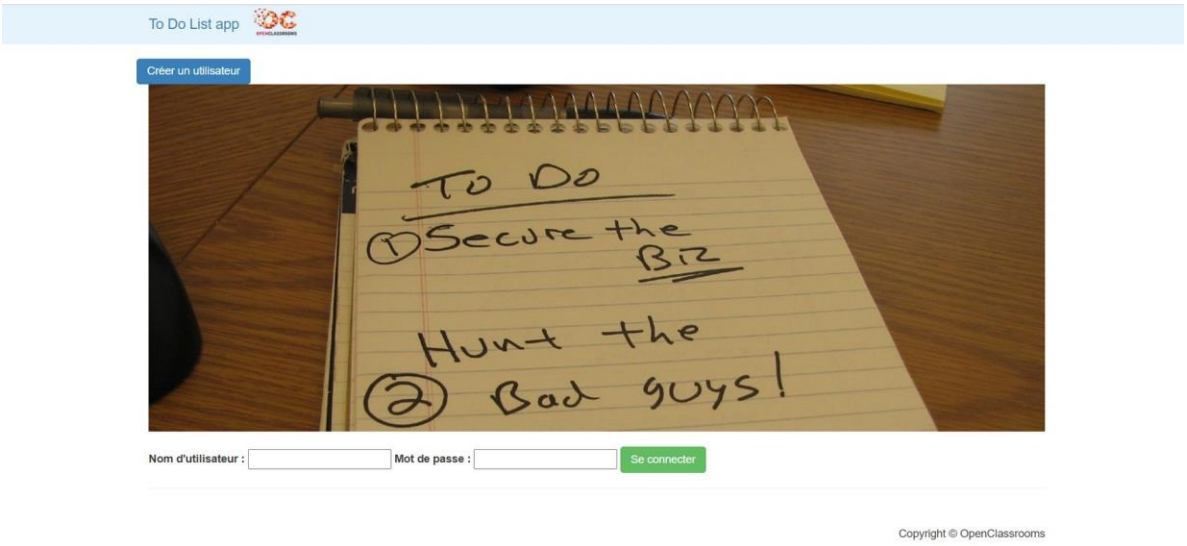
Documentation technique sur l’authentification

Sommaire

Avant tout chose, il faut savoir que tous les utilisateurs enregistrés dispose du rôle user et qu'un utilisateur non authentifié est obligatoirement redirigé sur la page de login.



Le composant de sécurité de Symfony :

Le composant de Sécurité de Symfony est un outil que nous offre le Framework afin de gérer facilement la sécurité de notre application.

En effet ce composant nous permet de gérer l' authentification, à savoir d'où proviennent les utilisateurs et comment gérer l'authentification.

Il permet également de mettre en place un système d' autorisations, afin de gérer l'accès à certaines ressources selon le rôle défini de l'utilisateur (visiteur non authentifié, utilisateur authentifié ou administrateur).

L'entité user :

L'utilisateur est représenté par la classe User.

```php
/**
 * @ORM\Table("user")
 * @ORM\Entity
 * @UniqueEntity("username")
 * @UniqueEntity("email")
 */
class User implements UserInterface
{

    /**
     * @ORM\Column(type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
```

Si l'on veut que le composant de Sécurité de Symfony comprenne et puisse utiliser notre entité User comme celle qui représente les utilisateurs de notre système, il faut que l'entité User implémente une interface particulière : la User Interface.

Elle hérite donc des fonctions de cette classe qui sont essentielles à la gestion des utilisateurs.

De plus il est à noter qu'une contrainte d'unicité a été appliquée sur l'attribut "email" et "username".

Enfin les utilisateurs de l'application pourront créer des tâche : on leur a donc donné une relation 1 à plusieurs avec l'entité Task.

Encodage des mots de passe :

```php
/**
 * @Route("/users/create", name="user_create")
 */
public function createAction(Request $request, UserPasswordEncoderInterface $encoder)
{
    $user = new User();
    $form = $this->createForm( type: UserType::class, $user);

    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $password = $encoder->encodePassword($user, $user->getPassword());
        $user->setPassword($password);


        $em->persist($user);
        $em->flush();

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

        return $this->redirectToRoute( route: 'user_list');
    }
```

Il est absolument proscrit d'avoir des mots de passe en clair dans une base de données dans le cas où celle-ci viendrait à être compromise, il a donc fallu encoder les mots de passe : nous avons donc utilisé la UserPasswordEncoderInterface de Symfony.

La même méthode a bien sûr été utilisée aussi pour la modification des utilisateurs.

Le fichier security.yaml :

Tous les paramètres concernant la sécurité du framework, notamment pour l'autorisation et pour l'authentification sont regroupés au sein du fichier **config/packages/security.yaml**

C'est dans ce fichier que la sécurité est entièrement configurée.

Les utilisateurs inscrits sont stockés dans la base de données de l'application, dans la table user.

Les « providers » permettent d'indiquer à notre application comment trouver ces utilisateurs. Dans notre cas, nous utilisons un unique provider, de type entity, qui correspond à notre entité User (définie dans src/Entity/User.php).

L'option property permet d'indiquer quelle propriété de l'entité User sert d'identifiant pour l'authentification.

```
security:
    encoders:
        App\Entity\User: bcrypt

    providers:
        doctrine:
            entity:
                class: App:User
                property: username
```

L'encoder indique quel encodage est utilisé pour crypter les mots de passe des utilisateurs avant de les stocker.

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

        main:
            anonymous: ~
            pattern: ^/
            form_login:
                login_path: login
                check_path: login_check
                always_use_default_target_path:  true
                default_target_path:  /
            logout:
                path: logout
```

C'est le firewall qui prend en charge l'authentification: Ils permet de définir les différentes authentifications possibles sur l'application : Ainsi il faudra que le visiteur soit authentifié pour que le firewall l'autorise à passer.

Les firewalls permettent également de configurer la route du formulaire d'identification avec form_login et la route pour se déconnecter de l'application avec le logout.

Par contre les autorisations sont prises en charge par l'access_control.

Ce dernier permet de définir les différentes authentifications possibles sur l'application : Nous pouvons restreindre l'accès à certaines parties du site uniquement aux visiteurs qui sont membres par exemple.

Autrement dit, il faudra que le visiteur soit authentifié pour que l'application l'autorise à passer.

```
27      access_control:
28          - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
29          - { path: ^/users, roles: ROLE_ADMIN }
30          - { path: ^/, roles: ROLE_USER }
```

Voici donc l'access_control qui nous permet de définir les accès aux différentes routes de notre application.

Les routes *login* sont accessibles aux utilisateurs non authentifiés.
La route *users* est accessible seulement aux administrateurs alors que le reste de l'application est accessible aux utilisateurs ayant le rôle user.