Documentation technique ToDo & Co

Authentification

Auteur : Ferreira José
Date de dernière mise à jour : 14/10/2022

Lien documentation :

Security (Symfony Docs)

Prérequis :

- Librairie : symfony/security-bundle

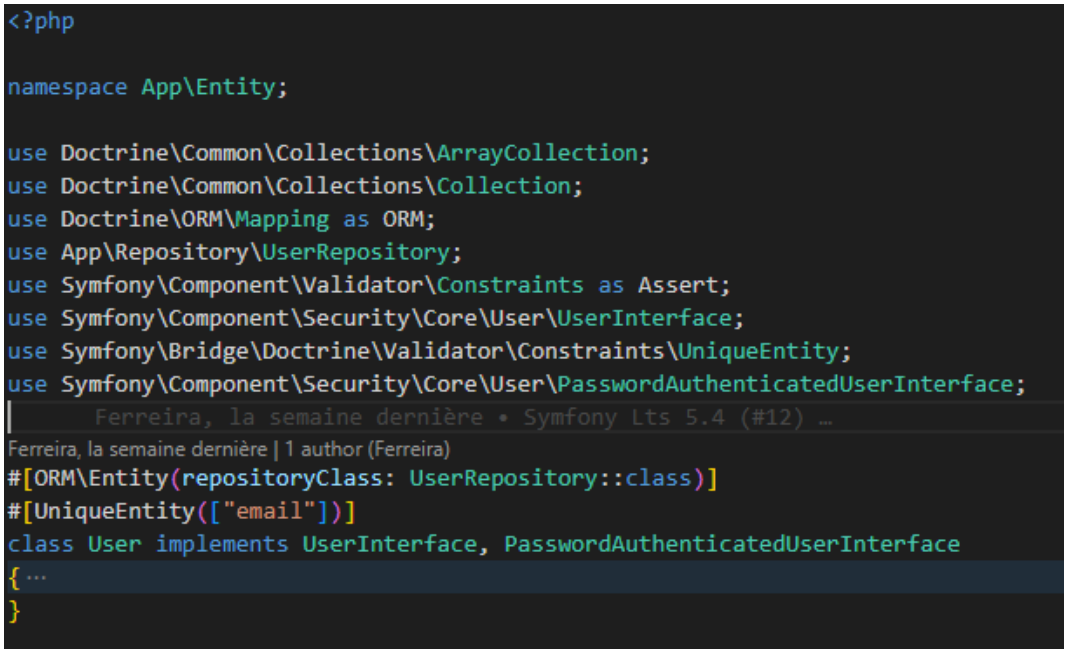
L'authentification pour l'application ToDo & Co fonctionne avec le bundle Symfony ci-dessous.

Fichier nécessaire à l'authentification :

- src/Entity/User.php
- config/packages/security.yaml
- src/Controller/SecurityController.php
- templates/security/login.html.twig

User.php

La class User représente notre utilisateur elle doit implémenter la UserInterface et la PasswordAuthenticatedUserInterface afin d'être valide pour le bundle security de Symfony et implémenter les fonctions nécessaires.



L'entité doit être modifiée dans le cadre d'ajout de champs en base de données ou pour changer le fonctionnement par défaut tels que les rôles, la méthode getUserIdentifier qui permet de retourner la valeur qui permet d'identifier l'utilisateur comme l'email, username.

Security.yaml

Le fichier security.yaml sert à configurer le comportement que doit avoir le bundle security et le type d'authentification utilisé.

Vous devez lui configurer plusieurs chose afin que celui-ci fonctionne correctement :

La configuration doit être mise à jour dans le cadre de changement de noms de route d'authentification, si vous changez de système d'authentification ou le provider.

Providers (providers)

Ce fournisseur d'utilisateurs sait comment (re)charger des utilisateurs à partir d'un stockage (par exemple une base de données) sur la base d'un identifiant d'utilisateur" (par exemple l'adresse e-mail ou le nom d'utilisateur de l'utilisateur). La configuration ci-dessus utilise Doctrine pour charger l' User entité en utilisant le username comme identifiant d'utilisateur.

Pour notre provider nous utilisons l'entité User, et l'identifiant unique est le username ce qui donne ceci :

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

Password Hashers (password_haschers)

Indique quel hachage doit utiliser le bundle afin d'utiliser le bon service sur l'entité User pour vérifier le mot de passe de l'utilisateur.

Pour le système de hachage de mot de passe nous laissons Symfony choisir le meilleur algorithme de hachage disponible et nous configurons notre entité User ce qui donne ceci :

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

Firewalls (firewalls)

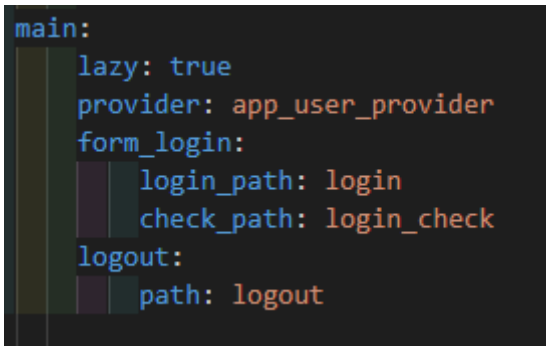
Un « pare-feu » est votre système d'authentification : le pare-feu définit quelles parties de votre application sont sécurisées et comment vos utilisateurs pourront s'authentifier.

Nous utilisons un formulaire de connexion sur ToDo & Co pour ce faire nous devons lui spécifié 3 éléments obligatoire dans le firewall :

- Le provider à utiliser
- Le type d'authentification les 3 principaux sont :
    - Form Login (form_login)
    - JSON Login (json_login)
    - HTTP Basic (http_basic)
- La route de déconnexion afin que l'utilisateur soit déconnecté

Dans le form_login nous devons spécifier 2 choses login_path et check_path c'est 2 routes seront configurer dans le contrôleur SecurityController, mais la check_path ne sera jamais appelé car elle déclenche un événement qui permet au bundle de vérifier les informations soumise par l'utilisateur.

Notre firewall devrait ressembler à ceci :



SecurityController.php

Notre SecurityController est là uniquement pour afficher le formulaire de connexion et créer les 2 autres routes login_check et logout mais qui sont des évènement donc les fonctions ne sont jamais appelé.

La class SecurityController ne devrait pas être modifiée mais si vous désirez vous pouvez changer les paths afin de changer les urls.

Notre controller ressemble à ceci :

```php
class SecurityController extends AbstractController
{
    #[Route(path: '/login', name: 'login')]
    public function loginAction(AuthenticationUtils $authenticationUtils): Response
    {
        if ($this->getUser() != null) {
            return $this->redirectToRoute('homepage');
        }

        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();

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

    #[Route(path: '/login_check', name: 'login_check')]
    public function loginCheck()
    {
        // This code is never executed.
    }

    #[Route(path: '/logout', name: 'logout')]
    public function logoutCheck()
    {
        // This code is never executed.      You, le mois dernier • Clone projet from
    }
}
```

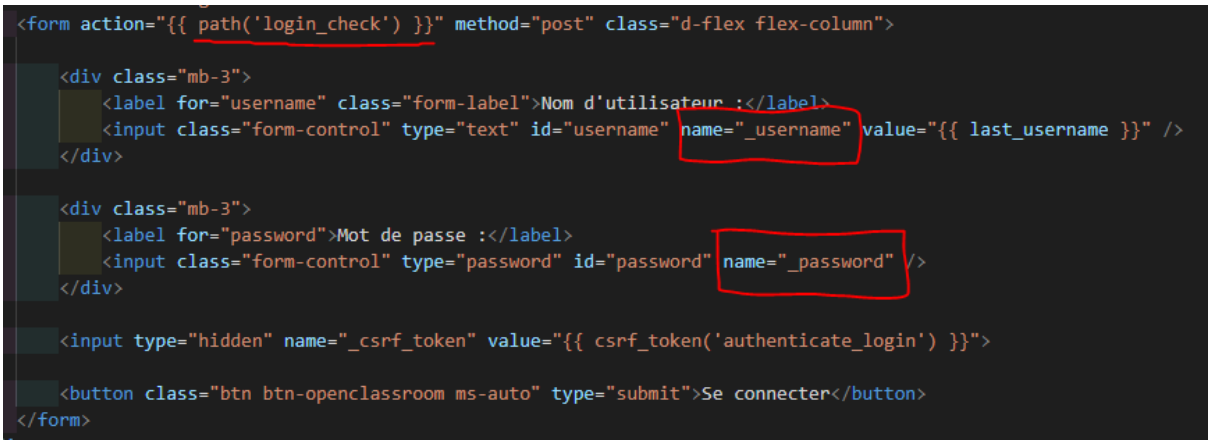
Security/login.html.twig

Le fichier de templates login.html.twig sert à faire le rendu HTML de la page /login avec le formulaire de connexion.

Le fichier peut être modifier à volonté car vous ne faite que modifier la structure HTML mais vous devez respectez 2 points qui sont :

- L'action du formulaire doit être login_check si vous voulez changer vous devrait modifier le SecurityController et la security configuration pour le login_check
- Vous devez respectez la charte de nommage pour les name des inputs du formulaire pour le username et le password elle devront être écrite comme ceci :
    - _username
    - _password

Si vous respectez ses 2 règles votre formulaire devrait toujours avoir cette structure :



```
<form action="{{ path('login_check') }}" method="post" class="d-flex flex-column">

    <div class="mb-3">
        <label for="username" class="form-label">Nom d'utilisateur :</label>
        <input class="form-control" type="text" id="username" name="_username" value="{{ last_username }}" />
    </div>

    <div class="mb-3">
        <label for="password">Mot de passe :</label>
        <input class="form-control" type="password" id="password" name="_password" />
    </div>

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

    <button class="btn btn-openclassroom ms-auto" type="submit">Se connecter</button>
</form>
```

Stockage utilisateur :

Les utilisateurs sont stockés dans une base de données MySQL afin que l'application puisse se connecter à celle-ci pour récupérer les informations de l'utilisateur.

id	username	password	email	roles
1	Admin	$2y$13$YEvlqvNI5JN/8q/iZk0hl.81uKOFQZQKFOmkphXW7un...	admin@test.fr	["ROLE_ADMIN"]
2	Test	$2y$13$JgEaTQ6P9MB0lL9xQmi90ONXcbMcbWHPgjrjeUsX8wD...	test@test.fr	[]

Bonne pratique :

Le code que vous produirez devra respecter les normes PSR-1, PSR-2, PSR-4, PSR-12 afin de respecter ces normes vous pouvez utiliser l'outil PHP CS FIXER.

Vous veillerez à contrôler que vos commit soit vérifier par un collègue et via Codacy.

Pour aller plus loin vous pouvez vous inspirer des bonnes pratique Symfony