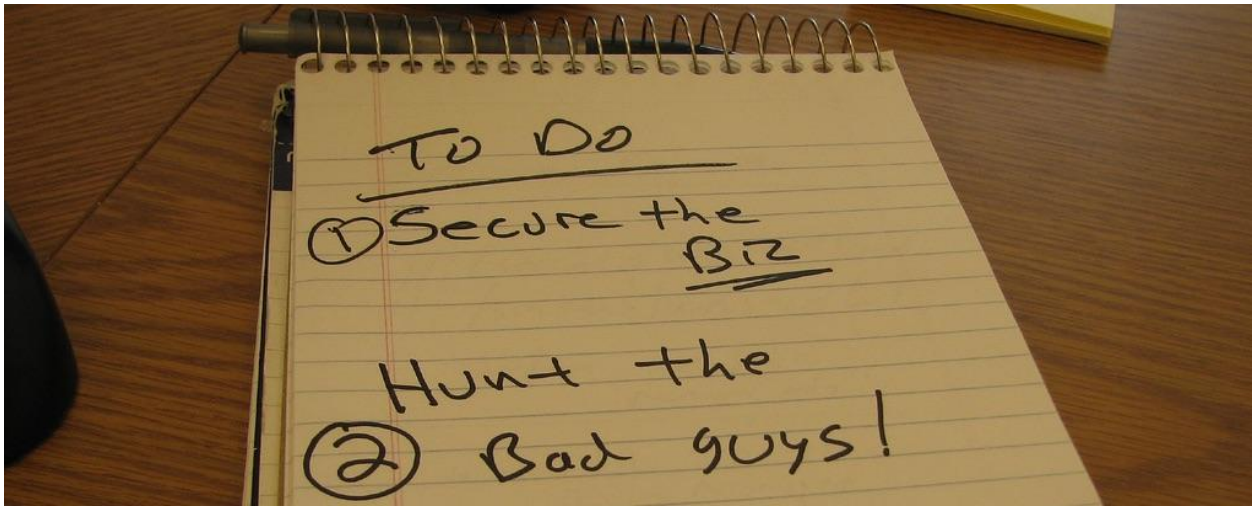

Documentation technique Projet - TODOLIST

05/03/2023

—

Implémentation de la couche de sécurité via Symfony

Dans symfony il est possible d'ajouter un système d'authentification simple login password via les commandes : "composer require symfony/security-bundle" et "php bin/console make:auth"

La première installe tous les prérequis pour la gestion de l'auth et ajoute un fichier de configuration. La deuxième commande va modifier le fichier de configuration et ajouter 3 autres fichier :

- LoginFormAuthenticator.php
- SecurityController.php
- Login.html.twig

I. LoginFormAuthenticator

On peut voir cette classe comme un guard, c'est elle qui va valider si un utilisateur à le droit ou non de se connecter à l'application. Elle va traiter les informations transmises par le formulaire de connexion /login.

II. SecurityController

Ce controller fonctionne comme un controller symfony standard. Il renvoie la vue lorsque nous souhaitons accéder aux URL.

III. Login.html.twig

Vue twig contenant le formulaire de login.

Zoom sur le fichier de configuration "config\packages\security.yaml"

Ce fichier est très important, c'est dans celui-ci que nous allons définir beaucoup de règles relatives à l'authentification.

I. Encoders

Les encoders sont utiles pour choisir avec quel algorithme le mot de passe d'utilisateur va être encodé.

```
security:
  password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: "auto"
```

II. Providers

Le provider est un lien entre la base de donné et symfony c'est ici que nous indiquons où trouver les vrai utilisateurs.

```yaml
providers:
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

III. Firewall

Le pare-feu permet de donner l'accès ou non à certaines pages. En général, un utilisateur non connecté doit pouvoir accéder à la page login. C'est dans cette section que nous allons appeler notre Guard "custom_authenticator"

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        lazy: true
        provider: app_user_provider
        custom_authenticator: App\Security\LoginFormAuthenticator
        logout:
            path: app_logout
```

IV. Gestion des accès

Il est possible d'ajouter directement dans ce fichier les accès au page en fonction des rôles.

```yaml
access_control:
    # - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }
```

Cependant dans ce projet nous avons fait le choix d'ajouter directement l'access_control sur chaque méthod dans les controllers pour plus de précision.

```php
#[Route('/users/{id}/edit', name: 'user_edit')]
#[Security("is_granted('ROLE_ADMIN') or currentUser === user", message: "Vous n'avez pas les droits suffisants")]
public function editAction(User $currentUser, Request $request)
```

Dans cet exemple pour modifier un utilisateur il faut soit être un administrateur ou alors être le bon utilisateur “owner”

V. Les Rôles

Si vous voulez ajouter un système d’héritage entre chaque rôles c’est possible. Dans notre cas, nous pouvons imaginer que tous les accès d’un Admin hérite d’un User.

```
role_hierarchy:
  ROLE_ADMIN: ROLE_USER
```