UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

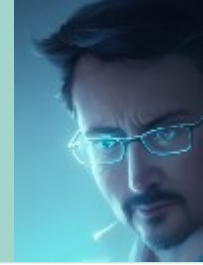
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Objetivos de la sesión:

- *Realizar consultas simples con* ***@Query***
- *Utilizar los* ***filtros simples por atributo*** *que proporciona JpaRepository*
- ***Ordenar los resultados*** *de las consultas mediante las facilidades que nos proporciona JpaRepository.*
- *Saber realizar* ***consultas con filtros y ordenaciones en Servicios RESTful.***
- ***Documentar API's Rest con OpenAPI y Swagger.***

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



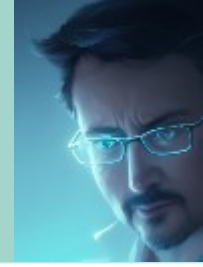
Hemos visto como hacer CRUD sobre tablas, pero …

¿ Que pasa si necesitamos hacer una consulta sobre la base de datos ?
Por ejemplo, queremos listar los alumnos de DAW directamente contra la BD sin tener que recuperar todos los registros (que podrían ser miles) y luego tener que filtrarlos ¿ Como lo podemos hacer?

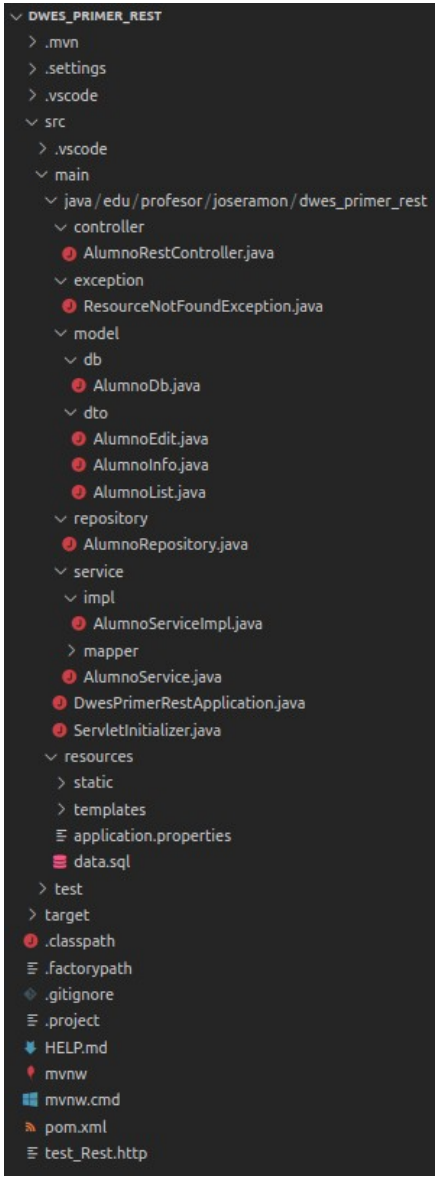
En esta práctica vamos a utilizar y ampliar el servicio REST que hacia CRUD sobre la tabla “Alumnos” para analizar las facilidades que proporciona Spring Data JPA y la interfaz del repositorio “JpaRepository” para realizar consultas sobre la base de datos.

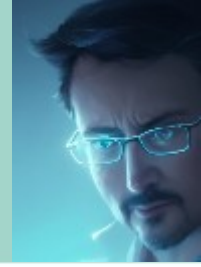
UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



Utiliza como base tu primer proyecto RESTFul “dwes_primer_rest” :





1º @Query:

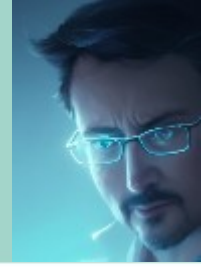
Esta anotación permiten realizar consultas directamente sobre la BD, tanto con JPQL como con sentencias SQL nativas. Realiza el primer cambio sobre AlumnoRepository.java :

AlumnoRepository.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > La

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

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import edu.profesor.joseramon.dwes_primer_rest.model.db.AlumnoDb;
import java.util.Collection;
import org.springframework.data.jpa.repository.Query;

@Repository
public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{
    //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
    //Utilizamos el nombre de la clase 'AlumnoDb'
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
    Collection<AlumnoDb> findAllAlumnoDbDAW();
    //Podemos utilizar SQL nativo (native SQL)
    //Utilizamos el nombre de la tabla 'alumnos' en la BD
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbDAM();
}
```

¿ Cual es el motivo/ventaja de utilizar un Collection<AlumnoDb>?



... continuación 1º @Query:

Collection es la interface padre de otras interfaces y si nos interesa podríamos hacer que en el servicio devolviera un conjunto (Set) en vez de una lista (List). Realiza los demás cambios :

AlumnoRepository.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java

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

import org.springframework.data.jpa.repository.Query;…

@Repository
public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{

    //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
    Collection<AlumnoDb> findAllAlumnoDbDAW();

    //Podemos utilizar SQL nativo (native SQL)
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbDAM();
}
```

AlumnoService.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > service > AlumnoService

```java
package edu.profesor.joseramon.dwes_primer_rest.service;

import java.util.List;…

public interface AlumnoService {
    public Optional<AlumnoEdit> getAlumnoEditByDni(String dni);
    public Optional<AlumnoInfo> getAlumnoInfoByDni(String dni);
    public AlumnoEdit save(AlumnoEdit alumnoEdit);
    public String deleteByDni(String dni);
    public Optional<AlumnoEdit> update(AlumnoEdit alumnoEdit);
    public List<AlumnoList> findAllAlumnoList();
    public List<AlumnoList> findAllAlummnosListDAW();
    public List<AlumnoList> findAllAlummnosListDAM();
}
```

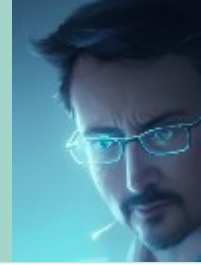
AlumnoServiceImpl.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > service > impl > AlumnoServiceImpl.java > Language Support for Java(TM) by Red Hat > AlumnoS

```java
    }

    @Override
    public List<AlumnoList> findAllAlummnosListDAW() {
        return AlumnoMapper.INSTANCE.alumnosToAlumnosList((List<AlumnoDb>) alumnoRepository.findAllAlumnoDbDAW());
    }

    @Override
    public List<AlumnoList> findAllAlummnosListDAM() {
        return AlumnoMapper.INSTANCE.alumnosToAlumnosList((List<AlumnoDb>) alumnoRepository.findAllAlumnoDbDAM());
    }

}
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



... continuación 1º @Query:

Ya podemos añadir los métodos en el controller y las pruebas en “test_Rest.http” para probarlo:

AlumnoRestController.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller >

```
33      }
34
35      @GetMapping("/alumnos")
36      public List<AlumnoList> getAlumnosList() {
37          return alumnoService.findAllAlumnoList();
38      }
39
40      @GetMapping("/alumnosDAW")
41      public List<AlumnoList> getAlumnosListDAW() {
42          return alumnoService.findAllAlummnosListDAW();
43      }
44
45      @GetMapping("/alumnosDAM")
46      public List<AlumnoList> getAlumnosListDAM() {
47          return alumnoService.findAllAlummnosListDAM();
48      }
```

test_Rest.http

test_Rest.http > GET /api/v1/alumnosDAM

```
1   ## getAlumnosList
    Send Request
2   GET http://localhost:8080/api/v1/alumnos HTTP/1.1
3   Content-Type: application/json
4   ###
5   ## getAlumnosListDAW
    Send Request
6   GET http://localhost:8080/api/v1/alumnosDAW HTTP/1.1
7   Content-Type: application/json
8   ###
9   ## getAlumnosListDAM
    Send Request
10  GET http://localhost:8080/api/v1/alumnosDAM HTTP/1.1
11  Content-Type: application/json
12  ###
13
14
15  ## newAlumnoEdit
    Send Request
```

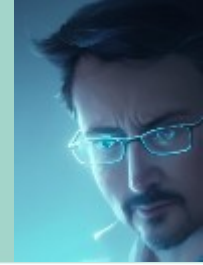
Response(180ms)

```
1   HTTP/1.1 200
2   Content-Type: application/json
3   Transfer-Encoding: chunked
4   Date: Sun, 21 Feb 2021 18:23:30 GMT
5   Connection: close
6
7   [
8     {
9       "dni": "11111111A",
10      "nombre": "Jose Garcia",
11      "edad": 21,
12      "ciclo": "DAM",
13      "curso": 1,
14      "erasmus": false,
15      "modificado": null,
16      "erasmusChecked": ""
17    }
```

Si eres un alumno curioso tu siguiente pregunta podría ser:
¿Como podemos parametrizar estas consultas para devolver los datos en función de dichos parámetros? Por ejemplo indicarle el ciclo y que devuelva todos los alumnos de dicho ciclo ...

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



… continuación 1º @Query:

Para introducir parámetros es bastante sencillo tanto en JPQL como en SQL nativo. A continuación se puede ver los cambios en AlumnoRepository.java, AlumnoRestController.java y test_Rest.http. Realiza los cambios necesarios en AlumnoService y AlumnoServiceImpl para poder ejecutar las llamadas REST remarcadas:

AlumnoRepository.java

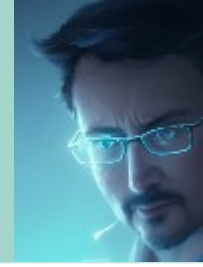
```
src > main > java > edu > profesor > joseramon > dwes_p
4   import org.springframework.stereot
5   import edu.profesor.joseramon.dwes_
6   import java.util.Collection;
7   import org.springframework.data.jp
8   import org.springframework.data.re
9
10  @Repository
11  public interface AlumnoRepository
12      //Por defecto se utiliza Jakar
13      //Utilizamos el nombre de la c
14      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
15      Collection<AlumnoDb> findAllAlumnoDbDAW();
16      //Podemos utilizar SQL nativo (native SQL)
17      //Utilizamos el nombre de la tabla 'alumnos' en la BD
18      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
19      nativeQuery = true)
20      Collection<AlumnoDb> findAllAlumnoDbDAM();
21
22      //CONSULTAS CON PARÁMETROS:
23      //JPQL: AlumnoDb
24      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
25      Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
26      //native SQL: alumnos
27      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
28      nativeQuery = true)
29      Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
30                                                    @Param("pais") String pais);
31  }
```

AlumnoRestController.java

```
src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller > AlumnoRestController.java > Language Support fo
93      @GetMapping("/alumnos/ciclo/{ciclo}")
94      public List<AlumnoList> getAlumnosListCiclo(@PathVariable("ciclo") String ciclo) {
95          return alumnoService.findAllAlummnosListCiclo(ciclo);
96      }
97      @GetMapping("/alumnos/ciclo/{ciclo}/pais/{pais}")
98      public List<AlumnoList> getAlumnosListCicloPais(@PathVariable("ciclo") String ciclo,
99                                                      @PathVariable("pais") String pais) {
100         return alumnoService.findAllAlummnosListCicloPais(ciclo,pais);
101     }
102 }
```

test_Rest.http

```
test_Rest.http > ...
12  ###
13  ## getAlumnosListCiclo
    Send Request
14  GET http://localhost:8080/api/v1/alumnos/ciclo/DAW HTTP/1.1
15  Content-Type: application/json
16  ###
17  ## getAlumnosListCicloPais
    Send Request
18  GET http://localhost:8080/api/v1/alumnos/ciclo/DAW/pais/ES HTTP/1.1
19  Content-Type: application/json
20  ###
```



2º Filtros simples por atributo en JpaRepository :

Si AlumnosDb tiene todos estos atributos, ¿Existe alguna manera de filtrar por esos atributos de manera individual sin tener que utilizar una Query?

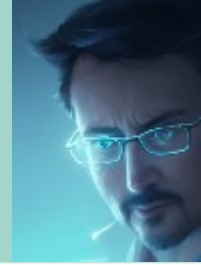
AlumnoDb.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > model > db > AlumnoDb.java > {} edu.profesor.joseramon.dwes_primer_rest.mod

```java
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@Table(name = "alumnos")
public class AlumnoDb implements Serializable{
    private static final long serialVersionUID = -818542778373595260L;
    @Id
    @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y una letra")
    private String dni;
    @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 carácteres")
    private String nombre;
    @NotNull(message = "La edad no puede estar vacia")
    @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor a 18 y menor o igual a 99")
    @Digits(integer = 2,fraction = 0, message = "La edad no puede tener decimales ni más de 2 dígitos")
    private Integer edad;
    private String ciclo;
    @NotNull(message = "El curso no puede estar vacio")
    @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato incorrecto")
    @Range(min = 1, max = 2, message = "El curso solo admite los valores 1 o 2")
    private Integer curso;
    private boolean erasmus=false;
    private String lenguajeFavorito="";
    private String genero;
    private String horario;
    private String pais;
    private String hobbies;
}
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



... continuación 2º Filtros simples por atributo en JpaRepository :

Añade en AlumnoRepository un nuevo método que comience por f...

AlumnoDb.java

```
19  @NoArgsConstructor
20  @AllArgsConstructor
21  @Data
22  @Entity
23  @Table(name = "alumnos")
24  public class AlumnoDb implements Serializable{
25      private static final long serialVersionUID = -8185427783735952600L;
26      @Id
27      @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y una letra")
28      private String dni;
29      @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 carácteres")
30      private String nombre;
31      @NotNull(message = "La edad no puede estar vacia")
32      @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor a 18 y menor o igual a 99")
33      @Digits(integer = 2,fraction = 0, message = "La edad no puede tener decimales ni más de 2 dígitos")
34      private Integer edad;
35      private String ciclo;
36      @NotNull(message = "El curso no puede estar vacio")
37      @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato incorrecto")
38      @Range(min = 1, max = 2, message = "El curso solo admite los valores 1 o 2")
39      private Integer curso;
40      private boolean erasmus=false;
41      private String lenguajeFavorito="";
42      private String genero;
43      private String horario;
44      private String pais;
45      private String hobbies;
46  }
```

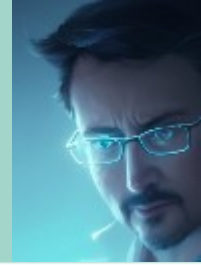
AlumnoRepository.java

```
9
10  @Repository
11  public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{
12      //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
13      //Utilizamos el nombre de la clase 'AlumnoDb'
14      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
15      Collection<AlumnoDb> findAllAlumnoDbDAW();
16      //Podemos utilizar SQL nativo (native SQL)
17      //Utilizamos el nombre de la tabla 'alumnos' en la BD
18      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
19      nativeQuery = true)
20      Collection<AlumnoDb> findAllAlumnoDbDAM();
21
22      //CONSULTAS CON PARÁMETROS:
23      //JPQL: AlumnoDb
24      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
25      Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
26      //native SQL: alumnos
27      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
28      nativeQuery = true)
29      Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
30                                                    @Param("pais") String pais);
31
32      //Métodos automáticmos en Spring Data JPA
33      f
34          findByCiclo(String ciclo);
35  }       findByCurso(Integer curso);
36          findByDni(String dni);
            findByEdad(Integer edad);
            findByGenero(String genero);
            findByHobbies(String hobbies);
            findByHorario(String horario);
            findByLenguajeFavorito(String lenguajeFavori…
            findByNombre(String nombre);
            findByPais(String pais);
            findAll() : List<AlumnoDb>
            findAll(Example<S> example) : List<S>
```

Como observamos tenemos varios método ya creados que podemos utilizar sin tener que indicar la Consulta:

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



... continuación 2º Filtros simples por atributo en JpaRepository :

Añade en AlumnoRepository los siguientes métodos teniendo en cuenta como se ha hecho en "findByDni": Nota: Aunque por defecto nos cree un List<AlumnoDb> cambiemoslo a Collection<AlumnoDb>

- findByDni
- findByCiclo
- findByHorario
- findByEdad

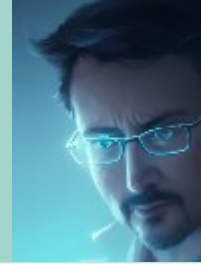
AlumnoRepository.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > ...

```
9
10  @Repository
11  public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{
12      //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
13      //Utilizamos el nombre de la clase 'AlumnoDb'
14      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
15      Collection<AlumnoDb> findAllAlumnoDbDAW();
16      //Podemos utilizar SQL nativo (native SQL)
17      //Utilizamos el nombre de la tabla 'alumnos' en la BD
18      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
19      nativeQuery = true)
20      Collection<AlumnoDb> findAllAlumnoDbDAM();
21
22      //CONSULTAS CON PARÁMETROS:
23      //JPQL: AlumnoDb
24      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
25      Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
26      //native SQL: alumnos
27      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
28      nativeQuery = true)
29      Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
30                                                    @Param("pais") String pais);
31
32      //Métodos automáticmos en Spring Data JPA
33      Collection<AlumnoDb> findByDni(String dni);
34
35  }
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



... continuación 2º Filtros simples por atributo en JpaRepository :

Añade en el Service y su implementación los métodos necesarios teniendo en cuenta el siguiente AlumnoRestController.java y test_Rest.http:

AlumnoRestController.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller > AlumnoRestController.java > ...

```java
105     //Métodos Spring Data JPA
106     @GetMapping("/alumnosList/dni/{dni}")
107     public List<AlumnoList> getAlumnosListByDni(@PathVariable("dni") String dni) {
108         return alumnoService.findAlummnosListByDni(dni);
109     }
110
111     @GetMapping("/alumnosList/ciclo/{ciclo}")
112     public List<AlumnoList> getAlumnosListByCiclo(@PathVariable("ciclo") String ciclo) {
113         return alumnoService.findAlummnosListByCiclo(ciclo);
114     }
115
116     @GetMapping("/alumnosList/horario/{horario}")
117     public List<AlumnoList> getAlumnosListByHorario(@PathVariable("horario") String horario) {
118         return alumnoService.findAlummnosListByHorario(horario);
119     }
120
121     @GetMapping("/alumnosList/edad/{edad}")
122     public List<AlumnoList> getAlumnosListByEdad(@PathVariable("edad") Integer edad) {
123         return alumnoService.findAlummnosListByEdad(edad);
124     }
125 }
```

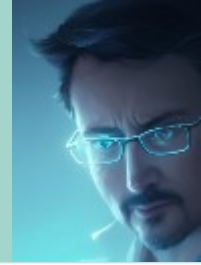
test_Rest.http

test_Rest.http > ...

```
115 ###
116 ## findAlummnosListByDni
    Send Request
117 GET http://localhost:8080/api/v1/alumnosList/dni/11111111A HTTP/1.1
118 Content-Type: application/json
119 ###
120 ## getAlumnosListByCiclo
    Send Request
121 GET http://localhost:8080/api/v1/alumnosList/ciclo/DAW HTTP/1.1
122 Content-Type: application/json
123 ###
124 ## getAlumnosListByHorario
    Send Request
125 GET http://localhost:8080/api/v1/alumnosList/horario/T HTTP/1.1
126 Content-Type: application/json
127 ###
128 ## getAlumnosListByEdad
    Send Request
129 GET http://localhost:8080/api/v1/alumnosList/edad/21 HTTP/1.1
130 Content-Type: application/json
131 ###
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



3º Ordenación en JpaRepository :

Si queremos ordenar existe la primera opción de incluir la ordenación en la consulta SQL del repositorio como podemos ver en el ejemplo sombreado:

Pero si filtramos por N criterios (ciclo, horario, edad,…) y queremos en cada criterio de filtrado tener la opción de poder ordenar por cualquier otro atributo (M atributos), tendremos que hacer

NxM métodos distintos!!

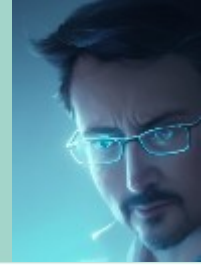
Y eso que no contamos el cambio de dirección (ASC, DESC)

AlumnoRepository.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > Language Support for

```
10  @Repository
11  public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{
12      //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
13      //Utilizamos el nombre de la clase 'AlumnoDb'
14      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
15      Collection<AlumnoDb> findAllAlumnoDbDAW();
16      //Podemos utilizar SQL nativo (native SQL)
17      //Utilizamos el nombre de la tabla 'alumnos' en la BD
18      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
19      nativeQuery = true)
20      Collection<AlumnoDb> findAllAlumnoDbDAM();
21
22      //CONSULTAS CON PARÁMETROS:
23      //JPQL: AlumnoDb
24      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
25      Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
26      //native SQL: alumnos
27      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
28      nativeQuery = true)
29      Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
30                                                    @Param("pais") String pais);
31
32      //Métodos automáticos en Spring Data JPA
33      Collection<AlumnoDb> findByDni(String dni);
34      Collection<AlumnoDb> findByCiclo(String ciclo);
35      Collection<AlumnoDb> findByHorario(String horario);
36      Collection<AlumnoDb> findByEdad(Integer edad);
37
38      //Ordenación de resultados
39      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo ORDER BY a.curso ASC")
40      Collection<AlumnoDb> findAllAlumnoDbCicloOrderByCurso(@Param("ciclo") String ciclo);
41  }
```

¿Existe alguna manera de ordenar los resultados de manera más sencilla sin tener que hacer tantos métodos?



... continuación 3º Ordenación en JpaRepository :

Los métodos del repositorio que utilizan consultas pueden tener un parámetro adicional de tipo “Sort” y servirá para ordenar el resultado según ese criterio de ordenación gracias a la clase ***org.springframework.data.domain.Sort*** :

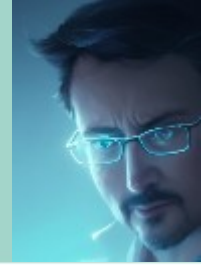
AlumnoRepository.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > Language Support for J

```java
import org.springframework.data.domain.Sort;

@Repository
public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{
    //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
    //Utilizamos el nombre de la clase 'AlumnoDb'
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
    Collection<AlumnoDb> findAllAlumnoDbDAW();
    //Podemos utilizar SQL nativo (native SQL)
    //Utilizamos el nombre de la tabla 'alumnos' en la BD
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbDAM();

    //CONSULTAS CON PARÁMETROS:
    //JPQL: AlumnoDb
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
    Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
    //native SQL: alumnos
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
                                                  @Param("pais") String pais);

    //Métodos automáticos en Spring Data JPA
    Collection<AlumnoDb> findByDni(String dni);
    Collection<AlumnoDb> findByCiclo(String ciclo);
    Collection<AlumnoDb> findByHorario(String horario);
    Collection<AlumnoDb> findByEdad(Integer edad);

    //Ordenación de resultados
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo ORDER BY a.curso ASC")
    Collection<AlumnoDb> findAllAlumnoDbCicloOrderByCurso(@Param("ciclo") String ciclo);
    //Consultas con varias ordenaciones en Spring Data JPA
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
    Collection<AlumnoDb> findByCicloOrderBy(String ciclo,Sort sort);
}
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



... continuación 3º Ordenación en JpaRepository :

Si lo ejecutamos veremos como nos devuelve los alumnos de DAW ordenados por edad y podemos comprobar en el log la consulta que se realiza:

AlumnoRestController.java

```
src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller > AlumnoRestController.java > Language Support for Jav
127
128     //Ordenación de resultados
129     @GetMapping("/alumnos/ciclo/{ciclo}/OrderByCurso")
130     public List<AlumnoList> getAlumnosListCicloOrderByCurso(@PathVariable("ciclo") String cicl
131         return alumnoService.findAllAlummnosListCicloOrderByCurso(ciclo);
132     }
133     //Métodos automáticos con Ordenación en Spring Data JPA
134     @GetMapping("/alumnosList/ciclo/{ciclo}/OrderBy/{atributoOrden}/{direccion}")
135     public List<AlumnoList> getAlumnosListByCicloOrderBy(@PathVariable("ciclo") String ciclo,
136                                                          @PathVariable("atributoOrden") String atributoOrden,
137                                                          @PathVariable("direccion") String direccion) {
138         return alumnoService.findByCicloOrderBy(ciclo,Sort.by(Direction.fromString(direccion),atributoOrden));
139     }
```

test_Rest.http

```
test_Rest.http > ...
136  ## getAlumnosListByCicloOrderBy
     Send Request
137  GET http://localhost:8080/api/v1/alumnosList/ciclo/DAW/OrderBy/edad/DESC HTTP/1.1
138  Content-Type: application/json
139  ###
```

Realiza los cambios para que el controlador pueda llamar a :

- findByCicloOrderBy
- findByEdadOrderBy

Y añade la petición:

```
## getAlumnosListByEdadOrderBy
Send Request
GET http://localhost:8080/api/v1/alumnosList/edad/43/OrderBy/dni/ASC HTTP/1.1
Content-Type: application/json
###
```

```
Hibernate:
    select
        alumnodb0_.dni as dni1_0_,
        alumnodb0_.ciclo as ciclo2_0_,
        alumnodb0_.curso as curso3_0_,
        alumnodb0_.edad as edad4_0_,
        alumnodb0_.erasmus as erasmus5_0_,
        alumnodb0_.genero as genero6_0_,
        alumnodb0_.hobbies as hobbies7_0_,
        alumnodb0_.horario as horario8_0_,
        alumnodb0_.lenguaje_favorito as lenguaje9_0_,
        alumnodb0_.nombre as nombre10_0_,
        alumnodb0_.pais as pais11_0_
    from
        alumnos alumnodb0_
    where
        alumnodb0_.ciclo=?
    order by
        alumnodb0_.edad desc
```

Response(186ms)

```
1   HTTP/1.1 200
2   Content-Type: application/json
3   Transfer-Encoding: chunked
4   Date: Sun, 30 Jan 2022 21:45:30 GMT
5   Connection: close
6
7   [
8     {
9       "dni": "33333333C",
10      "nombre": "Pedro Martínez",
11      "edad": 43,
12      "ciclo": "DAW",
13      "curso": 2,
14      "erasmus": false,
15      "modificado": null,
16      "erasmusChecked": ""
17    },
18    {
19      "dni": "22222222B",
20      "nombre": "Maria Gonzalez",
21      "edad": 32,
22      "ciclo": "DAW",
23      "curso": 2,
24      "erasmus": true,
25      "modificado": null,
26      "erasmusChecked": "checked"
27    }
28  ]
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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

... continuación 3º Ordenación en JpaRepository :

Para un alumno curioso, ahora la pregunta podría ser:

¿No se puede realizar ordenaciones con los métodos automáticos sin tener que especificar la consulta?

AlumnoRepository.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > AlumnoRepository

```java
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.data.domain.Sort;

@Repository
public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{
    //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
    //Utilizamos el nombre de la clase 'AlumnoDb'
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
    Collection<AlumnoDb> findAllAlumnoDbDAW();
    //Podemos utilizar SQL nativo (native SQL)
    //Utilizamos el nombre de la tabla 'alumnos' en la BD
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbDAM();

    //CONSULTAS CON PARÁMETROS:
    //JPQL: AlumnoDb
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
    Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
    //native SQL: alumnos
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
                                                  @Param("pais") String pais);

    //Métodos automáticos en Spring Data JPA
    Collection<AlumnoDb> findByDni(String dni);
    Collection<AlumnoDb> findByCiclo(String ciclo);
    Collection<AlumnoDb> findByHorario(String horario);
    Collection<AlumnoDb> findByEdad(Integer edad);

    //Ordenación de resultados
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo ORDER BY a.curso ASC")
    Collection<AlumnoDb> findAllAlumnoDbCicloOrderByCurso(@Param("ciclo") String ciclo);
    //Consultas con varias ordenaciones en Spring Data JPA
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
    Collection<AlumnoDb> findByCicloOrderBy(String ciclo,Sort sort);
    @Query("SELECT a FROM AlumnoDb a WHERE a.edad=:edad")
    Collection<AlumnoDb> findByEdadOrderBy(Integer edad,Sort sort);
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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

... continuación 3º Ordenación en JpaRepository :

La respuesta lógicamente es que 'Sí'. En la versión 2.6.2 de Spring Boot todavía no podíamos especificar que el tipo devuelto era Collection, sino que debíamos especificar List para que funcionara correctamente.

Sin embargo, en la versión 2.7.7 vemos que podemos hacerlo sin problemas :

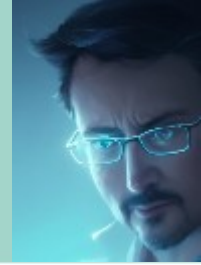
AlumnoRepository.java

src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > Language Support for Java(

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

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

public interface AlumnoRepository  extends JpaRepository<AlumnoDb,String>{
    //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
    //Utilizamos el nombre de la clase 'AlumnoDb'
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
    Collection<AlumnoDb> findAllAlumnoDbDAW();
    //Podemos utilizar SQL nativo (native SQL)
    //Utilizamos el nombre de la tabla 'alumnos' en la BD
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbDAM();

    //CONSULTAS CON PARÁMETROS:
    //JPQL: AlumnoDb
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
    Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
    //native SQL: alumnos
    @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
    nativeQuery = true)
    Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
                                                  @Param("pais") String pais);

    //Métodos automáticos en Spring Data JPA
    Collection<AlumnoDb> findByDni(String dni);
    Collection<AlumnoDb> findByCiclo(String ciclo);
    Collection<AlumnoDb> findByHorario(String horario);
    Collection<AlumnoDb> findByEdad(Integer edad);

    //Ordenación de resultados
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo ORDER BY a.curso ASC")
    Collection<AlumnoDb> findAllAlumnoDbCicloOrderByCurso(@Param("ciclo") String ciclo);
    //Consultas con varias ordenaciones en Spring Data JPA
    @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
    Collection<AlumnoDb> findByCicloOrderBy(String ciclo,Sort sort);
    @Query("SELECT a FROM AlumnoDb a WHERE a.edad=:edad")
    Collection<AlumnoDb> findByEdadOrderBy(Integer edad,Sort sort);

    //Métodos automáticos en Spring Data JPA
    //OLD: versión 2.6.2: List<AlumnoDb> findByCiclo(String ciclo,Sort sort);
    Collection<AlumnoDb> findByCiclo(String ciclo,Sort sort);
    Collection<AlumnoDb> findByHorario(String horario,Sort sort);
    Collection<AlumnoDb> findByEdad(Integer edad,Sort sort);
}
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



… continuación 3º Ordenación en JpaRepository :

Para comprobarlo modifica AlumnoService y AlumnoServiceImpl para añadir un método nuevo findByCiclo(String ciclo,Sort sort) que permita filtrar por ciclo y especificar el criterio de ordenación del resultado mediante un método automático. Modifica AlumnoController y test_Rest.http para realizar la siguiente comprobación:

AlumnoRestController.java

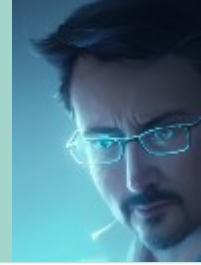
```
src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller > AlumnoRestController.ja
127
128     //Ordenación de resultados
129     @GetMapping("/alumnos/ciclo/{ciclo}/OrderByCurso")
130     public List<AlumnoList> getAlumnosListCicloOrderByCurso(@PathVariable
131         return alumnoService.findAllAlummnosListCicloOrderByCurso(ciclo);
132     }
133     //Métodos con Ordenación SQL en Spring Data JPA
134     @GetMapping("/alumnosList/ciclo/{ciclo}/OrderBy/{atributoOrden}/{dire
135     public List<AlumnoList> getAlumnosListByCicloOrderBy(@PathVariable(
136                                         @PathVariable("atributoOrden"
137                                         @PathVariable("direccion") St
138         return alumnoService.findByCicloOrderBy(ciclo,Sort.by(Direction.
139     }
140     @GetMapping("/alumnosList/edad/{edad}/OrderBy/{atributoOrden}/{direc
141     public List<AlumnoList> getAlumnosListByEdadOrderBy(@PathVariable("e
142                                         @PathVariable("atributoOrden"
143                                         @PathVariable("direccion") St
144         return alumnoService.findByEdadOrderBy(edad,Sort.by(Direction.fr
145     }
146     //Métodos automáticos con Ordenación en Spring Data JPA
147     @GetMapping("/alumnos/ciclo/{ciclo}/OrderBy/{atributoOrden}/{direccion}")
148     public List<AlumnoList> getAlumnosByCicloOrderBy(@PathVariable("ciclo") String ciclo,
149                                         @PathVariable("atributoOrden") String atributoOrden,
150                                         @PathVariable("direccion") String direccion) {
151         return alumnoService.findByCiclo(ciclo,Sort.by(Direction.fromString(direccion),atributoOrden));
152     }
153 }
```

test_Rest.http

```
test_Rest.http > ...
    Send Request
133 GET http://localhost:8080/api/v1/alumnos/ciclo/DAW/OrderByCurso HTTP/1.1
134 Content-Type: application/json
135 ###
136 ## getAlumnosListByCicloOrderBy
    Send Request
137 GET http://localhost:8080/api/v1/alumnosList/ciclo/DAW/OrderBy/edad/DESC HTTP/1.1
138 Content-Type: application/json
139 ###
140 ## getAlumnosListByEdadOrderBy
    Send Request
141 GET http://localhost:8080/api/v1/alumnosList/edad/43/OrderBy/dni/ASC HTTP/1.1
142 Content-Type: application/json
143 ###
144 ## getAlumnosByCicloOrderBy
    Send Request
145 GET http://localhost:8080/api/v1/alumnos/ciclo/DAW/OrderBy/edad/DESC HTTP/1.1
146 Content-Type: application/json
147 ###
```

UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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

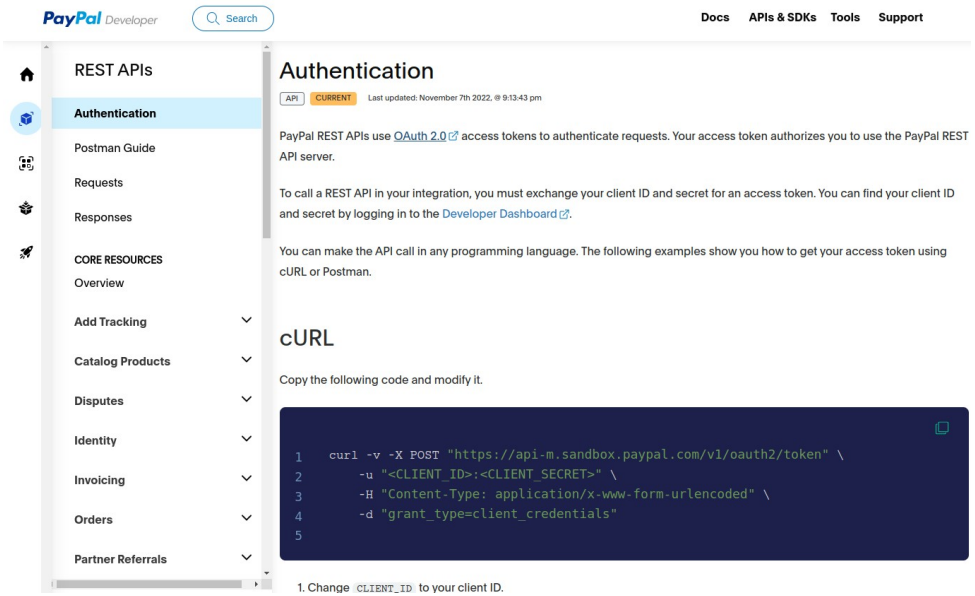


4º Documentar la API Rest con OpenAPI y Swagger:

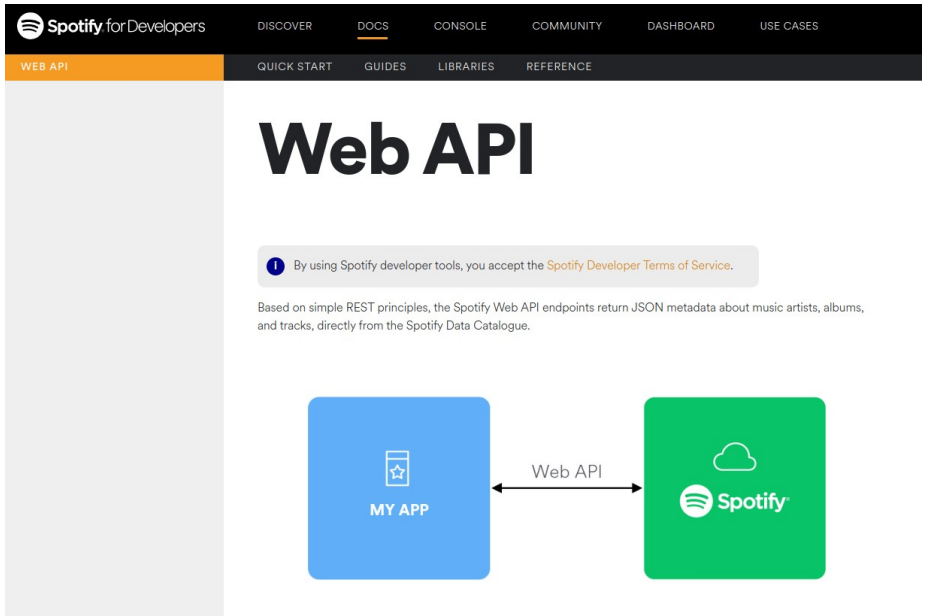
Actualmente muchas aplicaciones webs suelen dividirse en ***2 proyectos distintos***, el projecto que contiene el ***front-end*** y el projecto que contiene el ***back-end***. Normalmente *el back-end expone una API Rest que consume el front-end*, por lo que es importante que exista una ***documentación clara con las especificaciones de la API del back-end*** para que pueda utilizarse sin problemas.

Ejemplos de API's públicas son :

Paypal:
https://developer.paypal.com/api/rest/authentication/



Spotify:
https://developer.spotify.com/documentation/web-api/



... continuación 4º Documentar la API Rest con OpenAPI y Swagger:

Hay confusión entre los terminos OpenAPI y Swagger por lo que se suelen utilizar para referirse a lo mismo. Siendo estrictos:

· La Especificación ***OpenAPI*** (anteriormente especificación Swagger) es un *standard de descripción de API para API REST*. Un archivo OpenAPI le permite describir y documentar la lista de recursos y sus operaciones disponibles en una API Rest de manera informativa y fácil de entender.

· ***Swagger*** es un *conjunto de herramientas de código abierto creadas en torno a la especificación OpenAPI* que puede ayudar a diseñar, crear, documentar y consumir API REST.

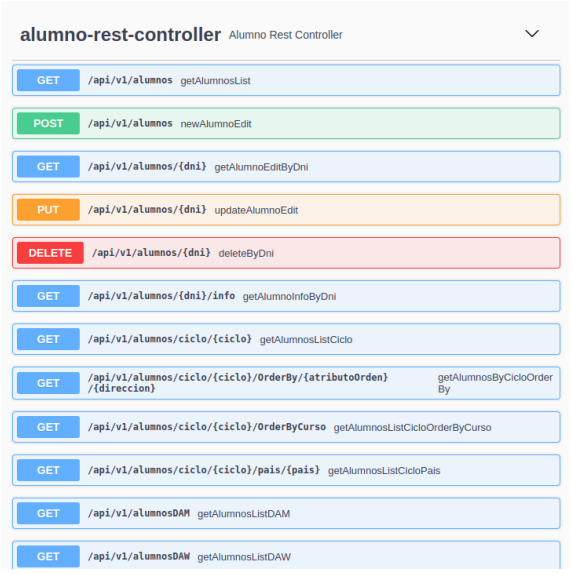
En Spring boot vamos a utilizar la ***dependencia Springfox***, un conjunto de librerias de Spring escritas en Java que extiende el soporte de Swagger permitiendo automatizar la generación de especificaciones para las API's que devuelven JSON's.

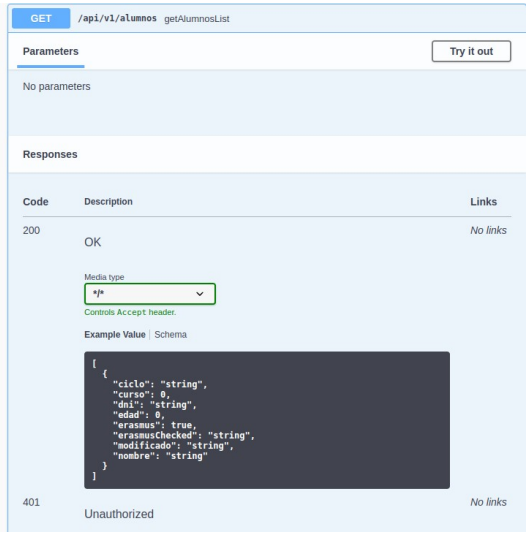
UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

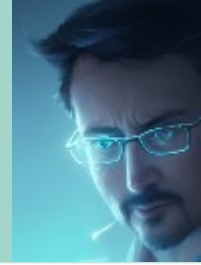
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación 4º Documentar la API Rest con OpenAPI y Swagger:

Podremos especificar de una forma rápida y sencilla la lista de parámetros y la estructura tanto del cuerpo de la solicitud como el cuerpo de la respuesta para realizar una operación:





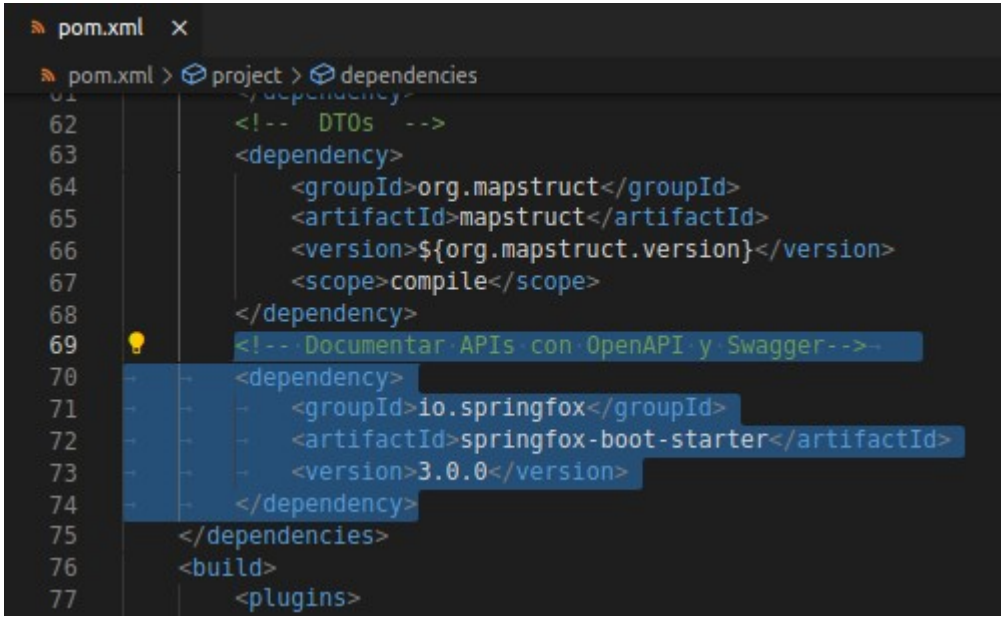


... continuación 4º Documentar la API Rest con OpenAPI y Swagger:

Para utilizar Springfox y Swagger en nuestro proyecto bastará con realizar estos **2 pasos**:

1º Paso: Añadir la ***dependencia de springfox*** al pom.xml:

```
<!-- Documentar APIs con OpenAPI y Swagger-->
        <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-boot-starter</artifactId>
                <version>3.0.0</version>
        </dependency>
```



… continuación 4º Documentar la API Rest con OpenAPI y Swagger:

2º Paso: Configurar una nueva linea en application.properties:

#Configuración OpenAPI y Swagger
spring.mvc.pathmatch.matching-strategy=ant-path-matcher

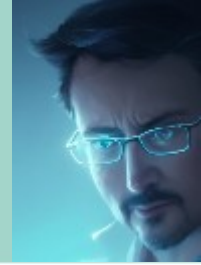
application.properties

src > main > resources > application.properties

```
17  spring.jpa.properties.hibernate.generate_statistics=true
18  # Habilitar LOGGER de las estadísticas de hibernate
19  logging.level.org.hibernate.stat=debug
20  # Mostrar que consultas esta realizando Hibernate
21  spring.jpa.show-sql=true
22  # Formatear las consultas
23  spring.jpa.properties.hibernate.format_sql=true
24  # Mostrar los parametros que estan enviandose a las consultas
25  logging.level.org.hibernate.type=debug
26  #FIN CONFIGURACIÓN SOLO durante las pruebas
27
28  #Configuración OpenAPI y Swagger
29  spring.mvc.pathmatch.matching-strategy=ant-path-matcher
```

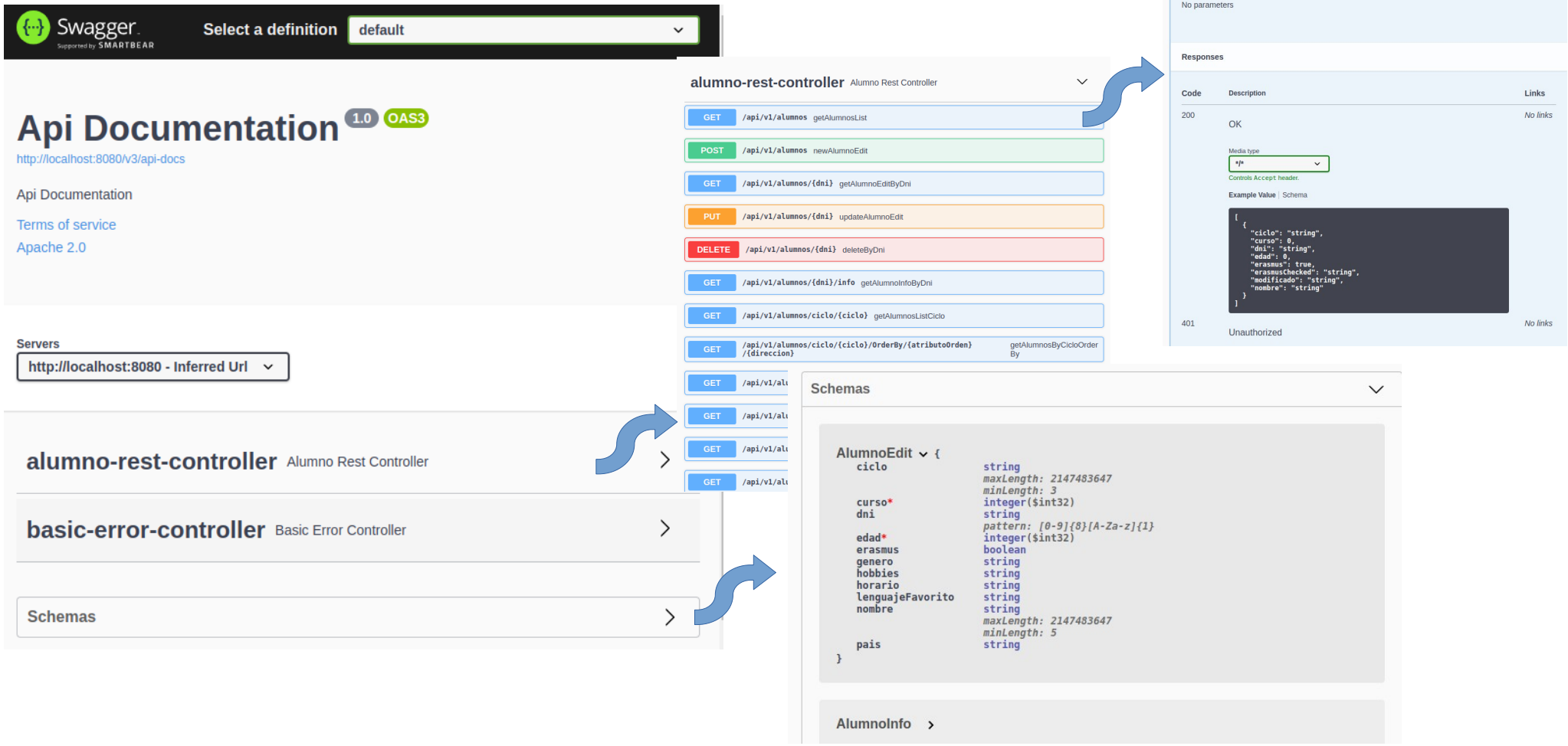
UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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



... continuación 4º Documentar la API Rest con OpenAPI y Swagger:

Y si lo ejecutamos y ponemos la url http://localhost:8080/swagger-ui/index.html podemos ver el resultado y desplegar en los subapartados:

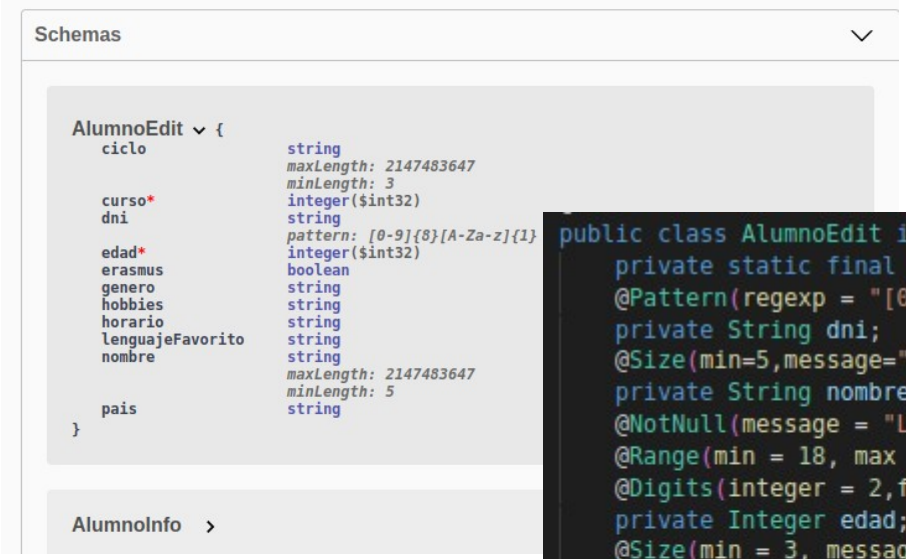


UD 3: Bases de datos y servicios REST
5.- Consultas en Spring Data JPA

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

... continuación 4º Documentar la API Rest con OpenAPI y Swagger:

Si nos fijamos en 'AlumnoEdit' ha sido capaz de leer las validaciones y ponerlas en la documentación. En otra práctica aundaremos más en como personalizar la salida:




```java
public class AlumnoEdit implements  Serializable{
    private static final long serialVersionUID = 1L;
    @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y una letra")
    private String dni;
    @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 carácteres")
    private String nombre;
    @NotNull(message = "La edad no puede estar vacia")
    @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor a 18 y menor o igual a 99")
    @Digits(integer = 2,fraction = 0, message = "La edad no puede tener decimales ni más de 2 dígitos")
    private Integer edad;
    @Size(min = 3, message = "El ciclo debe tener almenos 3 carácteres")
    private String ciclo;
    @NotNull(message = "El curso no puede estar vacio")
    @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato incorrecto")
    @Range(min = 1, max = 2, message = "El curso solo admite los valores 1 o 2")
    private Integer curso;
    private boolean erasmus=false;
    private String lenguajeFavorito="";
    private String genero;
    private String horario;
    private String pais;
    private String hobbies;
}
```

UD 3: Bases de datos y servicios REST
5.- Consultas 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 “Run As \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 UD3_practica5_nombreAlumno.tar.gz al fichero comprimido 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.