ToDo & Co – TodoList

Audit de qualité du code & performance

Table des matières

1. Contexte

1.1 Présentation

L'application TodoList appartient à la jeune startup ToDO & Co. L'application a dû être développée à toute vitesse pour présenter le concept à de potentiels investisseurs. Suite à la présentation, l'entreprise a réussi à lever des fonds pour le développement de l'application et son expansion.

1.2 Urgence de version

L'application a été mise à jour vers une version plus ressente par souci de maintenance, mais aussi de sécurité. Il faut savoir que la version 3.1 de symfony n'est plus maintenue, il était donc normal de faire une mise à jour avec urgence. La version 4.4 a été choisie pour une plus longue période de maintenance (https://symfony.com/releases/4.4).

Application avant modification

Symfony 3.1.10
php en 5.5.9
doctrine-bundle 1.6
doctrine-orm 2.5
database MySql

Application après modification

Symfony 4.4.*
php en 7.1.3
doctrine-bundle 2.0.7
doctrine-orm 2.7

1.3 Note importante

Tous les tests et informations fournis dans ce document sont basés après le changement de versions de symfony.

2. Qualité du code

2.1 Analyses de code

Pour faire l'analyse du site je me suis appuyée sur 2 sites différents. Codacy, pour les erreurs de code et les bonnes pratiques. Et Code Climate, pour la partie maintenabilité mais aussi la qualité du code. Les deux sites sont très similaires au premier abord, mais une deuxième vérification du code est toujours bénéfique, et les deux analyses sont grandement complémentaires.

2.1.1 Rapport Codacy

Selon Codacy, les statistiques de complexité et de duplication de code sont à 0 %, le nombre d'issues est de 2 %. La note globale du site est « A », ce qui est la meilleure note obtenable. Ce qui, dans la globalité, présente une très bonne qualité de code. À savoir : le grade est calculé sur le nombre d'issues pour 1k lignes de code.

Les différentes issues sont répertoriées par catégories, mais aussi par niveau, elles représentent les problèmes de code venant du site.

Info : Le type de problème le moins critique apparaîtra en bleu ; par exemple, les problèmes de style de code.

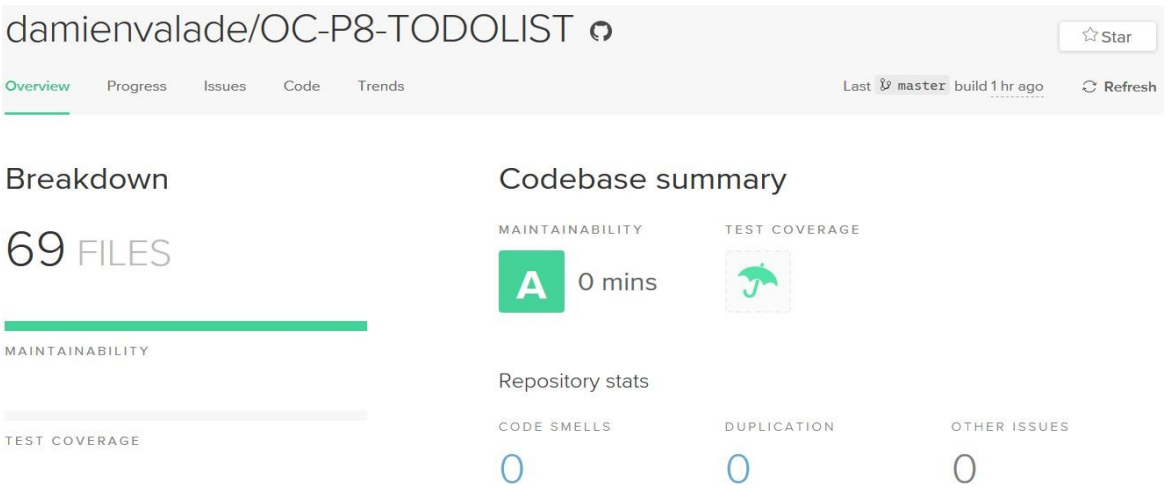
Avertissement : ce type de problème apparaîtra en jaune. Vous devez être prudent avec ceux-ci, ils sont basés sur les normes et les conventions du code.

Erreur : les types de problèmes les plus dangereux s'affichent en rouge. Prenez le temps de les corriger, bien que le code puisse s'exécuter, ces problèmes montrent que le code est très susceptible de poser des problèmes. Ces problèmes sont sujets aux bogues et / ou peuvent avoir de graves problèmes de sécurité et de compatibilité.

Une configuration des exclusions de fichiers a été faite pour ne pas prendre en comptes les lignes de code généré par symfony et les différentes librairies utilisées.

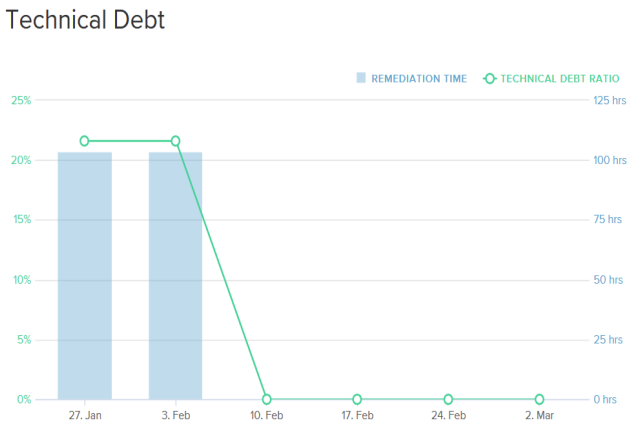
Dashboard codacy du projet : https://app.codacy.com/manual/damienvalade/OC-P8-TODOLIST/dashboard?bid=16465333

2.1.2 Rapport CodeClimate



Selon Code Climate, la maintenabilité de l'application est optimal, car il n'y a pas de mauvaises pratiques (Code Smells), duplications et autres problèmes. Ce qui représente les meilleures notes possibles à avoir sur son site.

Une configuration des exclusions de fichiers a été faite comme pour Codacy, pour ne pas prendre en comptes les lignes de code généré par symfony et les différentes librairies utilisées.

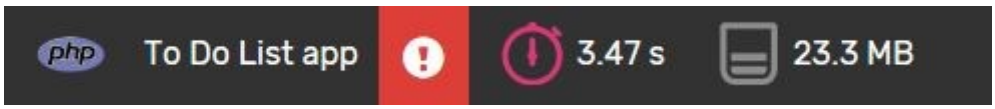


On peut voir sur le site, ce schéma. Il représente l'évolution de la dette technique du projet. Il faut savoir que le projet a été suivi depuis le début, même avant la mise à jour du site. On peut aisément dire que la mise à jour de symfony a réglé beaucoup de soucis.

Dashboard CodeClimate du projet : https://codeclimate.com/github/damienvalade/OC-P8-TODOLIST

2. Performances

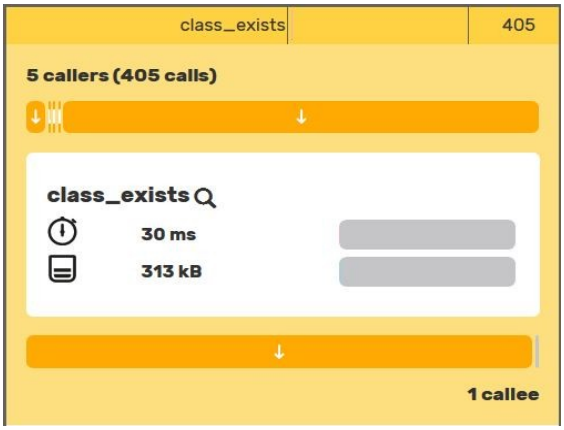
2.1 Analyse Blackfire

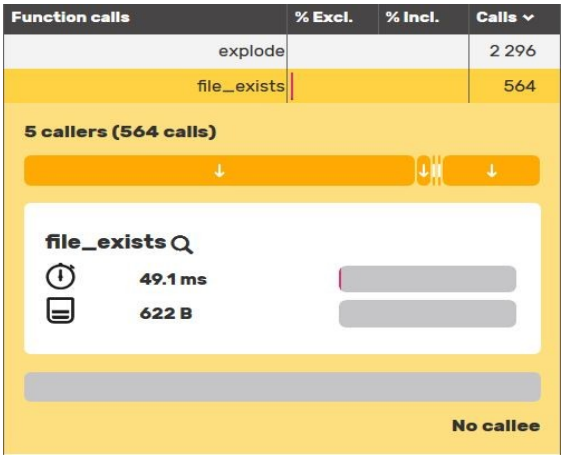


L'analyse de Blackfire donne : le temps de chargement du site, la mémoire utiliser pour son chargement, les fonctions utilisées et son nombre d'appels. Chaque fonction est détaillée avec son temps d'exécution et la mémoire utilisée. Un schéma est aussi consultable pour suivre le plan d'exécution du site, avec une couleur spécifique pour les fonctions les plus gourmande. Les tests ici présents, sont faits avec la licence gratuite de blackfire, ce qui limite l'analyse du site. À savoir: avec la version payante des recommandations d'optimisation sont proposés.

Function calls	% Excl.	% Incl.	Calls
explode			2 296
file_exists			564
class_exists			405
…sLoader::findFileWithExtension			350
…Autoload\ClassLoader::findFile			350
…DebugClassLoader::checkClass			344
…lassLoader::checkAnnotations			340
…r\Cloner\VarCloner::castObject			295
spl_autoload_call			233
…r\DebugClassLoader::loadClass			226
…IDevDebugContainer::{closure}			107
…ebugClassLoader::loadClass@1			76
spl_autoload_call@1			76
…ug\WrappedListener::__invoke			38
…ebugClassLoader::loadClass@2			32
…EventDispatcher::sortListeners			32
spl_autoload_call@2			32
file_get_contents			30
…ceptionCaster::castFrameStub			26
…xceptionCaster::extractSource			24
…Injection\Container::getService			22

Lors de la première analyse j'ai pu constater un nombre très important d'appel des fonctions file_exist et class_exist. Qui à elles seul font : 41,8 ms + 29,5 ms, ainsi que 552B + 313kB. Ce qui m'a amené vers l'optimisation de l'autoloader de composer.





2.2 Optimisation

Pour optimiser cette partie de l'application il faut exécuter une simple commande dans le terminal à la racine du projet :

```
composer dump-autoload --no-dev –classmap-authoritative
```

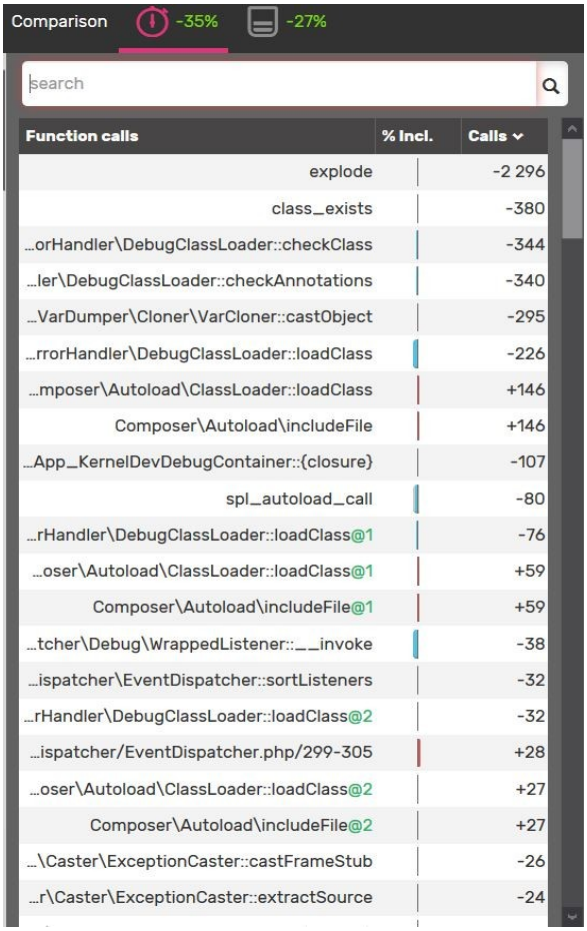
Cette commande sert à mettre en cache les classes utiles à l'application, toutefois, ci de nouvelles classes sont ajoutés il faudra absolument relancer cette même commande !

- **--no-dev** exclut les classes qui ne sont nécessaires que dans l'environnement de développement (c'est-à-dire les dépendances require-dev et les règles autoload-dev)
- **--classmap-authoritative** crée un mappage de classe pour les classes compatibles PSR-0 et PSR-4 utilisées dans votre application, et empêche Composer d'analyser les classes qui ne se trouvent pas dans le mappage de classe

L'autoloader charge automatiquement les classes utiles à l'application. Dans le cas de composer, l'autoloader est dans un dossier « vendor » à la racine du projet. Avec la commande écrite un peu plus haut, des fichiers sont créés pour faire une carte des classes à charger pour l'application, ce qui évite de charger tous les chemins à chaque chargement de pages.

Documentation : https://getcomposer.org/doc/articles/autoloader-optimization.md

2.3 Bilan



Voici les résultats de l'optimisation de l'autoloader, le temps a eu une baisse de -35 % et la consommation de données -27 %. Beaucoup de fonctions ont eu une réduction significative d'appel, voir même n'est plus du tout appeler.

2.4 Pistes d'axes d'améliorations

Différentes optimisations simples à faire peuvent être mis en place, comme l'activation d'OPCache, la modification de realpathCache, l'utilisation de varnish. Mais bien entendu, l'optimisation des performances passe par une configuration du serveur optimale.

2.4.1 OPCache

« OPcache améliore 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. » (src :https://fr.docs.wp-rocket.me/article/675-quest-ce-que-opcache)

Liens pour optimisation par OPCache : https://www.ekino.fr/articles/php-comment-configurer-utiliser-et-surveiller-opcache

2.4.2 RealpathCache

« RealpathCache, lorsqu'un chemin relatif est transformé en son chemin réel et absolu, PHP met en cache le résultat pour améliorer les performances. Les applications qui ouvrent de nombreux fichiers PHP, tels que les projets symfony sont beaucoup impactés par ces options-là »

Liens pour optimisation avec RealpathCache: https://symfony.com/doc/current/performance.html

2.4.3 Varnish

« Varnish est un accélérateur HTTP open source puissant, capable de servir rapidement du contenu mis en cache et de prendre en charge Edge Side Include. Étant donné que le cache de Symfony utilise les en-têtes de cache HTTP standard, le proxy inverse Symfony peut être remplacé par tout autre proxy inverse. »

Liens pour Varnish : https://symfony.com/doc/current/http_cache/varnish.html

2.4.4 Hébergeur

Le choix de l'hébergeur est crucial pour proposer un site réactif et optimal, autant sur la connexion que sur la configuration. Le type de serveur est aussi très important, la responsabilité est plus tournée vers les sysAdmins.

Lien explicatif des différents choix possible : https://www.codeur.com/blog/comment-choisir-un-hebergeur-web/