**ESCUELA POLITÉCNICA NACIONAL**

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

**DESARROLLO DE APLICACIONES WEB**

|  |  |
| --- | --- |
|  | |
| ASIGNATURA: | Desarrollo de Aplicaciones Web |
| PROFESOR: | Ing. Loarte Byron |
| PERÍODO ACADÉMICO: |  |
|  | |

**LABORATORIO - 09**

|  |  |
| --- | --- |
|  | |
| TÍTULO:  **RELACIONES POLIMÓRFICAS – SEEDERS - FACTORIES** | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  |  |
|  |  |
| Como instalar Laravel 6 en mi servidor |  |
|  |  |
|  | |

**PROPÓSITO DE LA PRÁCTICA**

Familiarizar al estudiante con el manejo relaciones en Laravel.

**OBJETIVO GENERAL**

Determinar el manejo de relaciones.

**OBJETIVOS ESPECÍFICOS**

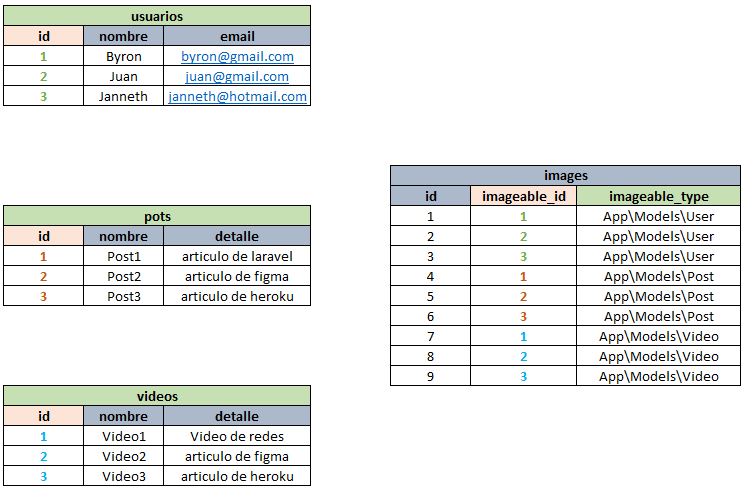
* Configurar las herramientas para el laboratorio.
* Crear un proyecto en Laravel.
* Comprender los archivos y directorios más importantes en Laravel.
* Comprender como funciona las relaciones.
* Comprender como funciona las relaciones polimórficas.
* Comprender como visualizar los resultados de las relaciones en un proyecto.
* Visualizar los resultados obtenidos.

**INSTRUCCIONES**

1. Se procede a trabajar con las siguientes relaciones polimórficas.

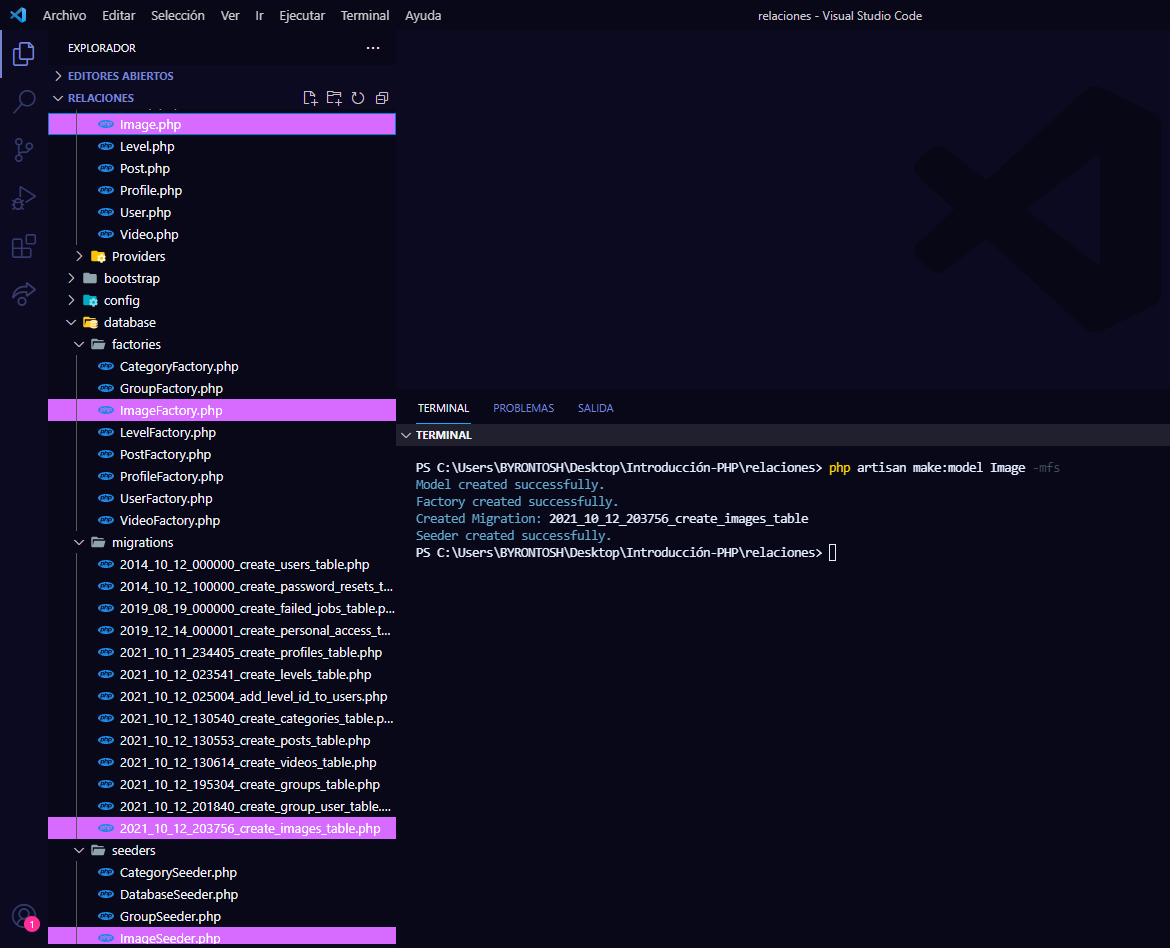
<https://laravel.com/docs/9.x/eloquent-relationships#polymorphic-relationships>

**RELACIÓN POLIMÓRFICA - UNO A UNO**



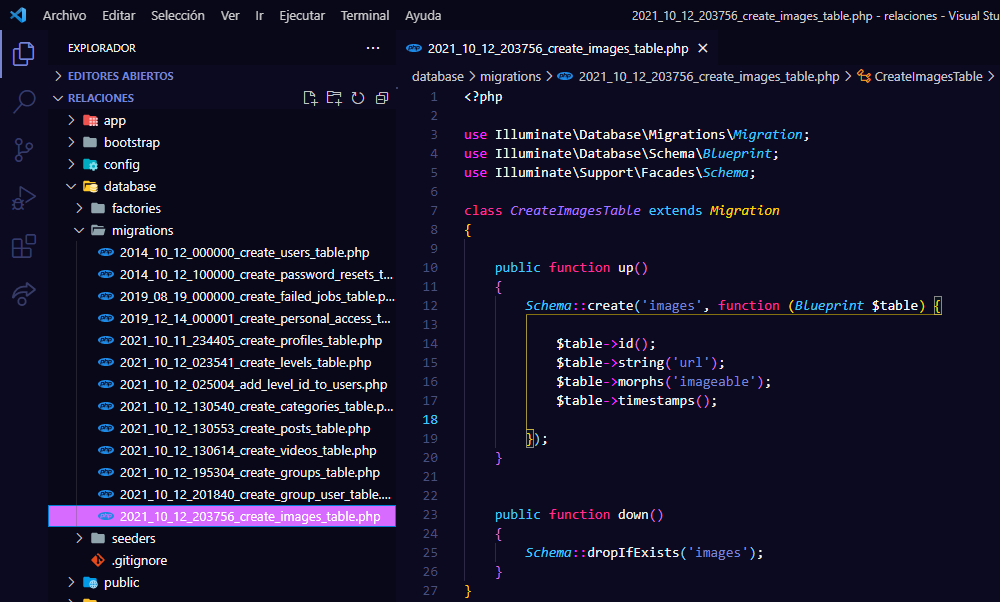
1. Se procede con la creación de un nuevo modelo llamado Image y adicional a ello se crea su migración, factory y sedder con el siguiente comando.

**php artisan make:model Image -mfs**



1. Ahora se procede a definir los campos (nombre de las columnas) para la tabla de la Base de datos, esto se lo realiza en la migración del modelo respectivo, recordando que:

**Los usuarios, posts y videos tienen una imagen**



            $table->id();

            $table->string('url');

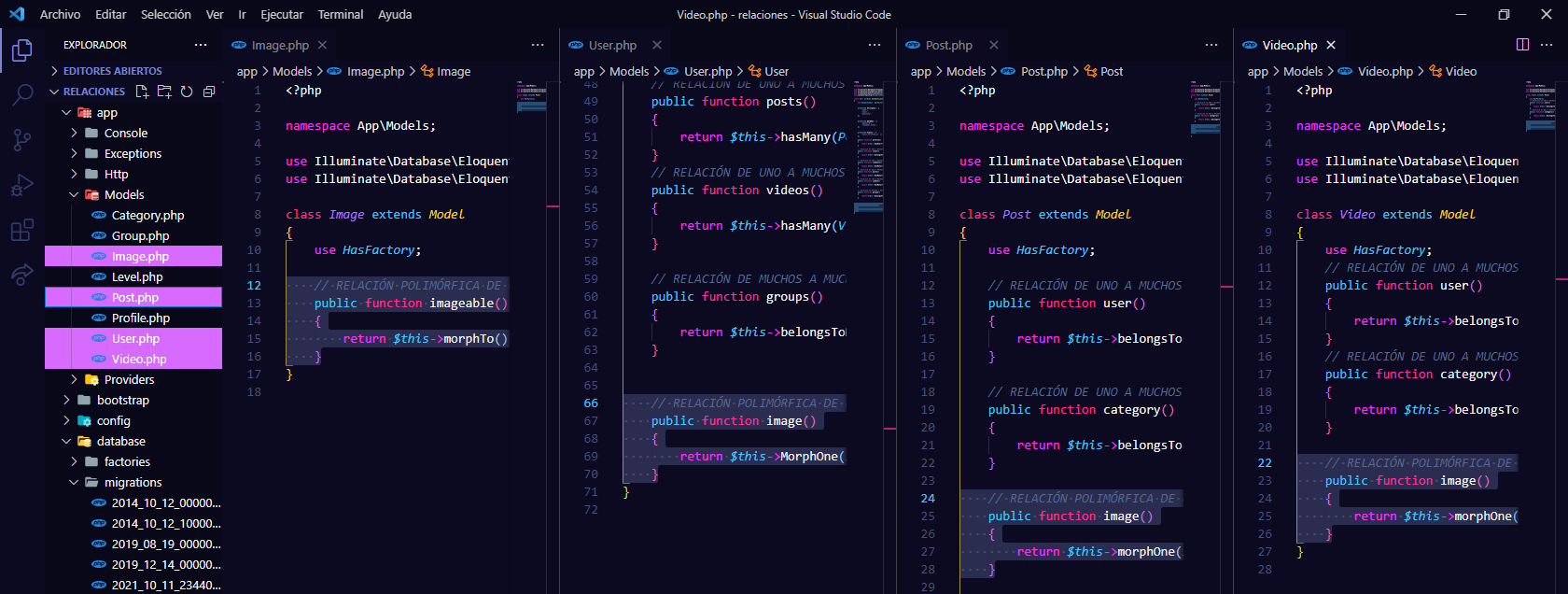
            $table->morphs('imageable');

            $table->timestamps();

1. Ahora se debe hacer el procedimiento a nivel de **Eloquent de Laravel**, teniendo en cuenta que:

**Los usuarios, posts y videos tienen una imagen**

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-one-polymorphic-model-structure>



    // RELACIÓN POLIMÓRFICA DE UNO A UNO

    public function imageable()

    {

        return $this->morphTo();

    }

    // RELACIÓN POLIMÓRFICA DE UNO A UNO

    public function image()

    {

        return $this->MorphOne(Image::class, 'imageable');

    }

    // RELACIÓN POLIMÓRFICA DE UNO A UNO

    public function image()

    {

        return $this->morphOne(Image::class, 'imageable');

    }

    // RELACIÓN POLIMÓRFICA DE UNO A UNO

    public function image()

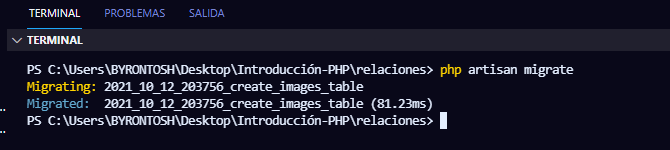
    {

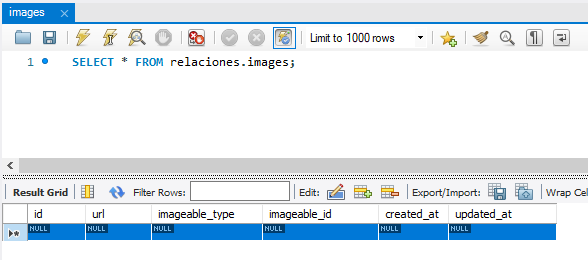
        return $this->morphOne(Image::class,'imageable');

    }

1. Ahora se procede a ejecutar la migración y verificar si realmente esta correcta la implementación de la relación muchos a muchos, ejecutando el siguiente comando.

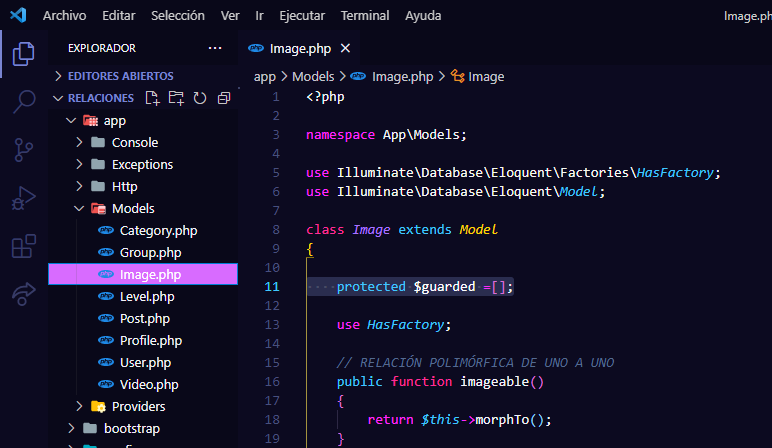
**php artisan migrate**





1. Ahora se debe permitir la asignación masiva en el modelo Image

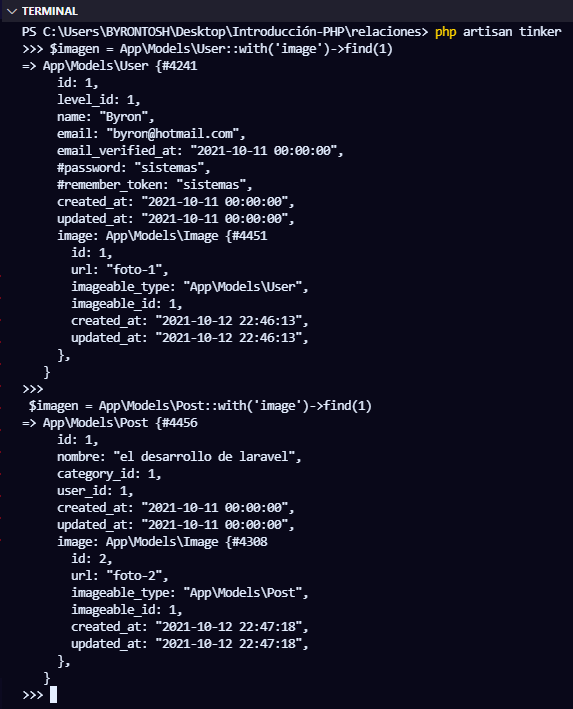
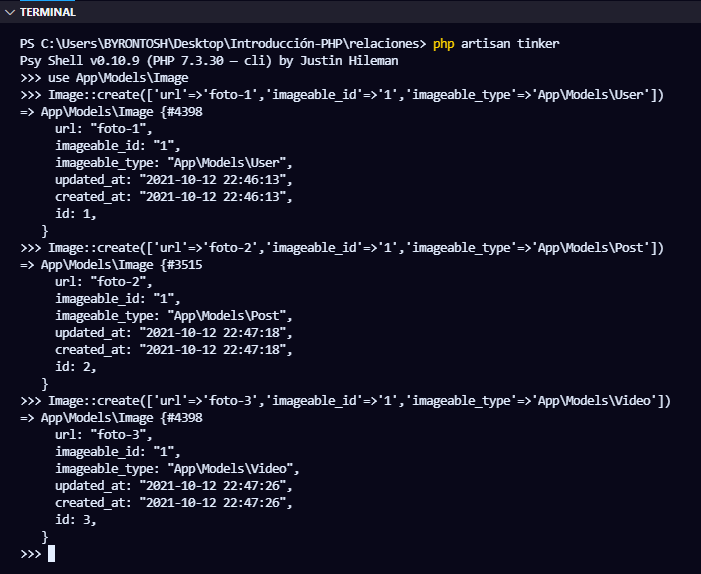
<https://laravel.com/docs/9.x/eloquent#mass-assignment>



1. Para probar se puede verificar ejecutando el siguiente comando

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-one-polymorphic-retrieving-the-relationship>

**php artisan tinker**



$imagen = App\Models\User::with('image')->find(1)

Image::create(['url'=>"foto-1","imageable\_id"=>1,"imageable\_type"=>"App\Models\User"])

$imagen = App\Models\Post::with('image')->find(1)

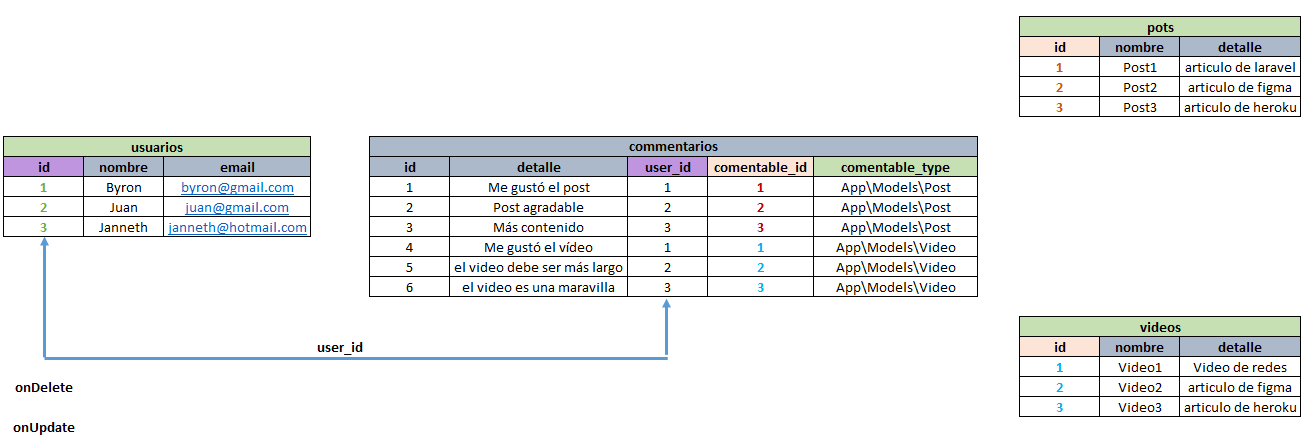
Image::create(['url'=>"foto-2","imageable\_id"=>1,"imageable\_type"=>"App\Models\Post"])

Image::create(['url'=>"foto-3","imageable\_id"=>1,"imageable\_type"=>"App\Models\Video"])

1. Ahora se procede a trabajar con la siguiente relación.

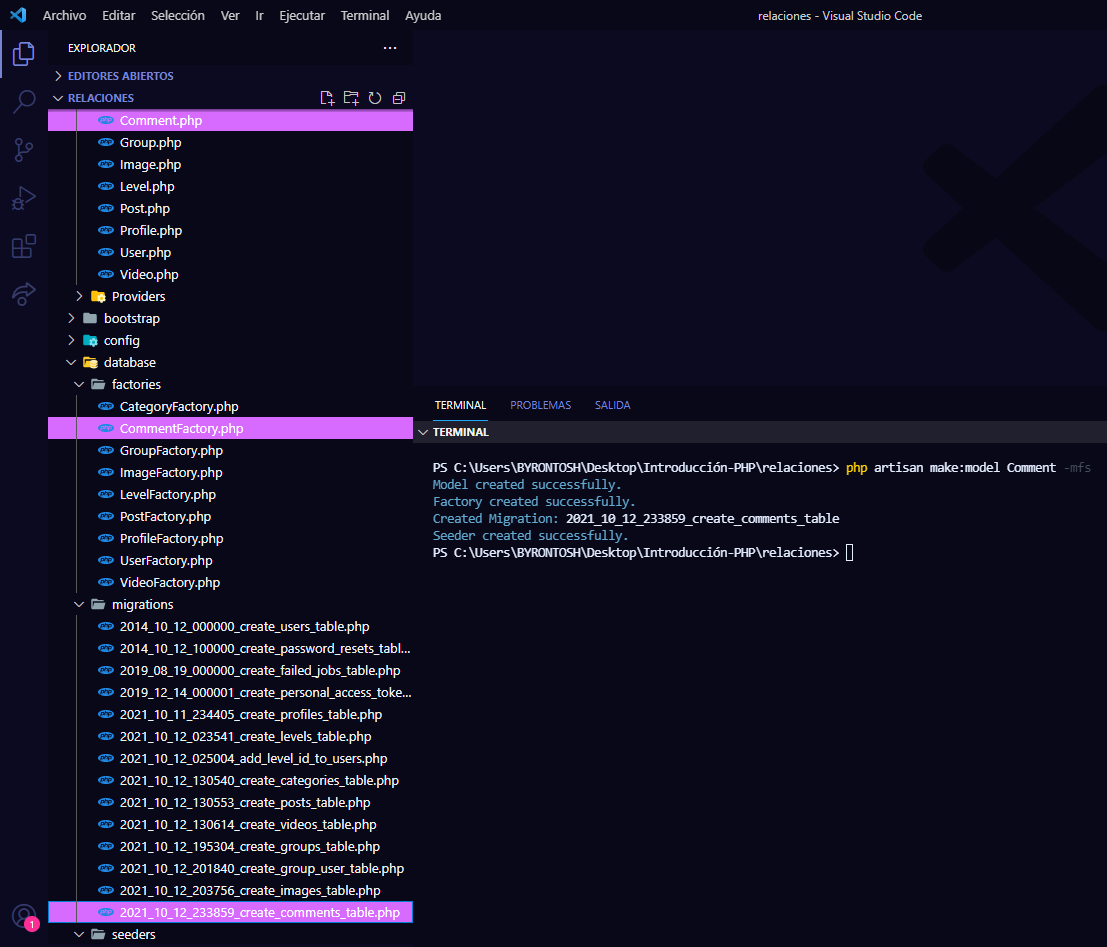
**RELACIÓN POLIMÓRFICA - UNO A MUCHOS**

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-many-polymorphic-relations>



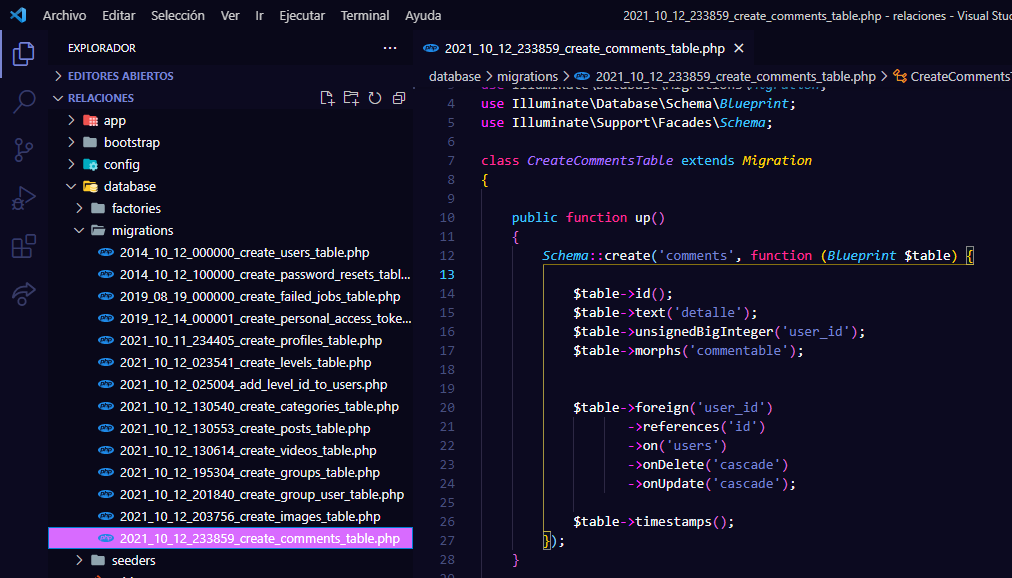
1. Se procede con la creación de un nuevo modelo llamado Comment y adicional a ello se crea su migración, factory y sedder con el siguiente comando.

**php artisan make:model Comment -mfs**



1. Ahora se procede a definir los campos (nombre de las columnas) para la tabla de la base de datos, esto se lo realiza en la migración del modelo respectivo, recordando que:

**Un usuario puede hacer muchos comentarios (post o video), pero un comentario le pertenece a un usuario**



            $table->id();

            $table->text('detalle');

            $table->unsignedBigInteger('user\_id');

            $table->morphs('commentable');

            $table->foreign('user\_id')

                   ->references('id')

                   ->on('users')

                   ->onDelete('cascade')

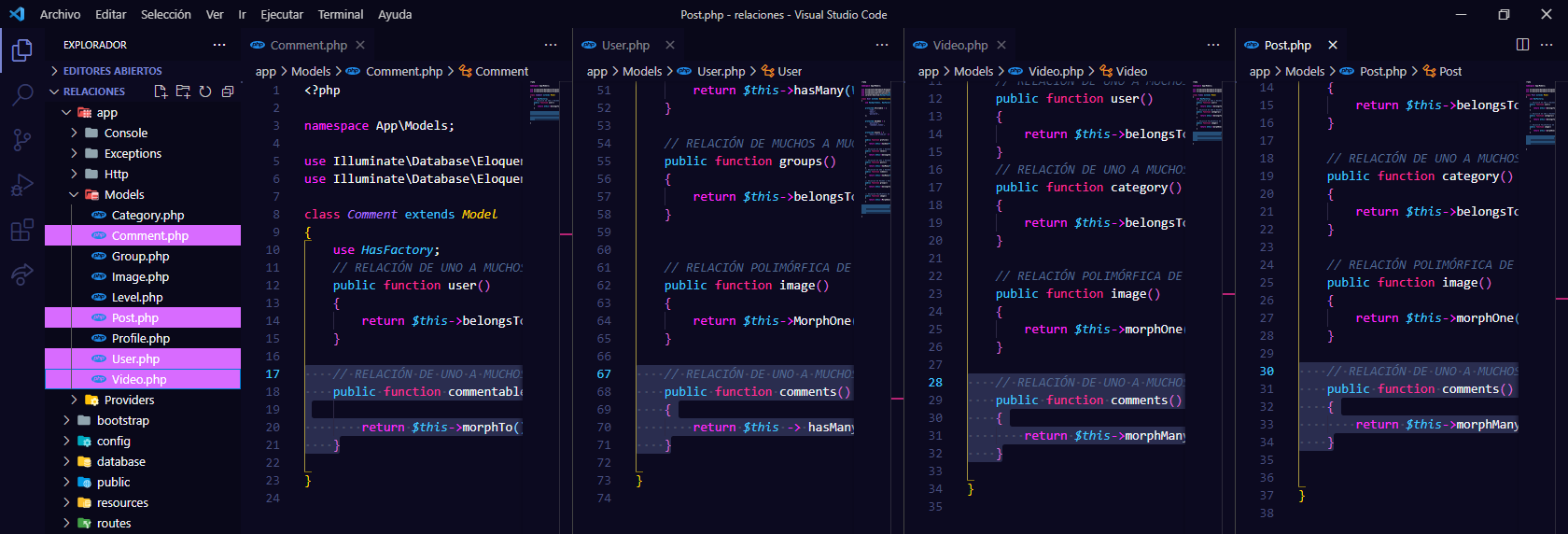
                   ->onUpdate('cascade');

            $table->timestamps();

1. Ahora se debe hacer el procedimiento a nivel de **Eloquent de Laravel**, teniendo en cuenta que:

<https://laravel.com/docs/9.x/eloquent-relationships#one-to-many-polymorphic-model-structure>

**Un usuario puede hacer muchos comentarios (post o video), pero un comentario le pertenece a un usuario**



    // RELACIÓN DE UNO A MUCHOS

    public function user()

    {

        return $this->belongsTo(User::class);

    }

    // RELACIÓN DE UNO A MUCHOS POLIMÓRIFCA

    public function commentable(){

        return $this->morphTo();

    }

    // RELACIÓN DE UNO A MUCHOS

    public function comments()

    {

        return $this->hasMany(Comment::class);

    }

    // RELACIÓN DE UNO A MUCHOS POLIMÓRIFCA

    public function comments()

    {

        return $this->morphMany(Comment::class,'commentable');

    }

    // RELACIÓN DE UNO A MUCHOS POLIMÓRIFCA

    public function comments()

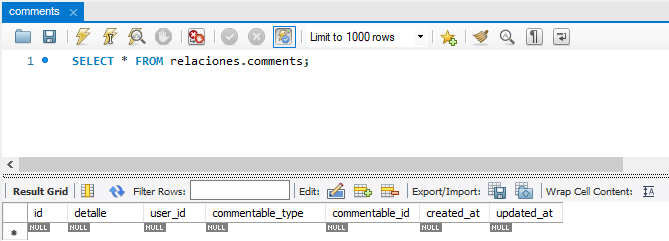
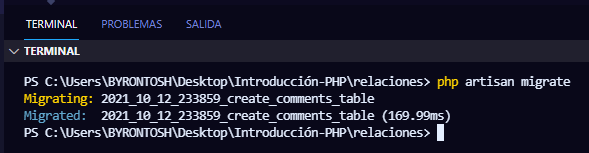
    {

        return $this->morphMany(Comment::class, 'commentable');

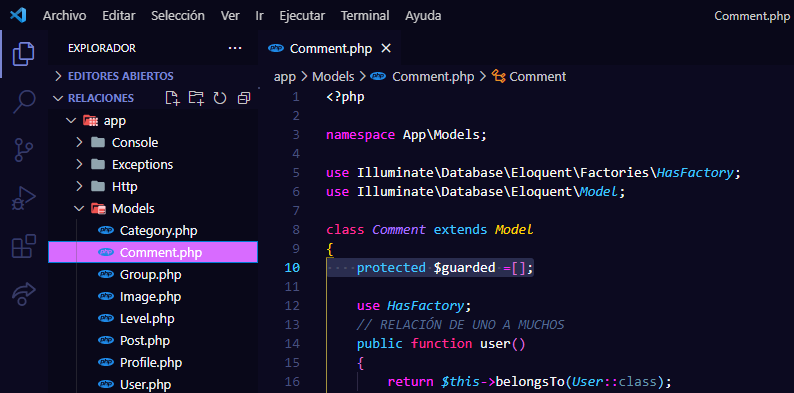
    }

1. Ahora se procede a ejecutar la migración y verificar si realmente esta correcta la implementación de la relación muchos a muchos, ejecutando el siguiente comando.

**php artisan migrate**



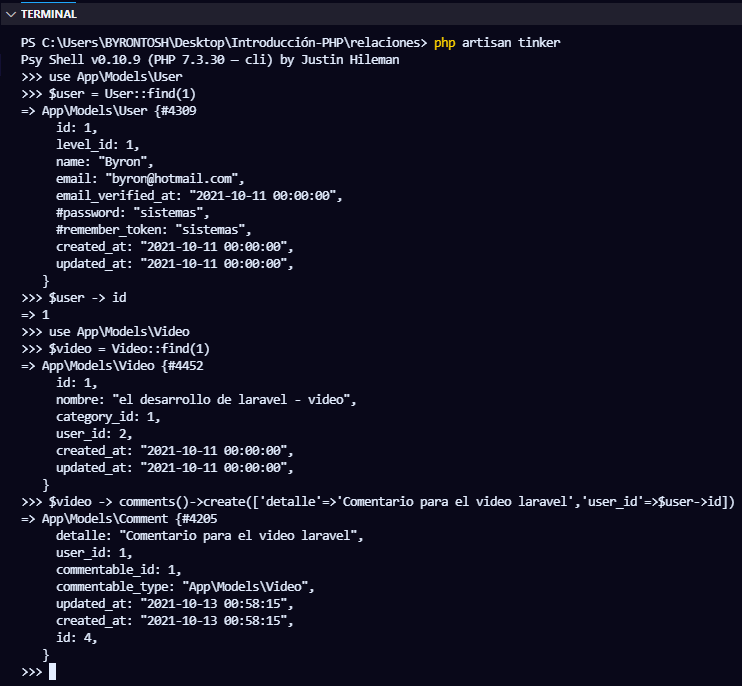
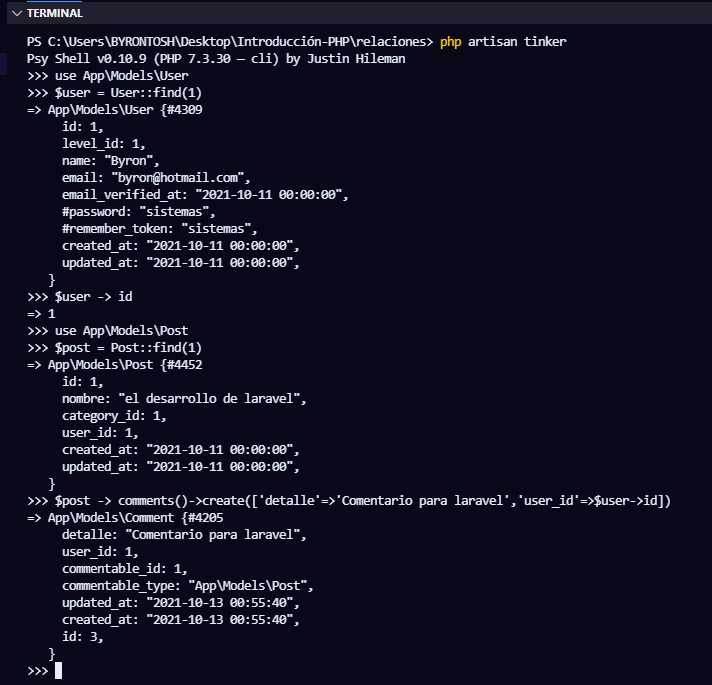
1. Ahora se debe permitir la asignación masiva en el modelo Comment.



1. Para probar se puede verificar ejecutando

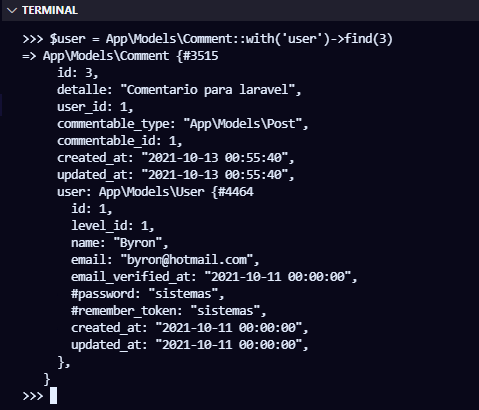
<https://laravel.com/docs/9.x/eloquent-relationships#one-to-many-polymorphic-retrieving-the-relationship>

**php artisan tinker**



$video -> comments()->create(['detalle'=>"Me gustó el video","user\_id"=>1])

$post -> comments()->create(['detalle'=>"Me gustó el post","user\_id"=>1])

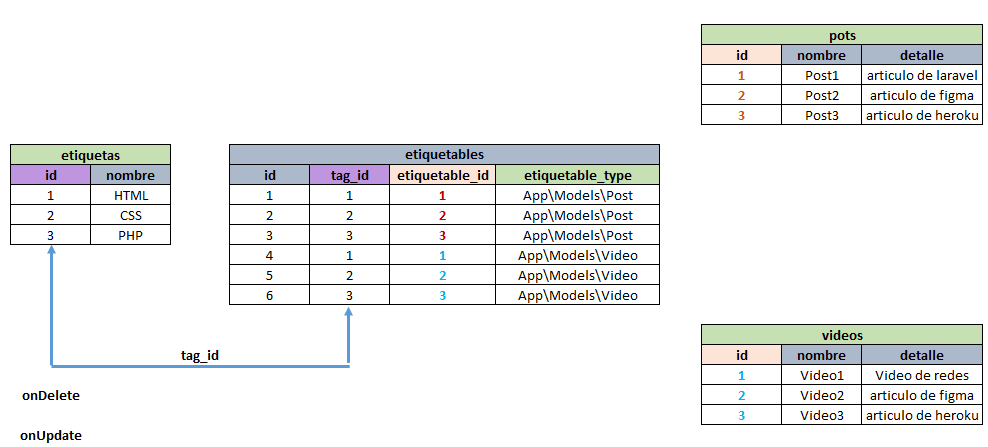


$user = App\Models\Comment::with('user')->find(99)

1. Ahora se procede a trabajar con las relaciones.

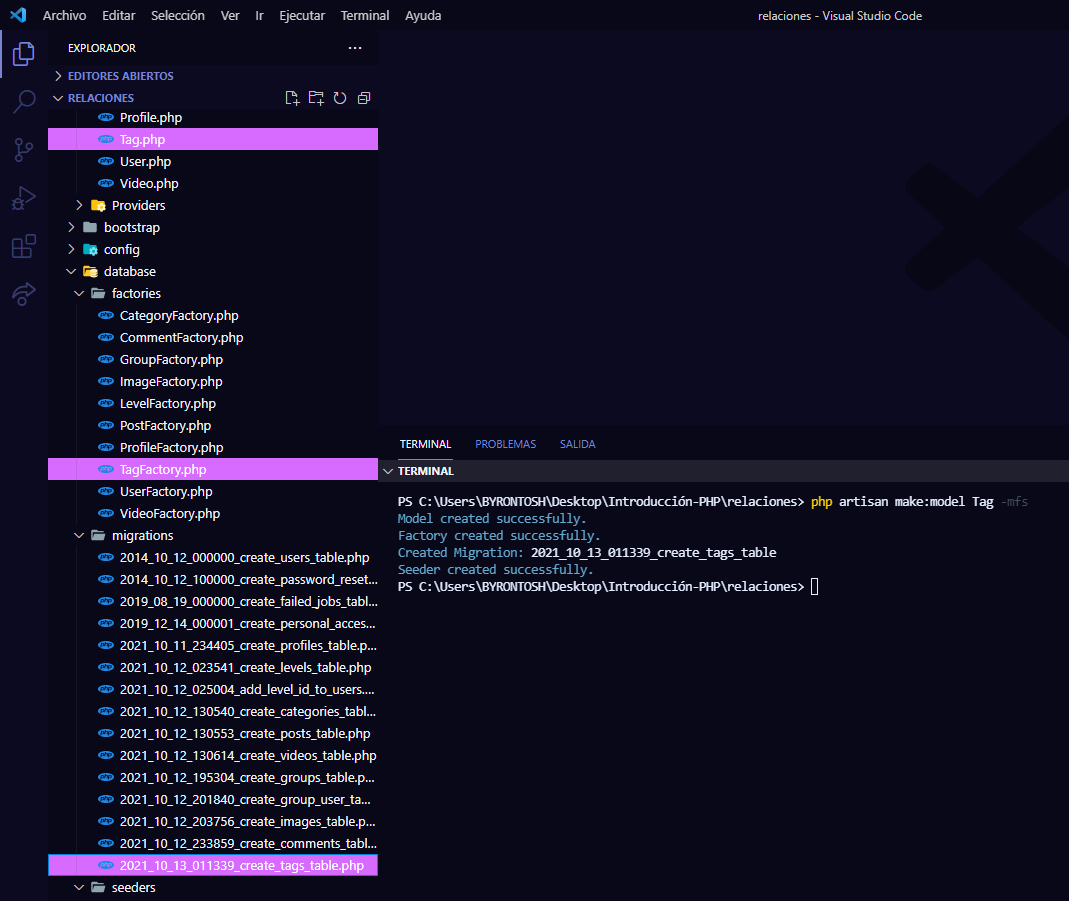
<https://laravel.com/docs/9.x/eloquent-relationships#many-to-many-polymorphic-relations>

**RELACIÓN POLIMÓRFICA - MUCHOS A MUCHOS**



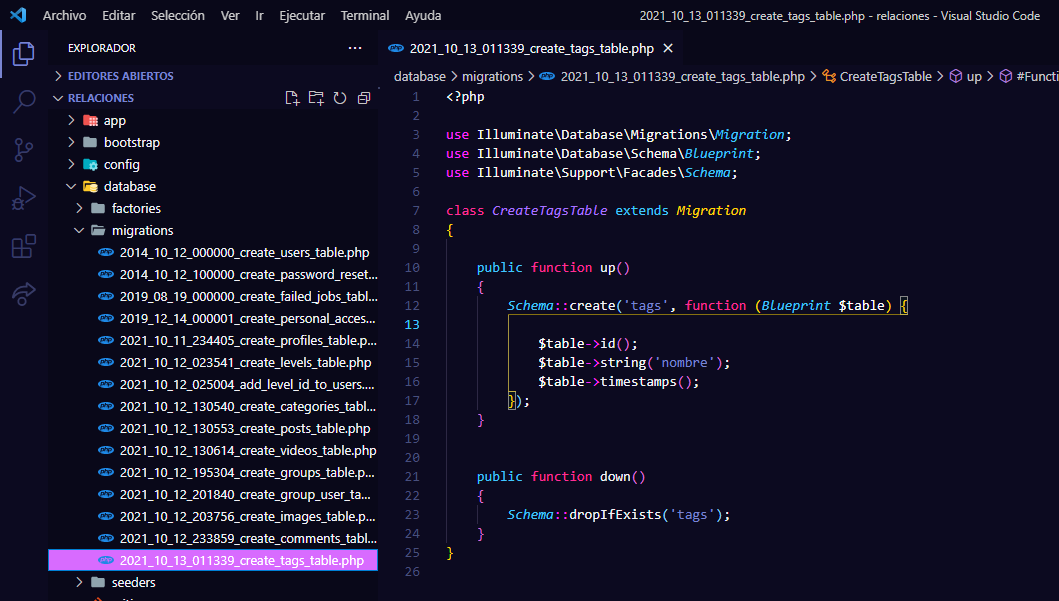
1. Se procede con la creación de un nuevo modelo llamado Tag y adicional a ello se crea su migración, factory y sedder con el siguiente comando.

**php artisan make:model Tag -mfs**



1. Ahora se procede a definir los campos (nombre de las columnas) para la tabla de la base de datos, esto se lo realiza en la migración del modelo respectivo, recordando que:

**Un post y/o video puede tener muchas etiquetas y que una etiqueta puede estar en muchos posts y/o videos**



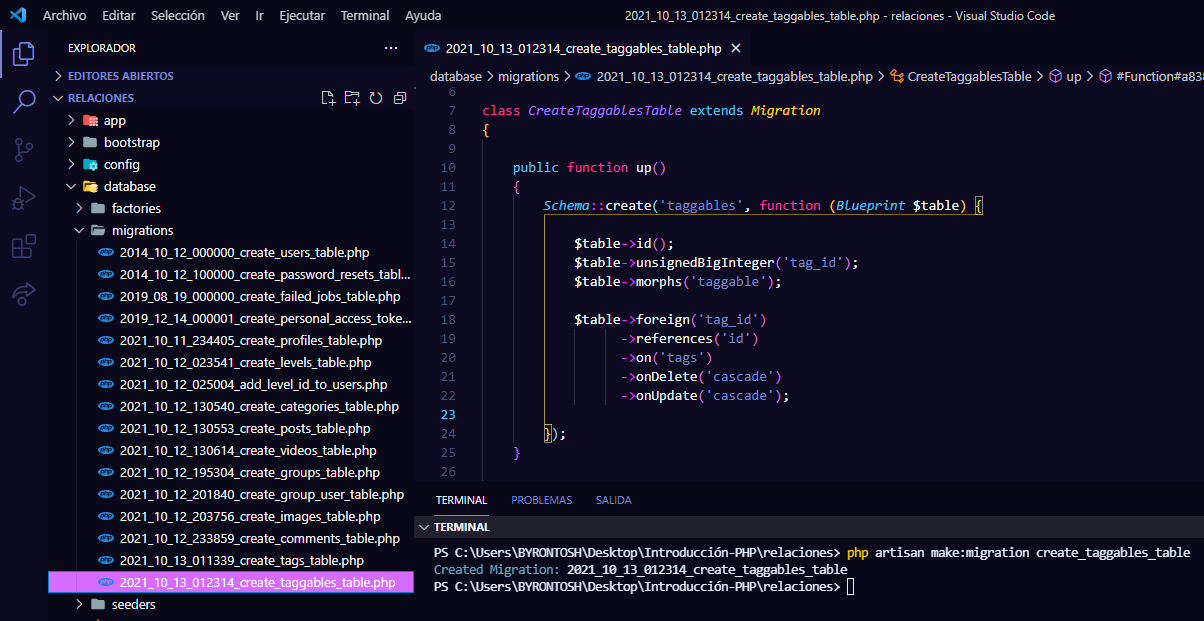
            $table->id();

            $table->string('nombre');

            $table->timestamps();

1. No obstante, hay que tener en cuenta que si se trata de una relación de muchos a muchos se debe trabajar con una tabla intermedia o tabla pivote que haga referencia a las dos tablas que son videos y posts. Para lo cual se debe crear una nueva migración, ejecutando el siguiente comando:

**php artisan make:migration create\_taggables\_table**



            $table->id();

            $table->unsignedBigInteger('tag\_id');

            $table->morphs('taggable');

            $table->foreign('tag\_id')

                  ->references('id')

                  ->on('tags')

                  ->onDelete('cascade')

                  ->onUpdate('cascade');

1. Ahora se debe hacer el procedimiento a nivel de **Eloquent de Laravel**, teniendo en cuenta que:

<https://laravel.com/docs/9.x/eloquent-relationships#many-to-many-polymorphic-model-structure>

**Un post y/o video puede tener muchas etiquetas y que una etiqueta puede estar en muchos posts y/o videos**



    // RELACIÓN DE MUCHOS A MUCHOS POLIMÓRIFCA

    public function videos()

    {

        return $this->morphedByMany(Video::class,'taggable');

    }

    // RELACIÓN DE MUCHOS A MUCHOS POLIMÓRIFCA

    public function posts()

    {

        return $this->morphedByMany(Post::class,'taggable');

    }

    // RELACIÓN DE MUCHOS A MUCHOS POLIMÓRIFCA

    public function tags()

    {

        return $this->morphToMany(Tag::class,'taggable');

    }

    // RELACIÓN DE MUCHOS A MUCHOS POLIMÓRIFCA

    public function tags()

    {

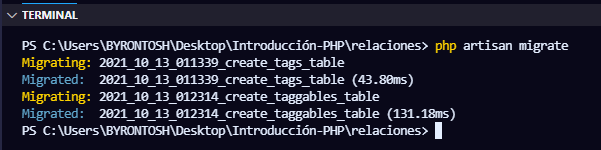
        return $this->morphToMany(Tag::class, 'taggable');

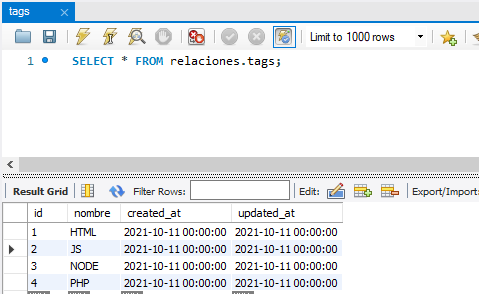
    }

1. Ahora se procede a ejecutar la migración y verificar si realmente esta correcta la implementación de la relación muchos a muchos, ejecutando el siguiente comando.

**php artisan migrate**

[https://laravel.com/docs/9.x/eloquent-relationships#many-to-many-polymorphic-retrieving-the-relationship](https://laravel.com/docs/9.x/eloquent-relationships" \l "many-to-many-polymorphic-retrieving-the-relationship)





INSERT INTO `relaciones`.`tags` (`id`, `nombre`, `created\_at`, `updated\_at`) VALUES ('1', 'HTML', '2021-10-11 00:00:00', '2021-10-11 00:00:00');

INSERT INTO `relaciones`.`tags` (`id`, `nombre`, `created\_at`, `updated\_at`) VALUES ('2', 'JS', '2021-10-11 00:00:00', '2021-10-11 00:00:00');

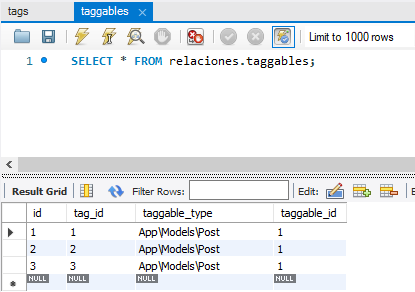
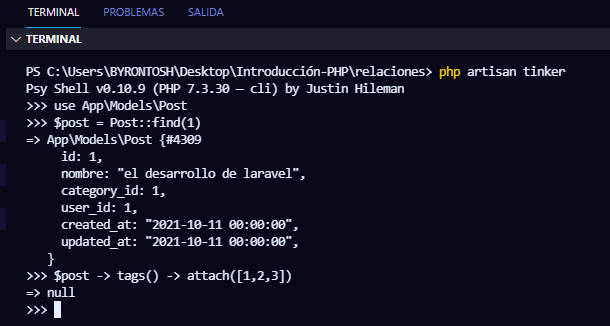
INSERT INTO `relaciones`.`tags` (`id`, `nombre`, `created\_at`, `updated\_at`) VALUES ('3', 'NODE', '2021-10-11 00:00:00', '2021-10-11 00:00:00');

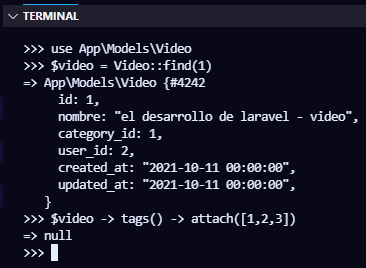
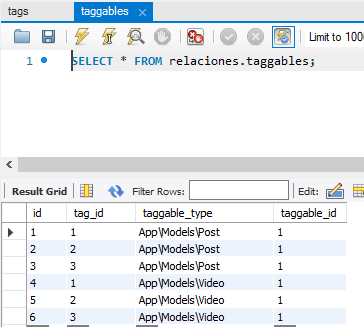
INSERT INTO `relaciones`.`tags` (`id`, `nombre`, `created\_at`, `updated\_at`) VALUES ('4', 'PHP', '2021-10-11 00:00:00', '2021-10-11 00:00:00');

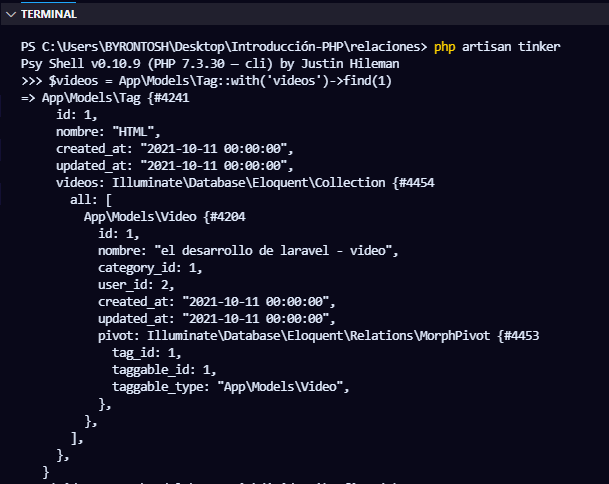
1. Para probar se puede verificar ejecutando

**php artisan tinker**

[https://laravel.com/docs/9.x/eloquent-relationships#updating-many-to-many-relationships](https://laravel.com/docs/9.x/eloquent-relationships" \l "updating-many-to-many-relationships)

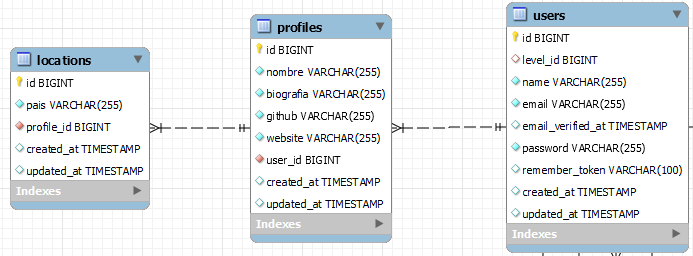




1. Ahora se procede a trabajar con las relaciones

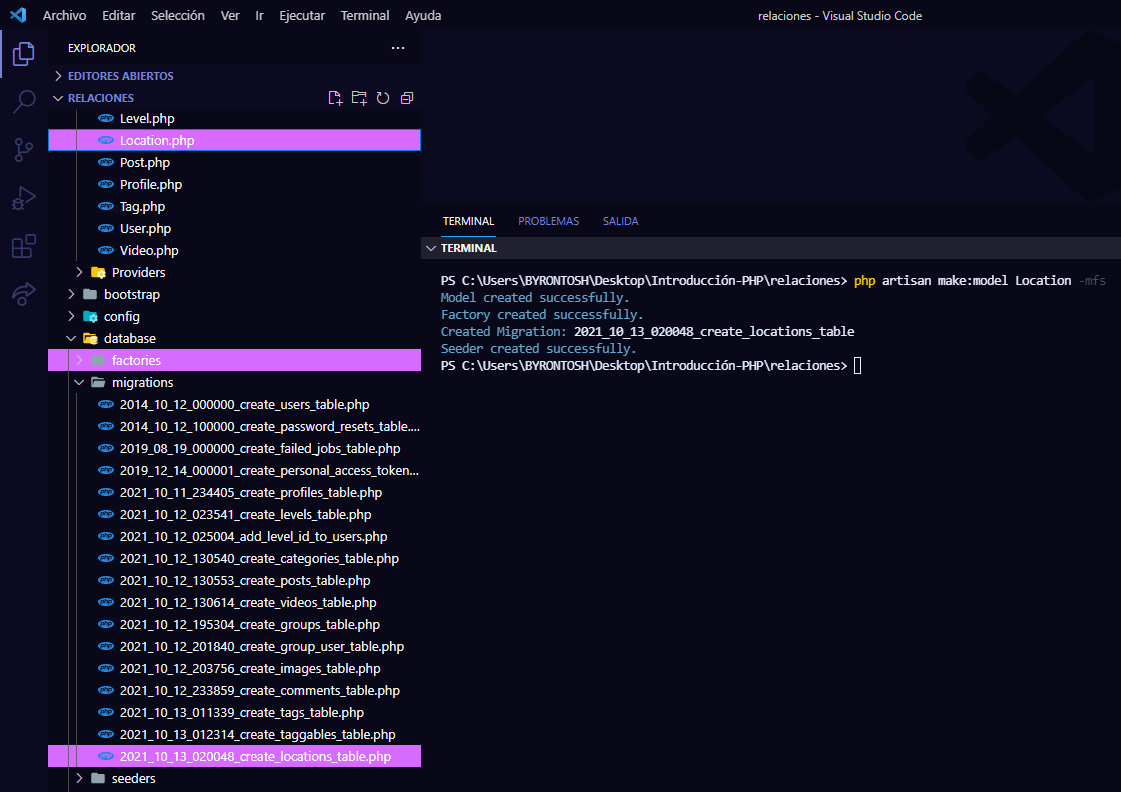
**RELACIÓN DE UNO A UNO A TRAVÉS DE**



<https://laravel.com/docs/9.x/eloquent-relationships#has-one-through>

1. Se procede con la creación de un nuevo modelo llamado Location y adicional a ello se crea su migración, factory y sedder con el siguiente comando.

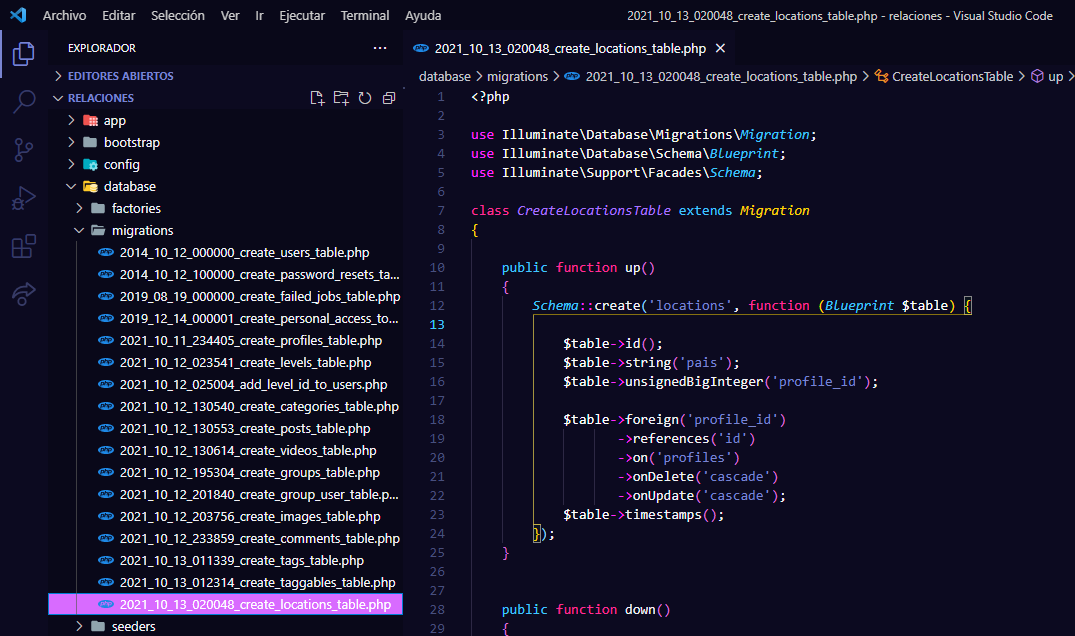
**php artisan make:model Location -mfs**



1. Ahora se procede a definir los campos (nombre de las columnas) para la tabla de la base de datos, esto se lo realiza en la migración del modelo respectivo, recordando que:

**Recordar que un usuario a través de un perfil tiene una localización**

1. Con este procedimiento se realizó la relación a nivel de tablas de la **Base de Datos** únicamente.



            $table->id();

            $table->string('pais');

            $table->unsignedBigInteger('profile\_id');

            $table->foreign('profile\_id')

                   ->references('id')

                   ->on('profiles')

                   ->onDelete('cascade')

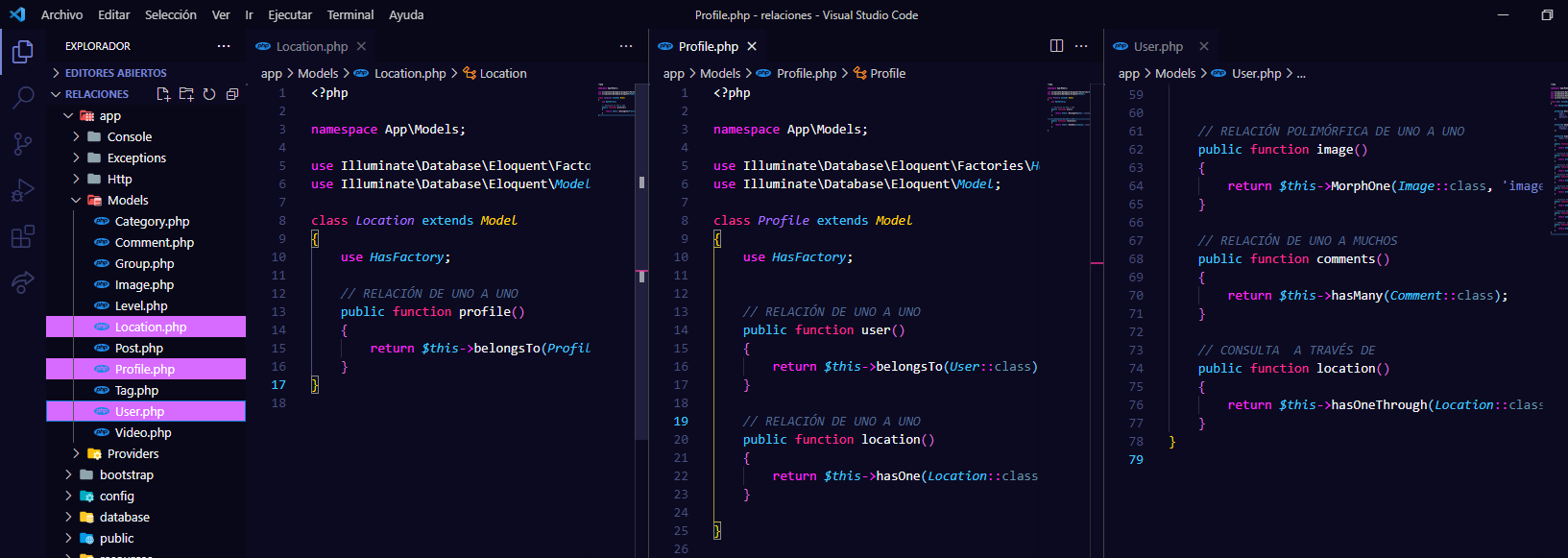
                   ->onUpdate('cascade');

            $table->timestamps();

1. Ahora se debe hacer el procedimiento a nivel de Eloquent de Laravel, teniendo en cuenta que:

<https://laravel.com/docs/9.x/eloquent-relationships#has-one-through>

**Recordar que un usuario a través de un perfil tiene una localización**



    // RELACIÓN DE UNO A UNO

    public function profile()

    {

        return $this->belongsTo(Profile::class);

    }

    // RELACIÓN DE UNO A UNO

    public function location()

    {

        return $this->hasOne(Location::class);

    }

    // CONSULTA  A TRAVÉS DE

    public function location()

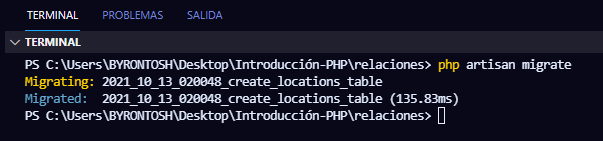
    {

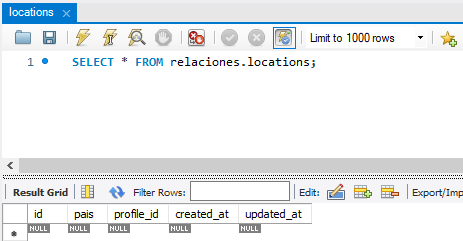
        return $this->hasOneThrough(Location::class, Profile::class);

    }

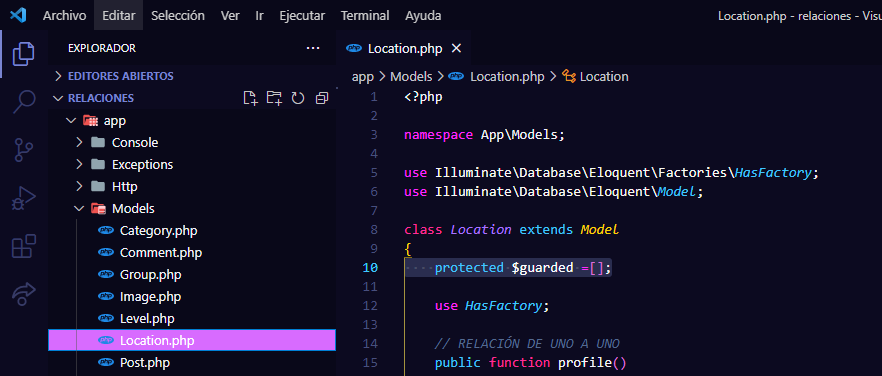
1. Ahora se procede a ejecutar la migración y verificar si realmente esta correcta la implementación de la relación uno a uno, ejecutando el siguiente comando.

**php artisan migrate**



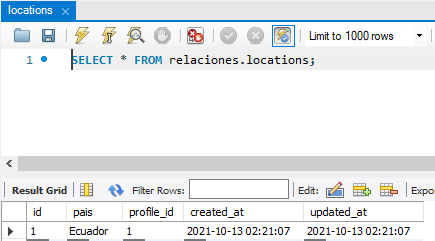
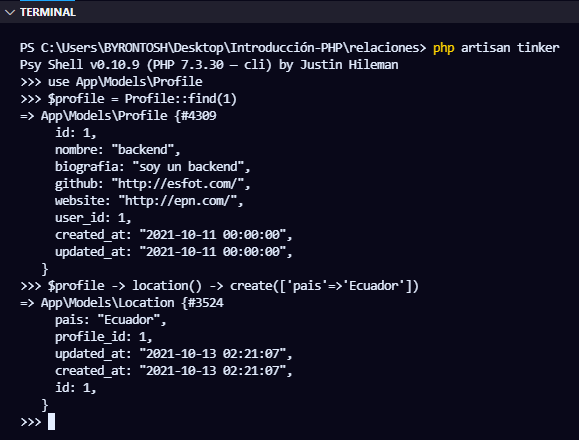


1. Ahora se debe permitir la asignación masiva en el modelo Location.

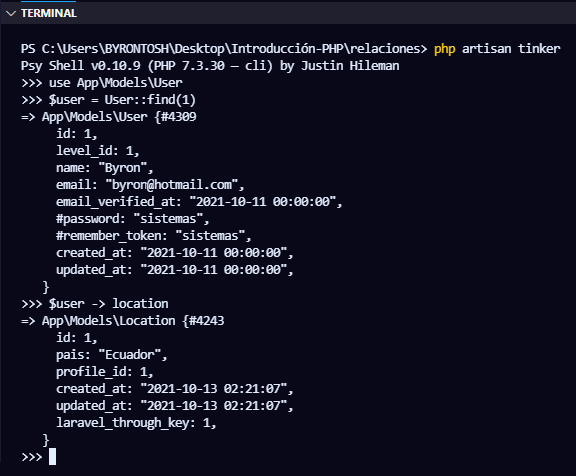


1. Para probar se puede verificar ejecutando.

**php artisan tinker**



$profile -> location() -> create(['pais'=>'Ecuador'])



**SEEDERS Y FACTORIES**

[https://laravel.com/docs/9.x/seeding#introduction](https://laravel.com/docs/9.x/seeding" \l "introduction)

1. Para lo cual se procede a ejecutar el siguiente comando para restablecer las migraciones y vaciar los datos de la Base de Datos

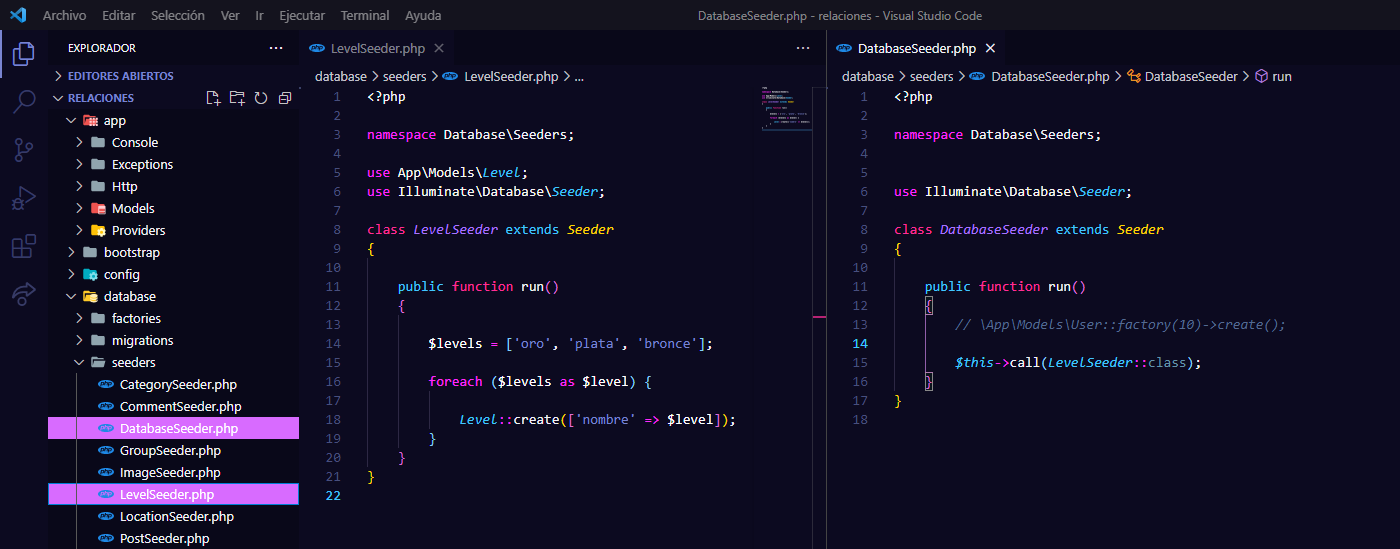
**php artisan migrate:fresh**



1. Ahora se procede a trabajar con el seeder del modelo Level para llenar con datos mios que sean propios.

<https://laravel.com/docs/9.x/seeding#writing-seeders>

<https://laravel.com/docs/9.x/seeding#calling-additional-seeders>



        $levels = ['oro', 'plata', 'bronce'];

        foreach ($levels as $level) {

            Level::create(['nombre' => $level]);

        }

        $this->call(

            [

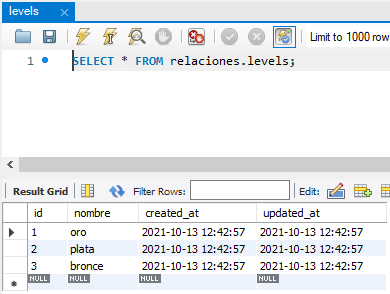
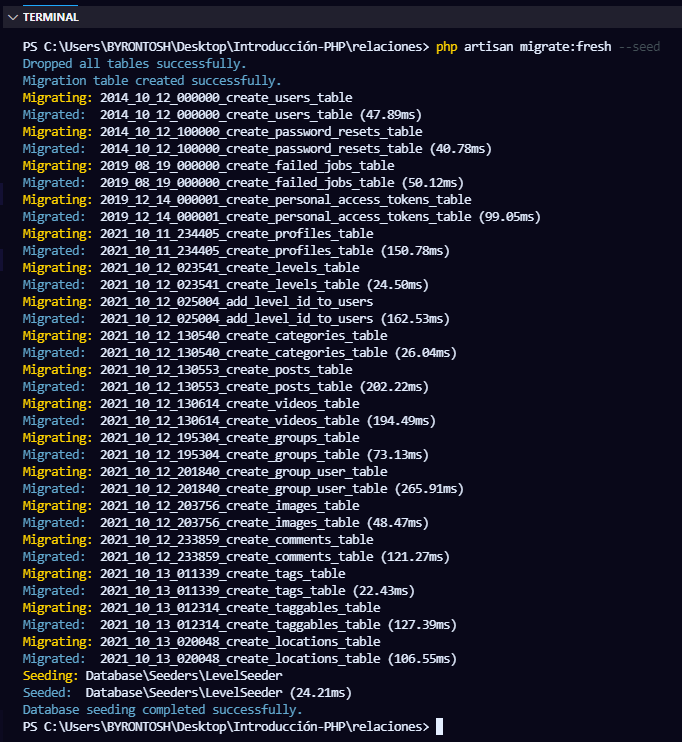
                LevelSeeder::class,

            ]);

1. Luego se ejecuta el siguiente comando para restablecer las migraciones y ejecutar los seeders creados.

<https://laravel.com/docs/9.x/seeding#running-seeders>

**php artisan migrate:fresh --seed**

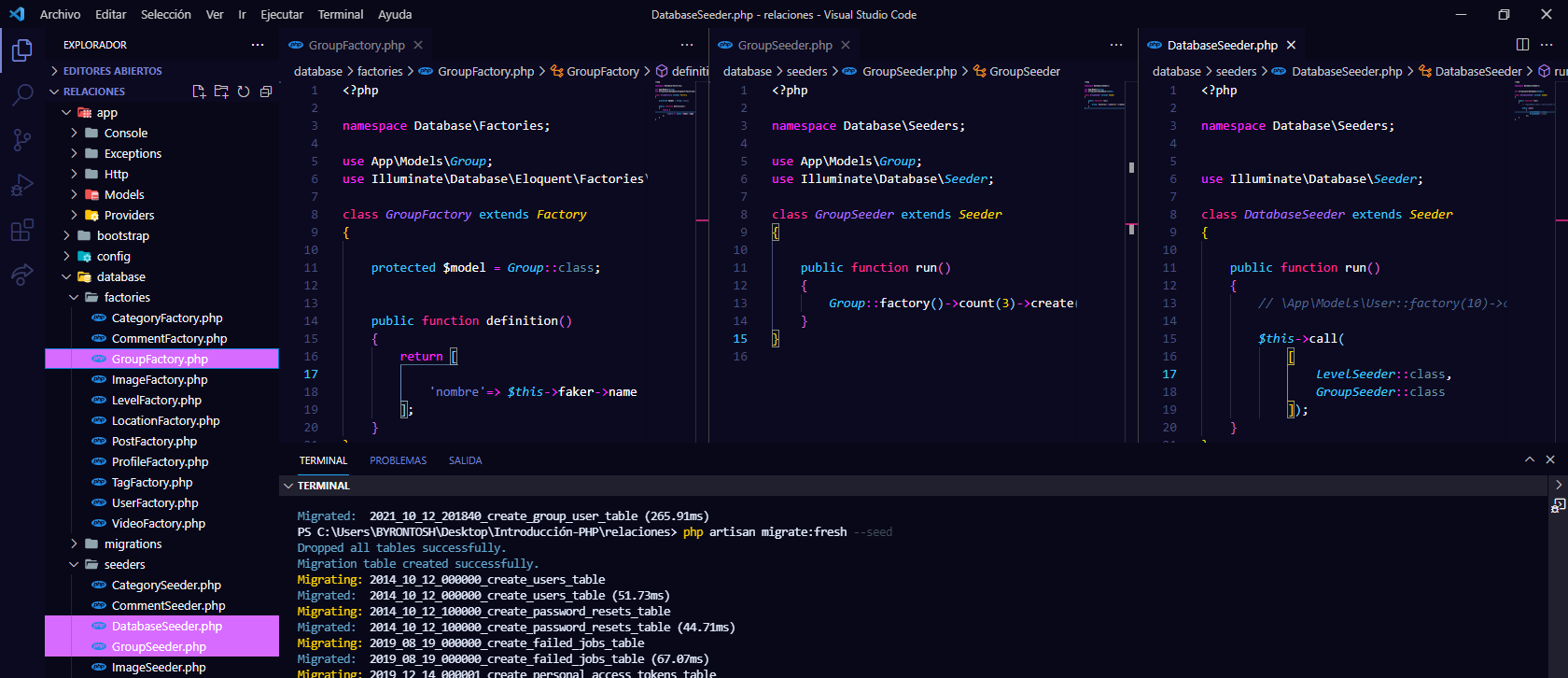


1. Ahora se procede a trabajar con datos fake para llenar las demás tablas de la Base de Datos, para lo cual se procede a trabajar con el factory de Group y luego con el seeder e invocarlo desde DatabaseSeeder.

<https://laravel.com/docs/9.x/database-testing#concept-overview>

<https://github.com/FakerPHP/Faker>

<https://laravel.com/docs/9.x/database-testing#instantiating-models>



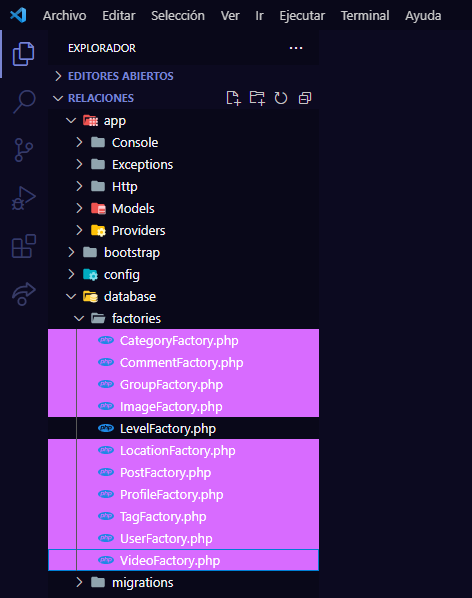
'nombre'=> $this->faker->name

Group::factory()->count(3)->create();

GroupSeeder::class,



1. Ahora se procede a trabajar con todos los Factories restantes para establecer datos faker.



'nombre'=> $this->faker->name

            'detalle'=> $this->faker->sentence,

            'user\_id'=> rand(1,5)

'url'=> $this->faker->imageUrl(640,480)

'pais'=> $this->faker->country

            'nombre'=> $this->faker->sentence,

            'category\_id'=> rand(1,4),

            'user\_id'=> rand(1,5)

            'nombre'=> $this->faker->name,

            'biografia'=> $this->faker->sentence,

            'github'=> $this->faker->userName,

            'website'=> $this->faker->url,

            'nombre'=> $this->faker->name

            'level\_id' => rand(1,3),

            'name' => $this->faker->name(),

            'email' => $this->faker->unique()->safeEmail(),

            'email\_verified\_at' => now(),

            'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password

            'remember\_token' => Str::random(10),

            'nombre'=> $this->faker->name,

            'category\_id'=> rand(1,4),

            'user\_id'=> rand(1,5)

1. Ahora se procede a trabajar con los Seeeder respectivos (Category y Tag), luego llamarlos en el DatabaseSeeder y por último ejecutar el comando respectivo para verificar la inserción de datos.

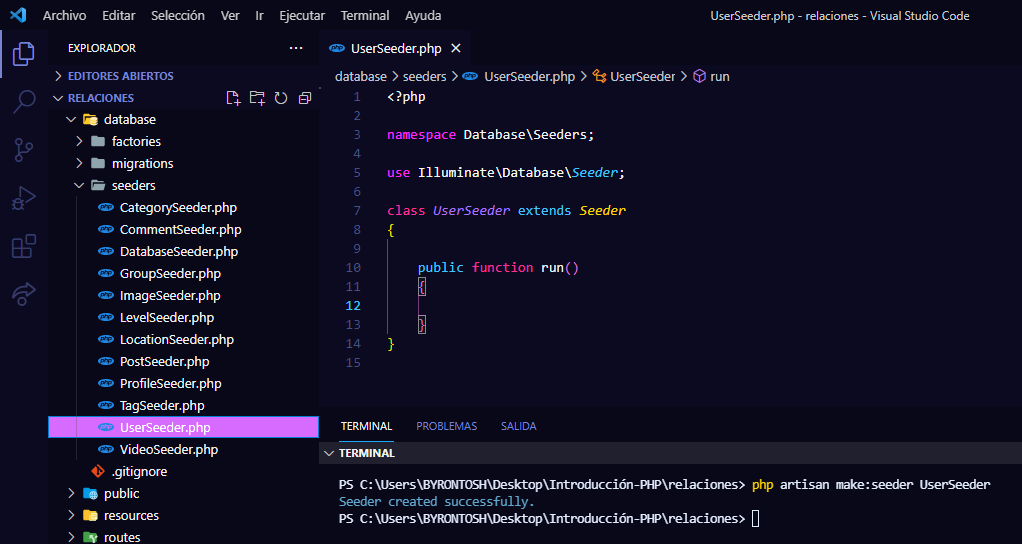
       Category::factory()->count(4)->create();

        Tag::factory()->count(5)->create();

En ese sentido, se procede con la creación de un nuevo seeder para trabajar sobre él, ejecutando el siguiente comando.

<https://laravel.com/docs/9.x/seeding#writing-seeders>

**php artisan make:seeder UserSeeder**



1. Ahora se procede a trabajar el seeder creado.

<https://laravel.com/docs/9.x/collections#method-each>

<https://laravel.com/docs/9.x/database-testing#creating-models-using-factories>

<https://laravel.com/docs/9.x/eloquent#inserting-and-updating-models>



        User::factory()->count(5)->create();

        $users = User::all();

        $users->each(function ($user)

        {

            $user->profile()->save(Profile::factory()->make());

        });

        $profiles = Profile::all();

        $profiles->each(function ($profile)

        {

            $profile->location()->save(Location::factory()->make());

        });

        $users->each(function ($user)

        {

            $user->image()->save(Image::factory()->make(['url' => "https://picsum.photos/id/$user->id/100/100"]));

        });

        $users->each(function ($user)

        {

            $group = rand(1, 3);

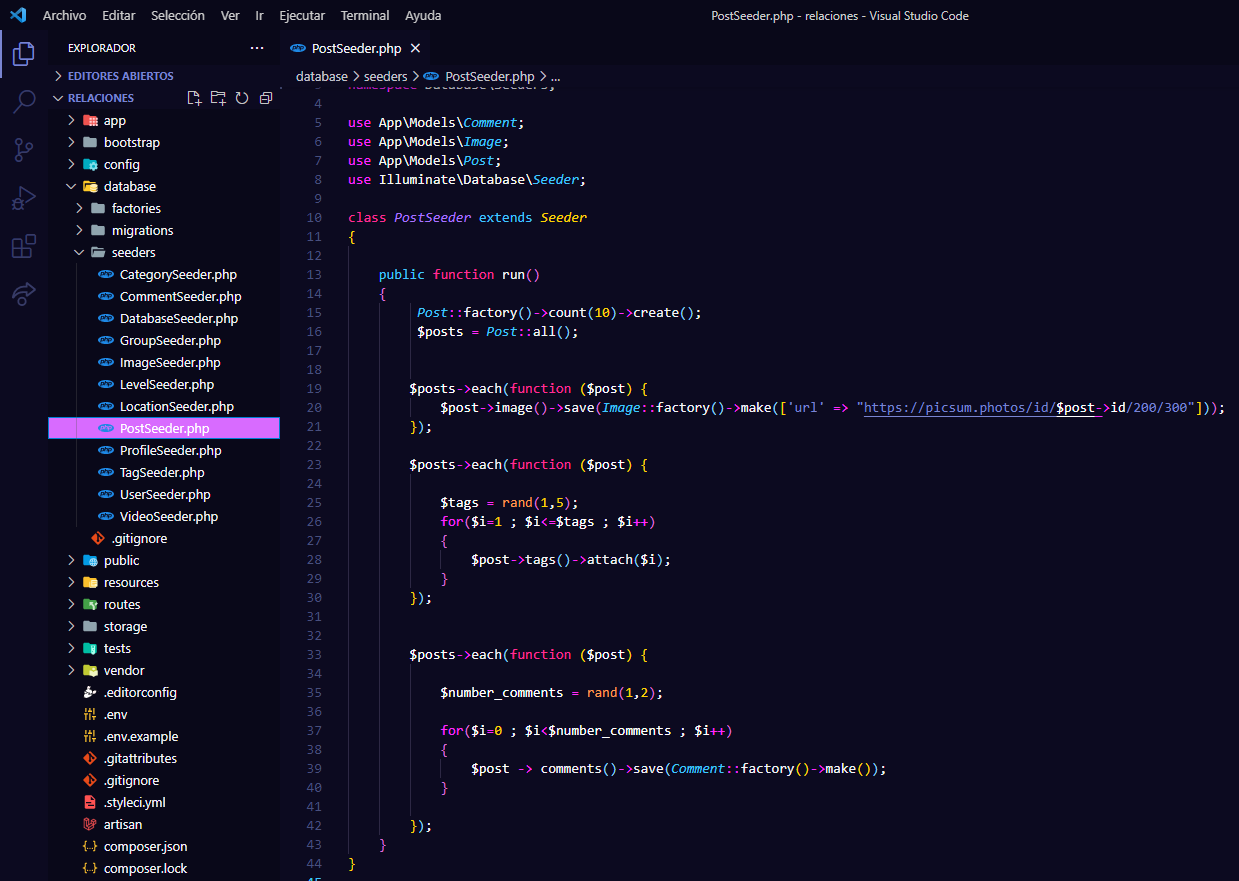
            for ($i = 1; $i <= $group; $i++) {

                $user->groups()->attach($i);

            }

        });

1. Luego el con el seeder de post.



        Post::factory()->count(10)->create();

        $posts = Post::all();

       $posts->each(function ($post)

       {

           $post->image()->save(Image::factory()->make(['url' => "https://picsum.photos/id/$post->id/150/150"]));

       });

       $posts->each(function ($post)

       {

           $tags = rand(1,5);

           for($i=1 ; $i<=$tags ; $i++)

           {

               $post->tags()->attach($i);

           }

       });

       $posts->each(function ($post)

       {

           $number\_comments = rand(1,2);

           for($i=0 ; $i<$number\_comments ; $i++)

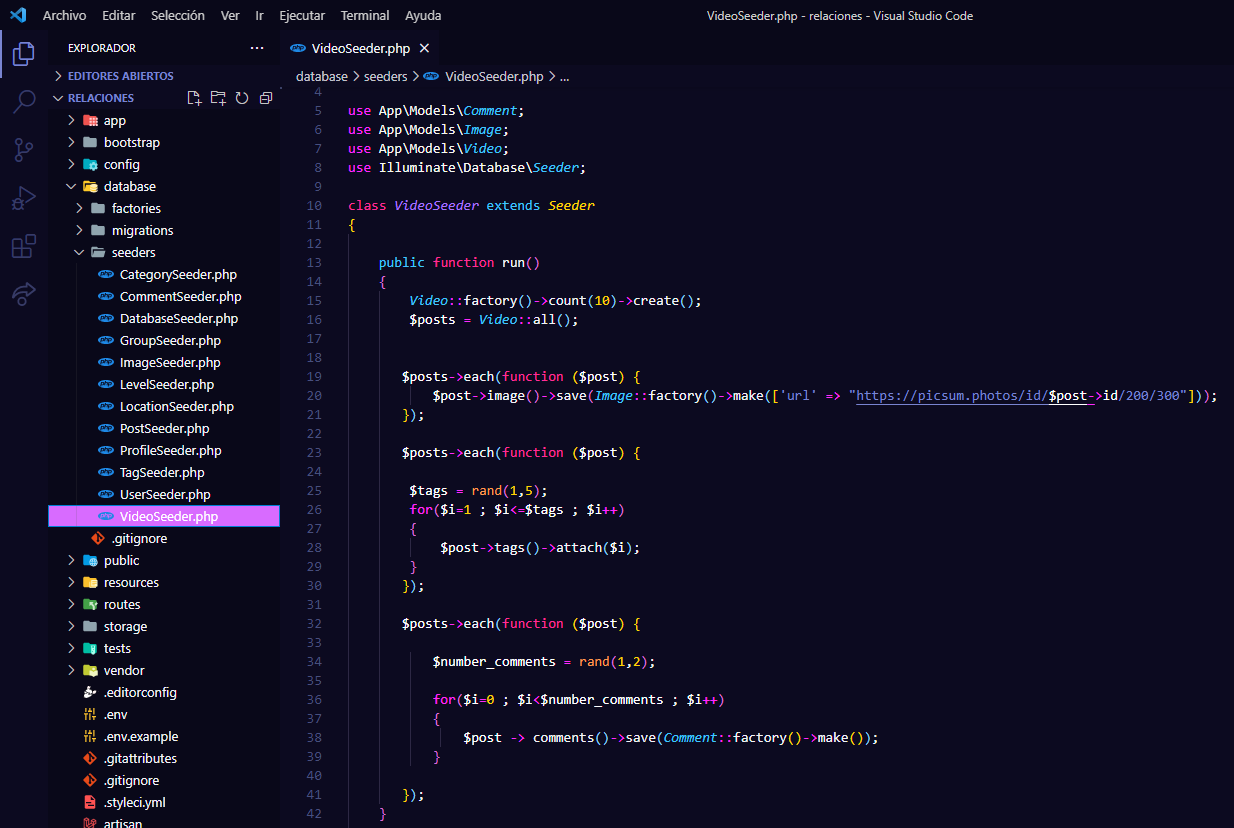
           {

               $post -> comments()->save(Comment::factory()->make());

           }

       });

1. Luego con el seeder de video.



        Video::factory()->count(10)->create();

        $posts = Video::all();

       $posts->each(function ($post)

       {

           $post->image()->save(Image::factory()->make(['url' => "https://picsum.photos/id/$post->id/150/150"]));

       });

       $posts->each(function ($post)

       {

        $tags = rand(1,5);

        for($i=1 ; $i<=$tags ; $i++)

        {

            $post->tags()->attach($i);

        }

       });

       $posts->each(function ($post)

       {

           $number\_comments = rand(1,2);

           for($i=0 ; $i<$number\_comments ; $i++)

           {

               $post -> comments()->save(Comment::factory()->make());

           }

       });

1. Finalmente se procede a llamar a cada seeder programado en el DatabaseSeeder y ejecutar el siguiente comando.

                LevelSeeder::class,

                GroupSeeder::class,

                UserSeeder::class,

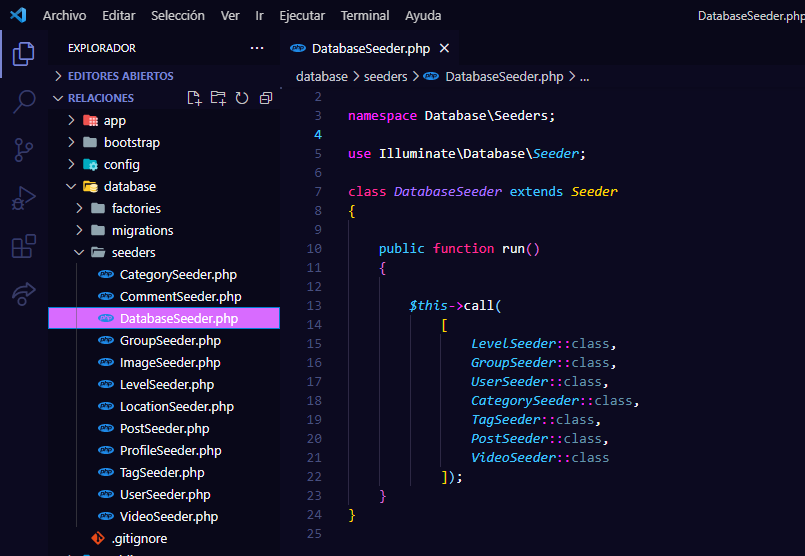
                CategorySeeder::class,

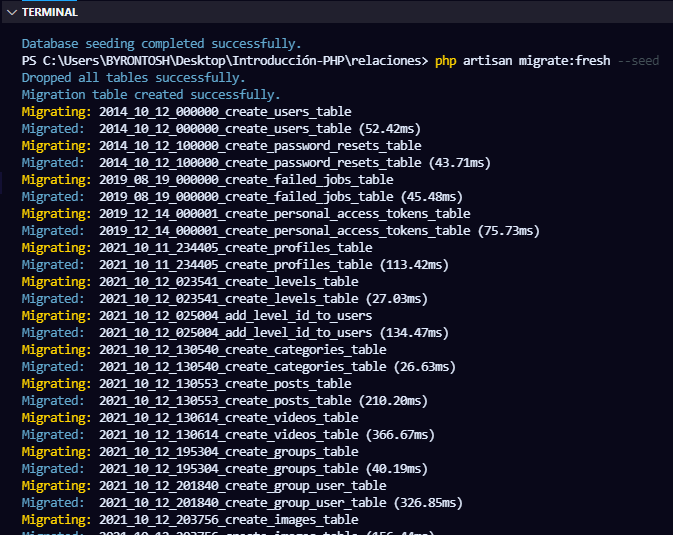
                TagSeeder::class,

                PostSeeder::class,

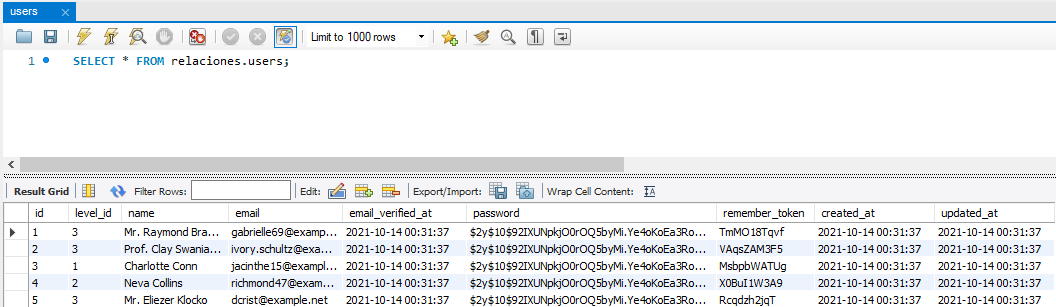
                VideoSeeder::class

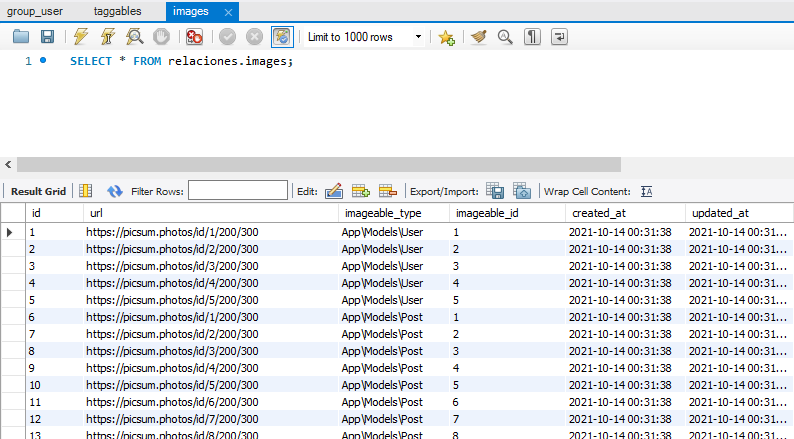
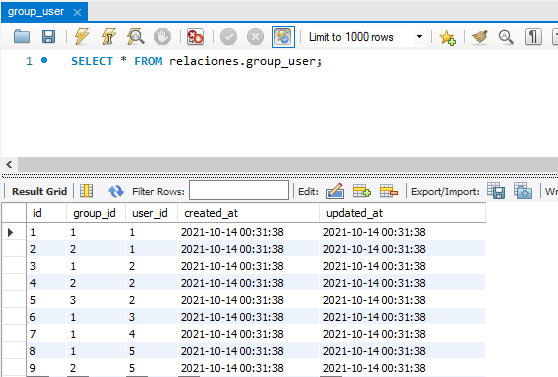
**php artisan migrate:fresh --seed**



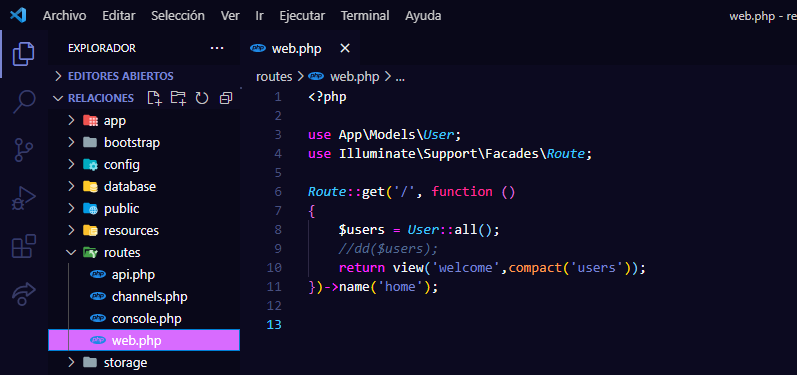


1. Finalmente verificar en la base de datos que los datos fakes estén llenados en cada una de las tablas.





1. Listo, ahora se procede a trabajar en la vista para presentar toda la información respectiva para lo cual se procede a trabajar en la ruta y luego en la vista.



Route::get('/', function ()

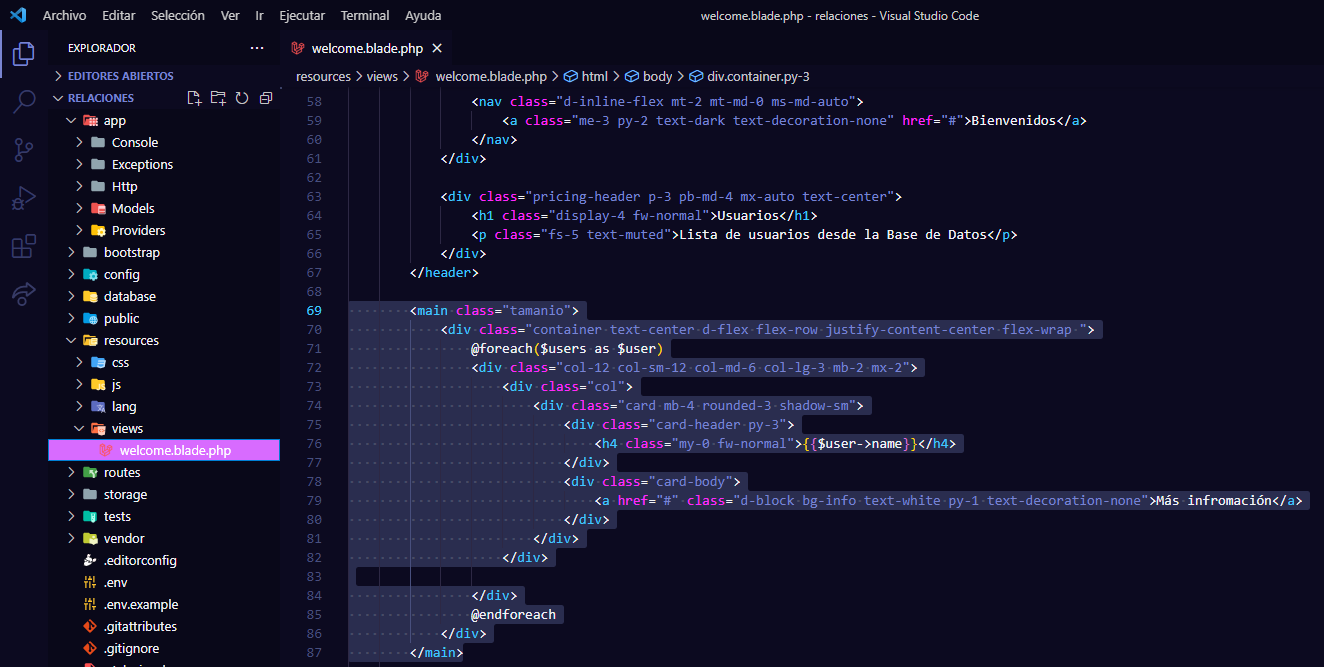
{

    $users = User::all();

    //dd($users);

    return view('welcome',compact('users'));

})->name('home');



<!doctype html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <meta name="description" content="">

    <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">

    <meta name="generator" content="Hugo 0.88.1">

    <title>ByronTosh</title>

    <link rel="canonical" href="https://getbootstrap.com/docs/5.1/examples/pricing/">

    <!-- CSS only -->

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

    <style>

        .bd-placeholder-img {

            font-size: 1.125rem;

            text-anchor: middle;

            -webkit-user-select: none;

            -moz-user-select: none;

            user-select: none;

        }

        @media (min-width: 768px) {

            .bd-placeholder-img-lg {

                font-size: 3.5rem;

            }

        }

        .tamanio {

            width: 100%;

            height: 100vh;

        }

    </style>

</head>

<body>

    <svg xmlns="http://www.w3.org/2000/svg" style="display: none;">

        <symbol id="check" viewBox="0 0 16 16">

            <title>Check</title>

            <path d="M13.854 3.646a.5.5 0 0 1 0 .708l-7 7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0 1 .708 0z" />

        </symbol>

    </svg>

    <div class="container py-3">

        <header>

            <div class="d-flex flex-column flex-md-row align-items-center pb-3 mb-4 border-bottom">

                <a href="/" class="d-flex align-items-center text-dark text-decoration-none">

                    <svg xmlns="http://www.w3.org/2000/svg" width="40" height="32" class="me-2" viewBox="0 0 118 94" role="img">

                        <title>Bootstrap</title>

                        <path fill-rule="evenodd" clip-rule="evenodd" d="M24.509 0c-6.733 0-11.715 5.893-11.492 12.284.214 6.14-.064 14.092-2.066 20.577C8.943 39.365 5.547 43.485 0 44.014v5.972c5.547.529 8.943 4.649 10.951 11.153 2.002 6.485 2.28 14.437 2.066 20.577C12.794 88.106 17.776 94 24.51 94H93.5c6.733 0 11.714-5.893 11.491-12.284-.214-6.14.064-14.092 2.066-20.577 2.009-6.504 5.396-10.624 10.943-11.153v-5.972c-5.547-.529-8.934-4.649-10.943-11.153-2.002-6.484-2.28-14.437-2.066-20.577C105.214 5.894 100.233 0 93.5 0H24.508zM80 57.863C80 66.663 73.436 72 62.543 72H44a2 2 0 01-2-2V24a2 2 0 012-2h18.437c9.083 0 15.044 4.92 15.044 12.474 0 5.302-4.01 10.049-9.119 10.88v.277C75.317 46.394 80 51.21 80 57.863zM60.521 28.34H49.948v14.934h8.905c6.884 0 10.68-2.772 10.68-7.727 0-4.643-3.264-7.207-9.012-7.207zM49.948 49.2v16.458H60.91c7.167 0 10.964-2.876 10.964-8.281 0-5.406-3.903-8.178-11.425-8.178H49.948z" fill="currentColor"></path>

                    </svg>

                    <span class="fs-4">Relaciones-Laravel</span>

                </a>

                <nav class="d-inline-flex mt-2 mt-md-0 ms-md-auto">

                    <a class="me-3 py-2 text-dark text-decoration-none" href="#">Bienvenidos</a>

                </nav>

            </div>

            <div class="pricing-header p-3 pb-md-4 mx-auto text-center">

                <h1 class="display-4 fw-normal">Usuarios</h1>

                <p class="fs-5 text-muted">Lista de usuarios desde la Base de Datos</p>

            </div>

        </header>

        <main class="tamanio">

            <div class="container text-center d-flex flex-row justify-content-center flex-wrap ">

                @foreach($users as $user)

                <div class="col-12 col-sm-12 col-md-6 col-lg-3 mb-2 mx-2">

                    <div class="col">

                        <div class="card mb-4 rounded-3 shadow-sm">

                            <div class="card-header py-3">

                                <h4 class="my-0 fw-normal">{{$user->name}}</h4>

                            </div>

                            <div class="card-body">

                                <a href="#" class="d-block bg-info text-white py-1 text-decoration-none">Más infromación</a>

                            </div>

                        </div>

                    </div>

                </div>

                @endforeach

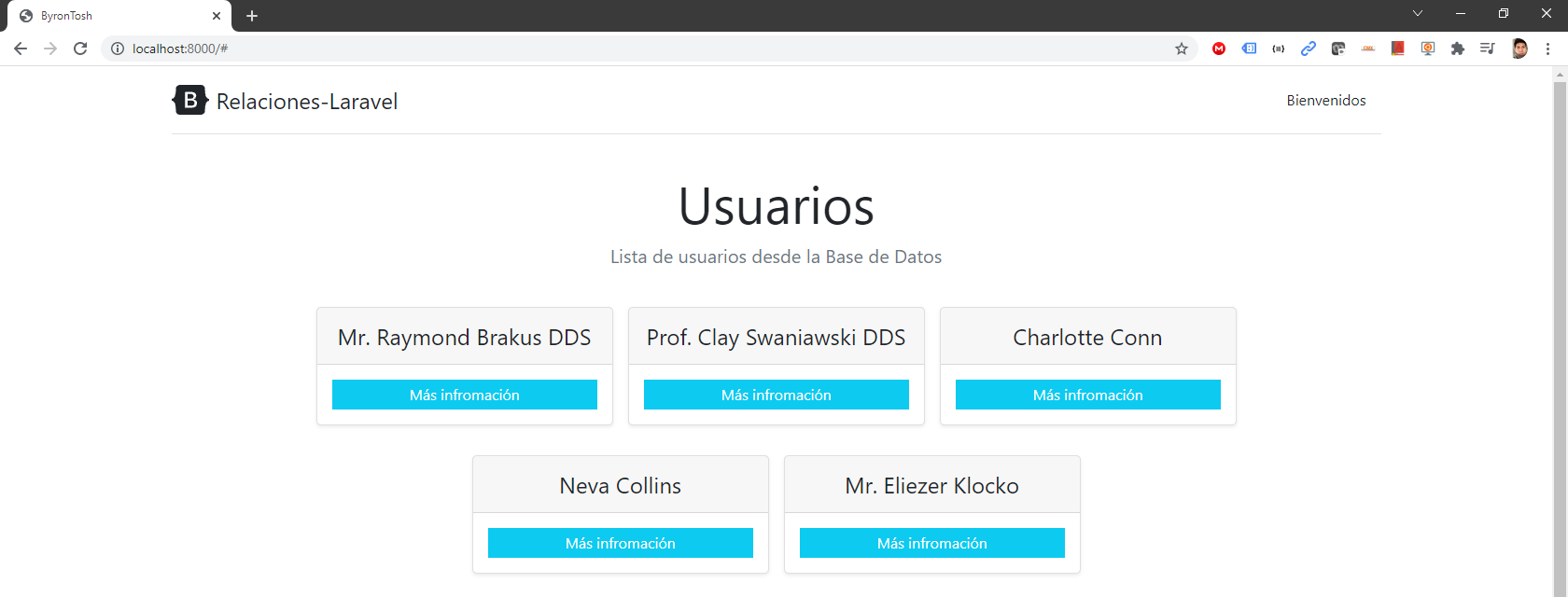
            </div>

        </main>

    </div>

</body>

</html>



1. Ahora se procede con la creación de una nueva ruta y vista respectiva para trabajar con la información respectiva.

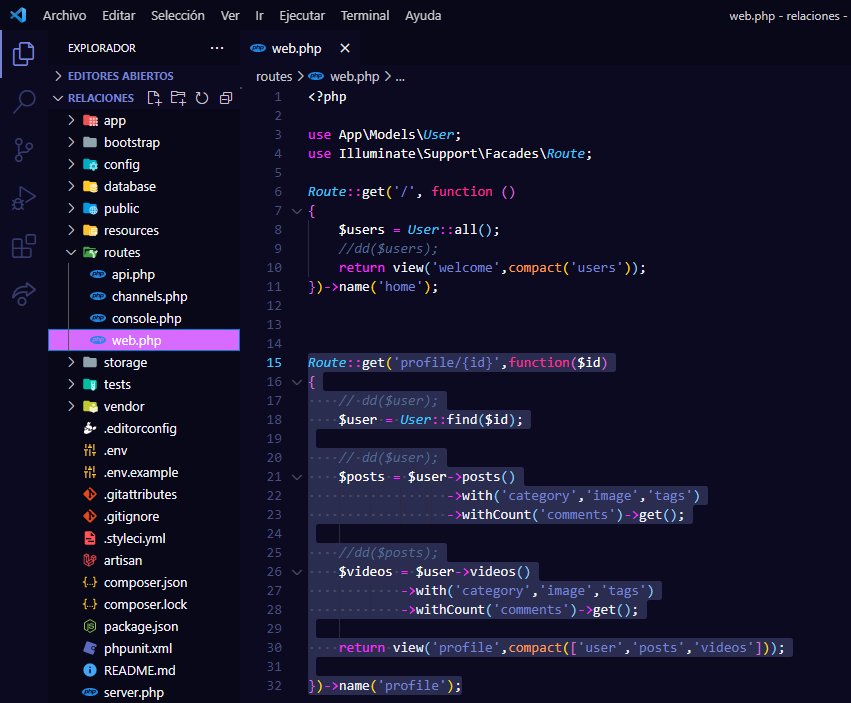


<a href="{{route('profile',$user->id)}}" class="d-block bg-info text-white py-1 text-decoration-none">Más infromación</a>

<https://laravel.com/docs/9.x/eloquent-relationships#eager-loading>

<https://laravel.com/docs/9.x/eloquent-relationships#counting-related-models>

<https://laravel.com/docs/9.x/helpers#method-str-plural>



Route::get('profile/{id}',function($id)

{

    // dd($user);

    $user = User::find($id);

    // dd($user);

    $posts = $user->posts()

                  ->with('category','image','tags')

                  ->withCount('comments')->get();

    //dd($posts);

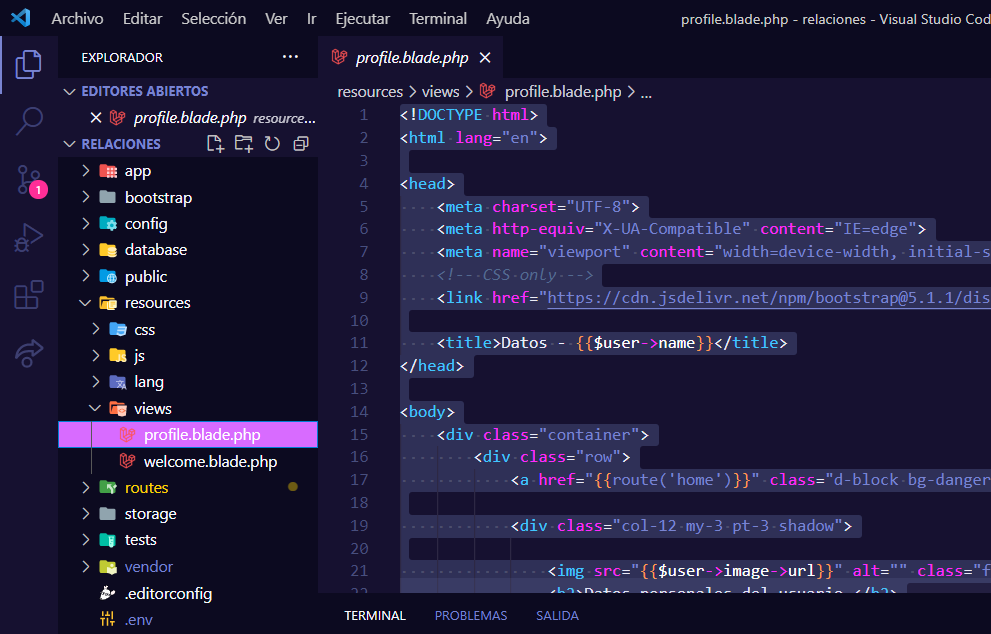
    $videos = $user->videos()

            ->with('category','image','tags')

            ->withCount('comments')->get();

    return view('profile',compact(['user','posts','videos']));

})->name('profile');



<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

*<!-- CSS only -->*

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJkqJByhZMI3AhiU" crossorigin="anonymous">

    <title>Datos - {{$user->name}}</title>

</head>

<body>

    <div class="container">

        <div class="row">

            <a href="{{route('home')}}" class="d-block bg-danger text-white py-1 text-decoration-none ">Regresar</a>

            <div class="col-12 my-3 pt-3 shadow">

                <img src="{{$user->image->url}}" alt="" class="float-left rounded-circle mb-4">

                <h2>Datos personales del usuario </h2>

                <ul>

                    <li>

                        <b> Nombre:</b> {{$user->name}}

                    </li>

                    <li>

                        <b> Email:</b> {{$user->email}}

                    </li>

                </ul>

                <hr>

                <h3 class="mt-4">Redes sociales</h3>

                <ol>

                    <li>{{$user->profile->github}}</li>

                    <li>

                        <a href="{{$user->profile->website}}" target="\_blank">{{$user->profile->website}}</a>

                    </li>

                </ol>

                <p>

                    <strong>País</strong>: {{$user->location->pais}}

                    <br>

                    <strong>Nivel</strong>:

                    @if($user->level)

                    {{$user->level->nombre}}

                    @else

                    el usuario no posee un nivel.

                    @endif

                </p>

                <hr>

                <strong>Grupos:</strong>

                @forelse($user->groups as $group)

                <span class="badge bg-success">{{$group->nombre}}</span>

                @empty

                <em>No pertenece a ningun grupo</em>

                @endforelse

                <h3 class="mt-4 text text-center my-4">Posts</h3>

                <div class="row">

                    @forelse($posts as $post)

                    <div class="col-6">

                        <div class="card mb-3">

                            <div class="row">

                                <div class="col-md-4">

                                    <img src="{{$post->image->url}}" alt="" class="card-img">

                                </div>

                                <div class="col-md-8">

                                    <h6 class="card-subtitle text-muted">

                                        <br>

                                        <b class="text-success"> Título: </b>{{$post->nombre}}

                                        <br>

                                        <b class="text-info"> Categoría: </b> {{$post->category->nombre}}

                                        <br>

                                        {{$post->comments\_count}}

                                        {{*Str*::plural('comentario',$post->comments\_count)}}

                                    </h6>

                                    <p class="card-text small">

                                        @foreach($post->tags as $tag)

                                        <span>

                                            #{{$tag->nombre}}

                                        </span>

                                        @endforeach

                                    </p>

                                </div>

                            </div>

                        </div>

                    </div>

                    @empty

                    <em>No existen posts para este usuario</em>

                    @endforelse

                </div>

                <h3 class="mt-4 text text-center my-4">Vídeos</h3>

                <div class="row">

                    @forelse($videos as $video)

                    <div class="col-6">

                        <div class="card mb-3">

                            <div class="row">

                                <div class="col-md-4">

                                    <img src="{{$video->image->url}}" alt="" class="card-img">

                                </div>

                                <div class="col-md-8">

                                    <h6 class="card-subtitle text-muted">

                                        <br>

                                        <b class="text-success"> Título: </b>{{$video->nombre}}

                                        <br>

                                        <b class="text-info"> Categoría: </b> {{$video->category->nombre}}

                                        <br>

                                        {{$video->comments\_count}}

                                        {{*Str*::plural('comentario',$video->comments\_count)}}

                                    </h6>

                                    <p class="card-text small">

                                        @foreach($video->tags as $tag)

                                        <span>

                                            # {{$tag->nombre}}

                                        </span>

                                        @endforeach

                                    </p>

                                </div>

                            </div>

                        </div>

                    </div>

                    @empty

                    <em>No existen vídeos para este usuario</em>

                    @endforelse

                </div>

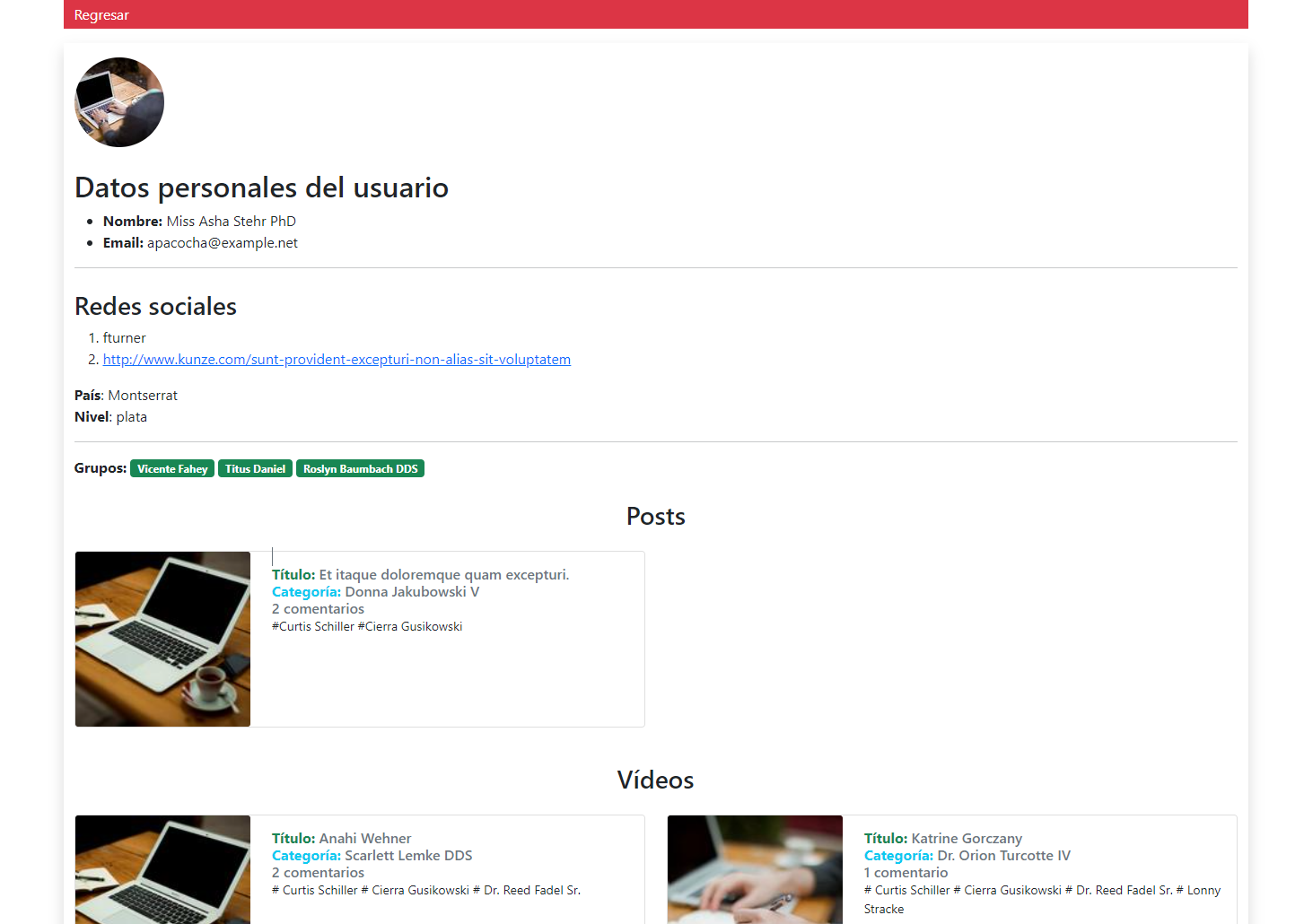
            </div>

        </div>

    </div>

</body>

</html>

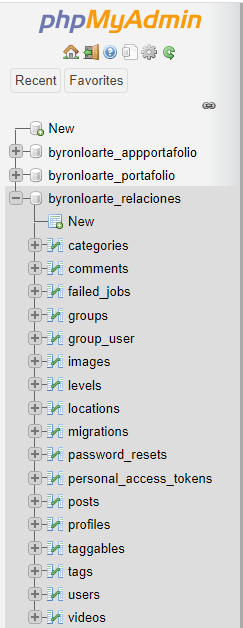


<https://github.com/Byrontosh/04_Relaciones>

***TAREA DESAFÍO***



1. Realizar el despliegue de este proyecto en las siguientes plataformas:
   * Sistema web en Heroku
   * BDD en alwaysdata



<https://apprelaciones.herokuapp.com/>