Documentation Technique –

Systeme d'Authentification

Jérôme Dubus – Développeur d'application PHP/Symfony

Table des matières

Contexte

L'application ToDoList est accessible uniquement aux utilisateurs authentifiés.

Seul un administrateur peut avoir accès à la gestion des utilisateurs de l'application.

L'authentification passe par un formulaire de connexion en renseignant un identifiant et un mot de passe.

Un système de rôle a été mis en place pour restreindre l'accès à certaines pages de l'application, ainsi qu'une restriction concernant la suppression d'une tâche qui doit être faite uniquement par l'auteur.

Dans ce document, vous retrouverez comment a été implémenté le système d'authentification du projet sous Symfony avec :

- Les différents fichiers utiles pour le paramétrage
- La procédure d'authentification
- La gestion des rôles utilisateur

Système d'authentification

Les Utilisateurs

Une authentification a été mise en place afin de définir l'identité des utilisateurs et de restreindre l'accès uniquement à certaines parties.

Afin de s'authentifier, un nom d'utilisateur et un mot de passe devront être renseignés dans le formulaire d'authentification.

On distinguera trois types d'utilisateurs qui sont les suivants :

- **Utilisateur non authentifié :** Lorsqu'un utilisateur n'est pas authentifié il sera considéré comme anonyme et se verra automatiquement redirigé vers la page d'authentification s'il souhaite parcourir l'application.

- **Utilisateur :** Celui-ci dispose du rôle utilisateur simple (ROLE_USER) et ne pourra que créer ou modifier ses propres tâches. Il pourra également consulter toutes ses tâches (terminées ou pas).

- **Administrateur :** Cet utilisateur se voit attribuer les droits les plus importants de par son rôle (ROLE_ADMIN). Il pourra créer et modifier un utilisateur, mais aussi modifier ses droits d'accès à l'application. Créer, modifier et supprimer ses propres tâches ou celles ayant un auteur anonyme.

Les Fichiers

Voici la liste des fichiers utiles pour la réalisation de l'authentification :

Emplacement	Fichier	Description
src/Entity/	User.php	Classe de l'entité Utilisateur
src/Controller/	SecurityController.php	Contrôleur contenant les routes de login et logout
src/Security/Voter	TaskVoter.php	Système de vérification des autorisations pour les taches
config/packages/	Security.yaml	Paramétrage du Firewall
templates/security/	Login.html.twig	Template du formulaire de connexion
src/Form/	UserType	Formulaire de connexion

La Sécurité

La gestion de la sécurité et des utilisateurs s'effectue via le fichier **security.yaml** qui contient plusieurs paramètres importants pour le bon fonctionnement de l'application.

Nous allons détailler ce fichier :

```
security:
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
'auto'
```

Le fichier commence par le **type d'encryptage du mot de passe :**

Le mode d'encryptage du mot de passe est défini sur « auto » car il est recommandé de laisser le Framework Symfony choisir la méthode d'encryptage la plus sécurisée pour sa version.
En effectuant les montées de Version, ce mode d'encryptage a changé de nom (de encoder à hasher).

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

Le provider indique où aller chercher les informations nécessaires à l'authentification :
Ici c'est la classe User que l'on utilise avec comme propriété le username (nom d'utilisateur) pour la connexion avec le mot de passe ainsi que l'utilisation de Doctrine pour l'accès aux données.

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

```
            logout: ~
            access_denied_handler: App\Security\AccessDeniedHandler
            entry_point: App\Security\AuthenticationEntryPoint
```

Le firewall (pare-feu) prend l'authentification en compte et restreint l'accès à certaines routes de l'application.
Ces paramètres indiquent que si l'utilisateur n'est pas authentifié, il se verra rediriger vers la page de connexion. (Login).

```
  role_hierarchy:
        ROLE_ADMIN: ROLE_USER
```

La hiérarchie indique au Framework qu'un rôle contient aussi les droits liés à un autre rôle : Ici, on spécifie que le rôle ADMIN détient les droits du rôle USER en plus de ceux qui lui sont attribués.

```
«   access_control:
        - { path: ^/login, roles: PUBLIC_ACCESS }
        - { path: ^/users, roles: ROLE_ADMIN }
        - { path: ^/, roles: ROLE_USER }
```

Le contrôle des accès se fait via ces paramètres :
« path » définie la route accessible par l'utilisateur.
« roles » représente le niveau d'authentification nécessaire pour accéder à cette route. On notera qu'un utilisateur non identifié n'aura accès qu'à la page de connexion (login) avant de se connecter et qu'on lui octroie l'accès à d'autres routes.

Documentation : https://symfony.com/doc/6.4/security.html

Stockage des données

Les utilisateurs sont stockés en base de données, au sein de la table user représentée dans l'application par la classe src/Entity/User.php .

Grace à l'ORM Doctrine, Symfony peut créer notre base de données et nos tables uniquement grâce aux paramètres de nos entités.

```php
#[ORM\Entity]
#[UniqueEntity('email', message: 'Email déjà utilisé')]
#[UniqueEntity('username', message: "nom d'utilisateur déjà utilisé")]
#[ORM\Table('user')]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Column(type: 'integer')]
    #[ORM\Id]
    #[ORM\GeneratedValue(strategy: 'AUTO')]
    private ?int $id = null;

    #[ORM\Column(type: 'string', length: 25, unique: true)]
    #[Assert\NotBlank(message: "Vous devez saisir un nom d'utilisateur.")]
    private string $username;

    #[ORM\Column(type: 'string', length: 64)]
    #[Assert\PasswordStrength(message:'Mot de passe trop simple à deviner')]
    private string $password;

    #[ORM\Column(type: 'string', length: 60, unique: true)]
    #[Assert\NotBlank(message: 'Vous devez saisir une adresse email.')]
    #[Assert\Email(message: "Le format de l'adresse n'est pas correcte.")]
    private ?string $email = null;
```

L'implémentation de UserInterface et de PasswordAuthenticatedUserInterface sont absolument nécessaires au bon fonctionnement de notre authentification.

Les paramètres de chaque données de notre table peuvent être modifiés à l'aide des attributs.

Voici à quoi ressemble notre table User en base de données :

	id	username	password	email	roles
▶	1	test	$2y$04$cItZ56nVdYIMD6/iXAeOieKrQfKVONL9e...	test@email.com	["ROLE_USER"]
	2	admin	$2y$04$4loQDITtTSAYw64mRbM/Ie0.qb6Yysv/...	admin@email.com	["ROLE_ADMIN"]

Documentation : https://symfony.com/doc/6.4/doctrine.html

Autorisation

Les Voters constituent le moyen le plus puissant de Symfony pour gérer les autorisations. Ils vous permettent de centraliser toute la logique d'autorisation, puis de les réutiliser à de nombreux endroits.

```
class TaskVoter extends Voter
{
    public const DELETE = 'TASK_DELETE';
    public const MODIFY = 'TASK_MODIFY';

    protected function supports(string $attribute, mixed $subject): bool
    {
        // replace with your own logic
        // https://symfony.com/doc/current/security/voters.html
        return \in_array($attribute, [self::DELETE, self::MODIFY], true)
            && $subject instanceof Task;
    }
```

Voter::supports(string $attribute, mixte $subject)
Lorsque isGranted() (ou denyAccessUnlessGranted()) est appelé, le premier argument est passé ici en tant que $attribute (par exemple ROLE_USER, task) et le deuxième argument (le cas échéant) est passé en tant que $subject (par exemple null, un objet Task).
Votre travail consiste à déterminer si votre électeur doit voter sur la combinaison attribut/sujet.
Si vous retournez true, voteOnAttribute() sera appelé. Sinon, votre Voter a terminé.

```
    protected function voteOnAttribute(string $attribute, mixed $subject,
TokenInterface $token): bool
```

```
    {
        $user = $token->getUser();
        // if the user is anonymous, do not grant access
        if (!$user instanceof User) {
            return false;
        }
        /** @var Task $task */
        $task = $subject;

        switch ($attribute) {
            case self::DELETE:
                return $this->canDelete($task, $user);
            case self::MODIFY:
                return $this->canModify($task, $user);
        }

        return false;
    }
```

voteOnAttribute(string $attribute, mixte $subject, TokenInterface $token)
Si vous retournez true depuis supports(), alors cette méthode est appelée.
Votre travail consiste à renvoyer true pour autoriser l'accès et false pour refuser l'accès. Le $token peut être utilisé pour trouver l'objet utilisateur actuel (le cas échéant).

La partie ci-dessous renferme toute la logique métier pour déterminer l'accès.

```
    private function canDelete(Task $task, User $user): bool
    {
        return $user === $task->getUser() || (null === $task->getUser() && $user->getRoles() === ['ROLE_ADMIN']);
    }

    private function canModify(Task $task, User $user): bool
    {
        return $user === $task->getUser() || (null === $task->getUser() && $user->getRoles() === ['ROLE_ADMIN']);
    }
}
```

Documentation : https://symfony.com/doc/6.4/security/voters.html

Formulaire

L'utilisateur non authentifié, comme vu précédemment, sera redirigé à chaque demandes vers la page de connexion. (templates/security/login.twig.html)

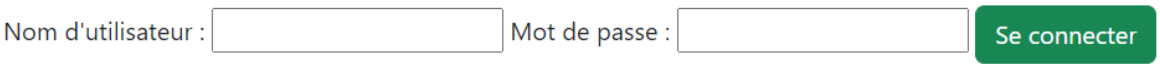
```php
#[Route(path: '/login', name: 'login')]
public function login(AuthenticationUtils $authenticationUtils): Response
{
    if ($this->getUser()) {
        return $this->redirectToRoute('homepage');
    }

    $error = $authenticationUtils->getLastAuthenticationError();
    $lastUsername = $authenticationUtils->getLastUsername();
```

Afin d'accéder à cette page la méthode **login ()** du **SecurityController** est exécutée. Celle-ci va alors générer notre vue et renvoyer des données si nécessaire.

```php
    return $this->render('security/login.html.twig', [
        'last_username' => $lastUsername,
        'error'         => $error,
    ]);
}
```

Un formulaire est alors soumis à l'utilisateur pour poursuivre sa navigation.



Les identifiants, une fois renseignés, sont alors validés à l'aide de la librairie security-bundle après consultation de la base de données.
S'ils correspondent à un utilisateur, celui-ci se verra redirigé vers la page d'accueil. Dans le cas contraire une erreur s'affichera et il devra réitérer sa tentative.

Documentation : https://symfony.com/doc/6.4/formulaire.html

Contrôleur

Le contrôleur SecurityController renferme les routes pour l'authentification et la déconnexion.

```php
class SecurityController extends AbstractController
{
    /**
     * manage login form
     */
    #[Route(path: '/login', name: 'login')]
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        if ($this->getUser()) {
            return $this->redirectToRoute('homepage');
        }

        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security/login.html.twig', [
            'last_username' => $lastUsername,
            'error'         => $error,
        ]);
    }

    /**
     * Check login
     */
    #[Route(path: '/login_check', name: 'login_check')]
    public function loginCheck(): void
    {
        // This code is never executed.
    }

    /**
     * Check logout
     */
```

```
    #[Route(path: '/logout', name: 'logout')]
    public function logoutCheck(): void
    {
        // This code is never executed.
    }
}
```

- Login : pour l'affichage du formulaire de connexion
- Login_check : récupérer par les listeners pour le traitement du formulaire et donc permettre l'authentification grâce à l'authenticator de base de Symfony
- Logout : pour la déconnexion