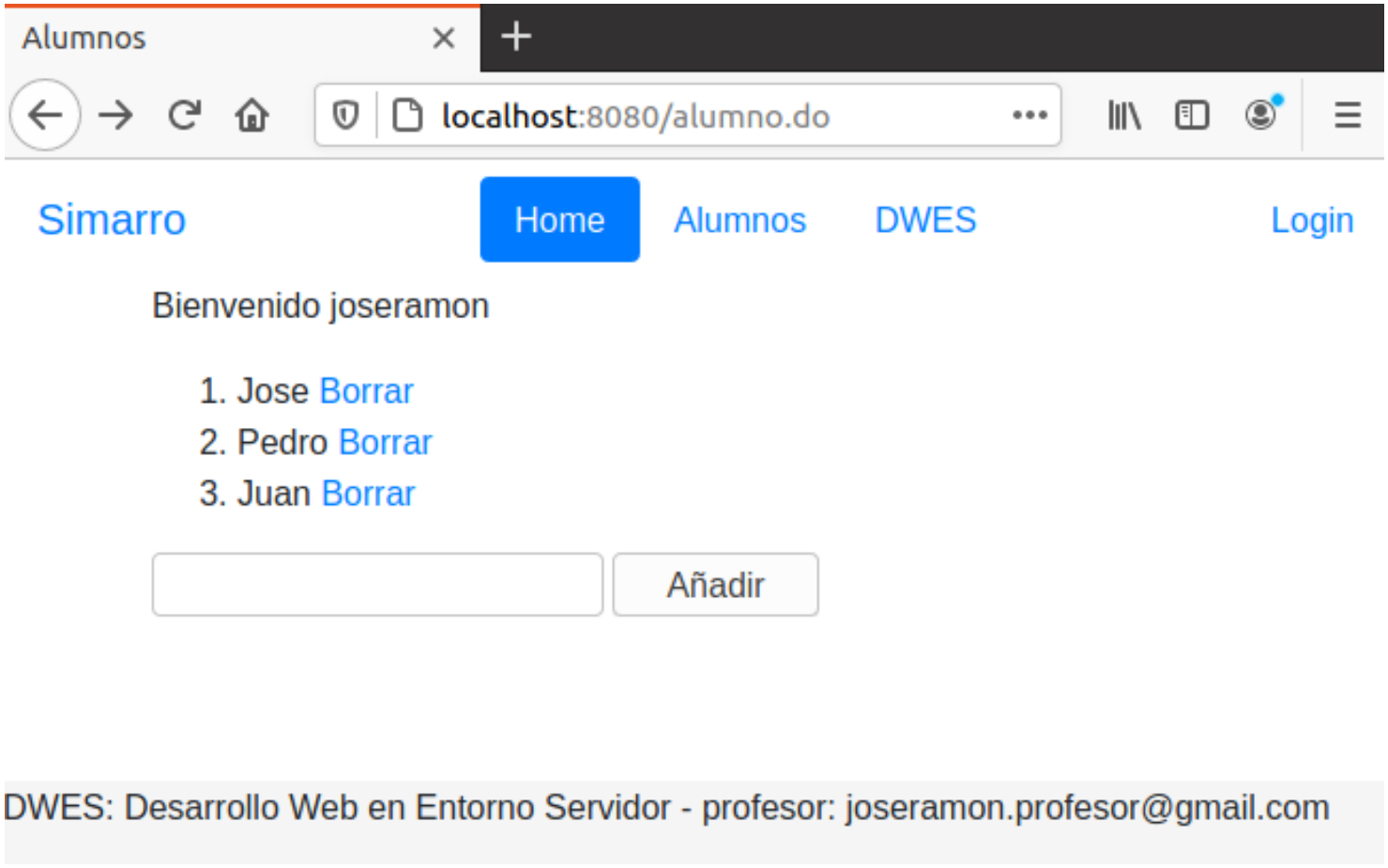
Objetivos de la sesión:

- ***Mejorar la presentación*** de la webapp utilizando ***Bootstrap***
- ***Controlar el login*** a la webapp
- ***Controlar el logout*** de la webapp

Ahora que ya tenemos una funcionalidad básica (listar, añadir y borrar) en nuestra aplicación es hora de darle un cambio estético utilizando Bootstrap.

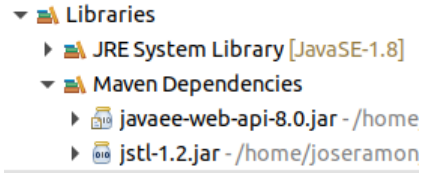


¿Como podemos utilizar Bootstrap en nuestro proyecto de una forma sencilla?

Bootstrap:

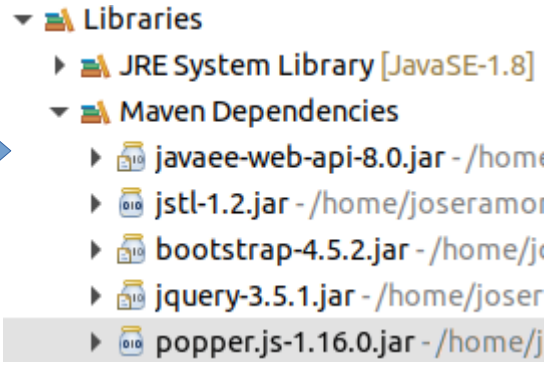
· La primera idea seria descargar manualmente el Boostrap, pero la ***mejor solución pasa por utilizar Maven*** para descargar la librería de Bootstrap como si una dependencia más se tratara.

· Si nos vamos a las librerias y dentro a las dependencias de Maven vemos que tenemos javaee-web-api-8.0.jar y jstl-1.2.jar. Vamos a añadir la dependencia bootstrap y ver que pasa:



mi-primera-webapp-jee/pom.xml

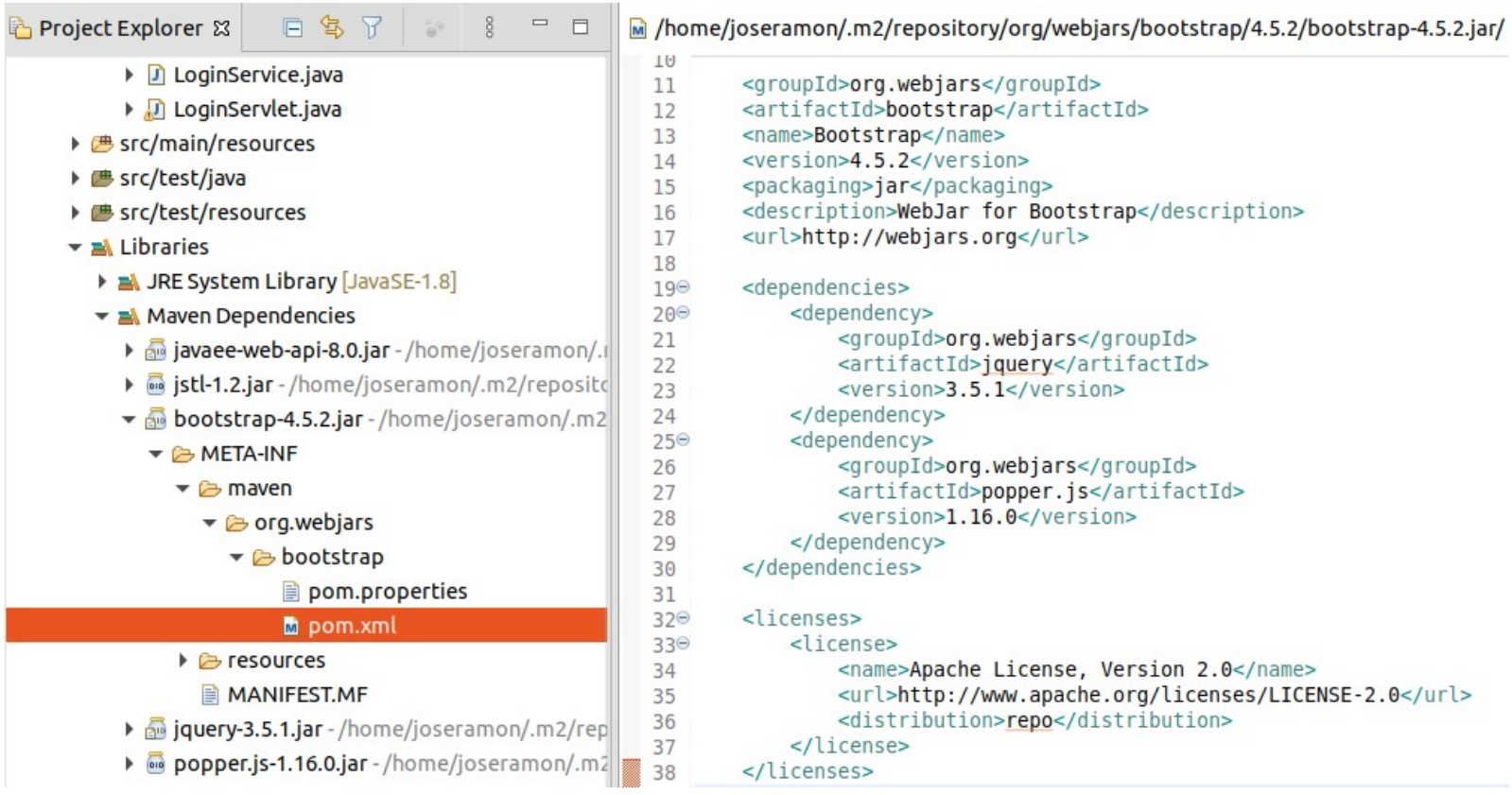
```
8    <dependencies>
9        <dependency>
10           <groupId>javax</groupId>
11           <artifactId>javaee-web-api</artifactId>
12           <version>8.0</version>
13           <scope>provided</scope>
14       </dependency>
15       <dependency>
16           <groupId>jstl</groupId>
17           <artifactId>jstl</artifactId>
18           <version>1.2</version>
19       </dependency>
20       <dependency>
21           <groupId>org.webjars</groupId>
22           <artifactId>bootstrap</artifactId>
23           <version>4.5.2</version>
24       </dependency>
25   </dependencies>
```



¿Porque se añade JQuery y Popper?

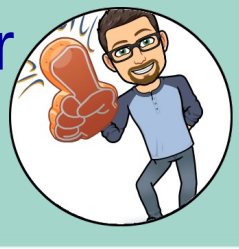
... continuación **Bootstrap:**

· ***La respuesta la tenemos dentro de la propia librería de bootstrap***. En el fichero pom.xml de la librería de bootstrap se establece que este artefacto tiene 2 dependencias y Maven las descarga para que Bootstrap funcione correctamente. Esa es la ventaja de Maven, no tienes que pararte a pensar si para que funcione Bootstrap te hace falta alguna otra librería, porque Maven lo hace por tí:



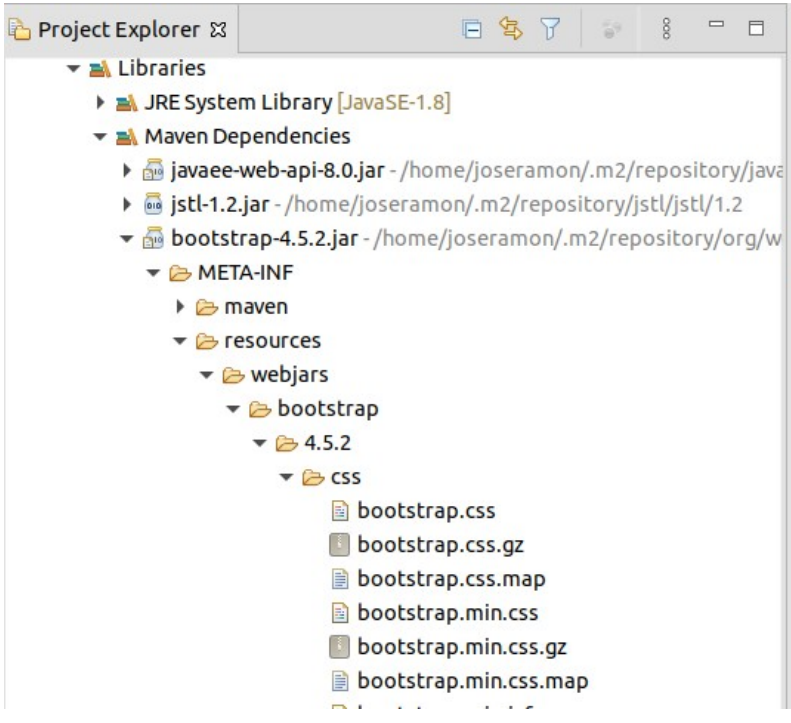
... continuación **Bootstrap:**

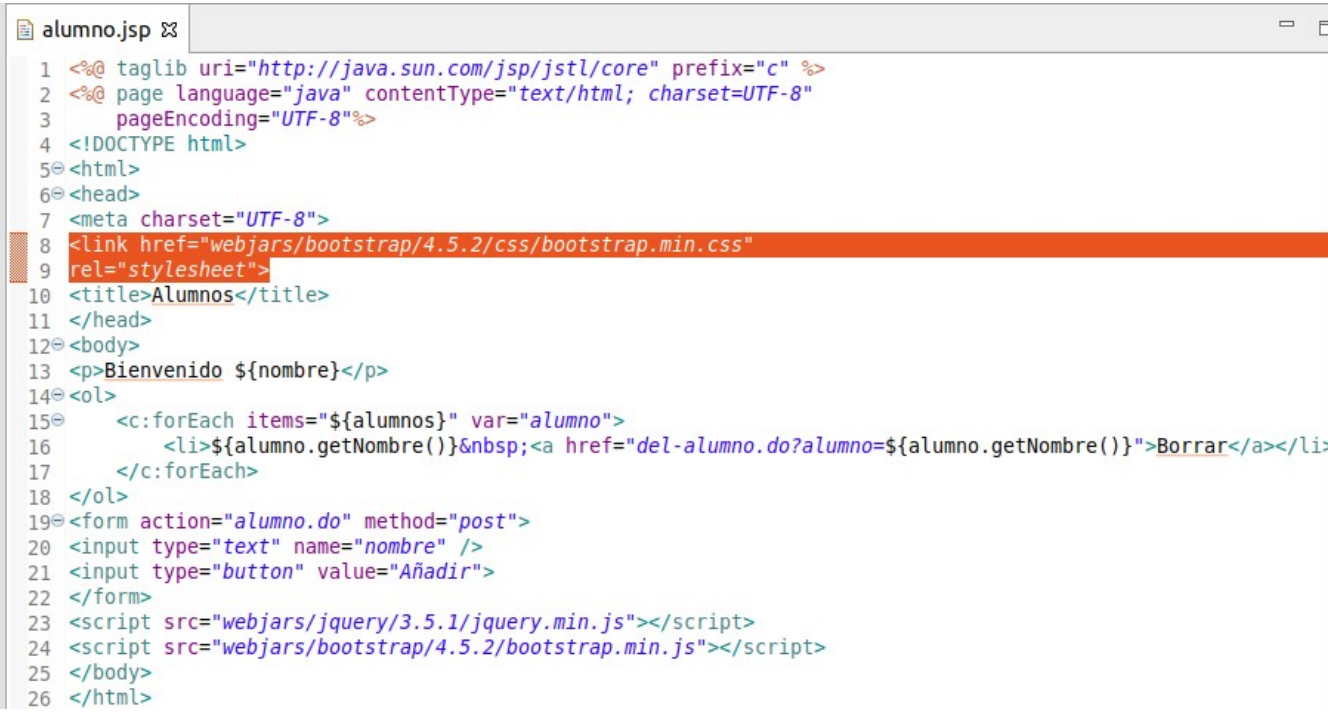
¿Como incorporamos la librería Bootstrap en nuestra página JSP?



... continuación **Bootstrap:**

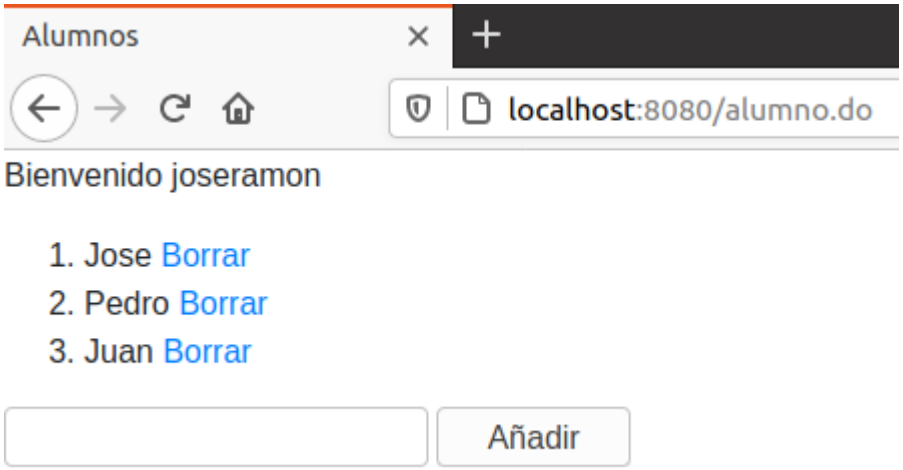
· ***La respuesta la tenemos otra vez dentro de la propia librería de bootstrap***. En la ***carpeta "resources" de la librería Bootstrap*** tenemos la ruta de los ficheros que necesitamos y en nuestra pagina alumno.jsp debemos insertar el link al css antes del "title" (linea 8) y debemos insertar los 2 scripts de javascript (jquery y bootstrap) antes de acabar el "body" (linea 23 y 24). Estos ficheros javascript sirven para que Bootstrap funcione correctamente. Para que funcione pararemos el servidor y recargaremos la web:





... continuación **Bootstrap:**

- Solo incorporando Bootstrap vemos que ya empieza a verse una diferencia:



Mejoremos realmente el aspecto de nuestra página web con la DWES_UD1_06_plantillaBootstrapSimarro.jsp disponible en el Drive. Descargate la plantilla. No vamos a explicar en detalle esta página porque se sale del alcance de este módulo.

... continuación **Bootstrap:**

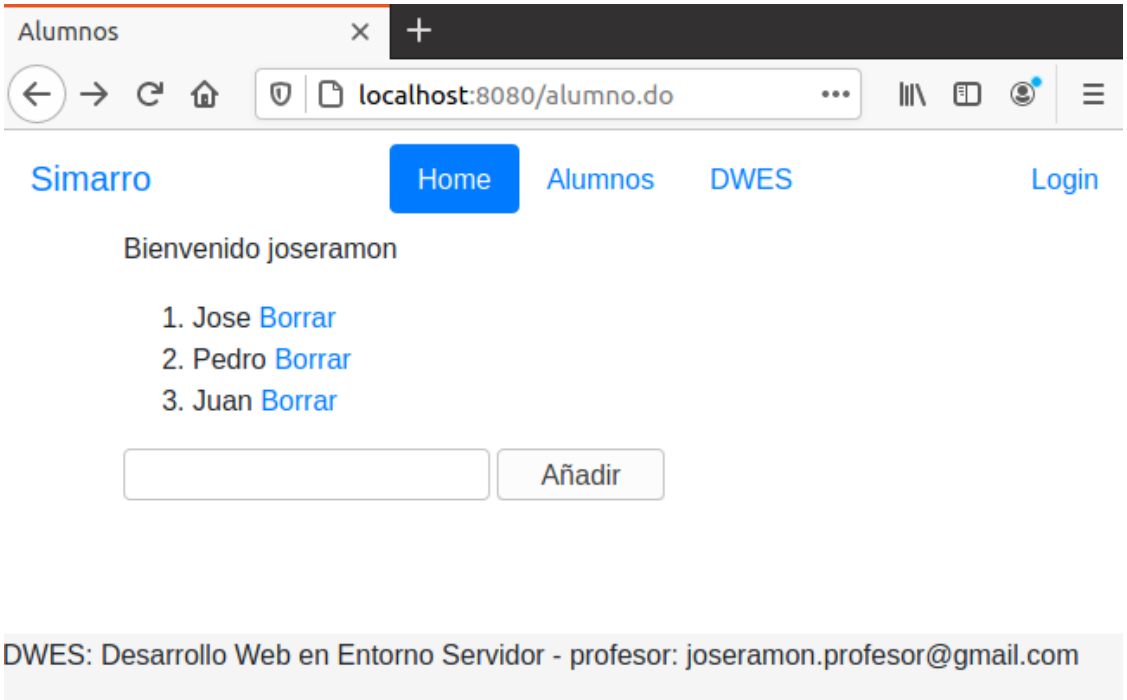
· Ahora vamos a cambiar la estética de alumno.jsp. Abrimos la plantilla y sustituimos el contenido del div “container” por el “body” de alumno.jsp. Después lo copiamos todo y actualizamos el contenido de alumno.jsp entero con la plantilla modificada como se muestra a continuación:

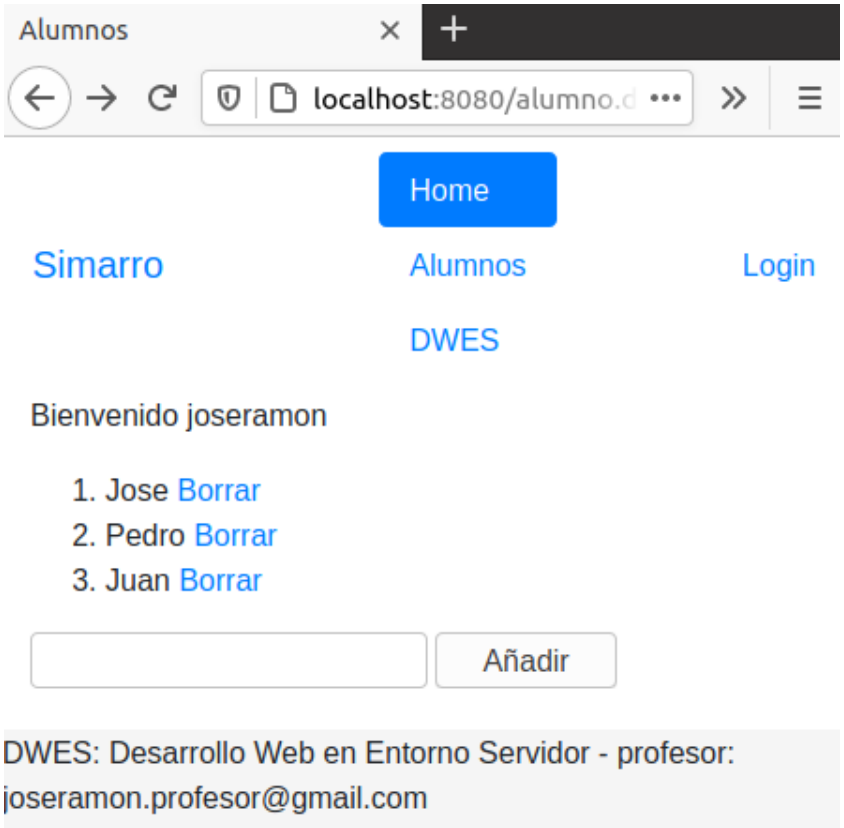
alumno.jsp

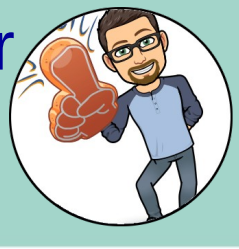
```
39
40          <ul class="nav navbar-nav navbar-right">
41              <li><a href="/login.do">Login</a></li>
42          </ul>
43
44      </nav>
45
46      <div class="container">
47          <p>Bienvenido ${nombre}</p>
48          <ol>
49              <c:forEach items="${alumnos}" var="alumno">
50                  <li>${alumno.getNombre()} <a href="del-alumno.do?alumno=${alumno.getNombre()}">Borrar</a></li>
51              </c:forEach>
52          </ol>
53          <form action="alumno.do" method="post">
54              <input type="text" name="nombre" />
55              <input type="button" value="Añadir">
56          </form>
57      </div>
58
59      <footer class="footer">
60          <p>DWES: Desarrollo Web en Entorno Servidor - profesor: joseramon.profesor@gmail.com</p>
61      </footer>
62
63      <script src="webjars/jquery/3.5.1/jquery.min.js"></script>
64      <script src="webjars/popper.js/1.16.0/umd/popper.min.js"></script>
65      <script src="webjars/bootstrap/4.5.2/js/bootstrap.min.js"></script>
66
67  </body>
```

... continuación Bootstrap:

· Si ahora actualizamos el navegador ya empezamos a ver cambios. Si reducimos el ancho del navegador veremos que tenemos un menú responsive (cambia según el ancho disponible) gracias a Bootstrap. No vamos a entrar en detalles de como se consigue porque excede el contenido de este módulo.





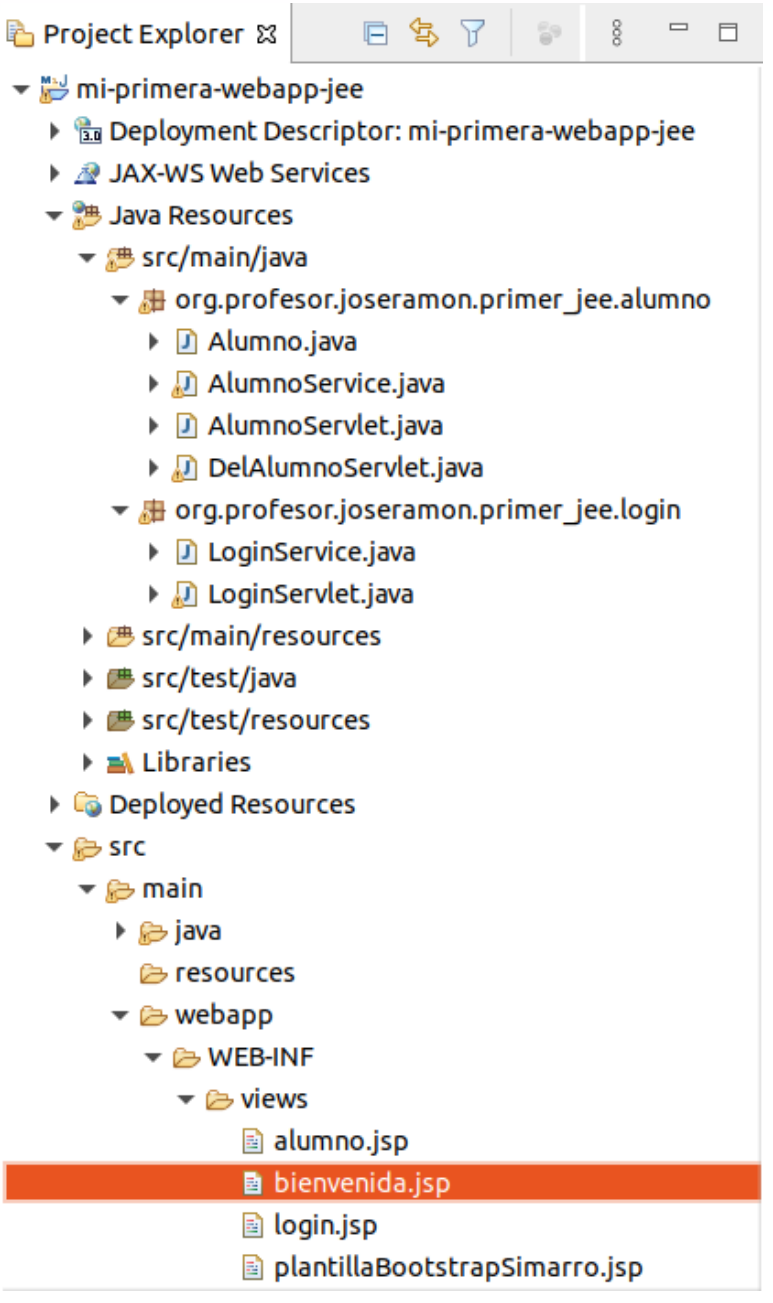


Refactorizar:

Si nos fijamos en la estructura de nuestra aplicación hay algunas cosas mejorables que deberiamos cambiar:

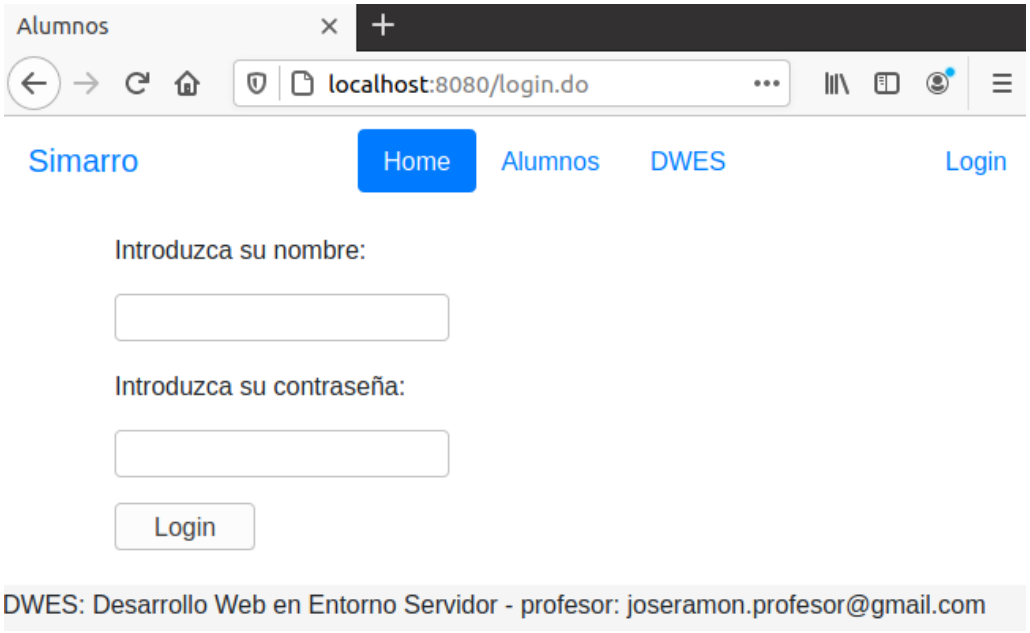
· "bienvenida.jsp" no se esta utilizando, por lo que debemos borrar el fichero bienvenida.jsp.

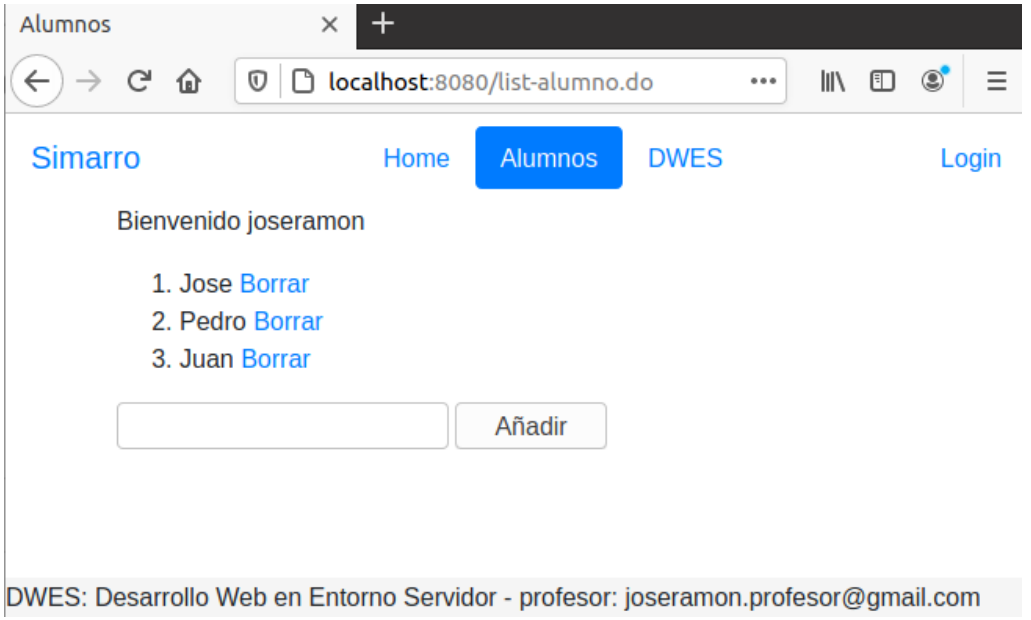
· Tenemos "AlumnoServlet" con la funcionalidad de listar y añadir "Alumnos" y "DelAlumnoServlet" con la funcionalidad de borrar. Debemos dividir "AlumnoServlet" en 2. Para ello renombraremos "AlumnoServlet" como "ListAlumnoServlet" y crearemos una copia "AddAlumnoServlet". Un servlet deberá contener solo el método para listar (llamando a "list-alumno.do") y el otro para añadir "Alumnos" (llamando a "add-alumno.do"). Igual hay que tocar algó más en la aplicación para que funcione el añadir,borrar y listar ...



... continuación Refactorizar:

· Ahora debemos modificar “login.jsp” para que utilice la plantilla de Bootstrap y debemos modificar “alumno.jsp” para que el elemento del menu activo no sea “Home” sino “Alumnos” (“Alumnos” debe quedar en azul como quedaba “Home”) :





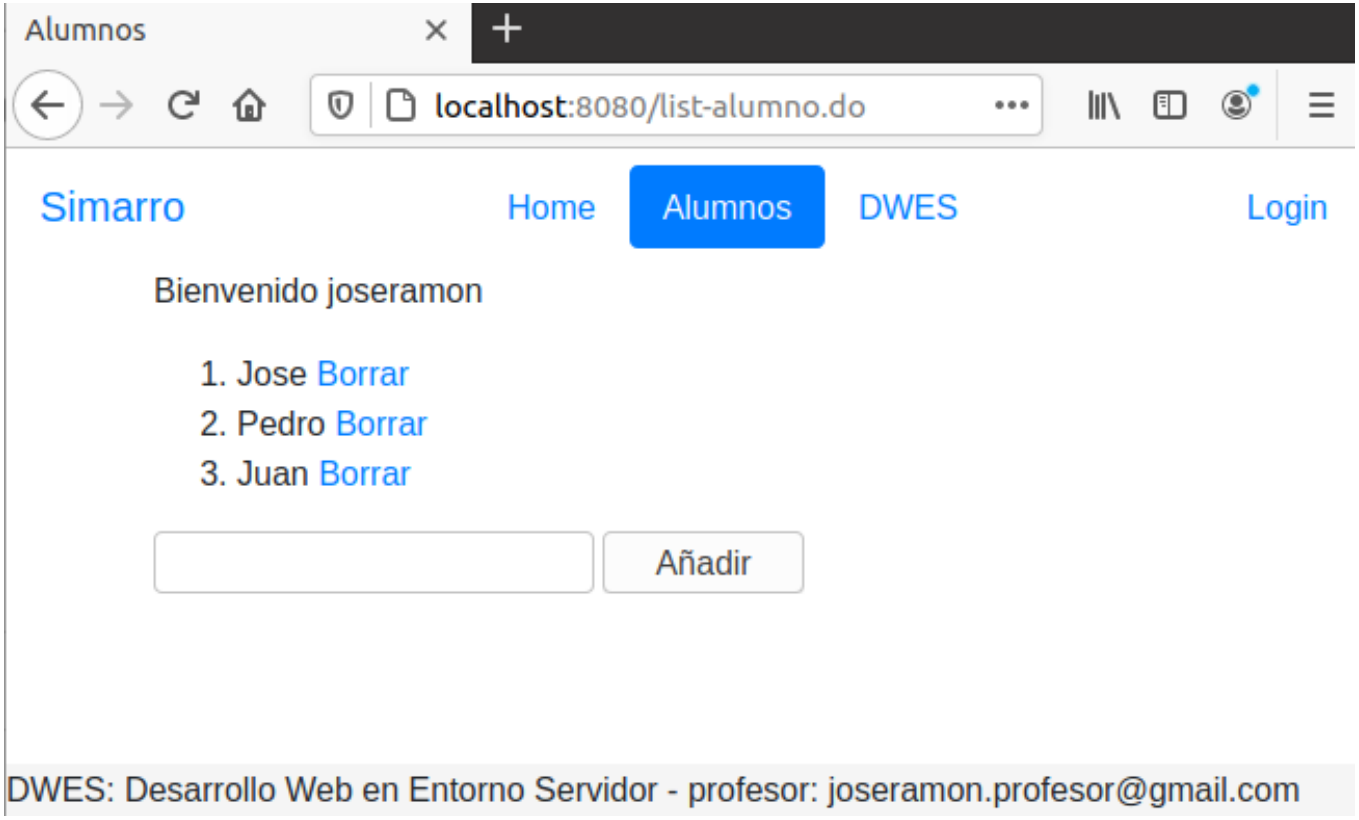
Filtros: Login

· Hasta ahora hemos asumido que el usuario entra siempre logeandose primero, pero …

¿Que pasa si un usuario teclea en el navegador list-alumno.do directamente sin logearse?

¿Que debemos hacer para comprobar si un usuario está logeado?

¿Donde tenemos que realizar estas comprobaciones?



... continuación Filtros: Login

· Cuando un usuario introduce sus credenciales correctamente en LoginServlet (doPost()) se almacena su nombre en la sesión. Este puede ser un buen sistema (el nombre esta definido o no) para comprobar si el usuario esta logeado o no .

LoginServlet.java

```
42              HttpServletResponse response) throws IOException, ServletExce
43          request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(
44      }
45      @Override
46      protected void doPost(HttpServletRequest request,
47              HttpServletResponse response) throws IOException, ServletExce
48
49          String nombre =request.getParameter("nombre");
50          String password =request.getParameter("password");
51
52          if (loginServicio.usuarioValido(nombre, password)) {
53              request.getSession().setAttribute("nombre", nombre);
54              //validación correcta: Redirigir al Servlet de alumno
55              response.sendRedirect("list-alumno.do");
56          } else {
57              //validación incorrecta
58              request.setAttribute("errores", "Usuario '" + nombre + "' o c
59              request.getRequestDispatcher("/WEB-INF/views/login.jsp").forw
60          }
61      }
62  }
63
```

¿Tenemos que comprobarlo en todos los servlets ? O ¿Existe algún sistema mejor?

... continuación Filtros: Login

· JEE8 permite interceptar las llamadas a los Servlets con una clase que herede de ***Filter***. Esta subclase puede servir para filtrar todas las peticiones (filtro /*) o solo las que cumplen ciertos criterios (filtro /*.do).

· El ***algoritmo*** que deberemos seguir será:

Si hay un nombre en la sesión entonces
Permitimos la petición
sino
Redirigimos a login.do
FinSi

Veamos en las siguientes diapositivas como implementar este filtro!!

... continuación Filtros: Login

· Para implementar este sistema de control de usuarios creamos una nueva clase "ComprobarLoginFilter" que implemente la clase "Filter" de JEE8 (javax.servlet.Filter) en org.alumno.NombreAlumno.primer_jee.filtros. Para crear el método doFilter() pasamos el ratón por encima de la clase y nos sugiere "crear los métodos no implementados" ("add unimplemented methods").

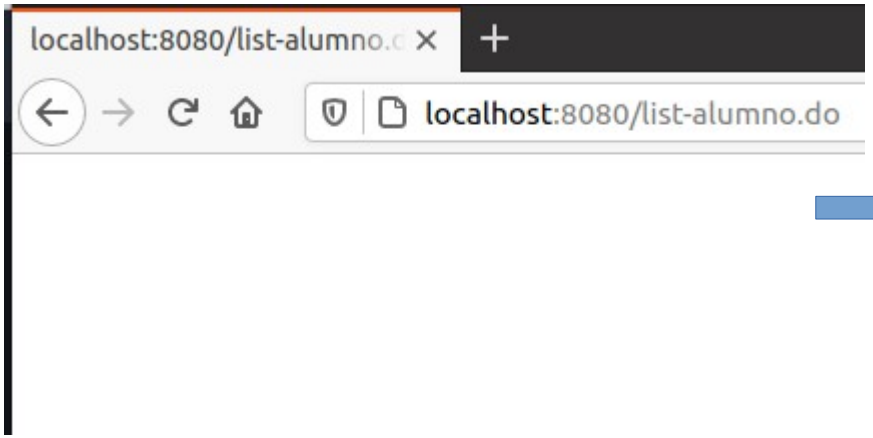
Debemos añadir la notación WebFilter para filtrar solo las llamadas a servlets "*.do":

ComprobarLoginFilter.java

```
 1 package org.profesor.joseramon.primer_jee.filtros;
 2 import java.io.IOException;
12
13 @WebFilter(urlPatterns="*.do")
14 public class ComprobarLoginFilter implements Filter {
15
16     @Override
17     public void init(FilterConfig filterConfig) throws ServletException {}
18     @Override
19     public void doFilter(ServletRequest servletRequest,
20             ServletResponse servletResponse, FilterChain chain)
21             throws IOException, ServletException {
22
23         HttpServletRequest request= (HttpServletRequest) servletRequest;
24         System.out.println(request.getRequestURI());
25     }
26     @Override
27     public void destroy() {}
28
29 }
```

... continuación Filtros: Login

¿Porque imprime la petición que se ha hecho en la consola pero no se muestra nada en el navegador?



ComprobarLoginFilter.java

```
1  package org.profesor.joseramon.primer_jee.filtros;
2  import java.io.IOException;
12
13 @WebFilter(urlPatterns="*.do")
14 public class ComprobarLoginFilter implements Filter {
15
16     @Override
17     public void init(FilterConfig filterConfig) throws ServletEx
18     @Override
19     public void doFilter(ServletRequest servletRequest,
20             ServletResponse servletResponse, FilterChain chain)
21             throws IOException, ServletException {
22
23         HttpServletRequest request= (HttpServletRequest) servlet
24         System.out.println(request.getRequestURI());
25     }
26     @Override
27     public void destroy() {}
28
29 }
```

Markers | Properties | Servers | Data Source Explorer | Snippets

```
mi-primera-webapp-jee [Maven Build] /usr/lib/jvm/java-13-openjdk-amd64/bin/java
INFORMACIÓN: Starting ProtocolHandler ["http-bio-8080"]
/list-alumno.do
```

... continuación Filtros: Login

· Sin añadir la linea “chain.doFilter()” la petición se para sin reenviar la petición al servlet. Debemos añadir la linea 25 para que se reenvie la petición al servlet:

ComprobarLoginFilter.java

```
1  package org.profesor.joseramon.primer_jee.filtros;
2  import java.io.IOException;
12
13 @WebFilter(urlPatterns="*.do")
14 public class ComprobarLoginFilter implements Filter {
15
16     @Override
17     public void init(FilterConfig filterConfig) throws ServletException {}
18     @Override
19     public void doFilter(ServletRequest servletRequest,
20             ServletResponse servletResponse, FilterChain chain)
21             throws IOException, ServletException {
22
23         HttpServletRequest request= (HttpServletRequest) servletRequest;
24         System.out.println(request.getRequestURI());
25         chain.doFilter(request, servletResponse);
26     }
27     @Override
28     public void destroy() {}
29
30 }
```

Comprobemos que ya funciona.

... continuación Filtros: Login

· Ahora nos queda modificar el método para que siga el siguiente algoritmo y realice la comprobación:

Si hay un nombre en la sesión entonces
Permitimos la petición
sino
Redirigimos a login.do
FinSi

Pero realmente es un poco más complejo. Nos podemos encontrar la situación de que no estamos logeados, pero estamos en la página de login, con lo cual no debemos reenviar al login porque entrariamos en un bucle infinito. Para solucionarlo modificaremos la lógica en la sección del “sino” como veremos a continuación.

... continuación Filtros: Login

- Realiza los siguientes cambios:

```
public void doFilter(ServletRequest servletRequest,
        ServletResponse servletResponse, FilterChain chain)
        throws IOException, ServletException {

    HttpServletRequest request= (HttpServletRequest) servletRequest;
    System.out.println(request.getRequestURI());
    if (request.getSession().getAttribute("nombre")!=null)
        //Estamos logeados -> Dejamos pasar
        chain.doFilter(request, servletResponse);
    else //No estamos logeados
        if (request.getRequestURI().contains("login.do")) {
            //No estamos logeados pero estamos en la página de login-> Dejamos pasar
            chain.doFilter(request, servletResponse);
        } else {
            //No estamos logeados y debemos reenviar al login
            //request.getRequestDispatcher("login.do").forward(request, servletResponse);
            HttpServletResponse response = (HttpServletResponse) servletResponse;
            response.sendRedirect("login.do");
        }
}
```

Aunque cerremos el navegador, si no ha pasado el timeout podemos tener la sesión abierta. Para asegurarnos que la sesión esta cerrada deberemos parar el Tomcat y volverlo a poner en marcha.

Paremos el servidor y volvamos a ponerlo en marcha. Comprobemos que aunque llamamos a “list-alumno.do” directamente primero nos pide login.

Logout:

· Ahora que ya tenemos una funcionalidad básica (listar, añadir y borrar) en nuestra aplicación y controlamos que todo usuario que realiza una petición esta logeado es hora de implementar la funcionalidad “Logout” para poder salir de la aplicación y cerrar la sesión. Hasta ahora cuando estabamos en el listado de alumnos y pulsabamos “Login” nos ibamos a la página de login, pero no cerrábamos la sesión.

· Aunque Tomcat tiene un timeout que cierra las sesiones que no se están utilizando, si cerramos la sesión en cuanto no queremos continuar en la aplicación web ahorraremos recursos que podría necesitar el servidor en periodos de alta congestión.

· Para cerrar la sesión utilizaremos ***“request.getSession().invalidate()”*** como veremos en la siguiente diapositiva.

Logout:

Para ello hay que copiar LoginServlet como LogoutServlet cuya url será "logout.do", dejar solo el método doGet con 2 sentencias, la invalidación de la sesión y la redirección a login.jsp. Habrá que cambiar alumno.jsp para que en vez de "Login" se muestre un "Logout" que redireccione al servlet LogoutServlet.

```
@WebServlet(urlPatterns = "/logout.do")
public class LogoutServlet extends HttpServlet {
    LoginService loginServicio = new LoginService();
    AlumnoService alumnoServicio= new AlumnoService();

    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws IOException, ServletException {
        request.getSession().invalidate();
        //request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);
        response.sendRedirect("login.do");
    }
}
```

Adicionalmente debemos actualizar alumno.jsp para poder ejecutar el logout:

```
34      <nav class="nav nav-pills flex-column flex-sm-row">
35          <a class="nav-link" href="login.do">Home</a>
36          <a class="nav-link active" href="/list-alumno.do">Alumnos</a>
37          <a class="nav-link" href="https://aules.edu.gva.es/fp/course/view.php?id=60536">DWES</a>
38      </nav>
39
40      <ul class="nav navbar-nav navbar-right">
41          <li><a href="logout.do">Logout</a></li>
42      </ul>
43
44  </nav>
```

EJERCICIO:

Sigue todos los pasos de los PDF y 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 al fichero comprimido UD1_practica6_nombreAlumno.tar.gz donde nombreAlumno es el nombre del alumno que entrega la práctica.

3º Súbela al moodle.

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