Une image contenant texte, document

Description générée automatiquement

**ToDo & Co**

**Documentation technique**

Table des matières

[ToDo & Co 1](file:////Users/vincentorru/Documents/openclassrooms/Word/P8-TodoList-documentation-technique.docx#_Toc115697860)

[Introduction 2](#_Toc115697861)

[Contexte 2](#_Toc115697862)

[Migration de Symfony 3.1 à 5.4 2](#_Toc115697863)

[L’Authentification 3](#_Toc115697864)

[Liaison des tâches à l’utilisateur 4](#_Toc115697865)

[Ajout des différents rôles à l’utilisateur 5](#_Toc115697866)

[Gestion des utilisateurs uniquement par l’administrateur 6](#_Toc115697867)

[Intégration des tests 7](#_Toc115697868)

[Tests unitaires 7](#_Toc115697869)

[Tests fonctionnels 8](#_Toc115697870)

[Tests coverage 9](#_Toc115697871)

[Les DataFixtures 10](#_Toc115697872)

# Introduction

Cette documentation technique a pour but de vous présenter les évolutions liées à l’amélioration continue de l’application PHP/Symfony Todo & Co.

Celle-ci vous détaillera les améliorations apportées au système d’authentification de l’utilisateur ainsi qu’envers la gestion de la tâche par l’utilisateur authentifié.

# Contexte

A la prise de connaissance du cahier des charges nous avons convenu d’opérer plusieurs modifications :

\_ Migration de Symfony 3.1 à 5.4

\_ L’authentification

\_ Liaison des tâches à l’utilisateur

\_ Ajout des différents rôles à l’utilisateur (user & admin)

\_ Gestion des utilisateurs uniquement par l’administrateur

\_ Intégration des tests unitaires

# Migration de Symfony 3.1 à 5.4

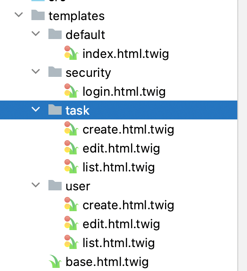
Afin de maintenir l’application à jour ainsi que dans le but d’optimiser la sécurité.

Nous avons dû opérer un changement majeur en migrant la version de Symfony vers la version 5.4.

Une image contenant texte

Description générée automatiquementPour cela, nous avons repris un web-skeleton de base symfony 5.4 tout en y adaptant le dossier « src » de l’ancien projet dans le but de pouvoir commencer les modifications. Le sous dossier AppBundle n‘existe plus. Celui-ci comporte les Controllers les Entité ainsi que les Formulaires.

Le contenu du dossier « web » regroupe les ressources accessibles (html, css, js, images) nécessaires au bon fonctionnement navigateur a été renommé « public ». Votre server devra pointer sur ce dossier car il contient le fichier index.php.

Les dossier app/Resources/view a été remplacé par le dossier templates qui est accessible directement depuis la racine du projet.

# L’Authentification

Une image contenant texte

Description générée automatiquement

L’authentification est gérée par le controller :

* src/Controller/SecurityController.php

Celui-ci regroupe 2 fonctions l’une « index » et l’autre « logout »

La fonction index :

SecurityController est un controller spécial de Symfony qui gère l’authentification via la page index du formulaire de login. Celui vérifie lui-même sans avoir à ajouter du code si la soumission du formulaire du login est correcte afin d’authentifier le bon utilisateur.

La fonction logout :

Ne peut être appelée qu’en cas d’authentification reconnue de l’application Symfony.

Elle permet à l’utilisateur qui le souhaite d’arrêter sa session.

Ici, non plus c’est Symfony on ne surcharge pas cette fonctionnalité.

On a juste à activer Logout dans fichier de config suivant :

config/packages/security.yaml



# Liaison des tâches à l’utilisateur

Ajout de la liaison « ManyToOne » entre le « User » et « Task »   dans l’entité Task dont le chemin est

Une image contenant texte

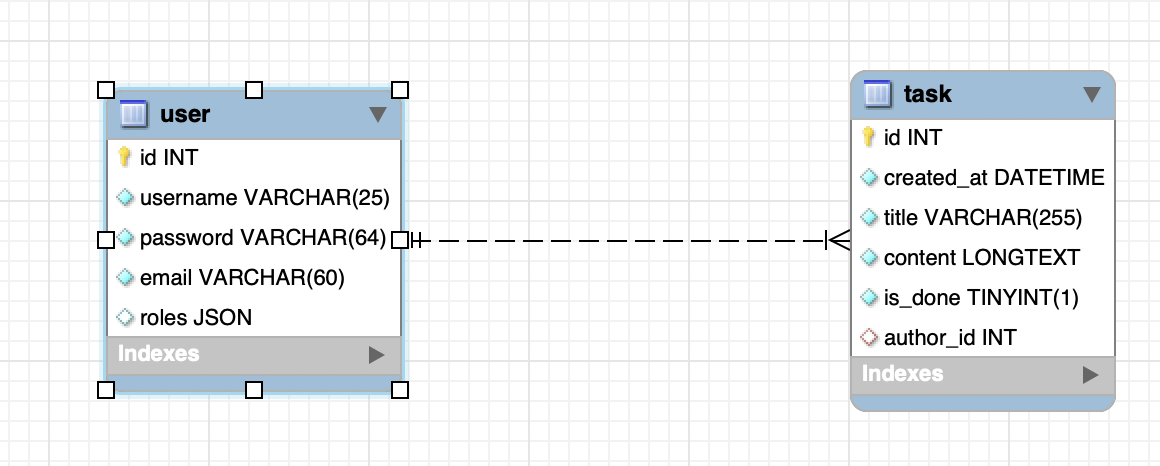
Description générée automatiquement

* src/Entity/Task.php
* Command client : php bin/console make:task
* Selectionner « relation » lors du choix du type
* Choisir ManyToOne

Nous pouvons alors lancer la migration pour mettre à jour le schéma de base données :

Avec les commandes :

* php bin/console make:migration
* php bin/console doctrine:migrations:migrate



Après la migration nous observons bien que la migration a bien été répercutée sur le schéma.

Les Getter et les setters sont ajoutés automatiquement à la fin de la commande.

# Ajout des différents rôles à l’utilisateur

Une image contenant texte

Description générée automatiquement

* src/Entity/User.php

Une image contenant texte

Description générée automatiquementPour ce champ spécifique la donnée est enregistrée dans un tableau. Ce tableau est envoyé ensuite vers la Base de données par Doctrine et est ensuite converti au format Json.

La classe User étant une interface spécifique de Symfony « User » nous ajoutons les méthodes Getter & Setter comme exigée par la nouvelle version de Symfony 5.4 .

A chaque ajout d’un nouveau type de rôle utilisateur dans Symfony il faut renseigner la variable « access\_control » dans le dossier de configuration au chemin

=> config/packages/security.yaml

Comme suivi :

Une image contenant texte

Description générée automatiquement

Nous pouvons alors lancer la migration pour mettre à jour le schéma de base donnée :

Avec les commandes :

* php bin/console make:migration
* php bin/console doctrine:migrations:migrate

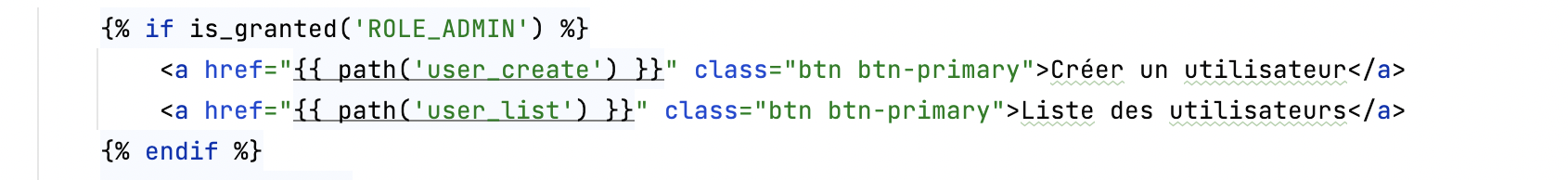
Une image contenant texte

Description générée automatiquement

Nous avons bien la colonne rôle qui ajoutée à notre base de données.

# Gestion des utilisateurs uniquement par l’administrateur

**FRONT** :



Affichage des liens des routes uniquement si l’utilisateur authentifié est admin

Avec la fonction twig « is\_granted »

**BACK** :



Ajout dans chaque routes restreintes aux non « Admin » de la fonction « denyAccessUnlessGranted »

# Intégration des tests

Une fois les fonctionnalités développées nous devons vérifier le bon fonctionnement de celles-ci afin de maintenir le code de l’application Symfony.

Une image contenant texte

Description générée automatiquementComme dans l’ancien projet nous retrouvons le dossier tests qui comporte les tests unitaires lié aux Entités (Entity) ainsi que les tests fonctionnels pour les Controllers

Nous pouvons lancer le test en configurant l’IDE mais pour être plus efficace nous pouvons lancer PHP/unit directement avec la console :

* **php bin/phpunit « path »**

## Tests unitaires

La classe php doit extends « TestCase » pour un test unitaire

Nous retrouvons ici la façon dont il faut procéder dans le cadre d’un test unitaire destiné à tester une entité (Task.php). Nous testons ici un getter et un setter de l’attribut « createdAt ».

Une image contenant texte

Description générée automatiquement

## Tests fonctionnels

La classe php doit extends « WebTestCase » pour un test fonctionnel.

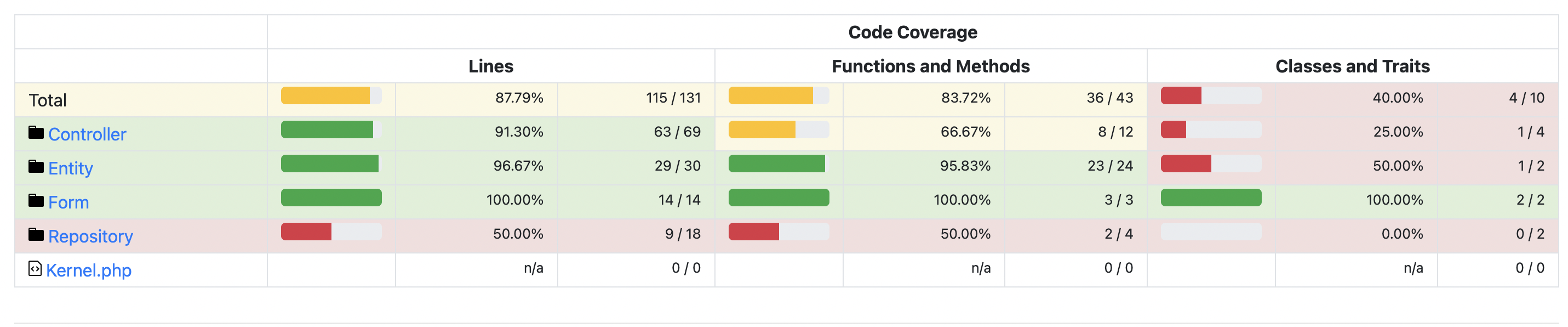
Nous vérifions en imitant un utilisateur qui effectue des actions via le client du site en implémentant le test des différentes fonctions du Controller. Les tests fonctionnels ont pour intérêt de faire gagner du temps car ils automatisent des tâches répétitives.

Par exemple, ici, nous vérifions si dans le cadre d’un ajout d’une tâche si la page redirige bien vers une autre comportant le message « la tâche a bien été ajouté »

**Une image contenant texte

Description générée automatiquement**

## Tests coverage

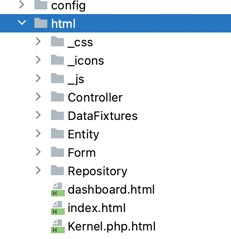


En complémentarité, le test Coverage a pour but d’exécuter l’intégralité des tests pour générer un Dashboard indiquant les taux de couverture des tests unitaires et fonctionnels à l’intérieur d’un projet.

Pour obtenir la génération du Dashboard en HTML on lance une commande via la console qui va analyser la couverture du projet :

php bin/phpunit --coverage-html « nom-du-dossier-souhaité »

Vous pouvez trouver le contenu du coverage actuel dans : /coverage/index.html



Nous obtenons dans le projet un nouveau dossier qui comporte une page : index.html que nous pouvons ouvrir via un navigateur.

## Les DataFixtures

Une image contenant texte

Description générée automatiquement

Dans le cadre du bon déroulement des test j’ai dû créer des classes fixtures (voir doc : [DoctrineFixtureBundle](https://symfony.com/bundles/DoctrineFixturesBundle/current/index.html) ). Ce Bundle permet d’insérer des fausses données en base de données de test afin de pouvoir initialiser correctement les tests fonctionnels avec les bonnes valeurs.

Ci-dessous, nous créons 2 objets task en bdd au lancement de la commande fixture :

php bin/console doctrine:fixtures:load –env=test

Une image contenant texte

Description générée automatiquement

Attention à chaque lancement de commande les données enregistrées seront supprimées.