Arquitectura y Diseño de Sistemas Web y C/S

Práctica 4:

Tecnologías Lado Servidor: Servlets, JSPs y el patrón Modelo Vista Controlador

Grupo 9

Contenido

Introducción

Esta práctica se centra en tres aspectos fundamentales del desarrollo web en Java: Servlets, JavaServer Pages (JSP) y el Patrón de Diseño Modelo-Vista-Controlador (MVC).

Los Servlets son componentes Java que extienden la funcionalidad de un servidor web. Actúan como controladores que manejan las solicitudes y respuestas HTTP. En esta práctica, exploraremos la creación de Servlets para procesar formularios, interactuar con bases de datos y generar dinámicamente contenido HTML.

JSP es una tecnología que simplifica la creación de páginas web dinámicas en Java. Permite la inclusión de código Java directamente en el HTML, facilitando la generación de contenido dinámico. En esta práctica, abordaremos ejemplos de JSP para acceder a bases de datos, gestionar sesiones y crear páginas web interactivas.

MVC es un patrón de diseño que separa la lógica de la aplicación en tres componentes principales: Modelo, Vista y Controlador. El Modelo representa los datos y la lógica de negocio, la Vista se encarga de la presentación y el Controlador maneja las interacciones del usuario. En el contexto de una aplicación de simulación de ganancia de potencia en circuitos de F1, implementaremos este patrón para estructurar de manera eficiente la aplicación, facilitando la escalabilidad y mantenibilidad del código.

A lo largo de la práctica, exploraremos cómo estos conceptos se combinan para desarrollar aplicaciones web robustas y eficientes en Java.

1. Práctica de Servlets

Como se ha comentado anteriormente, uno de los objetivos de esta práctica es utilizar el API de los Servlets, para ello vamos a trabajar con distintos ejemplos en los que vamos a ver su ejecución, además de explicar someramente su funcionamiento.

En este caso hemos creado una Pagina.jsp que va a ser como un menú para acceder a los 5 códigos propuestos. Para que el servidor arranque directamente en esta página ahí que configurar dentro de la carpeta WEB-INF el fichero web.xml de esta forma:

```
<!-- Configuración para establecer la página de inicio -->
<welcome-file-list>
    <welcome-file>Paginas.jsp</welcome-file>
</welcome-file-list>
```

El contenido de la pagina.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Páginas JSP</title>
</head>
<body>
<h2><center>Enlaces a Páginas JSP</center></h2>
<ul>
    <li><a href="calculadora.jsp">Calculadora</a></li>
    <li><a href="encuesta.jsp">Encuesta</a></li>
    <li><a href="formulario.jsp">Formulario</a></li>
    <li><a href="HelloServer.jsp">Hello Server</a></li>
    <li><a href="primitiva.jsp">Primitiva</a></li>
</ul>
</body>
</html>
```

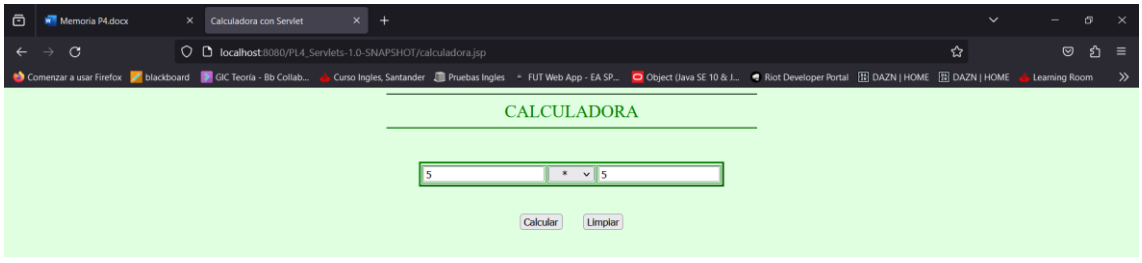
Los 5 códigos tanto la parte de los servlets como la parte de jsp ha sido proporcionada por el profesor por que lo que no nos vamos a centrar en la explicación del código si no en el funcionamiento.

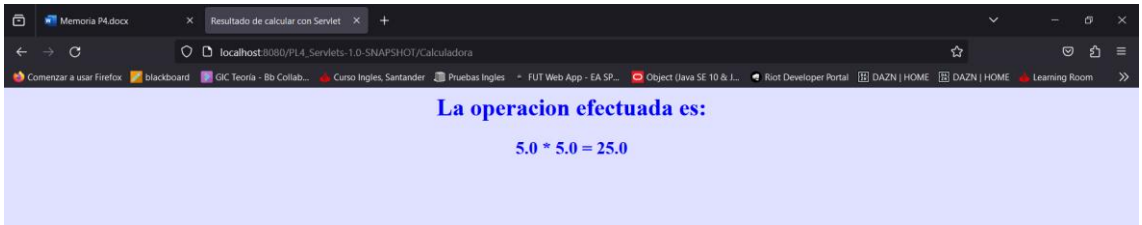
Una vez instalado el servidor GlassFish y configurado los puertos y el xml arrancamos el servidor entrando a la pagina de inicio en la que tendremos:

1.1. Hola Mundo

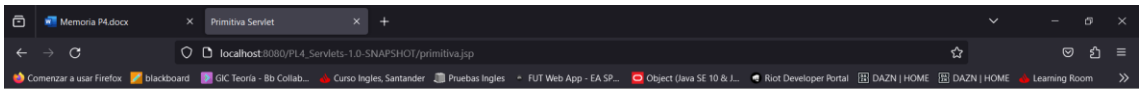
1.2. Acceso Básico a Formularios

1.3. Calculadora





1.4. Primitiva

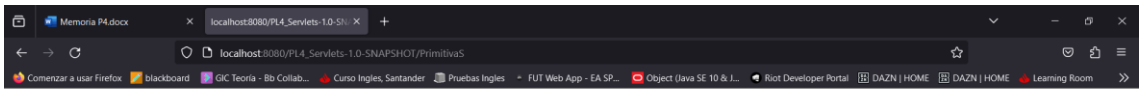


Primitiva Servlet

Introduce tu combinación y pulsa el botón de enviar

NUM1: 2
NUM2: 3
NUM3: 12
NUM4: 1
NUM5: 17
NUM6: 33

Enviar Borrar



Primitiva Servlet

Tu combinación es:

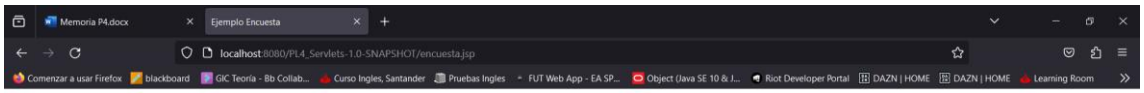
2 3 12 1 17 33

Números acertados: **0**

La combinación ganadora es:

8 9 29 37 39 44

1.5. Acceso a datos



Por favor rellene todos los datos

Nombre:

David

E-Mail:

david@gmail.com

Pregunta: ¿Piensas utilizar a los Servlets para los proyectos a partir de ahora?

Si ◉ No ○

Enviar Borrar

2. Práctica de Java Server Pages

Vamos a crear un nuevo proyecto al que hemos llamado AccesoDatos. Para este hemos usado el entorno netbeans pues el manejo que tiene de la base de datos derby es mucho más simple. Pero antes hemos tenido que configurar los drivers de la base de datos para enlazarlo a derby.

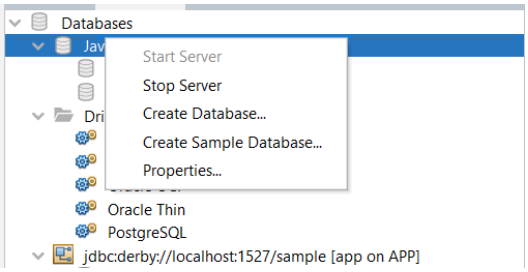
Primero necesitamos descargar Derby desde la web:

https://db.apache.org/derby/derby_downloads.html

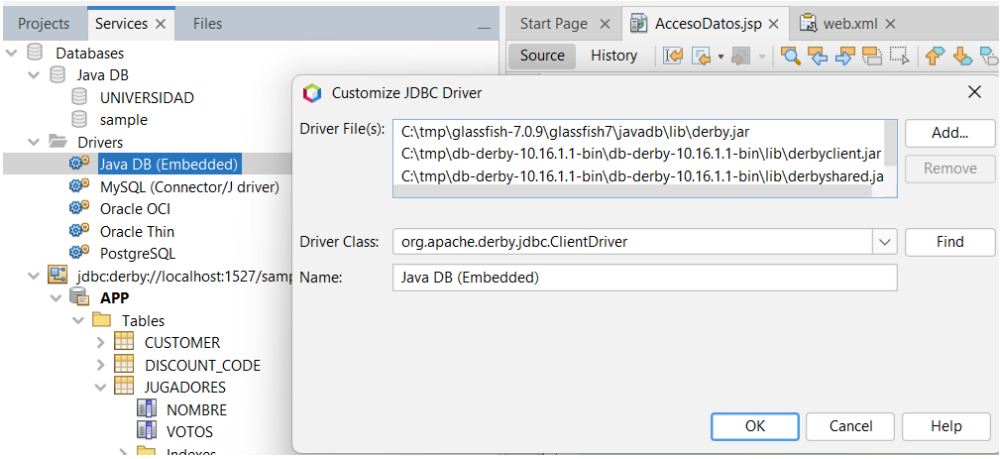
Una vez aquí descargamos aquella compatible para nuestra versión de JDK.

Una vez descargado podemos acceder a NetBeans para empezar la configuración.

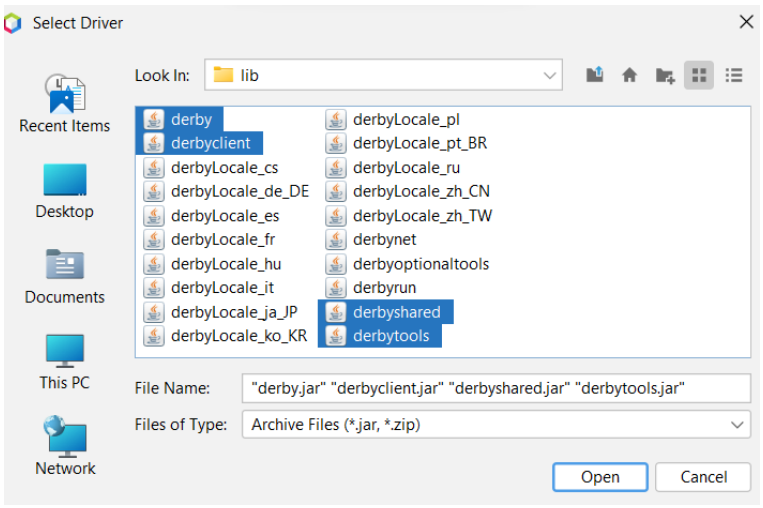
1. iniciamos el servidor de la base de datos.



2. Añadimos los .jar necesarios para conectarnos a Derby en nuestro driver para JDBC:



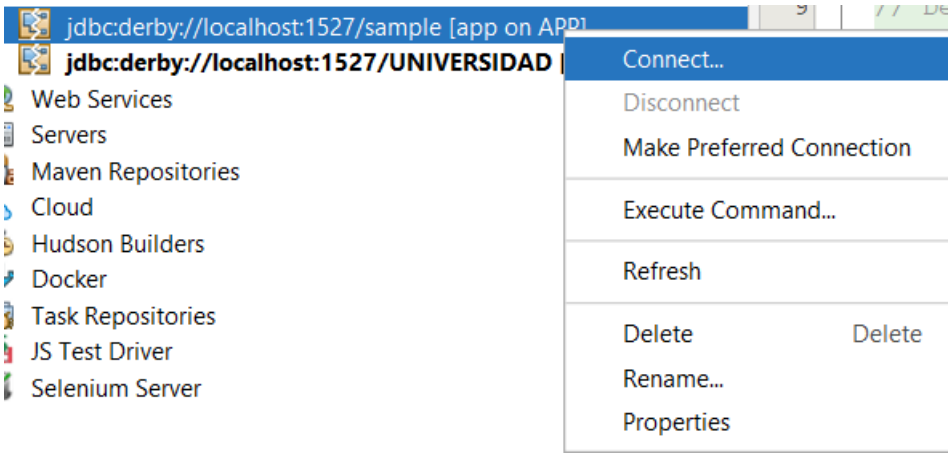
Y añadimos los siguientes:



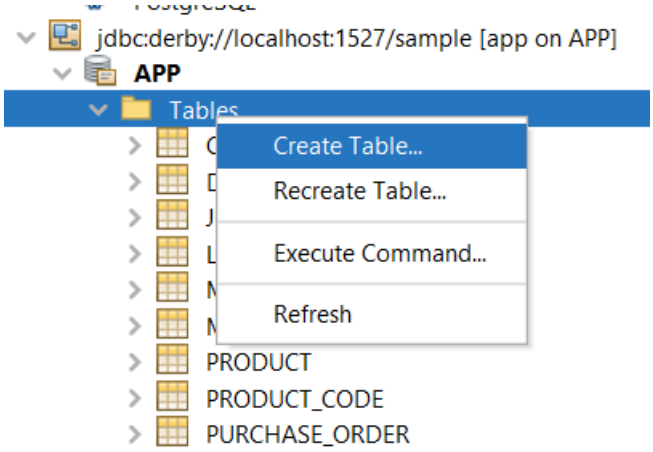
Ya podemos comenzar con la parte de la práctica.

2.1. Acceso a datos (I)

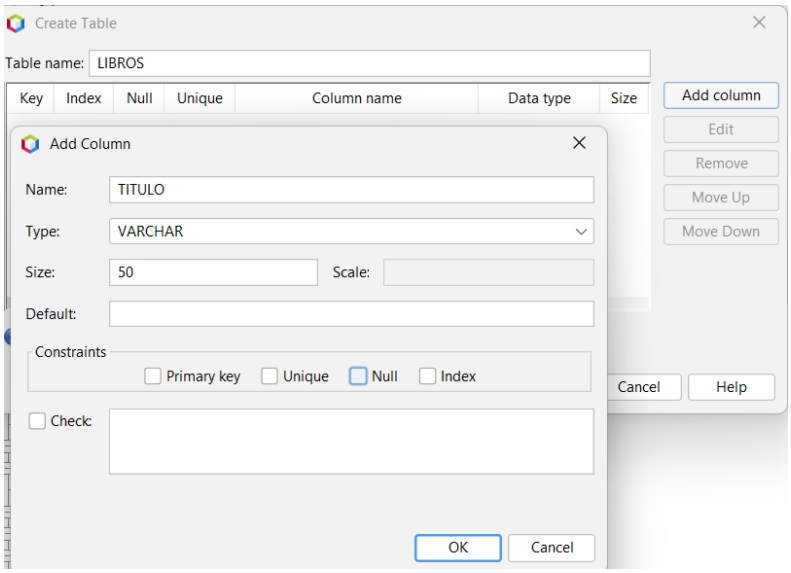
Para poder realizar esta parte tenemos que añadir una tabla como indica el enunciado. Esto se hace iniciando la base de datos Derby e introduciendo una tablanueva con sus columnas(atributos).



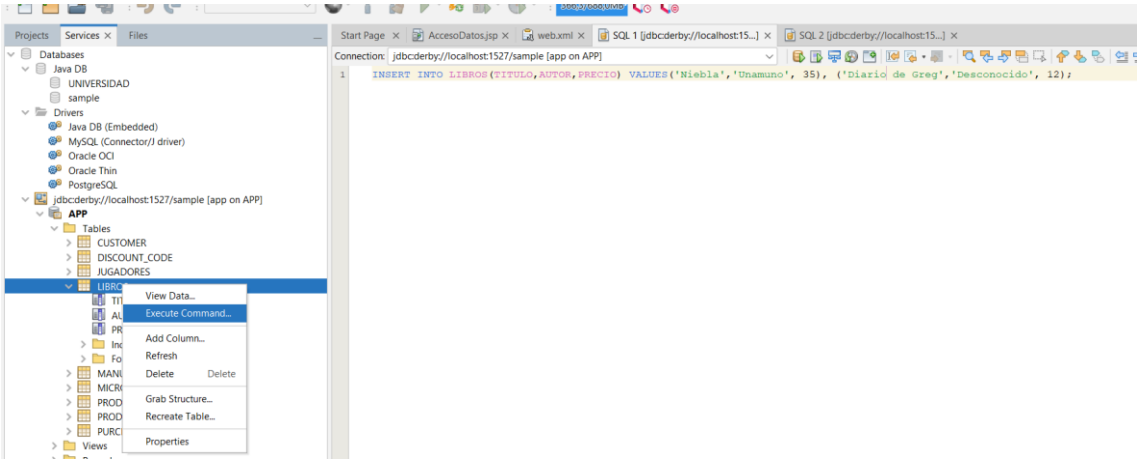
Creamos la tabla.



Añadimos columnas a la tabla:



Solo nos faltaría añadir datos a nuestra tabla para que nos muestre algunos al ejecutar el programa facilitado:

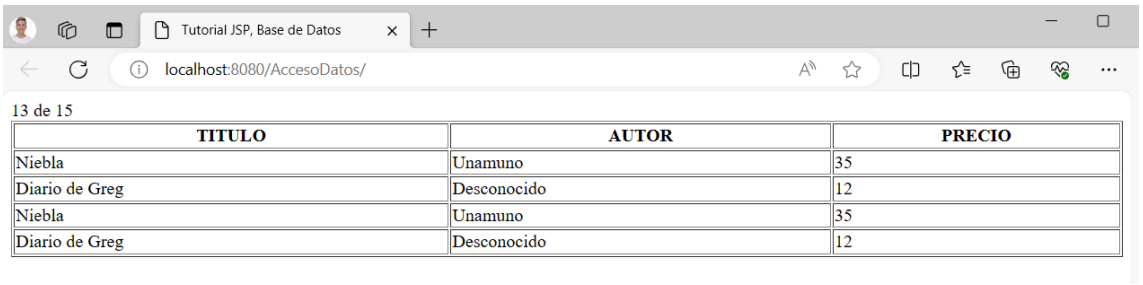


Creamos ahora dentro de WebPages nuestro AccesoDatos.jsp, donde pegaremos el código del profesor.

```
<html>

<head>

<title>Tutorial JSP, Base de Datos</title>

</head>

<body>

<%@ page import="java.sql.*" %>

<%!

// Declaraciones de las variables utilizadas para la conexión a la base de datos y para la
//recuperación de datos de las tablas

Connection c;

Statement s;

ResultSet rs;

ResultSetMetaData rsmd;

%>

<%

// Inicialización de las variables necesarias para la conexión a la base de datos y realización de
//consultas

c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample/", "app", "app");

s = c.createStatement();

rs = s.executeQuery("SELECT * FROM LIBROS");

rsmd = rs.getMetaData();
```

```
%>
13 de 15
<table width="100%" border="1">
<tr>
<% for( int i=1; i <= rsmd.getColumnCount(); i++ ) { %>
<%-- Obtenemos los nombres de las columnas y los colocamos
como cabecera de la tabla --%>
<th><%= rsmd.getColumnLabel( i ) %></th>
<% } %>
</tr>
<% while( rs.next() ) { %>
<tr>
<% for( int i=1; i <= rsmd.getColumnCount(); i++ ) { %>
<%-- Recuperamos los valores de las columnas que
corresponden a cada uno de los registros de la
tabla. Hay que recoger correctamente el tipo de
dato que contiene la columna --%>
<% if( i == 3 ) { %>
<td><%= rs.getInt( i ) %></td>
<% } else { %>
<td><%= rs.getString( i ) %></td>
<% } } %>
</tr>
<% } %>
</table>
</body>
</html>
```

Mostrandonos al ejecutar el proyecto:



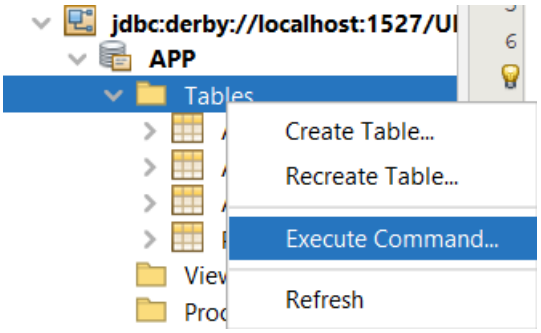
TITULO	AUTOR	PRECIO
Niebla	Unamuno	35
Diario de Greg	Desconocido	12
Niebla	Unamuno	35
Diario de Greg	Desconocido	12

2.2. Acceso a datos (II)

Ahora tenemos que crear una lógica y crear tres tablas diferentes, de manera que tengamos foreing keys y una lógica más comleja.

Para crear las tablas esta vez usaremos comandos SQL, pues al tener un modelo relacional necesitamos foreing keys y es mucho más sencillo si se hace de esta manera.

Primero tenemos que conectarnos a la base de datos para ejecutar un comando:



Nos mostrara una pantalla en blanco donde introduciremos los siguientes comandos. Hay que añadir que hay que ejecutar cada create table de manera singular, no ejecuta todos los comandos:

```
-- Tabla PROFESOR
CREATE TABLE PROFESOR (
    DNI VARCHAR(30) PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50),
    sueldo DOUBLE,
    -- Otros campos relacionados con el profesor
);

-- Tabla ALUMNO
CREATE TABLE ALUMNO (
    DNI VARCHAR(30) PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
```

```
    apellido VARCHAR(50)
    -- Otros campos relacionados con el alumno
);

-- Tabla ASIGNATURA
CREATE TABLE ASIGNATURA (
    cod_asig VARCHAR(30) PRIMARY KEY,
    nombre VARCHAR(30) NOT NULL,
    descripcion VARCHAR(100),
    profesor_asig VARCHAR(30),
    FOREIGN KEY (profesor_asig) REFERENCES PROFESOR(DNI)
);

-- Tabla para representar la relación muchos a muchos entre ALUMNO y ASIGNATURA
CREATE TABLE ALUMNO_ASIGNATURA (
    DNI_alumno VARCHAR(30),
    cod_asig_asignatura VARCHAR(30),
    PRIMARY KEY (DNI_alumno, cod_asig_asignatura),
    FOREIGN KEY (DNI_alumno) REFERENCES ALUMNO(DNI),
    FOREIGN KEY (cod_asig_asignatura) REFERENCES ASIGNATURA(cod_asig)
);
```

Como se puede ver hay una tabla llamada alumno_asignatura que no esta presente en el modelo. Esta es necesaria para poder manejar la relación N:N entre alumnos y asignaturas.

Una vez creadas las tablas falta añadir datos:

```
-- Datos de ejemplo para PROFESOR
INSERT INTO PROFESOR (DNI, nombre, apellido, sueldo) VALUES
('123456789A', 'Juan', 'Pérez', 50000.00),
('987654321B', 'María', 'Gómez', 48000.50),
('555555555C', 'Pedro', 'Martínez', 52000.75);

-- Datos de ejemplo para ALUMNO
```

```
INSERT INTO ALUMNO (DNI, nombre, apellido) VALUES
('111111111X', 'Laura', 'Rodríguez'),
('222222222Y', 'Carlos', 'García'),
('333333333Z', 'Ana', 'López');

-- Datos de ejemplo para ASIGNATURA
INSERT INTO ASIGNATURA (cod_asig, nombre, descripcion, profesor_asig) VALUES
('MAT101', 'Matemáticas I', 'Introducción a las matemáticas', '123456789A'),
('FIS102', 'Física II', 'Termodinámica y óptica', '987654321B'),
('INF201', 'Programación Avanzada', 'Desarrollo de software', '555555555C');

-- Datos de ejemplo para ALUMNO_ASIGNATURA
INSERT INTO ALUMNO_ASIGNATURA (DNI_alumno, cod_asig_asignatura) VALUES
('111111111X', 'MAT101'),
('111111111X', 'FIS102'),
('222222222Y', 'MAT101'),
('333333333Z', 'INF201');
```

Al igual que para crear las tablas hay que ejecutar cada insert de manera individual.

Para ver si se han insertado bien:

```
SELECT * FROM APP.ALUMNO FETCH FIRST 100 ROWS ONLY;
```

ELECT * FROM APP.ALUMNO ... ×

Max. rows: 100 Fetched Rows: 3

#	DNI	NOMBRE	APELLIDO
	11111111X	Laura	Rodríguez
	22222222Y	Carlos	García
	33333333Z	Ana	López

Solo nos falta modificar el código y realizar la consulta que nos indica el enunciado como sigue:

Hemos creado un jsp para crear el formulario y mostrar la lista de nombres de profesores, este llama al AccesoDatos.jsp que se encarga del acceso a la base de datos y realiza la consulta.

- Formulario.html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Formulario Consulta</title>
  </head>
  <body>
    <h3>Consultar los alumnos a los que imparte el profesor</h3>
    <form action="AccesoDatos2.jsp" method="post">
      <label for="nombreProfesor">Nombre del Profesor:</label>
      <input type="text" name="nombreProfesor" id="nombreProfesor" required>
      <input type="submit" value="Consultar Alumnos">
    </form>

  <body>
```

```
<%@ page import="java.sql.*" %>
<%!
// Declaraciones de las variables utilizadas para la conexión a la base de datos y para la
//recuperación de datos de las tablas
Connection c;
Statement s;
ResultSet rs;
ResultSetMetaData rsmd;
%>
<%
// Inicialización de las variables necesarias para la conexión a la base de datos y
realización de //consultas
c = DriverManager.getConnection("jdbc:derby://localhost:1527/UNIVERSIDAD/",
"app", "app");
s = c.createStatement();
rs = s.executeQuery("SELECT nombre FROM PROFESOR");
rsmd = rs.getMetaData();
%>
<table width="100%" border="1">
  <tr>
    <% for( int i=1; i <= rsmd.getColumnCount(); i++ ) { %>
    <%-- Obtenemos los nombres de las columnas y los colocamos
    como cabecera de la tabla --%>
    <th><%= rsmd.getColumnLabel( i ) %></th>
      <% } %>
  </tr>
  <% while( rs.next() ) { %>
  <tr>
    <% for( int i=1; i <= rsmd.getColumnCount(); i++ ) { %>
    <%-- Recuperamos los valores de las columnas que
    corresponden a cada uno de los registros de la
    tabla. Hay que recoger correctamente el tipo de
```

```
            dato que contiene la columna --%>
            <% if( i == 3 ) { %>
            <td><%= rs.getInt( i ) %></td>
            <% } else { %>
            <td><%= rs.getString( i ) %></td>
            <% } } %>
          </tr>
          <% } %>
      </table>
  </body>
</html>
```

- AccesoDatos2.jsp:

```
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.sql.*" %>
<html>
<head>
  <title>Consulta de Alumnos</title>
</head>
<body>

  <%
      Connection c = null;
      Statement s = null;
      ResultSet rs = null;
      ResultSetMetaData rsmd = null;
      boolean profesorEncontrado = false;

      String nombreProfesor = request.getParameter("nombreProfesor");
```

```
    if (nombreProfesor != null && !nombreProfesor.isEmpty()) {
        try {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            c = DriverManager.getConnection("jdbc:derby://localhost:1527/UNIVERSIDAD/",
"app", "app");
            s = c.createStatement();
            String query = "SELECT ALUMNO.DNI, ALUMNO.nombre " +
                    "FROM ALUMNO " +
                    "JOIN ALUMNO_ASIGNATURA ON ALUMNO.DNI =
ALUMNO_ASIGNATURA.DNI_alumno " +
                    "JOIN ASIGNATURA ON ALUMNO_ASIGNATURA.cod_asig_asignatura =
ASIGNATURA.cod_asig " +
                    "JOIN PROFESOR ON ASIGNATURA.profesor_asig = PROFESOR.DNI " +
                    "WHERE PROFESOR.nombre = '" + nombreProfesor + "'";
            rs = s.executeQuery(query);
            rsmd = rs.getMetaData();

            // Verificamos si se encontraron resultados
            profesorEncontrado = rsmd != null && rs.next();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
  %>

  <% if (profesorEncontrado) { %>
    <table width="100%" border="1">
        <tr>
            <% for (int i = 1; i <= rsmd.getColumnCount(); i++) { %>
                <th><%= rsmd.getColumnLabel(i) %></th>
            <% } %>
```

```
        </tr>
        <% do { %>
          <tr>
            <% for (int i = 1; i <= rsmd.getColumnCount(); i++) { %>
              <td><%= rs.getString(i) %></td>
            <% } %>
          </tr>
        <% } while (rs.next()); %>
      </table>
    <% } else if (nombreProfesor != null && !nombreProfesor.isEmpty()) { %>
      <script>
        alert("No se encontraron resultados para el profesor <%= nombreProfesor %>");
        window.location.href = "formulario.html";
      </script>
    <% } %>


</body>
</html>
```

Obtenemos el nombre gracias a un getSession que veremos más adelante además de que si el nombre no existe lo indica para introducir otro.

2.3. Sesiones

En esta parte veremos como se pueden realizar sesiones para mantener información dentro del servidor. En este pequeño ejemplo almacenaremos un nombre que luego aparecerá en una página diferente gracias a las funciones setAtribute y getAtribute.

Para realizar esto hemos usado el ejemplo del profesor, que consta de 4 archivos:

- Eje3Sesion.html:

```
<HTML><head>
<title> Ejemplo de Sesión </title></head>
<body>
<h1> Ejemplo de sesión </h1>
<form method="post" action="sesionEje.jsp">
Por favor, introduce tu nombre:
<input type="text" name="nombre">
<input type="submit" value="enviar información"> </form> </body>
</HTML>
```

- sesionEje.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<HTML>
   <head> <title> Ejemplo de Sesión </title> </head>
   <body>
      <%
         String val = request.getParameter("nombre");
         if (val != null) session.setAttribute("Nombre",val);
      %>
   <center> <h1>Ejemplo de Sesión</h1>
      Donde quieres ir!!!
      <a href=sesionEje1.jsp>Ir a Página 1</a>
      <a href=sesionEje2.jsp>Ir a Página 2</a>
   </body>
</HTML>
```

- sesionEje1.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<HTML><head> <title> Ejemplo de Sesión </title> </head> <body>
   <center> <h1>Ejemplo de Sesión</h1>
      Hola, <%=session.getAttribute("Nombre")%>
      Bienvenido a la página 1
   </body>
</HTML>
```

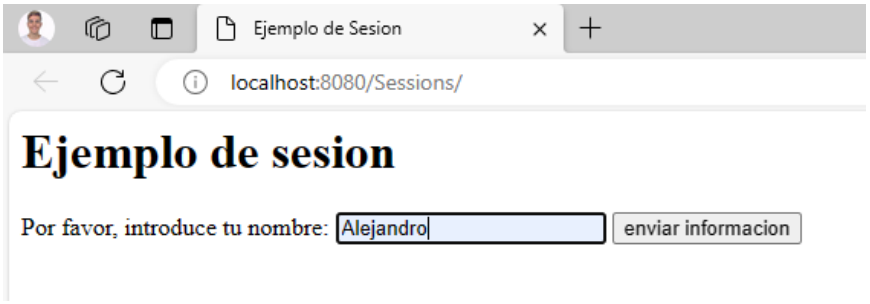
- sesionEje2.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
```

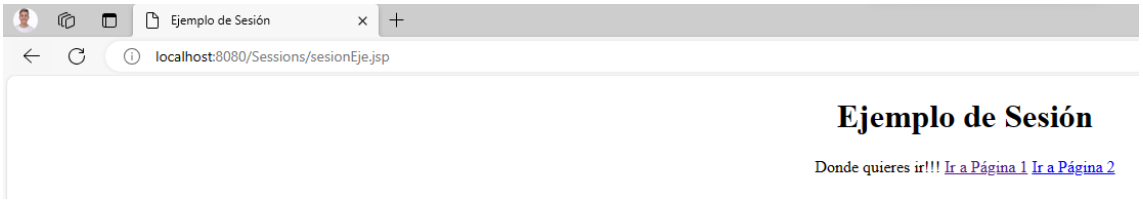
```
<HTML><head> <title> Ejemplo de Sesión </title> </head> <body>
  <center> <h1>Ejemplo de Sesión</h1>
    Hola, <%=session.getAttribute("Nombre")%>
    Bienvenido a la página 2
  </body>
</HTML>
```

Esto nos mostrará algo así:

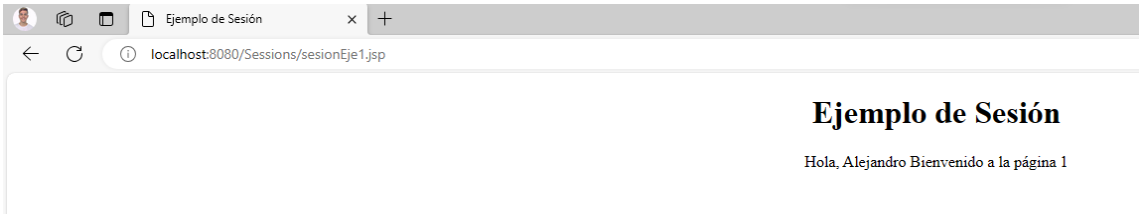
- Eje3sesion.html:



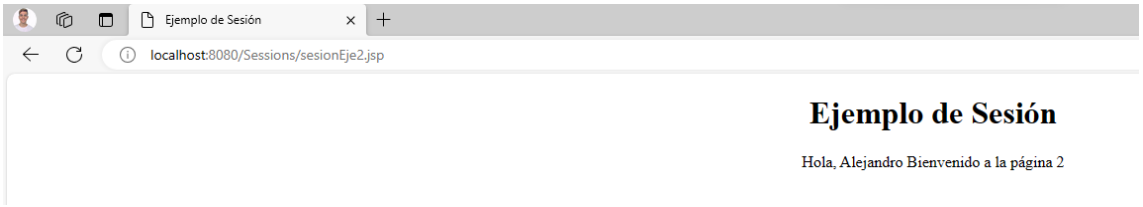
- sesionEje.jsp:



- SesiopnEje1.jsp:

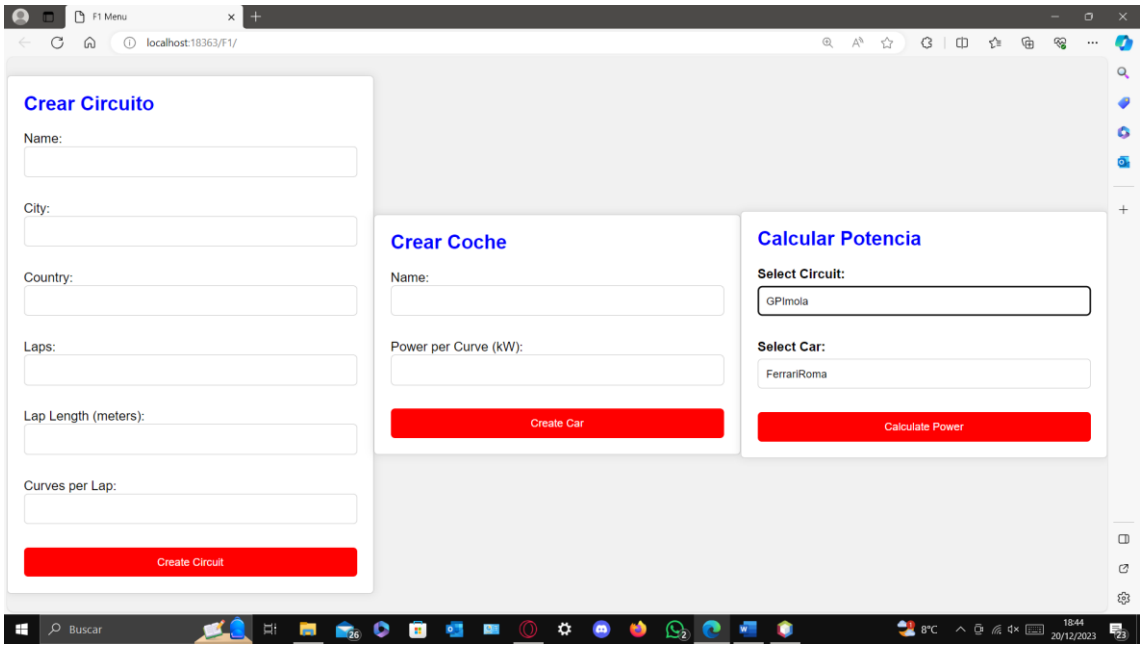


- SesionEje2.jsp:



3. Patrón de Diseño MVC

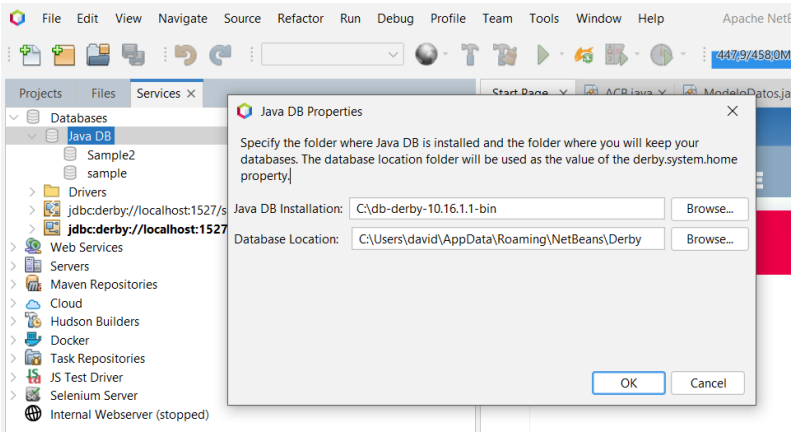
Para el desarrollo de esta miniaplicación se ha seguido el patron software modelo vista controlador en el que tenemos una tecnología jsp que actúa como vista, que es lo que nosotros veremos en el navegador web:



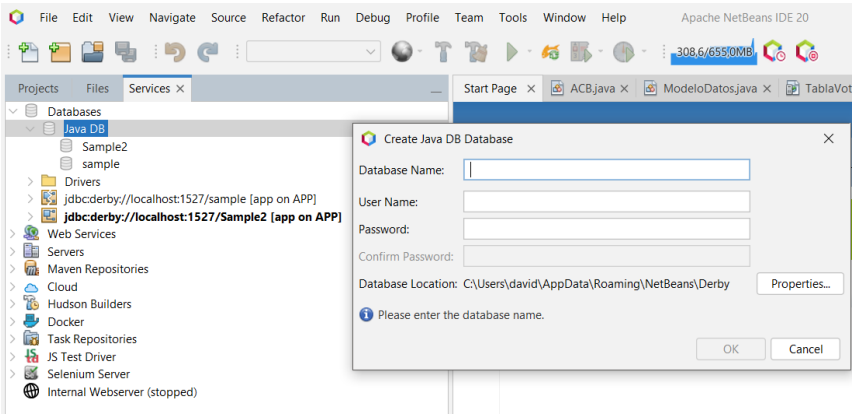
Desde nuestra vista enviamos peticiones al controlador que es la tecnología servlet , peticiones como crear el circuito, crear el coche , calcular la potencia. También desde en controlador podemos llevar un control de la gestión de sesiones en el que guardamos información o la recuperamos.

Por último tenemos el modelo que usa la tecnología JDBC, que se conecta a nuestra base de datos en este caso de apache derby. **MUY IMPORTANTE EN EL PROYECTO QUE SE HA ENVIADO SOLO ESTA EL CODIGO EN JAVA LA BASE DE DATOS NO SE PUEDE ENVIAR SE HA DE TENER INSTALADA AL IGUAL QUE GLASFISH PARA PODER USAR ESTE CODIGO, VAMOS A ENSEÑAR COMO SE HA DE CONFIGURAR CORRECTMANETE LA BASE DE DATOS PARA QUE ESTO FUNCIONE PASO A PASO:**

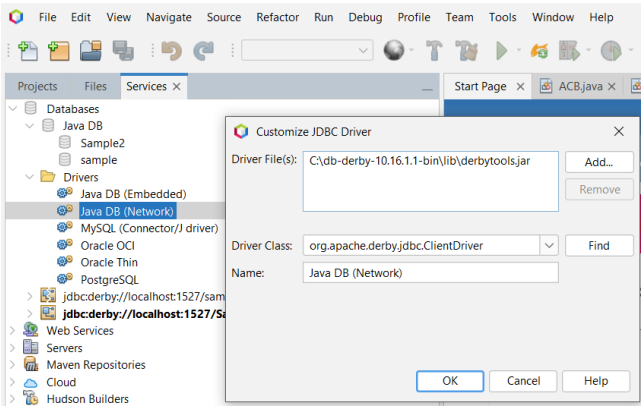
PASO 1) DENTRO DE SERVICES EN SELECCIONAMOS LA BASE DE DATOS Y EN PROPIEDADES INCLUIMOS EL ARCHIVO DE DERBY INSTALADO EN ESTE CASO LA VERSION 16:



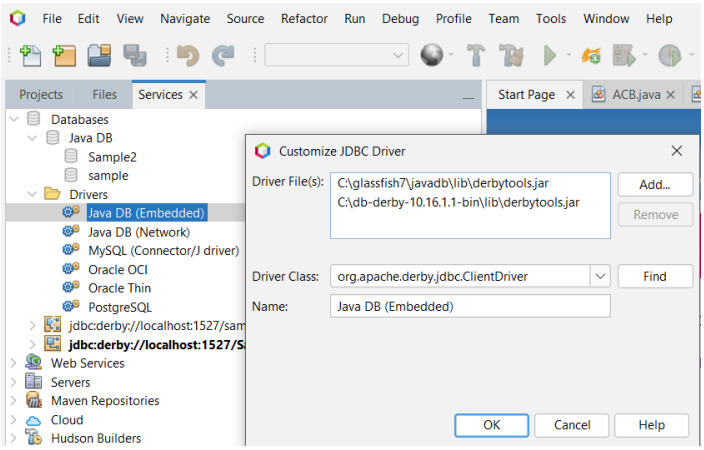
PASO 2) LE DAMOS A CREAR LA BASE DE DATOS, LE PONEMOS UN NOMBRE UN USUARIO Y UNA CONTRASEÑA, EN NUESTRO CASO HEMOS CREADO UNA QUE SE LLAMA SAMPLE2, Y EL USUARIO Y CONTRASEÑA ES admin admin:



PASO 3) DENTRO DE LOS DRIVERS EN JAVA DB (NETWORK) LE DAMOS A CUSTOMIZE Y INCLUIMOS EL JAR DE DERBY LLAMADO DERBYTOOLS, SE ENCUENTRA EN LA CARPETA INSTALADA DENTRO DE LIB:

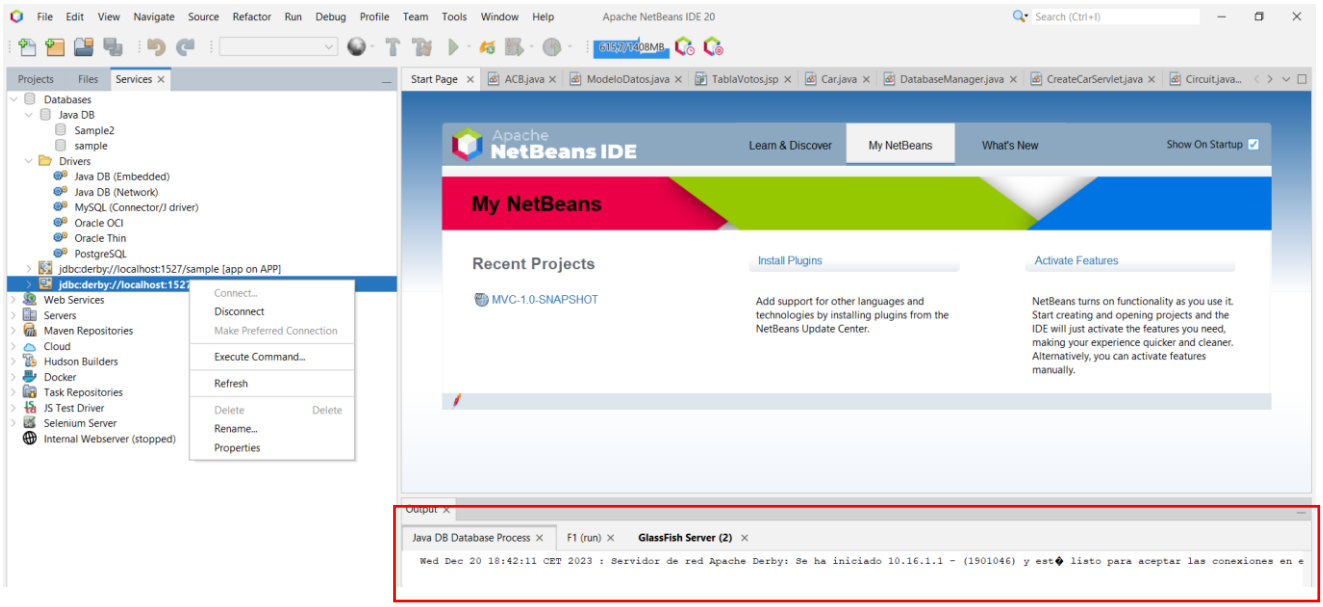


PASO 4) EN JAVA DB (EMBEDDED) AÑADIMOS EL DERBYTOOLS DE DERBY Y EL DE GLASFISH

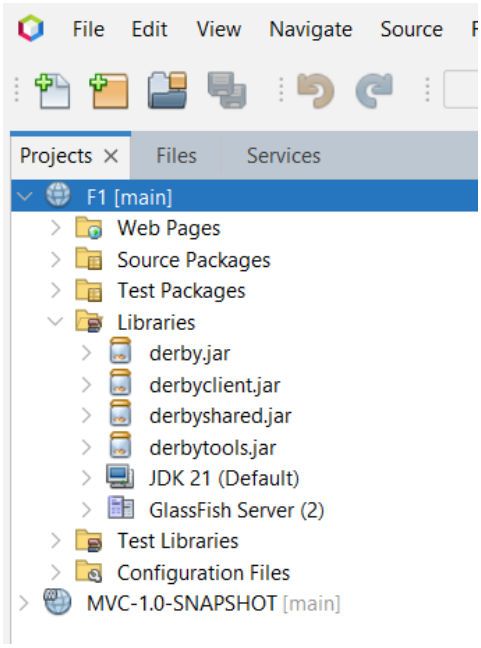


EN EL RESTO DE LOS DRIVERS NO HAY QUE TOCAR NADA SE DEJA EL VALOR POR DEFECTO

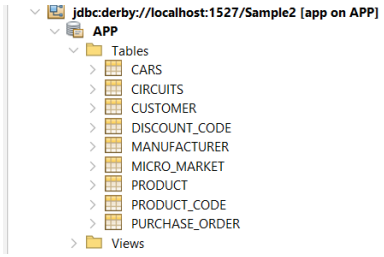
PASO 5) UNA VEZ TODO AÑADIDO CORRECTAMENTE LE DAMOS A CONECTAR LA BASE DE DATOS Y NOS DEBERIA SALER ABAJO UN MENSAJE DE CONFIRMACION DE SERVICIO:



AHORA DENTRO DEL PROYECTO QUE ES DE TIPO JAVA WITH ANT- JAVA WEB- WEB APLICATION, EN LAS LIBRERIAS INCLUIMOS ESTOS .JAR:



AHORA VAMOS A LAS TABLAS DE LA BASE DE DATOS



Y CREAMOS LA TABLA CON EL NOMBRE CARS Y CIRCUITS YA QUE ESTA ASI DISEÑADO EN EL CODIGO:

DENTRO DE LA TABLA CARS CREAMOS LOS ATRIBUTOS

NAME→VARCHAR

POWERPERCURVE→INTEGER

SELECT * FROM APP.CARS FE... ×

Max. rows: 100 Fetched Rows: 9

#	NAME	POWERPERCURVE
1	FerrariRoma	9
2	RenautClio	5
3	NissanTerrano2	5
4	Peugot208	5
5	BmwM3	8
6	AudiA6	7
7	NissanMontero	6
8	TeslaModelSPlaidPlus	9
9	NissanPatrolGR	8

DENTRO DE LA TABLA DE CIRCUITOS CREAMOS LOS ATRIBUTOS

NAME→VARCHAR

CITY→VARCHAR

COUNTRY→VARCHAR

LAPS→INTEGER

CURVES→INTEGER

LAPLENGTH→INTEGER

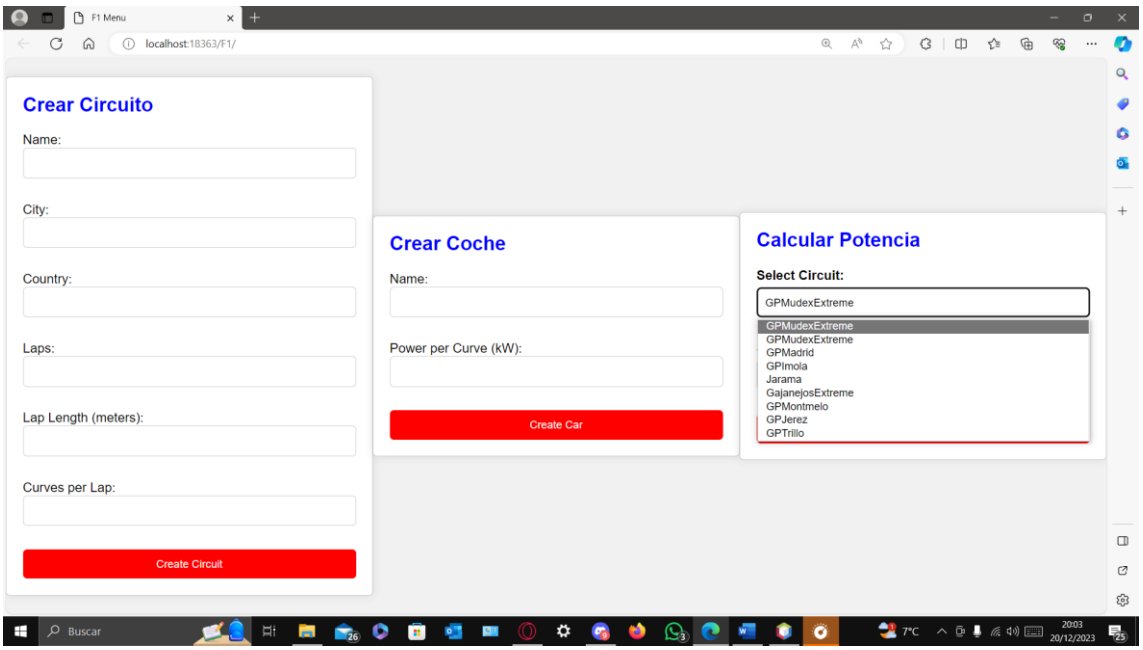
SELECT * FROM APP.CIRCUIT... ×

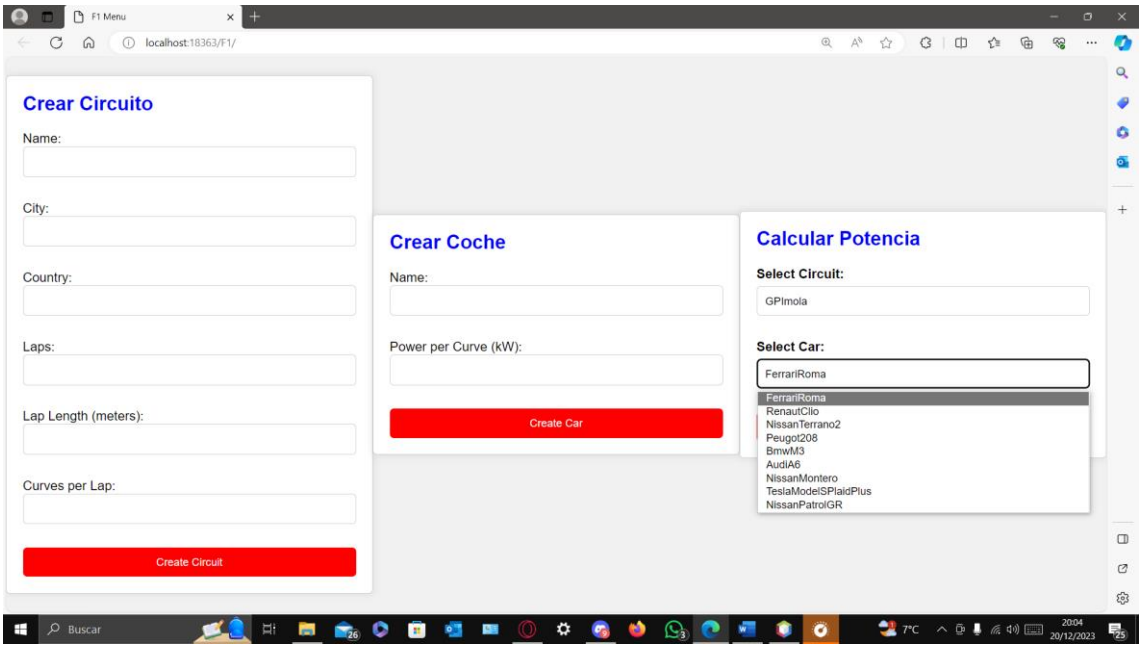
Max. rows: 100 Fetched Rows: 9 Matching Rows:

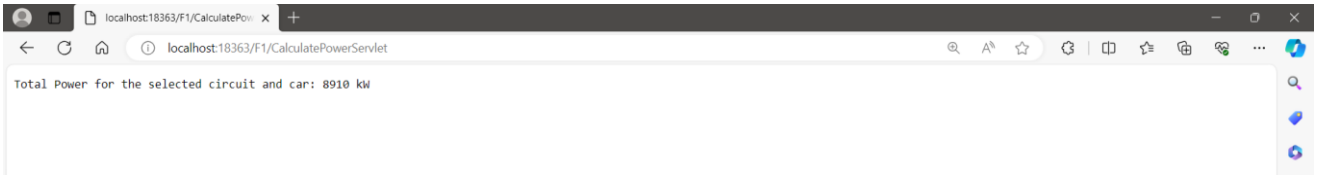
#	NAME	CITY	COUNTRY	LAPS	CURVES	LAPLENGTH
1	GPMudexExtreme	Guadalajara	Spain	55	18	8000
2	GPMudexExtreme	Guadalajara	Spain	55	18	8000
3	GPMadrid	Madrid	Spain	44	14	3333
4	GPImola	Imola	Italia	55	18	4375
5	Jarama	Madrid	Spain	60	15	4000
6	GajanejosExtreme	Guadalajara	Spain	49	14	3000
7	GPMontmelo	Barcelona	Spain	55	19	4355
8	GPJerez	Jerez	Spain	65	16	3900
9	GPTrillo	Guadalajara	Spain	56	19	6500

SI NO SE PONEN ESTOS NOMBRES NO VA A FUNCIONAR YA QUE CUANDO EL CODIGO HACE CONSULTAS CON ESOS NOMBRES, SUPER IMPORTANTE

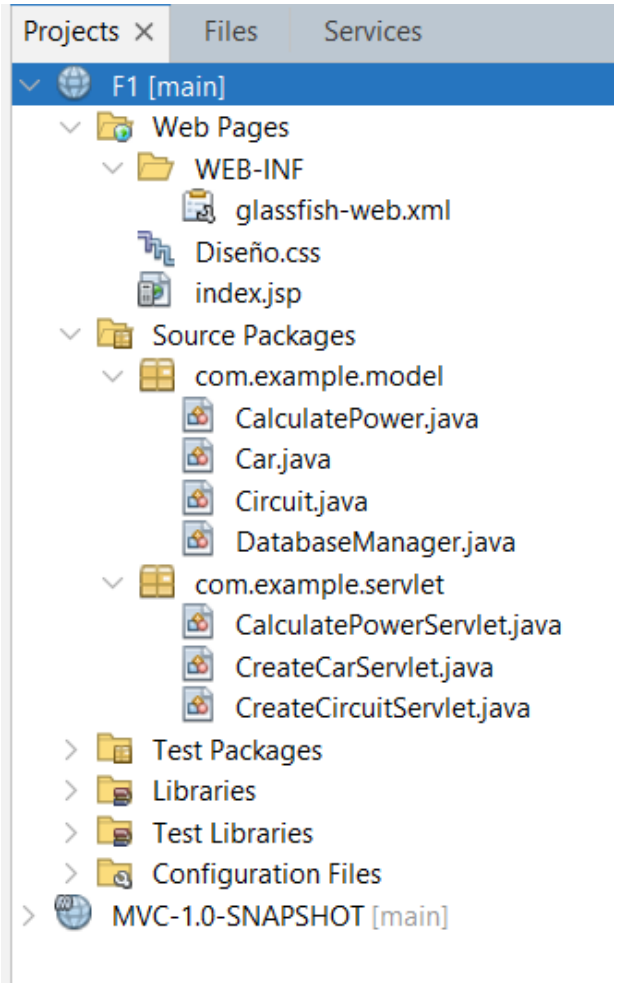
PASAMOS A VER EL FUNCIONAMIENTO LO PRIMERO ES CREAR UN COCHE Y UN CIRCUITO O VARIOS, LOS QUE CADA UNO DESEE, Y LUEGO NOS SALDRA UNA LISTA DE LOS COCHES Y DE LOS CIRCUITOS CREADOS:







PASAMOS A VER LA ESTRUCTURA DEL CODIGO:



Tenemos el código divido en dos partes por un lado tenemos la vista que es el index.jsp que es lo que vemos en el navegador web, luego en el controlador tenemos por un lado los servlets para calcular potencia , para crear el coche y el circuito y luego las 3 clases .java que simplemente definimos los atributos de cada clase.

Y por último la clase mas importante que es la DatabaseManager que es desde donde hacemos todas las consultas sql y accedemos y guardamos los datos de la clase, vamos a pasar a explicar la clase:

Lo primero es conectarnos a la base de datos y crear la función de abrir y cerrar conexión:

```
14
15  public class DatabaseManager {
16      private static final String DB_URL = "jdbc:derby://localhost:1527/Sample2;user=app;password=app";
17      private static Connection connection;
18
19      public static void abrirConexion() {
20          try {
21              Class.forName("org.apache.derby.jdbc.ClientDriver");
22              DriverManager.registerDriver(new org.apache.derby.jdbc.ClientDriver());
23              connection = DriverManager.getConnection(DB_URL);
24              System.out.println("Se ha conectado");
25          } catch (Exception e) {
26              System.out.println("No se ha conectado");
27              e.printStackTrace();
28          }
29      }
30
31      public static void cerrarConexion() {
32          try {
33              if (connection != null && !connection.isClosed()) {
34                  connection.close();
35                  System.out.println("Se ha cerrado la conexión");
36              }
37          } catch (SQLException e) {
38              e.printStackTrace();
39          }
40      }
41
```

Luego creamos los tres métodos para guardar en la base de datos el circuito y el coche, para devolver todos los circuitos y todos los coches, esta clase será la clave para luego ver en la vista una lista con el nombre de todos los circuitos creados y luego una vez seleccionado tenemos la función de getid, para saber que circuito tenemos o coche por el id y poder econtrarlo fácilmente en la base de datos:

```
2       public static void saveCar(Car car) throws SQLException {
3           abrirConexion();
4           try {
5               String sql = "INSERT INTO cars (name, powerPerCurve) VALUES (?, ?)";
6               try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
7                   preparedStatement.setString(1, car.getName());
8                   preparedStatement.setInt(2, car.getPowerPerCurve());
9                   preparedStatement.executeUpdate();
0               }
1           } finally {
2               cerrarConexion();
3           }
4       }
5
6       public static List<Car> getAllCars() throws SQLException {
7           abrirConexion();
8           List<Car> cars = new ArrayList<>();
9           try {
0               String sql = "SELECT * FROM cars";
1               try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
2                   try (ResultSet resultSet = preparedStatement.executeQuery()) {
3                       while (resultSet.next()) {
4                           Car car = new Car(
5                                   resultSet.getString("name"),
6                                   resultSet.getInt("powerPerCurve")
7                           );
8                           cars.add(car);
9                       }
0                   }
1               }
2           } finally {
3               cerrarConexion();
```

```java
// Métodos para obtener un coche por ID
public static Car getCarById(String carId) throws SQLException {
    abrirConexion();
    try {
        String sql = "SELECT * FROM cars WHERE name = ?";
        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
            preparedStatement.setString(1, carId);
            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                if (resultSet.next()) {
                    return new Car(
                            resultSet.getString("name"),
                            resultSet.getInt("powerPerCurve")
                    );
                }
            }
        }
    } finally {
        cerrarConexion();
    }
    return null;
}
```

```java
public static void saveCircuit(Circuit circuit) throws SQLException {
    abrirConexion();
    try {
        String sql = "INSERT INTO circuits (name, city, country, laps, lapLength, curves  ) VALUES (?, ?, ?, ?, ?, ?)"
        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
            preparedStatement.setString(1, circuit.getName());
            preparedStatement.setString(2, circuit.getCity());
            preparedStatement.setString(3, circuit.getCountry());
            preparedStatement.setInt(4, circuit.getLaps());
            preparedStatement.setInt(5, circuit.getLapLength());
            preparedStatement.setInt(6, circuit.getCurves());
            preparedStatement.executeUpdate();
        }
    } finally {
        cerrarConexion();
    }
}
```

```java
// Dentro del método getAllCircuits
public static List<Circuit> getAllCircuits() throws SQLException {
    abrirConexion();
    List<Circuit> circuits = new ArrayList<>();
    try {
        String sql = "SELECT * FROM circuits";
        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {
            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                while (resultSet.next()) {
                    Circuit circuit = new Circuit(
                            resultSet.getString("name"),
                            resultSet.getString("city"),
                            resultSet.getString("country"),
                            resultSet.getInt("laps"),
                            resultSet.getInt("lapLength"),
                            resultSet.getInt("curves")
                    );
                    circuits.add(circuit);
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace(); // Agrega este mensaje de registro
    } finally {
        cerrarConexion();
    }
    return circuits;
}
```

Ahora vamos a ver como hacemos una lista de despegables gracias a la tecnología java server pages:

```
<form action="CalculatePowerServlet" method="post">
    <h2>Calcular Potencia</h2>
    <label>Select Circuit:</label>
    <select name="circuitId">
        <% List<Circuit> circuits = new ArrayList<>();
            try {
                circuits = DatabaseManager.getAllCircuits();
            } catch (Exception e) {
                e.printStackTrace();
            }
            for (Circuit circuit : circuits) { %>
        <option value="<%= circuit.getName() %>"><%= circuit.getName() %></option>
        <% } %>
    </select><br>

    <label>Select Car:</label>
    <select name="carId">
        <% List<Car> cars = new ArrayList<>();
            try {
                cars = DatabaseManager.getAllCars();
            } catch (Exception e) {
                e.printStackTrace();
            }
            for (Car car : cars) { %>
        <option value="<%= car.getName() %>"><%= car.getName() %></option>
        <% } %>
    </select><br>

    <input type="submit" value="Calculate Power">
</form>
```

Conclusión

En resumen, esta práctica ha explorado Servlets, JSP y el Patrón MVC en el desarrollo web en Java. Los Servlets actuaron como controladores para gestionar solicitudes y respuestas, mientras que JSP simplificó la creación de contenido dinámico. La implementación del Patrón MVC en una aplicación de simulación de F1 demostró eficiencia al separar la lógica de la aplicación en Modelo, Vista y Controlador, facilitando la escalabilidad y mantenibilidad del código. En conjunto, estos conceptos ofrecen herramientas poderosas para construir aplicaciones web robustas y eficientes en Java.