DOCUMENTATION TECHNIQUE
Guide d'authentification

V.1

Date de la dernière mise à jour : 15/06/2024

sommaire

1. Contexte

1.1 Présentation du projet

Introduction générale

Le projet To Do & Co est une application symfony permettant de gérer facilement les tâches quotidiennes.

Elle a été initialement développée sous Symfony `3.1` et a été réécrite sous Symfony `7.0.4`.
Lors de cette réécriture, de nouvelles fonctionnalités ont été mises à jour et/ou créées afin de répondre aux standards de sécurité actuels.
Cette documentation en détaillera l'implémentation et la procédure à suivre.

Fonctionnalités principales

- Gestion des tâches :
  - CRUD + fonction de changement de statut

    Chaque utilisateur à la main sur le CRUD de ses propres tâches et peuvent modifier le statut de n'importe quelle tâche.
- Gestion des utilisateurs
  - CRUD + gestion des tâches anonyme par l'administrateur

    Seul un administrateur a accès au CRUD des utilisateurs. Eux seuls peuvent également modifier ou supprimer une tâche anonyme.

Framework & Library

- Framework Symfony `v 7.0.6`
- MySql `v 8.0.31`
- Php : `v 8.2`
- Symfony/security-bundle : `v 7.0.7`

1.2 Authentification

Principaux composants utilisés dans le projet

L'authentification dans un projet Symfony repose sur plusieurs composants.

- le composant `Security` qui gère l'authentification des utilisateurs via différentes méthodes telles que le formulaire de login, OAuth, ou l'authentification HTTP.
- le composant `PasswordHasher` qui gère les mots de passe sécurisés des utilisateurs qui sont définis dans des entités.
- le fichier `security.yaml` qui permet de configurer l'authentification, il est essentiel de définir les pare-feux et les fournisseurs d'utilisateurs.
- L'accès aux différentes parties de l'application est contrôlé par des rôles et des permissions, définis par des annotations ou des configurations.
- le service `UserProvider` et les événements de sécurité, Symfony permet de personnaliser le processus d'authentification selon les besoins spécifiques du projet.

2. Gestion de l'authentification

2.1 Configuration Composant Security de Symfony

2.2.1 - UserProvider

Utilisation

Le UserProvider est un composant du système de sécurité de Symfony utilisé pour l'authentification. Il est responsable de la récupération des utilisateurs depuis une ou plusieurs sources de données. To Do & Co utilise une base de données dont le détails sera donné par la suite.

Rôle

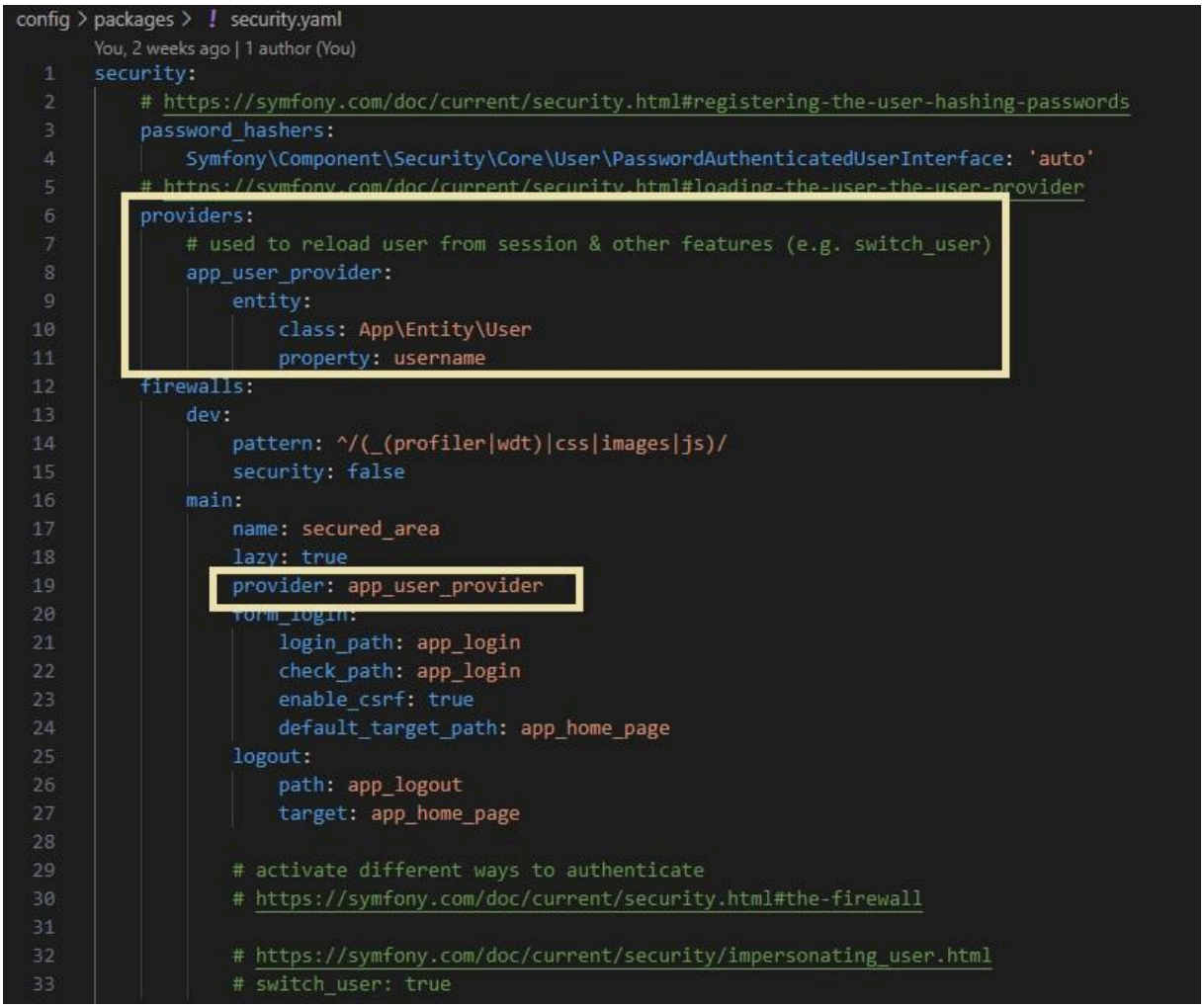
Il joue un rôle essentiel dans le processus d'authentification et d'autorisation en fournissant les informations nécessaires sur les utilisateurs pour les valider et leur attribuer des permissions.

Fonctionnement

Il charge les utilisateurs en fonction d'un identifiant unique, ici le `username` de la classe `App\Entity\User.`

Il doit implémenter `UserProviderInterface`, qui définit les méthodes nécessaires pour charger et rafraîchir les utilisateurs. Dans To Do & Co, nous utilisons directement la

configuration dans le fichier `security.yaml` pour déterminer le provider.



2.2.2 - Entité User

L'entité User est nécessaire au composant Security pour le bon fonctionnement de l'authentification. C'est une classe PHP qui représente un utilisateur.

Propriétés

Certaines propriétés sont nécessaires à l'authentification:

- `username`, identifiant unique permettant d'identifier un utilisateur.
- `password`, mot de passe sécurisé et hashé permettant l'authentification.
- `role`, tableau de rôles permettant de gérer les accès des utilisateurs.
- `tasks`, liste des tâches rattachées à chaque utilisateur.

Méthodes

En découle que ces méthodes sont importantes pour l'authentification:

- `getUserIdentifier()`, permet d'obtenir une chaîne de caractère représentant l'identité de l'utilisateur.
- `getRoles()`, permet d'obtenir la liste des rôles de l'utilisateur sous forme de tableau. Pour être pris en compte par l'authentification de Symfony, chacune doit commencer par `ROLE_`.
- `setPassword()`, permet de définir le mot de passe de l'utilisateur haché.
- `getTasks()`, permet d'obtenir la liste des tâches rattachées à l'utilisateur grâce à une validation SQL en utilisant l'ORM Doctrine et une relation OneToMany.

2.2.3 - Session utilisateur

La gestion de la session utilisateur de Symfony offre une solution robuste et flexible pour maintenir l'état de l'utilisateur connecté et donc de sécuriser l'application.

Fonctionnement

- Suite à l'authentification, Symfony crée une session pour l'utilisateur stockant les informations de celui-ci qui sont accessibles dans les contrôleurs via la méthode `$this->getUser()` ou dans l'objet `Security`.
- Cela permet d'appliquer des logiques métiers selon l'utilisateur.
  La session a une durée de vie configurée pour durer le temps de la connexion, mais elle peut être configurée si nécessaire dans le fichier `framework.yaml`.

Voici un exemple :

config > packages > framework.yaml

You, 2 minutes ago | 1 author (You)

```yaml
# see https://symfony.com/doc/current/reference/configuration/framework.html
framework:
    secret: '%env(APP_SECRET)%'
    csrf_protection: true

    # Note that the session will be started ONLY if you read or write from it.
    session: true
        cookie_lifetime: 86400
```

2.2 Méthodes d'authentification

2.2.1 Formulaire d'enregistrement

Ce formulaire nous permet de récupérer les informations nécessaires à la création sécurisée d'un utilisateur. Il est soumis à différentes contraintes.

- Chaque entrée est typée

```
->add('username', TextType::class,
```

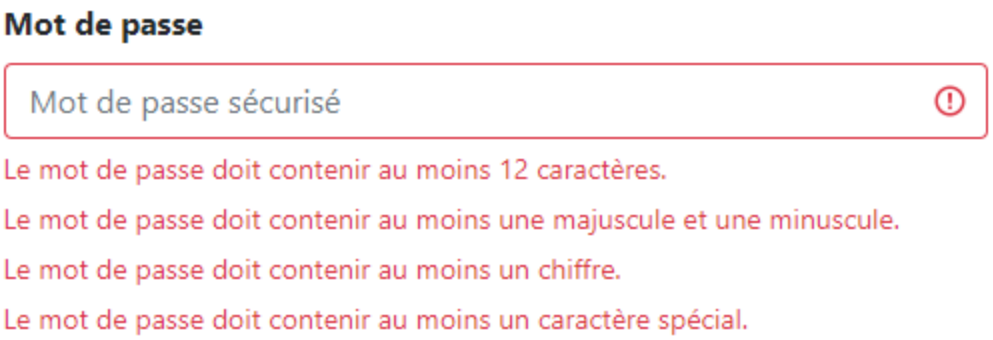
- L'unicité de propriété par une assertion avec des annotations doctrine. `username` pour l'authentification et de l'`email` pour éviter les doublons d'utilisateurs.

```
#[ORM\Column(length: 255, unique: true)]
private ?string $username = null;
```

- Le mot de passe haché. La configuration du hachage est en automatique, Symfony détermine automatiquement le meilleur algorithme de hachage. A ce jour, il utilise `bcrypt` avec un coût de 13, qui est une option de stockage sécurisé.

```
config > packages > ! security.yaml
       You, 2 weeks ago | 1 author (You)
1      security:
2          # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
3          password_hashers:
4              Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

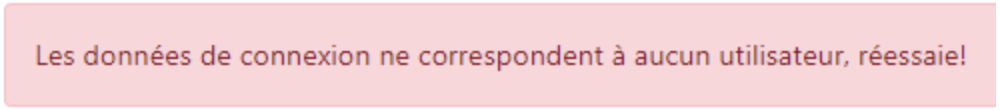
- Les mots de passe doivent suivre les recommandations de la CNIL (https://www.cnil.fr/fr/generer-un-mot-de-passe-solide) qui sont concrétisé dans une contrainte personnalisée `src\Validator\Constraints\PasswordRequirements.php` . Elle a été créée en étendant Compound.



- Activation de la vérification des token CSRF généré par Symfony lors des soumissions de formulaire.

2.2.2 Formulaire de connexion

- Vérification des données de connexion. Après soumission du formulaire de connexion, Symfony valide pour s'assurer qu'elles respectent les contraintes définies
- Récupération de l'utilisateur via le `UserProvider` et de ses données en utilisant `username`
- Comparaison du mot de passe hashé stocké en base de données avec le mot de passe soumis par l'utilisateur après hachage
- Authentification de l'utilisateur si l'identification est valide et que le mot de passe correspond.
- Gestion et affichage des erreurs de connexion en cas d'échec d'authentification



3. Gestion des autorisations

3.1 Les rôles

Définition

Deux rôles sont définis dans To Do & co

```
public const ROLE_USER = 'ROLE_USER';

public const ROLE_ADMIN = 'ROLE_ADMIN';
```

`"ROLE_USER"` : détermine les droits et accès d'un utilisateur

`"ROLE_ADMIN"` : détermine les droits et accès d'un administrateur

Un troisième statut est possible. Un administrateur a la possibilité de supprimer tous les rôles d'un utilisateur, il est possible de le configurer grâce aux formulaires de création ou d'édition en sélectionnant "visiteur".

Attribution

Le rôle `"ROLE_USER"` est attribué par défaut à chaque création d'utilisateur lors de l'utilisation des formulaires, sauf sélection explicite dans ceux-ci.

3.2 Voter

Définition

Deux voters personnalisés sont utilisés dans l'application

- `TaskVoter.php`
- `UserVoter.php`

Fonctionnement

Des constantes sont déterminées afin de gérer l'accès à certaines méthodes des entités Task et User et permettent d'appliquer une logique d'autorisation via un processus booléen.

```php
public const CREATE = 'TASK_CREATE';
public const DELETE = 'TASK_DELETE';
public const LIST = 'TASK_LIST';
public const EDIT = 'TASK_EDIT';
public const TOGGLE = 'TASK_TOGGLE';
```

Les voters personnalisés étendent la classe `Voter`, qui implémentent deux méthodes nécessaires au fonctionnement de l'autorisation d'accès

- `supports()`
- `voteOnAttribute()`

Vérification de l'auteur d'une tâche

Le `TaskVoter` permet également de vérifier que l'utilisateur avec un ROLE_USER est bien le propriétaire de la tâche qu'il souhaite modifier. Pour cela la méthode `checkOwner()` est utilisée lors de l'appel des constantes `"TASK_EDIT"` ou `"TASK_DELETE"`. Elle utilise la méthode `getOwner()` de l'entité `User` et le compare à l'utilisateur authentifié.

```php
/**
 * Function to check Owner of Task.
 *
 * @param mixed          $subject the subject of voter
 * @param TokenInterface $token   The token
 * @return bool|AccessDeniedException
 */
protected function checkOwner(mixed $subject, TokenInterface $token): bool|AccessDeniedException
{
    $user = $token->getUser();
    $checkIdUser = $subject?->getOwner() === $user;

    return $checkIdUser ? $checkIdUser : throw new AccessDeniedException("Vous n'êtes pas le propriétaire
}
```

3.3 Contrôle des accès dans les vues

Deux méthodes sont utilisées dans les vues pour autoriser les accès à certaines pages ou fonctionnalités:

```twig
{% if is_granted('USER_CREATE') %}
```

Cette méthode vérifie si le rôle de l'utilisateur authentifié valide la permission `"USER_CREATE"`, qui est une constante du Voter `UserVoter.php`. Si c'est le cas, l'utilisateur a accès au code contenu dans le `if.`

```
{% if app.user %}
```

Cette méthode vérifie si l'utilisateur est authentifié quel que soit leur rôle, comme par exemple pour l'affichage des boutons "Connexion" et "Déconnexion".

3.4 Contrôle des accès aux routes

Le contrôle d'accès aux routes peut se faire de plusieurs manières. Actuellement l'annotation `@IsGranted` du composant `Security` de Symfony est utilisée pour accorder la permission ou non à l'utilisateur d'accéder à la route. Les constantes des voters sont utilisées avec l'annotation.

```
#[Route('/_list/{status}/{page}', name: '_list')]
#[IsGranted('TASK_LIST')]
public function listAction(TaskStatus $status, int $page): Response
{
```

Il est également possible de gérer l'accès aux routes avec le fichier `security.yaml` en configurant l'`access_control.`

```
# 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 }
    # - { path: ^/profile, roles: ROLE_USER }
```

4. Bonnes pratiques & Documentations officielles

Bonnes pratiques

Voici les bonnes pratiques selon Symfony.

- Utiliser des Hashs de mot de passe sécurisés.
- Utiliser une connexion sécurisée HTTPS.
- Limiter les tentatives de connexion pour prévenir les attaques par force brute.
- Vérification des données d'entrée pour éviter les attaques par injection.
- Protection contre les CRSF.
- Gestion des sessions.
- Stockage sécurisé des données d'authentification.
- Double authentification.
- Utilisation de bibliothèques et de frameworks. Utilisez les composants de sécurité éprouvés pas Symfony.
- Audit et journaux, enregistrer toutes les tentatives de connexion pour détecter les comportements suspects.
- Mises à jour et correctifs. Maintenir le logiciel à jour, permet d'obtenir les derniers correctifs de sécurité.

Documentation officielle

Pour plus de détails, consulter la documentation officielle.

- Composant Security
- Contrainte Compound
- Utilisation et création des Voters
- Utilisation de IsGranted dans twig