AMÉLIOREZ UNE APPLICATION EXISTANTE DE TODO & CO

PROJET 8

DÉVELOPPEUR D'APPLICATION - PHP / SYMFONY

DOCUMENTATION TECHNIQUE

Sommaire

1. Présentation Projet 3

1.1 Technologies 3

1.2 Librairies 3

2. Paramétrage du site 3

2.1 Introduction 3

2.2 Fichiers d'authentification 3

2.3 L'entité User 3

2.4 Les Providers 3

2.5 Les encoders 4

2.6 Les Firewalls 4

2.7 Les Access_Control 4

2.8 Les Role_Hierarchy 5

1. Présentation Projet

1.1 Technologies

Le projet a été mis à jour de la version 3 vers une version de Symfony plus récente et stable : v5.4.
Pour le bon fonctionnement du projet, la version 7.3.2 de PHP est requise.

1.2 Librairies

Les libraires sont toutes installés par Composer, et sont donc visible sur le site https://packagist.org.
Toutes les librairies sont listées dans le fichier composer.json à la racine du projet, il faut pas hésiter à faire des mises à jours, car cela peut corriger de potentielles failles de sécurité. Cependant, une consultation de la documentation est obligatoire pour éviter toutes complications ! Par exemple un conflit de dépendance.

2. Paramétrage du site

2.1 Introduction

La sécurité concernant l'authentification est configurée dans le fichier config/packages/security.yaml.
Documentation officielle de Symfony, pour plus d'information

2.2 Fichiers d'authentification

Type	Fichier	Description
Configuration	config/packages/security.yaml	Configuration du processus d'authentification
Entité	src/Entity/User.php	Entité utilisateur
Contrôleur	src/Controller/SecurityController.php	Contrôleur connexion/déconnexion
Authentification	src/Security/LoginFormAuthenticator.php	Méthodes du processus d'authentification de l'application
Vue	templates/security/login.html.twig	Template du formulaire de connexion

2.3 L'entité User

Avant toute de chose, il est nécessaire d'avoir défini une entité qui représentera l'utilisateur connecté. Cette classe doit implémenter l'interface « UserInterface » et donc implémenter les différentes méthodes définis dans celle-ci. Dans ce cas-ci, cette classe a déjà été implémentée et se situe dans le fichier « src/Entity/User.php ».

2.4 Les Providers

Un provider va nous permettre d'indiquer où se situe les informations que l'on souhaite utiliser pour authentifier l'utilisateur, dans ce cas-ci, on indique qu'on récupérera les utilisateurs grâce à l'entité « User » dont la propriété « username » sera utilisé pour s'authentifier sur le site. Attention, on peut indiquer ici la classe User car celle-ci implémente l'interface Userinterface |

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

2.5 Les encoders

Un encoder va simplement nous permettre de déterminer quel est l'algorithme que l'on souhaite utilisé lors de l'encodage d'une certaine information dans une certaine entité. L'encryptage est en mode automatique pour le moment, ce qui correspond au meilleur choix pour la version de Symfony du site. Il est bien entendu possible de changer le mode dans le fichier de configuration « secrity.yaml » dans le bloc « **password_hashers** ».

```
password_hashers:
        # Use native password hasher, which auto-selects and migrates the best
        # possible hashing algorithm (which currently is "bcrypt")
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

2.6 Les Firewalls

Un firewall va définir comment nos utilisateurs vont être authentifiés sur certaines parties du site. Le firewall **dev** ne concerne que le développement ainsi que le profiler et ne devra à priori pas être modifié. Le firewall **main** englobe l'entièreté du site à partir de la racine défini via **pattern:** ^/, l'accès y est autorisé en anonyme
C'est-à-dire sans être authentifié, on y indique que c'est le provider "doctrine" qui sera utilisé. Afin de s'authentifier, on définit un formulaire de connexion via **form login:** où sont indiqués le nom des routes correspondant à ce formulaire, la route de vérification du login ainsi que la route vers laquelle l'utilisateur devra être redirigé par défaut après son authentification.

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

    main:
        lazy: true
        provider: app_user_provider
        form_login:
            login_path: login
            check_path: login_check
        logout:
            path:   /logout
            target: /
```

2.7 Les Access_Control

Un **access_control** va définir les limitations d'accès à certaines parties du site. Dans ce cas-ci, on indique que : - L'url /login est accessible sans authentification. - **L'url /users** n'est accessible qu'en étant authentifié avec un utilisateur ayant le rôle "ROLE_ADMIN". - Tout le reste du site n'est accessible qu'aux utilisateurs authentifiés c-à-d ayant le rôle "ROLE_USER".

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

2.8 Les Role_Hierarchy

Un **role_hierarchy** permet de s'assurer qu'un utilisateur ayant un certain rôle aura automatiquement d'autres rôles. Dans ce cas-ci, un utilisateur possédant le rôle "ROLE_ADMIN"' aura automatiquement le rôle "ROLE_USER".

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

3. Les bases de données

3.1 Modifications de la base de données

Pour un éventuel changement du nom de base de données. Il faudra changer les informations de connexion dans le fichier. env à la racine du projet.

```
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
```

Détail configuration BDD :
https://symfony.com/doc/current/doctrine.html#configuring-thedatabase

3.2 Stockage des utilisateurs

Les données utilisateurs sont stockés dans une base de données MySql. Plus précisément dans la table user. Il faut savoir que le champ **Username** est unique et il est possible de rajouter un autre champ unique. Il faudra simplement aller dans l'entité user : src/entity/user, et de rajouter
« unique = true » à l'annotation **@Column**.

Exemple :

```
/**
*@ORM\Column(type="string", length=180, unique=true)
*/
private $username;
```

4. Fonctionnement de l'authentification

4.1 Page authentification

Dans un premier temps, la personne va devoir rentrer ces identifiants sur la page ***login.html.twig*** à la route ***/login***. Pour accéder à cette page, la méthode login est exécutée, sur le Controller : ***SecurityController***. Cette méthode sert à générer la page, et à envoyer des données à la vue avec un render.

4.2 Soumission du formulaire

Quand le formulaire est validé par l'utilisateur, les données sont récupérées par la classe ***LoginFormAuthenticator***, qui se charge de l'authentification. Cette classe est générique à la création du système d'authentification de la librairie ***Security-Bundle***.
L'authentification s'opère de cette manière :

- Récupération des champs du formulaire grâce à la méthode getCredentials
- Récupération de l'utilisateur dans la BDD par son username, en passant par la vérification de validité d'un token Csrf à la méthode getUser
- Vérification du password associé à l'utilisateur par la méthode checkCredentials

4.3 Redirection de l'utilisateur

Suivant ci, la personne c'est bien authentifier ou pas. La méthode ***onAuthenticationSuccess***, la redirigera vers la dernière page qu'elle a consulté, ou la page de défaut, en cas de succès. A contrario, un message d'erreur sera affiché au-dessus du formulaire de connexion.

Détail création login form : https://symfony.com/doc/current/security/form_login_setup.html