# Spring Data JPA - Quick Example

### File: DemoApplication.java

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.web.bind.annotation.\*;

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

import jakarta.persistence.\*;

import org.springframework.beans.factory.annotation.Autowired; import java.util.List;

@SpringBootApplication public class DemoApplication {

public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args);

}

}

@Entity class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name; private String email;

public User() {} // default constructor

// Getters & Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; } public String getName() { return name; }

public void setName(String name) { this.name = name; } public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

}

@Repository

interface UserRepository extends JpaRepository<User, Long> {}

@RestController @RequestMapping("/users") class UserController {

@Autowired

private UserRepository userRepository;

@PostMapping

public User createUser(@RequestBody User user) { return userRepository.save(user);

}

@GetMapping

public List<User> getAllUsers() { return userRepository.findAll();

}

@GetMapping("/{id}")

public User getUserById(@PathVariable Long id) { return userRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public User updateUser(@PathVariable Long id, @RequestBody User newUser) { return userRepository.findById(id).map(user -> {

user.setName(newUser.getName()); user.setEmail(newUser.getEmail()); return userRepository.save(user);

}).orElse(null);

}

@DeleteMapping("/{id}")

public void deleteUser(@PathVariable Long id) { userRepository.deleteById(id);

}

}

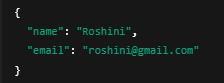
# application.properties

spring.datasource.url=jdbc:h2:mem:testdb spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=sa spring.datasource.password= spring.jpa.hibernate.ddl-auto=update spring.h2.console.enabled=true

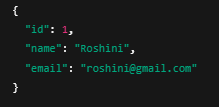
spring.jpa.show-sql=true

# Output (in Postman) :

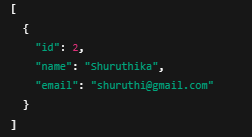
**POST /users**

****

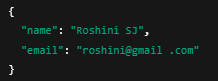
**Response:**

****

**GET /users**

****

**PUT /users/1**

****

# Implement services for managing Country

### Create ,

1. Country entity
2. CountryRepository
3. CountryService
4. CountryController
5. application.properties

### Country Entity

package com.example.demo.model; import jakarta.persistence.\*; @Entity

public class Country { @Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name; private String capital;

// Constructors public Country() {}

public Country(String name, String capital) { this.name = name;

this.capital = capital;

}

// Getters & Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; } public String getName() { return name; }

public void setName(String name) { this.name = name; } public String getCapital() { return capital; }

public void setCapital(String capital) { this.capital = capital; }

}

### CountryRepository

package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository; import com.example.demo.model.Country;

public interface CountryRepository extends JpaRepository<Country, Long> {}

### CountryService

package com.example.demo.service; import com.example.demo.model.Country;

import com.example.demo.repository.CountryRepository;

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; public List<Country> getAllCountries() {

return countryRepository.findAll();

}

public Country getCountryById(Long id) {

return countryRepository.findById(id).orElse(null);

}

public Country addCountry(Country country) { return countryRepository.save(country);

}

public Country updateCountry(Long id, Country updatedCountry) { Optional<Country> existing = countryRepository.findById(id); if (existing.isPresent()) {

Country c = existing.get(); c.setName(updatedCountry.getName()); c.setCapital(updatedCountry.getCapital()); return countryRepository.save(c);

}

return null;

}

public void deleteCountry(Long id) { countryRepository.deleteById(id);

}

}

### CountryController

package com.example.demo.controller;

import com.example.demo.model.Country;

import com.example.demo.service.CountryService;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.\*;

import java.util.List; @RestController @RequestMapping("/countries") public class CountryController {

@Autowired

private CountryService countryService; @GetMapping

public List<Country> getAllCountries() { return countryService.getAllCountries();

}

@GetMapping("/{id}")

public Country getCountryById(@PathVariable Long id) { return countryService.getCountryById(id);

}

@PostMapping

public Country addCountry(@RequestBody Country country) { return countryService.addCountry(country);

}

@PutMapping("/{id}")

public Country updateCountry(@PathVariable Long id, @RequestBody Country country) { return countryService.updateCountry(id, country);

}

@DeleteMapping("/{id}")

public void deleteCountry(@PathVariable Long id) { countryService.deleteCountry(id);

}

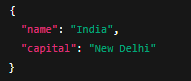
}

1. **application.properties** spring.datasource.url=jdbc:h2:mem:testdb spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=sa spring.datasource.password= spring.jpa.hibernate.ddl-auto=update spring.h2.console.enabled=true

# Output Request via Postman

### Add Country POST

<http://localhost:8080/countries>



### Get All Countries GET

<http://localhost:8080/countries>

# Difference between JPA, Hibernate and Spring Data JPA

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature / Term** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| Type | Specification (interface) | Implementation of JPA | Abstraction built on top of JPA |
| What it is | API that defines ORM for Java | Framework that implements JPA | Spring module that simplifies JPA usage |
| Provider | Defined by Oracle (as part of Java EE) | Provided by Hibernate (Red Hat) | Provided by Spring Framework |
| Setup Requiremen t | Needs an implementation like Hibernate | Needs configuration of SessionFactory, Transaction, etc. | Very minimal setup with Spring Boot; automatic configuration |
| Usage Level | Low-level (more manual coding) | Medium-level (manual config, but powerful) | High-level (boilerplate-free, auto-query generation) |
| Example (Create) | em.persist(new Country("India", "Delhi")); | session.save(new Country("India", "Delhi")); | countryRepository.save(new Country("India", "Delhi")); |
| Example (Read) | em.find(Country.clas s, 1L); | session.get(Country.clas s, 1L); | countryRepository.findById(1L  ); |
| Querying | JPQL, Criteria API | JPQL, Criteria, Native SQL | Method naming (findByName), @Query, paging, sorting |
| Boilerplate Code | More | Less than JPA | Very minimal, most is auto- generated |
| Annotations Used | @Entity, @Id, @OneToMany, @GeneratedValue | Same JPA annotations + additional Hibernate- specific annotations (e.g., @Where) | Uses JPA annotations + Spring's @Repository, @Service, etc. |
| Ease of Testing | Complex | Easier than raw JPA | Simplified using Spring Boot Test |
| Integration | Can integrate with any ORM framework | Mostly used standalone or as JPA provider | Deeply integrated with Spring ecosystem |

**Find a country based on country code**

### CountryApp.java

package com.example.country;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication public class CountryApp {

public static void main(String[] args) { SpringApplication.run(CountryApp.class, args);

}

}

### Country.java (Model)

package com.example.country.model; import jakarta.persistence.\*;

@Entity

public class Country {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name; private String capital; private String countryCode;

public Country() {}

public Country(String name, String capital, String countryCode) { this.name = name;

this.capital = capital; this.countryCode = countryCode;

}

// Getters and Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; } public String getCapital() { return capital; }

public void setCapital(String capital) { this.capital = capital; }

public String getCountryCode() { return countryCode; }

public void setCountryCode(String countryCode) { this.countryCode = countryCode; }

}

### CountryRepository.java

package com.example.country.repository; import com.example.country.model.Country;

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

public interface CountryRepository extends JpaRepository<Country, Long> { Country findByCountryCode(String countryCode);

}

### CountryService.java

package com.example.country.service; import com.example.country.model.Country;

import com.example.country.repository.CountryRepository; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

public Country getCountryById(Long id) {

return countryRepository.findById(id).orElse(null);

}

public Country getCountryByCode(String code) { return countryRepository.findByCountryCode(code);

}

public Country addCountry(Country country) { return countryRepository.save(country);

}

public Country updateCountry(Long id, Country updatedCountry) { return countryRepository.findById(id).map(c -> {

c.setName(updatedCountry.getName()); c.setCapital(updatedCountry.getCapital()); c.setCountryCode(updatedCountry.getCountryCode()); return countryRepository.save(c);

}).orElse(null);

}

public void deleteCountry(Long id) { countryRepository.deleteById(id);

}

}

### CountryController.java

package com.example.country.controller;

import com.example.country.model.Country;

import com.example.country.service.CountryService;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.\*;

import java.util.List; @RestController @RequestMapping("/countries") public class CountryController {

@Autowired

private CountryService countryService;

@GetMapping

public List<Country> getAllCountries() { return countryService.getAllCountries();

}

@GetMapping("/{id}")

public Country getCountryById(@PathVariable Long id) { return countryService.getCountryById(id);

}

@GetMapping("/code/{code}")

public Country getCountryByCode(@PathVariable String code) { return countryService.getCountryByCode(code);

}

@PostMapping

public Country addCountry(@RequestBody Country country) { return countryService.addCountry(country);

}

@PutMapping("/{id}")

public Country updateCountry(@PathVariable Long id, @RequestBody Country country) { return countryService.updateCountry(id, country);

}

@DeleteMapping("/{id}")

public void deleteCountry(@PathVariable Long id) { countryService.deleteCountry(id);

}

}

### application.properties

spring.datasource.url=jdbc:h2:mem:countrydb spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=sa spring.datasource.password= spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true spring.h2.console.enabled=true

**Output Requests in Postman**

## Add Country

### POST

<http://localhost:8080/countries>



## Get Country by Country Code

### GET

<http://localhost:8080/countries/code/IN>

# Add a new country

**API Endpoint to Add a Country**

## Method:

POST /countries

### CountryController.java

@PostMapping

public Country addCountry(@RequestBody Country country) { return countryService.addCountry(country);

}

### And in the service:

public Country addCountry(Country country) { return countryRepository.save(country);

}

**Output :**

****

**Demonstrate implementation of Query Methods feature of Spring Data JPA**

**User Entity**

### User.java (Entity)

package com.example.demo.model; import jakarta.persistence.\*; @Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name; private String email; private String city;

// Constructors public User() {}

public User(String name, String email, String city) { this.name = name;

this.email = email; this.city = city;

}

// Getters and Setters

// ...

}

### UserRepository.java (with Query Methods)

package com.example.demo.repository; import com.example.demo.model.User;

import org.springframework.data.jpa.repository.JpaRepository; import java.util.List;

public interface UserRepository extends JpaRepository<User, Long> {

// Find by name

List<User> findByName(String name);

// Find by city

List<User> findByCity(String city);

// Find by email ending with a domain

List<User> findByEmailEndingWith(String domain);

// Find users by name ignoring case

List<User> findByNameIgnoreCase(String name);

// Find by name containing keyword

List<User> findByNameContaining(String keyword);

}

### UserService.java

package com.example.demo.service; import com.example.demo.model.User;

import com.example.demo.repository.UserRepository;

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

import java.util.List; @Service

public class UserService {

@Autowired

private UserRepository userRepository;

public List<User> getByName(String name) { return userRepository.findByName(name);

}

public List<User> getByCity(String city) { return userRepository.findByCity(city);

}

public List<User> searchByEmailDomain(String domain) { return userRepository.findByEmailEndingWith(domain);

}

public List<User> searchByNameKeyword(String keyword) { return userRepository.findByNameContaining(keyword);

}

}

### UserController.java

package com.example.demo.controller; import com.example.demo.model.User;

import com.example.demo.service.UserService;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController @RequestMapping("/users") public class UserController {

@Autowired

private UserService userService;

@GetMapping("/name/{name}")

public List<User> getByName(@PathVariable String name) { return userService.getByName(name);

}

@GetMapping("/city/{city}")

public List<User> getByCity(@PathVariable String city) { return userService.getByCity(city);

}

@GetMapping("/email/domain")

public List<User> getByEmailDomain(@RequestParam String domain) { return userService.searchByEmailDomain(domain);

}

@GetMapping("/search")

public List<User> searchByName(@RequestParam String keyword) { return userService.searchByNameKeyword(keyword);

}

}

**JSON Response**

****

# Demonstrate implementation of O/R Mapping

One Student → Many Courses

### Student.java

package com.example.demo.model; import jakarta.persistence.\*;

import java.util.List;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name;

@OneToMany(mappedBy = "student", cascade = CascadeType.ALL) private List<Course> courses;

// Getters and Setters

}

### Course.java

package com.example.demo.model; import jakarta.persistence.\*; @Entity

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String courseName; @ManyToOne

@JoinColumn(name = "student\_id") private Student student;

// Getters and Setters

}

### StudentRepository.java

package com.example.demo.repository; import com.example.demo.model.Student;

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

public interface StudentRepository extends JpaRepository<Student, Long> {}

### CourseRepository.java

package com.example.demo.repository; import com.example.demo.model.Course;

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

public interface CourseRepository extends JpaRepository<Course, Long> {}

### StudentController.java

package com.example.demo.controller; import com.example.demo.model.Student;

import com.example.demo.repository.StudentRepository;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController @RequestMapping("/students") public class StudentController {

@Autowired

private StudentRepository studentRepository;

@PostMapping

public Student addStudent(@RequestBody Student student) { return studentRepository.save(student);

}

@GetMapping

public List<Student> getAllStudents() { return studentRepository.findAll();

}

}

**JSON to Test in Postman POST /students**

# Output :

****

**Demonstrate writing Hibernate Query Language and Native Query**

### User.java

package com.example.demo.model; import jakarta.persistence.\*; @Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name; private String email;

// Getters and Setters

}

### UserRepository.java – Using HQL and Native Queries

package com.example.demo.repository; import com.example.demo.model.User;

import org.springframework.data.jpa.repository.\*;

import org.springframework.data.repository.query.Param; import org.springframework.stereotype.Repository;

import java.util.List; @Repository

public interface UserRepository extends JpaRepository<User, Long> {

// 1. HQL Query

@Query("SELECT u FROM User u WHERE u.name = :name") List<User> findUsersByNameHQL(@Param("name") String name);

// 2. Native SQL Query

@Query(value = "SELECT \* FROM user WHERE email = :email", nativeQuery = true) List<User> findUsersByEmailNative(@Param("email") String email);

// 3. Native query for partial name

@Query(value = "SELECT \* FROM user WHERE name LIKE %:keyword%", nativeQuery = true)

List<User> searchByNameNative(@Param("keyword") String keyword);

}

### UserService.java

package com.example.demo.service; import com.example.demo.model.User;

import com.example.demo.repository.UserRepository;

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

import java.util.List; @Service

public class UserService {

@Autowired

private UserRepository userRepository;

public List<User> getUsersByNameHQL(String name) { return userRepository.findUsersByNameHQL(name);

}

public List<User> getUsersByEmailNative(String email) { return userRepository.findUsersByEmailNative(email);

}

public List<User> searchByNameNative(String keyword) { return userRepository.searchByNameNative(keyword);

}

}

### UserController.java

package com.example.demo.controller; import com.example.demo.model.User;

import com.example.demo.service.UserService;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController @RequestMapping("/users") public class UserController {

@Autowired

private UserService userService;

@GetMapping("/hql/{name}")

public List<User> getByNameHQL(@PathVariable String name) { return userService.getUsersByNameHQL(name);

}

@GetMapping("/native/email")

public List<User> getByEmailNative(@RequestParam String email) { return userService.getUsersByEmailNative(email);

}

@GetMapping("/native/search")

public List<User> searchByName(@RequestParam String keyword) { return userService.searchByNameNative(keyword);

}

}

## HQL: Get users by name

### Request:

GET <http://localhost:8080/users/hql/John>

**Response (JSON):**

****

## Native: Get user by email

### Request:

GET [http://localhost:8080/users/native/email?email=john@gmail.com](http://localhost:8080/users/native/email?email=john%40gmail.com)

**Response:**



## Native: Search user by name keyword

### Request:

GET <http://localhost:8080/users/native/search?keyword=oh>

**Response:**

****