Les voters

Pour l'exemple, nous allons mettre en place un voter empêchant l'accès à la modification des informations utilisateur d'un autre profil, potentiellement accessible par l'url.

Création et configuration du voter

Nous allons créer un fichier `UserVoter.php` dans le dossier `src\Security`.

Nous allons créer dans ce fichier une classe `UserVoter` que nous allons étendre à la classe `Voter` (n'oubliez pas de faire l'import s'il ne se fait pas automatiquement).

Cette classe contiendra obligatoirement 2 méthodes :

- la méthode `support()` qui va déterminer selon quelles actions le voter sera appelé.
- la méthode `voteOnAttribute()` va définir quel utilisateur pourra effectuer cette action.

```
<?php
namespace App\Security;

use App\Entity\User;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
use Symfony\Component\Security\Core\Authorization\Voter\Voter;

class UserVoter extends Voter {

    protected function supports($attribute, $subject)
    {
        // TODO: Implement supports() method.
    }

    protected function voteOnAttribute($attribute, $subject, TokenInterface
$token)
    {
        // TODO: Implement voteOnAttribute() method.
    }
}
```

support()

Cette méthode va prendre 2 paramètres :

- `$attribut` qui est l'action définie
- `$user`, peut-être variable, on l'utilise ici car le contrôle se fait sur la modification d'un utilisateur. Nous aurions utilisé un autre paramètre si la modification se serait faite ailleurs ($products pour une modification sur la table products par exemple)

```
// définition d'un constante contenant la/les action(s) à surveiller
 const EDIT = 'edit';
```

```
protected function supports($attribute, $subject)
    {
        // l'attribut n'est pas dans la liste
        if (!in_array($attribute, [self::EDIT])) {
            return false;
        }
        // si $user n'est pas une instance de User => pas dans la liste des
utilisateur
        if (!$subject instanceof User) {
            return false;
        }
        // si retourne true, appel de voteOnAttribute()
        return true;
    }
```

Si la fonction `supports()` retourne `true`, on a donc bien un utilisateur connecté, qui cherche à avoir accès à l'édition du profil. La fonction `voteOnAttribute()` est ensuite exécutée.

voteOnAttribute()

Dans cette fonction, nous allons d'abord récupérer l'utilisateur courant :

```
$user = $token->getUser();
```

Nous vérifions si l'utilisateur passé en paramètre de la fonction est bien une instance de la classe `User` :

```
if (!$user instanceof User) {
            // l'utilisateur doit être connecté, sinon accès refusé
            return false;
        }
```

Grâce à la méthode `supports()` nous savons que `$subject` est un objet de la classe `User`, nous le stockons dans une variable :

```
$utilisateur = $subject;
```

enfin nous étudions les différents cas définis (edit pour l'exemple, il peut y en avoir plus selon la gestion des autorisations), et pour chaque cas, on va vérifier que l'utilisateur connecté est bien celui qui est attendu :

```
switch ($attribute) {
       case self::EDIT:
          return $user->getId() == $utilisateur->getId();
}

throw new \LogicException('This code should not be reached!');
```

Appel du voter dans le contrôleur

Reprenons notre contrôleur, la fonction `edit()` :

```
public function edit(Request $request, User $user): Response
    {

        $form = $this->createForm(UserType::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $this->getDoctrine()->getManager()->flush();

            return $this->redirectToRoute('user_index');
        }

        return $this->render('user/edit.html.twig', [
            'user' => $user,
            'form' => $form->createView(),
        ]);
    }
```

Nous allons y ajouter la commande suivante :

```
    $this->denyAccessUnlessGranted('edit', $user, 'non non non ...');
```

Cette commande va permettre de faire appel au voter précédemment établi.

On y passe en paramètre un 'attribut' (ici `edit`, l'action qui est limitée), l'utilisateur qui veut avoir accès à cette action (`$user` dans notre cas), et enfin le message qui va être affiché en cas de refus d'accès à l'action demander.

Ce message sera affiché sous forme d'exception en dev :

Mettez en application l'exemple ci-dessus pour éviter qu'un utilisateur ne puisse pas avoir accès à la modification d'un profil autre que le sien.

Profitez-en personnaliser la page d'erreur en utilisant la documentation.

Pour repérer le code d'erreur, pensez à regarder la console de debug.