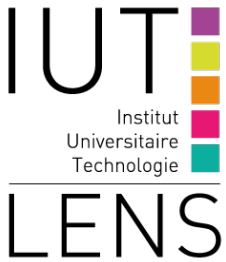
Programmation Web côté serveur

PHP Laravel Identification et Contrôle d'accès



Fred Hémery

2018/2019

IUT Lens

Département Informatique

Principe général de la sécurité

- Un mécanisme de sécurité doit garantir à chaque requête que le **sujet** est autorisé à exécuter l'**action** sur l'**objet**.

Sujet Le sujet correspond à l'identité de l'utilisateur ou de l'application obtenue après vérification des données d'authentifications (nom de login, adresse mail, . . . associé à un mot de passe certificat, . . .)

Action Une action provoque une modification dans le système d'informations (lire, écrire, modifier, supprimer)

Objet Un objet est la donnée dans le système d'informations que l'on veut sécuriser

Sécurité avec Laravel

- Laravel propose un mécanisme simple de mise en oeuvre de la sécurité.

  `php artisan make:auth`

- Ajoute Les contrôleurs : `RegisterController`, `LoginController`, `ResetPasswordController` et `ForgotPasswordController`

- Utilise le modèle `App\User`

```
CREATE TABLE IF NOT EXISTS `users` (
  `id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  `email` VARCHAR(255) NOT NULL,
  `password` VARCHAR(255) NOT NULL,
  `remember_token` VARCHAR(100)  DEFAULT NULL,
  `created_at` TIMESTAMP NULL DEFAULT NULL,
  `updated_at` TIMESTAMP NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `users_email_unique` (`email` ASC))
```

- Ajoute les vues, entre autres, `login.blade.php` et `register.blade.php` dans le répertoire `resources/views/auth` associées au template dans `resources/views/layouts`

IUT Institut Universitaire Technologique LENS

Auth façade d'accès à la sécurité

- La classe modèle `App\User` représente le sujet dans notre mécanisme de sécurité
- Les contrôleurs et les vues associées permettent de créer de nouveaux utilisateurs et aussi gérer son mot de passe.
- La classe façade `Auth` permet de
  - Vérifier qu'un utilisateur est connecté

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

if (Auth::check()) {
    // The user is logged in...
}
```

  - Récupérer les données sur l'utilisateur

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

// Get the currently authenticated user...
$user = Auth::user();

// Get the currently authenticated user's ID...
$id = Auth::id();
```

Utilisation de la façade `Auth`

- Laravel propose un filtre (une classe **middleware**) qui permet de vérifier si une route, une méthode est autorisée pour l'utilisateur identifié.

```
Route::get('profile', function () {
    // Only authenticated users may enter...
})->middleware('auth');
```

- La protection peut se faire au niveau d'un contrôleur, dans la méthode constructeur

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

- La déconnexion de l'utilisateur se fait à l'aide de l'instruction suivante :

```
    Auth::logout();
```

Les routes par défaut

- L'ajout de la ligne `Auth::routes()` dans le fichier `web.php` créer les routes suivantes :

```
// Authentication Routes
$this->get('login', 'Auth\LoginController@showLoginForm');
$this->post('login', 'Auth\LoginController@login');
$this->get('logout', 'Auth\LoginController@logout');

// Registration Routes
$this->get('register', 'Auth\RegisterController@showRegistrationForm');
$this->post('register', 'Auth\RegisterController@register');

// Password Reset Routes
$this->get('password/reset', 'Auth\
    ➥ForgotPasswordController@showLinkRequestForm');
$this->post('password/email', 'Auth\
    ➥ForgotPasswordController@sendResetLinkEmail');
$this->get('password/reset/{token}', 'Auth\
    ➥ResetPasswordController@showResetForm');
$this->post('password/reset', 'Auth\ResetPasswordController@reset');
```

L'identification avec Laravel

- Les éléments qui constituent l'identification avec Laravel
  - **guard** : indique le mécanisme utilisé pour gérer le contrôle à chaque requête. Par défaut le mécanisme utilisé est la session et les cookies. Il existe d'autres mécanismes (**passport** et les jetons)
  - **provider** : indique comment Laravel va récupérer les informations d'un utilisateur. Par défaut le mécanisme utilisé est **users** qui stocke les données dans une base de données.

- dans le fichier `config/auth.php`

```
<?php
return [
    'defaults' => [
        'guard' => 'web',
        'passwords' => 'users',
    ],
    'guards' => [
        'web' => [
            'driver' => 'session',
            'provider' => 'users',
        ],
        'api' => [
            'driver' => 'token',
            'provider' => 'users',
        ],
    ],
    ...
```

```
  ...
    'providers' => [
        'users' => [
            'driver' => 'eloquent',
            'model' => App\User::class,
        ],
    ],
    'passwords' => [
        'users' => [
            'provider' => 'users',
            'table' => '
     ➥password_resets',
            'expire' => 60,
        ],
    ],
];
```

IUT Institut Universitaire Technologique LENS

- Le contrôle d'accès peut se faire selon deux approches :

  Gate qui utilise une approche simplifiée, en général, sans être attaché à un modèle particulier mais plutôt à une vue particulière ;

  Policy qui est relié à un modèle

Gate

- On définit une Gate dans le fichier `App\Providers\AuthServiceProvider`

```
public function boot() {
    $this->registerPolicies();

    Gate::define('update-post', function ($user, $post) {
        return $user->id == $post->user_id;
    });
}
```

ou encore à l'aide de l'écriture `Classe@fonction`

```
public function boot()
{
  $this->registerPolicies();

  Gate::define('update-post', 'PostPolicy@update');
}
```

Gate Ressource

- Il est possible de définir en une fois les 4 méthodes de base CRUD

```
public function boot()
{
  $this->registerPolicies();

  Gate::resource('posts', 'PostPolicy');
}
```

Ce qui le même effet que

```
public function boot()
{
  $this->registerPolicies();

  Gate::define('posts.view', 'PostPolicy@view');
  Gate::define('posts.create', 'PostPolicy@create');
  Gate::define('posts.update', 'PostPolicy@update');
  Gate::define('posts.delete', 'PostPolicy@delete');
}
```

Gate Utilisation

- Pour gérer l'accès à l'aide d'une **Gate** on utilise les méthodes `allows`, `denies`. Dans l'exemple, l'utilisateur testé est celui qui est connecté.

```php
if (Gate::allows('update-post', $post)) {
    // The current user can update the post...
}

if (Gate::denies('update-post', $post)) {
    // The current user can't update the post...
}
```

- Si on veut tester pour un utilisateur en particulier, on utilise la méthode `forUser`.

```php
if (Gate::forUser($user)->allows('update-post', $post)) {
    // The user can update the post...
}

if (Gate::forUser($user)->denies('update-post', $post)) {
    // The user can't update the post...
}
```

Policy

- Une **Policy** est une classe qui est associée à un modèle par exemple le modèle `Post` est associé avec la politique d'accès définie dans la classe dans le fichier `PostPolicy`
- Laravel propose une commande pour générer une classe de politique d'accès pour un modèle

```
php artisan make:policy PostPolicy
```

ou pour les méthodes CRUD

```
php artisan make:policy PostPolicy  --model=Post
```

Policy Enregistrement

- Une fois créée, la politique d'accès doit être enregistrée dans le fichier `App\Providers\AuthServiceProvider`

```
<?php

use App\Post;
use App\Policies\PostPolicy;
use Illuminate\Support\Facades\Gate;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as
    ➥ServiceProvider;

class AuthServiceProvider extends ServiceProvider {
    protected $policies = [
        Post::class => PostPolicy::class,
    ];

    public function boot() {
        $this->registerPolicies();
    }
}
```

Policy Implémentation

- Avec la classe `PostPolicy` et la méthode `update` qui utilise deux paramètres :
  - L'utilisateur (`$user`) qui souhaite effectuer l'action, et
  - l'objet (`$post`) qui va être modifié

```php
<?php

namespace App\Policies;

use App\User;
use App\Post;

class PostPolicy {

    public function update(User $user, Post $post)  {
        return $user->id === $post->user_id;
    }
}
```

La méthode renvoie `true` si l'action est autorisée et `false` sinon.

- Si la méthode n'a pas besoin d'une instance de modèle, on utilise qu'un seul paramètre

```php
public function create(User $user)
{
    //
}
```

IUT Institut Universitaire Technologique LENS

Policy Utilisation

- En utilisant l'utilisateur connecté sur une instance d'un modèle

```
if ($user->can('update', $post)) {  // }
```

On vérifie que l'utilisateur à le droit d'effectuer l'action '`update`' sur l'objet `$post`.

- En utilisant l'utilisateur connecté sans instance de modèle

```
if ($user->can('create', Post::class)) {   //  }
```

- en utilisant un filtre

```
use App\Post;

Route::put('/post/{post}', function (Post $post) {
    // The current user may update the post...
})->middleware('can:update,post');

Route::post('/post', function () {
    // The current user may create posts...
})->middleware('can:create,App\Post');
```

Le filtre `can` vérifie que l'utilisateur connecté a le droit d'exécuter l'action `update` sur l'objet `post` qui correspond au paramètre (`{post}`) de la règle de routage. La méthode `create` utilise le nom de la classe comme 2ème paramètre.

Policy Utilisation (suite)

- Dans un contrôleur qui hérite de la classe `App\Http\Controllers\Controller`, il est possible d'utiliser la méthode `authorize`

```php
<?php
namespace App\Http\Controllers;

use App\Post;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller
{
    public function update(Request $request, Post $post)  {
      $this->authorize('update', $post);
        // The current user can update the blog post...
    }
    public function create(Request $request) {
      $this->authorize('create', Post::class);
        // The current user can create blog posts...
    }
}
```

En cas d'echec la méthode `authorize` génère une exception `Illuminate\Auth\Access\AuthorizationException` (renvoie une page avec le code HTTP 403)

Policy Utilisation (suite)

- Des directives sont disponibles dans le préprocesseur de templates `blade`.

```
@can('update', $post)
    <!-- The Current User Can Update
     ➥The Post -->
@elsecan('create', $post)
    <!-- The Current User Can Create
     ➥New Post -->
@endcan

@cannot('update', $post)
    <!-- The Current User Can`t Update
     ➥ The Post -->
@elsecannot('create', $post)
    <!-- The Current User Can`t Create
     ➥ New Post -->
@endcannot
```

equivalent de

```
@if (Auth::user()->can('update', $post
     ➥))
    <!-- The Current User Can Update
     ➥The Post -->
@endif

@unless (Auth::user()->can('update',
     ➥$post))
    <!-- The Current User Can`t Update
     ➥ The Post -->
@endunless
```

Si la méthode n'a pas besoin d'instance d'un modèle

```
@can('create', App\Post::class)
    <!-- The Current User Can Create Posts -->
@endcan

@cannot('create', App\Post::class)
    <!-- The Current User Can`t Create Posts -->
@endcannot
```

IUT LENS