

DAM  
Desarrollo de Aplicaciones Multiplataforma  
2º Curso

AD  
Acceso a Datos

UD 5  
Programación de componentes  
de acceso a datos  
(Parte 2)

IES BALMIS  
Dpto Informática  
Curso 2022-2023  
Versión 1 (11/2022)

## UD5 – Programación de componentes de acceso a datos

### ÍNDICE

#### 7. Programación de aplicaciones en Servidores Web

**7.1 Lenguaje de script de Servidor**

**7.2 Apache y lenguaje de script PHP**

**7.3 Apache Tomcat**

**7.4 Lenguaje de script JSP**

#### 8. Creación de aplicaciones web

**8.1 NetBeans con Apache Tomcat**

**8.2 NetBeans con Wildfly**

**8.3 Compilación y despliegue de Aplicaciones Web**

## 7. Programación de aplicaciones en Servidores Web

### 7.1 Lenguaje de script de Servidor

Cuando creamos aplicaciones web, necesitamos un servidor que además de ofrecer archivos (HTML, CSS, JS, ...) ejecute código que aporte funcionalidad a la aplicación.

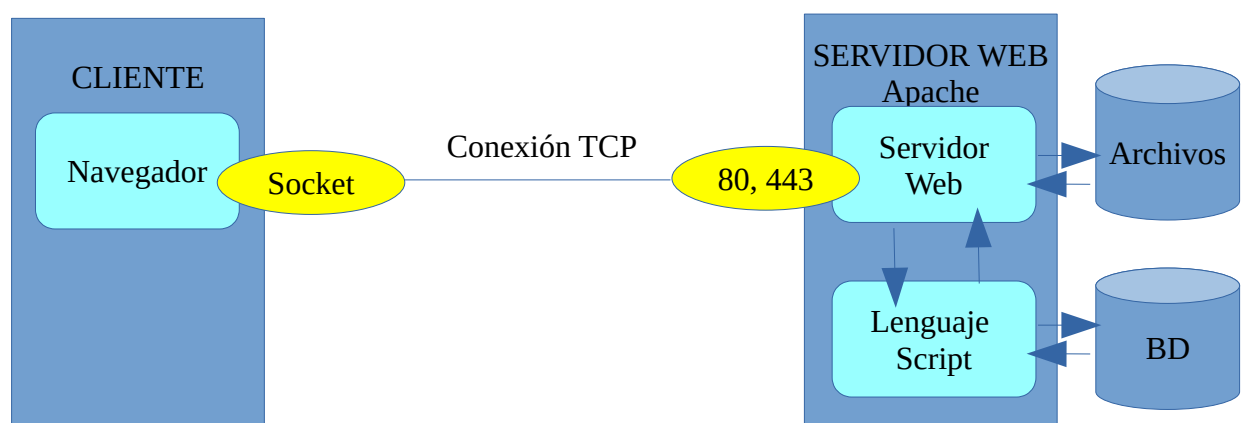
**Lenguaje del lado del servidor** es aquel que se ejecuta en el servidor web, inmediatamente antes de que el sitio web se envíe a través de Internet al usuario.

Los sitios web que se ejecutan en el servidor pueden realizar un amplio abanico de tareas hasta formar el propio sitio web que va a ver el usuario: acceso a base de datos, conexión en red, ...

Los lenguajes de lado servidor más ampliamente utilizados para el desarrollo de páginas dinámicas son el **PHP**, **ASP** y **JSP**, aunque también podemos usar por ejemplo JavaScript con **NodeJS**.

### 7.2 Apache y lenguaje de script PHP

Para poder implantar una solución Web que utilice en el servidor lenguaje PHP, necesitaremos un servidor web como **Apache** que ejecute el código y devuelva el resultado al navegador cliente:



#### Lenguaje de Script PHP

Aunque existen varios, en este caso utilizaremos **Apache +PHP**.

Podemos instalar manualmente Apache y PHP de forma independiente y luego configurar Apache para que utilice PHP. Otra opción para desarrollo es utilizar aplicaciones con **Apache y PHP** preconfigurados como **XAMPP**.

Para la instalación de los servidores web, crearemos una carpeta **C:\SERVIDORES**.

### **XAMPP → Apache + MySQL + PHP**

La primera instalación será XAMPP para disponer de un servidor web.  
Descargaremos el archivo portable .zip de la última versión:

<https://sourceforge.net/projects/xampp/files/XAMPP%20Windows/>

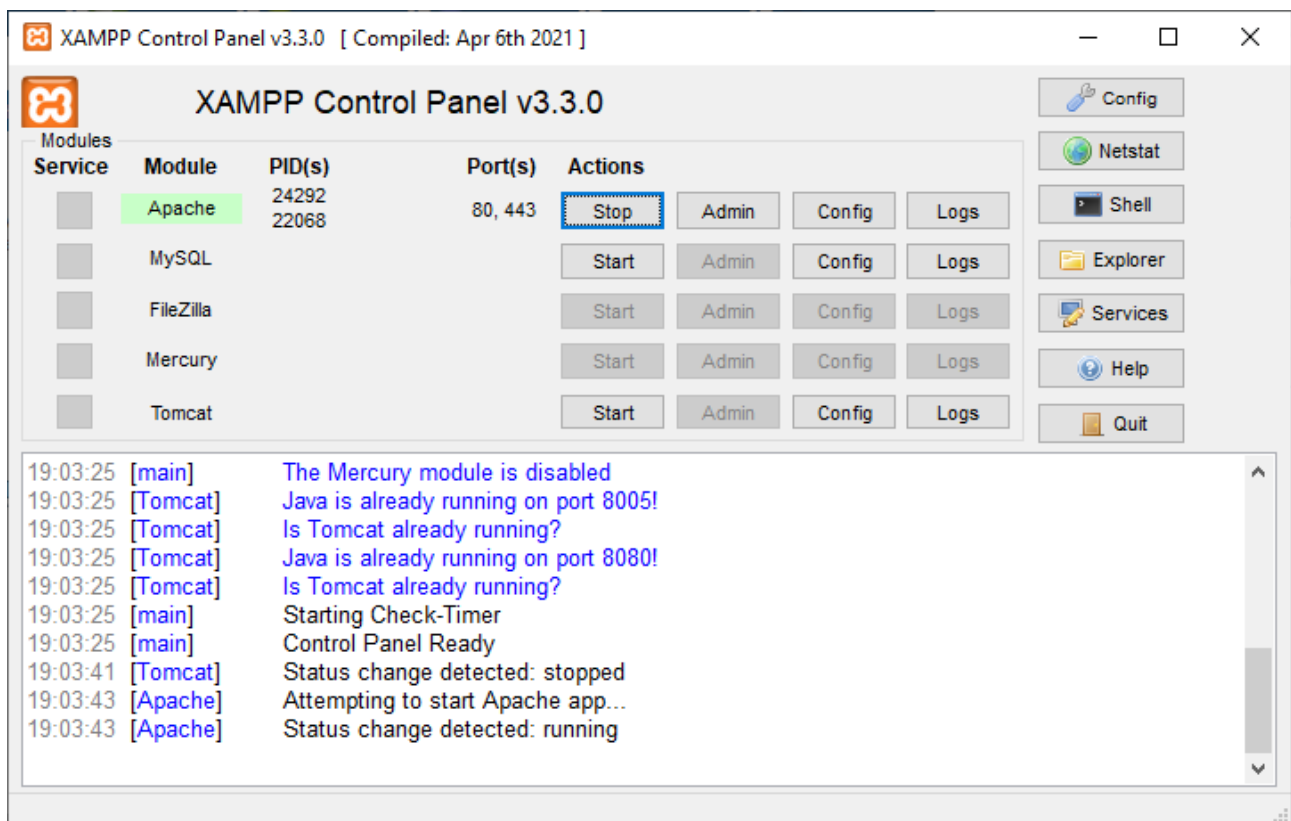
Y lo descomprimos en **C:\SERVIDORES**.

Para configurarlo ejecutaremos el archivo:

**C:\SERVIDORES\xampp\setup\_xampp.bat**

Ahora iniciaremos **Apache en XAMPP** utilizando el ejecutable y pulsando en el botón **Start**:

**C:\SERVIDORES\xampp\xampp-control.exe**

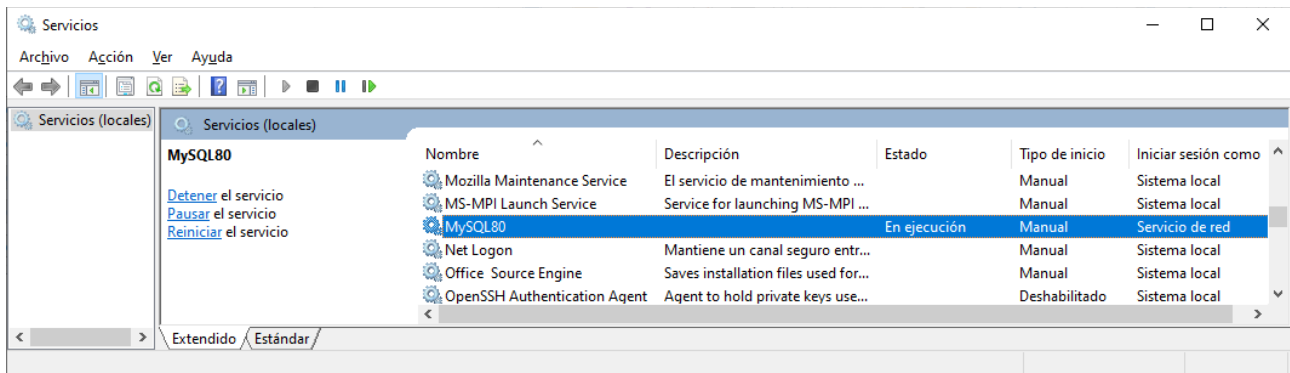


Para probar nuestros archivos **html** y **php**, eliminaremos el contenido de la carpeta **htdocs** y los copiaremos ahí.

Veremos el resultado de cada archivo abriendo en el navegador:

**<http://localhost>**

Hay que recordar que para que funcionen los archivos que conectan con **MySQL Server** el servicio **MySQL80** debe estar iniciado:



## **HTML**

El primer archivo que copiaremos en la carpeta **C:\SERVIDORES\xampp\htdocs** será:

**ejemplo\_table.html**

Este archivo muestra una tabla de datos de libros utilizando el lenguaje de marcas HTML5.

Veamos el contenido de esta página web básica en **HTML** con una tabla de datos:

#### ejemplo\_table.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8"/>
  <title>DAM - Tabla de Datos - HTML</title>
</head>
<body>
  <div>
    <p>Número de registros: 7</p>
  </div>
  <table border="1" style="border-collapse: collapse">
    <thead style="background-color: orange">
      <tr>
        <td>ID LIBRO</td>
        <td>TÍTULO</td>
        <td>AUTOR</td>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>Macbeth</td>
        <td>William Shakespeare</td>
      </tr>
      <tr>
        <td>2</td>
        <td>La Celestina (Tragicomedia de Calisto y Melibea)</td>
        <td>Fernando de Rojas</td>
      </tr>
      <tr>
        <td>3</td>
        <td>El Lazarillo de Tormes</td>
        <td>Anónimo</td>
      </tr>
      <tr>
        <td>4</td>
        <td>20.000 Leguas de Viaje Submarino</td>
        <td>Julio Verne</td>
      </tr>
      <tr>
        <td>5</td>
        <td>Alicia en el País de las Maravillas</td>
        <td>Lewis Carrol</td>
      </tr>
      <tr>
        <td>6</td>
        <td>Cien Años de Soledad</td>
        <td>Gabriel García Márquez</td>
      </tr>
      <tr>
        <td>7</td>
        <td>La tempestad</td>
        <td>William Shakespeare</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

## PHP

Para el caso anterior, otra opción es disponer de los datos en un **array** y recorrerlo para mostrar la tabla utilizando **lenguaje de script PHP**:

### ejemplo\_table.php

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8"/>
  <title>DAM - Tabla de Datos - PHP</title>
</head>
<body>
<?php

    $datos=array();

    $registro=array();
    $registro['id']=1;
    $registro['titulo']='Macbeth';
    $registro['autor']='William Shakespeare';
    $datos[]=$registro;

    $registro=array();
    $registro['id']=2;
    $registro['titulo']='La Celestina (Tragicomedia de Calisto y Melibea)';
    $registro['autor']='Fernando de Rojas';
    $datos[]=$registro;

    $registro=array();
    $registro['id']=3;
    $registro['titulo']='El Lazarillo de Tormes';
    $registro['autor']='Anónimo';
    $datos[]=$registro;

    $registro=array();
    $registro['id']=4;
    $registro['titulo']='20.000 Leguas de Viaje Submarino';
    $registro['autor']='Julio Verne';
    $datos[]=$registro;

    $registro=array();
    $registro['id']=5;
    $registro['titulo']='Alicia en el País de las Maravillas';
    $registro['autor']='Lewis Carrol';
    $datos[]=$registro;

    $registro=array();
    $registro['id']=6;
    $registro['titulo']='Cien Años de Soledad';
    $registro['autor']='Gabriel García Márquez';
    $datos[]=$registro;

    $registro=array();
    $registro['id']=7;
    $registro['titulo']='La tempestad';
    $registro['autor']='William Shakespeare';
    $datos[]=$registro;
```

?>

A continuación tendríamos el bucle para mostrar en HTML las líneas de la tabla:

```

<div>
    <p>Número de registros: <?php echo count($datos); ?></p>
</div>

<?php if (count($datos)>0) { ?>
    <table border="1" style="border-collapse:collapse">
        <thead style="background-color:orange">
            <tr>
                <td>ID LIBRO</td>
                <td>TÍTULO</td>
                <td>AUTOR</td>
            </tr>
        </thead>
        <tbody>
            <?php
                for ($i=0; $i<count($datos); $i++) {
                    $registro=$datos[$i];
                ?>
                    <tr>
                        <td><?php echo $registro['id']; ?></td>
                        <td><?php echo $registro['titulo']; ?></td>
                        <td><?php echo $registro['autor']; ?></td>
                    </tr>
                <?php
                }
            ?>
        </tbody>
    </table>
    <?php
    } else {
        ?>
        <div>
            <p>No hay datos que mostrar</p>
        </div>
        <?php
        }
    ?>
</body>
</html>

```

Como podemos observar el **código de script PHP** está intercalado con el lenguaje de marcas **HTML**, por lo que después de ejecutarse en el servidor, obtendremos una salida que contendrá exclusivamente código en HTML, manteniendo el HTML que ya existía en el archivo.



## PHP con acceso a BD

La gran potencia de los lenguajes script en servidores web radica en la utilización de **bases de datos**, es decir, para cargar los datos utilizaremos código que leerá a través de SQL de una BD.

Para este ejemplo deberemos arrancar nuestro **MySQL Server** que ya contiene la base de datos **bibliotecah**.

Cambiaremos por tanto solo la primera parte donde se carga el array de datos:

### ejemplo\_select.php

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8"/>
  <title>DAM - Tabla de Datos - PHP y SQL</title>
</head>
<body>
<?php
    $datos=array();

    $servername = "localhost";
    $username = "root";
    $password = "1234";
    $dbname = "bibliotecah";

    // Create connection
    $conn = new mysqli($servername, $username, $password, $dbname);

    // Check connection
    if ($conn->connect_error) {
        die("Error en la conexión: " . $conn->connect_error);
    }

    $sql = "SELECT * FROM libros";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        while($registro = $result->fetch_array()) {
            $campos=array();
            foreach ($registro as $campo => $valor) {
                if (!is_int($campo)) {
                    $campos[$campo]=$valor;
                }
            }
            $datos[]=$campos;
        }
    }
    $conn->close();
?>
```

El resultado es el mismo, pero ahora cambiando el contenido de la base de datos, el resultado de la página cambiará.

También podemos crear el contenido de los datos en formato JSON, en vez de HTML.

PHP proporciona funciones para crear este contenido fácilmente:

#### ejemplo\_select\_json.php

```
<?php
```

```
    $datos=array();

    $servername = "localhost";
    $username = "root";
    $password = "1234";
    $dbname = "bibliotecah";

    // Create connection
    $conn = new mysqli($servername, $username, $password, $dbname);
    // Check connection
    if ($conn->connect_error) {
        die("Error en la conexión: " . $conn->connect_error);
    }

    $sql = "SELECT * FROM libros";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        while($registro = $result->fetch_array()) {
            $campos=array();
            foreach ($registro as $campo => $valor) {
                if (!is_int($campo)) {
                    $campos[$campo]=$valor;
                }
            }
            $datos[]=$campos;
        }
    }
    $conn->close();

    header('Content-Type: application/JSON');
    echo json_encode($datos, JSON_PRETTY_PRINT);
```

```
?>
```

## 7.3 Apache Tomcat

**JSP (JavaServer Pages)** es una tecnología que permite generar documentos en formato HTML de manera dinámica, utilizando el lenguaje de programación **Java**.

**Applet** es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo, en un navegador web.

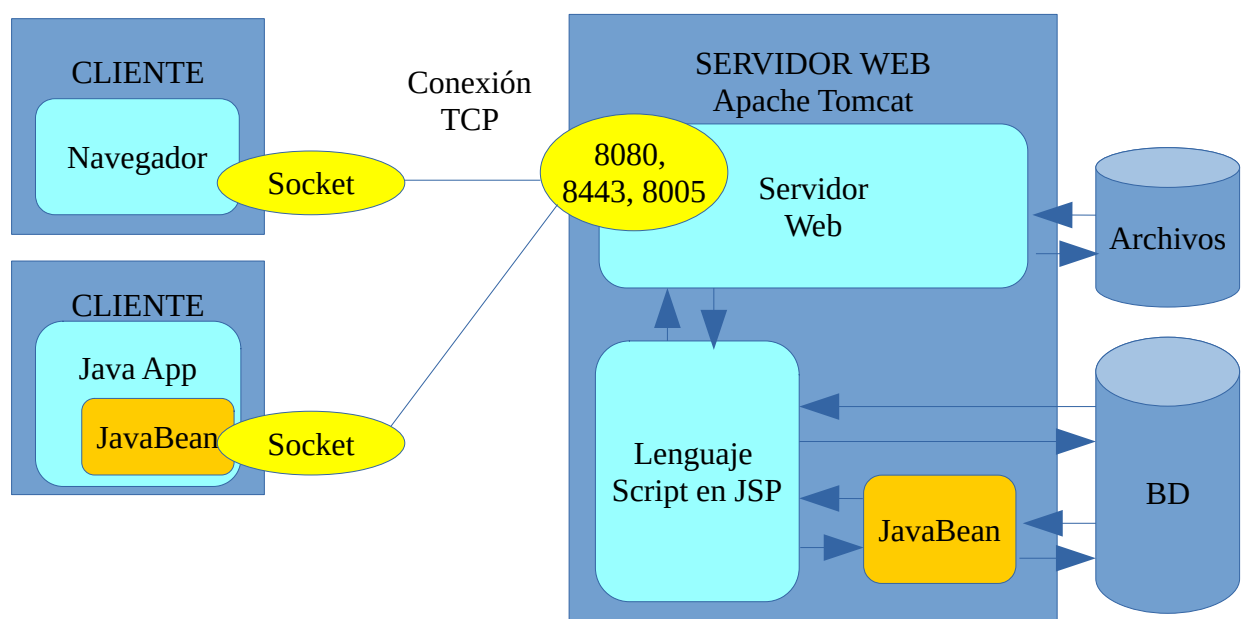
Un **Servlet** es una clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor.

Aunque los servlets pueden responder a cualquier tipo de solicitudes, estos son utilizados comúnmente para extender las aplicaciones alojadas por servidores web, de tal manera que pueden ser vistos como applets de Java que se ejecutan en servidores en vez de navegadores web.

Para poder utilizar **JSP (Java Server Pages)** o **Servlets** vamos a utilizar Apache Tomcat, uno de los servidores más conocidos por los desarrolladores de Java.

Existen también versiones preconfiguradas como la que incorpora **XAMPP** pero nosotros utilizaremos la oficial para disponer de la última versión.

**Apache Tomcat**, también llamado **Jakarta Tomcat** o sólo **Tomcat**, es un servidor web que utiliza como lenguaje script del servidor JSP (Java Server Pages). Incluye el compilador Jasper, compila JSPs convirtiéndolas en Servlets y también puede funcionar como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.



Podemos observar en el esquema que Apache Tomcat con JSP incorpora la posibilidad de acceder a JavaBean.

**JavaBean** es un componente de software para la plataforma Java SE.

Especificaciones que se debe tener en cuenta para que una clase sea un JavaBean:

- Debe tener un **constructor vacío**
- Debe implementar la interfaz **Serializable**
- Las **propiedades/atributos** deben ser **privados**.
- Debe tener métodos **getters o setters** o ambos que permitan acceder a sus propiedades

Esto nos permite utilizar archivos (JavaBean) empaquetados en JAR para ampliar las posibilidades de la aplicación, como muchas de las librerías que hemos añadido a nuestros proyectos.

### **Instalación**

XAMPP dispone de una versión de Apache Tomcat pero no es la última.

Además, es recomendable instalar solo Apache Tomcat y así evitarnos instalar otro software que no vayamos a usar.

Para instalar **Apache Tomcat**, basta con descargar el software y descomprimirlo en una carpeta del servidor. Nosotros descargaremos la **última versión**.

#### **Apache Tomcat – Web Oficial**

<https://tomcat.apache.org/>

<https://tomcat.apache.org/whichversion.html>

### **Configuración Apache Tomcat - Variables**

Es necesario disponer de dos variables al sistema para que pueda localizar Java.

**JAVA\_HOME=C:\Program Files\Java\jdk-~~X.X.X.X~~**

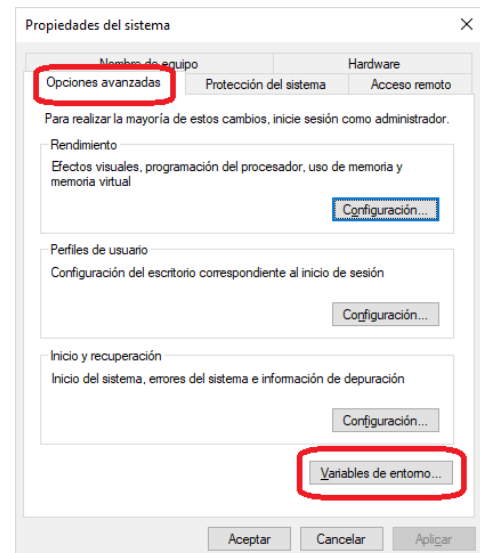
**JRE\_HOME=C:\Program Files\Java\jdk-~~X.X.X.X~~**

## Opción 1)

Para ello abriremos las propiedades del equipo con el comando **sysdm.cpl**. En "**Opciones avanzadas**", pulsar en "Variables de Entorno".

Seguramente **JAVA\_HOME** ya existirá y **faltarán JRE\_HOME**.

Deberemos tener las dos variables en el apartado de "**Variables de sistema**"



Variables del sistema	
Variable	Valor
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
JAVA_HOME	C:\Program Files\Java\jdk-11.0.12
JRE_HOME	C:\Program Files\Java\jdk-11.0.12

## Opción 2)

Crear archivo **setenv.bat** en la carpeta **bin** de **Apache Tomcat** con las siguientes instrucciones:

```
set JAVA_HOME=C:\Program Files\Java\jdk-X.X.X.X
set JRE_HOME=C:\Program Files\Java\jdk-X.X.X.X
exit /b 0
```

Estas variables hay que actualizarlas si se actualiza la versión de Java.

## Configuración Apache Tomcat – Carpetas

Apache tiene varias carpetas importantes:

<b>tomcat\webapps\ROOT</b>	Alojamiento de las páginas HTML y JSP
<b>tomcat\conf</b>	Archivos de configuración del servidor
<b>tomcat\lib</b>	Archivos jar cargados en memoria por el servidor y disponibles para los archivos JSP. Cada vez que se incluya uno nuevo, es necesario reiniciar el servidor. Los drivers JDBC irán en este directorio

## Configuración Apache Tomcat – Puertos

Los puertos que usa por defecto son: 8080, 8005 y 8443.

Estos puertos pueden cambiarse en el archivo:

**tomcat\conf\server.xml**

Recuerda que el comando para ver los puertos ocupados en windows desde CMD es:

```
C:\> netstat -a -p TCP | find "LISTENING"
```

## Configuración Apache Tomcat – Usuarios

Para configurar Apache Tomcat debemos crear un usuario administrador. Para ello

Para ello editaremos y añadiremos al archivo:

**tomcat\conf\tomcat-users.xml**

```
<tomcat-users ...>
...
<user roles="manager-gui, manager-script,admin-gui, admin-script"
    username="tomcat"
    password="1234" />
...
</tomcat-users>
```

## Configuración Apache Tomcat – Arrancar y parar

Para arrancar el servicio tenemos el archivo bat:

**tomcat\bin\startup.bat**

Para parar el servicio tenemos el archivo bat:

**tomcat\bin\shutdown.bat**

## Configuración Apache Tomcat – Añadir componentes JAR

Todos los archivos JAR que vayan a utilizar nuestra páginas web en JSP hay que copiarlos a la carpeta:

**tomcat\lib**

Y luego reiniciar el servicio

## **Configuración Apache Tomcat – URL**

Tendremos dos URL predefinidas para acceder a Apache Tomcat:

<b>URL Pública</b>	<a href="http://localhost:8080">http://localhost:8080</a>
<b>URL de Administración</b>	<a href="http://localhost:8080/manager/html">http://localhost:8080/manager/html</a>

La URL Pública navega sobre el contenido de la carpeta:

**tomcat\webapps\ROOT**

La URL de Administración navega sobre el Manager GUI

## **Configuración Apache Tomcat – Prueba**

Crearemos la carpeta donde alojaremos nuestros archivos HTML, CSS, JS y JSP.

**tomcat\webapps\ROOT\ad**

<b>Prueba del servidor Apache Tomcat</b>
Arranca el servicio de Apache Tomcat.  Prueba a copiar una página html <b>ejemplo_table.html</b> en la carpeta " <b>tomcat\webapps\ROOT\ad</b> " y acceder a ella desde el navegador con:  <a href="http://localhost:8080/ad/ejemplo_table.html">http://localhost:8080/ad/ejemplo_table.html</a>

Una vez configurado y probado, podemos parar el servicio y hacer un zip de la carpeta de Apache Tomcat para tener una copia.

## 7.4 Lenguaje de script JSP

Haciendo la equivalencia con PHP, podríamos tener la tabla de libros del ejemplo:

### ejemplo\_table.jsp

```
<%@page import="java.util.regex.Pattern"%>
<%@page import="java.util.ArrayList"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>DAM - Tabla de Datos - JSP</title>
</head>
<body>
<%
  ArrayList<String> datos = new ArrayList<>();

  datos.add("1|Macbeth|William Shakespeare");
  datos.add("2|La Celestina (Tragicomedia de Calisto y Melibea)|Fernando de Rojas");
  datos.add("3|El Lazarillo de Tormes|Anónimo");
  datos.add("4|20.000 Leguas de Viaje Submarino|Julio Verne");
  datos.add("5|Alicia en el País de las Maravillas|Lewis Carroll");
  datos.add("6|Cien Años de Soledad|Gabriel García Márquez");
  datos.add("7|La tempestad|William Shakespeare");
%>

  <div>
    <p>Número de registros: <%= datos.size() %></p>
  </div>

  <%
    if (datos.size()>0) { %>
      <table border="1" style="border-collapse:collapse">
        <thead style="background-color:orange">
          <tr>
            <td>ID LIBRO</td>
            <td>TÍTULO</td>
            <td>AUTOR</td>
          </tr>
        </thead>
        <tbody>
          <%
            for (int i=0; i<datos.size(); i++) {
              String[] registro = datos.get(i).split(Pattern.quote("|"));
            %>

              <tr>
                <td><% out.println(registro[0]); %></td>
                <td><% out.println(registro[1]); %></td>
                <td><% out.println(registro[2]); %></td>
              </tr>

            <%
              }
            %>
          </tbody>
        </table>
      <%
        } else {
      %>

      <div>
        <p>No hay datos que mostrar</p>
      </div>

      <%
        }
      %>
    </body>
  </html>
```



## JSP con acceso a BD

Cambiaremos ahora solo la primera parte para leer de la base de datos biblioteca como ya lo habíamos realizado en temas anteriores:

### ejemplo\_select.jsp

```
<%@page import="java.sql.Connection"%>
<%@page import="java.util.regex.Pattern"%>
<%@page import="java.util.ArrayList"%>
<%@page import="java.sql.Statement"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.DriverManager"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>DAM - Tabla de Datos - JSP y SQL</title>
</head>
<body>
<%
  //Estructura de datos para almacenar
  ArrayList<String> datos = new ArrayList<>();

  // Conexión a la BD
  String url;

  Class.forName("com.mysql.cj.jdbc.Driver");
  url = "jdbc:mysql://localhost:3306/bibliotecah?autoReconnect=true";
  url += "&useSSL=false&zeroDateTimeBehavior=convertToNull&serverTimezone=UTC";
  String usuario = "root";
  String password = "1234";
  Connection con = DriverManager.getConnection(url, usuario, password);

  // Crear Statement de la Consulta
  String sentenciaSQL = "SELECT id, titulo, autor FROM libros";
  Statement statement = con.createStatement();

  // ResultSet
  ResultSet rs = statement.executeQuery(sentenciaSQL);
  while (rs.next()) {
    datos.add(String.valueOf(rs.getInt(1))+"|" +rs.getString(2)+"|" +rs.getString(3));
  }
  rs.close();

  // Cerrar conexión
  statement.close();
  con.close();
%>
...

```

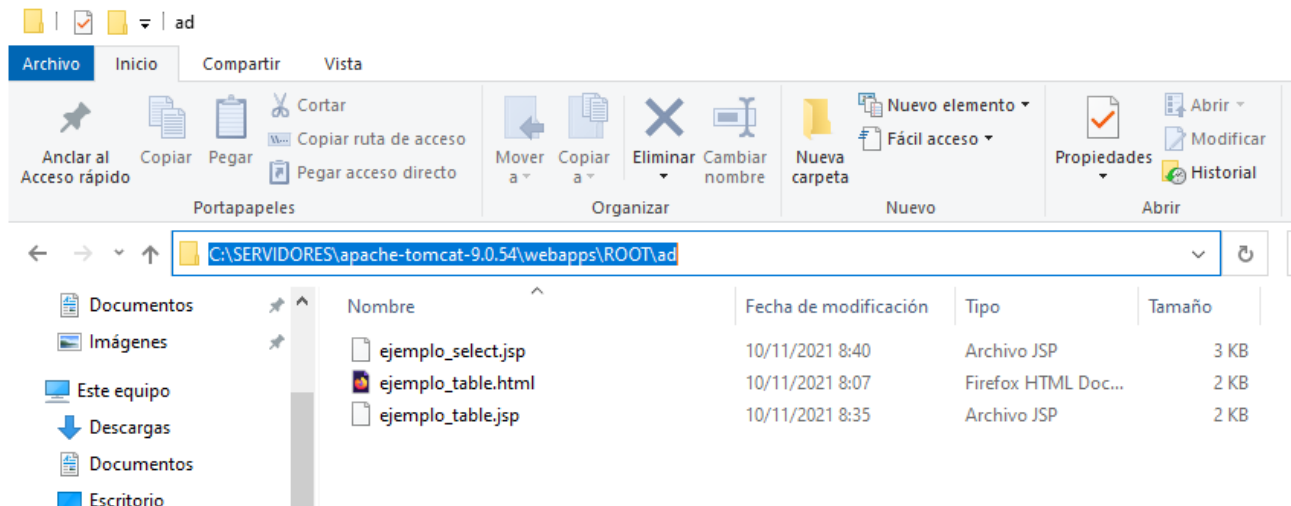
Para probar estos dos nuevos ejemplos, los copiaremos también en la carpeta:

**tomcat\webapps\ROOT\ad**

Recuerda que para que funcione correctamente el **ejemplo\_select.jsp** necesita de la librería jar del **driver JDBC de mysql**, por lo que previamente habrá que copiarla a la carpeta de **librerías de tomcat** y posteriormente reiniciar el servicio:

**tomcat\lib\mysql-connector-java-8.0.21-bin**

Iniciado el servidor web Apache Tomcat con el driver de MySQL Server y con los archivos en la carpeta indicada:



podremos navegar sobre ellos para ver el resultado:

[http://localhost:8080/ad/ejemplo\\_table.jsp](http://localhost:8080/ad/ejemplo_table.jsp)

[http://localhost:8080/ad/ejemplo\\_select.jsp](http://localhost:8080/ad/ejemplo_select.jsp)

Número de registros: 7

ID LIBRO	TÍTULO	AUTOR
1	Macbeth	William Shakespeare
2	La Celestina (Tragicomedia de Calisto y Melibea)	Fernando de Rojas
3	El Lazarillo de Tormes	Anónimo
4	20.000 Leguas de Viaje Submarino	Julio Verne
5	Alicia en el País de las Maravillas	Lewis Carrol
6	Cien Años de Soledad	Gabriel García Márquez
7	La tempestad	William Shakespeare

## 8. Creación de aplicaciones web

### 8.1 NetBeans con Apache Tomcat

Vamos a realizar nuestro primer proyecto Web usando **Apache Tomcat**.

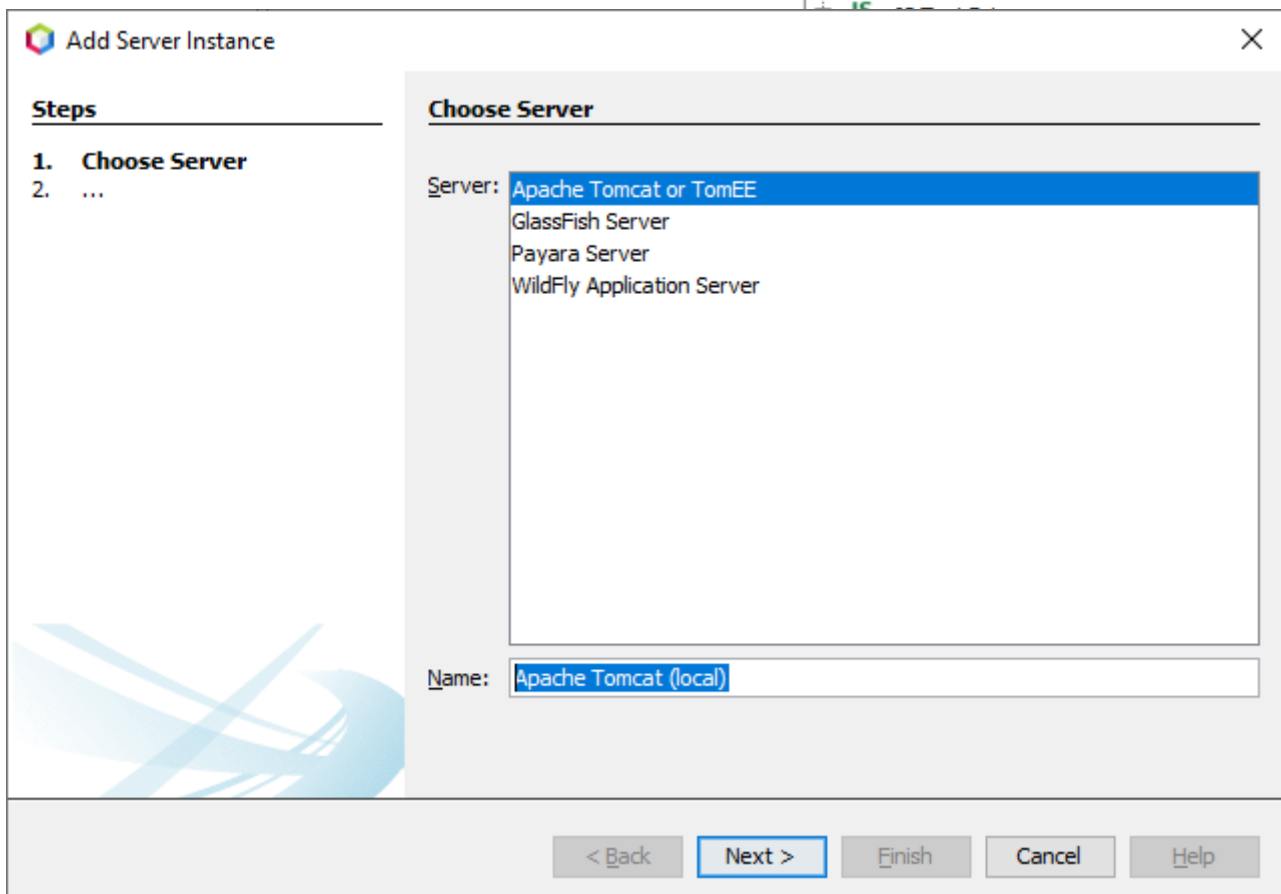
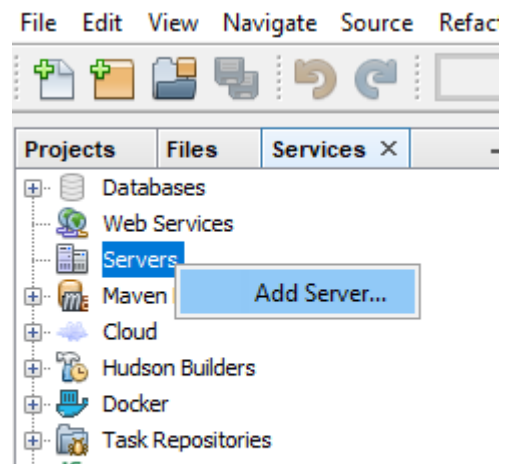
En desarrollo, es recomendable tener el servicio parado y que NetBeans gestione el arranque y la parada de nuestro servidores.

#### Añadir el servidor Apache Tomcat a Netbeans

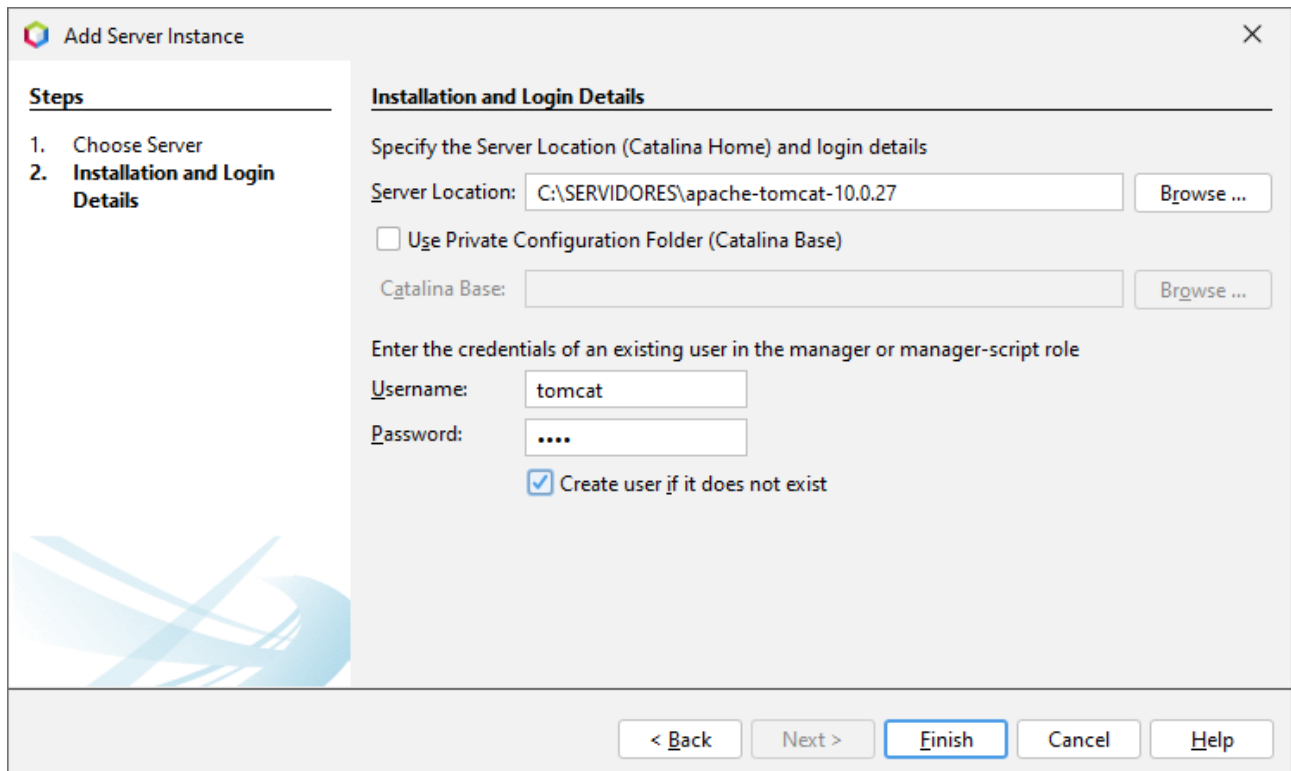
En la pestaña de **Services** añadiremos nuestro servidor **Apache Tomcat** en **Servers** pulsando con botón derecho:

Ahora seleccionamos el tipo de servidor web con el nombre de:

**Apache Tomcat (local)**



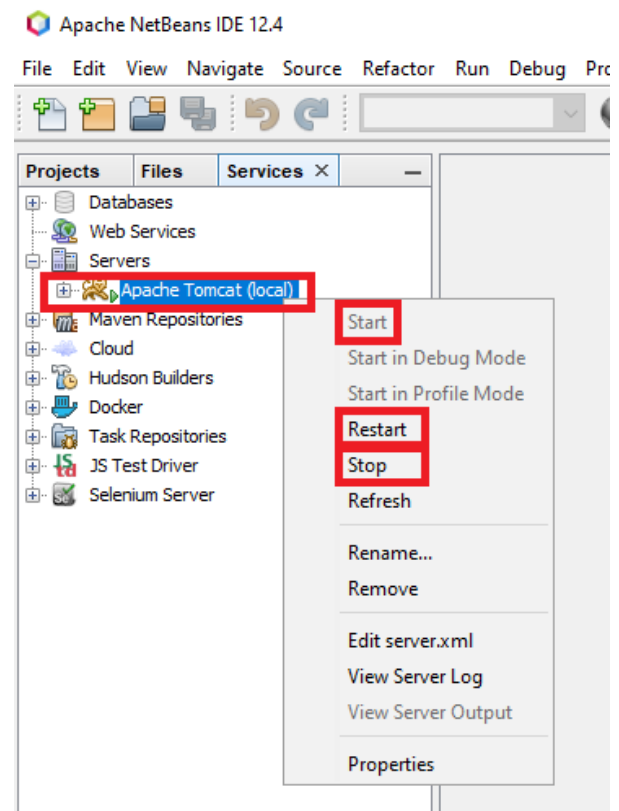
En la siguiente pantalla indicaremos la carpeta de instalación de nuestro Apache Tomcat y el usuario administrador **tomcat** con su contraseña **1234** que hemos configurado anteriormente.



Desde NetBeans podemos iniciar, reiniciar o parar el servidor utilizando el botón derecho.

**CUIDADO:** Si falla al arrancar, puede ser por tener el **proxy** de sistema activado. Debemos activar "**No proxy**" o "**Manual**" en

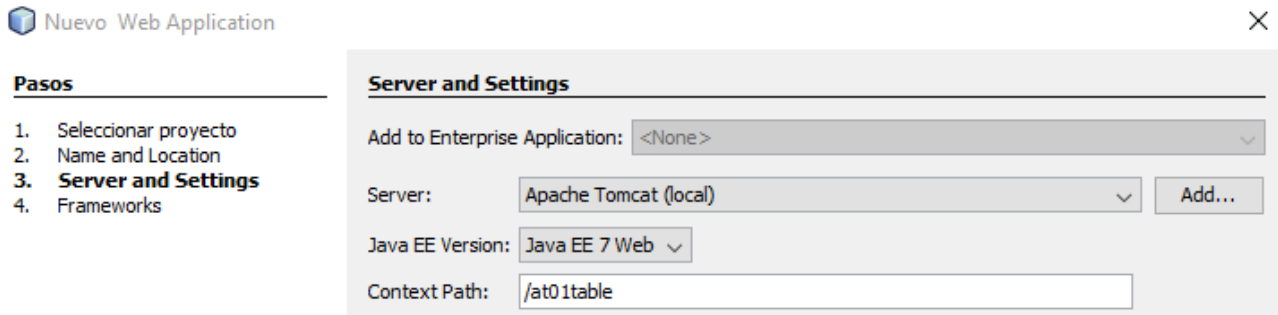
**Tools → Options → General**



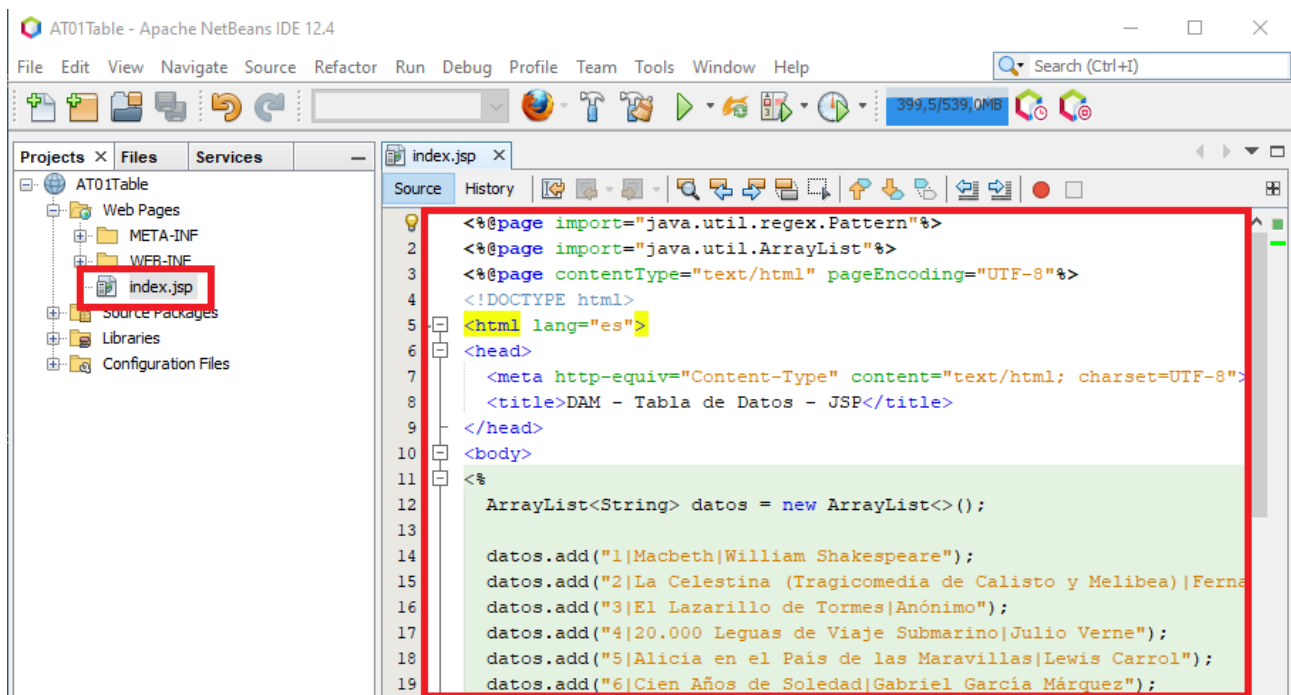
## Crear proyecto de Java Web => Web Application

Vamos a crear el proyecto **AT01Table** que ejecute el código de **ejemplo\_table.jsp**.

Crearemos un proyecto de tipo "**Java with Ant => Java Web => Web Application**" poniendo como path raíz del App Web (Context Path Root) **at01table** y como servidor nuestro **Apache Tomcat (local)**.



Ahora solo queda eliminar el **index.html** que ha generado NetBeans y generar un nuevo archivo en "Web Pages" de tipo "Web => JSP" denominado **index.jsp** con el contenido de nuestro **ejemplo\_table.jsp**.



Cuando ejecutemos el proyecto, si **Apache Tomcat** está parado, **NetBeans** iniciará el servidor (**Start**).

Hay que tener en cuenta que al cerrar NetBeans se parará (**Stop**).

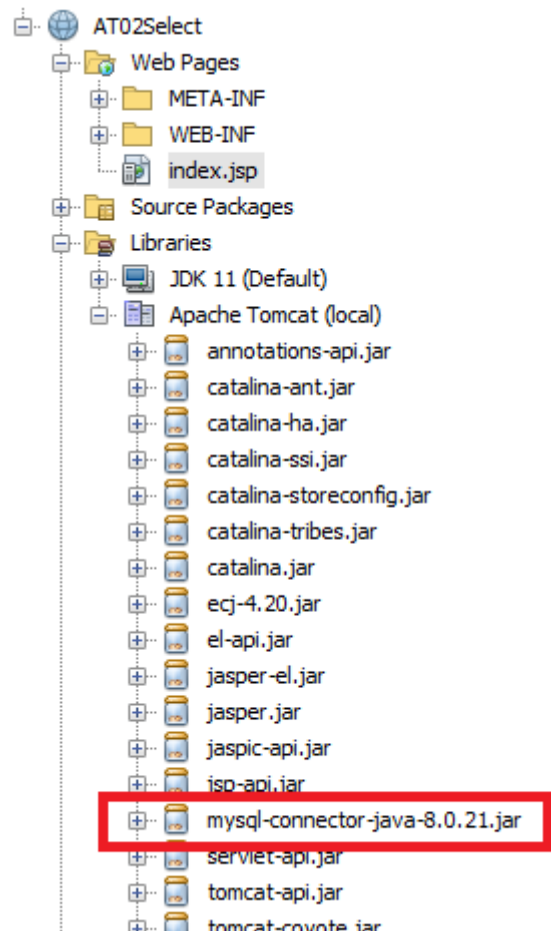
## Crear proyecto de Java Web => Web Application

Vamos a crear el proyecto **AT02Select** que ejecute el código de **ejemplo\_select.jsp**.

Realizaremos el mismo proceso que en el anterior proyecto, pero poniendo como path raíz del App Web (Context Path Root) **at02select** y copiando el contenido de nuestro **ejemplo\_select.jsp** en el archivo **index.jsp**.

No es necesario copiar la librería jar del driver de mysql a la aplicación porque ya está en **tomcat\lib** del Apache Tomcat y si desplegamos las librerías incluidas en el servidor la podremos ver.  
(ver imagen lateral)

De todas formas la podemos añadir de nuestra carpeta lib del proyecto como en proyectos de consola y se enviará al servidor.



## Crear proyecto de Java Web => Web Application

Vamos a crear el proyecto **AT03Persona** que use una librería nuestra (componente JAR).

Abrir NetBeans y realizar el mismo proceso que en el anterior proyecto, pero poniendo como path raíz del App Web (Context Path Root) **at03persona**.

Lo siguiente es copiar el componente **Persona.jar**, creado en apartados anteriores, a la carpeta **lib** del proyecto y añadir en **Librerías**.

Luego crearemos en el proyecto **AT03Persona** nuestro **index.jsp** donde copiaremos el siguiente código JSP. Como se aprecia en el ejemplo, usamos la clase **Persona** y **Domicilio** como en un proyecto "Java Application".

## index.jsp

```

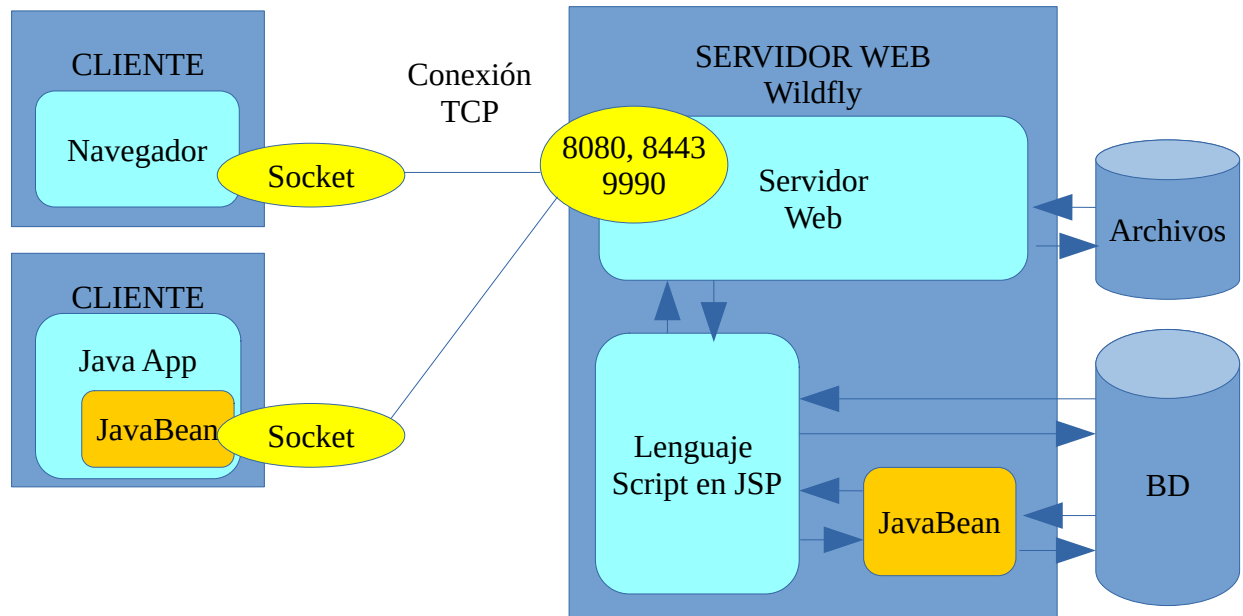
<%@page import="com.dam.persona.Domicilio"%>
<%@page import="com.dam.persona.Persona"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
    <style>
      table { border-collapse: collapse; }
      table, td { border: 1px solid black; }
      .cabecera { background-color: orange; font-weight: bold; }
    </style>
  </head>
  <body>
    <h1>Persona</h1>
    <p><%
      Persona persona = new Persona(21444555,"Sergio", "Fernández", "61277788",
        new Domicilio("C/Jaume de Scals, 35", "03100", "Xixona", "Alicante"));
      out.println(persona.toString());
    %></p>
    <table>
      <tr>
        <td class="cabecera">DNI</td>
        <td><%= persona.getDni() %></td>
      </tr>
      <tr>
        <td class="cabecera">NOMBRE</td>
        <td><%= persona.getNombre() %></td>
      </tr>
      <tr>
        <td class="cabecera">APELLIDOS</td>
        <td><%= persona.getApellidos() %></td>
      </tr>
      <tr>
        <td class="cabecera">TELÉFONO</td>
        <td><%= persona.getTelefono() %></td>
      </tr>
      <tr>
        <td class="cabecera">DIRECCIÓN</td>
        <td><%= persona.getDomicilio().getDireccion() %></td>
      </tr>
      <tr>
        <td class="cabecera">CÓDIGO POSTAL</td>
        <td><%= persona.getDomicilio().getCpostal() %></td>
      </tr>
      <tr>
        <td class="cabecera">POBLACIÓN</td>
        <td><%= persona.getDomicilio().getPoblacion() %></td>
      </tr>
      <tr>
        <td class="cabecera">PROVINCIA</td>
        <td><%= persona.getDomicilio().getProvincia() %></td>
      </tr>
    </table>
  </body>
</html>

```

## 8.2 NetBeans con Wildfly

### Instalación de Wildfly

El servidor de Wildfly tiene el mismo esquema de Apache Tomcat pero cambiando los puertos:



**Wildfly** es un servidor de aplicaciones de software libre, que implementa las tecnologías definidas en la plataforma **Java EE** y permite ejecutar aplicaciones que siguen esta especificación.

**WildFly** (formalmente WildFly Application Server), anteriormente conocido como JBoss AS, o simplemente **JBoss**, es un servidor de aplicaciones Java EE de código abierto implementado en Java puro, más concretamente la especificación Java EE. Al estar basado en Java, JBoss puede ser utilizado en cualquier sistema operativo para el que esté disponible la máquina virtual de Java.

**Java EE** incorpora **JAX-RS** para el desarrollo de API Rest facilitando la sintaxis y mejorando el desarrollo de aplicaciones web.

### Instalación

Para instalar Wildfly, basta con descargar el software y descomprimirlo en **C:\SERVIDORES**.

**Wildfly – Web Oficial**

<https://www.wildfly.org/downloads/>



## Configuración Wildfly – Carpetas

Wildfly tiene varias carpetas importantes:

<b>bin</b>	Utilidades
<b>welcome-content</b>	Alojamiento de las páginas HTML y JSP
<b>domain\configuration</b> <b>standalone\configuration</b>	Archivos de configuración del servidor
<b>modules\system\layers\base</b>	Archivos jar cargados en memoria por el servidor y disponibles para los archivos JSP. Cada vez que se incluya uno nuevo, es necesario reiniciar el servidor. Los drivers JDBC irían en este directorio

## Configuración Wildfly – Usuarios

Posteriormente hay que generar el usuario administrador. Para ello ejecutaremos el script **add-user.bat** que se encuentra en la carpeta **bin**:

```

What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a): a

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username: wildfly
Password: 1234
WFLYDM0099: Password should have at least 8 characters!
Are you sure you want to use the password entered yes/no? yes
Re-enter Password : 1234

What groups do you want this user to belong to? (Please enter a comma separated list, or leave
blank for none)[ ]: intro
About to add user 'wildfly' for realm 'ManagementRealm'
Is this correct yes/no? yes

Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to
server EJB calls.
yes/no? no
  
```

## Configuración Wildfly – Puertos

Los puertos que usa por defecto son: 8080 y 9990.

Estos puertos pueden cambiarse en el archivo:

**standalone/configuration/standalone-full.xml**

Buscamos el puerto 8080 que estará en la sección:

```
<socket-binding-group
  name="standard-sockets"
  default-interface="public"
  port-offset="{jboss.socket.binding.port-offset:0}">
...
  <socket-binding name="http" port="{jboss.http.port:8081}" />
...
```

Hay dos formas de cambiar los puertos, individual o bien todos los puertos a la vez. Para cambiar un puerto individualmente, buscamos en este grupo el puerto a cambiar y lo modificamos. Por ejemplo, podemos cambiar el valor del puerto http de 8080 a 8081.

Nosotros cambiaremos el puerto 8080 por el 8081 para poder tener Apache Tomcat y Wildfly funcionando simultáneamente.

Para cambiar todos los puertos a la vez podemos hacer uso del atributo **port-offset**. El atributo port-offset indica lo que le sumaremos al puerto por defecto para obtener el puerto que finalmente se usará. Esto nos permite modificar todos los puertos que usa el servidor con sólo cambiar este parámetro. Por defecto el valor es 0, lo que significa que, por ejemplo, el puerto http será 8080, que es el valor por defecto. De esta manera si ponemos un offset, por ejemplo, de 100, habremos cambiado este puerto a 8180 (8080 + 100). De la misma manera habremos cambiado el resto de puertos.

Recuerda que el comando para ver los puertos ocupados en windows es:

```
C:\> netstat -a -p TCP | find "LISTENING"
```

## Configuración Wildfly – REALIZADA

En nuestro caso, solo hemos:

- añadido el usuario **wildfly** y
- cambiado el puerto 8080 por **8081**.

### Configuración Wildfly – Arrancar y parar

Para arrancar el servicio tenemos el archivo bat:

**bin/standalone.bat**

Para parar el servicio desde CMD tenemos el archivo bat con el parámetro **connect**:

**bin/jboss-cli.bat --connect command=:shutdown**

### Configuración Wildfly – Añadir componentes JAR

Todos los archivos JAR que vayan a utilizar nuestra páginas web en JSP hay que copiarlos como en cualquier proyecto Java Application con una carpeta **lib**. Estos archivos se copiarán al desplegarse en el servidor en la carpeta **WEB-INF**.

### Configuración de JAVA para Wildfly

Por defecto usa la variable **JAVA\_HOME**, pero si deseamos indicar a Wildfly dónde se encuentra la versión de Java que deseamos utilizar se debe indicar la versión en la variable **JAVA\_HOME** que hay en el archivo:

**bin/standalone.conf**

### Configuración Wildfly – URL

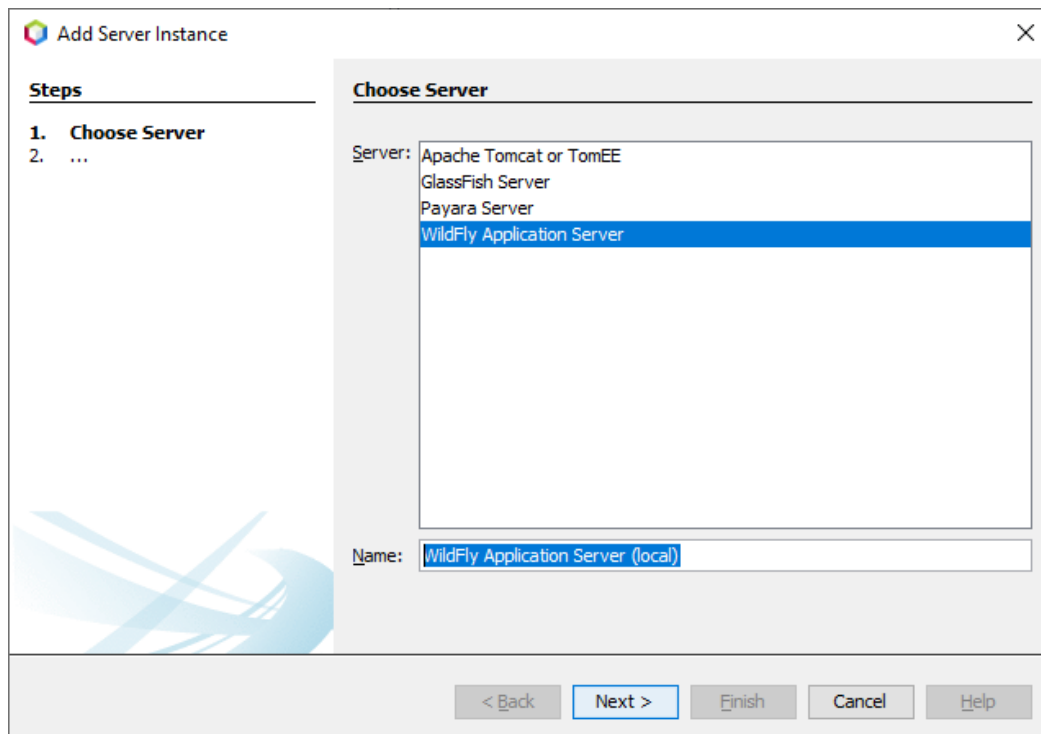
Una vez configurado e iniciado, tendremos dos URL predefinidas para acceder a Wildfly:

<b>URL Pública</b>	<a href="http://localhost:8081">http://localhost:8081</a>
<b>URL de Administración</b>	<a href="http://localhost:9990">http://localhost:9990</a>

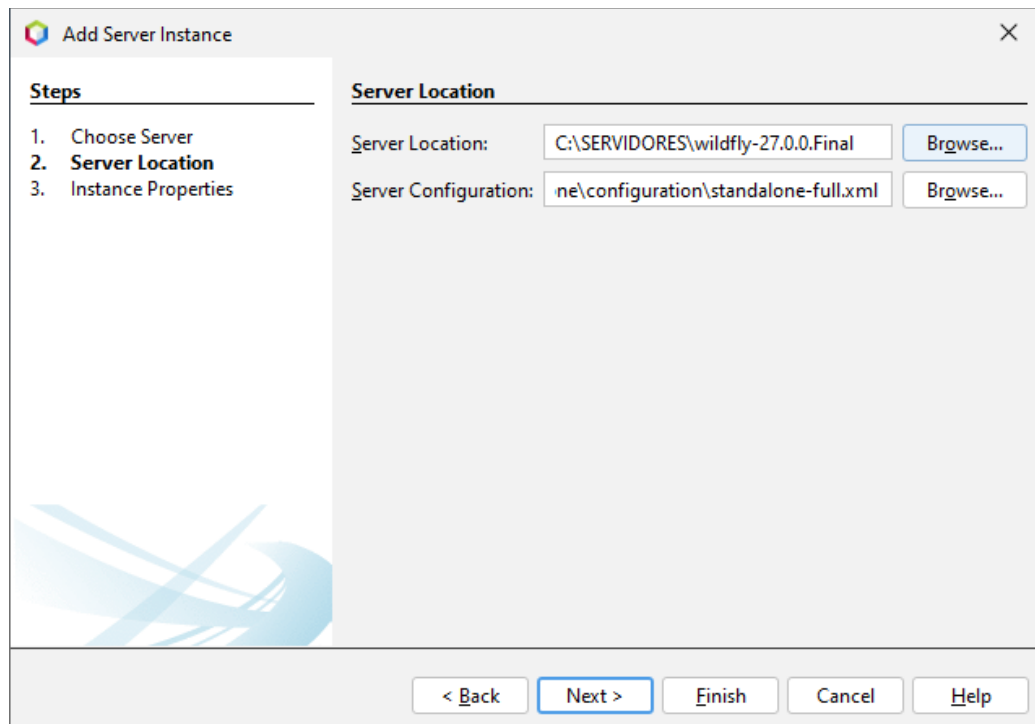
Nosotros usaremos Wildfly integrado en NeBeans, por lo que no necesitaremos iniciarlo desde los archivos bt.

## Añadir el servidor Wildfly a Netbeans

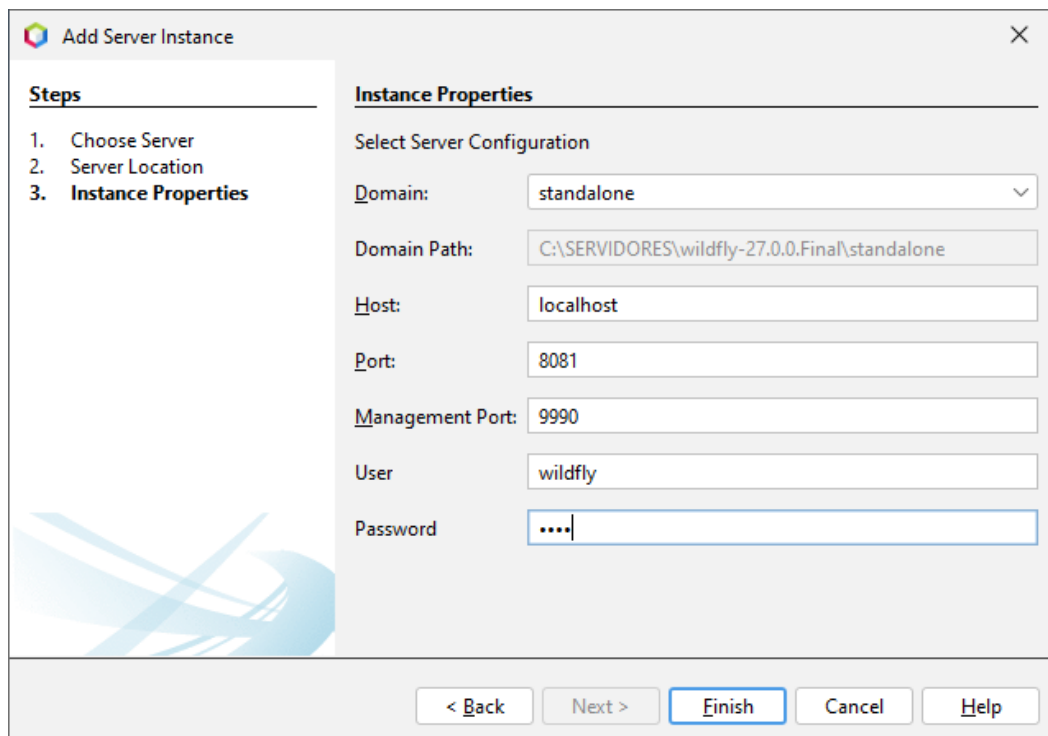
Debemos acceder a **Services** y añadirlo igual que el de Apache Tomcat con el nombre de **WildFly Application Server (local)**:



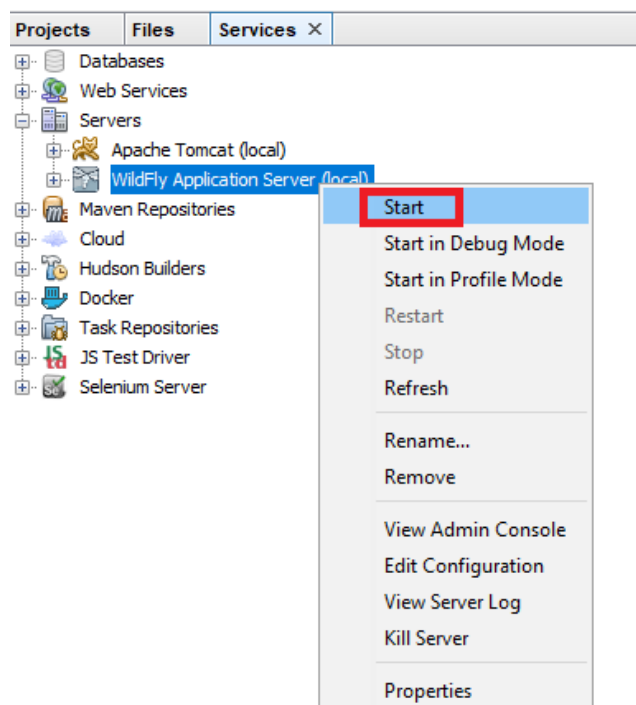
Luego indicaremos la carpeta de instalación de Wildfly y el fichero de configuración que por defecto es **standalone-full.xml**:



Y por último indicaremos el usuario creado **wildfly** con contraseña **1234** y los puertos, que en nuestro caso serán **8081** y **9990**:



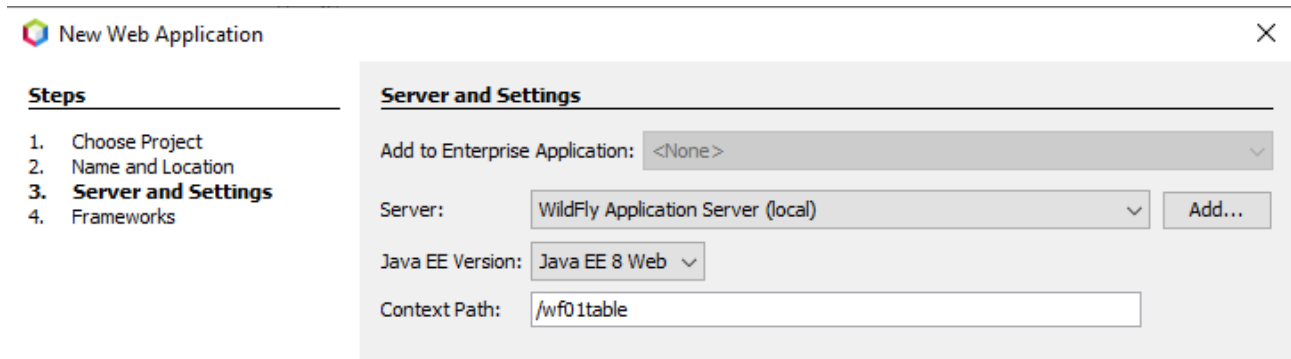
Al igual que con otros servidores, desde NetBeans podemos iniciar, reiniciar o parar el servidor utilizando el botón derecho.



## Crear proyecto de Java Web => Web Application

Vamos a crear el proyecto **WF01Table** que ejecute el código de **ejemplo\_table.jsp**.

Crearemos un proyecto de tipo "**Java Web => Web Application**" poniendo como path raíz del App Web (Context Path Root) **wf01table** y como servidor nuestro **Wildfly Application (local)**.



Ahora solo queda copiar el contenido de nuestro **ejemplo\_table.jsp** en el archivo **index.jsp** como en el ejemplo de Apache Tomcat.

Cuando ejecutemos el proyecto, si Wildfly está parado, se iniciará (Start). Hay que tener en cuenta que al cerrar NetBeans se parará (Stop).

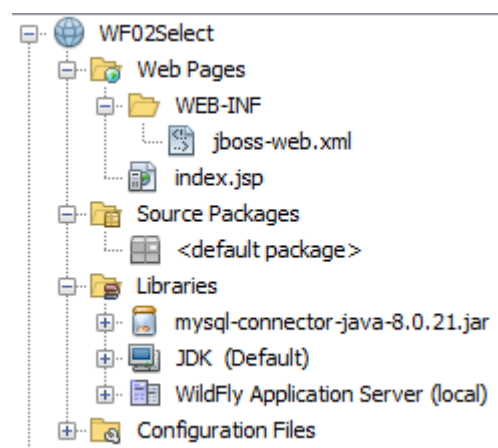
## Crear proyecto de Java Web => Web Application

Vamos a crear el proyecto **WF02Select** que ejecute el código de **ejemplo\_select.jsp**.

Realizaremos el mismo proceso que en el anterior proyecto, pero poniendo como path raíz del App Web (Context Path Root) **wf02select** y copiando el contenido de nuestro **ejemplo\_select.jsp** en el archivo **index.jsp**.

Si lo ejecutamos obtendremos un error porque no existe la librería del JDBC de mysql en el servidor de wildfly.

Para poder enviar las librerías junto a nuestras páginas JSP, copiaremos la carpeta **lib** como en los proyectos de tipo "**Java Application**" y lo añadiremos a **Libraries**:

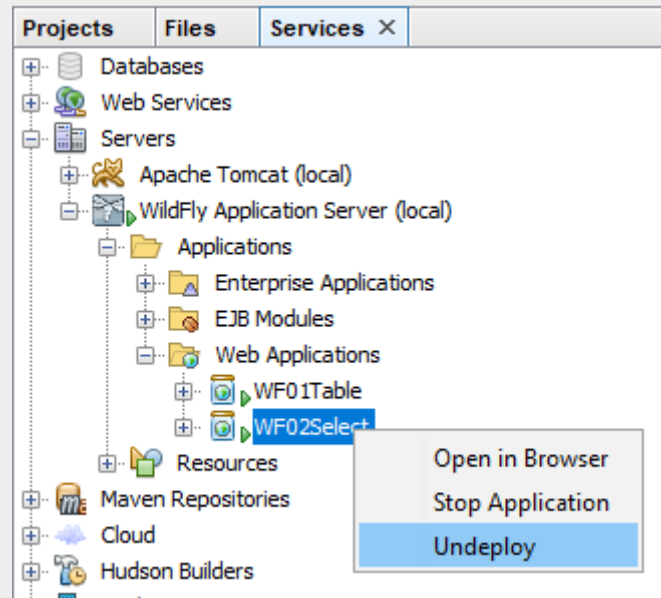


**Nota**

Es recomendable antes de ejecutar pulsar el botón **Clean and Build Project**



Si de todas formas no se puede desplegar porque ya está en ejecución, ir al servidor y realizar un **Undeploy**.

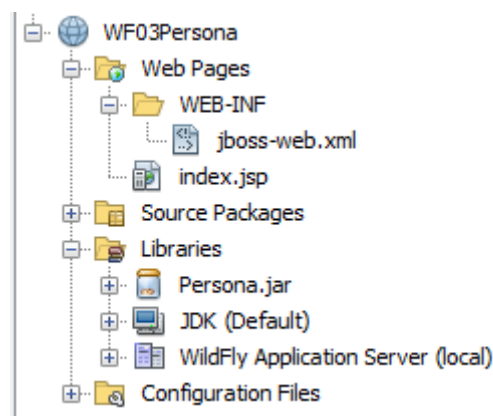


### **Crear proyecto de Java Web => Web Application**

Vamos a crear el proyecto **WF03Persona** que use una librería nuestra (componente JAR) con el context Path a **/wf03persona**

Lo primero es añadir a nuestro proyecto la librería **Persona.jar** como en la aplicación anterior.

Luego copiaremos el **index.jsp** con el mismo contenido publicado en Apache Tomcat.



## 8.3 Compilación y despliegue de Aplicaciones Web

Los archivos empaquetados obtenidos en las compilaciones de nuestro proyectos Java crean archivos JAR.

Un archivo **JAR (Java ARchive)** es un tipo de archivo que permite ejecutar aplicaciones y herramientas escritas en el lenguaje Java. Las siglas están deliberadamente escogidas para que coincidan con la palabra inglesa "jar" (tarro).

Los JAR están comprimidos con el formato ZIP y cambiada su extensión a .jar.

En el caso de aplicaciones Java Web el empaquetado obtenido es WAR.

Un archivo **WAR (de Web Application Archive - Archivo de aplicación web)** es un archivo JAR utilizado para distribuir una colección de JavaServer Pages, servlets, clases Java, archivos XML, bibliotecas de tags y páginas web estáticas (HTML y archivos relacionados) que juntos constituyen una aplicación web.

Los archivos **WAR** se ejecutan en Servidores Web, como ya hemos visto con Apache Tomcat o Wildfly.

Para instalar un archivo **WAR** en un Servidor Web se realiza un **despliegue** de la aplicación web. Para ello, los servidores web ofrecen varias formas de hacerlo:

- Desde entornos IDE, como Netbeans.
- Desde una web de administración
- Desde el explorador de archivos copiando el archivo WAR en una carpeta concreta.

En inglés, **Deploy** y **Undeploy** serán las acciones para instalar y desinstalar la aplicación web.



## APACHE TOMCAT

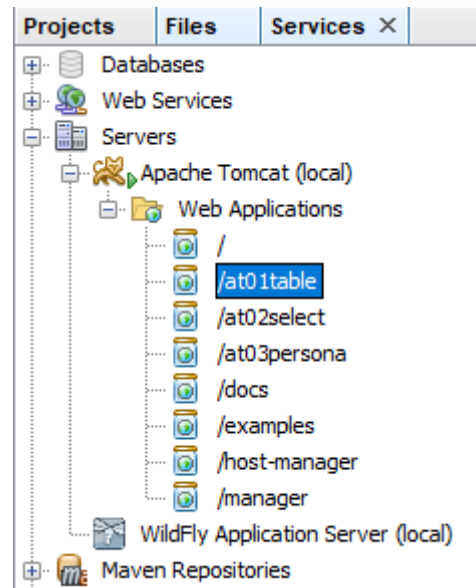
Trabajaremos con el proyecto **AT01Table**

### Despliegue automático desde NetBeans

Cuando ejecutamos un Java Web en NetBeans, la aplicación se despliega automáticamente en el Servidor asociado.

Una vez ejecutada y desplegada, podemos ver su instalación en la pestaña de **"Services => Servers"**.

Con botón derecho sobre la aplicación podemos desinstalarla con **Undeploy**.



Este despliegue no es definitivo porque se instala en la carpeta **work** del servidor.

### Obtener empaquetado (Archivo WAR)

Para distribuir nuestra aplicación, necesitaremos el archivo WAR.

Para obtener el archivo empaquetado tendremos que generar el proyecto con el **icono del martillo o F11** igual que para obtener un JAR en proyectos de tipo Java Application.

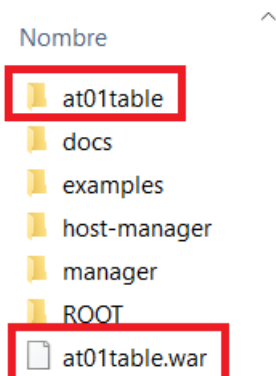
El fichero **AT01Table.war** estará en la carpeta **dist** de nuestro proyecto. Debemos renombrar con minúsculas el empaquetado para que lo use como Context Path, es decir, **at01table.war**.

### Despliegue manual de WAR en Apache Tomcat

Para desplegar manualmente nuestro WAR en un servidor Apache Tomcat lo copiaremos en la carpeta **webapps**, y automáticamente se desplegará y creando la carpeta de la aplicación web con los JSP que hayamos creado desde NetBeans.

Podemos probar su funcionamiento navegando sobre la carpeta del proyecto:

<http://localhost:8080/at01table>



## Despliegue de WAR desde Manager APP de Apache Tomcat

Para desplegar nuestro WAR utilizando el Manager App accederemos a:

<http://localhost:8080/manager/html>

y nos pedirá el usuario de **tomcat** que creamos en la instalación con la contraseña **1234**.

Desde esta consola podremos **instalar** (Desplegar/Deploy) aplicaciones **WAR**:

**Archivo WAR a desplegar**

Seleccione archivo WAR a cargar  AT01Table.war

También podríamos subirlo por ftp/sftp al servidor y desplegarlo, pudiendo cambiar su trayectoria de contexto (Context Path) por defecto:

**Desplegar**

Desplegar directorio o archivo WAR localizado en servidor

Trayectoria de Contexto (opcional):   
 Version (for parallel deployment):   
 URL de archivo de Configuración XML:   
 URL de WAR o Directorio:

y **desinstalar** (Replegar/Undeploy):

Aplicaciones					
Ruta	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado	Welcome to Tomcat	true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/>
/at01table	Ninguno especificado		true	0	<input type="button" value="Arrancar"/> <input type="button" value="Parar"/> <input type="button" value="Recargar"/> <input type="button" value="Replegar"/>
					<input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos
					<input type="button" value="Expirar sesiones"/> sin trabajar ≥ 30 minutos

## WILDFLY

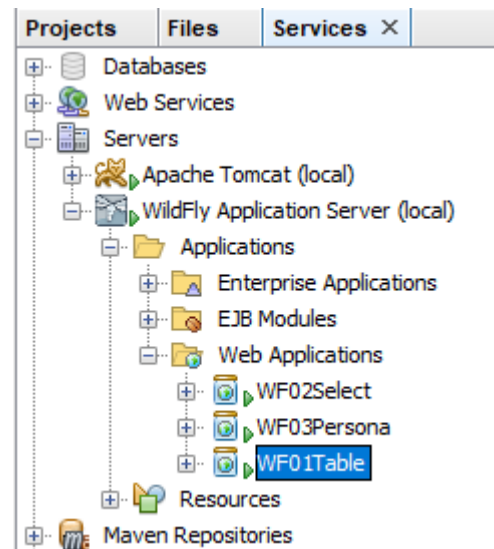
Trabajaremos con el proyecto **WF01Table**, siendo el proceso muy parecido al de Apache Tomcat.

### Despliegue automático desde NetBeans

Cuando ejecutamos un Java Web en NetBeans, la aplicación se despliega automáticamente en el Servidor asociado.

Una vez ejecutada y desplegada, podemos ver su instalación en la pestaña de **"Services => Servers"**.

Con botón derecho sobre la aplicación podemos desinstalarla con **Undeploy**.



### Obtener empaquetado (Archivo WAR)

Para obtener el archivo empaquetado **WAR** tendremos que generar el proyecto con el **icono del martillo o F11** igual que para obtener un JAR en proyectos de tipo Java Application.

### Despliegue manual de WAR en Wildfly

Para desplegar manualmente nuestro WAR en un servidor Wildfly lo copiaremos en la carpeta:

**standalone\deployments**

y automáticamente se desplegará y creará la carpeta de la aplicación web con los JSP que hayamos creado desde NetBeans.

Si ya existe, habrá que realizar **Undeploy** desde NetBeans y luego eliminar en **standalone\deployments\** la carpeta **WF01Table.war** y el archivo **WF01Table.war.undeployed**

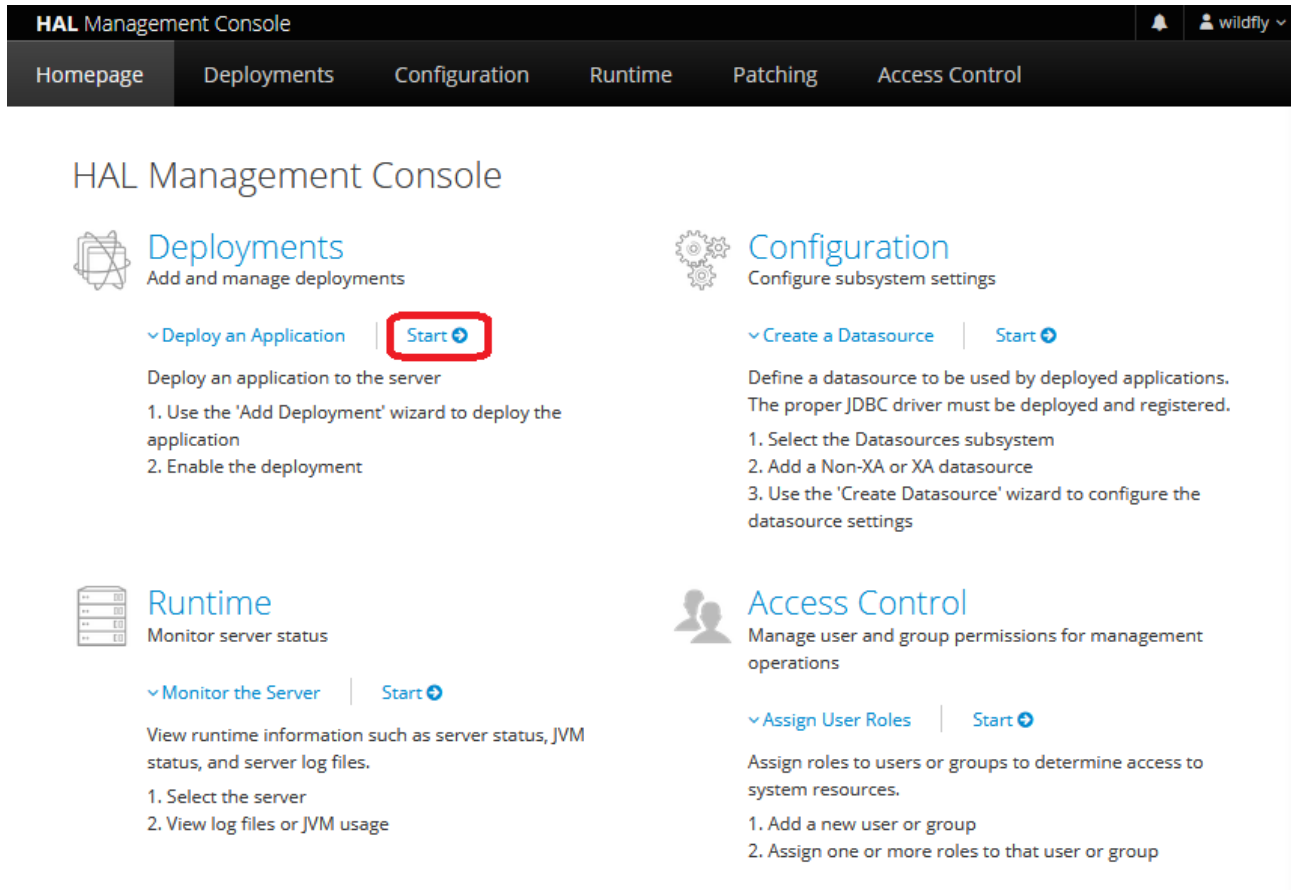
Podemos probar su funcionamiento navegando sobre la carpeta del proyecto:

<http://localhost:8081/wf01table/>

## Despliegue de WAR desde la consola de Wildfly

Para desplegar nuestro WAR utilizando la consola de Wildfly accederemos a:


<http://localhost:9990>



HAL Management Console

Deployments Configuration Runtime Patching Access Control

### HAL Management Console




#### Deployments

Add and manage deployments

▼ Deploy an Application | **Start**

Deploy an application to the server

1. Use the 'Add Deployment' wizard to deploy the application
2. Enable the deployment




#### Configuration

Configure subsystem settings

▼ Create a Datasource | **Start**

Define a datasource to be used by deployed applications. The proper JDBC driver must be deployed and registered.

1. Select the Datasources subsystem
2. Add a Non-XA or XA datasource
3. Use the 'Create Datasource' wizard to configure the datasource settings




#### Runtime

Monitor server status

▼ Monitor the Server | **Start**

View runtime information such as server status, JVM status, and server log files.

1. Select the server
2. View log files or JVM usage



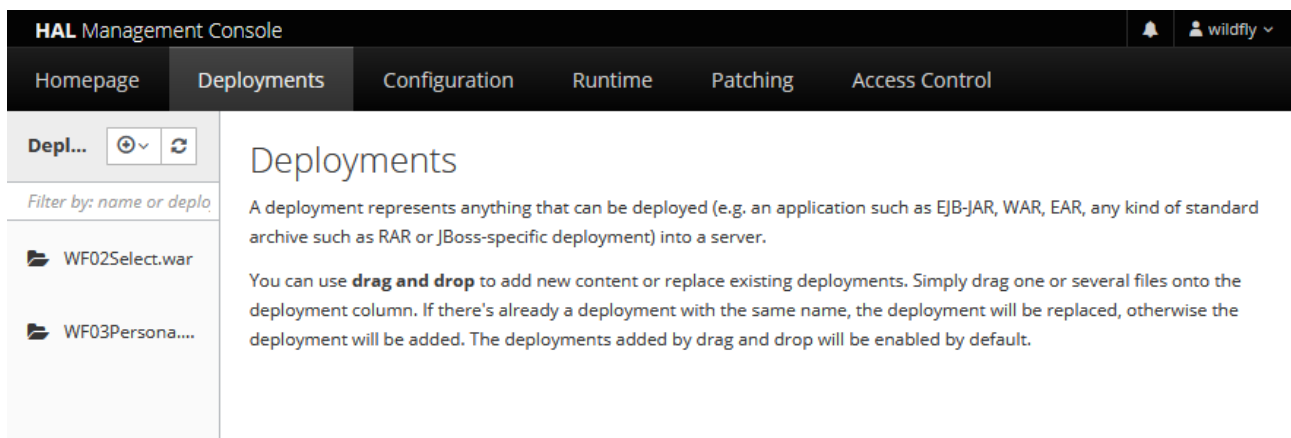
#### Access Control

Manage user and group permissions for management operations

▼ Assign User Roles | **Start**

Assign roles to users or groups to determine access to system resources.

1. Add a new user or group
2. Assign one or more roles to that user or group



HAL Management Console

Homepage Deployments Configuration Runtime Patching Access Control

Depl... [Filter] [Refresh]

Filter by: name or deployment

- WF02Select.war
- WF03Persona....

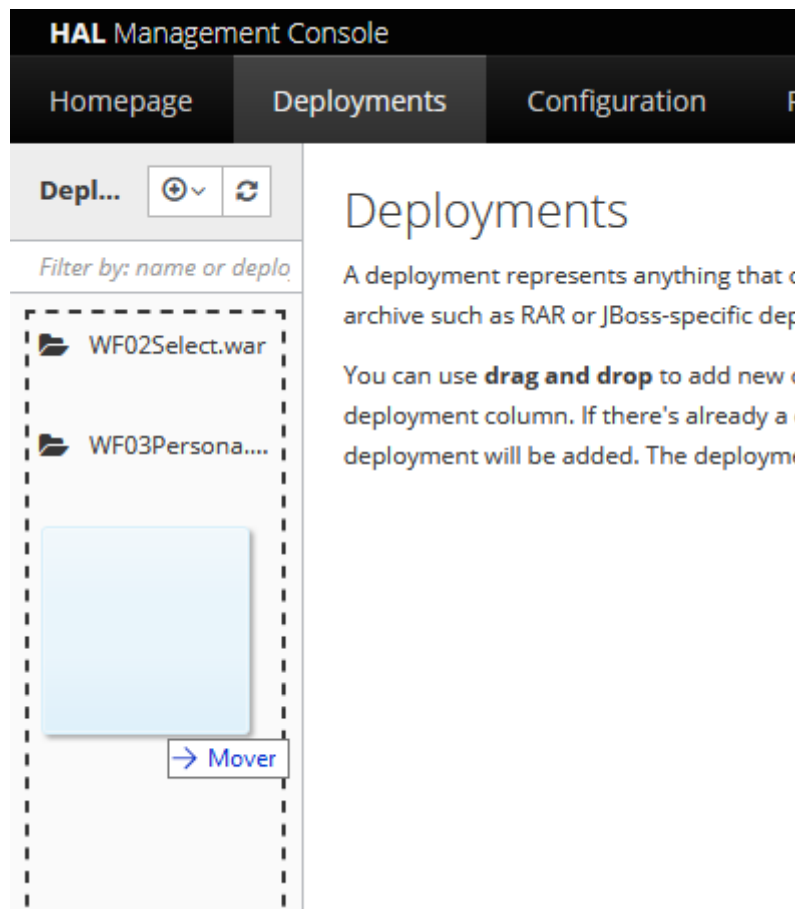
### Deployments

A deployment represents anything that can be deployed (e.g. an application such as EJB-JAR, WAR, EAR, any kind of standard archive such as RAR or JBoss-specific deployment) into a server.

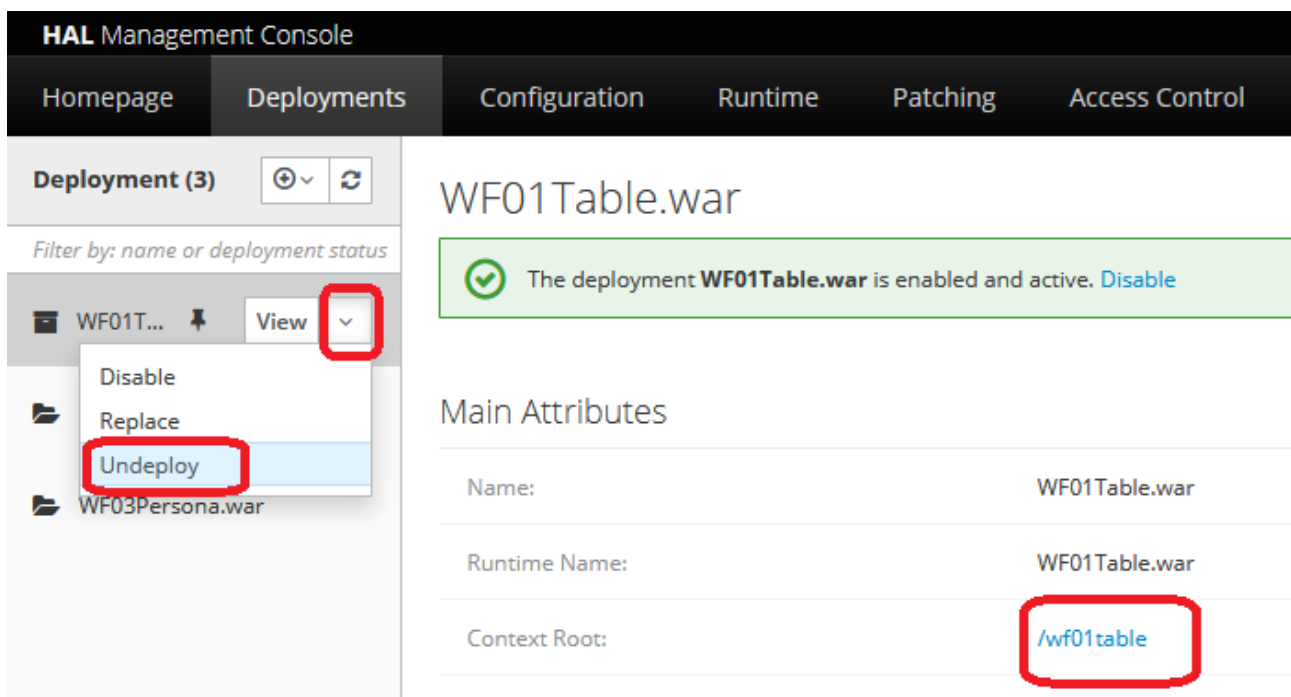
You can use **drag and drop** to add new content or replace existing deployments. Simply drag one or several files onto the deployment column. If there's already a deployment with the same name, the deployment will be replaced, otherwise the deployment will be added. The deployments added by drag and drop will be enabled by default.

Ahora podemos ver las aplicaciones que tenemos **desplegadas**.

Para desplegar basta con arrastrar nuestro archivo war al navegador en la zona donde se encuentran nuestras aplicaciones, en la barra izquierda:.



**Pulsando sobre ella (click)** nos aparecerá el enlace para probarla y un desplegable para poderla desinstalar (undeploy):



## **Componentes del DEPLOY en APACHE TOMCAT Y WILDFLY**

Como hemos podido comprobar, para utilizar componentes en aplicaciones de tipo Java Web es necesario que el Servidor Web cargue en memoria el componente. Para ello tenemos dos opciones:

1. Que el componente se encuentre en la carpeta de componentes a cargar por el servidor en el inicio y que podrán utilizar todas las App Web
2. Que el componente se envíe con la App Web

### **Componentes del Servidor**

En el primer caso necesitaremos copiar el componente en la carpeta correspondiente del servidor web como hemos hecho en los ejemplos anteriores:

- **En Apache Tomcat:** tomcat\lib
- **En Wildfly:** modules\system\layers\base

En Apache Tomcat se envía a NetBeans la información de todos los JAR cargados por el servidor y nos permite editar el código indicando los errores y completando métodos. En Wildfly no ocurre igual, por lo que seguramente no dispondremos de información en NeBeans.

Esta opción no suele usarse porque generalmente no tendremos acceso a modificar la configuración del servidor de forma habitual ya que necesitaríamos detenerlo y volverlo a iniciar.

### **Componentes en la App Web**

Añadiremos el archivo JAR como si fuera una aplicación de tipo Java Application, es decir, creando la carpeta lib con los archivos JAR y añadiéndolos en Libraries.

Al compilar el archivo WAR de la aplicación, los componentes serán copiados al servidor y cargados en su memoria. La carpeta donde se copian es: **WEB-INF\lib**

Para comprobarlo, basta con cambiar la extensión del WAR a ZIP y descomprimir su contenido como hace el servidor.

Si copiamos directamente nosotros la carpeta lib en Web Pages\WEB-INF\lib obtendremos el mismo resultado pero NetBeans mostrará errores en el código porque no detectará las clases que haya en los componentes.

### **Componentes en la App Web desplegada en el Servidor Web**

Las Web App con sus componentes en la carpeta **WEB-INF\lib** se encuentran instaladas en la carpeta:

- **En Apache Tomcat:** webapps
- **En Wildfly:** standalone\deployments