
M6.UF4.A6.P3
CRUD SPRING DE JDBC A
PERSISTENCIA

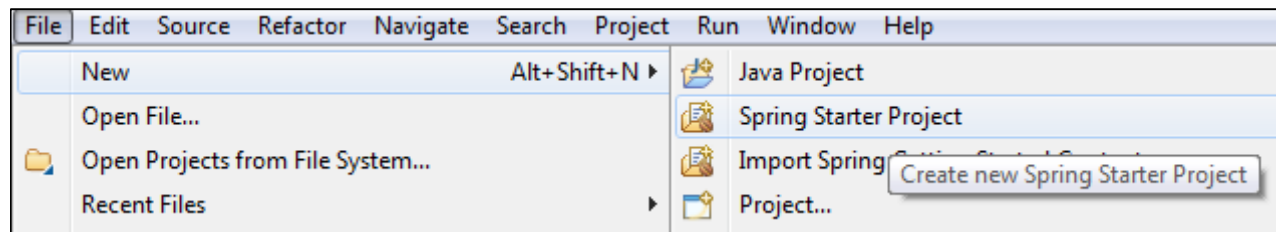
Eduard Lara

INDICE

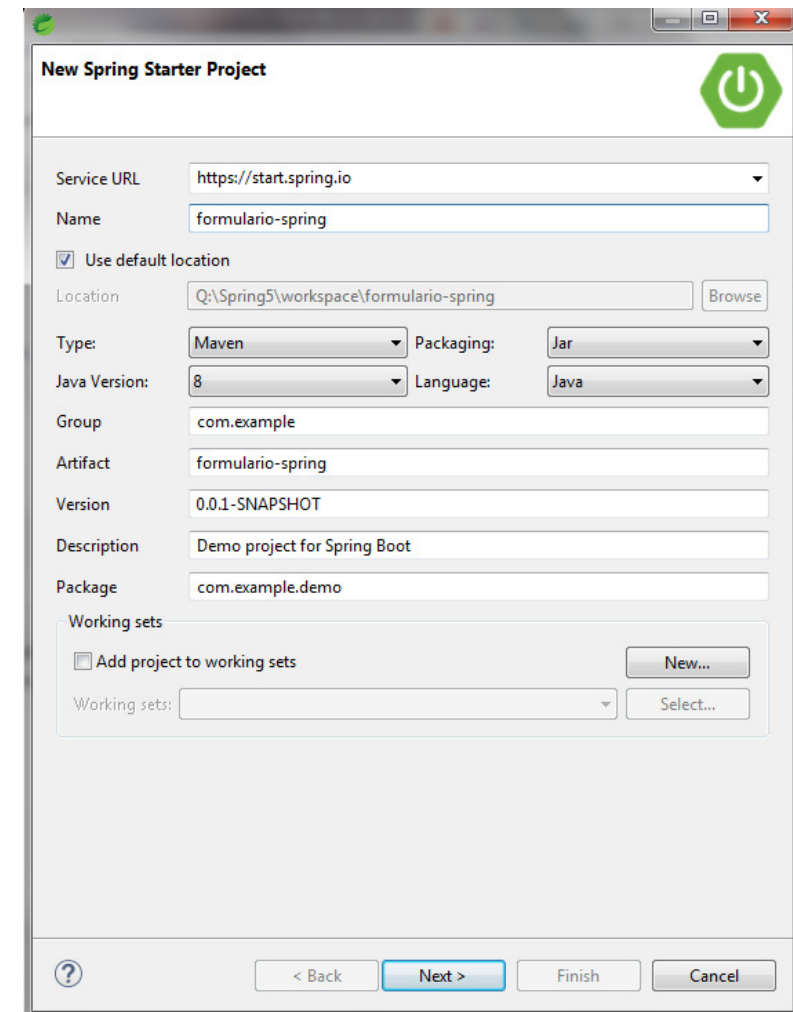
1. Construcción CRUD
2. Backend con ArrayList
3. Backend con JDBC
4. Backend con JPA Persistencia

1. CONSTRUCCION CRUD

Paso 1) Creamos un proyecto Spring Boot, en la opción de menu File/New/Spring Starter Project:



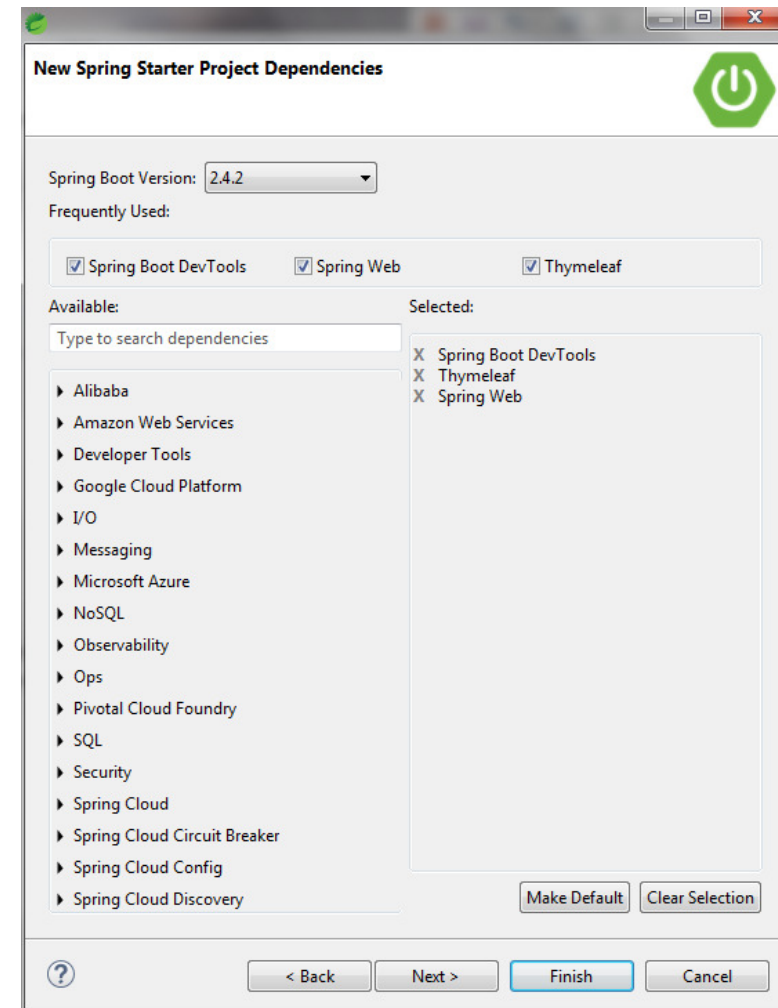
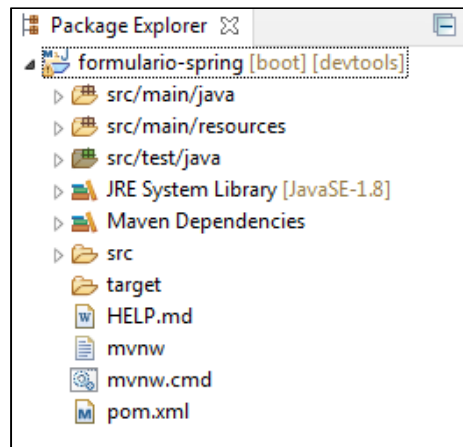
Podemos dejar por defecto los valores que nos presenta el wizard. Si se desea se puede cambiar el nombre de proyecto, el package raíz, el tipo de proyecto (Maven o Gradle) y/o la versión de Java.



1. CONSTRUCCION CRUD

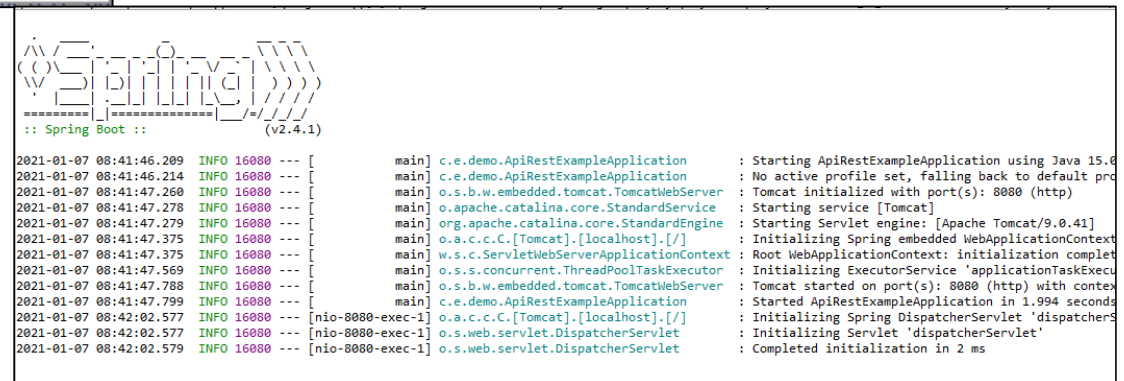
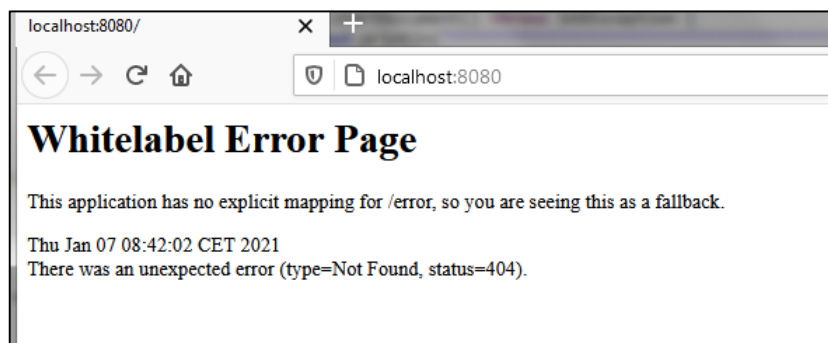
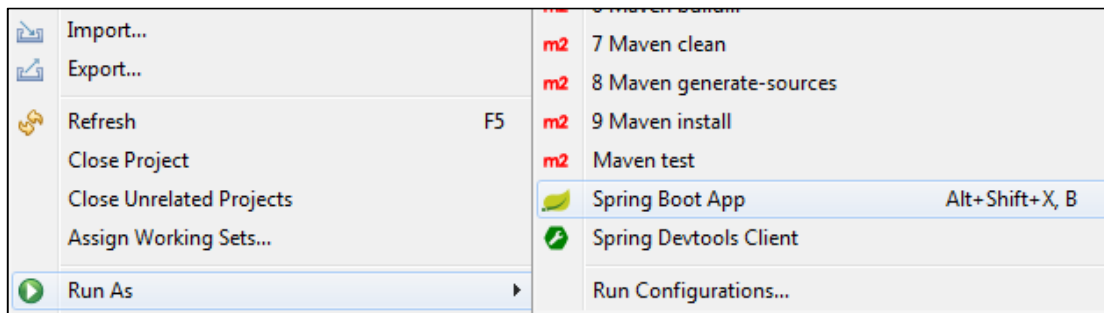
Paso 2) Agregamos las siguientes dependencias:

- Spring Web (necesaria)
- Spring Boot Dev Tools (muy importante ya que cualquier cambio que hagamos en nuestro código java, de forma automática se va a actualizar en el despliegue sin tener que reiniciar el servidor)
- Thymeleaf, para hacer uso de estas plantillas, que hacen el papel de las típicas jsp



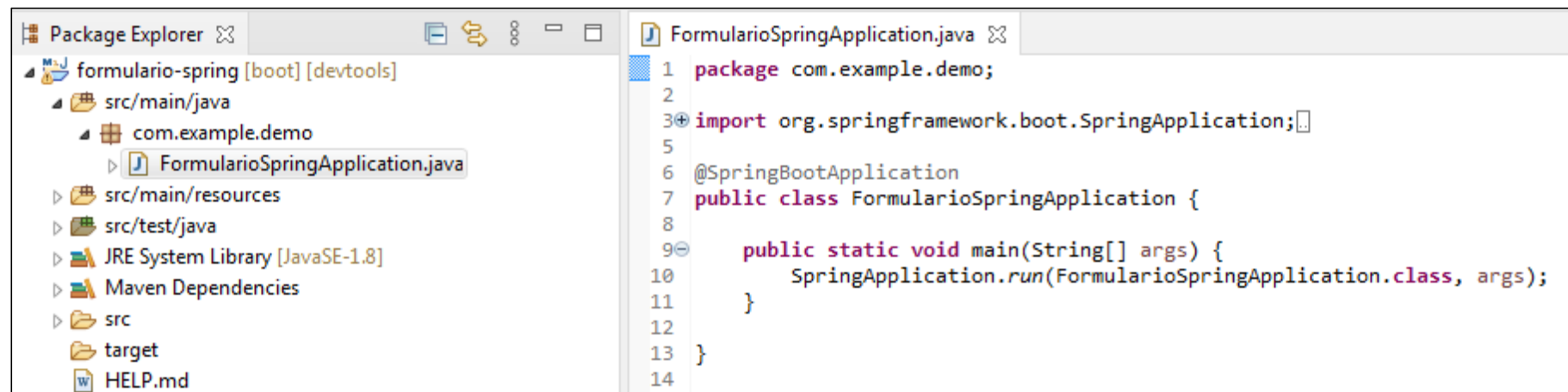
1. CONSTRUCCION CRUD

Paso 3) Probamos de ejecutar el proyecto, para ello levantamos el servidor Tomcat haciendo Run As/Spring Boot App. Una vez vemos que ha arrancado correctamente el servidor, vamos a un navegador y ponemos **localhost:8080**. Nos da error porque no tenemos ninguna página de inicio. Pero también significa que ya hay un servidor respondiendo en el puerto 8080.



1. CONSTRUCCION CRUD

Paso 4) Podemos observar en el package raíz indicado al principio en la creación del proyecto, la clase generada automáticamente que inicia nuestro servidor y la aplicación:



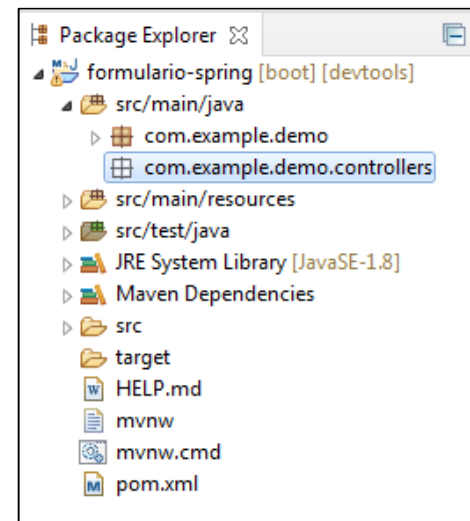
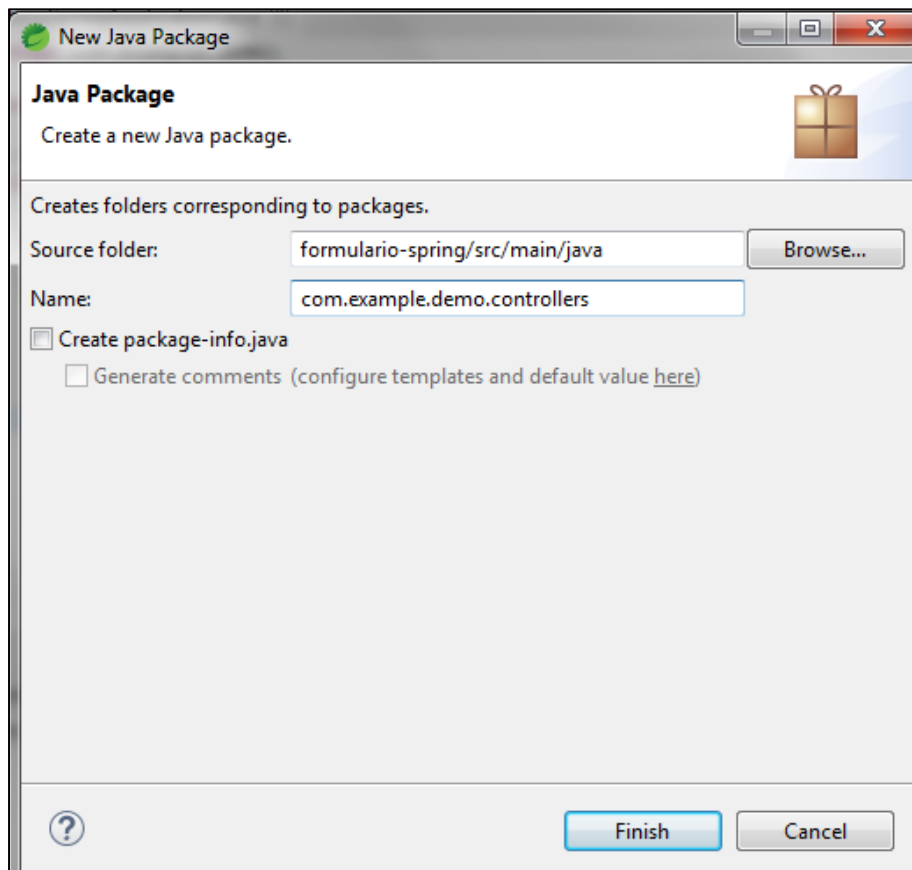
The screenshot shows an IDE window with two panes. The left pane, titled 'Package Explorer', displays a project structure for 'formulario-spring [boot] [devtools]'. It includes a 'src/main/java' directory containing a 'com.example.demo' package, which in turn contains the 'FormularioSpringApplication.java' file. Other visible elements include 'src/main/resources', 'src/test/java', 'JRE System Library [JavaSE-1.8]', 'Maven Dependencies', 'src', 'target', and 'HELP.md'. The right pane shows the source code of 'FormularioSpringApplication.java'. The code is as follows:

```
1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class FormularioSpringApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(FormularioSpringApplication.class, args);
11     }
12 }
13
14
```

1. CONSTRUCCION CRUD

Controlador

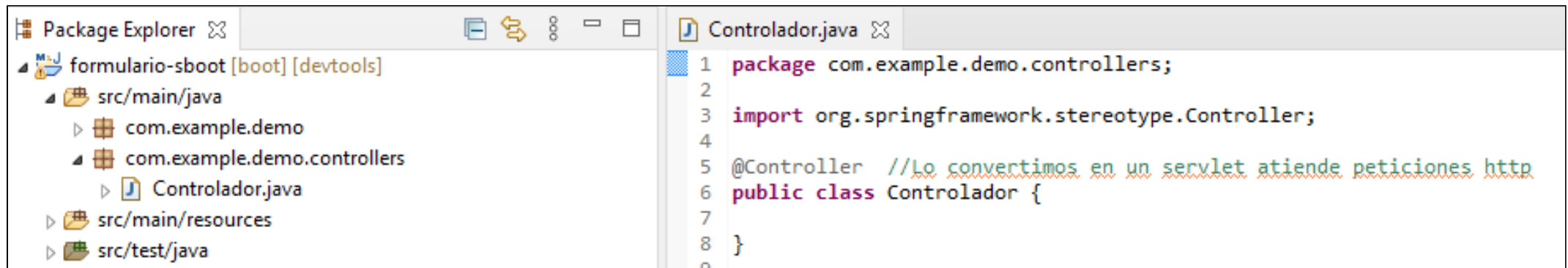
Paso 5) Generamos un package dentro del existente con la extensión controllers:



1. CONSTRUCCION CRUD

Controlador

Paso 6) Dentro de este package creamos una clase a la que le pondremos la etiqueta de controlador `@Controller`.



1. CONSTRUCCION CRUD

Controlador

Paso 7) Creamos dos métodos handler:

- iniciar → el cual nos dará entrada al formulario de login. Será accesible desde localhost:8080 con método Get.
- login → una vez logados correctamente nos dara entrada al listado de libros

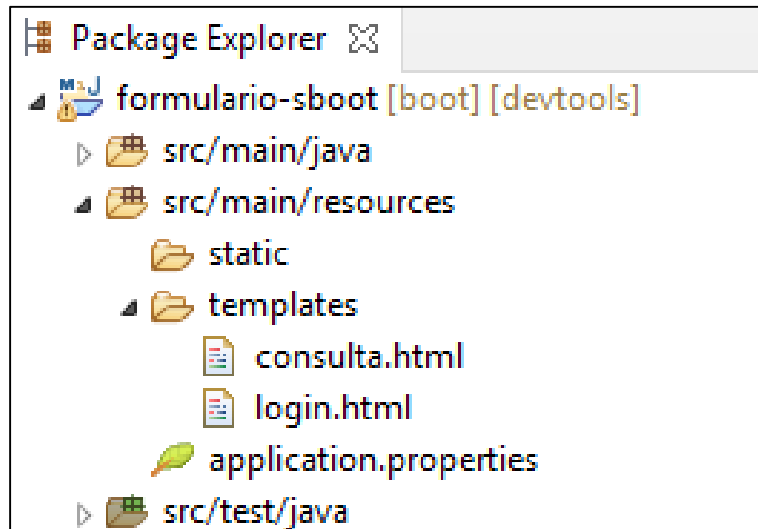
```
Controlador.java
1 package com.example.demo.controllers;
2
3 import org.springframework.stereotype.Controller;
4
5
6
7
8 @Controller //Lo convertimos en un servlet atiende peticiones http
9 @RequestMapping("")
10 public class Controlador {
11
12     @GetMapping("/")
13     public String iniciar() {
14         return "login";
15     }
16
17     @PostMapping("/")
18     public String login() {
19         return "consulta";
20     }
21 }
22
```

1. CONSTRUCCION CRUD

Vistas

Paso 8) En resources/templates vamos a crear dos plantillas:

- login.html → que mostrará el formulario de acceso.
- consulta.html → que mostrará el listado de libros y permitirá realizar un mantenimiento sobre esos datos



```
login.html
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="ISO-8859-1">
5  <title>Insert title here</title>
6  </head>
7  <body>
8
9  </body>
10 </html>
11
```

1. CONSTRUCCION CRUD

Vistas

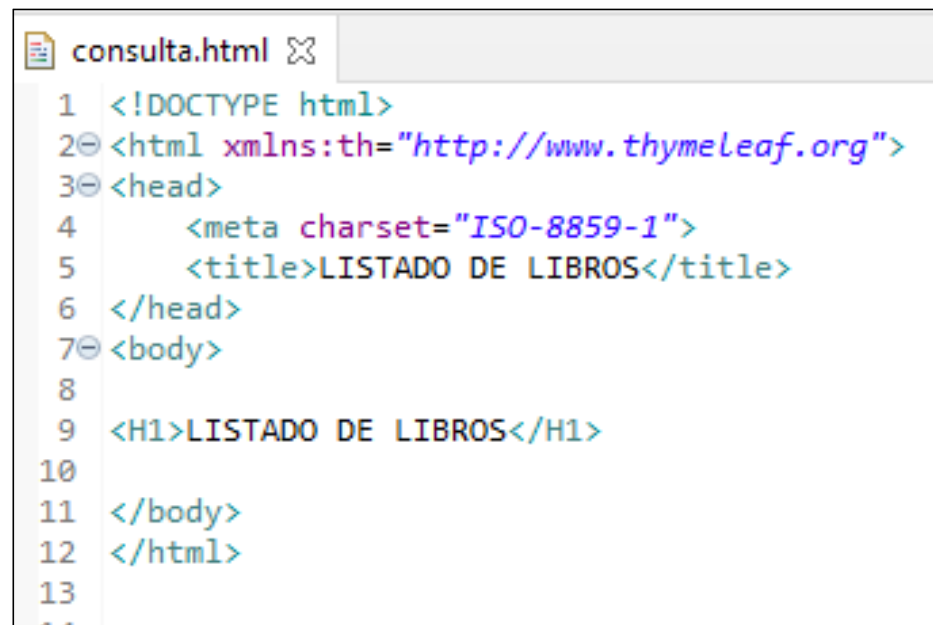
Paso 9) En login.html crea un formulario de acceso a la aplicación. Su action será "/", es decir localhost:8080/, y su método http de envío post:

```
login.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="UTF-8">
5     <title>FORMULARIO DE ACCESO</title>
6 </head>
7 <body>
8     <H1>FORMULARIO DE ACCESO</H1>
9     <form action="/" method="post">
10         Login: <input type="text" name="nombre"><br><br>
11         Password: <input type="password" name="password"><br><br>
12         <input type="submit" name="submit" value="Login">
13     </form>
14 </body>
15 </html>
16
17
```

1. CONSTRUCCION CRUD

Vistas

Paso 10) En consulta.html, mostraremos un inicial listado de libros (ya lo complementaremos posteriormente):

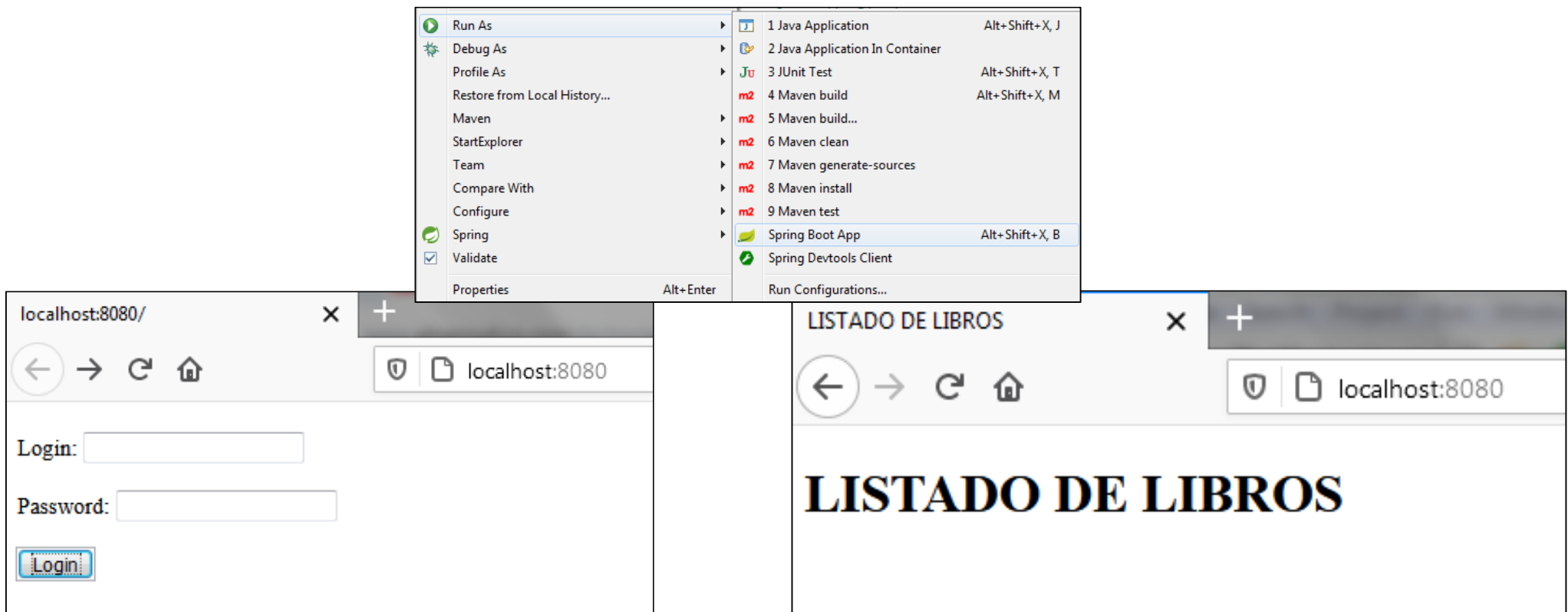


```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <meta charset="ISO-8859-1">
5     <title>LISTADO DE LIBROS</title>
6 </head>
7 <body>
8
9 <H1>LISTADO DE LIBROS</H1>
10
11 </body>
12 </html>
13
```

1. CONSTRUCCION CRUD

Vistas

Paso 11) En este punto podemos probar el funcionamiento de nuestra aplicación. Hacemos sobre el proyecto click botón derecho/Run As/Spring Boot App.



1. CONSTRUCCION CRUD

Vistas

Paso 12) Podemos convertir login.html en una plantilla thymeleaf para pasarle el parámetro titulo desde el handler iniciar del controlador:

```
login.html
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <meta charset="UTF-8">
5     <title th:text="${titulo}" />
6 </head>
7 <body>
8 <h1 th:text="${titulo}" />
9 <form action="/" method="post">
10     Login: <input type="text" name="nombre"><br><br>
11     Password: <input type="password" name="password"><br><br>
12     <input type="submit" name="submit" value="Login">
13 </form>
14 </body>
15 </html>
16
```

```
Controlador.java
1 package com.example.demo.controllers;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller //Lo convertimos en un servlet atiende peticiones http
6 @RequestMapping("/")
7 public class Controlador {
8
9     @GetMapping("/")
10     public String iniciar(Model model) {
11         model.addAttribute("titulo", "FORMULARIO DE ACCESO");
12         return "login";
13     }
14
15     @PostMapping("/")
16     public String login() {
17         return "consulta";
18     }
19 }
20
21
```

1. CONSTRUCCION CRUD

Vistas

Paso 13) El método handler “login” del controlador, puede recibir los datos enviados por el formulario de acceso uno a uno o empaquetados en una clase.

Método uno a uno

Debemos de insertar tantos parámetros con la etiqueta `@RequestParam`, como elementos html (básicamente input type) hayamos definido en el formulario.

Es muy importante que coincida el valor del atributo name del input type con el nombre de la variable usada para recuperar su valor en el controlador.

```
<form action="/" method="post">
    Login: <input type="text" name=nombre><br><br>
    Password: <input type="password" name=password><br><br>
    <input type="submit" name="submit" value="Login">
</form>
```

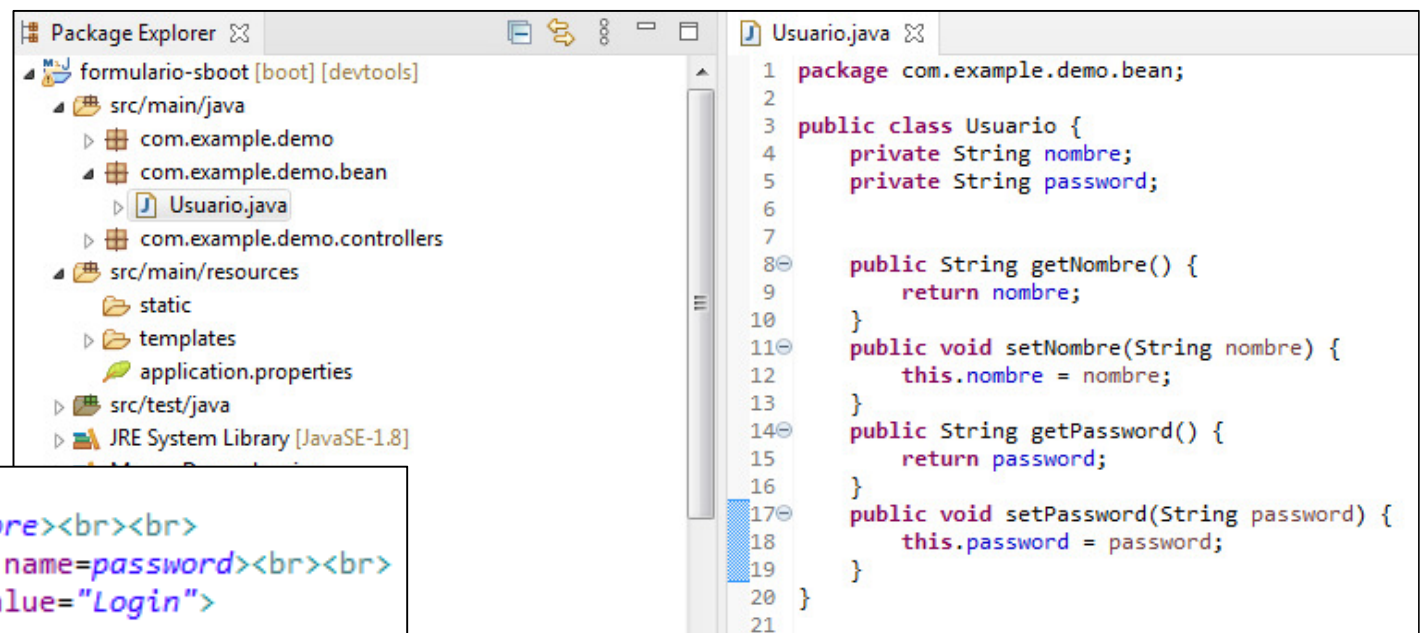
```
Controlador.java
3 import org.springframework.stereotype.Controller;
9
10 @Controller //Lo convertimos en un servlet atiende peticiones http
11 @RequestMapping("")
12 public class Controlador {
13
14     @GetMapping("/")
15     public String iniciar(Model model) {
16         model.addAttribute("titulo","FORMULARIO DE ACCESO");
17         return "login";
18     }
19
20     @PostMapping("/")
21     public String login(Model model,
22                         @RequestParam String nombre,
23                         @RequestParam String password) {
24         if (nombre.equals("edu") && password.equals("edu"))
25             return "consulta";
26         else
27             return "login";
28     }
29 }
30
```

1. CONSTRUCCION CRUD

Vistas

Paso 14) Para el método de todos los parámetros **empaquetados en una clase**, debemos definir una clase llamada Usuario, cuyos atributos deben de coincidir con el atributo name de los input text del formulario. La crearemos dentro de un package de nombre bean.

```
<form action="/" method="post">
  Login: <input type="text" name="nombre"><br><br>
  Password: <input type="password" name="password"><br><br>
  <input type="submit" name="submit" value="Login">
</form>
```



The screenshot shows an IDE with the Package Explorer on the left and the Usuario.java file open on the right. The Package Explorer shows the project structure: formulario-sboot [boot] [devtools] with sub-packages src/main/java, src/main/resources, and src/test/java. Under src/main/java, there are packages com.example.demo, com.example.demo.bean, and com.example.demo.controllers. The Usuario.java file is located in com.example.demo.bean. The code in Usuario.java is as follows:

```
1 package com.example.demo.bean;
2
3 public class Usuario {
4     private String nombre;
5     private String password;
6
7
8     public String getNombre() {
9         return nombre;
10    }
11    public void setNombre(String nombre) {
12        this.nombre = nombre;
13    }
14    public String getPassword() {
15        return password;
16    }
17    public void setPassword(String password) {
18        this.password = password;
19    }
20 }
21
```


1. CONSTRUCCION CRUD

Vistas

Paso 15) El método handler tendrá un parámetro que será un objeto de la clase Usuario. El acceso a las variables del formulario se hará mediante los getters de la clase:

```
Controlador.java
1 package com.example.demo.controllers;
2
3 import org.springframework.stereotype.Controller;
11
12 @Controller //Lo convertimos en un servlet atiende peticiones http
13 @RequestMapping("")
14 public class Controlador {
15
16     @GetMapping("/")
17     public String iniciar(Model model) {
18         model.addAttribute("titulo", "FORMULARIO DE ACCESO");
19         return "login";
20     }
21
22     @PostMapping("/")
23     public String login(Usuario usuario, Model model) {
24         if (usuario.getNombre().equals("edu") && usuario.getPassword().equals("edu"))
25             return "consulta";
26         else
27             return "login";
28     }
29 }
30
```

1. CONSTRUCCION CRUD

Vistas

Paso 16) Como complemento, una vez logados satisfactoriamente, se pueden pasar los datos del usuario a la consulta para mostrarlos:

```
Controlador.java
1 package com.example.demo.controllers;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller //lo convertimos en un servlet atiende peticiones http
6 @RequestMapping("")
7 public class Controlador {
8
9     @GetMapping("/")
10     public String iniciar(Model model) {
11         model.addAttribute("titulo","FORMULARIO DE ACCESO");
12         return "login";
13     }
14
15     @PostMapping("/")
16     public String login(Usuario usuario, Model model) {
17         if (usuario.getNombre().equals("edu") && usuario.getPassword().equals("edu")) {
18             model.addAttribute("usuario",usuario);
19             return "consulta";
20         }else
21             return "login";
22     }
23 }
24
25
26
27
28
29
30
31
```

```
consulta.html
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <meta charset="ISO-8859-1">
5     <title>LISTADO DE LIBROS</title>
6 </head>
7 <body>
8     <h3>Usuario: <span th:text="${usuario.nombre}"> </span></h3>
9     <h3>Password: <span th:text="${usuario.password}"> </span></h3>
10
11 <H1>LISTADO DE LIBROS</H1>
12
13 </body>
14 </html>
15
```

2. BACKEND CON ARRAYLIST

Paso 1) Primero crearemos la clase Libro dentro del package bean, que utilizaremos como clase Entity para la transferencia de información entre vistas y controlador:

```
Libro.java
1 package com.example.demo.bean;
2
3 public class Libro {
4     private int id;
5     private String titulo;
6     private String autor;
7     private String editorial;
8     private String fecha;
9     private String tematica;
10
11     public Libro(int id, String titulo, String autor,
12                 String editorial, String fecha, String tematica) {
13         this.id = id;
14         this.titulo = titulo;
15         this.autor = autor;
16         this.editorial = editorial;
17         this.fecha = fecha;
18         this.tematica = tematica;
19     }
```

Más adelante (en backend mediante JPA) será imprescindible declarar los correspondientes getters y setters de los atributos y declarar el constructor por defecto de la clase (sin parámetros)

2. BACKEND CON ARRAYLIST

Paso 2) Creamos la clase BaseDatos() dentro de un nuevo package repository, la cual en su constructor carga el ArrayList que utilizaremos como backend:



The screenshot shows an IDE with the Package Explorer on the left and the BaseDatos.java file open in the editor. The Package Explorer shows the following structure:

- formulario-sboot [boot] [devtools]
 - src/main/java
 - com.example.demo
 - com.example.demo.bean
 - com.example.demo.controllers
 - com.example.demo.repository
 - BaseDatos.java
 - src/main/resources
 - src/test/java
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - src
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

The BaseDatos.java file contains the following code:

```
5
6 public class BaseDatos {
7
8     ArrayList<Libro> libros = new ArrayList<Libro>();
9     public BaseDatos() {
10         libros.add(new Libro(1,"HARRY POTTER Y EL PRISIONERO DE AZKABAN","J.K ROWINS","SALAMANDRA","26/9/2006 0:00:00","INFANTIL"));
11         libros.add(new Libro(2,"EL GRAN LABERINTO","FERNANDO SABATER PEREZ","ARIEL","26/9/2006 0:00:00","FICCION"));
12         libros.add(new Libro(3,"ROMEO Y JULIETA","WILLIAM SHAKESPEARE","SALAMANDRA","26/9/2006 0:00:00","ROMANTICA"));
13         libros.add(new Libro(4,"LA CARTA ESFERICA","ARTURO PEREZ LOPEZ","SALAMANDRA","29/9/2006 0:00:00","FICCION"));
14         libros.add(new Libro(5,"CODIGO DA VINCI","DAN BROWN","ARIEL","29/9/2006 0:00:00","FICCION"));
15         libros.add(new Libro(6,"MUCHO RUIDO Y POCAS NUECES","WILLIAM SHAKESPEARE","SALAMANDRA","29/9/2006 0:00:00","ROMANTICA"));
16         libros.add(new Libro(7,"PROTOCOLO","JOSE LOPEZ MURILLO","SALAMANDRA","6/9/2006 0:00:00","SOCIAL"));
17         libros.add(new Libro(8,"LINUX","FERNANDO SABATER PEREZ","ARIEL","6/9/2006 0:00:00","INFORMATICA"));
18         libros.add(new Libro(9,"EL TUMULTO","H.P LOVERCRAFT","DEBATE","6/9/2006 0:00:00","CIENCIA"));
19         libros.add(new Libro(10,"PERSONAJES MITICOS","RICHARD HOLLIGHAM","DEBATE","7/9/2006 0:00:00","ENTRETENIMIENTO"));
20         libros.add(new Libro(11,"EL TIEMPO","J.K ROWINS","SALAMANDRA","7/9/2006 0:00:00","CIENCIA"));
21         libros.add(new Libro(12,"DIETAS MEDITERRANEAS","ARTURO PEREZ LOPEZ","ARIEL","16/9/2006 0:00:00","ASTRONOMIA"));
22         libros.add(new Libro(13,"ANGELES Y DEMONIOS","DAN BROWN","ARIEL","17/9/2006 0:00:00","FICCION"));
23         libros.add(new Libro(14,"FORTALEZA DIGITAL","DAN BROWN","ARIEL","6/10/2006 0:00:00","FICCION"));
24         libros.add(new Libro(15,"CAPITAN ALATRISTE","ARTURO PEREZ LOPEZ","ALFAGUARA","9/10/2006 0:00:00","FICCION"));
25         libros.add(new Libro(16,"PIEL DE TAMBOR","ARTURO PEREZ LOPEZ","ALFAGUARA","16/10/2006 0:00:00","FICCION"));
26         libros.add(new Libro(17,"TIEMPOS DE COLERA","GABRIEL GARCIA GARCIA","OVEJA NEGRA","1/9/2006 0:00:00","OCIO"));
27         libros.add(new Libro(18,"NOTICIA DE UN SECUESTRO","GABRIEL GARCIA GARCIA","ALFAGUARA","7/12/2006 0:00:00","FICCION"));
28     }
29     public ArrayList<Libro> getLibros() {
30         return libros;
31     }
32     public void setLibros(ArrayList<Libro> libros) {
33         this.libros = libros;
34     }
35 }
```

2. BACKEND CON ARRAYLIST

Paso 3) Dentro del controlador creamos una instancia de la clase BaseDatos. La utilizaremos dentro del método login para obtener el ArrayList de Libros, el cual pasaremos a la vista consulta:

```
Controlador.java
15
16 @Controller //Lo convertimos en un servlet atiende peticiones http
17 @RequestMapping("")
18 public class Controlador {
19
20     BaseDatos bd = new BaseDatos();
21
22     @GetMapping("/")
23     public String iniciar(Model model) {
24         model.addAttribute("titulo", "FORMULARIO DE ACCESO");
25         return "login";
26     }
27
28     @PostMapping("/")
29     public String login(Usuario usuario, Model model) {
30         if (usuario.getNombre().equals("edu") && usuario.getPassword().equals("edu")) {
31             ArrayList<Libro> libros = bd.getLibros();
32             model.addAttribute("usuario", usuario);
33             model.addAttribute("libros", libros);
34             return "consulta";
35         } else
36             return "login";
37     }
38
39 }
```

2. BACKEND CON ARRAYLIST

Paso 4) Modificamos consulta para que reciba la lista de libros y la muestre formateada en una tabla html:

```
consulta.html
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <meta charset="ISO-8859-1">
5   <title>LISTADO DE LIBROS</title>
6 </head>
7 <body>
8   <h3>Usuario: <span th:text="${usuario.nombre}"> </span> - Password: <span th:text="${usuario.password}"> </span></h3>
9
10  <H1>LISTADO DE LIBROS</H1>
11  <table border=1>
12    <thead>
13      <tr><th> Id <th> Titulo <th> Autor <th> Editorial <th> Fecha <th> Tematica
14    </tr>
15  </thead>
16  <tbody>
17    <tr th:if="${libros.empty}">
18      <td colspan="2"> No Books Available </td>
19    </tr>
20    <tr th:each="libro : ${libros}">
21      <td><span th:text="${libro.id}"></span></td>
22      <td><span th:text="${libro.titulo}"></span></td>
23      <td><span th:text="${libro.autor}"></span></td>
24      <td><span th:text="${libro.editorial}"></span></td>
25      <td><span th:text="${libro.fecha}"></span></td>
26      <td><span th:text="${libro.tematica}"></span></td>
27    </tr>
28  </tbody>
29 </table>
```

2. BACKEND CON ARRAYLIST

Paso 5) Podemos observar como queda el resultado:

LISTADO DE LIBROS

localhost:8080

Usuario: edu Password: edu

LISTADO DE LIBROS

Id	Titulo	Autor	Editorial	Fecha	Tematica
1	HARRY POTTER Y EL PRISIONERO DE AZKABAN	J.K ROWINS	SALAMANDRA	26/9/2006 0:00:00	INFANTIL
2	EL GRAN LABERINTO	FERNANDO SABATER PEREZ	ARIEL	26/9/2006 0:00:00	FICCION
3	ROMEO Y JULIETA	WILLIAM SHAKESPEARE	SALAMANDRA	26/9/2006 0:00:00	ROMANTICA
4	LA CARTA ESFERICA	ARTURO PEREZ LOPEZ	SALAMANDRA	29/9/2006 0:00:00	FICCION
5	CODIGO DA VINCI	DAN BROWN	ARIEL	29/9/2006 0:00:00	FICCION
6	MUCHO RUIDO Y POCAS NUECES	WILLIAM SHAKESPEARE	SALAMANDRA	29/9/2006 0:00:00	ROMANTICA
7	PROTOCOLO	JOSE LOPEZ MURILLO	SALAMANDRA	6/9/2006 0:00:00	SOCIAL
8	LINUX	FERNANDO SABATER PEREZ	ARIEL	6/9/2006 0:00:00	INFORMATICA
9	EL TUMULTO	H.P LOVERCRAFT	DEBATE	6/9/2006 0:00:00	CIENCIA
10	PERSONAJES MITICOS	RICHARD HOLLIGHAM	DEBATE	7/9/2006 0:00:00	ENTRETENIMIENTO
11	EL TIEMPO	J.K ROWINS	SALAMANDRA	7/9/2006 0:00:00	CIENCIA
12	DIETAS MEDITERRANEAS	ARTURO PEREZ LOPEZ	ARIEL	16/9/2006 0:00:00	ASTRONOMIA
13	ANGELES Y DEMONIOS	DAN BROWN	ARIEL	17/9/2006 0:00:00	FICCION
14	FORTALEZA DIGITAL	DAN BROWN	ARIEL	6/10/2006 0:00:00	FICCION
15	CAPITAN ALATRISTE	ARTURO PEREZ LOPEZ	ALFAGUARA	9/10/2006 0:00:00	FICCION
16	PIEL DE TAMBOR	ARTURO PEREZ LOPEZ	ALFAGUARA	16/10/2006 0:00:00	FICCION
17	TIEMPOS DE COLERA	GABRIEL GARCIA GARCIA	OVEJA NEGRA	1/9/2006 0:00:00	OCIO
18	NOTICIA DE UN SECUESTRO	GABRIEL GARCIA GARCIA	ALFAGUARA	7/12/2006 0:00:00	FICCION

2. BACKEND CON ARRAYLIST

Inserción

Paso 6) Para el proceso de inserción debemos crear un formulario debajo del listado de la vista consulta.html, tal y como se ve en la figura:

```
consulta.html
22      <td><span th:text="${libro.titulo}"> Title </span></td>
23      <td><span th:text="${libro.autor}"> Author </span></td>
24      <td><span th:text="${libro.editorial}"> Editorial </span></td>
25      <td><span th:text="${libro.fecha}"> Author </span></td>
26      <td><span th:text="${libro.tematica}"> Author </span></td>
27    </tr>
28  </tbody>
29 </table>
30 <form action="/insertar" method="post">
31   <br>ID: <input type="text" name="id">
32   TITULO: <input type="text" name="titulo">
33   AUTOR: <input type="text" name="autor"><br>
34   EDITORIAL: <input type="text" name="editorial">
35   FECHA: <input type="text" name="fecha">
36   TEMATICA: <input type="text" name="tematica"><br>
37   <input type="submit" name="submit" value="Insertar Libro">
38 </form>
39 </body>
40 </html>
41
```

LISTADO DE LIBROS

Usuario: edu - Password: edu

LISTADO DE LIBROS

Id	Título	Autor	Editorial	Fecha	Tematica
1	HARRY POTTER Y EL PRISIONERO DE AZKABAN	J.K ROWINS	SALAMANDRA	26/9/2006 0:00:00	INFANTIL
2	EL GRAN LABERINTO	FERNANDO SABATER PEREZ	ARIEL	26/9/2006 0:00:00	FICCION
3	ROMEO Y JULIETA	WILLIAM SHAKESPEARE	SALAMANDRA	26/9/2006 0:00:00	ROMANTICA
4	LA CARTA ESFERICA	ARTURO PEREZ LOPEZ	SALAMANDRA	29/9/2006 0:00:00	FICCION
5	CODIGO DA VINCI	DAN BROWN	ARIEL	29/9/2006 0:00:00	FICCION
6	MUCHO RUIDO Y POCAS NUECES	WILLIAM SHAKESPEARE	SALAMANDRA	29/9/2006 0:00:00	ROMANTICA
7	PROTOCOLO	JOSE LOPEZ MURILLO	SALAMANDRA	6/9/2006 0:00:00	SOCIAL
8	LINUX	FERNANDO SABATER PEREZ	ARIEL	6/9/2006 0:00:00	INFORMATICA
9	EL TUMULTO	H.P LOVERCRAFT	DEBATE	6/9/2006 0:00:00	CIENCIA
10	PERSONAJES MITICOS	RICHARD HOLLIGHAM	DEBATE	7/9/2006 0:00:00	ENTRETENIMIENTO
11	EL TIEMPO	J.K ROWINS	SALAMANDRA	7/9/2006 0:00:00	CIENCIA
12	DIETAS MEDITERRANEAS	ARTURO PEREZ LOPEZ	ARIEL	16/9/2006 0:00:00	ASTRONOMIA
13	ANGELES Y DEMONIOS	DAN BROWN	ARIEL	17/9/2006 0:00:00	FICCION
14	FORTALEZA DIGITAL	DAN BROWN	ARIEL	6/10/2006 0:00:00	FICCION
15	CAPITAN ALATRISTE	ARTURO PEREZ LOPEZ	ALFAGUARA	9/10/2006 0:00:00	FICCION
16	PIEL DE TAMBOR	ARTURO PEREZ LOPEZ	ALFAGUARA	16/10/2006 0:00:00	FICCION
17	TIEMPOS DE COLERA	GABRIEL GARCIA GARCIA	OVEJA NEGRA	1/9/2006 0:00:00	OCIO
18	NOTICIA DE UN SECUESTRO	GABRIEL GARCIA GARCIA	ALFAGUARA	7/12/2006 0:00:00	FICCION

ID: TITULO: AUTOR:
EDITORIAL: FECHA: TEMATICA:

2. BACKEND CON ARRAYLIST

Inserción

Paso 7) En el controlador debemos crear un nuevo handler con PostMapping insertar, que atenderá las peticiones de inserción de un nuevo libro

Como inicialmente ya pasábamos el usuario y el password a consulta.html, ahora debemos realizar una estrategia para continuar haciendo lo mismo.

```
Controlador.java
16 @Controller //Lo convertimos en un servlet atiende peticiones http
17 @RequestMapping("")
18 public class Controlador {
19
20     BaseDatos bd = new BaseDatos();
21     Usuario usuario;
22
23     @GetMapping("/")
24     public String iniciar(Model model) {
25         model.addAttribute("titulo","FORMULARIO DE ACCESO");
26         return "login";
27     }
28
29     @PostMapping("/")
30     public String login(Usuario usuario, Model model) {
31         if (usuario.getNombre().equals("edu") && usuario.getPassword().equals("edu")) {
32             ArrayList<Libro> libros = bd.getLibros();
33             model.addAttribute("usuario",usuario);
34             this.usuario = usuario;
35             model.addAttribute("libros",libros);
36             return "consulta";
37         }else
38             return "login";
39     }
40
41     @PostMapping("/insertar")
42     public String insertar(Libro libro, Model model) {
43         bd.inserta(libro);
44         ArrayList<Libro> libros = bd.getLibros();
45         model.addAttribute("usuario",this.usuario);
46         model.addAttribute("libros",libros);
47         return "consulta";
48     }
49 }
```

2. BACKEND CON ARRAYLIST

Inserción

Paso 8) Finalmente en el servicio BaseDatos.java, se debe de crear la función inserta libro.

```
BaseDatos.java
7
8 public class BaseDatos {
9
10     ArrayList<Libro> libros = new ArrayList<Libro>();
11     public BaseDatos() {
12         libros.add(new Libro(1,"HARRY POTTER Y EL PRISIONERO DE AZKABAN","J.K ROWINS","SALAMANDRA","26/9/2006 0:00:00","INFANTIL"));
13         libros.add(new Libro(2,"EL GRAN LABERINTO","FERNANDO SABATER PEREZ","ARIEL","26/9/2006 0:00:00","FICCION"));
14         libros.add(new Libro(3,"ROMEO Y JULIETA","WILLIAM SHAKESPEARE","SALAMANDRA","26/9/2006 0:00:00","ROMANTICA"));
15         libros.add(new Libro(4,"LA CARTA ESFERICA","ARTURO PEREZ LOPEZ","SALAMANDRA","29/9/2006 0:00:00","FICCION"));
16         libros.add(new Libro(5,"CODIGO DA VINCI","DAN BROWN","ARIEL","29/9/2006 0:00:00","FICCION"));
17         libros.add(new Libro(6,"MUCHO RUIDO Y POCAS NUECES","WILLIAM SHAKESPEARE","SALAMANDRA","29/9/2006 0:00:00","ROMANTICA"));
18         libros.add(new Libro(7,"PROTOCOLO","JOSE LOPEZ MURILLO","SALAMANDRA","6/9/2006 0:00:00","SOCIAL"));
19         libros.add(new Libro(8,"LINUX","FERNANDO SABATER PEREZ","ARIEL","6/9/2006 0:00:00","INFORMATICA"));
20         libros.add(new Libro(9,"EL TUMULTO","H.P LOVERCRAFT","DEBATE","6/9/2006 0:00:00","CIENCIA"));
21         libros.add(new Libro(10,"PERSONAJES MITICOS","RICHARD HOLLIGHAM","DEBATE","7/9/2006 0:00:00","ENTRETENIMIENTO"));
22         libros.add(new Libro(11,"EL TIEMPO","J.K ROWINS","SALAMANDRA","7/9/2006 0:00:00","CIENCIA"));
23         libros.add(new Libro(12,"DIETAS MEDITERRANEAS","ARTURO PEREZ LOPEZ","ARIEL","16/9/2006 0:00:00","ASTRONOMIA"));
24         libros.add(new Libro(13,"ANGELES Y DEMONIOS","DAN BROWN","ARIEL","17/9/2006 0:00:00","FICCION"));
25         libros.add(new Libro(14,"FORTALEZA DIGITAL","DAN BROWN","ARIEL","6/10/2006 0:00:00","FICCION"));
26         libros.add(new Libro(15,"CAPITAN ALATRISTE","ARTURO PEREZ LOPEZ","ALFAGUARA","9/10/2006 0:00:00","FICCION"));
27         libros.add(new Libro(16,"PIEL DE TAMBOR","ARTURO PEREZ LOPEZ","ALFAGUARA","16/10/2006 0:00:00","FICCION"));
28         libros.add(new Libro(17,"TIEMPOS DE COLERA","GABRIEL GARCIA GARCIA","OVEJA NEGRA","1/9/2006 0:00:00","OCIO"));
29         libros.add(new Libro(18,"NOTICIA DE UN SECUESTRO","GABRIEL GARCIA GARCIA","ALFAGUARA","7/12/2006 0:00:00","FICCION"));
30     }
31
32     public void inserta(Libro libro) {
33         libros.add(libro);
34     }
35 }
```

2. BACKEND CON ARRAYLIST

Borrado

Paso 9) Para el borrado crearemos una nueva columna paralela al listado de libros. Insertaremos un enlace a la url `/borrado/{id}`, de manera que nos redirigirá a un método del controlador que borrará el libro a través de su id.

```
consulta.html
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <meta charset="ISO-8859-1">
5   <title>LISTADO DE LIBROS</title>
6 </head>
7 <body>
8   <h3>Usuario: <span th:text="${usuario.nombre}"> </span> - Password: <span th:text="${usuario.password}"> </span></h3>
9
10  <h1>LISTADO DE LIBROS</h1>
11  <table border=1>
12    <thead>
13      <tr><th> Id <th> Titulo <th> Autor <th> Editorial <th> Fecha <th> Tematica <th> Borrado
14    </tr>
15  </thead>
16  <tbody>
17    <tr th:if="${libros.empty}">
18      <td colspan="2"> No Books Available </td>
19    </tr>
20    <tr th:each="libro : ${libros}">
21      <td><span th:text="${libro.id}"></span></td>
22      <td><span th:text="${libro.titulo}"></span></td>
23      <td><span th:text="${libro.autor}"></span></td>
24      <td><span th:text="${libro.editorial}"></span></td>
25      <td><span th:text="${libro.fecha}"></span></td>
26      <td><span th:text="${libro.tematica}"></span></td>
27      <td><a th:href="@{/borrado/${libro.id}}">Borrado</a></td>
28    </tr>
29  </tbody>
30 </table>
```

2. BACKEND CON ARRAYLIST

Borrado

Paso 10) Creamos un método Get en la url `/borrado/{id}` del controlador. Después de borrar el libro, se vuelve a obtener la lista actualizada de libros para pasársela a la vista.

Hay más parámetros que se pasan a la vista.

- El parámetro usuario es para mantenerlo ya que se pasó al inicio.
- Los parámetros botón y action es para modificar el formulario durante el proceso de modificación que veremos a continuación

```
Controlador.java
43
44 @PostMapping("/insertar")
45 public String insertar(Libro libro, Model model) {
46     bd.inserta(libro);
47     ArrayList<Libro> libros = bd.getLibros();
48     model.addAttribute("usuario",this.usuario);
49     model.addAttribute("libros",libros);
50     model.addAttribute("boton","Inserta Libro");
51     model.addAttribute("action","/insertar");
52     model.addAttribute("libro",null);
53     return "consulta";
54 }
55
56 @GetMapping("/borrado/{id}")
57 public String borrar(@PathVariable int id, Model model) {
58     bd.borrar(id);
59     ArrayList<Libro> libros = bd.getLibros();
60     model.addAttribute("libros",libros);
61     model.addAttribute("usuario",this.usuario);
62     model.addAttribute("boton","Inserta Libro");
63     model.addAttribute("action","/insertar");
64     return "consulta";
65 }
```

2. BACKEND CON ARRAYLIST

Borrado

Paso 11) Finalmente para el proceso de borrado debemos crear la función borrar en BaseDatos.java:

```
BaseDatos.java
38
39 public void borrar(int id) {
40     Iterator<Libro> it = libros.iterator();
41     while(it.hasNext()) {
42         Libro li = it.next();
43         if (li.getId()==id) {
44             it.remove();
45             break;
46         }
47     }
48 }
```

2. BACKEND CON ARRAYLIST

Modificación

Paso 12) Creamos una nueva columna “Modificar” al lado de Borrado, que contenga un enlace con el identificador del libro. Esto nos permita realizar la primera acción de la modificación que es mostrar el libro en el formulario de inserción:

```
consulta.html
7 <body>
8 <h3>Usuario: <span th:text="${usuario.nombre}" /> - Password: <span th:text="${usuario.password}" /> </span></h3>
9
10 <H1>LISTADO DE LIBROS</H1>
11 <table border=1>
12 <thead>
13 <tr><th> Id </th> <th> Titulo </th> <th> Autor </th> <th> Editorial </th> <th> Fecha </th> <th> Tematica </th> <th> Borrado </th> <th> Modificar </th>
14 </tr>
15 </thead>
16 <tbody>
17 <tr th:if="${libros.empty}">
18 <td colspan="2"> No Books Available </td>
19 </tr>
20 <tr th:each="libro : ${libros}">
21 <td><span th:text="${libro.id}" /></td>
22 <td><span th:text="${libro.titulo}" /></td>
23 <td><span th:text="${libro.autor}" /></td>
24 <td><span th:text="${libro.editorial}" /></td>
25 <td><span th:text="${libro.fecha}" /></td>
26 <td><span th:text="${libro.tematica}" /></td>
27 <td><a th:href="@{/borrado/{id}/${libro.id} }">Borrado</a></td>
28 <td><a th:href="@{/modificar/{id}/${libro.id} }">Modificar</a></td>
29 </tr>
30 </tbody>
31 </table>
32 <form action="/insertar" method="post">
33 <br>ID: <input type="text" name="id">
34 TITULO: <input type="text" name="titulo">
```

LISTADO DE LIBROS

Usuario: edu - Password: edu

LISTADO DE LIBROS

Id	Titulo	Autor	Editorial	Fecha	Tematica	Borrado	Modificar
1	HARRY POTTER Y EL PRISIONERO DE AZKABAN	J K ROWINS	SALAMANDRA	26/9/2006 0:00:00	INFANTIL	Borrado	Modificar
2	EL GRAN LABERINTO	FERNANDO SABATER PEREZ	ARIEL	26/9/2006 0:00:00	FICCION	Borrado	Modificar
3	ROMEO Y JULIETA	WILLIAM SHAKESPEARE	SALAMANDRA	26/9/2006 0:00:00	ROMANTICA	Borrado	Modificar
4	LA CARTA ESFERICA	ARTURO PEREZ LOPEZ	SALAMANDRA	26/9/2006 0:00:00	FICCION	Borrado	Modificar
5	CODIGO DA VINCI	DAN BROWN	ARIEL	26/9/2006 0:00:00	FICCION	Borrado	Modificar
6	MUCHO RUIDO Y POCAS NUECES	WILLIAM SHAKESPEARE	SALAMANDRA	26/9/2006 0:00:00	ROMANTICA	Borrado	Modificar
7	PROTOCOLO	JOSE LOPEZ MURILLO	SALAMANDRA	6/9/2006 0:00:00	SOCIAL	Borrado	Modificar
8	LINUX	FERNANDO SABATER PEREZ	ARIEL	6/9/2006 0:00:00	INFORMATICA	Borrado	Modificar
9	EL TUMULTO	H P LOVERCRAFT	DEBATE	6/9/2006 0:00:00	CIENCIA	Borrado	Modificar
10	PERSONAJES MITICOS	RICHARD HOLLIGHAM	DEBATE	7/9/2006 0:00:00	ENTRETENIMIENTO	Borrado	Modificar
11	EL TIEMPO	J K ROWINS	SALAMANDRA	7/9/2006 0:00:00	CIENCIA	Borrado	Modificar
12	DIETAS MEDITERRANEAS	ARTURO PEREZ LOPEZ	ARIEL	16/9/2006 0:00:00	ASTRONOMIA	Borrado	Modificar
13	ANGELES Y DEMONIOS	DAN BROWN	ARIEL	17/9/2006 0:00:00	FICCION	Borrado	Modificar
14	FORTALEZA DIGITAL	DAN BROWN	ARIEL	6/10/2006 0:00:00	FICCION	Borrado	Modificar
15	CAPITAN ALATRISTE	ARTURO PEREZ LOPEZ	ALFAGUARA	9/10/2006 0:00:00	FICCION	Borrado	Modificar
16	PIEL DE TAMBOR	ARTURO PEREZ LOPEZ	ALFAGUARA	16/10/2006 0:00:00	FICCION	Borrado	Modificar
17	TIEMPOS DE COLERA	GABRIEL GARCIA GARCIA	OVEJA NEGRA	1/9/2006 0:00:00	OCIO	Borrado	Modificar
18	NOTICIA DE UN SECUESTRO	GABRIEL GARCIA GARCIA	ALFAGUARA	7/12/2006 0:00:00	FICCION	Borrado	Modificar

ID: _____ TITULO: _____ AUTOR: _____
EDITORIAL: _____ FECHA: _____ TEMATICA: _____
[Insertar Libro](#)

2. BACKEND CON ARRAYLIST

Modificación

Paso 13) Creamos dos handlers dentro del controlador, uno que nos permita rellenar el formulario de actualización y otro que trate el envío post de este formulario.

```
Controlador.java
66
67 @GetMapping("/modificar/{id}")
68 public String modificar(@PathVariable int id, Model model) {
69     Libro libro = bd.getLibro(id);
70     ArrayList<Libro> libros = bd.getLibros();
71     model.addAttribute("libros", libros);
72     model.addAttribute("libro", libro);
73     model.addAttribute("usuario", this.usuario);
74     model.addAttribute("boton", "Actualiza Libro");
75     model.addAttribute("action", "/modificar");
76     return "consulta";
77 }
78
79 @PostMapping("/modificar")
80 public String modificar2(Libro libro, Model model) {
81     bd.modifica(libro);
82     ArrayList<Libro> libros = bd.getLibros();
83     model.addAttribute("usuario", this.usuario);
84     model.addAttribute("libros", libros);
85     model.addAttribute("libro", null);
86     model.addAttribute("boton", "Inserta Libro");
87     model.addAttribute("action", "/insertar");
88     return "consulta";
89 }
90 }
91
```


2. BACKEND CON ARRAYLIST

Modificación

Paso 14) Modificamos el formulario de consulta para que acepte los parámetros de un hipotético libro que se quiera modificar, mediante la sintaxis condicional `${libro?.id}`

```
consulta.html
21 <td><span th:text="${libro.id}"></span></td>
22 <td><span th:text="${libro.titulo}"></span></td>
23 <td><span th:text="${libro.autor}"></span></td>
24 <td><span th:text="${libro.editorial}"></span></td>
25 <td><span th:text="${libro.fecha}"></span></td>
26 <td><span th:text="${libro.tematica}"></span></td>
27 <td><a th:href="@{/borrado/}+${libro.id}">Borrado</a></td>
28 <td><a th:href="@{/modificar/}+${libro.id}">Modificar</a></td>
29 </tr>
30 </tbody>
31 </table>
32
33
34 <form th:action="${action}" method="post">
35   <br>ID: <input type="text" name="id" th:value="${libro?.id}">
36   TITULO: <input type="text" name="titulo" th:value="${libro?.titulo}">
37   AUTOR: <input type="text" name="autor" th:value="${libro?.autor}"><br>
38   EDITORIAL: <input type="text" name="editorial" th:value="${libro?.editorial}">
39   FECHA: <input type="text" name="fecha" th:value="${libro?.fecha}">
40   TEMATICA: <input type="text" name="tematica" th:value="${libro?.tematica}"><br>
41   <input type="submit" name="submit" th:value="${boton}">
42 </form>
43
44
45
46 </body>
47 </html>
```


2. BACKEND CON ARRAYLIST

Modificación

Paso 15) Finalmente para el proceso de modificación debemos crear la función modifica en BaseDatos.java

```
BaseDatos.java
49
50 public void modifica(Libro libro) {
51     Iterator<Libro> it = libros.iterator();
52     while(it.hasNext()) {
53         Libro li = it.next();
54         if (li.getId()==libro.getId()) {
55             li.setTitulo(libro.getTitulo());
56             li.setAutor(libro.getAutor());
57             li.setEditorial(libro.getEditorial());
58             li.setFecha(libro.getFecha());
59             li.setTematica(libro.getTematica());
60             break;
61         }
62     }
63 }
```

3. BACKEND CON JDBC

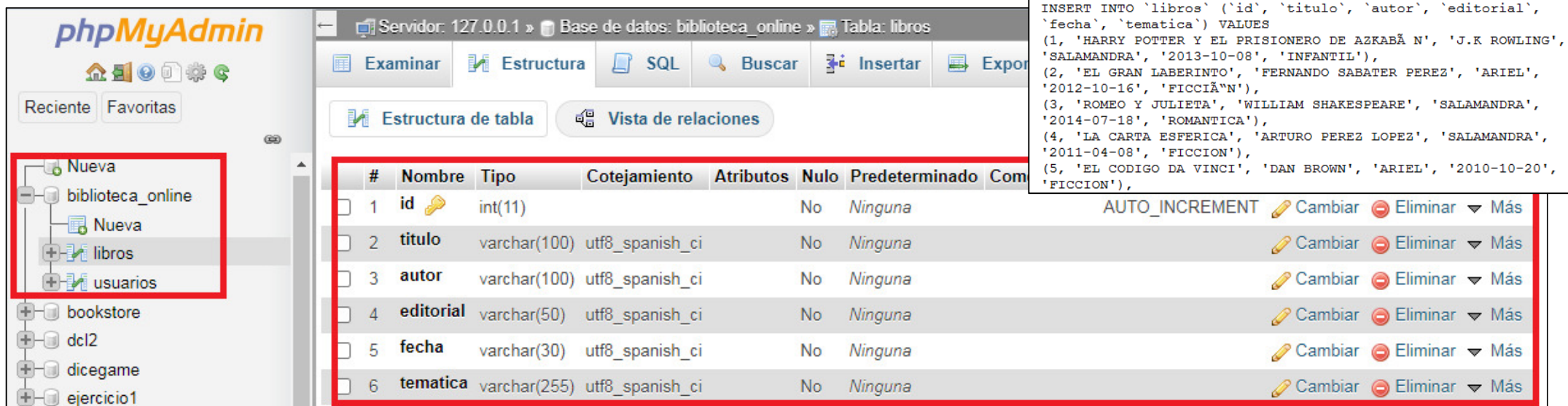
Paso 1) Introducimos las dependencias del driver mysql connector en el fichero pom.xml de maven.



```
23     </dependency>
24 <dependency>
25     <groupId>org.springframework.boot</groupId>
26     <artifactId>spring-boot-starter-web</artifactId>
27 </dependency>
28
29 <dependency>
30     <groupId>org.springframework.boot</groupId>
31     <artifactId>spring-boot-devtools</artifactId>
32     <scope>runtime</scope>
33     <optional>true</optional>
34 </dependency>
35 <dependency>
36     <groupId>org.springframework.boot</groupId>
37     <artifactId>spring-boot-starter-test</artifactId>
38     <scope>test</scope>
39 </dependency>
40
41 <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
42 <dependency>
43     <groupId>mysql</groupId>
44     <artifactId>mysql-connector-java</artifactId>
45     <version>8.0.23</version>
46 </dependency>
47
48 </dependencies>
```

3. BACKEND CON JDBC

Paso 2) Creamos la base de datos biblioteca_online y dentro la tabla libros. Mediante un script cargamos en esta tabla el contenido anterior del arrayList.



The screenshot shows the phpMyAdmin interface for a database named 'biblioteca_online'. The 'libros' table is selected, and its structure is displayed. The table has 6 columns: id (int(11), primary key, auto-increment), titulo (varchar(100)), autor (varchar(100)), editorial (varchar(50)), fecha (varchar(30)), and tematica (varchar(255)). The table is using the utf8_spanish_ci collation. The SQL script on the right shows the CREATE TABLE statement and an INSERT INTO statement with 5 rows of data.

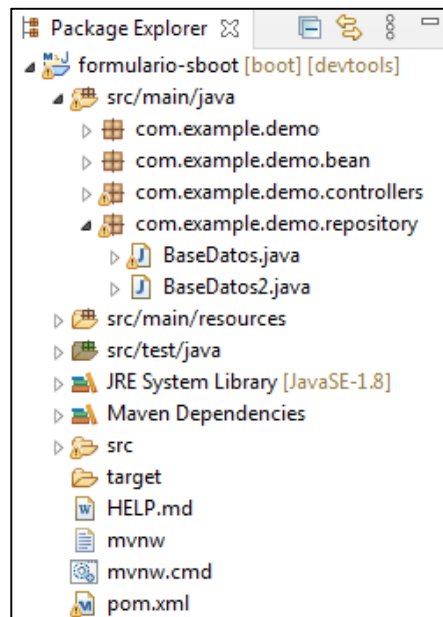
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Com
1	id	int(11)			No	Ninguna	AUTO_INCREMENT
2	titulo	varchar(100)	utf8_spanish_ci		No	Ninguna	
3	autor	varchar(100)	utf8_spanish_ci		No	Ninguna	
4	editorial	varchar(50)	utf8_spanish_ci		No	Ninguna	
5	fecha	varchar(30)	utf8_spanish_ci		No	Ninguna	
6	tematica	varchar(255)	utf8_spanish_ci		No	Ninguna	

```
CREATE TABLE `libros` (  
  `id` int(11) NOT NULL primary key auto_increment,  
  `titulo` varchar(100) NOT NULL,  
  `autor` varchar(100) NOT NULL,  
  `editorial` varchar(50) NOT NULL,  
  `fecha` varchar(30) NOT NULL,  
  `tematica` varchar(50) NOT NULL  
);  
  
--  
-- Volcado de datos para la tabla `libros`  
--  
INSERT INTO `libros` (`id`, `titulo`, `autor`, `editorial`,  
  `fecha`, `tematica`) VALUES  
(1, 'HARRY POTTER Y EL PRISIONERO DE AZKABÃ N', 'J.K ROWLING',  
  'SALAMANDRA', '2013-10-08', 'INFANTIL'),  
(2, 'EL GRAN LABERINTO', 'FERNANDO SABATER PEREZ', 'ARIEL',  
  '2012-10-16', 'FICCIA"N'),  
(3, 'ROMEO Y JULIETA', 'WILLIAM SHAKESPEARE', 'SALAMANDRA',  
  '2014-07-18', 'ROMANTICA'),  
(4, 'LA CARTA ESFERICA', 'ARTURO PEREZ LOPEZ', 'SALAMANDRA',  
  '2011-04-08', 'FICCION'),  
(5, 'EL CODIGO DA VINCI', 'DAN BROWN', 'ARIEL', '2010-10-20',  
  'FICCION'),
```

3. BACKEND CON JDBC

Paso 3) Creamos el fichero BaseDatos2.java con las funciones típicas del CRUD en versión JDBC:

- inserta
- borrar
- modifica
- getLibro



```
BaseDatos2.java
12
13 public class BaseDatos2 {
14
15     private Connection conexion;
16
17     public BaseDatos2() {
18         try {
19             Class.forName("com.mysql.cj.jdbc.Driver");
20             String conex="jdbc:mysql://localhost:3306/biblioteca_online";
21             this.conexion = DriverManager.getConnection (conex,"root","");
22
23         } catch (Exception e) {
24             e.printStackTrace();
25         }
26     }
27
28     public void inserta(Libro libro) {
29         String query = " insert into libros (id, titulo, autor, editorial, fecha, tematica)"
30             + " values (?, ?, ?, ?, ?, ?)";
31         try {
32             PreparedStatement preparedStmt;
33             preparedStmt = conexion.prepareStatement(query);
34             preparedStmt.setInt (1, libro.getId());
35             preparedStmt.setString (2, libro.getTitulo());
36             preparedStmt.setString (3, libro.getAutor());
37             preparedStmt.setString (4, libro.getEditorial());
38             preparedStmt.setString (5, libro.getFecha());
39             preparedStmt.setString (6, libro.getTematica());
40             preparedStmt.executeUpdate();
41         } catch (SQLException ex) {
42             System.out.print(ex.getMessage());
43         }
44     }
45 }
```

3. BACKEND CON JDBC

Paso 4) Aquí tenemos las funciones modifica, borrar y getLibro:

```
BaseDatos2.java
46 public void borrar(int id) {
47     String query = " delete from libros where id="+id;
48
49     try {
50         PreparedStatement preparedStmt = conexion.prepareStatement(query);
51         preparedStmt.executeUpdate();
52     } catch (SQLException ex) {
53         System.out.print(ex.getMessage());
54     }
55 }
56
57 public void modifica(Libro libro) {
58
59     String query = " update libros set titulo=?, autor=?, editorial=?, fecha=?, tematica=? "
60         + " where id=?";
61
62     try {
63         PreparedStatement preparedStmt = conexion.prepareStatement(query);
64         preparedStmt.setString (1, libro.getTitulo());
65         preparedStmt.setString (2, libro.getAutor());
66         preparedStmt.setString (3, libro.getEditorial());
67         preparedStmt.setString (4, libro.getFecha());
68         preparedStmt.setString (5, libro.getTematica());
69         preparedStmt.setInt (6, libro.getId());
70         System.out.print(preparedStmt.toString());
71
72         preparedStmt.executeUpdate();
73     } catch (SQLException ex) {
74         System.out.print(ex.getMessage());
75     }
76 }

public Libro getLibro(int id) {
    Libro libro = null;
    try {
        Statement s = conexion.createStatement();
        String sql = "SELECT * FROM LIBROS WHERE ID="+id;
        s.execute(sql);
        ResultSet rs = s.getResultSet();
        rs.next();
        libro = new Libro(rs.getInt(1),rs.getString(2),rs.getString(3),
            rs.getString(4), rs.getString(5), rs.getString(6));
    } catch (SQLException ex) {
        System.out.print(ex.getMessage());
    }

    return libro;
}
```

3. BACKEND CON JDBC

Paso 5) Aquí tenemos las funciones getLibros y compruebaUsuario:

```
BaseDatos2.java
94
95 public ArrayList<Libro> getLibros() {
96     ArrayList<Libro> lista = new ArrayList<Libro>();
97     try {
98         Statement s = conexion.createStatement();
99         String sql = "SELECT * FROM LIBROS";
100         s.execute(sql);
101         ResultSet rs = s.getResultSet();
102         while (rs.next()) {
103             Libro libro = new Libro(rs.getInt(1), rs.getString(2), rs.getString(3),
104                                     rs.getString(4), rs.getString(5), rs.getString(6));
105             lista.add(libro);
106         }
107     } catch (SQLException ex) {
108         System.out.print(ex.getMessage());
109     }
110     return lista;
111 }
112
113 public boolean compruebaUsuario(String usuario, String password){
114     boolean check=false;
115     try {
116         Statement s = conexion.createStatement();
117         String sql = "SELECT count(*) FROM USUARIOS WHERE usuario='"+usuario+" "
118                     + "and password='"+password+"'";
119         s.execute(sql);
120         ResultSet rs = s.getResultSet();
121         rs.next();
122         if (rs.getInt(1)>0)
123             check=true;
124     } catch (SQLException ex) {
125         System.out.print(ex.getMessage());
126     }
127     return check;
128 }
```

3. BACKEND CON JDBC

Paso 6) En el controlador creamos un atributo de tipo BaseDatos2. Arrancamos la aplicación y comprobamos si su funcionamiento es correcto.

Controlador.java

```
1 package com.example.demo.controllers;
2
3 import java.util.ArrayList;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 @Controller //Lo convertimos en un servlet atiende peticiones http
21 @RequestMapping("") //localhost:8080
22 public class Controlador {
23
24     Usuario usuario;
25     //BaseDatos bd = new BaseDatos();
26     BaseDatos2 bd = new BaseDatos2();
27
28     @GetMapping("/") //Da salida al form
29     public String iniciar(Model model) {
30         model.addAttribute("titulo", "FOR
31         return "login";
32     }
33 }
```

localhost:8080/modificar

Usuario: espai - Password: 123

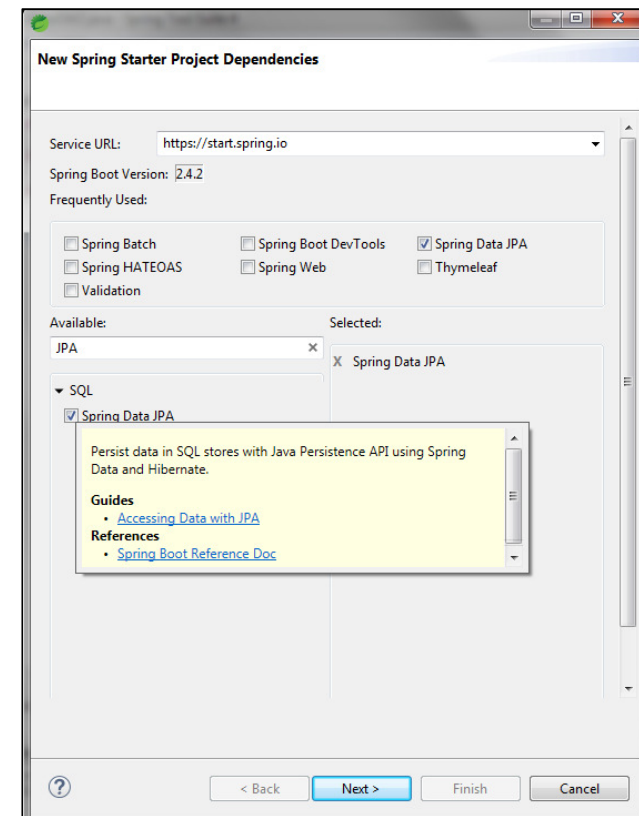
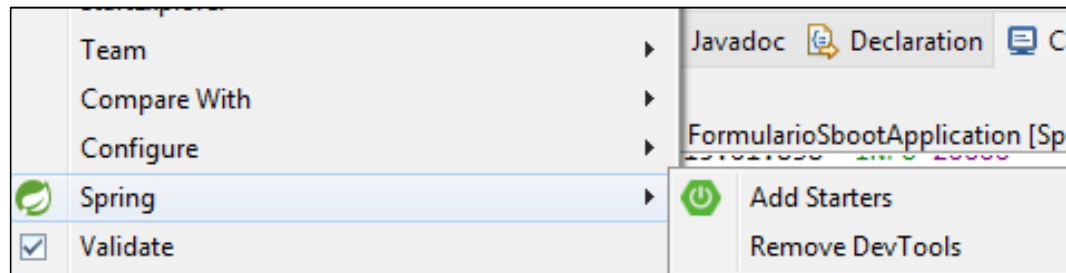
LISTADO DE LIBROS

Id	Titulo	Autor	Editorial	Fecha	Tematica	Borrado	Modificar
14	FORTALEZA DIGITAL	DAN BROWN	ARIEL	2011-04-19	FICCION	Borrado	Modificar
15	CAPITAN EDUARD	ARTURO LARA	ALFAGUARA	2001-04-17	HISTORICA	Borrado	Modificar
16	PIEL DE TAMBOR	ARTURO PEREZ REVERTE	ALFAGUARA	2009-04-14	HISTORICA	Borrado	Modificar
17	TIEMPOS DE COLERA	GABRIEL GARCIA	OVEJA NEGRA	2002-08-04	HISTORICA	Borrado	Modificar
18	NOTICIA DE UN SECUESTRO	GABRIEL GARCIA	ALFAGUARA	2013-05-13	INTRIGA	Borrado	Modificar

ID: TITULO: AUTOR:
EDITORIAL: FECHA: TEMATICA:

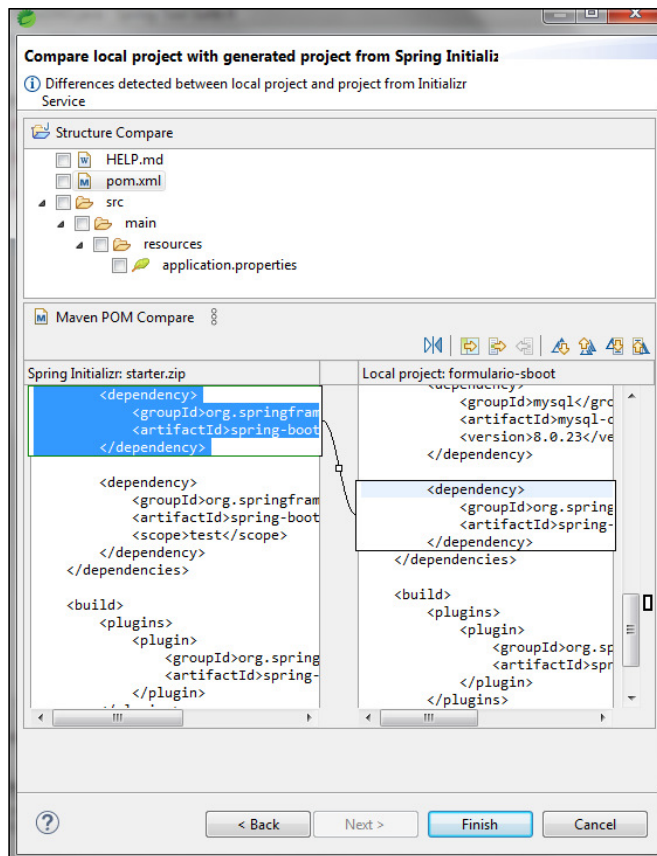
4. BACKEND CON JPA SPRING

Paso 1) Debemos agregar la librería JPA Spring Data necesaria para la persistencia. Hacemos click botón derecho encima del proyecto y seleccionamos Spring/Add Starters, Buscamos la librería JPA y la seleccionamos:



4. BACKEND CON JPA SPRING

Paso 2) Al final de este proceso vemos que ha quedado agregada la librería en el fichero pom.xml de Maven:



```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.23</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
</dependencies>
```

4. BACKEND CON JPA SPRING

Paso 3) A continuación transformamos la clase Libro en una clase Entity mediante las anotaciones JPA (recordar que se deben de generar el constructor por defecto y los getters y setters de los atributos de la clase):

@Entity y **@Table** → Indicamos la tabla a la que referencia esta clase

@Id → Indicamos cual es el clave primaria de la tabla.

@GeneratedValue → Indicamos que se trata de un atributo auto incremento

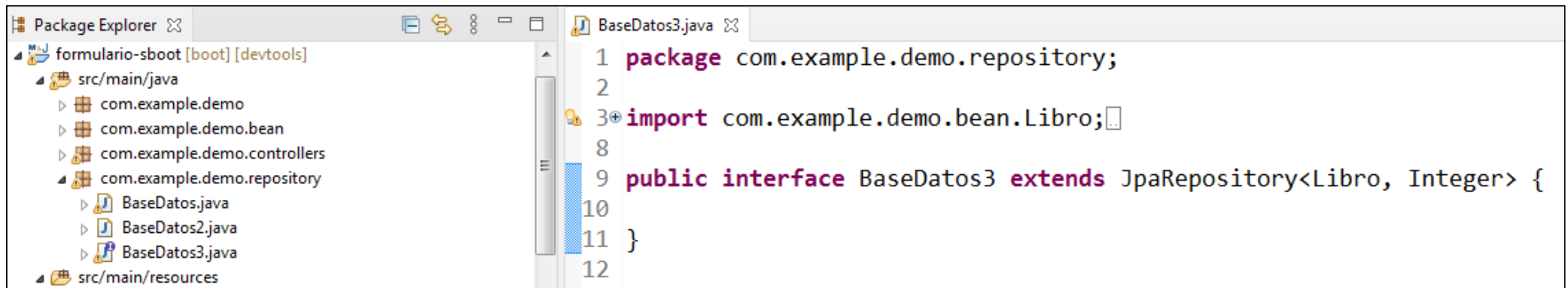
@Column → Indicamos la columna de la tabla a la cual corresponde el atributo.

Si coincide el nombre del atributo y el de la columna de la tabla no hace falta realizar esta indicación.

```
Libro.java
1 package com.example.demo.bean;
2
3 import javax.persistence.Column;
4
5
6
7
8
9
10 @Entity
11 @Table(name="libros")
12 public class Libro {
13
14     @Id
15     @Column(name="id")
16     @GeneratedValue(strategy=GenerationType.IDENTITY)
17     private int id;
18
19     @Column(name="titulo", nullable=false, length=30)
20     private String titulo;
21     private String autor;
22     private String editorial;
23     private String fecha;
24     private String tematica;
25 }
```

4. BACKEND CON JPA SPRING

Paso 4) Crearemos la interfaz DAO BaseDatos3.java que contendrá todas las funciones necesarias del CRUD al heredar de JpaRepository. Mediante la nomenclatura <Libro, Integer> indicamos la clase entity que va a participar en la persistencia y el tipo de dato de su clave primaria, en este caso un entero.



The screenshot shows an IDE with the Package Explorer on the left and the code editor on the right. The Package Explorer shows the project structure for 'formulario-sboot [boot] [devtools]'. The code editor shows the code for 'BaseDatos3.java'.

```
1 package com.example.demo.repository;
2
3 import com.example.demo.bean.Libro;
4
5
6
7
8
9 public interface BaseDatos3 extends JpaRepository<Libro, Integer> {
10
11 }
12
```

4. BACKEND CON JPA SPRING

Clase DAO

Interfaces CrudRepository vs JpaRepository en Spring Data JPA

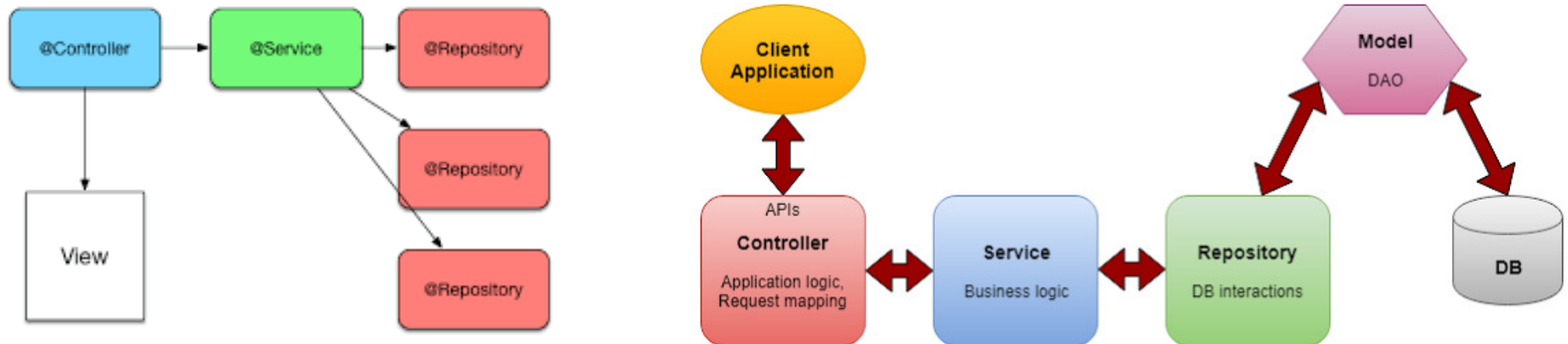
- **CrudRepository** proporciona principalmente funciones CRUD .
- **PagingAndSortingRepository** proporcionan métodos para hacer la paginación y ordenar los registros.
- **JpaRepository** proporciona algunos métodos relacionados con JPA, como el vaciado del contexto de persistencia y la eliminación de registros en un lote. JpaRepository tendrá todas las funciones de CrudRepository y PagingAndSortingRepository (es la mas completa())

Si no necesita que el repositorio tenga las funciones proporcionadas por JpaRepository y PagingAndSortingRepository utilizar CrudRepository

4. BACKEND CON JPA SPRING

Clase Service

Hemos creado la interfaz BaseDatos3 (que sigue el patrón DAO e implementa todas las funcionalidades del crud) dentro del package repository. Este repositorio JPA se puede conectar directamente al controlador. Pero en casos que queramos añadir nuevas funcionalidades, se suele crear un servicio adaptado entre el controlador y el repositorio



4. BACKEND CON JPA SPRING

Paso 5) Crearemos un servicio adaptado al repositorio JPA con el objetivo de que las llamadas a las funciones sean idénticas independientemente del repositorio usado. Sino hacemos esto, tenemos que cambiar el nombre de las funciones cuando usemos el repositorio JPA:

```
20 public class Controlador {
21     Usuario usuario;
22     //BaseDatos bd = new BaseDatos();
23     //BaseDatos2 bd = new BaseDatos2();
24     @Autowired
25     BaseDatos3 bd;
26
27     @PostMapping("/insertar")
28     public String insertar(Libro libro, Model model) {
29         bd.inserta(libro);
30         ArrayList<Libro> libros = bd.getLibros();
31         model.addAttribute("usuario", this.usuario);
32         model.addAttribute("libros", libros);
33         model.addAttribute("boton", "Inserta Libro");
34         model.addAttribute("action", "/insertar");
35         model.addAttribute("libro", null);
36         return "consulta";
37     }
38 }
```

```
public class Controlador {
    Usuario usuario;
    //BaseDatos bd = new BaseDatos();
    //BaseDatos2 bd = new BaseDatos2();
    @Autowired
    BaseDatos3 bd;

    @PostMapping("/insertar")
    public String insertar(Libro libro, Model model) {
        //bd.inserta(libro);
        bd.save(libro);
        //ArrayList<Libro> libros = bd.getLibros();
        ArrayList<Libro> libros = (ArrayList<Libro>) bd.findAll();
        model.addAttribute("usuario", this.usuario);
        model.addAttribute("libros", libros);
        model.addAttribute("boton", "Inserta Libro");
        model.addAttribute("action", "/insertar");
        model.addAttribute("libro", null);
        return "consulta";
    }
}
```

4. BACKEND CON JPA SPRING

Paso 6) La opción de agregar las nuevas funciones directamente en la interface DAO BaseDatos3 no es correcta, ya que al implementar el servicio nos pide eclipse que implementemos también todas las funciones del CRUD JPA:

```
public interface BaseDatos3 extends JpaRepository<Libro, Integer> {  
    public void inserta(Libro libro);           //save  
    public void borrar(int id);                 //deleteById  
    public void modifica(Libro libro);         //save  
    public Libro getLibro(int id);              //findById  
    public ArrayList<Libro> getLibros();        //findAll  
    public boolean compruebaUsuario(  
        String usuario, String password);  
}
```

```
BaseDatos3Service.java  
21 @Service  
22 public class BaseDatos3Service implements BaseDatos3 {  
23  
24     @Override  
25     public List<Libro> findAll() {  
26         // TODO Auto-generated method stub  
27         return null;  
28     }  
29  
30     @Override  
31     public List<Libro> findAll(Sort sort) {  
32         // TODO Auto-generated method stub  
33         return null;  
34     }  
35  
36     @Override  
37     public List<Libro> findAllById(Iterable<Integer> ids) {  
38         // TODO Auto-generated method stub  
39         return null;  
40     }  
41 }
```

4. BACKEND CON JPA SPRING

Paso 7) La opción correcta consiste en crear una interfaz del servicio IBaseDatos3 (dentro del package service) con las funciones necesarias. De esta forma el servicio BaseDatos3Service deberá implementar solo estas funciones:

```
@Service
public class BaseDatos3Service implements IBaseDatos3{

    @Override
    public void inserta(Libro libro) {}

    @Override
    public void borrar(int id) {}

    @Override
    public void modifica(Libro libro) {}

    @Override
    public Libro getLibro(int id) { return null;}

    @Override
    public ArrayList<Libro> getLibros() {return null;}

    @Override
    public boolean compruebaUsuario(String usuario,
                                    String password) {return false; }

}
```

```
public interface IBaseDatos3 {
    public void inserta(Libro libro);
    public void borrar(int id);
    public void modifica(Libro libro);
    public Libro getLibro(int id);
    public ArrayList<Libro> getLibros();
    public boolean compruebaUsuario(String usuario, String password);
}
```

```
public interface BaseDatos3 extends JpaRepository<Libro, Integer> {
    /*public void inserta(Libro libro);           //save
    public void borrar(int id);                   //deleteById
    public void modifica(Libro libro);           //save
    public Libro getLibro(int id);                 //findById
    public ArrayList<Libro> getLibros();           //findAll
    public boolean compruebaUsuario(
        String usuario, String password); //No*/
}
```


4. BACKEND CON JPA SPRING

Paso 8) Implementamos las funciones con terminología JDBC usando las funciones JPA de BaseDatos3. Para ello agregamos un atributo BaseDatos3 que será inicializado mediante la notación @Autowired:

```
BaseDatos3Service.java
17 @Service
18 public class BaseDatos3Service implements IBaseDatos3{
19
20     @Autowired
21     BaseDatos3 bd;
22
23     @Override
24     public void inserta(Libro libro) { bd.save(libro);}
25
26     @Override
27     public void borrar(int id) {bd.deleteById(id);}
28
29     @Override
30     public void modifica(Libro libro) { bd.save(libro);}
31
32     @Override
33     public Libro getLibro(int id) {
34         Optional<Libro> l = bd.findById(id);
35         return l.get();
36     }
37
38     @Override
39     public ArrayList<Libro> getLibros() { return (ArrayList<Libro>) bd.findAll();}
```

JDBC	JPA
inserta	save
borrar	deleteById
modifica	save
getLibro	findById
getLibros	findAll

4. BACKEND CON JPA SPRING

Paso 9) Implementamos la función `compruebaUsuario` dentro del servicio `BaseDatos3Service` mediante el driver JDBC:

```
BaseDatos3Service.java
36 @Override
37 public boolean compruebaUsuario(String usuario, String password) {
38     boolean check=false;
39     try {
40         Class.forName("com.mysql.cj.jdbc.Driver");
41         String conex="jdbc:mysql://localhost:3306/biblioteca_online";
42         Connection conexion = DriverManager.getConnection (conex,"root","");
43         Statement s = conexion.createStatement();
44         String sql = "SELECT count(*) FROM USUARIOS WHERE usuario='"+usuario+"' "
45                     + "and password='"+password+"'";
46         s.execute(sql);
47         ResultSet rs = s.getResultSet();
48         rs.next();
49         if (rs.getInt(1)>0)
50             check=true;
51     } catch (Exception ex) {
52         System.out.print(ex.getMessage());
53     }
54     return check;
55 }
```

4. BACKEND CON JPA SPRING

Paso 10) Debemos definir en el archivo `application.properties` las propiedades de conexión a nuestra base de datos (igual que el connect string de una aplicación JDBC) para que JPA Spring sepa los parámetros de conexión a Mysql

 `application.properties` 

```
1 spring.jpa.hibernate.ddl-auto=update
2 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
3 spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
4 spring.datasource.url=jdbc:mysql://localhost:3306/biblioteca_online
5 spring.datasource.username=root
6 spring.datasource.password=
```

4. BACKEND CON JPA SPRING

Paso 11) En el controlador solo tenemos que crear un atributo de tipo BaseDatos3Service, junto con la anotación @Autowired.

Esto recibe el nombre de inyección de dependencias: dejo que el sistema llame a una clase que implemente dicha interfaz y de esta manera ya podemos utilizar las funciones de dicha interfaz que se corresponde con las funciones de JpaRepository

```
Controlador.java
1 package com.example.demo.controllers;
2 import java.util.ArrayList;
17
18 @Controller //Lo convertimos en un servlet atiende peticiones http
19 @RequestMapping("") //localhost:8080
20 public class Controlador {
21     Usuario usuario;
22     //BaseDatos bd = new BaseDatos();
23     //BaseDatos2 bd = new BaseDatos2();
24 @Autowired
25     BaseDatos3Service bd;
26
27 @PostMapping("/insertar")
28     public String insertar(Libro libro, Model model) {
29         bd.inserta(libro);
30         ArrayList<Libro> libros = bd.getLibros();
31         model.addAttribute("usuario", this.usuario);
32         model.addAttribute("libros", libros);
33         model.addAttribute("boton", "Inserta Libro");
34         model.addAttribute("action", "/insertar");
35         model.addAttribute("libro", null);
36         return "consulta";
37     }
38 }
```

4. BACKEND CON JPA SPRING

Paso 12) Finalmente la aplicación queda de la siguiente manera:

localhost:8080

Usuario: espai - Password: 123

LISTADO DE LIBROS

Id	Titulo	Autor	Editorial	Fecha	Tematica	Borrado	Modificar
1	HARRY POTTER Y EL PRISIONERO DE AZKABÁN	J.K ROWLING	SALAMANDRA	2013-10-08	INFANTIL	Borrado	Modificar
2	EL GRAN LABERINTO	FERNANDO SABATER PEREZ	ARIEL	2012-10-16	FICCION	Borrado	Modificar
3	ROMEO Y JULIETA	WILLIAM SHAKESPEARE	SALAMANDRA	2014-07-18	ROMANTICA	Borrado	Modificar
4	LA CARTA ESFERICA	ARTURO PEREZ LOPEZ	SALAMANDRA	2011-04-08	FICCION	Borrado	Modificar
5	EL CODIGO DA VINCI	DAN BROWN	ARIEL	2010-10-20	FICCION	Borrado	Modificar
6	MUCHO RUIDO Y POCAS NUECES	WILLIAM SHAKESPEARE	SALAMANDRA	2011-09-16	ROMANTICA	Borrado	Modificar
7	PROTOCOLO	JOSE LOPEZ MURILLO	SALAMANDRA	2014-01-21	SOCIAL	Borrado	Modificar
8	LINUX	FERNANDO SABATER PEREZ	ARIEL	2012-02-02	INFORMATICA	Borrado	Modificar
9	EL TUMULTO	H.P LOVECRAFT	DEBATE	2001-07-07	CIENCIA	Borrado	Modificar
10	PERSONAJES MITICOS	RICHARD HOLLIGHAM	DEBATE	2013-01-07	ENTRETENIMIENTO	Borrado	Modificar
11	EL TIEMPO	J.K ROWLING	SALAMANDRA	1999-11-05	CIENCIA	Borrado	Modificar
12	DIETAS MEDITERRANEAS	ARTURO PEREZ LOPEZ	ARIEL	2014-09-16	COCINA	Borrado	Modificar
13	ANGELES Y DEMONIOS	DAN BROWN	ARIEL	2014-09-21	FICCION	Borrado	Modificar
14	FORTALEZA DIGITAL	DAN BROWN	ARIEL	2011-04-19	FICCION	Borrado	Modificar
15	CAPITAN ALATRISTE	ARTURO PEREZ REVERTE	ALFAGUARA	2001-04-17	HISTORICA	Borrado	Modificar
16	PIEL DE TAMBOR	ARTURO PEREZ REVERTE	ALFAGUARA	2009-04-14	HISTORICA	Borrado	Modificar
17	TIEMPOS DE COLERA	GABRIEL GARCIA	OVEJA NEGRA	2002-08-04	HISTORICA	Borrado	Modificar
18	NOTICIA DE UN SECUESTRO	GABRIEL GARCIA	ALFAGUARA	2013-05-13	INTRIGA	Borrado	Modificar

ID: TITULO: AUTOR:

EDITORIAL: FECHA: TEMATICA:

5. PRACTICA

Crearem un programa de gestió d'empleats molt senzill on depenent de la feina de l'empleat se li assignarà un salari automàticament. D'un treballador identifiquem el nom i la seva feina, estaria bé tenir un identificador únic per aquest treballador. Les feines són fixes, és a dir ja estan definits en un ENUMERABLE. Depenent de la feina s'assignarà un salari a l'empleat un cop es crea.

- Notes

- El servidor ha de tindre ben definits els dominis i repositoris
- El domini ha de tindre el CRUD al complet (Create, Read, Update, Delete), utilitzant els verbs HTTP associats.
- Crea una petició HTTP especial que busqui empleats per feina, a més de totes les que creen, llegeixen, actualitzen o esborren elements de tipus empleat
- Els objectes seran persistits únicament en memòria (per exemple amb un ArrayList)
- Has de tindre en compte les bones pràctiques de disseny de les API: utilitzi correctament els codis d'error i les respostes en cas d'invocacions incorrectes