SYMFONY – COMPOSANT SECURITY

https://symfony.com/doc/current/components/security.html
https://symfony.com/doc/current/security.html#installation

I- Installation.

composer require symfony/security-bundle

II- Créer une classe User.

https://symfony.com/doc/current/security.html#the-user

php bin/console make:user va permettre de :

- créer la classe User (Entité)
- indiquer comment persister les users (via doctrine par exemple)

Puis migration si doctrine :

php bin/console make:migration
php bin/console doctrine:migrations:migrate

Dans le fichier config/packages/security.yaml, il est paramétré l'utilisation du provider 'entité User' avec le email comme identifiant d'utilisateur.

Il y est également indiqué comment hasher les mots de passe des utilisateurs

III- L'enregistrement (registering) des users et/ou mot de passe hashé.

https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords

L'enregistrement des users (User Provider) peut être fait dans la bdd (Entity User Provider), LDAP (LDAP User Provider), fichier de configuration…

Utiliser le registration controller (Choisir Entity User Provider):

« The `make:registration-form` maker command can help you set-up the registration controller and add features like email address verification using the SymfonyCastsVerifyEmailBundle.

composer require symfonycasts/verify-email-bundle
php bin/console make:registration-form »

A la fin de l'exécution de l'instruction précédente, est affiché :
« Next:
1) In RegistrationController::verifyUserEmail():
* Customize the last redirectToRoute() after a successful email verification.
* Make sure you're rendering success flash messages or change the $this->addFlash() line.
2) Review and customize the form, controller, and templates as needed.
3) Run "php bin/console make:migration" to generate a migration for the newly added User::isVerified property.

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

Puis après avoir démarré le serveur, localhost:8000/register (Si erreur sur le mailer, décommenter la ligne avec MAILER_DSN dans le fichier .env)

Si besoin de Hasher un mot de passe manuellement:

(https://symfony.com/doc/current/security.html#the-user)
php bin/console security:hash-password

Puis tester, dans le navigateur, localhost:8000/register : Affichage du formulaire d'inscription puis validation. => l'utilisateur est créé dans la bdd

IV- Permettre aux utilisateurs de s'authentifier.

https://symfony.com/doc/current/security.html#authenticating-users

Différentes possibilités – ici par formulaire

- Création d'un controleur login php bin/console make:controller Login
  Ce qui crée une route /login

- Modifier le fichier de config partie firewall afin d'indiquer la route pour le formulaire d'auth :
  ```
  main:
      lazy: true
      provider: app_user_provider
      form_login:
          login_path: login
          check_path: login
  ```

- Modifier l'action associée à la route login (comme indiqué sur la doc Symfony)

- Modifier la vue twig associée (comme indiqué sur la doc Symfony)

- Après login ok, renvoyer vers la route ‘principal’ :
  avec default_target_path dans security.yaml:

```
form_login:
        login_path: login
        check_path: login
        default_target_path: /principal
```

  OU
  En décommentant le champ caché du formulaire correspondant au target_path :

```
<input type="hidden" name="_target_path" value="/principal"/>
```

  Revenir à la page sur laquelle on était après connexion :

```
<input type="hidden" name="_target_path" value="{{ app.request.headers.get('referer') }}"/>
```

  https://symfony.com/doc/current/reference/configuration/security.html#reference-security-firewall-form-login

- Prévoir le logout (dans config et la route associée)
  - Dans security.yaml :

```
logout:
    path: app_logout
    target: /principal                  # Route vers laquelle sera redirigé après logout
```

  - Créer la route associée :

```
/**
 * @Route("/logout", name="app_logout", methods={"GET"})
 */
public function logout(): void
{
    // controller can be blank: it will never be called!
    throw new \Exception('Don\'t forget to activate logout in security.yaml');
}
```

- Connaitre l’user connecté :
  Dans le twig :

```
{% if is_granted('IS_AUTHENTICATED_FULLY') %}
        Connecté en tant que {{ app.user.email }}
        <a href="{{ path('app_logout') }}"> logout </a>
{% else %}
        <a href="{{ path('login') }}"> Login </a>
{% endif %}
```

  Dans le contrôleur :

```
$user = $this→getUser();      // est un objet User
```

Rediriger vers une page selon son rôle :https://gkueny.fr/symfony-tips-rediriger-apres-le-login-selon-le-role-de-lutilisateur (Vérifier la version de symfony car fosuserbundle et date de 2016)

V- Access Control .

Par défaut, chaque utilisateur a comme rôle : ROLE_USER
Les noms des rôles doivent commencer par ROLE_

Notion de hiérarchie de rôles https://symfony.com/doc/current/security.html#hierarchical-roles
Indication de comment savoir si l'user appartient à un rôle

Dans config/packages/security.yaml

```
access_control:
    - { path: ^/accessible, roles: ROLE_ADMIN }
    - { path: ^/adherents, roles: ROLE_USER }
```

Une personne non connectée ne peut pas accéder à adherents ni à accessible

Une personne connectée ayant le rôle ROLE_USER peut accéder à adherents

Une personne connectée ayant le rôle ROLE_ ADMIN peut accéder à accessible

Affecter des rôles lors d'enregistrement des users :

Modifier le formulaire type de registration, en ajoutant les rôles possibles (Dans l'exemple ci-dessous, 4 rôles possibles) :

```
->add('roles',ChoiceType::class,[
        'choices'=>array(
                'admin'=>'ROLE_ADMIN','user'=>'ROLE_USER',
                'patient'=>'ROLE_PATIENT','medecin'=>'ROLE_MEDECIN'),
                'expanded'=>true,
                'multiple'=>true
                ]
)
```

Attention, par défaut le ROLE_USER est donné à tout utilisateur (cf getRoles de la classe User)

Tester et constater les valeurs dans la bdd

Dans l'appli des adhérents :

créer des rôles admin – adherent

Non connecté : page accueil sans aucune info – page des compets

connecté en adherent : ses infos – ses compets

connecté en admin : tout le reste

https://www.doctrine-project.org/projects/doctrine-orm/en/2.8/reference/inheritance-mapping.html#single-table-inheritance

- Créer les Entités Patient et Medecin sans aucun attribut
- Modifier les classes Patient et Medecin pour préciser qu'elles héritent de User (extends User)
- Ajouter les annotations suivantes à la classe User
  * @ORM\InheritanceType("SINGLE_TABLE")
  * @ORM\DiscriminatorColumn(name="discr", type="string")
  * @ORM\DiscriminatorMap({"patient" = "Patient", "medecin" = "Medecin"})
- Migration dans la bdd
- Retester la connexion – logiquement c'est ok
- Un médecin a des rdv, un patient a des rdv => création d'une entité Rdv avec un commentaire et les relations qui vont bien
- Migrer dans la bdd
- Lors de l'authentification, Symfony va instancier un objet Patient ou un objet Medecin selon la valeur du champ discr
- L'objet User correspondant à l'utilisateur connecté est alors soit un objet Patient soit un objet Medecin et a donc accès à tous ses attributs.