ToDo&Co

Guide
d'authentification

Auteur : MONTORO Stéphane
Date : 11/2022

Contexte

Ce guide d'authentification a pour objectif d'expliquer la mise en place du système d'authentification afin de permettre aux futurs développeurs de l'application de comprendre et de savoir quel sont les fichiers à modifier et comment peuvent-ils être modifiés afin de garder une bonne maintenabilité et une bonne évolutivité dans le temps de l'application.

Introduction

L'application a été développée avec le **Framework Symfony** qui fonctionne avec un composant se prénommant « **Security** », qui nous permet de gérer à la fois le *système d'authentification* des utilisateurs mais aussi le *système d'autorisation* (gestion de l'accès à des ressources selon le rôle défini pour l'utilisateur « ROLE_ADMIN » et « ROLE_USER » par exemple)

La grande majorité des modifications/configurations réside dans le fichier **config/packages/security.yaml**.

Système d'Authentification & d'Autorisation

Les utilisateurs

Les utilisateurs, à savoir, les personnes allant utiliser l'application, auront besoin de s'authentifier dans l'application et nécessiterons une configuration qui réside principalement dans le fichier **security.yaml**.

L'utilisateur est représenté par l'entité **User**.

La classe **User** étant géré par *Doctrine* (un composant propre à Symfony), les utilisateurs se verront être stockés en base de données.

Cette configuration est possible grâce au paramètre **security.providers**.

L'authentification a besoin d'un nom d'utilisateur (**username**) et d'un mot de passe (**password**). Le paramètre **security.providers.doctrine.entity.property** permet de définir quel attribut de la classe User sera utilisé pour établir l'authentification. Dans notre cas le username.

Pour des raisons de sécurité, le mot de passe n'est jamais stocker en clair dans la base de données, il passera par un système d'encodage avant d'être enregistrer en base de données. C'est le paramètre **security.password_hashers** qui permet cela.

Symfony dispose de plusieurs algorithmes d'encodage, la clé « **auto** » du paramètre permet de dire à Symfony d'utiliser le meilleur algorithme automatiquement, ce qui nous convient amplement.

Nous utiliserons l'interface **UserPasswordHasherInterface** pour la création d'un utilisateur en base de données.

```yaml
security:
    enable_authenticator_manager: true
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
    providers:
        users_in_memory: { memory: null }
        app_user_provider:
            entity:
                class: App\Entity\User
                property: username
        doctrine:
            entity:
                class: App:User
                property: username
```

Le système d'authentification

Le composant Security utilise la définition de « **firewalls** » qui permettent de définir comment les utilisateurs sont authentifiés, le principal firewall est le firewall **main**.

```yaml
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            pattern: ^/
            provider: app_user_provider
            form_login:
                # "login" is the name of the route created previously
                login_path: login
                check_path: login
                always_use_default_target_path:  true
                default_target_path:  /
                enable_csrf: true
            logout:
                path: logout
            # https://symfony.com/doc/current/security.html#form-login
```

Le 1er paramètre du firewalls main « lazy » permet d'éviter de créer une nouvelle session s'il n'y a pas besoin d'autorisation particulière, cela permet la mise en cache de toutes les url ne nécessitant pas d'utilisateur, améliorant ainsi les performances de l'application.

L'authentification utilise le provider d'authentification de Symfony « app_user_provider » via un formulaire de connexion (form_login) donc le nom de route est spécifié par le paramètre login_path.
Le formulaire est protégé par jeton CSRF et vérifier automatiquement par Symfony.

Le paramètre default_target_path permet d'indiquer la route sur laquelle seront redirigés les utilisateurs après une authentification réussi.

Le paramètre firewalls.main.logout permet de définir le nom de route utilisé pour la déconnexion des utilisateurs.

Doc Security Symfony : *https://symfony.com/doc/current/security.html*

Gestion des Autorisations

Le système d'autorisation permet de définir les restrictions d'accès à certaines ressources de l'application en fonction des rôles utilisateur (ROLE_ADMIN et ROLE_USER) et peut être réalisée de plusieurs manière

➔ Les rôles

La classe **User** possède une méthode **getRoles()** permettant à Symfony de récupérer lors de la connexion le/les rôle(s) de l'utilisateur, stocké sous forme d'un tableau en base de données.

Les différents rôles définis pour l'application sont le rôle utilisateur « **ROLE_USER** » et le rôle administrateur « **ROLE_ADMIN** ».

Symfony propose d'autres statuts comme pour la gestion des user anonyme « **IS_AUTHENTICATED_ANONYMOUSLY** » ou celle des users pleinement authentifié « **IS_AUTHENTICATED_FULLY** ».

Une hiérarchie est mise en place grâce au paramètre « **role_hierarchy** » présent dans le fichier security.yaml est permet de signifier que le rôle administrateur possède également les droits du rôle utilisateur.

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

➔ Les restrictions d'accès

Le paramètre **access_control** défini des règles de restrictions d'accès selon le rôle de l'utilisateur.

Dans notre configuration nous spécifions :

l'url « /login » est accessibles aux utilisateurs non authentifiés.

Toutes les url commençant par « /users » ne sont accessibles qu'aux utilisateurs ayant le rôle administrateur.

L'ensemble des autres url sont accessible aux utilisateurs authentifiés avec pour role « ROLE_USER »

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

Attribut @IsGranted

La configuration au sein du fichier **security.yaml** aurait pu être mise en place au sein des contrôleurs grâce à l'attribut **@isGranted(« ROLE_ADMIN »)** ou **@isGranted(« ROLE_USER »)** provenant de **Sensio\Bundle\FrameworkExtraBundle**, soit au niveau des annotations des différentes méthodes, soit dans les vues directement à travers Twig.

Un exemple existe dans le fichier « **base.html.twig** » dans le dossier « **template** » permet de gérer l'affichage de bouton selon le rôle de l'utilisateur. (voir photo ci-dessous)

```
<div class="container">
    <div class="row">
        {% if is_granted("ROLE_ADMIN") %}
            <a href="{{ path('user_create') }}" class="btn btn-primary">Créer un utilisateur</a>
            <a href="{{ path('user_list') }}" class="btn btn-primary">Voir les utilisateurs</a>
        {% endif %}
```

Voters

Le 3ème système proposé par le composant Security pour gérer les restrictions d'accès est le système de « **Voters** »

L'implémentation se fait au niveau des controlleurs et il est mis en place pour restreindre la possibilité de supprimer une tâche a l'auteur de la tâche ou encore la possibilité de supprimé une tache anonyme seulement pour les utilisateur ayant un rôle d'administrateur.

Pour la mise en place des voters, une class TaskVoter à été créée et implémente l'interface VoterInterface, son objectif est de gérer la suppression de la tâche.

La tâche peut être supprimer dans ces cas précis :

Si l'utilisateur authentifié possède le rôle administrateur et que la tâche est liée à un utilisateur anonyme

Si l'utilisateur authentifié est l'auteur de la tâche

```
private function canDelete(Task $task, User $user): bool
{
    if ($this->security->isGranted('ROLE_ADMIN') && (null === $task->getAuthor())) {
        return true;
    }
    return $user === $task->getAuthor();
}
```