

# **MANUAL PROGRAMADOR**

<b>1. Introducción</b>	<b>1</b>
<b>2. Requisitos:</b>	<b>2</b>
<b>3. Instalar requisitos</b>	<b>2</b>
<b>4. Crear base de datos</b>	<b>2</b>
<b>5. Crear proyecto</b>	<b>4</b>
<b>6. Instalación de Laravel Jetstream y Livewire</b>	<b>6</b>
<b>7. Crear Migraciones por defecto:</b>	<b>9</b>
<b>8. Crear seeder</b>	<b>10</b>
<b>9. Probamos el proyecto:</b>	<b>13</b>
<b>10. Crear Modelos, Factory y Migración</b>	<b>15</b>
<b>11. Rutas</b>	<b>32</b>
<b>14. Creación de componentes</b>	<b>39</b>
<b>15. Creación de las vistas</b>	<b>42</b>
<b>16. Añadimos Filtro de películas en taquilla</b>	<b>49</b>
<b>17. Añadir cuadro de búsqueda</b>	<b>53</b>
<b>18. Query String</b>	<b>59</b>
<b>19. Mejorando filtro en taquilla</b>	<b>63</b>
<b>20. Ordenación ascendente y descendente</b>	<b>66</b>
<b>21. Eliminación de datos</b>	<b>76</b>
<b>22. Añadir y editar datos</b>	<b>86</b>
<b>24. Traducción de los mensajes de validación y el resto de elementos</b>	<b>109</b>
<b>25. Con esto estarían todos los puntos a realizar en este proyecto .</b>	<b>113</b>

## **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:

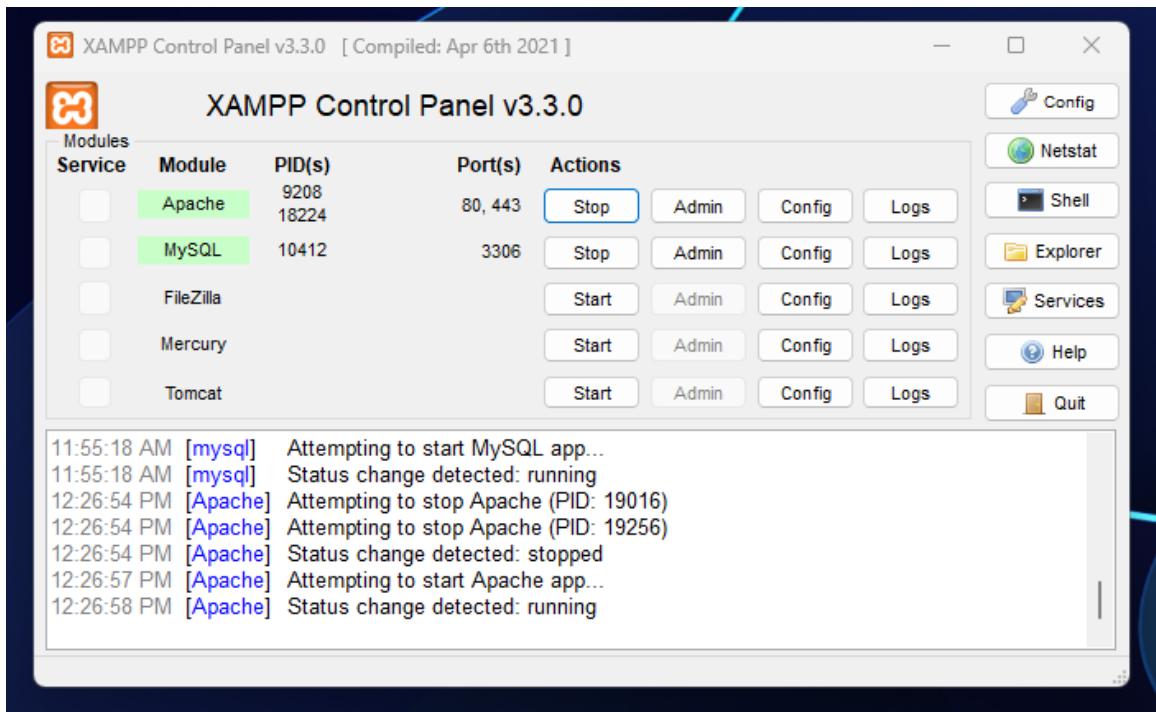
- 2.1. Composer
- 2.2. Artisan
- 2.3. Xampp (Base Local)
- 2.4. Node.js
- 2.5. Editor de texto o IDE (En este caso usamos Visual Studio Code)

## **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



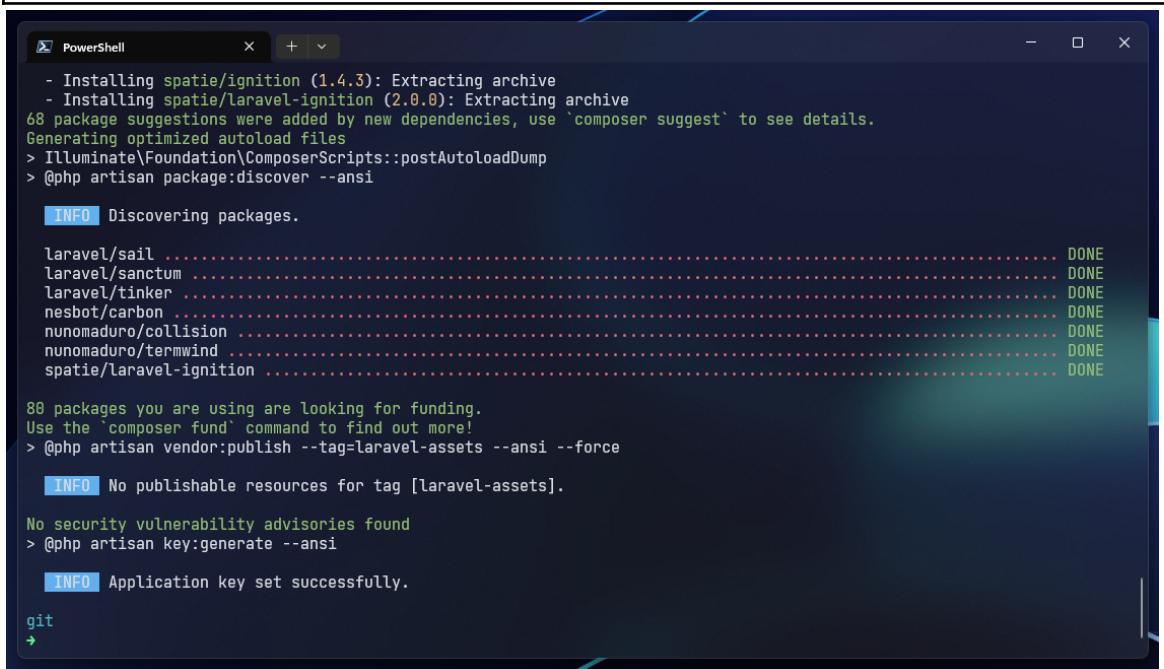
### 4.2. Abriremos phpMyAdmin y crearemos la base de datos “practicalaravel”

A screenshot of the phpMyAdmin interface. The left sidebar shows databases: Nueva, bank, information\_schema, mysql, performance\_schema, phpmyadmin, and test. The main area shows a "Bases de datos" table with columns: Base de datos, Cotejamiento, and Acción. A modal dialog titled "Crear base de datos" is open, showing the input field "practicalaravel" and a dropdown menu "utf8mb4\_general\_ci". At the bottom of the table, it says "Total: 6".

## 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
```

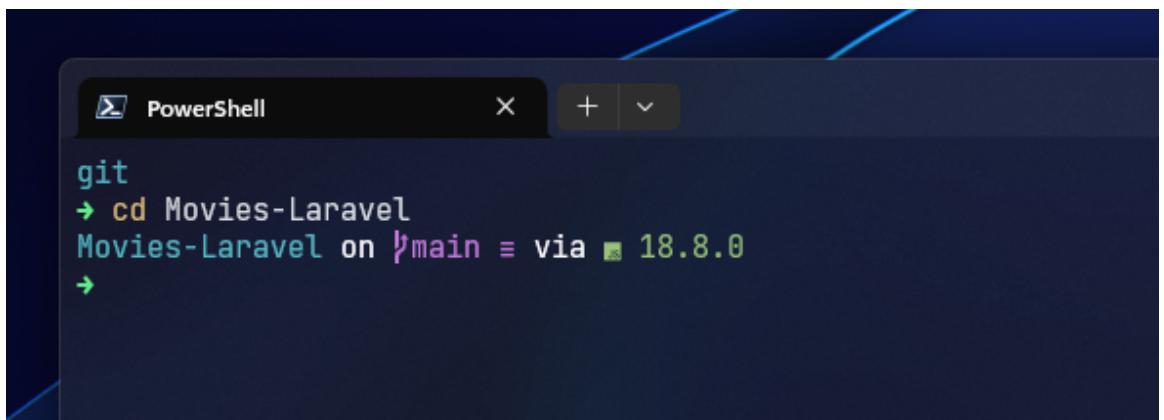


The screenshot shows a PowerShell window with the title bar "PowerShell". The command "composer create-project laravel/laravel Movies-Laravel" was run, and its output is displayed. The output includes:

- Installing spatie/ignition (1.4.3): Extracting archive
- Installing spatie/laravel-ignition (2.0.0): Extracting archive
- 68 package suggestions were added by new dependencies, use `composer suggest` to see details.
- Generating optimized autoload files
- > Illuminate\Foundation\ComposerScripts::postAutoloadDump
- > @php artisan package:discover --ansi
- INFO** Discovering packages.
- laravel/sail ..... DONE
- laravel/sanctum ..... DONE
- laravel/tinker ..... DONE
- nesbot/carbon ..... DONE
- nunomaduro/collision ..... DONE
- nunomaduro/termwind ..... DONE
- spatie/laravel-ignition ..... DONE
- 88 packages you are using are looking for funding.  
Use the 'composer fund' command to find out more!
- > @php artisan vendor:publish --tag=laravel-assets --ansi --force
- INFO** No publishable resources for tag [laravel-assets].
- No security vulnerability advisories found
- > @php artisan key:generate --ansi
- INFO** Application key set successfully.
- git
- 

5.2. Nos dirigimos a la carpeta del proyecto

```
cd Movies-Laravel
```



The screenshot shows a PowerShell window with the title bar "PowerShell". The command "cd Movies-Laravel" was run, and the output shows the current working directory has changed to "Movies-Laravel". The output is:

```
git
→ cd Movies-Laravel
Movies-Laravel on ✓main ≡ via 18.8.0
→
```

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

```
code .
```

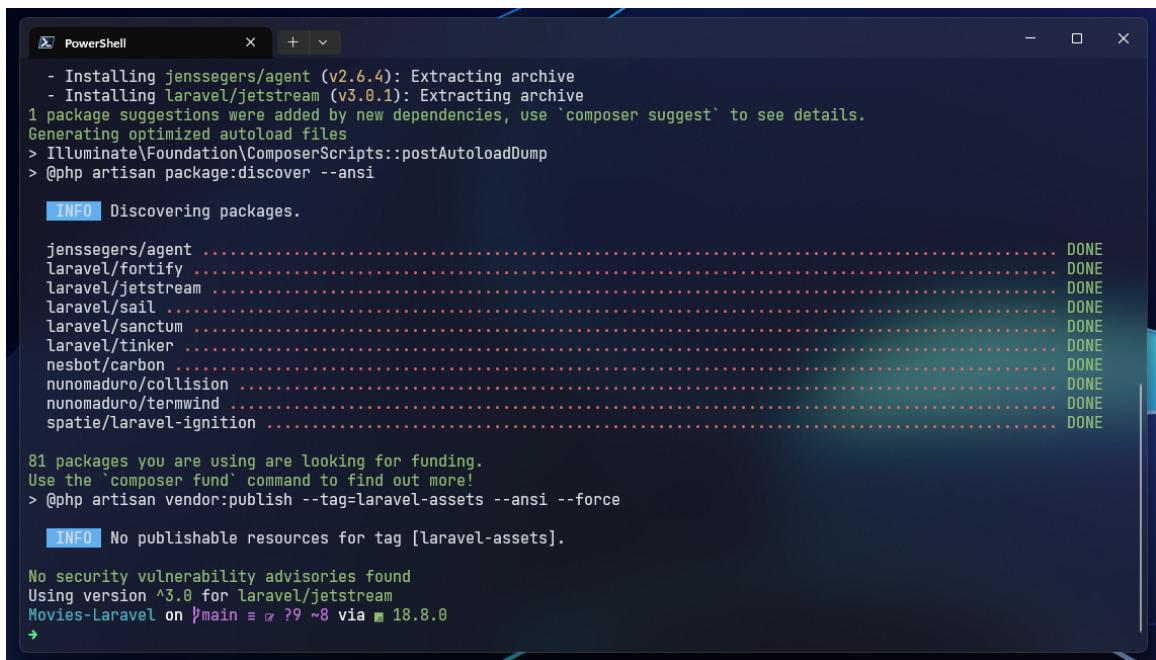
- 5.4. Nos dirigimos al archivo .env y lo editamos con el nombre de nuestra base de datos que creamos anteriormente.

```
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
```



```
- Installing jenssegers/agent (v2.6.4): Extracting archive
- Installing laravel/jetstream (v3.0.1): Extracting archive
1 package suggestions were added by new dependencies, use `composer suggest` to see details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

[INFO] Discovering packages.

jenssegers/agent ..... DONE
laravel/fortify ..... DONE
laravel/jetstream ..... DONE
laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition .....
```

81 packages you are using are looking for funding.  
Use the `composer fund` command to find out more!

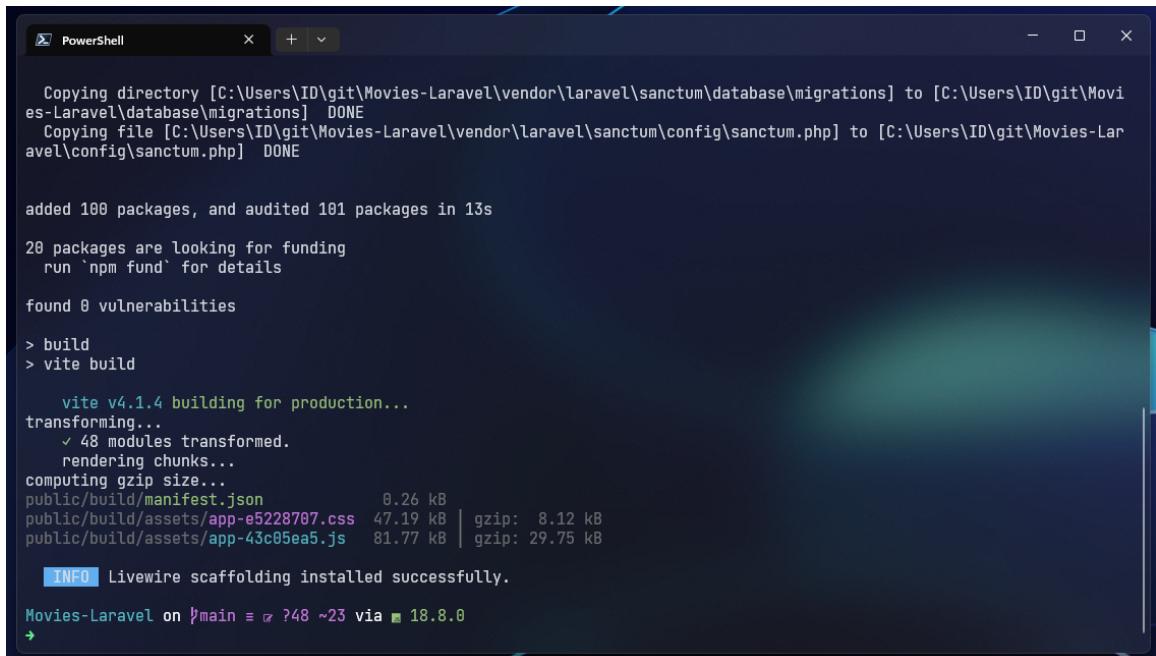
```
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

[INFO] No publishable resources for tag [laravel-assets].
```

No security vulnerability advisories found  
Using version ^3.0 for laravel/jetstream  
Movies-Laravel on \main = ✘ ?9 ~8 via ■ 18.8.0  
→

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

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



```
Copying directory [C:\Users\ID\git\Movies-Laravel\vendor\laravel\sanctum\database\migrations] to [C:\Users\ID\git\Movies-Laravel\database\migrations] DONE
Copying file [C:\Users\ID\git\Movies-Laravel\vendor\laravel\sanctum\config\sanctum.php] to [C:\Users\ID\git\Movies-Laravel\config\sanctum.php] DONE

added 100 packages, and audited 101 packages in 13s

28 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

> build
> vite build

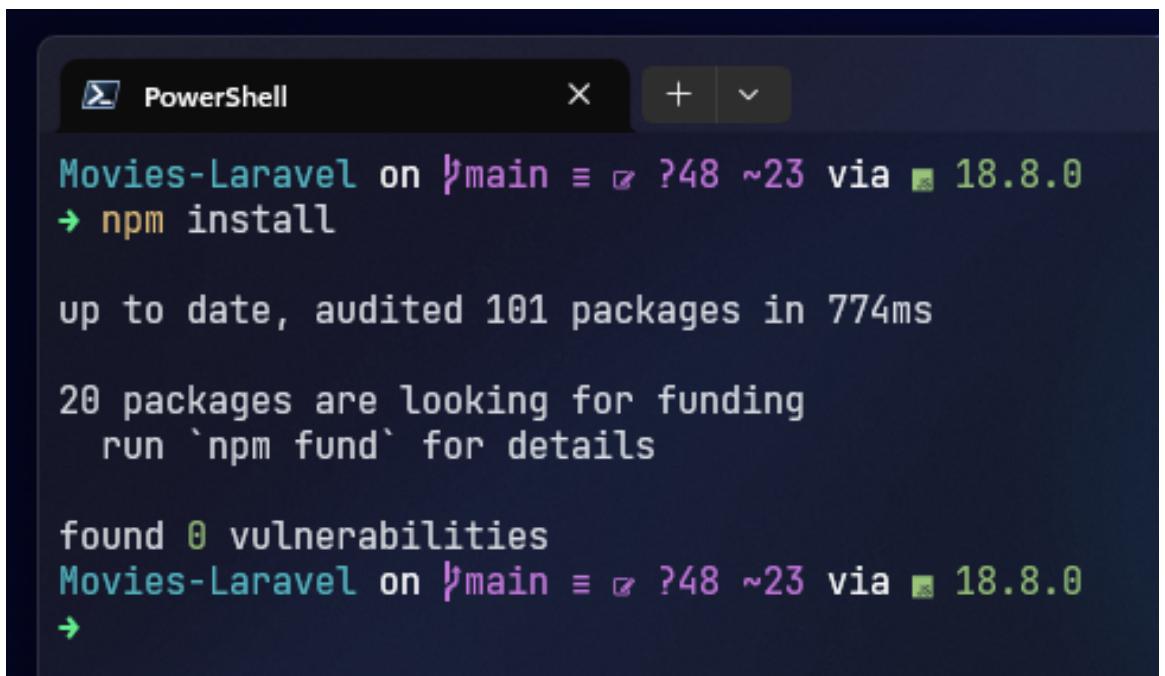
  vite v4.1.4 building for production...
transforming...
  ✓ 48 modules transformed.
    rendering chunks...
computing gzip size...
public/build/manifest.json      0.26 kB
public/build/assets/app-e5228707.css 47.19 kB | gzip: 8.12 kB
public/build/assets/app-43c05ea5.js 81.77 kB | gzip: 29.75 kB

[INFO] Livewire scaffolding installed successfully.

Movies-Laravel on \main = ✘ ?48 ~23 via ■ 18.8.0
→
```

6.3. Ejecutaremos:

```
npm install
```



The screenshot shows a PowerShell window titled "PowerShell". The command "npm install" was run, and the output indicates that 101 packages were audited in 774ms, with 20 packages looking for funding. No vulnerabilities were found. The terminal prompt ends with a green arrow.

```
Movies-Laravel on \main ✘ ?48 ~23 via █ 18.8.0
→ npm install

up to date, audited 101 packages in 774ms

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Movies-Laravel on \main ✘ ?48 ~23 via █ 18.8.0
→
```

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

```
npm run dev
```

```
VITE v4.1.4 ready in 751 ms
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h to show help

LARAVEL v10.1.5 plugin v0.7.4
→ APP_URL: http://localhost
```

## 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
```

```
Movies-Laravel on ⚡ main ✘ ?49 ~23 via m 18.8.0
+ php artisan migrate
INFO Preparing database.
Creating migration table ..... 26ms DONE
INFO Running migrations.
2014_10_12_000000_create_users_table ..... 33ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 45ms DONE
2014_10_12_200000_add_two_factor_columns_to_users_table ..... 11ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 31ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 39ms DONE
2023_02_25_121545_create_sessions_table ..... 78ms DONE

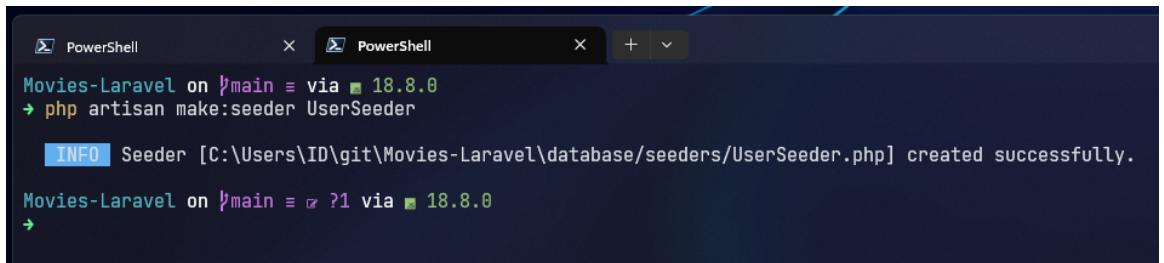
Movies-Laravel on ⚡ main ✘ ?49 ~23 via m 18.8.0
+
```

## 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:

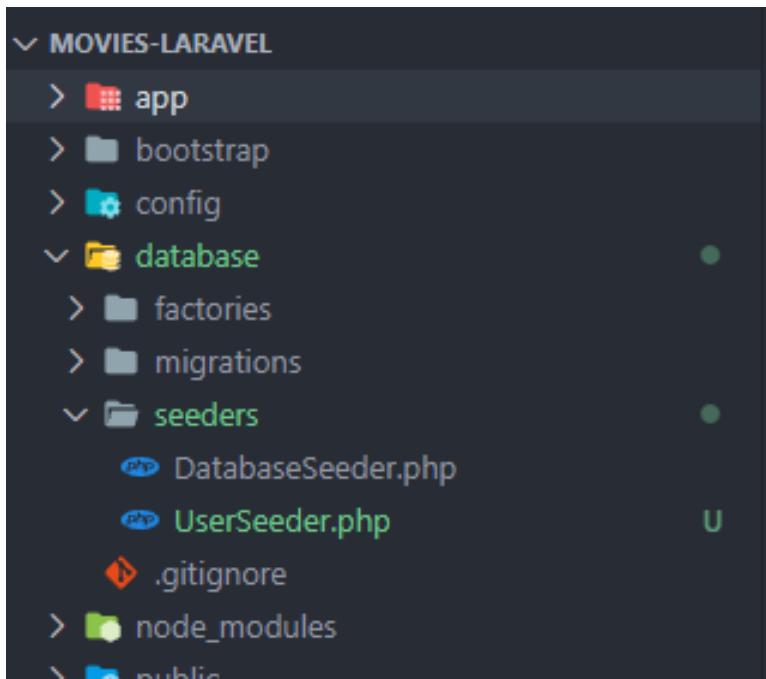
- 8.1. Comenzaremos creando nuestro seeder. Para eso vamos a nuestra consola y ejecutamos el siguiente comando.

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



The screenshot shows a terminal window with two tabs, both labeled "PowerShell". The current tab displays the command `php artisan make:seeder UserSeeder` and its output: `[INFO] Seeder [C:\Users\ID\git\Movies-Laravel\database\seeders\UserSeeder.php] created successfully.`. Below the command, the prompt shows the directory as `Movies-Laravel on \main ✘ 18.8.0`.

- 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\...
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6  use Illuminate\Database\Seeder;
7  use Illuminate\Support\Facades\DB;
8  use Illuminate\Support\Facades\Hash;
9
10
11
12 /**
13 * Run the database seeds.
14 */
15
16 public function run(): void
17 {
18     DB::table('users')->insert([
19         'name' => 'Gabriel',
20         'email' => 'gabriel@gmail.com',
21         'password' => Hash::make('12345678')
22     ]);
23 }
24
```

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

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

```

PowerShell
PowerShell

Movies-Laravel on \main ✘ ?1 via 18.8.0
→ php artisan db:seed --class=UserSeeder

INFO Seeding database.

Movies-Laravel on \main ✘ ?1 via 18.8.0
→

```

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

	id	name	email	email_verified_at	password
	1	Gabriel	gabriel@gmail.com	NULL	\$2y\$10\$ZwJQKm8qZBWk2Ec19YCXiu29Xd

## 9. Probamos el proyecto:

- 9.1. Para probarlo ejecutamos el comando:

```
php artisan serve
```

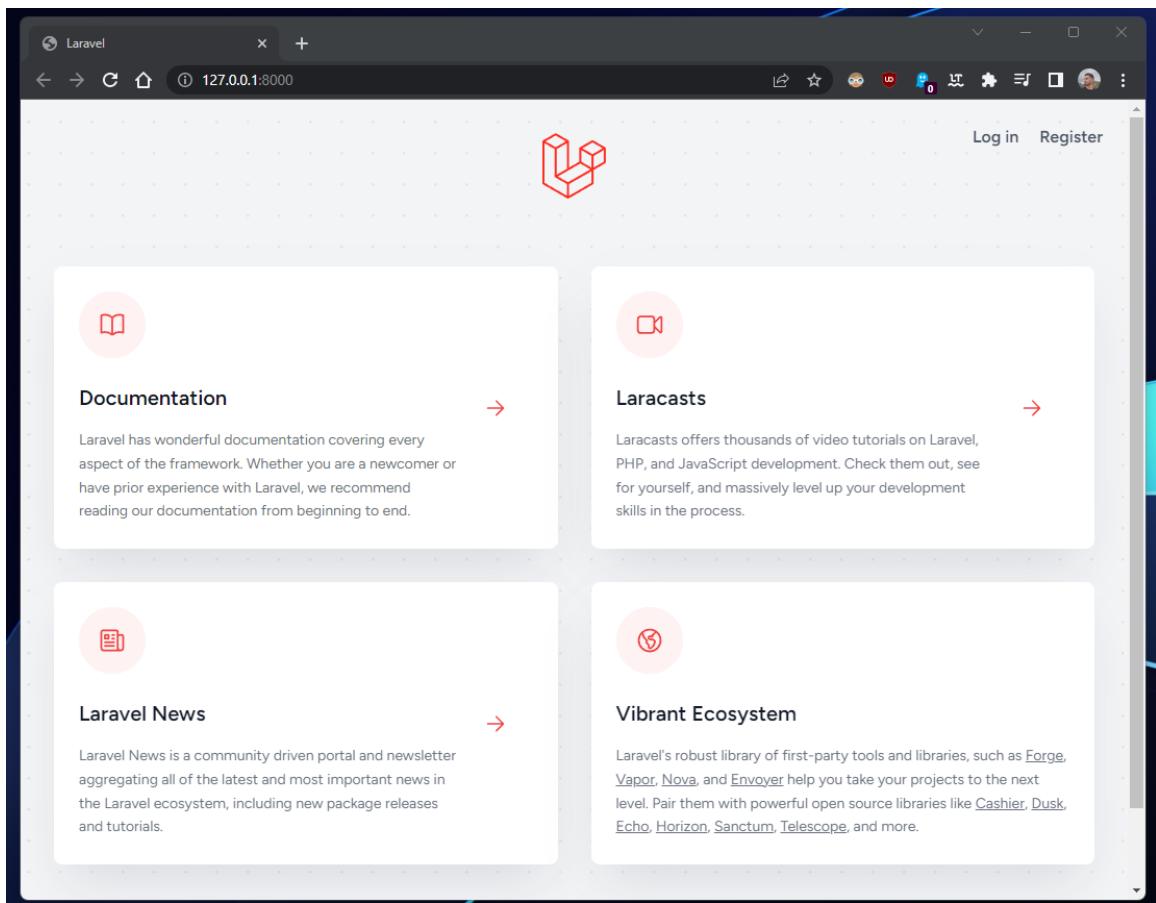
```

C:\WINDOWS\system32\cmd.exe ✘ PowerShell
Movies-Laravel on \main ✘ ?49 ~23 via 18.8.0
→ php artisan serve

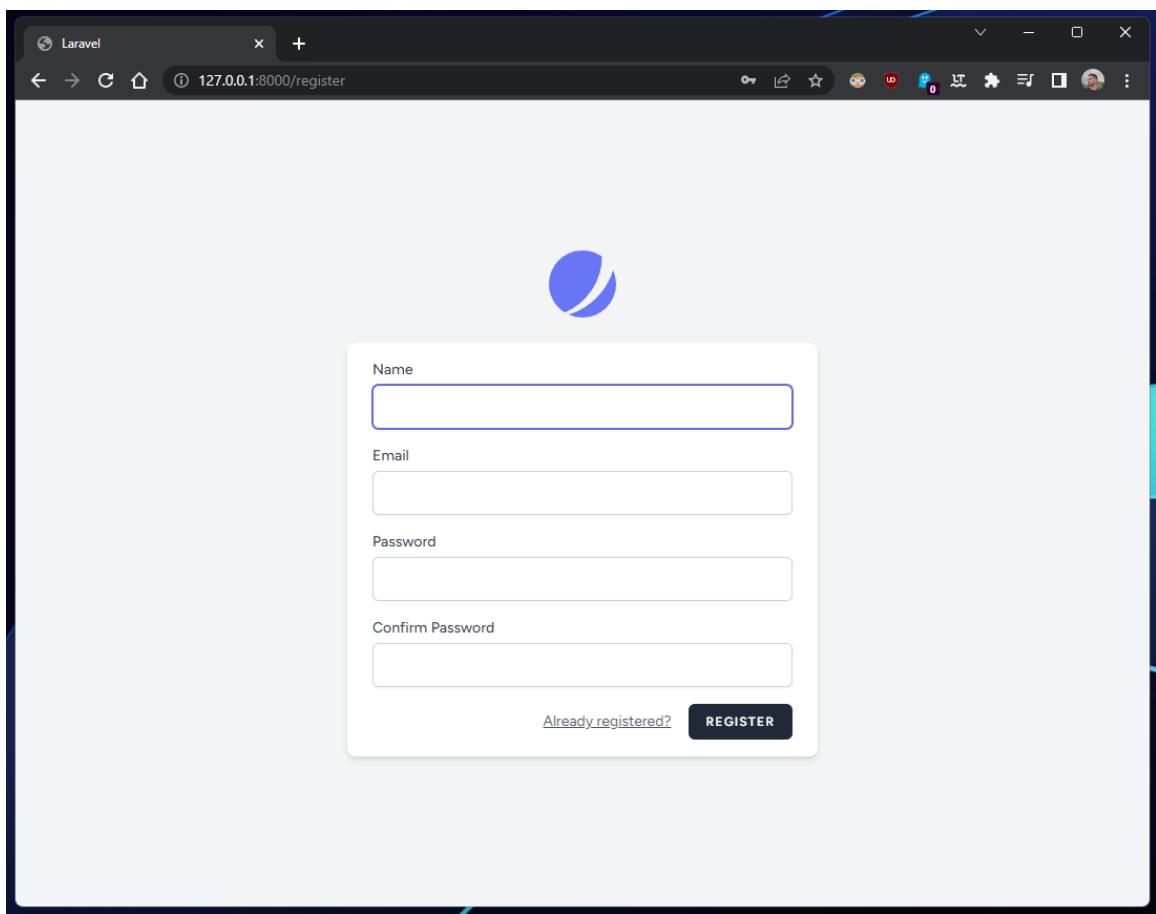
INFO Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server

```

- 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

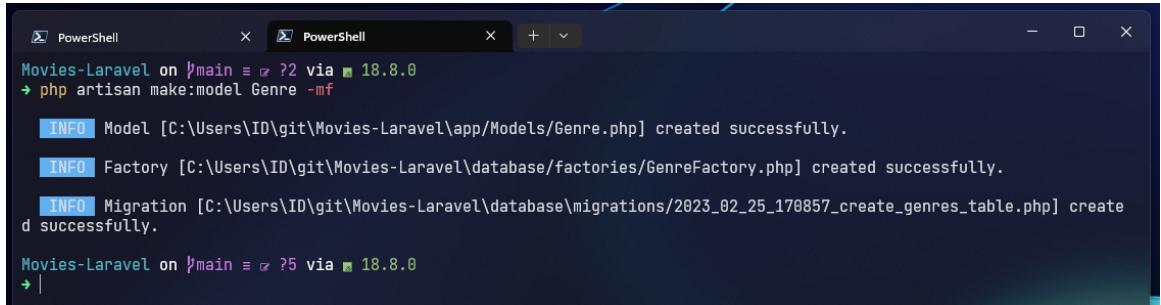
El framework de Laravel usa el paradigma Modelo-Vista-Controlador, por lo que es necesario crear cada uno de esos elementos.

Las Factories 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
```

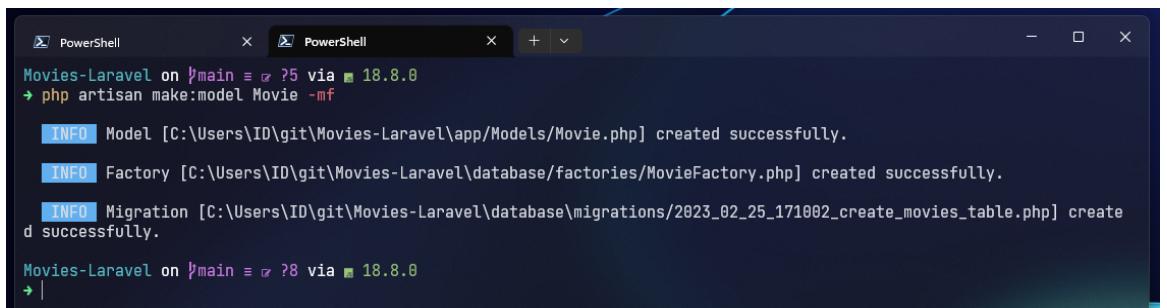


```
Movies-Laravel on \main ✘ 22 via █ 18.8.0
→ php artisan make:model Genre -mf

[INFO] Model [C:\Users\ID\git\Movies-Laravel\app\Models\Genre.php] created successfully.
[INFO] Factory [C:\Users\ID\git\Movies-Laravel\database\Factories\GenreFactory.php] created successfully.
[INFO] Migration [C:\Users\ID\git\Movies-Laravel\database\migrations\2023_02_25_170857_create_genres_table.php] created successfully.

Movies-Laravel on \main ✘ 25 via █ 18.8.0
→ |
```

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



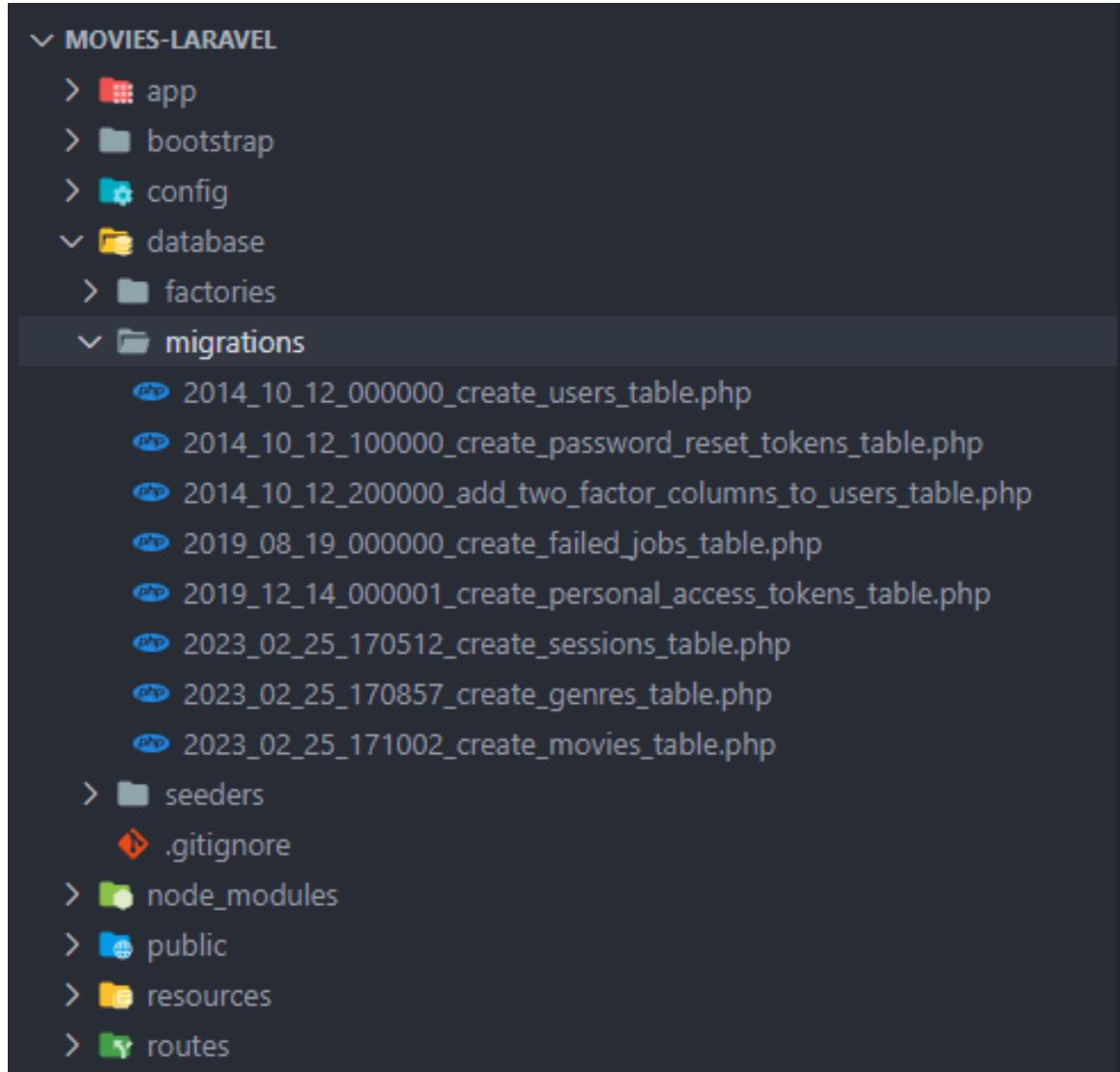
```
Movies-Laravel on \main ✘ 25 via █ 18.8.0
→ php artisan make:model Movie -mf

[INFO] Model [C:\Users\ID\git\Movies-Laravel\app\Models\Movie.php] created successfully.
[INFO] Factory [C:\Users\ID\git\Movies-Laravel\database\Factories\MovieFactory.php] created successfully.
[INFO] Migration [C:\Users\ID\git\Movies-Laravel\database\migrations\2023_02_25_171002_create_movies_table.php] created successfully.

Movies-Laravel on \main ✘ 28 via █ 18.8.0
→ |
```

## 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

```
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

```

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
```

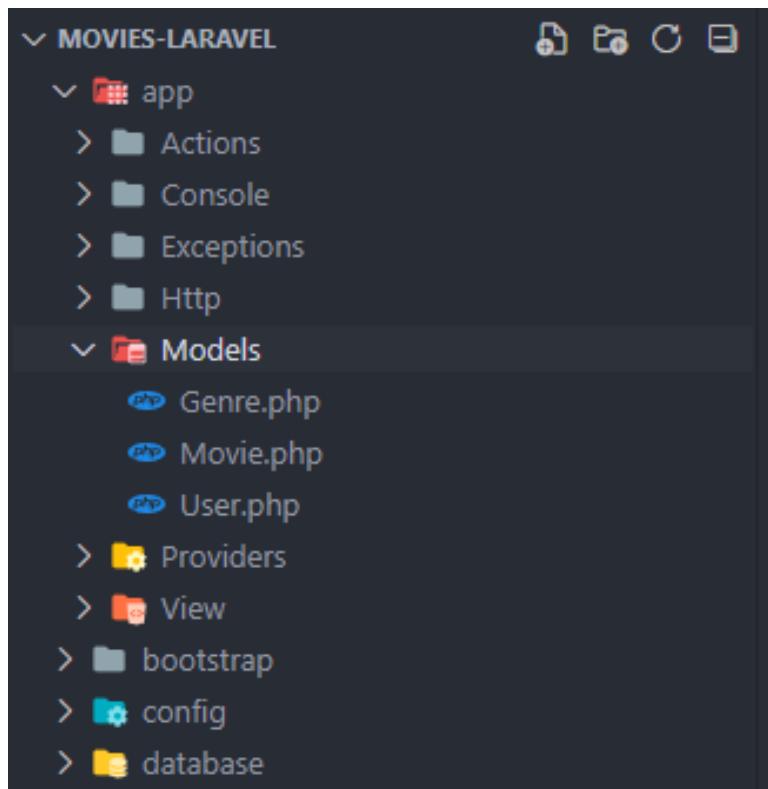
```

PowerShell
Movies-Laravel on \main  via m 18.8.0
→ php artisan migrate
[INFO] Running migrations.
2023_02_25_170857_create_genres_table : ..... 89ms DONE
2023_02_25_171002_create_movies_table : ..... 89ms DONE
Movies-Laravel on \main  via m 18.8.0
→

```

### 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...
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7 use App\Models\User;
8
9 reference | 0 implementations
9 class Genre extends Model
10 {
11     use HasFactory;
12     0 references
12     protected $fillable = ['name'];
13
14     0 references | 0 overrides
14     public function user() {
15         return $this->belongsTo(User::class);
16     }
17 }
18
```

10.3.3. Añadimos las siguientes para el modelo de película.

```
Movie.php M app\Models\Movie.php\PHP Intelephense\ Movie
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7 use App\Models\User;
8
9 1 reference | 0 implementations
10 class Movie extends Model
11 {
12     use HasFactory;
13
14     protected $fillable = ['name', 'duration', 'director', 'box_office'];
15
16     public function user() {
17         return $this->belongsTo(User::class);
18     }

```

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;
```

```
0 references | 0 overrides

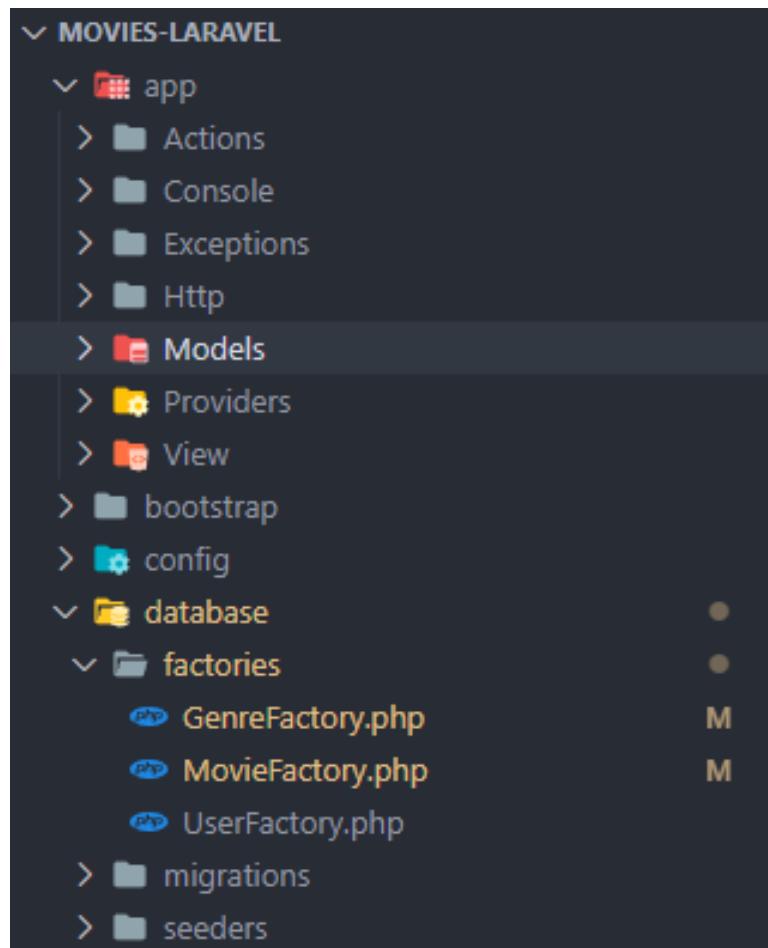
public function genres(){
    return $this->hasMany(Genre::class);
}
```

```
0 references | 0 overrides

public function movies(){
    return $this->hasMany(Movie::class);
}
```

## 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...
3 namespace Database\Factories;
4
5 use Illuminate\Database\Eloquent\Factories\Factory;
6 use App\Models\Genre;
7 use App\Models\User;
8
9 /**
10 * @extends \Illuminate\Database\Eloquent\Factories\Factory<App\Models\Genre>
11 */
12
13 class GenreFactory extends Factory
14 {
15     /**
16      * Define the model's default state.
17      *
18      * @return array<string, mixed>
19     */
20
21     public function definition(): array
22     {
23         return [
24             'user_id'=> User::factory(),
25             'name'=> $this->faker->word()
26         ];
27     }
28 }
```

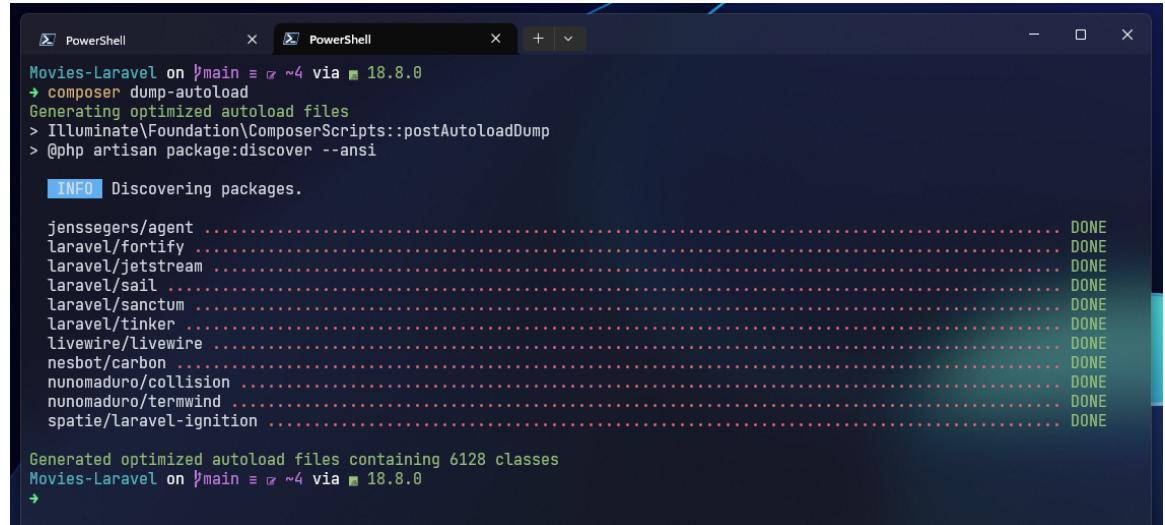
10.4.3. A continuación añadimos las siguientes líneas en película

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

5  ~ use Illuminate\Database\Eloquent\Factories\Factory;
6  ~ use App\Models\User;
7  ~ use App\Models\Genre;
8  ~ use App\Models\Movie;
9
10 ~ /**
11  * @extends \Illuminate\Database\Eloquent\Factories\Factory<App\Models\Movie>
12 */
13 class MovieFactory extends Factory
14 ~ {
15     protected $model = Movie::class;
16 ~ /**
17  * Define the model's default state.
18  *
19  * @return array<string, mixed>
20  */
21 public function definition(): array
22 ~ {
23     return [
24         'genre_id'=>Genre::factory(),
25         'user_id'=>User::factory(),
26         'name'=>$this->faker->word(),
27         'duration'=>$this->faker->numberBetween(90,120),
28         'director'=>$this->faker->word(),
29         'box_office'=>$this->faker->boolean()
30     ];
31 }
```

- 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



```
Movies-Laravel on 局长 ~4 via 18.8.0
→ composer dump-autoload
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

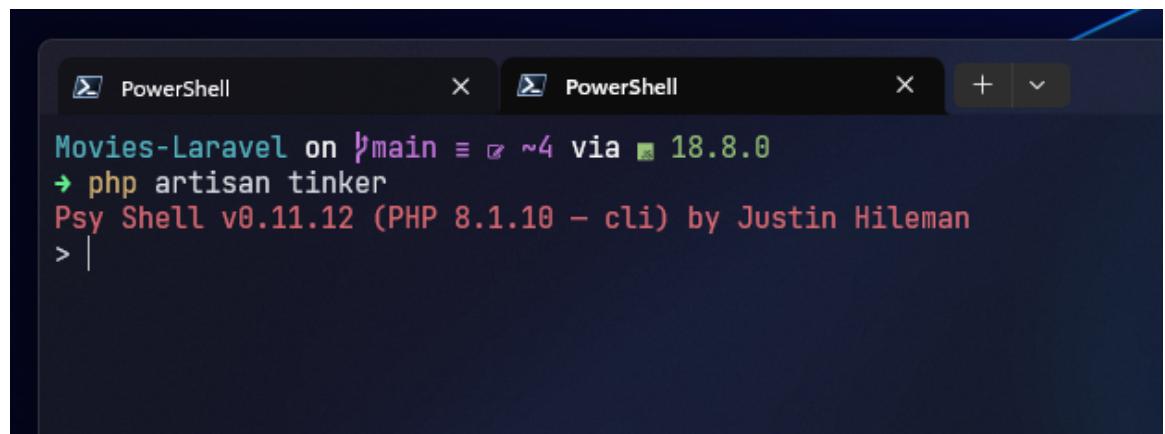
INFO Discovering packages.

jenssegers/agent ...
laravel/fortify ...
laravel/jetstream ...
laravel/sail ...
laravel/sanctum ...
laravel/tinker ...
livewire/livewire ...
nesbot/carbon ...
nunomaduro/collision ...
nunomaduro/termwind ...
spatie/laravel-ignition ...

Generated optimized autoload files containing 6128 classes
Movies-Laravel on 局长 ~4 via 18.8.0
→
```

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

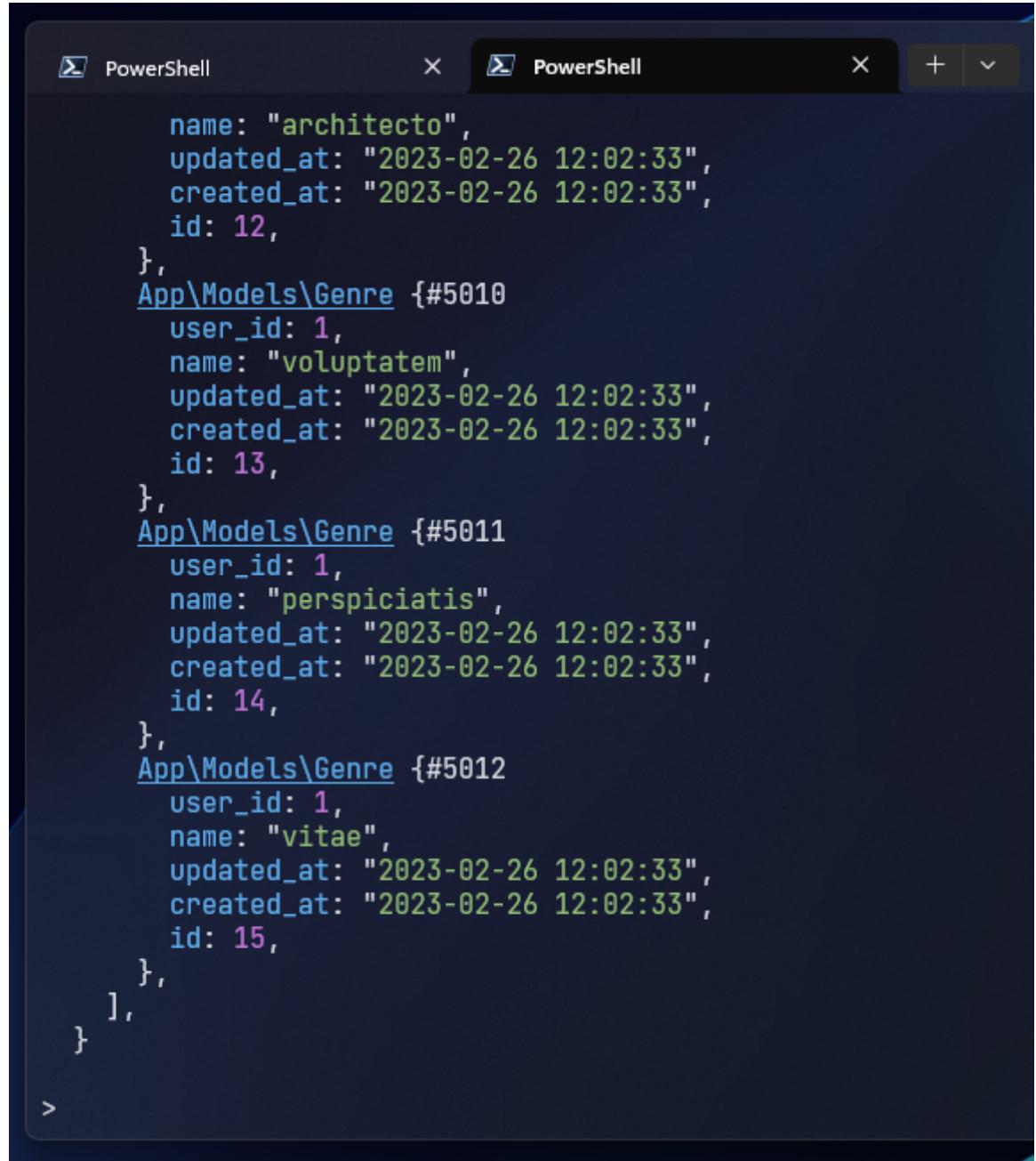
```
php artisan tinker
```



```
Movies-Laravel on 局长 ~4 via 18.8.0
→ php artisan tinker
Psy Shell v0.11.12 (PHP 8.1.10 - cli) by Justin Hileman
> |
```

- 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]);
```

A screenshot of a terminal window showing two PowerShell sessions. The left session displays a JSON array of five objects, each representing a 'Genre' model with attributes like name, updated\_at, created\_at, and id. The right session shows a single object, also a 'Genre' model, with similar attributes. Both sessions are running on a dark-themed interface.

```
name: "architecto",
updated_at: "2023-02-26 12:02:33",
created_at: "2023-02-26 12:02:33",
id: 12,
},
App\Models\Genre {#5010
    user_id: 1,
    name: "voluptatem",
    updated_at: "2023-02-26 12:02:33",
    created_at: "2023-02-26 12:02:33",
    id: 13,
},
App\Models\Genre {#5011
    user_id: 1,
    name: "perspiciatis",
    updated_at: "2023-02-26 12:02:33",
    created_at: "2023-02-26 12:02:33",
    id: 14,
},
App\Models\Genre {#5012
    user_id: 1,
    name: "vitae",
    updated_at: "2023-02-26 12:02:33",
    created_at: "2023-02-26 12:02:33",
    id: 15,
},
],
}
>
```

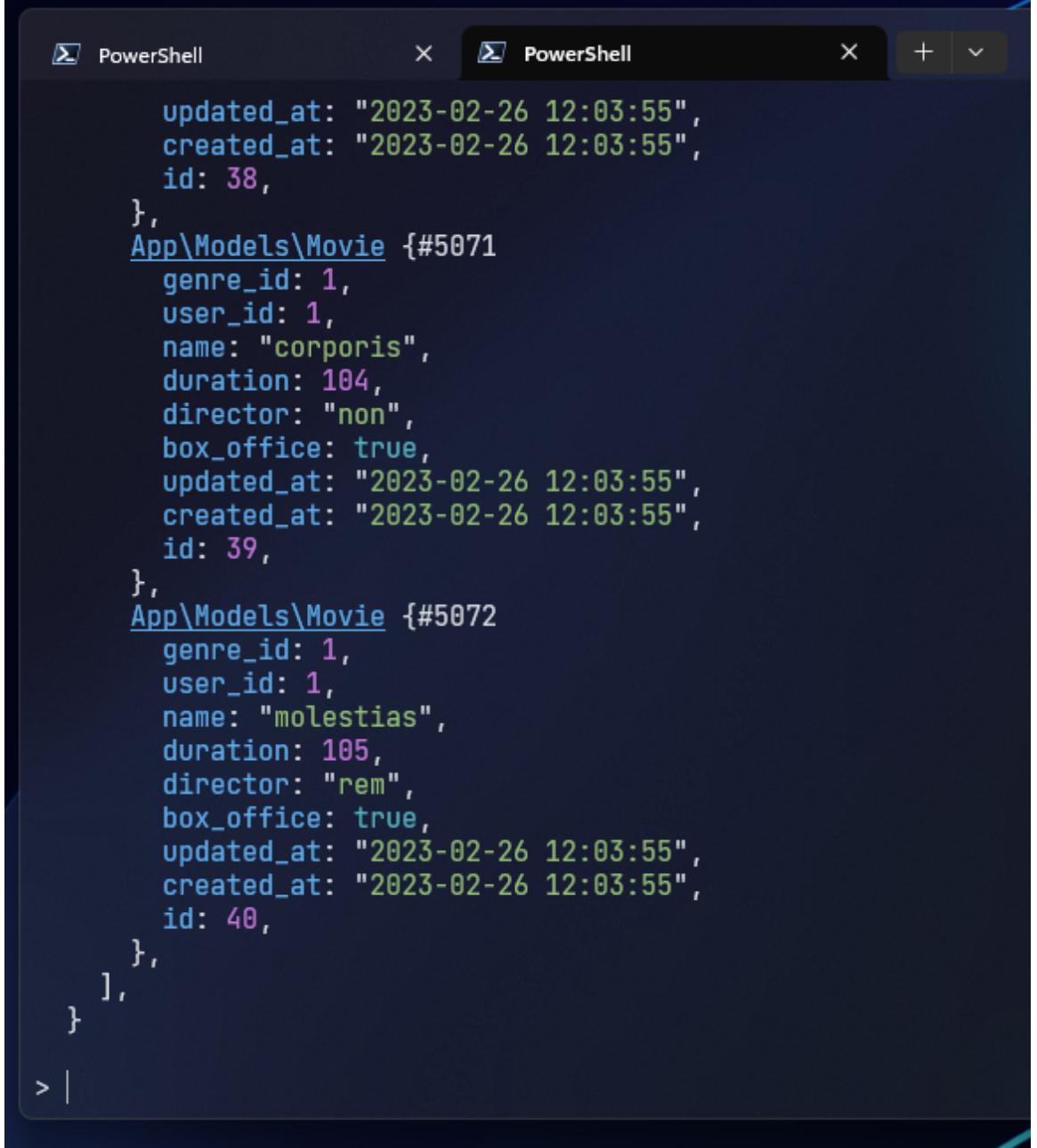
Como vemos a continuación se han generado correctamente.

The screenshot shows the phpMyAdmin interface for the 'practiclaravel' database. The left sidebar lists databases and tables, including 'Nueva', 'bank', 'information\_schema', 'mysql', 'performance\_schema', 'phpmyadmin', 'practiclaravel' (selected), 'test', and 'users'. The 'genres' table is selected under 'practiclaravel'. The main area displays the 'genres' table with 15 rows of data:

	id	user_id	name	created_at	updated_at
<input type="checkbox"/>	1	1	totam	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	2	1	velit	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	3	1	fugit	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	4	1	perferendis	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	5	1	sapiente	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	6	1	sed	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	7	1	ipsum	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	8	1	magnam	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	9	1	dolorem	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	10	1	atque	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	11	1	enim	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	12	1	architecto	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	13	1	voluptatem	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	14	1	persiciatis	2023-02-26 12:02:33	2023-02-26 12:02:33
<input type="checkbox"/>	15	1	vitae	2023-02-26 12:02:33	2023-02-26 12:02:33

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]);
```

A screenshot of a Windows taskbar showing two open PowerShell windows. Both windows have the title 'PowerShell'. The left window contains the command 'Get-Movie' followed by a long list of movie objects. The right window contains the command 'Get-Movie -id 1' followed by a single movie object. The movie objects are represented as PowerShell PSCustomObject instances with properties like 'name', 'duration', 'director', etc., displayed in green and blue text.

```
Get-Movie
{
    updated_at: "2023-02-26 12:03:55",
    created_at: "2023-02-26 12:03:55",
    id: 38,
},
App\Models\Movie {#5071
    genre_id: 1,
    user_id: 1,
    name: "corporis",
    duration: 104,
    director: "non",
    box_office: true,
    updated_at: "2023-02-26 12:03:55",
    created_at: "2023-02-26 12:03:55",
    id: 39,
},
App\Models\Movie {#5072
    genre_id: 1,
    user_id: 1,
    name: "molestias",
    duration: 105,
    director: "rem",
    box_office: true,
    updated_at: "2023-02-26 12:03:55",
    created_at: "2023-02-26 12:03:55",
    id: 40,
},
],
}
> |
```

Como vemos se ha creado correctamente.

The screenshot shows the phpMyAdmin interface for the 'practicalaravel' database. The 'movies' table is selected. The table has the following data:

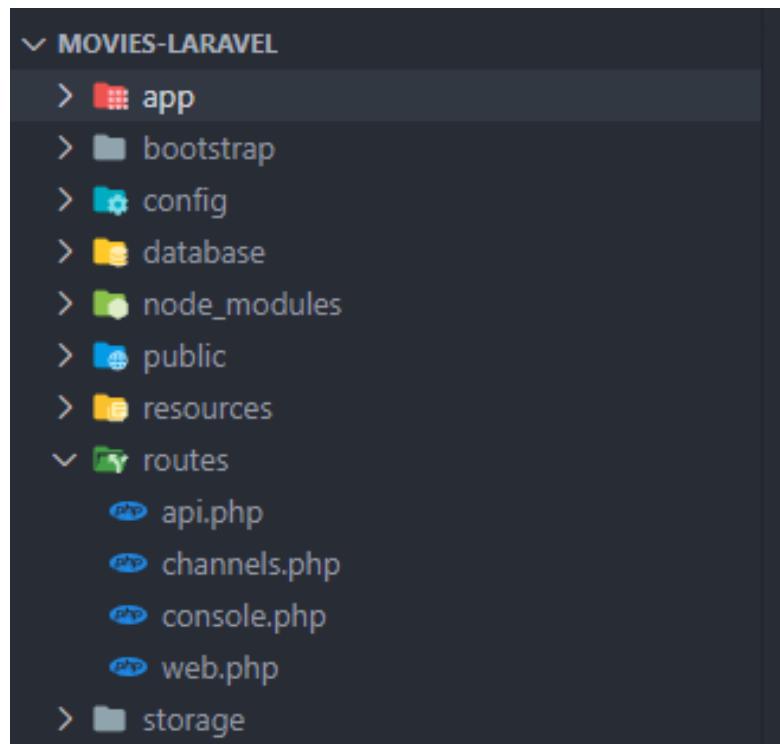
	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	<th>id</th>	id	<th>genre_id</th>	genre_id	<th>user_id</th>	user_id	<th>name</th>	name	<th>duration</th>	duration	<th>director</th>	director	<th>box_office</th>	box_office	<th>created_at</th>	created_at
1	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	1	1	1	et	91	voluptatem	1	2023-02-26 12:03:55								
2	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	2	1	1	nihil	96	sit	0	2023-02-26 12:03:55								
3	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	3	1	1	voluptates	118	iusto	1	2023-02-26 12:03:55								
4	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	4	1	1	assumenda	113	atque	1	2023-02-26 12:03:55								
5	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	5	1	1	saepe	111	officiis	1	2023-02-26 12:03:55								
6	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	6	1	1	sint	116	id	1	2023-02-26 12:03:55								
7	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	7	1	1	temporibus	114	sed	0	2023-02-26 12:03:55								
8	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	8	1	1	qui	110	rerum	0	2023-02-26 12:03:55								
9	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	9	1	1	molestiae	104	reprehenderit	1	2023-02-26 12:03:55								
10	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	10	1	1	rerum	105	sunt	1	2023-02-26 12:03:55								
11	<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	11	1	1	voluptatem	113	est	0	2023-02-26 12:03:55								

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.



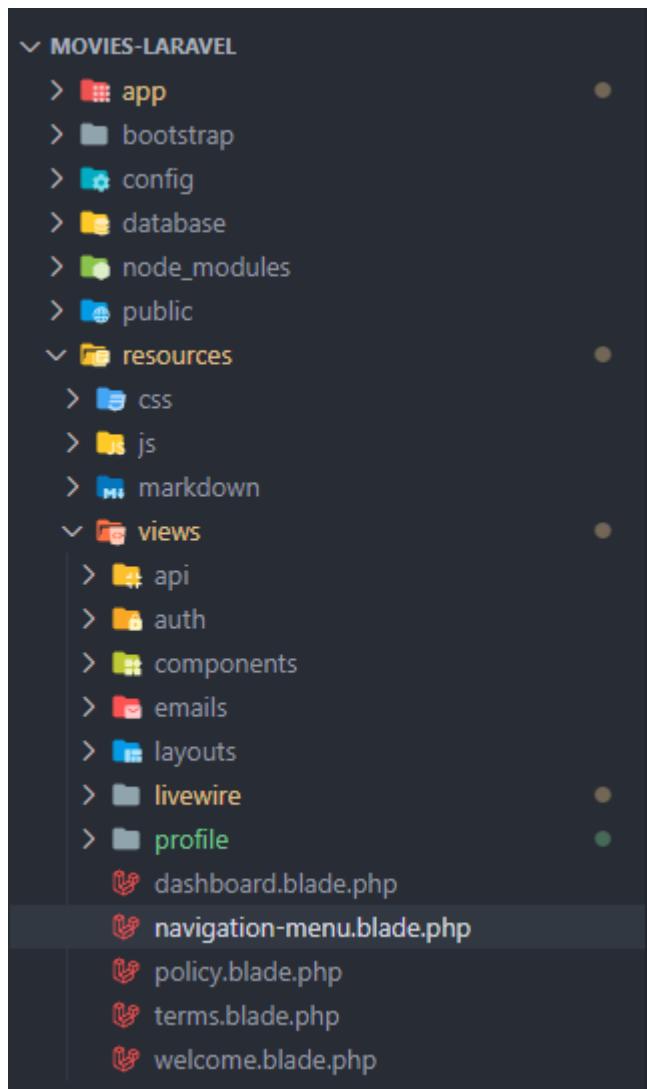
- 11.2. Y a continuación en el archivo web.php añadiremos las siguientes rutas. Como se ve devolveremos las vistas dentro de una carpeta que más adelante crearemos que se llamará content

```
web.php M routes\web.php\...
 9  /
10 / Here is where you can register web routes for your application. These
11 / routes are loaded by the RouteServiceProvider and all of them will
12 / be assigned to the "web" middleware group. Make something great!
13 /
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
20 Route::middleware(['auth:sanctum',config('jetstream.auth_session'),'verified'])->group(function () {
21     Route::get('/dashboard', function () {
22         return view('dashboard');
23     })->name('dashboard');
24 });
25
26 Route::middleware(['auth:sanctum',config('jetstream.auth_session'),'verified'])->group(function () {
27     Route::get('/genres', function () {
28         return view('content.genres');
29     })->name('genres');
30 });
31
32 Route::middleware(['auth:sanctum',config('jetstream.auth_session'),'verified'])->group(function () {
33     Route::get('/movies', function () {
34         return view('content.movies');
35     })->name('movies');
36 });
37
```

1 27 Col 1 Spaces:4 UFT-8 LF PHP @ Go Live 8

## 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>

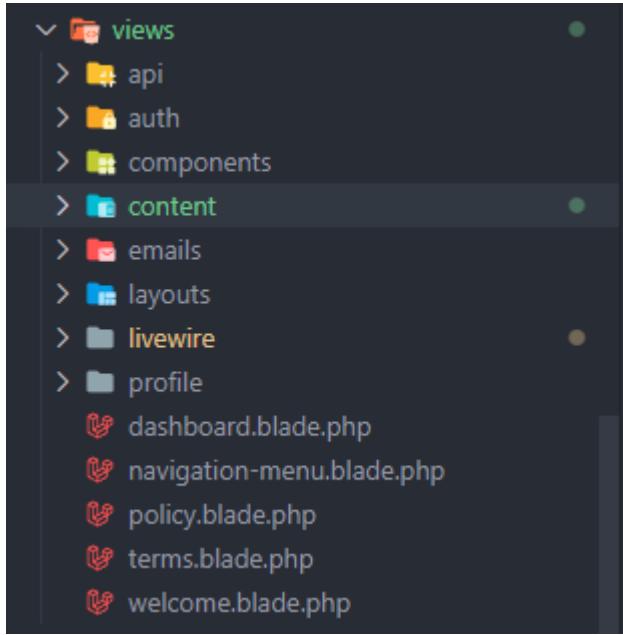
        <x-responsive-nav-link href="{{ route('genres') }}" :active="request()>routeIs('genres')">
            {{__('Genres') }}
        </x-responsive-nav-link>

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

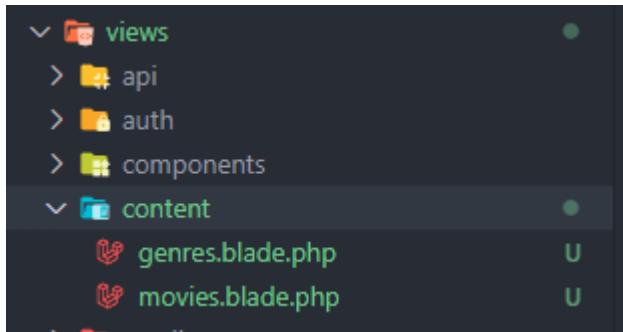
```

### 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...
1 <x-app-layout>
2   <x-slot name="header">
3     <h2 class="font-semibold text-xl text-gray-800 leading-tight">
4       {{ __('Genres') }}
5     </h2>
6   </x-slot>
7
8   <div class="py-12">
9     <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10       <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11         <livewire:genres />
12       </div>
13     </div>
14   </div>
15 </x-app-layout>
16
```

### 13.4. Ahora a continuación procedemos con películas

```
movies.blade.php U resources\views\content\movies.blade.php...
1 <x-app-layout>
2   <x-slot name="header">
3     <h2 class="font-semibold text-xl text-gray-800 leading-tight">
4       {{ __('Movies') }}
5     </h2>
6   </x-slot>
7
8   <div class="py-12">
9     <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10       <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11         <livewire:movies />
12       </div>
13     </div>
14   </div>
15 </x-app-layout>
16
```

## 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
```

```
Movies-Laravel on  main  ?2 ~2 via   18.8.0
→ php artisan make:livewire genres
COMPONENT CREATED 🌟

CLASS: app/Http/Livewire//Genres.php
VIEW: C:\Users\ID\git\Movies-Laravel\resources\views/livewire/genres.blade.php

Congratulations, you've created your first Livewire component! 🎉🎉🎉

Would you like to show some love by starring the repo? (yes/no) [no]:
> no

Movies-Laravel on  main  ?4 ~2 via   18.8.0
→
```

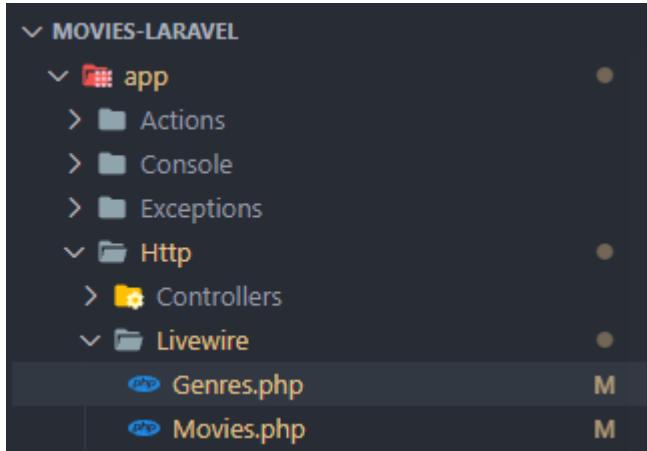
```
php artisan make:livewire movies
```

```
Movies-Laravel on  main  ?4 ~2 via   18.8.0
→ php artisan make:livewire movies
COMPONENT CREATED 🌟

CLASS: app/Http/Livewire//Movies.php
VIEW: C:\Users\ID\git\Movies-Laravel\resources\views/livewire/movies.blade.php

Movies-Laravel on  main  ?4 ~2 via   18.8.0
→ |
```

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



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

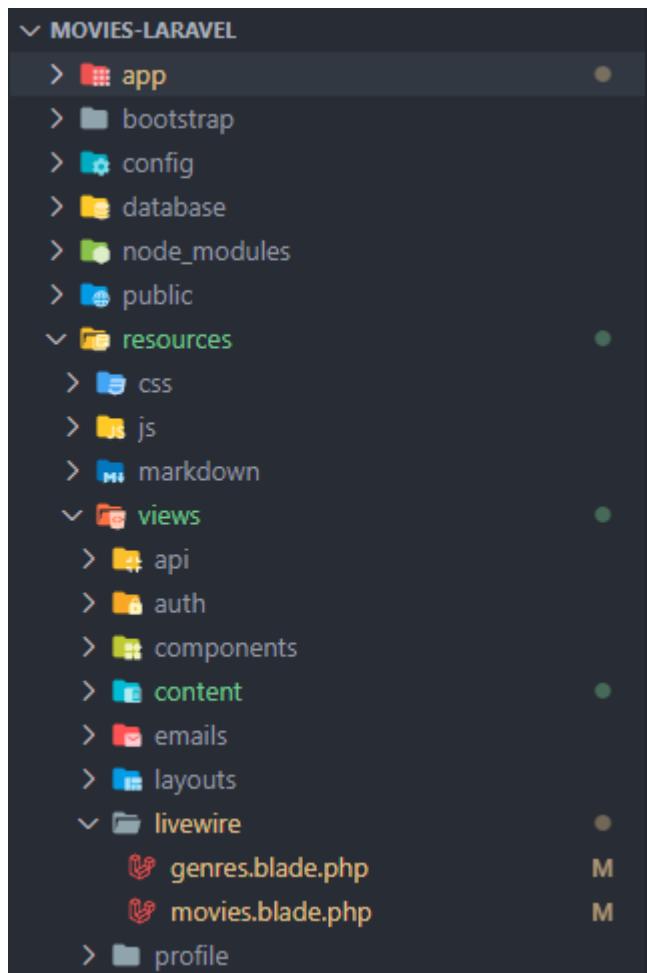
```
1 <?php
2
3 namespace App\Http\Livewire;
4
5 use Livewire\Component;
6 use App\Models\Genre;
7 use Livewire\WithPagination;
8
9 class Genres extends Component
10 {
11     use WithPagination;
12
13     public function render()
14     {
15         $genres = Genre::where('user_id', auth()->user()->id)->paginate(10);
16
17         return view('livewire.genres', [
18             'genres' => $genres,
19         ]);
20     }
}
```

14.4. Seguimos con el componente de películas:

```
⑥ Movies.php M app\Http\Livewire\Movies.php...
  2
  3 namespace App\Http\Livewire;
  4
  5 use Livewire\Component;
  6 use App\Models\Movie;
  7 use App\Models\Genre;
  8 use Livewire\WithPagination;
  9
 10
 11 class Movies extends Component
 12 {
 13
 14     public function render()
 15     {
 16         $movies = Movie::where('user_id', auth()->user()->id)->paginate(10);
 17
 18         $genres = Genre::where('user_id', auth()->user()->id);
 19
 20         return view('livewire.movies', [
 21             'movies' => $movies,
 22             'genres' => $genres
 23         ]);
 24     }
}
```

## 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\...
1   <div class="p-2 lg:p-8 bg-white border-b border-gray-200">
2     <x-application-logo class="block h-12 w-auto" />
3
4     <div class="mt-4 text-2xl font-medium text-gray-900">
5       <div>Genres</div>
6     </div>
7     <div class="mt-3">
8       <table class="table-auto w-full">
9         <thead>
10        <tr>
11          <th class="px-4 py-2">
12            <div class="flex items-center">Id</div>
13          </th>
14          <th class="px-4 py-2">
15            <div class="flex items-center">Name</div>
16          </th>
17        </tr>
18      </thead>
19      <tbody>
20        @foreach ($genres as $genre)
21          <tr>
22            <td class="rounded border px-4 py-2">{{$genre->id}}</td>
23            <td class="rounded border px-4 py-2">{{$genre->name}}</td>
24          </tr>
25        @endforeach
26      </tbody>
27    </table>
28  </div>
29  <div class="mt-4">{{$genres}}</div>
30 </div>
31 |

```

15.3. Y en el caso de películas pondremos el siguiente

```

<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.

The screenshot shows a web browser window titled "Laravel" with the URL "127.0.0.1:8000/genres". The page is part of a Laravel Jetstream application, featuring a sidebar with "Dashboard", "Genres", and "Movies" links, and a user profile for "Gabriel". The main content area is titled "Genres" and displays a table of genre data. The table has two columns: "Id" and "Name". The data is as follows:

Id	Name
1	totam
2	velit
3	fugit
4	perferendis
5	sapiente
6	sed
7	ipsum
8	magnam
9	dolorem
10	atque

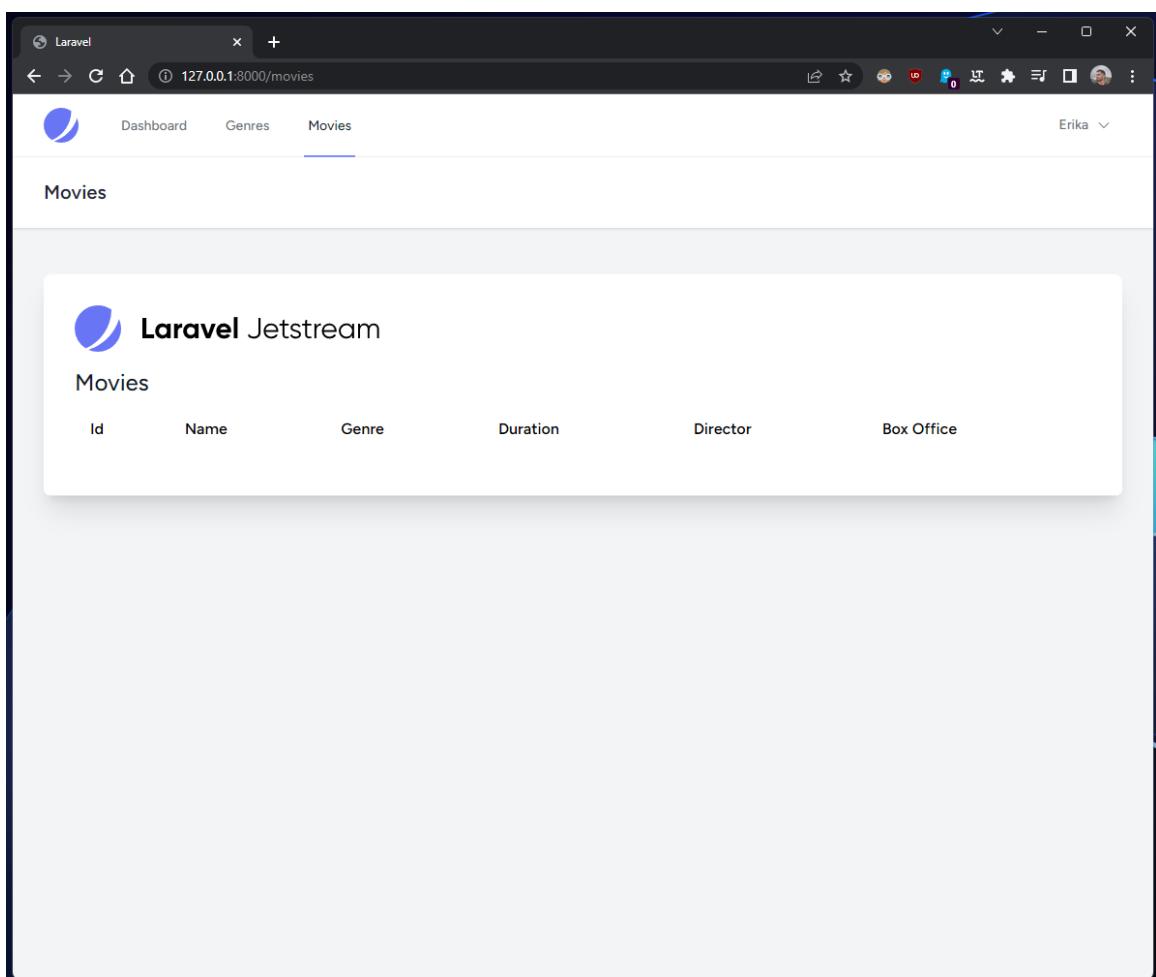
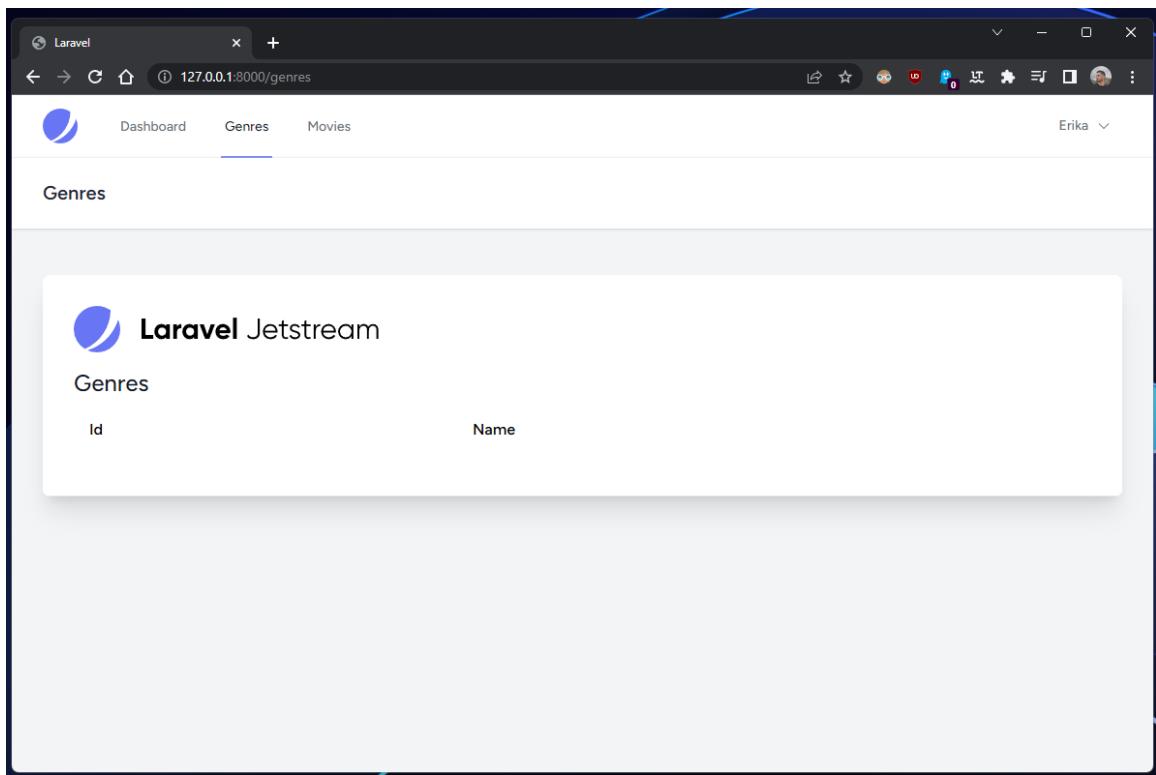
Below the table, a message says "Showing 1 to 10 of 15 results" and there are navigation buttons for pages 1, 2, and 3.

The screenshot shows a web browser window titled "Laravel" with the URL "127.0.0.1:8000/movies". The page is part of a Laravel Jetstream application, featuring a header with a user profile for "Gabriel". The main content area has a title "Movies" and displays a table of movie data. The table has columns: Id, Name, Genre, Duration, Director, and Box Office. The data is as follows:

Id	Name	Genre	Duration	Director	Box Office
1	et	totam	91	voluptatem	Yes
2	nihil	totam	96	sit	No
3	voluptates	totam	118	iusto	Yes
4	assumenda	totam	113	atque	Yes
5	saepe	totam	111	officiis	Yes
6	sint	totam	116	id	Yes
7	temporibus	totam	114	sed	No
8	qui	totam	110	rerum	No
9	molestiae	totam	104	reprehenderit	Yes
10	rerum	totam	105	sunt	Yes

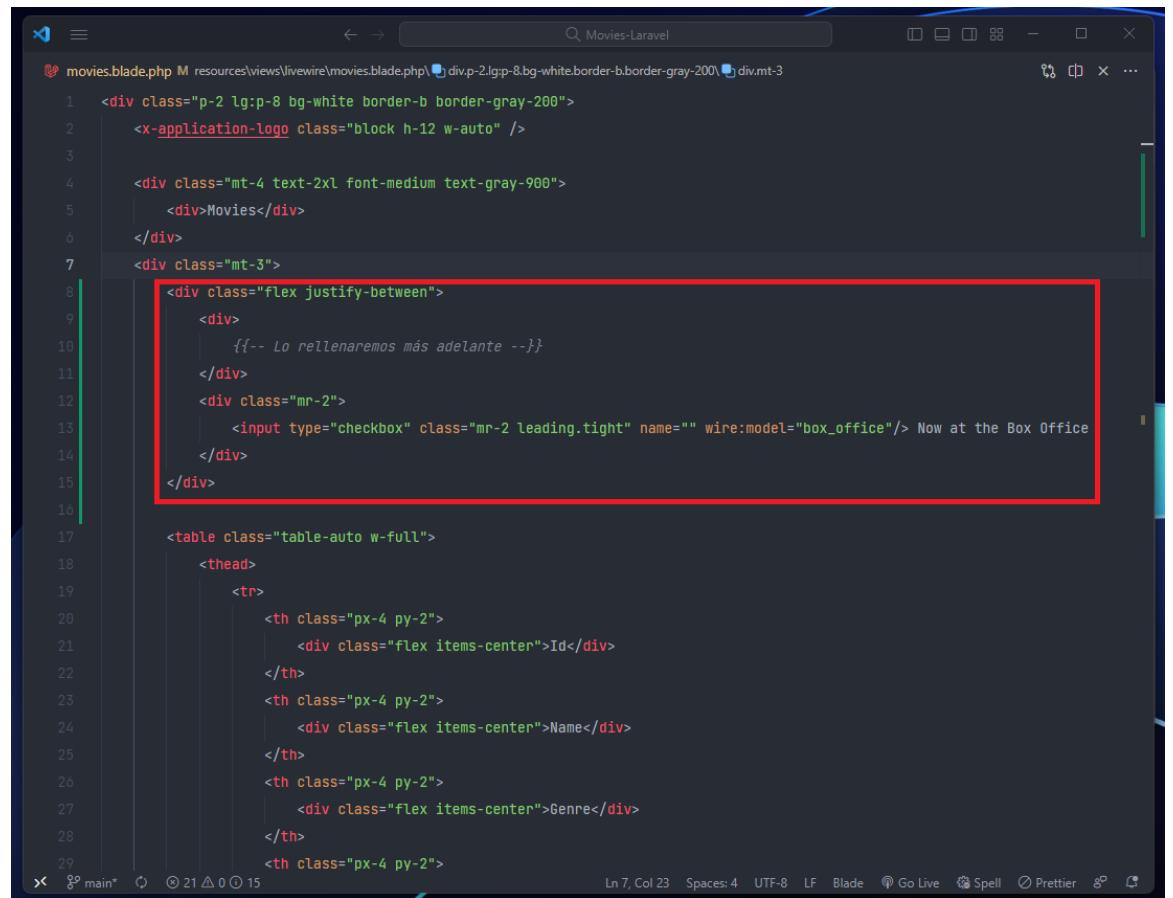
Below the table, a message says "Showing 1 to 10 of 40 results" and there is a navigation bar with links 1, 2, 3, 4, >.

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.



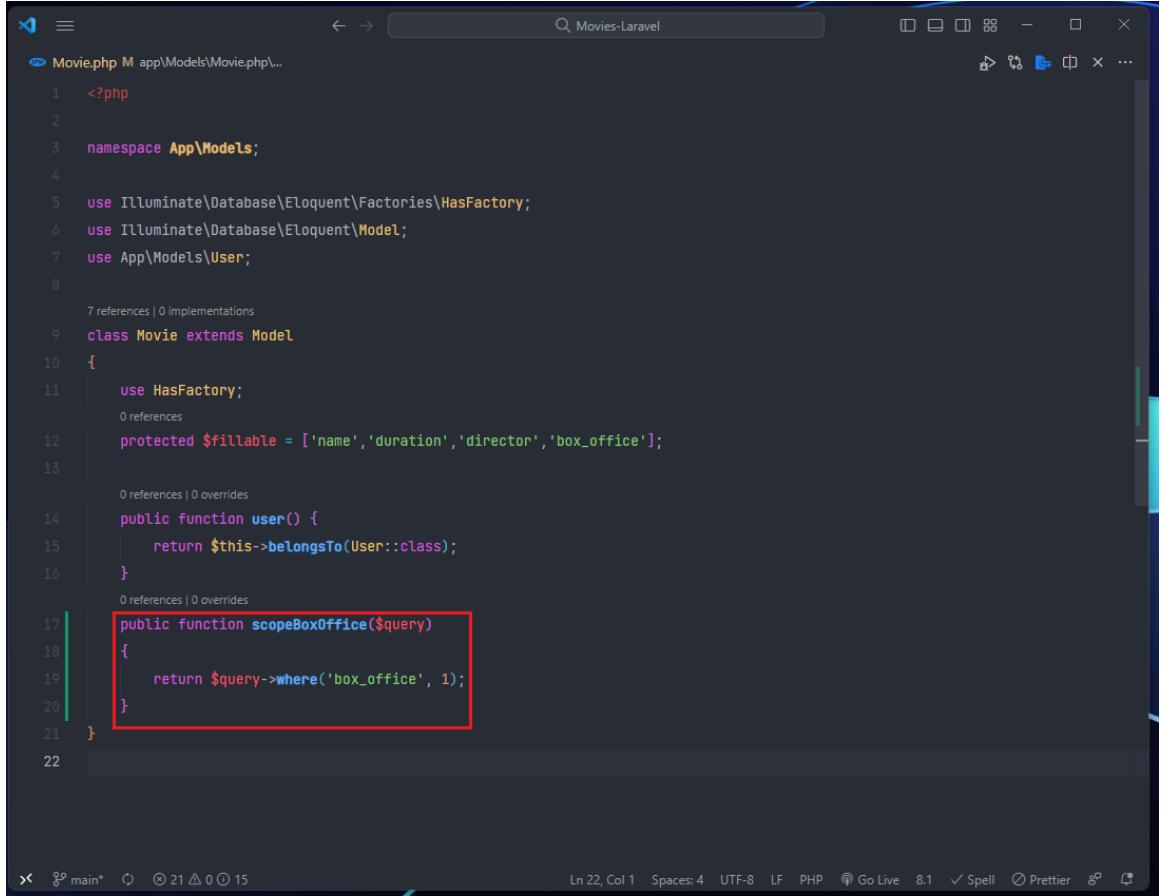
## 16. Añadimos Filtro de películas en taquilla

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.



```
1 <div class="p-2 lg:p-8 bg-white border-b border-gray-200">
2   <x-application-logo class="block h-12 w-auto" />
3
4   <div class="mt-4 text-2xl font-medium text-gray-900">
5     <div>Movies</div>
6   </div>
7   <div class="mt-3">
8     <div class="flex justify-between">
9       <div>
10      {{-- Lo rellenaremos más adelante --}}
11    </div>
12    <div class="mr-2">
13      <input type="checkbox" class="mr-2 leading.tight" name="" wire:model="box_office"/> Now at the Box Office
14    </div>
15  </div>
16
17  <table class="table-auto w-full">
18    <thead>
19      <tr>
20        <th class="px-4 py-2">
21          <div class="flex items-center">Id</div>
22        </th>
23        <th class="px-4 py-2">
24          <div class="flex items-center">Name</div>
25        </th>
26        <th class="px-4 py-2">
27          <div class="flex items-center">Genre</div>
28        </th>
29        <th class="px-4 py-2">
```

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



The screenshot shows a code editor window with the title "Movies-Laravel". The file being edited is "Movie.php" located at "app\Models\Movie.php". The code is a PHP class definition for "Movie" extending "Model". It includes various imports like "HasFactory" and "User". A method "scopeBoxOffice" is highlighted with a red box, which is defined as returning a query where the "box\_office" field is 1.

```
<?php  
namespace App\\Models;  
  
use Illuminate\\Database\\Eloquent\\Factories\\HasFactory;  
use Illuminate\\Database\\Eloquent\\Model;  
use App\\Models\\User;  
  
class Movie extends Model  
{  
    use HasFactory;  
    protected $fillable = ['name', 'duration', 'director', 'box_office'];  
  
    public function user()  
    {  
        return $this->belongsTo(User::class);  
    }  
  
    public function scopeBoxOffice($query)  
    {  
        return $query->where('box_office', 1);  
    }  
}  
In 22, Col 1 Spaces: 4 UTF-8 LF PHP Go Live 8.1 ✓ Spell ⚡ Prettier ⚡
```

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

```
10  class Movies extends Component
11  {
12      use WithPagination;
13
14      public $box_office;
15
16      public function render()
17      {
18          $movies = Movie::where('user_id', auth()->user()->id)
19              ->when($this->box_office, function ($query) {
20                  return $query->boxOffice();
21              });
22
23          $movies = $movies->paginate(10);
24
25          $genres = Genre::where('user_id', auth()->user()->id);
26
27          return view('livewire.movies', [
28              'movies' => $movies,
29              'genres' => $genres
30          ]);
31
32      }
33
34      public function updatingBoxOffice()
35      {
36          $this->resetPage();
37      }
38
39  }
```

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

The screenshot shows a web browser window titled "Laravel" with the URL "127.0.0.1:8000/movies". The page is part of a Laravel Jetstream application, featuring a header with a logo, "Dashboard", "Genres", and "Movies" links, and a user profile for "Gabriel". The main content area is titled "Movies" and displays a table of movie data. The table has columns: Id, Name, Genre, Duration, Director, and Box Office. A checkbox labeled "Now at the Box Office" is checked. Below the table, it says "Showing 1 to 10 of 24 results" and there is a pagination navigation bar with buttons for <, 1, 2, 3, >.

Id	Name	Genre	Duration	Director	Box Office
1	et	totam	91	voluptatem	Yes
3	voluptates	totam	118	iusto	Yes
4	assumenda	totam	113	atque	Yes
5	saepe	totam	111	officiis	Yes
6	sint	totam	116	id	Yes
9	molestiae	totam	104	reprehenderit	Yes
10	rerum	totam	105	sunt	Yes
13	quia	totam	106	sunt	Yes
16	ut	totam	97	explicabo	Yes
17	voluptas	totam	103	numquam	Yes

Como vemos funciona correctamente. Además de la forma en la que lo hemos hecho si por ejemplo tenemos 4 páginas y estamos en 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>
</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.

```

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 lo mismo pero con el componente de películas. Nos dirigimos a (app\Http\Livewire\Movies.php) y agregamos lo siguiente.

```

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);

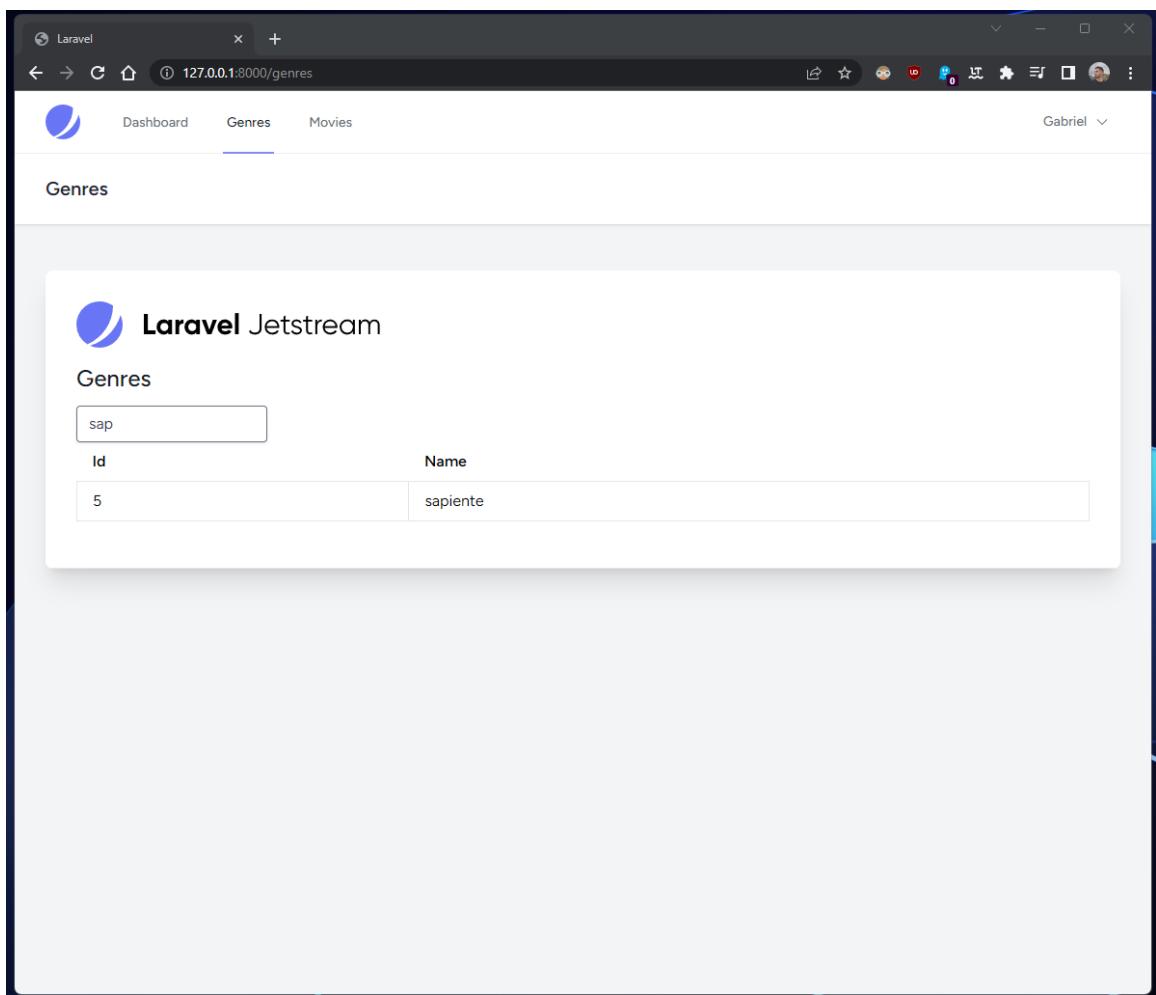
```

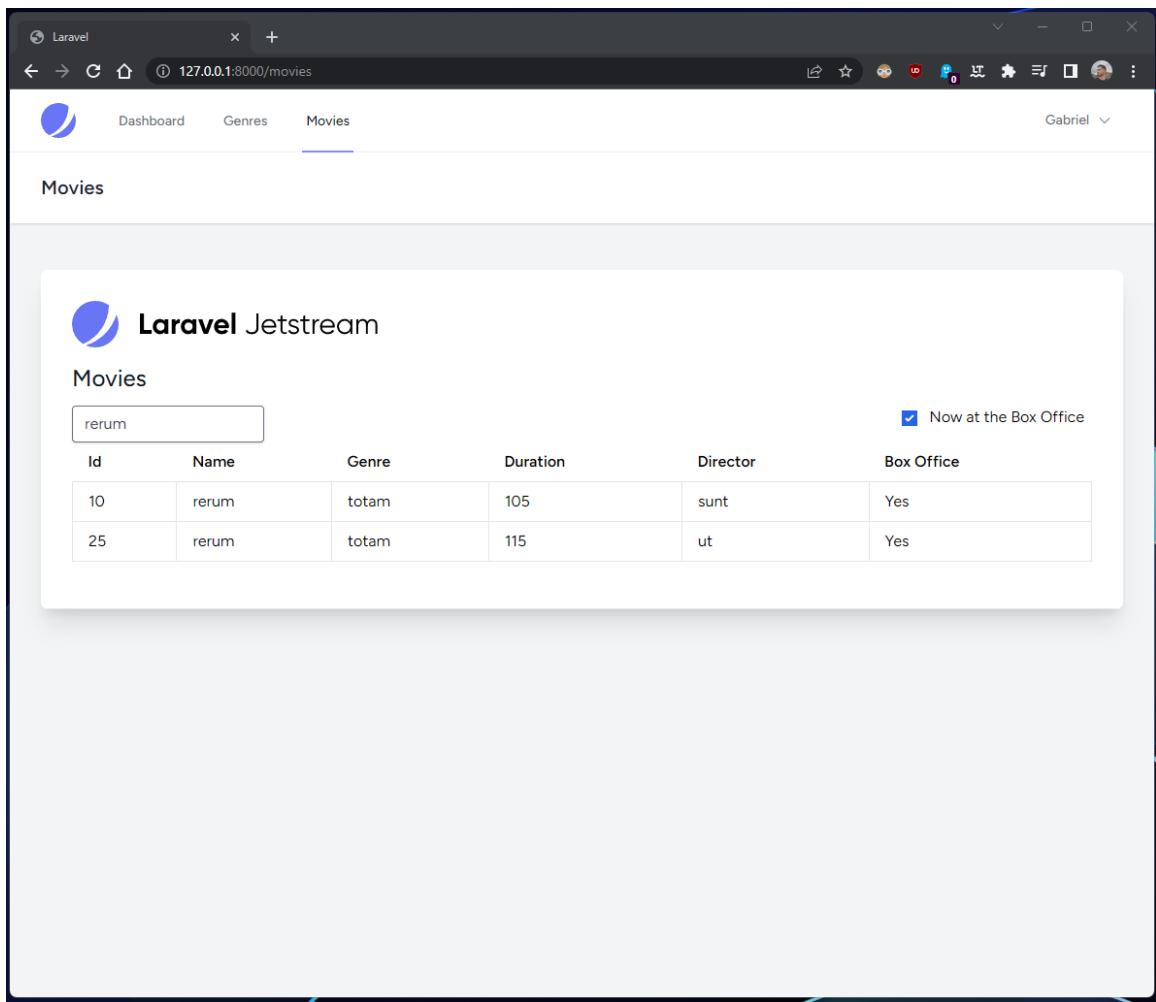
```
        return view('livewire.movies', [
            'movies' => $movies,
            'genres' => $genres
        ]);
    }

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.

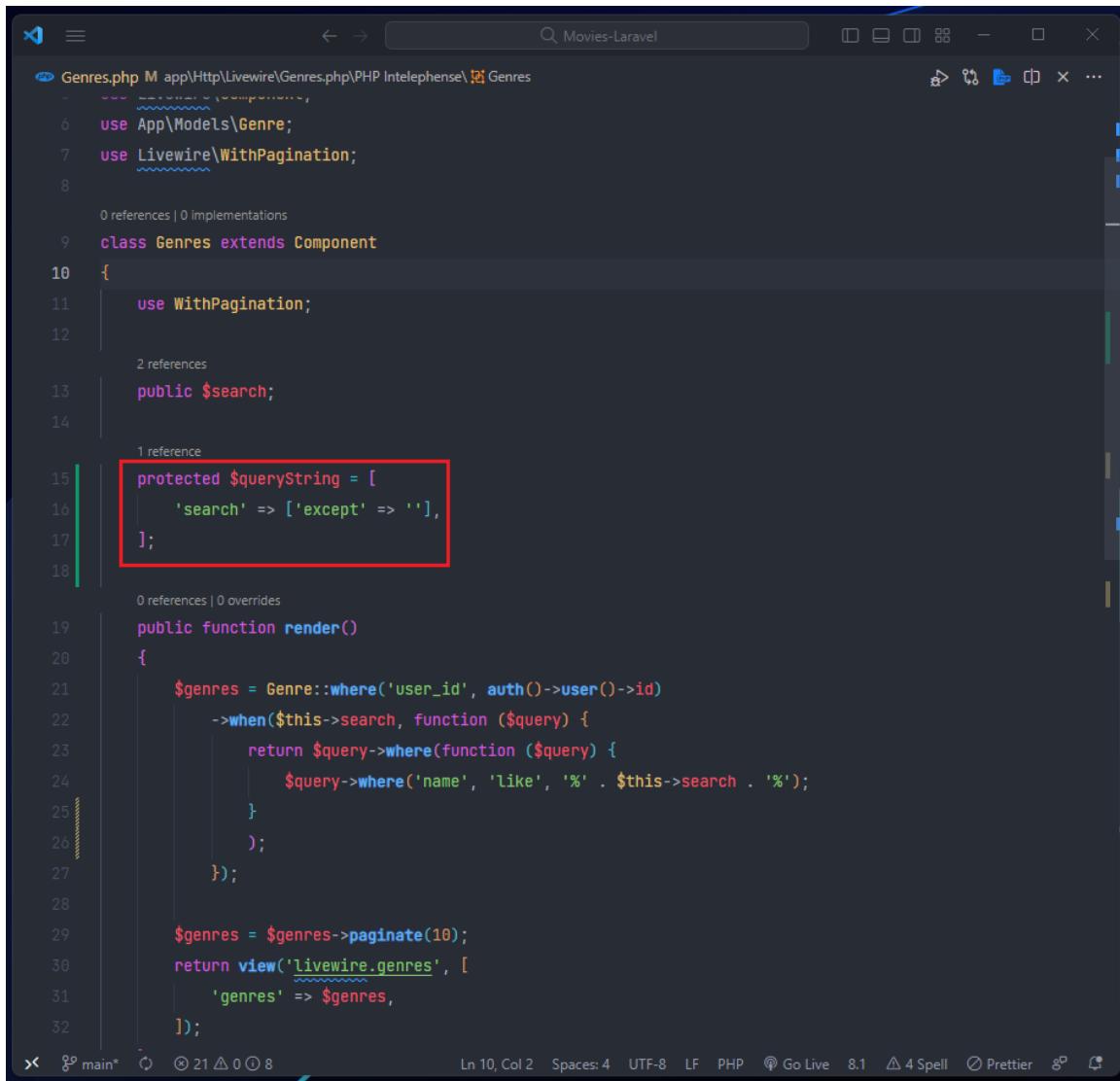




## 18. Query String

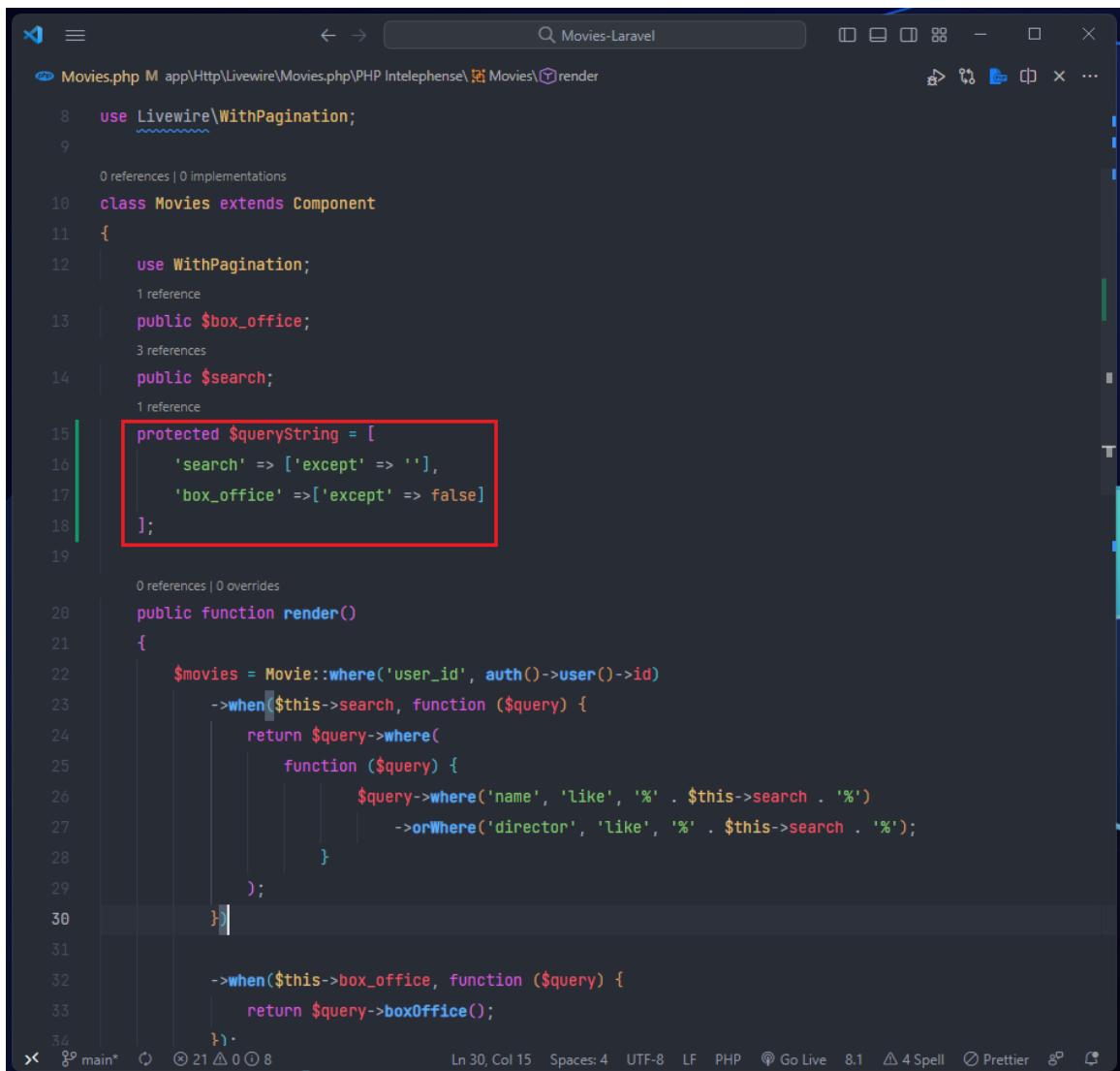
A continuación vamos a mejorar las url de nuestra página web para que al hacer búsquedas o aplicar filtros estas se muestren en la url de nuestra página. Pa eso haremos los siguientes pasos

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



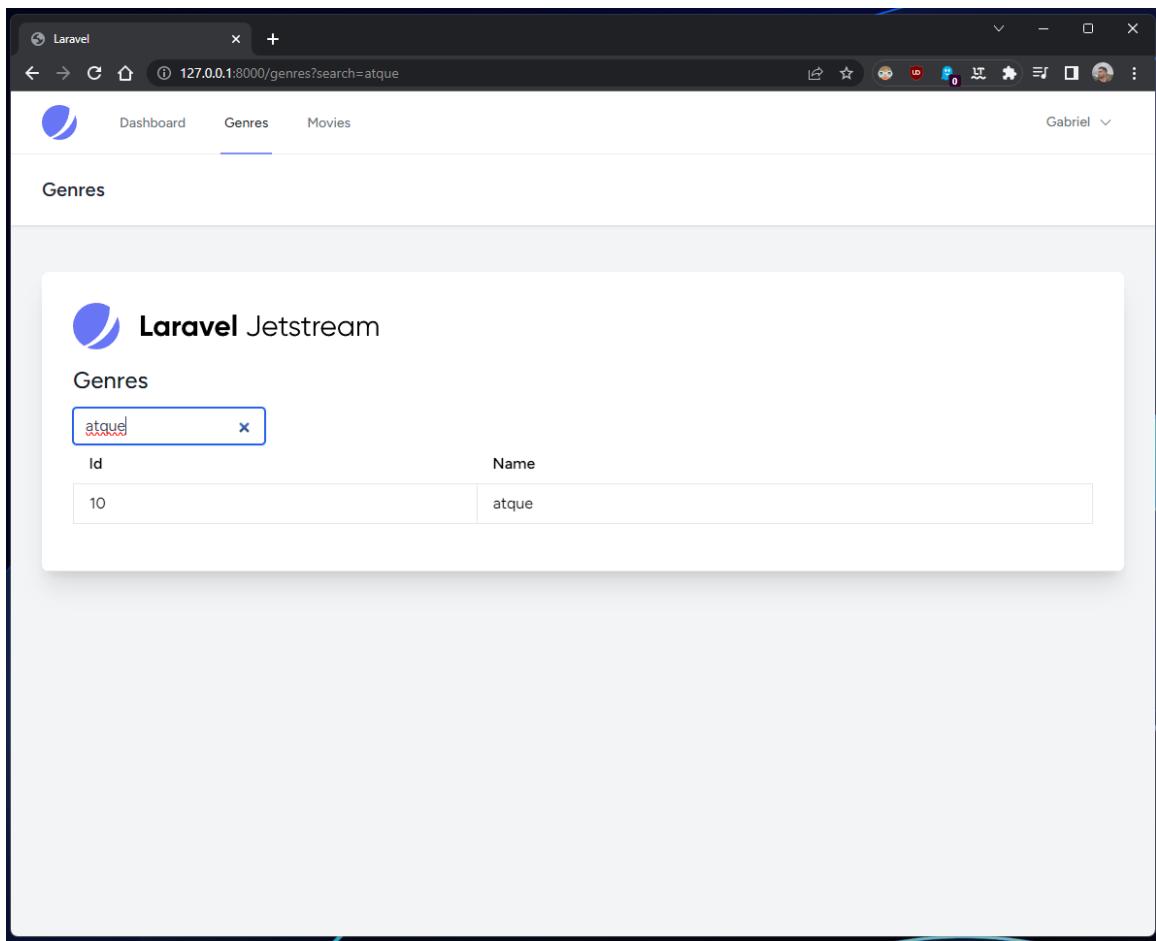
```
Genres.php M app\Http\Livewire\Genres.php|PHP Intelephense\Genres
1
2
3
4
5
6 use App\Models\Genre;
7 use Livewire\WithPagination;
8
9 class Genres extends Component
10 {
11     use WithPagination;
12
13     public $search;
14
15     protected $queryString = [
16         'search' => ['except' => ''],
17     ];
18
19     public function render()
20     {
21         $genres = Genre::where('user_id', auth()->user()->id)
22             ->when($this->search, function ($query) {
23                 return $query->where(function ($query) {
24                     $query->where('name', 'like', '%' . $this->search . '%');
25                 });
26             });
27
28
29         $genres = $genres->paginate(10);
30         return view('livewire.genres', [
31             'genres' => $genres,
32         ]);
33     }
34 }
```

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



```
use Livewire\WithPagination;
class Movies extends Component
{
    use WithPagination;
    public $box_office;
    public $search;
    protected $queryString = [
        'search' => ['except' => ''],
        'box_office' =>['except' => false]
    ];
}
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();
        });
}
```

- 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



The screenshot shows a browser window titled "Laravel" with the URL "127.0.0.1:8000/movies?search=saepe&box\_office=true". The page is part of a Laravel Jetstream application, featuring a header with a logo, "Dashboard", "Genres", and "Movies" (which is underlined, indicating it's the active page). A user profile "Gabriel" is visible in the top right. The main content area has a title "Movies" and displays a search result for "saepe". A checkbox labeled "Now at the Box Office" is checked. The results table has columns: Id, Name, Genre, Duration, Director, and Box Office. One row is shown with values: Id=5, Name=saepe, Genre=totam, Duration=111, Director=officiis, and Box Office=Yes.

## 19. Mejorando filtro en taquilla

Cuándo filtramos 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

The screenshot shows a code editor with a dark theme displaying a Blade template file named `movies.blade.php`. The file contains PHP-like syntax for generating an HTML table. The code includes several conditional blocks (`@if`) and loops (`@foreach`). Two specific sections of the code are highlighted with red boxes:

```
</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>
@if (!$box_office)
    <th class="px-4 py-2">
        <div class="flex items-center">Box Office</div>
    </th>
@endif
</tr>
</thead>
<tbody>
    @foreach ($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
        </tr>
    @endforeach

```

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

The screenshot shows a web browser window titled "Laravel" with the URL "127.0.0.1:8000/movies?box\_office=true". The page is titled "Movies" and features the Laravel Jetstream logo. It includes a search bar and a checkbox labeled "Now at the Box Office" which is checked. A table displays 10 movie entries from a database, with 24 results shown in total. The table columns are "Id", "Name", "Genre", "Duration", and "Director". The data is as follows:

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

At the bottom, it says "Showing 1 to 10 of 24 results" and has navigation buttons for pages 1, 2, 3, and >.

Movies					
	Name	Genre	Duration	Director	Box Office
1	et	totam	91	voluptatem	Yes
2	nihil	totam	96	sit	No
3	voluptates	totam	118	iusto	Yes
4	assumenda	totam	113	atque	Yes
5	saepe	totam	111	officiis	Yes
6	sint	totam	116	id	Yes
7	temporibus	totam	114	sed	No
8	qui	totam	110	rerum	No
9	molestiae	totam	104	reprehenderit	Yes
10	rerum	totam	105	sunt	Yes

Showing 1 to 10 of 40 results

< 1 2 3 4 >

## 20. Ordenación ascendente y descendente

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.

```
13     public $search;
14     public $sortBy = 'id';
15     public $sortAsc = true;
16
17     protected $queryString = [
18         'search' => ['except' => ''],
19         'sortBy' => ['except' => 'id'],
20         'sortAsc' => ['except' => true]
21     ];
22
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     }

```

```
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

The screenshot shows a code editor window with the following code:

```
15 |     public $sortBy = 'id';
16 |     public $sortAsc = true;
17 | 
18 |     protected $queryString = [
19 |         'search' => ['except' => ''],
20 |         'box_office' => ['except' => false],
21 |         'sortBy' => ['except' => 'id'],
22 |         'sortAsc' => ['except' => true]
23 |     ];
24 | 
25 |     0 references | 0 overrides
26 |     public function render()
27 |     {
28 |         $movies = Movie::where('user_id', auth()->user()->id)
29 |             ->when($this->search, function ($query) {
30 |                 return $query->where(
31 |                     function ($query) {
32 |                         $query->where('name', 'like', '%' . $this->search . '%')
33 |                             ->orWhere('director', 'like', '%' . $this->search . '%');
34 |                     }
35 |                 );
36 |             }
37 |             ->when($this->box_office, function ($query) {
38 |                 return $query->boxOffice();
39 |             }
40 |             ->orderBy($this->sortBy, $this->sortAsc ? 'ASC' : 'DESC');
41 | 
42 |         $movies = $movies->paginate(10);
43 |         $genres = Genre::where('user_id', auth()->user()->id);
```

The code is annotated with several red boxes highlighting specific sections of the code:

- A red box surrounds the properties `$sortBy` and `$sortAsc`.
- A red box surrounds the array in the `protected $queryString` property.
- A red box surrounds the `when` blocks for `search` and `box_office`.
- A red box surrounds the `orderBy` method call at line 40.

```

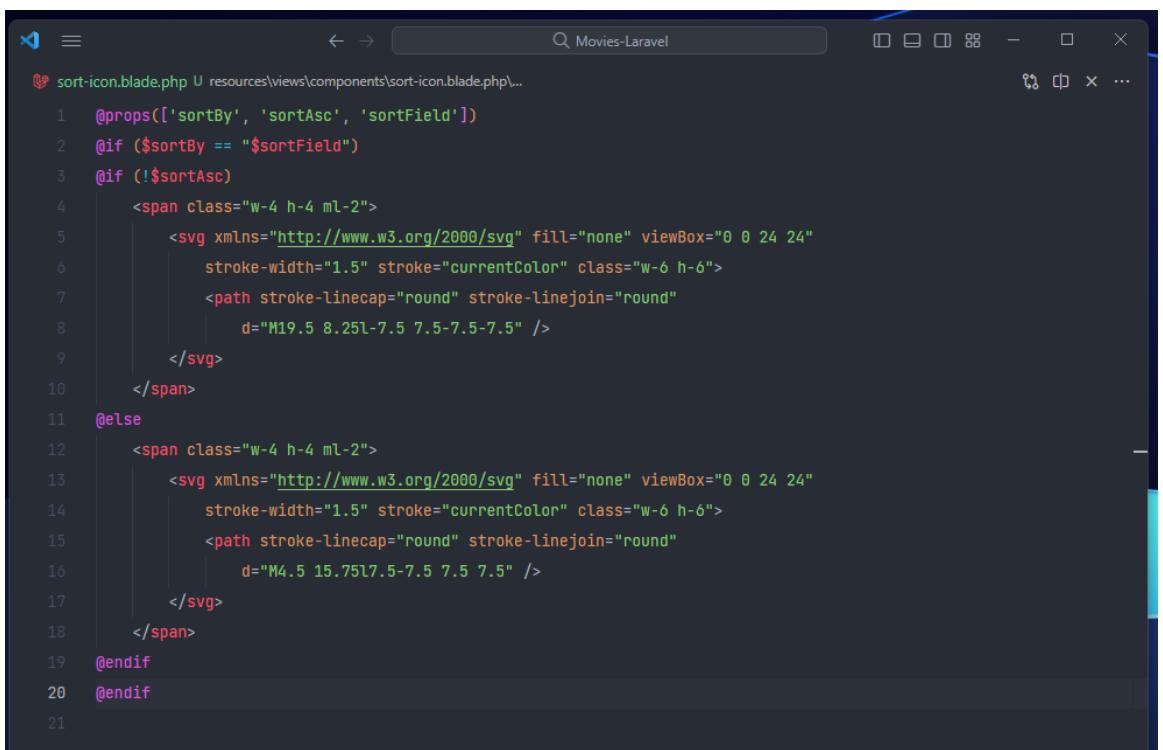
0 references | 0 overrides
public function updatingBoxOffice()
{
    $this->resetPage();
}

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 aadir 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.



```

sort-icon.blade.php U resources\views\components\sort-icon.blade.php...
1 @props(['sortBy', 'sortAsc', 'sortField'])
2 @if ($sortBy == "$sortField")
3 @if (!$sortAsc)
4     <span class="w-4 h-4 ml-2">
5         <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
6             stroke-width="1.5" stroke="currentColor" class="w-6 h-6">
7             <path stroke-linecap="round" stroke-linejoin="round"
8                 d="M19.5 8.25l-7.5 7.5-7.5-7.5" />
9         </svg>
10    </span>
11 @else
12    <span class="w-4 h-4 ml-2">
13        <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
14            stroke-width="1.5" stroke="currentColor" class="w-6 h-6">
15            <path stroke-linecap="round" stroke-linejoin="round"
16                d="M4.5 15.75l7.5-7.5 7.5 7.5" />
17        </svg>
18    </span>
19 @endif
20 @endif
21

```

- 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)

```
21 <thead>
22   <tr>
23     <th class="px-4 py-2">
24       <div class="flex items-center">
25         <button wire:click="sortBy('id')">Id</button>
26         <x-sort-icon sortField="id" :sortBy="$sortBy" :sortAsc="$sortAsc" />
27       </div>
28     </th>
29     <th class="px-4 py-2">
30       <div class="flex items-center">
31         <button wire:click="sortBy('name')">Name</button>
32         <x-sort-icon sortField="name" :sortBy="$sortBy" :sortAsc="$sortAsc" />
33       </div>
34     </th>
35     <th class="px-4 py-2">
36       <div class="flex items-center">
37         <button wire:click="sortBy('genre_id')">Genre</button>
38         <x-sort-icon sortField="genre_id" :sortBy="$sortBy" :sortAsc="$sortAsc" />
39       </div>
40     </th>
41     <th class="px-4 py-2">
42       <div class="flex items-center">
43         <button wire:click="sortBy('duration')">Duration</button>
44         <x-sort-icon sortField="duration" :sortBy="$sortBy" :sortAsc="$sortAsc" />
45       </div>
46     </th>
47     <th class="px-4 py-2">
48       <div class="flex items-center">
49         <button wire:click="sortBy('director')">Director</button>
50         <x-sort-icon sortField="director" :sortBy="$sortBy" :sortAsc="$sortAsc" />
51       </div>
52     </th>
```

- 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.

The screenshot shows a web browser window titled "Laravel" with the URL "127.0.0.1:8000/genres". The page is part of the Laravel Jetstream application, featuring a dark-themed header with navigation links for "Dashboard", "Genres", and "Movies", and a user profile for "Gabriel". The main content area is titled "Genres" and displays a table of genre names. A search bar is present at the top left of the table. The table has two columns: "Id" and "Name". The data is as follows:

Id	Name
1	totam
2	velit
3	fugit
4	perferendis
5	sapiente
6	sed
7	ipsum
8	magnam
9	dolorem
10	atque

Below the table, a message indicates "Showing 1 to 10 of 15 results" and a small navigation bar with arrows and page numbers (1, 2) is shown.

The screenshot shows a web browser window titled "Laravel" with the URL "127.0.0.1:8000/genres?sortAsc=false". The page is part of a Laravel Jetstream application, featuring a dark-themed header with navigation links for "Dashboard", "Genres", and "Movies", and a user profile for "Gabriel". The main content area is titled "Genres" and displays a table of genre data. The table has two columns: "Id" and "Name". The data is as follows:

Id	Name
15	vitae
14	perspiciatis
13	voluptatem
12	architecto
11	enim
10	atque
9	dolorem
8	magnam
7	ipsum
6	sed

Below the table, a message indicates "Showing 1 to 10 of 15 results" and there are navigation arrows for pagination.

The screenshot shows a web browser window titled "Laravel" with the URL "127.0.0.1:8000/movies?sortBy=name&sortAsc=false". The page is part of a Laravel Jetstream application, featuring a top navigation bar with links for "Dashboard", "Genres", and "Movies". A user profile "Gabriel" is visible on the right. The main content area is titled "Movies" and displays a table of movie data. The table has columns for "Id", "Name", "Genre", "Duration", "Director", and "Box Office". A search bar and a checkbox for "Now at the Box Office" are also present. The table data is as follows:

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

Below the table, it says "Showing 1 to 10 of 40 results" and there is a pagination control with buttons for <, 1, 2, 3, 4, >.

## 21. Eliminación de datos

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

```
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]
    ];
}
```

```
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).

```
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

```
20 |         <button wire:click="sortBy('id')">Id</button>
21 |         <x-sort-icon sortField="id" :sortBy="$sortBy" :sortAsc="$sortAsc" />
22 |     </div>
23 | </th>
24 | <th class="px-4 py-2">
25 |     <div class="flex items-center">
26 |         <button wire:click="sortBy('name')">Name</button>
27 |         <x-sort-icon sortField="name" :sortBy="$sortBy" :sortAsc="$sortAsc" />
28 |     </div>
29 | </th>
30 | <th class="px-4 py-2">Action</th>
31 | </tr>
32 | </thead>
33 | <tbody>
34 |     @foreach ($genres as $genre)
35 |     <tr>
36 |         <td class="rounded border px-4 py-2">{{ $genre->id }}</td>
37 |         <td class="rounded border px-4 py-2">{{ $genre->name }}</td>
38 |         <td class="rounded border px-4 py-2">
39 |             <x-danger-button wire:click="confirmGenreDeletion ({{ $genre->id }})">
40 |                 wire:loading.attr="disabled"
41 |                 {{ __('Remove') }}
42 |             </x-danger-button>
43 |         </td>
44 |     </tr>
45 |     @endforeach
46 | </tbody>
47 | </table>
```

```

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

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

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

```

## 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
  50
  51           <x-sort-icon sortField="director" :sortBy="$sortBy" :sortAsc="$sortAsc" />
  52       </div>
  53   </th>
  54   @if (!$box_office)
  55       <th class="px-4 py-2">
  56           <div class="flex items-center">Box Office</div>
  57       </th>
  58   @endif
  59   <th class="px-4 py-2">Action</th>
  60
  61 </tr>
  62 </thead>
  63 <tbody>
  64     @foreach ($movies as $movie)
  65         <tr>
  66             <td class="rounded border px-4 py-2">{{ $movie->id }}</td>
  67             <td class="rounded border px-4 py-2">{{ $movie->name }}</td>
  68             <td class="rounded border px-4 py-2">{{ $genres->find($movie->genre_id)->name }}</td>
  69             <td class="rounded border px-4 py-2">{{ $movie->duration }}</td>
  70             <td class="rounded border px-4 py-2">{{ $movie->director }}</td>
  71             @if (!$box_office)
  72                 <td class="rounded border px-4 py-2">{{ $movie->box_office ? 'Yes' : 'No' }}</td>
  73             @endif
  74             <td class="rounded border px-4 py-2">
  75                 <x-danger-button wire:click="confirmMovieDeletion ({{ $movie->id }})">
  76                     wire:loading.attr="disabled"
  77                     {{ __('Remove') }}
  78                 </x-danger-button>
  79             </td>
  80         </tr>
  81     @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.

The screenshot shows a web browser window with the title "Laravel" at the top. The address bar displays "127.0.0.1:8000/movies?page=4". The main content area features the "Laravel Jetstream" logo and the word "Movies". Below this is a search bar labeled "Search". A checkbox labeled "Now at the Box Office" is checked. A table lists 40 movie entries, each with columns for Id, Name, Genre, Duration, Director, Box Office status, and a "REMOVE" button. At the bottom, it says "Showing 31 to 40 of 40 results" and includes a navigation bar with links for <, 1, 2, 3, 4, and >.

Id	Name	Genre	Duration	Director	Box Office	Action
31	dolores	totam	102	vero	Yes	<button>REMOVE</button>
32	facere	totam	102	aut	Yes	<button>REMOVE</button>
33	dolores	totam	102	voluptas	No	<button>REMOVE</button>
34	eos	totam	115	velit	No	<button>REMOVE</button>
35	et	totam	99	animi	Yes	<button>REMOVE</button>
36	tempore	totam	91	ut	Yes	<button>REMOVE</button>
37	dolor	totam	113	odio	Yes	<button>REMOVE</button>
38	perferendis	totam	103	ullam	Yes	<button>REMOVE</button>
39	corporis	totam	104	non	Yes	<button>REMOVE</button>
40	molestias	totam	105	rem	Yes	<button>REMOVE</button>

The screenshot shows a Laravel application interface. At the top, there's a navigation bar with a logo, the word "Laravel", and a search bar labeled "Search". Below this is a table titled "Movies" with columns: Id, Name, Genre, Duration, Director, Box Office, and Action. The table contains 10 rows of movie data. A modal dialog box is overlaid on the page, centered over the 31st row. The modal has a title "Remove" and a message "Are you sure you want to delete this Movie?". It features two buttons: "CANCEL" (outline blue) and "REMOVE" (red). The background of the page is dimmed.

Id	Name	Genre	Duration	Director	Box Office	Action
31	dolores	totam	102	vero	Yes	<button>REMOVE</button>
32	facere	totam	102	aut	Yes	<button>REMOVE</button>
33	dolores	totam	102	voluptas	No	<button>REMOVE</button>
34	eos	totam	115	velit	No	<button>REMOVE</button>
35	et	totam	99	animi	Yes	<button>REMOVE</button>
36	tempore	totam	91	ut	Yes	<button>REMOVE</button>
37	dolor	totam	113	odio	Yes	<button>REMOVE</button>
38	perferendis	totam	103	ullam	Yes	<button>REMOVE</button>
39	corporis	totam	104	non	Yes	<button>REMOVE</button>
40	molestias	totam	105	rem	Yes	<button>REMOVE</button>

Showing 31 to 40 of 40 results

Movies						
	Name	Genre	Duration	Director	Box Office	Action
31	dolores	totam	102	vero	Yes	<button>REMOVE</button>
32	facere	totam	102	aut	Yes	<button>REMOVE</button>
33	dolores	totam	102	voluptas	No	<button>REMOVE</button>
34	eos	totam	115	velit	No	<button>REMOVE</button>
35	et	totam	99	animi	Yes	<button>REMOVE</button>
36	tempore	totam	91	ut	Yes	<button>REMOVE</button>
37	dolor	totam	113	odio	Yes	<button>REMOVE</button>
38	perferendis	totam	103	ullam	Yes	<button>REMOVE</button>
39	corporis	totam	104	non	Yes	<button>REMOVE</button>

Showing 31 to 39 of 39 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]
    ];
```

```

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

```
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',
    ];
}
```

```

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(); $genres

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

```

```
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;
}

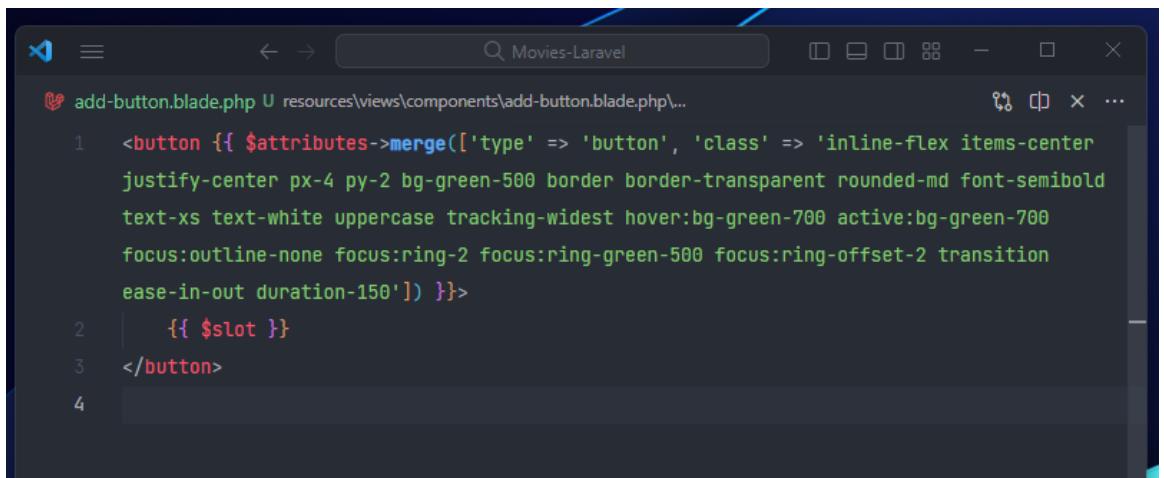
0 references | 0 overrides
public function confirmMovieEdit(Movie $movie)
{
    $this->movie = $movie;
    $this->confirmingMovieAdd = true;
}
```

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

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

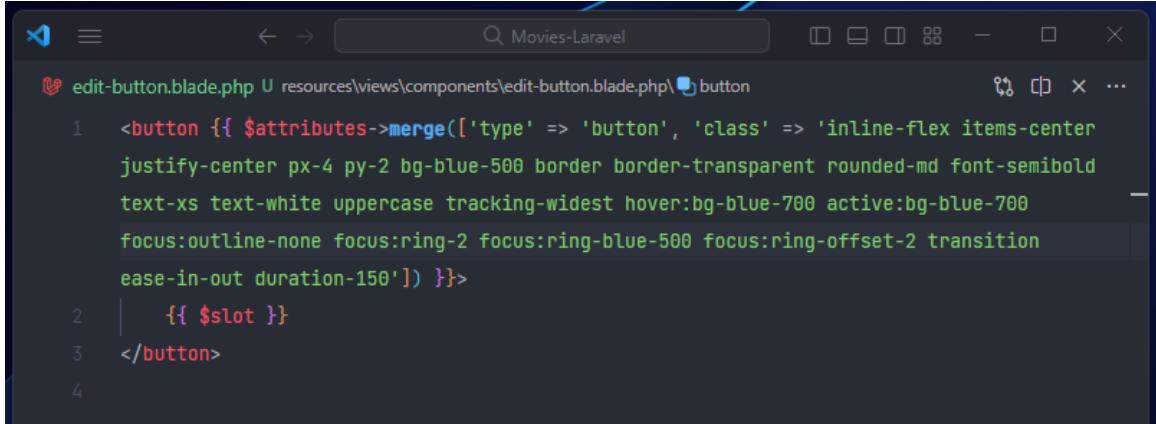
    0 references | 0 overrides
    public function user()
    {
        return $this->belongsTo(User::class);
    }
    0 references | 0 overrides
    public function scopeBoxOffice($query)
    {
        return $query->where('box_office', 1);
    }
}
```

- 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.



The screenshot shows a code editor window with the title "Movies-Laravel". The file being edited is "add-button.blade.php" located at "resources\views\components\add-button.blade.php". The code is as follows:

```
<button {{ $attributes->merge(['type' => 'button', 'class' => 'inline-flex items-center justify-center px-4 py-2 bg-green-500 border border-transparent rounded-md font-semibold text-xs text-white uppercase tracking-widest hover:bg-green-700 active:bg-green-700 focus:outline-none focus:ring-2 focus:ring-green-500 focus:ring-offset-2 transition ease-in-out duration-150']) }}>
    {{ $slot }}
</button>
```

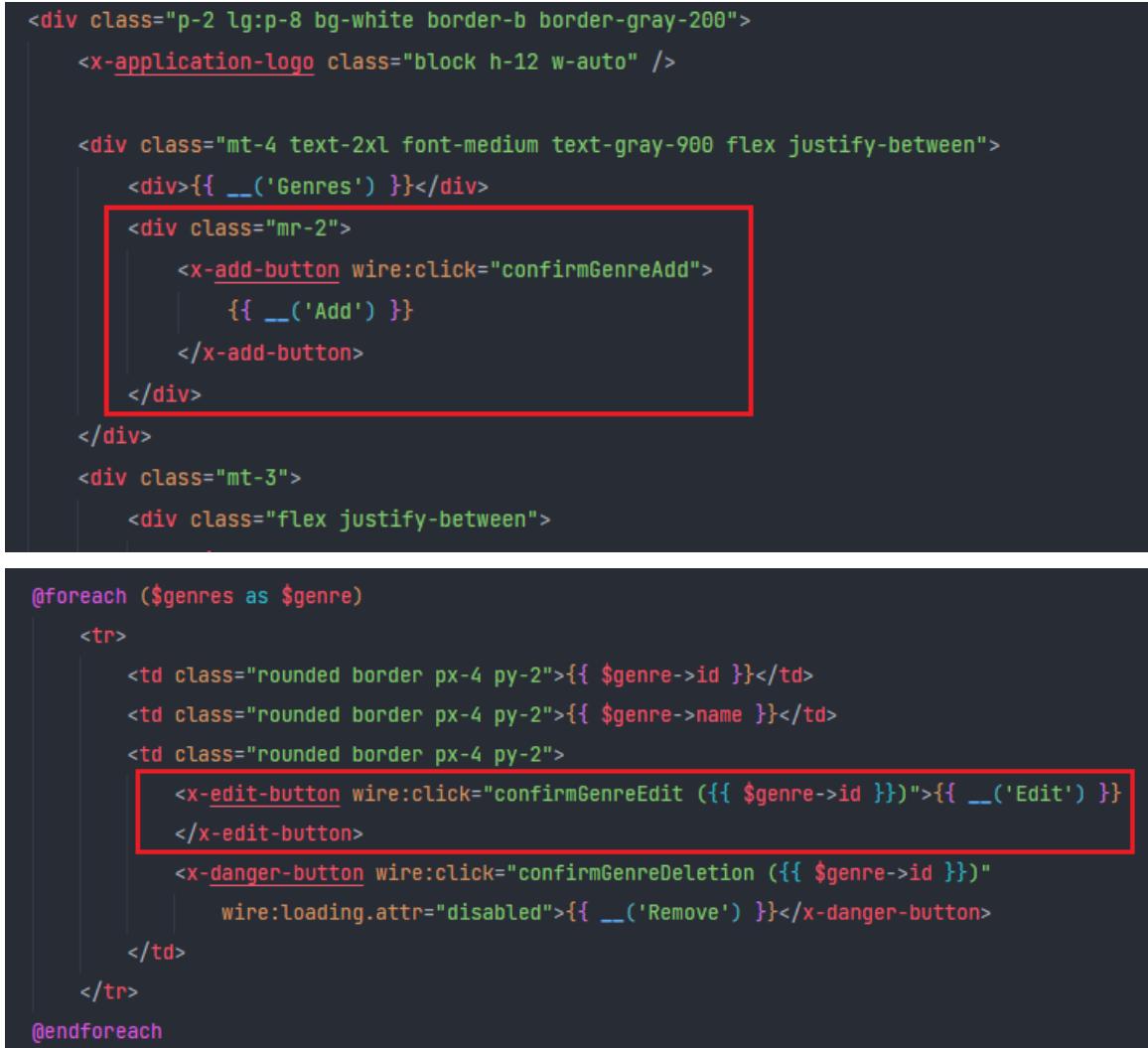


```

1 <button {{ $attributes->merge(['type' => 'button', 'class' => 'inline-flex items-center justify-center px-4 py-2 bg-blue-500 border border-transparent rounded-md font-semibold text-xs text-white uppercase tracking-widest hover:bg-blue-700 active:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 transition ease-in-out duration-150']) }}>
2   {{ $slot }}
3 </button>
4

```

- 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
    </div>
  </div>

```

```

<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 películas(resources\views\livewire\movies.blade.php)

```

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

```

```

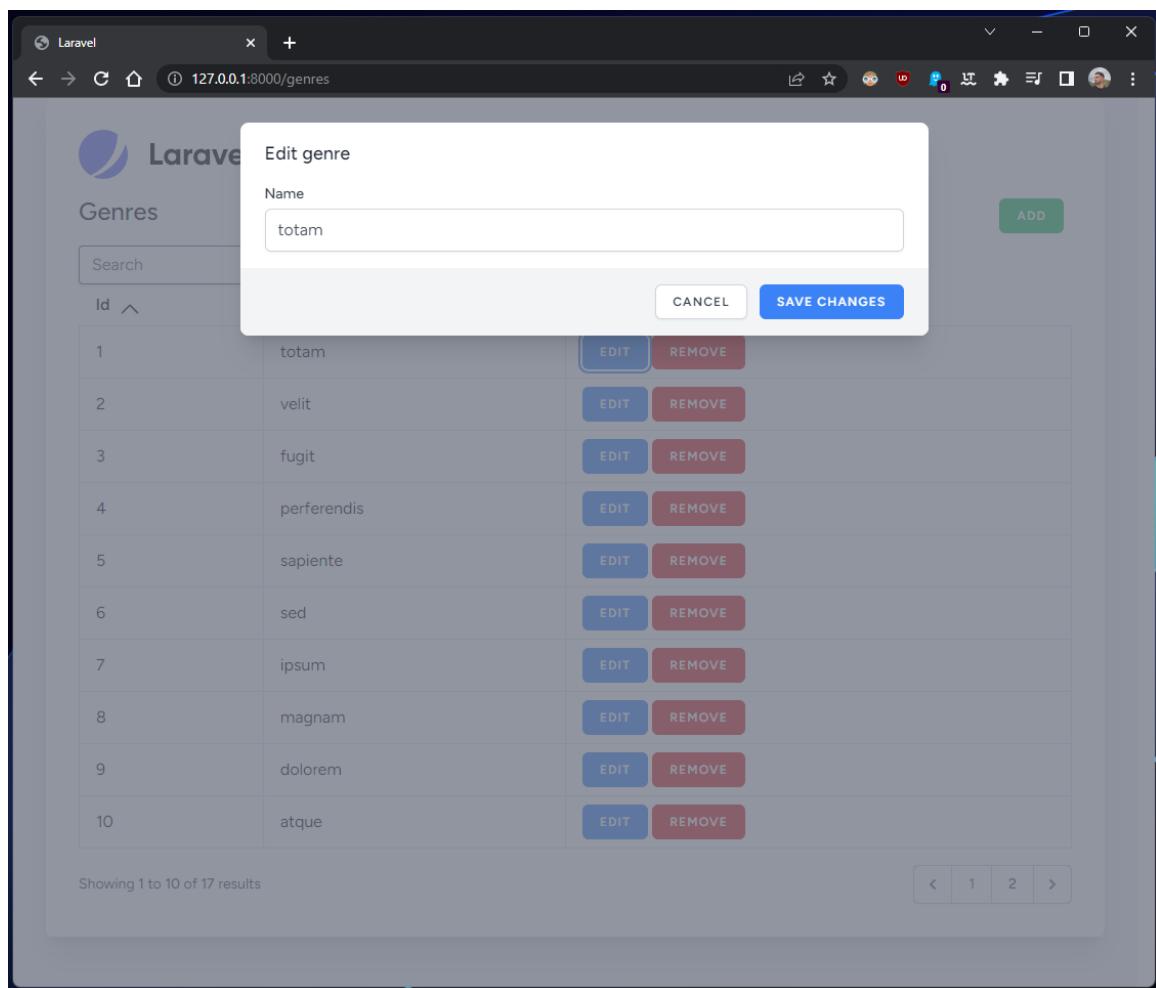
<div class="col-span-6 sm:col-span-4 mt-4">
    <input type="checkbox" wire:model.defer="movie.box_office" />
    <span class="ml-2 text-sm text-gray-600">{{ __('Box Office') }}</span>

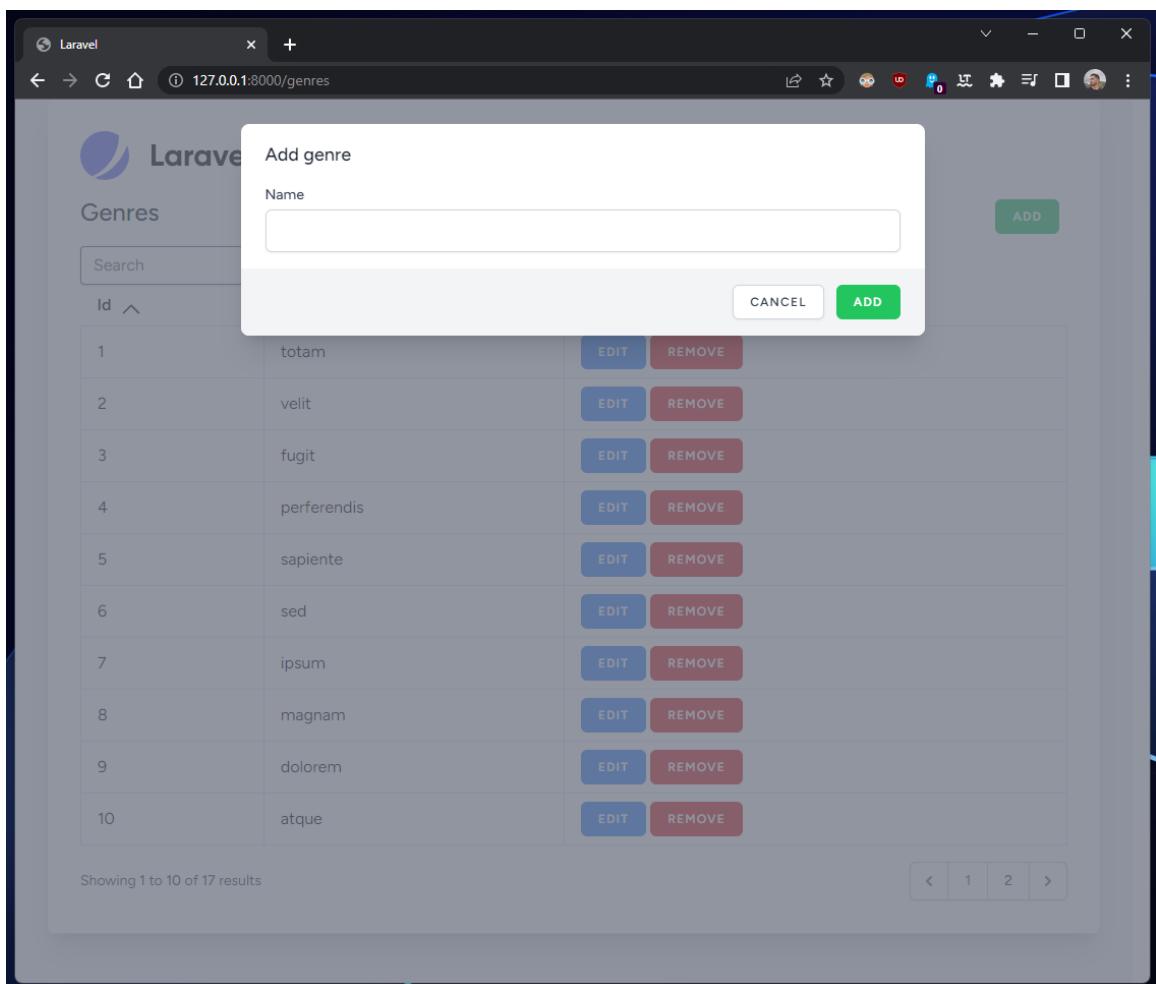
</div>
</x-slot>
<x-slot name="footer">
    <x-secondary-button wire:click="$toggle('confirmingMovieAdd', false)" wire:loading.attr="disabled">
        {{ __('Cancel') }}
    </x-secondary-button>

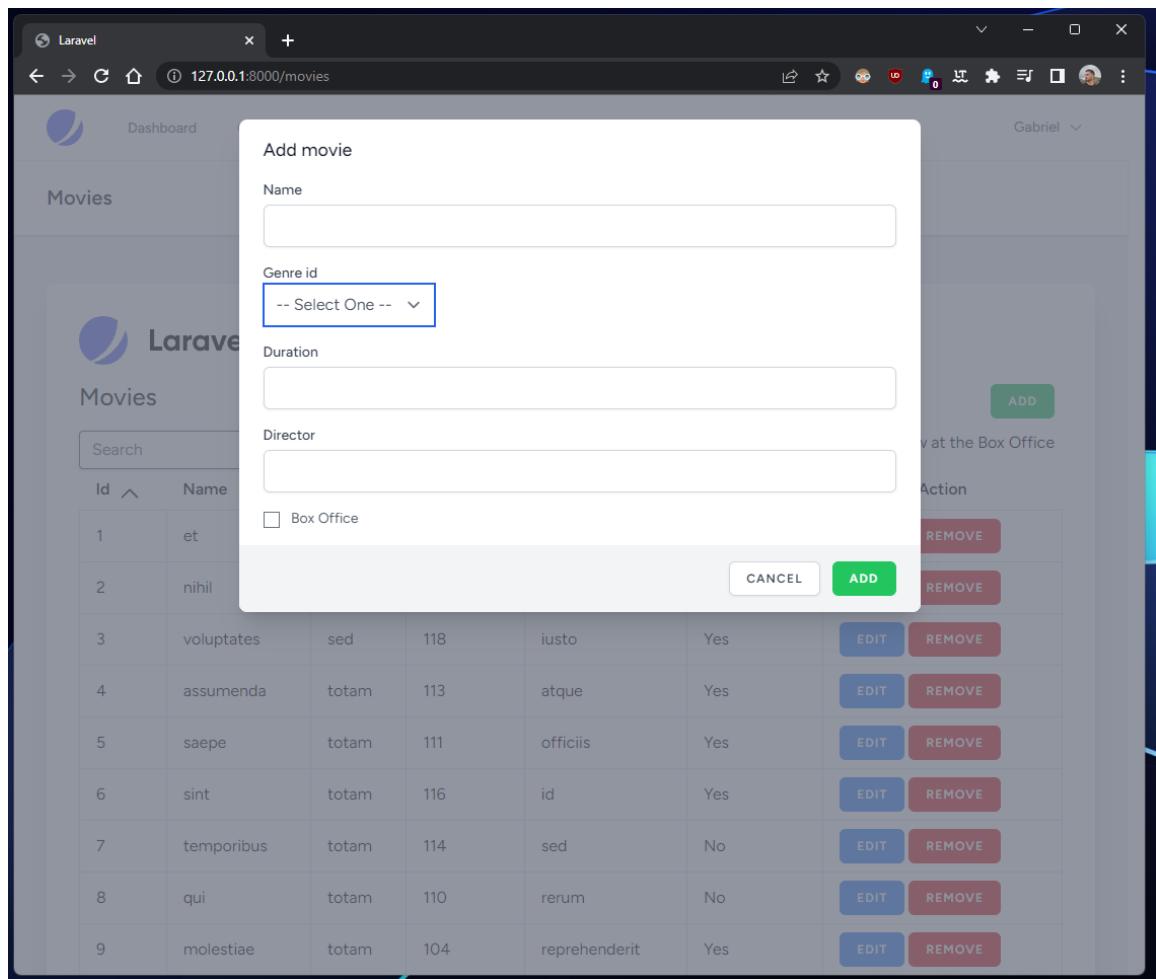
    @if (isset($this->movie->id))
        <x-edit-button class="ml-3" wire:click="addMovie ()" wire:loading.attr="disabled">
            {{ __('Save changes') }}
        </x-edit-button>
    @else
        <x-add-button class="ml-3" wire:click="addMovie ()" wire:loading.attr="disabled">
            {{ __('Add')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

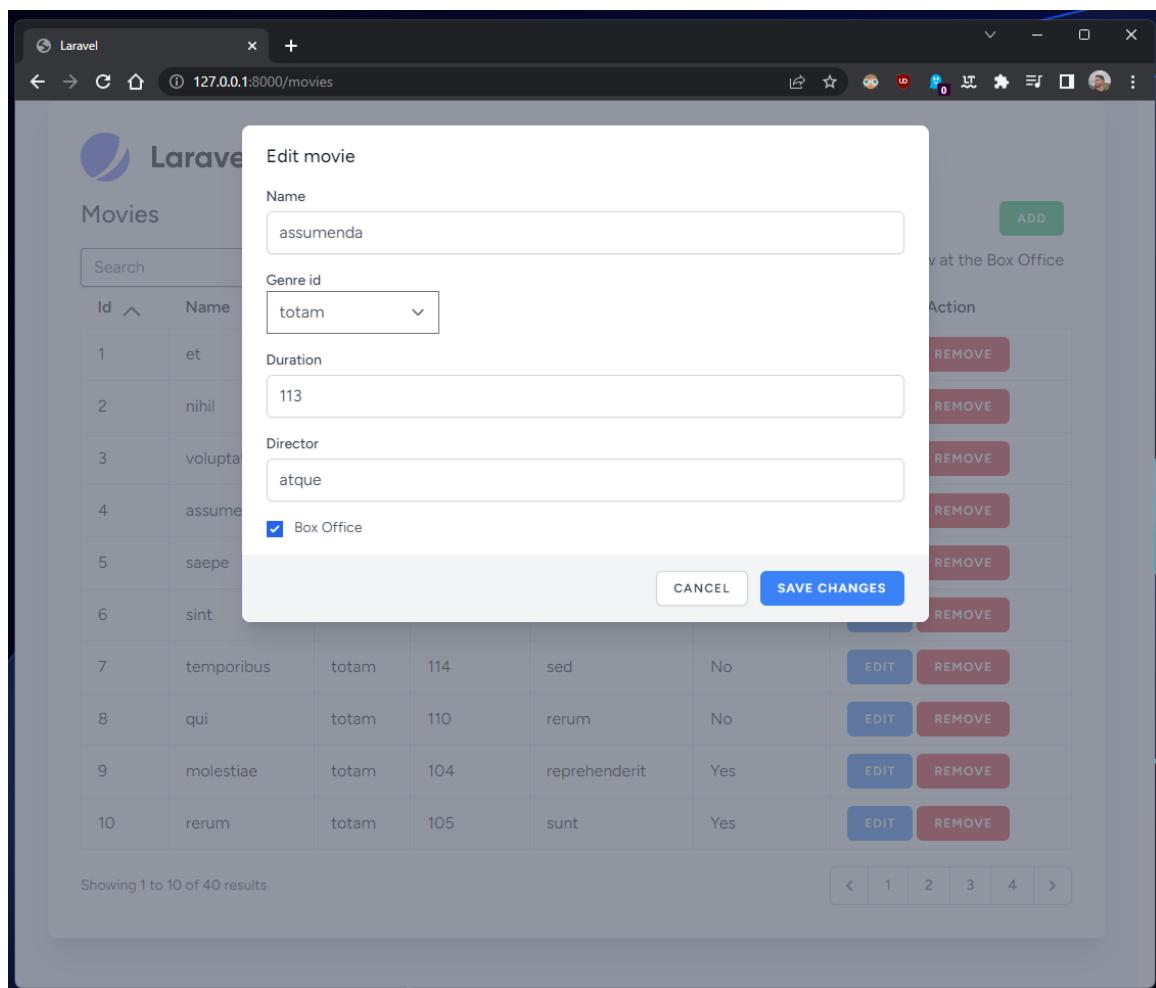
```

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









## 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.

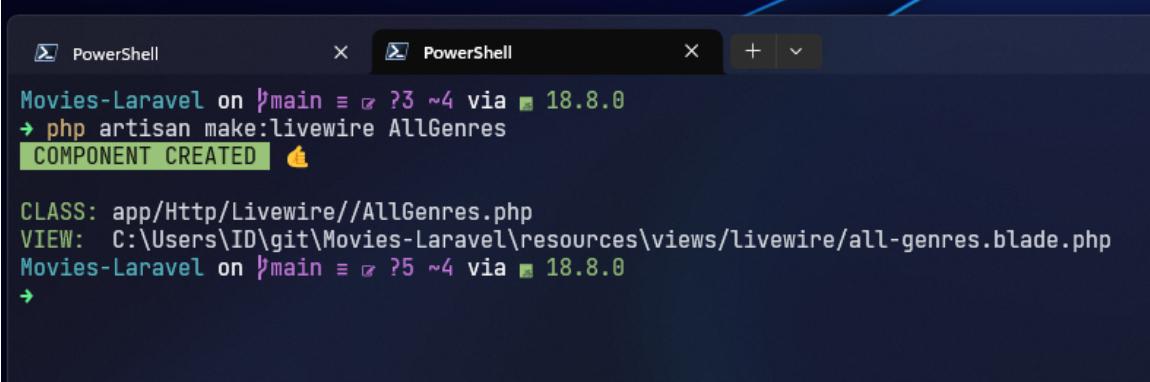
### 23.1. Antes de nada iremos a config\database.php y cambiaremos el siguiente valor de true a false

```
'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, // This line is highlighted with a red box
    '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
```



The screenshot shows two adjacent PowerShell windows. The left window displays the command `php artisan make:livewire AllGenres` and the output `COMPONENT CREATED`. The right window shows the generated files: `CLASS: app/Http/Livewire//AllGenres.php` and `VIEW: C:\Users\ID\git\Movies-Laravel\resources\views/livewire/all-genres.blade.php`. Both windows have a dark theme and are running on a Windows operating system.

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

```

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;
}
}
```

```

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');
    }
}

```

```

        $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;
}

}

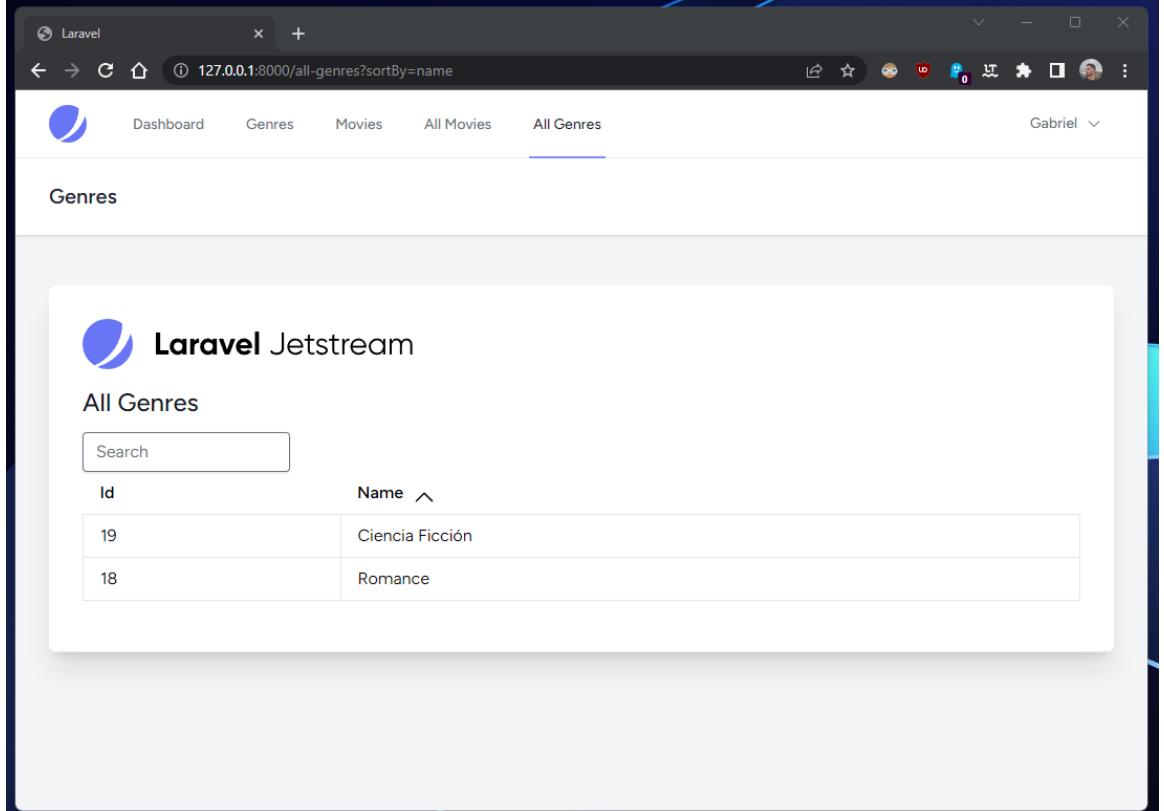
```

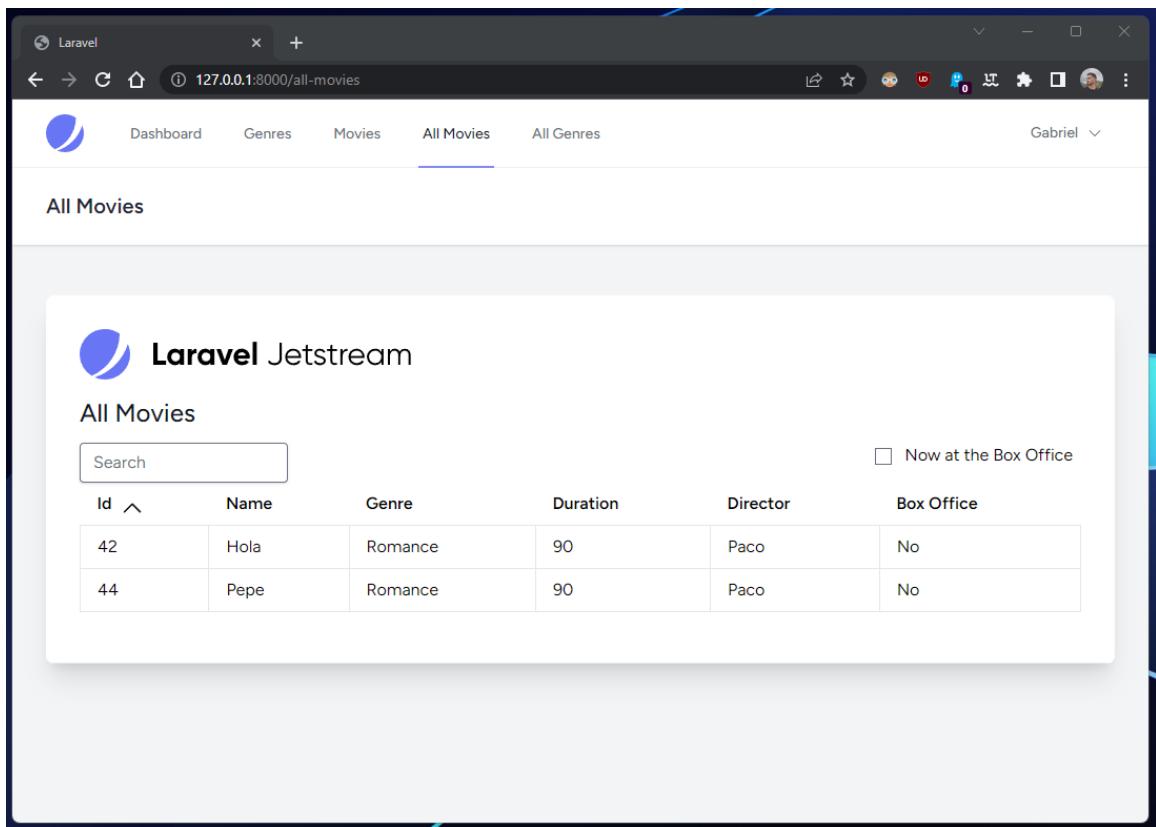
- 23.4. Vamos a la carpeta (resources\views\content) y creamos dos nuevos ficheros para nuestra vistas de la misma forma que hicimos anteriormente pero con all-genres y all-movies.
- 23.5. A continuación haremos la rutas en el archivo (routes\web.php)

```
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

- Vamos a la consola de comandos y ejecutamos

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

```
jenssegers/agent ..... DONE
laravel-lang/attributes ..... DONE
laravel-lang/http-statuses ..... DONE
laravel-lang/lang ..... DONE
laravel-lang/publisher ..... DONE
laravel/fortify ..... DONE
laravel/jetstream ..... DONE
laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
livewire/livewire ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE

92 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

[INFO] No publishable resources for tag [laravel-assets].
```

No security vulnerability advisories found  
Using version ^3.1 for laravel-lang/common  
Movies-Laravel on ✓main ✘?5 ~8 via █ 18.8.0  
→ |

- A continuación ejecutamos

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

The screenshot shows a terminal window with two tabs, both labeled "PowerShell". The current tab displays the command `php artisan lang:add es`. The output shows the process of adding the "es" language locale to the Laravel application. It includes logs for collecting source files, storing changes, and listing files processed, such as validation.php, http-statuses.php, json, auth.php, pagination.php, and passwords.php. The terminal also shows the system status "Movies-Laravel on main ✘ 25 ~8 via 18.8.0" and a prompt with a plus sign.

```
Movies-Laravel on \main ✘ 25 ~8 via 18.8.0
→ php artisan lang:add es

[INFO] Laravel\Lang\Attributes\Plugin.

Collect source ..... 4ms DONE
Collecting es ..... 2ms DONE

[INFO] Laravel\Lang\HttpStatuses\Plugin.

Collect source ..... 0ms DONE
Collecting es ..... 1ms DONE

[INFO] Laravel\Lang\Lang\Plugin.

Collect source ..... 4ms DONE
Collecting es ..... 2ms DONE

[INFO] Storing changes...

es/validation.php ..... 10ms DONE
es/http-statuses.php ..... 3ms DONE
es.json ..... 16ms DONE
es/auth.php ..... 3ms DONE
es/pagination.php ..... 6ms DONE
es/passwords.php ..... 4ms DONE

Movies-Laravel on \main ✘ 25 ~8 via 18.8.0
→ |
```

24.3. Seguimos ejecutando.

```
php artisan lang:update
```

The screenshot shows two separate PowerShell windows. The top window displays the command `php artisan local:collect` and its execution details:

```
INFO LaravelLang\HttpStatuses\Plugin.  
Collect source ..... 1ms DONE  
Collecting en ..... 1ms DONE  
Collecting es ..... 1ms DONE
```

The bottom window displays the command `php artisan lang:publish` and its execution details:

```
INFO LaravelLang\Lang\Plugin.  
Collect source ..... 4ms DONE  
Collecting en ..... 2ms DONE  
Collecting es ..... 3ms DONE
```

Both windows then show the command `php artisan config:cache` being run:

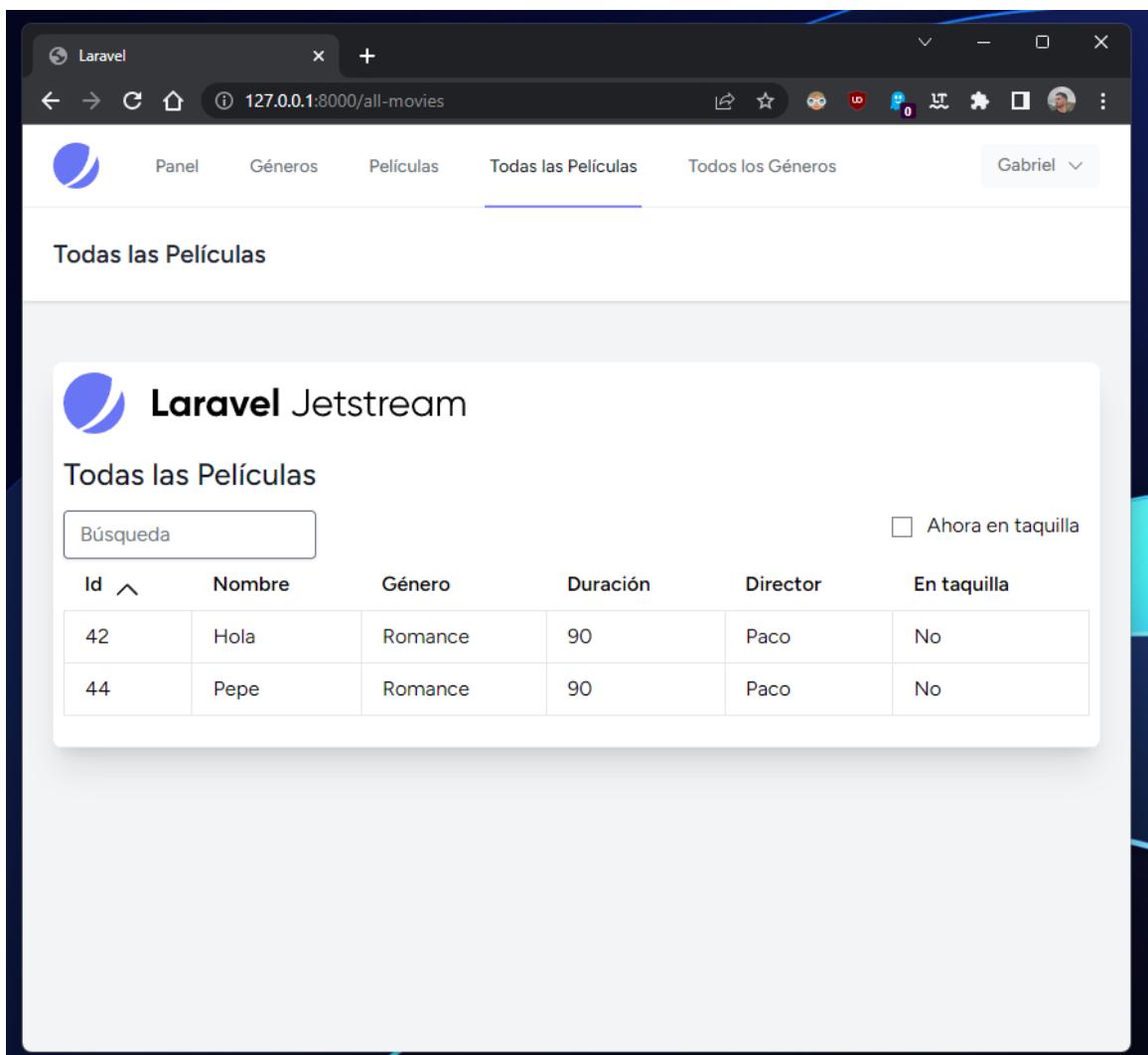
```
INFO Storing changes...  
en/validation.php ..... 11ms DONE  
en/http-statuses.php ..... 3ms DONE  
en.json ..... 10ms DONE  
en/auth.php ..... 2ms DONE  
en/pagination.php ..... 3ms DONE  
en/passwords.php ..... 5ms DONE  
es/validation.php ..... 17ms DONE  
es/http-statuses.php ..... 4ms DONE  
es.json ..... 10ms DONE  
es/auth.php ..... 2ms DONE  
es/pagination.php ..... 4ms DONE  
es/passwords.php ..... 2ms DONE
```

At the bottom of the windows, the status bar indicates "Movies-Laravel on main ✘ 26 ~8 via 18.8.0".

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

```
'locale' => '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.