Implémentation de l'authentification

Florian Jourde

Mars 2023

Introduction

L'authentification sur un site web peut se révéler être une **faille de sécurité** majeure, lorsqu'elle n'est pas correctement réalisée, ou plus maintenue.

Une bonne pratique à adopter lorsque nous développons une application sur Symfony est de se baser sur le **système d'authentification** fourni par le framework.

Ce système est robuste, mis à jour, et permet un **hachage** des **mots de passe** sûr. L'algorithme d'**encodage** des mots de passe en **SHA256**, jusqu'alors inviolable, permet d'encoder les mots de passe, sans pouvoir les décoder, pour quiconque aurait accès à la base de données.

Contexte

Le précédent système d'authentification étant sujet à des **dépréciations**, il a été nécessaire de le retravailler de toute pièce, en se basant sur les composants disponibles aujourd'hui.

Implémentation

Le système d'authentification mis en place sur **ToDo & Co** est basé sur la documentation officielle, que vous retrouverez ici :

https://symfony.com/doc/5.4/security/passwords.html

Pour mettre en place le système d'authentification, le processus d'intégration a été basé sur la commande suivante :

```
php bin/console make:auth
```

La **CLI** de Symfony étant très puissante, rapide et simple à exécuter, une nouvelle classe intitulée « **LoginFormAuthenticator.php** » est créée dans le dossier « **src/Security** ». Ce fichier comprend toute la logique de l'authentification.

Lors de la création de l'authentification, il a été déterminé que le composant d'authentification serait utilisé sur la classe **User**, soit nos utilisateurs. D'autres systèmes d'authentification peuvent être créés à l'avenir, selon les besoins.

```
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

Dans le fichier « **config/packages/security.yaml** », vous pouvez déterminer quelle variable sera utilisée pour réaliser la connexion. Ici, c'est le champ « **email** » qui sera utilisé par l'utilisateur pour se connecter. Ce champ peut, par exemple, être remplacé par le champ « **username** ».

Enfin, pour réaliser l'**enregistrement** d'un utilisateur en base de données et lui permettre de s'authentifier, tout en ayant haché son mot de passe au préalable, vous retrouverez, ci après, une capture d'écran présentant le mode de fonctionnement d'un contrôleur.

```php
    /**
     * @Route("/admin/users/create", name="app_user_create")
     */
    public function createAction(Request $request, UserPasswordHasherInterface $userPasswordHasher, EntityManagerInterface $em)
    {
        $user = new User();
        $form = $this->createForm( type: UserType::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $user->setPassword($userPasswordHasher->hashPassword($user, $form->get('password')->getData()));

            $em->persist($user);
            $em->flush();

            $this->addFlash( type: 'success', sprintf( format: 'L\'utilisateur <strong>%s</strong> a bien été ajouté.', $user->getUserIdentifier()));

            return $this->redirectToRoute( route: '/admin/users');
        }

        return $this->render( view: 'user/create', ['form' => $form->createView()]);
    }
```

Vous pourrez retrouver ce contrôleur dans le dossier suivant : « **src/Controller/UserController.php** ».

La dépendance utilisée par Symfony pour hacher le mot de passe est nommée « **UserPasswordHasherInterface** ».