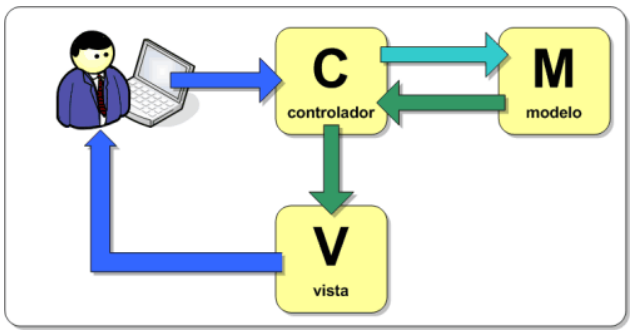
Arquitectura y Diseño de Sistemas WEB y C/S

Tema: MVC



Grupo 6

Integrantes

Daniel Ferreiro Rodríguez

Bianca Marinela Lupu

Carlos Javier Hellín Asensio

Francisco Calles Esteban

Darius Dumitras Tamas

Grado de Ingeniería Informática

Curso: 2021-2022

Contenido

Introducción

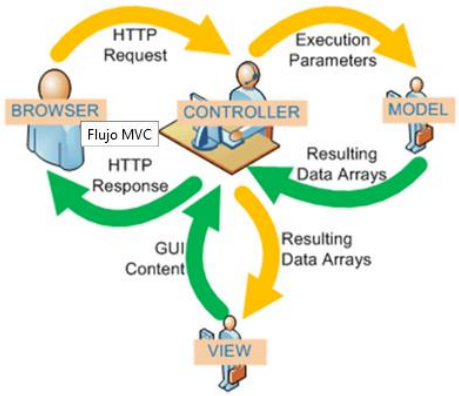
MVC (Model View Controller) es un patrón en el diseño de software que separa las interfaces de usuario, los datos y lógica de control. Gestiona los eventos y las comunicaciones. Modelo consagrado en el tiempo y usado para todo tipo de aplicaciones, lenguajes y plataformas de desarrollo.

Las tres partes del patrón de diseño de software MVC:

1. **Modelo**: Maneja datos y lógica de negocios.
2. **Vista**: Se encarga del diseño y presentación.
3. **Controlador**: Enruta comandos a los modelos y vistas.

Este patrón proporciona:

- Permitir la reutilización de código y crear aplicaciones con mayor calidad.
- Mejorar el mantenimiento.
- Evitar el código Espagueti.
- Perfecto para equipos multidisciplinarios ya que separa responsabilidades.



Ejemplo de flujo del MVC

Análisis del problema

Esta aplicación permite simular la ganancia de potencia en cada circuito de F1 gracias al sistema **KERS**. Esta energía eléctrica es almacenada para un uso futuro. La ganancia de potencia será directamente proporcional al número de vueltas del circuito ya que el coche tendrá que hacer una mayor cantidad de frenadas.

KERS (Kinetic Energy Recovery System) es un dispositivo que permite reducir la velocidad de un vehículo transformando parte de su energía cinética en energía eléctrica.

A continuación, se mostrará las tablas de la BBDD para almacenar los datos. Se verá las interfaces y código que permite añadir o modificar circuitos y coches. Que se ha hecho para calcular la ganancia de potencia. Como se gestionan los errores, éxitos y cómo se validan los valores de los campos a rellenar tanto en el cliente como el servidor. Por último, se encontrará un manual para que el usuario sepa todos los detalles de la aplicación.

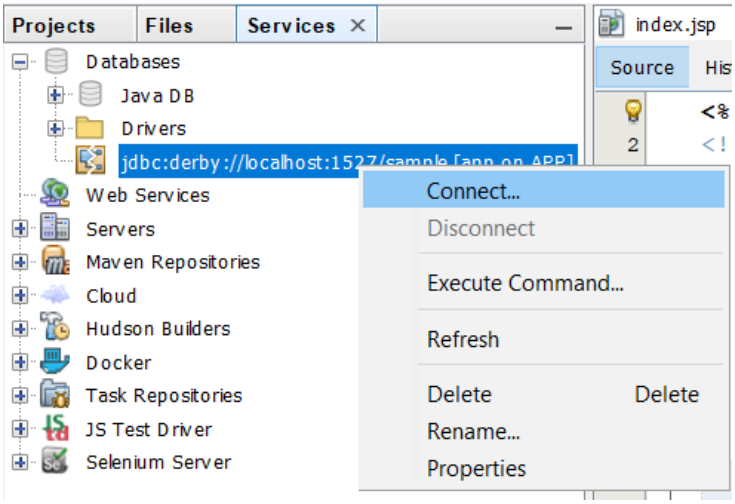
Implementación

Ejercicio práctico

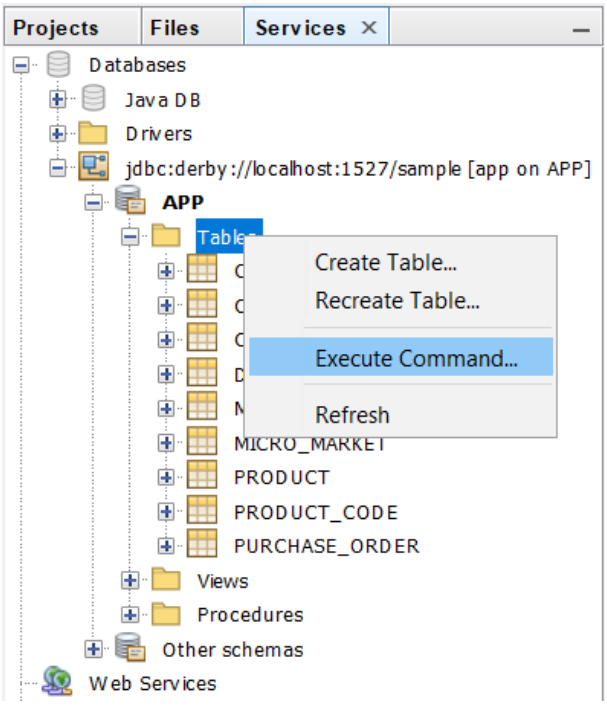
Creación de la BBDD

La base de datos empleada será Derby proporcionada por el propio Netbeans. Dentro de esta hay una creada por defecto denominada **sample** con unas tablas de ejemplo ya creadas, y es la utilizada para la realización de la práctica.

El primer paso es conectar el servicio, en este caso la propia base de datos cuyo puerto es 1527, y el esquema de base de datos por determinado **sample [app on APP]**.



Una vez conectada se accede dentro del esquema **APP** a la carpeta encargada de almacenar las tablas de datos **Table**, y se selecciona la opción de ejecutar comandos. Esto permite ejecutar cualquier tipo de acción sobre la base de datos empleando comandos sql.



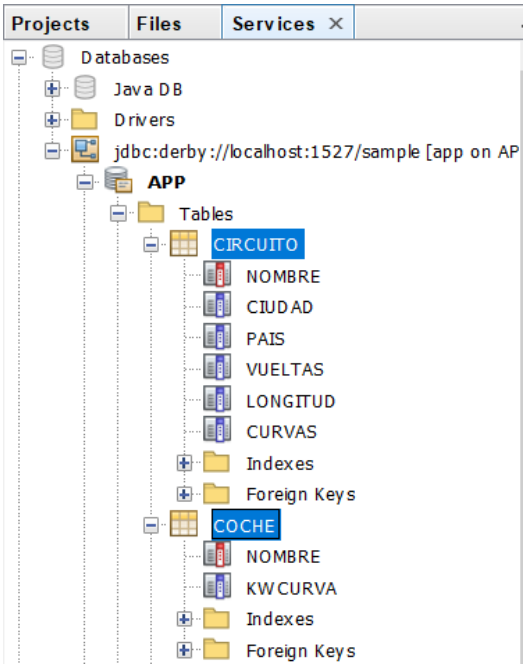
En este caso en concreto servirá para crear las tablas **circuito** y **coche** con sus respectivos atributos y características, indicando sus claves primarias y tipos de valores, y además se insertará un caso como ejemplo en cada tabla.

Todos los valores y comandos utilizados se encuentran en un fichero externo sql, y las acciones de creación de tablas e inserción se podrán observar en la ventana del lado izquierdo de la interfaz.

SQL 1 [jdbc:derby://localhost:15...]

Connection: jdbc:derby://localhost:1527/sample [app on APP]

```
1   CREATE TABLE "CIRCUITO" (
2           NOMBRE varchar(20) not null primary key,
3           CIUDAD varchar(20) not null,
4           PAIS varchar(20) not null,
5           VUELTAS integer not null,
6           LONGITUD integer not null,
7           CURVAS integer not null
8   );
9
10  CREATE TABLE "COCHE" (
11          NOMBRE varchar(20) not null primary key,
12          KWCURVA integer not null
13  );
14
15  INSERT INTO "CIRCUITO" VALUES ('Mónaco', 'Montecarlo', 'Mónaco', 78, 3337, 19);
16  INSERT INTO "COCHE" VALUES ('Alpine A521', 5);
17
```

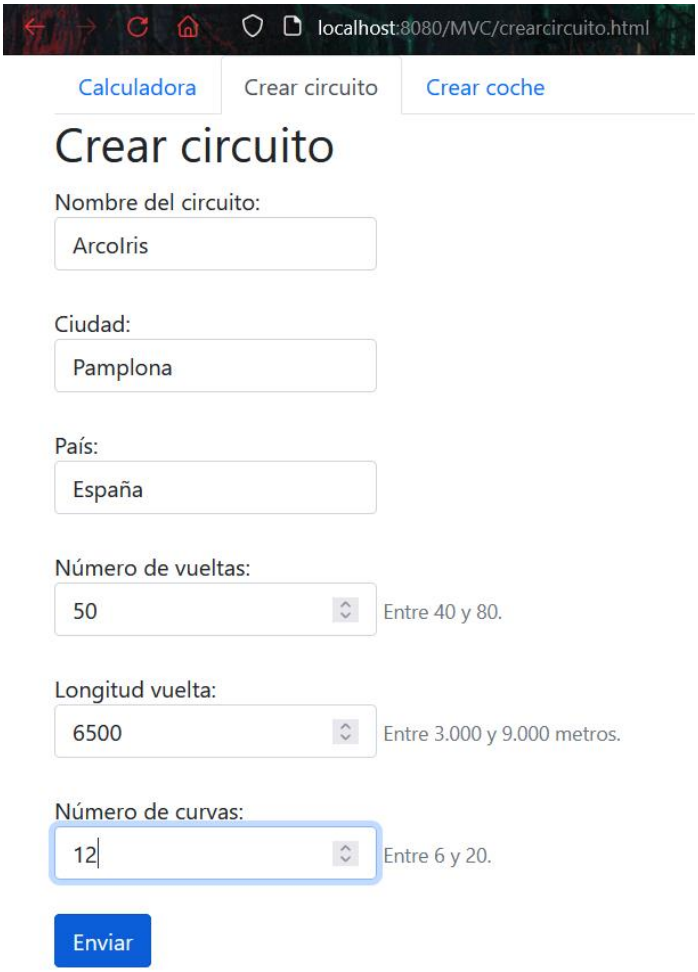


Para crear modificaciones, actualizar e insertar valores a las tablas se crea un archivo denominado **ModeloDatos.java** que además se encarga principalmente de crear las conexiones con la base de datos y de cerrarla cuando se deje de usar. Su estructura y funcionalidad se explica más adelante.

Crear circuito

La aplicación tiene el circuito de Mónaco y el coche Alpine Ar521 por defecto.

Se crea el circuito ArcoIris.



Resultado con insertar un circuito con éxito

Resultado de actualizar un circuito con éxito

Cuando el menú se clica en la opción **Crear circuito** para llamar a **crearcircuito.html** que presenta un formulario con un conjunto de campos (Nombre, Ciudad, País, Numeró de vueltas, Longitud vuelta, Número de curvas) para dar de alta a un circuito. Antes de enviar los datos insertados al servidor, se validan cada uno de los campos aprovechando las herramientas que brinda el HTML5.

```
<form method="post" action="CrearCircuito">
    <p>
        <label class="form-label">Nombre del circuito:
            <input class="form-control" type="text" name="nombre" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
        </label>
    </p>
    <p>
        <label class="form-label">Ciudad:
            <input class="form-control" type="text" name="ciudad" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
        </label>
    </p>
    <p>
        <label class="form-label">País:
            <input class="form-control" type="text" name="pais" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
        </label>
    </p>
    <p>
        <label class="form-label">Número de vueltas:
            <input class="form-control" type="number" name="vueltas" required min="40" max="80">
        </label>
        <span class="form-text">Entre 40 y 80.</span>
    </p>
    <p>
        <label class="form-label">Longitud vuelta:
            <input class="form-control" type="number" name="longitud" required min="3000" max="9000">
        </label>
        <span class="form-text">Entre 3.000 y 9.000 metros.</span>
    </p>
    <p>
        <label class="form-label">Número de curvas:
            <input class="form-control" type="number" name="curvas" required min="6" max="20">
        </label>
        <span class="form-text">Entre 6 y 20.</span>
    </p>
    <p><button class="btn btn-primary">Enviar</button></p>
</form>
```

El **CrearCircuito.java** hace la conexión con la BBDD **sample** y valida los campos enviados comprobando si están rellenos y sus valores se ajustan a los requerimientos. Si todo ha ido bien, éxito, el circuito se añade o se modifica en su tabla, en caso contrario se especifica el error.

En el método **init()** se instancia el objeto **ModeloDatos** para hacer conexión con la BBDD.

```
public class CrearCircuito extends HttpServlet
{
    private ModeloDatos bd;

    @Override
    public void init(ServletConfig config) throws ServletException
    {
        bd = new ModeloDatos();
        bd.abrirConexion();
    }
```

El método **processRequest()** recoge todos el valor de los campos del formulario que da de alta un circuito.

Se comprueba si los campos (nombre, ciudad y país) están vacíos o tienen una longitud superior a 20 (límite establecido). Las vueltas, la longitud y las curvas del circuito tienen que estar entre los valores establecidos.

Hay dos posibilidades:

- Si alguno de los valores de los campos no es válido: da un error (se especificando el motivo).
- Si todos los valores de los campos son válidos: Si el nombre del circuito está en la BBDD se actualiza sus valores sino se añade el circuito.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
{
    HttpSession s = request.getSession(true);

    String nombre = request.getParameter("nombre");
    String ciudad = request.getParameter("ciudad");
    String pais = request.getParameter("pais");
    Integer vueltas = Integer.parseInt(request.getParameter("vueltas"));
    Integer longitud = Integer.parseInt(request.getParameter("longitud"));
    Integer curvas = Integer.parseInt(request.getParameter("curvas"));

    s.setAttribute("tipo", "circuito");
    s.setAttribute("nombre", nombre);

    if (nombre.isEmpty() || ciudad.isEmpty() || pais.isEmpty())
    {
        s.setAttribute("error", "Hay algún campo que se encuentra vacío.");
    }
    else if (nombre.length() > 20 || ciudad.length() > 20 || pais.length() > 20)
    {
        s.setAttribute("error", "Hay algún campo de texto que excede la longitud de 20.");
    }
    else if ((vueltas < 40 || vueltas > 80)
            || (longitud < 3000 || longitud > 9000)
            || (curvas < 6 || curvas > 20))
    {
        s.setAttribute("error", "Hay algún campo numérico que está fuera de rango.");
    }

    if (s.getAttribute("error") != null)
    {
        response.sendRedirect(response.encodeRedirectURL("error.jsp"));
        return;
    }
```

```
    if (bd.existeCircuito(nombre))
    {
        bd.actualizarCircuito(nombre, ciudad, pais, vueltas, longitud, curvas);
        s.setAttribute("accion", "actualizado");
        response.sendRedirect(response.encodeRedirectURL("exito.jsp"));
    }
    else
    {
        bd.insertarCircuito(nombre, ciudad, pais, vueltas, longitud, curvas);
        s.setAttribute("accion", "insertado");
        response.sendRedirect(response.encodeRedirectURL("exito.jsp"));
    }
}
```

Importante: Cerrar la conexión de la Base de Datos en el método **destroy()**.

```
@Override
public void destroy()
{
    bd.cerrarConexion();
    super.destroy();
}
```

Por último, en la lógica de datos se crean cuatro funciones para la base de datos en **ModeloDatos.java**.

Conecta la aplicación con la Base de Datos **sample**, con el usuario y contraseña **app.**

```
public void abrirConexion()
{
    try
    {
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        con = DriverManager.getConnection("jdbc:derby://localhost:1527/sample", "app", "app");
        System.out.println("Se ha conectado");
    }
    catch(ClassNotFoundException | SQLException e)
    {
        System.out.println("No se ha conectado: " + e.getMessage());
    }
}
```

La función **existeCircuito(nombre)** comprueba si el nombre del circuito está en la base de datos.

```
public boolean existeCircuito(String nombre)
{
    boolean existe = false;
    String cad;

    try
    {
        set = con.createStatement();
        rs = set.executeQuery("SELECT * FROM CIRCUITO");

        while (rs.next())
        {
            cad = rs.getString("nombre");
            cad = cad.trim();
            if (cad.compareTo(nombre.trim()) == 0) existe = true;
        }

        rs.close();
        set.close();
    }
    catch(SQLException e)
    {
        System.out.println("No lee de la tabla CIRCUITO: " + e.getMessage());
    }

    return existe;
}
```

La función **actualizarCircuito(nombre, ciudad, país, vueltas, longitud, curvas)** actualiza los valores que se introducen por parámetros en la fila con el nombre del circuito que se especifica.

```
void actualizarCircuito(String nombre, String ciudad, String pais, Integer vueltas, Integer longitud, Integer curvas)
{
    try
    {
        set = con.createStatement();
        ps = con.prepareStatement("UPDATE CIRCUITO SET ciudad=?, pais=?, vueltas=?, longitud=?, curvas=? WHERE nombre = ?");
        ps.setString(1, ciudad);
        ps.setString(2, pais);
        ps.setInt(3, vueltas);
        ps.setInt(4, longitud);
        ps.setInt(5, curvas);
        ps.setString(6, nombre);
        ps.executeUpdate();
        ps.close();
    }
    catch(SQLException e)
    {
        System.out.println("No modifica la tabla CIRCUITO: " + e.getMessage());
    }
}
```

La función **insertarCircuito(nombre, ciudad, país, vueltas, longitud, curvas)** inserta en la Base de Datos el nuevo circuito con los valores que se especifican por parámetros.

```
    public void insertarCircuito(String nombre, String ciudad, String pais, Integer vueltas, Integer longitud, Integer curvas)
    {
        try
        {
            set = con.createStatement();
            set.executeUpdate("INSERT INTO CIRCUITO " + " (nombre, ciudad, pais, vueltas, longitud, curvas)"
                    + " VALUES ('" + nombre + "','" + ciudad + "','" + pais + "'," + vueltas + "," + longitud + "," + curvas + ")");
            rs.close();
            set.close();
        }
        catch(SQLException e)
        {
            System.out.println("No inserta en la tabla CIRCUITO: " + e.getMessage());
        }
    }
```

Crear coche

Para la creación de los coches también se sigue el modelo MVC.

Primero se implementa la siguiente interfaz de usuario:

En dicha interfaz se solicita el nombre del coche y la ganancia de potencia por cada curva mediante un formulario.

Para disponer la interfaz se crea la página HTML **crearcoche.html** mostrada a continuación:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Crear coche</title>
        <meta charset="utf-8">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
    </head>
    <body>
        <div class="container">
            <ul class="nav nav-tabs">
                <li class="nav-item">
                    <a class="nav-link" href="index.jsp">Calculadora</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="crearcircuito.html">Crear circuito</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link active" href="#">Crear coche</a>
                </li>
            </ul>
            <h1>Crear coche</h1>

            <form method="post" action="CrearCoche">
                <p>
                    <label class="form-label">Nombre del coche:
                        <input class="form-control" type="text" name="nombre" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
                    </label>
                </p>
                <p>
                    <label class="form-label">Ganancia de potencia:
                        <input input class="form-control" type="number" name="kwcurva" required min="4" max="10">
                    </label>
                    <span class="form-text">Entre 4 y 10 kW por curva.</span>
                </p>
                <p><button class="btn btn-primary">Enviar</button></p>
            </form>
        </div>
    </body>
</html>
```

En el documento se puede observar que se utiliza la clase del **container** para tener en la barra de navegación la página principal de la calculadora, el documento de creación de los circuitos y una referencia a la página actual.

El formulario contiene las etiquetas de los datos pedidos, los cuadros de texto para escribir y el botón para enviar el formulario.

La página utiliza **Bootstrap** para el estilo al igual que las otras páginas.

En cuanto a la lógica de negocio se crea el servlet **CrearCoche.java**.

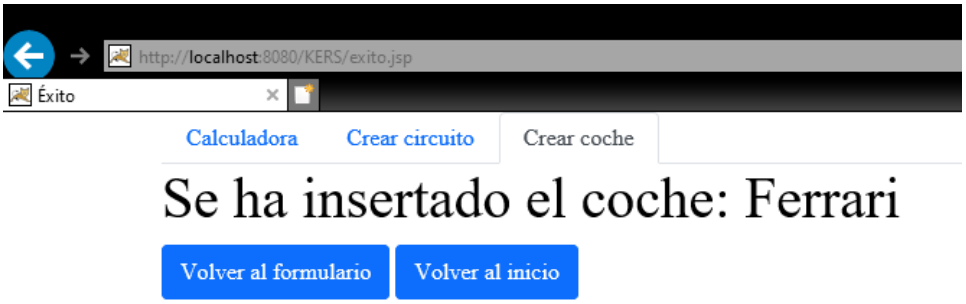
```
public class CrearCoche extends HttpServlet
{
    private ModeloDatos bd;

    @Override
    public void init(ServletConfig config) throws ServletException
    {
        bd = new ModeloDatos();
        bd.abrirConexion();
    }

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        HttpSession s = request.getSession(true);

        String nombre = request.getParameter("nombre");
        Integer kwcurva = Integer.parseInt(request.getParameter("kwcurva"));

        s.setAttribute("tipo", "coche");
        s.setAttribute("nombre", nombre);

        if (nombre.isEmpty())
        {
            s.setAttribute("error", "El campo nombre se encuentra vacío.");
        }
        else if (nombre.length() > 20)
        {
            s.setAttribute("error", "El campo nombre excede la longitud de 20.");
        }

        else if ((kwcurva < 4 || kwcurva > 10))
        {
            s.setAttribute("error", "El campo kW por curva está fuera de rango.");
        }

        if (s.getAttribute("error") != null)
        {
            response.sendRedirect(response.encodeRedirectURL("error.jsp"));
            return;
        }

        if (bd.existeCoche(nombre))
        {
            bd.actualizarCoche(nombre, kwcurva);
            s.setAttribute("accion", "actualizado");
            response.sendRedirect(response.encodeRedirectURL("exito.jsp"));
        }
        else
        {
            bd.insertarCoche(nombre, kwcurva);
            s.setAttribute("accion", "insertado");
            response.sendRedirect(response.encodeRedirectURL("exito.jsp"));
        }
    }

    @Override
    public void destroy()
    {
        bd.cerrarConexion();
        super.destroy();
    }

    HttpServlet methods. Click on the + sign on the left to edit the code.

}
```

En éste al inicio de su ejecución se abre una conexión con la base de datos del sistema.

En el proceso de petición se guarda el nombre del coche y la ganancia de potencia para utilizarlas en una sesión creada. En la sesión se crea un atributo para el tipo de dato, en este caso coche, y para el nombre del coche. Se hacen varias comprobaciones de los datos para ver que se cumplen con las restricciones. En dichas comprobaciones si no se cumple alguna de ellas se crea un atributo de error,

la respuesta de la petición mandada será el jsp **error.jsp** y se terminará la ejecución del servlet. Por último, se comprueba si el nombre del coche ya está registrado en la base de datos en la que está conectada, en caso de estarlo se actualiza con la nueva ganancia de potencia y si es un coche nuevo se inserta; en ambos casos se crea el atributo acción cuyo valor depende de las condiciones anteriores y se mandará la respuesta a la petición con el jsp **exito.jsp**.

El resultado de la ejecución del servlet por el envío del formulario con el nuevo coche "Ferrari" y ganancia de potencia "8" será el siguiente:



Por último, en la lógica de datos se crean tres funciones para la base de datos en **ModeloDatos.java**.

La función **existeCoche(nombre)** comprueba si el nombre del coche está en la base de datos.

```
public boolean existeCoche(String nombre)
{
    boolean existe = false;
    String cad;

    try
    {
        set = con.createStatement();
        rs = set.executeQuery("SELECT * FROM COCHE");

        while (rs.next())
        {
            cad = rs.getString("nombre");
            cad = cad.trim();
            if (cad.compareTo(nombre.trim()) == 0) existe = true;
        }

        rs.close();
        set.close();
    }
    catch(SQLException e)
    {
        System.out.println("No lee de la tabla COCHE: " + e.getMessage());
    }

    return existe;
}
```

La función **actualizarCoche(nombre,kwcurva)** cambia la ganancia de potencia en curva del coche pasado como argumento con el nuevo valor en la base de datos.

```
void actualizarCoche(String nombre, Integer kwcurva)
{
    try
    {
        ps = con.prepareStatement("UPDATE COCHE SET kwcurva=? WHERE nombre = ?");
        ps.setInt(1, kwcurva);
        ps.setString(2, nombre);
        ps.executeUpdate();
        ps.close();
    }
    catch(SQLException e)
    {
        System.out.println("No modifica la tabla COCHE: " + e.getMessage());
    }
}
```

La función **insertarCoche(nombre,kwcurva)** inserta en la base de datos el nuevo coche con el nombre y la ganancia de potencia en curva dados.

```
public void insertarCoche(String nombre, Integer kwcurva)
{
    try
    {
        set = con.createStatement();
        set.executeUpdate("INSERT INTO COCHE " + " (nombre, kwcurva) VALUES ('" + nombre + "'," + kwcurva + ")");
        rs.close();
        set.close();
    }
    catch(SQLException e)
    {
        System.out.println("No inserta en la tabla COCHE: " + e.getMessage());
    }
}
```

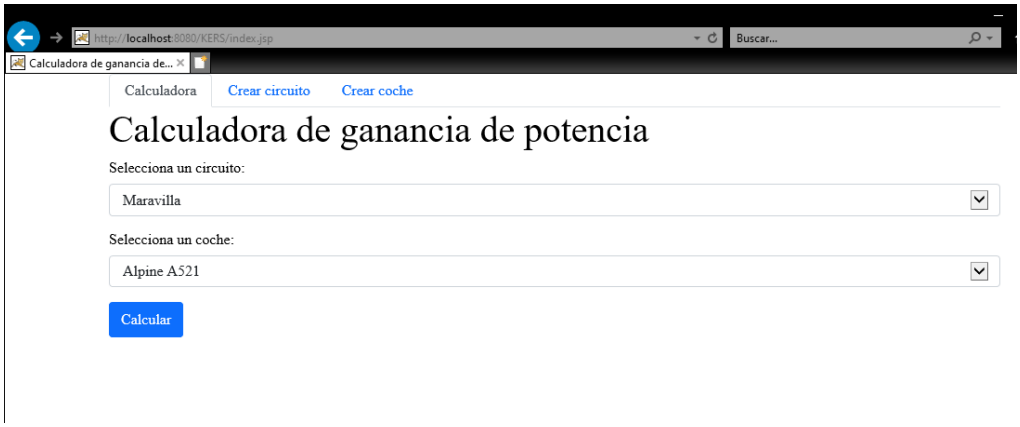
Cálculo de ganancia de potencia

Para el cálculo de ganancia de potencia se utiliza como interfaz el jsp **index.jsp**.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Calculadora de ganancia de potencia</title>
        <meta charset="utf-8">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
    </head>
    <body>
        <%@ page import="java.sql.*" %>
        <%
            Connection c = DriverManager.getConnection("jdbc:derby://localhost:1527/sample","app","app");
            PreparedStatement pstm1 = c.prepareStatement("SELECT nombre FROM CIRCUITO");
            PreparedStatement pstm2 = c.prepareStatement("SELECT nombre FROM COCHE");
            ResultSet circuitos = pstm1.executeQuery();
            ResultSet coches = pstm2.executeQuery();
        %>
        <div class="container">
            <ul class="nav nav-tabs">
                <li class="nav-item">
                    <a class="nav-link active" href="#">Calculadora</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="crearcircuito.html">Crear circuito</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="crearcoche.html">Crear coche</a>
                </li>
            </ul>

            <h1>Calculadora de ganancia de potencia</h1>

            <form method="post" action="CalcularGanancia">
                <p>
                    <label class="form-label">Selecciona un circuito:</label>
                    <select class="form-control" name="circuitos" required>
                        <% while (circuitos.next()) { %>
                        <option value="<%= circuitos.getString(1) %>"><%= circuitos.getString(1) %></option>
                        <% } %>
                    </select>
                </p>
                <p>

                    <label class="form-label">Selecciona un coche:</label>
                    <select class="form-control" name="coches" required>
                        <% while (coches.next()) { %>
                        <option value="<%= coches.getString(1) %>"><%= coches.getString(1) %></option>
                        <% } %>
                    </select>
                </p>
                <p><button class="btn btn-primary">Calcular</button></p>
                </div>
            </form>
        </div>
    </body>
</html>
```

En dicho jsp se accederá a la base de datos y se guardará los circuitos y coches creados. Al igual que en las otras páginas se crea el menú de navegación con la calculadora, la creación de los circuitos y de los coches. El formulario solicita el circuito y el coche de los que se quiere calcular la ganancia de potencia. Se crean dos select para acceder a las variables donde se almacenaban los datos de los circuitos y los coches y se muestran sus nombres.

La interfaz se muestra a continuación:



El envío del formulario de cálculo lo utilizará el servlet **CalcularGanancia.java.**

```
public class CalcularGanancia extends HttpServlet
{
    private ModeloDatos bd;

    @Override
    public void init(ServletConfig config) throws ServletException
    {
        bd = new ModeloDatos();
        bd.abrirConexion();
    }

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        HttpSession s = request.getSession(true);

        String circuito = request.getParameter("circuitos");
        String coche = request.getParameter("coches");

        Integer kwcurva = bd.obtenerCoche(coche);

        Integer[] array = bd.obtenerCircuito(circuito);
        Integer vueltas = array[0];
        Integer curvas = array[1];

        Integer ganancia = kwcurva * vueltas * curvas;

        s.setAttribute("ganancia", ganancia);
        s.setAttribute("kwcurva", kwcurva);
        s.setAttribute("vueltas", vueltas);
        s.setAttribute("curvas", curvas);
        response.sendRedirect(response.encodeRedirectURL("resultado.jsp"));
    }

    @Override
    public void destroy()
    {
        bd.cerrarConexion();
        super.destroy();
    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
```

En este servlet al igual que los anteriores se abre una conexión con la base de datos y se crea una sesión. En la petición se guarda el coche y el circuito introducidos. Del circuito se almacena el número de vueltas y de curvas en un array mediante la función de la base de datos **obtenerCircuito(circuito)** y del coche la ganancia de potencia por curva con la función **obtenerCoche(coche)**. Se realiza el cálculo de la ganancia con las variables descritas anteriormente y se crean atributos en la sesión para la ganancia total, la ganancia de potencia por curva, el número de vueltas y el número de curvas. Por último, se envía la respuesta a la petición ejecutando el jsp **resultado.jsp**.

Este jsp tendrá la siguiente implementación:

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Resultado</title>
        <meta charset="utf-8">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
    </head>
    <body>
        <%
            Integer ganancia = (Integer) session.getAttribute("ganancia");
            Integer kwcurva = (Integer) session.getAttribute("kwcurva");
            Integer vueltas = (Integer) session.getAttribute("vueltas");
            Integer curvas = (Integer) session.getAttribute("curvas");
        %>

        <div class="container">
            <ul class="nav nav-tabs">
                <li class="nav-item">
                    <a class="nav-link active" href="#">Calculadora</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="crearcircuito.html">Crear circuito</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="crearcoche.html">Crear coche</a>
                </li>
            </ul>

            <h1>Resultado de la ganancia</h1>

            <p>El resultado de la ganancia es: <strong><%= ganancia %></strong></p>
            <p>
                Se ha calculado a partir de los siguientes datos:
                <ul>
                    <li>El coche gana <%= kwcurva %> kW de potencia por cada curva</li>
                    <li>El circuito tiene un total de <%= vueltas %> vueltas</li>
                    <li>El circuito tiene <%= curvas %> curvas por cada vuelta</li>
                </ul>
            </p>

            <a class="btn btn-primary" href="index.jsp">Realizar otro cálculo de ganancia</a>
        </div>
    </body>
```

De la sesión creada en el servlet se guardan sus atributos con todos los valores generados. Se crea el menú de navegación y se muestran los datos del resultado de la ganancia con todos los atributos de la sesión.

Por último, en **ModeloDatos.java** se crean las funciones mencionadas anteriormente.

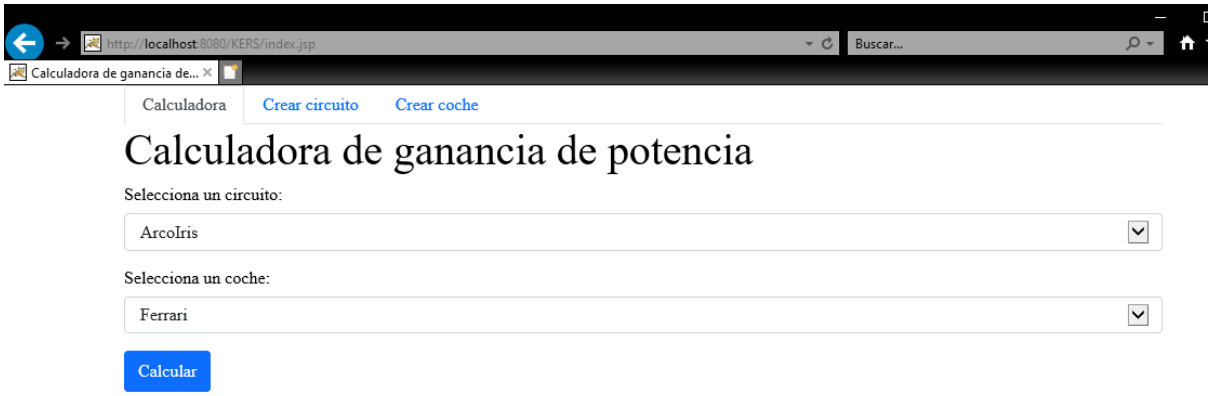
La función **obtenerCircuito(nombre)** devuelve el número de vueltas y curvas dado el nombre de un circuito que esté registrado en la base de datos.

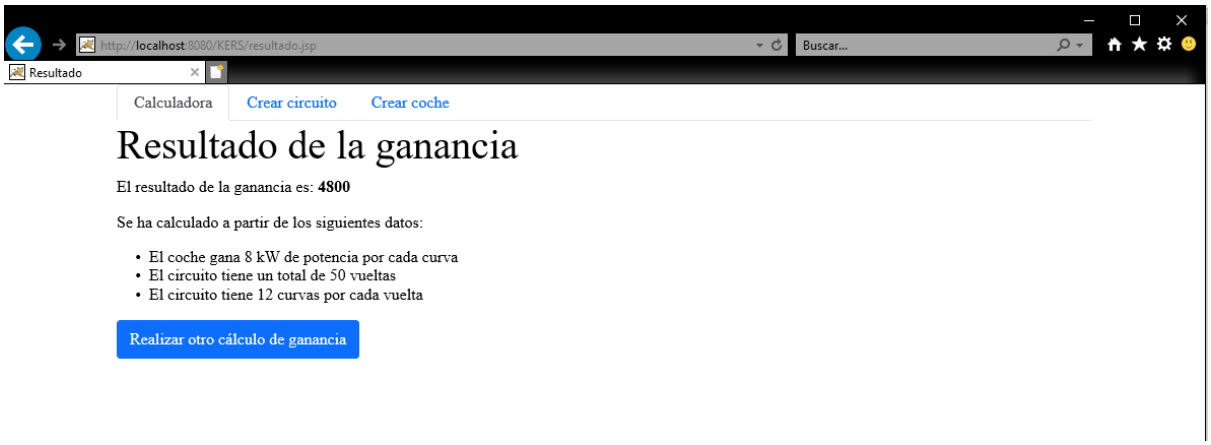
```
public Integer[] obtenerCircuito(String nombre)
{
    Integer[] array = new Integer[2];
    try
    {
        set = con.createStatement();
        rs = set.executeQuery("SELECT vueltas, curvas FROM CIRCUITO WHERE NOMBRE = '" + nombre + "'");
        rs.next();
        array[0] = rs.getInt(1);
        array[1] = rs.getInt(2);
        rs.close();
        set.close();
    }
    catch(SQLException e)
    {
        System.out.println("No lee de la tabla CIRCUITO: " + e.getMessage());
    }

    return array;
}
```

La función **obtenerCoche(nombre)** devuelve la ganancia de potencia por curva dado el nombre de un coche que esté registrado en la base de datos.

```
public Integer obtenerCoche(String nombre)
{
    Integer kwcurva = 0;
    try
    {
        set = con.createStatement();
        rs = set.executeQuery("SELECT kwcurva FROM COCHE WHERE NOMBRE = '" + nombre + "'");
        rs.next();
        kwcurva = rs.getInt(1);
        rs.close();
        set.close();
    }
    catch(SQLException e)
    {
        System.out.println("No lee de la tabla COCHE: " + e.getMessage());
    }

    return kwcurva;
}
```

El cálculo de ganancia de potencia con el coche “Ferrari” y circuito “ArcoIris” daría el siguiente resultado.





Comprobación de los datos en el cliente y servidor

Validación en el lado del cliente:

La comprobación se ha llevado a cabo en el fichero HTML, de una forma más simple en comparación con JavaScript, permitiendo el uso de Bootstrap sin necesidad de modificar el archivo js.

En la imagen siguiente se puede apreciar una validación en Crear Circuito, donde se especifica que los atributos son "required", es decir, requeridos, con type="number" o type="text" se realiza la validación de tipos.

Si se da el caso de que es un tipo texto, hay añadida una expresión regular que permite el uso de todos los caracteres alfanuméricos y vocales acentuadas, así como la ñ y Ñ.

También se puede observar en la línea marcada una validación del rango de un entero.

Todas las validaciones forman parte de la clase form-control, siendo este un diseño modular.

Crear circuito:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Crear circuito</title>
        <meta charset="utf-8">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
    </head>
    <body>
        <div class="container">
            <ul class="nav nav-tabs">
                <li class="nav-item">
                    <a class="nav-link" href="index.jsp">Calculadora</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link active" href="#">Crear circuito</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="crearcoche.html">Crear coche</a>
                </li>
            </ul>

            <h1>Crear circuito</h1>

            <form method="post" action="CrearCircuito">
                <p>
                    <label class="form-label">Nombre del circuito:
                        <input class="form-control" type="text" name="nombre" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
                    </label>
                </p>
                <p>
                    <label class="form-label">Ciudad:
                        <input class="form-control" type="text" name="ciudad" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
                    </label>
                </p>
                <p>
                    <label class="form-label">País:
                        <input class="form-control" type="text" name="pais" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
                    </label>
                </p>
                <p>
                    <label class="form-label">Número de vueltas:
                        <input class="form-control" type="number" name="vueltas" required min="40" max="80">
                    </label>
                    <span class="form-text">Entre 40 y 80.</span>
                </p>
                <p>
                    <label class="form-label">Longitud vuelta:
                        <input class="form-control" type="number" name="longitud" required min="3000" max="9000">
                    </label>
                    <span class="form-text">Entre 3.000 y 9.000 metros.</span>
                </p>
                <p>
                    <label class="form-label">Número de curvas:
                        <input class="form-control" type="number" name="curvas" required min="6" max="20">
                    </label>
                    <span class="form-text">Entre 6 y 20.</span>
                </p>
                <p><button class="btn btn-primary">Enviar</button></p>
            </form>
        </div>
    </body>
```

Crear Coche:

En cuanto a crear coche, las validaciones son iguales a las ya mencionadas.

```html
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Crear coche</title>
        <meta charset="utf-8">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
    </head>
    <body>
        <div class="container">
            <ul class="nav nav-tabs">
                <li class="nav-item">
                    <a class="nav-link" href="index.jsp">Calculadora</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="crearcircuito.html">Crear circuito</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link active" href="#">Crear coche</a>
                </li>
            </ul>
            <h1>Crear coche</h1>

            <form method="post" action="CrearCoche">
                <p>
                    <label class="form-label">Nombre del coche:
                        <input class="form-control" type="text" name="nombre" required maxlength="20" pattern="^[A-Za-z0-9ñÑáéíóúü ]*$">
                    </label>
                </p>
                <p>
                    <label class="form-label">Ganancia de potencia:
                        <input input class="form-control" type="number" name="kwcurva" required min="4" max="10">
                    </label>
                    <span class="form-text">Entre 4 y 10 kW por curva.</span>
                </p>
                <p><button class="btn btn-primary">Enviar</button></p>
            </form>
        </div>
    </body>
</html>
```

Validación en el lado del servidor:

El archivo **éxito.jsp** muestra un mensaje con el resultado de la operación. Se puede ver la acción (insertar o actualizar), tipo (circuito o coche) y el nombre que se especificó.

Hay dos botones: **Volver al formulario** y **Volver al inicio**.

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Éxito</title>
        <meta charset="utf-8">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
    </head>
    <body>
        <%
            String tipo = (String) session.getAttribute("tipo");
            String nombre = (String) session.getAttribute("nombre");
            String accion = (String) session.getAttribute("accion");
        %>

        <div class="container">
            <ul class="nav nav-tabs">
                <li class="nav-item">
                    <a class="nav-link" href="index.jsp">Calculadora</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link <% if (tipo.equals("circuito")) out.print(" active\" href=\"crearcircuito.html\""); else out.print("\" href=\"#\""); %>>
                    Crear circuito</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link <% if (tipo.equals("coche")) out.print(" active\" href=\"crearcoche.html\""); else out.print("\" href=\"#\""); %>>
                    Crear coche</a>
                </li>
            </ul>

            <h1>Se ha <%= accion %> el <%= tipo %>: <%= nombre %></h1>

            <a class="btn btn-primary" href="crear<%= tipo %>.html">Volver al formulario</a>
            <a class="btn btn-primary" href="index.jsp">Volver al inicio</a>
        </div>
    </body>
</html>
```

El archivo **error.jsp** especifica el tipo (coche o circuito) y el mensaje de error personalizado que se creó.

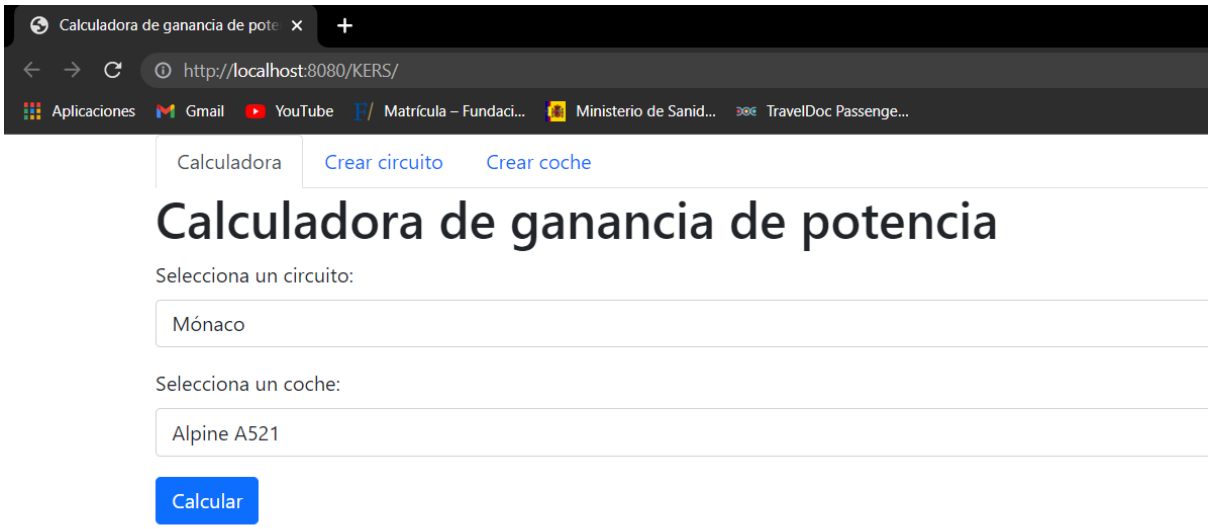
Hay dos botones: **Volver al formulario** y **Volver al inicio**.

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Error</title>
        <meta charset="utf-8">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
    </head>
    <body>
        <%
            String tipo = (String) session.getAttribute("tipo");
            String error = (String) session.getAttribute("error");
        %>

        <div class="container">
            <h1>Error al insertar <%= tipo %></h1>
            <p class="alert alert-danger"><%= error %></p>

            <a class="btn btn-primary" href="crear<%= tipo %>.html">Volver al formulario</a>
            <a class="btn btn-primary" href="index.jsp">Volver al inicio</a>
        </div>
    </body>
</html>
```

La funcionalidad que otorgan ambos ficheros JSP es la verificación en el servidor, que sumada a la verificación del cliente se obtiene el factor de doble verificación cliente-servidor.

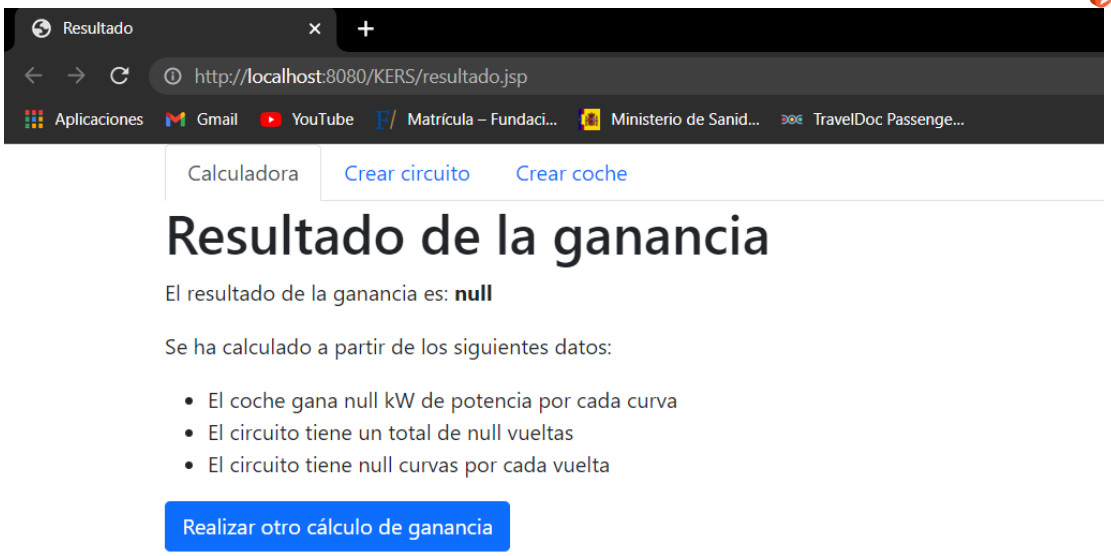
Manual del usuario

Al ejecutar el proyecto la primera ventana que se abre, la considerada como página principal es la **Calculadora de ganancia de potencia** donde permite elegir de la lista de circuitos y coches previamente creados con el fin de empleando dichos valores poder calcular la ganancia que tendrá el coche para el total de vueltas del circuito seleccionado.



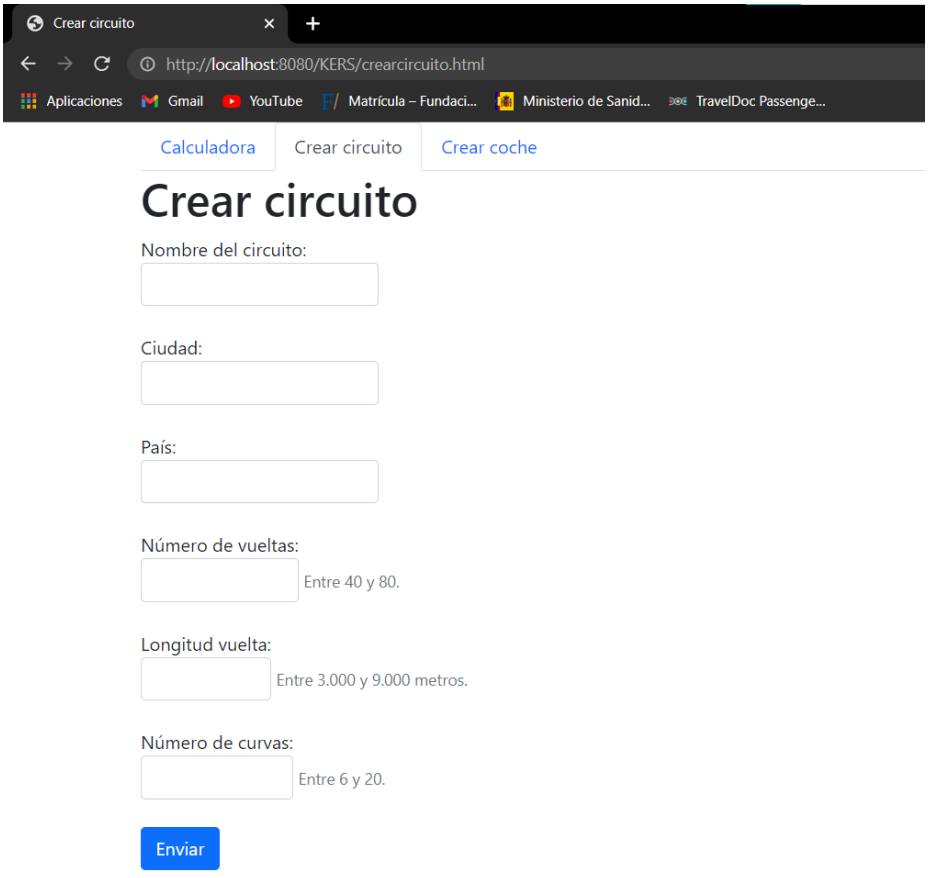
Seleccionando en cada campo se despliegan los listas con las opciones creadas anteriormente donde se elige una única opción de cálculo.

Una vez elegidos se pulsa **calcular** y se redirecciona a otra ventana que muestra el resultado, y además indica sobre qué valores concretos se ha calculado, en base a las opciones elegidas de las listas.

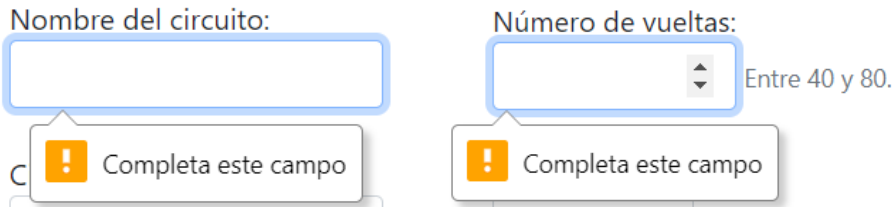


En el caso de pulsar el botón **Realizar otro cálculo de ganancia** se redirecciona a la página anterior y se puede repetir el proceso.

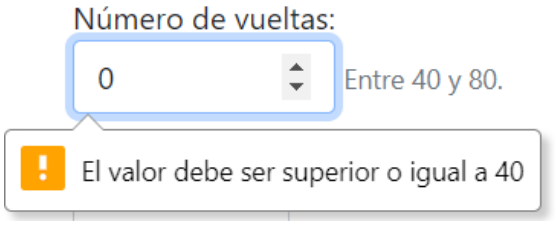
Otra opción posible es seleccionar crear un circuito o crear un coche. El primero se consigue pulsando **Crear circuito** que lleva al usuario al formulario encargado de dicho proceso. Dicho formulario está compuesto por una serie de campos a completar, los tres primeros con entrada de texto, mientras que los tres siguientes son números comprendidos entre ciertos rangos de valores.



En el caso de que el usuario intente enviar el formulario sin haber completado algún campo, bien sea de texto o numérico, el sistema le avisará mostrando un pequeño mensaje emergente de que debe completar el campo en cuestión.



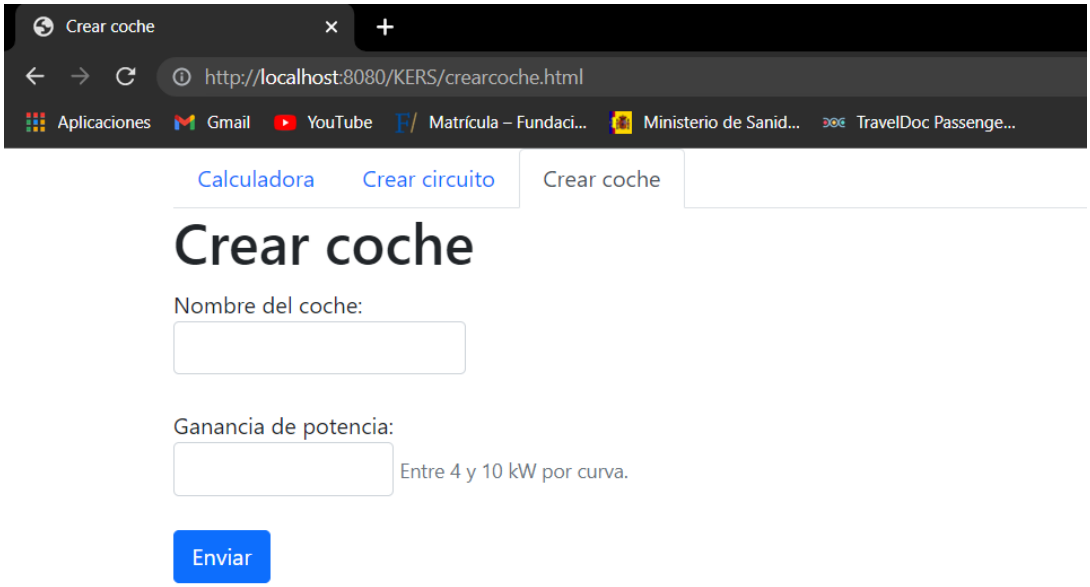
También se realizan comprobaciones sobre los valores numéricos para comprobar que estos pertenecen al rango indicado, de manera que si no se cumple, se pueda hacer saber al usuario.



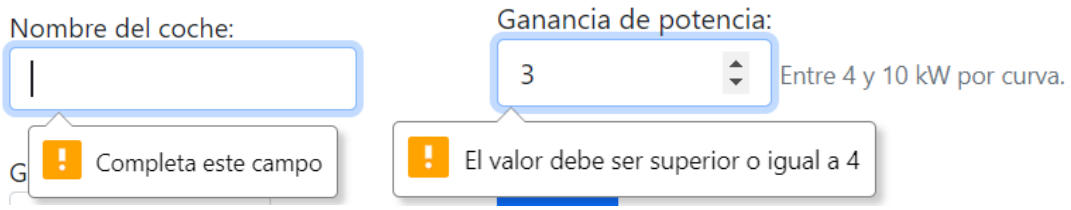
Tras completar los campos si se pulsa el botón **Enviar** se mostrará por pantalla que se ha insertado dicho circuito, y al mismo tiempo indicará al usuario las opciones de **volver al formulario** y poder crear otro circuito o bien **volver al inicio** y calcular la ganancia.

En el caso de que no se seleccione ninguna de las opciones anteriores y se elija **Crear coche** del menú de la parte superior, se mostrará por pantalla el formulario encargado de dicha acción.

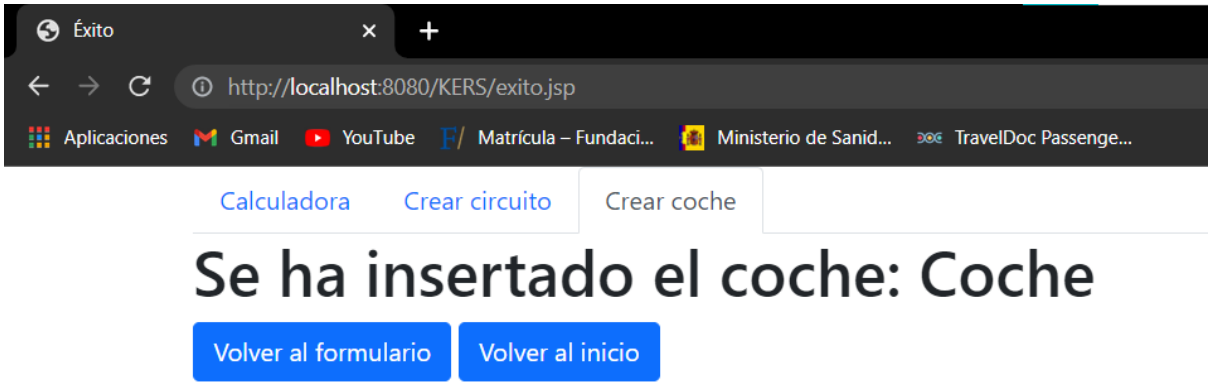
En él se puede ver los dos campos a completar junto con la pequeña indicación a un lado de este.



Como se ha mencionado en el formulario anterior este también tiene sus comprobaciones de valores de entrada, de manera que el usuario no puede dejar ningún campo vacío y además debe cumplir con las restricciones de rango en aquellos campos numéricos.



Tras completar los datos y pulsar el botón **Enviar** se muestra por pantalla un mensaje indicando que se ha insertado el coche en cuestión y da al usuario la opción de volver a declarar un nuevo coche o de volver a la página de inicio.



Conclusiones

La realización de esta práctica ha permitido una correcta implementación del modelo Vista-Controlador: modelo (manejo de datos), la vista (diseño y presentación) y el controlador (comandos modelos y vistas), el patrón por excelencia cuando se trata con páginas web. El patrón de por si permite la reutilización de código y sobre todo permite la creación de una aplicación de mayor calidad.