ToDo & Co – TodoList

Documentation technique

Authentification



Version: 1.0

Date de la dernière mise à jour : 6 août 2023

Ludovic Lemaître

Sommaire

Introduction

<u>Authentification et autorisation</u>

Contrôleur

Configuration

Références

Introduction

L'application ToDoList est accessible uniquement aux utilisateurs authentifiés. L'inscription d'un nouvel utilisateur se fait via un formulaire accessible à partir de la page d'accueil du site ou par le biais d'un compte administrateur.

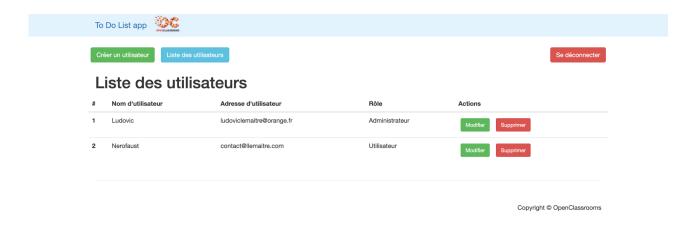
L'authentification passe par un formulaire de connexion en saisissant un identifiant (le nom d'utilisateur pour ce projet) et un mot de passe.

Un système de rôle a été mis en place pour restreindre l'accès à certaines pages de l'application :

- Seul un administrateur peut avoir accès à la gestion des utilisateurs de l'application
- L'opération de suppression d'une tâche ne peut être réalisée que par son auteur

Dans un contrôleur il est possible d'utiliser la fonction "\$this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY')" pour restreindre l'accès d'une route en particulier.

Il est également possible d'utiliser "throw \$this->createAccessDeniedException()" pour afficher un message d'erreur personnalisé.



Dans ce document vous trouverez comment a été implémenté le système d'authentification du projet sous Symfony.

Liste des fichiers utilisés :

src/Entity/User.php => Classe entité utilisateur
src/Security/SecurityController.php => Contrôleur contenant les routes de
connexion et de déconnexion
config/packages/security.yaml => Paramètrage d'authentification
templates/security/login.html.twig => Template du formulaire de connexion

Authentification et autorisation

L'entité User

L'entité correspond à un utilisateur indispensable pour le fonctionnement du système d'authentification du projet.

Dans un premier temps, l'application nécessite l'installation du bundle SecurityBundle. Ce bundle fournit toutes les fonctionnalités d'authentification et d'autorisation nécessaires pour sécuriser l'application :

composer require symfony/security-bundle

La création de l'entité "user" a été réalisé avec la commande suivante :

symfony console make:user

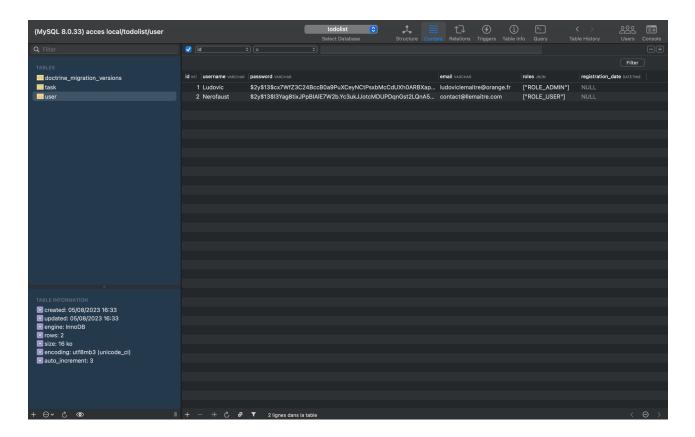
Cette commande permet de générer la classe "User", qui implémente l'interface UserInterface, en configurant différents éléments (nom de l'entité, utilisation de doctrine pour le stockage en base de données, le champ correspondant au login et l'encodage du mot de passe).

Dans l'entité « User », on retrouve notamment toutes les propriétés qui caractérisent un utilisateur avec une gestion de l'identifiant et de l'email unique.

Méthodes

La classe "User" implémentant l'interface UserInterface, l'entité doit comprendre les méthodes suivantes en plus des getters et setters :

- **getUserIdentifier()** permet de renvoyer l'identifiant qui représente un utilisateur. Dans le projet c'est l'attribut "username"
- getRoles() renvoi un tableau contenant les rôles d'un utilisateur. Par défaut, un utilisateur aura au minimum le rôle "Utilisateur" (ROLE_USER)
- eraseCredentials() permet de supprimer les données temporaires qui sont sensibles, notamment le mot de passe, contenu dans l'objet "User". Dans le projet la fonctionnalité n'étant pas utilisée, la méthode est donc vide
- getPassword() retourne le mot de passe haché de l'utilisateur
 Concernant le hachage du mot de passe, la classe implémente également l'interface "PasswordAuthenticatedUserInterface" fournie par "SecurityBundle"



Template Twig

Dans un template "Twig" il est possible d'utiliser :

```
{% if is_granted('ROLE_ADMIN') %}
  <a href="...">Liste des utilisateurs</a>
{% endif %}
```

pour afficher un lien seulement si l'utilisateur possède le rôle `ROLE_ADMIN`.

```
div class="row margin-buttons">

{% if (not app.user and 'app_user_create' != app.request.attributes.get('_route')) or is_granted('ROLE_ADMIN') %}

a href="{f path('app_user_create') }}" class="btn btn-success">Créer un utilisateur</a>

{% elseif is_granted('ROLE_USER') %}

a href="{f path('app_user_edit', f'id' : app.user.id}) }}" class="btn btn-primary">Mon compte</a>

{% endif %}

{% endif %}

a href="{f path('app_user_list') }}" class="btn btn-info">Liste des utilisateurs</a>

{% endif %}

{% endif %}

a href="{f path('app_logout') }}" class="pull-right btn btn-danger">Se déconnecter</a>

{% endif %}

{% if not app.user and 'app_login' != app.request.attributes.get('_route') %}

a href="{f path('app_login') }}" class="btn btn-success">Se connecter</a>

{% endif %}

{% endif %}
```

Contrôleur

Le contrôleur SecurityController

C'est le contrôleur qui comprend les routes pour l'authentification et la déconnexion :

- app_login : Affiche la page du formulaire de connexion
- **app_login_check**: Récupère les listeners pour le traitement du formulaire et permet l'authentification grâce à l'authenticator de base de Symfony
- app_logout : Utilisée pour la déconnexion

Configuration

Le fichier de configuration security.yaml

Le fichier security.yaml décrit les règles d'authentification et d'autorisation de l'application. Il comprend en partie les sections suivantes :

- **password_hashers :** Indique le hacheur de mot de passe à utiliser avec le format "auto" (Symfony choisira le niveau le plus élevé possible)
- providers: Indique comment (re)charger les utilisateurs à partir d'un stockage sur la base d'un "identifiant d'utilisateur". La configuration du projet utilise Doctrine pour charger l'entité "User" en utilisant la propriété "username" comme "identifiant d'utilisateur"
- **firewalls**: La partie essentielle du processus de sécurisation. C'est ce qui permet de définir quand il faut vérifier et authentifier un utilisateur. Pour la route "main" on indique qu'il faut utiliser le "provider" fournissant nos utilisateurs et le mode d'authentification choisi. Ici, on passe par un formulaire de connexion (form_login) en renseignant le path auquel un visiteur sera redirigé automatiquement lorsqu'il tente d'accéder à une page sécurisée de l'application (login_path). On précise également qu'on utilise un jeton CSRF pour la validation du formulaire de connexion.

```
firewalls:
       security: false
   main:
      lazy: true
       provider: app_user_provider
       # activate different ways to authenticate
       # https://symfony.com/doc/current/security.html#the-firewall
       # https://symfony.com/doc/current/security/impersonating_user.html
       # switch_user: true
       form login:
          # "app_login" is the name of the route created previously
          login_path: app_login
           check_path: app_login_check
           enable_csrf: true
       logout:
       entry_point: form_login
       access_denied_handler: App\Security\AccessDeniedHandler
```

- access_control: Permet de définir pour chaque pattern d'URL quel rôle peut y accéder. Il est possible de définir autant de modèles d'URL que l'on souhaite, mais un seul sera trouvé par requête. Symfony démarre en haut de la liste et s'arrête lorsqu'il trouve la première correspondance.

```
# 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: PUBLIC_ACCESS }

- { path: ^/users/create, roles: PUBLIC_ACCESS }

- { path: ^/, roles: ROLE_USER }
```

Il est également possible de rétreindre l'accès pour chaque route depuis les contrôleurs en utilisant les attributs :

```
#@ludoviclemaitre <contact@llemaitre.com> +1
#[Route('/users', name: 'app_user_list', methods: ['GET'])]
#[IsGranted('ROLE_ADMIN')]
public function listAction(): Response
{
}
```

Références

- Lien de la documentation Symfony "Security"
- La section "Authenticating Users" du composant "Security"
- Le lien de la page "<u>Comment utiliser les Voters pour vérifier les autorisations</u> des utilisateurs" du composant "Security"