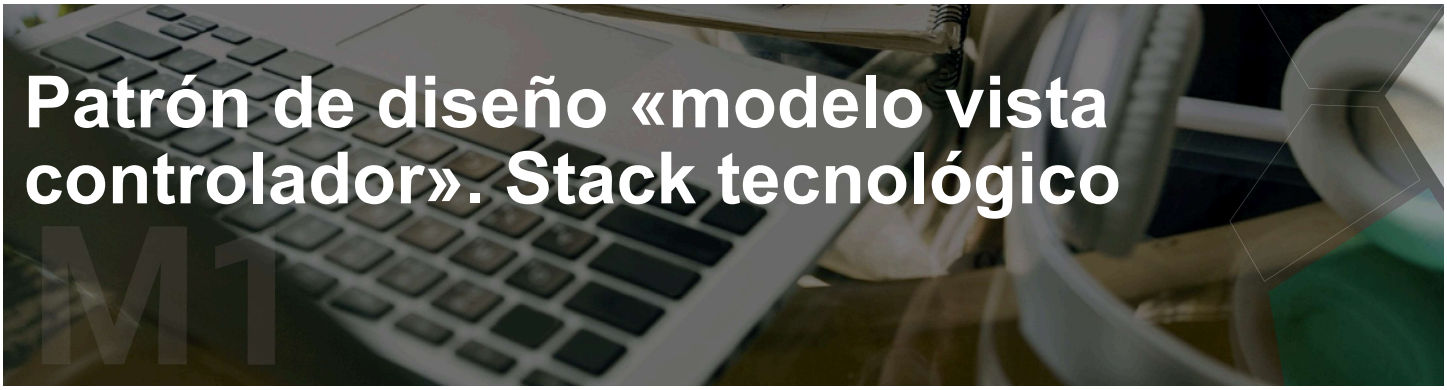


Patrón de diseño «modelo vista controlador». Stack tecnológico

M1

Introducción

El lenguaje de programación PHP por sí solo tiende al caos: código «espagueti», poca mantenibilidad y muchos dolores de cabeza. En esta lectura, conoceremos MVC (modelo-vista-controlador), un patrón de diseño de software para implementar interfaces de usuario, datos y lógica de control. La mayoría de los frameworks utilizados en desarrollo web se basan en dicho patrón, y también lo hace el seleccionado para esta materia: Laravel. Repasaremos algunas características del lenguaje de programación, PHP, conoceremos el resto del stack tecnológico y las implicancias de las tecnologías open source y de software libre escogidas.

1. PHP

Según la documentación oficial, **"PHP (acrónimo recursivo de PHP, hypertext preprocessor) es un lenguaje de código abierto muy popular, especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML**" (PHP, s.f., https://lc.cx/SB42rX).

La incipiente versión de PHP nació en 1994, como un proyecto personal llamado **Personal Home Page Tools.** Luego de algunas versiones, *se tornó un lenguaje mucho más consistente, confiable e incorporó la posibilidad de usarlo orientado a objetos*. En el 2020, PHP se encontraba en su versión 7.

> Si bien es un lenguaje que —sin un framework robusto— tiende al código «espagueti», es decir, con flujos complejos, enredados, con alto nivel de acoplamiento y difícil de mantener, existen en el mercado diferentes proyectos que proponen formas de trabajo más ordenadas, resultando en excelentes soluciones.

PHP resulta muy conveniente tanto para **scripting** como para **el desarrollo de código web a interpretarse en el servidor**, es decir, del lado del **backend**, por una herramienta que funcione de servidor web, como **Apache.** Es ***multiplataforma***, lo que implica que posee compatibilidad con la mayoría de sistemas operativos y servidores webs**. Es un lenguaje interpretado, que no pasa por un proceso previo de compilación.** Hasta la versión 6, era débilmente tipado; pero, a partir

de la versión 7, incorpora la posibilidad de predeterminar el tipo de parámetros que reciben los métodos, así como también sus retornos.

Para tener un servidor web sólido, robusto, estable y con un buen respaldo, **se suele utilizar el conjunto de herramientas con acrónimo LAMP:**

- Linux
- Apache
- MySQL
- PHP

Estas han demostrado ser **muy sencillas de instalar**, **open source** (ya ahondaremos sobre ello), **con amplia comunidad** y **muy compatibles entre sí.** A los grupos de herramientas que se escogen para una solución se los denomina ***stacks tecnológicos***. Es recomendable utilizar stacks tecnológicos que ya hayan sido ampliamente probados en conjunto.

MVC

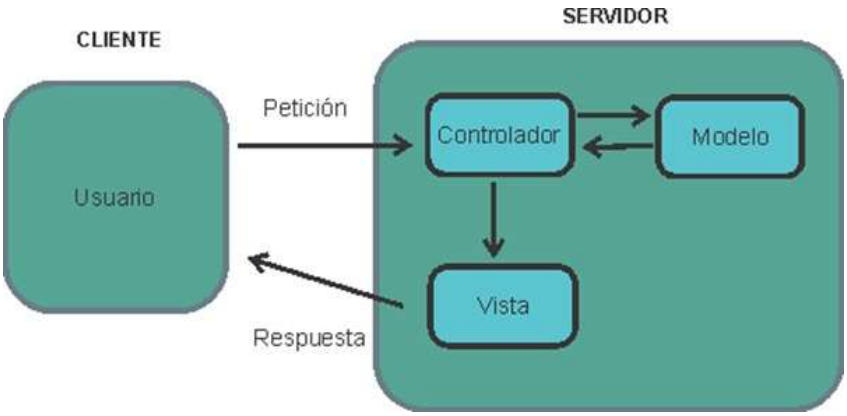
Como ya vimos, las siglas MVC significa «modelo-vista-controlador» (en inglés coinciden, model-view-controller). Este patrón de diseño de software propone que la lógica de negocios de nuestro proyecto esté bien separada de la visualización de los datos, lo cual nos facilita la división de tareas, la detección de errores y el mantenimiento del código.

Algunos otros patrones de diseño se basan en **MVC**, como **MVVM (modelo-vista-modelo de vista), MVP (modelo-vista-presentador)** y **MVW (modelo-vista-whatever).** Nos quedaremos con MVC, para entender la conceptualización que sugiere y, más adelante, ver cómo se implementa en el **framework Laravel.**

Abordemos, entonces, los tres elementos de MVC:

1. **Modelo:** maneja datos y lógica de negocios.
2. **Vista:** se encarga del diseño y presentación.
3. **Controlador:** enruta comandos a los modelos y vistas.

Figura 1. Esquema MVC

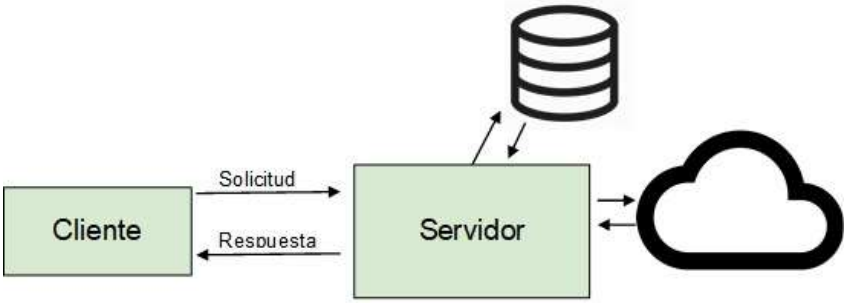


Fuente: elaboración propia

¿Cómo funciona esto en un desarrollo web del lado del servidor (lo que se conoce como backend)?

> Sin profundizar aún en este patrón, por la arquitectura cliente-servidor, ante la llegada de una nueva solicitud al servidor, con una URI y método HTTP, sucede «algo» en el servidor, y este elabora una respuesta para el cliente, comúnmente con un archivo HTML. Ese «algo» puede ser la simple obtención de un archivo HTML estático, disponible en el servidor, o puede implicar la consulta a una o más bases de datos, servicios webs, otros servicios, etcétera. Pero, finalmente, elabora una respuesta que también involucra, frecuentemente, un archivo HTML y lo retorna al cliente.

Figura 2. Solicitud y respuesta en arquitectura cliente-servidor



Fuente: elaboración propia

Cuando se utiliza el patrón de diseño MVC, **¿cómo funciona ese procedimiento del lado del servidor?** Cada vez que llega una nueva solicitud al servidor, dependiendo de la ruta solicitada (**URI y método HTTP**), esta es derivada al controlador que le corresponda. **El controlador consulta a el o los modelos necesarios, que le devolverán los datos que le permitirán construir la vista.** Luego, el controlador pasa dichos datos a la vista, que se encarga de acomodar la información en un archivo HTML, por ejemplo, y es devuelto al cliente.

Veamos un ejemplo. Recordemos el caso de la lectura 1 de este módulo: la editorial La Siglo y su panel de administración para la gestión de los libros.

A través del cliente, **un usuario inicia sesión** (dejaremos eso para más adelante) **e ingresa al panel de administración**. Lo primero que ve es un listado de libros. *¿Qué sucedió para que pueda ver en su browser esa tabla con columnas y filas con el nombre, código y precio de libros, con botones que llevan a ver el detalle, modificar o eliminar cada uno de ellos?*

El cliente accede a la URI/libros con el método GET. El servidor tiene configurado que todas las rutas que comiencen con «/libros», son derivadas al controlador «libros». De modo que llega a dicho controlador un método específico. Este tiene escrito en el código que, cuando le pidan obtener los libros, debe pedirle al modelo de «Libro» que obtenga todos los que existan y se los entregue ordenados.

El modelo «**Libro**» conoce la lógica de negocio, sabe que los libros tienen baja lógica, por lo cual se encargará de omitir de su respuesta aquellos que estén dados de baja. Además, sabe ordenar los libros con los distintos criterios disponibles. **Acude a la base de datos con esta información, obtiene las filas necesarias, las transforma en datos que a la aplicación le sirvan (básicamente, en objetos) y devuelve el conjunto de datos al controlador**.

El controlador no tuvo que involucrarse para saber de dónde se obtienen los datos, ni cómo se transforman, ni qué lógicas de negocio se han aplicado. Asimismo, si hubiera un error, ya sabríamos a dónde ir a buscarlo en el código: en el modelo «Libro». **El controlador envía estos datos a la vista. La vista se encargará de maquetar cómo se van, valga la redundancia, a visualizar los datos: armar los encabezados, la tabla, cada elemento con su estilo y estética, de modo que resulten en una mejor experiencia para el usuario.**

La vista es la encargada de hacer una repetición: por cada libro que le pasó el controlador, va a mostrar una nueva fila de datos, con nombre, código, precio del libro y botones que lleven a las rutas para ver los detalles, modificar o eliminar el libro en cuestión. También, **la vista se encargará de mostrar, en una barra lateral, el nombre de usuario que ha iniciado sesión y un botón para cerrar dicha sesión.** Podría tener alguna *estructura condicional;* por ejemplo, si el usuario tiene determinados permisos exclusivos, le muestra el botón para hacer un cambio masivo de precios y, si el usuario no tiene dicho permiso, no lo muestra.

La lógica de negocio, es decir, la decisión de si puede o no puede hacerlo, está en el modelo (en ese caso, en el modelo «Usuario»); pero la lógica de visualización (si puede, lo muestra, si no puede, no), está en la vista. Una vez que todo el código de la vista ha sido reemplazado por los datos correspondientes, se genera un archivo HTML (junto con algunos archivos

adicionales que veremos más adelante, como.js y .css). Con esos archivos, se arma la respuesta HTTP que se envía al cliente.

Cabe destacar que las diferentes implementaciones, dadas las características y decisiones en los **diferentes lenguajes, frameworks** y **desarrolladores**, pueden variar levemente respecto a las formas de usar el patrón *MVC*. Lo que se ha abordado aquí tiene que ver con una de las formas de uso bastante frecuente en **desarrollo web** y, en particular, en el **stack tecnológico** seleccionado para la materia.

Laravel

Laravel es un **framework** —*marco de trabajo, en español*— **para el desarrollo de aplicaciones y servicios webs en PHP.** Es de código abierto, significado que abordaremos más adelante. Laravel *propone una forma de organizar el código PHP que resulta simple*, está basado en el patrón de diseño MVC, con bastante influencia de otros **frameworks**, como **Ruby on Rails, Sinatra** y **ASP.NET MVC**. La primera versión fue lanzada en 2011, pero sigue teniendo frecuentes actualizaciones; además de una comunidad muy amplia y colaborativa. Estos son factores que nos ayudan mucho a la hora de elegir una tecnología, **tanto con fines pedagógicos como para un proyecto**.

Si bien no tiene demasiados requerimientos de software para funcionar, y al estar construido en PHP no requiere de una máquina virtual para ejecutarse per sé (abordaremos esto a continuación), **sus creadores recomiendan la instalación de Laravel Homestead, un entorno completo virtualizado.** Resulta muy práctico seguir este camino, y está disponible para Windows, Linux y MacOS.

Homestead funciona sobre alguna herramienta (como VirtualBox) que nos permite crear máquinas virtuales. Además, utiliza Vagrant, que es una herramienta que nos agiliza la creación y provisión de máquinas virtuales.

Las dependencias del proyecto en Laravel se instalan y actualizan a través de una herramienta llamada Composer, que es externa a Laravel. Dentro de la máquina virtual de Laravel Homestead, ya está instalada Composer. Usar un gestor de dependencias es muy recomendable, siempre que se desarrolle un proyecto con un stack tecnológico medianamente complejo; ya que, en caso contrario, deberíamos hacer verificaciones de versiones, dependencias e instalación de cada una herramienta de forma manual. Esto es un requisito si usamos determinados frameworks, como Laravel.

Referencias

PHP, (s.f.). *¿Qué es PHP?* https://www.php.net/manual/es/intro-whatis.php