# TP n°10 - Les autorisations


| | | | |
|---|---|---|---|
| Site: | Université de Reims Champagne-Ardenne | Imprimé par: | CYRIL RABAT |
| Cours: | INFO0303 - Technologies Web 2 | Date: | mercredi 15 novembre 2023, 14:18 |
| Livre: | TP n°10 - Les autorisations | | |

# Table des matières

# 1. Les gates et les policies

Dans le TP précédent, nous avons géré l'authentification des utilisateurs. Nous avons même restreint des fonctionnalités aux utilisateurs connectés. Dans un système plus complexe, il peut être nécessaire de gérer des droits de manière plus fine. Pour illustrer avec un exemple simple, nous allons modifier l'application développée dans les TP précédents (vous pouvez récupérer la version obtenue à la fin du TP précédent sur le *gitlab*) : les fournisseurs (*suppliers*) sont maintenant associés à des utilisateurs qui seront les seuls à pouvoir ajouter/modifier/supprimer des articles pour ce fournisseur.

*Laravel* fournit deux éléments pour gérer les autorisations : la *gate* permet un contrôle d'accès macroscopique alors que les *policies* permettent de grouper la logique autour d'un modèle ou d'une ressource.

## 1.1. Mise-à-jour du projet précédent

Pour spécifier les utilisateurs qui peuvent éditer les articles d'un fournisseur (nous les appellerons les éditeurs), nous créons une table pivot entre les tables `users` et `suppliers`.

1. Créez la migration permettant de créer la table pivot (basez vous sur la migration qui a permis d'ajouter les catégories aux articles).

2. Modifiez les deux modèles concernés : une méthode `editors` dans `Supplier` et `suppliers` dans `User`. qui permettront respectivement de récupérer les éditeurs du fournisseur et la liste des fournisseurs pour lesquels l'utilisateur est un éditeur.

3. Créez un *seeder* pour les utilisateurs (modèle `User`) et utilisez la *factory* existante pour créer 10 utilisateurs aléatoires (à noter que le mot de passe est toujours "password"). Ajoutez également un utilisateur par défaut pour simplifier l'utilisation. Pour cela, vous pouvez utiliser le code suivant :

```
$user = User::create([
    'name' => 'Chuck Norris',
    'email' => 'chuck@norris.fr',
    'email_verified_at' => now(),
    'password' => Hash::make('password'),
    'remember_token' => Str::random(10),
  ]);
```

N'oubliez pas d'ajouter les instructions `use` nécessaires :

```
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;
```

4. Dans le *seeder* des fournisseurs, ajoutez un éditeur aléatoire pour chaque fournisseur. Vous pouvez exploiter le code utilisé pour associer des catégories aux articles (dans le *seeder* des articles).

5. N'oubliez pas d'ajouter l'exécution du seeder des utilisateurs dans la classe `DatabaseSeeder` (en première position).

6. Dans le *seeder* des fournisseurs, ajoutez les instructions suivantes pour donner le rôle d'éditeur pour tous les fournisseurs à votre utilisateur :

```
$chuck = User::where('name', '=', 'Chuck Norris')->first();
$chuck->suppliers()->sync(Supplier::all());
```

7. Vous pouvez lancer la migration avec le peuplement.

## 1.2. Création et utilisation d'une gate

Pour le moment, les utilisateurs connectés peuvent tous modifier les articles, en créer, etc. Pour illustrer le fonctionnement des *gates*, nous allons en créer une pour savoir si un utilisateur peut mettre à jour un article.

1. Ouvrez la classe `App\Providers\AuthServiceProvider` et dans la méthode `boot`, ajoutez le code suivant :

```
Gate::define('item-editor', function (User $user, Item $item) {
    return $user->suppliers->contains($item->supplier);
 });
```

2. Modifiez la méthode `edit` du contrôleur des articles :

   - Modifiez le paramètre `$string $id` par `Item $item` (*Laravel* est capable de charger automatiquement le modèle)
   - Remplacez tout le code par le suivant :

```
if(!Gate::allows('item-editor', $item)) {
  abort(403);
}

return view('items.edit', [
    'item' => $item,
    'suppliers' => Supplier::orderBy('name', 'asc')->get(),
    'categories' => Category::orderBy('title', 'asc')->get()
  ]);
```

La première instruction permet de rediriger l'utilisateur vers la page correspondant à l'erreur 403 (accès non autorisé) si la *gate* n'est pas vérifiée.

3. Testez le fonctionnement de l'application : connectez-vous avec un utilisateur quelconque (vérifiez qu'il est bien éditeur d'au moins un fournisseur) et cliquez sur l'édition d'un article pour lequel il n'a pas l'accès et un autre pour lequel il a un accès.

Il est possible de rediriger l'utilisateur et lui afficher un message d'erreur plutôt que de le rediriger vers la page d'erreur 403.

4. Modifiez le *template* et insérez le code ci-dessous avant l'instruction *Blade* `@yield('content')` par celui-ci :

```
@if(session('error'))
  <div class="alert alert-danger">
    {{ session('error') }}
  </div>
@endif
```

5. Modifiez le code dans la méthode `edit` du contrôleur par celui-ci :

```
if(!Gate::allows('item-editor', $item)) {
  return redirect('/item')->with('error', "Vous n'êtes pas autorisé à modifier cet article.");
}
```

La méthode `with` permet de passer une valeur en session qui est détruite dès qu'elle est utilisée (si le message est affiché et que vous rafraîchissez la page, il n'apparait plus). Vous pouvez l'utiliser pour passer d'autres types de messages. De même, vous pouvez utiliser une bibliothèque *JavaScript* pour afficher des *popups*.

Pour le moment, les boutons sont affichés alors qu'ils ne devraient l'être que pour les articles éditables. Avec *blade*, nous pouvons utiliser les instructions *@can*. Ici, comme nous générons les boutons lors de l'appel AJaX (méthode `index` du contrôleur), il suffit de modifier le test :

6. Dans la méthode `index` du contrôleur remplacez le test suivant :

```
if(Auth::check()) {
  ...
}
```

par

```
if(Gate::allows('item-editor', Item::find($row['id']))) {
  ...
}
```

## 1.3. Création et utilisation d'une policy

Comme dit précédemment, les *gates* peuvent être compliquées à utiliser si nous devons en créer plusieurs pour chaque modèle (droits de lecture, d'édition, de suppression). *Laravel* propose d'utiliser des *policies*.

1. Créez une nouvelle *policy* pour les articles à l'aide de la commande suivante :

```
php artisan make:policy ItemPolicy --model=Item
```

Un nouveau fichier `ItemPolicy` a été créé dans le répertoire `app\Policies`. Il contient une méthode pour chaque action possible vis-à-vis du modèle spécifié. Il ne reste qu'à renseigner chaque méthode.

2. Placez cette instruction dans la méthode `update` de la classe `ItemPolicy` (n'oubliez pas les instructions `use` pour les deux modèles) :

```
return $user->suppliers->contains($item->supplier);
```

3. Dans la classe `AuthServiceProvider`, vous devez renseigner la *policy* dans l'attribut `policies` :

```
protected $policies = [
    Item::class => ItemPolicy::class,
  ];
```

4. Commentez l'instruction que nous avions ajoutée dans la méthode `boot` (la *gate* est maintenant inutile).

5. Dans la méthode `index` du contrôleur, le test suivant :

```
if(Gate::allows('item-editor', Item::find($row['id'])))
```

devient :

```
if(Auth::check() && Auth::user()->can('update', Item::find($row['id'])))
```

6. Dans la méthode `update`, ajoutez comme premier paramètre `Request $request` et modifiez l'instruction suivante :

```
if(!Gate::allows('item-editor', $item)) {
  return redirect('/item')->with('error', "Vous n'êtes pas autorisé à modifier cet article.");
```

par :

```
if($request->user()->cannot('update', $item))
  return redirect('/item')->with('error', "Vous n'êtes pas autorisé à modifier cet article.");
```

Pour créer un nouvel article, il faut que l'utilisateur soit un éditeur, autrement dit, il faut qu'il soit associé à au moins un fournisseur.

7. Spécifiez la méthode `create` de la classe `ItemPolicy` puis ajoutez le code suivant dans la méthode `create` de la classe `ItemController` (ici, il n'y a pas d'article donc le paramètre est la classe correspondant au modèle) :

```
if($request->user()->cannot('create', Item::class))
  return redirect('/item')->with('error', "Vous n'êtes pas autorisé à créer un article.");
```

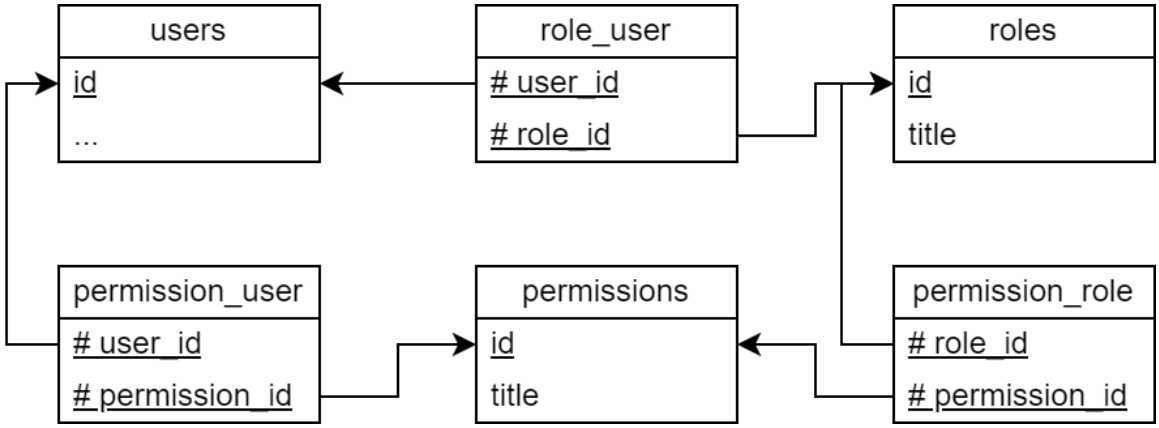
8. Pour cacher le bouton de création pour les utilisateurs non-éditeurs, ajoutez les annotations *Blade* suivantes autour du bouton de création dans la vue `list.blade.php` :

```
@can('create', App\Models\Item::class)
...
@endcan
```

9. Complétez les classes `ItemController` et `ItemPolicy` pour empêcher la suppression d'un article dont l'utilisateur n'est pas un éditeur du fournisseur correspondant.

## 2. Les rôles et les permissions

Il est parfois nécessaire de définir des rôles différents pour les utilisateurs d'une application, ainsi que les permissions associées. Il est bien entendu possible de réaliser cette gestion manuellement comme le montre le MPD ci-dessous. Un utilisateur peut être associé à plusieurs rôles, les permissions peuvent être associées à plusieurs rôles et des utilisateurs peuvent être associés à des permissions indépendamment de leur rôle.




L'accès aux sections peut alors être restreint en fonction d'un rôle de l'utilisateur ou d'une ou plusieurs permissions.

Une autre solution est d'utiliser un composant (*package*) tel que celui proposé par spatie. Il est très complet et existe maintenant depuis plusieurs années. Nous allons présenter ici quelques éléments (n'hésitez pas à consulter la documentation pour plus d'informations).

## 2.1. Installation du composant

Comme tout composant, il faut l'installer grâce à *composer*.

1. Placez-vous dans le répertoire de l'application puis tapez la commande suivante :

```
composer require spatie/laravel-permission
```

2. Publiez-le à l'aide de la commande suivante :

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

3. Lancez la migration pour créer les tables.

Le composant propose le trait `HasRoles`.

4. Ajoutez le trait dans le modèle `User` et n'oubliez pas l'instruction `use` :

```
use Spatie\Permission\Traits\HasRoles;
```

Vous trouverez dans la documentation l'ensemble des méthodes apportées (vous pouvez également ouvrir le trait qui est situé dans le répertoire `vendor/spatie/laravel-permission/src/Traits`).

## 2.2. Spécifier des rôles et des permissions

La création des rôles et des permissions est généralement effectuée dans un *seeder*. Dans notre cas, nous allons juste créer le rôle d'éditeur et trois permissions associées (création, édition et suppression des articles).

Certains éléments de code qui ont été écrits dans le chapitre précédent devront être supprimés pour éviter des conflits entre les différentes gestions (*gate*/*policies* et rôles/permissions)

1. Créez un *seeder* nommé `RoleSeeder` et renseignez-le dans la classe `DatabaseSeeder`.

Les classes `Role` et `Permission` possèdent une méthode `create` qui prend en paramètre un tableau avec une clé `name` et la valeur associée (par exemple `[ "name" => "editor" ]` pour le rôle). Sur le rôle, on peut utiliser la méthode `syncPermissions` qui prend un tableau d'objets `Permission`.

2. Dans le *seeder*, créez un rôle nommé *editor* et les trois permissions nommées "create item", "edit item" et "delete item".

3. Associez ces permissions au rôle *editor*.

4. Dans le *seeder* des utilisateurs, associez le rôle *editor* à Chuck Norris (ou à l'utilisateur que vous aviez créé dans le chapitre précédent).

5. Lancez le peuplement de la base (ou tout le processus de migration).

6. Ouvrez la base de données (avec *phpMyAdmin* ou *HeidiSQL*) et vérifiez que le rôle et les permissions ont été créés, et que le rôle *editor* a été associé à Chuck Norris.

## 2.3. Restriction des accès

Maintenant que les rôles et les permissions ont été créés, ils sont disponibles directement dans le code. Bien entendu, si vous avez créé des *gates* et des *policies*, il n'est pas nécessaire de les remplacer (elles peuvent coexister avec les rôles et les permissions). Ici, l'objectif est d'illustrer le fonctionnement du composant.

1. Dans la vue `list.blade.php`, remplacez la directive *Blade* qui permet d'afficher de manière conditionnelle le bouton d'ajout d'un article :

```
@can('create', App\Models\Item::class)
```

par :

```
@can('create item')
```

2. Dans la méthode `create` de la classe `ItemPolicy`, remplacez l'instruction :

```
return $user->suppliers()->get()->count() > 0;
```

par :

```
return $user->can('create item');
```

3. Vérifiez que votre code est toujours fonctionnel.

L'application est loin d'être complète : il reste à gérer les fournisseurs et les utilisateurs. Mais l'ajout des droits peut être réalisé dynamiquement lors de la modification d'un fournisseur, par exemple.