MANUAL PROGRAMADOR

1. Introducción

Para comenzar haré una breve introducción del proyecto.

El objetivo es hacer una web en la que cada usuario pueda registrar las películas que ha visto clasificadas por géneros. Por lo tanto vamos a relacionar los géneros y películas a cada usuario. Además habrá dos vistas más en las que se verán los distintos géneros y películas de todos los usuarios.

Ahora, en el apartado técnico para conseguir los puntos extra usaremos Jetstream para la autenticación, Tailwind para el css. Además meteremos las vistas dentro de una carpeta que denominaremos content para cumplir con el apartado de vistas dentro de carpetas. Además agregaremos traducciones a los mensajes de validación. Y por último todo el proyecto estará alojado en github.

2. Requisitos:

Para poder crear nuestro proyecto en Laravel necesitamos tener instalados los siguientes componentes:

2.1. Composer

2.2. Artisan

2.3. Xampp (Base Local)

2.4. Node.js

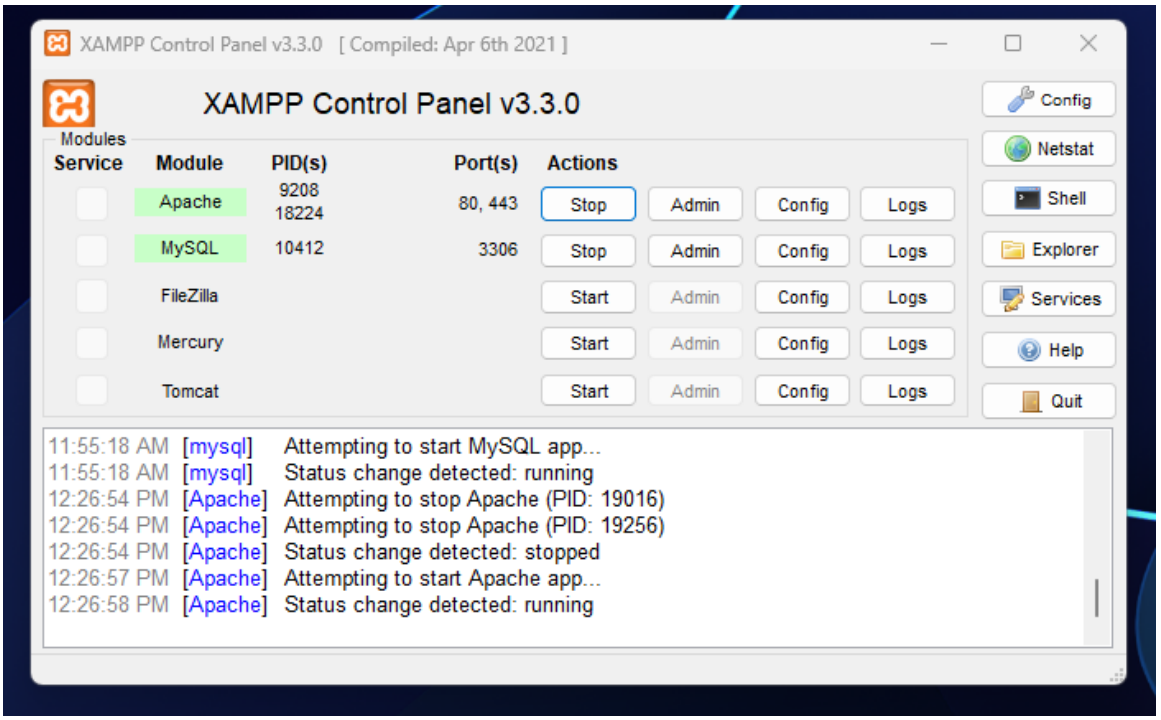
2.5. Editor de texto o IDE (En este caso usamos Visual Studio Code)

3. Instalar requisitos

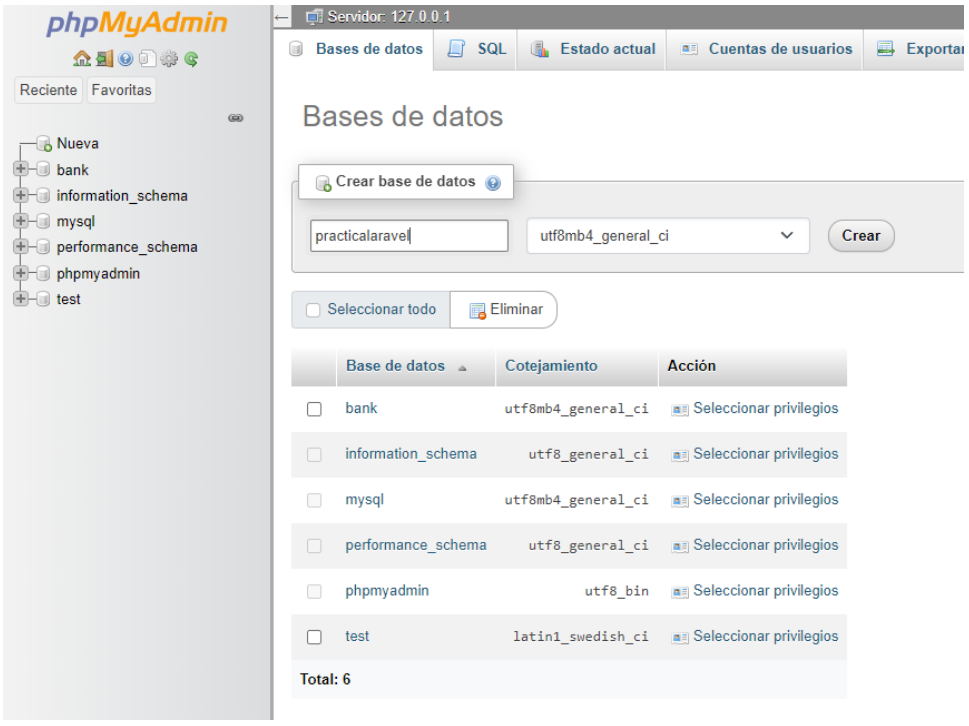
Lo primero que debemos hacer es instalar tanto Xampp, Composer y Node.js

4. Crear base de datos

4.1. Antes de nada ejecutaremos Apache y MySQL desde Xampp



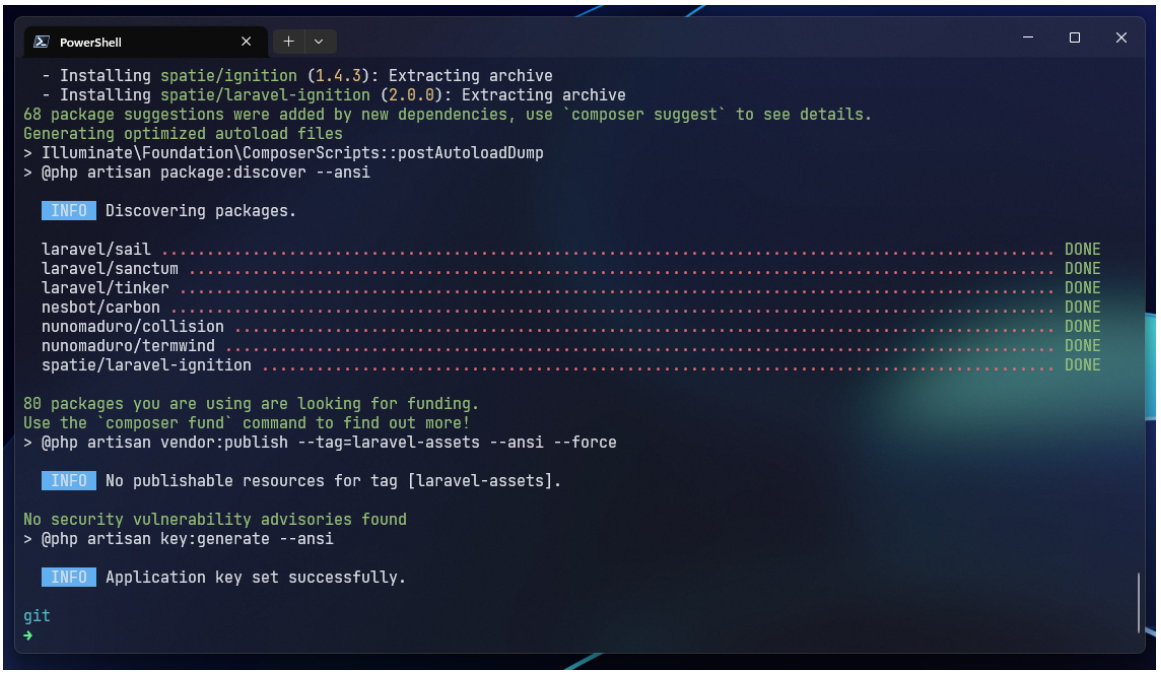
4.2. Abriremos phpMyAdmin y crearemos la base de datos “practicalaravel”



5. Crear proyecto

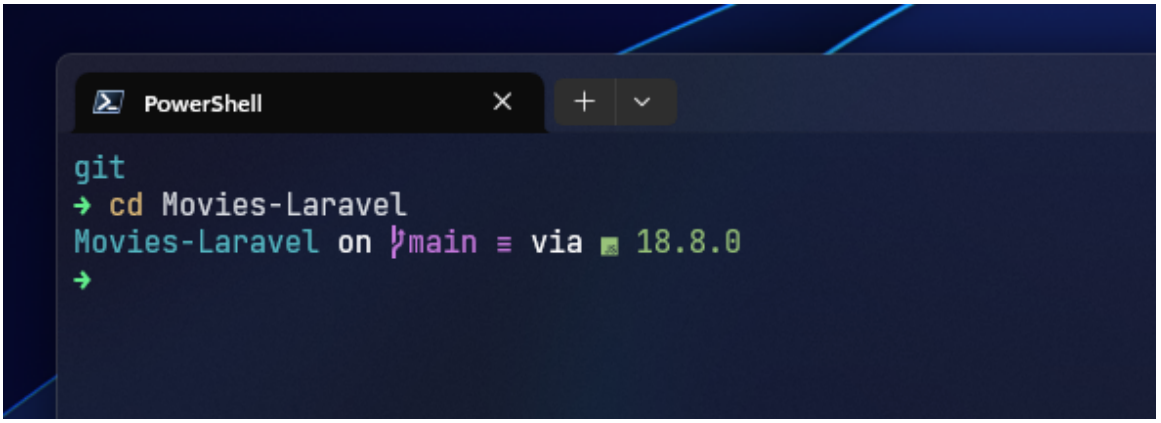
5.1. Una vez creada la base de datos debemos abrir una consola y lanzar el siguiente comando.

```
composer create-project laravel/laravel Movies-Laravel
```



5.2. Nos dirigimos a la carpeta del proyecto

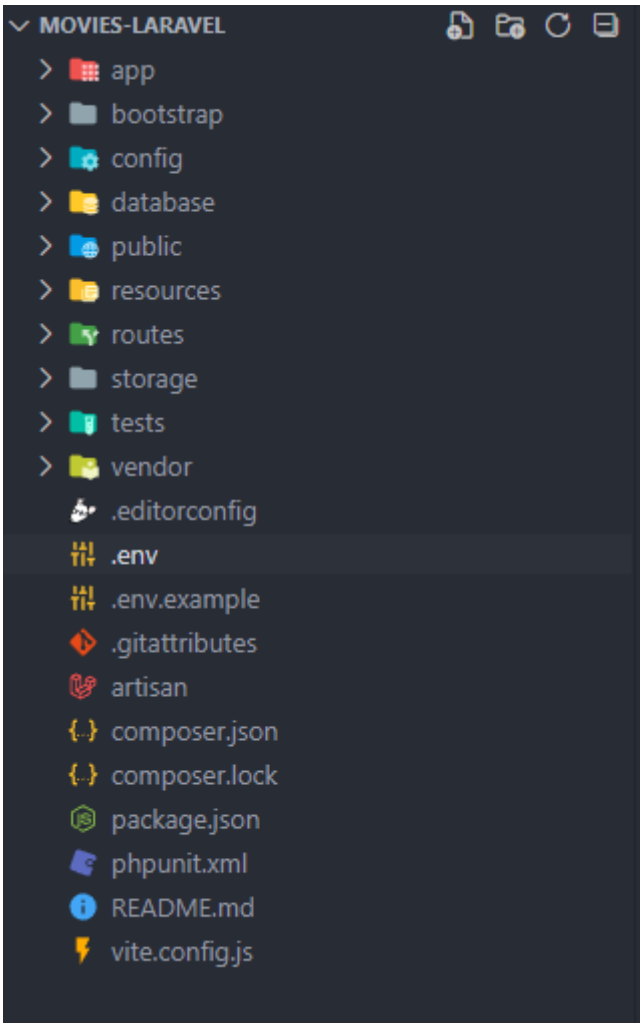
```
cd Movies-Laravel
```



5.3. Lo abrimos con vscode. Podemos abrirlo directamente con el comando:

```
code .
```

5.4. Nos dirigimos al archivo .env y lo editamos con el nombre de nuestra base de datos que creamos anteriormente.

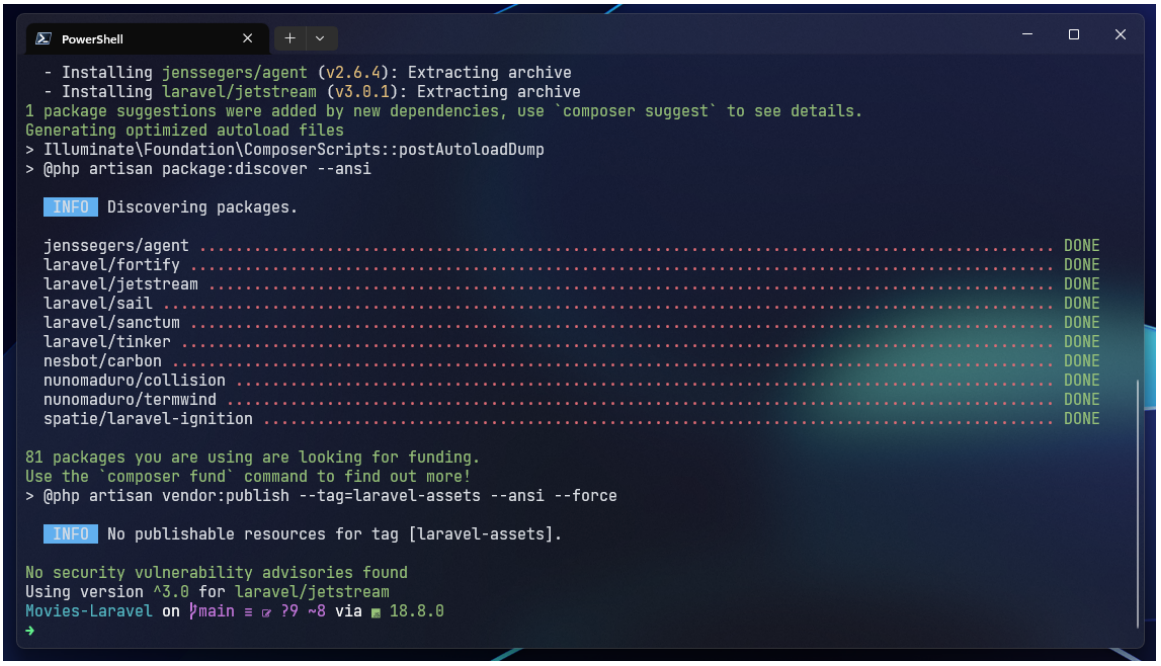



```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=practicalaravel
DB_USERNAME=root
DB_PASSWORD=
```

6. Instalación de Laravel Jetstream y Livewire

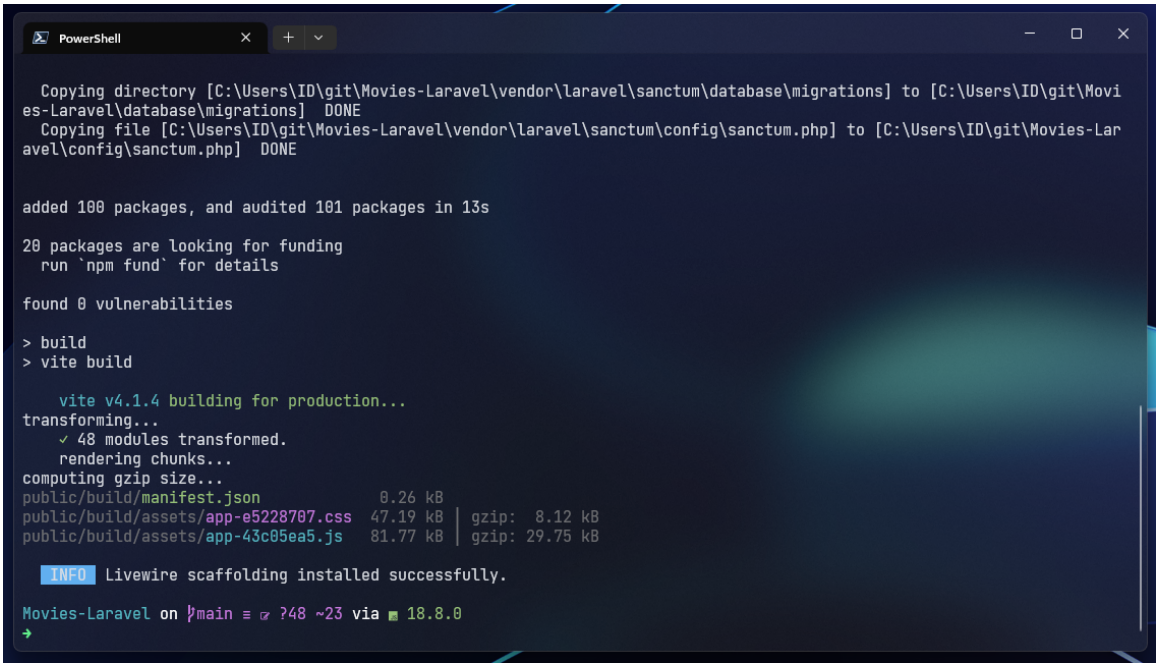
6.1. Comenzaremos con la instalación de Jetstream. Volvemos a la terminal y ejecutamos el siguiente comando:

```
composer require laravel/jetstream
```



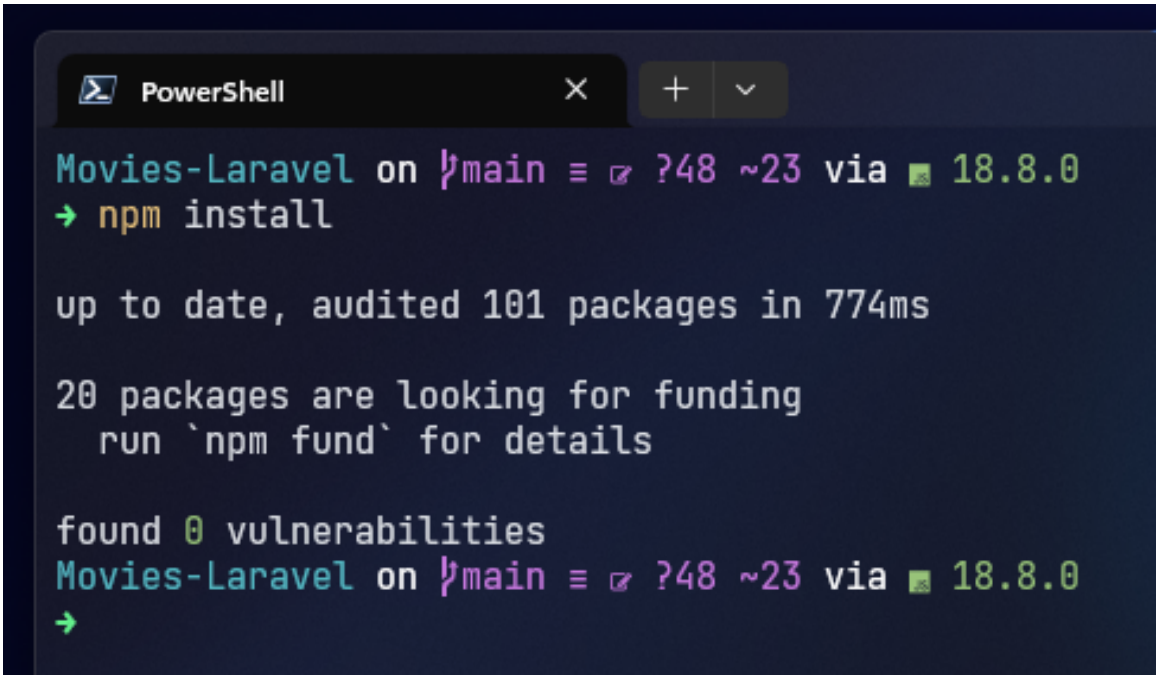
6.2. A continuación procederemos a la instalación de Livewire con el siguiente comando.

```
php artisan jetstream:install livewire
```



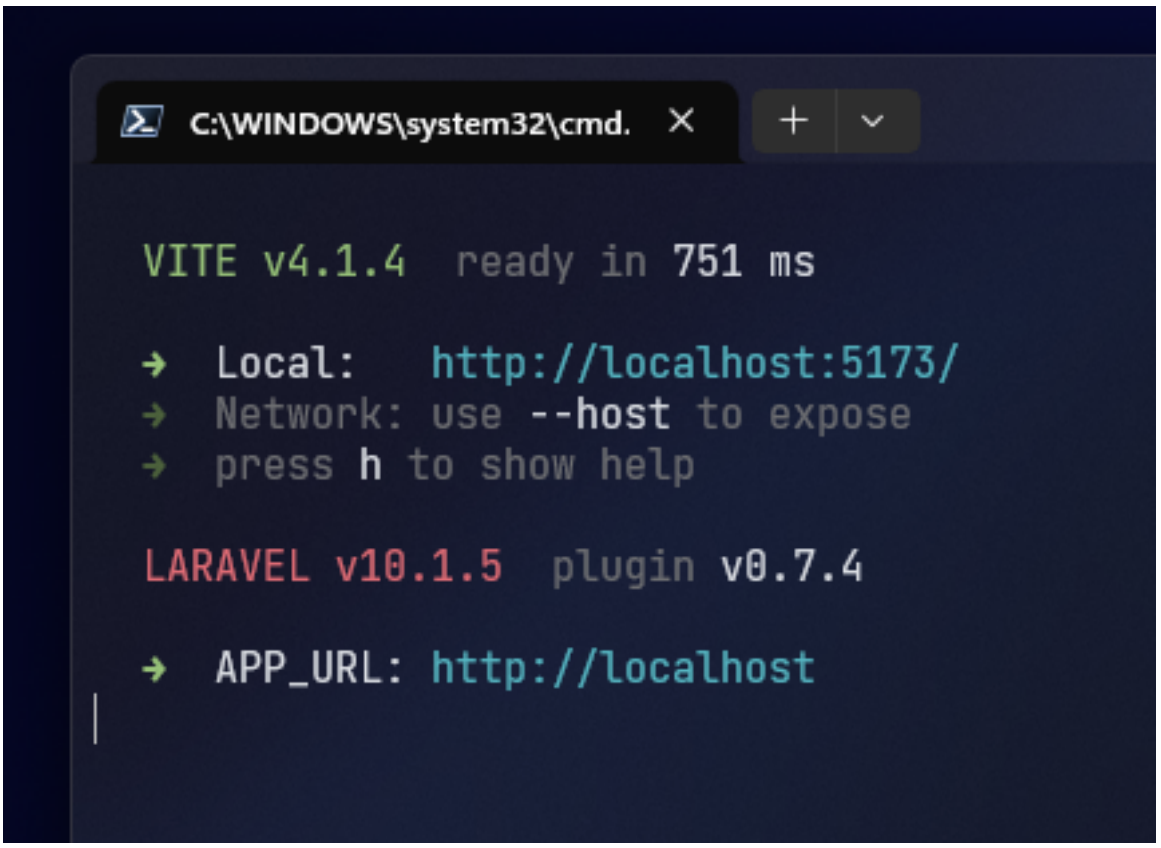
6.3. Ejecutaremos:

```
npm install
```



6.4. Este comando lo debemos ejecutar en otro terminal y dejarlo ejecutándose antes de que ejecutemos nuestro proyecto.

```
npm run dev
```



7. Crear Migraciones por defecto:

En otra consola desde la misma ruta ejecutaremos el siguiente comando para crear las tablas por defecto que trae Jetstream

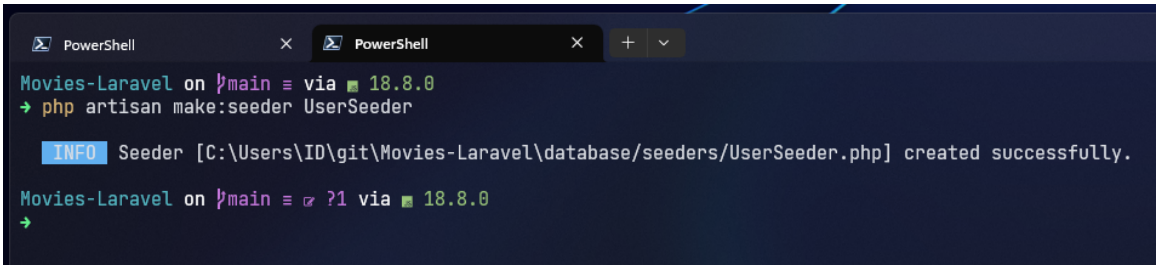
```
php artisan migrate
```

8. Crear seeder

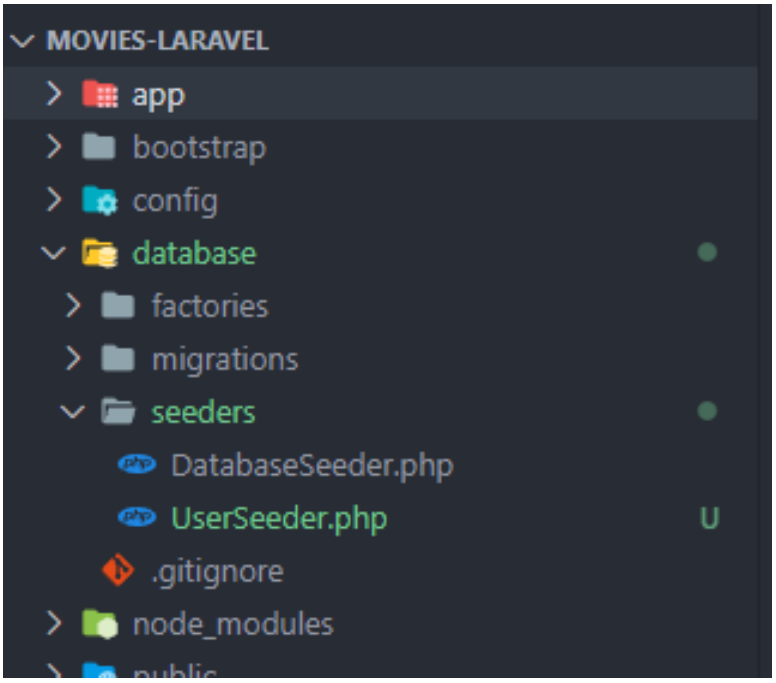
Como se documenta en las especificaciones del proyecto debemos registrar nuestro primer usuario en la tabla de usuario mediante un seeder. Para ello ejecutaremos los siguientes pasos:

8.1. Comenzaremos creando nuestro seeder. Para eso vamos a nuestra consola y ejecutamos el siguiente comando.

```
php artisan make:seeder UserSeeder
```



8.2. A continuación nos dirigimos a la carpeta de seeders (database\seeders\UserSeeder.php). Como vemos nuestro seeder se ha creado correctamente.



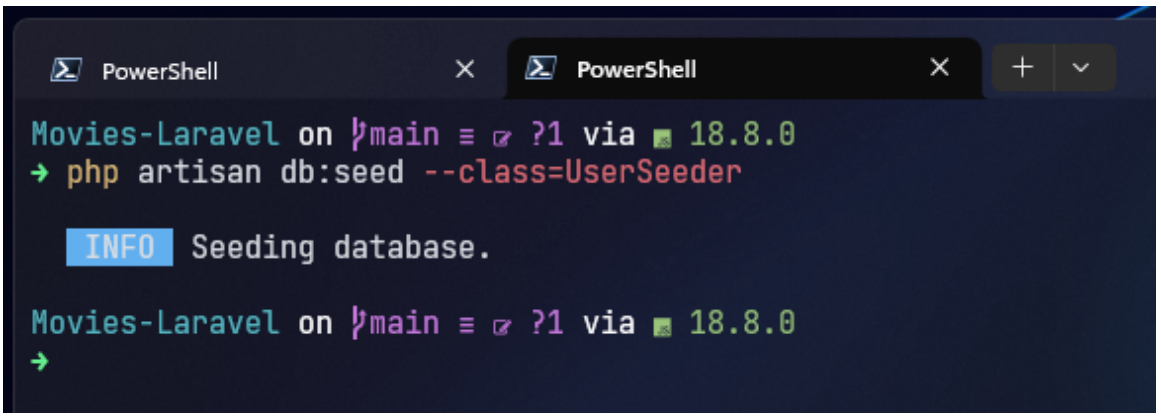
8.3. Y editaremos el código para crear nuestro primer usuario mediante el seeder

UserSeeder.php U database\seeders\UserSeeder.php\...

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;

0 references | 0 implementations
class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    0 references | 0 overrides
    public function run(): void
    {
        DB::table('users')->insert([
            'name' => 'Gabriel',
            'email' => 'gabriel@gmail.com',
            'password' => Hash::make('12345678')
        ]);
    }
}
```

8.4. Por último volvemos a la terminal y ejecutamos el siguiente comando.

```
php artisan db:seed --class=UserSeeder
```

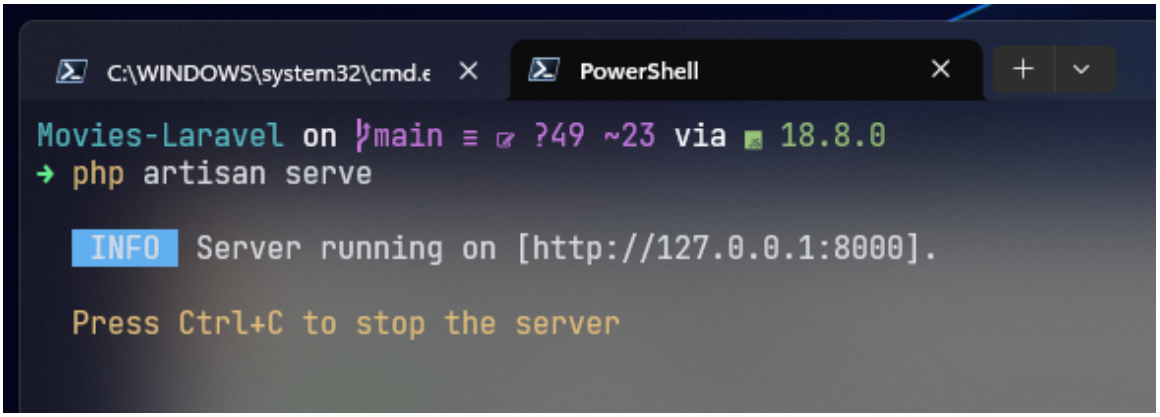


8.5. Nuestro usuario se ha creado correctamente como vemos a continuación.

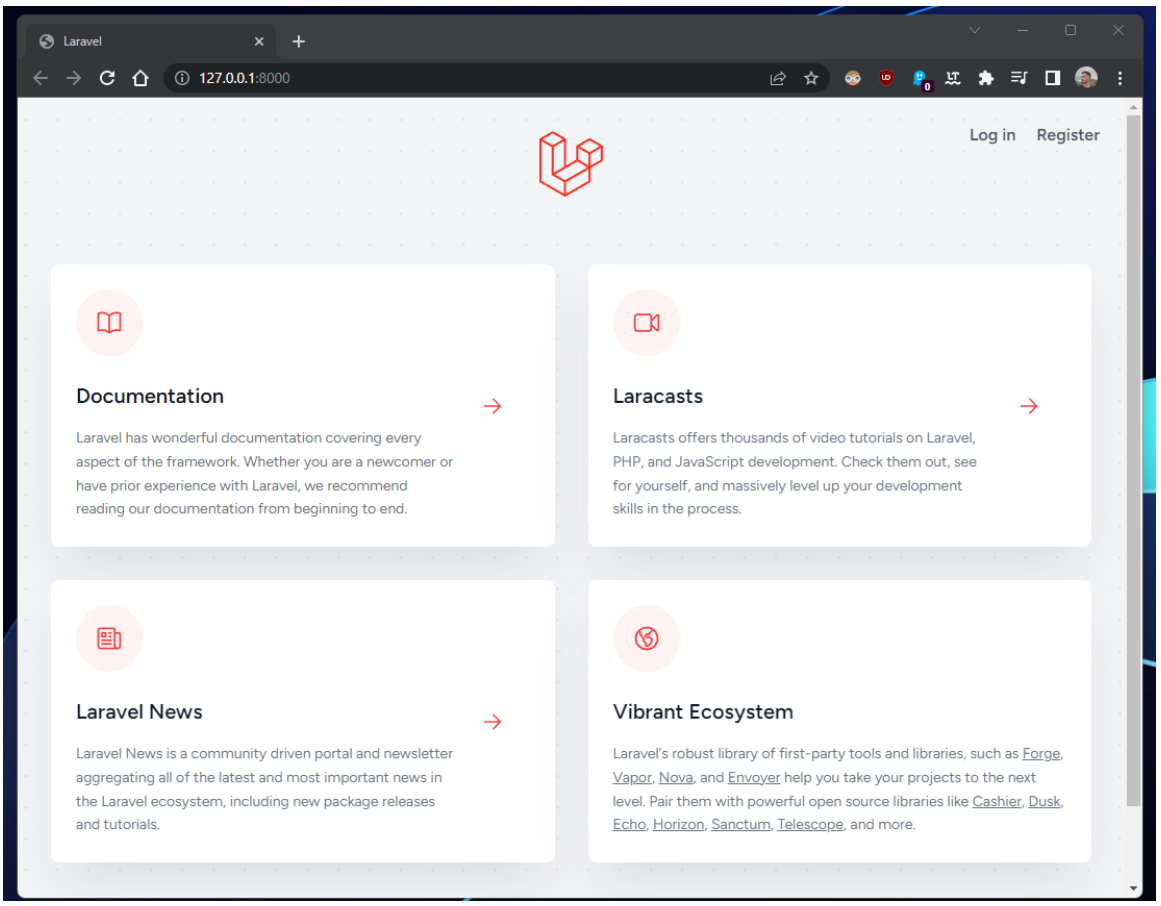
9. Probamos el proyecto:

9.1. Para probarlo ejecutamos el comando:

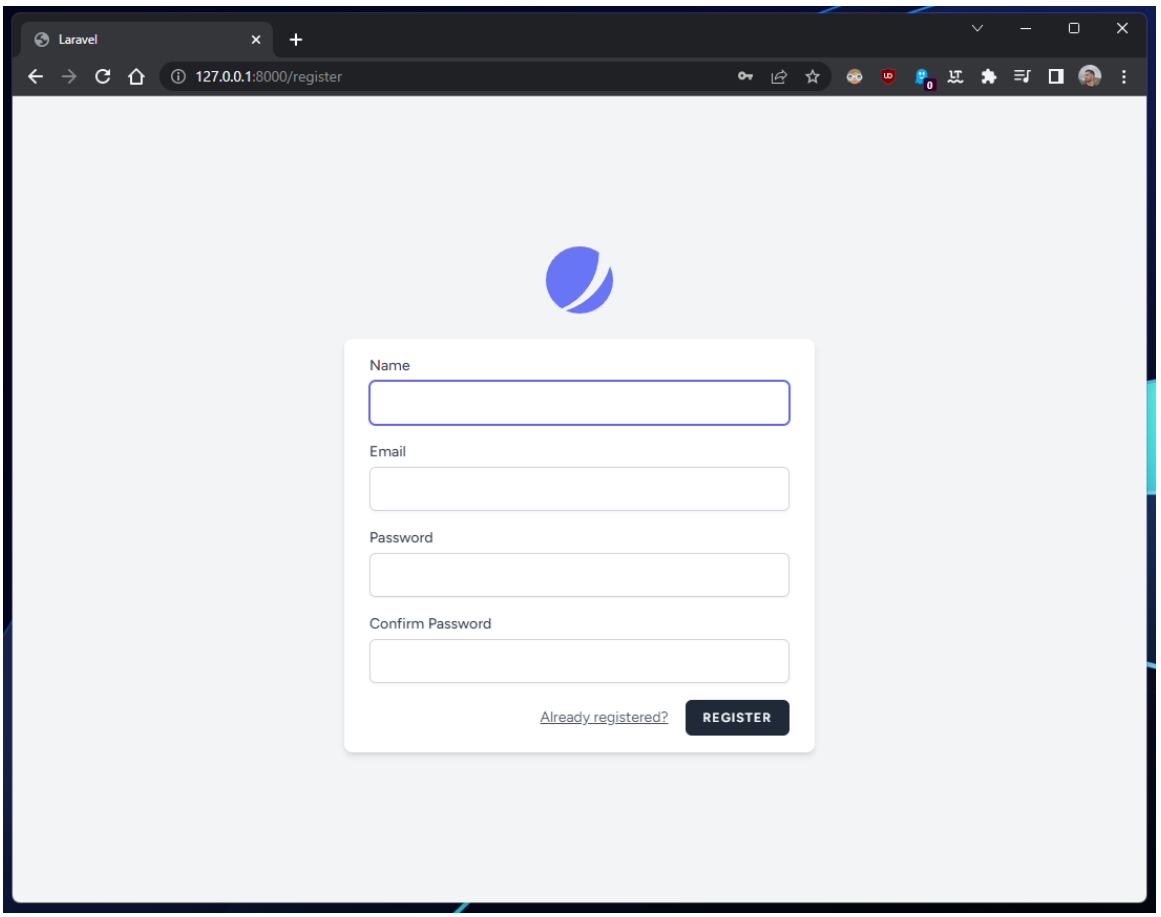
```
php artisan serve
```



9.2. Abrimos un navegador con la ruta que nos muestra la consola, y deberíamos ver algo así:



9.3. Podemos probar ya a logearnos con el usuario que insertamos anteriormente con el seeder.



10. Crear Modelos, Factory y Migración

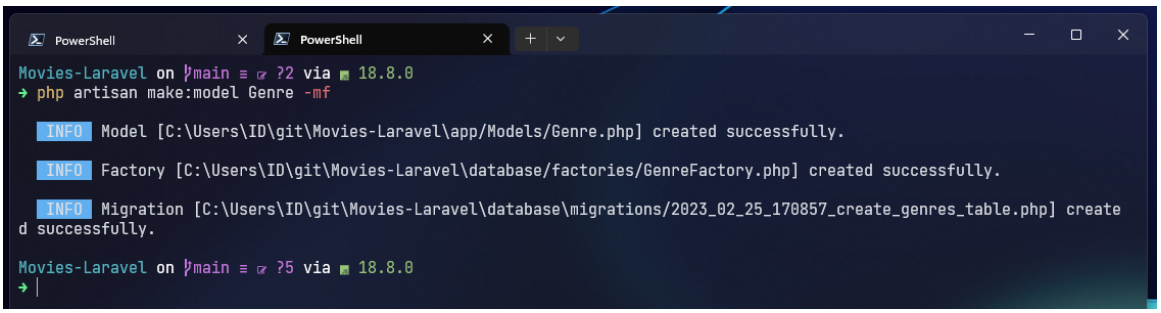
El framework de Laravel usa el paradigma Modelo-Vista-Controlador, por lo que es necesario crear cada uno de esos elementos.

Las Factorys nos permiten autogenerar datos para nuestras tablas y así trabajar más cómodamente.

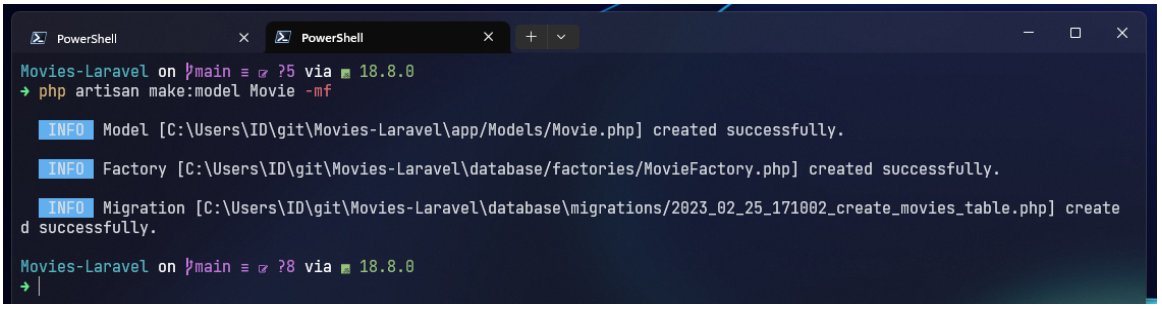
10.1. Creación de Modelos, Factory y Migración

En una nueva consola desde la misma ruta ejecutamos los siguientes comandos para crear los modelos y las factories. Con la -m le indicamos que cree la migración y con la -f que cree la factory.

```
php artisan make:model Genre -mf
```

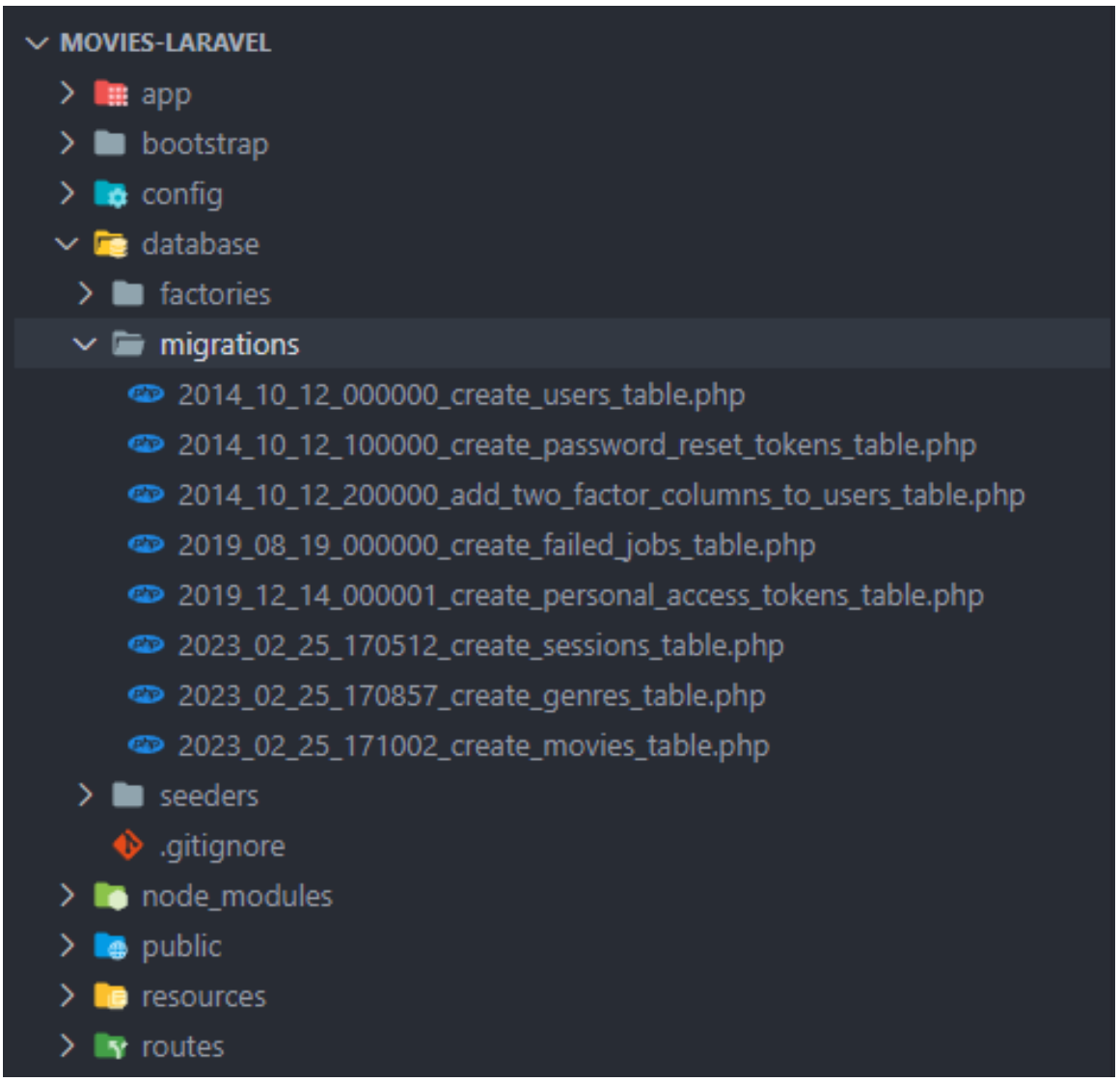


php artisan make:model Movie -mf



10.2. Migraciones

10.2.1. A continuación volvemos al visual y nos dirigimos a la carpeta de migraciones.



10.2.2. Comenzaremos con la migración de géneros

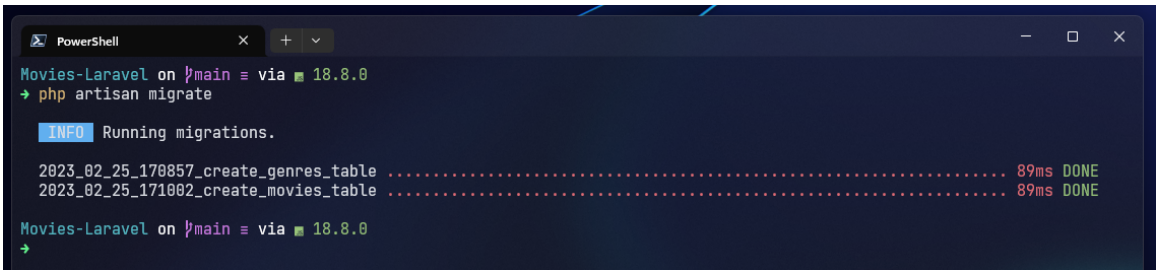
```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('genres', function (Blueprint $table) {
            $table->engine="InnoDB";
            $table->id();
            $table->integer('user_id')->index();
            $table->string('name');
            $table->timestamps();
        });
    }
```

10.2.3. Y a continuación hacemos la migración de películas

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('movies', function (Blueprint $table) {
            $table->engine="InnoDB";
            $table->id();
            $table->bigInteger('genre_id')->unsigned();
            $table->integer('user_id')->index();
            $table->string('name');
            $table->integer('duration')->unsigned();
            $table->string('director');
            $table->boolean('box_office');
            $table->timestamps();
            $table->foreign("genre_id")->references('id')->on('genres')->cascadeOnDelete();
        });
    }
```

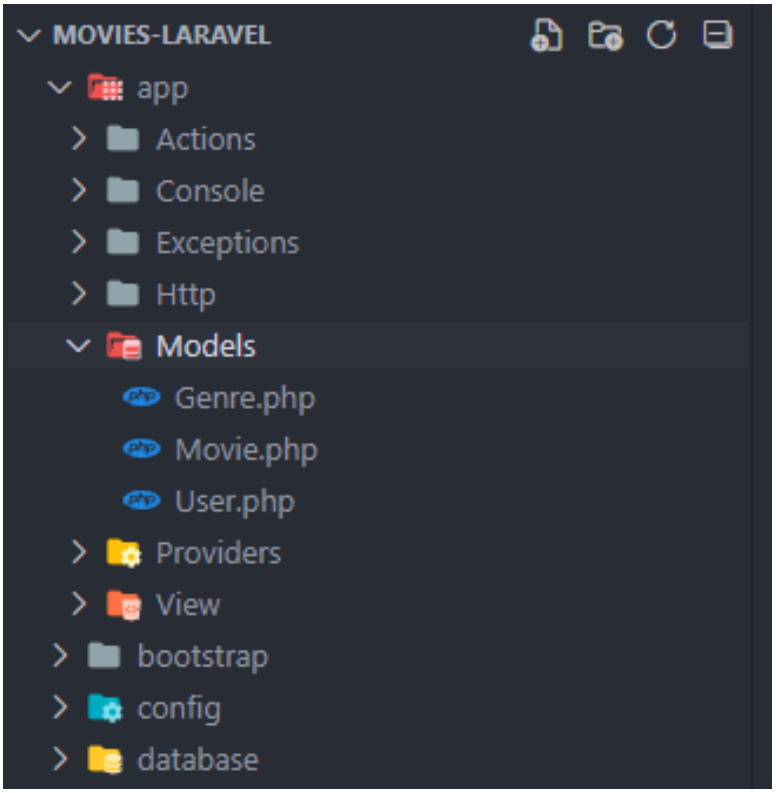
10.2.4. A continuación ejecutamos desde consola el siguiente comando:

```
php artisan migrate
```



10.3. Modelos

10.3.1. A continuación volvemos al visual y nos dirigimos a la carpeta de modelos.

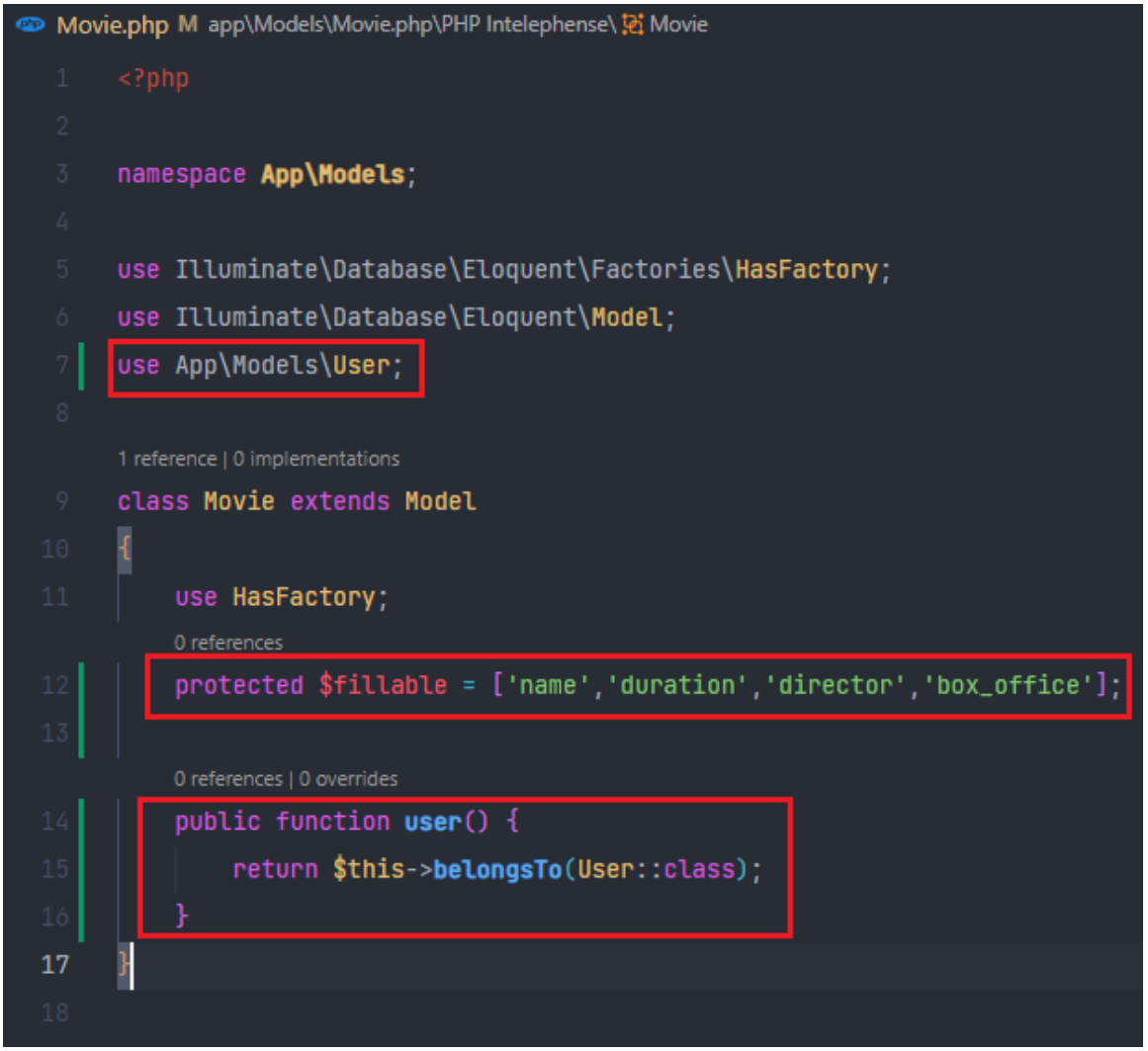


10.3.2. Añadimos las siguientes líneas al modelo para género.

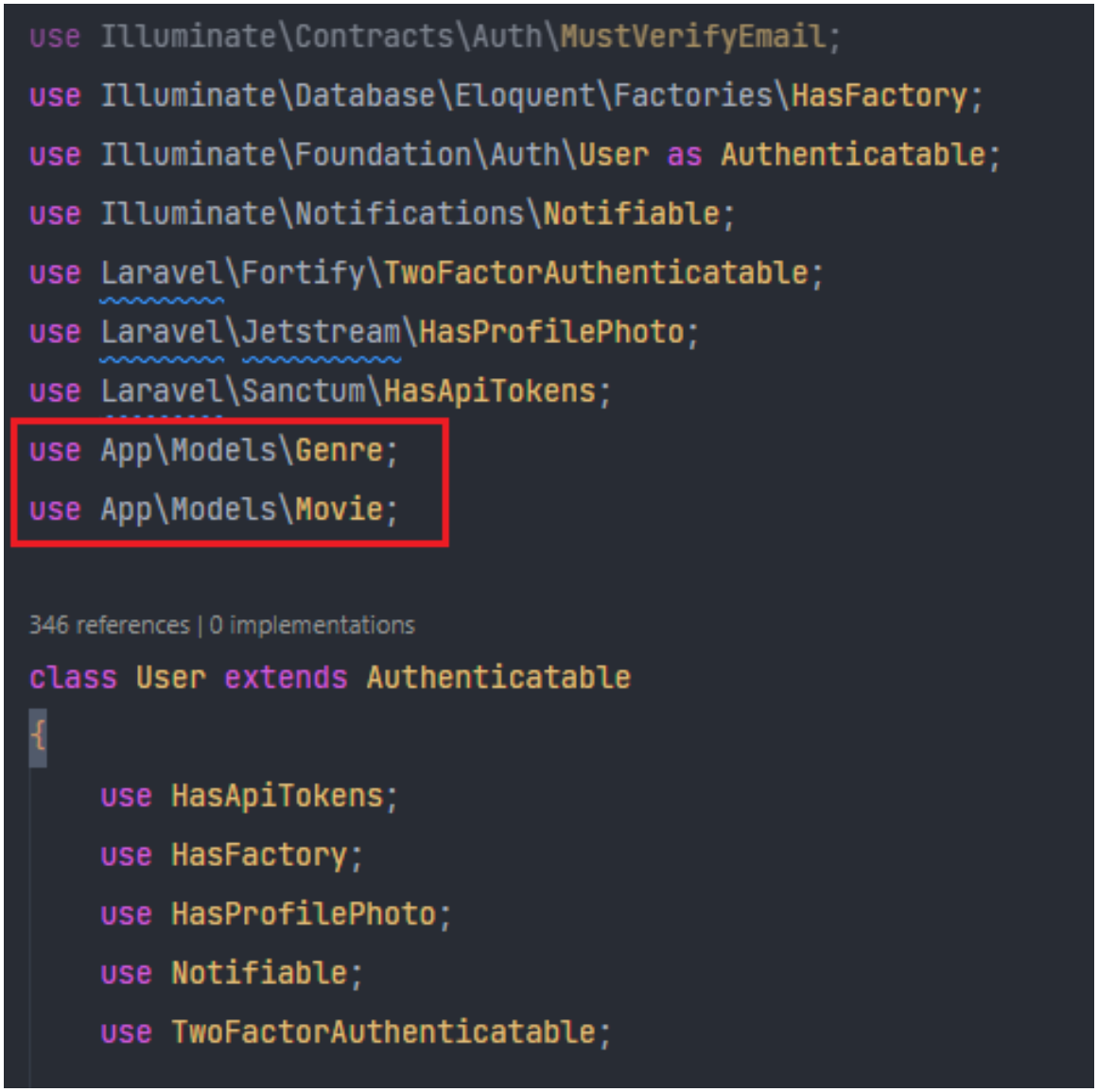
Genre.php M app\Models\Genre.php\...

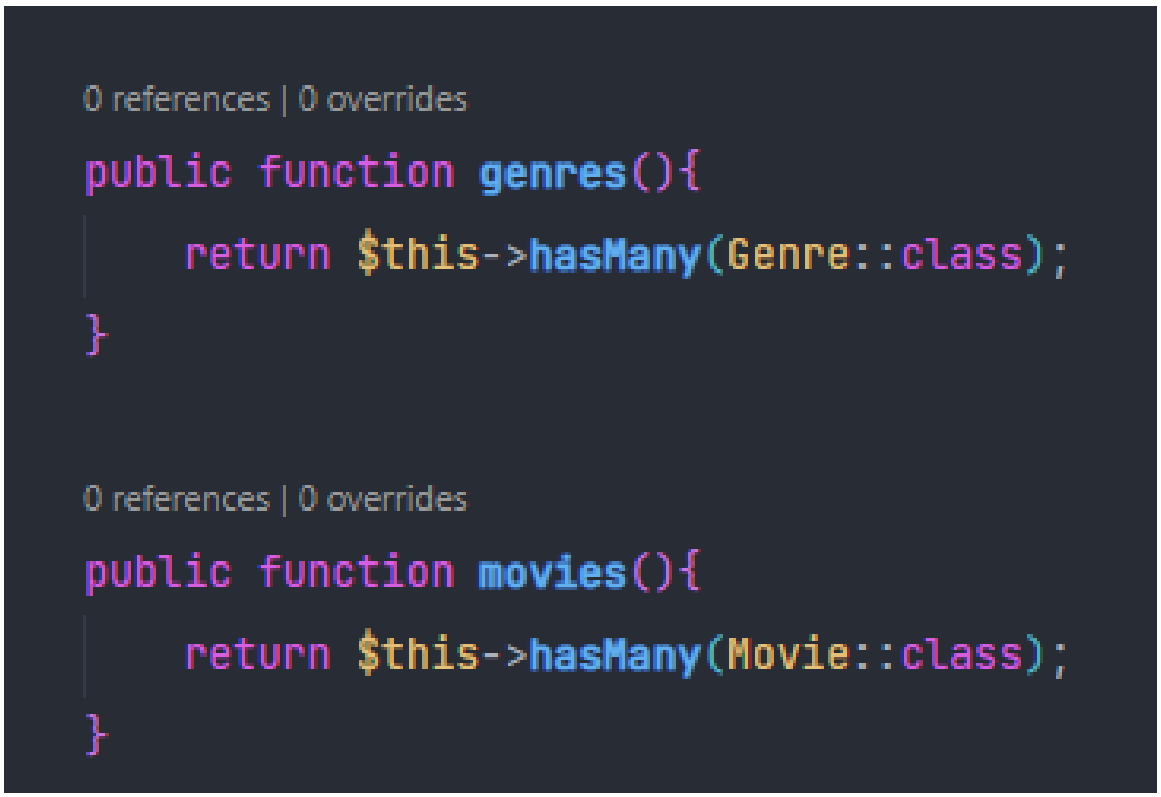
```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\User;

1 reference | 0 implementations
class Genre extends Model
{
    use HasFactory;

    0 references
    protected $fillable = ['name'];

    0 references | 0 overrides
    public function user() {
        return $this->belongsTo(User::class);
    }
}
```

10.3.3. Añadimos las siguientes para el modelo de película.



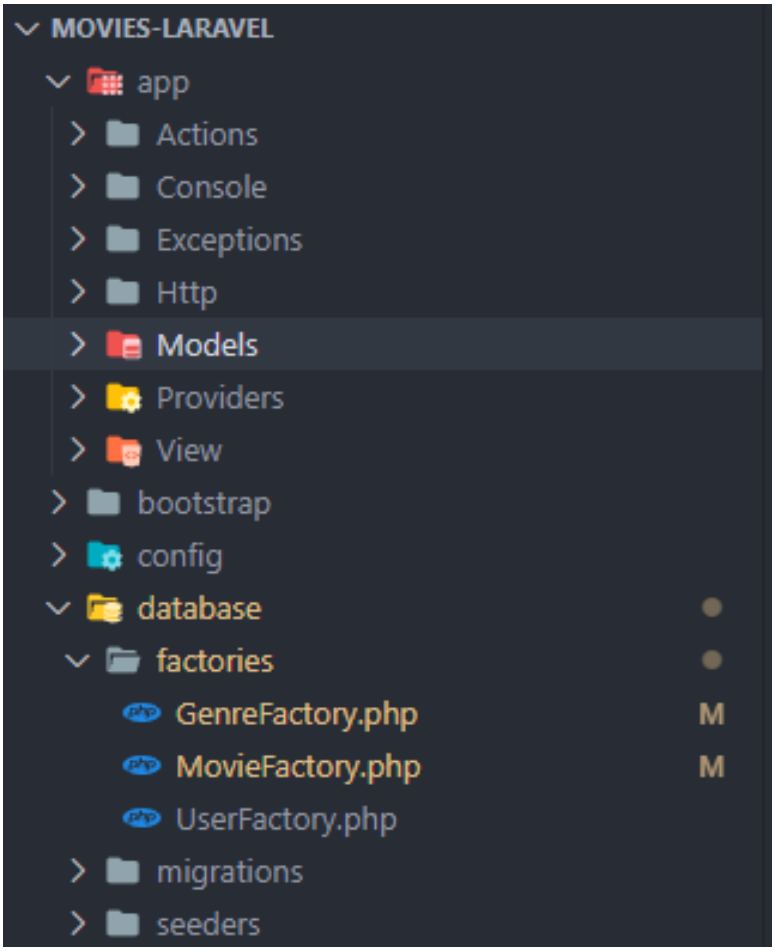
10.3.4. Y en el Modelo de usuarios añadimos los siguientes métodos


use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Fortify\TwoFactorAuthenticatable;
use Laravel\Jetstream\HasProfilePhoto;
use Laravel\Sanctum\HasApiTokens;
use App\Models\Genre;
use App\Models\Movie;
346 references | 0 implementations
class User extends Authenticatable
{
use HasApiTokens;
use HasFactory;
use HasProfilePhoto;
use Notifiable;
use TwoFactorAuthenticatable;



10.4. Factories

10.4.1. Nos dirigimos a la carpeta de factories



10.4.2. Comenzaremos con géneros añadiendo las siguientes líneas

GenreFactory.php M database\factories\GenreFactory.php\...

```php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\Genre;
use App\Models\User;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Genre>
 */
0 references | 0 implementations
class GenreFactory extends Factory
{
    1 reference
    protected $model = Genre::class;
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    1 reference | 0 overrides | prototype
    public function definition(): array
    {
        return [
            'user_id'=> User::factory(),
            'name'=> $this->faker->word()
        ];
```

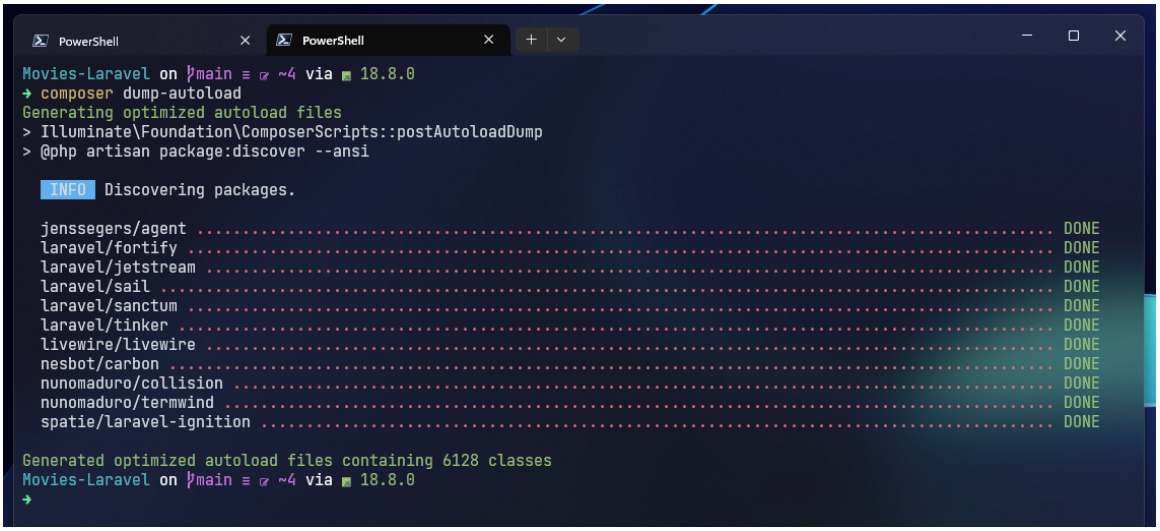
10.4.3. A continuación añadimos las siguientes líneas en película

MovieFactory.php M database\factories\MovieFactory.php\PHP Intelephense\ MovieFactory\ definition

```php
use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\User;
use App\Models\Genre;
use App\Models\Movie;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Movie>
 */
0 references | 0 implementations
class MovieFactory extends Factory
{
    1 reference
    protected $model = Movie::class;
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    1 reference | 0 overrides | prototype
    public function definition(): array
    {
        return [
            'genre_id'=>Genre::factory(),
            'user_id'=>User::factory(),
            'name'=>$this->faker->word(),
            'duration'=>$this->faker->numberBetween(90,120),
            'director'=>$this->faker->word(),
            'box_office'=>$this->faker->boolean()
        ];
    }
```

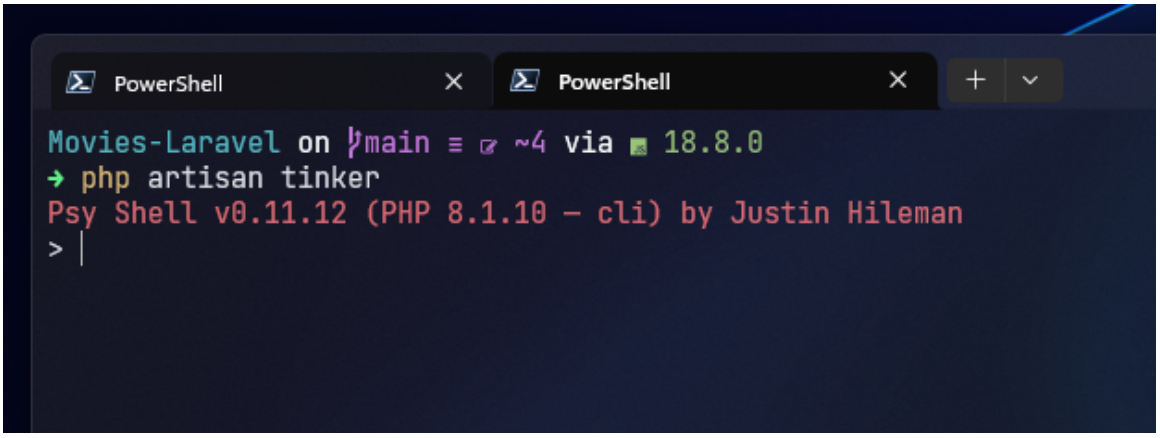
10.4.4. Ahora nos dirigimos a la consola y antes de ejecutar los siguientes pasos es recomendable limpiar la caché de laravel así que utilizaremos el siguiente comando.

```
composer dump-autoload
```



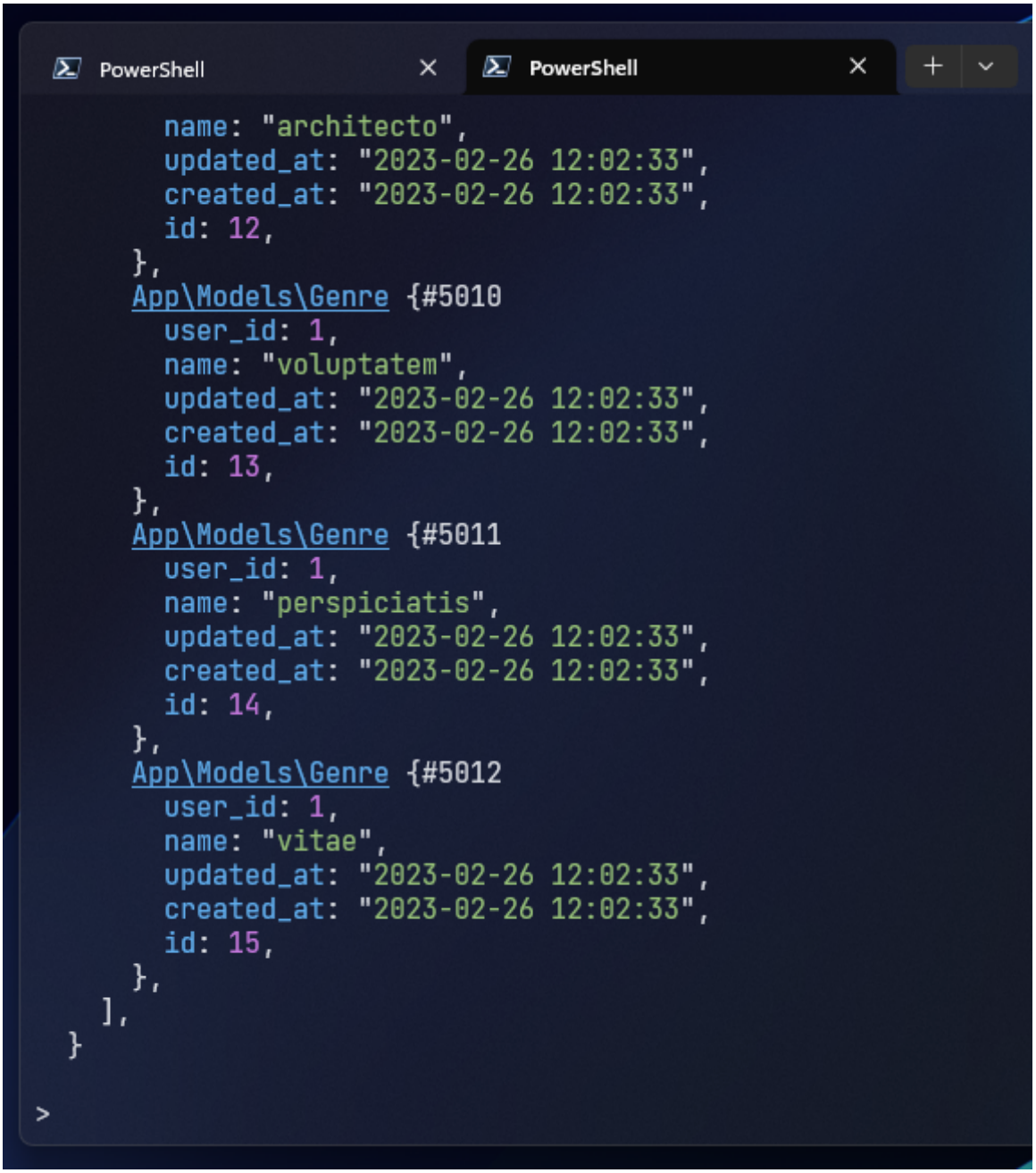
10.4.5. A continuación abrimos la consola de comandos de laravel, Tinker ejecutando el siguiente comando.

```
php artisan tinker
```

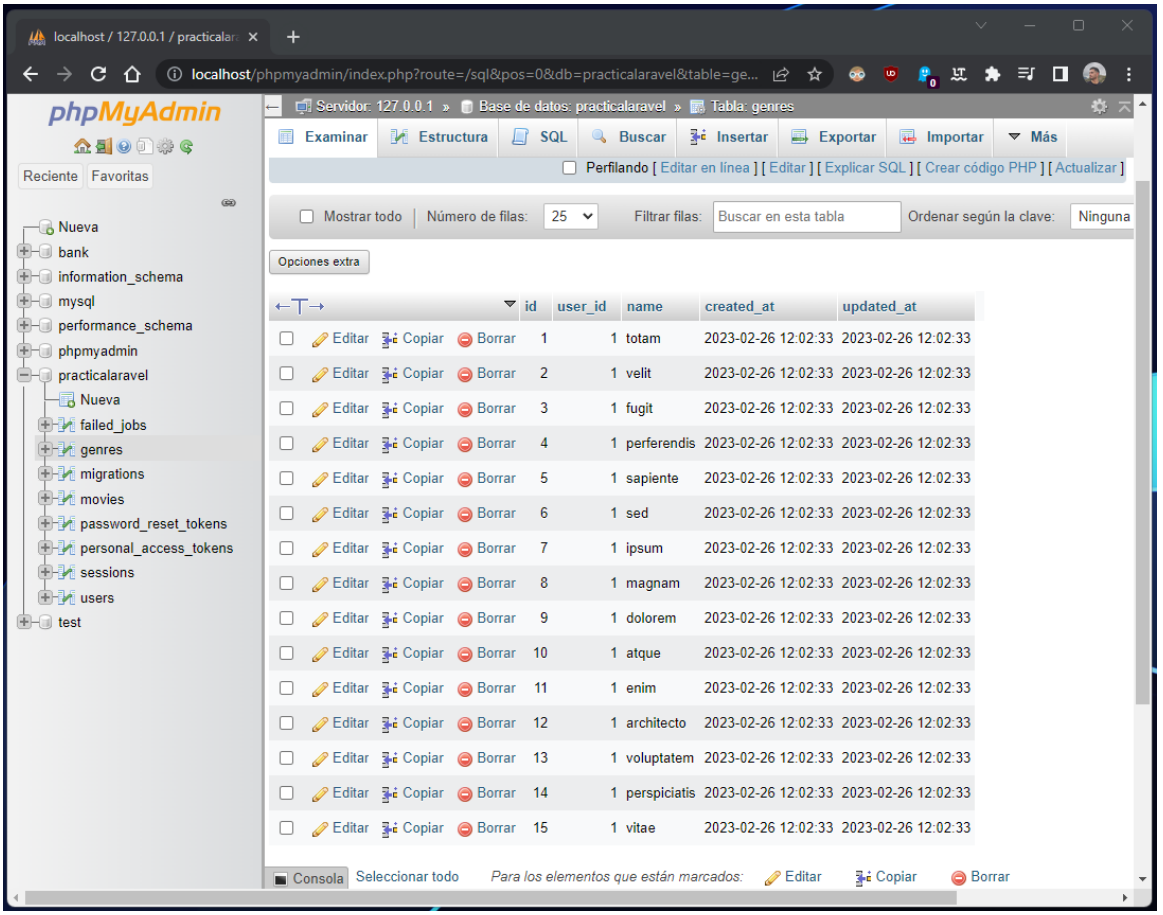


10.4.6. Procedemos a crear 15 géneros de prueba para el usuario 1 . Con el siguiente comando.

```
Genre::factory()-count(15)-create([‘user_id’ =>1]);
```

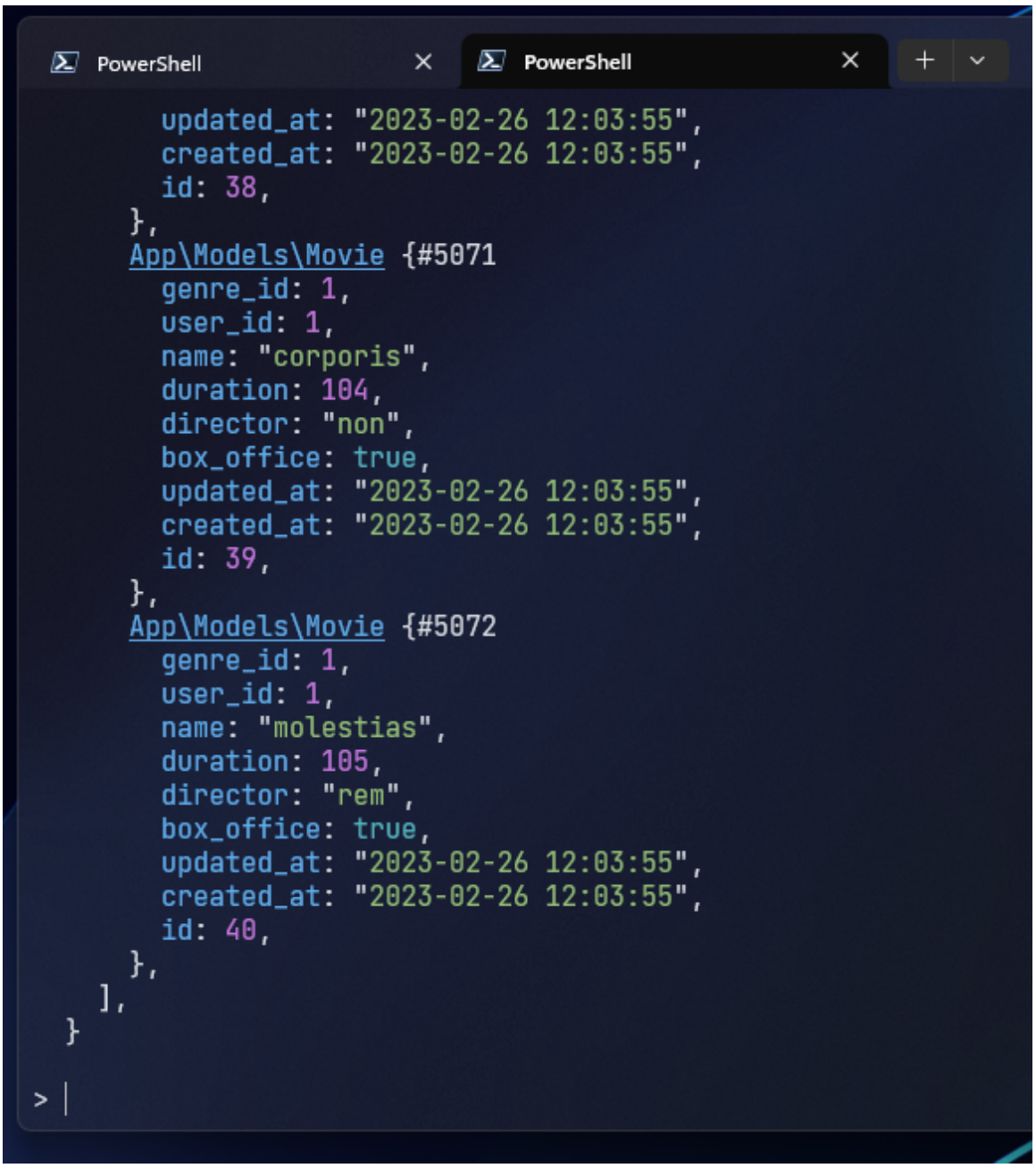


Como vemos a continuación se han generado correctamente.

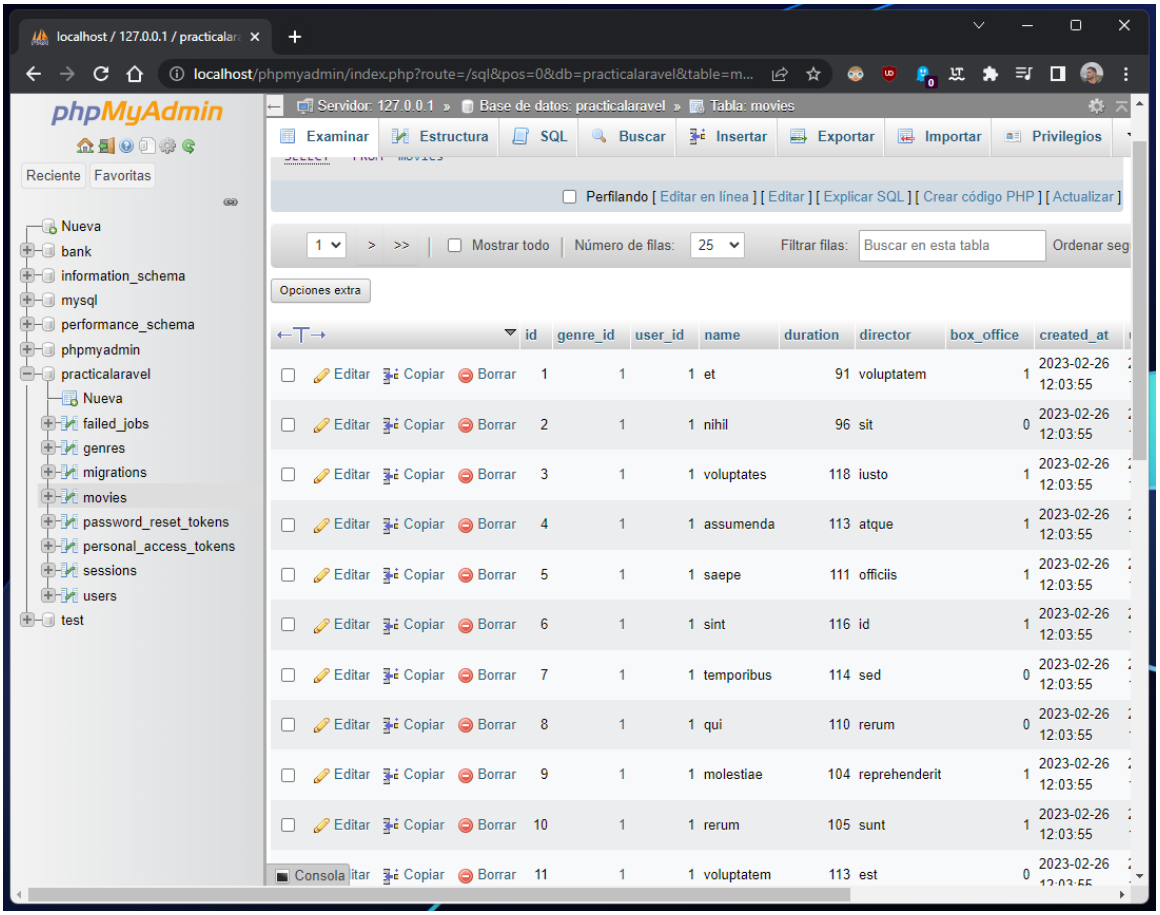


10.4.7. Procedemos ahora a crear 40 películas para el usuario 1, con el género 1 que hemos creado.

```
Movie::factory()-count(40)-create([‘user_id’ => 1, ‘genre_id’ =>1]);
```



Como vemos se ha creado correctamente.

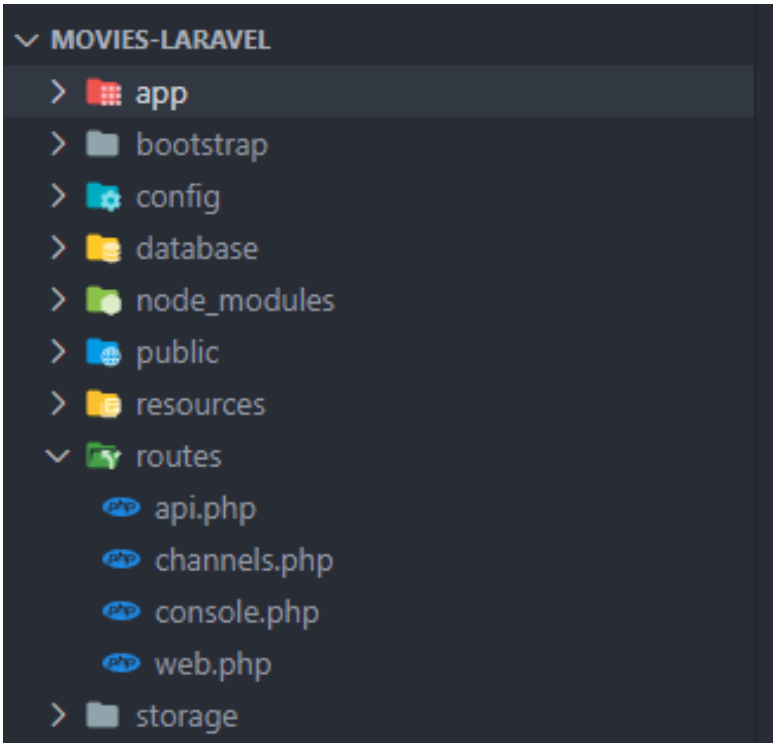


10.4.8. Para acabar podemos salirnos de tinker con el siguiente comando.

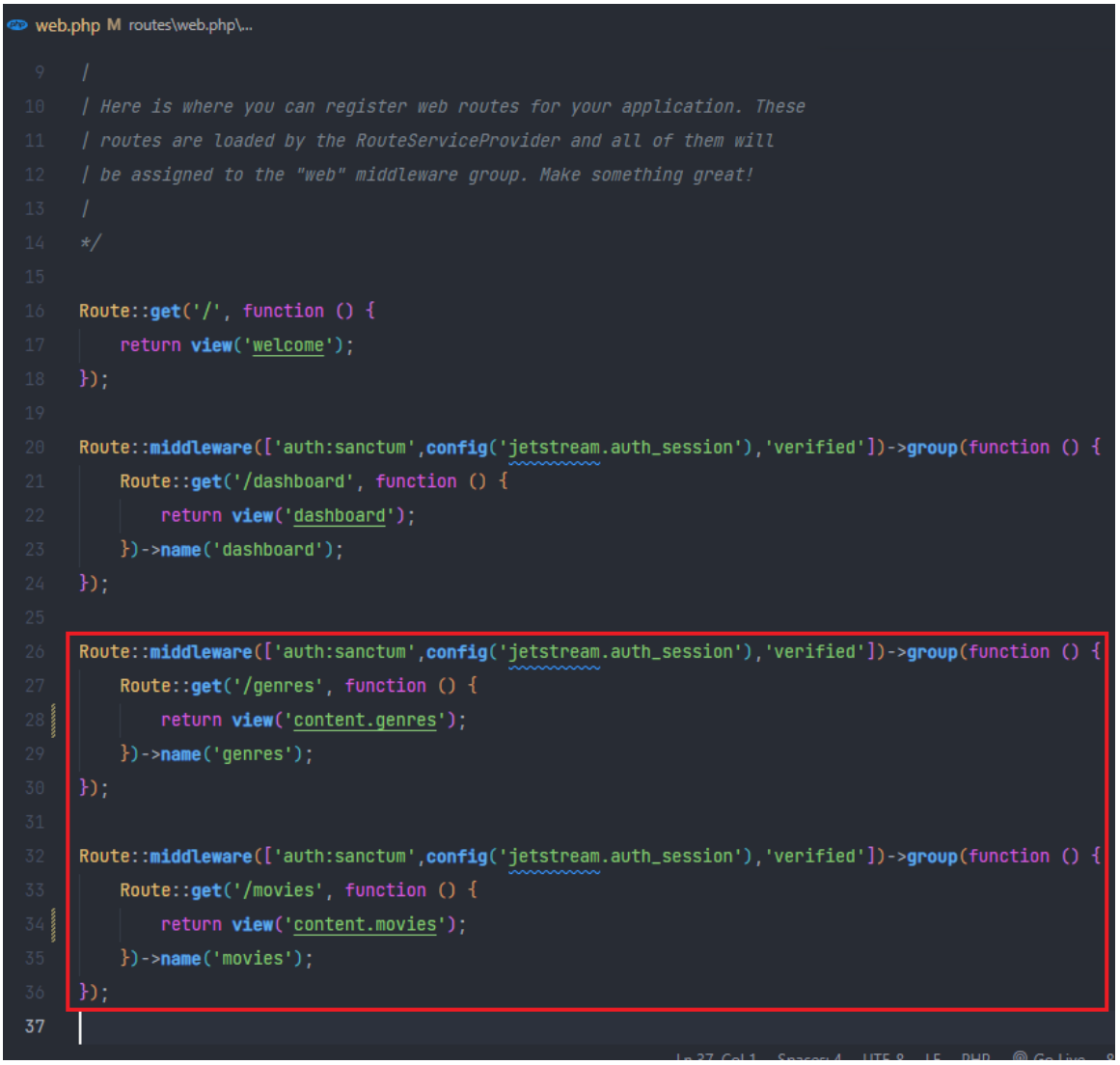
```
exit
```

11. Rutas

11.1. Lo primero será dirigirnos a la carpeta routes.

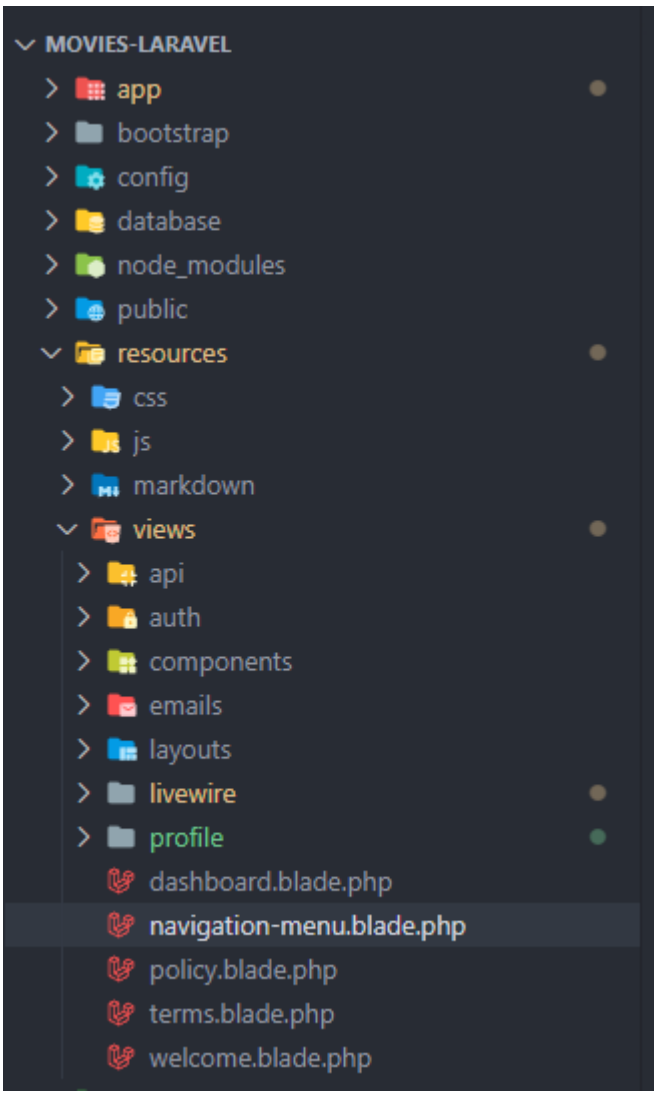


11.2. Y a continuación en el archivo web.php añadiremos las siguientes rutas. Como se ve devolveremos las vistas dentro de una carpeta que más adelante crearemos que se llamará content



12. Creación de los enlaces a Géneros y Películas

12.1. Nos dirigimos a navigation-menu.blade.php en resources/views



12.2. Para que se muestren los enlaces a nuestros géneros y películas vamos a ”Navigation Links” y agregamos lo siguiente:

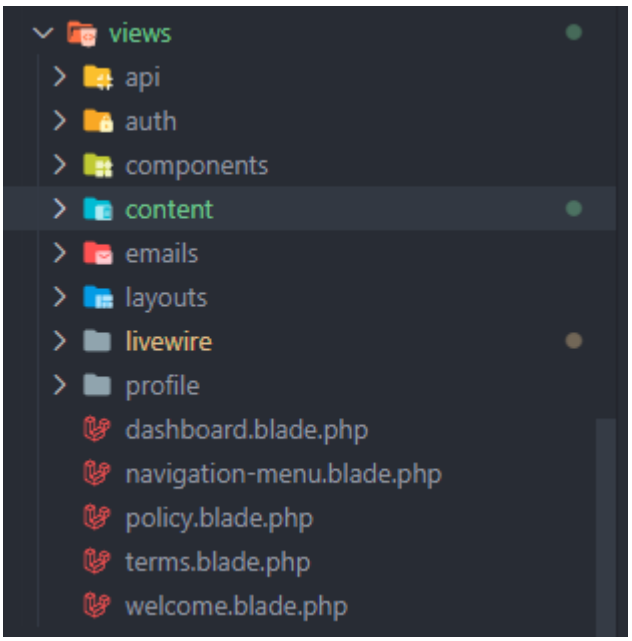
```
<div class="flex">
    <!-- Logo -->
    <div class="shrink-0 flex items-center">
        <a href="{{ route('dashboard') }}">
            <x-application-mark class="block h-9 w-auto" />
        </a>
    </div>

    <!-- Navigation Links -->
    <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">
        <x-nav-link href="{{ route('dashboard') }}" :active="request()->routeIs('dashboard')">
            {{ __('Dashboard') }}
        </x-nav-link>

        <x-nav-link href="{{ route('genres') }}" :active="request()->routeIs('genres')">
            {{ __('Genres') }}
        </x-nav-link>

        <x-nav-link href="{{ route('movies') }}" :active="request()->routeIs('movies')">
            {{ __('Movies') }}
        </x-nav-link>
    </div>
</div>
```

12.3. Para que estos se muestren también en la versión móvil debemos bajar hasta “Responsive Navigation Menu” y agregar lo siguiente:

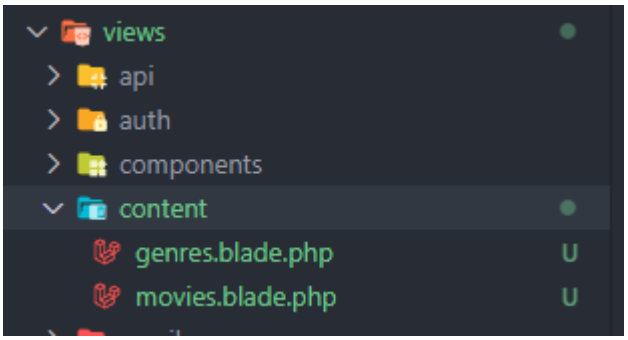
```
<!-- Responsive Navigation Menu -->
<div :class="{'block': open, 'hidden': ! open}" class="hidden sm:hidden">
    <div class="pt-2 pb-3 space-y-1">
        <x-responsive-nav-link href="{{ route('dashboard') }}" :active="request()->routeIs('dashboard')">
            {{ __('Dashboard') }}
        </x-responsive-nav-link>

        <x-responsive-nav-link href="{{ route('genres') }}" :active="request()->routeIs('genres')">
            {{ __('Genres') }}
        </x-responsive-nav-link>

        <x-responsive-nav-link href="{{ route('movies') }}" :active="request()->routeIs('movies')">
            {{ __('Movies') }}
        </x-responsive-nav-link>
    </div>
```

13. Creación de vistas

13.1. Como comentamos en la introducción lo primero será hacer la carpeta content dentro de vistas.



13.2. Y dentro de la carpeta content crearemos la vista para géneros y películas. Es decir crearemos dos ficheros llamados genres.blade.php y movies.blade.php



13.3. Empezamos con la vista de géneros. Insertamos lo siguiente.

genres.blade.php U resources\views\content\genres.blade.php\...

```blade
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Genres') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
                <livewire:genres />
            </div>
        </div>
    </div>
</x-app-layout>

```

13.4. Ahora a continuación procedemos con películas

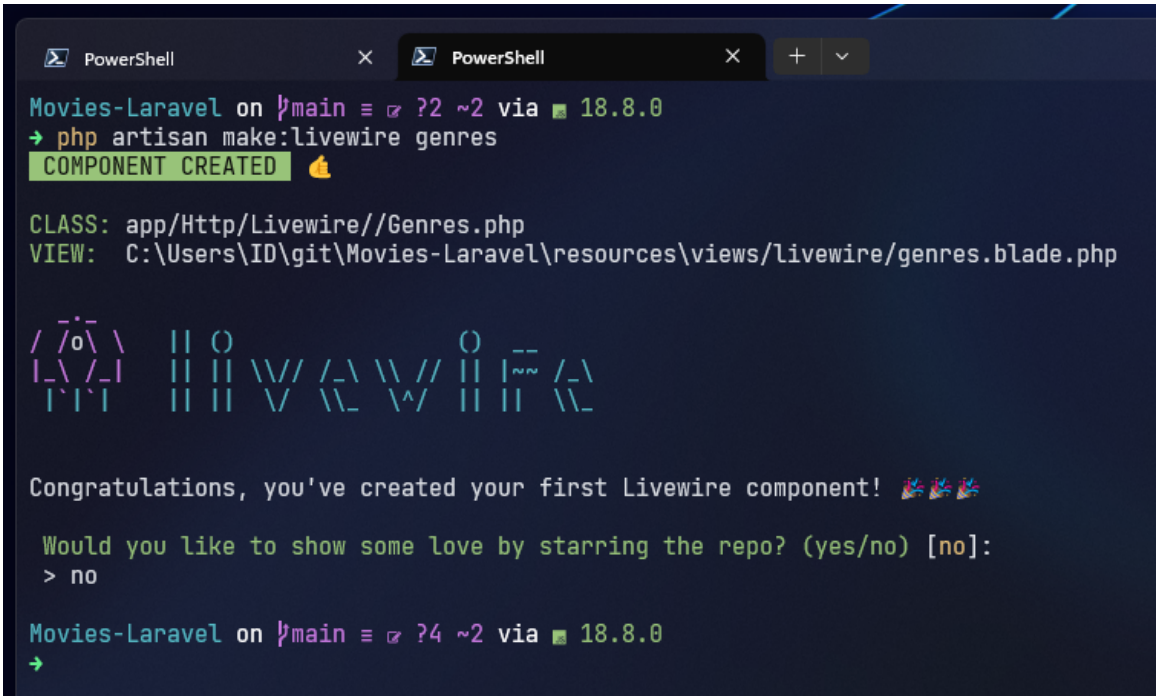
movies.blade.php U resources\views\content\movies.blade.php\...

```blade
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Movies') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
                <livewire:movies />
            </div>
        </div>
    </div>
</x-app-layout>

```

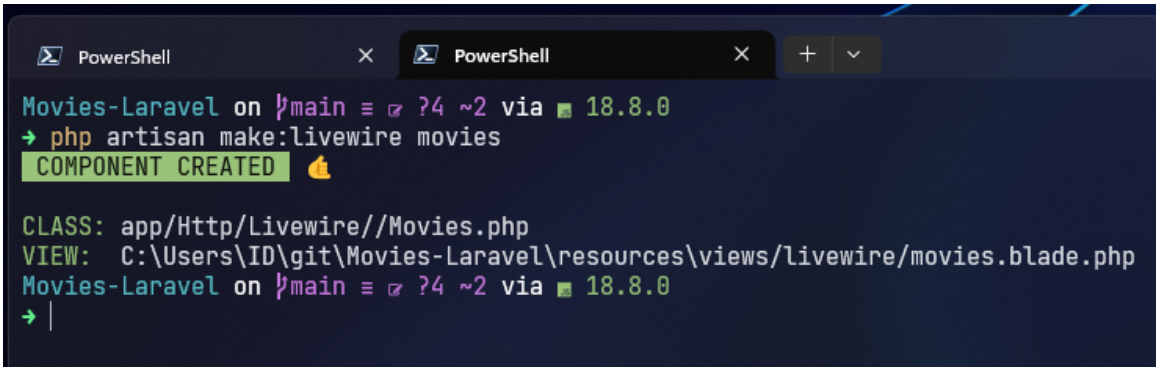
14. Creación de componentes

14.1. A continuación ejecutamos los siguientes comandos para crear los componentes de livewire para ambas vistas.

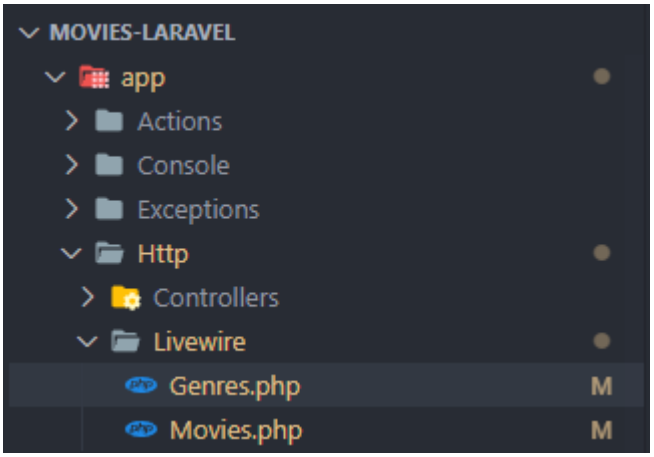
```
php artisan make:livewire genres
```




```
php artisan make:livewire movies
```



14.2. A continuación nos dirigimos a la carpeta app/Http/Livewire



14.3. Y programamos el componente de géneros con lo siguiente:

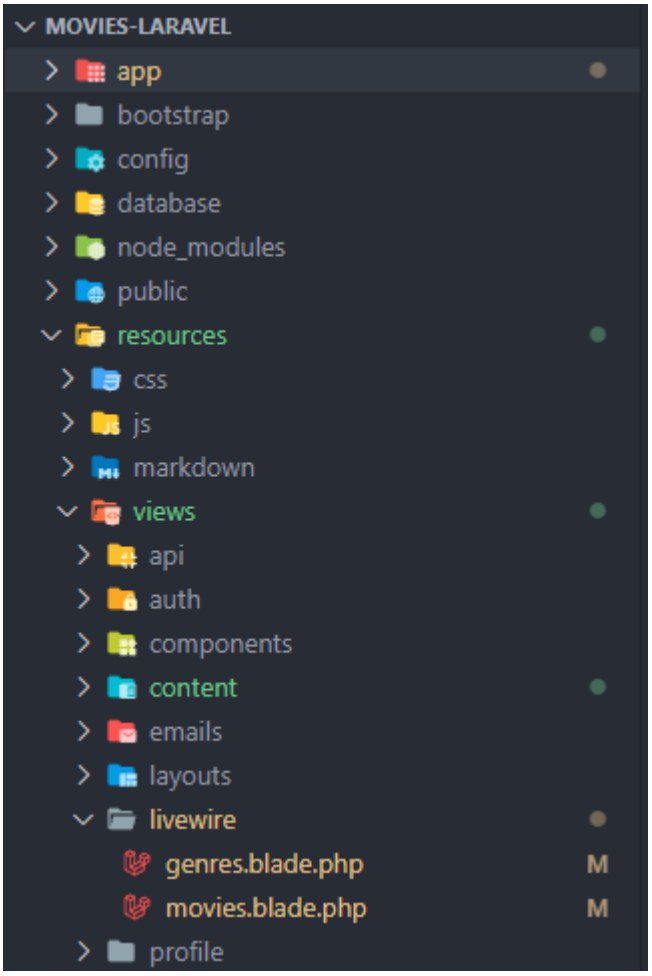
Genres.php M app\Http\Livewire\Genres.php\...

```php
<?php

namespace App\Http\Livewire;

use Livewire\Component;
use App\Models\Genre;
use Livewire\WithPagination;

0 references | 0 implementations
class Genres extends Component
{
    use WithPagination;

    0 references | 0 overrides
    public function render()
    {
        $genres = Genre::where('user_id', auth()->user()->id)->paginate(10);

        return view('livewire.genres', [
            'genres' => $genres,
        ]);
    }
}
```

14.4. Seguimos con el componente de películas:

15. Creación de las vistas

15.1. Nos dirigimos a:



15.2. Comenzamos editando géneros añadiendo el siguiente código

genres.blade.php resources\views\livewire\genres.blade.php\...

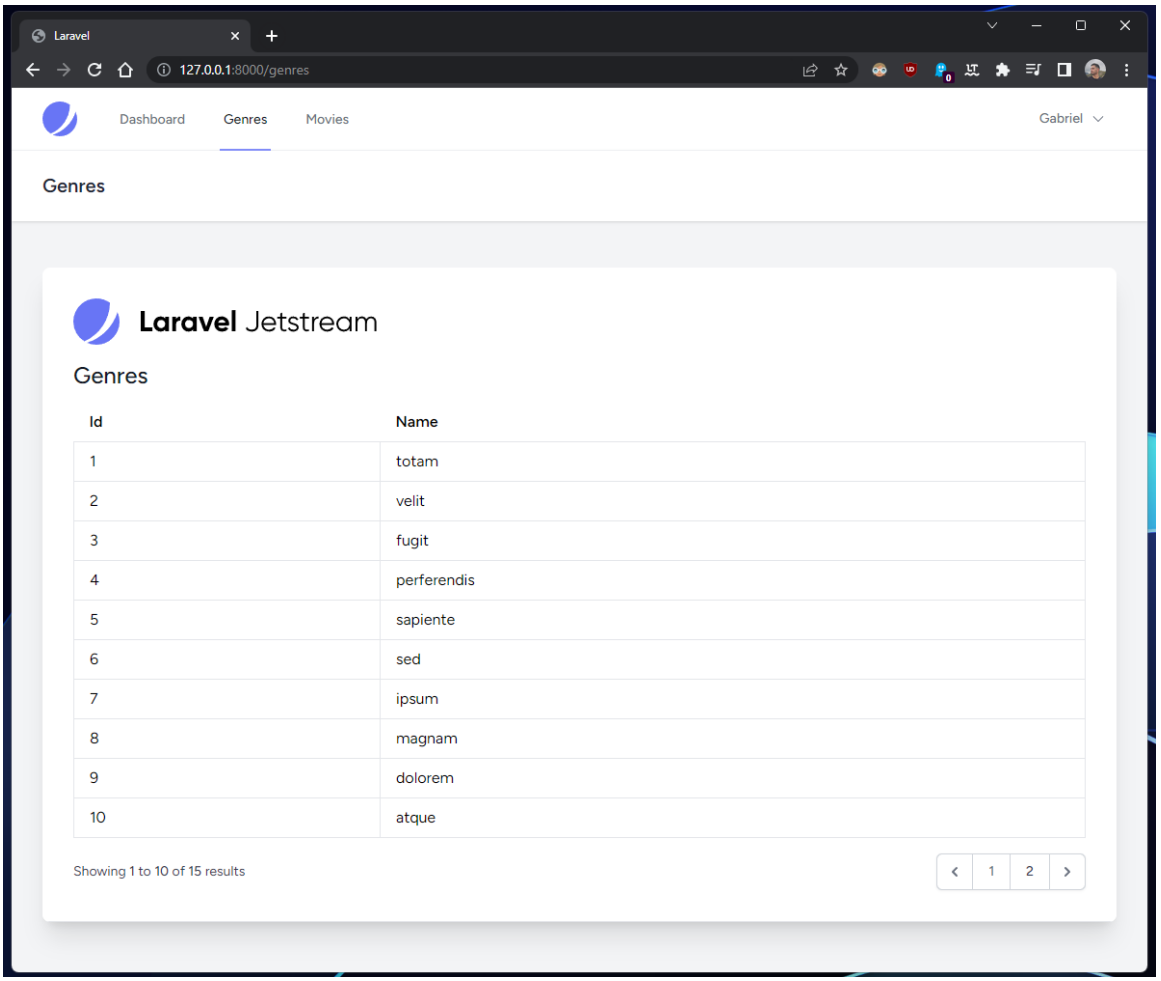
```blade
<div class="p-2 lg:p-8 bg-white border-b border-gray-200">
    <x-application-logo class="block h-12 w-auto" />

    <div class="mt-4 text-2xl font-medium text-gray-900">
        <div>Genres</div>
    </div>
    <div class="mt-3">
        <table class="table-auto w-full">
            <thead>
                <tr>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Id</div>
                    </th>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Name</div>
                    </th>
                </tr>
            </thead>
            <tbody>
                @foreach ($genres as $genre)
                    <tr>
                        <td class="rounded border px-4 py-2">{{$genre->id}}</td>
                        <td class="rounded border px-4 py-2">{{$genre->name}}</td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    </div>
    <div class="mt-4">{{$genres}}</div>
</div>

```

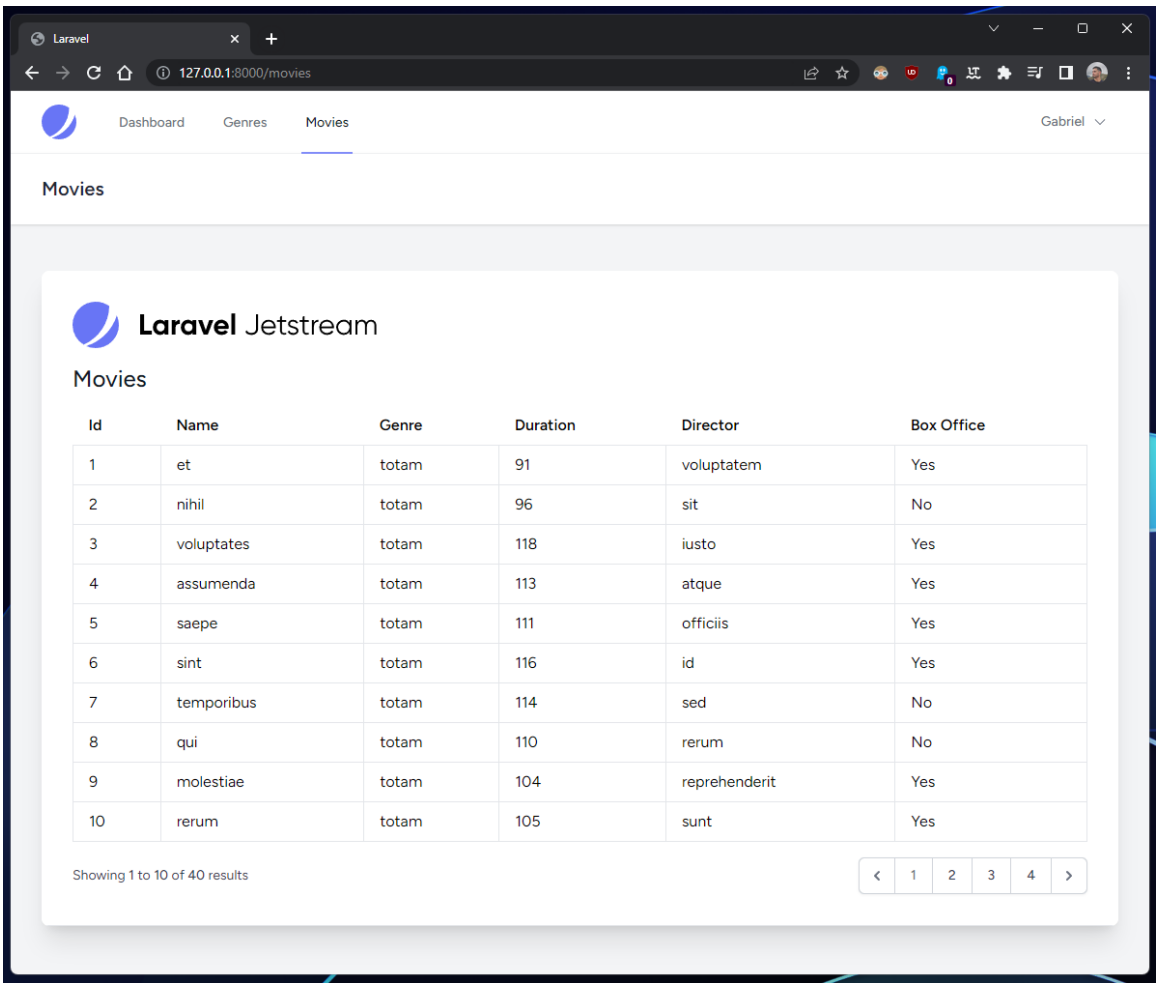
15.3. Y en el caso de películas pondremos el siguiente

```blade
<div class="p-2 lg:p-8 bg-white border-b border-gray-200">
    <x-application-logo class="block h-12 w-auto" />

    <div class="mt-4 text-2xl font-medium text-gray-900">
        <div>Movies</div>
    </div>
    <div class="mt-3">
        <table class="table-auto w-full">
            <thead>
                <tr>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Id</div>
                    </th>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Name</div>
                    </th>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Genre</div>
                    </th>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Duration</div>
                    </th>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Director</div>
                    </th>
                    <th class="px-4 py-2">
                        <div class="flex items-center">Box Office</div>
                    </th>
                </tr>
            </thead>
            <tbody>
                @foreach ($movies as $movie)
                    <tr>
                        <td class="rounded border px-4 py-2">{{ $movie->id }}</td>
                        <td class="rounded border px-4 py-2">{{ $movie->name }}</td>
                        <td class="rounded border px-4 py-2">{{ $genres->find($movie->genre_id)->name }}</td>
                        <td class="rounded border px-4 py-2">{{ $movie->duration }}</td>
                        <td class="rounded border px-4 py-2">{{ $movie->director }}</td>
                        <td class="rounded border px-4 py-2">{{ $movie->box_office ? 'Yes' : 'No' }}</td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    </div>
    <div class="mt-4">{{ $movies }}</div>
</div>
```

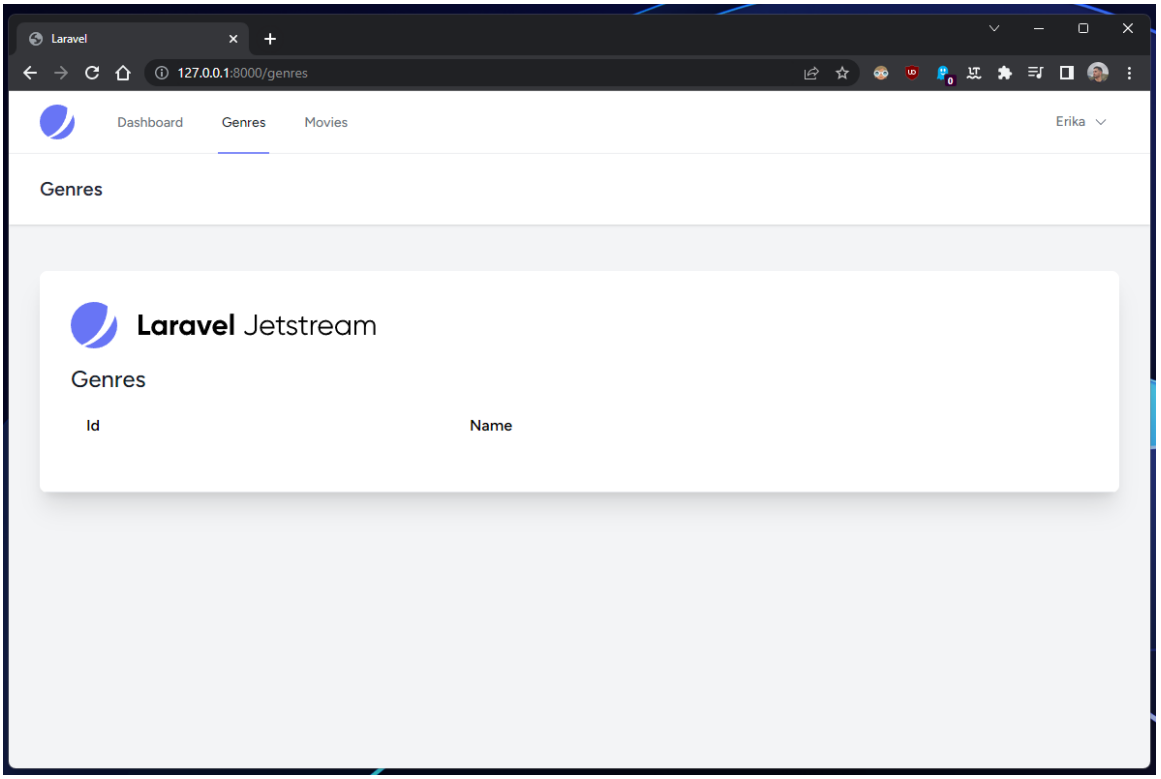
15.4. Probamos el proyecto

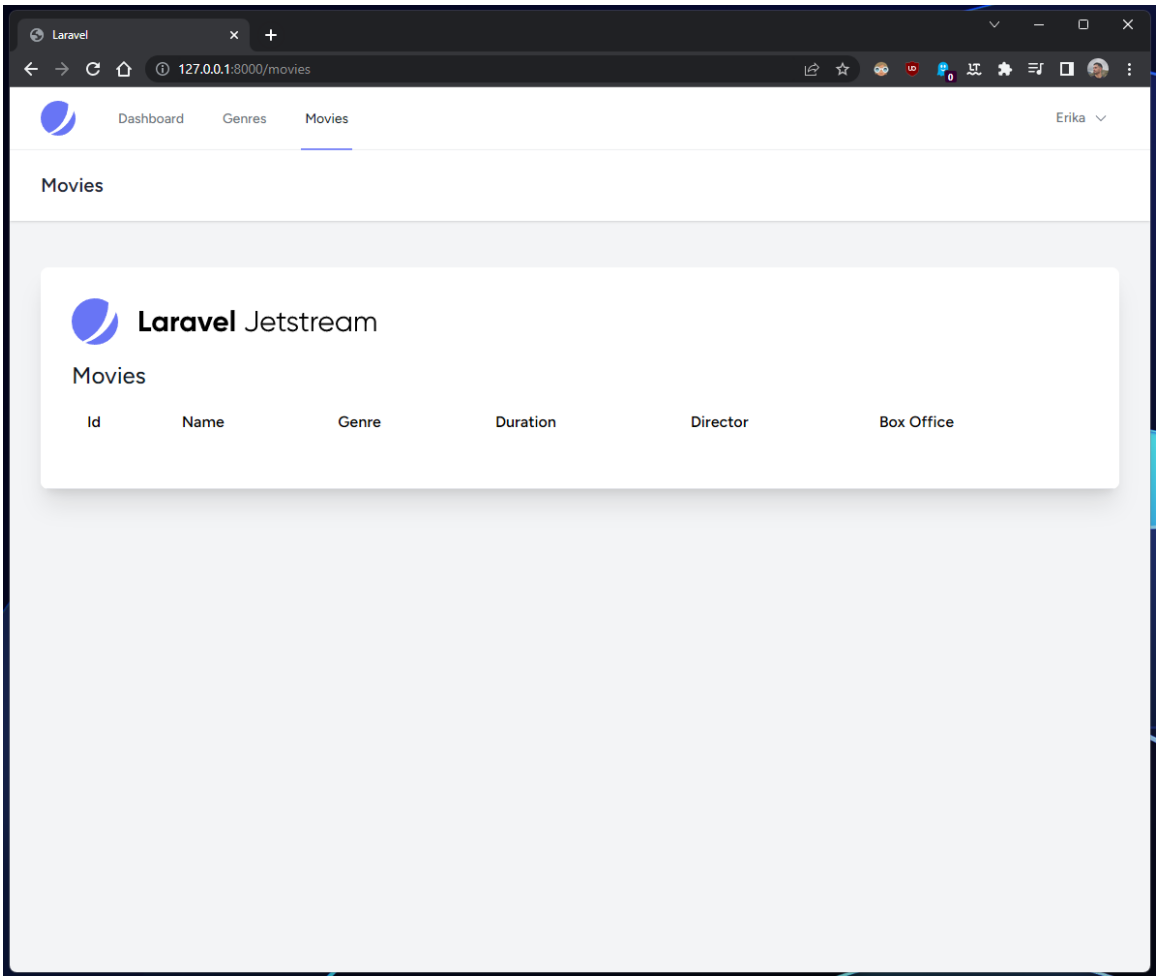
A continuación volvemos a abrir el proyecto con el primer usuario que creamos. Como vemos tenemos los datos que hemos generado con el factory en ambas tablas y además la paginación está funcionando correctamente.


Laravel
127.0.0.1:8000/genres
Dashboard
Genres
Movies
Gabriel
Genres
Laravel Jetstream
Genres
Id
Name
1
totam
2
velit
3
fugit
4
perferendis
5
sapiente
6
sed
7
ipsum
8
magnam
9
dolorem
10
atque
Showing 1 to 10 of 15 results
1
2



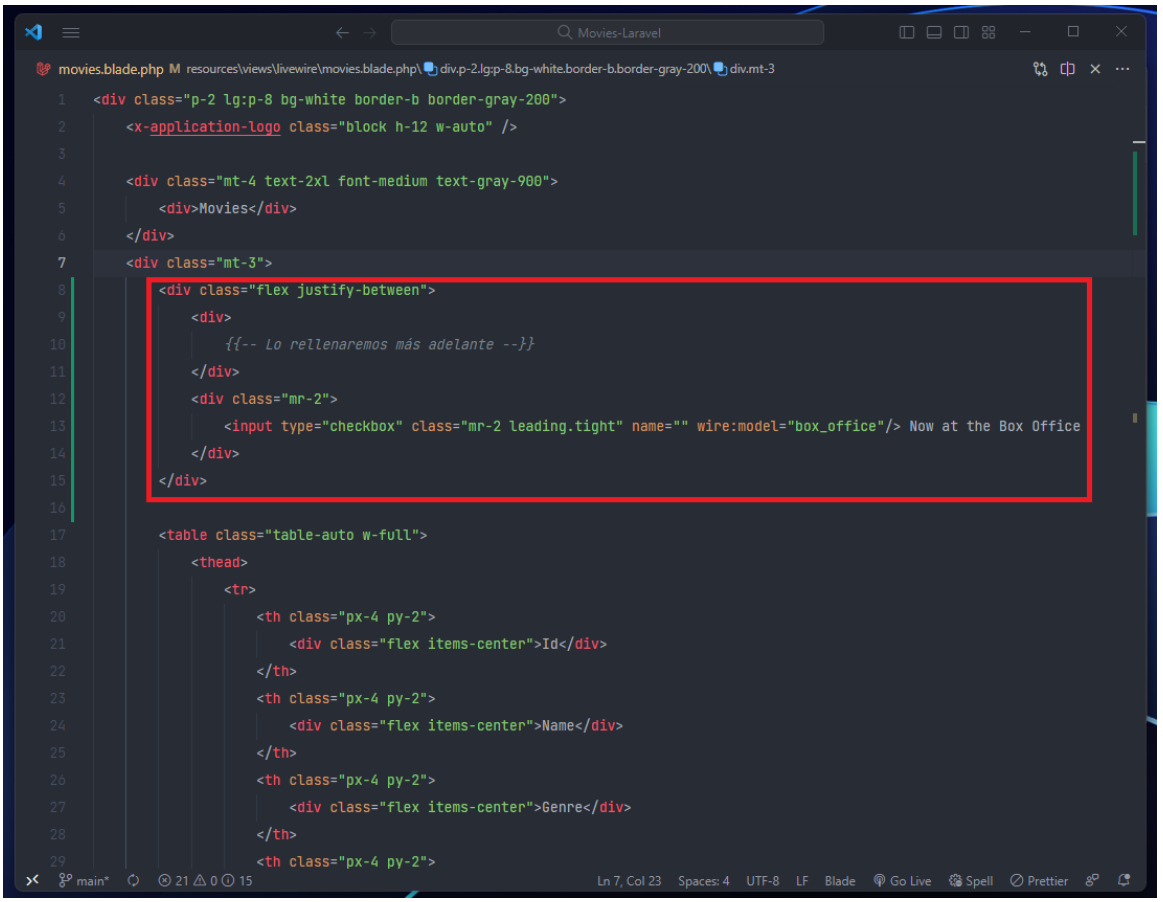
Por el contrario si probamos a hacer otro usuario este no tiene datos ya que no tiene datos insertados para ese usuario. Por lo que todo está funcionando como deseamos.


Laravel
127.0.0.1:8000/genres
Dashboard
Genres
Movies
Erika
Genres
Laravel Jetstream
Genres
Id
Name

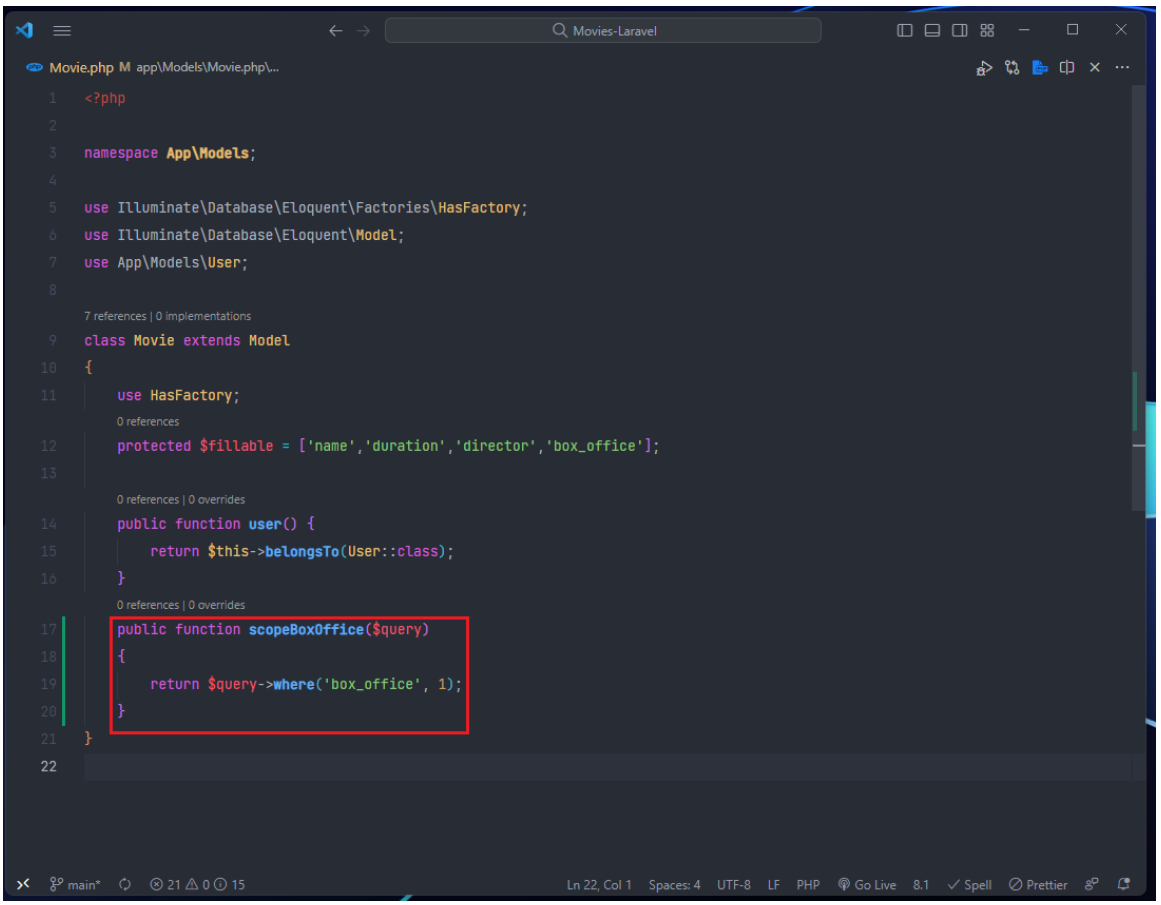

Laravel
127.0.0.1:8000/movies
Dashboard
Genres
Movies
Erika
Movies
Laravel Jetstream
Movies
Id
Name
Genre
Duration
Director
Box Office

16. Añadimos Filtro de películas en taquilla

16.1. A continuación vamos a agregar el filtro de taquillas en la vista de películas. Para eso volvemos a la vista de películas (resources\views\livewire\movies.blade.php) y agregaremos lo siguiente.

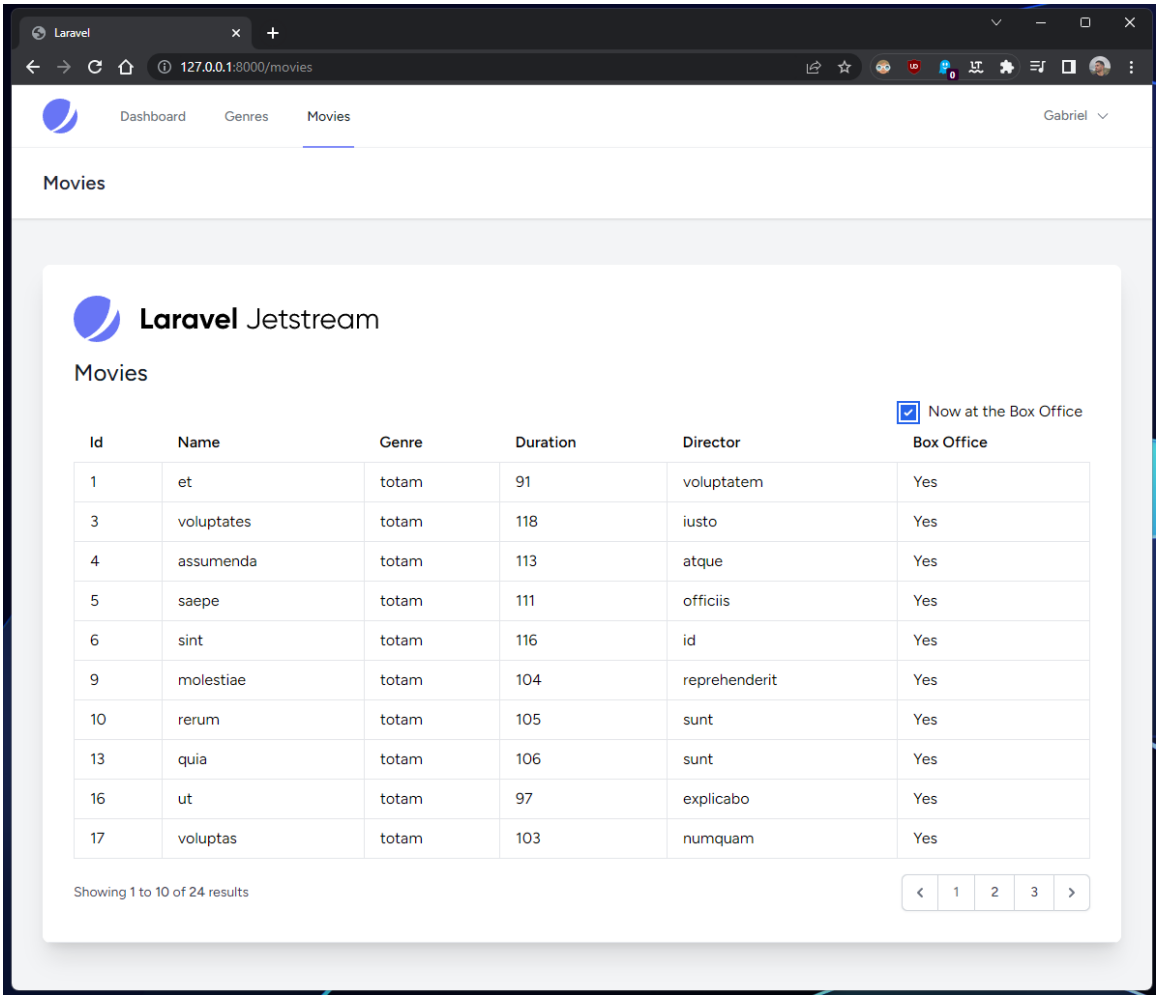


16.2. A continuación nos dirigimos al modelo de películas (app\Models\Movie.php) y creamos el siguiente método.



16.3. A continuación nos dirigimos al componente de películas y lo modificaremos de la siguiente forma.

16.4. Una vez hecho esto volvemos a nuestra página web y probamos el filtro



Como vemos funciona correctamente. Además de la forma en la que lo hemos hecho si por ejemplo tenemos 4 páginas y estamos es la página 4 y lo activamos y en número de páginas se reduce a 3 no pasa nada porque la web nos redirige siempre a la primera al activar el filtro.

17. Añadir cuadro de búsqueda

17.1. Para empezar iremos a la vista de géneros (resources\views\livewire\genres.blade.php) y agregaremos lo siguiente

```
<div class="mt-4 text-2xl font-medium text-gray-900">
    <div>Genres</div>
</div>
<div class="mt-3">
    <div class="flex justify-between">
        <div>
            <input wire:model.debounce.300ms="search" type="search" placeholder="Search"
                class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none
                focus:shadow-outline"
                name="">
        </div>
    </div>
    <table class="table-auto w-full">
        <thead>
            <tr>
                <th class="px-4 py-2">
                    <div class="flex items-center">Id</div>
                </th>
                <th class="px-4 py-2">
                    <div class="flex items-center">Name</div>
                </th>
            </tr>
        </thead>
```

17.2. A continuación haremos lo mismo pero para la vista de películas (resources\views\livewire\movies.blade.php)

```
<div class="mt-3">
    <div class="flex justify-between">
        <div>
            <input wire:model.debounce.300ms="search" type="search" placeholder="Search"
                class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none
                focus:shadow-outline"
                name="">
        </div>
        <div class="mr-2">
            <input type="checkbox" class="mr-2 leading.tight" name="" wire:model="box_office" /> Now at the
            Box Office
        </div>
    </div>
```

17.3. Una vez hechas las vistas procedemos a hacer los componentes. Comenzaremos con el de géneros. Nos dirigimos a (app\Http\Livewire\Genres.php) y agregamos lo siguiente.

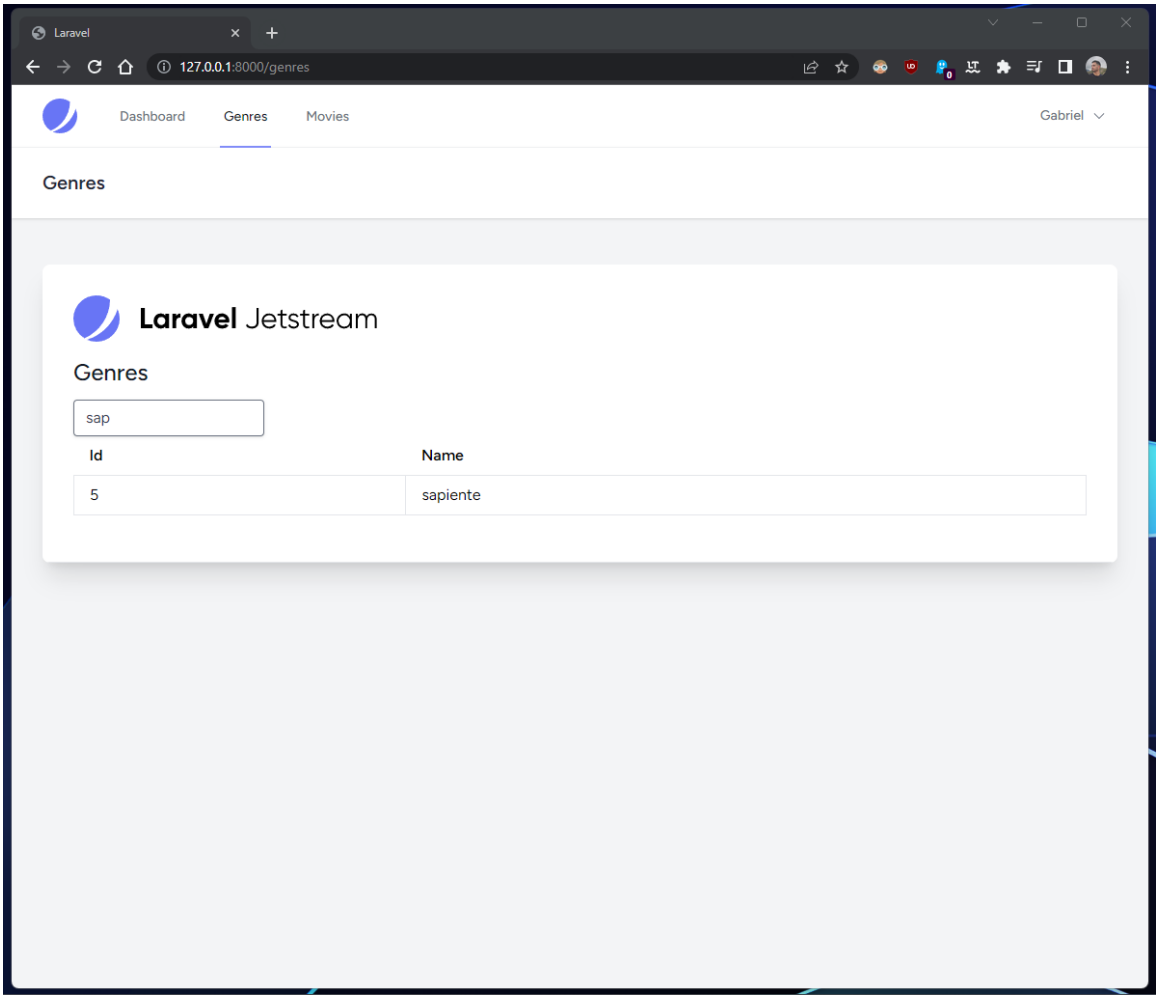
```php
class Genres extends Component
{

    use WithPagination;


    2 references
    public $search;

    0 references | 0 overrides
    public function render()
    {
        $genres = Genre::where('user_id', auth()->user()->id)
            ->when($this->search, function ($query) {
                return $query->where(function ($query) {
                    $query->where('name', 'like', '%' . $this->search . '%')
                });
            });

        $genres = $genres->paginate(10);
        return view('livewire.genres', [
            'genres' => $genres,
        ]);
    }
    0 references | 0 overrides
    public function updatingSearch(){
        $this->resetPage();
    }
}
```

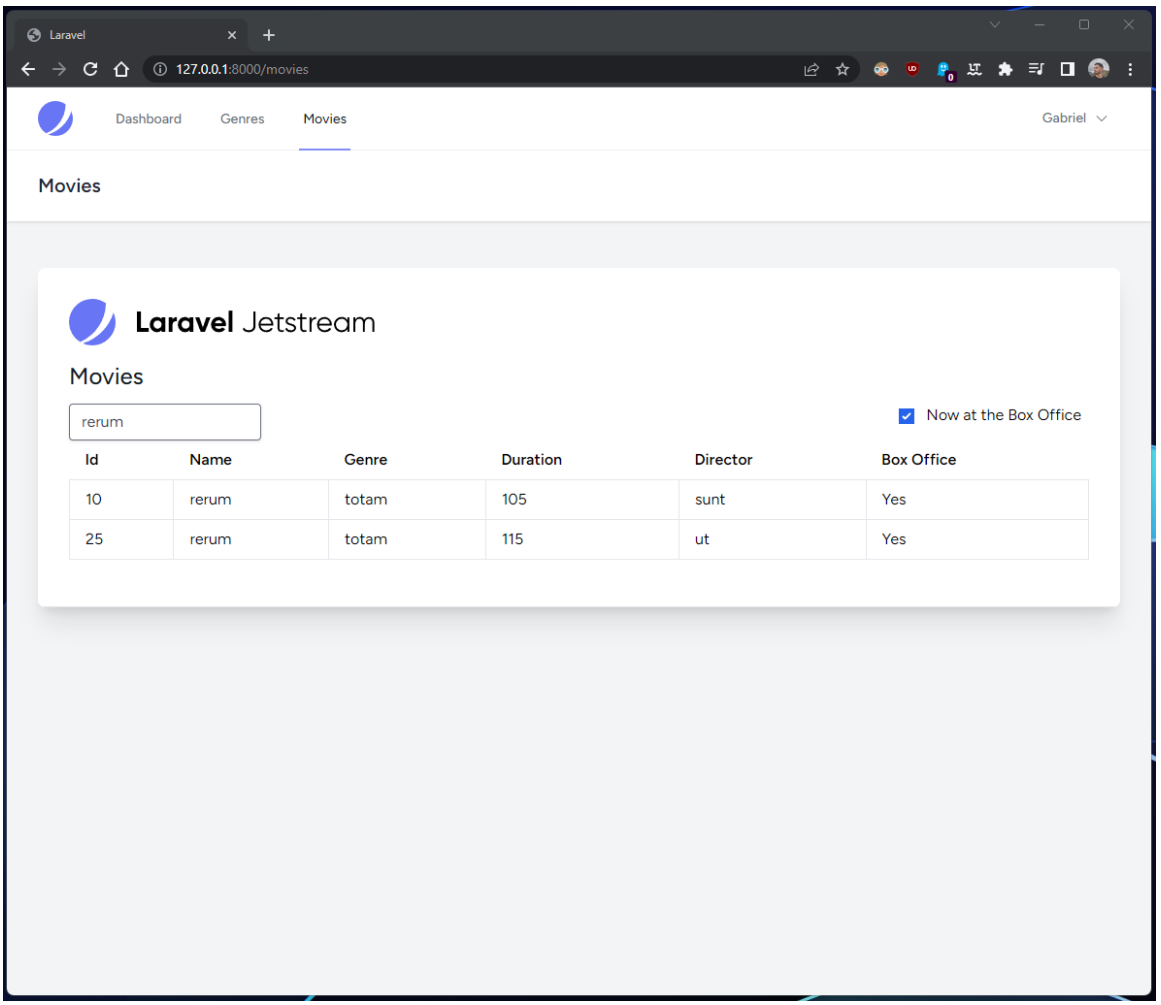
17.4. Y a continuación procedemos a hacer los mismo pero con el componente de películas. Nos dirigimos a (app\Http\Livewire\Movies.php) y agregamos lo siguiente.

```php
class Movies extends Component
{
    use WithPagination;

    1 reference
    public $box_office;

    3 references
    public $search;

    0 references | 0 overrides
    public function render()
    {
        $movies = Movie::where('user_id', auth()->user()->id)
            ->when($this->search, function ($query) {
                return $query->where(
                    function ($query) {
                            $query->where('name', 'like', '%' . $this->search . '%')
                                ->orWhere('director', 'like', '%' . $this->search . '%');
                        }
                );
            })

            ->when($this->box_office, function ($query) {
                return $query->boxOffice();
            });

        $movies = $movies->paginate(10);
        $genres = Genre::where('user_id', auth()->user()->id);
```

```
        return view('livewire.movies', [
            'movies' => $movies,
            'genres' => $genres
        ]);
    }

    0 references | 0 overrides
    public function updatingBoxOffice()
    {
        $this->resetPage();
    }
    0 references | 0 overrides
    public function updatingSearch()
    {
        $this->resetPage();
    }
```

17.5. Con esto ya tenemos implementadas nuestras búsquedas en ambos casos. Volvemos a nuestra web para probarlas. Como vemos funcionan como se esperaba.

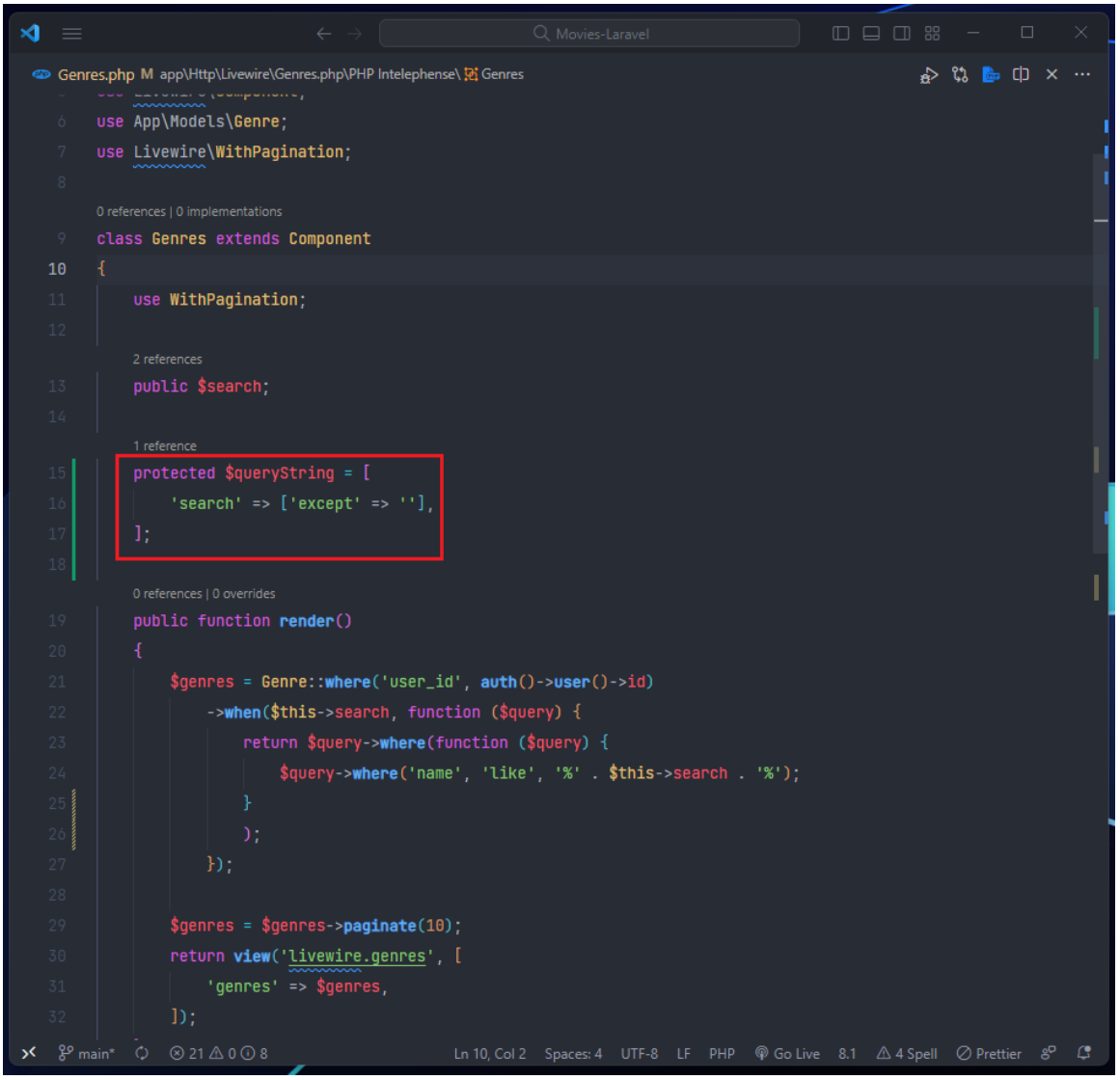

Laravel
127.0.0.1:8000/genres
Dashboard
Genres
Movies
Gabriel
Genres
Laravel Jetstream
Genres
sap
Id
Name
5
sapiente



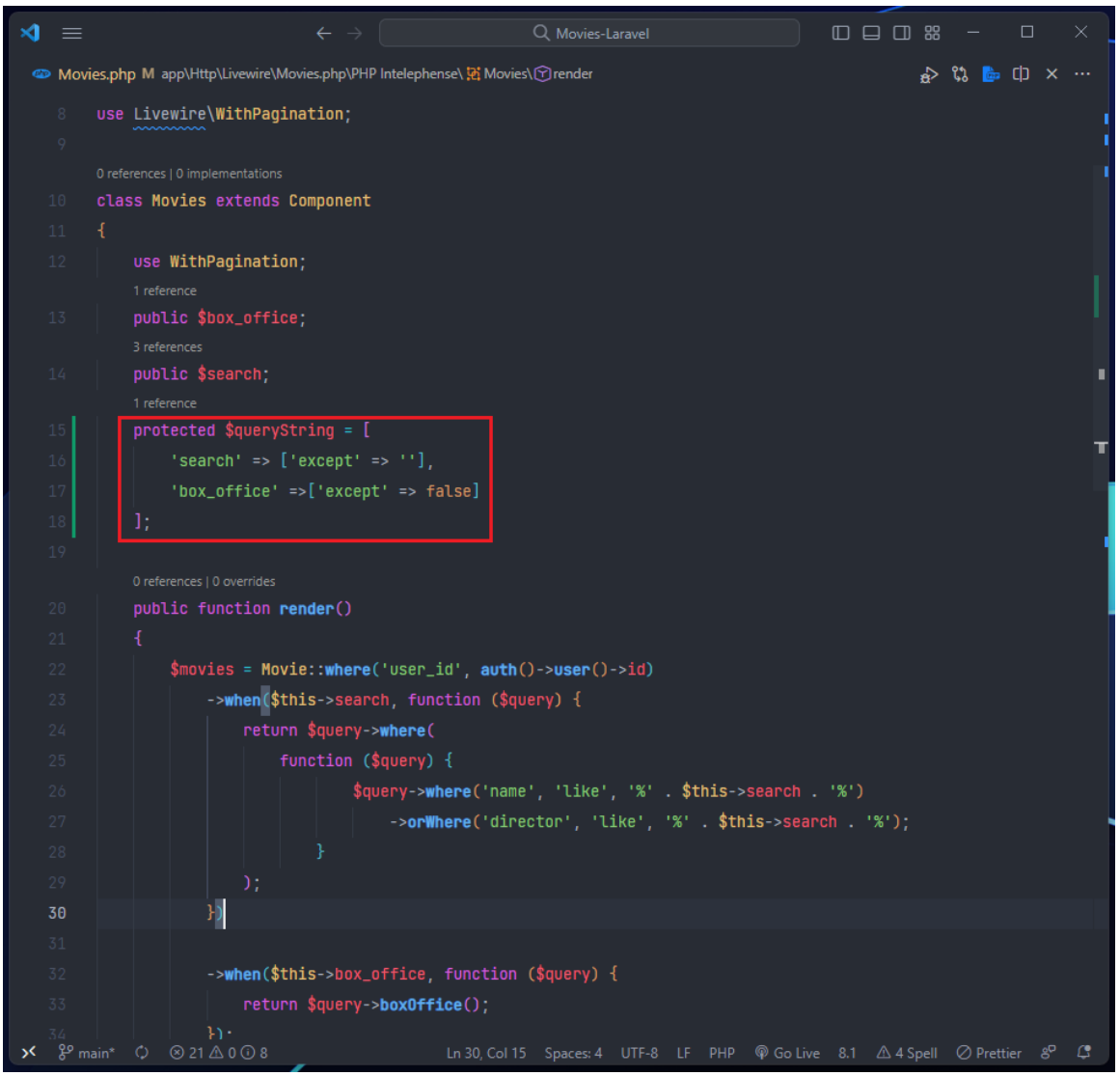
18. Query String

A continuación vamos a mejorar las url de nuestra página web para que al hacer búsquedas o aplicar filtros estas se muestran en la url de nuestra página. Pa eso haremos los siguientes pasos

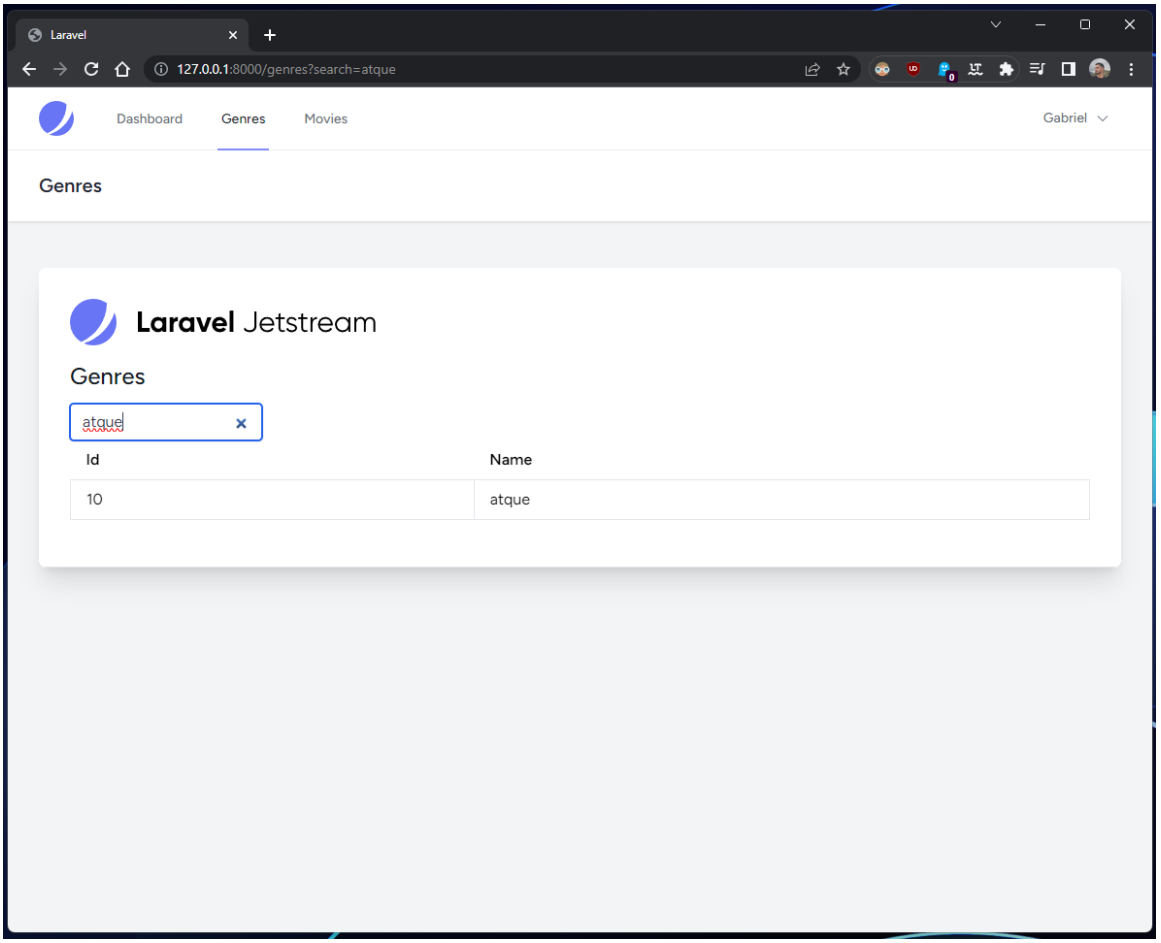
18.1. Empezaremos con el componente de géneros (app\Http\Livewire\Genres.php). Y añadiremos lo siguiente.

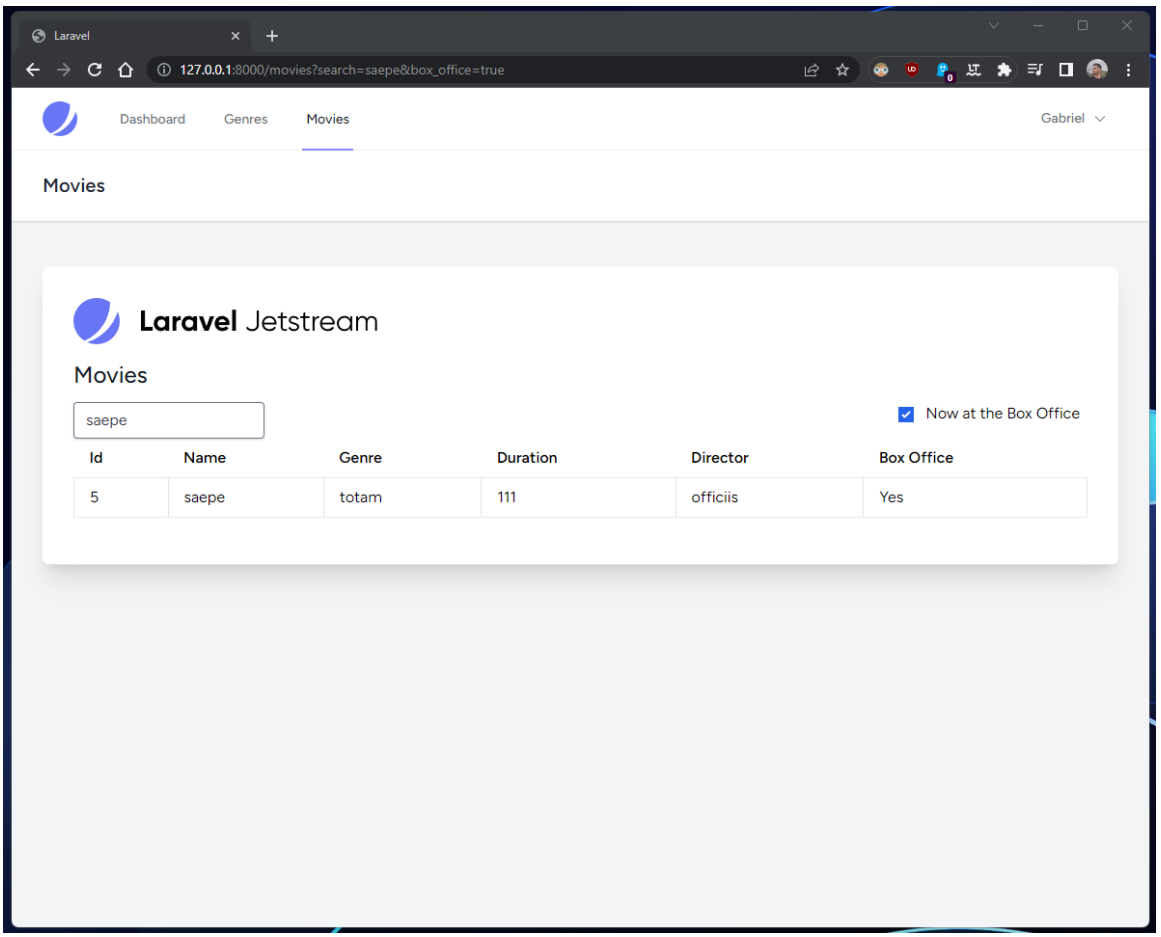


18.2. A continuación hacemos lo mismo pero para el componente películas (app\Http\Livewire\Movies.php).



18.3. Por último volvemos al navegador y probamos el funcionamiento. Como vemos a continuación en la url podemos ver datos del cuadro de búsqueda y los filtros que tenemos activos


Laravel
127.0.0.1:8000/genres?search=atque
Dashboard
Genres
Movies
Gabriel
Genres
Laravel Jetstream
Genres
atque
Id
Name
10
atque

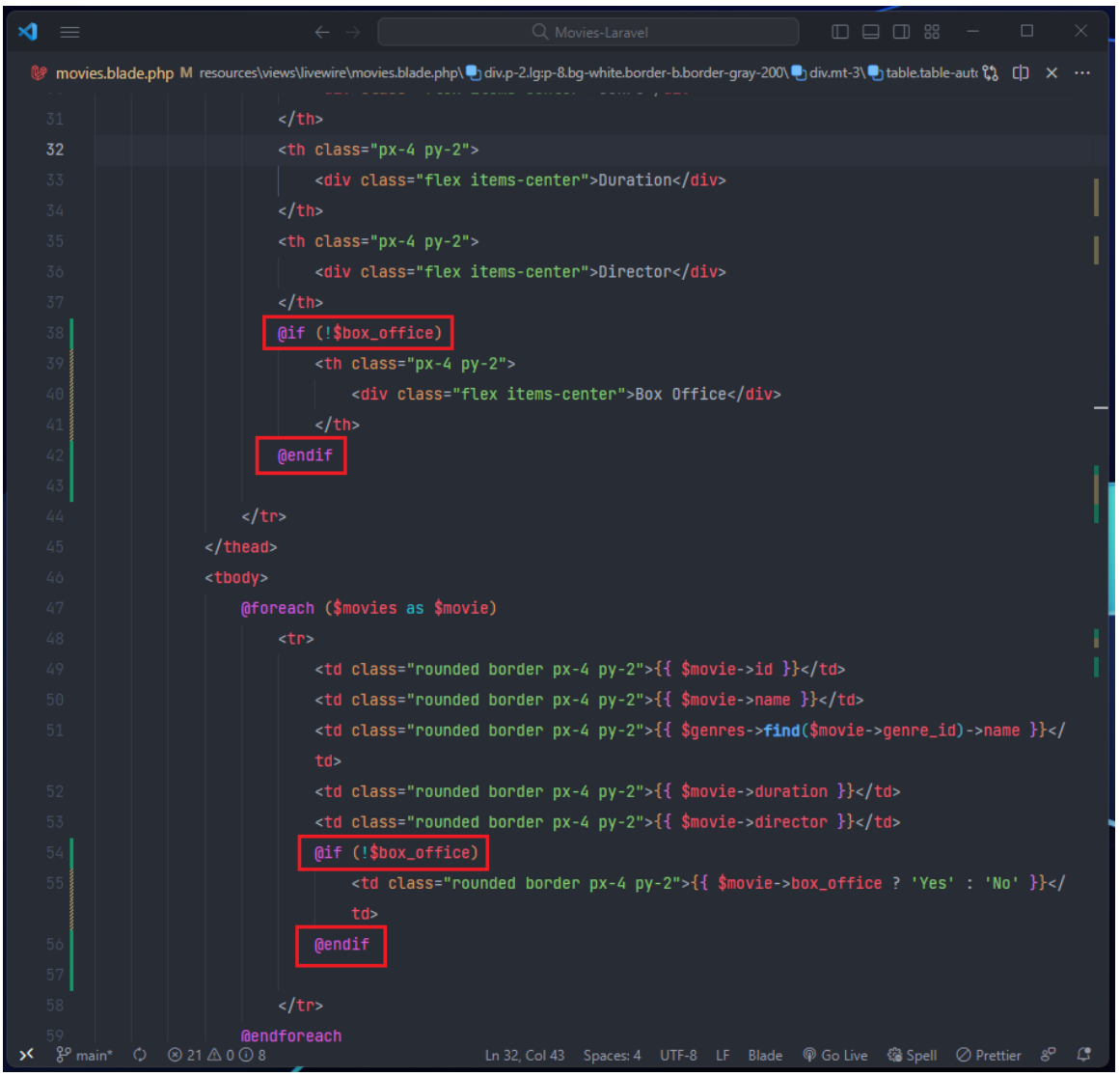


19. Mejorando filtro en taquilla

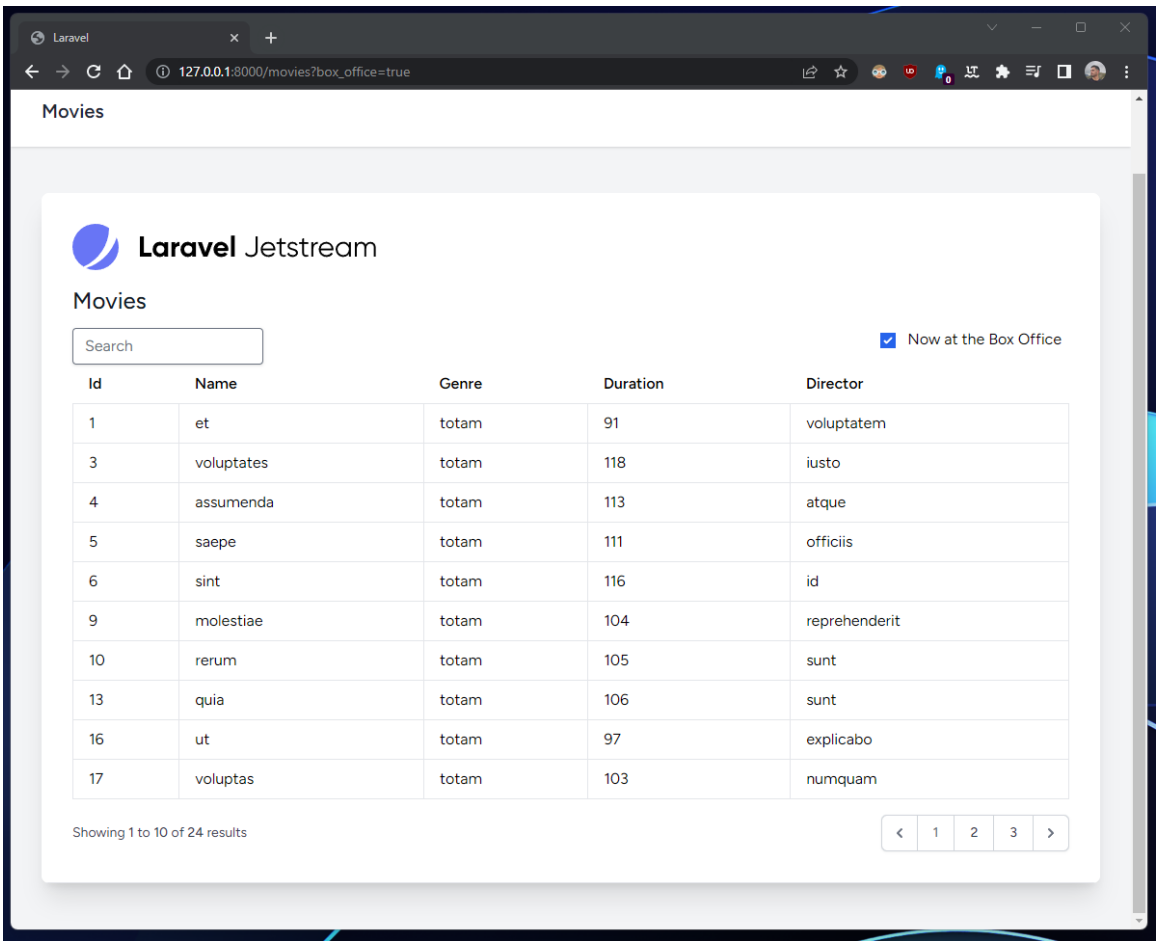
Cuándo filtremos por las películas ahora en taquilla (Now at the Box Office) no tiene mucho sentido mostrar la columna de box office porque ya sabemos que el dato que contendrá es si.

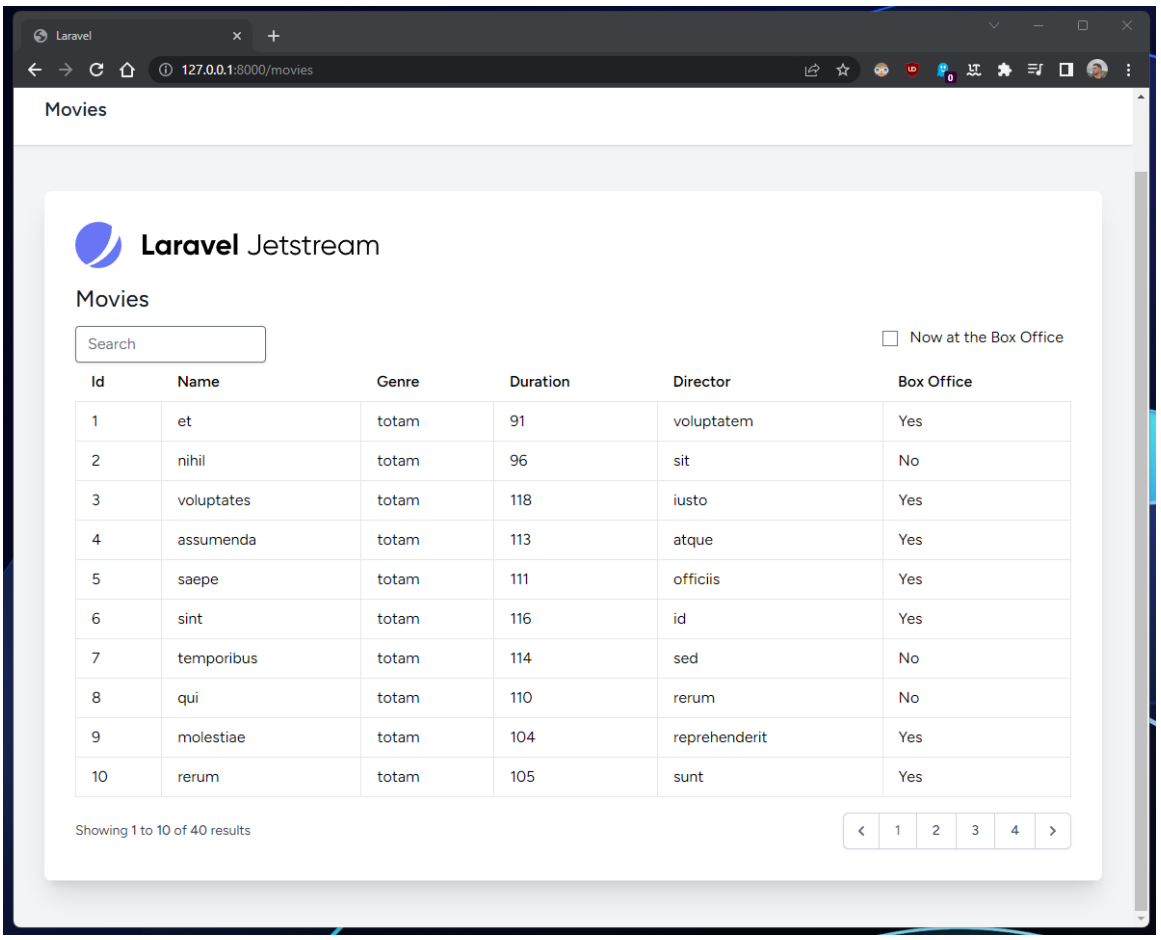
Entonces vamos a hacer que se oculte cuándo el filtro esté activo.

19.1. Para ello iremos a la vista de peliculas (resources\views\livewire\movies.blade.php) y agregaremos las siguientes líneas



19.2. Y abrimos la web para probarlo. Si marcamos el filtro la columna desaparece, pero en cuanto la volvemos a marcar esta aparece.

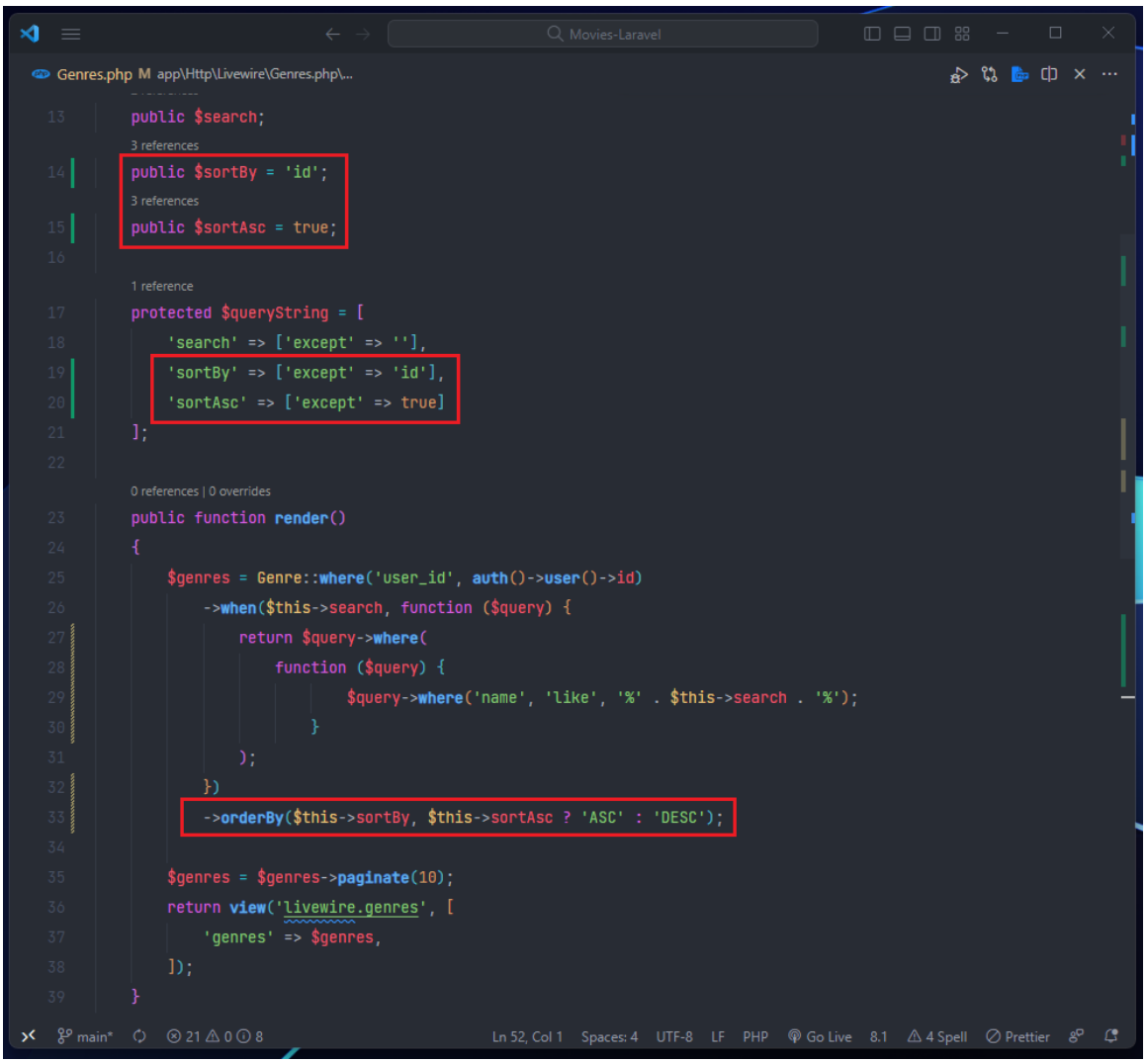

Laravel
127.0.0.1:8000/movies?box_office=true
Movies
Laravel Jetstream
Movies
Search
Now at the Box Office
Id Name Genre Duration Director
1 et totam 91 voluptatem
3 voluptates totam 118 iusto
4 assumenda totam 113 atque
5 saepe totam 111 officiis
6 sint totam 116 id
9 molestiae totam 104 reprehenderit
10 rerum totam 105 sunt
13 quia totam 106 sunt
16 ut totam 97 explicabo
17 voluptas totam 103 numquam
Showing 1 to 10 of 24 results
1 2 3



20. Ordenación ascendente y descente

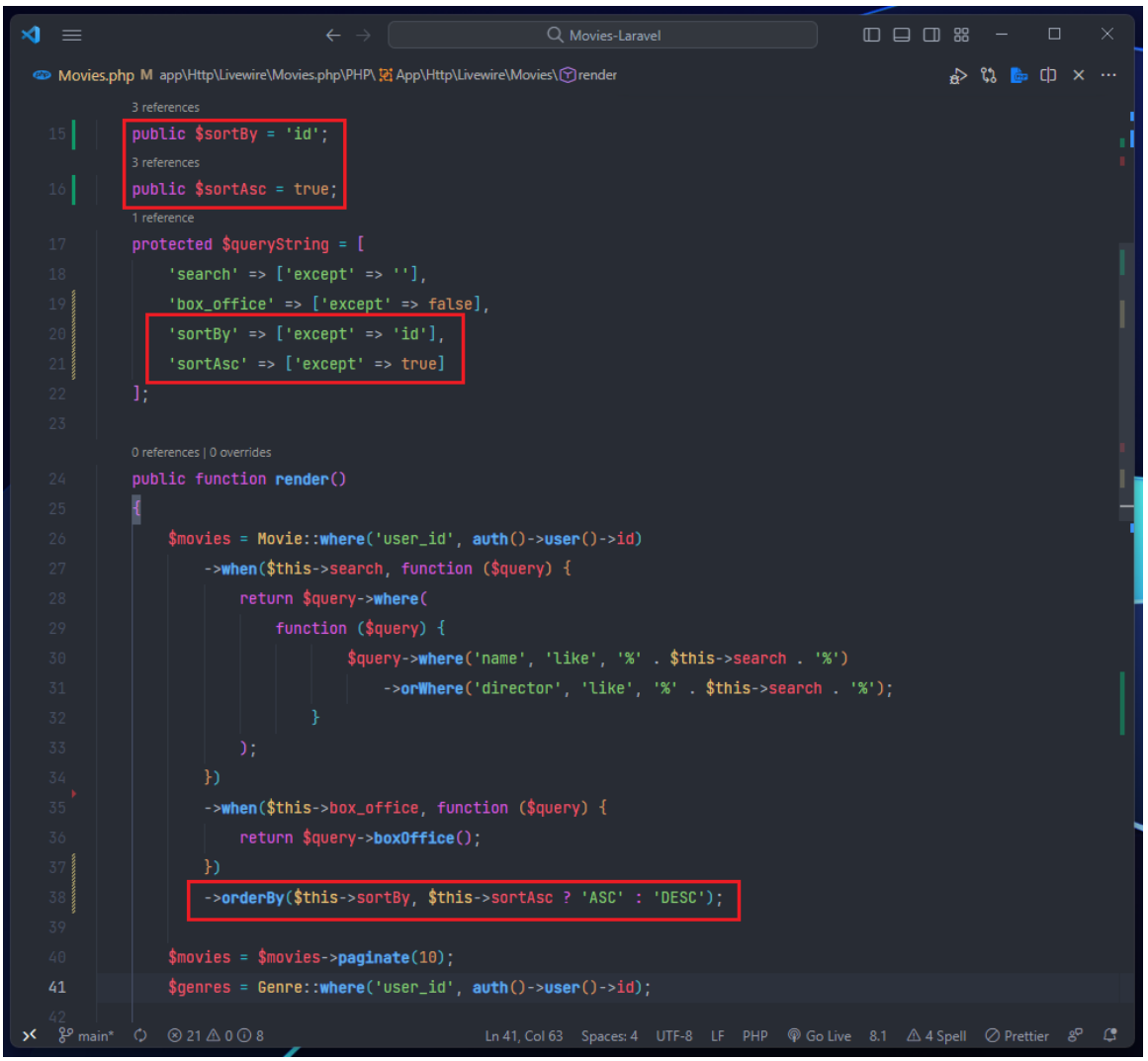
Vamos a implementar la funcionalidad de ordenar algunas de las columnas de la tabla de manera ascendente o descendente.

20.1. Comenzaremos modificando los componentes. En concreto con el de géneros (app\Http\Livewire\Genres.php), añadiendo lo siguiente.


Movies-Laravel
Genres.php M app\Http\Livewire\Genres.php\...
13 public $search;
3 references
14 public $sortBy = 'id';
3 references
15 public $sortAsc = true;
16
1 reference
17 protected $queryString = [
18 'search' => ['except' => ''],
19 'sortBy' => ['except' => 'id'],
20 'sortAsc' => ['except' => true]
21];
22
0 references | 0 overrides
23 public function render()
24 {
25 $genres = Genre::where('user_id', auth()->user()->id)
26 ->when($this->search, function ($query) {
27 return $query->where(
28 function ($query) {
29 $query->where('name', 'like', '%' . $this->search . '%');
30 }
31);
32 })
33 ->orderBy($this->sortBy, $this->sortAsc ? 'ASC' : 'DESC');
34
35 $genres = $genres->paginate(10);
36 return view('livewire.genres', [
37 'genres' => $genres,
38]);
39 }
main*
21 0 8
Ln 52, Col 1
Spaces: 4
UTF-8
LF
PHP
Go Live
8.1
4 Spell
Prettier

```
0 references | 0 overrides
public function updatingSearch()
{
    $this->resetPage();
}
0 references | 0 overrides
public function sortBy($field)
{
    if ($field == $this->sortBy) {
        $this->sortAsc = !$this->sortAsc;
    }
    $this->sortBy = $field;
}
}
```

20.2. A continuación hacemos lo mismo en el de películas (app\Http\Livewire\Movies.php) añadiendo lo siguiente


Movies-Laravel
Movies.php M app\Http\Livewire\Movies.php\PHP\ App\Http\Livewire\Movies\ render
3 references
public $sortBy = 'id';
3 references
public $sortAsc = true;
1 reference
protected $queryString = [
'search' => ['except' => ''],
'box_office' => ['except' => false],
'sortBy' => ['except' => 'id'],
'sortAsc' => ['except' => true]
];
0 references | 0 overrides
public function render()
{
$movies = Movie::where('user_id', auth()->user()->id)
->when($this->search, function ($query) {
return $query->where(
function ($query) {
$query->where('name', 'like', '%' . $this->search . '%')
->orWhere('director', 'like', '%' . $this->search . '%');
}
);
})
->when($this->box_office, function ($query) {
return $query->boxOffice();
})
->orderBy($this->sortBy, $this->sortAsc ? 'ASC' : 'DESC');
$movies = $movies->paginate(10);
$genres = Genre::where('user_id', auth()->user()->id);
main* 21 0 8 Ln 41, Col 63 Spaces: 4 UTF-8 LF PHP Go Live 8.1 4 Spell Prettier

```php
0 references | 0 overrides
public function updatingBoxOffice()
{
    $this->resetPage();
}
0 references | 0 overrides
public function updatingSearch()
{
    $this->resetPage();
}
0 references | 0 overrides
public function sortBy($field)
{
    if ($field == $this->sortBy) {
        $this->sortAsc = !$this->sortAsc;
    }
    $this->sortBy = $field;
}
```

20.3. Ahora vamos añadir los heroicons de tailwind. Para eso creamos un nuevo fichero en la ruta **resources\views\components** con el nombre **sort-icon.blade.php** y dentro escribimos lo siguiente.

Movies-Laravel

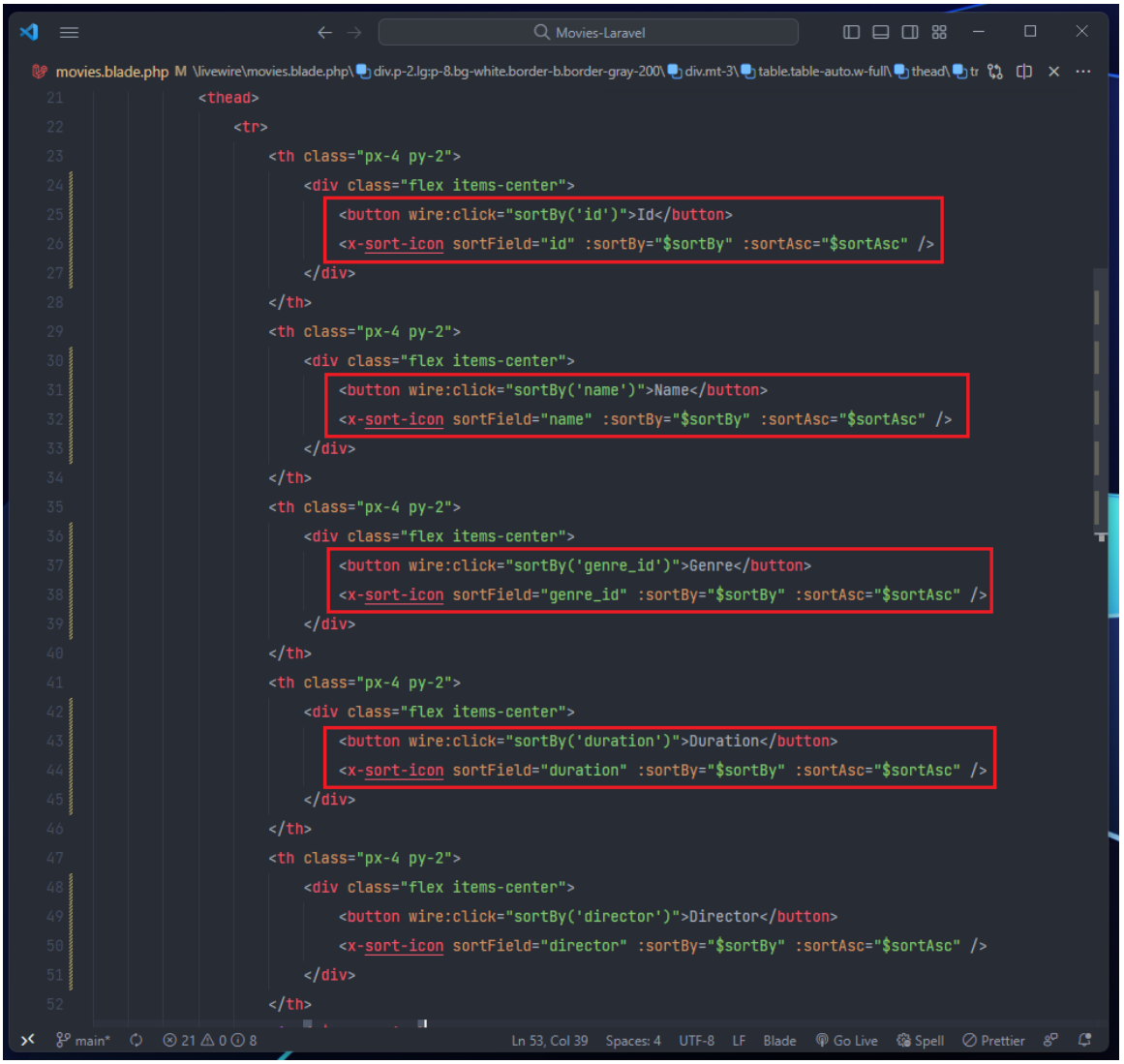
sort-icon.blade.php U resources\views\components\sort-icon.blade.php\...

```blade
@props(['sortBy', 'sortAsc', 'sortField'])
@if ($sortBy == "$sortField")
@if (!$sortAsc)
    <span class="w-4 h-4 ml-2">
        <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
            stroke-width="1.5" stroke="currentColor" class="w-6 h-6">
            <path stroke-linecap="round" stroke-linejoin="round"
                d="M19.5 8.25l-7.5 7.5-7.5-7.5" />
        </svg>
    </span>
@else
    <span class="w-4 h-4 ml-2">
        <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
            stroke-width="1.5" stroke="currentColor" class="w-6 h-6">
            <path stroke-linecap="round" stroke-linejoin="round"
                d="M4.5 15.75l7.5-7.5 7.5 7.5" />
        </svg>
    </span>
@endif
@endif
```

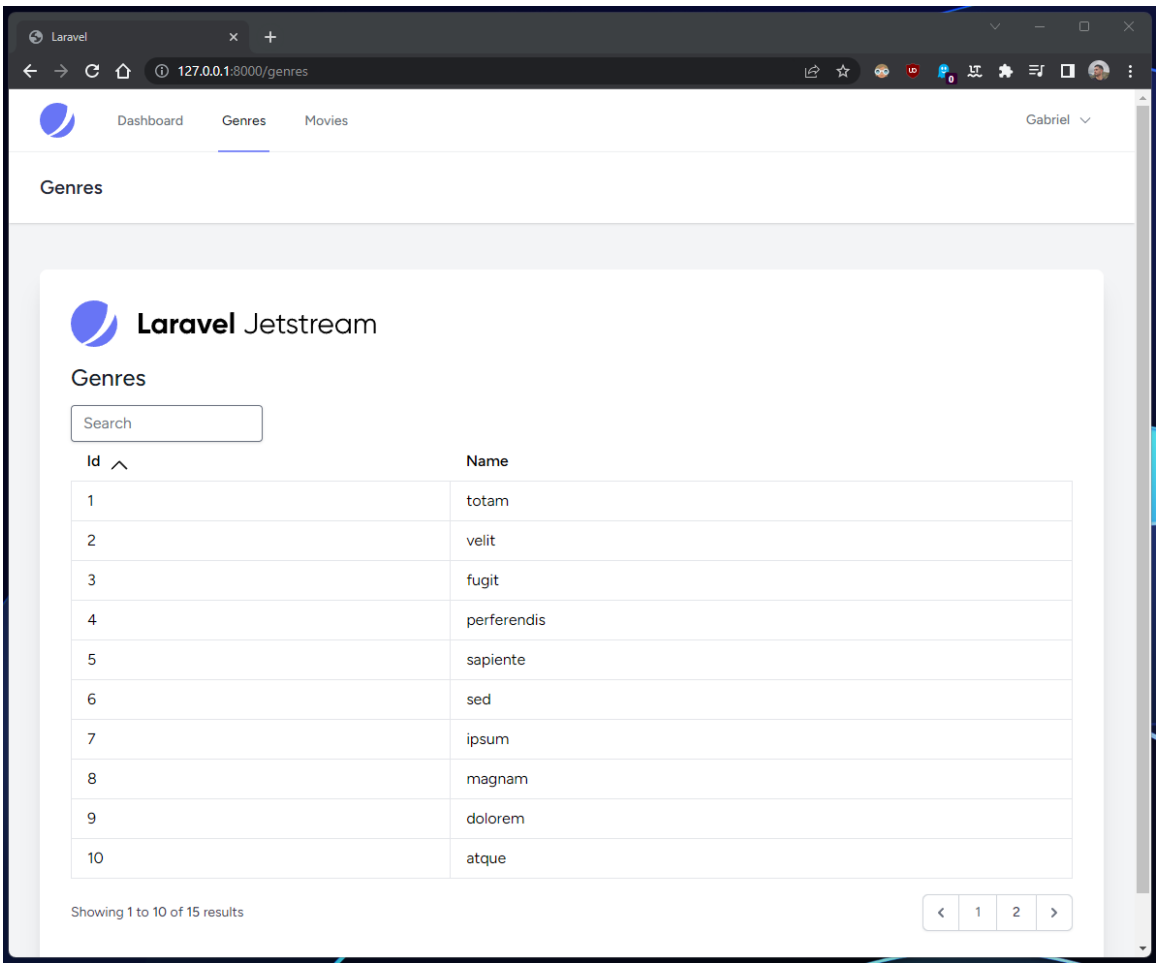
20.4. Por último nos queda modificar las vistas así que comenzaremos modificando la vista de género (resources\views\livewire\genres.blade.php) reemplazando las anteriores líneas de código por las nuevas marcadas.

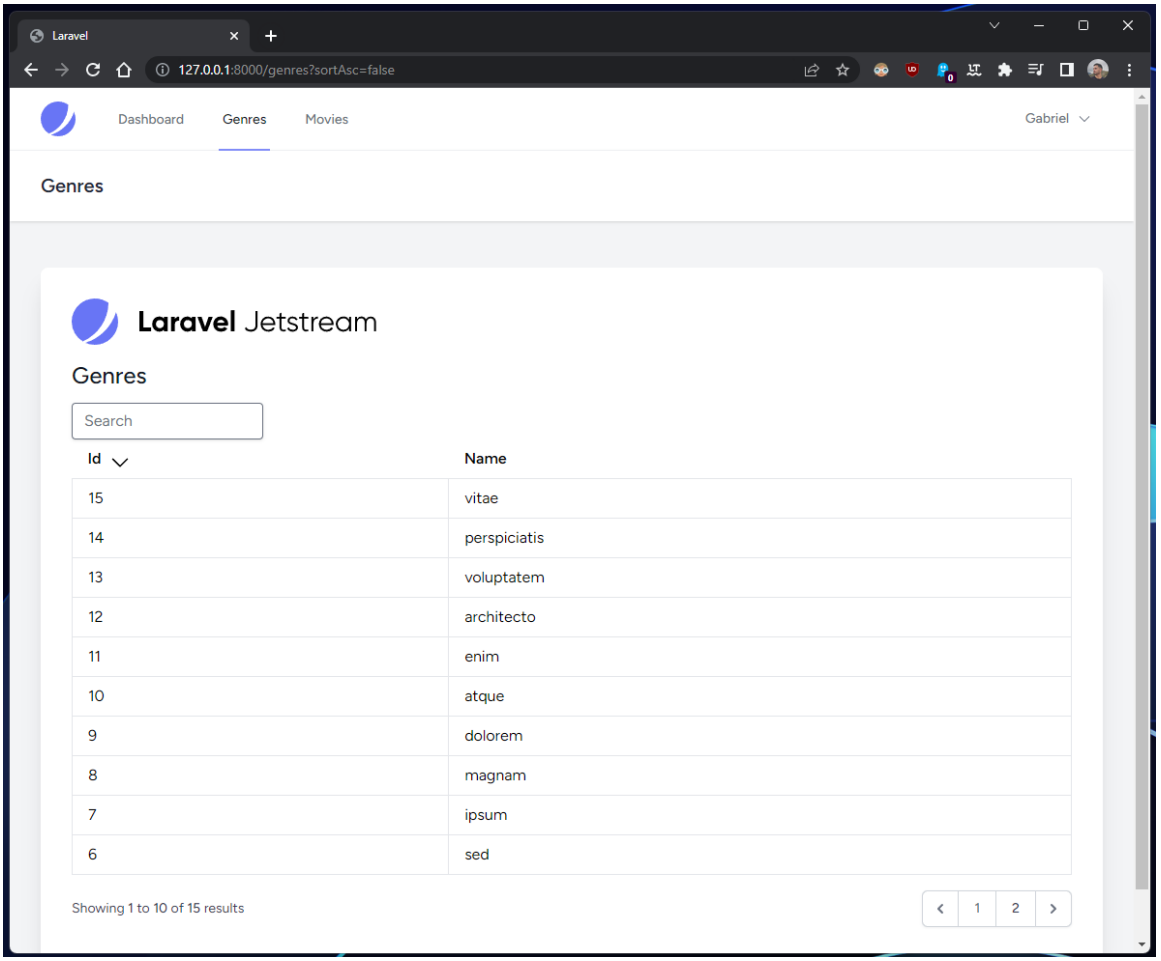
```
<div class="mt-3">
    <div class="flex justify-between">
        <div>
            <input wire:model.debounce.300ms="search" type="search" placeholder="Search"
                class="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight
                focus:outline-none focus:shadow-outline"
                name="">
        </div>
    </div>
    <table class="table-auto w-full">
        <thead>
            <tr>
                <th class="px-4 py-2">
                    <div class="flex items-center">
                        <button wire:click="sortBy('id')">Id</button>
                        <x-sort-icon sortField="id" :sortBy="$sortBy" :sortAsc="$sortAsc" />
                    </div>
                </th>
                <th class="px-4 py-2">
                    <div class="flex items-center">
                        <button wire:click="sortBy('name')">Name</button>
                        <x-sort-icon sortField="name" :sortBy="$sortBy" :sortAsc="$sortAsc" />
                    </div>
                </th>
            </tr>
        </thead>
        <tbody>
```

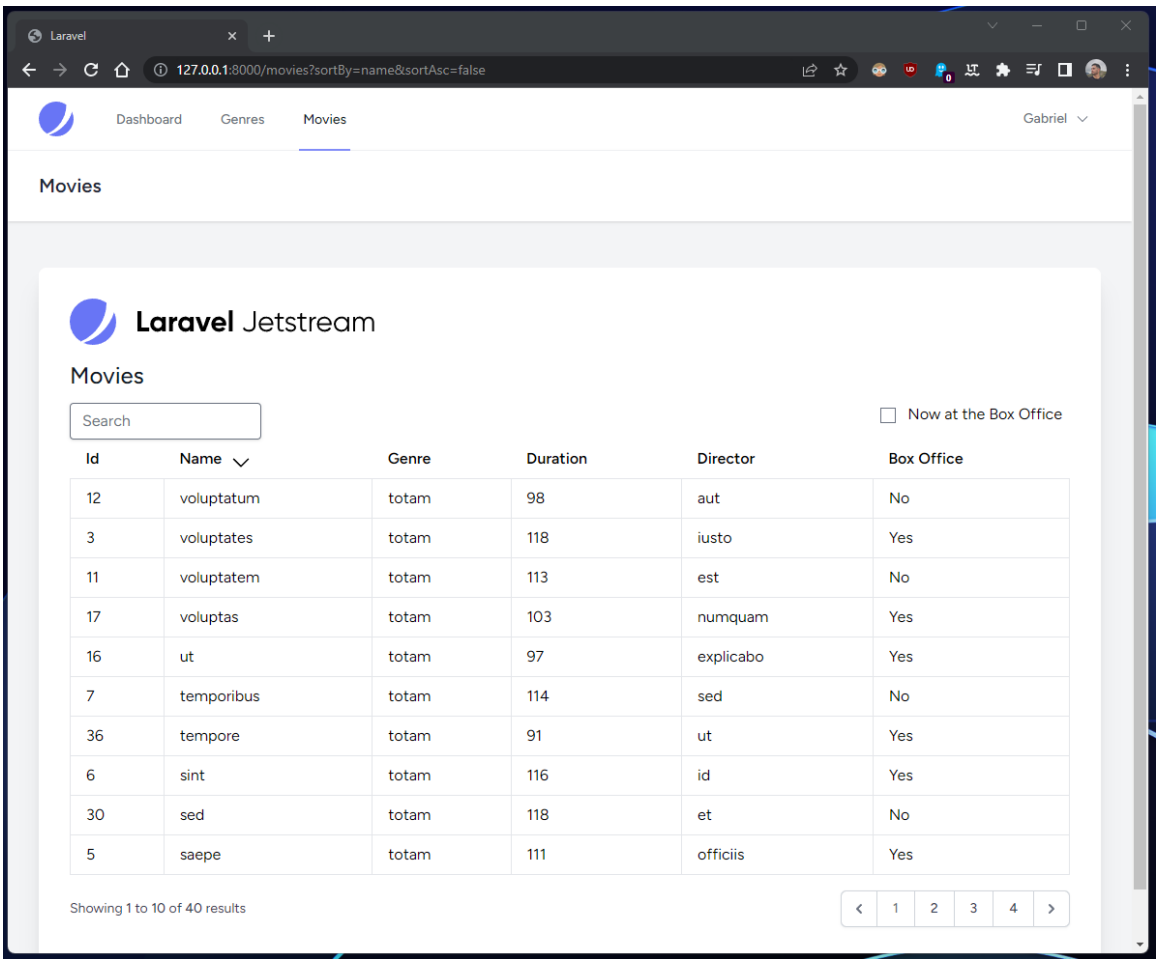
20.5. Repetimos el paso anterior pero ahora en la vista de películas (resources\views\livewire\movies.blade.php)



20.6. Ya sólo nos queda probarlo. Como se ve ahora pinchando en el nombre de las columnas en las que hemos implementado la ordenación nos permite ordenar los datos de la tabla de manera ascendente y descendente.


Laravel
127.0.0.1:8000/genres
Dashboard
Genres
Movies
Gabriel
Genres
Laravel Jetstream
Genres
Search
Id
Name
1 totam
2 velit
3 fugit
4 perferendis
5 sapiente
6 sed
7 ipsum
8 magnam
9 dolorem
10 atque
Showing 1 to 10 of 15 results
1
2

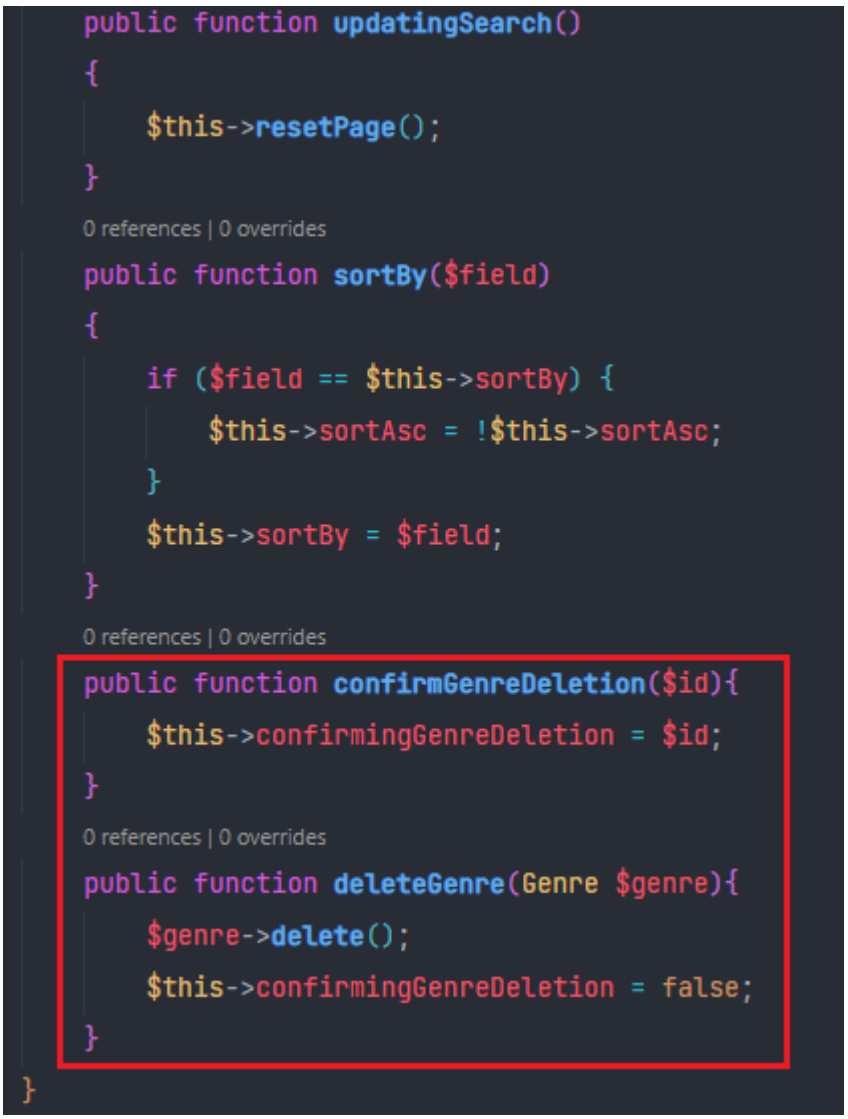

Laravel
127.0.0.1:8000/genres?sortAsc=false
Dashboard
Genres
Movies
Gabriel
Genres
Laravel Jetstream
Genres
Search
Id
Name
15 vitae
14 perspiciatis
13 voluptatem
12 architecto
11 enim
10 atque
9 dolorem
8 magnam
7 ipsum
6 sed
Showing 1 to 10 of 15 results
1
2


Laravel
127.0.0.1:8000/movies?sortBy=name&sortAsc=false
Dashboard
Genres
Movies
Gabriel
Movies
Laravel Jetstream
Movies
Search
Now at the Box Office
Id
Name
Genre
Duration
Director
Box Office
12 voluptatum totam 98 aut No
3 voluptates totam 118 iusto Yes
11 voluptatem totam 113 est No
17 voluptas totam 103 numquam Yes
16 ut totam 97 explicabo Yes
7 temporibus totam 114 sed No
36 tempore totam 91 ut Yes
6 sint totam 116 id Yes
30 sed totam 118 et No
5 saepe totam 111 officiis Yes
Showing 1 to 10 of 40 results
1
2
3
4

21. Eliminación de datos

21.1. Comenzaremos modificando el componente de Géneros (app\Http\Livewire\Genres.php). Añadiendo lo siguiente.

```php
namespace App\Http\Livewire;

use Livewire\Component;
use Livewire\WithPagination;
use App\Models\Genre;

0 references | 0 implementations
class Genres extends Component
{
    use WithPagination;

    2 references
    public $search;
    3 references
    public $sortBy = 'id';
    3 references
    public $sortAsc = true;
    2 references
    public $confirmingGenreDeletion = false;


    1 reference
    protected $queryString = [
        'search' => ['except' => ''],
        'sortBy' => ['except' => 'id'],
        'sortAsc' => ['except' => true]
    ];
```



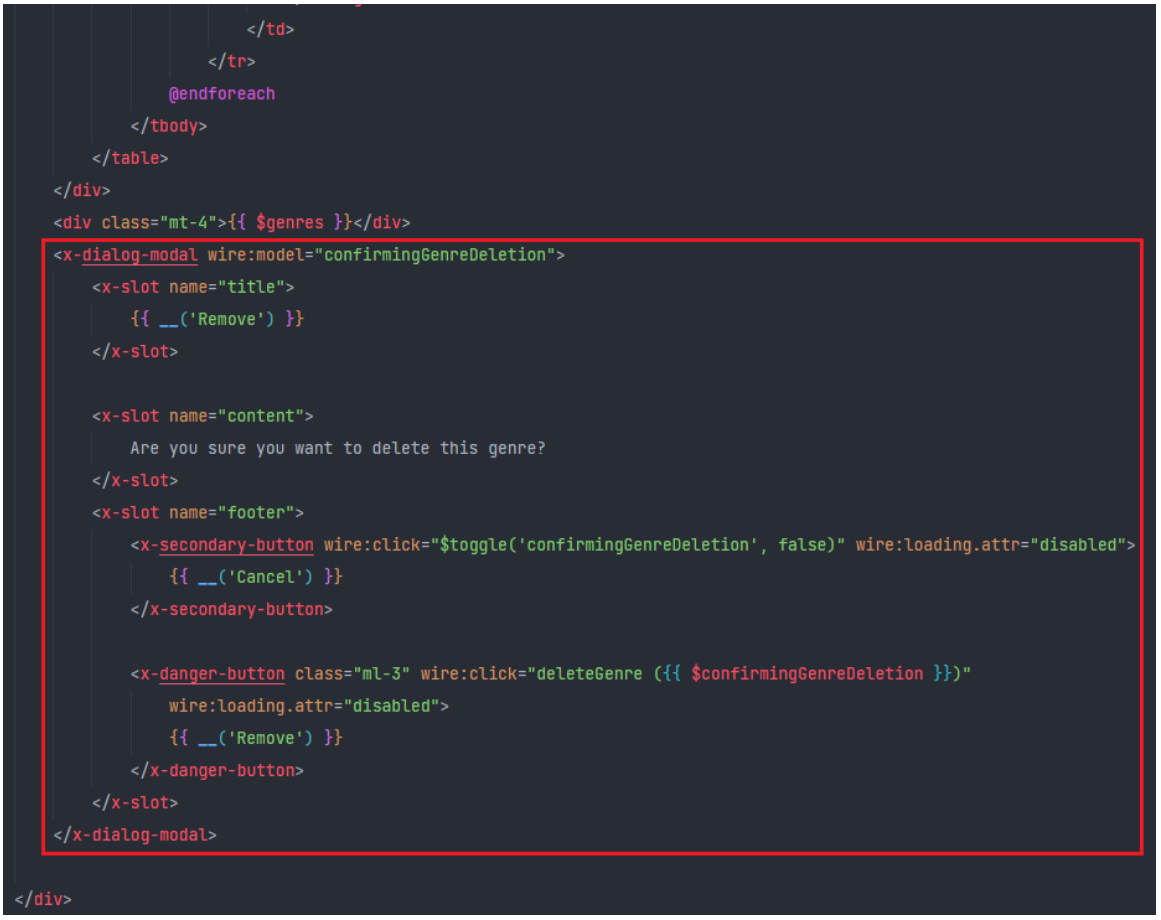
```php
    public function updatingSearch()
    {
        $this->resetPage();
    }

    0 references | 0 overrides
    public function sortBy($field)
    {
        if ($field == $this->sortBy) {
            $this->sortAsc = !$this->sortAsc;
        }
        $this->sortBy = $field;
    }

    0 references | 0 overrides
    public function confirmGenreDeletion($id){
        $this->confirmingGenreDeletion = $id;
    }

    0 references | 0 overrides
    public function deleteGenre(Genre $genre){
        $genre->delete();
        $this->confirmingGenreDeletion = false;
    }
}
```

21.2. A continuación hacemos lo mismo con el componente de películas (app\Http\Livewire\Movies.php).

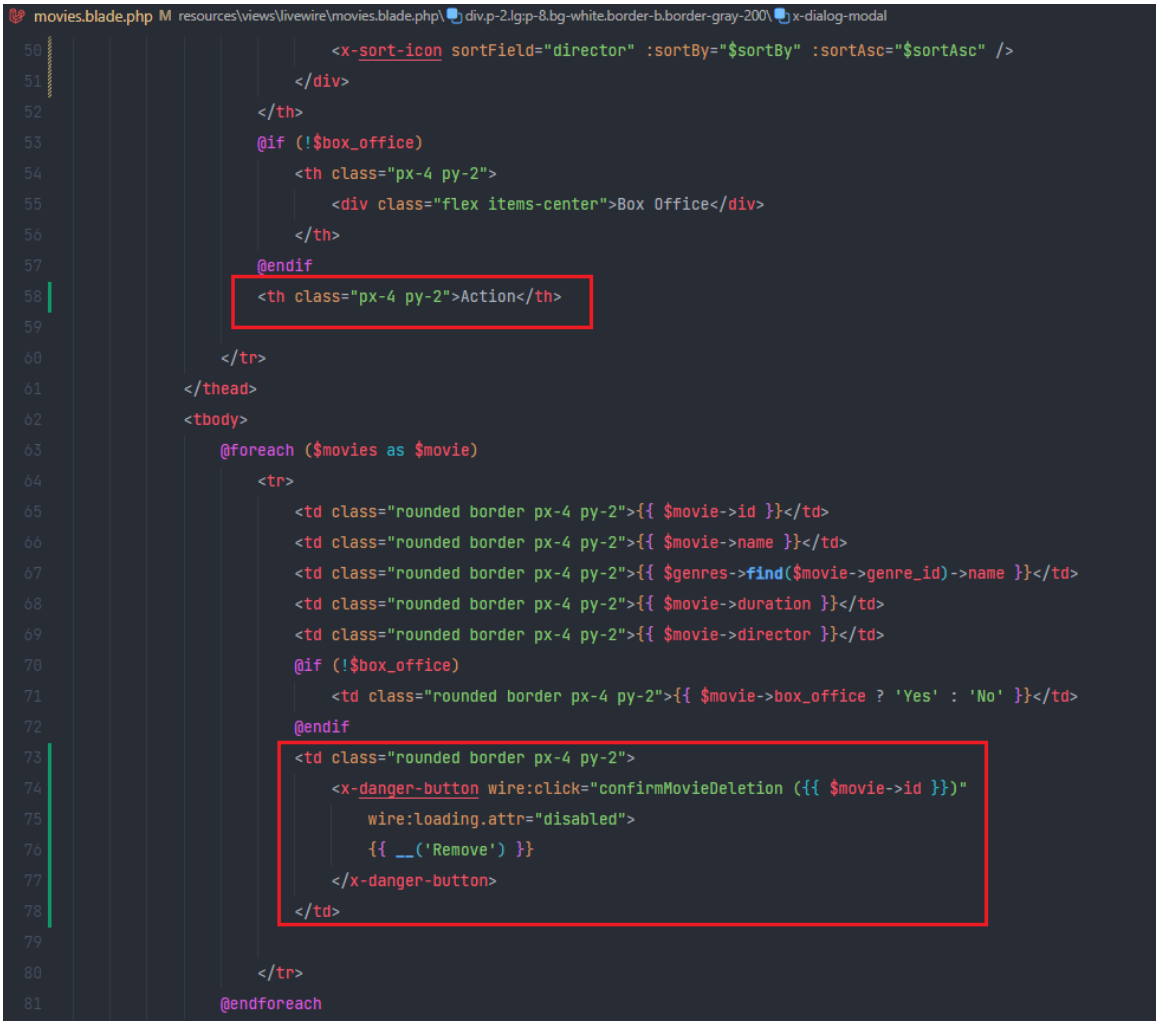
```php
class Movies extends Component
{
    use WithPagination;

    1 reference
    public $box_office;

    3 references
    public $search;

    3 references
    public $sortBy = 'id';

    3 references
    public $sortAsc = true;

    2 references
    public $confirmingMovieDeletion = false;

    public function updatingSearch()
    {
        $this->resetPage();
    }

    0 references | 0 overrides
    public function sortBy($field)
    {
        if ($field == $this->sortBy) {
            $this->sortAsc = !$this->sortAsc;
        }
        $this->sortBy = $field;
    }

    0 references | 0 overrides
    public function confirmMovieDeletion($id){
        $this->confirmingMovieDeletion = $id;
    }

    0 references | 0 overrides
    public function deleteMovie(Movie $movie){
        $movie->delete();
        $this->confirmingMovieDeletion = false;
    }
}
```

21.3. Seguimos añadiendo una nueva vista modal y modificando la vista actual. Para ello nos vamos a la vista de géneros (resources\views\livewire\genres.blade.php) y añadimos

```
                        <button wire:click="sortBy('id')">Id</button>
                        <x-sort-icon sortField="id" :sortBy="$sortBy" :sortAsc="$sortAsc" />
                    </div>
                </th>
                <th class="px-4 py-2">
                    <div class="flex items-center">
                        <button wire:click="sortBy('name')">Name</button>
                        <x-sort-icon sortField="name" :sortBy="$sortBy" :sortAsc="$sortAsc" />
                    </div>
                </th>
                <th class="px-4 py-2">Action</th>
            </tr>
        </thead>
        <tbody>
            @foreach ($genres as $genre)
                <tr>
                    <td class="rounded border px-4 py-2">{{ $genre->id }}</td>
                    <td class="rounded border px-4 py-2">{{ $genre->name }}</td>
                    <td class="rounded border px-4 py-2">
                        <x-danger-button wire:click="confirmGenreDeletion ({{ $genre->id }})"
                            wire:loading.attr="disabled">
                            {{ __('Remove') }}
                        </x-danger-button>
                    </td>
                </tr>
            @endforeach
        </tbody>
    </table>
```



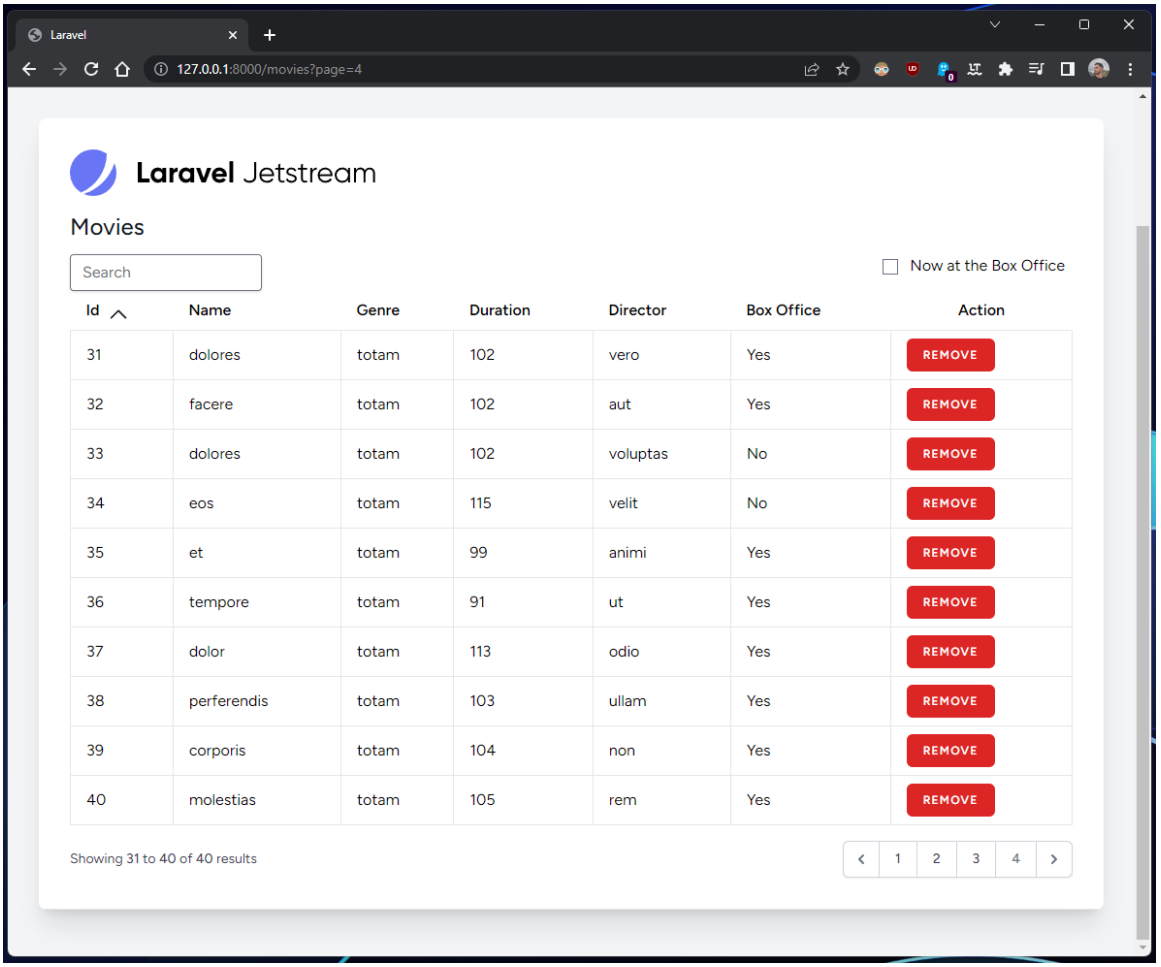
21.4. Hacemos lo mismo en la vista de películas (resources\views\livewire\movies.blade.php)

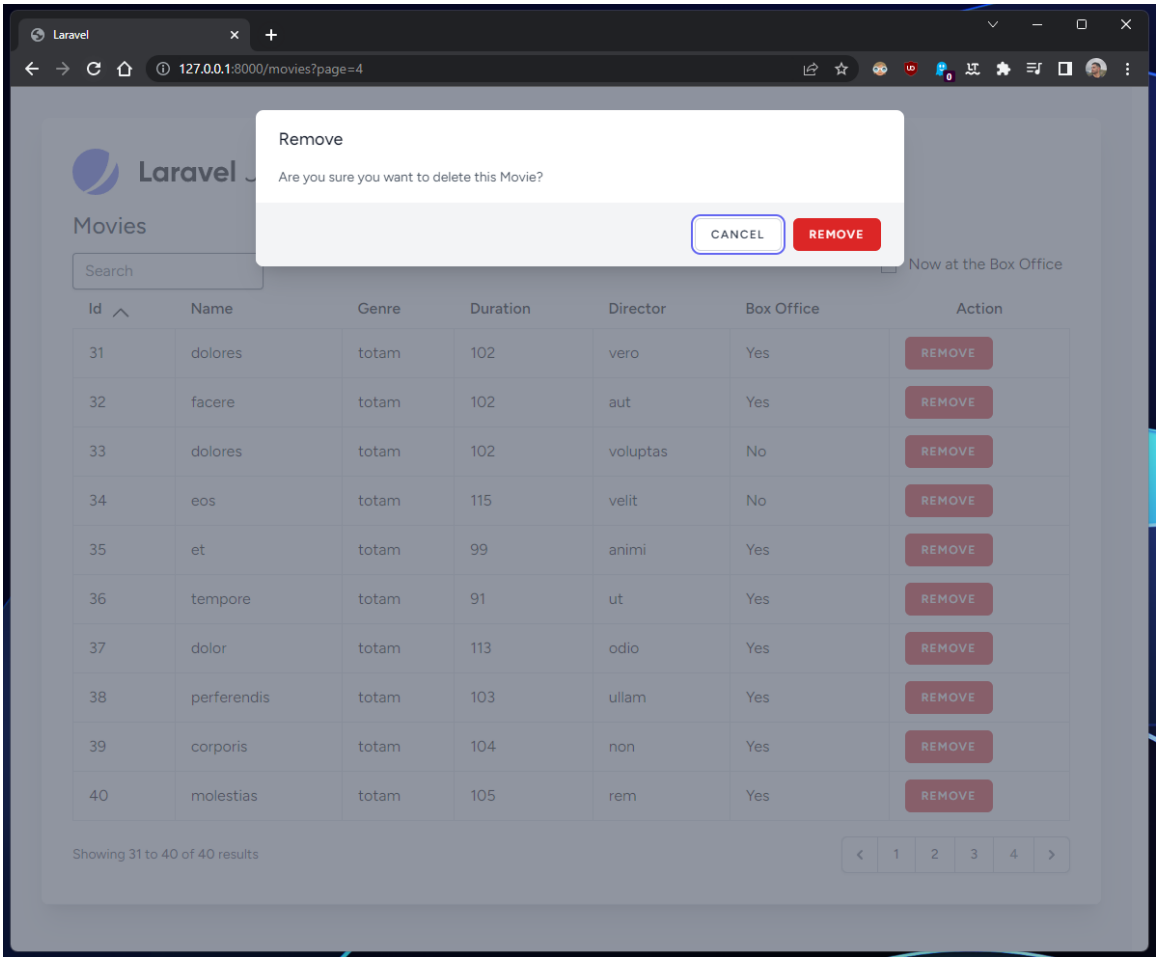

movies.blade.php M resources\views\livewire\movies.blade.php\ div.p-2.lg:p-8.bg-white.border-b.border-gray-200\ x-dialog-modal
<x-sort-icon sortField="director" :sortBy="$sortBy" :sortAsc="$sortAsc" />

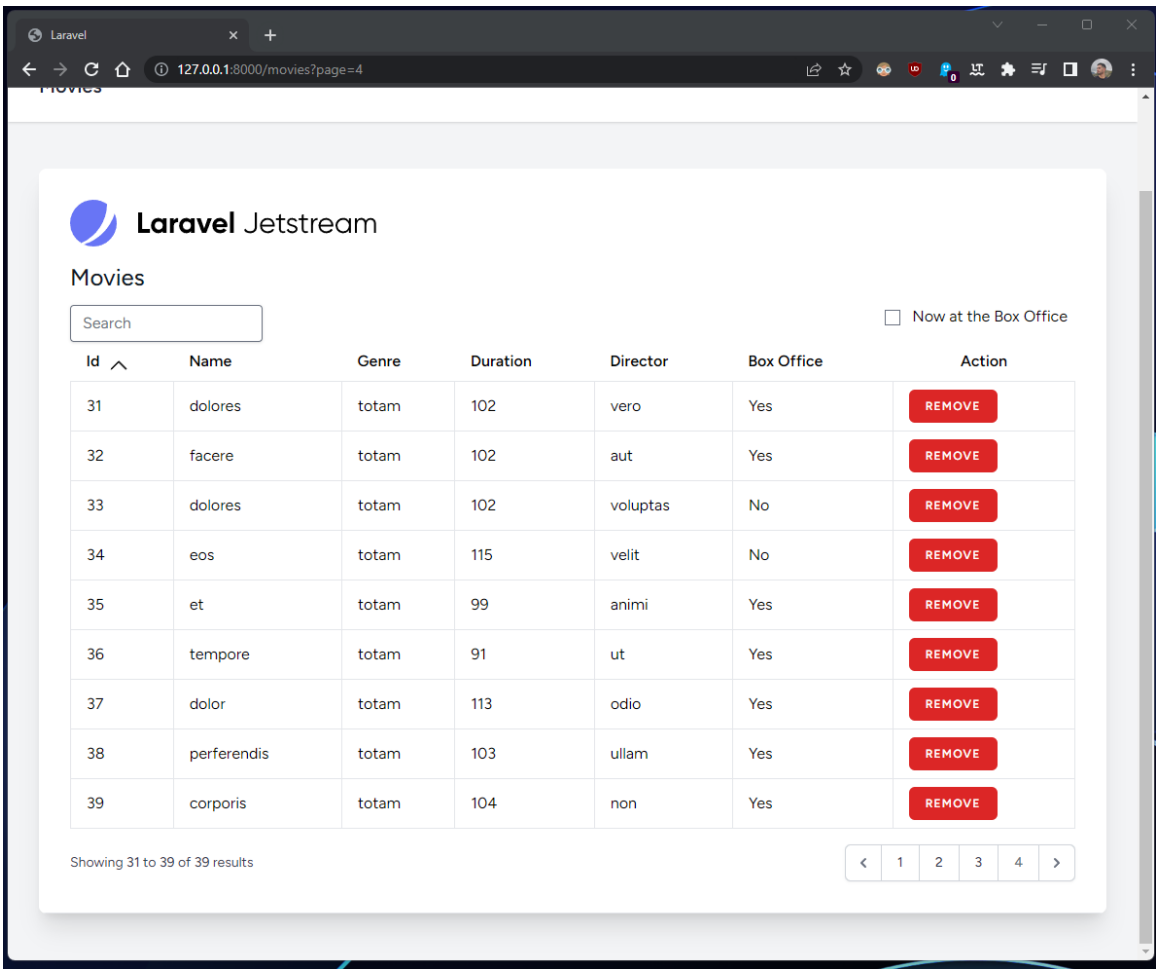
</th>
@if (!$box_office)
<th class="px-4 py-2">
Box Office
</th>
@endif
<th class="px-4 py-2">Action</th>
</tr>
</thead>
<tbody>
@foreach ($movies as $movie)
<tr>
<td class="rounded border px-4 py-2">{{ $movie->id }}</td>
<td class="rounded border px-4 py-2">{{ $movie->name }}</td>
<td class="rounded border px-4 py-2">{{ $genres->find($movie->genre_id)->name }}</td>
<td class="rounded border px-4 py-2">{{ $movie->duration }}</td>
<td class="rounded border px-4 py-2">{{ $movie->director }}</td>
@if (!$box_office)
<td class="rounded border px-4 py-2">{{ $movie->box_office ? 'Yes' : 'No' }}</td>
@endif
<td class="rounded border px-4 py-2">
<x-danger-button wire:click="confirmMovieDeletion ({{ $movie->id }})"
wire:loading.attr="disabled">
{{ __('Remove') }}
</x-danger-button>
</td>
</tr>
@endforeach

```
                    </tr>
                @endforeach
            </tbody>
        </table>
    </div>
    <div class="mt-4">{{ $movies }}</div>
    <x-dialog-modal wire:model="confirmingMovieDeletion">
        <x-slot name="title">
            {{ __('Remove') }}
        </x-slot>

        <x-slot name="content">
            Are you sure you want to delete this Movie?
        </x-slot>
        <x-slot name="footer">
            <x-secondary-button wire:click="$toggle('confirmingMovieDeletion', false)" wire:loading.attr="disabled">
                {{ __('Cancel') }}
            </x-secondary-button>

            <x-danger-button class="ml-3" wire:click="deleteMovie ({{ $confirmingMovieDeletion }})"
                wire:loading.attr="disabled">
                {{ __('Remove') }}
            </x-danger-button>
        </x-slot>
    </x-dialog-modal>
</div>
```

21.5. A continuación vamos a nuestra web y probamos si funciona. Como vemos la web se visualiza correctamente y podemos eliminar tanto películas como géneros.

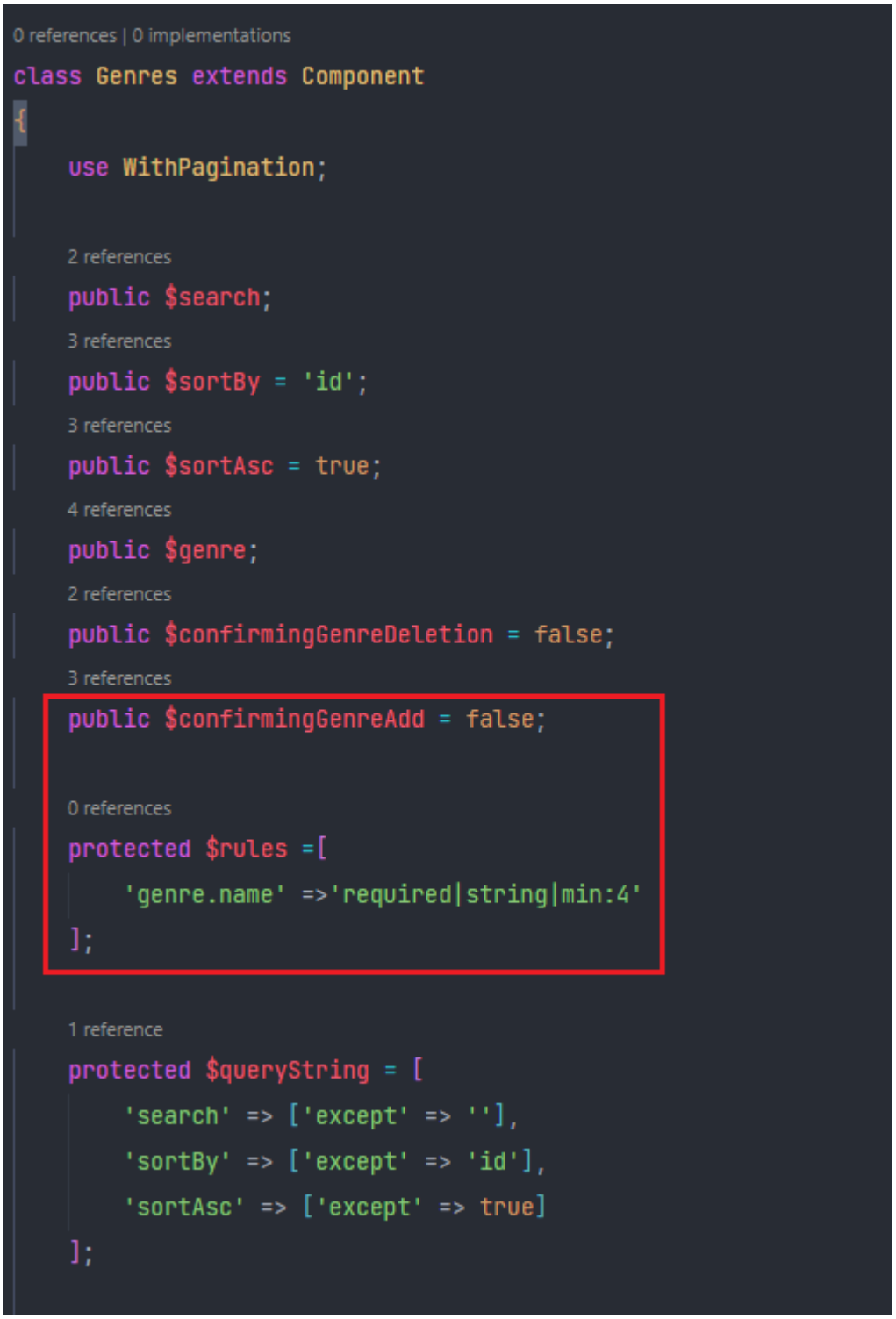

Laravel
127.0.0.1:8000/movies?page=4
Laravel Jetstream
Movies
Search
Now at the Box Office
Id
Name
Genre
Duration
Director
Box Office
Action
31 dolores totam 102 vero Yes REMOVE
32 facere totam 102 aut Yes REMOVE
33 dolores totam 102 voluptas No REMOVE
34 eos totam 115 velit No REMOVE
35 et totam 99 animi Yes REMOVE
36 tempore totam 91 ut Yes REMOVE
37 dolor totam 113 odio Yes REMOVE
38 perferendis totam 103 ullam Yes REMOVE
39 corporis totam 104 non Yes REMOVE
40 molestias totam 105 rem Yes REMOVE
Showing 31 to 40 of 40 results
1 2 3 4


Laravel
127.0.0.1:8000/movies?page=4
Remove
Are you sure you want to delete this Movie?
CANCEL
REMOVE
Laravel
Movies
Search
Now at the Box Office
Id
Name
Genre
Duration
Director
Box Office
Action
31 dolores totam 102 vero Yes REMOVE
32 facere totam 102 aut Yes REMOVE
33 dolores totam 102 voluptas No REMOVE
34 eos totam 115 velit No REMOVE
35 et totam 99 animi Yes REMOVE
36 tempore totam 91 ut Yes REMOVE
37 dolor totam 113 odio Yes REMOVE
38 perferendis totam 103 ullam Yes REMOVE
39 corporis totam 104 non Yes REMOVE
40 molestias totam 105 rem Yes REMOVE
Showing 31 to 40 of 40 results
1
2
3
4



22. Añadir y editar datos

22.1. Comenzaremos modificando los componentes. En concreto empezaremos con el de géneros(app\Http\Livewire\Genres.php) .



```
0 references | 0 implementations
class Genres extends Component
{
    use WithPagination;

    2 references
    public $search;
    3 references
    public $sortBy = 'id';
    3 references
    public $sortAsc = true;
    4 references
    public $genre;
    2 references
    public $confirmingGenreDeletion = false;
    3 references
    public $confirmingGenreAdd = false;

    0 references
    protected $rules =[
        'genre.name' =>'required|string|min:4'
    ];

    1 reference
    protected $queryString = [
        'search' => ['except' => ''],
        'sortBy' => ['except' => 'id'],
        'sortAsc' => ['except' => true]
    ];
```

```php
0 references | 0 overrides
public function deleteGenre(Genre $genre)
{
    $genre->delete();
    $this->confirmingGenreDeletion = false;
}
0 references | 0 overrides
public function confirmGenreAdd()
{
    $this->reset(['genre']);
    $this->confirmingGenreAdd = true;
}
0 references | 0 overrides
public function addGenre()
{
    $this->validate();
    if (isset($this->genre->id)) {
        $this->genre->save();
    } else {
        auth()->user()->genres()->create([
            'name' => $this->genre['name']
        ]);
    }
    $this->confirmingGenreAdd = false;
}
0 references | 0 overrides
public function confirmGenreEdit(Genre $genre)
{
    $this->genre = $genre;
    $this->confirmingGenreAdd = true;
}
```

22.2. Haremos lo mismo pero para el componente de películas

```php
class Movies extends Component
{
    use WithPagination;

    1 reference
    public $box_office;

    3 references
    public $search;

    3 references
    public $sortBy = 'id';

    3 references
    public $sortAsc = true;

    8 references
    public $movie;

    2 references
    public $confirmingMovieDeletion = false;

    3 references
    public $confirmingMovieAdd = false;


    0 references
    protected $rules = [
        'movie.genre_id' => 'required|int|min:1',
        'movie.name' => 'required|string|min:4',
        'movie.duration' => 'required|int|between:1,1000',
        'movie.director' => 'required|string|min:4',
        'movie.box_office' => 'boolean',
    ];
```

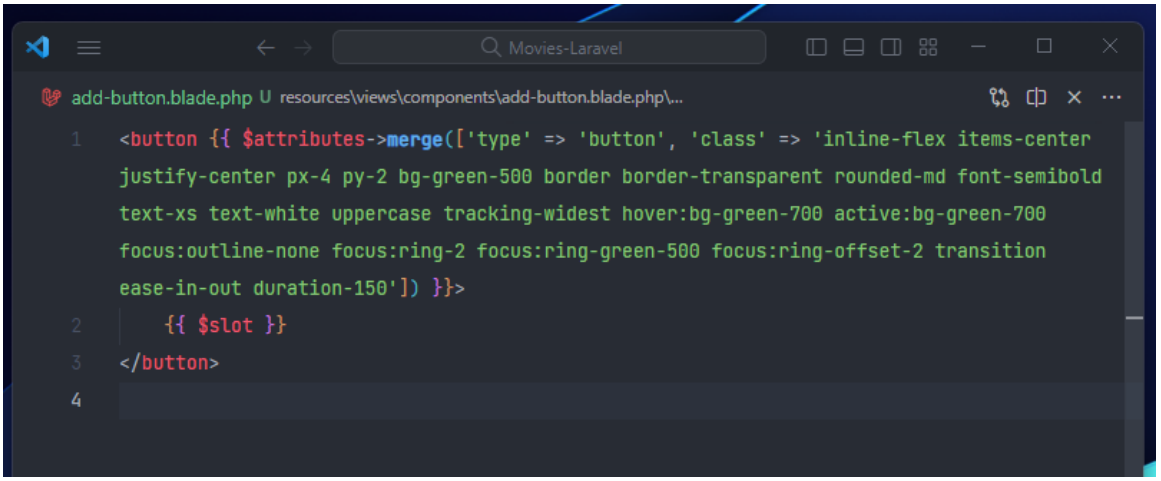
```php
public function render()
{
    $movies = Movie::where('user_id', auth()->user()->id)
        ->when($this->search, function ($query) {
            return $query->where(
                function ($query) {
                        $query->where('name', 'like', '%' . $this->search . '%')
                            ->orWhere('director', 'like', '%' . $this->search . '%');
                    }
            );
        })
        ->when($this->box_office, function ($query) {
            return $query->boxOffice();
        })
        ->orderBy($this->sortBy, $this->sortAsc ? 'ASC' : 'DESC');

    $movies = $movies->paginate(10);
    $genres = Genre::where('user_id', auth()->user()->id)->get();

    return view('livewire.movies', [
        'movies' => $movies,
        'genres' => $genres,
    ]);
}
```

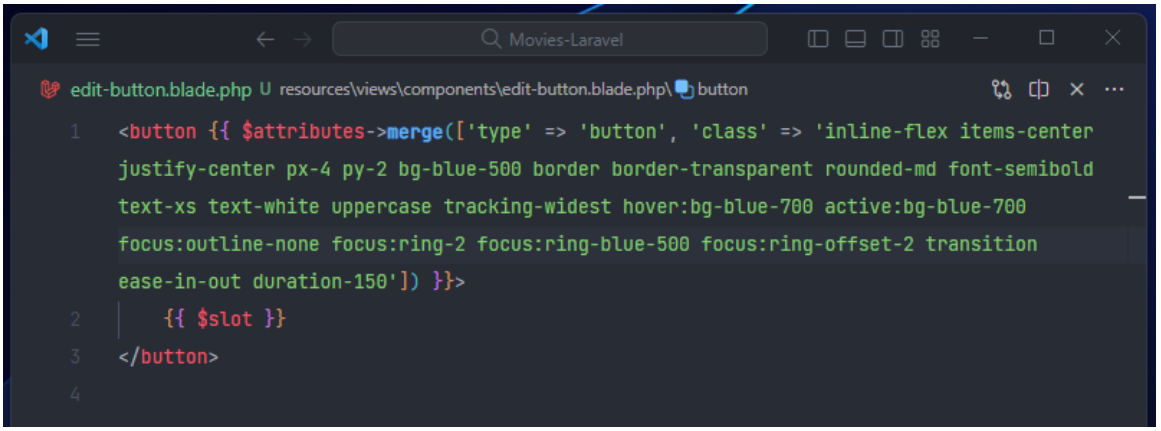
```php
public function deleteMovie(Movie $movie)
{
    $movie->delete();
    $this->confirmingMovieDeletion = false;
}

0 references | 0 overrides
public function confirmMovieAdd()
{
    $this->reset(['movie']);
    $this->confirmingMovieAdd = true;
}

0 references | 0 overrides
public function addMovie()
{
    $this->validate();
    if (isset($this->movie->id)) {
        $this->movie->save();
    } else {
        auth()->user()->movies()->create([
            'genre_id' => $this->movie['genre_id'],
            'name' => $this->movie['name'],
            'duration' => $this->movie['duration'],
            'director' => $this->movie['director'],
            'box_office' => $this->movie['box_office'] ?? 0
        ]);
    }
    $this->confirmingMovieAdd = false;
}


0 references | 0 overrides
public function confirmMovieEdit(Movie $movie)
{
    $this->movie = $movie;
    $this->confirmingMovieAdd = true;
}
```

22.3. Además en el caso de películas deberemos ir al modelo(app\Models\Movie.php) y modificarlo ligeramente.

```php
class Movie extends Model
{
    use HasFactory;
    0 references
    protected $fillable = ['genre_id','name','duration','director','box_office'];

    0 references | 0 overrides
    public function user() {
        return $this->belongsTo(User::class);
    }
    0 references | 0 overrides
    public function scopeBoxOffice($query)
    {
        return $query->where('box_office', 1);
    }
}
```

22.4. A continuación irémos a la carpeta resources\views\components y crearemos dos componentes uno será el botón de añadir(add-button.blade.php) y otro el de editar(edit-button.blade.php). El código quedará de la siguiente manera.





22.5. A continuación es momento de modificar las vistas. Empezaremos con la de géneros (resources\views\livewire\genres.blade.php).

```
<div class="p-2 lg:p-8 bg-white border-b border-gray-200">
    <x-application-logo class="block h-12 w-auto" />

    <div class="mt-4 text-2xl font-medium text-gray-900 flex justify-between">
        <div>{{ __('Genres') }}</div>
        <div class="mr-2">
            <x-add-button wire:click="confirmGenreAdd">
                {{ __('Add') }}
            </x-add-button>
        </div>
    </div>
    <div class="mt-3">
        <div class="flex justify-between">
```

```
@foreach ($genres as $genre)
    <tr>
        <td class="rounded border px-4 py-2">{{ $genre->id }}</td>
        <td class="rounded border px-4 py-2">{{ $genre->name }}</td>
        <td class="rounded border px-4 py-2">
            <x-edit-button wire:click="confirmGenreEdit ({{ $genre->id }})">{{ __('Edit') }}
            </x-edit-button>
            <x-danger-button wire:click="confirmGenreDeletion ({{ $genre->id }})"
                wire:loading.attr="disabled">{{ __('Remove') }}</x-danger-button>
        </td>
    </tr>
@endforeach
```

```blade
<x-dialog-modal wire:model="confirmingGenreAdd">
    <x-slot name="title">
        {{ isset($this->genre->id) ? __('Edit genre') : __('Add genre') }}
    </x-slot>

    <x-slot name="content">
        <div class="col-span-6 sm:col-span-4 mt-4">
            <x-label for="name" value="{{ __('Name') }}" />
            <x-input id="genre.name" type="text" class="mt-1 block w-full" wire:model.defer="genre.name" />
            <x-input-error for="genre.name" class="mt-2" />
        </div>
    </x-slot>
    <x-slot name="footer">
        <x-secondary-button wire:click="$toggle('confirmingGenreAdd', false)" wire:loading.attr="disabled">
            {{ __('Cancel') }}
        </x-secondary-button>

        @if (isset($this->genre->id))
            <x-edit-button class="ml-3" wire:click="addGenre" wire:loading.attr="disabled">
                {{ __('Save changes') }}
            </x-edit-button>
        @else
            <x-add-button class="ml-3" wire:click="addGenre" wire:loading.attr="disabled">
                {{ __('Add') }}
            </x-add-button>
        @endif
    </x-slot>
</x-dialog-modal>
```

22.6. Haremos lo mismo con nuestra vista de peliculas(resources\views\livewire\movies.blade.php)

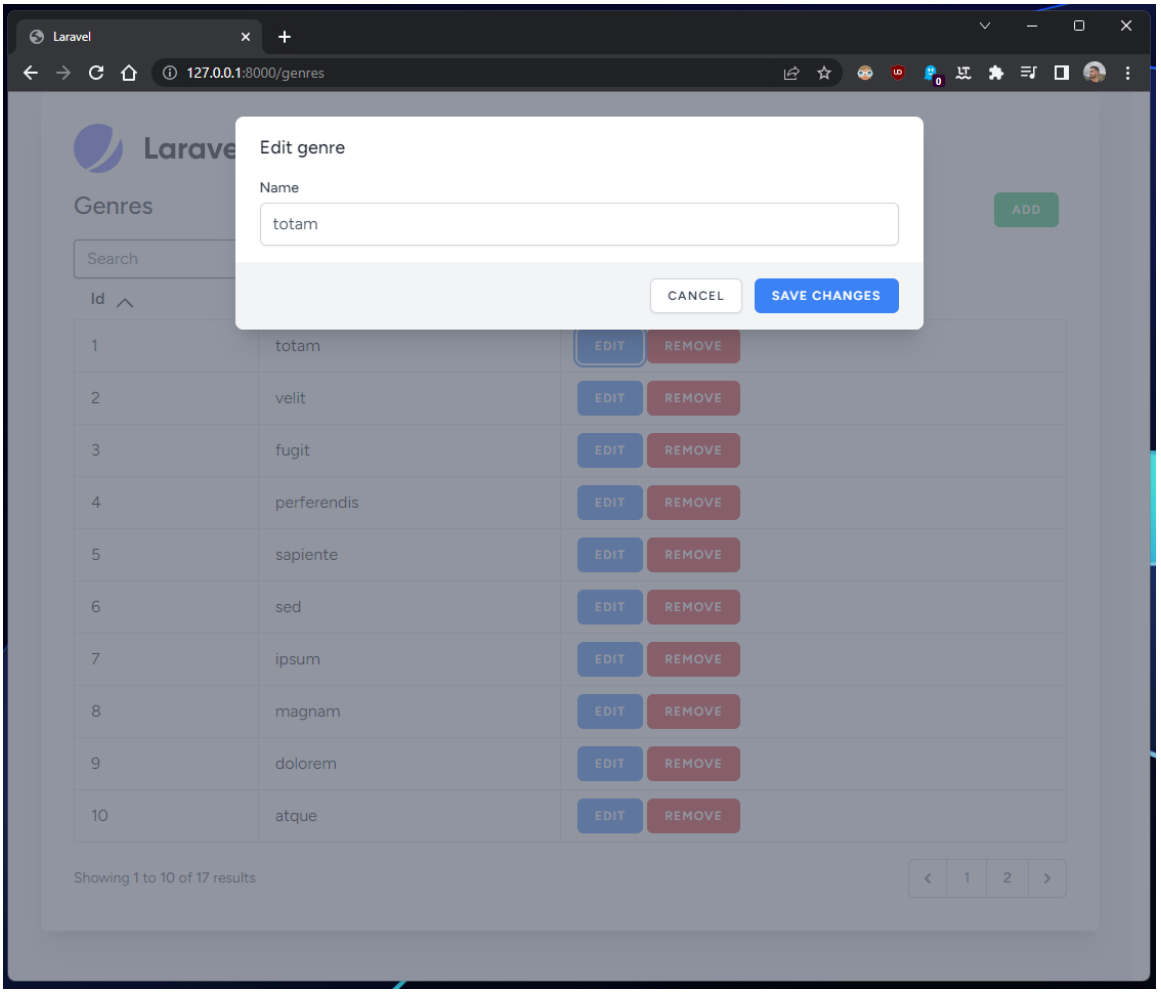
```blade
<div class="mt-4 text-2xl font-medium text-gray-900 flex justify-between">
    <div>{{ __('Movies') }}</div>
    <div class="mr-2">
        <x-add-button wire:click="confirmMovieAdd">
            {{ __('Add') }}
        </x-add-button>
    </div>
</div>
```

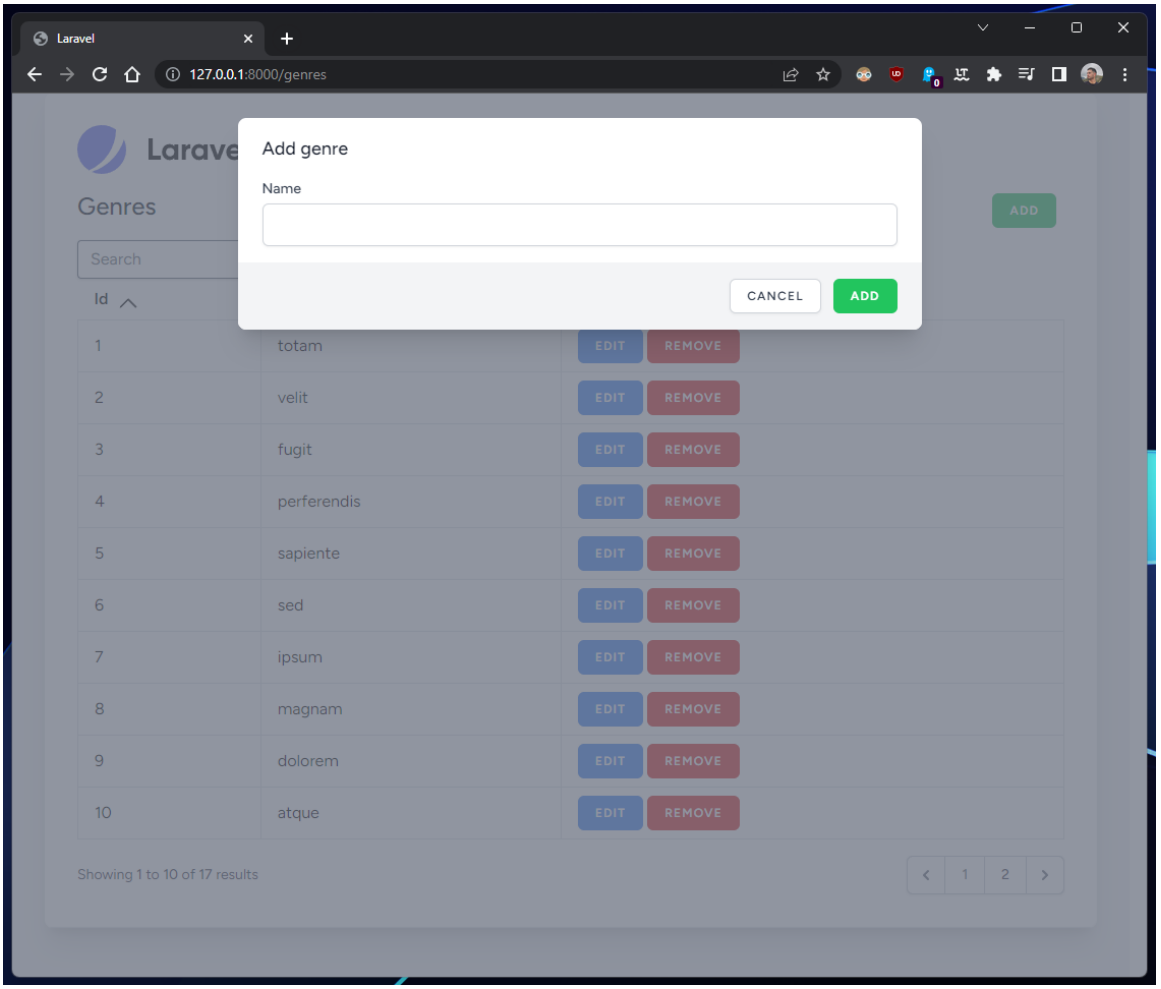
```
<tbody>
    @foreach ($movies as $movie)
        <tr>
            <td class="rounded border px-4 py-2">{{ $movie->id }}</td>
            <td class="rounded border px-4 py-2">{{ $movie->name }}</td>
            <td class="rounded border px-4 py-2">{{ $genres->find($movie->genre_id)->name }}</td>
            <td class="rounded border px-4 py-2">{{ $movie->duration }}</td>
            <td class="rounded border px-4 py-2">{{ $movie->director }}</td>
            @if (!$box_office)
                <td class="rounded border px-4 py-2">{{ $movie->box_office ? 'Yes' : 'No' }}</td>
            @endif
            <td class="rounded border px-4 py-2">
                <x-edit-button wire:click="confirmMovieEdit ({{ $movie->id }})">{{ __('Edit') }}</x-edit-button>
                <x-danger-button wire:click="confirmMovieDeletion ({{ $movie->id }})" wire:loading.attr="disabled">{{ __('Remove') }}</x-danger-button>
            </td>

        </tr>
    @endforeach
</tbody>
```

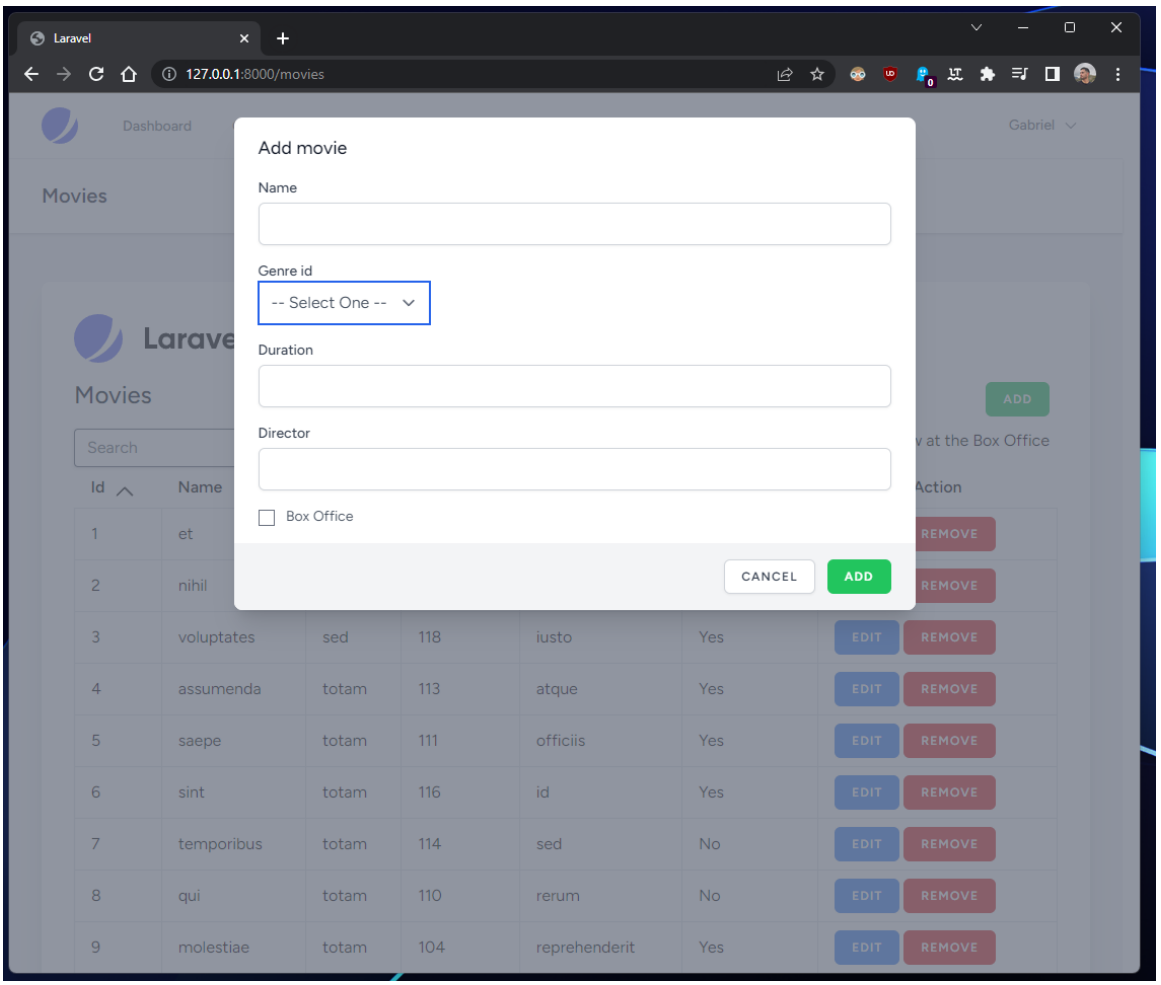
```
<x-dialog-modal wire:model="confirmingMovieAdd">
    <x-slot name="title">
        {{ isset($this->movie->id) ? __('Edit movie') : __('Add movie') }}
    </x-slot>

    <x-slot name="content">
        <div class="col-span-6 sm:col-span-4 mt-4">
            <x-label for="name" value="{{ __('Name') }}" />
            <x-input id="movie.name" type="text" class="mt-1 block w-full" wire:model.defer="movie.name" />
            <x-input-error for="movie.name"  class="mt-2" />
        </div>
        <div class="col-span-6 sm:col-span-4 mt-4">
            <x-label for="name" value="{{ __('Genre id') }}" />
            <select name="genre_id" wire:model.defer="movie.genre_id">
                <option value="">-- Select One --</option>
                @foreach ($genres as $genre)
                    <option value="{{ $genre->id }}">{{ $genre->name }}</option>
                @endforeach
            </select>
            <x-input-error for="movie.genre_id" class="mt-2" />
        </div>
        <div class="col-span-6 sm:col-span-4 mt-4">
            <x-label for="name" value="{{ __('Duration') }}" />
            <x-input id="movie.duration" type="number" class="mt-1 block w-full"
                wire:model.defer="movie.duration" />
            <x-input-error for="movie.duration" class="mt-2" />
        </div>
        <div class="col-span-6 sm:col-span-4 mt-4">
            <x-label for="name" value="{{ __('Director') }}" />
            <x-input id="movie.director" type="text" class="mt-1 block w-full"
                wire:model.defer="movie.director" />
            <x-input-error for="movie.director" class="mt-2" />
        </div>
```

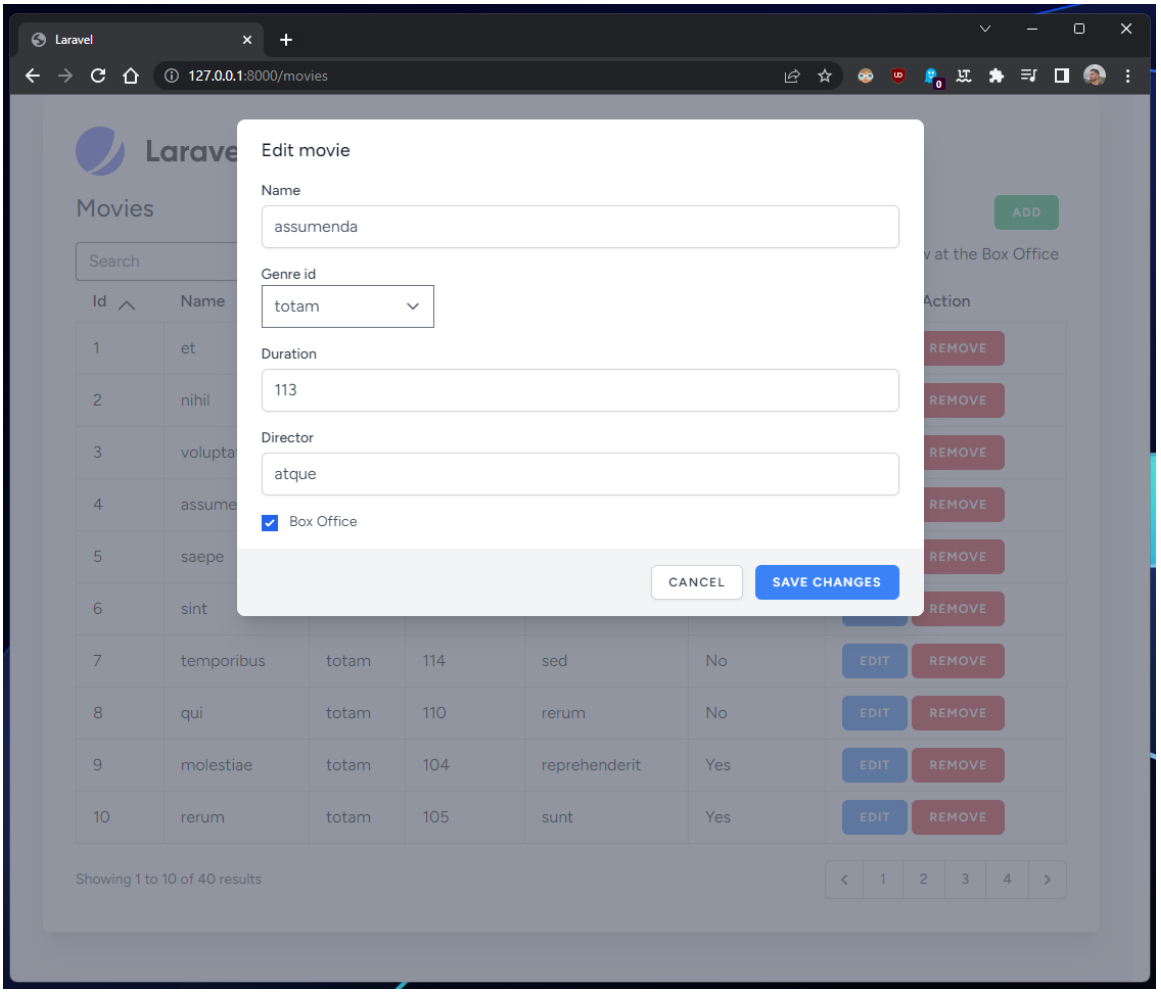
```
        <div class="col-span-6 sm:col-span-4 mt-4">
            <input type="checkbox" wire:model.defer="movie.box_office" />
            <span class="ml-2 text-sm text-gray-600">{{ __('Box Office') }}</span>

        </div>
    </x-slot>
    <x-slot name="footer">
        <x-secondary-button wire:click="$toggle('confirmingMovieAdd', false)" wire:loading.attr="disabled">
            {{ __('Cancel') }}
        </x-secondary-button>

        @if (isset($this->movie->id))
        <x-edit-button class="ml-3" wire:click="addMovie ()" wire:loading.attr="disabled">
            {{ __('Save changes') }}
        </x-edit-button>
        @else
            <x-add-button class="ml-3" wire:click="addMovie ()" wire:loading.attr="disabled">
                {{ __('Add') }}
            </x-add-button>
        @endif

    </x-slot>
</x-dialog-modal>
```

22.7. Con esto ya podéis añadir y editar géneros y películas.


Laravel
127.0.0.1:8000/genres
Edit genre
Name
totam
CANCEL
SAVE CHANGES
Genres
ADD
Search
Id
1 totam EDIT REMOVE
2 velit EDIT REMOVE
3 fugit EDIT REMOVE
4 perferendis EDIT REMOVE
5 sapiente EDIT REMOVE
6 sed EDIT REMOVE
7 ipsum EDIT REMOVE
8 magnam EDIT REMOVE
9 dolorem EDIT REMOVE
10 atque EDIT REMOVE
Showing 1 to 10 of 17 results
1
2


Laravel
127.0.0.1:8000/genres
Genres
Search
ADD
Add genre
Name
CANCEL
ADD
Id
1 totam
2 velit
3 fugit
4 perferendis
5 sapiente
6 sed
7 ipsum
8 magnam
9 dolorem
10 atque
EDIT
REMOVE
Showing 1 to 10 of 17 results
1
2


Laravel
127.0.0.1:8000/movies
Dashboard
Gabriel
Movies
Add movie
Name
Genre id
-- Select One --
Duration
Director
Box Office
CANCEL
ADD
Laravel
Movies
ADD
Search
Id
Name
Action
1 et REMOVE
2 nihil REMOVE
3 voluptates sed 118 iusto Yes EDIT REMOVE
4 assumenda totam 113 atque Yes EDIT REMOVE
5 saepe totam 111 officiis Yes EDIT REMOVE
6 sint totam 116 id Yes EDIT REMOVE
7 temporibus totam 114 sed No EDIT REMOVE
8 qui totam 110 rerum No EDIT REMOVE
9 molestiae totam 104 reprehenderit Yes EDIT REMOVE



23. Añadiendo dos vistas más.

Añadiremos dos vistas más en la que usuarios que estén o no registrados podrán ver todas las películas y géneros, eso sí con nombres distintos. Es decir no mostraremos las repetidas. Además sólo podrán verlos.

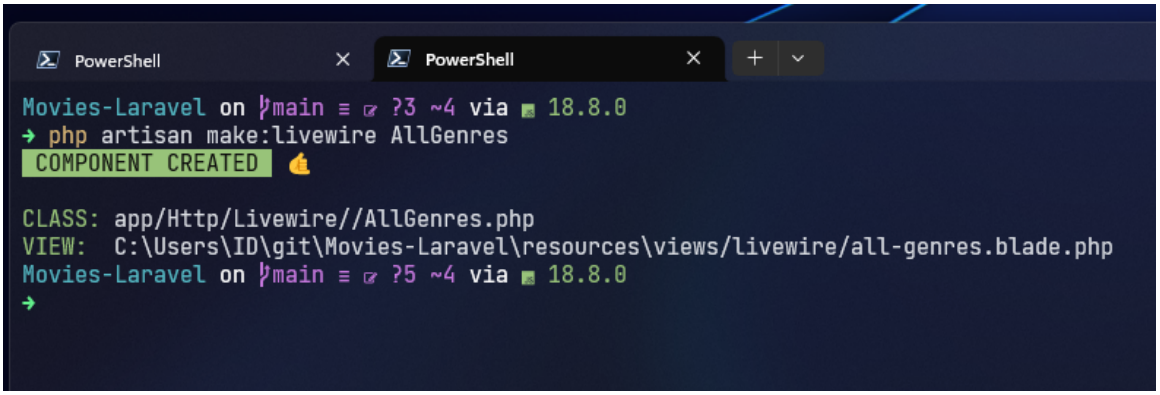
23.1. Antes de nada iremos a config\database.php y cambiaremos el siguiente valor de true a false

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => false,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
    ]) : [],
],
```

23.2. Comenzamos añadiendo las vistas y los controladores. Vamos a la consola y ejecutamos lo siguiente.

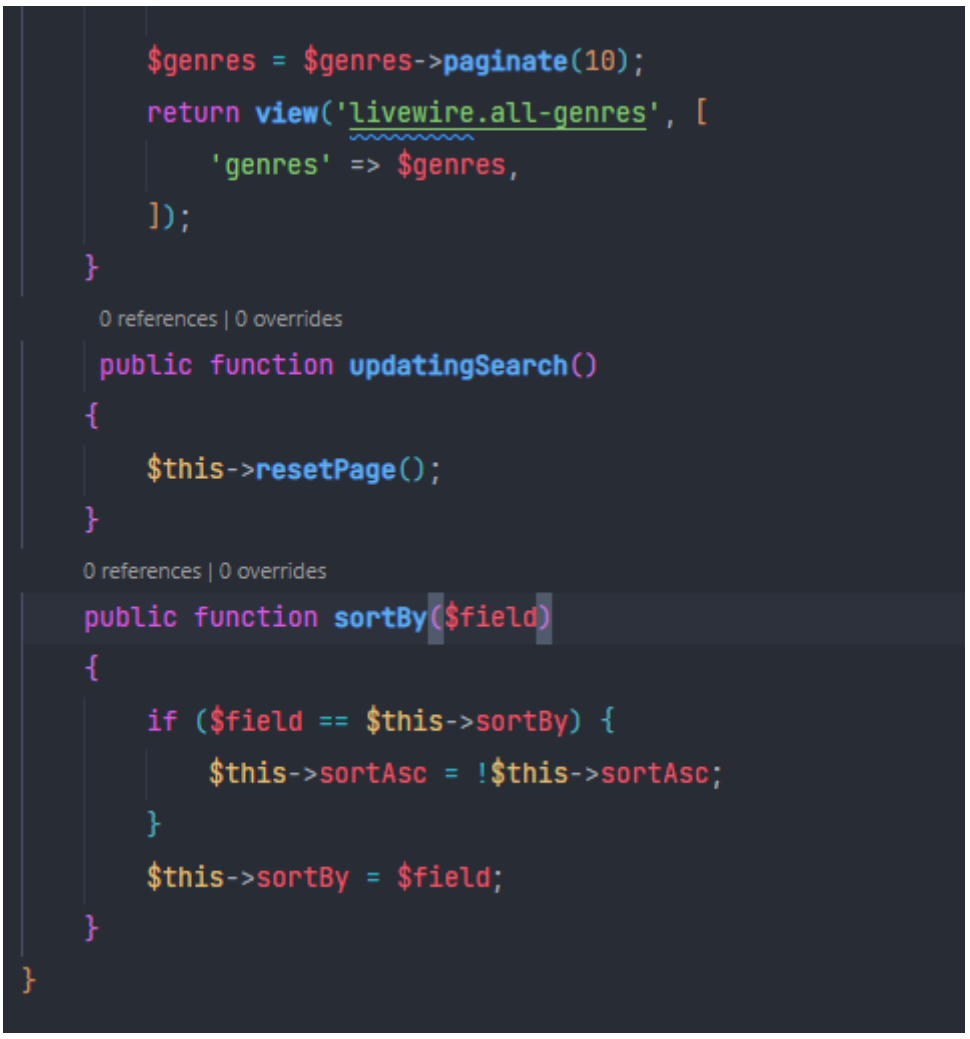
```
php artisan make:livewire AllGenres
```

```
php artisan make:livewire AllMovies
```



23.3. Empezaremos modificando los controladores (app\Http\Livewire\AllGenres.php) y (app\Http\Livewire\AllMovies.php) deben quedarnos algo así.

```php
namespace App\Http\Livewire;

use Livewire\Component;
use App\Models\Genre;

0 references | 0 implementations
class AllGenres extends Component
{
    2 references
    public $search;
    3 references
    public $sortBy = 'id';
    3 references
    public $sortAsc = true;
    1 reference
    protected $queryString = [
        'search' => ['except' => ''],
        'sortBy' => ['except' => 'id'],
        'sortAsc' => ['except' => true]
    ];
    0 references | 0 overrides
    public function render()
    {
        $genres = Genre::select('*')->groupBy('name')
            ->when($this->search, function ($query) {
                return $query->where(
                    function ($query) {
                            $query->where('name', 'like', '%' . $this->search . '%');
                        }
                );
            })
            ->orderBy($this->sortBy, $this->sortAsc ? 'ASC' : 'DESC');
```



```
        $genres = $genres->paginate(10);
        return view('livewire.all-genres', [
            'genres' => $genres,
        ]);
    }
    0 references | 0 overrides
    public function updatingSearch()
    {
        $this->resetPage();
    }
    0 references | 0 overrides
    public function sortBy($field)
    {
        if ($field == $this->sortBy) {
            $this->sortAsc = !$this->sortAsc;
        }
        $this->sortBy = $field;
    }
}
```

```php
use App\Models\Genre;
use App\Models\Movie;

0 references | 0 implementations
class AllMovies extends Component
{
    use WithPagination;

    1 reference
    public $box_office;
    3 references
    public $search;
    3 references
    public $sortBy = 'id';
    3 references
    public $sortAsc = true;

    1 reference
    protected $queryString = [
        'search' => ['except' => ''],
        'box_office' => ['except' => false],
        'sortBy' => ['except' => 'id'],
        'sortAsc' => ['except' => true]
    ];

    0 references | 0 overrides
    public function render()
    {
        $movies = Movie::select('*')->groupBy('name')
            ->when($this->search, function ($query) {
                return $query->where(
                    function ($query) {
                            $query->where('name', 'like', '%' . $this->search . '%')
                                ->orWhere('director', 'like', '%' . $this->search . '%');
                        }
                );
            })
            ->when($this->box_office, function ($query) {
                return $query->boxOffice();
            })
            ->orderBy($this->sortBy, $this->sortAsc ? 'ASC' : 'DESC');
```

```php
        $movies = $movies->paginate(10);
        $genres = Genre::select('*')->get();

        return view('livewire.all-movies', [
            'movies' => $movies,
            'genres' => $genres,
        ]);
    }
    0 references | 0 overrides
    public function updatingBoxOffice()
    {
        $this->resetPage();
    }
    0 references | 0 overrides
    public function updatingSearch()
    {
        $this->resetPage();
    }
    0 references | 0 overrides
    public function sortBy($field)
    {
        if ($field == $this->sortBy) {
            $this->sortAsc = !$this->sortAsc;
        }
        $this->sortBy = $field;
    }

}
```

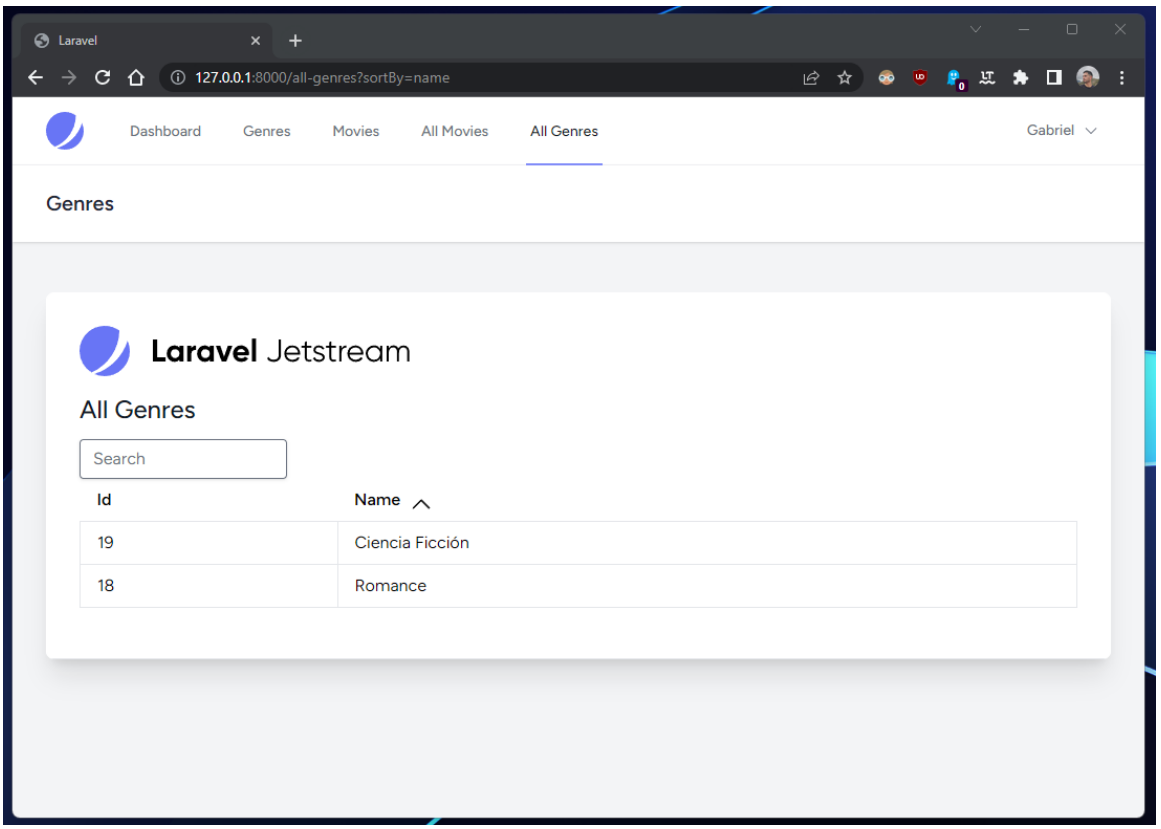
23.4. Vamos a la carpeta (resources\views\content) y creamos dos nuevos ficheros para nuestra vistas de la misma forma que hicimos anteriormente pero con all-genres y all-movies.

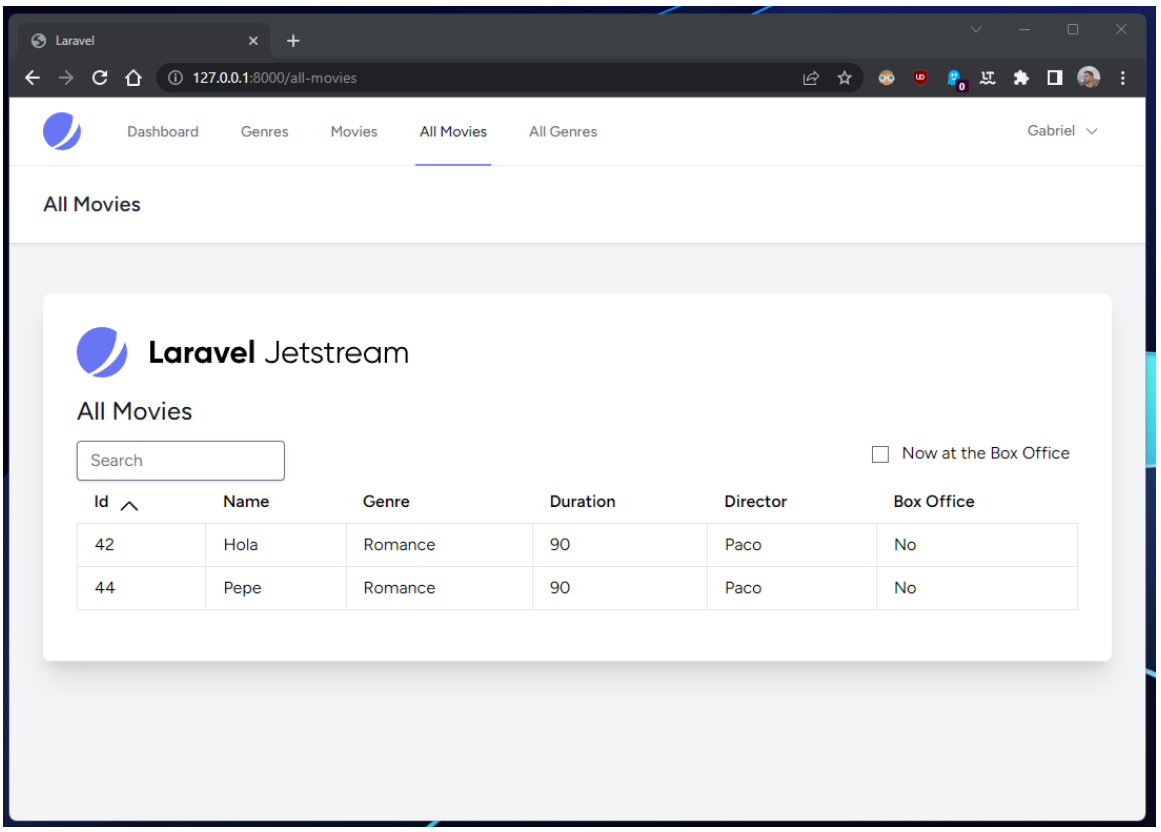
23.5. A continuación haremos la rutas en el archivo (routes\web.php)

```php
Route::get('/all-movies', function () {
    return view('content.all-movies');
})->name('all-movies');

Route::get('/all-genres', function () {
    return view('content.all-genres');
})->name('all-genres');
```

23.6. Y por último ya sólo tendríamos que hacer los links en (resources\views\navigation-menu.blade.php) de la misma forma que hicimos anteriormente.

23.7. Por último sólo nos quedará probarlo.

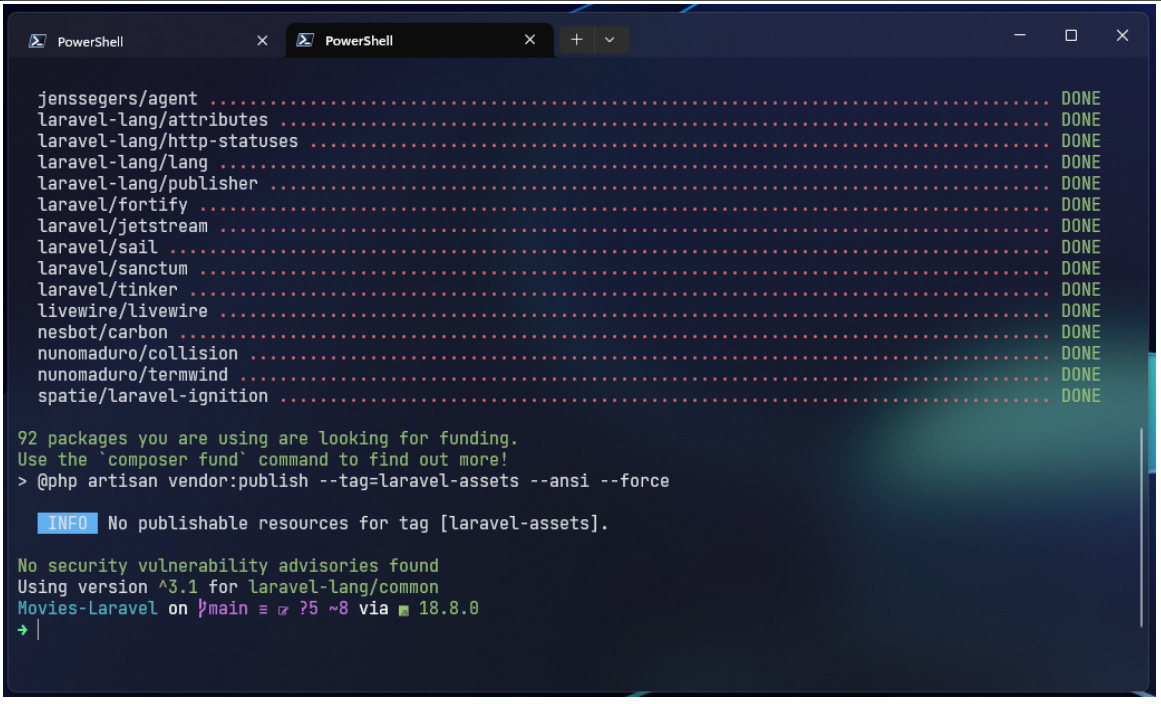




24. Traducción de los mensajes de validación y el resto de elementos

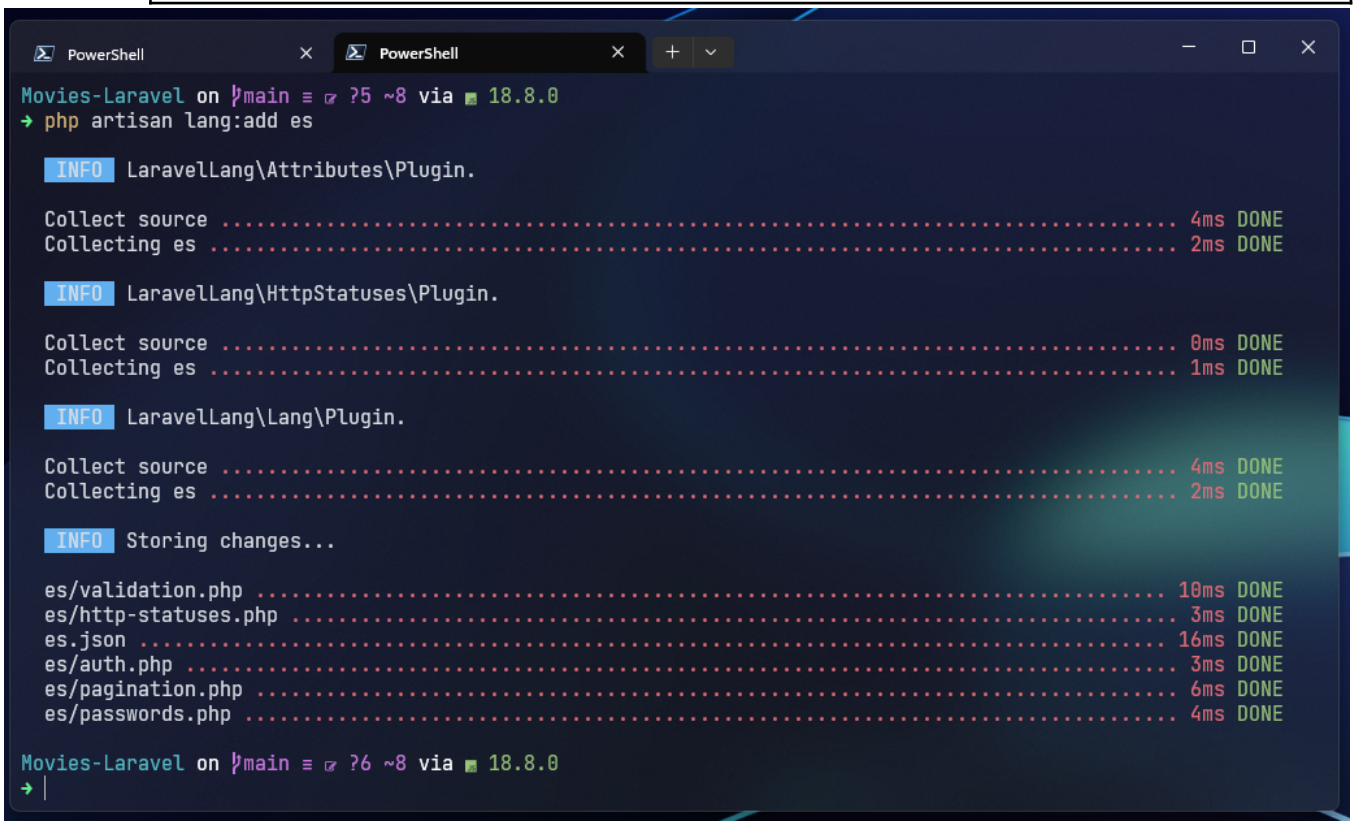
24.1. Vamos a la consola de comandos y ejecutamos

```
composer require laravel-lang/common --dev
```



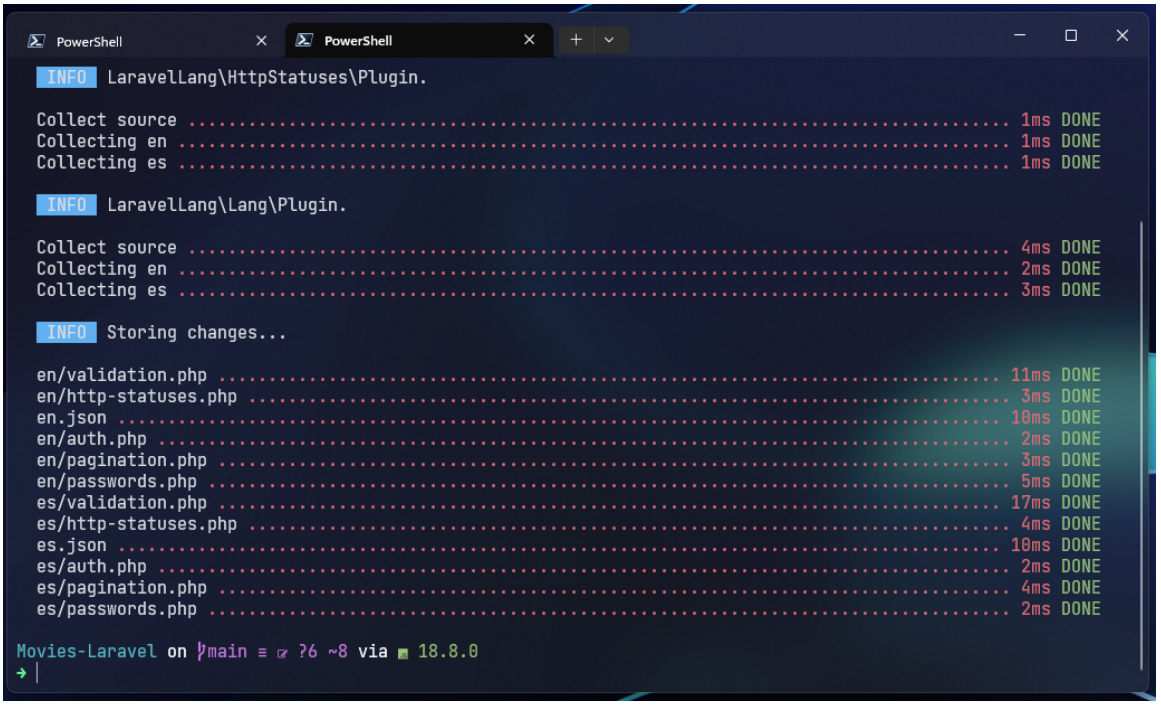
24.2. A continuación ejecutamos

```
php artisan lang:add es
```

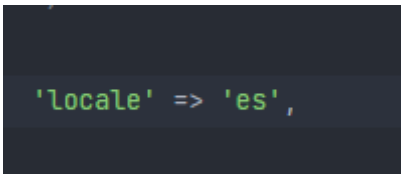


24.3. Seguimos ejecutando.

```
php artisan lang:update
```

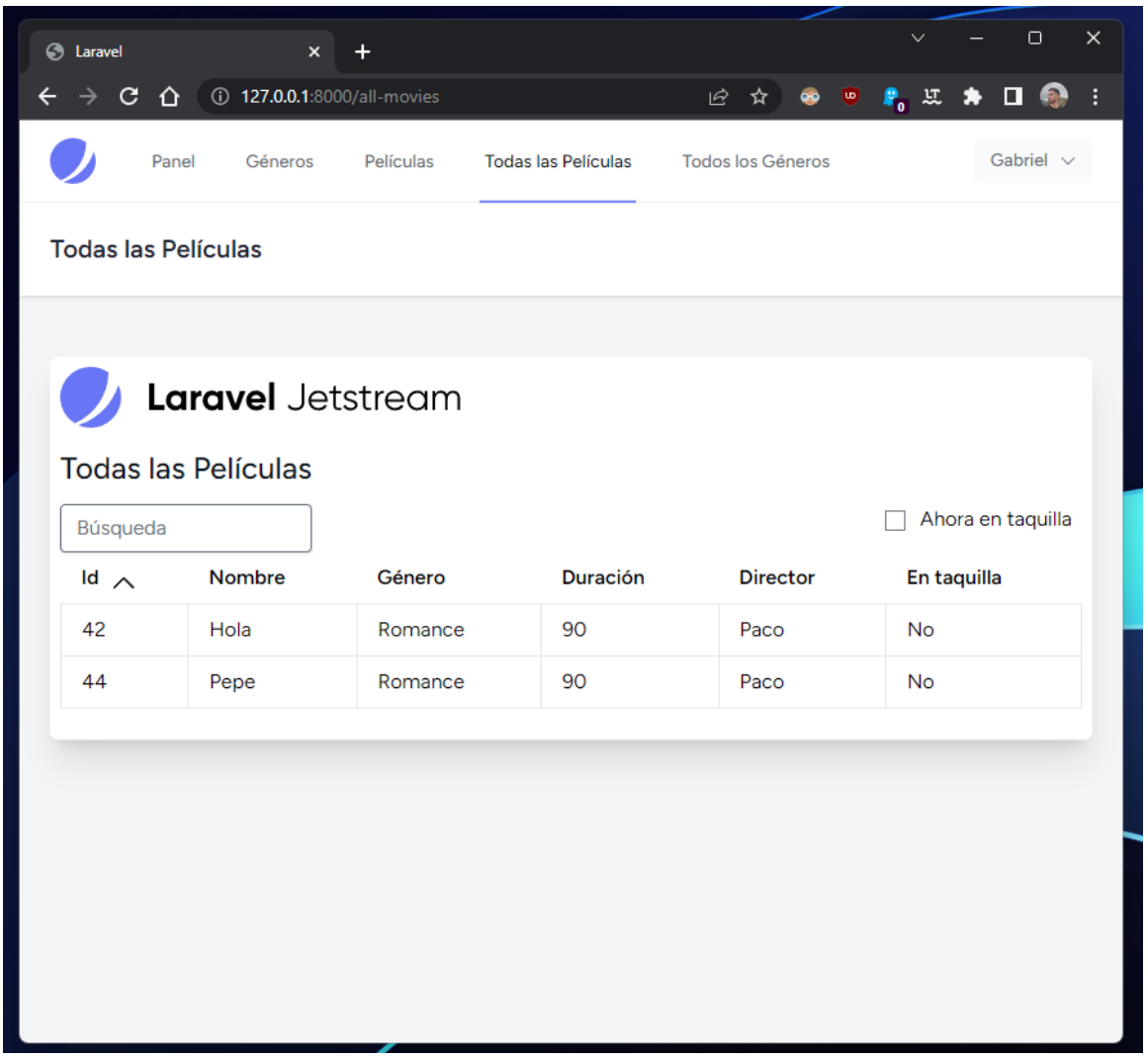


24.4. Nos dirigimos a (config\app.php) y cambiamos el siguiente valor de en a es.



24.5. Por último vamos a (lang\es.json) y añadimos las traducciones de los elementos de nuestra web. No es necesario añadir los de validación porque los comandos anteriores ya nos los generó por nosotros.

24.6. Vamos a nuestra web y vemos que todo está perfectamente corregido.



25. Con esto estarían todos los puntos a realizar en este proyecto .

26. Futuras mejoras:

26.1. Más vistas para usuarios no logueados que puedan ver las películas

26.2. Más opciones en las tablas. Como una descripción.

26.3. Incluir opción para votar películas.