

Autenticación de Usuarios y Roles en Laravel 5.5



Claudio Vallejo [Follow](#)

Sep 10, 2017 · 4 min read

Me he inspirado en el extraordinario post publicado por [everton zp](#) y luego el repositorio de [Kali Dass](#) en github (<https://github.com/karoys/laravel-native-roles-auth>). Esta guía es una traducción al español y actualización de las bases referidas anteriormente.

Partamos realizando una nueva **copia de Laravel 5.5**:



Fuente de la imagen: <https://laravel-news.com/laravel-5-5>

```
$ laravel new proyecto_rols
```

Luego ejecutar el comando `php artisan make:auth` para crear el recurso de autenticación Auth:

```
$ php artisan make:auth
```

Crear el modelo **Role** con su respectiva migración (parámetro `-m`):

```
$ php artisan make:model Role -m
```

. . .

Editar la clase *CreateRolesTable* en la carpeta de migrations:

```
public function up()
{
    Schema::create('roles', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->string('description');
        $table->timestamps();
    });
}

public function down()
{
    Schema::dropIfExists('roles');
}
```

. . .

Crear una nueva migración para la tabla dinámica *role_user*:

```
$ php artisan make:migration create_role_user_table
```

Editar la clase **CreateRoleUserTable** en la carpeta database/migrations:

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

public function down()
{
    Schema::dropIfExists('role_user');
}
```

. . .

Ahora hay que generar una relación *many-to-many* entre el **User** y el **Role**.

Abrir el modelo **User.php** y agregar el siguiente método:

```
public function roles()
{
    return $this
        ->belongsToMany('App\Role')
        ->withTimestamps();
}
```

Hacer lo mismo con el modelo **Role.php**:

```
public function users()
{
    return $this
        ->belongsToMany('App\User')
        ->withTimestamps();
}
```

. . .

Es momento de crear algunos seeders y agregar roles y usuarios en la base de datos:

```
$ php artisan make:seeder RoleTableSeeder

$ php artisan make:seeder UserTableSeeder
```

Editar la clase ***RoleTableSeeder*** (se encuentra dentro de la carpeta: database/seeds/) agregando el siguiente código al método ***run***:

```
use Illuminate\Database\Seeder;
use App\Role;

class RoleTableSeeder extends Seeder
{
    public function run()
    {
        $role = new Role();
        $role->name = 'admin';
        $role->description = 'Administrator';
        $role->save();

        $role = new Role();
        $role->name = 'user';
        $role->description = 'User';
        $role->save();
    }
}
```

Hacer lo mismo con la clase ***UserTableSeeder***:

```
use Illuminate\Database\Seeder;
use App\User;
use App\Role;

class UserTableSeeder extends Seeder
{
    public function run()
    {
        $role_user = Role::where('name', 'user')->first();
        $role_admin = Role::where('name', 'admin')->first();
    }
}
```

```

        $user = new User();
        $user->name = 'User';
        $user->email = 'user@example.com';
        $user->password = bcrypt('secret');
        $user->save();
        $user->roles()->attach($role_user);

        $user = new User();
        $user->name = 'Admin';
        $user->email = 'admin@example.com';
        $user->password = bcrypt('secret');
        $user->save();
        $user->roles()->attach($role_admin);
    }
}

```

Editar la clase **DatabaseSeeder** (ubicada en la carpeta: database/seeds/) agregándole el siguiente código al método **run**:

```

public function run()
{
    // La creación de datos de roles debe ejecutarse primero
    $this->call(RoleTableSeeder::class);

    // Los usuarios necesitarán los roles previamente
    generados
    $this->call(UserTableSeeder::class);
}

```

. . .

Ya queda poco... está casi todo listo. Ahora es tiempo de correr las migraciones y generar el contenido:

Hay que asegurarse de tener definidas las variables de entorno en nuestro archivo .env relativas a la base de datos que utilizaremos

```
$ php artisan migrate:refresh --seed
```

. . .

Abrir el modelo **User.php** y agregar estos tres pequeños métodos:

```
public function authorizeRoles($roles)
{
    if ($this->hasAnyRole($roles)) {
        return true;
    }
    abort(401, 'Esta acción no está autorizada.');
```

```
public function hasAnyRole($roles)
{
    if (is_array($roles)) {
        foreach ($roles as $role) {
            if ($this->hasRole($role)) {
                return true;
            }
        }
    } else {
        if ($this->hasRole($roles)) {
            return true;
        }
    }
    return false;
}
```

```
public function hasRole($role)
{
    if ($this->roles()->where('name', $role)->first()) {
        return true;
    }
    return false;
}
```

. . .

Abrir el archivo `app/Http/Controllers/Auth/RegisterController.php` y cambiar el método **create()** para definir por defecto el Role para los nuevos Users:

```
use App\Role;

class RegisterController ...

protected function create(array $data)
{
    $user = User::create([
```

```

        'name' => $data['name'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
    ]);

    $user
        ->roles()
        ->attach(Role::where('name', 'user')->first());

    return $user;
}

```

. . .

Por último el paso final. Ahora, todo lo que se necesita es llamar al método `authorizeRoles()` dentro de la acciones del controlador y pasar el array con los roles de usuario y el nivel de acceso que se desee.

```

class HomeController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index(Request $request)
    {
        $request->user()->authorizeRoles(['user', 'admin']);

        return view('home');
    }

    /**
     * public function someAdminStuff(Request $request)
     * {
     *     $request->user()->authorizeRoles('admin');
     *
     *     return view('some.view');
     * }
     */
}

```

. . .

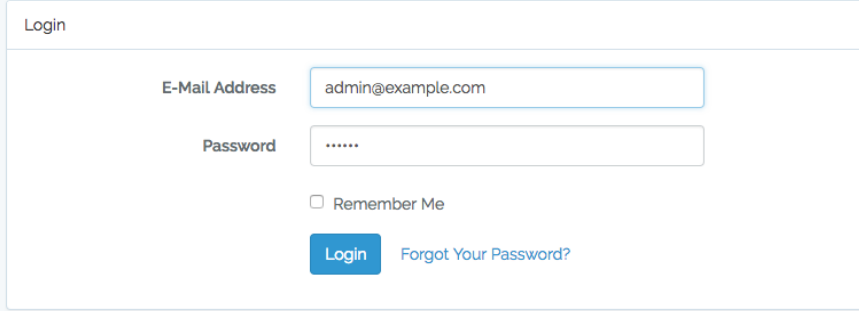
Opcional:

Define parámetros de selección para la vista Home (resources/views/):

```
@if(Auth::user()->hasRole('admin'))  
    <div>Acceso como administrador</div>  
@else  
    <div>Acceso usuario</div>  
@endif  
  
You are logged in!
```

Si utilizas valet puedes visualizar muy fácilmente tu nuevo proyecto a través de la url: http://proyecto_rols.dev/

Al estar ambos usuarios ya creados, puedes ingresar como administrador:



Login

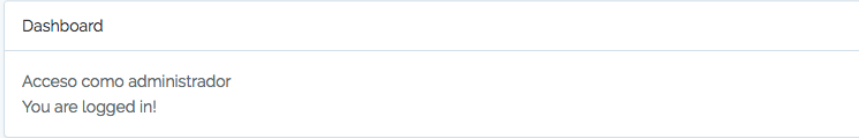
E-Mail Address

Password

☐ Remember Me

[Forgot Your Password?](#)

la contraseña es secret



Dashboard

Acceso como administrador
You are logged in!

O como un usuario:

Login

E-Mail Address

user@example.com

Password

☐ Remember Me

Login

[Forgot Your Password?](#)

contraseña secret

Dashboard

Acceso usuario

You are logged in!

. . .

Después de este punto es factible proceder con el flujo normal de desarrollo. Construir una interface CRUD para manejar los roles y asignaciones para usuarios.

. . .

Puedes seguirme también en twitter:

Claudio Vallejo (@cvallejo) | Twitter

The latest Tweets from Claudio Vallejo (@cvallejo).
Wine, programmer, diver & photographer lover....
[twitter.com](https://twitter.com/cvallejo)



