TP9 : Gérer les rôles utilisateurs avec Symfony 4

On se propose dans cet atelier de configurer et gérer les rôles des utilisateurs.

On aura les rôles suivants :

- *ROLE_USER* : rôle par défaut de tous les utilisateurs
- *ROLE_EDITOR* : l'utilisateur possédant ce rôle pourra ajouter et modifier les articles
- *ROLE_ADMIN* : c'est le rôle de l'administrateur de l'application. L'utilisateur possédant ce rôle a le droit de tout faire y compris d'attribuer des rôles aux utilisateurs inscrits.

NB : L'utilisateur possédant le rôle *ROLE_ADMIN* bénéficiera automatiquement du rôle *ROLE_EDITOR*

Objectifs :

- Ajouter l'attribut *roles* à l'entité User,
- La déclaration des rôles dans le fichier security.yaml,
- Création d'une interface pour gérer les utilisateurs et leurs rôles,
- Restreindre l'accès à certaines fonctionnalités avec l'annotation **@isGranted,**
- Cacher des liens aux utilisateurs ne possédant pas un certain rôle.

Ajouter l'attribut *roles* à l'entité User

1. Ajouter à l'entité ***User*** , l'attribut *roles* et la fonction setRoles et modifier la fonction getRoles comme suit :

```
    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

   public function getRoles(): array
    { $roles = $this->roles;

    // garantit que chaque utilisateur possède le rôle ROLE_USER
    // équivalent à array_push() qui ajoute un élément au tabeau
          $roles[] = 'ROLE_USER';

    //array_unique élémine des doublons
        return array_unique($roles);
    }

    public function setRoles(array $roles): self
    {
```

```
        $this->roles = $roles;

        return $this;
    }
```

2. Créer une migration et l'exécuter sur la BD :

   php bin/console make:migration

   php bin/console doctrine:migrations:migrate

La déclaration des rôles

3. Pour déclarer les rôles, modifier le fichier config/packages/security.yaml comme suit :

```
access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
       # - { path: ^/profile, roles: ROLE_USER }
    role_hierarchy:
        ROLE_EDITOR: ROLE_USER
        ROLE_ADMIN:  ROLE_EDITOR
```

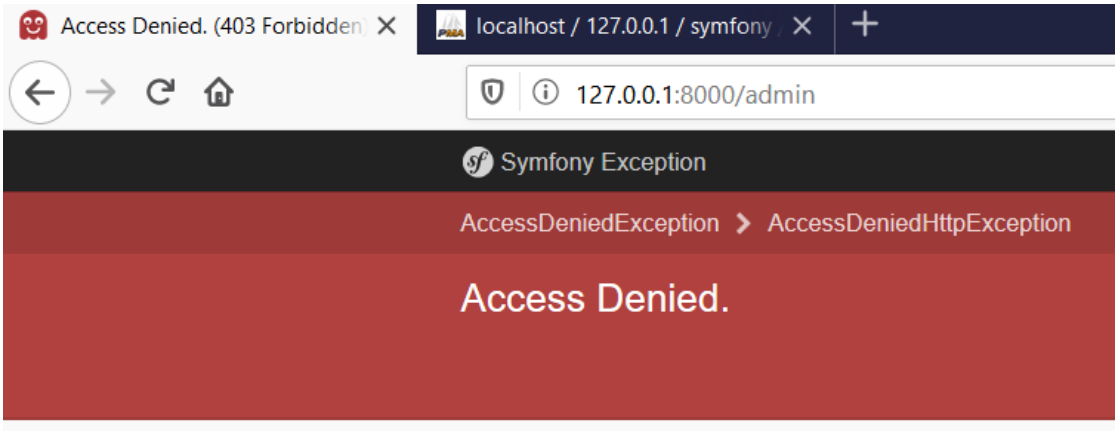
4. Ajouter directement dans la BD le rôle ROLE_ADMIN à un utilisateur :

	id	email	username	password	roles
Éditer Copier Supprimer	3	MALAK@yahoo.com	malak	$2y$13$xu.brfnp6wDRznCjxJ6yDuw7fNTQTZClWC99SCV/MOs...	
Éditer Copier Supprimer	4	nadhemb1@yahoo.com	nadhem	$2y$13$yeF48tQiX/thRrWfYosR.O0/B9O5VtHRFSqMo7i5rQD...	["ROLE_ADMIN"]
Éditer Copier Supprimer	5	louka@gmail.com	louka	$2y$13$NiAi48c3AWK0d9MKmPmr5O6hRxBlRVW44Oqyh/gys4j...	[]

5. Créer le contrôleur AdminController :

   php bin/console make:controller

6. Tester l'accès à la router 127.0.0.1:8000/admin en étant connecté avec un utilisateur qui ne possède pas le rôle *ROLE_ADMIN*, puis avec un utilisateur possédant ce rôle



Création d'une interface admin pour afficher la liste des utilisateurs

7. Attribuer la route /admin à la classe AdminController, avec le name (admin_). Ajouter la fonction usersList au contrôleur AdminController comme suit :

```php
<?php

namespace App\Controller;

use App\Repository\UserRepository;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;


/**
 * @Route("/admin", name="admin_")
*/
class AdminController extends AbstractController
{
     /**
     * @Route("/utilisateurs", name="utilisateurs")
     */
    public function usersList(UserRepository $user) {
        return $this->render("admin/users.html.twig",[
            'users' => $user->findAll()
        ]);

    }
}
```

8. Créer la vue admin/users.html.twig pour afficher la liste des utilisateurs :

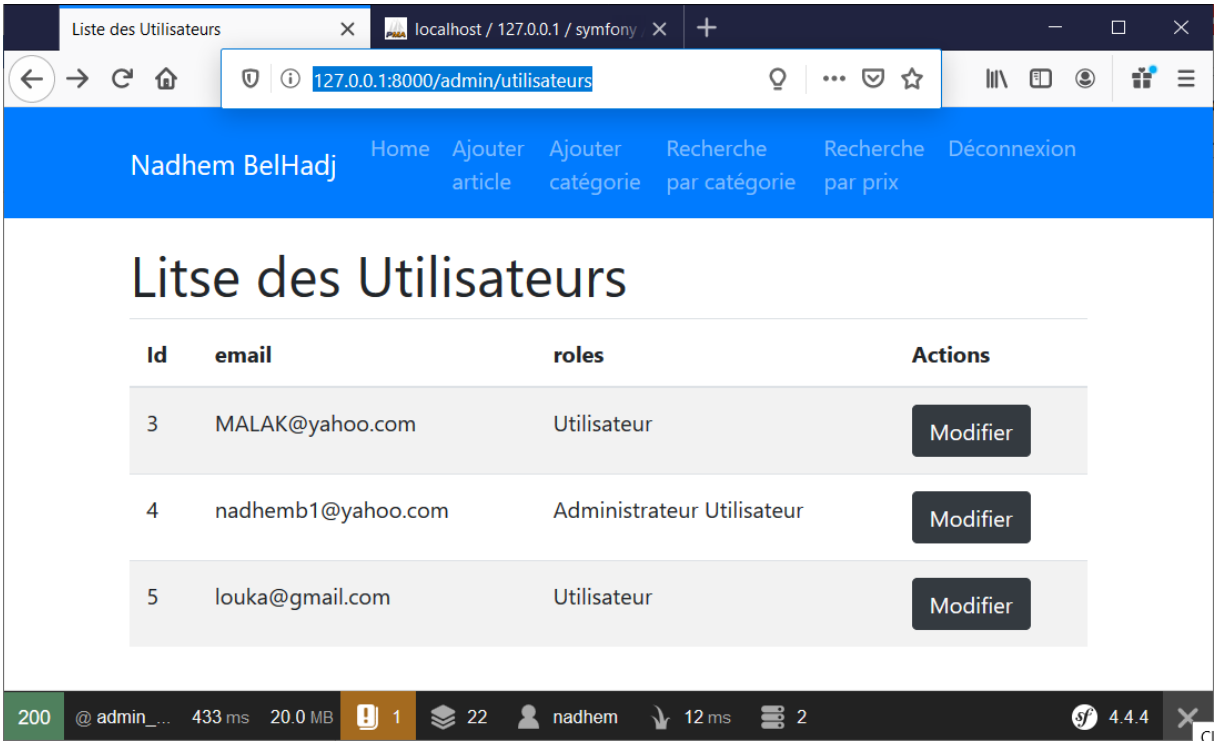
```twig
{% extends 'base.html.twig' %}
{% block title%} Liste des Utilisateurs{% endblock %}

{% block body %}
<h1>Litse des Utilisateurs</h1>
 <table id="users" class="table table-striped">
      <thead>
        <tr>
          <th>Id</th>
          <th>email</th>
          <th>roles</th>
          <th>Actions</th>
```

```
        </tr>
      </thead>
      <tbody>
        {% for user in users %}
          <tr>
            <td>{{ user.id }}</td>
            <td>{{ user.email }}</td>
             <td>
                {% for role in user.roles %}
                 {% if role=="ROLE_USER" %}
                      Utilisateur
                 {% elseif role=="ROLE_EDITOR" %}
                      Editeur
                {% elseif role=="ROLE_ADMIN" %}
                      Administrateur
                 {% endif %}
                {% endfor %}
              </td>
           <td>
 <a href="{{ path('admin_modifier_utilisateur', {'id': user.id }) }}" class="btn btn-
dark">Modifier</a>
             </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>

{% endblock %}
```

9. Connectez-vous avec un utilisateur possédant le role ROLE_ADMIN et tester http://127.0.0.1:8000/admin/utilisateurs



Création d'une interface d'attribution des rôles aux utilisateurs

10. Créer le formulaire EditUserType, basé sur l'entité user en utilisant la commande :

    php bin/console make:form EditUserType

11. Modifier la fonction ***buildForm*** comme suit :

```
public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('email')
            ->add('roles',ChoiceType::class,[
                'choices' => [
                   'Utilisateur' => 'ROLE_USER',
                   'Editeur' => 'ROLE_EDITOR',
                   'Administrateur' => 'ROLE_ADMIN'
                ],
                'expanded' => true,
                'multiple' => true,
                'label' => 'Rôles'
            ]);
    }
```

12. Ajouter la fonction editUser au contrôleur AdminController comme suit :

```
/**
     * @Route("/utilisateurs/modifier/{id}", name="modifier_utilisateur")
     */
    public function editUser(Request $request, User $user, EntityManagerInterface
  $em) {

        $form = $this->createForm(EditUserType::class,$user);

        $form->handleRequest($request);
        if($form->isSubmitted() && $form->isValid()) {
          $em->flush();

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

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

13. Créer la vue admin/editUser.html.twig :

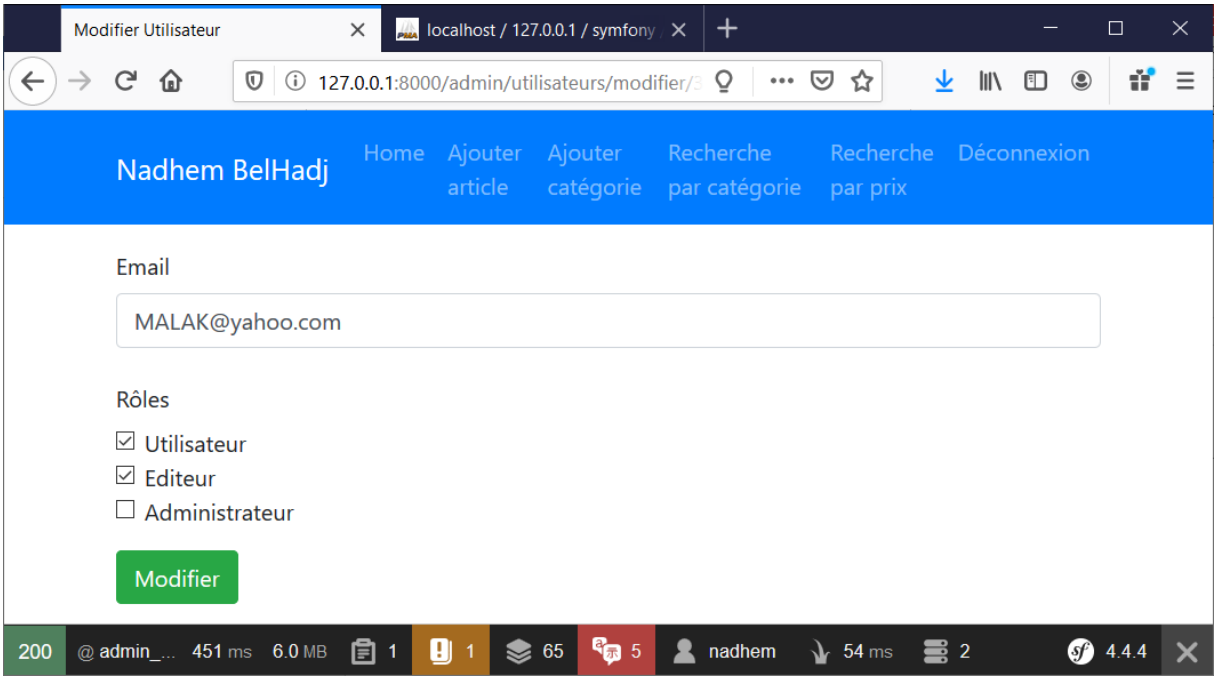
```
{% extends 'base.html.twig' %}

{% block title %}Modifier Utilisateur{% endblock %}
{% block body %}
 {{ form_start(formUser) }}
 {{ form_widget(formUser) }}
   <button type="submit" class="btn btn-success">Modifier</button>
 {{ form_end(formUser) }}
{% endblock %}
```

14. Au niveau de l'entité User enlever la contrainte EqualTo sur l'attribut password :

```
*  @Assert\EqualTo(propertyPath = "confirm_password",
     *  message="Vous n'avez pas saisi le même mot de passe !" )
```

15. Tester la modification d'un utilisateur en lui attribuant des rôles :



Restreindre l'accès à certaines fonctionnalités avec l'annotation @isGranted

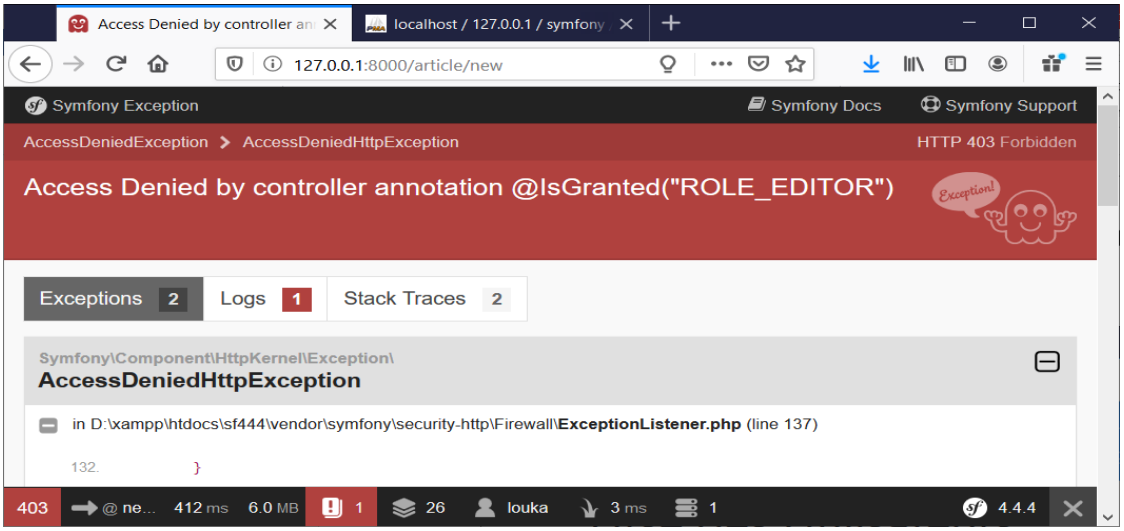
On se propose à présent d'interdire la création des articles aux utilisateurs qui ne possèdent ni le rôle ROLE_EDITOR ni le rôle ROLE_ADMIN.

16. Modifier l'annotation de la fonction new du contrôleur IndexController comme suit :

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
```

```
/**
     * @IsGranted("ROLE_EDITOR")
     * @Route("/article/new", name="new_article")
     * Method({"GET", "POST"})
     */
    public function new(Request $request) {
…
```

17. Tester la création d'un article en se connectant avec un utilisateur ne possédant pas le rôle ROLE_EDITOR :



18. Toujours dans le contrôleur *IndexController*, faire de même pour la modification et la suppression des articles :

```
/**
     * @IsGranted("ROLE_EDITOR")
     * @Route("/article/edit/{id}", name="edit_article")
     * Method({"GET", "POST"})
     */
    public function edit(Request $request, $id) {
…
```

```
/**
     * @IsGranted("ROLE_EDITOR")
     * @Route("/article/delete/{id}",name="delete_article")
     * @Method({"DELETE"})
     */
    public function delete(Request $request, $id) {
```

19. Tester la modification puis la suppression d'un article en se connectant avec un utilisateur ne possédant pas le rôle ROLE_EDITOR :

Ajouter dans la navbar le lien « Administration » qui sera visible que pour les utilisateurs possédant le rôle ROLE_ADMIN

20. Modifier le fichier inc/navbar.html.twig comme suit :

```
 {% if is_granted('ROLE_ADMIN') %}
      <li class="nav-item">
        <a href="{{ path('admin_utilisateurs')}}" class="nav-
link">Administration</a>
      </li>
      {% endif %}
```

Cacher les liens « Ajouter article » et « Ajouter catégorie » aux utilisateurs ne possédant pas le rôle ROLE_EDITOR

21. Modifier le fichier inc/navbar.html.twig comme suit :

```
 {% if  is_granted('ROLE_EDITOR') %}
      <li class="nav-item">
        <a href="{{ path('new_article')}}" class="nav-link">Ajouter article</a>
      </li>
      <li class="nav-item">
        <a href="{{ path('new_category')}}" class="nav-link">Ajouter catégorie</a>
      </li>
    {% endif %}
```

Cacher les boutons « Modifier » et « Supprimer » aux utilisateurs ne possédant pas le rôle ROLE_EDITOR

22. Modifier le fichier articles/index.html.twig comme suit :

```
<td>
    <a href="/article/{{ article.id }}" class="btn btn-dark">Détails</a>
  {% if  is_granted('ROLE_EDITOR') %}
   <a href="/article/edit/{{ article.id }}" class="btn btndark">Modifier</a>
   <a href="/article/delete/{{ article.id }}" class="btn btn-danger"
   onclick="return confirm('Etes-
vous sûr de supprimer cet article?');">Supprimer</a>
   {% endif %}
 </td>
```