UD10.1 – Laravel pasos previos

2º CFGS
Desarrollo de Aplicaciones Web

2022-23

1.- Frameworks PHP

Un **framework** (marco de trabajo) proporciona una serie de conceptos, prácticas y tecnologías para resolver un problema.

En el caso de **PHP** la mayoría de frameworks incorporan una estructura de directorios para trabajar con el patrón **MVC**.

También ofrecen una implementación del **modelo de datos**, de la **conexión a la base de datos** y de un **sistema de URL's amigables** entre otras características.

1.- Frameworks PHP

- **Laravel**: (2011) Su filosofía es poder desarrollar proyectos de manera elegante y simple. Tiene una gran comunidad detrás.
- **Symfony**: (2005) Al tener más "camino" que Laravel tiene una estructura más consolidada. A principio se necesitaban instalar muchos módulos que luego no se usaban, actualmente es más modular.
- **CodeIgniter**: (2006) Más ligero que los dos anteriores, tuvo una época de abandono pero ha vuelto a coger fuerza seguramente por su simplicidad.
- **CakePHP**: (2005) Similar a CodeIgniter en cuanto a simplicidad y facilidad de uso.
- **Zend**: (2006) Bastante popular pero con menos visibilidad que los anteriores, a la altura que CakePHP.
- **Phalcon**: (2012) Potente capacidad de procesamiento de paginas PHP. Posibilidad de ser usado como microframework (más ligero y con funcionalidades muy específicas) o como framework completo.
- ...

1.- Frameworks PHP

¿Cuál elegir?

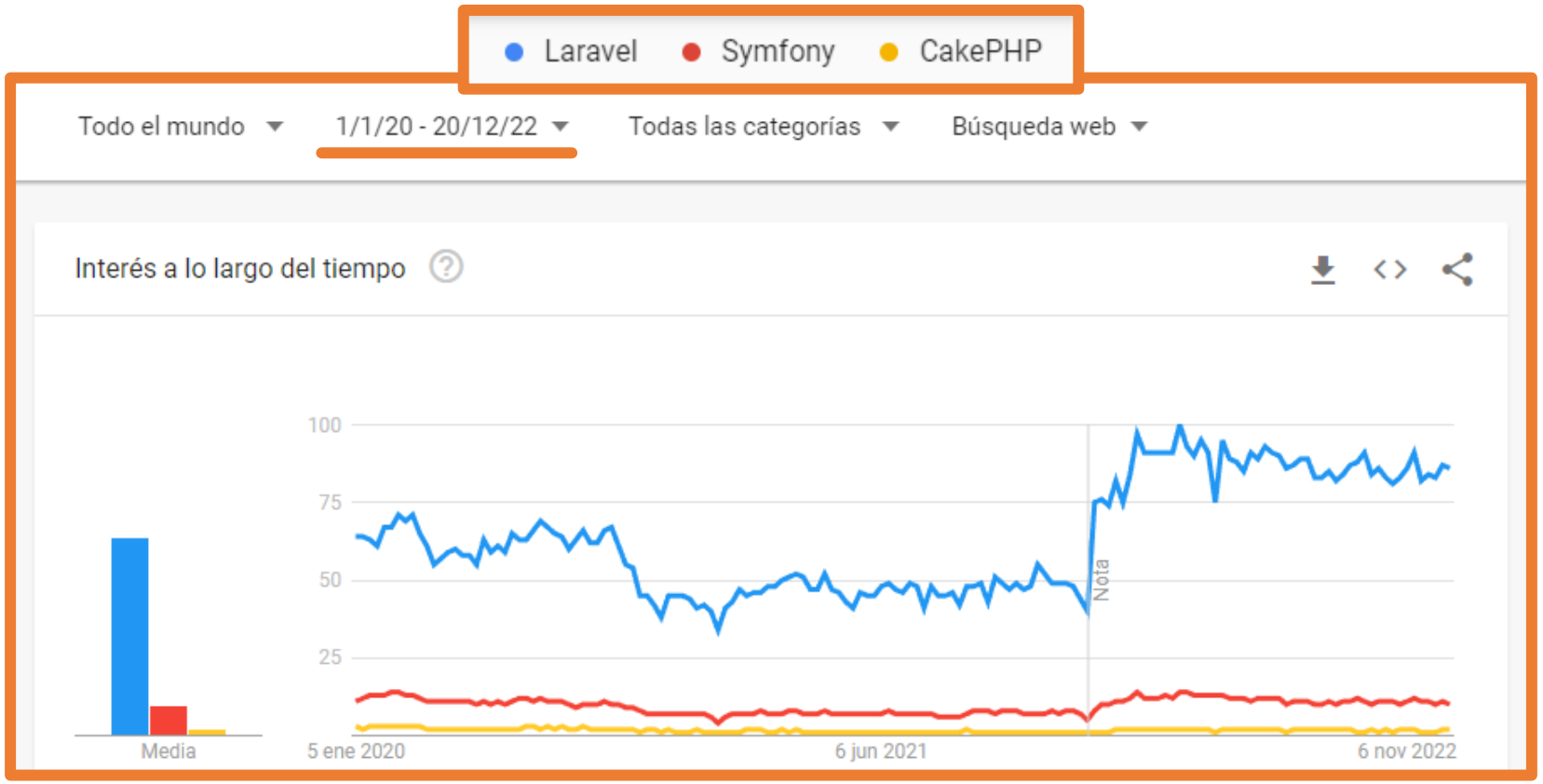
Siempre va a depender de varios parámetros:

- Tipo de proyecto a desarrollar.
- Conocimientos previos.
- La curva de aprendizaje del framework.
- La comunidad que haya detrás.
- …

1.- Frameworks PHP

¿Por qué Laravel? → Muy popular.

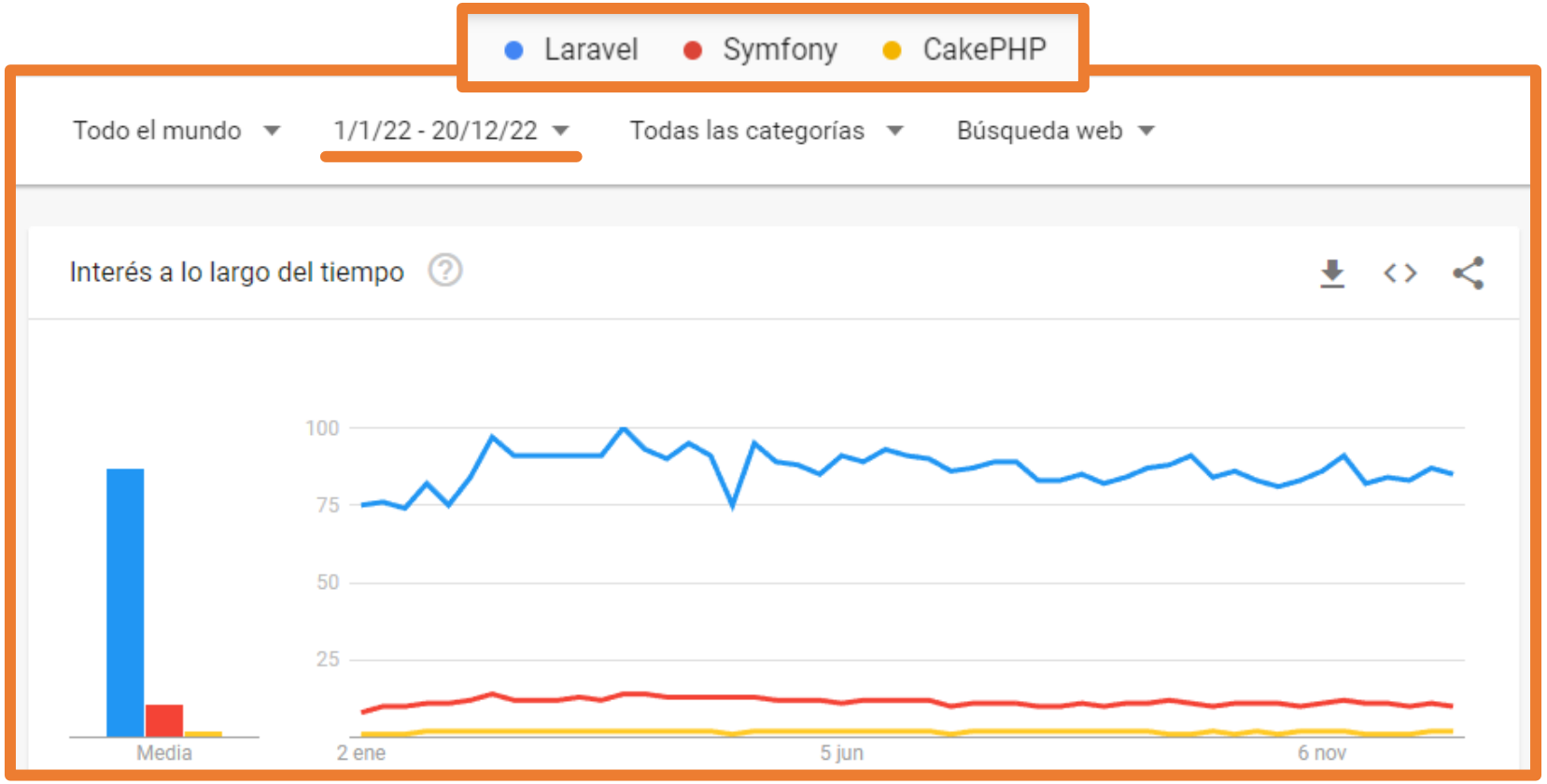
Cantidad de búsquedas 1 enero 2020 a 20 diciembre 2022 (Google Trends)



1.- Frameworks PHP

¿Por qué Laravel? → Muy popular.

Cantidad de búsquedas 1 enero 2022 a 20 diciembre 2022 (Google Trends)



1.- Frameworks PHP

¿Por qué Laravel?

- Gran comunidad detrás.
- Buena documentación tanto en la web oficial como en otras páginas.
- Librerías adicionales que añaden funcionalidades muy útiles como son un ORM o plantillas.

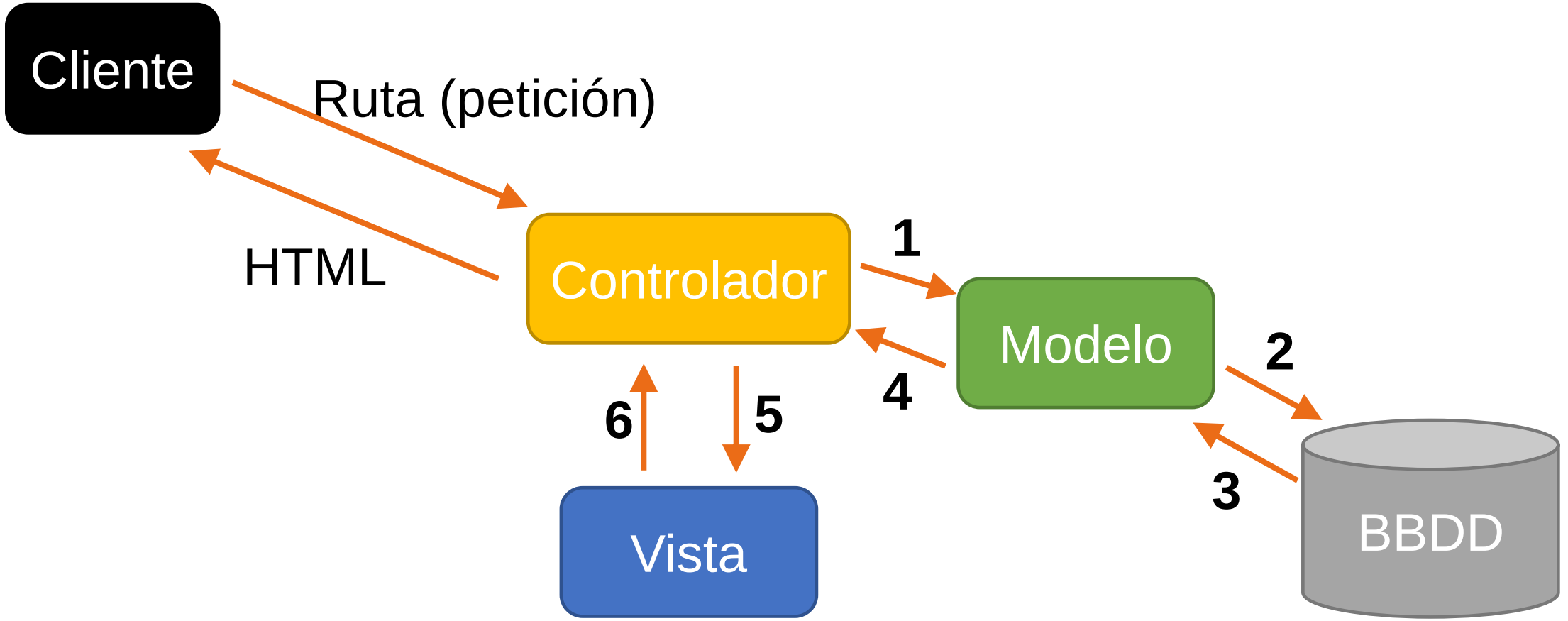
1.- Frameworks PHP

Una vez se conoce un framework de PHP es muy sencillo asimilar cualquiera de los otros llegado el momento.

Así que por sus características Laravel es una buena opción para empezar.

1.- Frameworks PHP

La mayoría de frameworks de PHP trabajan con MVC de la siguiente manera



2.- Requisitos ejecución Laravel 9

- Servidor web con soporte >= PHP 8.0
- PHP (mínimo versión 8.0).
- Servidor de base de datos.

Se puede utilizar un sistema XAMPP o bien instalar cada componente individualmente (Despliegue de Aplicaciones Web).

3.- Visual Studio Code

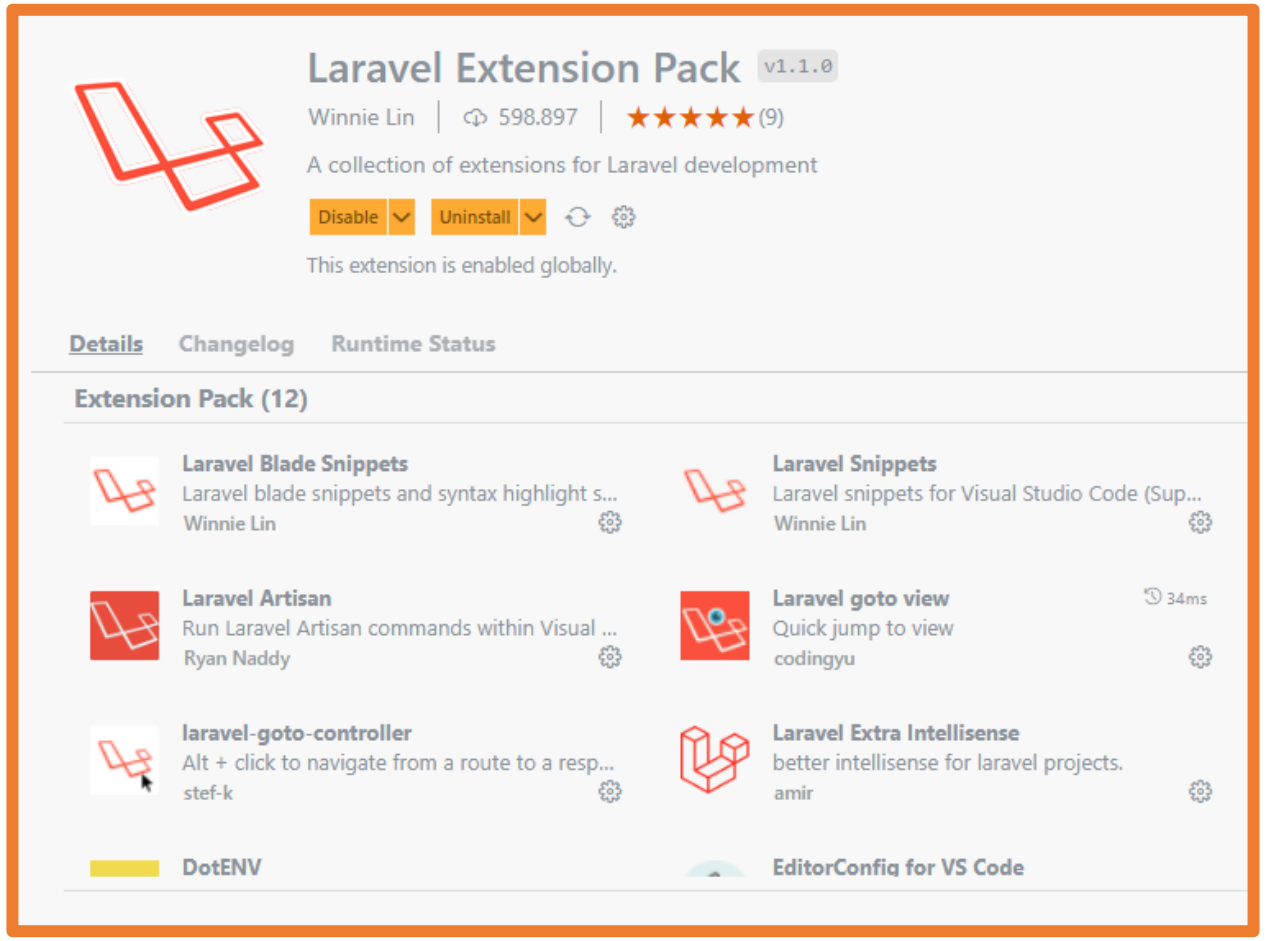
Como ya se sabe, para desarrollar una aplicación web simplemente se necesita un editor de código.

Para un proyecto Laravel, el editor **Visual Studio Code** (VSC) es idóneo puesto que permite trabajar con carpetas como si fueran proyectos de manera que el propio VSC es capaz de tener un control sobre todos los archivos del proyecto.

Además, incorpora un **terminal integrado** muy útil ya que el terminal se usa bastante cuando se desarrolla un proyecto Laravel. Al abrir el terminal integrado ya se encuentra situado en la carpeta del proyecto.

3.- Visual Studio Code

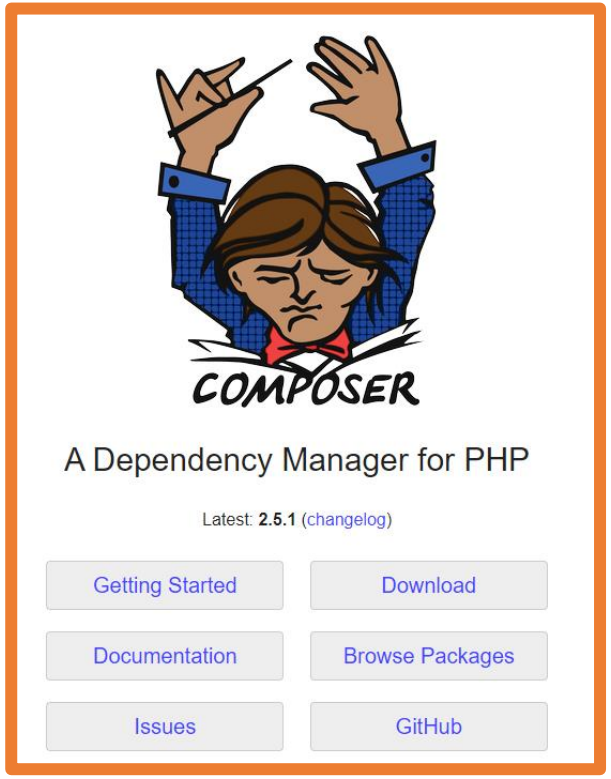
VSC dispone del conjunto de extensiones **Laravel Extension Pack** que instalará extensiones muy útiles para el desarrollo de aplicaciones Laravel.



4.- Composer

Composer es un gestor de dependencias de PHP. Composer se encarga de "instalar" todos los archivos necesarios para un proyecto Laravel.

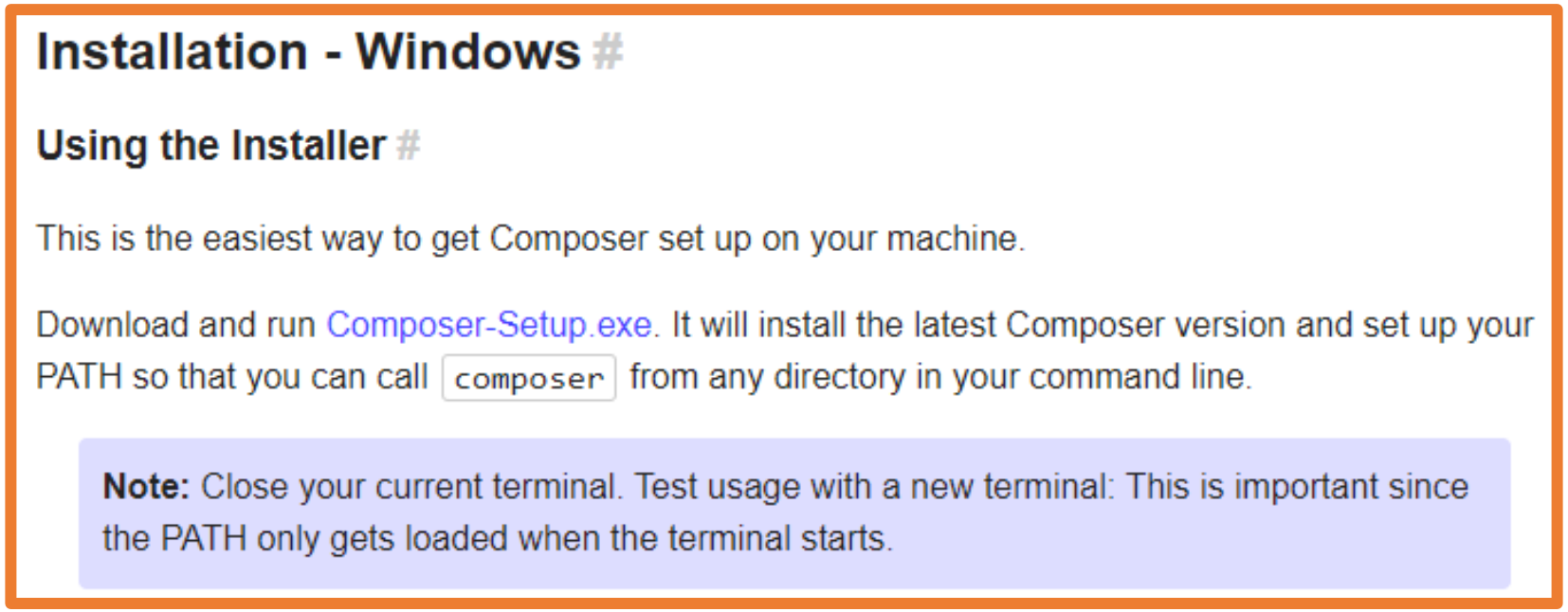
https://getcomposer.org/



4.- Composer

Windows

Instalar Composer es tan sencillo como descargar el instalador, ejecutarlo y seguir los pasos.



Installation - Windows

Using the Installer

This is the easiest way to get Composer set up on your machine.

Download and run Composer-Setup.exe. It will install the latest Composer version and set up your PATH so that you can call `composer` from any directory in your command line.

Note: Close your current terminal. Test usage with a new terminal: This is important since the PATH only gets loaded when the terminal starts.

4.- Composer

Linux y macOS

En Linux se deben realizar algunos pasos más que están detallados en la web oficial de Composer.

En clase ya se realizó la instalación y está disponible para su uso.

4.- Composer

Composer se utiliza mediante la línea de comandos, simplemente se debe escribir "composer" seguido de las opciones que se necesiten.

En Linux se deberá escribir "php composer.phar" a no ser que se cambie siguiendo la documentación oficial.

Algunas ejecuciones importantes de Composer:

```
# composer -V
# composer self-update
```

5.- Instalando el comando laravel

El comando "composer" se usa para crear el proyecto Laravel y para actualizar sus dependencias.

Mediante el comando composer se puede instalar un comando nuevo para facilitar la creación de proyectos Laravel, el comando "laravel".

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

Añadir al $Path el directorio bin de Composer

- Windows: %USERPROFILE%\AppData\Roaming\Composer\vendor\bin
- macOS: $HOME/.composer/vendor/bin
- Distribuciones GNU/Linux: $HOME/.config/composer/vendor/bin
  $HOME/.composer/vendor/bin

Actualizar el comando (sacado de webs externas a Laravel):

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

Puede ser que que el anterior comando no sea suficiente si el framework ha sufrido cambios sustanciales:

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

6.- Extra: Node.js

Aunque en clase no se utilizará Node.js, a continuación se indica cómo instalarlo por si se necesita usar junto a Laravel.

Node.js es un entorno de ejecución de JavaScript para programación del lado del servidor.

Aunque Node.js es un ecosistema diferente a Laravel, con la instalación de Node.js se incorpora una herramienta útil para cualquier aplicación que use librerías Javascript como Bootstrap o jQuery.

Esta herramienta es **NPM** (Node Package Manager).

https://nodejs.org/es/download

6.- Extra: Node.js

Instalación

- **Windows**:

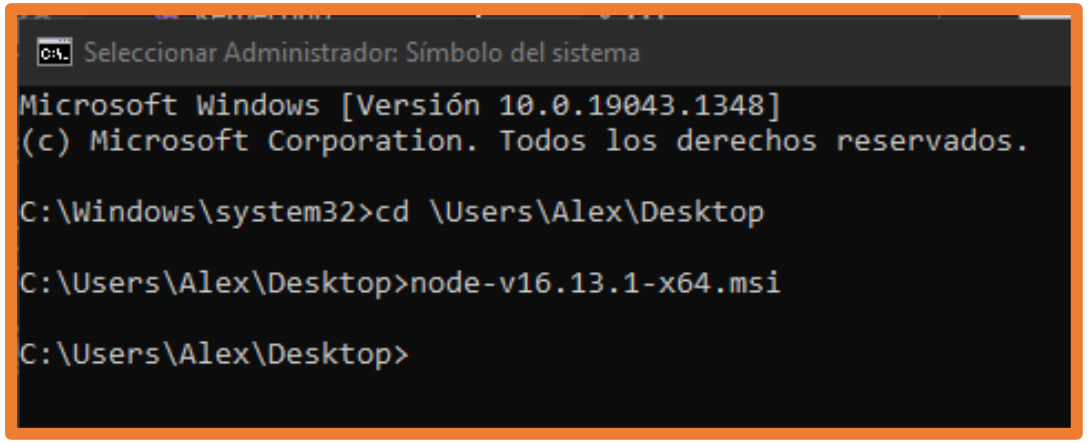
  Descargar el instalador

  Para poder ejecutar el instalador se necesitan permisos de administrador.

  Ejecutar Símbolo de sistema como administrador.

  Ir al directorio de descarga y ejecutar el archivo.

  Se abrirá el instalador. Seguir los pasos.



6.- Extra: Node.js

Instalación

- **MAC**: descargar el instalador y ejecutarlo

- **Linux**: mediante la herramienta NVM

  Instalar NVM mediante uno de los dos comandos siguientes:

```
# curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
# wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
```

  Comprobar en su GitHub oficial el comando.

  Usar el comando nvm para instalar Node.js

- `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

6.- Extra: Node.js

Una vez instalados desde la línea de comandos se puede comprobar usando los comandos:

```
# node -v

# npm -v
```

```
C:\Users\Alex\Desktop>node -v
v18.12.1

C:\Users\Alex\Desktop>npm -v
9.2.0
```

7.- Crear un proyecto Laravel

Para trabajar con proyectos Laravel se debe tener a mano siempre un terminal.

Con Visual Studio Code se incorpora un terminal que facilita mucho la tarea.

Al crear un proyecto Laravel se crearán toda la estructura de carpetas y archivos necesarios para comenzar a trabajar.

No es necesario crear el proyecto dentro de la carpeta htdocs, pero esto se explicará cuando se estudie la estructura de un proyecto Laravel.

7.- Crear un proyecto Laravel

- Mediante el comando **composer**: si no se indica la versión composer siempre crea el proyecto con la última versión disponible de Laravel.

```
# composer create-project --prefer-dist laravel/laravel nombreProyecto

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

En clase se usará el comando "composer" en su variante de Linux: "php composer.phar"

```
# php composer.phar create-project --prefer-dist laravel/laravel nombreProyecto
```

Además, en clase con Lliurex, se aconseja no crear los proyectos dentro de la carpeta "Documentos" ya que puede dar errores y la ejecución es mucho más lenta.

7.- Crear un proyecto Laravel

- Mediante el comando **laravel**: si se ha instalado el comando laravel se puede utilizar para crear un proyecto de una manera más sencilla.

  Con el comando laravel se creará el proyecto con la última versión instalada con el comando laravel por lo que es importante tener actualizado el comando.

  ```
  # laravel new nombreProyecto
  ```

8.- Comando php artisan

Cuando se crea un proyecto Laravel también se instala en el directorio raíz del proyecto la herramienta **artisan**.

Artisan es un interfaz de línea de comandos (CLI: Command Line Interface).

Gracias a **artisan** se facilita la gestión de un proyecto Laravel.

Desde el directorio del proyecto con la línea de comandos se puede ejecutar el siguiente comando para comprobar que se ha instalado correctamente:

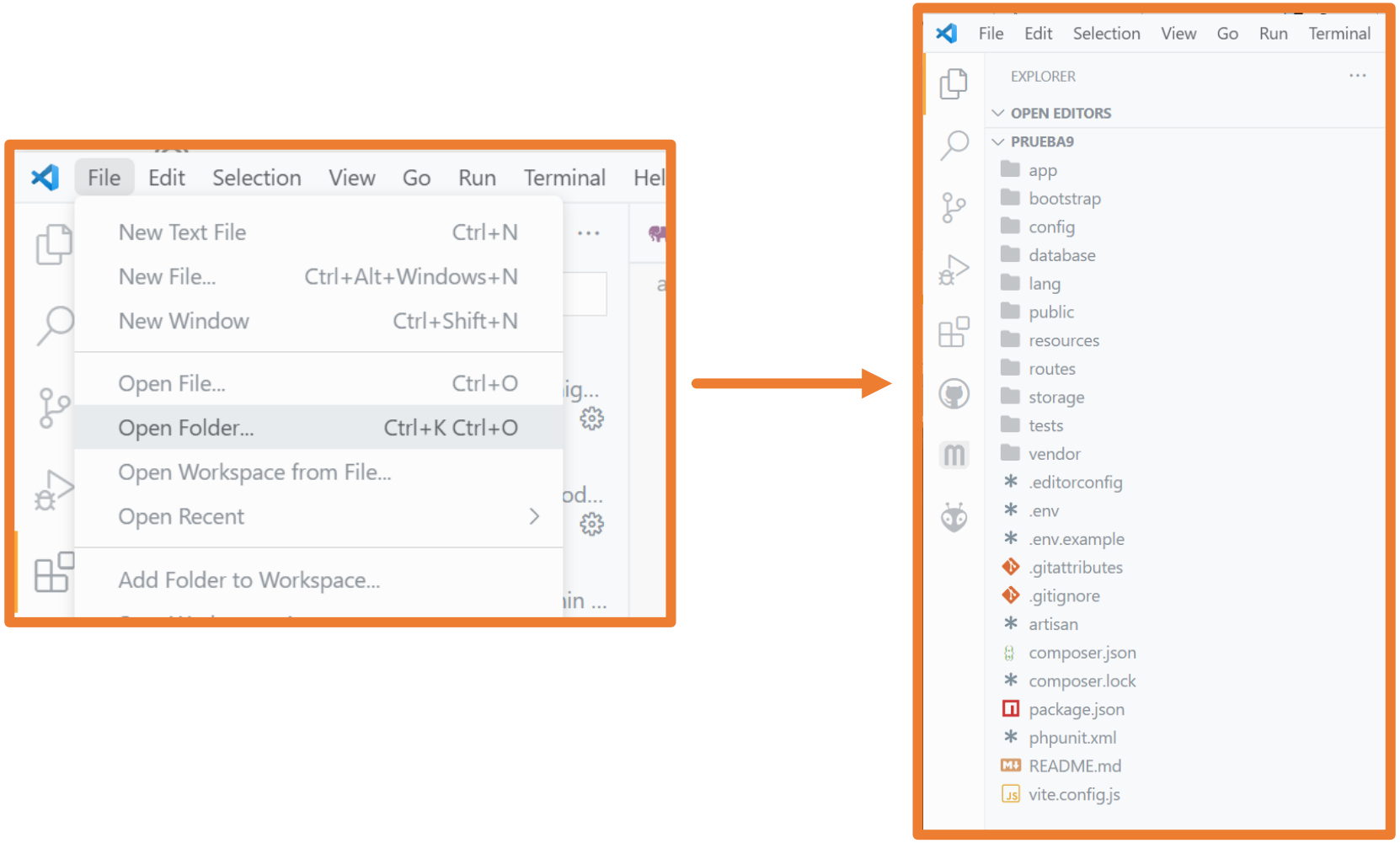
```
# php artisan
```

8.- Comando php artisan

```
PS C:\xampp\htdocs\Proyectos Laravel\prueba9> php artisan
Laravel Framework 9.45.1

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display help for the given command. When no command is given display help fo
r the list command
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi|--no-ansi  Force (or disable --no-ansi) ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --env[=ENV]       The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose
output and 3 for debug

Available commands:
  about                Display basic information about your application
  clear-compiled       Remove the compiled class file
  completion           Dump the shell completion script
  db                   Start a new database CLI session
  docs                 Access the Laravel documentation
  down                 Put the application into maintenance / demo mode
  env                  Display the current framework environment
  help                 Display help for a command
  inspire              Display an inspiring quote
  list                 List commands
  migrate              Run the database migrations
  optimize             Cache the framework bootstrap files
  serve                Serve the application on the PHP development server
  test                 Run the application tests
  tinker               Interact with your application
```

9.- Estructura de un proyecto Laravel

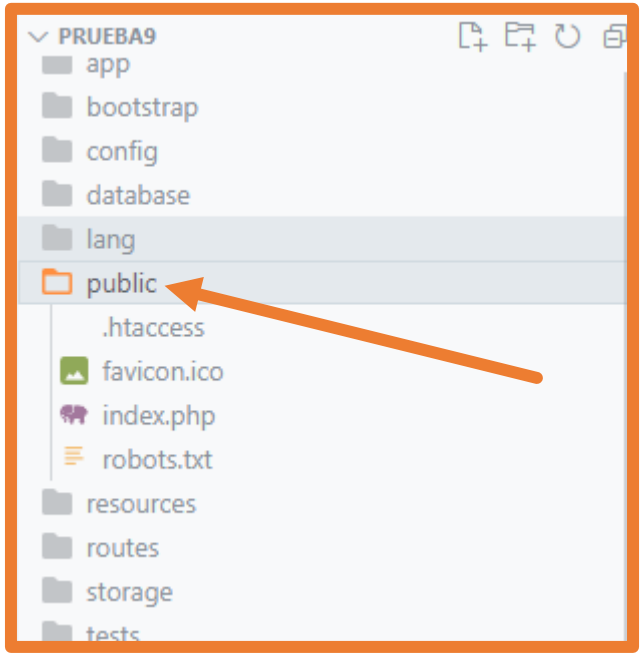
Desde Visual Studio Code se deberá abrir el directorio recién creado:



9.- Estructura de un proyecto Laravel

Como ya se estudió generalmente los frameworks usan el patrón Modelo-Vista-Controlador y Laravel no es la excepción.

De todos los archivos y directorios de un proyecto Laravel, los únicos que serán "visibles" para el navegador web (cliente) serán los incluidos en la carpeta **public**.



Así, el directorio raíz del mismo será la carpeta **public**, y en ella se deberán añadir las imágenes, los estilos css y los scripts JavaScript.

El resto de funcionalidad se añadirá en otras partes del proyecto como se estudiará durante la unidad.

9.- Estructura de un proyecto Laravel

- **app**: contiene código fuente de la aplicación.
  - Console: permite definir comandos propios.
  - Exceptions: permite definir excepciones propias.
  - **Http**: contiene los **controladores** y el middleware.
  - Providers: contiene los proveedores por defecto y los propios.
  - Se pueden crear carpetas adicionales por ejemplo Events o carpetas para almacenar el modelo o clases.

- Bootstrap: contiene el archivo **app.php** el cual es el inicio de la aplicación. También contiene la carpeta **cache** donde se almacenan los archivos ya cargados para acelerar las peticiones.

  Bootstrap → empujar/arranque. No confundir con CSS Bootstrap.

9.- Estructura de un proyecto Laravel

- **app**: contiene código fuente de la aplicación.
  - Console: permite definir comandos propios.
  - Exceptions: permite definir excepciones propias.
  - **Http**: contiene los **controladores** y el middleware.
  - **Models**: contiene los **modelos** para trabajar con la base de datos.
  - Providers: contiene los proveedores por defecto y los propios.
  - Se pueden crear carpetas adicionales por ejemplo Events o carpetas para almacenar clases propias.

- Bootstrap: contiene el archivo **app.php** el cual es el inicio de la aplicación. También contiene la carpeta **cache** donde se almacenan los archivos ya cargados para acelerar las peticiones.

  Bootstrap → empujar/arranque. No confundir con CSS Bootstrap.

9.- Estructura de un proyecto Laravel

- **config**: contiene los archivos de **configuración** de la aplicación, de ahí se obtienen las variables del entorno. Aunque es recomendable configurar las variables del entorno desde el archivo **.env** ubicado en la raíz.

- **database**: almacena los elementos de gestión de la **base de datos** como migraciones, seeders…

- **public**: Contenido **"visible"** de la aplicación. Contiene el archivo **index.php** punto de entrada de la aplicación. En esta carpeta deberá colocarse el contenido estático como CSS, JavaScript, imágenes…

- **resources**: contiene las **vistas**. También se pueden almacenar los archivos CSS no compilados (SASS) y JavaScript (sin minimizar). También es el lugar donde incluir los archivos de traducción si la aplicación es multidioma.

9.- Estructura de un proyecto Laravel

- **routes**: almacena las rutas de la aplicación, tanto el contenido normal en **web.php** como el contenido de los servicios web **api.php**.

- **storage**: contiene las vistas compiladas y otros archivos generados por Laravel como los logs o las sesiones.

- **test**: para almacenar los test/pruebas a realizar sobre la aplicación.

- **vendor**: se almacenan las dependencias/librerías adicionales.

9.- Estructura de un proyecto Laravel

Por defecto hay directorios y archivos que se omiten para el control de versiones (archivo .gitignore) como el archivo .env o el directorio vendor entre otros.

Esto es así para no compartir contenido sensible o contenido que se genera al lanzar la aplicación.

10.- Probando un proyecto Laravel

La forma más rápida de probar un proyecto Laravel es mediante el comando artisan:

Estando en el directorio del proyecto Laravel se debe ejecutar:

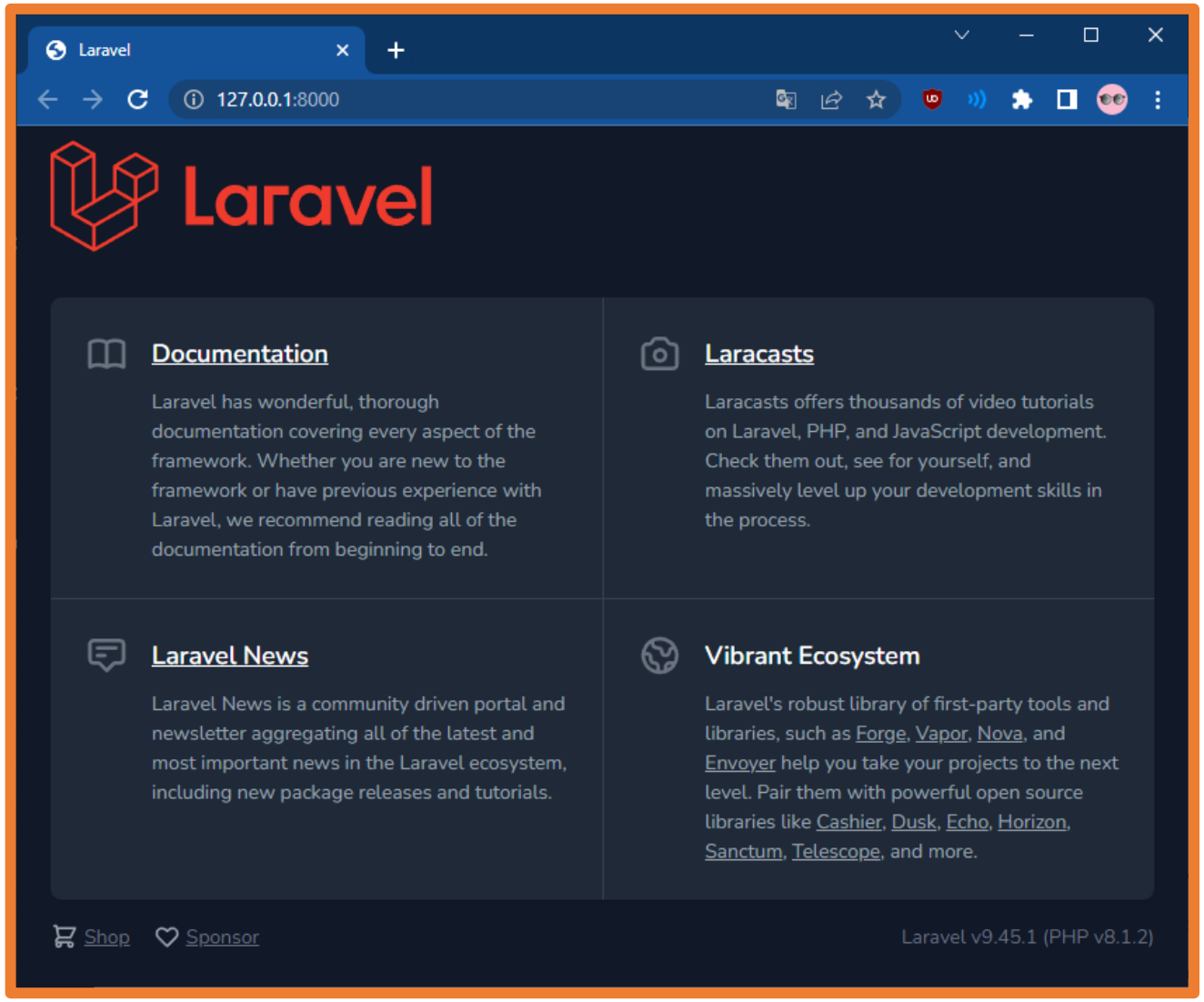
```
# php artisan serve
```

Mediante este método se dispone de un servidor web temporal.

10.- Probando un proyecto Laravel

```
# php artisan serve
```

```
PS C:\xampp\htdocs\Proyectos Laravel\prueba9> php artisan serve

  INFO  Server running on [http://127.0.0.1:8000].

  Press Ctrl+C to stop the server
```

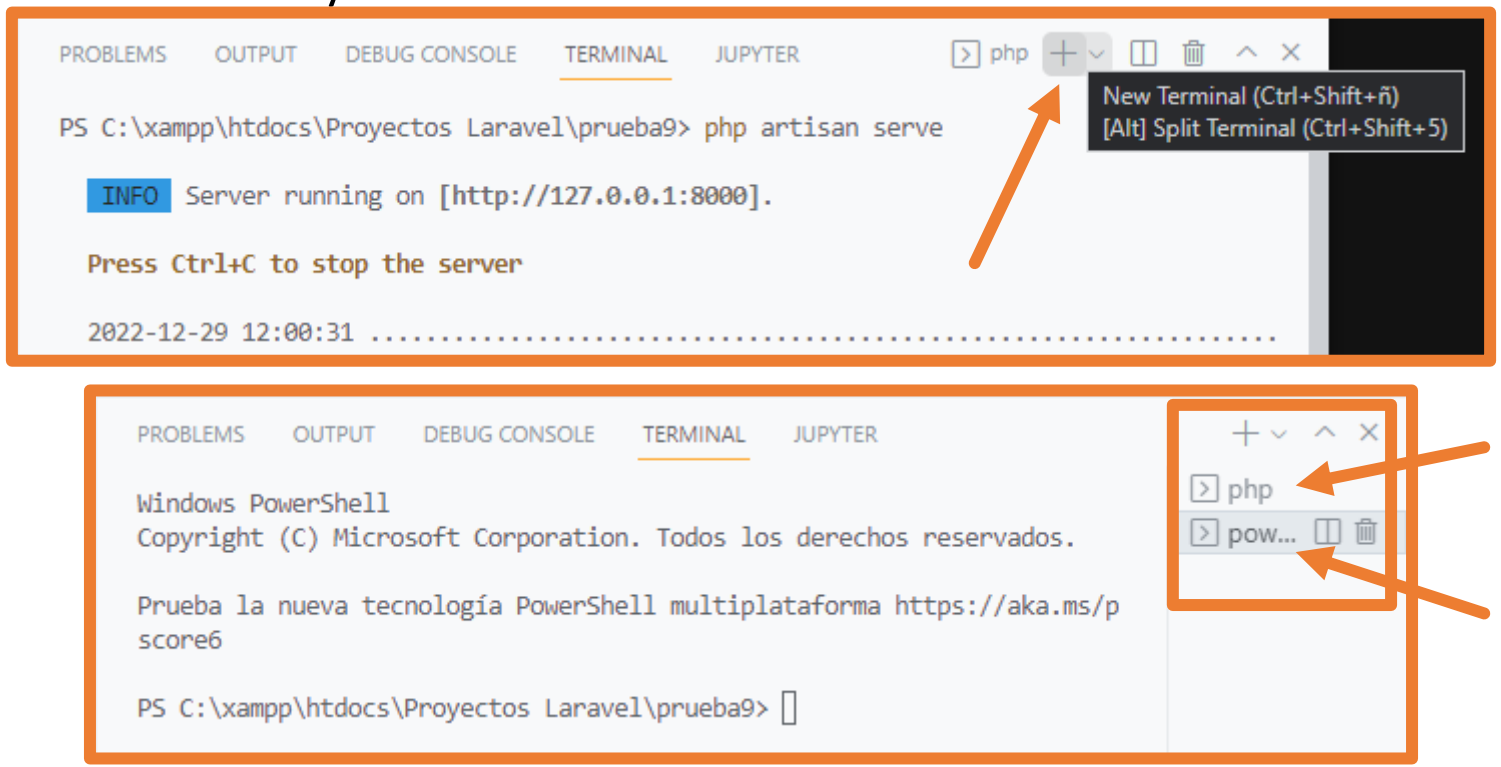


10.- Probando un proyecto Laravel

Cuando se desarrolla un proyecto Laravel se utiliza constantemente la línea de comandos, así que si se usa este método se recomienda tener dos terminales abiertas:

- Una para la ejecución del servidor.
- Una para ejecutar los comandos para crear componentes.

Usando Visual Studio Code es muy sencillo:



10.- Probando un proyecto Laravel

La otra forma de probar el proyecto es crear un host virtual.

Para que el proyecto funcione el directorio repositorio del host virtual debe ser el directorio **public** del proyecto Laravel.

```
<VirtualHost *:80>
   DocumentRoot "C:/blog/public"
   ServerName localhost.blog
   ErrorLog "logs/blog-error.log"
   CustomLog "logs/blog-access.log" common
</VirtualHost>
```

10.- Probando un proyecto Laravel

En Linux/MAC, además, se deberán dar permisos a las carpetas:

- storage
- bootstrap/cache.

Con la línea de comandos desde el directorio del proyecto:

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

Si posteriormente aparece un error al intentar crear logs:

```
# sudo chmod -R 777 storage/logs
```

11.- Importando un proyecto Laravel

Al guardar el proyecto en el repositorio del control de versiones hay archivos/directorios que se ignoran, por ello **después de clonar por primera vez** el repositorio desde GitHub a un ordenador se deberán seguir unos pasos para que se generen dichos archivos y que todo funcione.

Desde el directorio del proyecto:

- Configurar el archivo .env
- Generar una clave de seguridad nueva: php artisan key:generate
- Cargar dependencias PHP: composer install
- Cargar dependencias JavaScript: npm install

11.- Importando un proyecto Laravel

Si se quiere enviar un proyecto Laravel de un ordenador a otro sin usar un repositorio de control de versiones, para aligerar el proyecto se deben eliminar si existen las carpetas **node_modules** y **vendor**.

Posteriormente, en el equipo donde se copie el proyecto se deben seguir los mismos pasos que cuando se clona por primera vez el proyecto desde un repositorio de control de versiones.

12.- Actualizar la versión de Laravel del proyecto

En el archivo **composer.json** se pueden encontrar todas las dependencias que composer deberá instalar.

La línea marcada indica que se debe instalar una versión de Laravel 9.19 o superior si estuviera disponible.

Para actualizar las dependencias se debe ejecutar el comando:

```
# composer update
```

Práctica

Actividad 1:

Creando y probando un proyecto Laravel