[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 rutas 30](#_Toc11)

[Parametros de ruta (https://laravel.com/docs/9.x/routing#route-parameters) 32](#_Toc12)

[Establecer archivos de migracion y sus modelos 32](#_Toc13)

[Migraciones 33](#_Toc14)

[Creacion de tablas 34](#_Toc15)

[Modificadores de columnas 35](#_Toc16)

[Modificacion de las columnas 37](#_Toc17)

# Laravel/Lumen, conocimientos basicos y generales

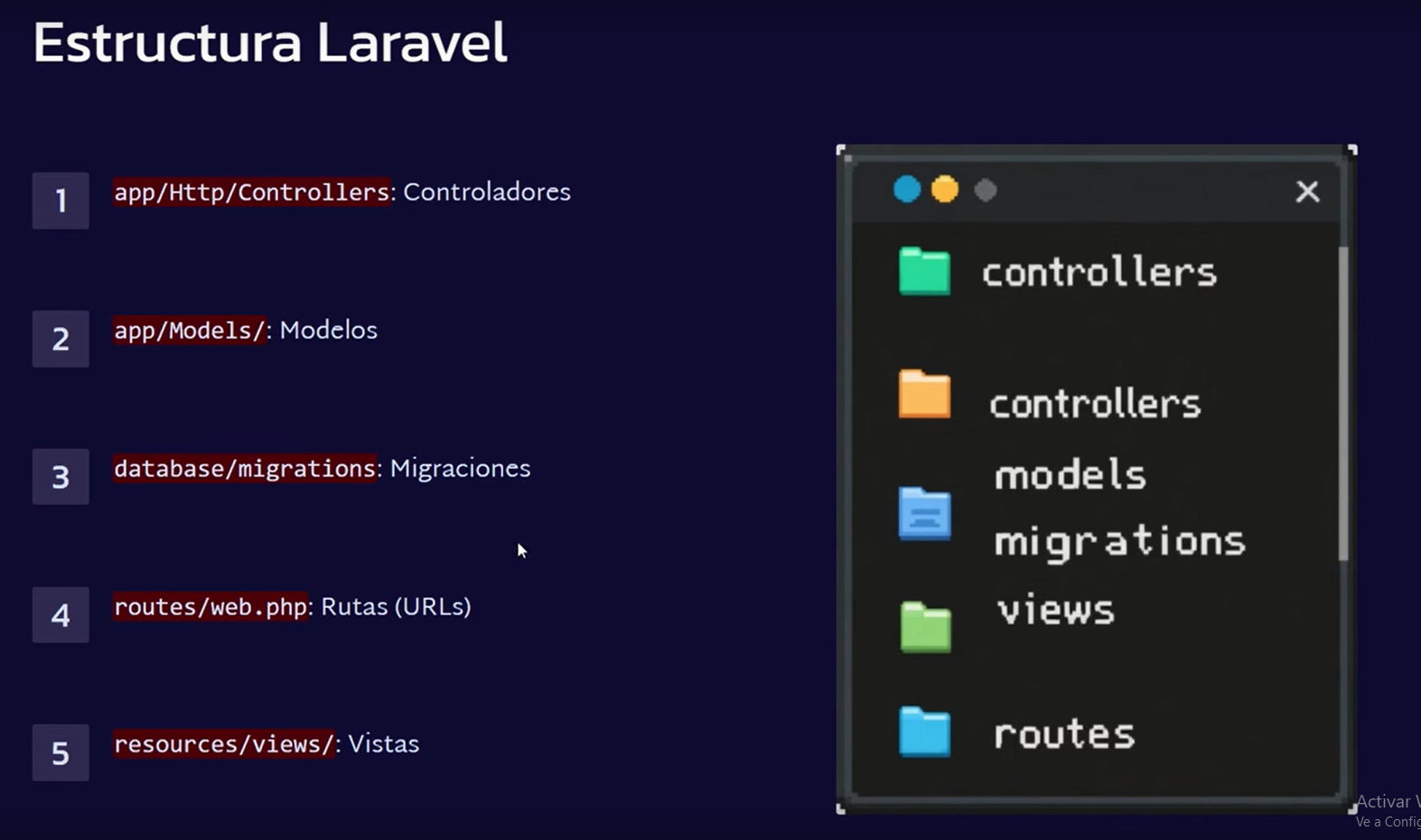
## Como iniciar un proyecto

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

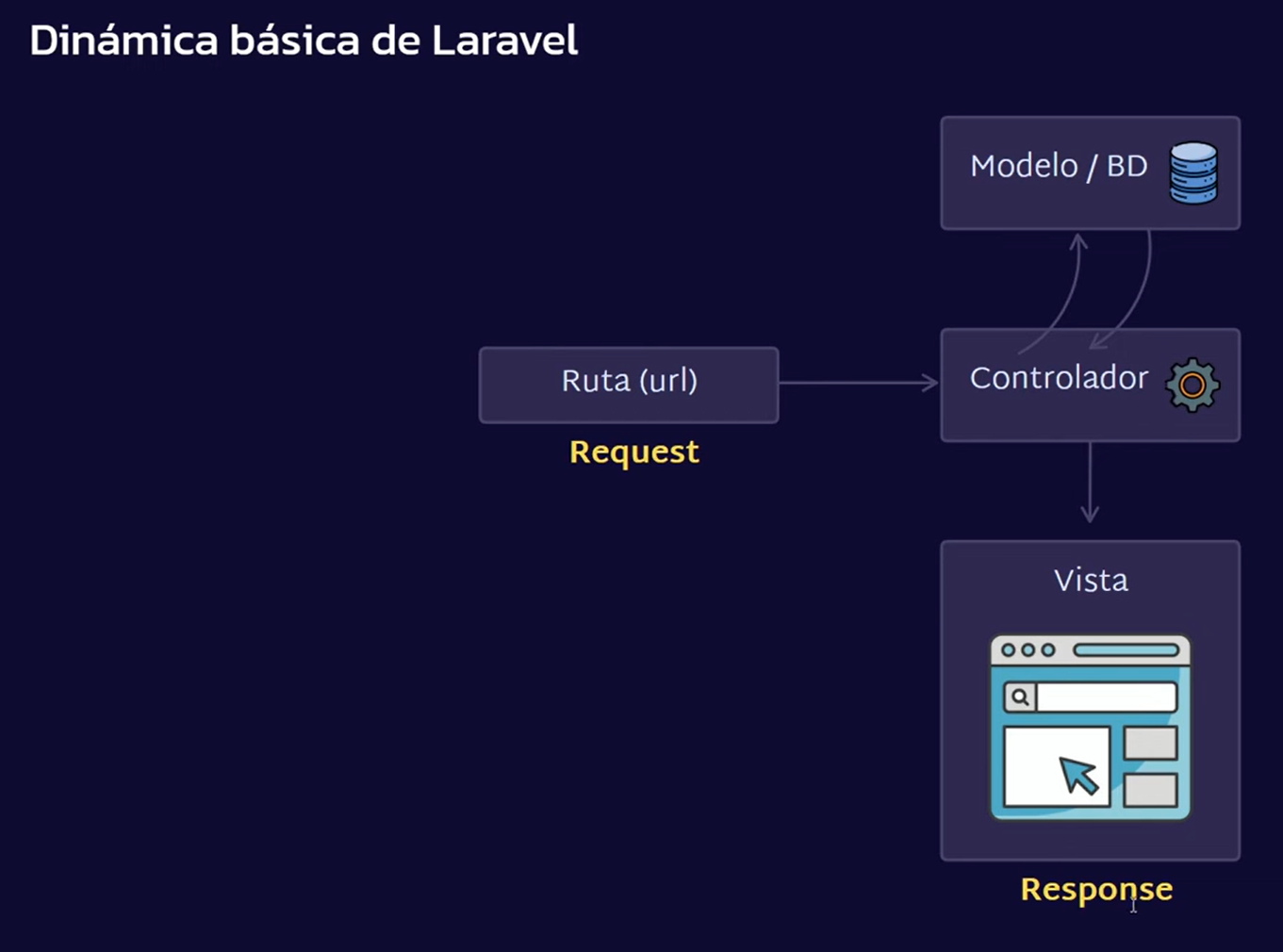
## Laravel Herd

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

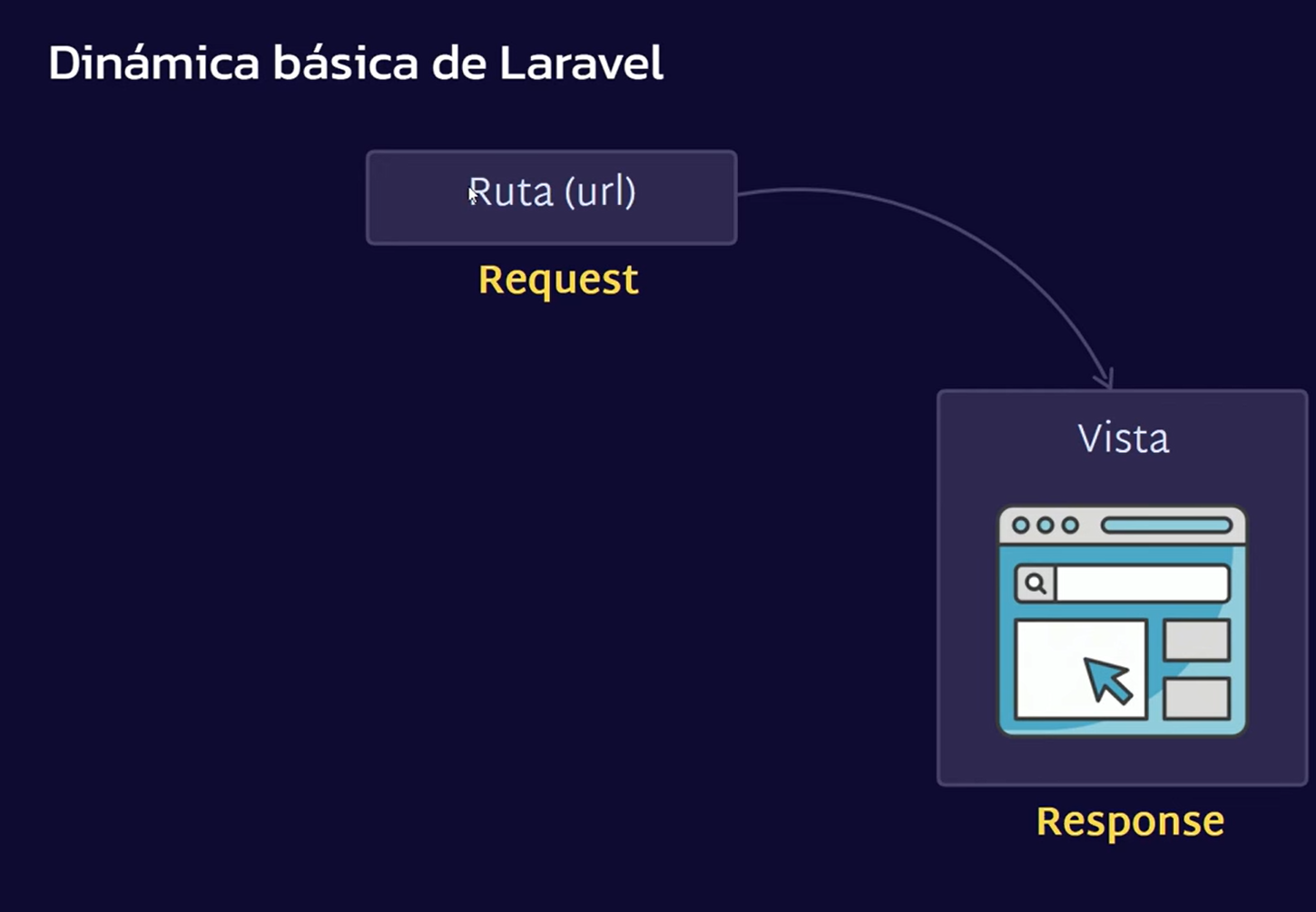
Estructura Laravel  
https://youtu.be/PaYwBzH\_WSI?si=G-4f4lxmv9vWNlQm



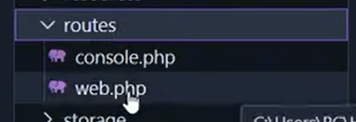
La dinamica basica de Laravel  
<https://youtu.be/PaYwBzH_WSI?si=K4KcpqJHNYTrMm0I&t=408>



La dinámica de Laravel puede reducirse a:



Las rutas se guardan en:



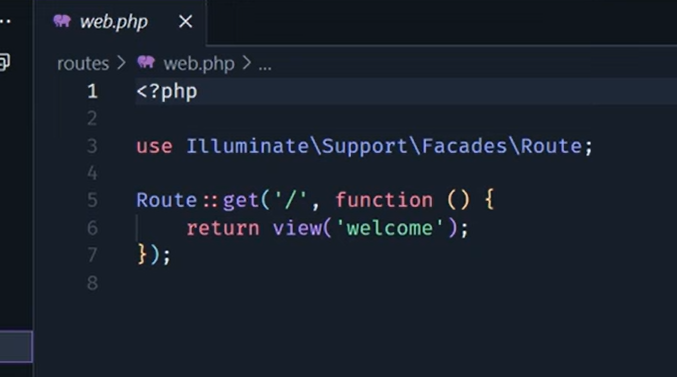
Ejemplo:

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.

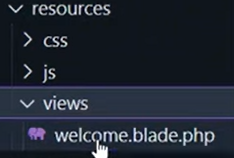
Hay una ruta y una respuesta.



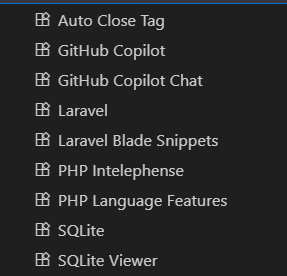
Las vistas:

Se encuentran en resources/views.

Tienen la extensión blade.

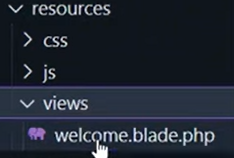


Lista de extensiones usadas para desarrollar en Laravel:

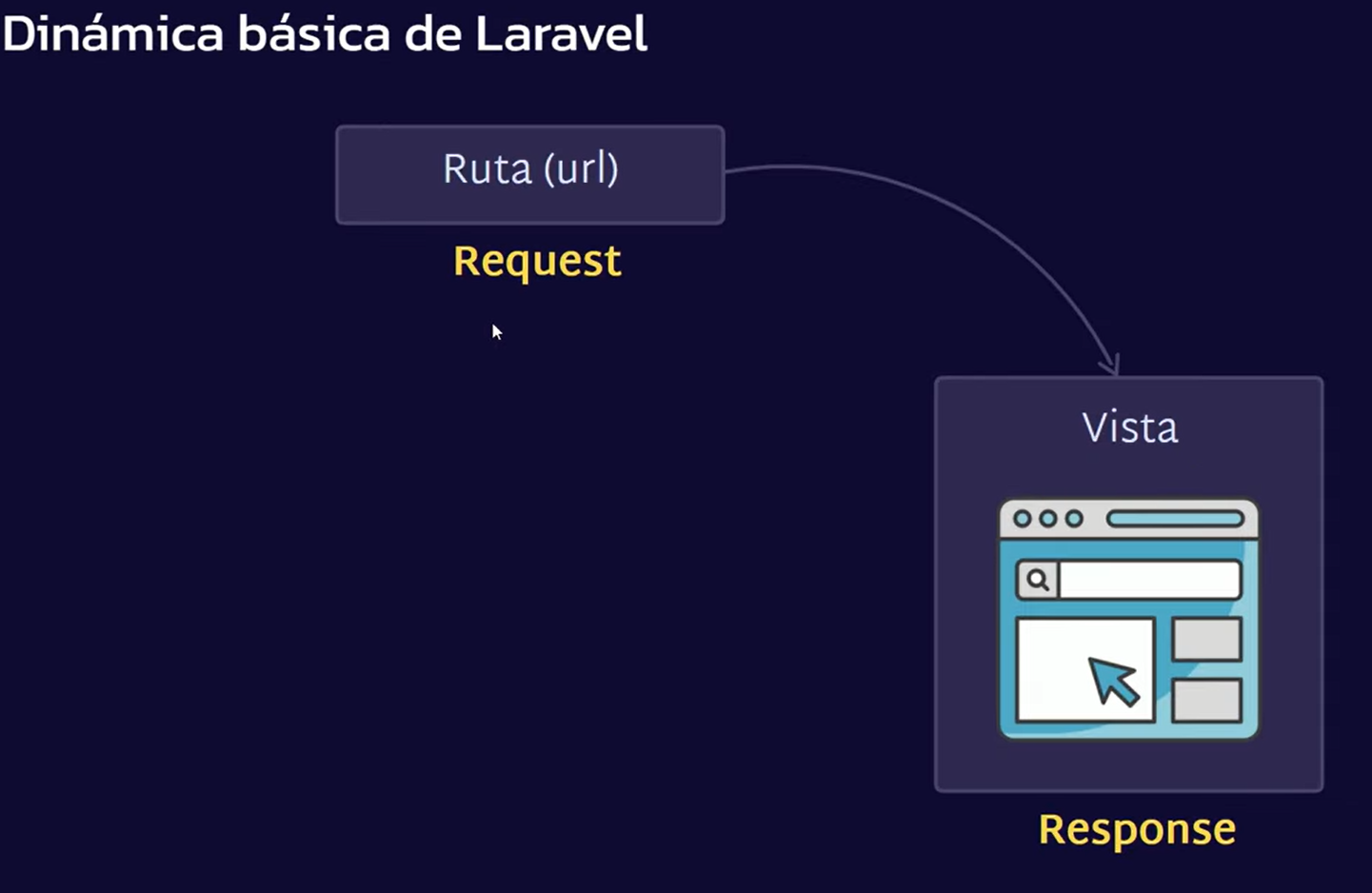


Blade es el motor de plantillas que simplifica el código. Porque se nos facilitan directivas Blades.

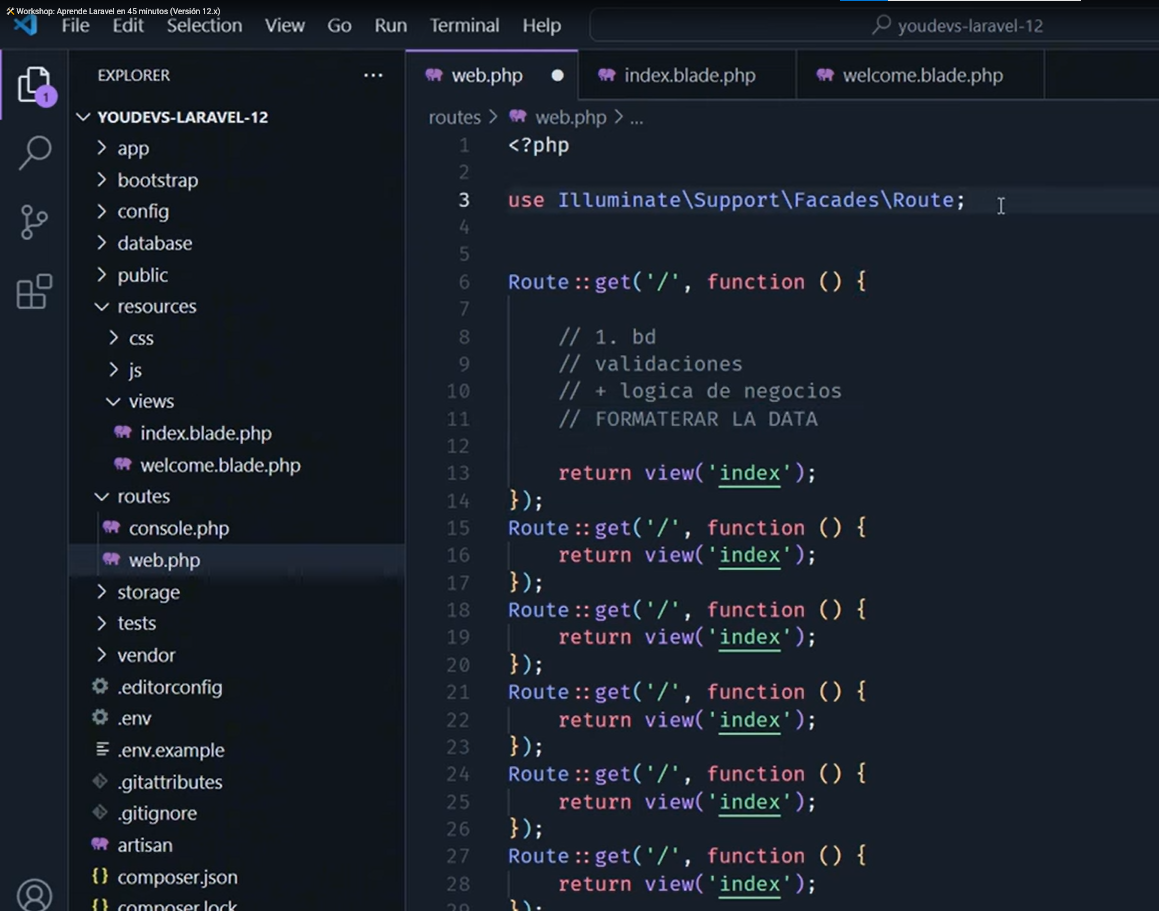
En las rutas la extensión blade no se especifica pues no es necesario, Laravel lo detecta por su cuenta.



Con el flujo básico de Laravel Ruta Vista

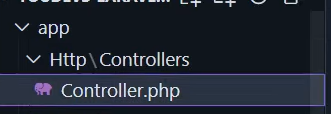


Tendríamos que escribir todas las rutas en el archivo web.php. Consultar la bd, validaciones, etc todo declarado en routes.



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

Se puede crear nuevos controladores.

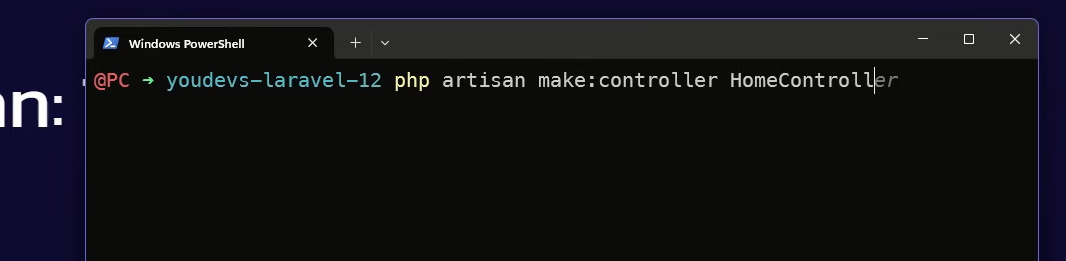


Artisan

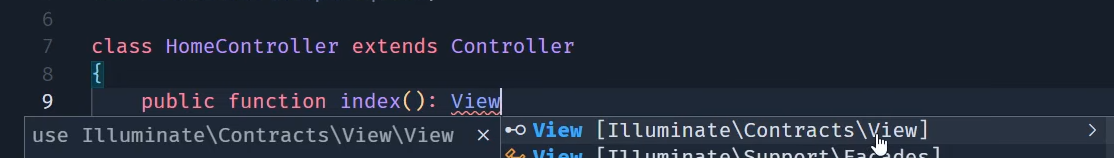


Para crear un controlador

php artisan make:controller HomeController



Para importar vistas:

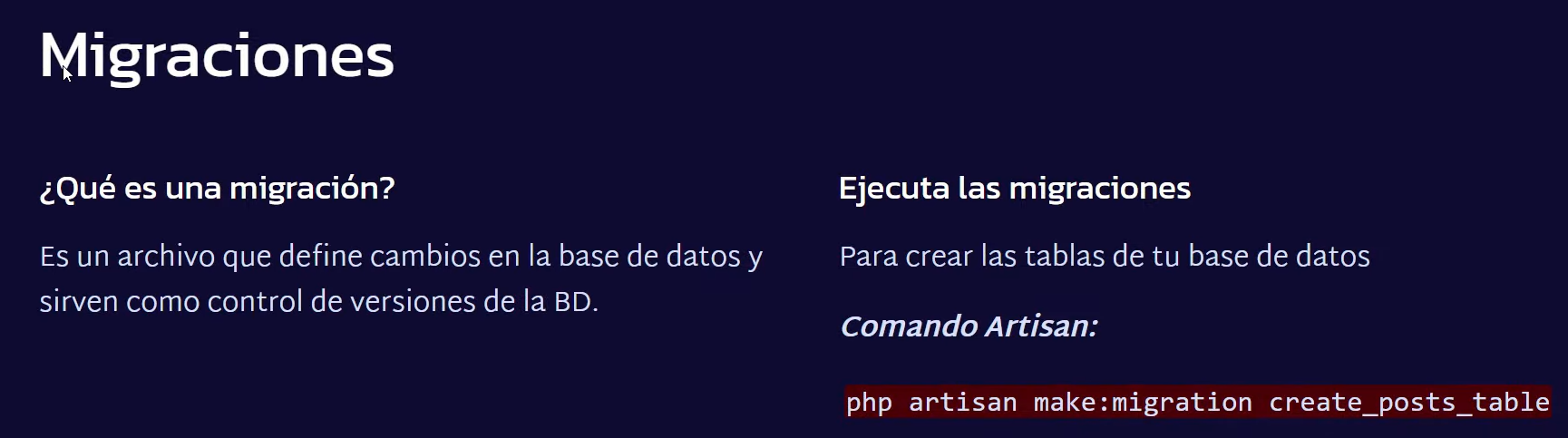


## Modelos para consultar BD



php artisan migrate: ejecuta las migraciones

## ¿Qué son las migraciones?



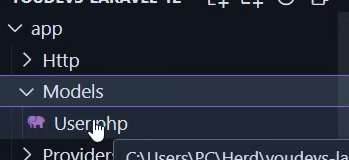
Al migrar, los archivos de las migraciones toman efecto. Solo toman efecto las migraciones que se encuentren en el archivo ../database/migrations

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

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

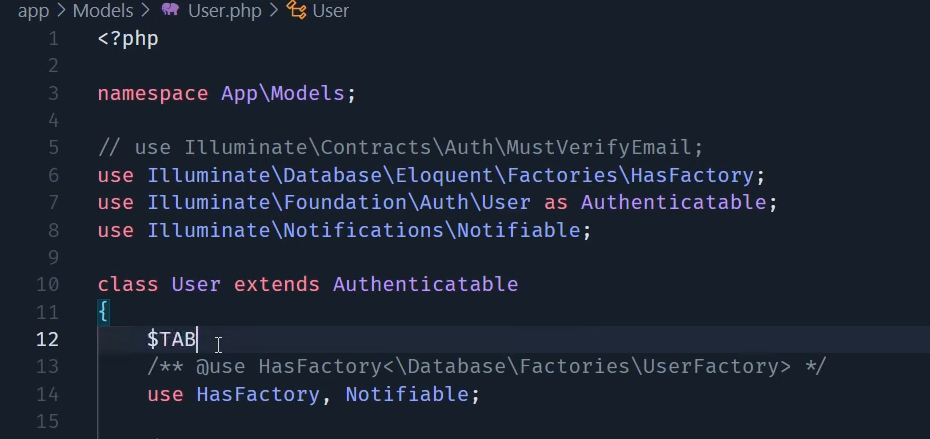
Modelos

Los modelos se encuentran en la ruta:

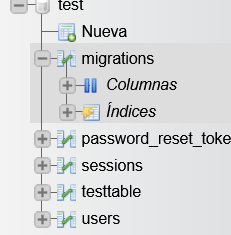


Todo esto nos permite interactuar con la BD sin la ayuda de SQL, sino con php.

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");

    }

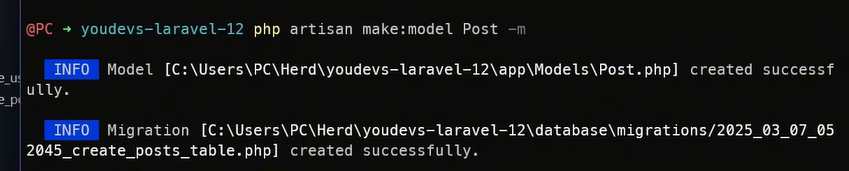
}

## Como enviar datos a la vista

Crear tabla con php artisan:



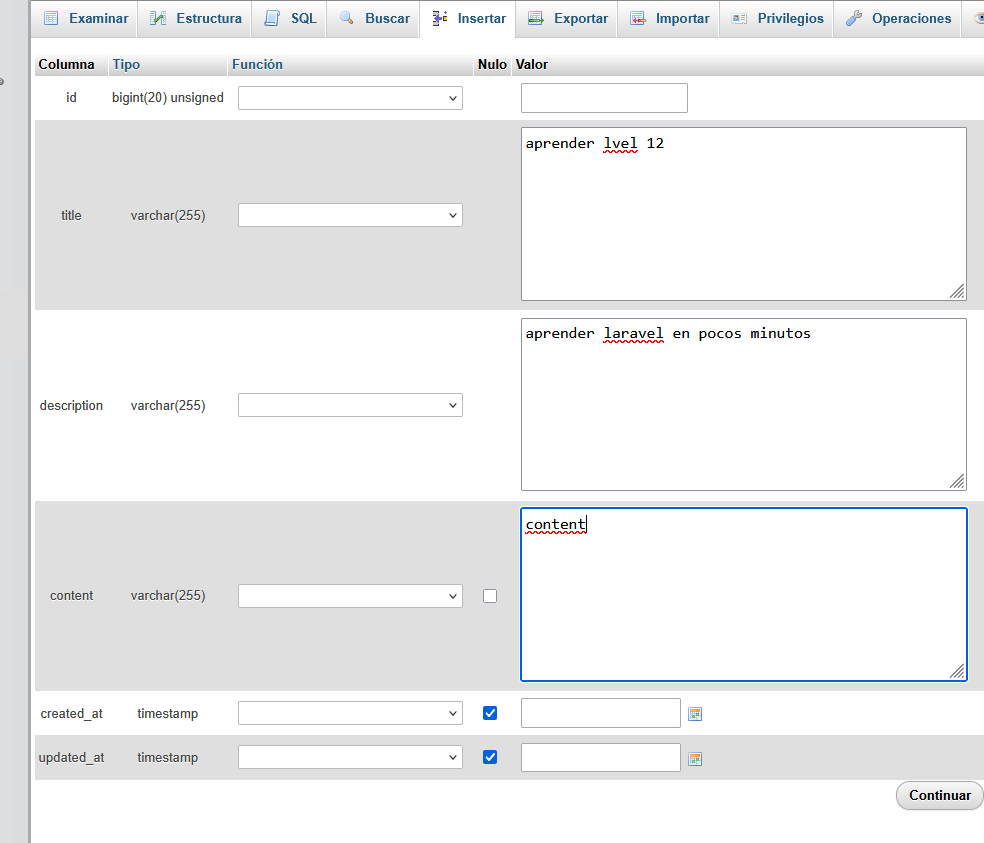
Creación de modelo y archivo de migración



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.

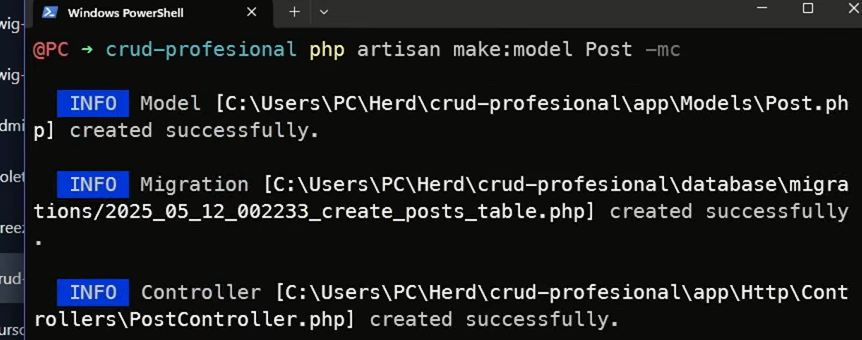


# CRUD Laravel

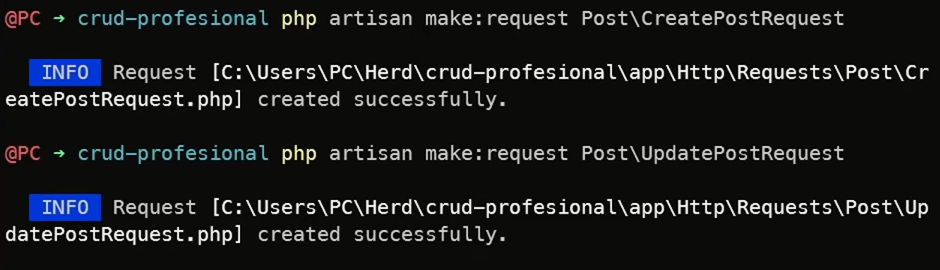
1. Mediante el modelo se maneja la información de las tablas mediante código php.
2. La migración crea la forma de la tabla.



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

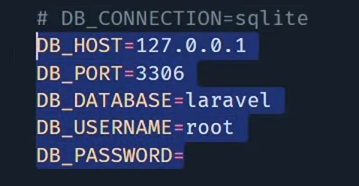


Creacion de FormResquest:



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());

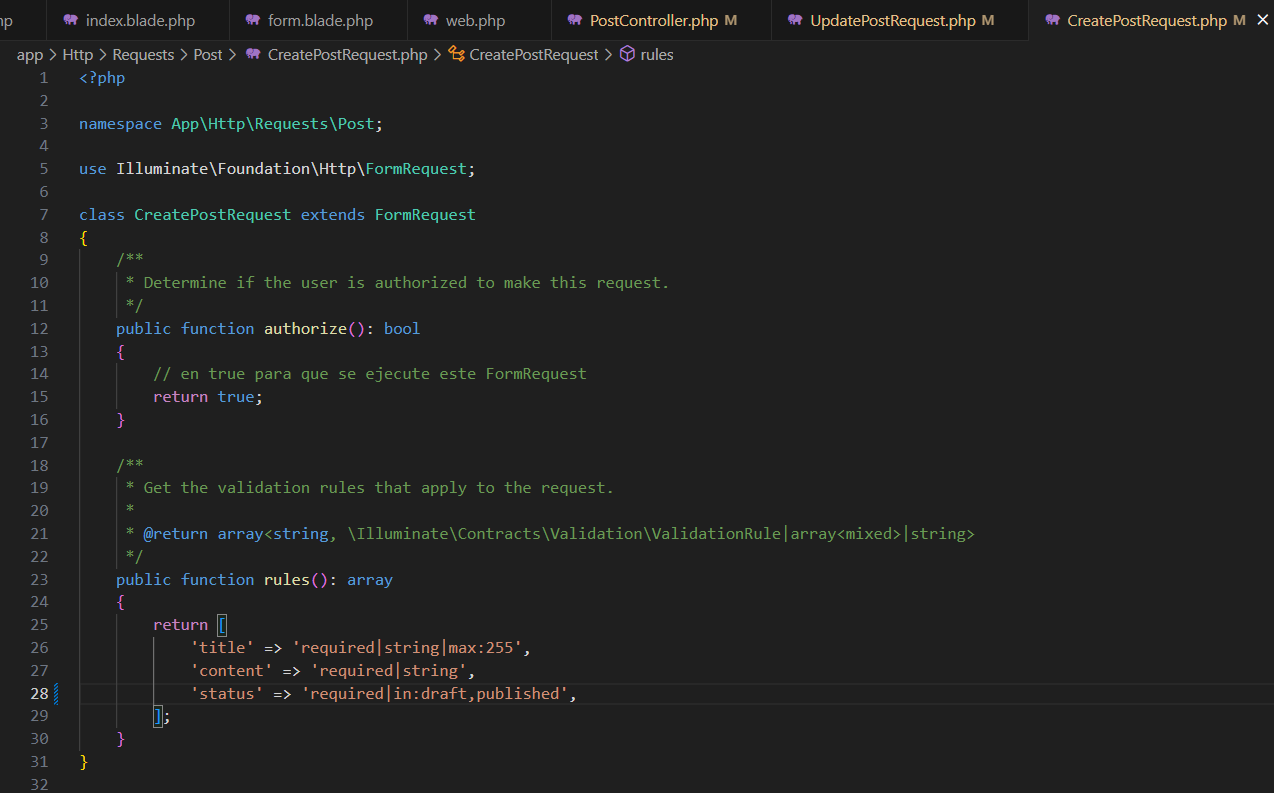
        return redirect()->route('postsTwo.index')->with('message', 'Post creado exitosamente!');

    }

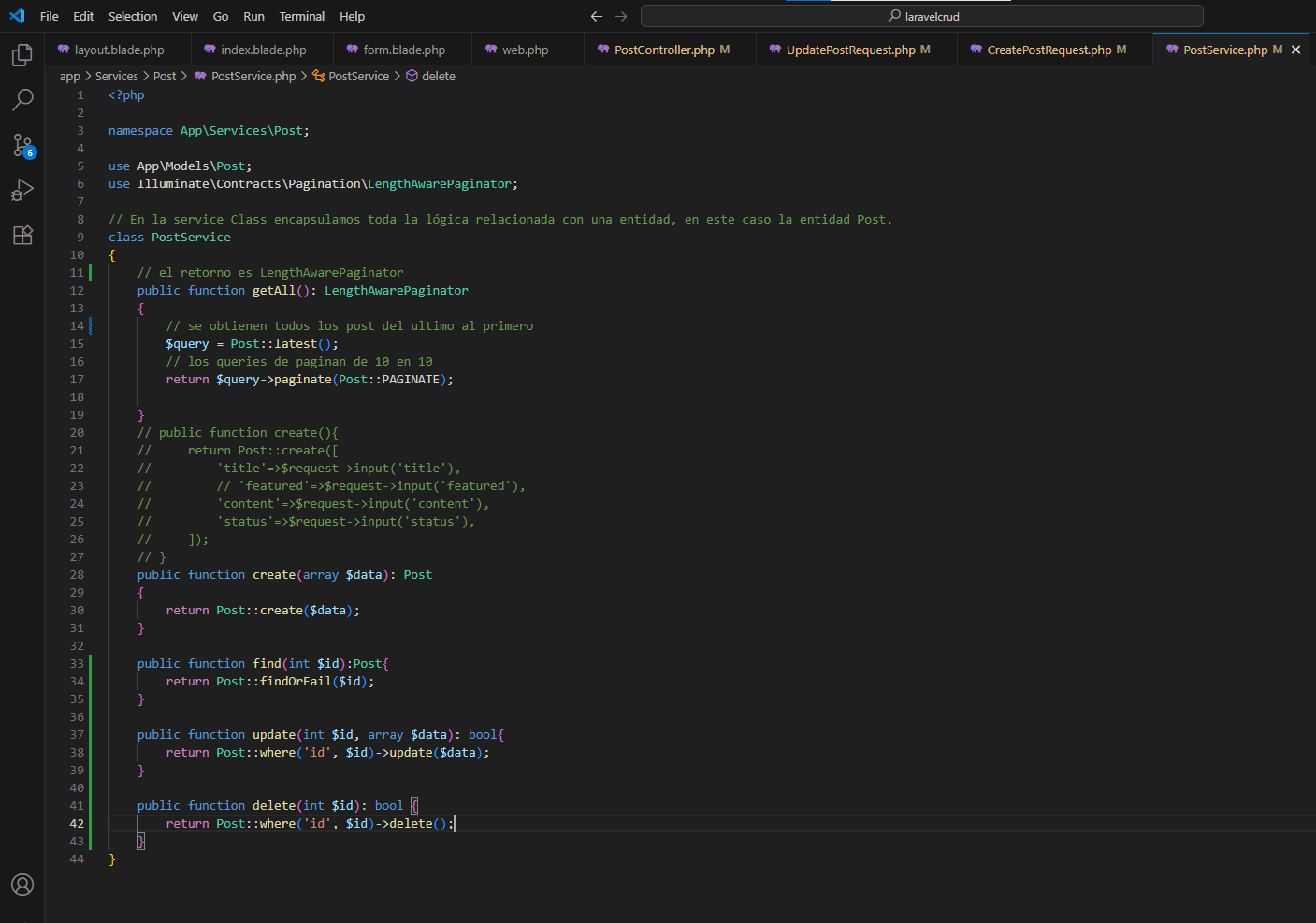
Antes de seguir crearemos las vistas minimas para seguir con el proceso.

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 rutas

Debemos establecer a las rutas con las que accederemos a la información.

Los archivos de rutas se deben establecer dentro de la carpeta routes por defecto tenemos el archivo web.php.

Aqui se habla de middlewares web middleware group y api middleware group: <https://laravel.com/docs/9.x/routing#the-default-route-files.>

Deberemos acceder a las rutas a tarves del nagavor de la siguiente forma <http://example.com/user.>

Routes defined in the routes/api.php file are nested within a route group by the RouteServiceProvider. Within this group, the /api URI prefix is automatically applied so you do not need to manually apply it to every route in the file. You may modify the prefix and other route group options by modifying your RouteServiceProvider class.

Metodos de rutas disponibles: <https://laravel.com/docs/9.x/routing#available-router-methods>

HTTP multiple verbs:

Metodo match o metodo any para cualquier verbo.

Route::match(['get', 'post'], '/', function () {

//

});

Route::any('/', function () {

//

});

Redireccionamiento de rutas: <https://laravel.com/docs/9.x/routing#redirect-routes>  
——————————————————————————————————————————**Rutas en Lumen**

$router->get('/', function () use ($router) {

    return $router->app->version();

});

$router->get('/usuarios/ordenes', [

    'uses'=>'UsuarioController@showAndOrdenes'

]);

——————————————————————————————————————————  
**Rutas en Laravel**

Route::get('/user', [UserController::class, 'index']);

Rutas para vistas (Laravel)

If your route only needs to return a view, you may use the Route::view method. Like the redirect method, this method provides a simple shortcut so that you do not have to define a full route or controller. The view method accepts a URI as its first argument and a view name as its second argument. In addition, you may provide an array of data to pass to the view as an optional third argument:

Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);

## Parametros de ruta (https://laravel.com/docs/9.x/routing#route-parameters)

Parametros opcionales: <https://laravel.com/docs/9.x/routing#parameters-optional-parameters>

**https://laravel.com/docs/12.x/routing#cors**

## 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.  
——————————————————————————————————————————  
o mejor aun:  
php artisan make:model CustomerPurchase -mr  
pues genera todos los archivos necesarios con los nombres usuarios  
——————————————————————————————————————————  
De querer crear los nombres de los modelos en español, se debe tener cuidado y asegurarse que es en español que se define el nombre de las tablas en las migraciones y también en los modelos se debe establecer el nombre de correcto de maner forzosa:

    protected $table = 'ordenes';

### 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>

Uso de JWT para proteger las rutas

La **tokenización**, cuando se aplica a la seguridad de los datos, se refiere al proceso de sustitución de un elemento de datos sensible por un equivalente no sensible denominado token.

COMO PROTEGER RUTAS de API con JWT y MIDDLEWARE LARAVEL LUMEN PT1 - YouTube (https://www.youtube.com/watch?v=n9v0xk67B8E)

(00:00) Api privada entonces lo que tenemos que hacer básicamente es que tenemos que empezar a generar rutas protegidas estas rutas protegidas lo que nos van a permitir es que a fuerza tengamos que loguearnos para que esto pueda estar funcionando.

Trabajamos mediante tokens Por qué mediante tokens porque el árabe porque el Lumen es un microframwork de Laravel, es decir con base en la estructura de Laravel es como crear un Lumen sin embargo una de las características importantes es que el **Lumen no tiene la parte de las sesiones** entonces la forma más fácil de controlarlo es mediante tokens para eso podemos **instalarnos una librería de JW t** que en este caso podría ser con   
composer require firebase/php-jwt   
esta librería de firebase php jwt lo que nos hace es que nos instala una librería para poder trabajar con JW t

**generar nuestro controlador de autenticación**

Así que en este caso de nueva cuenta podemos poner php artisant make controller y le vamos a decir que nuestro controlador se va a llamar out controller y listo con eso ya nos empezó a generar nuestro out controller si nos pasamos a la sección de nueva cuenta de App http controller si ya tenemos un out controller y observen como en este caso a diferencia del user no nos puso el Index ni el create ni el Store ni nada por el estilo

(02:32) Y esto por qué fue Pues fue justamente porque no le di la bandera de resource entonces para eso es lo que nos sirve la bandera de resource ahora dentro de out controller pues podemos empezar a generar muchas cosas podemos empezar a generar algunas partes importantes de esto lo primero que vamos a hacer es que necesitamos empezar a trabajar con nuestro modelo Así que le vamos a decir use up diagonal model diagonal user es el primer paso que debemos de hacer después de eso también Vamos a empezar a trabajar con la librería de jwt por lo

(03:11) tanto le vamos a decir use firebase diagonal invertida jwt diagonal invertida JW t porque así se llama la clase la siguiente que vamos a utilizar es la de laravel Lumen del controlador de enrutamiento entonces Aquí le voy a decir private y en este caso le voy a decir request y después de eso me voy a crear una función un método public función que en este caso va a ser mi constructor y le voy a decir que mi constructor va a traer del objeto request un request y le voy a decir que Dish request va a ser igual a

(03:50) el objeto rico es perfecto después de que tengamos esto Ahora lo siguiente Es que le vamos a decir public function y en este caso le voy a llamar JW t y JW t va a tener un una un argumento que viene del modelo user y que le voy a decir user Entonces el argumento user va a traer en principio un pay load y este paylor debe de traer algunos elementos el primero es el iss el issue for token en este caso le puedo decir por ejemplo que sea un Happy YouTube jwt el siguiente argumento es el suv que en este caso Pues básicamente es el subjet

(04:31) yo le puedo decir que esto puede venir de user en su campo ID el siguiente es el iat que es el tiempo cuando fue creado el jwt le puedo decir Time y en siguiente es el XP x que es el de expiración y le puedo decir que sea Time más 60 por 60 esto es porque acuérdense que son segundos Entonces más o menos esto debe durar alrededor de una hora una vez que ya tengo estos datos le puedo decir return jwt dos puntos en code y que me pase el payloud coma y puedo utilizar mis variables de entorno que en este caso va a ser el JW t- bajo secreto y le

(05:12) voy a decir también que el algoritmo de encriptación es el hs 256 si crees que mi contenido te aporta valor te pido por favor que me dejes un buen like y un comentario ya que esto ayuda al posicionamiento del mismo ahora de dónde saqué toda esta información o por qué me pide toda esta información esta información básicamente viene directamente de JW t si nosotros buscamos jwt en la documentación oficial Pues aquí nos dice justamente que el paylor para poder enviar Pues debe de estar más o menos en este formato y debe

(05:48) tener como ciertas características chicas Aquí está también el argumento Y si vemos aquí por ejemplo tenemos varias librerías si nos vamos a la sección de php la que yo estoy utilizando por ejemplo es la de firebase phpjwt pero podríamos utilizar cualquier otra obviamente revisando su documentación entonces de ahí es de donde sale toda esta información que nosotros estamos trabajando en este momento después de eso puedo crear otro método public función y en este caso el método va a ser el método autenticate y también va a

(06:22) traer el modelo user y va a traer un user entonces pues bueno qué le voy a decir lo primero que le voy a decir es que quiero que valide esto por lo tanto le voy a decir this validate Y en este caso va Lo que traiga esta respuesta este rico es y va a validar justamente el email con que tiene que ser de tipo requerido y que debe ser un email y el siguiente es que debe de ser un Y en este caso debe de ser requerido también y listo con esto ya tenemos nuestra validación del autenticate ahora después de haber validado justamente este

(07:02) autenticate pues tenemos varias cosas lo primero que tenemos que hacer es si Existen los datos pues lo siguiente Es que de user vamos a crear un una variable user que va a consultar al modelo user Y tenemos un método web es decir este método viene directamente de este modelo y este método le vamos a decir que el campo que vamos a buscar va a ser el email y que queremos que la información la compare con dis request input de dónde va a obtener del email también acuérdense que todo esto esto que tenemos aquí en realidad es lo que

(07:40) está viniendo de la petición de la petición que tenemos aquí es decir si la petición aquí se llamará correo Pues aquí también le tendría que poner correo y email bien es el que viene de la base de datos entonces una vez que ya tenemos esta parte Pues le decimos que obtenga el primer valor listo con esto en teoría ya podemos empezar a trabajar y ahora le decimos que si user existe si user no existe Entonces le podemos retornar un response una respuesta en formato Jason y le podemos decir que esto sería un error y la respuesta dice sería el email

(08:19) el correo no existe y le podemos pasar otro argumento que en este caso sería el número de estatus que sería el 400 y user aquí Me faltó la variable Perfecto entonces con esto estamos haciendo una guarda justamente para que en dado caso de que no exista el correo pues bueno ya lo podemos trabajar No ahora si el correo si existe pues se pasa a la siguiente sintaxis que sería this request Y en este caso input pass y le vamos a decir que CDs rico es import password sea igual a user password es decir si lo que yo le estoy enviando

(08:57) como password es igual el password que se encuentra dentro de la base de datos entonces que me retorne un response en formato Jason y va a ser un arreglo que va a traer un 200 el estatus code de Esto va a ser un 200 y aquí simplemente le tenemos que decir que el token va a ser igual a this jwt y le mandamos user acuérdense que response en realidad es un método y aquí nos faltó el la asignación perfecto en dado caso de que esto no funcione en cualquiera de los dos pues le vamos a retornar un response en formato Jason y le vamos a decir que

(09:38) Esto va a ser un 400 también pero le vamos a dar aquí que tenemos otro error y en este caso el error es que el correo o el password están Están incorrectos y perfecto ya con esto ya tenemos nuestro out controller generado y terminado Entonces ahora que nos queda Pues nos queda justamente empezar a trabajar con este out controller es decir ya ya definimos nuestro out controller ahora lo que nos tocaría el turno Pues básicamente sería ponerlo dentro de nuestra dentro de nuestras rutas Eso es todo por el momento nos vemos hasta la

(10:19) próxima

Para los JWT token se definen

$router->post('/auth/login', [

    // que use nuestro AuthController

    'uses'=> 'AuthController@authenticate'

]);

Y el auhtcontroller con el metodo authenticate para otorgar el token a request que presenten los credenciales correspondientes.

Luego un middleware, el cual requiere cambios en el arhivo app.php y definimos el jwt middlewre

php artisan make:middleware JWT

Modelos

Recuperacion de datos mediante modelos, esta reuperacion se hara generalmente en los Controllers.

Mediante Eloquents models y Query Builders

## Query Builders <https://laravel.com/docs/12.x/queries#introduction>

Para hacer uso de un query builder debemos hacer uso del facade table y el metodo table, con el nombre de la tabla de la base de satos, este metodo siempre devolvera un Query Builder, al cual se puede encadenar mas restricciones (constraints), al finalizar la cadena debemos hacer uso del metodo get().

Si solo queremos obtener la primera fila de un resultado podemos hacer uso del metodo first() o firstOrFail(), este ultimo devuelve un 404 en caso de no hallar una fila que haga coincidencia.

Esta el metodo where.

Esta el metodo id, para buscar coincidencias en la tabla mediante la columna id.

Agreggates <https://laravel.com/docs/12.x/queries#aggregates>  
  
Como count, max, min, avg and sum.

Determinacion de existencia: <https://laravel.com/docs/12.x/queries#determining-if-records-exist>

Union ALL: <https://laravel.com/docs/12.x/queries#unions>

Where Clauses: <https://laravel.com/docs/12.x/queries#basic-where-clauses>

El primer argumento es el nombre de la columna.   
El segundo argumento es un operador, que puede ser cualquiera de los operadores compatibles con la base de datos.   
El tercer argumento es el valor a comparar con el valor de la columna.

$users = DB::table('users')

->where('votes', '=', 100)

->where('age', '>', 35)

->get();  
—————————-—————————-—————————-—————————-—————For convenience, if you want to verify that a column is = to a given value, you may pass the value as the second argument to the where method. Laravel will assume you would like to use the = operator:

$users = DB::table('users')->where('votes', 100)->get();

Ordenamiento, agrupamiento, Limite y compensacion <https://laravel.com/docs/12.x/queries#ordering-grouping-limit-and-offset>

$users = DB::table('users')

->orderBy('name', 'desc')

->get();  
—————————-—————————-—————————-—————————-—————  
To sort by multiple columns, you may simply invoke orderBy as many times as necessary:

$users = DB::table('users')

->orderBy('name', 'desc')

->orderBy('email', 'asc')

->get();

El ultimo y el mas viuejo <https://laravel.com/docs/12.x/queries#latest-oldest>

Grouping <https://laravel.com/docs/12.x/queries#grouping>

<https://laravel.com/docs/12.x/queries#conditional-clauses>

<https://laravel.com/docs/12.x/queries#insert-statements>