ToDo & Co – TodoList

Audit de qualité du code & performance

Table des matières

1. Contexte

1.1 Présentation

L'application TodoList est créé par 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

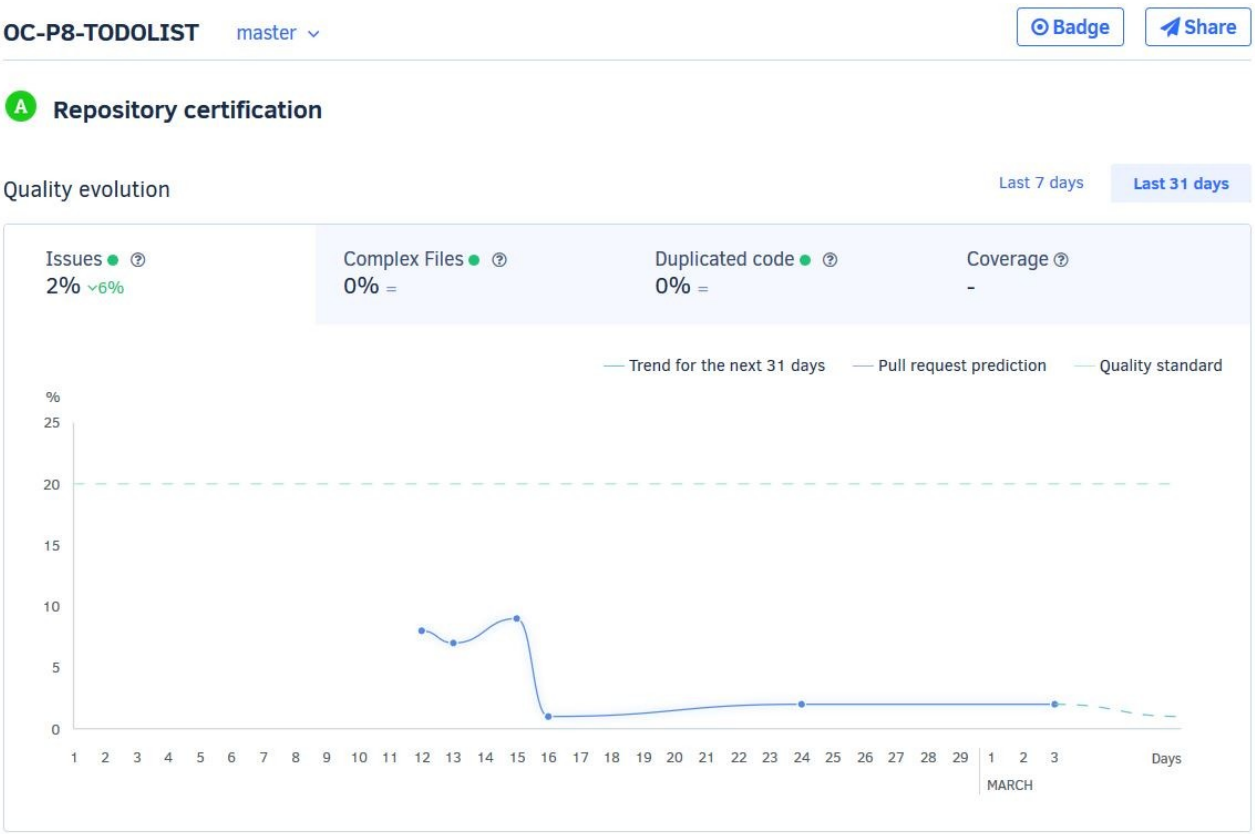
Tous les tests et informations présents dans ce document sont basés après le changement de versions de symfony.

2. Qualité du code

2.1 Analyses de code

Pour faire l'analyse du site je me suis appuyée de 2 sites différents. Codacy, pour les erreurs de code et de 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 stats de complexité et de duplication de code sont à 0 %. Et le nombre d'issues est à 2 %, ce qui représente de très bonnes statistiques. Et la note globale du site est « A ». 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 :

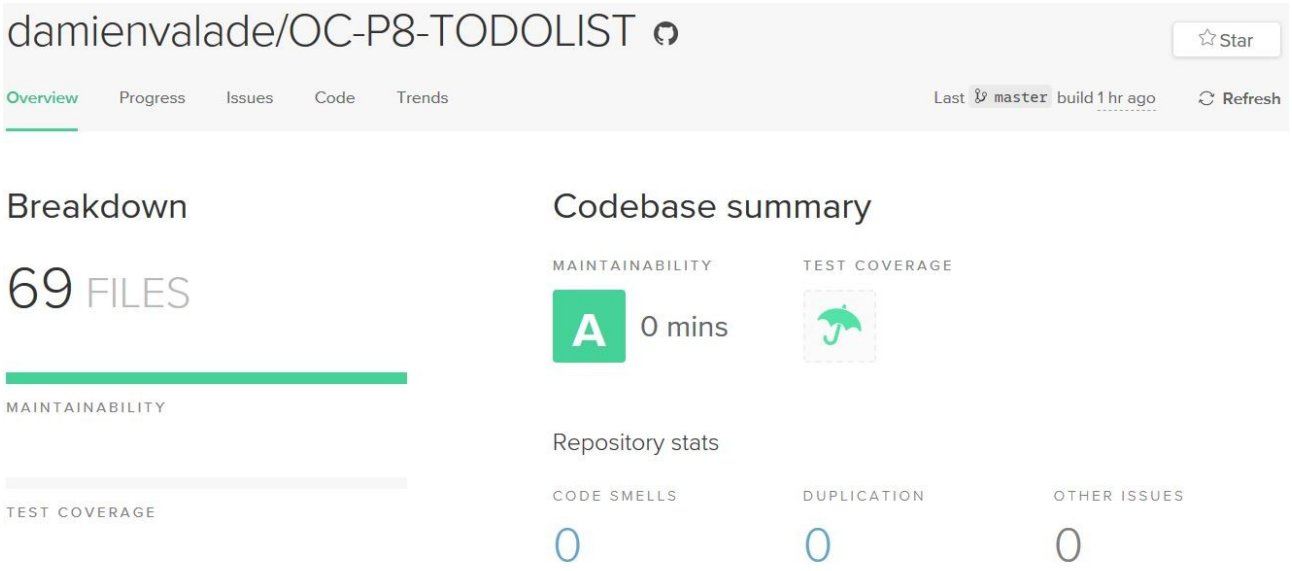
Info : Le type de problème le moins critique apparaîtra en bleu ; par exemple, les problèmes de style de code sont présentés de cette façon.

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.

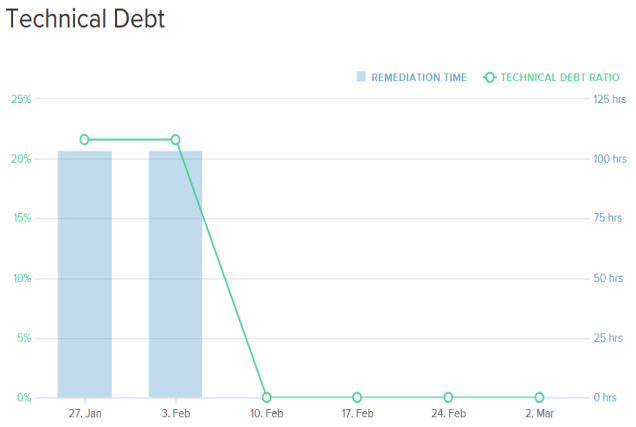
Erreur : les types de problèmes les plus dangereux s'affichent en rouge. Prenez le temps de les corriger, bien que le code puisse s'exécuter, ces problèmes montrent que le code est très susceptible de poser des problèmes. Ces problèmes sont sujets aux bogues et / ou peuvent avoir de graves problèmes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

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.

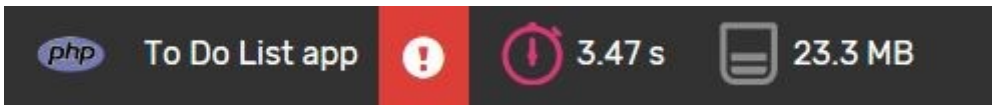


On peut voir sur le site ce schéma, qui représente l'évolution de la dette technique du projet. Il faut savoir que le projet a été suivi depuis le début de la mise à jour du site, et on peut aisément dire que la mise à jour de symfony à régler beaucoup de soucis. Il faut ajouter à cela un paramétrage des patterns à exclure, car il y a des fichiers propres à symfony qui ne sont pas toujours « qualitatif » pour CodeClimate.

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

2. Performances

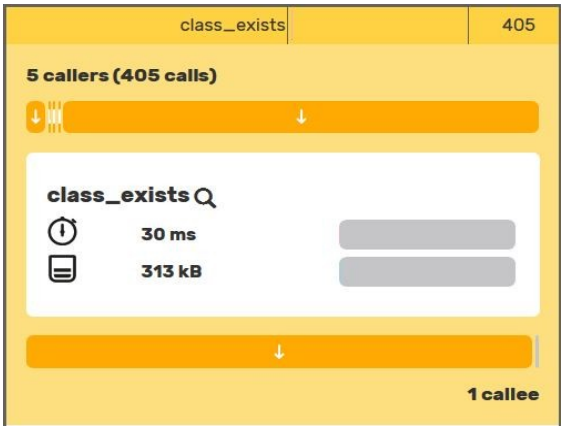
2.1 Analyse Blackfire

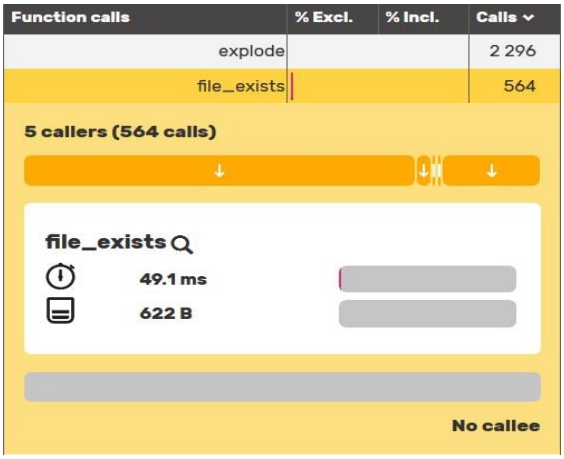


L'analyse de Blackfire donne le temps de chargement du site et la mémoire utiliser pour le chargement. Il donne aussi en précision le nombres de fois qu'une fonction est appelée, où elle est appelée, le temps d'exécution mais aussi la mémoire utilisée.

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

On peut voir un nombre important d'appel de file_exist et class_exist est fais pour l'utilisation de l'application. Qui à elles seul font 41,8 ms + 29,5 ms ainsi que 552B + 313kB

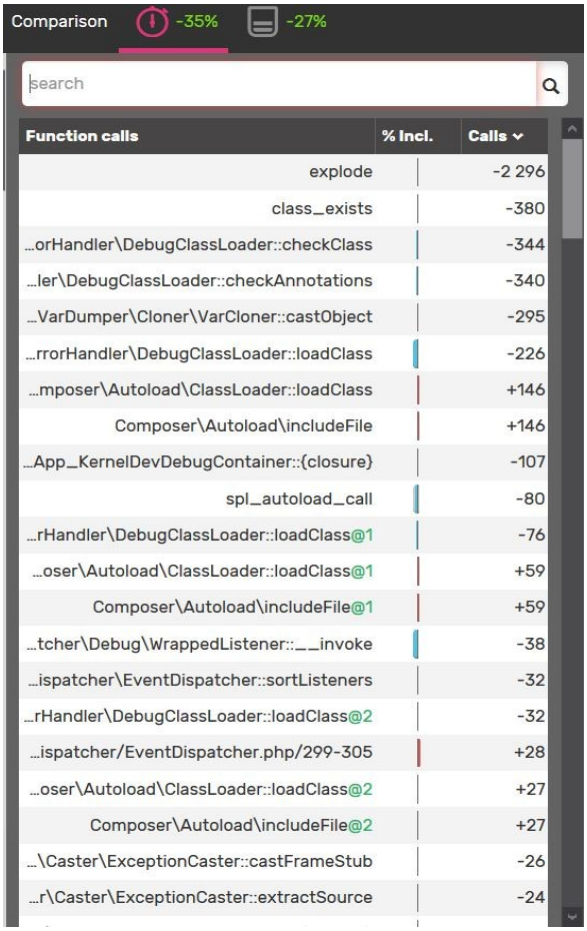




2.2 Optimisation

Pour optimiser cette partie de l'application une simple commande dans la console est utile: 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 relancer cette même commande !

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 n'est plus du tout appeler.

2.4 Pistes axes d'améliorations

Opcache est une piste à suivre :

Documentation : https://symfony.com/doc/current/performance.html