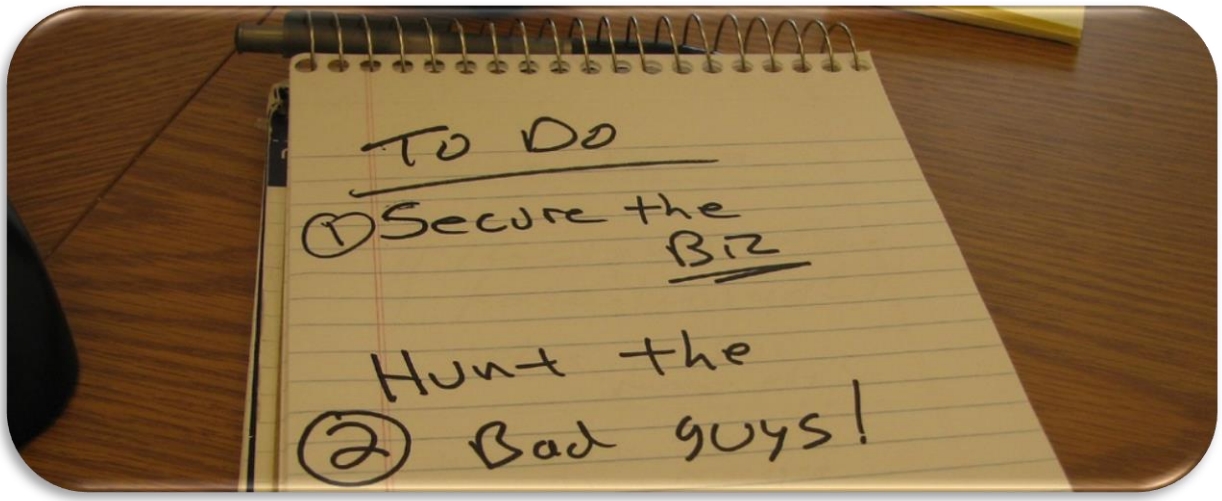


Documentation technique

Implémentation de l'authentification

ToDo & Co

Version : 1
Date de la dernière mise à jour :

SOMMAIRE

1. Introduction

La sécurité sous Symfony est une partie très importante dans notre développement. Symfony a bien séparé deux mécanismes différents : l'authentification et l'autorisation.

L'authentification est le processus qui va définir qui vous êtes, en tant que visiteur. Soit vous ne vous êtes pas identifié sur le site et vous êtes un anonyme, soit vous êtes identifié (via le formulaire d'identification ou via un cookie) et vous êtes un membre du site. Ce qui gère l'authentification dans Symfony s'appelle un « firewall ».

La configuration est réalisée dans le fichier ***security.yml*** au niveau de la clé « firewall ».

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false

    main:
        anonymous: ~
        pattern: ^/
        form_login:
            login_path: login
            check_path: login_check
            always_use_default_target_path:  true
            default_target_path:  /
        logout: ~
```

L'autorisation détermine si vous avez le droit d'accéder à la ressource demandée. Elle est gérée dans Symfony grâce à « l'access control »

2. Les utilisateurs

2.1. Définir nos utilisateurs

Les utilisateurs de notre application sont représentés par la classe ***AppBundle : User*** qui implémente l'interface « *UserInterface* **»**.

```php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

/**
 * @ORM\Table("user")
 * @ORM\Entity
 * @UniqueEntity("email")
 */
class User implements UserInterface
{
```

Cela nous a permis également de définir la classe de l'entité à utiliser pour le fournisseur permettant de charger nos entités et définir l'attribut « username » de la classe qui sert d'identifiant.

La configuration est réalisée dans le fichier ***security.yml*** au niveau de la clé « providers ».

```
providers:
    doctrine:
        entity:
            class: AppBundle:User
            property: username
```

2.2. Encodage du mot de passe

L'encodage permet de crypter le mot de passe de l'utilisateur et d'assurer une sécurité de ses données. La configuration et le choix de l'encodage est réalisé dans le fichier ***security.yml*** au niveau de la clé « encoders ». Pour notre application, nous avons décidé de choisir l'encodeur bcrypt.

```
security:
    encoders:
        AppBundle\Entity\User: bcrypt
```

3. Type d'authentification

3.1. Authentification classique

IS_AUTHENTICATED_REMEMBERED	Utilisateur authentifié automatiquement grâce au cookie « remember_me » ou le formulaire de connexion
IS_AUTHENTICATED_FULLY	Utilisateur authentifié obligatoirement via le formulaire de connexion
IS_AUTHENTICATED_ANYMOUSLY	Tous les utilisateurs même ceux qui ne sont pas authentifiés.

Nous pouvons utiliser ces rôles dans :

- Le fichier ***security.yml*** afin de sécuriser nos URL au niveau de la clé « access_control ».

```
access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
```

- Un contrôleur grâce à la méthode $this->get('security.authorization_checker')->isGranted('IS_AUTHENTICATED_REMEMBERED'))
- Une vue Twig avec {% if is_granted('IS_AUTHENTICATED_REMEMBERED') %}

3.2. Rôle des utilisateurs

La définition des rôles des utilisateurs permet la gestion des autorisations.

Un exemple : L'accès à l'URL /admin est disponible uniquement pour les utilisateurs authentifiés ayant le rôle Admin.

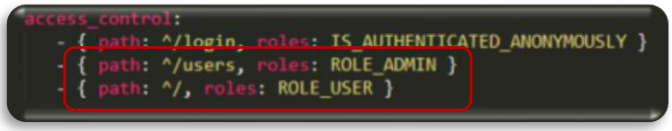
Dans notre application, nous avons défini deux rôles permettant de gérer les autorisations :

- ROLE_USER
- ROLE_ADMIN

Remarque : Pour que Symfony reconnaisse la définition d'un rôle, il faut obligatoirement que celui-ci commence par ROLE_.

Nous pouvons utiliser ces rôles dans :

- Le fichier ***security.yml*** afin de sécuriser nos URL au niveau de la clé « access_control ».



- Un contrôleur grâce à la méthode **$this->get('security.authorization_checker')->isGranted('ROLE_ADMIN'))**
- Une vue Twig avec {% if is_granted('ROLE_ADMIN') %}

Nous pouvons également définir une hiérarchie des rôles, c'est-à-dire qu'un rôle peut hériter des droits d'un autre rôle.

La définition de cette hiérarchie est définie dans le fichier « security.yml » au niveau de la clé « role_hierarchy ».

Dans notre cas par exemple, le ROLE_ADMIN hérite des droits du ROLE_USER et non le contraire.