Documentation technique Application To-Do-List

ToDo & Co

Sommaire

Introduction

L'application ToDoList suit la structure MVC (Model Vue Contrôleur) propre aux projets web PHP/Symfony4. Une connaissance des bases de ce framework est donc nécessaire pour appréhender le fonctionnement de l'application. (La documentation officiel du framework : https://symfony.com/doc/current/)

1 Fonctionnement de l'authentification via *Symfony/Security.*

Pour plus d'information vous pouvez consulter la documentation officielle : https://symfony.com/doc/current/reference/configuration/security.html

1.1 Le fichier *security.yml*

Ce fichier placé dans le dossier *./config/* gère les droits d'accès grâce à un système de Token liée à la table '*User'* de la base de données. Par défault un visiteur non identifié à un Token '*anonymous'*.

1.1.1 Partie 'encoder'

Spécifie le cryptage employé pour le mot de passe (ici *bcrypt*).
'AppBundle\Entity\User' spécifie que c'est sur le mot de passe de la table 'User' qu'est utilisé le cryptage.
Les formats possibles nativement sont bcrypt, sha512, 'pbkdf2', argon2i (pour php7.2 uniquement) ou plaintext. 'plaintext' signifie que le code est stocké dans la base de données sans cryptage.

1.1.2 Partie 'providers'

Indique que c'est l'élément *'username'* de l'entité '*AppBundle:User'* qui identifie l'utilisateur (indique son nom/login).
'*memory'* permet de déclarer des utilisateurs sans ajouter d'entré dans une base de données. Dans l'exemple on déclare un utilisateur nommé Admin, dont le mot de passe est MDP et dont les rôles/droits sont ROLE_ADMIN et ROLE_USER.

1.1.3 Partie 'firewalls'

Cette partie détermine quels chemins nécessitent des droits spécifiques pour y accédé via des règles au nom précisées au début (dev, main). Si plusieurs règles peuvent s'appliquer, seule la première règle est prise en compte.
'pattern :' indique le ou les chemins qui sont concernés par cette règle.
'security: false' signifie qu'aucune restriction n'est imposé.
'form_login:' spécifie les chemins des pages pour l'identification :

- *'login_path:'* Chemin vers le formulaire identifient/mot de passe.
- *'check_path:'* Chemin vers la validation des données d'identification.

'logout: ' spécifie le chemin du logout. 'logout: ~' si le logout n'a pas à effectuer d'action spécial.

1.1.4 Partie 'access_control'

Cette partie détermine quels chemins nécessitent des rôles spécifique pour y accédé. La même règle de priorité que pour les 'firewalls' est appliquée.
'path :' indique le ou les chemins qui sont concernés par cette règle.
'roles :' indique le ou les rôles qui sont concernés par cette règle.
Pour plus d'informations sur le fonctionnement de *'access_control'* :
https://symfony.com/doc/current/security/access_control.html.

1.2 Assembler les différents paramètres

1.2.1 Lien avec les contrôleurs

Pour interdire l'accès depuis le contrôleur, if faut faire appelle à la fonction denyAccessUnlessGranted(). Il est cependant recommander de ne pas utiliser cette méthode affin d'éviter toute confusion avec les droit d'accès déclaré dans le fichier *'security.yml'*.

Exemple :
```
$this->denyAccessUnlessGranted('ROLE_ADMIN', null, 'Unable to access this page!');
```

1.2.2 Les éléments *'Twig'*

Twig permet une mise en page facile des pages HTML via l'utilisation de patrons (Template). Les templates Twig sont disponible dans le dossier *templates/* . Les fichiers ce présentent sous la forme de code HTML mais avec des éléments script propre à Twig. Pour plus de détaille sur la composition des templates Twig : https://twig.symfony.com/doc/2.x/.
Il est possible d'accéder au rôle de l'utilisateur qui visite une page via la fonction Twig *is_granted().*

Exemple :
```
{% if is_granted('ROLE_ADMIN') %} ... {% endif %}
```

2 Gestion des '*User*' dans la base de donnée.

2.1 L'entité User et Doctrine.

Doctrine est un ORM (Object-relational mapping) qui reproduit la structure de la class User pour crée la table du même nom dans la base de donnée. (Pour en savoir plus : https://symfony.com/doc/current/doctrine.html).
L'entité User doit respecter certaines conventions affin de permettre le cryptage du code tel que déclaré dans la partie *'encoder'* du fichier *'security.yml'*. Les fonctions getSalt() et eraseCredentials() doivent être déclarées de plus la variable du mot de passe doit s'appeler 'password' et le login de l'utilisateur doit correspondre au nom déclaré dans la partie *'provider'* du fichier *'security.yml'* (*username*).

2.2 Appel à User via le contrôleur.

Les contrôleurs ce trouve dans le dossier *'src/controller'*.
Les routes *'/logout'*(déconnection) et *'/login_check'*(validation de la requête de connexion) sont prise en charge par security mais pas *'/login'* qui est déclaré dans le contrôleur *'SecurityController.php*'.
Les opérations sur la table User (création/suppression/édition/lister) sont groupées dans le contrôleur *'UserController.php*'. C'est lord des créations/éditions que les mots de passe doivent êtres codés via la fonction *'password_encoder'* de security.

Exemple :

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

3 Utilisation des tests automatisés.

3.1 Lancer des tests et tests automatique.

3.1.1 PHPUnit.

Les fichiers de tests sont dans le dossier 'tests/' et utilise la librairie PHPUnit. Pour lancer les tests il suffie de taper la commande *'vendor/bin/phpunit --env=test'* le paramètre *'env=test'* est très important car il évite que le test soit effectué avec la base de donnée de l'application. Sous un environement windows il faut préférer la commande *'vendor/bin/phpunit.bat --env=test'*.
Il est possible de n'effectué qu'une partie des tests grâce au paramètre *'--filter=nom_de_la_fonction_de_test_choisie'*.
Pour plus de détaille sur PHPUnit : https://phpunit.readthedocs.io/fr/latest/.

3.1.2 Travis.

Travis est un service d'intégration continue. Il récupère les changements du code à chaque push sur github et teste automatiquement via PHPUnit ces changements. Les résultats sont accessibles sur le site https://travis-ci.org/ ou via une bannière placée dans le fichier README.md du projet. Les réglages du logiciel Travis sont localisés dans le fichier *'.travis.yml'* à la racine du projet. (Pour plus d'information : https://docs.travis-ci.com/user/getting-started/).

3.2 Ajouté des tests.

3.2.1 Tests fonctionnel.

Les tests fonctionnels sont utilisés pour tester les fonctions des contrôleurs et sont placés dans le dossier *'tests/Controller/'*. Ces testes simulent des connections au serveur et vérifie que les retours correspondent à ceux spécifiés.
Chaque contrôleur a son fichier de test associé et chaque fonction sa fonction de test. Le contrôleur 'UserController.php' par exemple, est testé par 'UserControllerTest.php' et sa fonction 'listAction()' y est testée par la fonction 'testListAction()'.
Les différentes connections de tests sont créés à l'initialisation par la fonction 'setUp()'. Les tests utilise la fonction 'request()' pour envoyer des requêtes puis l'élément 'crawler' pour lire/évaluer la réponse. Enfin les fonctions d'assertions (tel que *assertSame*) permette de contrôler la validité de la réponse. (Voir https://phpunit.readthedocs.io/fr/latest/).

Exemple :

```
protected $client;
protected $admin;

/*  Cette fonction est exécutée avant les autres et initialise les connexions.  */
public function setUp()
{
    /*  Login à une connexion avec des droits User.  */
    $this->client=static::createClient(array(), array('PHP_AUTH_USER' => 'Test',
        'PHP_AUTH_PW'   => 'Test'));
    /*  Login à une connexion avec des droits Admin.   */

    $this->admin=static::createClient(array(), array('PHP_AUTH_USER' => 'Admin',
        'PHP_AUTH_PW'   => 'Admin'));

}

/*  Cette fonction teste la fonction listAction() du contrôleur UserController.php.  */
public function testListAction()
{
    /*  On fait une requête à l'adresse '/users' avec les deux connections.  */
    $crawler = $this->admin->request('GET', '/users');
    $this->client->request('GET', '/users');

    /*  On verifie que Les ROLE_USER n'ont pas aces à la liste des utilisateurs.  */
    $this->assertEquals(200, $this->client->getResponse()->getStatusCode());
    /*  On verifie que Les ROLE_ADMIN ont aces à la liste des utilisateurs.  */
    $this->assertSame("Liste des utilisateurs", $crawler->filter('h1')->first()->text());
}
```

La mise en place des données de test ce fait via la fonction *'setUpBeforeClass()'* présente dans le fichier *'LoginControllerTest.php'* et qui est appelé par PHPUnit au début des tests.

3.2.2 Tests unitaires.

Les tests unitaires ce concentrent sur le test des fonctions indépendamment du code d'où elles proviennent. Chaque fonctions doit respecter ces spécifications d'entré et de sortie.

Exemple :

```php
/*  Cette fonction teste les fonctions setEmail() et getEmail() de l'entitée User.php.  */
public function testEmail()
{
    $user = new User();

    $user->setEmail("Email");
    $this->assertSame("Email", $user->getEmail());
}
```

Les tests unitaires sont utilisés pour tester indépendamment toute les class des fichiers source à l'exclusion du fichier '*AppLoader.php*' qui permet de créer des '*fixture*' et des contrôleurs qui sont testés par les tests fonctionnels.

3.2.3 Environnement de test.

Les tests on lieu dans un environnement séparé (base de donnée todolist_test au lieu de todolist). La configuration de cet environnement ce fait via des fichiers YAML dans le dossier 'config\packages\test\' et vien suplenté les configurations de base présentes dans le dossier 'config\packages\'. (https://symfony.com/doc/current/configuration.html).
Cette base de donnée de test est automatiquement générée par la fonction *setUpBeforeClass()* du fichier *'LoginControllerTest.php'* avent les tests PHPUnit. Si vous effectuez un test unique sans inclure le fichier *'LoginControllerTest.php',* il faut alors initialiser manuellement la base de donnée de test grâce au commandes suivantes :

```
>php bin/console doctrine:database:drop --env=test --force
>php bin/console doctrine:database:create --env=test
>php bin/console doctrine:schema:create --env=test
>php bin/console doctrine:fixtures:load -n --env=test
>
```