UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Objetivos de la sesión:

- ***Practicar*** *como realizar un nuevo* ***proyecto REST desde cero*** *con acceso a una base de datos.*
- *Aprender a realizar* ***paginaciones con filtrados y ordenaciones en Spring*** *para devolver solo un subconjunto del total de los datos.*

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Hasta ahora solo hemos utilizado tablas con pocos datos, pero si tenemos una tabla que tiene cientos de registros no es lógico mostrarlos todos cuando realizamos un listado.

¿ Como podemos mostrar solo un subconjunto de estos datos?

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Para esta práctica vamos a crear otro proyecto REST CRUD sobre una BD. Sobre este nuevo proyecto practicaremos los nuevos conceptos:

Recordemos los pasos a seguir para crear un API RESTful:

1º Crear el proyecto Spring Boot
2º Configurar la Base de Datos
3º Crear las entidades JPA
4º Inicializar con datos la BD
5º Crear el repositorio para cada entidad JPA
6º Crear la capa de servicios
7º Crear la capa de control (controller)
8º Pruebas

1º Crear el proyecto Spring Boot:

Para empezar vamos a crear nuestro proyecto REST llamado **"dwes_futbol_rest"** . Le damos este nombre porque en una futura práctica ampliaremos la funcionalidad de este proyecto.

Lo hacemos igual como en "dwes_primer_rest":

Java Projects\Añadir (+) \Spring Boot \Proyecto Maven \ versión 2.6.3 \Java

e introducimos los siguientes datos:

Group Id: edu.alumno.NombreAlumno
Artifact Id: dwes_futbol_rest
Package: war
Versión de Java: 11

Dependencias:

Deberemos seleccionar las mismas dependencias del otro proyecto: "Spring Web" ,"H2 Database" ,"Spring Data JPA", "Lombok" y añadir "DevTools" y "Validation" y le damos al intro.

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

… continuación 1º Crear el proyecto Spring Boot:

Para continuar deberemos de copiar la dependencia de MapStruct y plugins como el compilador de maven. Lo más cómodo es que vayamos al proyecto "dwes_primer_rest" y copiemos de su fichero pom.xml de properties hacia abajo:

pom.xml 1

dwes_futbol_rest > pom.xml > ...

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>edu.profesor.joseramon</groupId>
    <artifactId>dwes_futbol_rest</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>dwes_futbol_rest</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
        <org.projectlombok.version>1.18.16</org.projectlombok.version>
        <org.mapstruct.version>1.4.1.Final</org.mapstruct.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

2º Configurar la Base de Datos:

Podemos copiar y pegar de la práctica “dwes_primer_rest” la configuración de la base de datos en “application.properties”:

application.properties

src > main > resources > application.properties

```
#Activar consola para acceder a la BD H2 via navegador
# localhost:puertoConfigurado/h2-console
spring.h2.console.enabled=true
#Configuración de la BD H2
spring.datasource.url=jdbc:h2:mem:testDwesDb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=dwes
spring.datasource.password=Simarro@1
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

#En la versión 2.4.2 no hace falta, pero en la
# 2.6.2 hace falta para poder hacer inserts en data.sql
spring.jpa.hibernate.ddl-auto=none

#CONFIGURACIÓN SOLO durante las pruebas:
# Habilitar estadísticas hibernate
spring.jpa.properties.hibernate.generate_statistics=true
# Habilitar LOGGER de las estadísticas de hibernate
logging.level.org.hibernate.stat=
# Mostrar que consultas esta realizando Hibernate
spring.jpa.show-sql=true
# Formatear las consultas
spring.jpa.properties.hibernate.format_sql=true
# Mostrar los parametros que estan enviandose a las consultas
logging.level.org.hibernate.type=debug
#FIN CONFIGURACIÓN SOLO durante las pruebas
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

3º Crear las entidades JPA:

Empezaremos de momento por crear la entidad “CiudadDb.java” :

CiudadDb.java

dwes_futbol_rest > src > main > java > edu > profesor > joseramon > dwes_futbol_rest > model > db > CiudadDb.java > ...

```
1   package edu.profesor.joseramon.dwes_futbol_rest.model.db;
2
3   import java.io.Serializable;
14
15  @NoArgsConstructor
16  @AllArgsConstructor
17  @Data
18  @Entity
19  @Table(name = "ciudades")
20  public class CiudadDb implements Serializable{
21      @Id
22      @GeneratedValue(strategy = GenerationType.IDENTITY)
23      private Long id;
24      @Size(min=4,message="El nombre debe de tener un tamaño mínimo de 4 carácteres")
25      private String nombre;
26      @Column(nullable = true)
27      private Long habitantes;
28  }
```

4º Inicializar con datos la BD:

Al igual que el proyecto “dwes_primer_rest”, para inicializar la Base de Datos H2 hay que crear el fichero “data.sql” en la carpeta “src\main\resources” y copiar las sentencias siguientes que también están en el DRIVE (data.sql):

data.sql

src > main > resources > data.sql

```sql
-- Copyright Jose Ramón Cebolla - 2022
-- ----------------------------------------------------
-- Table `ciudades`
-- ----------------------------------------------------
DROP TABLE IF EXISTS ciudades;
CREATE  TABLE IF NOT EXISTS ciudades(
id IDENTITY,
nombre VARCHAR(50),
habitantes BIGINT,
 CONSTRAINT pk_ciudades PRIMARY KEY(id));

-- ----------------------------------------------------
-- INSERCIÓN DE DATOS
-- ----------------------------------------------------

INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (2,'Alacant',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (3,'Albacete',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (4,'Almeria',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (5,'Astúries',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (6,'Àvila',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (7,'Badajoz',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (8,'Barcelona',1615000);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (9,'Bilbao',353000);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (10,'Biscaia',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (11,'Burgos',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (12,'Càceres',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (13,'Cadis',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (14,'Cantàbria',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (15,'Castelló',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (16,'Ceuta',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (17,'ciudad Real',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (18,'Còrdova',NULL);
INSERT INTO `ciudades` (`id`,`nombre`,`habitantes`) VALUES (19,'Cornellà de Llobregat',85000);
```

5º Crear el repositorio para cada entidad JPA:

Para hacer **CRUD (Create Read Update Delete)** sobre la **tabla "Ciudades"** volvemos a utilizar la interface JpaRepository de Spring y si queremos poder ordenar los resultados de un listado añadiremos el método con la clase Sort:

CiudadRepository.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > repository > CiudadRepository.java

```java
package edu.profesor.joseramon.dwes_futbol_rest.repository;

import java.util.List;

import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import edu.profesor.joseramon.dwes_futbol_rest.model.db.CiudadDb;

@Repository
public interface CiudadRepository extends JpaRepository<CiudadDb,Long>{
    //Métodos automáticos en Spring Data JPA
    List<CiudadDb> findAll(Sort sort);
}
```

6º Crear la capa de servicios:

Para crear la capa de servicios vamos a crear primero una interface con los servicios que se ofrecerán y luego implementaremos una clase que rellene dichos métodos de la interface. De momento solo queremos poder consultar un dato o el listado ordenado o sin ordenar:

CiudadService.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > srv > CiudadService.java > ...

```java
package edu.profesor.joseramon.dwes_futbol_rest.srv;

import java.util.List;
import java.util.Optional;

import org.springframework.data.domain.Sort;

import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadInfo;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadList;

public interface CiudadService {
    public Optional<CiudadInfo> getCiudadInfoById(Long id);
    public List<CiudadList> findAllCiudadList();
    public List<CiudadList> findAllCiudadList(Sort sort);
}
```

... continuación 6º Crear la capa de servicios:

Mappers:

Para poder realizar la implementación del servicio necesitaremos configurar CiudadMapper.java:

CiudadMapper.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > srv > mapper > CiudadMapper.java > ...

```java
package edu.profesor.joseramon.dwes_futbol_rest.srv.mapper;

import java.util.List;

import org.mapstruct.Mapper;
import org.mapstruct.factory.Mappers;

import edu.profesor.joseramon.dwes_futbol_rest.model.db.CiudadDb;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadInfo;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadList;

@Mapper
public interface CiudadMapper {
    CiudadMapper INSTANCE= Mappers.getMapper(CiudadMapper.class);

    CiudadInfo CiudadDbToCiudadInfo(CiudadDb ciudadDb);
    CiudadList CiudadDbToCiudadList(CiudadDb ciudadDb);
    List<CiudadList> ciudadesToCiudadList(List<CiudadDb> ciudadesDb);
}
```

NOTA: No se han implementado todos los métodos (guardar, modificar,…) porque en esta practica se pretende analizar como se puede realizar consultas devolviendo un subconjunto de los datos.

... continuación 6º Crear la capa de servicios:

Y a continuación implementaremos los métodos del servicio:

CiudadServiceImpl.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > srv > impl > CiudadServiceImpl.java > ...

```java
package edu.profesor.joseramon.dwes_futbol_rest.srv.impl;

import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import edu.profesor.joseramon.dwes_futbol_rest.model.db.CiudadDb;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadInfo;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadList;
import edu.profesor.joseramon.dwes_futbol_rest.repository.CiudadRepository;
import edu.profesor.joseramon.dwes_futbol_rest.srv.CiudadService;
import edu.profesor.joseramon.dwes_futbol_rest.srv.mapper.CiudadMapper;

@Service
public class CiudadServiceImpl implements CiudadService{
    private final CiudadRepository ciudadRepository;

    @Autowired
    public CiudadServiceImpl(CiudadRepository ciudadRepository){
        this.ciudadRepository=ciudadRepository;
    }

    public List<CiudadList> findAllCiudadList(){
        return CiudadMapper.INSTANCE.ciudadesToCiudadList(ciudadRepository.findAll());
    }

    public List<CiudadList> findAllCiudadList(Sort sort){
        return CiudadMapper.INSTANCE.ciudadesToCiudadList(ciudadRepository.findAll(sort));
    }

    public Optional<CiudadInfo> getCiudadInfoById( Long id){
        Optional<CiudadDb> ciudadDb=ciudadRepository.findById(id);
        if (ciudadDb.isPresent()){
            return Optional.of(CiudadMapper.INSTANCE.CiudadDbToCiudadInfo(ciudadDb.get()));

        }else{
            return Optional.empty();
        }
    }
}
```

7º Crear la capa de control (controller):

Ahora implementamos los métodos de consulta del controlador:

CiudadRestController.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > controller > CiudadRestController.java > ...

```java
package edu.profesor.joseramon.dwes_futbol_rest.controller;

import java.util.Collection;

@RestController
@RequestMapping("/api/v1/")
public class CiudadRestController {

    private CiudadService ciudadService;

    @Autowired
    public CiudadRestController(CiudadService ciudadService) {
        this.ciudadService=ciudadService;
    }

    @GetMapping("/ciudades")
    public Collection<CiudadList> getCiudadesList() {
        return ciudadService.findAllCiudadList();
    }

    @GetMapping("/ciudades/orden/{direccionOrden}")
    public Collection<CiudadList> getCiudadesListOrderByName(
        @PathVariable("direccionOrden") String direccionOrden) {
        return ciudadService.findAllCiudadList(Sort.by(Direction.fromString(direccionOrden),"nombre"));
    }

    @GetMapping("/ciudades/{id}/info")
    public ResponseEntity<CiudadInfo> getCiudadInfoById(
    @PathVariable(value = "id") Long id) throws RuntimeException {
        Optional<CiudadInfo> alumnoInfo = ciudadService.getCiudadInfoById(id);
        if (alumnoInfo.isPresent())
            return ResponseEntity.ok().body(alumnoInfo.get());
        else throw new ResourceNotFoundException("Ciudad not found on :: "+ id);
    }
}
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

8º Pruebas:

Si realizamos pruebas vemos que funciona: (Fijate que 'Àvila' tiene acento)

test_Rest.http

```
## getCiudadesList
Send Request
GET http://localhost:8080/api/v1/ciudades HTTP/1.1
Content-Type: application/json
###
##  getCiudadesListOrderByName
Send Request
GET http://localhost:8080/api/v1/ciudades/orden/desc HTTP/1.1
Content-Type: application/json
###
##  getCiudadInfoById
Send Request
GET http://localhost:8080/api/v1/ciudades/56/info HTTP/1.1
Content-Type: application/json
###
```

Response(29ms)

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 03 Feb 2022 09:32:30 GMT
Connection: close

{
  "id": 56,
  "nombre": "València",
  "habitantes": 798000
}
```

Response(27ms)

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 03 Feb 2022 10:04:03 GMT
Connection: close

[
  {
    "id": 6,
    "nombre": "Àvila",
    "habitantes": null
  },
  {
    "id": 59,
    "nombre": "Zamora",
    "habitantes": null
  },
  {
    "id": 58,
    "nombre": "Vigo",
    "habitantes": 297000
  },
  {
    "id": 56,
    "nombre": "València",
    "habitantes": 798000
  },
  {
    "id": 57,
    "nombre": "Valladolid",
    "habitantes": 313000
  },
  {
    "id": 55,
    "nombre": "Toledo",
    "habitantes": null
```

Response(268ms)

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 03 Feb 2022 09:32:02 GMT
Connection: close

[
  {
    "id": 2,
    "nombre": "Alacant",
    "habitantes": null
  },
  {
    "id": 3,
    "nombre": "Albacete",
    "habitantes": null
  },
  {
    "id": 4,
    "nombre": "Almeria",
    "habitantes": null
  },
  {
    "id": 5,
    "nombre": "Astúries",
    "habitantes": null
  },
  {
    "id": 6,
    "nombre": "Àvila",
    "habitantes": null
  },
```

Paginación, filtrado y ordenación:

Pretendemos tener peticiones que nos permitan filtrar, ordenar y paginar de manera independiente con urls similares a:

api/v1/ciudades
api/v1/ciudades?size=5
api/v1/ciudades?sort=nombre,desc&page=1&size=5
api/v1/ciudades?nombre=Val&page=1&size=3
api/v1/ciudades?sort=habitantes,asc& sort=id,asc&page=4&size=10

¿ Como podemos hacerlo?

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

… continuación **Paginación, filtrado y ordenación:**

Empezamos preguntandonos si cuando filtramos por un String existe alguna funcionalidad que nos permite hacer un 'LIKE' para comprobar si el String contiene una palabra sin tener que hacer una Query. La respuesta es si gracias a "Containing":

CiudadRepository.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > repository > CiudadRepository.java > ...

```java
package edu.profesor.joseramon.dwes_futbol_rest.repository;

import java.util.List;

@Repository
public interface CiudadRepository extends JpaRepository<CiudadDb,Long>{
    //Métodos automáticos en Spring Data JPA
    List<CiudadDb> findAll(Sort sort);
    //Filtrado por nombre usando equivalente a 'LIKE' usando "Containing"
    List<CiudadDb> findByNombreContaining(String nombre,Sort sort);
}
```

Implementa el service y el controller para que te devuelva el siguiente resultado:

test_Rest.http

```
Content-Type: application/json
###
##  getCiudadInfoById
Send Request
GET http://localhost:8080/api/v1/ciudades/56/info HTTP/1.1
Content-Type: application/json
###
##  getCiudadesByNombreContainingListOrderByName
Send Request
GET http://localhost:8080/api/v1/ciudades/nombre/Val/orden/desc HTTP/1.1
Content-Type: application/json
###
```

Response(43ms)

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 03 Feb 2022 10:28:02 GMT
Connection: close

[
  {
    "id": 56,
    "nombre": "València",
    "habitantes": 798000
  },
  {
    "id": 57,
    "nombre": "Valladolid",
    "habitantes": 313000
  }
]
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación Paginación, filtrado y ordenación:

Para realizar la paginación vamos a utilizar la interfaz de **Spring Data "Page"** que a su vez hereda de **"Slice"**:

```java
public interface Page<T> extends Slice<T> {
  static <T> Page<T> empty();
  static <T> Page<T> empty(Pageable pageable);
  long getTotalElements();
  int getTotalPages();
  <U> Page<U> map(Function<? super T,? extends U> converter);
}
```

```java
public interface Slice<T> extends Streamable<T> {
  int getNumber();
  int getSize();
  int getNumberOfElements();
  List<T> getContent();
  boolean hasContent();
  Sort getSort();
  boolean isFirst();
  boolean isLast();
  boolean hasNext();
  boolean hasPrevious();
  ...
}
```

No vamos a utilizar Slice, simplemente comentar que si alguna vez el rendimiento necesita mejorarse seria hora de utilizar Slice, porque tiene menos información que Page.

… continuación Paginación, filtrado y ordenación:

Spring Data Pageable:

Pageable es una interfaz que se utiliza como parámetro en el Repository y que permite aplicar paginación y ordenación a la base de datos.

Pageable contiene información sobre la página solicitada, como podría ser el número de página o el tamaño de la página.

```java
public interface Pageable {
  int getPageNumber();
  int getPageSize();
  long getOffset();
  Sort getSort();
  Pageable next();
  Pageable previousOrFirst();
  Pageable first();
  boolean hasPrevious();
  ...
}
```

Cuando queremos utilizar la paginación (con o sin filtro/ordenación) podemos añadir el método como parámetro:

CiudadRepository.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > repository > CiudadRepository.java > Langu

```java
package edu.profesor.joseramon.dwes_futbol_rest.repository;

import java.util.List;

@Repository
public interface CiudadRepository extends JpaRepository<CiudadDb,Long>{
    //Métodos automáticos en Spring Data JPA
    List<CiudadDb> findAll(Sort sort);
    //Filtrado por nombre usando equivalente a 'LIKE' usando "Containing"
    List<CiudadDb> findByNombreContaining(String nombre,Sort sort);
    //Paginación
    Page<CiudadDb> findAll(Pageable pageable);
    Page<CiudadDb> findByNombreContaining(String nombre,Pageable pageable);
}
```

... continuación Paginación, filtrado y ordenación:

Ahora la pregunta seria:

¿Que más hace falta tocar y como? ¿Hace falta tocar el mapper? ¿Y los servicios?

... continuación Paginación, filtrado y ordenación:

El mapper no hace falta tocarlo porque no es capaz de convertir un Page<CiudadDb> a un Page<CiudadList>.

Entonces ¿Como podemos devolver una Page<CiudadList>?

… continuación Paginación, filtrado y ordenación:

El sistema más cómodo es NO hacerlo. Realmente solo necesitamos una estructura de datos genérica (que permita pasar cualquier tipo de lista) donde se indique:

- El número de la página solicitada
- El tamaño de pagina que queremos
- El total de elementos que coinciden con la consulta
- El total de páginas si tenemos el tamaño especificado de página
- El contenido (lista de datos)
- El criterio de ordenación

En la siguiente página vamos a generar una nueva clase donde almacenar toda esta información de manera genérica. Esto significa que igual permite almacenar un List<CiudadList> que un List<CiudadInfo>.

¿Como?

... continuación Paginación, filtrado y ordenación:

Cuando especificamos <T> estamos permitiendo cualquier tipo de dato y conseguimos una estructura capaz de almacenar cualquier tipo de lista. Crea PaginaDto.java:

PaginaDto.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > model > dto > PaginaDto.java > ...

```java
package edu.profesor.joseramon.dwes_futbol_rest.model.dto;

import java.util.List;

import org.springframework.data.domain.Sort;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class PaginaDto<T> {
    int number; //número de página solicitada
    int size; //tamaño de la página
    long totalElements; //total de elementos devueltos por la consulta sin paginación
    int totalPages; //total páginas teniendo en cuenta el tamaño de cada página

    List<T> content; //lista de elementos
    Sort sort; //ordenación de la consulta
}
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación **Paginación, filtrado y ordenación:**

Ahora desde el servicio y su implementación podemos crear un nuevo método que devuelva una PaginaDto<CiudadList>:

CiudadServiceImpl.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > srv > impl > CiudadServiceImpl.java > CiudadServiceImpl > findAllPageCiudadList(Pageable)

```java
package edu.profesor.joseramon.dwes_futbol_rest.srv.impl;

import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import edu.profesor.joseramon.dwes_futbol_rest.model.db.CiudadDb;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadInfo;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.CiudadList;
import edu.profesor.joseramon.dwes_futbol_rest.model.dto.PaginaDto;
import edu.profesor.joseramon.dwes_futbol_rest.repository.CiudadRepository;
import edu.profesor.joseramon.dwes_futbol_rest.srv.CiudadService;
import edu.profesor.joseramon.dwes_futbol_rest.srv.mapper.CiudadMapper;

@Service
public class CiudadServiceImpl implements CiudadService{
    private final CiudadRepository ciudadRepository;

    @Autowired
    public CiudadServiceImpl(CiudadRepository ciudadRepository){
        this.ciudadRepository=ciudadRepository;
    }

    public List<CiudadList> findAllCiudadList(){
        return CiudadMapper.INSTANCE.ciudadesToCiudadList(ciudadRepository.findAll());
    }

    public List<CiudadList> findAllCiudadList(Sort sort){
        return CiudadMapper.INSTANCE.ciudadesToCiudadList(ciudadRepository.findAll(sort));
    }

    public PaginaDto<CiudadList> findAllPageCiudadList(Pageable paging){
        Page<CiudadDb> paginaCiudadDb=ciudadRepository.findAll(paging);
        return new PaginaDto<CiudadList>(
                    paginaCiudadDb.getNumber(),//número de página solicitada
                    paginaCiudadDb.getSize(), //tamaño de la página
                    paginaCiudadDb.getTotalElements(),//total de elementos devueltos por la consulta sin paginación
                    paginaCiudadDb.getTotalPages(), //total páginas teniendo en cuenta el tamaño de cada página
                    CiudadMapper.INSTANCE.ciudadesToCiudadList(paginaCiudadDb.getContent()),//lista de elementos
                    paginaCiudadDb.getSort()); //ordenación de la consulta
    }
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

… continuación **Paginación, filtrado y ordenación:**

Y desde el Controller podemos llamar a ese método del servicio.

Fijate que igual que habíamos creado ‘findAllPageCiudadList’ en el servicio deberemos de crear también en el servicio ‘findByNombreContaining’:

CiudadRestController.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > controller > CiudadRestController.java > ...

```
38      /* DESACTIVAMOS EL ANTIGUO /ciudades:
39       @GetMapping("/ciudades")
40       public Collection<CiudadList> getCiudadesList() {
41           return ciudadService.findAllCiudadList();
42       } */
43
44       @GetMapping("/ciudades")
45      public ResponseEntity<Map<String, Object>> getAllCiudades(
46          @RequestParam(required = false) String nombre,
47          @RequestParam(defaultValue = "0") int page,
48          @RequestParam(defaultValue = "3") int size,
49          @RequestParam(defaultValue = "id,asc") String sort) {
50
51        try {
52          // Crear sorts (ordenación de los datos)
53          String[] orden = sort.split(",");
54          Sort sorts;
55          if (orden.length > 1)
56            sorts = Sort.by(Direction.fromString(orden[1]), orden[0]);
57          else // por defecto asc si no se dice nada
58            sorts = Sort.by(Direction.fromString("asc"), orden[0]);
59
60          // Crear solicitud de página nº 'page' de tamaño 'size'
61          // utilizando el orden 'sorts'
62          Pageable paging = PageRequest.of(page, size, sorts);
63
64          PaginaDto<CiudadList> paginaCiudadesList;
65          if (nombre == null) // Sin filtrar por nombre
66            paginaCiudadesList = ciudadService.findAllPageCiudadList(paging);
67          else // filtrando nombre
68            paginaCiudadesList = ciudadService.findByNombreContaining(nombre, paging);
69
70          // Rellenar datos a devolver en el servicio REST
71          List<CiudadList> ciudades = paginaCiudadesList.getContent();
72          Map<String, Object> response = new HashMap<>();
73          response.put("data", ciudades);
74          response.put("currentPage", paginaCiudadesList.getNumber());
75          response.put("pageSize", paginaCiudadesList.getSize());
76          response.put("totalItems", paginaCiudadesList.getTotalElements());
77          response.put("totalPages", paginaCiudadesList.getTotalPages());
78          return new ResponseEntity<>(response, HttpStatus.OK);
79        } catch (Exception e) { // Si hay cualquier tipo de error
80          return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
81        }
82      }
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación **Paginación, filtrado y ordenación:**

Si probamos las consultas siguientes resaltadas veremos que funcionan:

test_Rest.http

```
###
##  getCiudadesByNombreContainingListOrderByName
Send Request
GET http://localhost:8080/api/v1/ciudades/nombre/Val/orden/desc HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades?size=5 HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades?sort=nombre,desc&page=1&size=5 HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades?nombre=Val&page=0&size=10 HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades?sort=nombre,asc&nombre=Val&page=0&size=10 HTTP/1.1
Content-Type: application/json
###
```

... continuación Paginación, filtrado y ordenación:

Solo nos queda la petición en la que le indicamos más de un método de ordenación (primero por habitantes y luego por id):

```
test_Rest.http
test_Rest.http > ...
31  Content-Type: application/json
32  ###
33  ##  getAllCiudades
    Send Request
34  GET http://localhost:8080/api/v1/ciudades?sort=nombre,asc&nombre=Val&page=0&size=10 HTTP/1.1
35  Content-Type: application/json
36  ###
37  ## getAllCiudades
    Send Request
38  GET http://localhost:8080/api/v1/ciudades?sort=habitantes,asc&sort=id,asc&page=4&size=10 HTTP/1.1
39  Content-Type: application/json
40  ###
41
```

¿Como especificamos más de un criterio de ordenación? ¿Que clases debemos cambiar?

… continuación Paginación, filtrado y ordenación:

Lo único que deberemos cambiar es el controller porque la clase 'Sort' nos permite especificar más de un criterio de ordenación gracias a la clase "Order" de org.springframework.data.domain.Sort. Fijate que el parámetro 'sort' del método ahora no es un String sino un vector (String[]):

CiudadRestController.java

src > main > java > edu > profesor > joseramon > dwes_futbol_rest > controller > CiudadRestController.java > CiudadRestContro

```
46      @GetMapping("/ciudades")
47      public ResponseEntity<Map<String, Object>> getAllCiudades(
48          @RequestParam(required = false) String nombre,
49          @RequestParam(defaultValue = "0") int page,
50          @RequestParam(defaultValue = "3") int size,
51          @RequestParam(defaultValue = "id,asc") String[] sort) {
52
53          try {
54              // Crear sorts (ordenación de los datos)
55              List<Order> criteriosOrdenacion= new ArrayList<Order>();
56              //El primer criterio de ordenacion siempre deberá de contener el orden(asc,desc)
57              if(sort[0].contains(",")){ //Hay más de un criterio de ordenación
58                  //Tenemos un vector de ordenaciones en 'sort' y debemos leerlos
59                  for (String criterioOrdenacion : sort) {
60                      String[] orden = criterioOrdenacion.split(",");
61                      if (orden.length > 1)
62                          criteriosOrdenacion.add(new Order(Direction.fromString(orden[1]), orden[0]));
63                      else // por defecto asc si no se dice nada
64                          criteriosOrdenacion.add(new Order(Direction.fromString("asc"), orden[0]));
65                  }
66              } else{ //Solo hay un criterio de ordenación
67                      //El primer elemento del vector de sort es la dirección y el segundo el campo
68                      criteriosOrdenacion.add(new Order(Direction.fromString(sort[1]), sort[0]));
69              }
70              Sort sorts= Sort.by(criteriosOrdenacion);
71
72              // Crear solicitud de página nº 'page' de tamaño 'size'
73              // utilizando el orden 'sorts'
74              Pageable paging = PageRequest.of(page, size, sorts);
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación **Paginación, filtrado y ordenación:**

Si lo probamos las 2 consultas devuelven lo mismo:

test_Rest.http

```
###
##  getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades?sort=nombre,asc&nombre=Val&page=0&size=10 HTTP/1.1
Content-Type: application/json
###
##  getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades?sort=habitantes,asc&sort=id,asc&page=4&size=10 HTTP/1.1
Content-Type: application/json
###
##  getAllCiudades
Send Request
GET http://localhost:8080/api/v1/ciudades?sort=habitantes,asc&sort=id&page=4&size=10 HTTP/1.1
Content-Type: application/json
###
```

Response(335ms)

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 04 Feb 2022 08:05:18 GMT
Connection: close

{
  "totalItems": 58,
  "data": [
    {
      "id": 55,
      "nombre": "Toledo",
      "habitantes": null
    },
    {
      "id": 59,
      "nombre": "Zamora",
      "habitantes": null
    },
    {
      "id": 19,
      "nombre": "Cornellà de Llobregat",
      "habitantes": 85000
    },
    {
      "id": 21,
      "nombre": "Getafe",
      "habitantes": 169000
    },
    {
      "id": 47,
      "nombre": "Sant Sebastià",
      "habitantes": 186000
    },
    {
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

… continuación Paginación, filtrado y ordenación:

Para afianzar los conceptos vistos en esta práctica se pide al alumno **ampliar el servicio rest con las mismas llamadas que en la tabla 'Ciudades' pero sobre la tabla 'Jornadas'** que el alumno puede encontrar en el DRIVE. En data.sql tenéis la definición de la tabla y los inserts:

test_Rest.http | data.sql

src > main > resources > data.sql

```sql
-- Copyright Jose Ramón Cebolla - 2022
-- ----------------------------------------------------
-- Table `ciudades`
-- ----------------------------------------------------
DROP TABLE IF EXISTS ciudades;
CREATE  TABLE IF NOT EXISTS ciudades(
id IDENTITY,
nombre VARCHAR(50),
habitantes BIGINT,
 CONSTRAINT pk_ciudades PRIMARY KEY(id));


-- ----------------------------------------------------
-- Table `jornadas`
-- ----------------------------------------------------
DROP TABLE IF EXISTS `jornadas` ;

CREATE TABLE IF NOT EXISTS `jornadas` (
  `num` INT(11) NOT NULL ,
  `fecha` VARCHAR(10) NULL DEFAULT NULL ,
  CONSTRAINT pk_jornadas PRIMARY KEY (`num`) );
-- ----------------------------------------------------
-- INSERCIÓN DE DATOS
-- ----------------------------------------------------
```

UD 3: Bases de datos y servicios REST
6.- Paginación en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

EJERCICIO:

Sube la aplicación final al moodle.

Para ello:

1º Haz un "Maven Clean" para dejar solo los fichero fuentes y quitar momentaneamente los necesarios para ejecutar la aplicación (dependencias).

2º Comprime la carpeta de tu aplicación y ponle como nombre al fichero comprimido UD3_practica6_nombreAlumno.tar.gz donde nombreAlumno es el nombre del alumno que entrega la práctica.

3º Súbela al moodle.

IMPORTANTE: No comprimir en RAR, porque Ubuntu no lo lee bien y en clase tenemos Ubuntu. Si tuviesemos Windows, podemos comprimir en ZIP.