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'appelle SecurityBundle.
- Si vous ne l'avez pas dans votre application, installer le via la commande

composer require security

Fichier de configuration security.yml

```yaml
security:
    # https://symfony.com/doc/current/security/authenticator_manager.html
    enable_authenticator_manager: true
    # https://symfony.com/doc/current/security.html#c-hashing-passwords
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
    providers:
        users_in_memory: { memory: null }
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: users_in_memory
            # switch_user: true
    # Easy way to control access for large sections of your site
    # Note: Only the *first* access control that matches will be used
    access_control:
    # - { path: ^/admin, roles: ROLE_ADMIN }
```

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 :

symfony 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***.

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

UserProvider

- 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
```

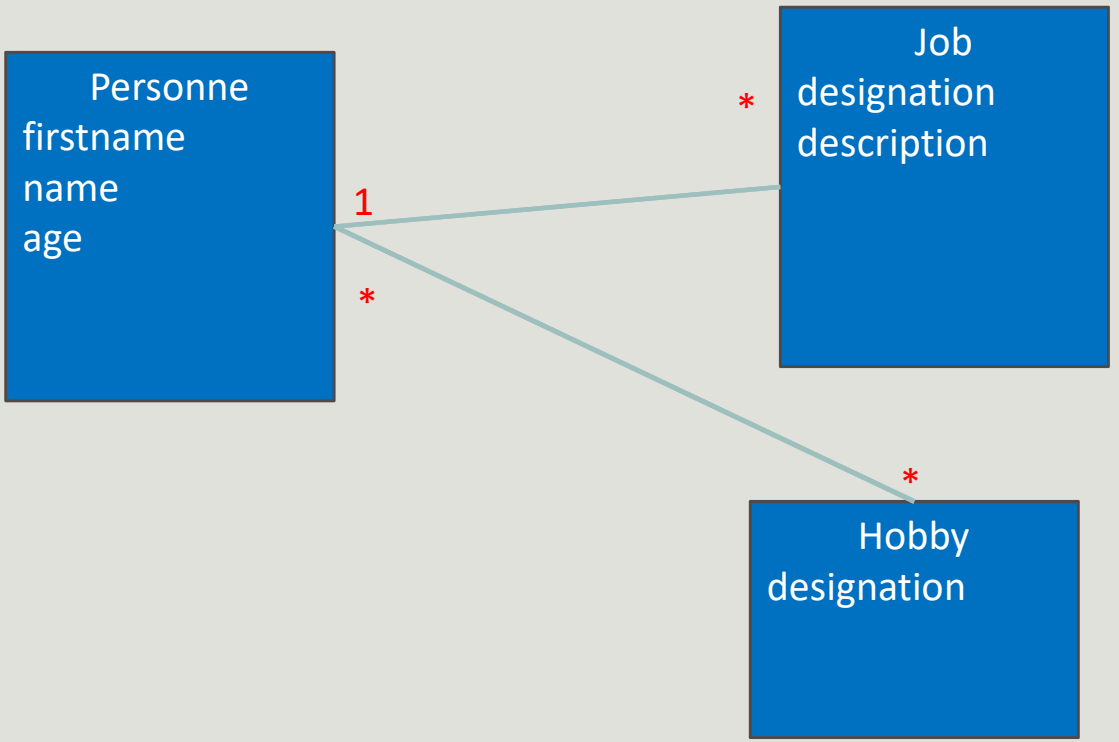
https://symfony.com/doc/current/security/user_provider.html

Exercice

Reprenez votre diagramme de classe.

Ajouter la classe user.

Ajouter les relations nécessaires.



Encoder le mot de passe

- Vous n'avez pas toujours besoin de mot de passe
- En cas de besoin, vous pouvez configurer la manière avec lequel votre mot de passe doit être géré dans le fichier security.yml.

```yaml
# config/packages/security.yaml
security:
    # ...
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
        # use your user class name here
        App\Entity\User:
# Use native password hasher, which auto-selects the best
# possible hashing algorithm (starting from Symfony 5.3 this is "bcrypt")
            algorithm: auto
```

A partir de Symfony 5,3

```yaml
security:
    encoders:
        App\Entity\User:
            algorithm: auto
```

Avant Symfony 5,3

Encoder le mot de passe

- Le service responsable de l'encodage des mots de passe est le service UserProviderEncoder (avant Symfony 5.3) ou UserPasswordHasher à partir de Symfony 5.3 .
- Afin de l'utiliser, et comme tous les services de Symfony, il suffit de l'injecter.

```
private $userPasswordHasher;
    public function __construct( UserPasswordHasherInterface $userPasswordHasher)
        {
                $this->userPasswordHasher = $userPasswordHasher;
        }
```

```
public function __construct( private UserPasswordHasherInterface  $passwordEncoder)
{
}
```

Exercice

- Créer un fixture qui permet de créer quelques utilisateurs.
- Ne lancer que les fixtures du user. Pour se faire, les fixtures 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 méthode 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

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, e.g. API Token, Formulaire d'authentification.

```
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 « **authentication provider** ».
- « L'authentication provider » est du code qui s'exécute **automatiquement avant chaque appel d'un contrôleur**.
- Symfony possède un ensemble d' « authentication 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.

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 abstraite associée au besoin, e.g. 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
Symfony 5.3

- A partir de la version 5.3, vous devez implémenter uniquement les méthodes **authenticate** et la méthode **onAuthenticationSuccess.**
- Il y a aussi des méthodes optionnelles que vous pouvez surcharger :
  - **supports**
  - **onAuthenticationFailure**
  - **start**
- Symfony utilises à partir de la version 5,3 **un nouveau Authenticator based Security**
- **Pour la gestion des utilisateurs elle utilises Passport**

Guard Authenticator

Symfony 5.3

```php
public function authenticate(Request $request): PassportInterface
{
    $username = $request->request->get('username', '');
    $request->getSession()->set(Security::LAST_USERNAME, $username);
    return new Passport(
        new UserBadge($username),
        new PasswordCredentials($request->request->get('password', '')),
        [
            new CsrfTokenBadge('authenticate', $request->get('_csrf_token')),
        ]
    );
}
```

```php
public function onAuthenticationSuccess(Request $request, TokenInterface $token,
string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }
throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
}
```

Activer le guard (Symfony 5.3)

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    appLogin:
        pattern: ^/login
        custom_authenticator: App\Security\LoginFormAuthenticator
        logout:
            path: app_logout
            # where to redirect after logout
            # target: app_any_route
```

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 avec la route associé à votre méthode logout.**
- Vous pouvez débuger les autres options de logout avec la commande

symfony console debug:config security

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

}
```

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

Exercice

- Créer un LoginForm en utilisant la commande

  symfony console make:auth.

- Terminer les étapes 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

```
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()
    }
}
```

Exercice

- Fait en sorte que le lien login de votre template vous envoi vers la page de login.
- Ajouter un lien pour le logout.

Register

- Afin de permettre l'ajout ou l'enregistrement de vos utilisateurs, vous pouvez utiliser la commande :

symfony console make:registration-form

- Cette fonctionnalité n'a rien de particulier, elle permet tout simplement d'ajouter un utilisateur dans votre base de données.
- Vous pouvez personnaliser le contrôleur généré comme vous le voulez.

Authentification manuelle d'un utilisateur

- Une fois l'utilisateur inscrit, vous pourrez l'authentifier d'une façon manuelle en injectant le service **UserAuthenticatorInterface**.
- Utiliser sa méthode authenticateUser qui prend en paramètre le user, votre authenticator et l'objet request

```
public function register(Request $request, UserPasswordHasherInterface
$userPasswordHasherInterface, LoginFormAuthenticator $authenticator,
UserAuthenticatorInterface  $userAuthenticator): Response
{
    //…
    if ($form->isSubmitted() && $form->isValid()) {
        //…
        // Authenticate user
       // retourne un Objet Response, celui  généré par la méthode onAuthenticationSuccess
        return  $userAuthenticator->authenticateUser($user, $authenticator, $request);
}
```

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

Définir la route à activer en cas d'erreur 401

- Par défaut lorsque un utilisateur non authentifié essaye d'accéder à une ressource protégé, un page d'erreur apparait.
- Cependant ce n'est pas le comportement standard. Ce qu'on veut généralement c'est de le rediriger vers la page de login.
- Pour ce faire, vous devez implémenter la méthode **start**.
- A l'intérieur de cette méthode implémenter le comportement que vous voulez, **c'est cette méthode qui sera appelé en cas de 401.**

https://symfony.com/doc/current/security/access_denied_handler.html#customize-the-unauthorized-response

Sécuriser Les contrôleurs

Vous pouvez directement sécuriser vos contrôleurs en utilisant :

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

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

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

```
/**
 * 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**

```
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_*')

```
{% 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

- PUBLIC_ACCESS : 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.