Trabajo Obligatorio

PASARELA GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN

DESARROLLO DE APLICACIONES WEB II

Fernando del Fresno García

UNIVERSIDAD CATÓLICA DE ÁVILA
UCAV

1) RESUMEN

El patrón de diseño o arquitectura de aplicaciones software separa la lógica de negocio de la parte visual o interfaz de usuario y los datos. Es una evolución de años de desarrollo de software donde el código o lógica computacional de la aplicación se veía mezclado con la parte visual y podía entorpecer el entendimiento y por lo tanto el desarrollo o mantenimiento. El **modelo MVC consigue separar conceptos para un mejor entendimiento y gestión de un desarrollo software**.

Este patrón de diseño mejora el desarrollo por separado de la interfaz gráfica, la lógica de negocio y los datos, permitiendo la separación de código y por lo tanto **la reutilización de código.**

Actualmente se usa en multitud de Frameworks de desarrollo en cualquier lenguaje de programación como Java, PHP, Ruby onRails, ASP, etc.

2) FUNDAMENTOS DEL MODELO MVC

La principal ventaja de este patrón es la separación del Modelo-Vista-Controlador, donde cada parte se encarga directamente de gestionar su cometido ofreciendo al equipo de desarrollo mayor flexibilidad.

En este patrón se encuentran los siguientes componentes:

- **Vista**. Contenido de la interfaz de usuario. En este caso en HTML.
- **Controlador**. En nuestro caso código Java, que accede a los datos (modelo) y dependiendo de la lógica usada redirecciona a la vista correspondiente.
- **Modelo**. Clases que se corresponden directamente con objetos de la base de datos.

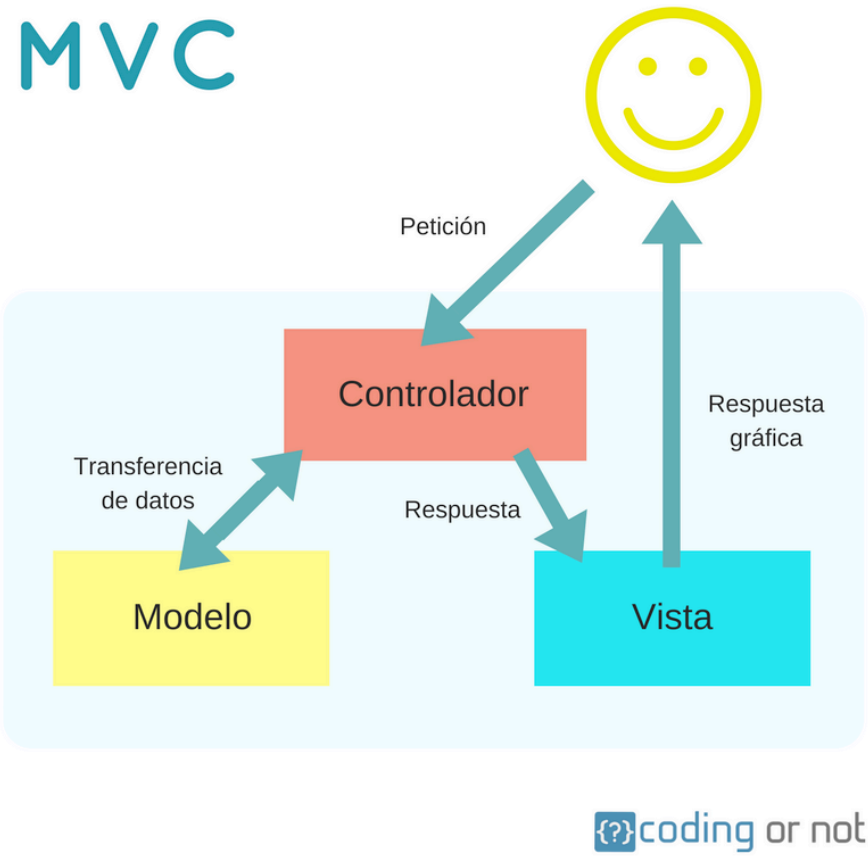
Para hacer un seguimiento de una supuesta petición de un usuario en un entorno web diseñada con el modelo MVC podríamos detallar este flujo:

1. Desde la interfaz de usuario (VISTA), el usuario realizaría cualquier acción (evento).

2. A través de los mecanismos de control (ficheros de configuración), el CONTROLADOR correspondiente recibiría esa acción y la trataría haciendo uso del MODELO modificando o usando los datos que maneja el modelo. Esto es lo que se llama 'lógica de negocio'.

3. A partir de unos valores de retorno, el CONTROLADOR devuelve el control a la VISTA con los datos del MODELO para que se puedan mostrar los resultados.

4. Así repetidamente se va generando ciclos dependiendo de las acciones o eventos dependiendo de las interacciones del usuario.



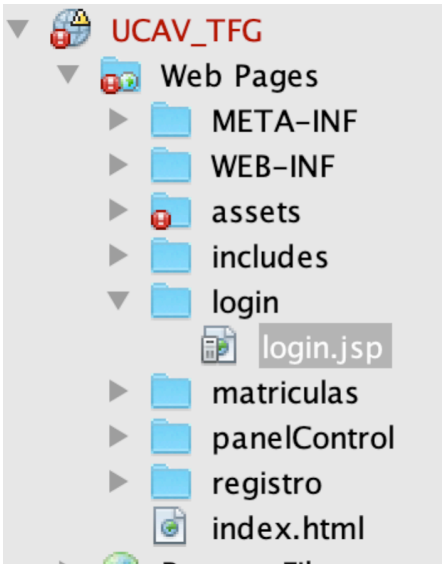
3) IMPLEMENTACIÓN MVC. TRABAJO OLIGATORIO

A continuación detallo como he implementado el patrón de diseño MVC en la aplicación desarrollada para la asignatura 'Desarrollo de Aplicaciones Web II'.

Este ejemplo de implementación lo he basado en el formulario de *login* y su posterior chequeo de acceso, por su simplicidad y para poder mostrar todos los componentes que intervienen.

VISTA

Las vistas están organizadas por carpetas intentando separarlas unas de otras para su mayor comprensión y mantenimiento.



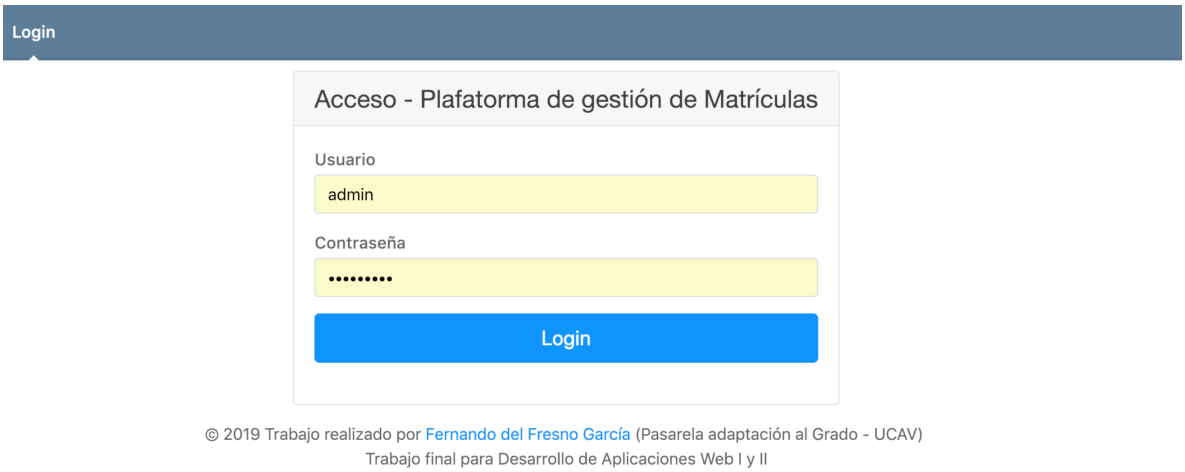
La vista inicial que muestra la aplicación como página de bienvenida (*index.html*) está configurada en el fichero web.xml:

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

En el fichero *struts.xml* se define la acción que muestra la vista del *login.jsp*

```
<action name="Login">
    <result>login/login.jsp</result>
</action>
<action name="CheckLogin" class="actions.login.CheckLoginAction">
        <result name="SUCCESS">panelControl/dashboard.jsp</result>
        <result name="INPUT">login/login.jsp</result>
</action>
```

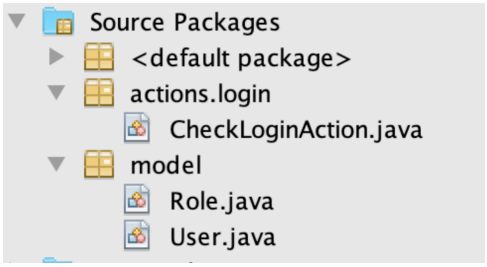
Desde la página de bienvenida el usuario puede lanzar un evento (*enlace*) que activa la acción con el nombre "Login" y muestra el formulario de acceso.



Tras introducir el usuario y contraseña en el formulario, se pulsa sobre el botón 'Login' que activa la acción configurada con el nombre "*CheckLogin*" en el fichero *struts.xml*

Como se puede ver, la acción tiene una clase asociada '*action.login.CheckLoginAction*'. En ese momento entra en juego el controlador.

CONTROLADOR



El controlador '*CheckLoginAction*' esta dentro del paquete '*actions.login*' igualmente como parte de la organización del proyecto.

El controlador debe tener los atributos de los campos que vienen en el formulario junto con sus métodos *setters y getters,* además del método *execute* donde se controlará la lógica de negocio y se accederá al modelo.

Aunque en la imagen se ve que el controlador hace uso de la sesión, no detallaré nada de ello en este documento, ya que la función de este documento es la explicación del MVC usado en el trabajo obligatorio.

```java
/**
 *
 * @author fernandofresno
 */
public class CheckLoginAction extends ActionSupport implements SessionAware {

    private String username;
    private String password;
    Map<String, Object> appSession;


    public void setUsername(String username) {
        this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }
```

El método *execute* es el que realiza la lógica de esta acción. Se ve como se conecta a la base de datos.

```java
// all struts logic here
public String execute() throws Exception {

    //open context
    Context ctx = new InitialContext();
    if ( ctx == null ) throw new Exception("Error al cargar el contexto");
    //DataSource y connection
    DataSource pool;
    pool = (DataSource)ctx.lookup("java:/comp/env/jdbc/ConexionMySQL");
    Connection conn = pool.getConnection();
    //Statements and execute query
    Statement stmt = conn.createStatement();
    String query = "SELECT * FROM USERS WHERE USERNAME = '"
                    + this.getUsername() +
                    "' AND PASSWORD = '" + this.getPassword() +"'";

    ResultSet rs = stmt.executeQuery(query);
```

Después de realizar la lógica necesaria, se procede a devolver unas cadenas de texto. He usado la cadena de texto "SUCCESS" para indicar que la operación se ha realizado con éxito y la cadena "INPUT" para indicar que hay algún contratiempo en los datos introducidos. En este caso, indica que el usuario no existe en base de datos.

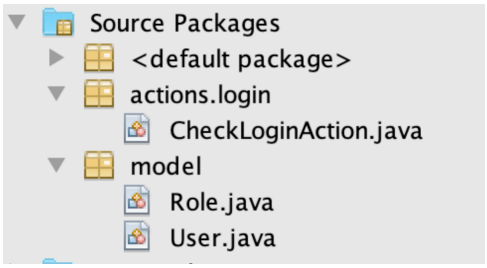
```
//user exist
if (rs.next()) {
    User usuario = new User();
    usuario.setIdUser(rs.getInt("ID_USER"));
    usuario.setUsername(rs.getString("USERNAME"));
    usuario.setEmail(rs.getString("EMAIL"));

    //set user to session
    setUserToSession(usuario);
    //Close
    rs.close();
    stmt.close();
    conn.close();
    //return success
    return "SUCCESS";
}

//only when user not exist
this.setUsername("");
this.setPassword("");
return "INPUT";
```

Estas cadenas se corresponden con los valores de respuesta que configuramos en el fichero '*struts.xml*' para la acción "*CheckLogin*" que devuelve el control a una vista u otra dependiendo del resultado de vuelta del controlador.

```
<action name="Login">
    <result>login/login.jsp</result>
</action>
<action name="CheckLogin" class="actions.login.CheckLoginAction">
        <result name="SUCCESS">panelControl/dashboard.jsp</result>
        <result name="INPUT">login/login.jsp</result>
</action>
```

MODELO

En este caso la parte del modelo que se ha usado en el método '*execute*' es la clase User.java que define un usuario y que se ha utilizado para comprobar que el usuario existe o no en base de datos.



Como se puede ver en la imagen las clases que formen parte del modelo de la aplicación se almacenarán sobre el paquete *model*.