La gestion d'utilisateurs

Achref El Mouelhi

Plan

La sécurité

But de la sécurité

Empêcher un utilisateur d'accéder à une ressource à laquelle il n'a pas droit

La sécurité

But de la sécurité

Empêcher un utilisateur d'accéder à une ressource à laquelle il n'a pas droit

Deux étapes

- Qui veut accéder ?
- Est-ce qu'il a le droit d'accéder à cette ressource ?

La sécurité

Configuration de la sécurité

- Fichier `app/config/security.yml` (format par défaut
- Mais on peut aussi utiliser :
  - le format `XML`
  - les tableaux imbriqués de `PHP`

La sécurité

Configuration de la sécurité

- Fichier `app/config/security.yml` (format par défaut
- Mais on peut aussi utiliser :
  - le format `XML`
  - les tableaux imbriqués de `PHP`

Il est également possible de placer le contenu du fichier `security.yml` dans le fichier `app/config/config.yml`

Le fichier security.yml : contenu

```
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
    demo_login:
      pattern:  ^/demo/secured/
       login$
      security: false
    demo_secured_area:
      pattern:    ^/demo/secured/
      form_login:
        check_path:
         _demo_security_check
        login_path: _demo_login
      logout:
        path:   _demo_logout
        target: _demo
  access_control:
```

Il faut 4 espaces pour différencier les niveaux dans un fichier yml (on n'utilise jamais les tabulations)

Le fichier `security.yml` : contenu

Une section principale dans ce fichier : `security`

- La sécurité dans Symfony est gerée par un bundle `SecurityBundle`
- La configuration de ce fichier correspond en vrai à la configuration de ce bundle

La section `encoders`

```
encoders:
    Symfony\Component\Security\Core\User\User:
     plaintext
```

La section `encoders`

- c'est l'objet qui s'occupe des mots de passe
- il nous permet de choisir l'encodeur de nos mots de passe
- `plaintext` : encodeur par défaut utilisé par Symfony
- on peut utiliser `sha512` pour crypter les mots de passe

La section role_hierarchy

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

La section role_hierarchy

- Ça correspond à la notion de rôle et privilèges en Système de Gestion de Base de Données Relationnelle
- Chaque type d'utilisateur peut avoir un ou plusieurs rôles dans notre application
- Le contrôleur consulte cette partie pour vérifier si un utilisateur, qui demande une ressource, à le droit d'y accéder
- Le nom d'un rôle doit commencer par `Role_`

La section providers

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

La section providers

- Quand l'utilisateur s'authentifie, le système d'authentification essaye de faire correspondre les données saisies avec un ensemble d'utilisateurs.
- Les ensembles d'utilisateurs (définis par des fournisseurs) peuvent provenir de plusieurs sources :
  - une table de base de données
  - un service web
  - un fichier de configurations

La section providers

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

La section providers

Symfony2 comprend en standard deux des fournisseurs

- le premier appelé in_memory car les utilisateurs ne sont pas sauvegardés dans une BD mais plutôt dans un fichier de configuration
  - Deux utilisateurs sont définis ici : admin et user
  - Symfony fournit un Bundle FOSUserBundle qui permet de sauvegarder les utilisateurs
- un second qui charge ses utilisateurs d'une table de BD

La section firewalls

```
firewalls:
    dev:
        pattern:  ^/(_(profiler|wdt)|css|images|js)/
        security: false
    demo_login:
        pattern:  ^/demo/secured/login$
        security: false
    demo_secured_area:
        pattern:    ^/demo/secured/
        form_login:
            check_path: _demo_security_check
            login_path: _demo_login
        logout:
            path:   _demo_logout
            target: _demo
```

La section `firewalls`

La section `firewalls`

- Le rôle du pare-feu est de déterminer si un utilisateur (ne) doit (pas) être authentifié, et s'il doit l'être, de retourner une réponse à l'utilisateur afin d'entamer le processus d'authentification.
- Par défaut, 3 pare-feux sont définis
- Un pare-feu est activé lorsque l'URL d'une requête correspond à une expression régulière contenue dans la configuration du pare-feu
- Le `pattern: ^/...` est un masque d'URL qui désigne toutes les URL commençant par / et contenant la chaîne qui suit.

La section access_control

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

La section access_control

- C'est le composant qui s'occupe du contrôle d'accès aux données
- Le contrôle d'accès peut aussi se faire directement dans les contrôleurs

Exemple

Créons notre pare-feu et plaçons le après le `dev`

```
notStrict:
    pattern:    ^/
    anonymous: true
```

Signification

- `notStrict` : nom de notre pare-feu
- `pattern: ^/` : notre masque signifiant que toutes les URL sont protégées par ce pare-feu `notStrict`
- `anonymous: true` : les utilisateurs anonymes sont acceptés

Remarque

Si une url appartient aux patterns de deux pare-feux différents, alors elle sera protégée par celui est est défini avant.

Exemple

Rajoutons les données sur l'authentification

```
notStrict:
    pattern:   ^/
    anonymous: true
    form_login:
        login_path: login
        check_path: login_check
    logout:
        path:       logout
        target:     login
```

form_login

méthode d'authentification à utiliser pour ce pare-feu. elle a 2 options

- `login_path: login` : route du formulaire d'authentification
- `check_path: login_check` : route de validation du formulaire d'authentification

Exemple

Rajoutons les données sur l'authentification

```
notStrict:
    pattern:    ^/
    anonymous: true
    form_login:
        login_path: login
        check_path: login_check
    logout:
        path:       logout
        target:     login
```

logout

Par défaut il est impossible de se déconnecter une fois authentifié.
Pour ce, on définit une méthode de déconnexion qui a aussi 2 options

- `path: logout` : route du formulaire de déconnexion
- `target: login` : nom de la route vers laquelle sera redirigé le visiteur après sa déconnexion

Exemple

Configurons le routeur général `app/config/routing.yml`

```
login:
    path: /login
    defaults:
        _controller: RootUserBundle:Security:login

login_check:
    path: /login_check

logout:
    path: /logout
```

Exemple

Configurons le routeur général `app/config/routing.yml`

```
login:
    path: /login
    defaults:
        _controller: RootUserBundle:Security:login

login_check:
    path: /login_check

logout:
    path: /logout
```

Ce n'est pas le routeur de notre application

Exemple

Configurons le routeur général `app/config/routing.yml`

```
login:
    path: /login
    defaults:
        _controller: RootUserBundle:Security:login

login_check:
    path: /login_check

logout:
    path: /logout
```

Ce n'est pas le routeur de notre application

Sans les trois routes `login`, `login_check` et `logout` on risque d'avoir une erreur 404.

Exemple

Configurons le routeur général `app/config/routing.yml`

```
login:
    path: /login
    defaults:
        _controller: RootUserBundle:Security:login

login_check:
    path: /login_check

logout:
    path: /logout
```

Ce n'est pas le routeur de notre application

Sans les trois routes `login`, `login_check` et `logout` on risque d'avoir une erreur 404.

`RootUserBundle` est le bundle d'authentification de notre application

Exemple

Créons le bundle `RootUserBundle`

```
php app/console generate:bundle
```

Exemple

Créons le bundle `RootUserBundle`

```
php app/console generate:bundle
```

Supprimons les données (fichiers) par défaut (qu'on n'utilisera pas)

- Le contrôleur par défaut : `DefaultController.php`
- Les tests unitaires : `Tests/Controller`
- Le répertoire de vues : `Resources/views/Default`
- Le routeur du bundle : `Resources/config/routing.yml`
- Dans le routeur général (`app/config/routing.yml`), supprimons :

```
root_user:
    resource: "@RootUserBundle/Resources/config/
     routing.yml"
    prefix:   /
```

Exemple

Préparons le contrôleur `SecurityController.php`

- Si l'utilisateur s'est déjà authentifié, on le redirige vers la page de recherche d'une personne qu'on a réalisée dans un des chapitres précédents (Généralement, on doit le rediriger vers l'accueil ou vers la page qu'il a demandé)
- Le service `security.next` permet de récupérer plusieurs informations sur
  - l'utilisateur
  - les rôles d'un utilisateur
  - ...

```
if ($this->get('security.context')->isGranted('
 IS_AUTHENTICATED_REMEMBERED')){
    return $this->redirectToRoute('
     root_personne_search');
}
```

Exemple

Préparons le contrôleur `SecurityController.php`

3 niveaux de connexion sont possibles

- `IS_AUTHENTICATED_ANONYMOUSLY` : automatiquement affectée à un utilisateur protégé par un pare-feu alors qu'il ne s'est pas encore authentifié. Ceci est seulement possible lorsque l'accès anonyme est autorisé.
- `IS_AUTHENTICATED_REMEMBERED` : automatiquement affectée à un utilisateur connecté via les cookies.
- `IS_AUTHENTICATED_FULLY` : automatiquement affectée à un utilisateur qui a fourni ses informations de connexion au cours de la session en cours.

```
if ($this->get('security.context')->isGranted('
 IS_AUTHENTICATED_REMEMBERED')){
    return $this->redirectToRoute('root_personne_search')
     ;
}
```

Exemple

Préparons le contrôleur `SecurityController.php`

3 niveaux de connexion sont possibles

- Si on a le rôle `IS_AUTHENTICATED_REMEMBERED`, on a également le rôle `IS_AUTHENTICATED_ANONYMOUSLY`.
- Si on a le rôle `IS_AUTHENTICATED_FULLY`, on a également les deux autres rôles.
- En d'autres termes, ces rôles représentent trois niveaux croissants d'authentification.

```
if ($this->get('security.context')->isGranted('
 IS_AUTHENTICATED_REMEMBERED')){
    return $this->redirectToRoute('root_personne_search')
     ;
}
```

Exemple

Préparons le contrôleur `SecurityController.php`

Le service `authentication_utils` (disponible depuis Symfony 2.6) permet de récupérer le nom d'utilisateur et l'erreur dans le cas où le formulaire a déjà été soumis mais était invalide (saisie incorrecte par exemple)

```
$authenticationUtils = $this->get('security.
 authentication_utils');
```

Exemple

Préparons le contrôleur `SecurityController.php`

On redirige l'utilisateur vers la vue d'authentification si

- l'utilisateur doit (ou veut) se connecter
- la saisie précédente est erronée

```
return $this->render('RootUserBundle:Security:login.
 html.twig',array('last_username'=>
 $authenticationUtils->getLastUsername(),'error'=>
 $authenticationUtils->getLastAuthenticationError(),
    ));
  }
}
```

Exemple

Créons le contrôleur `SecurityController.php`

```
namespace Root\UserBundle\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\Security\Core\SecurityContext;
class SecurityController extends Controller{
  public function loginAction()  {
    if ($this->get('security.context')->isGranted('
     IS_AUTHENTICATED_REMEMBERED')){
      return $this->redirectToRoute('root_personne_search');
    }
    $authenticationUtils = $this->get('security.
     authentication_utils');
    return $this->render('RootUserBundle:Security:login.html.
     twig',array('last_username'=>$authenticationUtils->
     getLastUsername(),'error'=>$authenticationUtils->
     getLastAuthenticationError(),
    ));
  }
}
```

Exemple

Créons la vue `login.html.twig`

```
{% block body %}
  {% if error %}
    <div>{{ error.message }}</div>
  {% endif %}
  <form action="{{ path('login_check') }}" method="post">
    <label for="username">Login :</label>
    <input type="text" id="username" name="_username" value="{{
     _last_username }}"/>
    <label for="password">Mot de passe :</label>
    <input type="password" id="password" name="_password" />
    <br/>
    <input type="submit" value="Connexion" />
  </form>
{% endblock %}
```

Exemple

Créons la vue `login.html.twig`

```
{% block body %}
  {% if error %}
    <div>{{ error.message }}</div>
  {% endif %}
  <form action="{{ path('login_check') }}" method="post">
    <label for="username">Login :</label>
    <input type="text" id="username" name="_username" value="{{
     last_username }}"/>
    <label for="password">Mot de passe :</label>
    <input type="password" id="password" name="_password" />
    <br/>
    <input type="submit" value="Connexion" />
  </form>
{% endblock %}
```

Pourquoi un formulaire HTML ?

Tout simplement car notre formulaire n'est pas associé à une `entity`

Exemple

Récupérons l'utilisateur courant depuis le contrôleur

```
$utilisateur = $this->getUser();
if (null === $utilisateur) {
    ...
} else {
    ...
}
```

La méthode `getUser()`

- permet de récupérer l'utilisateur courant s'il est authentifié derrière un pare-feu et non-anonyme
- retourne un objet de type `User`

Exemple

Récupérons l'utilisateur courant depuis le contrôleur

```
$utilisateur = $this->getUser();
if (null === $utilisateur) {
    ...
} else {
    ...
}
```

La méthode `getUser()`

- permet de récupérer l'utilisateur courant s'il est authentifié derrière un pare-feu et non-anonyme
- retourne un objet de type `User`

La classe `User` est disponible dans `/vendor/symfony/symfony/src/Symfony/Component/Security/Core/User`

Exemple

La classe `User.php`

```php
namespace Symfony\Component\Security\Core\User;

/**
 *
 * @author Fabien Potencier <fabien@symfony.com>
 */
final class User implements AdvancedUserInterface{
    private $username;
    private $password;
    private $enabled;
    private $accountNonExpired;
    private $credentialsNonExpired;
    private $accountNonLocked;
    private $roles;
```

Exemple

Récupérons l'utilisateur courant depuis la vue

```
{{ app.user.username }}
```

La variable globale `app`

- `app.user` retourne l'objet `user`
- retourne `null` si l'utilisateur n'est pas authentifié

Exemple

Créons les rôles `security.yml`

```
role_hierarchy:
    ROLE_ADMIN:       [ROLE_SIMPLE, ROLE_ADVANCED]
    ROLE_SUPER_ADMIN: [ROLE_ADMIN,
     ROLE_ALLOWED_TO_SWITCH]
```

La variable globale `app`

- `ROLE_SIMPLE` : permet de créer des objets `Personne` sans les objets `Adresse` et `Sport`
- `ROLE_ADVANCED` : permet de créer des objets `Personne` avec les objets `Adresse` et `Sport`
- `ROLE_ADMIN` : qui hérite les deux rôles précédents

Exemple

Créons les rôles `security.yml`

```
role_hierarchy:
    ROLE_ADMIN:       [ROLE_SIMPLE, ROLE_ADVANCED]
    ROLE_SUPER_ADMIN: [ROLE_ADMIN,
     ROLE_ALLOWED_TO_SWITCH]
```

La variable globale `app`

- `ROLE_SIMPLE` : permet de créer des objets `Personne` sans les objets `Adresse` et `Sport`
- `ROLE_ADVANCED` : permet de créer des objets `Personne` avec les objets `Adresse` et `Sport`
- `ROLE_ADMIN` : qui hérite les deux rôles précédents

Le rôle `USER` n'existe plus, il n'est pas obligatoire

Exemple

Attribuons les rôles

Plusieurs façons pour le faire

- En annotant les méthodes de contrôleurs
- En rajoutant un test sur le type d'utilisateur dans les méthodes de contrôleur
- En rajoutant un test sur le type d'utilisateur dans les vues
- En paramétrant les contrôles d'accès

Exemple

En effectuant un test dans une méthode d'un contrôleur :

```
class PersonneController extends Controller{
  public function addAction(Request $request)  {
    if (!$this->get('security.authorization_checker'
     )->isGranted('ROLE_ADVANCED')) {
      throw new AccessDeniedException('Access␣
       Limited␣to␣ADVANCED');
  ...
  }
...
```

Exemple

En effectuant un test dans une méthode d'un contrôleur :

```
class PersonneController extends Controller{
  public function addAction(Request $request)  {
    if (!$this->get('security.authorization_checker'
     )->isGranted('ROLE_ADVANCED')) {
      throw new AccessDeniedException('Access␣
       Limited␣to␣ADVANCED');
  ...
  }
...
```

N'oublions pas le namespace

```
use Symfony\Component\Security\Core\Exception
\AccessDeniedException;
```

Exemple

En utilisant les annotations :

```
class PersonneController extends Controller{
  /**
   * @Security("has_role('ROLE_ADVANCED') and
    has_role('ROLE_ADMIN')")
   */
  public function addAction(Request $request)  {
  ...
  }
...
```

Exemple

En utilisant les annotations :

```
class PersonneController extends Controller{
  /**
   * @Security("has_role('ROLE_ADVANCED') and
    has_role('ROLE_ADMIN')")
   */
  public function addAction(Request $request)  {
  ...
  }
...
```

N'oublions pas le namespace

```
use Sensio\Bundle\FrameworkExtraBundle
\Configuration\Security;
```

Exemple

En utilisant les annotations :

```
...
{% if is_granted('ROLE_ADVANCED') %}
  <li><a href="{{ path('root_personne_add') }}">
   Ajouter une personne</a></li>
{% endif %}
...
```

Exemple

En utilisant les annotations :

```
...
{% if is_granted('ROLE_ADVANCED') %}
  <li><a href="{{ path('root_personne_add') }}">
   Ajouter une personne</a></li>
{% endif %}
...
```

`is_granted()` est un raccourci de la méthode `isGranted()` du service `security.context`

Exemple

En configurant `access_control` de `security.yml` :

Avec cette méthode

Il faut bien choisir le nom de chaque `path` dans le `routing.yml`

```
access_control:
    - { path: ^/form, roles: ROLE_ADVANCED }
```

Ainsi

On autorise l'accès aux URL commençant par `/form` seulement aux utilisateurs ayant le rôle `ROLE_ADVANCED`

Exemple

En configurant `access_control` de `security.yml` :

Avec cette méthode

Il faut bien choisir le nom de chaque `path` dans le `routing.yml`

```
access_control:
    - { path: ^/form, roles: ROLE_ADVANCED }
```

Ainsi

On autorise l'accès aux URL commençant par `/form` seulement aux utilisateurs ayant le rôle `ROLE_ADVANCED`

On peut aussi décuriser par adresse IP ou par protocole

```
    - { path: ^/form, ip: 127.0.0.1,
     requires_channel: https }
```

Exemple

Créons notre entité `User` dans le bundle `RootUserBundle`

```
php app/console doctrine:generate:entity
```

Les attributs sont :

- `username` de type `string`
- `password` de type `string`
- `salt` de type `string`
- `roles` de type `array`

Notre classe `User` doit

- implémenter de l'interface `UserInterface`
- implémenter la méthode `eraseCredentials()`

Exemple

```
use Symfony\Component\Security\
 Core\User\UserInterface;

class User implements
 UserInterface{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type
      ="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(
      strategy="AUTO")
     */
    private $id;
    /**
     * @var string
     *
     * @ORM\Column(name="username
      ", type="string", length
      =255, unique=true)
     */
    private $username;
    /**
     * @var string
     *
     * @ORM\Column(name="password
      ", type="string", length
      =255)
     */
    private $password;
    /**
     * @var string
     *
     * @ORM\Column(name="salt",
      type="string", length=255)
     */
    private $salt;
    /**
     * @var array
     *
     * @ORM\Column(name="roles",
      type="array")
     */
    private $roles;
```

Exemple

N'oublions pas d'exécuter

```
php app/console doctrine:schema:update
```

Exemple

Utilisons les `DataFixture` pour alimenter la table `USER`

- Les fixtures permettent de remplir la base de données
  - avec des données statiques
  - avec des données de test
- sans avoir besoin de
  - faire de plusieurs `INSERT INTO`
  - créer des interfaces qui permettent l'ajout dans la BD
- Par convention,
  - le nom d'un objet `DataFixture` doit commencer par Load
  - l'objet DataFixture doit implémenter l'interface `FixtureInterface` et écrire la méthode `load()`

Exemple

Mettre en place les fixtures : ajouter les lignes suivantes dans `composer.json`

```
"require": {
  // ...
  "incenteev/composer-parameter-handler": "~2.0",
  "doctrine/doctrine-fixtures-bundle": "~2.3"
},
```

Mettre à jour Composer

```
composer update
```

Exemple

Ajouter les lignes suivantes dans `AppKernel.php`

```
if (in_array($this->getEnvironment(), array('dev', '
 test'))) {
    // ...
    $bundles[] = new Doctrine\Bundle\FixturesBundle\
     DoctrineFixturesBundle();
}
```

Exemple

Ajouter les lignes suivantes dans `AppKernel.php`

```
if (in_array($this->getEnvironment(), array('dev', '
 test'))) {
    // ...
    $bundles[] = new Doctrine\Bundle\FixturesBundle\
     DoctrineFixturesBundle();
}
```

Maintenant

On peut utiliser les `DataFixtures`

Exemple

Préparons nos fixtures `LoadUser.php`

```php
<?php
// src/Root/UserBundle/DataFixtures/ORM/LoadUser.php

namespace Root\UserBundle\DataFixtures\ORM;

use Doctrine\Common\DataFixtures\FixtureInterface;
use Doctrine\Common\Persistence\ObjectManager;
use Root\UserBundle\Entity\User;
class LoadUser implements FixtureInterface{
  public function load(ObjectManager $manager)  {
    $noms = array('Wick', 'Travolta', 'Wolf');
    foreach ($noms as $nom) {
      $user = new User;
      $user->setUsername($nom);
      $user->setPassword($nom);
      $user->setSalt('');
      $user->setRoles(array('ROLE_USER'));
      $manager->persist($user);
    }
    $manager->flush();
  }
}
```

Exemple

Envoyons les données dans la BD

```
php app/console doctrine:fixtures:load --append
```

Exemple

Modifions le `security.yml`

```
encoders:
    Symfony\Component\Security\Core\User\User:
     plaintext
    Root\UserBundle\Entity\User: plaintext
```

On spécifie le nom de l'encodeur.

Exemple

Modifions le `security.yml`

```
providers:
    project:
        entity:
            class:    Root\UserBundle\Entity\User
            property: username
```

On spécifie que le nouveau fournisseur est de type entity et qu'elle prend deux paramètres :

- `class` : nom complet ou raccourci de l'entité
- `property` : l'attribut qui sert d'identifiant

Exemple

Modifions le `security.yml`

```
firewalls:
    project:
        pattern:   ^/
        anonymous: true
        provider:  project
        form_login:
            login_path: login
            check_path: login_check
        logout:
            path:       logout
            target:     login
```

On demande au pare-feu d'utiliser le nouveau fournisseur à la place de `in_memory`

Exemple

Un bundle de gestion d'utilisateurs créé par Symfony : `FOSUserBundle`

- disponible sur `https://github.com/FriendsOfSymfony/FOSUserBundle`
- très utilisé par la communauté Symfony2
- complet et développé par des experts