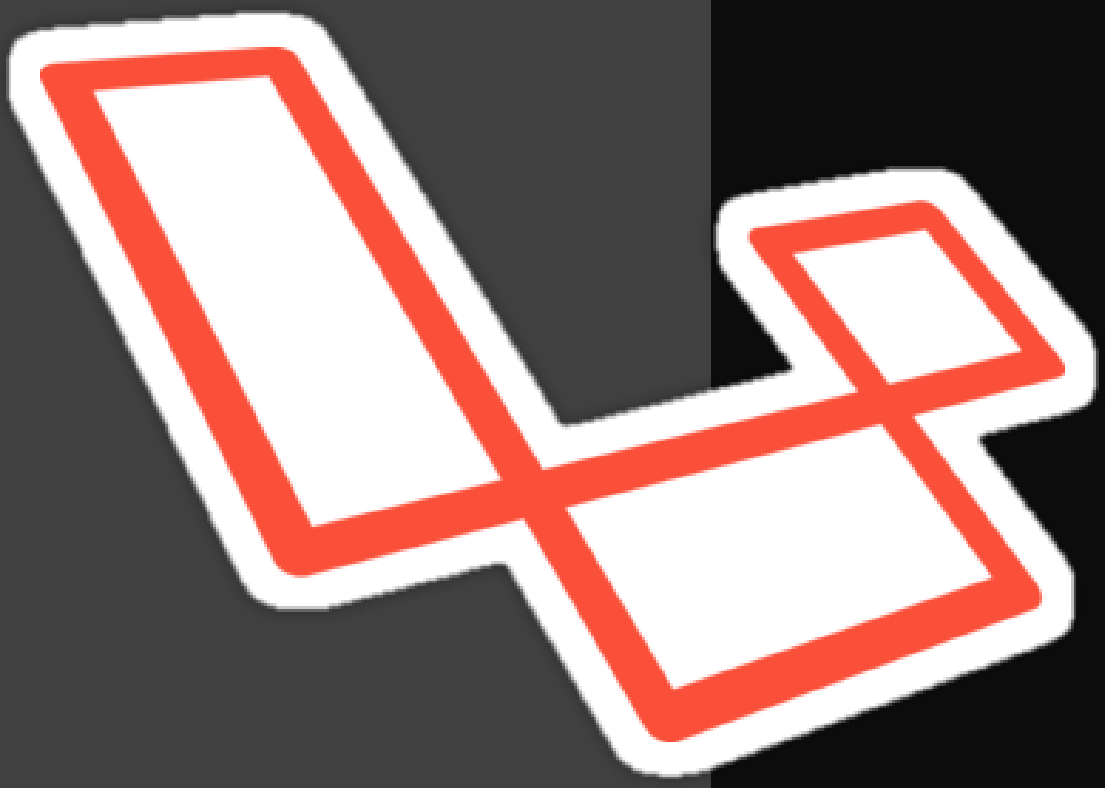
Laravel Portal



José Antonio García Torrecillas

IES Al-Ándalus – 2ºDAW

Contenido

1.- TABLAS 2
2.- PROYECTO LARAVEL Y SQLITE 2
2.1.- CREAR PROYECTO LARAVEL 2
2.2.- BASE DE DATOS SQLITE 2
2.3.- IDIOMA DEL PROYECTO 3
3.- TABLA PERFIL 4
3.1.- FACTORY 4
3.2.- MIGRATION 5
3.3.- MODEL 5
3.4.- SEEDER 6
3.5.- CREAR DATOS 6
3.6.- INDEX 6
3.6.1.- CONTROLADOR 9
3.7.- CREATE 9
3.7.1.- CONTROLADOR 9
3.8.- UPDATE 10
3.8.1.- CONTROLADOR 11
3.9.- DELETE 11
3.9.1.- CONTROLADOR 12
3.10.- SHOW 12
3.10.1.- CONTROLADOR 13
4.- TABLA USUARIO 13
4.1.- FACTORY 13
4.2.- MIGRATION 14
4.3.- MODEL 14
4.4.- SEEDER 15
4.5.- CREAR DATOS 15
4.6.- INDEX 15
4.6.1.- CONTROLADOR 16
4.7.- CREATE 16
4.7.1.- CONTROLADOR 16
4.8.- UPDATE 17
4.8.1.- CONTROLADOR 18
4.9.- DELETE 19
4.9.1.- CONTROLADOR 19
4.10.- SHOW 19
4.10.1.- CONTROLADOR 20
5.- RELACIÓN 1:N 20
5.1.- CREATE Y UPDATE DE USUARIOS 21
5.2.- SHOW PERFIL CON USUARIOS 22
5.3.- SHOW USUARIOS CON PERFIL 22

1.- TABLAS

Creamos las tablas perfil y usuarios (1:N). Donde perfil (nombre unique, descripción text) tiene varios usuarios (nomusu unique, mail unique y localidad).

```
perfiles(nombre unique, descripcion)
usuarios(nomusu unique, mail unique, localidad)
```

2.- PROYECTO LARAVEL Y SQLITE

2.1.- CREAR PROYECTO LARAVEL

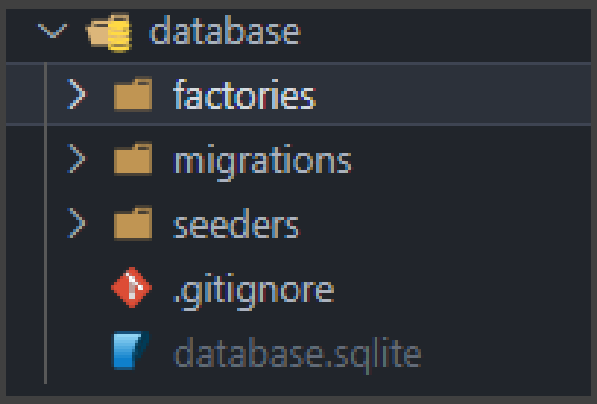
Para crear la carpeta debemos escribir este comando en la carpeta donde queremos trabajar

```
composer create-project laravel/laravel proyecto
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Versión 10.0.19042.985]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Jose\Documents\GitHub>composer create-project laravel/laravel LaravelPortal
Creating a "laravel/laravel" project at "./LaravelPortal"
Installing laravel/laravel (v8.5.18)
  - Installing laravel/laravel (v8.5.18): Extracting archive
Created project in C:\Users\Jose\Documents\GitHub\LaravelPortal
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
```

2.2.- BASE DE DATOS SQLITE

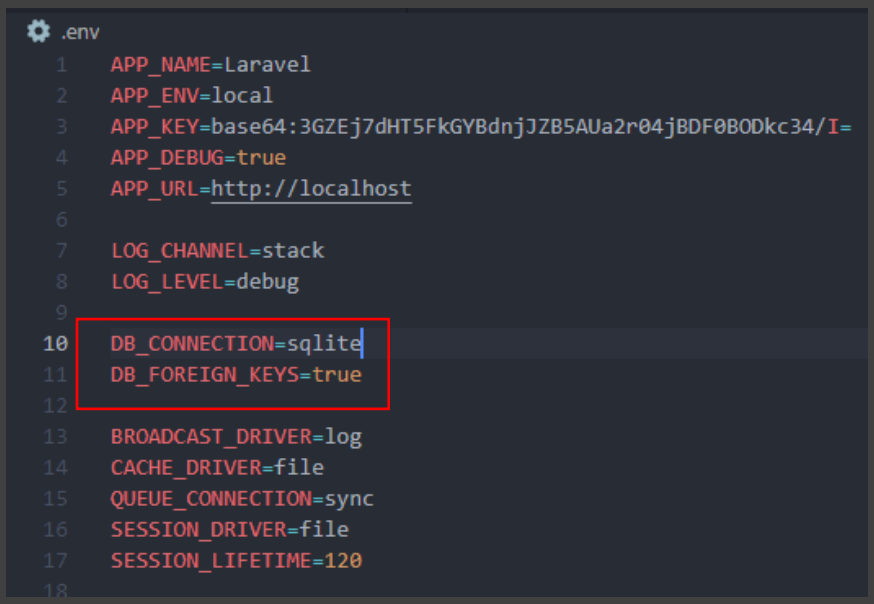
Creamos el archivo database.sqlite dentro de la carpeta database del proyecto laravel.



Dentro del archivo .env borramos las líneas dedicadas a la base de datos por defecto y lo sustituimos por estas líneas para establecer la conexión con la base de datos sqlite

DB_CONNECTION=sqlite

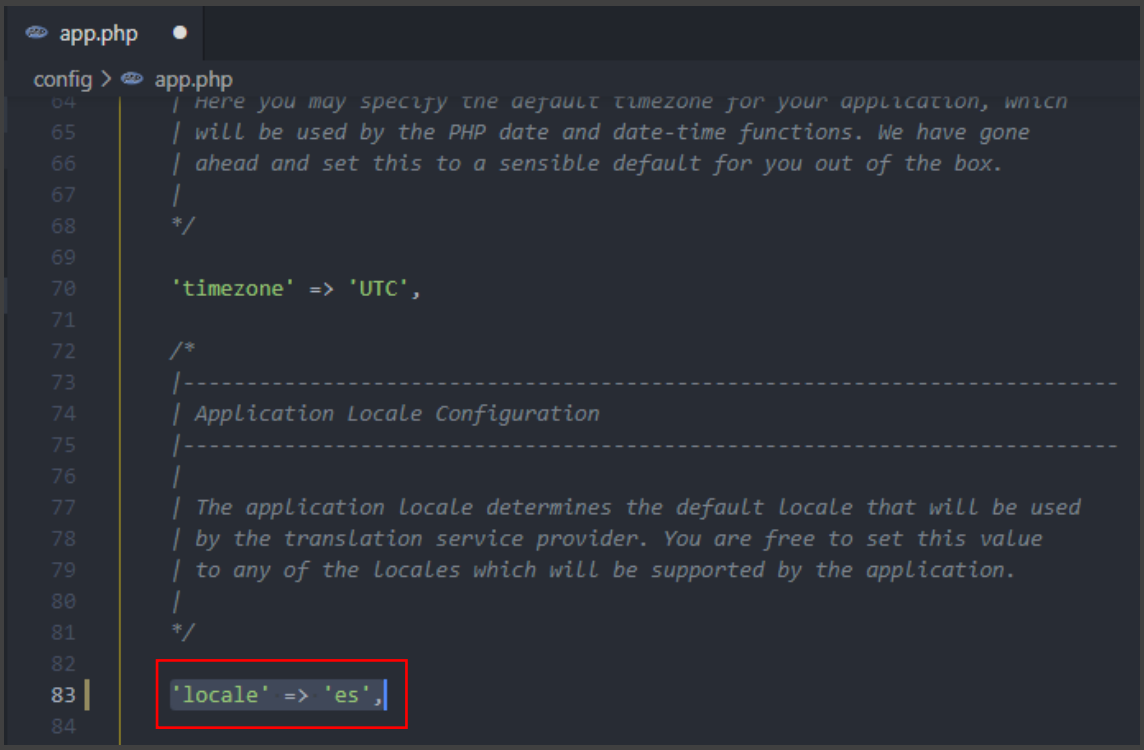
DB_FOREIGN_KEYS=true

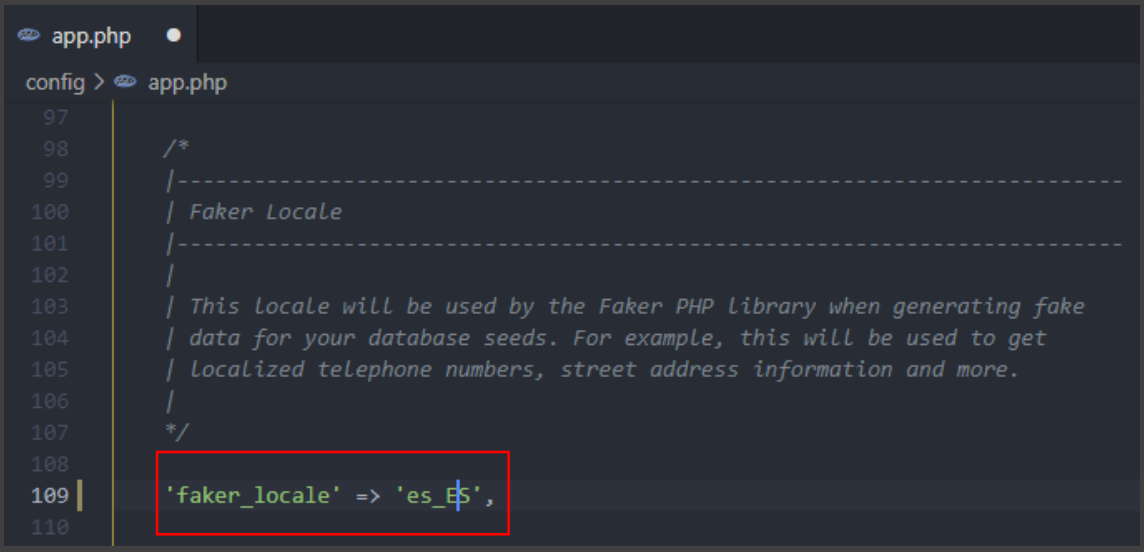


2.3.- IDIOMA DEL PROYECTO

Realizamos este intercambio de idiomas para que cree registros aleatorios faker y se obtenga datos aleatorios de formato español.

'locale' => 'es', 'faker_locale' => 'es_ES'





3.- TABLA PERFIL

Para crear la tabla perfil con su factory, migrate, seeder y model se debe teclear este comando.

php artisan make:model Perfil -mfcr

```
C:\Users\Jose\Documents\GitHub\LaravelPortal>php artisan make:model Perfil -mfcr
Model created successfully.
Factory created successfully.
Created Migration: 2021_06_01_172459_create_perfils_table
Controller created successfully.
```

3.1.- FACTORY

Insertaremos los atributos de perfil factory. Factory se encarga de crear objetos perfil con atributos aleatorios. Estos elementos aleatorios se generan gracias a los faker.

Métodos faker

'atributo' => $this -> faker -> metodo

- Uso randomElement para que cree los elementos del array que proporciono y los proyecte cuando haga la migración.
- Uso text para que me proporcione un texto random (tiene un máximo de 120 caracteres) para que lo proyecte en la migración.

```
    public function definition()
    {
        return [
            'nombre' => $this->faker->unique()->randomElement(['Publico', 'Privado', 'Comunitario', 'Administrativo', 'Premium']),
            'descripcion' => $this->faker->text()
        ];
    }
```

LaravelPortal / database / factories / PerfilFactory.php

3.2.- MIGRATION

Se creará una tabla “perfils” con los atributos proporcionados y se alojaran en el archivo migration. Este archivo se encarga de crear la tabla perfil con los atributos recogidos de factory. En la creación de la tabla se crea su id autoincrementado id() y también su fecha de creación y actualización timestamps().

Documentación migration

$table -> string/ integer(‘nombreDelAtributo’);

```
public function up()
    {
        Schema::create('perfils', function (Blueprint $table) {
            $table->id();
            $table->string('nombre')->unique();
            $table->string('descripcion');
            $table->timestamps();
        });
    }
```

LaravelPortal / database / migrations / 2021_06_01_172459_create_perfils_table.php

3.3.- MODEL

Se crea el objeto perfil con sus atributos nombre y descripción con $fillable para que obtenga los datos de cada perfil y se realice la migración en seeder.

```
    protected $fillable = [
        'nombre',
        'descripcion'
    ];
```

LaravelPortal / app / Models / Perfil.php

3.4.- SEEDER

En el factory para que inserte cinco objetos de tipo perfil dentro de la tabla proporcionada por migration.

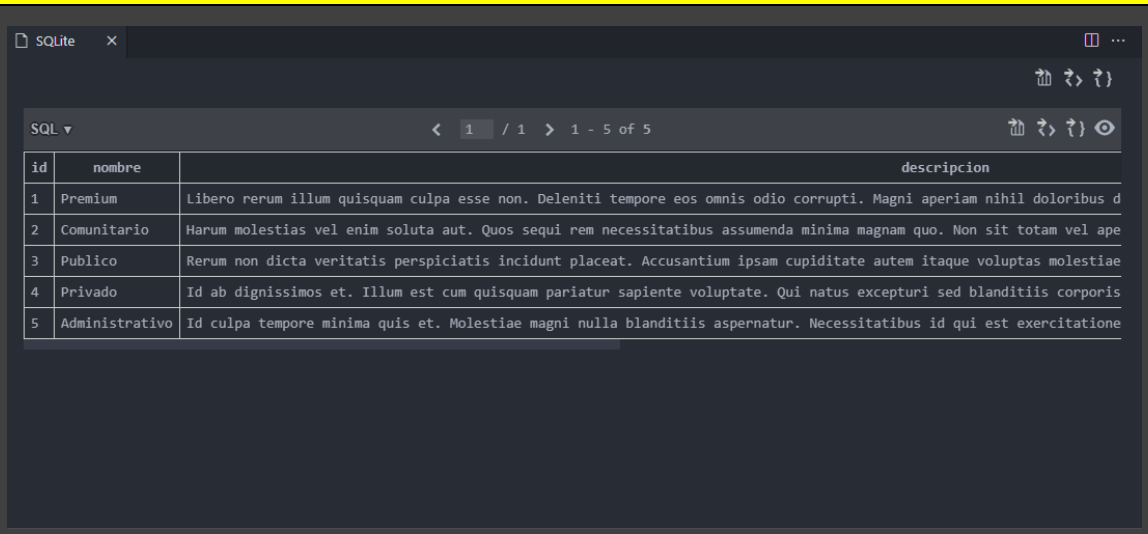
```
\App\Models\Perfil::factory(5)->create();
```

LaravelPortal / database / seeders / DatabaseSeeder.php

3.5.- CREAR DATOS

Una vez hecho las migraciones, seeders y factorys. Ejecutamos la migración

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



id	nombre	descripcion
1	Premium	Libero rerum illum quisquam culpa esse non. Deleniti tempore eos omnis odio corrupti. Magni aperiam nihil doloribus d
2	Comunitario	Harum molestias vel enim soluta aut. Quos sequi rem necessitatibus assumenda minima magnam quo. Non sit totam vel ape
3	Publico	Rerum non dicta veritatis perspiciatis incidunt placeat. Accusantium ipsam cupiditate autem itaque voluptas molestiae
4	Privado	Id ab dignissimos et. Illum est cum quisquam pariatur sapiente voluptate. Qui natus excepturi sed blanditiis corporis
5	Administrativo	Id culpa tempore minima quis et. Molestiae magni nulla blanditiis aspernatur. Necessitatibus id qui est exercitatione

3.6.- INDEX

Para tener un código más limpio se utilizará x-form para evitar mucho código html sin embargo no se puede tener mucha personalización de esta forma.

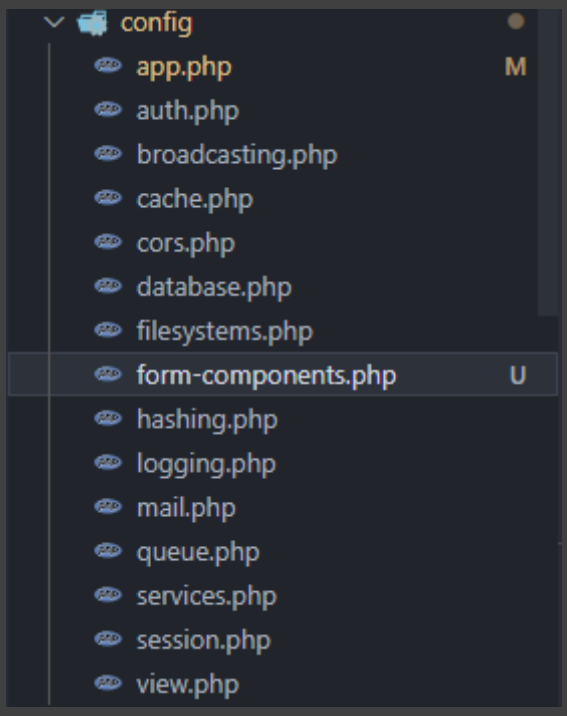
```
composer require protonemedia/laravel-form-components
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Users\Jose\Documents\GitHub\LaravelPortal>composer require protonemedia/laravel-form-components
Using version ^2.5 for protonemedia/laravel-form-components
./composer.json has been updated
Running composer update protonemedia/laravel-form-components
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking protonemedia/laravel-form-components (2.5.4)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing protonemedia/laravel-form-components (2.5.4): Extracting archive
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/sail
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Discovered Package: protonemedia/laravel-form-components
Package manifest generated successfully.
75 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

Para efectuar la extracción de la librería para que cargue en nuestro vendor

```
php artisan vendor:publish
```

```
C:\Users\Jose\Documents\GitHub\LaravelPortal>php artisan vendor:publish

 Which provider or tag's files would you like to publish?:
  [0 ] Publish files from all providers and tags listed below
  [1 ] Provider: Facade\Ignition\IgnitionServiceProvider
  [2 ] Provider: Fideloper\Proxy\TrustedProxyServiceProvider
  [3 ] Provider: Fruitcake\Cors\CorsServiceProvider
  [4 ] Provider: Illuminate\Foundation\Providers\FoundationServiceProvider
  [5 ] Provider: Illuminate\Mail\MailServiceProvider
  [6 ] Provider: Illuminate\Notifications\NotificationServiceProvider
  [7 ] Provider: Illuminate\Pagination\PaginationServiceProvider
  [8 ] Provider: Laravel\Sail\SailServiceProvider
  [9 ] Provider: Laravel\Tinker\TinkerServiceProvider
  [10] Provider: ProtoneMedia\LaravelFormComponents\Support\ServiceProvider
  [11] Tag: config
  [12] Tag: cors
  [13] Tag: flare-config
  [14] Tag: ignition-config
  [15] Tag: laravel-errors
  [16] Tag: laravel-mail
  [17] Tag: laravel-notifications
  [18] Tag: laravel-pagination
  [19] Tag: sail
  [20] Tag: views
 > 10
```



En este código se mostrará la tabla perfil con los atributos nombre y descripción con los botones editar, borrar, mostrar y el botón crear. Con la paginación gracias a los links.

- Para ello creamos una carpeta en resources llamado components para crear una plantilla y no repetir <!html>

- Una vez creado plantilla.blade.php creamos la carpeta perfil con su index.blade.php con x-mensaje proporcionado desde components

- Creamos la tabla con sus datos correspondientes gracias a foreach($perfil)

- Para paginar los perfiles sin acumulaciones de filas debemos proyectar links y se llaman a Bootstrap desde appserviceprovider

3.6.1.- CONTROLADOR

Ordenaremos por nombre la tabla perfil y lo paginaremos de 5 en 5 devolviendo la vista index con su segundo parámetro perfil

```
    public function index()
    {
        $perfil=Perfil::orderBy('nombre')->paginate(5);
        return view('perfiles.index', compact('perfil'));
    }
```

LaravelPortal / app / Http / Controllers / PerfilController.php

3.7.- CREATE

Para llevar a cabo la función de crear nuevos registros en la tabla créate primero se debe dedicar un botón en index para que dirigirnos a la ventana créate con su formulario.

```
<a href="{{route('perfil.create')}}" class="btn btn-success mb-2"><i class="fas fa-user-plus"></i> Crear perfil</a>
```

Introducimos los campos nombre y descripción dentro del formulario (form) de método POST para que ejecute y obtenga los datos para ser procesador en el método store del controlador. El @csfr se introduce por motivos de seguridad.

Convocamos el x-error, que procede de la carpeta components

Método error

3.7.1.- CONTROLADOR

Para llevar a cabo la transición entre index y create debemos convocar el método create devolviendo la vista de resources.

```
    public function create()
    {
        return view('perfiles.create');
    }
```

Validamos los atributos obtenidos de los inputs del créate (resources) si la validación es correcta guardara el objeto y se añadirá a la tabla con el mensaje procedente de x-mensaje si no se guarda en condiciones agregara un mensaje de error.

```
    public function store(Request $request)
    {
        $request->validate([
            'nombre'=>['string', 'required', 'max:20', 'min:1'],
            'descripcion'=>['string', 'required', 'max:200', 'min:1']
        ]);
        try{
            Perfil::create($request->all());
        }catch(PDOException $ex){
            return redirect()->route('perfil.index')->with("mensaje","Error al
guardar perfil");
        }
        return redirect()->route('perfil.index')->with("mensaje","Perfil creado");
    }
```

3.8.- UPDATE

Para llevar a cabo la función de actualizar los registros primero se debe dedicar un botón en cada fila/registro así que debemos recoger el id del perfil seleccionado.

```
<a href="{{route('perfil.edit', $item)}}" class="btn btn-warning"><i class="fas fa-
edit"></i> Editar</a>
```

Obtenemos la información gracias al method(PUT) y bind(perfil) si los inputs se corresponden a los names del objeto y del atributo del objeto perfil.

Cambiamos los campos nombre y descripción dentro del formulario (form) de método POST para que ejecute y obtenga los datos para ser procesador en el método update del controlador. El @csfr se introduce por motivos de seguridad.

Convocamos el x-error, que procede de la carpeta components

Método error

3.8.1.- CONTROLADOR

Para llevar a cabo la transición entre index y edit debemos convocar el método edit devolviendo la vista de resources.

```
    public function edit(Perfil $perfil)
    {
        return view('perfiles.edit', compact('perfil'));
    }
```

Validamos los atributos obtenidos de los inputs del update (resources) si la validación es correcta actualizara el objeto gracias al parámetro perfil y se actualizara el perfil en la tabla con el mensaje procedente de x-mensaje.

```
    public function update(Request $request, Perfil $perfil)
    {
        $request->validate([
            'nombre'=>['string', 'required', 'max:20', 'min:1',
'unique:perfils,nombre'.$perfil->id],
            'descripcion'=>['string', 'required', 'max:200', 'min:1']
        ]);
        try{
            $perfil->update($request->all());
            return redirect()->route('perfil.index')->with("mensaje","Perfil
actualizado");
        }catch(PDOException $ex){
            return redirect()->route('perfil.index')->with("mensaje","Error al
actualizar perfil");
        }
    }
```

3.9.- DELETE

Para llevar a cabo la función de borrar los registros primero se debe dedicar un botón en cada fila/registro así que debemos recoger el id del perfil seleccionado para borrar.

Ya que no necesitamos una vista para borrar entonces se dedicará el csfr y el método DELETE con el id de la fila seleccionada.

```
<form name="delete" method="POST" action="{{route('perfil.destroy', $item)}}">

@csrf

@method("DELETE")

<button type="submit" class="btn btn-danger"><i class="fas fa-trash"></i>
Borrar</button>
```

3.9.1.- CONTROLADOR

Una vez obtenido el perfil procedente de la fila clickeada de index llamamos al método delete y si no hay problemas lo borrara y actualizara de nuevo el index redigiendonos de nuevo al index.

```
public function destroy(Perfil $perfil)

    {

        try{

            $perfil->delete();

        }catch(PDOException $ex){

            return redirect()->route('perfil.index')->with("mensaje","Error al borrar
perfil");

        }

        return redirect()->route('perfil.index')->with("mensaje","Perfil borrado");

    }
```

3.10.- SHOW

Tan solo es obtener el id de la fila donde se ha seleccionado el botón info y recoger su id y dirigirlo a la nueva vista.

```
<th scope="row">

   <a href="{{route('perfil.show', $item)}}" class="btn btn-info"><i class="fas fa-
info"></i> Info</a>

 </th>
```

Y obtenemos la información y lo escribimos gracias al id obtenido del index que se nombró anteriormente (ignorare el foreach de usuarios hasta completar la siguiente tabla)

3.10.1.- CONTROLADOR

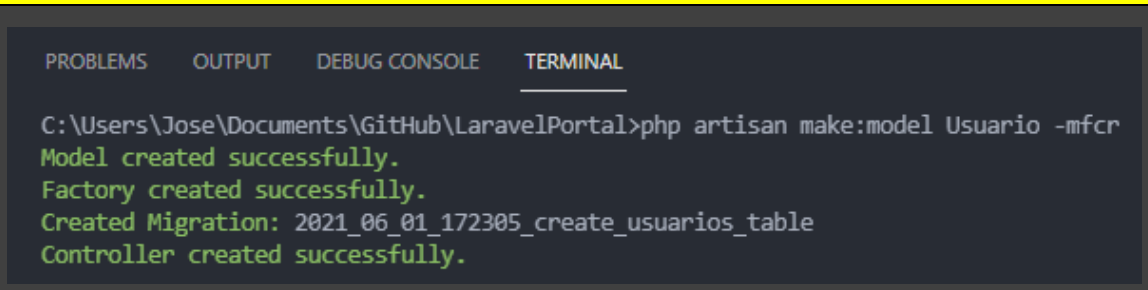
Solo obtengo el id de la fila donde se ha seleccionado el botón info y recoger su id y dirigirlo a la nueva vista.

```
public function show(Perfil $perfil)
{
    return view('perfiles.show', compact('perfil'));
}
```

4.- TABLA USUARIO

Para crear la tabla usuario con su factory, migrate, seeder y model se debe teclear este comando.

php artisan make:model Usuario -mfcr



4.1.- FACTORY

Insertaremos los atributos de usuario factory. Factory se encarga de crear objetos usuario con atributos aleatorios. Estos elementos aleatorios se generan gracias a los faker.

Métodos faker

'atributo' => $this -> faker -> metodo

- Uso nomusu para que me proporcione un usuario único para que lo proyecte en la migración.
- Uso mail para que me proporcione un email random para que lo proyecte en la migración.
- Uso localidad para que me proporcione una localidad random para que lo proyecte en la migración.

```php
public function definition()
{
    $perfiles=Perfil::all('id');
    return [
        'nomusu' => $this->faker->unique()->userName(),
        'mail' => $this->faker->unique()->freeEmail(),
        'localidad' => $this->faker->city
    ];
}
```

LaravelPortal / database / factories / UsuarioFactory.php 

4.2.- MIGRATION

Se creará una tabla “usuarios” con los atributos proporcionados y se alojaran en el archivo migration. Este archivo se encarga de crear la tabla usuario con los atributos recogidos de factory. En la creación de la tabla se crea su id autoincrementado id() y también su fecha de creación y actualización timestamps().

Documentación migration

$table -> string/ integer(‘nombreDelAtributo’);

```php
public function up()
{
    Schema::create('usuarios', function (Blueprint $table) {
        $table->id();
        $table->string('nomusu')->unique();
        $table->string('mail')->unique();
        $table->string('localidad');
        $table->timestamps();
    });
}
```

LaravelPortal / database / migrations / 2021_06_01_172305_create_usuarios_table.php 

4.3.- MODEL

Se crea el objeto usuario con sus atributos nombre y descripción con $fillable para que obtenga los datos de cada usuario y se realice la migración en seeder.

```php
protected $fillable = [
    'nomusu',
    'mail',
    'localidad'
```

```
    ];
```

LaravelPortal / app / Models / Usuario.php

4.4.- SEEDER

En el factory para que inserte treinta objetos de tipo usuario dentro de la tabla proporcionada por migration.

```
        \App\Models\Usuario::factory(30)->create();
```

LaravelPortal / database / seeders / DatabaseSeeder.php

4.5.- CREAR DATOS

Una vez hecho las migraciones, seeders y factorys. Ejecutamos la migración

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

SQL ▾ ‹ 1 / 1 › 1 - 30 of 30

id	nomusu	mail	localidad	created_at	updated_at
1	puga.natalia	requena.ursula@hotmail.com	El Román	2021-06-01 18:01:56	2021-06-01 18:01:56
2	manuel.asensio	cperales@hispavista.com	Las Soler de Lemos	2021-06-01 18:01:56	2021-06-01 18:01:56
3	arnau.corona	noelia56@yahoo.es	Los Palomo del Bages	2021-06-01 18:01:56	2021-06-01 18:01:56
4	blanca.herrera	cdelgadillo@terra.com	Las Velázquez	2021-06-01 18:01:56	2021-06-01 18:01:56
5	sandra.montenegro	olivia.cordoba@live.com	Vanegas de Ulla	2021-06-01 18:01:56	2021-06-01 18:01:56
6	iabad	ana73@yahoo.com	O Alemán	2021-06-01 18:01:56	2021-06-01 18:01:56
7	apelaez	vsanz@yahoo.es	Polanco Baja	2021-06-01 18:01:56	2021-06-01 18:01:56
8	pesparza	noelia.clemente@gmail.com	Los Miguel	2021-06-01 18:01:56	2021-06-01 18:01:56
9	yeray.carrasco	lola82@latinmail.com	Las Aponte	2021-06-01 18:01:56	2021-06-01 18:01:56
10	puente.santiago	guillem.garibay@yahoo.com	Rincón del Pozo	2021-06-01 18:01:56	2021-06-01 18:01:56
11	martina.rosales	amparo74@hotmail.com	O Rubio del Mirador	2021-06-01 18:01:56	2021-06-01 18:01:56
12	saldivar.enrique	garrido.manuel@live.com	L' Hernández del Pozo	2021-06-01 18:01:56	2021-06-01 18:01:56
13	franciscojavier45	raquel.prieto@yahoo.es	Ordoñez de Ulla	2021-06-01 18:01:56	2021-06-01 18:01:56

4.6.- INDEX

En este código se mostrará la tabla usuario con los atributos nomusu, mail, localidad con los botones editar, borrar, mostrar. Con la paginación gracias a los links.

- Para ello creamos una carpeta en resources llamado components para crear una plantilla y no repetir <!html>

- Una vez creado plantilla.blade.php creamos la carpeta usuario con su index.blade.php con x-mensaje proporcionado desde components

- Creamos la tabla con sus datos correspondientes gracias a foreach($usuario)

- Para paginar los usuarios sin acumulaciones de filas debemos proyectar links y se llaman a Bootstrap desde appserviceprovider

4.6.1.- CONTROLADOR

Ordenaremos por nomusu la tabla usuario y lo paginaremos de 10 en 10 devolviendo la vista index con su segundo parámetro usuario

```
    public function index()
    {
        $usuario=Usuario::orderBy('nomusu')->paginate(10);
        return view('usuarios.index', compact('usuario'));
    }
```

LaravelPortal / app / Http / Controllers / UsuarioController.php

4.7.- CREATE

Para llevar a cabo la función de crear nuevos usuarios en la tabla create primero se debe dedicar un botón en index para que dirigirnos a la ventana create con su formulario.

```
        <a href="{{route('usuario.create')}}" class="btn btn-success mb-2"><i
class="fas fa-user-plus"></i> Crear usuario</a>
```

Introducimos los campos nomusu, mail, localidad dentro del formulario (form) de método POST para que ejecute y obtenga los datos para ser procesador en el método store del controlador. El @csfr se introduce por motivos de seguridad.

Convocamos el x-error, que procede de la carpeta components

Método error

4.7.1.- CONTROLADOR

Para llevar a cabo la transición entre index y create debemos convocar el método create devolviendo la vista de resources.

```
    public function create()
    {
        return view('usuarios.create', compact('perfils'));
    }
```

Validamos los atributos obtenidos de los inputs del créate (resources) si la validación es correcta guardara el objeto y se añadirá a la tabla con el mensaje procedente de x-mensaje si no se guarda en condiciones agregara un mensaje de error.

```
    public function store(Request $request)
    {
        $request->validate([
            'nomusu'=>['string', 'required', 'max:20', 'min:1'],
            'mail'=>['string', 'required', 'unique:usuarios,mail'],
            'localidad'=>['string', 'required', 'max:25', 'min:1'
        ]);
        try{
            Usuario::create($request->all());
            return redirect()->route('usuario.index')->with("mensaje","Usuario creado");
        }catch(PDOException $ex){
            return redirect()->route('usuario.index')->with("mensaje","Error al crear usuario");
        }
    }
```

4.8.- UPDATE

Para llevar a cabo la función de actualizar los registros primero se debe dedicar un botón en cada fila/registro así que debemos recoger el id del usuario seleccionado.

```
<a href="{{route('usuario.edit', $item)}}" class="btn btn-warning"><i class="fas fa-user-edit"></i> Editar</a>
```

Obtenemos la información gracias al method(PUT) y bind(usuario) si los inputs se corresponden a los names del objeto y del atributo del objeto usuario.

Cambiamos los campos nombre y descripción dentro del formulario (form) de método POST para que ejecute y obtenga los datos para ser procesador en el método update del controlador. El @csfr se introduce por motivos de seguridad.

Convocamos el x-error, que procede de la carpeta components

Método error

4.8.1.- CONTROLADOR

Para llevar a cabo la transición entre index y edit debemos convocar el método edit devolviendo la vista de resources.

```
    public function edit(Usuario $usuario)
    {
        return view('usuarios.edit', compact('usuario','perfils'));
    }
```

Validamos los atributos obtenidos de los inputs del update (resources) si la validación es correcta actualizara el objeto gracias al parámetro usuario y se actualizara el usuario en la tabla con el mensaje procedente de x-mensaje.

```
    public function update(Request $request, Usuario $usuario){
        $request->validate([
            'nomusu'=>['string', 'required', 'max:20', 'min:1',
'unique:usuarios,nomusu'.$usuario->id],
            'mail'=>['string', 'required', 'unique:usuarios,mail'.$usuario->id],
            'localidad'=>['string', 'required', 'max:25', 'min:1'],
        ]);
        try{
            $usuario->update($request->all());
  return redirect()->route('usuario.index')->with("mensaje","Usuario actualizado");
        }catch(PDOException $ex){
return redirect()->route('usuario.index')->with("mensaje","Error al actualizar usu");
        }
    }
```

4.9.- DELETE

Para llevar a cabo la función de borrar los registros primero se debe dedicar un botón en cada fila/registro así que debemos recoger el id del usuario seleccionado para borrar.

Ya que no necesitamos una vista para borrar entonces se dedicará el csfr y el método DELETE con el id de la fila seleccionada.

```
<form name="delete" method="POST" action="{{route('usuario.destroy', $item)}}">

@csrf

@method("DELETE")

<button type="submit" class="btn btn-danger"><i class="fas fa-trash"></i>
Borrar</button>

</form>
```

4.9.1.- CONTROLADOR

Una vez obtenido el usuario procedente de la fila clickeada de index llamamos al método delete y si no hay problemas lo borrara y actualizara de nuevo el index redigiendonos de nuevo al index.

```
    public function destroy(Usuario $usuario)

    {

        try{

            $usuario->delete();

        }catch(PDOException $ex){

            return redirect()->route('usuario.index')->with("mensaje","Error al
borrar usuario");

        }

        return redirect()->route('usuario.index')->with("mensaje","Usuario
borrado");;

    }
```

4.10.- SHOW

Tan solo es obtener el id de la fila donde se ha seleccionado el botón info y recoger su id y dirigirlo a la nueva vista.

```
<a href="{{route('usuario.show', $item)}}" class="btn btn-info"><i class="fas fa-info"></i>
Info</a>
```

Y obtenemos la información y lo escribimos gracias al id obtenido del index que se nombró anteriormente (ignorare el foreach de usuarios hasta completar la siguiente tabla)

4.10.1.- CONTROLADOR

Solo obtengo el id de la fila donde se ha seleccionado el botón info y recoger su id y dirigirlo a la nueva vista.

```
public function show(Usuario $usuario)
{
    return view('usuarios.show', compact('usuario'));
}
```

5.- RELACIÓN 1:N

Para establecer la conexión entre perfil y usuarios debemos dirigirnos al usuario factory y llamar a los ids de todos los perfiles para que recoja todos los ids perfil

```
$perfiles=Perfil::all('id');
```

Obtenemos los perfiles procedentes de la anterior llamada de la declaración de perfiles y obtenemos todos los ids.

```
'perfil_id' => $perfiles->get(rand(0, count($perfiles)-1))
```

Ahora en migration realizaremos la conexión entre perfil y usuarios como si fuera una secuencia sql con un atributo de conexión perfil_id haciendo referencia entre el id del perfil con el perfil_id de usuarios.

```
$table->foreignId('perfil_id');
$table->foreign('perfil_id')
->references("id")->on('perfils')
->onDelete("cascade")->onUpdate("cascade");
```

Realizamos este método para que devuelva en la vista crear y actualizar un select donde se mostrara los nombres de los perfiles.

```
public static function array(){
    $perfil=Perfil::orderBy('nombre')->get();
    $array=[];
    foreach($perfil as $item){
        $array[$item->id]=$item->nombre;
    }
    return $array;
}
```

Establecemos la conexión con usuarios desde perfil hacia muchos usuarios.

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

Establecemos la conexión con perfil desde usuarios hacia un perfil.

```
public function perfil(){
    return $this->belongsTo(Perfil::class);
}
```

5.1.- CREATE Y UPDATE DE USUARIOS

Traemos los perfiles procedentes del controlador de usuarios por el array de perfiles. Este array procede del modelo perfil

```
<x-form-select name="perfil_id" :options="$perfils" label="Perfil"/>
```

Donde se trae el array de perfiles gracias al método declarado en el modelo del perfil

```
public function create()
{
    $perfils=Perfil::array();
    return view('usuarios.create', compact('perfils'));
}
```

Es necesario aplicar el atributo perfil_id en usuarios para aplicar el guardado y actualización del usuario con sus selects.

```
            'perfil_id'=>['require']
```

5.2.- SHOW PERFIL CON USUARIOS

Se realiza un foreach llamando el perfil y sus usuarios gracias al controlador show

```
                <ul>
                    @foreach ($perfil->usuarios as $item)
                        <li>{{$item->nomusu}}</li>
                    @endforeach
                </ul>
```

5.3.- SHOW USUARIOS CON PERFIL

Lo mismo que el anterior apartado pero llamando el perfil desde usuario gracias al controlador show

```
                    {{$usuario->perfil->nombre}}
```