Rapport de sécurité sous Symfony 4.4.3

Sommaire

Introduction

Symfony intègre un système d'authentification sécurisé afin de protéger l'accès a certaines pages ainsi qu'a certaines méthode. La configuration de la sécurité sous Symfony est inscrite dans le fichier *config/packages/security.yaml*. Vous trouverez plus d'information a son sujet dans la documentation officielle de Symfony

```yaml
security:
    encoders:
        App\Entity\User:
            algorithm: bcrypt
    # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
    providers:
        doctrine:
            entity:
                class: App\Entity\User
                property: username
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: ~
            pattern: ^/
            form_login:
                login_path: /login
                check_path: /login_check
                always_use_default_target_path:  true
                default_target_path:  /
            logout:
                path: /logout
                target: /

            # activate different ways to authenticate
            # https://symfony.com/doc/current/security.html#firewalls-authentication

            # https://symfony.com/doc/current/security/impersonating_user.html
            # 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: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/users, roles: ROLE_ADMIN }
    - { path: ^/, roles: IS_AUTHENTICATED_FULLY }
    # - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }

```

Configuration

La première chose indispensable au bon fonctionnement de l'authentification, c'est la sélection des identifiants, il est donc nécessaire dans un premier temps de créer la classe correspondante en l'implémentant de la l'interface UserInterface. Cette opération peut être réalisée manuellement ou bien a l'aide de la commande Symfony :

```
php bin/console make:user
```

Encoders

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

C'est ici que nous préciserons l'algorithme d'encodage des mots de passe ainsi que la classe dans laquelle se trouve l'attribut correspondant. Pour plus d'information la documentation décris davantage l'encodage des mots de passe.

Providers

```
providers:
    doctrine:
        entity:
            class: App\Entity\User
            property: username
```

Le provider nous permet de préciser déterminer la classe et l'attribut utilisé pour l'authentification.

Access Control

```
access_control:
- { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/users, roles: ROLE_ADMIN }
- { path: ^/, roles: IS_AUTHENTICATED_FULLY }
```

Cette section définit les routes nécessitant ou non l'usage de l'authentification via des Uri ou des regex.

Firewalls

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        anonymous: ~
        pattern: ^/
        form_login:
            login_path: /login
            check_path: /login_check
            always_use_default_target_path:  true
            default_target_path:  /
        logout:
            path: /logout
            target: /
```

ce sont les configurations et les routes prises en compte par le système d'authentification pour activer et définir son fonctionnement.

Ici se distinguent 2 sections, dev et main, dev correspond au chemin permettant d'accéder au profiler (interface de debugging), qui peut être installé a l'aide de la commande

```
composer require --dev symfony/profiler-pack
```

On observe ici qu'il n'est pas nécessaire d'être connecté pour accéder au profiler ou la sécurité est désactivée (security : false), en revanche pour ce qui concerne le site (main) on y définit :

- Pattern : la route a partir de laquelle se gère l'authentification.
- Form_login : les routes de connexion ainsi que la route de redirection post connexion.
- Logout : les routes de déconnexion et de redirection post déconnexion.

SecurityController

Afin de permettre l'authentification, ce contrôleur définira une méthode login pour répondre au formulaire de connexion.
Elle utilise l'objet AuthenticationUtils qui lui sera fournit en paramètre.
Les autres routes concernant l'authentification seront définies dans le fichier config/routes.yaml.

Ce contrôleur peut être créer dynamiquement a l'aide de la console Symfony et du Maker bundle :

```
composer require symfony/maker-bundle --dev
```

```
bin/console make:auth
```

Autorisation

Une fois l'authentification mise en place, reste a savoir si l'utilisateur connecté possède ou non les droits nécessaires lui permettant d'accéder a la page demandée.

Etat

Symfony nous permet de vérifier assez simplement si un utilisateur est connecté dans un contrôleur a l'aide de la méthode :

```
$this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');
```

Pour plus d'information, rendez vous sur la documentation Symfony

Role

Les rôles sont définis dans l'entité utilisateur utilisée en tant que provider. Pour vérifier les droits de la session courante on récupère l'utilisateur courant, soit par le biais de l'objet ou alors via le service security a condition de récupérer la classe en question :

```
$this->getUser();
```

```
$this->security->getUser();
```

Il ne reste donc plus qu'a récupérer le tableau des droits a l'aide de la méthode présente dans l'entité getRoles().

Condition

Des lors que l'on souhaite tester directement les droits de l'utilisateur, nous utiliserons l'une des deux méthodes fournit avec le security-bundle de Symfony :

Soit isGranted() (littéralement : possède l'accès) en lui précisant les droits minimums nécessaires, Symfony va par la suite vérifier que la session courante dispose ou non des droits précisés.

Soit denyAccessUnlessGranted() (littéralement : refuser l'accès sauf si) en lui précisant un paramètre de droit nécessaire et si souhaité, un message en cas d'erreur.
Les paramètres définis dans Symfony sont les suivant :

- IS_AUTHENTICATED_REMEMBERED,
- IS_AUTHENTICATED_FULLY,
- IS_AUTHENTICATED_ANONYMOUSLY

Pour plus d'information la documentation précisera la raison de chacun d'eux.

Ou alors tout simplement a l'aide de l'access_control précédemment évoqué.

Voters

Si la vérification se fait via la méthode denyAccessUnlessGranted(), il est également possible de créer nos propres paramètres et ainsi de voir leurs comportement évoluer en fonction de nos besoins. On nommera cette classe : Voter. On la liera au contrôleur l'utilisant.
On crée ce ficher, soit dynamiquement a l'aide du maker-bundle en lui précisant le contrôleur, il se trouvera dans src/Security/Voter/.

```
bin/console make:voter
```

Soit manuellement en l'étendant de la classe voter

```php
class TaskVoter extends Voter
{

    public const DELETE = 'delete';
    public const EDIT = 'edit';
```

Dans un premier temps on défini les constantes correspondants aux paramètres nécessaire

```php
    /**
     * @var Security
     */
    private $security;

    /**
     * TaskVoter constructor.
     * @param Security $security
     */
    public function __construct(Security $security)
    {
        $this->security = $security;
    }
```

Dans le constructeur on récupère le composant security en usant de l'injection de dépendance permettant l'usage d'un composant directement sans l'instancier, Symfony le fait lui même. Celui ci nous servira a récupérer l'utilisateur via sa méthode getUser().

```php
protected function supports($attribute, $subject)
{
    // https://symfony.com/doc/current/security/voters.html
    if (!in_array($attribute, [self::DELETE, self::EDIT], strict: true)) {
        return false;
    }

    if (!$subject instanceof Task) {
        return false;
    }

    return true;
}
```

Dans la méthode supports() on vérifie dans un premier temps que l'attribut fournis en paramètre est défini dans le voter et également que le l'attribut $subject correspond bien a la classe attendu, ici **Task**, dans les cas contraire, une sortie négative est effectuée n'effectuant donc pas la requête.

```php
protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
{
    $user = $token->getUser();
    // if the user is anonymous, do not grant access
    if (!$user instanceof User) {
        return false;
    }
    $task = $subject;
    // ... (check conditions and return true to grant permission) ...
    switch ($attribute) {
        case self::EDIT:
            return $this->canEdit($user);
            break;
        case self::DELETE:
            return $this->canDelete($task, $user);
            break;
    }

    return false;
}
```

Ensuite vient la méthode voteOnAttribute(), qui va définir l'action a effectuée en fonction d'un switch-case, dans l'exemple ci-dessus le retour se fait via une autre méthode a définir par la suite.

```php
private function canEdit(User $user)
{
    return $user ? true : false;
}

private function canDelete(Task $task, User $user)
{
    if ($user === $task->getUser()) {
        return true;
    }

    return ($this->security->isGranted( attributes: 'ROLE_ADMIN') && 'anonyme' === $task->getUser());
}
```

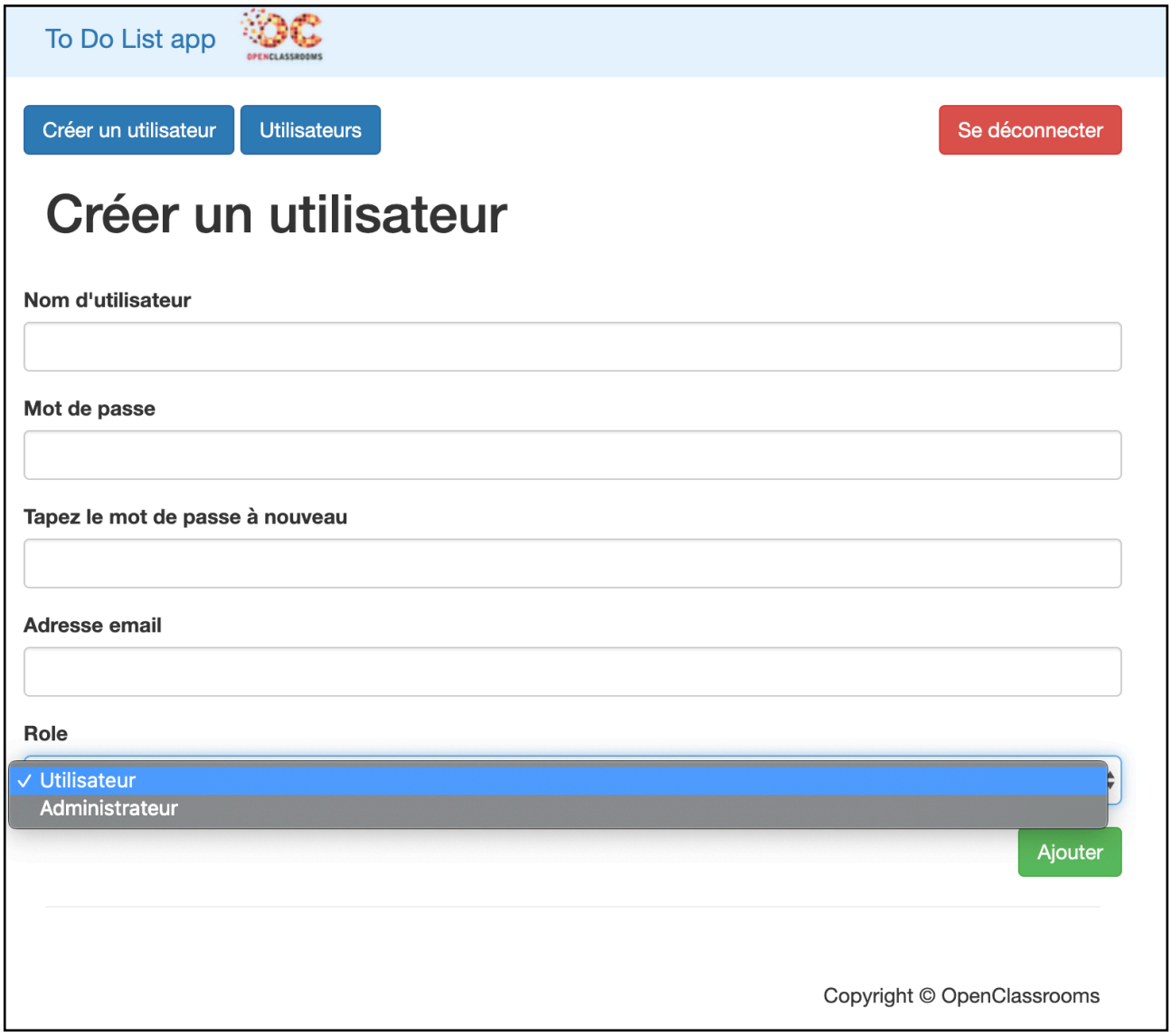
Pour la première méthode, canEdit(), tous les utilisateurs ont ce droit, si $user est défini alors un utilisateur est connecté. Il aurait également été possible de faire un retour de :

```php
$this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');
```

Pour la seconde méthode, on vérifie que l'utilisateur effectuant la requête soit le propriétaire de la tache, si ce n'est le cas, on vérifie que si l'utilisateur est un admin et si la tache est attribué a l'utilisateur anonyme, dans quel cas True sera renvoyer, sinon False

Création d'un administrateur

La mise en place d'un administrateur peut se faire a l'aide de deux procédures distinctes, Soit en passant par l'interface utilisateur si elle est développée, dans le cas de todo&co, la route « user_create » via son url a condition d'être connecté et d'être sois même un administrateur,



Soit en passant par la base de donnée, il sera cependant nécessaire d'inscrire un hash et non le mot de passe de l'utilisateur, ceci peut être effectué a l'aide des commandes fournis par Symfony

```
$ php bin/console security:encode-password
```