Autorisation- Laravel 9:

1. Introduction

Laravel fournit un moyen simple d'autoriser les actions des utilisateurs sur des ressources spécifiques. Avec l'autorisation, vous pouvez autoriser de manière sélective les utilisateurs à accéder à certaines ressources tout en refusant l'accès aux autres. Laravel fournit une API simple pour gérer les autorisations utilisateur à l'aide de `Gates` et de `Policies`. `Gates` `AuthServiceProvider` approche simple de l'autorisation basée sur la fermeture à l'aide de `AuthServiceProvider` tandis que les `Policies` vous permettent d'organiser la logique d'autorisation autour des modèles utilisant des classes.

2. Création d'une politique (Policy)

Les **policies** (politiques) sont des classes qui vous aident à organiser la logique d'autorisation autour d'une ressource de modèle. En utilisant notre exemple précédent, nous pourrions avoir un CarPolicy qui gère l'accès des utilisateurs au modèle de Car. Pour rendre CarPolicy, laravel fournit une commande artisan. Il suffit d'exécuter la commande :

```
php artisan make:policy CarPolicy
```

Cela fera une classe de politique vide et placera dans le dossier **app/Policies**. Si le dossier n'existe pas, Laravel le créera et placera la classe à l'intérieur. Une fois créée, les stratégies doivent être enregistrées pour aider Laravel à savoir quelles politiques utiliser lors de l'autorisation d'actions sur des modèles. AuthServiceProvider de Laravel, fourni avec toutes les nouvelles installations Laravel, possède une propriété $policies qui lie vos modèles Eloquent à leurs règles d'autorisation. Tout ce que vous devez faire ajoute le mappage au tableau.

```
protected $policies = [ Car::class => CarPolicy::class ];
```

3. Ecriture de la politique :

L'écriture de **Policies** la forme comme celle-ci:

```
 function view(User $user, Car $car) {
return $user->id==$car->user_id;
}
```

La politique peut contenir plus de méthodes que nécessaire pour prendre en charge tous les cas d'autorisation d'un modèle.

4. Autoriser des actions en utilisant une politique (Policy):

a. Via le contrôleur

Laravel fournit une méthode **authorize** utile à tous vos contrôleurs qui étendent la classe de base **App\Http\Controllers\Controller**.

Comme la méthode **can**, cette méthode accepte le nom de l'action que vous souhaitez autoriser et le modèle correspondant. Si l'action n'est pas autorisée, la méthode **authorize** lèvera une exception **Illuminate\Auth\Access\AuthorizationException** que le gestionnaire d'exceptions de Laravel convertira automatiquement en une réponse HTTP avec un code d'état 403 :

```
pubic function show(Car $car) {
$this->authorize('view', $car);
 /* user can view content */
 }
```

- **Actions qui ne nécessitent pas de modèle**

Comme nous l'avons vu précédemment, certaines méthodes de politique, comme **create**, ne nécessitent pas d'instance de modèle. Dans ces situations, vous devez transmettre un nom de classe à la méthode **authorize**. Le nom de la classe sera utilisé pour déterminer la politique à utiliser lors de l'autorisation de l'action :

```
public function create(Request $request)
{
    $this->authorize('create', Car::class);
```

```
    // The current user can create content...
}
```

b. Via le modèle Blade

Lorsque vous rédigez des modèles Blade, vous pouvez souhaiter afficher une partie de la page uniquement si l'utilisateur est autorisé à effectuer une action donnée. Par exemple, vous pouvez souhaiter afficher un formulaire de mise à jour pour un article de blog uniquement si l'utilisateur peut effectivement mettre à jour l'article. Dans ce cas, vous pouvez utiliser les directives **@can** et **@cannot :**

```
@can('update', $post)
    <!-- The current user can update the post... -->
@elsecan('create', App\Models\Post::class)
    <!-- The current user can create new posts... -->
@else
    <!-- ... -->
@endcan

@cannot('update', $post)
    <!-- The current user cannot update the post... -->
@elsecannot('create', App\Models\Post::class)
    <!-- The current user cannot create new posts... -->
@endcannot
```

- **Actions qui ne nécessitent pas de modèle**

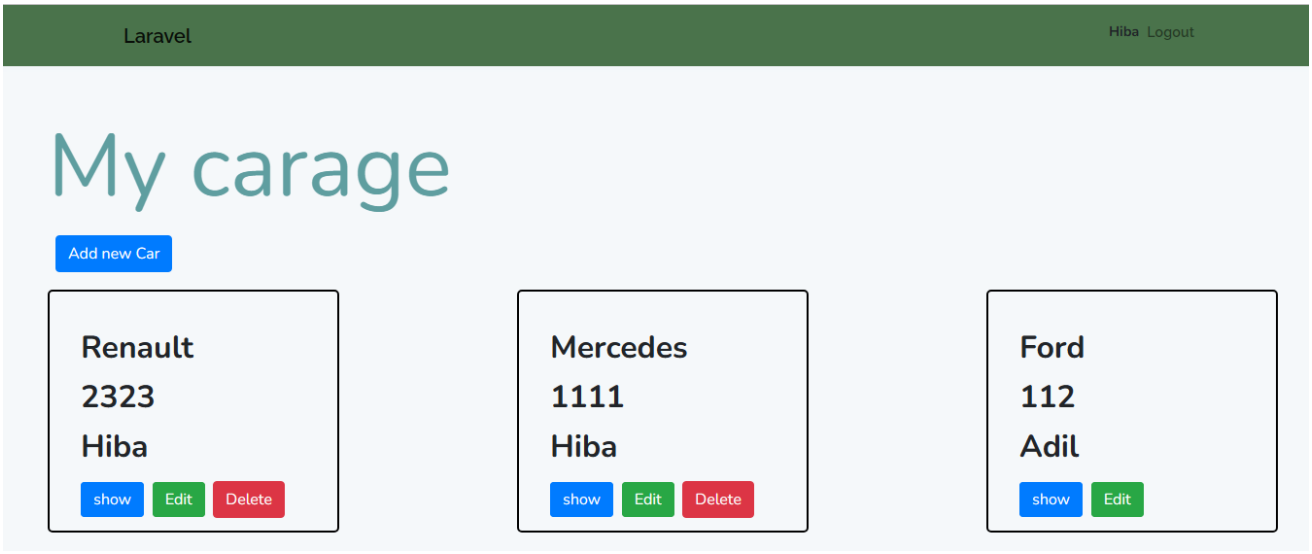
Comme la plupart des autres méthodes d'autorisation, vous pouvez passer un nom de classe aux directives **@can** et @**cannot** si l'action ne nécessite pas d'instance de modèle :

```
@can('create', App\Models\Content::class)
    <!-- The current user can create contents... -->
@endcan

@cannot('create', App\Models\Content::class)
    <!-- The current user can't create contents... -->
@endcannot
```

Travail à faire:

Dans cette section, nous allons créer une politique pour le modèle Car qui sera utilisée pour autoriser toutes les actions CRUD.



Etape 1 : Créer les modèles

La commande artisan de Laravel est votre meilleur ami lorsqu'il s'agit de créer du code Vous pouvez utiliser les commandes artisan suivante pour créer un modèle Car.

```
php artisan make:model Car -mcr
```

app/Models/Car

```php
<?php

namespace App\Models;

use App\Models\User;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Car extends Model
{
    use HasFactory;
    public function user(){
```

```
        return $this->belongsTo(User::class);

       }
       protected $fillable = [
        'brand',
        'price',
        'picture',
        'origin'
    ];
}
```

app/Models/User

```
<?php

namespace App\Models;

use App\Models\Car;
use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;
    public function cars(){
        return $this->hasMany(Car::class);
    }

    protected $fillable = [
        'name',
        'email',
        'password',
    ];


    protected $hidden = [
        'password',
        'remember_token',
    ];


    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
```

```
}
```

Etape 2 : Ajouter une colonne à la table des utilisateurs (users)

Pour que les choses restent simples et agréables, commençons par attribuer à un utilisateur le statut d'administrateur. Les administrateurs seront en mesure de voir du contenu supplémentaire et d'effectuer des tâches additionnelles dans l'application. Pour mettre en place cette fonctionnalité, nous allons ajouter une colonne booléenne à la table des utilisateurs, **isAdmin**. Ajoutez ce qui suit à votre fichier de migration des utilisateurs :

```
php artisan make:migration add_column_isAdmin_table_users --table users
```

app/database/migrations/2023_XXXx_add_column_isAdmin_table_users.php.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->boolean('isAdmin')->default(0);
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('is_Admin');
```

```
        });
    }
};
```

Exécutez maintenant la migration :

```
php artisan migrate
```

id	name	email	email_verified_at	password	remember_token	created_at	updated_at	isAdmin
24	Hiba	hiba@gmail.com	2023-01-29 15:27:36	$2y$10$D3XxhH4tcjCenfGq228gwOFMRY2lJ1sS41mvh7y6FLJ...	*NULL*	2023-01-29 15:27:16	2023-01-29 15:27:36	1
25	Adil	adil@gmail.com	2023-01-31 09:54:45	$2y$10$R.EjUdpMnGwPkLlz3Fgx3O7Eg9z4tuldz9tQ31rDWpD...	*NULL*	2023-01-31 09:54:01	2023-01-31 09:54:45	0

id	brand	price	origin	picture	user_id	created_at	updated_at
1	Renault	2323	French	cars/DpJp7nrLamZtGIOBEOYNYc4SyKbzDJntwZyOldkW.png	24	2023-01-31 09:46:14	2023-02-01 19:49:52
2	Mercedes	1111	Germany	cars/5z3D3KlsOm67fbGBwU7u1yTtwD0u4AaCbBPiFLWk.jpg	24	2023-01-31 09:51:15	2023-02-01 19:44:43
3	Ford	112	USA	cars/DQUxHeuX8eCOiVaWBlRa55eOpbtLAatEx6u2lfie.png	25	2023-01-31 09:57:15	2023-02-01 20:00:00

Etape 3 : Création de *Policie*

```
php artisan make:policy CarPolicy --model=Car
```

Comme vous pouvez le voir, nous avons fourni l'argument **--model=Car** pour qu'il crée toutes les méthodes CRUD. En l'absence de cela, il créera une classe de politique vierge. Vous pouvez localiser la classe de politique nouvellement créée à :

app/Policies/CarPolicy.php.

```
<?php

namespace App\Policies;

use App\Models\Car;
```

```php
use App\Models\User;
use Illuminate\Auth\Access\HandlesAuthorization;

class CarPolicy
{
    use HandlesAuthorization;


    public function before(User $user,$ability){
        if($user->isAdmin and $ability!='delete')
        return true;
    }
    public function viewAny(User $user)
    {
        //
    }

    public function view(User $user, Car $car)
    {
        return $user->id===$car->user_id;
    }


    public function create(User $user)
    {
       return true;
    }

    public function update(User $user, Car $car)
    {
        return $user->id===$car->user_id;
    }


    public function delete(User $user, Car $car)
    {
        return $user->id===$car->user_id;
    }


    public function restore(User $user, Car $car)
    {
        //
    }

    public function forceDelete(User $user, Car $car)
    {
        //
    }
```

```
}
```

Pour pouvoir utiliser notre classe de politique, nous devons l'enregistrer en utilisant le fournisseur de services de Laravel, comme le montre l'extrait suivant.

```php
<?php

namespace App\Providers;

// use Illuminate\Support\Facades\Gate;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The model to policy mappings for the application.
     *
     * @var array<class-string, class-string>
     */
    protected $policies = [
    'App\Models\Car' => 'App\Policies\CarPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();

        //
    }
}
```

Nous avons ajouté le **mapping** de notre politique dans la propriété **$policies**. Elle indique à Laravel d'appeler la méthode de politique correspondante pour autoriser l'action CRUD.

Etape 4 : Utiliser l'autorisation dans le contrôleur :

On va donc protéger les méthodes **edit, destroy et show** :

```php
<?php

namespace App\Http\Controllers;

use App\Models\Car;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;

class CarController extends Controller
{
    public function index()
    {
      if(Auth::user()->isAdmin)
        $cars=Car::all();
        else
        $cars=Auth::user()->cars;

        return view('cars.index',[
            "cars"=>$cars
        ]);
    }

    public function create()
    { $this->authorize('create',Car::class);
       return view('cars.create') ;
    }



    public function store(Request $request)
    {
      $rules=[
           "brand"=>'required',
           "price"=>['required','integer'],
           "origin"=>'required',
           "picture"=>['required','max:2048','image']

      ];
      $messages=[
        'picture.image'=>'please import an image'
      ];
      $validateData=Validator::make($request->all(),$rules,$messages);
      if($validateData->fails())   {
```

```
        return redirect()->back()->withErrors($validateData)->withInput();
      }
         //save data
        $path_image=$request->picture->store('cars','public');
         $car=new Car();
         //strip_tags pour empecher les scripts malveillents de s'executer
         $car['brand']=$request->input('brand');
         $car['origin']=$request->input('origin');
         $car['price']=$request->input('price');
         $car['picture']=$path_image;
         $car['user_id']=Auth::user()->id;
         $car->save();
                return redirect()->route('cars.index')->withSuccess('Car added
successfully');
    }



    public function show(Car $car)
    {
  $this->authorize('view',$car);
        return view('cars.show',[
            "car"=> $car
        ]);
    }


    public function edit(Car $car)//
    {
        $this->authorize('update',$car);
        return view('cars.edit',[
            "car"=>$car
        ]);
    }


    public function update(Request $request, $id)
    {
        $request->validate([
            "brand"=>'required',
            "price"=>['required','integer'],
            "origin"=>'required'

       ]);
          $car=Car::findOrFail($id);
          //strip_tags pour empecher les scripts malveillents de s'executer
          $car['brand']=strip_tags($request->input('brand'));
          $car['origin']=strip_tags($request->input('origin'));
          $car['price']=strip_tags($request->input('price'));
```

```
        $car->save();
                return redirect()->route('cars.show',$car->id);
    }


    public function destroy(Car $car)
    {
        $this->authorize('delete',$car);
         Car::delete($id);

        return Route::redirect('car.index');
    }
}
```

Et les routes correspondantes qu'on protège avec les middlewares **auth** et **verified**:

```
Route::resource('cars', CarController::class)->middleware(['auth','verified']);
```

Etape 5: Créer les vues en Blade

Ici, nous devons créer des fichiers de blade pour layout, login, register and index page. Donc créons un par un les fichiers :

resources/views/cars/index.blade.php

```
@extends('layouts.layout')
@section('title',"cars")
@section('content')

    <div class="carscontainer">
      <h1>My carage</h1>

      <a href="{{route('cars.create')}}" class="btn btn-primary m-2">Add new Car</a>
            <div class="cars">
  @forelse ($cars as $car)

     <div class="car">
      <span> {{$car['brand'] }}</span>
      <span>{{$car['price']}}</span>
      {{-- <span>{{$item['origin'] }}</span> --}}
      <span>{{$car['user']->name}}</span>
      <div class='groupbtn'>


  <form action="{{route('cars.destroy',$car)}}" method="POST">
    @method('DELETE')
```

```
    @csrf
     <a href="{{route('cars.show',$car)}}" class="btn btn-primary"> show</a>
     <a href="{{route('cars.edit',$car)}}" class="btn btn-success">Edit</a>
     @can('delete', $car)
            <button class="btn btn-danger">Delete</button>
     @endcan

  </form>
      </div>


  </div>



  @empty
      <h3>there is no cars</h3>

  @endforelse


    </div></div>
@endsection
```