

ToDo & Co – ToDoList

## Documentation technique

Authentication



Version : 1.0

Date de la dernière mise à jour : 6 août 2023

Ludovic Lemaître

# Sommaire

[Introduction](#)

[Authentification et autorisation](#)

[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.

To Do List app



Créer un utilisateur

Liste des utilisateurs

Se déconnecter

### Liste des utilisateurs

#	Nom d'utilisateur	Adresse d'utilisateur	Rôle	Actions
1	Ludovic	ludoviclemaitre@orange.fr	Administrateur	<div>Modifier</div> <div>Supprimer</div>
2	Nerofaust	contact@ilemaitre.com	Utilisateur	<div>Modifier</div> <div>Supprimer</div>

Copyright © OpenClassrooms

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** => Paramétrage 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"

(MySQL 8.0.33) acces local/todolist/user

todoist

Select Database

Structure Content Relations Triggers Table Info Query Table History Users Console

Filter

TABLES

- doctrine\_migration\_versions
- task
- user

TABLE INFORMATION

- created: 05/08/2023 16:33
- updated: 05/08/2023 16:33
- engine: InnoDB
- rows: 2
- size: 16 ko
- encoding: utf8mb3 (unicode\_ci)
- auto\_increment: 3

id	username	password	email	roles	registration_date
1	Ludovic	\$2y\$13\$cX7WfZ3C24BccB0a9PuXCeyNctPsbMcCdUXh0ARBXap...	ludoviclemaitre@orange.fr	["ROLE_ADMIN"]	NULL
2	Nerofaust	\$2y\$13\$I3Yag8tixJppBIAIE7W2b.Yc3ukJJotcMDUPDqnGst2LQnA5...	contact@llemaitre.com	["ROLE_USER"]	NULL

2 lignes dans la table

## 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'.

```
29 <div class="row margin-buttons">
30     {% if (not app.user and 'app_user_create' != app.request.attributes.get('_route')) or is_granted('ROLE_ADMIN') %}
31         <a href="{{ path('app_user_create') }}" class="btn btn-success">Créer un utilisateur</a>
32     {% elseif is_granted('ROLE_USER') %}
33         <a href="{{ path('app_user_edit', {'id' : app.user.id}) }}" class="btn btn-primary">Mon compte</a>
34     {% endif %}
35     {% if is_granted('ROLE_ADMIN') %}
36         <a href="{{ path('app_user_list') }}" class="btn btn-info">Liste des utilisateurs</a>
37     {% endif %}
38     {% if app.user %}
39         <a href="{{ path('app_logout') }}" class="pull-right btn btn-danger">Se déconnecter</a>
40     {% endif %}
41     {% if not app.user and 'app_login' != app.request.attributes.get('_route') %}
42         <a href="{{ path('app_login') }}" class="btn btn-success">Se connecter</a>
43     {% endif %}
44 </div>
```

## 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.

```

13  firewalls:
14      dev:
15          pattern: ^/(_(profiler|wdt)|css|images|js)/
16          security: false
17      main:
18          lazy: true
19          provider: app_user_provider
20
21          # activate different ways to authenticate
22          # https://symfony.com/doc/current/security.html#the-firewall
23
24          # https://symfony.com/doc/current/security/impersonating_user.html
25          # switch_user: true
26
27          form_login:
28              # "app_login" is the name of the route created previously
29              login_path: app_login
30              check_path: app_login_check
31              enable_csrf: true
32
33          logout:
34              path: app_logout
35
36          entry_point: form_login
37          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.

```

39  # Easy way to control access for large sections of your site
40  # Note: Only the *first* access control that matches will be used
41  access_control:
42      - { path: ^/login, roles: PUBLIC_ACCESS }
43      - { path: ^/users/create, roles: PUBLIC_ACCESS }
44      - { path: ^/, roles: ROLE_USER }

```

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

```

30  @ludoviclemaitre <contact@llemaitre.com> +1
31  #[Route('/users', name: 'app_user_list', methods: ['GET'])]
32  #[IsGranted('ROLE_ADMIN')]
33  public function listAction(): Response
34  {

```

## Références

- Lien de la documentation Symfony "[Security](#)"
- La section "[Authenticating Users](#)" du composant "Security"
- Le lien de la page "[Comment utiliser les Voters pour vérifier les autorisations des utilisateurs](#)" du composant "Security"