**Atelier 9: Contrôle d’accès et gestion des rôles**

**Travail à faire** :

Créer un système d’utilisateurs (authentification/ inscription / gestion de rôles) avec Symfony selon les indications suivantes : (suivez les étapes dans le lien )

<https://aymeric-cucherousset.fr/symfony-6-utilisateurs/>

**Partie Théorique :**

Le concept d’autorisation permet de restreindre l’accès à une ressource.

1. **Rôles**

**Mécanique des rôles**

Lorsqu’un utilisateur se connecte au site web, Symfony appelle la méthode getRoles() sur l’entité User afin de récupérer les rôles de l’utilisateur en question. Chaque utilisateur connecté se voit par défaut toujours attribuer au moins le rôle : ROLE\_USER.

Ces rôles sont stockés en base de données sous forme de tableau, dans la colonne roles de la table user.

class User {

/\*\*

\* @ORM\Column(type="json")

\*/

private $roles = [];

// ...

public function getRoles(): array {

$roles = $this->roles;

// guarantee every user at least has ROLE\_USER

$roles[] = 'ROLE\_USER';

return array\_unique($roles);

}

}

**Hiérarchie des rôles**

Pensez à ajouter tous les rôles exemple(["ROLE\_ADMIN"]) dans la base de donnée dans la table user .

Au lieu de donner plusieurs rôles à chaque utilisateur, il est d’usage de définir une hiérarchie de rôles en définissant des règles d’héritage dans le fichier security.yaml :

security:

# ...

role\_hierarchy:

ROLE\_AUTHOR: ROLE\_USER

ROLE\_ADMIN: [ROLE\_AUTHOR, ROLE\_ALLOWED\_TO\_SWITCH]

ROLE\_SUPER\_ADMIN: [ROLE\_ADMIN, ROLE\_ALLOWED\_TO\_SWITCH]

Les utilisateurs avec le rôle **ROLE\_AUTHOR** ont aussi le rôle **ROLE\_USER.**

Les utilisateurs avec le rôle **ROLE\_ADMIN** ont aussi le rôle **ROLE\_ALLOWED\_TO\_SWITCH** et par extension les rôles **ROLE\_AUTHOR et ROLE\_USER.**

Les utilisateurs avec **ROLE\_SUPER\_ADMIN** auront automatiquement **ROLE\_ADMIN, ROLE\_ALLOWED\_TO\_SWITCH et ROLE\_USER** (hérité de ROLE\_ADMIN).

**Attention :**

Les valeurs role\_hierarchy sont statiques : **vous ne pouvez pas, par exemple, stocker la hiérarchie des rôles dans une base de données.**

Si vous en avez besoin, créez un **VOTER** de sécurité personnalisé qui recherche les rôles d'utilisateur dans la base de données.

1. **Contrôler l'accès**

Il existe deux manières de refuser l'accès à quelque chose :

1. **Sécuriser les URLs dans security.yaml :** vous permet de protéger les modèles d'URL (par exemple /admin/\*). Plus simple, mais moins flexible ;
2. **Dans le code**
3. **Sécuriser les URLs**

Une fois nos rôles définis, nous allons pouvoir définir au niveau du pare-feu :

1. Quels URL protéger
2. Quels rôles sont nécessaires pour chaque URL

Pour cela, il faut éditer le fichier de configuration security.yaml :

**security**:

access\_control:

# permettre aux utilisateurs non authentifiés d'accéder au login form

- { path: ^/admin/login, roles: PUBLIC\_ACCESS }

# Les admin peuvent accéder à /admin\*

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

# correspond à /admin/users/\*

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

# Les admin et auteurs peuvent accéder à /admin/formation\* et /admin/category\*

- { path: '^/admin/(formation|category)', roles: ROLE\_AUTHOR }

# Les utilisateurs authentifiés peuvent accéder à /formations\*

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

# Les utilisateurs authentifiés peuvent accéder aux URLs de la forme /api/formations/7298 et /api/category/528491

- { path: '^/api/(formations|category)/\d+$', roles: ROLE\_USER }

**roles** : la liste des rôles autorisés à accéder à la ressource. Il est tout à fait possible de spécifier un tableau de rôles en valeur. Exemple : roles: [ROLE\_1, ROLE\_2]

1. **Dans le code :**
2. **Sécuriser les templates**

La fonction Twig is\_granted() permet d’afficher une portion de template, de manière conditionnée par le rôle d’un utilisateur.

Exemples :

En twig, on peut ajouter une condition pour vérifier si l'utilisateur connecté est admin

{% if is\_granted('ROLE\_ADMIN') %}

Ou bien un rôle author :

{% if is\_granted('ROLE\_AUTHOR') %}

<a href="#">Editer</a>

<a href="#">Supprimer</a>

{% endif %}

Réf : <https://laconsole.dev/formations/symfony/espace-membre/#d%C3%A9connexion>

## **Symfony : Comment vérifier le rôle d’un utilisateur en respectant la hiérarchie des rôles**

comment faire pour vérifier qu’un **utilisateur** a bien le ROLE :

<https://www.remipoignon.fr/symfony-comment-verifier-le-role-dun-utilisateur-en-respectant-la-hierarchie-des-roles/>

1. **Sécuriser les Controllers :**

**Exemple**

**Limiter l’accès à l’admin** :

Ajouter le code pour que le contrôleur sera utilisé par l'utilisateur authentifié.

// src/Controller/AdminController.php

// ...

public function adminDashboard(): Response

{

$this->denyAccessUnlessGranted('ROLE\_ADMIN');

// or add an optional message - seen by developers

$this->denyAccessUnlessGranted('ROLE\_ADMIN', null, 'User tried to access a page without having ROLE\_ADMIN');

}

Ou utiliser les annotations : #[IsGranted('ROLE\_…….')]

Exemple :

// src/Controller/AdminController.php

// ...

use Symfony\Component\Security\Http\Attribute\IsGranted;

#[IsGranted('ROLE\_ADMIN')]

class AdminController extends AbstractController

{

// Optionally, you can set a custom message that will be displayed to the user

#[IsGranted('ROLE\_SUPER\_ADMIN', message: 'You are not allowed to access the admin dashboard.')]

public function adminDashboard(): Response

{

// ...

}

}

**Vérifier si un utilisateur est connecté** :

Utiliser IS\_AUTHENTICATED ou IS\_AUTHENTICATED\_FULLY

Exemple :

// ...

public function adminDashboard(): Response

{

$this->denyAccessUnlessGranted('IS\_AUTHENTICATED');

// ...

}

**Pour plus de traitement de permissions admin :**

<https://symfonycasts.com/screencast/easyadminbundle/permissions>

**Pour plus d’Authentification avec Security bundle** : (sv5)

<https://cours.davidannebicque.fr/symfony/semestre-3/securite>

**Le contrôle d’accès** :

<https://symfony.com/doc/current/security.html#security-securing-controller>

<https://nuage-pedagogique.fr/cours-chapitre08-symfony>

<https://nouvelle-techno.fr/articles/live-coding-gerer-les-roles-utilisateur-avec-symfony-4>

<https://www.linkedin.com/pulse/cr%C3%A9ez-lauthentification-des-utilisateurs-avec-symfony-cyril-vassallo/>

**Tout sur EasyAdmin :**

<https://symfonycasts.com/screencast/easyadminbundle/install>

**Token :**

<https://www-inf.telecom-sudparis.eu/COURS/CSC4101/tp-09/>

# **Symfony 6 JSON Web Token(JWT) Authentication :**

<https://www.binaryboxtuts.com/php-tutorials/symfony-6-json-web-tokenjwt-authentication/>

**S'authentifier et d'interagir avec l'*API*:**

<https://iut-info.univ-reims.fr/users/jonquet/intranet/but/r401/tp/user/>

# **Gestion avancée des permissions avec le bundle Easy Admin sous Symfony**

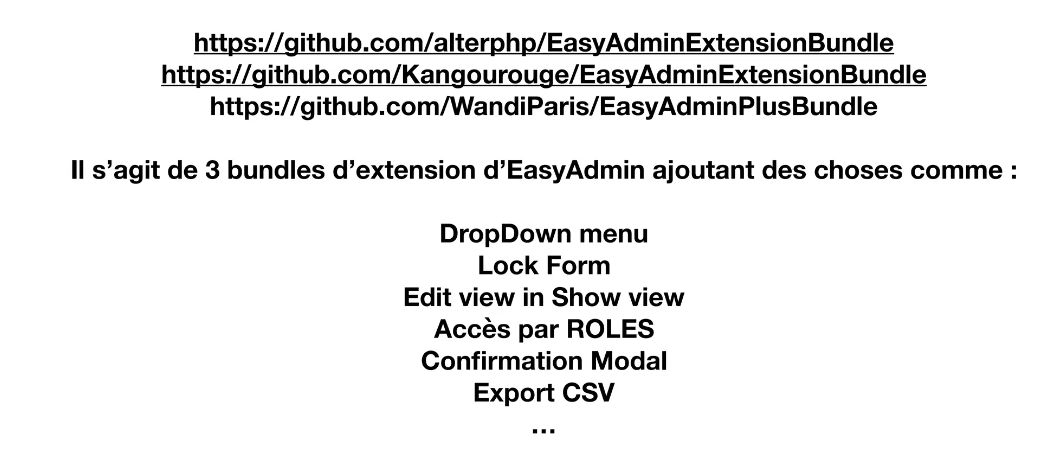
<https://afsy.fr/avent/2017/01-easyadminbundle-l-arriere-guichet-easy-peasy>

<https://fabien-lemoine.medium.com/gestion-avanc%C3%A9e-des-permissions-avec-le-bundle-easy-admin-sous-symfony-926b06c3a8a1>

**Voter :**

<https://symfonycasts.com/screencast/easyadminbundle/voter>

**Extensions easyAdmin** :



**Mise en place d'une API sécurisée servant une liste de publications (...très privées) à une application Angular / symfony:**

un système d'authentification par token en utilisant Symfony pour la partie back-end, en incluant le bundle Lexik JWT Authentication.

<https://blog.eleven-labs.com/fr/angular2-symfony3-comment-creer-rapidement-systeme-dauthentification/>

## **Sommaire**

1. Mise en place de la partie Back-end
   1. Installation d'une application Symfony
   2. Installation des bundles nécessaires
   3. Création d'utilisateurs
   4. Création d'un jeu de données
2. Mise en place de la partie Front-end
   1. Création d'une application Angular2 via Angular CLI
   2. Création des différents composants
   3. Mise en place d'un système de routing
   4. Authentifier ses requêtes grâce à Angular2-jwt
   5. Protéger les routes authentifiées avec AuthGuard