UD 2: Modelo Vista Controlador
1.- Introducción

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

Objetivos de la sesión:

- *Entender la arquitectura web basada en el patrón de diseño **Modelo Vista Controlador (MVC)**.*

- *Entender el **patrón de diseño modificado "Front Controller"** de **Spring MVC.***

- *Crear nuestra **primera aplicación Spring MVC**:*
  - *Crear un nuevo proyecto con las dependencias y configuración de Spring MVC*
  - *Configurar el DispatcherServlet que actua de "Front Controller"*
  - *Configurar el controlador de login para atender la petición del navegador a la url de login.*
  - *Configurar la vista ("View Resolver") para poder mostrar una página JSP con los datos del modelo (ModelMap).*
  - *Inyectar como dependencias los servicios que utilizará el controlador.*

Arquitectura web: Modelo 1:

· Hace 15-20 años ya existían diversas tecnologías para desarrollar páginas web (jsp (hechas en Java), php , asp (hechas en Visual Basic Script) , pero en todas esas tecnologías ***las páginas (jsp,php,asp,...) se encargaban de consultar los datos, implementar la lógica de negocio y presentar los datos al usuario***:

¡La gran desventaja de esta arquitectura era la gran dificultad de mantenimiento !

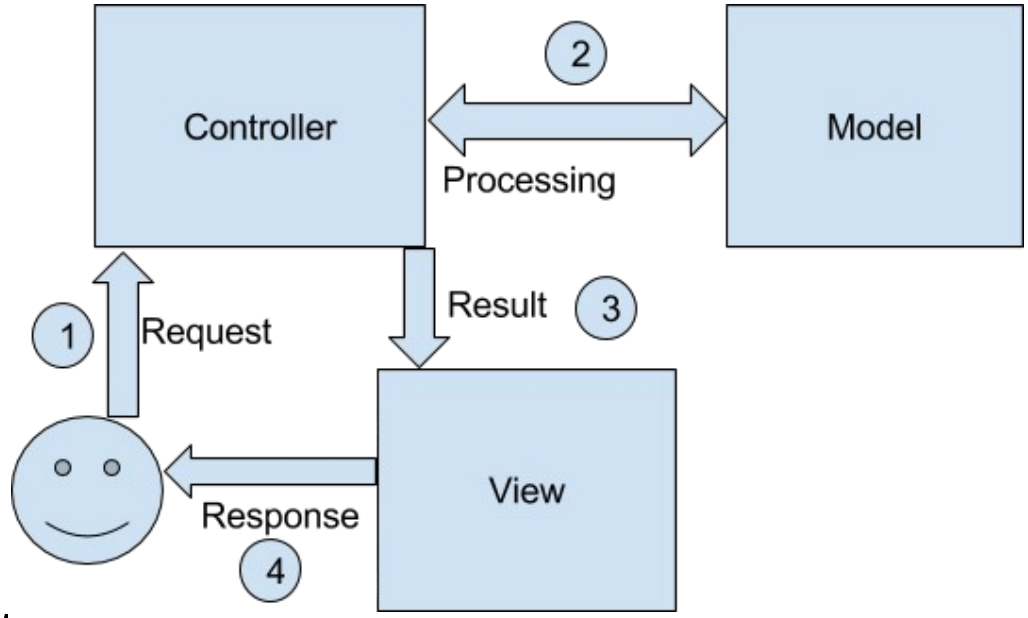
Arquitectura web: Modelo 2: Patrón de diseño MVC

· El patrón de diseño Modelo Vista Controlador (Model View Controller) nos permite ***dividir una aplicación en unidades o capas independientes con una funcionalidad concreta,*** lo cual nos permite cumplir el ***principio de separación de responsabilidades***:

Modelo:
Alumno.java, Modulo.java, LogError.java

Vista:
login.jsp, list-alumno.jsp, add-alumno.jsp

Controlador:
LoginServlet.java,AddAlumnoServlet.java



· El **modelo** es transferido de una capa a otra
· La **vista** es responsable de mostrar los datos de la aplicación al usuario
· El **controlador** es responsable de aceptar una petición del usuario, consultar o modificar el modelo que contiene los datos (utilizando servicios como AlumnoService.java) y reenviar el modelo a la vista.

... continuación Arquitectura web: Modelo 2: Patrón de diseño MVC

Al utilizar el patrón de diseño MVC obtenemos varias **ventajas**:

· Desarrollo de aplicaciones más rápidas:
Si definimos la capa del modelo por adelantado y pasamos esta información al equipo de desarrollo de la interfaz gráfica (UI), pueden empezar haciendo las vistas mientras que el equipo de Backend desarrollan los controladores y los servicios.

· Mayor flexibilidad ante cambios:
Si cambiamos el mecanismo utilizado por la vista, para el controlador este cambio es transparente. El controlador solo conoce el nombre de una vista lógica. Por lo tanto , es muy fácil cambiar a una vista basada en velocity o freemaker sin afectar al controlador.

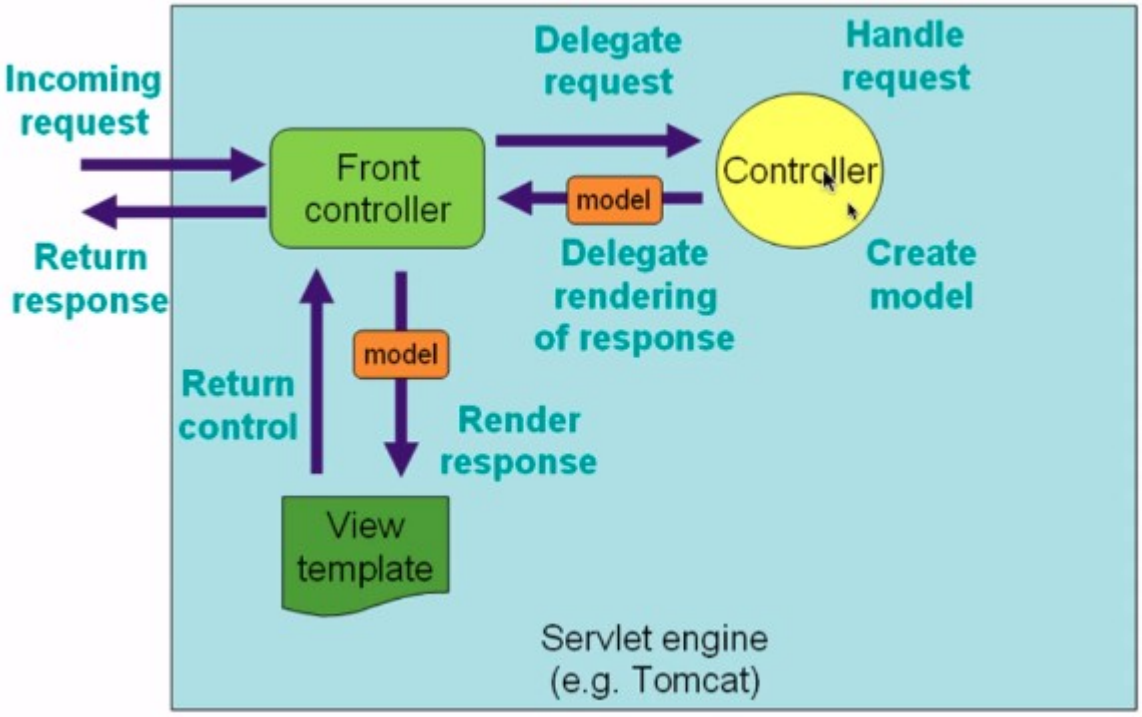
· Reducción de tareas de mantenimiento por el bajo acoplamiento:
La capa de vista es independiente de la capa de servicio. Aunque la capa del controlador depende de la capa de servicio, si se utilizan interfaces tampoco depende de la implementación concreta. Este hecho implica que no hace falta realizar tantos cambios cuando modificamos algo. En el modelo 1 los cambios implicaría realizar tareas de mantenimiento en más de un lugar.

UD 2: Modelo Vista Controlador
1.- Introducción

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

Arquitectura web: Modelo 2 modificado : Patrón de diseño Front Controller

Spring MVC tiene una arquitectura basada en el patrón de diseño MVC , pero algo modificado:



En vez de recibir la petición el controlador correspondiente, **todas las peticiones son preprocesadas por un Servlet especial (el dispatcher Servlet)** que centraliza las peticiones, analiza a quien va dirigida la petición y reenvía dicha petición al controlador correspondiente.

Mi primera aplicación con Spring MVC:

Vamos a crear una nueva aplicación para utilizar el patrón de diseño MVC con los siguientes **10 pasos**:

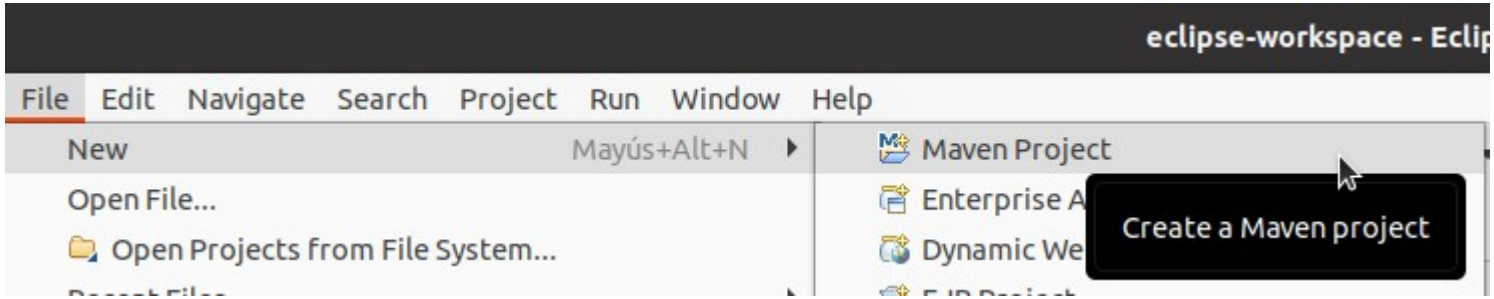
1º Creamos un nuevo proyecto maven sin arquetipo
2º Rellenar los datos del nuevo artefacto
3º Actualizar el fichero /pom.xml
4º Forzamos la descarga de las nuevas dependencias (incluida spring mvc)
5º Configurar el dispatcher servlet en src\main\webapp\WEB-INF\web.xml
6º Crear el fichero de configuración del contexto de Spring:
src\main\webapp\WEB-INF\alumno-servlet.xml
7º Crear las clases Java y los ficheros JSP para hacer el login inicial
8º Comprobar que el login funciona
9º Crear nuestro primer controlador Spring MVC : LoginController.java
10º Comprobar que la web nos da la bienvenida

UD 2: Modelo Vista Controlador
2.- Spring MVC: Controlador

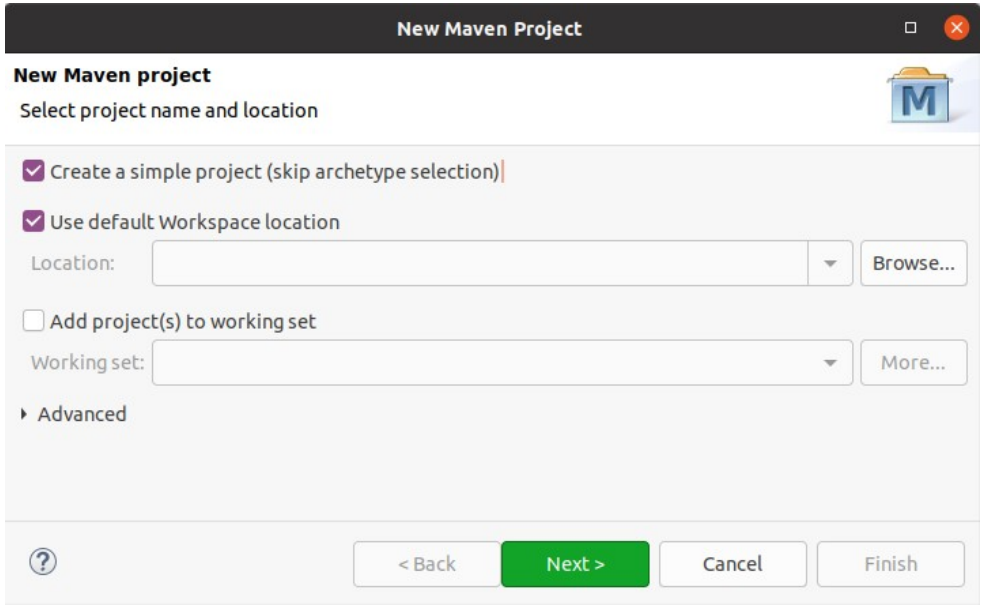
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación Mi primera aplicación con Spring MVC:

1º Creamos un nuevo proyecto maven: File \ New \ Maven Project



Le indicamos que queremos un proyecto simple (Create a simple project) que no tenga un arquetipo concreto.



... continuación Mi primera aplicación con Spring MVC:

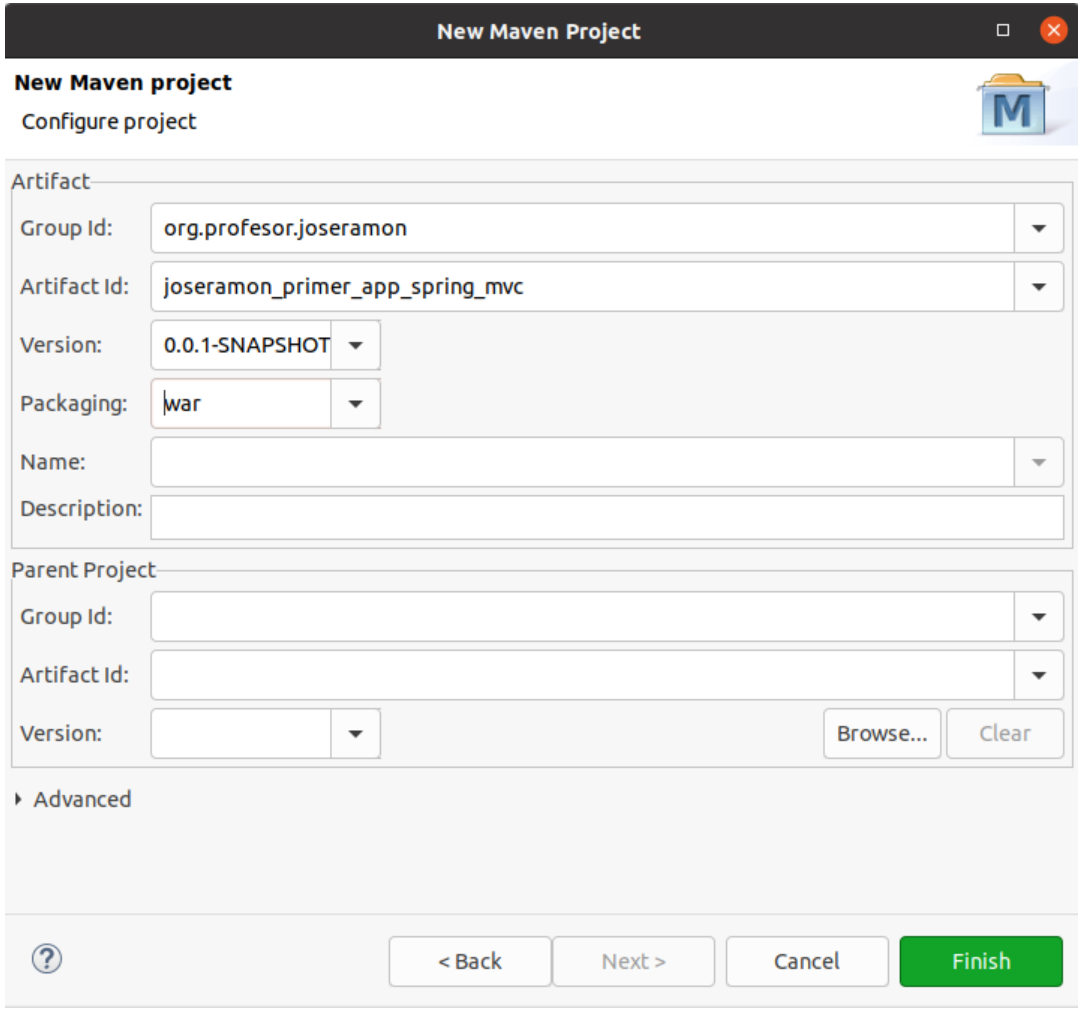
2º Rellenamos los datos del nuevo artefacto:

Es importante **rellenar exactamente con los siguientes datos** para que el profesor pueda cargar correctamente el proyecto de todos los alumnos para su revisión:

Group Id:
org.alumno.nomAlumno
(en minusculas todo)

Artifact Id:
nomAlumno_primer_app_spring_mvc

Packing:
war



... continuación Mi primera aplicación con Spring MVC:

3º Actualizar el fichero /pom.xml:

Vamos a modificar el fichero pom.xml para dejarlo como en nuestro proyecto de la UD1 copiando todo el contenido por debajo de packaging (dependencias y plugins) : Es normal que “packaging” esté en rojo si no tenemos WEB-INF\web.xml

joseramon_primer_app_spring_mvc/pom.xml

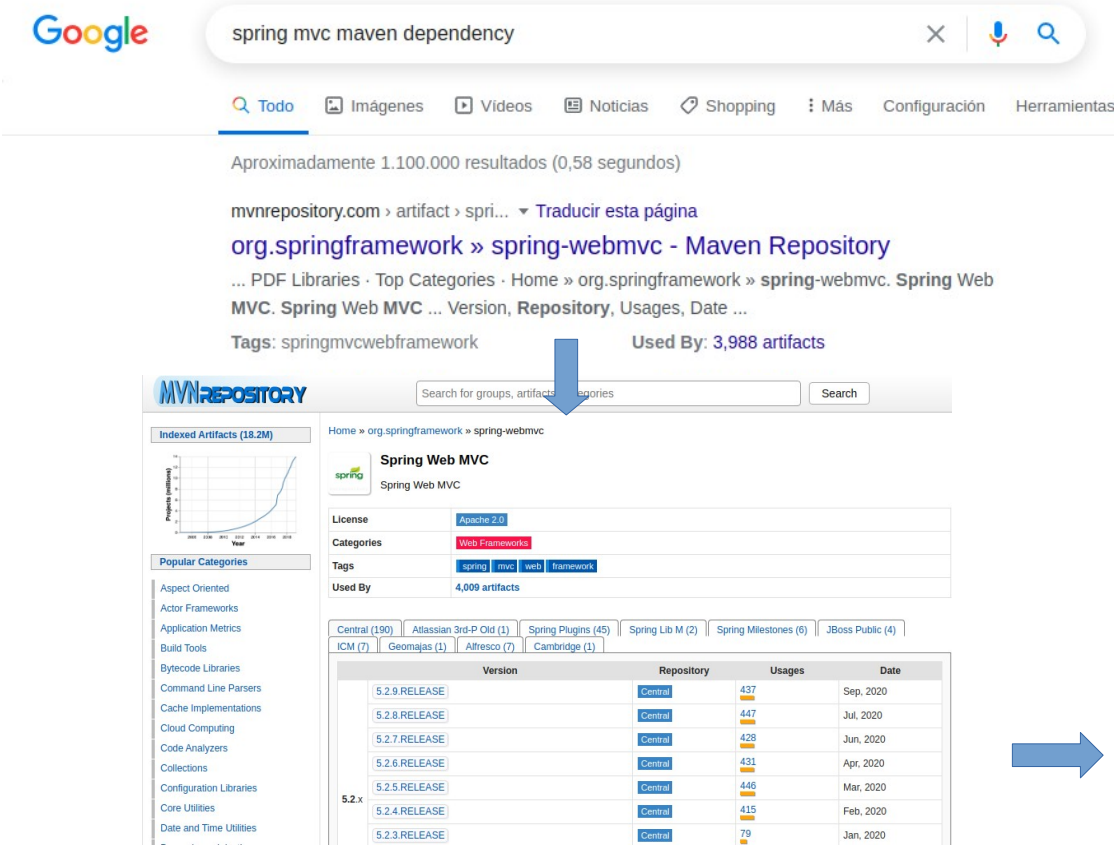
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xs
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.profesor.joseramon</groupId>
  <artifactId>joseramon_primer_app_spring_mvc</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>8.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>bootstrap</artifactId>
            <version>4.5.2</version>
        </dependency>
    </dependencies>
    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                    <configuration>
                        <verbose>true</verbose>
                        <source>1.8</source>
                        <target>1.8</target>
                        <showWarnings>true</showWarnings>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.tomcat.maven</groupId>
                    <artifactId>tomcat7-maven-plugin</artifactId>
                    <version>2.2</version>
                    <configuration>
                        <path>/</path>
                        <contextReloadable>true</contextReloadable>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

... continuación Mi primera aplicación con Spring MVC:

... continuación 3º Actualizar el fichero /pom.xml:

Ahora vamos a añadir la dependencia de "spring MVC" al fichero pom.xml:

Para ello podemos buscar cual es la última dependencia de spring MVC de Maven en Google:

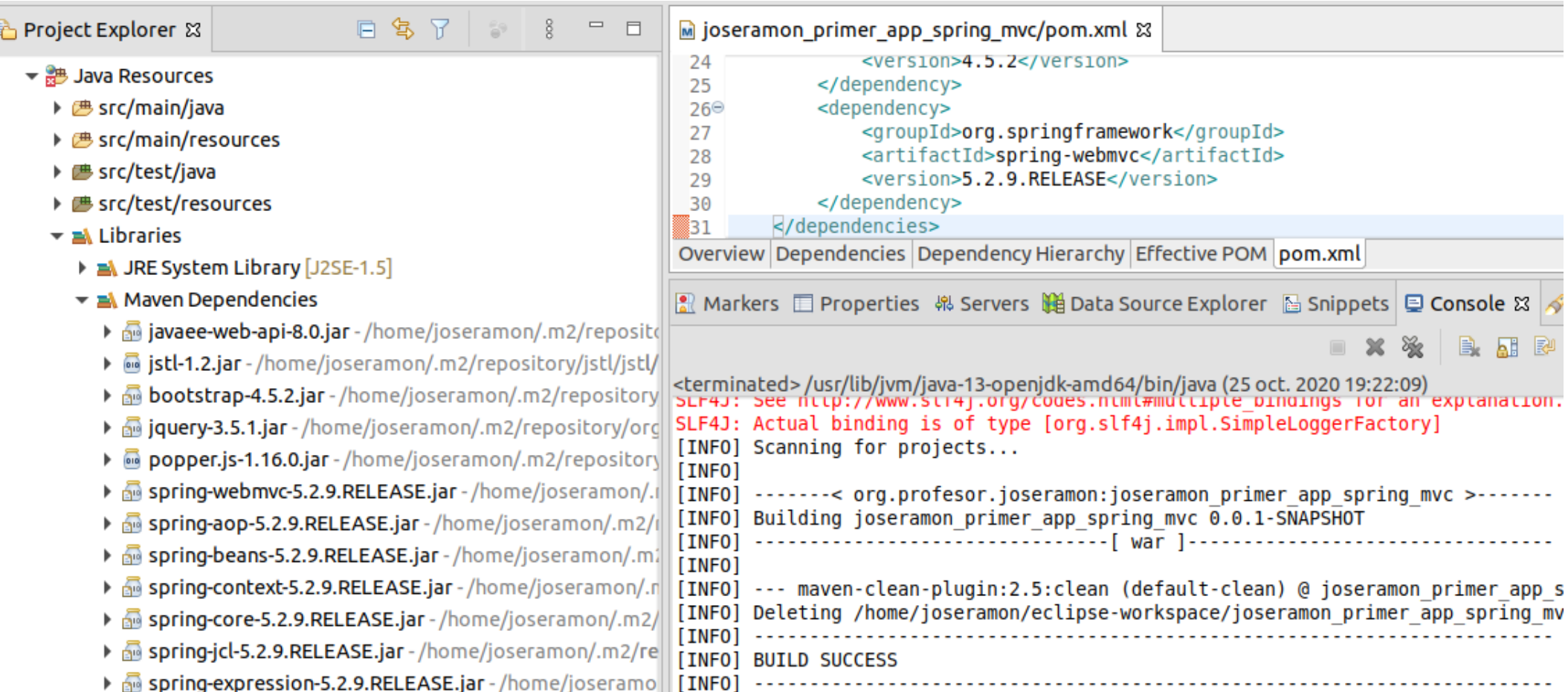


joseramon_primer_app_spring_mvc/pom.xml

```
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>8.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>4.5.2</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.2.9.RELEASE</version>
    </dependency>
</dependencies>
```

... continuación Mi primera aplicación con Spring MVC:

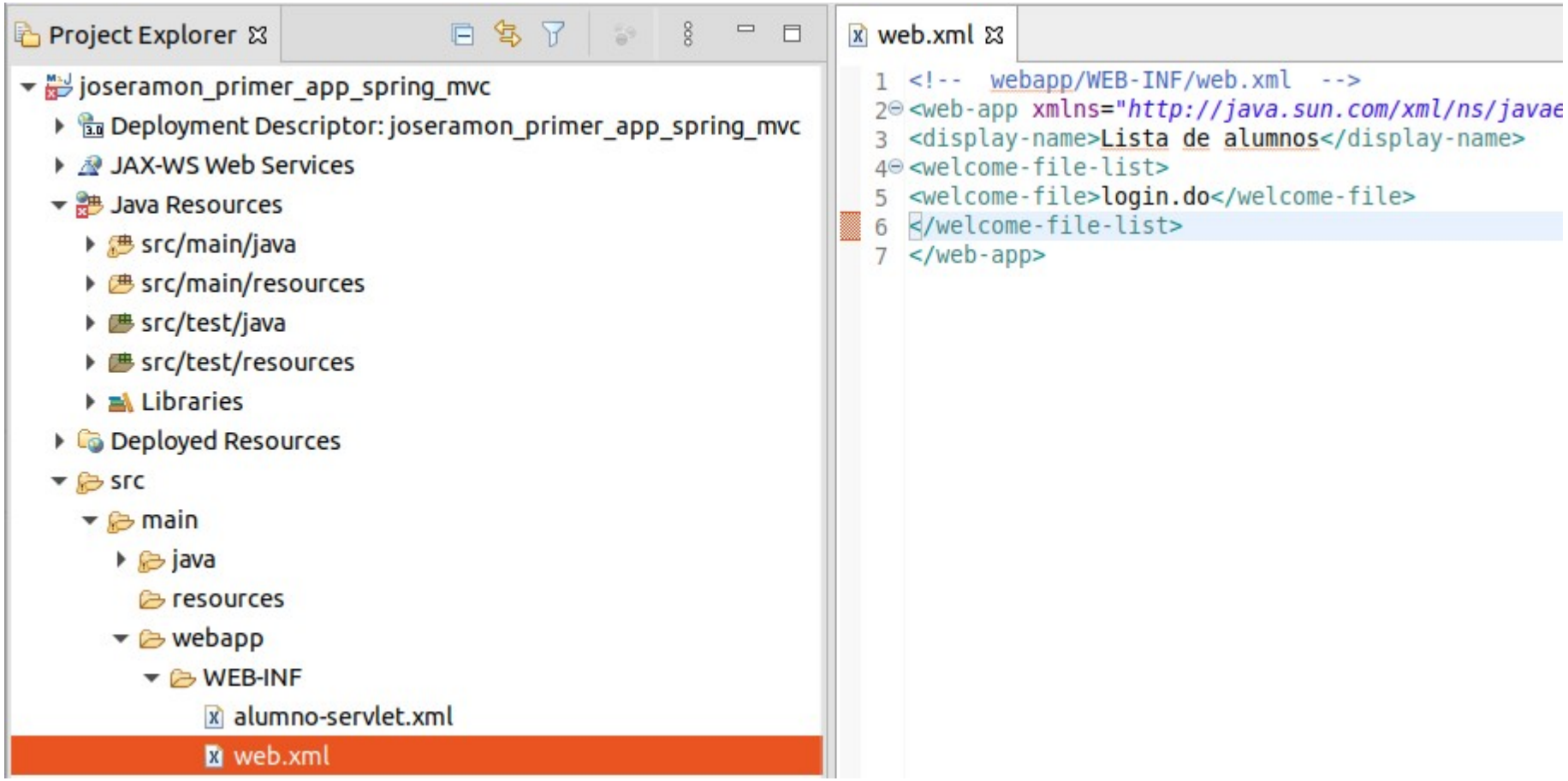
4º Forzamos la descarga de las nuevas dependencias:
Para ello ejecutamos Run As\Maven clean y comprobamos en "Library\ Maven Dependencies" si nos hemos descargado las librerías de spring :



... continuación Mi primera aplicación con Spring MVC:

5º Crear y modificar src\main\webapp\WEB-INF\web.xml:

Si copiamos el fichero web.xml del proyecto de la UD1 obtenemos el siguiente fichero:



... continuación Mi primera aplicación con Spring MVC:

... continuación 5º Crear y modificar src\main\webapp\WEB-INF\web.xml:

Ahora ya podemos añadir el Dispacher Servlet añadiendo el siguiente código a web.xml antes de cerrar el tag <web-app> :

```
<servlet>
	<servlet-name>dispatcher</servlet-name>
	<servlet-class>
	    org.springframework.web.servlet.DispatcherServlet
	</servlet-class>
	<init-param>
	   <param-name>contextConfigLocation</param-name>
	   <param-value>/WEB-INF/alumno-servlet.xml</param-value>
	</init-param>
	    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
	<servlet-name>dispatcher</servlet-name>
	<url-pattern>/spring-mvc/*</url-pattern>
</servlet-mapping>
```

... continuación Mi primera aplicación con Spring MVC:

... continuación 5º Crear y modificar src\main\webapp\WEB-INF\web.xml:

El fichero web.xml debería quedar de la siguiente manera:

web.xml

```
1  <!--  webapp/WEB-INF/web.xml  -->
2  <web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http:
3  <display-name>Lista de alumnos</display-name>
4  <welcome-file-list>
5  <welcome-file>login.do</welcome-file>
6  </welcome-file-list>
7  <servlet>
8          <servlet-name>dispatcher</servlet-name>
9          <servlet-class>
10              org.springframework.web.servlet.DispatcherServlet
11         </servlet-class>
12         <init-param>
13             <param-name>contextConfigLocation</param-name>
14             <param-value>/WEB-INF/alumno-servlet.xml</param-value>
15         </init-param>
16           <load-on-startup>1</load-on-startup>
17 </servlet>
18 <servlet-mapping>
19         <servlet-name>dispatcher</servlet-name>
20         <url-pattern>/spring-mvc/*</url-pattern>
21 </servlet-mapping>
22 </web-app>
```

... continuación Mi primera aplicación con Spring MVC:

6º Crear src\main\webapp\WEB-INF\alumno-servlet.xml:

Hay que crear un un nuevo fichero alumno-servlet.xml (el fichero xml de configuración del contexto de Spring) acordandose de cambiar base-package="org.alumno.nombreAlumno":

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:mvc="http://www.springframework.org/schema/mvc"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
	http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
	http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
	<context:component-scan base-package="org.profesor.joseramon" />
	<mvc:annotation-driven />

</beans>
```

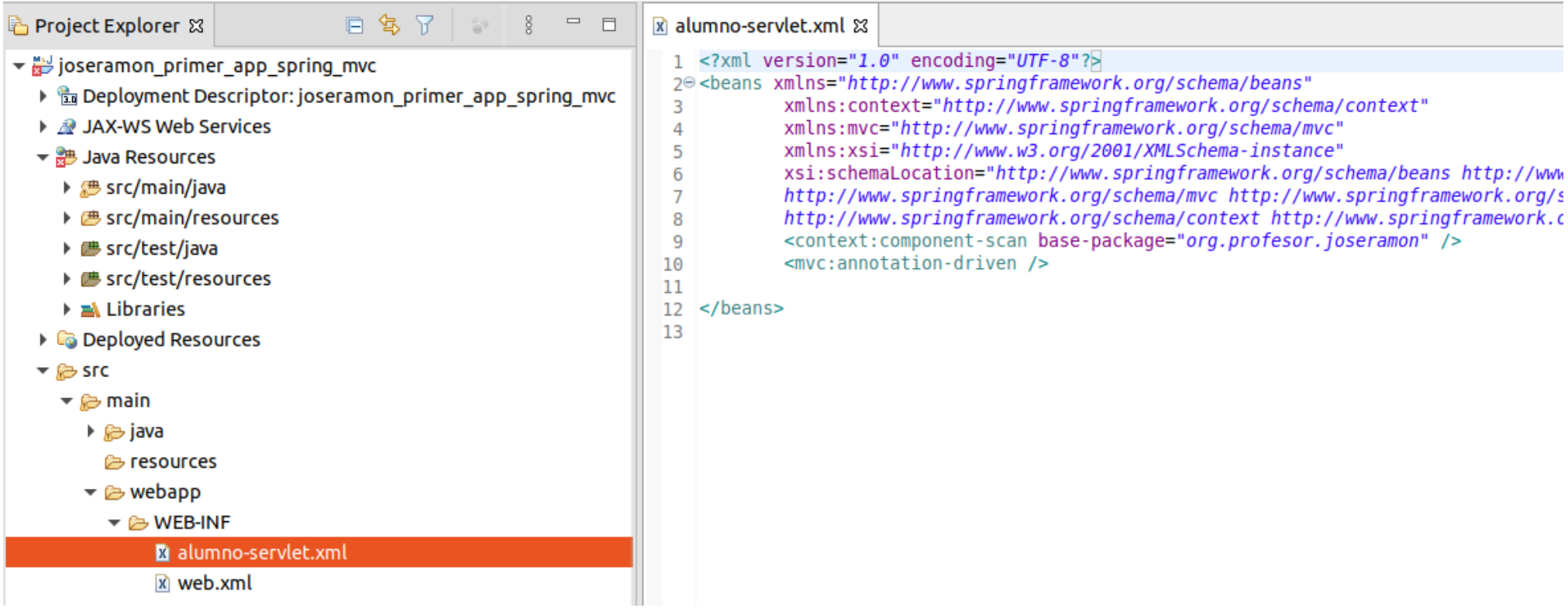
UD 2: Modelo Vista Controlador
2.- Spring MVC: Controlador

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

... continuación Mi primera aplicación con Spring MVC:

... continuación 6º Crear src\main\webapp\WEB-INF\alumno-servlet.xml:

El fichero alumno-servlet.xml debería quedar de la siguiente manera:

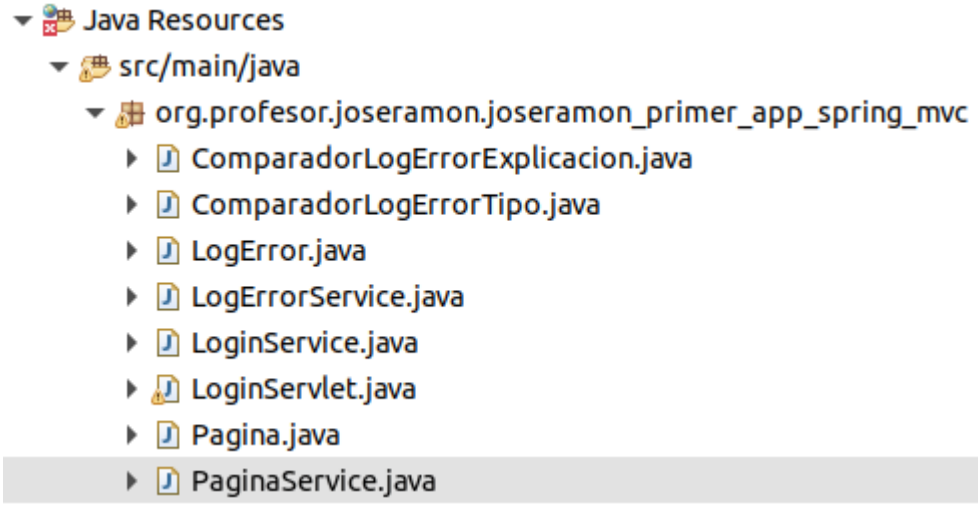


... continuación Mi primera aplicación con Spring MVC:

7º Crear las clases Java y los ficheros JSP para hacer el login inicial:

¿Que clases Java nos hacen falta?

Las vamos a crear inicialmente todas en el mismo paquete (como veremos en las siguientes diapositivas) y más adelante explicaremos como distribuirlas:

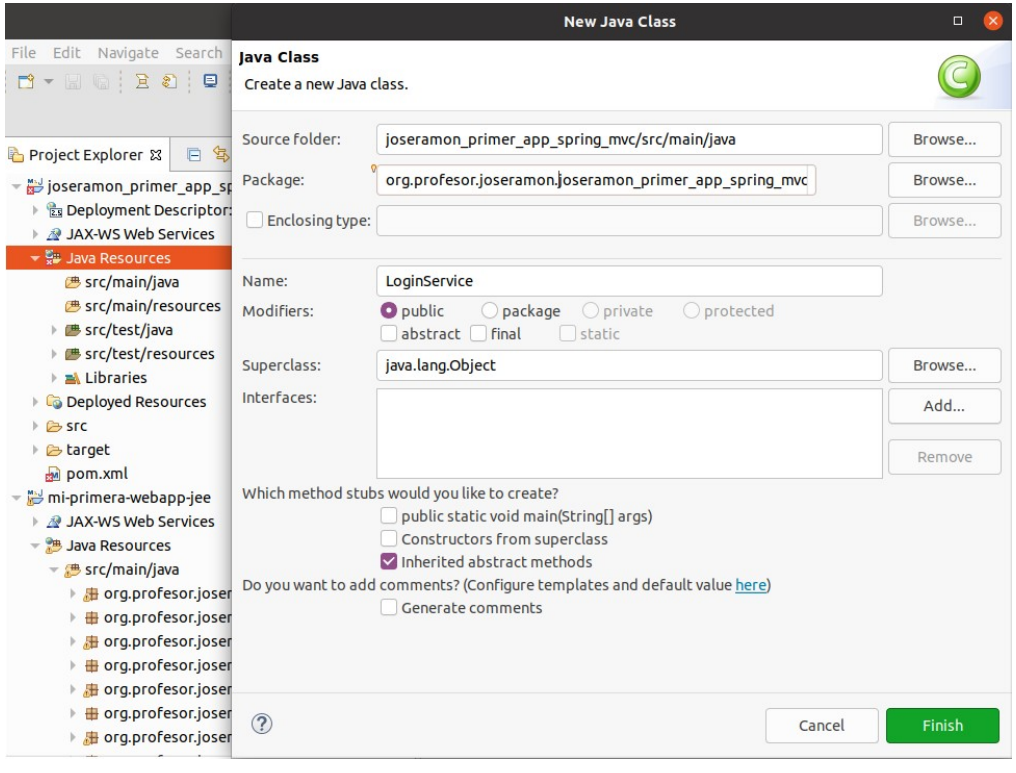


... continuación Mi primera aplicación con Spring MVC:

... continuación 7º Crear las clases Java y los ficheros JSP para hacer el login inicial:

Para crear la estructura de paquetes correcta nos colocamos en “Java Resources” y creamos una nueva clase (Java Resources\New \Class) LoginService:

Aunque sea repetitivo es importante que la estructura de paquetes sea **org.alumno.nombreAlumno.nombreAlumno_primer_app_spring_mvc** para que el profesor pueda revisar las prácticas de forma homogenea a todos igual:



... continuación Mi primera aplicación con Spring MVC:

... continuación 7º Crear las clases Java y los ficheros JSP para hacer el login inicial:

Ahora podemos copiar el contenido del fichero LoginService.java que teniamos en la UD1 pero bajo el nuevo paquete:

LoginService.java

```java
package org.profesor.joseramon.joseramon_primer_app_spring_mvc;

public class LoginService {
    public boolean usuarioValido(String usuario,String password) {
        //En condiciones normales se accedería a una base de datos para
        //confirmar la constraseña, pero por simplicidad lo validamos
        //comprobando si son unos valores fijos.
        //El alumno debe cambiar los valores del usuario y contraseña validos
        if (usuario.contentEquals("joseramon") && password.contentEquals("miPassword"))
            return true;
        return false;
    }
}
```

... continuación Mi primera aplicación con Spring MVC:

... continuación 7º Crear las clases Java y los ficheros JSP para hacer el login inicial:

Y debemos hacer lo mismo con las demás clases Java. En el caso de LoginServlet.java haremos una modificación para que vaya a la página "bienvenida.jsp" de momento:

LoginServlet.java

```
 8 import javax.servlet.http.HttpServletResponse;
 9
10 @WebServlet(urlPatterns = "/login.do")
11 public class LoginServlet extends HttpServlet {
12     LoginService loginServicio = new LoginService();
13     LogErrorService logErroresServicio= new LogErrorService();
14     PaginaService paginaServicio=new PaginaService();
15     static Pagina paginaLogin=new Pagina("Login","login.do");
16
17     @Override
18     protected void doGet(HttpServletRequest request,
19             HttpServletResponse response) throws IOException, ServletException {
20         paginaServicio.setPagina(paginaLogin);
21         request.getSession().setAttribute("pagina", paginaServicio.getPagina());
22         request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);
23     }
24
25     @Override
26     protected void doPost(HttpServletRequest request,
27             HttpServletResponse response) throws IOException, ServletException {
28         String nombre =request.getParameter("nombre");
29         String password =request.getParameter("password");
30         paginaServicio.setPagina(paginaLogin);
31         request.getSession().setAttribute("pagina", paginaServicio.getPagina());
32         request.getSession().setAttribute("numModulosInsertados",0);
33         if (loginServicio.usuarioValido(nombre, password)) {
34             request.getSession().setAttribute("nombre", nombre);
35             //validación correcta: Redirigir al Servlet de alumno
36             //response.sendRedirect("list-alumno.do");
37             request.setAttribute("nombre", nombre );
38             request.getRequestDispatcher("/WEB-INF/views/bienvenida.jsp").forward(request, response);
39         } else {
40             //validación incorrecta
41             logErroresServicio.addLogError("Login incorrecto","Login incorrecto de '"+nombre+"'.");
42             request.setAttribute("errores", "Usuario '" + nombre + "' o contraseña incorrecta");
43             request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);
44         }
45     }
46 }
```

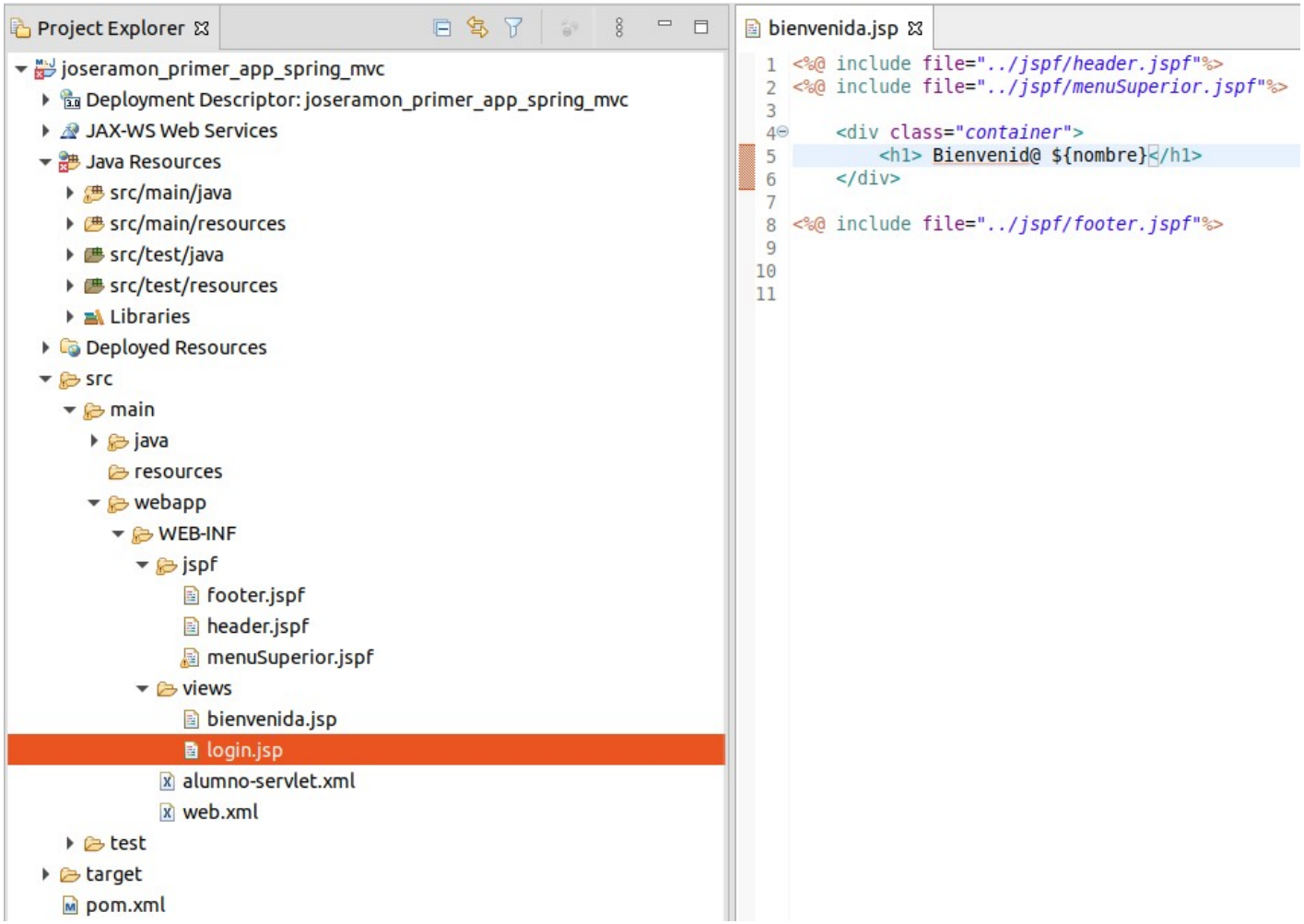
UD 2: Modelo Vista Controlador
2.- Spring MVC: Controlador

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

... continuación Mi primera aplicación con Spring MVC:

... continuación 7º Crear las clases Java y los ficheros JSP para hacer el login inicial:

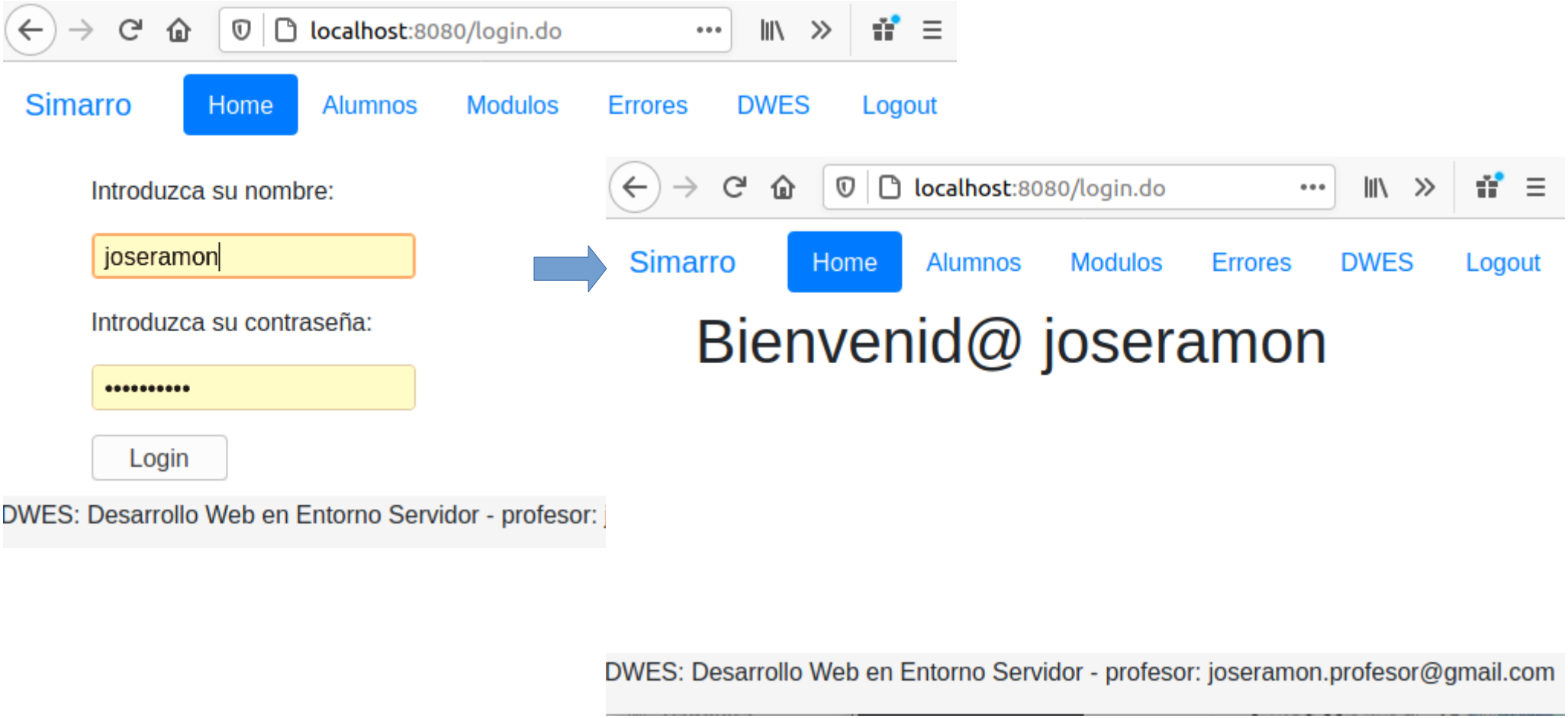
Y ahora copiamos las siguientes páginas .jsp y creamos bienvenida.jsp:



... continuación Mi primera aplicación con Spring MVC:

8º Comprobar que el login funciona:

Aunque la sección de alumnos , modulos y errores no funciona , podemos arrancar la web y probar que hacemos login correctamente y vamos a la página de bienvenida :



... continuación Mi primera aplicación con Spring MVC:

9º Crear nuestro primer controlador Spring MVC : LoginController.java

Realmente todavía estamos trabajando con los Servlets de Java Enterprise Edition 8 (JEE8).

Creemos nuestro primer controlador LoginController.java con el código de la siguiente página en un subpaquete “springmvc”:

Antes de codificar debemos entender que hacen las siguientes **anotaciones Spring MVC** :

@Controller :
Indicamos que se trata de un controlador Spring MVC

@RequestMapping:
Indicamos la ruta del controlador (url accesible desde el navegador)

@ResponseBody:
Indicamos que lo que devolvemos es el cuerpo de la respuesta.

... continuación Mi primera aplicación con Spring MVC:

... continuación 9º Crear nuestro primer controlador Spring MVC : LoginController.java

Si comparamos LoginServlet.java que muestra una página HTML con la notación JEE8 y su equivalente LoginController (con la notación Spring MVC) nos damos cuenta que es mucho más sencillo Spring MVC si entendemos las anotaciones :

LoginServlet.java

```
1  package webapp;
2
3  import java.io.IOException;
10
12 * Browser sends Http Request to Web Server
19
20 /*Java Platform, Enterprise Edition (Java EE) JEE
32 @WebServlet(urlPatterns = "/login.do")
33 public class LoginServlet extends HttpServlet {
34     @Override
35     protected void doGet(HttpServletRequest request
36         PrintWriter out = response.getWriter();
37         out.println("<html>");
38         out.println("<head>");
39         out.println("<title>Mi primer Servlet</titl
40         out.println("</head>");
41         out.println("<body>");
42         out.println("Hola Mundo desde mi primer Ser
43         out.println("</body>");
44         out.println("</html>");
45     }
46 }
```

LoginController.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.ResponseBody;
6
7  @Controller
8  public class LoginController {
9      @RequestMapping(value="/login")
10     @ResponseBody
11     public String holaMundo() {
12         return "Hola Mundo desde Spring MVC!!";
13     }
14 }
15
```

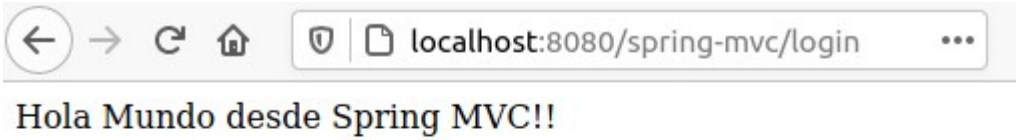
Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
joseramon_primer_app_spring_mvc [Maven Build] /usr/lib/jvm/java-13-openjdk-amd64/bin/java
INFORMACIÓN: Initializing Spring DispatcherServlet 'dispatcher'
[INFO] Initializing Servlet 'dispatcher'
[INFO] Completed initialization in 600 ms
```

... continuación Mi primera aplicación con Spring MVC:

10º Comprobar que la web nos da la bienvenida:

Solo nos queda comprobar si funciona accediendo a la ruta spring-mvc/login:

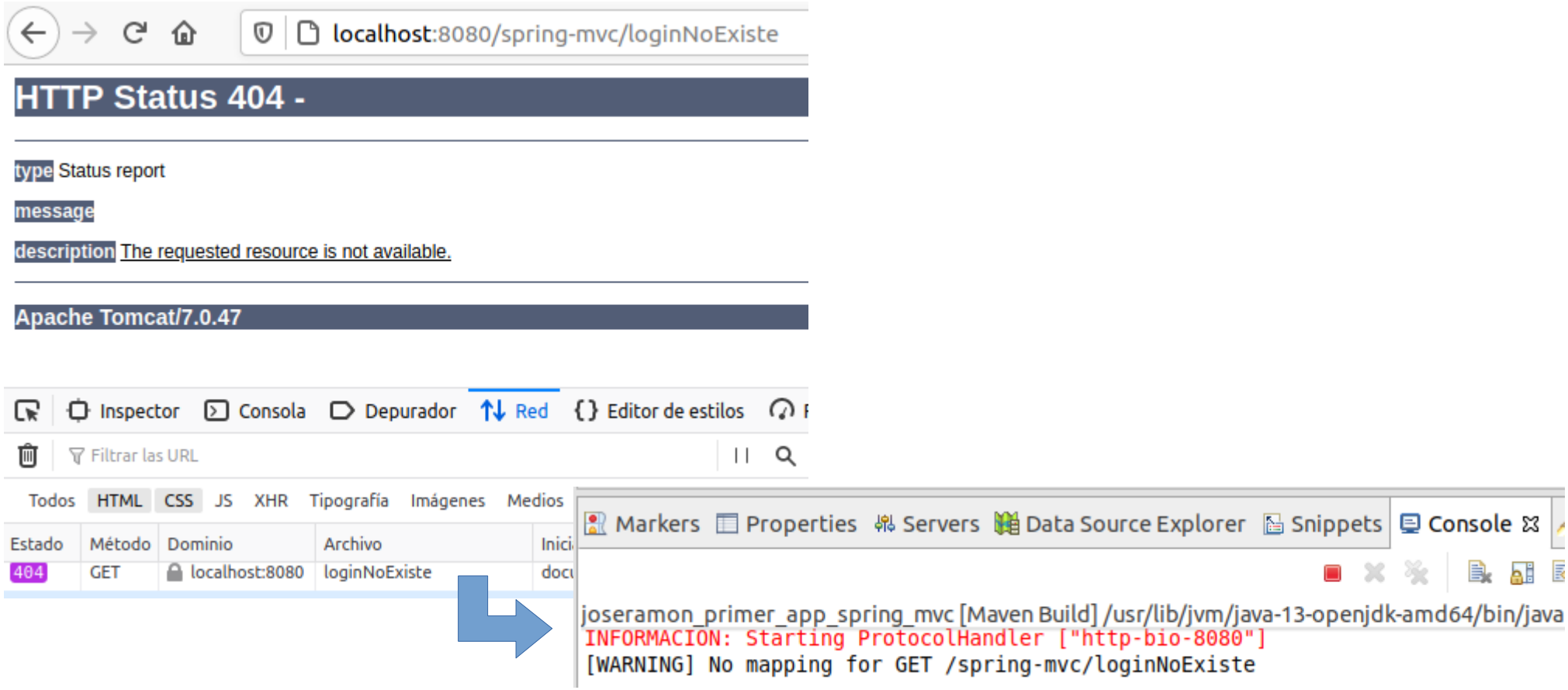


¿Que pasa si introducimos una url incorrecta?

... continuación Mi primera aplicación con Spring MVC:

... continuación 10º Comprobar que la web nos da la bienvenida:

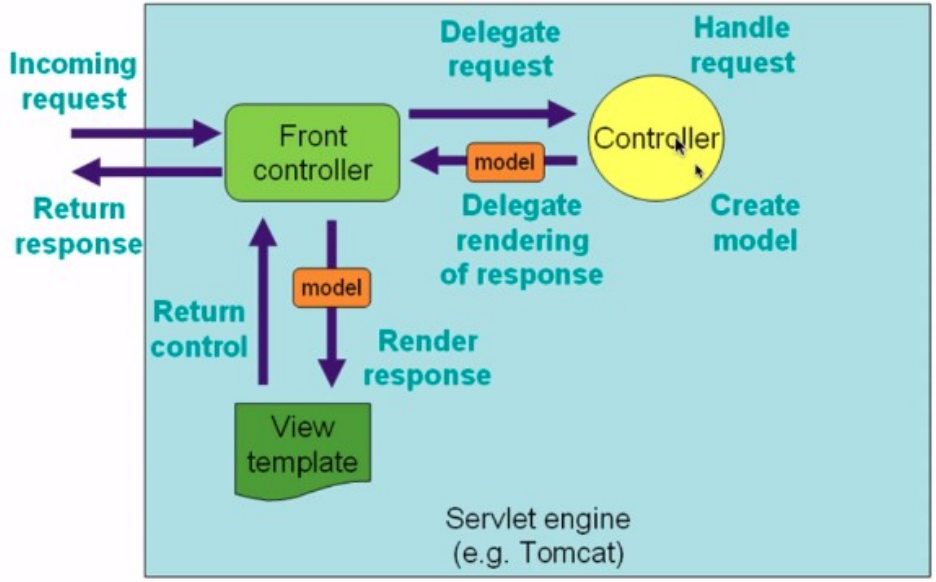
Si introducimos un url inexistente podemos ver el aviso que se genera en la consola :



Recapitulemos lo que hemos visto hasta ahora:

En nuestra primera aplicación hemos configurado nuestro ***DispatcherServlet*** en ***web.xml*** que actua de Front controller recibiendo todas las peticiones. Para ello hemos tenido que configurar el contexto en ***alumno-servlet.xml***.

Adicionalmente hemos configurado un ***controlador básico (LoginController)*** con las anotaciones ***@Controller y @RequestMapping*** . Por último, hemos hecho que el controlador devuelva un texto al navegador gracias a la anotación ***@ResponseBody***.



Ahora nos queda ver como configurar la vista (ViewResolver) para poder mostrar una página JSP con los datos del modelo (ModelMap). Con esta práctica tendremos un ejemplo completo del patrón FrontController de Spring MVC.

UD 2: Modelo Vista Controlador
2.- Spring MVC: Vista

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

Continuemos configurando nuestra primera web con Spring MVC:

Para mostrar una página JSP en la vista primero que nada vamos a borrar la anotación @ResponseBody, renombrar el método holaMundo() por mostrarLogin() e indicarle en el return cual es el nombre de la página JSP queremos mostrar (login):

LoginController.java

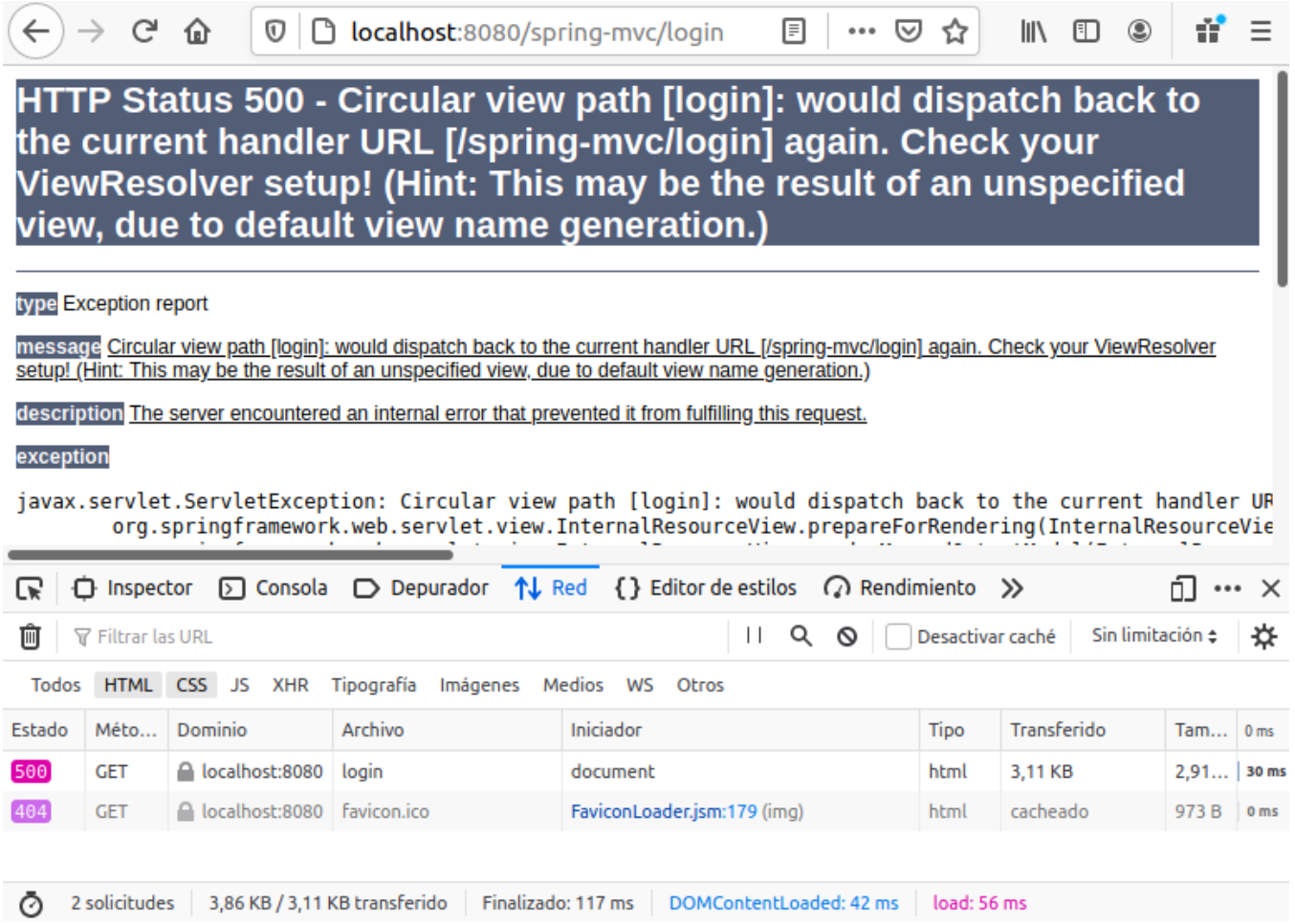
```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.ResponseBody;
6
7  @Controller
8  public class LoginController {
9      @RequestMapping(value="/login")
10     @ResponseBody
11     public String holaMundo() {
12         return "Hola Mundo desde Spring M
13     }
14 }
```

LoginController.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5
6  @Controller
7  public class LoginController {
8      @RequestMapping(value="/login")
9      public String mostrarLogin() {
10         return "login";
11     }
12 }
```

Solo hemos puesto “login”, ¿Que debemos configurar para que al decir “login” vaya a “WEB-INF/views/login.jsp”?

Si no configuramos nada más, cuando accedamos a la web nos dará el siguiente error porque al no tener configurado el “View Resolver” se cree que el return es una url, o lo que es lo mismo , piensa que desde login estamos enviando a login y así infinitamente:



UD 2: Modelo Vista Controlador
2.- Spring MVC: Vista

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

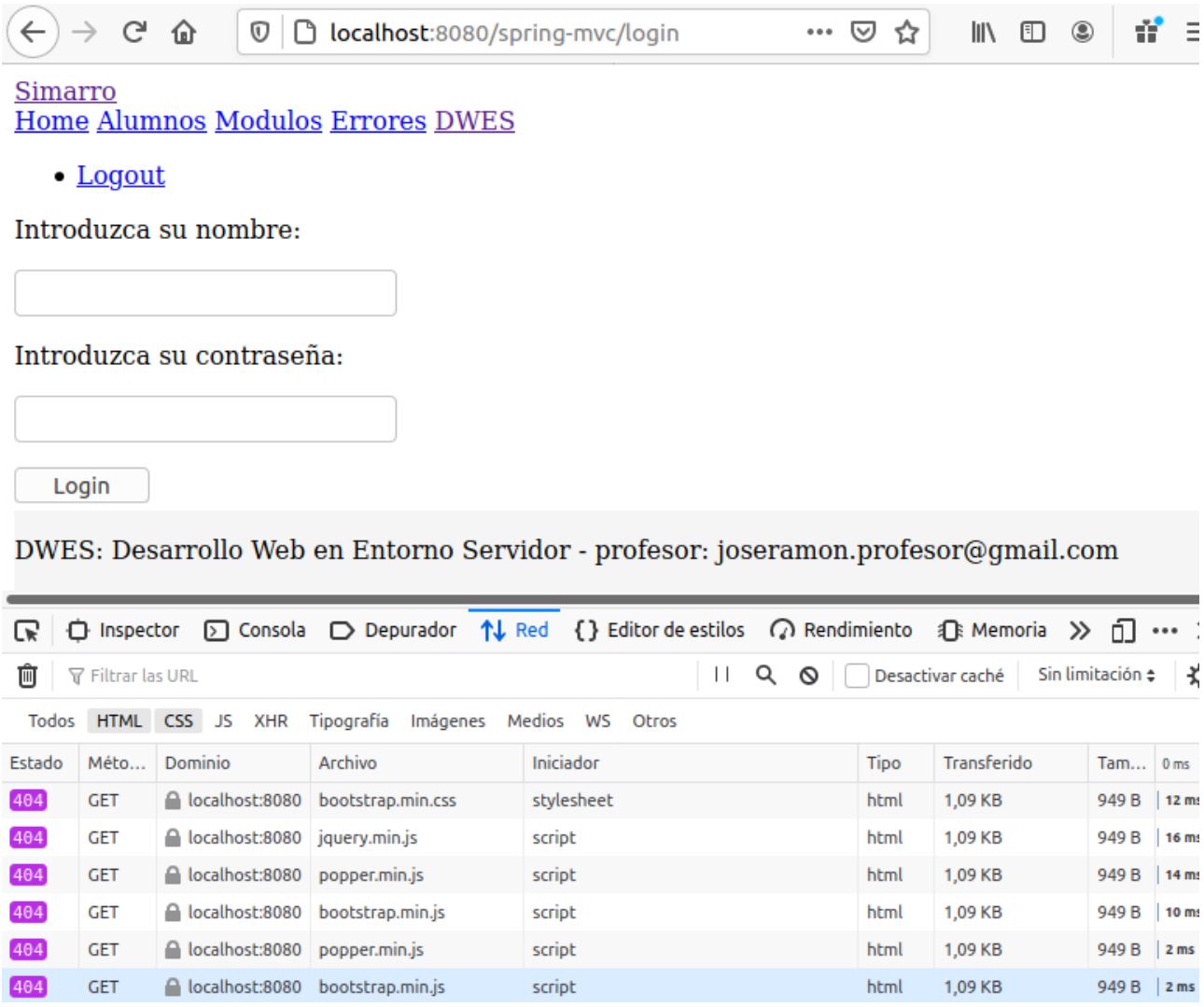
Para redirigir a la vista “login.jsp” debemos configurar el View Resolver en alumno-servlet.xml que utilizará el Despatcher Servlet cuando no encuentre @RequestBody. Para que solo haga falta escribir “login” y se añada automáticamente el prefijo “/WEB-INF/view/” y el sufijo “.jsp” deberemos añadir el siguiente código en alumno-servlet.xml:

```
<bean
	class="org.springframework.web.servlet.view.InternalResourceViewResolver">
	<property name="prefix">
	   <value>/WEB-INF/views/</value>
	</property>
	<property name="suffix">
	   <value>.jsp</value>
	</property>
</bean>
```

alumno-servlet.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3  	xmlns:context="http://www.springframework.org/schema/context"
4  	xmlns:mvc="http://www.springframework.org/schema/mvc"
5  	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6  	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.
7  	http://www.springframework.org/schema/mvc http://www.springframework.org/sc
8  	http://www.springframework.org/schema/context http://www.springframework.or
9  	<context:component-scan base-package="org.profesor.joseramon" />
10 	<mvc:annotation-driven />
11 <bean
12 	class="org.springframework.web.servlet.view.InternalResourceViewResolver">
13 	<property name="prefix">
14 	    <value>/WEB-INF/views/</value>
15 	</property>
16 	<property name="suffix">
17 	    <value>.jsp</value>
18 	</property>
19 </bean>
20 </beans>
```

Sin embargo, un problema que podemos tener es que no se visualice bien la página al consultar el navegador ...



El problema se produce porqué no encuentran los ficheros de Bootstrap.

header.jspf

```
1  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <link href="webjars/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
```

footer.jspf

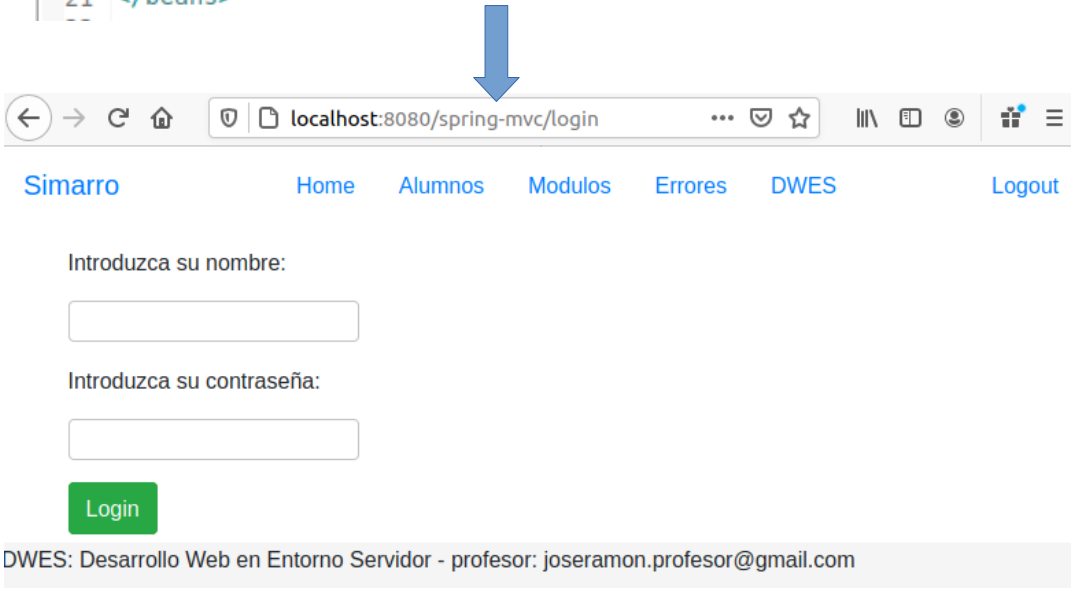
```
1      <footer class="footer">
2          <p>DWES: Desarrollo Web en Entorno Servidor - profesor: joseramon
3      </footer>
4
5      <script src="webjars/jquery/3.5.1/jquery.min.js"></script>
6      <script src="webjars/popper.js/1.16.0/umd/popper.min.js"></script>
7      <script src="webjars/bootstrap/4.5.2/js/bootstrap.min.js"></script>
8
9  </body>
10
11 </html>
```

Para incorporar correctamente esos ficheros en la siguiente diapositiva introduciremos la ruta en el fichero de configuración del contexto alumno-servlet.xml

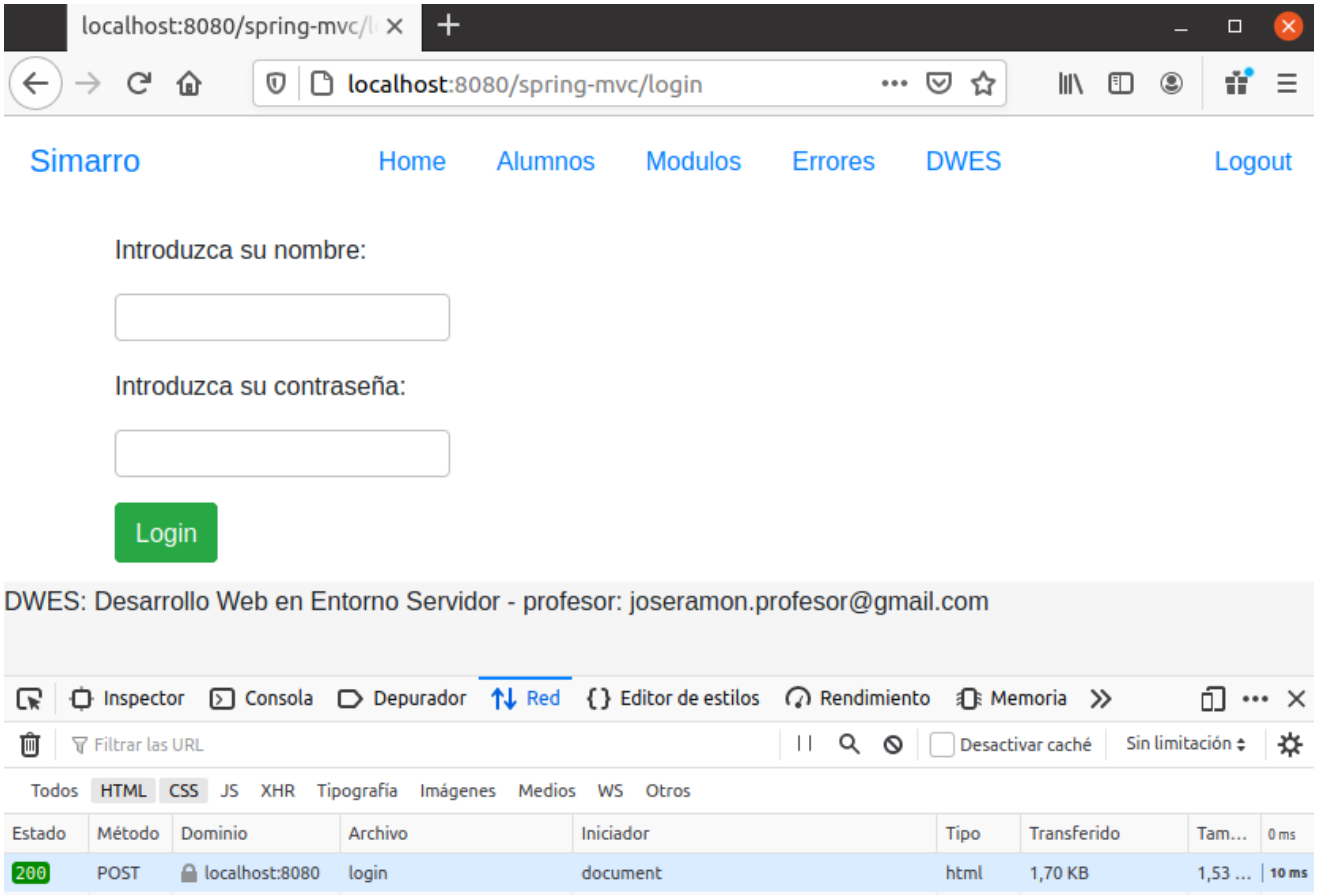
Si añadimos la linea que incorpora la carpeta webjars en alumno-servlet.xml solucionamos el problema:

alumno-servlet.xml

```
6         xsi:schemaLocation="http://www.springframework.org/sc
7         http://www.springframework.org/schema/mvc http://www.
8         http://www.springframework.org/schema/context http://
9         <context:component-scan base-package="org.profesor.jo
10        <mvc:annotation-driven />
11  <bean
12        class="org.springframework.web.servlet.view.InternalR
13        <property name="prefix">
14            <value>/WEB-INF/views/</value>
15        </property>
16        <property name="suffix">
17            <value>.jsp</value>
18        </property>
19  </bean>
20  <mvc:resources mapping="/webjars/**" location="/webjars/"/>
21  </beans>
```



Si comprobamos en el navegador el funcionamiento vemos que al pulsar en el botón verde “Login” nos devuelve a la misma página de Login:



¿Por que se comporta así?

El problema viene porque no le hemos especificado que hacer ante un POST, por lo que mostrarLogin() se ejecuta tanto para un GET como para un POST.

¿Como se lo decimos en Spring si no existe el doGet() ni doPost() de JEE8?

LoginController.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;
2
3  import org.springframework.stereotype.Controller;
5
6  @Controller
7  public class LoginController {
8      @RequestMapping(value="/login")
9      public String mostrarLogin() {
10         return "login";
11     }
12 }
```

Para solucionarlo debemos añadir un nuevo parámetro a @RequestMapping indicando el método (method) y un nuevo método “procesaLogin()” que atenderá la petición de POST:

LoginController.java

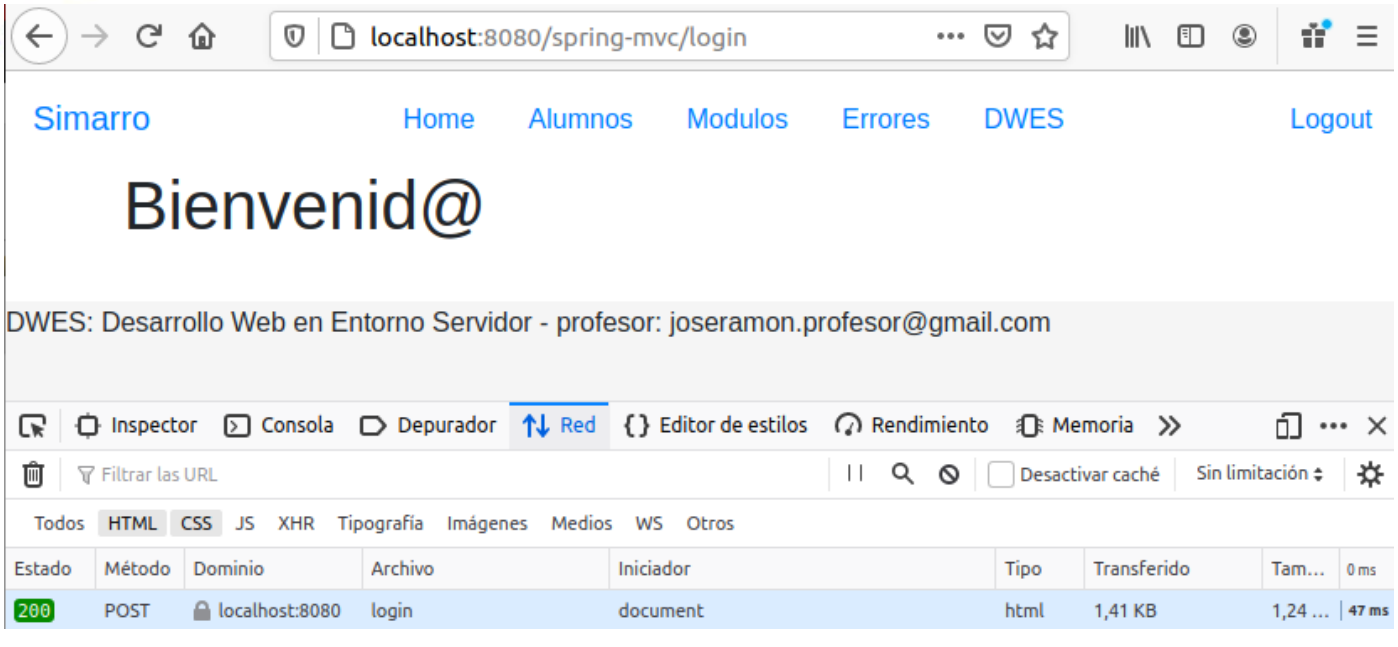
```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;
2
3  import org.springframework.stereotype.Controller;
6
7  @Controller
8  public class LoginController {
9      @RequestMapping(value="/login",method = RequestMethod.GET)
10     public String mostrarLogin() {
11         return "login";
12     }
13
14     @RequestMapping(value="/login",method = RequestMethod.POST)
15     public String procesaLogin() {
16         return "bienvenida";
17     }
18 }
```

Ahora la pregunta es:

¿ Por que no se muestra el usuario aunque se lo indicamos a bienvenida.jsp?

bienvenida.jsp

```
1 <%@ include file="../jspf/header.jspf"%>
2 <%@ include file="../jspf/menuSuperior.jspf"%>
3
4     <div class="container">
5         <h1> Bienvenid@ ${nombre}</h1>
6     </div>
7
8 <%@ include file="../jspf/footer.
```



UD 2: Modelo Vista Controlador
2.- Spring MVC: Modelo

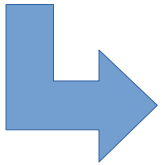
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

El request.getParameter() de JEE8 se debe convertir en un parámetro de entrada (@RequestParam) del método en Spring MVC y la forma de configurar un atributo en Spring MVC para mostrarse en el jsp es utilizando una nueva clase ModelMap como parámetro de la función y añadiendole (model.put()) los datos que necesitemos.

!!Realiza los cambios!!

LoginController.java

```java
package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class LoginController {
    @RequestMapping(value="/login",method = RequestMethod.GET)
    public String mostrarLogin() {
        return "login";
    }

    @RequestMapping(value="/login",method = RequestMethod.POST)
    public String procesaLogin(@RequestParam String nombre,
            ModelMap model) {
            model.put("nombre", nombre);
            return "bienvenida";
    }
}
```

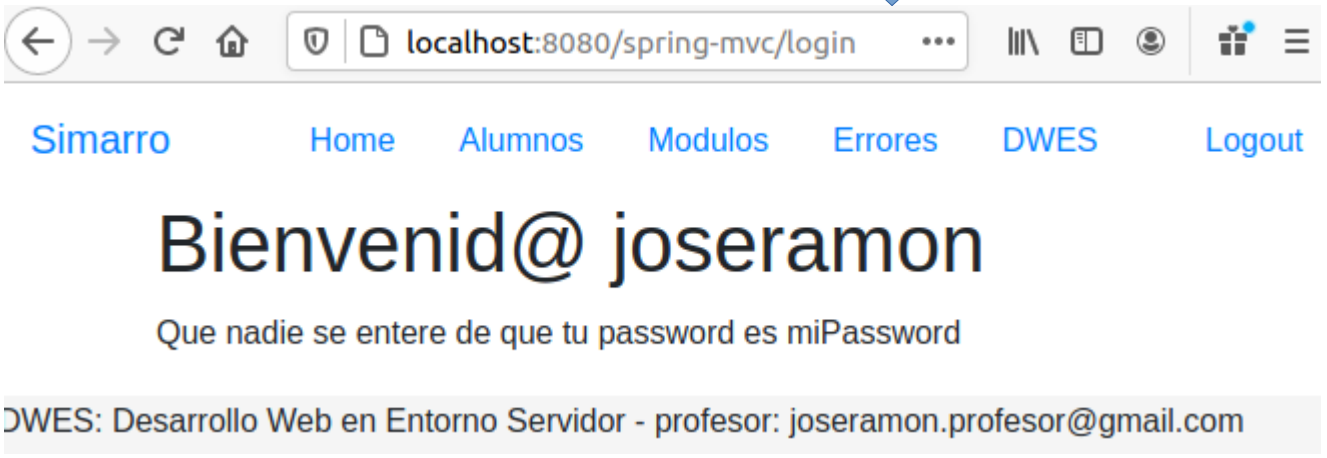


De igual manera , si queremos leer en el controlador más de un valor del formulario de la página jsp, se lo debemos añadir como parámetro al método. Modifica el método y el jsp para leer también el password del usuario y mostrarlo en el jsp:

```java
@RequestMapping(value="/login",method = RequestMethod.POST)
public String procesaLogin(@RequestParam String nombre,
        @RequestParam String password,
        ModelMap model) {
        model.put("nombre", nombre);
        model.put("password", password);
        return "bienvenida";
}
```

bienvenida.jsp

```jsp
<%@ include file="../jspf/header.jspf"%>
<%@ include file="../jspf/menuSuperior.jspf"%>

    <div class="container">
        <h1> Bienvenid@ ${nombre}</h1>
        Que nadie se entere de que tu password es ${password}
    </div>
<%@ include file="../jspf/footer.jspf"%>
```



UD 2: Modelo Vista Controlador
2.- Spring MVC: Servicio

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

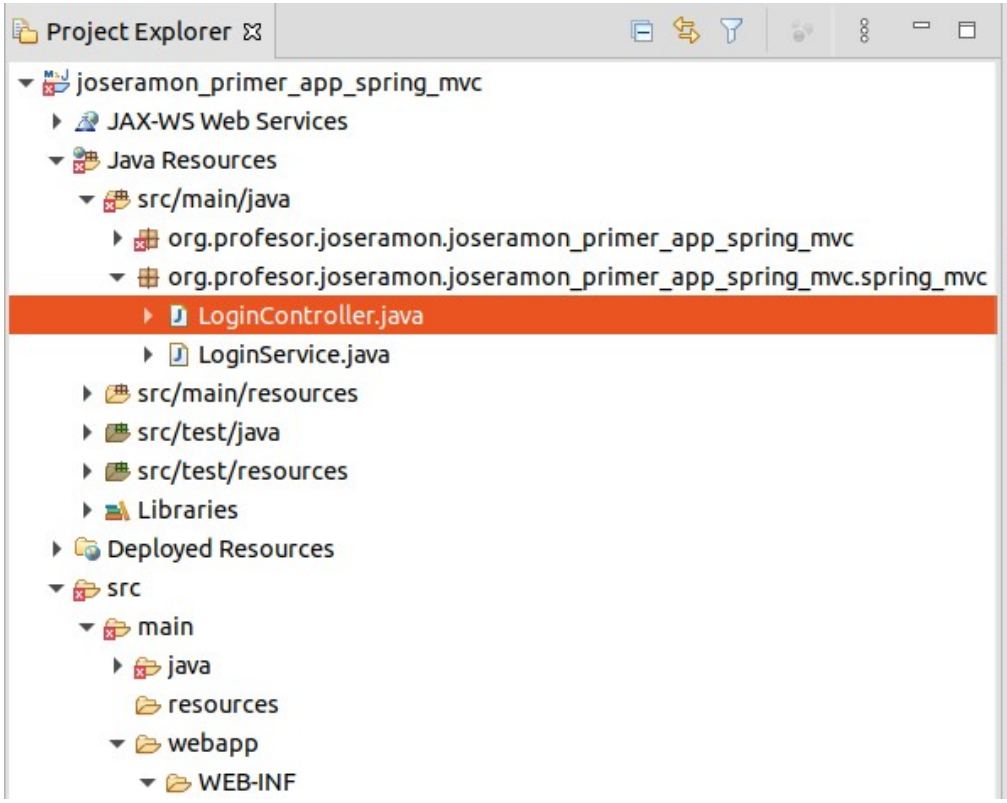
Ya hemos visto como funciona Spring MVC, pero nos queda una parte importante. El controlador solo debe recibir peticiones, procesarlas y llamar a la vista correspondiente. Para su procesamiento se debe apoyar en los servicios que tiene a su disposición para que el código del controlador sea lo más limpio posible.

¿ Como utilizamos los servicios desde un controlador?

UD 2: Modelo Vista Controlador
2.- Spring MVC: Servicio

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

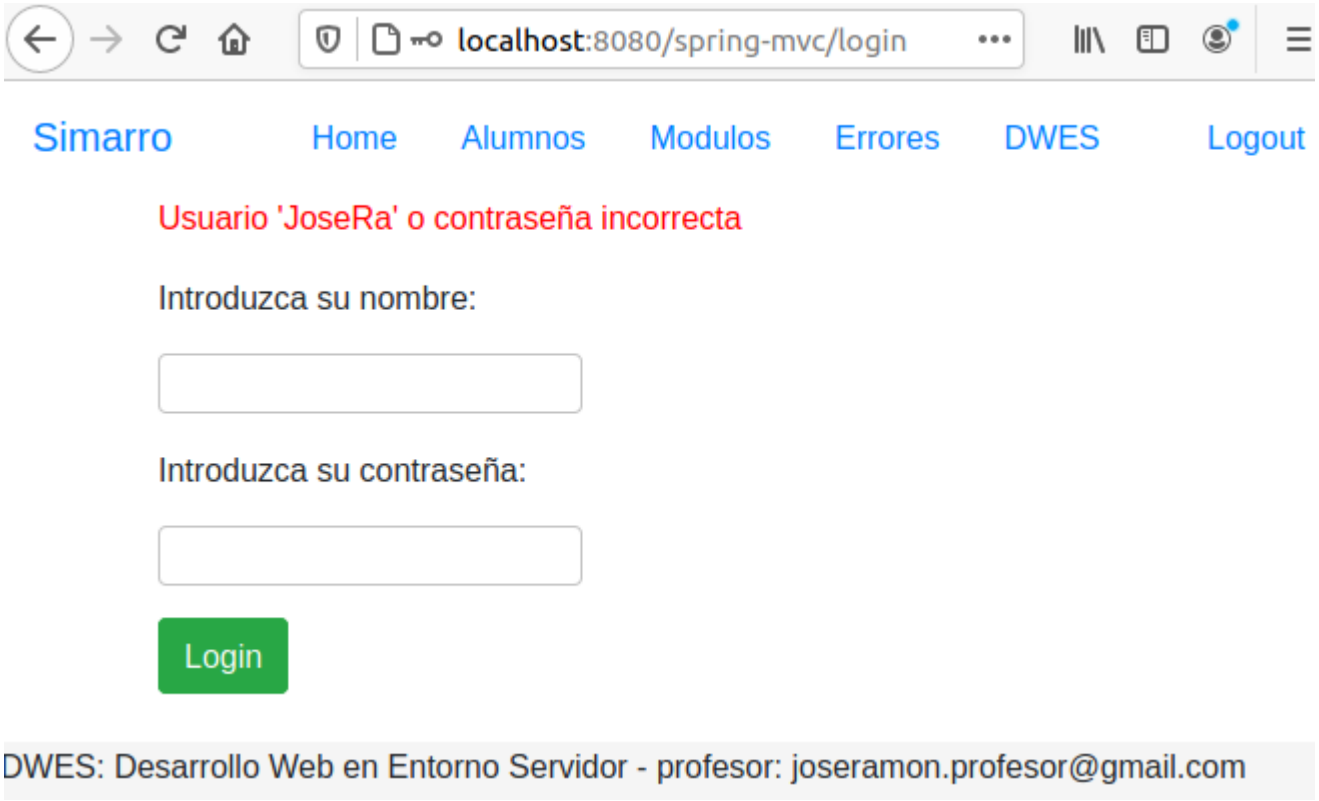
La primera aproximación es hacerlo al estilo JEE8. Mueve el servicio LoginServicio.java al paquete spring_mvc y modifica el controlador para incorporar el servicio que chequea el login y en función de si es correcto o no vuelve a reenviar a la pagina de login o a la de bienvenida.



LoginController.java

```
7  import org.springframework.web.bind.annotation.RequestParam;
8
9  @Controller
10 public class LoginController {
11     LoginService loginService= new LoginService();
12
13     @RequestMapping(value="/login",method = RequestMethod.GET)
14     public String mostrarLogin() {
15         return "login";
16     }
17
18     @RequestMapping(value="/login",method = RequestMethod.POST)
19     public String procesaLogin(@RequestParam String nombre,
20             @RequestParam String password,
21             ModelMap model) {
22         if (!loginService.usuarioValido(nombre, password)) {
23             //usuario inválido, volver a intentar logearse
24             return "login";
25         }
26         // Si llega aquí es porque el usuario era valido
27         model.put("nombre", nombre);
28         model.put("password", password);
29         return "bienvenida";
30     }
31 }
```

Si compruebas el funcionamiento esta todo bien menos una cosa. En caso de introducir las credenciales del usuario incorrectas vuelve al login pero no avisa del error. Modifica el controlador para que se visualice el mensaje de login incorrecto como hacíamos con JEE8:



UD 2: Modelo Vista Controlador
2.- Spring MVC: Servicio

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

Ahora vamos a sacarle todo el partido a Spring MVC. No vamos a realizar una explicación teórica, pero es una mala práctica crear el servicio desde el controlador. Es mucho mejor para que el código este desacoplado que el servicio este creado y en el controlador simplemente enlace con dicho servicio.

LoginController.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.ui.ModelMap;
5  import org.springframework.web.bind.annotation.RequestMapping;
6  import org.springframework.web.bind.annotation.RequestMethod;
7  import org.springframework.web.bind.annotation.RequestParam;
8
9  @Controller
10 public class LoginController {
11     LoginService loginService= new LoginService();
12
14     public String mostrarLogin() {
17
18     @RequestMapping(value="/login",method = RequestMethod.POST)
19     public String procesaLogin(@RequestParam String nombre,
20             @RequestParam String password,
21             ModelMap model) {
22         if (!loginService.usuarioValido(nombre, password)) {
```

Para solucionarlo, **Spring** tiene 2 funcionalidades que son la base de su **éxito: La inyección de dependencias y el autoenlace a componentes instanciados.**

UD 2: Modelo Vista Controlador
2.- Spring MVC: Servicio

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

Para enlazar LoginService en LoginController debemos hacer que LoginService tenga una instancia valida. Para elllo añadimos la anotación @Service en LoginService.java,con lo cual Spring en primer lugar instanciará el servicio que después se inyectará en LoginController sin necesidad de que LoginController cree una instancia nueva de la clase LoginService:

LoginService.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;
2
3  import org.springframework.stereotype.Service;
4
5  @Service
6  public class LoginService {
7      public boolean usuarioValido(String usuario,String password) {
8          //En condiciones normales se accedería a una base de datos para
9          //confirmar la constraseña, pero por simplicidad lo validamos
10         //comprobando si son unos valores fijos.
11         //El alumno debe cambiar los valores del usuario y contraseña validos
12         if (usuario.contentEquals("joseramon") && password.contentEquals("miPassword"))
13             return true;
14         return false;
15     }
16 }
```

UD 2: Modelo Vista Controlador
2.- Spring MVC: Servicio

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

Para que el controlador LoginController puede utilizar este servicio debemos enlazar LoginService en LoginController y para ello inyectaremos la dependencia LoginService en LoginController mediante la anotación @Autowired. Fijate que ahora ya no creamos la instancia desde el controlador:

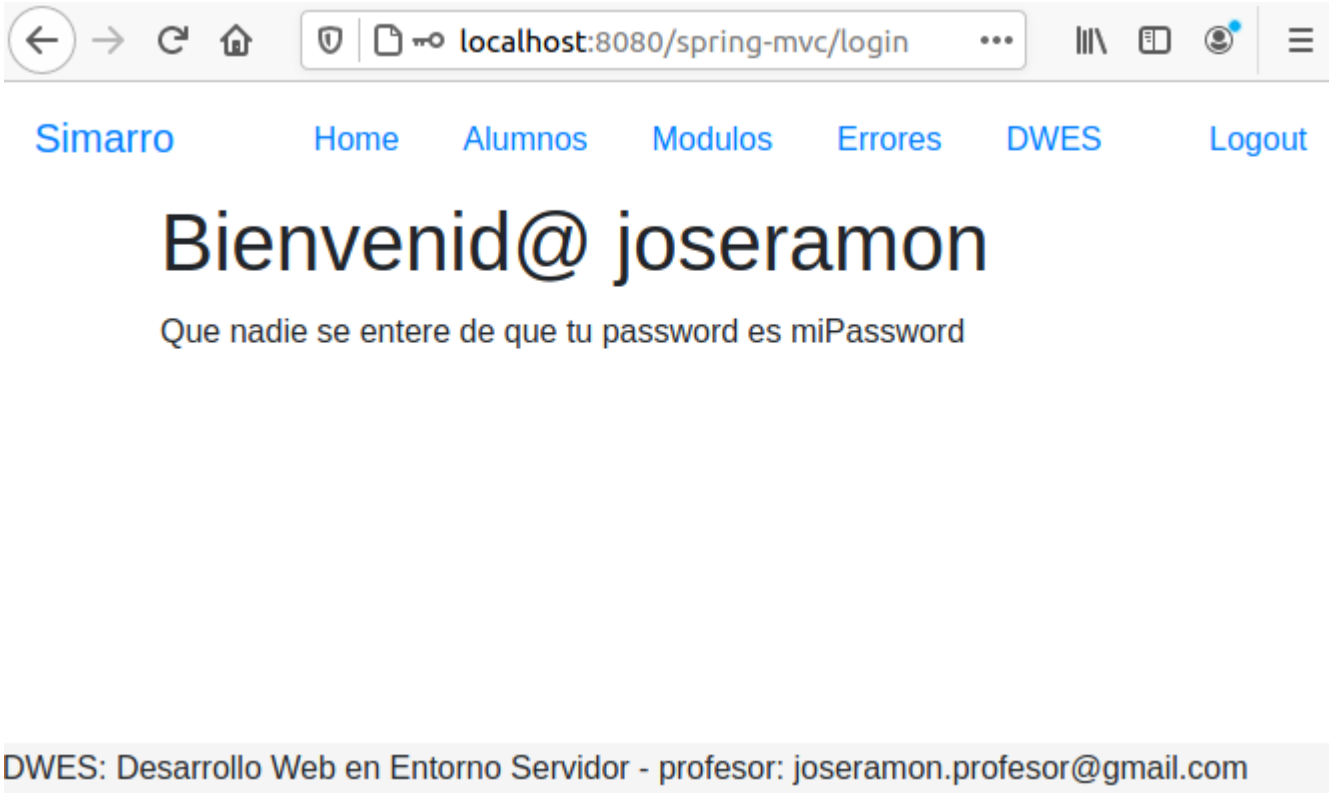
LoginController.java

```java
package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;

import org.springframework.stereotype.Controller;

@Controller
public class LoginController {
    LoginService loginService= new LoginService();

    @RequestMapping(value="/login",method = RequestMethod.GET)
    public String mostrarLogin() {
        return "login";
    }

    @RequestMapping(value="/login",method = RequestMethod.POST)
    public String procesaLogin(@RequestParam String nombre,
            @RequestParam String password,
            ModelMap model) {
        if (!loginService.usuarioValido(nombre, password)) {
            //usuario inválido, volver a intentar logearse
            model.put("errores", "Usuario '" + nombre + "' o contraseña incorrecta");
            return "login";
        }
        // Si llega aquí es porque el usuario era valido
        model.put("nombre", nombre);
        model.put("password", password);
        return "bienvenida";
    }
}
```

LoginController.java

```java
package org.profesor.joseramon.joseramon_primer_app_spring_mvc.spring_mvc;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class LoginController {
    @Autowired
    LoginService loginService;

    @RequestMapping(value="/login",method = RequestMethod.GET)
    public String mostrarLogin() {
        return "login";
    }

    @RequestMapping(value="/login",method = RequestMethod.POST)
    public String procesaLogin(@RequestParam String nombre,
            @RequestParam String password,
            ModelMap model) {
        if (!loginService.usuarioValido(nombre, password)) {
            //usuario inválido, volver a intentar logearse
            model.put("errores", "Usuario '" + nombre + "' o contraseña incorrecta");
            return "login";
        }
        // Si llega aquí es porque el usuario era valido
        model.put("nombre", nombre);
        model.put("password", password);
        return "bienvenida";
    }
}
```

Realiza los cambios anteriores y comprueba que la web sigue funcionando. Puede que necesites parar y relanzar la web:



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 UD2_practica1_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.