Documentation technique :
Authentification ***ToDoList***

Date : 2021-01-15

Version : 1.0

Objectif : Expliquer le fonctionnement global du système d'authentification sous l'application ToDoList.

Sommaire :

1. Introduction

L'Application ToDoList est une application php initialement développée avec la version 3.1 du framework Symfony. Dans le cadre du projet 8 du parcours de développeur d'application PHP/Symfony il nous a été demandé de :

1 - Procéder à la correction de certains bugs.

2 – Implémenter de nouvelles fonctionnalités.

Avant même de procéder à la réalisation de ces deux étapes, j'ai commencé par migrer l'application vers la dernière version LTS du Framework (SF 4.4).

C'est donc sur la base de la version 4.4 du Framework que cette documentation a été rédigée.

Dans le Framework Symfony la sécurité s'appuie sur deux mécanismes principaux :

a. L'Authentification qui permet de définir l'utilisateur géré par le pare-feu.
b. L'Autorisation qui autorise un utilisateur ou un simple visiteur à accéder aux différentes ressources de notre application.

2. L'Authentification

A- Les Utilisateurs

- L'Entité User représente un utilisateur de l'application ToDoList. Cette entité implémente l'interface UserInterface afin d'obtenir l'ensemble des méthodes nécessaires à la gestion des utilisateurs dans Symfony.
  Cette classe User est une entité Doctrine et l'identifiant unique de nos utilisateurs est le « username ». Nous retrouvons ces informations dans le fichier ***config/packages/security.yml*** sous la clé **providers** :

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

- Le mot de passe de nos utilisateurs étant stockés en base nous nous devons donc de les sécuriser. Pour cela nous utilisons un encoder ici définit sur auto dans notre fichier ***config/packages/security.yml*** sous la clé encoder :

```yaml
    encoders:
        App\Entity\User:
            algorithm: auto
```

B- Le pare Feu

- Dans notre fichier ***config/packages/security.yml*** plusieurs informations sont renseignées sous la section firewalls :

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

        main:
            anonymous: lazy
            http_basic: ~
            provider: app_user_provider
            guard:
                authenticators:
                    - App\Security\AppAuthenticator
            logout:
                path: logout
                # where to redirect after logout
                target: login
```

- Le comportement à adopter concernant le visiteur non authentifié (clé **anonymous**) avec la valeur lazy. Cela indique à Symfony de ne charger l'utilisateur (et de démarrer la session) que si l'application accède réellement à l'objet utilisateur. En d'autres termes, toutes nos URL / actions qui n'ont pas besoin de l'utilisateur seront publiques et mises en cache, améliorant ainsi les performances de notre application.

- La clé **http_basic** est ici uniquement pour permettre à un client de s'authentifier simplement dans nos tests fonctionnels.

- Le **Provider** utilisé comme expliqué ci-dessus.

- Le **Guard** qui est l'authentificateur utilisé (représenté ici par une classe) AppAuthenticator.

- La clé **path** : qui contient la route utilisée pour se déconnecter.

- La clé **target** : qui contient la route préférée (/ login) pour rediriger un utilisateur non authentifié qui souhaite accéder à une URL protégée.

C- Tester l'authentification dans votre code

Par défaut, la sécurité Symfony est utilisée pour définir si un utilisateur est authentifié. Vous pouvez utiliser certains attributs pour le vérifier :

a. IS_AUTHENTICATED_ANONYMOUSLY ce qui signifie tous les utilisateurs, même ceux qui ne sont pas authentifiés.
b. IS_AUTHENTICATED_FULLY ce qui signifie des utilisateurs authentifiés pendant la session en cours.

Utilisez ces attributs dans le fichier ***config/packages/security.yml*** pour sécuriser les modèles d'URL sous la clé **access_control** :

```yaml
# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/users, roles: ROLE_ADMIN }
    - { path: ^/tasks, roles: ROLE_USER }
    - { path: ^/, roles: IS_AUTHENTICATED_ANONYMOUSLY }
```

Remarque : Il est possible d'utiliser ces attributs dans un contrôleur, via la méthode ***$this->isGranted('IS_AUTHENTICATED_FULLY')***.

Il est également possible de les utiliser depuis une vue Twig ***{% if is_granted ('IS_AUTHENTICATED_FULLY')%}***.

Une fois qu'un Utilisateur est authentifié, vous pouvez y accéder dans un contrôleur via la méthode ***$this->getUser()*** ou dans une vue Twig via la variable globale ***app.user***.

3. Autorisation

Comme indiqué plus haut, l'autorisation, elle se charge de vérifier qui a le droit d'accéder à telle ou telle ressource. Pour se faire plusieurs éléments interviennent :

A- Les Rôles

Au sein de notre application ToDoList deux rôles distincts ont été créés :
- Le ROLE_USER
- Le ROLE_ADMIN

Chacun de ces deux rôles disposera de droits bien particuliers pour accéder aux différentes ressources de l'application. Il serait possible de créer davantage de rôles si nécessaire en respectant la nomenclature suivante : **ROLE**_*NOMDUROLE*

Une hiérarchie entre les rôles peut-être également mise en place afin d'affiner au mieux les autorisations : ainsi dans notre application Le ROLE_ADMIN se verra par défaut affecter le ROLE_USER :

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

B- Les Contrôles d'accès

Nos rôles sont utilisés sous la clé **access_control** de notre fichier ***config/packages/security.yml*** pour sécuriser nos différentes ressources à travers les URLs de notre application :

```yaml
access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/users, roles: ROLE_ADMIN }
    - { path: ^/tasks, roles: ROLE_USER }
    - { path: ^/, roles: IS_AUTHENTICATED_ANONYMOUSLY }
```

Ces contrôles d'accès peuvent s'opérer également directement depuis nos Controllers et mêmes depuis nos vues Twig.

Un dernier élément vient compléter la partie gestion des autorisations avec la mise en place d'un Voter pour sécuriser l'entité « Task ». Cette classe permet de gérer si un utilisateur à le droit de supprimer ou non une tâche au sein de l'application. Pour plus d'information sur l'utilisation des Voters, vous piuvez consulter la documentation officielle de Symfony.