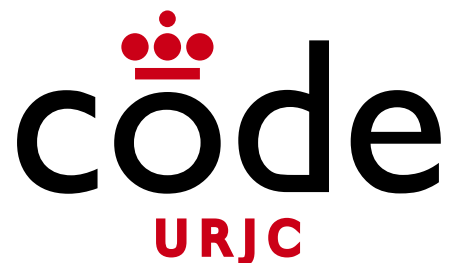


## Desarrollo Web

# Tecnologías de servidor web

## Tema 2.4: Consultas



©2025

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto, Iván Chicano

Algunos derechos reservados


Este documento se distribuye bajo la licencia  
“Atribución-CompartirIgual 4.0 Internacional”  
de Creative Commons Disponible en  
<https://creativecommons.org/licenses/by-sa/4.0/deed.es>

# Consultas

- Estrategias para especificación de consultas con SpringData:
  - **Métodos en los repositorios**
  - Java Persistence Query Language (JPQL)
  - QueryByExample
- Paginación

- Métodos en los repositorios
  - En base al nombre del método y el tipo de retorno se construye la consulta a la BBDD

```
public interface PostRepository extends JpaRepository<Post, Long> {  
    List<Post> findByUsername(String username);  
    List<Post> findByTitle(String title);  
}
```



Consultas con un  
único campo como  
criterio

- Métodos en los repositorios
  - Utilizamos esta consulta en el controlador

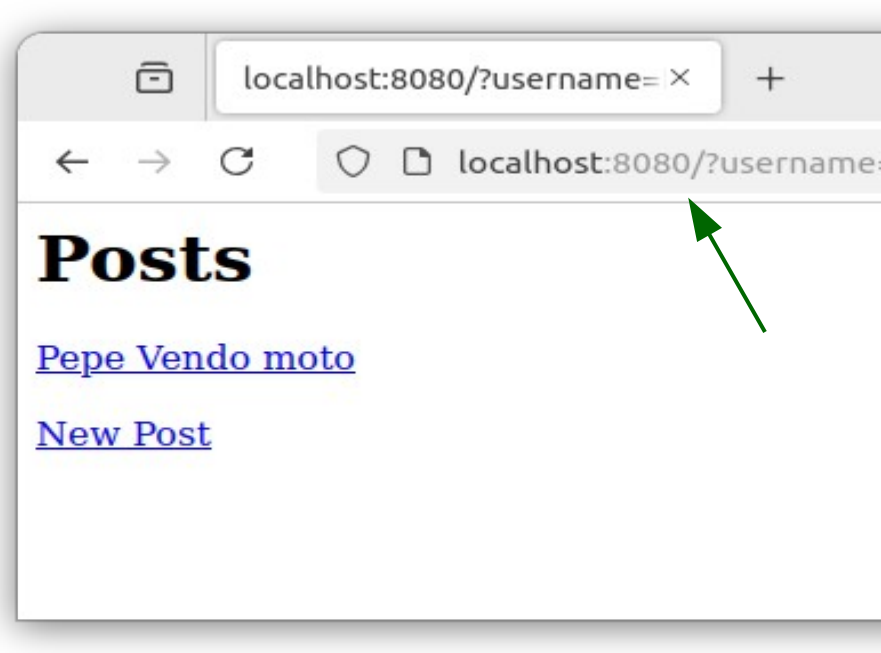
```
@GetMapping("/")
public String showPosts(Model model, @RequestParam(required = false) String username)
{
    if (username != null) {
        model.addAttribute("posts", posts.findByUsername(username));
    } else {
        model.addAttribute("posts", posts.findAll());
    }
    return "index";
}
```



Parametro opcional para buscar por *username*

- Métodos en los repositorios

ejem11



## • Métodos en los repositorios

Consultas  
combinando varios  
campos

```
public interface PersonRepository extends Repository<Person, Long> {  
  
    List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);  
  
    // Enables the distinct flag for the query  
    List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);  
    List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);  
  
    // Enabling ignoring case for an individual property  
    List<Person> findByLastnameIgnoreCase(String lastname);  
  
    // Enabling ignoring case for all suitable properties  
    List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);  
  
    // Enabling static ORDER BY for a query  
    List<Person> findByLastnameOrderByFirstnameAsc(String lastname);  
    List<Person> findByLastnameOrderByFirstnameDesc(String lastname);  
  
}
```

Ordenación,  
IgnoreCase, Order...

- **Métodos en los repositorios**
  - **Combinación lógica:** And, Or
  - **Comparadores:** Between, LessThan, GreatherThan
  - **Modificadores:** IgnoreCase
  - **Ordenación:** OrderBy...Asc / OrderBy...Desc



- Métodos en los repositorios

- Consulta

- List<Elem> find...By...(...)
    - List<Elem> read...By...(...)
    - List<Elem> query...By...(...)
    - List<Elem> get...By...(...)

- Número de elementos

- int count...By...(...)

- **Métodos en los repositorios**
  - Propiedades de los objetos relacionados
    - No sólo podemos filtrar por una propiedad de la entidad, también podemos filtrar por un **atributo de otra entidad** con la que esté relacionada
      - **Person** con un atributo **address**
      - **Address** tiene un atributo **zipCode**

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

- **Métodos en los repositorios**

- Ordenación

- Podemos pasar un parámetro Sort que controla la ordenación
    - Existe el método `findAll(Sort sort)`

```
repository.findAll(Sort.by("nombre"));  
repository.findAll(Sort.by(Order.asc("nombre"))));
```

- Podemos añadir métodos personalizados

```
List<User> findByLastname(String lastname, Sort sort);
```

- Métodos en los repositorios
  - Ordenación
    - El objeto Sort se puede crear desde la URL

```
http://localhost:8080/posts/?sort=nombre,desc
```

```
@GetMapping("/posts/")  
public List<Post> getPosts(Sort sort) {  
    return posts.findAll(sort);  
}
```

# Consultas

- Métodos en los repositorios
  - Limitar los resultados

```
User findFirstBy...();
User findTopBy...();
User findTopDistinctBy...();
List<User> queryFirst10By...();
List<User> findTop3By...();
List<User> findFirst10By...();
```

# Consultas

- Estrategias para especificación de consultas con SpringData:
  - Métodos en los repositorios
  - **Java Persistence Query Language (JPQL)**
  - QueryByExample
- Paginación

- Java Persistence Query Language (JPQL)
  - JPQL es un lenguaje similar a SQL pero referencia conceptos de las entidades JPA

ejem12

```
public interface TeamRepository extends JpaRepository<Team, Long> {  
  
    @Query("select t from Team t where t.name = ?1")  
    List<Team> findByName(String name);  
}
```

**NOTA:** En este ejemplo la query JPQL no sería necesaria porque el nombre del método define justo esa consulta

- Java Persistence Query Language (JPQL)
  - Estructura de sentencia SELECT

```
SELECT ... FROM ...  
[WHERE ...]  
[GROUP BY ... [HAVING ...]]  
[ORDER BY ...]
```

- Actualización y borrado

```
DELETE FROM ... [WHERE ...]  
  
UPDATE ... SET ... [WHERE ...]
```



- Java Persistence Query Language (JPQL)
  - Ejemplos

```
SELECT e from Employee e  
WHERE e.salary BETWEEN 30000 AND 40000
```

```
SELECT e from Employee e WHERE e.name LIKE 'M%'
```

```
SELECT DISTINCT b FROM Blog b JOIN b.comments c  
WHERE c.author=?1
```

- Java Persistence Query Language (JPQL)

ejem13

```
public interface PostRepository extends JpaRepository<Post, Long> {  
  
    @Query("SELECT DISTINCT p FROM Post p JOIN p.comments c WHERE c.author=?1")  
    List<Post> findByCommentsAuthor(String author);  
  
}
```

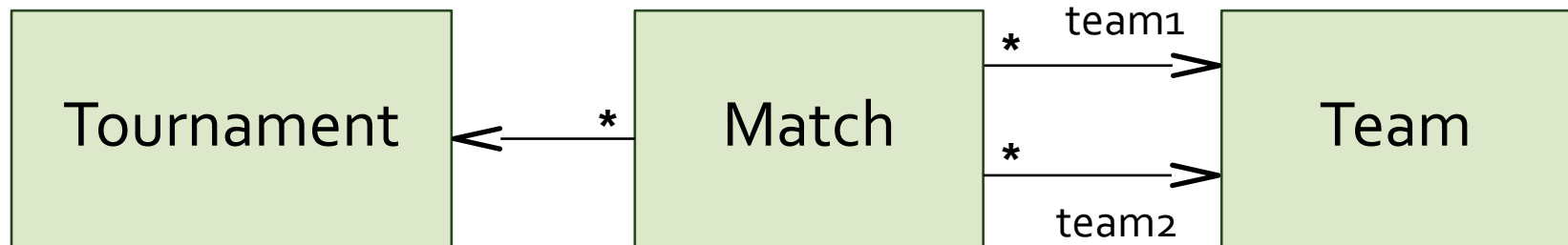
NOTA: La query JPQL no sería necesaria porque el método del repositorio ejecuta justo esa consulta

- Structured Query Language (SQL)
  - Incluso podemos usar SQL nativo directamente

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    @Query(  
        value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1",  
        nativeQuery = true)  
    User findByEmailAddress(String emailAddress);  
  
}
```

- Las consultas permiten relacionar varias entidades sin necesidad de tener atributos directos

- **Modelo**



- ¿Qué equipos juegan en un torneo?
- ¿En qué torneos juega un equipo?

```
@Entity
public class Tournament {

    @Id
    @GeneratedValue(...)
    private long id;

    private String name;
}
```

```
@Entity
public class Team {

    @Id
    @GeneratedValue(...)
    private long id;

    private String name;
}
```

```
@Entity
public class Match {

    @Id
    @GeneratedValue(...)
    private long id;

    private String name;

    @ManyToOne
    private Team team1;

    @ManyToOne
    private Team team2;

    @ManyToOne
    Tournament tournament;
}
```

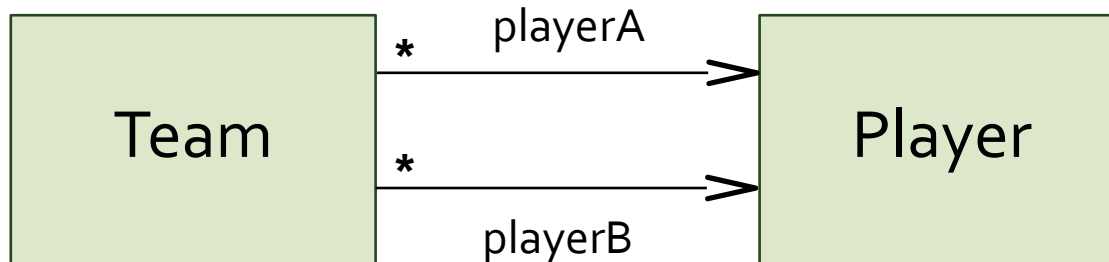
# Consultas

```
public interface MatchRepository extends JpaRepository<Match, Long> {  
    @Query("SELECT m FROM Match m WHERE m.tournament = :t")  
    public List<Match> getMatchesByTournament(Tournament t);  
}
```

```
public interface TeamRepository extends JpaRepository<Team, Long> {  
    @Query("SELECT distinct team FROM Match m, Team team "  
        + "WHERE (m.team1 = team OR m.team2 = team) AND m.tournament = :t")  
    public List<Team> getTeamsByTournament(Tournament t);  
}
```

```
public interface TournamentRepository extends JpaRepository<Tournament, Long> {  
    @Query("SELECT distinct t FROM Match m JOIN m.tournament t "  
        + "WHERE m.team1 = :team OR m.team2 = :team")  
    public List<Tournament> getTournamentsByTeam(Team team);  
}
```

- Ejemplo de consultas avanzadas
  - ¿Qué jugadores juegan conmigo?



```
public interface PlayerRepository extends JpaRepository<Player, Long> {

    @Query("""
        select distinct u from Player u, Team t
        where (t.playerA = u and t.playerB = :player) or (t.playerB = u and t.playerA = :player)
        """)
    List<Player> findPairsOf(Player player);
}
```

# Consultas

- Estrategias para especificación de consultas con SpringData:
  - Métodos en los repositorios
  - Java Persistence Query Language (JPQL)
  - **QueryByExample**
- Paginación

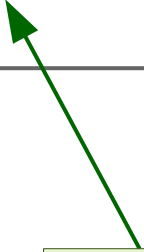


- **QueryByExample**

- Permite definir la consulta basándose en un ejemplo
- Un ejemplo es una instancia de una clase a la que podemos asignar valor a uno o varios de sus atributos
- Se buscará uno o todos registros de la base de datos que coincidan con el ejemplo
- Proporciona una interfaz sencilla de búsqueda

- QueryByExample

```
Person person = new Person();  
person.setFirstname("Dave");  
  
Example<Person> example = Example.of(person);  
repository.findAll(example);
```



Los atributos sin valor (null) no  
son considerados para la  
búsqueda

## • QueryByExample

ejem15

```
@GetMapping("/teams")
public String getTeams(Model model, @RequestParam(required = false) String name) {
    if(name != null) {
        Team team = new Team();
        team.setName(name);
        ExampleMatcher matcher = ExampleMatcher.matching()
                                                    .withIgnorePaths("id", "ranking");
        Example<Team> example = Example.of(team, matcher);
        model.addAttribute("teams", teamRepository.findAll(example));
    }else{
        model.addAttribute("teams", teamRepository.findAll());
    }
    return "teams";
}
```

Los tipos primitivos es necesario ignorarlos si no los vamos a utilizar

## • QueryByExample

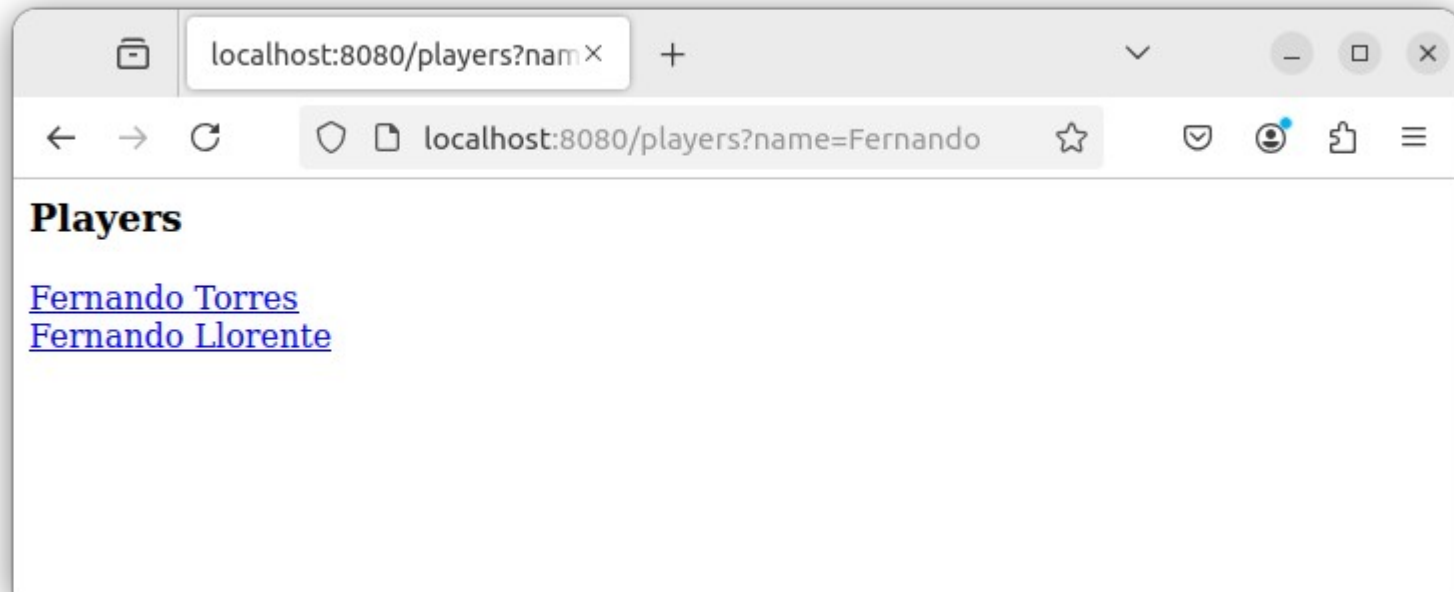
ejem15

```
@GetMapping("/players")
public String getPlayers(Model model, @RequestParam(required = false) String name) {
    if(name != null) {
        Player player = new Player();
        player.setName(name);
        ExampleMatcher matcher = ExampleMatcher.matching()
            .withStringMatcher(ExampleMatcher.StringMatcher.CONTAINING)
            .withIgnorePaths("id", "goals");
        Example<Player> example = Example.of(player, matcher);
        model.addAttribute("players", playerRepository.findAll(example));
    } else {
        model.addAttribute("players", playerRepository.findAll());
    }
    return "players";
}
```

Podemos hacer búsquedas por partes del texto (en este caso, por una parte del nombre del jugador)

- QueryByExample

ejem15



# Consultas

- Estrategias para especificación de consultas con SpringData:
  - Métodos en los repositorios
  - Java Persistence Query Language (JPQL)
  - QueryByExample
- **Paginación**

# Paginación

- Con SpringData es muy sencillo que los resultados de las consultas se devuelvan **paginados**
- Basta poner el parámetro **Pageable** en el método del controlador y pasar ese parámetro al método del repositorio
- Una Page contiene la información de una **página** (info, nº pág, nº total elementos)
- Se integra con Spring MVC para las **webs** y **APIs REST**

# Paginación

ejem16

Devolvemos un objeto  
**Page<Post>**

En nuestros métodos  
podemos añadir un  
parámetro **Pageable**

```
public interface PostRepository extends JpaRepository<Post, Long> {  
    Page<Post> findByTitle(String title, Pageable page);  
}
```

```
Page<Post> a = postRepository.findByTitle("Spring", PageRequest.of(0, 50));  
Page<Post> a = postRepository.findAll(PageRequest.of(3, 20));
```

El método findAll también tiene  
versión **Paginable**

Creamos un objeto **PageRequest**  
para indicar en num página y tamaño



# Paginación

ejem16

- En el controlador, podemos crear directamente el objeto Pageable como parámetro

```
@GetMapping("/")
public String getPosts(Model model, @RequestParam(required = false) String title,
Pageable page){
    if (title != null) {
        model.addAttribute("posts", postRepository.findByTitle(title, page));
    } else {
        model.addAttribute("posts", postRepository.findAll(page));
    }
    return "index";
}
```

# Paginación

ejem16

- Con la URL habitual se devuelve la **primera página** con **20 elementos**

```
http://localhost:8080/posts/
```

- En las peticiones se pueden incluir **parámetros** para solicitar cualquier página

```
http://localhost:8080/posts/?page=1&size=3
```

- Válido para **webs** y para **APIs REST**

# Paginación

ejem17

- Podemos utilizar los métodos del objeto Pageable para crear una navegación desde HTML

```
@GetMapping("/")
public String getPosts(Model model, Pageable page){

    model.addAttribute("posts", postRepository.findAll(page));

    boolean hasPrev = page.getPageNumber() >= 1;
    boolean hasNext = (page.getPageNumber() * page.getPageSize()) < postRepository.count();

    model.addAttribute("hasPrev", hasPrev);
    model.addAttribute("prev", page.getPageNumber() - 1);
    model.addAttribute("hasNext", hasNext);
    model.addAttribute("next", page.getPageNumber() + 1);

    return "index";
}
```

# Paginación

- Podemos utilizar los métodos del objeto Pageabe para crear una navegación desde HTML

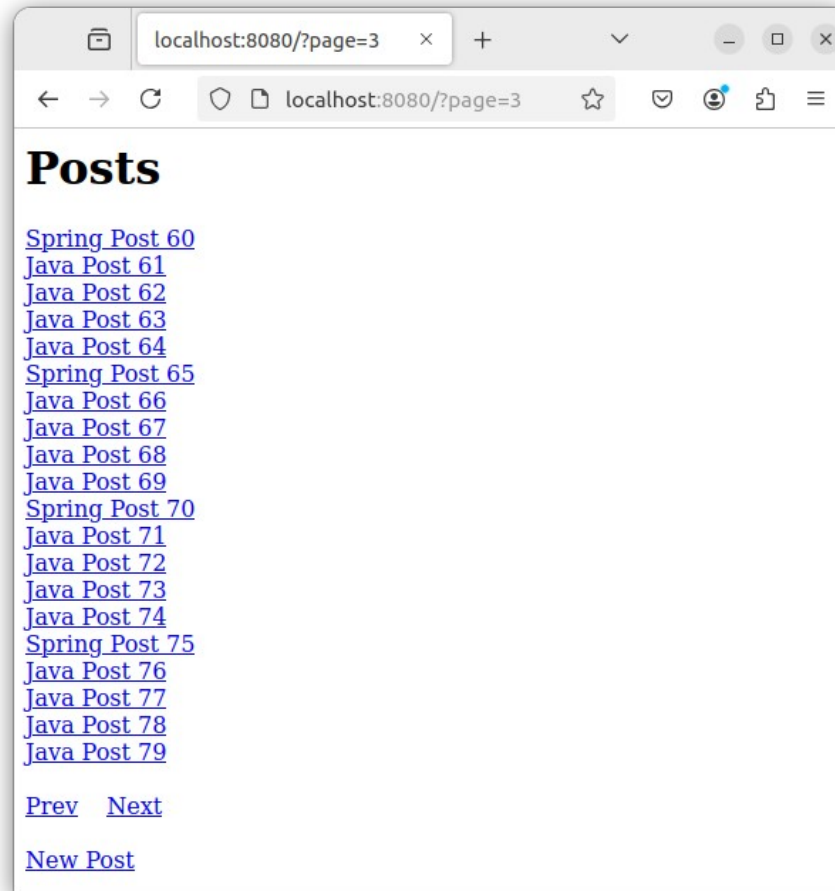
```
<html>
<body>
  <h1>Posts</h1>

  {{#posts}}
  <a href="/posts/{{id}}">{{title}}</a>
  <br>
  {{/posts}}

  <div>
    <noBr>
      {{#hasPrev}}<a href="/?page={{prev}}">Prev</a>&nbsp;&nbsp;&nbsp;{{/hasPrev}}
      {{#hasNext}}<a href="/?page={{next}}">Next</a>{{/hasNext}}
    </noBr>
  </div>

  <br>
  <a href="new_post.html">New Post</a>
</body>
</html>
```

# Paginación



# Paginación

- Ordenación

- El objeto Pageable incluye la información de ordenación de la URL

```
http://localhost:8080/?page=1&size=3&sort=title
```