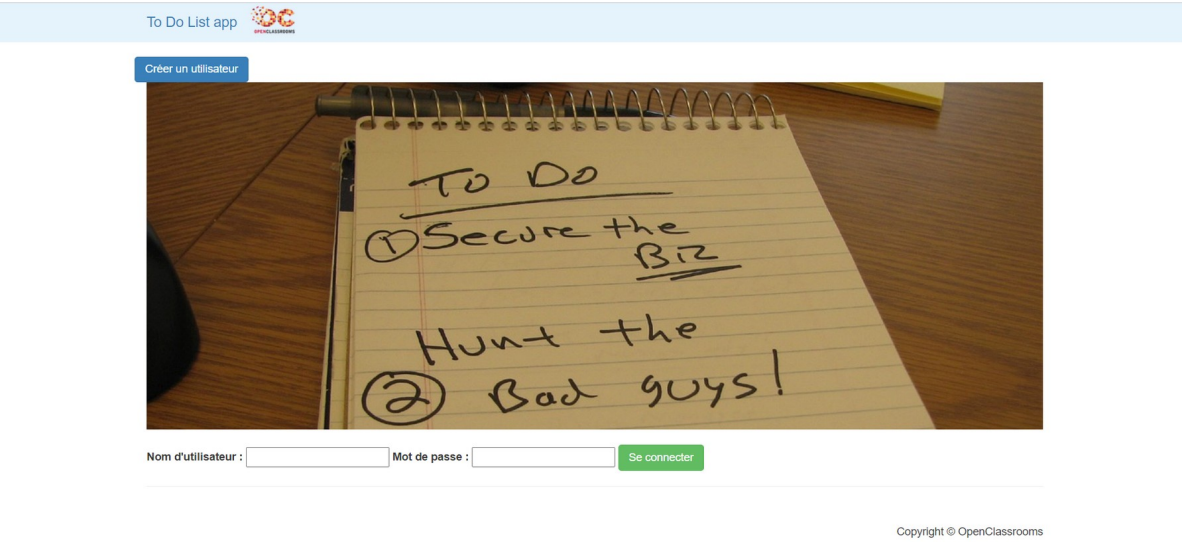
Documentation technique sur l'authentification

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.

src > Entity > User.php > PHP Intelephense > User

```
11
12  /**
13   * @ORM\Table("user")
14   * @ORM\Entity(repositoryClass=UserRepository::class)
15   * @UniqueEntity("email")
16   */
17  class User implements UserInterface
18  {
19      /**
20       * @ORM\Column(type="integer")
21       * @ORM\Id
22       * @ORM\GeneratedValue(strategy="AUTO")
23       */
24      private $id;
25
```

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 UserInterface.

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

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

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

Encodage des mots de passe :

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.

src > Controller > UserController.php > ...

```php
    /**
     * @Route("/users/create", name="user_create")
     */
    public function createAction(Request $request, UserPasswordEncoderInterface $encoder)
    {
        $user = new User();
        $form = $this->createForm(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('success', "L'utilisateur a bien été ajouté.");

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

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

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.

Pour que le composant de Sécurité comprenne que nos utilisateurs viennent de la base de données, il faut le lui préciser en créant ce qu'on appelle un provider (un fournisseur de données utilisateurs).

config > packages > ! security.yaml

```yaml
security:
    encoders:
        App\Entity\User:
            algorithm: bcrypt
    # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
    providers:
        users_in_memory: { memory: null }
        users_in_database:
            entity:
                class: App\Entity\User
                property: username
```

Les encoders vont encoder les mots de passe des utilisateurs : ainsi nous pouvons indiquer à l'application quel encodage nous allons utiliser pour le mot de passe : bcrypt est l'un des plus courant et est une référence dans le monde de la sécurité.

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

            provider: users_in_database

            form_login:
                login_path: login
                check_path: login_check

            logout:
                path: /logout
                target: /
```

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.

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

config > packages > ! security.yaml

```yaml
37
38      # Easy way to control access for large sections of your site
39      # Note: Only the *first* access control that matches will be used
40      access_control:
41          # - { path: ^/admin, roles: ROLE_ADMIN }
42          # - { path: ^/profile, roles: ROLE_USER }
43          - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
44          - { path: ^/users/create, roles: IS_AUTHENTICATED_ANONYMOUSLY }
45          - { path: ^/users, roles: ROLE_ADMIN }
46          - { path: ^/, roles: ROLE_USER }
47
48
```

Les routes *login et users/*create 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.

La suppression des tâches :

Pour gérer la suppression des tâches dans l'application, nous avons décidé d'utiliser le système des **Voters**.

Ce système fait parti du composant Security du framework et permet de gérer les restrictions d'accès.

src > Security > Voter > TaskVoter.php > PHP Intelephense > TaskVoter > supports

```php
29
30      protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
31      {
32          $user = $token->getUser();
33
34          if (!$user instanceof UserInterface) {
35              return false;
36          }
37
38          switch ($attribute) {
39
40              case 'DELETE':
41
42                  if ($subject->getUser() === $user) {
43                      return true;
44                  }
45
46                  if ($this->security->isGranted('ROLE_ADMIN')
47                      && $subject->getUser()->getUsername() === "anonyme") {
48                      return true;
49                  }
50          }
51      }
```

Pour ce faire, une classe TaskVoter a été créée et implémente l'interface VoterInterface : Cette dernière se trouve dans le dossier src.

Elle a pour objectif d'autoriser la suppression de la tâche seulement :

- Si l'utilisateur authentifié est administrateur et que la tâche en question est liée à un utilisateur "anonyme".
- Si l'utilisateur authentifié est bien l'auteur de la tâche.

Il est possible dans un futur proche de retravailler cette classe afin d'ajouter des autorisations concernant la vue des tâches ou leur modification.

Ensuite il faut ajouter une ligne de code dans le controlleur dédié, pour lui indiquer que l'action de supprimer une tâche est soumise aux autorisations présentes dans le Voter : Dans le cas présent à l'aide de denyAccesUnlessGranted.

src > Controller > TaskController.php > ...

```php
    /**
     * @Route("/tasks/{id}/delete", name="task_delete")
     */
    public function deleteTaskAction(Task $task)
    {
        $this->denyAccessUnlessGranted('DELETE', $task);
        $em = $this->getDoctrine()->getManager();
        $em->remove($task);
        $em->flush();

        $this->addFlash('success', 'La tâche a bien été supprimée.');

        return $this->redirectToRoute('task_list');
    }
}
```

Il est aussi possible de réalisé la même action avec des annotations : Pour plus d'informations, veuillez vous référer à la documentation officielle de Symfony concernant les Voters.