MINI-CRM

Walkthrough complet - TP8

Autorisations et gestion des droits d’accès

Projet : mini-crm de BUT2

Objectif : distinguer l’utilisateur connecté de l’utilisateur autorisé et centraliser les règles dans des Policies.

1. Objectifs et matrice des droits

Deux rôles sont utilisés : l'administrateur possède tous les droits et l'utilisateur standard peut consulter, créer et modifier, mais ne peut pas supprimer.

Action	Administrateur	Utilisateur
Voir un client / produit	Autorisé	Autorisé
Créer	Autorisé	Autorisé
Modifier	Autorisé	Autorisé
Supprimer	Autorisé	Refusé
Page Administration	Autorisé	Refusé

2. Colonne is_admin et modèle User

La migration ajoute `is_admin` avec la valeur par défaut `false`. Le modèle User autorise le remplissage contrôlé de cet attribut et le convertit automatiquement en booléen.

Extrait / commande

```
$table->boolean('is_admin')->default(false);

'is_admin' => 'boolean',
```

Le seeder utilise `updateOrCreate` pour pouvoir être rejoué sans créer de doublons.

3. Policies Laravel

Les classes `ClientPolicy` et `ProductPolicy` centralisent les règles. Laravel les découvre automatiquement grâce au nom du modèle et au suffixe Policy.

Extrait / commande

```
public function update(User $user, Client $client): bool
{
    return true;
}

public function delete(User $user, Client $client): bool
{
    return $user->is_admin;
}
```

La Policy répond à la question « cet utilisateur peut-il effectuer cette action sur cette ressource ? ». Elle évite de recopier des conditions dans chaque contrôleur.

4. Autoriser avant d'écrire

Les contrôleurs utilisent `Gate::authorize` avant les opérations sensibles. Une action refusée déclenche automatiquement une réponse HTTP 403 et aucune suppression n'est effectuée.

Extrait / commande

```
Gate::authorize('delete', $client);
$client->delete();
```

Les actions `create`, `view` et `update` sont également vérifiées. Le middleware `auth` identifie d'abord l'utilisateur ; la Policy vérifie ensuite son droit.

5. Adapter l'interface avec @can

Les boutons Modifier et Supprimer sont entourés de directives Blade `@can`. Marie ne voit donc pas l'action Supprimer, tandis que Jean la voit.

Extrait / commande

```
@can('delete', $client)
    <form method="POST" action="{{ route('clients.destroy', $client) }}">
        @csrf
        @method('DELETE')
        <button type="submit">Supprimer</button>
    </form>
@endcan
```

Masquer un bouton améliore l'expérience, mais ne constitue pas la sécurité principale : la protection réelle reste `Gate::authorize` dans le contrôleur.

6. Bonus : page Administration

La page `/admin` est protégée par un Gate global `viewAdmin`. Elle est accessible à Jean et renvoie 403 à Marie.

Extrait / commande

```
Gate::define('viewAdmin', fn (User $user): bool => $user->is_admin);

Route::view('/admin', 'admin')
    ->middleware('can:viewAdmin')
    ->name('admin');
```

7. Vérification et tests

La suite couvre la matrice des Policies, les suppressions interdites et autorisées, le masquage des boutons et la page d'administration.

Extrait / commande

```
php artisan migrate --force
php artisan db:seed --class=UserSeeder --force
php artisan test --compact
PHPUnit: 20 tests, 175 assertions, tous réussis
```

Test manuel conseillé : se connecter avec Marie, vérifier que les boutons Supprimer sont absents et appeler directement une URL DELETE : Laravel répond 403. Refaire avec Jean : la suppression est autorisée.

Architecture finale : requête -> middleware auth -> validation Form Request -> Policy / Gate -> contrôleur -> Eloquent -> SQLite.

Conclusion : le TP8 ajoute une autorisation centralisée, testée et visible dans l'interface, tout en conservant les protections côté serveur.