<https://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design>

[https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador#Frameworks\_MVC](https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador%23Frameworks_MVC)

<https://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices>

<https://es.wikipedia.org/wiki/Middleware>

<https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93modelo_de_vista>

<https://www.netguru.com/blog/mvvm-architecture>

<https://builtin.com/software-engineering-perspectives/mvvm-architecture>

<https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/>

<https://medium.com/@onurcem.isik/introduction-to-mvvm-architecture-5c5558c3679>

<https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>

1. ¿Qué es MVC?

El Modelo-Vista-Controlador (MVC) es un patrón de diseño que separa una aplicación en tres componentes principales:​

Modelo: Representa la lógica de negocio y los datos de la aplicación. Se encarga de acceder a la base de datos y procesar la información, de controles de permisos y privilegios y demás gestión o almacenamiento de datos. Las peticiones de datos llegan al modelo desde la vista a través del controlador.

Vista: Es la interfaz de usuario. Muestra los datos proporcionados por el modelo y envía las acciones del usuario al controlador.​ Puede contener código que pertenezca a la lógica interna del controlador, véase ciertos condicionales, etc.

Controlador: Actúa como intermediario entre el modelo y la vista. Procesa las entradas del usuario, actualiza el modelo y refresca la vista.​ Puede incluso ejecutar comandos sobre la vista y, en algunos casos, sobre el modelo, se encarga de gestionar y responder a eventos que normalmente son acciones del usuario

Características:

Fomenta la separación de responsabilidades (cada parte, M, V o C, cada método o incluso cada instrucción sólo es responsable de su parte, lo demás le tiene que ser totalmente indiferente), facilitando el mantenimiento y escalabilidad de la aplicación.​ Se promueve la desconexión o el desacoplamiento de las tres partes. La vista y el modelo suelen permanecer al 100 indiferentes al otro, es decir, uno no conoce la existencia de otro. Para gestionar su comunicación se encarga el controlador. Eso no quiere decir, como hemos dicho antes, que la vista no tenga código lógico, aunque se intenta que sea el mínimo posible.

Permite que diferentes desarrolladores trabajen en el modelo, la vista y el controlador de manera independiente.​ Gracias a buscar una separación máxima entre las tres partes y al desacoplamiento que hemos mencionado se puede trabajar a la vez en cada una de las partes sin preocuparse demasiado lo que pasa en las otras dos, ya que el desarrollo de una no influye en la otra.

Mantenimiento: Al estar separados los componentes, es más sencillo realizar cambios sin afectar otras partes del sistema.​

Escalabilidad: Facilita la incorporación de nuevas funcionalidades sin complicaciones.​

El sistema posea una elevada complejidad Inicial, requiere una planificación detallada y puede ser más complejo de implementar al principio.​

La Vista suele contener más código porque recibe datos del modelo a través del controlador y también puede incluir algo de lógica de presentación.

No está 100% desacoplada del controlador ni del modelo. Puede tener funciones o condiciones para mostrar ciertos elementos, por ejemplo: if (user.isLoggedIn) { ... }.

Por eso, en muchos casos la vista en MVC mezcla HTML/JS o UI con lógica condicional, y eso puede dificultar su mantenimiento si se vuelve muy compleja.

Como dato a tener en cuenta, pero que no deberíamos de comentar mucho, es que este modelo tiene gran conexión con el patrón de diseño observador, que a su vez tiene relación con el patrón mediador y el singleton. No vamos a comentar demasiado. Simplemente se refiere, a que en casos de tener que crear relaciones de uno a muchos y una clase, objeto o tipo de datos necesita estar pendiente del estado de otros, para en caso de sufrir un cambio comunicárselo a todos los dependientes. Esto lo deberíamos de saber pero un poco por encima, no se si alguien ha elegido este patrón para desarrollarlo. Se pone mucho ejemplo en este patrón con ejemplos en java y Python, pero supongo que se usará en más.

Ejemplos de lenguajes o frameworks

Ruby on Rails, ASP.NET MVC, Django.

2. ¿Qué es MVVM?

El Modelo-Vista-Modelo de Vista (MVVM) es otro patrón de diseño que busca separar la lógica de presentación de la interfaz de usuario. Según he leído se entiende como una variante del MVC aunque otra gente lo considera una variante de otro patrón, el presentation model. En principio, es mucho más restrictivo a la hora de ejecutar el principio de responsabilidad. La vista no debe de tener nada de código lógico ni debe de conocer la existencia del modelo y viceversa. Para eso está el viewmodel.

Modelo: Al igual que en MVC, representa los datos y la lógica de negocio.​

Vista: Es la interfaz de usuario que muestra los datos.​

VistaModelo: Es una abstracción de la vista que expone datos y comandos que la vista puede enlazar. Facilita la comunicación entre la vista y el modelo.​

Utiliza el enlace de datos (data binding) para sincronizar automáticamente la vista con el modelo.​

Permite que diseñadores y desarrolladores trabajen de manera más independiente, ya que la lógica de presentación está desacoplada de la vista, de la misma forma que en el MVC. Aunque en el MVC la vista tiene, en principio algo más de código lógico o lógica de UI.

Cuando se va escalando el proyecto, según el creador, la complejidad puede llegar a ser abrumadora y, para aplicaciones pequeñas, considera este modelo con el adjetivo de overkill a la hora de desarrollar interfaces demasiado simples.

Debido al uso del databinding se considera difícil de debuggear, ya que entra más en la programación declarativa.

Desacoplamiento: La vista y la lógica de presentación están separadas, lo que facilita cambios y mantenimiento.​

Pruebas: Es más sencillo realizar pruebas unitarias de la lógica de presentación.​

Curva de Aprendizaje: Puede ser más difícil de entender e implementar correctamente al principio.​

La Vista está mucho más limpia. Su función principal es mostrar lo que el ViewModel le dice, usando data binding (enlazado de datos).

Toda la lógica de presentación (como decisiones sobre qué mostrar, validaciones, estados, etc.) se mueve al ViewModel.

Esto permite que la vista sea más simple, o sea, más fácil de mantener y de cambiar sin romper la lógica.

El objetivo es que modelo, vista y ViewModel estén lo más desacoplados posible. Cada uno tiene responsabilidades muy claras.

Angular, react con hooks, vue.js

Diferencias Principales entre MVC y MVVM:

Comunicación entre Componentes: En MVC, el controlador maneja las interacciones y actualiza la vista. En MVVM, la vista y el vista-modelo se comunican principalmente a través de enlace de datos, el databinding mencionado hasta ahora, reduciendo la necesidad de código adicional.​

En el MVC la comunicación es en un solo sentido, el controlador tiene una referencia a la vista. En MVVM la comunicación es en dos sentidos entre la vista y el viewmodel gracias al databinding.

MVC es uno de los modelos más antiguos, mientras que MVVC es una variante mucho más moderna.

Complejidad y Casos de Uso: MVC es adecuado para aplicaciones más simples o cuando se requiere un control más directo sobre la interfaz. MVVM es más beneficioso en aplicaciones complejas con interfaces ricas, especialmente cuando se utiliza enlace de datos.​

Independencia de la Vista: MVVM permite un mayor grado de independencia entre la lógica de presentación y la interfaz de usuario, facilitando cambios en la vista sin afectar la lógica que está por detrás.​

La elección entre MVC y MVVM depende de las necesidades específicas del proyecto. MVC puede ser más adecuado para aplicaciones con interfaces menos complejas, mientras que MVVM ofrece ventajas en aplicaciones con interfaces de usuario más dinámicas y complejas, gracias a su enfoque en el enlace de datos y la separación de responsabilidades.

En MVC, la vista puede tener lógica de presentación y está más acoplada.

En MVVM, la vista es más liviana, más centrada en el diseño y mucho más desacoplada gracias al ViewModel.