TODO LIST

Documentation de l'authentification

Table des matières

Configuration de la sécurité

Le fichier de security.yaml se trouver dans le chemin suivant project_8_todolist/config/packages/security.yaml.

Le fichier sert de configuration pour l'encodage du mot de passe, le provider, le login, access control et la hiérarchie des rôles.

Password hashers :

```yaml
password_hashers:
  Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

Il est conseillé de laisser le password hashers en auto afin de choisir le meilleur algorithme selon l'environnement.

Provider :

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

Définit « app_user_provider » pour récupérer les utilisateurs depuis l'entité « User » en se basant sur le champ « username ».

Firewalls :

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

  main:
    lazy: true
    provider: app_user_provider
    pattern: ^/
    custom_authenticator: App\Security\LoginFormAuthenticator
    # Configurer la déconnexion
    logout:
      path: /logout
```

Dans le pare feu nous disposons de 2 environnement, « dev » et « main ».

Dans « main » nous retrouvons le système de connexion avec « app_user_provider » qui a été défini plus haut.
Il y a de présent l'authenticator qui ici se trouve dans « App\Security\LoginFormAuthenticator » et le chemin pour la déconnexion.

Access control :

```yaml
  access_control:
    - { path: ^/login$, roles: PUBLIC_ACCESS }
    - { path: ^/logout$, roles: IS_AUTHENTICATED }
    - { path: ^/admin, roles: ROLE_ADMIN }
```

Ici de défini dans l'access control, nous avons les routes /login qui est accessible pour tous, /logout seulement si on est authentifié et /admin si on a le rôle admin.

Role hierarchy :

```yaml
role_hierarchy:
  ROLE_ADMIN: ROLE_USER
```

Dans le role hierarchy, nous avons le fait qu'un admin aura aussi le rôle user même si en BDD, il n'aura que « ROLE_ADMIN »

SecurityController

Le controller Security gère les routes « app_login » et « app_logout ».

```php
#[Route(path: '/login', name: 'app_login', methods: ['GET', 'POST'])]
public function login(AuthenticationUtils $authenticationUtils): Response
{
    if ($this->getUser()) {
        return $this->redirectToRoute( route: 'homepage');
    }

    $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]
}
```

Dans la route « app_login », nous avons une condition qui vérifie si l'utilisateur est connecté et le redirige si c'est le cas.

LoginFormAuthenticator

La constante « LOGIN_ROUTE » défini la route de connexion.

Authenticate :

```php
public function authenticate(Request $request): Passport
{
    $username = $request->request->get( key: 'username', default: '');

    $request->getSession()->set(SecurityRequestAttributes::LAST_USERNAME, $username);

    return new Passport(
        new UserBadge($username),
        new PasswordCredentials($request->request->get( key: 'password', default: '')),
        [
            new CsrfTokenBadge( csrfTokenId: 'authenticate', $request->request->get( key: '_csrf_token')),
        ]
    );
}
```

Cette méthode renvoie un Passport (class Symfony) en ayant récupéré les informations suivantes du formulaire :

- Username
- Password
- Csrf token

OnAuthenticationSuccess :

```php
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    return new RedirectResponse($this->urlGenerator->generate( name: 'homepage'));
}
```

Redirige l’utilisateur vers la page qu’il essayait d’atteindre avant de se connecter ou vers la page d’accueil.