UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

WEBMASTER

MVC aplicado a sitios web

Universidad Tecnológica Nacional - Facultad Regional Buenos Aires
Medrano 951 (C1179AAQ) C.A.B.A. Tel: (54 11) 4867-7511

Qué es MVC

MVC, Model - View - Controller o Modelo - Vista - Controlador un patrón de diseño de software para programación que propone separar el código de los programas por sus diferentes responsabilidades.

En líneas generales, MVC es una propuesta de diseño de software utilizada para implementar sistemas donde se requiere el uso de interfaces de usuario. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

Su fundamento

Es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores.

MVC es un "invento" que ya tiene varias décadas y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web.

MVC es útil para cualquier desarrollo en el que intervengan interfaces de usuario.

Por qué MVC

La rama de la ingeniería del software se preocupa por crear procesos que aseguren calidad en los programas que se realizan y esa calidad atiende a diversos parámetros que son deseables para todo desarrollo, como la estructuración de los programas o reutilización del código, lo que debe influir positivamente en la facilidad de desarrollo y el mantenimiento.

Los ingenieros del software se dedican a estudiar de qué manera se pueden mejorar los procesos de creación de software y una de las soluciones a las que han llegado es la arquitectura basada **en capas que separan el código** en función de sus responsabilidades o conceptos.

Por tanto, MVC nos ayuda a crear aplicaciones con **mayor calidad**.

Para que nos queden claras las **ventajas** del MVC podamos ver algunos ejemplos:

1. Ejemplo sencillo: el HTML y CSS. Al principio, en el HTML se mezclaba tanto el contenido como la presentación. Es decir, en el propio HTML tenemos etiquetas como "font" que sirven para definir las características de una fuente, o atributos como "bgcolor" que definen el color de un fondo. El resultado es que tanto el contenido como la presentación estaban juntos y si algún día pretendíamos cambiar la forma con la que se mostraba una página, estábamos obligados a cambiar cada uno de los archivos HTML que componen una web, tocando todas y cada una de las etiquetas que hay en el documento. Con el tiempo se observó que eso no era práctico y se creó el lenguaje CSS, en el que se separó la responsabilidad de aplicar el formato de una web.

2. Al escribir programas en lenguajes como PHP, cualquiera de nosotros comienza mezclando tanto el código PHP como el código HTML (e incluso el Javascript) en el mismo archivo. Esto produce lo que se denomina el "Código Espagueti". Si algún día pretendemos cambiar el modo en cómo queremos

que se muestre el contenido, estamos obligados a repasar todas y cada una de las páginas que tiene nuestro proyecto. Sería mucho más útil que el HTML estuviera separado del PHP.

3. Si queremos que en un equipo intervengan perfiles distintos de profesionales y trabajen de manera autónoma, como diseñadores o programadores, ambos tienen que tocar los mismos archivos y el diseñador se tiene necesariamente que relacionar con mucho código en un lenguaje de programación que puede no sea familiar, siendo que a éste quizás solo le interesan los bloques donde hay HTML. De nuevo, sería mucho más fácil la separación del código.

4. Durante la manipulación de datos en una aplicación es posible que estemos accediendo a los mismos datos en lugares distintos. Por ejemplo, podemos acceder a los datos de un artículo desde la página donde se muestra éste, la página donde se listan los artículos de un manual o la página de backend donde se administran los artículos de un sitio web. Si un día cambiamos los datos de los artículos (alteramos la tabla para añadir nuevos campos o cambiar los existentes porque las necesidades de nuestros artículos varían), estamos obligados a cambiar, página a página, todos los lugares donde se consumían datos de los artículos. Además, si tenemos el código de acceso a datos disperso por decenas de lugares, es posible que estemos repitiendo las mismas sentencias de acceso a esos datos y por tanto no estamos reutilizando código.

Son solo son simples ejemplos, habiendo decenas de casos similares en los que resultaría útil aplicar una arquitectura como el **MVC**, con la que nos obliguemos a separar nuestro código atendiendo a sus responsabilidades.

Ahora que ya podemos tener una idea de las ventajas que nos puede aportar el MVC, analicemos las diversas partes o conceptos en los que debemos separar el código de nuestras aplicaciones.

Modelos

Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una **base de datos**, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los **correspondientes selects, updates, inserts, etc.**

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

Vistas

Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos **permitirá renderizar los estados de nuestra aplicación en HTML**. En las vistas nada más tenemos los códigos **HTML y PHP** que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera.

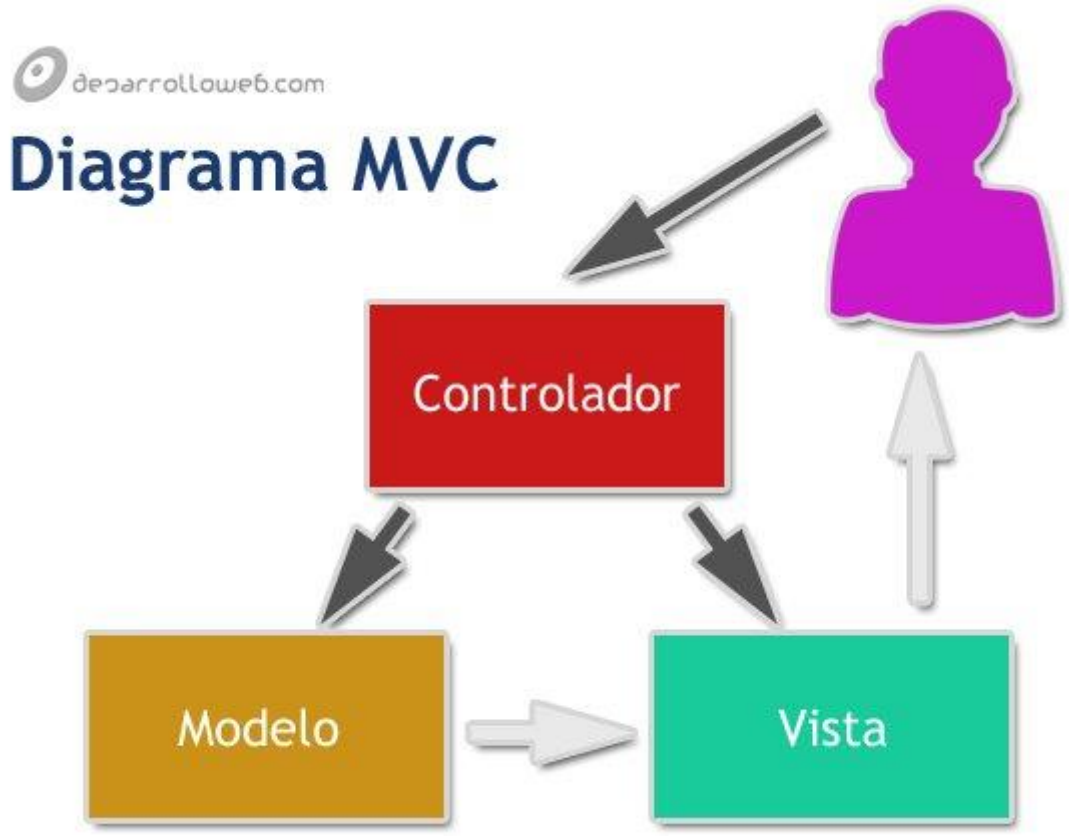
Controladores

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad es una capa que **sirve de enlace entre las vistas y los modelos**, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de **enlace entre los modelos y las vistas** para implementar las diversas necesidades del desarrollo.

Arquitectura de aplicaciones MVC

A continuación encontrarás un diagrama que te servirá para entender un poco mejor cómo colaboran las distintas capas que componen la arquitectura de desarrollo de software en el patrón MVC.



En esta imagen hemos representado con flechas los modos de colaboración entre los distintos elementos que formarían una aplicación MVC, junto con el usuario. Como se puede ver, **los controladores**, con su lógica de negocio, hacen de **puente** entre los modelos y las vistas. Pero además en algunos casos los modelos pueden enviar datos a las vistas. Veamos paso a paso cómo sería el flujo de trabajo característico en un esquema MVC.

1. **El usuario** realiza una solicitud a nuestro sitio web. Generalmente estará desencadenada por acceder a una página de nuestro sitio. Esa solicitud le llega al controlador.
2. **El controlador** comunica tanto con modelos como con vistas. A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica del negocio.

3. Para producir la salida, en ocasiones las vistas pueden solicitar más **información a los modelos**. En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre unos y otros.
4. **Las vistas** envían al usuario la salida. Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente, por eso he puesto la flecha en otro color.

Lógica de negocio / Lógica de la aplicación

Hay un concepto que se usa mucho cuando se explica el MVC que es la "**lógica de negocio".** Es un conjunto de reglas que se siguen en el software para reaccionar ante distintas situaciones. En una aplicación el usuario se comunica con el sistema por medio de una interfaz, pero cuando acciona esa interfaz para realizar acciones con el programa, se ejecutan una serie de procesos que se conocen como la lógica del negocio. Este es un concepto de desarrollo de software en general.

La lógica del negocio, aparte de marcar un comportamiento cuando ocurren cosas dentro de un software, también tiene normas sobre lo que se puede hacer y lo que no se puede hacer. Eso también se conoce como reglas del negocio. Bien, pues en el MVC la **lógica del negocio queda del lado de los modelos**. Ellos son los que deben saber cómo operar en diversas situaciones y las cosas que pueden permitir que ocurran en el proceso de ejecución de una aplicación.

Por ejemplo, pensemos en un sistema que implementa usuarios. Los usuarios pueden realizar comentarios. Pues si en un modelo nos piden eliminar un usuario nosotros debemos borrar todos los comentarios que ha realizado ese usuario también. Eso es una responsabilidad del modelo y forma parte de lo que se llama la lógica del negocio.

Nota: Si no queremos que esos comentarios se pierdan otra posibilidad sería mantener el registro del usuario en la tabla de usuario y únicamente borrar sus datos personales. Cambiaríamos el nombre del usuario por algo como "Jon Nadie" (o cualquier otra cosa), de modo que no perdamos la integridad referencial de la base de datos entre la tabla de comentario y la tabla de usuario (no debe haber comenarios con un id_usuario que luego no existe en la tabla de usuario). Esta otra lógica también forma parte de lo que se denomina lógica del negocio y se tiene que implementar en el modelo.

Incluso, en nuestra aplicación podría haber usuarios especiales, por ejemplo "administradores" y que no está permitido borrar, hasta que no les quitemos el rol de administrador. Eso también lo deberían controlar los modelos, realizando las comprobaciones necesarias antes de borrar efectivamente el usuario.

Sin embargo existe otro concepto que se usa en la terminología del MVC que es la **"lógica de aplicación**", que es algo que pertenece a **los controladores**. Por ejemplo, cuando me piden ver el resumen de datos de un usuario. Esa acción le llega al controlador, que tendrá que acceder al modelo del usuario para pedir sus datos. Luego llamará a la vista apropiada para poder mostrar esos datos del usuario. Si en el resumen del usuario queremos mostrar los artículos que ha publicado dentro de la aplicación, quizás el controlador tendrá que llamar al modelo de artículos, pedirle todos los publicados por ese usuario y con ese listado de artículos invocar a la vista correspondiente para mostrarlos. Todo ese conjunto de acciones que se realizan invocando métodos de los modelos y mandando datos a las vistas forman parte de la lógica de la aplicación.

Otro ejemplo:

Tenemos un sistema para borrar productos. Cuando se hace una solicitud a una página para borrar un producto de la base de datos, se pone en marcha un **controlador** que recibe el identificador del producto que se tiene que borrar. **Entonces le pide al modelo** que lo borre y a continuación se comprueba si el

modelo nos responde que se ha podido borrar o no. En caso que se haya borrado queremos mostrar una vista y en caso que no se haya borrado queremos mostrar otra. Este proceso también está en los controladores y lo podemos denominar como lógica de la aplicación.