S4 - Introduction à l'authentification en Laravel

Laravel propose un système d'authentification intégré qui permet de gérer facilement les utilisateurs, la connexion, l'enregistrement, ainsi que la protection des routes. Le système utilise les migrations, les modèles, les contrôleurs, et les vues pour offrir une expérience fluide de gestion des utilisateurs.

---

2. Configuration de l'authentification

Avant de commencer, assurez-vous que votre application Laravel est prête à gérer l'authentification.

Étape 1 : Installation des fonctionnalités d'authentification

Laravel 8+ propose un package d'authentification très pratique appelé **Laravel Breeze** ou **Laravel Jetstream**. Pour ce cours, nous utiliserons **Laravel Breeze**.

1. Exécutez la commande suivante pour installer Laravel Breeze :

   ```
   composer require laravel/breeze --dev
   ```

2. Ensuite, installez Breeze avec la commande :

   ```
   php artisan breeze:install
   ```

3. Compilez les assets pour que les vues soient prêtes à l'emploi :

   ```
   npm install && npm run dev
   ```

4. Exécutez la migration de la base de données pour créer les tables nécessaires pour l'authentification :

   ```
   php artisan migrate
   ```

Cela céera une table `users` et ajoutera les champs nécessaires pour gérer les utilisateurs.

---

3. Routes et Contrôleurs d'Authentification

Laravel Breeze génère automatiquement les routes et les contrôleurs nécessaires pour gérer l'authentification.

1. **Routes d'authentification :**

   Dans `routes/web.php`, vous trouverez des routes pour la connexion et l'enregistrement générées par Breeze :

```
Route::get('/login', [AuthenticatedSessionController::class, 'create'])
        ->name('login');
Route::post('/login', [AuthenticatedSessionController::class, 'store']);

Route::get('/register', [RegisteredUserController::class, 'create'])
        ->name('register');
Route::post('/register', [RegisteredUserController::class, 'store']);
```

---

4. Enregistrement d'un utilisateur (Register)

Étape 1 : Création du formulaire d'enregistrement

Lorsque vous utilisez Laravel Breeze, la vue d'enregistrement est générée automatiquement. Vous pouvez la trouver dans le fichier `resources/views/auth/register.blade.php`. Il contient un formulaire avec les champs suivants :

- Nom
- Adresse e-mail
- Mot de passe

Étape 2 : Enregistrement dans le contrôleur

Le contrôleur `RegisteredUserController` s'occupe de la logique d'enregistrement. Le code suivant se trouve dans `app/Http/Controllers/Auth/RegisteredUserController.php` :

```
use App\Models\User;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
use Illuminate\Http\Request;

public function store(Request $request)
{
    $validated = $request->validate([
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'confirmed', 'min:8'],
    ]);

    $user = User::create([
        'name' => $validated['name'],
        'email' => $validated['email'],
        'password' => Hash::make($validated['password']),
    ]);

    Auth::login($user);

    return redirect(RouteServiceProvider::HOME);
}
```

Explication :

- La méthode `store()` valide les entrées du formulaire et crée un nouvel utilisateur dans la base de données.
- Ensuite, l'utilisateur est automatiquement connecté après l'enregistrement grâce à `Auth::login()`.

---

5. Connexion d'un utilisateur (Login)

Étape 1 : Création du formulaire de connexion

Le formulaire de connexion est également généré automatiquement par Laravel Breeze et est situé dans `resources/views/auth/login.blade.php`.

Ce formulaire contient les champs suivants :

- E-mail
- Mot de passe

Étape 2 : Connexion dans le contrôleur

La logique de connexion est gérée par le contrôleur `AuthenticatedSessionController`.
Voici comment fonctionne la méthode `store()` :

```
use Illuminate\Support\Facades\Auth;
use Illuminate\Http\Request;

public function store(Request $request)
{
    $validated = $request->validate([
        'email' => ['required', 'email'],
        'password' => ['required'],
    ]);

    if (Auth::attempt($validated)) {
        return redirect()->intended(RouteServiceProvider::HOME);
    }

    return back()->withErrors([
        'email' => 'Les informations fournies sont incorrectes.',
    ]);
}
```

Explication :

- La méthode `store()` valide les informations de connexion, puis utilise `Auth::attempt()` pour tenter de connecter l'utilisateur.
- Si les informations sont correctes, l'utilisateur est redirigé vers la page d'accueil (`HOME`), sinon un message d'erreur est renvoyé.

---

6. Vérification des rôles d'utilisateur

Étape 1 : Ajout de rôles aux utilisateurs

Vous pouvez ajouter un champ `role` à votre table `users` pour gérer les rôles. Commencez par créer une migration pour ajouter ce champ :

```
php artisan make:migration add_role_to_users_table
```

Ensuite, modifiez la migration générée pour ajouter une colonne `role` :

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('role')->default('user');
    });
}
```

Après avoir ajouté la migration, exécutez-la avec :

```
php artisan migrate
```

Étape 2 : Vérification du rôle

Maintenant que vous avez un champ `role`, vous pouvez vérifier le rôle d'un utilisateur avant de lui accorder l'accès à certaines routes.

Dans votre contrôleur ou dans vos middlewares, vous pouvez utiliser le code suivant pour vérifier si l'utilisateur est un **admin** ou un **utilisateur** :

```
if (Auth::user()->role == 'admin') {
    // Effectuer des actions réservées aux admins
}
```

---

7. Protection des routes par rôle

Étape 1 : Utilisation des middlewares

Un **middleware** est un intermédiaire entre la requête (request) et la réponse (response) dans une application web.

Il s'agit d'une fonction qui s'exécute avant qu'une requête n'atteigne sa destination finale (le contrôleur ou la route ciblée).

🔷 Rôle d'un middleware :

- **Filtrer** ou **modifier** les requêtes et les réponses.
- **Vérifier l'authentification** d'un utilisateur.
- **Contrôler les permissions** (ex : vérifier si un utilisateur a le bon rôle).
- **Logger** les requêtes.

- Vous pouvez créer un middleware pour vérifier le rôle d'un utilisateur avant d'accéder à une route spécifique.

Exécutez la commande pour créer un middleware :

```
php artisan make:middleware EnsureUserIsAdmin
```

Dans `app/Http/Middleware/EnsureUserIsAdmin.php`, modifiez le code pour vérifier si l'utilisateur est un **admin** :

```
public function handle($request, Closure $next)
{
    if (Auth::user()->role != 'admin') {
        return redirect('home');
    }

    return $next($request);
```

Étape 2 : Protéger les routes avec le middleware

Ensuite, vous pouvez protéger les routes sensibles en utilisant ce middleware. Par exemple, dans `routes/web.php` :

```
Route::middleware(['auth', 'admin'])->group(function () {
    Route::get('/admin/dashboard', [AdminController::class, 'index']);
});
```

Dans ce cas, la route `/admin/dashboard` est protégée et accessible uniquement par les utilisateurs ayant le rôle `admin`.