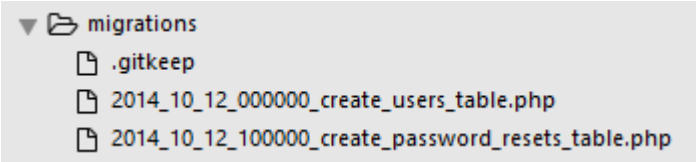
Cours Laravel 5.3 – les données – l'authentification

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.

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).

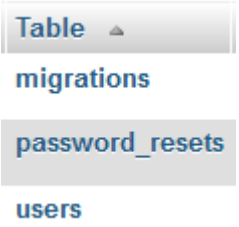
Lors de l'installation il existe deux migrations présentes :



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

```
λ php artisan migrate
Migration table created successfully.
Migrated: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_100000_create_password_resets_table
```

Vous devriez normalement obtenir ces 3 tables :



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

La table users

Par défaut Laravel considère que cette table existe et il s'en sert comme référence pour l'authentification. On a déjà créé et utilisé cette table dans le chapitre précédent.

La table password_reset

Lors des précédents chapitres je vous ai fait supprimer la migration pour cette table parce qu'on avait juste besoin de la table des utilisateurs. Pour ce chapitre on va avoir besoin de la seconde table qui 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' => \Illuminate\Auth\Middleware\Authenticate::class,
    ...
];
```

On voit que la classe se trouve dans le framework et on ne va donc pas pouvoir le modifier directement si on veut changer son fonctionnement.

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

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** :

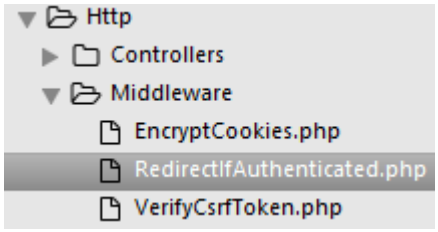
```
public function __construct()
{
    $this->middleware('auth')->only(['create', 'update']);
}
```

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,
    ...
];
```

Cette fois la classe se trouve dans l'application :



Pourquoi celui-ci est-il dans l'application ?

Pour comprendre pourquoi c'est le cas regardez le code :

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;

class RedirectIfAuthenticated
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @param  string|null  $guard
     * @return mixed
     */
    public function handle($request, Closure $next, $guard = null)
    {
        if (Auth::guard($guard)->check()) {
            return redirect('/home');
        }

        return $next($request);
    }
}
```

Si l'utilisateur est authentifié on fait une redirection vers la route « /home ». Si la classe était dans le framework on aurait du mal à changer cette url ! Alors que là c'est accessible et modifiable.

De la même manière que **auth**, le middleware **guest** 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 y a une commande d'Artisan :

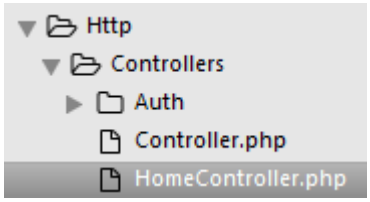
```
λ php artisan make:auth
Authentication scaffolding generated successfully.
```

Regardez ce qui a été ajouté dans le fichier **routes/web.php** :

```
Auth::routes();

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

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	/		Closure	web
GET|HEAD	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		App\Http\Controllers\Auth\LoginController@logout	web
POST	password/email		App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail	web,guest
GET|HEAD	password/reset		App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm	web,guest
POST	password/reset		App\Http\Controllers\Auth\ResetPasswordController@reset	web,guest
GET|HEAD	password/reset/{token}		App\Http\Controllers\Auth\ResetPasswordController@showResetForm	web,guest
GET|HEAD	register		App\Http\Controllers\Auth\RegisterController@showRegistrationForm	web,guest
POST	register		App\Http\Controllers\Auth\RegisterController@register	web,guest

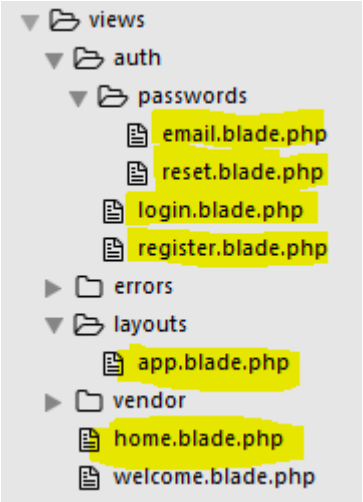
Vous voyez que **Auth::routes()** produit pas mal de routes !

Vous pouvez voir également l'utilisation des deux middlewares 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 :

LOGIN REGISTER

On trouve ce code dans la page d'accueil :

```
@if (Route::has('login'))
    <div class="top-right links">
        <a href="{{ url('/login') }}">Login</a>
        <a href="{{ url('/register') }}">Register</a>
    </div>
@endif
```

Avec un **@if** on vérifie si dans les routes (**Route**) on a (**has**) une route « login » et si c'est le cas on génère les deux liens.

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

```
GET|HEAD | 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;
```

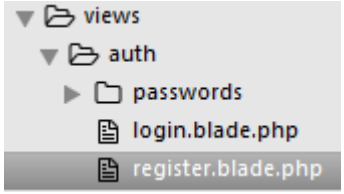
L'espace de nom complet est : **Illuminate\Foundation\Auth\RegistersUsers**. Ce trait est donc dans le framework, et voici la méthode **showRegistrationForm** :

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

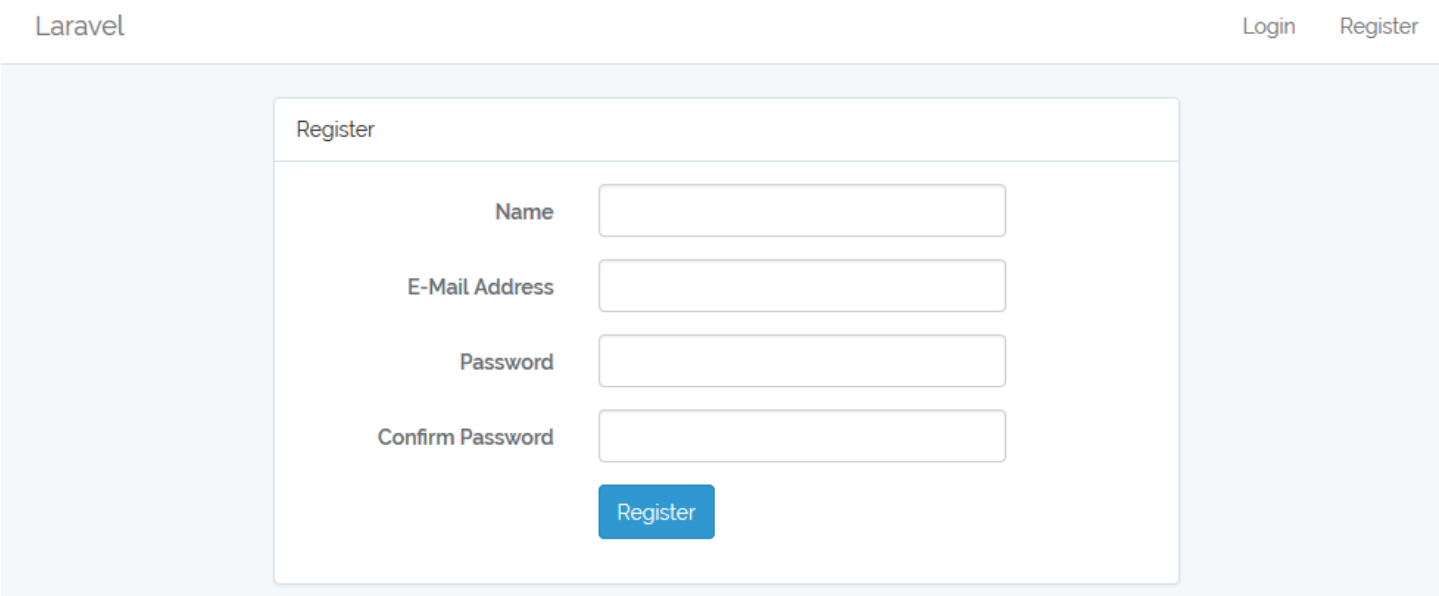
Donc pour changer le nom de la vue il faut surcharger cette méthode dans le contrôleur.

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 :

```
@if (Auth::guest())
    <li><a href="{{ url('/login') }}">Login</a></li>
    <li><a href="{{ url('/register') }}">Register</a></li>
@else
    <li class="dropdown">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-expanded="false">
            {{ Auth::user()->name }} <span class="caret"></span>
        </a>
        ...
    </li>
@endif
```

On utilise la façade **Auth** pour vérifier avec la méthode **guest** qu'on a affaire à un utilisateur non authentifié :

```
@if (Auth::guest())
```

En effet dans ce cas on génère les liens pour le login et l'enregistrement. On pourrait tout aussi bien utiliser l'helper **auth** :

```
@if (auth()->guest())
```

On dispose de la méthode **auth** pour vérifier l'inverse : qu'un

utilisateur est authentifié.

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. Ici aussi on pourrait utiliser l'helper :

auth()->user()->name

Comme la vue n'utilise pas le composant **laravelcollective/html** le jeton (token)pour la protection CSRF est généré avec cet helper :

{{ csrf_field() }}

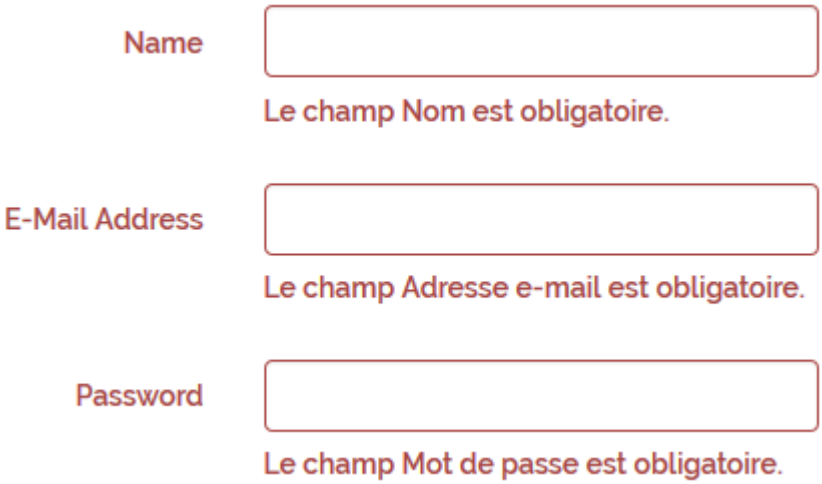
La validation

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

```
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => 'required|max:255',
        'email' => 'required|email|max:255|unique:users',
        'password' => 'required|min:6|confirmed',
    ]);
}
```

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

On peut vérifier que tout fonctionne :



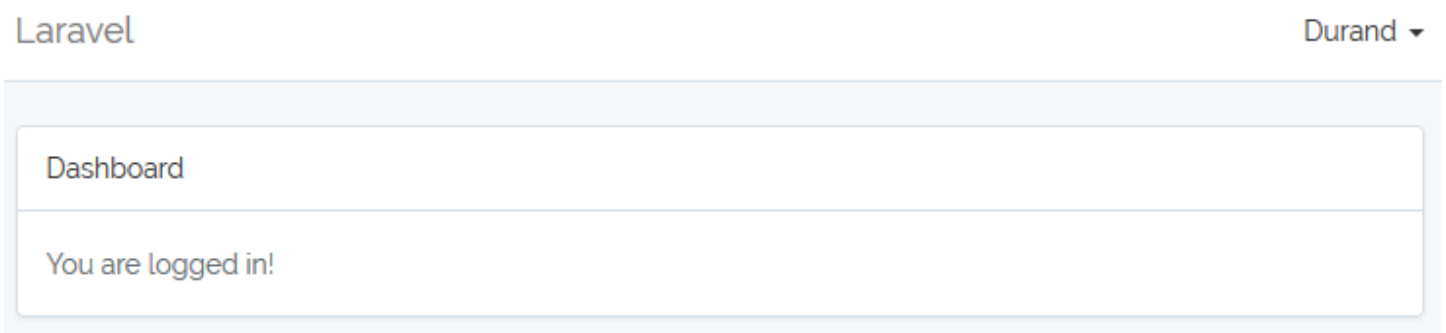
La création de l’utilisateur

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

```
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => bcrypt($data['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 » :

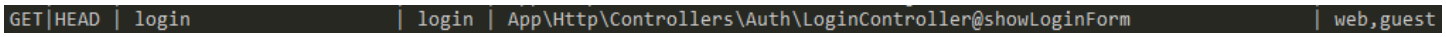


La connexion d’un utilisateur

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 :

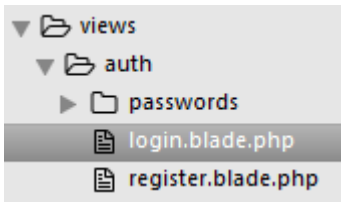
```
use AuthenticatesUsers;
```

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

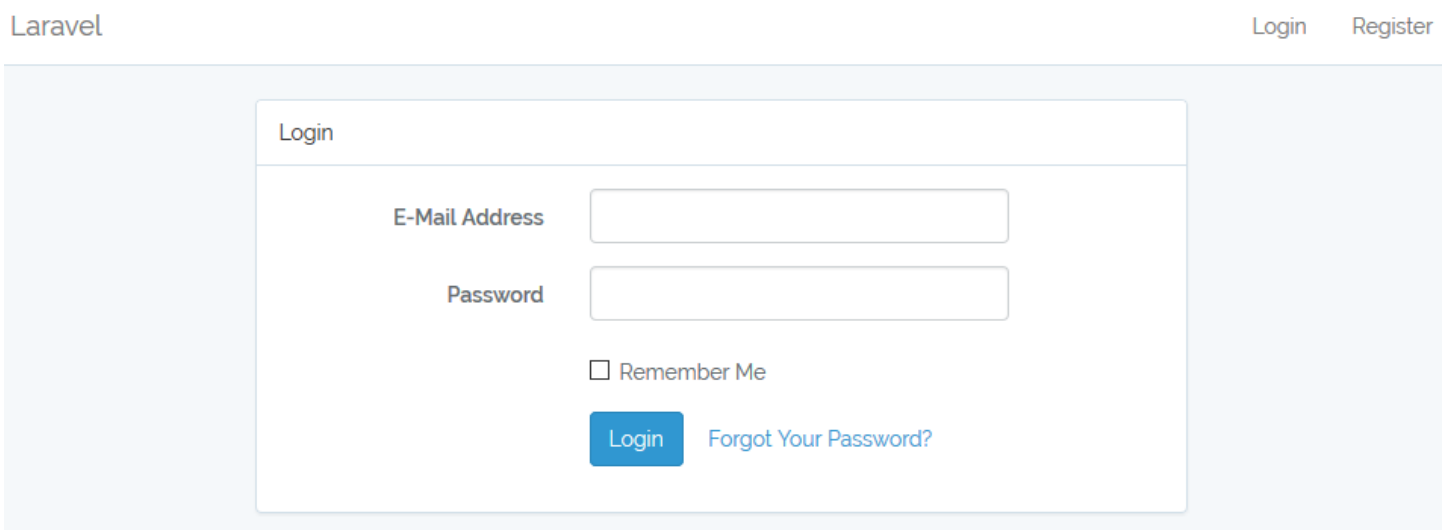
```
public function showLoginForm()
{
    return view('auth.login');
}
```

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** :

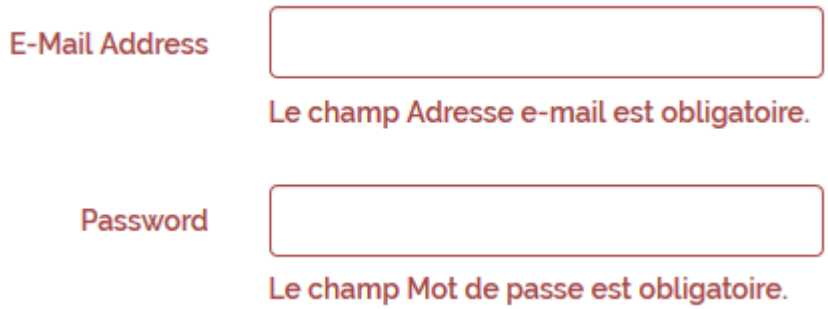
```
protected function validateLogin(Request $request)
{
    $this->validate($request, [
        $this->username() => 'required', 'password' => 'required',
    ]);
}
```

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 :

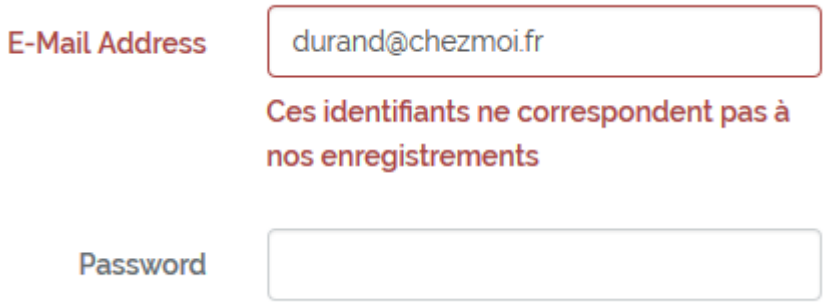
```
public function username()
{
    return 'email';
}
```

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

On peut vérifier que tout fonctionne :

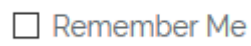


Il est ensuite effectué une vérification de la présence du couple **email/password** (**credentials**) dans la base de données :

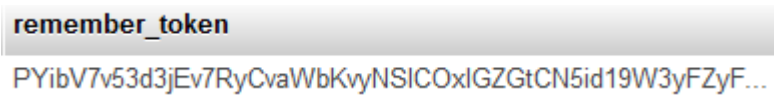


Se rappeler de moi

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

Par défaut on redirige sur « home » ou sur la valeur de la propriété **redirectTo** si elle existe. Donc pour changer la redirection par défaut il suffit de créer cette propriété 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.

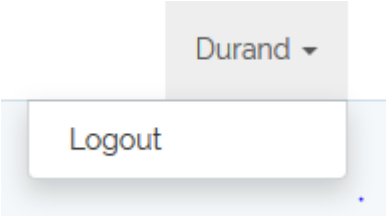
Sans entrer dans les détails la route désirée est mémorisée dans la session et voici le code pour la redirection :

```
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** :

```
POST     | logout                |        | App\Http\Controllers\Auth\LoginController@logout                | web
```

Remarquez que la méthode est POST, c’est une nouveauté dans cette version 5.3 et plus en cohérence avec le HTTP. Mais du coup un simple lien ne suffit pas. Si vous regardez dans le code du template (**resources/views/layouts/app.blade.php**) vous y trouvez ces lignes :

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

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

On a un formulaire avec la méthode POST et une soumission avec JavaScript.

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()->flush();

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

    return redirect('/');
}
```

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

La méthode **regenerate** modifie l'identifiant de session pour éviter les attaques de fixation de session.

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.
- La commande d'Artisan **make:auth** génère les contrôleurs, 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.