



Dépôt Github : <https://github.com/magali-thuaire/oc-todolist>

AUDIT

QUALITE DE CODE ET PERFORMANCE

SOMMAIRE

1. CONTEXTE	3
2. AMELIORATIONS DE L'EXISTANT	4
2.1 MONTEES DE VERSION	4
2.2 CORRECTION D'ANOMALIES ET AJOUT DE NOUVELLES FONCTIONNALITES.....	4
2.2.1 Tâches.....	4
2.2.2 Utilisateurs	4
2.2.3 Application	5
2.3 MISE EN PLACE DE TESTS AUTOMATISES	5
3. QUALITE DE CODE	6
3.1 LIBRAIRIES PHP EN LIGNE DE COMMANDE (LINTERS)	6
3.2 REVUE DE CODE AUTOMATISEE	6
4. PERFORMANCE	7
4.1 BLACKFIRE	7
4.2 ETAT DES LIEUX.....	7
4.3 OPTIMISATIONS.....	8
4.3.1 Optimisation du chargement des classes.....	8
4.3.2 Réduction du nombre de requêtes SQL.....	8

1. CONTEXTE

La startup **ToDo & Co** a développé une application permettant de gérer les tâches quotidiennes (ToDoList).

L'application a initialement été développée avec le **framework Symfony version 3.1** (version 1.0 du projet) et se trouve actuellement dans sa **version 6.1**. Diverses modifications et améliorations ont ensuite été ajoutées à la suite de cette montée de version (version 3.0 du projet).

Ce document réalise un audit sur la **qualité de code** et sur la **performance** de la version actuelle du projet (version 3.0). Cette analyse a été réalisée avec les outils suivants :

- Amélioration de l'existant : tests automatisés via **PHPUnit**
- Qualité de code : librairies **PHP_CodeSniffer** et **PHP_CS_Fixer**, analyseur de code **Codacy**
- Performance : **Blackfire.io**

2. AMELIORATIONS DE L'EXISTANT

Suite à la récupération du projet en version initiale, les améliorations/corrections/modifications suivantes ont été réalisées.

2.1 Montées de version

Version	
Symfony 3.1	Projet initial
Symfony 3.4	Mise à jour des dépendances + corrections des dépréciations
Symfony 4.4	Montée vers PHP 7.4 + corrections des dépréciations
Symfony 5.4	Mise à jour des recettes Flex + corrections des dépréciations
Symfony 6.0	Montée vers PHP 8.1 + mise à jour des recettes Flex + corrections des dépréciations
Symfony 6.1	Mise à jour des recettes Flex + corrections des dépréciations

2.2 Correction d'anomalies et ajout de nouvelles fonctionnalités

2.2.1 Tâches

- Rattachement d'une tâche à un utilisateur
- Modification du listing des tâches non terminées pour exclure les tâches terminées
- Correction du message affiché lorsqu'une tâche non terminée est marquée comme faite
- Suppression du doublon sur le bouton permettant de créer une nouvelle tâche lorsque le listing des tâches est vide
- Demande de confirmation avant suppression d'une tâche
- Pagination sur le listing des tâches
- Création d'une page affichant le listing des tâches terminées

2.2.2 Utilisateurs

- Affectation d'un rôle (utilisateur/administrateur) lors de la création/modification d'un utilisateur
- Enregistrement du mot de passe sans stockage temporaire de la version non hachée
- Affichage en version française du message d'erreur de connexion
- Envoi d'un email permettant la modification de mot de passe
- Pagination/Recherche/Filtrage de la liste des utilisateurs
- Suppression d'un utilisateur

2.2.3 Application

- Accès par autorisation :
 - Gestion des utilisateurs uniquement par les administrateurs
 - Suppression des tâches uniquement par le propriétaire de la tâche ou par les administrateurs
 - Suppression des tâches anonymes uniquement par les administrateurs
- Sécurité CSRF sur le formulaire de connexion
- Menu de navigation pour accéder aux différentes pages































2.3 Mise en place de tests automatisés

Pour s'assurer que le fonctionnement de l'application est bien en adéquation avec les demandes, des tests automatisés ont été implémentés avec **PHPUnit**.

La mise en place de tests automatisés permet de valider la non-régression à la suite d'évolutions ou de modifications dans le code. Cela permet d'éviter les bugs. Dans ce but, les tests ont été intégrés à la version initiale du projet.

Les nouvelles fonctionnalités de l'application ont été implémentées via la **méthodologie TDD** (TestDriven Development) qui consiste à écrire le test avant de développer le code source.

Le **taux de couverture** effectif est de **87,28%**, conforme à la spécification demandée d'un minimum de 70%.

	Code Coverage							
	Lines		Functions and Methods			Classes and Traits		
Total		87.28%	295 / 338		83.64%	92 / 110		45.83% 11 / 24
Controller		88.61%	70 / 79		75.00%	15 / 20		20.00% 1 / 5
DataFixtures		0.00%	0 / 10		0.00%	0 / 1		0.00% 0 / 1
Entity		90.00%	54 / 60		91.67%	33 / 36		33.33% 1 / 3
Factory		100.00%	25 / 25		100.00%	9 / 9		100.00% 2 / 2
Form		95.45%	21 / 22		66.67%	2 / 3		66.67% 2 / 3
Manager		92.19%	59 / 64		86.67%	13 / 15		66.67% 2 / 3
Repository		81.25%	26 / 32		86.67%	13 / 15		66.67% 2 / 3
Security		82.61%	19 / 23		66.67%	4 / 6		0.00% 0 / 2
Service		91.30%	21 / 23		60.00%	3 / 5		50.00% 1 / 2
Kernel.php		n/a	0 / 0		n/a	0 / 0		n/a 0 / 0

Legend

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

Generated by [php-code-coverage 9.2.15](#) using [PHP 8.1.6](#) and [PHPUnit 9.5.20](#) at Tue Jun 14 15:08:34 CEST 2022.

```
PHPUnit 9.5.20 #StandWithUkraine

Testing
..... 65 / 81 ( 80%)
..... 81 / 81 (100%)

Time: 01:27.189, Memory: 104.50 MB

OK (81 tests, 377 assertions)
```

3. QUALITE DE CODE

3.1 Librairies PHP en ligne de commande (linters)

Pour respecter l'uniformité du code et le respect des conventions de codage (PSR 1 et 12), deux librairies PHP en ligne de commande ont été mises en place sur ce projet :

- **PHP_CodeSniffer**
 - L'exécutable **phpcs** (sniffer) analyse les fichiers PHP, JavaScript et CSS afin de détecter des violations définies par les standards de développement de PHP
 - L'exécutable **phpcbf** (fixer) corrige au maximum les erreurs détectées par **phpcs**
- **PHP_CS_Fixer** : vérifie et corrige le formatage du code **PHP**

3.2 Revue de code automatisée

Un **audit de qualité de code** a été réalisé au niveau de l'application à l'aide de l'outil **Codacy**.

Cet outil permet d'obtenir un code respectant les standards et conventions du langage PHP (PSR). Il a été utilisé et lié au **dépôt Github** du projet pour analyser le code source dès sa version initiale.

Grâce à l'analyse statique de la révision du code, **Codacy** remonte les problèmes de sécurité, de duplication et de complexité de code pour chaque commit.

Il a permis de :

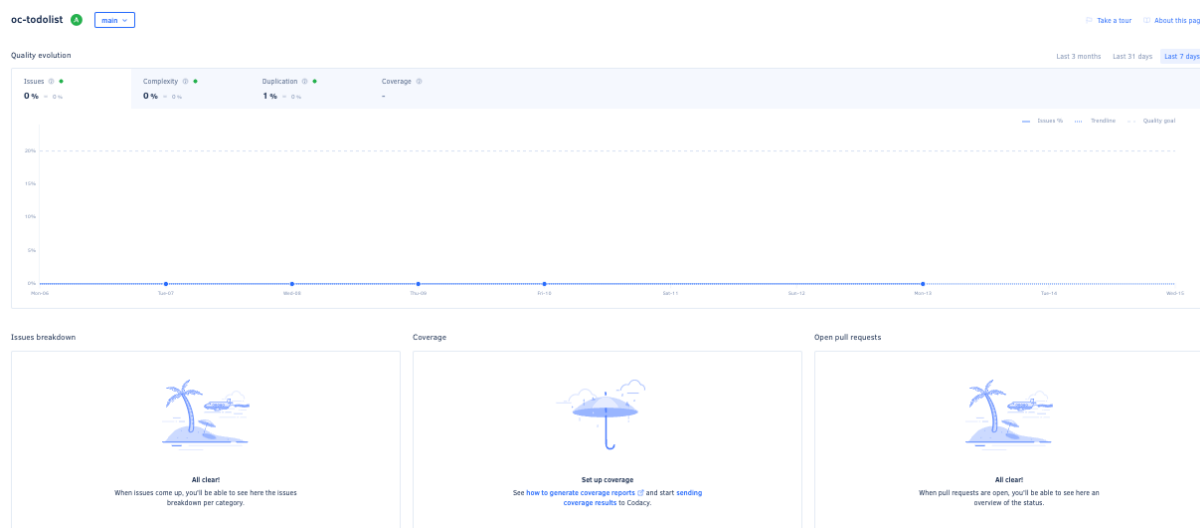
- Corriger la dette technique initiale du projet
- Vérifier que chaque modification/ajout de fonctionnalité n'entraîne pas de problème de qualité de code

Les différentes modifications de code ont été réalisées sur une branche Git secondaire puis analysées par **Codacy**. Après corrections des éventuelles erreurs, la branche secondaire a été fusionnée sur la branche principale du projet.

Une configuration sur l'exclusion des fichiers a été paramétrée pour ne pas prendre en compte les fichiers des librairies externes utilisées dans l'application.

Codacy attribue une **note globale** au projet pour évaluer la qualité du code.

Le projet présente un **badge de niveau A**.



4. PERFORMANCE

4.1 Blackfire

Afin d'analyser la performance de l'application, l'outil **Blackfire.io** développé par SensioLabs a été utilisé. Il s'agit un **profileur de performance d'application PHP**.

Il permet de mesurer la performance de l'application, c'est-à-dire de mesurer l'utilisation des ressources matérielles mobilisées pour son exécution.

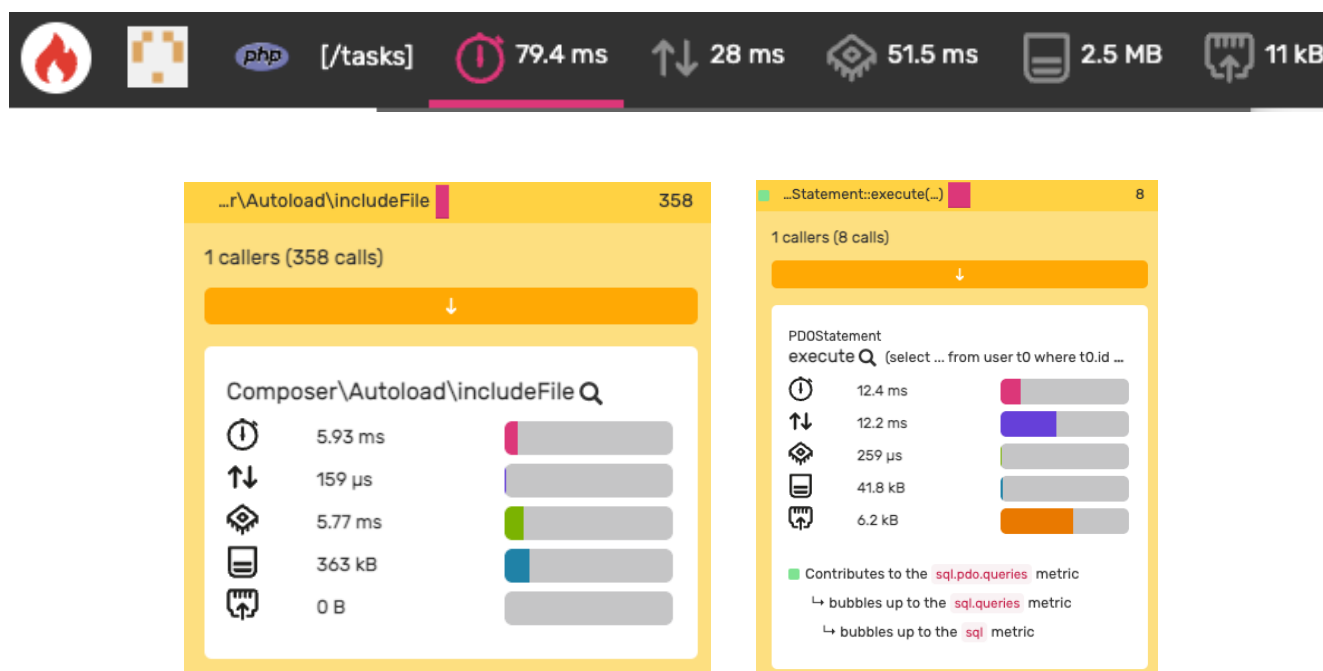
Pour chaque profilage Blackfire.io permet d'identifier différentes métriques :

- Le temps de génération de la page en PHP, découpé en temps d'accès au **processeur** et temps d'accès au **système de fichiers**
- La **mémoire** utilisée
- L'utilisation du **réseau**

Les mesures ci-après ont été réalisé en environnement de production sur le serveur local de développement.

4.2 Etat des lieux

Le profilage de la page listant les tâches à faire est le suivant :



En examinant les méthodes les plus énergivore du profil, nous pouvons identifier deux goulots d'étranglement problématiques : **Composer\Autoload\includeFile** et **PDOStatement execute**.

Recommandations blackfire.io

You should execute less SQL queries

metrics.sql.queries.count 11 <= 10

4.3 Optimisations

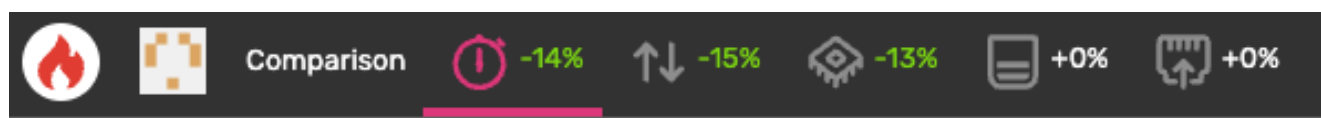
4.3.1 Optimisation du chargement des classes

Par défaut, le chargeur automatique **Composer** s'exécute relativement rapidement. Cependant, en raison de la façon dont les règles de chargement automatique PSR-4 sont configurées, il doit vérifier le système de fichiers avant de résoudre un nom de classe de manière concluante. Cela ralentit l'application, mais est utile dans les environnements de développement car lorsqu'une nouvelle classe est ajoutée, elle peut être immédiatement utilisée sans avoir à reconstruire la configuration du chargeur automatique.

En production, il est possible d'**optimiser ce chargement** en exécutant la commande suivante dans la console :

```
composer dump-autoload -optimize
```

Comparaison de profils



La comparaison des profils avant et après modification permet de valider cette optimisation qui permet de **réduire de 14%** le temps d'exécution sur la page affichant le listing des tâches à faire.

4.3.2 Réduction du nombre de requêtes SQL

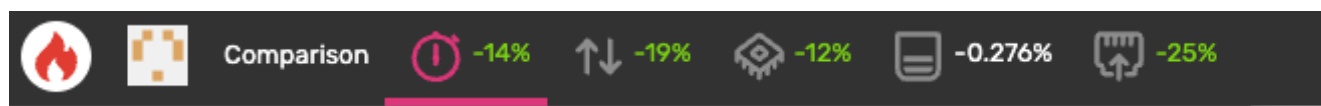
Blackfire.io préconise de réduire le nombre de requête SQL sur la page affichant le listing des tâches à faire : actuellement **11 requêtes** sont exécutées.

Par défaut Doctrine (comme de nombreux ORM) génère des classes Proxy afin de ne récupérer les relations enfant que lorsqu'elles sont demandées. On parle du problème N+1

Sur cette page, Doctrine effectue une première requête (+1) pour récupérer la liste des tâches à faire puis pour chaque tâche exécute une requête pour récupérer le propriétaire et pouvoir l'afficher (N).

La solution pour éviter cela est bien évidemment d'utiliser une **jointure SQL** afin de récupérer les informations du propriétaire de la tâche dans la même requête.

Comparaison de profils



La comparaison des profils avant et après modification permet de valider cette optimisation qui permet de **réduire de 14%** le temps d'exécution sur la page affichant le listing des tâches à faire.