

## 7.5. *Servlet con acceso a base de datos con JDBC*

En esta práctica desarrollaremos una aplicación basada en un *Servlet* que accederá a una base de datos *MySQL* mediante un conector *JDBC*.

### 1. Instalación de MySQL.

- 1.1. Inicia sesión en el equipo **DesarrolloW7XX**.
- 1.2. Para simplificar la instalación descarga el instalador para *Windows* de *MySQL* de la siguiente dirección <http://dev.mysql.com/downloads/installer/>.
- 1.3. También necesitarás el instalador independiente de *Microsoft .NET Framework 4*, que puedes descargar de la dirección:  
<http://www.microsoft.com/en-us/download/details.aspx?id=17718>.
- 1.4. Ejecuta el instalador de *Microsoft .NET Framework 4* y reinicia el equipo.
- 1.5. A continuación ejecuta el instalador de *MySQL*, eligiendo las siguientes opciones:
  - Tipo de instalación: *Developer Default*.
  - *Instalation Path: C:\MySQL\*.
  - *MySQL Server Configuration Type: Development Machine*.
  - *Enable TCP/IP Networking: Port Number 3306*.
  - *MySQL Root Password: despliegue*.
  - *Create Windows Service, Windows Service Name: MySQL56*.
- 1.6. Una vez concluida la instalación comprueba en **Panel de control, Servicios**, que el servicio *MySQL56* está iniciado.

reación de la base de datos TiendaLibros.

1. Inicia sesión en MySQL con el comando:

```
C:\MySQL\bin> mysql -u root -p
```

e introduce el *password* que establecimos en la instalación. También puedes iniciar el **MySQL 5.6 Command Line Client**.

2. Para crear la base de datos **TiendaLibros** y la tabla **libros**, crea el siguiente *script* en el fichero C:\MySQL\TiendaLibros.sql:

```
create database TiendaLibros;
```

```
use TiendaLibros;
```

```
create table libros (
    id      int,
    titulo  varchar(50),
    autor   varchar(50),
    precio  float,
    cantidad int,
    primary key (id));
```

```
insert into libros values
(1001, 'Servicios de Red e Internet', 'Alvaro Garcia', 25, 100);
insert into libros values
(1002, 'Apache Tomcat 7', 'Aleksa Vukotic', 22.22, 22);
insert into libros values
(1003, 'Beginning JSP, JSF, and Tomcat Web Development', 'Giulio Zambon', 33.33, 33);
insert into libros values
(1004, 'Tomcat, the definitive guide', 'Jason Brittain', 55.55, 55);
insert into libros values
(1005, 'Profesional Apache Tomcat', 'Vivek Chopra', 66.66, 66);
insert into libros values
(1006, 'Despliegue de aplicaciones Web', 'Alvaro Garcia', 22, 100);
```

```
select * from libros;
```

3. Ejecuta el *script* con el comando:

```
source C:\MySQL\TiendaLibros.sql
```

scarga del conector JDBC.

BC es un *API* de Java para ejecutar sentencias *SQL*. Está formado por un conjunto de clases e interfaces programadas con el propio Java. Permite interactuar con bases de datos de forma transparente al tipo de la misma. Es decir, es una forma única de programar el acceso a bases de datos desde Java, independiente del tipo de la base de datos.

4. En nuestro caso vamos a utilizar un *driver* nativo JDBC que permite la conectividad entre Java y MySQL. Puedes descargar este conector desde la propia página de MySQL: <http://www.mysql.com/products/connector/>, Figura 7.40.

## MySQL Connectors

MySQL provides standards-based drivers for JDBC, ODBC, and .Net enabling developers to embed MySQL directly into their applications.

Developed by MySQL	
ADO.NET Driver for MySQL (Connector/NET)	<a href="#">Download</a>
ODBC Driver for MySQL (Connector/ODBC)	<a href="#">Download</a>
JDBC Driver for MySQL (Connector/J)	<a href="#">Download</a>
C++ Driver for MySQL (Connector/C++)	<a href="#">Download</a>
C Driver for MySQL (Connector/C)	<a href="#">Download</a>
C API for MySQL (mysqlclient)	<a href="#">Download</a>
MySQL Connector for OpenOffice.org	

Figura 7.40: *Driver JDBC para MySQL*

- 3.2. En concreto, descarga el archivo **.zip** con el conector. En el momento de hacer esta documentación era **mysql-connector-java-5.1.25.zip**. Una vez descargado descomprímelo y comprueba que está el archivo **mysql-connector-java-5.1.25-bin.jar** que contiene el conector.

### 4. Servlet de acceso a MySQL con JDBC.

- 4.1. Inicia *Eclipse* y crea un nuevo proyecto *Dynamic Web Project* llamado **AccesoDatos**. Recuerda marcar la opción para que se cree un descriptor de despliegue **web.xml**.
- 4.2. Para empezar a desarrollar la aplicación vamos a crear una página *HTML* que será la interfaz de entrada con la que interactuará el usuario. Añade una página **ConsultaLibros.html** a la carpeta *WebContent* de tu proyecto **AccesoDatos**. Su código será el siguiente:

```
<html>
<head>
    <title>TiendaLibros</title>
</head>
<body>
    <h2>Tienda Libros</h2>
    <form method="get" action="http://localhost:8080/AccesoDatos/consulta">
        <b>Elige un autor:</b>
        <input type="checkbox" name="autor" value="Alvaro Garcia">Alvaro Garcia
        <input type="checkbox" name="autor" value="Aleksa Vukotic">Aleksa Vukotic
        <input type="checkbox" name="autor" value="Giulio Zambon">Giulio Zambon
        <input type="submit" value="Buscar">
    </form>
</body>
</html>
```

- 4.3. Ejecuta el proyecto desde *Eclipse* para acceder a la página <http://localhost:8080/AccesoDatos/ConsultaLibros.html>. Para ello pulsa con el botón derecho del ratón sobre el fichero **ConsultaLibros.html**, elige la opción **Run as, Run on server**, Figuras 7.41, 7.42. Esto hará que se inicie el servidor *Tomcat* que tenemos registrado en *Eclipse* y que se ejecute la página seleccionada.

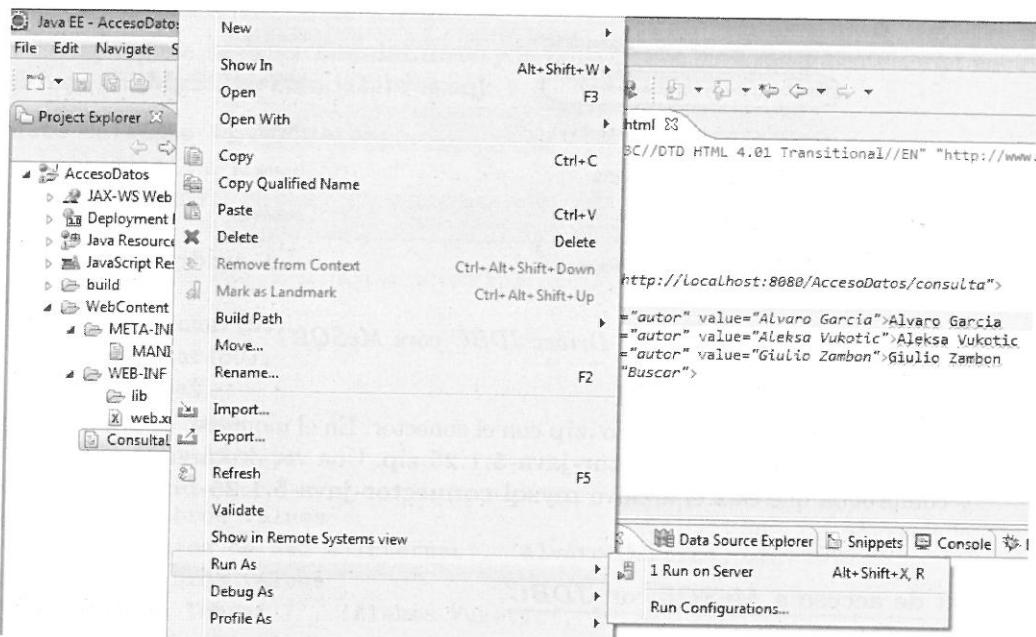


Figura 7.41: Ejecutar aplicación desde *Eclipse*

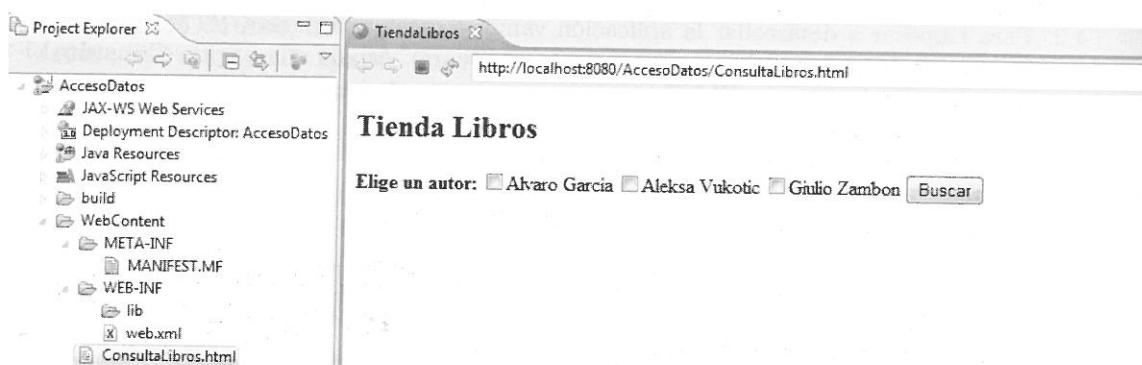


Figura 7.42: Página *HTML* de acceso a la aplicación

- 4.4. Si pulsamos el botón **Buscar** no funcionará, porque hemos establecido que al enviar la página se ejecute la *URL* `http://localhost:8080/AccesoDatos/consulta` que corresponderá al *Servlet* que todavía no hemos desarrollado.

```
<form method="get" action="http://localhost:8080/AccesoDatos/consulta">
```

- 4.5. Ahora vamos a desarrollar el *Servlet* de acceso a la base de datos. Lo primero que tenemos que realizar, para que el servidor *Tomcat* pueda ejecutar una aplicación utilizando el *driver JDBC*, es colocar el fichero `mysql-connector-java-5.1.20-bin.jar` en el directorio **CATALINA\_HOME\lib**.
- 4.6. Reinicia el servidor desde *Eclipse* para que sea posible utilizar el conector.
- 4.7. Las clases que conforman el *API JDBC* del conector se encuentran agrupadas en el paquete `java.sql`. Este paquete contiene clases para cargar los *drivers*, realizar las conexiones a las bases de datos, consultar los datos y manejar un conjunto de registros. También posee las clases para el manejo de excepciones que se produzcan en el acceso a bases de datos.
- 4.8. A continuación veremos los pasos que se realizan en una aplicación con acceso a datos mediante *JDBC*:

- 1 Cargar el *driver JDBC*.
- 2 Conectarse a la Base de Datos utilizando la clase *Connection*.
- 3 Crear sentencias *SQL*, utilizando objetos de tipo *Statement*.
- 4 Ejecutar las sentencias *SQL* a través de los objetos de tipo *Statement*.
- 5 En caso de que sea necesario, procesar el conjunto de registros resultante utilizando la clase *ResultSet*.

- 4.9. Ahora crearemos el *Servlet* que se encargará de hacer la consulta a la base de datos. Añade el fichero **ConsultaServlet.java** a la carpeta **Java Resources\src** del proyecto **AccesoDatos** e introduce el siguiente código **Java**.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ConsultaServlet extends HttpServlet {

    // El método doGet() se ejecuta una vez por cada petición HTTP GET.
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // Establecemos el tipo MIME del mensaje de respuesta
        response.setContentType("text/html");
        // Creamos un objeto para poder escribir la respuesta
        PrintWriter out = response.getWriter();

        Connection conn = null;
        Statement stmt = null;
        try {
```

```
//Paso 1: Cargar el driver JDBC.  
Class.forName("com.mysql.jdbc.Driver").newInstance();  
  
// Paso 2: Conectarse a la Base de Datos utilizando la clase Connection  
String userName="root";  
String password="despliegue";  
//URL de la base de datos(equipo, puerto, base de datos)  
String url="jdbc:mysql://localhost/TiendaLibros";  
conn = DriverManager.getConnection(url, userName, password);  
  
// Paso 3: Crear sentencias SQL, utilizando objetos de tipo Statement  
stmt = conn.createStatement();  
  
// Paso 4: Ejecutar las sentencias SQL a través de los objetos Statement  
String sqlStr = "select * from libros where autor = "  
+ "'" + request.getParameter("autor") + "'";  
  
// Generar una página HTML como resultado de la consulta  
out.println("<html><head><title>Resultado de la consulta</title></head><body>");  
out.println("<h3>Gracias por tu consulta.</h3>");  
out.println("<p>Tu consulta es: " + sqlStr + "</p>");  
ResultSet rset = stmt.executeQuery(sqlStr);  
  
// Paso 5: Procesar el conjunto de registros resultante utilizando ResultSet  
int count = 0;  
while(rset.next()) {  
    out.println("<p>" + rset.getString("autor")  
        + ", " + rset.getString("titulo")  
        + ", " + rset.getDouble("precio") + "</p>");  
    count++;  
}  
out.println("<p>===== " + count + " registros encontrados =====</p>");  
out.println("</body></html>");  
} catch (Exception ex) {  
    ex.printStackTrace();  
} finally {  
    out.close(); // Cerramos el flujo de escritura  
    try {  
        // Cerramos el resto de recursos  
        if (stmt != null) stmt.close();  
        if (conn != null) conn.close();  
    } catch (SQLException ex) {  
        ex.printStackTrace();  
    }  
}
```

- 4.10. Finalmente y para que este *Servlet* sea accesible edita el descriptor de despliegue añadiendo la siguiente información dentro de la etiqueta **web-app**:

```
...
<servlet>
    <servlet-name>ConsultaUsuario</servlet-name>
    <servlet-class>ConsultaServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>ConsultaUsuario</servlet-name>
    <url-pattern>/consulta</url-pattern>
</servlet-mapping>
</web-app>
```

- 4.11. Prueba a ejecutar el proyecto desde *Eclipse* para comprobar su correcto funcionamiento. Recuerda que la página `http://localhost:8080/AccesoDatos/ConsultaLibros.html` será el punto de entrada a la aplicación.

- 4.12. Marca la opción **Alvaro Garcia** y pulsa sobre el botón **Buscar**. Una petición *HTTP Get* es enviada a la *URL* especificada en el atributo *action* de la etiqueta *form*:

```
<form method="get" action="http://localhost:8080/AccesoDatos/consulta">
```

Observa la *URL* de la petición *HTTP GET*:

```
http://localhost:8080/AccesoDatos/consulta?autor=Alvaro+Garcia
```

La petición consiste en dos partes, una *URL* correspondiente al atributo *action* y separado por una "?" aparece la pareja "name=value" extraída de la etiqueta *input*. Observa cómo los blancos son reemplazados por "+" porque los espacios en blanco no están permitidos en las *URLs*.

- 4.13. Si ocurriera algún error en la ejecución realizada desde *Eclipse* se mostraría un mensaje en la consola, Figura 7.43.



Figura 7.43: La consola mostrará los errores de ejecución

- 4.14. Ten también presente que no es necesario volver a iniciar el servidor cada vez que realicemos un cambio en el *Servlet*. Cualquier cambio en el código es automáticamente reflejado en el servidor *Tomcat* registrado en *Eclipse*.

4.15. Despliegue de la aplicación de Acceso a Datos con JDBC.

Una vez que hemos comprobado en el entorno de desarrollo que la aplicación funciona, dentendremos el servidor *Tomcat* desde *Eclipse*, Figura 7.44.

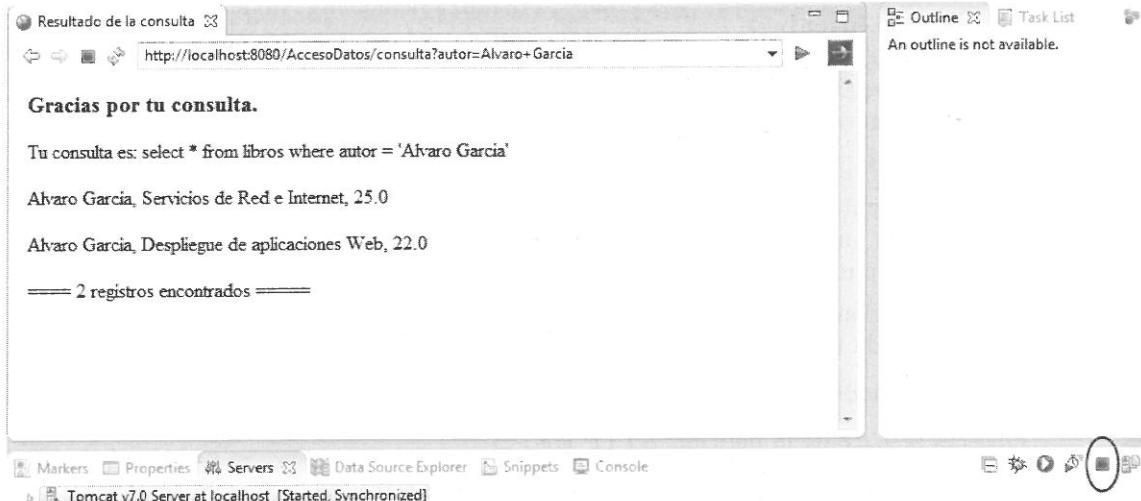


Figura 7.44: Para el servidor *Tomcat* desde *Eclipse*

- 4.16. Es interesante indicar que la ejecución del servidor *Tomcat* que realizamos desde *Eclipse* utiliza sus propios ficheros de configuración, Figura 7.45, diferentes de los ubicados **CATALINA\_HOME\conf**

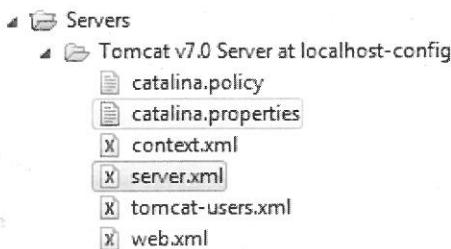


Figura 7.45: Ficheros de configuración de *Tomcat* en *Eclipse*

- 4.17. Exporta el proyecto **AccesoDatos** a un fichero **AccesoDatos.war**, marcando la opción sobreescribir si el fichero ya existe.
- 4.18. Inicia ahora el servidor *Tomcat*, ejecutando el *script* de arranque *startup*, situado en **CATALINA\_HOME\bin**
- 4.19. Para desplegar el proyecto en *Tomcat*, copia el fichero **AccesoDatos.war** en la carpeta **CATALINA\_HOME\webapps**.



## Gracias por tu consulta.

Tu consulta es: select \* from libros where autor = 'Alvaro Garcia'

Alvaro Garcia, Servicios de Red e Internet, 25.0

Alvaro Garcia, Despliegue de aplicaciones Web, 22.0

===== 2 registros encontrados =====

Figura 7.46: Funcionamiento de la aplicación una vez desplegada

4.20. Puedes comprobar cómo funciona desde el navegador, Figura 7.46

### 5. Peticiones POST.

HTTP define los métodos para realizar peticiones: *GET* y *POST*. En este ejemplo hemos usado el método *GET*, como especificamos en la página **ConsultaLibros.html**:

```
<form method="get" action="http://localhost:8080/AccesoDatos/consulta">
```

La petición *GET* es procesada por el método *doGet* del *Servlet ConsultaServlet*.

¿Cuál es la diferencia entre los dos métodos?. En las peticiones *POST* los parámetros se envían en el cuerpo de mensaje, por lo que no son visibles en la *URL* como pasa con las peticiones *GET*. Las ventajas radican en que:

- El cliente no ve una extraña cadena, *query string*, en la *URL*.
- Existe una limitación en la longitud de la *query string* de la petición *GET*, mientras que la petición *POST* carece de esa limitación.

En la práctica es habitual utilizar el mismo método para gestionar tanto las peticiones *GET* y *POST*. Vamos a aplicar esta idea a nuestro ejemplo para que el *servlet* sea capaz de atender tanto las peticiones *GET* como las *POST*.

### 6. Añade una nueva página llamada página **ConsultaLibrosPost.html** con el siguiente código:

```
<html>
<head>
    <title>TiendaLibros</title>
</head>
<body>
    <h2>Tienda Libros</h2>
    <form method="post" action="consulta">
```

```

<b>Elige un autor:</b>
<input type="checkbox" name="autor" value="Alvaro Garcia">Alvaro Garcia
<input type="checkbox" name="autor" value="Aleksa Vukotic">Aleksa Vukotic
<input type="checkbox" name="autor" value="Giulio Zambon">Giulio Zambon
<input type="submit" value="Buscar">
</form>
</body>
</html>

```

Observa cómo hemos cambiado el método por el cual se enviará la petición *method="post"*. También hemos cambiado la forma de indicar qué recurso será el destinatario de la acción. Ahora indicamos *action="consulta"*, sin indicar su ruta absoluta *http://localhost:8080/AccesoDatos/consulta*, como hicimos en **ConsultaLibros.html**.

7. Respecto al *servlet* que procesará la petición *POST* no es necesario escribir uno nuevo, simplemente invocaremos al método *doGet* desde el método *doPost* que tratará la petición *POST*. Añade el siguiente código al *servlet ConsultaServlet*:

```

public void doPost (HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
    doGet(request, response); // Redirecciona la petición POST al método doGet()
}

```

8. Observa cómo al utilizar la página *ConsultaLibrosPost.html* no aparecen los parámetros en la *URL*, Figura 7.47:

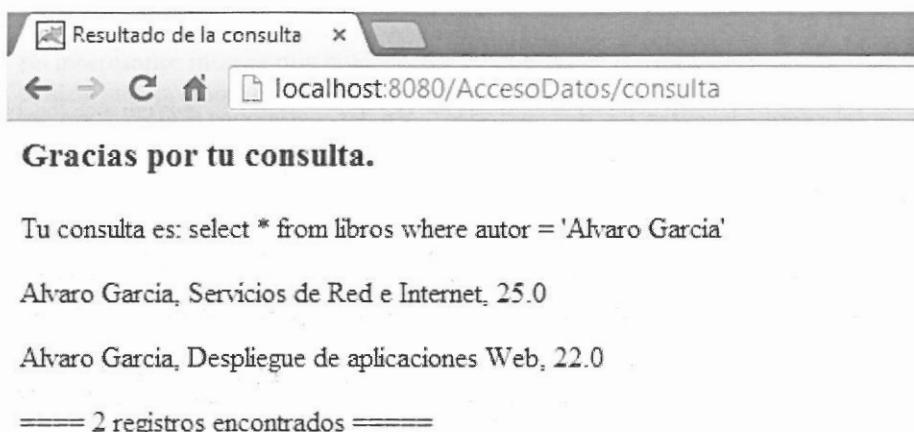


Figura 7.47: Funcionamiento de la petición *POST*