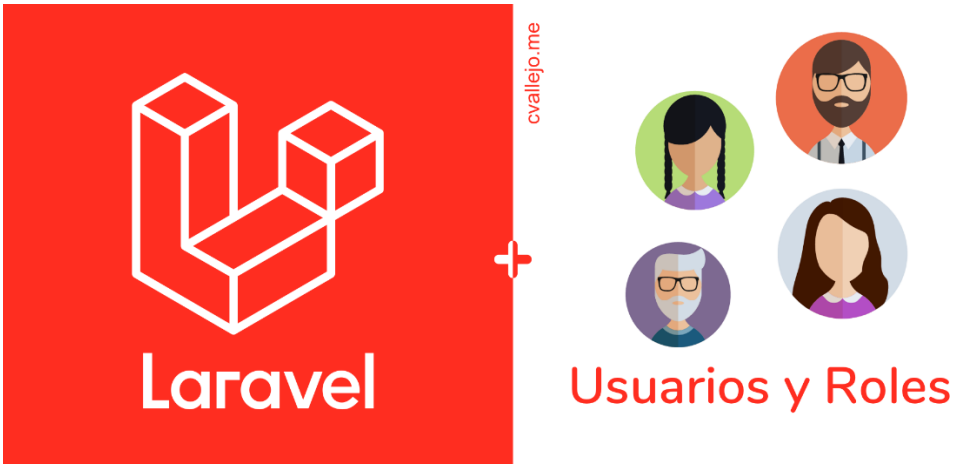
UNIDAD VIII. SECURITY

Tema 1.2: Manejo de Roles



Laravel te permite restringir el acceso a rutas o controladores mediante middlewares.

Un middleware intercepta la solicitud antes de que llegue al controlador y verifica, por ejemplo, si el usuario tiene un rol autorizado.

Para el ejemplo práctico primero veremos las rutas con el método resource.

Route::resource

El método resource no se usa con un método del controlador, sino con solo el controlador.

Laravel automáticamente crea las rutas RESTful clásicas para:

index, create, store, show, edit, update, destroy

Método	URI	Acción	Nombre
GET	/productos	index	producto.index
GET	/productos/create	create	producto.create
POST	/productos	store	producto.store
GET	/productos/{producto}	show	producto.show
GET	/productos/{producto}/edit	edit	producto.edit
PUT/PATCH	/productos/{producto}	update	producto.update

DELETE	/productos/{producto}	destroy	producto.destroy

Ejemplo:

Forma Clásica:

```php
// Producto
Route::get('/productos', [ProductoController::class, 'index'])->name('producto.index');
Route::get('/productos/data', [ProductoController::class, 'data'])->name('producto.data');
Route::post('/productos', [ProductoController::class, 'store'])->name('producto.store');
Route::post('/productos/destroy/{id}', [ProductoController::class,'destroy'])->name('productos.destroy');
Route::get('/productos/show/{id}', [ProductoController::class, 'show'])->name('producto.show');
Route::post('/productos/{id}', [ProductoController::class,'update'])->name('productos.update');
```

Método Resource:

```php
Route::resource('productos', ProductoController::class);
```

1. Para nuestro ejemplo manejaremos el Controller Producto de forma Clásica y el Controller Cliente con el método resource.

Para lo cual creamos el Controller Cliente:

php artisan make:controller ClienteController --resource

Laravel creará el archivo en:

```
app/Http/Controllers/ClienteController.php
```

Crea el controller con los métodos correspondientes:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ClienteController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     */
    public function show(string $id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit(string $id)
    {
        //
    }
```

Route::resource() es una forma rápida de registrar todas las rutas CRUD (crear, leer, actualizar, eliminar) de un recurso —por ejemplo "clientes"— con una sola línea de código.

```php
Route::resource('clientes', ClienteController::class);
```

Laravel automáticamente genera las 7 rutas RESTful estándar, enlazadas a los métodos del controlador ClienteController.

Acción	Método HTTP	URL	Método del controlador	Nombre de ruta
Listar todos los clientes	GET	/clientes	index()	clientes.index
Mostrar formulario de creación	GET	/clientes/create	create()	clientes.create
Guardar un nuevo cliente	POST	/clientes	store()	clientes.store
Mostrar un cliente específico	GET	/clientes/{cliente}	show($id)	clientes.show
Mostrar formulario de edición	GET	/clientes/{cliente}/edit	edit($id)	clientes.edit
Actualizar cliente existente	PUT/PATCH	/clientes/{cliente}	update($request, $id)	clientes.update
Eliminar un cliente	DELETE	/clientes/{cliente}	destroy($id)	clientes.destroy

Laravel genera automáticamente esos **7 métodos** vacíos que se vinculan con las rutas anteriores.

```php
public function index()   // GET /clientes
public function create()  // GET /clientes/create
public function store(Request $request) // POST /clientes
public function show($id) // GET /clientes/{cliente}
public function edit($id) // GET /clientes/{cliente}/edit
public function update(Request $request, $id) // PUT/PATCH /clientes/{cliente}
public function destroy($id) // DELETE /clientes/{cliente}
```

2. Creamos la ruta resource para el Controller Cliente:

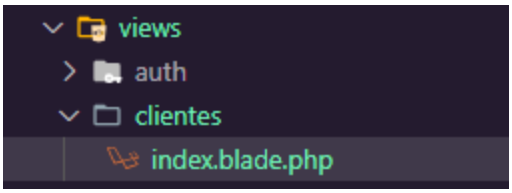
```php
use App\Http\Controllers\ClienteController;

// Cliente
Route::resource('cliente', ClienteController::class);
```

```php
<?php

use App\Http\Controllers\HolaMundoController;
use App\Http\Controllers\ProductoController;

use App\Http\Controllers\ClienteController;

use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Auth::routes();

Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
Route::get('/hola_mundo', [HolaMundoController::class, 'mostrar_hola_mundo'])->name('hola_mundo');
Route::get('/usuario/index', [UserController::class, 'index'])->name('usuario.index');

// Producto
Route::get('/productos', [ProductoController::class, 'index'])->name('producto.index');
Route::get('/productos/data', [ProductoController::class, 'data'])->name('producto.data');
Route::post('/productos', [ProductoController::class, 'store'])->name('producto.store');
Route::post('/productos/destroy/{id}', [ProductoController::class,'destroy'])->name('productos.destroy');
Route::get('/productos/show/{id}', [ProductoController::class, 'show'])->name('producto.show');
Route::post('/productos/{id}', [ProductoController::class,'update'])->name('productos.update');

// Cliente
Route::resource('cliente', ClienteController::class);
```

3. En el método index creamos la vista **cliente.index.blade.php**

```php
public function index()
    {
        // 👁 Retorna la vista
        return view('clientes.index');
    }
```

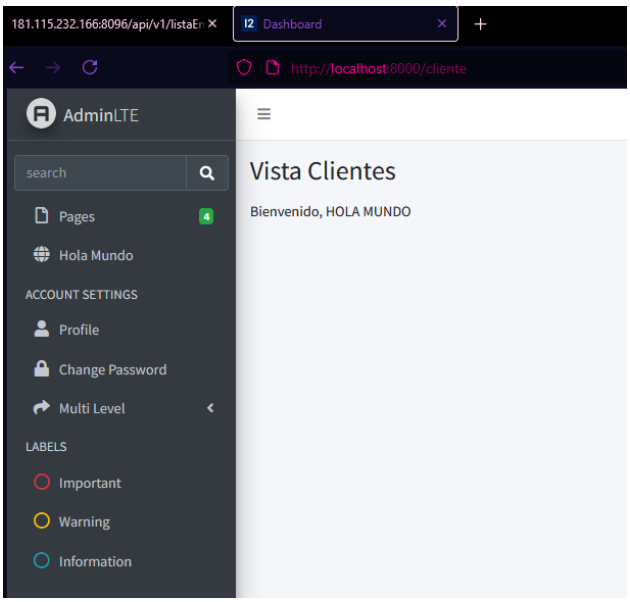
4. Luego creamos la vista correspondiente



5. En la Vista adicionamos el template base de AdminLTE:

```
@extends('adminlte::page')

@section('title', 'Dashboard')

@section('content_header')
    <h1>Vista Clientes</h1>
@stop

@section('content')
    <p>Bienvenido, HOLA MUNDO</p>
@stop

@section('css')
    {{-- Add here extra stylesheets --}}
    {{-- <link rel="stylesheet" href="/css/admin_custom.css"> --}}
@stop

@section('js')
    {{-- <script> console.log("Hi, I'm using the
        Laravel-AdminLTE package!"); </script> --}}
@stop
```

6. El Resultado



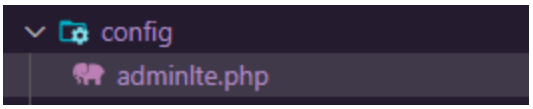
ROLES Y USUARIOS:

Para el ejemplo crearemos dos roles, el Rol que administre los Productos y el Rol que administre los Clientes:

Rol	Descripción
Administrador de Productos (`admin_productos`)	Tiene permisos para crear, editar, eliminar y visualizar productos.
Administrador de Clientes (`admin_clientes`)	Tiene permisos para registrar, actualizar, eliminar y consultar clientes.

Nombre visible	Nombre interno (slug)
Administrador de Productos	`admin_productos`
Administrador de Clientes	`admin_clientes`

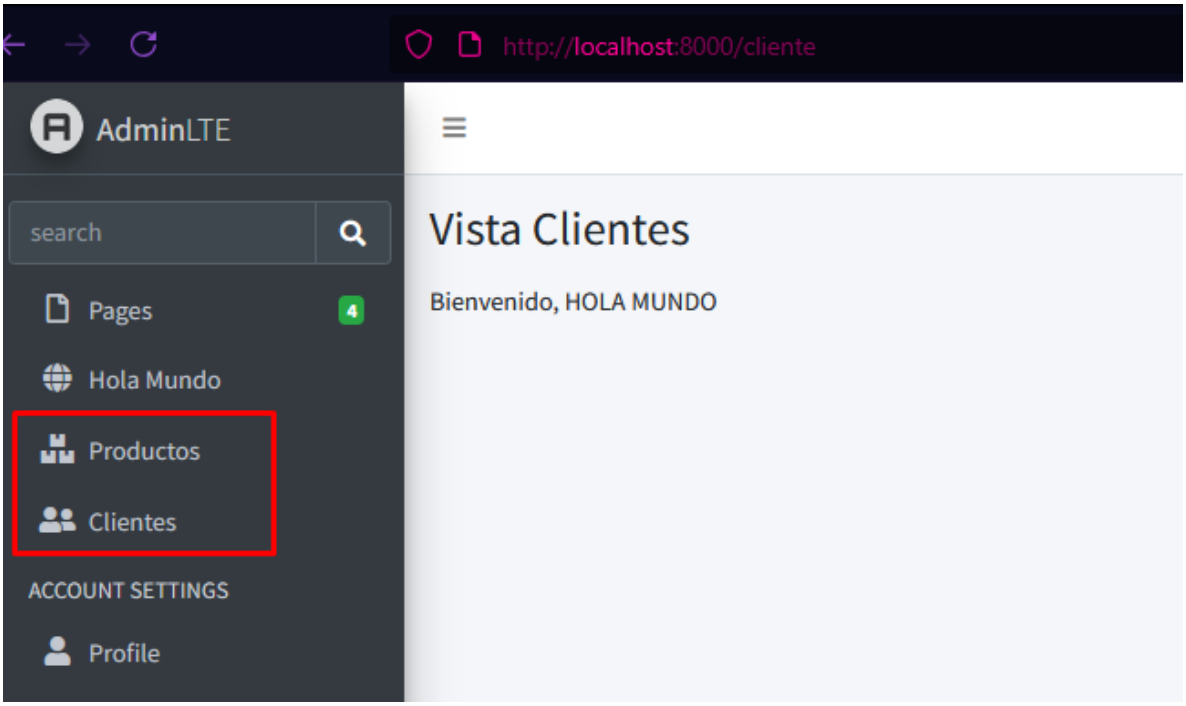
7. Primeramente, pondremos las rutas principales en el Sidebar del Template:
8. Editamos el Archivo de configuración de Admin LTE



9. Adicionamos los enlaces según el modelo:

```php
//Producto
[
    'text' => 'Productos',
    'url'  => '/productos',
    'icon' => 'fas fa-boxes',
],

//Cliente
[
    'text' => 'Clientes',
    'url'  => '/cliente',
    'icon' => 'fas fa-user-friends',
],
```

10. El Resultado:



CREACIÓN DEL MIDDLEWARE

Un middleware en Laravel es un “filtro” que se ejecuta antes o después de una petición HTTP.

Sirve para verificar permisos, autenticación o condiciones antes de permitir el acceso a una ruta.

11. Laravel te permite restringir el acceso a rutas o controladores mediante middlewares.
    Un middleware intercepta la solicitud antes de que llegue al controlador y verifica, por ejemplo, si el usuario tiene un rol autorizado.
    Ejecutá estos dos comandos en tu terminal:

php artisan make:middleware ProductoMiddleware

php artisan make:middleware ClienteMiddleware

12. Esto crea dos archivos:

app/Http/Middleware/ProductoMiddleware.php

app/Http/Middleware/ClienteMiddleware.php

13. Ahora ajustamos los middlewares **ProductoMiddleware** y **ClienteMiddleware** para consultar el rol real del usuario:

app\Http\Middleware\ProductoMiddleware.php

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\UserRol;
use App\Models\ParamRol;

class ProductoMiddleware
{
    public function handle(Request $request, Closure $next)
    {
       // 🔧 ID del rol permitido para este middleware (admin_productos = 4)
        $rolId = 4;

        if (!Auth::check()) {
            return redirect('/login');
        }

        $idUser = Auth::id();

        // 🔍 Verifica si el usuario logueado tiene el rol con ese ID
        $tieneRol = UserRol::where('id_user', $idUser)
            ->where('id_rol', $rolId)
            ->where('estado', 1)
            ->exists();

        if (!$tieneRol) {
            abort(403, 'Acceso denegado al módulo de Productos.');
```

```
        }

        return $next($request);
    }
}
```

app\Http\Middleware\ClienteMiddleware.php

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\UserRol;
use App\Models\ParamRol;

class ClienteMiddleware
{
    public function handle(Request $request, Closure $next)
    {
       // 🔧 ID del rol permitido para este middleware (admin_clientes = 5)
        $rolId = 5;

        if (!Auth::check()) {
            return redirect('/login');
        }

        $idUser = Auth::id();

        // 🔍 Verifica si el usuario logueado tiene el rol con ese ID
        $tieneRol = UserRol::where('id_user', $idUser)
            ->where('id_rol', $rolId)
            ->where('estado', 1)
            ->exists();

        if (!$tieneRol) {
            abort(403, 'Acceso denegado al módulo de Clientes.');
        }

        return $next($request);
```

```
    }
}
```

14. Desde Laravel 11 en adelante (y por tanto también en Laravel 12), el registro de middlewares ya no se hace en **app/Http/Kernel.php**, sino en el archivo:

bootstrap/app.php

```php
use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

use App\Http\Middleware\ProductoMiddleware;
use App\Http\Middleware\ClienteMiddleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__ . '/../routes/web.php',
        commands: __DIR__ . '/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {

        // Alias de middlewares personalizados
        $middleware->alias([
            'producto' => ProductoMiddleware::class,
            'cliente'  => ClienteMiddleware::class,
        ]);

    })
    ->withExceptions(function (Exceptions $exceptions): void {
        //
    })->create();
```

```php
use App\Http\Middleware\ProductoMiddleware;
use App\Http\Middleware\ClienteMiddleware;
.
.
.

->withMiddleware(function (Middleware $middleware): void {

        // Alias de middlewares personalizados
        $middleware->alias([
            'producto' => ProductoMiddleware::class,
            'cliente'  => ClienteMiddleware::class,
        ]);

    })
```

uso en rutas

Cómo usarlo:

```php
Route::middleware('producto')->group(function () {
   // Rutas Producto
});

Route::middleware('cliente')->group(function () {
    // Rutas Clientes
});
```

Un middleware en Laravel es un “filtro” que se ejecuta antes o después de una petición HTTP.

Sirve para verificar permisos, autenticación o condiciones antes de permitir el acceso a una ruta.

- **auth** → solo deja entrar a usuarios logueados.
- **producto** → solo deja entrar a usuarios con rol ID 4 (admin_productos).
- **cliente** → solo deja entrar a usuarios con rol ID 5 (admin_clientes).

```php
Route::middleware('producto')->group(function () {
    // Rutas Producto
});
```

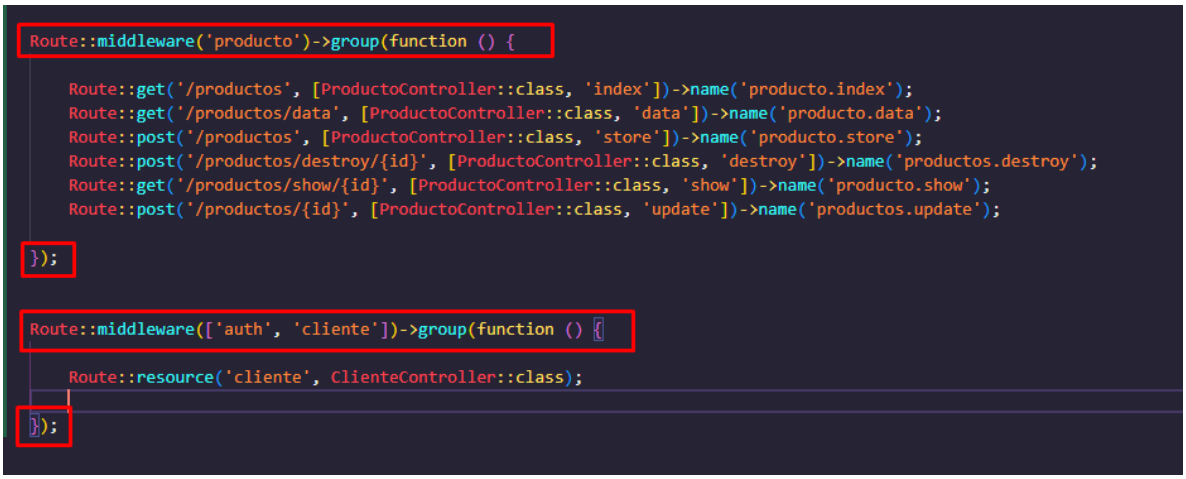
Esto significa que todas las rutas dentro del grupo estarán protegidas por el middleware producto.

Es como decir:

“Antes de entrar a cualquiera de estas rutas, Laravel debe ejecutar el middleware ProductoMiddleware y verificar si el usuario tiene permiso.”

15. Protegemos nuestras rutas:

```php
Route::middleware('producto')->group(function () {

Route::get('/productos', [ProductoController::class, 'index'])->name('producto.index');
Route::get('/productos/data', [ProductoController::class, 'data'])->name('producto.data');
Route::post('/productos', [ProductoController::class, 'store'])->name('producto.store');
Route::post('/productos/destroy/{id}', [ProductoController::class, 'destroy'])-
>name('productos.destroy');
Route::get('/productos/show/{id}', [ProductoController::class, 'show'])->name('producto.show');
Route::post('/productos/{id}', [ProductoController::class, 'update'])->name('productos.update');
});


Route::middleware(['auth', 'cliente'])->group(function () {

    Route::resource('cliente', ClienteController::class);
});
```



```php
Route::middleware('producto')->group(function () {

    Route::get('/productos', [ProductoController::class, 'index'])->name('producto.index');
    Route::get('/productos/data', [ProductoController::class, 'data'])->name('producto.data');
    Route::post('/productos', [ProductoController::class, 'store'])->name('producto.store');
    Route::post('/productos/destroy/{id}', [ProductoController::class, 'destroy'])->name('productos.destroy');
    Route::get('/productos/show/{id}', [ProductoController::class, 'show'])->name('producto.show');
    Route::post('/productos/{id}', [ProductoController::class, 'update'])->name('productos.update');

});

Route::middleware(['auth', 'cliente'])->group(function () {

    Route::resource('cliente', ClienteController::class);

});
```

['auth', 'producto']

→ Primero verifica que el usuario esté logueado (auth),

→ Luego ejecuta tu middleware personalizado ProductoMiddleware (rol 4).

['auth', 'cliente']

→ Hace lo mismo, pero usando el ClienteMiddleware (rol 5).

Route::resource('cliente', ClienteController::class)

→ Genera automáticamente las rutas REST (index, create, store, show, edit, update, destroy)

todas protegidas por el middleware cliente.

16. Verificamos si existe algún error ejecutamos en consola

php artisan route:list

nos despliega

17. Si sale algún error Ejecutá desde tu terminal:

php artisan optimize:clear

composer dump-autoload

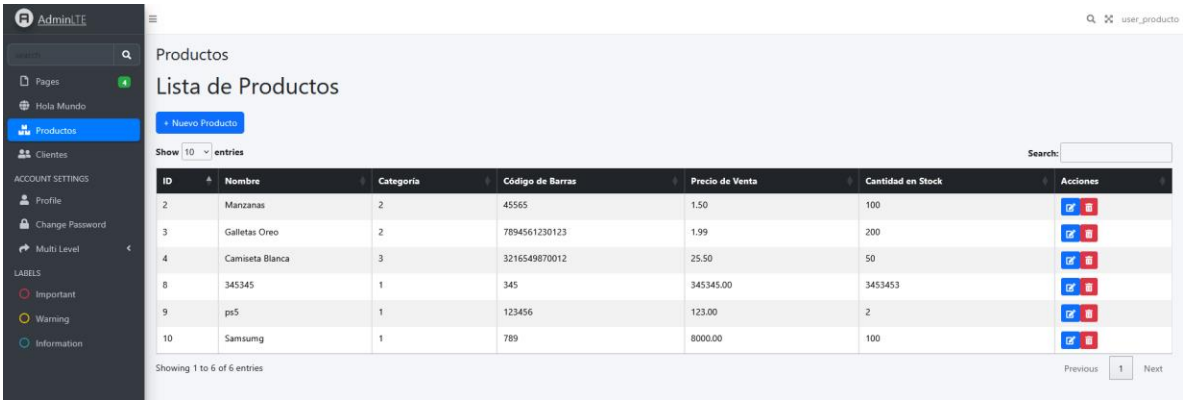
php artisan route:clear

php artisan config:clear

Esto borra los archivos de caché donde Laravel guarda el listado de middlewares y rutas antiguas.

18. Probamos con el usuario user_producto

Con acceso a Productos



Sin acceso a Clientes

PROTECCIÓN VISUAL

19. Protegemos visualmente nuestras rutas:

Crear el archivo **app/Providers/AuthServiceProvider.php** y pega todo este contenido:

```php
<?php

namespace App\Providers;

use Illuminate\Foundation\Support\Providers\AuthServiceProvider as Service-
Provider;
use Illuminate\Support\Facades\Gate;
use App\Models\UserRol;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * Las políticas de autorización para los modelos.
     *
     * @var array<class-string, class-string>
     */
    protected $policies = [
        // 'App\Models\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Registrar los servicios de autorización.
     */
    public function boot(): void
    {
        // 🔒 Gate para ver el módulo Productos
        Gate::define('ver-productos', function ($user) {
            $roles = UserRol::where('id_user', $user->id)
                ->where('estado', 1)
                ->pluck('id_rol')
                ->toArray();

            // Solo si tiene el rol ID 4 (admin_productos)
            return in_array(4, $roles);
        });
```

```
        // 🔒 Gate para ver el módulo Clientes
        Gate::define('ver-clientes', function ($user) {
            $roles = UserRol::where('id_user', $user->id)
                ->where('estado', 1)
                ->pluck('id_rol')
                ->toArray();

            // Solo si tiene el rol ID 5 (admin_clientes)
            return in_array(5, $roles);
        });
    }
}
```

20. Registrar el provider en tu aplicación

Abrí el archivo:

bootstrap/app.php

Y buscá la parte donde se configuran los providers.

```
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

use App\Http\Middleware\ProductoMiddleware;
use App\Http\Middleware\ClienteMiddleware;

use App\Providers\AuthServiceProvider; // 👈 agrega esta línea


return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__ . '/../routes/web.php',
        commands: __DIR__ . '/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {

        // Alias de middlewares personalizados
        $middleware->alias([
            'producto' => ProductoMiddleware::class,
            'cliente'  => ClienteMiddleware::class,
        ]);

    })
    ->withProviders([
        AuthServiceProvider::class, // 👈 registra tu provider de Gates
    ])
    ->withExceptions(function (Exceptions $exceptions): void {
        //
    })->create();
```

```php
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

use App\Http\Middleware\ProductoMiddleware;
use App\Http\Middleware\ClienteMiddleware;

use App\Providers\AuthServiceProvider; // 👈 agrega esta línea


return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__ . '/../routes/web.php',
        commands: __DIR__ . '/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {

        // Alias de middlewares personalizados
        $middleware->alias([
            'producto' => ProductoMiddleware::class,
            'cliente'  => ClienteMiddleware::class,
        ]);

    })

    ->withProviders([
        AuthServiceProvider::class, // 👈 registra tu provider de Gates
    ])

    ->withExceptions(function (Exceptions $exceptions): void {
        //
    })->create();
```

Qué hace esto:use App\Providers\AuthServiceProvider; → importa tu provider.

- ->withProviders([...]) → le dice a Laravel que cargue ese provider durante el arranque.

- Así, tus Gates (ver-productos, ver-clientes) estarán listos antes de que AdminLTE los consulte.

21. Abrí config/adminlte.php, buscá la parte que dice algo como esto:

```
// Productos
        [
            'text' => 'Productos',
            'url' => '/productos',
            'icon' => 'fas fa-boxes',
            'can'  => 'ver-productos', // visible solo si pasa el Gate
        ],

 //Clientes
        [
            'text' => 'Clientes',
            'url' => '/cliente',
            'icon' => 'fas fa-user-friends',
            'can'  => 'ver-clientes', // visible solo si pasa el Gate
        ],
```

Adicionamos la opción can con los nombres de los Gates configurados

Qué pasa después:

Usuario (roles en `user_rol`)	Verá en menú
ID 4 → admin_productos	Productos
ID 5 → admin_clientes	Clientes
ID 4 y 5	Admin (Ambos)
sin roles	ninguno

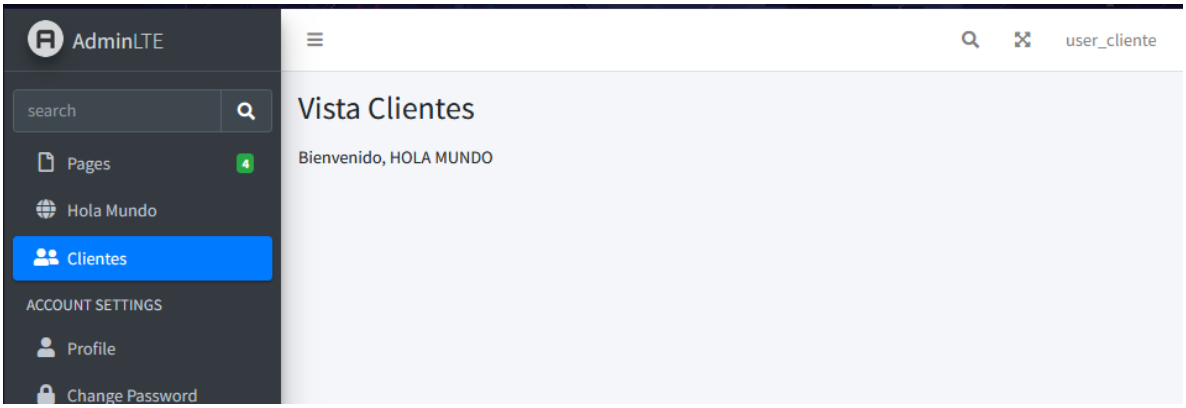
22. Ejecuta

php artisan optimize:clear

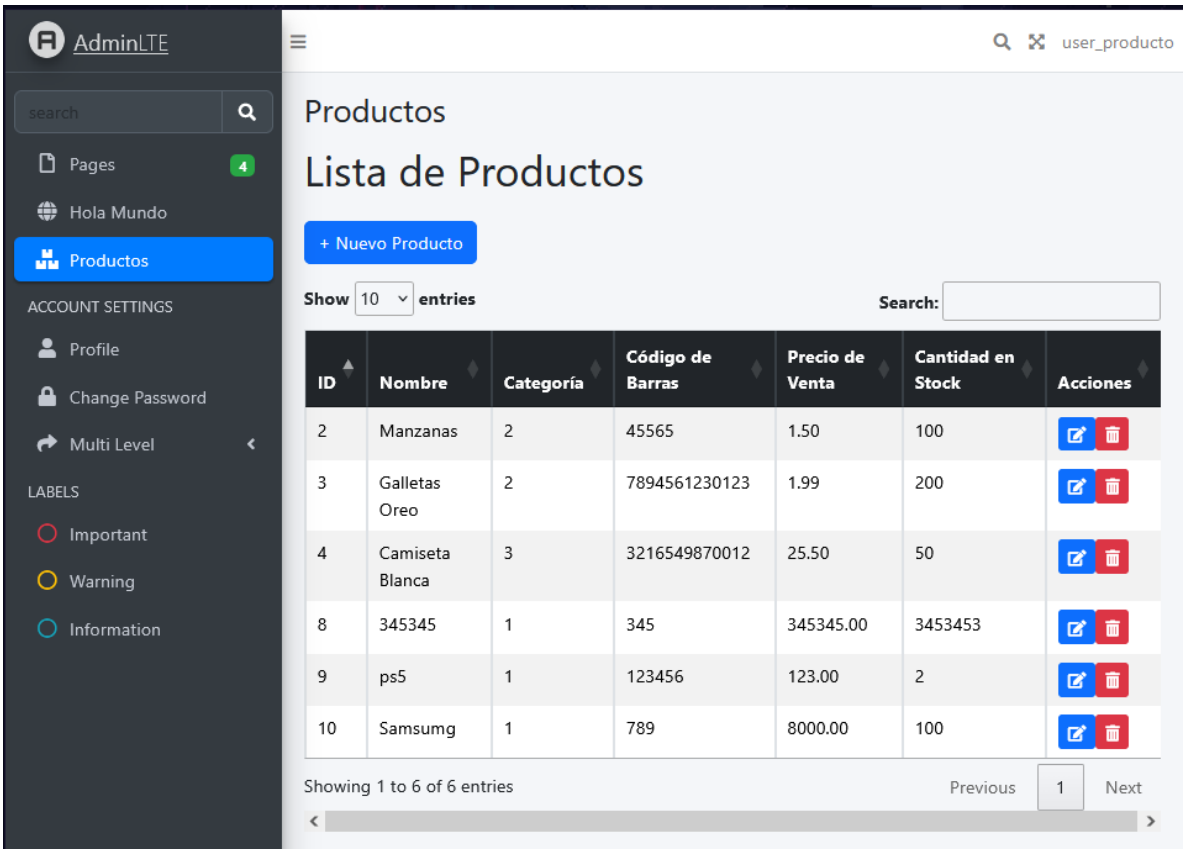
Y cuando recargues tu panel AdminLTE,

23. Probamos:

Usuario cliente



Usuario producto



TAREA: CREACIÓN DEL ROL "ADMINISTRADOR" CON ACCESO A PRODUCTOS Y CLIENTES

🎯 Objetivo de la práctica

El objetivo de esta tarea es que el estudiante (o usuario del sistema) aprenda a crear un nuevo rol denominado "Administrador" que tenga la capacidad de acceder tanto al módulo de Productos como al módulo de Clientes, aplicando los conceptos de roles, permisos y gates en Laravel.

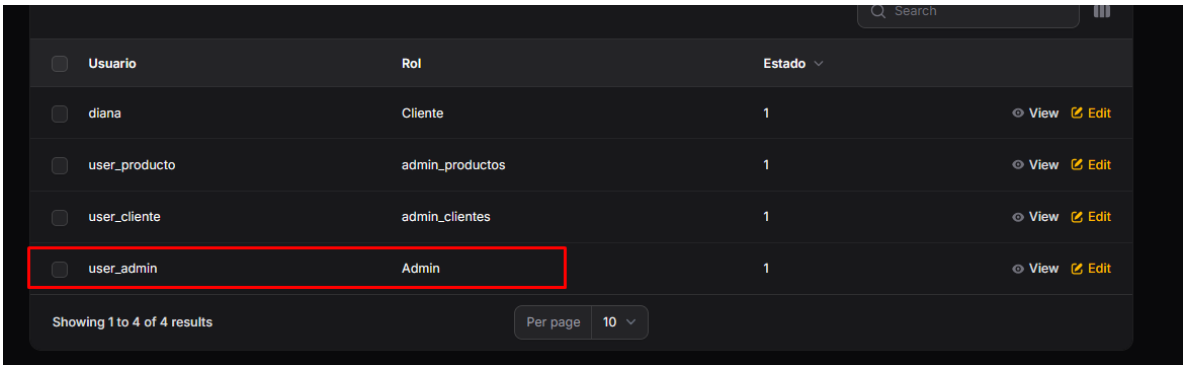
Instrucciones

1. Investigar qué es un "Gate" en Laravel
    a. Explica brevemente para qué sirve un Gate y cómo se utiliza para controlar el acceso a ciertas secciones del sistema.
    b. Menciona la diferencia entre un Gate y un Middleware.
2. Analizar la estructura actual de roles
    a. Observa la tabla user_rol donde se asignan los roles a los usuarios.
    b. Identifica los roles existentes:
        i. Rol ID 4 → admin_productos
        ii. Rol ID 5 → admin_clientes
3. Crear un nuevo rol llamado "Administrador"
    a. Inserta un nuevo registro en la tabla param_rol con los siguientes datos de ejemplo:
        i. id = 6
        ii. rol = admin_general
        iii. descripcion = Acceso completo a Productos y Clientes
        iv. estado = 1
    b. Asigna este rol a un usuario en la tabla user_rol.
4. Modificar los Gates en AuthServiceProvider
    a. Agrega un nuevo Gate para el rol administrador.
    b. Este Gate debe permitir ver ambos módulos (productos y clientes) si el usuario tiene el rol ID 6.
5. Probar el acceso
    a. Inicia sesión con un usuario que tenga:
        i. Rol 4 → debería ver solo Productos.
        ii. Rol 5 → debería ver solo Clientes.
        iii. Rol 6 (Administrador) → debería ver ambos menús.

6. Verifica que los Gates funcionen correctamente en el panel AdminLTE (menú lateral) y que no aparezcan secciones que no correspondan al rol.

RESTRICCIONES DEL PANEL ADMIN FILAMENT

Creamos el Usuario y Rol Administrador Correspondiente



Crea un middleware

php artisan make:middleware FilamentOnlyRolAdmin

Edita el middleware así

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Models\UserRol;
use Symfony\Component\HttpFoundation\Response;

class FilamentOnlyRolAdmin
{
    public function handle(Request $request, Closure $next): Response
    {
        // Si el usuario no está autenticado, dejar que el middleware Authenticate lo maneje
        if (! Auth::check()) {
            return $next($request);
        }

        $user = Auth::user();

        // Buscar el registro de rol del usuario
        $userRol = UserRol::where('id_user', $user->id)->first();
```

```
        // Validar si existe un registro y si el rol no es 2
        if (! $userRol || (int) $userRol->id_rol !== 2) {
            // Puedes mostrar un mensaje o redirigir a otra ruta
            abort(403, 'No tienes permiso para acceder al panel de administración.');
        }

        return $next($request);
    }
}
```

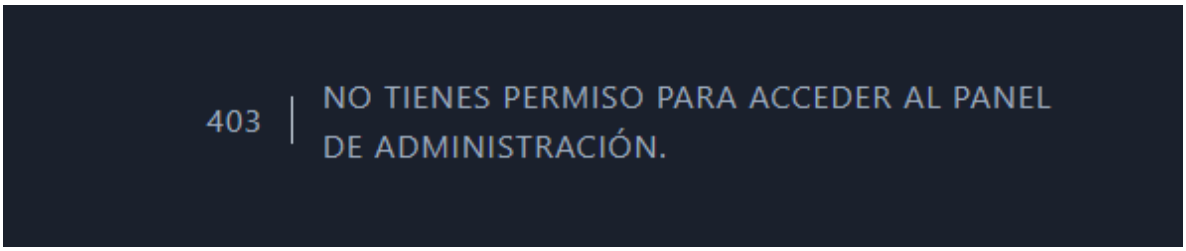
Agrégalo a tu panel en AdminPanelProvider.php
app\Providers\Filament\AdminPanelProvider.php

```
            FilamentInfoWidget::class,
        ])
        ->middleware([
            EncryptCookies::class,
            AddQueuedCookiesToResponse::class,
            StartSession::class,
            AuthenticateSession::class,
            ShareErrorsFromSession::class,
            VerifyCsrfToken::class,
            SubstituteBindings::class,
            DisableBladeIconComponents::class,
            DispatchServingFilamentEvent::class,
        ])
        ->authMiddleware([
            Authenticate::class,

            \App\Http\Middleware\FilamentOnlyRolAdmin::class,

        ]);
    }
}
```

```
\App\Http\Middleware\FilamentOnlyRolAdmin::class,
```

Cuando un usuario accede a /admin:

- Si **no está logueado**, el middleware Authenticate lo redirige al login.
- Si **está logueado**, FilamentOnlyRol2 revisa su rol en la tabla user_rol.
- Si su **id_rol != 2** → se lanza un 403 Acceso denegado.
- Si su **id_rol == 2** → puede entrar al panel sin problema.



Si quieres que el usuario sea redirigido a otra ruta (por ejemplo, al inicio del sitio) en lugar de ver el error 403, cambia esta línea:

```
abort(403, 'No tienes permiso para acceder al panel de administración.');
```

por:

```
return redirect('/')->with('error', 'No tienes acceso al panel de administración.');
```

ROL ADMINISTRADOR

Modificamos los Midlewares de Productos y Clientes para que:

1. **ClienteMiddleware** → deja pasar si es cliente o admin
2. **ProductoMiddleware** → deja pasar si es producto o admin

ProductoMidleware:

```
    public function handle(Request $request, Closure $next)
    {
// 🔧 ID del rol permitido para este middleware (admin_productos = 4, admin = 2)
        // $rolId = 5;
        $rolesPermitidos = [2, 6];

        if (!Auth::check()) {
            return redirect('/login');
        }

        $idUser = Auth::id();

        // 🔍 Verifica si el usuario logueado tiene el rol con ese ID
        $tieneRol = UserRol::where('id_user', $idUser)
            // ->where('id_rol', $rolId)
            ->whereIn('id_rol', $rolesPermitidos)
```

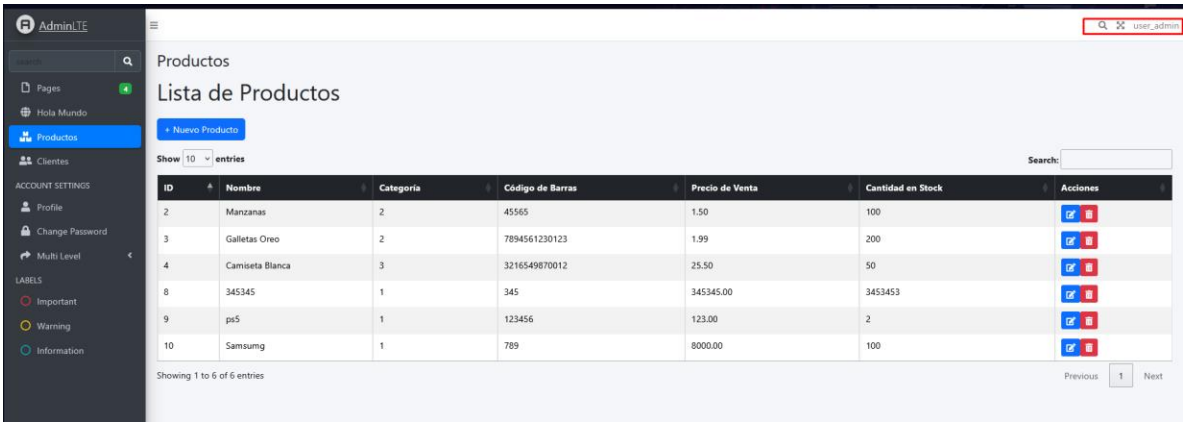
```
            ->where('estado', 1)
            ->exists();
        if (!$tieneRol) {
            abort(403, 'Acceso denegado al módulo de Productos.');
        }
        return $next($request);
    }
```

ClienteMidleware:

```
public function handle(Request $request, Closure $next)
    {
        // 🔧 ID del rol permitido admin_clientes = 6 y para admin = 2
        // $rolId = 6;
        $rolesPermitidos = [2, 6];

        if (!Auth::check()) {
            return redirect('/login');
        }

        $idUser = Auth::id();

        // 🔍 Verifica si el usuario logueado tiene el rol con ese ID
        $tieneRol = UserRol::where('id_user', $idUser)
            // ->where('id_rol', $rolId)
            ->whereIn('id_rol', $rolesPermitidos)
            ->where('estado', 1)
            ->exists();
        if (!$tieneRol) {
            abort(403, 'Acceso denegado al módulo de Clientes.');
        }
        return $next($request);
    }
```

Antes	Después
Solo rol 6 podía entrar	Rol 2 (admin) **y** 6 (cliente) pueden entrar
`where('id_rol', $rolId)`	`whereIn('id_rol', [2, 6])`
No permitía al admin acceder	Ahora sí lo permite

Probamos:



CORRECCION DE LOS GATES

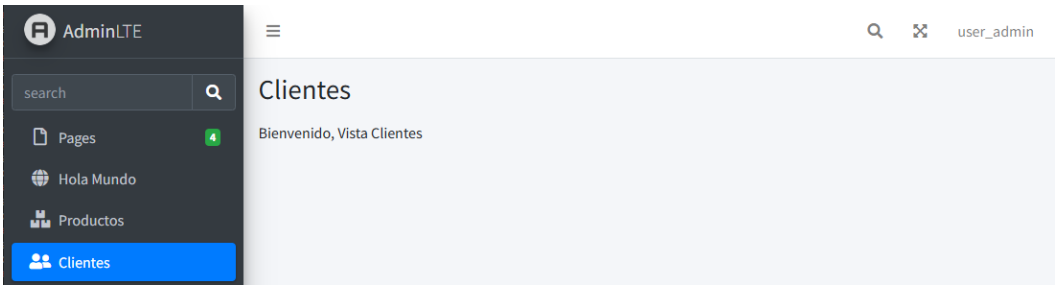
Ahora que hemos realizado los cambios de la protección visual, tenemos que corregir los gates para que el rol Admin puede ver las rutas de Productos y Clientes, por lo que modificamos el archivo:

app\Providers\AuthServiceProvider.php

```php
public function boot(): void
{

    // Gate para ver el módulo Productos
    Gate::define('ver-producto', function ($user) {

        // Roles que pueden ver Productos
        $rolesPermitidos = [2, 4];

        $roles = UserRol::where('id_user', $user->id)
            ->where('estado', 1)
            ->pluck('id_rol')
            ->toArray();

        // Solo si tiene el rol ID 5 (admin_productos)
        // return in_array(4, $roles);
        return !empty(array_intersect($rolesPermitidos, $roles));
    });
    // Gate para ver el módulo Clientes
    Gate::define('ver-cliente', function ($user) {

        // Roles que pueden ver Productos
        $rolesPermitidos = [2, 5];

        $roles = UserRol::where('id_user', $user->id)
            ->where('estado', 1)
            ->pluck('id_rol')
            ->toArray();

        // Solo si tiene el rol ID 6 (admin_clientes)
        // return in_array(5, $roles);
        return !empty(array_intersect($rolesPermitidos, $roles));

    });
}
```

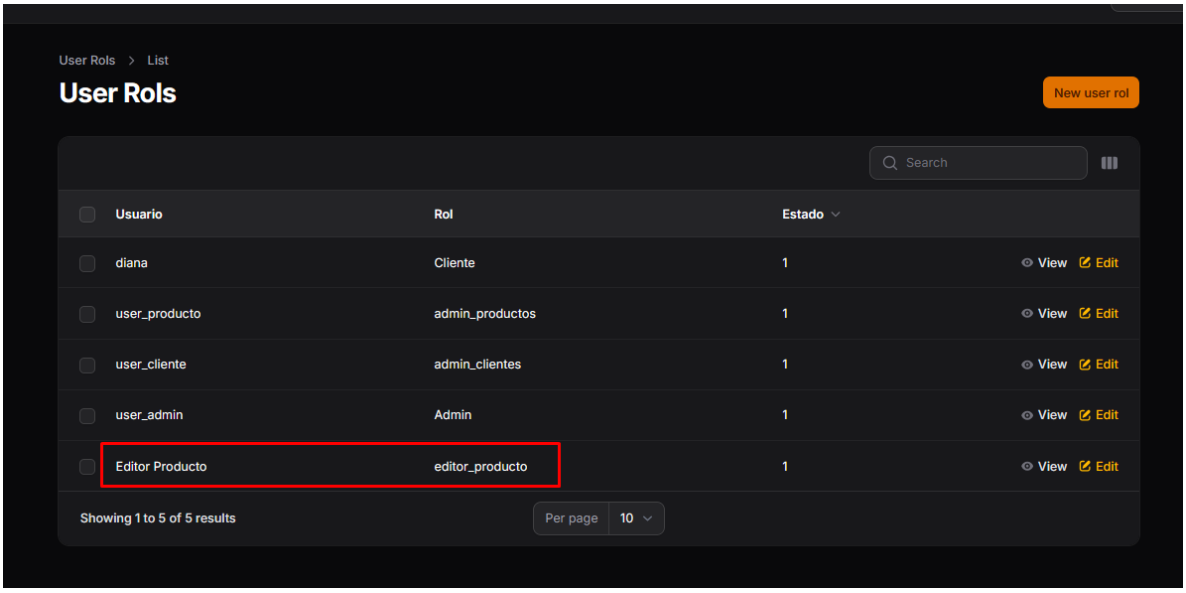
```php
// Gate para ver el módulo Productos
        Gate::define('ver-producto', function ($user) {

            // Roles que pueden ver Productos
            $rolesPermitidos = [2, 4];

            $roles = UserRol::where('id_user', $user->id)
                ->where('estado', 1)
                ->pluck('id_rol')
                ->toArray();

            // Solo si tiene el rol ID 5 y 2 (admin_productos, admin)
            // return in_array(4, $roles);
            return !empty(array_intersect($rolesPermitidos, $roles));
        });
// Gate para ver el módulo Clientes
        Gate::define('ver-cliente', function ($user) {

            // Roles que pueden ver Clientes
            $rolesPermitidos = [2, 5];

            $roles = UserRol::where('id_user', $user->id)
                ->where('estado', 1)
                ->pluck('id_rol')
                ->toArray();

            // Solo si tiene el rol ID 6 y 2 (admin_clientes, admin)
            // return in_array(5, $roles);
            return !empty(array_intersect($rolesPermitidos, $roles));

        });
```

Prueba el resultado



EJEMPLO DE ROLES APROBADOR Y EDITOR PARA PRODUCTOS

Creamos el Usuario y Rol EditorProducto Correspondiente



donde ambos roles pueden acceder a "productos", pero con diferentes privilegios:

- **Aprobador (rol producto, p. ej. id = 5)** → puede hacer todo.
- **Editor (rol editorProducto, p. ej. id = 6)** → puede ver, crear, listar, pero no eliminar ni actualizar.

Desactivamos **temporalmente** el can de la ruta productos para las pruebas, por lo que modificamos **config\adminlte.php**

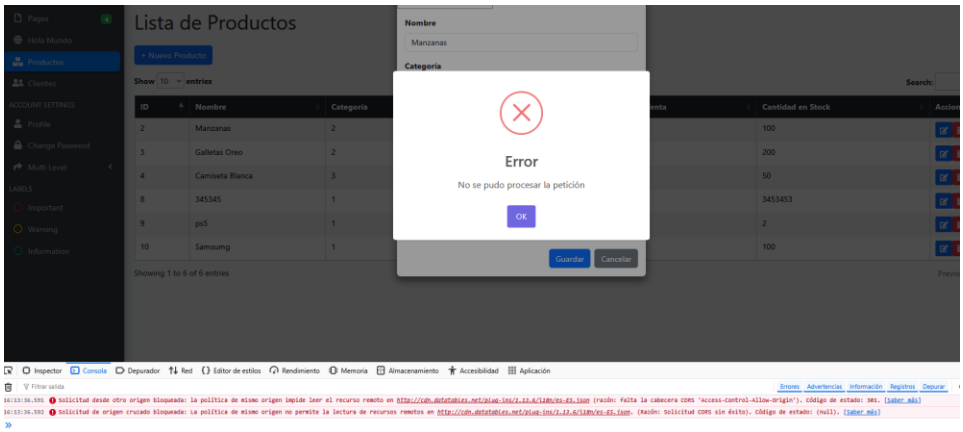
```
//Producto
        [
            'text' => 'Productos',
            'url' => '/productos',
            'icon' => 'fas fa-boxes',
            /* 'can' => 'ver-producto', */
        ],
```

Modificamos **ProductoMiddleware**

```php
class ProductoMiddleware
{
    public function handle(Request $request, Closure $next)
    {
        // 🔧 IDs de roles permitidos para el módulo Productos
        // 2 = Admin, 4 = AprobadorProducto, 6 = EditorProducto
        $rolesPermitidos = [2, 4, 6];

        if (!Auth::check()) {
            return redirect('/login');
        }

        $idUser = Auth::id();

        // 🔍 Verifica si el usuario logueado tiene el rol con ese ID
        $tieneRol = UserRol::where('id_user', $idUser)
            // ->where('id_rol', $rolId)
            ->whereIn('id_rol', $rolesPermitidos)
            ->where('estado', 1)
            ->exists();
        if (!$tieneRol) {
            abort(403, 'Acceso denegado al módulo de Productos.');
        }

        // 🚫 Si el usuario es "EditorProducto" (id_rol = 6)
        // restringimos las rutas de eliminar o actualizar
        if ($tieneRol == 6) {
            $rutaActual = $request->route()->getName();

            $rutasRestringidas = [
                'productos.destroy',
                'productos.update',
            ];

            if (in_array($rutaActual, $rutasRestringidas)) {
                abort(403, 'No tienes permiso para eliminar o actualizar productos.');
            }
        }

        return $next($request);
    }
}
```

```php
public function handle(Request $request, Closure $next)
    {
        // 🔧 IDs de roles permitidos para el módulo Productos
        // 2 = Admin, 4 = AprobadorProducto, 6 = EditorProducto
        $rolesPermitidos = [2, 4, 6];

        if (!Auth::check()) {
            return redirect('/login');
        }

        $idUser = Auth::id();

        // 🔍 Verifica si el usuario logueado tiene el rol con ese ID
        $tieneRol = UserRol::where('id_user', $idUser)
            // ->where('id_rol', $rolId)
            ->whereIn('id_rol', $rolesPermitidos)
            ->where('estado', 1)
            ->exists();
        if (!$tieneRol) {
            abort(403, 'Acceso denegado al módulo de Productos.');
        }

        // 🚫 Si el usuario es "EditorProducto" (id_rol = 6)
        // restringimos las rutas de eliminar o actualizar
        if ($tieneRol == 6) {
            $rutaActual = $request->route()->getName();

            $rutasRestringidas = [
                'productos.destroy',
                'productos.update',
            ];

            if (in_array($rutaActual, $rutasRestringidas)) {
     abort(403, 'No tienes permiso para eliminar o actualizar productos.');
            }
        }

        return $next($request);
    }
```

Probamos:



Solicitud desde otro origen bloqueada: la política de mismo origen impide leer el recurso remoto

Ahora podemos realizar una protección visual a la lista de productos para que el rol **editor_productos** no pueda ver los botones de editar o eliminar.

Modificamos:

app\Http\Controllers\ProductoController.php

```php
use Illuminate\Support\Facades\Auth;
use App\Models\UserRol;

class ProductoController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        // Primera Forma
        // $productos = Producto::all();

        // Segunda Forma
        // $productos = Producto::select('id', 'nombre', 'codigo_barras', 'precio_venta', 'cant
        //     ->where('estado', true)
        //     ->orderBy('id', 'desc')
        //     ->get();

        $categorias = Categoria::select('id', 'descripcion')
            ->where('estado', true)
            ->orderBy('id', 'asc')
            ->get();

        // 🔍 Obtener el rol del usuario autenticado
        $rolUsuario = UserRol::where('id_user', Auth::id())
            ->where('estado', 1)
            ->value('id_rol');

        return view('productos.index', compact('categorias, rolUsuario'));
    }
```

En la vista aplicamos la restricción:

```php
use Illuminate\Support\Facades\Auth;
use App\Models\UserRol;

.
.
.
.

        // 🔍 Obtener el rol del usuario autenticado
        $rolUsuario = UserRol::where('id_user', Auth::id())
            ->where('estado', 1)
            ->value('id_rol');

        return view('productos.index', compact('categorias, rolUsuario'));
```

```blade
{{-- dataTable JSON --}}
<script>
    // 🔒 Convertimos la variable PHP a JSON válido para JS
    const rolUsuario = @json($rolUsuario);

    $(function() {
        $('#productos').DataTable({
            ajax: '{{ route('producto.data') }}', // 🔷 Llama a la ruta JSON
            columns: [{
                    data: 'id'
                },
                {
                    data: 'nombre'
                },
                {
                    data: 'categoria_id'
                },
                {
                    data: 'codigo_barras',
                },
                {
                    data: 'precio_venta',
                },
                {
                    data: 'cantidad_stock'
                },
                {
                    "data": "id",
                    "render": function(data, type, row, meta) {

                        // 🚫 Si el usuario tiene rol 6 (editor de productos)
                        if (rolUsuario === 6) {
                            return ''; // No mostrar botones de editar/eliminar
                        }

                        var eliminarproducto =
                            '<button type="button" class="btn btn-danger btn-sm" onclick="eliminarProducto(' +
                            data + ')">' +
                            '<i class="fas fa-trash-alt"></i>' +
                            '</button>';

                        var editarproducto =
                            '<button type="button" class="btn btn-primary btn-sm" onclick="editarProducto(' +
                            data + ')">' +
```

```
<script>

// 🔒 Convertimos la variable PHP a JSON válido para JS
const rolUsuario = @json($rolUsuario);
.
.
.
.


{
      "data": "id",
      "render": function(data, type, row, meta) {

      // 🚫 Si el usuario tiene rol 6 (editor de productos)
      if (rolUsuario === 6) {
       return ''; // No mostrar botones de editar/eliminar
}
```

- Laravel tiene una directiva especial **@json()** que convierte una variable PHP en JSON válido para JavaScript.
- DataTables obtiene el id del producto (data).
- Ejecuta la función render().
- Dentro de esa función, antes de crear los botones, verificamos el rol de usuario de la variable **rolUsuario**

Rol de usuario	Valor de `rolUsuario`	Resultado en la tabla
Admin	2	✅ Ve botones editar y eliminar
Admin Productos	4	✅ Ve botones editar y eliminar
Editor Productos	6	🚫 No ve botones

Nota.

Corregir el Gate para que Editor Productos pueda ver la ruta Productos y volver a habilitarlo

TAREA: SEPARAR LAS RUTAS DEL RECURSO CLIENTE

Y APLICAR CONTROL DE ROLES PARA REPORTECLIENTES

🎯 Objetivo

Separar las rutas de **Route::resource('cliente', ClienteController::class)** en rutas individuales para tener control granular sobre los permisos, de modo que:

- Los usuarios con rol 2 (**Admin**) y rol 5 (**AdminClientes**) puedan acceder a todas las rutas del CRUD.
- Los usuarios con rol 7 (por ejemplo) o el que definamos como **ReporteClientes** solo puedan listar (index) y ver detalles (show), pero no crear, editar ni eliminar.

ROL	DESCRIPCIÓN	PERMISOS
2	Admin	✓ Crear, Editar, Eliminar, Ver, Listar
5	AdminClientes	✓ Crear, Editar, Eliminar, Ver, Listar
7 (por ejemplo)	ReporteClientes	✓ Listar, Ver ✗ No crear, editar ni eliminar