ToDo & Co – TodoList

Documentation technique

Authentification

1. Mise en place du processus d'authentification

1.1. Installation

Le package utilisé pour la gestion de sécurité et le composant « security » de Symfony .

Pour une installation du webskeleton il sera directement installé mais sinon il suffit de lancer la commande suivante pour l'installer :

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

1.2. Configuration

Lors de l'installation, un ficher de configuration est crée dans le dossier `config/packages`. C'est le fichier `security.yaml`

Voyons quelles sont les principaux éléments de ce fichier de configuration et comment l'utiliser.

encoder :

L'encodeur permet de hacher les mots de passe pour nos utilisateurs et lui précisant l'entité sur laquelle il va travailler. On définit aussi l'algorithme qui sera utilisé lors du cryptage : `bcrypt` dans notre cas.

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

providers :

Pour que le composant de Sécurité comprenne que nous utilisateurs viennent de la base de données, il faut le lui préciser en créant ce qu'on appelle un **provider** (un fournisseur de données utilisateurs).

```
providers:
    users_in_memory: { memory: null }

    in_database:
        entity:
            class: App\Entity\User
            property: username
```

firewalls :

Les firewalls définissent les parties de notre application et comment elles sont sécurisées. Le firewall `main` représente toute notre application.

Ici nous indiquons que nous utilisons le provider indiqué en haut et que pour s'authentifier il faudra passer par le formulaire de login accessible sur la route `account_login`. Le `check_path` assure que une fois le formulaire de login soumis ses données sont traitées par la route `account_login`.

La section logout définit la route de déconnexion et la route cible (route de redirection) une fois déconnecté.

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

    main:
        anonymous: true
        provider: in_database

        form_login:
            login_path: account_login
            check_path: account_login

        logout:
            path: account_logout
            target: account_login
```

Access_control :

Ici il s'agit de définir la sécurisation d'accès (autorisation) pour une section d'URLs. Par exemple ici nous indiquons que le chemin /login est accessible sans authentification mais que le chemin /users requiert un rôle administrateur.

```
access_control:
    # - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/users, roles: ROLE_ADMIN }
    - { path: ^/, roles: ROLE_USER }
```

role_hierarchy:

Un rôle peut hériter d'autres rôles. Ici un administrateur (ROLE_ADMIN) possède également le rôle d'un utilisateur standard (ROLE_USER).

```
role_hierarchy:
    ROLE_ADMIN: ROLE_USER
```

2. Autorisation

L'autorisation consiste à permettre à un utilisateur authentifié d'accéder à une fonctionnalité en fonction de son rôle.

Il existe plusieurs façons d'autoriser un utilisateur :

a) Dans le fichier de configuration de la sécurité (security.yaml) dans sa section access_control décrite précédemment.
b) Au sein d'un contrôleur :
    - Au dessus de la méthode qui représente l'action à exécuter avec une annotation @IsGranted

- A l'intérieur de la méthode avec la fonction isGranted()

```
if ($task->getAuthor() == $this->getUser() || ('anonyme' == $task->getAuthor()->getUsername() && $this->isGranted('ROLE_ADMIN'))) {
```

c) Dans la vue twig avec la fonction is_granted().

d) Ou avec le système de voters où dans une classe dédiée on définit, sur un objet, les différents droits en fonctions des rôles et cela de façon centralisée.

```
class TaskVoter extends Voter
{
    protected function supports(string $attribute, $subject): bool
    {
        // replace with your own logic
        // https://symfony.com/doc/current/security/voters.html
        return in_array($attribute, ['CAN_EDIT', 'CAN_TOGGLE', 'CAN_DELETE'])
            && $subject instanceof \App\Entity\Task;
    }

    protected function voteOnAttribute(string $attribute, $subject, TokenInterface $token): bool
    {
        $user = $token->getUser();
        // if the user is anonymous, do not grant access
        if (!$user instanceof UserInterface) {
            return false;
        }

        // ... (check conditions and return true to grant permission) ...
        switch ($attribute) {
            case 'CAN_EDIT':
                return $subject->getAuthor() === $user || ('anonyme' === $subject->getAuthor()->getUs
            case 'CAN_TOGGLE':
                return $subject->getAuthor() === $user || ('anonyme' === $subject->getAuthor()->getUs
            case 'CAN_DELETE':
```

Et dans le l'action du contrôleur on n'autorisera que celui qui a le droit de faire cette action :

```
/**
 * Permet de supprimer une tâche.
 *
 * @Route("/tasks/{id}/delete", name="task_delete")
 */
public function deleteTaskAction(Task $task)
{
    $this->denyAccessUnlessGranted('CAN_DELETE', $task, "Vous n'êtes le propriétaire de cette tâch
```

3. Conclusion

Pour plus de détails sur le composant « security » de Symfony rendez vous sur le site officiel de Symfony ;