Documentation Technique

ToDo&Co

Sommaire

1 - Entité User

L'entité User et celle qui va s'occuper de la gestion utilisateur, sécurité, …
Cette entité doit implémenté l'interface "UserInterface" nécessaire pour créer une classe utilisateur.

L'implémentation requière d'ajouter des méthode:

- **getRoles** : Retourne les rôles de l'utilisateur.
- **GetPassword** : Récupération du mot de passe encodé.
- **GetSalt** : Retourne le sel pour l'encodage du mot de passe.
- **GetUsername** : Récupère le nom de l'utilisateur.
- **EraseCredentials** : Efface les données sensible comme le mot de passe.

2 - Composant de sécurité Guard

2.1 - Encoder

L'encodage pour les mot de passes qui sera utilisé par l'encodeur est le "Bcrypt".

app/config/security.yml

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

Grâce à l'interface "PasswordEncoderInterface", vous pouvez encodé le mot de passe.

src/Listener/EncodePasswordLister

```php
$encoded = $passwordEncoder->encodePassword($entity->getPlainPassword(), $entity->getSalt());
$entity->setPassword($encoded);
```

2.2 - Provider

Le provider permet à Symfony de savoir ou cherché l'utilisateur.

app/config/security.yml

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

2.3 - Firewalls

Le Firewall (Pare feu) est la configuration pour l'authentification.

app/config/security.yml

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

    main:
        anonymous: true
        pattern: ^/
        guard:
            authenticators:
                - user.security.login_form_authenticator
        logout:
            path: /logout
        logout_on_user_change: true
        switch_user: ~
        remember_me:
            secret: '%secret%'
            lifetime: 604800
```

Voici à quoi correspond les lignes contenu dans le "main" :

- anonymous : Les personnes qui ne sont pas connecté peuvent accéder au contenu du site.
- pattern : Les url qui doivent être protégé (^/ = Toutes les url).
- guard :
  - authenticators : La couche de sécurité guard faisant la vérification lors de l'envois du formulaire (Voir image ci-dessous).
- logout :
  - path : Url de déconnexion.
- logout_on_user_change : Permet de déconnecté l'utilisateur si celui-ci change.
- switch_user : Permet de se connecté sur un compte utilisateur sans connaître le mot de passe.
- remember_me :
  - secret : Chaine de caractère permettant la génération du token.
- lifetime : Temps avant expiration du token.

app/config/services.yml

```yaml
user.security.login_form_authenticator:
    class: AppBundle\Security\LoginFormAuthenticator
    arguments: [
        '@doctrine.orm.entity_manager' EntityManager ,
        '@form.factory' FormFactoryInterface ,
        '@security.user_password_encoder.generic' UserPasswordEncoder ,
        '@router' RouterInterface ]
    public: true
```

app/confg/parameters.yml

```
secret: tHg:k6&"YGsdtiBz
```

2.4 - Role hierarchy

La hiérarchie des rôle permet de définir quels sont les héritages des rôles.
Sur l'image ci-dessous, le rôle "ROLE_ADMIN" hérite du rôle "ROLE_USER", cela veut dire qu'il aura les même droit que le rôle "ROLE_USER" plus celui de l'administrateur.

app/config/security.yml

```
role_hierarchy:
    ROLE_ADMIN: ROLE_USER
```

2.5 - Access control

L'access control permet de configuré des routes qui seront soumis à la hiérarchie des rôles.

Dans l'image ci-dessous, toutes les routes commençant par "/login" sont accessible si l'utilisateur n'est pas connecté.
Les routes commençant par "/users" requière d'avoir le rôle "ROLE_ADMIN" pour y accéder. Si l'utilisateur n'a pas le rôle, il verra refusé l'accès.
Sur la dernière ligne, toutes les routes du site demande d'avoir le rôle "ROLE_USER" pour accéder au contenu. Si la personne n'est pas connecter, il sera automatiquement redirigé vers la page de connexion.

a*pp/config/security.yml*

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

3 - Gestion utilisateur

3.1 - Inscription

Lors de l'inscription, l'entité User est appelé (Voir la partie 1). Cette entité implémente "UserInterface" où il va appliquer ses méthodes.

```
interface UserInterface
{
    public function getRoles();

    public function getPassword();

    public function getSalt();

    public function getUsername();

    public function eraseCredentials();
}
```

3.2 - Connexion

Lors de la connexion, le composant de Symfony du nom de "Guard" est appelé pour gérer toute la partie authentification. Lors de l'envoi du formulaire de connexion, le firewalls qui lui appel Guard en demandant le service "user.security.login_form_authenticator".

app/config/security.yml

```
firewalls:
    main:
        guard:
            authenticators:
                - user.security.login_form_authenticator
```

app/config/services.yml

```
user.security.login_form_authenticator:
    class: AppBundle\Security\LoginFormAuthenticator
    arguments: [
        '@doctrine.orm.entity_manager' EntityManager ,
        '@form.factory' FormFactoryInterface ,
        '@security.user_password_encoder.generic' UserPasswordEncoder ,
        '@router' RouterInterface ]
    public: true
```

Le service appel le fichier "LoginFormAuthenticator" qui étend de "AbstractGuardAuthenticator". Dans ce fichier 7 méthodes sont demandées.

src/AppBundle/Security/LoginFormAuthenticator

```php
class LoginFormAuthenticator extends AbstractGuardAuthenticator
{
    public function start(Request $request, AuthenticationException $authException = null)
    {

    }

    public function getCredentials(Request $request)
    {

    }

    public function getUser($credentials, UserProviderInterface $userProvider)
    {

    }

    public function checkCredentials($credentials, UserInterface $user)
    {

    }

    public function onAuthenticationFailure(Request $request, AuthenticationException $exception)
    {

    }

    public function onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)
    {

    }

    public function supportsRememberMe()
    {

    }
}
```

Voici les fonctionnalités de chacune des méthodes :

- **start** : Renvois sur la page définis si l'utilisateur essaye d'accédé à une page qui demande d'être authentifié.
- **getCredentials** : Permet d'extraire le contenu des données envoyé, qui seront transmit à la méthode getUser.
- **getUser** : Récupère les données envoyé par le getCredentials.
- **checkCredentials** : Si la méthode getUser retourne un objet User, alors la méthode vérifiera les données et vérifié si celle-ci existe.
- **onAuthenticationFaillure** : Si l'authentification a echoué, alors la méthode est appelé et renvois un message d'erreur.
- **onAuthenticationSuccess** : Si l'authentification a réussi, alors la méthode est appelé.
- **supportsRememberMe** : Si la méthode retourne "True", alors le firewalls sera appelé et demandera d'ajouter le "remember_me". Le "remember_me" permet d'ajouter un token à la connexion qui aura un temps de validité (en secondes) si l'utilisateur à activé le bouton "se souvenir de moi" lors de la connexion. Si le token expire, l'utilisateur va devoir s'authentifie à nouveau.

app/config/security.yml

```yaml
remember_me:
    secret: '%secret%'
    lifetime: 604800
```