**WEEK-3**

**Implement services for managing Country**

**Implementing Country Management Services using Spring Data JPA**

**Overview**

This document outlines the implementation of core functionalities required for managing country data within a Spring Boot application using Spring Data JPA. The following features will be implemented as services:

* Find a country by its country code
* Add a new country
* Update existing country details
* Delete a country
* Search countries by partial country name

**Hibernate DDL-Auto Configuration**

The ddl-auto property in Hibernate defines how Hibernate should behave when interacting with the database schema:

| **Value** | **Description** |
| --- | --- |
| create | Drops existing tables and data, then recreates the tables. |
| validate | Validates that the schema exists and matches the entities. Throws error if not. |
| update | Updates the database schema to match the entities. Adds missing tables or columns. |
| create-drop | Creates tables on startup and drops them on shutdown. |

**Recommended Setting:**

spring.jpa.hibernate.ddl-auto=validate

This ensures that Hibernate verifies the structure without modifying the schema or dropping existing data.

**Country Table Population**

Before implementing the service methods, we need to ensure that the country table is populated with the correct data.

**Step 1: Clear existing country data**

DELETE FROM country;

**Step 2: Insert countries**

Use the following SQL script to populate the country table with actual country data. This list includes most countries in the world:

insert into country (co\_code, co\_name) values ("AF", "Afghanistan");

insert into country (co\_code, co\_name) values ("AL", "Albania");

insert into country (co\_code, co\_name) values ("DZ", "Algeria");

insert into country (co\_code, co\_name) values ("AS", "American Samoa");

insert into country (co\_code, co\_name) values ("AD", "Andorra");

insert into country (co\_code, co\_name) values ("AO", "Angola");

insert into country (co\_code, co\_name) values ("AI", "Anguilla");

insert into country (co\_code, co\_name) values ("AQ", "Antarctica");

insert into country (co\_code, co\_name) values ("AG", "Antigua and Barbuda");

insert into country (co\_code, co\_name) values ("AR", "Argentina");

insert into country (co\_code, co\_name) values ("AM", "Armenia");

insert into country (co\_code, co\_name) values ("AW", "Aruba");

insert into country (co\_code, co\_name) values ("AU", "Australia");

insert into country (co\_code, co\_name) values ("AT", "Austria");

insert into country (co\_code, co\_name) values ("AZ", "Azerbaijan");

-- (continue for remaining countries...)

**Note:** If needed, the full SQL script containing all countries can be imported via MySQL Workbench or any database GUI tool.

**Next Steps**

Once the database is populated, the following tasks can be implemented in the CountryService class:

**1. Find a Country by Code**

public Country findCountryByCode(String code);

**2. Add a New Country**

public void addCountry(Country country);

**3. Update an Existing Country**

public void updateCountry(String code, String newName);

**4. Delete a Country**

public void deleteCountry(String code);

**5. Find Countries Matching Partial Name**

public List<Country> findByNameContaining(String keyword);

Each of these methods can internally use CountryRepository, which should extend JpaRepository<Country, String> and leverage Spring Data JPA's built-in query capabilities.

**Conclusion**

This setup provides a strong foundation for building a country management module. With Spring Data JPA, the development of CRUD and query operations becomes streamlined and reduces the need for boilerplate code.  
These services can now be tested and further integrated into REST controllers or front-end clients as needed.