L’authentification sur le site Todolist

Contenu

[Objectif 1](#_Toc488399706)

[Principe du système d’authentification 1](#_Toc488399707)

[La classe « User » 1](#_Toc488399708)

[La configuration de la sécurité 2](#_Toc488399709)

[La connexion 3](#_Toc488399710)

[La déconnexion 4](#_Toc488399711)

[La création et la modification d’un utilisateur 5](#_Toc488399712)

[La création 5](#_Toc488399713)

[La modification 6](#_Toc488399714)

[Les instructions pour connaitre l’utilisateur connecté à l’application 6](#_Toc488399715)

# Objectif

Ce document a pour objectif de présenter le fonctionnement actuel de l’authentification et de la gestion des autorisations de l’application « TodoList ».

# Principe du système d’authentification

L’authentification est basée sur le composant « Security » de Symfony. La documentation complète du composant est disponible ici : <http://symfony.com/doc/current/security.html>

Le composant « Security » offre deux fonctionnalités importantes :

* L’authentification qui permet de connaître l’identité de l’utilisateur,
* L’autorisation qui gère l’accès aux différentes ressources de l’application.

L’authentification vérifie l’identité de l’utilisateur en se fondant sur son nom (« username ») et son mot de passe (« password »).

L’autorisation utilise la notion de rôle associé à un utilisateur pour gérer les accès aux différentes fonctionnalités ou routes du site.

Notion de rôle : à la connexion, chaque utilisateur se voit attribuer un ou plusieurs rôles.

# La classe « User »

L’authentification pour fonctionner nécessite une classe « User » qui implémente l’interface « UserInterface »

use Symfony\Component\Security\Core\User\UserInterface;

…

class User implements UserInterface

{

….

}

Définir cette classe comme « user provider », en enregistrant le paramétrage suivant dans le fichier : app/config/security.yml

security:

....

providers:

doctrine:

entity:

class: AppBundle:User

property: username

Dans l’application « TodoList », le fichier source de la classe « User » est défini ici par : AppBundle\Entity\User.

L’utilisateur est identifié par son nom (« username »), cette information doit être unique :

/\*\*

\* @ORM\Column(type="string", length=25, unique=true)

\* @Assert\NotBlank(message="Vous devez saisir un nom d'utilisateur.")

\*/

private $username;

Tous les utilisateurs héritent du rôle « ROLE\_USER » en plus de ces rôles définis en base de données

public function getRoles()

{

$roles = $this->roles;

// give everyone ROLE\_USER!

if (!in\_array('ROLE\_USER', $roles)) {

$roles[] = 'ROLE\_USER';

}

return $roles;

}

La classe « user » est stockée par Doctrine en base de données, dans la table « user » définie comme ceci :

* id : qui représente l’identifiant unique de l’utilisateur dans le site
* username : nom de l’utilisateur
* password : mot de passe crypté de l’utilisateur
* email : son mail
* roles : définit les droits de l’utilisateur sur le site.

# La configuration de la sécurité

Le fichier « security.yml » contient la configuration de la sécurité du site.

C’est ici, que vont être paramétré, le cryptage des mots de passe, la classe « User » qui définit les utilisateurs pouvant se connecter à l’application, le comportement de la page de connexion et les règles d’accès aux différentes routes.

security:

encoders:

Définition du cryptage utilisé pour crypter les mots de passe

AppBundle\Entity\User: bcrypt

Paramétrage de la classe php sur laquelle se base l’authentification

providers:

doctrine:

entity:

class: AppBundle:User

property: username

…

firewalls:

dev:

pattern: ^/(\_(profiler|wdt)|css|images|js)/

security: false

main:

anonymous: ~

pattern: ^/

Paramétrage du formulaire de login

form\_login:

login\_path: login

check\_path: login\_check

always\_use\_default\_target\_path: true

default\_target\_path: /

logout: ~

Paramétrage des règles d’accès au site

access\_control:

Les routes /login… sont accessibles à tout le monde

- { path: ^/login, roles: IS\_AUTHENTICATED\_ANONYMOUSLY }

Les routes /users sont accessibles uniquement aux utilisateurs ayant un rôle admin

- { path: ^/users, roles: ROLE\_ADMIN }

Les routes / sont accessibles uniquement aux utilisateurs ayant un rôle user

- { path: ^/, roles: ROLE\_USER }

# La connexion

Pour permettre à un visiteur de se connecter au site, il faut disposer d’un formulaire de connexion.

La connexion se divise en deux actions, la première affiche le formulaire de connexion et la deuxième réceptionne la soumission du formulaire.

La route ‘/login’ permet d’afficher le formulaire de connexion. Le service « security.authentification\_utilis » permet d’afficher le dernier « username » utilisé et l’éventuelle dernière erreur de connexion.

/\*\*

\* @Route("/login", name="login")

\*/

public function loginAction(Request $request)

{

$authenticationUtils = $this->get('security.authentication\_utils');

$error = $authenticationUtils->getLastAuthenticationError();

$lastUsername = $authenticationUtils->getLastUsername();

return $this->render('security/login.html.twig', array(

'last\_username' => $lastUsername,

'error' => $error,

));

}

Le formulaire de connexion renvoie la réponse sur la route /login\_check (path(‘login\_check’)), avec les variables « \_username » et « \_password ».

{% extends 'base.html.twig' %}

{% block body %}

{% if error %}

<div class="alert alert-danger" role="alert">{{ error.messageKey|trans(error.messageData, 'security') }}</div>

{% endif %}

<form action="{{ path('login\_check') }}" method="post">

<label for="username">Nom d'utilisateur :</label>

<input type="text" id="username" name="\_username" value="{{ last\_username }}" />

<label for="password">Mot de passe :</label>

<input type="password" id="password" name="\_password" />

<button id="btn-login" class="btn btn-success" type="submit">Se connecter</button>

</form>

{% endblock %}

Le formulaire est envoyé à la route /login\_check pour validation de l’authentification.

Actuellement l’authentification est faite par le composant « Security », mais il est possible de redéfinir le processus d’authentification, en surchargeant la classe abstraite « AbstractFormLoginAuthenticator » .

# La déconnexion

Pour se déconnecter il suffit de faire un ‘GET’ sur la route /logout et l’application renvoie le visiteur sur la page de login.

# La création et la modification d’un utilisateur

Pour créer un nouvel utilisateur il suffit de créer une instance de la classe « User » et d’enregistrer cette instance en base de données, en utilisant « entityManager » du composant doctrine.

Actuellement, c’est le contrôleur « UserController » qui se charge de la création et de l’édition des utilisateurs.

## La création

L’action ‘createAction’ reliée à la route ‘users/create’ permet de créer un nouvel utilisateur.

Il faut créer une instance $user de classe User

$user = new User();

Il faut instancier un formulaire de saisie relié à l’instance $user

$form = $this->createForm(UserType::class, $user);

Si le formulaire de saisie est validé

$form->handleRequest($request);

if ($form->isValid()) {

$em = $this->getDoctrine()->getManager();

Alors le mot de passe est crypté

$password = $this->get('security.password\_encoder') ->encodePassword($user, $user->getPassword());

$user->setPassword($password);

Les rôles saisis sont associés à l’utilisateur

$role = $form->get('roles')->getData();

$roles = [];

foreach ($role as $key => $value) {

$roles[] = $value;

}

$user->setRoles($roles);

L’action se termine par la sauvegarde en base de données du nouvel utilisateur :

$em->persist($user);

$em->flush();

$this->addFlash('success', "L'utilisateur a bien été ajouté.");

return $this->redirectToRoute('user\_list');

}

Affichage de la page de connexion :

return $this->render('user/create.html.twig', ['form' => $form->createView()]);

## La modification

L’action ‘editAction’ reliée à la route /users/{id}/edit permet de modifier les rôles d’un utilisateur déjà existant.

On instancie un formulaire de saisi prérempli.

       $form = $this->createForm(UserEditType::class, $user);

Si le formulaire est validé

$form->handleRequest($request);

if ($form->isValid()) {

Les rôles saisis sont affectés à l’utilisateur

$role = $form->get('roles')->getData();

$roles = [];

foreach ($role as $key => $value) {

$roles[] = $value;

}

$user->setRoles($roles);

Les modifications sont enregistrées

$this->getDoctrine()->getManager()->flush();

$this->addFlash('success', "L'utilisateur a bien été modifié");

return $this->redirectToRoute('user\_list');

}

Affichage de la page d’édition d’un utilisateur :

return $this->render('user/edit.html.twig', ['form' => $form->createView(), 'user' => $user]);

}

# Les instructions pour connaitre l’utilisateur connecté à l’application

Au sein de l’application, il est utile de connaitre l’utilisateur ou ses droits

L’instruction suivante permet de connaître à tout moment l’utilisateur connecté

$this->container->get('security.token\_storage')

->getToken()->getUser();

L’instruction suivante permet de connaître dans une vue twig l’utilisateur connecté

app.user

L’instruction suivante permet de tester les droits de l’utilisateur connecté

$this->get('security.authorization\_checker')

->isGranted('ROLE\_ADMIN')

L’instruction suivante permet de tester les droits de l’utilisateur connecté dans une vue twig

is\_granted("ROLE\_ADMIN")