



# 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 - Jose Ramon.profesor@gmail.com



Utiliza como base tu primer proyecto RESTful “dwes\_primer\_rest” :

```
▼ DWES_PRIMER_REST
  > .mvn
  > .settings
  > .vscode
  ▼ src
    > .vscode
    ▼ main
      ▼ java/edu/profesor/joseramon/dwes_primer_rest
        ▼ controller
          ● AlumnoRestController.java
        ▼ exception
          ● ResourceNotFoundException.java
        ▼ model
          ▼ db
            ● AlumnoDb.java
          ▼ dto
            ● AlumnoEdit.java
            ● AlumnoInfo.java
            ● AlumnoList.java
        ▼ repository
          ● AlumnoRepository.java
        ▼ service
          ▼ impl
            ● AlumnoServiceImpl.java
          > mapper
            ● AlumnoService.java
          ● DwesPrimerRestApplication.java
          ● ServletInitializer.java
        ▼ resources
          > static
          > templates
          ≡ application.properties
          ● data.sql
        > test
      > target
      ● .classpath
      ≡ .factorypath
      ● .gitignore
      ≡ .project
      ● HELP.md
      ● mvnw
      ● mvnw.cmd
      ● pom.xml
      ≡ test_Rest.http
```



# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

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



### 1º @Query:

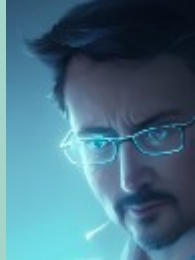
Esta anotación permiten realizar consultas directamente sobre la BD, tanto con JPQL como con sentencias SQL nativas. Realiza el primer cambio sobre `AlumnoRepository.java` :

```
AlumnoRepository.java X
src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > La
1  package edu.profesor.joseramon.dwes_primer_rest.repository;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4  import org.springframework.stereotype.Repository;
5  import edu.profesor.joseramon.dwes_primer_rest.model.db.AlumnoDb;
6  import java.util.Collection;
7  import org.springframework.data.jpa.repository.Query;
8
9  @Repository
10 public interface AlumnoRepository extends JpaRepository<AlumnoDb,String>{
11     //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
12     //Utilizamos el nombre de la clase 'AlumnoDb'
13     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
14     Collection<AlumnoDb> findAllAlumnoDbDAW();
15     //Podemos utilizar SQL nativo (native SQL)
16     //Utilizamos el nombre de la tabla 'alumnos' en la BD
17     @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
18     nativeQuery = true)
19     Collection<AlumnoDb> findAllAlumnoDbDAM();
20 }
```



¿ 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 x
src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java >
1 package edu.profesor.joseramon.dwes_primer_rest.repository;
2
3 > import org.springframework.data.jpa.repository.Query; ...
8
9 @Repository
10 public interface AlumnoRepository extends JpaRepository<AlumnoDb,String>{
11
12     //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
13     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
14     Collection<AlumnoDb> findAllAlumnoDbDAW();
15
16     //Podemos utilizar SQL nativo (native SQL)
17     @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
18     nativeQuery = true)
19     Collection<AlumnoDb> findAllAlumnoDbDAM();
20 }
```

```
AlumnoService.java x
src > main > java > edu > profesor > joseramon > dwes_primer_rest > service > AlumnoService
1 package edu.profesor.joseramon.dwes_primer_rest.service;
2
3 > import java.util.List; ...
9
10 public interface AlumnoService {
11     public Optional<AlumnoEdit> getAlumnoEditByDni(String dni);
12     public Optional<AlumnoInfo> getAlumnoInfoByDni(String dni);
13     public AlumnoEdit save(AlumnoEdit alumnoEdit);
14     public String deleteByDni(String dni);
15     public Optional<AlumnoEdit> update(AlumnoEdit alumnoEdit);
16     public List<AlumnoList> findAllAlumnoList();
17     public List<AlumnoList> findAllAlumnosListDAW();
18     public List<AlumnoList> findAllAlumnosListDAM();
19 }
```

```
AlumnoServiceImpl.java x
src > main > java > edu > profesor > joseramon > dwes_primer_rest > service > impl > AlumnoServiceImpl.java > Language Support for Java(TM) by Red Hat > AlumnoS
77 }
78
79 @Override
80 public List<AlumnoList> findAllAlumnosListDAW() {
81     return AlumnoMapper.INSTANCE.alumnosToAlumnosList((List<AlumnoDb>) alumnoRepository.findAllAlumnoDbDAW());
82 }
83
84 @Override
85 public List<AlumnoList> findAllAlumnosListDAM() {
86     return AlumnoMapper.INSTANCE.alumnosToAlumnosList((List<AlumnoDb>) alumnoRepository.findAllAlumnoDbDAM());
87 }
88
89 }
```



... 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 > jose.ramon > dwes_primer_rest > controller >
32
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.findAllAlumnosListDAW();
43 }
44
45 @GetMapping("/alumnosDAM")
46 public List<AlumnoList> getAlumnosListDAM() {
47     return alumnoService.findAllAlumnosListDAM();
48 }
```

```
test_Rest.http
test_Rest.http > GET /api/v1/alumnosDAM
1 ## getAlumnosList
2 Send Request
3 GET http://localhost:8080/api/v1/alumnos HTTP/1.1
4 Content-Type: application/json
5 ###
6 ## getAlumnosListDAW
7 Send Request
8 GET http://localhost:8080/api/v1/alumnosDAW HTTP/1.1
9 Content-Type: application/json
10 ###
11 ## getAlumnosListDAM
12 Send Request
13 GET http://localhost:8080/api/v1/alumnosDAM HTTP/1.1
14 Content-Type: application/json
15 ###
16 ## newAlumnoEdit
17 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  }
]
```



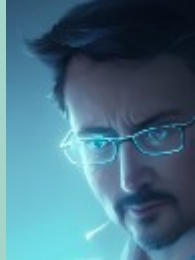
Si eres un alumno curioso tu siguiente pregunta podría ser:  
¿Como podemos parametrizar estas consultas para devolver los datos en función de dichos parámetros? Por ejemplo indicarle el ciclo y que devuelva todos los alumnos de dicho ciclo ...



# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

Desarrollo Web en Entorno Servidor - Jose Ramon.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_primer_rest > repository > AlumnoRepository.java > Language Support fo
1  import org.springframework.data.repository.CrudRepository;
2  import edu.profesor.joseramon.dwes_primer_rest.model.AlumnoDb;
3
4  import org.springframework.stereotype.Repository;
5  import edu.profesor.joseramon.dwes_primer_rest.model.AlumnoDb;
6  import java.util.Collection;
7  import org.springframework.data.jpa.repository.Query;
8  import org.springframework.data.repository.query.Param;
9
10 @Repository
11 public interface AlumnoRepository
12     //Por defecto se utiliza Jakarta
13     //Utilizamos el nombre de la clase
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.findAllAlumnosListCiclo(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.findAllAlumnosListCicloPais(ciclo,pais);
101 }

test_Rest.http
test_Rest.http > ...
11 Content-Type: application/json
12 ###
13 ## getAlumnosListCiclo
14 Send Request
15 GET http://localhost:8080/api/v1/alumnos/ciclo/DAW HTTP/1.1
16 Content-Type: application/json
17 ###
18 ## getAlumnosListCicloPais
19 Send Request
20 GET http://localhost:8080/api/v1/alumnos/ciclo/DAW/pais/ES HTTP/1.1
21 Content-Type: application/json
22 ###
23 ## newAlumnoEdit

```



# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

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



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

19 @NoArgsConstructor
20 @AllArgsConstructor
21 @Data
22 @Entity
23 @Table(name = "alumnos")
24 public class AlumnoDb implements Serializable{
25     private static final long serialVersionUID = -818542778373595260L;
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 caracteres")
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 }
```





# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

Desarrollo Web en Entorno Servidor - Jose Ramon - jose.ramon.profesor@gmail.com



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

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

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

19 @NoArgsConstructor
20 @AllArgsConstructor
21 @Data
22 @Entity
23 @Table(name = "alumnos")
24 public class AlumnoDb implements Serializable{
25     private static final long serialVersionUID = -818542778373595260L;
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 caracteres")
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 }
```

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

```
AlumnoRepository.java 3
src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > Language Support

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 automáticos en Spring Data JPA
33     f
34     findByCiclo(String ciclo);
35     findByCurso(Integer curso);
36     findByDni(String dni);
37     findByEdad(Integer edad);
38     findByGenero(String genero);
39     findByHobbies(String hobbies);
40     findByHorario(String horario);
41     findByLenguajeFavorito(String lenguajeFavori...
42     findByNombre(String nombre);
43     findByPais(String pais);
44     findAll() : List<AlumnoDb>
45     findAll(Example<S> example) : List<S>
```



# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

Desarrollo Web en Entorno Servidor - Jose Ramon.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 X
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 automáticos 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 - Jose Ramon.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 x
src > main > java > edu > profesor > joseramon > dwes_primer_rest > controller > AlumnoRestController.java > ...
105 //Métodos Spring Data JPA
106 @GetMapping("/alumnosList/dni/{dni}")
107 public List<AlumnoList> getAlumnosListByDni(@PathVariable("dni") String dni) {
108     return alumnoService.findAlumnosListByDni(dni);
109 }
110
111 @GetMapping("/alumnosList/ciclo/{ciclo}")
112 public List<AlumnoList> getAlumnosListByCiclo(@PathVariable("ciclo") String ciclo) {
113     return alumnoService.findAlumnosListByCiclo(ciclo);
114 }
115
116 @GetMapping("/alumnosList/horario/{horario}")
117 public List<AlumnoList> getAlumnosListByHorario(@PathVariable("horario") String horario) {
118     return alumnoService.findAlumnosListByHorario(horario);
119 }
120
121 @GetMapping("/alumnosList/edad/{edad}")
122 public List<AlumnoList> getAlumnosListByEdad(@PathVariable("edad") Integer edad) {
123     return alumnoService.findAlumnosListByEdad(edad);
124 }
125 }
```

```
test_Rest.http x
test_Rest.http > ...
115 ###
116 ## findAlumnosListByDni
117 Send Request
118 GET http://localhost:8080/api/v1/alumnosList/dni/11111111A HTTP/1.1
119 Content-Type: application/json
120 ###
121 ## getAlumnosListByCiclo
122 Send Request
123 GET http://localhost:8080/api/v1/alumnosList/ciclo/DAW HTTP/1.1
124 Content-Type: application/json
125 ###
126 ## getAlumnosListByHorario
127 Send Request
128 GET http://localhost:8080/api/v1/alumnosList/horario/T HTTP/1.1
129 Content-Type: application/json
130 ###
131 ## getAlumnosListByEdad
132 Send Request
133 GET http://localhost:8080/api/v1/alumnosList/edad/21 HTTP/1.1
134 Content-Type: application/json
135 ###
```





# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

Desarrollo Web en Entorno Servidor - Jose Ramon.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 x
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?



# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

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



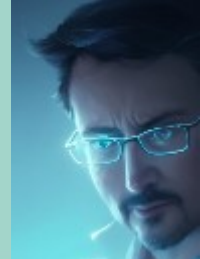
### ... continuación 3º Ordenación en JpaRepository :

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

9  import org.springframework.data.domain.Sort;
10
11  @Repository
12  public interface AlumnoRepository extends JpaRepository<AlumnoDb,String>{
13      //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
14      //Utilizamos el nombre de la clase 'AlumnoDb'
15      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
16      Collection<AlumnoDb> findAllAlumnoDbDAW();
17      //Podemos utilizar SQL nativo (native SQL)
18      //Utilizamos el nombre de la tabla 'alumnos' en la BD
19      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
20            nativeQuery = true)
21      Collection<AlumnoDb> findAllAlumnoDbDAM();
22
23      //CONSULTAS CON PARAMETROS:
24      //JPQL: AlumnoDb
25      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
26      Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
27      //native SQL: alumnos
28      @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
29            nativeQuery = true)
30      Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
31            @Param("pais") String pais);
32
33      //Métodos automáticos en Spring Data JPA
34      Collection<AlumnoDb> findByDni(String dni);
35      Collection<AlumnoDb> findByCiclo(String ciclo);
36      Collection<AlumnoDb> findByHorario(String horario);
37      Collection<AlumnoDb> findByEdad(Integer edad);
38
39      //Ordenación de resultados
40      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo ORDER BY a.curso ASC")
41      Collection<AlumnoDb> findAllAlumnoDbCicloOrderByCurso(@Param("ciclo") String ciclo);
42      //Consultas con varias ordenaciones en Spring Data JPA
43      @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
44      Collection<AlumnoDb> findByCicloOrderBy(String ciclo,Sort sort);
45  }
```





... 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

```

127
128 //Ordenación de resultados
129 @GetMapping("/alumnos/ciclo/{ciclo}/OrderByCurso")
130 public List<AlumnoList> getAlumnosListCicloOrderByCurso(@PathVariable("ciclo") String ciclo)
131     return alumnoService.findAllAlumnosListCicloOrderByCurso(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

```

136 ## getAlumnosListByCicloOrderBy
137 Send Request
138 GET http://localhost:8080/api/v1/alumnosList/ciclo/DAW/OrderBy/edad/DESC HTTP/1.1
139 Content-Type: application/json
140 ###

```

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": "333333333C",
10    "nombre": "Pedro Martinez",
11    "edad": 43,
12    "ciclo": "DAW",
13    "curso": 2,
14    "erasmus": false,
15    "modificado": null,
16    "erasmusChecked": ""
17  },
18  √ {
19    "dni": "222222222B",
20    "nombre": "Maria Gonzalez",
21    "edad": 32,
22    "ciclo": "DAW",
23    "curso": 2,
24    "erasmus": true,
25    "modificado": null,
26    "erasmusChecked": "checked"
27  }
28 ]

```

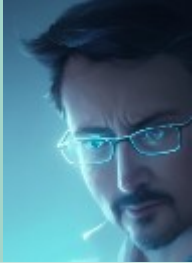




# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

Desarrollo Web en Entorno Servidor - Jose Ramon.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 X
src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > AlumnoRepository.java > AlumnoRepository

9  import org.springframework.data.jpa.repository.Query;
10 import org.springframework.data.repository.query.Param;
11 import org.springframework.data.domain.Sort;
12
13 @Repository
14 public interface AlumnoRepository extends JpaRepository<AlumnoDb,String>{
15     //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
16     //Utilizamos el nombre de la clase 'AlumnoDb'
17     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
18     Collection<AlumnoDb> findAllAlumnoDbDAW();
19     //Podemos utilizar SQL nativo (native SQL)
20     //Utilizamos el nombre de la tabla 'alumnos' en la BD
21     @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
22           nativeQuery = true)
23     Collection<AlumnoDb> findAllAlumnoDbDAM();
24
25     //CONSULTAS CON PARÁMETROS:
26     //JPQL: AlumnoDb
27     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
28     Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
29     //native SQL: alumnos
30     @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
31           nativeQuery = true)
32     Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
33           @Param("pais") String pais);
34
35     //Métodos automáticos en Spring Data JPA
36     Collection<AlumnoDb> findByDni(String dni);
37     Collection<AlumnoDb> findByCiclo(String ciclo);
38     Collection<AlumnoDb> findByHorario(String horario);
39     Collection<AlumnoDb> findByEdad(Integer edad);
40
41     //Ordenación de resultados
42     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo ORDER BY a.curso ASC")
43     Collection<AlumnoDb> findAllAlumnoDbCicloOrderByCurso(@Param("ciclo") String ciclo);
44     //Consultas con varias ordenaciones en Spring Data JPA
45     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
46     Collection<AlumnoDb> findByCicloOrderBy(String ciclo,Sort sort);
47     @Query("SELECT a FROM AlumnoDb a WHERE a.edad=:edad")
48     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 - [Joseamon.profesor@gmail.com](mailto:Joseamon.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 X
src > main > java > edu > profesor > joseramon > dwes_primer_rest > repository > J AlumnoRepository.java > Language Support for Java

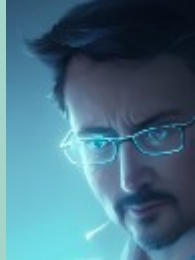
1 package edu.profesor.joseramon.dwes_primer_rest.repository;
2
3 > import org.springframework.data.jpa.repository.JpaRepository; -
4
5
6
7
8
9
10 public interface AlumnoRepository extends JpaRepository<AlumnoDb,String>{
11     //Por defecto se utiliza Jakarta Persistence Query Language (JPQL)
12     //Utilizamos el nombre de la clase 'AlumnoDb'
13     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo='DAW'")
14     Collection<AlumnoDb> findAllAlumnoDbDAW();
15     //Podemos utilizar SQL nativo (native SQL)
16     //Utilizamos el nombre de la tabla 'alumnos' en la BD
17     @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo='DAM'",
18         nativeQuery = true)
19     Collection<AlumnoDb> findAllAlumnoDbDAM();
20
21     //CONSULTAS CON PARÁMETROS:
22     //JPQL: AlumnoDb
23     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
24     Collection<AlumnoDb> findAllAlumnoDbCiclo(@Param("ciclo") String ciclo);
25     //native SQL: alumnos
26     @Query(value="SELECT * FROM alumnos as a WHERE a.ciclo=:ciclo AND a.pais=:pais",
27         nativeQuery = true)
28     Collection<AlumnoDb> findAllAlumnoDbCicloPais(@Param("ciclo") String ciclo,
29         @Param("pais") String pais);
30
31     //Métodos automáticos en Spring Data JPA
32     Collection<AlumnoDb> findByDni(String dni);
33     Collection<AlumnoDb> findByCiclo(String ciclo);
34     Collection<AlumnoDb> findByHorario(String horario);
35     Collection<AlumnoDb> findByEdad(Integer edad);
36
37     //Ordenación de resultados
38     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo ORDER BY a.curso ASC")
39     Collection<AlumnoDb> findAllAlumnoDbCicloOrderByCurso(@Param("ciclo") String ciclo);
40     //Consultas con varias ordenaciones en Spring Data JPA
41     @Query("SELECT a FROM AlumnoDb a WHERE a.ciclo=:ciclo")
42     Collection<AlumnoDb> findByCicloOrderBy(String ciclo,Sort sort);
43     @Query("SELECT a FROM AlumnoDb a WHERE a.edad=:edad")
44     Collection<AlumnoDb> findByEdadOrderBy(Integer edad,Sort sort);
45
46     //Métodos automáticos en Spring Data JPA
47     //OLD: versión 2.6.2: List<AlumnoDb> findByCiclo(String ciclo,Sort sort);
48     Collection<AlumnoDb> findByCiclo(String ciclo,Sort sort);
49     Collection<AlumnoDb> findByHorario(String horario,Sort sort);
50     Collection<AlumnoDb> findByEdad(Integer edad,Sort sort);
51 }
```



# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

Desarrollo Web en Entorno Servidor - Jose Ramon.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

```

127
128 //Ordenación de resultados
129 @GetMapping("/alumnos/ciclo/{ciclo}/OrderByCurso")
130 public List<AlumnoList> getAlumnosListCicloOrderByCurso(@PathVariable String ciclo) {
131     return alumnoService.findAllAlumnosListCicloOrderByCurso(ciclo);
132 }
133 //Métodos con Ordenación SQL 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 }
140 @GetMapping("/alumnosList/edad/{edad}/OrderBy/{atributoOrden}/{direccion}")
141 public List<AlumnoList> getAlumnosListByEdadOrderBy(@PathVariable("edad") String edad,
142                                                         @PathVariable("atributoOrden") String atributoOrden,
143                                                         @PathVariable("direccion") String direccion) {
144     return alumnoService.findByEdadOrderBy(edad, Sort.by(Direction.fromString(direccion), atributoOrden));
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

```

133 GET http://localhost:8080/api/v1/alumnos/ciclo/DAW/OrderByCurso HTTP/1.1
134 Content-Type: application/json
135 ###
136 ## getAlumnosListByCicloOrderBy
137 Send Request
138 GET http://localhost:8080/api/v1/alumnosList/ciclo/DAW/OrderBy/edad/DESC HTTP/1.1
139 Content-Type: application/json
140 ###
141 ## getAlumnosListByEdadOrderBy
142 Send Request
143 GET http://localhost:8080/api/v1/alumnosList/edad/43/OrderBy/dni/ASC HTTP/1.1
144 Content-Type: application/json
145 ###
146 ## getAlumnosByCicloOrderBy
147 Send Request
148 GET http://localhost:8080/api/v1/alumnos/ciclo/DAW/OrderBy/edad/DESC HTTP/1.1
149 Content-Type: application/json
150 ###

```





### 4º Documentar la API Rest con OpenAPI y Swagger:

Actualmente muchas aplicaciones webs suelen dividirse en **2 proyectos distintos**, el proyecto que contiene el **front-end** y el proyecto 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/>

The screenshot shows the PayPal Developer API documentation page for Authentication. The left sidebar lists various REST APIs, with 'Authentication' selected. The main content area is titled 'Authentication' and includes a 'CURRENT' status badge. It explains that PayPal REST APIs use OAuth 2.0 access tokens. Below this, it provides instructions on how to get an access token using cURL or Postman. A cURL example is shown in a code block:

```
1 curl -v -X POST "https://api-m.sandbox.paypal.com/v1/oauth2/token" \
2 -u "<CLIENT_ID>:<CLIENT_SECRET>" \
3 -H "Content-Type: application/x-www-form-urlencoded" \
4 -d "grant_type=client_credentials"
5
```

At the bottom, there is a note: '1. Change CLIENT\_ID to your client ID.'

Spotify:

<https://developer.spotify.com/documentation/web-api/>

The screenshot shows the Spotify Developer Web API documentation page. The top navigation bar includes links for DISCOVER, DOCS, CONSOLE, COMMUNITY, DASHBOARD, and USE CASES. The 'DOCS' section is active, and the 'WEB API' sub-section is selected. The main heading is 'Web API'. Below this, there is a note about accepting the Spotify Developer Terms of Service. A diagram illustrates the interaction between 'MY APP' (represented by a blue box with a star icon) and 'Spotify' (represented by a green box with the Spotify logo). A double-headed arrow labeled 'Web API' connects the two boxes, indicating that the app uses the Spotify Web API to interact with the service.



## 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:

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 librerías 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.



### ... 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:

alumno-rest-controller Alumno Rest Controller	
GET	/api/v1/alumnos getAlumnosList
POST	/api/v1/alumnos newAlumnoEdit
GET	/api/v1/alumnos/{dni} getAlumnoEditByDni
PUT	/api/v1/alumnos/{dni} updateAlumnoEdit
DELETE	/api/v1/alumnos/{dni} deleteByDni
GET	/api/v1/alumnos/{dni}/info getAlumnoInfoByDni
GET	/api/v1/alumnos/ciclo/{ciclo} getAlumnosListCiclo
GET	/api/v1/alumnos/ciclo/{ciclo}/OrderBy/{atributoOrden} /{direccion} getAlumnosByCicloOrder By
GET	/api/v1/alumnos/ciclo/{ciclo}/OrderByCurso getAlumnosListCicloOrderByCurso
GET	/api/v1/alumnos/ciclo/{ciclo}/pais/{pais} getAlumnosListCicloPais
GET	/api/v1/alumnosDAM getAlumnosListDAM
GET	/api/v1/alumnosDAW getAlumnosListDAW

GET /api/v1/alumnos getAlumnosList		
Parameters		
No parameters		
Responses		
Code	Description	Links
200	OK	No links
Media type: */* Controls Accept header. Example Value Schema		
<pre>{   "ciclo": "string",   "curso": 0,   "dni": "string",   "edad": 0,   "erasmus": true,   "erasmusChecked": "string",   "modificado": "string",   "nombre": "string" }</pre>		
401	Unauthorized	No links

Schemas	
<b>AlumnoEdit</b> ▾ { ciclo curso* dni edad* erasmus genero hobbies horario lenguajeFavorito nombre pais	string maxLength: 2147483647 minLength: 3 integer(\$int32) string pattern: [0-9]{8}[A-Za-z]{1} integer(\$int32) boolean string string string string string string maxLength: 2147483647 minLength: 5 string
AlumnoInfo >	





# UD 3: Bases de datos y servicios REST

## 5.- Consultas en Spring Data JPA

Desarrollo Web en Entorno Servidor - Jose Ramon.profesor@gmail.com

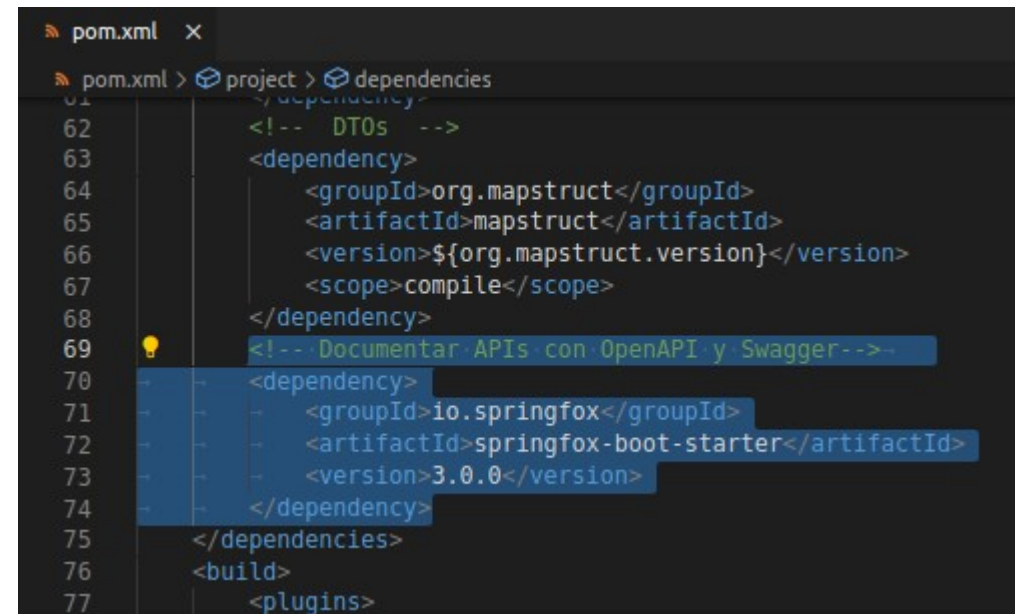


... continuación 4º Documentar la API Rest con OpenAPI y Swagger:

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>
```





# 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:

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

#Configuración OpenAPI y Swagger

spring.mvc.pathmatch.matching-strategy=ant-path-matcher

```
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
```





### ... 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:

The screenshot displays the Swagger UI interface for the 'alumno-rest-controller' API. The interface is organized into several sections:

- Header:** Includes the Swagger logo, 'Select a definition' dropdown (set to 'default'), and 'Api Documentation' with version '1.0' and 'OAS3'.
- Servers:** A dropdown menu showing 'http://localhost:8080 - Inferred Url'.
- Controllers:** A list of controllers including 'alumno-rest-controller' (Alumno Rest Controller), 'basic-error-controller' (Basic Error Controller), and 'Schemas'.
- API Endpoints:** A list of endpoints for the 'alumno-rest-controller':
  - GET /api/v1/alumnos getAlumnosList
  - POST /api/v1/alumnos newAlumnoEdit
  - GET /api/v1/alumnos/{dni} getAlumnoEditByDni
  - PUT /api/v1/alumnos/{dni} updateAlumnoEdit
  - DELETE /api/v1/alumnos/{dni} deleteByDni
  - GET /api/v1/alumnos/{dni}/info getAlumnoInfoByDni
  - GET /api/v1/alumnos/ciclo/{ciclo} getAlumnosListCiclo
  - GET /api/v1/alumnos/ciclo/{ciclo}/ordenBy/{atributoOrden} getAlumnosByCicloOrder By
  - GET /api/v1/alumnos/ciclo/{ciclo}/ordenBy/{atributoOrden} getAlumnosByCicloOrder By
  - GET /api/v1/alumnos/ciclo/{ciclo}/ordenBy/{atributoOrden} getAlumnosByCicloOrder By
  - GET /api/v1/alumnos/ciclo/{ciclo}/ordenBy/{atributoOrden} getAlumnosByCicloOrder By
  - GET /api/v1/alumnos/ciclo/{ciclo}/ordenBy/{atributoOrden} getAlumnosByCicloOrder By
- Response Detail:** A detailed view of the GET /api/v1/alumnos getAlumnosList endpoint, showing:
  - Parameters:** No parameters.
  - Responses:**
    - 200 OK: Media type 'application/json', Controls 'Accept header', Example Value: { "ciclo": "string", "curso": 0, "dni": "string", "edad": 0, "erasmus": true, "erasmusChecked": "string", "modificado": "string", "nombre": "string" }.
    - 401 Unauthorized: No links.
- Schemas:** A section showing the JSON Schema for 'AlumnoEdit':
 

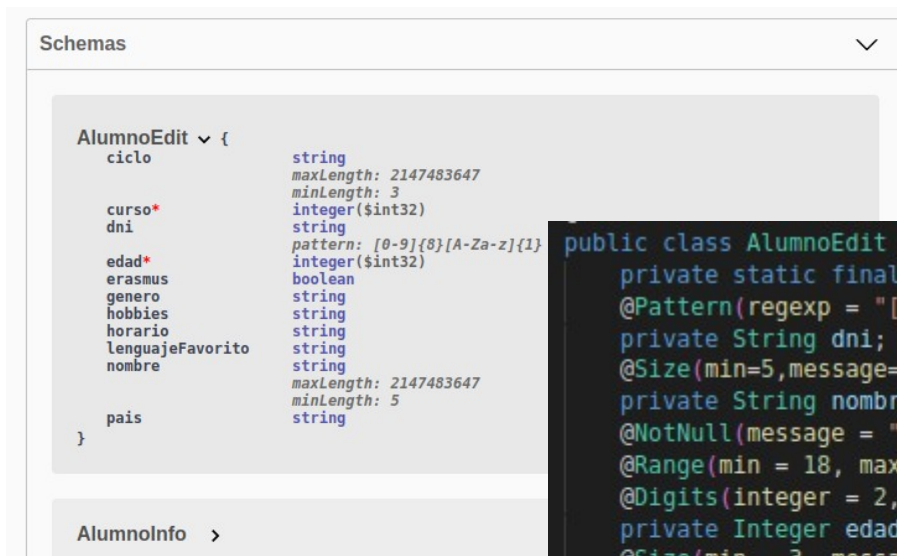
```

AlumnoEdit {
  ciclo string
    maxLength: 2147483647
    minLength: 3
  curso* integer($int32)
  dni string
    pattern: [0-9]{8}[A-Za-z]{1}
  edad* integer($int32)
  erasmus boolean
  genero string
  hobbies string
  horario string
  lenguajeFavorito string
  nombre string
    maxLength: 2147483647
    minLength: 5
  pais string
}
      
```



## ... 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:



```
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 caracteres")
    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 caracteres")
    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.