Implémentation de l'authentification

Dans le cadre de ce projet, l'implémentation de l'authentification ce fait principalement via 3 fichiers.

Configuration avec security.yml

app\config\security.yml qui est le fichier de configuration des différentes options concernant le déroulé de l'authentification. Il y différents type d'options utiliser dans le cadre de ce projet :

- **Le choix de l'encoder** qui détermine la manière dont sera encodé le mot de passe de l'utilisateur, il faut ici choisir un encodage pour une Entité. Dans ce projet c'est bcrypt qui est utilisé.

```
    encoders:
        AppBundle\Entity\User: bcrypt
```

- **La hiérarchie des rôles** qui détermine quels rôles seront disponibles dans l'application, vous pouvez également faire hériter un rôle d'un ou plusieurs autre rôles en ajoutant un tableau de rôle au rôle que vous souhaitez faire hériter. C'est notamment le cas dans ce projet de ROLE_ADMIN qui hérite de ROLE_USER, ce qui fera que les utilisateurs ayant le ROLE_ADMIN auront également les droits attaché à ROLE_USER.

```
role_hierarchy:
    ROLE_ADMIN:       [ROLE_USER]
```

- **Le provider** qui détermine d'où proviendra la liste des utilisateurs utilisés pour procéder à l'authentification et quelle propriété sera utilisée pour procéder à l'authentification. Dans le cadre de ce projet nous utilisons une entité doctrine **AppBundle:User** et sa propriété **username.**

```
providers:
    doctrine:
        entity:
            class: AppBundle:User
            property: username
```

- **Les firewalls** qui déterminent différentes sections du site qui pourront être ou non protégé par une authentification, il y est également possible de déterminer d'autres options comme par exemple les routes permettant l'authentification. Dans ce projet nous avons 2 firewalls, le premier correspond au route disponible dans le mode développement, la sécurité est désactiver pour permettre aux différents développeurs d'accéder notamment au profiler. Le second s'applique au reste de l'application, il permet une connexion anonyme nécessaire pour accéder à la page de connexion, définit les routes nécessaires à la connexion, il définit la page d'accueil comme étant la page vers laquelle est redirigé l'utilisateur une fois connecté et il permet la déconnexion.

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false

    main:
        anonymous: ~
        pattern: ^/
        form_login:
            login_path: login
            check_path: login_check
            always_use_default_target_path:  true
            default_target_path:  /
        logout: ~
```

- **Les access_control** qui déterminent quelle section du site doit être accessible à quel type d'utilisateur en fonction de l'url. Dans le cas de cette application, seul les pages relative à la connexion sont accessible à des anonyme, les pages relative à la gestion des utilisateurs ne sont accessible qu'aux administrateurs et toute les autres pages sont accessible au utilisateur connecté.

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

Stockage des utilisateurs via User.php

Dans cette application, les différents utilisateurs sont stocké grâce à l'entité **User** en base de donné. Tout d'abord pour que l'entité **User** puissent être compatible avec le système d'authentification de symfony il est nécessaire d'implémenter l'interface **UserInterface.**

```php
interface UserInterface
{
    /**
     * Returns the roles granted to the user.
     *
     * <code>
     * public function getRoles()
     * {
     *     return array('ROLE_USER');
     * }
     * </code>
     *
     * Alternatively, the roles might be stored on a ``roles`` property,
     * and populated in any number of different ways when the user object
     * is created.
     *
     * @return (Role|string)[] The user roles
     */
    public function getRoles();

    /**
     * Returns the password used to authenticate the user.
     *
     * This should be the encoded password. On authentication, a plain-text
     * password will be salted, encoded, and then compared to this value.
     *
     * @return string The password
     */
    public function getPassword();

    /**
     * Returns the salt that was originally used to encode the password.
     *
     * This can return null if the password was not encoded using a salt.
     *
     * @return string|null The salt
     */
    public function getSalt();

    /**
     * Returns the username used to authenticate the user.
     *
     * @return string The username
     */
    public function getUsername();

    /**
     * Removes sensitive data from the user.
     *
     * This is important if, at any given point, sensitive information like
     * the plain-text password is stored on this object.
     */
    public function eraseCredentials();
}
```

Pour enregistrer les différents éléments de l'utilisateur vous devez ajouter des attributs à la classe **User** et accompagner ces derniers d'annotation doctrine ou vous pourrez configurer les options du champ correspondant à l'attribut dans la base de donnée.

Classe permettant l'utilisation des annotations Doctrine : **Doctrine\ORM\Mapping**

Vous pouvez également joindre aux attributs de la classe des annotations permettant d'ajouter des contraintes liée au formulaire permettant de déclencher un message d'erreur en cas de non respects

de la contrainte lors de la validation d'un formulaire. Il est également possible d'imposer une contrainte d'unicité au formulaire sur un attribut.

Classe permettant l'ajout de contrainte aux formulaires : **Symfony\Component\Validator\Constraints.**

Classe permettant l'ajout d'une contrainte d'unicité au niveau des formulaires : **Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity**

Mise en place des routes avec SecurityController.php

L'authentification dans cette application nécessite la fourniture de quelque route :

- La route **login** qui contient le formulaire de connexion et qui sera la route vers laquelle sera renvoyé un utilisateur non connecté qui tenterait d'accéder à une page protégé.
- La route **login_check** qui sera la route qui déclenchera la vérification des identifiants et la connexion ou non de l'utilisateur.
- La route **logout** qui permettra la déconnexion de l'utilisateur.

Toutes ces routes doivent être définit dans **SecurityController** avec l'ajout d'action correspondante accompagné d'annotation de paramétrage de ces routes.

Dans le controller, les actions correspondantes aux routes **login_check** et **logout** ne contiennent rien car les opérations correspondantes à ces routes sont exécuter via des événements et des listeners natif fourni par Symfony, il est donc inutile d'ajouter du code à l'intérieur de ces actions.

Dans le cas de la route **login,** l'action correspondante fait appel au service **security.authentication_utils** qui permet d'accéder à des informations relatives à l'authentification comme le dernier nom d'utilisateur utilisé ou la listes des erreurs relative à l'authentification. L'action transmet ensuite ces informations à la vue qui affiche le formulaire connexion au site web.

Déroulement de l'authentification

Dans notre projet le processus d'authentification se déroule en plusieurs étapes :

- **Processus de connexion** :
    - **L'utilisateur entre ces identifiants**, si la forme de ces identifiants est incorrect, l'application renvoie les différentes erreurs et les affichent dans la vue correspondant au formulaire de connexion.
    - **L'application vérifie que les identifiants correspondent à un utilisateur** issu du provider défini dans la configuration (cf. security.yml), si ce n'est pas le cas l'application renvoie les différentes erreurs et les affichent dans la vue correspondant au formulaire de connexion.
    - **L'application renvoie l'utilisateur vers la page d'accueil** car dans la configuration la page d'accueil est définit comme étant la page vers laquelle sont systématiquement redirigé les utilisateurs après leur connexion.
- **Processus d'autorisation :**
    - **L'utilisateur demande à accéder à une page**

- **Le firewall contrôle si l'utilisateur peut accéder à la section du site qu'il protège**, si l'utilisateur ne peut pas accéder à cette section du site en l'état, il est redirigé sur la page de connexion
- **L'application contrôle les autorisations de l'utilisateur** et vérifie que ces autorisations correspondent aux autorisations nécessaire pour accéder à cette partie du site web, si ce n'est pas le cas une erreur 403 est renvoyé.
- **L'application renvoie la page demandé par l'utilisateur.**