Chapitre 6

Authentification des utilisateurs

L'authentification est un processus qui permet de vérifier l'identité d'un utilisateur. Elle est utilisée pour garantir que seules les personnes ou les entités autorisées ont accès à des informations ou à des ressources spécifiques. Dans le contexte d'un site web ou d'une application, l'authentification est souvent utilisée pour contrôler l'accès à des fonctionnalités ou à des pages spécifiques. Par exemple, un site web de commerce électronique peut utiliser l'authentification pour permettre aux utilisateurs de se connecter à leur compte et d'accéder à leur historique de commandes ou à leurs informations de paiement. L'authentification peut également être utilisée pour contrôler l'accès à des systèmes informatiques ou à des réseaux d'entreprise. Dans ce cas, les utilisateurs peuvent être obligés de saisir un nom d'utilisateur et un mot de passe pour accéder à certaines ressources.

En résumé, l'authentification est utilisée pour garantir que seules les personnes ou les entités autorisées ont accès à des informations ou à des fonctionnalités, ce qui contribue à renforcer la sécurité des systèmes et des données.

6.1 Packages d'authentification

Laravel fournit plusieurs packages pour gérer l'authentification, chacun ayant ses propres fonctionnalités et avantages. Voici une liste des packages d'authentification les plus couramment utilisés dans Laravel :

— **Laravel UI** [1] **:** UI est un package qui fournit des modèles d'interface utilisateur pour l'authentification et la gestion des utilisateurs, il génère ces interfaces graphiques en utilisant le framework CSS connu "Bootstrap". Ce package était largement utilisé sur les anciennes versions de Laravel (avant la version 8), mais même s'il fonctionne toujours sur les nouvelles versions, ce package n'est plus supporté et donc son usage n'est pas recommandé.

— **Laravel Breeze** [2] **:** C'est le remplaçant direct du package UI, il est disponible à partir de la version 8 de laravel. Il fournit un ensemble de modèles de vue et de routes pour l'authentification, la gestion des sessions et la réinitialisation des mots de passe . . . etc. Breeze est un package minimaliste qui permet de rapidement configurer une authentification complète. Il utilise Tailwind CSS pour une interface utilisateur responsive et moderne, et offre également la possibilité d'ajouter des fonctionnalités supplémentaires si nécessaire. Breeze est généralement recommandé pour les projets qui nécessitent une authentification simple et rapide. Il peut être installé avec Composer et sa configuration est facilement personnalisable comme nous allons voir dans les sections suivantes.

— **Laravel Sanctum** [3] **:** Sanctum est un package qui fournit une authentification stateless basée sur des jetons. Il est particulièrement utile développer une application back-end ou une API prête à l'utilisation par une application fron-end ou mobile.

— **Laravel Fortify** [4] **:** Laravel Fortify est un autre package d'authentificationqui fournit des fonctionnalités d'authentification avancées pour Laravel, telles que l'authentification à deux facteurs, la réinitialisation des mots de passe et la vérification par e-mail. Fortify est souvent utilisé pour des projets nécessitant une authentification avancée, il ne fournit aucune interface utilisateur mais il fournit seulement la logique backend.

— **Laravel Jetstream** [5] **:** Jetstream est un package qui fournit des modèles d'inter-

1. https://laravel.com/docs/7.x/authentication
2. https://laravel.com/docs/10.x/starter-kits#laravel-breeze
3. https://laravel.com/docs/10.x/sanctum
4. https://laravel.com/docs/10.x/fortify
5. https://jetstream.laravel.com/

face utilisateur pré-construits pour l'authentification et la gestion des utilisateurs. Il inclut également des fonctionnalités avancées telles que l'authentification en deux étapes et l'authentification par jeton API. "Jetstream" est basé sur les deux packages "Fortify" et "Sanctum", mais il rajoute une couche d'interface utilisateur par dessus.

Ces packages sont tous facilement accessibles via Composer, et chaque package a sa propre documentation détaillée. En fonction de vos besoins spécifiques, vous pouvez choisir celui qui convient le mieux à votre projet d'authentification. Dans la suite de ce chapitre nous allons nous focaliser sur le package "Breeze", un package recommandé pour les débutants car il fournit une configuration simple et rapide sans demander des prérequis difficiles, contrairment à "Jetstream" par exemple qui demande une conaissance préalable de `Livewire`[6] ou `Inertia.js`[7].

6.2 Installation de Breeze

Pour installer Laravel Breeze, vous pouvez devez premièrement utiliser composer pour le télécharger. Pour cela, ouvrez une fenêtre de commandes sur le répertoire de votre application et écrivez :

```
composer require laravel/breeze
```

Attendez la fin du téléchargement, puis tapez la commande `artisan` suivante :

```
php artisan breeze:install
```

Cette commande va modifier le fichier de routes, ajouter/modifier quelques vues et créer des contrôleurs pour gérer l'authentification. Il est parfois nécessaire d'exécuter les deux commandes suivantes après l'installation de "Breeze" :

```
npm install
npm run dev
```

6. https://laravel-livewire.com/
7. https://inertiajs.com/

Il faut avoir `npm` (Node.js pour windows) installé pour pouvoir lancer ces deux commandes. Ces deux commandes permettent de télécharger puis installer les ressources frontales de l'applications (principalement du CSS et Javascript).

6.2.1 Fichiers créés

Après l'installation de Laravel Breeze, plusieurs fichiers sont créés dans le répertoire de votre application. Voici une liste des fichiers les plus importants avec leur utilité :

- **/routes/auth.php :** ce fichier est inclus dans le fichier de routes principale (`/routes/web.php`). le fichier `/routes/auth.php` contient les routes permettant l'authentification, l'inscription, la confirmation du mot de passe ou de l'email, la déconnexion réinitialisation du mot de passe ... etc.
- **/app/Http/Controllers/Auth :** ce répertoire contient les contrôleurs d'authentification pour votre application Laravel. Les contrôleurs pour l'inscription, la connexion, la déconnexion, la réinitialisation du mot de passe et la vérification d'adresse e-mail sont aussi générés automatiquement après l'installation de Breeze.
- **/app/Http/Controllers/ProfileController :** un autre contrôleurs contenant des méthodes permettant de gérer le profil d'utilisateur, comme la modifications des informations, du mot de passe ou même la suppression du compte.
- **/resources/views/auth :** ce répertoire contient les fichiers de vues pour les pages d'authentification de votre application Laravel. Après l'installation de Breeze, plusieurs fichiers de vue sont ajoutés à ce répertoire pour gérer l'inscription, la connexion, la réinitialisation du mot de passe et la vérification de l'adresse e-mail.
- **/config/auth.php :** ce fichier contient la configuration d'authentification pour votre application. Après l'installation de Breeze, plusieurs options d'authentification sont ajoutées à ce fichier pour configurer les fonctionnalités d'authentification. Vous devez modifier ce fichier à partir du moment ou vous voulez utiliser plusieurs modèles pour l'authentification, ou même utiliser un modèle différent de `User`.

En plus de la création de ces dossiers et fichiers, l'installation de "Breeze" va égale-

ment introduire des modifications sur certains fichiers existants, ces fichiers incluent `/resources/views/welcome.blade.php` et `/routes/web.php`.

Après avoir terminé l'installation, vous allez voir apparaître deux boutons `Login` et `Register` sur la page d'accueil de votre application. Cliquez dessus et testez les différentes nouvelles fonctionnalités.

6.3 Reconnaître l'utilisateur authentifié

Il est souvent nécessaire pour votre code de récupérer les informations de l'utilisateur authentifié afin d'afficher les bonnes informations ou exposer les bonnes fonctionnalités. Pour ce faire, vous pouvez utiliser la classe `Auth` définie dans `Illuminate\Support\Facades\`. Cette classe définie trois (3) méthodes statiques très importantes à utiliser pour assurer le bon fonctionnement de votre application.

— **Auth : :user() :** qui retourne un objet contenant les informations de l'utilisateur authentifié.

— **Auth : :id() :** qui retourne l'identifiant de l'utilisateur authentifié.

— **Auth : :check() :** qui retourne `true` si une session est ouverte et `false` sinon.

6.3.1 Les directives @auth et @guest

Les directives `@auth` et `@guest` sont des directives Blade fournies permettant de vérifier si un utilisateur est authentifié ou non dans votre application.

La directive `@auth` est utilisée pour vérifier si un utilisateur est authentifié dans votre application. Si l'utilisateur est authentifié, le contenu à l'intérieur de la directive sera affiché. Sinon, le contenu sera ignoré. Vous pouvez aussi utiliser la directive `@else` dans pour le cas ou aucun utilisateur est authentifié. Voici un exemple :

```
@auth
    <p>Bienvenue {{ Auth::user()->name }}</p>
@else
```

```
    <p>Aucun utilisateur authentifié
@endauth
```

L'exemple précédent affiche le nom de l'utilisateur s'il est connecté, et un message sinon.

La directive `@guest` fait l'inverse de `@auth`. Elle permet de vérifier que l'utilisateur n'est pas authentifié dans votre application. Le contenu à l'intérieur de la directive ne sera affiché que si l'utilisateur n'est pas authentifié. Voici un exemple :

```
@guest
    <a href="{{ route('login') }}">Connexion</a>
@else
    <a href="{{ route('logout') }}">Déconnexion</a>
@endguest
```

Le code précédent affiche un lien de connexion si l'utilisateur n'est pas connecté, et un lien de déconnexion sinon.

6.4 Les middlewares

Les middlewares sont des fonctions intermédiaires qui peuvent être exécutées avant une requête HTTP dans une application Laravel. Les middlewares sont utilisés pour ajouter des fonctionnalités à une application, comme l'authentification, la validation de formulaire, la gestion des cookies, la mise en cache, la journalisation . . . etc. Lorsqu'une requête HTTP est reçue par une application Laravel, elle est traitée par une série de middlewares qui effectuent des actions spécifiques sur la requête. Les middlewares peuvent modifier la requête, la valider ou même la bloquer complètement. Une fois que tous les middlewares ont été exécutés, la requête est transmise à la route appropriée pour être traitée.

Nous allons ici voir quelques middlewares prédifinies dans Laravel, mais vous pouvez définir vos propres middlewares si nécessaire. Nous nous intéressons particulièrement aux middlewares d'authetification.

— **auth :** Le middleware "auth" vérifie si l'utilisateur est authentifié. Si l'utilisateur n'est pas authentifié, il est redirigé vers la page de connexion.

— **guest :** Le middleware "guest" vérifie si l'utilisateur n'est pas authentifié. Si l'utilisateur est authentifié, il est redirigé vers son dashboard.

— **password.confirm :** demande à l'utilisateur actuel de confirmer son mot de passe avant d'accéder à certaines pages protégées. Il est souvent utilisé pour protéger l'utilisateur contre des actions irréversibles comme la modification du mot passe ou la suppression du compte.

6.4.1 Utiliser un middleware

Pour protéger une route avec un middleware, nous utilisons la méthode `middleware` en passant comme paramètre le nom du middleware utilisé.

```
Route::get('/profile', [ProfileController::class, 'edit'])->middleware('auth');
```

Dans l'exemple précédent nous empêchons les utilisateurs non authentifiés de visiter l'URL `/profile`, ils seront automatiquement redirigés vers la page de connexion. Il est aussi possible de définir un seul middleware pour plusieurs routes de la façon suivante :

```
Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit']);
    Route::put('/profile', [ProfileController::class, 'update']);
    Route::delete('/profile', [ProfileController::class, 'destroy']);
});
```

Dans cet exemple, toutes les routes dans la méthode `group` sont protégées par le middleware `auth`. Vu que toutes les fonctions dans cette exemple appartiennent au même contrôleur, il est possible de placer le middleware `auth` dans le constructeur du contrôleur en question comme le montre l'exemple ci-dessous.

```
...
class ProfileController extends Controller {
```

```
        ...
        public function __construct() {
            $this->middleware('auth');
        }
        ...
    }
```

de cette façon, même si les routes change, le middleware reste toujours lié à ce contrôleur. Si nous voulons utiliser le middleware pour quelques fonctions seulement, ou nous voulons exclure quelques fonctions, nous pouvons utiliser les méthodes `only` et `except` qui définissent respectivement les méthodes sur lesquelles il faut, ou il ne faut pas utiliser un middleware.