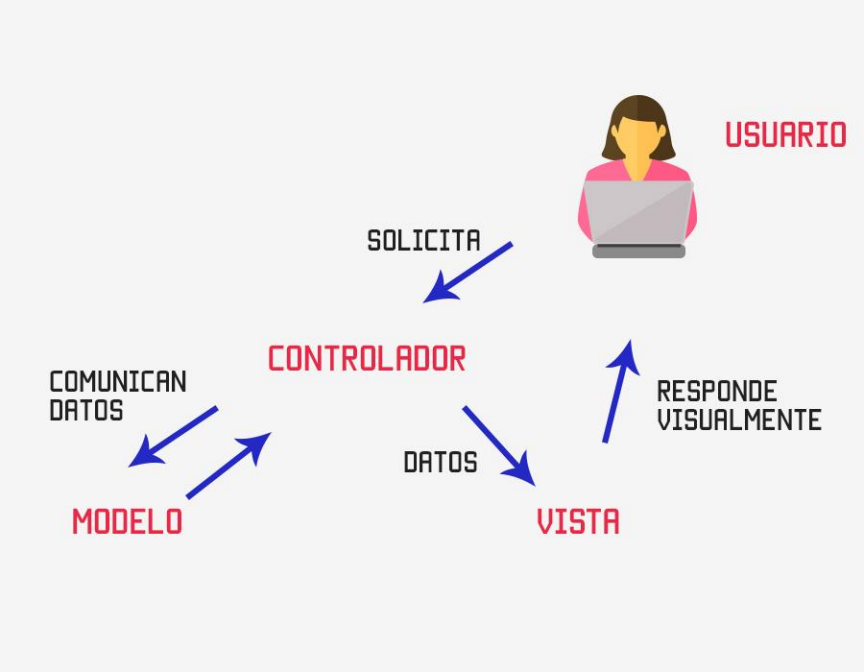
NOMBRE: Ricardo Antonio De los Santos Manzueta
MATRICULA: 100306344
SECCION: 01

Modelo MVC

MVC (Model, View, Controller)

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Models y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación del MVC) para la arquitectura, entre ellos podemos mencionar a Ruby on Rails, Django, AngularJS y muchos otros más. En este pequeño artículo intentamos introducirte a los conceptos del MVC.

Una imagen



Una analogía

Una que me gusta mucho es la de la televisión. En tu televisión puedes ver distintos canales distribuidos por tu tu proveedor de cable o televisión (que representa al modelo), todos los canales que puedes **ver** son la vista, y tú cambiando de canal, **controlando qué ves** representas al controlador.

La explicación

Los puntos anteriores so sólo para proveer background, y que ojalá puedas utilizar las referencias ahora que vamos a explicar qué es.

Antes que nada, me gustaría mencionar por qué se utiliza el **MVC**, la razón es que nos permite separar los componentes de nuestra aplicación dependiendo de la responsabilidad que tienen, esto significa que cuando hacemos un cambio en alguna parte de nuestro código, esto no afecte otra parte del mismo. Por ejemplo, si modificamos nuestra Base de Datos, sólo deberíamos modificar el modelo que es **quién se encarga de los datos** y el resto de la aplicación debería permanecer intacta. Esto respeta el principio de la responsabilidad única. Es decir, una parte de tu código no debe de saber qué es lo que hace toda la aplicación, sólo debe de tener una responsabilidad.

En web, el MVC funcionaría así. Cuando el usuario manda una petición al navegador, digamos quiere ver el curso de AngularJS, el controlador responde a la solicitud, porque él es el que controla la lógica de la app, una vez que el controlador nota que el usuario solicitó el curso de Angular, le pide al modelo la información del curso.

El modelo, que se encarga de los datos de la app, consulta la base de datos y digamos, obtiene todos los vídeos del curso de AngularJS, la información del curso y el título, el modelo responde al controlador con los datos que pidió (nota como en la imagen las flechas van en ambos sentidos, porque el controlador pide datos, y el modelo responde con los datos solicitados).

Una vez el controlador tiene los datos del curso de AngularJS, se los manda a la vista, la vista aplica los estilos, organiza la información y construye la página que ves en el navegador.

Resumamos entonces los conceptos.

Modelo

Se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. todo eso va aquí, en el modelo.

Controlador

Se encarga de... controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

Vistas

Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

Podemos definir de una manera sencilla a MVC como un modelo que separa el software en 3 cosas: Modelo, Vista y Controlador:

Modelo: Es la representación de la información que vamos a utilizar en el software, solicitudes de información, cambiar estados etc.

Vista: Es lo que maneja la visualización de la información

Controlador: Es el encargado de la lógica, controla las entradas y salidas de información, responde a los eventos y acciones que pasan.

Después de las definiciones que leímos podemos decir que el controlador es el que se encarga de mantener el orden en el software, tendremos reglas y acciones que debemos de ejecutar, al ser el que mantiene el orden queda claro que coordina las peticiones que van sucediendo como actualizar.

El modelo es el encargado de las consultas, por ejemplo si tienes una tienda online y quieres ver los productos más baratos de determinada categoría, (llamémosle categoría X), entonces el modelo trae los datos de la categoría X que corresponden con el filtro que le han pasado, en este caso el precio más bajo. Si por ejemplo una web hace una paginación de los resultados al momento de dar click a otra "pagina" el modelo traerá los siguientes 10 o los 10 elementos previos.

La vista por su lado será la encargada de recibir la información que llega por el controlador y empezar a realizar el acomodo de los elementos y mostrarle al usuario, desde la posición de los elementos, los colores, el diseño y la semántica dentro de una web son correspondientes a la vista que se ha definido previamente.

Ahora vamos a describir el flujo normal de una compra por internet de cuando utilizamos este modelo:

El usuario abre una página web

El usuario da click a una opción de la página web, un botón para comprar.

La vista recibe el click, pero el controlador es el encargado de trabajar con las acciones como por ejemplo enviar o recibir información.

El controlador accede al modelo para por ejemplo efectuar el pago o la compra.

El controlador al tener la respuesta pasa a la vista para indicar que es lo que debe mostrar al usuario.

La vista entonces trabaja con lo que recibe del controlador para mostrar la pantalla correspondiente al usuario.

Esto lo podríamos detallar tan complejo como la cadena de ADN y hasta utilizar algunos patrones para aumentar algunos puntos débiles, pero la idea principal del artículo es explicar de la manera más sencilla el modelo MVC y espero haberlo logrado.

La figura 1 muestra la relación entre los módulos de MVC:

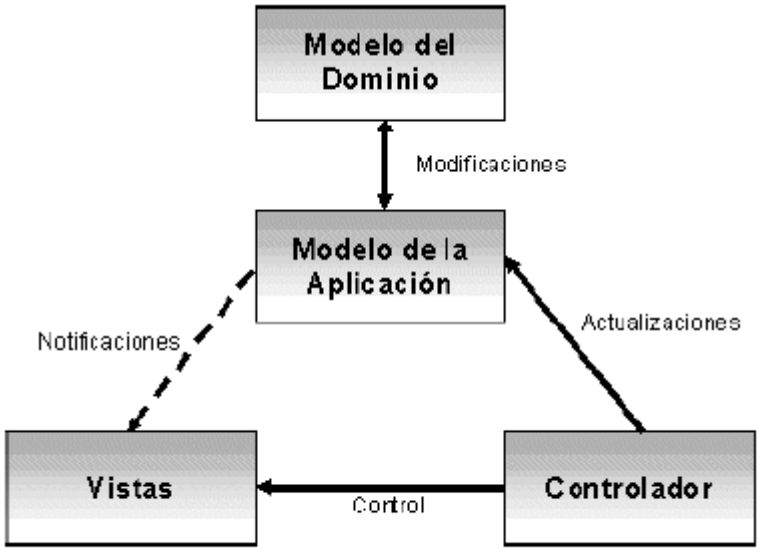


Figura 1: Relación entre los módulos del patrón MVC.

Ejemplo de cómo crear un sistema MVC pasó a paso (1)

Para poder entender las ventajas de utilizar el patrón MVC, vamos a transformar un script PHP en una aplicación que sigue la arquitectura MVC. Un buen ejemplo para ilustrar esta explicación es el de mostrar una lista con todos los libros dados de alta en la base de datos del proyecto.

Fichero index.php

```
<head>
 <title>LIBRERIA UAZON</title>
</head>
<body>
  <h1>Libros dados de alta en nuestra libreria</h1>
  <table border="1">
    <tr>
      <th>TITULO</th>
      <th>PRECIO</th>
    </tr>
    <?php
    $user='comprador';
    $password = 'proweb2013';
    $db = new PDO('mysql:host=localhost;dbname=uazon', $user, $password);
    $result = $db->query('SELECT titulo, precio FROM libros');
    while ($libro = $result->fetch())
    {
     ?>
     <tr>
      <td><?php echo $libro['titulo']?></td>
      <td><?php echo number_format($libro['precio'],2)?></td>
     </tr>
    <?php
    }
    ?>
    </table>
   </body>
</html>
```

El script anterior es fácil de escribir y rápido de ejecutar, pero muy difícil de mantener y actualizar. Los principales problemas del código anterior son:

- No existe protección frente a errores (¿qué ocurre si falla la conexión con la base de datos?).

- El código HTML y el código PHP están mezclados en el mismo archivo e incluso en algunas partes están entrelazados.

Aquí vemos las capas entremezcladas en el código

VISTA

```
<html>
   <head>
      <title>LIBRERIA UAZON</title>
   </head>
   <body>
     <h1>Libros dados de alta en nuestra libreria</h1>
     <table border="1">
        <tr>
           <th>TITULO</th>
           <th>PRECIO</th>
        </tr>
```

MODELO

```
<?php
   $db = new PDO('mysql:host=localhost;dbname=uazon', 'comprador', 'proweb2013');
   $result = $db->query('SELECT titulo, precio FROM libros');
   while ($libro = $result->fetch()) {
?>
```

VISTA

```
<tr>
   <td><?php echo $libro['titulo']?></td>
   <td><?php echo number_format($libro['precio'],2)?></td>
</tr>
<?php
}
?>
</table>
</body>
</html>
```

Problemas

Si quisiéramos presentar los mismos libros como RSS (otra vista) o PDF, deberíamos copiar también todo el bucle. Es, por supuesto, una duplicación del código responsable de la descarga de datos.

Intentemos ahora resolver los problemas identificados con ayuda del patrón MVC y separar el modelo de la vista.

El fragmento relacionado con la vista es fácil de separar en un fichero independiente. Podríamos incluir (include) el script correspondiente con la descripción del modo de presentación en función del equipo terminal en el cual vemos este servicio.

Lo primero es separar el código relacionado con el modelo del código relacionado con la vista.

En el fichero **index.php** movemos el modelo al inicio del fichero y la vista justo después.

```
<?php
   $user = 'comprador';
   $pwd = 'proweb2013';
   $db = new PDO('mysql:host=localhost;dbname=uazon', $user, $pwd);
   $result = $db->query('SELECT titulo, precio FROM libros');
   $libros = array();
   while ($libro = $result->fetch())
   {
      $libros[] = $libro;
   }
?>
<html>
   <head>
      <title>LIBRERIA UAZON</title>
   </head>
   <body>
      <h1>Libros dados de alta en nuestra libreria</h1>
      <table border="1">
         <tr>
            <th>TITULO</th>
            <th>PRECIO</th>
         </tr>
         <?php foreach ($libros as $libro): ?>
            <tr>
               <td><?php echo $libro['titulo'] ?></td>
               <td><?php echo number_format($libro['precio'],2) ?></td>
            </tr>
         <?php endforeach; ?>
      </table>
   </body>
</html>
```

De esta forma procedemos como nos recomienda el patrón MVC. Primero obtenemos los datos del modelo y posteriormente lo pasamos a la vista. El código HTML de vista contiene cierto código PHP utilizando la sintaxis alternativa de PHP que es ideal para las plantillas.

Más adelante hablaremos de lo que son plantillas pero suelen ser ficheros PHP que contienen código HTML entremezclado con código PHP (variables principalmente) y que se utilizan solamente para mostrar datos al usuario (Vista).

Una buena regla general para determinar si la parte de la vista está suficientemente limpia de código es que debería contener una cantidad mínima de código PHP, la suficiente como para que un diseñador HTML sin conocimientos de PHP pueda entenderla. Las instrucciones más comunes en la parte de la vista suelen ser **echo**, **if/endif**, **foreach/endforeach** y poco más. Además, no se deben incluir instrucciones PHP que generen etiquetas HTML.