

.... GUIA DE LARAVEL 6 y 7

Aprende a crear el sistema de autenticación de laravel, controladores, rutas, migraciones, modelos, relaciones, seeders y sistema de roles y permisos con Gates y Políticas.

Autor:

“Freddy Alcarazo”

creatorpart@gmail.com

<https://github.com/alcarazolabs>

<https://www.youtube.com/channel/UCy6ttBvsjtroucqQocj3P7Q>

Agradecimiento Especial a:

“JHONATAN DAVID FERNANDEZ ROSA”

Fundador de:

<https://solibeth.net/>

<https://www.youtube.com/c/JHONATANDAVIDFERNANDEZROSA/videos>

“Por ser la fuente del conocimiento de esta obra”.



Aclaración:

En esta guía, obra, libro, bitácora como le quisiéramos llamar no encontraran una guía definitiva para desarrollar aplicaciones laravel, es una guía que guarda cosas técnicas de laravel. Es común que a los desarrolladores se nos olviden comandos, procedimientos para hacer algo en especial con el framework laravel ya que para hacer una aplicación necesitamos de conocer comandos, reglas para crear una aplicación. Si te pasa eso, esta guía es para ti. Guarda esta guía y cuando quieras volver a empezar esta será tu mejor aliada porque contiene lo fundamental para iniciarse en laravel.

En esta parte verás cómo instalar laravel, crear rutas, controladores, modelos. Todo lo que se hace en laravel 6 en esta guía aplica para laravel 7.

Instalar laravel 6

1. Instalar el programa “Laragon” el cual trae instalado npm, nodejs, composer, php, mysql entre otros como el servidor apache.

Página Descarga: <https://laragon.org/download/>

Como agregar phpMyAdmin a laragon:

<https://www.youtube.com/watch?v=whXdCZ9j0cM>

2. Abrir una terminal de laragon y ejecutar el comando:

```
<# composer global require laravel/installer
```

3. Para crear un proyecto ahora se ejecuta el comando

```
<# laravel new "nombre del proyecto"
```

3.1 Crear el proyecto con composer y especificar versión:

```
<# composer create-project laravel/laravel laravel6 "6.0.*"
```

3.2 Ejecutar el proyecto:

```
<# php artisan serve
```

4. SISTEMA DE AUTENTICACIÓN DE LARAVEL 6

En versiones anteriores laravel proveía el comando:

```
<# php artisan make:auth
```

De esa manera automáticamente laravel traia todo lo necesario para el sistema de autenticación. En laravel 6 se decidió que la parte autenticación y otra cosas como el frontend bootstrap, react, vue lo han eliminado y lo han puesto con una librería con composer dado a que laravel es más usado para el backend, al tener todo eso predefinido en el core hace pesado al proyecto, esto da la opción al usuario que decida si va usar el frontend con react, vue, bootstrap.. o

si solamente va a necesitar api rest.. entonces no tiene necesidad de instalar todos los componentes lo que ara que el proyecto sea grande entonces ahora laravel provee una librería **laravel/ui**:

```
# composer require laravel/ui (ejecutar dentro del proyecto)
```

```
C:\laragon\www\laravel6
λ composer require laravel/ui
Using version ^1.1 for laravel/ui
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing laravel/ui (v1.1.1): Downloading (100%)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
```

```
C:\laragon\www\laravel6
λ
```

- Ahora si escribimos el comando:

```
# php artisan
```

Aparece el “ui:auth”

```
storage:link      Create a symbolic link from "public/storage" to "storage/app/public"
ui
  ui:auth        Scaffold basic login and registration views and routes
  vendor
  vendor:publish Publish any publishable assets from vendor packages
  view
  view:cache     Compile all of the application's Blade templates
  view:clear     Clear all compiled view files
```

- Para ver lo que contiene escribimos el comando:
- `# php artisan ui -h`

```
C:\laragon\www\laravel6
λ php artisan ui -h
Description:
  Swap the front-end scaffolding for the application

Usage:
  ui [options] [--] <type>          Ahora si escribimos el comando:
                                                # php artisan

Arguments:
  type                         The preset type (bootstrap, vue, react)

Options:
  --auth                        Install authentication UI scaffolding
  --option[=OPTION]             Pass an option to the preset command (multiple values allowed)
  -h, --help                     Display this help message
  -q, --quiet                   Do not output any message
  -V, --version                 Display this application version
  --ansi                        Force ANSI output
  --no-ansi                     Disable ANSI output
  -n, --no-interaction          Do not ask any interactive question
  --env[=ENV]                   The environment the command should run under
  -v|vv|vvv, --verbose          Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

C:\laragon\www\laravel6
```

5. Instalar el sistema de autenticación en laravel 6

A partir de lo anterior entonces se decide instalar el sistema de autenticación en el proyecto con “ui” con “vue” para el frontend.

```
<#php artisan ui vue
```

```
C:\laragon\www\laravel6
λ php artisan ui vue
Vue scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.

C:\laragon\www\laravel6
λ
```

Ahora dice que ejecutemos:

```
<# npm install && npm run dev
```

Pero antes de ejecutar el comando si instalar el sistema de autenticación:

```
#php artisan ui:auth
```

```
C:\laragon\www\laravel6
λ php artisan ui:auth
Authentication scaffolding generated successfully.

C:\laragon\www\laravel6
λ
```

Lo cual nos instalará todos los archivos que son necesarios para que se observe el sistema de autenticación:

Si ejecutamos:

```
<#php artisan serve
```

Aparecer el login en el sistema:

Laravel

DOCS LARACASTS NEWS BLOG NOVA FORGE VAPOR GITHUB

[Laravel](#)

- [Login](#)
- [Register](#)

Login

E-Mail Address

Password

Remember Me

[Login](#) [Forgot Your Password?](#)

Por lo tanto para ver el proyecto compilado con los archivos del frontend como bootstrap etc..

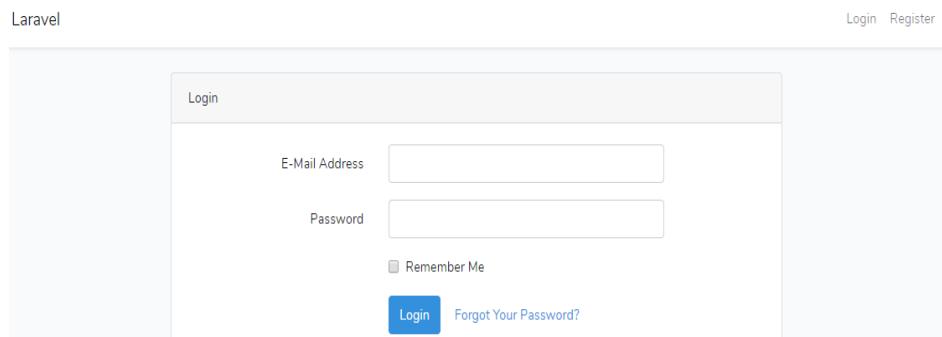
<# npm install && npm run dev

```
DONE Compiled successfully in 9493ms

      Asset      Size  Chunks      Chunk Names
/css/app.css   196 KiB  /js/app  [emitted]  /js/app
/js/app.js    1.39 MiB  /js/app  [emitted]  /js/app

C:\laragon\www\laravel6
```

<#php artisan serve



Register

Name

E-Mail Address

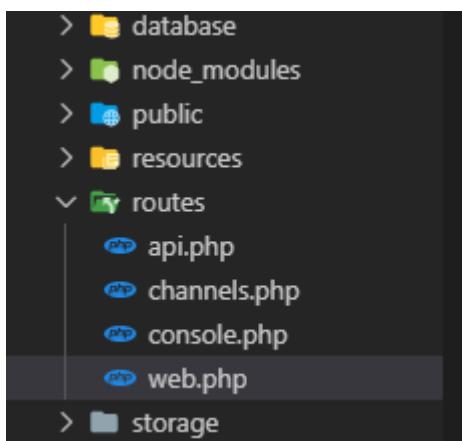
Password

Confirm Password

Register

6. Rutas

Dentro del proyecto ver carpeta “routes”

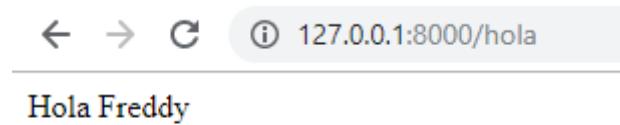


El archivo a editar de las rutas será **web.php**

6.1 Ejemplo de Rutas simple tipo GET:

```
Route::get('hola', function(){
    return 'Hola Freddy';
});
```

Resultado en el navegador:



6.2 Ruta con parámetro:

```
26     Route::get('usuario/{nombre}', function($nombre){  
27         return 'Usuario '.$nombre;  
28     });
```

← → C ⓘ 127.0.0.1:8000/usuario/freddy

Usuario freddy

Ruta con parámetro con valor por defecto:

```
Route::get('usuario/{nombre?}', function($nombre='Invitado'){  
    return 'Usuario '.$nombre;  
});
```

← → C ⓘ 127.0.0.1:8000/usuario

Usuario Invitado

Ruta con doble parámetro:

```
Route::get('usuario/{nombre}/comentario/{id}', function($nombre, $id){  
    return 'Usuario '.$nombre.' ID comentario: '.$id;  
});
```

← → C ⓘ 127.0.0.1:8000/usuario/freddy/comentario/2

Usuario freddy ID comentario: 2

Ruta con condición con expresión regular (solo caracteres del alfabeto)

```
Route::get('user/{nombre}', function($nombre){  
    return 'Usuario '.$nombre;  
})->where('nombre', '[A-Za-z]+');
```

← → C ⓘ 127.0.0.1:8000/user/fred

Usuario fred

Ruta con condición de parámetro numérico:

```
Route::get('user1/{id}', function($id){  
    return 'Usuario ID='.$id;  
})->where('id', '[0-9]+');
```

← → ⌂ ⓘ 127.0.0.1:8000/user1/1

Usuario ID=1

En caso de no ser numérico retorna un 404.

404 | Not Found

Ruta con dos parámetros validados (numérico y alfabetico)

```
Route::get('user2/{id}/{nombre}', function($id,$nombre){  
    return 'Usuario ID='.$id.' Nombre: '.$nombre;  
})->where(['id' => '[0-9]+',  
           'nombre' => '[A-Za-z]+']);
```

← → ⌂ ⓘ 127.0.0.1:8000/user2/99/fred

Usuario ID=99 Nombre: fred

Ruta que redirecciona a otra ruta:

```
47 Route::get('prueba', function(){  
48     return 'Pagina de prueba';  
49 })->name('pruebaruta');  
50  
51 Route::get('redirigir', function(){  
52     return redirect()->route('pruebaruta');  
53 });
```

- En este caso la ruta llamada “redirigir” redirecciona a la ruta ‘prueba’, esta ruta tiene como nombre el alias ‘pruebaruta’ el cual es usado dentro de la ruta ‘redirigir’ para redireccionar.

Redireccionar a una ruta que recibe parámetro:

```
53     Route::get('usuario/{nombre?}', function($nombre='Invitado'){
54         return 'Usuario '.$nombre;
55     })->name('usuarioruta');
56
57     Route::get('redirigir2', function(){
58         return redirect()->route('usuarioruta', ['nombre'=>'freddy']);
59     });
60
```

Vemos que desde la ruta “redirigir2” se redirecciona a la ruta ‘usuarioruta’ pero se le envía un parámetro.

Redireccionamiento simple:

```
Route::redirect(['prueba3', 'prueba']);
```

Se redirecciona desde la ruta “prueba3” a la ruta “prueba”.

7. Controladores

Para crear controladores se usa el siguiente comando:

```
<# php artisan make:controller NombreController
```

```
C:\laragon\www\laravel6
λ php artisan make:controller UsuarioController
Controller created successfully.
```

Luego en la carpeta app->http->controller esta creado el controlador:

✓	laravel6	4
✓	app	4
>	Console	4
>	Exceptions	4
✓	Http	5
✓	Controllers	5
>	Auth	5
>	Controller.php	5
>	HomeController.php	5
>	UsuarioController.php	5

Ejemplo Controlador y Ruta:

```
52
53 Route::get('usuario/{nombre?}', 'UsuarioController@usuariounparametro')->name('usuarioruta');
54

7 class UsuarioController extends Controller
8 {
9     public function usuariounparametro($nombre='invitado'){
10         return 'Usuario '.$nombre;
11     }
12 }
```

← → ⌂ ⓘ 127.0.0.1:8000/usuario/freddy

Usuario freddy

Controlador con dos parámetros:

```
public function usuariodosparametros($nombre, $id){  
    return 'Usuario '.$nombre.' ID comentario: '.$id;  
}
```

← → ⌂ ⓘ 127.0.0.1:8000/usuario/freddy/comentario/33

Usuario freddy ID comentario: 33

Crear controlador que no recibe ningún parámetro solo retorna algo:

```
<# php artisan make:controller holaController –invokable
```

```
C:\laragon\www\laravel6
λ php artisan make:controller holaController --invokable
Controller created successfully.
```

```
6
7     class holaController extends Controller
8     {
9         /**
10          * Handle the incoming request.
11          *
12          * @param  \Illuminate\Http\Request  $request
13          * @return \Illuminate\Http\Response
14          */
15         public function __invoke(Request $request)
16         {
17             //
18         }
19     }
20 }
```

La ruta que va a utilizar dicho controlador quedaría de la siguiente manera:

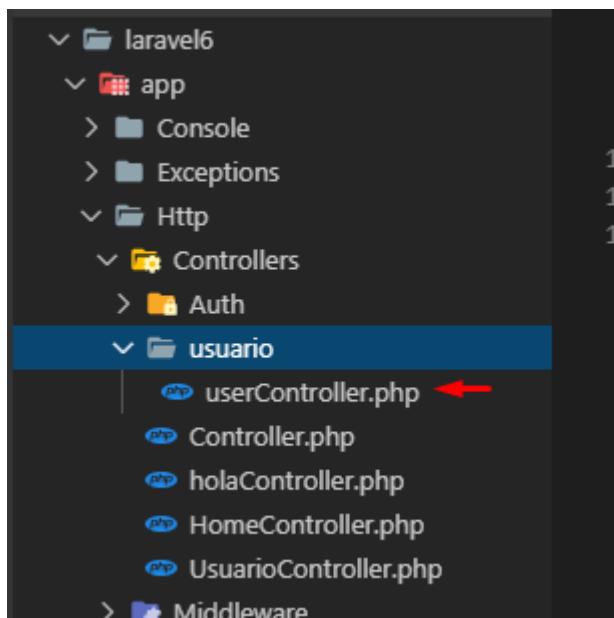
```
22     Route::get('hola', 'holaController');
23 }
```

```
7     class holaController extends Controller
8     {
9         /**
10          * Handle the incoming request.
11          *
12          * @param  \Illuminate\Http\Request  $request
13          * @return \Illuminate\Http\Response
14          */
15         public function __invoke(Request $request)
16         {
17             return 'Hola Freddy';
18         }
19     }
```

Crear controlador dentro de una carpeta:

```
C:\laragon\www\laravel6
λ php artisan make:controller usuario/userController
Controller created successfully.
```

Se crea la carpeta usuario y el controlador userController



Ruta con controlador que esta dentro de una carpeta:

```
23
24     Route::get('user/{nombre}', 'usuario\userController@user')->where('nombre', '[A-Za-z]+');
```

- En este caso “usuario” es la carpeta y el resto luego de “\” es el controlador seguido de @ para especificar la función en este caso es user.

```
3 class UserController extends Controller
4 {
5     public function user($nombre){
6         return 'Usuario '.$nombre;
7     }
8 }
```

← → ⌂ ⓘ 127.0.0.1:8000/user/freddy

Usuario freddy

Otros ejemplos:

Rutas que llaman al mismo controlador, pero con diferente método:

```
24 Route::get('user/{nombre}', 'usuario\userController@user')->where('nombre', '[A-Za-z]+');  
25  
26 Route::get('user1/{id}', 'usuario\userController@user1')->where('id', '[0-9]+');  
27  
28 Route::get('user2/{id}/{nombre}', 'usuario\userController@user2')->where(['id' => '[0-9]+',  
29 | | | 'nombre' => '[A-Za-z]+']);  
30
```

```
class userController extends Controller  
{  
    public function user($nombre){  
        return 'Usuario '.$nombre;  
    }  
    public function user1($id){  
        return 'User ID: '.$id;  
    }  
    public function user2($id,$nombre){  
        return 'Usuario ID='.$id.' Nombre: '.$nombre;  
    }  
}
```

Controladores tipo Resource

Para crear este tipo de controladores se ejecuta el siguiente comando:

```
C:\laragon\www\laravel6  
λ php artisan make:controller variosmetodoscursos --resource  
Controller created successfully.
```

```

6
7     class variosmetodoscursos extends Controller
8     {
9         /**
10          * Display a listing of the resource.
11          *
12          * @return \Illuminate\Http\Response
13          */
14         public function index()
15         {
16             //
17         }
18
19         /**
20          * Show the form for creating a new resource.
21          *
22          * @return \Illuminate\Http\Response
23          */
24         public function create()
25         {
26             //
27         }
28
29         /**

```

Este tipo de controladores crea varios métodos como index, create, show, update, edit y destroy.

Creando la ruta:

```

51
52     Route::resource('varios', 'variosmetodoscursos');
53

```

Listando las rutas:

web		varios	varios.index	App\Http\Controllers\variosmetodoscursos@index
GET HEAD		varios	varios.store	App\Http\Controllers\variosmetodoscursos@store
web		varios/create	varios.create	App\Http\Controllers\variosmetodoscursos@create
POST		varios/{vario}	varios.show	App\Http\Controllers\variosmetodoscursos@show
web		varios/{vario}/edit	varios.update	App\Http\Controllers\variosmetodoscursos@update
PUT PATCH		varios/{vario}/destroy	varios.destroy	App\Http\Controllers\variosmetodoscursos@destroy
DELETE		varios/{vario}/edit	varios.edit	App\Http\Controllers\variosmetodoscursos@edit
web				

Se aprecia que para acceder a la ruta “varios” se puede acceder desde

- varios/create
- varios/

Pero si deseamos solo que se acceda a determinadas operaciones del controlador como por ejemplo, solo que se acceda a varios.index y a varios.show la ruta se crea de la siguiente manera:

```
53 Route::resource('varios1','variosmetodoscursos')->only(['index','show']);
```

	web	varios	varios.index	App\Http\Controllers\variosmetodoscursos@index
GET HEAD	varios	varios	varios.store	App\Http\Controllers\variosmetodoscursos@store
POST		varios/create	varios.create	App\Http\Controllers\variosmetodoscursos@create
GET HEAD		varios/{vario}	varios.show	App\Http\Controllers\variosmetodoscursos@show
PUT PATCH		varios/{vario}	varios.update	App\Http\Controllers\variosmetodoscursos@update
DELETE		varios/{vario}	varios.destroy	App\Http\Controllers\variosmetodoscursos@destroy
GET HEAD		varios/{vario}/edit	varios.edit	App\Http\Controllers\variosmetodoscursos@edit
GET HEAD		varios1	varios1.index	App\Http\Controllers\variosmetodoscursos@index
GET HEAD		varios1/{varios1}	varios1.show	App\Http\Controllers\variosmetodoscursos@show

Otra manera de omitir el uso de algunas funciones del controlador de tipo Resource es hacer uso de la regla “except”:

```
53
54 Route::resource('varios2','variosmetodoscursos')->except(['create','store','index','destroy','edit']);
```

Luego listando las rutas:

```
# php artisan route:list
```

	varios1	varios1.index	App\Http\Controllers\variosmetodoscursos@index
	php artisan route:list		
	varios1/{varios1}	varios1.show	App\Http\Controllers\variosmetodoscursos@show
	varios2/{varios2}	varios2.show	App\Http\Controllers\variosmetodoscursos@show
	varios2/{varios2}	varios2.update	App\Http\Controllers\variosmetodoscursos@update

Vemos que solo esta disponible “show” y “update”

También se puede renombrar las funciones como por ejemplo, no se desea que se llame “varios.index” si no “varios.inicio”

```
55
56 Route::resource('varios3','variosmetodoscursos')->only(['index','show'])->names([
57   'index' => 'varios.inicio'
58 ]);
```

	varios2/{varios2}	varios2.update	App\Http\Controllers\variosmetodoscursos@update
	varios3	varios.inicio	App\Http\Controllers\variosmetodoscursos@index
	varios3/{varios3}	varios3.show	App\Http\Controllers\variosmetodoscursos@show

Redireccionando desde el controlador

varios	varios.store	App\Http\Controllers\variosmetodoscursos@store
varios	varios.index	App\Http\Controllers\variosmetodoscursos@index
varios/create	varios.create	App\Http\Controllers\variosmetodoscursos@create
varios/{vario}	varios.show	App\Http\Controllers\variosmetodoscursos@show
varios/{vario}	varios.update	App\Http\Controllers\variosmetodoscursos@update
varios/{vario}	varios.destroy	App\Http\Controllers\variosmetodoscursos@destroy
varios/{vario}/edit	varios.edit	App\Http\Controllers\variosmetodoscursos@edit

```
|    "
public function index()
{
    return redirect()->action('holaController');
```

Cuando se acceda 127.0.0.1/varios lo que se ará es una redirección al controlador ‘holaController’

- Redireccionar a controlador que recibe un parámetro:

```
public function index()
{
    return redirect()->action('UsuarioController@usuariounparametro',['nombre'=>'Fredd Alc']);
```

Vemos que se a pasado el valor del parámetro ‘nombre’ como ‘freddy alc’ a el controlador UsuarioController en el método usuariounparámetro.

- Redireccionar a una URL

```
public function index()
{
    //return redirect()->action('UsuarioController@usuariounparametro',['nombre'=>'Fredd Alc']);
    return redirect('hola');
```

Se redirecciona a la ruta ‘hola’

Redireccionar a Google.com

```
public function index()
{
    //return redirect()->action('UsuarioController@usuariounparametro',['nombre'=>'Fredd Alc']);
    return redirect('http://www.google.com');
```

Migraciones

Es el mecanismo para administrar una tabla específica (crear campos, índices, vistas, llaves primarias, eliminar llaves primarias etc) El modelo es el mecanismo para acceder a los datos que están contenidos en una tabla.

Hacer migración:

1. Configurar la base de datos en el archivo .env

```
9  DB_CONNECTION=mysql  
10 DB_HOST=127.0.0.1  
11 DB_PORT=3306  
12 DB_DATABASE=laravel6  
13 DB_USERNAME=root  
14 DB_PASSWORD=toor  
15
```

2. Ejecutar php artisan migrate

```
C:\laragon\www\laravel6  
λ php artisan migrate  
Migration table created successfully.  
Migrating: 2014_10_12_000000_create_users_table  
Migrated: 2014_10_12_000000_create_users_table (0.75 seconds)  
Migrating: 2014_10_12_100000_create_password_resets_table  
Migrated: 2014_10_12_100000_create_password_resets_table (0.6 seconds)  
Migrating: 2019_08_19_000000_create_failed_jobs_table  
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.35 seconds)
```

Con migrate lo que se hace es buscar la carpeta de migraciones e inicia a subir cada archivo a la bd..

Crear migración “tabla”

```
C:\laragon\www\laravel6  
λ php artisan make:migration create_notas_table  
Created Migration: 2019_12_07_214500_create_notas_table
```

El comando:

```
<# php artisan make:migration create_notas_table
```

Debe de siempre tener la siglas create_ y el nombre de la tabla en plural y al final _table

- Modificando la tabla:

```
{  
    Schema::create('notas', function (Blueprint $table) {  
        $table->bigIncrements('id');  
        $table->string('titulo');  
        $table->string('detalle');  
        $table->timestamps();  
    });  
}
```

La propiedad **bigIncrements** crea una columna de llave primaria y **autoincremental**, el **timestamps** crea una fecha de inicio y fecha de actualización, el **string** es equivalente a una columna varchar.

- Ejecutar migración otra vez y ver los cambios en la base de datos:

```
C:\laragon\www\laravel6  
λ php artisan migrate  
Migrating: 2019_12_07_214500_create_notas_table  
Migrated: 2019_12_07_214500_create_notas_table (0.26 seconds)  
C:\laragon\www\laravel6
```

```
SELECT * FROM `notas`
```

	id	titulo	detalle	created_at	updated_at
--	-----------	---------------	----------------	-------------------	-------------------

- En caso de no se pueda hacer la migración poner el comando:
- # php artisan migrate –force

Tambien se puede hacer rollback es decir volver a un estado anterior o eliminar migraciones, para hacer un rollback seguir los sgts pasos:

Rollback para la ultima migración:

```
# php artisan migrate:rollback
```

Rollback para las ultimas 5 migraciones;

```
# php artisan migrate:rollback –step=5
```

Rollback para eliminar todas las tablas:

```
# php artisan migrate:reset
```

Rollback para resfrescar migraciones (elimina y vuelve a migrar):

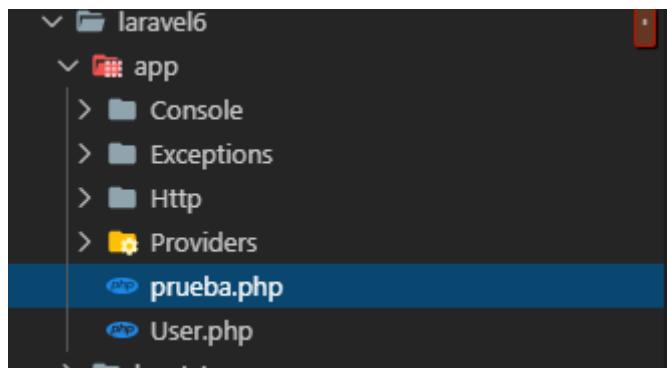
```
# php artisan migrate:refresh
```

Modelos y Eloquent ORM

Los modelos son archivos php que tiene una clase, la idea de los modelos es poder acceder a datos de una tabla x. Eloquent ORM es el mecanismo que usan los modelos para poder recoger la información o poder crear datos en una tabla.

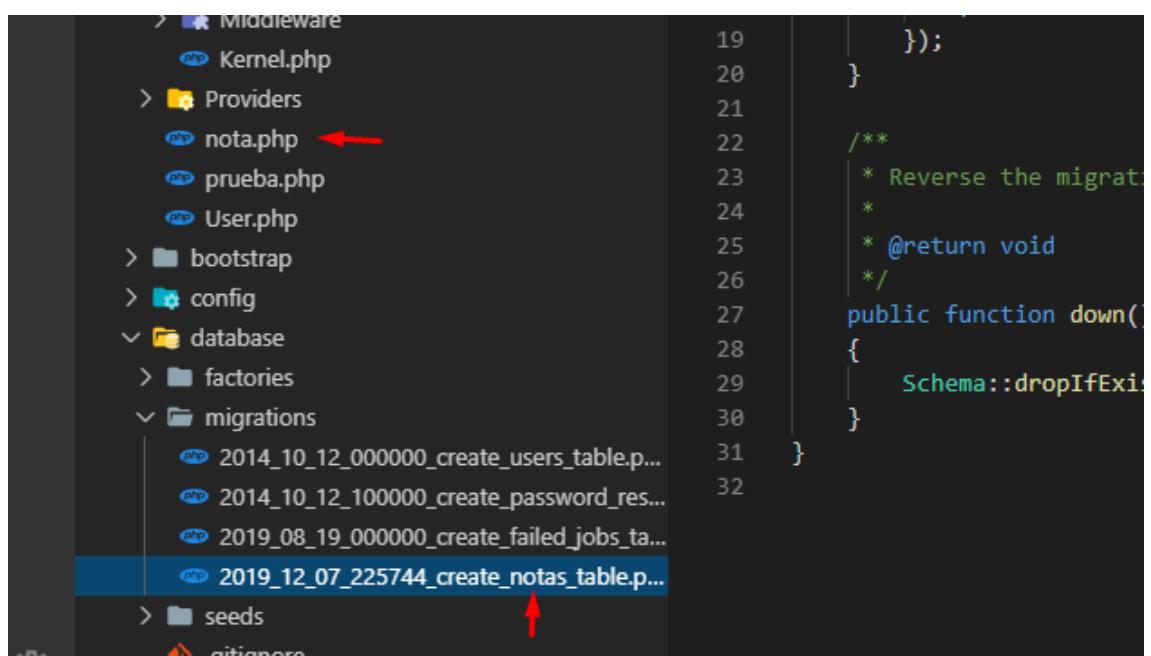
Crear un modelo:

```
<# php artisan make:model prueba
```



Crear modelo y migración al mismo tiempo:

```
<# php artisan make:model nota -m
```



Registrar datos:

Dentro del controlador:

```
6  use App\info;
7
27 |     public function create()
28 |     {
29 |         $info = new info;
30 |         $info->nombre='Freddy Alcarazo';
31 |         $info->descripcion='Supervisor';
32 |         $info->save();
33 |
34 |         info::create([
35 |             'nombre'=>'Daniel',
36 |             'descripcion'=> 'Gerente TI'
37 |         ]);
38 |
39 |         return 'Datos guardados correctamente';
40 |
41 |     }
```

Para poder ejecutar la segunda forma dentro del modelo se debe de agregar la propiedad fillable:

```
7  class info extends Model
8  {
9      protected $fillable = ['nombre', 'descripcion'];
10 }
```

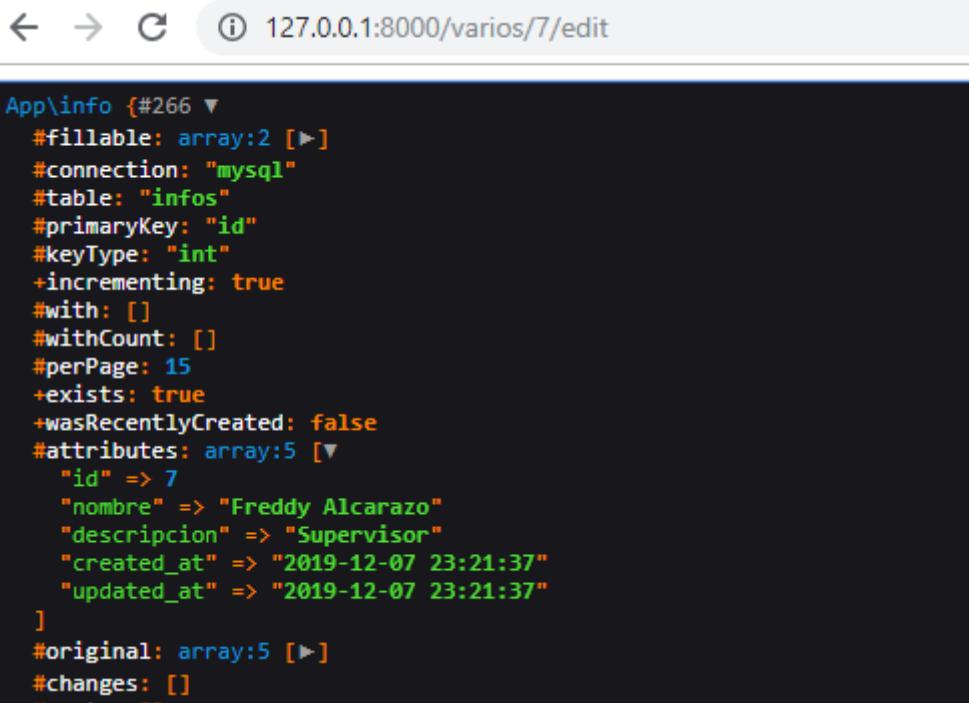
Mostrar registros modo debugger:

```
1
2     public function index()
3     {
4         //return redirect()->action('UsuarioController@index');
5         // return redirect('http://www.google.com');
6
7         $info = info::all();
8         dd($info);
9     }
10
```

```
Illuminate\Database\Eloquent\Collection {#264 ▶
  #items: array:2 [▶]
}
```

Buscar registros por ID:

```
public function edit($id)
{
    $info = info::find($id);
    dd($info);
    return 'Hola';
}
```



```
App\info {#266 ▾
  #fillable: array:2 [▶]
  #connection: "mysql"
  #table: "infos"
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  #perPage: 15
  +exists: true
  +wasRecentlyCreated: false
  #attributes: array:5 [▼
    "id" => 7
    "nombre" => "Freddy Alcarazo"
    "descripcion" => "Supervisor"
    "created_at" => "2019-12-07 23:21:37"
    "updated_at" => "2019-12-07 23:21:37"
  ]
  #original: array:5 [▶]
  #changes: []
  #casts: []
```

Buscar y editar nombre:

```
/*
public function edit($id)
{
    // $info = info::find($id);
    // $info = info::findOrFail($id); // retorna 404 si no existe
    // $info = info::where('id', $id)->first();
    $info = info::where('id', $id)->firstOrFail();
    $info->nombre = 'Rosa';
    $info->save();
    return 'actualizado';
    // dd($info);
    // return print_r($info);
}
```

Busqueda tiepo Like:

```
$info::where('campo', 'Like', "%$name%");
```

Eliminar registro:

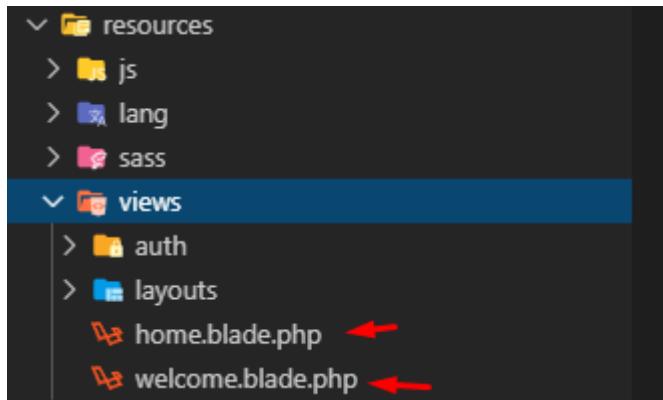
```
public function show($id)
{
    $info = info::where('id',$id)->firstOrFail();
    $info->delete();
    return 'Eliminado correctamente';
}
```

Views y Blade

- Vistas: Mecanismo para mostrar las informaciones.
- Blade: Motor de plantillas de laravel para tratar las vistas y provee ciertas funcionalidades como evitar inyección de código.

<https://laravel.com/docs/6.x/blade>

Las vistas se vienen guardando dentro de la carpeta “RESOURCES”



Desde las rutas para retornar una vista se hace de la sgt manera:

```
13
14     Route::get('/', function () {
15         return view('welcome');
16     });
17     /*
```

Como se aprecia en la línea 15 la ruta “/” retorna la vista welcome.

Crear una vista: Simplemente se crea un nuevo archivo dentro de la carpeta views sin olvidar extensión blade.php

Pasar datos a una vista:

Desde el controlador:

```
public function index()
{
    //return redirect()->action('UsuarioController@usu
    // return redirect('http://www.google.com');

    $info = info::all();
    return view('varios')->with(['info',$info]);
    // dd($info);
}
```

En la Vista:

The screenshot shows a code editor with three tabs: 'web.php', 'varios.blade.php', and 'varios'. The 'varios.blade.php' tab is active, displaying the following code:

```
laravel6 > resources > views > varios.blade.php
1 @foreach($info as $infoitem)
2 | <p> {{ $infoitem->nombre }} </p>
3 @endforeach
..
```

Resultado:

Daniel

Freddy

Amanda

Varias formas de pasar variables a las vistas:

```
$info = info::all();
//return view('varios')->with('info',$info);
//return view('varios')->with(['info'=>$info]);
//return view('varios', ['info'=>$info]);
return view('varios', compact('info'));
```

Directa if:

```
/resources/views/varios.blade.php
@foreach($info as $infoitem)
    @if($infoitem->nombre=='Amanda')
        <p> {{ $infoitem->nombre }} - {{ $infoitem->descripcion }} * Usuario Destacado</p>
    @else
        <p> {{ $infoitem->nombre }} - {{ $infoitem->descripcion }}</p>
    @endif
@endforeach
```

Verificar si esta vacío:

```
8 |     @empty($infoitem->descripcion)
9 |         <p>Sin descripción</p>
10|     @endempty
..|
```

Extender en Blade

Primero se debe de poner `@yield` para especificar las variables que se van a cambiar dinámicamente. En este caso se agregaron dentro una plantilla:

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css">
<head>
    <title>@yield('titulo','plantilla')</title>
</head>
<body>
    <div>
        <nav>
            @yield('contenido')
        <!-- Optional JavaScript -->
```

Para extender la plantilla desde otra vista se hace uso de `extends` y para cambiar el valor de las directivas `@yield` se hace con `@section`

```
vel6 > resources > views > varios > create.blade.php
1 @extends('plantilla.plantilla')
2 @section('contenido')
3     <h1>archivo Create</h1>
4 @endsection
5
6 @section('titulo','Create')
```

Proyecto CRUD agenda telefónica:

1. Crear proyecto.

```
composer create-project laravel/laravel agendatelefónica "6.0.*"
```

2. Crear sistema de autenticación de laravel

```
<# php artisan ui vue  
<# npm install && npm run dev  
<# php artisan ui:auth
```

3. Crear migración y modelo

```
<# php artisan make:model Agenda -m
```

4. Crear controlador

```
<# php artisan make:controller AgendaController --resource
```

```
<# php artisan route:list
```

web				
web	POST	agenda	agenda.store	App\Http\Controllers\AgendaController@store
web	GET HEAD	agenda	agenda.index	App\Http\Controllers\AgendaController@index
web	GET HEAD	agenda/create	agenda.create	App\Http\Controllers\AgendaController@create
web	DELETE	agenda/{agenda}	agenda.destroy	App\Http\Controllers\AgendaController@destroy
web	PUT PATCH	agenda/{agenda}	agenda.update	App\Http\Controllers\AgendaController@update
web	GET HEAD	agenda/{agenda}	agenda.show	App\Http\Controllers\AgendaController@show
web	GET HEAD	agenda/{agenda}/edit	agenda.edit	App\Http\Controllers\AgendaController@edit

Proyecto CRUD Productos:

Este proyecto se realiza con llaves foráneas.

1. Crear el proyecto

```
<# composer create-project laravel/laravel agendatelefónica "6.0.*"
```

2. Crear migraciones y modelos:

Se crean las migraciones y el modelo en orden, dado a que se van a crear llaves foráneas, primero se ha de crear el modelo y la migración de la tabla “Categorías” y luego “Productos”

```
C:\laragon\www\Productos  
λ php artisan make:model Categoría -m ←  
Model created successfully.  
Created Migration: 2019_12_14_214944_create_categorias_table  
  
C:\laragon\www\Productos  
λ php artisan make:model Producto -m ←  
Model created successfully.  
Created Migration: 2019_12_14_215029_create_productos_table
```

Por lo tanto de lo anterior se crearon los dos modelos y sus migraciones o tablas respectivas.

3. Crear Controlador

Se crea un controlador de tipo resource dado a que se va hacer un crud y este se caracteriza por traer varios métodos como edit, destroy, watch entre otros definidos..

```
C:\laragon\www\Productos
λ php artisan make:controller ProductosController --resource
Controller created successfully.
```

Lo siguiente es definir la ruta para enlazar al controlador, luego se diseña una plantilla dentro de resources o en las vistas mejor dicho la cual ha de ser extendida por las vistas de index, editar etc..

Ruta general:

```
Route::resource('productos', 'ProductosController');
```

Tablas Categoría:

```
public function up()
{
    Schema::create('categorias', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('nombre');
        $table->timestamps();

        $table->engine = 'InnoDB';
    });
}
```

Tabla Productos:

```
{
    Schema::create('productos', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('nombre',50);
        $table->decimal('precio',8,2);
        $table->integer('cantidad');

        $table->unsignedInteger('categoria_id');
        $table->foreign('categoria_id')->references('id')->on('categorias')->onDelete('cascade');
        $table->timestamps();

        $table->engine = 'InnoDB';
    });
}
```

En caso de tener problemas para crear la tablas con la base de datos se agrega lo siguiente en el archivo database.php que esta dentro de Config:

```
Config->database.php
'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
```

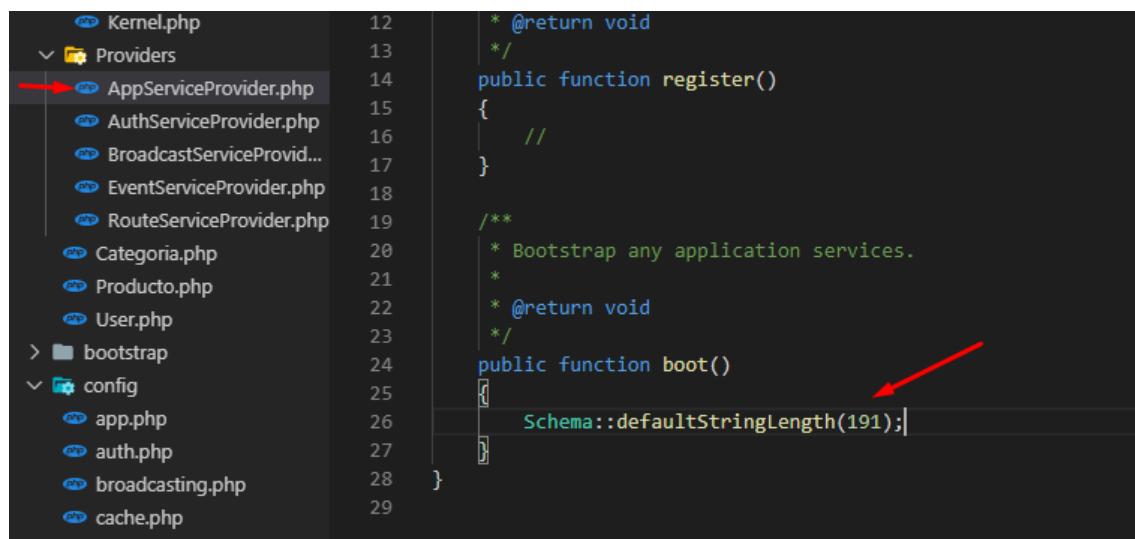
Se cambio por esto:

```
'charset' => 'utf8',
'collation' => 'utf8_unicode_ci',
```

En caso de que salte un error con que para el campo 'name' de la tabla users es muy grande su longitud se agrega lo siguiente en AppServiceProvider:

```
use Illuminate\Support\Facades\Schema;
```

```
Schema::defaultStringLength(191);
```



```
Kernel.php
  Providers
    AppServiceProvider.php
    AuthServiceProvider.php
    BroadcastServiceProvider...
    EventServiceProvider.php
    RouteServiceProvider.php
    Categoria.php
    Producto.php
    User.php
  bootstrap
  config
    app.php
    auth.php
    broadcasting.php
    cache.php
```

```
12   * @return void
13   */
14   public function register()
15   {
16       //
17   }
18
19   /**
20   * Bootstrap any application services.
21   *
22   * @return void
23   */
24   public function boot()
25   {
26       Schema::defaultStringLength(191);
27   }
28 }
```

Relacion uno a muchos:

En este caso se deseó obtener el nombre de las categorías que pertenecen a un producto, para ello se agrego dentro del modelo Producto lo siguiente:

```
class Producto extends Model
{
    /*
    public function categoria() {

        return $this->belongsTo(App\Categoría::class);

    }*/
    public function categoria () {
        return $this->belongsTo('App\Categoría', 'categoria_id');
    }
}
```

Finalmente, en la vista dentro del ciclo foreach se obtuvo el nombre de la categoría de los productos de la siguiente manera:

```
<tbody>
@foreach($productos as $producto)
<tr>
    <th scope="row">{{$producto->id}}</th>
    <td>{{$producto->nombre}}</td>
    <td>{{$producto->precio}}</td>
    <td>{{$producto->cantidad}}</td>
    <td>{{$producto->categoría->nombre}}</td>
<td>
    <a href="{{ route('producto.edit',$producto->id)}}><i class="fa fa-edit"></i> Editar </a>
    <a href="{{ route('producto.confirm',$producto->id)}}><i class="fa fa-trash-alt"></i> Eliminar </a>
</td>
</tr>
@endforeach
</tbody>
</table>
```

Roles y Permisos – Laravel 7.

Fuente:

<https://www.youtube.com/watch?v=bFJDOIYh4ic&list=PLtg6DxcGyHSvB6xvQbacVfL83UoFEvOGz>

1. Crear Proyecto

Ejecutar comando:

```
#> composer create-project laravel/laravel laravel6 "7.0.*"
```

Crear base de datos “rolespermisos”.

Realizar migración:

```
#> php artisan migrate
```

2. Instalar sistema de autenticación

```
#> composer require laravel/ui
```

```
#> php artisan ui vue --auth
```

```
#> npm install && npm run dev
```

3. Crear relación entre el modelo “roles” y “usuario”

❖ Crear modelo “roles”:

```
#> php artisan make:model FreddyPermisos/Models/Role -m
```

El comando de arriba crea dos carpetas (FreddyPermisos y dentro de esta Models) y dentro de “models” el modelo “Role” y con el comando “-m” se indica que cree la migración.

❖ Crear tabla “roles”:

```
public function up()
{
    Schema::create('roles', function (Blueprint $table) {
        $table->id();
        $table->string('name')->unique();
        $table->string('slug')->unique();
        $table->text('description')->nullable();
        $table->enum('full-access',['yes','no'])->nullable();
        $table->timestamps();
    });
}
```

Crear tabla intermedia, debido a que se va utilizar la relación “muchos-a-muchos” entre la tabla “usuario” y la tabla “roles”. Esto permitirá a nivel de programación seleccionar si el “usuario” tendrá un solo rol o varios roles.

La tabla intermedia no lleva un modelo como tal, solo es una tabla para conectar la tabla “usuario” y “roles”.

```
#> php artisan make:migration create_role_user_table
```

Esta convención la idea es la siguiente: para crear la tabla intermedia se debe saber de cual de los dos modelos, la primera letra es “menor”, en este caso la que es menor es “r” del modelo “Role”. Si hubiésemos tenido el modelo “Roles” y “Usuarios” seria:

```
#> php artisan make migration create_roles_usuarios_table
```

Por lo tanto, se debe de tener en cuenta si los nombres de los modelos están en singular o en plural y tener en cuenta cuál de ellos es menor en la letra inicial para ponerlos en orden en la línea de comando.

Fuente: https://youtu.be/UI-6MkA5_Zs

Captura de pantalla del modelo-tabla intermedia creada:

```
public function up()
{
    Schema::create('role_user', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

❖ **Modificar la tabla “role_user” para agregar las llaves foráneas:**

Primero agregar:

```
$table->foreignId('role_id')->references('id')->on('roles')->onDelete('cascade')
```

Esto significa: ‘role_id’ es la llave foránea la cual primero tiene el nombre del “modelo” “role” en minúscula – el modelo creado se llama “Role.php” – seguido de “underscore” “id” se pone “id” por que hace referencia a que la tabla/migración “roles” tiene por defecto un “id” para eso se puso “references” id “on-en” roles. Se puso onDelete cascade.

De igual forma se creó la llave foránea de la tabla **users**.

```
Schema::create('role_user', function (Blueprint $table) {
    //esta tabla tiene la union de dos tablas
    $table->id();
    $table->foreignId('role_id')->references('id')->on('roles')->onDelete('cascade');
    $table->foreignId('user_id')->references('id')->on('users')->onDelete('cascade');
    $table->timestamps();
```

Seguidamente ya se puede hacer “php artisan migrate”:

```
#> php artisan migrate
```

Luego de terminar el proyecto, la idea es copiar las migraciones a la carpeta FreddyPermisos/Models y crear otra carpeta llamada “migrations” para poner todos esos datos.

❖ **Probar que las relaciones “roles,usuarios y tabla intermedia” estén correctas:**

Ir al modelo “Role” tal como se hace en el modelo usuario, agregar:

```
class Role extends Model
{
    //es. desde aqui
    //en. from here
    protected $fillable = [
        'name', 'slug', 'description','full-access',
    ];
}
```

Desde aquí comienza el código.

Esto sirve para agregar de manera masiva los datos.

En el mismo archivo crear la función pública “users()”:

La idea de esta función es que cuando se seleccione un “rol” traiga todos los usuarios asociados con dicho rol.

```
class Role extends Model
{
    //es. desde aqui
    //en. from here
    protected $fillable = [
        'name', 'slug', 'description','full-access',
    ];
    public function users(){
        //relación muchos a muchos.
        return $this->belongsToMany('App\User')->withTimestamps();
    }
}
```

De la misma manera se hace en el modelo “Usuario” se crea la función “roles” para poder acceder a todos los “roles”.

```
];
//es. desde aqui
//en. from here
public function roles(){
    //relación muchos a muchos.
    return $this->belongsToMany('App\FreddyPermisos\Models\Role')->withTimestamps();
}
}
```

El namespace es diferente “tener en cuenta eso”.

❖ **Crear ruta “pruebas” para hacer pruebas:**

Primero agregar los namespace:

```
RolesPermisos > routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  use App\FreddyPermisos\Models\Role;
6  use App\User;
7  /*
8  | -----
9  | Web Routes
```

Crear rol de prueba:

```
Route::get('/test', function (){
    return Role::create([
        'name' => 'Admin',
        'slug' => 'admin',
        'description' => 'Administrator',
        'full-access' => 'yes',
    ]);
});
```

Resultado en el navegador:

```
1 // 20200908153236
2 // http://localhost:8000/test
3
4 {
5     "name": "Admin",
6     "slug": "admin",
7     "description": "Administrator",
8     "full-access": "yes",
9     "updated_at": "2020-09-08T20:32:36.000000Z",
10    "created_at": "2020-09-08T20:32:36.000000Z",
11    "id": 1
12 }
```

Se aprecia que retorna el “id” 1 que significa que se creo el rol.

Crear rol “guest” de prueba:

```
34     return Role::create([
35         'name' => 'Guest',
36         'slug' => 'guest',
37         'description' => 'guest',
38         'full-access' => 'no',
39     ]);
40 }
```

Se observa que se creo el rol correctamente:

```
{
    "name": "Guest",
    "slug": "guest",
    "description": "guest",
    "full-access": "no",
    "updated_at": "2020-09-08T20:36:21.000000Z",
    "created_at": "2020-09-08T20:36:21.000000Z",
    "id": 2
}
```

Se crea un tercer role de prueba:

```
]);*/
return Role::create([
    'name' => 'Test',
    'slug' => 'test',
    'description' => 'test',
    'full-access' => 'no',
]);
});
```

De la misma manera en el navegador se observó que se creo.

```
4  {
5      "name": "Test",
6      "slug": "test",
7      "description": "test",
8      "full-access": "no",
9      "updated_at": "2020-09-08T20:38:19.000000Z",
10     "created_at": "2020-09-08T20:38:19.000000Z",
11     "id": 3
12 }
```

Retornar el usuario número 1 en la misma ruta:

```
$user = User::find(1);
return $user;
});

{
    "id": 1,
    "name": "Freddy Alcarazo",
    "email": "freddyalcarazo@hotmail.com",
    "email_verified_at": null,
    "created_at": "2020-09-08T16:22:59.000000Z",
    "updated_at": "2020-09-08T16:22:59.000000Z"
}
```

Si hacemos lo siguiente para obtener los roles del usuario 1:

```
$user = User::find(1)->roles;
return $user;
}.
```

El resultado es que “el usuario no tiene roles”:

```
1 // 20200908154617
2 // http://localhost:8000/test
3
4 [
5
6 ]
```

Asignarle roles a al usuario con “attach”. Attach es usado para relaciones “muchos a muchos”. El siguiente ejemplo agrega los roles 1 y 3 al usuario con id=1:

```
$user = User::find(1);
$user->roles()->attach([1,3]);

return $user->roles;
});
```

La opción “attach” tiene la desventaja de repetir los roles, si recargamos varias veces la ruta “test” en el navegador se agregarán los mismos roles repetidos.

```

{
  "id": 1,
  "name": "Admin",
  "slug": "admin",
  "description": "Administrator",
  "full-access": "yes",
  "created_at": "2020-09-08T20:32:36.000000Z",
  "updated_at": "2020-09-08T20:32:36.000000Z",
  "pivot": {
    "user_id": 1,
    "role_id": 1,
    "created_at": "2020-09-08T20:58:22.000000Z",
    "updated_at": "2020-09-08T20:58:22.000000Z"
  }
},
{
  "id": 3,
  "name": "Test",
  "slug": "test",
  "description": "test",
  "full-access": "no",
  "created_at": "2020-09-08T20:38:19.000000Z",
  "updated_at": "2020-09-08T20:38:19.000000Z",
  "pivot": {
    "user_id": 1,
    "role_id": 3,
    "created_at": "2020-09-08T20:58:22.000000Z",
    "updated_at": "2020-09-08T20:58:22.000000Z"
  }
}

```

La imagen de arriba muestra los roles 1 y 3 asignados al usuario con id=1.

Eliminar roles con “detach”.

Esta función es la contraparte de “Attach” es vez de agregar los elimina. El siguiente ejemplo permitirá eliminar todos los roles del usuario con el rol con id=1.

```

47 |     $user = User::find(1);
48 |     $user->roles()->detach([1]);
49 |
50 |     return $user->roles;
51 |
52 });

```

El siguiente ejemplo permitirá eliminar todos los roles del usuario con el rol con id=3.

```

49 |     $user = User::find(1);
50 |     $user->roles()->detach([3]);
51 |
52 |     return $user->roles;

```

Si vemos en el navegador el usuario 1 ya no tiene roles:

```
1 // 20200908160443
2 // http://localhost:8000/test
3
4 [
5
6 ]
```

(*) Sin embargo esto es muy incómodo en estar usando “attach” o “detach” para ello laravel nos provee la opción “sync()”.

Antes de usarla agreguemos se agregan los 3 roles al usuario con “attach”:

```
'$user = User::find(1);
$user->roles()->attach([1,2,3]);
//$user->roles()->detach([3]);

return $user->roles;
});'
```

En el navegador se observará que tiene asignado los tres roles (admin, guest y test).

Ahora se utiliza la función/opción “sync()” la cual eliminara todos otros roles y dejarnos solo con el “rol” 1. Y Si refrescamos la misma ruta en el navegador no se realizaran nuevos registros, solo se conserva el rol con id=1.

```
'$user = User::find(1);
//$user->roles()->attach([1,2,3]);
//$user->roles()->detach([3]);
$user->roles()->sync([1]);
return $user->roles;
});'
```

El siguiente ejemplo le pasamos a “sync” el rol con id=3, lo que hará es crear el rol para el usuario, por lo tanto, si el rol no existe se lo crea.

```
'$user = User::find(1);
//$user->roles()->attach([1,2,3]);
//$user->roles()->detach([3]);
$user->roles()->sync([1,3]);
return $user->roles;
});'
```

Lo siguiente será también comunicar los roles con los permisos, por lo tanto, de esta manera se crean la relación muchos a muchos la cual se utilizará también para los permisos dado a que los roles se tienen que comunicar con los permisos.

❖ Crear relación entre roles y permisos

Primero se debe crear el modelo “Permission” y su tabla respectiva:

```
#> php artisan make:model FreddyPermisos\Models\Permission -m
```

Seguidamente se crea la “tabla” <permission_role>. En este caso al momento de crear esta tabla debemos de tener en cuenta como en antes al crear la tabla “role_user” se debe tener en cuenta que “modelo” tiene la letra inicial mas menor. En este caso el modelo “Permission” es menor que “Role” ya que la “P” es menor que la “R” de role. Por lo tanto se crea la tabla pivot:

```
#> php artisan make:migration create_permission_role_table
```

Crear atributos de la tabla “Permission”:

```
public function up()
{
    Schema::create('permissions', function (Blueprint $table) {
        $table->id();
        $table->string('name')->unique();
        $table->string('slug')->unique();
        $table->text('description')->nullable();
        $table->timestamps();
    });
}
```

Crear atributos de la tabla intermedia “permission_role”:

```
Schema::create('permission_role', function (Blueprint $table) {
    $table->id();
    $table->foreignId('role_id')->references('id')->on('roles')->onDelete('cascade');
    $table->foreignId('permission_id')->references('id')->on('permissions')->onDelete('cascade');
    $table->timestamps();
});
```

Esta tabla tiene las llaves foráneas (role_id y permission_id)

Crear las relaciones en los modelos:

En el modelo Permission se crear la relación belongsToMany(muchos-a-muchos) con roles:

```
class Permission extends Model
{
    public function roles(){
        //relación muchos a muchos.
        return $this->belongsToMany('App\FreddyPermisos\Models\Role')->withTimestamps();
    }
}
```

Además al modelo “Permission” se le debe agregar el **Fillable**;

```

class Permission extends Model
{
    protected $fillable = [
        'name', 'slug', 'description',
    ];

    public function roles(){
        //relación muchos a muchos.
        return $this->belongsToMany('App\FreddyPermisos\Models\Role')->withTimestamps();
    }
}

```

Esto del fillable permite que llenemos de manera masiva la tabla Permission y obliga a que cuando se creen “Permissions” utilicen esos campos (name, slug y descripción).

De esta manera en el archivo **web.php** que contiene las rutas se agrega el namespace de Permission para poder hacer las pruebas de registros de permisos.

```

1
2
3 use Illuminate\Support\Facades\Route;
4
5 use App\FreddyPermisos\Models\Role;
6 use App\User;
7 use App\FreddyPermisos\Models\Permission;
8 /*
9 */

```

Agregar relación muchos a muchos en el modelo “Role” con “Permission”:

```

class Role extends Model
{
    //es. desde aqui
    //en. from here
    protected $fillable = [
        'name', 'slug', 'description','full-access',
    ];
    public function users(){
        //relación muchos a muchos.
        return $this->belongsToMany('App\User')->withTimestamps();
    }

    public function permissions(){
        //relación muchos a muchos.
        return $this->belongsToMany('App\FreddyPermisos\Models\Permission')->withTimestamps();
    }
}

```

Ahora para poder acceder a los permissions que tiene los roles se consulta de la siguiente manera:

```
58     $role = Role::find(1);
59     // $user->roles()->sync([1,3]);
60     return $role->permissions;
61
62
63});
```

```
// 20200908181045
// http://localhost:8000/test

[
]
```

Se aprecia en el navegador que el Role con id=1 no tiene ningún permission asignado. Antes de poder asignar “permissions” a los roles se crean estos permissions de la siguiente manera:

```
return Permission::create([
    'name' => 'Create Product',
    'slug' => 'product.create',
    'description' => 'An user can create a permission',
]);
```

El **slug** indica primero el nombre del controlador en este caso “**product**” seguido de un “punto” más el método del controlador en este caso “**create**”. En la descripción en este caso debio ser “an user can create a producto” sin embargo con fines prácticos no hay problema.

De la misma manera se crea otro permiso:

```
return Permission::create([
    'name' => 'List Product',
    'slug' => 'product.index',
    'description' => 'An user can list a permission',
]);
```

Asignar permissions a un “Role” en específico:

Primero se verifica que en este ejemplo que el “Role” con id=2 no tiene “Permissions” asignados o creados:

```
$role = Role::find(2);
// $user->roles()->sync([1,3]);
return $role->permissions;
```

```

1 // 20200908183231
2 // http://localhost:8000/test
3
4 [
5
6 ]

```

Ahora lo que se hace es con “sync()” asignar un permission al “Role” con id=2 un “permission” con id=3:

```

$role = Role::find(2);
$role->permissions()->sync([3]);
return $role->permissions();

```

El resultado en el navegador es el siguiente:

```

[
{
  "id": 3,
  "name": "Create Product",
  "slug": "product.create",
  "description": "An user can create a permission",
  "created_at": "2020-09-08T23:30:01.000000Z",
  "updated_at": "2020-09-08T23:30:01.000000Z",
  "pivot": {
    "role_id": 2,
    "permission_id": 3,
    "created_at": "2020-09-08T23:37:02.000000Z",
    "updated_at": "2020-09-08T23:37:02.000000Z"
  }
}
]

```

Por el momento solo se tiene registrados dos diferentes permissions:

Opciones		id	name	slug	description	created_at	updated_at
<input type="checkbox"/>	Editar Copiar Borrar	3	Create Product	product.create	An user can create a permission	2020-09-08 23:30:01	2020-09-08 23:30:01
<input type="checkbox"/>	Editar Copiar Borrar	4	List Product	product.index	An user can list a permission	2020-09-08 23:30:18	2020-09-08 23:30:18

El siguiente ejemplo se asigna el “Permission” con id=4 al “Role” con id=2, en este caso este “Role” ya tendrá dos “Permissions” asignados:

```
$role = Role::find(2);
$role->permissions()->sync([3,4]);
return $role->permissions;
```

Resultado en el navegador:

```
{
  "id": 3,
  "name": "Create Product",
  "slug": "product.create",
  "description": "An user can create a permission",
  "created_at": "2020-09-08T23:30:01.000000Z",
  "updated_at": "2020-09-08T23:30:01.000000Z",
  "pivot": {
    "role_id": 2,
    "permission_id": 3,
    "created_at": "2020-09-08T23:37:02.000000Z",
    "updated_at": "2020-09-08T23:37:02.000000Z"
  }
},
{
  "id": 4,
  "name": "List Product",
  "slug": "product.index",
  "description": "An user can list a permission",
  "created_at": "2020-09-08T23:30:18.000000Z",
  "updated_at": "2020-09-08T23:30:18.000000Z",
  "pivot": {
    "role_id": 2,
    "permission_id": 4,
    "created_at": "2020-09-08T23:43:26.000000Z",
    "updated_at": "2020-09-08T23:43:26.000000Z"
  }
}
```

Fuente:

<https://www.youtube.com/watch?v=OFwzpRHN5YQ&list=PLtg6DxcGyHSvB6xvQbacVfL83UoFEvOGz&index=3>

❖ Crear Seeder con tablas relacionadas múltiples tablas - Roles y Permisos

Esta sección tiene como fuente el video:

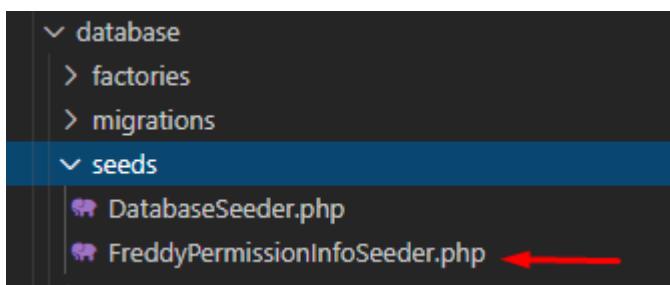
<https://www.youtube.com/watch?v=h1MVjfvmweU>

Los seeders permiten poblar con datos la base de datos en lugar de hacerlo en la propia base de datos.

En este ejemplo se crea el “Seeder” llamado “FreddyPermissionInfoSeeder” es importante que el nombre del “Seeder” a crear termine al final con la palabra “Seeder”.

```
#> php artisan make:seeder FreddyPermissionInfoSeeder
```

El seeder creado se debe de encontrar dentro de la carpeta “database->seeds”



Luego para ejecutar el seeder se hace con “composer” para cargarlo a base de datos

```
#> composer dump-autoload
```

Eso generará un archivo.. según el profe luego creará un video de como hacer eso..

```
C:\laragon\www\RolesPermisos
λ composer dump-autoload
Generating optimized autoload files> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
Generated optimized autoload files containing 4458 classes

C:\laragon\www\RolesPermisos
```

Lo siguiente es abrir el archivo “DatabaseSeeder.php” y agregar lo siguiente:

```
    public function run()
    {
        // $this->call(UsersTableSeeder::class);

        $this->call(FreddyPermissionInfoSeeder::class);
    }
}
```

Donde “FreddyPermissionInfoSeeder” es el nombre de la clase obtenido del archivo creado “FreddyPermissionInfoSeeder.php”. Así en el archivo “DatabaseSeeder.php”

Laravel sabe cual seeder ejecutar primero.

Para ejecutar los seeders en un servidor de prueba se usa el comando:

```
#> php artisan migrate:fresh --seed
```

Si nos encontramos en un servidor para poner el Sistema en producción se debe de ejecutar el siguiente comando:

```
#> php artisan migrate --seed
```

```
C:\laragon\www\RolesPermisos
λ php artisan migrate --seed
Nothing to migrate.
Seeding: FreddyPermissionInfoSeeder
Seeded: FreddyPermissionInfoSeeder (0 seconds)
Database seeding completed successfully.
```

Donde “FreddyPermissionInfoSeeder” es el nombre de la clase obtenido del archivo creado “FreddyPermissionInfoSeeder.php”.

Laravel sabe cual seeder ejecutar primero.

```
C:\laragon\www\RolesPermisos
λ |
```

Para ejecutar los seeders en un servidor de prueba se usa el comando:

Otra forma de ejecutar los seeder a partir de laravel 7 es de la siguiente manera:

```
#> php artisan db:seed
```

```
C:\laragon\www\RolesPermisos
λ php artisan db:seed
Seeding: FreddyPermissionInfoSeeder
Seeded: FreddyPermissionInfoSeeder (0 seconds)
Database seeding completed successfully.
```

Otra forma de ejecutar los seeder a partir de laravel 7 es de la siguiente manera:

```
C:\laragon\www\RolesPermisos
λ |
```

Para poblar el seeder “FreddyPermissionInfoSeeder” primero se comenzara creando un usuario. Para ello copiamos toda la estructura de un usuario disponible dentro de “database->factories->UserFactory.php”

```
$factory->define(User::class, function (Faker $faker) {
    return [
        'name' => $faker->name,
        'email' => $faker->unique()->safeEmail,
        'email_verified_at' => now(),
        'password' => '$2y$10$92IXUNpkj00r0Q5byMi.Ye4oKoEa3Ro9llC/og/at2.uheWG/igi',
        'remember_token' => Str::random(10),
    ];
});
```

Copiamos eso que esta dentro del cuadro rojo y lo ponemos dentro de FreddyPermissionInfoSeeder:

```
    public function run()
    {
        //Crear usuario Admin
        'name' => '',
        'email' => '',
        'password' => ''
    }
}
```

No olvidemos de también poner los namespace a utilizar de los modelos “User, Role y Permission”:

```
esPermisos > database > seeds >  FreddyPermissionInfoSeeder.php
1 <?php
2
3 use Illuminate\Database\Seeder;
4
5 use App\User;
6 use App\FreddyPermisos\Models\Role;
7 use App\FreddyPermisos\Models\Permission;
```

Entonces para crear el usuario admin quedaría de la siguiente manera:

```
//Crear usuario Admin
$useradmin = User::create([
    'name' => 'admin',
    'email' => 'admin@admin.com',
    'password' => Hash::make('admin')
]);
```

Si nos damos cuenta para crear la contraseña se esta haciendo uso de “Hash” para ello se puso el siguiente namespace:

```
RolesPermisos > database > seeds >  FreddyPermissionInfoSeeder.php
1 <?php
2
3 use Illuminate\Database\Seeder;
4
5 use App\User;
6 use App\FreddyPermisos\Models\Role;
7 use App\FreddyPermisos\Models\Permission;
8 use Illuminate\Support\Facades\Hash;
9
```

De esa manera laravel creara una contraseña y la encriptara en este caso la palabra “admin”.

Ahora para probar que el seeder funciona ejecutamos:

```
#> php artisan db:seed
```

```
C:\laragon\www\RolesPermisos
λ php artisan db:seed
Seeding: FreddyPermissionInfoSeeder
Seeded:  FreddyPermissionInfoSeeder (0.28 seconds)
Database seeding completed successfully.

C:\laragon\www\RolesPermisos
```

Verificamos en la base de datos que el usuario se creo:

	+ Opciones	id	name	email	email_verified_at	password
<input type="checkbox"/>	Editar Copiar Borrar	1	Freddy Alcarazo	freddyalcarazo@hotmail.com	NULL	\$2y\$10\$U0Npus.Dzf94Y6GfTOqT3O7BV
<input type="checkbox"/>	Editar Copiar Borrar	2	admin	admin@admin.com	NULL	\$2y\$10\$lvqq6eBjaajjd90gKtdKpuJkoD4!

De la misma manera si hacemos login debería de funcionar.

Si volvemos a ejecutar el mismo seeder como esta hasta ahora se producirá un error debido a que el “email” es único de acuerdo a la estructura de la tabla “user” para ello entonces lo que debemos de hacer es hacer una verificación:

```
//Crear usuario Admin
$useradmin = User::where('email','admin@admin.com')->first();
if($useradmin){
    $useradmin->delete();
}

$useradmin = User::create([
    'name' => 'admin',
    'email' => 'admin@admin.com',
    'password' => Hash::make('admin')
]);
```

En este caso lo que hace es: Si encuentra un registro con el “email” = ‘admin@admin.com’ lo borrara y creara el usuario seguidamente.

Luego debajo de haber creado el usuario admin, se crea el rol “Admin”:

```
//Crear Role admin:
$roladmin = Role::create([
    'name' => 'Admin',
    'slug' => 'admin',
    'description' => 'Administrator',
    'full-access' => 'yes',
]);
```

En este caso el rol admin tiene el full-access en caso de no tenerlo se verificará que permisos tiene y de acuerdo a ello el usuario con dicho rol podrá navegar.

Asignar el rol de “admin” al usuario ‘admin’:

```

//Crear Role admin:
$roladmin = Role::create([
    'name' => 'Admin',
    'slug' => 'admin',
    'description' => 'Administrator',
    'full-access' => 'yes',
]);
//Asignar el rol 'admin' al usuario 'admin'.
$useradmin->roles()->sync( $roladmin->id );

```

Sin embargo debido a que anteriormente el rol “admin” ya se había creado en el archivo web.php cuando se accedía a la ruta “/test” si ejecutamos el seeder otra vez se presentará un error de llaves únicas debido a que el campo “name” de la tabla “roles” es único:

```

SQLSTATE[23000]: Integrity constraint violation: 1062 Duplicate entry 'Admin' for key 'roles_name_unique' (SQL: insert into `roles` ('name', 'full-access', 'updated_at', 'created_at') values ('Admin', 'admin', 'Administrator', yes, 2020-09-09 16:52:19, 2020-09-09 16:52:19))

at C:\laragon\www\RolesPermisos\vendor\laravel\framework\src\Illuminate\Database\Connection.php:671
 667|     // If an exception occurs when attempting to run a query, we'll format the error
 668|     // message to include the bindings with SQL, which will make this exception a
 669|     // lot more helpful to the developer instead of just the database's errors.
 670|     catch (Exception $e) {
> 671|         throw new QueryException(
 672|             $query, $this->prepareBindings($bindings), $e
 673|         );
 674|     }
 675| }

1 C:\laragon\www\RolesPermisos\vendor\laravel\framework\src\Illuminate\Database\Connection.php:464
PDOException::__SQLSTATE[23000]: Integrity constraint violation: 1062 Duplicate entry 'Admin' for key 'roles_name_unique'()

2 C:\laragon\www\RolesPermisos\vendor\laravel\framework\src\Illuminate\Database\Connection.php:464
PDOStatement::execute()

```

entonces para evitar estar haciendo validaciones todo el tiempo así como se hizo antes para crear el usuario “admin” lo que debemos de hacer es un “truncate” esto significa vaciar/eliminar todo los records de la tablas con llaves foraneas y de esta manera resetear los ids desde el comienzo.

Para hacer el truncate primero se debe de importar el “namespace” DB:

```

use Illuminate\Support\Facades\Hash;
//Truncate
use Illuminate\Support\Facades\DB; ←

```

Luego se realiza los truncates respectivos:

```

20     public function run()
21     {
22         //Hacer truncate a tablas sin modelo.
23         DB::table('role_user')->truncate();
24         DB::table('permission_role')->truncate();
25         //truncate a tablas con modelo:
26         Permission::truncate();
27         Role::truncate();
28
29
30         //Crear usuario Admin
31         $useradmin = User::where('email','admin@admin.com')->first();
32         if($useradmin){
33             $useradmin->delete();
34

```

Sin embargo eso si ejecutamos el “Seeder” va dar error por que la tabla “permission_role” tiene llaves foráneas y otras también como “role_user”.

```

λ php artisan db:seed
Seeding: FreddyPermissionInfoSeeder
 Illuminate\Database\QueryException
SQLSTATE[42000]: Syntax error or access violation: 1701 Cannot truncate a table referenced in a foreign key constraint ('rolespermisos`.`permission_role` FOREIGN KEY (`permission_id`) REFERENCES `rolespermisos`.`permissions` (`id`)) (SQL: truncate table `permissions`)

at C:\laragon\www\RolesPermisos\vendor\laravel\framework\src\Illuminate\Database\Connection.php:671
667|     // If an exception occurs when attempting to run a query, we'll format the error
668|     // message to include the bindings with SQL, which will make this exception a
669|     // lot more helpful to the developer instead of just the database's errors.
670|     catch (Exception $e) {
671|         throw new QueryException(
672|             $query, $this->prepareBindings($bindings), $e
673|         );
674|     }
675|
1  C:\laragon\www\RolesPermisos\vendor\laravel\framework\src\Illuminate\Database\Connection.php:464
 PDOException::("SQLSTATE[42000]: Syntax error or access violation: 1701 Cannot truncate a table referenced in a foreign key constraint ('rolespermisos`.`permission_role` FOREIGN KEY (`permission_id`) REFERENCES `rolespermisos`.`permissions` (`id`))")
2  C:\laragon\www\RolesPermisos\vendor\laravel\framework\src\Illuminate\Database\Connection.php:464
 PDOStatement::execute()

C:\laragon\www\RolesPermisos

```

Para solucionar eso debemos de “desactivar todos los foreign keys” ponerlos en cero. Luego terminar de hacer los truncates se volverán a encender.

```

20     public function run()
21     {
22         //Desactivar todos los foreign keys:
23         DB::statement("SET foreign_key_checks=0");
24         //Hacer truncate a tablas sin modelo.
25         DB::table('role_user')->truncate();
26         DB::table('permission_role')->truncate();
27         //truncate a tablas con modelo:
28         Permission::truncate();
29         Role::truncate();
30         //habilitar los foreign keys:
31         DB::statement("SET foreign key checks=1");
32

```

Ahora si ejecutamos el seeder no debería de haber problemas:

```

C:\laragon\www\RolesPermisos
λ php artisan db:seed
Seeding: FreddyPermissionInfoSeeder
Seeded: FreddyPermissionInfoSeeder (0.95 seconds)
Database seeding completed successfully.

```

Si verificamos en la base de datos:

	id	name	slug	description	full-access	created_at	updated_at
<input type="checkbox"/>	1	Admin	admin	Administrator	yes	2020-09-09 17:15:19	2020-09-09 17:15:19

Como podemos ver en la imagen de arriba el id del rol creado comienza en 1. Por lo tanto, si creamos otro rol en la base de datos y volvemos a ejecutar el seeder este eliminará todos los registros de las tablas que se afectan y volverá a contar desde 1 en adelante.

No se hace el truncate a la tabla “users” para no afectar la integridad, por que el sistema puede tener muchos usuarios guardados y podemos afectar la integridad.

Lo siguiente es crear permisos y asignarlos al rol “admin”.

Lo primero es crear un array para almacenar los id de los permisos que se vayan creando:

```

53
54 //Crear array de permisos
55 $permission_all = [];
56

```

Lo siguiente es crear el primer permiso:

```

53
54 //Crear array de permisos
55 $permission_all = [];
56 //permission_role
57 $permission = Permission::create([
58     'name' => 'List role',
59     'slug' => 'role.index',
60     'description' => 'An user can list a role',
61 ]);
62
63 $permission_all[] = $permission->id;
64

```

En la línea 63 vemos que el permiso creado se agrega al array \$permission_all.

Por lo tanto, se crearon 5 permisos:

```

//permission_role
$permission = Permission::create([
    'name' => 'List role',
    'slug' => 'role.index',
    'description' => 'An user can list a role',
]);
$permission_all[] = $permission->id;

$permission = Permission::create([
    'name' => 'Show role',
    'slug' => 'role.show',
    'description' => 'An user can see a role',
]);
$permission_all[] = $permission->id;

$permission = Permission::create([
    'name' => 'Create role',
    'slug' => 'role.create',
    'description' => 'An user can create a role',
]);
$permission_all[] = $permission->id;

$permission = Permission::create([
    'name' => 'Edit role',
    'slug' => 'role.edit',
    'description' => 'An user can Edit a role',
]);
$permission_all[] = $permission->id;

$permission = Permission::create([
    'name' => 'Destroy role',
    'slug' => 'role.destroy',
    'description' => 'An user can destroy a role',
]);
$permission_all[] = $permission->id;

```

Lo siguiente es hacer un sync() para poblar la tabla permission_role con los ids respectivos de cada rol creado.

```

//tabla permission_role
$roladmin->permissions()->sync( $permission_all );
}

C:\laragon\www\RolesPermisos
$ php artisan db:seed
Seeding: FreddyPermissionInfoSeeder
Seeded:  FreddyPermissionInfoSeeder (1.59 seconds)
Database seeding completed successfully.

```

Si observamos la base de datos: **Tabla permissions:**

+ Opciones		id	name	slug	description	created_at	updated_at
<input type="checkbox"/>	Editar Copiar Borrar	1	List role	role.index	An user can list a role	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	2	Show role	role.show	An user can see a role	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	3	Create role	role.create	An user can create a role	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	4	Edit role	role.edit	An user can Edit a role	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	5	Destroy role	role.destroy	An user can destroy a role	2020-09-09 17:59:10	2020-09-09 17:59:10

Tabla permission_role:

+ Opciones		id	role_id	permission_id	created_at	updated_at
<input type="checkbox"/>	Editar Copiar Borrar	1	1	1	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	2	1	2	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	3	1	3	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	4	1	4	2020-09-09 17:59:10	2020-09-09 17:59:10
<input type="checkbox"/>	Editar Copiar Borrar	5	1	5	2020-09-09 17:59:10	2020-09-09 17:59:10

En efecto todos permisos se crearon y fueron asignados al “Role” con id=1 en este caso un role llamado “Admin”.

Lo siguiente es crear el permiso para Usuarios:

```

$permission = Permission::create([
    'name' => 'List user',
    'slug' => 'user.index',
    'description' => 'An user can list a user',
]);
$permission_all[] = $permission->id;

$permission = Permission::create([
    'name' => 'Show user',
    'slug' => 'user.show',
    'description' => 'An user can see a user',
]);
$permission_all[] = $permission->id;

$permission = Permission::create([
    'name' => 'Edit user',
    'slug' => 'user.edit',
    'description' => 'An user can Edit a user',
]);
$permission_all[] = $permission->id;

$permission = Permission::create([
    'name' => 'Destroy user',
    'slug' => 'user.destroy',
    'description' => 'An user can destroy a user',
]);
$permission_all[] = $permission->id;
/*
Laravel crea los usuarios, si la empresa ha eliminado la opción
register entonces se habilita este permiso.
$permission = Permission::create([
    'name' => 'Create user',
    'slug' => 'user.create',
    'description' => 'An user can create a user',
]);
$permission_all[] = $permission->id;

```

No se agregar el permiso Crear usuario solo esta comentado debido a que las empresas usan por defecto la opción de register de laravel pero si por alguna razón se ha eliminado esa opción entonces se debe de descomentar el permiso en el seeder para crear usuarios. Se debe de preguntar si la empresa sigue el proceso tradicional de laravel o la empresa ha creado una ventana para poder registrar usuarios, dependiendo de lo que digan se podrá habilitar el permiso.

Ejecutamos el sync al final luego de haber definido todos los permisos para la tabla role, esta línea de código antes estaba arriba, se ha puesto debajo de todos los permisos y el sync ejecuta el arreglo \$permission_all que tiene todos los “id” de los permisos creados.

```

138
139     //tabla permission_role
140     $roladmin->permissions()->sync( $permission_all );
141

```

Finalmente podemos apreciar los permisos creados en la base de datos:

Tabla permissions

+ Opciones		id	name	slug	description	created_at	updated_at
<input type="checkbox"/>	Editar Copiar Borrar	1	List role	role.index	An user can list a role	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	2	Show role	role.show	An user can see a role	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	3	Create role	role.create	An user can create a role	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	4	Edit role	role.edit	An user can Edit a role	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	5	Destroy role	role.destroy	An user can destroy a role	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	6	List user	user.index	An user can list a user	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	7	Show user	user.show	An user can see a user	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	8	Edit user	user.edit	An user can Edit a user	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	9	Destroy user	user.destroy	An user can destroy a user	2020-09-09 18:21:56	2020-09-09 18:21:56

Tabla permission_role

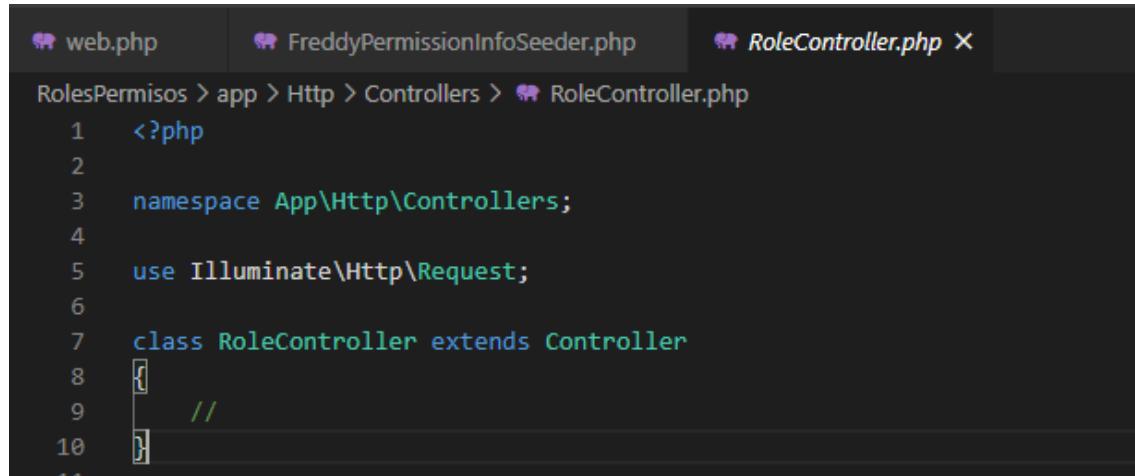
+ Opciones		id	role_id	permission_id	created_at	updated_at
<input type="checkbox"/>	Editar Copiar Borrar	1	1	1	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	2	1	2	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	3	1	3	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	4	1	4	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	5	1	5	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	6	1	6	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	7	1	7	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	8	1	8	2020-09-09 18:21:56	2020-09-09 18:21:56
<input type="checkbox"/>	Editar Copiar Borrar	9	1	9	2020-09-09 18:21:56	2020-09-09 18:21:56

❖ Crear controlador para los roles y la vista index - Roles y Permisos Laravel 7

Lo primero que se hace en esta sección es crear el controlador Role. La convención es que al crear un controlador al final del nombre debemos de agregar la palabra Controller.

Entonces con base a esto creamos el primer controlador Role:

```
#> php artisan make:controller RoleController
```

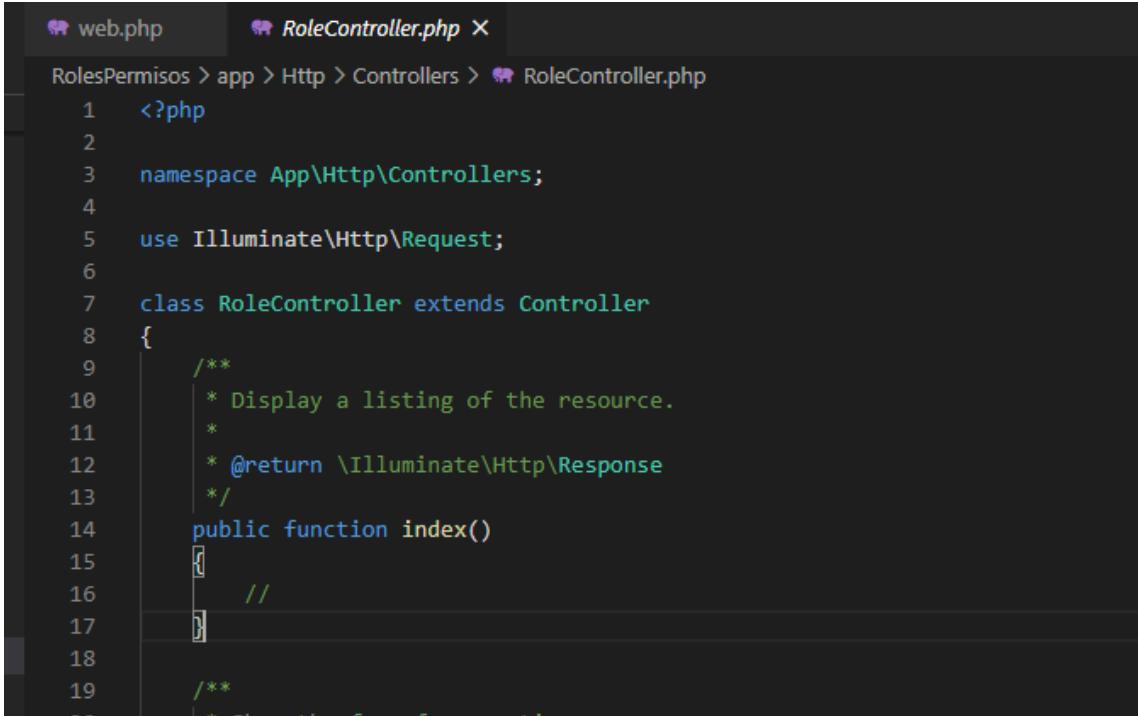


```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class RoleController extends Controller
8 {
9     //
10 }
```

Los controladores creados se ubican dentro de App->http->controllers

Sin embargo el controlador creado esta prácticamente basio, para ello eliminamos el controlador creado y ahora creamos un controlador tipo “resource” el cual ya traerá creado métodos como “create, index, destroy, edit”..

```
#> php artisan make:controller RoleController --resource
```



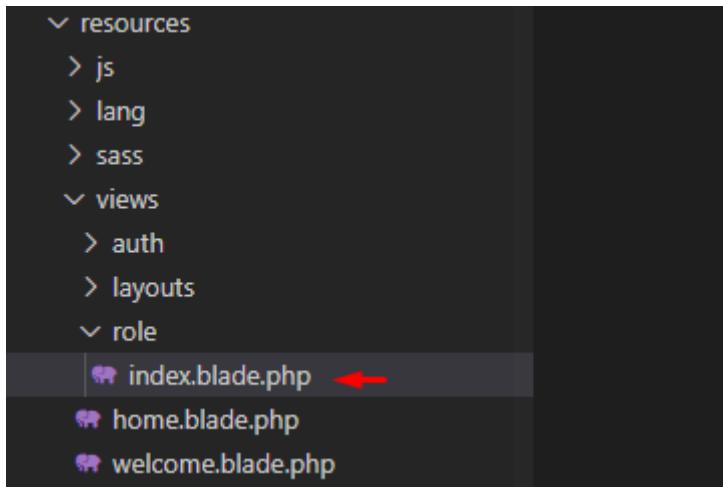
The screenshot shows a code editor with two tabs: "web.php" and "RoleController.php". The "RoleController.php" tab is active, displaying the following PHP code:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class RoleController extends Controller
8 {
9     /**
10      * Display a listing of the resource.
11      *
12      * @return \Illuminate\Http\Response
13      */
14     public function index()
15     {
16         //
17     }
18
19     /**
20      * Show the form for creating a new resource.
21      *
22      * @return \Illuminate\Http\Response
23      */
24     public function create()
25     {
26         //
27     }
28
29     /**
30      * Store a newly created resource in storage.
31      *
32      * @param \Illuminate\Http\Request $request
33      * @return \Illuminate\Http\Response
34      */
35     public function store(Request $request)
36     {
37         //
38     }
39
40     /**
41      * Display the specified resource.
42      *
43      * @param int $id
44      * @return \Illuminate\Http\Response
45      */
46     public function show($id)
47     {
48         //
49     }
50
51     /**
52      * Show the form for editing the specified resource.
53      *
54      * @param int $id
55      * @return \Illuminate\Http\Response
56      */
57     public function edit($id)
58     {
59         //
60     }
61
62     /**
63      * Update the specified resource in storage.
64      *
65      * @param \Illuminate\Http\Request $request
66      * @param int $id
67      * @return \Illuminate\Http\Response
68      */
69     public function update(Request $request, $id)
70     {
71         //
72     }
73
74     /**
75      * Remove the specified resource from storage.
76      *
77      * @param int $id
78      * @return \Illuminate\Http\Response
79      */
80     public function destroy($id)
81     {
82         //
83     }
84 }
```

Crear la vista “index.blade.php”

Lo que faremos ahora es ir a la carpeta “resources->views” y crear la carpeta “role”

Una vez creada la carpeta “role” dentro de esta misma creamos la vista “index.blade.php”



El contenido de la vista “role->index.blade.php” será el contenido de la vista [home.blade.php](#) por lo tanto copiamos todo el código de la vista [home.blade.php](#) y lo pegamos dentro index.blade.php:

The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right.

EXPLORER

- > OPEN EDITORS
- > UNTITLED (WORKSPACE)
 - > app
 - > bootstrap
 - > config
 - > database
 - > node_modules
 - > public
 - > resources
 - > js
 - > lang
 - > sass
 - > views
 - > auth
 - > layouts
 - > role
 - index.blade.php** (selected)
 - home.blade.php
 - welcome.blade.php
 - > routes
 - > api.php

EDITOR

RolesPermisos > resources > views > role > **index.blade.php**

```

1  @extends('layouts.app')
2
3  @section('content')
4  <div class="container">
5    <div class="row justify-content-center">
6      <div class="col-md-8">
7        <div class="card">
8          <div class="card-header">{{ __('Dashboard') }}</div>
9
10         <div class="card-body">
11           @if (session('status'))
12             <div class="alert alert-success" role="alert">
13               {{ session('status') }}
14             </div>
15           @endif
16
17           {{ __('You are logged in!') }}
18         </div>
19       </div>
20     </div>
21   </div>
22 </div>
23 @endsection

```

Antes de crear la ruta para el controlador “RoleController” lo quearemos es guardar todo el código utilizado antes en web.php especialmente el contenido de la ruta “/test” en donde están los ejemplos de “attach, detach y sync” copiamos todo eso en un archivo llamado “comandosusados.php” que lo creamos dentro de la carpeta “routes” no olvidemos poner al inicio del archivo el comando “<?php” para que el sistema reconosca eso como un archivo “.php”.

The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right.

TERMINAL

comandosusados.php - Untitled

EDITOR

RolesPermisos > routes > **comandosusados.php**

```

1  <?php
2

```

EDITOR

RolesPermisos > routes > **comandosusados.php**

```

1  <?php
2  Route::get('/test', function (){
3    /*
4      return Role::create([
5        'name' => 'Admin',

```

El archivo web.php quedaría de la siguiente manera:

Terminal Help web.php - Untitled (Workspace) - Visual Studio Code

web.php index.blade.php home.blade.php

```
RolesPermisos > routes > web.php
12 |
13 | Here is where you can register web routes for your application. These
14 | routes are loaded by the RouteServiceProvider within a group which
15 | contains the "web" middleware group. Now create something great!
16 |
17 */
18
19 Route::get('/', function () {
20     return view('welcome');
21 });
22
23 Auth::routes();
24
25 Route::get('/home', 'HomeController@index')->name('home');
26 |
```

Bien ahora lo que hacemos es definir nuestra ruta para el controlador de tipo “Resource” “Role” si vamos a crear una ruta que va ejecutar un método en ese caso usamos “->name” en lugar de “->names”.

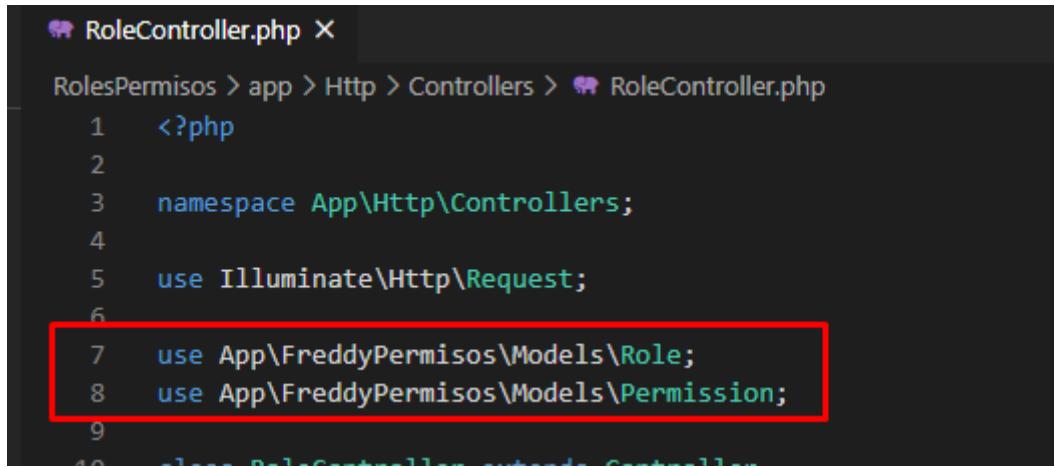
```
22
23     Auth::routes();
24
25     Route::get('/home', 'HomeController@index')->name('home');
26
27     Route::resource('/role', 'RoleController')->names('role');
28 |
```

Para consultar desde la consola todas las rutas disponibles ejecutamos el comando:

#> php artisan route:list

C:\laragon\www\RolesPermisos	php artisan route:list				
Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	api/user		Closure	api
	GET HEAD	home	home	App\Http\Controllers\HomeController@index	auth:api
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web
	POST	login		App\Http\Controllers\Auth\LoginController@login	auth
	POST	logout	logout	App\Http\Controllers\Auth\LoginController@logout	web
	POST	password/confirm	logout	App\Http\Controllers\Auth\ConfirmPasswordController@confirm	guest
	GET HEAD	password/confirm	password.confirm	App\Http\Controllers\Auth\ConfirmPasswordController@showConfirmForm	web
	POST	password/email	password.email	App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail	auth
	POST	password/reset	password.update	App\Http\Controllers\Auth\ResetPasswordController@reset	web
	GET HEAD	password/reset/{token}	password.reset	App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm	web
	POST	register	register	App\Http\Controllers\Auth\ResetPasswordController@showResetForm	web
	GET HEAD	register	register	App\Http\Controllers\Auth\RegisterController@register	guest
	GET HEAD	role	role.index	App\Http\Controllers\RoleController@index	web
	POST	role	role.store	App\Http\Controllers\RoleController@store	web
	GET HEAD	role/create	role.create	App\Http\Controllers\RoleController@create	web
	GET HEAD	role/{role}	role.show	App\Http\Controllers\RoleController@show	web
	PUT PATCH	role/{role}	role.update	App\Http\Controllers\RoleController@update	web
	DELETE	role/{role}	role.destroy	App\Http\Controllers\RoleController@destroy	web
	GET HEAD	role/{role}/edit	role.edit	App\Http\Controllers\RoleController@edit	web

Luego en el archivo del controlador “RoleController.php” agregamos los namespace necesarios:



```
RoleController.php X
RolesPermisos > app > Http > Controllers > RoleController.php
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 use App\FreddyPermisos\Models\Role;
8 use App\FreddyPermisos\Models\Permission;
9
10 class RoleController extends Controller
```

Ahora dentro del método “index” agregamos lo siguiente:

```
public function index()
{
    $roles = Role::orderBy('id', 'Desc')->paginate(2);

    return view('role.index', compact('roles'));
}
```

Como vemos se obtienen los roles en forma “Descendente” y están paginados en 2 por página. Luego se retorna la vista, ahí primero esta el nombre de la carpeta de la vista en este caso se llama “role” seguido de un punto más el nombre del archivo “index” y se pasa la variable a través de compact llamada “roles”.

Ahora dentro de la vista Role->index.blade.php se va a crear una tabla:

Esta tabla basado en estilos de “bootstrap” obtenida de:

<https://getbootstrap.com/docs/4.5/content/tables/>

```
<table class="table table-bordered">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td>
    </tr>
    <tr>
```

```

<th scope="row">2</th>
<td>Jacob</td>
<td>Thornton</td>
<td>@fat</td>
</tr>
<tr>
    <th scope="row">3</th>
    <td colspan="2">Larry the Bird</td>
    <td>@twitter</td>
</tr>
</tbody>
</table>

```

Lo quearemos será adecuar esta tabla a nuestro estilo:

Primeramente:

```

<table class="table table-bordered">
    <thead>
        <tr>
            <th scope="col">#</th>
            <th scope="col">name</th>
            <th scope="col">slug</th>
            <th scope="col">description</th>
            <th scope="col">full-access</th>
            <th colspan="3"></th>
        </tr>
    </thead>
    <tbody>
        <tr>
            @foreach($roles as $role)
            <th scope="row">{{ $role->id }}</th>
            <td>{{ $role->name }}</td>
            <td>{{ $role->slug }}</td>
            <td>{{ $role->description }}</td>
            <td>{{ $role }}</td>
            <td>boton</td>
            @endforeach
        </tr>
    </tbody>
</table>

```

Probando en el navegador vemos que el full-access es un arreglo:

List of Roles					
#	name	slug	description	full-access	
1	Admin	admin	Administrator	{"id":1,"name":"Admin","slug":"admin","description":"Administrator","full-access":"yes","created_at":"2020-09-09T21:10:56.000000Z","updated_at":"2020-09-09T21:10:56.000000Z"}	boton

¿Cómo podemos resolver para mostrar la variable con guion “full-access”?

Para solucionar ese problema lo que hacemos es usar corchetes:

```
<tbody>
    <tr>
        @foreach($roles as $role)
        <th scope="row">{{ $role->id }}</th>
        <td>{{ $role->name }}</td>
        <td>{{ $role->slug }}</td>
        <td>{{ $role->description }}</td>
        <td>{{ $role['full-access'] }}</td>
        <td>boton</td>
    @endforeach
</tr>
```

Lo siguiente es agregar los botones con los enlaces a las rutas respectivas:

```
<tbody>
    <tr>
        @foreach($roles as $role)
        <th scope="row">{{ $role->id }}</th>
        <td>{{ $role->name }}</td>
        <td>{{ $role->slug }}</td>
        <td>{{ $role->description }}</td>
        <td>{{ $role['full-access'] }}</td>
        <td>
            <a class="btn btn-
info" href="{{ route('role.show', $role->id)}}> Show </a>
            <a class="btn btn-
success" href="{{ route('role.edit', $role->id)}}> Edit </a>
            <a class="btn btn-
danger" href="{{ route('role.destroy', $role->id)}}> Delete </a>
        </td>
    @endforeach
</tr>

</tbody>
```

```
<tbody>
  <tr>
    @foreach($roles as $role)
      <th scope="row">{{ $role->id }}</th>
      <td>{{ $role->name }}</td>
      <td>{{ $role->slug }}</td>
      <td>{{ $role->description }}</td>
      <td>{{ $role['full-access'] }}</td>
      <td>
        <a class="btn btn-info" href="{{ route('role.show', $role->id)}}> Show </a>
        <a class="btn btn-success" href="{{ route('role.edit', $role->id)}}> Edit </a>
        <a class="btn btn-danger" href="{{ route('role.destroy', $role->id)}}> Delete </a>
      </td>
    @endforeach
  </tr>
</tbody>
```

Al final debajo de la tabla ponemos la paginación:

```
</tbody>
</table>

{{ $roles->links() }}
```

❖ Método create y vista create - Roles y Permisos Laravel 7

Lo primero que vamos a hacer es crear un botón que nos permita acceder a la vista "create" para ello en el archivo views->role->index.blade.php agregamos lo siguiente:

```
8         <div class="card-header"> <h3>List of Roles </h3> </div>
9
10        <div class="card-body">
11
12            <a href="{{ route('role.create') }}">
13                class="btn btn-primary float-right">
14                    Create
15                </a>
16                <br><br>
17
18            @if (session('status'))
19                <div class="alert alert-success" role="alert">
20                    {{ session('status') }}
21
22
23
24
25
26
27
28
29
30
31
```

Ahora nos vamos a dirigir al método "create" que esta dentro del controlador "RoleController.php"

```
Http
  Controllers
    > Auth
    & Controller.php
    & HomeController.php
    & RoleController.php
    > Middleware
    & Kernel.php
  23
  24
  25
  26
  27
  28
  29
  30
  31
```

**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
 public function create()

Lo siguiente es agregar lo siguiente:

```
public function create()
{
    $permissions = Permission::get(); //::all
    return view('role.create', compact('permissions'));
}
```

Básicamente aquí lo que se esta haciendo es retornar todos los permissions habidos y por haber.

Si vamos al navegador es nos arrojará un error debido a que no existe la vista "create" dentro de la carpeta de vistas llamada "role", entonces creamos la vista.

```
views
  > auth
  > layouts
  > role
    & create.blade.php
    & index.blade.php
```

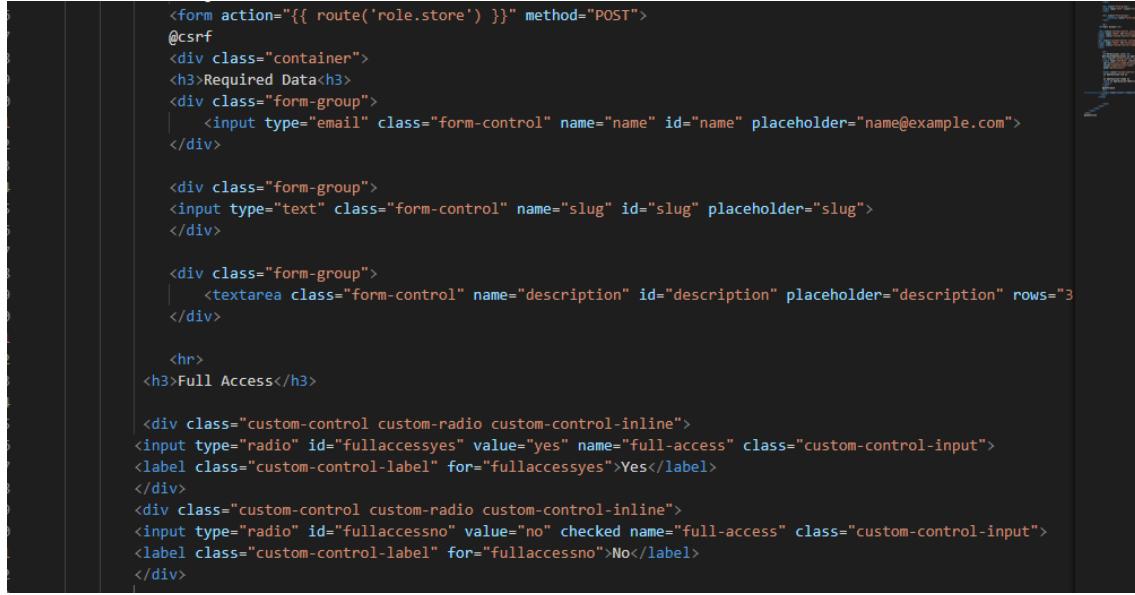
El contenido inicial de la vista “create.blade.php” será el contenido de la vista “home.blade.php”



```
RoleController.php          create.blade.php X      index.blade.php      home.blade.php

RolesPermisos > resources > views > role > create.blade.php
1  @extends('layouts.app')
2
3  @section('content')
4  <div class="container">
5      <div class="row justify-content-center">
6          <div class="col-md-8">
7              <div class="card">
8                  <div class="card-header">{{ __('Dashboard') }}</div>
9
10                 <div class="card-body">
11                     @if (session('status'))
12                         <div class="alert alert-success" role="alert">
13                             {{ session('status') }}
14                         </div>
15                     @endif
16
17                     {{ __('You are logged in!') }}
18                 </div>
19             </div>
20         </div>
21     </div>
22 </div>
23 @endsection
24
```

Luego creamos el formulario respectivamente:



```
<form action="{{ route('role.store') }}" method="POST">
@csrf
<div class="container">
<h3>Required Data</h3>
<div class="form-group">
<input type="email" class="form-control" name="name" id="name" placeholder="name@example.com">
</div>

<div class="form-group">
<input type="text" class="form-control" name="slug" id="slug" placeholder="slug">
</div>

<div class="form-group">
<textarea class="form-control" name="description" id="description" placeholder="description" rows="3"></textarea>
</div>

<hr>
<h3>Full Access</h3>

<div class="custom-control custom-radio custom-control-inline">
<input type="radio" id="fullaccessyes" value="yes" name="full-access" class="custom-control-input">
<label class="custom-control-label" for="fullaccessyes">Yes</label>
</div>
<div class="custom-control custom-radio custom-control-inline">
<input type="radio" id="fullaccessno" value="no" checked name="full-access" class="custom-control-input">
<label class="custom-control-label" for="fullaccessno">No</label>
</div>
```

```

<hr>
<h3>Permissions List</h3>
@foreach($permissions as $permission)
<div class="custom-control custom-checkbox">
<input type="checkbox" class="custom-control-input"
id="permission_{{$permission->id}}"
value="{{$permission->id}}"
name="permission[]"
>
<label class="custom-control-label" for="permission_{{$permission->id}}">
{{ $permission->id }}
-
{{ $permission->name }}
<em>({{ $permission->description }})</em>
</label>
</div>
@endforeach
<hr>
<input type="submit" class="btn btn-primary" value="Save">
</div>
</form>

```

Código del formulario:

```

<form action="{{ route('role.store') }}" method="POST">
    @csrf
    <div class="container">
        <h3>Required Data</h3>
        <div class="form-group">
            <input type="email" class="form-control" name="name" id="name" placeholder="name@example.com">
        </div>

        <div class="form-group">
            <input type="text" class="form-control" name="slug" id="slug" placeholder="slug">
        </div>

        <div class="form-group">
            <textarea class="form-control" name="description" id="description" placeholder="description" rows="3"></textarea>
        </div>

        <hr>
        <h3>Full Access</h3>

        <div class="custom-control custom-radio custom-control-inline">
            <input type="radio" id="fullaccessyes" value="yes" name="full-access" class="custom-control-input">
            <label class="custom-control-label" for="fullaccessyes">Yes</label>
        </div>
    </div>

```

```
<div class="custom-control custom-radio custom-control-
inline">
    <input type="radio" id="fullaccessno" value="no" checked name
="full-access" class="custom-control-input">
    <label class="custom-control-
label" for="fullaccessno">No</label>
</div>

<hr>
<h3>Permissions List</h3>
@foreach($permissions as $permission)
<div class="custom-control custom-checkbox">
<input type="checkbox" class="custom-control-input"
id="permission_{{$permission->id}}"
value="{{$permission->id}}"
name="permission[]"
>
<label class="custom-control-
label" for="permission_{{$permission->id}}">
{{ $permission->id }}
-
{{ $permission->name }}
<em>({{ $permission->description }})</em>
</label>
</div>
@endforeach
<hr>
<input type="submit" class="btn btn-
primary" value="Save">
</div>
</form>
```

Resultado:

Create Role

Required Data

name@example.com

slug

description

Full Access

Yes No

Permissions List

- 1 - List role (An user can list a role)
- 2 - Show role (An user can see a role)
- 3 - Create role (An user can create a role)
- 4 - Edit role (An user can Edit a role)
- 5 - Destroy role (An user can destroy a role)
- 6 - List user (An user can list a user)
- 7 - Show user (An user can see a user)
- 8 - Edit user (An user can Edit a user)
- 9 - Destroy user (An user can destroy a user)

Save

No debemos olvidar que en todo formulario que va a guardar datos con laravel debemos de poner la directiva @csrf

Como se aprecia en formulario:

```
16 | 
17 |     <form action="{{ route('role.store') }}" method="POST">
|     @csrf
```

Lo que faremos ahora es ir al controlador y hacer lo siguiente en el método “store()”:

```
public function store(Request $request)
{
    return $request->all();}
```

Bien esto lo que va hacer es retornar todos los campos enviados en el formulario.

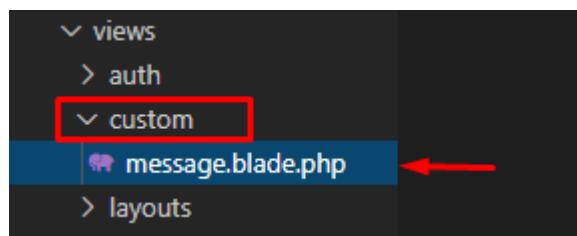
El resultado en el navegador es:

```
4  {
5      "_token": "H9f0SJsrAInSlv1mS0LMN8UCwQUGvxnS01ouqqUO",
6      "name": "test@gmail.com",
7      "slug": "testsd",
8      "description": "descr",
9      "full-access": "no",
10     "permission": [
11         "1",
12         "6"
13     ]
14 }
```

Eso es el resultado luego de haber enviado el formulario con esos datos.

❖ Método store y guardar datos de tablas relacionadas - Roles y Permisos Laravel 7

Antes realizar el guardado de los datos vamos a crear una carpeta llamada “custom” dentro de la carpeta “views” y además creamos dentro de esa carpeta un archivo llamado “message.blade.php”:



El contenido del archivo será una parte del código del archivo “role->index.blade.php”

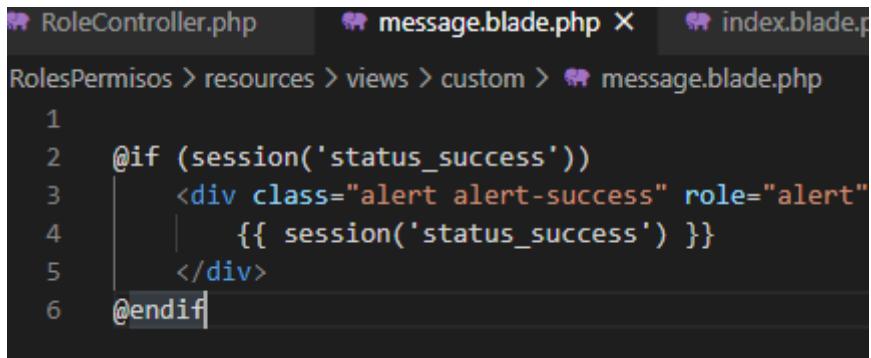
Cortamos esto:

```
RoleController.php message.blade.php index.blade.php create.blade.php
RolesPermisos > resources > views > role > index.blade.php
1 @extends('layouts.app')
2
3 @section('content')
4 <div class="container">
5   <div class="row justify-content-center">
6     <div class="col-md-8">
7       <div class="card">
8         <div class="card-header"> <h3>List of Roles </h3> </div>
9
10        <div class="card-body">
11
12          <a href="{{ route('role.create') }}">
13            class="btn btn-primary float-right">
14              Create
15            </a>
16            <br><br>
17
18          @if (session('status'))
19            <div class="alert alert-success" role="alert">
20              {{ session('status') }}
21            </div>
22          @endif
23        <table class="table table-hover">
24          <thead>
```

Cortamos ese código y se lo agregamos al “message.blade.php”:

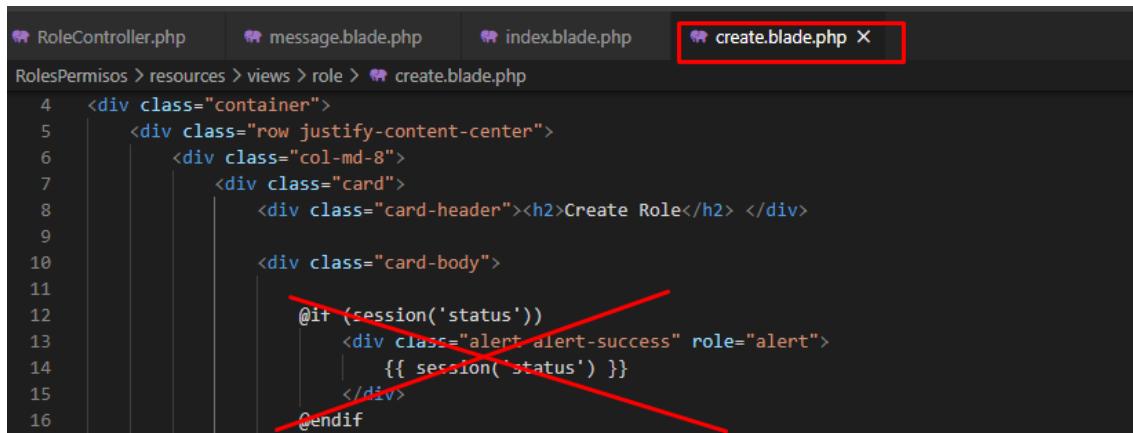
```
RoleController.php message.blade.php index.blade.php create.blade.php
RolesPermisos > resources > views > custom > message.blade.php
1
2 @if (session('status'))
3   <div class="alert alert-success" role="alert">
4     {{ session('status') }}
5   </div>
6 @endif
```

Ahora hacemos un cambio a “session(‘status’)” en este caso el parámetro de la sesión se llamará “sesión(‘status_success’)”:



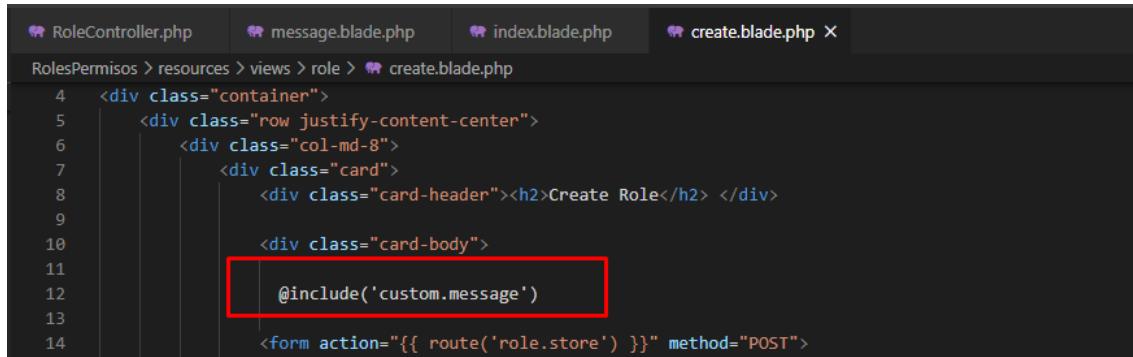
```
1
2 @if (session('status_success'))
3     <div class="alert alert-success" role="alert"
4         {{ session('status_success') }}
5     </div>
6 @endif
```

Ahora bien, de la misma manera hacemos en el archivo “role->create.blade.php”

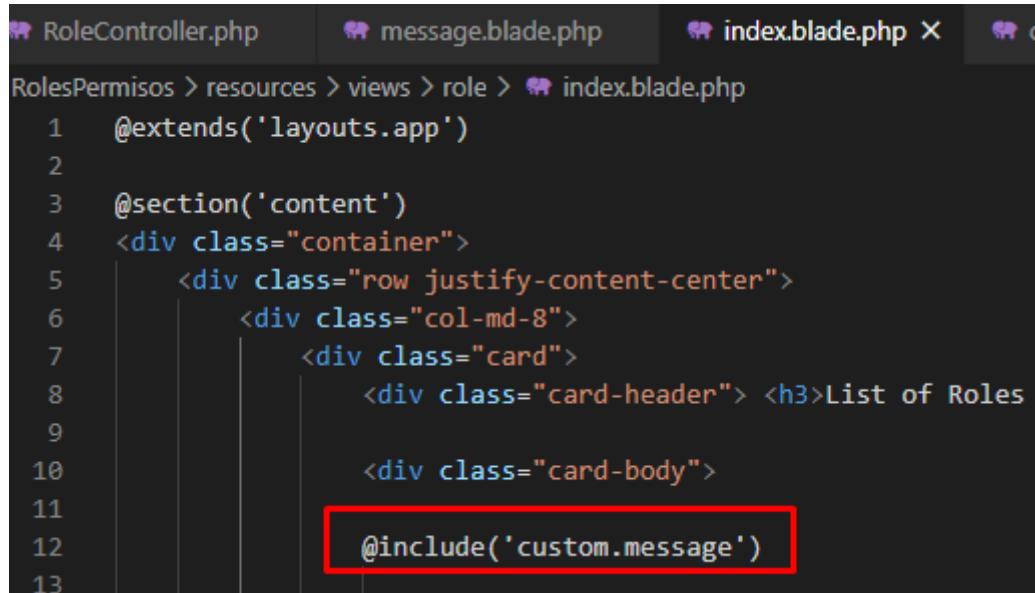


```
4 <div class="container">
5     <div class="row justify-content-center">
6         <div class="col-md-8">
7             <div class="card">
8                 <div class="card-header"><h2>Create Role</h2> </div>
9
10                <div class="card-body">
11
12                    @if (session('status'))
13                        <div class="alert alert-success" role="alert">
14                            {{ session('status') }}
15                        </div>
16                    @endif
```

Eliminamos ese código. Lo que faremos a continuación será incluir nuestra vista creada “message.blade.php” la vamos a incluir dentro de create.blade.php y en index.blade.php en efecto ser el reemplazo del código cortado/eliminado.

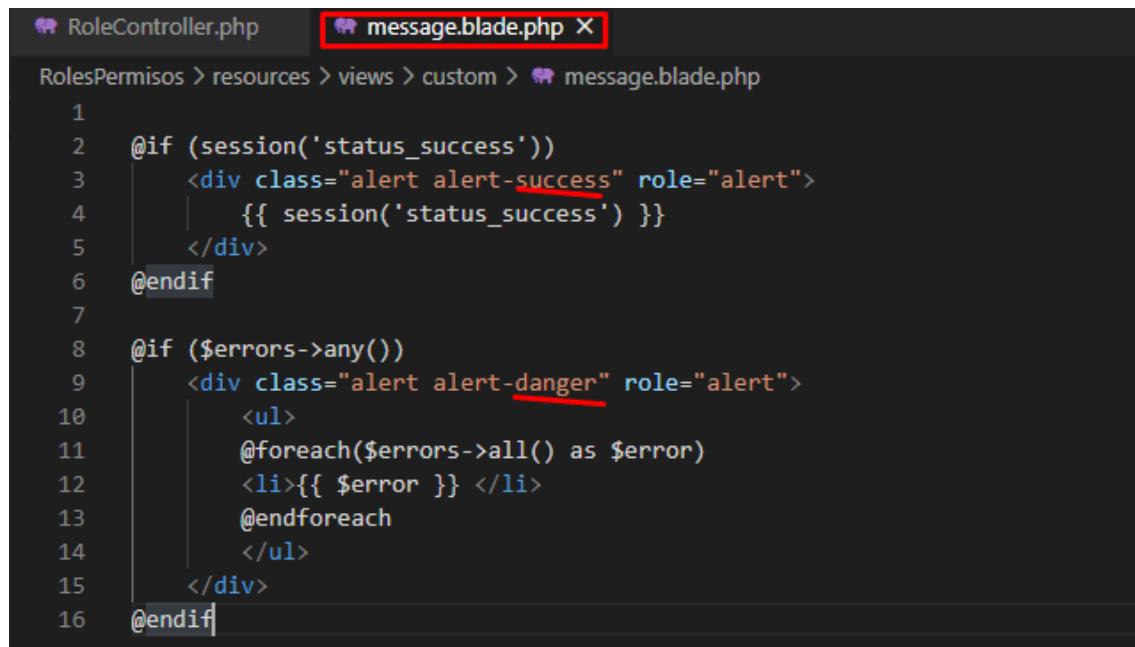


```
4 <div class="container">
5     <div class="row justify-content-center">
6         <div class="col-md-8">
7             <div class="card">
8                 <div class="card-header"><h2>Create Role</h2> </div>
9
10                <div class="card-body">
11
12                    @include('custom.message')
13
14                <form action="{{ route('role.store') }}" method="POST">
```



```
1  @extends('layouts.app')
2
3  @section('content')
4  <div class="container">
5      <div class="row justify-content-center">
6          <div class="col-md-8">
7              <div class="card">
8                  <div class="card-header"> <h3>List of Roles
9
10                 <div class="card-body">
11                     @include('custom.message')
12
13
```

Luego en el archivo creado llamado “message.blade.php” agregamos un código extra:



```
1
2  @if (session('status_success'))
3      <div class="alert alert-success" role="alert">
4          {{ session('status_success') }}
5      </div>
6  @endif
7
8  @if ($errors->any())
9      <div class="alert alert-danger" role="alert">
10         <ul>
11             @foreach($errors->all() as $error)
12                 <li>{{ $error }} </li>
13             @endforeach
14         </ul>
15     </div>
16  @endif
```

Básicamente lo que se esta haciendo ahí es mostrar dos alertas una la cual será de color verde y otra la danger que será de color rojo la cual mostrará los errores para eso se hizo un foreach. En el if se pregunta ¿existe algún error? Entonces se muestran en el foreach. La idea de ese archivo de message.blade.php es incluirlo en las vistas para mostrar los mensajes retornados desde el controlador RoleController.php

(*) Corrección: en el formulario de la vista “create.blade.php” el campo “name” era de tipo “mail” se ha cambiado a tipo “text” por solo se requiere un nombre y no un correo.

Lo siguiente es en el controlador agregar en el método “store()” las validaciones:

```
42     public function store(Request $request)
43     {
44         $request->validate([
45             'name' => 'required|max:50|unique:roles,name',
46             'slug' => 'required|max:50|unique:roles,slug',
47             'full-access' => 'required|in:yes,no'
48         ]);
49
50         return $request->all();
51     }
52 
```

RoleController.php

Explicación:

Aquí se valida que el campo “name” sea obligatorio por eso tiene la propiedad “required”. Que tenga 50 caracteres como mínimo y que sea único para eso se usa la propiedad “unique” la cual busca en la tabla “roles” el campo “name”. De la misma manera se hace con el “slug”.

El campo “full-access” solo admite con la propiedad “in” que sea su valor que le envíen “yes” o “no”.

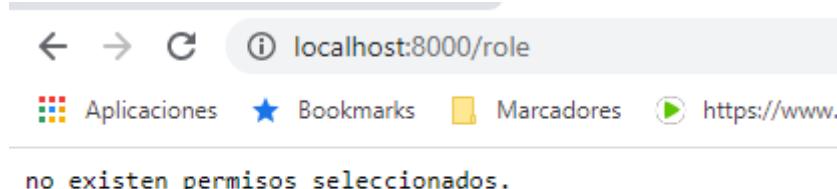
The screenshot shows a browser window with the URL `localhost:8000/role/create`. The page title is "Create Role". Below the title, there is a red box containing two error messages: "The name has already been taken." and "The slug has already been taken.". Below this error box, there is a section titled "Required Data" with three input fields: "Name", "slug", and "description". The browser's address bar and toolbar are visible at the top.

Antes de crear el “Role” debemos de validar que se seleccionen “Permisos” en el formulario de lo contrario no deberían de crear el rol. Otra cosa importante es que al momento de crear un Role solo se crearan los campos que se especificaron en el fillable en el modeo “Role”:

```
protected $fillable = [  
    'name', 'slug', 'description', 'full-access',  
];
```

```
public function store(Request $request)  
{  
    $request->validate([  
        'name' => 'required|max:50|unique:roles,name',  
        'slug' => 'required|max:50|unique:roles,slug',  
        'full-access' => 'required|in:yes,no'  
    ]);  
  
    //validar que los permissions existen:  
    if($request->get('permission')){  
        return $request->all();  
    }else{  
        return 'no existen permisos seleccionados.';  
    }  
    return $request->all();  
}
```

Bueno luego se prueba en el navegador sin haber marcado ningún Permission y se obtiene el mensaje de error:



Validado esto, entonces primero se registra el Role y luego se registran los permission para dicho rol con el método “sync()”.

Luego de registrar se hace un redireccionamiento al la vista “index” que se encuentra dentro de la carpeta de views “role” y le mandamos un mensaje con “with” en este caso mandamos la variable “status_success” con un mensaje.

```

public function store(Request $request)
{
    $request->validate([
        'name' => 'required|max:50|unique:roles,name',
        'slug' => 'required|max:50|unique:roles,slug',
        'full-access' => 'required|in:yes,no'
    ]);
    //Crear Role
    $role = Role::create($request->all());

    //validar que los permissions existen:
    if($request->get('permission')){
        //return $request->all();
        //Registrar los permisos en la tabla permission_role
        $role->permissions()->sync( $request->get('permission') );
    }
    return redirect()->route('role.index')
        ->with('status_success','Role saved successfully');
}

```

Lo siguiente es probar:

Creamos el “Role” llamado “test”:

```

public function store(Request $request)
{
    $request->validate([
        'name' => 'required|max:50|unique:roles,name',
        'slug' => 'required|max:50|unique:roles,slug',
        'full-access' => 'required|in:yes,no'
    ]);
    //Crear Role
    $role = Role::create($request->all());

    //validar que los permissions existen:
    if($request->get('permission')){
        //return $request->all();
        //Registrar los permisos en la tabla permission_role
        $role->permissions()->sync( $request->get('permission') );
    }
    return redirect()->route('role.index')
        ->with('status_success','Role saved successfully');

    //return $request->all();
}

```

Le damos en guardar y finalmente nos redirecciona a la vista index.

List of Roles

#	Name	Slug	Description	Full-access		1	Admin	admin	Administrator	yes	
3	test	test	test	no		Show	1	Admin	admin	Administrator	yes

Para corregir ese desorden en la tabla ponemos los <tr> dentro del foreach:

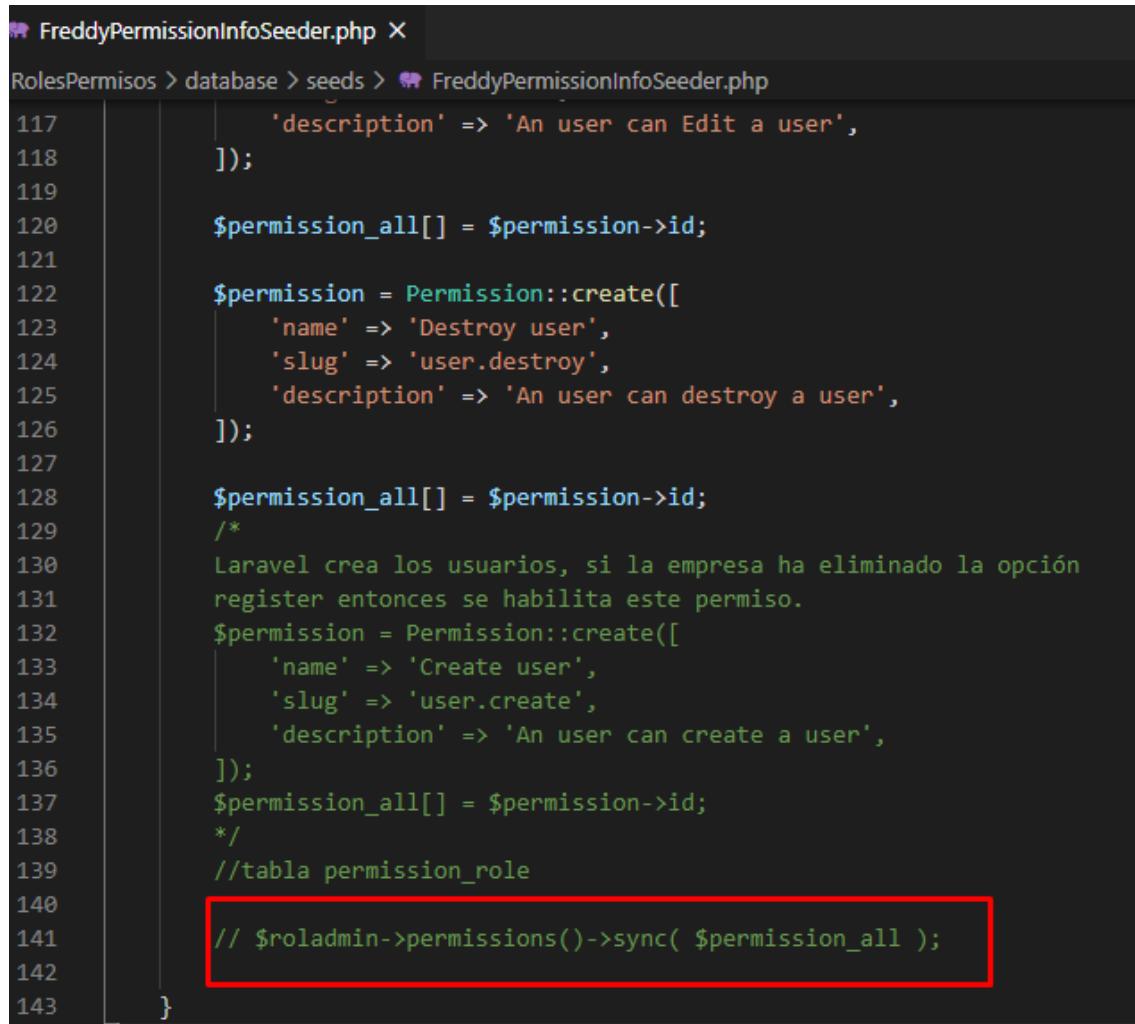
```
<tbody>
    @foreach($roles as $role)
        <tr>
            <th scope="row">{{ $role->id }}</th>
            <td>{{ $role->name }}</td>
            <td>{{ $role->slug }}</td>
            <td>{{ $role->description }}</td>
            <td>{{ $role['full-access'] }}</td>
            <td>
                <a class="btn btn-info" href="{{ route('role.show', $role->id)}}> Show </a>
                <a class="btn btn-success" href="{{ route('role.edit', $role->id)}}> Edit </a>
                <a class="btn btn-danger" href="{{ route('role.destroy', $role->id)}}> Delete </a>
            </td>
        </tr>
    @endforeach

```

En la base de datos podemos confirmar que se ha creado el Role y se le han asignado los permisos:

+ Opciones			id	role_id	permission_id	created_at	updated_at	permission_role
<input type="checkbox"/>				1	1	1	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				2	1	2	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				3	1	3	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				4	1	4	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				5	1	5	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				6	1	6	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				7	1	7	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				8	1	8	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				9	1	9	2020-09-09 21:10:56	2020-09-09 21:10:56
<input type="checkbox"/>				10	3	1	2020-09-10 17:58:17	2020-09-10 17:58:17
<input type="checkbox"/>				11	3	3	2020-09-10 17:58:17	2020-09-10 17:58:17
<input type="checkbox"/>				12	3	4	2020-09-10 17:58:17	2020-09-10 17:58:17

Luego se hace un comentario en el archivo “FreddyPermissionInfoSeeder” a la línea de código que le asigna todos los permisos creados al Role “admin”.



```
 117     'description' => 'An user can Edit a user',
 118 );
 119
 120 $permission_all[] = $permission->id;
 121
 122 $permission = Permission::create([
 123     'name' => 'Destroy user',
 124     'slug' => 'user.destroy',
 125     'description' => 'An user can destroy a user',
 126 ]);
 127
 128 $permission_all[] = $permission->id;
 129 /*
 130 Laravel crea los usuarios, si la empresa ha eliminado la opción
 131 register entonces se habilita este permiso.
 132 $permission = Permission::create([
 133     'name' => 'Create user',
 134     'slug' => 'user.create',
 135     'description' => 'An user can create a user',
 136 ]);
 137 $permission_all[] = $permission->id;
 138 */
 139 //tabla permission_role
 140
 141 // $roladmin->permissions()->sync( $permission_all );
 142
 143 }
```

Basicamente como el rol ‘admin’ ya tiene el full-access en yes no es necesario asignarle permissions.

Luego vamos a reniciar las tablas volviendo a hacer las migraciones y ejecutar el archivo del seeder:

```
#> php artisan migrate:fresh --seed
```

```
C:\laragon\www\RolesPermisos
λ php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table #> php artisan migrate:fresh --seed
Migrated: 2014_10_12_000000_create_users_table (0.54 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.42 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.26 seconds)
Migrating: 2020_09_08_170810_create_roles_table
Migrated: 2020_09_08_170810_create_roles_table (0.77 seconds)
Migrating: 2020_09_08_182701_create_role_user_table
Migrated: 2020_09_08_182701_create_role_user_table (1.53 seconds)
Migrating: 2020_09_08_215500_create_permissions_table
Migrated: 2020_09_08_215500_create_permissions_table (0.67 seconds)
Migrating: 2020_09_08_220408_create_permission_role_table
Migrated: 2020_09_08_220408_create_permission_role_table (1.69 seconds)
Seeding: FreddyPermissionInfoSeeder
Seeded: FreddyPermissionInfoSeeder (1.29 seconds)
Database seeding completed successfully.
```

Luego de haber echo esto podemos ver que solo existe el usuario administrador:

List of Roles					Create
#	Name	Slug	Description	Full-access	
1	Admin	admin	Administrator	yes	Show Edit Delete

Bien luego creamos un usuario llamado “Registered user” el cual solo tendrá los permisos de “Show Role” y “Show User”:

Required Data

Registered user

registereduser

Registered user

Full Access

Yes No

Permissions List

- 1 - List role (An user can list a role)
- 2 - Show role (An user can see a role)
- 3 - Create role (An user can create a role)
- 4 - Edit role (An user can Edit a role)
- 5 - Destroy role (An user can destroy a role)
- 6 - List user (An user can list a user)
- 7 - Show user (An user can see a user)
- 8 - Edit user (An user can Edit a user)

List of Roles

Role saved successfully

Create

#	Name	Slug	Description	Full-access		
2	Registered user	registereduser	Registered user	no	Show	Edit
1	Admin	admin	Administrator	yes	Show	Edit

Fuente: <https://www.youtube.com/watch?v=PpUv8FUSIxM>

❖ Uso de old en text, textarea, radio y checkbox para no perder datos - Roles y Permisos Laravel 7

En esta parte del sistema de roles y permisos lo que se hará es hacer uso de “old” para cuando se envie un formulario y si algo salio mal, esos datos en el formulario no se pierdan, ya que actualmente los datos se pierden cuando escribimos un Role repetido en la bd el controlador nos retorna un mensaje de error pero los datos del formulario se pierden, entonces para evitar eso hacemos uso de old.

Para probar el “old()” esta funcionando lo agregamos debajo del cierre de etiqueta del formulario en la vista role->create.blade.php

```
61 |         </div>
62 |     </form>
63 |
64 |     {!! dd( old() ) !!}
65 |
66 |     </div>
```

Intentamos crear un role llamado “admin” debido a que ya existe el servidor nos responderá con los mensajes de error y lo que veremos abajo es que los datos con old si se conservan.

- 6 - List user (*An user can list a user*)
- 7 - Show user (*An user can see a user*)
- 8 - Edit user (*An user can Edit a user*)
- 9 - Destroy user (*An user can destroy a user*)

Save

```
array:6 [▼
  "_token" => "voprtcWcr3gjApyhu3WvicWX7kvrzs99tOK5pI6P"
  "name" => "admin"
  "slug" => "admin"
  "description" => "test"
  "full-access" => "yes"
  "permission" => array:2 [▼
    0 => "1"
    1 => "2"
  ]
]
```

Lo que debemos hacer ahora es aplicar old() a los input del formulario, dentro de la etiqueta value.

```

12     @include('custom.message')
13
14     <form action="{{ route('role.store') }}" method="POST">
15     @csrf
16     <div class="container">
17     <h3>Required Data</h3>
18     <div class="form-group">
19         <input type="text" class="form-control" value="{{ old('name') }}" name="name" id="name" placeholder="Name">
20     </div>
21
22     <div class="form-group">
23         <input type="text" class="form-control" value="{{ old('slug') }}" name="slug" id="slug" placeholder="slug">
24     </div>
25
26     <div class="form-group">
27         <textarea class="form-control" name="description" id="description" placeholder="description" rows="3">
28             {{ old('description') }}
29         </textarea>
30     </div>
31
32     <hr>

```

Ahora para los radiobutton tenemos que hacer una consulta con "if":

```

<h3>Full Access</h3>

<div class="custom-control custom-radio custom-control-inline">
<input type="radio" id="fullaccessyes" value="yes" name="full-access"
       class="custom-control-input"
       @if (old('full-access')=="yes")
           checked
       @endif
       >
<label class="custom-control-label" for="fullaccessyes">Yes</label>
</div>
<div class="custom-control custom-radio custom-control-inline">
<input type="radio" id="fullaccessno" value="no"
       @if (old('full-access')=="no")
           checked
       @endif
       name="full-access" class="custom-control-input">
<label class="custom-control-label" for="fullaccessno">No</label>
</div>

```

Sin embargo, con los **checkbox** es un poco diferente.

```

53
54     <hr>
55     <h3>Permissions List</h3>
56     @foreach($permissions as $permission)
57     <div class="custom-control custom-checkbox">
58         <input type="checkbox" class="custom-control-input"
59             id="permission_{{$permission->id}}"
60             value="{{$permission->id}}"
61             name="permission[]"
62             @if(is_array(old('permission')) && in_array("{$permission->id}", old('permission')))
63                 checked
64             @endif
65
66             >
67             <label class="custom-control-label" for="permission_{{$permission->id}}">
68                 {{ $permission->id }} -
69                 {{ $permission->name }}
70                 <em>({{ $permission->description }})</em>
71             </label>
72         </div>
73     @endforeach

```

Basicamente lo que hace el código es consultar si en el array “is_array()” existen el id “in_array()” entonces ponerle checked.

Lo que falta ahora es que cuando accedamos a la página “create.blade.php” se marque por defecto el radiobutton “no” porque hasta ahora no se marca ninguno por defecto solo se ha conservado su estado en caso de que algo salga mal.

Entonces eso se hace de la siguiente manera:

```
43 |         </div>
44 |         <div class="custom-control custom-radio custom-control-inline">
45 |             <input type="radio" id="fullaccessno" value="no"
46 |             @if (old('full-access')=="no")
47 |                 checked
48 |             @endif
49 |
50 |             @if (old('full-access')==null)
51 |                 checked
52 |             @endif
```

Básicamente le decimos que si el checkbox es null lo ponemos en checked, eso sucede al inicio cuando se carga la página por primera vez.

Fuente: <https://www.youtube.com/watch?v=2y8p44yNV9E>

❖ Metodos edit y update - Roles y Permisos Laravel 7

Lo primero que se hace es crear la vista “edit.blade.php” dentro de la carpeta de las vistas “role”. A esta vista lo que hacemos es copiar el código de “create.blade.php”.

```
1  @extends('layouts.app')
2
3  @section('content')
4  <div class="container">
5      <div class="row justify-content-center">
6          <div class="col-md-8">
7              <div class="card">
8                  <div class="card-header"><h2>Edit Role</h2> </div>
9
10             <div class="card-body">
11
12                 @include('custom.message')
13
14             <form action="{{ route('role.store') }}" method="POST">
15                 @csrf
16                 <div class="container">
17                     <h3>Required Data</h3>
18                     <div class="form-group">
19                         <input type="text" class="form-control" value="{{ old('name') }}" name="name" id="name"
20                         </div>
21             
```

Luego iremos haciendo la modificación.

En el controlador “RoleController.php” en el método “edit()” por defecto este método recibe como parámetro el \$id de la información a editar, sin embargo en este caso nosotros le vamos a pasar el nombre del modelo Role..

Lo normal se hace esto:

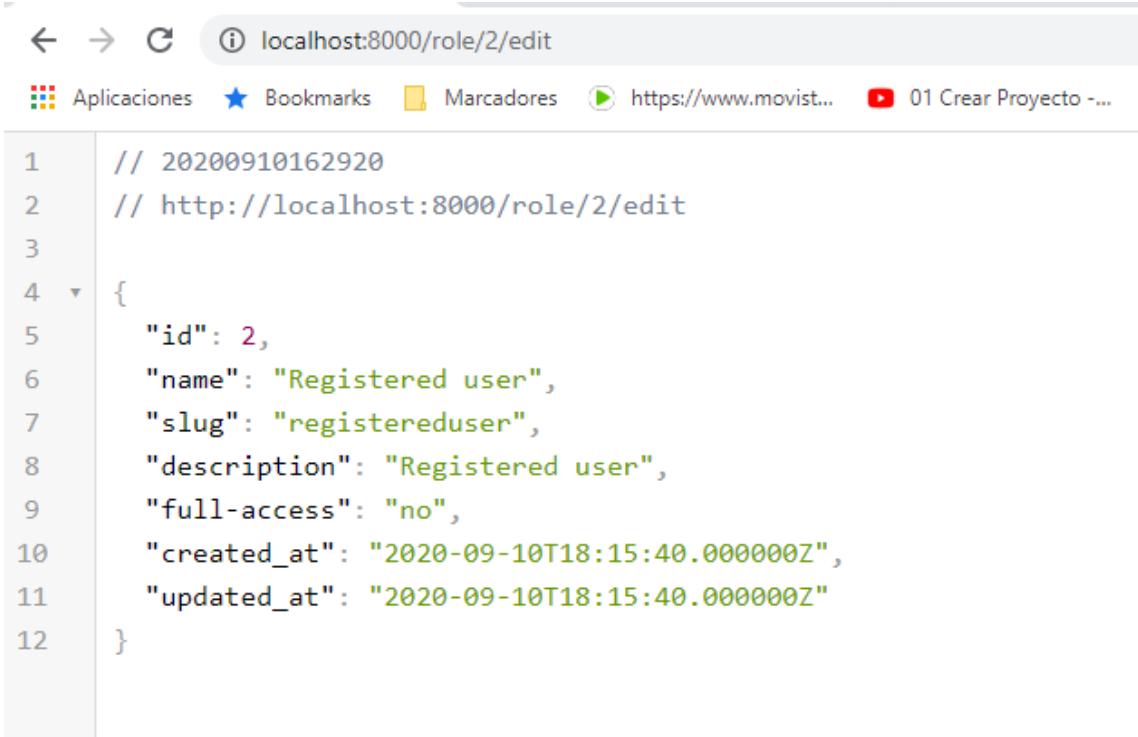
```
public function edit($id)  
{  
    return $id;  
}
```

Hasta ahí eso retornaría el \$id del role a editar

Al hacer esto:

```
    */
    public function edit(Role $role)
    {
        return $role;
    }
}
```

Eso retornaría toda la información del Role con un id determinado



The screenshot shows a browser window with the URL `localhost:8000/role/2/edit`. The page content is a JSON object representing a role:

```
1 // 20200910162920
2 // http://localhost:8000/role/2/edit
3
4 {
5     "id": 2,
6     "name": "Registered user",
7     "slug": "registereduser",
8     "description": "Registered user",
9     "full-access": "no",
10    "created_at": "2020-09-10T18:15:40.000000Z",
11    "updated_at": "2020-09-10T18:15:40.000000Z"
12 }
```

Por lo tanto eso nos evita hacer lo siguiente:

```
public function edit(Role $role)
{
    $role = Role::findOrFail($id);
    return $role;
}
```

Para evitar buscarlo entonces mejor le pasamos como parámetro el nombre del Modelo y la información quedaría guardada en la variable `$role`. Entonces lo que hacemos ahora es enviarle a la vista `edit`, los permission y el `$role` a editar:

```
public function edit(Role $role)
{
    $permissions = Permission::get();
    return view('role.edit', compact('permissions','role'));
}
```

Ahora en la vista “`edit.blade.php`”:

Modificamos el action del formulario pasándole el id al método update y especificamos que vamos a usar el método “PUT”

```
6   <div class="col-md-8">
7     <div class="card">
8       <div class="card-header"><h2>Edit Role</h2> </div>
9
10      <div class="card-body">
11        @include('custom.message')
12
13        <form action="{{ route('role.update', $role->id) }}" method="POST">
14          @csrf
15          @method('PUT')
16        <div class="container">
17          <h3>Required Data</h3>
18          <hr>
```

Tambien agregamos junto al old() que se encuentra dentro de los value la variable \$role que contiene el valor del role actual a editar.

```
<form action="{{ route('role.update', $role->id) }}" method="POST">
@csrf
@method('PUT')
<div class="container">
<h3>Required Data</h3>
<div class="form-group">
  <input type="text" class="form-control" value="{{ old('name', $role->name) }}" name="name" id="name">
</div>

<div class="form-group">
  <input type="text" class="form-control" value="{{ old('slug', $role->slug) }}" name="slug" id="slug" placeholder="slug">
</div>

<div class="form-group">
  <textarea class="form-control" name="description" id="description" placeholder="description" rows="3">
</div>

<hr>
<h3>Full Access</h3>
```

Respecto a los radiobutton esta parte de código no nos va a funcionar por lo tanto lo borramos:

```
34    <div class="custom-control custom-radio custom-control-inline">
35      <input type="radio" id="fullaccessyes" value="yes" name="full-access"
36      class="custom-control-input"
37      @if (old('full-access')=="yes")
38        checked
39      @endif
40    >
41    <label class="custom-control-label" for="fullaccessyes">Yes</label>
42  </div>
43  <div class="custom-control custom-radio custom-control-inline">
44    <input type="radio" id="fullaccessno" value="no"
45    class="custom-control-input"
46    @if (old('full-access')=="no")
47      checked
48    @endif
49    @if (old('full-access')===null)
50      checked
51    @endif
52    name="full-access" class="custom-control-input">
53    <label class="custom-control-label" for="fullaccessno">No</label>
54  </div>
```

Ahora el código de los radiobutton será diferente, en este caso se hace una validación accediendo al campo ‘full-access’ dependiendo de su valor el radiobutton tendrá la propiedad checked.

```
<div class="custom-control custom-radio custom-control-inline">
<input type="radio" id="fullaccessyes" value="yes" name="full-a
class="custom-control-input"
@if ($role['full-access']=="yes")
    checked
@elseif (old('full-access')=="yes")
    checked
@endif
>
<label class="custom-control-label" for="fullaccessyes">Yes</la
</div>
<div class="custom-control custom-radio custom-control-inline">
<input type="radio" id="fullaccessno" value="no"
@if ($role['full-access']=="no")
    checked
@elseif (old('full-access')=="no")
    checked
@endif
name="full-access" class="custom-control-input">
<label class="custom-control-label" for="fullaccessno">No</labe
</div>
```

Ahora para establecer los checkbox de acuerdo a los datos que traiga dicho “role” la función in_array() no es suficiente, para que funcione la búsqueda en in_array() los id deben de estar ordenados por coma, no podemos pasarle el objeto completo de todo el registro de los permisos.

Para ello iremos al controlador y agregar lo siguiente:

```
81
82     public function edit(Role $role)
83     {
84         $permission_role = [];
85         foreach($role->permissions as $permission){
86             $permission_role[] = $permission->id;
87         }
88         return $permission_role;
89
90
91         $permissions = Permission::get();
92         return view('role.edit', compact('permissions', 'role'));
93
94     }
```

De esa manera se obtendrán un array de id de los permission lo cual es compatible para lo que requiere la función “in_array()”.

```

1 // 20200910171540
2 // http://localhost:8000/role/3/edit
3
4 [ 1,
5   2,
6   3
7 ]
8 ]

```

De la siguiente manera se hace que se activen los checkbox de permissions que tiene asignado un determinado rol “fijémonos en el elseif” por que en la sentencia “if” se valida para cuando se envía el formulario pero en caso de retornar errores esos datos del checkbox se conserven “de repente el usuario asigne un rol mas por lo tanto no se pierde y se conserva que se ha marcado”... En el “elseif” estamos activando los permissions de acuerdo al array “\$permission_role” el cual es enviado desde el controlador:

```

<?php
    foreach($permissions as $permission)
        <div class="custom-control custom-checkbox">
            <input type="checkbox" class="custom-control-input"
                id="permission_{$permission->id}"
                value="{{$permission->id}}"
                name="permission[]"
            @if(is_array(old('permission')) && in_array("$permission->id", old('permission')))
                checked
            @elseif(is_array($permission_role) && in_array("$permission->id", $permission_role))
                checked
            @endif
        >
            <label class="custom-control-label" for="permission_{$permission->id}">
                {{ $permission->id }} -
                {{ $permission->name }}
                <em>({{ $permission->description }})</em>
            </label>
        </div>
    endforeach
    <hr>

```

El código del método “update” del controlador “RoleController.php” recibe toda la información. Primero recibe como parámetro “Role” directamente este ará la búsqueda del dato a actualizar. Luego se hacen las validaciones y se ha agregado al lado del campo único una “coma” y la variable con el \$id del role, esto lo que hace es hacer una excepción para poder conservar el “nombre del role en cuestión”, si el usuario ingresa un nombre distinto se realizará la validación correspondiente.

```
public function update(Request $request, Role $role)
{
    $request->validate([
        'name' => 'required|max:50|unique:roles,name,' . $role->id,
        'slug' => 'required|max:50|unique:roles,slug,' . $role->id,
        'full-access' => 'required|in:yes,no'
    ]);
    //Actualizar Role
    $role->update($request->all());

    //validar que los permissions existen:
    if($request->get('permission')){
        //return $request->all();
        //actualizar los permisos en la tabla permission_role
        $role->permissions()->sync( $request->get('permission') );
    }
    return redirect()->route('role.index')
        ->with('status_success','Role updated successfully');
}
```

Fuente: <https://www.youtube.com/watch?v=WV6PAgg0gdU>

❖ Métodos view y destroy - Roles y Permisos Laravel 7

Lo primero que se va hacer en esta etapa, es hacer una modificación dentro de la tabla que lista los Roles. Lo que haremos será poner un formulario que permita eliminar directamente un role:

```
54      @foreach($roles as $role)
55      <tr>
56          <th scope="row">{{ $role->id }}</th>
57          <td>{{ $role->name }}</td>
58          <td>{{ $role->slug }}</td>
59          <td>{{ $role->description }}</td>
60          <td>{{ $role['full-access'] }}</td>
61          <td>
62              <a class="btn btn-info" href="{{ route('role.show', $role->id)}}> Show </a>
63              <a class="btn btn-success" href="{{ route('role.edit', $role->id)}}> Edit </a>
64
65              <form action="{{ route('role.destroy', $role->id)}}" method="POST">
66                  @csrf
67                  @method('DELETE')
68                  <button class="btn btn-danger"> Delete </button>
69              </form>
70
71          </td>
72      </tr>
73  @endforeach
74
```

Lo siguiente es hacer las modificaciones dentro del RoleController.php. Lo primero que se hace es modificar el método “show()”:

```
public function show(Role $role)
{
    $permission_role = [];
    foreach($role->permissions as $permission){
        $permission_role[] = $permission->id;
    }
    $permissions = Permission::get();

    return view('role.view', compact('permissions','role','permission_role'));
}
```

Por el momento no hemos creado la vista “view.blade.php” pero loharemos luego.

Lo siguiente es modificar el método “destroy()”:

```
137  public function destroy(Role $role)
138  {
139      //Eliminar role
140      $role->delete();
141      return redirect()->route('role.index')
142          ->with('status_success','Role successfully removed!');
143
144 }
```

Lo siguiente es crear la vista “view.blade.php” dentro de la carpeta “role” de las vistas y el código que tendrá esta vista será el código de la vista “edit.blade.php” ahí realizaremos solo unos ligeros cambios agregando unas propiedades a los input, radiobuttons y checkbox.

El objetivo de esta vista es mostrar los datos al usuario pero de forma deshabilitada para ello agregamos la siguientes propiedades:

Input tipo text “**readonly**”, para los radiobuttons y checkbox les agregamos “**disabled**”

```
11 |     @include('custom.message')
12 |
13 |     @form action="{{ route('role.update', $role->id) }}" method="POST">
14 |     @csrf
15 |     @method('PUT')
16 |     <div class="container">
17 |         <h3>Required Data</h3>
18 |         <div class="form-group">
19 |             <input readonly type="text" class="form-control" value="{{ old('name', $role->name) }}" name="name"
20 |         </div>
21 |
22 |
23 |         <div class="form-group">
24 |             <input readonly type="text" class="form-control" value="{{ old('slug', $role->slug) }}" name="slug" id="slug"
25 |         </div>
26 |
27 |         <div class="form-group">
28 |             <textarea readonly class="form-control" name="description" id="description" placeholder="description">
29 |         </div>
30 |
31 |         <hr>
32 |         <h3>Full Access</h3>
```



```
33 |         <div class="custom-control custom-radio custom-control-inline">
34 |             <input disabled type="radio" id="fullaccessyes" value="yes" name="full-access"
35 |             class="custom-control-input"
36 |             @if ($role['full-access']=="yes")
37 |                 checked
38 |             @elseif (old('full-access')=="yes")
39 |                 checked
40 |             @endif
41 |         >
42 |         <label class="custom-control-label" for="fullaccessyes">Yes</label>
43 |     </div>
44 |     <div class="custom-control custom-radio custom-control-inline">
45 |         <input disabled type="radio" id="fullaccessno" value="no"
46 |         class="custom-control-input"
47 |         @if ($role['full-access']=="no")
48 |             checked
49 |         @elseif (old('full-access')=="no")
50 |             checked
51 |         @endif
52 |         name="full-access" class="custom-control-input">
53 |         <label class="custom-control-label" for="fullaccessno">No</label>
54 |     </div>
55 |
56 |         <hr>
```

Show Role

Required Data

Test

test

Testarea

Full Access

Yes No

Permissions List

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)
- 3 - Create role (*An user can create a role*)
- 4 - Edit role (*An user can Edit a role*)
- 5 - Destroy role (*An user can destroy a role*)
- 6 - List user (*An user can list a user*)
- 7 - Show user (*An user can see a user*)
- 8 - Edit user (*An user can Edit a user*)
- 9 - Destroy user (*An user can destroy a user*)

Tambien agregamos dos botones en la vista view.blade.php justo antes de cerrar el formulario. Este botón, el primero permitirá ir a editar el rol que se esta viendo y el segundo permitirá regresar al index.

```
</div>
@endforeach
<hr>

<a class="btn btn-success" href="{{ route('role.edit', $role->id) }}> Edit </a>
<a class="btn btn-danger" href="{{ route('role.index') }}> Back </a>
</div>
</form>
```

Fuente: <https://www.youtube.com/watch?v=VdoOfQfZRnI>

❖ Blindar el controlador con Gates - Roles y Permisos Laravel 7

Laravel provee dos tipos de blindajes, atraves de “gates” y “políticas”.

Lo primero que vamos a hacer es crear una ruta llamada “/test” en el archivo web.php

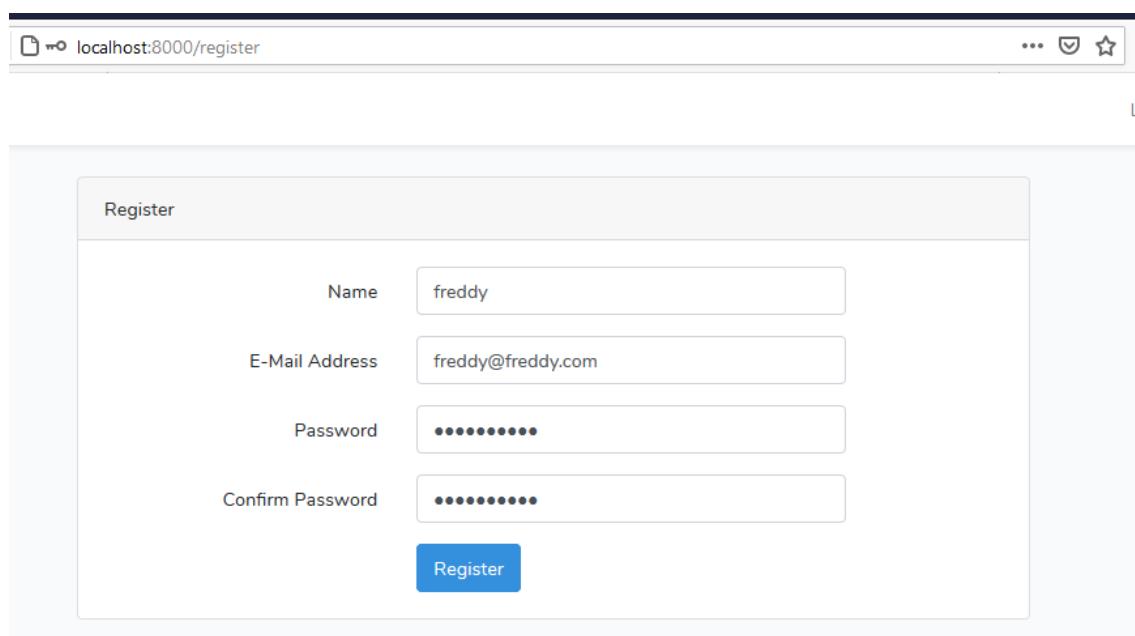
```
29  Route::get('/test', function () {  
30      return User::get();  
31  });
```

No debemos olvidar de agregar el “namespace” del user en el archivo web.php.

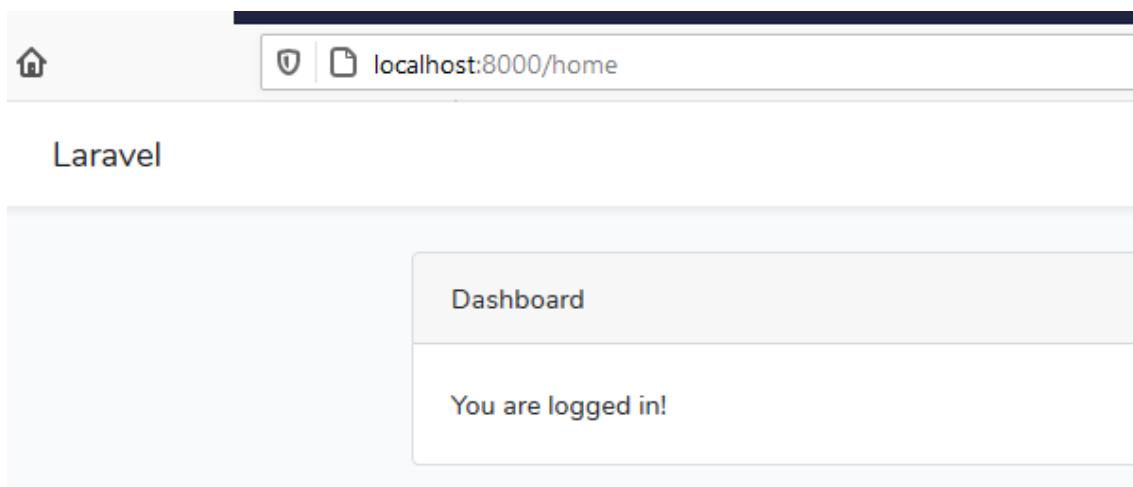
Verificamos en el navegador cuantos usuarios tenemos accediendo a la ruta creada:

```
1 // 20200911115448  
2 // http://localhost:8000/test  
3  
4 [  
5 {  
6     "id": 1,  
7     "name": "admin",  
8     "email": "admin@admin.com",  
9     "email_verified_at": null,  
10    "created_at": "2020-09-10T18:14:12.000000Z",  
11    "updated_at": "2020-09-10T18:14:12.000000Z"  
12 }  
13 ]
```

Lo siguientes es abrir “otro navegador” y crear un usuario cualquiera:



The screenshot shows a browser window with a registration form. The URL in the address bar is `localhost:8000/register`. The form is titled "Register". It has four input fields: "Name" (value: "freddy"), "E-Mail Address" (value: "freddy@freddy.com"), "Password" (value: masked), and "Confirm Password" (value: masked). Below the form is a blue "Register" button.



De esta forma tenemos dos usuarios, uno que es el admin y este ultimo nuevo que se ha creado.

Lo siguiente es asignarle permisos al “role” “registereduser” solo le asignamos los permisos seleccionados (list role).

A screenshot of a user role creation or assignment form. It has three main sections: 1. "Required Data" which contains input fields for "Registered user" (with "registereduser" selected), "registereruser" (disabled), and "Registered user" (disabled). 2. "Full Access" which includes a radio button for "No" (selected) and "Yes". 3. "Permissions List" which lists four permissions with checkboxes: 1 - List role (checked), 2 - Show role (unchecked), 3 - Create role (unchecked), and 4 - Edit role (unchecked).

Ahora si refrescamos la ruta “/test”, veremos que existen ya dos usuarios:

```

1 // 20200911120206
2 // http://localhost:8000/test
3
4 [
5   {
6     "id": 1,
7     "name": "admin",
8     "email": "admin@admin.com",
9     "email_verified_at": null,
10    "created_at": "2020-09-10T18:14:12.000000Z",
11    "updated_at": "2020-09-10T18:14:12.000000Z"
12  },
13  {
14    "id": 2,
15    "name": "freddy",
16    "email": "freddy@freddy.com",
17    "email_verified_at": null,
18    "created_at": "2020-09-11T16:58:39.000000Z",
19    "updated_at": "2020-09-11T16:58:39.000000Z"
20  }
21 ]

```

Ahora modificamos la ruta “/test” y le decimos que nos traiga solo el usuario con id=2 es decir estamos traendo el usuario que registramos últimamente.

```

Route::get('/test', function () {
    $user = User::find(2);

    return $user;
});|

```

```

1 // 20200911120420
2 // http://localhost:8000/test
3
4 {
5   "id": 2,
6   "name": "freddy",
7   "email": "freddy@freddy.com",
8   "email_verified_at": null,
9   "created_at": "2020-09-11T16:58:39.000000Z",
10  "updated_at": "2020-09-11T16:58:39.000000Z"
11 }

```

Lo que faremos a continuación es asignarle un “role” a este usuario, el role a asignarle será el del “registereduser” entonces le asignamos el role:

Para ello primero vemos que “id” tiene ese role, en este caso es “2”

+ Opciones		← →	id	name	slug	description	full-access	created_at	updated_at
<input type="checkbox"/>	Editar	Copiar	Borrar	1	Admin	admin	Administrator	yes	2020-09-10 18:14:12
<input type="checkbox"/>	Editar	Copiar	Borrar	2	Registered user	registereduser	Registered user	no	2020-09-10 18:15:40

Entonces le asignamos el role con “sync()”:

```
28
29     Route::get('/test', function () {
30         $user = User::find(2);
31
32         $user->roles()->sync([ 2 ]);
33         return $user;
34    });
```

Vamos al navegador y actualizamos la ruta “/test” para que se le asigne el role. Luego volveremos y comentaremos la línea de código que tiene la función “sync()”:

```
Route::get('/test', function () {
    $user = User::find(2);

    // $user->roles()->sync([ 2 ]);
    return $user->roles();
});
```

Ahora si volvemos a actualizar el navegador con la ruta “/test” veremos que el usuario ya tiene el role asignado:

```

1 // 20200911121035
2 // http://localhost:8000/test
3
4 [
5   {
6     "id": 2,
7     "name": "Registered user",
8     "slug": "registereduser",
9     "description": "Registered user",
10    "full-access": "no",
11    "created_at": "2020-09-10T18:15:40.000000Z",
12    "updated_at": "2020-09-10T18:15:40.000000Z",
13    "pivot": {
14      "user_id": 2,
15      "role_id": 2,
16      "created_at": "2020-09-11T17:10:21.000000Z",
17      "updated_at": "2020-09-11T17:10:21.000000Z"
18    }
19  }
20 ]

```

Lo siguiente es crear una función llamada “havePermission()” para ver si el usuario tiene el acceso , en este caso a ‘role.index’ definimos de esta forma la llamada a la función havePermission() en la ruta “/test”

```

29 Route::get('/test', function () {
30   $user = User::find(2);
31
32   // $user->roles()->sync([ 2 ]);
33
34   return $user->havePermission('role.index');
35 });
36

```

Luego nos iremos al modelo “User” que esta dentro de App>User.php y definimos la función en donde codificaremos toda la lógica para controlar el permiso:

```

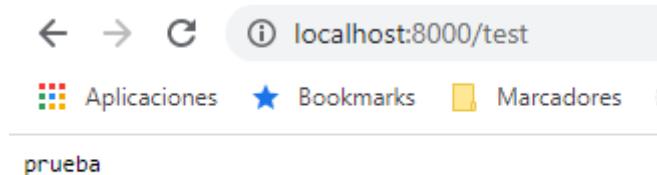
39 // es. desde aquí
40 // en. from here
41 public function roles(){
42   // relación muchos a muchos.
43   return $this->belongsToMany('App\FreddyPermisos\Models\Role')->withTimestamps();
44 }
45
46 public function havePermission($permission){
47   // lógica
48
49 }
50
51

```

Escribimos un código de prueba:

```
5      public function havePermission($permission){  
6          //logica  
7          return 'prueba';  
8      }  
9  
10  
11
```

Y refrescamos la ruta “/test” en el navegador y veremos que sale el mensaje “prueba”:



Ahora tratamos de retornar los roles:

```
45      public function havePermission($permission){  
46          //logica  
47          return $this->roles;  
48      }  
49  
50  
51
```

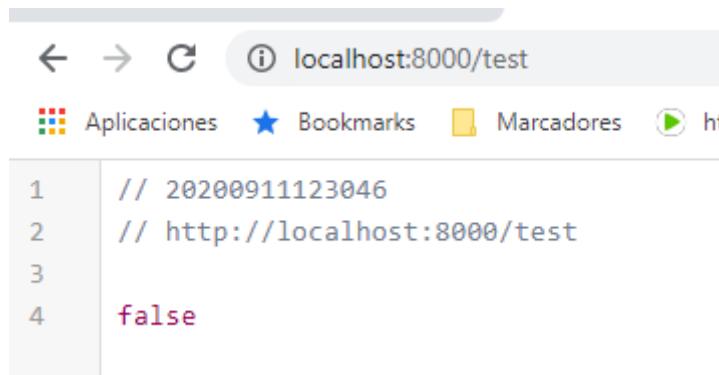
Refrescamos la ruta “/test” y veremos que se traen los roles:

```
4  [  
5  {  
6      "id": 2,  
7      "name": "Registered user",  
8      "slug": "registereduser",  
9      "description": "Registered user",  
10     "full-access": "no",  
11     "created_at": "2020-09-10T18:15:40.000000Z",  
12     "updated_at": "2020-09-10T18:15:40.000000Z",  
13     "pivot": {  
14         "user_id": 2,  
15         "role_id": 2,  
16         "created_at": "2020-09-11T17:10:21.000000Z",  
17         "updated_at": "2020-09-11T17:10:21.000000Z"  
18     }  
19 }  
20 ]
```

Lo que haremos ahora es verificar si el usuario tiene uno o mas roles asignados a la vez mediante un ciclo foreach. Lo primero es verificar si tiene “full-access” a modo de prueba en este caso se esta retornando en “texto”.

```
public function havePermission($permission){  
    //logica  
    //Esto verifica si el usuario tiene uno o mas r  
    foreach($this->roles as $role){  
        if($role['full-access']=='yes'){  
            return 'true full-access';  
        }  
    }  
    return 'false';  
  
    //return $this->roles;  
}
```

Ahora vamos al navegador y actualizamos:



Por lo tanto vemos que el usuario últimamente creado no tiene “full-access”, ahora vamos a editar el “role” que le asignamos y le decimos que si tiene full-access, además le quitamos el permiso de “List role”.

Required Data

Registered user
registereduser
Registered user

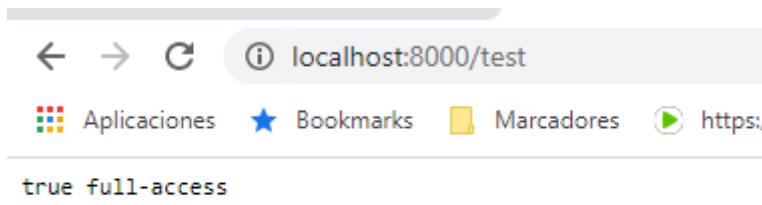
Full Access

Yes No

Permissions List

1 - List role (An user can list a role)
 2 - Show role (An user can see a role)
 3 - Create role (An user can create a role)
 4 - Edit role (An user can Edit a role)
 5 - Destroy role (An user can destroy a role)
 6 - List user (An user can list a user)
 7 - ...

Guardamos y ahora vamos a actualizar la ruta “/test”:



Vemos que se retorna “true full-access” sin embargo sucede un problema, si verificamos que el role “registereduser” se actualizo correctamente veremos que aún todavía tiene marcado el permission de “list role” a pesar de que ya se lo habíamos quitado:

Required Data

Registered user

registereduser

Registered user

Full Access

Yes No

Permissions List

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)

Para solucionar eso vamos a ir al controlador “RoleController.php” y en los métodos (store y update) comentamos lo siguiente:

```
41
42     public function store(Request $request)
43     {
44         $request->validate([
45             'name' => 'required|max:50|unique:roles,name',
46             'slug' => 'required|max:50|unique:roles,slug',
47             'full-access' => 'required|in:yes,no'
48         ]);
49         //Crear Role
50         $role = Role::create($request->all());
51
52         //validar que los permissions existen:
53         //if($request->get('permission')){
54             //return $request->all();
55             //Registrar los permisos en la tabla permission_role
56             $role->permissions()->sync( $request->get('permission') );
57         //}
58
59         return redirect()->route('role.index')
60             ->with('status_success','Role saved successfully');
61
62         //return $request->all();
63     }
```

```

public function update(Request $request, Role $role)
{
    $request->validate([
        'name' => 'required|max:50|unique:roles,name,'.$role->id,
        'slug' => 'required|max:50|unique:roles,slug,'.$role->id,
        'full-access' => 'required|in:yes,no'
    ]);
    //Actualizar Role
    $role->update($request->all());

    //validar que los permissions existen:
    //if($request->get('permission')){
        //return $request->all();
        //actualizar los permisos en la tabla permission_role
        $role->permissions()->sync( $request->get('permission') );
    //}
    return redirect()->route('role.index')
        ->with('status_success', 'Role updated successfully');
}

```

Esa verificamos decía que si “no hay ningún permiso marcado que no haga nada”. Pero si tiene por lo menos un permiso marcado que lo guarde con “sync()” Recordemos que sync() se encarga de recibir un arreglo y en función de este actualiza los datos (bien elimina o agrega/actualiza mas). En efecto si se guarda un role sin permisos se van a eliminar todos los permisos.

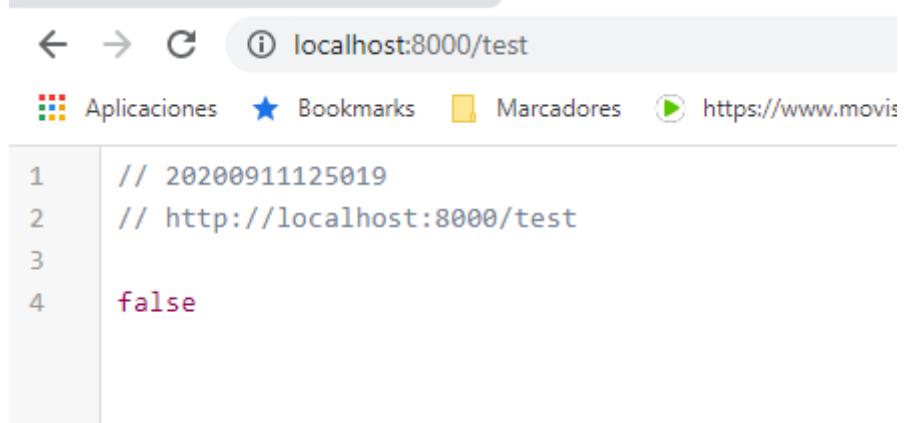
Ahora agregamos el siguiente código en el método havePermission() que esta dentro del modelo App>User.php

```

public function havePermission($permission){
    //lógica
    //Esto verifica si el usuario tiene uno o mas roles asi
    foreach($this->roles as $role){
        if($role['full-access']=='yes'){
            return 'true full-access';
        }
        //verificar si el role tiene un permiso asignado
        foreach($role->permissions as $perm){
            if($perm->slug == $permission){
                return 'true por permiso';
            }
        }
    }
    return 'false';
    //return $this->roles;
}

```

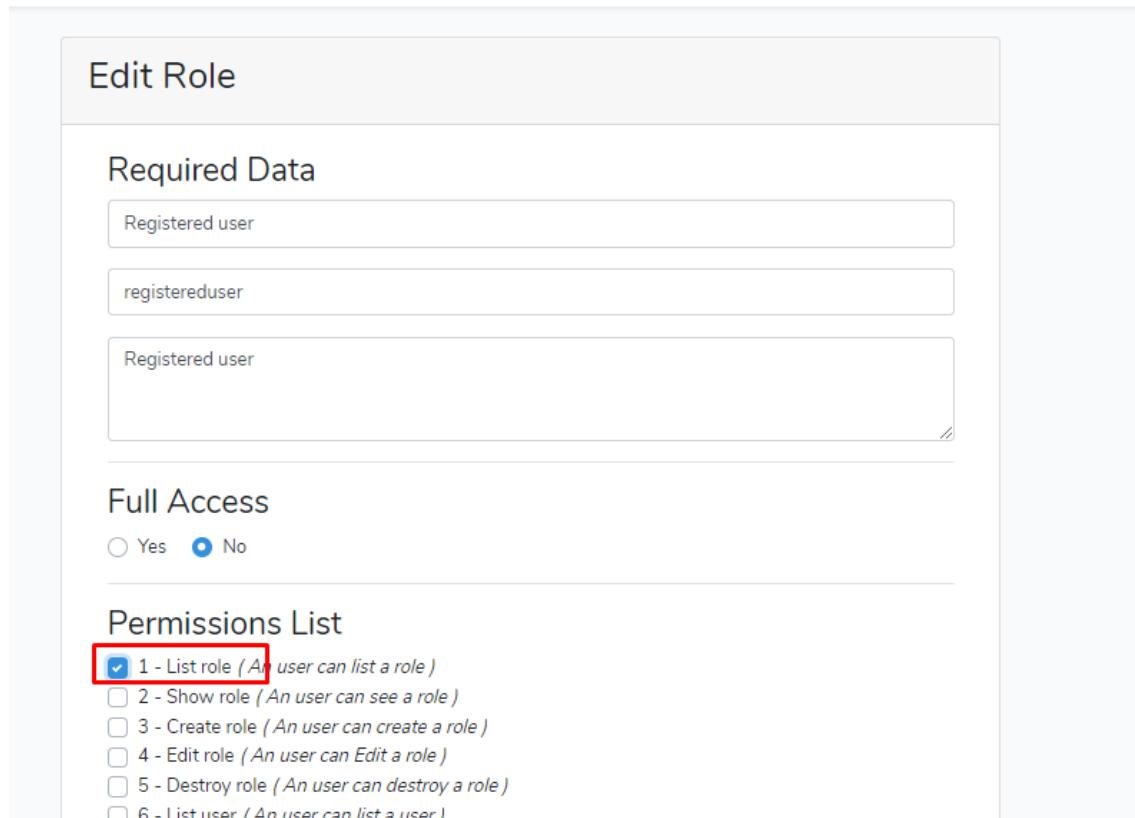
Si actualizamos la ruta “/test” veremos que retornaremos un “false” ya que el usuario últimamente creado con el role “registereduser” no tiene ningún permiso asignado y además no tiene full-access por eso se retornará false:



A screenshot of a web browser window. The address bar shows "localhost:8000/test". Below the address bar, there are links for "Aplicaciones", "Bookmarks", and "Marcadores". The main content area displays the following text:

```
1 // 20200911125019
2 // http://localhost:8000/test
3
4 false
```

Ahora si le asignamos un permiso al role que tiene asignado el usuario:



The screenshot shows a "Edit Role" form. It has two main sections: "Required Data" and "Permissions List".

Required Data:

- Role Name: registereduser
- Role Name: registereduser
- Role Name: registereduser

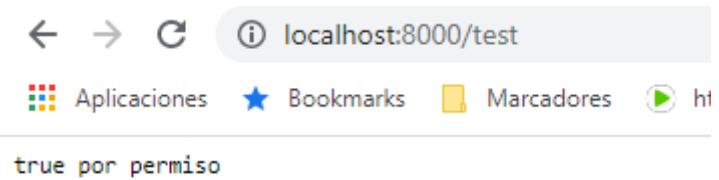
Full Access:

Yes No

Permissions List:

- 1 - List role (An user can list a role)
- 2 - Show role (An user can see a role)
- 3 - Create role (An user can create a role)
- 4 - Edit role (An user can Edit a role)
- 5 - Destroy role (An user can destroy a role)
- 6 - List user (An user can list a user)

Guardamos, luego actualizamos la ruta “/test” y veremos que se retorna “true”



El permiso asignado coincide con el parámetro ‘role.list’ en la ruta “/test”:

```
Route::get('/test', function () {
    $user = User::find(2);

    // $user->roles()->sync([ 2 ]);

    return $user->havePermission('role.index');
});
```

En este caso ‘role.index’ significa que el usuario puede listar. Pero ahora le cambiamos el parámetro y le ponemos ‘role.show’ esto indicaría que seria el permiso de ‘show role’ como el role “registereduser” no tiene asignado ese rol devolverá falso:

```
1 // 20200911125841
2 // http://localhost:8000/test
3
4 false
```

Ahora si volvemos si le asignamos al role ‘registereduser’ el permiso de “show role” devolverá true y de la misma manera si le ponemos full Access.

Required Data

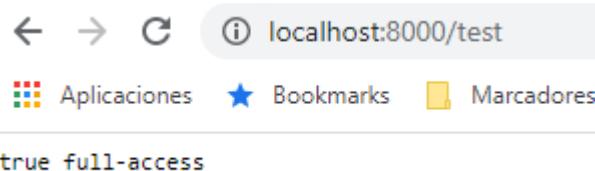
Registered user
registereduser
Registered user

Full Access

Yes No

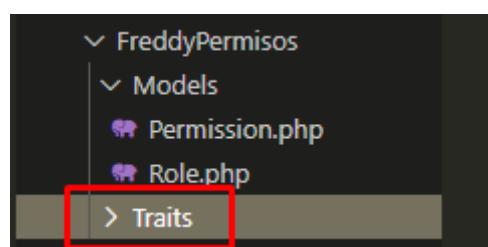
Permissions List

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)
- 3 - Create role (*An user can create a role*)

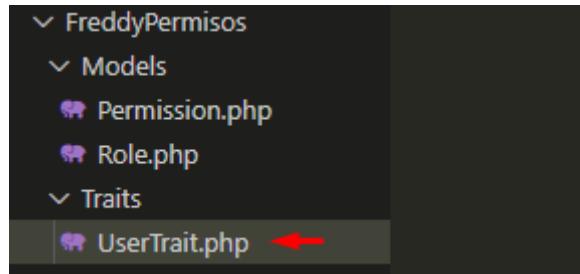


Lo haremos ahora crear un “Trait” La idea es de que este sistema se pueda instalar en otros sistema de otros clientes. La idea es de decirle al usuario/cliente que solo modifique el namespace del modelo User para poder incluir el Trait que se creara en la carpeta “FreddyPermisos”.. y nos llevaremos todo el código de la función havePermission() y las relaciones que existan dentro del modelo User al “trait”.

Entonces creamos la carpeta “Traits” dentro de “FreddyPermisos”:



Luego dentro de la carpeta “Traits” creamos el archivo “UserTrait.php”



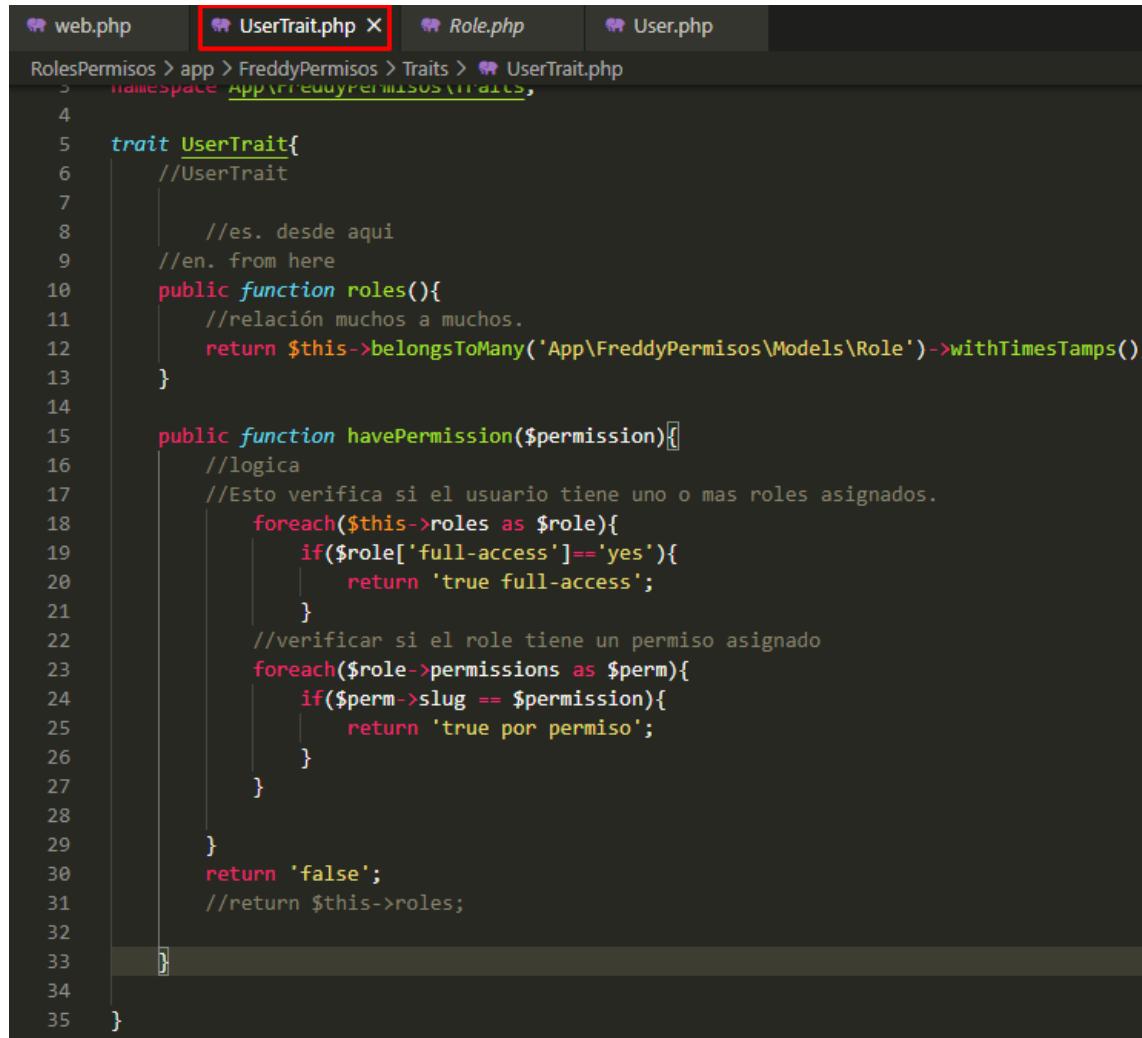
Y agregamos lo siguiente:

```
1 <?php
2
3
4
5 trait UserTrait{
6     //UserTrait
7
8
9 }
```

Luego agregamos el namespace:

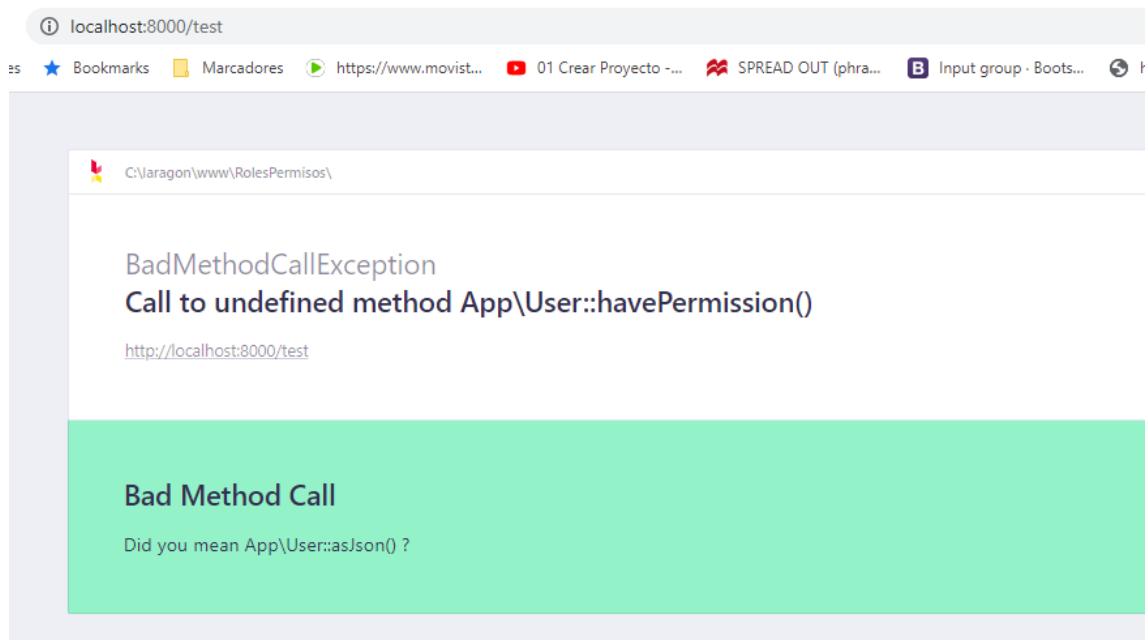
```
1 <?php
2
3 namespace App\FreddyPermisos\Traits;
4
5 trait UserTrait{
6     //UserTrait
7
8
9 }
```

Luego cortamos el código de la relación “roles” y la función “havePermission()” que existe en el modelo “User” y lo pegamos dentro del trait creado.



```
web.php          UserTrait.php X      Role.php      User.php
RolesPermisos > app > FreddyPermisos > Traits > UserTrait.php
3  namespace App\FreddyPermisos\Traits;
4
5  trait UserTrait{
6      //UserTrait
7
8      //es. desde aqui
9      //en. from here
10     public function roles(){
11         //relación muchos a muchos.
12         return $this->belongsToMany('App\FreddyPermisos\Models\Role')->withTimesTamps();
13     }
14
15     public function havePermission($permission){
16         //lógica
17         //Esto verifica si el usuario tiene uno o mas roles asignados.
18         foreach($this->roles as $role){
19             if($role['full-access']=='yes'){
20                 return 'true full-access';
21             }
22             //verificar si el role tiene un permiso asignado
23             foreach($role->permissions as $perm){
24                 if($perm->slug == $permission){
25                     return 'true por permiso';
26                 }
27             }
28         }
29         return 'false';
30         //return $this->roles;
31     }
32
33 }
34
35 }
```

Si vamos a la ruta “/test” y refrescamos nos dará un error:



Lo que haremos será usar el “Trait” llamado “UserTrait” dentro del modelo “User.php”

```
web.php UserTrait.php Role.php User.php X
RolesPermisos > app > User.php
1 <?php
2
3 namespace App;
4
5 use Illuminate\Contracts\Auth\MustVerifyEmail;
6 use Illuminate\Foundation\Auth\User as Authenticatable;
7 use Illuminate\Notifications\Notifiable;
8
9 use App\FreddyPermisos\Traits\UserTrait;
10
11 class User extends Authenticatable
12 {
13     use Notifiable, UserTrait;
14
15     /**
16      * The attributes that are mass assignable.
17      *
18      * @var array
```

The code editor shows the User.php file. Line 9, which contains 'use App\FreddyPermisos\Traits\UserTrait;', is highlighted with a red box. The UserTrait.php file is also visible in the tabs above the code editor.

Ahora si vamos al navegador y actualizamos la ruta “/test” debería de funcionar.

Esto nos va beneficiar que todo el código que le instalamos al cliente se realice dentro un Trait y solo tendrá que incluirlo dentro del modelo User.

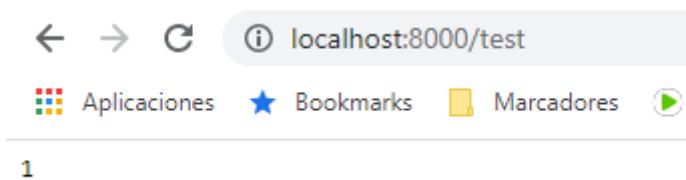
Bien ahora dentro de la función “havePermission()” ponemos los true y false respectivos sin comillas.



```
public function havePermission($permission){
    //lógica
    //Esto verifica si el usuario tiene uno o mas roles asignados
    foreach($this->roles as $role){
        if($role['full-access']=='yes'){
            return 'true full-access';
        }
    }
    //verificar si el role tiene un permiso asignado
    foreach($role->permissions as $perm){
        if($perm->slug == $permission){
            return 'true por permiso';
        }
    }
}
return 'false';
//return $this->roles;
```

```
public function havePermission($permission){
    //lógica
    //Esto verifica si el usuario tiene uno o mas roles asignados
    foreach($this->roles as $role){
        if($role['full-access']=='yes'){
            return true;
        }
    }
    //verificar si el role tiene un permiso asignado
    foreach($role->permissions as $perm){
        if($perm->slug == $permission){
            return true;
        }
    }
}
return false;
//return $this->roles;
```

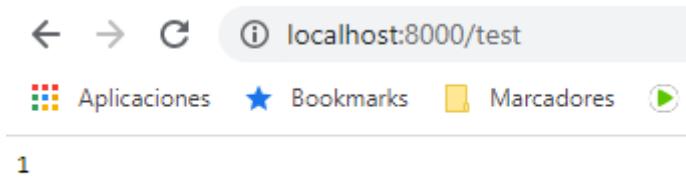
Ahora si vamos a la ruta “/test” y actualizamos retornará 1.



Ahora si vamos al archivo web.php y en la ruta “/test” le decimos que pase el parámetro “role.create” esto debería de retornar false debido a que el role “registereduser” no tiene asignado dicho role:

```
29 Route::get('/test', function () {
30     $user = User::find(2);
31
32     // $user->roles()->sync([ 2 ]);
33
34     return $user->havePermission('role.create');
35 });
36
```

Sin embargo, retorna 1:



Sin embargo esto no es problema, esta retornando 1 por que el role “registereduser” tiene asignado “full-access”:

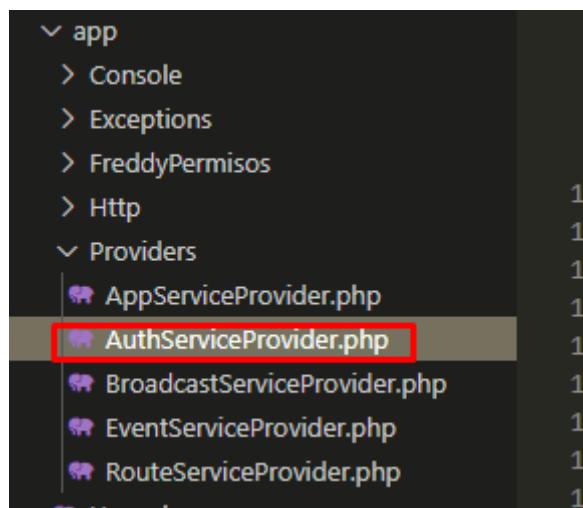
Full Access

Yes No

Ahora que ya tenemos todo esto creado por fin vamos a acceder a los “Gate” de laravel para tener todo parametrizado.

Lo quearemos será dirigirnos a la carpeta “App->Providers->AuthServiceProvider.php”

Y abrimos el archivo “AuthServiceProvider.php”



Aquí en este archivo es donde podemos hacer la magia de lo que vamos a hacer.

Existen dos formas de blindar la aplicación con laravel es con gates y políticas, para poder decirle al usuario si tiene acceso o no de lo contrario mostrar un error 403 de acceso restringido.

En este parte solo trabajaremos con “Gates”:

¿Cuál es la diferencia entre gates y políticas?

Gates: permite crear funciones anónimas y la idea es que nosotros podamos tener funciones de acceso no necesariamente para un modelo en específico sino algo genérico. Así con gates a través de una sola función en el Trait podemos controlar los accesos sin importar en qué modelo nos encontramos. Este aplica para todos los modelos en específico.

Políticas: Por otro lado, las políticas solo son aplicadas a un modelo en específico caso que no sucede con los gate.

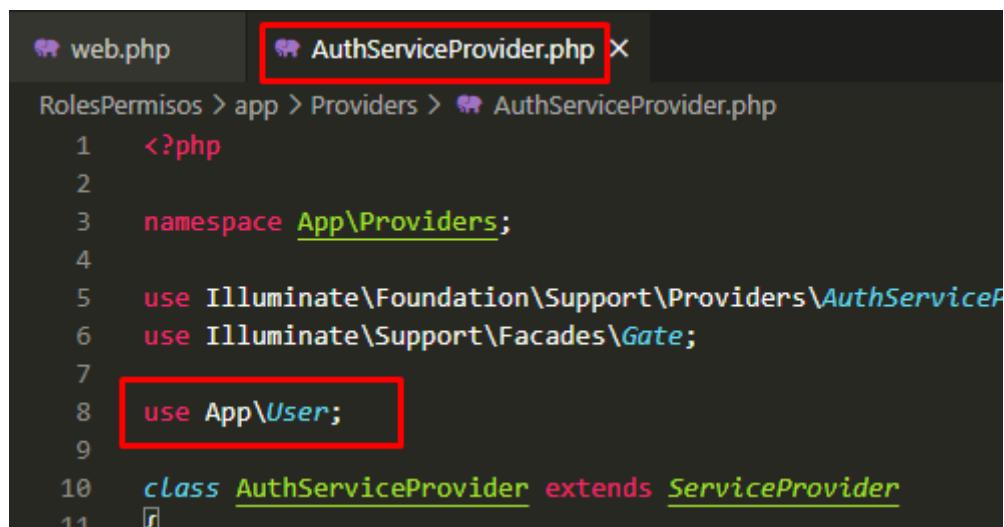
```
    /**
 * @var array
 */
protected $policies = [
    // 'App\Model' => 'App\Policies\ModelPolicy',
];
/**
```

(*) Los gates funcionan muy parecido a las rutas que están en el archivo web.php

Entonces dentro del archivo ubicamos la función “boot()” y agregamos lo siguiente:

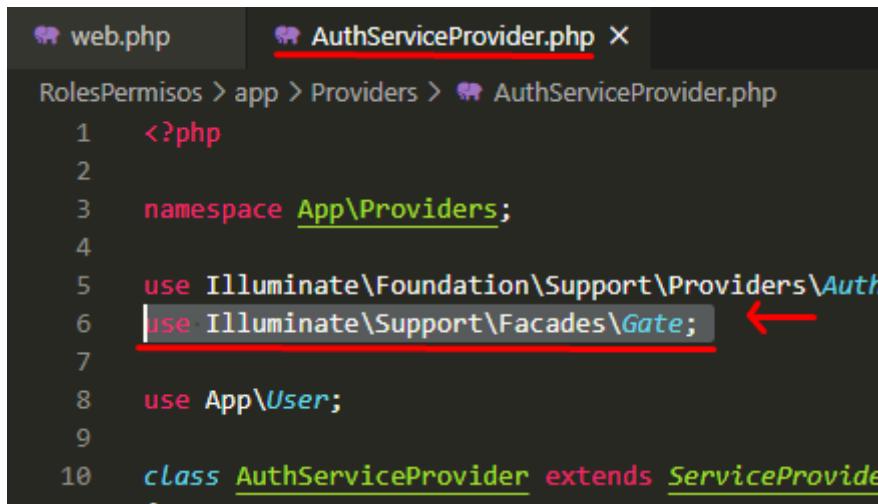
```
6     public function boot()
7     {
8         $this->registerPolicies();
9         //Definir Gate llamado 'haveaccess':
10        Gate::define('haveaccess', function(User $user){
11            return true;
12        });
13    }
14 }
```

Debido a que dentro de la función anónima se está pasando el modelo “User” como parámetro debemos de definir el namespace/uso:



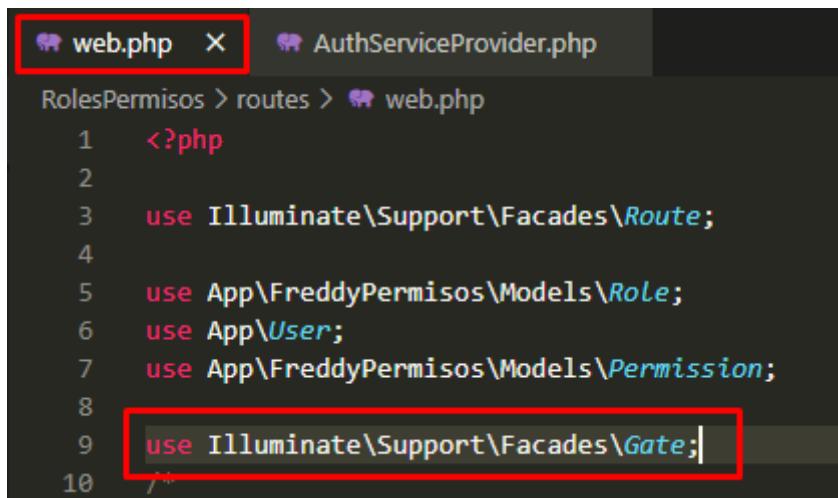
```
RolesPermisos > app > Providers > AuthServiceProvider.php X
1  <?php
2
3  namespace App\Providers;
4
5  use Illuminate\Foundation\Support\Providers\AuthServiceProvider;
6  use Illuminate\Support\Facades\Gate;
7
8  use App\User;
9
10 class AuthServiceProvider extends ServiceProvider
11 {
```

Ahora vamos a copiar el “namespace” de “Facades\Gate” que está dentro dentro de “AuthServiceProvider.php”:



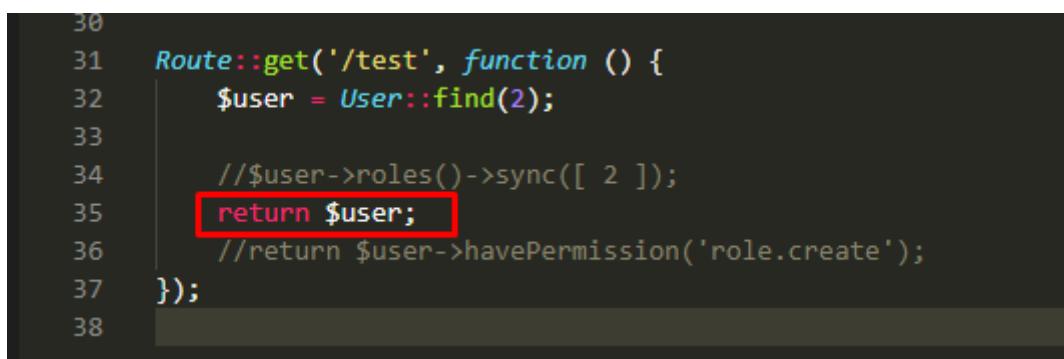
```
1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
6 use Illuminate\Support\Facades\Gate; ←
7
8 use App\User;
9
10 class AuthServiceProvider extends ServiceProvider
11 {
12 }
```

Lo copiamos y lo pegamos dentro de web.php:



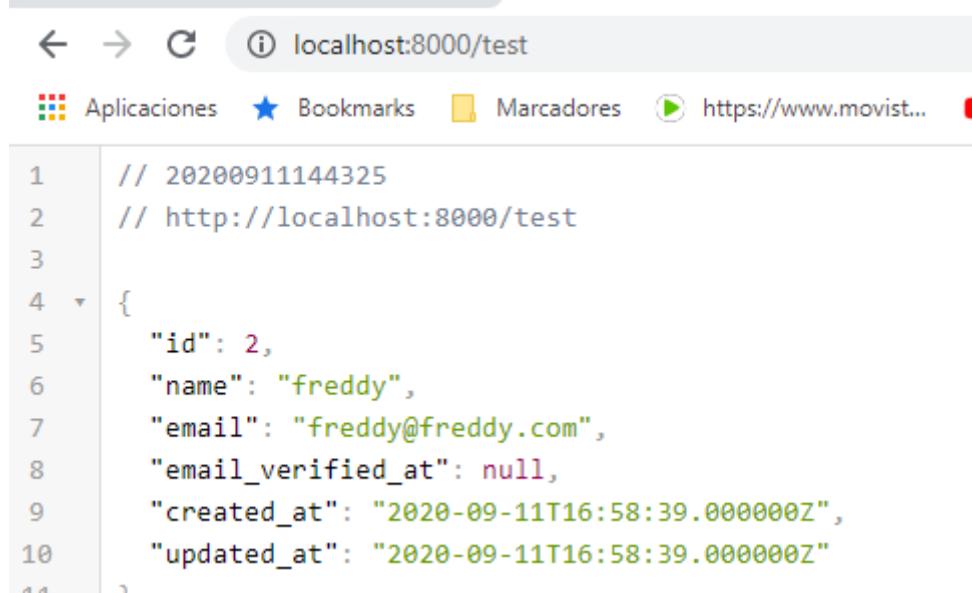
```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 use App\FreddyPermisos\Models\Role;
6 use App\User;
7 use App\FreddyPermisos\Models\Permission;
8
9 use Illuminate\Support\Facades\Gate; ←
10 /*
```

Ahora dentro de la ruta “/test” retornamos solo el usuario y verificamos en el navegador:



```
30
31 Route::get('/test', function () {
32     $user = User::find(2);
33
34     // $user->roles()->sync([ 2 ]);
35     return $user; ←
36     // return $user->havePermission('role.create');
37 });
38
```

Vemos a continuación que no hay problema:



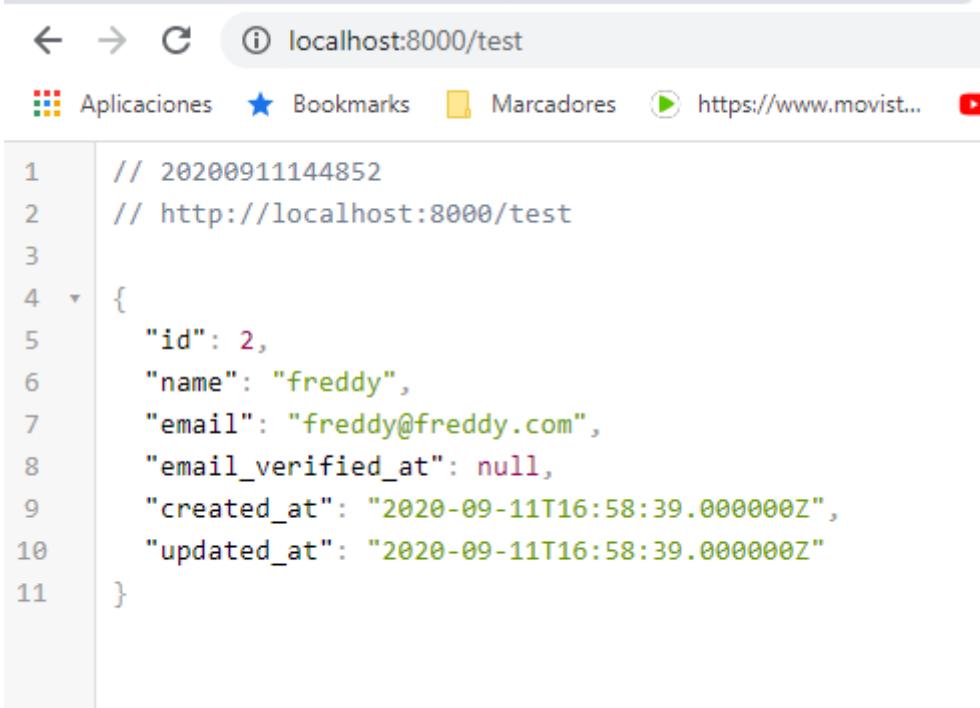
A screenshot of a web browser window. The address bar shows 'localhost:8000/test'. Below the address bar is a navigation bar with icons for back, forward, and refresh, followed by a search bar containing 'Aplicaciones' and a bookmark icon. The main content area displays a JSON object with the following structure:

```
1 // 20200911144325
2 // http://localhost:8000/test
3
4 {
5     "id": 2,
6     "name": "freddy",
7     "email": "freddy@freddy.com",
8     "email_verified_at": null,
9     "created_at": "2020-09-11T16:58:39.000000Z",
10    "updated_at": "2020-09-11T16:58:39.000000Z"
```

Ahora si definimos lo siguiente:

```
31 Route::get('/test', function () {
32     $user = User::find(2);
33     // $user->roles()->sync([ 2 ]);
34     Gate::authorize('haveaccess');
35     return $user;
36     // return $user->havePermission('role.create');
37 });
38 |
```

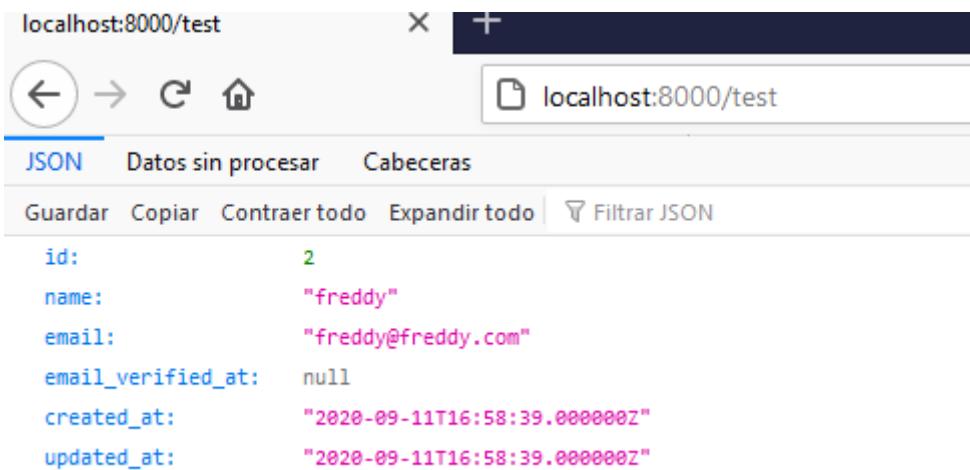
Primero verificamos que estamos logueados como “admin” y mostrará la información del usuario. Si no estamos logueados como ‘admin’ mostrará acceso restringido:



A screenshot of a web browser window. The address bar shows "localhost:8000/test". Below the address bar is a toolbar with icons for Applications, Bookmarks, and Favorites. The main content area displays the following JSON data:

```
1 // 20200911144852
2 // http://localhost:8000/test
3
4 {
5     "id": 2,
6     "name": "freddy",
7     "email": "freddy@freddy.com",
8     "email_verified_at": null,
9     "created_at": "2020-09-11T16:58:39.000000Z",
10    "updated_at": "2020-09-11T16:58:39.000000Z"
11 }
```

Verifico en el otro navegador “Firefox” logueado con el usuario freddy y vemos que podemos acceder sin embargo si no estamos logueados sale acceso no autorizado.



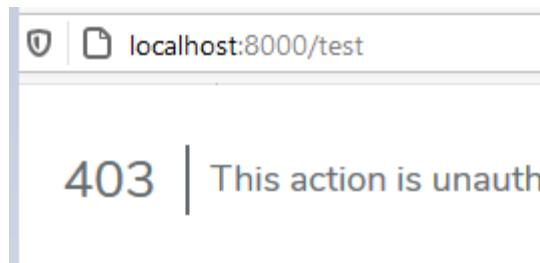
A screenshot of a Firefox browser window. The address bar shows "localhost:8000/test". The page content is a JSON object with the following fields and values:

```
id: 2
name: "freddy"
email: "freddy@freddy.com"
email_verified_at: null
created_at: "2020-09-11T16:58:39.000000Z"
updated_at: "2020-09-11T16:58:39.000000Z"
```

Ahora si al Gate le decimos que retorne “false”:

```
public function boot()
{
    $this->registerPolicies();
    //Definir Gate llamado 'haveaccess':
    Gate::define('haveaccess', function(User $user){
        return false;
    });
}
```

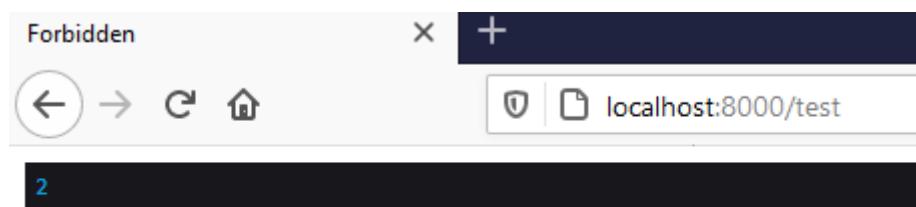
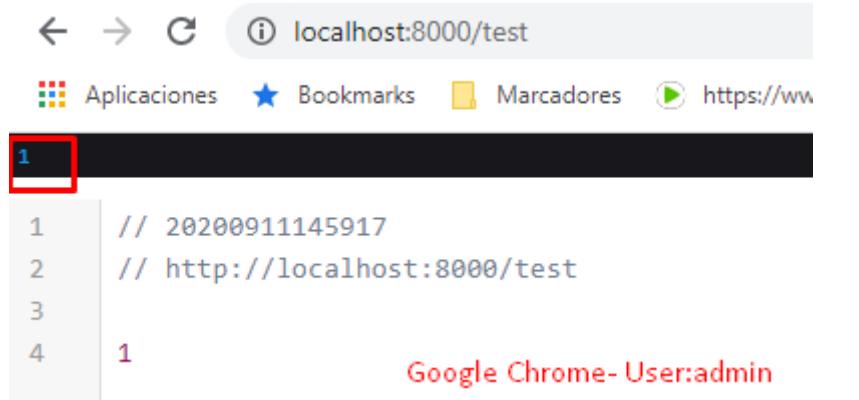
Se obtendrá acceso no autorizado:



Ahora veremos que el “Gate” si necesitad de decirle el id del usuario logueado este nos lo retornará: Esto sucede por que obtiene el id del usuario autenticado.

```
public function boot()
{
    $this->registerPolicies();
    //Definir Gate llamado 'haveaccess':
    Gate::define('haveaccess', function(User $user){
        dd($user->id);

        return true;
    });
}
```



Por lo tanto, los permisos se aplican al usuario logueado como tal.

Entonces lo único que vamos a hacer dentro de la función “boot()” que esta dentro del archivo AuthServiceProvider.php es lo siguiente:

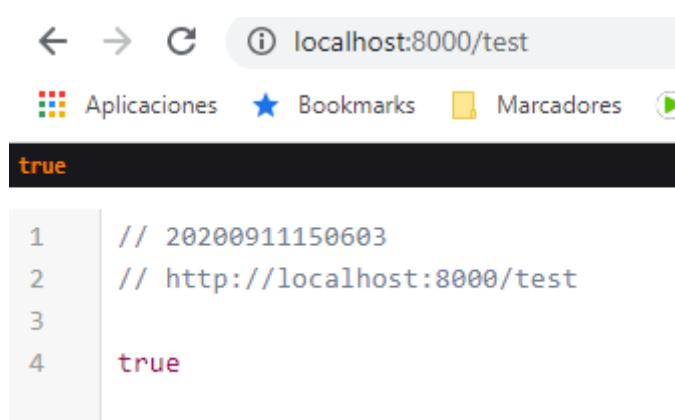
Aparte de agregar el parámetro del usuario vamos a agregar un nuevo parámetro llamado \$perm de “permiso” y vamos a retornar ese parámetro con “dd(\$perm)” para verificar que parámetro le estamos pasando desde la ruta “/test”:

¿Cómo le pasamos ese parámetro? Pues de la siguiente manera:

```
Route::get('/test', function () {
    $user = User::find(2);
    // $user->roles()->sync([ 2 ]);
    Gate::authorize('haveaccess', true);

    return $user;
    //return $user->havePermission('role.create');
});
```

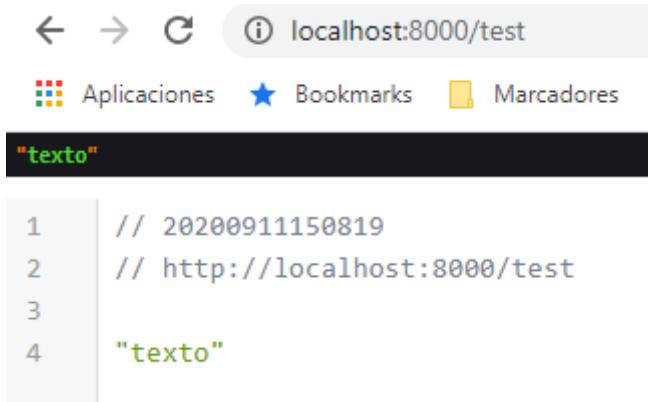
Eso retorna “true”:



Si le pasamos “false” retornará “false” si le pasamos un texto retornará un “texto”.

```
Route::get('/test', function () {
    $user = User::find(2);
    // $user->roles()->sync([ 2 ]);
    Gate::authorize('haveaccess', 'texto');

    return $user;
    //return $user->havePermission('role.create');
});
```



```
1 // 20200911150819
2 // http://localhost:8000/test
3
4 "texto"
```

Lo que necesitamos ahora es pasarle el “slug” del role que yo quiero. A continuación le pasaremos por ejemplo ‘role.index’

```
30
31 Route::get('/test', function () {
32     $user = User::find(2);
33     //$user->roles()->sync([ 2 ]);
34     Gate::authorize('haveaccess', 'role.index');
35
36     return $user;
37     //return $user->havePermission('role.create');
38 });
39
```

Ahora dentro de la función boot en el Gate vamos a retornar el permiso que se le pasa como parámetro:

```
/*
public function boot()
{
    $this->registerPolicies();
    //Definir Gate llamado 'haveaccess':
    Gate::define('haveaccess', function(User $user, $perm){
        //dd($perm);
        //$user->havePermission('role.create');
        return $perm;
    });
}
```

Ahora desde el archivo web.php si le pasamos como parámetro “false” nos dirá acceso no autorizado:

```
Route::get('/test', function () {
    $user = User::find(2);
    // $user->roles()->sync([ 2 ]);
    Gate::authorize('haveaccess', false);

    return $user;
    //return $user->havePermission('role.create'
});
```

403 | This action is unauthorized.

Si le pasamos ‘role.index’ nos permitirá acceder y obtener la información del usuario:

The screenshot shows a browser window with the URL 'localhost:8000/test'. The page displays a 403 error message: '403 | This action is unauthorized.'

```
1 // 20200911151400
2 // http://localhost:8000/test
3
4 {
5     "id": 2,
6     "name": "freddy",
7     "email": "freddy@freddy.com",
8     "email_verified_at": null,
9     "created_at": "2020-09-11T16:58:39.000000Z",
10    "updated_at": "2020-09-11T16:58:39.000000Z"
11 }
```

Ahora al role “registereduser” le quitamos todos los permisos y el ‘full-access’ en “no”.

Required Data

Registered user

registereduser

Registered user

Full Access

Yes No

Permissions List

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)
- 3 - Create role (*An user can create a role*)
- 4 - Edit role (*An user can Edit a role*)
- 5 - Destroy role (*An user can destroy a role*)
- 6 - List user (*An user can list a user*)
- 7 - Show user (*An user can see a user*)
- 8 - Edit user (*An user can Edit a user*)
- 9 - Destroy user (*An user can destroy a user*)



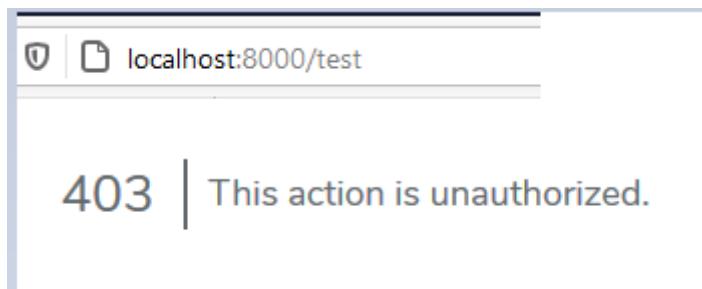
Ahora a nuestro Gate retornamos lo siguiente:

```
public function boot()
{
    $this->registerPolicies();
    //Definir Gate llamado 'haveaccess':
    Gate::define('haveaccess', function(User $user, $perm){
        //dd($perm);
        return $user->havePermission($perm);
        //return $perm;
    });
}
```

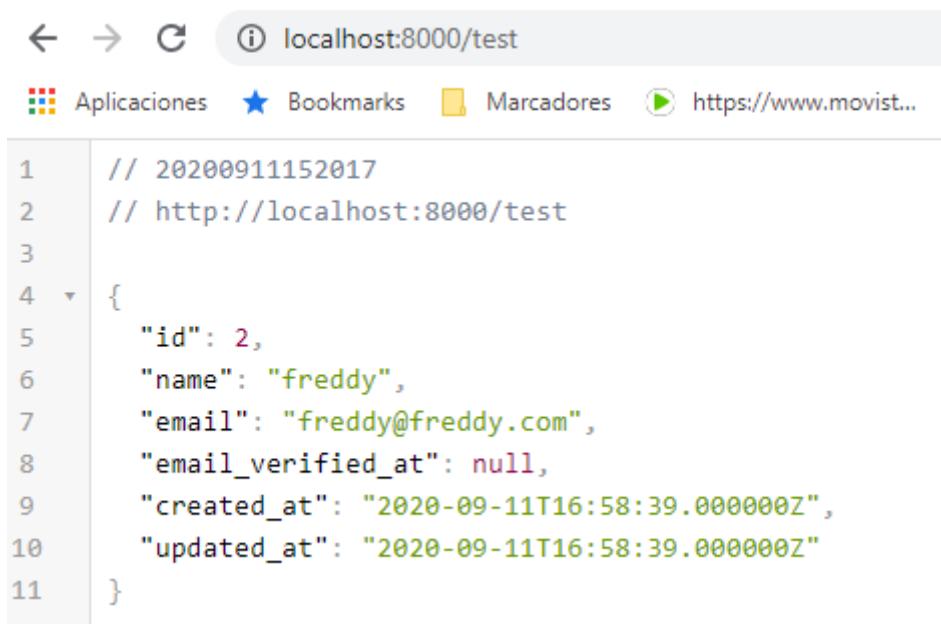
Ahora si intentamos con el usuario últimamente creado acceder a la ruta “/test” le dirá acceso restringido debido a que no tiene ningún permiso y el permiso que le estamos enviando como parámetro es ‘role.index’ lo cual no lo tiene asignado el role ‘registereduser’:

```
0
1 Route::get('/test', function () {
2     $user = User::find(2);
3     // $user->roles()->sync([ 2 ]);
4     Gate::authorize('haveaccess','role.index');
5
6     return $user;
7     //return $user->havePermission('role.create');
8 });
9
```

Por lo tanto, obtiene acceso restringido:



Sin embargo, el usuario 'admin' si tiene acceso:



Ahora si le agregamos el permiso al role 'registereduser' de 'List role' el usuario creado si podrá ver la información:

Permissions List

- 1 - List role (*An user can list a role*)

Verificamos en el navegador:



The screenshot shows a browser window with the address bar set to 'localhost:8000/test'. The page content is a JSON object representing a user with the following fields and values:

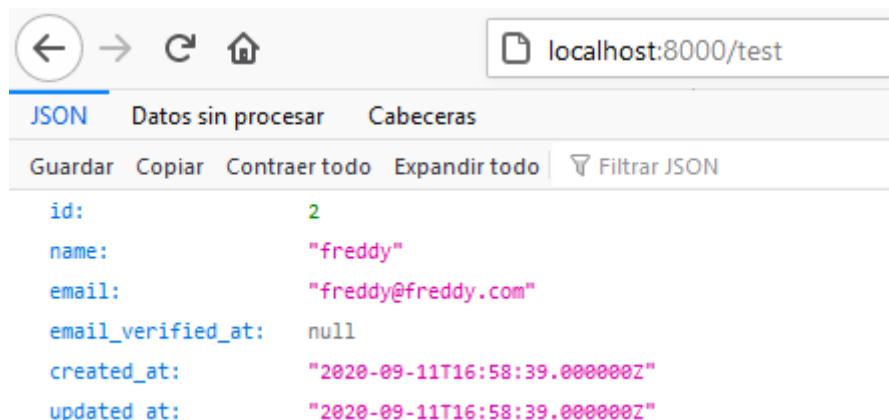
```
id: 2
name: "freddy"
email: "freddy@freddy.com"
email_verified_at: null
created_at: "2020-09-11T16:58:39.000000Z"
updated_at: "2020-09-11T16:58:39.000000Z"
```

Ahora le agregamos el permiso 'Show Role' al role 'registeruser' para que el usuario registrado pueda ver:

```
Route::get('/test', function () {
    $user = User::find(2);
    // $user->roles()->sync([ 2 ]);
    Gate::authorize('haveaccess', 'role.show');

    return $user;
    //return $user->havePermission('role.create');
});
```

Verificamos que puede ver:



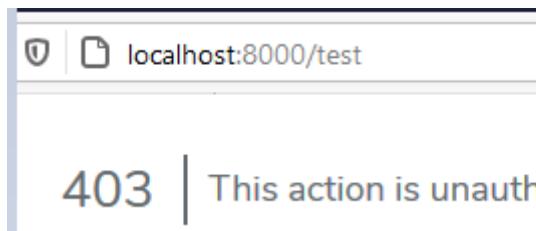
The screenshot shows a browser window with the address bar set to 'localhost:8000/test'. The page content is a JSON object representing the same user with the following fields and values:

```
id: 2
name: "freddy"
email: "freddy@freddy.com"
email_verified_at: null
created_at: "2020-09-11T16:58:39.000000Z"
updated_at: "2020-09-11T16:58:39.000000Z"
```

Ahora se lo quitamos:

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)
- 3 - Create role (*An user can create a role*)

El resultado es:



Ahora vamos a agregar los Gate dentro del controlador ‘RoleController.php’:

Primero lo hacemos dentro de la función index:

```
    */
public function index()
{
    Gate::authorize('haveaccess', 'role.index');

    $roles = Role::orderBy('id', 'Desc')->paginate(2);

    return view('role.index', compact('roles'));
}
```

Y también dentro del controlador agregamos el namespace del “Gate”:

```
... web.php RoleController.php X
RolesPermisos > app > Http > Controllers > RoleController.php
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 use App\FreddyPermisos\Models\Role;
8 use App\FreddyPermisos\Models\Permission;
9
10 use Illuminate\Support\Facades\Gate;
11
```

De igual manera agregamos el “Gate” dentro del método show:

```
public function show(Role $role)
{
    Gate::authorize('haveaccess', 'role.show');

    $permission_role = [];
    foreach($role->permissions as $permission){
        $permission_role[] = $permission->id;
    }
    $permissions = Permission::get();

    return view('role.view', compact('permissions', 'role', 'permission_role'));
}
```

De la misma manera en “create()”:

```
public function create()
{
    Gate::authorize('haveaccess', 'role.create');

    $permissions = Permission::get(); //::all
    return view('role.create', compact('permissions'));
}
```

De la misma manera en “edit()”:

```
public function edit(Role $role)
{
    Gate::authorize('haveaccess', 'role.edit');

    $permission_role = [];
    foreach($role->permissions as $permission){
        $permission_role[] = $permission->id;
    }

    $permissions = Permission::get();
    return view('role.edit', compact('permissions', 'role', 'permission_role'));
}
```

Ahora como no tenemos un permiso llamado 'Store role' pues esto se omitió por que es lo mismo que el permiso 'Create role' entonces le decimos al gate que si tenemos el permiso con slug 'role.create' nos permita guardar los datos.

```
public function store(Request $request)
{
    Gate::authorize('haveaccess','role.create');

    $request->validate([
        'name' => 'required|max:50|unique:roles,name',
        'slug' => 'required|max:50|unique:roles,slug',
        'full-access' => 'required|in:yes,no'
    ]);

    //Crear Role
    $role = Role::create($request->all());

    //validar que los permissions existen:
    //if($request->get('permission')){
    //    return $request->all();
    //    //Registrar los permisos en la tabla permission_role
    //    $role->permissions()->sync( $request->get('permission') );
    //}

    return redirect()->route('role.index')
        ->with('status_success','Role saved successfully');

    //return $request->all();
}
```

De la misma manera con el método 'update()':

```
"/"
public function update(Request $request, Role $role)
{
    Gate::authorize('haveaccess','role.edit');

    $request->validate([
        'name' => 'required|max:50|unique:roles,name,'.$role->id,
        'slug' => 'required|max:50|unique:roles,slug,'.$role->id,
        'full-access' => 'required|in:yes,no'
    ]);

    //Actualizar Role
    $role->update($request->all());
```

Actualizamos el método show, y vemos esto actualizado que también podemos de otra manera aplicar el Gate:

```
'public function show(Role $role)
{
    $this->authorize('haveaccess','role.show');

    $permission_role = [];
    foreach($role->permissions as $permission){
        $permission_role[] = $permission->id;
    }
    $permissions = Permission::get();

    return view('role.view', compact('permissions','role','permission_role'));
}
```

Sin necesidad de usar Gate:: en su lugar usamos \$this.

De la misma manera agregamos en “destroy()”:

```
'public function destroy(Role $role)
{
    Gate::authorize('haveaccess','role.destroy');

    //Eliminar role
    $role->delete();
    return redirect()->route('role.index')
        ->with('status_success','Role successfully removed!');
}
```

Y donde sea necesario si hace falta!.

En la imagen de a continuación veremos que el usuario registrado si pudo crear un Role debido a que tiene el permiso de crear:

Role creado con el usuario freddy:

List of Roles

Role saved successfully

Create

#	Name	Slug	Description	Full-access	Options
3	test1	test1	somdexcrip	no	<button>Show</button> <button>Edit</button> <button>Delete</button>
2	Registered user	registereduser	Registered user	no	<button>Show</button> <button>Edit</button> <button>Delete</button>

« 1 2 »

user=freddy

Ahora le damos el permiso de eliminar:

freddy ▾

List of Roles

Role successfully removed!

Create

#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	no	<button>Show</button> <button>Edit</button> <button>Delete</button>
1	Admin	admin	Administrator	yes	<button>Show</button> <button>Edit</button> <button>Delete</button>

Lo siguiente será crear la vista usuario y controlador mismo y ocultar botones y aplicar políticas.

FUENTE: <https://www.youtube.com/watch?v=BNY7IGv8JgQ>

❖ Crear controlador y vistas de los usuarios - Roles y Permisos Laravel

7

Esto será necesario para luego poder aplicar las políticas, ocultar botones/opciones del sistema.

Lo primero haremos será crear el controlador de tipo resource “UserController.php”:

```
#> php artisan make:controller UserController --resource
```

```
C:\laragon\www\RolesPermisos
└ php artisan make:controller UserController --resource
Controller created successfully.
```

Luego creamos la ruta “/user” y denegamos el acceso a los métodos del controlador “store” y “create” debido a que los usuarios ya pueden registrarse con la autenticación de laravel.

```
Route::resource('/role', 'RoleController')->names('role');

Route::resource('/user', 'UserController', ['except'=>
    'create', 'store']]->names('user');
```

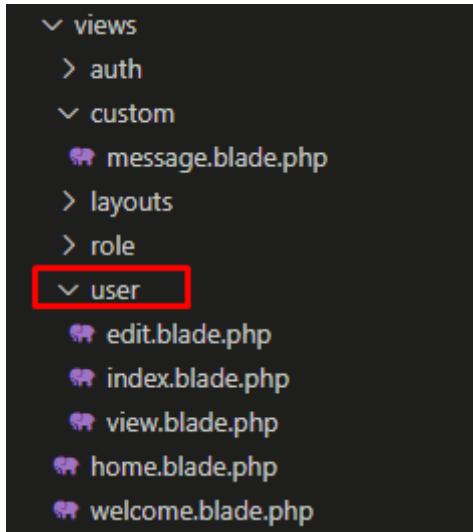
Listamos las rutas:

```
#> php artisan route:list
```

Veremos que el controlador “UserController” solo tiene acceso a las siguientes rutas y métodos:

route	name	closure
user	user.index	App\Http\Controllers\UserController@index
user/{user}	user.show	App\Http\Controllers\UserController@show
user/{user}	user.update	App\Http\Controllers\UserController@update
user/{user}	user.destroy	App\Http\Controllers\UserController@destroy
user/{user}/edit	user.edit	App\Http\Controllers\UserController@edit

Luego copiamos y pegamos la carpeta de vistas llamada “Role” con el nombre “User” y eliminamos la página “create.blade.php”



Como vemos hemos eliminado la página “create.blade.php”.

Luego en el controlador “UserController” agregamos los namespace de “User y Role”:

```
UserController.php X
RolesPermisos > app > Http > Controllers > UserController.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  use App\User;
8  use App\FreddyPermisos\Models\Role;
9
10 class UserController extends Controller
```

En el método index() del controlador “UserController” agregamos lo siguiente para retonar todos los usuarios:

```
public function index()
{
    $users = User::orderBy('id', 'Desc');

    return view('user.index', compact('users'));
}
```

Ahora nos vamos a la vista “User->index.blade.php” y hacemos las modificaciones pertinentes:

```
6   <div class="col-md-8">
7     <div class="card">
8       <div class="card-header"> <h3>List of Users </h3> </div>
9
10      <div class="card-body">
11
12        @include('custom.message')
13
14        <a href="{{ route('role.create') }}" class="btn btn-primary float-right">
15          Create
16        </a>
17        <br><br>
```

Como no creamos usuarios debido a que se registran, eliminamos ese código.

Antes de continuar modificando la vista “User->index.blade.php” lo que vamos hacer es decirle a la consulta que nos traiga los usuarios con los “roles” para ello usamos la opción “with()”:

Actualizamos en UserController.php:

```
public function index()
{
    $users = User::with('roles')->orderBy('id', 'Desc')->paginate(2);
    return $users;

    return view('user.index', compact('users'));
}
```

Eso nos traerá los usuarios con sus roles asociados:

```
r {
    "current_page": 1,
    "data": [
        {
            "id": 2,
            "name": "freddy",
            "email": "freddy@freddy.com",
            "email_verified_at": null,
            "created_at": "2020-09-11T16:58:39.000000Z",
            "updated_at": "2020-09-11T16:58:39.000000Z",
            "roles": [
                {
                    "id": 2,
                    "name": "Registered user", ←
                    "slug": "registereduser",
                    "description": "Registered user",
                    "full-access": "no",
                    "created_at": "2020-09-10T18:15:40.000000Z",
                    "updated_at": "2020-09-11T20:20:07.000000Z",
                    "pivot": {
                        "user_id": 2,
                        "role_id": 2,
                        "created_at": "2020-09-11T17:10:21.000000Z",
                        "updated_at": "2020-09-11T17:10:21.000000Z"
                    }
                }
            ]
        }
    ]
}
```

Sin embargo esa consulta retornará error si los usuarios no tiene roles.. entonces que debemos hacer..

Primero en la vista “User->index.blade.php” antes del luego del foreach que recorre todos los usuarios agregamos lo siguiente:

```

<table class="table table-hover">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">Name</th>
      <th scope="col">Email</th>
      <th scope="col">Role(s)</th>
      <th colspan="3">Options</th>
    </tr>
  </thead>
  <tbody>

    @foreach($users as $user)

      {{ $user }}

      <tr>
        <th scope="row">{{ $user->id }}</th>
        <td>{{ $user->name }}</td>
        <td>{{ $user->email }}</td>
        <td></td>

        <td>
          <a class="btn btn-info" href="{{ route('user.show', $user->id)}}> Show
          <form action="{{ route('user.destroy', $user->id)}}> Delete
        </td>
      </tr>
    @endforeach
  </tbody>

```

Con eso veremos que cada “user” trae un array “roles[]” el cual es la tabla

List of Users				
#	Name	Email	Role(s)	Options
2	freddy	freddy@freddy.com	[{"id":2,"name":"Registered user","slug":"registereduser","description":"Registered user","full-access":"no","created_at":"2020-09-10T18:15:40.000000Z","updated_at":"2020-09-11T17:10:21.000000Z","pivot":{"user_id":2,"role_id":2,"created_at":"2020-09-11T17:10:21.000000Z"}]]	Show Delete
1	admin	admin@admin.com	[{"id":1,"name":"Administrator","slug":"administrator","description":"Administrator","full-access":"yes","created_at":"2020-09-10T18:14:12.000000Z","updated_at":"2020-09-10T18:14:12.000000Z","pivot":{"user_id":1,"role_id":1,"created_at":"2020-09-10T18:14:12.000000Z"}]]	Show Delete

¿Cómo acceder a esa propiedad?

```

@foreach($users as $user)

{{ $user->roles[] }}

<tr>
<th scope="row">{{ $user->id }}</th>

```

Luego:

```

@foreach($users as $user)

{{ $user->roles[0]->name }}

```

Y vemos que obtenemos los roles:

List of Users		
#	Name	Email
1	Registered user Admin	admin@example.com

Entonces agregamos a la celda que está dentro del foreach dicho código:

```

@foreach($users as $user)
<tr>
<th scope="row">{{ $user->id }}</th>
<td>{{ $user->name }}</td>
<td>{{ $user->email }}</td>
<td>{{ $user->roles[0]->name }}</td>
<td>
<a class="btn btn-info" href="{{ route('user.show', $user->id) }}>View</a>

```

Y veremos que esto al parecer funciona:

List of Users

#	Name	Email	Role(s)	Options
2	freddy	freddy@freddy.com	Registered user	<button>Show</button> <button>Delete</button>
1	admin	admin@admin.com	Admin	<button>Show</button> <button>Delete</button>

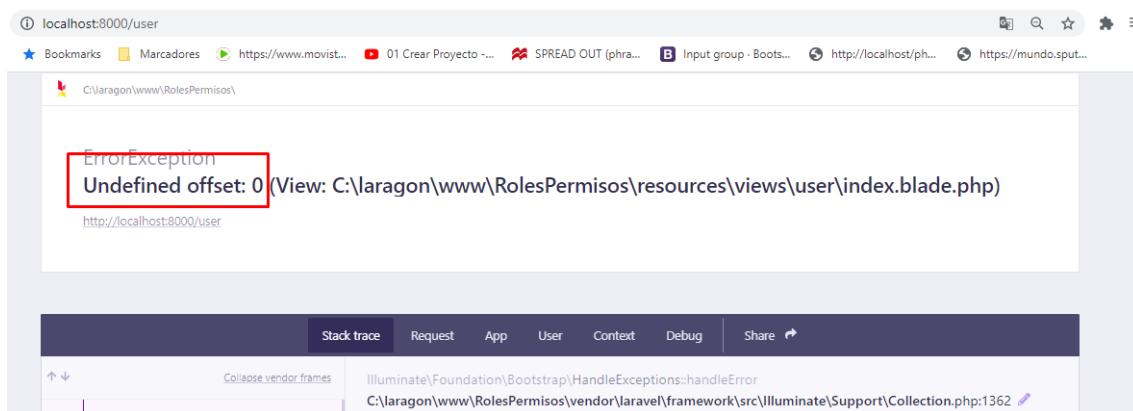
Como se aprecia si se esta trayendo correctamente el rol asociado a cada usuario, sin embargo ahora sucederá un error cuando registremos un nuevo usuario.

+ Opciones

	Editar	Copiar	Borrar	id	name	email	email_verified_at	password
<input type="checkbox"/>				1	admin	admin@admin.com	NULL	\$2y\$10\$36.H0i
<input type="checkbox"/>				2	freddy	freddy@freddy.com	NULL	\$2y\$10\$fh8IHf
<input type="checkbox"/>				3	aaa	aaa@aaa.com	NULL	\$2y\$10\$TN1O

El usuario fue registrado desde la página solo que olvide tomar captura así lo saque de la base de datos.

Entonces ahora recargamos la vista “User->index.blade.php” y veremos que existe un error:



localhost:8000/user

Bookmarks Marcadores https://www.movist... 01 Crear Proyecto ... SPREAD OUT (phra... Input group - Boots... http://localhost/ph... https://mundo.sput...

C:\laragon\www\RolesPermisos\

ErrorException
Undefined offset: 0 (View: C:\laragon\www\RolesPermisos\resources\views\user\index.blade.php)
http://localhost:8000/user

Stack trace Request App User Context Debug Share ↗

Illuminate\Foundation\Bootstrap\HandleExceptions::handleError
C:\laragon\www\RolesPermisos\vendor\laravel\framework\src\Illuminate\Support\Collection.php:1362 ↗

Para solucionar eso, realizaremos una validación. Para ello utilizaremos un helper llamado “@isset()”

```

@foreach($users as $user)
|  |  |  |  |
| --- | --- | --- | --- |
| {{ $user->id }} | {{ $user->name }} | {{ $user->email }} | @if(isset($user->roles[0]->name))     {{ $user->roles[0]->name }} @endif |

```

Básicamente la validación pregunta ¿existe un usuario con un rol? Si es así lo imprime de lo contrario lo deja en blanco.

#	Name	Email	Role(s)	Options
3	aaa	aaa@aaa.com		Show Delete
2	freddy	freddy@freddy.com	Registered user	Show Delete

Como vimos funcionó exitosamente. Ahora agregaremos un botón para ir a editar el usuario.. y

```

</td>

<td>
Show </a>
Edit </a>
<form action="{{ route\\('user.destroy', \\$user->id\\) }}" method="POST">
    @csrf
    @method\\('DELETE'\\)
    <button class="btn btn-danger"> Delete </button>
</form>

```

#	Name	Email	Role(s)	Options
3	aaa	aaa@aaa.com		Show Edit Delete

Luego iremos al controlador “UserController” a programar el método “edit()”:

```
public function edit(User $user)
{
    //obtener roles en orden ascendente.
    $roles = Role::orderBy('name')->get();

    return view('user.edit', compact('roles','user'));
}
```

Si deseamos ver para probar que roles existen podemos retornar la variable \$roles y en la vista “User->edit.blade.php” verianos un json con los roles.

Entonces ahora en la misma vista modificamos el formulario y lo dejamos con los campos necesarios, luego agregaremos un combobox para listar ahí los roles del usuario:

```
<div class="card-body">
    @include('custom.message')

    <form action="{{ route('user.update', $user->id) }}" method="POST">
        @csrf
        @method('PUT')
        <div class="container">
            <h3>Required Data</h3>
            <div class="form-group">
                <input type="text" class="form-control" value="{{ old('name', $user->name) }}" name="name" id="name" placeholder="Name" required>
            </div>

            <div class="form-group">
                <input type="email" class="form-control" value="{{ old('email', $user->email) }}" name="email" id="email" placeholder="Email" required>
            </div>

            <input type="submit" class="btn btn-primary" value="Save">
            <a class="btn btn-danger" href="{{ route('role.index') }}> Back </a>
        </div>
    </form>
</div>
```

Entonces agregamos el combobox que tendrá el role del usuario en primer plano, por el momento listamos todos los roles en el combobox:

```
<div class="form-group">
    <input type="email" class="form-control" value="{{ old('email', $user->email) }}" name="email" id="email" placeholder="Email" required>
</div>

<div class="form-group">
    <select name="roles" id="roles" class="form-control">
        @foreach($roles as $role)
        <option value="{{ $role->id }}> {{ $role->name }} </option>
        @endforeach
    </select>
</div>

<input type="submit" class="btn btn-primary" value="Save">
<a class="btn btn-danger" href="{{ route('role.index') }}> Back </a>
</div>
```

El resultado:

Edit User

Required Data

aaa

aaa@aaa.com

Admin

Save Back

Ahora necesitamos acceder a la relación “roles” que tiene el “User” para ello hacemos lo siguiente:

```
<div class="form-group">
  {$user->roles}
  <select name="roles" id="roles" class="form-control">
    @foreach($roles as $role)
    <option value="{{ $role->id}}> {{ $role->name}} </option>
    @endforeach
  </select>
</div>
```

Veremos que para el usuario “aaa” no trae nada:

aaa

aaa@aaa.com

Admin

Save Back

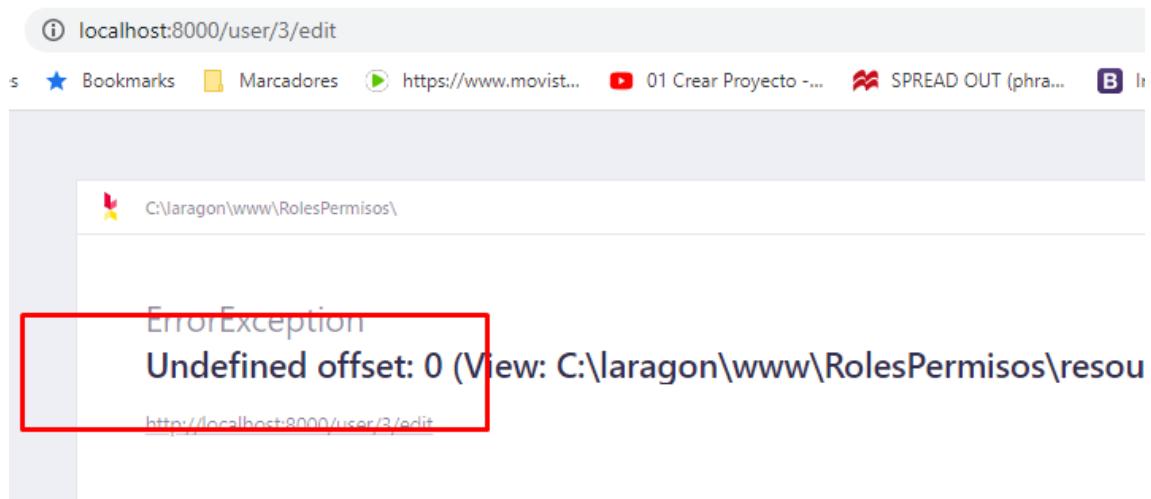
Primero por que el usuario registrado no tiene un rol asignado, ahora trataremos de acceder a la propiedad “name”:

```

<div class="form-group">
    {{ $user->roles[0]->name}}
    <select name="roles" id="roles" class="form-control">
        @foreach($roles as $role)
        <option value="{{ $role->id}}"> {{ $role->name}} </option>
        @endforeach
    </select>

```

Como el usuario no tiene role asignado nos traerá error:



Entonces con ese código realizaremos una validación con "@isset()":

```

<div class="form-group">
    <select name="roles" id="roles" class="form-control">
        @foreach($roles as $role)
            <option value="{{ $role->id}}">
                @isset($user->roles[0]->name)
                    @if($role->name == $user->roles[0]->name)
                        selected
                    @endif
                @endisset
            >
            {{ $role->name}} </option>
        @endforeach
    </select>
</div>

```

Si existe un role entonces le ponemos “selected” para los usuarios que no tiene role asignado por defecto se mostrarán los roles en forma “Ascendente” pero para los que tiene se pondrá primero el rol asignado.

Edit User

Required Data

Save
Back

Ahora copiaremos todo el contenido de la vista “User->edit.blade.php” a la vista “view.blade.php” y ahí desactivaremos los input con “disabled”:

```

UserController.php          edit.blade.php          view.blade.php X          RoleController.php
RolesPermisos > resources > views > user > view.blade.php

4  <div class="container">
5      <div class="row justify-content-center">
6          <div class="col-md-8">
7              <div class="card">
8                  <div class="card-header"><h2>Show User</h2> </div>
9
10             <div class="card-body">
11                 @include('custom.message')
12
13                 <form action="{{ route('user.update', $user->id) }}" method="POST">
14                     @csrf
15                     @method('PUT')
16                     <div class="container">
17                         <h3>Required Data</h3>
18                         <div class="form-group">
19                             <input disabled="" type="text" class="form-control" value="{{ old('name', $user->name) }}" name="name">
20                         </div>
21
22                         <div class="form-group">
23                             <input disabled="" type="email" class="form-control" value="{{ old('email', $user->email) }}" name="email">
24                         </div>
25
26                         <div class="form-group">
27                             <select disabled="" name="roles" id="roles" class="form-control">
28                                 @foreach($roles as $role)
29                                     <option value="{{ $role->id }}>
30                                         @isset($user->roles[0]->name)
31                                             @if($role->name == $user->roles[0]->name)
32                                                 selected
33                                             @endif
34                                         </option>
35                             </select>
36                         </div>
37                         <a class="btn btn-success" href="{{ route('user.edit', $user->id) }}"> Edit </a>
38
39                         <a class="btn btn-danger" href="{{ route('user.index') }}"> Back </a>
40                     </div>
41                 </form>

```

Luego iremos al método “show()” del controlador “UserController” y copiaremos lo mismo que hay en el método edit() al método show() y actualizamos la vista que deseamos cargar:

```
/*
public function show(User $user)
{
    //obtener roles en orden ascendente.
    $roles = Role::orderBy('name')->get();
    //return $roles;
    return view('user.view', compact('roles', 'user'));
}
```

Como veremos a continuación la vista “User->view.blade.php” se muestra correctamente:

The screenshot shows a user interface titled "Show User". Below it, a section titled "Required Data" contains three input fields: a text field with "aaa", an email field with "aaa@aaa.com", and a dropdown menu set to "Admin". At the bottom of the form are two buttons: a green "Edit" button and a red "Back" button.

Posteriormente nos dirigimos al método “update()” del controlador “UserController” y modificamos el método update():

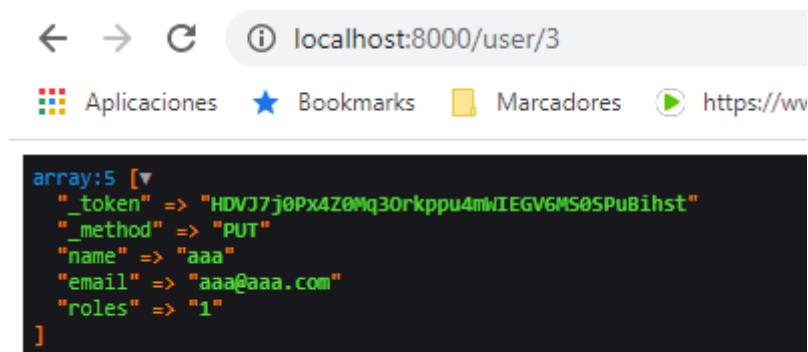
```

public function update(Request $request, User $user)
{
    //realizar validaciones
    $request->validate([
        'name' => 'required|max:50|unique:users,name,' . $user->id,
        'email' => 'required|max:50|unique:users,email,' . $user->id,
    ]);
    //Actualizar usuario:
    dd($request->all());

    $user->update($request->all());
    //actualizar relación:
    $user->roles()->sync( $request->get('roles') );
    //redireccionar:
    return redirect()->route('user.index')
        ->with('status_success','User updated successfully');
}

```

Vemos que se hacen las validaciones y por el momento estamos haciendo un debug con “dd()” para ver que todos los campos llegan correctamente:



```

array:5 [▼
  "_token" => "HDVJ7j0Px4Z0Mq30rkppu4mWIEGV6MS0SPuBihst"
  "_method" => "PUT"
  "name" => "aaa"
  "email" => "aaa@aaa.com"
  "roles" => "1"
]

```

Luego comentamos esa línea de código y permitiremos que se actualice y posteriormente redirigir.

Comentar //dd();

Finalmente actualizamos el role para el usuario creado llamado “aaa”:

#	Name	Email	Role(s)	Options
3	aaa	aaa@aaa.com	Registered user	Show Edit Delete
2	freddy	freddy@freddy.com	Registered user	Show Edit Delete

1 2 3

Finalmente nos resta programar el método `destroy()` del controlador “UserController”:

```
public function destroy(User $user)
{
    //Eliminar user
    $user->delete();
    //redireccionar:
    return redirect()->route('user.index')
        ->with('status_success', 'User successfully removed!');

}
```

En la vista “User->index.blade.php” queda programado el formulario que nos permitirá eliminar el usuario:

```
</td>

<td>
<a class="btn btn-info" href="{{ route('user.show', $user->id)}}> Show </a>
<a class="btn btn-success" href="{{ route('user.edit', $user->id)}}> Edit </a>
<form action="{{ route('user.destroy', $user->id)}}> method="POST">
    @csrf
    @method('DELETE')
    <button class="btn btn-danger"> Delete </button>
</form>
```

Probamos:

List of Users

User successfully removed!

#	Name	Email	Role(s)	Options
2	freddy	freddy@freddy.com	Registered user	<button>Show</button> <button>Edit</button> <button>Delete</button>
1	admin	admin@admin.com	Admin	<button>Show</button> <button>Edit</button> <button>Delete</button>

En efecto vemos que el usuario “aaa” se ha eliminado.

En la siguiente parte se aplicaran las políticas.

Fuente: <https://www.youtube.com/watch?v=K1b66zQZxmo>

❖ 13 como usar las políticas para blindar - Roles y Permisos Laravel 7

En esta parte vamos a ver varias cosas tales como:

- 1) ¿Qué son las políticas?
- 2) ¿Cómo utilizarlas para blindar nuestra aplicación?
- 3) ¿Cómo poder pasar variables adicionales a las políticas?
- 4) Diferencia entre gates y políticas.
- 5) Blindaremos completamente la aplicación.

Bien antes de iniciar nos aseguramos del que el role “registereduser” no tenga ningún permiso asignado y guardamos:

The screenshot shows a configuration interface for a policy. At the top, it says "Required Data" with three input fields containing "Registered user". Below that is a section titled "Full Access" with a "No" radio button selected. The final section is "Permissions List" containing a list of 9 permissions, each with a checkbox. The first checkbox is checked, and the others are empty.

Permission	Status
1 - List role (An user can list a role)	Selected
2 - Show role (An user can see a role)	Not Selected
3 - Create role (An user can create a role)	Not Selected
4 - Edit role (An user can Edit a role)	Not Selected
5 - Destroy role (An user can destroy a role)	Not Selected
6 - List user (An user can list a user)	Not Selected
7 - Show user (An user can see a user)	Not Selected
8 - Edit user (An user can Edit a user)	Not Selected
9 - Destroy user (An user can destroy a user)	Not Selected

Lo primero que vamos a hacer es crear un archivo de políticas llamado “UserPolicy” y lo vamos a asociar con el modelo “User” utilizando el comando “--model” entonces creamos la política de la siguiente manera:

```
#> php artisan make:policy UserPolicy --model=User
```

The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates "UserPolicy.php - Untitled (Workspace) - Visual Studio Code". The Explorer sidebar on the left shows the workspace structure: OPEN EDITORS, UNTITLED (WORKSPACE), RolesPermisos, app, Policies, and UserPolicy.php. A red arrow points to UserPolicy.php in the Policies folder. The main editor area contains the following PHP code:

```

1 <?php
2
3 namespace App\Policies;
4 Esto se agrega gracias a que usamos
5 use App\User; ← --model=User
6 use Illuminate\Auth\Access\HandlesAuthorization;
7
8 class UserPolicy
9 {
10     use HandlesAuthorization;
11
12     /**
13      * Determine whether the user can view any models.
14      *
15      * @param \App\User $user
16      * @return mixed
17      */
18     public function viewAny(User $user)
19     {
20         //
21     }
22
23     /**
24      * Determine whether the user can view the model.
25      *
26      * @param \App\User $user ←
27      * @param \App\User $model ←
28      * @return mixed
29      */
30     public function view(User $user, User $model)
31     {
32         //
33     }

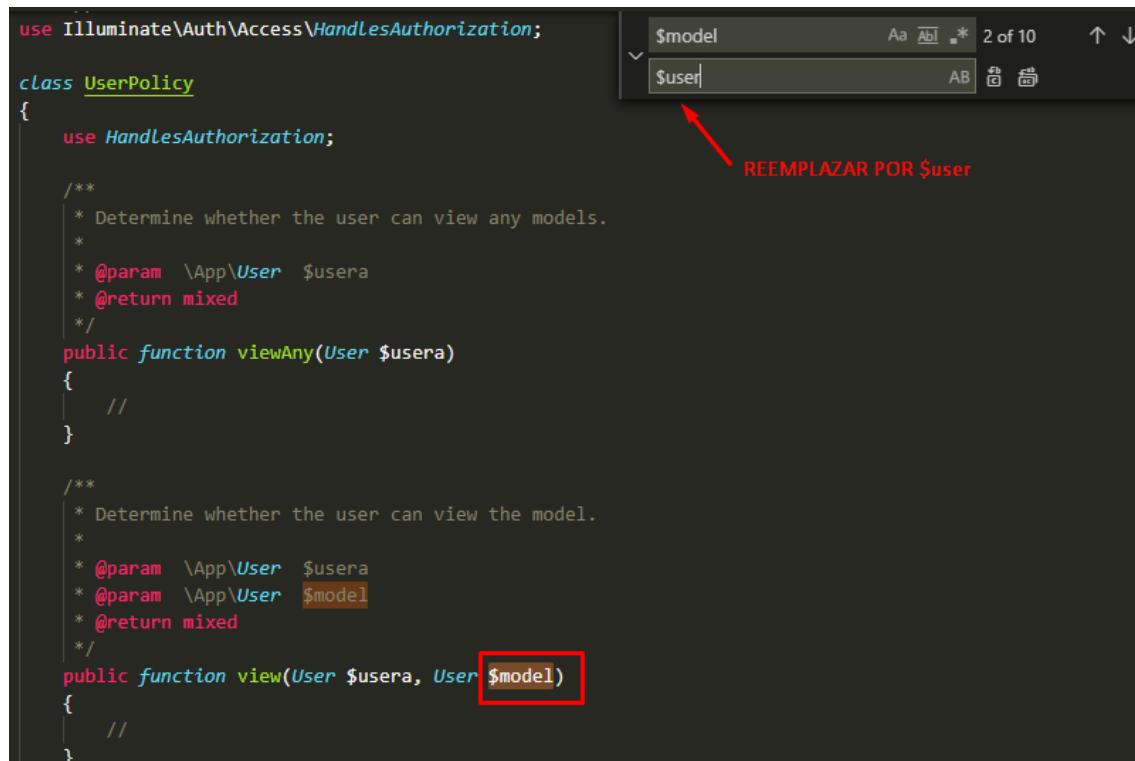
```

Así como creamos un controlador de recursos se crea un archivo de políticas con diversas funciones.

Lo siguiente que vamos a hacer es reemplazar todas las variables llamadas “\$user” por “\$usera” que será sinónimo de “user autenticado”:

The screenshot shows the Visual Studio Code interface with the UserPolicy.php file open. The code is identical to the one in the previous screenshot. A red arrow points to the first occurrence of \$user in the viewAny() method. Another red arrow points to the second occurrence of \$user in the view() method. A third red arrow points to the second occurrence of \$model in the view() method. In the status bar at the bottom right, there is a tooltip with the text: "Reemplazar todos los \$user por \$usera que significa: <user autenticado>".

Lo siguiente será también reemplazar la variable llamada “\$model” por “\$user”, le damos en reemplazar en todos.



```
use Illuminate\Auth\Access\HandlesAuthorization;

class UserPolicy
{
    use HandlesAuthorization;

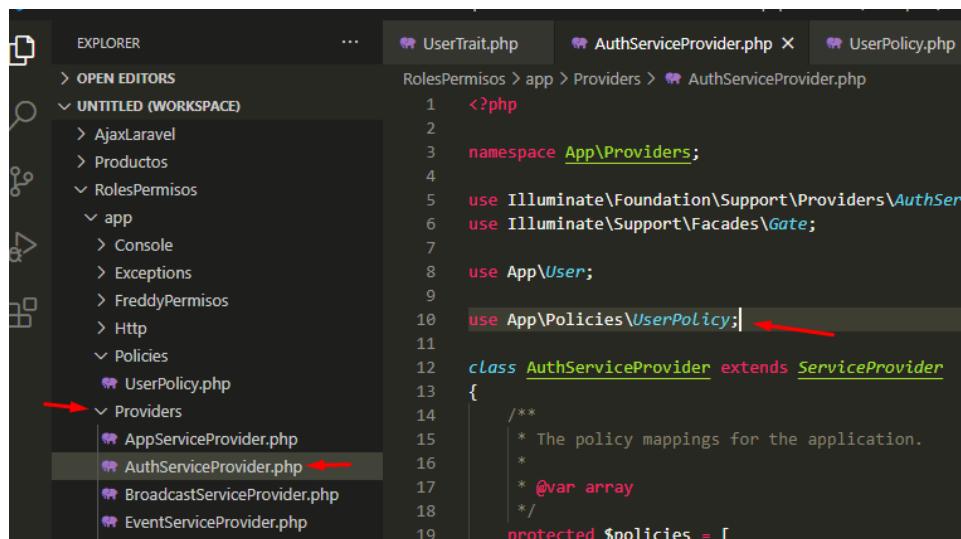
    /**
     * Determine whether the user can view any models.
     *
     * @param \App\User $usera
     * @return mixed
     */
    public function viewAny(User $usera)
    {
        //
    }

    /**
     * Determine whether the user can view the model.
     *
     * @param \App\User $usera
     * @param \App\User $model
     * @return mixed
     */
    public function view(User $usera, User $model)
    {
        //
    }
}
```

Tanto políticas como gates son usados para blindar la aplicación, pero una “política” se usar para un modelo en específico. En cambio el gate se puede usar en cualquier modelo de manera genérica.

Luego dentro del archivo Providers->AuthServiceProvider.php vamos a definir/enlazar/registrar nuestra política al modelo en específico:

Primero agregamos el name space “use App\Policies\UserPolicy\” dentro de AuthServiceProvider:



```
use Illuminate\Foundation\Support\Providers\AuthServiceProvider;
use Illuminate\Support\Facades\Gate;
use App\User;
use App\Policies\UserPolicy;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        //
```

Lo siguiente es vincular el modelo “User” con su política respectiva:

```
1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
6 use Illuminate\Support\Facades\Gate;
7
8 use App\User;
9
10 use App\Policies\UserPolicy; ←
11
12 class AuthServiceProvider extends ServiceProvider
13 {
14     /**
15      * The policy mappings for the application.
16      *
17      * @var array
18      */
19     protected $policies = [
20         // 'App\Model' => 'App\Policies\ModelPolicy'
21         User::class => UserPolicy::class, ←
22     ];
23
24     /**
25      * Register any authentication / authorization services.
26      *
```

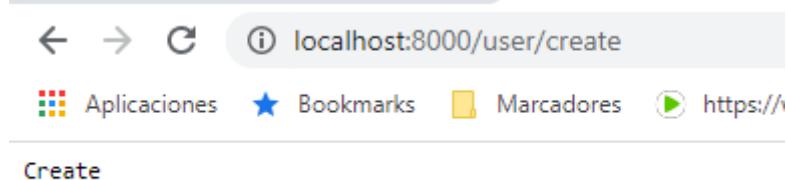
Ahora vamos a hacer unas pruebas: En el archivo web.php de las rutas, vamos a quitar del “Except” de la ruta “/user” el método “create” para que nos deje crear, de esa manera la ruta solo tiene como excepción el método “store”:

```
30     /*
31     Route::resource('/user', 'UserController', ['except'=>
32     |     'create', 'store']]);
33     */
34     Route::resource('/user', 'UserController', ['except'=>
35     |     'store']]);
36 
```

Ahora vamos al método create del controlador “UserController.php” y retornamos esto:

```
27     * Show the form for creating a new ...
28     *
29     * @return \Illuminate\Http\Response
30     */
31     public function create()
32     {
33         return 'Create'; ←
34     }
35
36 
```

Si ahora vamos al navegador y acceder a la ruta “/user/create” veremos que podemos acceder:



Esto funciona tanto con el usuario “admin” y el usuario “freddy”:

¿Cómo podemos utilizar la política para el método “create” del controlador “UserController”?

La política se aplica de la siguiente manera, en este caso le decimos que aplique para el método ‘create’ y le pasamos todo el modelo ‘User::class’ para otros métodos como el view tenemos que pasarle el objeto user “User \$user”..

```
public function create()
{
    //Aplicar política al método "create"
    $this->authorize('create', User::class);

    return 'Create';
}
```

UserController.php

A screenshot of a code editor showing the UserController.php file. A specific code block within the create() method is highlighted with a yellow background. The code inside the block is:

```
public function create()
{
    //Aplicar política al método "create"
    $this->authorize('create', User::class);

    return 'Create';
}
```

The file name 'UserController.php' is visible at the bottom right of the code block.

Además verificamos que el método “create” del controlador “UserController.php” es igual al método “create” que esta dentro de “UserPolicy”:

```
* @param \App\User $usera
* @return mixed
*/
public function create(User $usera)
{
    //
}
```

A screenshot of a code editor showing the UserPolicy.php file. A specific code block within the create() method is highlighted with a yellow background. The code inside the block is:

```
* @param \App\User $usera
* @return mixed
*/
public function create(User $usera)
{
    //
}
```

The file name 'UserPolicy.php' is visible at the top left of the code block. A red arrow points from the left margin to the line number 41, and another red arrow points from the right margin to the start of the method definition.

En ese método es donde definiremos la política para el método “create”.

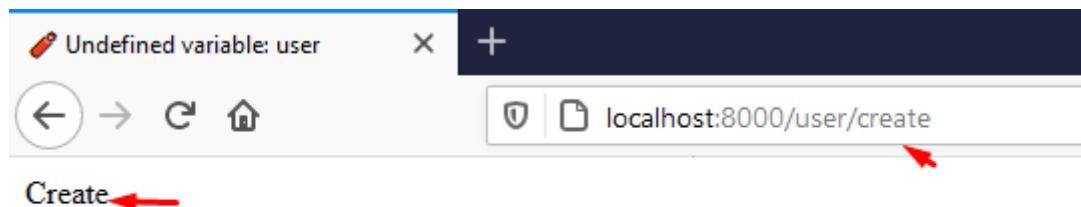
La política requiere de manera obligatoria para trabajar con cualquier modelo requerirá de user de manera obligatoria, como vemos en el método create le estamos pasando como parámetro el objeto “User \$user”, esto ya nos trae el usuario autentica y no es necesario verificar.

Si le decimos lo siguiente que verifique si el usuario tiene “id” mayor a cero nos dejara ver el método create de lo contrario nos dirá “acceso no autorizado”.

```
public function create(User $usera)
{
    return $usera->id > 0;
}
```

UserPolicy.php

Verificamos con un usuario autenticado;



Ahora nos deslogueamos y probamos a acceder a la ruta, pues nos dice “acceso no autorizado”

403 | This action is unauthorized.

El return que estamos haciendo tiene como condición que si el id del usuario es mayor 0 lo cual significa que si esta registrado por que todos los usuarios registrados tienen un id que comienza desde el 1.

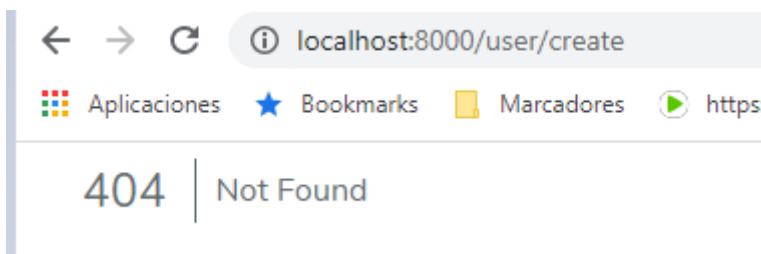
Entonces de esa manera verificamos que la política funciona como se esperaba, ahora vamos a dejar como estaba la ruta “/user” con el “excep” del método “create” lo agregamos otra vez:

```
33     */
34     Route::resource('/user', 'UserController', ['except'=>[
35         'create', 'store']]);
36 
```

Y comentamos el código del método “create” que están dentro del controlador “UserController.php”:

```
public function create()
{
    //Aplicar política al método "create"
    //$this->authorize('create', User::class);
    //return 'Create';
}
```

Ahora si intentamos acceder a la ruta “/user/create” nos dirá error 404:



Ahora por el momento vamos a trabajar con el método “Update” y “View” del archivo “UserPolicy” los demás métodos los podemos eliminar pero también mas adelante trabajaremos con el método “viewAny”.

Lo primero que haremos será trabajar con el método “view()” en este caso el este método aplicaría para el método “show()” del controlador “UserController.php”.

(*) Corrección antes si recordamos le dimos a reemplazar a la variable “\$user” por “\$usera” en el archivo “UserPolicy” sin embargo olvide reemplazar para la variable “\$model” por “\$user” asi le reemplazo en todos:

The screenshot shows a code editor with three tabs: "UserPolicy.php", "UserController.php", and "web.php". The "UserPolicy.php" tab is active. The code is as follows:

```
use HandlesAuthorization;
...
public function viewAny(User $usera)
{
}
...
public function view(User $usera, User $model)
```

A search result for the variable "\$model" is highlighted in the code editor's search results panel. A red arrow points from the text "reemplazo en todos:" to the "\$model" placeholder in the search results panel.

Ahora si nos fijamos en el método “view()” de “UserPolicy” veremos que este recibe dos parámetros:

The screenshot shows a code editor with three tabs: "UserController.php", "Middleware", "Kernel.php", and "Policies". The "Policies" folder is expanded, and "UserPolicy.php" is selected. The code is as follows:

```
public function view(User $usera, User $user)
```

Two red arrows point to the parameters "\$usera" and "\$user" in the "view" method signature.

Aquí la variable \$user hace referencia al objeto que le estaremos pasando desde el controlador, esto también puede ser por ejemplo un objeto llamado “\$producto”.. Ahora vamos al método “show()” del controlador “UserController” y aplicamos la política:

```

Auth
Controller.php
HomeController.php
RoleController.php
UserController.php
Middleware
Kernel.php
Policies
UserPolicy.php
Providers

```

```

55      */
56  public function show(User $user)
57  {
58      //aplicar politica:
59      $this->authorize('view', $user);
60
61      //obtener roles en orden ascendente.
62      $roles = Role::orderBy('name')->get();
63      //return $roles;
64      return view('user.view', compact('roles', 'user'));
65
66 }

```

Vemos que le estamos pasando como parámetro ‘view’ que significa el método que esta dentro de “UserPolicy” y el parámetro “\$user” que es el objeto, como dije antes también puede ser un objeto llamado \$producto.

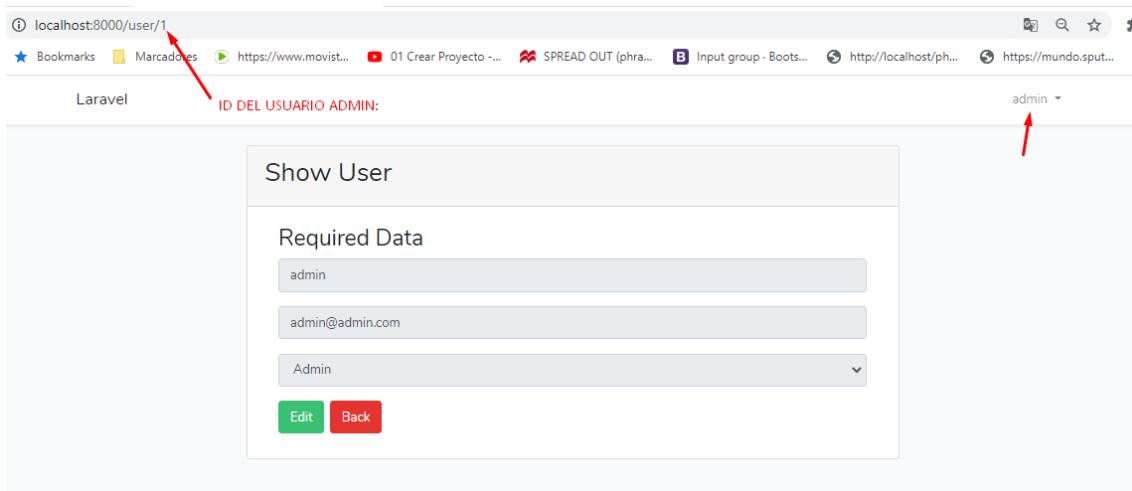
Entonces ahora para validar que un usuario solo pueda ver su usuario, debemos de hacer lo siguiente:

```

30  public function view(User $usera, User $user)
31  {
32      //verificar que el usuario id sea igual a su id
33      //si un usuario tiene id diferente no podra ver.
34      return $usera->id == $user->id;
35
36 }

```

A continuación, veremos que el usuario “admin” puede ver su propio usuario:



Que pasa ahora si el usuario “admin” que esta logueado intenta ver el usuario que tiene “id=2”, pues le dira acceso no autorizado:

List of Users

#	Name	Email	Role(s)	Options
2	freddy	freddy@freddy.com	Registered user	Show Edit Delete
1	admin	admin@admin.com	Admin	Show Edit Delete

localhost:8000/user/2

403 | This action is unauthorized.

De la misma manera hacemos en el método “update();” del archivo “UserPolicy”:

```

    > Middleware
    ↘ Kernel.php
    ↘ Policies
    ↗ UserPolicy.php
    > Providers
    ↘ User.php
    > bootstrap
    > config
    52
    53
    54
    55 * @param \App\User $user
    * @return mixed
    */
    public function update(User $usera, User $user)
    {
        //si un usuario tiene id diferente no podra ver.
        return $usera->id == $user->id;
    }
    /**

```

Y también en el controlador “UserController” en el método “edit()” aplicamos la política de update, para que solo el usuario logueado pueda actualizar su información, en este caso ni el administrador podrá modificar/editar:

```

    RoleController.php
    UserController.php -> UserController.php
    > Middleware
    & Kernel.php
    < Policies
    & UserPolicy.php
    > Providers
    & User.php
    > bootstrap
    > config
    > database
    > node_modules
    > ...

```

```

71 * @param int $id
72 * @return \Illuminate\Http\Response
73 */
74 public function edit(User $user)
75 {
76     $this->authorize('update', $user);
77
78     //obtener roles en orden ascendente.
79     $roles = Role::orderBy('name')->get();
80
81     //return $roles;
82     return view('user.edit', compact('roles', 'user'));
83 }
84

```

Como hemos podido ver cada usuario gracias a las políticas un usuario solo puede actualizar su información, en este caso la política se está aplicando de manera general, a pesar de tener un usuario “admin” este no puede ir y actualizarle la información al usuario “freddy”, entonces para solucionar eso vamos a ir a crear por lo menos otros dos permisos mas.

Nos vamos al archivo database->sedes->FreddyPermissionInfoSeeder.php y agregamos los dos siguientes permisos.

```

    > Providers
    & User.php
    > bootstrap
    > config
    < database
    > factories
    > migrations
    < seeds
    & DatabaseSeeder.php
    & FreddyPermissionInfoSeeder.php -> FreddyPermissionInfoSeeder.php
    & .gitignore
    > node_modules
    > public
    > resources

```

```

130 //permisos nuevos para politicas:
131 $permission = Permission::create([
132     'name' => 'Show own user',
133     'slug' => 'userown.show',
134     'description' => 'An user can see a user',
135 ]);
136
137 $permission_all[] = $permission->id;
138
139 $permission = Permission::create([
140     'name' => 'Edit own user',
141     'slug' => 'userown.edit',
142     'description' => 'An user can Edit own user',
143 ]);
144
145 /*

```

Ahora para evitar refrescar los seeder y volver a hacerlo otra vez lo que vamos a hacer es abrir en la consola y abrir el php artisan tinker para crear ahí mismo esos permisos y se agreguen a la base de datos:

#> php artisan tinker

```

1. RolesPermisos 2. RolesPermisos
C:\laragon\www\RolesPermisos λ php artisan tinker
Psy Shell v0.10.4 (PHP 7.2.19 - cli) by Justin Hileman
>>> |

```

Agregamos el namespace de la clase Permission:

```
C:\laragon\www\RolesPermisos
λ php artisan tinker
Psy Shell v0.10.4 (PHP 7.2.19 - cli) by Justin Hileman
>>> use App\FreddyPermisos\Models\Permission;
>>>
```

Luego copiamos el primer permiso agregado al seeder últimamente: (arriba hemos agregado dos, entonces ponemos uno por uno);

```
C:\laragon\www\RolesPermisos
λ php artisan tinker
Psy Shell v0.10.4 (PHP 7.2.19 - cli) by Justin Hileman
>>> use App\FreddyPermisos\Models\Permission;
>>>         $permission = Permission::create([
...     'name' => 'Show own user',
...     'slug' => 'userown.show',
...     'description' => 'An user can see a user',
... ]);
=> App\FreddyPermisos\Models\Permission {#4128
    name: "Show own user",
    slug: "userown.show",
    description: "An user can see a user",
    updated_at: "2020-09-12 22:26:12",
    created_at: "2020-09-12 22:26:12",
    id: 10,
}
>>> |
```

Solo le di copiar y pegar y luego enter, ahora nos resta copiar y pegar en la consola del tinker el ultimo permiso:

```
...
]); => App\FreddyPermisos\Models\Permission {#4118
    name: "Edit own user",
    slug: "userown.edit",
    description: "An user can Edit own user",
    updated_at: "2020-09-12 22:27:07",
    created_at: "2020-09-12 22:27:07",
    id: 11,
}
>>> |
```

Ahora si vemos los permissions:

Permissions List

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)
- 3 - Create role (*An user can create a role*)
- 4 - Edit role (*An user can Edit a role*)
- 5 - Destroy role (*An user can destroy a role*)
- 6 - List user (*An user can list a user*)
- 7 - Show user (*An user can see a user*)
- 8 - Edit user (*An user can Edit a user*)
- 9 - Destroy user (*An user can destroy a user*)
- 10 - Show own user (*An user can see own user*)
- 11 - Edit own user (*An user can Edit own user*)

Edit **Back**

En efecto vemos que se han agregado.

Ahora vamos a agregar un array de “slugs” a la política que se esta aplicando en el método “show()” del controlador UserController.php. En este caso le estamos diciendo que si en caso de tener el “full-access” que nos deje ver todos los registros, pero si no lo tengo y tengo “Show user” le decimos con el slug “user.show” que nos deje ver todos los usuarios, y si no tengo este y tengo el “Show own user” le decimos que permita verse solo su propio usuario para eso le pasamos el slug “userown.show”:

```

> FreddyPermisos          55   */
  ↘ Http                  56   public function show(User $user)
  ↘ Controllers           57   {
  ↘ Auth                  58       // aplicar politica:
  ↘ Controller.php        59       $this->authorize('view', [$user, ['user.show', 'userown.show']]);
  ↘ HomeController.php    60       // obtener roles en orden ascendente.
  ↘ RoleController.php    61       $roles = Role::orderBy('name')->get();
  ↘ UserController.php    62       //return $roles;
  ↘                         63       return view('user.view', compact('roles', 'user'));
  ↘ Middleware            64
  ↘ Kernel.php            65
  ↘ Policies              66
  ↘                         67

```

Con eso ya estamos listos para modificar el método “view()” de UserPolicy.php:

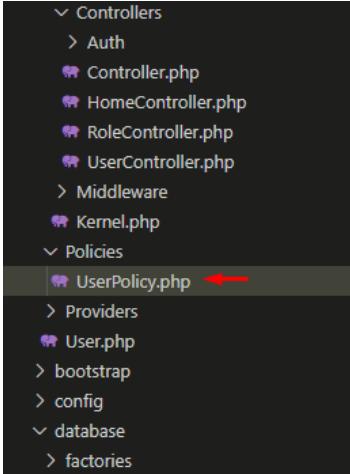
```

public function view(User $usera, User $user)
{
    //verificar que el usuario id sea igual a su id
    //si un usuario tiene id diferente no podra ver.
    return $usera->id == $user->id;
}

```

Ahora vamos a modificarlo para poder recibir el array de slugs..

El código agregado al método “view” le hemos agregado un parámetro en “null” \$perm=null para que no de problemas en caso de que el usuario no tenga ningun permiso asignado y en ese caso retornará falso. De acuerdo a los slug que le hemos pasado desde el método show, este hace una condición primero evaluando si el usuario tiene el permiso en el indice 0, si lo tiene retorna “true” o dejar ver en este caso es el slug “user.show” que significar que es del permiso que el usuario puede ver todos los usuarios, el segundo indice “1” hace referencia al slug “user.userown” que quiere decir que solo el usuario puede verse a si mismo pero no puede ver otros el de otros. Y por ultimo si no cumple con ninguno retorna false.



```

28     * @return mixed
29
30     */
31    public function view(User $usera, User $user, $perm=null)
32    {
33        if($usera->havePermission($perm[0])){
34            //permitir ver [user.show]
35            return true;
36        }else if($usera->havePermission($perm[1])){
37            //#[userown.show]=index=1
38            //permitir ver solo dueño
39            return $usera->id == $user->id;
40        }else{
41            return false;
42        }
43        //verificar que el usuario id sea igual a su id
44        //si un usuario tiene id diferente no podra ver.
45        //return $usera->id == $user->id;
46    }

```

Ahora si vamos al navegador a probar, el administrador tiene acceso a todo a los usuarios o mejor dicho puede ver la información tanto de el mismo con de otros usuarios.

Ahora como el usuario “freddy” tiene el role “registered user” por el momento el no tiene acceso a ver su información por que no tiene ningun role.

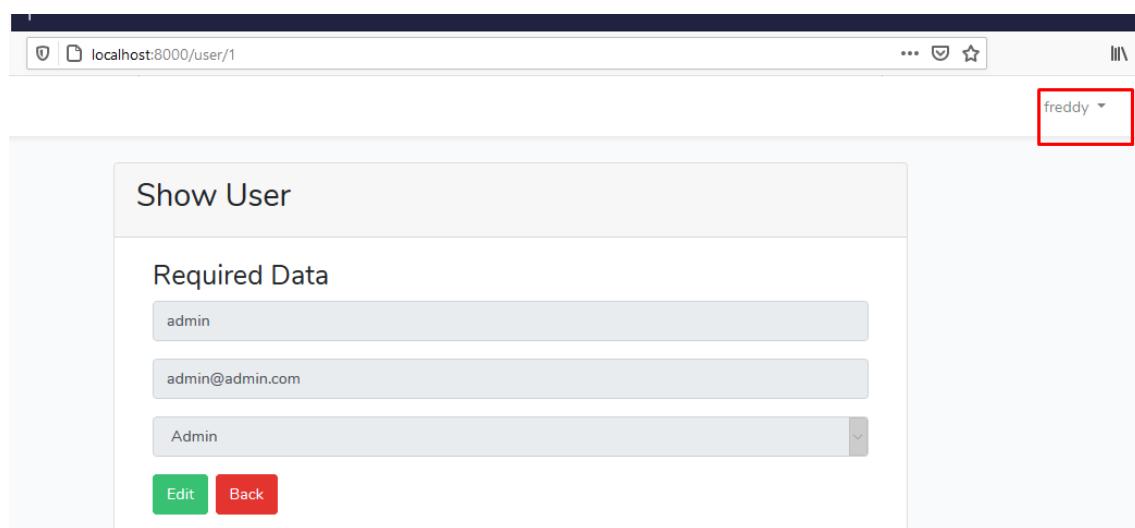
#	Name	Email	Role(s)	Options
3	freddy	freddy@freddy.com	Registered user	Show Edit Delete
1	admin	admin@admin.com	Admin	Show Edit Delete

Ahora le asignamos el role de full-access:

#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	yes	<button>Show</button> <button>Edit</button> <button>Delete</button>
1	Admin	admin	Administrator	no	<button>Show</button> <button>Edit</button>

Probamos desde el usuario freddy si puede ver la información de admin:

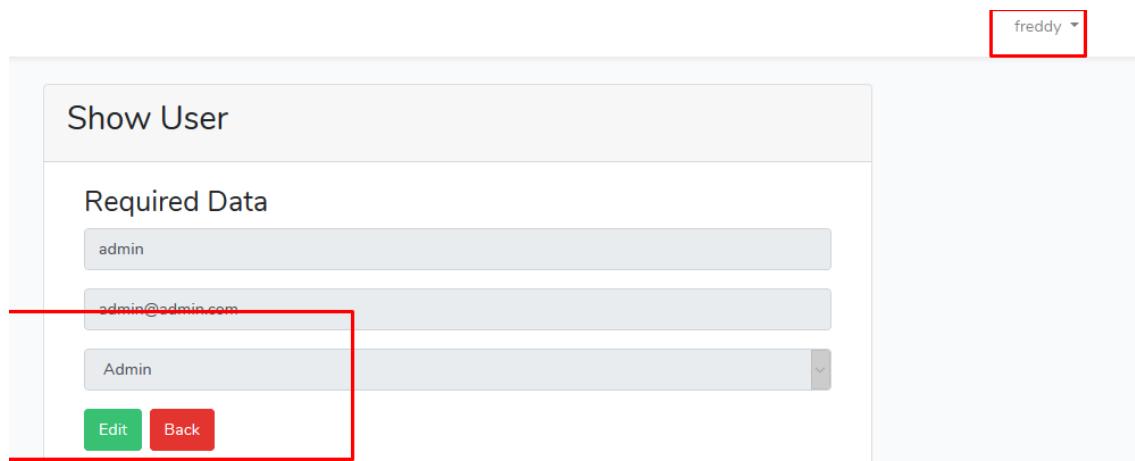
En efecto vemos que si puede:



The screenshot shows a 'Show User' page for the user 'admin'. The URL in the address bar is 'localhost:8000/user/1'. A red box highlights the user dropdown menu where 'freddy' is selected. The page displays the following required data for the user 'admin': Name: admin, Email: admin@admin.com, Role: Admin. There are 'Edit' and 'Back' buttons at the bottom.

Ahora le agregamos el permiso “show user” al role “registered user” que tiene asignado el usuario “freddy” y le quitamos el full-access:

Vemos que también puede ver la información del admin:



The screenshot shows a 'Show User' page for the user 'admin'. The URL in the address bar is 'localhost:8000/user/1'. A red box highlights the 'Edit' and 'Back' buttons at the bottom of the page. The page displays the following required data for the user 'admin': Name: admin, Email: admin@admin.com, Role: Admin. The 'Edit' and 'Back' buttons are also highlighted with a red box.

Ahora solo decimos que tenga el role “Show Own user”:

Full Access

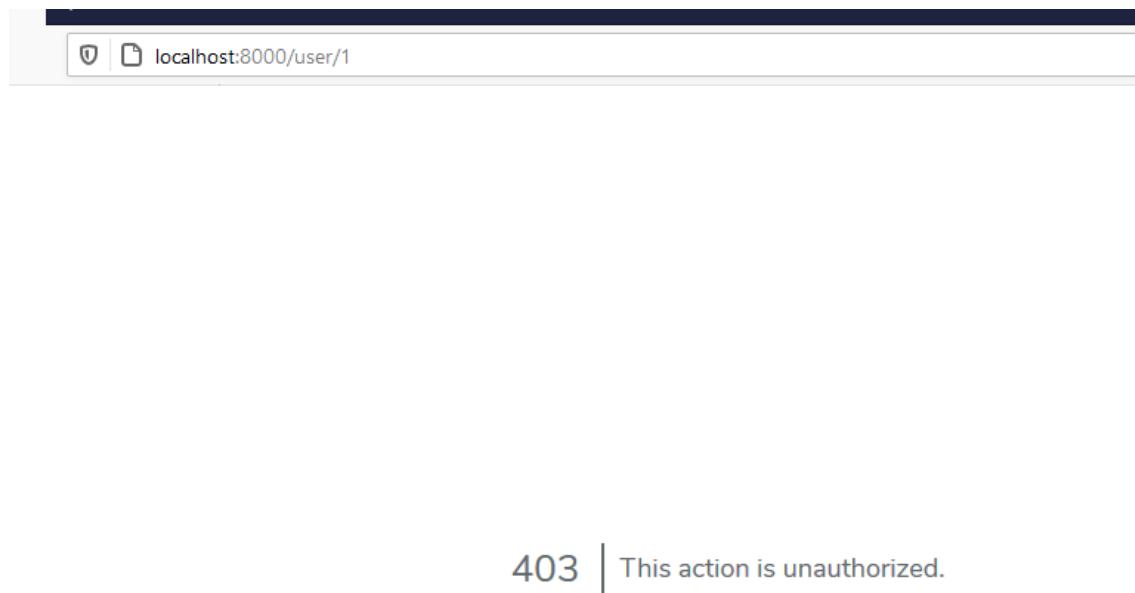
Yes No

Permissions List

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)
- 3 - Create role (*An user can create a role*)
- 4 - Edit role (*An user can Edit a role*)
- 5 - Destroy role (*An user can destroy a role*)
- 6 - List user (*An user can list a user*)
- 7 - Show user (*An user can see a user*)
- 8 - Edit user (*An user can Edit a user*)
- 9 - Destroy user (*An user can destroy a user*)
- 10 - Show own user (*An user can see own user*)
- 11 - Edit own user (*An user can Edit own user*)

[Blue](#) [Red](#)

Verificamos ahora si puede ver la información del admin:



The screenshot shows a browser window with a dark header bar. Below it, the address bar displays a shield icon and the URL "localhost:8000/user/1". The main content area of the browser shows a 403 error page with the text "403 | This action is unauthorized." in a light blue font.

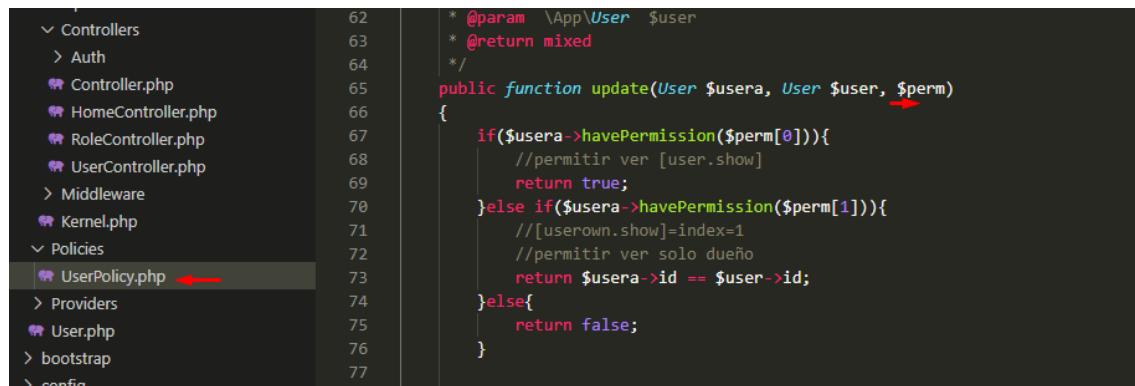
En efecto vemos que le dice “Acceso restringido” o código “403”.

Ahora verificamos que el usuario puede ver su propia información:



The screenshot shows a web application interface. At the top, there's a header bar with a shield icon, a 'localhost:8000/user/3' address bar, and some other icons. A red box highlights the 'freddy' dropdown in the top right corner. Below the header is a title 'Show User'. Underneath, a section titled 'Required Data' contains three input fields: one with 'freddy', another with 'freddy@freddy.com', and a dropdown menu set to 'Registered user'. At the bottom of this section are two buttons: a green 'Edit' button and a red 'Back' button.

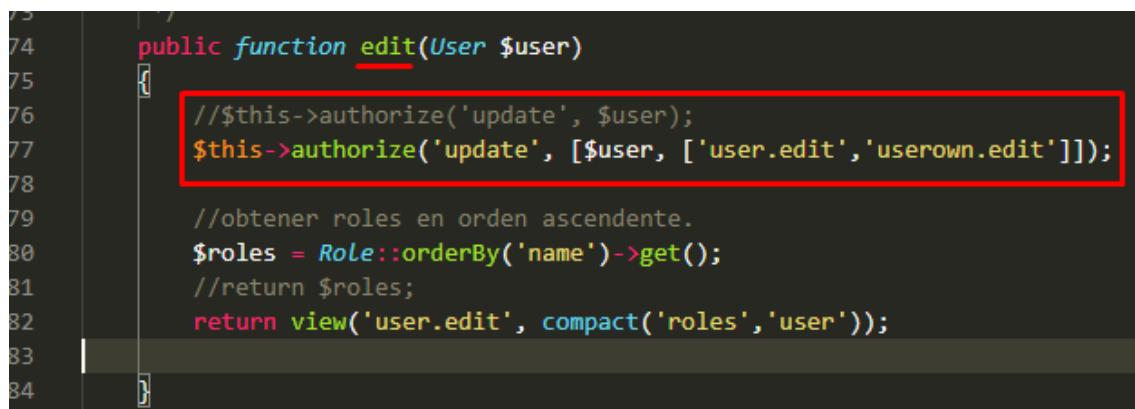
En efecto vemos que funciona. De la misma manera hacemos con el método “update()” de “UserPolicy”:



```
1 Controllers
2 > Auth
3   Controller.php
4   HomeController.php
5   RoleController.php
6   UserController.php
7 > Middleware
8   Kernel.php
9 Policies
10 <-- UserPolicy.php ←
11 Providers
12 User.php
13 bootstrap
14 config
```

62 * @param \App\User \$user
63 * @return mixed
64 */
65 public function update(User \$usera, User \$user, \$perm)
66 {
67 if(\$usera->havePermission(\$perm[0])){
68 //permitir ver [user.show]
69 return true;
70 }else if(\$usera->havePermission(\$perm[1])){
71 //#[userown.show]=index=1
72 //permitir ver solo dueño
73 return \$usera->id == \$user->id;
74 }else{
75 return false;
76 }
77 }

De la misma manera hacemos con el método “edit()” de UserController, actualizamos el método y aplicamos la política. De esta manera un usuario puede editar solo su información si tiene el role “Edit own user” o de todos si tiene el role “Edit User”:



```
73
74     public function edit(User $user)
75     {
76         //[$this->authorize('update', $user);
77         $this->authorize('update', [$user, ['user.edit','userown.edit']]);
78
79         //obtener roles en orden ascendente.
80         $roles = Role::orderBy('name')->get();
81         //return $roles;
82         return view('user.edit', compact('roles','user'));
83     }
84 }
```

Pero si no tiene ningún permiso asignado y su role tiene full-access puede acceder a todo.

Probar!

Lo que vamos hacer ahora es blindar toda la aplicación:

Blindamos el método index de “UserController”:

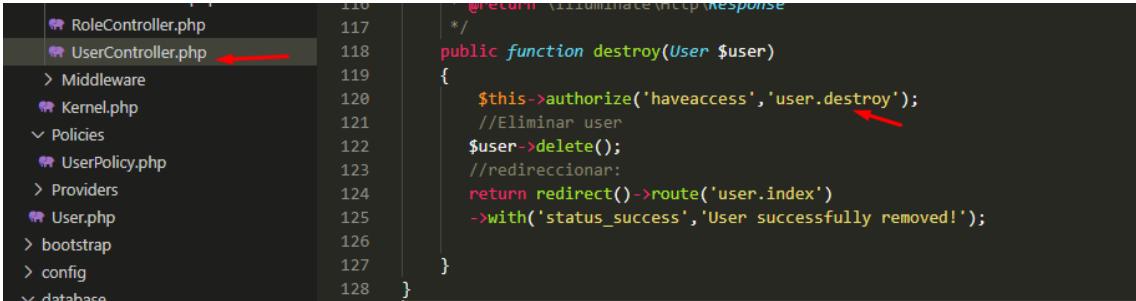


```
public function index()
{
    $this->authorize('haveaccess', 'user.index');

    $users = User::with('roles')->orderBy('id', 'Desc')->paginate(2);
    //return $users;

    return view('user.index', compact('users'));
}                                         UserController.php
```

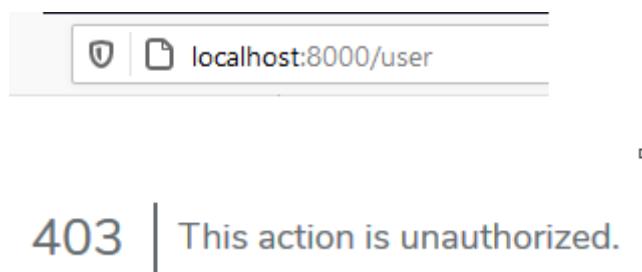
De la misma manera con el método “destroy()” en “UserController.php”:



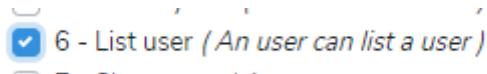
```
RoleController.php      116     return Illuminate\Http\Response
UserController.php ← 117     /*
UserController.php      118     public function destroy(User $user)
Middleware          119     {
Kernel.php           120         $this->authorize('haveaccess', 'user.destroy');
Policies            121         //Eliminar user
UserPolicy.php       122         $user->delete();
Providers           123         //redireccionar:
User.php             124         return redirect()->route('user.index')
bootstrap           125         ->with('status_success', 'User successfully removed!');
config              126
database            127     }
128 }                         129 }
```

En el método “update()” de “UserController.php” no lo blindamos por que va a depender del método “edit()” del mismo controlador, si tiene la opción/permiso de editar su propio usuario entonces si le concedería acceso.. por lo tanto todo depende de que si tiene o no permiso a “edit()”.

Ahora si intentamos que el usuario “freddy” el cual tiene el role “registered user” y este rol no tiene por ahora el permiso de “list user” veremos que no podrá listar los usuarios:



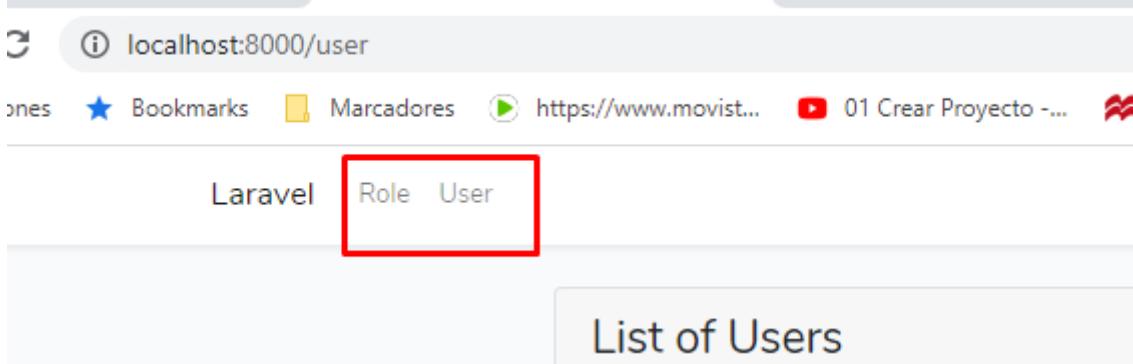
Para eso tenemos que asignar el permiso “List User” al role “registered user”:



De esa manera los usuarios que no sean admin podrán listar los usuarios.

Lo que vamos a hacer ahora es trabar con el “blade”.

Lo primero que vamos a hacer es agregar dos opciones en la barra de Menu de laravel para tener acceso a “User” y “Role”:



Así nos evitaremos estar accediendo manualmente a las dos rutas. Para hacer eso tenemos que ir a la carpeta “Resources->Views->layouts>**app.blade.php**” abrimos ese archivo y agregamos los dos ítems dentro de la etiqueta `` que tiene la clase “navbar-nav mr-auto”

```
<!-- Left Side Of Navbar -->
<ul class="navbar-nav mr-auto">
    <li class="nav-item">
        <a href="{{route('role.index')}}" class="nav-link">Role</a>
    </li>
    <li class="nav-item">
        <a href="{{route('user.index')}}" class="nav-link">User</a>
    </li>
</ul>
```

The code block shows the `app.blade.php` file with two new menu items, "Role" and "User", added to the `` list. These items are highlighted with a red box.

¿Pero como blindamos a nivel del **blade** nuestra aplicación? Ya que al momento el usuario puede ver la opción role y acceder..

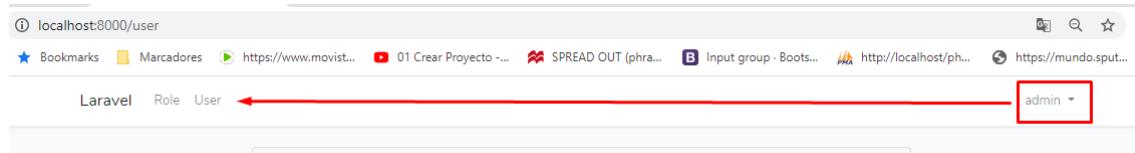
Lo que vamos a hacer es utilizar un helper llamado “@can” que sirve tanto para los Gates y Políticas.

Agregamos el “@can” dentro del menú y básicamente le pasamos el nombre de la función ‘haveaccess’ que hemos utilizado en los Gate y Políticas más el slug del permiso. En este caso el slug ‘role.index’ hace referencia a la página de listado de roles y de la misma manera ‘user.index’ para ir a listar los usuarios...

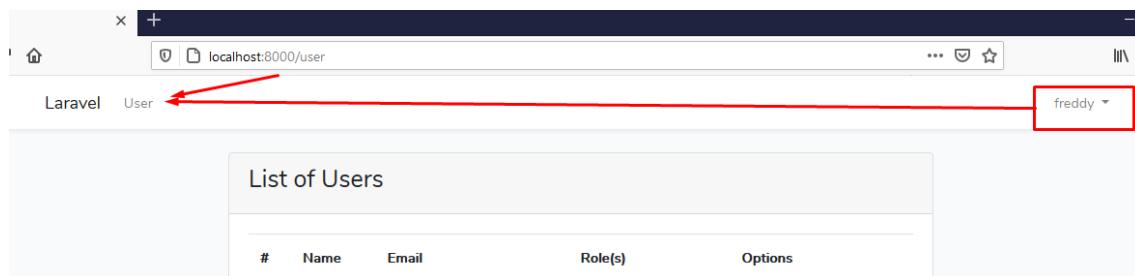


```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <!-- Left Side Of Navbar -->
    <ul class="navbar-nav mr-auto">
        @can('haveaccess', 'role.index') ←
            <li class="nav-item">
                <a href="{{route('role.index')}}" class="nav-link">Role</a>
            </li>
        @endcan
        @can('haveaccess', 'user.index') ←
            <li class="nav-item">
                <a href="{{route('user.index')}}" class="nav-link">User</a>
            </li>
        </ul>
    @endcan
```

Ahora desde el navegador en donde esta logueado el usuario ‘admin’ vemos que tiene acceso o puede ver los dos menus por que tiene ‘full-access’.



Si embargo desde el otro navegador logueado con el otro usuario llamado ‘freddy’ este solo tiene acceso a una opción:



Así, si el usuario con su role ‘registered user’ si tiene asignado el permiso ‘show role’ podrá ver esa opción de lo contrario no.

Ahora lo que vamos a hacer es trabajar en el ‘index’ del role.

Abrimos el archivo “Resources->views->role->index.blade.php y blindamos el botón crear:

List of Roles					
Role updated successfully					
#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	no	Show Edit Delete
1	Admin	admin	Administrator	yes	Show Edit Delete

Si el usuario con el role ‘registered user’ no tiene el permiso de ‘Create role’ obviamente el botón no se le mostrará.

```
✓ role
  ↘ create.blade.php
  ↘ edit.blade.php
  ↗ index.blade.php
  ↘ view.blade.php
> user
  ↘ home.blade.php
  ↘ welcome.blade.php
✓ routes
  ↘ api.php
  ↘ channels.php
  ↘ comandosusados.php
      8
      9
      10
      11
      12
      13
      14
      15
      16
      17
      18
      19
      20
      21
      <div class="card-header"> <h3>List of Roles </h3> </div>
      <div class="card-body">
        @include('custom.message')
        @can('haveaccess', 'role.create')
          <a href="{{ route('role.create') }}" class="btn btn-primary float-right">
            Create
          </a>
        @endcan
      <br><br>
```

A continuación al usuario 'freddy' con su role 'registered user' le dimos el permiso de 'Show Role' pero no tiene el permiso de 'create role' por lo tanto no se le visualiza el botón:

#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	no	Show Edit Delete
1	Admin	admin	Administrator	yes	Show Edit Delete

De la misma manera hacemos con el resto de botones dentro del formulario:

```

> config           36
> database        37
> node_modules   38
> public          39
✓ resources       40
  > js             41
  > lang           42
  > sass            43
  ✓ views          44
    > auth           45
    > custom          46
    ✓ layouts         47
      app.blade.php 48
      <role>
        & create.blade.php 49
        & edit.blade.php 50
        index.blade.php 51
        view.blade.php 52
      </role>
    > user            53
    & home.blade.php 54
    & welcome.blade.php 55
  </views>
> user            56
> home            57
> welcome         58
> routes          59
> middleware     60

```

```

@foreach($roles as $role)
  <tr>
    <th scope="row">{{ $role->id }}</th>
    <td>{{ $role->name }}</td>
    <td>{{ $role->slug }}</td>
    <td>{{ $role->description }}</td>
    <td>{{ $role['full-access'] }}</td>
    <td>
      @can('haveaccess', 'role.show')
      <a class="btn btn-info" href="{{ route('role.show', $role->id)}}> Show </a>
      @endcan
      @can('haveaccess', 'role.edit')
      <a class="btn btn-success" href="{{ route('role.edit', $role->id)}}> Edit </a>
      @endcan
      @can('haveaccess', 'role.destroy')
      <form action="{{ route('role.destroy', $role->id)}}> Delete </button>
      @method('DELETE')
      </form>
      @endcan
    </td>
  </tr>

```

Ahora si vamos y actualizamos la página de la ruta '/role' veremos que el usuario freddy con el role 'registered user' que tiene el permiso de 'list role' no podrá ver los botones por que no tiene los permisos.

List of Roles

#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	no	
1	Admin	admin	Administrator	yes	

Ahora al role 'registered user' le damos permiso de 'show role':

- 1 - List role (An user can list a role)
- 2 - Show role (An user can see a role) 
- 3 - Create role (An user can create a role)

Vemos a continuación que tiene acceso o puede ver el botón.

#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	no	<button>Show</button>
1	Admin	admin	Administrator	yes	<button>Show</button>

Usuario: Freddy

Le damos el permiso de 'Edit role':

- 1 - List role (An user can list a role)
- 2 - Show role (An user can see a role)
- 3 - Create role (An user can create a role)
- 4 - Edit role (An user can Edit a role) 

#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	no	<button>Show</button> <button>Edit</button>
1	Admin	admin	Administrator	yes	<button>Show</button> <button>Edit</button>

Vemos que tiene ahora la opción para eliminar.

Ahora le damos la opción ‘Destroy Role’:

Permissions List

- 1 - List role (An user can list a role)
- 2 - Show role (An user can see a role)
- 3 - Create role (An user can create a role)
- 4 - Edit role (An user can Edit a role)
- 5 - Destroy role (An user can destroy a role) 

Vemos que puede ver la opción “delete”

#	Name	Slug	Description	Full-access	Options
2	Registered user	registereduser	Registered user	no	Show Edit Delete
1	Admin	admin	Administrator	yes	Show Edit Delete

Ahora si el usuario intenta acceder directamente a las rutas no podrá tener acceso por que desde el backend también le hemos blindado.

Lo que vamos hacer ahora es también blindar a nivel de “blade” la vista ‘user.blade.php’:

Para continuar le damos permiso de ‘list user’ al role ‘registered user’:

- 1 - List role (An user can list a role)
- 2 - Show role (An user can see a role)
- 3 - Create role (An user can create a role)
- 4 - Edit role (An user can Edit a role)
- 5 - Destroy role (An user can destroy a role)
- 6 - List user (An user can list a user) 
- 7 - Show user (An user can see a user)

Como podemos apreciar el usuario puede listar/ver los usuarios.

The screenshot shows a 'List of Users' table. There are two rows:

#	Name	Email	Role(s)	Options
3	freddy	freddy@freddy.com	Registered user	Show Edit Delete
1	admin	admin@admin.com	Admin	Show Edit Delete

Ahora para blindar los botones que hay en esta vista vamos a hacer uso de las funciones de las políticas, antes usamos ‘haveaccess’ pero ahoraaremos algo parecido a lo que hicimos dentro de los métodos/funciones del controlador ‘UserController.php’ cuando aplicamos políticas.

Añadiremos será utilizar las funciones ‘view’ y ‘update’ que son parte de las políticas del “UserPolicy” y para ocultar el botón “delete” usaremos la función ‘haveaccess’:

Bien entonces editamos el archivo “Resources->views->user->**index.blade.php**”

```
</td>
<td>
    @can('view', [$user, ['user.show', 'userown.show']])
        <a class="btn btn-info" href="{{ route('user.show', $user->id) }}"> Show </a>
    @endcan
    @can('update', [$user, ['user.edit', 'userown.edit']])
        <a class="btn btn-success" href="{{ route('user.edit', $user->id) }}"> Edit </a>
    @endcan

    @can('haveaccess', 'user.destroy')
        <form action="{{ route('user.destroy', $user->id) }}" method="POST">
            @csrf
            @method('DELETE')
            <button class="btn btn-danger"> Delete </button>
        </form>
    @endcan
</td>
</tr>
@endforeach
```

De esta forma es entonces como podemos blindar nuestra aplicación. Lo que resta es probar asignadole varios permisos al role 'registered user'.

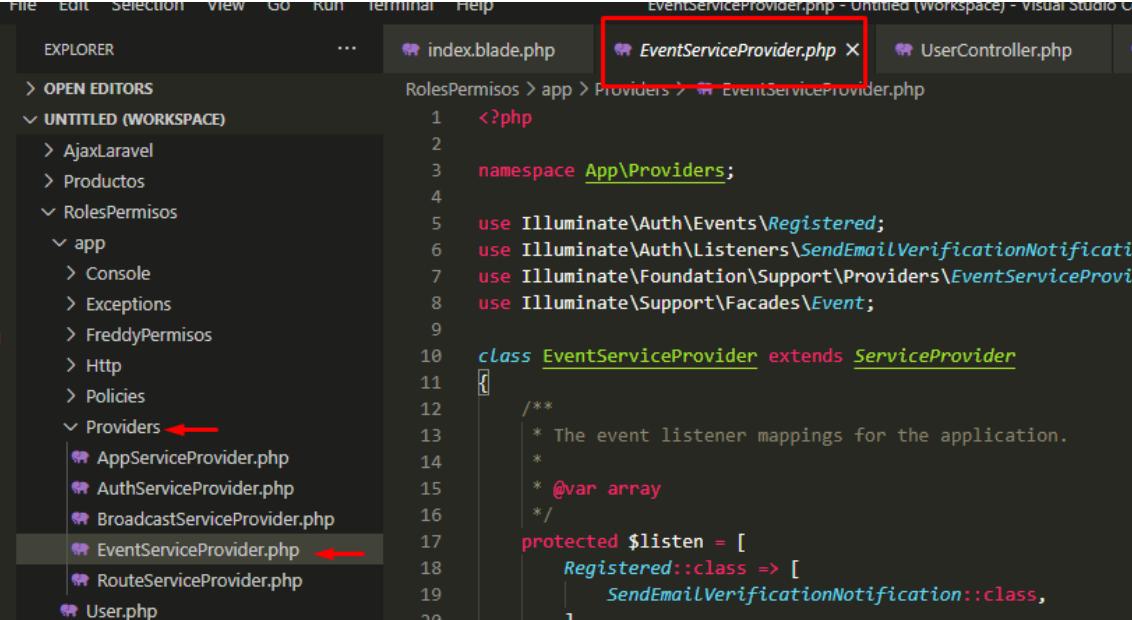
List of Users				
#	Name	Email	Role(s)	Options
3	freddy	freddy@freddy.com	Registered user	<button>Show</button> <button>Edit</button>
1	admin	admin@admin.com	Admin	<button>Show</button>

Fuente: <https://www.youtube.com/watch?v=6nCbiljC07U>

❖ Bonus - agregar un rol al registrarse un usuario - Roles y Permisos Laravel 7

El objetivo de esta última parte es que cuando un usuario se registre se le asigne un 'role' automáticamente.

Para ello nos vamos al archivo 'EventServiceProvider.php' que se encuentra dentro de 'app->providers'.



```
File Edit Selection View Go Run Terminal Help
EXPLORER ...
OPEN EDITORS ...
UNTITLED (WORKSPACE)
> AjaxLaravel
> Productos
< RolesPermisos
  < app
    > Console
    > Exceptions
    > FreddyPermisos
    > Http
    > Policies
  < Providers
    AppServiceProvider.php
    AuthServiceProvider.php
    BroadcastServiceProvider.php
    EventServiceProvider.php
    RouteServiceProvider.php
    User.php
RolesPermisos > app > Providers > EventServiceProvider.php
1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Auth\Events\Registered;
6 use Illuminate\Auth\Listeners\SendEmailVerificationNotification;
7 use Illuminate\Foundation\Support\Providers\EventServiceProvider;
8 use Illuminate\Support\Facades\Event;
9
10 class EventServiceProvider extends ServiceProvider
11 {
12     /**
13      * The event listener mappings for the application.
14      *
15      * @var array
16     */
17     protected $listen = [
18         Registered::class => [
19             SendEmailVerificationNotification::class,
20         ],
21     ];
22 }
```

Aquí, en esta clase es donde vamos a definir que 'archivo' a ejecutar cuando se produzca el evento 'Registered'. Laravel maneja varios eventos 'Registered, logout' etc.. Basicamente en este archivo le diremos a laravel que clase ejecutar cuando se produzca el evento 'registered' para ello agregamos lo siguiente dentro del array '\$listen':

```

1 <?php
2
3 namespace App\Providers;
4
5 use Illuminate\Auth\Events\Registered;
6 use Illuminate\Auth\Listeners\SendEmailVerificationNotification;
7 use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServiceProvider;
8 use Illuminate\Support\Facades\Event;
9
10 class EventServiceProvider extends ServiceProvider
11 {
12     /**
13      * The event listener mappings for the application.
14      *
15      * @var array
16      */
17     protected $listen = [
18         Registered::class => [
19             SendEmailVerificationNotification::class,
20         ],
21         Registered::class => [
22             'App\Listeners\RegisteredEvent',
23         ],
24     ];
25 }

```

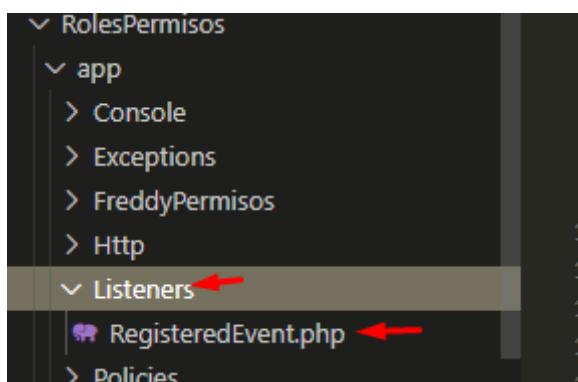
Clase a ejecutar cuando se produzca evento 'Registered'

Lo que resta ahora es decirle a laravel que genere automáticamente esa clase llamada 'RegisteredEvent' que estará dentro de la carpeta Listeners, entonces para hacer eso hacemos lo siguiente ejecutando el siguiente comando:

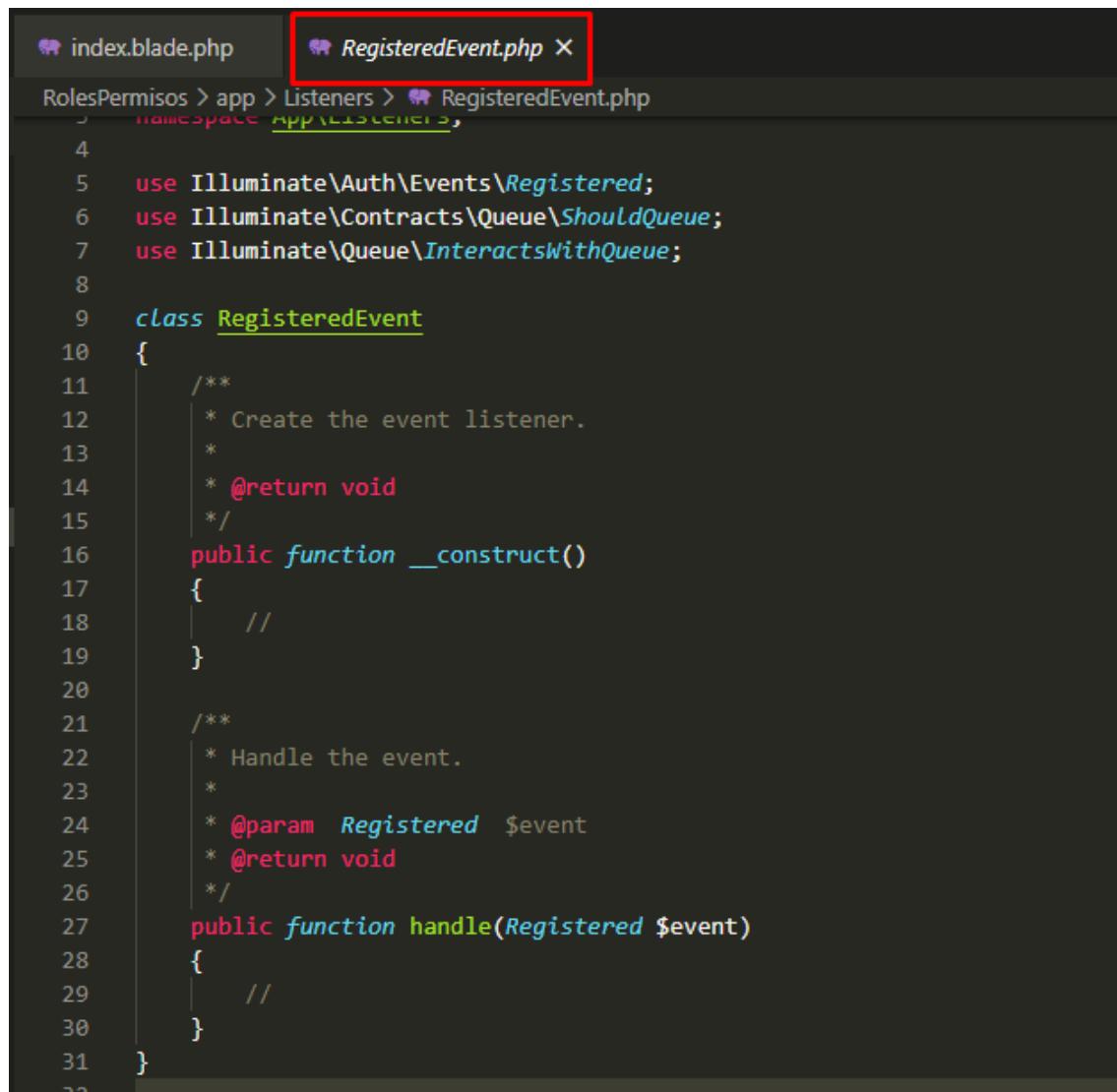
```
#> php artisan event:generate
```

```
C:\laragon\www\RolesPermisos
λ php artisan event:generate
Events and listeners generated successfully!
```

Vemos ahora en el IDE Visual Studio Code que se ha generado la carpeta 'Listeners' y la clase 'RegisteredEvent':



El contenido inicial de la clase es el siguiente:



```
index.blade.php RegisteredEvent.php X
RolesPermisos > app > Listeners > RegisteredEvent.php
namespace App\Listeners;
4
5 use Illuminate\Auth\Events\Registered;
6 use Illuminate\Contracts\Queue\ShouldQueue;
7 use Illuminate\Queue\InteractsWithQueue;
8
9 class RegisteredEvent
10 {
11     /**
12      * Create the event listener.
13      *
14      * @return void
15      */
16     public function __construct()
17     {
18         //
19     }
20
21     /**
22      * Handle the event.
23      *
24      * @param Registered $event
25      * @return void
26      */
27     public function handle(Registered $event)
28     {
29         //
30     }
31 }
32
```

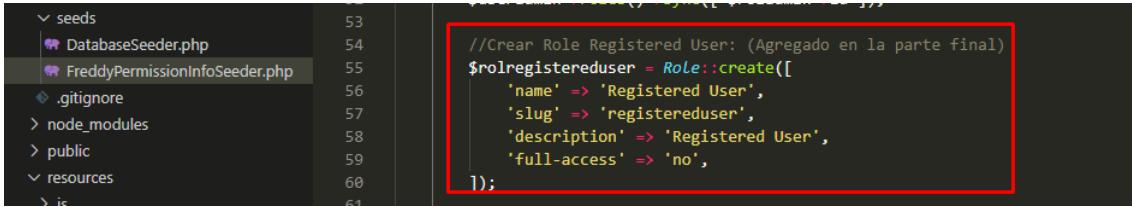
Donde pondremos toda nuestra lógica será dentro de la función 'handle()'.

Ahora para probar hacemos un 'dd()' de la variable/objeto '\$event':

```
public function handle(Registered $event)
{
    dd($event);
}
```

Ahora vamos a ir al archivo “database->seeds->FreddyPermissionInfoSeeder” y agregamos un role más:

En este caso estamos agregando el role llamado ‘Registered User’; Este role fue agregado debajo del role ‘admin’..

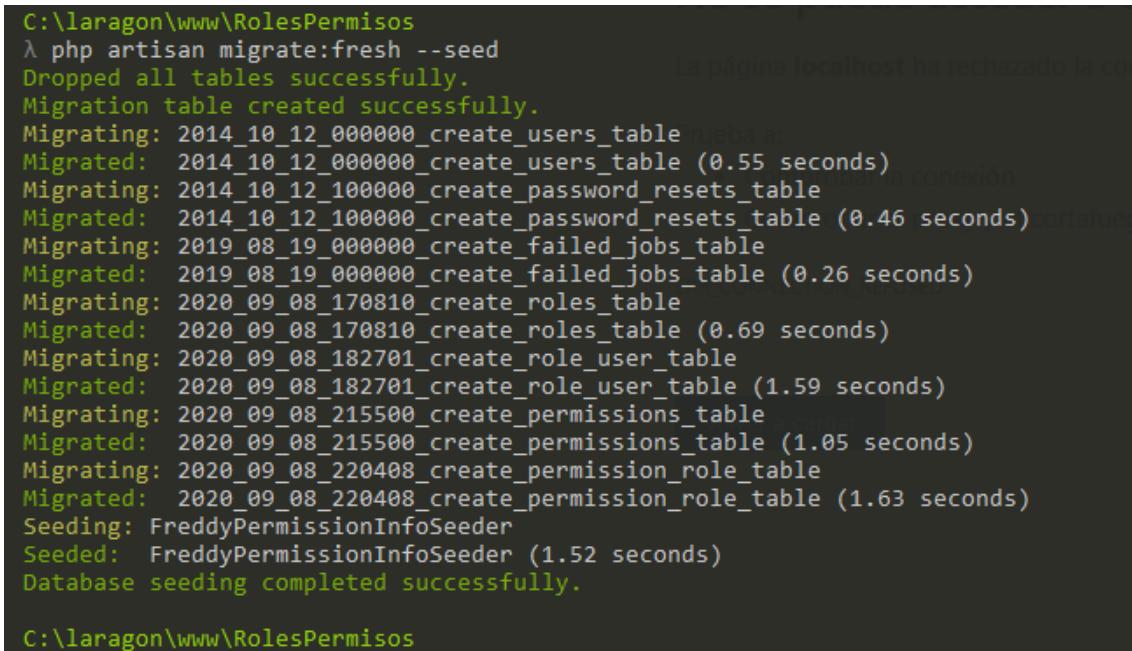


```
53
54
55 //Crear Role Registered User: (Agregado en la parte final)
56 $rolregistereduser = Role::create([
57     'name' => 'Registered User',
58     'slug' => 'registereduser',
59     'description' => 'Registered User',
60     'full-access' => 'no',
61 ]);
```

Lo que vamos a hacer ahora es volver a ejecutar las migraciones y el seeder que crea el usuario ‘admin’, role ‘admin’ y ‘registered user’:

Para hacer esto ejecutamos el siguiente comando:

```
#> php artisan migrate:fresh --seed
```



```
C:\laragon\www\RolesPermisos
λ php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.55 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.46 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.26 seconds)
Migrating: 2020_09_08_170810_create_roles_table
Migrated: 2020_09_08_170810_create_roles_table (0.69 seconds)
Migrating: 2020_09_08_182701_create_role_user_table
Migrated: 2020_09_08_182701_create_role_user_table (1.59 seconds)
Migrating: 2020_09_08_215500_create_permissions_table
Migrated: 2020_09_08_215500_create_permissions_table (1.05 seconds)
Migrating: 2020_09_08_220408_create_permission_role_table
Migrated: 2020_09_08_220408_create_permission_role_table (1.63 seconds)
Seeding: FreddyPermissionInfoSeeder
Seeded: FreddyPermissionInfoSeeder (1.52 seconds)
Database seeding completed successfully.

C:\laragon\www\RolesPermisos
```

En efecto vemos que se han creado las migraciones y el ‘Seeder’ se ha ejecutado.

Si verificamos en la base de datos, veremos que los roles se han creado exitosamente:



+ Opciones	← →		id	name	slug	description	full-access	created_at	updated_at
<input type="checkbox"/>		Editar	1	Admin	admin	Administrator	yes	2020-09-13 17:50:59	2020-09-13 17:50:59
<input type="checkbox"/>		Editar	2	Registered User	registereduser	Registered User	no	2020-09-13 17:50:59	2020-09-13 17:50:59

Además vemos en la tabla ‘role_user’ que el usuario ‘admin’ tiene asignado el ‘role’ con id=1.

	Mostrar todo	Número de filas:	25	Filtrar filas:	Buscar en esta tabla	
tabla: role_user						
	← T →	id	role_id	user_id	created_at	updated_at
	<input type="checkbox"/> Editar Copiar Borrar	1	1	1	2020-09-13 17:50:59	2020-09-13 17:50:59

Ahora nos vamos registrar un usuario para ver que se active el evento ‘Registered’ y veremos lo que retorna la función ‘handle()’. Hasta el momento solo retornamos la información con ‘dd()’:

Registraremos el usuario llamado ‘test’;

n Register

Register

Name: test

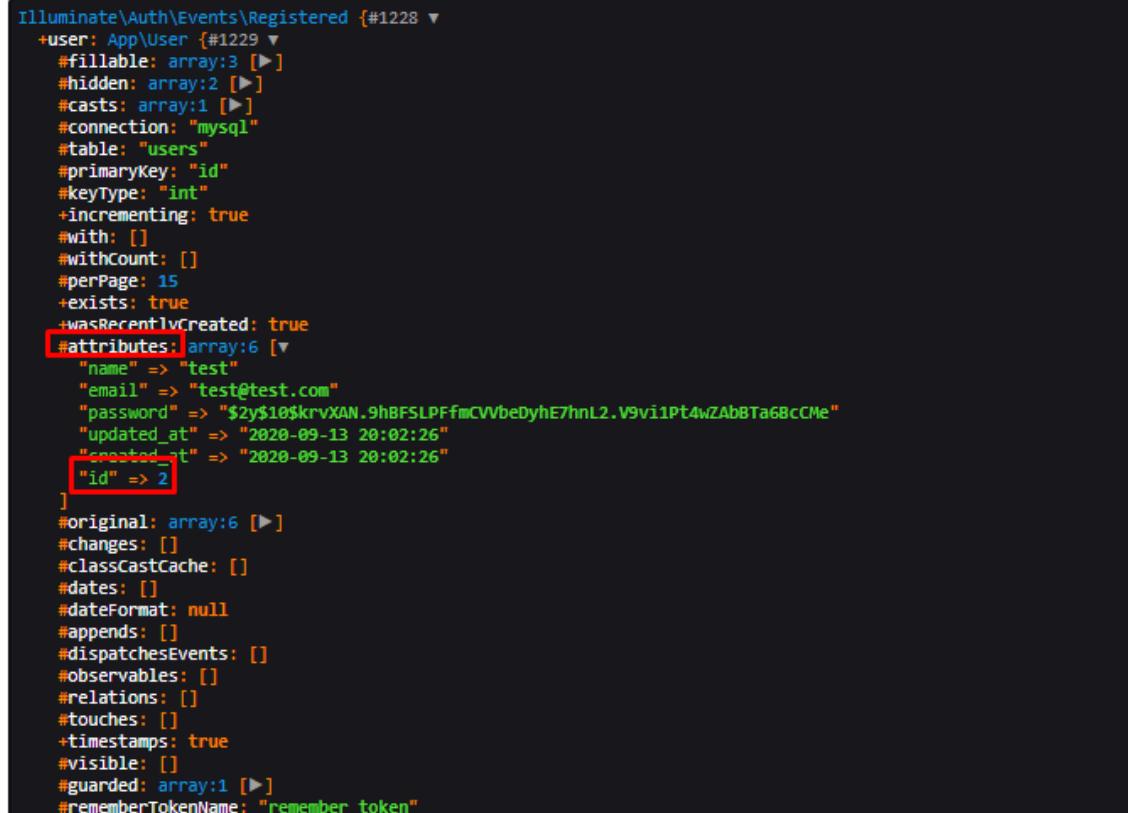
E-Mail Address: test@test.com

Password: *****

Confirm Password: *****

Register

Y el resultado es;



```
Illuminate\Auth\Events\Registered {#1228 ▾
  +user: App\User {#1229 ▾
    #fillable: array:3 [▶]
    #hidden: array:2 [▶]
    #casts: array:1 [▶]
    #connection: "mysql"
    #table: "users"
    #primaryKey: "id"
    #keyType: "int"
    +incrementing: true
    #with: []
    #withCount: []
    #perPage: 15
    +exists: true
    +wasRecentlyCreated: true
    #attributes: array:6 [▼
      "name" => "test"
      "email" => "test@test.com"
      "password" => "$2y$10$krvXAN.9hBFSLPFFmCVVbeDyhE7hnL2.V9vi1Pt4wZAbBTa68cCMe"
      "updated_at" => "2020-09-13 20:02:26"
      "created_at" => "2020-09-13 20:02:26"
      "id" => 2
    ]
    #original: array:6 [▶]
    #changes: []
    #classCastCache: []
    #dates: []
    #dateFormat: null
    #appends: []
    #dispatchesEvents: []
    #observables: []
    #relations: []
    #touches: []
    +timestamps: true
    #visible: []
    #guarded: array:1 [▶]
    #rememberTokenName: "remember_token"
  }
}
```

Vemos que estamos obteniendo la información del usuario registrado.

Ahora volvemos al método ‘handle()’ del archivo ubicado en la ruta “app->listeners->RegisteredEvent.php”

Y aplicamos la función ‘sync()’ al usuario que se registra. Aquí le estamos pasando el ‘id’ del role ‘registered user’ revisamos en la base de datos de que este tiene el id ‘2’ luego de ejecutar el seeder.

```
27     public function handle(Registered $event)
28     {
29         //dd($event);
30         //Asignar el role 'registered user' id=2
31         //a un usuario que registra.
32         $event->user->roles()->sync([ 2 ]);
33     }
34 }
```

Lo siguientes es probar. Primero registremos un usuario llamado: ‘final’;

Register

Name	<input type="text" value="Final"/>
E-Mail Address	<input type="text" value="final@final.com"/>
Password	<input type="password" value="*****"/>
Confirm Password	<input type="password" value="*****"/>
	<input type="button" value="Register"/>

Lo registramos:

Laravel Final

Dashboard
You are logged in!

Luego verificamos en la base de datos:

Vemos que en la tabla 'users' existe;

+ Opciones

	id	name	email	email_verified_at	password	remember_token
<input type="checkbox"/> Editar Copiar Borrar	1	admin	admin@admin.com	NULL	\$2y\$10\$WnkMYg26LQN28wB/dgoqU.l/aE8FcnYWluwD7DuLGaO...	NULL
<input type="checkbox"/> Editar Copiar Borrar	2	test	test@test.com	NULL	\$2y\$10\$krvXAN.9hBFSLPFfmCVVbeDyhE7hnL2.V9vi1Pt4wZA...	NULL
<input type="checkbox"/> Editar Copiar Borrar	3	Final	final@final.com	NULL	\$2y\$10\$uR14G65ID.2Xs6aXKC7QA.8f3Ewl4qAakigmiLaF2jz...	NULL

[↑](#) [Seleccionar todo](#) Para los elementos que están marcados: [Editar](#) [Copiar](#) [Borrar](#) [Expedir](#)

Y finalmente vemos que el usuario 'final' con 'id=3' tiene role con 'id=2' el cual es el role 'registered user'.

+ Opciones

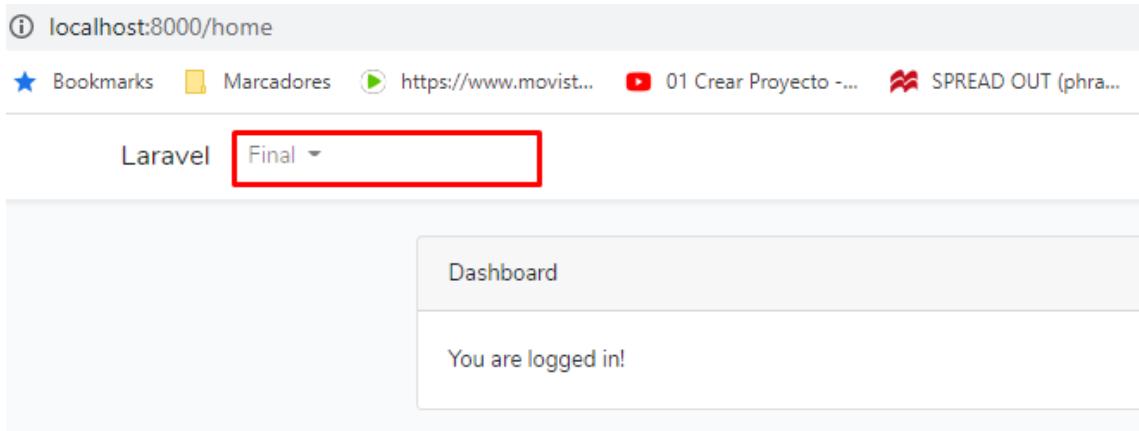
```
SELECT * FROM `role_user`
```

[Perilar](#)

Mostrar todo | Número de filas: 25 ▾ Filtrar filas: Buscar en esta tabla Sort by key: Ninguna

	id	role_id	user_id	created_at	updated_at
<input type="checkbox"/> Editar Copiar Borrar	1	1	1	2020-09-13 17:50:59	2020-09-13 17:50:59
<input type="checkbox"/> Editar Copiar Borrar	2	2	3	2020-09-13 20:31:41	2020-09-13 20:31:41

[↑](#) [Seleccionar todo](#) Para los elementos que están marcados: [Editar](#) [Copiar](#) [Borrar](#) [Expedir](#)



Sin embargo el usuario registrado ‘final’ con el role ‘registered user’ este por defecto **no tiene asignado ningún permiso**. Lo que vamos a hacer es asignarle el permiso de ‘List user’ al role ‘registered user’ para que cuando el usuario se registre luego pueda ver la opción de menú ‘user’, así cada usuario registrado podrá ver los usuarios.

Nos vamos al archivo ‘FreddyPermissionInforSeeder’ o como se llame el de usted, abrimos el archivo del seeder y nos ubicamos debajo del permiso ‘List User’ y llamamos a la variable ‘\$registereduser’ la cual contiene el ‘id’ del role ‘registrado’ y le pasamos el id del permiso;

```
105     //Permiso para Users
106     $permission = Permission::create([
107         'name' => 'List user',
108         'slug' => 'user.index',
109         'description' => 'An user can list a user',
110     ]);
111     //asignar permiso de 'list user' al role 'registered user';
112     $rolregistereduser->permissions()->sync([ $permission->id ]);
```

Luego volvemos a hacer las migraciones y ejecutar el seeder;

```
#> php artisan migrate:fresh --seed
```

```
C:\laragon\www\RolesPermisos
λ php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table #> php artisan migrate:fresh
Migrated: 2014_10_12_000000_create_users_table (0.41 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.4 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (0.27 seconds)
Migrating: 2020_09_08_170810_create_roles_table
Migrated: 2020_09_08_170810_create_roles_table (0.63 seconds)
Migrating: 2020_09_08_182701_create_role_user_table
Migrated: 2020_09_08_182701_create_role_user_table (1.48 seconds)
Migrating: 2020_09_08_215500_create_permissions_table
Migrated: 2020_09_08_215500_create_permissions_table (0.68 seconds)
Migrating: 2020_09_08_220408_create_permission_role_table
Migrated: 2020_09_08_220408_create_permission_role_table (1.45 seconds)
Seeding: FreddyPermissionInfoSeeder
Seeded: FreddyPermissionInfoSeeder (1.33 seconds)
Database seeding completed successfully.
```

Verificamos en la base de datos en la tabla ‘permission_role’ de que el role con ‘id=2’ o ‘registered user’ tiene el permiso con ‘id=6’ que es ‘List User’;

Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,0021 segundos.)						
SELECT * FROM `permission_role`						
<input type="checkbox"/> Mostrar todo Número de filas: 25 <input type="button" value="▼"/> Filtrar filas: <input type="text" value="Buscar en esta tabla"/>						
+ Opciones						
← T →	▼	id	role_id	permission_id	created_at	updated_at
<input type="checkbox"/>	<input type="button" value="Editar"/>	<input type="button" value="Copiar"/>	<input type="button" value="Borrar"/>	1	2	6

Ahora volvemos a registrar nuestro usuario ‘final’;

```
#> php artisan serve
```

Register

Name

E-Mail Address

Password

Confirm Password

Register

Luego de hacer 'login' vemos que puede ver la opción 'User';

The screenshot shows a web browser window with the URL `localhost:8000/home`. The page title is "Dashboard". On the left, there is a sidebar with two items: "Laravel" and "User", where "User" is highlighted with a red box. The main content area displays the message "You are logged in!".

Finalmente si accede a esa opción el usuario puede ver los usuarios.

List of Users				
#	Name	Email	Role(s)	Options
2	Final	final@final.com	Registered User	
1	admin	admin@admin.com	Admin	

Y como es obvio el role 'registered user' solo tiene un permiso por defecto:

Permissions List

- 1 - List role (*An user can list a role*)
- 2 - Show role (*An user can see a role*)
- 3 - Create role (*An user can create a role*)
- 4 - Edit role (*An user can Edit a role*)
- 5 - Destroy role (*An user can destroy a role*)
- 6 - List user (*An user can list a user*)
- 7 - Show user (*An user can see a user*)
- 8 - Edit user (*An user can Edit a user*)
- 9 - Destroy user (*An user can destroy a user*)
- 10 - Show own user (*An user can see own user*)
- 11 - Edit own user (*An user can Edit own user*)

GITHUB: <https://github.com/alcarazolabs/Jhonatan-permisos>

FIN

