**Les Policies**

Les policies permettent de définir certains rôles aux administrateurs ou aux utilisateurs pour des tâches particulières; cela peut être considéré comme une autre couche de sécurité.

Etapes à suivre:

**Création d’une migration:**

**php artisan make:migration AddRoleToUsers**

Ensuite on rajoute la column **‘rule’** à notre migration et on fait la migration. puis on on crée la **attachée** au model:

**php artisan make:policy PropertyPolicy --model=Property**

on rajoute **--model=Property** pour dire au framework que cette policy est liée au modèle **Property.**

Implémentation de la policy:

On retourne true pour toutes les autres méthode, mais en ce qui concerne la méthode **delete(), restore(), forceDelete(),**  on fait :

**return $user->rule==='admin';**

Puis on enregistre cette policy dans le **AuthServiceProvider**

protected $policies = [

Property::class => PropertyPolicy::class

// 'App\Models\Model' => 'App\Policies\ModelPolicy',

];

NB: Laravel peut aussi découvrir automatiquement les policies associées à un modèle mais pour cela il faut certaines politiques de nommage. Pour cela, il va falloir regarder dans la documentation

Maintenant, si on veut vérifier si l’utilisateur à le droit d’éditer un bien en particulier, on peut utiliser soit la méthode de **Gates** ou une méthode particulière: exemple en ci-dessous.

public function edit(Property $property)

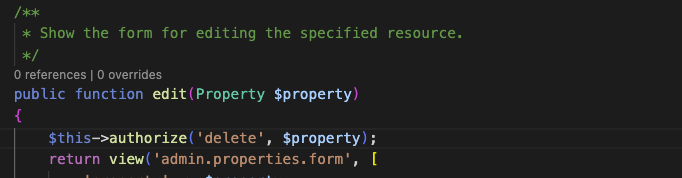
{

dd(Auth::user()->can('delete', $property));

return view('admin.properties.form', [

On lui passe également la **propriété** en paramètre.

Une autre fonction que l’on peut utiliser est la fonction **authorise**; on l’utilise seulement si notre policy hérite de notre controller. Exemple ci-dessous.



Autre cas : dans le cas d’une ressource comme la nôtre, nous allons implémenter le **constructeur** dans notre controller afin que cela puisse être effectif pour tous les **users.**

**Policy pour les images**

On commence par créer une policy pour les images:

**php artisan make:policy PicturePolicy --model=Picture**

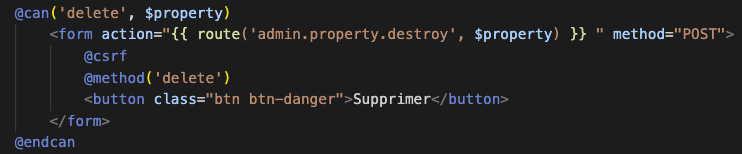
Après avoir créé la **policy**, on implémente notre **PolicyPicture** comme pour la property.

* Dans ce nouveau cas nous implémenterons la règle de policy directement sur nos routes

Nous pouvons le faire de 2 manières:

* La 1ère en utilisant le mot clé middleware
* Et la deuxième méthode plus simple sans utiliser le middleware

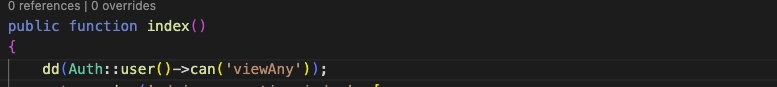
Et une dernière chose à faire, c’est au niveau des vues, ne pas afficher de bouton supprimer si l’utilisateur n’est pas **admin**; dans notre vue index, nous aurons:



Exemple de tout ce qu'on peut faire avec cela.



Une dernière chose, nous allons gérer le cas de **viewAny(),** vu que ce n’est racheter à aucun **model.**

Si on débug cela dans notre **controller**, ça sera **false**, et ce, même si on est **admin. **

Ci-dessous le résultat dans le navigateur.

Mais en lui passant **Property::class**, dans la méthode **can()**, cela reviendra à **true,** vu que n’importe qui peut voir les biens.

**?? À REGARDER DANS LA DOCUMENTATION :**

* Gate response
* Policy Filter qui est très intéressant dans un cas où l’utilisateur est **super admin** et à le droit sur tout.

**NB:**

Nous pouvons aller en profondeur avec cela et faire en sorte que l'utilisateur n’ait le droit d’éditer les biens dont il en est l’auteur seulement mais en étant super admin, il pourra éditer tout bien quelconque.