Authentification Guide

La sécurité concernant l'authentification est configurée dans le fichier `config/packages/security.yaml` Vous trouverez plus d'informations concernant ce fichier et ses différentes parties dans la documentation officielle de Symfony.

L'entité User

Avant toute de chose, il est nécessaire d'avoir défini une entité qui représentera l'utilisateur connecté. On crée une entité en passant par le terminal et la console Symfony : `php bin/console make:entity` 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 la fichier `src/Entity/User.php`.

Security - password_hashers:

Habituellement, le même hacheur de mot de passe est utilisé pour tous les utilisateurs en le configurant pour qu'il s'applique à toutes les instances d'une classe spécifique. Une autre option consiste à utiliser un hachage "nommé", puis à sélectionner le hachage que vous souhaitez utiliser dynamiquement.

Par défaut, l' autoalgorithme est utilisé pour App\Entity\User. Il sélectionne automatiquement le meilleur hasher disponible (actuellement Bcrypt). Il produit des mots de passe hachés avec la fonction de hachage de mot de passe bcrypt . Les mots de passe hachés contiennent des 60caractères, alors assurez-vous d'allouer suffisamment d'espace pour qu'ils soient conservés. Sa seule option de configuration est cost, qui est un entier compris dans la plage de 4-31(par défaut, 13). Chaque augmentation du coût double le temps nécessaire pour hacher un mot de passe. Vous pouvez modifier le coût à tout moment, même si vous avez déjà haché des mots de passe avec un coût différent.

```
security:
    password_hashers:
       App\Entity\User: 'auto'
```

Les Providers

Un provider va nous permettre d'indiquer où se situe les informations que l'on souhaite utiliser pour authentifier l'utilisateur. On indique qu'on récupérera les utilisateurs via Doctrine grâce à l'entité User et que la propriété Email sera utilisé pour s'authentifier sur le site. Plusieurs providers peuvent-être configurés.

config/packages/security.yaml

```
    providers:
       users:
           entity:
               class: App\Entity\User
               property: email
```

Les Firewalls

Le pare-feu est au cœur de la sécurisation de votre application. Chaque demande au sein du pare-feu est vérifiée si elle nécessite un utilisateur authentifié. Le pare-feu se charge également d'authentifier cet utilisateur.

anonymous : Défini si l'on peut-être connecté comme utilisateur anonyme sur l'application. pattern : Une regex définissant les URL filtrées. Ici toutes les URL sont filtrés.

La plupart des sites Web ont un formulaire de connexion où les utilisateurs s'authentifient à l'aide d'un identifiant (par exemple, une adresse électronique ou un nom d'utilisateur) et d'un mot de passe. Cette fonctionnalité est fournie par le form login authenticator. Le login_path et le check_path prennent en charge les URL et les noms de route. Une fois activé, le système de sécurité redirige les visiteurs non authentifiés vers le login_pathlorsqu'ils tentent d'accéder à un lieu sécurisé Symfony fournit une protection de base contre les attaques de connexion par force brute . Vous devez l'activer en utilisant le login_throttling paramètre : logout : Autorise la déconnexion.

Exemple de processus : L'utilisateur tente d'accéder à une ressource protégée (par exemple /admin) ; Le pare-feu initie le processus d'authentification en redirigeant l'utilisateur vers le formulaire de connexion (/login) ; La /loginpage affiche le formulaire de connexion via la route et le contrôleur créés dans cet exemple ; L'utilisateur soumet le formulaire de connexion à /login; Le système de sécurité (c'est-à-dire l' form_loginauthenticateur) intercepte la demande, vérifie les informations d'identification soumises par l'utilisateur, authentifie l'utilisateur si elles sont correctes et renvoie l'utilisateur au formulaire de connexion si elles ne le sont pas.

```
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
            lazy: true
            provider: users
            form_login:
                login_path: login
                check_path: login
            login_throttling:
                max_attempts: 3
                interval: '15 minutes'
            logout:
                path: logout
```

Les Access_Control

La façon la plus simple de sécuriser une partie de l'application est de sécuriser un modèle d'URL entier Dans ce cas-ci, on indique que : - L'url /admin n'est accessible qu'en étant authentifié avec un utilisateur ayant le rôle "ROLE_ADMIN". - L'url /tasks n'est accessible qu'aux utilisateurs authentifiés c-à-d ayant le rôle "ROLE_USER". - Le reste du site (login, register) est accéssible à tous

config/packages/security.yaml

```
    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
        - { path: ^/, roles: ROLE_USER }
```

Les Role_Hierarchy

Lorsqu'un utilisateur se connecte, Symfony appelle la méthode getRoles() sur l'entité utilisateur pour en déterminer ses rôles Dans ce cas-ci, un utiliseur possédant le rôle "ROLE_ADMIN" aura automatiquement le rôle "ROLE_USER".

```
# config/packages/security.yaml
role_hierarchy:
        ROLE_ADMIN: ROLE_USER
```

Contrôle d'accés dans les templates Twig

Pour vérifier si l'utilisateur actuel a un rôle dans un modèle, utilisez la fonction is_granted() :

```
{% if is_granted('ROLE_ADMIN') %}
<a href="...">Delete</a>
{% endif %}
```

Vérifier si un utilisateur est connecté

vous pouvez utiliser un "attribut" spécial à la place d'un rôle :

```
public function adminDashboard(): Response
{
    $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');

    // ...
}
```