Práctica 7.10: *Arquitectura MVC*

En esta práctica abordaremos el empleo del patrón **Modelo Vista Controlador MVC**.

Como introducción, podemos hablar de los **patrones de software** como soluciones reutilizables en el diseño de una aplicación. En nuestro caso la utilización del patrón **MVC** proporciona una arquitectura uniforme que permite una fácil expansión, mantenimiento y modificación de una aplicación.

El principal objetivo de la arquitectura **MVC** es aislar tanto los datos de la aplicación como el estado (**modelo**) de la misma, del mecanismo utilizado para representar (**vista**) dicho estado, así como para modularizar esta vista y modelar la transición entre estados del modelo (**controlador**).

Las aplicaciones **MVC** se dividen en tres grandes áreas funcionales:

- **Vista**: Presentación de los datos.
- **Controlador**: Atenderá las peticiones y toma de decisiones de la aplicación.
- **Modelo**: La lógica del negocio y los datos asociados con la aplicación.

El propósito del **MVC** es aislar los cambios. Es una arquitectura preparada para los cambios, que desacopla datos y lógica de negocio de la lógica de presentación, permitiendo la actualización y desarrollo independiente de cada uno de los citados componentes.

Las aplicaciones de **MVC** pueden ser implementadas con *J2EE* utilizando *JSP* para las **vistas**, *servlets* como **controladores** y *JavaBeans* para el **modelo**, siendo los *JavaBeans* componentes de código reutilizables en cualquier aplicación, Figura 1.

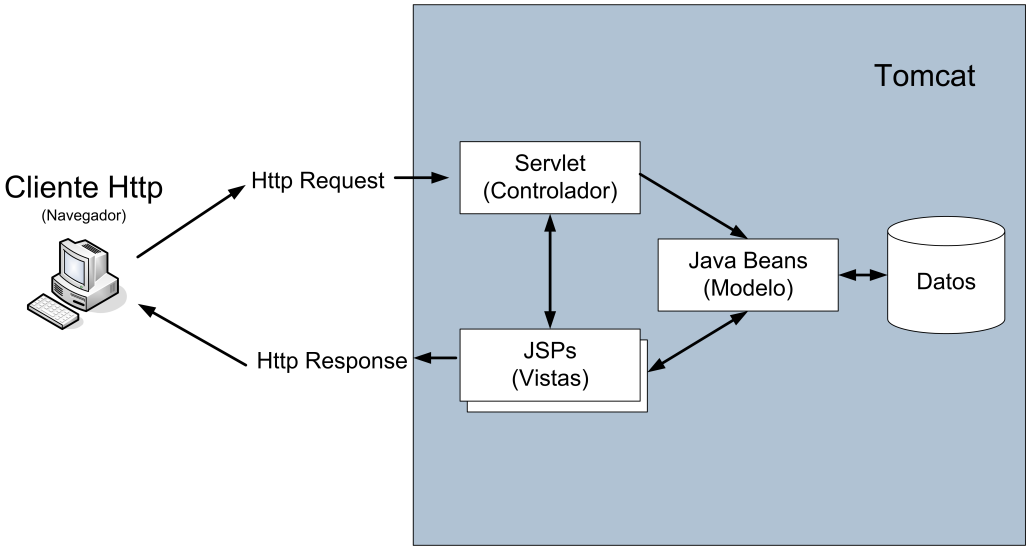

Figura 1: Arquitectura MVC

1. **Diseño de la aplicación LibreríaMVC**.

   1.1. Como primer paso vamos a diseñar la aplicación utilizando el patrón **MVC**, Figura 2.

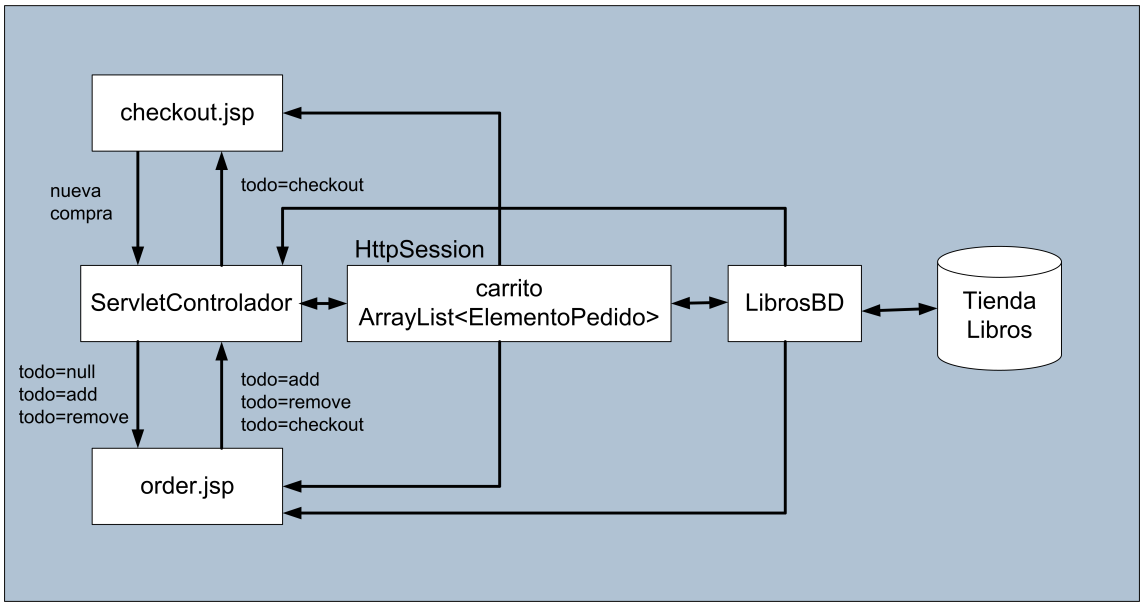

Figura 2: Diseño LibreríaMVC

   1.2. Observa qué elementos se corresponden con cada uno de los componentes del patrón.

   - **Vista**: La página **order.jsp** se encarga de mostrar la página de compras mientras que la página **checkout.jsp** se encargará de mostrar el resumen de la compra.
   - **Controlador**: El *servlet* **ServletControlador** se encarga de atender las peticiones y dirigir la aplicación.
   - **Modelo**: La lógica del negocio está representada por la clase **ElementoPedido.java**, que representa un elemento del pedido y **LibrosBD.java** que se encarga de la comunicación con la base de datos.

   1.3. Observa también cómo en el diagrama se muestra la variable de sesión **carrito**, que permitirá que cada cliente guarde los libros que va seleccionando.

2. **Codificación de la aplicación LibreríaMVC**.

   2.1. Crea un nuevo *Dynamic Web Project* llamado **LibreriaMVC**.

   2.2. Pulsando con el botón derecho del ratón sobre **Java Resources\src** añadimos la clase **LibrosBD.java** con el siguiente código:

```
package practica;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;

/**
 * LibrosBD
 * Encapsula la comunicación con la base de datos
 * Almacena títulos, autores y precios en tres arrays
 */
public class LibrosBD {

   private static final int MAX_SIZE=5;
   private static String[] titulos=new String[MAX_SIZE];
   private static String[] autores=new String[MAX_SIZE];
   private static float[] precios=new float[MAX_SIZE];

   public static void cargarDatos(){
   Connection conn = null;
   Statement stmt = null;
   try {
   //Paso 1: Cargar el driver JDBC.
   Class.forName("com.mysql.jdbc.Driver").newInstance();
   // Paso 2: Conectarse a la Base de Datos utilizando la clase Connection
   String userName="root";
   String password="despliegue";
   //URL de la base de datos(equipo, puerto, base de datos)
   String url="jdbc:mysql://localhost/TiendaLibros";
   conn = DriverManager.getConnection(url, userName, password);
   // Paso 3: Crear sentencias SQL, utilizando objetos de tipo Statement
   stmt = conn.createStatement();
   // Paso 4: Ejecutar las sentencias SQL a través de los objetos Statement
   String sqlStr = "select * from libros;";
   ResultSet rset = stmt.executeQuery(sqlStr);
   // Paso 5: Procesar el conjunto de registros resultante utilizando ResultSet
   int count = 0;
   while(rset.next()) {
   titulos[count]= rset.getString("autor");
       autores[count]=rset.getString("titulo");
       precios[count]=(float)rset.getDouble("precio");
        count++;
   }
      // Cerramos el resto de recursos
   if (stmt != null) stmt.close();
   if (conn != null) conn.close();
   } catch (Exception ex){
     ex.printStackTrace();
   }
   }

   /** Devuelve el número de libros */
   public static int tamanyo() {
```

```
      return titulos.length;
   }

   /** Devuelve el título del libro  idLibro*/
   public static String getTitulo(int idLibro) {
      return titulos[idLibro];
   }

   /** Devuelve el autor del libro  idLibro */
   public static String getAutor(int idLibro) {
      return autores[idLibro];
   }

   /** Devuelve el precio del libro idLibro */
   public static float getPrecio(int idLibro) {
      return precios[idLibro];
   }
}
```

2.3. Esta clase se encarga de cargar en unos *arrays* los datos de los libros existentes en la base de datos.

2.4. De igual forma añadimos la clase **ElementoPedido.java** con el siguiente código:

```
package practica;

/**
 * Representa un elemento del pedido
 * Incluye identificador del libro y cantidad
 */

public class ElementoPedido {
   private int idLibro;
   private int cantidad;

   public ElementoPedido(int idLibro, int cantidad) {
      this.idLibro = idLibro;
      this.cantidad = cantidad;
   }

   public int getIdLibro() {
      return idLibro;
   }

   public void setIdLibro(int idLibro) {
      this.idLibro = idLibro;
   }

   public int getCantidad() {
      return cantidad;
   }
```

```
   public void setCantidad(int cantidad) {
      this.cantidad = cantidad;
   }

   public String getAutor() {
      return LibrosBD.getAutor(idLibro);
   }

   public String getTitulo() {
      return LibrosBD.getTitulo(idLibro);
   }

   public float getPrecio() {
      return LibrosBD.getPrecio(idLibro);
   }
}
```

2.5. Los objetos creados con esta clase representaran los libros que estan en el carrito.

2.6. Pulsando con el botón derecho del ratón sobre **Java Resources\src** añadiremos el *servlet* **ServletControlador.java** con el siguiente código:

```
package practica;


import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletControlador extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @Override
   public void init(ServletConfig conf) throws ServletException {
      super.init(conf);
      LibrosBD.cargarDatos();
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      doPost(request, response);
   }

   @Override
   protected void doPost(HttpServletRequest request,
         HttpServletResponse response) throws ServletException, IOException {
      // Recupera la sesión actual o crea una nueva si no existe.
      HttpSession session = request.getSession(true);
```

```
// Recupera el carrito de la sesión actual
List<ElementoPedido> elCarrito =
                     (ArrayList<ElementoPedido>) session.getAttribute("carrito");

// Determina a que página jsp redirigirá
String nextPage = "";
String todo = request.getParameter("todo");

if (todo == null) {
   // Primer acceso - redirección a order.jsp
   nextPage = "/order.jsp";
} else if (todo.equals("add")) {
   // Mandado por order.jsp con los parámetros idLibro y cantidad.
   // crea un elementoPedido y lo añade al carrito
   ElementoPedido nuevoElementoPedido = new ElementoPedido(
           Integer.parseInt(request.getParameter("idLibro")),
           Integer.parseInt(request.getParameter("cantidad")));
   if (elCarrito == null) { // el carrito está vacío
      elCarrito = new ArrayList<>();
      elCarrito.add(nuevoElementoPedido);
      // enlaza el carrito con la sesión
      session.setAttribute("carrito", elCarrito);
   } else {
      // Comprueba si el libro está en el carrito
      // si lo está actualiza la cantidad
   // si no lo añade
      boolean encontrado = false;
      Iterator iter = elCarrito.iterator();
      while (!encontrado && iter.hasNext()) {
         ElementoPedido unElementoPedido = (ElementoPedido)iter.next();
         if (unElementoPedido.getIdLibro() == nuevoElementoPedido.getIdLibro())
         {
          unElementoPedido.setCantidad(unElementoPedido.getCantidad()
                  + nuevoElementoPedido.getCantidad());
            encontrado = true;
         }
      }
      if (!encontrado) { // Lo añade al carrito
         elCarrito.add(nuevoElementoPedido);
      }
   }
   // Vuelve a order.jsp para más pedidos
   nextPage = "/order.jsp";
} else if (todo.equals("remove")) {
   // Enviado por order.jsp con el parámetro indiceElemento
   // Borra el elemento indiceElemento del carrito
   int indiceCarrito = Integer.parseInt(request.getParameter("indiceElemento"));
   elCarrito.remove(indiceCarrito);
   // Vuelve a order.jsp para más pedidos
```

```
            nextPage = "/order.jsp";
        } else if (todo.equals("checkout")) {
            // Enviado por order.jsp.
            // Calcula el precio total de todos los elementos del carrito
            float precioTotal = 0;
            int cantidadTotalOrdenada = 0;
            for (ElementoPedido item: elCarrito) {
               float precio = item.getPrecio();
               int cantidadOrdenada = item.getCantidad();
               precioTotal += precio * cantidadOrdenada;
               cantidadTotalOrdenada += cantidadOrdenada;
            }
            // Formatea el precio con dos decimales
            StringBuilder sb = new StringBuilder();
            Formatter formatter = new Formatter(sb);
            formatter.format("%.2f", precioTotal);
            // Coloca el precioTotal y la cantidadTotal en el request
            request.setAttribute("precioTotal", sb.toString());
            request.setAttribute("cantidadTotal", cantidadTotalOrdenada + "");
            // Redirige a checkout.jsp
            nextPage = "/checkout.jsp";
        }

        ServletContext servletContext = getServletContext();
        RequestDispatcher requestDispatcher =
                          servletContext.getRequestDispatcher(nextPage);
        requestDispatcher.forward(request, response);
    }

}
```

2.7. Observa las últimas líneas de código, utiliza la interface *RequestDispatcher* que se utiliza para encaminar solicitudes a otros *Servlet* o *JSP* de la misma aplicación, permitiendo delegar el procesamiento de solicitud respuesta en ellos.

2.8. Para que el *servlet* sea accesible añade la siguiente información al descriptor de despliegue:

```
<servlet>
   <servlet-name>Controlador</servlet-name>
   <servlet-class>practica.ServletControlador</servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>Controlador</servlet-name>
   <url-pattern>/shopping</url-pattern>
 </servlet-mapping>
 <session-config>
   <session-timeout>30</session-timeout>
 </session-config>
 <welcome-file-list>
   <welcome-file>shopping</welcome-file>
 </welcome-file-list>
```

2.9. Añade sobre **WebContent** la página **order.jsp** :

```
<%-- Página de Ordenes  --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page session="true" import="java.util.*, practica.*" %>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Pedido a la Librería MVC</title>
   </head>
   <body>
      <h2>Librería MVC</h2>
      <hr /><br />

      <p><strong>Elige un libro y la cantidad:</strong></p>
      <form name="AnyadirForm" action="shopping" method="POST">
         <input type="hidden" name="todo" value="add">
         Título: <select name=idLibro>
            <%
            // Scriptlet 1: Carga los libros en el select.
            for (int i = 0; i < LibrosBD.tamanyo(); i++) {
               out.println("<option value='" + i + "'>");
               out.println(LibrosBD.getTitulo(i) + " | " + LibrosBD.getAutor(i)
                       + " | " + LibrosBD.getPrecio(i));
               out.println("</option>");
            }
            %>
         </select>
           Cantidad: <input type="text" name="cantidad" size="10" value="1">
         <input type="submit" value="Añadir a la cesta">
      </form>
      <br /><hr /><br />

      <%
      // Scriptlet 2: Chequea si la cesta esta vacia.
      List<ElementoPedido> cesta =
                  (List<ElementoPedido>) session.getAttribute("carrito");
      if (cesta != null && cesta.size() > 0) {
      %>
      <p><strong>Tu cesta contiene:</strong></p>
      <table border="1" cellspacing="0" cellpadding="5">
         <tr>
            <th>Título</th>
            <th>Autor</th>
            <th>Precio</th>
            <th>Cantidad</th>
            <th> </th>
         </tr>
         <%
```

```
            // Scriptlet 3: Muestra los libros del carrito.
            for (int i = 0; i < cesta.size(); i++) {
               ElementoPedido elementoPedido = cesta.get(i);
            %>
            <tr>
               <form name="borrarForm" action="shopping" method="POST">
                  <input type="hidden" name="todo" value="remove">
                  <input type="hidden" name="indiceElemento" value="<%= i %>">
                  <td><%= elementoPedido.getTitulo() %></td>
                  <td><%= elementoPedido.getAutor() %></td>
                  <td align="right"><%= elementoPedido.getPrecio() %></td>
                  <td align="right"><%= elementoPedido.getCantidad() %></td>
                  <td><input type="submit" value="Eliminar de la cesta"></td>
               </form>
            </tr>
            <%
            }
            %>
         </table><br />

         <form name="checkoutForm" action="shopping" method="POST">
            <input type="hidden" name="todo" value="checkout">
            <input type="submit" value="Checkout">
         </form>
         <%
         }  // if
         %>
      </body>
   </html>
```

2.10. Añade sobre **WebContent** la página **checkout.jsp**:

```
<%-- Página de Checkout   --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page session="true" import="java.util.*, practica.*" %>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Libreria MVC Checkout</title>
   </head>
   <body>
      <h2>Libreria MVC - Checkout</h2>
      <hr /><br />

      <p><strong>Has comprado los siguientes libros:</strong></p>
      <table border="1" cellspacing="0" cellpadding="5">
         <tr>
            <th>Título</th>
            <th>Autor</th>
            <th>Precio</th>
```

```
            <th>Cantidad</th>
         </tr>
         <%
         // Scriptlet 1: Muestra los elementos del carrito
         List<ElementoPedido> cesta =
                              (List<ElementoPedido>) session.getAttribute("carrito");
         for (ElementoPedido item : cesta) {
         %>
         <tr>
            <td><%= item.getTitulo()%></td>
            <td><%= item.getAutor()%></td>
            <td align="right"><%= item.getPrecio()%></td>
            <td align="right"><%= item.getCantidad()%></td>
         </tr>
         <%
         } // for
         session.invalidate();
         %>
         <tr>
            <th align="right" colspan="2">Total</th>
            <th align="right"><%= request.getAttribute("precioTotal")%></th>
            <th align="right"><%= request.getAttribute("cantidadTotal")%></th>
         </tr>
      </table>
      <br />

      <a href="shopping">Pulsa aquí para comprar más libros</a>
   </body>
</html>
```

3. **Funcionamiento de la aplicación LibreríaMVC**.

   3.1. Fijate en que el *servlet* será el punto de entrada a la aplicación. Podemos acceder indicando la *URL* asociada al *servlet* `http://localhost:8080/LibreriaMVC/shopping`, o la *URL* de la aplicación `http://localhost:8080/LibreriaMVC`, porque en el descriptor de despliegue hemos indicado como *welcome-file-list* de la aplicación el recurso *shopping*.

   3.2. En el primer acceso se crea un sesión HTTP que mantendrá el carrito de compra en el atributo de sesión **"carrito"**. Después se redirige la salida a la pagína **order.jsp** (todo=null).

   3.3. La página **order.jsp** muestra los libros disponibles y el carrito de compra. Cada vez que el usuario añade o borra un libro del carrito (una petición *request* con los parámetros "todo=add" o "todo=remove" es enviada al **ServletControlador**.

   3.4. Cuando el usuario pulsa en **Checkout** (una petición *request* con los parámetros "todo=checkout" es enviada al **ServletControlador**.

   3.5. Cuando el *ServletControlador* recibe los parámetros "todo=add" o "todo=remove", el controlador accede a la variable de sesión y actualiza el carrito, después redirige a **order.jsp** para seguir comprando.

3.6. Cuando el *ServletControlador* recibe el parámetro "todo=checkout", el controlador contabiliza el precio total y la cantidad total de libros y se redirige a **checkout.jsp**.

3.7. Ejecuta la aplicación desde *Eclipse* y comprueba su funcionamiento, Figura 3, 4:

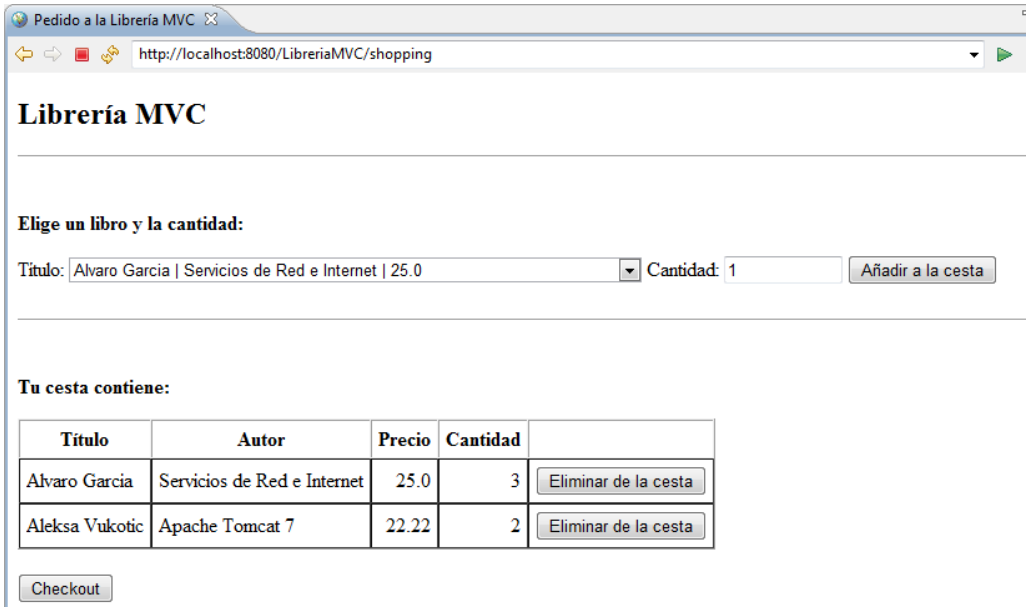

Figura 3: Funcionamiento LibreríaMVC

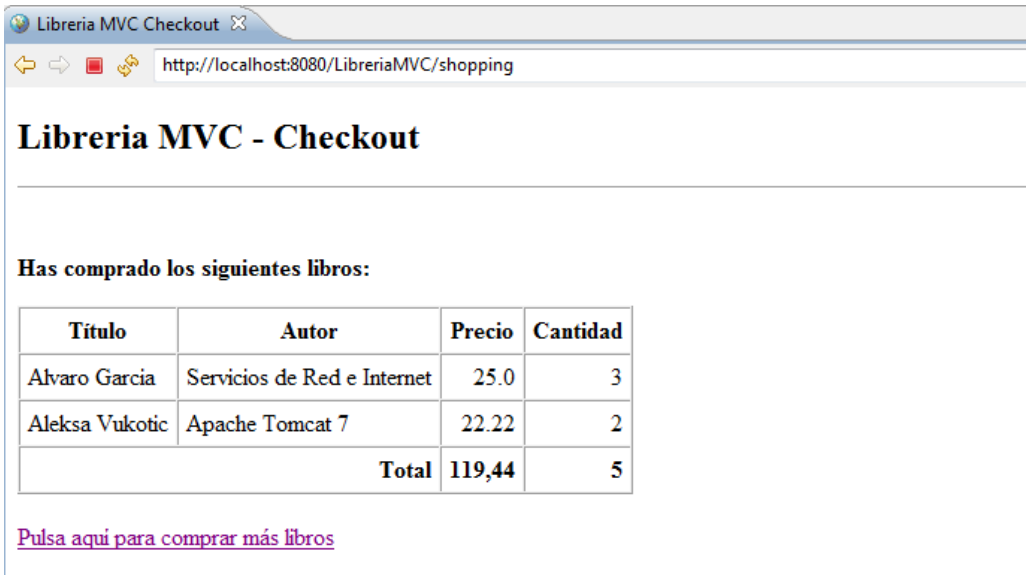

Figura 4: Funcionamiento LibreríaMVC

4. Finalmente, exporta la aplicación como fichero **.war**, despliegala y accede a la misma simultáneamente con los navegadores *Internet Explorer* y *Firefox*. Comprueba cómo se mantienen las dos sesiones independientes, Figura 5:

5. La aplicación está desarrollada suponiendo que ambos navegadores aceptan *cookies* para la gestión de la sesión. Si esto no fuera así el funcioamiento no sería el esperado.

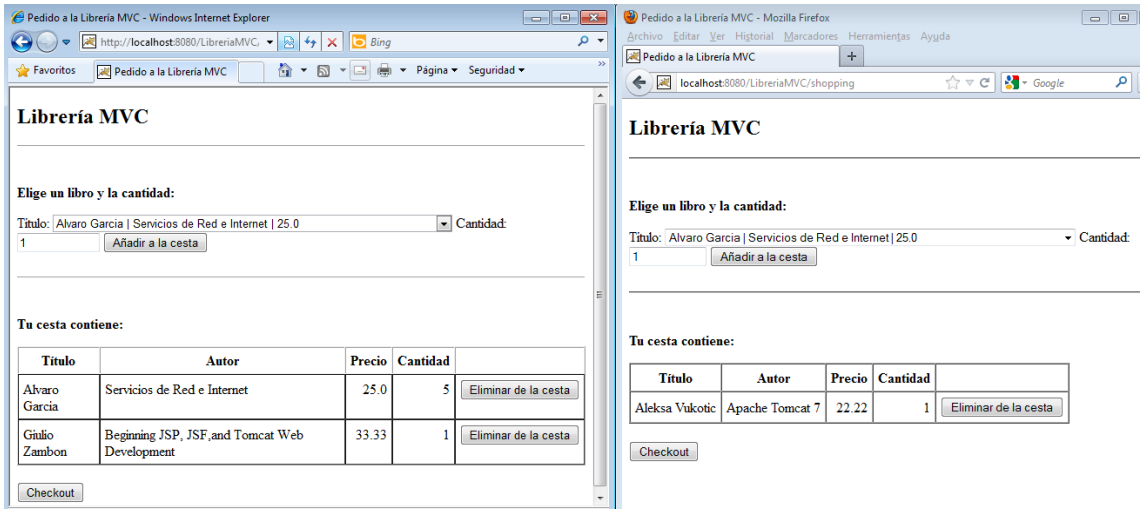

Figura 5: Sesiones simultáneas LibreríaMVC

◇