

ÍNDICE DE CONTENIDOS



LARAVEL: UN FRAMEWORK DE PHP



Enfoca tu esfuerzo en **tu programa**,
laravel te lo ubica en **tu aplicación** .
*Programa tus programas, no tus
herramientas*

CONCEPTOS GENERALES

¿Qué es un framework

Por qué usar un framework



- ***Laravel ofrece un entorno de desarrollo***
- ***Altamente funcional,***
- ***Interfaces de línea de comandos intuitivas y expresivas. (Artisan)***
- ***Además, Laravel utiliza el mapeo objeto-relacional (ORM) para simplificar el acceso y la manipulación de datos.***

Framework o Librería

Imagen publicada en facebook
<https://www.facebook.com/EDteamLat/photos/a.634364769952437/4908906269164911/?type=3>



INTRODUCCIÓN

- > Lenguaje de programación: [PHP](#)
- > Fecha del lanzamiento inicial: junio de 2011
- > Desarrollador: [Taylor Otwell](#)
- > Licencia: [Licencia MIT](#)
- > Tipo de programa: Framework
- > Versión actual **10**
- > Lanzamiento de próxima versión **Primer trimestre 2024**
- > Versión de php **8.2**

Laravel es un **framework** de código abierto.

Usa tecnología **php** para desarrollar aplicaciones de forma **elegante y simple.**

Tiene una curva de aprendizaje muy suave, lo que permite no necesitar demasiado tiempo para desarrollar aplicaciones.

INTRODUCCIÓN

Por qué usar un framework

- Nos va a marcar **una organización de la estructura completa de nuestro proyecto**
- Nos ofrece **librerías y métodos** para realizar gran cantidad de *trabajo típico de desarrollos* (acceso a base de datos, gestión de cookies, autenticación, ...)
- Estos conceptos **tienen un tiempo de aprendizaje** que hay que dedicar con un poco de paciencia las ventajas son muy significativas

INTRODUCCIÓN

Qué necesito para empezar con laravel

- Sobre todo **ganas de aprender y de desarrollar una aplicación**
- Tener **la confianza** que puedes llegar a hacer productos profesionales
- Laravel es relativamente fácil de aprender.
 - *Debemos tener una comprensión general de php*
 - *Tener conceptos de programación orientada a objetos (POO).*
 - *Saber HTML y CSS.*
 - *Gestión de bases de datos*

02 INSTALACIÓN *TODO*

¿Qué hay que instalar ?

Laravel **no** es ni un **EDI**, ni un **lenguaje de programación**, es un **framework**.

Necesitamos un programa que nos permita crear un proyecto nuevo con la estructura y todas las utilidades que ofrece el framework

Podemos instalar un **instalador de laravel** con composer:

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

Una vez instalado, disponemos de un ejecutable por línea de comandos llamado **laravel**.

Este programa es un **instalador de proyectos de laravel** que es una herramienta en línea de comandos que nos va a permitir crear un proyecto base de laravel. Para ello escribimos

```
laravel new nombre_proyecto
```

Puede ser que dé algún problema y no reconozca la orden laravel. en este caso deberíamos de actualizar el **path** del sistema incluyendo el directorio donde haya instalado laravel

02 INSTALACIÓN *TODO*

También podríamos crear un nuevo proyecto nuevo de laravel directamente con composer

```
composer create-project laravel/laravel nombre_proyecto
```

Usaremos habitualmente el instalador de laravel **laravel**, por considerarlo más intuitivo.

Las opciones disponibles las vemos escribiendo **laravel**:

Actualmente (21/12/2023), la versión del instalador es la **5.2**

```
C:\Users\Profesor\laravel>laravel
Laravel Installer 4.5.0

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display help for the given command.
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
  --ansi|--no-ansi         Force (or disable --no-ansi) ANSI o
  -n, --no-interaction     Do not ask any interactive question
  -v|vv|vvv, --verbose     Increase the verbosity of messages:
  ug

Available commands:
  completion  Dump the shell completion script
  help        Display help for a command
  list        List commands
  new         Create a new Laravel application

C:\Users\Profesor\laravel>
```

02 Pasos iniciales

Creando un proyecto con laravel

Mejor creamos una carpeta llamada laravel y ahí crearemos nuestros proyectos

```
laravel new estudios
```

El proceso de instalación realiza una serie de preguntas, a las que vamos a responder por defecto

Nos habrá creado una carpeta en el directorio dónde estuviéramos con el nuevo proyecto llamado estudios

Abrimos la carpeta con **phpstorm** y podremos analizar la estructura de carpetas

02 Pasos iniciales

Creando un proyecto con phpstorm

Mejor creamos una carpeta llamada laravel y ahí crearemos nuestros proyectos

```
laravel new net
```

El proceso de instalación realiza una serie de preguntas, a las que vamos a

```
alumno@alumno-20:~/laravel$ laravel new net
```

r

The image shows the Laravel logo, which consists of the word "Laravel" in a stylized, outlined font. The letters are red and have a dashed outline.

Would you like to install a starter kit? _____

- > ☒ No starter kit
- Laravel Breeze
- Laravel Jetstream

Which testing framework do you prefer? _____

- > ☒ PHPUnit
- Pest

Would you like to initialize a Git repository? _____

- Yes / ☒ No

02 Pasos iniciales

Estructura de carpetas

- Cada carpeta tiene una utilidad donde se destinarán ficheros determinados
- Cuando se empieza a utilizar, es cuando realmente se entienden los directorios y ficheros.
- Vamos a ver por encima su intención
- Se pueden crear carpetas si

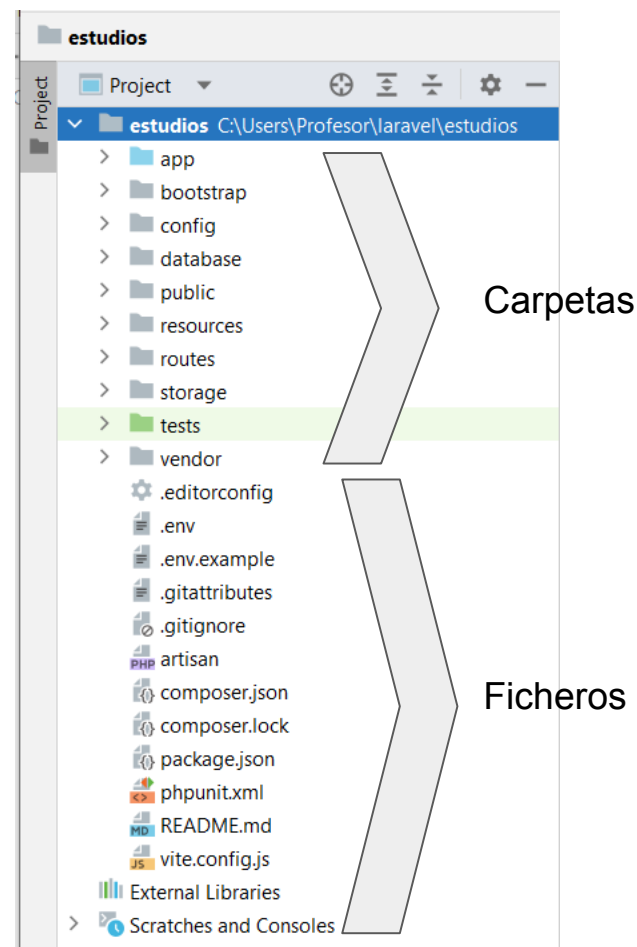


```
Total de archivos en la lista:
7888 archivos      50.753.646 bytes
3791 dirs  137.449.369.600 bytes libres
/Users/Profesor/laravel/estudios>
```

- Si borro la carpeta **vendor**



```
Total de archivos en la lista:
93 archivos      806.697 bytes
113 dirs  137.229.082.624 bytes libres
C:\Users\Profesor\laravel\estudios>
```



Estructura de directorios :

App

- ✓ app
 - > Console
 - > Exceptions
 - > Http
 - > Models
 - > Providers

Código fuente de nuestra aplicación

Es el ***directorio de la aplicación***

En él ubicamos las clases que ofrece la funcionalidad a la aplicación, es decir *la parte principal de programación en php*

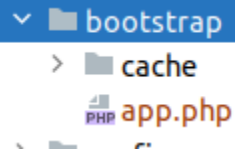
Dentro de él, el subdirectorio

Http contiene elementos importantes (Controladores (**Controllers**) y Filtros o **Middleware**).

Models o modelos que son las clases que interactúan con las tablas a través de un ORM

- ✓ app
 - > Console
 - > Exceptions
 - ✓ Http
 - ✓ Controllers
 - Controller.php
 - > Middleware
 - Kernel.php
 - ✓ Models
 - User.php
 - > Providers

Estructura de directorios : Bootstrap

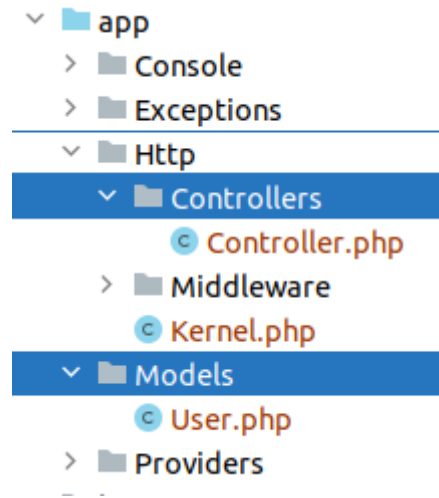


Arranca el framework en nuestra app

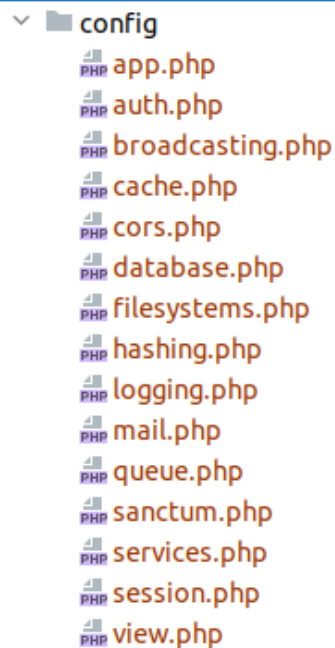
Es **directorio de inicio de nuestra aplicación**,

Nada tiene que ver con el framework de css

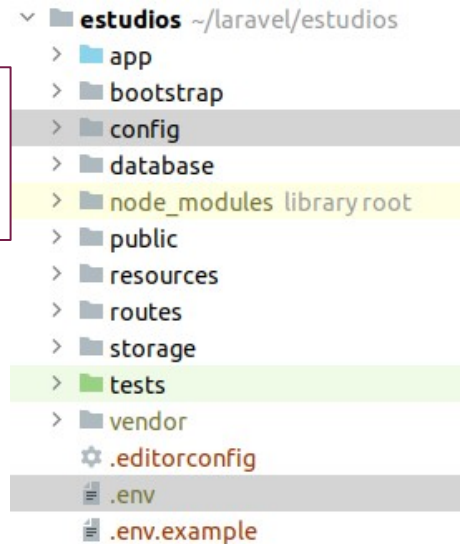
En **cache**, laravel cachea páginas para mejorar rendimiento, se almacenan página ya procesadas por laravel y así no tendrá que volver a procesarlas



02 Estructura de directorios :



Configuración de nuestra app
**Mejor modificar las directivas el
fichero .env**



Configuraremos los parámetros de nuestra aplicación (lang, name,
...)

Fichero .env para establecer directamente valores a los
parámetros

helper **env** para recoger los valores de los parámetros del
fichero **.env**

Estructura de directorios : Database

Acciones directas con la base de datos

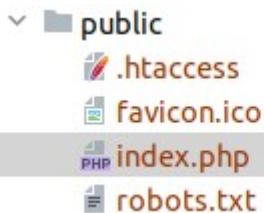
- ▼ database
 - ▼ factories
 - Ⓢ UserFactory.php
 - ▼ migrations
 - 📄 2014_10_12_000000_create_users_ta
 - 📄 2014_10_12_100000_create_passwor
 - 📄 2019_08_19_000000_create_failed_jc
 - 📄 2019_12_14_000001_create_persona
 - ▼ seeders
 - Ⓢ DatabaseSeeder.php
 - 📄 .gitignore

Migrations : para crear/modificar/borrar tablas en la base de datos

Factory: factoría o fábricas para producir tuplas, valores para las tablas

Seeders: semillero con la intención de insertar o poblar las tablas

Estructura de directorios : Public



Punto de entrada a todas las solicitudes a nuestra aplicación a través del index.php

Único directorio accesible públicamente

Todos los ficheros y carpetas a los que queramos que **el cliente tenga acceso**, deberán de estar aquí (css, vídeos, imágenes, javascript)

Estructura de directorios : Resources

La parte front de nuestra aplicación

▼ resources

> css

> js

▼ views

 welcome.blade.php

Vistas de la aplicación (html)

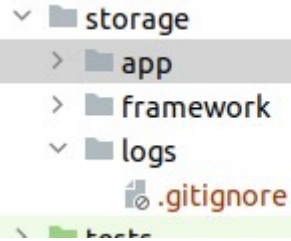
Ficheros no compilados (transpilados) de css y js

Estos ficheros luego se compilarán y pasarán a la carpeta public

*Para este proceso usaremos **vite** (versiones anteriores **webpack**)*

Estructura de directorios : Storage

Archivos generados por nuestra aplicación



```

  ▾ storage
    > app
    > framework
    ▾ logs
      .gitignore

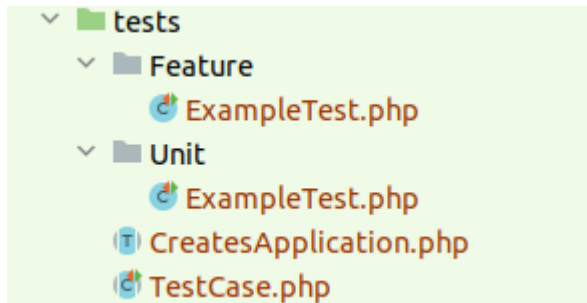
```

El directorio app, suele haber un enlace simbólico a la carpeta public, por ejemplo para recoger ficheros que puedan subir

framework ficheros que genera nuestro framework

logs para guardar los errores que se generen en nuestra aplicación

Estructura de directorios : test



Ficheros para realizar los test de nuestra aplicación

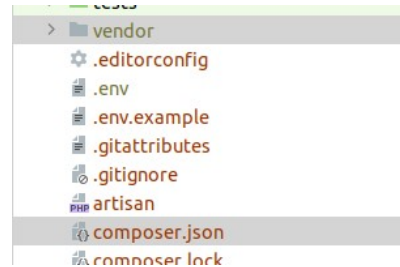
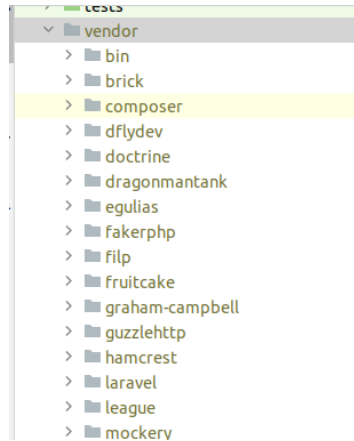
Ya viene con un ejemplo

Tenemos dos tipos de test:

- Feature
 - a. Test de características dónde se testean diferentes componentes interactuando entre ellos
- Unit
 - a. Que testean métodos concretos del código

Estructura de directorios : vendor

Es la carpeta del framework.



Contiene el grueso en cuanto a número de ficheros y directorios

Esta carpeta la genera composer cuando orquestamos
No hay que subirla a git, no hay que escribir en ella.
Se genera a partir del fichero composer.json

Estructura de directorios : los ficheros

⚙️ .editorconfig

📄 .env

📄 .env.example

📄 .gitattributes

📄 .gitignore

📄 artisan

📄 composer.json

📄 composer.lock

📄 package.json

📄 package-lock.json

📄 phpunit.xml

📄 postcss.config.js

📄 README.md

📄 tailwind.config.js

📄 vite.config.js

Configuración con valores para directivas

los ficheros/directorios ignorados al subir el proyecto a git

paquetes a instalar en el back al orquestar

paquetes/aplicaciones a instalar en el front al hacer un **npm install**

fichero para transpilar los ficheros de resources

fichero de configuración de tailwind (hay que instalarlo)

Fichero de configuración para la transpilación

Artisan

La interfaz por línea de comandos

Artisan es una herramienta de consola, escrita en PHP, que viene con Laravel para realizar tareas cotidianas en tu aplicación de forma automatizada.

Para ejecutar la aplicación en desarrollo, no tenemos que usar apache

Empecemos por levantar el servidor para ver la aplicación ejecutándose

Muy importante estar en la carpeta del proyecto (da igual si es windows o linux)

```
php artisan serve
```

```
estudios git:(main) x php artisan serve
```

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

```
Press Ctrl+C to stop the server
```

Artisan

La interfaz por línea de comandos

Artisan es una herramienta de consola, escrita en PHP, que viene con Laravel para realizar tareas cotidianas en tu aplicación de forma automatizada.

Para ejecutar la aplicación en desarrollo, no tenemos que usar apache

Empecemos por levantar el servidor para ver la aplicación ejecutándose

Muy importante estar en la carpeta del proyecto (da igual si es windows o linux)

```
php artisan serve
```

```
estudios git:(main) x php artisan serve
```

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

```
Press Ctrl+C to stop the server
```

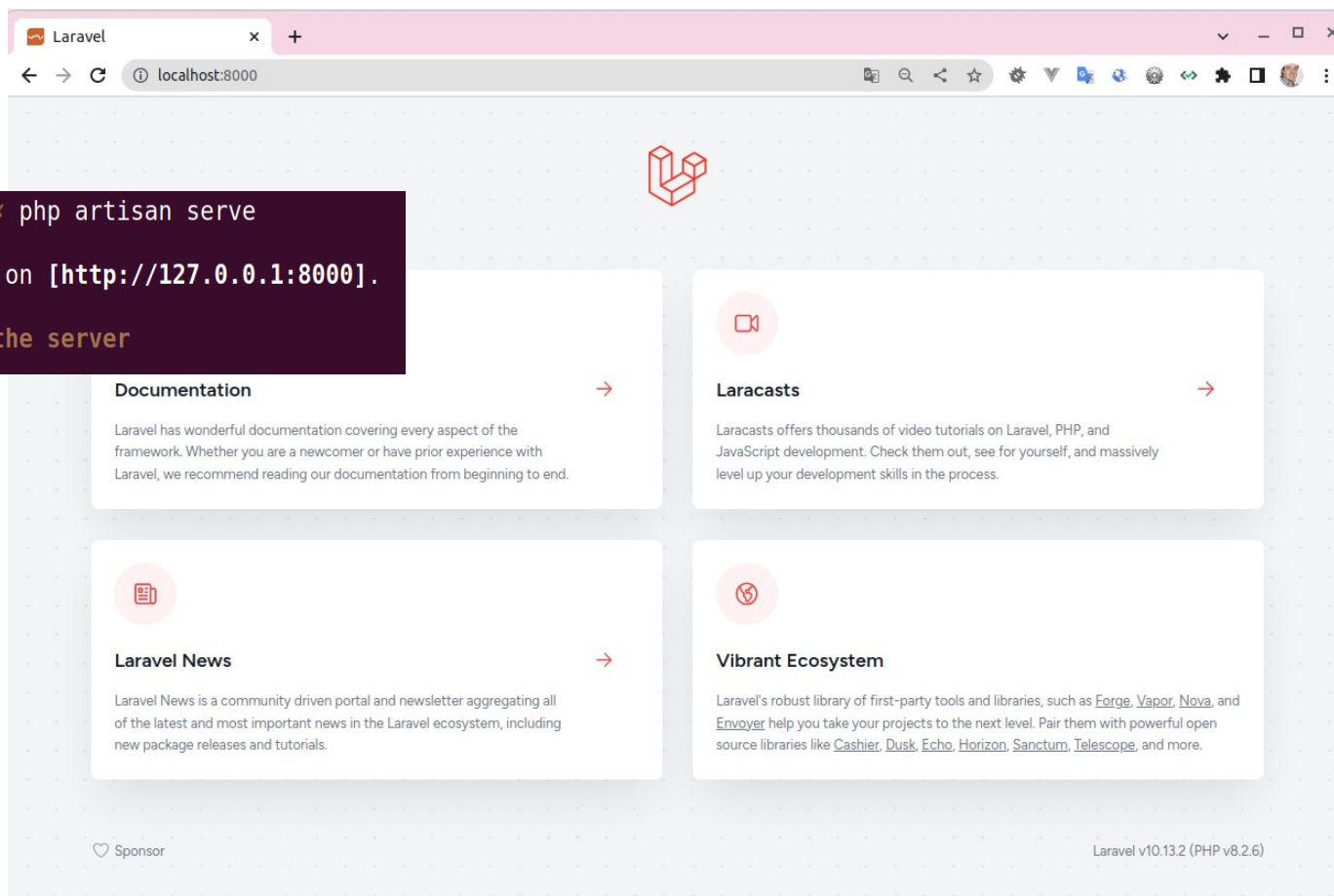

php

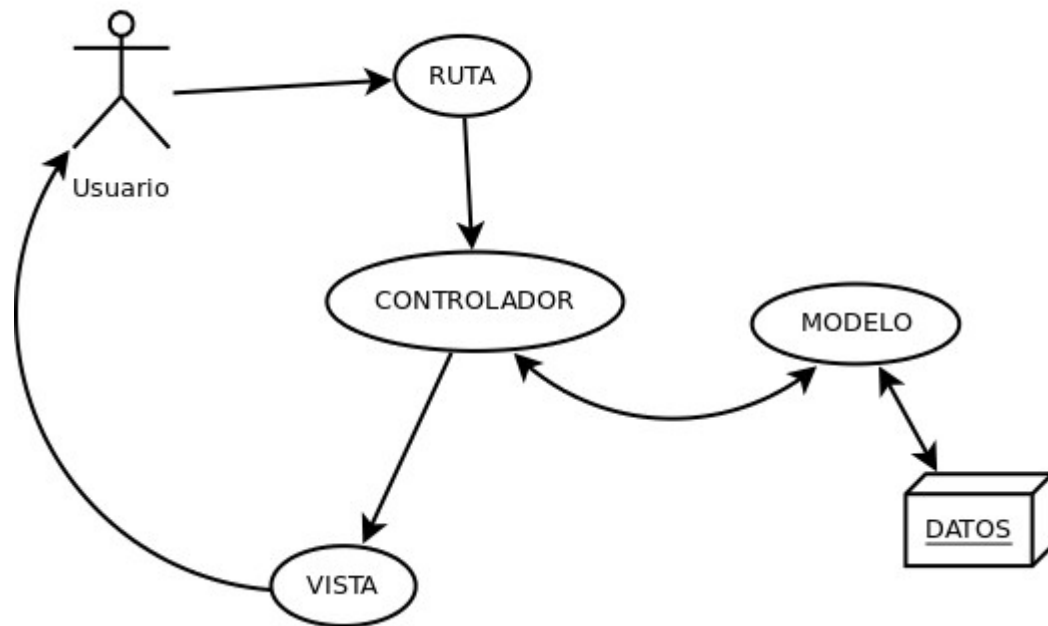
```
estudios git:(main) x php artisan serve
```

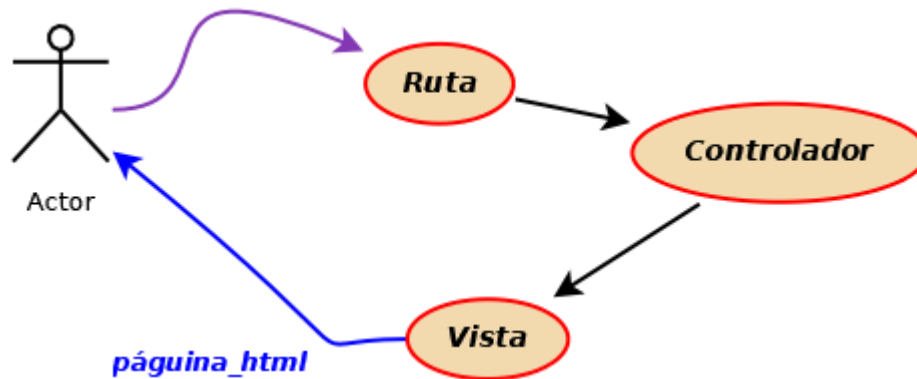
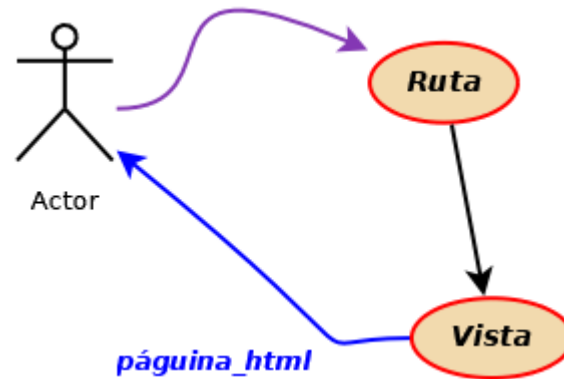
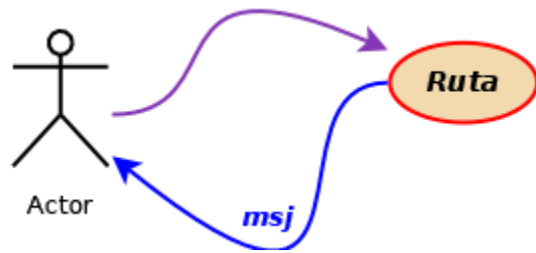
```
INFO Server running on [http://127.0.0.1:8000].
```

```
Press Ctrl+C to stop the server
```

Veremos más comandos de artisan según vayamos avanzando, se usa mucho





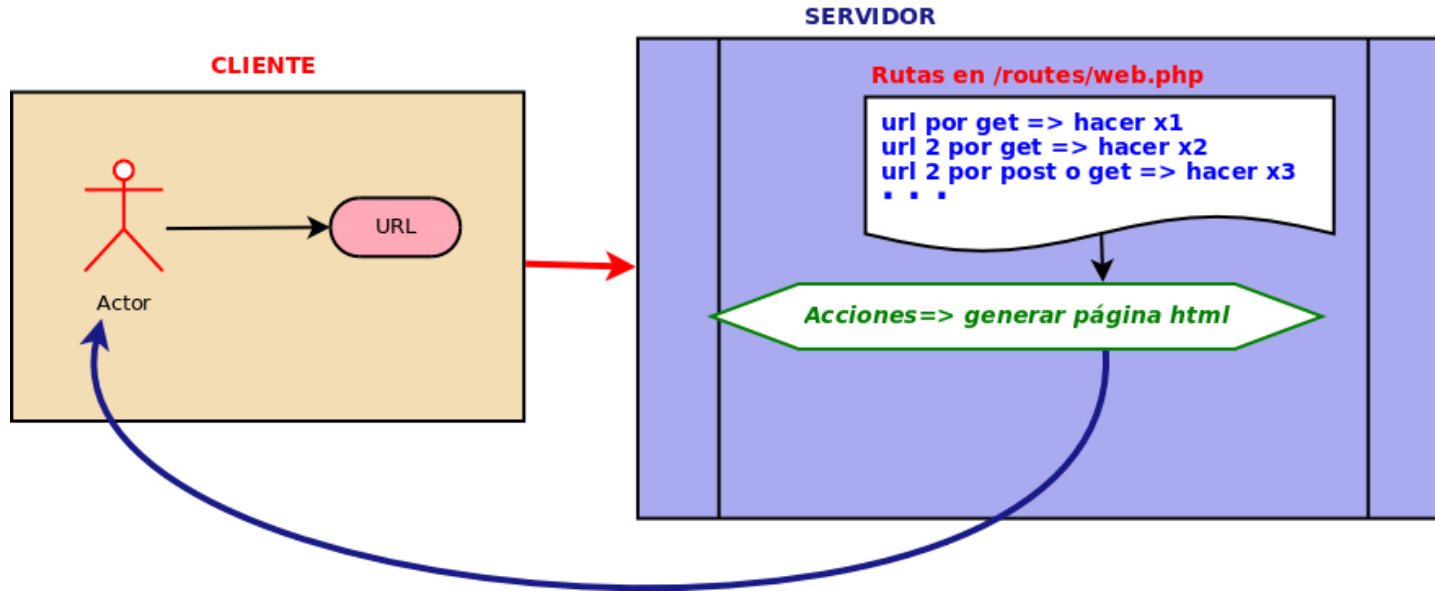


Las rutas

Solicitando páginas o recursos

Hasta ahora los recursos o páginas que se solicitaban eran páginas que existían en el servidor

Ahora los recursos disponibles van a ser entradas que establezcamos en un fichero de rutas



Las rutas

<https://laravel.com/10.x/dev/routing>

Fichero `routes/web.php`

Facade

Recurso que se ofrece

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

**Tipo de solicitud,
Verbo http**

Acción a realizar

- Devolver una vista
- Hacer algo
- Ejecutar un método de un controlador

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

```
//Cualquiera de los métodos get o post
Route::match(['get', 'post'], '/', function () { // });
//Cualquier método o verbo http
Route::any('foo', function () { // });
```

02 Las rutas practicando

Creamos 4 rutas que nos muestren 4 páginas



```
Route::get('about', function () {
    return view("about");
});
Route::get('alumnos', function () {
    return view("alumnos");
});
Route::get('contacta', function () {
    return view("contacta");
});
Route::get('proyectos', function () {
    return view("proyectos");
});
```

views

about.blade.php
alumnos.blade.php
contacta.blade.php
proyectos.blade.php
welcome.blade.php

```
<!doctype html>
<html lang="en">
<head>
.....
<title>estudiantes</title>
<meta name="description" content="Listado
de estudiantes ">
</head>
<body>
<h1>Estoy en About</h1>
</body>
</html>
```

```
<nav>
<ul>
<li><a href="about">About</a></li>
<li><a href="contacta">Contacta</a></li>
<li><a href="alumnos">Alumnos</a></li>
<li><a href="/">Volver</a></li>
</ul>
</nav>
```

Las rutas método view

Si una ruta solo retorna una vista, mejor usar el método view de la facade Route

```
Route::view("about", "about");
```

```
Route::view("alumnos", "alumnos");
```

```
Route::view("contacta", "contacta");
```

```
Route::view("proyectos", "proyectos");
```

Para ver con **artisan** las *rutas creadas*

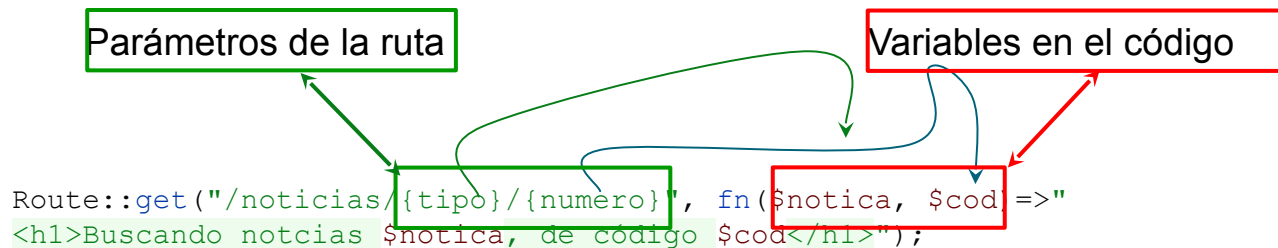
php artisan route:list

```
estudios git:(main) x php artisan route:list

GET|HEAD  / .....
POST      _ignition/execute-solution ignition
GET|HEAD  _ignition/health-check ..... ignit
POST      _ignition/update-config ... ignitic
GET|HEAD  about .....
GET|HEAD  alumnos .....
GET|HEAD  api/user .....
GET|HEAD  contacta .....
GET|HEAD  proyectos .....
GET|HEAD  sanctum/csrf-cookie ..... S
```

02 Las rutas parametrizadas

Podemos hacer que parte de la ruta se convierta en un parámetro



Muy útil para consultas en bd y pasar códigos

02 Las rutas parametrizadas

Podemos restringir el valor de parámetros con expresiones regulares

```
Route::get("/noticias/{tipo}/{numero}",  
  fn($noticia, $cod)=>"  
  <h1>Buscando notcias $noticia, de código  
  $cod</h1>")  
->where("tipo", '[a-zA-Z]+')  
->where('numero', '[0-9]+');
```

Observa cómo se concatenan métodos sin ningún problema.
Esta forma de programar corresponde **al patrón de diseño Fluent**
hace un código más legible y fluido (Interfaz Fluída o Fluent Interface).

02 Las rutas parametrizadas

1. `where`: Permite aplicar una expresión regular para validar un parámetro de la ruta. Ejemplo: `->where('parametro', 'expresion-regular')`.
2. `whereAlpha`: Valida que el parámetro de la ruta contenga solo caracteres alfabéticos. Ejemplo: `->whereAlpha('parametro')`.
3. `whereAlphaNum`: Valida que el parámetro de la ruta contenga solo caracteres alfanuméricos. Ejemplo: `->whereAlphaNum('parametro')`.
4. `whereNumeric`: Valida que el parámetro de la ruta contenga solo caracteres numéricos. Ejemplo: `->whereNum('parametro')`.
5. `whereUuid`: Valida que el parámetro de la ruta sea un UUID válido. Ejemplo: `->whereUuid('parametro')`.
6. `whereRegex`: Permite aplicar una expresión regular personalizada para validar un parámetro de la ruta. Ejemplo: `->whereRegex('parametro', 'expresion-regular')`.

Las rutas nombre de rutas

Es interesante y puede resultar útil poner nombre a las rutas.
De esta forma si cambiara la url o referencia, no afectaría a su invocación

html

```
<li>  
  <a href="{{route('contacto')}}">Concata con nosotros</a>  
</li>
```

web.php

```
Route::view("contact", "contact")->name("contacto");
```

```
Route::view("contactanos", "contact")->name("contacto");
```

Las rutas que no existen fallback

Puedes establecer un método que se ejecute cuando no existe la ruta solicitada

```
Route::fallback(function() {  
    return "<h1>La ruta no existe</h1>";  
});
```

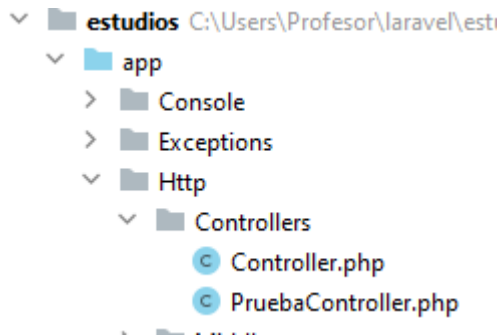
También puedes obtener el nombre de la ruta

```
Route::fallback(function() {  
    $ruta = request()->getRequestUri();  
    return "<h1>La ruta $ruta no existe</h1>";  
});
```

02 Laravel controlador

1.- Creamos un controlador con php artisan

```
php artisan make:controller PruebaController
```



2.- Escribimos un método en el controlador

```
class PruebaController extends Controller
{
    //
    public function index() {
        return view('welcome', ['mensaje'=>"hola"]);
    }
}
```

3.- Invocamos al método desde la ruta **web.php**

```
Route::get("prueba",[PruebaController::class,'index']);
```

Las vistas

Ciclo de vida de una aplicación de laravel

Laravel utiliza un gestor de plantillas llamado Blade
Una plantilla en Blade es un fichero html, pero que puede tener alguna directiva propia del gestor de plantillas

Laravel transpila (transforma) los ficheros .blade.php a php

