

# Authentication Documentation technique



## Informations

Nom du projet	ToDo List
Type de document	Documentation technique
Date	23/12/2019
Version	1.0
Auteur	Fabien Stenneler

## Révisions

Version	Date	Auteur	Description
1.0	23/12/2019	Fabien Stenneler	Rédaction du support initial

## Table des matières

<b>Objet du document</b>	<b>6</b>
<b>Technologies utilisées</b>	<b>7</b>
Extensions PHP requises	7
Librairies Symfony utilisées	7
Gestionnaire de dépendances	8
<b>Fonctionnement de l'application</b>	<b>9</b>
Description	9
Architecture	9
Configuration globale	9
Structure MVC	9
Formulaires	10
Authentification	11
Fichiers publics	11
Configuration	11
Tests	12
Fixtures	12
Annotations	13
Contrôleurs	13
Exemple	13
Entités	13
Exemple	14
Modèle de données	15

Console de Symfony	15
Lancement du serveur de développement	15
Création / mise à jour d'une entité	15
Création des fichiers de migration	15
Mise à jour de la structure de la base de données selon les migrations	15
Création d'un formulaire	16
Réinitialisation du cache	16
Création des fixtures	16
Lancement des tests	16
Consultation du conteneur de services	16
Mises à jour	16
Mise à jour de Symfony et de ses dépendances	16
Installation d'une nouvelle librairie	16
<b>Authentification</b>	<b>18</b>
Fichiers concernés	18
Configuration	18
Cryptage des mots de passe	18
Passphrase	19
Provider	19
Pare-feu	20
Contrôle des accès	21
Stockage des utilisateurs	21
Processus d'authentification	22
Affichage du formulaire	22
Soumission du formulaire	24



## **1. Objet du document**

Ce document est la document technique officielle de l'application ToDo List de la société ToDo & Co, à destination des collaborateurs du projet.

Il présente l'architecture technique globale de de l'application, et de façon détaillée le processus d'authentification.

## 2. Technologies utilisées

Le projet est basé sur le framework PHP Symfony 5.

Documentation officielle : <https://symfony.com/doc/>

La version de PHP requise est la version 7.1.3 ou supérieure.

### Extensions PHP requises

- Ctype
- iconv
- JSON
- PCRE
- Session
- SimpleXML
- Tokenizer
- Xdebug

### Librairies Symfony utilisées

- twig/twig
- twig/extra-bundle
- doctrine/doctrine-bundle
- symfony/web-profiler-bundle
- symfony/monolog-bundle
- symfony/debug-bundle
- doctrine/doctrine-migrations-bundle
- sensio/framework-extra-bundle
- symfony/security-bundle
- symfony/maker-bundle
- symfony/web-server-bundle
- doctrine/doctrine-fixtures-bundle
- phpunit/phpunit
- fzaninotto/faker

## **Gestionnaire de dépendances**

Les dépendances sont gérées par Composer.

Documentation officielle : <https://getcomposer.org/doc/>



## 3. Fonctionnement de l'application

### 3.1. Description

L'application a pour objet la gestion des tâches d'un ensemble d'utilisateurs.

Celles-ci sont présentées sous forme de pense-bête, et comportent :

- un titre
- un auteur
- une date de création
- une description
- un statut : tâche réalisée ou à effectuer

Les utilisateurs sont gérés par un administrateur, et peuvent ajouter, modifier, supprimer des tâches ou changer leur statut.

### 3.2. Architecture

#### 3.2.1. Configuration globale

La configuration globale du projet est définie dans le fichier **.env** à la racine du projet. Ce fichier contient notamment :

- la variable définissant la bascule du passage de développement en production
- les paramètres de connexion au serveur de base de données
- la passphrase permettant le cryptage des mots de passe
- les paramètres du serveur d'envoi d'emails

#### 3.2.2. Structure MVC

L'application est conçue selon le pattern MVC.

Catégorie	Fichier	Description
Modèle	src/Entity/User.php	Entité utilisateur

	src/Entity/Task.php	Entité tâche
	src/Repository/TaskRepository.php	Dépôt tâche
<b>Contrôleurs</b>	src/Controller/DefaultController.php	Contrôleur page d'accueil
	src/Controller/SecurityController.php	Contrôleur authentification
	src/Controller/UserController.php	Contrôleur gestion des utilisateurs
	src/Controller/TaskController.php	Contrôleur gestion des tâches
<b>Vues</b>	templates/default/*	Template page d'accueil
	templates/security/*	Template authentification
	templates/task/*	Template gestion des tâches
	templates/user/*	Template gestion des utilisateurs
	templates/_navbar.html.twig	Template barre de navigation
	templates/base.html.twig	Template de base (head, body)

### 3.2.3. Formulaires

Les formulaires définissent les champs à afficher dans les vues.

Catégorie	Fichier	Description
<b>Utilisateurs</b>	src/Form/UserType.php	Formulaire ajout/modification d'un utilisateur
<b>Tâches</b>	src/Form/TaskType.php	Formulaire ajout/modification d'une

		tâche
--	--	-------

### 3.2.4. Authentification

Catégorie	Fichier	Description
Authentification	src/Security/AppAuthenticator.php	Méthodes du processus d'authentification de l'application

### 3.2.5. Fichiers publics

Les fichiers accessibles depuis un navigateur sont stockés dans le dossier public.

Catégorie	Fichier	Description
Racine	public/index.php	Routeur principal
	public/robots.txt	Instructions pour les robots d'exploration
	public/site.webmanifest	Instructions à destination du navigateur
Styles	public/css/*	Feuilles de style CSS, librairies CSS des framework JQuery et Bootstrap
Javascripts	public/js/*	Scripts Javascripts, librairies JS des framework JQuery et Bootstrap
Polices	public/fonts/*	Polices utilisées dans le thème
Images	public/img/*	Images utilisées dans le thème

### 3.2.6. Configuration

Les fichiers de configuration de Symfony sont stockés dans le dossier config.

Catégorie	Fichier	Description
Routes	config/routes.yaml	Configuration du routeur principal

principales		
Services	config/services.yaml	Configuration des services de l'application
Routes secondaires	config/routes/*	Configuration des routes des librairies
Librairies	config/package/*	Configuration des librairies

### 3.2.7. Tests

Les tests automatisés unitaires et fonctionnels sont stockés dans le dossier tests.  
Les résultats de la couverture des tests est stockée dans le dossiers test-coverage.

Catégorie	Fichier	Description
Tests unitaires	tests/Unit/Controller/*	Tests unitaires des contrôleurs
	tests/Unit/Entity/*	Tests unitaires des entités
	tests/Unit/Form/*	Tests unitaires des formulaires
Tests fonctionnels	tests/Functionnal/*	Tests fonctionnels de l'application
Configuration	phpunit.xml.dist	Configuration de la librairie PHP Unit

### 3.2.8. Fixtures

Les fixtures sont stockées dans le dossier src/DataFixtures.

Catégorie	Fichier	Description
Fixtures globales	src/DataFixtures/AppFixtures.php	Fixtures User et Task

### 3.3. Annotations

#### 3.3.1. Contrôleurs

La configuration des routes des contrôleurs est réalisée dans les annotations des méthodes des contrôleurs concernés.

#### Exemple

```
// src/Controller/TaskController.php
// ...

class TaskController extends AbstractController
{
    /**
     * @Route(
     *     "/tasks/{filter}",
     *     name="task_list",
     *     defaults={"filter" = null},
     *     requirements={"filter"="^(to-do|done)$"}
     * )
     */
    public function listAction($filter)
    {
        // ...
    }
}
```

#### 3.3.2. Entités

La configuration des champs utilisés par l'ORM est effectuée en annotations des attributs des entités concernées.

## Exemple

```
// src/Entity/Task.php
// ...

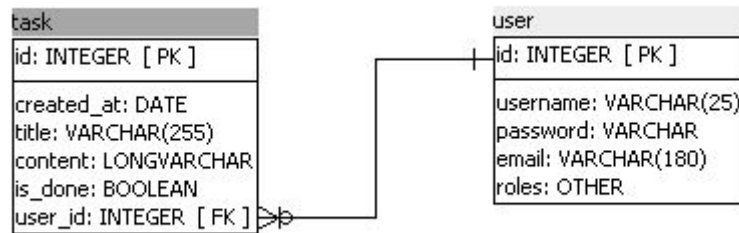
/**
 * @ORM\Entity
 * @ORM\Table
 * @ORM\Entity(repositoryClass="App\Repository\TaskRepository")
 */
class Task
{
    /**
     * @ORM\Column(type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @ORM\Column(type="datetime")
     */
    private $createdAt;

    // ...

}
```

### 3.4. Modèle de données



### 3.5. Console de Symfony

La gestion de l'application est facilitée par l'utilisation des commandes de la console du framework Symfony. Ci-dessous un récapitulatif des commandes les plus utilisées.

Les commandes doivent être lancées dans un interpréteur de commandes depuis la racine du projet.

#### Lancement du serveur de développement

```
$ php bin/console server:run
```

#### Création / mise à jour d'une entité

```
$ php bin/console make:entity
```

#### Création des fichiers de migration

```
$ php bin/console make:migrations
```

#### Mise à jour de la structure de la base de données selon les migrations

```
$ php bin/console doctrine:migrations:migrate
```

### Création d'un formulaire

```
$ php bin/console make:form
```

### Réinitialisation du cache

```
$ php bin/console cache:clear
```

### Création des fixtures

```
$ php bin/console doctrine:fixtures:load
```

### Lancement des tests

```
$ php bin/phpunit
```

### Consultation du conteneur de services

```
$ php bin/console debug:container
```

## 3.6. Mises à jour

Les mises à jour de Symfony et des librairie se fait par l'intermédiaire du gestionnaire de dépendances Composer.

### Mise à jour de Symfony et de ses dépendances

```
$ composer update
```

### Installation d'une nouvelle librairie

La liste officielle des librairies est consultable sur le site <https://packagist.org/>.



```
$ composer require [vendeur/librairie]
```

## 4. Authentification

### 4.1. Fichiers concernés

Le processus d'authentification est réalisé grâce au composant de sécurité du framework Symfony.

Documentation officielle : <https://symfony.com/doc/current/security.html>

Catégorie	Fichier	Description
Configuration	config/packages/security.yaml	Configuration du processus d'authentification
Entité	src/Entity/User.php	Entité utilisateur
Contrôleur	src/Controller/SecurityController.php	Contrôleur connexion / déconnexion
Authentification	src/Security/AppAuthenticator.php	Méthodes du processus d'authentification de l'application
Vue	templates/security/login.html.twig	Template du formulaire de connexion

### 4.2. Configuration

La configuration du composant de sécurité s'effectue dans le fichier :

config/packages/security.yaml

#### 4.2.1. Cryptage des mots de passe

Les mots de passe sont cryptés par le composant de sécurité de Symfony. Afin de modifier l'algorithme de cryptage, il faut modifier la section **encoders** du fichier de configuration.

```
// config/packages/security.yaml
security:
    encoders :
        App\Entity\User:
            algorithm: auto
```

## Passphrase

La passphrase permettant le cryptage des mots de passe est définie dans le fichier **.env** situé à la racine du projet (clé APP\_SECRET).

**Attention :** le contenu de ce fichier doit rester strictement confidentiel. La passphrase doit être **impérativement** modifiée lors de la mise en production de l'application.

### 4.2.2. Provider

Il est possible de configurer plusieurs fournisseurs d'authentification. L'entité User étant utilisée ici, elle est configurée dans la section **providers**. On définit également l'attribut de la classe User qui sera utilisé comme identifiant de connexion.

```
// config/packages/security.yaml
security:
    // ...
    providers:
        in_memory: { memory: null }
        app_user_provider:
            entity:
                class: App\Entity\User
                property: userName
```

### 4.2.3. Pare-feu

La section **firewalls** permet de définir les parties de l'application doivent être gérées par le composant de sécurité (**pattern**). Elle définit également :

- la classe chargée de réaliser l'authentification (**guard** : **authenticators**)
- le nom du provider utilisé (**provider**)
- la route permettant la déconnexion (**logout**)

```
// config/packages/security.yaml
security:
    // ...
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            pattern: ^/
            provider: app_user_provider
            anonymous: true
            guard:
                authenticators:
                    - App\Security\AppAuthenticator
            logout:
                path: app_logout
                target: homepage
```

**Attention** : le pare-feu indique quelle partie du site doit être gérée par le composant de sécurité, mais ne protège pas l'application. Les zones à protéger sont configurées dans la partie **access\_control** (voir ci-dessous).

### 4.2.4. Contrôle des accès

La partie **access\_control** permet de définir quelles parties de l'application doivent être protégées par le processus d'authentification. Elle permet également de définir les rôles autorisés pour chaque partie.

```
// config/packages/security.yaml
security:
    // ...
    access_control:
        - { path: ^/login$, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/users, roles: [ ROLE_ADMIN ] }
        - { path: ^/, roles: [ ROLE_USER, ROLE_ADMIN ] }
```

### 4.3. Stockage des utilisateurs

Les utilisateurs sont stockés dans la table **user**.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/> 1	<b>id</b> 🔑	int(11)			Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/> 2	<b>username</b> 🔑	varchar(25)	utf8_unicode_ci		Non	Aucun(e)		
<input type="checkbox"/> 3	<b>password</b>	varchar(255)	utf8_unicode_ci		Non	Aucun(e)		
<input type="checkbox"/> 4	<b>email</b> 🔑	varchar(180)	utf8_unicode_ci		Non	Aucun(e)		
<input type="checkbox"/> 5	<b>roles</b>	json			Non	Aucun(e)		

Les champs username et email sont des champs uniques.

L'accès aux données se fait via l'instanciation d'un objet de la classe **src/Entity/User.php** grâce à l'utilisation de la librairie Doctrine.

Exemple :

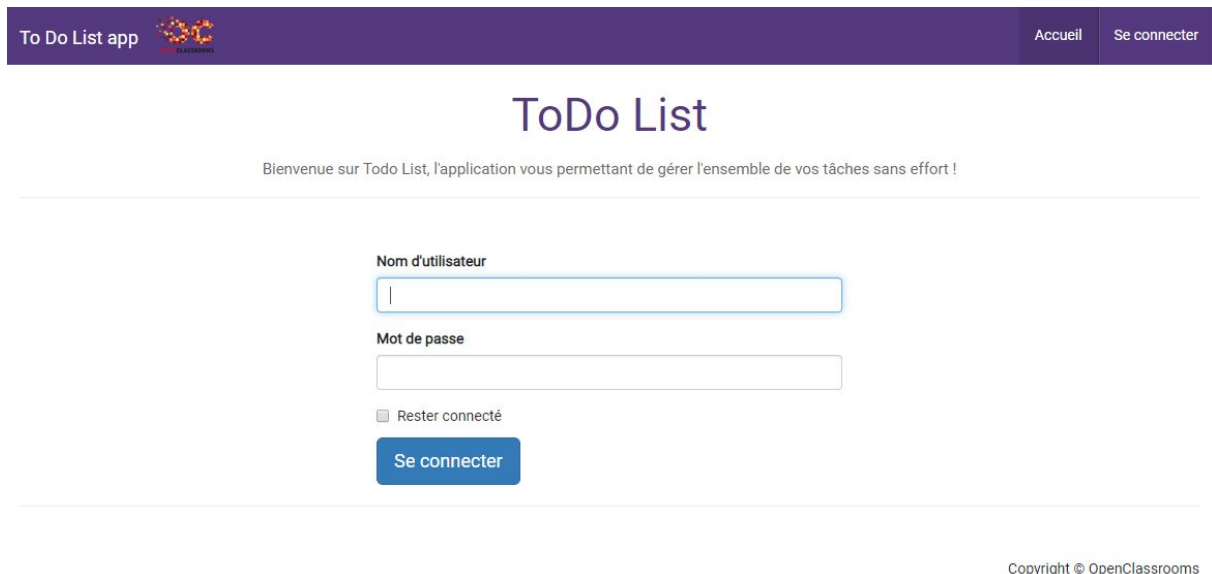
```
$user = $this->getDoctrine()
    ->getRepository(User::class)
    ->findOneBy(['username' => 'test']);
```

### 4.4. Processus d'authentification

L'authentification est effectuée par le composant de sécurité de Symfony par l'intermédiaire de la classe **src/Security/AppAuthenticator.php**.

#### 4.4.1. Affichage du formulaire

L'accès au formulaire se fait par l'url **/login**.



Copyright © OpenClassrooms

La route correspondant à cette url est configurée dans l'annotation de la méthode `login()` du contrôleur **src/Controller/SecurityController.php**.

```
// src/Controller/SecurityController.php
// ...

class SecurityController extends AbstractController
{
    /**
     * @Route("/login", name="app_login")
     */
    public function login(AuthenticationUtils $authenticationUtils):
    Response
    {
        if ($this->getUser()) {
            return $this->redirectToRoute('homepage');
        }

        // get the login error if there is one
        $error = $authenticationUtils->getLastAuthenticationError();
```

```
// last username entered by the user
$lastUsername = $authenticationUtils->getLastUsername();

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

Le contrôleur a pour rôle de lancer la création de la vue grâce au template Twig **templates/security/login.html.twig**.

```
// templates/security/login.html.twig
// ...

{% if error %}
    <div class="alert alert-danger" role="alert">{{
error.messageKey|trans(error.messageData, 'security') }}</div>
{% endif %}

<form method="post" class="login-form">

    <div class="form-group">
        <label for="inputUsername">Nom d'utilisateur</label>
        <input type="text" value="{{ last_username }}" name="username"
id="inputUsername" class="form-control" required autofocus>
    </div>

    <div class="form-group">
        <label for="inputPassword">Mot de passe</label>
        <input type="password" name="password" id="inputPassword"
class="form-control" required>
    </div>
```

```
<div class="checkbox mb-3">
  <label>
    <input type="checkbox" name="_remember_me"> Rester connecté
  </label>
</div>

<input type="hidden" name="_csrf_token" value="{{
csrf_token('authenticate') }}">

<button class="btn btn-lg btn-primary py-3 px-4" type="submit">Se
connecter</button>

</form>
```

#### 4.4.2. Soumission du formulaire

##### Étape 1 - Soumission

L'utilisateur remplit le formulaire et le valide.

## ToDo List

Bienvenue sur Todo List, l'application vous permettant de gérer l'ensemble de vos tâches sans effort !

---

Nom d'utilisateur

Mot de passe

☐ Rester connecté

Se connecter

---



## Étape 2 - Interception

La soumission du formulaire est interceptée automatiquement par le composant de sécurité de Symfony et traitée par la classe **src/Security/AppAuthenticator.php**, définie en tant que classe d'authentification du firewall.

```
// src/Security/AppAuthenticator.php
// ...
use
Symfony\Component\Security\Guard\Authenticator\AbstractFormLoginAuthenticator;

class AppAuthenticator extends AbstractFormLoginAuthenticator
{
    // ...
    public function supports(Request $request)
    {
        return 'app_login' === $request->attributes->get('_route')
            && $request->isMethod('POST');
    }
}
```

## Étape 3 - Récupération des champs du formulaire

Une fois la soumission interceptée, les données envoyées sont stockées dans un tableau **\$credentials** et la valeur du champ username est conservée en variable de session.

```
// src/Security/AppAuthenticator.php
// ...
class AppAuthenticator extends AbstractFormLoginAuthenticator
{
    // ...
    public function getCredentials(Request $request)
    {
        $credentials = [
            'username' => $request->request->get('username'),
            'password' => $request->request->get('password'),
        ];
    }
}
```

```
        'csrf_token' => $request->request->get('_csrf_token'),
    ];
    $request->getSession()->set(
        Security::LAST_USERNAME,
        $credentials['username']
    );

    return $credentials;
}
}
```

#### Étape 4 - Chargement de l'utilisateur

Après vérification de la validité du token CSRF, l'utilisateur correspondant à la valeur du champ username envoyée va être recherché dans la base de données par l'utilisation de l'entité User et de la librairie Doctrine.

Si l'utilisateur est trouvé, il est stocké dans un objet **\$user**. Sinon une exception est levée.

```
// src/Security/AppAuthenticator.php
// ...
class AppAuthenticator extends AbstractFormLoginAuthenticator
{
    // ...
    public function getUser($credentials, UserProviderInterface
    $userProvider)
    {
        $token = new CsrfToken('authenticate', $credentials['csrf_token']);
        if (!$this->csrfTokenManager->isTokenValid($token)) {
            throw new InvalidCsrfTokenException();
        }

        $user =
        $this->entityManager->getRepository(User::class)->findOneBy(['username' =>
        $credentials['username']]);
    }
}
```

```
        if (!$user) {
            // fail authentication with a custom error
            throw new CustomUserMessageAuthenticationException('Username
could not be found.');
```

```
        }

        return $user;
    }
}
```

## Étape 5 - Vérification du mot de passe

Le mot de passe crypté envoyé est testé pour vérifier sa correspondance avec celui de l'utilisateur chargé.

```
// src/Security/AppAuthenticator.php
// ...
class AppAuthenticator extends AbstractFormLoginAuthenticator
{
    // ...
    public function checkCredentials($credentials, UserInterface $user)
    {
        return $this->passwordEncoder->isPasswordValid($user,
$credentials['password']);
    }
}
```

## Étape 6 - Connexion et redirection

En cas de succès l'utilisateur est connecté et une redirection est effectuée :

- soit vers la dernière url visitée si elle existe
- soit vers la liste des tâches

```
// src/Security/AppAuthenticator.php
// ...
class AppAuthenticator extends AbstractFormLoginAuthenticator
{
    // ...
    public function onAuthenticationSuccess(Request $request, TokenInterface
$token, $providerKey)
    {
        if ($targetPath = $this->getTargetPath($request->getSession(),
$providerKey)) {
            return new RedirectResponse($targetPath);
        }

        return new
RedirectResponse($this->urlGenerator->generate('task_list'));
    }
}
```

En cas d'échec la méthode login() du contrôleur est appelée à nouveau, et le formulaire est rechargé. L'erreur est affichée.

## 4.5. Gestion des rôles

Les rôles sont définis dans le champ rôles de la table user. Ils sont accessible après avoir instancié l'entité User par la méthode **getRoles()**.

Pour autoriser ou refuser l'accès à une route ou une fonctionnalité, deux solutions principales sont possibles.

### Annotations

```
// src/Controller/TaskController.php
// ...
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
```

```
class TaskController extends AbstractController
{
    // ...
    /**
     * @Route("/users", name="user_list")
     * @IsGranted("ROLE_ADMIN")
     *
     * or use @Security for more flexibility:
     *
     * @Security("is_granted('ROLE_ADMIN') and
is_granted('ROLE_FRIENDLY_USER')")
     */
    public function listAction()
    {
        return $this->render('user/list.html.twig', ['users' =>
$this->getDoctrine()->getRepository(User::class)->findAll()]);
    }
}
```

### Méthode denyAccessUnlessGranted()

```
// src/Controller/TaskController.php
// ...
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;

class TaskController extends AbstractController
{
    // ...
    /**
     * @Route("/users", name="user_list")
     */
    public function listAction()
    {
        $this->denyAccessUnlessGranted('ROLE_ADMIN');
        return $this->render('user/list.html.twig', ['users' =>
$this->getDoctrine()->getRepository(User::class)->findAll()]);
    }
}
```

```
}  
}
```