M4D203D - POO Avancée

David Annebicque

2016

1 Sécurité dans Symfony et installation d'un bundle tiers

1.1 Gérer la sécurité dans Symfony

La sécurité dans Symfony est un concept très poussé, que l'on peut personnaliser très finement, mais qui reste très simple à mettre en place.

Symfony se base sur deux concepts pour gérer la sécurité : l'authentification et l'autorisation.

1.1.1 L'authentification

Ce processus permet de définir qui est le visiteur présent sur le site. Cette procédure permet de dire si vous êtes un utilisateur anonyme (non identifié/authentifié) soit un utilisateur connecté (via un formulaire, un cookie, ...). Cette authentification se fait par l'intermédiaire d'un *firewall* dans Symfony.

Grâce a ce firewall vous pouvez restreindre tout ou partie de votre site aux utilisateurs authentifiés. S'ils ne le sont pas Symfony les redirigera vers une page d'authentification.

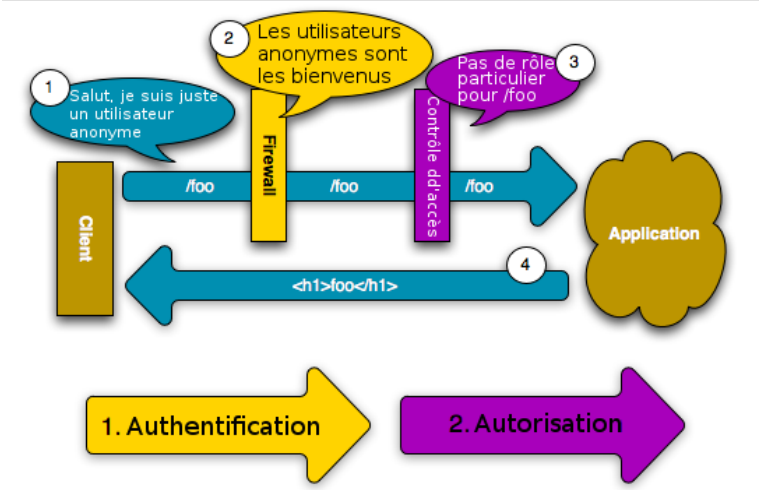
1.1.2 L'autorisation

Ce processus se déclenche une fois que vous êtes authentifié. Il va regarder si l'utilisateur est autorisé à accéder à la page demandée. C'est ce que Symfony appelle l'*Access Control*.

Cette autorisation permet par exemple d'autoriser certains membres à poster des éléments sur votre site, et d'autres à modifier et supprimer des éléments du site.

1.1.3 Exemple

Exemple 1 : Je suis anonyme, et je veux accéder à la page /foo qui ne requiert pas de droits (source Symfony.com / OpenClassRoom)

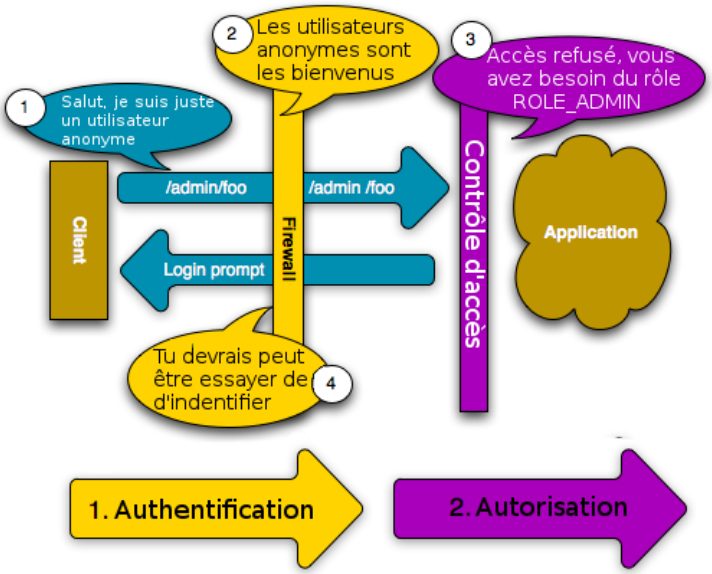


Dans cet exemple, un visiteur anonyme souhaite accéder à la page /foo. Cette page ne requiert pas de droits particuliers, donc tous ceux qui ont réussi à passer le firewall peuvent y avoir accès. La figure ci-dessus montre le processus.

Sur ce schéma, vous distinguez bien le firewall d'un côté et l'access control (contrôle d'accès) de l'autre. Reprenons-le ensemble pour bien comprendre :

1. Le visiteur n'est pas identifié, il est anonyme, et tente d'accéder à la page /foo.
2. Le firewall est configuré de telle manière qu'il n'est pas nécessaire d'être identifié pour accéder à la page /foo. Il laisse donc passer notre visiteur anonyme.
3. Le contrôle d'accès regarde si la page /foo requiert des droits d'accès : il n'y en a pas. Il laisse donc passer notre visiteur, qui n'a aucun droit particulier.
4. Le visiteur a donc accès à la page /foo.

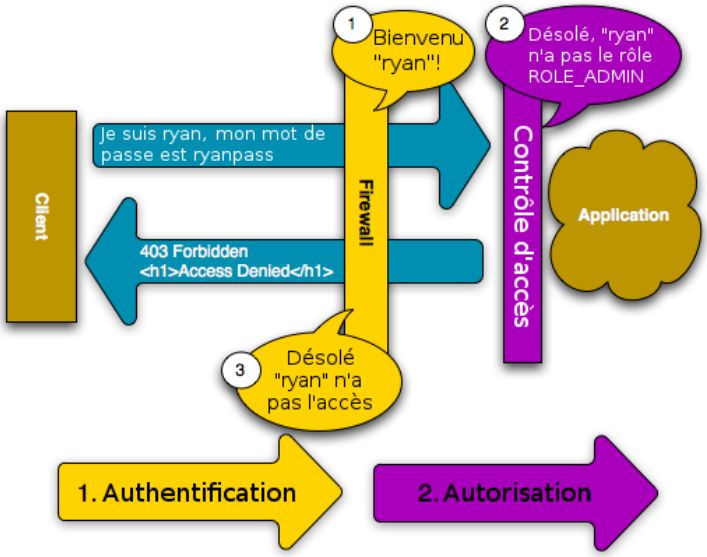
Exemple 2 : Je suis anonyme, et je veux accéder à la page /admin/foo qui requiert certains droits



Dans cet exemple, c'est le même visiteur anonyme qui veut accéder à la page /admin/foo. Mais cette fois, la page /admin/foo requiert le rôle ROLE_ADMIN; c'est un droit particulier, un niveau d'autorisation requis. Notre visiteur va se faire refuser l'accès à la page, la figure ci-dessus montre comment.

1. Le visiteur n'est pas identifié, il est toujours anonyme, et tente d'accéder à la page /admin/foo.
2. Le firewall est configuré de manière qu'il ne soit pas nécessaire d'être identifié pour accéder à la page /admin/foo. Il laisse donc passer notre visiteur.
3. Le contrôle d'accès regarde si la page /admin/foo requiert des droits d'accès : oui, il faut le rôle ROLE_ADMIN. Le visiteur n'a pas ce rôle, donc le contrôle d'accès lui interdit l'accès à la page /admin/foo.
4. Le visiteur n'a donc pas accès à la page /admin/foo, et se fait rediriger sur la page d'identification.

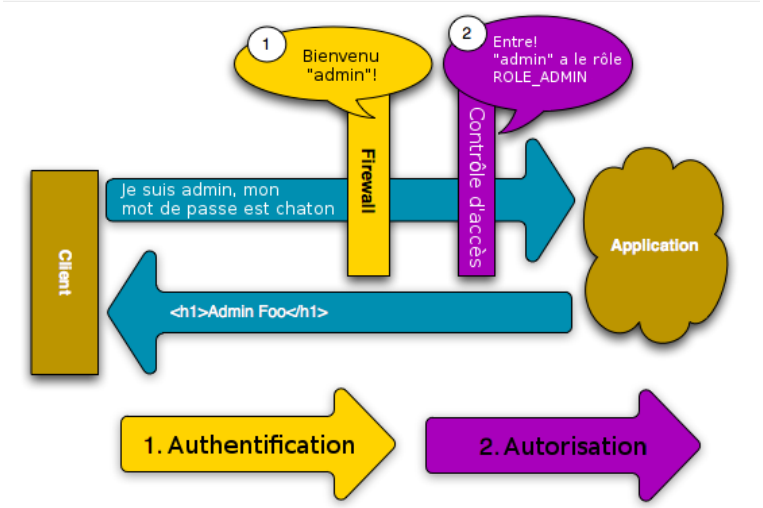
Exemple 3 : Je suis identifié, et je veux accéder à la page /admin/foo qui requiert certains droits



Cet exemple est le même que précédemment, sauf que cette fois notre visiteur est identifié, il s'appelle Ryan. Il n'est donc plus anonyme.

1. Ryan s'identifie et il tente d'accéder à la page /admin/foo. D'abord, le firewall confirme l'authentification de Ryan (c'est son rôle !). Visiblement c'est bon, il laisse donc passer Ryan.
2. Le contrôle d'accès regarde si la page /admin/foo requiert des droits d'accès : oui, il faut le rôle ROLE_ADMIN, que Ryan n'a pas. Il interdit donc l'accès à la page /admin/foo à Ryan.
3. Ryan n'a pas accès à la page /admin/foo non pas parce qu'il ne s'est pas identifié, mais parce que son compte utilisateur n'a pas les droits suffisants. Le contrôle d'accès lui affiche donc une page d'erreur lui disant qu'il n'a pas les droits suffisants.

Exemple 4 : Je suis identifié, et je veux accéder à la page /admin/foo qui requiert des droits que j'ai.



Ici, nous sommes maintenant identifiés en tant qu'administrateur, on a donc le rôle ROLE_ADMIN ! Du coup, nous pouvons accéder à la page /admin/foo, comme le montre la figure ci-dessus.

1. L'utilisateur admin s'identifie, et il tente d'accéder à la page /admin/-foo. D'abord, le firewall confirme l'authentification d'admin. Ici aussi, c'est bon, il laisse donc passer admin.
2. Le contrôle d'accès regarde si la page /admin/foo requiert des droits d'accès : oui, il faut le rôle ROLE_ADMIN, qu'admin a bien. Il laisse donc passer l'utilisateur.
3. L'utilisateur admin a alors accès à la page /admin/foo, car il est identifié et il dispose des droits nécessaires.

1.1.4 Principe général

Lorsqu'un utilisateur tente d'accéder à une ressource protégée, le processus est finalement toujours le même, le voici :

1. Un utilisateur veut accéder à une ressource protégée ;
2. Le *firewall* redirige l'utilisateur au formulaire de connexion ;
3. L'utilisateur soumet ses informations d'identification (par exemple login et mot de passe) ;
4. Le firewall authentifie l'utilisateur ;
5. L'utilisateur authentifié renvoie la requête initiale ;
6. L'*access control* vérifie les droits de l'utilisateur, et autorise ou non l'accès à la ressource protégée.

Ces étapes sont simples, mais très flexibles. Il est en effet possible d'utiliser toutes les méthodes d'authentification possible. Des méthodes que vous développez : un formulaire de connexion classique, ou des méthodes d'authentifications via Facebook, Google, etc., ou encore via les certificats X.509, etc.

1.2 Fichier de sécurité (security.yml)

Une bonne partie de la configuration de votre sécurité se fait dans un fichier sépcifique qui se trouve dans le répertoire app/config de votre projet. C'est le fichier security.yml

```
# app/config/security.yml
security:
  encoders:
    Symfony\Component\Security\Core\User\User: plaintext

  role_hierarchy:
    ROLE_ADMIN:       ROLE_USER
    ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]

  providers:
    in_memory:
      memory:
        users:
          user:  { password: userpass, roles: [ 'ROLE_USER' ] }
          admin: { password: adminpass, roles: [ 'ROLE_ADMIN' ] }

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

  access_control:
      #- { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY, requires_channel: ht
```

Ce fichier ci-dessus est le fichier par défaut. Il ne fait pas grand chose dans les faits, c'est juste un exemple.

Notez qu'on retrouve les deux notions évoquées précédemment : firewall et access control. A noter également que ce fichier permet en fait de configurer un bundle de symfony (SecurityBundle).

Vous pouvez lire la documentation officielle de Symfony pour comprendre les possibilités de ces lignes.

Dans notre cas, nous allons utiliser un bundle qui permet de mettre en place la sécurité, avec tous les formulaires et la table nécessaire.

1.3 Exemple de FosUserBundle

Comme annoncé dès l'introduction du cours sur Symfony, il est possible d'intégrer de nombreux bundles développés par la communauté. Certains répondent à des problématiques très spécifiques, d'autres sont des classiques tant ils répondent à des problématiques quasiment systématique sur les sites.

Vous trouvez la liste des bundles disponibles sur `http://knpbundles.com/`.

Dans cet exemple, nous allons installer FosUserBundle `http://knpbundles.com/FriendsOfSymfony/FOSUserBundle`

FOS : Friends Of Symfony est un groupe de développeur très actif dans la communauté Symfony qui propose plusieurs bundle très intéressant.

Cette communauté et ce bundle sont si indispensable dans Symfony que la documentation est maintenant sur le site officiel de Symfony : `http://symfony.com/doc/current/bundles/FOSUserBundle/index.html`

A noter que selon votre application et vos besoin, utiliser ce bundle peut être inutile tant il apporte de fonctionnalités qui pourraient ne pas être nécessaire à votre application (gestion de profile, de groupe, ...).

Si vous souhaitez développer votre propre module de connexion, vous pouvez consulter le cours sur OpenClassRoom : `https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony/securite-et-gestion-de`

1.3.1 Installation

Etape 1 : Installation

L'installation d'un bundle se fait avec la console et composer :

```
1 composer require friendsofsymfony/user-bundle "~2.0@dev"
```

Vous pouvez spécifié la version souhaitée. Ici la 2.0 en version de développement. Grâce à cette commande composer va installer FOSUserBundle dans le répertoire vendor de votre projet. Il va également télécharger les autres bundles qui pourraient être nécessaire au bon fonctionnement.

Etape 2 : Activation du bundle Une fois l'installation terminée, il faut activer le bundle. Cela peut déjà être fait, mais il convient de vérifier. Pour cela dans le fichier app/AppKernel.php vous devez avoir (ou ajouter la ligne ci-dessous).

```php
<?php
// app/AppKernel.php

public function registerBundles()
{
  $bundles = array(
     // ...
     new FOS\UserBundle\FOSUserBundle(),
     // ...
  );
}
```

Etape 3 : Création d'une classe User Les deux étapes précédentes sont classiques à toute installation. A partir de cette étape c'est spécifique à FOSUserBundle.

FOSUserBundle nécessite que vous ajoutiez une classe dans vos entités pour gérer la table dans la base de données. Cette entité va étendre l'entité de FOSUserBundle. Vous pouvez donc ajouter librement des champs dans cette table.

Le code minimal de cet entité est le suivant :

```php
<?php
// src/AppBundle/Entity/User.php

namespace AppBundle\Entity;

use FOS\UserBundle\Model\User as BaseUser;
use Doctrine\ORM\Mapping as ORM;

/**
* @ORM\Entity
* @ORM\Table(name="fos_user")
*/
class User extends BaseUser
{
  /**
  * @ORM\Id
  * @ORM\Column(type="integer")
  * @ORM\GeneratedValue(strategy="AUTO")
  */
  protected $id;

```

```
22    public function __construct()
23    {
24      parent::__construct();
25      // your own logic
26    }
27  }
```

Etape 4 : Configuration du fichier security.yml

Il est maintenant nécessaire d'expliquer à Symfony comment authentifier et autoriser les visiteurs en utilisant FOSUserBundle.

Le fichier ci-dessous est le fichier minimal pour faire fonctionner FOSUserBundle:

```
# app/config/security.yml
security:
  encoders:
    FOS\UserBundle\Model\UserInterface: bcrypt

  role_hierarchy:
    ROLE_ADMIN:       ROLE_USER
    ROLE_SUPER_ADMIN: ROLE_ADMIN

  providers:
    fos_userbundle:
    id: fos_user.user_provider.username

  firewalls:
    main:
      pattern: ^/
      form_login:
        provider: fos_userbundle
        csrf_token_generator: security.csrf.token_manager

      logout:       true
      anonymous:    true

  access_control:
    - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/admin/, role: ROLE_ADMIN }
```

Etape 5 : Configuration de FOS

Dans le fichier de configuration, il faut ajouter quelques lignes pour dire :

— Le type de base de données
— Le nom du firewall (on peut avoir plusieurs firewall)
— Le nom complet de l'entité.

```
# app/config/config.yml
fos_user:
    db_driver: orm # other valid values are 'mongodb', 'couchdb' and 'propel'
    firewall_name: main
    user_class: AppBundle\Entity\User
```

Etape 6 : Import des routes de FOS

Il faut activer les routes vers le bundle (route pour le login, logout, ...). Ajouter ces lignes dans votre fichier de routing :

```
# app/config/routing.yml
fos_user:
    resource:     "@FOSUserBundle/Resources/config/routing/all.xml"
```

Etape 7 : Mise à jour de la base de données

Il faut mettre à jour votre base de données avec la nouvelle table :

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

Votre site est maintenant sécurisé, et l'accès aux routes commencant par "admin" nécessitera une authentification et une autorisation de niveau "ROLE_ADMIN".

Vous pouvez regarder dans la table la structure de FOSUserBundle.

Il est bien sûr possible de personnaliser les pages de login, d'inscription et de logout.