Implémentation Authentification

Ce document qui a pour but d'accompagner les collaborateurs du projet Todo List va détailler les points suivants :

- comprendre quel(s) fichier(s) il faut modifier et pourquoi
- comment s'opère l'authentification
- Où sont stockés les utilisateurs.

Architecture et fichiers d'authentification

Dans une architecture comme **Symfony** qui est construite en PHP sur le modèle MVC il est important de comprendre comment cela fonctionne.

Les fichiers en lien avec l'authentification sont :

- /config/Package/Security.yml
- /Controller/SecurityController.php

Dans l'exemple suivant, la clé "**Entity"** qui se trouve dans "**provider"** définit quelle entité allons nous utiliser pour s'authentifier ainsi que le champs username pour se connecter, cela pourrait très bien être l'email.

Dans "**main**" nous avons la clé "**form_login**" cela définit les options de la connexion par exemple nous avons le chemin de la page login dans "**login_path**"
La clé "**default_target_path**" définit la route à suivre une fois que l'utilisateur est authentifié.

Nous avons également la clé "**logout**" qui sert à déconnecter l'utilisateur, cela est entièrement géré par Symfony. Cela intercepte la route "/logout" qui déconnecte automatiquement l'utilisateur à son appel.

Les premières lignes que nous observons toujours sur le document ci-dessous sont utilisées pour décrire l'algorithme de hashage que Symfony utilise pour hasher ses mots de passe et dans quelle entité nous allons faire cela. Pour l'instant nous pouvons voir que l'application utilise l'algorithme de hashage Bcrypt, mais nous pouvons mettre "auto" ou un autre algorithme si besoin.

Nous avons la clé "**role_hierarchy**" qui indique les droits des utilisateurs en fonction de leur rôle, à savoir qu'un administrateur aura les mêmes droits qu'un utilisateur mais également ses propres droits. Le rôle supérieur récupère les droits du rôle inférieur.

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

    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: username

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        main:
            http_basic: ~
            anonymous: lazy
            pattern: ^/
            form_login:
                login_path: login
                check_path: login_check
                always_use_default_target_path:  true
                default_target_path:  /
            logout: ~

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

    role_hierarchy:
        ROLE_ADMIN: [ROLE_ADMIN,ROLE_USER]
```

la clé “**access_control**” permet de déterminer quelles routes sont accessibles selon le rôle du membre, nous pouvons observer ici que la route “/users” pour la création de membre est réservée aux administrateurs.

Le fichier “Security.yml” bien que modifié dans cet exemple est issu de la recette de Symfony/flex, lorsque nous installons “symfony/security-bundle” avec composer, Symfony va chercher la configuration et génère un security.yml depuis le Symfony/recipes sur github : https://github.com/symfony/recipes/blob/master/symfony/security-bundle/4.4/config/packages/security.yaml

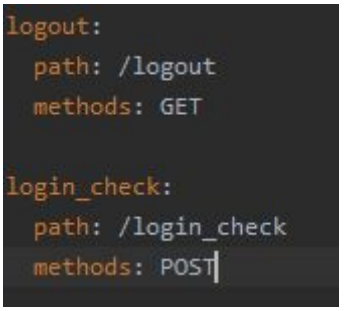
Nous avons ensuite le **SecurityController** :

```php
<?php

namespace App\Controller;


use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class SecurityController extends AbstractController
{
    /**
     * @Route("/login", name="login")
     */
    public function loginAction(Request $request, AuthenticationUtils $authenticationUtils)
    {
        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();

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

    /**
     * @Route("/login_check", name="login_check")
     */
    public function loginCheck()
    {
        // This code is never executed.
    }

    /**
     * @Route("/logout", name="logout")
     */
    public function logoutCheck()
    {
        // This code is never executed.
    }
}
```

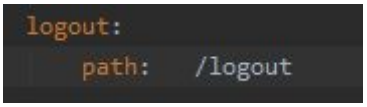
Cette classe à plusieurs méthodes donc plusieurs routes, la méthode “**loginAction**” est appelée sur la route “**/login**“

Celle-ci se contente simplement de renvoyer une vue et d’utiliser la classe “**AuthenticationUtils**” qui est gérée par Symfony.
Cela permet de renvoyer une erreur dans la vue ou alors le “l**ast_username**”

Les méthodes “**loginCheck**” et “**logoutCheck**” proviennent de l’ancienne architecture sur Symfony 3.1, sur l’architecture de Symfony 5.0 nos routes sont référencées dans le “**routes.yml**” comme ceci :

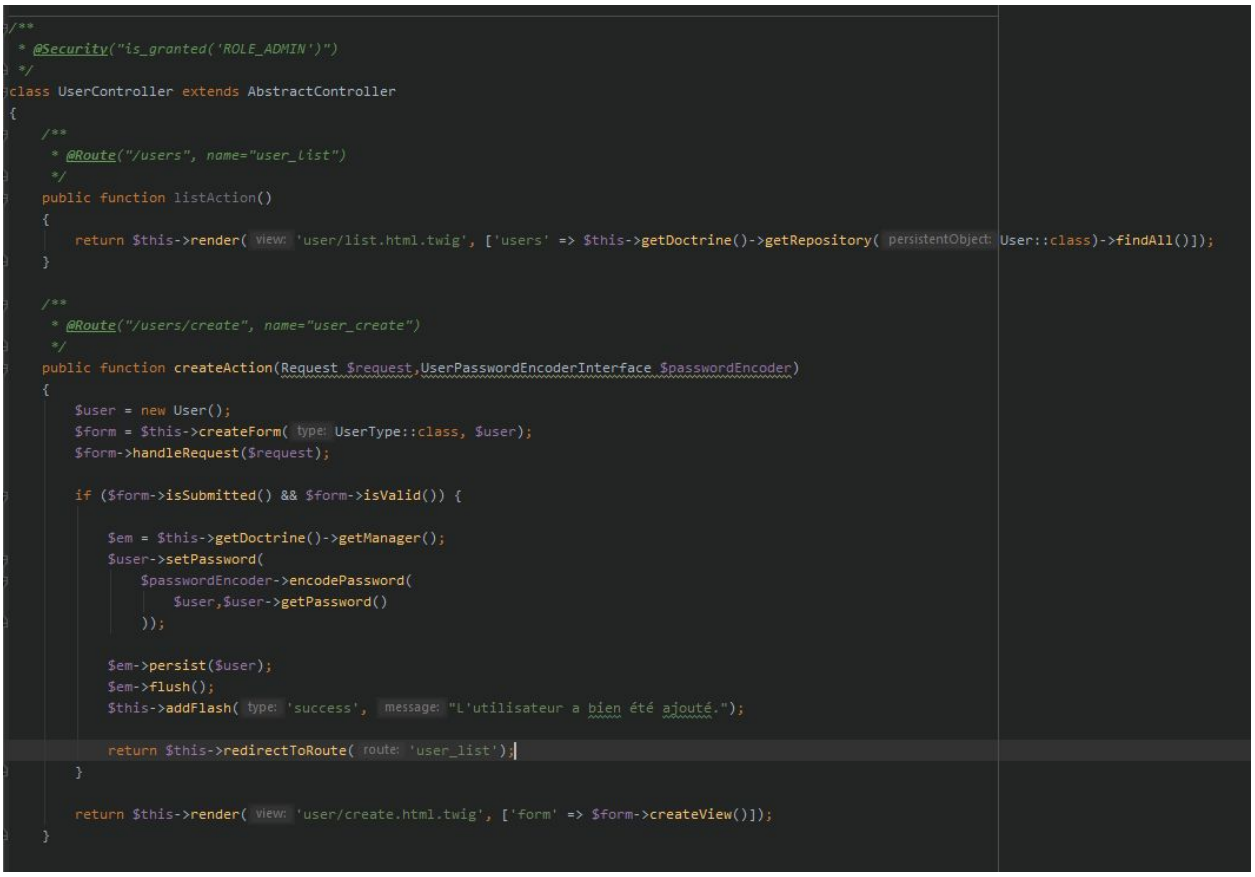


Pour que cela puisse fonctionner j’ai également effectué une modification au niveau du “**security.yml**” :



Les utilisateurs disponibles pour se connecter sont les utilisateurs enregistrés en base de données, seul un membre qui est administrateur peut créer un autre membre. La fonction de création de membre est interdite aux utilisateurs simples.

Pour se connecter il faut simplement se rendre à la racine du projet, n’étant pas connecté de base, l’utilisateurs sera redirigé vers la route “**/login**”.
Une fois connecté l’utilisateur sera redirigé à la racine du projet pour avoir la liste des tâches.



```php
/**
 * @Security("is_granted('ROLE_ADMIN')")
 */
class UserController extends AbstractController
{
    /**
     * @Route("/users", name="user_list")
     */
    public function listAction()
    {
        return $this->render( view: 'user/list.html.twig', ['users' => $this->getDoctrine()->getRepository( persistentObject: User::class)->findAll()]);
    }

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

        if ($form->isSubmitted() && $form->isValid()) {

            $em = $this->getDoctrine()->getManager();
            $user->setPassword(
                $passwordEncoder->encodePassword(
                    $user,$user->getPassword()
                ));

            $em->persist($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()]);
    }
```

Dans cette image nous observons bien que cette classe est réservée seulement aux administrateurs, donc un utilisateur simple n’aura pas accès à ses fonctionnalités et donc à la création de membre mais également l’édition de membres.