**MVC (Model, View, Controller) explicado.**

Lo lees en 3 Min.

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](https://codigofacilito.com/cursos/rails), [Django](https://codigofacilito.com/cursos/django), [AngularJS](https://codigofacilito.com/cursos/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](https://codigofacilito.com/cursos/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 vez 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.

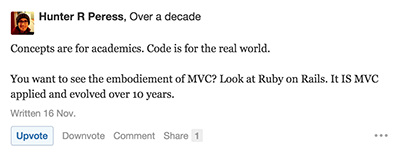
**Controlador**

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.

**Conclusión**



Si eres una persona práctica como [este usuario](http://www.quora.com/Hunter-R-Peress) en [quora](http://www.quora.com/Can-somebody-explain-clearly-MVC-architecture) tal vez quieras intentar con algún framework que utilice MVC. Mis recomendaciones **personales** son Ruby on Rails en el mejor de los casos, promueve muchas buenas prácticas, Laravel si lo tuyo es PHP, Django si lo tuyo es Python, [este tutorial](https://www.youtube.com/watch?v=tmkmh3eIakk) de @jorge\_vgut si lo tuyo es Android, AngularJS, EmberJS o Backbone si lo tuyo es javaScript en el fronted, SailJS o Express si eres más de NodeJS. No es necesario que **re escribas el hilo negro** y crees tu propio sistema MVC, allá afuera hay muchos buenos frameworks con años de desarrollo que aplican este patrón de diseño.

En mi experiencia, es importante que para que apliques los principios y buenas prácticas del MVC tengas un background fuerte en **programación orientada a objetos**.

Si tienes dudas **por favor déjalas en los comentarios** trataremos de solucionarlas :) Si te gusto el artículo, puedes compartirlo