

TODO-LIST

Sécurité de l'application

Section 1 : Sécurité

1.1 Fichier de Sécurité de Symfony

1.2 L'authentification d'un utilisateur

1.3 Stockage des utilisateurs

Section 1 : Sécurité

La système d'authentification sous Symfony ou aussi appelé "Guard" est assez facile d'accès. Cependant certaines choses sont à savoir pour comprendre son fonctionnement dans notre application

1.1 Fichier de Sécurité de Symfony

La configuration de la sécurité de l'application se trouve dans le fichier "security.yml". Celui-ci est importé dans le fichier de configuration (config.yml ou se trouve notamment la configuration du framework etc...)

```
imports:
- { resource: parameters.yml }
- { resource: security.yml }
- { resource: services.yml }
```

Donc comme indiqué ci-dessus, ce fichier contient la configuration de la sécurité de l'application, à savoir:

- Les paramètres d'authentifications,
- Les paramètres d'autorisations,
- Les roles etc...

1.2 L'authentification d'un utilisateur

L'application est accessible par tous.

Ceci dit, en étant un utilisateur enregistré, plus de possibilités s'offrent à nous.

Afin d'identifier ces utilisateurs on doit les authentifier.

Donc dans ce même fichier "security.yml", voici les informations essentiel pour le paramétrage de l'authentification dans l'application:

```

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: /
    #Add logout to homepage
    logout:
      path: /logout
      target: /

```

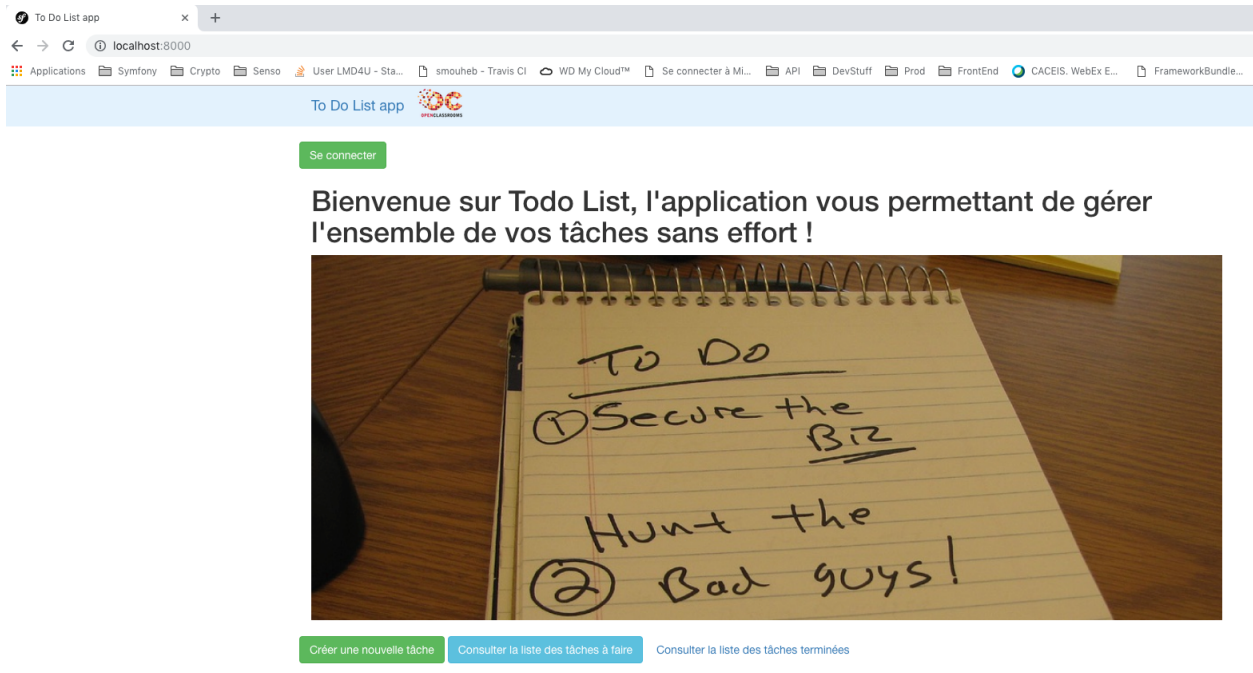
Firewalls

Le Firewall (Par feu) , est donc la porte de protection de l'application, il vérifie en gros *l'identité* de l'utilisateur. C'est ainsi que nous allons indiquer la stratégie d'authentification, ou la manière d'authentifier un utilisateur.

Sur la copie d'écran ci-dessus, nous voyons qu'il y a dans cette section "Firewalls" deux sous section "dev" et "main" (ceci est juste une convention mais on peut les nommer comme on le souhaite):

- La partie "**dev**" n'a aucun contrôle particulier (security : false)
 - Le pattern nous indique que sur l'url /_profiler (ou wdt), il n'y a aucune sécurité afin => l'outil profiler en dev est assez primordial
- La partie "**main**", prendra en charge toutes les autres url (en fait toutes les urls finalement, puisque "/" est la racine de l'application i.e. la page d'accueil)
 - Pour ce qui est de l'authentification, tous les utilisateurs (non créés) seront authentifié par défaut en tant que "anonymous" . Ce qui finalement est un utilisateur technique nous permettant d'accéder à certaine partie de l'application sans forcément être "membre" .
En allant sur la page d'accueil du site (en Dev) on peut voir dans la barre du profiler que l'utilisateur est bien 'anon' (anonyme) .
Cependant, des parties de l'applications sont protéger (et ne peuvent y accéder que les utilisateurs ayant le ou les "rôle" nécessaires, voir section

“access control”), celles ci sont répertoriés dans la section “access control” de ce même fichier de configuration.

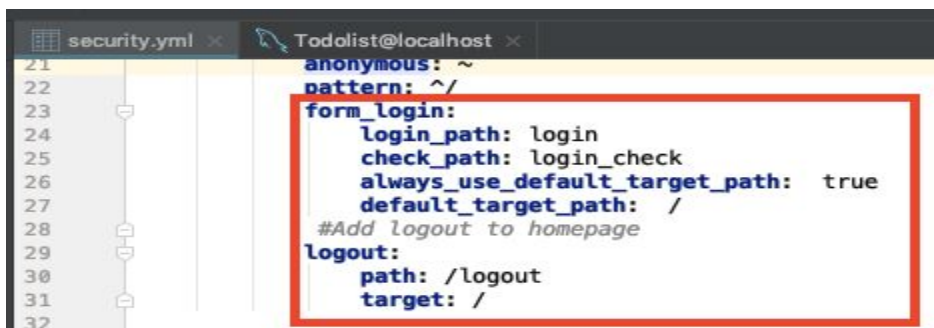


Copyright © OpenClassrooms



Lorsqu'un utilisateur essaie d'accéder à une page protégée, le système va le rediriger vers la page login comme indiqué dans le “Firewall”

Celui ci indique que l'authentification d'un utilisateur (pas anonyme car celui ci est encore une fois un utilisateur technique donné par défaut) se fera par “**form_login**”, en d'autres termes en utilisant un formulaire, dont le chemin est renseigné par le “**login_path**” => “**login**”



Pour "check_path", en indiquant la route "login_check", le framework va donc intercepter la requête et se chargera de la suite de l'authentification.

Donc une fois que l'utilisateur rentre ses identifiants (username et password), vient là le rôle de l'"**encoder**" et du "**provider**" (qui sont elles mêmes des section séparés dans le fichier de configuration, mais qui remplissent un rôle précis)

Encoders et Providers:

Sous le provider (fournisseur), on voit que c'est doctrine qui est renseigné. On indique que les utilisateurs sont en base de donnée et on y accède par l'entité "User" pour persister ou récupérer les utilisateurs en utilisant comme paramètre le "username".

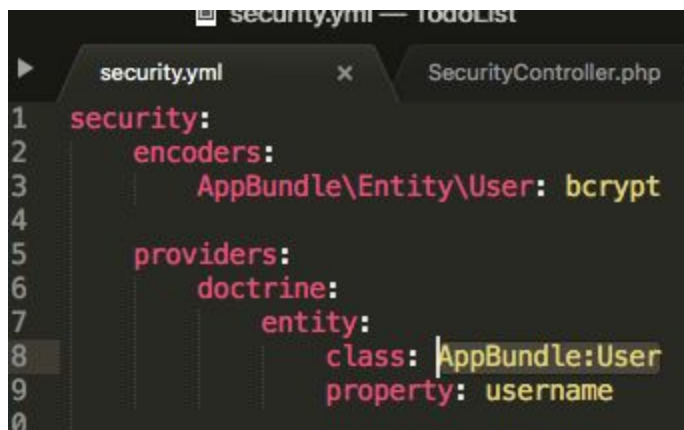
L'encoder lui indique à Symfony comment le mot de passe est "encodé" (ou comment il doit être encodé lorsque nous créons un nouvel utilisateur).

Donc dans le cas d'une authentification, une fois le formulaire soumis:

- L'encoder indique quelle encodage est fait (ici avec l'algorithme d'encryptage "bcrypt")
- Ensuite une requête est effectuée pour récupérer l'utilisateur en base de donnée

(Les utilisateurs entrent leurs username et password et le système va rechercher l'entité "User" et vérifier en base de donnée avec comme paramètre le "username").

Toutes ces informations sont renseignées dans le fichier de configuration:



```
1 security:
2   encoders:
3     AppBundle\Entity\User: bcrypt
4
5   providers:
6     doctrine:
7       entity:
8         class: AppBundle:User
9         property: username
```

Pour revenir rapidement sur la section Firewall:

Une fois l'authentification effectuée, à l'heure actuelle l'utilisateur sera redirigé vers la page d'accueil, ce qui correspond à la configuration "always_use_default_target_path" qui est à **true**.

Ceci peut être changé en mettant comme valeur **false** afin de renvoyer l'utilisateur à la dernière page ou il était (i.e. url enregistré en session) avant l'authentification (si aucune url n'était en session il sera renvoyé à la page d'accueil par défaut).

Access Control:

Cette section de la configuration, gère l'accès à certaines partie de l'application.



```
14
15 firewalls:
16   dev:
17     pattern: ^/(_(profiler|wdt)|css|images|js)/
18     security: false
19
20   main:
21     anonymous: ~
22     pattern: ^/
23     form_login:
24       login_path: login
25       check_path: login_check
26       always_use_default_target_path: true
27       default_target_path: /
28       #Add logout to homepage
29     logout:
30       path: /logout
31       target: /
32
33   access_control:
34     - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
35     - { path: ^/users, roles: ROLE_ADMIN }
```

Cela se decompose en deux:

- “path “ => Cette partie précise ll'url à protéger
- “rôles” => Cette partie indique qu'elle rôle doit avoir l'utilisateur pour y accéder

Ces rôles sont définis d'une part dans la section “role_hierarchy”, dont le rôle est (comme son nom l'indique) => établir la hiérarchie des rôles dans l'application (plus de détail à la section “role_hierarchy”).

Cependant, cette section “access_control” vient en complément des contrôles établis au niveau des “Controller” de l'application.

Certaines urls sont ici sécurisées, cependant, des actions dans des parties de l'applications sont elles protégées par les controllers qui s'assurent que le rôle alloué à l'utilisateur en

question lui permet en effet de faire l'action demandé ex:

```
//Test whether the user has the right role to delete that task
if($this->get('security.authorization_checker')->isGranted('ROLE_ADMIN'))
{
    $em = $this->getDoctrine()->getManager();
    $em->remove($task);
    $em->flush();

    $this->addFlash('type: 'success', 'message: 'La tâche a bien été supprimée.');
```

Dans la copie d'écran ci-dessus, la condition test si l'utilisateur a le rôle "Admin" lui permettant de supprimer la tâche.

Note: il y a plusieurs possibilité de sécuriser l'application, soit par l'access_control, les controllers, twig, les annotations etc... et tous ceux là sont aussi bien complémentaires.

Exemple d'un utilisateur autorisé et d'un autre dont l'accès est interdit:

Si je suis authentifié avec un utilisateur ayant les droit nécessaire dans ce cas j'y ai accès "granted"

The screenshot shows the Symfony Profiler interface. The left sidebar contains various tool categories like Request / Response, Performance, Forms, Exception, Logs, Events, Routing, Security, Twig, Doctrine, E-Mails, Debug, and Configuration. The main content area displays the security authorization details for the user 'Smael'.

User Information:

- Username: Smael
- Authenticated: Yes (indicated by a green checkmark)

Property Value Table:

Property	Value
Roles	[ROLE_ADMIN, ROLE_USER]
Inherited Roles	[ROLE_USER]
Token class	Symfony\Component\Security\Core\Authentication\Token\UsernamePasswordToken

Security Voters (3):

affirmative

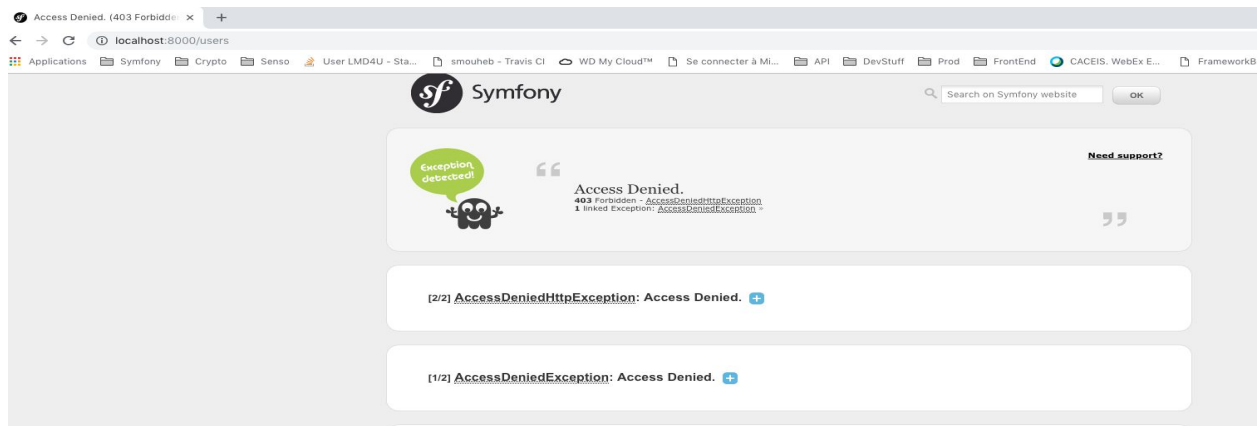
Strategy

#	Voter class
1	Symfony\Component\Security\Core\Authorization\Voter\RoleHierarchyVoter
2	Symfony\Component\Security\Core\Authorization\Voter\ExpressionVoter
3	Symfony\Component\Security\Core\Authorization\Voter\AuthenticatedVoter

Access decision log:

#	Result	Attributes	Object
1	GRANTED	ROLE_ADMIN	Symfony\Component\HttpFoundation\Request (GET /users HTTP/1.1 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8 Accept-Encoding: gzip, deflate, br Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7 Connection: keep-alive Cookie: Phpstorm-dcda2eeb=c9b0bf3c-27c0-49b9-839e-4321d40891a0; Phpstorm-dcda32aa=69b60b05-fa03-4411-9792-2d3ce945307e; PHPSESSID=7ksm8jseigtqactvd30tievr Host: localhost:8000 Purpose: prefetch Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 X-Php-Ob-Level: 1)

Sinon l'accès est refusé (erreur http 403=> forbidden):



Rôle en base donnée:

Les rôles associés à chaque utilisateur sont persister dans la table “user” column “rôle”

	id	username	password	email	role
1	1	Smael	\$2y\$13\$0XnkTu4qwkf0gvBG1/QjJeeKANntbWcjuNtDUaRcT.4tA8wHfD48u	smael.mouheb@test.com	["ROLE_ADMIN"]
2	2	Anonymous			
3	3	Yo	\$2y\$13\$f5kg5BE0H5DcMKnB0wyZx0dYHnSZL0dsVqu2SATk6m9sKMDSIwq/i	test@test.com	["ROLE_USER"]
4	4	testadmin	\$2y\$13\$0wUheJOhy3L3sQD1DQT0ue6n9BS9dqLtPazCsQ08VPayPgLKaes.	test@tt.com	
5	6	Smaeluser	\$2y\$13\$kG1BASXRGZqfL8WDWYQ7NuMF2L0qnldbsi9vxAujrV.CtNuc4531q	user@user.com	["ROLE_USER"]
6	7	Test1	\$2y\$13\$1GubYLv0IwHbWBvNUTUT.LDbM0VaJW2srN8Ea3Pg7nZW.9HUncfq	test1@test.com	["ROLE_USER"]
7	10	Smaeladmin	\$2y\$13\$qe0SDNKaAKPwexe2Kmsfuy/yRCekH5TUdud0yUJXvyfM.Pg.eRS	smaeladmin@admin.com	["ROLE_ADMIN"]

Role_hierarchy:

Cette section permet simplement d'établir une hiérarchie entre les roles/ utilisateurs, Ici en l'occurrence on voit que :

- Un role admin “ROLE_ADMIN” contient le role user “ROLE_USER” ce qui veut dire que tout ce que l'utilisateur aura droit de faire, l'admin le pourra aussi.
- Idem pour le role “ROLE_SUPER_ADMIN”, tout ce que l'admin (ainsi que le ROLE_USER)pourra faire le super admin le pourra aussi

```
role_hierarchy:
  ROLE_ADMIN: ROLE_USER
  ROLE_SUPER_ADMIN: ROLE_ADMIN
```

Ici, seul la hiérarchie est ainsi spécifiée, ensuite ces rôles sont à définir dans les “controller”, ou bien les acces_control etc... (comme expliqué à la section “Access Control”).

Note: le nom des rôles est totalement laissé libre, seule obligation, commencé par "ROLE_" ensuite peu importe le nom qu'on renseigne, le tout est d'être cohérent avec ce que l'on veut en faire.

1.3 Stockage des utilisateurs

Comme indiqué à la section précédente, les utilisateurs sont stockés en base de données dans la table "user".

On a comme couche de persistance => Doctrine

On l'utilise, pour le CRUD de notre application et plus spécifiquement ici, pour notre entité utilisateur nous permettant d'authentifier nos utilisateurs.