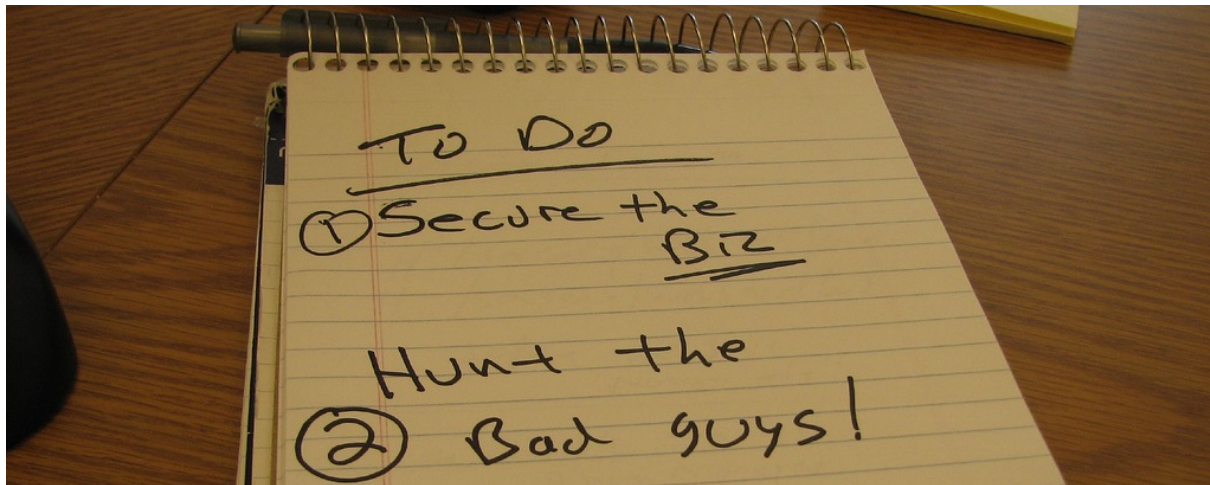


AUDIT DE QUALITÉ ET PERFORMANCE



ToDo & Co

Par Murat CAN

Mise à jour : 31/03/2021

SOMMAIRE

| | |
|---|----|
| Introduction..... | 1 |
| Mise à jour de Symfony..... | 2 |
| Etat des lieux et corrections..... | 3 |
| Méthodologie pour un code de qualité..... | 4 |
| Performance..... | 6 |
| Les points d'optimisations..... | 8 |
| Conclusion..... | 10 |

INTRODUCTION

L'objectif de ce document est de vous faire un état des lieux de la dette technique de la première version de l'application web ToDoList et en parallèle les améliorations qui ont été apportées afin de la réduire le plus possible.

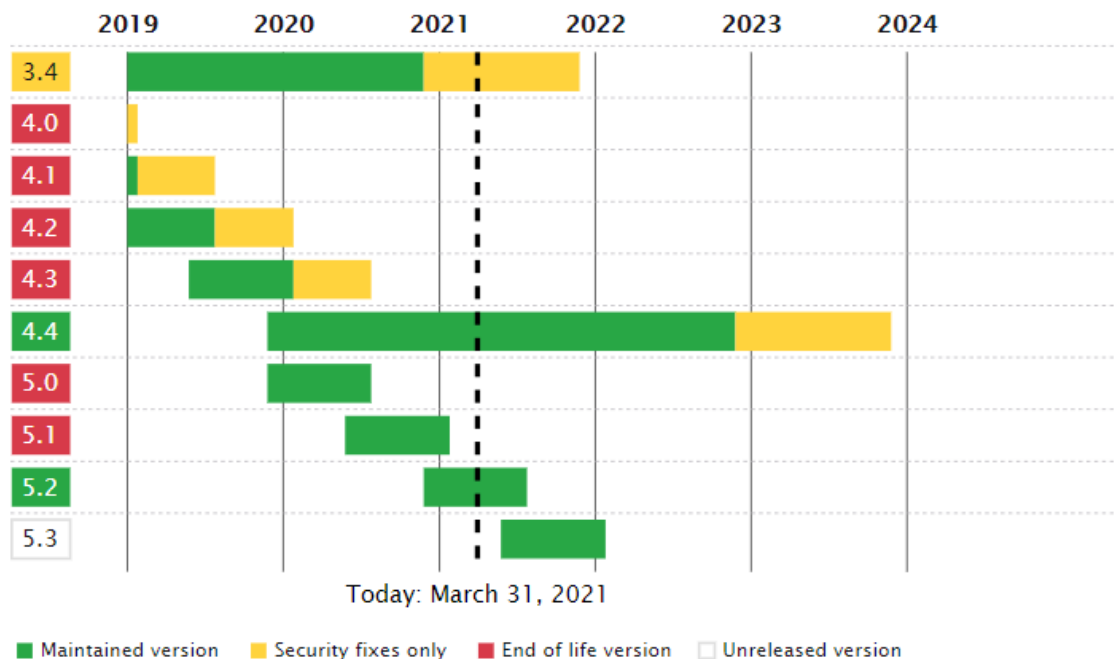
Nous allons donc réaliser un audit de la qualité du code et de la performance de l'application.

Afin de réaliser cette évaluation, je me suis basé sur mon expérience en tant que développeur pour percevoir la qualité du code et je me suis également appuyé sur des outils d'analyse de code comme CodeClimate, Travis-Ci et Blackfire.

MISE À JOUR DE SYMFONY

L'application a été réalisée avec la version 3.1 du Framework Symfony. Cette version n'est plus maintenue par SensioLabs. Vous pouvez voir sur leur site l'état de maintenance du Framework Symfony <https://symfony.com/roadmap>.

Symfony Releases Calendar



L'objectif est donc de mettre Symfony à jour vers la version 4.4. C'est la dernière version avec support à long terme. Elle sera maintenue jusqu'à la fin 2022. Il est recommandé par Symfony d'utiliser cette version pour les nouveaux projets. Comme l'objectif est de remettre à niveau l'application afin qu'elle puisse évoluer et satisfaire vos nouveaux clients. Nous avons fait le choix de la version 4.4 qui offre de meilleures performances avec PHP 7.4.

ETAT DES LIEUX ET CORRECTIONS

Dans cette partie, nous allons vous présenter un état des lieux de l'application après utilisation de l'application et analyse du code. Et dans un second temps, nous proposons une liste d'améliorations apportées à la dette technique.

ETAT DES LIEUX :

- Version 3.1 qui n'est plus maintenue.
- Nous pouvons ajouter une tâche mais elle n'est rattachée à aucun utilisateur.
- Un utilisateur peut supprimer ou éditer une tâche qu'il n'a pas créé lui même.
- Lors de la création d'un utilisateur, le formulaire ne propose pas le choix d'un rôle.
- Tout utilisateur peut se connecter, accéder à la gestion des tâches et également à la gestion des utilisateurs.
- Il n'y a pas la possibilité de supprimer un utilisateur.
- Nous pouvons accéder à la liste des utilisateurs sans être identifiés.
- Au niveau du code, il n'y a aucune couverture(ni tests fonctionnels, ni tests unitaires).
- Aucune fixture fournie avec l'application.
- Duplications de code présent dans les controllers.
- Trop de logique métier présent dans les controllers.
- Le code n'est pas optimisé selon les principes solides.
- Le code n'est pas documenté.
- Aucune documentation présente sur l'application (absence de diagrammes, fichier Readme.md non détaillé).

CORRECTIONS APPORTÉES :

- Mise à jour vers la version 4.4
- Les tâches créées sont liées à l'utilisateur en cours.
- Ajout propriété \$user dans l'entité Task.
- Les tâches plus anciennes sans auteur sont considéré comme rattaché à anonyme
- Deux rôles ont été implémentés et peuvent désormais être affectés à l'utilisateur lors de sa création: ROLE_USER & ROLE_ADMIN
- Modification du formulaire Form \ UserType en ajoutant un champ de rôles.
- La zone de gestion des utilisateurs est réservée aux utilisateurs admin (ROLE_ADMIN).
- L'utilisateur peut supprimer seulement les tâches qu'il a créé.
- Les tâches liées à l'utilisateur anonyme peut seulement être effacées par des utilisateurs ayant le rôle admin.
- Mise en place de test, utilise commande vendor/bin/simple-phpunit – coverage-html pour créer un rapport de couverture en html.
- Utilisation de Travis.ci avec Github pour la vérification de code de chaque Pull Request.
- Création de Fixtures avec Doctrine Data Fixtures Bundle.
- Chaque getters et setters ont été documentés.
- Création d'un pdf pour expliquer la gestion de l'authentification.
- Création d'un fichier markdown pour expliquer comment contribuer au projet.
- Amélioration du Readme.md
- Création des diagrammes UML

MÉTHODOLOGIE POUR UN CODE DE QUALITÉ



Il est important quand on parle de qualité du code de mettre en place une méthodologie de travail afin d'assurer une meilleure maintenabilité du code. Chaque Pull Request doit être analysé pour vérifier la qualité du code. Il y a deux étapes importantes.

TESTS

Tout d'abord , il est important de tester son code avec des tests fonctionnels et unitaires. Le composant PHPUnit Bridge permet de réaliser de nombreux tests pour toutes les fonctionnalités de l'application. Avec la commande `vendor/bin/simple-phpunit --coverage-html` vous créez un rapport html. Cela vous donne le taux de couverture du code de votre application. Il est important que ce dernier soit supérieur à 70%. Important : Aucune couverture de code sur la première version de ToDoList. Nous y avons remédié.

| | | Lines | |
|--------------|------------------------|---------|-----------|
| Total | <div><div></div></div> | 74.15% | 152 / 205 |
| Controller | <div><div></div></div> | 81.25% | 52 / 64 |
| DataFixtures | | n/a | 0 / 0 |
| Entity | <div><div></div></div> | 100.00% | 47 / 47 |
| Form | <div><div></div></div> | 100.00% | 42 / 42 |
| Repository | <div><div></div></div> | 100.00% | 4 / 4 |

Lorsque le taux de couverture est réalisé. Il faut ensuite utiliser l'outil Travis CI. C'est un outil d'intégration continue qui exécute automatiquement l'intégralité de nos tests unitaires et fonctionnels à chaque Pull Request.

 murat49370 / todo-and-co  build passing

Current Branches Build History Pull Requests

✓ main Merge pull request #21 from murat49370/developement

add technical documentation

Commit cde2e94

Compare b43b5d2..cde2e94

Branch main

Murat CAN

#36 passed

Ran for 48 sec

a day ago

Job log View config

1 Worker information

6

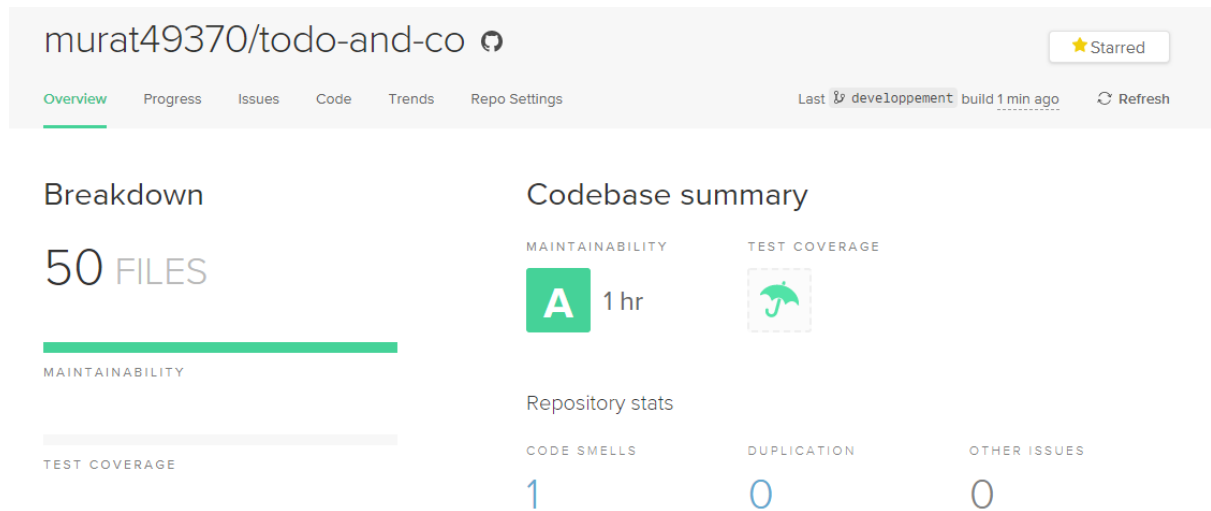
7 Build system information

161

EVALUER LE CODE

Dans un deuxième temps, il est important d'utiliser un outil comme CodeClimate pour vérifier la dette technique du code, les bonnes pratiques, les duplications de code, le code inutilisé entre autres.

Nous pouvons voir qu'avec les différentes modifications apportées avec la feature/refactoring la qualité du code de l'application est passée de C à A.



PERFORMANCE

RAPPORT DE PERFORMANCE

Pour parvenir à établir un rapport de performance, l'outil Blackfire développé par SensioLabs est utilisé. L'objectif est de vérifier le temps de réponse et la consommation de mémoire pour chaque requête. Il est important qu'une réponse soit rapide et qu'il n'y ait pas de consommation de mémoire anormale.

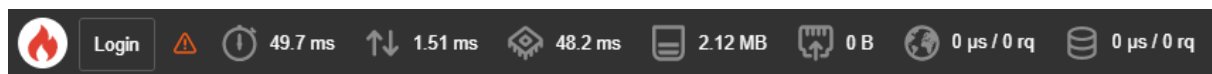
Important: Cet audit de performance a été réalisé sur un serveur web local en condition de production et dans un environnement Windows. Windows est plus lent que Linux pour la lecture de fichiers. Les réponses seront bien plus rapides sur Linux.

Avec Symfony, l'audit de performance doit être réalisé en environnement de production avec le mode debug à 0. En effet, les outils de développement de Symfony tel que le profiler influencent beaucoup sur les résultats.

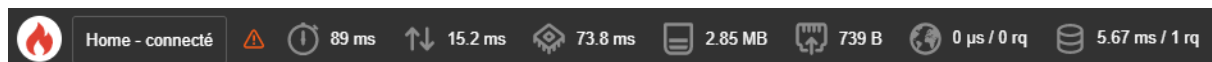
L'outil qui a été utilisé pour faire cet audit de performance est Blackfire.

Audit sans optimisation

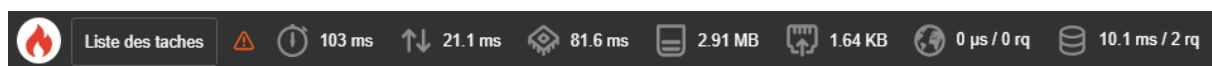
Page de login :



Page d'accueil (connecté) :



Consulter la liste des tâches :



Consulter la liste des utilisateurs :



Optimisation possible

Optimisation de l'autoloader de Composer :

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

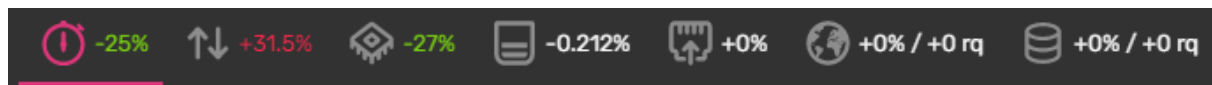
Conseils d'optimisation par Symfony

<https://symfony.com/doc/4.4/performance.html>

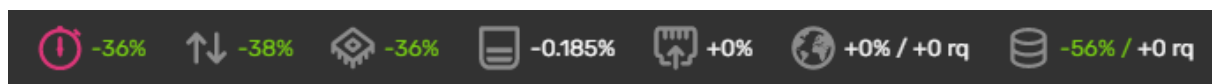
Comparaison après une optimisation

L'optimisation réalisée dans cet exemple est celle de l'autoloader de Composer.

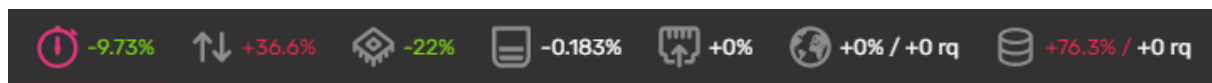
Page de login :



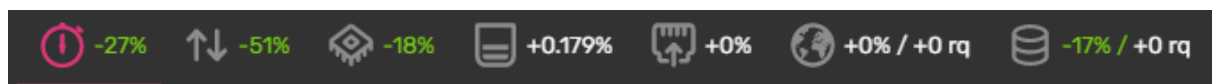
Page d'accueil (connecté) :



Consulter la liste des tâches :



Consulter la liste des utilisateurs :



Nous constatons avec l'optimisation que le temps de réponse est plus que correct pour ce type d'application et qu'il n'y a rien d'anormal à signaler sur la mémoire utilisée. Les résultats pour la version améliorée sont sensiblement équivalents. Nous constatons un temps de réponse assez proche et un traitement en mémoire légèrement supérieur pour la deuxième version.

LES POINTS D'OPTIMISATIONS

Il est important de réfléchir à l'optimisation d'une application web afin d'apporter un gain de performance en production. SensioLabs préconise d'utiliser certains outils afin d'optimiser au mieux son application. <https://symfony.com/doc/current/performance.html>

Nous allons évoquer quelques points que nous avons retenu pour l'application ToDoList.

Configuration de PHP

Il est recommandé d'utiliser PHP 7 ou supérieur qui est plus rapide que les versions antérieures. Il faut configurer le fichier php.ini afin de tirer le meilleur de PHP.

- Désactiver l'utilisation des balises courtes «<? ?>». On doit ouvrir en écrivant explicitement «<?php» `short_open_tag = Off`
- Temps maximum d'exécution après lequel un script sera arrêté `max_execution_time = 30`
- Temps maximum qu'un script peut mettre pour recevoir des données input `max_input_time = 60`
- Taille de mémoire qu'un script peut consommer `memory_limit = 512M`

Il y a de nombreuses autres propriétés qui peuvent être configurées en fonction des besoins de l'application. Il faut retenir que PHP 7 et Symfony 4 utilisés ensemble renvoie le meilleur taux de performance (rapport de comparaison ici réalisé par PHP Benchmarks).

APCu

APCu est un cache de données qui permet de gérer le cache utilisateur. C'est une extension PHP qui reprend le code d'APC qui est moins utilisé au profit d'OPCache mais ce dernier ne gère pas le cache utilisateur d'où l'intérêt d'APCu.

Dans le fichier php.ini, il faut donc l'activer

`extension=php_apcu.dll`

`apc.enabled=1`

`apc.shm_size=32M`

`apc.ttl=7200`

`apc.enable_cli=1`

`apc.serializer=php`

OPcache

Il faut savoir que le script est analysé, décomposé et interprété pour générer un opcode qui sera exécuté. Si vous savez que vous n'allez pas apporter de nouvelles modifications à votre script, il est intéressant de mettre cet opcode en mémoire pour gagner en performance. Pour faire cela, il faut activer l'OPcache (Soyez vigilant sur le fait de mettre en place un système manuel de reset de l'OPcache).

Configuration dans le php.ini :

- Mémoire maximale que OPcache peut utiliser pour stocker des fichiers PHP compilés `opcache.memory_consumption=256`
- Nombre maximal de fichiers pouvant être stockés dans le cache `opcache.max_accelerated_files=20000`
- Désactiver la revalidation du script `opcache.validate_timestamps=0`

PHP realpath cache

Lorsqu'un chemin relatif est transformé en son chemin réel et absolu, PHP met en cache le résultat pour améliorer les performances.

Configuration dans le php.ini :

- Mémoire maximale allouée pour stocker les résultats `realpath_cache_size = 4096K`
- Enregistrer les résultats pendant 10 minutes (600 secondes) `realpath_cache_ttl = 600`

Autoloader

Il faut optimiser l'autoloader de composer. En développement, il permet une vérification globale des fichiers ce qui est pratique. En effet, quand une nouvelle classe est créée, elle peut être directement utilisée. Cependant en production, nous souhaitons simplement reconstruire la configuration à chaque déploiement et ainsi éviter l'apparition de nouvelles classes entre les déploiements. Il faut donc optimiser l'autoloader avec cette ligne de commande par exemple `php composer.phar dump-autoload --optimize`. En fonction des projets, le gain de performance peut atteindre 20 à 30%.

Pour finir, il y a également d'autres pistes intéressantes pour optimiser l'application en fonction de son évolution dans le futur. En voici quelques unes :

Optimisation de Doctrine

Doctrine possède un mécanisme de cache permettant d'accélérer ses performances. Le cache peut être activé à 3 niveaux :

- Metadata cache : Cela permet à Doctrine de ne pas avoir à lire les fichiers de configuration sur le disque afin de pouvoir récupérer les métadonnées d'une entité. Il est indispensable en production.
- Query cache : Ce cache intervient lorsque Doctrine doit transformer une requête DQL en SQL. L'activation de ce cache fait gagner un temps considérable dans le cas où la même requête est exécutée un grand nombre de fois.

- Result cache : Ce cache garde en mémoire le résultat des requêtes envoyées à la base de données afin de ne pas avoir à l'interroger une seconde fois par la suite.

Il peut également être nécessaire de spécifier l'option `fetch="EAGER"` dans la configuration d'une relation Doctrine afin de diminuer le nombre de requêtes. EAGER permet d'utiliser une jointure et de récupérer les deux entités en relation avec une seule requête.

Sinon il y a l'option `fetch="EXTRA_LAZY"` qui permet d'interagir avec une collection mais en évitant au maximum de la charger entièrement en mémoire. Par exemple, l'association est récupérée en Lazy lors du premier accès et ensuite vous pouvez appeler la méthode `count()` sur la collection sans déclencher un chargement complet de la collection.

HTTP_cache

A chaque demande du client, le cache stockera chaque réponse jugée "pouvant être mise en cache". Si la même ressource est à nouveau demandée, le cache envoie la réponse mise en cache au client, en ignorant entièrement votre application. Ce type de cache Http est non négligeable. Symfony est fourni avec un proxy inverse (c'est-à-dire un cache de passerelle) écrit en PHP. C'est un excellent moyen pour commencer.

CONCLUSION

Nous avons donc réalisé un audit de l'application ToDoList. Un bilan de la dette technique a été fait et les améliorations nécessaires ont été effectuées. Nous avons été tout particulièrement attentifs à la stabilité du Framework, la qualité du code, à la documentation nécessaire pour comprendre cette application et à la méthodologie de travail à adopter. Il est également important de suivre les recommandations d'optimisations pour la mise en production de votre application. A cela vous rajouterez un code de qualité et votre application sera performante. Des points d'améliorations sont encore possibles comme par exemple un travail sur l'ergonomie.