



Simon GAUTIER



Sécurité

Gestion de l'authentification

Gestion des autorisations

Généralités sur la sécurité

- Sécurité : élément crucial dans une application web
- Symfony fournit tous les outils pour sécuriser une application
 - Authentification : permet de savoir qui est connecté
 - Autorisation : permet de savoir si on a le droit d'effectuer telle ou telle action
- Prérequis : installer le recipe associé : **\$ composer req security**
- Configuration de la sécurité : **config/packages/security.yaml**

Authentication – Exemple simple d'authentification HTTP basique

```
security:
  # Les encoders fournissent les stratégies d'encodage des mots de passe
  encoders:
    # Encoder pour les mots de passe pour chaque type d'utilisateur
    Symfony\Component\Security\Core\User\User: plaintext # aucun encodage

  # Les providers décrivent comment récupérer ou créer des utilisateurs
  providers:
    my_provider_in_memory: # Nom arbitraire pour le provider
      memory: # Le provider est en mémoire
        users:
          john: # john est le login
            password: doe # doe est le mot de passe
            # rôle : nom arbitraire qui commence par convention par "ROLE_"
            # un utilisateur peut avoir plusieurs rôles
            roles: ROLE_USER

  # Les firewalls définissent le mécanisme d'authentification utilisé dans chaque partie de l'application
  firewalls:
    # L'ordre des firewalls est important : le premier pour lequel l'URL courante correspond est pris en compte
    dev: # Chaque firewall a un nom arbitraire
      pattern: ^/(_(profiler|wdt)|css|images|js)/
      security: false # Toutes les URLs qui matchent la regexp ne sont pas sécurisées
    main:
      pattern: ^/
      http_basic: ~ # Toutes les URLs qui matchent la regexp sont sécurisées avec authentification basique HTTP

  # access_control : définit les permissions nécessaires pour accéder à chaque partie de l'application
  # Bonne pratique : à utiliser pour des patterns globaux. ex : ^/admin, ...
  access_control:
    - { path: "^/admin", roles: ROLE_ADMIN } # Pour tout l'admin du site, le rôle ROLE_ADMIN est requis
```

Authentication requise
http://127.0.0.1:8000

Nom d'utilisateur

Mot de passe

Authentication

Fonctionnement des encoders

- Chaque encoder doit implémenter 2 méthodes :
 - **encodePassword()** : calcule la valeur encodée du mot de passe à partir de sa valeur en clair

```
// $user est un objet qui implémente UserInterface ==> vu plus loin dans ce chapitre

$encoder = $this->get('security.password_encoder');
$password = $encoder->encodePassword(
    $user, 'mon_mot_de_passe'
);
$user->setPassword($password);
```

- **isPasswordValid()** : permet de vérifier si un mot de passe en clair correspond avec une valeur encodée

```
$encoder = $this->get('security.password_encoder');
$result = $encoder->isPasswordValid(
    $user, 'mon_mot_de_passe'
);
```

Authentication

Fonctionnement des encoders

- security.yaml : définir pour chaque « type » d'utilisateur défini dans l'application la stratégie d'encodage des mots de passe
- Liste des encodages supportés :

Encoder	Type d'encodage
plaintext	Aucun encodage. ATTENTION : non sécurisé ! À utiliser uniquement pour les tests.
bcrypt	Encodage Bcrypt : recommandé avant Symfony 5.
argon2i	Algorithme Argon2 (alternative à bcrypt) → https://fr.wikipedia.org/wiki/Argon2
pbkdf2	Encodage PBKDF2 : https://fr.wikipedia.org/wiki/PBKDF2
auto	Depuis Symfony 5, méthode recommandée !

Authentication

Fonctionnement des encoders

- Possibilité de définir des options pour chaque encoder
- Cf. doc pour détails :

symfony.com/doc/current/reference/configuration/security.html

```
security:
  encoders:
    Symfony\Component\Security\Core\User\User: plaintext

    App\Entity\User1:
      algorithm: bcrypt
      cost: 13

    App\Entity\User2:
      algorithm: plaintext
      ignore_case: false
```

Authentication

Fonctionnement des encoders

- Générer un mot de passe encodé avec le bon encoder depuis la console :
php bin/console security:encode-password
- Puis utilisation de la valeur encodée :

```
security:
  encoders:
    Symfony\Component\Security\Core\User\User: bcrypt

  providers:
    my_provider_in_memory:
      memory:
        users:
          john:
            # Ci-dessous le mot de passe "doe" encodé en bcrypt depuis la console :
            password: $2y$13$ON1MvgusDUCElRuyjcHyV./PkctzJ8aUcIyI1WAvTRer2hRHR7RhK
            # ...
```

Authentication

Fonctionnement des providers

- Objectif d'un provider : créer / récupérer des utilisateurs
 - Providers fournis par Symfony en standard :
 - entity : récupération des utilisateurs depuis une entité Doctrine
 - memory : récupération des utilisateurs depuis le fichier de configuration (cf. exemple)
 - ldap : récupération des utilisateurs depuis un serveur LDAP
- Exemples : symfony.com/doc/current/reference/configuration/security.html
- Possibilité de créer d'autres providers ➔ symfony.com/doc/current/security/user_provider.html#creating-a-custom-user-provider


Authentication

Créer sa propre classe d'utilisateurs

- Elle DOIT implémenter l'interface suivante :

\Symfony\Component\Security\Core\User\UserInterface

```
interface UserInterface
{
    public function getRoles();
    public function getPassword();
    public function getSalt();
    public function getUsername();
    public function eraseCredentials();
}
```



- Exemple d'implémentation (utilisateurs utilisés par le provider « memory ») :

\Symfony\Component\Security\Core\User\User

Authentication

Fonctionnement des firewalls

- Un firewall détermine si l'utilisateur a besoin d'être authentifié sur telle ou telle URL et si oui selon quelle stratégie, notamment :
 - Quel provider ?
 - Quel moyen pour s'authentifier ?
 - Quel moyen pour se déconnecter ?
- Exemple présenté : authentication HTTP standard ➡ dans cette partie, présentation d'une authentication via un formulaire HTML

Authentification

Fonctionnement des firewalls

- Liste des authentifications supportées par les firewalls :
 - Cf. détails dans la configuration :
symfony.com/doc/current/reference/configuration/security.html#firewalls

Mode d'authentification	Détails
http_basic	Authentification HTTP standard, mot de passe en clair. Utilisé dans l'exemple précédent.
http_basic_ldap	Idem, mais en contrôlant le mot de passe dans un serveur LDAP.
http_digest	Authentification HTTP standard, mot de passe hashé.
x509	Authentification des utilisateurs avec certificat X.509.
form_login	Authentification via formulaire.
form_login_ldap	Idem, mais en contrôlant le mot de passe dans un serveur LDAP.
remote_user	Authentification via Apache.

Authentication

Firewall « **form_login** »

Etape préliminaire : s'assurer que les sessions sont actives
(dans **config/packages/framework.yaml**) :

```
framework:
  # ...
  session: # La ligne ne doit pas être commentée
```

```
security:
  # ...
  firewalls:
    dev:
      pattern: ^/(_(profiler|wdt)|css|images|js)/
      security: false
    main:
      pattern: ^/
      form_login:
        # Mettre un path ou un nom de route (nom de route plus "stable" car
        # on peut changer le path sans que ça n'impacte le nom de route)
        # La route n'a pas besoin de réellement exister car Symfony va catcher l'évènement
        check_path: app_login_check

        # Route vers laquelle l'utilisateur est redirigé lorsqu'on doit lui demander de s'authentifier
        login_path: app_login

        # Sans la ligne suivante, un utilisateur anonyme ne pourra pas s'authentifier => boucle de redirection infinie
        anonymous: ~
```

```

<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class SecurityController extends Controller
{
    /**
     * @Route("login", name="app_login")
     *
     * @return Response
     */
    public function loginAction(Request $request, AuthenticationUtils $authUtils)
    {
        // Récupération des erreurs s'il y en a eu lors de la précédente authentification
        $error = $authUtils->getLastAuthenticationError();

        // Login précédemment saisi par l'utilisateur
        $lastUsername = $authUtils->getLastUsername();

        return $this->render('security/login.html.twig', array(
            'last_username' => $lastUsername,
            'error'         => $error,
        ));
    }

    /**
     * @Route("login_check", name="app_login_check")
     * On déclare une route mais l'action ne sera jamais exécutée car Symfony catche l'évènement
     *
     * @return Response
     * @throws \Exception
     */
    public function loginCheckAction()
    {
        // Non censé entrer ici car la route doit être catchée par Symfony ==> L'exception joue un rôle de garde fou
        throw new \Exception('Unexpected loginCheck action');
    }
}

```

Authentication

Firewall « **form_login** » - Template login

```
{% extends 'base.html.twig' %}

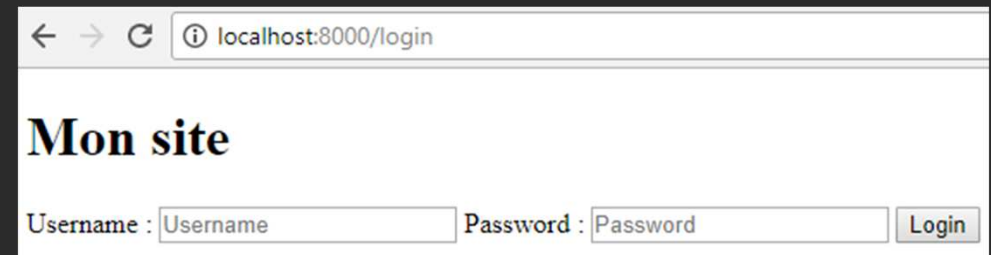
{% block body %}
    {% if error %}
        <div>{{ error.messageKey|trans(error.messageData, 'security') }}</div>
    {% endif %}

    <form action="{{ path('app_login_check') }}" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="_username" value="{{ last_username }}" placeholder="{{ 'security.username'|trans }}" />

        <label for="password">Password:</label>
        <input type="password" id="password" name="_password" placeholder="{{ 'security.password'|trans }}" />

        {#
            If you want to control the URL the user
            is redirected to on success (more details below)
        <input type="hidden" name="_target_path" value="/account" />
        #}

        <button type="submit">{{ 'security.validate'|trans }}</button>
    </form>
{% endblock %}
```



← → ↻ ⓘ localhost:8000/login

Mon site

Username : Password :

Authentication

Firewall « **form_login** » - Déconnexion

```
main:
  pattern: ^/
  form_login:
    check_path: app_login_check
    login_path: app_login

  logout:
    # Route utilisée pour déconnecter un utilisateur
    # La route n'a pas besoin de réellement exister car Symfony va catcher l'évènement
    path: app_logout

    target: app_login # route où l'utilisateur sera redirigé après déconnexion

  anonymous: ~
```

```
class SecurityController extends Controller
{
    // ...

    /**
     * @Route("logout", name="app_logout")
     *
     * @return Response
     * @throws \Exception
     */
    public function logoutAction()
    {
        // Non censé entrer ici car la route doit être catchée par Symfony ==> L'exception joue un rôle de garde fou
        throw new \Exception('Unexpected logout action');
    }
}
```

Authentication

Comment accéder à l'utilisateur connecté ?

- Dans un contrôleur qui hérite du contrôleur de base Symfony :

```
dump($this->getUser());
```
- Dans un contrôleur avec injection de dépendance de l'objet **Symfony\Component\Security\Core\Security** :

```
dump($security->getUser());
```
- Dans un template Twig :

```
{{ dump(app.user) }}
```
- Synthèse concernant l'authentification :
 - Provider : permet de créer et récupérer des utilisateurs
 - Encoder : permet de fournir une stratégie d'encodage des mots de passe
 - Firewall : permet de dire à Symfony pour quelles URLs il faut être authentifié
- Etape suivante : l'autorisation permet de savoir si un utilisateur a le droit d'accéder à une URL en particulier

Autorisations – Les rôles


- Comme présenté dans l'exemple, un rôle est une simple chaîne de caractères qui par convention commence par « **ROLE_** »
 - Seuls les rôles nommés « **ROLE_** » sont automatiquement pris en compte par Symfony
- Héritage possible entre rôles. Exemple de rôles :

```
security:
  role_hierarchy:
    ROLE_AUTHOR: ROLE_USER # Un utilisateur avec le rôle ROLE_AUTHOR aura à minima les droits d'un ROLE_USER
    ROLE_ADMIN:  ROLE_USER # idem pour un utilisateur ROLE_ADMIN
    ROLE_OWNER:  [ROLE_ADMIN, ROLE_MANAGER] # un utilisateur ROLE_OWNER aura à minima les droits de 2 rôles
```

- Gestion des autorisations :
 - Dans **security.yaml** : restrictions en fonction d'URLs → **access_control**
 - Avec les annotations (sur une action ou sur toutes les actions d'un contrôleur)
 - Dans le code (un contrôleur, Twig, ...)
 - Restrictions fines via les voters (+ complexe à mettre en place)

Autorisations – Les rôles

- Assignment des rôles à un utilisateur :
 - Le user doit retourner un tableau de string ou de **\Symfony\Component\Security\Core\Role\Role**
 - Cf. commentaires dans l'interface **UserInterface** :



```
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();

    public function getPassword();
    public function getSalt();
    public function getUsername();
    public function eraseCredentials();
}
```

Autorisations – Via **security.yaml**

- Pour des parties complètes de l'application (recommandé) :

```
security:
  # ...
  access_control:
    - { path: "^/customer", roles: [ROLE_USER, ROLE_AUTHOR] } # Pour tout le compte client, le rôle ROLE_USER ou ROLE_AUTHOR est requis
    - { path: "^/admin", roles: ROLE_ADMIN } # Pour tout l'admin, le rôle ROLE_ADMIN est requis
```

- Options avancées, exemple :

```
security:
  # ...
  access_control:
    - { path: ^/admin, roles: ROLE_USER_IP, ip: 127.0.0.1 }
    - { path: ^/admin, roles: ROLE_USER_HOST, host: symfony\.com$ }
    - { path: ^/admin, roles: ROLE_USER_METHOD, methods: [POST, PUT] }
    - { path: ^/admin, roles: ROLE_USER }
```

- Cf. doc : symfony.com/doc/current/security/access_control.html

Autorisations – Avec les annotations

- Protection d'une action : 2 méthodes identiques :
 - Utiliser de préférence l'annotation « IsGranted »

```
/**
 * @Route("/", name="home")
 * @IsGranted("ROLE_USER")
 */
public function indexAction()
{
    // ...
}
```

```
/**
 * @Route("/", name="home")
 * @Security("has_role('ROLE_USER')")
 */
public function indexAction()
{
    // ...
}
```

- Protection de toutes les actions d'un contrôleur : même principe :
 - Les 2 méthodes sont cumulables

```
/**
 * Class DefaultController
 * @IsGranted("ROLE_USER")
 * @package App\Controller
 */
class DefaultController extends Controller
{
    // ...
}
```



Si on accède à une action non autorisée, une erreur 403 (Forbidden) est générée

Autorisations – Dans une action

- Méthode **isGranted()** accessible à la fois depuis le contrôleur ou depuis l'objet **Symfony\Component\Security\Core\Security**

```
public function indexAction(Security $security)
{
    // ...
    if (!$security->isGranted('ROLE_USER')) {
        // ...
    }
    // ...
    return $this->render('hello_world.html.twig');
}
```

```
public function indexAction()
{
    if (!$this->isGranted('ROLE_ADMIN')) {
        throw new AccessDeniedException("Vous ne passerez pas !");
    }
    return $this->render('hello_world.html.twig');
}
```

- Méthode **denyAccessUnlessGranted()** :

```
public function indexAction()
{
    $this->denyAccessUnlessGranted('ROLE_ADMIN', null, "Vous ne passerez pas !");

    return $this->render('hello_world.html.twig');
}
```

Écriture
équivalente



Autorisations – Dans un template Twig

Affiché pour tous

```
{% if is_granted('ROLE_ADMIN') %}  
    Affiché seulement pour les utilisateurs qui ont le rôle ROLE_ADMIN  
{% endif %}
```

Affiché pour tous

Autorisations – Se connecter en tant que

- Il est possible de donner le droit à certains utilisateurs de se connecter en tant qu'un utilisateur quelconque
 - À réserver uniquement pour les administrateurs !

```
firewalls:  
  # ...  
  main:  
    pattern: ^/  
    # Permet d'autoriser le switch d'utilisateurs avec ce firewall  
    switch_user: ~
```

```
providers:  
  my_provider_in_memory:  
    memory:  
      users:  
        # L'utilisateur aura le droit de se faire passer pour un utilisateur uniquement  
        # si le rôle ROLE_ALLOWED_TO_SWITCH lui est affecté  
        one_to_rule_them_all:  
          password: $2y$13$YqlF3/liC8ukwP1dcZNkNesmNUfDEmqYZoHrM3MXj4MFhcvUVF3fW  
          roles: [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
```

Autorisations – Se connecter en tant que

- Se connecter avec l'utilisateur qui a les droits
- Ajouter à l'url le paramètre spécial « `_switch_user` ». Exemple :
http://localhost:8000/?_switch_user=john
➔ On se retrouve connecté en tant que john
- Pour sortir de ce contexte, utiliser la valeur « `_exit` » :
http://localhost:8000/?_switch_user=exit
➔ On se retrouve connecté avec l'utilisateur initial

TP Sécurité



- Mettre en place une sécurité simple :
 - Utilisateurs en mémoire
 - Encodage des mots de passe en auto
 - 2 rôles : ROLE_ADMIN et ROLE_USER, ROLE_ADMIN héritant de ROLE_USER
 - Identification via formulaire, gestion de la déconnexion. Au niveau du template Twig général, afficher le lien de connexion si aucun utilisateur n'est connecté, afficher sinon le nom de l'utilisateur connecté + lien de déconnexion
- Créer 2 utilisateurs : un disposant du rôle ROLE_ADMIN, l'autre disposant du rôle ROLE_USER
- Protéger les actions du contrôleur :
 - Les actions d'ajout / édition / suppression sont autorisées pour un utilisateur ayant le rôle ROLE_ADMIN, le reste par un utilisateur ayant le rôle ROLE_USER → attention à la liste des derniers articles affichée dans le menu (risque de redirection infinie !)