



**Audit de qualité et de performance
de l'application Todo & Co**

Sommaire

Contexte du projet	3
Objectifs du projet	3
Environnement technique du projet	3
Qualité et maintenabilité du code	4
Analyse de l'architecture de l'application	5
Choix de la version du nouveau projet	5
Mesures de performance avec blackfire.io	7
Axes d'amélioration	9

Contexte du projet :

Todo & co est une startup dont le cœur de métier est une application permettant de gérer ses tâches quotidiennes.

L'entreprise vient tout juste d'être créée, et l'application a dû être développée à toute vitesse pour permettre de montrer à de potentiels investisseurs que le concept est viable : La première version du projet a donc été réalisée à l'aide du Framework PHP Symfony en version 3.1 afin de leur présenter un MVP (Minimum Viable Product).

Objectifs du projet :

L'entreprise a réussi à lever des fonds pour permettre le développement de l'entreprise et surtout de l'application.

Il faut désormais améliorer la qualité de l'application, notamment en :

- Implémentant de nouvelles fonctionnalités.
- Corrigant quelques anomalies.
- Implémentant des tests automatisés.
- Analysant le projet grâce à des outils permettant d'avoir une vision d'ensemble de la qualité du code et des différents axes de performance de l'application.

Environnement technique du projet :

[Home](#) / [What is Symfony](#) / [Symfony Releases](#) / [Symfony 3.1 Release](#)

Symfony 3.1 Release

Status:	Released:	End of bug fixes:	End of security fixes:
Unmaintained	May 2016	January 2017	July 2017

Latest version: 3.1.10 [Code](#) [Documentation](#) [Changelog](#) [New features](#)

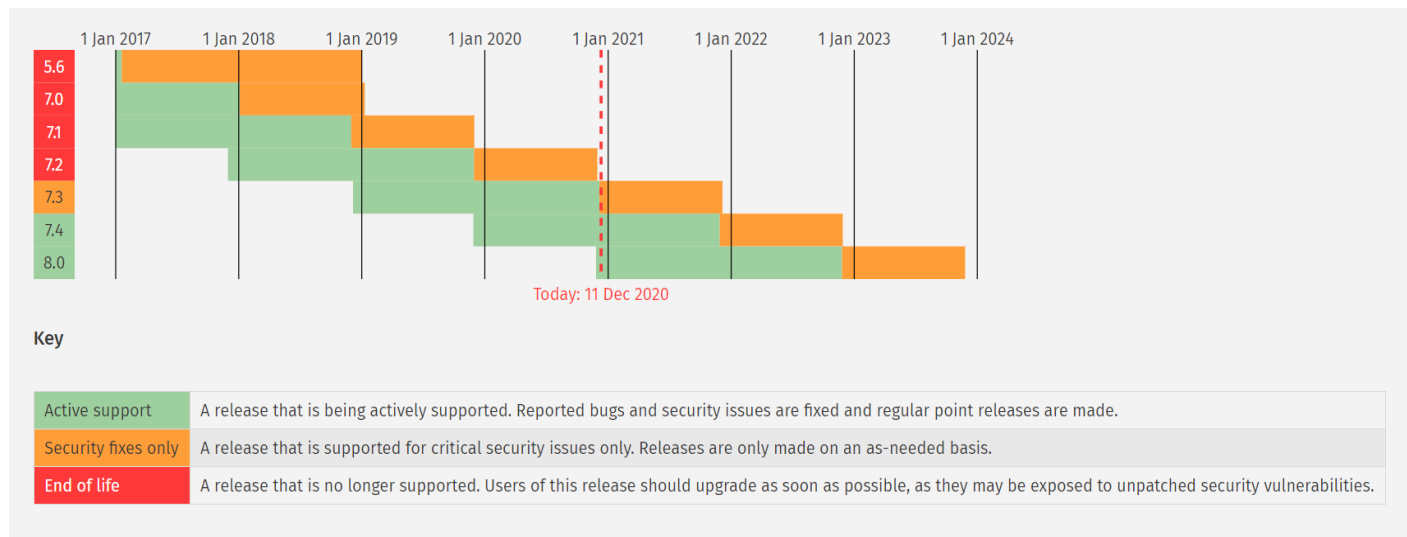
This version is no longer maintained. Consider upgrading to Symfony 5.2.

Tout d'abord en se rendant sur le site officiel de Symfony nous pouvons obtenir des informations concernant la version utilisée pour le projet TodoList, à savoir la version 3.1. Cette version date donc de 2016 et n'est plus maintenue depuis environ quatre ans.

Cela entraîne d'ailleurs le risque de dépréciation de plusieurs composants actuellement utilisés par cette version du framework.

Le site officiel nous conseille donc vivement de passer tout projet fonctionnant sur cette version vers une version plus récente telle que la version 5.2.

De plus la version 3.1 fonctionne uniquement sur des versions PHP qui ne sont plus maintenues.



La version 3.1 est compatible au maximum avec la version PHP 7.2 qui est passée le 11 décembre 2020 au statut « En of life ».

Cela représente tout d'abord un énorme risque concernant la sécurité pour l'application : Les hébergeurs recommandant fortement de rester sur les versions bénéficiant toujours du support.

De plus cela joue aussi énormément sur les performances du serveur et permet ainsi de charger plus rapidement les pages web.

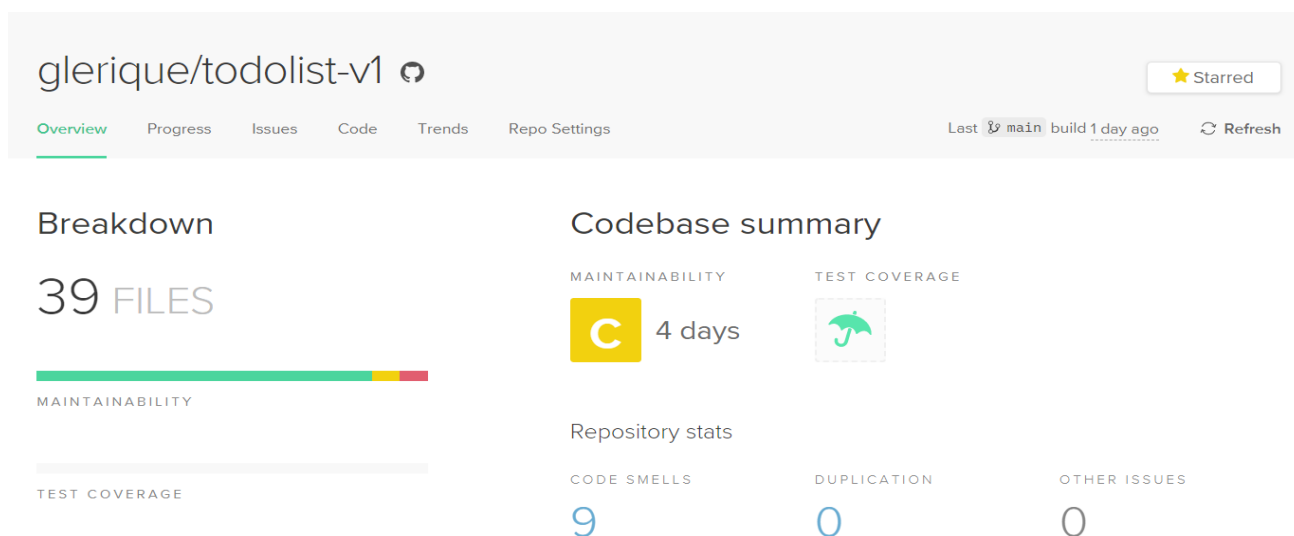
Qualité et maintenabilité du code :

Une analyse automatisée de la qualité du code a été réalisé via l'outil CodeClimate.

Cette dernière permettant de mettre en lumière les différents problèmes pouvant être liés à la qualité du code et la performance de l'application.

La note de C a été obtenue : L'idéal étant d'obtenir une note de A ou au minimum de B.

L'analyse met donc en avant des soucis de complexité de code : Certaines fonctions présentes dans la classe `SymfonyRequirements`, propre à la version `Symfony 3.1` souffrent d'un nombre trop important de lignes et d'arguments.



Analyse de l'architecture de l'application.

Malgré de bons résultats aux niveau de la duplication et du respect de l'architecture MVC, l'on peut se rendre compte en parcourant le projet que :

- Ce dernier présente une architecture obsolète due à Symfony 3.1 avec notamment l'ensemble du code métier présent dans le sous-dossier **AppBundle** du dossier **src**.
- De la présence de la logique métier dans les Controllers, alors que la bonne pratique conseil de la placer dans des Services afin d'assurer une meilleure maintenabilité et compréhension du code.

Choix de la version du nouveau projet :

L'utilisation d'une version stable du framework, dite LTS (Long Term Support) est souvent recommandée pour garantir la pérennité du développement d'une application : La dernière version LTS actuellement maintenue est la version 4.4.

Mais afin de suivre les recommandations du site officiel de Symfony et pour profiter pleinement des améliorations du framework, nous avons opté pour la version la plus récente : La dernière version stable étant la 5.2.

Notre projet étant amené à évoluer, il sera mis à jour régulièrement par une équipe de développeurs. Dans cette optique, il est préférable de partir sur la 5.2.

Sa durée de vie étant plus courte, l'équipe de développeurs sera tenue de faire évoluer les versions au fur et à mesure de leurs sorties tout en implémentant de nouvelles fonctionnalités. Ainsi le projet sera toujours à jour, sécurisé et les toutes dernières fonctionnalités du framework pourront être utilisées.

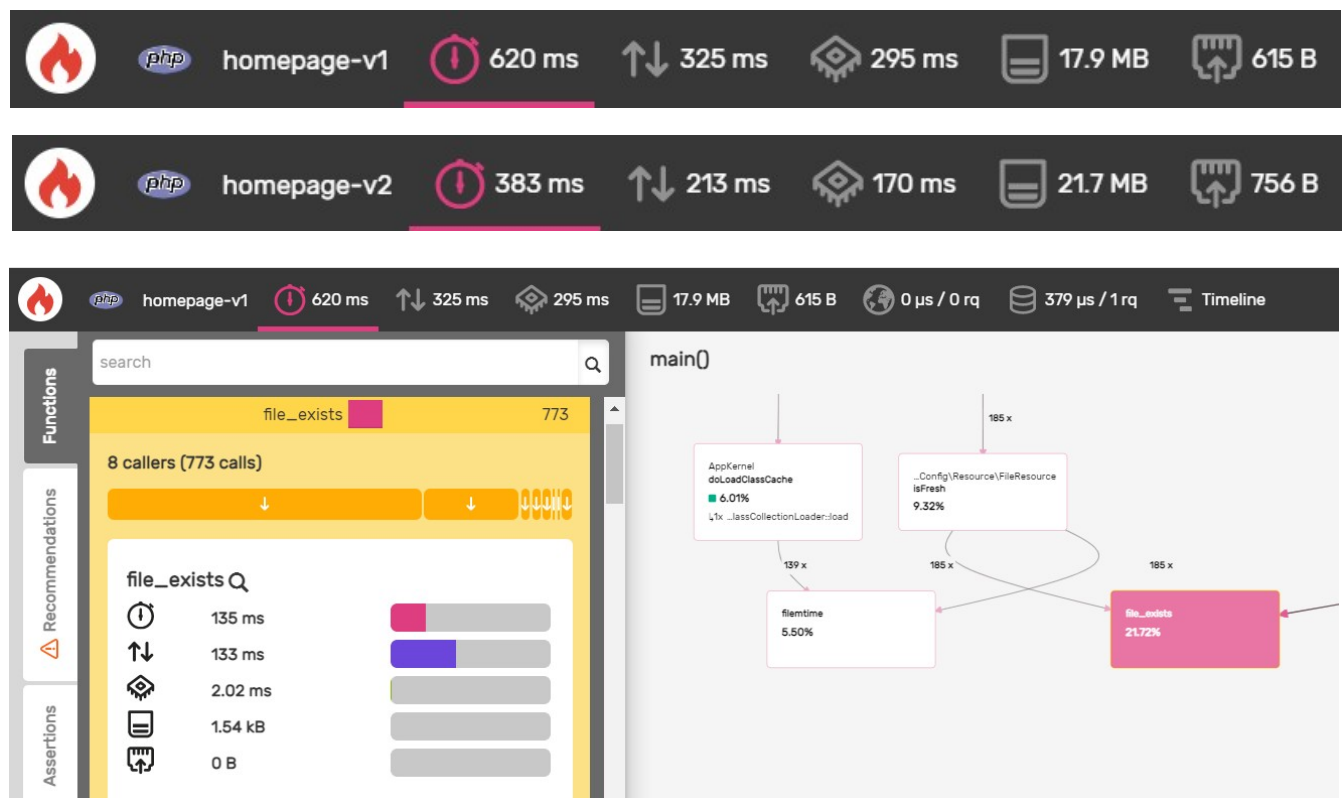
Autre point positif, une future migration coûtera moins cher si elle est faite au fil de l'eau plutôt que d'une version bénéficiant d'une autre architecture.

Ainsi il sera possible d'effectuer quelques mises à jour mineures tout en continuant à faire évoluer l'application et de migrer simplement vers la prochaine LTS. Cette dernière devrait d'ailleurs sortir d'ici deux ans et la durée de support sera plus conséquente par rapport au choix de la version 4.4.

Mesures de performance avec blackfire.io :

L'audit de performance a été réalisé à l'aide de l'outil Blackfire.io. Ce dernier a permis de produire des analyses précises de notre projet. Nous avons donc pu évaluer les performances de l'ancien projet et ensuite les comparer avec la dernière version mis en place par nos soins.

Tout d'abord la première série de mesures a mis en évidence deux soucis importants : Un énorme temps de réponse : Plus de 300 ms et des soucis provenant du nombre d'appels très élevés des fonctions `file_exists` et `includeFile` de l'autoloader de Composer.



Afin de remédier à ces deux soucis et pour ainsi ne pas fausser les analyses :



























- L'utilisation de la commande **composer dump-autoload --o** a permis de régler le problème d'appel.
- L'utilisation d'OPCache dans les fichiers `php.ini` a permis de réduire les temps de réponses de l'application.

Ensuite pour avoir une bonne vue d'ensemble, nous avons choisi quatre routes de l'application pour effectuer les analyses et nous avons ainsi obtenu les mesures suivantes :

- la page par défaut de l'application, à savoir la route /login
- la route /login_check qui permet aux utilisateurs de se connecter
- la page d'accueil une fois connecté avec la route /homepage
- la liste des tâches avec la route /tasks



























Les mesures ci-dessous montrent les mesures prises avant et après le passage en Symfony 5.2 ainsi que leurs comparatifs : Nous nous sommes particulièrement intéressés au temps de chargement de la page et à la mémoire allouée.

Mesures de la route login:

	 login-v1	 195 ms	 120 ms	 75.2 ms	 5.54 MB	 0 B	 0 µs / 0 rq	 0 µs / 0 rq
	 login-v2	 94.7 ms	 42.9 ms	 51.8 ms	 5.34 MB	 0 B	 0 µs / 0 rq	 0 µs / 0 rq
	Comparison	 -51%	 -64%	 -31%	 -3.6%	 +0%	 +0% / +0 rq	 +0% / +0 rq



























Pour la route /login, nous obtenons une baisse de 51 % du temps de chargement et de 3,6 % de la mémoire allouée.

Mesures de route login_check :

	 login_check-v1	 1.6 s	 498 ms	 1.1 s	 6.34 MB	 634 B	 0 µs / 0 rq	 8.05 ms / 1 rq
	 login_check-v2	 827 ms	 130 ms	 697 ms	 6.62 MB	 775 B	 0 µs / 0 rq	 10.3 ms / 1 rq
	Comparison	 -48%	 -74%	 -37%	 +4.42%	 +22.2%	 +0% / +0 rq	 +27.5% / +0 rq

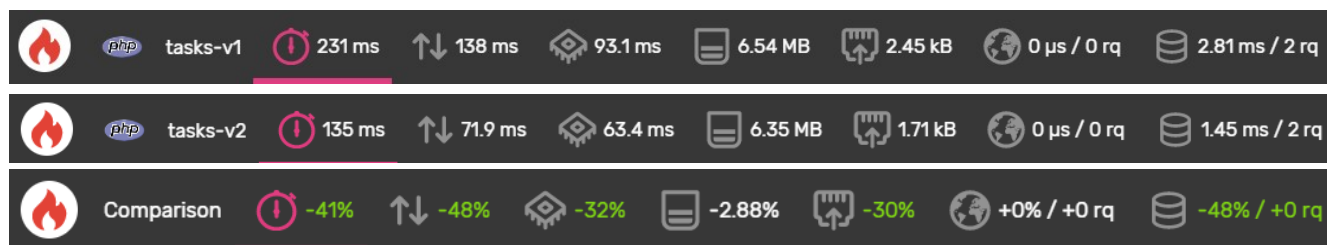
Après comparatif de cette route, nous constatons une baisse de 48 % pour le temps de réponse.

Mesures de la route homepage :

	 homepage-v1	 208 ms	 127 ms	 81.3 ms	 6.41 MB	 615 B	 0 µs / 0 rq	 346 µs / 1 rq
	 homepage-v2	 114 ms	 56 ms	 58 ms	 6.22 MB	 756 B	 0 µs / 0 rq	 507 µs / 1 rq
	Comparison	 -45%	 -56%	 -29%	 -2.96%	 +22.9%	 +0% / +0 rq	 +46.5% / +0 rq

Pour la page d'accueil une baisse de 45 % du temps de chargement a été observé, ainsi qu'un gain de 2,96 % de la mémoire allouée.

Mesures de la route tasks:



Enfin pour la route /tasks, une baisse de 41 % du temps de chargement est observé et une baisse de 2,88 % de la mémoire allouée.

Comme on peut le constater grâce aux optimisations, le temps de chargement des différentes routes a été drastiquement amélioré : Des baisses allant de 41 % à 51 % ont été observées. Cela confirme une meilleure réactivité de l'application et donc de l'amélioration des performances à l'issue de la migration.

Tests de couverture de code de l'application :

Après la migration vers Symfony 5.2.1, l'implémentation des nouvelles fonctionnalités et l'application de certains correctifs : Des tests unitaires et fonctionnels ont été mis en place pour s'assurer que le fonctionnement de l'application est bien en adéquation avec les demandes.

D:\symfony\todolist-v2\src / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div><div></div></div>	71.43%	145 / 203	<div><div></div></div>	90.57%	48 / 53	<div><div></div></div>	76.92%	10 / 13
Controller	<div><div></div></div>	100.00%	68 / 68	<div><div></div></div>	100.00%	14 / 14	<div><div></div></div>	100.00%	4 / 4
DataFixtures	<div><div></div></div>	0.00%	0 / 42	<div><div></div></div>	0.00%	0 / 2	<div><div></div></div>	0.00%	0 / 1
Entity	<div><div></div></div>	100.00%	47 / 47	<div><div></div></div>	100.00%	27 / 27	<div><div></div></div>	100.00%	2 / 2
Form	<div><div></div></div>	100.00%	13 / 13	<div><div></div></div>	100.00%	3 / 3	<div><div></div></div>	100.00%	2 / 2
Repository	<div><div></div></div>	100.00%	4 / 4	<div><div></div></div>	100.00%	2 / 2	<div><div></div></div>	100.00%	2 / 2
Security	<div><div></div></div>	92.86%	13 / 14	<div><div></div></div>	66.67%	2 / 3	<div><div></div></div>	0.00%	0 / 1
Kernel.php	<div><div></div></div>	0.00%	0 / 15	<div><div></div></div>	0.00%	0 / 2	<div><div></div></div>	0.00%	0 / 1

Legend

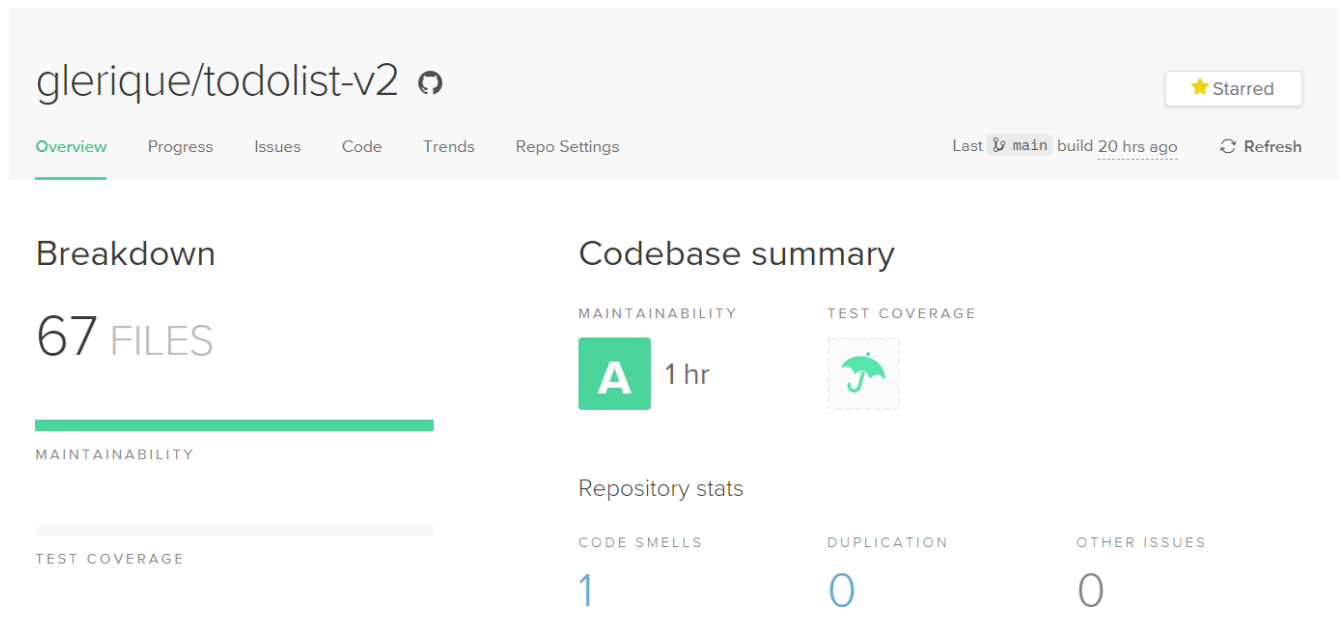
Low: 0% to 50% Medium: 50% to 90% High: 90% to 100%

Generated by php-code-coverage 6.1.4 using PHP 7.3.12 with Xdebug 3.0.2 and PHPUnit 7.5.20 at Tue Jan 26 13:16:34 UTC 2021.

Le taux de couverture supérieur à 70 % a été respecté : Les cas explicités dans le cahier des charges sont couverts par cette série de tests.

Axes d'amélioration :

Suite à la migration de l'application avec ses nouvelles fonctionnalités, une nouvelle analyse de la qualité du code a été réalisée via l'outil CodeClimate.



La note de A a ainsi été obtenue : Le projet est donc passé de C à A, ce qui permet de mettre en valeur une nette réduction de la dette technique de l'application.

Afin d'orienter les futurs développements, voici une proposition de plan d'action regroupant plusieurs axes d'améliorations :

- Créer des Classes services afin de retirer la logique des contrôleurs et retravailler le code métier de l'application en utilisant l'injection de dépendance.
- Améliorer les pages de gestion des tâches et des utilisateurs avec notamment un système de pagination pour mieux organiser le nombre de résultats : Cela permettra de générer moins d'objet et donc améliorera les temps de réponse de l'application.
- Améliorer le système de message d'erreurs et flash de l'application, en les rendant systématique à chaque opération de l'utilisateur (Validation d'une tâche, création d'une tâche, inscription etc ...).

- Mettre en place un lien vers la page de gestion des utilisateurs pour les administrateurs : La route `/users` n'étant pour l'instant reliée à aucun lien dans l'application.
- Ajouter la possibilité de supprimer les utilisateurs en ayant le rôle d'administrateur.
- Améliorer la sécurité des comptes utilisateur en ajoutant la vérification et la validation de l'adresse e-mail lors de l'inscription.
- Renforcer la sécurité de la classe User en ajoutant une contrainte d'unicité sur le username.
- Permettre aux utilisateurs de modifier leurs informations et leurs mot de passe sans être administrateur.
- Améliorer l'aspect graphique de l'application et le responsive design avec notamment une version plus récente de bootstrap : L'utilisation de thèmes Bootswatch pourrait être une bonne opportunité pour la future équipe de développeurs.