Guide d’utilisation du composant de sécurité



Sommaire

1 – L’entité User

* Particularités de l’entité User

2 - Le composant de sécurité Symfony

Configuration du fichier security.yml

* Encoder
* Provider
* Firewall et Guard
* Roles hierarchy
* Access control
* Voter

2.1 - Le formulaire de connexion : login\_form

3 – Mettre en place la sécurité dans une application Symfony

* Dans les contrôleurs
* Dans une vue1 – L’entité User
* L’entité User doit obligatoirement implémenté l’interface UserInterface du composant de sécurité de Symfony car elle oblige d’inclure certaines méthodes que le firewall et le système d’authentification utiliseront pour la vérification de l’utilisateur qui tente de se connecter à votre application.

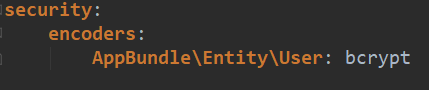
Les méthodes obligatoires : getRoles(), getPassword(), getSalt(), getUsername(), eraseCredentials().

*Réf :* [*https://symfony.com/doc/current/security.html#a-create-your-user-class*](https://symfony.com/doc/current/security.html#a-create-your-user-class)

2 - Le composant de sécurité Symfony

* **Encoder : Encodage des mots de passe**

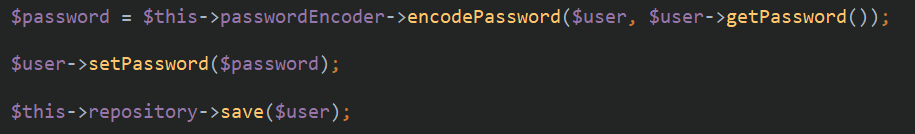
Fichier : app/config/security.yml



En ajoutant ces lignes de code Symfony c’est qu’il doit encoder les mots de passe de l’entité AppBundle\Entity\User.

Vous pouvez maintenant utilisez l’interface : UserPasswordEncoderInterface

dans vos contrôleurs ou handler pour encoder un mot passe avant sa sauvegarde en base de données.



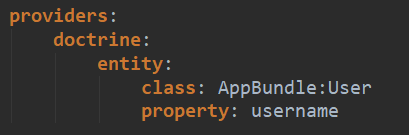
*Réf :* [*https://symfony.com/doc/current/security.html#c-encoding-passwords*](https://symfony.com/doc/current/security.html#c-encoding-passwords)

* **Provider : Indiquer à Symfony ou se trouve l’utilisateur**

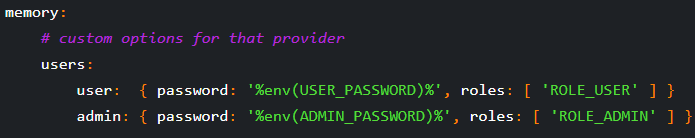
Fichier : app/config/security.yml

Il existe plusieurs types de Provider pour indiquer à Symfony ou aller chercher les utilisateurs qui pourront se connecter à l’application.

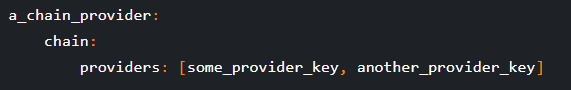
Depuis la base de données : entity



Depuis la mémoire : le nom d’utilisateur et son passe sont indiqué directement dans le fichier security.yml



Chain : Permet à Symfony de rechercher les utilisateurs depuis plusieurs Provider



*Réf :* [*https://symfony.com/doc/current/security/user\_provider.html*](https://symfony.com/doc/current/security/user_provider.html)

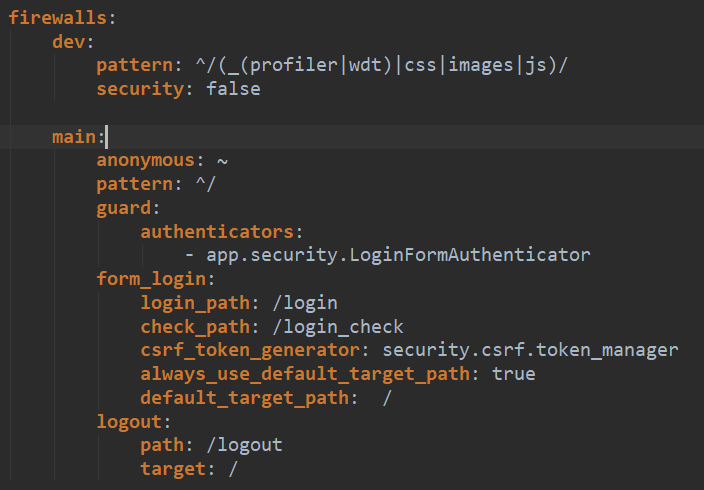
* **Firewall : La manière par laquelle les utilisateurs s’authentifient**

Fichier : app/config/security.yml

C’est via la configuration du Firewall que l’on va indiquer par quelle manière les utilisateurs vont se connecter.

Un utilisateur peut se connecter par un formulaire classique d’une page web, ou par une token API.

La figure suivante montre comment le configurer avec une authentification par formulaire.



On indique les routes à surveiller **/login** pour la connexion, **/logout** pour la déconnexion lors de l’utilisation du formulaire de connexion **form\_login**.

Le service **Guard** de Symfony ajoute une couche de sécurité supplémentaire en vérifiant en base de données : le nom d’utilisateur, l’encodage du mot de passe et le CSRF Token (protection contre les attaques d’injection de code) envoyé par le formulaire de connexion.

*(app.security.LoginFormAuthenticator est un service qui doit être déclaré dans le fichier service.yml)*

*service.PNG*

*Réf :* [*https://symfony.com/doc/current/security/form\_login.html*](https://symfony.com/doc/current/security/form_login.html)

*Réf :* [*https://symfony.com/doc/current/security.html#a-authentication-firewalls*](https://symfony.com/doc/current/security.html#a-authentication-firewalls)

* **Roles hierarchy : Héritage de rôle**

Fichier : app/config/security.yml

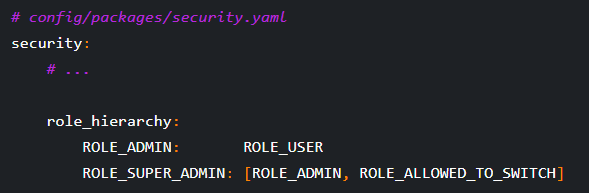
Ce paramètre dans le fichier sécurity.yml permet d’appliquer un héritage au niveau des rôles des utilisateurs.

Dans l’exemple ci-dessous l’administrateur aura les mêmes droits que l’utilisateur USER

**roles_hierarchy.PNG**

Grace au Roles hierarchy vous pouvez créer plusieurs niveau d’autorisation le super administrateur ROLE\_SUPER\_ADMIN héritera du rôle de l’administrateur ROLE\_ADMIN qui héritera du rôle de l’utilisateur ROLE\_USER.

De ce fait l’utilisateur ROLE\_USER bénéficiera du rôle ayant le plus bas niveau d’autorisation.



*Réf :* [*https://symfony.com/doc/current/security.html#hierarchical-roles*](https://symfony.com/doc/current/security.html#hierarchical-roles)

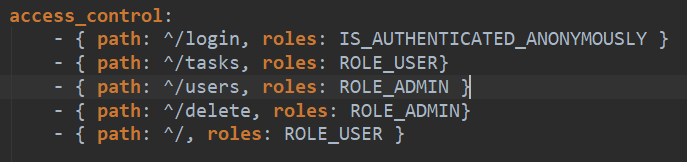
* **Access Control : Un accès aux routes suivant le rôle**

Fichier : app/config/security.yml

Avec la clé de paramètre **access\_control** il est possible d’indiquer quel utilisateur aura accès à tel ou tel route.

Vous pouvez donc facilement interdire l’accès à une route si l’utilisateur n’a pas le niveau suffisant d’autorisation.

L’exemple ci-dessous montre que pour accéder à /login (page d’authentification) l’utilisateur n’a pas besoin d’avoir un rôle particulier **IS\_AUTHENTICATED\_ANONYMOUSLY** contrairement aux routes /tasks, /users, /delete ou un rôle est obligatoire pour y accéder donc l’utilisateur doit être authentifié et autorisé par le système de sécurité de Symfony.

****

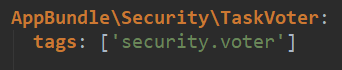
*Réf :* [*https://symfony.com/doc/current/security/access\_control.html*](https://symfony.com/doc/current/security/access_control.html)

* **Voter : Autoriser un utilisateur à effectuer une action sur un objet**

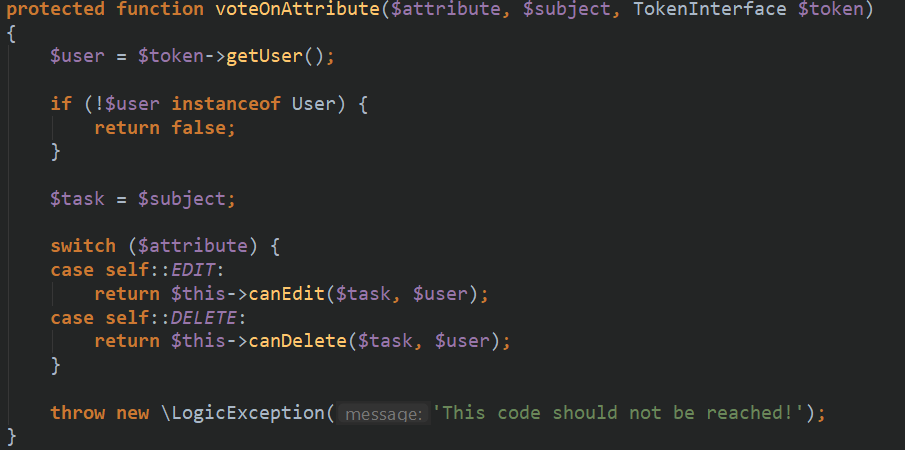
**voter1.PNG**

1 code dans le contrôleur qui vérifie si l'utilisateur connecté peut éditer une tâche

Pour mettre en place un système de Voter il faut tout d’abord créer une classe « TaskVoter » (AppBundle/security/TaskVoter). Cette classe permet de définir la logique à exécuter sur une action précise sur un objet. Cette classe doit être déclarée dans le fichier service.yml.



La classe **TaskVoter** à une méthode qui vérifie si l’utilisateur est bien une instance de la classe User et que User est bien le propriétaire de la tâche, si c’est le cas il est autorisé à édité **canEdit** ou supprimé **canDelete** une tâche.

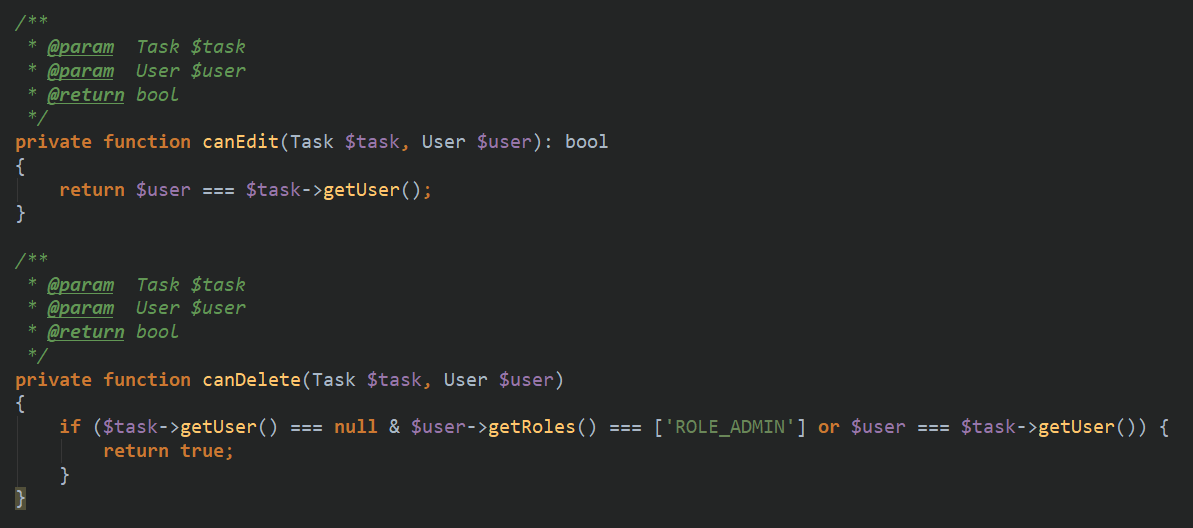


La méthode **canEdit** vérifie si l’utilisateur connecté est bien le propriétaire de la tâche.

La méthode **canDelete** a deux conditions :

Dans le cas ou la tâche n’a pas de propriétaire « **anonyme** » et que l’utilisateur connecter à un **ROLE\_ADMIN** cela l’autorisera à supprimer la tâche.

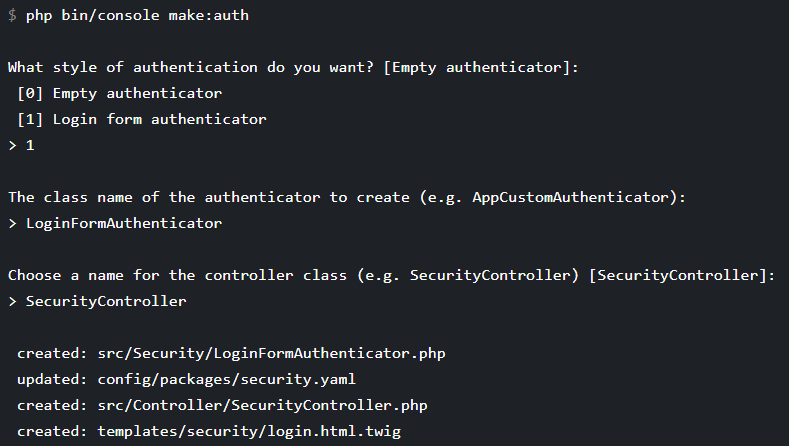
Dans le cas pu l’utilisateur connecter est bien le propriétaire de la tâche cela lui permettra de la supprimé.



*Réf :* <https://symfony.com/doc/3.4/security/voters.html>

2.1 - Le formulaire de connexion : login\_form

Mettre en place un formulaire de connexion est très simple avec grâce au composant MakerBundle. Avec quelques lignes à entrées sur votre terminal MakerBundle vous génère le formulaire et toutes les classes dont vous avez besoin pour un système d’authentification sécurisé.



La classe LoginFormAuthenticator (Guard), la mise à jour du fichier security.yml ainsi que le contrôleur et sa vue.



*Réf :* [*https://symfony.com/doc/current/security/form\_login\_setup.html*](https://symfony.com/doc/current/security/form_login_setup.html)

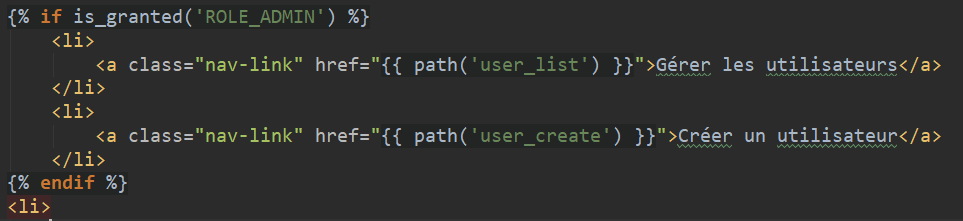
3 – Mettre en place la sécurité dans une application Symfony

Vous pouvez faire appel au composant de sécurité dans le code de votre application.

Dans vos contrôleurs ou handler pour interdire ou donner l’accès a certaines fonctions suivant le rôle de l’utilisateur, dans les vues pour afficher par exemple un formulaire, une barre de navigation ou même du texte suivant le rôle de l’utilisateur actuellement connecter.

Quelques exemples :

Dans cette vue twig nous affichons ce code seulement si l’utilisateur connecter à le rôle ROLE\_ADMIN.



Dans un contrôleur vous aurez surement besoin de récupérer l’utilisateur actuellement connecté. Pour ce faire vous pouvez utilisez la l’interface TokenStorageInterface du composant de sécurité de Symfony.

token_storage.PNG

Vous pouvez également dans un contrôleur mettre directement des restrictions sur une méthode via les annotations avec la classe Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted

