Processus d'authentification

Le processus d'authentification est configuré dans le fichier de configuration de sécurité `security.yaml` situé dans le dossier `config > packages`.
La documentation officielle de Symfony est disponible ici.

L'entité User est utilisée pour l'authentification basée sur sa propriété username combinée de façon classique avec la propriété password.

Les hashers de mot de passe

L'application nécessite qu'un utilisateur se connecte avec un mot de passe. Pour ces applications, le SecurityBundle fournit des fonctionnalités de hachage et de vérification de mot de passe.

C'est pourquoi la classe User implémente l'interface `PasswordAuthenticatedUserInterface`. L'interface `PasswordAuthenticatedUserInterface` mise en œuvre sélectionne et migre automatiquement le meilleur algorithme de hachage grâce au paramètre 'auto'.

Cette configuration définit comment les mots de passe des utilisateurs doivent être hachés et vérifiés lors de l'authentification.

On peut encore utiliser le service `UserPasswordHasherInterface` pour hacher les mots de passe avant de sauvegarder les utilisateurs dans la base de données.

Le user provider

Toute section sécurisée de l'application fait appel à cette notion d'utilisateur (user). Le user provider charge les utilisateurs depuis la base de données du projet, sur la base d'un identifiant utilisateur ou user identifier.

Le user provider est utilisé pour des fonctionnalités optionnelles comme "se souvenir de moi" (avec par exemple l'implémentation de l'alimentation automatique des identifiants avec le last username), pour recharger l'utilisateur depuis la session & d'autres fonctionnalités (par exemple, switch_user). La propriété définie est celle utilisée pour se connecter, dans notre cas, elle est définie sur le username qui est forcément unique.

Le user provider sait comment (re)charger les utilisateurs depuis un stockage (par exemple, une base de données) sur la base d'un "user identifier" (par exemple, l'adresse email ou le nom d'utilisateur).

config/packages/security.yaml

```yaml
providers:
    app_user_provider:
        entity:
            class: App\Entity\User
            property: username
```

La configuration ci-dessus utilise Doctrine pour charger l'entité User en utilisant la propriété username comme "identifiant utilisateur".

Le pare-feu

Le pare-feu est le cœur de la sécurisation de l'application. Chaque requête à l'intérieur du pare-feu est checkée pour savoir si elle nécessite un utilisateur authentifié. Le pare-feu prend également en charge l'authentification de cet utilisateur (par exemple, à l'aide d'un formulaire de connexion) et définit quelles parties de l'application sont sécurisées et comment les utilisateurs pourront s'authentifier.

En mode développement, la configuration indique que les URL commençant par /_(profiler|wdt)|css|images|js)/ ne nécessitent pas de sécurité et sont donc accessibles sans restriction. Cela permet d'accéder aux outils de débogage (Profiler, Web Debug Toolbar) ainsi qu'aux ressources statiques (CSS, images, JS) dans l'environnement de développement sans avoir à se soucier de l'authentification.

Il configure également comment le processus d'authentification sera géré. Dans notre cas, en mode de production, le LoginAuthenticator est utilisé sur la base du `app_user_provider` mentionné ci-dessus, qui gère la récupération des informations utilisateur depuis la base de données.

Le LoginAuthenticator est responsable de la vérification des identifiants et de la gestion des flux d'authentification. Il existe également des paramètres pour la déconnexion : son chemin "/logout" et l'URL de redirection via le paramètre cible.

Le 'lazy: true' active le chargement du pare-feu en mode minimal. Cela signifie que le processus de sécurité sera déclenché uniquement lorsque cela sera nécessaire, ce qui peut améliorer les performances en évitant les opérations de sécurité coûteuses si elles ne sont pas nécessaires pour une demande spécifique.

La classe `LoginAuthenticator` se trouve dans le dossier `src/Security`.

On peut changer le chemin de redirection après la connexion grâce à la ligne :

return new RedirectResponse($this->urlGenerator->generate('homepage'));

Il est possible de changer le nom de la route `homepage` par un autre si on a besoin d'une autre redirection après la connexion.

Un `LoginSubscriber` est disponible dans le dossier `src/EventListener` pour personnaliser les messages flash lors de connexion / déconnexion réussies, grâce aux événements `LoginSuccessEvent` et `LogoutEvent`.

Le contrôle d'accès et la hiérarchie des rôles

Contrôle d'accès

D'abord, le contrôle d'accès affine l'autorisation nécessaire pour accéder à certains chemins, par exemple certains chemins peuvent être rendus accessibles à tout utilisateur ou seulement aux utilisateurs admins. Il permet de définir quelles parties de l'application sont accessibles par quelles catégories d'utilisateurs selon leurs rôles.

Nous devons spécifier des règles pour les chemins (URL) de l'application et les rôles nécessaires pour y accéder. Cela s'appelle l'autorisation, et son rôle est de décider si un utilisateur peut accéder à une ressource (une URL, un objet de modèle, un appel de méthode).

Le processus d'autorisation a deux facettes différentes :

L'utilisateur reçoit un rôle spécifique lors de sa création (par exemple, ROLE_ADMIN).
Il suffira alors d'ajouter l'attribut :

```
#[IsGranted('ROLE_ADMIN')]
```

au-dessus d'une méthode pour que la route vers celle-ci ne soit accessible qu'aux admins.

Si ce n'est pas le cas, il sera redirigé vers une page d'erreur 403 non autorisée.

Une autre façon de faire est de modifier la configuration dans le `security.yaml` dans `access_control` : { path: ^/users, roles: [ROLE_ADMIN, ROLE_SUPER_ADMIN]

Dans mon application, j'ai également mis en place des Voter afin de renforcer la sécurité sur les diverses routes de l'application.

Les Voter

Les "Voters" dans Symfony sont des composants qui aident à la prise de décision concernant l'autorisation au niveau de l'accès à certaines ressources ou certaines actions. Au lieu de simplement vérifier si un utilisateur possède un certain rôle, les "Voters" permettent une logique d'autorisation plus fine et contextuelle.

Voici comment fonctionnent les "Voters" dans Symfony :

1. **Principe de base** : Un "Voter" est appelé pour vérifier si un utilisateur donné a le droit d'exécuter une certaine action sur une certaine ressource. Par exemple, vous pourriez vouloir vérifier si un utilisateur a le droit de modifier un article de blog spécifique, et pas simplement s'il a le rôle `ROLE_EDITOR`.
2. **Deux méthodes principales** : Chaque "Voter" a principalement deux méthodes :
    - `supports()`: Cette méthode vérifie si le "Voter" supporte l'attribut et le sujet donnés. L'attribut pourrait être une action comme `view`, `edit`, etc., et le sujet est généralement l'objet sur lequel l'action doit être effectuée, comme un objet `Article`.
    - `voteOnAttribute()`: Si la méthode `supports()` renvoie `true`, cette méthode est appelée pour effectuer la logique d'autorisation réelle. Elle renvoie `true` si l'accès est accordé, et `false` sinon.
3. **Utilisation** : Pour utiliser un "Voter", vous pouvez invoquer le service `security.authorization_checker` dans un contrôleur ou un service et appeler la méthode `isGranted` avec l'attribut et le sujet appropriés. Par exemple :
   php

```
$this->denyAccessUnlessGranted('edit', $article);
```

1. **Décision** : Si vous avez plusieurs "Voters" qui supportent le même attribut et le même sujet, la décision finale est prise en fonction de la stratégie définie dans le fichier `security.yaml` sous la clé `access_decision_manager.strategy`. Les stratégies courantes sont `affirmative` (au moins un voteur vote `true`), `consensus` (la majorité des voteurs doivent être d'accord) et `unanimous` (tous les voteurs doivent voter `true`).
2. **Cas d'utilisation typiques** :
    - Vérifier si un utilisateur est le propriétaire d'une ressource avant de lui permettre de la modifier.
    - Autoriser l'accès basé sur des attributs dynamiques de l'utilisateur ou de la ressource, comme un champ `isEnabled` ou `expirationDate`.
    - Implémenter des règles d'affaires complexes qui ne se résument pas simplement à des rôles.
3. **Flexibilité** : Les "Voters" peuvent dépendre d'autres services, utiliser des requêtes de base de données, ou implémenter n'importe quelle autre logique nécessaire pour arriver à une décision. Cela les rend extrêmement flexibles et puissants.

En résumé, les "Voters" dans Symfony offrent un moyen puissant et flexible de gérer des autorisations complexes qui dépassent la simple vérification des rôles. Ils sont particulièrement utiles lorsque vous avez besoin d'une logique d'autorisation contextuelle ou basée sur des règles métier spécifiques.

Utiliser à la fois l'attribut `#[IsGranted('ROLE_ADMIN')]` et l'appel de la méthode `de voter denyAccessUnlessGranted()` dans le même contrôleur ou la même action peut sembler redondant, mais chacun a une utilité spécifique et ils peuvent être utilisés ensemble pour renforcer la sécurité dans certains scénarios. Expliquons leur utilisation individuelle et comment ils interagissent :

1. **#[IsGranted('ROLE_ADMIN')]** :
    - C'est un attribut qui sert de décorateur pour une méthode de contrôleur (ou même une classe de contrôleur).
    - Il vérifie si l'utilisateur actuellement authentifié possède le rôle spécifié (`ROLE_ADMIN` dans cet exemple).
    - Si l'utilisateur n'a pas le rôle requis, une exception `AccessDeniedException` est lancée, empêchant ainsi l'accès à cette action.
2. **$this->denyAccessUnlessGranted('edit', $article);** :
    - C'est une vérification d'autorisation au niveau du code.
    - Elle vérifie si l'utilisateur actuellement authentifié est autorisé à effectuer une certaine action (`delete` dans cet exemple) sur un objet spécifique (`$task` par cet exemple).

Est-ce que cela renforce la sécurité ? Dans des scénarios plus complexes, la combinaison des deux renforce la sécurité. Par exemple, vous pourriez vouloir que seuls les administrateurs (`ROLE_ADMIN`) puissent éditer des articles, mais seulement s'ils en sont les auteurs ou s'ils ont une certaine permission. Dans ce cas, l'attribut s'assurerait que seuls les administrateurs accèdent à l'action, tandis que le "Voter" vérifierait la logique métier supplémentaire.