# SPRING DATA JPA:
# CONSULTAS BÁSICAS

# CONSULTAS

Spring Data JPA ofrece varios mecanismos

- **Derivadas del nombre del método**
- **@Query**
  - **Consultas JPQL**
  - **Consultas nativas SQL**
- Consultas con QueryDSL
- Consultas con Query By Example
- Consultas con Criteria API
- …

# 1.

# DERIVADAS DEL NOMBRE DEL MÉTODO

# CONSULTAS DERIVADAS DEL NOMBRE DEL MÉTODO

- Definimos métodos en nuestro repositorio.
- Dichos métodos no necesitan implementación (se la da Spring Data).
- El nombre del método debe seguir una serie de reglas para que de él se pueda derivar la consulta.

# ESTRUCTURA DE LA CONSULTA

La consulta debe comenzar por

- *find...By*
- *read...By*
- *query...By*
- *count...By*
- *get...By*

- *By* es el delimitador para definir los criterios.

# TRADUCCIÓN DE LA CONSULTA

***List<Person> findByLastname(String Lastname)***

- Repositorio sobre *Person*.
- Incluye una propiedad llamada *Lastname*.
- Queremos aquellas instancias de *Person* cuyo *Lastname* coincida con el que pasamos como argumento.

SELECT * FROM PERSON WHERE LASTNAME = '...';

# ELEMENTOS TÍPICOS DE CONSULTAS

- *Distinct*
- *And, or*
- *Between*
- *IsNull, IsNotNull, NotNull*
- *Like, NotLike*
- *In, Not in*
- *Containing*
- *OrderBy*
- ...

# EJEMPLOS DE CONSULTAS

- List<...> findByEmailAddressAndLastname(
  EmailAddress emailAddress, String Lastname)

- List<...> findByStartDateBetween(
  Date
  start, Date end)

- List<...> findByLastnameIn(
  Collection<String>
  Lastnames)

# LIMITACIÓN DE LOS RESULTADOS DE UNA CONSULTA

- *First, Top*

  ***Person findTopByOrderByAgeDesc()***
- Podemos usar Optional (Java 8) para aquellas que devuelven un solo resultado

  ***Optional<Person>***

  ***findTopByOrderByAgeDesc()***
- Se pueden acompañar de un número

  ***List<Person>***

***findTop10ByOrderByAgeDesc()***

# CONVIRTIENDO RESULTADOS DE CONSULTAS EN UN STREAM

- Spring Data permite devolver los resultados como un Stream de Java 8, para poder procesarlos después.

  ***Stream<User> readAllByFirstnameNotNull();***

# EN NUESTRO PROYECTO

Algunos cambios para trabajar

- Comentamos la validación del campo ID.
- Modificamos el formulario (¡ojo! Es una propuesta de ejercicio de la lección anterior. Si aún no lo has intentado, trata de conseguir modificar el código por tu cuenta)

# EN NUESTRO PROYECTO

## Implementemos un buscador

- Añadir datos de ejemplo aleatorios (al menos, 40 empleados).
- Añadir un formulario de búsqueda en el listado.
- Implementar una consulta.
- Modificar el controlador para mostrar los resultados.

# 2.

# CONSULTAS CON @QUERY

# JPQL

- *Java Persistence Query Language*
- Lenguaje de consulta de JPA
- Inspirado en SQL y HQL (*Hibernate Query Language*)
- Orientado a objetos
- Consultas SELECT, UPDATE, WHERE

# @QUERY

- Nos permite lanzar una consulta JPQL a través de un método.

```
@Query("select p from Persona p where u.nombre = ?1")
List<Persona> findByNombre(String nombre);
```

- Pasamos argumentos con *?1*, *?2*, …, *?n* o parámetros con nombre *:nombre*, *:apellidos*.
- Más flexibilidad que el método anterior.
- Recomendado si venimos del mundo SQL.

# @QUERY

- También podemos usar SQL mediante consultas nativas.

```
@Query(value="select * from Persona where nombre = ?1",
      nativeQuery=true)
List<Persona> findByNombre(String nombre);
```

# 3.

# OTROS MÉTODOS DE CONSULTA

# QUERYDSL

- Framework que permite la construcción de consultas.
- Estáticas, tipadas y a través de un API.
- Integrable no solo con Spring Data JPA, sino con otros módulos (por ejemplo, Spring Data MongoDB
- Cambios en el pom.xml

# QUERYDSL

- Extendemos *QuerydslPredicateExecutor*

```
public interface QuerydslPredicateExecutor<T>
{

  Optional<T> findById(Predicate predicate);
  Iterable<T> findAll(Predicate predicate);
  long count(Predicate predicate);
  boolean exists(Predicate predicate);
  // … más metodos
}
```

# QUERYDSL

- Extendemos *QuerydslPredicateExecutor*

```
public interface UserInterface extends
                CrudRepositorio<User, Long>,
             QuerydslPredicateExecutor<User> {
}
```

# QUERYDSL

- Podemos usar predicados para consultar

```
Predicate predicate =
user.firstname.equalsIgnoreCase("pepe")
        .and(user.lastname.startsWithIgnoreCase("s"));

userRepository.findAll(predicate);
```

# QUERYBYEXAMPLE

- Incluido dentro de Spring Data JPA
- *CrudRepository* + *QueryByExcampleExecutor* o *JpaRepository*
- La idea subyacente es buscar dando un ejemplo (*Example<S>*) de lo que queremos encontrar.

# QUERYBYEXAMPLE

- Búsqueda sencilla

```
Empleado filter = new Empleado();
empleado.setTelefono("954000000");

Example<Empleado> example = Example.of(filter);
List<Empleado> result =

        repositorio.findAll(example);
```

# QUERYBYEXAMPLE

- Búsqueda algo más compleja

```
Empleado filter = new Empleado();
empleado.setNombre("mar");
ExampleMatcher matcher = ExampleMatcher.matching()
        .withStringMatcher(StringMatcher.CONTAINING
)
        .withIgnoreCase();
Example<Empleado> example =
                Example.of(filter, matcher);
List<Empleado> result =

        repositorio.findAll(example);
```

# ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

# MÁS CONSULTAS

- Trata de incluir más consultas en el repositorio, y añade la lógica necesaria en el servicio y el controlador para poder visualizar los resultados.
- Por ejemplo:
  - modifica los teléfono para que sean aleatorios, y haz una búsqueda específica por teléfono.
  - Busca a los empleados que tengan nulos en algún campo.

# CONSULTAS
# JPA

Accede a nuestro Curso de Hibernate y JPA para saber más sobre consultas, sobre todo si utilizas más de una entidad.