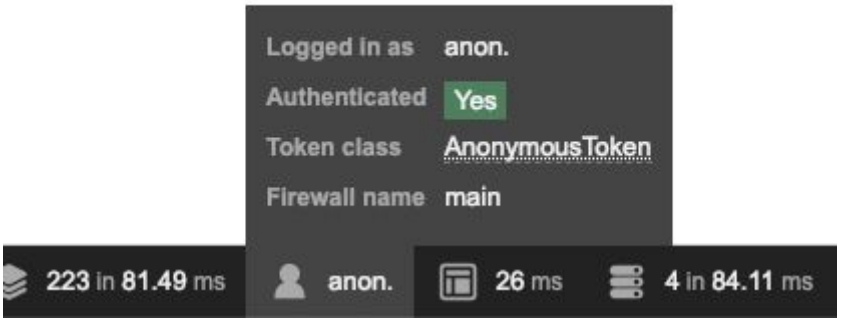
Introduction

Dans Symfony il y a 2 concepts importants à saisir lorsqu'on parle de sécurité : **L'authentification** et **l'autorisation (contrôle d'accès)**. Être authentifié signifie "**savoir qui on est**", être autorisé signifie "**avoir le droit d'accès**".

L'authentification aura lieu avant le contrôle d'accès, et il faut noter qu'un utilisateur anonyme est déjà authentifié comme on peut le voir dans la toolbar :



app/config/security.yml

La configuration de la sécurité se passe dans le fichier **app/config/security.yml**.

On va retrouver dans ce fichier 5 éléments importants:

firewalls permet de configurer des pare-feux. Par défaut il y a un pare-feu nommé **dev** qui est là pour permettre l'accès à certaines ressources en environnement de dev. On va en général créé un deuxième pare-feu nommé **main** qui sera utilisé pour toute notre application.

```
    firewalls:
        # disables authentication for assets and the profiler, adapt it
according to your needs
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        # Private area
        main:
            access_denied_url: /private/accessdenied
            pattern:   ^/
            anonymous: true
            logout_on_user_change: true
```

```
            provider:  user_provider
            form_login:
                check_path: login
                login_path: login
                default_target_path: dashboard
            logout:
                path:   logout
                target: /
            anonymous:    true
```

providers permet de configurer différents type de fournisseur d'utilisateurs qui seront utilisé par le pare-feu. Un utilisateur peut-être défini directement dans le fichier security, provenir d'une base de donnée, d'un service externe (Facebook), etc.

```
#
https://symfony.com/doc/current/security.html#b-configuring-how-users-ar
e-loaded
    providers:
        user_provider:
          entity:
            class: AppBundle:User
            property: email
```

role_hierarchy permet de définir la hiérarchie / l'héritage des rôles. Un rôle doit être préfixé par **ROLE_**
Les rôles vont être utilisés par le contrôleur d'accès pour donner le droit d'accéder ou non à une ressource.

```
    role_hierarchy:
        ROLE_USER_PENDING: [ROLE_USER_PENDING]
        ROLE_USER: [ROLE_USER]
        ROLE_ADMIN: [ROLE_USER, ROLE_ADMIN]
```

encoders permet de définir différentes manières d'encoder les mots de passes d'un utilisateur. Ca peut être en clair (non recommendé), hashé, etc.

```
    encoders:
        AppBundle\Entity\User:
            algorithm: bcrypt
```

access_control permet de contrôler d'un manière globale l'accès aux ressources de l'application. On va pouvoir définir quels rôles ou adresses ips peuvent accéder à une URL

donnée, tout en précisant si besoin le type de protocol requis (http, https). Il est également possible de gérer le contrôle d'accès depuis le code PHP / dans un template Twig, grâce aux services **authorization_checker** ou **security**.

```
    access_control:
    # require ROLE_ADMIN for /private/administration
    - { path: ^/private/administration, roles: [ROLE_ADMIN] }
    # require ROLE_USER or ROLE_USER_PENDING for /private/denied
    - { path: ^/private/accessdenied, roles: [ROLE_USER,
ROLE_USER_PENDING] }
    # require ROLE_USER for /private
    - { path: ^/private, roles: ROLE_USER }
    # no auth needed for /
    - { path: ^/, role: IS_AUTHENTICATED_ANONYMOUSLY }
```

Exemple connexion http basique

```
# To get started with security, check out the documentation:
# https://symfony.com/doc/current/security.html
security:

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

  role_hierarchy:
    ROLE_ADMIN:               [ROLE_USER]

  #
https://symfony.com/doc/current/security.html#b-configuring-how-users-ar
e-loaded
  providers:
    in_memory:
      memory:
        users:
          utilisateur_lambda:
            password: user
            roles: 'ROLE_USER'
          utilisateur_admin:
            password: admin
            roles: 'ROLE_ADMIN'

  firewalls:
    # disables authentication for assets and the profiler, adapt it
according to your needs
```

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

    main:
      anonymous: ~
      # activate different ways to authenticate

      #
https://symfony.com/doc/current/security.html#a-configuring-how-your-use
rs-will-authenticate
      http_basic:
        provider: in_memory

      # https://symfony.com/doc/current/security/form_login_setup.html
      #form_login: ~

  access_control:
    - { path: ^/users, roles: ROLE_USER }
    - { path: ^/admin, roles: ROLE_ADMIN }
```

Contrôle d'accès en dehors du fichier security.yml

Annotations sur un contrôleur

Dans ce cas c'est le service **authorization_checker** qui est utilisé

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

/**
 * @Security("has_role('ROLE_ADMIN')")
 */
public function helloAction($name)
{
    // ...
}
```

Sans annotations

```
// Vérifier que l'utilisateur est authentifié. Attention,
IS_AUTHENTICATED_FULLY n'est pas un rôle
if
(!$this->get('security.authorization_checker')->isGranted('IS_AUTHENTICA
```

```
TED_FULLY')) {
    throw $this->createAccessDeniedException();
}
// via un helper :
$this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');
```

Dans un service grâce au service security

```
// src/AppBundle/Service/MyService.php

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

class MyService
{
    protected $security;

    public function __construct(Security $security)
    {
        $this->security = $security;
    }

    public function doSomething()
    {
        if (!$this->security->isGranted('ROLE_USER_COMPLETE')) {
            throw new AccessDeniedException();
        }

        // ...
    }

}
```

Dans Twig :

```
{% if is_granted('ROLE_ADMIN') %}
    <a href="...">Delete</a>
{% endif %}
```

Récupérer l’objet utilisateur

Dans un contrôleur

```
$this->getUser(); // Retourne null si anonyme
```

Dans un service

```
class ContactService
{

    protected $user;

    public function __construct(TokenStorageInterface $tokenStorage)
    {
        $this->user = $tokenStorage->getToken()->getUser();
    }

    // ...
}
```

Dans Twig

Depuis **app.user**

```
{# afficher le username de l'utilisateur #}
{% if is_granted('IS_AUTHENTICATED_FULLY') %}
    <p>Username: {{ app.user.username }}</p>
{% endif %}
```