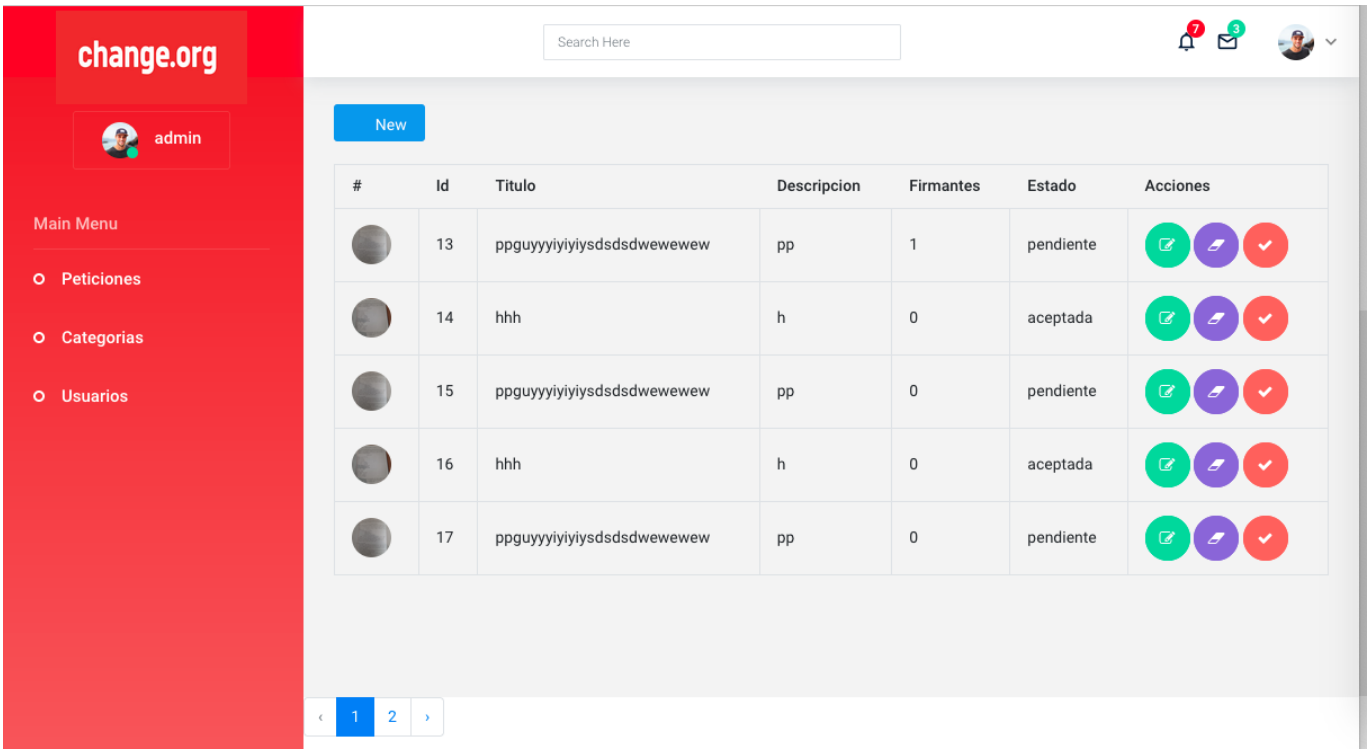
Table of Contents

Admin panel

- Layout
- Admin controllers and views
- ### Authentication redirect
- Authorization
  - Routing by middleware for admin users
  - Authorization by Policies

Admin panel

Now, we are going to create an admin panel in order a user with admin rol, to able to manage `Peticiones`, `Users` and `Categorias`.



Layout

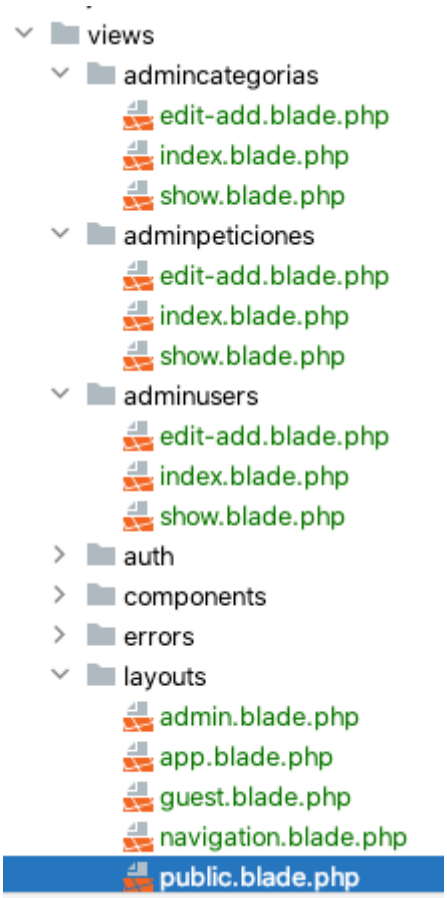
Create a new layout called `admin.blade.php`.

Admin controllers and views

Create the corresponing controllers for the admin actions:

```
php artisan make:controller AdminCategoriasController
php artisan make:controller AdminPeticionesController
php artisan make:controller AdminUsersController
```

Inside of them create the corresponding methods and when needed create the views inside of admin folder like this:



Authentication redirect

Right now, in case any kind of user signs in our application, a redirect to the home will happen.

Imagine we want to redirect to different places depending on the kind of role we have. In order to do so, open `app/Http/Controllers/Auth/AuthenticatedSessionController.php` file. Inside of it, we need to change the `store` method were the user is authenticated and the session is given. After that in case everything goes well, we are going to redirect to different places depending on the role_id:

```
...
if(Auth::user()->role_id==2){
     return redirect()->route('admin.home');
}

return redirect()->intended(RouteServiceProvider::HOME);
...
```

Authorization

Routing by middleware for admin users

Inside of the routing file `web.php` create the corresponding routes for the admin actions related with each of the entities like:

```
Route::middleware('admin')-
>controller(\App\Http\Controllers\AdminPeticionesController::class)-
>group(function () {
    Route::get('admin', 'index')->name('admin.home');
    Route::get('admin/peticiones/index', 'index')-
>name('adminpeticiones.index');
    Route::get('admin/peticiones/{id}', 'show')->name('adminpeticiones.show');
    Route::get('admin/peticion/add', 'create')->name('adminpeticiones.create');
    Route::get('admin/peticiones/edit/{id}', 'edit')-
>name('adminpeticiones.edit');
    Route::post('admin/peticiones', 'store')->name('adminpeticiones.store');
    Route::delete('admin/peticiones/{id}', 'delete')-
>name('adminpeticiones.delete');
    Route::put('admin/peticiones/{id}', 'update')-
>name('adminpeticiones.update');
    Route::put('admin/peticiones/estado/{id}', 'cambiarEstado')-
>name('adminpeticiones.estado');
});
//follow the same for categories and users
```

As you can see in the code above, we are calling to a middleware called `admin`. Whenever we define a middleware and associate it to a route, that means that if the execution of that middleware return true the routing module will give access to that action, and will not in case it returns false.

```
php artisan make:middleware Admin
```

Inside of it:

```
class Admin
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request):
(\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        if (Auth::check() && Auth::user()->role_id == 2) {
            return $next($request);
        }
        else {
            return redirect('/');
        }
    }
}
```

In order the middleware to be accesible by the alias `admin`, go into `app/Http/Kernel.php` and add that alias:

```
protected $middlewareAliases = [
...

'admin' => \App\Http\Middleware\Admin::class
];
```

Authorization by Policies

Even more, sometimes an action can or cannot be made by an specific user. We could use [Policies](Authorization - Laravel - The PHP Framework For Web Artisans) for that. **AGAIN, EVEN IF YOU CONTROL WHICH ACTIONS ARE CLICKABLE BY AN SPECIFIC ROLE ON THE FRONTEND, YOU SHOULD VALIDATE THAT ON THE BACKEND, because actually requests can be emulated by anyone.**

Launch this command on the terminal:

```
php artisan make:policy PeticionePolicy --model=Peticione
```

If you open the file created on `app/Policies/PeticionePolicy.php` there you will find the actions that give access or not to `Peticione` model.

There we can define, for instance, that in case the user has an admin role, we return true and give always access, or in case a peticione belongs to an specific user we can give to that user permission to make an update. These are only examples and policies can vary a lot depending on the application logic you want to implement.

```php
class PeticionePolicy
{
    public function before(User $user, string $ability)
    {
        if( $user->role_id==2){
            return true;
        }
    }

    /**
     * Determine whether the user can view the model.
     */
    public function view(User $user, Peticione $peticione): bool
    {
        return true;
    }

    /**
     * Determine whether the user can create models.
     */
    public function create(User $user): bool
    {
        return true;
    }

    /**
     * Determine whether the user can update the model.
     */
    public function update(User $user, Peticione $peticione): bool
    {
        if($user->role_id==1 && $peticione->user_id=$user->id){
            return true;
        }
        return false;
    }

    /**
     * Determine whether the user can delete the model.
     */
    public function delete(User $user, Peticione $peticione): bool
    {
        if($user->role_id==1 && $peticione->user_id=$user->id){
            return true;
        }
        return false;
    }
```

Once you have define your policy you have to include it on the `app/Providers/AuthServiceProvider` class like this:

```
    protected $policies = [
        Peticione::class => PeticionePolicy::class,
    ];
```

You will have to follow these steps in order to include any other policy you need on your application.