Le fonctionnement de l'authentification

Le framework Symfony propose des outils pour gérer l'authentification d'une manière sécuriser. J'y ai eu recours. Pour cela, il est nécessaire d'installer le **SecurityBundle.**

L'entité User

Symfony a besoin d'un objet User pour gérer l'authentification et la sécurité. Il s'agit d'une entité gérée par doctrine (et donc stockée en base de données) et située dans le dossier Entity. Il est nécessaire qu'elle implémente l'interface *UserInterface*. Elle a donc :

- un champ roles pour définir les permissions (ici, ROLE_USER et ROLE_ADMIN),
- la méthodes getUserIdentifier (qui retourne l'identifiant unique),
- eraseCredentials (si on stocke dans la class le mot de passe à hasher).

On étend aussi l'interface PasswordAuthenticatedUserInterface qui nécessité d'avoir la méthode getPassword et donc une propriété $password.

Le fichier *security.yaml*

La sécurité est configurée dans le fichier **security.yaml**, situé dans le dossier package, du dossier config. Pour comprendre en détail ce fichier, on peut se reporter à la documentation de Symfony : https://symfony.com/doc/current/security.html. Nous verrons ici les principaux points nécessaires pour que l'authentification de l'application fonctionne.

Sous la clé *security*, on définit la façon de hasher les mots de passe. J'ai pris celle suggéré par la documentation :

```
password_hashers:

      Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterfa
      ce: 'auto'
```

Sous la clé *providers*, on définit dans **app_user_provider** la classe utilisée pour stocker l'utilisateur en base de données : notre entité User. Nous indiquons l'identifiant unique de connexion, ici **username**.

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

On doit ensuite indiquer le firewall (le système d'authentification). Pour l'authentification des Users de l'application, tout se passe dans main. Devant provider, on indique **app_user_provider**. Devant **custom_authenticator**, on donne le nom de la classe qui va authentifier l'utilisateur et le rediriger. Si par exemple, on souhaite modifier la redirection, c'est donc dans cette classe qu'il faut aller.

```
firewalls:
        main:
            lazy: true
            provider: app_user_provider
            custom_authenticator: App\Security\AppAuthenticator
```

Devant la clé **access_control**, on définit quel rôle à accès à quoi. On a la route /login, accessible à tous, la route /users, accessible aux admins et la route /profile pour que les utilisateurs ne puissent modifier que leurs propres informations.

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

Le formulaire de connexion

Pour authentifier un utilisateur, on passe par un formulaire. Tout se passe dans le controller SecurityController. Le controller affiche le formulaire et les erreurs via AuthenticationUtils. En réalité, tout est géré par Symfony via le FormLoginAuthenticator. Il faut que la requête POST possède un champ _username et un champ _password.

Pour résumé le procédé, l'utilisateur soumet son formulaire, le système de sécurité, ici FormLoginAuthenticator, intercepte la requête, vérifie les identifiants et le connecte en cas de succès ou renvoie l'utilisateur sur le formulaire en cas d'erreur.

En tant que développeur, on peut donc être appelé à modifier le fichier security.yaml et la classe User. Dans une moindre mesure, on peut avoir besoin de modifier le fichier AppAuthenticator (par exemple pour ajouter la possibilité de rester connecter avec la case à cocher « se souvenir de moi »).