



TODO & CO
PLAN TO BE PRODUCTIVE

L'authentification et gestion des droits – Fonctionnement

Guide par Clément Thuet

Le système d'authentification

On utilise pour l'application ToDoCo l'authentification manager de Symfony

Documentation :

<https://symfony.com/doc/4.4/components/security/authentication.html>)

Les principaux fichiers impliqués

1. L'entité User : src\Entity**User.php**
2. La configuration de la sécurité : config/packages/**security.yaml**
3. Le formulaire de connexion : templates/security**login.html.twig**
4. Le SecurityController avec les fonctions login() et logout() :
src\Controller**SecurityController.php**
5. Les voters : src\Security\Voters**UserVoter.php**
6. Le AccessDeniedHandler : src\Security**AccessDeniedHandler.php**

I - L'entité User

Chaque utilisateur de l'application est représenté par l'entité User et enregistré en base de données. On y stock les informations utiles au traitement de la connexion et de la gestion des droits. On retrouve notamment le nom d'utilisateur, le mot de passe et les rôles.

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

    /**
     * @ORM\Column(type="string", length=25, unique=true)
     * @Assert\NotBlank(message="Vous devez saisir un nom d'utilisateur.")
     */
    private $username;

    /**
     * @ORM\Column(type="string", length=64)
     */
    private $password;

    /**
     * @ORM\Column(type="string", length=60, unique=true)
     * @Assert\NotBlank(message="Vous devez saisir une adresse email.")
     * @Assert\Email(message="Le format de l'adresse n'est pas correcte.")
     */
    private $email;

    /**
     * @ORM\Column(name="roles", type="json", nullable=true)
     */
    private $roles;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Task", mappedBy="user")
     */
    private $tasks;
}
```

Cette entité User doit implémenter *Symfony\Component\Security\Core\User\UserInterface* afin de s'assurer qu'elle dispose des méthodes **getRoles()**, **getPassword()**, **getSalt()**, **getUsername()**, **eraseCredentials()**.

Veillez donc à ce que ces méthodes sont bien présentes dans l'entité User.

II – Configuration de la sécurité

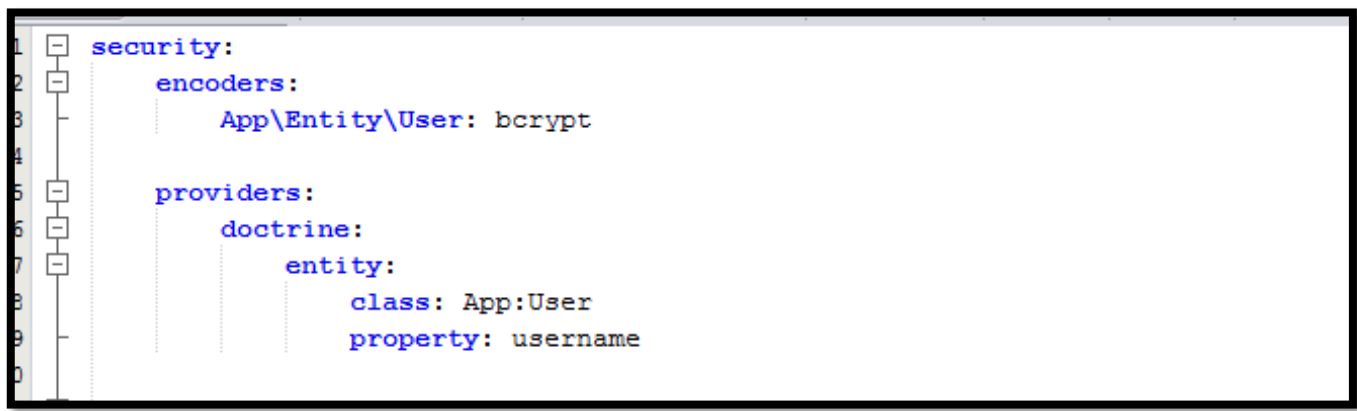
Le fichier security.yaml (config/packages) contient la configuration de la sécurité, elle se décompose en 3 parties :

1 – Le provider

Il s'agit du fournisseur des utilisateurs, on y indique comment ils vont être chargé, dans notre cas, d'une entité : **entity**

La classe d'où ils proviennent : **App: User**

Et la propriété sur laquelle va se faire l'authentification : **username**



```
1 security:
2   encoders:
3     App\Entity\User: bcrypt
4
5   providers:
6     doctrine:
7       entity:
8         class: App\User
9         property: username
10
```

2 – L'encoder

Il définit l'encodage qui sera appliqué au mot de passe dans la base de données, ici bcrypt.

3 – Le firewall

Il est possible de définir plusieurs firewalls, celui qui nous intéresse est « main ».

La clé « form_login » indique quelle route sera utilisée pour le formulaire de login (login_path) et pour la connexion (check_path).

On définit également ici « access_denied_handler » qui permet de configurer les redirections en cas d'accès refusé couplé aux voters (voir partie V et VI).

```
11  firewalls:
12      dev:
13          pattern: ^/(_(profiler|wdt)|css|images|js)/
14          security: false
15
16      main:
17          anonymous: ~
18          pattern: ^/
19          form_login:
20              login_path: login
21              check_path: login_check
22              always_use_default_target_path: true
23              default_target_path: /
24          logout: ~
25          access_denied_handler: App\Security\AccessDeniedHandler
```

4 – Le contrôleur d'accès

Il faut également savoir qu'il est possible de gérer les chemins qui seront autorisés ou non selon le rôle des utilisateurs.

Dans notre cas on a une configuration plus fine grâce aux voters et cette section n'est pas utilisée.

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

III – Formulaire de login

Il s'agit d'un formulaire tout simple avec les champs username et password renvoyant vers la route **login_check** comme définit dans security.yaml.

Attention, les champs « name » doivent avoir pour valeur **_username** et **_password** pour être reconnus par Symfony.

```
{% 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 class="btn btn-success" type="submit">Se connecter</button>
    </form>
{% endblock %}
```

IV – Security Controller

Le SecurityController comporte 3 méthodes :

- Login qui retourne le formulaire de connexion (login.html.twig).
- loginCheck doit être laissé vide et est utilisé par Symfony pour connecter l'utilisateur après soumission du formulaire.
- logoutCheck, même utilité mais pour la déconnexion.

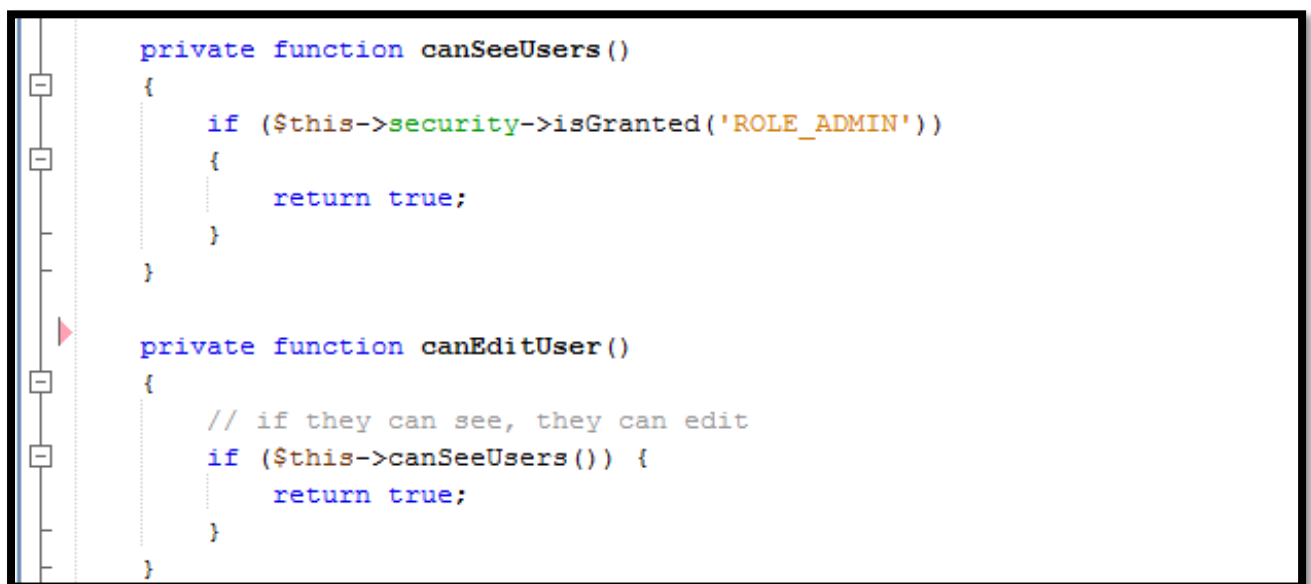
```
8
9  class SecurityController extends Controller
10 {
11     /**
12      * @Route("/login", name="login")
13      */
14     public function loginAction(Request $request)
15     {
16         $authenticationUtils = $this->get('security.authentication_utils');
17
18         $error = $authenticationUtils->getLastAuthenticationError();
19         $lastUsername = $authenticationUtils->getLastUsername();
20
21         return $this->render('security/login.html.twig', array(
22             'last_username' => $lastUsername,
23             'error'         => $error,
24         ));
25     }
26
27     /**
28      * @Route("/login_check", name="login_check")
29      */
30     public function loginCheck()
31     {
32         // This code is never executed.
33     }
34
35     /**
36      * @Route("/logout", name="logout")
37      */
38     public function logoutCheck()
39     {
40         // This code is never executed.
41     }
42 }
43
```

V – Voters

Les voters permettent une gestion plus granulaire des droits et accès.

Ils sont répartis sur deux fichiers, **UserVoter.php** et **TaskVoter.php**, chacun s'occupant d'une partie différente de l'application.

UserVoter s'occupent des droits de consultation et de modification des utilisateurs, si l'utilisateur connecté a le rôle **ROLE_ADMIN** il peut réaliser ces deux actions.



```
private function canSeeUsers()
{
    if ($this->security->isGranted('ROLE_ADMIN'))
    {
        return true;
    }
}

private function canEditUser()
{
    // if they can see, they can edit
    if ($this->canSeeUsers()) {
        return true;
    }
}
```

Je ne vais pas détailler le fonctionnement des voters ici, la documentation est consultable à cette adresse :

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

Le TaskVoter a une utilisation un peu plus fine, il permet notamment de gérer le cas où l'utilisateur peut supprimer une tâche s'il en est l'auteur OU si la tâche est rattachée à l'utilisateur anonyme et que l'utilisateur connecté a le rôle ROLE_ADMIN .

```
private function canCreate()
{
    if ($this->security->isGranted('ROLE_USER', 'ROLE_ADMIN'))
    {
        return true;
    }
}

private function canDelete(Task $task, User $user)
{
    if ($task->getUser() == $user || ($task->getUser()->getUsername() == "ANONYMOUS" && $this->security->isGranted('ROLE_ADMIN')))
    {
        return true;
    }
}

private function canEdit(Task $task, User $user)
{
    // if they can delete, they can edit
    if ($this->canDelete($task, $user)) {
        return true;
    }
}
```

VI – AccessDeniedHandler

Comme définit dans security.yaml, ce fichier est appelé lorsque l'utilisateur tente d'accéder à une page à laquelle il n'a pas accès.

Dans ce cas, selon la page à laquelle il a voulu accéder on ajoute un message d'erreur flash et on le redirige vers la page la plus adaptée.

Par exemple si via l'URL il tente de supprimer une tâche qu'il ne possède pas on l'informe qu'il n'a pas le droit de faire cela et on le redirige vers la liste des tâches.

```
if(strpos($request->getRequestUri(), "delete") == true )
{
    $request->getSession()->getFlashBag()->add('error', 'Vous n\'avez pas la permission de supprimer cette tâche.');
```

```
    return new RedirectResponse('/tasks');
```

```
}
```

```
if($request->getRequestUri() == "/users" )
{
    $request->getSession()->getFlashBag()->add('error', 'Vous n\'avez pas la permission d\'accéder à la liste des utilisateurs.');
```

```
    return new RedirectResponse('/');
```

```
}
```