Spring Data JPA

Usar un repositorio como fuente de datos en un proyecto Spring Boot generalmente implica integrar Spring Data JPA (Java Persistence API) para gestionar las interacciones con la base de datos. Aquí tienes una guía paso a paso sobre cómo configurarlo:

1. Agregar Dependencias

Primero, necesitas agregar las dependencias necesarias a tu archivo `pom.xml` si estás usando Maven. Para Spring Data JPA, necesitarás lo siguiente:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

2. Configurar la Base de Datos

Configura los ajustes de tu base de datos en el archivo `application.properties`. Por ejemplo, si estás usando una base de datos H2 en memoria:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

3. Crear una Entidad

Define una clase de entidad que se mapea a una tabla de la base de datos. Usa anotaciones JPA para especificar el mapeo:

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // Getters y setters
}
```

4. Crear una Interfaz de Repositorio

Crea una interfaz de repositorio que extienda `JpaRepository`. Esta interfaz proporcionará operaciones CRUD para la entidad `User`:

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

public interface UserRepository extends JpaRepository<User, Long> {
    // Aquí se pueden definir métodos de consulta personalizados
}
```

5. Usar el Repositorio en un Servicio

Inyecta el repositorio en una clase de servicio para realizar operaciones en la base de datos:

```java
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> getAllUsers() {
        return userRepository.findAll();
    }

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

    public User saveUser(User user) {
        return userRepository.save(user);
    }

    public void deleteUser(Long id) {
        userRepository.deleteById(id);
    }
}
```

6. Crear un Controlador

Crea un controlador REST para exponer puntos finales para interactuar con la entidad `User`:

```
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
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @GetMapping("/{id}")
    public User getUserById(@PathVariable Long id) {
        return userService.getUserById(id);
```

```
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.saveUser(user);
    }

    @DeleteMapping("/{id}")
    public void deleteUser(@PathVariable Long id) {
        userService.deleteUser(id);
    }
}
```

7. Ejecutar la Aplicación

Ejecuta tu aplicación Spring Boot. Ahora deberías poder realizar operaciones CRUD en la entidad `User` usando los puntos finales REST.

Esta configuración proporciona una estructura básica para usar un repositorio como fuente de datos en un proyecto Spring Boot. Puedes extenderla aún más agregando consultas personalizadas, manejando excepciones e implementando lógica de negocio más compleja.