Contrôle d'accès

Temps de lecture : 4 minutes

Création du contrôleur pour le profil

Créez un nouveau contrôleur :

```
symfony console make:controller Profile
```

L'autorisation avec le composant `Security`

Pour rappel, **l'autorisation est le fait d'autoriser ou non un utilisateur à accéder à une route en fonction de ses rôles**.

Les rôles

Lorsqu'un utilisateur se connecte, il reçoit a minimal le rôle `ROLE_USER`, cela provient du `getter getRoles()` dans le fichier `src/Entity/User.php` :

```php
<?php

public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';

    return array_unique($roles);
}
```

Lorsqu'un utilisateur se connecte, vous pouvez bien sûr lui attribuer d'autres rôles avec le `setter setRoles()`. **Il est nécessaire que tous les rôles commencent par `ROLE_`**.

Une fois que vous avez défini les rôles des utilisateurs, il y a deux moyens de contrôler leurs accès.

La propriété `access_control` du fichier `security.yaml`

La propriété `access_control` du fichier `security.yaml` permet de sécuriser l'accès à des URLs en utilisant des expressions régulières, appelées `patterns`.

Par exemple :

```yaml
security:
  # ...
  access_control:
    - { path: '^/admin', roles: ROLE_ADMIN }
```

Ici, toutes les routes commençant par `/admin` seront protégées et accessibles uniquement aux utilisateurs ayant le rôle `ROLE_ADMIN`.

Vous pouvez bien sûr autorisé une `URL` pour plusieurs rôles en utilisant un tableau :

```yaml
security:
  # ...
  access_control:
    - { path: '^/profile', roles: [ROLE_USER, ROLE_ADMIN] }
```

Si vous définissez plusieurs règles c'est la première (du haut vers le bas) dont le `pattern` correspond à la route qui est appliquée.

Si vous n'utilisez pas de rôles, vous pouvez utiliser `IS_AUTHENTICATED_FULLY` **qui signifie simplement que l'utilisateur est authentifié** :

Par exemple :

```yaml
security:
  # ...
  access_control:
    - { path: '^/profile', roles: IS_AUTHENTICATED_FULLY }
```

Le contrôle d'accès dans les contrôleurs

Une autre option est de contrôler l'accès à une route directement dans le contrôleur.

Vous pouvez utiliser la méthode `denyAccessUnlessGranted()` et lui passer un rôle ou un tableau de rôles :

```php
<?php

namespace App\Controller;
```

```php
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ProfileController extends AbstractController
{
  #[Route('/profile', name: 'profile')]
  public function index(): Response
  {

    $this->denyAccessUnlessGranted('ROLE_USER');

    return $this->render('profile/index.html.twig', [
      'controller_name' => 'ProfileController',
    ]);
  }
}
```

Si l'utilisateur n'a pas le rôle, **il sera redirigé sur la page de connexion s'il n'est pas connecté**. S'il est connecté mais n'a pas le ou les rôles requis, il aura une erreur `403 Forbidden`.

Plus besoin de `sensio/framework-extra-bundle` à partir de la version 6.2 de Symfony.

Vous pouvez ainsi faire directement :

```php
<?php

namespace App\Controller;

use Symfony\Component\Security\Http\Attribute\IsGranted;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ProfileController extends AbstractController
{
  #[Route('/profile', name: 'profile')]
  #[IsGranted('ROLE_USER')]
  public function index(): Response
  {
    return $this->render('profile/index.html.twig', [
```

```
      'controller_name' => 'ProfileController',
    ]);
  }
}
```

Comme pour les routes, vous pouvez utiliser l'attribut sur la classe du contrôleur pour l'appliquer à toutes les méthodes du contrôleur :

```php
<?php

namespace App\Controller;

use Symfony\Component\Security\Http\Attribute\IsGranted;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

#[IsGranted('ROLE_USER')]
class ProfileController extends AbstractController
{
  #[Route('/profile', name: 'profile')]
  public function index(): Response
  {
    return $this->render('profile/index.html.twig', [
      'controller_name' => 'ProfileController',
    ]);
  }
}
```

Si vous ne souhaitez pas rediriger l'utilisateur mais exécuter du code en fonction du rôle de l'utilisateur, il faut utiliser la méthode `isGranted()` :

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ProfileController extends AbstractController
{
  #[Route('/profile', name: 'profile')]
```

```
  public function index(): Response
  {
    if ($this->isGranted('ROLE_ADMIN')) {
      // faire quelque chose
    } else {
      // faire autre chose
    }

    return $this->render('profile/index.html.twig', [
      'controller_name' => 'ProfileController',
    ]);
  }
}
```

Récupérer l'utilisateur connecté

Il est extrêmement simple de récupérer l'utilisateur connecté avec le composant `Security`.

Depuis un contrôleur

Il suffit d'utiliser le `getter getUser()` :

```
<?php

namespace App\Controller;

use Symfony\Component\Security\Http\Attribute\IsGranted;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

#[IsGranted('ROLE_USER')]
class ProfileController extends AbstractController
{
  #[Route('/profile', name: 'profile')]
  public function index(): Response
  {

    $user = $this->getUser();

    return $this->render('profile/index.html.twig', [
      'controller_name' => 'ProfileController',
    ]);
```

```
    }
}
```

Depuis un `template`

Il suffit d'utiliser **la variable** `app.user` .

Vous devez vérifier que l'utilisateur est authentifié soit dans le contrôleur soit directement dans le `template` sinon vous aurez une erreur si la variable n'est pas définie.

```
{% if is_granted('ROLE_USER') %}
  <p>Email : {{ app.user.email }}</p>
{% endif %}
```

Depuis un service

Il suffit d'injecter le service `Security` **puis d'utiliser la méthode** `getUser()` :

```
<?php

// ...

use Symfony\Bundle\SecurityBundle\Security;

class UnService
{
  public function __construct(private Security $security)
  {}

  public function uneMethode()
  {
    $user = $this->security->getUser();
  }
}
```