Sesión 1. Introducción a los frameworks PHP y a Laravel

1. Frameworks PHP

Un **framework** es una herramienta que proporciona una serie de módulos que ayudan a organizar y desarrollar un producto software. En el caso concreto de los frameworks PHP, la mayoría de ellos proporcionan una serie de comandos o herramientas para crear proyectos con una estructura determinada (normalmente, siguiendo el patrón MVC que veremos después), de forma que ya dan una base de trabajo hecha, y facilidades para poder crear el modelo de datos, la conexión a la base de datos, las rutas de las diferentes secciones de la aplicación, etc.

1.1. Ejemplos de frameworks PHP

Actualmente existe una gran variedad de frameworks PHP que elegir para desarrollar nuestras aplicaciones. Algunos de los más populares son:

- **Laravel**, un framework relativamente reciente (fue creado en 2011), y que ha ganado bastante popularidad en los últimos años. Su filosofía es el poder desarrollar proyectos de forma elegante y simple. Cuenta con una amplia comunidad de soporte detrás, y se le augura un futuro bastante consolidado.
- **Symfony**, creado en 2005, cuenta con más camino hecho que Laravel, y una estructura más consolidada. En sus primeras versiones se presentaba como un framework más monolítico (se instalaban demasiados módulos que luego no necesitábamos), pero recientemente ha adaptado su estructura para hacerla más modular.
- **CodeIgniter**, un framework más ligero que los anteriores, pero también con un amplio grupo de seguidores y desarrolladores. Fue creado en 2006 y, aunque ha sufrido una etapa de abandono, ha vuelto a coger fuerza en los últimos años, quizá debido a su simplicidad de uso.
- **CakePHP**, creado en 2005, es otro framework similar a CodeIgniter en cuanto a simplicidad y facilidad de uso. Tiene una amplia comunidad también detrás que le da soporte.
- **Zend**, creado en 2006, es otro framework bastante popular, aunque quizá con menor visibilidad que los anteriores hoy en día, a la altura de CakePHP.
- **Phalcon**, otro framework de reciente creación (2012), con una potente capacidad de procesamiento de páginas PHP, y la posibilidad de trabajar como microframework (más ligero, para ofrecer funcionalidades muy específicas) o como framework completo. De hecho, muchos frameworks más antiguos también han incorporado recientemente la posibilidad de ejecutarlos como microframeworks.
- ... etc.

Casi todos los frameworks PHP tienen una serie de características comunes, como son el uso del patrón MVC para desarrollar sus proyectos, la inyección de dependencias para gestionar recursos tales como

conexiones a bases de datos, o elementos compartidos por toda la aplicación, la posibilidad de desarrollar tanto webs completas como servicios REST accesibles desde diversos clientes, etc.

1.2. ¿Cuál elegir?

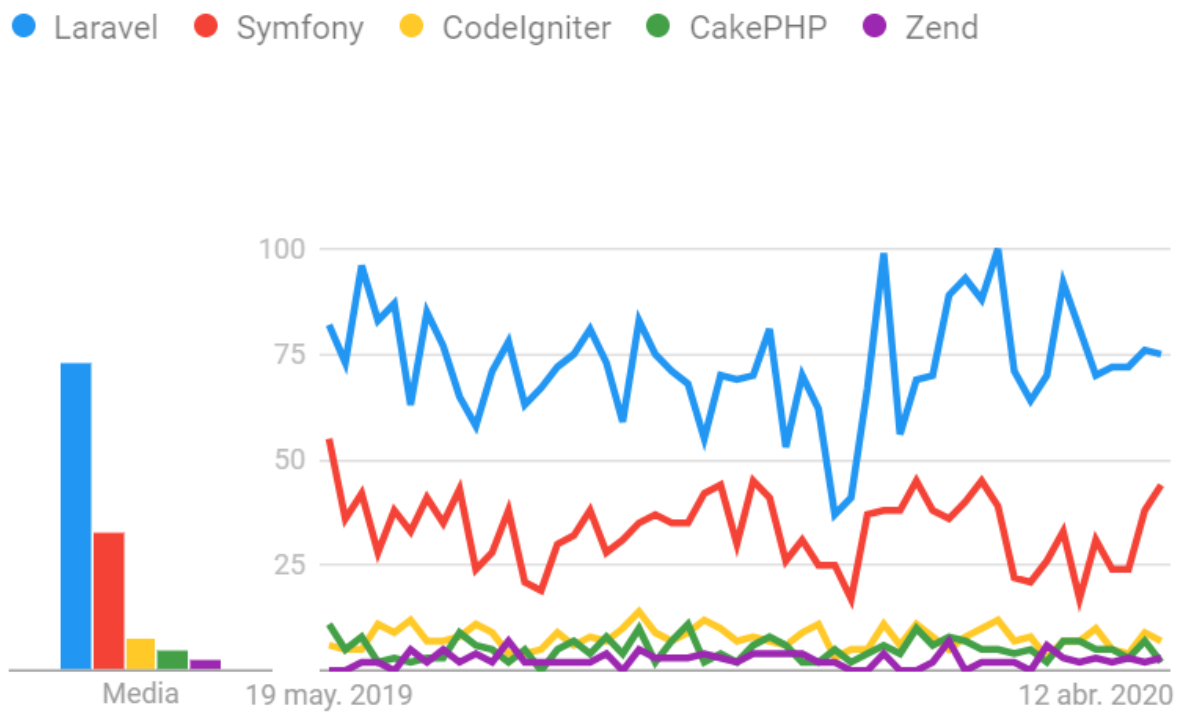
A la hora de decantarnos por uno u otro framework, no nos deberíamos dejar engañar por la popularidad del mismo, en términos de cuota de mercado. En ese terreno, Symfony y Laravel probablemente sean los más beneficiados, pero la curva de aprendizaje en ellos puede que sea más pronunciada que en otros a priori más sencillos, como CodeIgniter o CakePHP.

Cada framework puede estar mejor orientado que otro para determinados tipos de proyectos o necesidades. Si queremos aprender algo rápido para lanzar la aplicación cuanto antes mejor, quizá Symfony no sea la mejor opción. Si, por el contrario, preferimos empaparnos de un framework con una comunidad importante detrás que nos pueda dar soporte y nos garantice un tiempo de vida largo, entonces Symfony o Laravel pueden ser mejores candidatos.

¿Por qué Laravel?

Llegados a este punto... ¿qué características tiene Laravel que nos hayan hecho elegirlo para este curso frente a otros frameworks? Es un framework que ofrece bastantes ventajas.

- Es bastante popular hoy en día, quizá el que más, si atendemos a diferentes webs estadísticas. Por ejemplo, si comparamos las búsquedas en *Google Trends* de los principales frameworks PHP, observamos que Laravel es el más destacado:



- También es uno de los frameworks PHP que más demanda laboral tiene. Si hacemos una búsqueda en *InfoJobs* de estos cinco frameworks comparados antes a nivel nacional obtendremos, a fecha de Mayo de 2020, estos datos aproximados:

Framework	Ofertas encontradas
Laravel	75
Symfony	70
CodeIgniter	10
Zend	7
CakePHP	4

- Tiene una gran comunidad detrás, lo que permite encontrar fácilmente ayuda para problemas que tengamos
- Tiene una buena documentación, tanto por terceras partes como a través de su propia página oficial
- Dispone de algunas librerías adicionales que permiten añadir funcionalidad muy interesante, como el motor de plantillas Blade, o el ORM Eloquent, que veremos más adelante, así como librerías de terceras partes que podemos incorporar a nuestros proyectos.

En realidad, una vez se conoce uno de estos frameworks, es más sencillo asimilar el resto, llegado el momento. Así que Laravel puede ser un buen punto de partida. En concreto, durante el curso utilizaremos la versión 7 del framework, que se apoya en PHP 7 para funcionar, como veremos a continuación.

1.3. Recursos previos

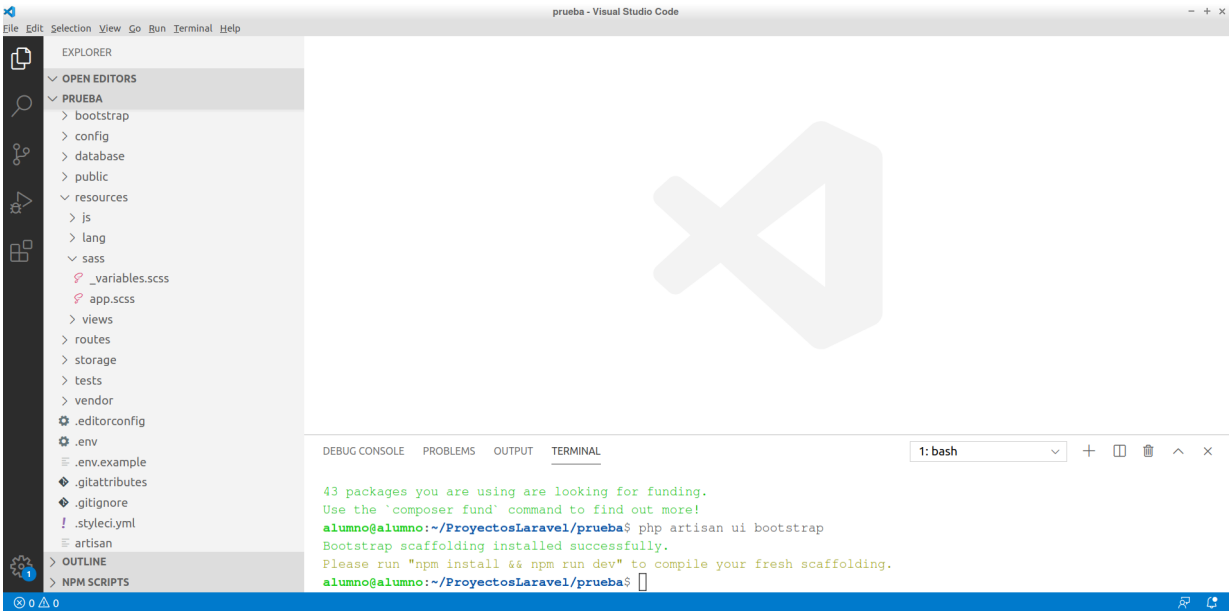
A la hora de trabajar con Laravel (versión 7), necesitamos tener previamente instalados en nuestro sistema una serie de recursos software, como son:

1. Un IDE (entorno de desarrollo) con el que editar el código de nuestros proyectos. Emplearemos Visual Studio Code en estos apuntes, aunque existen otras alternativas similares, como PHPStorm, Sublime Text, etc.
2. Un servidor web que soporte PHP 7.2.5 o posterior. En nuestro caso, utilizaremos Apache.
3. Un servidor de bases de datos en el que almacenar la información de nuestras aplicaciones. Emplearemos un servidor MariaDB/MySQL.
4. PHP (versión 7.2.5 o posterior).
5. El propio framework Laravel. Se necesitará instalar la herramienta `composer` para, después, instalar Laravel.
6. Además, necesitaremos el gestor de paquetes `npm` para instalar dependencias del lado del cliente en proyectos Laravel. Este gestor se instala con el framework Node.js.

Para cumplir con los requisitos 2, 3, y 4 de la lista anterior utilizaremos un sistema XAMPP. En el documento de *Instalación del software necesario* se tienen los pasos a seguir para instalar cada uno de estos elementos. Recuerda, no obstante, que dispones de una máquina virtual con todo el software necesario instalado, con lo que puedes omitir este paso si no quieres perder tiempo con él.

1.4. Algunas nociones básicas sobre Visual Studio Code

En el caso de que no hayáis trabajado todavía con Visual Studio Code, o no lo dominéis lo suficiente, no os preocupéis, es un IDE realmente sencillo de comprender y manejar. Como otros IDEs similares, se basa en carpetas. Podemos hacer clic derecho en una carpeta y elegir abrirla con Visual Studio Code, o en el caso de no tener esa opción, podemos abrir Visual Studio Code y arrastrarle la carpeta que queremos abrir. Automáticamente aparecerán los contenidos de esa carpeta en la parte izquierda del editor.

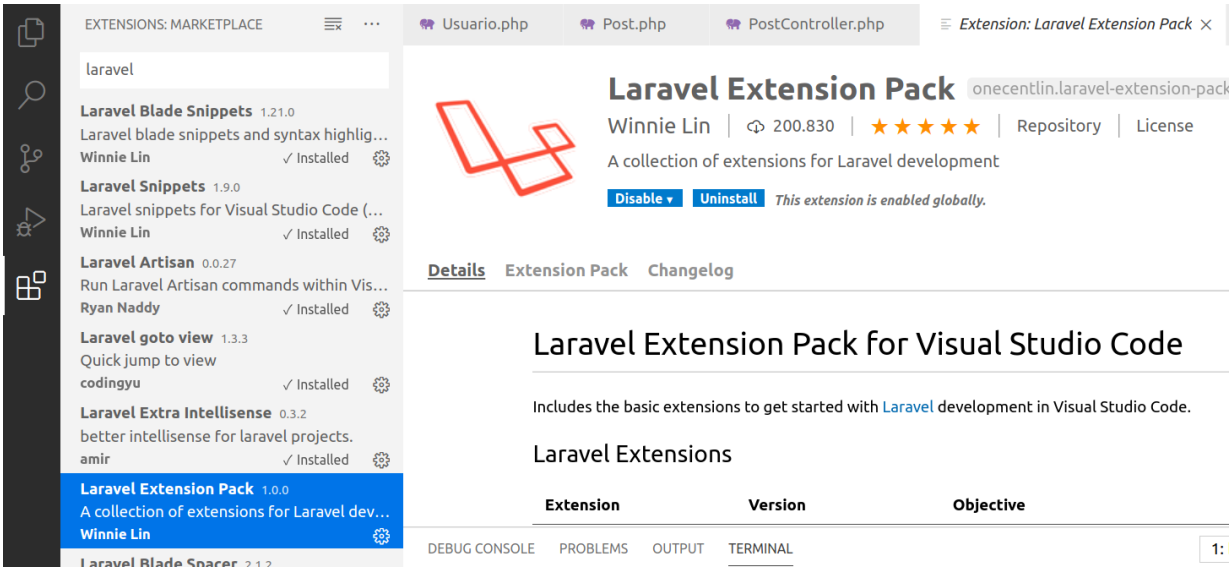


Además, podemos mostrar un **terminal integrado** en la parte inferior del editor, como el que se ve en la imagen anterior, yendo al menú *Ver > Terminal*. Esto es muy útil, porque el terminal se encuentra ya ubicado en la carpeta raíz del proyecto, y nos servirá para ejecutar algunos comandos para instalar dependencias o añadir componentes al proyecto, como veremos después.

1.4.1. Extensiones útiles

Cuando utilizamos Visual Studio Code para desarrollar aplicaciones en una determinada tecnología (Laravel, Node, etc), puede ser conveniente instalar algún *plugin* o extensión que nos facilite el desarrollo de dichas aplicaciones. Mediante estas extensiones podemos, por ejemplo, resaltar la sintaxis de los archivos que editemos, ayudarnos a autocompletar código, etc.

En el caso de Laravel, podemos ir a la seccion de *Extensiones* del menú izquierdo de Visual Studio Code, y buscar la extensión *Laravel Extension Pack*. Después, pulsamos en el botón de *Instalar* para que se instale dicha extensión, que a su vez contiene un paquete de extensiones útiles para desarrollar aplicaciones Laravel.



En concreto, podremos resaltar la sintaxis de las vistas HTML que hagamos con el motor de plantillas Blade, autocompletar ciertas partes del código, auto-corregir ciertos errores de código, etc.

2. Primeros pasos con Laravel

Ahora que ya tenemos todo el sistema preparado para desarrollar proyectos Laravel, veamos los primeros pasos que debemos dar para crear estos proyectos.

Para tener localizados todos los proyectos, comenzaremos por crear una carpeta llamada `ProyectosLaravel` en nuestra carpeta personal (en la carpeta `/home/alumno`, si usamos la máquina virtual). Los proyectos que hagamos a partir de ahora los ubicaremos dentro de esta carpeta, cada uno en su propia subcarpeta.

2.1. Crear proyectos Laravel

Para crear proyectos Laravel, emplearemos el comando `laravel` que ya deberíamos tener disponible, si hemos seguido los pasos dados en el documento de *Instalación del software necesario*, o si estamos utilizando la máquina virtual proporcionada, donde ya viene el comando instalado. Nos deberemos ubicar en la carpeta donde queramos crear el proyecto (la carpeta "ProyectosLaravel" que hemos comentado anteriormente), y escribir este comando:

```
laravel new nombre_proyecto
```

Por ejemplo, para las pruebas que iremos construyendo poco a poco en las siguientes sesiones, vamos a crear una web de libros, por lo que, en nuestra carpeta `ProyectosLaravel`, comenzaremos escribiendo este comando:

```
laravel new biblioteca
```

Esto creará un proyecto `biblioteca` en una subcarpeta con el mismo nombre. Alternativamente, también se puede emplear la herramienta `composer` para crear el proyecto, usando la siguiente sintaxis (también desde la carpeta donde queramos ubicar el proyecto):

```
composer create-project --prefer-dist laravel/laravel biblioteca
```

En cualquiera de los dos casos, se creará una carpeta "biblioteca" con el contenido inicial del proyecto dentro, empleando la última versión de Laravel que tengamos instalada.

2.1.1. Crear proyectos Laravel usando versiones anteriores

En el caso de que necesitemos crear un proyecto Laravel que no utilice la última versión, sino alguna anterior, necesitamos utilizar la herramienta `composer` para especificar el número de versión de Laravel que queremos utilizar. Por ejemplo, este comando crea un proyecto llamado "prueba" utilizando Laravel 5.8:

```
composer create-project --prefer-dist laravel/laravel prueba 5.8
```

2.2. El comando *artisan*

Cuando se crea un proyecto Laravel, se instala una herramienta llamada `artisan` en la raíz del proyecto. Es una interfaz de línea de comandos (CLI, *Command Line Interface*), que proporciona una serie de opciones adicionales que nos vendrán bien en nuestra gestión de proyectos Laravel para, por ejemplo, crear controladores, migrar datos a una base de datos, etc.

Para comprobar que está instalada y las opciones que ofrece, podemos escribir el siguiente comando en un terminal desde la carpeta del proyecto que hayamos creado (recuerda que puedes utilizar el terminal integrado de Visual Studio Code, con el proyecto abierto, para ejecutar el comando directamente desde la raíz del proyecto):

```
php artisan list
```

Y obtendremos un listado de las opciones que ofrece *artisan*. Algunas, como decimos, las utilizaremos más adelante. Para empezar, podemos ejecutar este comando para comprobar la versión de Laravel del proyecto en el que estamos:

```
php artisan --version
```

2.3. Estructura de un proyecto Laravel

Cuando creamos un proyecto Laravel, se crea una estructura de carpetas y archivos predefinida. Explicamos ahora brevemente en qué consisten las principales carpetas y archivos que se generan:

- `app` : contiene el código fuente de la aplicación. Gran parte de las clases que definamos estarán en esta carpeta. Inicialmente, se incluyen algunas subcarpetas dentro:
  - `Console` : para definir nuestros propios comandos
  - `Exceptions` : para definir nuestras propias excepciones
  - `Http` : contiene los controladores y el *middleware*
  - `Providers` : contiene los proveedores de servicios de la aplicación, más los que podamos definir nosotros.
  - Además, aquí se incluyen, o se pueden incluir, carpetas adicionales para nuestra aplicación, como la carpeta `Events` para definir los eventos que ocurran, o distintas carpetas para almacenar el modelo de datos o clases de nuestra aplicación.
- `bootstrap` : contiene el archivo `app.php` , que es el que pone en marcha la aplicación. Además, contiene la carpeta `cache` , donde se almacenan los archivos ya cargados por Laravel para acelerar su acceso en futuras peticiones.
- `config` : contiene los archivos de configuración de la aplicación, donde se tienen variables de entorno, o si nuestra aplicación está en desarrollo o producción, o los parámetros de conexión a la base de datos, entre otras cosas. Sin embargo, los cambios de configuración es preferible hacerlos en el archivo `.env` , ubicado en la raíz del proyecto Laravel, de modo que en este último archivo almacenaremos los datos privados (usuario y password de la base de datos, por ejemplo), y en los correspondientes archivos de `config` accederemos a estas variables de entorno definidas en `env` . Por ejemplo, podemos definir las propiedades del archivo `.env` de este modo para el proyecto *biblioteca* que hemos creado anteriormente:

```
APP_NAME=Biblioteca
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=biblioteca
DB_USERNAME=root
DB_PASSWORD=
...
```

> **NOTA:** Además, el archivo `.env` está configurado (o debe configurarse, de lo contrario) para ser ignorado por Git, de modo que no se suba a repositorios, y se evite un acceso a datos confidenciales

> que pongan en riesgo el acceso al sistema.

- `database`: almacena los elementos de gestión de la base de datos, tales como generadores de objetos, migraciones, etc.
- `public`: contenido visible de la web. Contiene el archivo `index.php`, punto de entrada de todas las peticiones a la web, además de carpetas donde ubicar el contenido estático del cliente (imágenes, hojas de estilo CSS, archivos JavaScript...).
- `resources`: contiene, por un lado, las vistas de nuestra aplicación. Por otro lado, también contiene archivos no compilados de CSS (archivos *sass*) y JavaScript (archivos sin minimizar). Además, también almacena los archivos de traducción, en el caso de que queramos hacer sitios multi idioma.
- `routes`: almacena las rutas de la aplicación, tanto para acceder a contenido web normal (`web.php`), como para servicios web (`api.php`), como para comandos y otras opciones.
- `storage`: contiene las vistas compiladas, y otros archivos generados por Laravel, como los logs o las sesiones.
- `test`: para almacenar los tests o pruebas sobre los componentes de nuestra aplicación
- `vendor`: donde se almacenan las dependencias o librerías adicionales que se requieren en nuestro proyecto Laravel. Nuevamente, esta carpeta debería ser ignorada por Git, y regenerada cada vez que se clone el repositorio remoto, para evitar almacenar demasiada información innecesaria.

Aunque algunos de los conceptos vistos aquí pueden no estar claros aún (como el concepto de *middleware*, o los proveedores de servicios), los iremos viendo poco a poco durante el curso.

2.4. Arquitectura de un proyecto Laravel

Una vez vista la estructura de carpetas y archivos que se genera cuando creamos un proyecto Laravel, es importante también tener unas nociones básicas de cómo se interconectan los elementos internamente, y qué hace que un proyecto Laravel se pueda poner en marcha.

2.4.1. Los proveedores de servicios (*service providers*)

Los proveedores de servicios son los principales responsables del arranque o puesta en marcha de todo proyecto Laravel, lo que se conoce como *bootstrapping*. Se encargan de registrar los componentes del proyecto, dependencias externas, clases y métodos definidos por nosotros, para hacerlos accesibles al resto de la aplicación.

Si abrimos el archivo `config/app.php` de nuestro proyecto Laravel, veremos entre otras cosas una sección denominada `providers`, donde se define un array con todos los proveedores de servicios que se ponen en marcha al arrancar la aplicación. Entre otras cosas, hay proveedores de servicios para acceso a la base de datos (*DatabaseServiceProvider*), autenticación de usuarios (*AuthServiceProvider*), etc.

2.4.2. Las clases del núcleo de Laravel

Para poder desarrollar los componentes de las aplicaciones Laravel, es necesario contar con una infraestructura previa de clases que nos faciliten esta tarea. Así, a lo largo de las siguientes sesiones haremos uso de algunas clases proporcionadas por Laravel que vienen preinstaladas con el framework,

tales como `Model`, `Route`, etc, y que nos permiten o bien heredar de ellas para crear otras subclases (como es el caso de los modelos de datos) o bien utilizar algunos métodos de utilidad que estas clases proporcionan (como es el caso de la clase `Route`, por ejemplo).

Conviene tener presente que todas estas clases pertenecientes al núcleo de Laravel parten de un espacio de nombres común llamado `Illuminate`, por lo que, en los archivos fuente donde las utilicemos, será frecuente encontrar instrucciones `use` que comiencen por dicho espacio de nombres. Por ejemplo:

```
use Illuminate\Database\Eloquent\Model;
```

2.4.3. Otros elementos

Además de los dos pilares anteriores sobre los que se sustenta fundamentalmente el desarrollo de proyectos en Laravel, podemos hablar de otros elementos que nos pueden resultar de utilidad en el desarrollo, como son los *facades* y los *contracts*.

Las *facades* proporcionan una interfaz estática a los elementos de la aplicación, de forma que facilitan el acceso a ciertos métodos o utilidades. Por ejemplo, la *facade* `Cache` permite acceder de forma sencilla con su método `get` a ciertas propiedades cacheadas previamente.

```
return Cache::get('key');
```

Los *contracts* son un conjunto de interfaces que proporcionan el núcleo de servicios ofrecidos por Laravel. Por ejemplo, métodos para enviar e-mails, o encolar tareas en una cola de prioridad, etc.

2.4.4. El patrón MVC

Como hemos comentado anteriormente, tanto Laravel como otros muchos frameworks de desarrollo web se apoyan en el patrón MVC. MVC son las siglas de *Modelo-Vista-Controlador* (o en inglés, *Model-View-Controller*), y es el patrón por excelencia ahora mismo en el mundo de las aplicaciones web, e incluso muchas aplicaciones de escritorio.

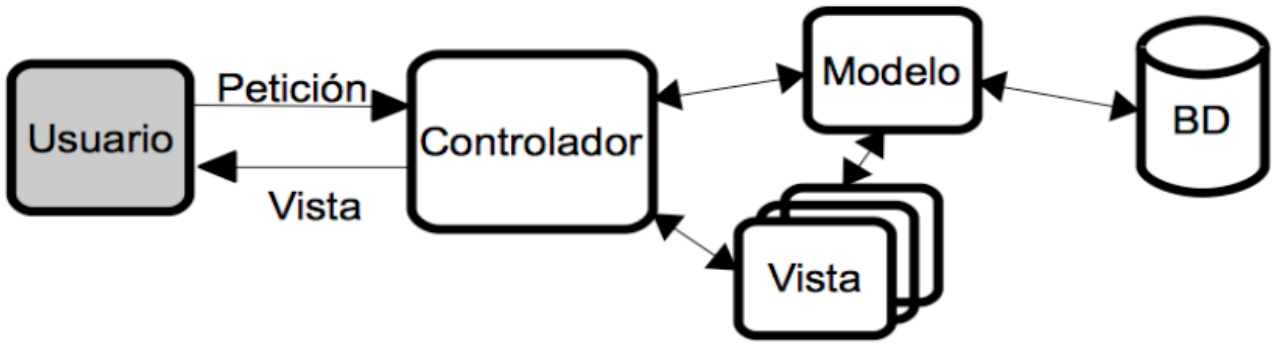
Como su nombre indica, este patrón se basa en dividir el diseño de una aplicación web en tres componentes fundamentales:

- El **modelo**, que podríamos resumir como el conjunto de todos los datos o información que maneja la aplicación. Típicamente serán variables u objetos extraídos de una base de datos o cualquier otro sistema de almacenamiento, por lo que el código del modelo normalmente estará formado clases o elementos donde almacenar los datos que extraigamos o vayamos a almacenar en esa base de datos. Generalmente, el modelo no tendrá conocimiento del resto de componentes del sistema.
- La **vista**, que es el intermediario entre la aplicación y el usuario, es decir, lo que el usuario ve en pantalla de la aplicación. Por lo tanto, la vista la compondrán las diferentes páginas, formularios, etc, que la aplicación mostrará al usuario para interactuar con él.

- El **controlador** (o controladores), que son los fragmentos de código encargados de coordinar el funcionamiento general de la aplicación. Ante peticiones de los usuarios, las recogen, las identifican, y utilizan el modelo para actualizar o recuperar datos, y a su vez, deciden qué vista mostrarle al usuario a continuación de la acción que acaba de realizar.

Es un patrón de diseño muy conciso y bien estructurado, lo que le ha valido la fama que tiene hoy en día. Entre sus muchas ventajas, permite aislar el código de los tres elementos involucrados (vista, modelo y controlador), de forma que el trabajo es mucho más modular y divisible, pudiendo encargarse de las vistas, por ejemplo, un diseñador web que no tenga mucha idea de programación en el servidor, y del controlador un programador PHP que no tenga muchas nociones de HTML.

En forma de esquema, podríamos representarlo así:



2.5. Prueba de proyectos Laravel

Para poder probar un proyecto Laravel, hay que realizar una serie de pasos previos, tales como asociarlo a un *virtual host* (de Apache, en nuestro caso), y habilitar ciertos permisos en ciertas carpetas. A continuación indicamos los pasos a seguir, y conviene tener presente que:

- El paso 1 deberemos hacerlo **sólo una vez**, cuando demos de alta nuestro primer *virtual host* con Apache.
- El resto de pasos deberemos hacerlos **una vez por proyecto** para configurar dicho proyecto en Apache y establecer los permisos de lectura y escritura adecuados.

1. Habilitar los *virtual hosts* en Apache

Los *virtual hosts* son un mecanismo que ofrecen los servidores web, como Apache, para poder asociar carpetas arbitrarias del sistema, externas a la estructura de Apache, al propio servidor, de forma que, accediendo a una URL o nombre de dominio determinado, le indicamos a Apache que cargue los contenidos de esa carpeta.

En primer lugar debemos habilitar los *virtual hosts* en Apache, editando el archivo `/opt/lampp/etc/httpd.conf` y descomentando la línea que hace referencia al lugar donde se definen dichos *virtual hosts*. Debe quedar así:

```
# Virtual hosts
Include etc/extra/httpd-vhosts.conf
```

2. Añadir el nuevo nombre de dominio

Después, debemos editar el archivo `/etc/hosts` y asignar un nombre de dominio (local) a nuestro proyecto. Por ejemplo, para el proyecto *biblioteca* que hemos creado antes, podríamos definir algo como esto (al final, o entre los otros registros de nombres existentes en dicho archivo):

```
127.0.0.5   biblioteca
```

Lo que hemos hecho ha sido asignar la IP local 127.0.0.5 (puede ser la IP local que nosotros queramos) al nombre "biblioteca". De este modo, cuando carguemos localmente el proyecto podremos acceder a él mediante la URL *http://biblioteca* o bien con *http://127.0.0.5*.

3. Definir la configuración del nuevo *virtual host*

A continuación, debemos editar el archivo al que hacía referencia la línea que hemos descomentado antes, `/opt/lampp/etc/extra/httpd-vhosts.conf` y añadir una nueva configuración para nuestro nuevo *virtual host*. Por ejemplo:

```
<VirtualHost 127.0.0.5:80>
  DocumentRoot "/home/alumno/ProyectosLaravel/biblioteca/public"
  DirectoryIndex index.php

  <Directory "/home/alumno/ProyectosLaravel/biblioteca/public">
    Options All
    AllowOverride All
    Require all granted
  </Directory>
</VirtualHost>
```

4. Establecer los permisos en las carpetas del proyecto

Para poder probar nuestro proyecto Laravel, además de configurar el servidor web (por ejemplo, Apache) para que apunte a la carpeta `public` del proyecto, y asignarle si es el caso un *virtual host*, es necesario habilitar permisos de acceso y escritura a ciertas carpetas del proyecto:

- Carpeta `storage` junto con sus subcarpetas y contenidos. En esta carpeta se compilarán las vistas, se generarán los archivos de log, etc, por lo que conviene que esta carpeta tenga permisos de escritura.
- Subcarpeta `bootstrap/cache`, donde se almacenará la caché de los archivos ya cargados.

Para habilitar los permisos en estas carpetas, podemos ejecutar estos comandos desde la raíz del proyecto (la opción `-R` aplica los permisos de forma recursiva):

```
sudo chmod -R 777 bootstrap/cache
sudo chmod -R 777 storage
sudo chmod -R 777 storage/logs
```

> **NOTA** El tercer comando no sería necesario en principio, ya que la subcarpeta `logs` está dentro de la carpeta `storage`, y se aplican los cambios de forma recursiva. Sin embargo, es posible que en algunas situaciones esta subcarpeta se cree a posteriori y no tenga los permisos adecuados. Comprobaremos al cargar la aplicación desde el navegador si existe algún error al inicio. En este caso, el propio error indicará que no puede generar el *log*, y deberemos escribir ese comando.

Tras estos pasos, ya podremos acceder a la página de inicio de nuestro proyecto, escribiendo la URL *http://biblioteca*, en este caso.

Más adelante aprenderemos a modificar esta página de inicio, obviamente, y a ir añadiendo otras.

2.5.1. Otra forma alternativa de probar los proyectos

A través de la herramienta `artisan`, tenemos una alternativa algo más rápida para probar nuestros proyectos Laravel de forma local. Si optamos por esta opción, podemos omitir los pasos 1, 2 y 3 del caso anterior, ya que no vamos a utilizar Apache como servidor. Sólo necesitamos realizar el paso 4 para dar permisos de escritura a las carpetas necesarias, la primera vez que pongamos en marcha la aplicación.

Después, desde la raíz del proyecto, ejecutamos este comando:

```
php artisan serve
```

Esto pondrá en marcha un pequeño servidor para probar el proyecto, y podremos acceder a él desde la URL *http://localhost:8000*, aunque, de todas formas, al ejecutar el comando nos informará de la URL con la que acceder.

En este punto, puedes realizar el Ejercicio 1 de los propuestos al final de la sesión.

3. Primeros pasos con las rutas en Laravel

Las rutas (*routes*) son un mecanismo que permite a Laravel establecer qué respuesta enviar a una petición que intenta acceder a una determinada URL. Estas rutas se especifican en diferentes archivos dentro de la carpeta `routes` de nuestro proyecto Laravel.

En el archivo `routes/web.php` definiremos las rutas web, que son las más habituales, para recuperar contenidos típicamente en formato HTML. Inicialmente ya existe una ruta predefinida hacia la raíz del proyecto, que carga la página de bienvenida al mismo

```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('/', function() {
    return view('welcome');
});
```

Para definir una ruta en Laravel, se hace una llamada al método estático `Route::get`, de la clase `Route`. Como primer parámetro, especificaremos la URL de la ruta (la ruta raíz, en el ejemplo anterior), y como segundo parámetro, la función que se va a ejecutar cuando algún cliente haga una petición a esa ruta.

Podemos añadir, por ejemplo, una segunda ruta mediante la cual, si accedemos a la URL *http://biblioteca/fecha* nos muestre la fecha y hora actuales. Añadimos para ello la siguiente ruta bajo la anterior que ya estaba definida:

```php
Route::get('fecha', function() {
    return date("d/m/y h:i:s");
});
```

Si accedemos ahora a *http://biblioteca/fecha*, deberíamos ver la fecha y hora actuales, en texto plano.

Además de utilizar el método `get`, desde la clase `Route` también podemos acceder a otros métodos estáticos útiles, como `Route::post` (útil para recoger datos de formularios, por ejemplo), o también `Route::put`, `Route::delete`... Los veremos con más detalle en sesiones posteriores.

En este punto, puedes realizar el Ejercicio 2 de los propuestos al final de la sesión.

4. Importando un proyecto Laravel

Para finalizar esta primera sesión, vamos a indicar unas instrucciones necesarias en el caso de que queramos importar un proyecto Laravel a un nuevo ordenador, descargándolo de, por ejemplo, un repositorio GitHub. Dado que ciertas carpetas y archivos no se suben a dicho repositorio (o no deberían subirse), es conveniente saber cómo regenerar estos elementos en la nueva ubicación del proyecto.

1. El archivo de configuración de variables de entorno *.env*

Uno de estos archivos no incluidos es el archivo `.env`, que contiene información sensible, como la contraseña de acceso a la base de datos. Sin embargo, lo que sí se incluye es una copia inicial del mismo, en el archivo `.env.backup` o `.env.example`, dependiendo de la versión de Laravel que utilicemos. Basta con hacer una copia de dicho archivo en la carpeta raíz del proyecto...

```
cp .env.example .env
```

... y luego editar dicho archivo para establecer la configuración oportuna en el lugar donde hayamos importado el proyecto: parámetros de conexión a la base de datos, y otras variables de entorno que iremos viendo en estas sesiones.

2. La clave del proyecto

Laravel necesita de una clave en la variable de entorno APP_KEY del archivo `.env` anterior, que por defecto está vacía. Dicha clave es un código aleatorio de 32 caracteres, que Laravel emplea para encriptar cookies. Podemos generar una clave con el comando `php artisan` (desde la raíz del proyecto):

```
php artisan key:generate
```

y ya la tendremos lista en nuestro archivo `.env`.

3. Dependencias PHP

Otro de los elementos del proyecto que no se comparte en repositorios es la carpeta `vendor`, donde vienen descargadas las dependencias PHP de nuestro proyecto. Por defecto, al generar un nuevo proyecto Laravel, se presuponen algunas de ellas, incluidas en el archivo `composer.json` de la raíz del proyecto. Para volverlas a instalar en donde hayamos clonado el proyecto, ejecutamos este comando desde la raíz del proyecto (suponiendo que ya tengamos instalado el comando `composer` de pasos anteriores):

```
composer install
```

4. Dependencias JavaScript

Del mismo modo, existen algunas dependencias para la parte de cliente (como por ejemplo *Bootstrap*, o *jQuery*), definidas en el archivo `package.json` de la raíz del proyecto, y que se encuentran preinstaladas en la carpeta `node_modules`. Esta carpeta, sin embargo, tampoco se comparte en el repositorio, así que para volverla a generar en el proyecto clonado, y suponiendo que también tendremos instalada la herramienta `npm` de pasos anteriores, ejecutamos el comando siguiente desde la raíz del proyecto:

```
npm install
```

5. Ejercicios propuestos

Ejercicio 1

Crea un proyecto Laravel llamado **blog**. Configura un *virtual host* en Apache con el mismo nombre, y asócialo a una IP local (la que tú quieras, por ejemplo, 127.0.0.6). Prueba a acceder a la página de inicio de este nuevo proyecto (con *http://blog* o con *http://127.0.0.6*) y haz una captura de pantalla de dicha página de inicio, donde se vea también la URL introducida en la barra del navegador. Guarda la captura como `captura1.png`, de forma que se pueda ver la URL de acceso a la web también. Esta captura será lo que deberás entregar como prueba de este ejercicio.

Ejercicio 2

Sobre el proyecto anterior, edita el fichero `routes/web.php` y añade una nueva ruta a la URL `posts`. Al acceder a esta ruta (*http://blog/posts*), deberemos ver un mensaje con el texto "Listado de posts".

Accede a la URL para probarla, y cuando funcione, comprime el proyecto *blog* en un archivo ZIP llamado `blog_01.zip`. Excluye de este archivo ZIP las carpetas `vendor` y `node_modules` (si existe), para que no ocupe tanto. Esto será lo que deberás entregar como prueba de este ejercicio.

> **NOTA IMPORTANTE**: cuando comprimas el proyecto Laravel para su entrega, procura comprimir la carpeta entera desde fuera, para así incluir también los archivos ocultos, como `.env` o `.env.example`. De lo contrario, no será posible poner en marcha el proyecto en el lugar donde se utilice. Así que, simplemente, elimina las carpetas pesadas (`vendor` y `node_modules`), sal a la carpeta padre y comprímela.