

SPRING BOOT

DAW 2 - DAW

APLICACIÓN WEB SPRING BOOT

- Spring Boot:
 - ❖ Proyecto creado a partir de Spring, el cual permite desarrollar y arrancar de forma muy rápida aplicaciones basadas en Spring.

APLICACIÓN WEB SPRING BOOT

- 1er paso: Crear un proyecto java maven.
- En el pom.xml se añaden las dependencias específicas de Spring boot:
- <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web>

APLICACIÓN WEB SPRING BOOT

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
</parent>

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

APLICACIÓN WEB SPRING BOOT

- Para poder compilar y ejecutar páginas JSP se necesita incluir la dependencia [tomcat-embed-jasper](#)

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

APLICACIÓN WEB SPRING BOOT

- También se necesitan las dependencias del servidor Tomcat como contenedor Web.
- No obstante, para evitar que las dependencias proporcionadas por nuestra aplicación entren en conflicto con las proporcionadas por el servidor Tomcat en tiempo de ejecución se deben establecer estas dos dependencias con el scope *provided*:

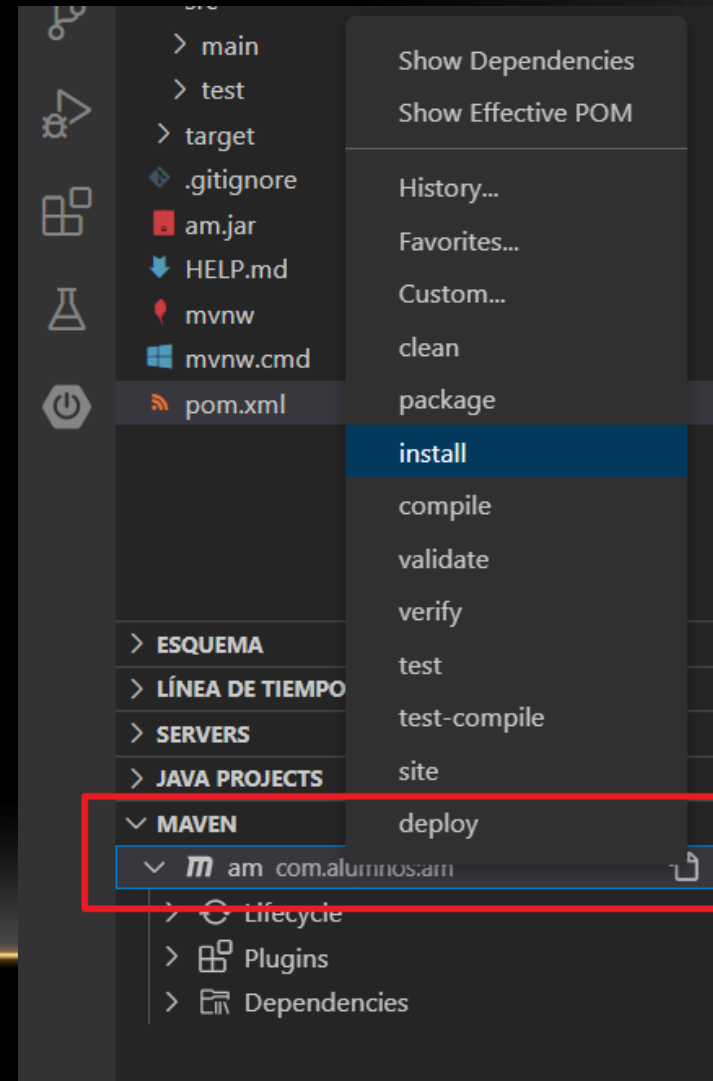
APLICACIÓN WEB SPRING BOOT

```
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>

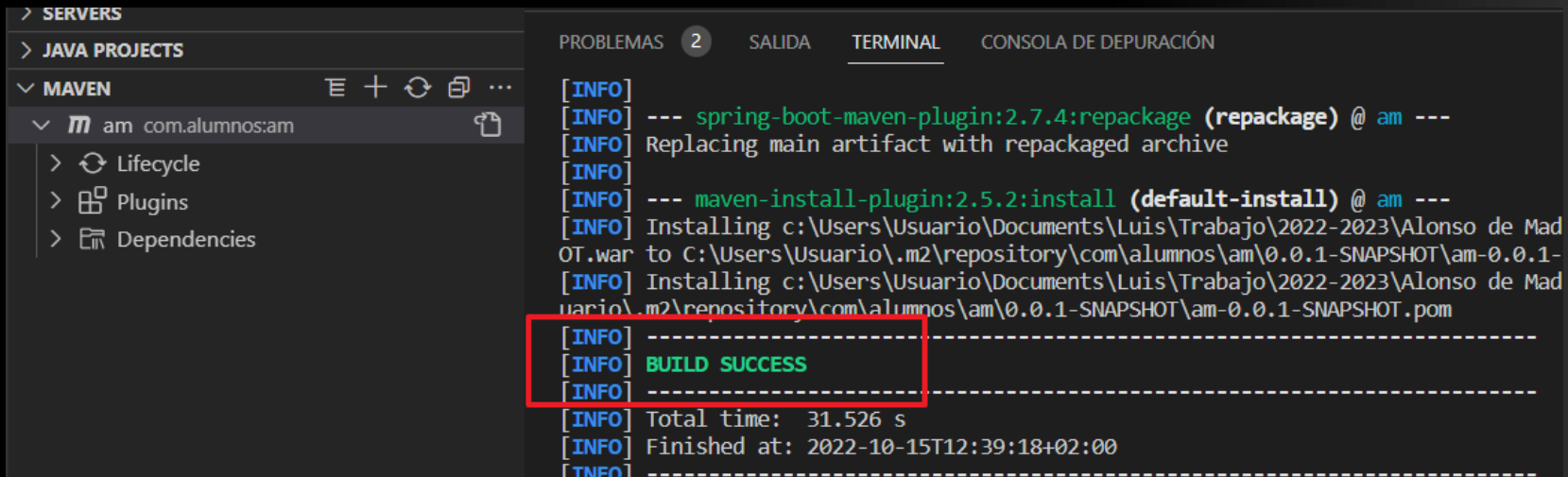
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <version>2.4.4</version>
  <scope>provided</scope>
</dependency>
```

APLICACIÓN WEB SPRING BOOT

- 2º. Se instalan las dependencias de maven en el proyecto:



APLICACIÓN WEB SPRING BOOT



```
> SERVERS
> JAVA PROJECTS
v MAVEN
  v m am com.alumnos:am
    > Lifecycle
    > Plugins
    > Dependencies

[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.4:repackage (repackage) @ am ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ am ---
[INFO] Installing c:\Users\Usuario\Documents\Luis\Trabajo\2022-2023\Alonso de Mad
OT.war to C:\Users\Usuario\.m2\repository\com\alumnos\am\0.0.1-SNAPSHOT\am-0.0.1-
[INFO] Installing c:\Users\Usuario\Documents\Luis\Trabajo\2022-2023\Alonso de Mad
uario\.m2\repository\com\alumnos\am\0.0.1-SNAPSHOT\am-0.0.1-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 31.526 s
[INFO] Finished at: 2022-10-15T12:39:18+02:00
[INFO] -----
```

APLICACIÓN WEB SPRING BOOT

- **3er paso: Clase principal:**
- **Toda aplicación en java debe contener una clase principal con un método main. Dicho método, en caso de implementar una aplicación con Spring, deberá llamar al método run de la clase SpringApplication.**

APLICACIÓN WEB SPRING BOOT

```
1 @Configuration
2 @EnableAutoConfiguration
3 @ComponentScan
4 public class Application {
5
6     public static void main(String[] args) throws Exception {
7         SpringApplication.run(Application.class, args);
8     }
9 }
```

APLICACIÓN WEB SPRING BOOT

- **@Configuration**: indica que la clase en la que se encuentra contiene la configuración principal del proyecto.
- **@EnableAutoConfiguration**: indica que se aplicará la configuración automática del starter que hemos utilizado. Solo debe añadirse en un sitio, y es muy frecuente situarla en la clase main.
- **@ComponentScan**: ayuda a localizar elementos etiquetados con otras anotaciones cuando sean necesarios.
- **@SpringBootApplication**: engloba las anteriores, por lo que es más simple poner solo esta.

APLICACIÓN WEB SPRING BOOT

```
1 @SpringBootApplication
2 public class Application {
3
4     public static void main(String[] args) throws Exception {
5         SpringApplication.run(Application.class, args);
6     }
7 }
```

APLICACIÓN WEB SPRING BOOT

- Para poder desplegar la aplicación Web se necesita extender de ***SpringBootServletInitializer***.

@SpringBootApplication

```
public class AmApplication extends SpringBootServletInitializer {
```

```
    @Override
```

```
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
```

```
        return builder.sources(AmApplication.class);
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(AmApplication.class, args);
```

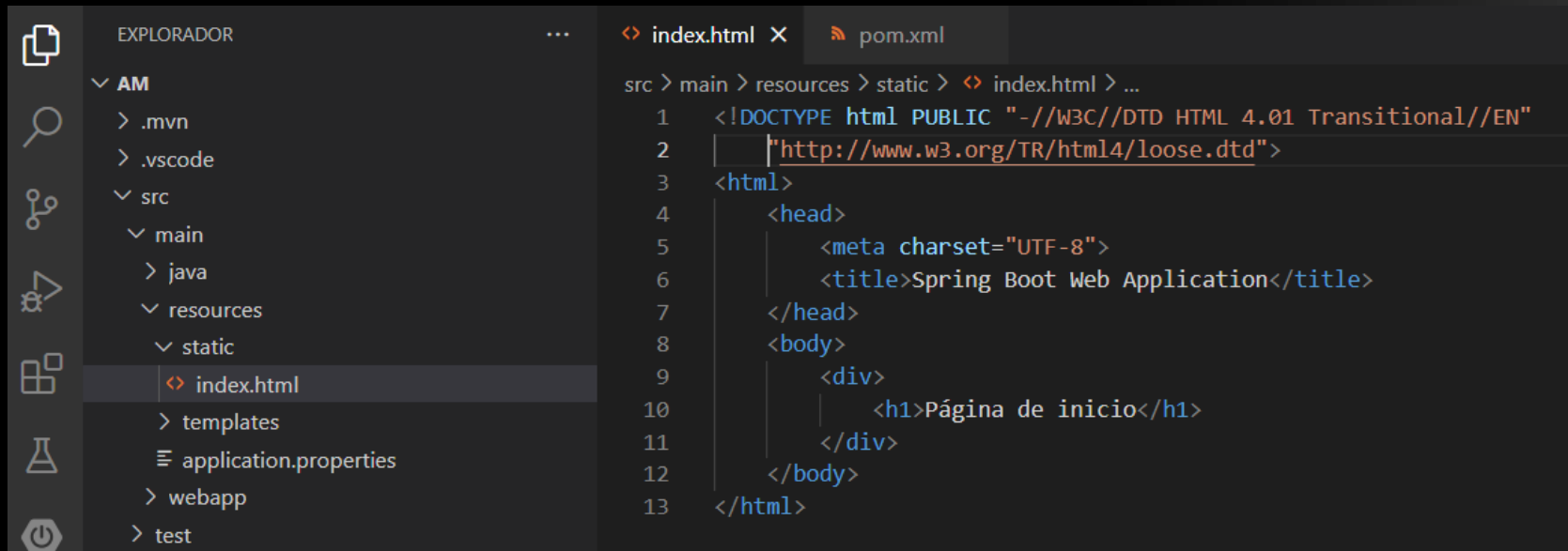
```
    }
```

```
}
```

APLICACIÓN WEB SPRING BOOT

- 4º paso: index
- La página de inicio será un *index.html*, que por defecto, en la ruta por defecto de la aplicación será la página que se vea:

APLICACIÓN WEB SPRING BOOT



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure:

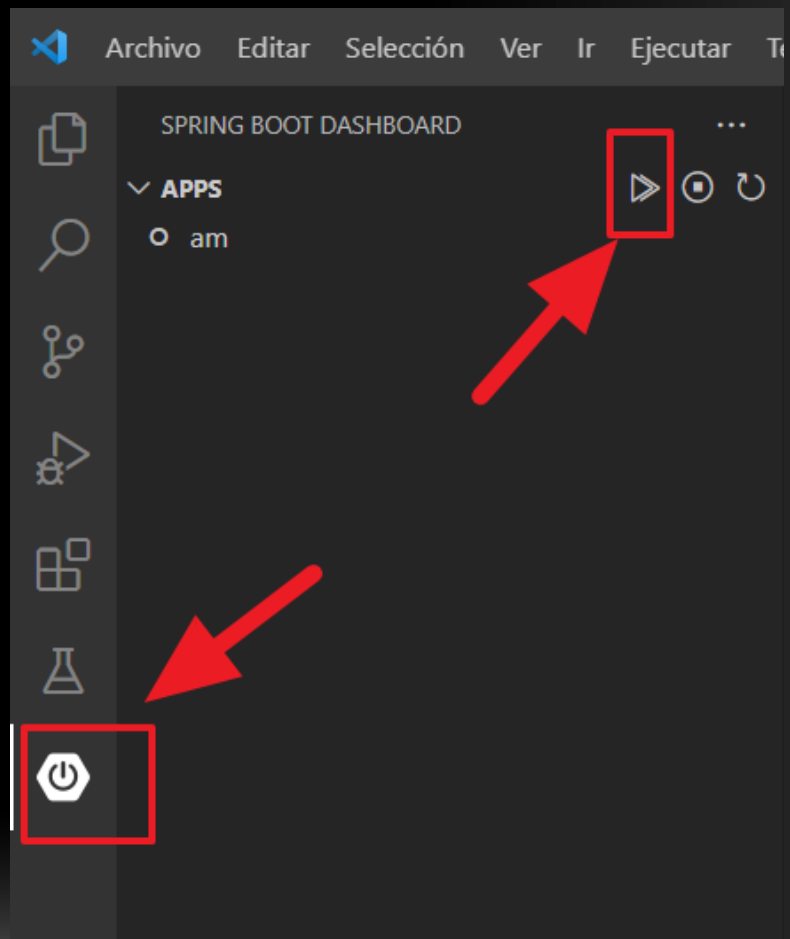
- EXPLORADOR
 - AM
 - .mvn
 - .vscode
 - src
 - main
 - java
 - resources
 - static
 - index.html**
 - templates
 - application.properties
 - webapp
 - test

The main editor area shows the content of `index.html`:

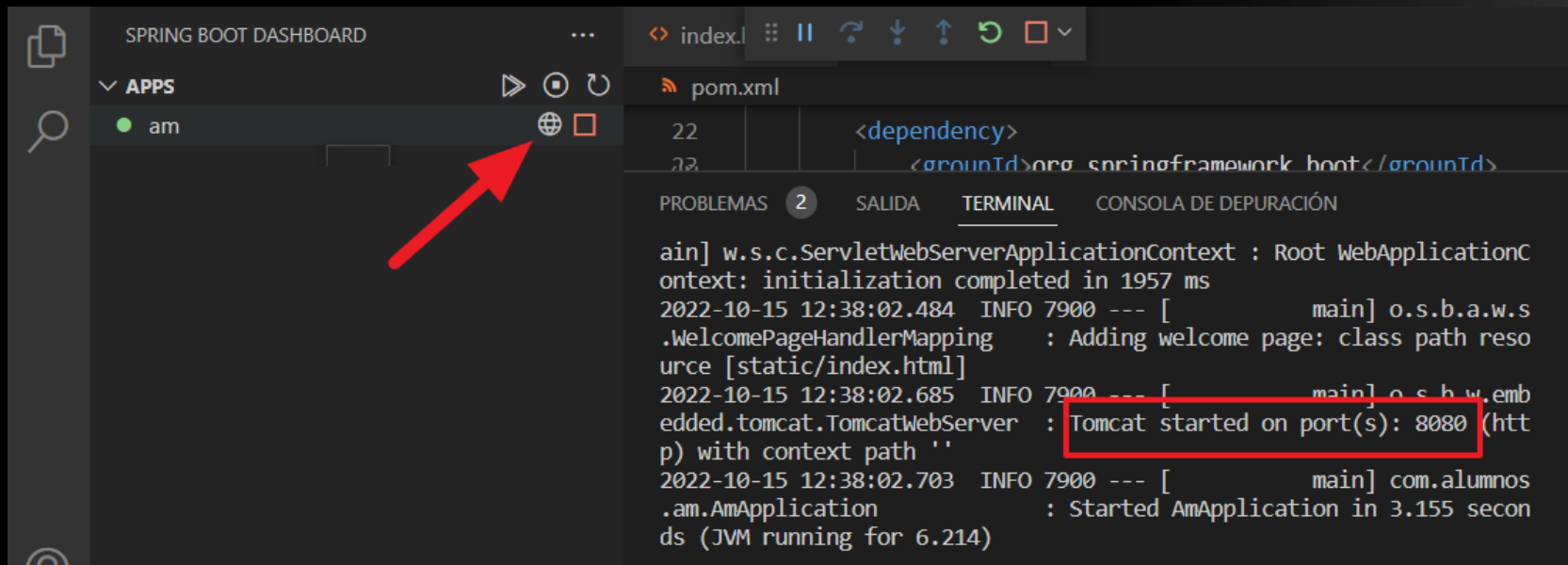
```
src > main > resources > static > index.html > ...
1  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2  | "http://www.w3.org/TR/html4/loose.dtd">
3  <html>
4  |   <head>
5  |       <meta charset="UTF-8">
6  |       <title>Spring Boot Web Application</title>
7  |   </head>
8  |   <body>
9  |       <div>
10 |           <h1>Página de inicio</h1>
11 |       </div>
12 |   </body>
13 </html>
```


APLICACIÓN WEB SPRING BOOT

- Si se ejecuta la aplicación lo que se verá será esta página de inicio:



APLICACIÓN WEB SPRING BOOT



SPRING BOOT DASHBOARD

APPS

am

Run

index.html

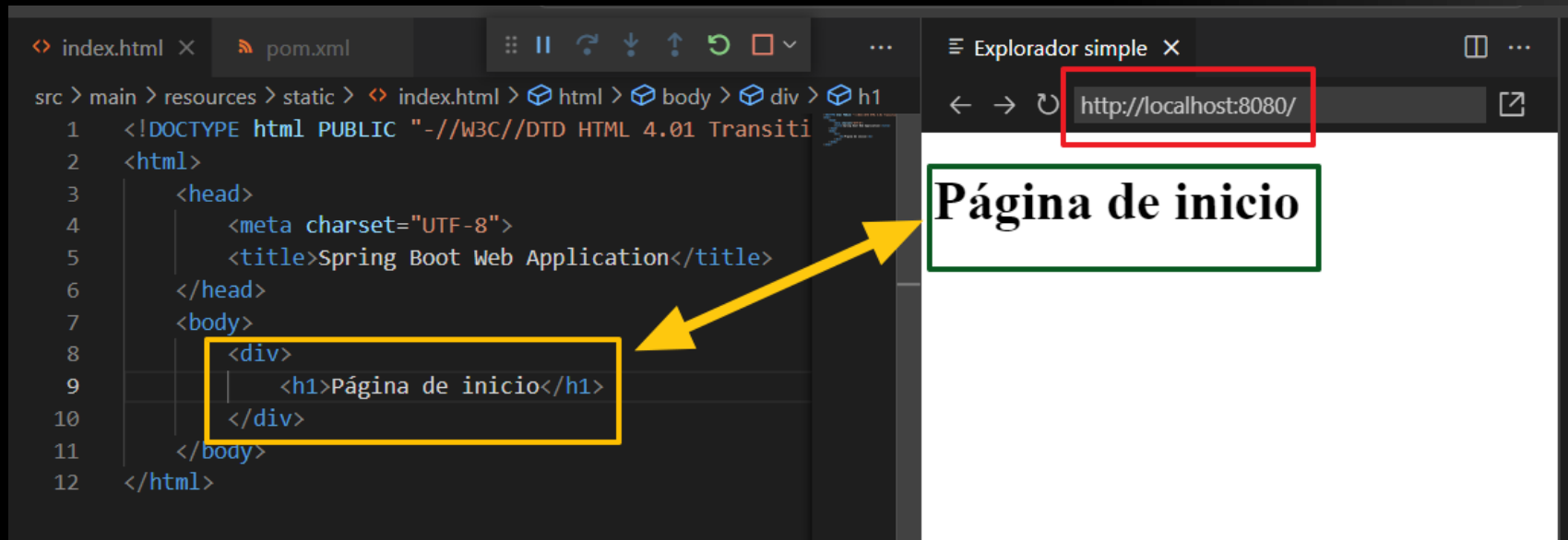
pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
```

PROBLEMAS 2 SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
ain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationC
ontext: initialization completed in 1957 ms
2022-10-15 12:38:02.484 INFO 7900 --- [main] o.s.b.a.w.s
.WelcomePageHandlerMapping : Adding welcome page: class path reso
urce [static/index.html]
2022-10-15 12:38:02.685 INFO 7900 --- [main] o.s.b.w.emb
edded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (htt
p) with context path ''
2022-10-15 12:38:02.703 INFO 7900 --- [main] com.alumnos
.am.AmApplication : Started AmApplication in 3.155 secon
ds (JVM running for 6.214)
```

APLICACIÓN WEB SPRING BOOT



APLICACIÓN WEB SPRING BOOT

- La clase *controller* es la que maneja la navegación entre páginas:
- Hay dos formas de “pintar” la página:
 - **RestController:** El método devuelve un String que es el Html.
 - **Controller:** El método devuelve la dirección de la JSP que se quiere abrir.

APLICACIÓN WEB SPRING BOOT

```
import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@RestController

public class AmRestController {

    @RequestMapping(value="/URL_Rest", method = RequestMethod.GET)

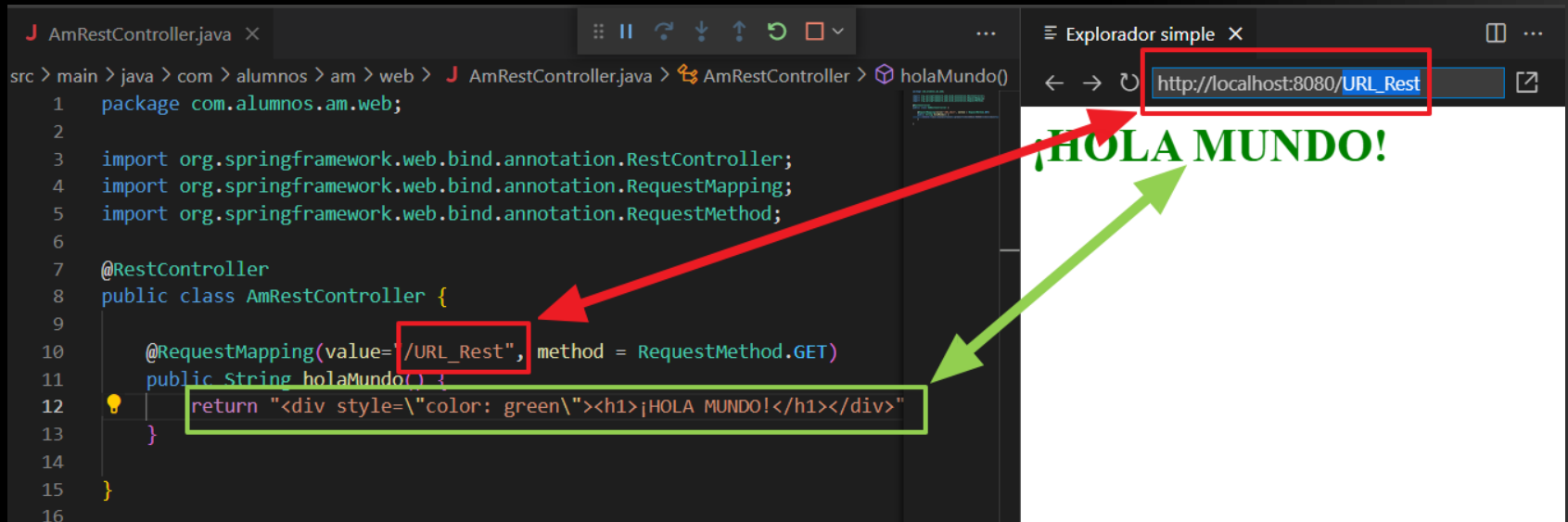
    public String holaMundo() {

        return "<div style=\"color: red\"><h1>¡HOLA MUNDO!</h1></div>";

    }

}
```

APLICACIÓN WEB SPRING BOOT



The image shows a development environment with two windows. The left window is a code editor showing the file `AmRestController.java` with the following code:

```
1 package com.alumnos.am.web;
2
3 import org.springframework.web.bind.annotation.RestController;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RequestMethod;
6
7 @RestController
8 public class AmRestController {
9
10     @RequestMapping(value="/URL_Rest", method = RequestMethod.GET)
11     public String holaMundo() {
12         return "<div style=\"color: green\"><h1>¡HOLA MUNDO!</h1></div>";
13     }
14
15 }
16
```

The right window is a web browser titled "Explorador simple" showing the URL `http://localhost:8080/URL_Rest`. The browser displays the text **¡HOLA MUNDO!** in green. A red arrow points from the `URL_Rest` value in the code to the browser's address bar. A green arrow points from the `return` statement in the code to the displayed text in the browser.

APLICACIÓN WEB SPRING BOOT

- El fichero *controller*, maneja las llamadas igual que el *RestController*, pero lo que devuelve es la dirección de la JSP que se quiere cargar.

APLICACIÓN WEB SPRING BOOT

- En el fichero *application.properties*, localizado en el directorio *resources*, se debe añadir la información de las JSP que va a usar el proyecto.
- Para no tener que escribir todas las JSP se pone lo siguiente:

```
spring.mvc.view.prefix=/WEB-INF/jsp/  
spring.mvc.view.suffix=.jsp
```


APLICACIÓN WEB SPRING BOOT

- **@Controller:** Con esta anotación Spring podrá detectar la clase `SampleController` cuando realice el escaneo de componentes.
- **@Autowired:** A través de esta anotación Spring será capaz de llevar a cabo la inyección de dependencias sobre el atributo marcado. En este caso, estamos inyectando la capa de servicio, y por eso no tenemos que instanciarla.
- **@RequestMapping:** Con esta anotación especificamos la ruta desde la que escuchará el servicio, y qué método le corresponde.
- **@ResponseBody:** Con ella definimos lo que será el cuerpo de la respuesta del servicio.
- **@PathVariable:** Sirve para indicar con qué variable de la url se relaciona el parámetro sobre el que se esté usando la anotación.
- También se puede usar la etiqueta **@RestController** en lugar de **@Controller**, que sustituye al uso de **@Controller** + **@ResponseBody**, quedando el controlador de la siguiente forma:

APLICACIÓN WEB SPRING BOOT

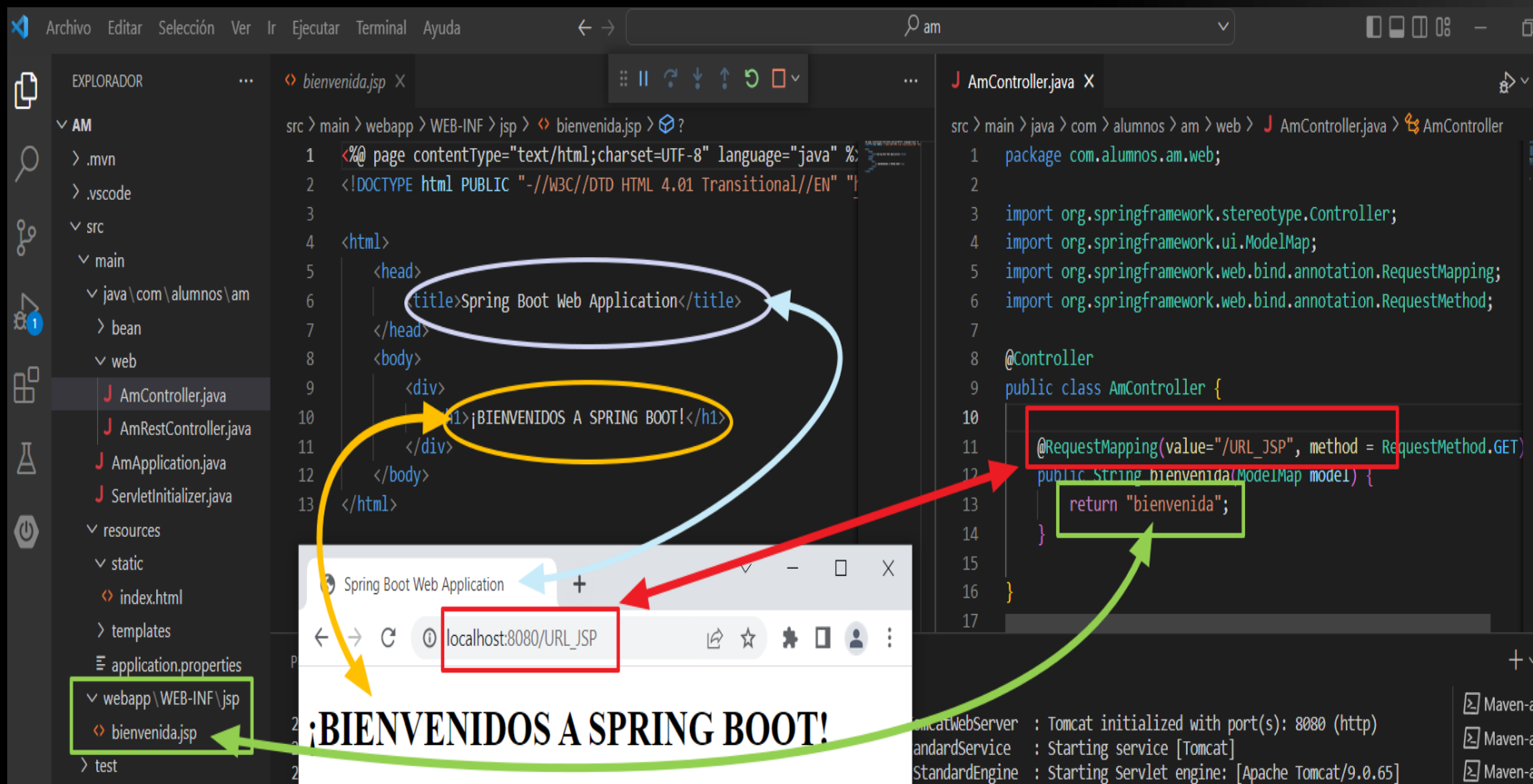
```
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class AmController {

    @RequestMapping(value="/URL_JSP", method = RequestMethod.GET)
    public String bienvenida(ModelMap model) {
        return "bienvenida";
    }

}
```

APLICACIÓN WEB SPRING BOOT



src > main > webapp > WEB-INF > jsp > bienvenida.jsp > ?

```
1 <%@ page contentType="text/html;charset=UTF-8" language="java" %>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
3
4 <html>
5   <head>
6     <title>Spring Boot Web Application</title>
7   </head>
8   <body>
9     <div>
10      <h1>BIENVENIDOS A SPRING BOOT!</h1>
11    </div>
12  </body>
13 </html>
```

src > main > java > com > alumnos > am > web > J AmController.java > AmController

```
1 package com.alumnos.am.web;
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
8 @Controller
9 public class AmController {
10
11   @RequestMapping(value="/URL_JSP", method = RequestMethod.GET)
12   public String bienvenida(ModelMap model) {
13     return "bienvenida";
14   }
15
16 }
17
```

Spring Boot Web Application

localhost:8080/URL_JSP

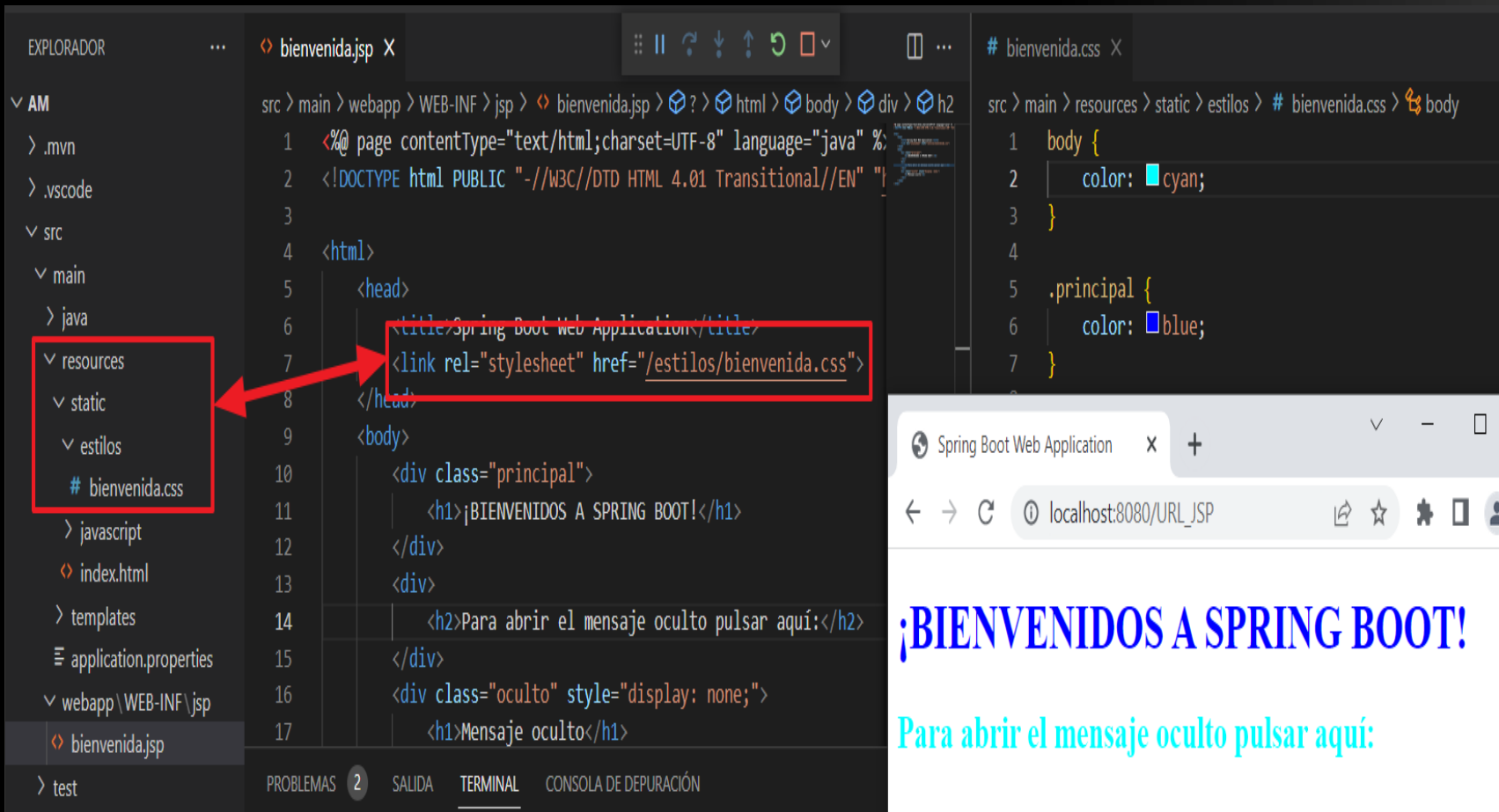
BIENVENIDOS A SPRING BOOT!

TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
StandardService : Starting service [Tomcat]
StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]

APLICACIÓN WEB SPRING BOOT

- En esta JSP se puede añadir contenido estático, como los estilos (css), javascript o imágenes.

APLICACIÓN WEB SPRING BOOT



The screenshot displays an IDE with the following components:

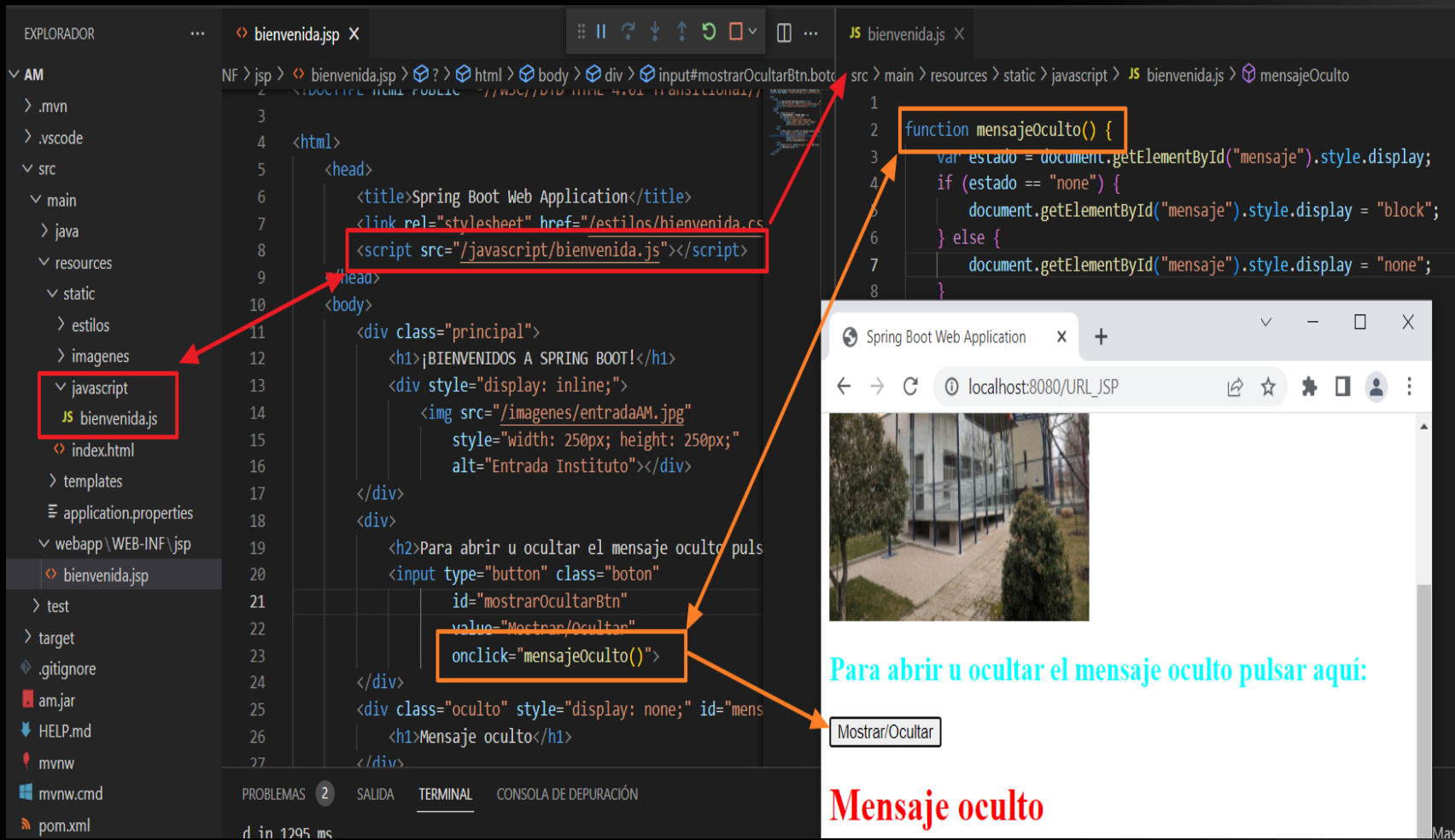
- EXPLORADOR (Left):** A file explorer showing the project structure. The `resources` folder is expanded, showing `static`, `estilos`, and `bienvenida.css`. A red box highlights the `estilos` folder and `bienvenida.css`, with a red arrow pointing to the `href` attribute in the `bienvenida.jsp` file.
- bienvenida.jsp (Center):** The main JSP file. It contains the following code:

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/strict.dtd">
3
4 <html>
5   <head>
6     <title>Spring Boot Web Application</title>
7     <link rel="stylesheet" href="/estilos/bienvenida.css">
8   </head>
9   <body>
10    <div class="principal">
11      <h1>BIENVENIDOS A SPRING BOOT!</h1>
12    </div>
13    <div>
14      <h2>Para abrir el mensaje oculto pulsar aquí:</h2>
15    </div>
16    <div class="oculto" style="display: none;">
17      <h1>Mensaje oculto</h1>
18    </div>
19  </body>
20 </html>
```
- bienvenida.css (Right):** The CSS file. It contains the following code:

```
1 body {
2   color: cyan;
3 }
4
5 .principal {
6   color: blue;
7 }
```
- Browser Preview (Bottom Right):** A browser window titled "Spring Boot Web Application" showing the rendered page at `localhost:8080/URL_JSP`. The page displays:
 - A large blue heading: **BIENVENIDOS A SPRING BOOT!**
 - A cyan text link: **Para abrir el mensaje oculto pulsar aquí:**

"Un itinerario con éxito..."

APLICACIÓN WEB SPRING BOOT



The image shows a development environment for a Spring Boot web application. The file explorer on the left shows the project structure, with the `javascript` folder and `bienvenida.js` file highlighted. The main editor displays the `bienvenida.jsp` file, which includes a `<script src="/javascript/bienvenida.js"></script>` tag and a button with `onclick="mensajeOculto()"`. The right panel shows the `bienvenida.js` file with a `mensajeOculto()` function. A browser preview at the bottom right shows the application running at `localhost:8080/URL_JSP`, displaying a button labeled "Mostrar/Ocultar" and a hidden message.

HTML Code (bienvenida.jsp):

```
<html>
<head>
<title>Spring Boot Web Application</title>
<link rel="stylesheet" href="/estilos/bienvenida.css">
<script src="/javascript/bienvenida.js"></script>
</head>
<body>
<div class="principal">
<h1>¡BIENVENIDOS A SPRING BOOT!</h1>
<div style="display: inline;">
</div>
</div>
<div>
<h2>Para abrir u ocultar el mensaje oculto puls
<input type="button" class="boton"
id="mostrarOcultarBtn"
value="Mostrar/Ocultar"
onclick="mensajeOculto()">
</div>
<div class="oculto" style="display: none;" id="mens
<h1>Mensaje oculto</h1>
</div>
```

JavaScript Code (bienvenida.js):

```
function mensajeOculto() {
var estado = document.getElementById("mensaje").style.display;
if (estado == "none") {
document.getElementById("mensaje").style.display = "block";
} else {
document.getElementById("mensaje").style.display = "none";
}
```

Browser Preview:

Spring Boot Web Application x +

localhost:8080/URL_JSP

Para abrir u ocultar el mensaje oculto pulsar aquí:

Mostrar/Ocultar

Mensaje oculto

APLICACIÓN WEB SPRING BOOT

- Una vez implementada la parte estática vamos a ver cómo navegar entre ventanas por la aplicación.
- En la JSP se puede poner un link para que vaya a otra ventana o JSP:

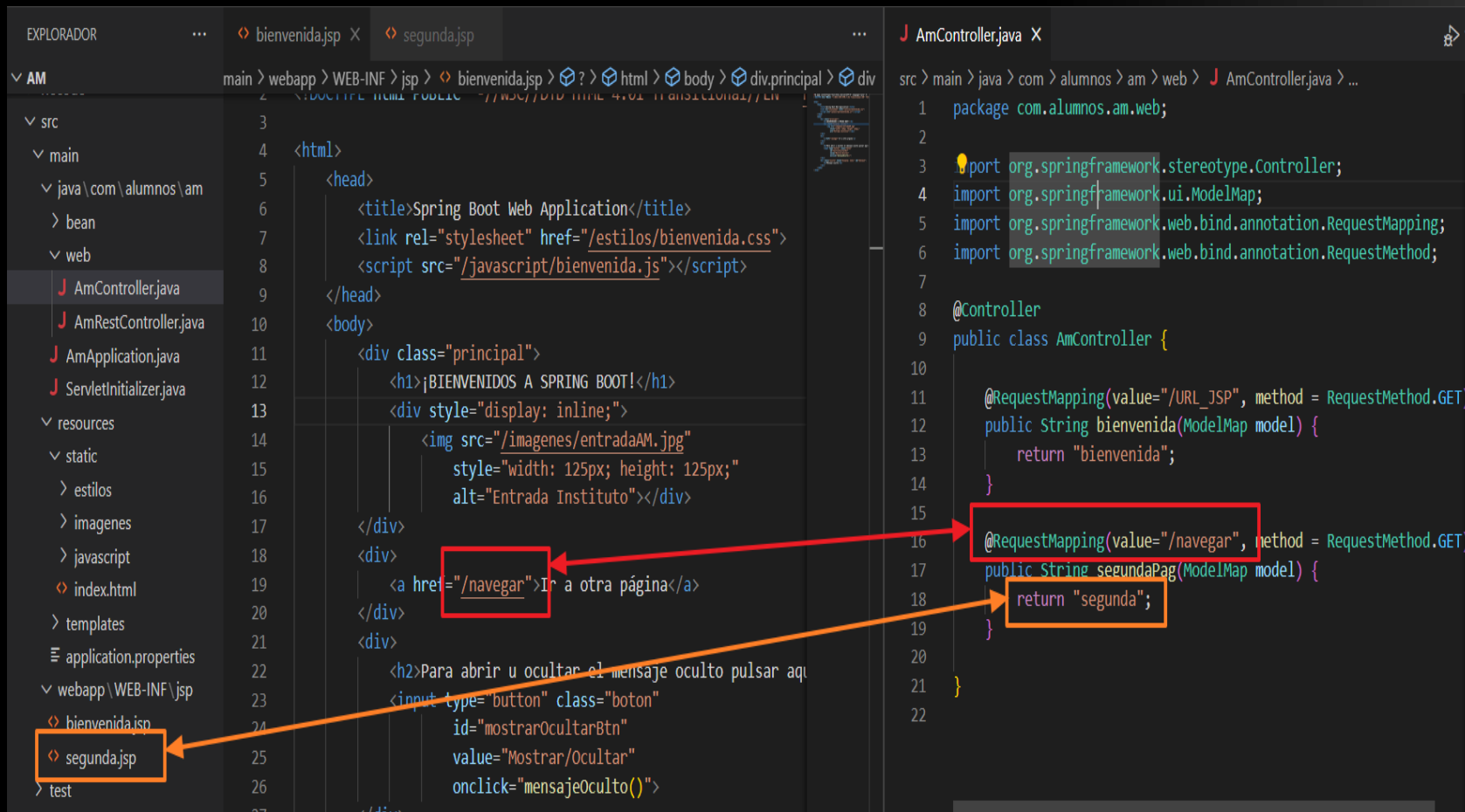
`Ir a otra página`

APLICACIÓN WEB SPRING BOOT

- Para que la aplicación entienda esa llamada debe crearse un método en el Controller que reciba esa llamada “/navegar” y que indique a qué JSP se redirige la aplicación:

```
@RequestMapping(value="/navegar", method = RequestMethod.GET)
public String segundaPag(ModelMap model) {
    return "segunda";
}
```

APLICACIÓN WEB SPRING BOOT



EXPLORADOR

main > webapp > WEB-INF > jsp > bienvenida.jsp > ? > html > body > div.principal > div

src

main

java \com \alumnos \am

bean

web

AmController.java

AmRestController.java

AmApplication.java

ServletInitializer.java

resources

static

estilos

imagenes

javascript

index.html

templates

application.properties

webapp \WEB-INF \jsp

bienvenida.jsp

segunda.jsp

test

```
<html>
<head>
<title>Spring Boot Web Application</title>
<link rel="stylesheet" href="/estilos/bienvenida.css">
<script src="/javascript/bienvenida.js"></script>
</head>
<body>
<div class="principal">
<h1>¡BIENVENIDOS A SPRING BOOT!</h1>
<div style="display: inline;">
</div>
</div>
<div>
<a href="/navegar">Ir a otra página</a>
</div>
<div>
<h2>Para abrir u ocultar el mensaje oculto pulsar aquí
<input type="button" class="boton"
id="mostrarOcultarBtn"
value="Mostrar/Ocultar"
onclick="mensajeOculto()">
</div>
</div>
```

AmController.java

```
package com.alumnos.am.web;

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

@Controller
public class AmController {

    @RequestMapping(value="/URL_JSP", method = RequestMethod.GET)
    public String bienvenida(ModelMap model) {
        return "bienvenida";
    }

    @RequestMapping(value="/navegar", method = RequestMethod.GET)
    public String segundaPag(ModelMap model) {
        return "segunda";
    }
}
```

APLICACIÓN WEB SPRING BOOT

- Intercambiar información entre el controller y la JSP:
- Vamos a crear un *controller* que maneje esto y una JSP con un formulario.
- Para acceder a esta JSP se añade un enlace a esta JSP (*formulario.jsp*) desde *index.html* o desde la JSP que se quiera:

```
<a href="/abrirFormulario">Formulario</a>
```

APLICACIÓN WEB SPRING BOOT

- En el *controller* se añade un método que reciba la llamada y que redirija a esta JSP:

@Controller

```
public class FormularioController {
```

```
    @RequestMapping(value="/abrirFormulario", method = RequestMethod.GET)
```

```
    public String irAFormulario() {
```

```
        return "formulario";
```

```
    }
```

```
}
```

APLICACIÓN WEB SPRING BOOT

- También se crea un método que reciba los datos del formulario y los trate.
- Para ello se usa la siguiente anotación:
 - `@RequestParam`
- Esta anotación va con los parámetros del método.

APLICACIÓN WEB SPRING BOOT

```
@RequestMapping(value = "/abrirFormulario", method = RequestMethod.POST)
public String crearMensaje(ModelMap model,
    @RequestParam String nombre,
    @RequestParam String apellido) {
    StringBuilder mensajeAEnviar = new StringBuilder("¡Hola ");
    mensajeAEnviar.append(nombre);
    mensajeAEnviar.append(" ");
    mensajeAEnviar.append(apellido);
    mensajeAEnviar.append("!");
    model.put("mensaje", mensajeAEnviar);
    return "formulario";
}
```

APLICACIÓN WEB SPRING BOOT

- En el RequestMapping está la URL de la JSP del formulario, y en el return se vuelve a poner la JSP del formulario, ya que no se sale de esta ventana.
- Los datos que se quieran pasar a la JSP se insertan en el ModelMap:

```
model.put(key, value);
```

APLICACIÓN WEB SPRING BOOT

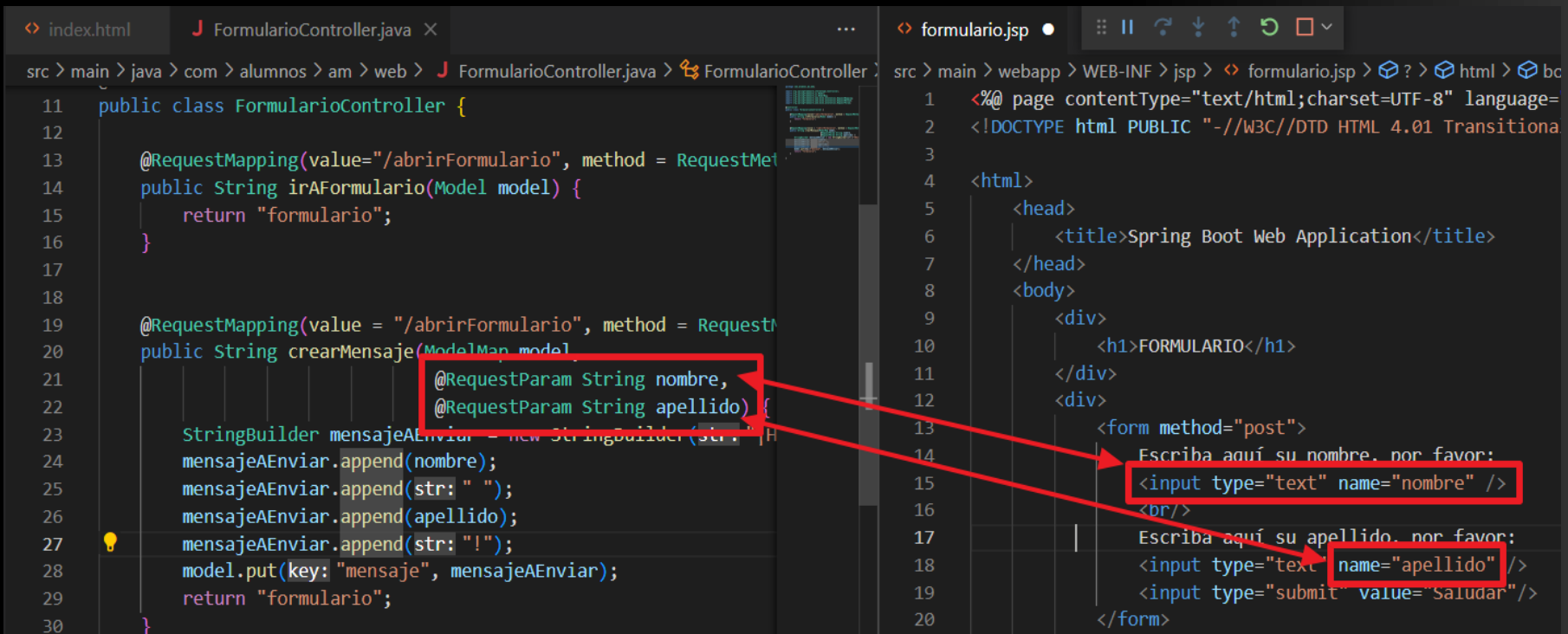
- En la JSP hay que añadir los siguientes tags:

```
<form method="post">
```

```
<input type="text" name="key" />
```
- Todo lo que haya dentro de este *form* irá por POST al hacer submit.
- El método “post” o “get” debe coincidir con el RequestMethod del método del Controller que recibe la llamada.
- Es recomendable siempre pasar los datos de un formulario por POST.

APLICACIÓN WEB SPRING BOOT

- La key del input es la key del *RequestParam* del Controller:



```
src > main > java > com > alumnos > am > web > J FormularioController.java > FormularioController
11 public class FormularioController {
12
13     @RequestMapping(value="/abrirFormulario", method = RequestMethod.GET)
14     public String irAFormulario(Model model) {
15         return "formulario";
16     }
17
18
19     @RequestMapping(value = "/abrirFormulario", method = RequestMethod.POST)
20     public String crearMensaje(ModelMap model) {
21         @RequestParam String nombre,
22         @RequestParam String apellido) {
23
24         StringBuilder mensajeAEnviar = new StringBuilder("H");
25         mensajeAEnviar.append(nombre);
26         mensajeAEnviar.append(str: " ");
27         mensajeAEnviar.append(apellido);
28         mensajeAEnviar.append(str: "!");
29         model.put(key: "mensaje", mensajeAEnviar);
30         return "formulario";
31     }
32 }
```

```
src > main > webapp > WEB-INF > jsp > formulario.jsp
1 <%@ page contentType="text/html; charset=UTF-8" language=
2 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional
3
4 <html>
5     <head>
6         <title>Spring Boot Web Application</title>
7     </head>
8     <body>
9         <div>
10             <h1>FORMULARIO</h1>
11         </div>
12         <div>
13             <form method="post">
14                 Escriba aquí su nombre. por favor:
15                 <input type="text" name="nombre" />
16                 <br/>
17                 Escriba aquí su apellido. por favor:
18                 <input type="text" name="apellido" />
19                 <input type="submit" value="Saludar"/>
20             </form>
21         </div>
22     </body>
23 </html>
```

APLICACIÓN WEB SPRING BOOT

- Otra forma de hacerlo es, en la JSP añadir además el tag “*action*”:

```
<form method="post" action="Accion">
```

```
<input type="text" name="key" />
```

- El valor que se ponga en el *action* debe ser el que se ponga como valor en el RequestMapping del Controller, es decir, que en el tag *action* se pone la URL a la que se hace POST y que recibe el Controller.

APLICACIÓN WEB SPRING BOOT

```
FormularioController.java X
src > main > java > com > alumnos > am > web > J FormularioController.java > FormularioController
11 public class FormularioController {
12
13     @RequestMapping(value="/abrirFormulario", method = RequestMethod.GET)
14     public String irAFormulario(Model model) {
15         return "formulario";
16     }
17
18     @RequestMapping(value = "/actionForm", method = RequestMethod.POST)
19     public String crearMensaje(Model model,
20                               @RequestParam String nombre,
21                               @RequestParam String apellido) {
22         StringBuilder mensajeAEnviar = new StringBuilder("¡Hola ");
23         mensajeAEnviar.append(nombre);
24         mensajeAEnviar.append(" ");
25         mensajeAEnviar.append(apellido);
26         mensajeAEnviar.append("!");
27         model.put("mensaje", mensajeAEnviar);
28         return "formulario";
29     }
30 }
31
32
33
```

```
formulario.jsp X
src > main > webapp > WEB-INF > jsp > formulario.jsp > ? > html > body > div
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "%
3
4 <html>
5     <head>
6         <title>Spring Boot Web Application</title>
7     </head>
8     <body>
9         <div>
10             <h1>FORMULARIO</h1>
11         </div>
12         <div>
13             <form method="post" action="actionForm">
14                 Escriba aquí su nombre, por favor: <input type="text" />
15                 <br/>
16                 Escriba aquí su apellido, por favor: <input type="text" />
17                 <input type="submit" value="Saludar" />
18             </form>
19             <h1><font color="red">${mensaje}</font></h1>
20         </div>
21         <div>
22             <h3><a href="/">Salir</a></h3>
23         </div>
24     </body>
25 </html>
```

APLICACIÓN WEB SPRING BOOT

- Para escribir los valores que se pasan desde el Controller a la JSP en la JSP se debe poner de la siguiente manera:

`${key}`

APLICACIÓN WEB SPRING BOOT

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>

  <head>
    <title>Spring Boot Web Application</title>
  </head>
  <body>
    <div>
      <h1>FORMULARIO</h1>
    </div>
    <div>
      <form method="post">
        Escriba aquí su nombre, por favor: <input type="text" name="nombre" />
        <br/>
        Escriba aquí su apellido, por favor: <input type="text" name="apellido" />
        <input type="submit" value="Saludar"/>
      </form>
    </div>
  </body>
</html>
```

APLICACIÓN WEB SPRING BOOT

```
<h1><font color="red">${mensaje}</font></h1>  
</div>  
<div>  
  <h3><a href="/">Salir</a></h3>  
</div>  
</body>  
</html>
```

APLICACIÓN WEB SPRING BOOT

index.html

FormularioController.java

src > main > java > com > alumnos > am > web > J FormularioController.java > FormularioController

```
11 public class FormularioController {
12
13     @RequestMapping(value="/abrirFormulario", method = RequestMethod.GET)
14     public String irAFormulario(Model model) {
15         return "formulario";
16     }
17
18
19     @RequestMapping(value = "/abrirFormulario", method = RequestMethod.POST)
20     public String crearMensaje(ModelMap model,
21                               @RequestParam String nombre,
22                               @RequestParam String apellido) {
23         StringBuilder mensajeAEnviar = new StringBuilder(str: "¡H
24         mensajeAEnviar.append(nombre);
25         mensajeAEnviar.append(str: " ");
26         mensajeAEnviar.append(apellido);
27         mensajeAEnviar.append(str: "!");
28         model.put(key: "mensaje", mensajeAEnviar);
29         return "formulario";
30     }
31 }
32
33
```

formulario.jsp

src > main > webapp > WEB-INF > jsp > formulario.jsp > ? > html > body

```
1 <%@ page contentType="text/html; charset=UTF-8" language="ja
2 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
3
4 <html>
5     <head>
6         <title>Spring Boot Web Application</title>
7     </head>
8     <body>
9         <div>
10             <h1>FORMULARIO</h1>
11         </div>
12         <div>
13             <form method="post">
14                 Escriba aquí su nombre, por favor:
15                 <input type="text" name="nombre" />
16                 <br/>
17                 Escriba aquí su apellido, por favor:
18                 <input type="text" name="apellido" />
19                 <input type="submit" value="Saludar"/>
20             </form>
21             <h1><font color="red">${mensaje}</font></h1>
22         </div>
23         <h3><a href="/">Salir</a></h3>
24     </div>
25
```

Spring Boot Web Application

localhost:8080/abrirFormulario

Incógnito

PROBLEMAS 2

FORMULARIO

2022-10-19 19:15:36 ms
dex.html]

2022-10-19 19:15:36 ms
path ''

2022-10-19 19:15:36 ms
for 3.384)

2022-10-19 19:15:36 ms
rvlet'

2022-10-19 19:15:36 ms

2022-10-19 19:15:36 ms

Escriba aquí su nombre, por favor:

Escriba aquí su apellido, por favor:

Saludar

¡Hola Clase ASIR!

[Salir](#)

Adding welcome page: class path resource [static/in

Tomcat started on port(s): 8080 (http) with context

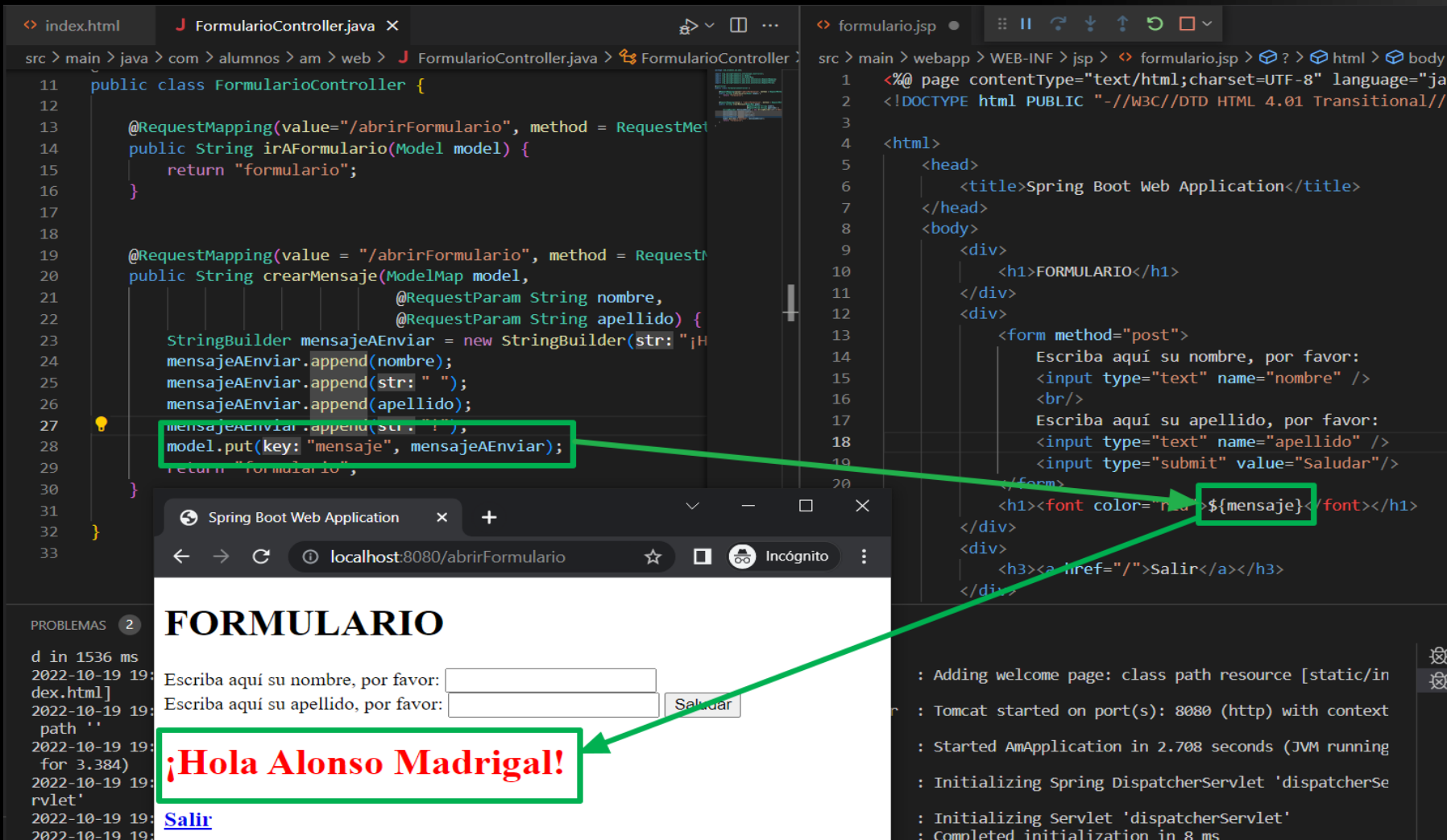
Started AmApplication in 2.708 seconds (JVM running

Initializing Spring DispatcherServlet 'dispatcherSe

Initializing Servlet 'dispatcherServlet'

Completed initialization in 8 ms

APLICACIÓN WEB SPRING BOOT



The screenshot displays the development environment for a Spring Boot web application. It shows three main components:

- FormularioController.java:** The Java controller class. It has two methods: `irAFormulario` and `crearMensaje`. The `crearMensaje` method uses `@RequestParam` to capture `nombre` and `apellido`, constructs a `mensajeAEnviar` string, and stores it in the model using `model.put(key: "mensaje", mensajeAEnviar);`. This line is highlighted with a green box.
- formulario.jsp:** The JSP template. It contains HTML for a form with input fields for `nombre` and `apellido`, and a submit button. Below the form, it displays the message using `<h1>${mensaje}</h1>`. The `${mensaje}` expression is highlighted with a green box.
- Spring Boot Web Application:** The browser view at `localhost:8080/abrirFormulario`. It shows the rendered HTML with the title "FORMULARIO" and the message "¡Hola Alonso Madrigal!". The message is highlighted with a green box. The browser also shows a "Salir" link.

Arrows indicate the data flow: from the `mensajeAEnviar` variable in the controller to the `${mensaje}` expression in the JSP, and finally to the rendered message in the browser.

APLICACIÓN WEB SPRING BOOT

- **5. Servicio:**
- **Un método de servicio definirá una operación a nivel de negocio, por ejemplo, dar un mensaje de bienvenida.**
- **Los métodos de servicio estarán formados por otras operaciones más pequeñas, las cuales estarán definidas en la capa de repositorio.**

APLICACIÓN WEB SPRING BOOT

- Para indicar que una clase se va a usar como servicio de Spring Boot hay que marcarla como tal con la anotación `@Service`.
- La anotación `@Service` funciona de forma parecida a la anotación `@Controller`, ya que permite que Spring reconozca a `DemoService` como servicio al escanear los componentes de la aplicación.

APLICACIÓN WEB SPRING BOOT

- Para poder acceder al servicio debe hacerse desde una interfaz pública:

```
public interface AmServicio {  
    String crearMensaje(String nombre, String apellido);  
}
```

- Y el implementador correspondiente:

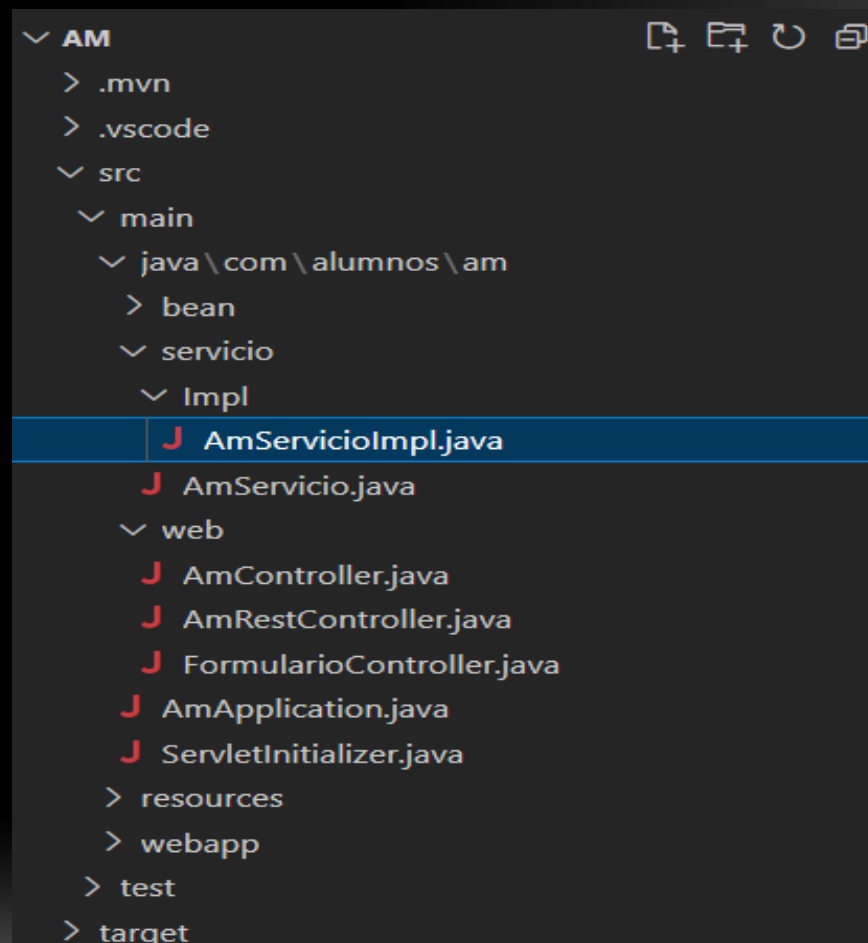
APLICACIÓN WEB SPRING BOOT

@Service

```
public class AmServicioImpl implements AmServicio {  
  
    public String crearMensaje(String nombre, String apellido) {  
        StringBuilder mensajeAEnviar = new StringBuilder("¡Hola ");  
        mensajeAEnviar.append(nombre);  
        mensajeAEnviar.append(" ");  
        mensajeAEnviar.append(apellido);  
        mensajeAEnviar.append("!");  
        return mensajeAEnviar.toString();  
    }  
}
```

APLICACIÓN WEB SPRING BOOT

- La estructura de carpetas sería algo como lo siguiente:



APLICACIÓN WEB SPRING BOOT

- Para poder usar el servicio, es decir, que el Controller pueda llamarlo, se inyecta el servicio en el controller mediante la anotación `@Autowired`

`@Autowired`

`private [nombre de la clase Servicio] [nombre de la variable];`

APLICACIÓN WEB SPRING BOOT

@Autowired

private AmServicio servicio;

@RequestMapping(value = "/actionForm", method = RequestMethod.POST)

public String crearMensaje(ModelMap model,
 @RequestParam String nombre,
 @RequestParam String apellido) {

String mensajeCreado = servicio.crearMensaje(nombre, apellido);

model.put("mensaje", mensajeCreado);

return "formulario";

}

APLICACIÓN WEB SPRING BOOT

- Para no tener que pasar todos los parámetros uno a uno es mejor crear un objeto que se pueda mover entre capas con todos los datos.
- Para ello se crea un Bean con las variables privadas que se tengan que usar.

APLICACIÓN WEB SPRING BOOT

- La estructura de carpetas sería algo como lo siguiente:

```
▼ AM
  > .mvn
  > .vscode
  ▼ src
    ▼ main
      ▼ java \ com \ alumnos \ am
        ▼ bean
          J AmBean.java
        > servicio
        > web
          J AmApplication.java
          J ServletInitializer.java
        > resources
        > webapp
```

APLICACIÓN WEB SPRING BOOT

- En esta clase se crean las variables que se quieren trasladar entre capas, es decir, los datos del formulario que se quieren llevar hasta la base de datos y viceversa.

APLICACIÓN WEB SPRING BOOT

```
public class AmBean {  
  
    private String nombre;  
    private String apellido;  
    private String mensaje;  
  
}
```

APLICACIÓN WEB SPRING BOOT

- Como las variables son privadas, para poder informarlas y que se puedan consultar sus valores hay que generar sus getters y setters.
- El get es para obtener el valor de la variable.
- El set es para informar el valor de la variable.

APLICACIÓN WEB SPRING BOOT

- Desde el Visual Studio Code, con el botón secundario del ratón sobre la clase se selecciona *“Acción de código fuente”* / *“Source action”*.
- Después se selecciona *“Generate getters and setters”*.
- Se seleccionan todas las variables y se pulsa OK.

APLICACIÓN WEB SPRING BOOT

J AmBean.java 3

```
src > main > java > com > alumnos > am > bean > J AmBean.java > ...  
1 package com.alumnos.am.bean;  
2  
3 public class AmBean {  
4  
5     private String nombre;  
6     private String apellido;  
7     private String mensaje;  
8  
9 }  
10
```

Ir a definición	F12
Ir a la definición de tipo	
Ir a Implementaciones	Ctrl+F12
Ir a Referencias	Mayús+F12
Go to Super Implementation	
Go to Test	
Ver	>
Buscar todas las referencias	Mayús+Alt+F12
Buscar todas las implementaciones	
Mostrar jerarquía de llamadas	Mayús+Alt+H
Show Type Hierarchy	
Cambiar el nombre del símbolo	F2
Cambiar todas las ocurrencias	Ctrl+F2
Dar formato al documento	Mayús+Alt+F
Dar formato al documento con...	
Refactorizar...	Ctrl+Mayús+R
Acción de código fuente...	

APLICACIÓN WEB SPRING BOOT

```
package com.alumnos.am.bean;
```

```
public class AmBean {
```











```
    private String nombre;
```

```
    private String apellido;
```

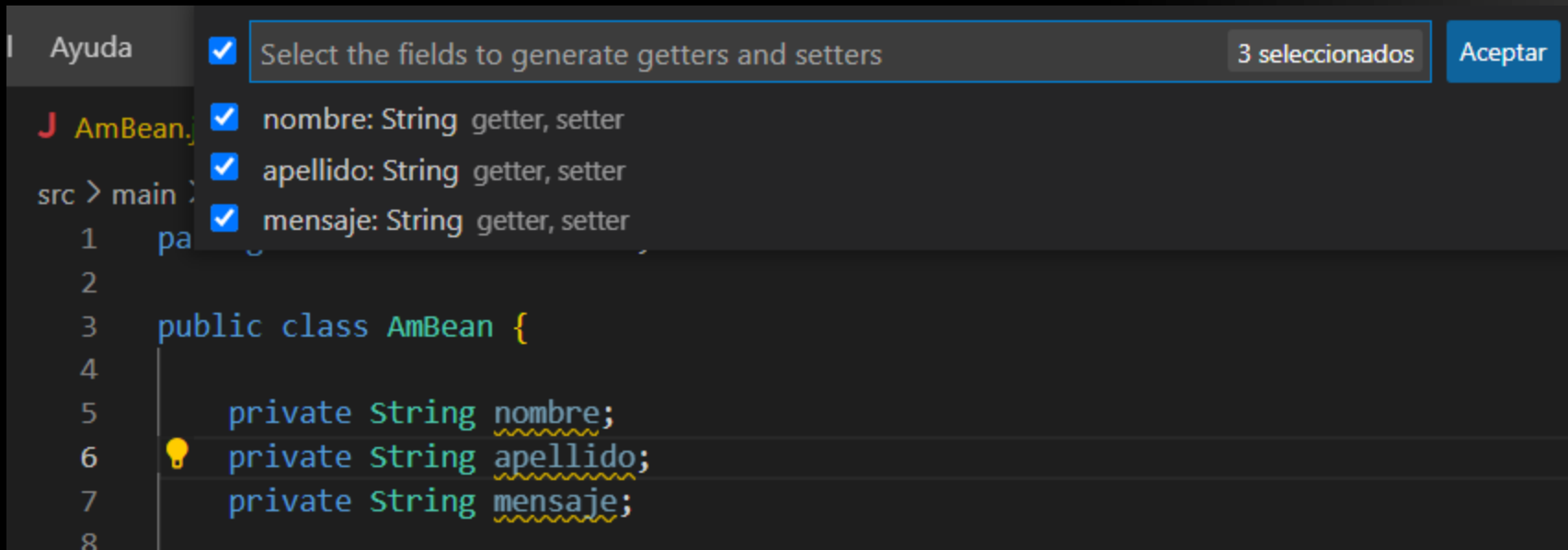
```
    private String mensaje;
```

```
}
```

Acción de origen...

-  Generate Tests...
-  Organize imports Mayús + Alt + O
-  **Generate Getters and Setters...**
-  Generate Getters...
-  Generate Setters...
-  Generate Constructors...
-  Generate hashCode() and equals()...
-  Generate toString()...
-  Override/Implement Methods...
-  Generate Delegate Methods...

APLICACIÓN WEB SPRING BOOT



APLICACIÓN WEB SPRING BOOT

- Quedando algo como lo siguiente:

```
public class AmBean {  
  
    private String nombre;  
    private String apellido;  
    private String mensaje;  
  
    public String getNombre() {  
        return nombre;  
    }  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
    public String getApellido() {  
        return apellido;  
    }  
    public void setApellido(String apellido) {  
        this.apellido = apellido;  
    }  
    public String getMensaje() {  
        return mensaje;  
    }  
    public void setMensaje(String mensaje) {  
        this.mensaje = mensaje;  
    }  
}
```



APLICACIÓN WEB SPRING BOOT

- Una vez creado el bean se hacen los ajustes necesarios para usarlo.
- En el servicio, el parámetro de entrada se cambia para que sea este bean.
- En el método del servicio, en lugar de usar los parámetros de entrada se obtienen con los get los valores guardados en este objeto.

APLICACIÓN WEB SPRING BOOT

```
AmBean.java  J AmServicioImpl.java X
c > main > java > com > alumnos > am > servicio > Impl > J AmServicioImpl.java > ...

3  import org.springframework.stereotype.Service;
4
5  import com.alumnos.am.bean.AmBean;
6  import com.alumnos.am.servicio.AmServicio;
7
8  @Service
9  public class AmServicioImpl implements AmServicio {
10
11      public String crearMensaje(AmBean usuario) {
12          StringBuilder mensajeAEnviar = new StringBuilder(str: "¡Hola ");
13          mensajeAEnviar.append(usuario.getNombre());
14          mensajeAEnviar.append(" ");
15          mensajeAEnviar.append(usuario.getApellido());
16          mensajeAEnviar.append(str: "!");
17          return mensajeAEnviar.toString();
18      }
19
20  }
```

```
J AmServicio.java X
src > main > java > com > alumnos > am > servicio > J AmSe

1  package com.alumnos.am.servicio;
2
3  import com.alumnos.am.bean.AmBean;
4
5  public interface AmServicio {
6      String crearMensaje(AmBean usuario);
7  }
8
```

APLICACIÓN WEB SPRING BOOT

- Con el cambio en el servicio hay que cambiar el parámetro que se manda desde el Controller.
- Para que no haya que informar el objeto desde el Controller se hace que desde la JSP el usuario al rellenar el formulario informe al objeto.

APLICACIÓN WEB SPRING BOOT

- En la JSP se van a usar etiquetas especiales, para lo cual hay que importar un taglib:

`<%@` taglib prefix="form"
uri="http://www.springframework.org/tags/form" %>

```
J AmBean.java  J FormularioController.java  <> formulario.jsp X
src > main > webapp > WEB-INF > jsp > <> formulario.jsp > ? > ? > ? html > ? head > ? title
1  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2  <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
3  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4
5  <html>
6  <head>
```

APLICACIÓN WEB SPRING BOOT

- Para que se pueda usar el objeto dentro del formulario hay que añadirle la etiqueta “modelAttribute” con el nombre del objeto que se pasa desde el Controller:

```
<div>  
  <form action="saludar" method="post" modelAttribute="usuario">
```

- Y en el Controller hay que pasar ese objeto como atributo del modelo:

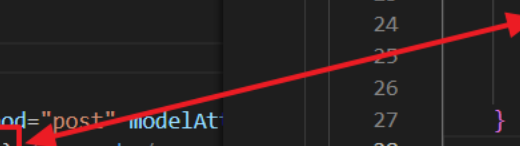
```
@RequestMapping(value="/abrirFormulario", method = RequestMethod.GET)  
public String irAFormulario(Model model) {  
    AmBean usuario = new AmBean();  
    model.addAttribute(attributeName: "usuario", usuario);  
    return "formulario";  
}
```

APLICACIÓN WEB SPRING BOOT

- Si en el Controller se informa alguna variable antes de pasar el objeto como atributo a la JSP esta se vería en la pantalla:

```
<title>spring boot web Application</title>
</head>
<body>
  <div>
    <h1>FORMULARIO</h1>
  </div>
  <div>
    <form action="/saludar" method="post" modelAttribute="usuario">
      <span>${usuario.mensaje}</span><br/>
    </form>
  </div>
</body>
</html>
```

```
21 @RequestMapping(value="/abrirFormulario", method = RequestMethod.GET)
22 public String irAFormulario(Model model) {
23     AmBean usuario = new AmBean();
24     usuario.setMensaje(mensaje: "Inserte su nombre y apellido");
25     model.addAttribute(attributeName: "usuario", usuario);
26     return "formulario";
27 }
28
```



APLICACIÓN WEB SPRING BOOT

- Para que los valores insertados en pantalla lleguen de la JSP al Controller al hacer submit se debe usar la siguiente etiqueta:

```
<form:input path="[atributo].[variable]"/>
```


APLICACIÓN WEB SPRING BOOT

```
<form action="saludar" method="post" modelAttribute="usuario">  
  <span>${usuario.mensaje}</span><br/>  
  <span>Nombre:</span>  
  <form:input path="usuario.nombre"/><br/>  
  <form:label path="usuario.apellido">Apellido:</form:label>  
  <form:input path="usuario.apellido"/><br/>  
  <input type="submit" value="Saludo"/>  
</form>
```

APLICACIÓN WEB SPRING BOOT

- En el método del Controller hay que cambiar dos cosas:
- Para indicar el *action* del formulario con los parámetros que llegan por POST, de esta manera ya no hace falta indicar que el método es POST:
- @RequestMapping → @PostMapping

APLICACIÓN WEB SPRING BOOT

- La información que le llega ahora al Controller ya no es como parámetro, ya que el objeto se ha insertado en la JSP como atributo del modelo, por lo que se recibe como tal:

`@RequestParam` → `@ModelAttribute`

APLICACIÓN WEB SPRING BOOT

- Y el método del Controller cuando se haga submit quedaría de la siguiente manera:

```
@PostMapping("/saludar")  
public String metodoSaludar(ModelMap model,  
    @ModelAttribute("usuario") AmBean usuario) {  
    String mensajeCreado = servicio.crearMensaje(usuario);  
    model.put("mensaje", mensajeCreado);  
    return "formulario";  
}
```

APLICACIÓN WEB SPRING BOOT

```
formulario.jsp X
src > main > webapp > WEB-INF > jsp > formulario.jsp > ? > ? > ? > html > body > div > form
1 <%@ page contentType="text/html;charset=UTF-8" language="java" %>
2 <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w
4
5 <html>
6 <head>
7 <title>Spring Boot Web Application</title>
8 </head>
9 <body>
10 <div>
11 <h1>FORMULARIO</h1>
12 </div>
13 <div>
14 <form action="saludar" method="post" modelAttribute="usuario">
15 <span>${usuario.mensaje}</span><br/>
16 <span>Nombre:</span>
17 <form:input path="usuario.nombre"/><br/>
18 <form:label path="usuario.apellido">Apellido:</form:label>
19 <form:input path="usuario.apellido"/><br/>
20 <input type="submit" value="Saludo"/>
21 </form>
22 <h1><font color="red">${mensaje}</font></h1>
23 </div>

FormularioController.java
src > main > java > com > alumnos > am > web > FormularioController.java > FormularioController
14
15 @Controller
16 public class FormularioController {
17
18     @Autowired
19     private AmServicioImpl servicio;
20
21     @RequestMapping(value="/abrirFormulario", method = RequestMethod.GET)
22     public String irAFormulario(Model model) {
23         AmBean usuario = new AmBean();
24         usuario.setMensaje(mensaje: "Inserte su nombre y apellidos, por favor")
25         model.addAttribute(attributeName: "usuario", usuario);
26         return "formulario";
27     }
28
29     @PostMapping("/saludar")
30     public String metodoSaludar(ModelMap model,
31                                @ModelAttribute("usuario") AmBean usuario) {
32         String mensajeCreado = servicio.crearMensaje(usuario);
33         model.put(key: "mensaje", mensajeCreado);
34         return "formulario";
35     }
36 }
```

APLICACIÓN WEB SPRING BOOT

- Para completar esta aplicación web y que sea ejemplo de cualquier aplicación web normal falta poder comunicarse con la capa de persistencia, es decir, con una base de datos.
- En este ejemplo la base de datos será una *mySQL* y el método de acceso mediante *myBatis*. Otra herramienta muy usada y similar a *myBatis* es *Hibernate*.

APLICACIÓN WEB SPRING BOOT

- Para poder usar myBatis se necesita importar las librerías necesarias, en este caso mediante maven, en el *pom.xml*:

```
<dependency>  
  <groupId>org.mybatis.spring.boot</groupId>  
  <artifactId>mybatis-spring-boot-starter</artifactId>  
  <version>2.2.2</version>  
</dependency>
```

APLICACIÓN WEB SPRING BOOT

- Para poder conectarse con la base de datos se necesita añadir el conector o driver, en este caso mediante maven, en el *pom.xml*:

```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <scope>runtime</scope>  
</dependency>
```


APLICACIÓN WEB SPRING BOOT

- En el fichero *application.properties*, localizado en el directorio *resources*, se debe añadir la información de la base de datos a la que se conecta la aplicación (*testdb* sería el nombre de la base de datos):

Configuración conexión a la base de datos

```
spring.datasource.url=jdbc:mysql://localhost:3306/alonsomadrigal?useUnicode=true&characterEncoding=utf-8
```

```
spring.datasource.username=root
```

```
#spring.datasource.password=root
```

APLICACIÓN WEB SPRING BOOT

- También se añade en este *application.properties* el driver a usar para conectarse a la base de datos.
- En este caso es con una base de datos **mySQL**:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

APLICACIÓN WEB SPRING BOOT

- También se añade en este *application.properties* la dirección de las clases Bean que se van a usar en myBatis.
- Esto sirve para poder escanear desde el xml dónde están los ficheros Bean:

#mybatis entity scan packages

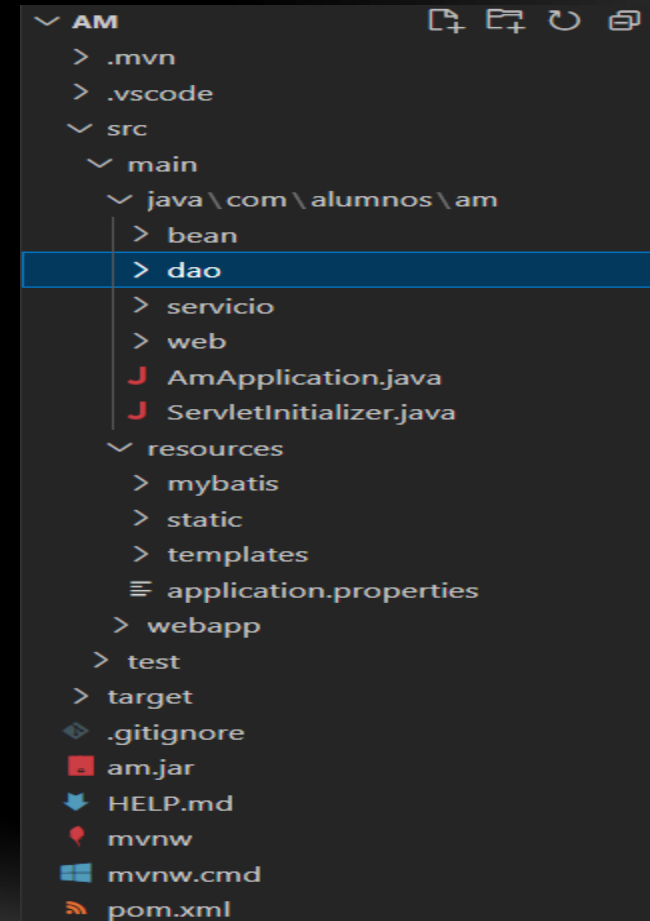
mybatis.type-aliases-package=com.alumnos.am.bean

APLICACIÓN WEB SPRING BOOT

- Al igual que se escanea desde los xml dónde están los Bean, hay que escanear dónde están los ficheros que se van a conectar con la base de datos.
- Estos ficheros que llamaremos Mapper los ubicaremos en una carpeta llamada “dao” (Data Access Object), que estará en el mismo nivel que las carpetas web o servicio.

APLICACIÓN WEB SPRING BOOT

- La estructura de carpetas quedaría de la siguiente manera:



APLICACIÓN WEB SPRING BOOT

- Ahora, para que pueda haber comunicación entre los xml que serán los enlaces directos con la base de datos y los Mapper de java que estarán en la carpeta dao hay que añadir el escaner de los Mapper en la clase *Application.java*:

```
@SpringBootApplication
```

```
@MapperScan("com.alumnos.am.dao")
```

```
public class AmApplication extends SpringBootServletInitializer {
```

APLICACIÓN WEB SPRING BOOT

```
application.properties  AmApplication.java X

src > main > java > com > alumnos > am > AmApplication.java > AmApplication

3  import org.mybatis.spring.annotation.MapperScan;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6  import org.springframework.boot.builder.SpringApplicationBuilder;
7  import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;
8
9  @SpringBootApplication
10 @MapperScan("com.alumnos.am.dao")
11 public class AmApplication extends SpringBootServletInitializer {
12
13     @Override
14     protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
15         return builder.sources(...sources: AmApplication.class);
16     }
17
18     Run | Debug
19     public static void main(String[] args) {
20         SpringApplication.run(primarySource: AmApplication.class, args);
21     }
22
23 }
```

APLICACIÓN WEB SPRING BOOT

- El mapper, por último, contendrá las operaciones de acceso a datos que serán invocadas por el repositorio.
- En esta capa es en donde se definen las consultas a base de datos, a través de interfaces denominadas mappers.

APLICACIÓN WEB SPRING BOOT

- Una vez configurado dónde van a estar los ficheros Mapper (dao) se crea el que se va a usar dentro de esa carpeta dao:

@Mapper

```
public interface UsuariosMapper {
```

```
    UsuarioBean obtenerPassword(UsuarioBean user);
```

```
}
```

APLICACIÓN WEB SPRING BOOT

- En el implementador del servicio se crea la variable del Mapper con la anotación Autowired:

@Autowired

```
private UsuariosMapper mapper;
```

- Con esto se puede llamar a los métodos del Mapper desde el servicio.

APLICACIÓN WEB SPRING BOOT

```
J AmServicioImpl.java X
src > main > java > com > alumnos > am > servicio > Impl > J AmServicioImpl.java > AmServicioImpl
import com.alumnos.am.servicio.AmServicio;

10
11 @Service
12 public class AmServicioImpl implements AmServicio {
13
14     @Autowired
15     private UsuariosMapper mapper;
16
17     public String crearMensaje(AmBean usuario) {
18
19         UsuarioBean user = new UsuarioBean();
20         user.setNombre(usuario.getNombre());
21         user.setApellidos(usuario.getApellido());
22         user = mapper.obtenerPassword(user);
23
24         StringBuilder mensajeAEnviar = new StringBuilder(str: "¡Hola ");
25         mensajeAEnviar.append(usuario.getNombre());
26         mensajeAEnviar.append(str: " ");
27         mensajeAEnviar.append(usuario.getApellido());
28
29         if (user != null) {
30             mensajeAEnviar.append(str: "! Su contraseña es: ");
31             mensajeAEnviar.append(user.getPassword());
32         } else {
33             mensajeAEnviar.append(str: "! Usted no está en el sistema");
34         }
35         return mensajeAEnviar.toString();
36     }
37 }
38 }
```

APLICACIÓN WEB SPRING BOOT

- El método del Mapper, lo que hace es ejecutar una sentencia SQL.
- Esta sentencia a ejecutar está definida en un xml.
- La ubicación de este xml se define en el *application.properties*.

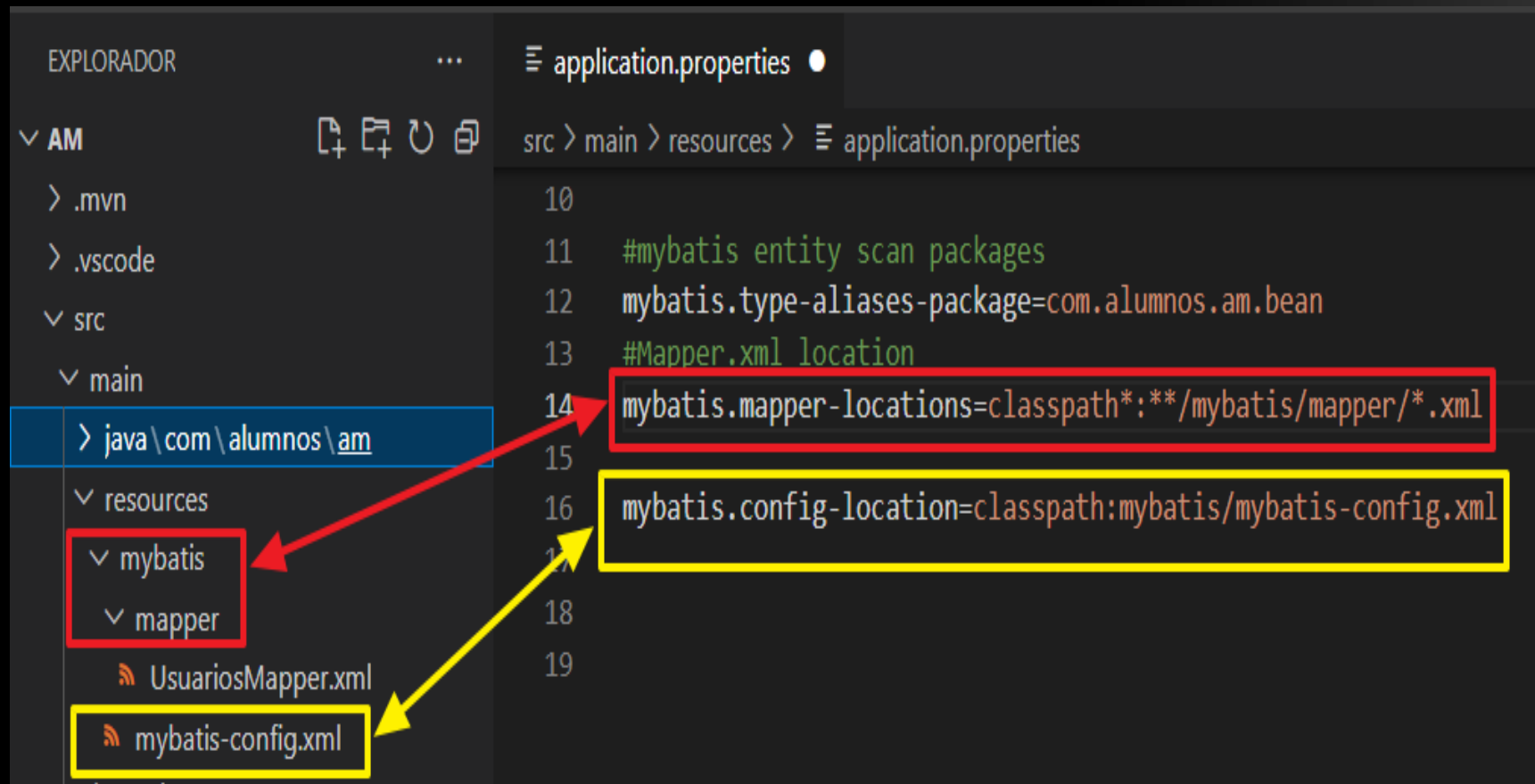
APLICACIÓN WEB SPRING BOOT

#Mapper.xml location

mybatis.mapper-locations=classpath*:**/mybatis/mapper/*.xml

mybatis.config-location=classpath:mybatis/mybatis-config.xml

APLICACIÓN WEB SPRING BOOT



EXPLORADOR

... application.properties

src > main > resources > application.properties

```
10
11 #mybatis entity scan packages
12 mybatis.type-aliases-package=com.alumnos.am.bean
13 #Mapper.xml location
14 mybatis.mapper-locations=classpath:*/mybatis/mapper/*.xml
15
16 mybatis.config-location=classpath:mybatis/mybatis-config.xml
17
18
19
```

AM

- > .mvn
- > .vscode
- > src
 - > main
 - > java \com \alumnos \am
 - > resources
 - mybatis
 - mapper
 - UsuariosMapper.xml
 - mybatis-config.xml

APLICACIÓN WEB SPRING BOOT

- En el fichero xml de configuración de myBatis se definen los objetos que se ven a usar.
- En esta definición se especifican los tipos de datos y los bean que se usarán en los xml de sentencias de SQL.

APLICACIÓN WEB SPRING BOOT

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias alias="Integer" type="java.lang.Integer" />
    <typeAlias alias="Long" type="java.lang.Long" />
    <typeAlias alias="HashMap" type="java.util.HashMap" />
    <typeAlias alias="LinkedHashMap" type="java.util.LinkedHashMap" />
    <typeAlias alias="ArrayList" type="java.util.ArrayList" />
    <typeAlias alias="LinkedList" type="java.util.LinkedList" />

    <typeAlias type="com.alumnos.am.bean.UsuarioBean" alias="UsuarioBean" />
  </typeAliases>
</configuration>
```


APLICACIÓN WEB SPRING BOOT

- El xml donde se programan las sentencias SQL hay que definirlo como un mapper de myBatis:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper  
3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
```

APLICACIÓN WEB SPRING BOOT

- En el xml lo primero es definir qué Mapper es el que está usando dicho xml.
- Esta definición viene dada por el namespace:

```
<mapper namespace="com.alumnos.am.dao.UsuariosMapper">
```

APLICACIÓN WEB SPRING BOOT

- En el xml, la parte de la sentencia SQL tiene dos partes:
 - La sentencia SQL:
 - Hay que especificar el id, que es el nombre del método del Mapper (dao).
 - El mapeo de resultados:
 - Se mapean las columnas SQL con las variables java del Bean donde se guardan.

APLICACIÓN WEB SPRING BOOT

```
<resultMap id="passwordResult" type="UsuarioBean" >
  <id column="idUserio" property="idUserio" jdbcType="INTEGER" />
  <result column="nombre" property="nombre" jdbcType="VARCHAR" />
  <result column="apellidos" property="apellidos" jdbcType="VARCHAR" />
  <result column="password" property="password" jdbcType="VARCHAR" />
</resultMap>

<select id="obtenerPassword" resultMap="passwordResult">
  SELECT idUsuario, nombre, apellidos, password
  FROM usuarios
  WHERE nombre = #{nombre}
  AND apellidos = #{apellidos}
</select>
```

¡GRACIAS!