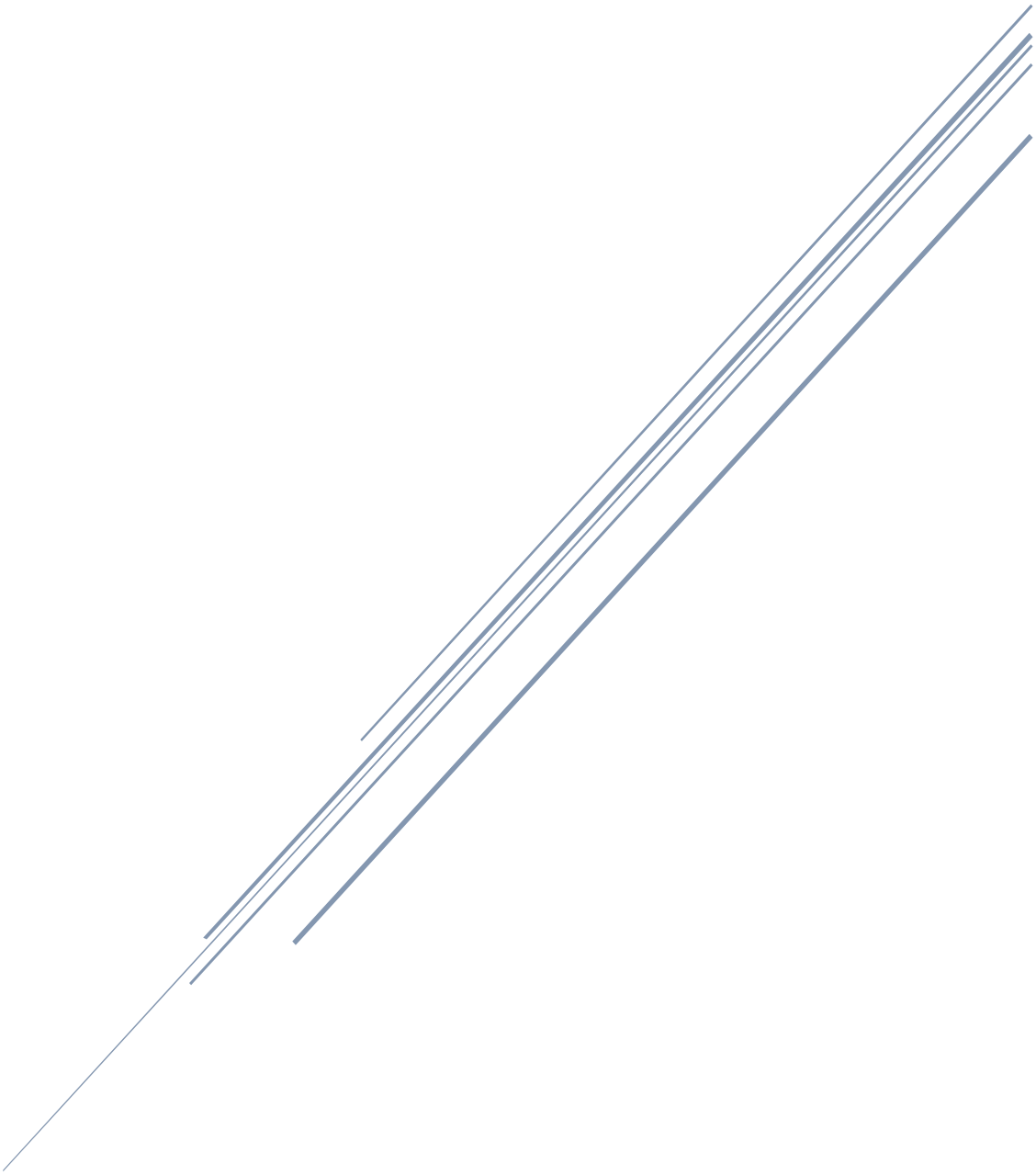
AUTHENTIFICATION SOUS SYMFONY 3.4

Projet ToDo and Co

Rédacteur : Maxime THIERRY



OpenclassRooms

Améliorez une application existante de ToDo & Co

Table des matières

Configuration initiale du fichier security.yml (authentification)

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

    providers:
        doctrine:
            entity:
                class: AppBundle\Entity\User
                property: username

    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:
                path: /logout
              target: /

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

Description du fichier :

- encoders

L'encoder permet de déterminer avec quel type d'algorithme les mots de passes vont être haché.

Dans notre cas nous avons choisi algorithme : bcrypt.

Il est similaire à la fonction PHP suivante :

```php
password_hash('password', PASSWORD_BCRYPT)
```

- providers

Le provider fournis l'utilisateur, il permet de configurer la manière dont l'authentification va être faite, dans notre cas nous utilisons le champ username de l'entité User qui a pour nom de class User.

- Firewalls

Le firewalls (pare-feu) permet entre autres de définir des configurations pour l'authentification et les autorisations.

```yaml
# app/config/security.yml

# ...
security:
    firewalls:
        secured_area:
            pattern: ^/admin
            methods: [GET, POST]

            # ...
```

Dans cet exemple nous déclarons un firewall qui a pour nom secured_area (ligne 6). Il définit le pattern suivant '^/admin', qui est une regex (expression régulière). Il est aussi possible de définir les méthodes qui seront acceptées pour accéder aux url répondant à la regex défini.

Dans cet exemple, le pare-feu ne sera activé que si le chemin commence par /admin. Si le chemin ne correspond pas à ce pattern, le pare-feu ne sera pas activé et la lecture des pares-feux suivants pourront établir une correspondance pour notre requête.

Donc l'ordre de déclaration des firewalls est important et dois se faire du plus spécifique au généraliste.

Dans notre secutity.yml nous avons déclaré un firewall main, qui permet de s'authentifier à travers un formulaire 'form_login'.

- access_control

L'access_control quant à lui, permet de définir des patterns d'URL, chacun de ces pattern est une regex, mais il ne peut en y avoir qu'un qui devra correspondre à la présente requête.

Symfony examine chaque pattern en commençant par le haut (dans l'ordre de lecture), et s'arrête dès qu'il trouve une correspondance pour l'URL.

```yaml
{ path: ^/users, roles: ROLE_ADMIN }
```

Dans cet exemple nous pouvons voir que pour accéder aux url qui commence par /users, l'authentification avec un rôle administrateur est nécessaire.

Pour plus d'informations :

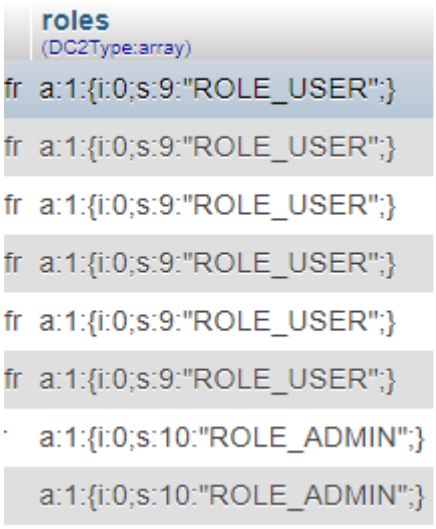
https://symfony.com/doc/3.4/security/access_control.html

Les rôles et autres autorisations

Rôles

Les rôles sont une propriété définie à la connexion de l'utilisateur,

Lorsqu'un utilisateur se connecte, il reçoit un ensemble de rôles (par exemple ROLE_ADMIN). Dans notre application les rôles sont stockés dans la basse de données et assignés en fonction de l'utilisateur qui vient de se connecter.



Vérifier si un utilisateur est connecté (IS_AUTHENTICATED_FULLY)

A présent, grâce à la mise en place des ROLES nous pouvons vérifier n'importe où dans nos Controller, si l'utilisateur connecté a le rôle user ou admin par exemple.

```php
public function helloAction($name)
{
    $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');

    // ...
}
```

Ce code permet de vérifier si un utilisateur est connecté, nous pouvons bien entendu replacer 'IS_AUTHENTICATED_FULLY' par 'ROLE_USER' pour s'assurer que l'utilisateur connecté a bien le ROLE user.

De plus, il est possible de vérifier directement dans les vues le rôle de l'utilisateur connecté exemple en twig:

```twig
{% if is_granted('ROLE_ADMIN') %}
<a href="{{ path('user_create') }}" class="btn btn-primary">Créer un
utilisateur</a>
{% endif %}
```

Dans cet exemple, le bouton n'est affiché que si l'utilisateur a le rôle ROLE_ADMIN.

Récupération de l'objet utilisateur

Nous pouvons aussi récupérer l'objet utilisateur (celui qui est connecter) de cette façon dans le code.

```
public function indexAction(TokenStorage $token)
{
    $user = $token->getUser();
}
```

Il est aussi possible de récupérer des informations directement dans le template twig comme suit :

```
{% if is_granted('IS_AUTHENTICATED_FULLY') %}
    <p>Username: {{ app.user.username }}</p>
{% endif %}
```

Déconnexion

Pour qu'un utilisateur puisse se déconnecter il faut définir la route à appeler dans le firewall,

Dans notre exemple :

Fichier security.yml

```
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: /logout
```

Fichier routing.yml

```
logout:
    path: /logout
    methods: GET
```

La route /logout doit être appeler pour permettre la déconnection de l'utilisateur, une fois utilisateur déconnecté, les informations qui était stocker sont supprimé.