

To Do & Co

Audit qualité et performance

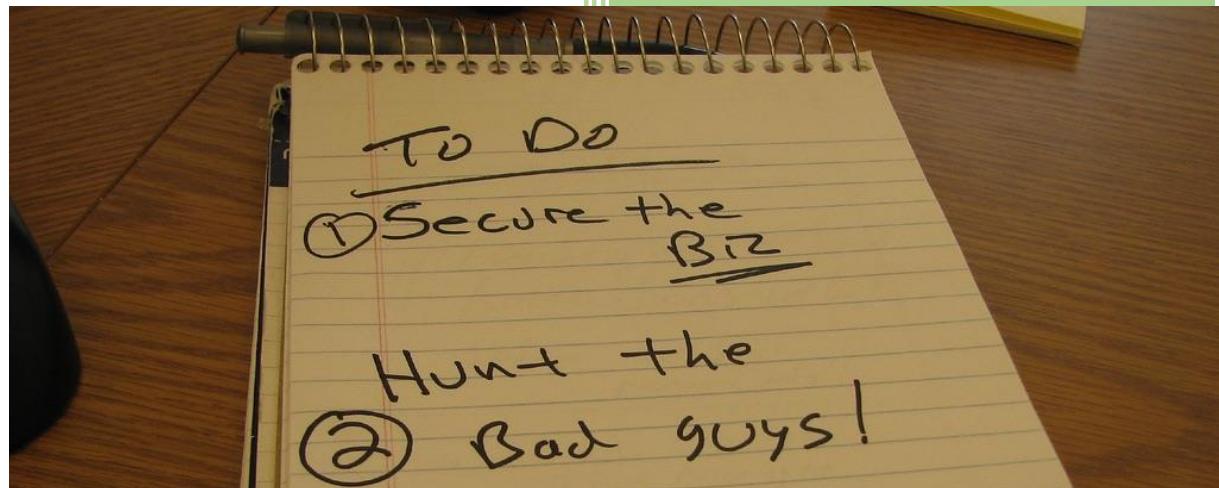
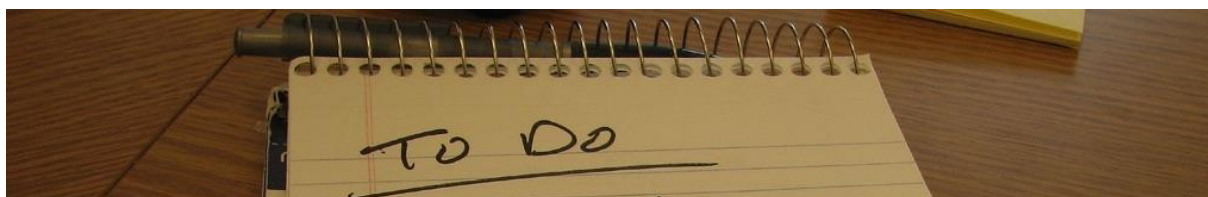


Table des matières



1. Introduction	2
2. Qualité de code initiale.....	2
1. Code coverage PHPUnit.....	2
2. Code climate	2
3. Page accueil : blackfire.	3
3. Modifications apportées.	3
1. Application.....	3
2. Nouvelles fonctionnalités.....	4
3. Optimisation de code	4
4. Qualité de code optimisée	5
1. Code coverage PHPUnit.....	5
2. Code climate	5
3. Analyse Blackfire	5

1. Introduction

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 montée, et l'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).

En tant que développeur, j'ai été en charge d'améliorer l'application en ajoutant de nouvelles fonctionnalités, en corrigeant des anomalies et en implantant des tests.

Le but de ce document est de présenter la dette technique de l'application, ainsi qu'un rapport de l'audit de qualité de code et des performances après avoir effectué les modifications demandées. Ce document a également pour but de fournir des pistes pour continuer d'améliorer l'application.





2. Qualité de code initiale

1. [Code coverage PHPUnit](#)



PHP Unit est basé sur l'idée que les développeurs devraient être capables de trouver rapidement les erreurs dans leur code nouvellement engagé et d'affirmer qu'aucune régression de code n'a eu lieu dans d'autres parties de la base de code.

C:\wamp64\www\symfony\todo-and-co\src\AppBundle / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		0.00%	0 / 167		0.00%	0 / 34		0.00%	0 / 8
 Controller		0.00%	0 / 88		0.00%	0 / 12		0.00%	0 / 4
 Entity		0.00%	0 / 60		0.00%	0 / 20		0.00%	0 / 2
 Form		0.00%	0 / 19		0.00%	0 / 2		0.00%	0 / 2
 AppBundle.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

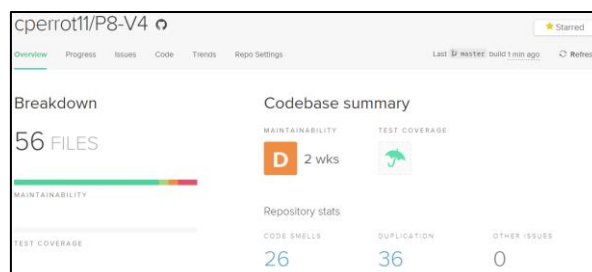
La version initiale ne contenait aucun test.

2. [Code climate](#)



Code climate permet de vérifier la qualité du code en analysant automatiquement sa vitesse d'exécution, sa consommation de mémoire et les temps d'accès base de données. Les versions payantes offrent également des propositions améliorations basées sur les standards.

Le projet initial est en symfony 3.1 et ne respecte pas les recommandations actuelles.



Les statistiques indiquent 26 améliorations et 36 duplications de code.

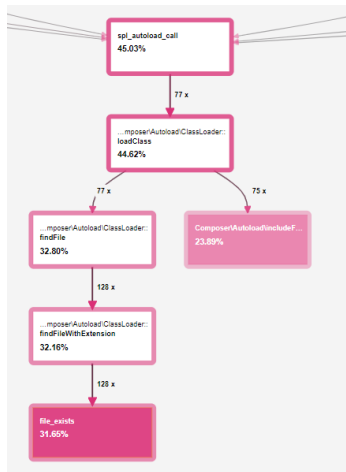
3. Page accueil : blackfire.

<https://blackfire.io/profiles/23c92109-5300-47b3-b699-41d265e95e6f/graph>



La fonction `file_exists()` est appelée 137 fois et représente 32% du temps consommé.

`IncludeFile()` de composer appelé 75 fois représente elle près de 9% du temps.



Function calls	% Excl.	% Incl.	Calls
dirname			464
file_exists			137
Composer\Autoload\ClassLoader::findFileWithExtension			128
Composer\Autoload\ClassLoader::findFile			128
spl_autoload_call			78
Composer\Autoload\ClassLoader::loadClass			77
Composer\Autoload\ClassLoader::includeFile			75
...ony\Component\EventDispatcher\EventDispatcher::addListener			42
Composer\Autoload\ClassLoader::includeFile@1			30
Composer\Autoload\ClassLoader::loadClass@1			30

3. Modifications apportées.

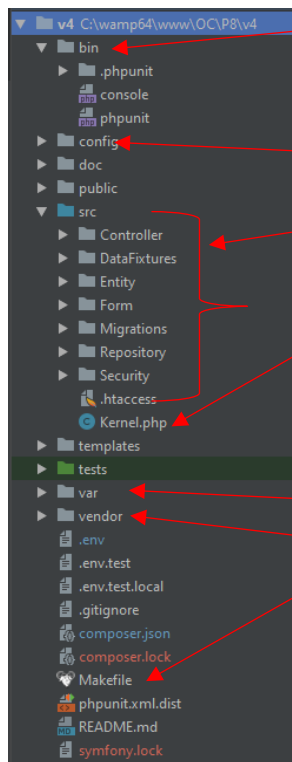
1. Application.

Des mises à jour ont été appliquées à l'application, afin de lui assurer une meilleure maintenabilité.

En effet, la première version de l'application avait été réalisée sous Symfony 3.1 qui n'est à ce jour plus maintenue. Bien que la version 5 de Symfony soit sortie, c'est la **version 4.4** qui a été retenue, afin d'éviter trop de changement, et parce qu'elle la version majeure maintenue par Sensio.

De multiples améliorations ont été apportés au niveau de l'**interface home machine** afin d'améliorer l'expérience utilisateur (message alerte, filtre tâche terminées, auteur et statut sur les tâches, etc...)

L'**architecture Flex** recommandée pour les projets Symfony 4 a été mise en place. Voici l'organisation des principaux fichiers.



- **bin** : contient l'outil console qui permet d'effectuer les tâches de routine pour créer ou gérer un projet. On retrouve l'outil PHPUnit utilisé pour les tests unitaires et fonctionnels.
- **config** : contient les fichiers de configuration propre à chaque environnement de l'application (dev, prod et test).
- **src** : Contient les classes principales, on y retrouve les contrôleurs, les entités, les fixtures (jeux de données fictifs), les formulaires et les migrations (syntaxe décrivant la base de donnée).
- **kernel.php** : le noyau de Symfony est responsable de la mise en place de tous les bundles utilisés par l'application et de leur fournir la configuration. Il crée ensuite le conteneur de service avant de servir les requêtes dans sa méthode handle ().
- **var** : contient les caches et les logs.
- **vendor** : les classes des Bundles installés via composer.json.
- **Makefile** : Fichier de script permettant d'accélérer le développement. Permet de lancer un enchainement de séquence de code console via des alias.
- **README.md** : Description de l'application visible sur Github.

2. Nouvelles fonctionnalités.

- **Sécurité** : La partie authentification a été ajoutée afin de lier la classe utilisateurs de la base de données avec l'internaute utilisant l'application. Tous les détails du fonctionnement se trouvent dans le fichier 'doc/authentification.pdf'.
- **Rôle** : Un internaute peut être anonyme, simple utilisateur ou administrateur. Cette nouvelle notion permet de limiter l'accès à certaines parties du site. Notamment la gestion des utilisateurs ou la suppression de tâche.
- **Auteur** : Chaque nouvelle tâche est maintenant associée à son propriétaire et celui-ci ne peut pas être modifié.
- **Page erreur** : Les pages d'erreur ont été personnalisées pour ne pas perturber l'internaute.
- **DataFixtures** : Le framework Faker a été utilisé pour créer 10 utilisateurs et 20 avec des données d'exemples. C'est très pratique pour manipuler l'application en mode développement.

3. Optimisation de code

La première analyse BlackFire (cf ci-dessous) a mis en évidence que l'autoloader « tourne en rond » 😊. Les règles de chargement automatique PSR-4 demandent de vérifier le système de fichier avant de résoudre définitivement le nom d'une classe. En production, ça peut être évité avec certaines configurations ([doc composer](#)):

- **Class map génération** :
 - Set "optimize-autoloader": true inside the config key of composer.json
 - Call install or update with -o / --optimize-autoloader
 - Call dump-autoload with -o / --optimize

La carte de classe accélère l'accès aux classes dont l'existence est garantie par Composer. De plus la carte est stockée en cache avec OpCache.

- **Authoritative class maps :**
 - Set "classmap-authoritative": true inside the config key of composer.json
 - Call install or update with -a / --classmap-authoritative
 - Call dump-autoload with -a / --classmap-authoritative

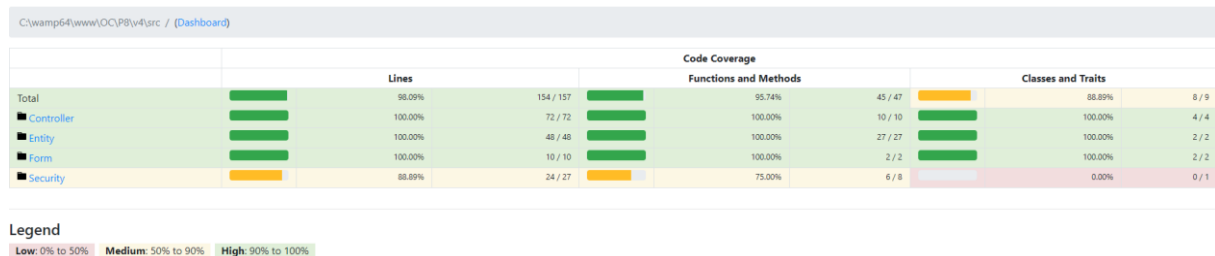
Cette option est très simple, elle dit que si quelque chose n'est pas trouvé dans le classmap, alors il n'existe pas et l'autochargeur ne devrait pas essayer de regarder le système de fichiers selon les règles PSR-4.

4. Qualité de code optimisée

Après avoir travailler sur l'optimisation du code en conformité avec les bonnes pratique actuelle, l'application a maintenant un code de bien meilleure qualité.

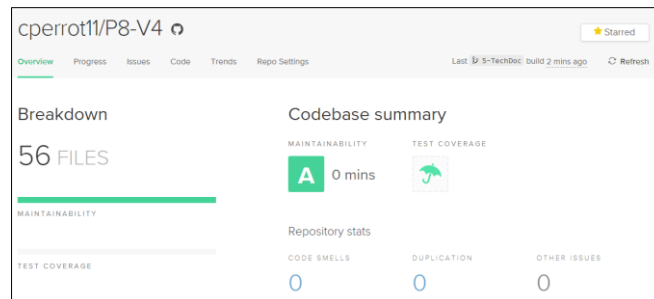
1. Code coverage PHPUnit

ToDo & Co dispose maintenant de plus de 40 codes unitaires ou fonctionnels permettant de garantir une bonne couverture de l'application (**98%**).



2. Code climate

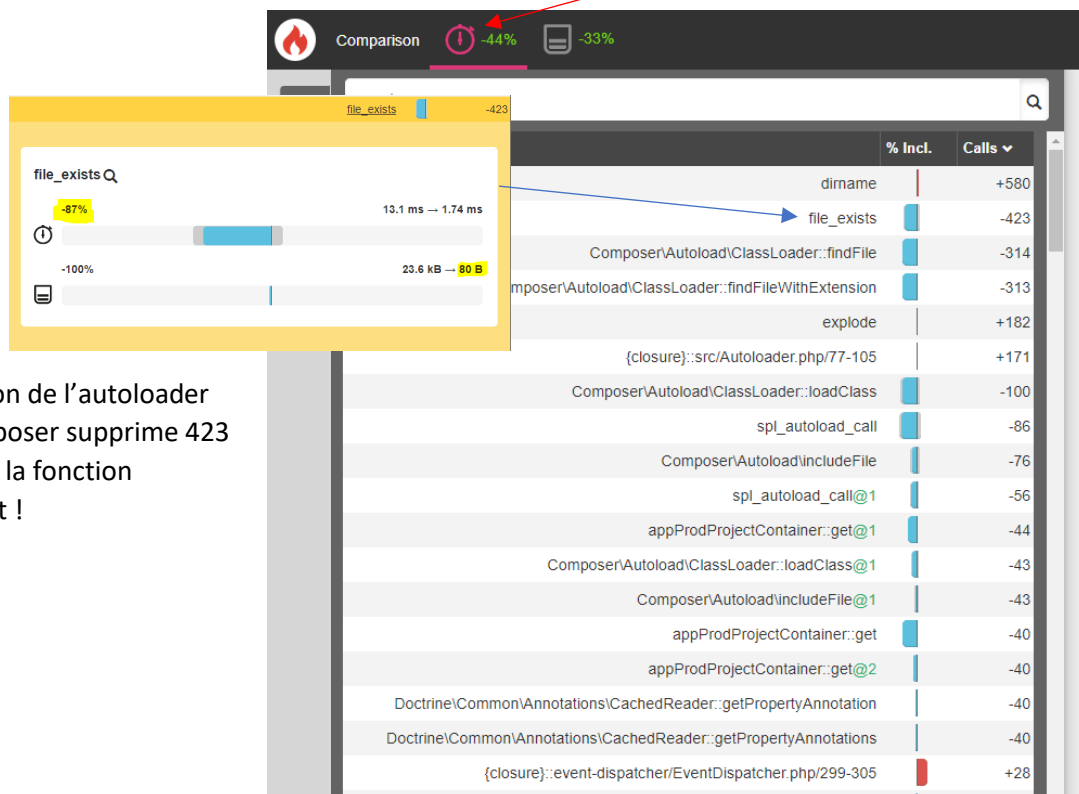
Le projet a obtenu le label A et ne nécessite aucune amélioration.



3. Analyse Blackfire

	Avant		Après		
Route	Chargement (ms)	Mémoire (MB)	Chargement (ms)	Mémoire (MB)	
/	71,9	6,37	19.5	1.87	lien
/login			21.4	1.97	lien
/tasks	85,7	6,81	39.9	2.71	lien
/tasks/create	80,7	7,02	42.4	3.39	lien
/admin/users	78,8	6,38	35.9	2.61	lien
/admin/users/create	91,3	7,47	44.6	3.75	lien

L'optimisation en matière de chargement de classe est importante, gain de 44% sur la vitesse d'exécution et 33% sur l'utilisation de la mémoire.



La gestion de l'autoloader via composer supprime 423 appels à la fonction file_exists !