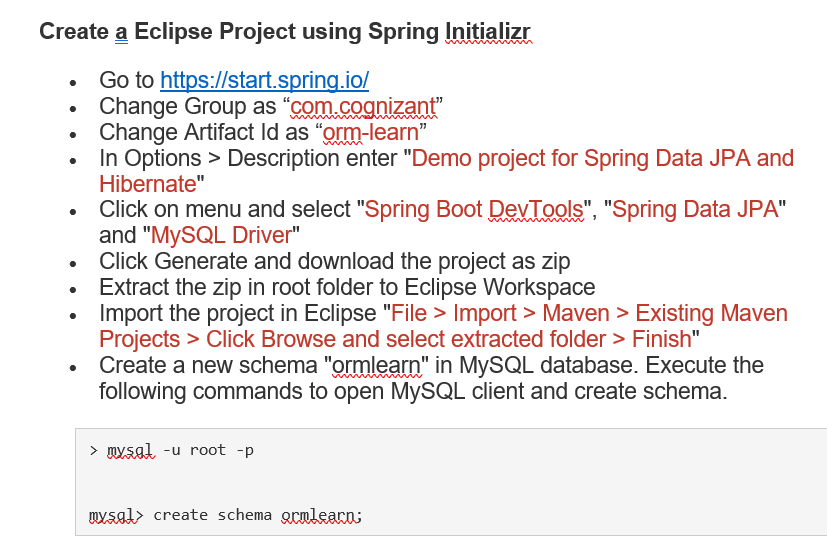
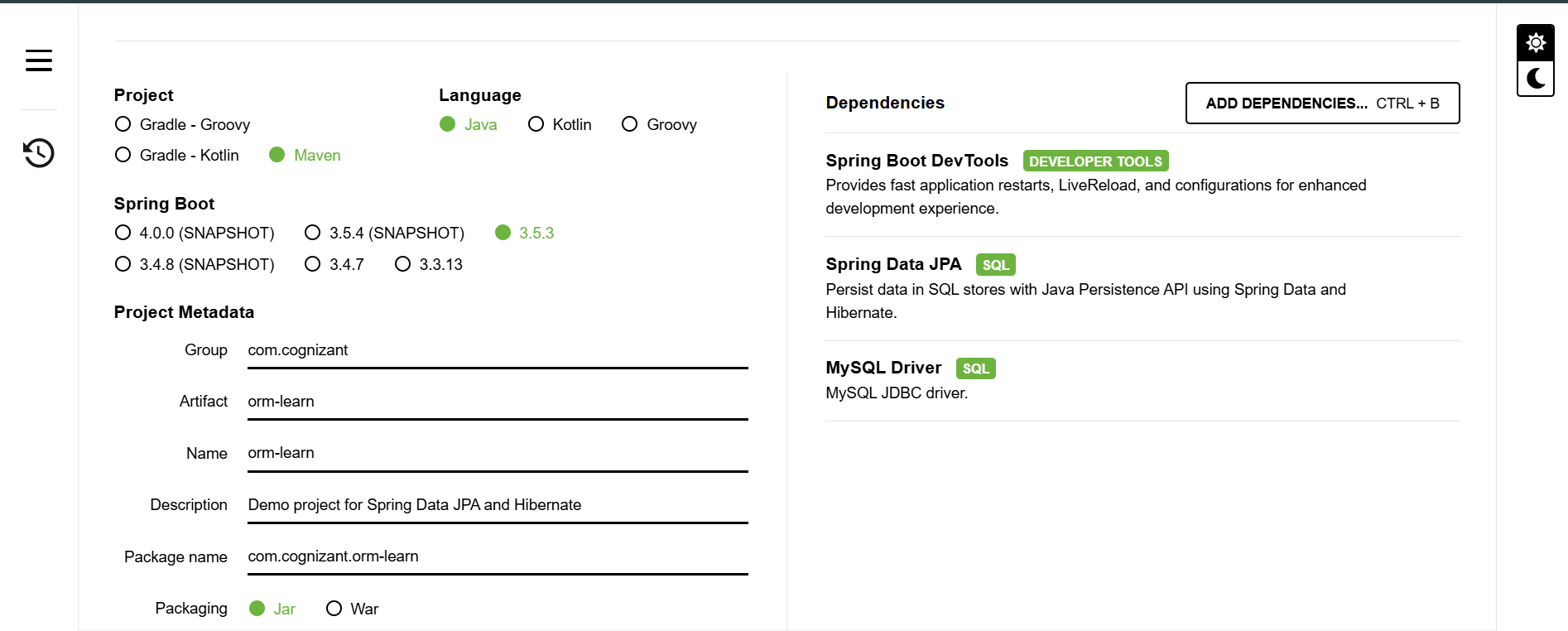
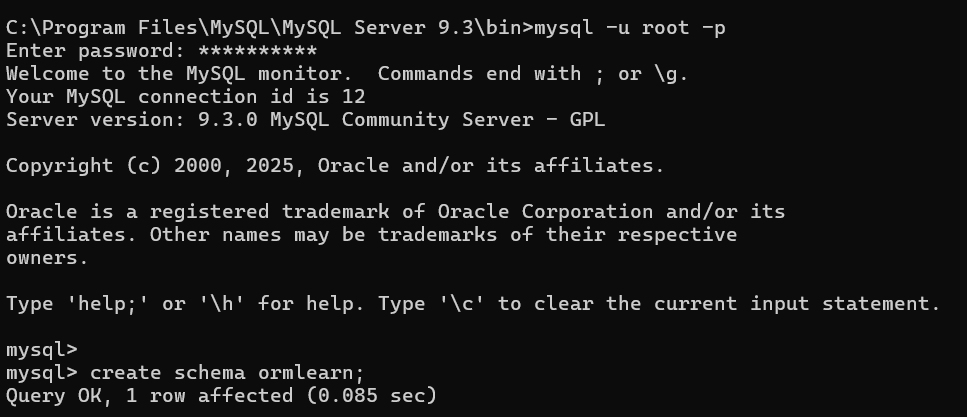
**WEEK 3:Spring Data JPA with Spring Boot Hibernate(Solutions)**

**Spring Data JPA - Quick Example**

***NOTE : - (I AM USING IntelliJ IDEA INSTEAD OF ECLIPSE)***  




**OUTPUT :**



* In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

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

# Log pattern

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

# Database configuration

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

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

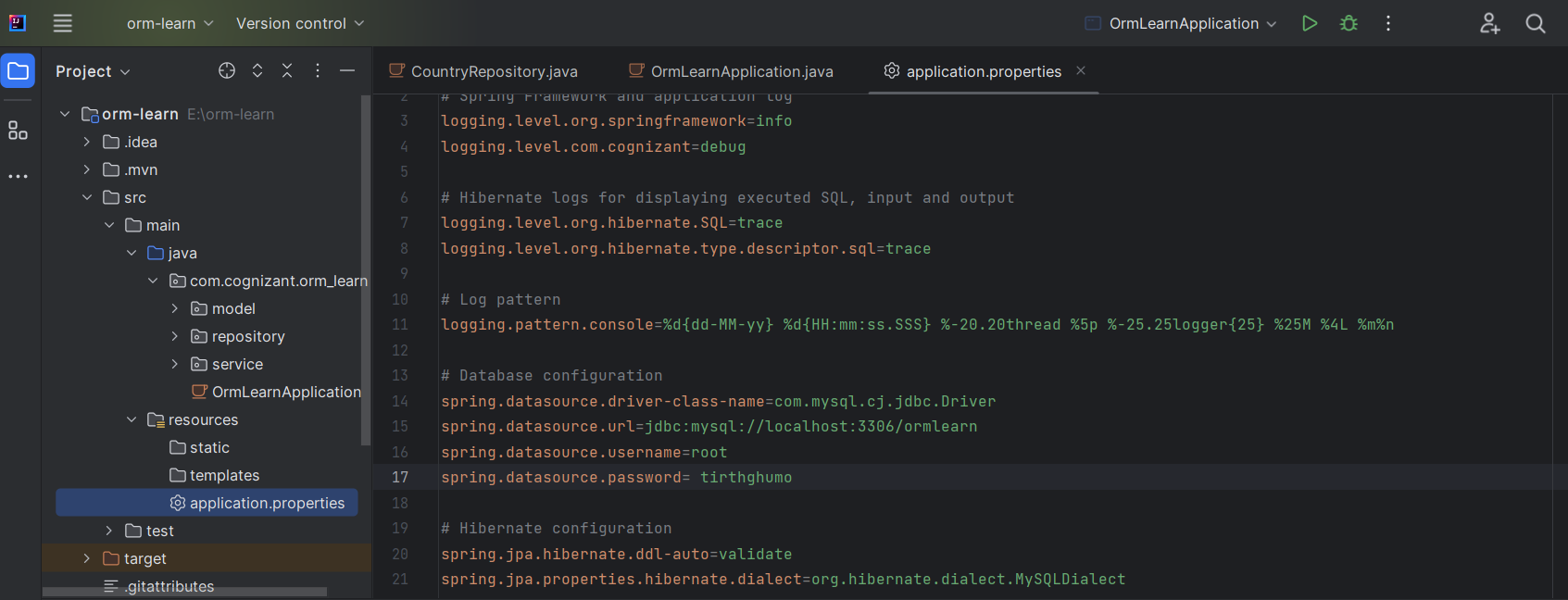
spring.datasource.username=root

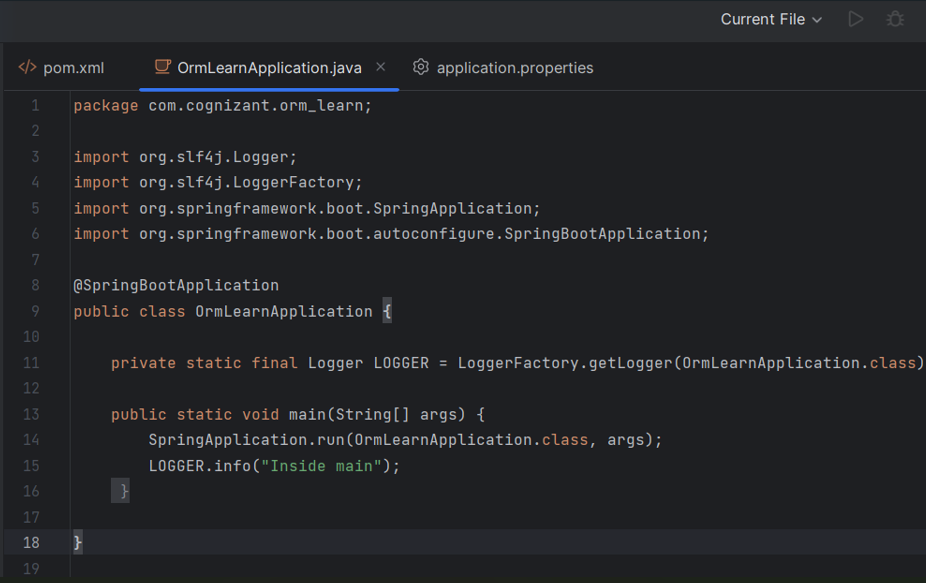
spring.datasource.password=root

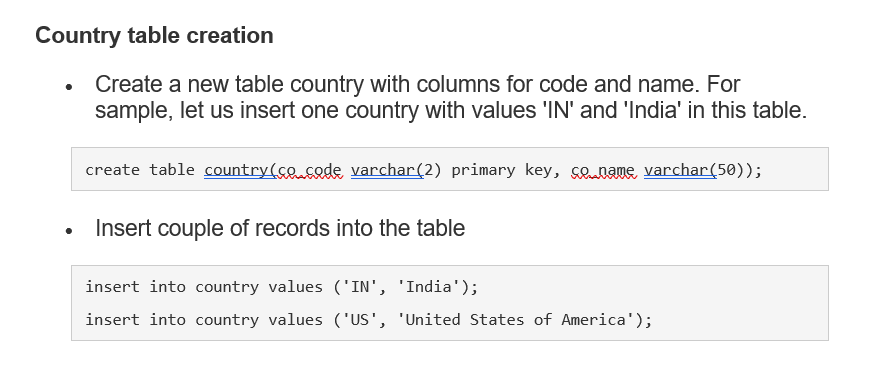
# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

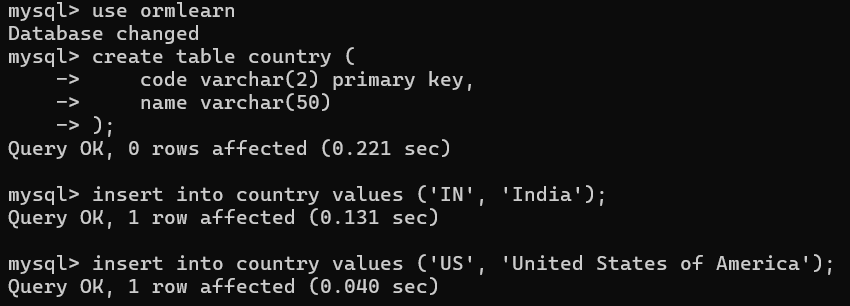
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect







**OUTPUT :**



**Persistence Class - com.cognizant.orm-learn.model.Country**

* Open Eclipse with orm-learn project
* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.
* Include @Entity and @Table at class level
* Include @Column annotations in each getter method specifying the column name.

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="country")

public class Country {

  @Id

    @Column(name="code")

    private String code;

    @Column(name="name")

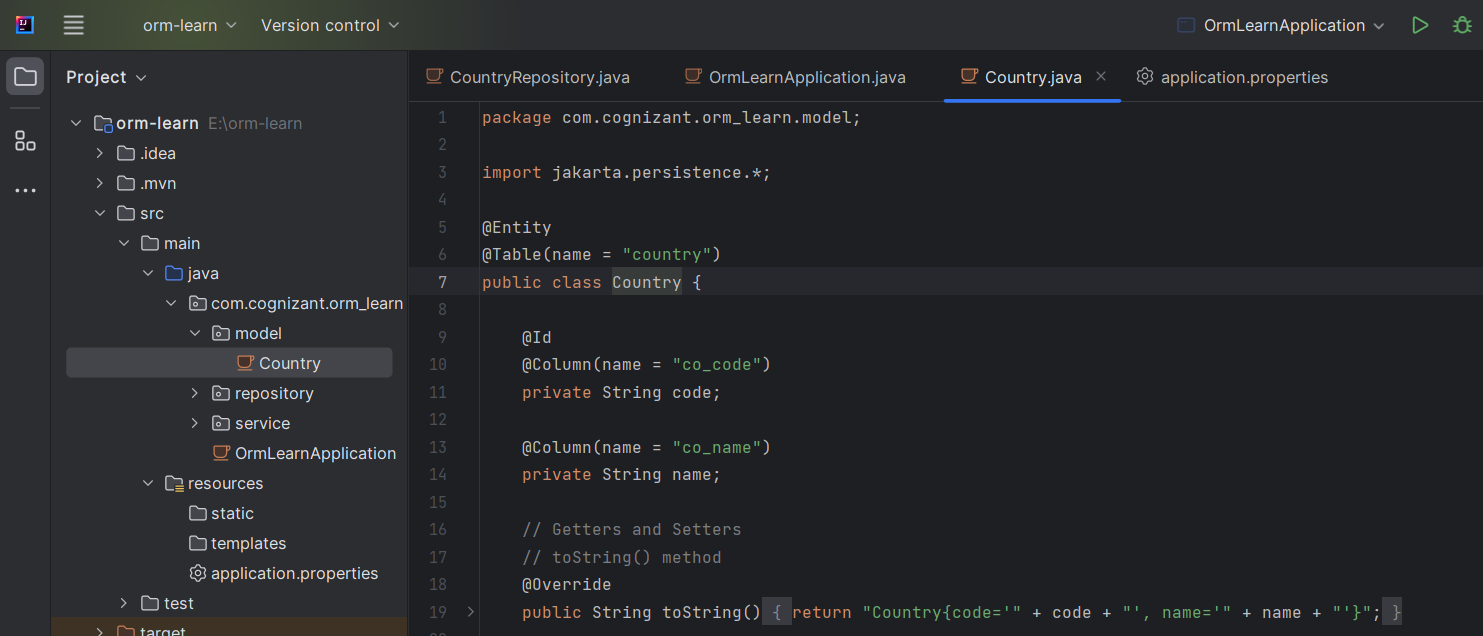
    private String name;

// getters and setters

  // toString()

}

**OUTPUT :**



**Repository Class - com.cognizant.orm-learn.CountryRepository**

* Create new package com.cognizant.orm-learn.repository
* Create new interface named CountryRepository that extends JpaRepository<Country, String>
* Define @Repository annotation at class level

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

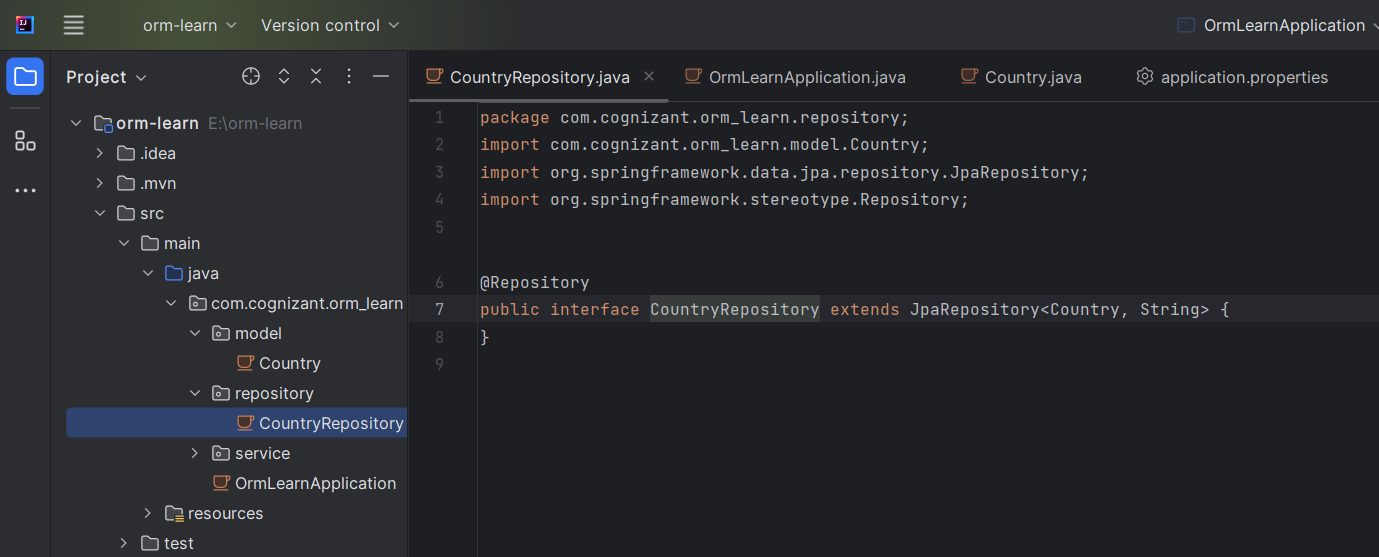
import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

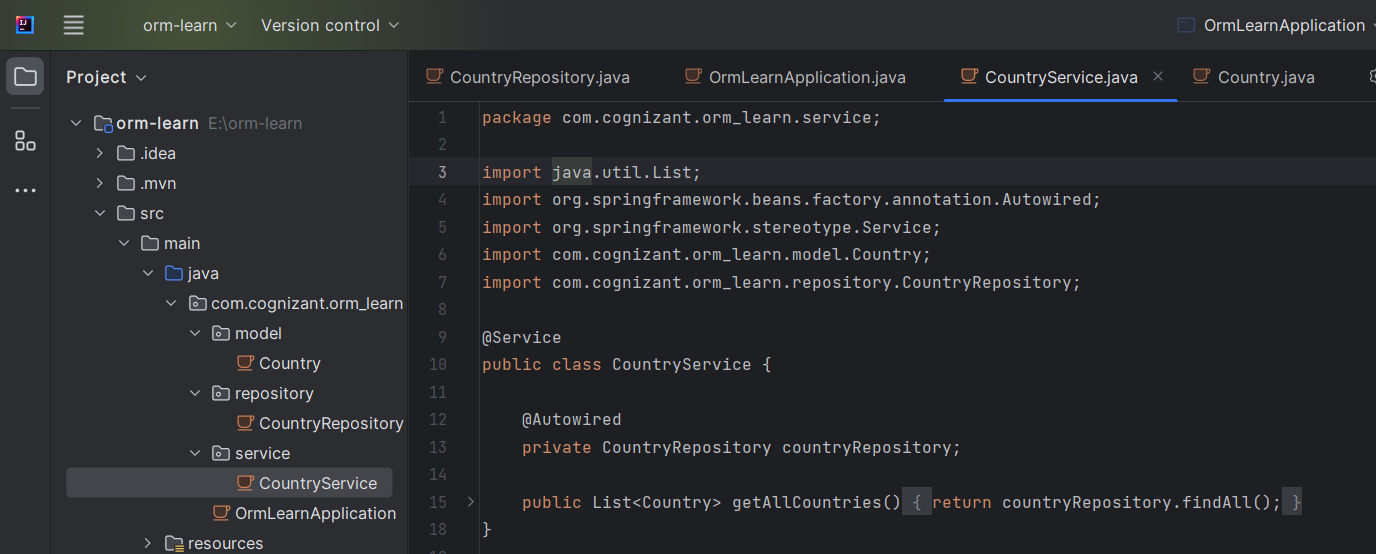
public interface CountryRepository extends JpaRepository<Country, String> {

}



**Service Class - com.cognizant.orm-learn.service.CountryService**

* Create new package com.cognizant.orm-learn.service
* Create new class CountryService
* Include @Service annotation at class level
* Autowire CountryRepository in CountryService
* Include new method getAllCountries() method that returns a list of countries.
* Include @Transactional annotation for this method
* In getAllCountries() method invoke countryRepository.findAll() method and return the result



**Testing in OrmLearnApplication.java**

* Include a static reference to CountryService in OrmLearnApplication class

private static CountryService countryService;

* Define a test method to get all countries from service.

    private static void testGetAllCountries() {

        LOGGER.info("Start");

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

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

        LOGGER.info("End");

    }

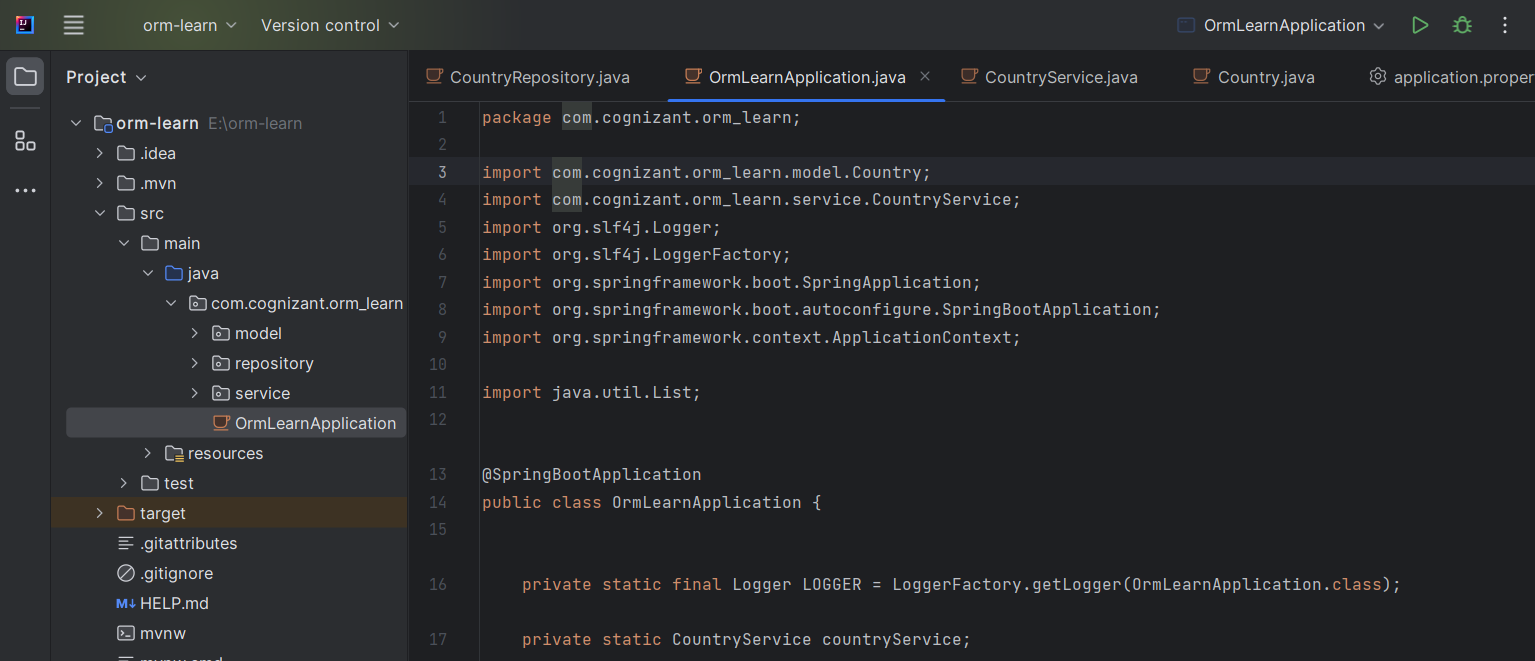
* Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

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

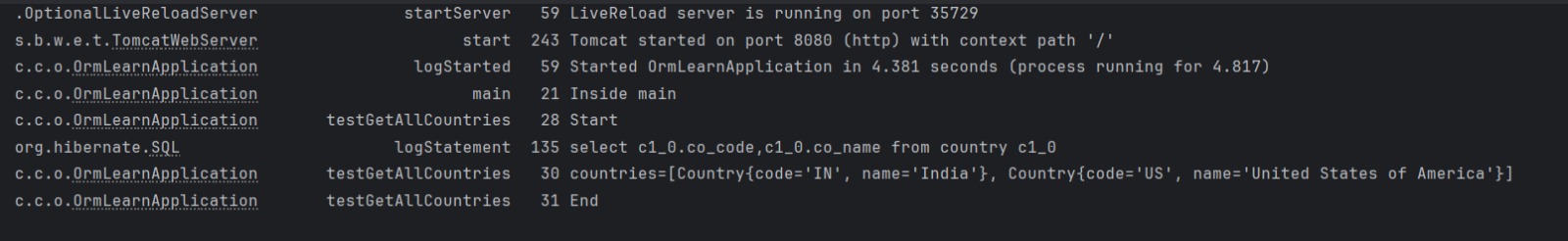
        countryService = context.getBean(CountryService.class);

        testGetAllCountries();

* Execute main method to check if data from ormlearn database is retrieved.





**OUTPUT :**

**Difference between JPA, Hibernate and Spring Data JPA**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Aspect |  |  | JPA |  |  | Hibernate | Spring Data JPA |
| Type |  |  | Specification (API) |  |  | Implementation (ORM/JPA Provider) | Abstraction/Framework over JPA Provider |
| Purpose |  |  | Defines how Java objects map to DB |  |  | Implements JPA, adds extra ORM features | Reduces boilerplate, auto-generates code |
| Standalone Usage |  |  | No (needs implementation) |  |  | Yes | No (needs JPA provider & Spring) |
| Package |  |  | javax.persistence / jakarta.persistence |  |  | org.hibernate | org.springframework.data.jpa.repository |
| Query Language |  |  | JPQL |  |  | HQL (and JPQL) | Derived queries, JPQL, native SQL |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Feature |  |  | JPA |  |  | Hibernate | Spring Data JPA |
| What is it? |  |  | Specification (API) |  |  | Implementation (ORM/JPA Provider) | Abstraction/Framework over JPA Provider |
| Can be used standalone? |  |  | No |  |  | Yes | No |
| Boilerplate code |  |  | Moderate |  |  | More (if using native API) | Minimal |
| Querying |  |  | JPQL |  |  | HQL, JPQL, Criteria | Derived queries, JPQL, @Query |
| CRUD Operations |  |  | EntityManager |  |  | Session, EntityManager | JpaRepository, CrudRepository |
| Integration |  |  | With any JPA provider |  |  | Standalone or with JPA | With Spring ecosystem |
| Use Case |  |  | Standard persistence logic |  |  | Advanced ORM features, fine control | Rapid development, less boilerplate |

**1. JPA (Java Persistence API)**

* **What it is:** A specification that defines how Java objects are persisted to relational databases. It does not provide actual code, only interfaces and annotations[[1]](#fn1)[[2]](#fn2)[[3]](#fn3).
* **Code Example:**

@Entity  
public class User {  
 @Id  
 @GeneratedValue  
 private Long id;  
 private String name;  
}  
// Using JPA EntityManager  
EntityManager em = ...;  
em.persist(new User("John"));

**2. Hibernate**

* **What it is:** The most popular **JPA implementation** (provider). It implements the JPA interfaces and adds extra features (caching, HQL, advanced mappings)[[1]](#fn1)[[4]](#fn4)[[5]](#fn5)[[2]](#fn2)[[3]](#fn3).
* **Code Example:**

// Using Hibernate Session API (native Hibernate, not pure JPA)  
Session session = sessionFactory.openSession();  
session.beginTransaction();  
session.save(new User("John"));  
session.getTransaction().commit();  
session.close();

**3. Spring Data JPA**

* **What it is:** A **Spring framework module** that sits on top of JPA (and a provider like Hibernate). It **reduces boilerplate** by providing repository interfaces and auto-generating queries from method names[[1]](#fn1)[[4]](#fn4)[[5]](#fn5)[[6]](#fn6)[[3]](#fn3).
* **Code Example:**

public interface UserRepository extends JpaRepository<User, Long> {  
 List<User> findByName(String name);  
}  
// Usage in a Spring service  
@Autowired  
UserRepository userRepository;  
userRepository.save(new User("John"));  
List<User> users = userRepository.findByName("John");

==============================================================================