[Laravel/Lumen, conocimientos basicos y generales 2](#_Toc1)

[Como iniciar un proyecto 2](#_Toc2)

[Laravel Herd 2](#_Toc3)

[Modelos para consultar BD 12](#_Toc4)

[¿Qué son las migraciones? 12](#_Toc5)

[Como enviar datos a la vista 16](#_Toc6)

[CRUD Laravel 18](#_Toc7)

[Service Class 22](#_Toc8)

[Conocimientos generales 29](#_Toc9)

[Establecer una bd y las credenciales en .env 29](#_Toc10)

[Establecer archivos de migracion y sus modelos 30](#_Toc11)

[Migraciones 30](#_Toc12)

[Creacion de tablas 31](#_Toc13)

[Modificadores de columnas 32](#_Toc14)

# Laravel/Lumen, conocimientos basicos y generales

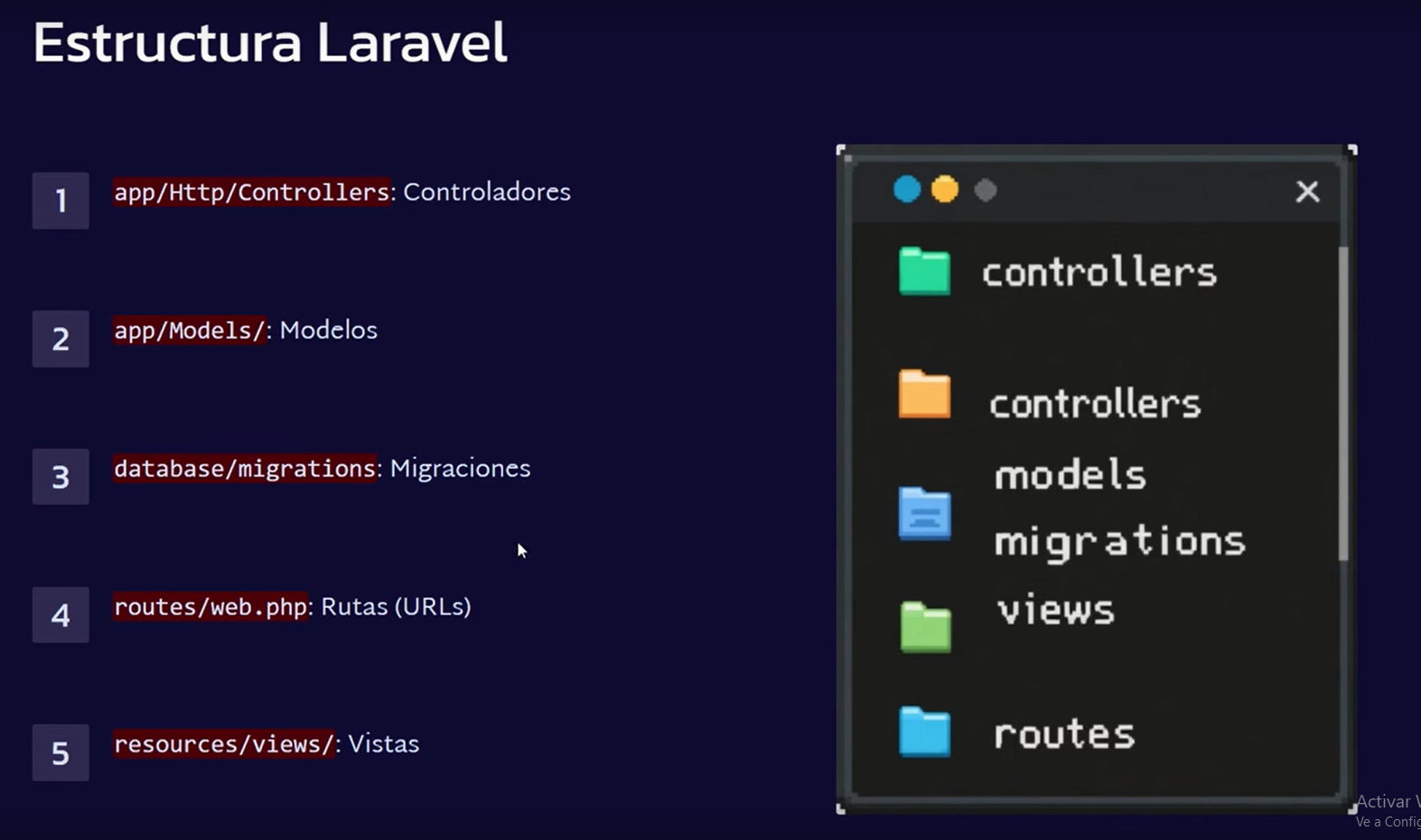
## Como iniciar un proyecto

|  |  |
| --- | --- |
| Aprende Laravel en 45 minutos (Versión 12.x) | 🛠️Workshop | https://youtu.be/PaYwBzH\_WSI?si=oDbONMitR29-n0by |

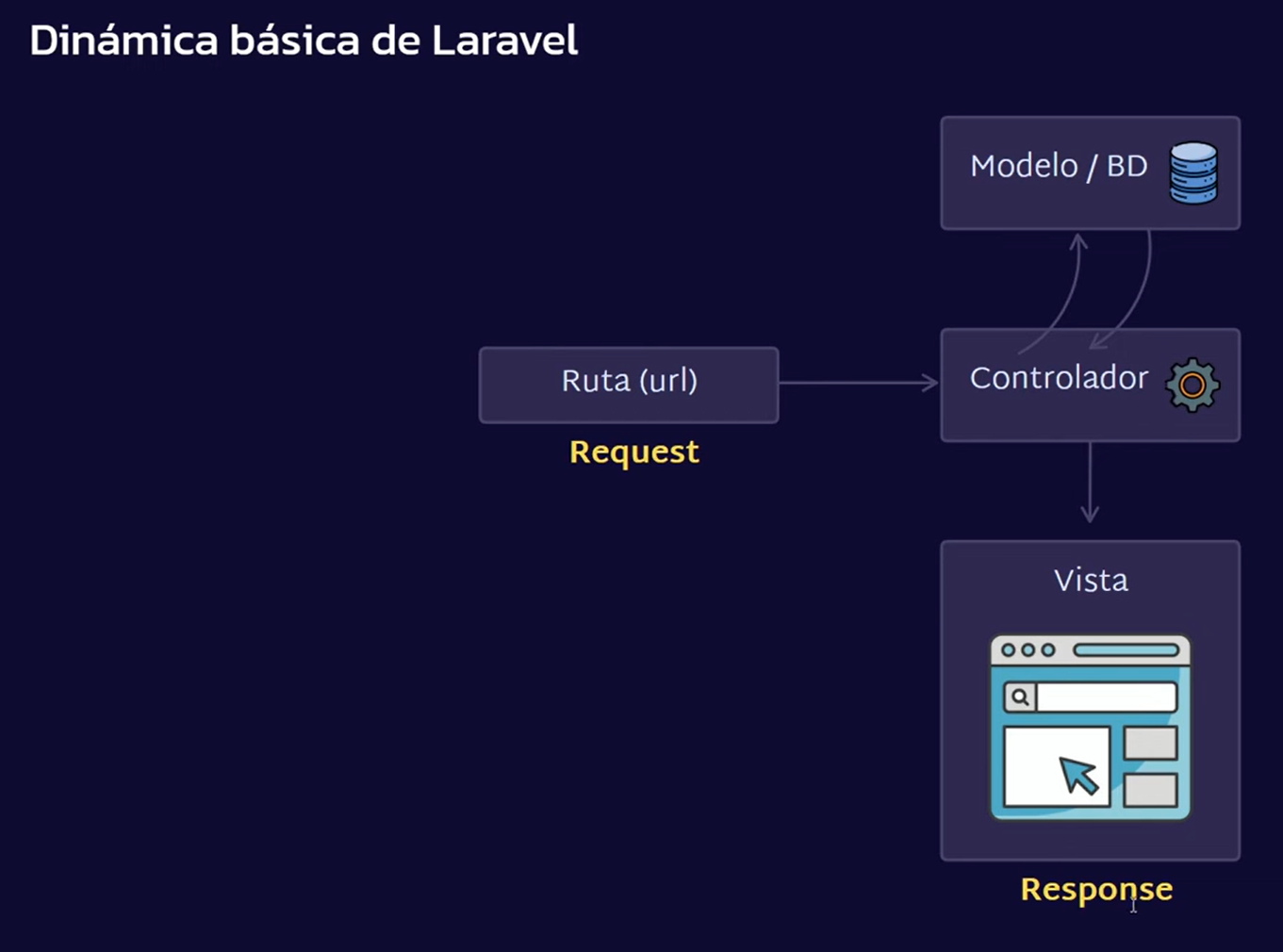
## Laravel Herd

<https://youtu.be/PaYwBzH_WSI?si=W13SXfLTe8mtmzYU&t=234>

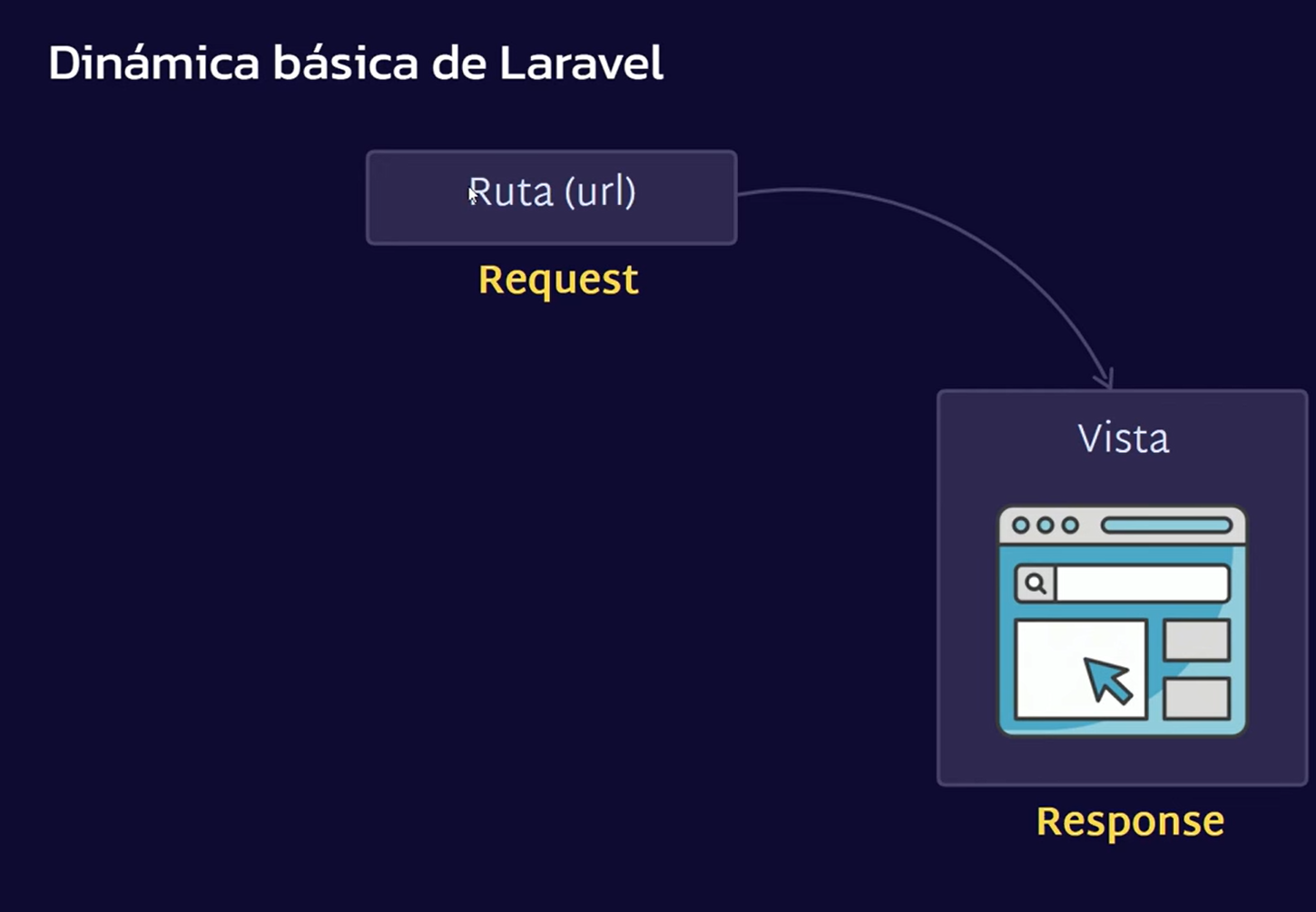
Estructura Laravel  
https://youtu.be/PaYwBzH\_WSI?si=G-4f4lxmv9vWNlQm



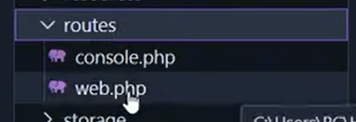
La dinamica basica de Laravel  
<https://youtu.be/PaYwBzH_WSI?si=K4KcpqJHNYTrMm0I&t=408>



La dinámica de Laravel puede reducirse a:



Las rutas se guardan en:



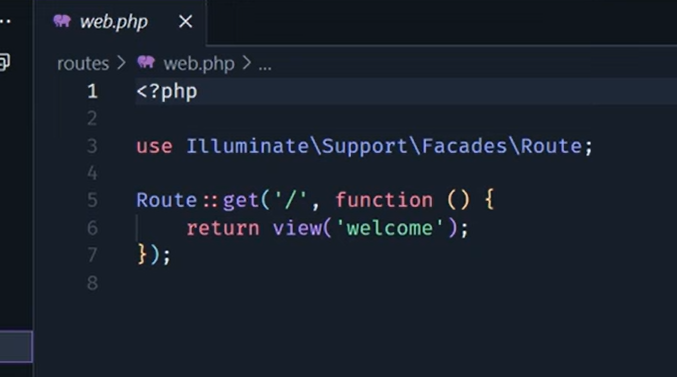
Ejemplo:

Ruta que hace una petición Get, o la que queramos.

Que se dirige a la raíz del proyecto “/”.

También hay un Callback “function” o tambien llamada closure. Que retorna una vista.

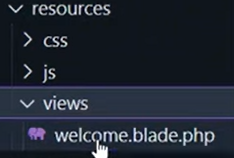
Hay una ruta y una respuesta.



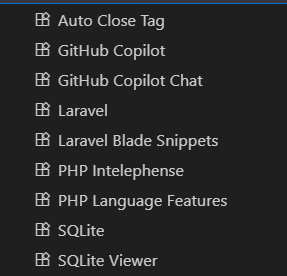
Las vistas:

Se encuentran en resources/views.

Tienen la extensión blade.

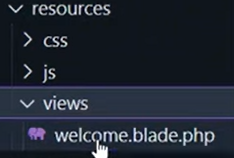


Lista de extensiones usadas para desarrollar en Laravel:

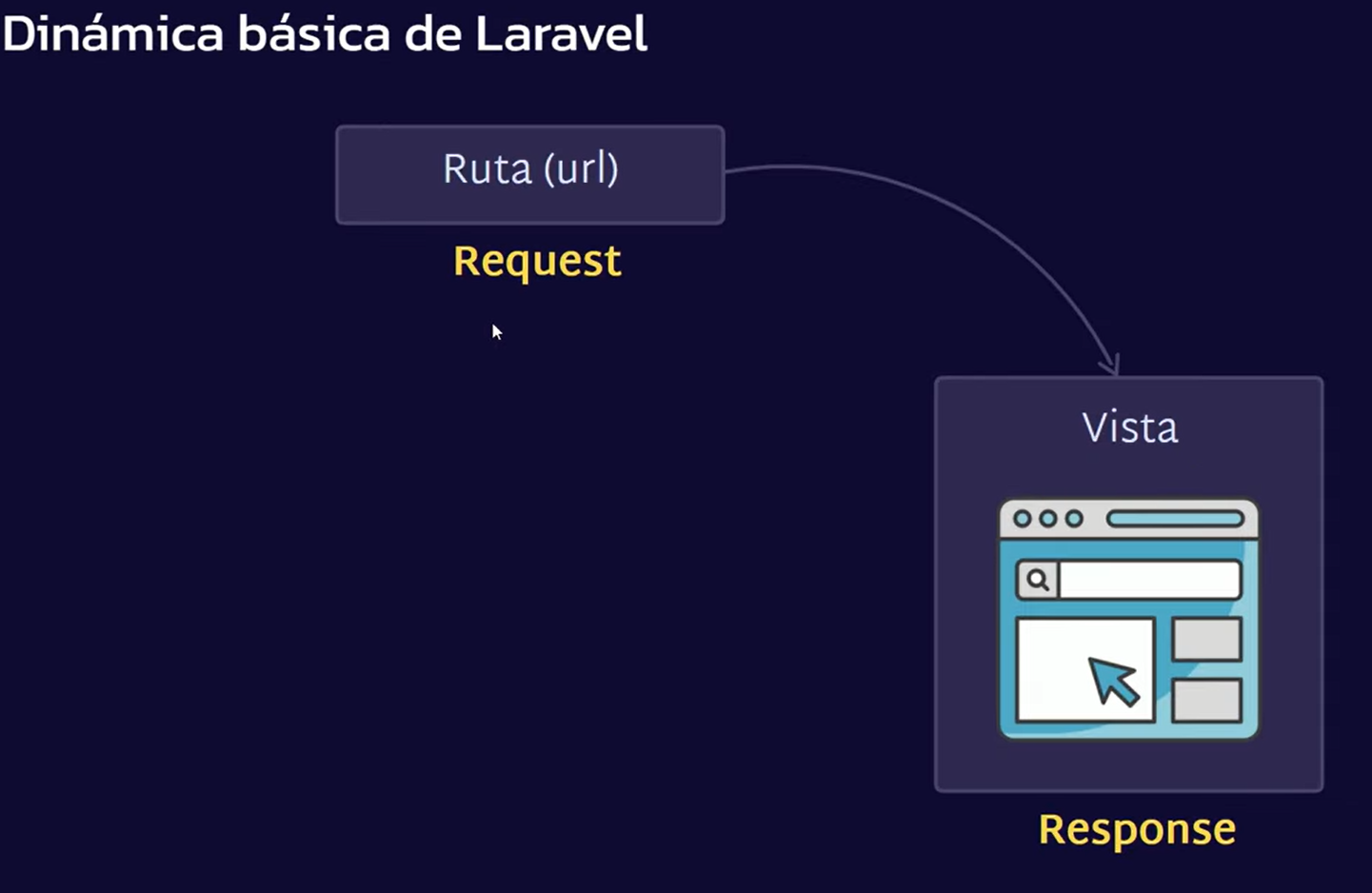


Blade es el motor de plantillas que simplifica el código. Porque se nos facilitan directivas Blades.

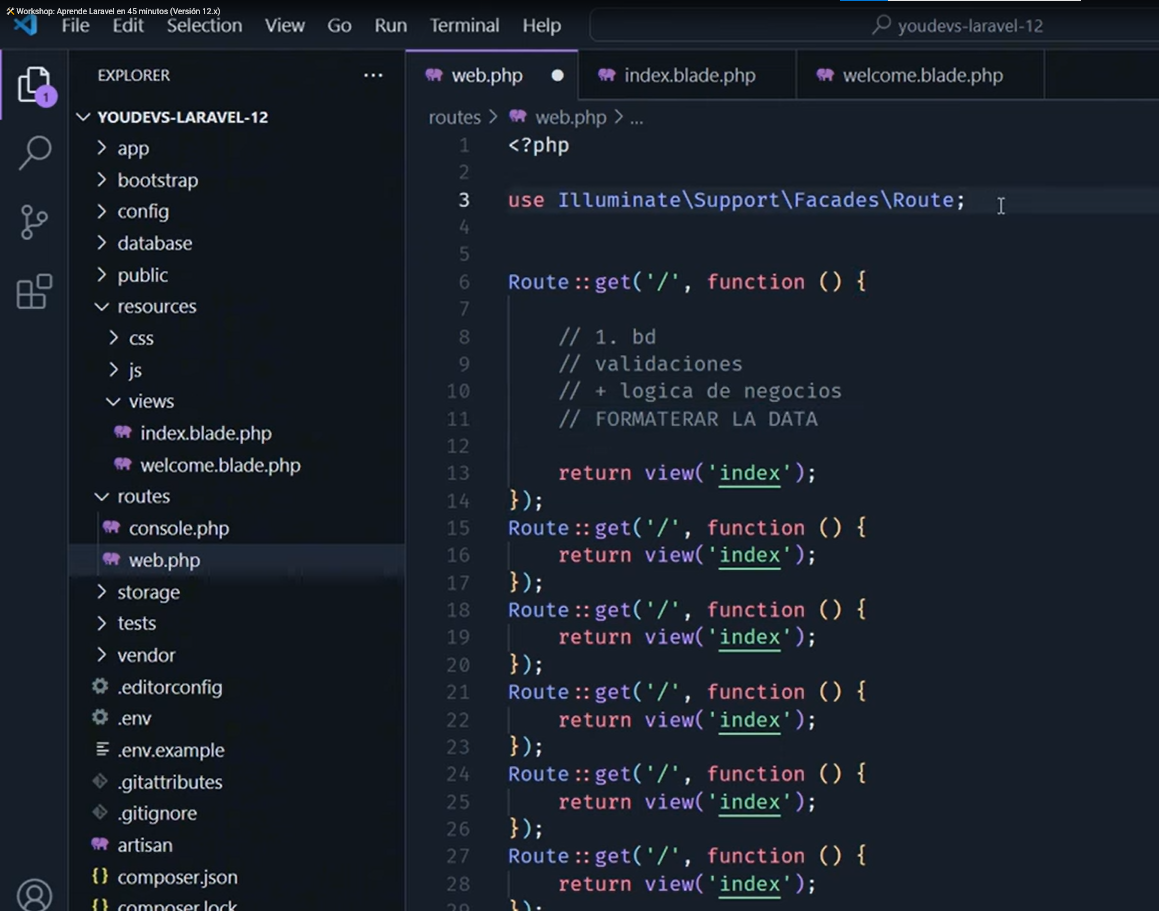
En las rutas la extensión blade no se especifica pues no es necesario, Laravel lo detecta por su cuenta.



Con el flujo básico de Laravel Ruta Vista

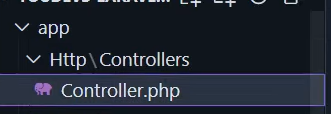


Tendríamos que escribir todas las rutas en el archivo web.php. Consultar la bd, validaciones, etc todo declarado en routes.



Para evitar eso, en lugar de conectar las rutas a una vista, las conectamos a un controlador.

Se puede crear nuevos controladores.

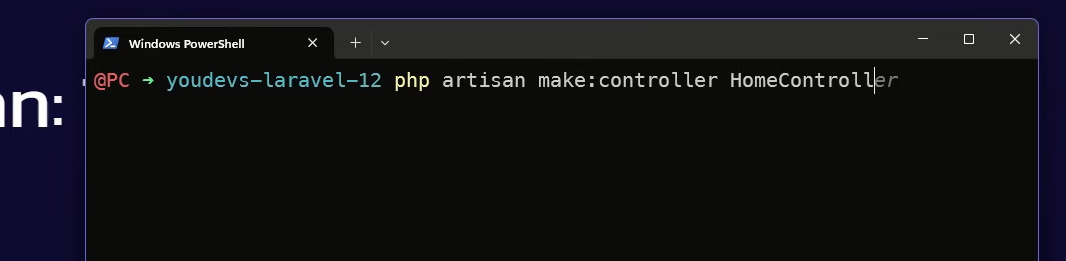


Artisan

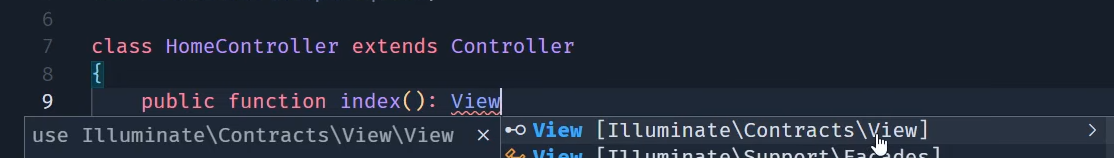


Para crear un controlador

php artisan make:controller HomeController



Para importar vistas:

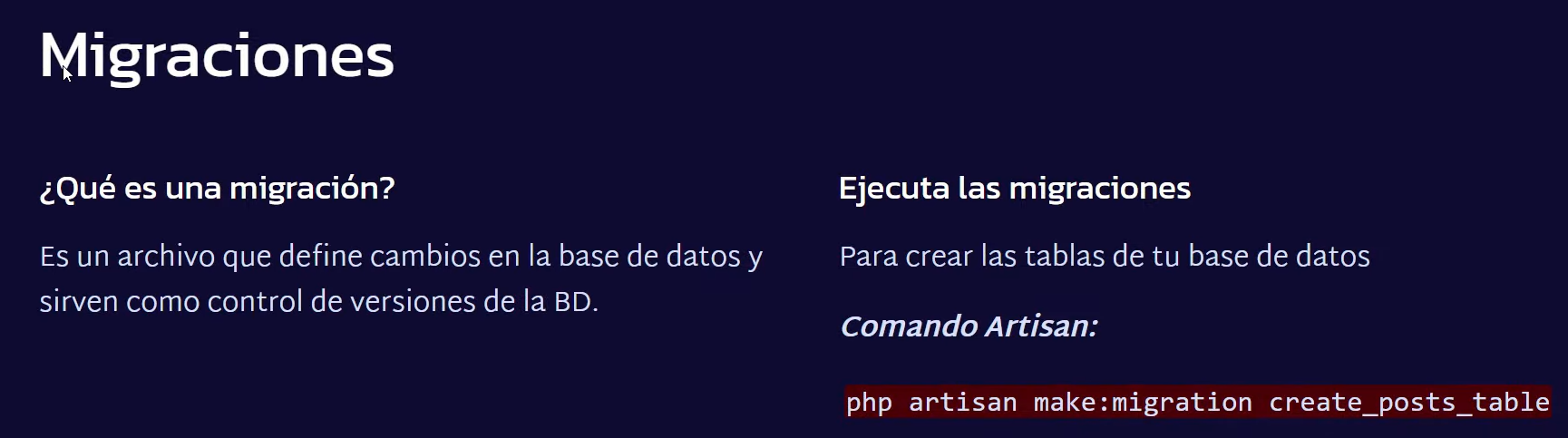


## Modelos para consultar BD



php artisan migrate: ejecuta las migraciones

## ¿Qué son las migraciones?



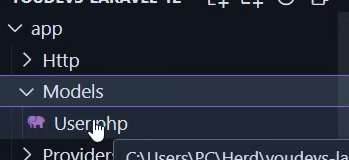
Al migrar, los archivos de las migraciones toman efecto. Solo toman efecto las migraciones que se encuentren en el archivo ../database/migrations

**¿Por qué sirve como un sistema de versiones de BD?**

Porque crea un Timestamp que se hacen y deshacen en orden especifico

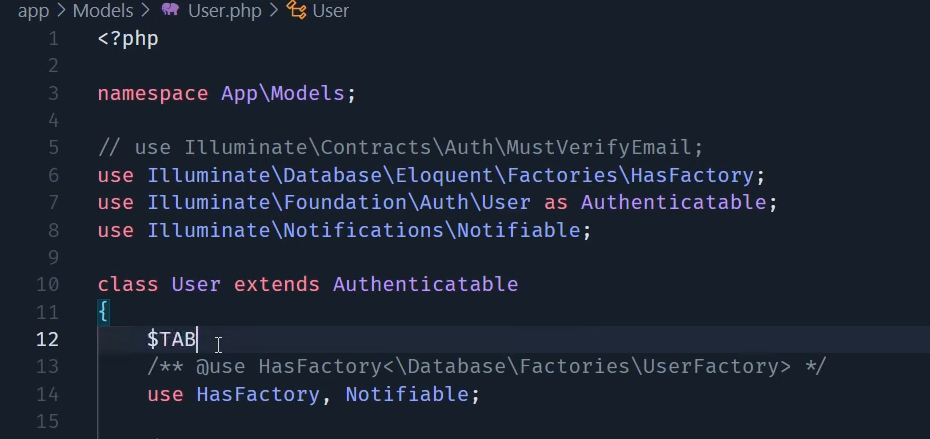
Modelos

Los modelos se encuentran en la ruta:

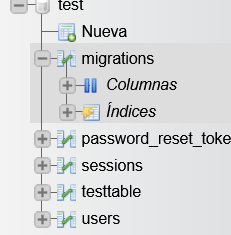


Todo esto nos permite interactuar con la BD sin la ayuda de SQL, sino con php.

Es una clase que está en singular:



Mientras que la tabla real está en plural:



<?php

namespace App\Http\Controllers;

use Illuminate\Contracts\View\View;

use Illuminate\Http\Request;

use App\Models\User;

class HomeController extends Controller

{

    public function metodo(): View {

        //conexion a la tabla con eloquent

        //find busca por id, al 1 en este caso

        $user = User::find(1);

        //vardump die

        // die(var\_dump($user));

        dd($user);

        return view("index");

    }

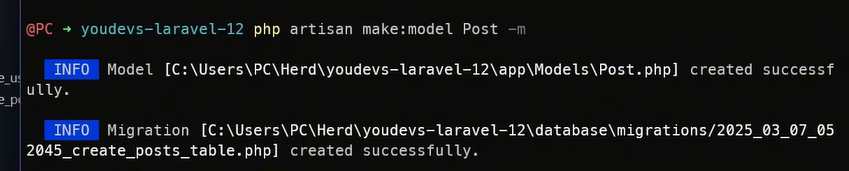
}

## Como enviar datos a la vista

Crear tabla con php artisan:



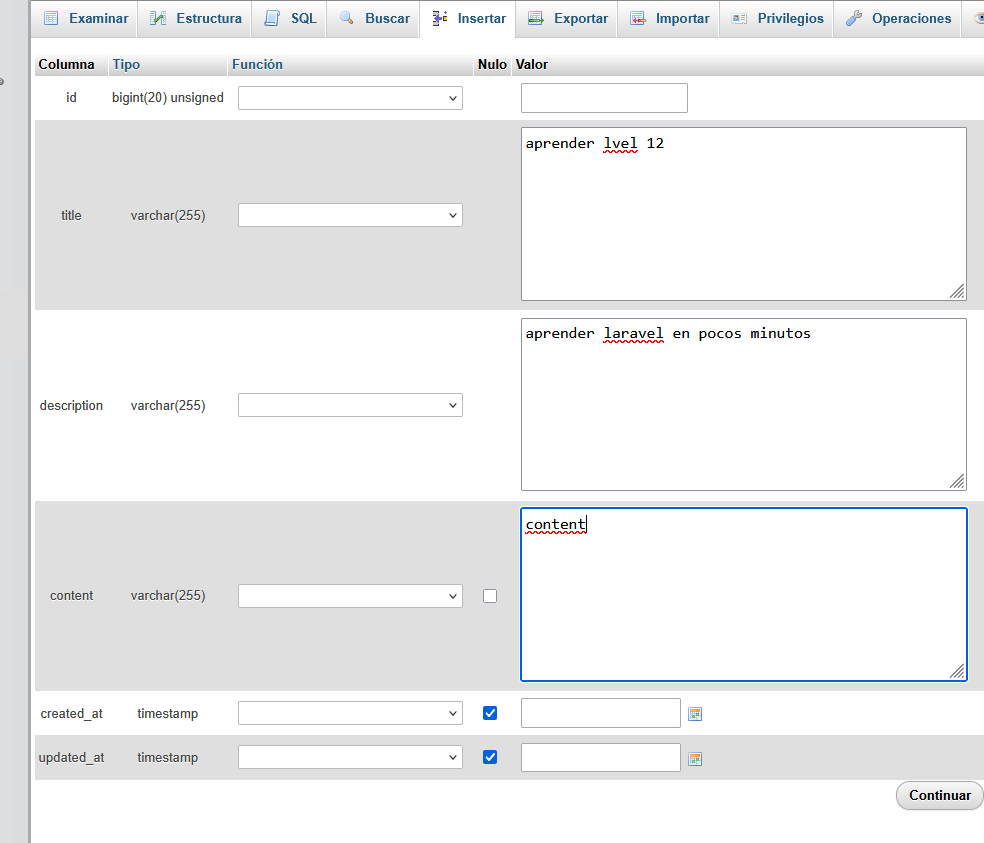
Creación de modelo y archivo de migración



Migración

Php artisan migrate

Create y updated son campos que se llenan cuando se hace con código de php, pero no cuando se hace el proceso manual.

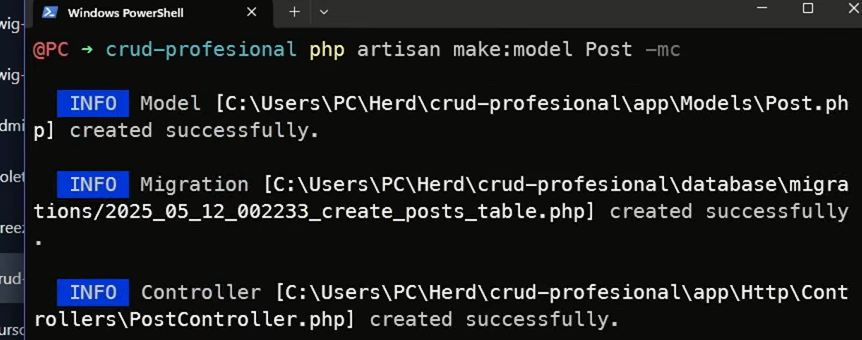


# CRUD Laravel

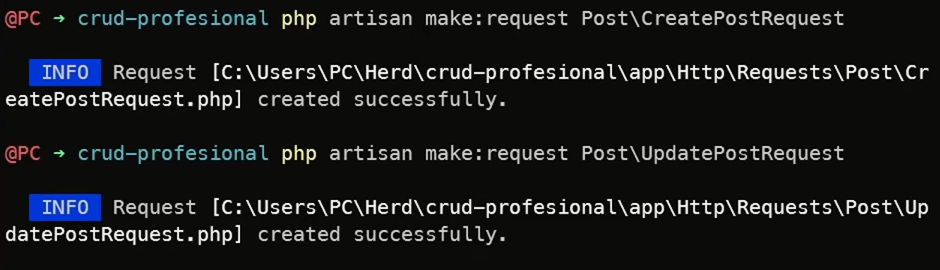
1. Mediante el modelo se maneja la información de las tablas mediante código php.
2. La migración crea la forma de la tabla.



Crear el modelo, migración y controlar de una sola vez.

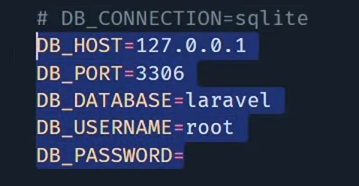


Creacion de FormResquest:



Uno para crear y otro para actualizar la request.

EN laravel 12, esta listo para trabajar en sqlite.



Necesitamos un controlador de recursos un ResourceController:

Este comando no nos sirve: php artisan make:model PostTwo -mc

Este si: php artisan make:controller PostTwoController –resource

Tenemos que separar responsabilidades idealmente:

<https://youtu.be/R94WgaWQJjo?si=rKFnL02Psc8Ddm-5&t=815>



Aquí debemos separar   
la validación de   
la lógica de creacion y   
la repuesta de redirección en un solo lugar.

La validación va en el Form Request

## Service Class

Para la lógica de creación crearemos una ServiceClass:



En la service Class encapsulamos toda la lógica relacionada con una entidad, en este caso la entidad PostTwo.

Los servicios se crean en:



La ruta app/Services/<Entidad>/<Entidad>Service.php

En este caso el controlador solo se encarga de mantener el flujo.

public function store(CreatePostTwoRequest $request)

    {

        $this->service->create($request->validated());

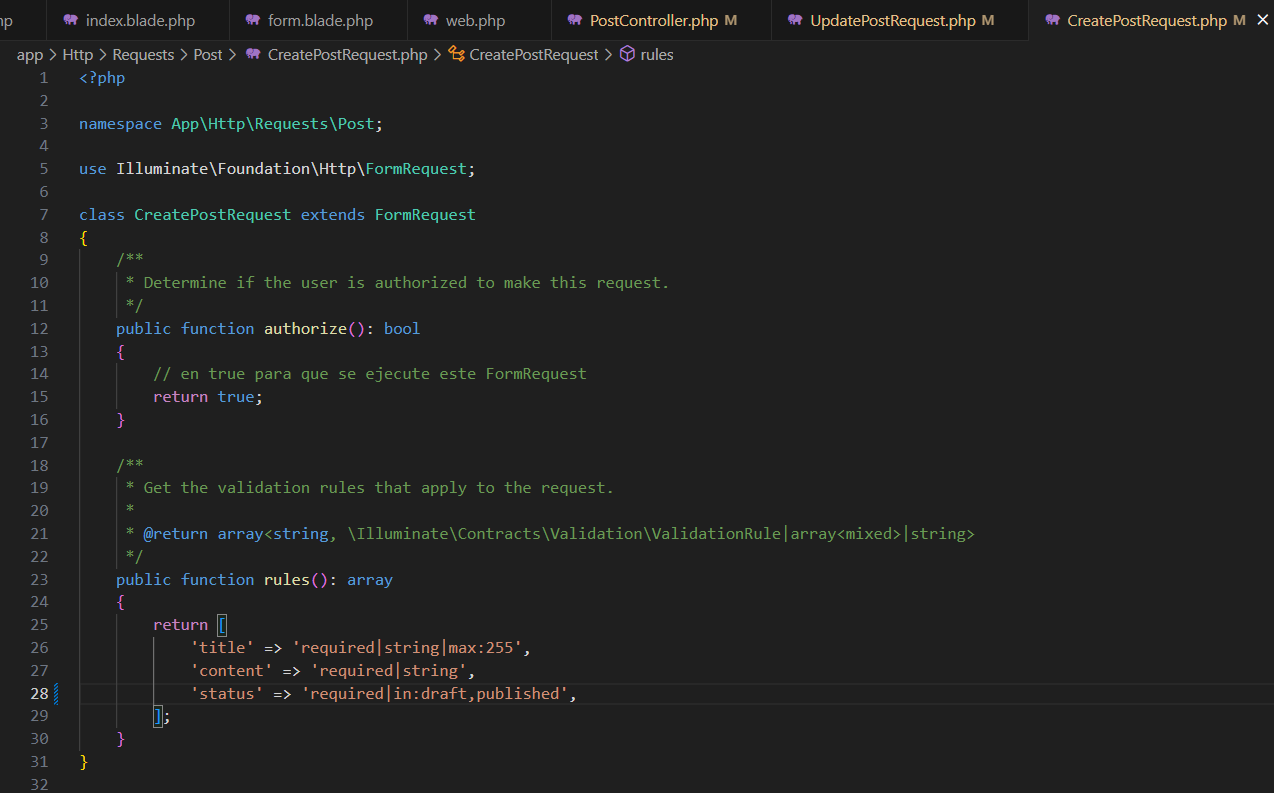
        return redirect()->route('postsTwo.index')->with('message', 'Post creado exitosamente!');

    }

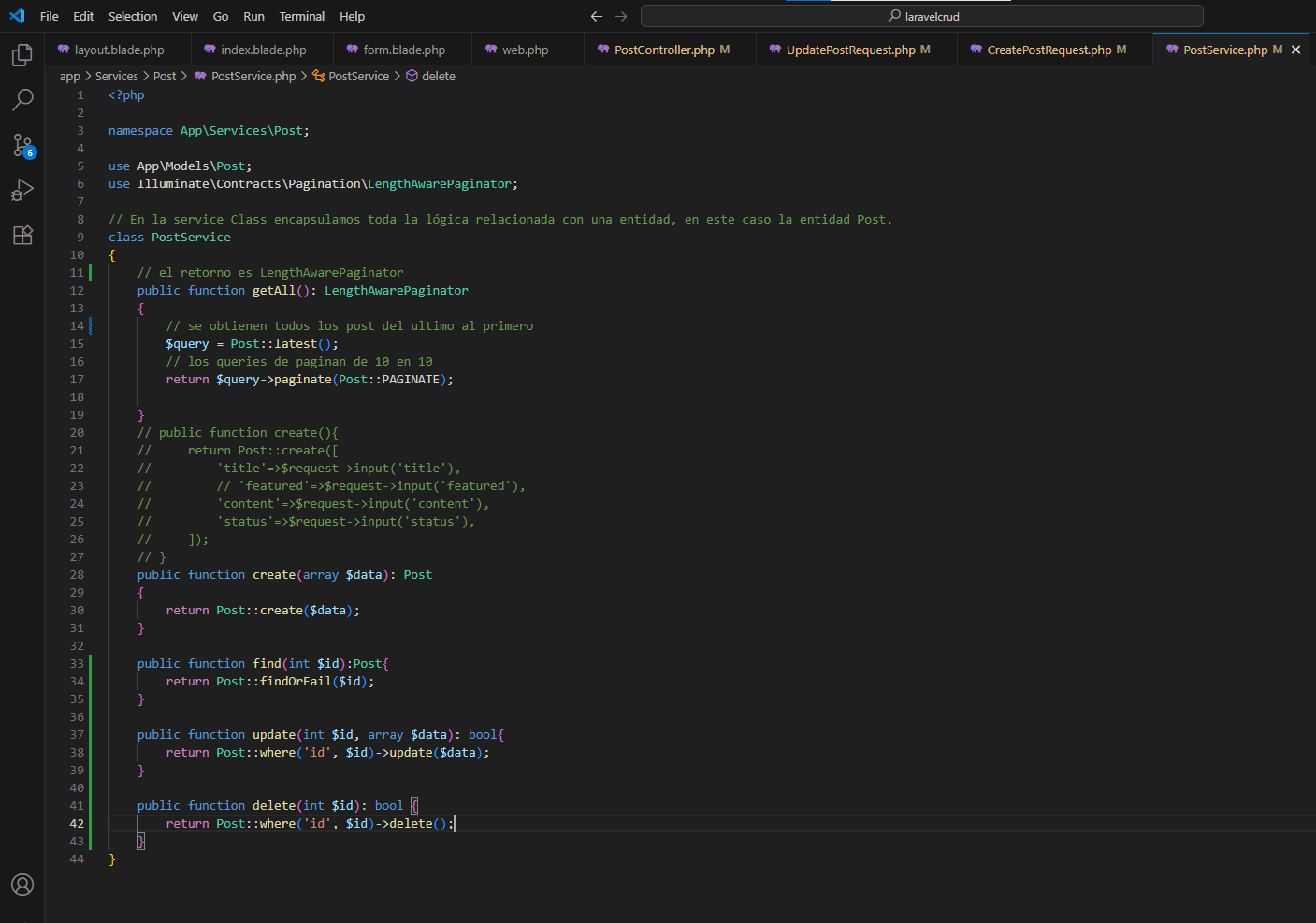
Antes de seguir crearemos las vistas minimas para seguir con el proceso.

Escribimos el comando:   
php artisan make:component Layout

Las validaciones van en los FormRequest:



La lógica de negocios va en el ServiceClass:



Y el controlador controla el flujo de como se ejecuta esto.

Búsqueda por filtro: <https://youtu.be/R94WgaWQJjo?si=ima8p6B_PF3ZmIhb&t=2081>

# Conocimientos generales

Primero, siempre se crea el proyecto.

Con Laravel podemos usar Herd, donde ya tenemos composer instalado, aunque tal vez necesitemos instalar composer.  
——————————————————————————————————————————  
Para Lumen tendremos que usar los comandos.

#instalar composer si no lo teniamos

composer create-project —prefer-dist laravel/lumen <nombre del proyecto>

Para poder usar artisan dentro del proyecto de Lumen instalamos:  
composer require flipbox/lumen-generator

Ingresamos a la carpeta del proyecto generada con el primer comando y luego pegamos estas lineas:  
$app->register(App\Providers\AppServiceProvider::class);

$app->register(App\Providers\AuthServiceProvider::class);

$app->register(App\Providers\EventServiceProvider::class);

$app->register(Flipbox\LumenGenerator\LumenGeneratorServiceProvider::class);  
——————————————————————————————————————————Si vamos a usar el ORM de Laravel, Eloquent, entonces debemos descomentar:  
$app->withEloquent();  
——————————————————————————————————————————

# Establecer una bd y las credenciales en .env

En este paso lo que debemos hacer es acceder al archivo .env y cambiar las credenciales segun lo requerido.

## Establecer archivos de migracion y sus modelos

Dentro de los achivos de migracion y modelos se debe especificar las tablas y relaciones que tendran. Para ello primero generamos los archivos.  
Crear el modelo, migración y controlar de una sola vez, todos los archivos minimos necesarios para realizar el proyecto.  
php artisan make:model <Nombre del modelo> –mc  
——————————————————————————————————————————  
Pero mejor podriamos crear inicialmente el modelo y la migracion:  
php artisan make:model <Nombre del modelo> –m

Y despues generar el resource controller. El cual no es mas que un controlador normal con los nombres de metodos necesarios para el CRUD de cualquier proyecto.  
php artisan make:controller <Nombre del controlador> –r  
#por convencion YO escribo el nombre de controlador de la siguiente forma: UserController.

### Migraciones

Los archivos de migracion nos sirven como un control de versiones de nuestra base de datos, estos archivos permiten establecer y compartir la definicion esquematica de la BD de la aplicacion.

Para generar los archivos de migracion usamos el comando:  
php artisan make:migration <Nombre declarativo de la migracion, o sea del proposito con el que se crea>.

Un archivo de migracion recien creado lucira asi:

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    public function up(): void

    {

        Schema::create('posts', function (Blueprint $table) {

            $table->id();

            $table->timestamps();

        });

    }

    public function down(): void

    {

        Schema::dropIfExists('posts');

    }

};

### Creacion de tablas

Para crea una nueva tabla en la base de datos usamos el metodo create de la facade(fachada) Schema dentro del metodo ya definido ‘up’.

El metodo create acepta 2 argumentos, el primero es el nombre de la tabla y el segundo es un closure el cual recibe como argumento un objeto Blueprint que sera usado para definir las solumnas de la tabla.

Por defecto cada migracion creada generara en el codigo hace que las columas basicas sean el id y timestamps, id es el PrimaryKey de la tabla, y timestamps generara las columnas created\_at y updated\_at.

Tipos de columnas

En el link se ecnuentran todos de la version 12: <https://laravel.com/docs/12.x/migrations#available-column-types>

Generalmente se usa este convencion para definir las columnas:  
$table->tipo de la tabla('nombre de la tabla');

Pero a mi parecer los mas importantes son:

**El metodo id (https://laravel.com/docs/12.x/migrations#available-column-types)**al cual se le debe pasar un nombre como argumento si se desea que la tabla tenga un nombre distinto al de id.  
$table->id(‘idloquesea’);

**El metodo string() (https://laravel.com/docs/12.x/migrations#column-method-string**)  
The string method creates a VARCHAR equivalent column of the given length:

$table->string('name', length: 100);

**El metodo integer** [**https://laravel.com/docs/12.x/migrations#column-method-integer**](https://laravel.com/docs/12.x/migrations#column-method-integer)$table->integer('votes');

Los tipos de fecha y hora (https://laravel.com/docs/12.x/migrations#dates-and-times-method-list)

**El medo foreignIdFor() (https://laravel.com/docs/12.x/migrations#column-method-foreignIdFor)**Asigna un nombre a la ForeingKey en base al nombre de la tabla con la que esta relacionada el modelo.  
$table->foreignUlid('user\_id');

**# Unsigned can hold a larger positive value and no negative value.**

#### Modificadores de columnas

Los mas importantes son:

**null**

to make the column "nullable", you may use the nullable method:

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

Schema::table('users', function (Blueprint $table) {

$table->string('email')->nullable();

});

https://laravel.com/docs/12.x/migrations#column-modifiers  
->autoIncrement()

->default($value)

->invisible()

->nullable($value = true)

## Modificacion de las columnas

Para ello debera crear otra migracion y dentro con el metodo table de la facade Schema. Establecer en el primer argumento el nombre de la tabla que desea cambiar, luego en el closure define el nombre de la columna seguido del metodo change().

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration {

    public function up(): void

    {

        Schema::table('usuarios', function (Blueprint $table) {

            $table->string('password', 80)->change();

        });

    }

    public function down(): void

    {

        Schema::table('usuarios', function (Blueprint $table) {

            $table->string('password', 40)->change();

        });

    }

};

Renombrar Tablas  
https://laravel.com/docs/12.x/migrations#renaming-columns

Eliminar Tablas  
<https://laravel.com/docs/12.x/migrations#dropping-columns>