

SERVLETS Y WEB APLICACIONES

DAW2 – DAW

Alejandro Sainz Sainz

6-10-2025

1 TOMCATS Y SERVLETS.....	3
1.1 AHORA SI, COMENZANDO.....	3
1.2 PREPARANDO EL ARCHIVO LOCAL.....	10
2 PROBANDO SOBRE NUESTRO SERVIDOR.....	13
3 CREANDO UN SERVLET PROPIO.....	16
4 CONCLUSIONES.....	19
4.1 COMPLEJIDAD DE LA ACTIVIDAD.....	19
4.2 ¿CONOCES OTRO SERVIDOR DE APLICACIONES?.....	20

Índice de figuras

Figura 1: ARRANCAMOS TOMCAT.....	4
Figura 2: CREAMOS EL WORKSPACE DE ECLIPSE.....	5
Figura 3: CREACIÓN DEL SERVER DE TOMCAT EN ECLIPSE.....	5
Figura 4: SEGUNDA PARTE DE LA CREACIÓN DEL SERVIDOR.....	6
Figura 5: SERVIDOR CREADO Y A LA ESPERA.....	6
Figura 6: PREPARAMOS EL NUEVO PROYECTO.....	7
Figura 7: CREACIÓN DE UN SERVLET.....	7
Figura 8: MODIFICANDO EL SERVLET.....	8
Figura 9: ARBOL DE DIRECTORIOS.....	8
Figura 10: ARCHIVO JSP.....	9
Figura 11: PRUEBA DE EJECUCIÓN.....	10
Figura 12: TODO CORRECTO.....	10
Figura 13: EXPORTANDO NUESTRO PROYECTO.....	12
Figura 14: EXPORTANDO.....	12
Figura 15: COMPROBAMOS EN EL EXPLORADOR.....	13
Figura 16: EN PRINCIPIO, TODO CORRECTO POR AHORA.....	14
Figura 17: SERVIDOR TOMCAT.....	15

1 TOMCAT Y SERVLETS

Vamos a comenzar a documentar esta actividad acto seguido de iniciar el servicio de Tomcat. Hago esto ya que es el mismo proceso que en la actividad anterior salvo que usaremos Tomcat como ya he explicado en vez de PHP y Apache.

Tampoco voy a entrar en los detalles de descarga de las aplicaciones necesarias, tanto el JRE y Eclipse, pues esos ya son procesos de sobra conocidos por todos. No suponen ninguna novedad en comparación con lo hecho en otras actividades.

Evidentemente, tras la descarga proseguimos con la instalación.

1.1 AHORA SI, COMENZANDO

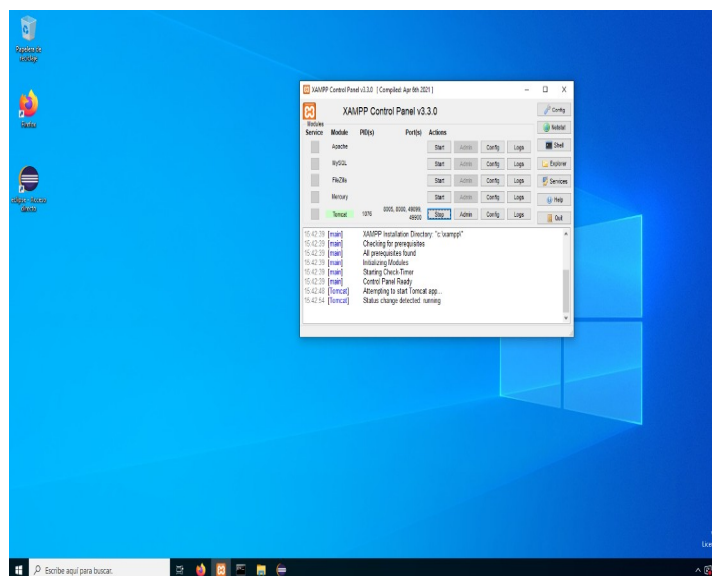


FIGURA 1: ARRANCAMOS TOMCAT

Bien, pues como vemos en la imagen lo primero que hacemos es arrancar el servidor Tomcat, para luego poder trabajar sobre él.

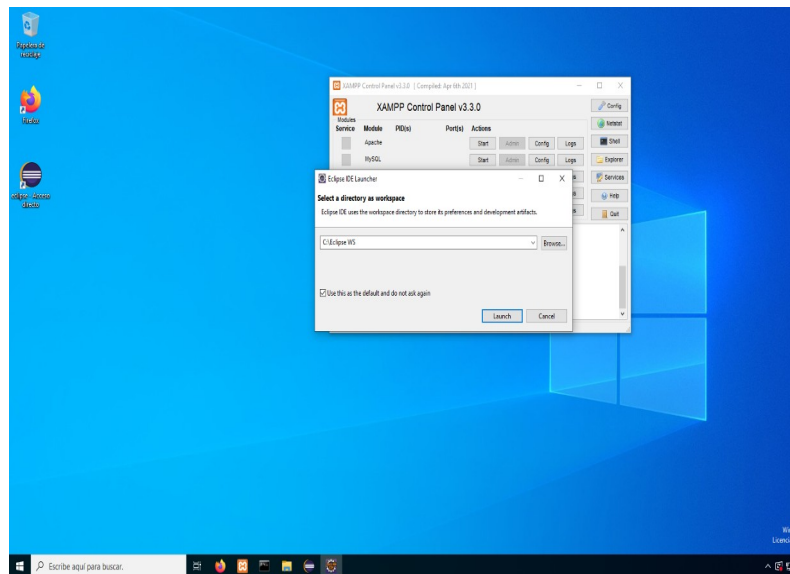


FIGURA 2: CREAMOS EL WORKSPACE DE ECLIPSE

Cuando arrancamos Eclipse lo primero que se nos solicita es crear o elegir una carpeta para el workspace de Eclipse. Elijo una dirección, a ser posible una fácil de encontrar por si luego no es necesario pasar alguna ruta.

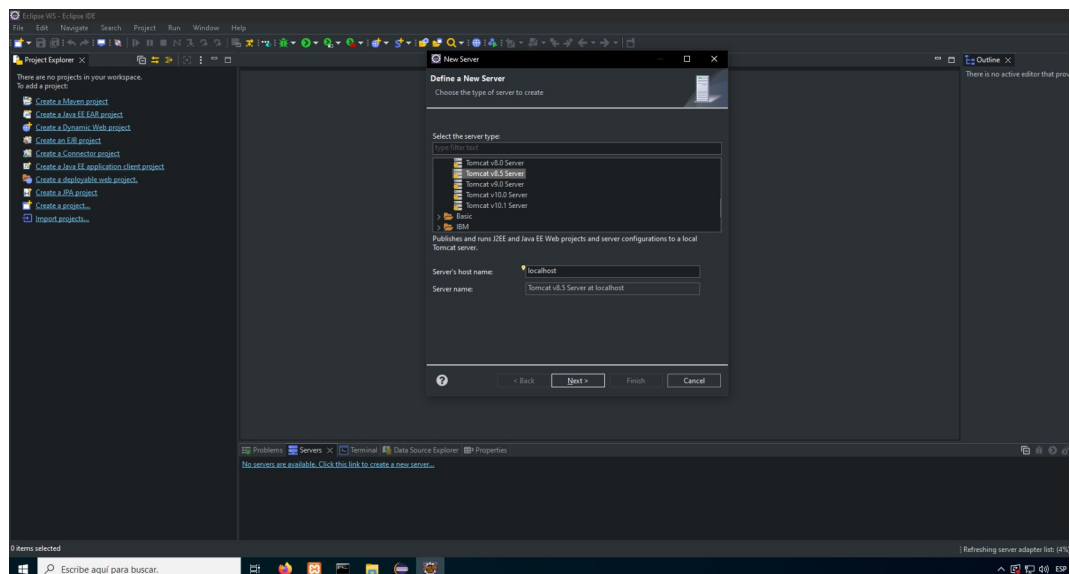


FIGURA 3: CREACIÓN DEL SERVER DE TOMCAT EN ECLIPSE

Seguimos las indicaciones del manual de la actividad al pie de la letra, más que nada para evitar futuros problemas y algún que otro dolor de cabeza luego.

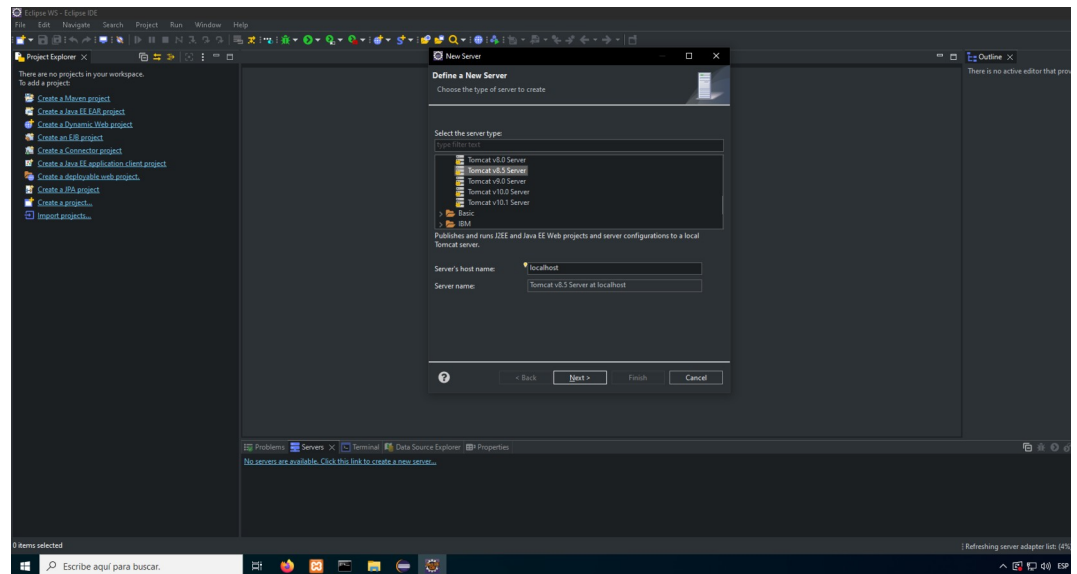


FIGURA 4: SEGUNDA PARTE DE LA CREACIÓN DEL SERVIDOR

Como en el paso anterior no toco nada raro ni dejo lugar para la improvisación. Seguimos todo al pie de la letra.

Finalmente, en la consola del eclipse podemos ver que, aunque no se esté ejecutando el servidor, si está creado.

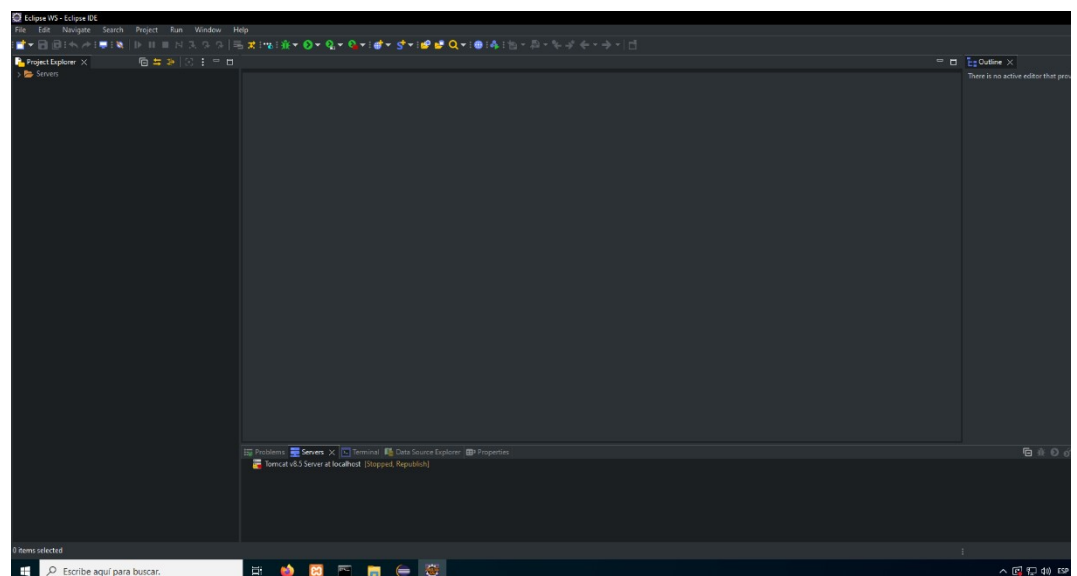


FIGURA 5: SERVIDOR CREADO Y A LA ESPERA

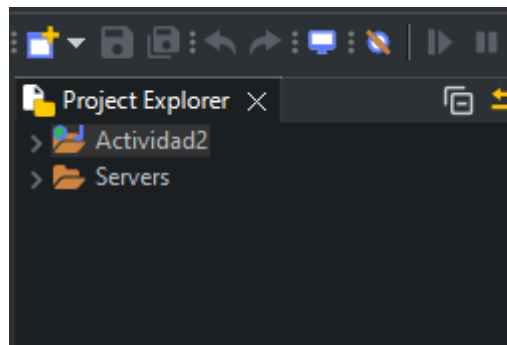


FIGURA 6: PREPARAMOS EL NUEVO PROYECTO

Como nos indica la actividad, creamos un nuevo proyecto web dinámico para empezar a trabajar con el.

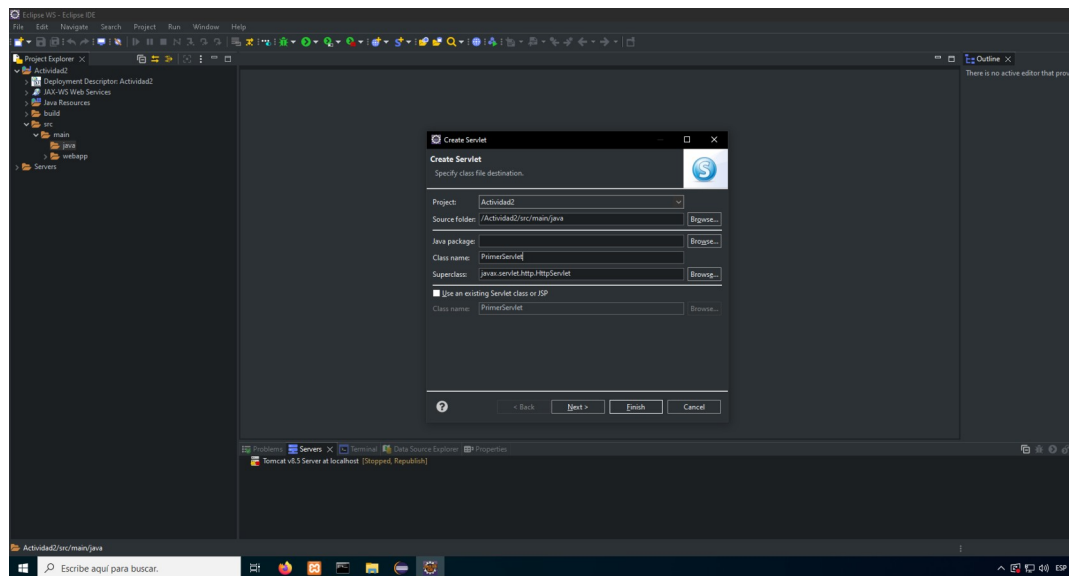


FIGURA 7: CREACIÓN DE UN SERVLET

Como se nos pide en la actividad creamos el primer servlet para poder realizar las pruebas y los ejercicios que se nos piden.

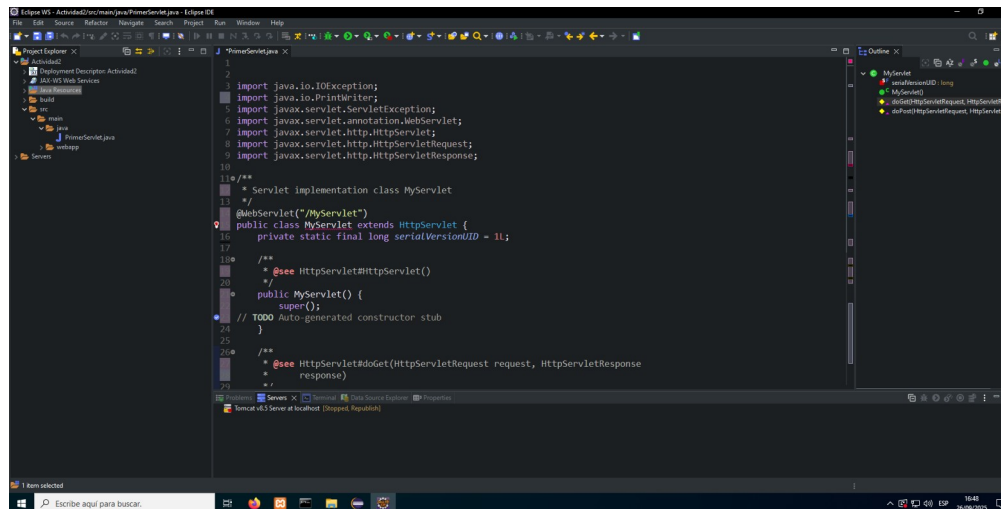


FIGURA 8: MODIFICANDO EL SERVLET

Para seguir con la parte de muestra de la actividad, lo que hago es copiar el texto que se nos indica y modificar el servlet que se crea por defecto. Una vez hecho seguiré con el siguiente paso de la actividad.

Antes de continuar, voy a intentar asegurarme de que las imágenes se ven de forma correcta, porque hasta ahora salvo un par de ellas, las veo todas bastante borrosas. Bien es cierto que es un calco de aquello que se me va indicando en la guía paso a paso pero, a partir de aquí muchas de las cosas van a ser las modificaciones propias que haga yo sobre la aplicación. Así que voy a intentar que se vean todas más nítidas para que se pueda entender todo de forma más visual.

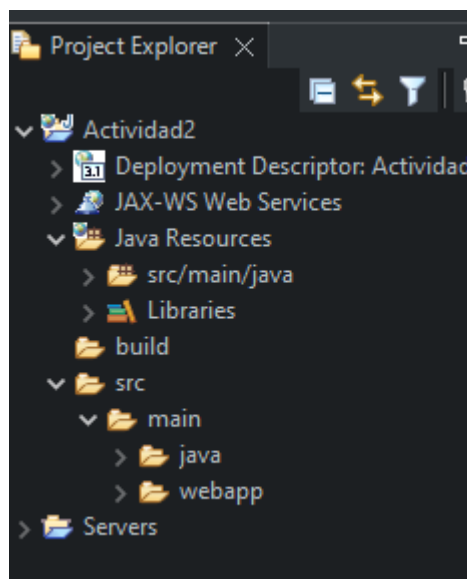
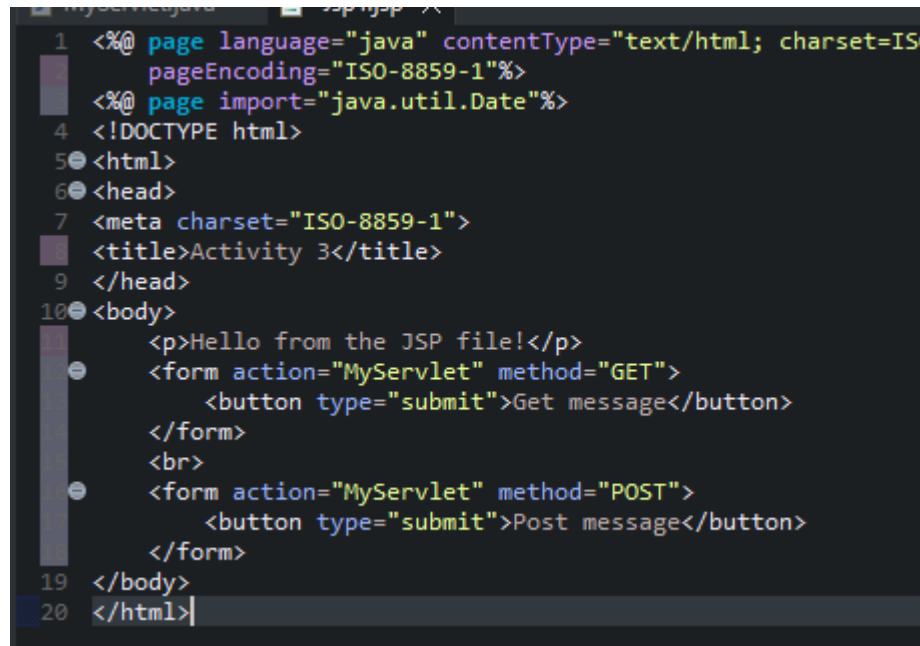


FIGURA 9: ARBOL DE DIRECTORIOS

Aquí podemos ver como tenemos estructurados los directorios de nuestro proyecto.

Como nos indica el enunciado del ejercicio debemos de crear a continuación un archivo JSP dentro de nuestra carpeta webapp.

Creamos el archivo y copiamos el fragmento de código que se nos proporciona para el ejercicio.

A screenshot of a code editor showing a JSP file. The code is as follows:

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"%>
  pageEncoding="ISO-8859-1"%>
2 <%@ page import="java.util.Date"%>
3 <!--
4 <!DOCTYPE html>
5 <html>
6 <head>
7 <meta charset="ISO-8859-1">
8 <title>Activity 3</title>
9 </head>
10 <body>
11 <p>Hello from the JSP file!</p>
12 <form action="MyServlet" method="GET">
13   <button type="submit">Get message</button>
14 </form>
15 <br>
16 <form action="MyServlet" method="POST">
17   <button type="submit">Post message</button>
18 </form>
19 </body>
20 </html>
```

FIGURA 10: ARCHIVO JSP

Como vemos en la figura superior ya tenemos preparado nuestro archivo JSP.

Antes de seguir con lo que propone la actividad, en clase se comento que se podía tratar de ejecutar esta aplicación como si de una aplicación por consola se tratase, pero dentro del propio servidor lanzando la misma desde Eclipse. Es lo que voy a probar ahora.

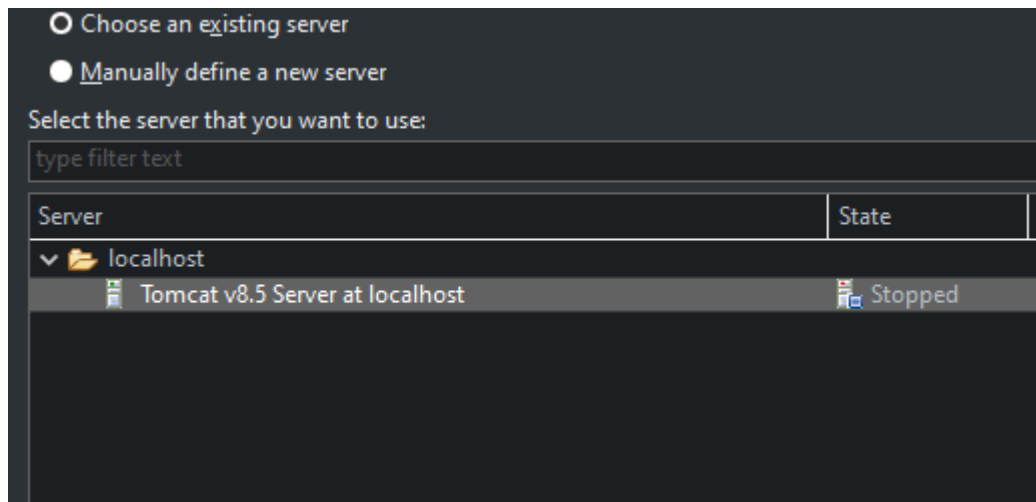


FIGURA 11: PRUEBA DE EJECUCIÓN

Vamos a tratar de ejecutarlo. El server aparece como parado, veremos si el propio eclipse lo arranca o debo de hacerlo yo manualmente antes de intentar ejecutar la aplicación.

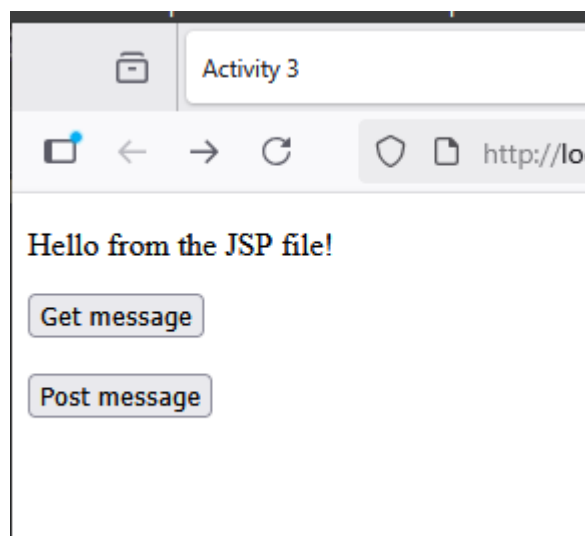


FIGURA 12: TODO CORRECTO

Ha ido bastante bien. Se ha ejecutado sin problemas a la primera, desde eclipse. Ahora de todas formas, probaré lo que indica la actividad.

1.2 PREPARANDO EL ARCHIVO LOCAL

Una vez hemos hecho la prueba exitosa, vamos a exportar nuestro proyecto como un archivo .war, para luego probar su funcionamiento con el proyecto debidamente instalado dentro de nuestro servidor de aplicaciones.

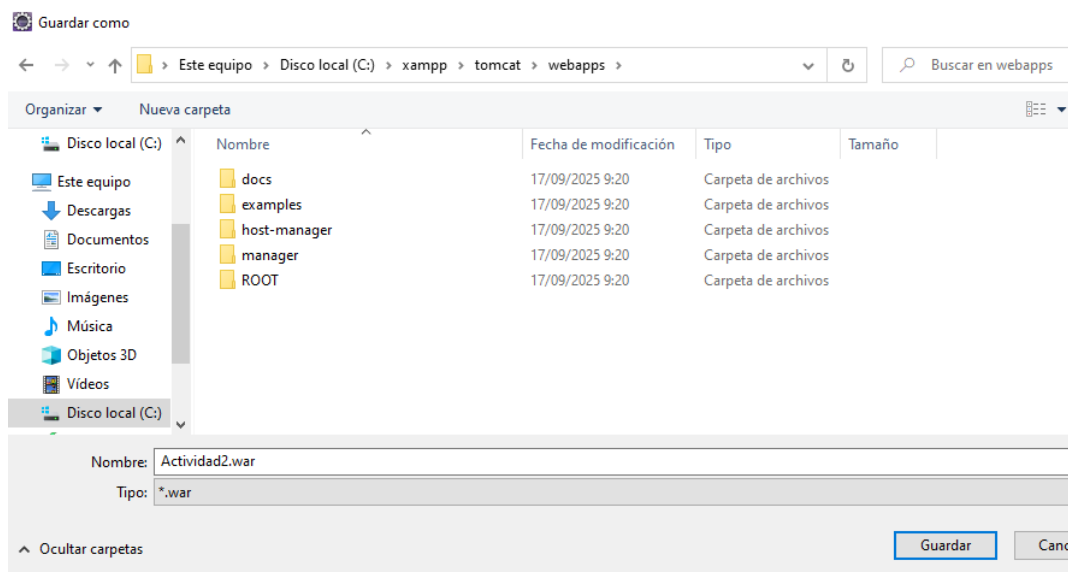


FIGURA 13: EXPORTANDO NUESTRO PROYECTO

Como vemos en la imagen, elegimos la dirección indicada para exportar nuestro archivo .war.

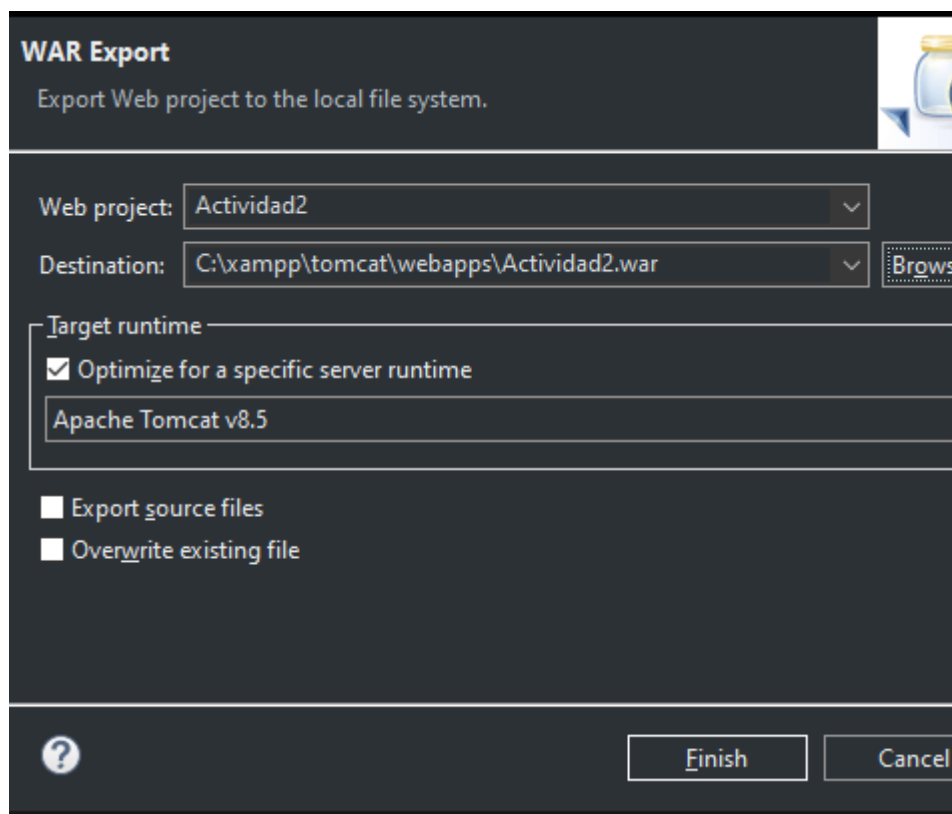


FIGURA 14: EXPORTANDO

Como no se me indica nada al respecto no marco ninguna de las opciones adicionales que se me muestran.

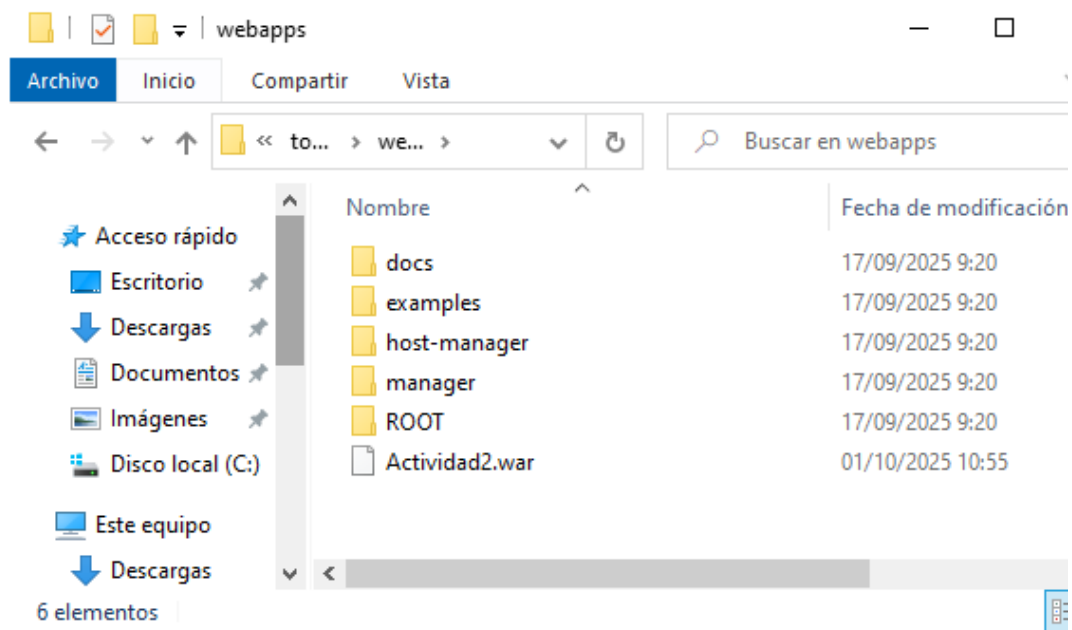


FIGURA 15: COMPROBAMOS EN EL EXPLORADOR

Una vez eclipse termina de exportar, compruebo en el navegador de archivos que efectivamente se ha exportado nuestro proyecto de forma exitosa.

2 PROBANDO SOBRE NUESTRO SERVIDOR

Una vez hemos hecho las pruebas y comprobaciones necesarias vamos a arrancar desde xampp nuestro servidor Tomcat para luego probar desde una navegador si todo funciona correctamente.

Pequeña anotación. Si después de hacer la prueba en eclipse no detenemos el servidor, si intentamos arrancar Tomcat desde el panel de control de Xampp nos da problemas. Lo primero, nada más abrir el panel de control, en su propia consola ya encontramos mensajes que nos indican que Java está actualmente corriendo en uno de dos puertos, ya sea el 8005 o el 8080. Si pulsamos en start junto a Tomcat, el programa se cuelga y el servicio no responde. Traté de acceder a la dirección que se nos indica, marcando también el puerto y nos da el error 404.

Lo que he hecho por ahora, es para el servidor arrancado en eclipse. Vuelvo al panel de control de Xampp e intento arrancar los diferentes servicios y, esta vez si, todo funciona de forma correcta.

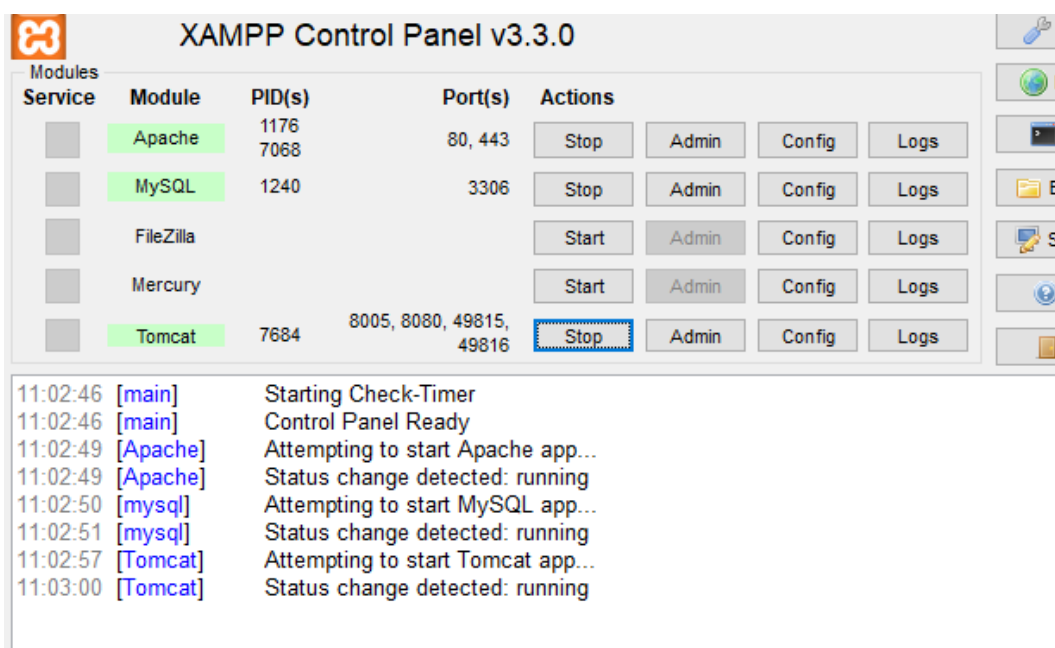


FIGURA 16: EN PRINCIPIO, TODO CORRECTO POR AHORA

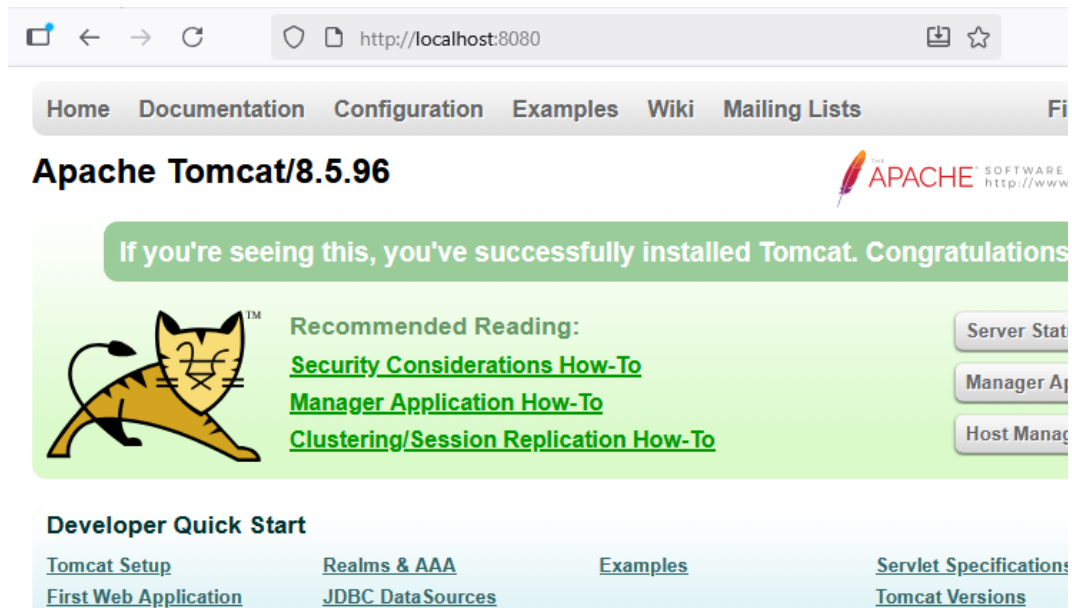


FIGURA 17: SERVIDOR TOMCAT

Si hago una prueba sólo sobre <http://localhost:8080> esta vez sí, aparece la página principal del servidor sin ningún tipo de problema. Vamos a probar ahora añadiendo actividad2.

Bueno, después de mucho jaleo y de los comentado en clase, un solo nombre es lo que tenía la culpa.

```
1 @WebServlet("/Actividad2")
2 public class MyServlet extends HttpServlet {
3     private static final long serialVersionUID = 1L;
4 }
```

Ilustración 1 LO QUE HA COSTADO ENCONTRAR EL FALLO

Pues después de mucho dar vueltas y de probar cosas raras en casa, y después de lo que miramos en clase para poder ver lo que era, comprobando el error que daba cuando no cargaba probé a cambiar el nombre que pone en la primera línea que se ve en la imagen superior. Al poner Actividad2 en vez de MyServlet, entonces si carga de forma correcta.

Ya, ahora sí, si busco la URL <http://localhost:8080/Actividad2> obtengo el siguiente resultado:

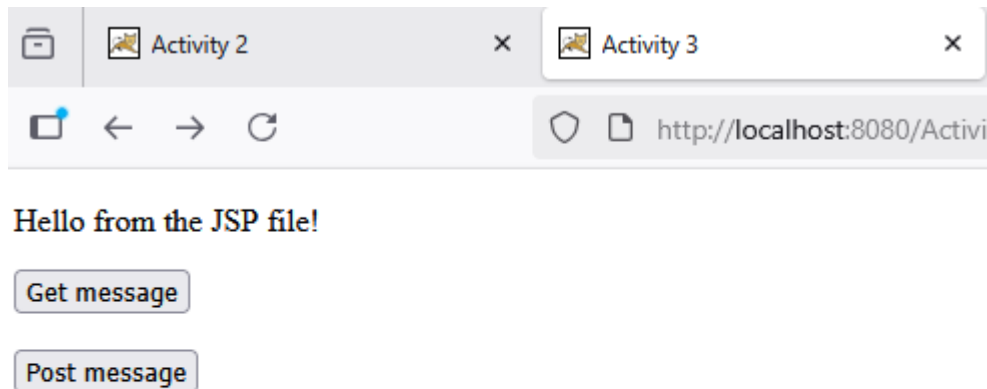


Ilustración 2 POR FIN

Pues ya está. No hace falta que ponga imágenes de las pruebas, la cosa es que funciona.

A partir de aquí hice un par de pruebas para modificar cosas, el problema es que aunque volviese a exportar el proyecto siempre se me cargaba el mismo. Por lo tanto, lo que voy a hacer es crear un nuevo proyecto, desde cero, pero copiando la base de lo que tenemos en los apuntes, e ir editando lo que pueda y haciendo alguna que otra prueba.

3 CREANDO UN SERVLET PROPIO

El primer paso es crear un nuevo Dinamic Web Project. Y lo primero que hago esta vez para asegurarme de que vayamos por el buen camino es cambiar ciertas cosas.

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <html>
3 <head>
4   <title>Verificación de Edad</title>
5 </head>
6 <body>
7   <h2>Introduce tu edad</h2>
8   <form action="EdadServlet" method="post">
9     Edad: <input type="text" name="edad" />
10    <input type="submit" value="Verificar" />
11  </form>
12 </body>
13 </html>
14
```

Ilustración 3: MI ARCHIVO JSP

```
1 import java.io.IOException;
2 import javax.servlet.ServletException;
3 import javax.servlet.annotation.WebServlet;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7
8 @WebServlet("/Prueba")
9 public class Prueba extends HttpServlet {
10    /**
11     *
12     */
13    private static final long serialVersionUID = 1L;
14
```

Ilustración 4: PRIMERA PARTE DEL SERVLET

Ya que hemos subsanado un error no es cosa de volver a repetirlo de forma gratuita, que sienta mal al hígado. Me aseguro que el nombre del servlet coincida bien con el nombre del proyecto que voy a exportar, y sobre todo, en la parte que indica la dirección que se indique bien.

Ahora genero el resto del código.

Ahora voy a copiar lo que crea que necesite del archivo de ejemplo de los apuntes.

```
try {
    int edad = Integer.parseInt(edadParam);
    String mensaje;
    if (edad >= 18) {
        mensaje = "Eres mayor de edad.";
    } else {
        mensaje = "No eres mayor de edad.";
    }
    response.getWriter().println("<html><body>");
    response.getWriter().println("<h2>" + mensaje + "</h2>");
    response.getWriter().println("<a href='formularioEdad.jsp'>Volver</a>");
    response.getWriter().println("</body></html>");
} catch (NumberFormatException e) {
    response.getWriter().println("<html><body>");
    response.getWriter().println("<h2>Por favor, introduce un número vál");
    response.getWriter().println("<a href='formularioEdad.jsp'>Volver</a>");
    response.getWriter().println("</body></html>");
}

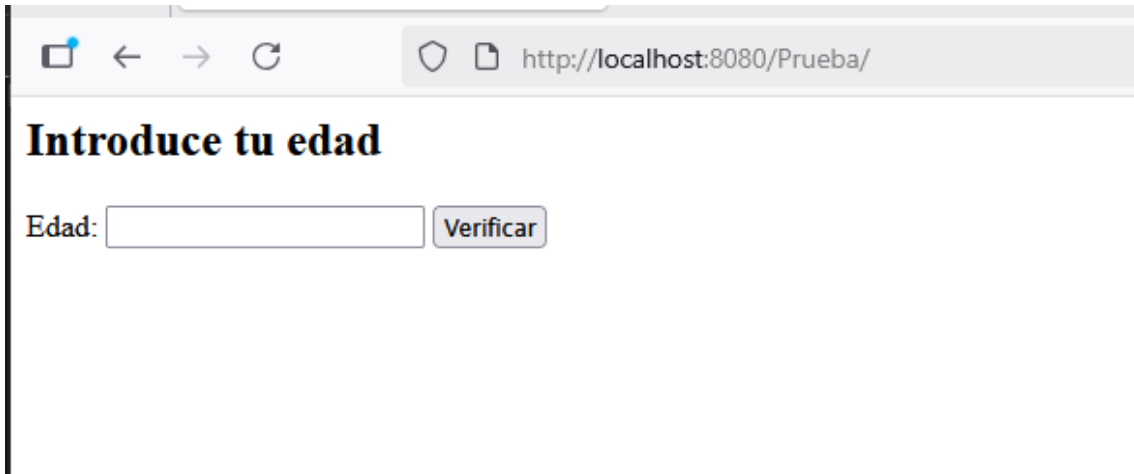
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) {
    response.sendRedirect("formularioEdad.jsp");
}
```

Ilustración 5: COMPROBAR LA MAYORIA DE EDAD

Seguro que lo podía crear mejor para que el input solo aceptase valores numéricos, pero bueno, con alguna línea más lo tengo.

Ahora habrá que hacer las pruebas.



A screenshot of a web browser window. The address bar shows 'http://localhost:8080/Prueba/'. The page title is 'Introduce tu edad'. Below the title, there is a label 'Edad:' followed by a text input field. To the right of the input field is a button labeled 'Verificar'.

Ilustración 6: AHORA HA IDO MÁS FACIL

Ya sabiendo lo que había hecho mal en el ejemplo, ahora ha ido todo como la seda.

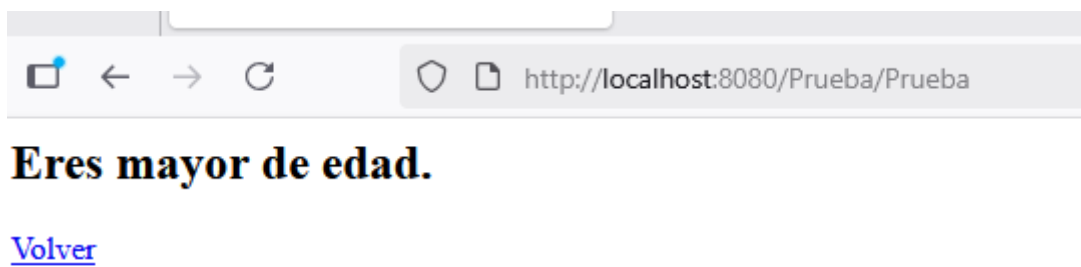


Ilustración 7: COMPROBANDO QUE FUNCIONA CORRECTAMENTE

Pues por ahora todo funciona como debiera.

Después de esto sólo toco un poco el css para que tenga algo de mejor aspecto, pero nada complicado.

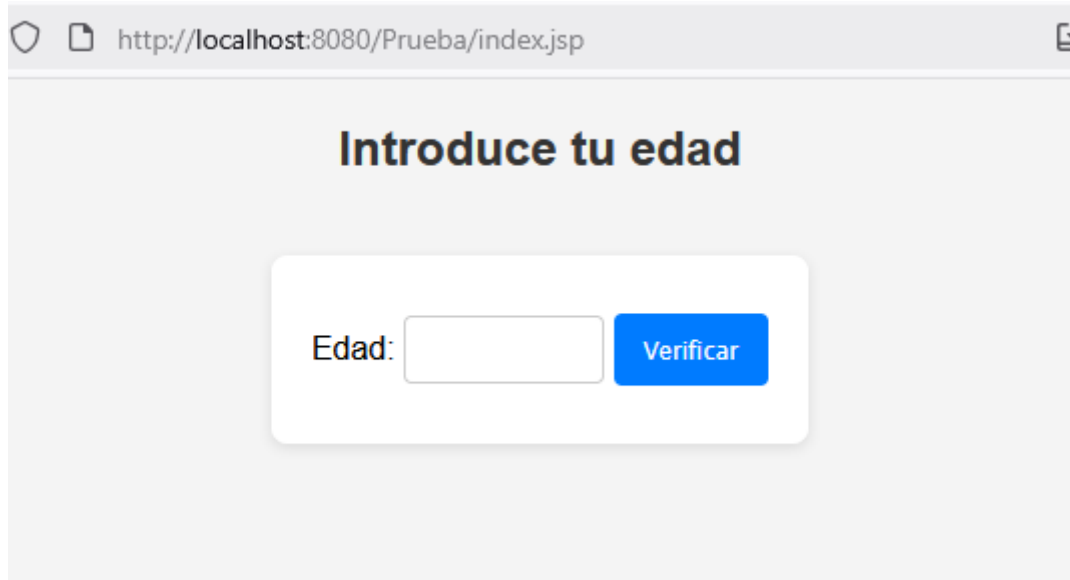


Ilustración 8: MEJORANDO LA IMAGEN

Y con esto lo doy por acabado. No me voy a complicar más, que siempre me pasa.

4 CONCLUSIONES

Vamos con la sección de las preguntas. Esta vez no se me olvida.

4.1 COMPLEJIDAD DE LA ACTIVIDAD

Bueno, pues como ya vimos en clase, todo el lío aquel con los nombres. Es lo que más quebraderos de cabeza me generó. El tener que buscar por internet incluso la forma visual de añadir un archivo .war en tomcat. Aún y con todo no me funcionó muy bien la creación de usuarios y tampoco fui capaz.

En otro orden de cosas, el hacerme a la idea de que para el ejemplo que cree yo casi me daba igual hacerlo todo en el get como en el post. Se me hace un poco raro. Hasta que me ubico y me queda claro que en este caso es sólo una petición en local y que me da un poco igual como lo haga, me cuesta más. Por lo demás, no es nada raro ni fuera del otro mundo, por lo menos a este nivel.

4.2 ¿CONOCES OTRO SERVIDOR DE APLICACIONES?

Pues en este caso si. En la clase de Entorno Servidor se intentó una actividad parecida, pero creando una Enterprise Project o similar usando WildFly.

Hasta donde yo se, la principal diferencia entre uno y otro, es que Tomcat está más pensado para contener sólo una parte del todo, como el servlet, los jsp, etcetera. Sin embargo WildFly está pensado para otro standar, JavaEE, que es como el pack completo.

Para completa ciertas cosas a Tomcat se le pueden añadir otros frameworks, como SpringBoot, mientras que WildFly se encarga de todo.

Tomcat es el taller dentro de la fábrica mientras que WildFly es la fábrica completa.