

Aplicaciones Distribuidas (AD)

Práctica 1: Desarrollo de una aplicación web utilizando páginas html, jsp y servlets que conectan con una base de datos

En esta práctica se va a desarrollar una aplicación web utilizando *Apache Netbeans* como entorno de desarrollo. El objetivo es realizar una aplicación web simple basada en J2EE (Java Enterprise Edition) que conecte con la base de datos JavaDB, que utilizaremos en las siguientes prácticas. Esta práctica tiene dos entregas.

En la primera entrega tenéis que indicar quiénes sois los componentes del grupo de prácticas (están pensadas para realizarlas en parejas) y responder a una serie de preguntas planteadas en el enunciado. El formato del fichero a entregar puede ser texto, un documento Word o un fichero pdf. No es necesario entregar código.

En la segunda entrega deberéis entregar la aplicación web desarrollada en un archivo zip.

Consultad las fechas de cada entrega en el Racó y leed este documento hasta el final antes de empezar a realizar la práctica.

Primera Entrega: Creación de una aplicación web J2EE

Creación

Para desarrollar la aplicación, se deberá crear un nuevo proyecto de tipo Web Application (File → New Project → Java with Maven → Web application). Cuando pregunte por el servidor, elegid **GlassFish Server**, que es el que nos ofrece todas las funcionalidades que necesitaremos en la asignatura (ver Figura 1).

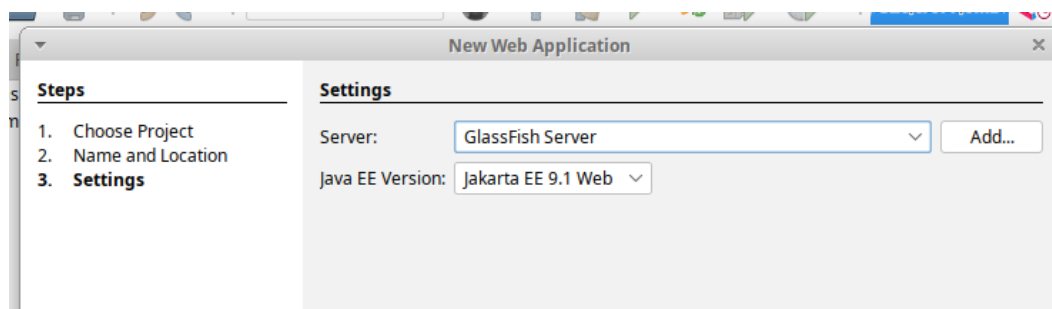


Figura 1. Seleccionar servidor

El nuevo proyecto contiene un fichero de tipo .html (index.html) en el directorio raíz. Eliminad ese fichero y cread uno de tipo .jsp (index.jsp) (New → JSP, ver Figura 2).

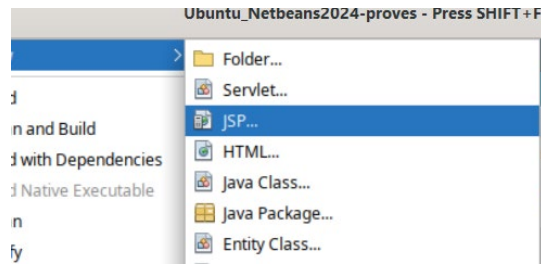


Figura 2. Crear una nueva JSP

En este ejercicio se pide:

- Comprobar que la aplicación se ha creado correctamente, poniéndola en marcha. Se tiene que abrir una ventana de navegador que se conecta a la siguiente dirección (`http://localhost:<puerto>/<nombreDeTuWebApp>/`) y aparece un mensaje en la página. El puerto dependerá de cómo esté configurado el domain de GlassFish. Lo habitual es que el puerto por defecto sea el 8080.
- Analiza el código fuente que ha recibido el navegador y compáralo con el código de la página `index.jsp`. ¿Qué diferencia o diferencias has encontrado?

Modificación del código

Para añadir código en una página jsp es necesario añadir los siguientes símbolos:

```
<% /* Código */ %>
```

Dentro del código, se puede añadir código Java con el objetivo de mostrar información en la página web.

- Añade la siguiente línea de código en la página
`<% out.println("<h2>Primera Práctica - Aplicaciones web - Aplicaciones Distribuidas</h2>"); %>` y ejecuta de nuevo la aplicación. Analiza de nuevo la página html que recibe el navegador, ¿qué se ha añadido a la página? ¿Para qué sirve la etiqueta `<h2>`?
- Añade otras líneas de código java en la página y observa los cambios que se vayan produciendo en el código generado cuando visualices la página en el navegador.
- Para finalizar este apartado, busca la opción de menú de *Netbeans* que te permite ver el servlet que se ha generado y comprueba que el código Java del servlet genera la página html que se envía al navegador. ¿Qué instrucciones en lenguaje java muestran el código html no generado con `out.println` en la página jsp?

Problema conocido curso 2025/2026:

Las nuevas versiones de GlassFish Server utilizan el package **jakarta** y no **javax** como se hacía hasta ahora en los servlets. La solución es cambiar javax por jakarta y los errores de importación de clases desaparecen (ver Figuras 3 y 4).



```
4  L  */
5  package servlet;
6
7  import java.io.IOException;
8  import java.io.PrintWriter;
9  import javax.servlet.ServletException;
10 import javax.servlet.annotation.WebServlet;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14
15 /**
16  *
17  * @author alumne
18  */
19 @WebServlet(name = "testServlet", urlPatterns = {"/testServlet"})
20 public class testServlet extends HttpServlet {
21
```

Figura 3. Error de importación de clases a la hora de crear un nuevo servlet



```
4  L  */
5  package servlet;
6
7  import java.io.IOException;
8  import java.io.PrintWriter;
9  import jakarta.servlet.ServletException;
10 import jakarta.servlet.annotation.WebServlet;
11 import jakarta.servlet.http.HttpServlet;
12 import jakarta.servlet.http.HttpServletRequest;
13 import jakarta.servlet.http.HttpServletResponse;
14
15 /**
16  *
17  * @author alumne
18  */
19 @WebServlet(name = "testServlet", urlPatterns = {"/testServlet"})
20 public class testServlet extends HttpServlet {
21
```

Figura 4. Cambio de package para resolver los errores

Nota: Para instalar el entorno en vuestras máquinas personales tenéis que utilizar la versión Apache Netbeans IDE 17 (URL descarga: <https://netbeans.apache.org/download/nb17/index.html>). También tenéis que instalar el JDK 17 de Oracle (URL descarga: <https://www.oracle.com/es/java/technologies/downloads/#java17>). Además, tenéis que instalar el servidor Glassfish Server 6.2.5 (URL descarga: <https://download.eclipse.org/ee4j/glassfish/glassfish-6.2.5.zip>). Esta combinación de versiones es la que hay (y funciona) en la máquina virtual de laboratorio. Se pueden utilizar otras versiones, pero tened en cuenta que hay muchas dependencias y que algunas combinaciones pueden fallar.

Segunda Entrega: Conexión de la aplicación web con base de datos Java DB

El Anexo 2 describe la creación de una base de datos utilizando Java DB, que es una base de datos incluida en *Apache Netbeans*. Tenéis disponible en el Racó un fichero llamado `testJavaDB.java`, donde encontraréis ejemplos de conexión y acceso a la base de datos creada. Prueba de incorporarlo al proyecto (hay que copiarlo en la carpeta correcta). Si no funciona, crea un servlet nuevo y copia las instrucciones Java de acceso y gestión de la base de datos que necesites.

Realiza las siguientes operaciones sobre el servlet:

- a) En el método `processRequest`, utiliza el código de prueba de `testJavaDB.java` para crear la base de datos y sus tablas. Inserta algunos datos en las tablas creadas. En los parámetros de conexión tendréis que poner los que correspondan con vuestra base de datos (nombre, usuario y password).
- b) Crea un fichero html. En este fichero, añade un formulario html que llame al servlet anterior. Este paso es necesario para crear la base de datos y las tablas. En el Anexo 1 hay una breve explicación de cómo se crea un formulario y los parámetros que necesita. En el atributo `action` del formulario tenéis que poner la URL del servlet que será parecido a `<nombreServlet>` (sin nombre de servidor ni aplicación web, puesto que nos estamos conectando a la misma aplicación web que lo contiene), donde `nombreServlet` es el valor que se haya definido en la Anotación Java `@WebServlet(name = "nombreServlet", urlPatterns = {"/nombreServlet"})`, dentro del código del servlet. Si hubiera que conectar el formulario con otra aplicación web, entonces en el atributo `action` deberíais poner algo parecido a `http://localhost:<puerto>/<nombreDeTuWebApp>/<nombreServlet>`.
- c) Pon en marcha la aplicación y comprueba que se ha creado la base de datos. A continuación, realiza las siguientes pruebas, modificando el código del servlet o creando servlets nuevos: Añade nuevos datos en la base de datos, consulta los datos ya existentes añadiendo condiciones en la sentencia SQL y crea un formulario html para pasar los datos de imagen al usuario. Puedes crear tantos servlets, ficheros html o Java Server Pages (jsp) como necesites, utilizando el código de `testJavaDB.java` como base. **Nota:** Si la página no devuelve nada, revisa la ventana Output de Netbeans. Si los errores que aparecen no tienen sentido, utiliza el comando `Clean and build` para reconstruir y reinstalar la aplicación web en el servidor. Después debes volver a ejecutarla (botón verde Play o comando Run).

Anexo 1: Cómo enlazar servlets, jsp's y ficheros html

Para relacionar los ficheros html o jsp con los servlets, deberemos añadir el siguiente código html a nuestro formulario:

```
<form action = "login" method = "POST">  
    <!-- Añadir los campos del formulario que creas conveniente -->  
</form>
```

En este formulario estamos indicando que los datos que introduzca el usuario tenemos que enviarlos a un servlet identificado con la URL `login` y que utilizará el método POST de HTTP para el envío de los datos. La URL puede ser *case sensitive*.

Busca como recoger los datos del formulario desde el servlet (pista, es el objeto request). Estos datos se almacenan en un objeto Java que modela la petición HTTP.

Anexo 2: Soporte a la base de datos JavaDB

Este anexo describe cómo utilizar la JavaDB, una base de datos integrada con *Netbeans* y *GlassFish*. Sigue los siguientes pasos para poder acceder a una base de datos Java DB desde tu aplicación web.

Para iniciar la JavaDB, ve al panel Servicios y selecciona Start Server (dependiendo del idioma de *Netbeans*, el comando tendrá otro nombre). En la Figura 5, el servidor ya estaba iniciado, por eso no se puede seleccionar el comando. Después, selecciona Create Database y rellena los parámetros para crear la base de datos tal y como se muestra en la Figura 6. A continuación, conéctate con la base de datos como se puede ver en la Figura 7. La conexión `jdbc:derby://localhost:1527/pr2` se ha activado. Ábrela y ve a la carpeta Tables. Selecciona la opción Execute Command (Figura 8). En el nuevo fichero que aparecerá (Figura 9), copia los contenidos del fichero `database.sql` que encontraréis en el Racó.

El nombre de la base de datos, usuario y password es `pr2`. Podéis modificarlo, pero recordad lo que habéis puesto porque lo utilizaremos en las siguientes prácticas.

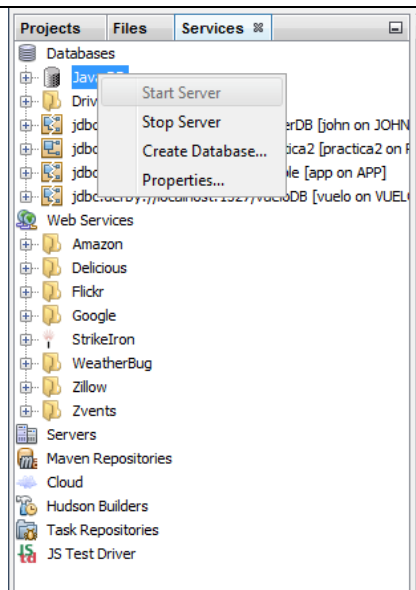


Figura 5. Iniciar base de datos

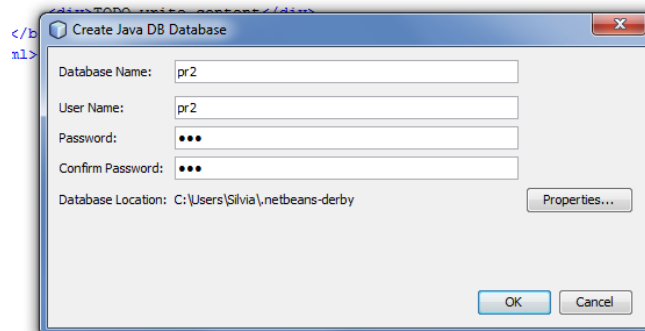


Figura 6. Crear base de datos

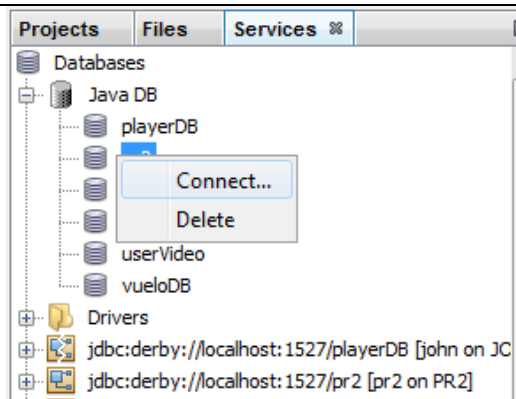


Figura 7. Conectar a la base de datos

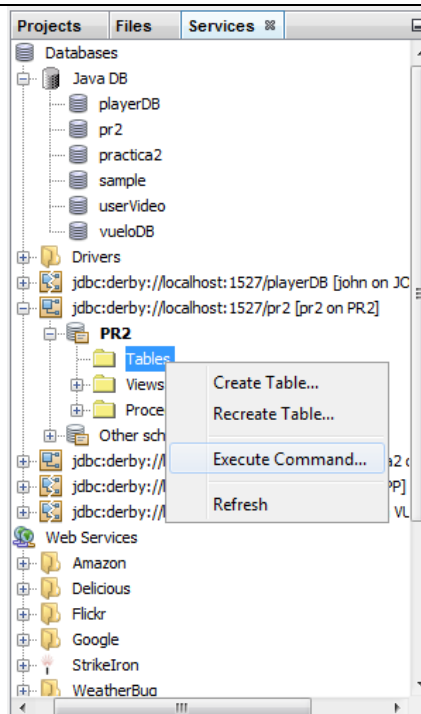
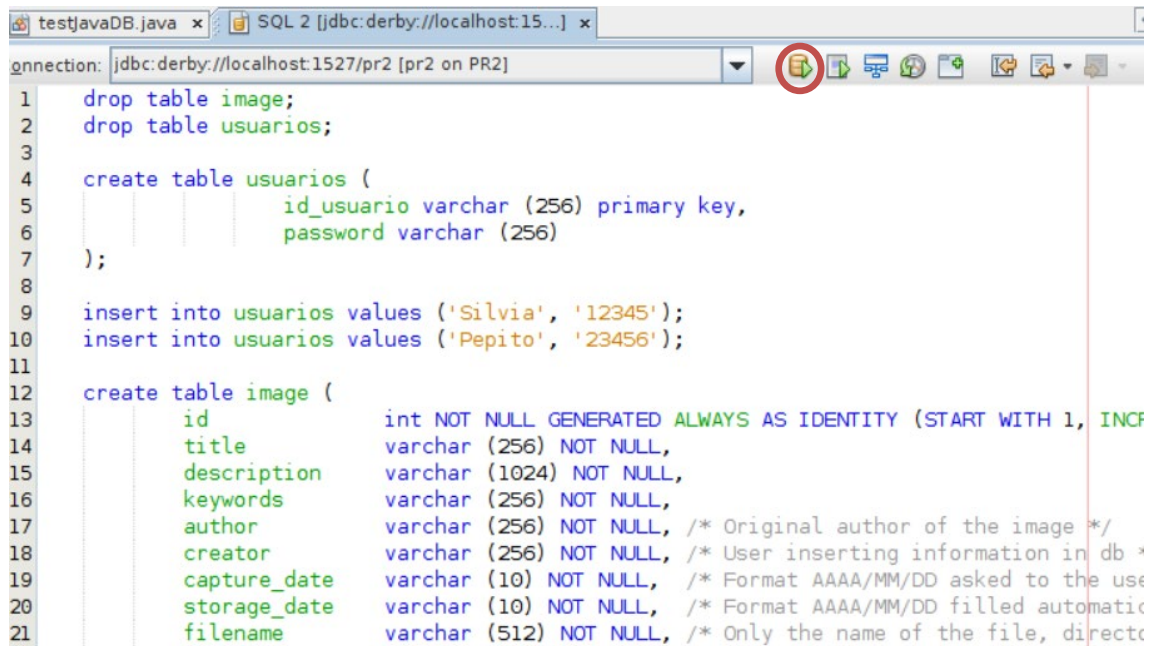


Figura 8. Ejecuta el comando SQL para crear la base de datos



```
1 drop table image;
2 drop table usuarios;
3
4 create table usuarios (
5     id_usuario varchar (256) primary key,
6     password varchar (256)
7 );
8
9 insert into usuarios values ('Silvia', '12345');
10 insert into usuarios values ('Pepito', '23456');
11
12 create table image (
13     id int NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCF
14     title varchar (256) NOT NULL,
15     description varchar (1024) NOT NULL,
16     keywords varchar (256) NOT NULL,
17     author varchar (256) NOT NULL, /* Original author of the image */
18     creator varchar (256) NOT NULL, /* User inserting information in db */
19     capture_date varchar (10) NOT NULL, /* Format AAAA/MM/DD asked to the use
20     storage_date varchar (10) NOT NULL, /* Format AAAA/MM/DD filled automatic
21     filename varchar (512) NOT NULL, /* Only the name of the file, directo
```

Figura 9. Fichero SQL ejemplo. Ver database.sql en el Racó

La base de datos se ha creado. Ahora para poder utilizarla desde vuestros servlets y jsp's, tenéis que utilizar el código que se muestra en la Figura 10 (que encontraréis en el fichero testJavaDB.java):

```
String query;
PreparedStatement statement;

Class.forName("org.apache.derby.jdbc.ClientDriver");

// create a database connection
connection = DriverManager.getConnection("jdbc:derby://localhost:1527/pr2;user=pr2;p
```

Figura 10. Cargar driver y conectar con la base de datos

Con estas instrucciones se cargará el driver de la JavaDB y os podréis conectar a la base de datos. Ahora, ya podéis utilizar las instrucciones executeQuery y executeUpdate del fichero testJavaDB.java. No es necesario crear y rellenar de nuevo la base de datos, leed los comentarios dentro de testJavaDB.java y realizad las modificaciones necesarias.

Nota: El fichero testJavaDB.java ya tiene la importación de clases correcta con el package jakarta.

Nota 2: Si ya hay una base de datos creada en la máquina virtual con el nombre pr2, borrad las tablas que hay y creadlas de nuevo porque no son las que necesitáis.