

# Programación Frontend y Backend

*BLOQUE SPRING*

Spring DATA

01

## Introducción



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



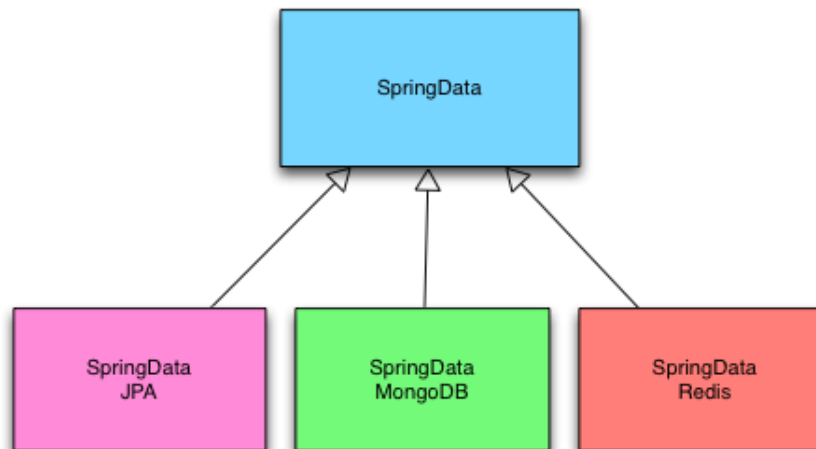
### Spring Data

Con Spring Data, los desarrolladores pueden interactuar con los datos utilizando una API consistente y coherente, independientemente del tipo de tecnología de almacenamiento subyacente que se utilice.

Spring Data también proporciona características como paginación, ordenamiento, consultas personalizadas y búsquedas por ejemplo, que hacen que sea fácil trabajar con datos en aplicaciones empresariales complejas. En general, Spring Data simplifica significativamente el trabajo con datos y hace que sea más fácil para los desarrolladores centrarse en la lógica de negocio en lugar de preocuparse por los detalles de acceso y manipulación de datos.

### Spring Data

Spring Data incluye soporte para una variedad de tecnologías de almacenamiento de datos, como JDBC, JPA, MongoDB, Cassandra, Redis, Neo4j y muchos otros.





02

## Configuración



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



```
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

```
<dependency>  
    <groupId>mysql</groupId>  
    <artifactId>mysql-connector-java</artifactId>  
    <version>8.0.11</version>  
</dependency>
```

## application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/ej_eoi?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.database-platform = org.hibernate.dialect.MySQL5Dialect
```



03

Uso



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro





### JPA Repository

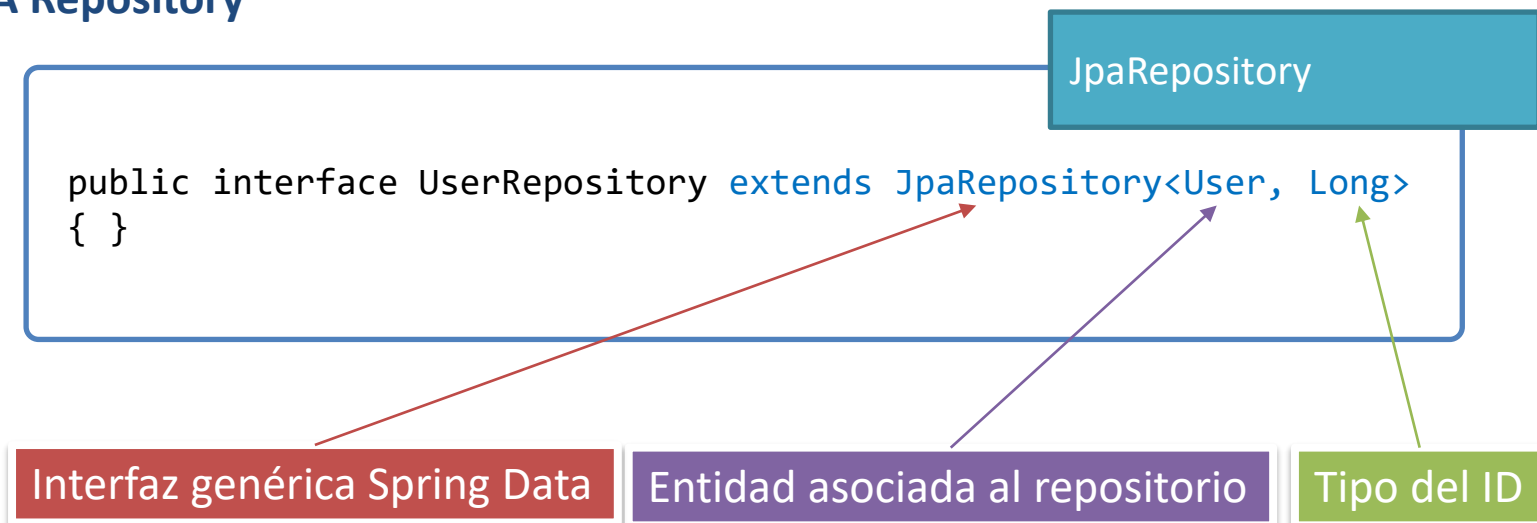
Permite la definición de objetos de acceso a datos simplemente especificando una interfaz que proporciona:

- Métodos CRUD genéricos (**C**reate-**R**ead-**U**ppdate-**D**elete).
- Métodos de consulta a partir de su nomenclatura.
- Métodos de consulta a partir de queries JPQL o mediante queries con nombre.

Todo esto **sin que sea necesario desarrollar la implementación.**

Spring lo hace automáticamente a partir de la interfaz.

## JPA Repository



### JPA Repository

extends ListCrudRepository<T,ID>, ListPagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>

1. <https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>

Me va a facilitar implementaciones por defecto para buscar conjuntos, ordenar o filtrar.

## JPA Repository: Ejemplo

Vamos a realizar un CRUD completo de una entidad Cliente con los campos id,nif y nombre, utilizando la clase JPARepository.

Introduciremos algunos clientes y comprobaremos que ahora podemos conseguir un findAll() sin tener que definir una JPQL.



### JPA Repository: Pageable

El Pageable de Spring Data es una interfaz utilizada para especificar opciones de paginación en las consultas de datos. Permite a los desarrolladores controlar la cantidad de datos que se recuperan de una base de datos en una sola consulta y proporciona información útil sobre la página actual, el tamaño de página y el ordenamiento.

En general, el objeto Pageable se utiliza para crear una consulta que devuelve un objeto Page que contiene la lista de resultados y la información de paginación. El objeto Pageable incluye métodos para acceder a la información de paginación, como la página actual, el tamaño de página, el ordenamiento y la dirección de ordenamiento.

## JPA Repository: Pageable , ejemplo:

Vamos a implementar un objeto de tipo pageable y pedirle que me devuelva únicamente 3 clientes.

Modifier and Type	Method	Description
Page<T>	<code>findAll(Pageable pageable)</code>	Returns a Page of entities meeting the paging restriction provided in the Pageable object.

[https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html#findAll\(org.springframework.data.domain.Pageable\)](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html#findAll(org.springframework.data.domain.Pageable))

## JPA Repository : MethodQuerys

Podemos añadir en la interfaz métodos con predicados que se convertiran en JPQL y así evitar tener que utilizar querys para consultas (SOLO CONSULTAS) que se salgan de las estándar que nos facilita JPARepository.

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    Page<User> findByLastName(String lastName, Pageable pageable);  
  
}
```

## JPA Repository : MethodQuerys

### A tener en cuenta:

Prefijos: find...By, read...By, query...By, count...By y get...By

### Añadir antes del primer By:

First: Devuelve solo el primer elemento de la lista. **findFirstBy**

Top + Número: Devuelve el número de elementos indicado: **findTop3By**

Distinct: Nos permite seleccionar un único resultado. **findTitleDistinctBy**



Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1

StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1(parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1(parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1(parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	... where UPPER(x.firstname) = UPPER(?1)

## ¡Pregunta! ¿qué hace cada query?:

```
public Todo findById(Long id);  
public List<Todo> findByTitleOrDescription(String title, String description);  
public long countByTitle(String title);  
public List<Todo> findDistinctByTitle(String title);  
public List<Todo> findFirstByTitleOrderByTitleAsc(String title);  
public List<Todo> findTop3ByTitleOrderByTitleAsc(String title);
```

### Ejercicio: Reporting

Vamos a ampliar la clase Cliente para tener ahora un campo fecha de **alta**, fecha de **nacimiento**, **apellidos** y **ciudad**. Crearemos ahora un servicio que permita obtener esta información utilizando las method queries en el repository:

1. Devolver todos los usuarios mayores de una edad dada por parámetro
2. Devolver los usuarios que se han dado de alta después de una fecha por parámetro
3. Devolver los usuarios que se han dado de alta entre 2 fechas (DIFICIL)
4. Devolver los usuarios ordenados alfabéticamente por los apellidos
5. Devolver todos los usuarios de una ciudad dada por parámetro
6. Devolver las ciudades ordenadas por el numero de usuarios creciente. (DIFICIL)
7. Devolver todos los usuarios cuyo apellido contenga parcialmente una cadena.



## Querys JPQL

Spring data nos permite escribir consultas personalizadas utilizando anotaciones en el método del repositorio.

Por ejemplo, si deseas buscar usuarios por su apellido y limitar el resultado a los primeros 10 usuarios, puedes utilizar la anotación `@Query` para definir una consulta personalizada de la siguiente manera:

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    @Query("SELECT u FROM User u WHERE u.lastName = :lastName")  
    List<User> findByLastName(String lastName);  
  
}
```

## Querys Nativas

Las consultas nativas permiten escribir consultas SQL nativas utilizando la anotación `@Query` y el parámetro `nativeQuery = true`. Por ejemplo, si deseas buscar usuarios por su nombre de usuario utilizando una consulta SQL nativa, puedes hacerlo de la siguiente manera:

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    @Query(value = "SELECT * FROM users WHERE username = :username", nativeQuery = true)  
    User findByUsername(@Param("username") String username);  
  
}
```

@Query

```
public interface UserRepository extends JpaRepository<User, Long> {  
    @Query("select u from User u where u.emailAddress = ?")  
    User findByEmailAddress(String emailAddress);  
}
```

@Query y @Param

```
public interface UserRepository extends JpaRepository<User, Long> {  
    @Query("select u from User u  
        where u.firstname = :firstname or u.lastname = :lastname")  
    User findByLastnameOrFirstname(@Param("lastname") String lastname,  
        @Param("firstname") String firstname);  
}
```

04

H2



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



### H2:

H2 es un sistema de gestión de bases de datos relacionales (RDBMS) de código abierto escrito en Java. Fue diseñado para ser muy rápido y eficiente en el uso de recursos, y es compatible con una amplia variedad de plataformas, incluyendo Windows, Linux, Mac y Android.

Algunas de las características de H2 incluyen:

- Soporte para SQL y JDBC
- Motor de base de datos en memoria y en disco
- Compatibilidad con múltiples usuarios y transacciones ACID
- Almacenamiento de datos cifrados
- Funciones de bases de datos avanzadas, como el soporte de tipos de datos espaciales y la replicación de bases de datos.

## H2: Configuración: Archivo properties

### # URL de la base de datos H2

**spring.datasource.url=jdbc:h2:mem:testdb**

### # Clase del controlador JDBC para H2

**spring.datasource.driverClassName=org.h2.Driver**

### # Credenciales de inicio de sesión para H2

**spring.datasource.username=your\_username**

**spring.datasource.password=your\_password**

### # Plataforma de base de datos para generar consultas JPA

**spring.jpa.database-platform=org.hibernate.dialect.H2Dialect**

### # Mostrar consultas SQL generadas por Hibernate en la consola de registro

**spring.jpa.show-sql=true**

### # Crear automáticamente tablas de la base de datos al iniciar la aplicación

**spring.jpa.hibernate.ddl-auto=create**

## H2: Configuración: Dependencias y Consola

```
<dependency>  
<groupId>com.h2database</groupId>  
<artifactId>h2</artifactId>  
</dependency>
```

La propiedad console se utiliza para habilitar la consola web de H2, que proporciona una interfaz web para administrar la base de datos

# Habilitar la consola web de H2

```
spring.h2.console.enabled=true
```

# URL de la consola web de H2

```
spring.h2.console.path=/h2-console
```



## H2: Configuración: Cargar Datos

Utilizando un archivo SQL: Puedes crear un archivo SQL que contenga las sentencias de inserción de datos y luego ejecutar ese archivo al iniciar la aplicación.

Para hacerlo en Spring Boot, puedes utilizar la propiedad `spring.datasource.initialization-mode` y especificar la ubicación del archivo SQL con la propiedad `spring.datasource.schema`.

Por ejemplo, si tienes un archivo `data.sql` en la ruta `src/main/resources` con sentencias SQL para insertar datos, puedes configurar las propiedades en tu archivo `application.properties` de la siguiente manera:

```
# Ejecutar archivo SQL al iniciar la aplicación
spring.datasource.initialization-mode=always
spring.datasource.schema=classpath:data.sql
```

## H2: Ejercicio

Cambiar ahora las propiedades de mi ejercicio de clientes y comprobar como se me generarn las tablas y puedo trabajar con esta h2.

El acceso a la bbdd se hará a través de localhost:8080/h2/console

05

## Ejercicios



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial










**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



## Ejercicios JRepository

Vamos a modificar el ejercicio anterior de **CLIENTES – CUENTAS – BANCOS**, suprimiremos el repositorio actual y utilizaremos un repositorio JPA Repository

Deberemos crear la siguiente estructura de paquetes:

- ▼  src/main/java
  - >  es.eoi
  - >  es.eoi.controller
  - >  es.eoi.dto
  - >  es.eoi.entity
  - >  es.eoi.repository
  - >  es.eoi.service