

# AUDIT DE QUALITE ET PERFORMANCE

Application Web ToDoList

Réalisé par Gallot Benjamin

Développeur d'Application Web PHP/Symfony

# SOMMAIRE

1. Introduction.....	1
2. Mise à jour version Symfony.....	2
3. Qualité du code.....	3
4. Méthodologie.....	5
5. Performance.....	7
6. Conclusion.....	9

## 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 Codacy, Travis-Ci et Blackfire.

## MISE A JOUR VERSION SYMFONY

Le premier constat est que l'application a été réalisée avec la version 3.1 de Symfony qui n'est plus stable. L'objectif est donc de mettre Symfony à jour vers la version 4.1. C'est la dernière version stable. Elle sera maintenue jusqu'à la fin du mois de juillet 2019.

Tout d'abord vérifier avant une mise à jour de version s'il y a des dépréciations de code. Nous pouvons vérifier avec le profiler de Symfony ou avec PHPUnit Bridge. Rien de majeur n'a été repéré. Nous avons donc commencé la mise à jour vers une version mineur stable 3.4.

Dans un deuxième temps, nous pouvons mettre à jour vers la version majeur 4.1. Mais avant cela vérifions les dépréciations de code. Plusieurs sont à corriger.

Log Messages		
Info. & Errors	1	Deprecations 10
Debug	33	PHP Notices 0
Container	573	

Log messages generated by using features marked as deprecated.

Time	Channel	Message
15:10:09	php	User Deprecated: Symfony\Component\HttpKernel\Kernel::loadClassCache() is deprecated since Symfony 3.3, to be removed in 4.0. <a href="#">Show context</a> <a href="#">Show trace</a>
15:10:09	php	User Deprecated: Symfony\Component\HttpKernel\Kernel::doLoadClassCache() is deprecated since Symfony 3.3, to be removed in 4.0. <a href="#">Show context</a> <a href="#">Show trace</a>
14:59:56	-	Using the unquoted scalar value "levent" is deprecated since Symfony 3.3 and will be considered as a tagged value in 4.0. You must quote it in "C:\IMAMP\htdocs\ToDoList\app/config/config_dev.yml" on line 20. <a href="#">Show context</a> <a href="#">Show trace</a>
14:59:56	-	Using the unquoted scalar value "levent" is deprecated since Symfony 3.3 and will be considered as a tagged value in 4.0. You must quote it in "C:\IMAMP\htdocs\ToDoList\app/config/config_dev.yml" on line 23. <a href="#">Show context</a> <a href="#">Show trace</a>
14:59:56	-	Using the unquoted scalar value "ldoctrine" is deprecated since Symfony 3.3 and will be considered as a tagged value in 4.0. You must quote it in "C:\IMAMP\htdocs\ToDoList\app/config/config_dev.yml" on line 23. <a href="#">Show context</a> <a href="#">Show trace</a>
14:59:56	-	The "framework.trusted_proxies" configuration key has been deprecated in Symfony 3.3. Use the Request::setTrustedProxies() method in your front controller instead. <a href="#">Show context</a> <a href="#">Show trace</a>
14:59:56	-	Not setting "logout_on_user_change" to true on firewall "main" is deprecated as of 3.4, it will always be true in 4.0. <a href="#">Show context</a> <a href="#">Show trace</a>
14:59:56	-(2 times)	Autowiring-types are deprecated since Symfony 3.3 and will be removed in 4.0. Use aliases instead for "Psr\Log\LoggerInterface". <a href="#">Show context</a> <a href="#">Show trace</a>
14:59:56	-	Symfony\Component\HttpKernel\DependencyInjection\Extension::addClassesToCompile() is deprecated since Symfony 3.3, to be removed in 4.0. <a href="#">Show context</a> <a href="#">Show trace</a>

## QUALITE DU CODE

Dans cette partie, vous pouvez observer la dette technique de la première version de l'application ToDoList et les corrections effectuées afin de la diminuer.

	Dette Technique	Correction Apportée
<u>Version</u>	Version 3.1 du Framework Symfony.	Mise à jour vers la version 4.1
<u>Authentification</u>	Les tâches ne sont pas liées aux utilisateurs.	Les tâches créées sont liées à l'utilisateur en cours. Ajout propriété \$user dans entité Task. Les tâches plus anciennes sans auteur peuvent être liés à un utilisateur anonyme avec la commande <code>php bin/console demo:load anonymous</code> .
	Nous ne sommes pas en mesure de choisir un rôle pour l'utilisateur lors de sa création.	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.
	Tous les utilisateurs ont été autorisés à accéder à la zone de gestion des utilisateurs.	La zone de gestion des utilisateurs est réservée aux utilisateurs admin (ROLE_ADMIN).
	Aucunes règles pour supprimer ou modifier des tâches.	Création du service TaskVoter.  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.
<u>Tests</u>	Aucune Couverture	Implementer tests avec PHPUnit Bridge. Couverture à 95% Utilise commande <code>vendor/bin/simple-phpunit --coverage-html cov\</code> pour créer un rapport de couverture en html. Utilisation de Travis.ci avec Github pour la vérification de code de chaque Pull Request.
	Absence de Fixtures	Création de Fixtures avec Doctrine Data Fixtures Bundle.

<u>Refactoring</u>	Code encodepassword dupliqué in UserController.	Creation d'un EventSubscriber avec UserSubscriber.php et declaration du service dans services.yaml.
	Logique métier présent dans les controllers.	Création des FormsHandler pour chaque action des controllers.
	Pages erreurs présentes	Gestion des pages d'erreurs. Création du fichier error403.html.twig pour gérer la page d'erreur en production(par exemple pour les accès refusés). error404.html.twig et error500.html.twig également créés.
	Erreur flash message ToggleisDone or unDone Message inversé.	Corrigé dans TasksController.php
	Propriété \$roles manquante dans l'entité User.	Ajout propriété \$role et ajout nullable = true Creation setRoles().
	@UniqueEntity pour username manquant dans l'entité User et message manquant pour email.	* @UniqueEntity(fields={"username"}, message="Ce nom est déjà utilisé par un utilisateur.") * @UniqueEntity(fields={"email"}, message="Cet email est déjà utilisé par un utilisateur.")
	Dans le controller \$form ->isValid seul est déprécié.	Remplacer par \$form->isSubmitted() && \$form->isValid().
	Dans SecurityController: Request \$request inutile comme argument dans loginAction	Simplification du code et effacement dépendance Request.
	Dans UserController deleteAction() manquant.	Creation de deleteAction() dans UserController and ajout bouton supprimer dans view User list.html.twig. Création de tests fonctionnels pour cette action.
	Le code n'est pas documenté dans les entités.	Chaque getters et setters ont été documentés.
	Certains boutons étaient de trop ou manquant	Suppression bouton "Consulter liste des tâches terminées" car route et action non existante et simplification à " Consulter liste des tâches". Ajout bouton "Consulter la liste des utilisateurs" et suppression bouton "Créer une tâche" quand tâches à zéro car redondant.
<u>Documentation</u>	Aucune documentation sur l'application ToDoList.	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 de Class Diagram – Use Case Diagram – Sequence Diagram - MPD
























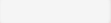
## METHODOLOGIE

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 cov/` vous créez un rapport html disponible à l'URI `/cov`. Cela vous donne le taux de couverture du code de votre application. Il est important que ce dernier soit supérieur à 90%.

**Important : Aucune couverture de code sur la première version de ToDoList. Nous y avons remédiés. Veuillez trouver ci-joint le taux de couverture.**

	Code Coverage							
	Lines		Functions and Methods			Classes and Traits		
Total		94.76% 217 / 229		92.42% 61 / 66		83.33% 15 / 18		
📁 Command		100.00% 33 / 33		100.00% 5 / 5		100.00% 1 / 1		
📁 Controller		100.00% 15 / 15		100.00% 11 / 11		100.00% 4 / 4		
📁 Entity		91.43% 32 / 35		91.67% 22 / 24		50.00% 1 / 2		
📁 EventSubscriber		100.00% 11 / 11		100.00% 5 / 5		100.00% 1 / 1		
📁 Form		100.00% 10 / 10		100.00% 2 / 2		100.00% 2 / 2		
📁 Handler		94.29% 99 / 105		92.86% 13 / 14		85.71% 6 / 7		
📁 Security		85.00% 17 / 20		60.00% 3 / 5		0.00% 0 / 1		
📄 AppBundle.php		n/a 0 / 0		n/a 0 / 0		n/a 0 / 0		

Lorsque que 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.

 **Travis CI**  
Succeeded — 6 hours ago

✓ Travis CI - Branch

### Travis CI - Branch

✓ Success

🕒 built 6 hours ago in 2 minutes

👁 34f6050 by @Benj972

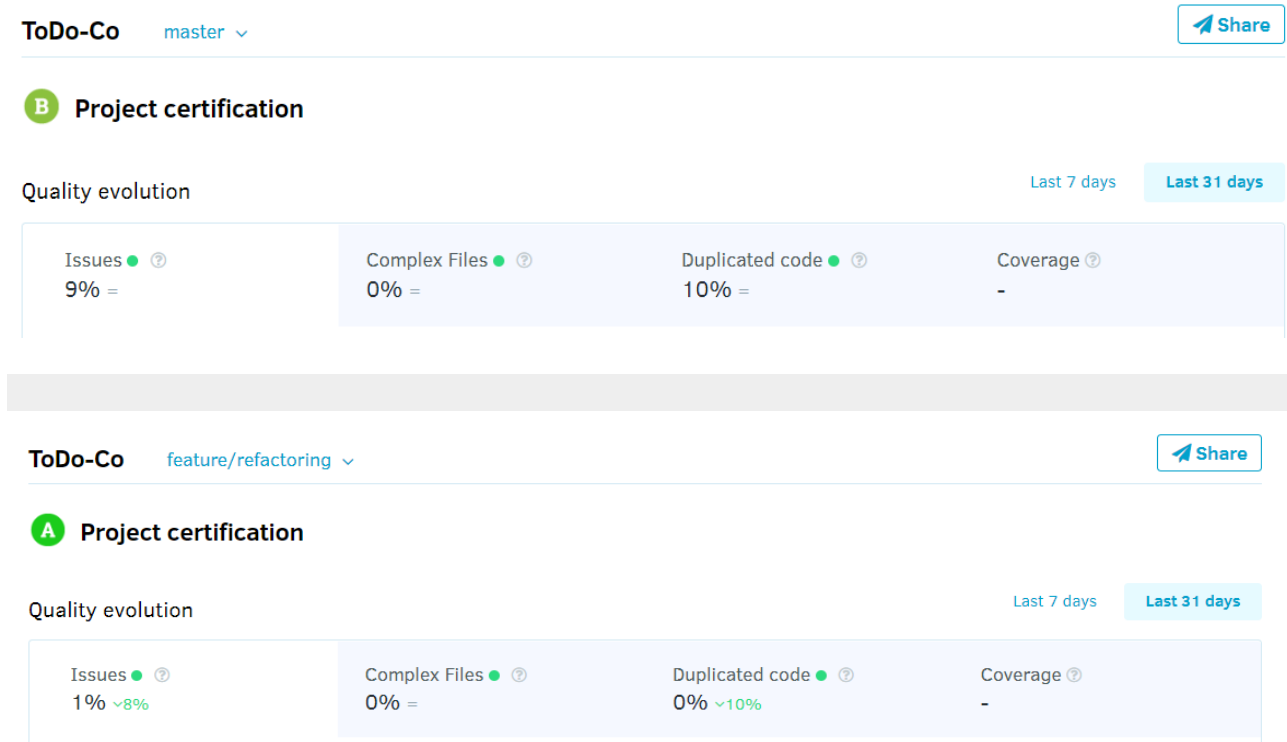
🔗 feature/refactoring

### Build Passed

✓ The build passed.

## EVALUER LE CODE


Dans un deuxième temps, il est important d'utiliser un outil comme Codacy pour vérifier la dette technique du code, les bonnes pratiques, les duplications de code, le code inutilisé entre autres. Cet outil doit également être intégré à chaque Pull Request afin qu'il y ait une vérification automatisée. Nous avons fait une analyse de la première version et à chaque création de nouvelles branches avec des améliorations. Voici un exemple :









*Nous pouvons voir qu'avec les différentes modifications apportées avec la feature/refactoring la qualité du code de l'application est passée de B à A. Le taux de problèmes à chuter de 9% et la duplication de code est passée à nul.*

## RESULTAT PULL REQUEST VALIDE

Add more commits by pushing to the **feature/refactoring** branch on Benj972/ToDo-Co.



-  **All checks have passed** [Hide all checks](#)  
3 successful checks
  -  **Codacy/PR Quality Review** — Up to standards. A positive pull request. [Details](#)
  -  **Travis CI - Branch** Successful in 1m — Build Passed [Details](#)
  -  **continuous-integration/travis-ci/push** — The Travis CI build passed [Details](#)
-  **This branch has no conflicts with the base branch**  
Merging can be performed automatically.

[Merge pull request](#)  You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

# PERFORMANCE

## RAPPORT DE PERFORMANCE

Pour parvenir à établir un rapport de performance, l'outil Blackfire développé par SensioLabs a été 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 inférieure à trois secondes et qu'il n'y est 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 rapide sur Linux.

Version MVP(Minimum Viable Product) :

URL	TEMPS	MEMOIRE
/login	132ms	6.08MB
/	230ms	9.53MB
/tasks	218ms	9.69MB
/tasks/create	290ms	12.2MB
/tasks/{id}/edit	280ms	12.2MB
/users	249ms	9.56MB
/users/create	275ms	12MB
/users/{id}/edit	277ms	12MB

Version améliorée :

URL	TEMPS	MEMOIRE
/login	151ms	6.3MB
/	208ms	9.73MB
/tasks	218ms	9.91MB
/tasks/create	276ms	12.5MB
/tasks/{id}/edit	282ms	12.5MB
/users	212ms	9.96MB
/users/create	286ms	12.6MB
/users/{id}/edit	277ms	12.7MB

Nous avons comparé différentes requêtes GET entre la version MVP et celle avec la dette technique améliorée.

Nous constatons pour la version MVP 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. Cela s'explique par l'ajout des services et des FormHandler entre autres.

En conclusion, toutes les modifications qui ont été apportées pour réduire la dette technique n'ont pas d'impact négatif sur les performances de l'application ToDoList.



## 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.

### Autoloader

Tout d'abord, 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%.

### 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*).

### 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.

Cette partie a pour but de vous proposer différentes solutions afin d'optimiser au mieux votre application web pour la mettre en production. Cette étape nécessite un travail spécifique qui peut être réalisé à la suite de cet audit de performance et qualité.

## CONCLUSION

Nous avons donc réalisé un audit sur ToDoList. Un bilan de la dette technique a été fait et les améliorations nécessaires ont été effectuées. Nous avons été tout particulièrement attentif 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. Cela fût une première approche, des points peuvent encore être amélioré comme l'implémentation du cache pour optimiser au mieux les performances ou encore un travail sur l'ergonomie du site avec le Framework d'interface Bootstrap.