

## Implémentation de l'authentification

<b>1/ Classe User</b>	<b>2</b>
a/ Classe User	2
b/ Stokage d'informations sur les utilisateurs	3
c/ Classe UserRepository	3
d/ Configuration de l'entité User	3
<b>2/ Authentification</b>	<b>4</b>
a/ Classe LoginFormController	4
b/ Configuration	5
c/ SecutrityController	5
d/ Template Login.html.twig	5
<b>3/ Autorisation</b>	<b>7</b>
a/ TaskVoter	7
b/ UserVoter	8
c/ IsGranted	9

## 1/ Classe User

Ressource : <https://symfony.com/doc/current/security.html>

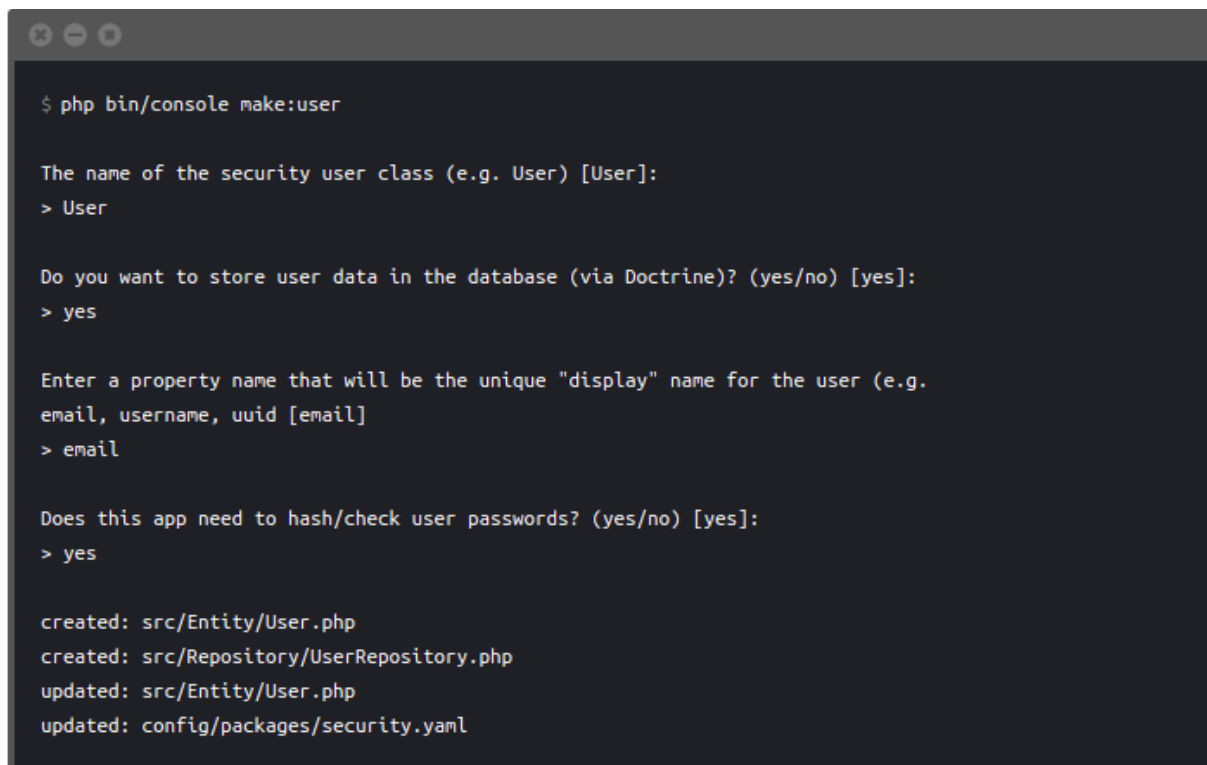
Pour notre application ToDo & Co, nous avons besoin de s'authentifier.

Installer le bundle de sécurité grâce à la commande :

- composer require symfony/security-bundle

Créer la classe utilisateur grâce à la commande :

- php bin/console make:user



```
$ php bin/console make:user

The name of the security user class (e.g. User) [User]:
> User

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
> yes

Enter a property name that will be the unique "display" name for the user (e.g.
email, username, uuid) [email]:
> email

Does this app need to hash/check user passwords? (yes/no) [yes]:
> yes

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml
```

## a/ Classe User

Elle doit implémenter les interfaces suivantes :

- Symfony\Component\Security\Core\User\UserInterface,
- Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface.

Les différents getters permettent de récupérer les valeurs des attributs correspondants.

Les différents setters permettent de modifier les valeurs des attributs correspondants.

Une relation d'un à plusieurs (OneToMany) a été faite avec l'entité task (tâche) permettant :

- de retourner une à plusieurs tâches (getTasks() ) associée(s) à un utilisateur,
- d'ajouter une tâche (addTask()),
- de supprimer la ou les tâche(s) associée(s) à l'utilisateur (removeTask()).

La méthode eraseCredentials() efface les credentials, méthode non utilisés dans l'application actuellement.

## **b/ Stokage d'informations sur les utilisateurs**

L'entité User possède des attributs :

- id : identifiant auto-incrémenté,
- username : prénom et nom, avec un lenght = 25 et une clé unique,
- email : adresse email, avec un lenght = 60 et une clé unique,
- password : mot de passe hashé avec un lenght = 255,
- rôles : tableau représentant le ROLE\_USER ou le ROLE\_ADMIN.

A l'aide de doctrine, les informations des attributs seront stockées dans la base de données.

## **c/ Classe UserRepository**

Un repository centralise tout ce qui touche à la récupération de vos entités.

Lors de la création de la classe User, un UserRepository lui est associé.

Une méthode upgradePassword permettra de modifier le mot de passe actuel.

## **d/ Configuration de l'entité User**

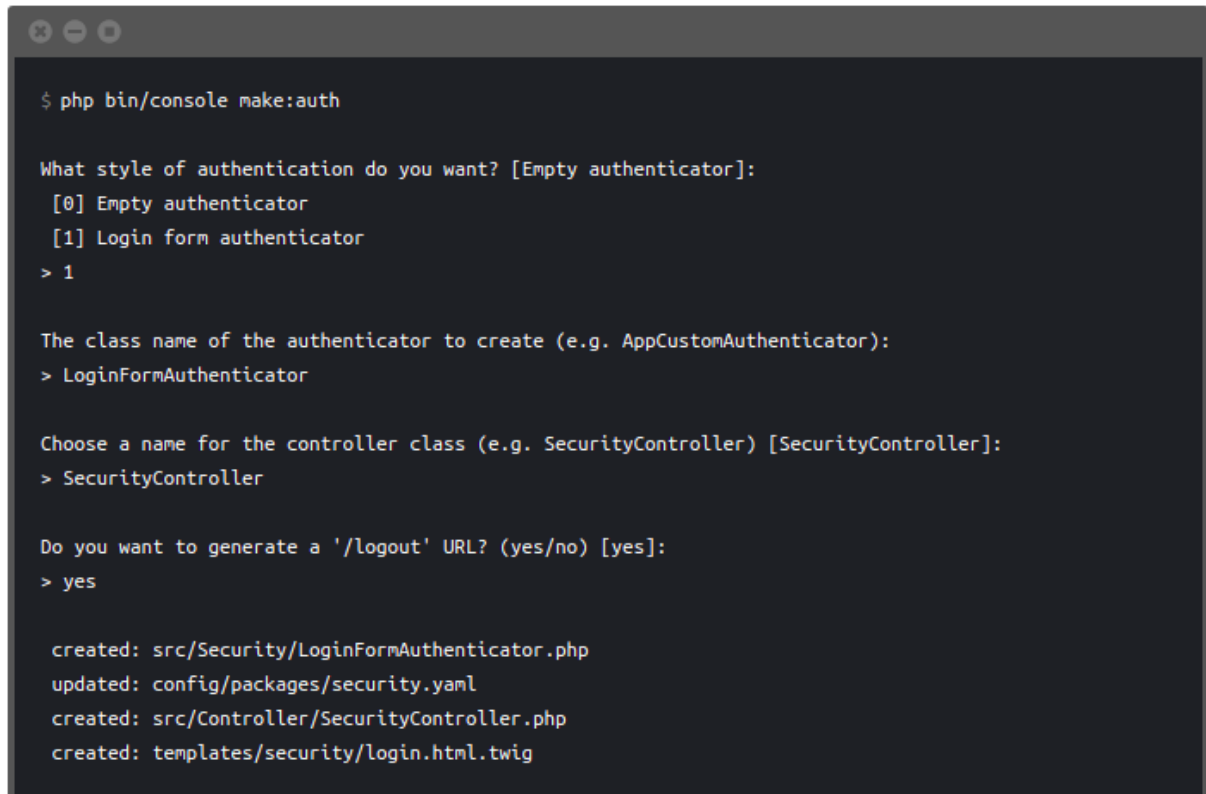
Le fichier security.yaml (config/packages/security.yaml) doit contenir les informations suivantes :

```
1 security:
2     enable_authenticator_manager: true
3
4     password_hashers:
5         Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
6         App\Entity\User:
7             algorithm: auto
8
9     providers:
10         app_user_provider:
11             entity:
12                 class: App\Entity\User
13                 property: username
```

## 2/ Authentication

Ressource : [https://symfony.com/doc/current/security/form\\_login\\_setup.html](https://symfony.com/doc/current/security/form_login_setup.html)

A l'aide de la commande `php bin/console make:auth`, les différentes méthodes vont être implémentées.



```
$ php bin/console make:auth

What style of authentication do you want? [Empty authenticator]:
[0] Empty authenticator
[1] Login form authenticator
> 1

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> LoginFormAuthenticator

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
> SecurityController

Do you want to generate a '/logout' URL? (yes/no) [yes]:
> yes

created: src/Security/LoginFormAuthenticator.php
updated: config/packages/security.yaml
created: src/Controller/SecurityController.php
created: templates/security/login.html.twig
```

### a/ Classe LoginFormAuthenticator

La classe doit étendre `AbstractLoginFormAuthenticator`.

L'utilisation de `TargetPathTrait` permet de récupérer et de définir la dernière URL visitée par l'utilisateur avant d'être obligé de s'authentifier.

L'implémentation de l'Url generator est faite dans un constructeur.

Pour qu'elle puisse fonctionner, elle fait appelle à :

- `authenticate`, retournant un objet de type `PasseportInterface`, permet d'ouvrir la session de l'utilisateur grâce à la récupération de l'username et à la vérification du password et du token,
- `onAuthenticatorSuccess` : la session est ouverte avec succès et redirige vers la page de la liste des tâches,
- `getLoginUrl` permet de générer la route pour se connecter.

## [b/ Configuration de l'authentification](#)

Le fichier security.yaml (config/packages/security.yaml) doit contenir les informations suivantes :

```
15     firewalls:
16         dev:
17             pattern: ^/(_(profiler|wdt)|css|images|js)/
18             security: false
19         main:
20             lazy: true
21             provider: app_user_provider
22             custom_authenticator: App\Security\LoginFormAuthenticator
23             logout:
24                 path: app_logout
25                 # where to redirect after logout
26                 target: app_login
27
28     # Easy way to control access for large sections of your site
29     # Note: Only the *first* access control that matches will be used
30     access_control:
31         # - { path: ^/admin, roles: ROLE_ADMIN }
32         # - { path: ^/profile, roles: ROLE_USER }
33
```

## [c/ SecurityController](#)

Elle étend l'AbstractController.

Elle a besoin de deux méthodes :

- login permettant de se connecter à la session,
- logout permettant de se déconnecter de la session.

## [d/ Template Login.html.twig](#)

Le template permet d'obtenir la vue pour se connecter à une session utilisateur.

Il est composé :

- de la base qui est étendue (à toutes les vues),
- d'un message en cas d'erreur,
- d'un bouton se déconnecter dans le cas où un utilisateur est connecté,
- des champs du titre "Se connecter", du nom de l'utilisateur, du mot de passe, de la vérification du token pour s'authentifier et d'un bouton Se connecter.

```

1  {% extends 'base.html.twig' %}
2
3  {% block title %}log'in{% endblock %}
4
5  {% block body %}
6      <form method="post">
7          {% if error %}
8              <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
9          {% endif %}
10
11          {% if app.user %}
12              <div class="mb-3">
13                  Tu es connecté en tant que {{ app.user.username }}, <a href="{{ path('app_logout') }}">Se déconnecter</a>
14              </div>
15          {% endif %}
16
17          <h1 class="h3 mb-3 font-weight-normal">Se connecter</h1>
18
19          <label for="inputUsername">Nom de l'utilisateur</label>
20          <input type="text" value="{{ last_username }}" name="username" id="inputUsername" class="form-control mb-2"
21              autocomplete="username" required autofocus>
22
23          <label for="inputPassword">Mot de passe</label>
24          <input type="password" name="password" id="inputPassword" class="form-control mb-2"
25              autocomplete="current-password" required>
26
27          <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">
28
29          <div class="d-flex justify-content-end">
30              <button class="btn btn-primary" type="submit">Se connecter</button>
31          </div>
32      </form>
33  {% endblock %}

```

### 3/ Autorisation

Ressource : <https://symfony.com/doc/current/security/voters.html>

Les voters (électeurs) sont le moyen le plus puissant de Symfony pour gérer les autorisations. Ils vous permettent de centraliser toute la logique d'autorisation, puis de les réutiliser à de nombreux endroits.

Créer un voter grâce à la commande :

- php bin/console make:voter UserVoter,
- php bin/console make:voter TaskVoter.

#### a/ TaskVoter

Au niveau des tâches, nous pouvons déterminer ce qu'il est possible de faire.

La méthode support nous permet de traiter une (ou des) demande(s) :

- est ce que mes attributs dans le tableau travaillent bien avec mes tâches ?

```
12 protected function supports(string $attribute, $task): bool
13 {
14     return in_array($attribute, ['TASK_DELETE', 'TASK_EDIT', 'TASK_TOGGLE'])
15         && $task instanceof Task;
16 }
```

Pour être autorisé à éditer ou à supprimer une tâche, nous pouvons vérifier :

- qu'une tâche créée par un utilisateur soit bien le même utilisateur de la session pour être autorisé à le faire,
- ou qu'elle soit bien rattachée à un rôle administrateur,
- et que l'utilisateur de la tâche est à null (donc anonyme).

Pour être autorisé à marquer une tâche comme faite ou non terminée, nous pouvons vérifier :

- que l'utilisateur est bien l'auteur de la tâche,
- ou que l'utilisateur a bien le rôle administrateur.

```

18 protected function voteOnAttribute(string $attribute, $task, TokenInterface $token): bool
19 {
20     $user = $token->getUser();
21     // if the user is anonymous, do not grant access
22     if (!$user instanceof UserInterface) {
23         return false;
24     }
25
26     // ... (check conditions and return true to grant permission) ...
27     switch ($attribute) {
28         case 'TASK_EDIT':
29         case 'TASK_DELETE':
30             return ($user === $task->getUser()) || (in_array(needle: 'ROLE_ADMIN', $user->getRoles())
31                 && null === $task->getUser());
32         case 'TASK_TOGGLE':
33             return $user === $task->getUser() || in_array(needle: 'ROLE_ADMIN', $user->getRoles());
34     }
35
36     return false;
37 }

```

## b/ UserVoter

Pour les utilisateurs, nous allons procéder comme pour les tâches.

Est ce que mes attributs 'USER\_LIST', 'USER\_EDIT' et 'USER-DELETE' travaillent bien avec mes utilisateurs ?

```

12 protected function supports(string $attribute, $subject): bool
13 {
14     return in_array($attribute, ['USER_LIST', 'USER_EDIT', 'USER_DELETE'])
15         && $subject instanceof User;
16 }

```

Mes attributs 'USER\_LIST', 'USER\_EDIT' et 'USER-DELETE' pourront être autorisés si l'utilisateur a bien le rôle administrateur.

```

18 protected function voteOnAttribute(string $attribute, $subject, TokenInterface $token): bool
19 {
20     $user = $token->getUser();
21     // if the user is anonymous, do not grant access
22     if (!$user instanceof UserInterface) {
23         return false;
24     }
25
26     // ... (check conditions and return true to grant permission) ...
27     switch ($attribute) {
28         case 'USER_LIST':
29         case 'USER_DELETE':
30         case 'USER_EDIT':
31             return in_array(needle: 'ROLE_ADMIN', $user->getRoles());
32     }
33
34     return false;
35 }

```



## c/ IsGranted

Les autorisations sont faites.

Il est possible de les utiliser grâce à l'annotation @IsGranted :

- dans UserController

```
80  /**
81   * @Route("/users/{id}/delete", name="user_delete")
82   *
83   * @IsGranted("USER_DELETE", subject="user", message="Tu ne peux pas supprimer des utilisateurs.")
84   */
85  public function delete(User $user, EntityManagerInterface $em)
86  {
87      $em->remove($user);
88      $em->flush();
89
90      $this->addFlash( type: 'success', message: 'L\'utilisateur a bien été supprimé. ');
91
92      return $this->redirectToRoute( route: 'user_list');
93  }
```

- dans TaskController

```
54  /**
55   * @Route("/tasks/{id}/edit", name="task_edit")
56   *
57   * @IsGranted("TASK_EDIT", subject="task", message="Tu ne peux modifier que tes propres tâches
58   * (sauf si tu es administrateur : les tâches anonymes)")
59   */
60  public function update(Task $task, Request $request, EntityManagerInterface $em)
61  {
62      $form = $this->createForm( type: TaskType::class, $task);
63
64      $form->handleRequest($request);
65
66      if ($form->isSubmitted() && $form->isValid()) {
67          $em->flush();
68
69          $this->addFlash( type: 'success', message: 'La tâche a bien été modifiée. ');
70
71          return $this->redirectToRoute( route: 'task_list');
72      }
73
74      return $this->render( view: 'task/edit.html.twig', [
75          'form' => $form->createView(),
76          'task' => $task,
77      ]);
78  }
```

Il est également possible d'utiliser `is_granted` pour lui affecter un rôle dans un template :

- `default/index.html.twig`

```
7  {% block body %}
8      <div class="row m-2">
9          <div class="d-flex justify-content-between">
10             <a href="{{ path('task_create') }}" class="btn btn-success">Créer une nouvelle tâche</a>
11
12             <a href="{{ path('task_list') }}" class="btn btn-info">Consulter la liste des tâches</a>
13
14             {% if is_granted('ROLE_ADMIN', app.user) %}
15                 <a href="{{ path('user_list') }}" class="btn btn-warning">Consulter la liste des utilisateurs</a>
16             {% endif %}
17         </div>
18     </div>
19 {% endblock %}
```