



TODO LIST APP

DOCUMENTATION

AUTHENTIFICATION
& AUTORISATION

2021

PRÉPARÉ PAR
GEOFFROY DUTOT

SOMMAIRE

- 1. INTRODUCTION
- 2. AUTHENTIFICATION
 - 1. USER
 - 2. SÉCURITÉ
 - 3. FORMULAIRES ET VUES
- 3. AUTHORISATION
 - 1. ROLES

INTRODUCTION

TODO LIST APP est un site web permettant de gérer ses tâches quotidiennes. Afin d'améliorer la sécurité du site, d'éviter que n'importe qui puisse ajouter, supprimer des informations, il a été décidé d'ajouter une authentification à cette application Symfony. L'authentification a été mise en place sous Symfony 4.4 (LTS). Précédemment le site web était sous Symfony 3.1 mais il a été décidé de faire la migration vers la LTS version de Symfony de manière à avoir une version plus récente, éviter des problèmes de sécurité, dépréciations etc...

L'authentification a été mise en place à l'aide du bundle *symfony/security-bundle*.

AUTHENTIFICATION USER

Tout d'abord il nous faut créer une entité qui possèdera les attributs nécessaires à l'implémentation de l'authentification. Nous allons définir les informations nécessaires à l'authentification et elles seront enregistrer en base de donnée. L'entité "**User**" possède des attributs **username** et un **password** qui seront utilisés pour connecter l'utilisateur. L'entité possède un tableau de **roles**, qui seront utiles afin de définir les différents droits et accès que l'utilisateur aura sur le site.

```
/**
 * @ORM\Table("user")
 * @ORM\Entity
 * @UniqueEntity("email")
 */
class User implements UserInterface
{
    /**
     * @ORM\Column(type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=25, unique=true)
     * @Assert\Type("string")
     * @Assert\Length(min = 1, max = 25, minMessage = "Le nom d'utilisateur d
     * @Assert\NotBlank(message="Vous devez saisir un nom d'utilisateur.")
     */
    private $username;

    /**
     * @ORM\Column(type="string")
     * @Assert\Type("string")
     * @Assert\NotBlank(message="Vous devez saisir un mot de passe.")
     */
    private $password;

    /**
     * @ORM\Column(type="string", length=60, unique=true)
     * @Assert\NotBlank(message="Vous devez saisir une adresse email.")
     * @Assert\Email(message="Le format de l'adresse n'est pas correcte.")
     */
    private $email;

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

    /**
```

AUTHENTIFICATION USER

Les données des utilisateurs sont enregistrées en base de donnée dans la table User. Les mots de passe sont enregistrés cryptés.

✓ Affichage des lignes 0 - 3 (total de 4, traitement en 0.0070 seconde(s).)

SELECT * FROM `user`

☐ Profilage [Éditer en ligne] [Éditer]

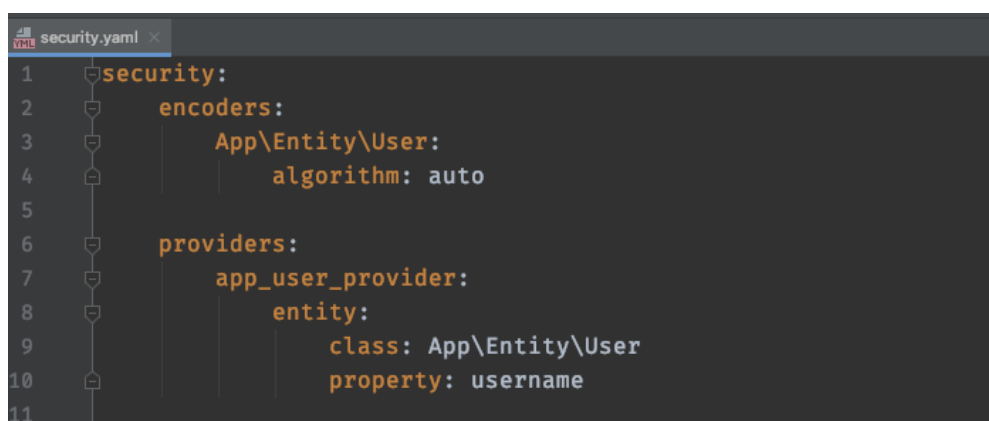
☐ Tout afficher | Nombre de lignes : 25 ▼ Filtre les lignes: Chercher dans cette table Trier par clé : Aucun(e) ▼

Options

			id	username	password	email	roles (DC2Type:json)	
<input type="checkbox"/>	✎ Éditer	📄 Copier	🗑 Supprimer	147	Geoffroy	\$argon2id\$v=19\$m=65536,t=4,p=1\$WE+iPgrMZxFrNU1ls8S...	geoffroy.dutot@gmail.com	["ROLE_ADMIN"]
<input type="checkbox"/>	✎ Éditer	📄 Copier	🗑 Supprimer	148	user	\$argon2id\$v=19\$m=65536,t=4,p=1\$PbDCfitcF2QdZHv7h+u...	user-contact@gmail.com	[]
<input type="checkbox"/>	✎ Éditer	📄 Copier	🗑 Supprimer	149	JamesT	\$argon2id\$v=19\$m=65536,t=4,p=1\$tC8D/f6/YkoBWWOYx5a...	james.t@gmail.com	[]
<input type="checkbox"/>	✎ Éditer	📄 Copier	🗑 Supprimer	150	James44	\$argon2id\$v=19\$m=65536,t=4,p=1\$KD4LF9AvOJFT9oSu+OB...	james44-contact@gmail.com	[]

AUTHENTIFICATION SÉCURITÉ

La configuration de la sécurité du projet se trouve dans "*config/packages/security.yaml*". C'est dans ce fichier que l'on retrouve des informations sur la configuration de l'authentification et des autorisations utilisateurs.



```
1 security:
2   encoders:
3     App\Entity\User:
4       algorithm: auto
5
6   providers:
7     app_user_provider:
8       entity:
9         class: App\Entity\User
10        property: username
11
```

Dans la partie "*encoders*" on retrouve l'entité qui nous permettra d'authentifier nos utilisateurs, on ajoute donc le namespace de notre entité **User** et la clé "*algorithm*" c'est ce qui nous permettra de crypter les mots de passe, elle est définie sur "*auto*" mais il est tout à fait possible de définir un algorithme en particulier comme "*bcrypt*", "*argon2i*"...

Dans "*providers*" à été ajouté un provider qui utilise notre classe **User** et dont nous avons défini la clé "*property*" sur "*username*", cela définit avec quel identifiant notre utilisateur va devoir fournir en plus du mot de passe pour pouvoir se connecter. Nous aurions pu aussi définir cette clé sur l'email par exemple.

AUTHENTIFICATION SÉCURITÉ

La partie "firewalls" définit la manière dont l'authentification va se faire. La clé "dev" permet juste de ne pas avoir d'erreurs en étant en environnement de développement. La clé "main" va définir l'accès à notre application, toutes les requêtes entrantes vont passer par ce pare-feu. On a défini la clé "anonymous" sur "true" nous permettant d'accéder au site sans devoir être authentifié sinon nous ne pourrions pas accéder à la page de login.. Pour ce qui concerne restreindre les accès des utilisateurs à certaines pages, actions, cela se trouve dans la partie "access control" (cf: partie Autorisation).

```
firewalls:
  dev:
    pattern: ^/(_(profiler|wdt)|css|images|js)/
    security: false
  main:
    anonymous: true
    guard:
      authenticators:
        - App\Security\LoginFormAuthenticator
    logout:
      path: app_logout
      target: app_login
```

Nous définissons dans "guard" et "authenticators" la classe auto-générée par le bundle sécurité de Symfony s'occupant de gérer toute la partie "login" en récupérant les informations fournies par le formulaire, vérifiant que les informations sont correctes effectuer une redirection en cas de succès ou d'erreur. Enfin dans "logout" nous indiquons les informations pour la déconnexion de l'utilisateur connecté, cela est gérée automatiquement par Symfony mais nous indiquons dans "path" le nom de la route de notre méthode logout dans le SecurityController et "target" pour définir la page où l'utilisateur sera redirigé après sa déconnexion ici nous mettons la route du login que l'on retrouve aussi dans le SecurityController.

AUTHENTIFICATION FORMULAIRES ET VUES

Nous gérons connexion et inscription des utilisateurs sur le site. Comme nous l'avons précédemment vu dans le "SecurityController" il y a la méthode login qui permet d'afficher la vue "templates/security/login.html.twig" qui contient un formulaire html dont le LoginFormAuthenticator va venir récupérer les informations pour valider ou non la connexion. Si les informations sont correctes l'utilisateur sera connecté redirigé vers l'accueil sinon la page de connexion affichera une erreur.

```
/**
 * @Route("/login", name="app_login")
 */
public function login(AuthenticationUtils $authenticationUtils): Response
{
    if ($this->getUser()) {
        return $this->redirectToRoute( route: 'home');
    }

    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();
    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

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

```
<form method="post">
    <label for="inputUsername">Nom d'utilisateur :</label>
    <input type="text" value="{{ last_username }}" name="username" id="inputUsername" autocomplete="username">

    <label for="inputPassword">Mot de passe :</label>
    <input type="password" name="password" id="inputPassword" autocomplete="current-password" required>

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

    <button class="btn btn-success" type="submit">Se connecter</button>
</form>
```


AUTHENTIFICATION FORMULAIRES ET VUES

En revanche l'inscription elle, est gérée dans le "UserController" avec la méthode "register". On peut voir dans celle-ci l'affichage de la vue "templates/user/create.html.twig"

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

% block header_title %<h1>Créer un utilisateur</h1>{% endblock %}
% block header_img %}{% endblock %}

% block body %
    {{ form_start(form, {'action' : path('user_create')}) }}
        {{ form_widget(form) }}

        <button type="submit" class="btn btn-success pull-right">Ajouter</button>
    {{ form_end(form) }}
% endblock %
```

On voit aussi la création du formulaire d'inscription via la classe "UserType". Dans cette classe nous voyons toutes les propriétés de l'utilisateur.

```
class UserType extends AbstractType

public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        →add( child: 'username', type: TextType::class, ['label' ⇒ "Nom d'utilisateur"])
        →add( child: 'password', type: RepeatedType::class, [
            'type' ⇒ PasswordType::class,
            'invalid_message' ⇒ 'Les deux mots de passe doivent correspondre.',
            'required' ⇒ true,
            'first_options' ⇒ ['label' ⇒ 'Mot de passe'],
            'second_options' ⇒ ['label' ⇒ 'Tapez le mot de passe à nouveau'],
        ])
        →add( child: 'email', type: EmailType::class, ['label' ⇒ 'Adresse email'])
        →add( child: 'Roles', type: ChoiceType::class, [
            'required' ⇒ true,
            'multiple' ⇒ false,
            'expanded' ⇒ false,
            'label' ⇒ 'Rôle',
            'choices' ⇒ [
                'Utilisateur' ⇒ 'ROLE_USER',
                'Administrateur' ⇒ 'ROLE_ADMIN'
            ],
        ])
    ];
}
```

AUTHENTIFICATION FORMULAIRES ET VUES

On peut aussi voir dans notre méthode "register" la vérification et validation du formulaire entraînant l'ajout de l'utilisateur en base de donnée et on remarque aussi que le mot de passe est "hasher".

```
/**
 * @Route("/users/create", name="user_create")
 */
public function register(Request $request, UserPasswordEncoderInterface $passwordEncoder): Response
{
    $user = new User();
    $form = $this->createForm( type: UserType::class, $user);

    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        // encode the plain password
        $user->setPassword(
            $passwordEncoder->encodePassword(
                $user,
                $form->get('password')->getData()
            )
        );

        $em = $this->getDoctrine()->getManager();
        $em->persist($user);
        $em->flush();

        $this->addFlash( type: 'success', message: "L'utilisateur a bien été ajouté.");

        return $this->redirectToRoute( route: 'user_list');
    }

    return $this->render( view: 'user/create.html.twig', ['form' => $form->createView()]);
}
```

AUTORISATION ROLES

Comme nous l'avons vu dans la partie authentification les utilisateurs possèdent des "rôles". Nous avons deux types de rôles :

- **ROLE_USER** : Tous les utilisateurs inscrits possèdent ce rôle. Dans notre site il permet de naviguer sur l'accueil les listes de tâches ainsi que la création et modification de celles-ci.
- **ROLE_ADMIN** : Ce rôle à plus de droits, moins d'utilisateurs possèdent ce rôle en plus des mêmes droits que le **ROLE_USER** ce rôle peut consulter la liste des utilisateurs, peut modifier et supprimer des utilisateurs. Il peut aussi supprimer les tâches qui n'ont pas d'auteur contrairement au **ROLE_USER**.

```
access_control:
  - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/users, roles: ROLE_ADMIN }
  - { path: ^/, roles: ROLE_USER }
```

Pour faire en sorte que certains rôles aient des accès particulier nous avons modifier la partie "access_control" dans "config/packages/security.yaml". Ci dessus nous pouvons voir que en mettant un "path" et des "roles" cela restreint l'accès à ces pages pour certains utilisateurs.

- **IS_AUTHENTICATED_ANONYMOUSLY** : permettra à tout visiteur non connecté de pouvoir accéder sur la page de connexion.
- **ROLE_ADMIN** : peut accéder aux pages de gestion des utilisateurs
- **ROLE_USER** : la page d'accueil et les pages des tâches

Il est aussi possible via les Controller d'empêcher certaines actions pour certains types d'utilisateurs. Exemple pour empêcher que les utilisateurs non admin puissent supprimer des tâches sans auteur nous avons utilisé la fonction "denyAccessUnlessGranted" permettant de limiter l'accès à certains rôles.

```
public function deleteTask(Task $task)
{
    if ($task->getAuthor() !== $this->getUser()) {
        $this->denyAccessUnlessGranted('ROLE_ADMIN');
    }
}
```

AUTORISATION ROLES

Il est aussi possible d'utiliser dans les Controllers des annotations pour empêcher l'accès à certains utilisateurs comme le fait la fonction "denyAccessUnlessGranted" via : "@IsGranted()"

En plus de limiter l'accès à certaines pages nous pouvons via les vues ne pas afficher certains éléments pour certains rôles, de manière à éviter qu'un utilisateur pense pouvoir effectuer une action tandis qu'il n'y a pas accès. Cela est possible grâce à Twig via "is_granted()".

```
{% if task.author is same as(app.user) or is_granted('ROLE_ADMIN') %}  
<form action="{{ path('task_delete', {'id' : task.id }) }}">  
    <button class="btn btn-danger btn-sm pull-right">Supprimer</button>  
</form>  
{% endif %}
```