

WebApp con Laravel – Filament – Livewire – PHP8



Laravel es un **framework de desarrollo web en PHP** de código abierto que sigue el **patrón MVC (Modelo-Vista-Controlador)**. Es uno de los frameworks más populares en el mundo del desarrollo web porque facilita la creación de aplicaciones robustas y escalables.

◆ ¿Para qué sirve Laravel?

Laravel ayuda a los desarrolladores a construir aplicaciones web de manera más rápida y eficiente, ya que proporciona herramientas y estructuras predefinidas para tareas comunes, como:

- **Gestión de rutas** (definir URLs y sus respectivas funciones)
- **Conexión con bases de datos** usando **Eloquent ORM**
- **Sistema de autenticación y permisos**
- **Gestión de sesiones y caché**
- **Colas de trabajos en segundo plano**
- **Plantillas con Blade** (su propio motor de plantillas)

◆ Características principales de Laravel

1. Eloquent ORM

- Es el sistema de mapeo objeto-relacional (ORM) de Laravel, que permite interactuar con bases de datos de forma más sencilla y fluida.

2. Blade (Motor de plantillas)

- Permite escribir HTML con una sintaxis más limpia y reutilizable.

3. Sistema de Rutas

- Definir rutas en Laravel es muy sencillo y permite un control total sobre cómo se estructuran las URL.

4. Middleware

- Se pueden definir filtros que controlen el acceso a ciertas partes de la aplicación.

5. Autenticación y Seguridad

- Laravel tiene un sistema de autenticación incorporado, lo que facilita la implementación de inicios de sesión y permisos.

6. Migraciones y Seeders

- Las migraciones permiten manejar la estructura de la base de datos mediante código, facilitando la versión y actualización del esquema de la base de datos.

7. API y JSON Responses

- Laravel facilita la creación de API RESTful con respuestas en JSON.

8. Colas y Trabajos en segundo plano

- Laravel permite gestionar tareas largas en segundo plano, mejorando el rendimiento de la aplicación.

9. Laravel Mix

- Es una herramienta que ayuda a compilar y optimizar archivos CSS y JavaScript.

10. Testing Integrado

- Incluye herramientas para pruebas unitarias y de integración.

◆ ¿Cuándo usar Laravel?

- ✓ Cuando necesitas desarrollar aplicaciones web modernas con PHP.
- ✓ Si buscas rapidez en el desarrollo y una estructura clara.
- ✓ Si trabajas en proyectos donde la seguridad y la escalabilidad son importantes.
- ✓ Para desarrollar APIs y aplicaciones web con paneles de administración.

¿Qué es Filament PHP? 🚀



Filament PHP es un **framework de administración** para **Laravel**, diseñado para crear paneles de administración modernos, rápidos y altamente personalizables con una experiencia de usuario intuitiva.

Es una alternativa a **Nova** (de Laravel) o **Backpack**, pero con la ventaja de ser **open-source y gratuito**. Su enfoque es **minimalista y basado en Livewire**, lo que permite crear interfaces dinámicas sin necesidad de escribir mucho JavaScript.

◆ Características principales

- ✓ Interfaz moderna y personalizable 
- ✓ Construido sobre Livewire y Alpine.js 
- ✓ CRUDs con pocos comandos 
- ✓ Autenticación y roles incorporados 
- ✓ Widgets y gráficos dinámicos 
- ✓ Relaciones avanzadas (BelongsTo, HasMany, etc.) 
- ✓ Soporte para múltiples paneles de administración 
- ✓ Filtros, búsquedas y paginación automáticas 

◆ Widgets y Paneles personalizados

Filament te permite agregar **widgets y dashboards personalizados** para visualizar datos.

◆ ¿Cuándo usar Filament?

- ✓ Si necesitas un **panel de administración rápido y elegante** para tu aplicación Laravel.
- ✓ Si quieres **evitar escribir mucho código** para CRUDs y formularios.
- ✓ Si buscas una alternativa **más ligera y gratuita** que Laravel Nova.
- ✓ Si te gusta trabajar con **Livewire y Alpine.js**.

¿Qué es Livewire? ⚡



LIVEWIRE

Livewire es un **framework full-stack** para **Laravel** que permite crear interfaces dinámicas sin necesidad de escribir mucho JavaScript.

Básicamente, **te permite construir componentes interactivos en Laravel usando solo PHP**. Es una alternativa a frameworks front-end como **Vue.js o React**, pero sin salir del ecosistema de Laravel.

◆ ¿Cómo funciona Livewire?

Livewire trabaja con **componentes** que se renderizan en el servidor y se actualizan automáticamente en el navegador sin recargar la página.

En lugar de usar **AJAX o JavaScript manualmente**, Livewire se encarga de manejar las peticiones y actualizaciones en segundo plano.

◆ Características principales

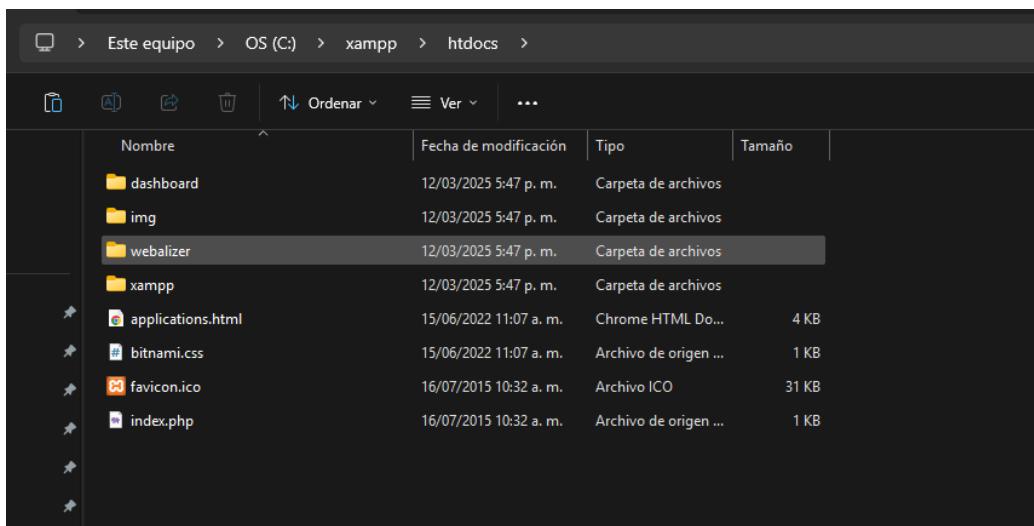
- Sin necesidad de JavaScript (aunque usa Alpine.js internamente) 💾
- Reactividad en tiempo real 🔍
- Fácil integración con Laravel 🎉
- Validaciones en el servidor 🔒
- Soporte para eventos, formularios y Livewire Polling 🕒
- Compatible con Alpine.js 🎉

◆ ¿Cómo empezar?

The screenshot shows a GitHub Gist page with the title 'install_laravel_webapp'. The file is named 'install_laravel_webapp.md'. The content starts with a rocket icon and the heading 'Instalación de Laravel, Filament, Livewire con PHP 8+ y XAMPP/WAMP'. It describes the steps to install and configure a webapp using Laravel, Filament, and Livewire with PHP 8+ and XAMPP or WAMP. Below this, there's a section titled '1. Instalar XAMPP o WAMP' with links to download XAMPP, WAMP, and Laragon, and a note to ensure Laragon is included.

Primero vamos a nuestro [gist](#) y hacemos la configuración necesaria para poder trabajar con nuestro entorno ya configurado.

En esta guía paso a paso trabajare con Xampp y composer, una vez configurados e iniciados procedemos a crear un proyecto nuevo abriendo la terminal en la ubicación de carpetas de proyecto



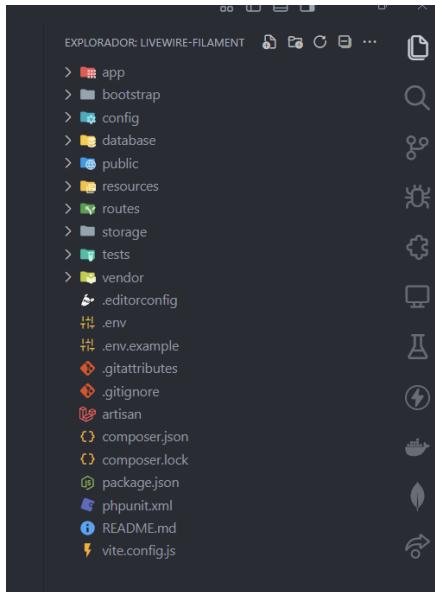
Abrimos la terminal y ejecutamos el siguiente comando

The screenshot shows a terminal window with the command 'composer create-project laravel/laravel livewire-filament --prefer-dist' being typed. The terminal interface includes tabs for 'Pair', 'Dispatch Beta', and keyboard shortcuts like 'Ctrl', 'Shift', and 'Alt'.

Si de casualidad arroja errores con el .zip de la paqueteria se debe ingresar al archivo de configuracion de php y habilitar el .zip

```
;extension=sqlite3  
;extension=tidy  
;extension=xsl  
extension=zip  
  
zend_extension=opcache
```

Una vez finalizada la instalación procedemos a abrir en visual studio code la carpeta del proyecto



Ahora vamos a modificar el proyecto para que trabaje con MySQL ingresamos a nuestro archivo de variables de entorno .ENV ubicamos lo siguiente

```
22
23 DB_CONNECTION=sqlite
24 # DB_HOST=127.0.0.1
25 # DB_PORT=3306
26 # DB_DATABASE=laravel
27 # DB_USERNAME=root
28 # DB_PASSWORD=
29
```

Y procedemos a dejarlo así

```
23 DB_CONNECTION=mysql
24 DB_HOST=127.0.0.1
25 DB_PORT=3306
26 DB_DATABASE=livewire-filament
27 DB_USERNAME=root
28 DB_PASSWORD=
29
```

El nombre por defecto de la base de datos es el nombre del proyecto en donde dejaremos todo habilitado, si en el caso de tener password la database, deben colocarlo allí.

Ahora debemos correr las migraciones en nuestro terminal con el siguiente comando

```
C:\xampp\htdocs\livewire-filament A Pair Ctrl I Dispatch Beta
php artisan migrate
```

Si la base de datos no esa creada nos saldrá un aviso y le escribimos que si

```
C:\xampp\htdocs\livewire-filament
php artisan migrate

WARN The database 'livewire-filament' does not exist on the 'mysql' connection
Would you like to create it? (yes/no) [yes]
> yes
```

Una vez realizado el proceso nos mostrara que se escribieron con éxito las migraciones de la base de datos

```
C:\xampp\htdocs\livewire-filament (56.447s)
php artisan migrate

WARN The database 'livewire-filament' does not exist on the 'mysql' connection.
Would you like to create it? (yes/no) [yes]
> yes

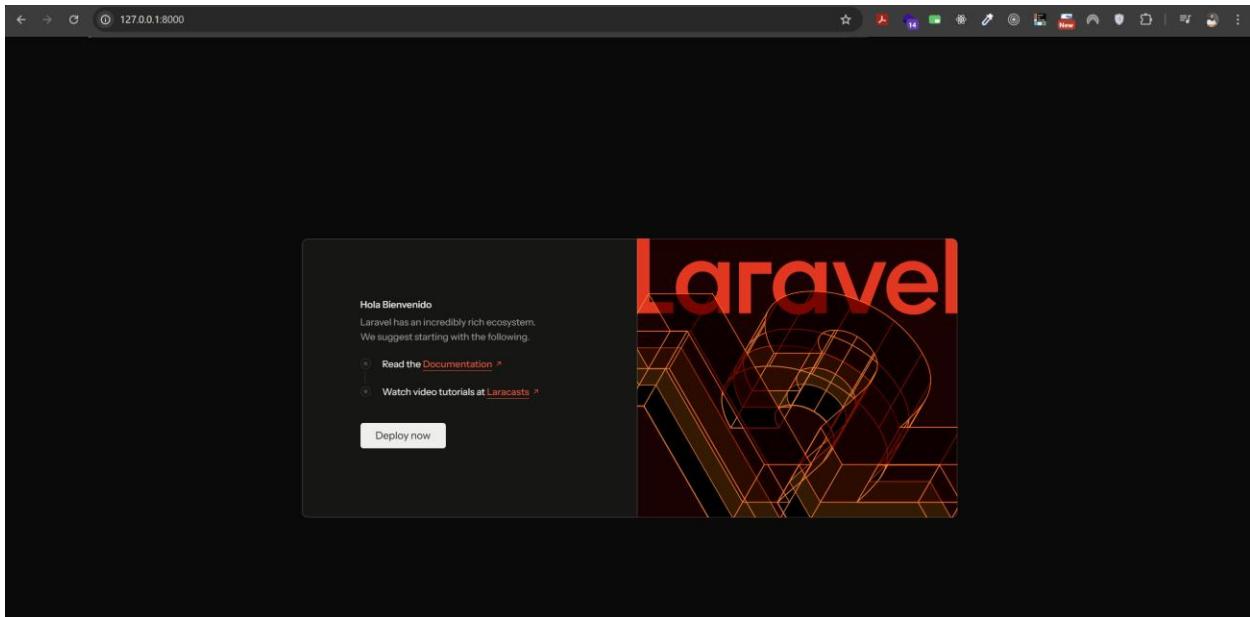
INFO Preparing database.
Creating migration table ..... 262.61ms DONE
INFO Running migrations.

0001_01_01_000000_create_users_table ..... 287.40ms DONE
0001_01_01_000001_create_cache_table ..... 28.20ms DONE
0001_01_01_000002_create_jobs_table ..... 124.42ms DONE
```

A continuación ejecutamos el servidor directamente en la terminal con el siguiente comando:

```
C:\xampp\htdocs\livewire-filament A Pair Ctrl I Dispatch Beta Ctrl
php artisan serve
```

De esa manera vamos a estar ejecutando el sitio y vamos al navegador a la ruta que nos muestra y podremos observar



Ahora vamos a utilizar un [template](#) básico para el proceso de creación del catalogo que vamos a implementar, ingresen al repositorio y lo clonan.

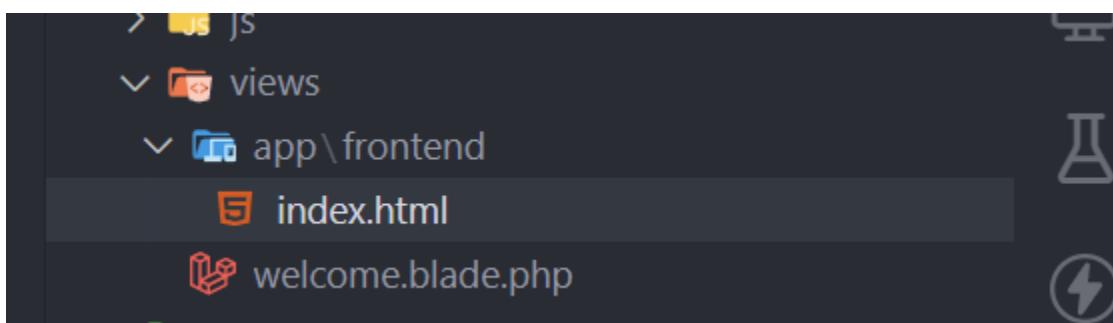


Vamos a continuar con las configuraciones del proyecto de laravel para ello vamos a modificar lo siguiente:

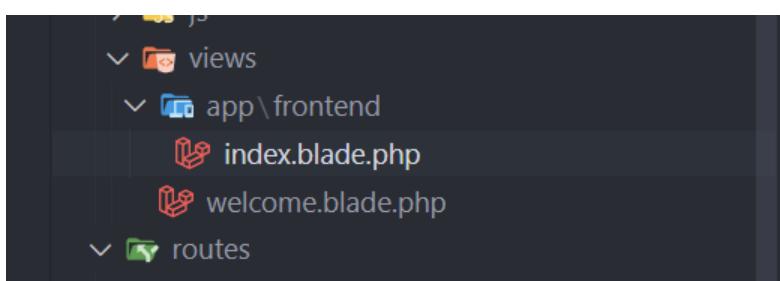
```
! .env
1 APP_NAME=Catalogo
2 APP_ENV=local
3 APP_KEY=base64:4egDd6DKc1gnqMkjCjCUur03WDT5aw7ozpGW0BX0b4k=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 APP_LOCALE=es
8 APP_FALLBACK_LOCALE=es
9 APP_FAKE_LOCALE=es_ES
10
```

Nombre de la app y los idiomas base en este caso español.

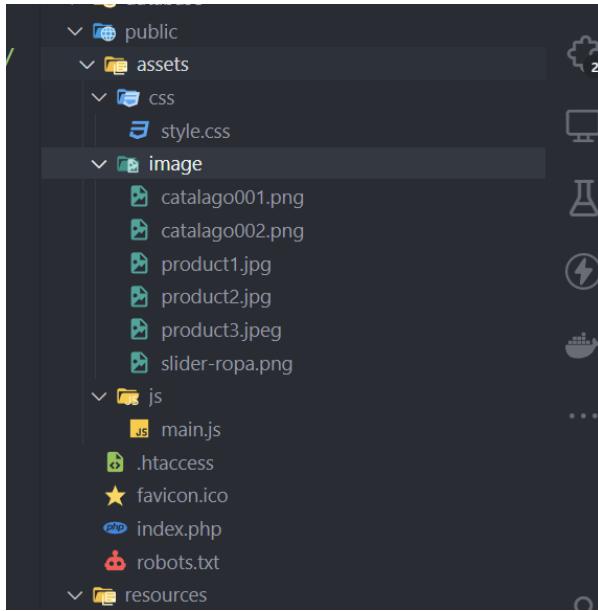
Vamos a organizar la carpeta de nuestra vista para el frontend y allí vamos a dejar nuestro template de catalogo de la siguiente manera: /resources/views/



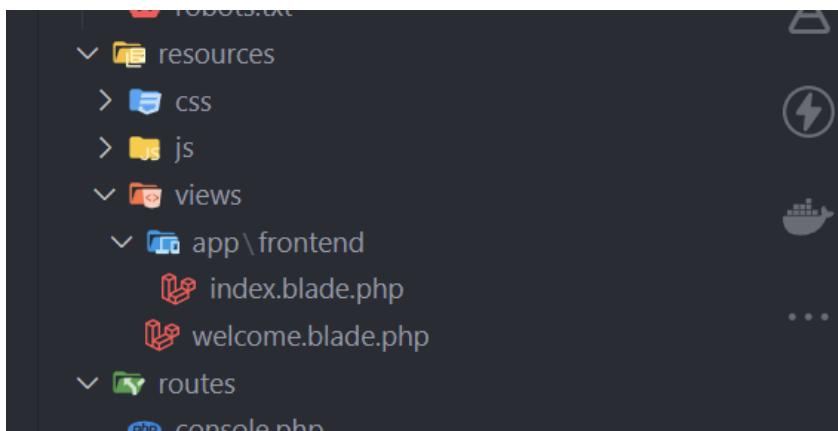
Pasamos solamente el index.html y lo vamos a renombrar de la siguiente manera:



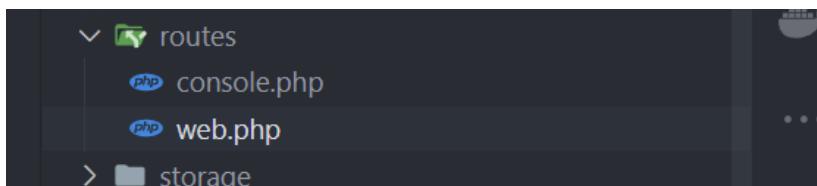
Y ahora vamos a pasar a nuestra carpeta de public/assets el resto de las carpetas y contenido con el cual vamos a trabajar:



Para comprobar que todo esto quedo ubicado correctamente se vera reflejado en la carpeta resources:



Ahora vamos a cambiar la ruta de nuestro Home a la ruta de nuestro index para ello nos ubicamos en routes/web.php:



Y aquí vamos a dejar de la siguiente manera:

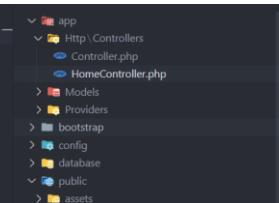
```
routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  Route::get(uri: '/', );
6 |
```

Ahora vamos a abrir nuestra terminal de nuevo y vamos a crear el controlador que nos permitirá personalizar la vista, escribimos el siguiente comando:

```
@ hdtoledo on livewire-filament
# php artisan make:controller HomeController
[INFO] Controller [C:\xampp\htdocs\livewire-filament\app\Http\Controllers\HomeController.php] created successfully.
```

Para verificarlo nos ubicamos dentro de nuestra carpeta app/http/controllers

```
app > Http > Controllers > HomeController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class HomeController extends Controller
8  {
9  }
10 }
```



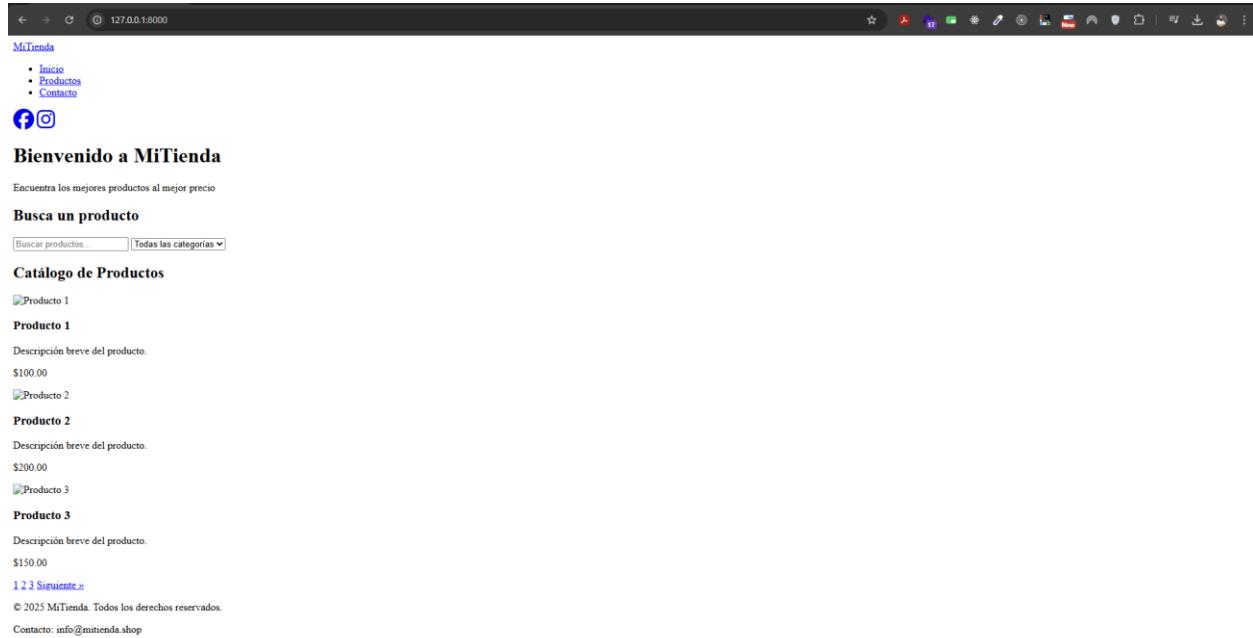
Ahora vamos a crear nuestro primer método para que nos genere la vista:

```
app > Http > Controllers > HomeController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class HomeController extends Controller
8  {
9  }
10 public function index(): View{
11     return view(view: "app.frontend.index");
12 }
13 }
```

Y ahora vamos a utilizar nuestro controlador en web.php de la siguiente manera:

```
routes > web.php
1  <?php
2
3  use App\Http\Controllers\HomeController;
4  use Illuminate\Support\Facades\Route;
5
6  Route::get(uri: '/', action: [HomeController::class, 'index']);
7 |
```

Esto permitirá que ejecutemos sobre la ruta de / nuestra vista del catálogo, si vamos a revisar en el navegador debe salirnos de la siguiente manera:



Ahora tenemos que configurar las rutas de nuestro index para que podamos conectarlo a nuestro css, imágenes y js, para ello en nuestro index.blade.php hacemos las modificaciones correspondientes:

```
6 |     <title>Catálogo de Productos</title>
7 |     <link rel="stylesheet" href=".//assets/css/style.css">
8 |     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/lib
```

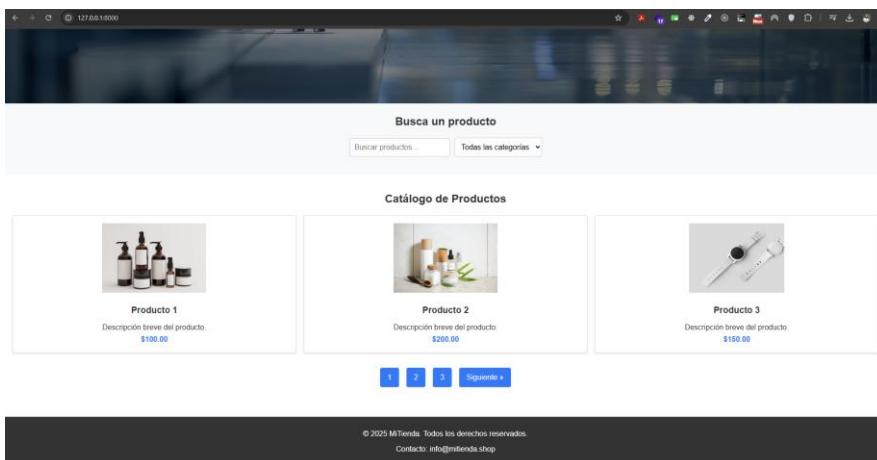
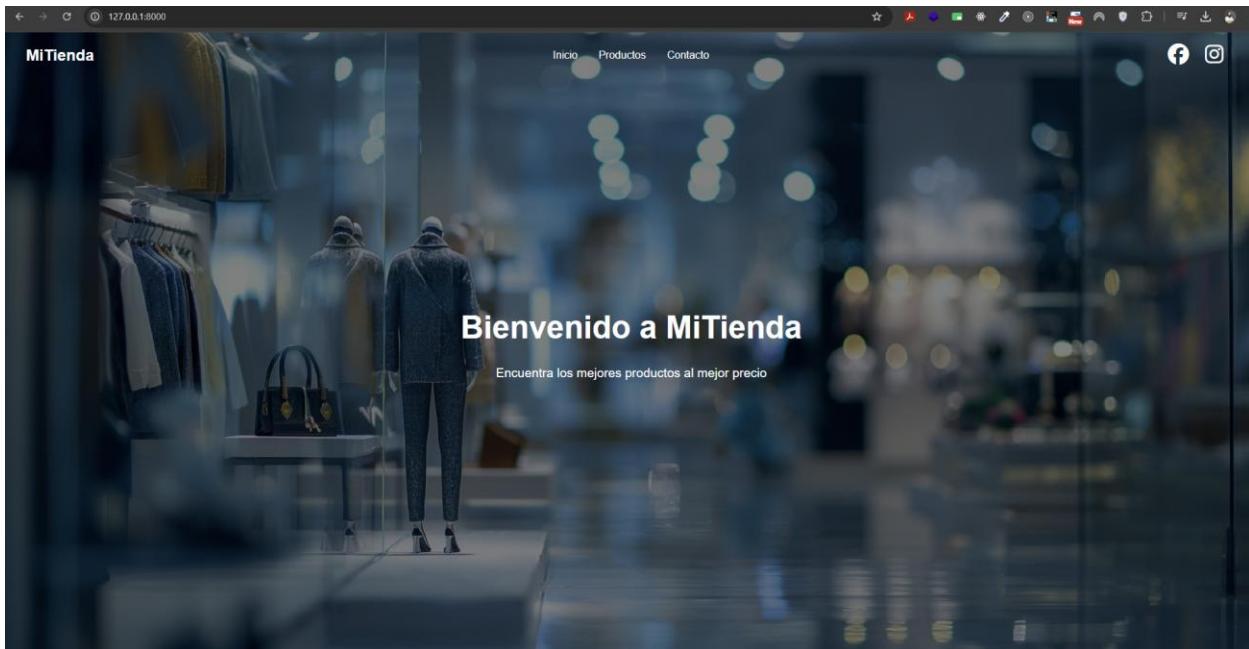
```
<div class="product-card">
    
    <h3>Producto 1</h3>
```

```
<div class="product-card">
    
    <h3>Producto 2</h3>
    <p>Descripción breve del producto.</p>
```

```
<div class="product-card">
    
    <h3>Producto 3</h3>
    <p>Descripción breve del producto.</p>
```

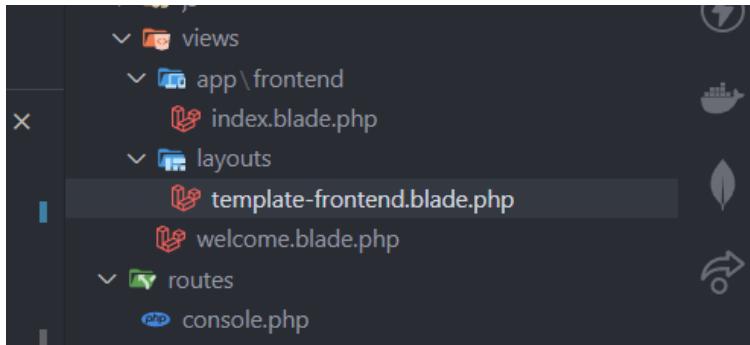
```
90 |     
91 |     <script src="../assets/js/main.js"></script>
92 |   </body>
```

De esta manera volvemos a revisar al navegador para verificar que todo este saliendo:



Esta manera sería la tradicional para poder pasar nuestros estilos al frente, sin embargo, acá vamos a manejar los layouts, un layout es una plantilla base que se utiliza para estructurar y reutilizar el diseño de las vistas en una aplicación web. En lugar de repetir código HTML en cada vista, Laravel permite definir un archivo de diseño principal y extenderlo en otras vistas secundarias.

Ahora vamos a crear en views/ nuestra carpeta de layouts y nuestro template:



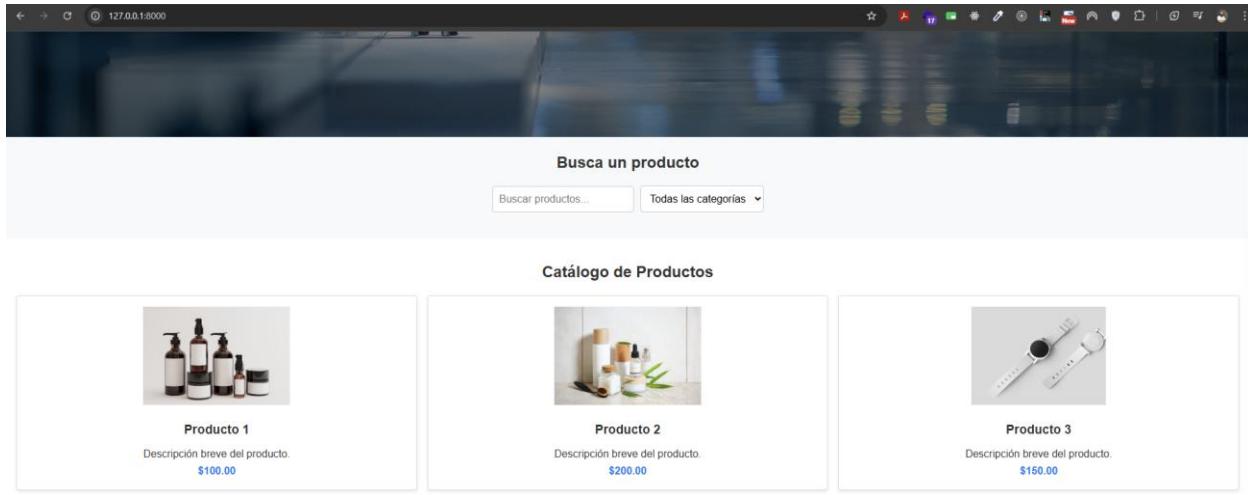
Ahora vamos a colocar todo el contenido de nuestro index.blade.php dentro de nuestro template-frontend.blade.php:

```
resources > views > layouts > template-frontend.blade.php > html
2  <html lang="es">
10 <body>
83   <footer class="footer">
84     <div class="footer-content">
86       <p>Contacto: info@mitienda.shop</p>
87     </div>
88   </footer>
89
90   <!-- Scripts -->
91   <script src=".//assets/js/main.js"></script>
92 </body>
93 </html>
94
```

Y para nuestro index.blade.php vamos a realizar una extensión de nuestro layout a través de la directiva de Blade, y dejamos de la siguiente manera:

```
resources > views > app > frontend > index.blade.php
1 @extends(view: 'layouts.template-frontend')
```

Revisamos en nuestro navegador y deberíamos tener lo siguiente:



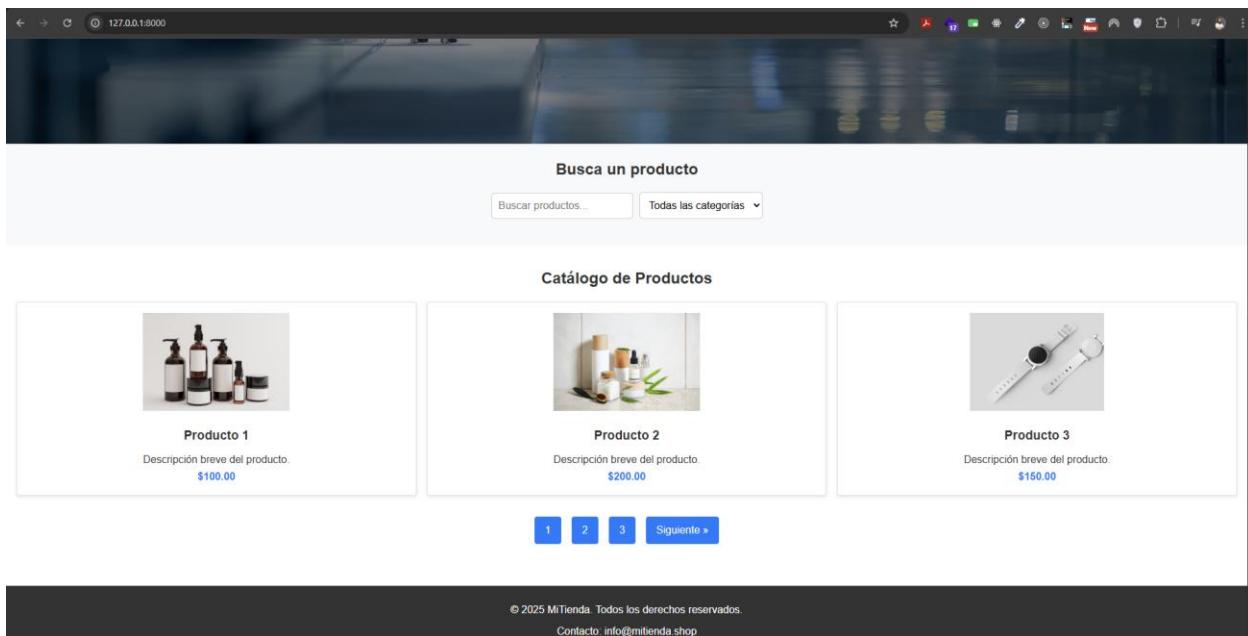
Para nuestras rutas vamos a implementar ahora la directiva de asset que nos permitirá redireccionar los diferentes archivos para ello vamos a actualizar nuestro código del template de la siguiente manera:

```

3   <meta name="viewport" content="width=device-width, initial-scale=1.0">
4   <title>Catálogo de Productos</title>
5   <link rel="stylesheet" href="{{ asset(path: 'assets/css/style.css') }}>
6   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.7.1
7   integrity="sha512-5Hs3dF2AEpkpNAR7UiOHba+lRSJNeM2ECkwxUIxClQ/FLyGTbNapWXB4tP889k5T5Ju
8
9
10  <!-- Producto 1 -->
11  <div class="product-card">
12      
13      <h3>Producto 1</h3>
14      <p>Descripción breve del producto.</p>
15
16  <!-- Producto 2 -->
17  <div class="product-card">
18      
19      <h3>Producto 2</h3>
20      <p>Descripción breve del producto.</p>
21
22  <!-- Producto 3 -->
23  <div class="product-card">
24      
25      <h3>Producto 3</h3>
26      <p>Descripción breve del producto.</p>
27
28
29  <!-- Scripts -->
30  <script src="{{ asset(path: '/assets/js/main.js') }}></script>
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92

```

Comprobamos en el navegador:



Instalación de Filamentphp

Vamos a revisar la documentación en su [pagina](#), y al revisar debemos tener ciertas características ya instaladas:

Requirements

Filament requires the following to run:

- PHP 8.1+
- Laravel v10.0+
- Livewire v3.0+

De acuerdo con nuestra instalación hasta el momento ya tenemos PHP 8.1 y Laravel, nos faltaría instalar [Livewire](#), para ello vamos a nuestra terminal y colocamos el siguiente comando:

```
○ @ hdtoledo on └─ livewire-filament └─
# composer require livewire/livewire
/composer.json has been updated
```

Una vez instalado vamos a proceder a instalar filamentphp con el siguiente comando:

```
@ hdtoledo on livewire-filament
# composer require filament/filament:^3.3" -W
```

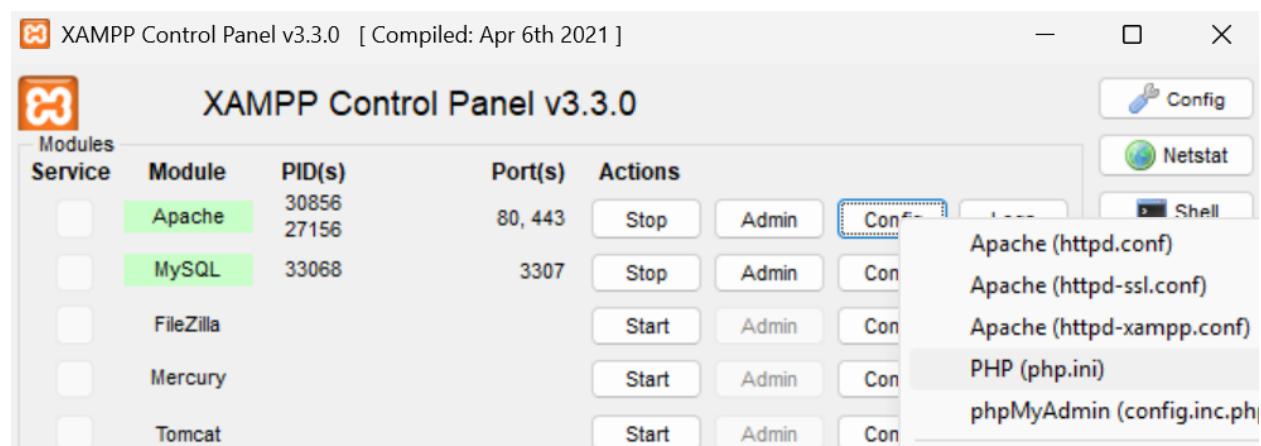
Dado el caso de que se presenten inconvenientes en la instalación pueden salir los siguientes errores de configuración de php:

```
The "3.3" constraint for "filament/filament" appears too strict and will likely not match what you want. See https://getcomposer.org/constraints
./composer.json has been updated
Running composer update filament/filament --with-all-dependencies
Loading composer repositories with package information
Updating dependencies
Your requirements could not be resolved to an installable set of packages.

  Problem 1
    - Root composer.json requires filament/filament 3.3 -> satisfiable by filament/filament[v3.3.0].
    - filament/filament v3.3.0 requires filament/support v3.3.0 -> satisfiable by filament/support[v3.3.0].
    - filament/support v3.3.0 requires ext-intl * -> it is missing from your system. Install or enable PHP's intl extension.

To enable extensions, verify that they are enabled in your .ini files:
- C:\xampp\php\php.ini
```

Esto indica que la extensión intl de PHP no está instalada o habilitada en tu sistema. Para solucionarlo, vamos a nuestro xampp y vamos a ubicarnos en este menú:



Allí vamos a ir a la ubicación de la ruta C:\xampp\php\php.ini en donde vamos a modificar:

```
;extension=gmp
;extension=intl
;extension=imap
```

Y lo dejamos así:

```
;extension=gmp  
extension=intl  
;extension=imap
```

De esa manera lo habilitaremos, guardamos los cambios y reiniciamos los servicios de nuestro xampp, ahora pasaremos a la instalación de filamentphp nuevamente y debemos tener la ejecución completa del instalador, una vez finalizada podemos hacer la instalación de los paneles con el siguiente comando:

```
@ hdtolledo on livewire-filament ✘  
# php artisan filament:install --panels
```

En este primer paso no pide un identificador para acceder al panel de control para ello vamos a dejarlo como dashboard:

```
What is the ID? [admin]  
› dashboard
```

Luego nos preguntara si queremos darle una votación al repo de filament en github, escribimos que no y continuamos.

```
All done! Would you like to show some love by starring the Filament repo on GitHub? (yes/no) [yes]  
› no
```

A continuación, vamos a crear nuestro usuario con el siguiente comando:

```
@ hdtolledo on livewire-filament ✘  
# php artisan make:filament-user
```

Allí vamos a colocar el nombre del usuario en este caso utilizaremos admin, también lo pueden personalizar sin ningún inconveniente:

```
Name:  
› admin
```

Colocamos el correo electrónico y el password que vamos a dejar para iniciar sesión:

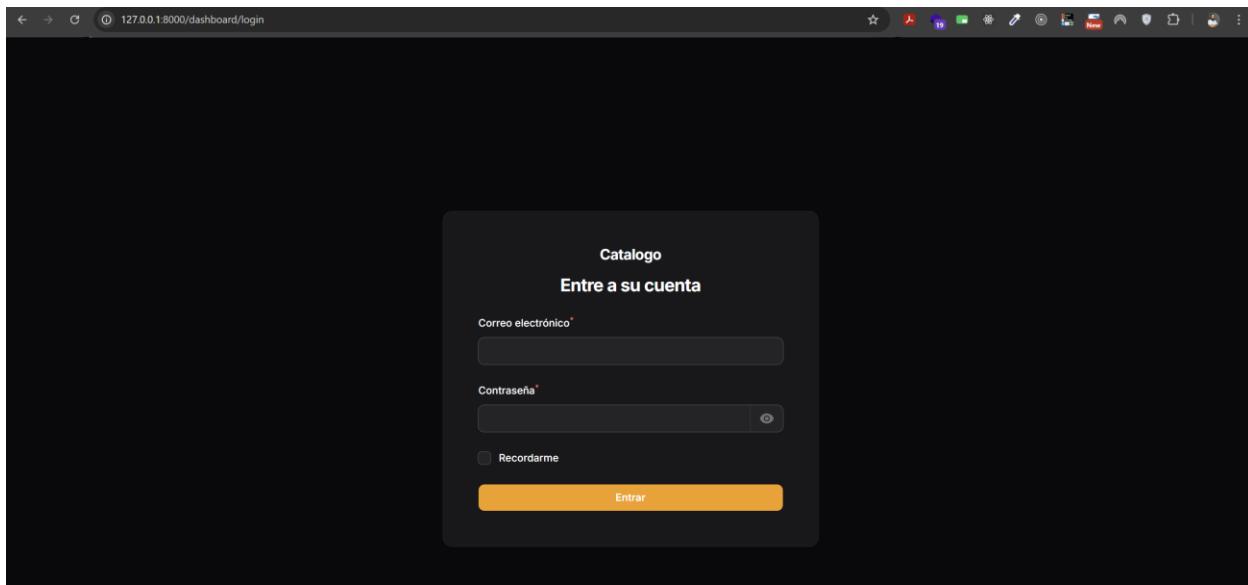
Email address:
› hdtoledo@gmail.com

Password:
›

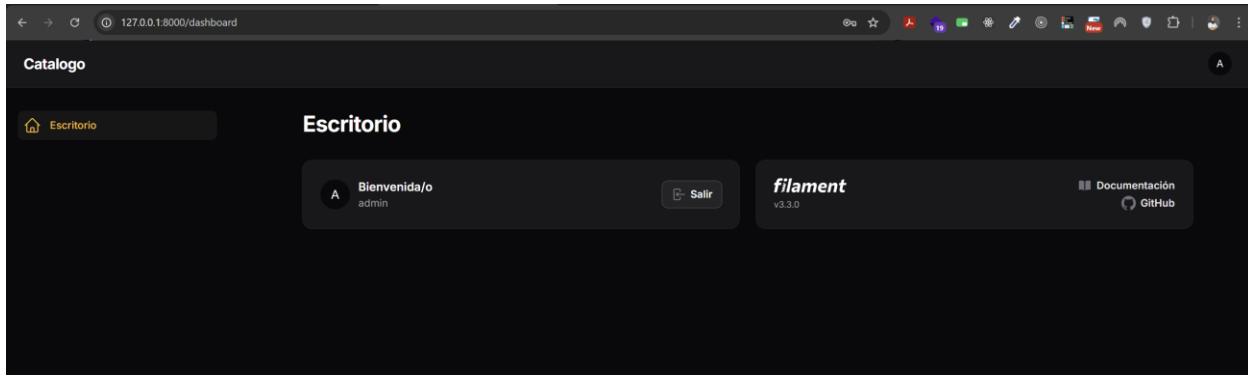
Si todo queda bien nos mostrara el aviso de que podemos iniciar sesión sobre esta ruta:

INFO Success! hdtoledo@gmail.com may now log in at <http://localhost/dashboard/login>.

Si todo esta correcto debemos ver nuestro login



Comprobamos iniciando la sesión con el correo y password y podremos ingresar a nuestro panel administrativo:



Modelos para utilizar en el catálogo

Para nuestra aplicación de catalogo vamos a hacer uso de dos modelos, uno que nos permita presentar los productos y otro categorizar, es decir : categoría y producto, vamos a tener las relaciones siguientes:

- Entidad categoría contendrá 1 o varios productos
- Productos va a estar en una o varias categorías

Para ello vamos a nuestra terminal y vamos a ejecutar el siguiente comando:

```
@ hdtoledo on livewire-filament
# php artisan make:model Category -m
```

Con este comando creamos el modelo category y realizaremos la migración de este así tendremos 2 archivos uno en models y otro en database para nuestra categoría:

```
app > Models > Category.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  0 references | 0 implementations
8  class Category extends Model
9  {
10     //
11 }
```

```
database > migrations > 2025_03_19_205818_create_categories_table.php > ...
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10      * Run the migrations.
11      */
12      public function up(): void
13      {
14          Schema::create('categories', function (Blueprint $table): void {
15              $table->id();
16              $table->timestamps();
17          });
18      }
19
20      /**
21      * Reverse the migrations.
22      */
23      public function down(): void
24      {
25          Schema::dropIfExists('categories');
26      }
27 };
```

Ahora definimos los campos que va a contener nuestra tabla para ello vamos a dejarlos así:

```
13 |     {
14 |         Schema::create('categories', function (Blueprint $table): void {
15 |             $table->id();
16 |             $table->string('name', 50);
17 |             $table->boolean('status');
18 |             $table->timestamps();
19 |         });
20 |
21 }
```

Con estos campos vamos a realizar la migración con el siguiente comando en la terminal:

```
② hdtoledo on └─ livewire-filament └─ JS
# php artisan migrate
```

Como consejo debemos hacer primero las tablas que no contienen dependencias es decir que no están ligadas con llaves foráneas de esa manera podremos hacer un orden correcto de las migraciones, ahora vamos a realizar la de Product:

```
② hdtoledo on └─ livewire-filament └─ JS
● # php artisan make:model Product -m
```

como sugerencia al crear el modelo dejarlo en singular y en inglés para no alterar como trabaja filamentphp ya que este lo pasa automáticamente a plural la tabla, ahora vamos a dejar los siguientes datos para nuestra tabla de productos:

```
12 |     public function up(): void
13 |     {
14 |         Schema::create('products', function (Blueprint $table): void {
15 |             $table->id();
16 |             $table->string('name', 100);
17 |             $table->text('description');
18 |             $table->decimal('price', 10, 2);
19 |             $table->string('image');
20 |             $table->foreignId('category_id')->constrained();
21 |
22 |         });
23 |
24 }
```

Notemos que hemos creado la llave foránea para incluir categorías, ejecutamos la migración con el siguiente comando:

```
② hdtolledo on livewire-filament
● # php artisan migrate
```

Y ahora vamos a verificar en nuestra base de datos la creación de nuestras migraciones

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None	AUTO_INCREMENT		Change Drop More
2	name	varchar(100)	utf8mb4_unicode_ci		No	None			Change Drop More
3	description	text	utf8mb4_unicode_ci		No	None			Change Drop More
4	price	decimal(10,2)			No	None			Change Drop More
5	image	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
6	category_id	bigint(20)		UNSIGNED	No	None			Change Drop More
7	created_at	timestamp			Yes	NULL			Change Drop More
8	updated_at	timestamp			Yes	NULL			Change Drop More

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	id	0	A	No	
Edit Rename Drop	products_category_id_foreign	BTREE	No	No	category_id	0	A	No	

Ahora vamos a generar la relación entre las tablas, para ello vamos a category.php y dejamos así:

```
app > Models > Category.php > PHP Intelephense > Category > products
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  0 references | 0 implementations
8  class Category extends Model
9  {
10     0 references | 0 overrides
11     public function products(): HasMany{
12         return $this->hasMany(Product::class);
13     }
14 }
```

En donde declaramos que categorías tiene relación de muchos productos y lo mismo vamos a hacer en el modelo de productos:

```
app > Models > Product.php > PHP > Product > category()
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Product extends Model
8  {
9      public function category(): BelongsTo
10     {
11         return $this->belongsTo(Category::class);
12     }
13 }
14
```

Acá relacionamos que productos pertenece a categorías y de esta manera dejamos configurado.

Ahora empezaremos a crear recursos mediante filamentphp, para ello revisaremos un poco la [documentación](#), para crear un recurso debemos ejecutar la siguiente sintaxis

```
php artisan make:filament-resource Customer
```

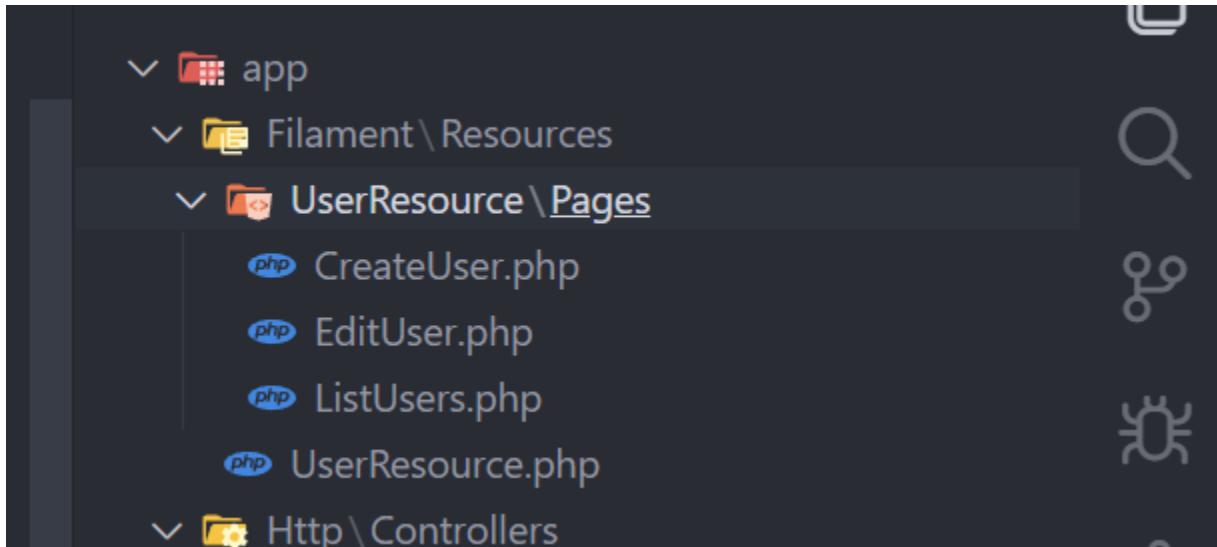
Una vez ejecutado vamos a observar como se crean en la ubicación de app/filament/Resources los diferentes archivos para poder trabajarlos

```
.
+-- CustomerResource.php
+-- CustomerResource
|   +-- Pages
|   |   +-- CreateCustomer.php
|   |   +-- EditCustomer.php
|   |   +-- ListCustomers.php
```

Vamos a proceder a crear los recursos para User con el siguiente comando

```
o ® hdtolledo on livewire-filament - n   JS
# php artisan make:filament-resource User
```

De esta manera observamos en la ubicación los diferentes archivos que nos acaba de crear:



Si nos dirigimos al navegador vamos a ver el menú de Users

Two screenshots of a web browser displaying the Filament dashboard. The top screenshot shows the main dashboard with a dark theme. It features a header with 'Catalogo' and a sidebar with 'Escritorio' and 'Users'. The main area displays a 'Bienvenida/o admin' message and navigation buttons for 'Salir', 'filament v3.0.0', 'Documentación', and 'GitHub'. The bottom screenshot shows the 'Users' list page. It has a header with 'Catalogo', 'Escritorio', and 'Users'. The main content area shows a table with two rows, a 'Crear user' button, and a search bar at the bottom. The URL in the address bar is '127.0.0.1:8000/dashboard/users'.

De esta manera observamos como ya tenemos la vista funcional, sin necesidad de insertar líneas de código, de eso se trata filament para poder agilizar la creación de estos procesos.

Notemos que en nuestra db tenemos un registro:

Showing rows 0 - 0 (1 total, Query took 0.00002 seconds.)

SELECT * FROM `users`

Profiling | [Edit inline](#) | [Edit](#) | [Explain SQL](#) | [Create PHP code](#) | [Refresh](#)

Show all | Number of rows: 25 | Filter rows: Search this table

[Extra options](#)

	Edit	Copy	Delete	Remember me	Created at	Updated at					
	id	name	email	email_verified_at	password	remember_token	created_at	updated_at			
<input type="checkbox"/>	Edit	Copy	Delete	1	admin	hdtoledo@gmail.com	NULL	\$2y\$12\$U8E9atTi54KzphZbncSP.dpNoeDVuqCgLsbFLbejyl...	NULL	2025-03-20 16:35:55	2025-03-20 16:35:55

[Check all](#) | With selected: [Edit](#) | [Copy](#) | [Delete](#) | [Export](#)

Show all | Number of rows: 25 | Filter rows: Search this table

sin embargo, este no se muestra en nuestra tabla de usuarios para ello vamos a dirigirnos a UserResource.php y vamos a agregar la siguiente línea sobre las columnas:

```
app > Filament > Resources > UserResource.php > PHP Intelephense > UserResource > table
17 class UserResource extends Resource
18 {
19     // ...
20 }
21
22     public static function table(Table $table): Table
23     {
24         return $table
25             ->columns(components: [
26                 TextColumn::make(name: 'id'),
27                 TextColumn::make(name: 'name'),
28                 TextColumn::make(name: 'email'),
29                 TextColumn::make(name: 'created_at')
30                     ->dateTime()
31                     ->label(label: 'Created At'),
32                 TextColumn::make(name: 'updated_at')
33                     ->dateTime()
34                     ->label(label: 'Updated At'),
35             ])
36             ->filters(filters: [
37                 //|
38             ])
39     }
40 }
```

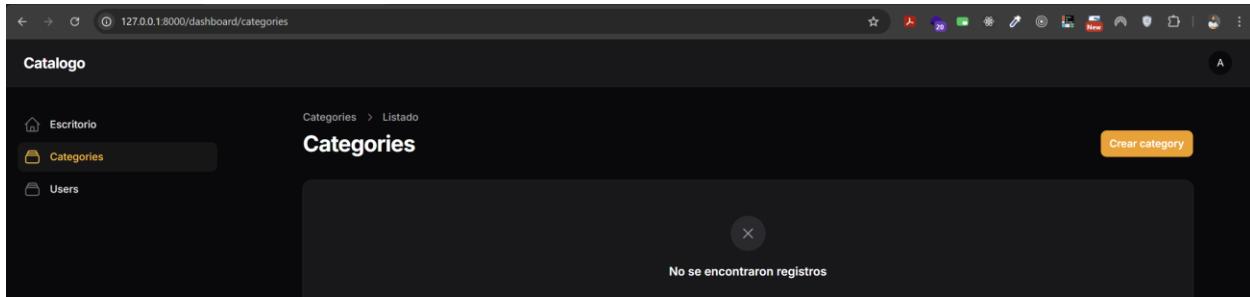
De esta manera vamos a mostrar los campos que nos interesa en nuestra tabla Users, revisemos en el navegador:

The screenshot shows a web browser window with the URL `127.0.0.1:8000/dashboard/users`. The page title is "Catalogo". On the left sidebar, there are two items: "Escritorio" and "Users", with "Users" being the active tab, indicated by a yellow background. The main content area has a breadcrumb navigation "Users > Listado". Below it, the word "Users" is displayed in large bold letters. To the right of the table, there is a yellow button labeled "Crear user". The table itself has columns: "Id", "Name", "Email", "Created At", and "Updated At". A single row is present with the following data: Id 1, Name admin, Email `hdtolledo@gmail.com`, Created At mar. 20, 2025 16:35:55, and Updated At mar. 20, 2025 16:35:55. There is also an "Editar" button next to the last column. At the bottom of the table, there is a message "Se muestra un resultado" and a pagination control "por página 10".

Vamos ahora a enfocarnos en la creación de los recursos del modelo de categoría para ello vamos a colocar el siguiente comando:

```
⑧ hdtolledo on └─ livewire-filament └─
# php artisan make:filament-resource Category
```

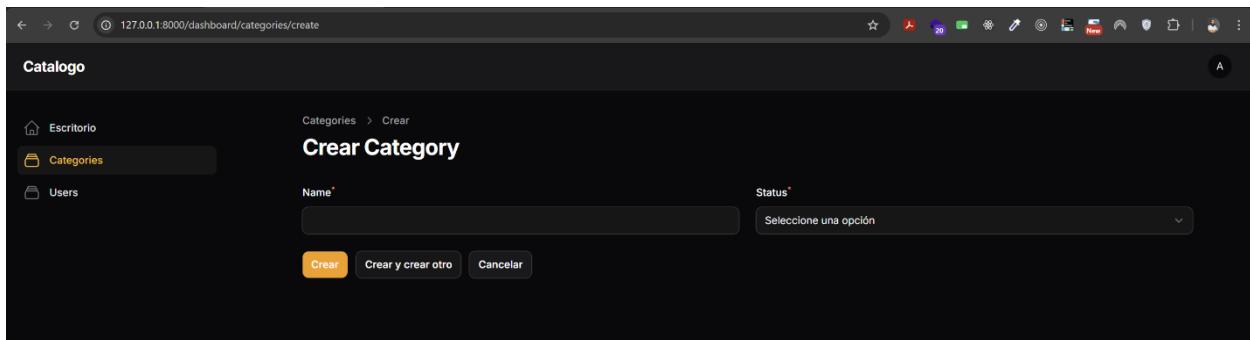
Verificamos en nuestro dashboard:



Vamos a generar los inputs que nos permitan ingresar la información de categorías, para ellos nos ubicamos en `CategoryResource.php` y vamos a trabajar sobre el método `Form` y dejamos de la siguiente manera:

```
app > Filament > Resources > CategoryResource.php > PHP Intelephense > CategoryResource > form
18 class CategoryResource extends Resource
19 {
20     public static function form(Form $form): Form
21     {
22         return $form
23             ->schema(components: [
24                 TextInput::make(name: 'name')
25                     ->required(),
26                 Select::make(name: 'status')
27                     ->required()
28                     ->options(options: [
29                         '1' => 'Activo',
30                         '0' => 'Inactivo',
31                     ])
32             ]);
33     }
34 }
```

Vamos ahora a verificar en nuestra vista del navegador en crear categoría y nos saldrá:



vamos a crear una categoría llamada Hombre:

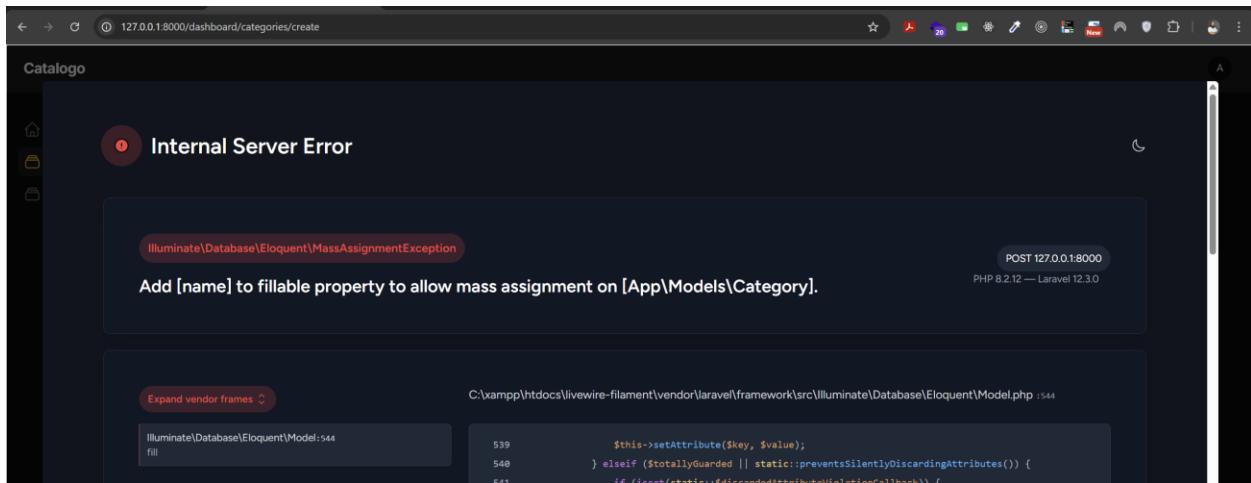
Categories > Crear

Crear Category

Name*

Status*

Y cuando le damos a crear vemos como nos genera un error, este error se debe a que aun no configuramos name y status



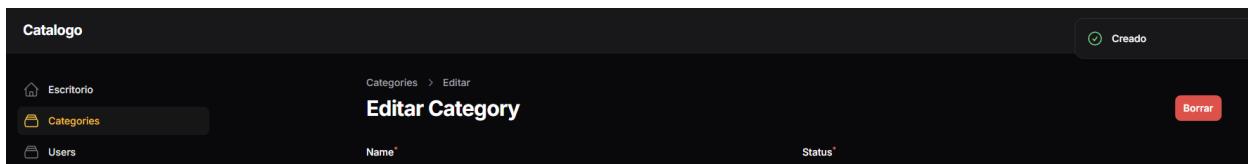
Así que vamos a dirigirnos a dentro del modelo category y allí definimos lo siguiente:

```

app > Models > Category.php > PHP Intelephense > Category
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Category extends Model
8  {
9      protected $fillable = ["name", "status"];
10     public function products(): HasMany{
11         return $this->hasMany(Product::class);
12     }
13 }
14

```

En la variable fillable pasamos los datos de name y status, ahora nuevamente procedemos a crear los datos:



Nos saldrá la notificación toast de creado en la parte superior y ahora validamos en nuestra base de datos:

	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	<input type="button" value="1"/>	<input type="button" value="Hombre"/>	<input type="button" value="1"/>	<input type="button" value="2025-03-25 02:00:54"/>	<input type="button" value="2025-03-25 02:00:54"/>
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>					

Ahora continuaremos modificando la vista para mostrar los elementos de la tabla:

```

app > Filament > Resources > CategoryResource.php > PHP > CategoryResource > table()
19   class CategoryResource extends Resource
20     public static function table(Table $table): Table
21     {
22       return $table
23         ->columns(components: [
24           TextColumn::make(name: 'name')
25             ->sortable()
26             ->searchable()
27             ->label(label: 'Nombre'),
28           TextColumn::make(name: 'status')
29             ->sortable()
30             ->searchable()
31             ->label(label: 'Estado')
32         ])
33         ->filters(filters: [
34           // ...

```

Y en nuestro navegador saldrá lo siguiente:

Ahora si observamos el estado nos esta saliendo con un 1 de valor para ello vamos a modificar y utilizar el método siguiente para cambiarlo:

```

47   ->label(label: 'Nombre'),
48   TextColumn::make(name: 'status')
49     ->sortable()
50     ->searchable()
51     ->label(label: 'Estado')
52     ->formatStateUsing(callback: function (string $state): string {
53       if ($state === '1') {
54         return 'Activo';
55       }
56     }),
57   ]
58   ->filters(filters: [
59     // ...

```

De esta manera nuestra vista quedara así:

The screenshot shows the 'Categories' list view. At the top right is a yellow button labeled 'Crear category'. Below it is a search bar with the placeholder 'Buscar'. There are two filter dropdowns: 'Nombre' set to 'Hombre' and 'Estado' set to 'Activo'. A table below shows one result: 'Se muestra un resultado'. At the bottom right of the table is a link 'Editar'.

Si notamos al momento de crear la categoría se nos esta quedando en la misma vista de creación y no nos redirige a la tabla principal, para ello vamos a crear una función que nos redirija, nos ubicamos en CreateCategory.php y hacemos lo siguiente:

```
app > Filament > Resources > CategoryResource > Pages > CreateCategory.php > PHP > CreateCategory > getRedirectUrl()
9  class CreateCategory extends CreateRecord
10 {
11     0 references
12     protected static string $resource = CategoryResource::class;
13
14     0 references | 0 overrides
15     protected function getRedirectUrl(): string
16     {
17         return $this->getResource()::getUrl('index');
18     }

```

Agreamos la función getRedirectUrl que se encargara de que al momento de terminar el proceso de creación nos lleve al index de nuestra tabla.

Ahora realizaremos una prueba más agregando otra categoría:

The screenshot shows the 'Crear Category' form. It has two input fields: 'Name*' with 'Mujer' and 'Status*' with 'Inactivo'. At the bottom are three buttons: a yellow 'Crear' button, a grey 'Crear y crear otro' button, and a grey 'Cancelar' button.

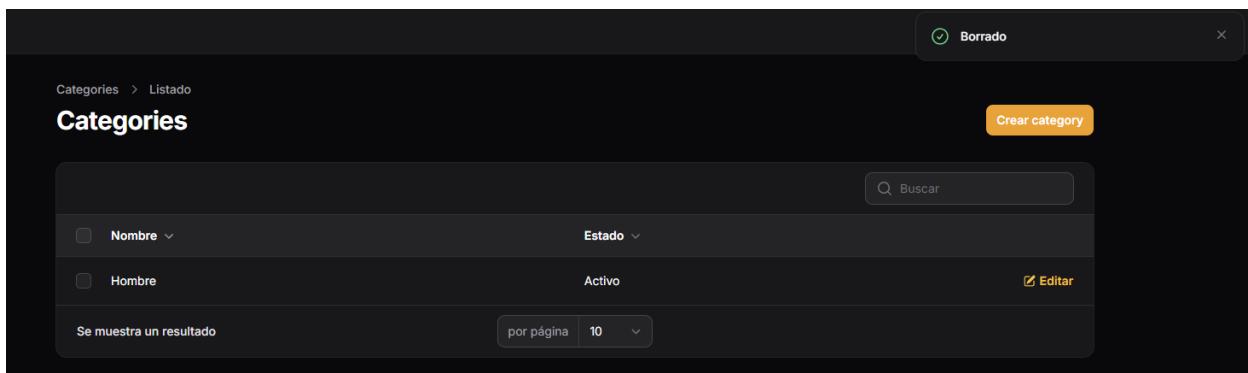
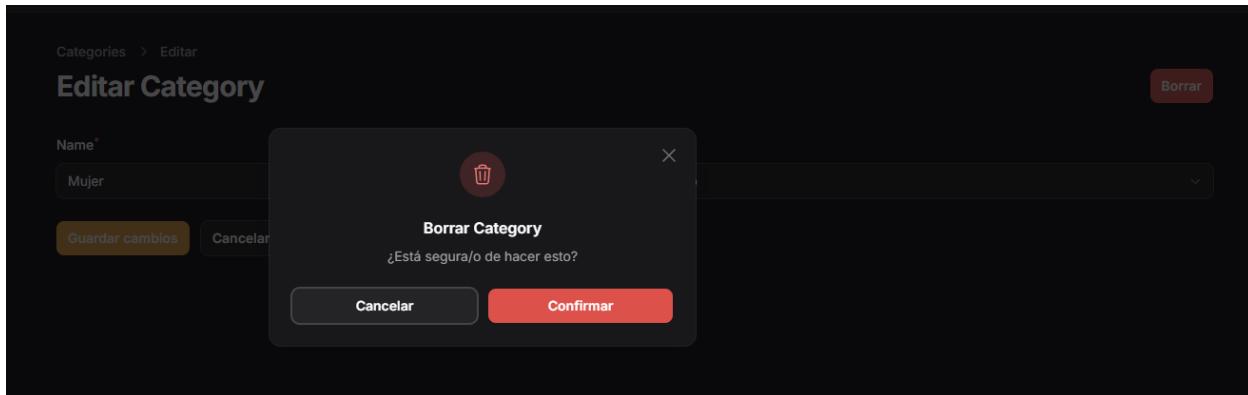
Categories > Listado		Crear category
Categories		
<input type="checkbox"/> Nombre ▾	<input type="checkbox"/> Estado ▾	
<input type="checkbox"/> Hombre	Activo	<input checked="" type="checkbox"/> Editar
<input type="checkbox"/> Mujer	Inactivo	<input checked="" type="checkbox"/> Editar
Se muestran de 1 a 2 de 2 resultados	por página	10 ▾

Editaremos el valor para activarlo:

Categories > Listado		Crear category
Categories		
<input type="checkbox"/> Nombre ▾	<input type="checkbox"/> Estado ▾	
<input type="checkbox"/> Hombre	Activo	<input checked="" type="checkbox"/> Editar
<input type="checkbox"/> Mujer	Activo	<input checked="" type="checkbox"/> Editar
Se muestran de 1 a 2 de 2 resultados	por página	10 ▾

Y por último procedemos a borrarlo para confirmar que nuestro CRUD se encuentre correctamente constituido:

Categories > Editar	
Editar Category	Borrar
Name*	Status*
<input type="text" value="Mujer"/>	<input style="width: 150px;" type="text" value="Activo"/>
Guardar cambios	Cancelar



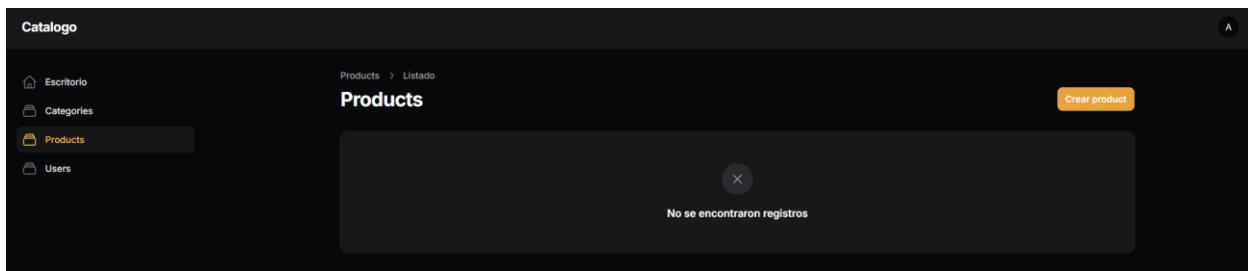
De esta manera comprobamos que todo esta correctamente funcional en nuestro CRUD de categorías.

Creación de Recursos de Filament para el modelo Productos

Vamos a crear los recursos para el modelo Productos, para ello nos ubicamos en la terminal y escribimos el siguiente comando:

```
○ ® hdtoledo on └─ livewire-filament └─
# php artisan make:filament-resource Product
```

Revisamos en nuestro navegador que este saliendo:



Ahora vamos a modificar nuestro archivo de ProductResource.php de la siguiente manera:

```
app > Filament > Resources > ProductResource.php > PHP > ProductResource > form()
20 class ProductResource extends Resource
21 {
22     public static function form(Form $form): Form
23     {
24         return $form
25             ->schema(components: [
26                 TextInput::make(name: 'name')
27                     ->label(label: 'Nombre')
28                     ->required()
29                     ->placeholder(placeholder: 'Nombre del producto')
30                     ->maxLength(length: 100),
31                 TextInput::make(name: 'description')
32                     ->label(label: 'Descripción')
33                     ->required()
34                     ->placeholder(placeholder: 'Descripción del producto')
35                     ->maxLength(length: 255),
36                 TextInput::make(name: 'price')
37                     ->label(label: 'Precio')
38                     ->required()
39                     ->placeholder(placeholder: 'Precio del producto')
40                     ->prefix(label: '$')
41                     ->numeric(),
42                 FileUpload::make(name: 'image')
43                     ->label(label: 'Imagen')
44                     ->required()
45                     ->placeholder(placeholder: 'Imagen del producto')
46                     ->image()
47                     ->directory(directory: 'products'),
48                 Select::make(name: 'category_id')
49                     ->label(label: 'Categoría del Producto')
50                     ->required()
51                     ->relationship(name: 'category', titleAttribute: 'name')
52                     ->placeholder(placeholder: 'Seleccione una categoría'),
53             ]);
54     }
55 }
56 
```

Recordemos que esto lo tenemos previamente configurado en nuestra base de datos:

Table structure

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)	utf8mb4_unicode_ci	UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(100)	utf8mb4_unicode_ci		No	None			Change Drop More
3	description	text	utf8mb4_unicode_ci		No	None			Change Drop More
4	price	decimal(10,2)			No	None			Change Drop More
5	image	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
6	category_id	bigint(20)		UNSIGNED	No	None			Change Drop More
7	created_at	timestamp			Yes	NULL			Change Drop More
8	updated_at	timestamp			Yes	NULL			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Spatial Fulltext

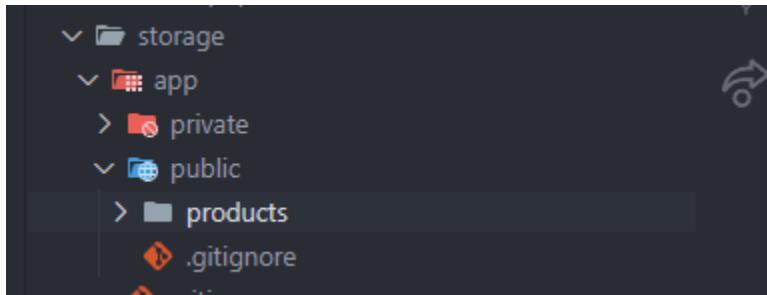
Print Propose table structure Move columns Normalize

Add 1 column(s) after updated_at

Indexes

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	id	0	A	No	
Edit Rename Drop	products_category_id_foreign	BTREE	No	No	category_id	0	A	No	

Antes de irnos a nuestra vista vamos a crear la carpeta de carga de las imágenes en la ubicación de storage/app/public/products



Ahora si vamos a nuestra vista para ver los resultados:

Catalogo

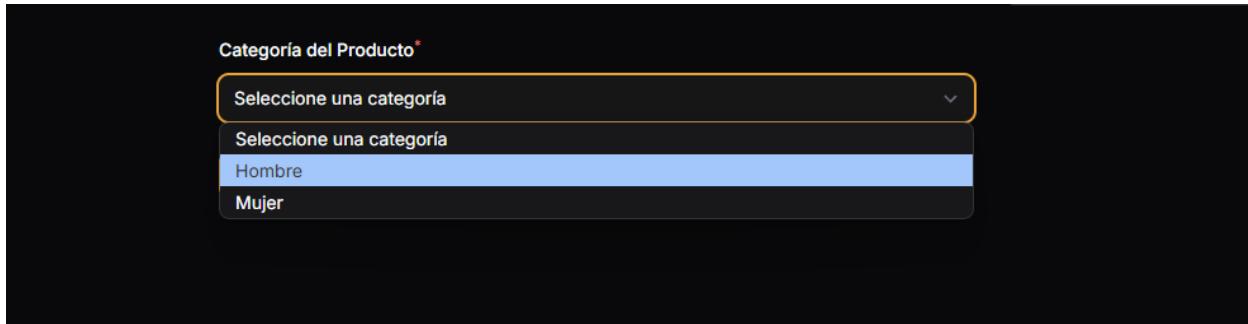
Products > Crear

Crear Product

Nombre*	Descripción*
Nombre del producto	Descripción del producto
Precio*	Imagen*
\$ Precio del producto	Imagen del producto
Categoría del Producto*	
Selección una categoría	

Create Create and create another Cancel

Si verificamos el select de categoría nos debería mostrar lo siguiente:



Ahora modificamos nuestro CreateProduct.php para redirigir una vez creado el producto al index de la tabla de productos:

```
app > Filament > Resources > ProductResource > Pages > CreateProduct.php > PHP > CreateProduct > getRedirectUrl()  
1  <?php  
2  
3  namespace App\Filament\Resources\ProductResource\Pages;  
4  
5  use App\Filament\Resources\ProductResource;  
6  use Filament\Actions;  
7  use Filament\Resources\Pages\CreateRecord;  
8  
9  class CreateProduct extends CreateRecord  
10 {  
11     protected static string $resource = ProductResource::class;  
12  
13     protected function getRedirectUrl(): string  
14     {  
15         return $this->getResource()::getUrl('index');  
16     }  
17 }  
18  
19
```

Antes de proceder a insertar un producto, vamos a modificar también el modelo del producto, /models/product.php

```
app > Models > Product.php > PHP Intelephense > Product  
1  <?php  
2  
3  namespace App\Models;  
4  
5  use Illuminate\Database\Eloquent\Model;  
6  
7  class Product extends Model  
8  {  
9      protected $fillable = ["name", "description", "price", "image", "category_id"];  
10     public function category(): BelongsTo  
11     {  
12         return $this->belongsTo(Category::class);  
13     }  
14 }  
15
```

Recordemos que pasamos a través de fillable las variables de nuestro formulario para que sean cargadas sin problema.

Y ahora si vamos a insertar información en el formulario para comprobar como esta todo:

Products > Crear

Crear Product

Nombre*
Camisa de Hombre

Descripción*
Camisa a cuadros roja

Precio*
\$ 50000

Imagen*
016517DELCUADROJO.jpg
1 MB
Subiendo 100%
toca para cancelar

Categoría del Producto*
Hombre

Subiendo archivo... Crear y crear otro Cancelar

Nos mostrara estados de color y animación de carga de la imagen y una vez terminado nos mostrara todo de la siguiente manera:

Products > Crear

Crear Product

Nombre*
Camisa de Hombre

Descripción*
Camisa a cuadros roja

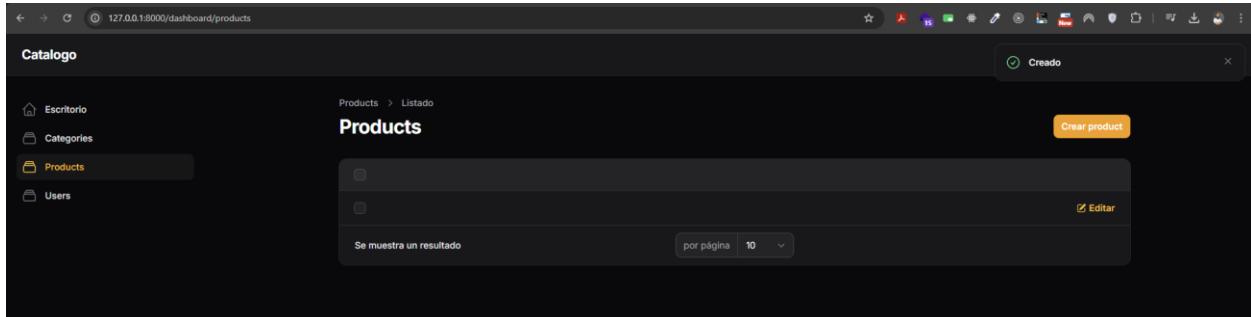
Precio*
\$ 50000

Imagen*
016517DELCUADROJO.jpg
1 MB
Subida completa
tocar para deshacer

Categoría del Producto*
Hombre

Crear Crear y crear otro Cancelar

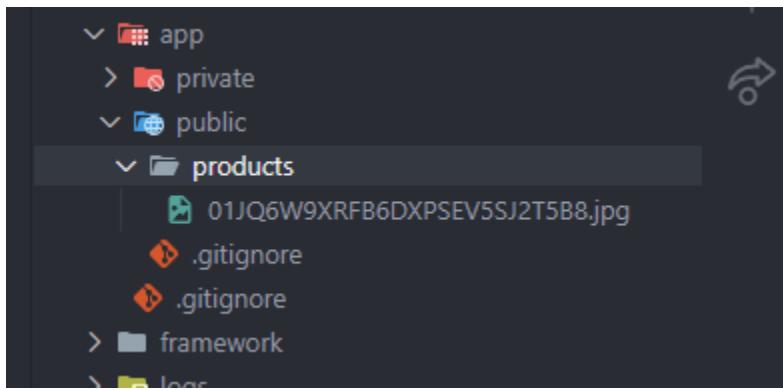
También tenemos función de drag and drop, arrastrar y soltar a la ventana del navegador para cargar la imagen, vamos a presionar el botón de crear:



Y nos debe llevar a la vista de la tabla de productos en donde actualmente no tenemos configurada la tabla, sin embargo, si revisamos la base de datos:

	<input type="checkbox"/> Edit	<input checked="" type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Edit	<input checked="" type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export	
1	Camisa de Hombre	Camisa a cuadros roja	50000.00	products/01JQ6W9XRFB6DXPSEV5SJ2T5B8.jpg		1	2025-03-25 14:46:32	2025-03-25 14:46:32

Y si revisamos también el directorio de carga de la imagen:



Observamos que todo esta cargado y funcional, ahora vamos a configurar nuestra tabla de productos, `productResource.php`:

```

app > Filament > Resources > ProductResource.php > PHP > ProductResource > table()
22 class ProductResource extends Resource
62 public static function table(Table $table): Table
63 {
64     return $table
65         ->columns(components: [
66             TextColumn::make(name: 'name')
67                 ->label(label: 'Nombre')
68                 ->searchable()
69                 ->sortable(),
70             TextColumn::make(name: 'description')
71                 ->label(label: 'Descripción')
72                 ->searchable()
73                 ->sortable(),
74             TextColumn::make(name: 'price')
75                 ->label(label: 'Precio')
76                 ->searchable()
77                 ->prefix(prefix: '$')
78                 ->formatStateUsing(callback: function (string $state): string {
79                     return number_format(num: $state, decimals: 2, decimal_separator: ',', thousands_separator: '.');
80                 })
81                 ->sortable(),
82             TextColumn::make(name: 'category.name')
83                 ->label(label: 'Categoria')
84                 ->searchable()
85                 ->sortable(),
86             ImageColumn::make(name: 'image')
87                 ->label(label: 'Imagen')
88                 ->sortable(),
89         ])
90         ->filters(filters: [
91             // ...
92         ])
93         ->actions(actions: [
94             Tables\Actions>EditAction::make(),

```

Vamos a linkear también nuestro storage para que cuando la imagen se muestre no tengamos problemas en la visualización con el siguiente comando en la terminal:

```

○ @ hdtoledo on livewire-filament
# php artisan storage:link

```

De esta manera vamos a permitir el acceso a esta carpeta, también asegúremos de que la APP_URL de nuestro archivo de entorno de variables este apuntando a la dirección correcta del localhost:

```

.env
1 APP_NAME=Catalogo
2 APP_ENV=local
3 APP_KEY=base64:QPoc+Vnyp07m2UiUqGVRFt00Npwa956hoXV1SdwIf7M=
4 APP_DEBUG=true
5 APP_URL=http://127.0.0.1:8000
6
7
8 APP_LOCALE=es

```

Una vez verificados estos pasos vamos a revisar la vista en nuestro navegador:

Catalogo

Products > Listado

Products

Crear producto

Nombre	Descripción	Precio	Categoría	Imagen
Camisa de Hombre	Camisa a cuadros roja	\$50.000,00	Hombre	

Buscar

Se muestra un resultado

por página 10

Así obtenemos una visualización correcta de los datos que tenemos almacenados en nuestra base de datos, ahora revisamos que editar este correcto y nos guarde la información:

Products > Editar

Editar Product

Borrar

Nombre*

Camisa de Hombre

Descripción*

Camisa a cuadros roja

Precio*

\$ 50000,00

Imagen*

Categoría del Producto*

Hombre

Guardar cambios Cancelar

Presentar los productos en el Frontend de la App

Para esta parte lo que necesitamos pasar a la vista es una colección de datos a la vista, es decir pasar los productos que tenemos en nuestra base de datos, para ello vamos a ubicarnos en nuestro HomeController.php y vamos a hacer lo siguiente:

```
app > Http > Controllers > HomeController.php > PHP Intelephense > HomeController > index
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Product;
6  use Illuminate\Http\Request;
7
8  class HomeController extends Controller
9  {
10     public function index(): View
11     {
12         $products = Product::all();
13         return view('app.frontend.index', compact('products'));
14     }
15 }
16
```

Ahora vamos a modificar nuestro template-frontend.blade.php para indicarle que la sección de búsqueda y productos sea parte del contenido dinámico, así que nos vamos a ubicar sobre el catalogo y vamos a modificar de la siguiente manera:

```
resources > views > layouts > template-frontend.blade.php > html > body
2  <html lang="es">
10 <body>
44     </section>
45
46     @yield('contenido')
47
48     <!-- Catálogo de productos -->
49     <section class="catalog">...
82     </section>
83
84     <!-- Footer -->
```

Utilizamos la directiva de Blade de yield para llamar nuestro contenido, y vamos a mover la section catalog para nuestro index.blade.php

```

resources > views > app > frontend > index.blade.php
1  @extends('layouts.template-frontend')
2
3  @section('contenido')
4
5  @endsection

```

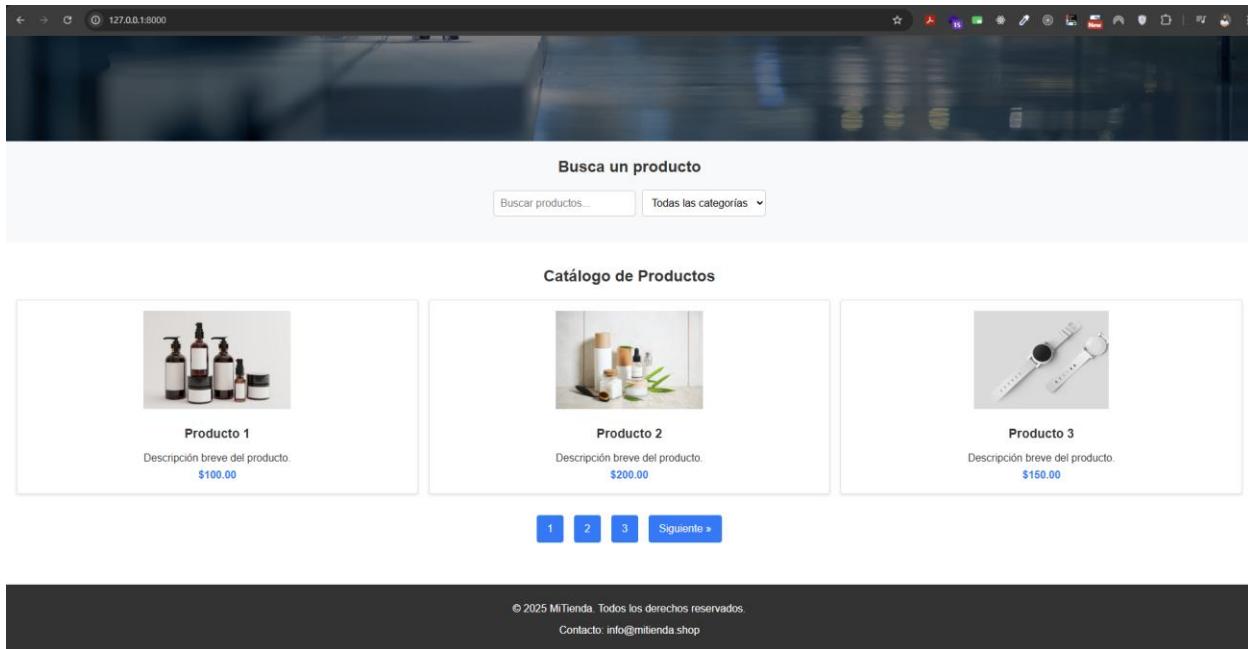
Creamos la section de contenido y dentro pegamos el catalog

```

resources > views > app > frontend > index.blade.php > section.catalog
1  @extends('layouts.template-frontend')
2
3  @section('contenido')
4
5      <!-- Catálogo de productos -->
6      <section class="catalog">
7          <h2>Catálogo de Productos</h2>
8          <div class="products">
9              <!-- Producto 1 -->
10             <div class="product-card">
11                 
12                 <h3>Producto 1</h3>
13                 <p>Descripción breve del producto.</p>
14                 <p class="price">$100.00</p>
15             </div>
16             <!-- Producto 2 -->
17             <div class="product-card">
18                 
19                 <h3>Producto 2</h3>
20                 <p>Descripción breve del producto.</p>
21                 <p class="price">$200.00</p>
22             </div>
23             <!-- Producto 3 -->
24             <div class="product-card">
25                 
26                 <h3>Producto 3</h3>
27                 <p>Descripción breve del producto.</p>
28                 <p class="price">$150.00</p>
29             </div>
30             <!-- Más productos aquí -->
31         </div>
32         <!-- Paginación -->
33         <div class="pagination">
34             <a href="#" class="page">1</a>
35             <a href="#" class="page">2</a>
36             <a href="#" class="page">3</a>
37             <a href="#" class="page">Siguiente &raquo;</a>
38         </div>
39     </section>
40
41  @endsection

```

Si todo va bien vamos a verificar que el contenido este aun saliendo en nuestro frente:



Ahora nuestra sección de catálogo de productos la vamos a hacer que funcione de acuerdo con lo que tenemos en nuestra base de datos, vamos a dejar modificando nuestro contenido de la siguiente manera:

```

resources > views > app > frontend > index.blade.php > section.catalog > div.products > div.product-card > p
1  @extends('layouts.template-frontend')
2
3  @section('contenido')
4
5  <!-- Catálogo de productos -->
6  <section class="catalog">
7      <h2>Catálogo de Productos</h2>
8      <div class="products">
9          <!-- Producto 1 -->
10         <div class="product-card">
11             
12             <h3>Producto 1</h3>
13             <p>Descripción breve del producto.</p>
14             <p class="price">$100.00</p>
15         </div>
16
17     </div>
18     <!-- Página -->
19     <div class="pagination">
20         <a href="#" class="page">1</a>
21         <a href="#" class="page">2</a>
22         <a href="#" class="page">3</a>
23         <a href="#" class="page">Siguiente &raquo;</a>
24     </div>
25
26
27 @endsection

```

Ya que vamos a dejar un ciclo para que recorra la colección de datos de productos, si vamos a verificar en nuestro frente nos debe salir la cantidad de productos que tenemos en la base de datos, obviamente sin mostrar detalles de los productos:

```

resources > views > app > frontend > index.blade.php > section.catalog > div.products
1 @extends('layouts.template-frontend')
2
3 @section('contenido')
4
5     
6     <section class="catalog">
7         <h2>Catálogo de Productos</h2>
8         <div class="products">
9             
10            ...
11            @foreach ($products as $product)
12                <div class="product-card">
13                    
14                    <h3>Producto 1</h3>
15                    <p>Descripción breve del producto.</p>
16                    <p class="price">$100.00</p>
17                </div>
18            @endforeach
19
20
21

```

The screenshot shows a web application running on a local server at 127.0.0.1:8000. The interface includes a header with a search bar and a dropdown for categories. The main content area is titled "Catálogo de Productos" and displays six product cards. Each card contains a placeholder image of several bottles, the text "Producto 1", a brief description, the price "\$100.00", and a "Siguiente »" button.

Imagen	Título	Descripción	Precio
	Producto 1	Descripción breve del producto.	\$100.00
	Producto 1	Descripción breve del producto.	\$100.00
	Producto 1	Descripción breve del producto.	\$100.00
	Producto 1	Descripción breve del producto.	\$100.00
	Producto 1	Descripción breve del producto.	\$100.00
	Producto 1	Descripción breve del producto.	\$100.00

En mi caso tengo ya registrados 6 productos:

Products > Listado

Products

[Crear producto](#)

<input type="checkbox"/>	Nombre	Descripción	Precio	Categoría	Imagen	Editar
<input type="checkbox"/>	Camisa de Hombre	Camisa a cuadros roja	\$50.000,00	Hombre		Editar
<input type="checkbox"/>	Camisa de Mujer	Camisa Jean Mujer	\$50.000,00	Mujer		Editar
<input type="checkbox"/>	Jean Hombre	Jean Hombre	\$80.000,00	Hombre		Editar
<input type="checkbox"/>	Jean Mujer	Jean Mujer	\$120.000,00	Mujer		Editar
<input type="checkbox"/>	Short Hombre	Short Hombre Jean	\$120.000,00	Hombre		Editar
<input type="checkbox"/>	Short Mujer	Short Mujer	\$120.000,00	Mujer		Editar

Se muestran de 1 a 6 de 6 resultados

por página

SELECT * FROM `products`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

[Extra options](#)

<input type="checkbox"/>	Edit	Copy	Delete	id	name	description	price	image	category_id	created_at	updated_at
<input type="checkbox"/>	Edit	Copy	Delete	2	Camisa de Hombre	Camisa a cuadros roja	50000.00		1	2025-03-25 15:09:27	2025-03-25 15:28:18
<input type="checkbox"/>	Edit	Copy	Delete	3	Camisa de Mujer	Camisa Jean Mujer	50000.00		3	2025-03-25 15:52:55	2025-03-25 15:52:55
<input type="checkbox"/>	Edit	Copy	Delete	4	Jean Hombre	Jean Hombre	80000.00		1	2025-03-25 15:53:40	2025-03-25 15:53:40
<input type="checkbox"/>	Edit	Copy	Delete	5	Jean Mujer	Jean Mujer	120000.00		3	2025-03-25 15:54:22	2025-03-25 15:54:22
<input type="checkbox"/>	Edit	Copy	Delete	6	Short Hombre	Short Hombre Jean	120000.00		1	2025-03-25 15:55:35	2025-03-25 15:55:35
<input type="checkbox"/>	Edit	Copy	Delete	7	Short Mujer	Short Mujer	120000.00		3	2025-03-25 15:56:14	2025-03-25 15:56:14

[Check all](#) [With selected:](#) [Edit](#) [Copy](#) [Delete](#) [Export](#)

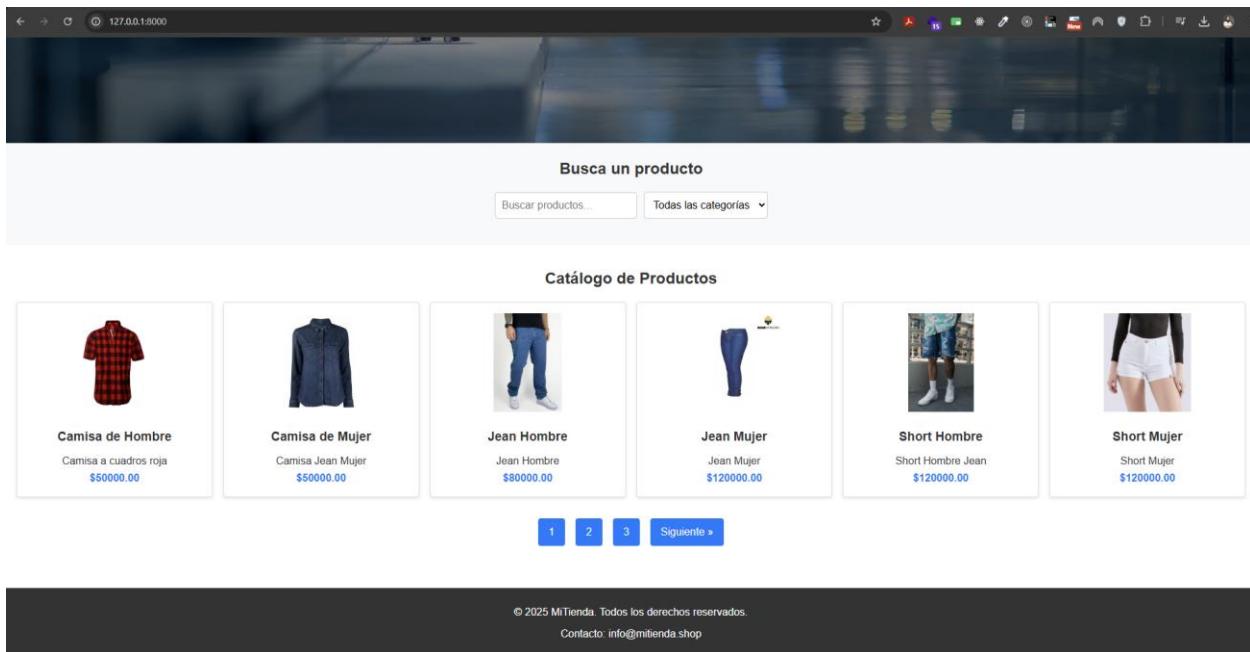
Ahora vamos a empezar a llamar la información de los campos que necesitamos de la siguiente manera:

```

resources > views > app > frontend > index.blade.php > section.catalog > div.products
1  @extends('layouts.template-frontend')
2
3  @section('contenido')
4
5      
6      <section class="catalog">
7          <h2>Catálogo de Productos</h2>
8          <div class="products">
9              
10
11             @foreach ($products as $product )
12                 <div class="product-card">
13                     name }}>
14                     <h3>{{ $product->name }}</h3>
15                     <p>{{ $product->description }}</p>
16                     <p class="price">${{ $product->price }}</p>
17                 </div>
18             @endforeach
19
20
21         </div>
22         
23         <div class="pagination">
24             <a href="#" class="page">1</a>
25             <a href="#" class="page">2</a>
26             <a href="#" class="page">3</a>
27             <a href="#" class="page">Siguiente &raquo;</a>
28         </div>
29     </section>
30
31
32     @endsection

```

Y si vamos a nuestro frontend:

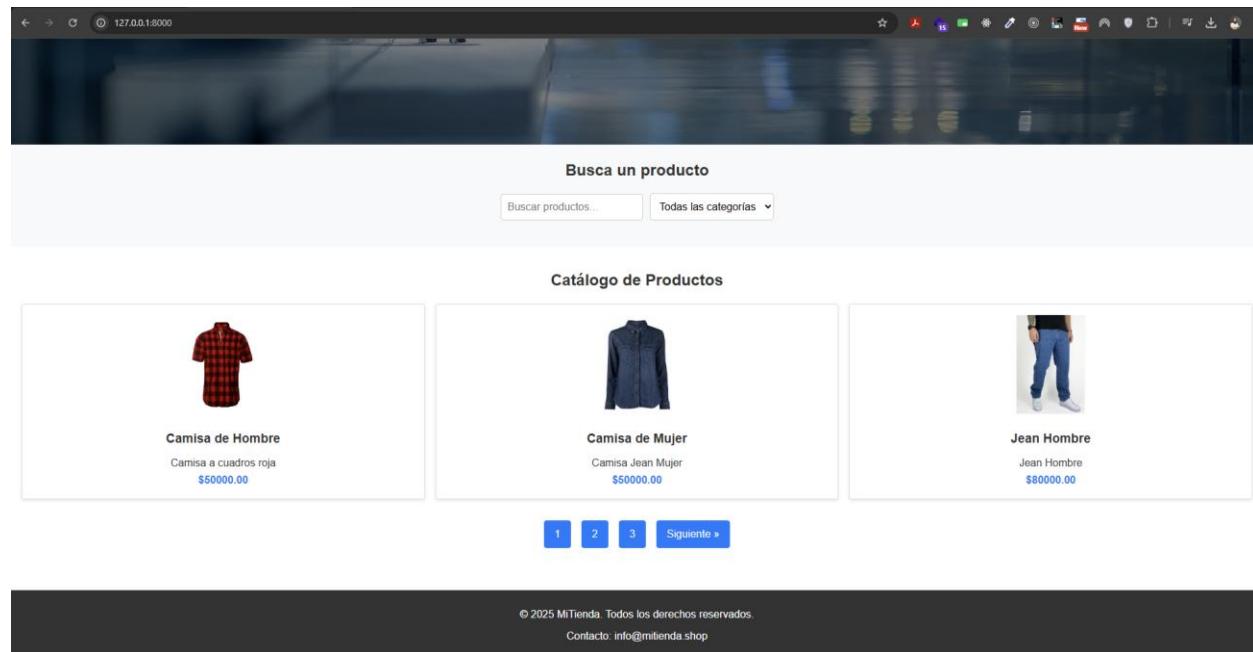


De esta manera estamos trayendo la información de nuestra base de datos.

Ahora vamos a modificar nuestro HomeController.php para que podamos paginar la información de nuestra colección, hacemos lo siguiente:

```
app > Http > Controllers > HomeController.php > PHP > HomeController > index()  
1  <?php  
2  
3  namespace App\Http\Controllers;  
4  
5  use App\Models\Product;  
6  use Illuminate\Http\Request;  
7  
8  class HomeController extends Controller  
9  {  
10     1 reference | 0 overrides  
11     public function index(): View  
12         $products = Product::paginate(3);  
13         return view('app.frontend.index', compact(var_name: 'products'));  
14  
15 }  
16
```

Aquí usaremos paginate para que nos pague la información de los datos a un total máximo de 3, si vamos al frente observamos lo siguiente:

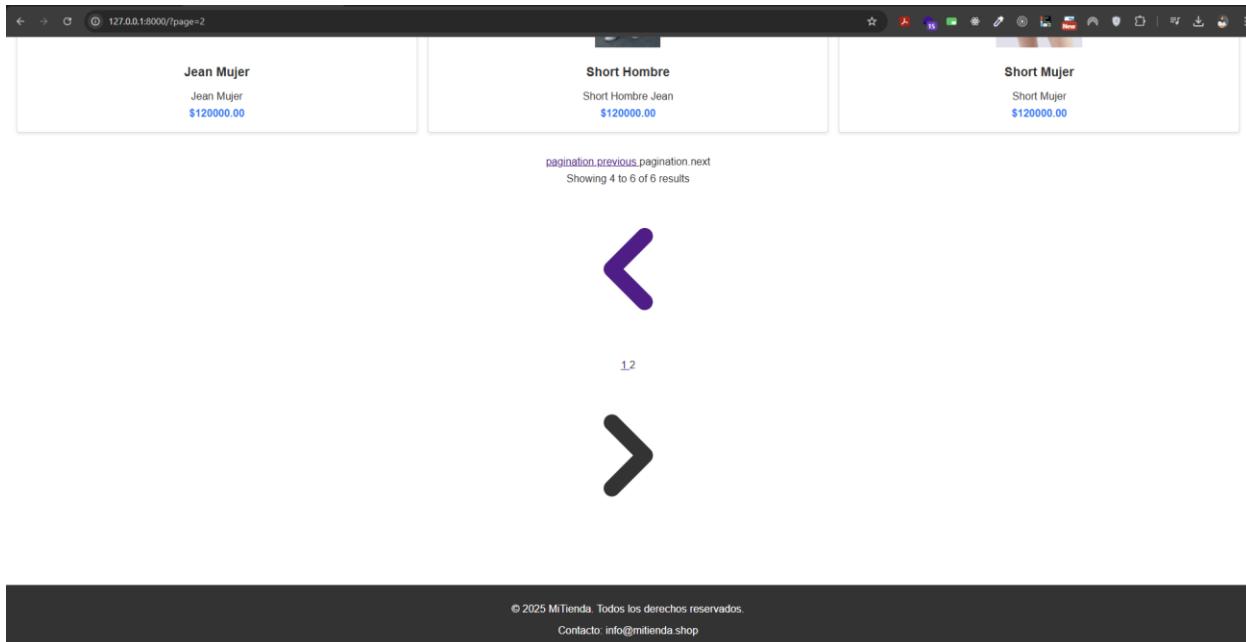


De esta manera se nos ha paginado los productos.

Ahora vamos a modificar nuestra paginación en index.blade.php de la siguiente manera:

```
resources > views > app > frontend > index.blade.php > section.catalog > div.pagination
6   <section class="catalog">
8     <div class="products">
<!--
11   </div>
12   <!-- Paginación -->
13   <div class="pagination">
14     {{-- <a href="#" class="page">1</a>
15     <a href="#" class="page">2</a>
16     <a href="#" class="page">3</a>
17     <a href="#" class="page">Siguiente &raquo; --}}
18
19     {{ $products->links() }}
20   </div>
21 </section>
22
23 @endsection
```

Esto nos mostrara lo siguiente en el frente:



unos iconos super grandes que nos permitirán ser funcionales, pero no estarán de acuerdo a la interfaz que tenemos.

Para dejar estos iconos y apariencia completamente funcional vamos a crear en la ubicación de resources/views/vendor/pagination/custom.blade.php un archivo de paginación personalizada:

template-frontend.blade.php

- vendor\pagination
- custom.blade.php
- welcome.blade.php

```

resources > views > vendor > pagination > custom.blade.php > ...
1 @if ($paginator->hasPages())
2     <div class="pagination">
3         {{-- Botón Anterior --}}
4         @if ($paginator->onFirstPage())
5             <span class="page disabled">&lquo; Anterior</span>
6         @else
7             <a href="{{ $paginator->previousPageUrl() }}" class="page">&lquo; Anterior</a>
8         @endif
9
10        {{-- Números de página --}}
11        @foreach ($elements as $element)
12            @if (is_string(value: $element))
13                <span class="page disabled">{{ $element }}</span>
14            @endif
15
16            @if (is_array(value: $element))
17                @foreach ($element as $page => $url)
18                    @if ($page == $paginator->currentPage())
19                        <span class="page active">{{ $page }}</span>
20                    @else
21                        <a href="{{ $url }}" class="page">{{ $page }}</a>
22                    @endif
23                @endforeach
24            @endif
25        @endforeach
26
27        {{-- Botón Siguiente --}}
28        @if ($paginator->hasMorePages())
29            <a href="{{ $paginator->nextPageUrl() }}" class="page">Siguiente &raqo;</a>
30        @else
31            <span class="page disabled">Siguiente &raqo;</span>
32        @endif
33    </div>
34 @endif
35

```

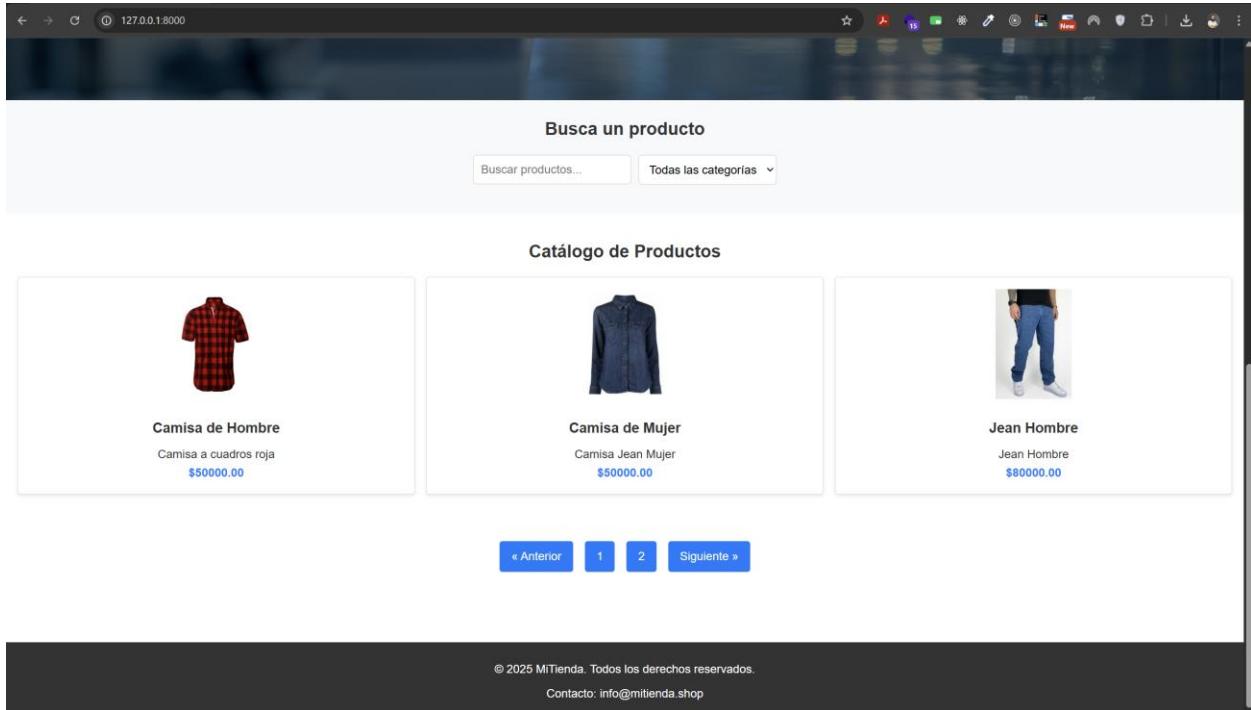
Y vamos a dejarlo de la siguiente manera en nuestro index.blade.php

```

resources > views > app > frontend > index.blade.php > section.catalog > div.pagination
6     <section class="catalog">
7         </div>
8         <!-- Página -->
9         <div class="pagination">
10            {{ $products->links('vendor.pagination.custom') }}
11        </div>
12    </section>
13 @endsection

```

De esta manera pasamos la personalización de los enlaces de paginación, si vamos al frente vamos a observar lo siguiente:



Implementación de Livewire para Búsqueda por criterio y Categoría

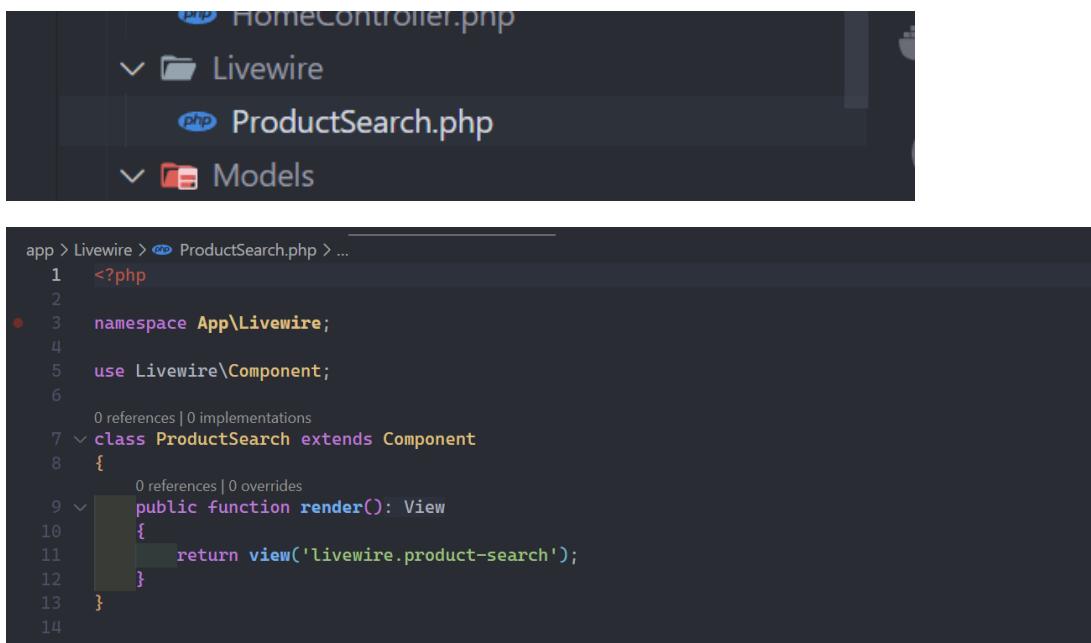
Vamos a generar las búsquedas sin necesidad de recargar la pagina y que nos muestren los resultados directamente, para ello vamos a nuestro terminal y vamos a colocar el siguiente comando:

```
○ @ hdtoledo on └─ livewire-filament └─
# php artisan make:livewire ProductSearch
```

Esto creara el componente de búsqueda a través de livewire

```
● @ hdtoledo on └─ livewire-filament └─
  # php artisan make:livewire ProductSearch
  COMPONENT CREATED 🌟
  CLASS: app/Livewire/ProductSearch.php
  VIEW: C:\xampp\htdocs\livewire-filament\resources\views\livewire\product-search.blade.php
```

Ahora dentro de nuestra carpeta app/http/livewire se creo el componente



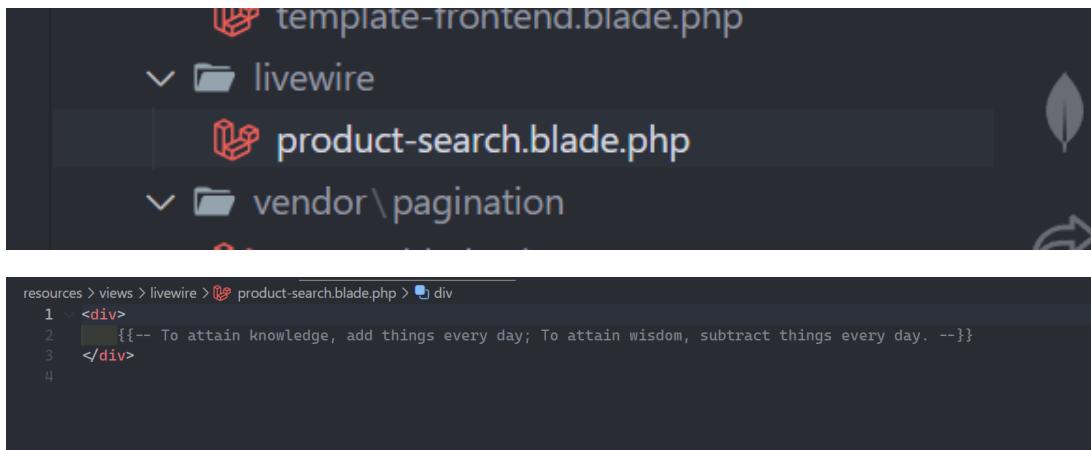
The screenshot shows a code editor with the following file structure:

```
app > Livewire > ProductSearch.php > ...
```

Content of ProductSearch.php:

```
1 <?php
2
3 namespace App\Livewire;
4
5 use Livewire\Component;
6
7 class ProductSearch extends Component
8 {
9     public function render(): View
10    {
11        return view('livewire.product-search');
12    }
13 }
14
```

También tendremos la vista que es la que estamos observando en la renderización



The screenshot shows a code editor with the following file structure:

```
resources > views > livewire > product-search.blade.php > div
```

Content of product-search.blade.php:

```
1 <div>
2 {{-- To attain knowledge, add things every day; To attain wisdom, subtract things every day. --}}
3 </div>
4
```

En esta instancia vamos a configurar la vista que queremos que se muestre directamente, para ello vamos a crear un Query que nos permita traer las búsquedas con los criterios, vamos a empezar a crear un método para inicializar la información para ello nos ubicamos en nuestro ProductSearch.php

```

app > Livewire > ProductSearch.php > PHP Intelephense > ProductSearch > mount
1  <?php
2
3  namespace App\Livewire;
4
5  use App\Models\Category;
6  use Livewire\Component;
7
8  0 references | 0 implementations
9  class ProductSearch extends Component
10 {
11     1 reference
12     public $categories;
13
14     0 references | 0 overrides
15     public function mount(): void
16     {
17         $this->categories = Category::all();
18     }
19
20     0 references | 0 overrides
21     public function render(): View
22     {
23         return view('livewire.product-search');
24     }
25
26 }

```

Ahora vamos a ubicar la sección que nos permitirá trabajar con el montaje de los datos en este caso recordemos que tenemos en template-frontend.blade.php la búsqueda

```

resources > views > layouts > template-frontend.blade.php > html > body > section#productos.search-section
2  <html lang="es">
3  <body>
4      <section class="search-section" id="productos">
5          <h2>Busca un producto</h2>
6          <form class="search-form">
7              <input type="text" placeholder="Buscar productos ..." class="search-input">
8
9              <select class="search-select" wire:model.live="category">
10                 <option value="">Todas las categorías</option>
11
12             </select>
13         </form>
14     </section>
15

```

Lo vamos a cortar y lo vamos a pegar en producto-search.blade.php

```

resources > views > livewire > product-search.blade.php > div > section#productos.search-section
1  <div>
2      <!-- Sección de búsqueda -->
3      <section class="search-section" id="productos">
4          <h2>Busca un producto</h2>
5          <form class="search-form">
6              <input type="text" placeholder="Buscar productos ..." class="search-input">
7
8              <select class="search-select" wire:model.live="category">
9                  <option value="">Todas las categorías</option>
10
11             </select>
12         </form>
13     </section>
14 </div>
15

```

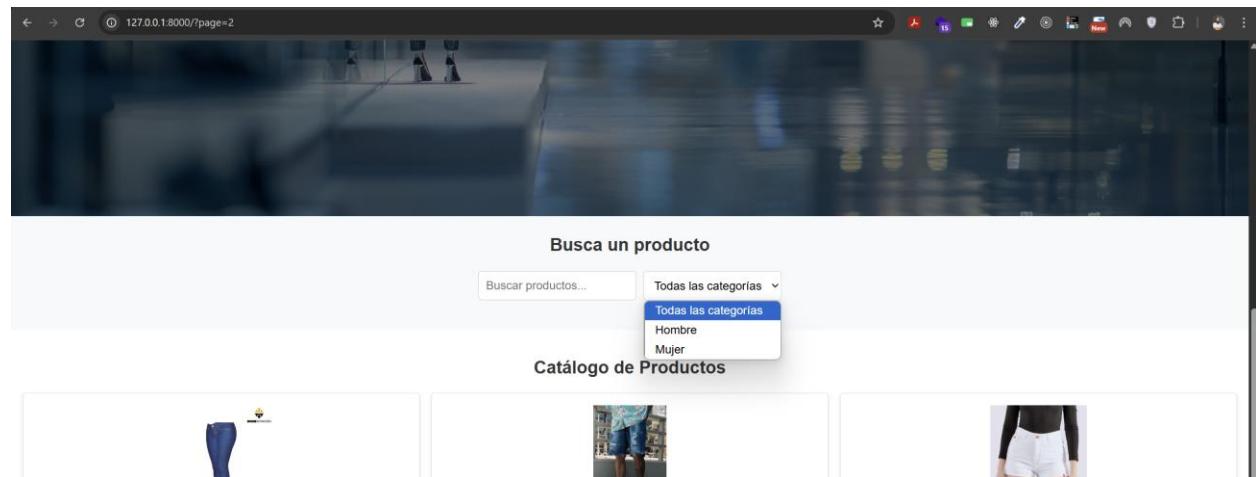
Ahora definimos la variable para cargar el contenido de categorías de la siguiente manera:

```
resources > views > livewire > product-search.blade.php > div > section#productos.search-section > form.search-form > select.search-select
1  <div>
2    <!-- Sección de búsqueda -->
3    <section class="search-section" id="productos">
4      <h2>Busca un producto</h2>
5      <form class="search-form">
6        <input type="text" placeholder="Buscar productos ..." class="search-input">
7        <select class="search-select" wire:model.live="category">
8          <option value="">Todas las categorías</option>
9          @foreach ($categories as $category)
10            <option value="{{ $category->id }}>{{ $category->name }}</option>
11          @endforeach
12        </select>
13      </form>
14    </section>
15  </div>
16
17
18
```

Y vamos a llamarlo también en nuestro index.blade.php

```
resources > views > app > frontend > index.blade.php > livewire:product-search
1  @extends('layouts.template-frontend')
2
3  @section('contenido')
4
5    <!-- Sección de búsqueda -->
6    <livewire:product-search>
7
8    <!-- Catálogo de productos -->
9    <section class="catalog">
10      <h2>Catálogo de Productos</h2>
11      <div class="products">
```

De esta manera lo llamamos y si vamos al frente vamos a observar que ya nos sale correctamente con las categorías:



Ahora vamos al componente de productsearch.php para crear dos propiedades que nos permitirán los criterios:

```
app > Livewire > ProductSearch.php > PHP > ProductSearch
1  <?php
2
3  namespace App\Livewire;
4
5  use App\Models\Category;
6  use Livewire\Component;
7
8      0 references | 0 implementations
9  class ProductSearch extends Component
10 {
11     0 references
12     public $query = "";
13     0 references
14     public $category = "";
15     1 reference
16     public $categories;
17
18     0 references | 0 overrides
19     public function mount(): void
20     {
21         $this->categories = Category::all();
22     }
23 }
```

Dentro de Query y category vamos a almacenar los valores de las búsquedas, recordemos que actualmente ya tenemos una vista que esta trayendo los productos en nuestro HomeController.php, así que vamos a modificar esta parte para hacerlo de una manera lógica y eficiente, nos ubicamos en nuestro HomeController.php y lo dejamos así:

```
app > Http > Controllers > HomeController.php > PHP Intelephense > HomeController > index
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Product;
6  use Illuminate\Http\Request;
7
8      2 references | 0 implementations
9  class HomeController extends Controller
10 {
11     1 reference | 0 overrides
12     public function index(): View
13     {
14         return view('app.frontend.index');
15     }
16 }
```

Ahora vamos a cortar la sección de catalogo de index.blade.php ya que debemos ubicarlo en producto-search.blade.php

```
resources > views > app > frontend > 🗂 index.blade.php > 🔍 livewire:product-search
1 @extends('layouts.template-frontend')
2
3 @section('contenido')
4
5     
6     <livewire:product-search>
7
8 @endsection
```

```
resources > views > livewire > 🗂 product-search.blade.php > 🔍 div > 🔍 section.catalog > 🔍 div.products
1 <div>
2     
3     <section class="search-section" id="productos">
4         <h2>Busca un producto</h2>
5         <form class="search-form">
6             <input type="text" placeholder="Buscar productos ..." class="search-input">
7
8             <select class="search-select" wire:model.live="category">
9                 <option value="">Todas las categorías</option>
10                @foreach ($categories as $category)
11                    <option value="{{ $category->id }}>{{ $category->name }}</option>
12                @endforeach
13            </select>
14        </form>
15    </section>
16
17    
18    <section class="catalog">
19        <h2>Catálogo de Productos</h2>
20        <div class="products">
21            
22
23            @foreach ($products as $product )
24            <div class="product-card">
25                name }}>
26                <h3>{{ $product->name }}</h3>
27                <p>{{ $product->description }}</p>
28                <p class="price">${{ $product->price }}</p>
29            </div>
30            @endforeach
31
32        </div>
33        
34        <div class="pagination">
35            {{ $products->links('vendor.pagination.custom') }}
36
37        </div>
38    </section>
39
40 </div>
```

Ahora para que nos funcione y podamos pasar los datos a través de product, nos ubicamos en ProductSearch.php y hacemos lo siguiente:

```
app > Livewire > ProductSearch.php > ...
1  <?php
2
3  namespace App\Livewire;
4
5  use App\Models\Category;
6  use App\Models\Product;
7  use Livewire\Component;
8
9  0 references | 0 implementations
10 class ProductSearch extends Component
11 {
12     2 references
13     public $query = "";
14     2 references
15     public $category = "";
16     1 reference
17     public $categories;
18
19     0 references | 0 overrides
20     public function mount(): void
21     {
22         $this->categories = Category::all();
23     }
24
25     0 references | 0 overrides
26     public function render(): View
27     {
28         $products = Product::query()
29             ->when($this->query, function (Builder<Product> $query): void {
30                 $query->where('name', 'like', '%' . $this->query . '%');
31             })
32             ->when($this->category, function (Builder<Product> $query): void {
33                 $query->where('category_id', $this->category);
34             })
35             ->paginate(3);
36         return view('livewire.product-search', compact(var_name: 'products'));
37     }
38 }
```

Acá estamos pasando los criterios de búsqueda a través de livewire en donde nos permitirá buscar por producto y categoría, además pasamos la paginación, si vamos a nuestro frente:

127.0.0.1:8000/?page=1

Busca un producto

Buscar productos... Todas las categorías

Catálogo de Productos

 Camisa de Hombre
Camisa a cuadros roja
[\\$50000.00](#)

 Camisa de Mujer
Camisa Jean Mujer
[\\$50000.00](#)

 Jean Hombre
Jean Hombre
[\\$80000.00](#)

« Anterior 1 2 Siguiente »

© 2025 MiTienda. Todos los derechos reservados.
Contacto: info@mitienda.shop

Ahora todo debería salirnos correctamente hasta este punto para verificar el filtro de categoría vamos a aplicarlo seleccionando la categoría de mujer:

127.0.0.1:8000

Busca un producto

Buscar productos... Mujer

Catálogo de Productos

 Camisa de Mujer
Camisa Jean Mujer
[\\$50000.00](#)

 Jean Mujer
Jean Mujer
[\\$120000.00](#)

 Short Mujer
Short Mujer
[\\$120000.00](#)

© 2025 MiTienda. Todos los derechos reservados.
Contacto: info@mitienda.shop

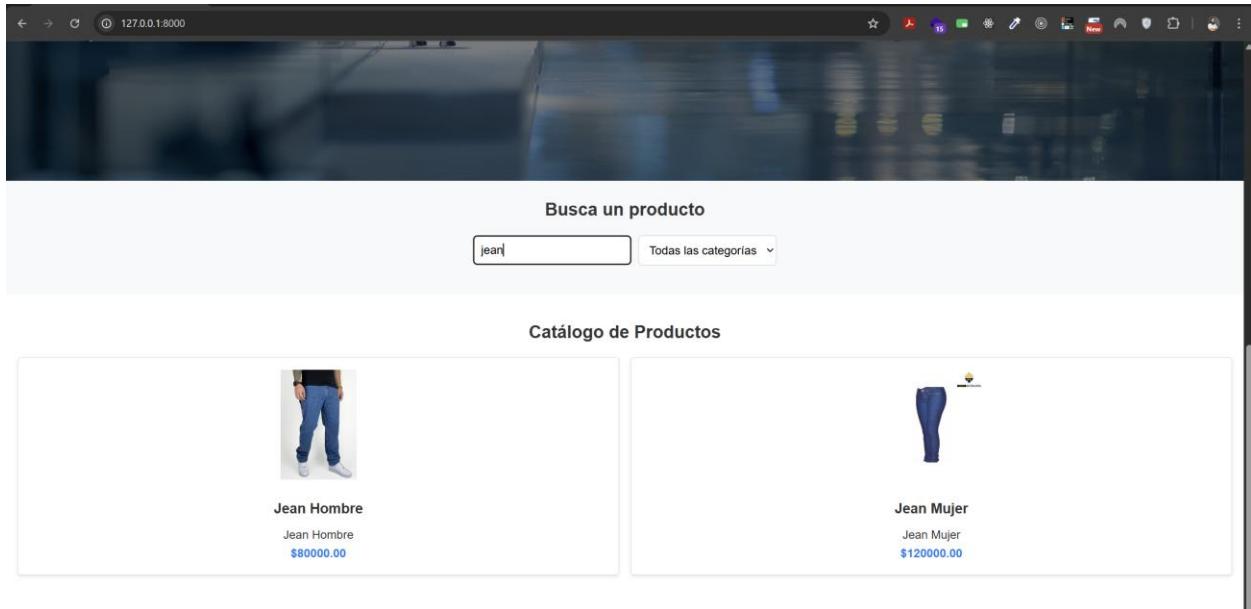
Y automáticamente me deben de salir los resultados del filtro por categoría, pero si escribimos camisa por ejemplo no va a funcionar porque nos falta pasar el Query, así

que vamos a continuar en product-search.blade.php en nuestro formulario y vamos a dejar lo siguiente:

```
resources > views > livewire > product-search.blade.php > div > section#productos.search-section > form.search-form > input.search-input
1  <div>
2    
3    <section class="search-section" id="productos">
4      <h2>Busca un producto</h2>
5      <form class="search-form" wire:submit.prevent>
6        <input type="text" placeholder="Buscar productos ..." class="search-input" wire:model.live="query">
7
8        <select class="search-select" wire:model.live="category">
9          <option value="">Todas las categorías</option>
10         @foreach ($categories as $category)
11           <option value="{{ $category->id }}>{{ $category->name }}</option>
12         @endforeach
13       </select>
```

Utilizamos wire:submit.prevent para detener el envío de datos al formulario y agregamos el atributo wire también en el input con el model.live y el Query.

Vamos a probarlo en el frente, en este caso buscara jeans



Y de esta manera me muestra los resultados, al aplicar el filtro de categoría podemos observarlo también:

A screenshot of a web browser showing a search results page. The URL in the address bar is 127.0.0.1:8000. The search bar at the top contains the word "jean" and a dropdown menu set to "Hombre". Below the search bar is a heading "Catálogo de Productos". A single product item is displayed: "Jean Hombre" (Men's jeans) with a price of \$80000.00. The page footer includes copyright information: "© 2025 MiTienda. Todos los derechos reservados." and "Contacto: info@mitienda.shop".

Y si borramos en el input la búsqueda y quitamos el filtro de categoría volverán a mostrarse los resultados normales:

A screenshot of a web browser showing a search results page. The URL in the address bar is 127.0.0.1:8000. The search bar at the top contains the placeholder "Buscar productos..." and a dropdown menu set to "Todas las categorías". Below the search bar is a heading "Catálogo de Productos". Three product items are displayed: "Camisa de Hombre" (Men's shirt) for \$50000.00, "Camisa de Mujer" (Women's shirt) for \$50000.00, and "Jean Hombre" (Men's jeans) for \$80000.00. At the bottom of the page are navigation buttons: « Anterior, 1, 2, Siguiente ». The page footer includes copyright information: "© 2025 MiTienda. Todos los derechos reservados." and "Contacto: info@mitienda.shop".

Ahora vamos a agregar un par de funciones adicionales para el momento en el que se actualicen los resultados y es que inicien desde la pagina 1 de los resultados, para ello vamos a agregar lo siguiente:

```
app > Livewire > ProductSearch.php > PHP Intelephense > ProductSearch > updatingCategory
1  <?php
2
3  namespace App\Livewire;
4
5  use App\Models\Category;
6  use App\Models\Product;
7  use Livewire\Component;
8
9  0 references | 0 implementations
10 class ProductSearch extends Component
11 {
12     2 references
13     public $query = "";
14     2 references
15     public $category = "";
16     1 reference
17     public $categories;
18
19     0 references | 0 overrides
20     public function mount(): void
21     {
22         $this->categories = Category::all();
23     }
24
25     0 references | 0 overrides
26     public function updatingQuery(): void{
27         $this->resetPage();
28     }
29     0 references | 0 overrides
30     public function updatingCategory(): void{
31         $this->resetPage();
32     }
33
34     0 references | 0 overrides
35     public function render(): View
36     {
37         $products = Product::query()
38             ->when($this->query, function (Builder<Product> $query): void {
39                 $query->where('name', 'like', '%' . $this->query . '%');
40             })
41             ->when($this->category, function (Builder<Product> $query): void {
42                 $query->where('category_id', $this->category);
43             })
44             ->paginate(3);
45         return view('livewire.product-search', compact(var_name: 'products'));
46     }
47 }
```

De esta manera terminaríamos dejando nuestra app funcional con laravel – filament – livewire.