Cours Laravel 6 – les données – l'authentification

laravel.sillo.org/cours-laravel-6-les-donnees-lauthentification/

bestmomo

August 30, 2019

L'authentification constitue une tâche fréquente. En effet il y a souvent des parties d'un site qui ne doivent être accessibles qu'à certains utilisateurs, ne serait-ce que l'administration. La solution proposée par Laravel est d'une grande simplicité parce que tout est déjà préparé comme nous allons le voir dans ce chapitre.

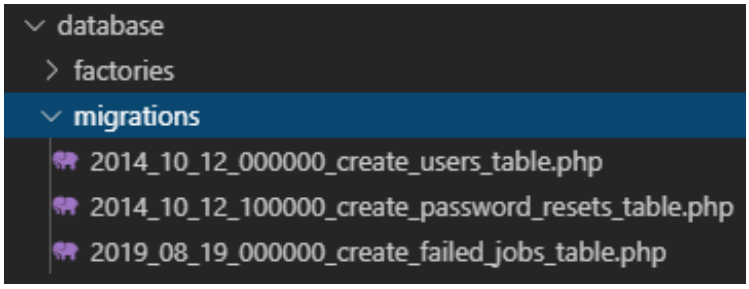
Mais avec Laravel 6 apparait une nouveauté. Précédemment il y avait la commande **php artisan make:auth** pour générer les vues. Cette commande a disparu au profit d'un package à installer.

La base de données

Par défaut la partie persistance de l'authentification (c'est à dire la manière de retrouver les renseignements des utilisateurs) avec Laravel se fait en base de données avec Eloquent et part du principe qu'il y a un modèle **App\User** (dans le dossier **app**).

Lors de l'installation on a vu dans le chapitre précédent qu'il existe déjà des migrations :

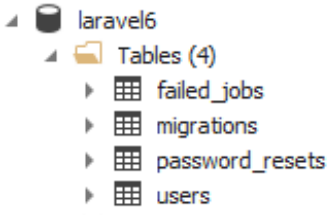
Repartez d'une installation vierge et faites la migration avec Artisan :




```
λ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (0.48 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (0.5 seconds)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (0.27 seconds)
```

Vous devriez normalement obtenir ces tables :

*Je rappelle que la table **migrations** sert seulement d'intendance pour les migrations et que vous ne devez pas y toucher.*



- table **users** : par défaut Laravel considère que cette table existe et il s'en sert comme référence pour l'authentification.

- table **password_resets** : cette table va nous servir pour la réinitialisation des mots de passe.

Les middlewares

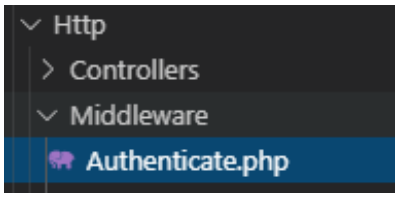
Je vous ai déjà parlé des middlewares qui servent à effectuer un traitement à l'arrivée (ou au départ) des requêtes HTTP. On a vu le middleware de groupe **web** qui intervient pour toutes les requêtes qui arrivent. Dans ce chapitre on va voir deux autres middlewares qui vont nous permettre de savoir si un utilisateur est authentifié ou pas pour agir en conséquence.

Middleware auth

Le middleware **auth** permet de n'autoriser l'accès qu'aux utilisateurs authentifiés. Ce middleware est déjà présent et déclaré dans **app\Http\Kernel.php** :

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    ...
];
```

On voit que la classe se trouve dans l'application (ce qui n'était pas le cas avant la version 6) :



On peut utiliser ce middleware directement sur une route :

```
Route::get('comptes', function() {
    // Réservé aux utilisateurs authentifiés
})->middleware('auth');
```

Ou un groupe de routes :

```
Route::middleware('auth')->group(function () {
    Route::get('/', function ()    {
        // Réservé aux utilisateurs authentifiés
    });
    Route::get('comptes', function () {
        // Réservé aux utilisateurs authentifiés
    });
});
```

*Le groupement de routes avec la méthode **group** permet de mutualiser des actions et de simplifier le code.*

Ou dans le constructeur d'un contrôleur :

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

Dans ce cas on peut désigner les méthodes concernées avec **only** ou non concernées avec **except** :

```
public function __construct()
{
    $this->middleware('auth')->only(['create', 'update']);
}
```

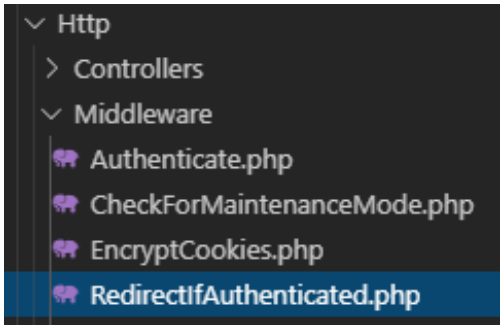
Middleware guest

Ce middleware est exactement l'inverse du précédent : il permet de n'autoriser l'accès qu'aux utilisateurs non authentifiés. Ce middleware est aussi déjà présent et déclaré dans **app\Http\Kernel.php** :

```
protected $routeMiddleware = [
    ...
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    ...
];
```

La classe se trouve aussi dans l'application :

De la même manière que **auth**, le middleware **guest**, comme d'ailleurs tous les middlewares, peut s'utiliser sur une route, un groupe de route ou dans le constructeur d'un contrôleur, avec la même syntaxe.



Les routes de l'authentification

Dans l'installation de base vous ne trouvez aucune route pour l'authentification. Pour les créer (et ça ne créera pas seulement les routes) il faut déjà installer un package :

```
composer require laravel/ui
```

```
λ composer require laravel/ui
Using version ^1.0 for laravel/ui
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing laravel/ui (v1.0.0): Downloading (100%)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Discovered Package: laravel/ui
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
```

On dispose alors d'un nouvelle commande d'artisan :

```
λ php artisan help ui
Description:
  Swap the front-end scaffolding for the application

Usage:
  ui [options] [--] <type>

Arguments:
  type                  The preset type (bootstrap, vue, react)

Options:
      --auth            Install authentication UI scaffolding
      --option[=OPTION] Pass an option to the preset command (multiple values allowed)
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi            Force ANSI output
      --no-ansi         Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --env[=ENV]       The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for
```

On voit qu'on dispose de 3 preset : bootstrap, vue et react. On a donc le choix ! On voit aussi qu'on a l'option **–auth** pour générer le scaffolding.

Le choix n'est pas encore vraiment important pour le moment et on va prendre par exemple **vue** :

```
λ php artisan ui vue --auth
Vue scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.
Authentication scaffolding generated successfully.
```

Comme précisé il faut ensuite utiliser ces deux commandes pour générer les fichiers CSS et Javascript :

```
npm install
npm run dev
```

Ca prend un peu de temps parce qu'il y a pas mal de code à charger. Je parlerai de cet aspect dans un article ultérieur.

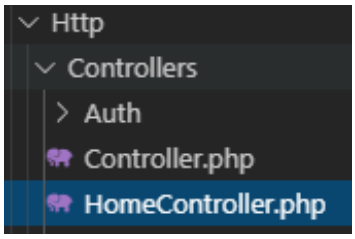
Pendant ce temps regardez ce qui a été ajouté dans le fichier **routes/web.php** :

```
Auth::routes();

Route::get('/home', 'HomeController@index')->name('home');
```

Apparemment il a aussi été créé le contrôleur **HomeController** :

Voyons quelles sont les routes générées (vous devez maintenant bien connaître la commande correspondante d'Artisan) :



Method	URI	Name	Action	Middleware
GET|HEAD	home	home	App\Http\Controllers\HomeController@index	web,auth
GET|HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web,guest
POST	login		App\Http\Controllers\Auth\LoginController@login	web,guest
POST	logout	logout	App\Http\Controllers\Auth\LoginController@logout	web
POST	password/email	password.email	App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail	web,guest
GET|HEAD	password/reset	password.request	App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm	web,guest
POST	password/reset	password.update	App\Http\Controllers\Auth\ResetPasswordController@reset	web,guest
GET|HEAD	password/reset/{token}	password.reset	App\Http\Controllers\Auth\ResetPasswordController@showResetForm	web,guest
GET|HEAD	register	register	App\Http\Controllers\Auth\RegisterController@showRegistrationForm	web,guest
POST	register		App\Http\Controllers\Auth\RegisterController@register	web,guest

Vous voyez que **Auth::routes()** produit pas mal de routes !

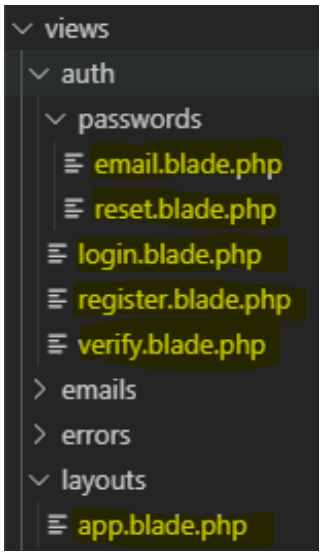
*Vous pouvez voir également l'utilisation des deux middlewares **auth** et **guest** dont je vous ai parlé plus haut.*

Nous allons voir bientôt tout ça en action.

Les vues de l'authentification

Il y a également eu la génération de vues :

On va aussi voir à quoi servent toutes ces vues.



L'enregistrement d'un utilisateur

Commençons par le commencement avec l'enregistrement d'un utilisateur. Si vous allez sur la page d'accueil vous allez remarquer la présence en haut à droite de la page de deux liens :

On trouve ce code dans la page d'accueil :

LOGIN REGISTER

```
@if (Route::has('login'))
    <div class="top-right links">
        @auth
            <a href="{{ url('/home') }}">Home</a>
        @else
            <a href="{{ route('login') }}">Login</a>

            @if (Route::has('register'))
                <a href="{{ route('register') }}">Register</a>
            @endif
        @endauth
    </div>
@endif
```

On en profite pour voir les opérateurs conditionnels de Blade.

Avec un **@if** on vérifie si dans les routes (**Route**) on a (**has**) une route **login** et si c'est le cas on vérifie si l'utilisateur est authentifié avec **@auth**, si c'est le cas on affiche un lien sur la page **home** et si ce n'est pas le cas (**@else**) on affiche les deux liens de l'authentification.

Si on clique sur **Register** on appelle la méthode **showRegistrationForm** du contrôleur **RegisterController** :

```
GET|HEAD | register | register | App\Http\Controllers\Auth\RegisterController@showRegistrationForm | web,guest
```

*Remarquez au passage la déclaration du middleware **guest** dans ce contrôleur :*

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

En effet si on est authentifié c'est qu'on n'a pas besoin de s'enregistrer !

Si vous regardez dans ce contrôleur vous n'allez pas trouver la méthode **showRegistrationForm** , par contre vous allez trouver un trait :

```
use RegistersUsers;
```

L'espace de nom complet est **Illuminate\Foundation\Auth\RegistersUsers**. Ce trait est donc dans le framework, et voici la méthode **showRegistrationForm** :

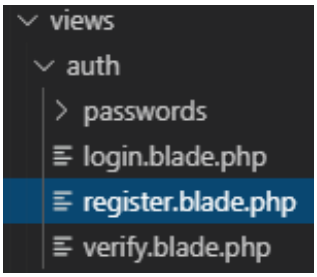
```
public function showRegistrationForm()
{
   return view('auth.register');
}
```

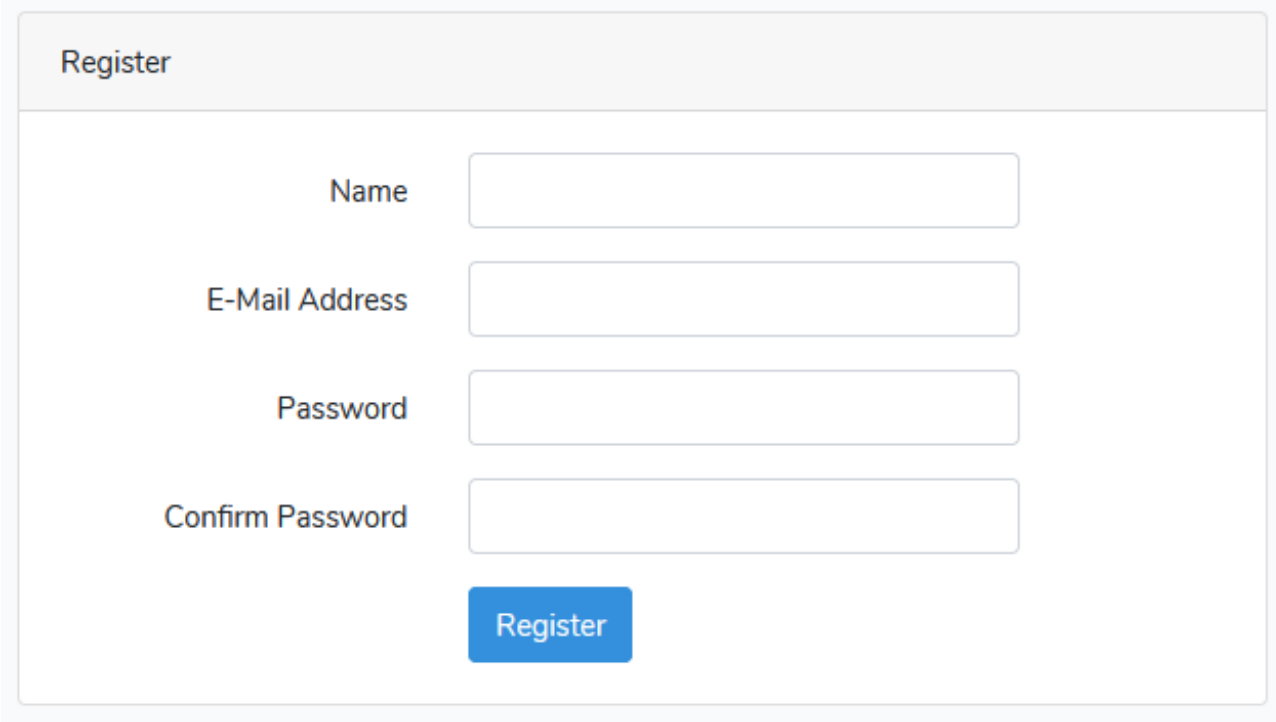
Donc pour changer le nom de la vue il faut surcharger cette méthode dans le contrôleur.

La vue register

On va donc chercher cette vue :

Elle a cet aspect :





Cette vue utilise le template **resources/views/layouts/app.blade.php**. Je ne vais pas entrer dans le détail de tout le code mais il y a des points intéressants.

Dans le template on trouve ces lignes de code :

```
@guest
    <li class="nav-item">
        <a class="nav-link" href="{{ route('login') }}">{{ __('Login') }}</a>
    </li>
    @if (Route::has('register'))
        <li class="nav-item">
            <a class="nav-link" href="{{ route('register') }}">{{ __('Register') }}</a>
        </li>
    @endif
@else
    <li class="nav-item dropdown">
        <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
            {{ Auth::user()->name }} <span class="caret"></span>
        </a>

        ...
    </li>
@endguest
```

On utilise **@guest** si on a affaire à un utilisateur non authentifié. En effet dans ce cas on génère les liens pour le login et l'enregistrement.

On trouve aussi dans le template ce code :

```
Auth::user()->name
```

La méthode **user** permet de disposer d'une instance du modèle pour l'utilisateur authentifié, ici du coup on peut connaître son nom. On pourrait utiliser l'helper **auth** :

```
auth()->user()->name
```

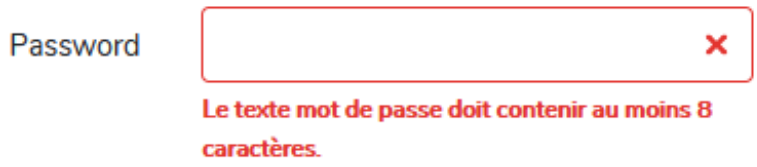
La validation

La validation pour l'enregistrement est présente dans le contrôleur **RegisterController** :

```
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:8|confirmed',
    ]);
}
```

De cette manière elle est facile à modifier si vous devez changer des règles ou ajouter un champ.

On peut vérifier que ça fonctionne :



La création de l'utilisateur

La création de l'utilisateur se fait dans le contrôleur **RegisterController** :

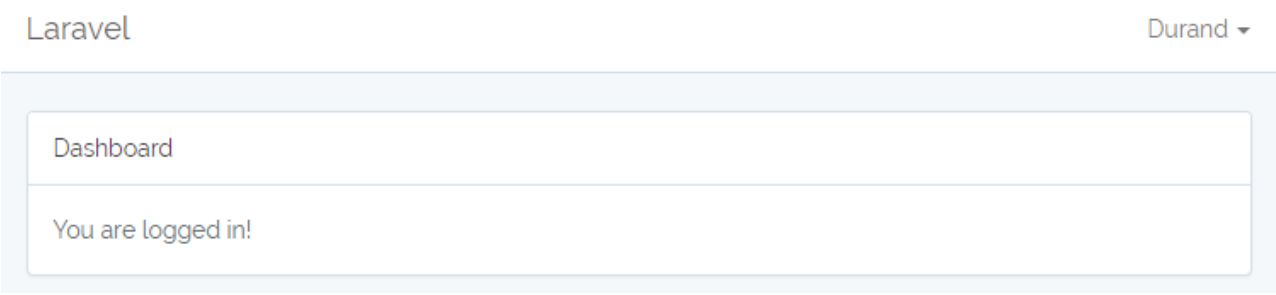
```
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);
}
```

On utilise la méthode **create** d'Eloquent comme on l'a déjà vu au chapitre précédent. Vous trouvez dans le modèle **App\User.php** la propriété **$fillable** renseignée :

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

Donc là encore il est facile d'apporter des modifications si nécessaire.

Une fois que l'utilisateur est créé dans la base il est automatiquement connecté et est redirigé sur **home** :

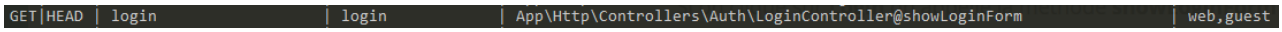


La connexion d'un utilisateur

Pour se connecter, un utilisateur doit cliquer sur le lien **login** de la page d'accueil :

Si vous venez juste d'enregistrer un utilisateur il faudra le déconnecter avant d'avoir accès au formulaire de login.

Si on clique sur **Login** on appelle la méthode **showLoginForm** du contrôleur **LoginController** :



Remarquez au passage la déclaration du middleware **guest** dans ce contrôleur :

```
public function __construct()
{
   $this->middleware('guest')->except('logout');
}
```

En effet si on est authentifié c'est qu'on n'a pas besoin de se connecter ! D'autre part avec l'option **except** on évite d'appliquer le middleware au logout.

Si vous regardez dans ce contrôleur vous n'allez pas trouver la méthode **showLoginForm**, par contre vous allez trouver un trait :

```
use AuthenticatesUsers;
```

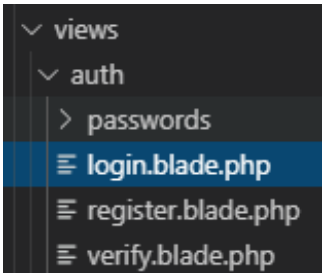
L'espace de nom complet est **Illuminate\Foundation\Auth\AuthenticateUsers**. Ce trait est donc dans le framework, et voici la méthode **showLoginForm** :

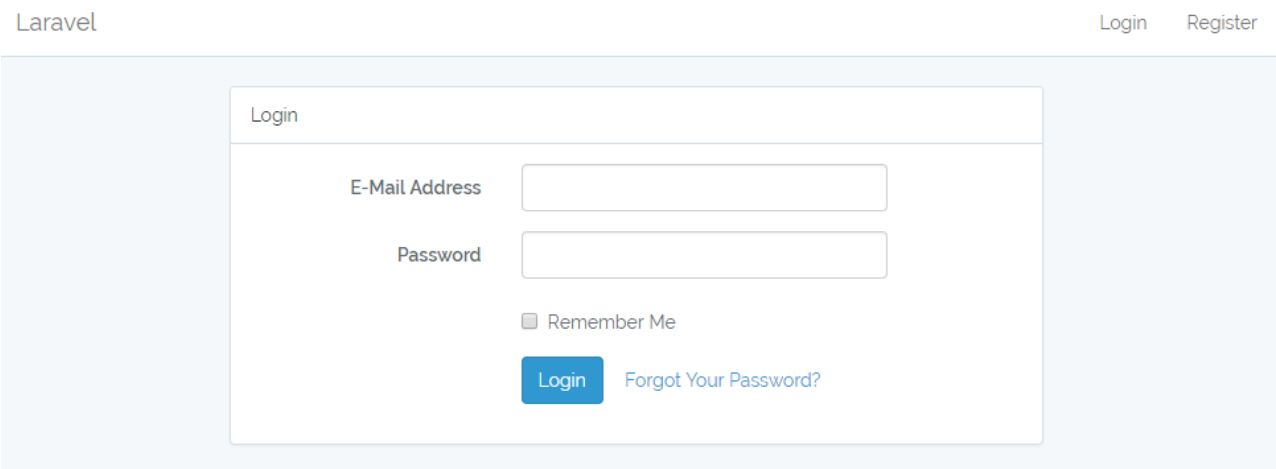
```
public function showLoginForm()
{
   return view('auth.login');
}
```

La vue login

On va donc chercher cette vue :

Elle a cet aspect :





Cette vue utilise aussi le template **resources/views/layouts/app.blade.php**. Il n'y a rien de particulier dans cette vue.

La validation

La validation pour la connexion est présente dans le trait **AuthenticatesUsers** :

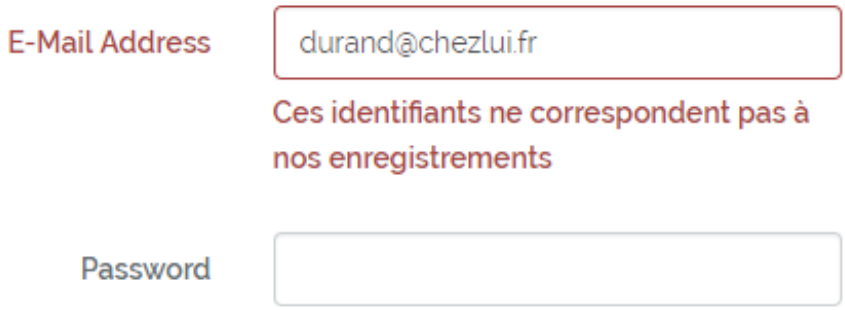
```
protected function validateLogin(Request $request)
{
   $request->validate([
      $this->username() => 'required|string',
      'password' => 'required|string',
   ]);
}
```

On peut surcharger cette méthode dans le contrôleur pour la modifier. Pour changer seulement le premier élément (**username**) il y a la méthode dans le trait :

```
public function username()
{
   return 'email';
}
```

Par défaut c'est l'email. Pour changer ça il faut surcharger cette méthode dans le contrôleur.

En plus de la validation il est effectué une vérification de la présence du couple email/password (credentials) dans la base de données :



Dans le formulaire de connexion il y a une case à cocher « se rappeler de moi » (remember me) :

Si on coche cette case on reste connecté indéfiniment jusqu'à ce qu'on se déconnecte intentionnellement. Pour que ça fonctionne il faut une colonne **remember_token** dans la table **users** :

Il se trouve que cette colonne est prévue dans la migration de base pour la table.

La redirection

Lorsque l'utilisateur est bien authentifié il est redirigé par la méthode **redirectPath** du trait **Illuminate\Foundation\Auth\RedirectsUsers** :

```
public function redirectPath()
{
   if (method_exists($this, 'redirectTo')) {
      return $this->redirectTo();
   }

   return property_exists($this, 'redirectTo') ? $this->redirectTo : '/home';
}
```

Par défaut on redirige sur **home** ou sur la valeur de la propriété ou la méthode **redirectTo** si l'une d'elles existe. Donc pour changer la redirection par défaut il suffit de créer cette propriété ou cette méthode avec la valeur voulue.

En fait ce que j'ai écrit ci-dessus est incomplet, il faut préciser quelque chose. On peut arriver sur le formulaire de connexion parce qu'on clique sur le lien correspondant, mais on peut aussi y arriver par l'action du middleware **auth**. En effet si on veut atteindre une

page pour laquelle il faut être authentifié le middleware **auth** va nous bloquer et nous rediriger sur le formulaire de connexion. Dans ce cas ce qui serait bien serait, à l'issue de la connexion, d'aller directement sur la page qu'on désirait atteindre au départ.

Sans entrer dans les détails la route désirée est mémorisée dans la session et voici le code pour la redirection :

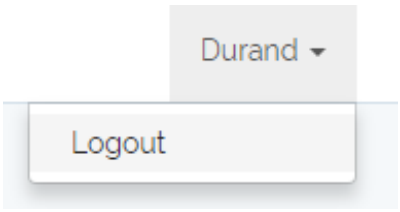
```
redirect()->intended($this->redirectPath())
```

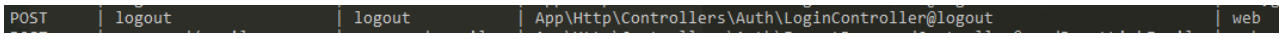
La méthode **intended** va vérifier si on a mémorisé une route dans la session et l'utiliser si c'est le cas, sinon on utilise la méthode **redirectPath** qu'on a vue ci-dessus.

La déconnexion d'un utilisateur

L'utilisateur dispose d'un lien pour se déconnecter :

Si on clique sur **Logout** on appelle la méthode **logout** du contrôleur **LoginController** :





Remarquez que la méthode est POST. Du coup la vue utilise un formulaire et une soumission en Javascript :

```
<a class="dropdown-item" href="{{ route('logout') }}"
   onclick="event.preventDefault();
                 document.getElementById('logout-form').submit();">
   {{ __('Logout') }}
</a>

<form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">
   @csrf
</form>
```

Si vous regardez dans le contrôleur **LoginController** vous n'allez pas trouver la méthode **logout**, par contre vous allez trouver un trait :

```
use AuthenticatesUsers;
```

On a déjà vu ce trait plus haut pour le login. Voici la méthode **logout** dans ce trait :

```
public function logout(Request $request)
{
    $this->guard()->logout();

    $request->session()->invalidate();

    return $this->loggedOut($request) ?: redirect('/');
}
```

La méthode **invalidate** efface toutes les informations de la session en cours.

On redirige sur l'url de base « / ».

En résumé

- L'authentification est totalement et simplement prise en charge par Laravel.
- Les migrations pour l'authentification sont déjà dans l'installation de base de Laravel.
- L'ajout du package **laravel/ui** permet de générer les routes et les vues pour l'authentification.
- Les middlewares **auth** et **guest** permettent d'interdire les accès aux routes qui concernent uniquement les utilisateurs authentifiés ou non authentifiés.