L’authentification dans l’application « Todo »

1 Introduction

L’application « Todo » utilise le framework Symfony et son système d’authentification.

La majorité de la configuration de l’authentification se fait via le fichier :

« config/packages/security.yaml »

2 Providers

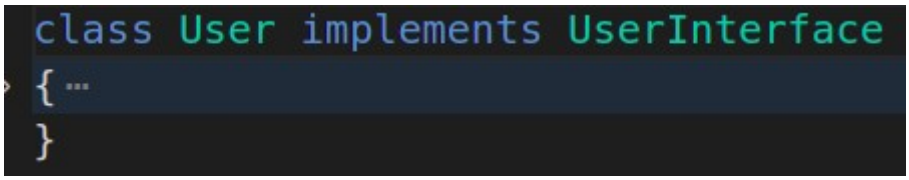
Pour configurer l’entité qui est utilisé pour représenter les utilisateur, il faut modifier le fichier « security.yaml » au niveau de la la section **provider**

On définit l’entité en renseignant l’élément **class**

On définit l’attribut de l’objet qui sera utiliser comme login en renseignant l’élément **property**

3 UserInterface

L’entité utilisé pour porter les informations d’authentification doit implémenter l’interface **« UserInterface »**



4 « Hashage » du mot de passe

En utilisant « algorithm : auto », on demande à Symfony de sélectionner automatiquement le meilleur hasher disponible (actuellement Bcrypt)

```yaml
security:
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
            algorithm: auto
            cost: 4 # Lowest possible value for bcrypt
            time_cost: 3 # Lowest possible value for argon
            memory_cost: 10 # Lowest possible value for argon
```

5 Firewalls

Le firewalls « dev » est vraiment un faux firewall : il s'assure que l'on ne bloque pas accidentellement les outils de développement de Symfony dont les URLs sont : « /_profiler » et « /_wdt ».

Dans le firewalls « main », le système de sécurité redirige les visiteurs non authentifiés vers la valeur de login_path. Ici, vers la route « login »

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        form_login:
            login_path: login
            check_path: login
        logout:
            path: logout
```

6 role_hierarchy

Les « role_hierarchy » permettent aux utilisateurs ayant un certain rôle d'hériter automatiquement d'un autre rôle.

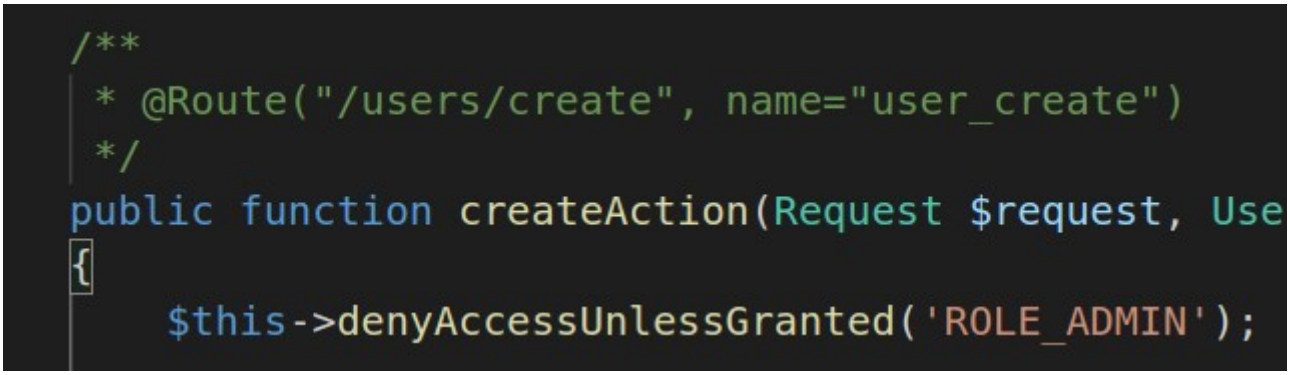
Dans l'exemple ci-dessous, tous les utilisateurs ayant le rôle « ROLE_ADMIN » héritent automatiquement du rôle « ROLE_USER ».

```yaml
role_hierarchy:
    ROLE_ADMIN: ROLE_USER
```

7 Contrôle d’accès via les contrôleur

Le contrôle d’accès a été implémenté via les contrôleurs.

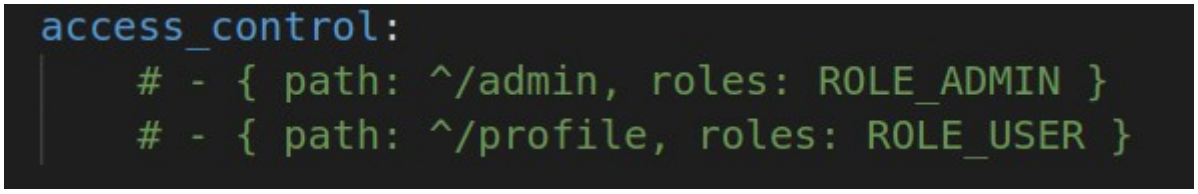
Dans l’exemple ci-dessous, l’authorisation d’accès est donné uniquement aux utilisateur qui possèdent le rôle « ROLE_ADMIN »



8 Contrôle d’accès via l’url

Le contrôle d’accès peut se faire sur les url via la configuration renseigné dans le fichier security.yaml.

Cette solution n’a pas été mis en place dans le projet mais pourra être implémenté à l’avenir :



Dans l’exemple ci-dessus, en enlevant les « # », on donnera uniquement accès :

- aux url de type **« ^/admin »** aux utilisateurs ayant le rôle **« ROLE_ADMIN »**

- aux url de type **« ^/profile »** aux utilisateurs ayant le rôle **« ROLE_USER »**