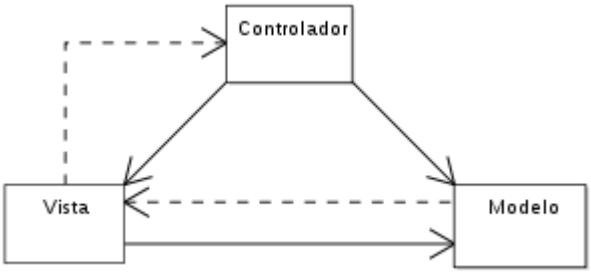
WIKIPEDIA
La enciclopedia libre

Modelo–vista–controlador

Modelo-vista-controlador (**MVC**) es un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**; es decir: por un lado define componentes para la representación de la información y, por otro lado, para la interacción del usuario.[1] [2] Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.[3] [4]



Un diagrama sencillo que muestra la relación entre el modelo, la vista y el controlador. Nota: las líneas sólidas indican una asociación directa, y principalmente las punteadas una indirecta (por ejemplo, patrón Observer).

Historia

El patrón MVC fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones.[5]

MVC fue introducido por Trygve Reenskaug (web personal (http://heim.ifi.uio.no/~trygver)) en Smalltalk-76 durante su visita a Xerox Parc[6] [7] en los años 70, seguidamente, en los años 80, Jim Althoff y otros implementaron una versión de MVC para la biblioteca de clases de Smalltalk-80.[8] Solo más tarde, en 1988, MVC se expresó como un concepto general en un artículo[9] sobre Smalltalk-80.

En esta primera definición de MVC el controlador se definía como «el módulo que se ocupa de la entrada» (de forma similar a como la vista «se ocupa de la salida»). Esta definición no tiene cabida en las aplicaciones modernas en las que esta funcionalidad es asumida por una combinación de la 'vista' y algún *framework* moderno para desarrollo. El 'controlador', en las aplicaciones modernas de la década de 2000, es un módulo o una sección intermedia de código, que hace de intermediario de la comunicación entre el 'modelo' y la 'vista', y unifica la validación (utilizando llamadas directas o el «observer» para desacoplar el 'modelo' de la 'vista' en el 'modelo' activo[10]).

Algunos aspectos del patrón MVC han evolucionado dando lugar a ciertas variantes del concepto original, ya que «las partes del MVC clásico realmente no tienen sentido para los clientes actuales»:[11]

- HMVC (MVC Jerárquico)
- MVA (Modelo-Vista-Adaptador)
- MVP (Modelo-Vista-Presentador)
- MVVM (Modelo-Vista Vista-Modelo)

- ... y otros que han adaptado MVC a diferentes contextos.

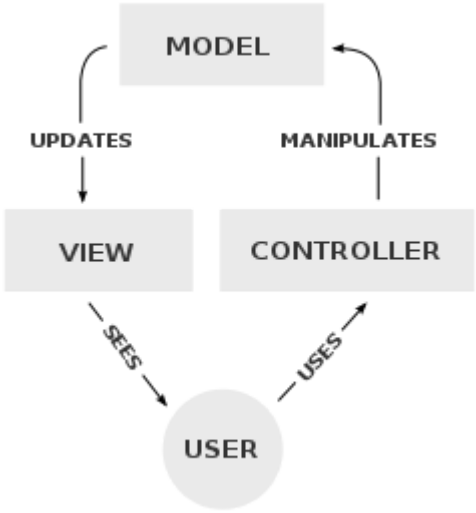
Descripción del patrón

De manera genérica, los componentes de MVC se podrían definir como sigue:

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.[1][2]
- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (véase ***Middleware***).
- La **Vista**: Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.

Interacción de los componentes

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo de control que se sigue generalmente es el siguiente:



Una típica colaboración entre los componentes de un MVC

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. Este uso del patrón Observador no es posible en las aplicaciones Web puesto que las clases de la vista están desconectadas del modelo y del controlador. En general el controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. *Nota: En algunas implementaciones la vista no tiene*

acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista. Por ejemplo en el MVC usado por Apple en su framework Cocoa. Suele citarse como Modelo-Interface-Control, una variación del MVC más puro

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente...

MVC y bases de datos

Muchos sistemas [informáticos] utilizan un sistema de gestión de base de datos el cual gestiona los datos que debe utilizar la aplicación; en líneas generales del **MVC** dicha gestión corresponde al modelo. La unión entre *capa de presentación* y *capa de negocio* conocido en el paradigma de la Programación por capas representaría la integración entre la **Vista** y su correspondiente **Controlador** de eventos y acceso a datos, MVC no pretende discriminar entre capa de negocio y capa de presentación pero sí pretende separar la capa *visual gráfica* de su correspondiente *programación y acceso a datos*, algo que mejora el desarrollo y mantenimiento de la *Vista* y el *Controlador* en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

Uso en aplicaciones Web

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de frameworks, comerciales y no comerciales, que implementan este patrón (ver apartado siguiente "*Frameworks MVC*"); estos frameworks se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor.[13]

Los primeros frameworks MVC para desarrollo web planteaban un enfoque de cliente ligero en el que casi todas las funciones, tanto de la vista, el modelo y el controlador recaían en el servidor. En este enfoque, el cliente manda la petición de cualquier hiperenlace o formulario al controlador y después recibe de la vista una página completa y actualizada (u otro documento); tanto el modelo como el controlador (y buena parte de la vista) están completamente alojados en el servidor. Como

las tecnologías web han madurado, ahora existen frameworks como JavaScriptMVC, Backbone o jQuery[14] que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente (véase AJAX).

Frameworks MVC

Lenguaje	Licencia	Nombre
.NET	Castle Project (http://www.castleproject.org/index.html)	MonoRail (http://www.castleproject.org/projects/monorail/)
.NET	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Spring.NET (http://www.springframework.net/)
.NET	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Maverick.NET (https://mavnet.sourceforge.net/)
.NET	MS-PL (http://www.opensource.org/licenses/ms-pl.html)	ASP.NET MVC (https://www.asp.net/mvc)
.NET	Microsoft Patterns & Practices (http://msdn.microsoft.com/practices/)	User Interface Process (UIP) Application Block (https://web.archive.org/web/20070219022606/http://msdn.microsoft.com/practices/compcat/default.aspx?pull=%2Flibrary%2Fen-us%2Fdnpag%2Fhtml%2Fuipab.asp)
C++	BSD license (https://web.archive.org/web/20170521001531/http://treefrogframework.github.io/treefrog-framework/user-guide/en/introduction/)	treefrog (http://www.treefrogframework.org)
Delphi/ Object Pascal	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Delphi MVC Framework (https://github.com/danieleteti/delphimvcframework)
Delphi/ Object Pascal	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	mORMot Framework (https://synopse.info/fossil/wiki/Synopse+OpenSource)
Objective C	Apple (http://developer.apple.com)	Cocoa
Ruby	MIT (http://opensource.org/licenses/mit-license.php)	Ruby on Rails (http://rubyonrails.org/)
Ruby	MIT (http://opensource.org/licenses/mit-license.php)	Merb
Ruby	MIT (http://opensource.org/licenses/mit-license.php)	Ramaze
Ruby	MIT (http://opensource.org/licenses/mit-license.php)	Rhodes
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Grails
Java	GPL (http://www.opensource.org/licenses/gpl-license.php)	Interface Java Objects (http://nt.tusoporte.es/IjoProject) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/http://nt.tusoporte.es/IjoProject), la primera versión (https://web.archive.org/web/1/http://nt.tusoporte.es/IjoProject) y la última (https://web.archive.org/web/2/http://nt.tusoporte.es/IjoProject)).
Java	LGPL (http://www.martincordova.com)	Framework Dinámica
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Struts
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Brutos framework
Java	Apache (http://beehive.apache.org/)	Beehive

Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Spring
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Tapestry (http://tapestry.apache.org/)
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Aurora (https://web.archive.org/web/20131230132142/http://auroramvc.org/)
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	JavaServerFaces (http://java.sun.com/javaee/javaserverfaces/)
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	PrimeFaces (https://www.primefaces.org/#primefaces)
Java	Apache (http://www.apache.org/licenses/LICENSE-2.0.html)	Vaadin (https://web.archive.org/web/20170511162553/https://vaadin.com/home/)
JavaScript	GPLv3 (http://www.gnu.org/copyleft/gpl.html)	Sails.JS (https://web.archive.org/web/20161118215029/http://sailsjs.org/)
JavaScript	GPLv3 (http://www.gnu.org/copyleft/gpl.html)	ExtJS 4 (http://www.sencha.com/products/extjs/)
JavaScript	MIT (https://github.com/angular/angular.js/blob/master/LICENSE)	AngularJS (https://angularjs.org/)
JavaScript	MIT (https://github.com/kamilmysliwiec/nest/blob/master/LICENSE)	Nest (https://github.com/kamilmysliwiec/nest)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	Mojolicious (http://mojolicio.us/)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	Catalyst (http://www.catalystframework.org/)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	CGI::Application (https://web.archive.org/web/20071229000621/http://cgiapp.erlbaum.net/)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	Gantry Framework (https://web.archive.org/web/20071229000533/http://www.usegantry.org/)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	Jifty (http://jifty.org/view/HomePage)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	Maypole (https://web.archive.org/web/20091117133112/http://maypole.perl.org/)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	OpenInteract2 (http://www.openinteract.org/)
Perl	Comercial	PageKit (http://pagekit.org/)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	Cyclone 3 (https://web.archive.org/web/20071215103304/http://www.cyclone3.org/home)
Perl	GPL (http://www.opensource.org/licenses/gpl-license.php)	CGI::Builder (http://search.cpan.org/perldoc?CGI::Builder)
PHP	GPL (http://www.opensource.org/licenses/gpl-license.php)	BitPHP (https://github.com/bitphp)
PHP	BSD (https://github.com/phalcon/cphalcon/blob/master/LICENSE.txt)	phalcon (https://phalcon.io/en-us)
PHP	MIT (http://opensource.org/licenses/mit-license.php)	Laravel (http://laravel.com/)
PHP	GPL (http://www.opensource.org/licenses/gpl-license.php)	Self Framework (php5, MVC, ORM, Templates, I18N, Múltiples DB) (https://web.archive.org/web/20111020013226/http://sites.google.com/site/phpframeworkpoo/download-framework)

PHP	LGPL (http://www.opensource.org/licenses/lgpl-license.php)	ZanPHP (https://web.archive.org/web/20110926095451/http://www.zanphp.com/)
PHP	LGPL (http://www.opensource.org/licenses/lgpl-license.php)	Tlalokes (https://web.archive.org/web/20090415200934/http://tlalokes.org/)
PHP	GPL (http://www.siaempresarial.com/mvc/mvc-license.php) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/http://www.siaempresarial.com/mvc/mvc-license.php), la primera versión (https://web.archive.org/web/1/http://www.siaempresarial.com/mvc/mvc-license.php) y la última (https://web.archive.org/web/2/http://www.siaempresarial.com/mvc/mvc-license.php)).	SiaMVC (http://siaempresarial.com/siamvc/) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/http://siaempresarial.com/siamvc/), la primera versión (https://web.archive.org/web/1/http://siaempresarial.com/siamvc/) y la última (https://web.archive.org/web/2/http://siaempresarial.com/siamvc/)).
PHP	LGPL (http://www.opensource.org/licenses/lgpl-license.php)	Agavi (https://web.archive.org/web/20080318151307/http://www.agavi.org/)
PHP	BSD (http://www.opensource.org/licenses/bsd-license.php)	Zend Framework (http://framework.zend.com/), proyecto continuado como Laminas Framework (https://getlaminas.org/)
PHP	MIT (http://www.opensource.org/licenses/mit-license.php)	CakePHP (http://www.cakephp.org/)
PHP	GNU/GPL (http://www.opensource.org/licenses/gpl-license.php)	KumbiaPHP (http://www.kumbiaphp.com/)
PHP	MIT (http://www.symfony-project.org/license)	Symfony (http://www.symfony.com/)
PHP	MIT (http://www.symfony-project.com/licenses/mit-license.php) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/http://www.symfony-project.com/licenses/mit-license.php), la primera versión (https://web.archive.org/web/1/http://www.symfony-project.com/licenses/mit-license.php) y la última (https://web.archive.org/web/2/http://www.symfony-project.com/licenses/mit-license.php)).	QCodo (http://qcodo.com/)
PHP	GNU/GPL (http://www.opensource.org/licenses/gpl-license.php)	CodeIgniter (http://codeigniter.com/)
PHP	GNU/GPL (http://www.opensource.org/licenses/gpl-license.php)	Polka-PHP (https://github.com/rotela/polka-php)
PHP	BSD (http://kohanaframework.org/)	Kohana (http://kohanaframework.org/)
PHP	MPL 1.1 (https://web.archive.org/web/20090103001154/http://php4e.codeman.cl/index.php?SECCION=msg&COD=59&MODULE=home)	PHP4ECore (https://web.archive.org/web/20080417032603/http://php4e.codeman.cl/)
PHP	GNU (http://www.practico.org/)	Practico (http://www.practico.org/)
PHP	GNU (https://web.archive.org/web/20100216021409/http://flavorphp.com/)	FlavorPHP (https://web.archive.org/web/20100216021409/http://flavorphp.com/)
PHP	Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0)	Yupp PHP Framework (https://code.google.com/p/yupp/)
PHP	BSD (http://www.yiiframework.com/license/)	Yii PHP Framework (http://www.yiiframework.com/)
PHP	GPL (http://www.opensource.org/licenses/gpl-license.php)	Osezno PHP Framework
PHP	MIT (https://github.com/alrik11es/sPHPf/blob/master/license.txt) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/https://github.com/alrik11es/sPHPf/blob/master/license.txt), la primera versión (https://web.archive.org/web/1/https://github.com/alrik11es/sPHPf/blob/master/license.txt) y la última (https://web.archive.org/web/2/https://github.com/alrik11es/sPHPf/blob/master/license.txt)).	(sPHPf) Simple PHP Framework (https://web.archive.org/web/20120111161846/http://sphpf.coldstarstudios.com/)

PHP	GNU/GPL (http://www.gvpontis.gva.es/cast/gvhidra-herramienta/gvhidra-que-es-gvhidra/) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/http://www.gvpontis.gva.es/cast/gvhidra-herramienta/gvhidra-que-es-gvhidra/), la primera versión (https://web.archive.org/web/1/http://www.gvpontis.gva.es/cast/gvhidra-herramienta/gvhidra-que-es-gvhidra/) y la última (https://web.archive.org/web/2/http://www.gvpontis.gva.es/cast/gvhidra-herramienta/gvhidra-que-es-gvhidra/)).	gvHidra (http://www.gvhidra.org) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/http://www.gvhidra.org/), la primera versión (https://web.archive.org/web/1/http://www.gvhidra.org/) y la última (https://web.archive.org/web/2/http://www.gvhidra.org/)).
Python	ZPL (https://web.archive.org/web/20060424193351/http://www.zope.org/Resources/License/)	Zope3 (https://web.archive.org/web/20070105081737/http://wiki.zope.org/zope3/MVC)
Python	Varias (http://docs.turbogears.org/1.0/License)	Turbogears (http://www.turbogears.org/)
Python	GPL (http://web2py.com/examples/default/license)	Web2py (http://web2py.com/)
Python	BSD (https://web.archive.org/web/20090525033300/http://pylonshq.com/project/pylonshq/browser/LICENSE)	Pylons (https://web.archive.org/web/20061205035643/http://pylonshq.com/)
Python	BSD (https://code.djangoproject.com/browser/django/trunk/LICENSE)	Django
AS3	Adobe Open Source (https://web.archive.org/web/20080826010704/http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm)	Cairngorm (https://web.archive.org/web/20080826010704/http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm)
AS3 y Flex	MIT License (http://www.opensource.org/licenses/mit-license.php)	CycleFramework (https://code.google.com/p/cycleframework/)

Véase también

- Ingeniería Técnica en Informática de Gestión
- Ingeniería de Software

Referencias

1. "More deeply, the framework exists to separate the representation of information from user interaction." The DCI Architecture: A New Vision of Object-Oriented Programming (http://www.artima.com/articles/dci_vision.html) Archivado (https://web.archive.org/web/20170929001352/http://www.artima.com/articles/dci_vision.html) el 29 de septiembre de 2017 en Wayback Machine. - Trygve Reenskaug and James Coplien - March 20, 2009.
2. "... the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of object." Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller (MVC) (https://web.archive.org/web/20120429161935/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html).
3. «Simple Example of MVC (Model View Controller) Design Pattern for Abstraction» (http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design).
4. «Best MVC Practices» (http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices).
5. http://c2.com/cgi/wiki?ModelViewControllerHistory Historia del Modelo-Vista-Controlador
6. Notes and Historical documents (http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html) from Trygve Reenskaug, inventor of MVC.
7. "A note on DynaBook requirements", Trygve Reenskaug, 22 March 1979, SysReq.pdf (http://folk.uio.no/trygver/1979/sysreq/SysReq.pdf).
8. How to use Model-View-Controller (MVC) (https://web.archive.org/web/20090801040629/http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html)
9. Krasner, Glenn E.; Stephen T. Pope (Aug/Sep de 1988). «A cookbook for using the model-view controller user interface paradigm in Smalltalk-80» (http://dl.acm.org/citation.cfm?id=50757.50759). *The JOT* (SIGS Publications). Also published as "A Description of the Model-View-

MVC

Introducción y conceptos #1

Programar utilizando MVC consiste en separar la aplicación en tres partes principales.

El *modelo* representa los datos de la aplicación, la *vista* hace una presentación del modelo de datos, y el controlador maneja y enruta las peticiones [requests] hechas por los usuarios.

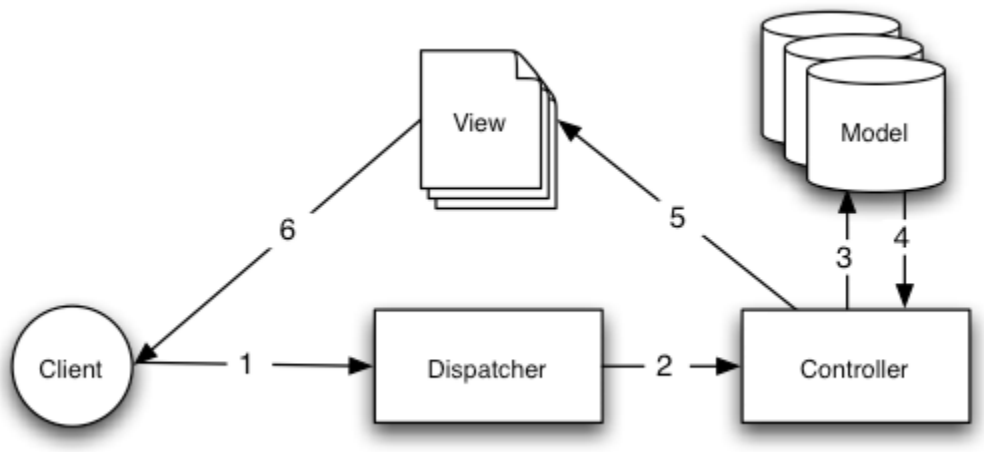


Figura 1

La figura 1 muestra un ejemplo sencillo de una petición [request] MVC.

A efectos ilustrativos, supongamos que un usuario llamado Ricardo acaba de hacer clic en el enlace "¡Comprar un pastel personalizado ahora!" de la página inicial de la aplicación.

1. Ricardo hace clic en el enlace apuntando a http://www.ejemplo.com/pasteles/comprar, y su navegador hace una petición al servidor web.
2. El despachador comprueba la URL de la petición (/pasteles/comprar), y le pasa la petición al controlador adecuado.
3. El controlador realiza lógica de aplicación específica. Por ejemplo, puede comprobar si Ricardo ha iniciado sesión.
4. El controlador también utiliza modelos para acceder a los datos de la aplicación. La mayoría de las veces los modelos representan tablas de una base de datos. En este ejemplo, el controlador utiliza un modelo para buscar la última compra de Ricardo en la base de datos.
5. Una vez que el controlador ha hecho su magia en los datos, se los pasa a la vista. La vista toma los datos y los deja listos para su presentación al usuario.
   La mayoría de las veces las vistas vienen en formato HTML, pero una vista puede ser fácilmente un PDF, un documento XML, o un objeto JSON, dependiendo de tus necesidades.
6. Una vez que el objeto encargado de procesar vistas ha utilizado los datos del modelo para construir una vista completa, el contenido se devuelve al navegador de Ricardo.

Casi todas las peticiones a tu aplicación seguirán este patrón básico.

Beneficios

¿Por qué utilizar MVC? Porque es un patrón de diseño de software probado y se sabe que funciona. Con MVC la aplicación se puede desarrollar rápidamente, de forma modular y mantenible.
Separar las funciones de la aplicación en *modelos*, *vistas* y *controladores* hace que la aplicación sea muy ligera. Estas características nuevas se añaden fácilmente y las antiguas toman automáticamente una forma nueva.

El diseño modular permite a los diseñadores y a los desarrolladores trabajar conjuntamente, así como realizar rápidamente el prototipado.
Esta separación también permite hacer cambios en una parte de la aplicación sin que las demás se vean afectadas.

Última modificación: Monday, 25 de March de 2019, 09:14

MVC

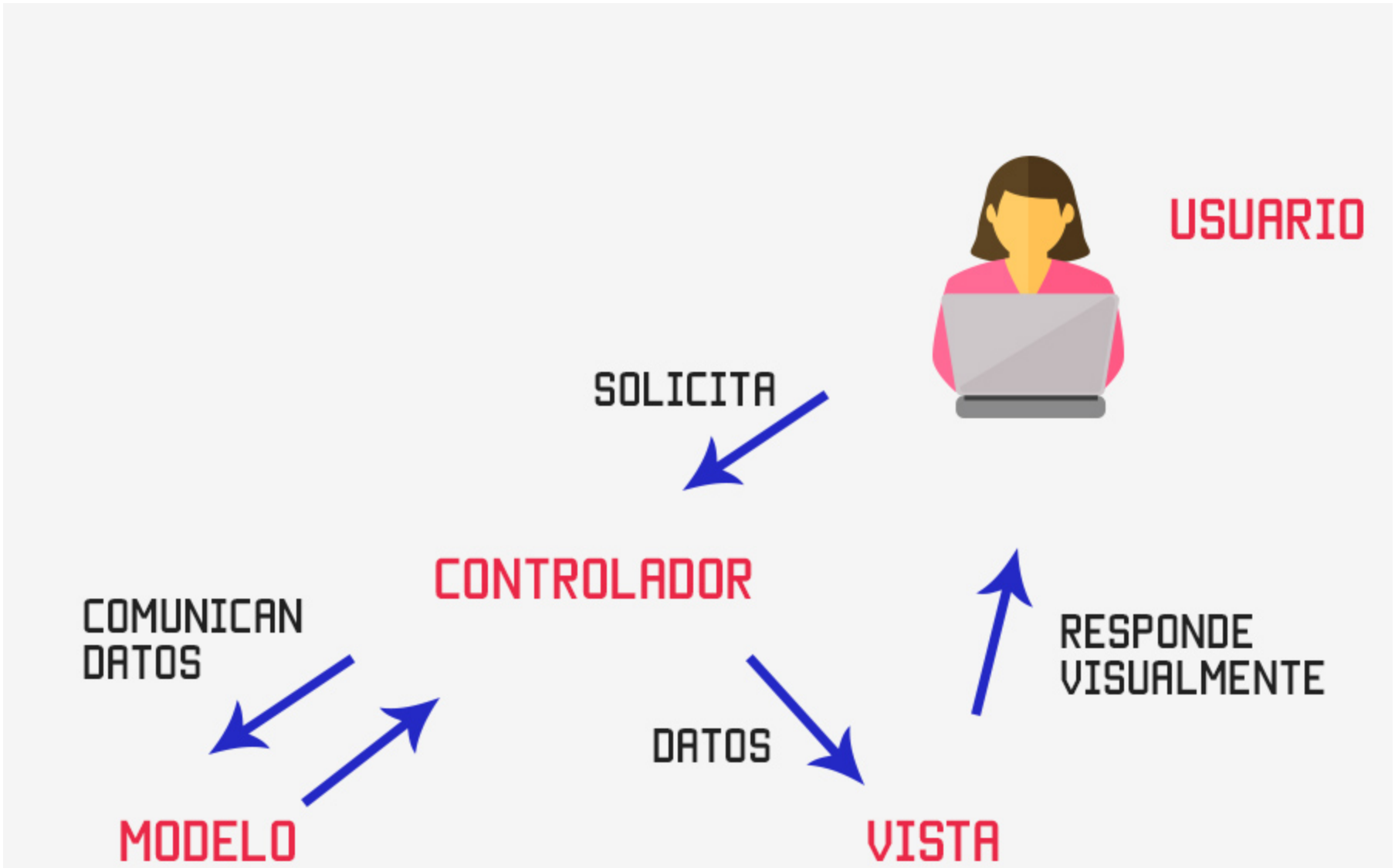
Introducción y conceptos #2

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.

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 aplicación, 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 aplicación, 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.

Última modificación: Monday, 25 de March de 2019, 09:18

◄ Introducción y conceptos #1

Ir a...

Presentación MVC ►