Documentation Technique

Cette documentation se destine aux prochains développeurs juniors qui rejoindront l'équipe.
Cette documentation a pour but de comprendre :

- Comment s'opère l'authentification ?
- Où sont stockés les utilisateurs ?
- Quel(s) fichier(s) il faut modifier et pourquoi ?

Table des matières

Introduction

L'application a été mise à jour vers Symfony 3.4, qui est toujours maintenu. Le choix de ne pas migrer vers la version 4.4 a été fait pour garantir une certaine stabilité dans le code déjà existant.

I. Comment s'opère l'authentification ?

1. Choix d'authentification

La gestion de l'authentification se fait directement dans les controllers de l'applications :

- DefaultController
- SecurityController
- TaskController
- UserController

Cela aurait pu être fait directement dans le fichier de configuration « security.yml » mais cela redirige simplement sans afficher de message clair à l'utilisateur.

2. La méthode créée pour l'authentification

Afin d'informer facilement le client qui est redirigé, un premier contrôle est fait directement dans les méthodes des controllers via la méthode privée « checkConnection » pour les controllers :

- TaskController (115)
- UserController (175)

En revanche pour Default Controller (15), comme il n'y a qu'une méthode cela est fait directement dans celle-ci.

Le but ici est de tester si le client qui essaye d'accéder à une page est authentifié. Cela se fait à travers la méthode « isGranted » :

```
• EXEMPLE: $this->get('security.authorization_checker')-
  >isGranted('IS_AUTHENTICATED_FULLY')
```

Cette méthode renvoie « true » si le client est authentifié ou « false » si le client n'est pas authentifié.

3. Le message d'erreur renvoyé

Si le client essaye de se connecter sans être authentifié, le message d'erreur renvoyé est situé en haut des controllers dans une variable privée « error », pour faciliter l'édition :

- TaskController (13)
- UserController (15)

4. Où s'effectuent les contrôles d'authentification

Les authentifications sont controllées pour les fonctions suivante :

- UserController :
  - listAction (23)
  - editAction (67)
- TaskController :
  - listAction (21)
  - listDoneAction (34)
  - createAction (47)
  - editAction (79)
  - toggleTaskAction (115)
  - toggleCancelTaskAction (133)
  - deleteTaskAction (151)

II. Où sont stockés les utilisateurs ?

Le stockage des utilisateurs se fait en base de données (BDD).

L'application ayant des tests, il est nécessaire de créer 2 BDD.

- BDD pour l'application
- BDD pour les tests

Attention, si aucune BDD n'est créée avant d'effectuer les tests, le lancement des tests écrasera la BDD existante de l'application.

1. Paramétrage BDD application

Pour paramétrer la BDD, il faut :

- Se rendre sur le fichier : App/config/parameters.yml
- Modifier les données :
  - database_host : localhost
  - database_port : 8000
  - database_name : ToDoList
  - database_user : root
  - database_password : password

2. Paramétrage BDD TEST

Pour paramétrer la BDD, il faut :

- Se rendre sur le fichier : App/config/parameters_test.yml
- Modifier les données :
  - database_host : localhost
  - database_port : 8000
  - database_name : ToDoList_test

- database_user : root
- database_password : password

III. Comprendre quel(s) fichier(s) il faut modifier et pourquoi ?

Les fichiers modifiables peuvent se diviser en 4 catégories :

- De Fonctionnement (Controllers, Entity & Form)
- De Configuration (Dossier config)
- De Ressources (Views, CSS & JS)
- De Test

1. Les fichiers de fonctionnements (Controllers, Entity & Form)

Ces fichiers sont les principaux fichiers qui vont permettre de :

- Créer, modifier ou supprimer des utilisateurs ou des tâches
- Controller les autorisations d'accès

Tous ces fichiers sont situés dans un même dossier :

- src/AppBundle/

Dans le dossiers **Controllers** on va trouver :

- DefaultController qui va contrôler les accès pour afficher la page d'accueil
- SecurityController qui va contrôler la partie authentification (login / logout)
- TaskController qui va contrôler toutes les méthodes liés à la création, modification et suppression d'une tâche
- UserController qui va contrôler toutes les méthodes liés à la création, modification et suppression d'un User

Les fichiers dans **Entity** ont pour but de déterminer les différentes caractéristiques d'un objet. Ici on parle d'objets tels qu'User ou Task.

Dans le dossier **Form**, il y aura les formulaires qui vont être chargés pour les formulaire de connexion ou publication d'une tâche par exemple.

2. Les fichiers de configuration

Ces fichiers sont aux cœurs de l'application et sont utilisés pour paramétrer les grands axes de l'application tels que :

- Les accès servers
- Les routes
- Les droits d'accès clients
- L'ajout d'un nouveau module comme service

Ces fichiers de configuration se trouvent dans :

- app/config/

a. Les accès servers

Comme déjà expliqué plus haut dans ce document, le paramètre de BDD se fera dans les fichiers :

- parameters.yaml (Pour la BDD de l'application)
- parameters_test.yaml (Pour la BDD de test)

b. Les routes

Quand une requête arrive, le système de routage essaie de faire correspondre une route à une URL entrante. La première route qui correspond gagne, ainsi l'ordre dans lequel les routes sont définies dans le fichier de configuration est important.

Le fichier de configuration routing.yml permet la définition ces routes.

c. Les droits d'accès clients

C'est ici que vous pouvez gérer la sécurité sur différent niveau, tout est très bien expliqué par la documentation Symfony : https://symfony.com/doc/3.4/reference/configuration/security.html

d. L'ajout d'un nouveau module comme service

Dans Symfony 3, le fichier services.yml joue un rôle clé dans l'ajout d'injection. C'est ici que l'on va ajouter un nouveau controller pour l'appeler ensuite comme service. Encore une fois, tout est expliqué sur la documentation : https://symfony.com/doc/3.4/service_container.html

3. Les fichiers de ressources (View, CSS & JS…)

Les fichiers de ressources sont les fichiers qui sont appelés lors du chargement de la page. Dans ces fichiers on va retrouver :

- Les fichiers de vues (views)
- Les fichiers de style (CSS)
- Les fichiers pour gérer des choses plus dynamique (JS)
- Les fichiers images

Les fichiers des vues sont de type twig, ce qui est assez classique pour symfony, toutes ces vues se trouvent dans :

- app/Resources/views

Pour les fichiers de ressources de type CSS, JS ou Images, ils vont tous être stockés dans le dossier web :

- web/css
- web/fonts
- web/img
- web/js

4. Les fichiers de tests

Les fichiers concernant les test se trouvent dans 2 dossiers :

- Tests/AppBundle
- Web/test-coverage

Dans le premier dossier, c'est ici que tous les tests vont être effectué. Donc si un test doit être modifié ou de nouveaux tests doivent être créés, c'est dans ce dossier qu'il faut se rendre.
La structure du dossier est similaire à celle du dossier de fonctionnement puisque le but est de tester que les fichiers méthodes contenu dans les fichiers de fonctionnement.

Dans le second dossier (test-coverage), c'est un dossier qui est généré lorsque l'on effectue les tests. Pour le générer, il faudra effectuer lancer cette commande dans la console :

```
vendor/bin/phpunit --coverage-html web/test-coverage
```

IV. Processus de qualité et de collaboration

Le code est développé en respectant le processus de qualité PSR-12 et PSR-4 et doit continuer ainsi comme Symfony le recommande.

Afin de gérer la collaboration, le projet peut être géré via des outils tel que GitLab ou GitHub.

Actuellement le projet est disponible est sur un directory sur GitHub. A partir de ce repository, les nouveaux développeurs devront le cloner sur leur local et ensuite créer leur propre branche.

En créant leur propre branche de développement cela limite les risques de conflits avec la version qui est déployée sur la version Master mais aussi avec les versions que d'autres développeurs pourraient push. Le processus pour installer et déployer le repository doit se faire comme suit :

- Cloner repository
- Création de sa propre branche
- Faire les développement en local sur sa branche
- Avant de pusher, faire un Pull afin de s'assurer qu'il n'y ait pas de conflits entre la version master déployée et la version modifiée en locale
- Pusher sa version local version dans une pull request
- Contrôle et validation de la pull request par un développeur Senior ou Lead Développer