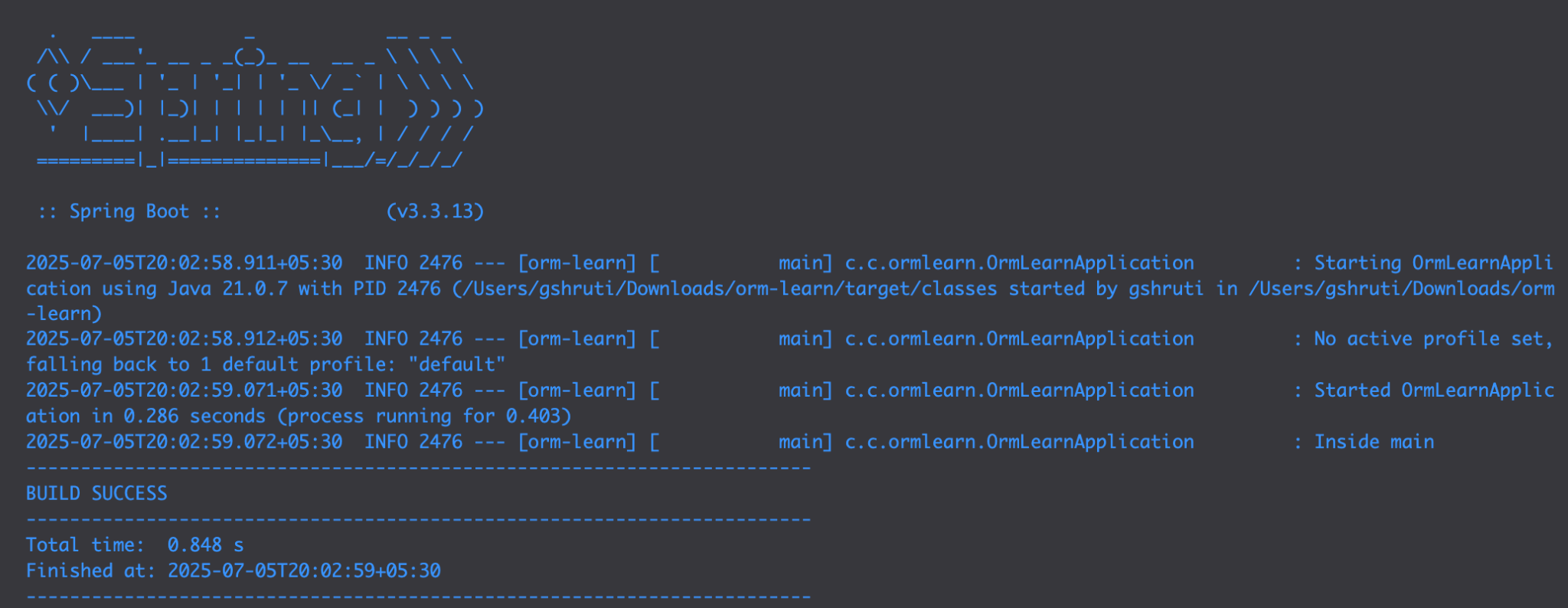
name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');

**Output:**

****

**Hands-On 2: Hibernate XML Config implementation walk through**

**Object to Relational Mapping in XML**

Hibernate uses mapping files (XML) to link Java classes and their properties to corresponding database tables and columns.

This XML mapping:

* Maps the Java class Country to the database table country
* Maps the field code to primary key column code
* Maps the field name to column name

**Key Hibernate Concepts**

* **SessionFactory**: A singleton, heavy-weight object created once per application. It reads the configuration file (hibernate.cfg.xml) and mapping files and creates sessions.
* **Session:** A lightweight, short-lived object used to interact with the database. Created by SessionFactory.
* **Transaction:** Optional but recommended. Represents a unit of work. Ensures atomicity.

**Important API Methods**

* **beginTransaction():** Begins a database transaction.
* **commit():** Commits changes made during the transaction to the DB.
* **rollback()**: Rolls back the changes in case of failure/error.
* **session.save(obj)**: Saves a new entity to the db, Returns the identifier.
* **session.get(Class,id):** Fetches an entity by primary key. Return null if not found.
* **session.createQuery().list():** Executes HQL query and returns list of results.
* **session.delete(obj)**: deletes the specified entity from the db.

**File Structure in XML Config Setup**

* **hibernate.cfg.xml:** Central configuration file with db details, mappings.
* **\*.hbm.xml:** Mapping file which maps Java class to db.
* **Java POJO:** Class with fields, getters, setters.
* **HibernateUtil.java:** Utility to create and manage SessionFactory.

**Sample Hibernate Configuration (hibernate.cfg.xml):**

<hibernate-configuration>

<session-factory>

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

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

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

<property name="hibernate.connection.password">root</property> <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>

<mapping resource="Country.hbm.xml"/>

</session-factory>

</hibernate-configuration

**Hands-on 3: Hibernate Annotation Configuration Walkthrough**

### **Object to Relational Mapping Using Annotations:**

### Hibernate allows you to map Java classes to database tables using annotations.

Example:

import jakarta.persistence.\*;

@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;

}

### **Annotations Explained (Point Format)**

* **@Entity**  
  → Marks the class as a Hibernate entity.
* **@Table(name = "employee")**→ Maps the class to the database table named **employee**.
* **@Id**  
  → Specifies the primary key field of the entity.
* **@GeneratedValue(strategy = GenerationType.IDENTITY)**→ Defines how the primary key is auto-generated.  
  → IDENTITY strategy uses the database's **auto-increment** feature.
* **@Column(name = "...")**  
  → Maps a class field to a specific **column name** in the table.

**Hibernate CRUD Operations in Code**

SessionFactory factory = new Configuration().configure("hibernate.cfg.xml")

.addAnnotatedClass(Employee.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

try {

Employee emp = new Employee("Shruti", "G", 30000);

session.beginTransaction();

session.save(emp);

session.getTransaction().commit();

} finally {

factory.close();

}

**Hands-on 4: Difference between JPA, Hibernate, and Spring Data JPA**

**Java Persistence API (JPA)**

* JPA is a Java specification (JSR 338) for persisting, reading, and managing data from Java objects.
* It is only an interface/contract, not a concrete implementation.
* It defines standard annotations like @Entity, @Id, @Table, etc.
* Examples of implementations: Hibernate, EclipseLink, DataNucleus.

**Hibernate**

* Hibernate is an ORM (Object Relational Mapping) tool.
* It is the most popular implementation of JPA.
* Provides extra features over JPA like:
  + Lazy/eager fetching strategies
  + Caching support
  + Better query language (HQL)
* Can be used both with or without JPA.

**Spring Data JPA**

* Not a JPA implementation.
* It's a part of the Spring ecosystem that provides:
  + A higher level abstraction over JPA.
  + Built-in support for common CRUD operations.
  + Reduces boilerplate code via interfaces like JpaRepository.
* Internally uses JPA implementations like Hibernate.
* Handles:
  + Repository pattern
  + Transaction management
  + Pagination, sorting, and query derivation by method names.

**Comparison: Hibernate vs Spring Data JPA**

Hibernate (Manual Session Management):

public Integer addEmployee(Employee employee) {

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

Spring Data JPA (Simplified)

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

**Hands-on 5: Implement Services for Managing Country**

**CountryService.java:**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

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 List<Country> getAllCountries() {

return countryRepository.findAll();

}

@Transactional

public Country findCountryByCode(String code) {

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

return result.orElse(null);

}

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

@Transactional

public void updateCountry(String code, String name) {

Country country = findCountryByCode(code);

if (country != null) {

country.setName(name);

countryRepository.save(country);

}

}

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

@Transactional

public List<Country> findCountriesByName(String name) {

return countryRepository.findByNameContainingIgnoreCase(name);

}

}

**CountryRepositrory.java:**

package com.cognizant.ormlearn.repository;

import com.cognizant.ormlearn.model.Country;

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String name);

}

**OrmLearnApplication.java:**

package com.cognizant.ormlearn;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

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 java.util.List;

@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);

countryService = context.getBean(CountryService.class);

LOGGER.info("Inside main");

testAddCountry();

testGetAllCountries();

testFindCountryByCode();

testUpdateCountry();

testDeleteCountry();

testFindCountriesByName();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

private static void testFindCountryByCode() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

private static void testAddCountry() {

LOGGER.info("Start");

Country country = new Country();

country.setCode("XY");

country.setName("NewTestCountry");

countryService.addCountry(country);

LOGGER.info(" New Country added: {}", countryService.findCountryByCode("XY"));

}

private static void testUpdateCountry() {

LOGGER.info("Start");

countryService.updateCountry("XY", "UpdatedCountryName");

LOGGER.info(" Country updated: {}", countryService.findCountryByCode("XY"));

}

private static void testDeleteCountry() {

LOGGER.info("Start");

countryService.deleteCountry("XY");

LOGGER.info(" Country deleted.");

}

private static void testFindCountriesByName() {

LOGGER.info("Start");

List<Country> countries = countryService.findCountriesByName("an");

LOGGER.debug("Countries with 'an' in name: {}", countries);

LOGGER.info("End");

}

}

**Country.java:**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

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;

}

@Override

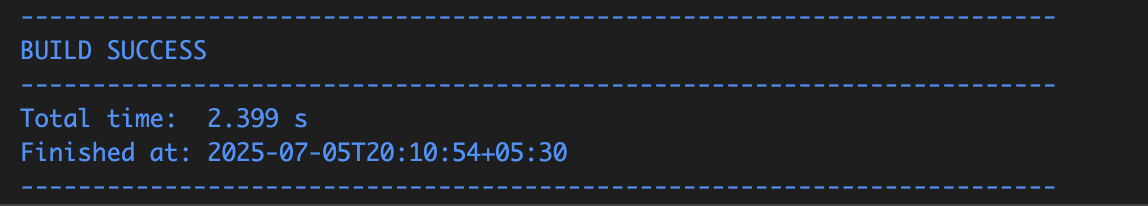
public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**Output:**



**Hands-on 6: Find a Country Based on Country Code**

**CountryNotFoundException.java:**

package com.cognizant.ormlearn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

**CountryService.java:**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import com.cognizant.ormlearn.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 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 not found for code: " + countryCode);

}

return result.get();

}

}

**OrmLearnApplication.java:**

Added this method to already present code

private static void testFindCountryByCodeException() {

LOGGER.info("Start");

try {

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

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

} catch (Exception e) {

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

}

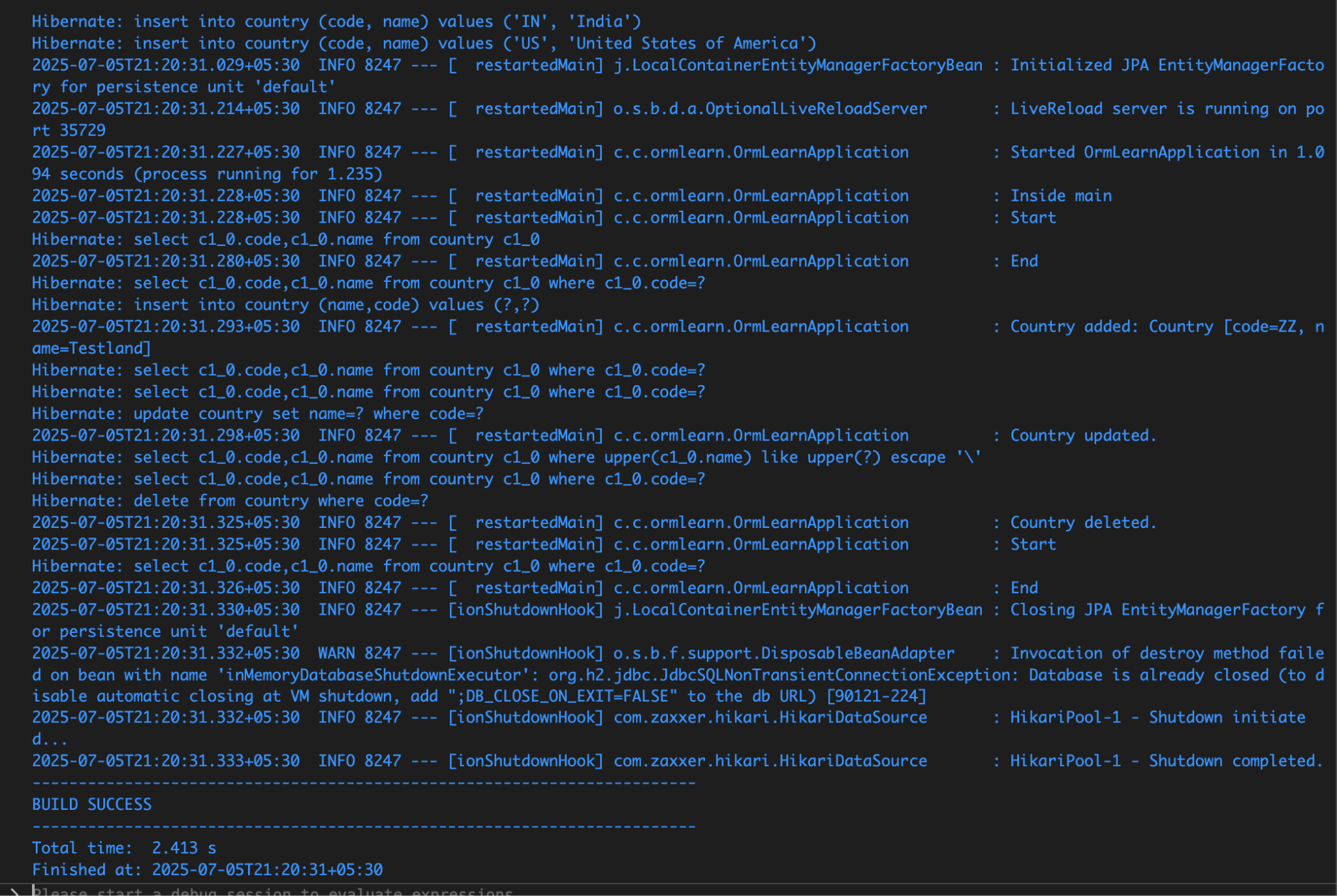
LOGGER.info("End");

}

Add this to the main method

testFindCountryByCodeException();

**Output:**



**Hands-on 7: Add a New Country**

**CountryService.java:**

Update the code already present by adding

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

**OrmLearnApplication:**

Update the already present code by adding this method

private static void testAddCountry() {

LOGGER.info("Start");

Country country = new Country();

country.setCode("ZZ");

country.setName("Testland");

countryService.addCountry(country);

try {

Country savedCountry = countryService.findCountryByCode("ZZ");

LOGGER.debug("Added Country: {}", savedCountry);

} catch (Exception e) {

LOGGER.error("Country not found after adding: {}", e.getMessage());

}

LOGGER.info("End");

}

Also adding this to the main method

public static void main(String[] args) {

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

countryService = context.getBean(CountryService.class);

testAddCountry(); }

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

**CountryService.java:**

Update the code already present by adding

@Transactional

public void updateCountry(String code, String name) {

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

if (result.isPresent()) {

Country country = result.get();

country.setName(name);

countryRepository.save(country);

} else {

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

}

}

**OrmLearnApplication:**

Update the already present code by adding this method

private static void testUpdateCountry() {

LOGGER.info("Start");

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

try {

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

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

} catch (Exception e) {

LOGGER.error("Country not found: {}", e.getMessage());

}

LOGGER.info("End");

}

Also adding this to the main method

public static void main(String[] args) {

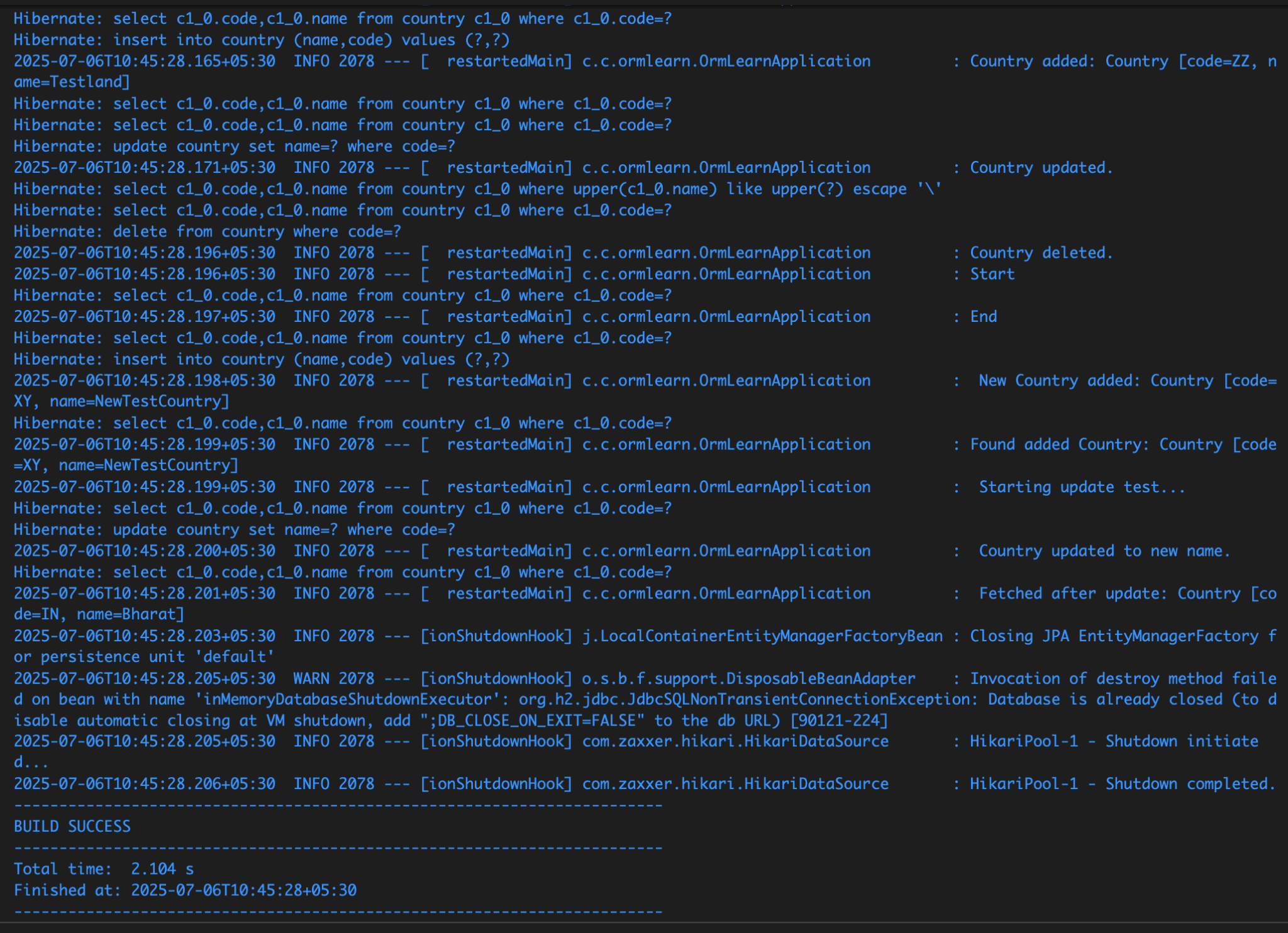
ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

testUpdateCountry();

}

**Output:**



**Hands-on 8: Update a Country Based on Code**

**CountryService.java:**

Update the code already present by adding

@Transactional

public void updateCountry(String code, String name) {

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

if (result.isPresent()) {

Country country = result.get();

country.setName(name);

countryRepository.save(country);

} else {

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

}

}

**OrmLearnApplication:**

Update the already present code by adding this method

private static void testUpdateCountry() {

LOGGER.info("Start");

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

try {

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

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

} catch (Exception e) {

LOGGER.error("Country not found: {}", e.getMessage());

}

LOGGER.info("End");

}

Also adding this to the main method

public static void main(String[] args) {

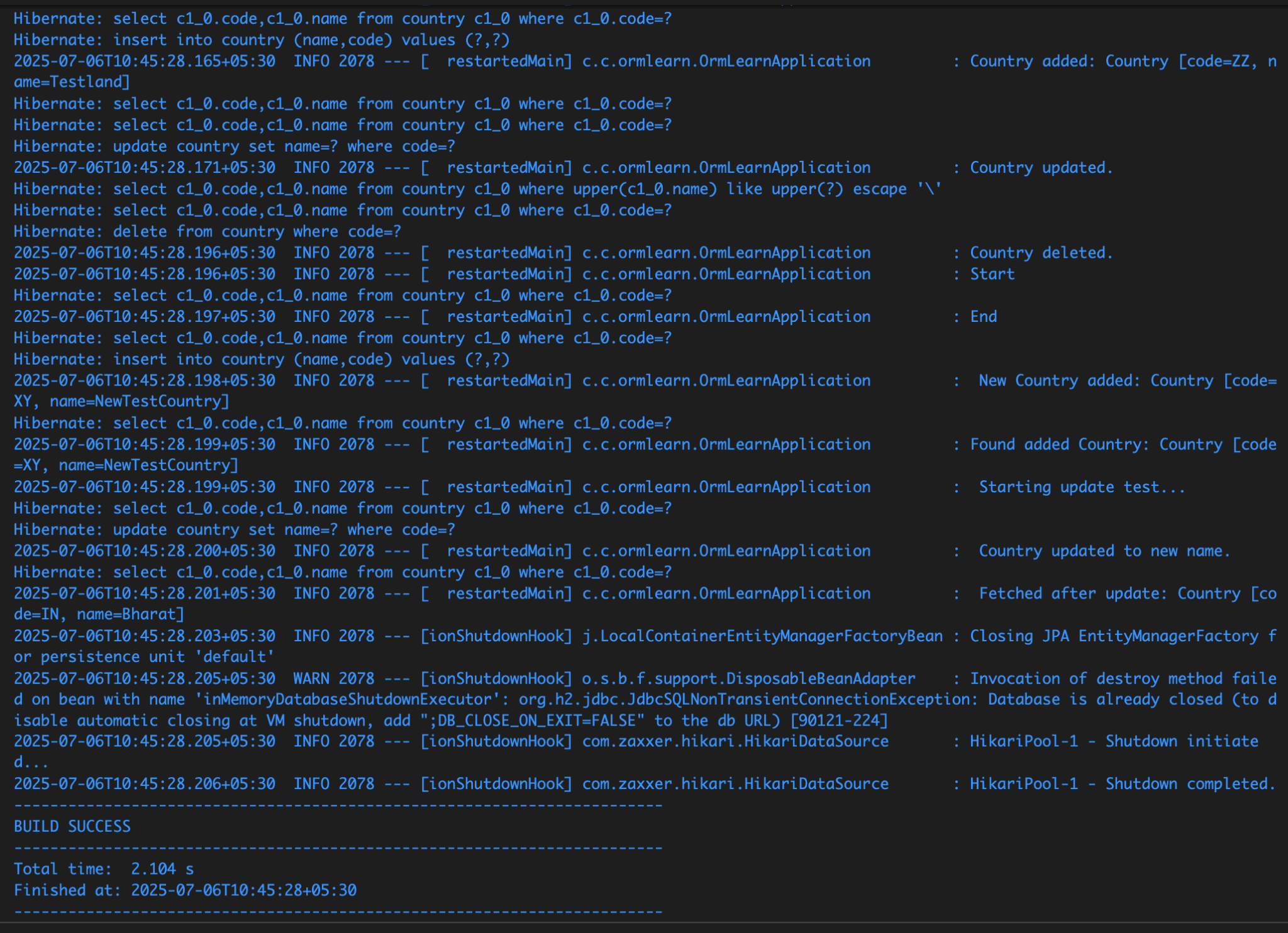
ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

testUpdateCountry();

}

**Output:**



**Hands-on 9: Delete a Country Based on Code**

**CountryService.java:**

package com.cognizant.ormlearn.service;

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import com.cognizant.ormlearn.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);

}

@Transactional

public void deleteCountry(String code) throws CountryNotFoundException {

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

if (!optionalCountry.isPresent()) {

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

}

countryRepository.deleteById(code);

}

public List<Country> searchCountriesByPartialName(String name) {

return countryRepository.findByNameContainingIgnoreCase(name);

}

}

**OrmLearnApplication.java:**

package com.cognizant.ormlearn;

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.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

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

@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);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

testAddCountry();

testFindCountryByCode("ZZ");

testUpdateCountry();

testSearchCountriesByName("land");

testDeleteCountry();

testFindCountryByCodeIndia();

testAddCountryNew();

testUpdateCountryNew();

testDeleteAddedCountry();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

LOGGER.debug("All countries: {}", countries);

LOGGER.info("End");`}

private static void testFindCountryByCode(String code) {

try {

Country country = countryService.findCountryByCode(code);

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

} catch (CountryNotFoundException e) {

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

}

}

private static void testAddCountry() {

Country newCountry = new Country();

newCountry.setCode("ZZ");

newCountry.setName("Testland");

countryService.addCountry(newCountry);

LOGGER.info("Country added: {}", newCountry);

}

private static void testUpdateCountry() {

try {

countryService.updateCountry("ZZ", "Updated Testland");

LOGGER.info("Country updated.");

} catch (CountryNotFoundException e) {

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

}

}

private static void testDeleteCountry() {

try {

countryService.deleteCountry("ZZ");

LOGGER.info("Country deleted.");

} catch (CountryNotFoundException e) {

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

}

private static void testSearchCountriesByName(String keyword) {

List<Country> results = countryService.searchCountriesByPartialName(keyword);

LOGGER.debug("Search results for '{}': {}", keyword, results);

}

private static void testFindCountryByCodeIndia() {

try {

LOGGER.info("Start");

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

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

LOGGER.info("End");

} catch (CountryNotFoundException e) {

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

}

}

private static void testAddCountryNew() {

Country newCountry = new Country();

newCountry.setCode("XY");

newCountry.setName("NewTestCountry");

countryService.addCountry(newCountry);

LOGGER.info(" New Country added: {}", newCountry);

try {

Country foundCountry = countryService.findCountryByCode("XY");

LOGGER.info("Found added Country: {}", foundCountry);

} catch (CountryNotFoundException e) {

LOGGER.error(" Country not found after add: {}", e.getMessage());

}

}

private static void testDeleteAddedCountry() {

try {

LOGGER.info(" Deleting test country 'XY'...");

countryService.deleteCountry("XY");

LOGGER.info(" Country 'XY' deleted.");

try {

countryService.findCountryByCode("XY");

} catch (CountryNotFoundException e) {

LOGGER.info(" Confirmed: Country 'XY' no longer exists.");

}

} catch (CountryNotFoundException e) {

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

}

}

private static void testUpdateCountryNew() {

try {

LOGGER.info(" Starting update test...");

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

LOGGER.info(" Country updated to new name.");

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

LOGGER.info(" Fetched after update: {}", updated);

} catch (CountryNotFoundException e) {

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

}

}

}

**Output:**

