**Spring Data JPA Hands-On Projects**

# Project Description

This document contains two hands-on projects using Spring Data JPA. The goal is to help understand how to implement basic CRUD operations and explore the differences between JPA, Hibernate, and Spring Data JPA.  
  
Both projects are designed to run as Spring Boot applications and can be executed in Eclipse IDE with Maven support.

# Project Structure (Common for Both)

spring-data-jpa-handson/  
├── src/  
│ ├── main/  
│ │ ├── java/  
│ │ │ └── com/  
│ │ │ └── example/  
│ │ │ └── country/  
│ │ │ ├── controller/  
│ │ │ ├── entity/  
│ │ │ ├── repository/  
│ │ │ └── service/  
│ └── resources/  
│ └── application.properties  
├── pom.xml  
└── SpringDataJpaHandsonApplication.java

# Execution Steps in Eclipse

1. Open Eclipse IDE and select a workspace.  
2. Go to File → New → Maven Project.  
3. Select Archetype: quickstart or spring-boot-starter-parent if available.  
4. Enter GroupId: com.example and ArtifactId: spring-data-jpa-handson.  
5. After project creation, replace the generated structure with the one given in this document.  
6. Add dependencies to pom.xml (Spring Boot Starter Data JPA, Web, H2/MySQL, etc.).  
7. Add code files in the specified package structure.  
8. Update application.properties for DB connection.  
9. Right-click the project → Run As → Spring Boot App or Java Application.

# Project 1: Quick Example with CRUD Operations

## Entity: Country.java

package com.example.country.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
  
@Entity  
public class Country {  
 @Id  
 private String code;  
 private String name;  
  
 // Getters and Setters  
}

## Repository: CountryRepository.java

package com.example.country.repository;  
  
import com.example.country.entity.Country;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface CountryRepository extends JpaRepository<Country, String> {}

## Service: CountryService.java

package com.example.country.service;  
  
import com.example.country.entity.Country;  
import com.example.country.repository.CountryRepository;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class CountryService {  
 private final CountryRepository repository;  
  
 public CountryService(CountryRepository repository) {  
 this.repository = repository;  
 }  
  
 public List<Country> getAllCountries() {  
 return repository.findAll();  
 }  
}

## Controller: CountryController.java

package com.example.country.controller;  
  
import com.example.country.entity.Country;  
import com.example.country.service.CountryService;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/countries")  
public class CountryController {  
 private final CountryService service;  
  
 public CountryController(CountryService service) {  
 this.service = service;  
 }  
  
 @GetMapping  
 public List<Country> getCountries() {  
 return service.getAllCountries();  
 }  
  
 @GetMapping("/{code}")  
 public Country getCountryByCode(@PathVariable String code) {  
 return service.getCountryByCode(code);  
 }  
}

# Project 2: Add Method to Find by Country Code

## CountryRepository.java (Addition)

Country findByCode(String code);

## CountryService.java (Addition)

public Country getCountryByCode(String code) {  
 return repository.findByCode(code);  
}

# Summary

These two projects help you understand how to work with Spring Data JPA for real-world scenarios like managing entities and performing dynamic queries.