Symfony Sécurité

AYMEN SELLAOUTI

Introduction

- Un site est généralement décomposé en deux parties :
  - Partie public : accessible à tous le monde
  - Partie privée : accessible à des utilisateurs particuliers.
    - Au sein même de la partie privée, certaines ressources sont spécifiques à des rôles ou des utilisateurs particuliers.

Nous identifions donc deux niveaux de sécurité :

Authentification

Autorisation

Authentification

- Processus permettant d'authentifier un utilisateur.
- Deux réponses possibles
  - Non authentifié : Anonyme.
  - Authentifié : membre

Security Bundle

Le Bundle qui gère la sécurité dans Symfony s'appèle SécurityBundle.

Si vous ne l'avez pas dans votre application, installer le via la commande :

```
composer require symfony/security-bundle
```

Fichier de configuration security.yml

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

    # https://symfony.com/doc/current/security.html#b-configuring-how-users-are-loaded
    providers:
        in_memory:
            memory: ~

    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: ~
```

Le Firewall qui gère la configuration de l'authentification de vos utilisateurs

Cette partie assure que le débuggeur de Symfony ne soit pas bloqué

La classe user

L'ensemble du système de sécurité est basé sur la classe user qui représente l'utilisateur de votre application.

Afin de créer la classe user utiliser la commande :

php bin/console make:user (si vous n'avez pas le MakerBundle, installer le).

Cette outils vous posera un ensemble de questions, selon votre besoin répondez y et il fera tout le reste.

UserProvider

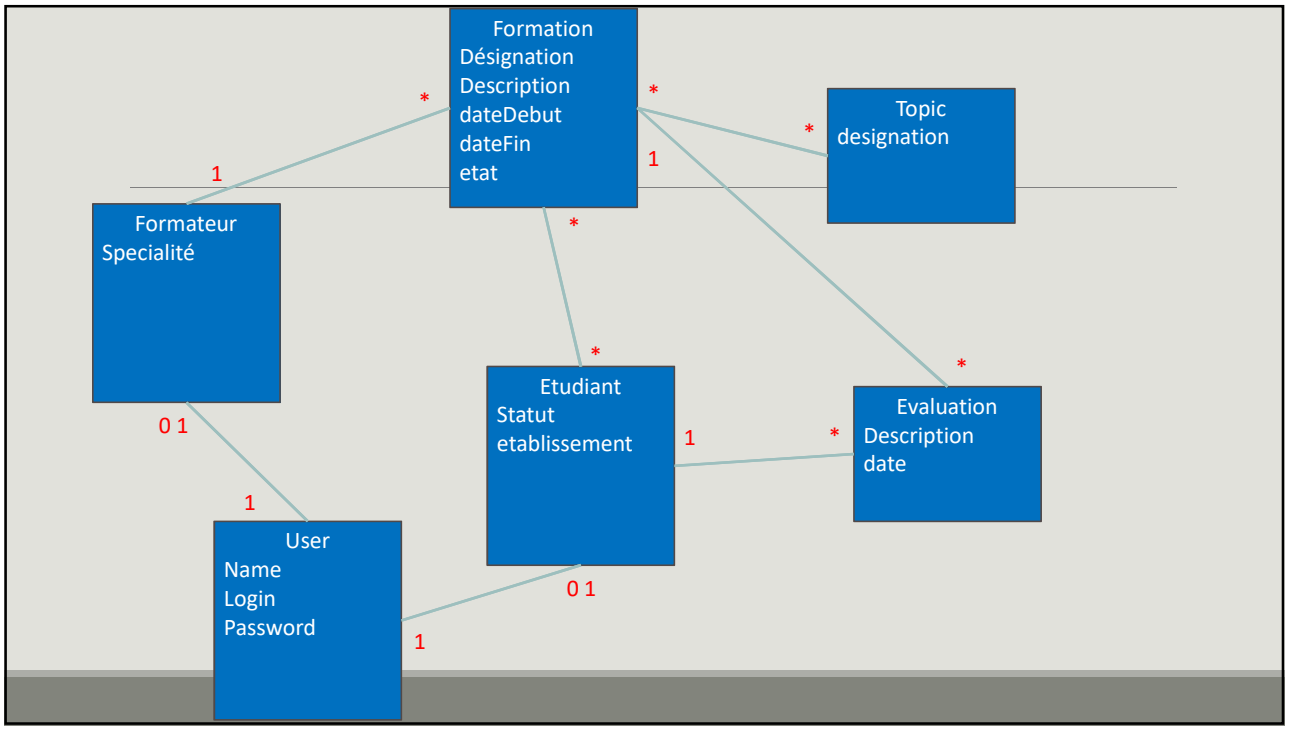
Le User Provider est un ensemble de classe associé au bundle Security de Symfony et qui ont deux rôles

- Récupérer le user de la session. En effet, pour chaque requête, Symfony charge l'objet user de la session. Il vérifie aussi que le user n'a pas changé au niveau de la BD.
- Charge l'utilisateur pour réaliser certaines fonctionnalités comme la fonctionnalité ***se souvenir de moi***.

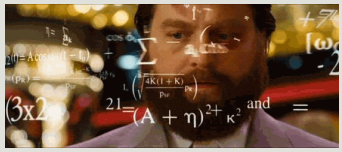
Afin de définir le userProvider que vous voulez utiliser passer par le fichier de configuration security.yaml

```yaml
providers:
    users:
        entity:
            # the class of the entity that represents users
            class: 'App\Entity\User'
            # the property to query by - e.g. username, email, etc
            property: email
```

https://symfony.com/doc/current/security/user_provider.html



Exercice



Reprenez votre diagramme de classe.

Ajouter la classe user.

Ajouter les relations nécessaires.

Authentification et Firewall

Le système d'authentification de Symfony est configuré au niveau de la partie **firewalls** de votre fichier security.yaml.

Cette section va définir comment vos utilisateurs seront authentifié.

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

Authentification et Firewall

- Comme décrite dans la documentation, l'authentification dans Symfony ressemble à de « la magie ».
- En effet, au lieu d'aller construire une route et un contrôleur afin d'effectuer le traitement, vous devez simplement activer un « authetication provider ».
- « L'authentication provider » est du code qui s'exécute automatiquement avant chaque appel d'un contrôleur.
- Symfony possède un ensemble d' « authetication provider » prêt à l'emploi. Vous trouverez leur description dans la documentation : https://symfony.com/doc/current/security/auth_providers.html
- Dans la documentation, il est conseillé de passer par les « Guard Authenticator » qui permettent un contrôle total sur toutes les parties de l'authentification.

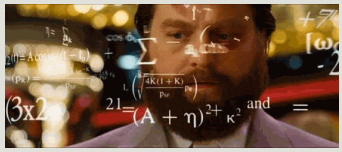
Encoder le mot de passe

Le service responsable de l'encodage des mots de passe et le service UserProviderEncoder.

Afin de l'utiliser et comme tous les services de Symfony, il suffit de l'injecter.

```
private $userPasswordEncoder;
   public function __construct(
      UserPasswordEncoderInterface $userPasswordEncoder
      )
      {
            $this->userPasswordEncoder =$userPasswordEncoder;
      }
```

Exercice



Créer un fixture qui permet de créer quelques utilisateurs.

Ne lancer que les fixture du user. Pour se faire, les fixture que vous voulez lancer doivent implémenter l'interface FixtureGroupeInterface. Ceci permettra de lancer les fixtures d'un groupe donnée.

En implémentant cette interface, vous devez implémenter la méthpde getGroupes. Cette méthode retourne un tableau contenant le nom des groupes auxquels appartient cette fixture.

Exemple : return ['groupeRedondant','groupeTest1'];

Enfin lancer la commande de chargement de fixture avec l'option –group=nomGroupe ajouter aussi l'option –append pour ne pas purger la base de données.

https://symfony.com/doc/master/bundles/DoctrineFixturesBundle/index.html

Les Guard Authenticator

Un « Guard authenticator » est une classe qui vous permet un control complet sur le processus d'authentification.

Cette classe devra ou implémenter l'interface AuthentiatorInterface ou étendre la classe abstraire associé au besoin (AbstractFormLoginAuthenticator en cas de formulaire d'authentification ou AbstractGuardAuthenticator en cas d'api)

La commande make:auth permet de générer automatiquement cette classe.

Une fois lancée, cette commande vous demande si vous voulez créer un « empty authenticator » ou un « login form authenticator ».

Guard Authenticator

Chaque guard devra implémenter les méthodes suivantes :

supports(Request $request)
Cette méthode est la première à être appelé. Elle sera appelé à chaque requête et votre travail consiste à décider si l'authentificateur doit être utilisé pour cette requête (return true) ou s'il doit être ignoré (return false).

getCredentials(Request $request)
Elle sera appelé à chaque requête et votre travail consiste à lire le Token (ou quelle que soit votre information "d'authentification") à partir de la demande et de les renvoyer. Ces informations d'identification sont ensuite transmises en tant que premier argument de getUser().

getUser($credentials, UserProviderInterface $userProvider)
$credentials représente la valeur retourné par getCredentials(). Votre rôle consiste donc à retourner un objet qui implémente la UserInterface qui sera donc un objet de votre classe User. Une fois retourné, la méthode checkCredentials() sera appelé. Si vous retourner null (ou throw an AuthenticationException) l'authentification sera avorté.

Guard Authenticator

checkCredentials($credentials, UserInterface $user)
Si getUser() retourne un objet User, cette méthode est appelé. Votre travail est de vérifier si les credentials sont correct. Pour un login form, C'est l'endroit ou vous allez vérifier si le mot de passe est correct. Pour passer l'authentification vous devez retourner true. Si vous retourner autre chose (ou throw an AuthenticationException), l'authentification va échouer.

onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)
Appelé après une authentification réussie. Votre travail est de retourner un objet Response qui sera envoyé au client ou null pour continuer la requête.

onAuthenticationFailure(Request $request, AuthenticationException $exception)
Appelé si une authentification échoue. Votre travail est de retourner un objet Response object qui sera envoyé au client. L'exception vous informera sur la cause de l'échec de l'authentification.

Activer le guard

L'activation d'un Guard se fait au niveau du fichier security sous la clé **firewalls** en utilisant la clé **guard**.

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        anonymous: true
        guard:
            authenticators:
                - App\Security\FormLoginAuthenticator
```

Se déconnecter

Afin de se déconnecter, il suffit d'ajouter la clé logout dans votre firewalls configuration dans security.yaml.

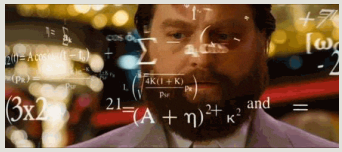
Ajouter ensuite une méthode vide logout dans votre securityController

```
/**
 * @Route("/logout", name="logout")
 */
public function logout() {

}
```

```
firewalls:
    main:
        logout:
            path: logout
```

Exercice



Créer un LoginForm en utilisant la commande php bin/console make:auth.

Terminer les étapes Next définies par la commande à la fin de son exécution.

Récupérer le user dans le contrôleur

Afin de récupérer le user dans un contrôleur, il suffit d'utiliser la méthode helper **getUser**.

Utiliser ensuite sa méthode

```php
public function list()
{
    $user = $this->getUser();
}
```

Récupérer le user dans le service

Afin de récupérer le user dans un service, il suffit d'injecter le Securiy Service.

Utiliser ensuite sa méthode **getUser**

```php
use Symfony\Component\Security\Core\Security;
class HelperService
{
    private $security;
    public function __construct(Security $security)
    {
        $this->security = $security;
    }
    public function sendMoney() {
        $user = $this->security->getUser()
    }
}
```

Remember Me

Afin d'activer la fonctionnalité 'se souvenir de moi' ajouter la configuration suivante dans le fichier security.yml :

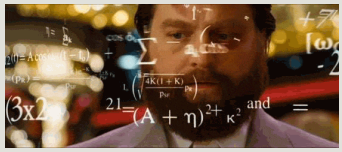
```yaml
main:
    anonymous: true
    remember_me:
        secret:   '%kernel.secret%'
        lifetime: 604800 # 1 week in seconds
```

Décommenter le code affichant le bouton remeber me dans la Twig login.html.twig

```twig
<div class="checkbox mb-3">
    <label>
        <input type="checkbox" name="_remember_me"> Remember me
    </label>
</div>
```

https://symfony.com/doc/current/security/remember_me.html

Exercice



Fait en sorte que le lien login de votre template vous envoi vers la page de login.

Ajouter un lien pour le logout.

Autorisation

- Processus permettant d'autoriser un utilisateur à accéder à une ressource selon son rôle.

Le processus d'authentification suit deux étape.

1- Lors de l'authentification l'utilisateur est associé à un ensemble de rôles.

2- Lors de l'accès à une ressource, on vérifie si l'utilisateur a le rôle nécessaire pour y accéder.

Les Rôles

Chaque utilisateur connecté a au moins un role : le ROLE_USER

Tous les rôles commencent par ROLE_

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

    return array_unique($roles);
}
```

Définir les droits d'accès

Les droits d'accès sont définit de deux façons :

1- Dans le fichier security.yaml

2- Directement dans la ressource

Sécuriser les patrons d'url

La méthode la plus basique pour sécuriser une partie de votre application est de sécuriser un patron d'url complet dans votre fichier security.yaml.

Ceci se fait sous la clé access_control. Chaque entrée est un objet avec comme clé :

-**path** : le pattern à protéger

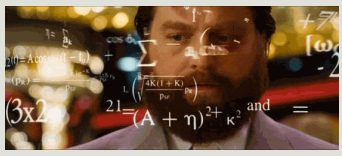
-**roles** : les rôles qui peuvent accéder à ce pattern.

Lorsque vous essayer d'accéder à une ressource, Symfony cherche dans cette rubrique s'il y a un matching avec la route recherché de haut vers le bas. Le premier qu'il trouve lui permet de vérifier si vous avez le bon rôle pour accéder à la ressource demandé ou non. **L'ordre donc est très important**.

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

https://symfony.com/doc/current/security/access_control.html

Exercice



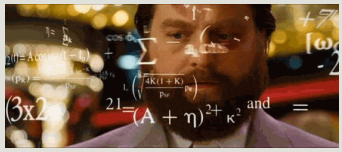
Créer une action avec la route '/admin'. Décommentre les **access_control** et essayer deux scénarios.

1. Connecter vous en tant que USER et essayer d'accéder à cette route. Que se passe t-il ?
2. Déconnecter vous et essayer d'y accéder. Que se passe t-il ?

Modifier votre classe UserFixture et faite en sorte d'ajouter un user avec le ROLE_ADMIN. N'effacer rien de votre base.

Connecter vous en tant qu'admin. Essayer d'accéder à la route /admin. Que se passe t-il ?

Exercice



Créer une action permettant d'inscrire des sutilisateurs.

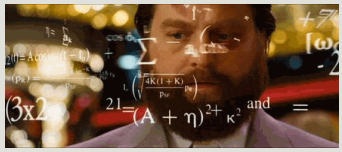
Créer une action pour l'admin lui permettant d'ajouter des utilisateurs avec le ROLE qu'il veut.

Authentification manuelle d'un utilisateur

Dans certains uses-cases vous devez connecter automatiquement un utilisateur. Dans ce cas, vous devez utiliser un **Authentcator** et un service appelé **GuardAuthenticatorHandler** et sa méthode **authenticateUserAndHandleSuccess**.

```php
/**
 * @param LoginFormAuthenticator $authenticator
 * @param GuardAuthenticatorHandler $guardHandler
 * @param Request $request
 * @return mixed
 * @Route("/register", name="register")
 */
public function register(
        LoginFormAuthenticator $authenticator, GuardAuthenticatorHandler $guardHandler, Request $request
){
    // Todo - Récupérer vos données, ajouter le user $user
    // Authentifier votre user
    return $guardHandler->authenticateUserAndHandleSuccess(
    $user,          // L'objet user que vous avez créé
    $request,       //La requete reçu
    $authenticator, // L'authenticator qui contient la méthode onAuthenticationSuccess que vous voulez utiliser
        'main'          // Le nom de votre firewall dans security.yaml
    );}
```

Exercice



Mettez à jour la fonction d'inscription afin d'automatiquement authentifier le user.

Sécuriser Les contrôleurs

Vous pouvez directement sécuriser vos contrôleurs en utilisant :

1- Le helper **denyAccessUnlessGranted**('ROLE_*');

2- En utilisant l'annotation @IsGranted('ROLE_*')

```php
/**
 * Matches /blog exactly
 *
 * @IsGranted("ROLE_ADMIN")
 * @Route("/blog", name="blog_list")
 */
public function list()
{
    // ...
}
```

```php
/**
 * Matches /blog exactly
 *
 * @Route("/blog", name="blog_list")
 */
public function list()
{
    $this->denyAccessUnlessGranted("ROLE_USER");
    // ...
}
```

Sécuriser un service

Afin de sécuriser un service, il suffit d'injecter le Securiy Service.

Utiliser ensuite sa méthode **isGranted**

```php
use Symfony\Component\Security\Core\Security;
class HelperService
{
    private $security;
    public function __construct(Security $security)
    {
        $this->security = $security;
    }
    public function sendMoney() {
        if ($this->security->isGranted("ROLE_ADMIN")){
          // Todo Send Money
        }
    }
}
```

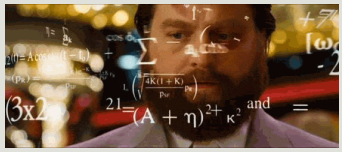
https://symfony.com/doc/current/security/securing_services.html

Sécuriser vos pages TWIG

Si vous voulez vérifier le role de l'utilisateur avant d'afficher une ressource ou des informations dans vos pages TWIG, utiliser la méthode **is_granted**('ROLE_*')

```twig
{% if is_granted('ROLE_ADMIN') %}
    <a href=« /admin">Administration</a>
{% endif %}
```

Exercice



Faite en sorte que le menu s'adapte au rôle de l'utilisateur connecté.

Hiérarchie de rôles

Vous pouvez définir une hiérarchie de rôles.

Dans le fichier **security.yaml** et sous la clé **role_hierarchy**, définissez le **rôle principale** suivie de **l'ensemble des rôles dont il hérite**.

Un use case très récurant est quand l'admin possédé tout les droits, donc l'admin devra hériter de tous les rôles.

```
role_hierarchy:
    ROLE_COMMERCIAL:        ROLE_AGENT
    ROLE_SECRETARY:       ROLE_COMMERCIAL
    ROLE_ADMIN:       [ROLE_PARTNER, ROLE_SECRETARY]
```

Ici un commercial a les accès de l'Agent.

L'admin peut avoir les accès du Partner et de la secrétaire.

Spécial Rôles

IS_AUTHENTICATED_ANONYMOSLY : Un utilisateur non authentifié

IS_AUTHENTICATED_REMEMBRED : Vérifie qu'un utilisateur est authentifié indépendamment de son ROLE.

IS_AUTHENTICATED_FULLY : Vérifie qu'un utilisateur est authentifié indépendamment de son ROLE. Cependant si le user est authentifié à cause de la fonctionnalité 'remember_me' alors il n'est pas authenticated fully.