21/04/2020

Jerome Sauvage

Todo&co

Audit de qualité

Application ToDoList



Table des matières

[Introduction 2](#_Toc40701683)

[- 1 – Audit du code initial backend / frontend 2](#_Toc40701684)

[1.1 Rapport CodeClimate : 2](#_Toc40701685)

[1.2 Rapport PHPMetrics et PHPCS 3](#_Toc40701686)

[1.3 Version de Symfony et PHP 5](#_Toc40701687)

[1.4 La sécurité : 6](#_Toc40701688)

[1.5 Mesure de performance BlackFire du code initial 6](#_Toc40701689)

[1.5 Mesure de performance BlackFire du code initial (suite) 7](#_Toc40701690)

[1.6 Principes SOLID 9](#_Toc40701691)

[1.7 Frontend 10](#_Toc40701692)

[1.8 Conclusion 11](#_Toc40701693)

[1.8.1 Backend 11](#_Toc40701694)

[1.8.2 Frontend 11](#_Toc40701695)

## Introduction

ToDo & Co est une startup dont le cœur de métier est une application de gestion des tâches quotidiennes.

L’entreprise vient tout juste d’être crée, et l’application a dû être développée rapidement pour permettre de présenter à de potentiels investisseurs que le concept est viable (Minimum Viable Product ou MVP) avec le Framework Symfony dans sa version 3.1.

La mission qui m’a été confier et de développer de nouvelles fonctionnalités, et d’analyser la qualité du code ainsi que ses performances.

Ce document explique l’audit réalisé ainsi que les points d’amélioration à réaliser.

## - 1 – Audit du code initial backend / frontend

### 1.1 Rapport CodeClimate :

Une image contenant capture d’écran

Description générée automatiquement

Des analyses de qualité de codes ont été effectuées avec CodeClimate.

Ces analyses n’ont pas révélé de problèmes particuliers sur le code de l’application.

Il y a des remontées sur le code du framework Symfony, elles ont été ignorées, ainsi qu’une maintenabilité classée D qui doit être améliorée.

Les PSR1 et PSR2 sont respectés mais la PHPDoc est absente, elle est importante pour une meilleure compréhension et maintenance du code.

De plus elle servira aux futurs développeurs qui pourraient reprendre l’application pour la faire évoluer.

### 1.2 Rapport PHPMetrics et PHPCS

Une image contenant capture d’écran

Description générée automatiquement

Le rapport montre un taux de complexité des classes de 6.07 qui doit être abaissé au maximum.

PHPCS scan le code de chaque classe est donne un rapport très détaillé sur les éventuelles erreurs de code et corrections qui doivent être apportés.

Le rapport de phpcs sur les contrôleurs rapporte que l’Indentation, ouverture et fermeture de parenthèses, annotations peuvent être corrigé grâce à l’outil intégré PHPCBF. Certaines corrections doivent être faite à la main. Voir figure 1, figure 2, figure 3.

Une image contenant assis, table

Description générée automatiquement

Figure 1

Une image contenant assis, noir, table, ordinateur

Description générée automatiquement

Figure 2

Une image contenant assis, table, ordinateur

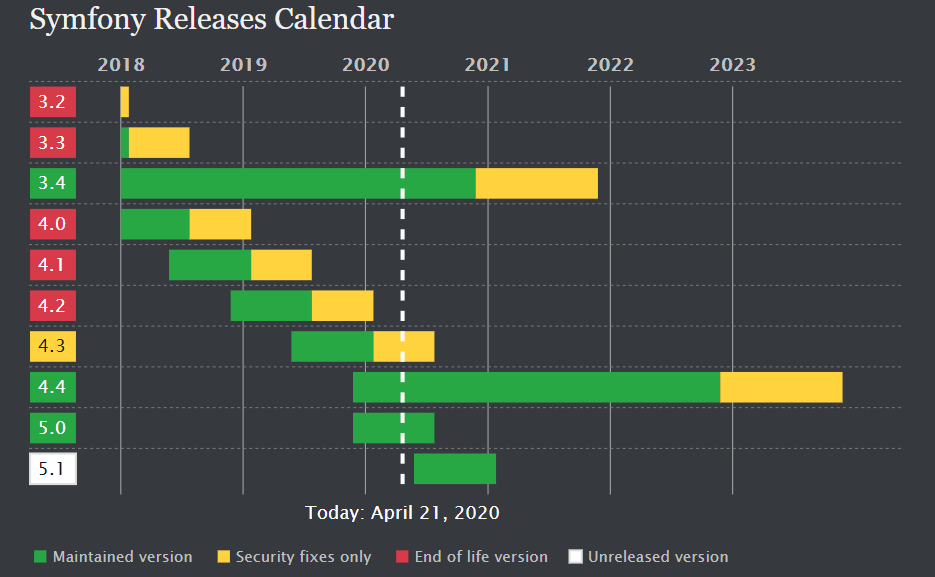
Description générée automatiquement

Figure 3

### 1.3 Version de Symfony et PHP

La version de Symfony initial de l’application est la 3.1.

Cette version a été mis en ligne en mai 2016 n’est plus maintenu depuis 2017 et n’apparaît plus dans la roadmap.



Passer sur la version 4.4 (LTS = Long Term Support) de Symfony permet de recevoir des mises à jour jusqu’en Novembre 2020 et des correctifs de sécurité jusqu’en Novembre 2022.

Seconde optimisation : Upgrader la version de PHP

La version de PHP utilisé est la 5.5.9, passer à la version 7.2 de PHP augmentera les performances au niveau de l’utilisation des ressources et offrira la possibilité d’utilisé les dernières fonctionnalités de PHP comme le typage des variables qui rendra le code plus stable.

Comparaison des performances de Symfony 3.1 avec PHP 5 et la version 4.4 de Symfony avec PHP 7.3 sur le site php benchmarks :

Une image contenant capture d’écran, différent, groupe, nombreux

Description générée automatiquement

Une image contenant capture d’écran

Description générée automatiquement

1.4 La sécurité :

Le niveau de sécurité initial est insuffisant, il permet d’accéder à certaines pages sans être authentifié.

La page qui liste les utilisateurs /users et qui permet leur modification /users/{id}/edit est accessible sans authentification. Un correctif doit être apporté pour que cette page soit accessible seulement à un utilisateur ayant les droits ROLE\_ADMIN.

Des améliorations sont à apportés au formulaire d’authentification, il n’est pas protégé contre la faille CSRF.

Source : <https://symfony.com/doc/current/security/form_login_setup.html>

### 1.5 Mesure de performance BlackFire du code initial

#### 1.5.1 Mesure de la page Home

Une image contenant moniteur, écran, assis, ordinateur

Description générée automatiquement

#### 1.5.2 Mesure de la page Login

Une image contenant capture d’écran

Description générée automatiquement

#### 1.5.3 Mesure de la page Logout

Une image contenant moniteur, écran, grand, horloge

Description générée automatiquement

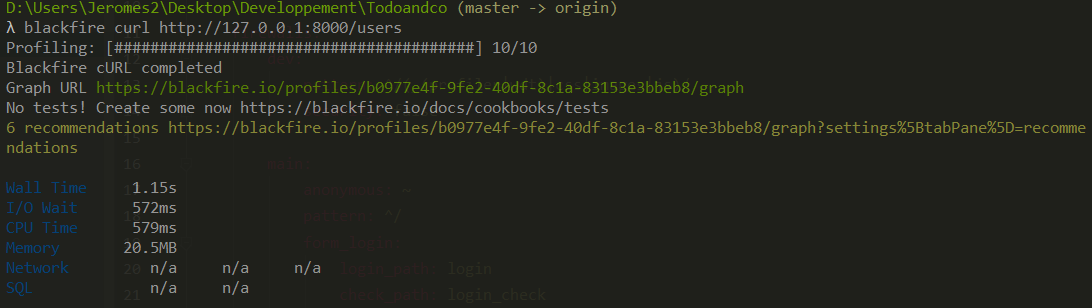
### 1.5 Mesure de performance BlackFire du code initial (suite)

#### 1.5.4 Mesure de la page Tasks

Une image contenant capture d’écran

Description générée automatiquement

#### 1.5.5 Mesure BlackFire de la page Users



#### 1.5.6 Synthèse et conclusion des mesures

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Mesure avant modifications | | Mesure après modifications | |
| Routes | Temps s ou ms | Mémoire mb |  |  |
| / | 1.28 s | 14.1 mb |  |  |
| /login | 660 ms | 14.4 mb |  |  |
| /logout | 582 ms | 13.8 mb |  |  |
| /tasks | 686 ms | 14.1 mb |  |  |
| /users | 1.15 s | 20.5 mb |  |  |
| Environnement de developpement | Windows | |  | |

Grace aux données récoltées par BlackFire on peut comprendre pourquoi une page mettra 1.28 s pour s’afficher et consommera 14.1 mb de mémoire.

Blackfire nous donne les Temps d’accès, l’exécution processeur, l’occupation en mémoire, temps de la requête sql ainsi nous pouvons voir ce qui peut être amélioré pour rendre l’application plus réactive.

L’article « Vitesse de chargement : pourquoi et comment l’améliorer ? » nous informe que « La vitesse de chargement d’un site est la première impression donnée au visiteur. A l’heure où tout doit être instantané, même sur mobile, l’utilisateur n’attend plus. Si votre site met trop longtemps à charger, il est susceptible de quitter la page pour retourner à sa recherche, ce qui va augmenter votre taux de rebond\*. Des études ont montré que 47% des utilisateurs s’attendent à ce qu’une page se charge en moins de 2 secondes et que 40% des internautes ferment une page si elle met plus de 3 secondes à s’afficher. Diminuer le temps de chargement va permettre de retenir l’utilisateur plus longtemps, d’augmenter le nombre de conversions (achats, inscriptions) et de donner une image positive à votre marque. Ainsi, le visiteur sera plus enclin à revenir sur vos pages. »

Source : [Genius Interactive](https://www.genious-interactive.com/vitesse-chargement-pourquoi-comment-ameliorer/?cn-reloaded=1)

## 

### 1.6 Principes SOLID

Un des principes SOLID dit qu’une classe, une fonction ou une méthode doit avoir une et une seule responsabilité. Le code sera implémenté en respectant certains des principes.

|  |  |  |
| --- | --- | --- |
|  | Principes | Définitions |
| S | Responsabilité unique (Single responsibility principle) | Une classe, une fonction ou une méthode doit avoir une et une seule responsabilité |
| O | Ouvert/fermé (Open/closed principle) | Une entité applicative (class, fonction, module ...) doit être ouverte à l'extension, mais fermée à la modification |
| L | Substitution de Liskov (Liskov substitution principle) | Une instance de type T doit pouvoir être remplacée par une instance de type G, tel que G sous-type de T, sans que cela ne modifie la cohérence du programme |
| I | Ségrégation des interfaces (Interface segregation principle) | Préférer plusieurs interfaces spécifiques pour chaque client plutôt qu'une seule interface générale |
| D | Inversion des dépendances (Dependency inversion principle) | Il faut dépendre des abstractions, pas des implémentations |

Source : [Wikipédia](https://fr.wikipedia.org/wiki/SOLID_(informatique))

Dans l’application, le fichier « TaskController » n’implémente pas le principe (SRP) qui explique qu’une classe n’a qu’une seule responsabilité. Le contrôleur de tâches implémente plusieurs responsabilités comme indiqué dans la figure 4.

Une image contenant capture d’écran

Description générée automatiquement

Figure

## 1.7 Frontend

Lors de nos tests de l’interface nous avons constaté un design manquant d’ergonomie et de fluidité.

Une image contenant table

Description générée automatiquement

Figure 5 : page de login

Une image contenant capture d’écran

Description générée automatiquement

Figure 6 : page de création d’un utilisateur

Nous voyons sur la figure 5 ci-dessus le bouton « créer un utilisateur » est situé en haut de page au-dessus de l’image principale de la page et le formulaire de connexion sous l’image.

Sur la figure 6 le formulaire a des proportions trop importantes.

Un test sur la responsive nous montre qu’il faut l’améliorer.

Une mise en forme plus adapté au standard actuel est nécessaire. Un design plus poussé rendra l’application plus attractive et intuitif à utiliser et offrira une meilleure expérience.

## 1.8 Conclusion

|  |  |
| --- | --- |
| Points d’améliorations | Backend |
| Symfony | Passage à la version 4.4 |
| Php | Passage à la version 7.3 |
| Sécurité | Voir la documentation de la sécurité |
| SOLID | Mise en place des principes SOLID |
| PSR | Respect des règles PSR |
| Maintenabilité | Rendre le code plus facilement maintenable |
| Documentation | Mise en place de la documentation |

L’audit de l’application ToDoList nous montre qu’il y a des améliorations à réaliser au niveau backend et frontend. Je propose l’amélioration des points suivant :

### 1.8.1 Backend

### 1.8.2 Frontend

|  |  |
| --- | --- |
|  | Frontend |
| Points d’améliorations |  |
| Design | Amélioration des interfaces |
| Responsive | Amélioration de la responsive |
| Contenu | Ajout d’une page terminée et d’une page d’erreur |