

Rapport d'audit

Todo&Co

# Table des matières

Contexte – Page 3

Outils d'analyse utilisés - Page 3

Analyse de l'existant - Page 4

1. Framework Symfony - Page 4

2. PHP - Page 4

3. Les anomalies - Page 4

- Les anomalies rapportées - Page 4

- Les anomalies identifiées - Page 5

4. Analyse du code – Page 6

- Codacy - Page 6

- PhpStan - Page 6

5. Analyse de performance – Page 7

Solutions apportées – Page 8

1. Mise à jour de Symfony, PHP et Bootstrap – Page 8

2. Corrections des anomalies – Page 9

- Corrections des anomalies rapportés – Page 9

- Correction des anomalies identifiées – Page 9

3. Ajout de nouvelles fonctionnalités – Page 10

- Autorisation – Page 10

- Implémentation des tests automatisés – Page 10

- Intégration continue – Page 12

4. Analyse Codacy – Page 12

5. Analyse de performance – Page 13

# Contexte

Todo & Co une startup dont le cœur de métier est une application nommé « Todo List » permettant de gérer ses tâches quotidiennes.

La stratégie de développement utilisé a été le MVP (soit Minimum Viable Product) dont cette dernière consiste à permettre à la startup de présenter et promouvoir son application le plus rapidement afin d'obtenir de possible retours d'éventuels investisseurs.

Il est à noter que ceci n'est que la première étape dans la phase du développement de l'application car Todo & Co recherche désormais à améliorer la qualité de l'application en y ajoutant de nouvelles fonctionnalités ainsi que la correction de certaines anomalies et en y implémentant des tests automatisés

Avant d'y apporter les travaux nécessaires il est donc important d'analyser le code du projet initial ainsi que les performances de ce dernier afin de connaître les mise à jour à lui apporter et également pour optimiser le suivi de développement de l'application et son utilisation par les futurs clients et usagers.

Nous trouverons donc dans ce document un rapport détaillé concernant :

- Une analyse de l'application dans sa toute première version
- Les axes d'améliorations possibles
- Une analyse sur sa version actuelle

## Outils d'analyse utilisés

- Codacy : Analyse statique du code
- PhpStan : Analyse statique du code afin d'analyser les erreurs
- Profiler (Symfony) : Collecte les informations sur l'application

# Analyse de l'existant

## **1 - Framework Symfony**

La version du framework Symfony utilisé pour la version initial est la 3.1 . Cette version est sortie en mai 2016 et n'est plus maintenue depuis juillet 2017. Il est indispensable d'effectuer une mise à jour du framework

## **2 – PHP**

La version de langage PHP utilisé est la version 7.2. Cette version n'est plus maintenue depuis le 30 novembre 2020 il est impératif de passer a une version de PHP plus récente afin d'éviter de s'exposer à d'éventuels failles de sécurité et bugs

## **3 – Les anomalies**

### Les anomalies rapportées :

Plusieurs anomalies on été rapportées par les développeurs du concept. Il nous est demandé de les corriger, voici le détail des modifications à apporter :

- Rattacher la tâche à l'utilisateur qui l'à créée.
- Lors de la modification de la tâche, l'auteur ne peut pas être modifié.
- Pour les tâches déjà créées, il faut qu'elles soient rattachées à un utilisateur "anonyme".
- Pouvoir choisir un rôle (utilisateur ou administrateur) lors de la création et de la modification d'un compte utilisateur

### Les anomalies identifiées :

De nombreuses anomalies ont été identifiées après examen :

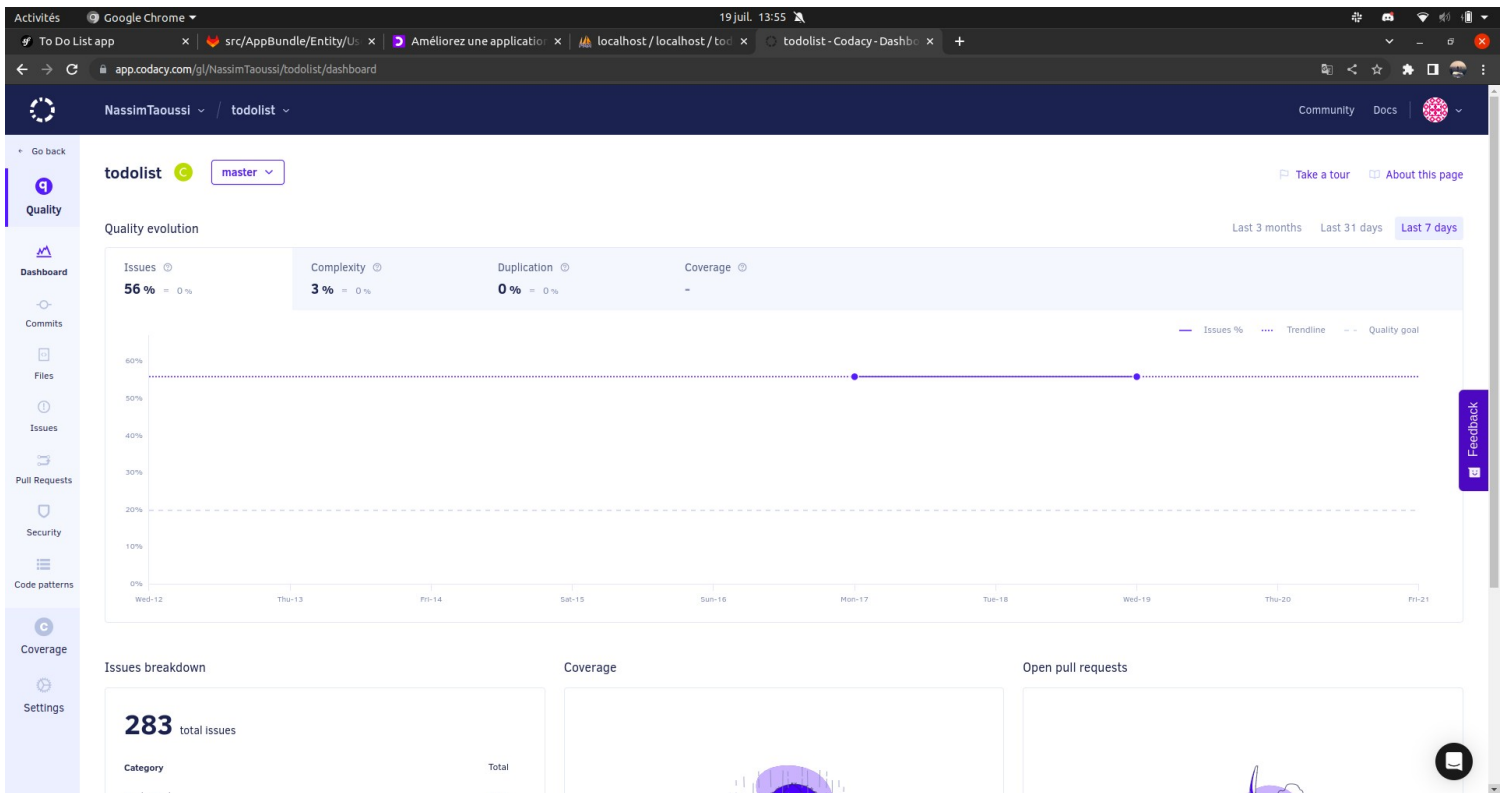
- Absence de lien vers la page d'accueil,
- Absence de page pour les tâches marquées comme « Terminées » (mais lien existant)
- Plusieurs problèmes de validation que les entités User et Task

*Exemple* : Les comptes utilisateurs peuvent avoir la même adresse mail et le même username .

- A la création d'un compte, la redirection s'effectue sur la page de gestion des utilisateurs (accès réservé aux administrateurs).
- Absence de messages flash sur certaines actions

## 4 – Analyse du code

### Rapport de qualité du code : Codacy



Ce rapport fait état de 283 issues et d'un pourcentage de fichiers complexes de 3 %. Codacy attribue un badge de niveau C à cette application. Les codes pattern utilisé sont : PHP\_CodeSniffer, PHP Mess Detector & PMD

Voici la répartition des issues par catégories

Category	Issues total
Code style	251
Security	29
Unused Code	3

### Rapport de qualité du code : PHPStan

(a été utilisé au niveau 8 afin d'avoir une analyse plus restrictive)

Un total de 55 erreurs on été recenser

## 5 – Analyse de performance

Avec l’outil Profiler de Symfony les différentes performances d’exécution des pages du site ont pu être analyser .

Page	Route	Method	Total execution time	Symfony initialization	Peak memory usage
Home page without login	/login	GET	28 ms	9 ms	2,00 MB
Home page with login	/	GET	16 ms	4 ms	2,00 MB
Login	/login_check	POST	547 ms	52 ms	2,00 MB
Home page after logout	/	GET	12 ms	4 ms	2,00 MB
Create user page without submission	users/create	GET	34 ms	8 ms	4,00 MB
after submission	users/create	POST	107 ms	5 ms	8,96 MB
Edit user page without submission	users/id/edit	GET	63 ms	12 ms	4,00 MB
after submission	users/id/edit	POST	114 ms	13 ms	8,96 MB
Create task page without submission	tasks/create	GET	43 ms	15 ms	4,00 MB
after submission	task	POST	108 ms	17 ms	8,96 MB
Delete task	tasks/id/delete	GET	41 ms	12 ms	2,00 MB
Toggle task	tasks/id/toggle	GET	54 ms	20 ms	2,00 MB
List task page	tasks	GET	36 ms	7 ms	2,00 MB

# Solutions apportées

## **1. Mise à jour de Symfony, PHP & Bootstrap**

**Mise à jour de PHP :** Une mise à jour du projet vers PHP 8.1 a été effectuée pour une bonne garantie de la sécurité ainsi que les nouvelles fonctionnalités disponibles.

**Mise à jour de Symfony :** Pour garantir une bonne stabilité à l'application, il est préférable d'utiliser une version LTS de Symfony donc pour la mise à jour apporter à l'application du point de vue Framework j'ai choisi la version LTS 5.4.21 sortie en Novembre 2021 de Symfony et qui sera maintenue jusqu'en Novembre 2024

Toutes les dépendances ont été mises à jour dans des versions supportées et d'autres ont été installées, par exemple :

- symfony/browser-kit
- symfony/css-selector
- symfony/maker-bundle
- symfony/phpunit-bridge

**Mise à jour de Bootstrap :** Une mise à jour vers bootstrap 5.2 a été réalisée à l'aide de Webpack ce qui nous permet de bénéficier de toutes les nouveautés et d'une meilleure optimisation.



## **2. Corrections des anomalies**

### **Correction des anomalies rapportés :**

- Chaque tâche créée est rattachée à son créateur.
- Lors de la modification de la tâche, l'auteur ne peut pas être modifié.
- Pour les tâches déjà créées, elles sont rattachées à un utilisateur « anonyme ».
- Il est possible de choisir le rôle de l'utilisateur lors de la création du compte ou lors de son édition.

### **Correction des anomalies identifiées :**

- Ajout des liens vers la page d'accueil.
- Création de la page des tâches terminées.
- Les contraintes de validation ont été ajoutées, il n'est notamment plus possible de créer des comptes avec le même Login ou la même adresse email. (Contrainte d'unicité).
- A la création d'un compte, l'utilisateur est redirigé vers la page de login.
- Ajout des messages flash de confirmation pour les actions qui n'en disposaient pas.
- Les boutons pour changer le statut des tâches indiquent le bon statut.
- La date de création et le créateur de la tâche est affiché pour chaque tâche.

### 3. Ajout de nouvelles fonctionnalités

#### Autorisation :

L'ajout de nouvelles fonctionnalités notamment sur la partie autorisation on été appliquer.

- Désormais seuls les administrateurs ont accès à la gestion des utilisateurs
- Les tâches ne peuvent être supprimées que par leurs auteurs.
- Pour les tâches anonymes, seuls les administrateurs peuvent les supprimer




































Le système d'annotation a été préféré à la rubrique access\_control du fichier security.yaml pour une meilleure compréhension du code sur les autorisations en fonction du rôle. Pour la suppression des tâches par leur auteur et pour la suppression des tâches anonymes par les administrateurs ainsi que l'édition, il a été nécessaire de mettre en place un système de Voter.

#### Implémentation des tests automatisés :

##### - Les tests unitaires :





























Les tests unitaires ont été réalisés sur les entités. Il s'agit de tests classiques sur les setters et sur les getters des propriétés.

Exemple pour la classe User :

C:\laragon\www\todolist\src / Entity / User.php									
	Code Coverage								
	Lines	Functions and Methods			Classes and Traits				
Total	 100.00%	18 / 18	 100.00%	14 / 14	 100.00%	1 / 1	 100.00%	1 / 1	
User	 100.00%	18 / 18	 100.00%	14 / 14	14	 100.00%	1 / 1	1 / 1	
__construct	 100.00%	1 / 1	 100.00%	1 / 1	1				
getId	 100.00%	1 / 1	 100.00%	1 / 1	1				
getUsername	 100.00%	1 / 1	 100.00%	1 / 1	1				
setUsername	 100.00%	2 / 2	 100.00%	1 / 1	1				
getPassword	 100.00%	1 / 1	 100.00%	1 / 1	1				
setPassword	 100.00%	2 / 2	 100.00%	1 / 1	1				
getEmail	 100.00%	1 / 1	 100.00%	1 / 1	1				
setEmail	 100.00%	2 / 2	 100.00%	1 / 1	1				
getRoles	 100.00%	1 / 1	 100.00%	1 / 1	1				
getUserIdentifier	 100.00%	1 / 1	 100.00%	1 / 1	1				
getSalt	 100.00%	1 / 1	 100.00%	1 / 1	1				
eraseCredentials	 100.00%	1 / 1	 100.00%	1 / 1	1				
getRole	 100.00%	1 / 1	 100.00%	1 / 1	1				
setRole	 100.00%	2 / 2	 100.00%	1 / 1	1				

## Exemple pour la classe Task :

C:\laragon\www\todolist\src / Entity / Task.php















	Code Coverage								
	Lines			Functions and Methods				Classes and Traits	
Total		100.00%	16 / 16		100.00%	11 / 11	CRAP		100.00% 1 / 1
Task		100.00%	16 / 16		100.00%	11 / 11	11		100.00% 1 / 1
<a href="#">getId</a>		100.00%	1 / 1		100.00%	1 / 1	1		
<a href="#">getCreatedAt</a>		100.00%	1 / 1		100.00%	1 / 1	1		
<a href="#">setCreatedAt</a>		100.00%	2 / 2		100.00%	1 / 1	1		
<a href="#">getTitle</a>		100.00%	1 / 1		100.00%	1 / 1	1		
<a href="#">setTitle</a>		100.00%	2 / 2		100.00%	1 / 1	1		
<a href="#">getContent</a>		100.00%	1 / 1		100.00%	1 / 1	1		
<a href="#">setContent</a>		100.00%	2 / 2		100.00%	1 / 1	1		
<a href="#">isDone</a>		100.00%	1 / 1		100.00%	1 / 1	1		
<a href="#">setIsDone</a>		100.00%	2 / 2		100.00%	1 / 1	1		
<a href="#">getAuthor</a>		100.00%	1 / 1		100.00%	1 / 1	1		
<a href="#">setAuthor</a>		100.00%	2 / 2		100.00%	1 / 1	1		

## - Les tests fonctionnels :

Les tests fonctionnels ont pour objectif de tester le bon fonctionnement des fonctionnalités de l'application, c'est à dire comme un utilisateur pourrait le faire. Il faut donc émuler une requête, grâce à l'implémentation d'un client, et veiller à obtenir la réponse attendue. Pour être effectuer ces tests utilisent la base de données.

## Exemple sur UserControllerTest :

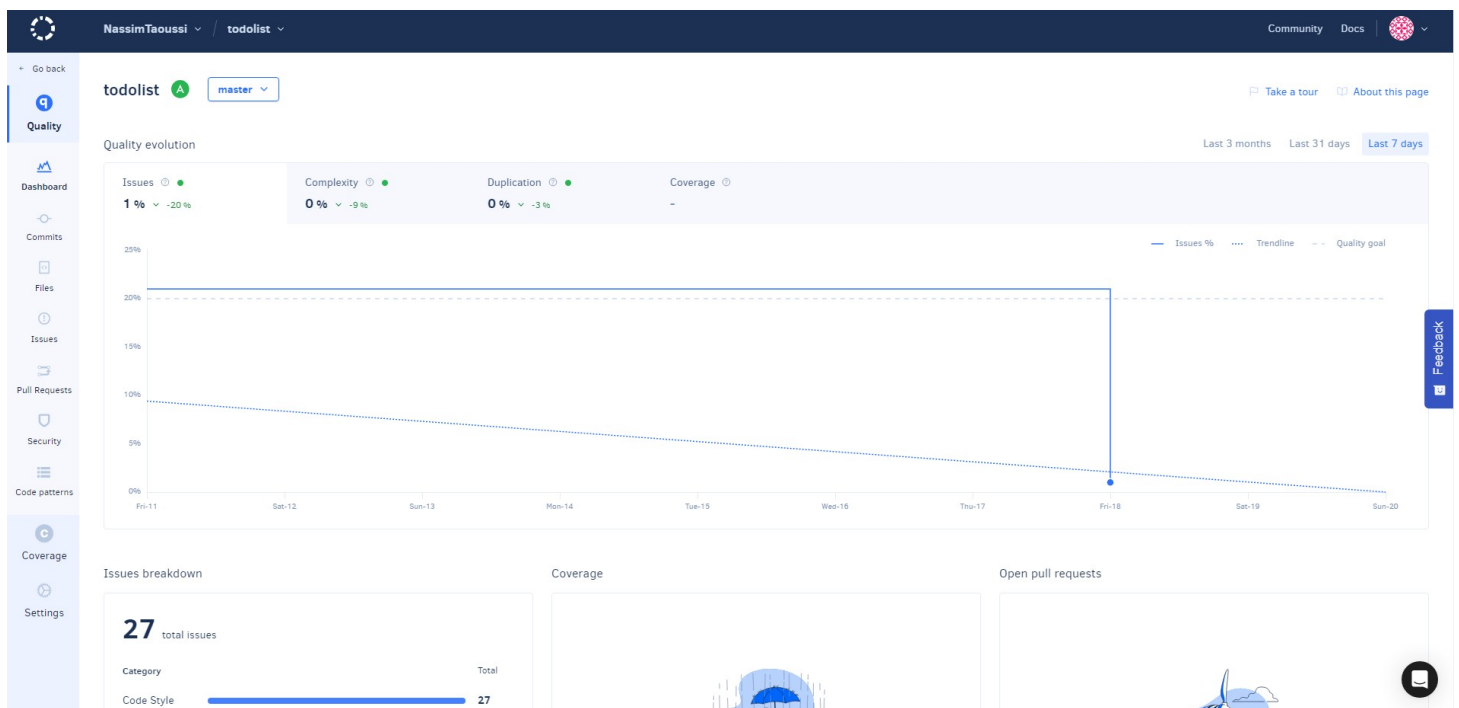
C:\laragon\www\todolist\src / Controller / UserController.php

	Code Coverage								
	Lines			Functions and Methods				Classes and Traits	
Total		100.00%	24 / 24		100.00%	4 / 4	CRAP		100.00% 1 / 1
<a href="#">UserController</a>		100.00%	24 / 24		100.00%	4 / 4	8		100.00% 1 / 1
<a href="#">index</a>		100.00%	4 / 4		100.00%	1 / 1	1		
<a href="#">createUser</a>		100.00%	9 / 9		100.00%	1 / 1	3		
<a href="#">editUser</a>		100.00%	9 / 9		100.00%	1 / 1	3		
<a href="#">deleteUser</a>		100.00%	2 / 2		100.00%	1 / 1	1		

**Intégration continue :** Pour l'intégration continue une documentation a été réaliser concernant le cheminement à suivre pour le développement ainsi que les bonnes pratique à appliquer et les tests a effectuer.

## 5. Analyse Codacy

Après le code refactorisé et remanié nous atteignons la note de A sur Codacy. Toujours avec les mêmes code pattern : PHP\_CodeSniffer, PHP Mess Detector & PMD



## 6. Analyse de performances

Lors de l'analyse effectuée sur l'application, il nous est apparu que le système d'autoloading de Composer (notamment l'utilisation de la fonction `includeFile()`), était un des principaux consommateurs de ressources. Après quelques recherches, il est indiqué qu'il est possible d'améliorer les performances de ce dernier grâce à la commande : **composer dump-autoload -o** Celle-ci permet de regrouper dans un fichier l'ensemble des classes et réduit les multiples appels à certaines de ses fonctions. Ainsi, lorsque Composer aura à utiliser une classe, il n'ira plus chercher si le fichier contenant la classe existe, il

consultera la liste des classes dans un fichier. En effet, après un nouveau test de la page login, en utilisant le Profile de Symfony, on a un gain de performance relativement substantiel :

Page	Route	Method	Total execution time	Symfony initialization	Peak memory usage
Home page without login	/login	GET	13 ms	4 ms	4,00 MB
Home page with login	/	GET	10 ms	1 ms	2,00 MB
Login	/login	POST	505 ms	3 ms	4,00 MB
Home page after logout	/	GET	6 ms	2 ms	2,00 MB
Logout	/logout	GET	24 ms	4 ms	6,00 MB
Create user page without submission	users/create	GET	22 ms	5 ms	4,00 MB
after submission	users/create	POST	531 ms	4 ms	4,00 MB
Edit user page without submission	users/id/edit	GET	26 ms	9 ms	4,00 MB
after submission	users/id/edit	POST	27 ms	5 ms	4,00 MB
Create task page without submission	tasks/create	GET	53 ms	3 ms	6,00 MB
after submission	task	POST	49 ms	4 ms	4,00 MB
Delete task	tasks/id/delete	GET	53 ms	5 ms	4,00 MB
Toggle task	tasks/id/toggle	GET	18 ms	4 ms	4,00 MB
List task page	tasks	GET	10 ms	2 ms	2,00 MB

Les performances font un léger bon même si c'est au prix d'une augmentation de la mémoire vive consommée et de l'utilisation du processeur. Hormis pour la création d'un utilisateur car le changement d'algorithme de hashage est plus gourmand que le précédent.