

SecurityBundle



Introduction

Un « bundle » est un module compatible avec n'importe quelle application Symfony. Il contient une multitude d'éléments comme des contrôleurs, des templates, des règles de routage ou des services. Un bundle est une sorte de plug-in.

Le SecurityBundle, fournit toutes les fonctionnalités d'authentification et d'autorisation nécessaires pour sécuriser votre application.

Deux notions majeures interviennent dans la conception de sécurité de Symfony :

- **Authentification** : Qui êtes vous ? ; vous pouvez vous authentifier de plusieurs manières (HTTP authentification, certificat, formulaire de login, API, OAuth etc).
- **Autorisation** : Avez vous accès à ? ; permet d'autoriser de faire telle ou telle action ou accéder à telle page sans forcément savoir qui vous êtes, utilisateur anonyme par exemple.

La sécurité dans symfony implique plusieurs éléments :

- **Le firewall** : qui est la porte d'entrée pour le système d'authentification, il va permettre de mettre en place le bon système de connexion pour l'url spécifiée via un pattern.
- **Le provider** : qui permet au firewall d'interroger une collection d'utilisateurs/mot de passe ; C'est une sorte de base de tous les utilisateurs avec les mots de passe. Il existe deux type par défaut :
 - *in memory* : directement dans le fichier security.yaml, les hash des mots de passes sont disponible dans un fichier.
 - *Entity* : n'importe quelle entité qui implémente à minima.
- **Un encoder** : qui permet de générer des hashes/d'encoder des mots de passe.
- **Les rôles** : qui permettent de définir le niveau d'accès des utilisateurs connectés (authentifiés) et de configurer le firewall en fonction de ces rôles. Les rôles peuvent être hiérarchisés.

Installation

Pour fonctionner, il est nécessaire d'ajouter le security-bundle via la commande :

```
$ composer require symfony/security-bundle
```

Ensuite il faut créer l'Entity User :

```
-> % php bin/console make:user

The name of the security user class (e.g. User) [User]:
> User

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
> yes

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
>

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).

Does this app need to hash/check user passwords? (yes/no) [yes]:
>
```

Cette entité est directement liée au SecurityBundle.

```
k?php

namespace App\Entity;

use ...

/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 * @UniqueEntity(fields={"email"}, message="There is already an account with this email")
 * @method string getUserIdentifier()
 * @Serializer\ExclusionPolicy("all")
 */
class User implements UserInterface
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     * @Serializer\Expose()
     */
    private $id;

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

    /**
     * @ORM\Column(type="json")
     * @Serializer\Expose()
     */
    private $roles = [];
```

Le fichier de configuration security.yml décrit les règles d'authentification et d'autorisation pour une application symfony.

```

1 security:
2   role_hierarchy:
3     ROLE_VISITEUR: [ROLE_USER]
4     ROLE_RH: [ROLE_USER]
5     ROLE_RD: [ROLE_USER]
6     ROLE_ADMIN: [ROLE_USER, ROLE_RH, ROLE_VISITEUR, ROLE_RD]
7     ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_RH, ROLE_ADMIN, ROLE_VISITEUR, ROLE_RD]
8
9   encoders:
10    App\Entity\User:
11      algorithm: auto
12
13    # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
14   providers:
15     # used to reload user from session & other features (e.g. switch_user)
16    app_user_provider:
17      entity:
18        class: App\Entity\User
19        property: email
20   firewalls:
21     dev:
22       pattern: ^/(_(profiler|wdt)|css|images|js)/
23       security: false
24     main:
25       anonymous: true
26       lazy: true
27       provider: app_user_provider
28       guard:
29         authenticators:
30           - App\Security\AppCustomAuthenticator
31       logout:
32         path: app_logout
33         # where to redirect after logout
34         # target: app_any_route
35
36     # activate different ways to authenticate
37     # https://symfony.com/doc/current/security.html#firewalls-authentication
38
39     # https://symfony.com/doc/current/security/impersonating_user.html
40     # switch_user: true
41
42     # Easy way to control access for large sections of your site
43     # Note: Only the *first* access control that matches will be used
44     access_control:
45       - { path: ^/admin, roles: ROLE_ADMIN }
46       - { path: ^/rh, roles: ROLE_RH }
47       - { path: ^/visiteur, roles: ROLE_VISITEUR }
48       - { path: ^/rapport/visiteur, roles: ROLE_VISITEUR }
49       - { path: ^/rapport, roles: [ROLE_VISITEUR, ROLE_RD] }
50       - { path: ^/secured, roles: ROLE_USER }
51

```

Il faut à présent mettre la base de données à jour :

```
$ php bin/console doctrine:schema:update --force
```

Mise en place de la partie connexion :

```
$ php bin/console make:auth
```

```

-> % php bin/console make:auth

What style of authentication do you want? [Empty authenticator]:
 [0] Empty authenticator
 [1] Login form authenticator
> 1

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> LoginController

The entry point for your firewall is what should happen when an anonymous user tries to access
a protected page. For example, a common "entry point" behavior is to redirect to the login page.
The "entry point" behavior is controlled by the start() method on your authenticator.
However, you will now have multiple authenticators. You need to choose which authenticator's
start() method should be used as the entry point (the start() method on all other
authenticators will be ignored, and can be blank. [App\Security\AppCustomAuthenticator]:
 [0] App\Security\AppCustomAuthenticator
 [1] App\Security>LoginControllerAuthenticator
> 1

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
>

Do you want to generate a '/logout' URL? (yes/no) [yes]:
>

```

Comme indiqué cette commande va créer plusieurs fichiers :

- **src/Security/LoginAuthenticator.php** : qui va contenir la logique de votre authentification. Que faire une fois l'authentification réussie, ou en cas d'échec. Comment récupérer les informations de l'utilisateur.
- **src/Controller/SecurityController.php** : qui va être le contrôleur gérant la partie sécurité et authentification. Par défaut la méthode login pour afficher le formulaire et traiter (avec l'aide de l'authenticator précédent), valider les données. C'est dans ce contrôleur que vous pouvez ajouter la déconnexion, l'enregistrement et le mot de passe perdu par exemple.
- **templates/security/login.html.twig** : la vue contenant le formulaire de connexion, que vous pouvez librement adapter.
- Mettre à jour le fichier **security.yaml** afin qu'il fasse le lien avec les différents éléments de sécurité.

Ainsi lors de la création d'un user, l'ajout de l'interface **UserPasswordEncoderInterface** permettra d'encoder le mot de passe.

```

/**
 * @Route("/new")
 */
public function newUser(Request $request, UserPasswordEncoderInterface $passwordEncoder ){
    $user = new User();
    $form = $request->request->get( key: "form");

    $user->setEmail($form["email"])
        ->setFirstname($form["firstname"])
        ->setName($form["name"])
        ->setPhone($form["phone"])
        ->setRoles($form["roles"])
        ->setPassword($passwordEncoder->encodePassword(
            $user, $form["password"]
        ));
    $this->getDoctrine()->getManager()->persist($user);
    $this->getDoctrine()->getManager()->flush();

    return $this->sendJson( data: 'success', status: 200);
}

```