TP8 Authentification en Symfony 4

On se propose dans cet atelier de créer un formulaire d'inscription et une interface de connexion en utilisant les composants *security* de Symfony 4.

Objectifs :

- Création de l'entité *User*,
- Création d'un formulaire d'inscription (Registration),
- Ajouter des contraintes de validation pour l'entité *User,*
- Crypter les mots de passe dans la base de données,
- Création d'un formulaire de login,
- Création d'un lien pour la Déconnexion (logout),
- Cacher des menus pour les utilisateurs non connectés

Création de l'entité User et sa table

1. Créer l'entité **User** (email, username, password)

   php bin/console make:entity User

   php bin/console make:migration

   php bin/console doctrine:migrations:migrate

Création du formulaire d'inscription

2. Créer un formulaire :

   php bin/console make:form RegistrationType

3. Ajouter la propriété confirm_password à l'entité User :

```
private $confirm_password;

  public function getConfirmPassword()

   {
       return $this->confirm_password;
   }

   public function setConfirmPassword($confirm_password)
   {
       $this->confirm_password = $confirm_password;

       return $this;
   }
```

4. Ajouter la propriété confirm_password au formulaire dans le fichier *RegistrationType :*

```
use Symfony\Component\Form\Extension\Core\Type\PasswordType;
```

```
 ->add('email')
            ->add('username')
            ->add('password',PasswordType::class)
            ->add('confirm_password',PasswordType::class)
```

5. Créer le contrôleur ***SercurityController :***

   php bin/console make:controller

```
<?php
namespace App\Controller;
use App\Entity\User;
use App\Form\RegistrationType;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Doctrine\ORM\EntityManagerInterface;


class SecurityController extends AbstractController
{
    /**
     * @Route("/inscription", name="security_registration")
     */
    public function registration(Request $request, EntityManagerInterface  $em )
    {
        $user = new User();

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

        $form->handleRequest($request);

        if($form->isSubmitted() && $form->isValid()) {
          //l'objet $em sera affecté automatiquement grâce à l'injection des dépen
dances de symfony 4
           $em->persist($user);
```

```
            $em->flush();
        }
        return $this->render('security/registration.html.twig',
                            ['form' =>$form->createView()]);
    }
}
```

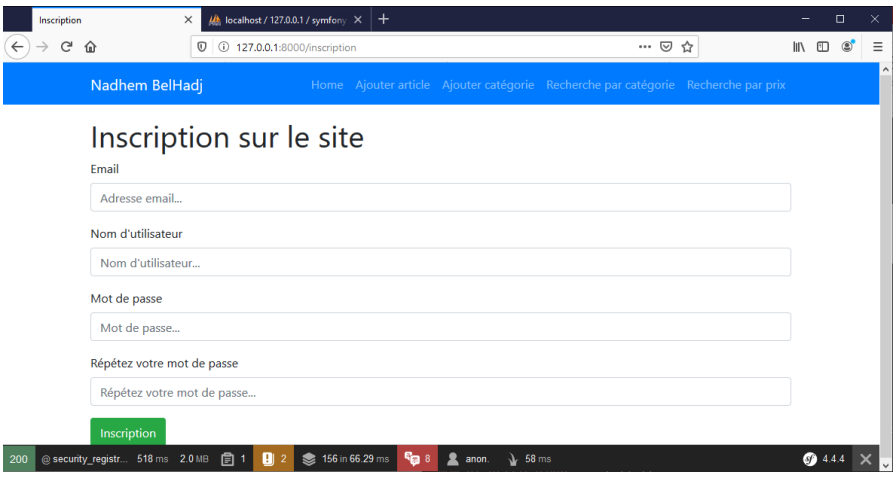
6. Créer la vue security/registration.html.twig :

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

{% block title %}Inscription{% endblock %}

{% block body %}
<h1>Inscription sur le site</h1>
{{ form_start(form) }}
 {{ form_row(form.email, { 'attr': {  'placeholder': 'Adresse email...' } }  ) }}
 {{ form_row(form.username, { 'label':'Nom d\'utilisateur', 'attr': {'placeholder'
:'Nom d\'utilisateur...' } }) }}
 {{ form_row(form.password, { 'label':'Mot de passe', 'attr': {'placeholder':'Mot
de passe...' } }) }}
 {{ form_row(form.confirm_password, { 'label':'Répétez votre mot de passe',
                                      'attr':{'placeholder':'Répétez votre mot de
passe...' } }) }}
  <button type="submit" class="btn btn-success">Inscription</button>
 {{ form_end(form) }}
{% endblock %}
```

7. Testez votre travail :



Ajouter des contraintes de validation

Pour plus d'informations sur les contraintes de validations vous pouvez consulter : https://symfony.com/doc/current/validation.html

8. Créer deux contraintes d'intégrité :
   - La longueur minimale du mot de passe est 8 caractères
   - Le mot de passe doit être confirmé

Modifier l'entité User comme suit :

```
use Symfony\Component\Validator\Constraints as Assert;
```

```
/**
     * @ORM\Column(type="string", length=255)
     *  @Assert\Length(
     *      min = 8,
     *      minMessage = "Votre mot de passe doit comporter au minimum {{ limit }}
 caractères")
     *  @Assert\EqualTo(propertyPath = "confirm_password",
     *  message="Vous n'avez pas saisi  le même mot de passe !" )
     */
    private $password;

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

Crypter les mots de passe dans la base de données

9. Modifier le fichier config/packages/security.yaml :

```
security:
    encoders:
        App\Entity\User:
            algorithm: bcrypt
```

10. Modifier la fonction *registration* du contrôleur SecurityController comme suit :

```
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;
```

```
public function registration(Request $request,
EntityManagerInterface  $em,  UserPasswordEncoderInterface $encoder)
    {
```

```
…
if($form->isSubmitted() && $form->isValid()) {
      $hash = $encoder->encodePassword($user,$user->getPassword());
            $user->setPassword($hash);

            $em->persist($user);
            $em->flush();
        }
```

11. L'entité User doit implémenter l'interface *UserInterface* pour qu'elle soit gérée par le composant Symfony security. Modifier l'entité User pour qu'elle implémente l'interface UserInterface et définit toutes ses méthodes :

```
use Symfony\Component\Security\Core\User\UserInterface;
```

```
class User implements UserInterface
{
….
```

```
public function getRoles()
    {
        return ['ROLE_USER'];
    }

    public function eraseCredentials() {}

    public function getSalt() {}
```

12. Tester votre travail en ajoutant un utilisateur puis consulter la BD

L'email saisi doit être unique

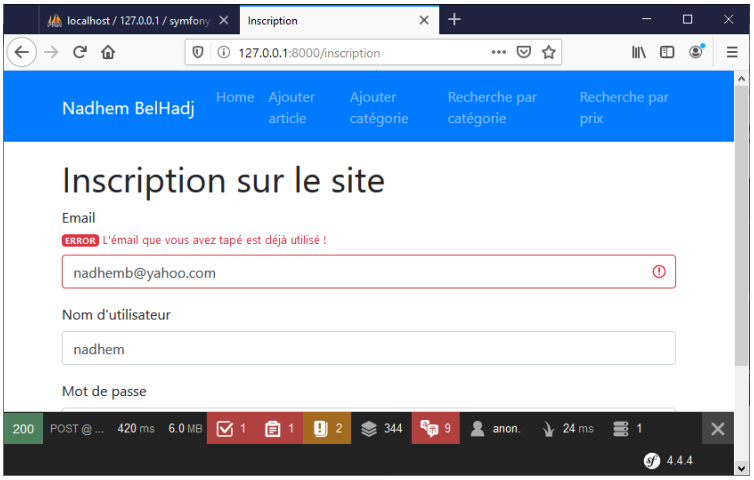
13. Modifier l'entité User pour ajouter la contrainte UniqueEntity

```
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
```

```
/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 * @UniqueEntity(
 *     fields={"email"},
```

```
 *      message="L'émail que vous avez tapé est déjà utilisé !"
 * )
 */
class User implements UserInterface
{….
```

14. Testez votre travail en créant deux utilisateurs :



Création d'un formulaire de login

15. Ajouter au fichier SecurityController la fonction login()

```
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;
```

```
/**
 * @Route("/connexion",name="security_login")
 */
    public function login(AuthenticationUtils  $authenticationUtils)
    {
         // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();

    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

      return $this->render('security/login.html.twig',
                ['lastUsername'=>$lastUsername,'error' => $error]);
    }
```

16. Créer la vue security/login.html.twig :

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

```
{% block body %}
<h1>Connexion !</h1>

{% if error %}
    <div>{{ error.messageKey|trans(error.messageData, 'security') }}</div>
{% endif %}

<form action="{{ path('security_login') }}" method="post">
    <div class="form-group">
        <input placeholder="Adresse email..." requied name="_username" value ="
{{lastUsername}}"
         type="text" class="form-control">
    </div>
    <div class="form-group">
        <input placeholder="Mode de passe..." requied name="_password"
         type="password" class="form-control">
    </div>
     <div class="form-group">
       <button type="submit" class="btn btn-success">Connexion</button>
    </div>
</form>
{% endblock %}
```

17. Modifier le fichier config/packages/security.yaml

```
providers:
       in_memory: { memory: null }
       in_database:
           entity:
               class: App\Entity\User
               property: email
```

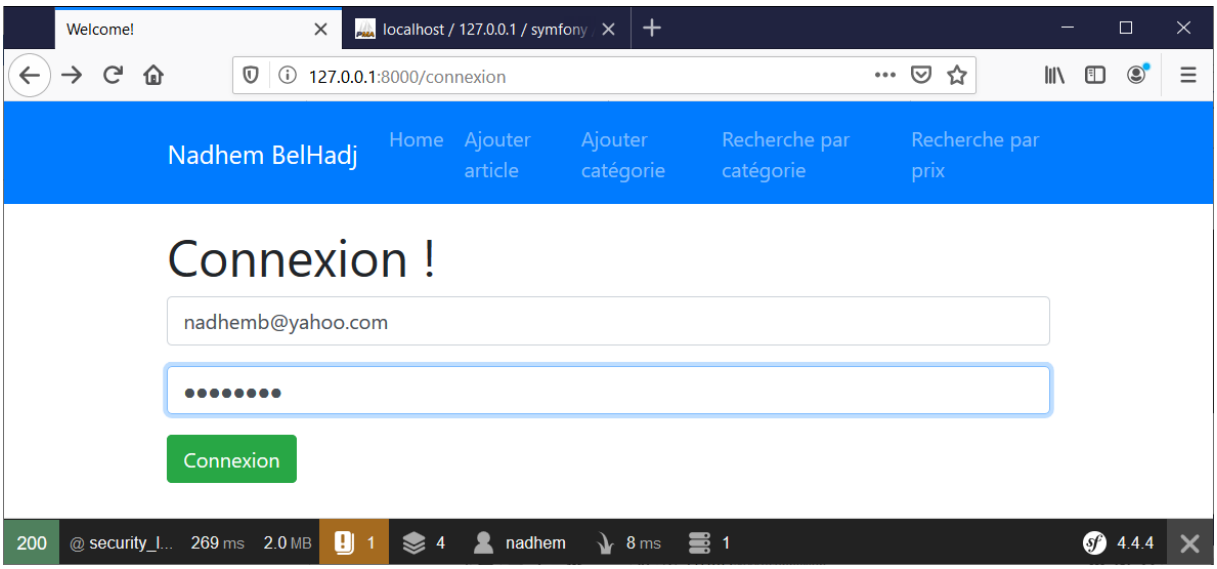
```
firewalls:
       dev:
           pattern: ^/(_(profiler|wdt)|css|images|js)/
           security: false
       main:
           anonymous: lazy
           provider: in_database
           form_login:
```

```
            login_path: security_login
            check_path: security_login
```

18. Modifier la fonction ***registration***() pour diriger l'utilisateur inscrit vers la page de login

```
if($form->isSubmitted() && $form->isValid()) {
…
            $em->flush();
            return $this->rederictToRoute('security_login');
        }
```

19. Tester le login :



Déconnexion

20. Ajouter la fonction logout() au contrôleur SecurityController :

```
/**
 * @Route("/deconnexion",name="security_logout")
 */
public function logout()
{ }
```

21. Modifier le fichier config/packages/security.yaml en ajoutant sous la rubrique main :

```
 logout:
     path: security_logout
     target:article_list
```

22. Ajouter au fichier inc/navbar.html.twig, les deux liens suivants :

```
    <li class="nav-item">
      <a href="{{ path('security_login')}}" class="nav-link">Connexion</a>
    </li>
    <li class="nav-item">
      <a href="{{ path('security_logout')}}" class="nav-link">Déconnexion</a>
    </li>
```

23. On se propose à présent d'afficher dans la navbar soit Connexion soit Déconnexion selon que l'utilisateur est connecté ou pas. Modifier le fichier inc/navbar.html.twig comme suit :

```
{% if not app.user %}
    <li class="nav-item">
      <a href="{{ path('security_login')}}" class="nav-link">Connexion</a>
    </li>
{% else %}
    <li class="nav-item">
      <a href="{{ path('security_logout')}}" class="nav-link">Déconnexion</a>
    </li>
{% endif %}
```

24. Tester

Cacher les menus « Ajouter Article » et « Ajouter Catégorie » pour les utilisateurs non connectés

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

```
{% if  app.user %}
     <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 %}
```