El patrón MVC: Front Controller

Las aplicaciones web, una vez diseñadas, suelen tener un ciclo de vida muy "movido" ocasionado por la necesidad de adaptarlas a las nuevas exigencias del mercado, de los usuarios o de la propia empresa. Esto supone un esfuerzo continuo en el desarrollo del código, ampliando ciertos aspectos ya implementados, eliminando otros o añadiendo nuevas funcionalidades, sin olvidar nunca que a veces, las aplicaciones terminan por integrarse o cooperar con otras. Todo ello nos hace reflexionar sobre la manera en que se plantean los armazones o estructuras básicas sobre las cuales se organiza el código.

Hace años que se comenzó a desarrollar y aplicar el concepto de patrón en el mundo del software, teniendo como objetivo resolver problemas concretos de forma que pudiera aplicarse de manera eficiente en un amplio espectro de problemas similares[63]. *Sun Microsystems* desarrolló todo un elenco de patrones de diseño orientados al diseño de aplicaciones Java, que en cierta medida, pueden ser aplicados también a otras tecnologías y lenguajes. Entre ellos se encuentra el patrón *Modelo-Vista-Controlador*[64], que suele identificarse mediante el acrónimo *MVC*. Este patrón tiene como objetivo separar en tres capas (*modelo*, *vista* y *controlador*) la arquitectura básica de una aplicación, ya sea web o no. Cada una de las capas posee una funcionalidad concreta y es independiente de las otras dos, salvo que las tres se comunican entre sí. El *modelo* representa toda la lógica de negocio de la aplicación, tanto datos como procesos, la *vista* se encarga de interaccionar con el usuario mostrándole una parte del modelo y el *controlador* sirve de intermediario entre la vista y el modelo, permitiendo procesar las peticiones que llegan del usuario a través de la vista llamando a los procedimientos adecuados en el modelo, recogiendo la información devuelta por éstos y proporcionándosela a la vista para que a su vez, ésta se los muestre al usuario como respuesta a su petición.

Para el caso de aplicaciones web que trabajan con servlets y páginas JSP, el patrón MVC se redefine como patrón *Front Controller* (*Controlador Frontal*), posibilitando la creación de un esqueleto estándar para el procesamiento de las peticiones HTTP. Normalmente el controlador de este patrón es un servlet aunque puede diseñarse haciendo uso de una página JSP. Todas las aplicaciones web tendrán por tanto un único punto de entrada y además el controlador podrá gestionar aspectos comunes como fuentes de datos (*datasources*), sesiones, etc.

La estructura de aplicación presentada en el curso no es más que una posible implementación para este patrón, por lo que no debe considerarse como algo rígido, sino todo lo contrario, como una base para desarrollar esqueletos más complejos. Como ejemplo de ello tenemos el framework *Struts*[65], basado en este mismo patrón.

Con el patrón planteado en clase se consigue implementar un sistema centralizado de administración de peticiones HTTP mediante una sintaxis estándar para todas ellas.

Una aplicación que se adapte a este patrón tendrá la siguiente estructura básica:

- Una clase denominada *Controlador*, encargado de hacer la función del *Front Controller*. Esta clase pertenece al paquete *controlador*.
- Varias clases de soporte a la clase *Controlador*, y que también pertenecen al paquete *controlador*. Éstas son: *AyudaSolicitud*, *FactoriaAcciones* y la interfaz *Accion*.
- El paquete *modelo.acciones*, que contendrá las clases de tipo *Accion* que ejecutarán la lógica de negocio de la aplicación y que serán llamadas desde el controlador.

63 Puede ampliarse información en:
http://es.wikipedia.org/wiki/Patrón_de_diseño
http://msdn.microsoft.com/es-es/library/bb972240.aspx
http://java.ciberaula.com/articulo/diseno_patrones_j2ee/
http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html
http://www.programacion.com/articulo/catalogo_de_patrones_de_diseno_j2ee_i_-_capa_de_presentacion_240/3
http://www.programacion.com/articulo/catalogo_de_patrones_de_diseno_j2ee_i_-_capa_de_presentacion_240/4

64 Más información en: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

65 Puede ampliarse información en:
http://struts.apache.org/
http://www.programacion.com/articulo/manual_basico_de_struts_156
http://www.casadellibro.com/libro-struts-2-el-framework-de-desarrollo-de-aplicaciones-java-ee/1702324/2900001380703
http://www.casadellibro.com/libro-struts-2/1230757/2900001290031

- El paquete *modelo.beans*, que contendrá los beans que se moverán a lo largo de la aplicación, bien como parte del modelo que procesan las vistas, bien como elementos de información de un contexto determinado (*request*, *session* o *application*) o como parte de los datos o procesos que conforman el modelo.
- Un conjunto de páginas JSP, que forman la vista y que son las encargadas de servir de interfaz entre el usuario y la aplicación.

Seguidamente se proporciona el código comentado de cada una de las clases que se proporcionan en el paquete *controlador*.

```
package fotogramas.controlador;

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

/**
 * Implementación del servlet Controlador
 * @author Eduardo A. Ponce
 * @version 1.0
 */
public class Controlador extends HttpServlet {
		private static final long serialVersionUID = 1L;
		private static final String CONTENT_TYPE = "text/html; charset=UTF-8";
		/* Es recomendable establecer como propiedades del servlet aquellos objetos
		 * que encapsulan fuentes de datos.
		*/
		/**
		 * Contiene el nombre del archivo que almacena los datos de los usuarios registrados.
		 */
		private static String fileUsers="/home/alumno/proyectos/fotograma/archivos/users.txt";
		/**
		 * Contiene el nombre del archivo que almacena los datos de los fotogramas.
		 */
		private static String fileFotogramas="/home/alumno/proyectos/fotograma/archivos/fotogramas.txt";

		/**
		* Este objeto es precisamente el objeto que encapsula toda la información
		* a nivel de aplicación. Se corresponde con el objeto application generado
		* por el contenedor web al generar los servlets de las páginas JSP.
		*/
		private ServletContext sc;

		 /**
		* Inicializa el servlet, y le proporciona un objeto, ServletConfig con
		* información de nivel de aplicación sobre el contexto de datos que rodea
		* al servlet en el contenedor web.
		* @see Servlet#init(ServletConfig)
		*/

		public void init(ServletConfig config) throws ServletException {
		// Imprescindible llamar a super.init(config) para tener acceso a la configuración
		// del servlet a nivel de contenedor web.
		    super.init(config);

		// En este punto se procedería a obtener las referencias a las fuentes de datos de la
		// aplicación. Como aún no se trabaja con bases de datos, basta con preparar el código
		// para trabajar con los dos archivos de datos establecidos.
		    sc = config.getServletContext(); //Se recupera el contexto de aplicación

		// Se añade a la sesión del usuario los atributos que permiten acceder a los nombres de
		// los archivos con los que trabajará la aplicación.
		    sc.setAttribute("fUsuarios", Controlador.getFileUsers());
		    sc.setAttribute("fFotogramas", Controlador.getFileFotogramas());
		}

		/**
		* Lo último que se debe hacer antes de que se elimine la instancia del
		* servlet.
		* @see Servlet#destroy()
```

```
*/

public void destroy() {
		// Elimina cualquier referencia a fuentes de datos del ámbito de aplicación,
		// liberando todos los recursos que tuviera asignados.
		sc.removeAttribute("fUsuarios");
		sc.removeAttribute("fFotogramas");
}

/**
* Procesa las peticiones que vienen por la vía GET.
* @param request La petición.
* @param response La respuesta.
* @throws javax.servlet.ServletException Error al ejecutar doPost()
* @throws java.io.IOException Error de E/S proviniente de doPost()
* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Se reenvía hacia el método doPost(), ya que tanto las peticiones GET como
    // las POST se procesarán igual, y de esta manera, se evita código redundante.
    doPost(request,response);
}

/**
* Procesa las peticiones que vienen por la vía POST.
* @param request La petición.
* @param response La respuesta.
* @throws javax.servlet.ServletException Puede ser lanzada por forward().
* @throws java.io.IOException Puede ser lanzada por forward().
* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //Se obtiene el objeto de ámbito sesión
    HttpSession sesion = request.getSession();
    // Obtener un objeto de ayuda para la solicitud
    AyudaSolicitud ayudaSol = new AyudaSolicitud(request);
    // Crear un objeto de acción partiendo de los parámetros asociados a la solicitud
    Accion accion = ayudaSol.getAccion();
    // Se proporciona el contexto del servlet (ámbito aplicación) a la acción
    accion.setSc(sc);
    // Se proporcionan las referencias a las fuentes de datos asociadas al servlet a la acción
    accion.setFteDatos(Controlador.getFileUsers(),Controlador.getFileFotogramas());
    // Se procesa la solicitud (lógica de empresa)
    if (accion.ejecutar(request,response))
    {
    // Si es correcto, obtener el componente relativo a la vista
      String vista = accion.getVista();

      // Añadir en la petición el modelo a visualizar
      request.setAttribute("modelo",accion.getModelo());

      // Enviar la respuesta a la solicitud
      RequestDispatcher rd = request.getRequestDispatcher(vista);
      rd.forward(request,response);
    }
    else
    {
    // Si la ejecución ha generado un error, procesarlo mediante el gestor centralizado de errores
      gesError(accion.getVista(),accion.getError(),request,response);
    }

}

/**
* Reenvía el proceso hacia una vista de gestión de errores.
* @param vistaError Página que gestionará el error.
* @param excepcion Objeto encapsulador de la excepción.
* @param request La petición.
* @param response La respuesta.
* @throws javax.servlet.ServletException Puede ser generada por forward().
```

```
	 * @throws java.io.IOException Puede ser generada por forward().
	 */
	private void gesError(String vistaError, Exception excepcion, HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException
	{
			RequestDispatcher rd = request.getRequestDispatcher(vistaError);
			request.setAttribute("error",excepcion);
			rd.forward(request,response);
	}

	/**
	 * @param fileUsers the fileUsers to set
	 */
	public static void setFileUsers(String fileUsers) {
			Controlador.fileUsers = fileUsers;
	}

	/**
	 * @return the fileUsers
	 */
	public static String getFileUsers() {
			return fileUsers;
	}

	/**
	 * @param fileFotogramas the fileFotogramas to set
	 */
	private static void setFileFotogramas(String fileFotogramas) {
			Controlador.fileFotogramas = fileFotogramas;
	}

	/**
	 * @return the fileFotogramas
	 */
	private static String getFileFotogramas() {
			return fileFotogramas;
	}
}
```

```
/**
 * Instancia objetos de tipo Acción.
 * Es una clase abstracta que impide que se puedan instanciar objetos de ella,
 * pero permite que se obtengan clases derivadas.
 * Se encarga de obtener los objetos Acción específicos para una determinada acción.
 */
package fotogramas.controlador;

import fotogramas.modelo.acciones.*;

/**
 * Factoría para las acciones
 * @author Eduardo A. Ponce
 * @version 1.0
 */
public abstract class FactoriaAcciones {
	public static Accion creaAccion(String accion)
	 {
	   // Devuelve objetos Accion en función del parámetro de acción proporcionado
	   if (accion.equals("login"))
	     // Comprobar login
	     return new AccionLogin();
	   if (accion.equals("index"))
	     return new AccionIndex();
	   if (accion.equals("registrar"))
	     return new AccionIndex();
	   /*
			if (accion.equals("logout"))
				// Se desconecta al usuario de la sesión actual
				return new AccionLogout();
			if (accion.equals("concursar"))
				// Proceso de concurso/adivinación de un fotograma de una película
				return new AccionConcursar();
```

```
                */
                // En caso de que no coincida con ninguna acción, se devuelve la acción que conduce a index.html.
                return new AccionIndex();
              }
}
```

```
/**
 * Permite crear un objeto de ayuda que acepta el objeto de solicitud
 * en su constructor. Proporciona funcionalidad adicional a la solicitud.
 */
package fotogramas.controlador;

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

/**
 * Preprocesa la solicitud
 * @author Eduardo A. Ponce
 * @version 1.0
 */
public class AyudaSolicitud {
              HttpServletRequest request;

              public AyudaSolicitud(HttpServletRequest request)
                throws ServletException, IOException
              {
                this.request = request;
              }

              public Accion getAccion()
              {
                String accion = (String) request.getParameter("accion");
                System.out.println("Accion: "+accion);
                // Antes de crear la acción puede haber procesamiento previo de la
                // solicitud, para dejarla preparada para el método creaAccion().
                return FactoriaAcciones.creaAccion(accion);
              }
}
```

```
/**
 * Obliga a que todas las acciones de la aplicación empleen la misma nomenclatura de
 * métodos y funcionalidad.
 * La arquitectura del sistema de administración de solicitudes permanece inalterable.
 */
package fotogramas.controlador;

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

/**
 * Interface Accion
 * @author Eduardo A. Ponce
 * @version 1.0
 */
public interface Accion {
  public boolean ejecutar(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException;

  public String getVista();

  public Object getModelo();

  public void setSc(ServletContext sc);

  public Exception getError();

  public void setFteDatos(String fUsuarios, String fFotogramas);

}
```

Los beans que emplea este ejemplo se ubican en el paquete *fotogramas.modelo.beans*, y son *BeanError* y *BeanUsuario*. El código es el que se muestra seguidamente:

```
/**
 * Encapsula la información de error que pueda producirse mientras
 * se ejecuta la aplicación.
 * Códigos de error y descripción
 * Código   Descripción
 * 1 -      El login y clave de usuario buscados no se han encontrado o son erróneos.
 * 2 -      Error en acceso de archivo de datos de usuarios.
 * 3 -      El login ya está registrado. Inténtelo con otro.
 */
package fotogramas.modelo.beans;

import java.io.*;

/**
 * Implementación del bean de Error
 * @author Eduardo A. Ponce
 * @version 1.0
 */
@SuppressWarnings("serial")
public class BeanError extends Exception implements Serializable{
	/**
	 * Código de error
	 */
	private int codError;
	/**
	 * Mensaje asociado al código de error
	 */
	private String mensError;
	/**
	 * Objeto Exception que encapsula la excepción asociada al error
	 */
	private Exception excepcion = null;

	/**
	 * Constructor que crea un objeto de error con código y mensaje de error, pero
	 * sin objeto Exception asociado.
	 * @param codError Código de error
	 * @param mensError Mensaje de error
	 */
	 public BeanError(int codError, String mensError)
	 {
	   super(mensError);
	   this.setCodError(codError);
	   this.setMensError(mensError);
	 }
	 /**
	  * Constructor que crea un objeto de error con código y mensaje de error, así como
	  * con el objeto Exception que encapsula la excepción producida.
	  * @param codError Código de error
	  * @param mensError Mensaje de error
	  * @param excepcion Excepción producida
	  */
	 public BeanError(int codError, String mensError, Exception excepcion)
	 {
	   super(mensError);
	   this.setCodError(codError);
	   this.setMensError(mensError);
	   this.setExcepcion(excepcion);
	 }

	/**
	 * Establece el valor del código de error.
	 * @param codError El código de error
	 */
	private void setCodError(int codError) {
		this.codError = codError;
	}
```

```
	/**
	 * Devuelve el valor del código de error
	 * @return El código de error
	 */

	private int getCodError() {
		return codError;
	}

	/**
	 * Establece el mensaje de error
	 * @param mensError El mensaje de error
	 */
	private void setMensError(String mensError) {
		this.mensError = mensError;
	}

	/**
	 * Devuelve el mensaje de error
	 * @return El mensaje de error
	 */
	@SuppressWarnings("unused")
	private String getMensError() {
		return mensError;
	}

	/**
	 * Establece el objeto Exception
	 * @param excepcion La excepción producida
	 */
	private void setExcepcion(Exception excepcion) {
		this.excepcion = excepcion;
	}

	/**
	 * Devuelve el objeto Exception
	 * @return La excepción producida
	 */
	@SuppressWarnings("unused")
	private Exception getExcepcion() {
		return excepcion;
	}

}
```

```
/**
 *
 */
package fotogramas.modelo.beans;

import java.io.*;

/**
 * Implementación del bean de Usuario
 * @author Eduardo A. Ponce
 * @version 1.0
 */
@SuppressWarnings("serial")
public class BeanUsuario implements Serializable {

	/**
	 * Login de usuario
	 */
	private String login;
	/**
	 * Clave del usuario
	 */
	private String clave;
	/**
	 * Nombre del usuario
	 */
```

```
private String nombre;
/**
 * Apellidos del usuario
 */
private String apellidos;
/**
 * Email del usuario
 */
private String email;
/**
 * Edad del usuario
 */
private int edad;

// Mensajes de error asociados a los datos asociados al usuario
/**
 * Mensaje de error asociado al login
 */
private String msgLogin=null;
/**
 * Mensaje de error asociado a la clave
 */
private String msgClave=null;
/**
 * Mensaje de error asociado al nombre
 */
private String msgNombre=null;
/**
 * Mensaje de error asociado a los apellidos
 */
private String msgApellidos=null;
/**
 * Mensaje de error asociado al email
 */
private String msgEmail=null;
/**
 * Mensaje de error asociado a la edad
 */
private String msgEdad=null;

/**
 * Indica si el bean posee datos o no
 */
private boolean vacio=true;

/**
 * Indica si la instancia recoge los datos de un usuario válido en la aplicación
 */
private boolean encontrado = false;

/**
 * Constructor por defecto.
 */
public BeanUsuario()
{

}
/**
 * Constructor sólo con login y clave. Establece que el bean no está vacío.
 * @param login Login del usuario
 * @param clave Clave del usuario
 */
public BeanUsuario(String login, String clave)
{
        this.login = login;
        this.clave = clave;
        this.setVacio(false);
}

/**
 * Constructor con todos los parámetros. Establece que el bean no está vacío.
 * @param login Login del usuario
```

```
     * @param clave Clave del usuario
     * @param nombre Nombre del usuario
     * @param apellidos Apellidos del usuario
     * @param email Email del usuario
     * @param edad Edad del usuario
     */
    public BeanUsuario(String login, String clave, String nombre, String apellidos, String email, int edad)
    {
            this.login = login;
            this.clave = clave;
            this.nombre = nombre;
            this.apellidos = apellidos;
            this.email = email;
            this.edad = edad;
            this.setVacio(false);
    }
    /**
     * Establece el valor del login.
     * @param login Login del usuario
     */
    public void setLogin(String login) {
            this.login = login;
    }
    /**
     * Devuelve el valor del login.
     * @return El login del usuario
     */
    public String getLogin() {
            return login;
    }
    /**
     * Establece el valor de la clave.
     * @param clave La clave del usuario
     */
    public void setClave(String clave) {
            this.clave = clave;
    }
    /**
     * Devuelve el valor de la clave.
     * @return La clave
     */
    public String getClave() {
            return clave;
    }
    /**
     * Establece el nombre del usuario.
     * @param nombre El nombre de usuario
     */
    public void setNombre(String nombre) {
            this.nombre = nombre;
    }
    /**
     * Devuelve el nombre del usuario.
     * @return El nombre del usuario
     */
    public String getNombre() {
            return nombre;
    }
    /**
     * Establece los apellidos del usuario.
     * @param apellidos Los apellidos del usuario
     */
    public void setApellidos(String apellidos) {
            this.apellidos = apellidos;
    }
    /**
     * Devuelve los apellidos del usuario.
     * @return Los apellidos
     */
    public String getApellidos() {
            return apellidos;
    }
```

```
/**
 * Establece el email del usuario.
 * @param email Email del usuario
 */
public void setEmail(String email) {
	this.email = email;
}
/**
 * Devuelve el email del usuario.
 * @return El email
 */
public String getEmail() {
	return email;
}
/**
 * Establece la edad del usuario.
 * @param edad La edad del usuario
 */
public void setEdad(int edad) {
	this.edad = edad;
}
/**
 * Devuelve la edad del usuario.
 * @return La edad
 */
public int getEdad() {
	return edad;
}
/**
 * Establece que la instancia recoge los datos de un usuario válido.
 * @param encontrado true si el usuario es válido, false en caso contrario
 */
public void setEncontrado(boolean encontrado) {
	this.encontrado = encontrado;
}
/**
 * Indica si el usuario es un usuario válido o no.
 * @return true si el usuario es válido, false en caso contrario
 */
public boolean isEncontrado() {
	return encontrado;
}

/**
 * Establece el mensaje de error asociado al login.
 * @param msgLogin El mensaje de error asociado al login
 */
public void setMsgLogin(String msgLogin) {
	this.msgLogin = msgLogin;
}

/**
 * Devuelve el mensaje de error asociado al login.
 * @return El mensaje de error asociado al login
 */
public String getMsgLogin() {
	return msgLogin;
}

/**
 * Establece el mensaje de error asociado a la clave.
 * @param msgClave El mensaje de error asociado a la clave
 */
public void setMsgClave(String msgClave) {
	this.msgClave = msgClave;
}

/**
 * Devuelve el mensaje de error asociado a la clave.
 * @return the msgClave
 */
public String getMsgClave() {
```

```
        return msgClave;
}

/**
 * Establece el mensaje de error asociado al nombre del usuario.
 * @param msgNombre El mensaje de error asociado al nombre del usuario
 */
public void setMsgNombre(String msgNombre) {
        this.msgNombre = msgNombre;
}

/**
 * Devuelve el mensaje de error asociado al nombre del usuario.
 * @return El mensaje de error asociado al nombre del usuario
 */
public String getMsgNombre() {
        return msgNombre;
}

/**
 * Establece el mensaje de error asociado a los apellidos del usuario.
 * @param msgApellidos El mensaje de error asociado a los apellidos del usuario
 */
public void setMsgApellidos(String msgApellidos) {
        this.msgApellidos = msgApellidos;
}

/**
 * Devuelve el mensaje de error asociado a los apellidos del usuario.
 * @return El mensaje de error asociado a los apellidos del usuario
 */
public String getMsgApellidos() {
        return msgApellidos;
}

/**
 * Establece el mensaje de error asociado al email del usuario.
 * @param msgEmail El mensaje de error asociado al email del usuario
 */
public void setMsgEmail(String msgEmail) {
        this.msgEmail = msgEmail;
}

/**
 * Devuelve el mensaje de error asociado al email del usuario.
 * @return El mensaje de error asociado al email del usuario
 */
public String getMsgEmail() {
        return msgEmail;
}

/**
 * Establece el mensaje de error asociado a la edad del usuario.
 * @param msgEdad El mensaje de error asociado a la edad del usuario
 */
public void setMsgEdad(String msgEdad) {
        this.msgEdad = msgEdad;
}

/**
 * Devuelve el mensaje de error asociado a la edad del usuario.
 * @return El mensaje de error asociado a la edad del usuario
 */
public String getMsgEdad() {
        return msgEdad;
}
/**
 * Establece si el estado de la instancia posee datos o no.
 * @param vacio true si posee datos, false en caso contrario
 */
public void setVacio(boolean vacio) {
        this.vacio = vacio;
```

```
    }
    /**
     * Indica si la instancia recoge, o no,datos de usuario.
     * @return true si aún no hay datos, false en caso contrario
     */
    public boolean isVacio() {
            return vacio;
    }

}
```

Ahora se muestran los códigos correspondientes a las páginas HTML y JSP que forman el proyecto:

Página: index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>El Fotograma Perdido MVC v1</title>
</head>
<body>
<form action="controlador" method="post">
<input type="hidden" name="accion" value="login">
Si ya es usuario de la aplicación:  <br>
<input name="login" value="login"
     type="submit" style="width: 84px; height: 33px">
</form>
<form action="controlador" method="post">
<input type="hidden" name="accion" value="registrar">
Si aún no es usuario de la aplicación:  <br>
<input name="registrar" value="registrar"
     type="submit" style="width: 84px; height: 33px">
</form>
</body>
</html>
```

Página: login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page errorPage="gesError.jsp?pagOrigen=login.jsp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>El Fotograma Perdido MVC v1</title>
</head>
<body>
Aquí se produce la petición de login y clave para un usuario registrado.
<form action="controlador" method="post">
<input type="hidden" name="accion" value="index">
<input name="volver" value="volver" type="submit">
</form>
</body>
</html>
```

Página: gesError.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page isErrorPage="true" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Página de gestión de Errores</title>
</head>
<body>
<%
     String pag=request.getParameter("pagOrigen");
%>
Se ha producido un error en la página <%=pag%> <br>
```

```
El mensaje de la excepción es: <%=exception%> <br>
getMessage: <%=exception.getMessage()%><br>
</body>
</html>
```

Las clases orientadas a realizar las acciones solicitadas por el usuario se ubican en el paquete *fotogramas.modelo.acciones*. En el ejemplo que se comenta sólo se han implementado dos acciones, *AccionIndex* y *AccionLogin*.

Seguidamente se muestran sus correspondientes códigos:

Clase: AccionIndex.java

```
/**
 * Acción: Mostrar la página index.html
 */
package fotogramas.modelo.acciones;

import java.io.IOException;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import fotogramas.controlador.Accion;

/**
 * Acción Index
 * @author Eduardo A. Ponce
 * @version 1.0
 */
public class AccionIndex implements Accion {


	/**
	 * Página que muestra la vista
	 */
	private String vista="index.html";
	/**
	 * Objeto que encapsula el modelo a procesar en la vista
	 */
	private String modelo = "";
	/**
	 * Ejecuta la acción.
	 * En este caso, se limita a preparar todo para que se muestre la página de índice.
	 */
	@Override
	public boolean ejecutar(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		return true;
	}

	/**
	 * Muestra la vista. En este caso, devuelve la cadena vacía.
	 * @return Cadena vacía
	 */
	public String getVista() {
		return vista;
	}

	/**
	 * Devuelve el modelo que deberá procesar la vista.
	 * @return El modelo a mostrar
	 */
	public Object getModelo() {
		return modelo;
	}

	/**
	 * Establece el modelo que deberá procesar la vista.
```

```
	 * @param modelo El modelo a establecer
	 */
	public void setModelo(Object modelo) {
		this.modelo = (String) modelo;
	}
	/**
	 * Establece el contexto del servlet.
	 * @param sc Contexto del servlet
	 */
	public void setSc(ServletContext sc) {

	}

	/**
	 * Devuelve el objeto Exception asociado al error producido en la ejecución de la acción.
	 * En este caso, como no se genera ninguno, se establece a null.
	 * @return null
	 */
	public Exception getError() {
		return null;
	}

	/**
	 * Establece el valor de las fuentes de datos, fichero de usuarios y fichero de fotogramas.
	 * @param fUsuarios Nombre del fichero de usuarios
	 * @param fFotogramas Nombre del fichero de fotogramas
	 */
	public void setFteDatos(String fUsuarios, String fFotogramas) {

	}

	/**
	 * Establece el nombre de la página JSP que mostrará el resultado de la petición.
	 * @param vista El nombre de la página JSP a establecer
	 */
	public void setVista(String vista) {
		this.vista = vista;
	}

}
```

Clase: AccionLogin.java

```
/**
 * Acción: procesar login usuario
 */
package fotogramas.modelo.acciones;

import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;
import fotogramas.controlador.Accion;
import fotogramas.modelo.beans.*;

/**
 * Procesa los datos de login y clave proporcionados por un usuario.
 * @author Eduardo A. Ponce
 * @version 1.0
 */

public class AccionLogin implements Accion {

	// Aquí se deben declarar las propiedades de la acción

	/**
	 * Página que muestra la vista.
	 */
	private String vista;
	@SuppressWarnings("unused")
	/**
	 * Constante que contiene la página JSP que se mostrará si el proceso de login ha sido correcto y se procede al concurso.
	 */
```

```
private final String vistaOK = "concursar.jsp";
@SuppressWarnings("unused")
/**
 * Constante que contiene la página JSP que se mostrará si el proceso ha generado un error.
 */
private final String vistaError = "gesError.jsp";
/**
 * Constante que almacena la vista que se emplea para la identificación de usuario.
 */
private final String vistaForm= "login.jsp";
/**
 * Objeto que encapsula el modelo a procesar en la vista.
 */
private BeanUsuario modelo;
@SuppressWarnings("unused")
/**
 * Contiene el nombre del archivo de usuarios.
 */
private String fUsuarios;
@SuppressWarnings("unused")
/**
 * Contiene el nombre del archivo de fotogramas.
 */
private String fFotogramas;

/**
 * Variable booleana que indica si el proceso se ha realizado correctamente o no.
 */
private boolean ok = true;

// Estas variables las necesitan todas las acciones
/**
 * Contexto del servlet
 */
private ServletContext sc;
/**
 * La sesión asociada al usuario
 */
private HttpSession sesion;
/**
 * Bean que encapsula un error producido durante la ejecución de la acción
 */
private fotogramas.modelo.beans.BeanError error;

/**
 * Constructor por defecto
 */
public AccionLogin()
{

}
/**
 * Ejecuta el proceso asociado a la acción.
 * @param request Objeto que encapsula la petición.
 * @param response Objeto que encapsula la respuesta.
 * @return true o false en función de que no se hayan producido errores o lo contrario.
 * @see fotogramas.controlador.Accion#ejecutar(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
 */
public boolean ejecutar(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException, IOException
{
        // Para validar el login
        //              ValidaLogin valogin = null;
        BeanUsuario bUsuario = null;
        HttpSession sesion = null;

        /*
         * Primero debe comprobar si el parámetro accion de la petición es "login", ya que en caso de serlo,
         * deberá reenviar a login.jsp con un modelo vacío. En caso contrario, procesará el login.
         */
        sesion = request.getSession();
```

```
        if (request.getParameter("accion").equals("login"))
        {
                setVista(vistaForm);
                bUsuario = new BeanUsuario();
                setModelo(bUsuario);
                sesion.setAttribute("beanUsuario",bUsuario);
        }
        else
                if (request.getParameter("accion").equals("validaLogin"))
                {
                        //Se recuperan los parámetros de la petición y se almacenan en el bean de usuario
                        ((BeanUsuario) sesion.getAttribute("beanUsuario")).setLogin(request.getParameter("login"));
                        ((BeanUsuario) sesion.getAttribute("beanUsuario")).setClave(request.getParameter("clave"));
                        ((BeanUsuario) sesion.getAttribute("beanUsuario")).setLogin(request.getParameter("login"));

                }
                else
                        ok=false;

        return ok;
}

/**
* Devuelve la vista actual.
* @param La vista a devolver al usuario.
*/
public String getVista()
{
        return vista;
}
/**
 * Establece la vista
 * @param vista Página que se enviará al usuario
 */
public void setVista(String vista)
{
        this.vista = vista;
}
/**
 * Devuelve el modelo con el que trabajará la vista.
 * @return El modelo a procesar por la vista.
 */
public Object getModelo()
{
        return modelo;
}
/**
 * Establece el modelo con el que trabajará  la vista.
 * @param modelo Conjunto de datos a procesar por la vista.
 */
public void setModelo(BeanUsuario modelo)
{
    this.modelo = modelo;
}
/**
 * Establece el contexto del servlet (nivel aplicación)
 * @param sc Objeto ServletContext que encapsula el ámbito de aplicación.
 */
public void setSc(ServletContext sc)
{
        this.sc = sc;
}
/**
 * Devuelve el contexto del servlet (nivel aplicación)
 * @return Objeto que encapsula el nivel de aplicación.
 */
public ServletContext getSc()
{
    return sc;
}
/**
 * Establece una situación de error.
```

```
     * @param error Objeto BeanError que encapsula el error.
     */
    public void setError(fotogramas.modelo.beans.BeanError error)
    {
        this.error = error;
    }
    /**
     * Devuelve un objeto de error asociado al procesamiento de la acción.
     * @return Objeto que encapsula una situación de error producida durante la ejecución de la acción.
     */
    public Exception getError() {
            return error;
    }

    /**
     * Establece la/s fuente/s de datos
     * @see fotogramas.controlador.Accion#setFteDatos(java.lang.String, java.lang.String)
     */
    public void setFteDatos(String fUsuarios, String fFotogramas) {
            this.fUsuarios = fUsuarios;
            this.fFotogramas = fFotogramas;
    }
    /**
     * Establece el objeto que encapsula la sesión actual.
     * @param sesion La sesión a establecer en la acción.
     */
    @SuppressWarnings("unused")
    private void setSesion(HttpSession sesion) {
            this.sesion = sesion;
    }
    /**
     * Devuelve la sesión actual del usuario.
     * @return sesion La sesión actual del usuario.
     */
    @SuppressWarnings("unused")
    private HttpSession getSesion() {
            return sesion;
    }

}
```

El descriptor de despliegue, *web.xml*, incorpora la configuración del servlet *Controlador*.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      id="WebApp_ID" version="2.5">
  <display-name>FotogramaPerdidoMVC1</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <description>Controlador del patrón MVC</description>
    <display-name>Controlador</display-name>
    <servlet-name>Controlador</servlet-name>
    <servlet-class>fotogramas.controlador.Controlador</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Controlador</servlet-name>
    <url-pattern>/controlador</url-pattern>
  </servlet-mapping>
</web-app>
```

Y jboss-web.xml es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
      <context-root>fotogramaperdidomvc1</context-root>
</jboss-web>
```