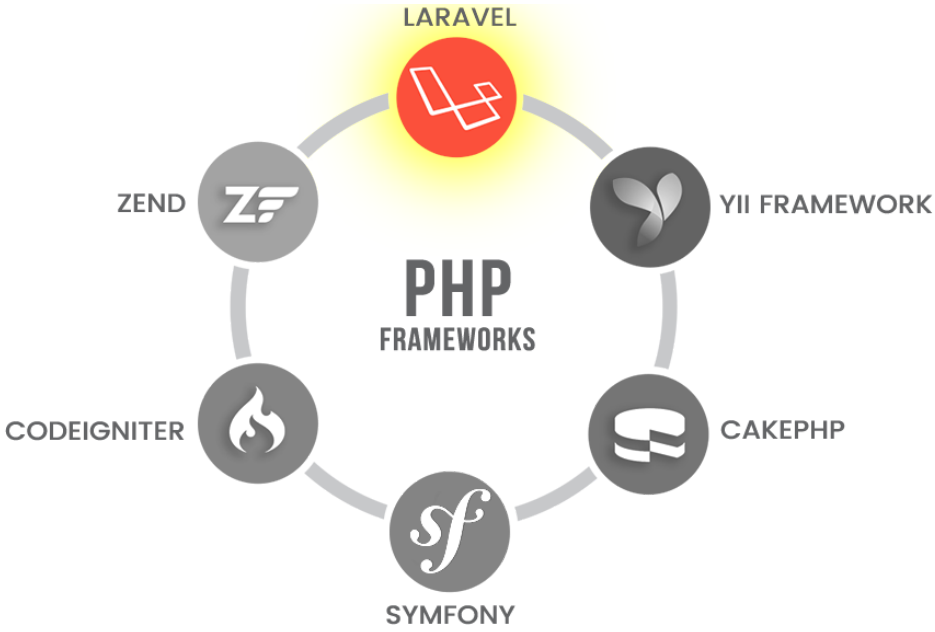
unidad didáctica 7

Laravel - Introducción



material: extraído de https://nachoiborraies.github.io/laravel/

1. **Frameworks PHP**
   1. 1. Ejemplos de frameworks PHP
   1. 2. ¿Cuál elegir?
   1. 3. El patrón MVC
2. **software necesario**
   2. 1. Visual Studio Code
   2. 2. Apache, PHP y MariaDB/MySQL con XAMPP
      2. 2. 1. Instalación
         2. 2. 1. 1. El manager de XAMPP
   2. 3. Laravel
      2. 3. 1. Instalando *composer*
      2. 3. 2. Instalando Laravel
      2. 3. 3. Actualizando Laravel
   2. 4. Node.js
      2. 4. 1. Instalación en Linux
      2. 4. 2. Instalación en Mac OSX
      2. 4. 3. Instalación en Windows
      2. 4. 4. Utilizando NVM
      2. 4. 5. Prueba de Node
   2. 5. Recursos adicionales
3. **Primeros pasos con Laravel**
   3. 1. Crear proyectos Laravel
      3. 1. 1. Usando el comando *laravel*
      3. 1. 2. Usando el comando *composer*
      3. 1. 3. Crear proyectos Laravel usando versiones anteriores
      3. 1. 4. El comando *artisan*
      3. 1. 5. Laravel y Visual Studio Code
   3. 2. Estructura de un proyecto Laravel
      3. 2. 1. Configuración general del proyecto
   3. 3. Arquitectura de un proyecto Laravel
      3. 3. 1. Los proveedores de servicios (*service providers*)
      3. 3. 2. Las clases del núcleo de Laravel
      3. 3. 3. Otros elementos
   3. 4. Prueba de proyectos Laravel
      3. 4. 1. Permisos en carpetas del proyecto
      3. 4. 2. Puesta en marcha con el comando *artisan*
      3. 4. 3. Puesta en marcha en carpeta predefinida de XAMPP
      3. 4. 4. Puesta en marcha como host virtual
   3. 5. Importando / Exportando un proyecto Laravel
      3. 5. 1. Exportar un proyecto
      3. 5. 2. Importar un proyecto existente
4. **referencias**

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. De hecho, podríamos considerar Symfony como un *metaframework*, es decir, un framework que, a su vez, sirve para desarrollar otros frameworks. Prueba de ello es que, por ejemplo, Laravel utiliza Symfony como base para ampliar esas funcionalidades.
- **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, aunque con menor popularidad.
- **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?

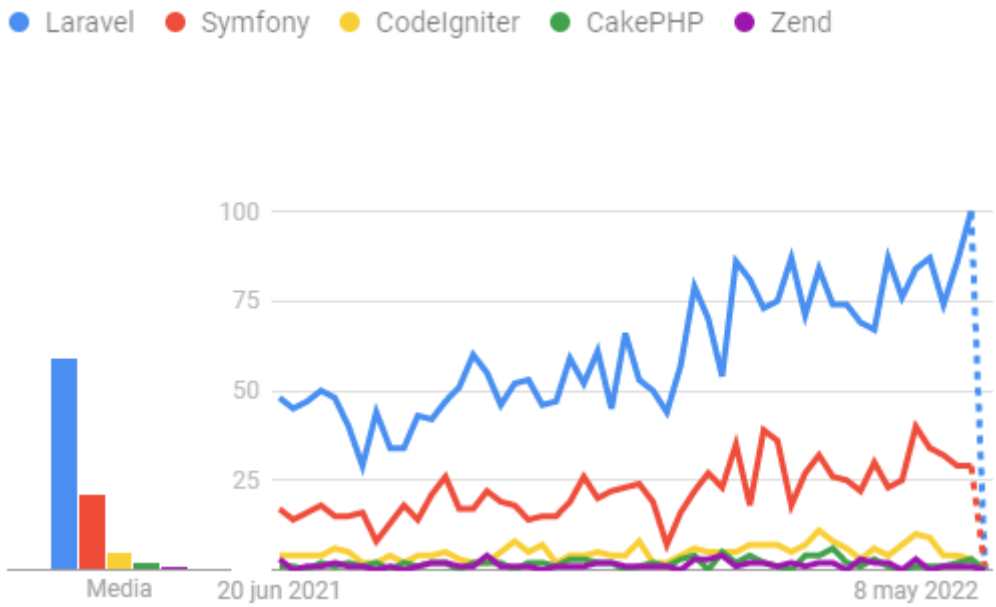
A la hora de decantarnos por uno u otro framework, no nos deberíamos dejar engañar por la popularidad del mismo, en términos de cuota de mercado. En ese terreno, Symfony y Laravel probablemente sean los más beneficiados, pero la curva de aprendizaje en ellos puede que sea más pronunciada que en otros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

Llegados a este punto... ¿qué características debemos mirar a la hora de decantarnos por uno u otro framework? Quizá algunas de las más importantes (y no necesariamente excluyentes entre sí) son:

Popularidad

La popularidad la podemos medir en base a diferentes webs estadísticas. Por ejemplo, si comparamos las búsquedas en *Google Trends* de los principales frameworks PHP, podemos determinar cuáles son los más buscados a nivel mundial:



Demanda laboral

Otro factor determinante para elegir un framework de desarrollo es la demanda laboral que tiene, las puertas que se nos pueden abrir al aprenderlo. Por ejemplo, si hacemos una búsqueda en el portal español de búsqueda de empleo *InfoJobs* de algunos frameworks PHP, a fecha de *Junio de 2022*, obtenemos estos datos aproximados:

Framework	Ofertas encontradas
Laravel	107
Symfony	76
CodeIgniter	15
Zend	4
CakePHP	3

Facilidad de aprendizaje

Es otro factor importante a tener en cuenta, especialmente cuando el tiempo de que disponemos para realizar el proyecto es escaso. En este sentido, frameworks como Symfony o Laravel suelen ser más "pesados de digerir", precisamente por su envergadura y la cantidad de opciones que ofrecen, mientras que otros más livianos como CodeIgniter o Zend son más fácilmente asimilables.

Soporte y documentación

Es importante analizar la documentación y la comunidad de desarrollo y soporte que hay detrás de cada framework, para saber si vamos a poder resolver dudas sobre su uso con facilidad. Por ejemplo, Laravel y Symfony, que son dos de los frameworks más difundidos, cuentan con una gran comunidad detrás (especialmente Laravel), y una documentación muy completa y actualizada.

- Documentación de Laravel
- Documentación de Symfony

Extensiones o *plug-ins*

También puede resultar un dato relevante el conocer la capacidad de ampliación de un framework, qué otras funcionalidades adicionales se le pueden incluir en caso necesario. Muchos de los frameworks PHP pueden hacer uso de la herramienta *composer* que veremos en otras secciones para instalar dependencias o módulos externos, y enriquecer así la funcionalidad de la aplicación.

En el caso de algunos frameworks, ya vienen con ciertas funcionalidades o extensiones incorporadas que suponen un valor añadido. Por ejemplo, Laravel incorpora un motor de plantillas llamado Blade para desarrollar fácilmente vistas HTML, así como un ORM llamado Eloquent para trabajar con bases de datos relacionales como si fueran objetos. Symfony también hace lo propio con el motor de plantillas Twig y el ORM Doctrine, respectivamente.

En realidad, una vez se conoce uno de estos frameworks, es más sencillo asimilar el resto, llegado el momento. Así que cualquiera de ellos, con unos motivos que se ajusten a nuestras necesidades, puede ser un buen punto de partida.

1.3. El patrón MVC

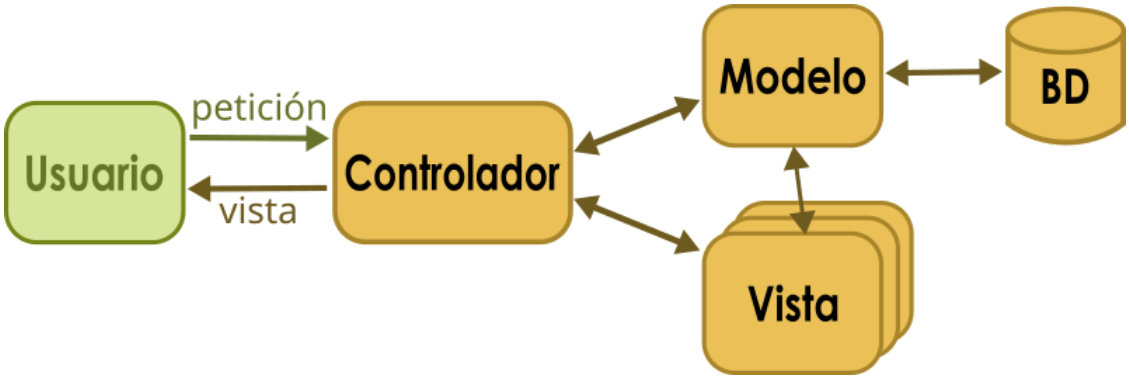
Como hemos comentado anteriormente, la gran mayoría de frameworks PHP se apoyan en el patrón MVC. MVC son las siglas de *Modelo-Vista-Controlador* (o en inglés, *Model-View-Controller*), y es el patrón por excelencia ahora mismo en el mundo de las aplicaciones web, e incluso muchas aplicaciones de escritorio.

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 por 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 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. software necesario

A la hora de trabajar con Laravel, 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, Atom, etc.
2. Un **servidor web** que soporte PHP. 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** actualizado a una versión compatible con la versión de Laravel que vayamos a utilizar. Por ejemplo, para Laravel versión 9 se necesita una versión de `PHP 8.0` o posterior.
5. El propio **framework** `Laravel`. Se necesitará instalar la herramienta **`composer`** para, después, instalar Laravel. Aunque también se pueden crear proyectos Laravel desde la propia herramienta `composer`, como veremos más adelante.
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`.
7. Otras herramientas adicionales que nos puedan venir bien, como por ejemplo alguna herramienta para probar el acceso a los servicios REST que desarrollemos.

Veremos a continuación los pasos necesarios para instalar todo el software que utilizaremos. Se darán las pautas de instalación en un sistema Linux basado en Debian, como por ejemplo Ubuntu, Lubuntu, Linux Mint, etc. También se facilitará un enlace a una máquina virtual donde poder instalar todo el software, así como otra máquina virtual con el software ya preinstalado y listo para utilizarse, usando las versiones que se indican a lo largo del documento.

Para **usuarios de otros sistemas**, como Windows o Mac OSX, la mayoría de opciones que veremos aquí son igualmente válidas (cambiando la ruta y/o el modo de instalación de algunas herramientas). En cualquier caso, se puede hacer uso de otras herramientas alternativas, como por ejemplo:

- Laragon para Windows, un sistema que integra los componentes de los puntos 2 a 6 vistos antes (servidor web, de base de datos, PHP, Node, etc).
- Laravel Homestead, un ecosistema basado en máquinas virtuales que también integra distintos componentes necesarios para desarrollar aplicaciones Laravel, aunque más complejo de instalar y configurar que Laragon o los pasos que seguiremos aquí en la máquina virtual.

En cualquier caso, no es el propósito de este curso aprender a instalar todo el software en todos los sistemas posibles, y es por ello que proporcionamos la máquina virtual indicada, para simplificar las opciones. Veremos algunas pinceladas de cómo instalar ciertas herramientas en varios sistemas, no obstante.

2.1. Visual Studio Code

Como IDE para desarrollar nuestras aplicaciones emplearemos **Visual Studio Code**, que es uno de los IDEs más versátiles que existen hoy en día para desarrollo web. Desde la web oficial de Visual Studio Code podemos descargarlo para la plataforma deseada.

> **versión requerida**: ninguna en particular, sirve con la última versión disponible.

Linux (Debian)

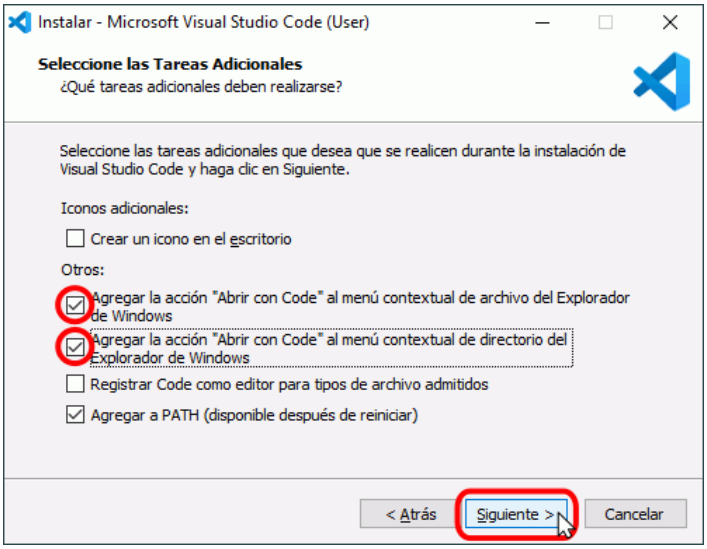
En el caso de nuestra máquina virtual o una distribución similar, descargaremos un archivo *.deb*. Una vez descargado, accedemos por terminal a la carpeta donde esté y ejecutamos este comando para instalarlo:

```
1 sudo dpkg -i nombre_del_archivo.deb
```

Se creará automáticamente un acceso directo en el menú de inicio, dentro de la sección de *Programación* en el caso de Lubuntu.

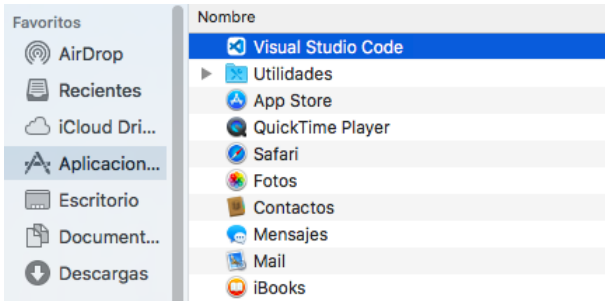
Windows

Para Windows descargamos el instalador y seguimos los pasos. No hay mucho que configurar; en todo caso, podemos dejar marcada la casilla para añadir el menú contextual "*Abrir con Code*" para poder abrir archivos y carpetas con VS Code desde el explorador de archivos directamente, con un clic derecho.



Mac OSX

Para **Mac OSX**, descargamos la aplicación y la podemos ejecutar directamente. También podemos moverla a la carpeta de *Aplicaciones* para tenerla localizada.



Extensiones para VS Code

Recomendable será instalar los siguientes plugins para Visual Studio Code.

Referentes a PHP:

- *PHP Intelephense*
- *PHP IntelliSense*
- *PHP Namespace Resolver*

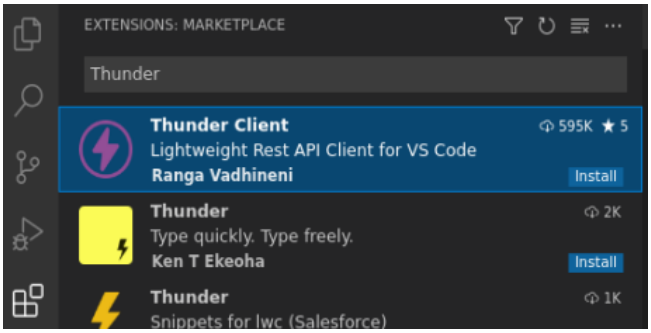
Referentes a Laravel:

- Laravel Blade Snippets
- *Laravel Snippets*
- *Laravel goto view*
- *Laravel Extra Intellisense*

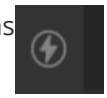
Referentes a API:

- *Thunder Client*

  Esta herramienta nos servirá para simular peticiones a servidores web, y recoger y analizar la respuesta. La emplearemos para probar los servicios *REST* que desarrollaremos en algunas sesiones. Se instala como una extensión de Visual Studio Code. La buscamos en el panel de extensiones y la instalamos:



  Nos aparecerá un icono en el panel izquierdo desde el que gestionaremos las conexiones y peticiones (aprenderemos a utilizarlo más adelante):



  Como alternativa, también podemos utilizar la herramienta **Postman**, o bien vía web o instalando la aplicación de escritorio desde su web oficial.

Referentes a CSS:

- *Tailwind CSS IntelliSense*

2.2. Apache, PHP y MariaDB/MySQL con XAMPP

Para poder tener un sistema con Apache, PHP y un gestor de bases de datos (como MariaDB/MySQL), y poderlo gestionar cómodamente, trabajaremos con un sistema AMPP, paquetes que integran en una sola instalación todas estas cosas. El ejemplo más conocido de estos sistemas es **XAMPP**, aunque existen otros como WAMPP, para Windows. Una de las ventajas que ofrecen es que, además de instalar Apache, PHP y MySQL y dejarlo todo integrado, nos proporciona un cliente web llamado **phpMyAdmin** para poder administrar las bases de datos desde Apache. Esto nos vendrá bien para crear o importar las bases de datos de los distintos ejercicios.

> **versión requerida**: depende sobre todo de la versión de Laravel con la que vayamos a trabajar. En el caso por ejemplo de Laravel 9, es necesario tener al menos una versión de PHP 8.0, y para ello debemos contar con una versión de XAMPP 8.x.

2.2.1. Instalación

Para instalar XAMPP, basta con descargarlo de su web oficial y seguir los pasos del asistente. Nos basta con tener instalado Apache, MySQL y PHP, así que podemos descartar otras opciones que nos ofrezca, si nos da a elegir.

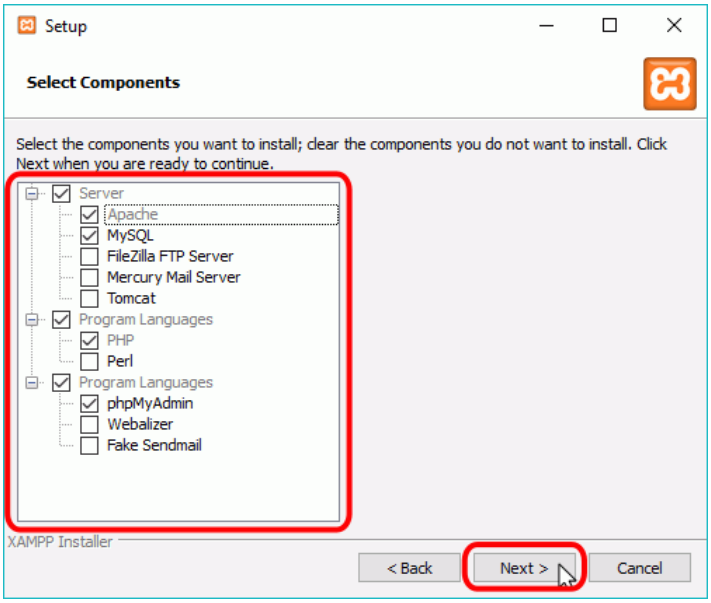
Linux

En el caso de Linux, debemos dar permisos de ejecución y ejecutar el archivo *.run* que descarguemos desde algún terminal, con permisos de administrador (*sudo*). Suponiendo que el archivo se llame *xampp-linux-x64-7.4.5-installer.run*, por ejemplo, los pasos son los siguientes (desde la carpeta donde lo hemos descargado):

```
sudo chmod +x xampp-linux-x64-7.4.5-installer.run
sudo ./xampp-linux-x64-7.4.5-installer.run
```

Windows y MacOSX

En el caso de **Windows** o **Mac OSX** simplemente hay que lanzar el instalador y seguir los pasos, eligiendo las opciones que nos interese instalar (al menos, Apache, MySQL y PHP), si nos dan a elegir. Así es como podemos dejarlo en el caso de Windows, por ejemplo:



2.2.1.1. El manager de XAMPP

XAMPP proporciona una herramienta *manager* o *panel de control* que nos permite gestionar en todo momento los servicios activos.

En el caso de **Linux** se encuentra en **/opt/lampp/manager-linux-x64.run**. Podemos acceder a la carpeta desde el terminal para ejecutarlo (con permisos de superusuario); o bien crear algún acceso directo en otra ubicación que nos resulte más cómoda. Por ejemplo, podemos crear un acceso directo en el escritorio con el editor *nano* o con el propio editor *Visual Studio Code* que hemos instalado previamente. Suponiendo la carpeta */home/alumno/Escritorio/* o */home/alumno/Desktop/*, como la que tenemos en la máquina virtual, podemos primero crear el archivo vacío:

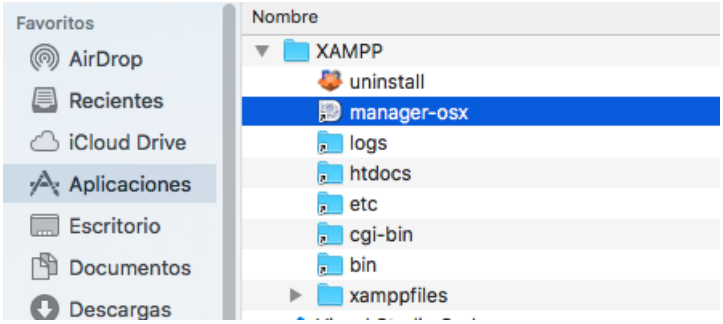
```
touch /home/alumno/Desktop/XAMPP.desktop
```

Editamos el contenido del archivo y añadimos las líneas de configuración para el acceso directo:

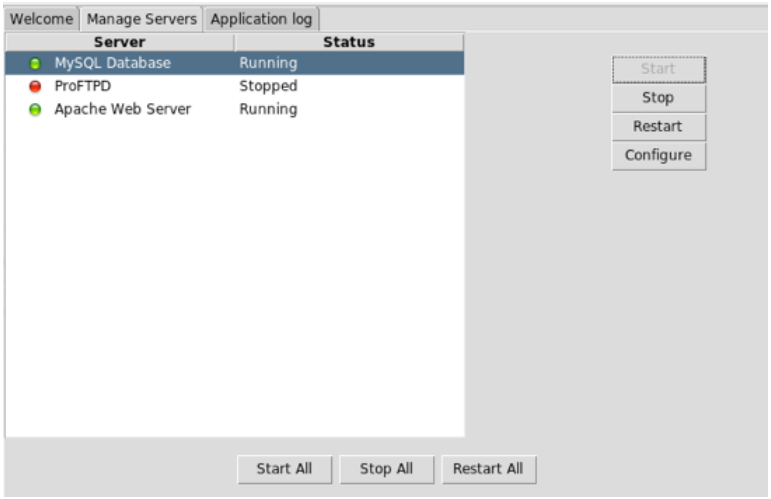
```
[Desktop Entry]
Encoding=UTF-8
Name=Manager XAMPP
Comment=Manager XAMPP
Exec=sudo /opt/lampp/manager-linux-x64.run
Icon=/opt/lampp/htdocs/favicon.ico
Categories=Aplicaciones;Programación;Web
Version=8.1.6
Type=Application
Terminal=1
```

> **nota**: la versión del programa (atributo *Version*) dependerá de la versión que hayamos instalado de XAMPP en el momento concreto. El atributo *Terminal* lo ponemos a 1 para poder especificar el password de superusuario al ejecutar, de lo contrario no funcionará.

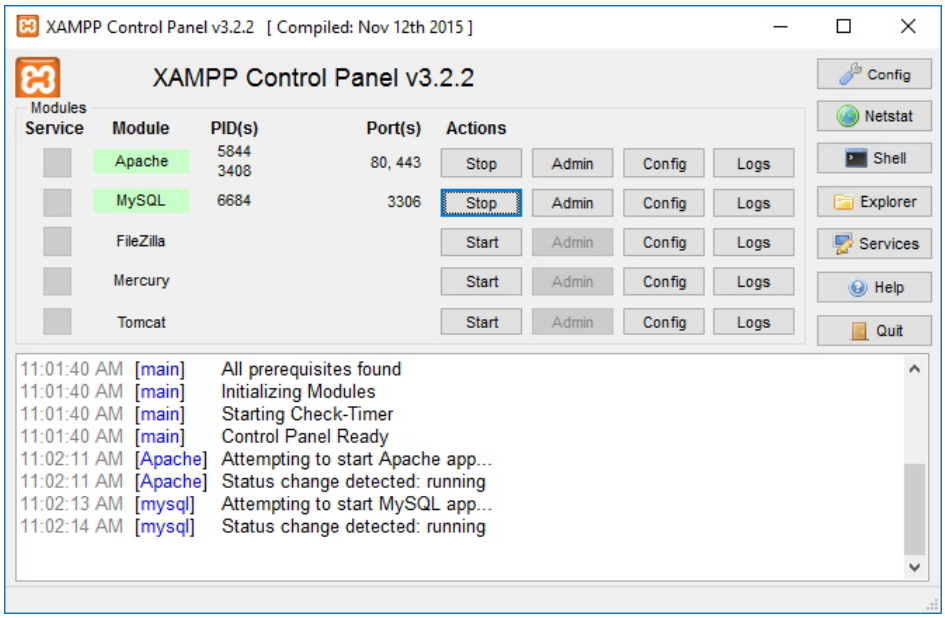
En el caso de **Windows**, dicho manager está en la carpeta de instalación (típicamente *C:\xampp*), en un archivo llamado *xampp-control.exe*, que podemos ejecutar. En el caso de **Mac OSX**, se habrá creado un acceso en la sección de *Aplicaciones* para poder poner en marcha este manager.



El manager nos permitirá lanzar o detener cada servidor. Para las pruebas que haremos deberemos tener iniciados tanto Apache como MySQL. En Linux y Mac OS X tendrá una apariencia como ésta aproximadamente:



En el caso de Windows la apariencia es algo diferente, aunque igualmente funcional:



Por defecto, Apache estará escuchando en el puerto 80 (o 443 para conexiones SSL), y MySQL en el 3306. Podemos modificar estos puertos en los respectivos archivos de configuración ("*httpd.conf*" y "*my.cnf*"), dentro de las carpetas de la instalación de XAMPP (la ubicación concreta de estos archivos varía entre versiones y entre sistemas operativos).

2.3. Laravel

Para trabajar con Laravel, será necesario instalar el gestor de paquetes **composer** mediante el que podremos tanto crear proyectos Laravel como gestionar las dependencias de otros módulos en un proyecto.

> **versión requerida**: en la última actualización de estos apuntes, se dispone de la versión 2.3.x de la herramienta **composer** y de la versión 9.x de Laravel.

2.3.1. Instalando *composer*

Como hemos comentado, la instalación de Laravel se realiza a través del gestor de paquetes **composer**. Ésta es una herramienta muy habitual en ecosistemas PHP, y su labor es similar a la que desempeña el gestor NPM para aplicaciones JavaScript: gestionar las dependencias de un determinado proyecto, descargando, actualizando o desinstalando los paquetes necesarios. En este caso, lo utilizaremos para descargar e instalar el propio framework Laravel.

Composer puede instalarse localmente para cada proyecto web, o de forma global para todo el sistema. Esta última opción es la recomendable en el caso de querer gestionar varios proyectos en nuestro equipo, para no tener que instalarlo en todos ellos.

Linux y Mac OSX

Para instalar *composer* en Linux y Mac OSX, debemos descargar el archivo `composer.phar` de la web oficial y copiarlo renombrado a `composer` desde donde lo hayamos descargado a alguna carpeta que forme parte del PATH del sistema, y activarlo como ejecutable. Por ejemplo:

```
1 mv composer.phar composer
2 sudo mv composer /usr/local/bin/composer
3 sudo chmod +x /usr/local/bin/composer
```

Como último paso, y ya que Composer utiliza el ejecutable de PHP, necesitamos que dicho ejecutable esté también en el PATH del sistema.

- Para Linux deberemos hacer lo siguiente:

```
echo "export PATH=$PATH:/opt/lampp/bin" >> ~/.bashrc
source ~/.bashrc
```

- Para Mac OSX, es posible que se tenga alguna versión previa de PHP instalada con alguna otra herramienta. Para superponer la nueva versión de XAMPP a esta otra, podemos hacer este cambio en el PATH:

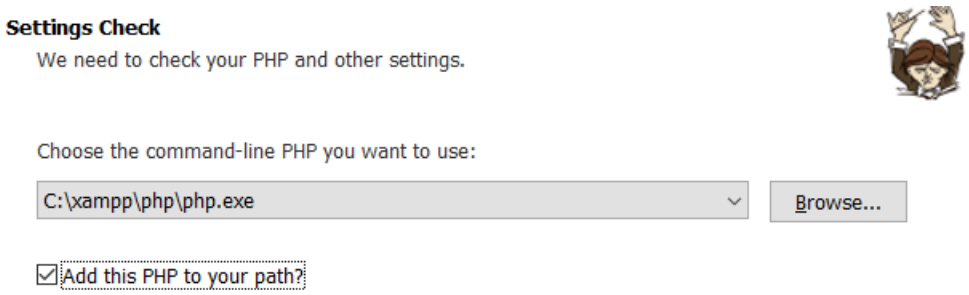
```
echo "export PATH=/Applications/XAMPP/xamppfiles/bin:$PATH" >> ~/.bash_profile
source ~/.bash_profile
```

Además, en el caso de Mac OSX quizá tengamos que editar el archivo de configuración de PHP (*/Applications/XAMPP/xamppfiles/etc/php.ini*) y añadir esta línea al final, para permitir que el comando PHP gestione la memoria:

```
pcre.jit=0
```

Windows

Para instalar *composer* en Windows debemos hacerlo a través de un instalador que también podemos descargar desde la web oficial, en la sección *Windows Installer*. Seguimos los pasos del asistente, y deberemos indicar la ruta donde se encuentra el comando *php*, necesario para poder utilizar la herramienta. Añadimos este comando al PATH del sistema también:



Comprobación de la instalación

Una vez instalado, podemos ejecutar el comando `composer` sin más en un terminal, y comprobar que muestra una salida con las opciones que ofrece:

```
C:\Users\Nacho>composer
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 2.1.6 2021-08-19 17:11:08

Usage:
  command [options] [arguments]

Options:
  -h, --help                     Display this help message
  -q, --quiet                    Do not output any message
  -V, --version                  Display this application version
      --ansi                     Force ANSI output
      --no-ansi                  Disable ANSI output
  -n, --no-interaction           Do not ask any interactive question
      --profile                  Display timing and memory usage information
      --no-plugins               Whether to disable plugins.
  -d, --working-dir=WORKING-DIR  If specified, use the given directory as working directory.
      --no-cache                 Prevent use of the cache
  -v|vv|vvv, --verbose           Increase the verbosity of messages: 1 for normal output, 2 for more verbose output
3 for debug
```

2.3.2. Instalando Laravel

A través de la herramienta `composer` se pueden crear directamente proyectos Laravel, como veremos en el curso. Sin embargo, la sintaxis del comando de creación es algo larga, si la comparamos con el instalador de Laravel, por lo que vamos a instalarlo también. Para hacerlo, usamos la propia herramienta `composer`, con este comando:

```
composer global require laravel/installer
```

En el caso de **Windows**, este comando ya deja el instalador `laravel` listo para poderse ejecutar desde terminal. Para Linux y Mac OSX, deberemos añadirlo al PATH del sistema (en realidad, añadimos la carpeta con las utilidades que *composer* instala de forma global al sistema).

Linux:

```
echo "export PATH=$PATH:$HOME/.config/composer/vendor/bin" >> ~/.bashrc
source ~/.bashrc
```

> **nota**: en algunos sistemas la carpeta que hay que incluir en el PATH es `$HOME/composer/vendor/bin` en lugar de la anterior.

Mac OSX:

```
echo "export PATH=$PATH:$HOME/.composer/vendor/bin" >> ~/.bash_profile
source ~/.bash_profile
```

Con esto, se habrá instalado un comando llamado `laravel`, que podemos utilizar a partir de ahora para crear los proyectos. Podemos probar a ejecutarlo en un terminal para que nos muestre las opciones disponibles, lo que indicará que está correctamente instalado y localizado.

2.3.3. Actualizando Laravel

En general, el comando `laravel` que se instala se encargará de crear proyectos empleando la última versión de Laravel que haya disponible. Así, si por ejemplo lo instalamos cuando aún existía la versión 7 de Laravel pero se publica la versión 8, automáticamente el comando `laravel` nos permitirá crear proyectos de la versión 8 desde ese punto. Sin embargo, es posible que con el tiempo se requiera actualizar el instalador para que los proyectos con nuevas versiones se sigan creando sin problemas.

Para poder actualizar a la versión más reciente de Laravel, tenemos dos opciones, aunque es cierto que ninguna de ellas está recogida en la documentación oficial de Laravel, y lo que aquí se menciona se basa en recomendaciones de webs externas a Laravel.

La primera forma de actualizar es utilizar el comando de actualización:

```
composer global update laravel/installer
```

Sin embargo, esta opción puede no ser suficiente si el cambio es demasiado brusco (por ejemplo, pasar de Laravel 5 a Laravel 7), ya que algunas dependencias que también haya instaladas harían inviable el cambio. En este caso, podemos optar por quitar la versión instalada por completo, e instalar la reciente:

```
composer global remove laravel/installer
composer global require laravel/installer
```

2.4. Node.js

A pesar de que podría parecer que *Node.js* es un ecosistema diferente a Laravel, lo cierto es que con la instalación de Node se incorpora una herramienta muy útil en cualquier aplicación web que utilice librerías JavaScript, como puedan ser Bootstrap o jQuery. Es la herramienta **NPM** (*Node Package Manager*), que permite instalar de forma sencilla estas librerías en cualquier proyecto.

> **versión requerida**: es recomendable tener instalada la última versión LTS (*Long Term Support*). En la última versión de estos apuntes, dicha versión es la 16.x.

Para instalar Node en cualquiera de los sistemas que estamos contemplando (Linux, Windows o Mac OSX) podemos optar por:

- Instalarlo a través del *instalador* correspondiente
- Utilizar la herramienta NVM (*Node Version Manager*). Esta herramienta nos va a permitir tener más de una versión de Node instalada, y poder elegir en todo momento cuál de ellas es la que queremos tener activa. En el caso de Windows, la herramienta NVM que podemos utilizar no es la "oficial", pero sí existe una especie de clon alternativo funcional.

2.4.1. Instalación en Linux

En el caso de distribuciones Linux, como es el caso de la máquina virtual proporcionada, podemos optar por ejecutar un instalador o por usar NVM. En la versión completa de la máquina virtual proporcionada se ha optado por esta última opción (NVM), pero damos aquí los pasos a seguir para ambas opciones.

Opción de instalador

Si optamos por instalar Node.js a través de un instalador, los pasos a seguir son:

1. En primer lugar, si no tenemos instalada la herramienta `curl`, la instalamos con:

```
sudo apt-get install curl
```

1. Después, utilizamos `curl` para recuperar la versión de Node que queramos (versión LTS actual, 14.x):

```
curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash -
```

1. Finalmente, instalamos Node.js

```
sudo apt-get install -y nodejs
```

Si quisiéramos actualizar la versión de Node.js en un futuro, bastaría con repetir los dos últimos comandos, poniendo en el paso 2 la versión a obtener en este caso.

Opción de NVM

Podemos consultar información de esta herramienta en su web oficial en GitHub. Para instalarla, debemos descargarla con el comando `curl` o `wget`, según se explica en la propia web de GitHub. Si optamos por `wget`, el comando es como sigue (en una sola línea):

```
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh |
bash
```

En el caso de no disponer del comando `wget` instalado, podemos o bien instalarlo, o bien emplear este otro comando equivalente, con la orden `curl` (también en una sola línea):

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh |
bash
```

nota: el número de versión `v0.39.1` puede variar. Es preferible consultar la web de GitHub para obtener el comando actualizado.

nota: después de ejecutar el comando anterior, será necesario cerrar el terminal y volverlo a abrir para poder utilizar el comando `nvm`. Si sigue sin reconocer el comando, quizá debamos actualizar con `source ~/.bashrc`

2.4.2. Instalación en Mac OSX

Para distribuciones Mac OSX, tenemos igualmente la opción de usar un instalador o la herramienta NVM.

Opción de instalador

El instalador de Node.js para Mac OSX es muy sencillo. Lo descargamos de la web oficial de Node y lo lanzamos. Si queremos actualizar versión en un futuro, basta con volver a descargar esa última versión y lanzar el instalador correspondiente para actualizarla.

Opción NVM

Para utilizar NVM en Mac OSX seguimos los mismos pasos que para Linux, comentados anteriormente, pero en este caso para actualizar el terminal ejecutaremos `source ~/.bash_profile`.

2.4.3. Instalación en Windows

Opción de instalador

El instalador de Node.js para Windows es igualmente muy sencillo. Lo descargamos de la web oficial de Node y lo lanzamos. Si queremos actualizar versión en un futuro, basta con volver a descargar esa última versión y lanzar el instalador correspondiente para actualizarla.

Opción NVM

Como comentábamos, la herramienta NVM oficial sólo está disponible para sistemas Unix, por lo que funciona en Linux y Mac OSX, pero no en Windows. Como alternativa, existe alguna implementación paralela de *nvm* que podemos hacer servir, como esta. Podemos descargar un instalador (*nvm-setup.zip*) y ejecutarlo para instalar este gestor. Después, desde línea de comandos tendremos disponibles una serie de comandos para gestionar las versiones de Node, como veremos a continuación.

2.4.4. Utilizando NVM

Si hemos optado por instalar Node a través de `nvm`, tendremos disponibles una serie de comandos en el terminal para instalar y gestionar las versiones de Node. Aquí resumimos los más importantes.

Linux y Mac OSX

En el caso de estos sistemas, habremos instalado la versión oficial de NVM, y los comandos que podemos utilizar son estos:

- `nvm install node`: instala la última versión disponible de Node
- `nvm install --lts`: instala la última versión LTS disponible
- `nvm install 12.16.0`: instala la versión especificada de Node
- `nvm uninstall 12.16.0`: desinstala la versión especificada de Node
- `nvm ls-remote`: muestra todas las versiones disponibles para instalar
- `nvm list`: muestra todas las versiones instaladas localmente
- `nvm current`: muestra la versión actualmente activa
- `nvm use 12.16.0`: marca la versión indicada como actualmente activa
- `nvm use --lts`: marca como activa la última versión LTS instalada

En nuestro caso, vamos a instalar la última versión LTS disponible, ya que éstas son las versiones que tienen soporte a largo plazo. Por lo tanto, ejecutaremos los comandos:

```
nvm install --lts
nvm use --lts
```

NOTA: el comando `nvm use` normalmente no es necesario, ya que la instalación automáticamente deja como activa la versión que elegimos. Pero si el terminal no termina de reconocer el comando `node` puede ser necesario ejecutarla.

Windows

En el caso de Windows con la versión alternativa de NVM, los comandos son ligeramente diferentes:

- `nvm install 12.16.0`: instala la versión especificada de Node
- `nvm uninstall 12.16.0`: desinstala la versión especificada de Node
- `nvm list`: muestra todas las versiones instaladas localmente
- `nvm list available`: muestra todas las versiones disponibles para instalar con esta adaptación de NVM.
- `nvm use 12.16.0`: marca como activa la versión de Node especificada (previamente instalada).

Para instalar la versión LTS disponible, tendremos que ver su número en la web oficial de Node, y luego ejecutar el comando correspondiente. Por ejemplo:

```
nvm install 16.16.0
nvm use 16.16.0
```

2.4.5. Prueba de Node

Podemos ejecutar ahora `node -v` en el terminal y comprobar que nos muestra el número de versión adecuado. También podemos ejecutar el comando `npm - v` para comprobar la versión que se ha instalado del gestor NPM (que no tiene por qué coincidir con la de Node).

2.5. Recursos adicionales

Para facilitar la labor de instalar y/o trabajar con el software propuesto, se proporcionan dos máquinas virtuales basadas en sistemas Lubuntu (versión 20). En ambos casos, se cuenta con un usuario *alumno* con password *alumno*.

- Máquina virtual base
- Máquina virtual con software instalado

Es necesario utilizar el programa VirtualBox para ejecutar estas máquinas virtuales. También se puede cambiar la configuración por defecto de dichas máquinas virtuales, para darles más memoria RAM o núcleos de procesador, si se cree conveniente.

instalar docker bitnami/Laravel

Otra opción (**vamos a utilizar esta en nuestros equipos linux**) será utilizar una *stack* de contenedores docker que contendrá servidor Apache, servidor BBDD MariaDB, Laravel y phpMyadmin.

1. Lo primero de todo es crear una carpeta con el nombre del proyecto y accedemos ella.

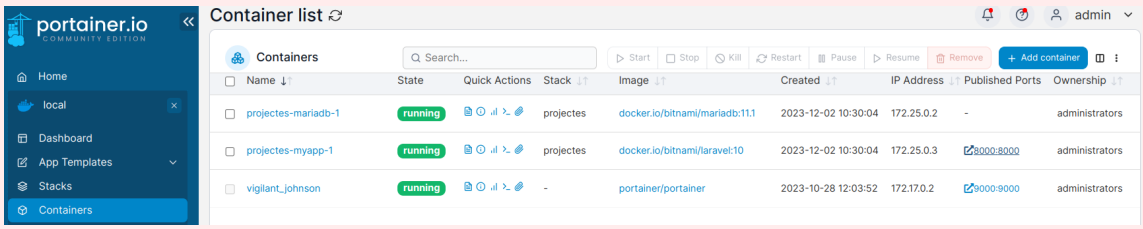
   Por ejemplo, creamos el proyecto *prularavel* dentro de nuestra carpeta de proyectos:

   ```
   $ mkdir ~/dwes/proyectos/nombreProyecto
   ```

2. Accedemos dentro de la carpeta de este nuevo proyecto.
3. Utilizar el fichero `docker-compose.yml` (que tenemos en nuestra carpeta del curso). Una vez descargado el archivo en nuestra carpeta que acabamos de crear con el nombre del proyecto, lanzamos el siguiente comando por consola para instalar todas las dependencias y crear las imágenes de Docker correspondientes.

   ```
   sudo docker-compose up -d
   ```

4. Si utilizamos el contenedor `Portainer` para la gestión de nuestros contenedores, podremos observar que estarán en marcha nuestros dos contenedores (pertenecientes al servidor web y servidor de bases de datos):



aporte

Un aporte, o instalación, a tener en cuenta, podría ser la de instalar `Tailwind CSS`. Este software nos va a proporcionar, de manera sencilla y cómoda, una opción de utilizar CSS.

Para ello, seguir las instrucciones del anexos - instalación de Tailwind CSS.

3. 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.

> **importante**
>
> En este enlace podremos encontrar la documentación oficial de Laravel (en este caso, la versión 10.x). No dudes en consultarlo cuando tengas dudas!

3.1. Crear proyectos Laravel

Para crear proyectos Laravel, tenemos dos alternativas:

- Emplear el comando `laravel` que hemos instalado en sesiones anteriores mediante *composer*
- Utilizar el propio comando `composer` para crear el proyecto. Esta opción será la recomendada si queremos crear proyectos con versiones de Laravel que no sean la última, o si por algún motivo no funciona la opción anterior.

3.1.1. Usando el comando *laravel*

Si empleamos el comando `laravel` para crear proyectos (asumiendo que ya lo tendremos instalado de la sesión de software necesario), nos deberemos ubicar en la carpeta donde queramos crear el proyecto 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 la carpeta donde queramos tener este proyecto, comenzamos escribiendo este comando:

```
laravel new biblioteca
```

Esto creará un proyecto `biblioteca` en una subcarpeta con el mismo nombre.

3.1.2. Usando el comando *composer*

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 nombre_proyecto
```

En nuestro caso, para el ejemplo de la biblioteca que vamos a ir desarrollando, escribiremos este comando:

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

Del mismo modo que en el caso anterior, se creará una carpeta `biblioteca` con el contenido inicial del proyecto dentro, empleando la última versión de Laravel disponible.

3.1.3. Crear proyectos Laravel usando versiones anteriores

Al crear un proyecto con el comando `laravel new`, se creará con la última versión disponible de Laravel. 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 7:

```
composer create-project --prefer-dist laravel/laravel prueba 7.x
```

3.1.4. 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:

```
php artisan list
```

Este otro comando muestra la versión de Laravel del proyecto en el que estamos:

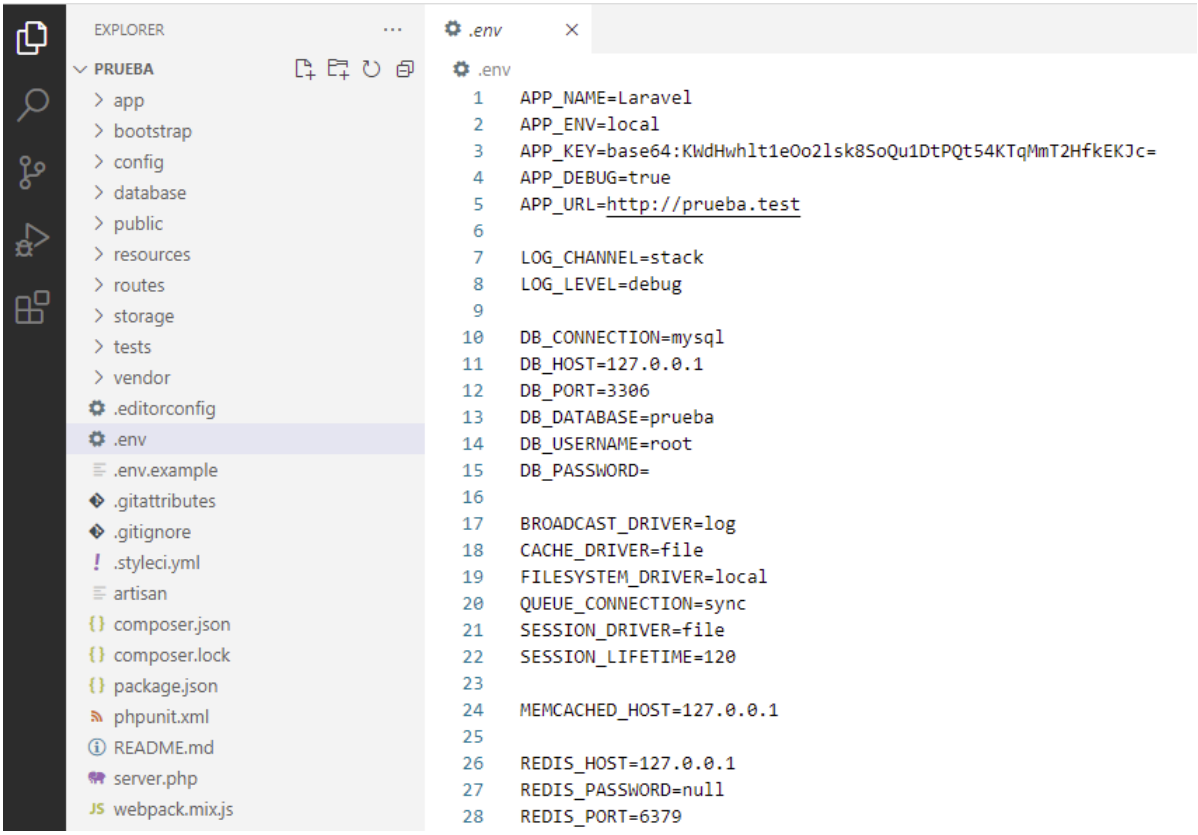
```
php artisan --version
```

Para escribir estos comandos, podemos abrir la carpeta del proyecto con Visual Studio Code, y abrir el terminal integrado en este IDE (menú *Ver* > *Terminal*). Esto nos ubicará automáticamente en la carpeta del proyecto, y podemos directamente ejecutar estos comandos desde ahí.

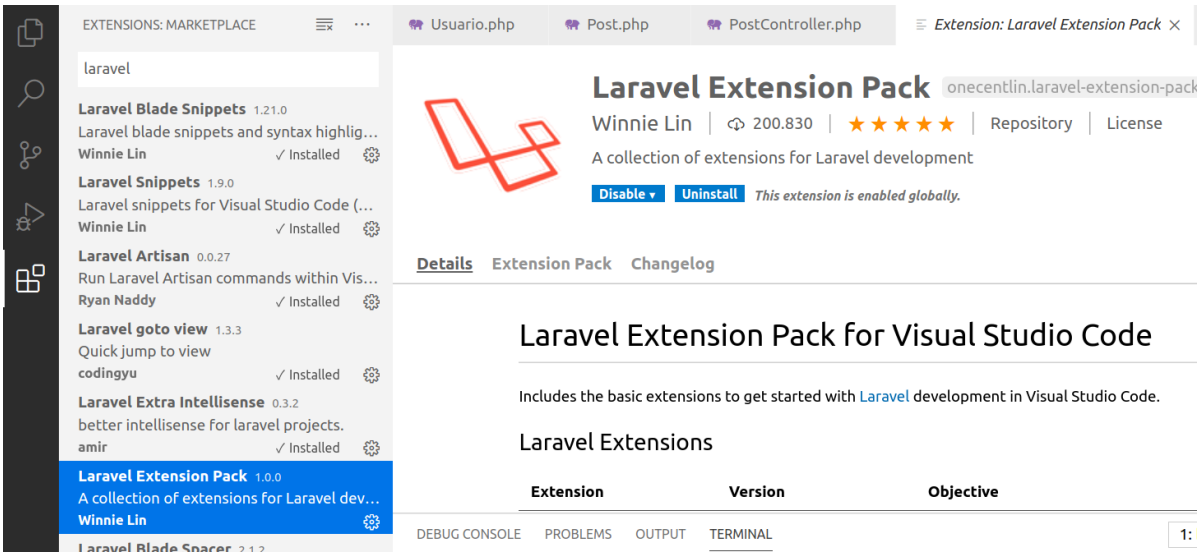
3.1.5. Laravel y Visual Studio Code

Como hemos comentado, podemos gestionar muy cómodamente los proyectos Laravel desde el IDE Visual Studio Code. Basta con abrir la carpeta principal del proyecto desde el propio IDE, y eso puede hacerse desde el menú *Archivo* > *Abrir Carpeta*, o bien arrastrando la carpeta a la zona principal del IDE, e incluso, dependiendo de cómo hayamos hecho la instalación, también podemos hacer clic derecho sobre la carpeta del proyecto en el explorador de archivos, y elegir *Abrir con Code* en el menú contextual.

Una vez abierto, tendremos la estructura del proyecto en la parte izquierda (pestaña de explorador de archivos), y podremos ir editando los archivos en la parte derecha.



Para facilitar aún más la gestión de proyectos Laravel, podemos instalar alguna extensión adicional a Visual Studio Code. Una de las más populares es *Laravel Extension Pack*, que a su vez contiene una serie de extensiones para resaltar sintaxis, editar vistas, etc:



3.2. Estructura de un proyecto Laravel

Los proyectos Laravel se pueden gestionar abriendo la carpeta directamente desde nuestro IDE (Visual Studio Code). 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, de acuerdo a la versión actualizada de estos apuntes.

- ```
  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*
    - `Models`: almacena los modelos de datos de la aplicación (clases que se utilizarán para gestionar los objetos que intervienen en el sistema).
    - `Providers`: contiene los proveedores de servicios de la aplicación, más los que podamos definir nosotros.
    - Además, podemos definir aquí otras carpetas que podamos necesitar. Por ejemplo, en versiones anteriores de Laravel se creaba aquí a mano la carpeta *Models* para gestionar los modelos de datos.
- `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.
- `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, y además podemos definir o generar 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 u optimizar). 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. 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.

3.2.1. Configuración general del proyecto

De entre la estructura de carpetas de un proyecto Laravel vista anteriormente, echaremos ahora un rápido vistazo a dónde se encuentra la configuración general del proyecto.

Por un lado, disponemos de un archivo `.env` en la raíz del proyecto, que básicamente contiene una serie de variables de entorno generales. Por ejemplo, se tiene la variable `APP_NAME` con el nombre que queremos que tenga la aplicación, o un conjunto de variables que utilizaremos más adelante para conectar con la base de datos, entre otras cosas:

```
APP_NAME=Laravel
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
...
```

En general, los cambios de configuración es preferible hacerlos en este archivo `.env`, de forma que en los archivos de la carpeta `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.

Por otra parte, la carpeta `config` contiene unos archivos generales de configuración. Iremos viendo algunos de ellos en sesiones posteriores, pero, para empezar, podemos echar un vistazo al archivo `config/app.php`, que contiene parámetros de configuración general de la aplicación. Por ejemplo, podemos modificar el nombre de la aplicación, en la propiedad `name`, aunque, como vemos, el nombre lo coge de la propiedad `APP_NAME` del archivo `.env`.

```
'name' => env('APP_NAME', 'Laravel'),
```

Es más habitual modificar los valores del archivo `.env` que los que hay en este archivo de configuración. Pero, en algunos casos, sí convendrá acudir a alguno de los archivos de esta carpeta y modificar la información que contiene.

3.3. 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.

3.3.1. 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. Por ejemplo, hay proveedores de servicios para acceso a la base de datos (*DatabaseServiceProvider*), autenticación de usuarios (*AuthServiceProvider*), etc.

3.3.2. 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:

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

3.3.3. 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.

```
1 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.

3.4. Prueba de proyectos Laravel

Existen varias formas de probar o poner en marcha un proyecto Laravel, dependiendo de si estamos desarrollándolo y probándolo (en cuyo caso buscamos ponerlo en marcha fácilmente), o si ya lo hemos puesto en producción. Resumidamente, veremos tres formas de poner en marcha el proyecto:

- De forma local al proyecto, a través de la herramienta `artisan`
- De forma local a XAMPP, ubicando el proyecto en una carpeta predefinida
- De forma general en el sistema, definiendo un *host virtual* conectado con Apache

La tercera forma será la que necesitemos utilizar si queremos poner la web en producción en un servidor real, ya que en estos casos cada aplicación tiene su propia configuración y ubicación diferente al resto. Las otras dos primeras opciones pueden resultarnos útiles para pruebas sencillas durante el desarrollo.

3.4.1. Permisos en carpetas del proyecto

Para poder probar nuestro proyecto Laravel, además de configurar y poner en marcha el servidor correspondiente es necesario habilitar permisos de acceso y escritura a ciertas carpetas del proyecto, especialmente en sistemas Linux o Mac, si movemos el proyecto a una carpeta con permisos reducidos.

- 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 bajo Linux o Mac OSX, 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.

3.4.2. Puesta en marcha con el comando *artisan*

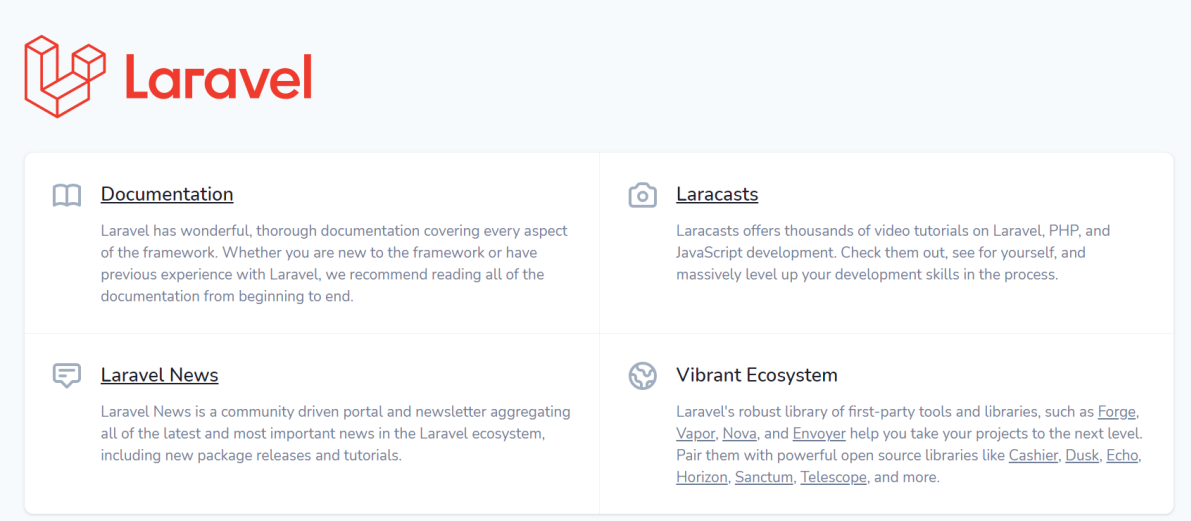
La forma más sencilla de probar nuestro proyecto es a través del comando `artisan`. Nos ubicamos en la carpeta raíz del proyecto (recuerda que esto se puede hacer automáticamente abriendo la carpeta con Visual Studio Code y abriendo su terminal), y ejecutamos este comando:

```
php artisan serve
```

Esto habilitará un pequeño servidor local, y se nos indicará en un mensaje en el terminal la URL que podemos emplear para probar el proyecto, que suele ser algo así:

```
Starting Laravel development server: http://127.0.0.1:8000
```

Conectando a la URL que se nos facilita, veremos la pantalla de inicio de nuestro proyecto, que será algo así (aunque el diseño de la página de inicio puede variar dependiendo de la versión de Laravel que usemos):



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

> **NOTA**: si nuestra aplicación necesita acceder a una base de datos, además de arrancarla con el comando `artisan` visto antes, también tendremos que tener en marcha el servidor de bases de datos correspondiente, a través del manager de XAMPP.

3.4.3. Puesta en marcha en carpeta predefinida de XAMPP

Como segunda alternativa relativamente rápida, podemos ubicar la carpeta del proyecto dentro de la carpeta predefinida de XAMPP para alojar webs:

- En el caso de *Windows* es la carpeta *htdocs* dentro de la carpeta de instalación de XAMPP. Típicamente la ruta es *C:\xampp\htdocs*
- En el caso de *Linux*, en particular en la distribución de nuestra máquina virtual, es la carpeta */opt/lampp/htdocs*
- En el caso de *Mac OSX*, la ruta es */Applications/XAMPP/xamppfiles/htdocs*

Por ejemplo, si nuestro proyecto se llama *biblioteca* y queremos moverlo a esta carpeta en Linux, escribimos este comando desde la carpeta que contiene nuestro proyecto:

```
1 sudo mv biblioteca /opt/lampp/htdocs
```

A continuación debemos poner en marcha el servidor Apache (empleando el *manager* de XAMPP), y luego acceder a la URL *http://localhost/biblioteca/public*. Obtendremos la misma página de inicio que en el caso anterior.

> **NOTA**: al mover la carpeta del proyecto a la ubicación de XAMPP en Linux, ten en cuenta que esa carpeta es de acceso restringido. Deberemos dar permisos a las carpetas indicadas en la sección 4.1 para que la aplicación se pueda poner en marcha adecuadamente.

3.4.4. Puesta en marcha como host virtual

En el caso de una puesta en producción real, o si queremos dejar una aplicación correctamente configurada con Apache, hay que realizar una serie de pasos previos, tales como asociarlo a un *virtual host* (de Apache, en nuestro caso). 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 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 de configuración de Apache:

- Archivo `/opt/lampp/etc/httpd.conf` en Linux
- Archivo `C:\xampp\apache\conf\httpd.conf` de la carpeta de instalación de XAMPP en Windows (suponiendo que lo hemos instalado en *C:\xampp*)
- Archivo `/Applications/XAMPP/xamppfiles/etc/httpd.conf` en Mac OSX

En todos los casos, debemos asegurarnos de que está descomentada la línea que hace referencia al lugar donde se definen dichos *virtual hosts*. Debe quedar así en Linux y Mac OSX:

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

O así en Windows, por ejemplo:

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

2. Añadir el nuevo nombre de dominio

Después, debemos editar el archivo *hosts* del sistema y asignar un nombre de dominio (local) a nuestro proyecto. Dicho archivo de hosts es:

- `/etc/hosts` en Linux
- `C:\Windows\System32\drivers\etc\hosts` en Windows
- `/private/etc/hosts` en Mac OSX

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 en el paso 1: el archivo `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>
```

NOTA: la ruta indicada en el atributo *DocumentRoot* dependerá, evidentemente, de la ruta donde tengamos alojado nuestro proyecto *biblioteca*. El ejemplo anterior se ha ilustrado suponiendo que el proyecto *biblioteca* se ubica en una carpeta *ProyectosLaravel* en la carpeta de usuario de nuestra máquina virtual.

3.5. Importando / Exportando un proyecto Laravel

Vamos a indicar ahora unas instrucciones necesarias en el caso de que queramos importar un proyecto Laravel a un nuevo ordenador, o llevarlo a otro diferente, descargándolo de, por ejemplo, un repositorio GitHub, o comprimido en un archivo ZIP o similar. 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.

3.5.1. Exportar un proyecto

Si queremos exportar o compartir un proyecto Laravel de forma externa, podemos:

- Compartirlo en un repositorio remoto tipo GitHub o similar. En este caso, el propio archivo `.gitignore` que se crea en el proyecto indica qué elementos no van a subirse al repositorio. De entre estos elementos, conviene destacar el archivo de configuración `.env`, y las carpetas `vendor` y `node_modules`.
- Comprimirlo en un archivo que poder distribuir o instalar en otras partes. En este caso, debemos tomar las mismas precauciones que cuando lo compartimos vía GitHub, y eliminar los elementos que no sean necesarios.

NOTA IMPORTANTE: cuando comprimas un proyecto Laravel, procura comprimir la carpeta entera desde fuera, para así incluir también los archivos ocultos, como `.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`), archivos que no quieras compartir directamente (`.env`, si es el caso) sal a la carpeta padre y comprímela.

3.5.2. Importar un proyecto existente

A la hora de importar en un nuevo ordenador un proyecto existente (bien descargándolo de GitHub, o descomprimiéndolo de un archivo ZIP), debemos dar estos pasos (una vez descargado o descomprimido el proyecto).

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

Como hemos comentado antes, el archivo `.env` es uno de los que no se incluye por defecto en un repositorio GitHub, ya 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 este curso.

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
```

4. referencias

- Nacho Iborra Baeza.