
ToDo & Co

Audit

Kévin Mulot



1. Introduction

1.1 Contexte

La société ToDo & Co, une jeune startup dont le cœur de métier est une application permettant de gérer ses tâches quotidiennes vient d'être créée. Une application a dû être développée à toute vitesse pour permettre de montrer à de potentiels investisseurs que le concept est viable (on parle de Minimum Viable Product ou MVP).

1.2 Mission

Il nous a été demandé d'améliorer la qualité global de global de l'application. Cela passe par un certain nombre d'étapes tel que :

- l'implémentation de nouvelles fonctionnalités
- la correction de quelques anomalies
- l'implémentation de tests automatisés

Ces tâches ayant déjà été effectuée nous nous concentrerons ici, sur les performances de l'application avec une analyse du code et explorerons les pistes à suivre pour améliorer la dette technique de ToDoList.

2. Qualité du code

2.1 Version

L'application fournie au départ était sous la version 3.1 de Symfony qui n'est plus maintenue depuis 2018 et utilisait PHP dans version 5.5.9. Une mise à jour à donc dû être effectuée pour assurer la pérennité et la sécurité de celle-ci. Nous sommes donc passé à Symfony dans sa version 4.4 LTS (version la plus stable et maintenue sur une longue période) utilisant PHP 7.1.3.

Par ailleurs, un grand nombre de bugs ont été résolus et l'ensembles des missions du cahier des charges ont été accomplies.

2.2 Outils d'analyse

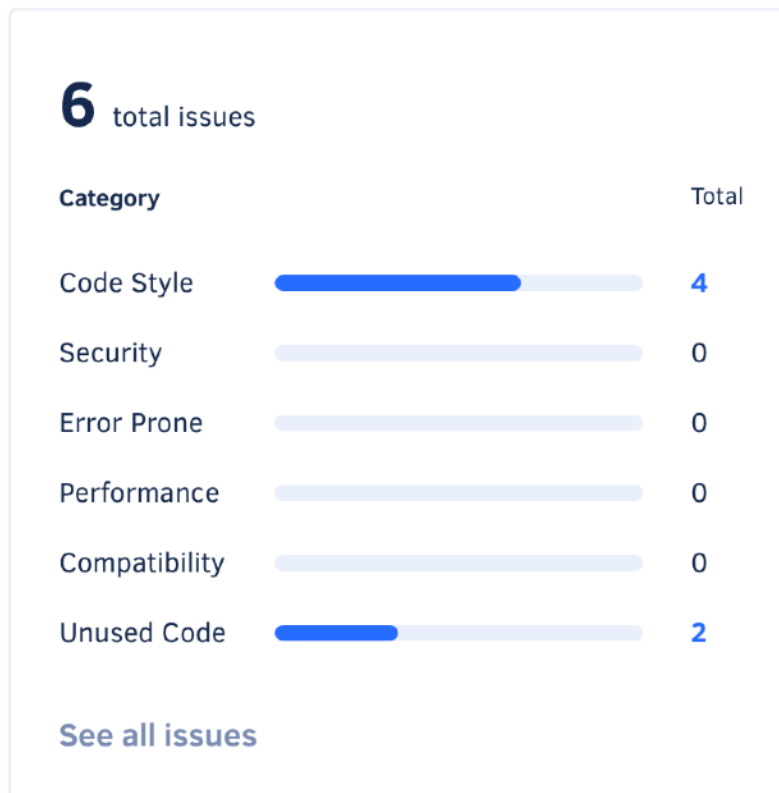
Afin d'analyser le code de l'application 2 outils ont été utilisés. Codacy pour mesurer les erreurs de code ainsi que le respect des bonnes pratiques. Code Climate, pour estimer la maintenabilité de l'application mais aussi les duplications de code.

2.3 Rapport Codacy



Après analyse de Codacy, on constate qu'il n'y a pas de complexité ni de duplications dans notre application. Le nombre d'issues ayant diminué à 1% suite aux différentes pull requests; la note globale de ToDoList est désormais de « A ». Le pourcentage étant calculé sur le nombre d'issues pour 1000 lignes de code. A noter que les fichiers du Framework Symfony sont volontairement exclus via la configuration pour une analyse plus pertinente.

Issues breakdown



Les différentes issues sont répertoriées par catégories et importance.

Info : Ces issues indiquent des Problèmes de style de code.

Avertissement : Elles préviennent que les normes ou conventions ne sont pas respectées.

Erreur : Ce sont les issues les plus urgentes. Il est impératif de les corriger. Bien que l'application fonctionne, vous pourriez rencontrer des bugs, problèmes de sécurité ou de compatibilité.

Codacy permet également une analyse automatisée de vos pull requests et vous indique la qualité de celles-ci grâce à GitHub. Il ne faut donc pas hésiter à revoir le code, si il présente des soucis avant de l'incorporer à la branche principale du projet.

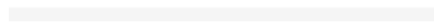
2.4 Rapport CodeClimate

Breakdown

94 FILES



MAINTAINABILITY



TEST COVERAGE

Codebase summary

MAINTAINABILITY

A 0 mins

TEST COVERAGE



Repository stats

CODE SMELLS

0

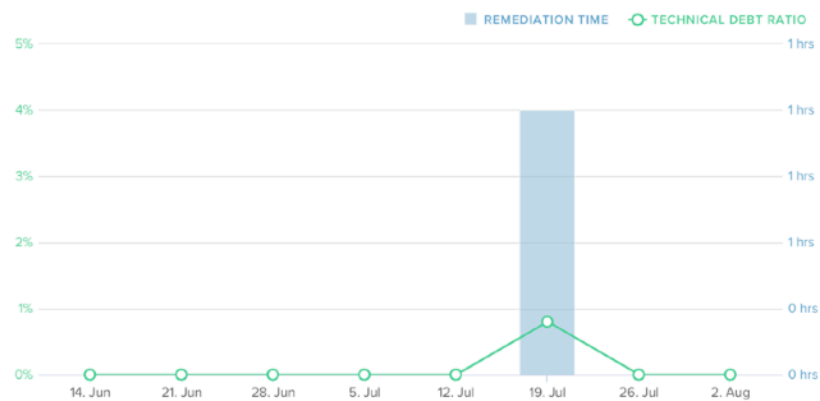
DUPPLICATION

0

OTHER ISSUES

0

Technical Debt



CodeClimate indique que la maintenabilité du site est de « A » et qu'il ne détecte pas d'erreurs, d'axes d'amélioration ou de duplications dans le code. Il ne faut cependant pas hésiter à consulter cette page après un changement dans le projet. Cette outil indique précisément les améliorations possible ainsi que la durée de leur mise en oeuvre.

3. Performance

3.1 Analyse Blackfire

L'application sera amenée à évoluer une fois sa mise en ligne et il convient donc d'en analyser les performances tout au long du processus de développement. Nous pourrions ainsi mieux anticiper les contraintes à venir et ainsi optimiser son fonctionnement.

Blackfire est une application web qui permet d'optimiser les ressources utilisées par l'application par une lecture simple et rapide des requêtes effectuées.

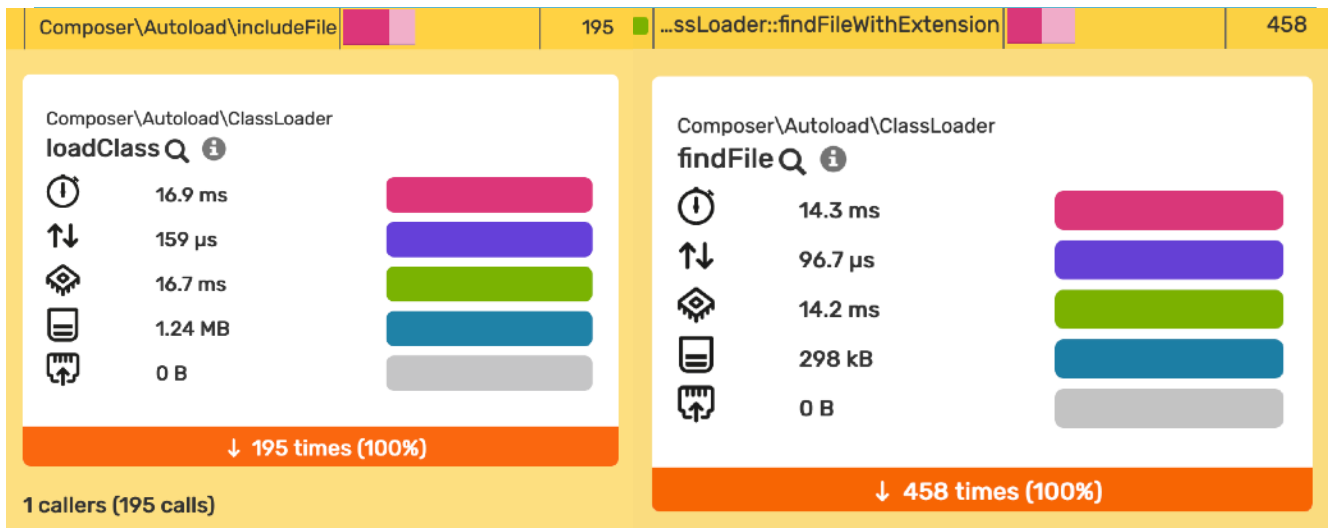
L'analyse est donc effectuée dans un environnement de production et localement. Cependant les chiffres énoncés sont amenés à évoluer une fois sa mise en ligne.



Function calls	% Excl. ▾	% Incl.	Calls
Composer\Autoload\includeFile			195
...ssLoader::findFileWithExtension			458
...poser\Autoload\includeFile@1			143
file_exists			474
.../EventDispatcher.php/299-305			28
...poser\Autoload\includeFile@2			66
...Autoload\ClassLoader::findFile			458
substr			1 424
...poser\Autoload\includeFile@3			33
...KernelProdContainer::closure			22
...utoload\ClassLoader::loadClass			195
twig_get_attribute			12
strrpos			831
...ionListener::onKernelResponse			1
...outerListener::onKernelRequest			1
...IProdContainer::getTwigService			1
...load\ClassLoader::loadClass@1			143
strtr			586
...Container::getTranslatorService			1
str_starts_with			93
...DefaultEntityManagerService			2
spl_autoload_call			196

Après analyse, il nous est donc fourni le temps de chargement de la page ainsi que la mémoire consommée pour son chargement. On peut également consulter chaque fonction avec leur temps d'exécution et la mémoire requise. Un schéma nous permet également d'observer le plan d'exécution de ces fonctions.



Cette liste indique les fonctions appelées en commençant par les plus gourmandes.



3.2 Optimisation

Après une première analyse on constate que les trois fonctions qui effectuent le plus grand nombre d'appels et prennent le plus de temps; concernent l'autoload de Composer. Il convient donc d'y remédier.

Blackfire nous fourni d'ailleurs des pistes d'améliorations pour résoudre notre souci.


 **PHP Preloading should be configured** 

When

```
is_extension_loaded("zend_opcache")
```

then:

```
runtime.configuration.opcache_preload = ""
```

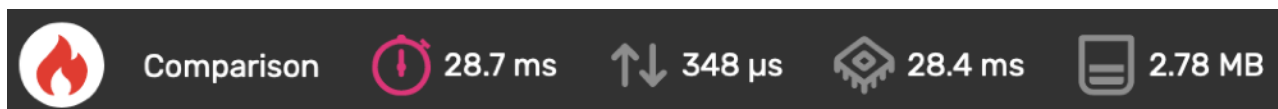
 **The Composer autoloader class map should be dumped in production** 

```
metrics.composer.autoload.find_file.count 458 <= 50
```

En l'occurrence, on nous indique que pour un chargement des classes plus efficace et plus rapide. Il est nécessaire de mettre en cache les classes utiles à notre application.

Une simple commande « **composer dump-autoload -o** » dans votre terminal à la racine du projet suffit, cependant il faudra penser à répéter l'opération si de nouvelles classes sont ajoutées.

Nouveaux résultats :



Lorsque l'on compare avec les anciens résultats, on constate que l'on gagne 36% de vitesse d'exécution sans affecter particulièrement la mémoire utilisée. Cependant le CPU est également allégé de 36% de temps d'activité. C'est donc une amélioration non négligeable.



3.3 Recommandations

Afin d'aller plus loin dans notre démarche d'optimisation, d'autres pistes sont à explorer pour conserver et optimiser les performances de notre application :

- OPCache : Son activation est fortement recommandée pour améliorer les performances de PHP en stockant le bytecode des scripts pré-compilés en mémoire partagée, faisant ainsi qu'il n'est plus nécessaire à PHP de charger et d'analyser les scripts à chaque demande.
- Migration vers la nouvelle version de Symfony LTS lorsqu'elle sera disponible.
- Choix d'un hébergeur fiable et performant pour la mise en ligne en prenant en compte le trafic attendu sur la page.
- Basculer vers la version payante de Blackfire pourrait vous donner de nombreux conseils pour améliorer les performances de l'application.