TODOLIST

Authentification et utilisateurs :

L'implémentation de l'authentification a été réalisée selon les standards Symfony. Pour la résumer brièvement avant de rentrer dans le détail, l'entité User de l'application implémente une UserInterface mise à disposition nativement par Symfony. Cette interface permet l'accès à différentes méthodes de gestion des Users depuis les Controllers, ainsi que le stockage en session de différentes informations de base auxquelles on peut ajouter ce qui nous est utile.

1. Fichiers concernés et configuration

▪ AppBundle/Entity/User.php

Entité user qui implémente l'interface Symfony « UserInterface » ligne 18. C'est par cette implémentation que tout le système d'authentification devient actif, et Symfony vient utiliser l'entité User que nous avons crée pour lui donner certaines méthodes obligatoires dont il peut se servir :

- getRoles()/SetRoles() qui servent à déterminer le rôle de l'utilisateur
- getSalt() qui sert au chiffrement/déchiffrement des mots de passe
- getPassword() qui récupère le mot de passe utilisateur
- eraseCredentials() qui efface toute trace d'accès stockés en clair si on en a besoin à un moment donné du fonctionnement de l'application
- getUsername() qui sert à reconnaître un utilisateur dans le process d'authentification (on peut utiliser n'importe quel attribut de l'objet User comme un email ou un uuid, il faut simplement retourner une string présente en base de donnée).

Ces méthodes ne sont pas obligatoires, elles peuvent être laissées vide selon la configuration de l'application. En l'occurrence, getSalt() et etraseCredentials() ne servent pas.

```php
        $this->password = $password;
    }

    public function getEmail()
    {
        return $this->email;
    }

    public function setEmail($email)
    {
        $this->email = $email;
    }

    public function getRoles()
    {
        $roles = $this->roles;
        // give everyone ROLE_USER
        if (!in_array( needle: 'ROLE_USER', $roles)) {
            $roles[] = 'ROLE_USER';
        }
        return $roles;
    }

    public function setRoles($roles)
    {
        $this->roles = $roles;
    }

    public function eraseCredentials()
    {
    }
}
```

- **app/config/security.yml**

Ce fichier sert à la configuration du système d'authentification et à la sécurité (rôles, routes protégées, firewall, password encoder). Ce qui nous intéresse se trouve ligne 5 : le provider. Dans le système Symfony, ce provider, qui est un objet réalisant certaines taches à notre place, va nous fournir une entité user à utiliser pour créer des sessions PHP et remplir ces sessions avec les informations que Symfony ira chercher en base de données via son entité User (et Doctrine) qui implémente sa UserInterface. Voilà donc ce qu'on déclare dans la partie providers :

providers : (on déclare un provider)

doctrine : (on déclare à symfony d'utiliser doctrine pour aller récupérer l'entité suivante)

entity : (on veut récupérer une entité)

class : AppBundle:User (on veut utiliser l'entité User avec l'interface Symfony)

property : (On veut reconnaître un User par son attribut récupéré par getUsername(), quel qu'il soit.)

Allant de pair avec le provider, l'access_controle détermine les routes protégées ou non par une authentification. On y déclare des paths (un par ligne), ainsi que le rôle nécessaire pour accéder à la ressource. Par rôle, il faut surtout entendre « droit », car ça comprend aussi bien les roles utilisateurs que les états de connexion. Dans notre cas, seule la route /login est accessible sans authentification :

- { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }

Il ne manque qu'une partie et la configuration sera complète : le firewall. C'est lui qui va décider de rediriger vers l'authentification, et quelle route intercepter à la submission du form (voir le SecurityController pour en savoir plus sur l'interception du form).

On déclare donc un paramètre form_login qui sert à répertorier ces informations, ainsi que :

- login_path : login (« /login » sera le chemin qui servira à afficher et gérer le form login)
- login_check : login_check (« login_check » sera la route à intercepter au submit du form)
- always_use_default_target_path: true (utiliser les chemins par défaut, sdauf config particulière)
- logout : ~ (Déclare à symfony que la route /logout est standard pour qu'il intercepte la requête et traite la déconnexion par lui même)

Il y a d'autres façons de faire, notamment via le Guard qui donne la main sur chaque étape, mais la documentation Symfony ou d'autres bundles de type FOSUserBundle conseillent le form_login.

Avec cette configuration, Symfony sait maintenant quel entité aller chercher pour remplir son système d'authentification, et comment les reconnaître les unes des autres.

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

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

    role_hierarchy:
        ROLE_ADMIN: ROLE_USER

    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: ~

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

AppBundle/Controller/SecurityController

Ce controller sert spécifiquement à l'authentification, et est appelé depuis la route /login de l'application. C'est la seule route accessible sans être authentifié, comme précisé dans le security.yml

Dans le système Symfony, l'authentification peut lui être entièrement déléguée. C'est à dire que toute la gestion du form UserLogin n'est pas réalisée comme elle l'aurait été dans un Controller d'une autre entité classique. Pour faire appel à ce système, il suffit de récupérer le service « security,authentication_utils », comme fait ligne 16 dans notre cas, afin d'avoir accès à un certain nombre de méthodes nous permettan de gérer le login.

On peut ainsi récupérer les erreurs renvoyées par le form et le dernier utilisateur enregistré de la session.

A noter que dans ce cas, le FormType n'est pas obligatoire, même s'il est préférable de toujours en passer par lui pour harmoniser tous les forms de l'application, notamment en terme de style automatique si c'est en place (via bootstrap ou autre). Dans notre cas, le form est simplement fait en HTML, et envoyé sur la route login_check.

Cette route ne comporte aucun code, et de fait, Symfony va intercepter le form avant qu'il atteigne le Controller pour faire le traitement par lui même en servant des différentes informations de la configuration comme vu précédemment. C'est le même cas pour la déconnexion via logoutCheck().

Sans qu'on s'en aperçoive directement, l'authenticator Symfony communique avec le SecurityController en stockant des informations dans la session PHP. C'est pourquoi on peut récupérer les erreurs de connexion dans le service authentication_utils.

```
12          * @Route("/login", name="login")
13          */
14         public function loginAction(Request $request)
15         {
16             $authenticationUtils = $this->get('security.authentication_utils');
17
18             $error = $authenticationUtils->getLastAuthenticationError();
19             $lastUsername = $authenticationUtils->getLastUsername();
20
21             return $this->render( view: 'security/login', array(
22                 'last_username' => $lastUsername,
23                 'error'         => $error,
24             ));
25         }
26
27         /**
28          * @Route("/login_check", name="login_check")
29          * @codeCoverageIgnore
30          */
31         public function loginCheck()
32         {
33             // This code is never executed.
34         }
35
36         /**
37          * @Route("/logout", name="logout")
38          * @codeCoverageIgnore
39          */
40         public function logoutCheck()
41         {
42             // This code is never executed.
43         }
```