TP6 : Authentification / Login

L'objectif principal est de créer un système d'authentification, via formulaire, pour permettre notamment de gérer l'accès ou non à certaines pages.
Ainsi, nous devons pouvoir nous connecter en tant qu'utilisateur « simple », ou en tant qu'administrateur pour pouvoir gérer certaines fonctionnalités ; nous devons de ce fait pouvoir créer un compte et le modifier si besoin.

1. Installation du bundle Sécurité

```
composer require symfony/security-bundle
```

2. Création de l'entité sécurisée « User »

- Documentation liée : « https://symfony.com/doc/current/security.html »

On repart sur nos classiques via le Maker :

```
php bin/console make:user
```

Nous voulons une classe « User », stockée en BDD, avec des mots de passe sécurisés.

=> Concernant le « send an email to verify the user » : c'est très bien, mais dans un environnement local avec des adresses en gmail, vous n'avez aucune sécurité sur votre environnement, et gmail va vous hurler dessus que NON vous ne passerez pas.
Pour ceux que ça intéresse, vous pouvez l'activer : ça nécessite l'installation de dépendances supplémentaires, de mettre à jour la BDD et la configuration dans le .env, et de prendre le risque d'avoir des erreurs à l'affichage, mais qui ne bloqueront pas la création de

compte, puisque le mail est envoyé après la création du compte en BDD.

```
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

 Will this app need to hash/check user passwords? Choose No if passwords are not needed or
will be checked/hashed by some other system (e.g. a single sign-on server).

 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


 Success!


 Next Steps:
   - Review your new App\Entity\User class.
   - Use make:entity to add more fields to your User entity and then run make:migration.
   - Create a way to authenticate! See https://symfony.com/doc/current/security.html
```

- Notre entité User implémente une interface : allez la voir.

  -> Qu'est-ce signifie le « unique=true » dans l'annotation liée à l'ORM sur l'attribut « email »?

- Ajouter les champs « FirstName » et « LastName » dans l'entité User, pensez aux getters / setters. (Et au form, et au twig :)).
  Puis mettez à jour votre BDD.

=> Comment apparait le password en BDD ? Pourquoi ? Dans quel fichier l'encodage est-il réalisé ?

3. Création d'un formulaire de création de compte

php bin/console make:registration-form

Mettez à jour la base de données via un doctrine:schema:update par la suite.

```
Creating a registration form for App\Entity\User

 Do you want to add a @UniqueEntity validation annotation on your User class to make
sure duplicate accounts aren't created? (yes/no) [yes]:
 > yes

 Do you want to send an email to verify the user's email address after registration? (yes/no)
[yes]:
 > no

 Do you want to automatically authenticate the user after registration? (yes/no) [yes]:
 > yes

 ! [NOTE] No Guard authenticators found - so your user won't be
 !        automatically authenticated after registering.

 What route should the user be redirected to after registration?:
  [0 ] _preview_error
  [1 ] _wdt
  [2 ] _profiler_home
  [3 ] _profiler_search
  [4 ] _profiler_search_bar
  [5 ] _profiler_phpinfo
  [6 ] _profiler_xdebug
  [7 ] _profiler_search_results
  [8 ] _profiler_open_file
  [9 ] _profiler
  [10] _profiler_router
  [11] _profiler_exception
  [12] _profiler_exception_css
  [13] app_auth
 > 13

 updated: src/Entity/User.php
 created: src/Form/RegistrationFormType.php
 created: src/Controller/RegistrationController.php
```

```
created: templates/registration/register.html.twig


 Success!


Next:
Make any changes you need to the form, controller & template.

Then open your browser, go to "/register" and enjoy your new form!
```

Tester sur l'url "/register" de créer un compte, et aller voir en base de données ce que cela donne !

4. Authentification qui va nous permettre de nous connecter sur le site

Nous allons créer le formulaire de login pour gérer notre authentification toujours via le maker :

```
php bin/console make:controller Auth
```

Choisissez de nommer la route de login et de logout comme vous le souhaitez : "/login", "/sign-in"...

Nous allons ensuite modifier la vue src/templates/auth/index.html.twig (vous pouvez d'ailleurs la renommer sign-in.html.twig ou de toute autre manière si vous le souhaitez) :

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


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

```
    <form action="{{ path('app_sign_in') }}" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="_username" value="{{ last_username }}"/>

        <label for="password">Password:</label>
        <input type="password" id="password" name="_password"/>

        {# If you want to control the URL the user is redirected to on success
    <input type="hidden" name="_target_path" value="/account"/> #}

        <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">

        <button type="submit">login</button>
    </form>
{% endblock %}
```

Il est nécéssaire alors de configurer “config/packages/security.yaml pour activer les routes pour le login et le logout :

```
security:
    # ...
    firewalls:
        # ...
        main:
            # ...
            form_login:
                login_path: app_sign_in
                check_path: app_sign_in
                enable_csrf: true
            logout:
                path: app_sign_out
```

En dernier lieu, éditer le “src/AuthController.php” pour définir les routes sign-in / sign-out (ou login/logout selon la manière dont vous les avez nommées) :

```
+ use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;
  # ...
  class AuthController extends AbstractController
  {
    #[Route('/sign-in', name: 'app_sign_in')]
    public function app_sign_in(AuthenticationUtils $authenticationUtils): Response
    {
        // get the login error if there is one
        $error = $authenticationUtils->getLastAuthenticationError();

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

        return $this->render('auth/sign-in.html.twig', [
            'last_username' => $lastUsername,
            'error'         => $error,
        ]);
    }

    #[Route('/sign-out', name: 'app_sign_out')]
    public function app_sign_out(): Response
    {
        // controller can be blank: it will never be called!
        throw new \Exception('Don\'t forget to activate logout in security.yaml');
    }
  }
```

Allez tester votre login sur l'url choisie "http://xxx/sign-in", entrer votre email / password et … Magie !

5. Restreindre les accès selon les utilisateurs connectés

- Nous voulons restreindre les accès aux méthodes « new », « edit », « delete » des concerts aux seuls administrateurs. Si jamais nous ne sommes pas connecter avec une utilisateur admin, il y aura automatiquement un « accès denied » ou une redirection vers le formulaire de login si je ne suis pas connecté.

Plusieurs possibilités pour réaliser cela :

- Décommentez l'access_control côté security.yml, et ajouter les '/admin' dans les routes concernées par le contrôle administrateur (ou un IS_GRANTED).
- Pensez à insérer le rôle admin via les fixtures pour pouvoir l'utiliser simplement derrière !

Ce lien permettra à Symfony de gérer la destruction de la session et des cookies associés.

6. Créer un nouvel onglet « Login » dans la navBar

Créer un nouvel onglet « Créer un compte » : je vous laisse le choix du graphisme, on peut imaginer un bouton dépliant « Login » qui contient l'option « Créer un compte » dans la navbar par exemple.
A vous de jouer !

7. Créer une page de modification de son compte

Let's go ! Vous avez toutes les compétences nécessaires pour le réaliser.

- Une page qui affiche les informations liées à son compte, accessible depuis la navbar.
- Un bouton qui permet de les modifier

=> Grâce à ces commandes, Symfony nous a permis d'initialiser un système d'authentification et de l'utiliser presque clé en main, à nous d'analyser ce dernier pour voir ce qu'il y a dedans.

Attention : Le fait d'utiliser des outils qui nous facilitent la vie, n'exclut pas de comprendre ces derniers !