Práctica 4

PROGRAMACIÓN WEB CON SERVLETS Y JSP

Tabla de contenidos

1. Introducción a JSP

1.1. Que es Java Server Page (JSP)?

Es una interfaz de programación de aplicaciones de servidores Web. En una página jsp se entremezclan bloques de HTML estáticos, y HTML dinámico generados con Java que se ejecutan en el servidor.

Una página jsp puede procesar formularios Web, acceder a bases de datos y redireccionar a otras páginas. Las páginas jsp son transformadas a un servlet y después compiladas.

El contenedor JSP proporciona un motor que interpreta y procesa una página JSP como un servlet (en *tomcat*, dicho motor se llama *jasper*). Al estar basadas en los servlets, las distintas peticiones a una misma página jsp son atendidas por una única instancia del servlet.

1.2. Diferencias entre JSP y Servlet

- En JSP, el código de presentación está separado de la lógica del programa, mientras que en un servlet, el código de presentación se compila dentro de la clase.
- En una página JSP el código de presentación puede ser actualizado por un diseñador web que no conozca Java.
- Los servlets se encuentran ya compilados, mientras que las páginas JSP se compilan bajo petición, lo que hace que la ejecución del servlet sea algo más rápida (en la primera petición).

1.3. Características de JSP

- Permiten separar la parte dinámica de la estática en una página web
- Las páginas jsp se almacenan en el servidor en archivos con extensión .jsp.
- El código JSP es java y se encierra entre: <% y %>, por ejemplo:

```
<H1>Hora: <%= new java.util.Date() %></H1>
```

- La sintaxis también se puede expresar en formato XML

```
<jsp:xxx> ... </jsp:xxx>
```

- En una página jsp hay varios objetos implícitos (predefinidos):

```
request, response, out, session, application, config, pageContext,
page y exception
```

- Cada página JSP es compilada automáticamente hacia un servlet por el motor JSP la primera vez que se accede a esa página.

- Desde una página JSP se puede llamar a un componente JavaBean donde se puede implementar la lógica de negocio.

1.4. Funcionamiento de JSP

Una página JSP es básicamente una página Web con HTML tradicional y código Java incrustado. La extensión de fichero de una página JSP es ".jsp" en vez de ".html" o ".htm", y eso le indica al servidor que esta página requiere un tratamiento especial que se conseguirá con una extensión del servidor o un plug-in.

El servidor web comprueba si la página ha sido solicitada con anterioridad. En ese caso el servlet correspondiente ya estará cargado en memoria. Si no es así, se notifica al motor de jsp y se generará un servlet para la página.

Cuando un cliente solicita una página jsp, se ejecuta en el servidor el código JSP de la página, dando como resultado una página HTML que se fusiona con el HTML original, generando una página HTML de respuesta que será enviada al cliente.

2. Programando con JSP

2.1. Tipos de construcciones JSP

- Elementos de script:
  - Permiten especificar código Java
  - Encerrados entre <%= y %>, <% y %>, <%! y %>
- Directivas
  - Permiten controlar la estructura general del servlet
  - Encerradas entre <%@ y %>
- Comentarios
  - Permiten insertar comentarios
  - Encerrados entre <%-- y --%>
- Acciones
  - Permiten especificar los componentes JavaBean que va a utilizar la JSP
  - Encerradas entre <jsp:xxx> y </jsp:xxx>

2.2. Elementos de scriptº

Expresiones: las expresiones permiten insertar valores Java en la salida. La sintaxis es:

```
<%= expresión Java %>
<jsp:expr> ... </jsp:expr>
```

Por ejemplo:

```
hora:<%= new java.util.Date() %>
```

Las expresiones tienen acceso a los objetos implícitos. Por ejemplo:

```
<%= "Método envío formulario: " + request.getMethod() %>
```

Ejemplo con XML:

```
<jsp:expr>
  "Método envío formulario: " + request.getMethod()
</jsp:expr>
```

Scriptlets: Permiten insertar código Java directamente en la página JSP.

```
<% codigo Java; %>
<jsp:scriptlet> ... </jsp:scriptlet>
```

Por ejemplo:

```
<% i = 7; %>
```

Los scriplets tienen acceso a los objetos implícitos. Por ejemplo:

```
<% String str = “hola”;
   out.println(str);    %>
```

Ejemplo con XML:

```
<jsp:scriptlet>
  String str = “hola”;
  out.println(str);
</jsp:scriptlet>
```

Declaraciones: Permiten definir las variables o métodos que serán utilizados en la página. No tienen acceso a los objetos implícitos. La sintaxis es:

```
<%! declaración Java; %>
<jsp:decl> ... </jsp:decl>
```

Por ejemplo:

```
<%! private int i = 5; %>
<%= i %>
```

Ejemplo con XML:

```
<jsp:decl>
int i = 5;
</jsp:decl>
<jsp:expr>
“i: " + i
</jsp:expr>
```

2.3. Directivas

Directiva page: Permite definir uno o más atributos.

```
<%@ page attributo=“valor” %>
<jsp:directive.page attributo=“valor”/>
```

Los atributos serían:

```
import="paquete.clase"
contentType="MIME-Type"
isThreadSafe="true|false"
session="true|false"
buffer="sizekb|none"
autoFlush="true|false"
extends="package.class"
info="message"
errorPage="url"
isErrorPage="true|false"
nlanguage="java“
```

Por ejemplo:

```
<%@ page import=“java.util.*” %>
```

Ejemplo con XML:

```
<jsp:directive.page import=“java.util.*”/>
```

Directiva include: Permite incluir ficheros en la pagina JSP. Inserta el fichero en tiempo de compilación. La sintaxis es:

```
<%@ include file=“url” %>
<jsp:directive.include file=“url”/>
```

Por ejemplo:

```
<%@ include file=“pagina.html” %>
```

Ejemplo con XML:

```
<jsp:directive.include file=“pagina.html”/>
```

3. La primera página JSP

3.1. Requisitos para probar JSP

- Cualquier servidor web que soporte Servlets (Apache Tomcat 5.X, se encuentra en la dirección http://tomcat.apache.org/)
- Instalar Web Tools Platform (WTP) plugin a eclipse, o descargar la versión de eclipse que o incluye (http://www.eclipse.org/webtools/main.php)
- Poner el API de los servlets y JSP en el CLASSPATH. Si estamos trabajando con el Eclipse, es necesario poner los como jars externos del proyecto (verificar que jsp-api y servlet-api están incluidos como jar externos).

3.2. Objetos Implícitos

Los objetos implícitos también son llamados variables predefinidas y se utilizan directamente, sin crearlos. A continuación se muestra una tabla con los objetos implícitos que pueden ser utilizados desde las páginas JSP:

Objeto	Descripción	Tipo	Ámbito
application	Representa el contexto en el que se ejecutan las JSP	javax.servlet.ServletContext	application
session	Sesión de un cliente	javax.servlet.http.HttpSession	session
request	Extrae datos del cliente	javax.servlet.http.HttpServletRequest	request
	Envía datos al cliente	javax.servlet.http.HttpServletResponse	page
out	Envía la salida al cliente	javax.servlet.jsp.JspWriter	page
exception	Información último error	java.lang.Throwable	page
config	Representa la configuración del Servlet	javax.servlet.ServletConfig	page
pageContext	Es de donde se obtienen algunos objetos implícitos	javax.servlet.jsp.PageContext	page
page	La propia página JSP	java.lang.Object	page

La mayoría de los objetos implícitos provienen de la clase abstracta *javax.servlet.jsp.PageContext*. Cuando se compila la JSP, el servlet que se obtiene, contiene la creación de estos objetos implícitos dentro del método _jspService:

```
public void _jspService
  ( HttpServletRequest request,
    HttpServletResponse  response ) throws IOException, ServletException
{
   PageContext pageContext = null;
   HttpSession session = null;
   ServletContext application = null;
   ServletConfig config = null;
   JspWriter out = null;
   Object page = this;
   ...
   pageContext =
       _jspxFactory.getPageContext(this, request, response, "", true,
                                   8192, true);
   application = pageContext.getServletContext();
   config = pageContext.getServletConfig();
   session = pageContext.getSession();
   out = pageContext.getOut();
   ...
}
```

3.3. Ejemplos de páginas JSP sencillas

En estos ejemplos se mostrará la utilización de las distintas construcciones JSP vistas anteriormente. Para la realización de las pruebas crearemos un nuevo proyecto Web Dinámico llamado jsp, en el cuál crearemos los estos ejemplos.

Ejemplo de Expresiones (Expresiones.jsp)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
       <head>
       <title>Expresiones JSP</title>
       </head>
       <body>
       <H1>Ejemplo de expresiones JSP</H1>
       <UL>
              <LI>Fecha actual: <%=new java.util.Date()%>
              <LI>Nombre del host: <%=request.getRemoteHost()%>
              <LI>ID de la sesión: <%=session.getId()%>
              <LI>El parámetro es: <%=request.getParameter("nombre")%>
       </UL>
       </body>
</html>
```

Para probarlo, publicar la página en el servidor web e invocarla desde el navegador:

http://localhost:8080/jsp/Expresiones.jsp
http://localhost:8080/jsp/Expresiones.jsp?nombre=Alumnos_Felices

Ejemplo de Scriplets (Color.jsp)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
    <title>Scriptlets JSP</title>
    <%
        String bgColor=request.getParameter("bgColor");
        boolean hayColor;
        if (bgColor!=null) hayColor=true; else
        {
            hayColor=false;
            bgColor="WHITE";
        }
    %>
    </head>
    <body BGCOLOR="<%=bgColor%>">
    <H1>Ejemplo de scriptlets JSP</H1>
    <%
        if (hayColor) out.println("Se ha utilizado el color: " + bgColor);
        else out.println("Se ha utilizado el color por defecto: WHITE");
     %>
     </body>
</html>
```

Para probarlo, publicar la página en el servidor web e invocarla desde el navegador:

http://localhost:8080/jsp/Color.jsp
http://localhost:8080/jsp/Color.jsp?bgColor=RED

Ejemplo de declaraciones y directiva page (Contador.jsp)

```
<%@page import="java.util.*"%>
<%-- Esto en un comentario de JSP --%>
<%!
    private int cont=0;
    private Date fecha= new Date();
    private String host="<I>Sin acceso previo</I>";
%>
    <p>Esta página ha sido accedida <b><%= ++cont%></b> veces desde que
    se inició el servidor.</p>
    <p>El último acceso ha sido desde: <b><%=host%></b> con fecha
    <b><%=fecha%></b></p>
<%
    host=request.getRemoteHost();
    fecha=new Date();
%>
```

Para probarlo, publicar la página en el servidor web e invocarla desde el navegador:

http://localhost:8080/jsp/Contador.jsp

Ejemplo de directiva include (Empresa.jsp)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
    <title>Ejemplo de uso de un contador incluido en un JSP</title>
    </head>
    <body>
        <H1>Ejemplo de uso de un contador incluido en un JSP</H1>
        <%@include file="Contador.jsp"%>
    </body>
</html>
```

Para probarlo, publicar la página en el servidor web e invocarla desde el navegador:

http://localhost:8080/jsp/Empresa.jsp

4. Utilización de Páginas de Error

4.1. Excepciones en JSP

La gestión de excepciones en JSP se lleva a cabo a través del objeto *exception*. Para manejar las excepciones en las JSP se deben seguir los siguientes pasos:

- Escribir un Servlet, JSP u otro componente para que lance excepciones en determinadas condiciones. Por ejemplo desde una JSP podría indicarse como:

  ```
  public Object metodo() throws NullPointerException { ... }
  ```

- Escribir una JSP que será la página de error usando la directiva page con isErrorPage="true". Es decir que si ocurre un error en otras JSP que usen el componente, se ejecutará esta página:

  ```
  <%@ page isErrorPage="true" import="java.util.*" %>
  ```

- En la página de error usar el objeto *exception* para obtener información de la excepción

  ```
  <%= exception.toString() %>
  <% exception.printStackTrace(); %>
  ```

- En las JSP que usan el componente, indicar qué página se ejecutará si se produce algún error, mediante la directiva page, estableciendo errorPage a la página de error.

  ```
  <%@ page isThreadSafe="false" import="java.util.*" errorPage="error.jsp" %>
  ```

4.2. Ejemplo de utilización de páginas de error

A continuación mostramos un ejemplo en el que intervienen dos páginas, la página que puede provocar la excepción o lanzar el error (Division.jsp) y la página que saldrá como tratamiento de la excepción (ErrDiv.jsp). La página *Division.jsp* podría quedar codificada de la siguiente forma:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
    <title>Manejo de errores de JSP</title>
    </head>
    <body >
    <%@page errorPage="ErrDiv.jsp"%>
    <H1>Ejemplo de manejo de errores en JSP</H1>
    <%!
        private double toDouble(String value)
        {
            return(Double.valueOf(value).doubleValue());
        }
    %>
    <%
        double op1 = toDouble(request.getParameter("op1"));
        double op2= toDouble(request.getParameter("op2"));
        double res = op1/op2;
    %>
    <TABLE border=1>
    <TR><TH></TH><TH>División</TH></TR>
    <TR><TH>Operando 1: </TH><Td><%=op1%></Td></TR>
    <TR><TH>Operando 2: </TH><Td><%=op2%></Td></TR>
    <TR><TH>Resultado: </TH><Td><%=res%></Td></TR>
    </TABLE>
    </body>
</html>
```

La página de error *ErrDiv.jsp* quedaría como:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
    <title>Página de errores en JSP</title>
    </head>
    <body >
    <%@page isErrorPage="true"%>
    <H1>Página de errores en JSP</H1>
    <P>Division.jsp ha reportado el siguiente error:
        <b><%=exception%></b>
    </P>
    <P>El error que ha ocurrido es: <pre>
    <% exception.printStackTrace(new java.io.PrintWriter(out)); %>
    </pre>
    </P>
    </body>
</html>
```

Para probarlo, publicar la página en el servidor web e invocarla desde el navegador:

http://localhost:8080/jsp/Division.jsp?op1=12&op2=6
http://localhost:8080/jsp/Division.jsp?op1=12&op2=0

Para que salga la página de error, uno de los operándoos tiene que ser una letra o no pasar parámetros:

http://localhost:8080/jsp/Division.jsp?op1=12&op2=
http://localhost:8080/jsp/Division.jsp?op1=12&op2=a

5. Ejemplo completo de Servlets y JSP con acceso a datos

Partiendo del ejemplo de los jugadores de fútbol de la práctica anterior, vamos a desarrollar un ejemplo en el que utilizaremos primeramente la página web Futbol.html que se ampliará para permitir votar por un jugador. Luego esta información se procesará con un servlet intermedio Futbol.java que se va a encargar de actualizar la tabla de *Jugadores* (actualizando los votos de jugadores existentes o insertando nuevos jugadores). Por último desde este servlet se dará paso al servlet de *TablaVotos* para mostrarle al cliente cómo han quedado las estadísticas de votos.

5.1. Requisitos previos

Es necesario tener disponible los ejemplos de la práctica anterior y la siguiente base de datos en MySQL:

- Se debe crear esta BD y sus tablas en *MySQL* con la siguiente estructura:

BDJugadores

Tabla Jugadores:

Campo	Tipo
Nombre	Varchar (50)
Votos	Integer

Tabla Registro:

Campo	Tipo
Nombre	Varchar (50)
Correo	Varchar (30)
Visitas	Integer

La tabla de *Jugadores* se rellenará inicialmente con datos de algunos jugadores y se pondrán a 0 los votos de cada uno. Por ejemplo:

Nombre	Votos
Roberto Carlos	0
Raul	0
Torres	0
Ronaldinho	0
Ronaldo	0
Figo	0
Zidane	0

5.2. Página HTML que realiza la petición al servlet

La página HTML *Futbol.html* permitirá que un cliente introduzca su nombre y correo y pueda emitir su voto por algún jugador de los que se visualizan. Si no le gusta ninguno, podrá añadir un nuevo jugador. Esta página enviará su petición al servlet *Futbol.java* para que éste se encargue de actualizar la base de datos. La página tendría el siguiente código:

```
<html>
<head>
<title>Estadísticas de Futbol</title>
</head>
<body>
<center><H1>Estadísticas de Jugadores de Futbol</H1></center>
<p align="center"><font color="#002424" size="7"><u>VOTE POR EL MEJOR
   JUGADOR</u></font></p>
<p align="center"><font color="#002424" size="7"><u>DE FUTBOL DE
```

```
   2003</u></font></p>
<form action="./Futbol" method="POST" left>
    <p align="left">Nombre del Visitante: <input type="text" size="20"
     name="txtNombre">
    eMail: <input type="text" size="20" name="txtMail"></p>
    <p align="left"><input type="radio" name="R1" value="Roberto Carlos">Roberto
     Carlos</p>
    <p align="left"><input type="radio" name="R1" value="Raul">Raul</p>
    ...
    <p align="left"><input type="radio" name="R1" value="Otros">Otros <input
     type="text" size="20" name="txtOtros"></p>
    <p align="left"><input type="submit" name="B1" value="Votar"> <input
     type="reset" name="B2" value="Reset"></p>
</form>
</body>
</html>
```

5.3. Servlet que se encarga de actualizar la base de datos

A continuación mostramos el código del servlet *Futbol.java* que atenderá la petición de *Futbol.html*, conectándose a la Base de Datos *BDJugadores* y actualizando la tabla *Jugadores*. En caso del que el jugador por el que se ha votado ya se encuentre en la tabla, se incrementará su voto en 1, si no está, se insertará el nuevo jugador y se le pondrán sus votos a 1.

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

public class Futbol extends HttpServlet
{
    private Connection con;
    private Statement set;
    private ResultSet rs;
    String cad;

    public void init(ServletConfig cfg) throws ServletException
    {
        String sURL="jdbc:mysql://localhost/BDJugadores";
        super.init(cfg);
        String userName = "root";
        String password = "admin";

        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            con = DriverManager.getConnection(sURL, userName, password);
             System.out.println("Se ha conectado");
        }
        catch(Exception e)
        {
            System.out.println( "No se ha conectado");
        }
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        // Obtener la sesion
        HttpSession s = req.getSession(true);

        // Guardar el nombre del cliente en la sesión
        // para poderlo utilizar en el siguiente servlet
        String nombreP=(String)req.getParameter("txtNombre");
        s.putValue("nombreCliente", nombreP);

        String nombre=(String)req.getParameter("R1");

        if (nombre.equals("Otros"))
            nombre=(String)req.getParameter("txtOtros");

        boolean existe = false;
        try
```

```
        {
            set = con.createStatement();

            rs = set.executeQuery("SELECT * FROM Jugadores");
            while (rs.next())
            {
                cad = rs.getString("Nombre");
                cad = cad.trim();
                if (cad.compareTo(nombre.trim())==0)
                existe = true;
            }

            rs.close();

            set.close();
        }

        catch(Exception e)
        {
            System.out.println("No lee de la tabla");
        }

        try
        {
            set = con.createStatement();
            if (existe)
                set.executeUpdate( "UPDATE Jugadores SET votos=votos+1 " +
                                   "WHERE nombre LIKE '%" + nombre + "%'");
            else
                set.executeUpdate( "INSERT INTO Jugadores " +
                                   "(nombre,votos) VALUES ('" + nombre + "',1)");

            rs.close();
            set.close();
        }
        catch(Exception e)
        {
            System.out.println( "No inserta ni modifica la tabla");
        }

        // Llamada al servlet que nos visualiza
        // las estadísticas de jugadores
        res.sendRedirect(res.encodeRedirectURL("./TablaVotos.jsp"));
    }

    public void doGet(HttpServletRequest req,   HttpServletResponse res)
        throws ServletException, IOException
    {
        try{doPost(req,res);}catch (Exception e)
        {
        }
    }

    public void destroy()
    {
        try
        {
            con.close();
        }
        catch (Exception e)
        {
        }

        super.destroy();
    }
}
```

5.4. Servlet que muestra los datos

Por último tendríamos la codificación del servlet que se encarga de mostrar las estadísticas de votos al cliente. Este es el *TablaVotos.java* que quedaría:

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

```
public class TablaVotos extends HttpServlet
{
    private Connection con;
    private Statement set;
    private ResultSet rs;

    public void init(ServletConfig cfg) throws ServletException
    {
        String sURL="jdbc:mysql://localhost/BDJugadores";
        super.init(cfg);
        String userName = "root";
        String password = "admin";

         try
         {
          Class.forName("com.mysql.jdbc.Driver").newInstance();
          con = DriverManager.getConnection(sURL, userName, password);
         System.out.println("Se ha conectado");
        }
        catch(Exception e)
        {
            System.out.println("No se ha conectado");
        }
    }
    public void doPost(HttpServletRequest req,HttpServletResponse res)
        throws ServletException, IOException
    {
        HttpSession s = req.getSession(true);
        //Leer el nombre del cliente de la sesión para darle las
        //gracias al final
        String nombreP = (String)s.getValue("nombreCliente");
        PrintWriter out = res.getWriter();
        res.setContentType("text/html");
        out.println("<html><head><title>JDBC</title></head>");
        out.println("<body><font size=10>");
        out.println("<h1>Ejemplo de JDBC</h1>");
        out.println("<table border=1>");
        out.println("<tr><td><b>Jugador</b></td><td><b>Votos" +
                    "</b></td></tr>");
        try
        {
            String nombre;
            int votos;
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores");
            while (rs.next())
            {
                nombre = rs.getString("Nombre");
                votos = rs.getInt("Votos");
                out.println("<tr><td>"+ nombre + "</td><td>" +
                    votos + "</td></tr>");
            }
            rs.close();
            set.close();
        }
        catch(Exception e)
        {
            System.out.println("No lee de la tabla");
        }
        out.println("</table>");
        out.println("<h3>Muchas gracias " + nombreP + " por su visita</h3>");
        out.println("</form></font></body></html>");
        out.close();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        try{doPost(req,res);}catch (Exception e){}
    }
    public void destroy()
    {
        try {con.close();}
        catch (Exception e){}
        super.destroy();
    }
}
```

6. Utilización de JSP

Convertiremos el servlet *TablaVotos.java* en una página JSP, puesto que es la que más se adecua, ya que la mayor parte de su código es la confección de la pagina HTML de respuesta y quedaría mucho más claro como página JSP que como servlet. Se llamará *TablaVotos.jsp* y su código sería:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@page import="java.io.*"%>
<%@page import="java.util.*"%>
<%@page import="java.sql.*"%>
<%@page import="java.util.*"%>
<html>
    <head><title>JDBC</title></head>
    <body><font size=10>
    <h1>Ejemplo de JDBC</h1>
    <table border=1>
    <tr><td><b>Jugador</b></td><td><b>Votos</b></td></tr>
    <%
        Connection con;
        Statement set;
        ResultSet rs;
        String sURL="jdbc:mysql://localhost/BDJugadores";
        String userName = "root";
        String password = "admin";
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            con = DriverManager.getConnection(sURL, userName, password);
            System.out.println("Se ha conectado");
            String nombre;
            int votos;
            set = con.createStatement();
            rs = set.executeQuery("SELECT * FROM Jugadores");
            while (rs.next())
            {
                nombre = rs.getString("Nombre");
                votos = rs.getInt("Votos");
                out.println("<tr><td>"+ nombre + "</td><td>" + votos
                    + "</td></tr>");
            }
            rs.close();
            set.close();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println("Error en acceso a BD");
        }
    %>
    </table>
    <h3>Muchas gracias por su visita</h3>
    </form></font></body>
</html>
```

7. Ampliación de la práctica

Para culminar la práctica se debe desarrollar por parte de los estudiantes las siguientes tareas:

- Crear páginas JSP para controlar los posibles errores de acceso a bases de datos.
- Convertir *Futbol.html* en una página JSP que dinámicamente se actualice teniendo en cuenta los nuevos jugadores que se van añadiendo. Es decir, que aparezcan los nuevos jugadores de la misma forma que los iniciales y que se siga permitiendo añadir nuevos por medio del cuadro de texto.