

ToDo & Co – TodoList

Audit Qualité du Code

1. Résumé du projet

1.1. Description du besoin

1.1.1. Contexte

1.1.2. Correction anomalies

1.1.3. Implémentation de nouvelles fonctionnalités

2. Audit de code

2.1. Tests sur l'application de base

2.1.1. check :security

2.1.2. Analyse du code

2.1.3. Tests unitaires et fonctionnels

2.1.4. Tests manuels

2.2. Tests sur l'application améliorée

2.2.1. Check :security

2.2.2. Analyse du code

2.2.3. Tests unitaires et fonctionnels

3. Tests de performances

3.1. Tests sur les 2 versions

4. Conclusion

1. Résumé du projet

1.1. Description du besoin

1.1.1. Contexte

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).

ToDo & Co a enfin réussi à lever des fonds pour permettre le développement de l'entreprise et surtout de l'application faite avec le framework PHP Symfony. Après avoir fait un audit sur la qualité du codé, il faut s'occuper des points suivants :

- l'implémentation de nouvelles fonctionnalités ;
- la correction de quelques anomalies ;
- et l'implémentation de tests automatisés.

1.1.2. L'implémentation de nouvelles fonctionnalités ;

Autorisation :

- Seuls les utilisateurs ayant le rôle administrateur (*ROLE_ADMIN*) doivent pouvoir accéder aux pages de gestion des utilisateurs.
- Les tâches ne peuvent être supprimées que par les utilisateurs ayant créé les tâches en question.
- Les tâches rattachées à l'utilisateur "anonyme" peuvent être supprimées uniquement par les utilisateurs ayant le rôle administrateur (*ROLE_ADMIN*).

1.1.3. 1.1.3 La correction de quelques anomalies :

- **Une tâche doit être attachée à un utilisateur** : Actuellement, lorsqu'une tâche est créée, elle n'est pas rattachée à un utilisateur.
- **Choisir un rôle pour un utilisateur** : Lors de la création d'un utilisateur, il doit être possible de choisir un rôle pour celui-ci (rôle utilisateur /*ROLE_USER* ou rôle administrateur/*ROLE_ADMIN*).

1.1.4. L'implémentation de tests automatisés.

- Il vous est demandé d'implémenter les tests automatisés (tests unitaires et fonctionnels) nécessaires pour assurer que le fonctionnement de l'application est bien en adéquation avec les demandes.
- Ces tests doivent être implémentés avec **PHPUnit** ; vous pouvez aussi utiliser Behat pour la partie fonctionnelle.
- Vous prévoyez des données de tests afin de pouvoir prouver le fonctionnement dans les cas explicités dans ce document.
- Il vous est demandé de fournir un rapport de couverture de code au terme du projet. Il faut que le taux de couverture soit supérieur à 70 %.

2. Audit de code

Pour garantir la qualité du code, nous avons utilisé plusieurs outils :

- ☞ Phpstan/php-cs-fixer pour le respect des normes et standards du code PHP;
- ☞ Codaxy qui est un outil bien connu de validation de qualité du code ;
- ☞ Travis CI pour l'intégration continue afin d'éviter une régression de code lorsqu'on pousse notre code modifié sur la branche principale du dépôt de déploiement ;
- ☞ Blackfire pour tester les performances de notre application.

2.1. Tests sur l'application de base

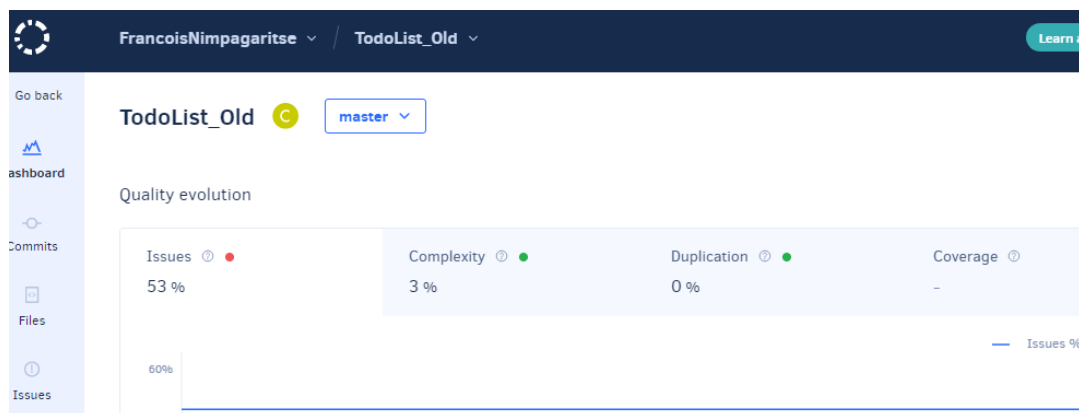
2.1.1. Security

Ici il s'agit de procéder à la vérification des vulnérabilités des packages installés. La version de symfony utilisée étant obsolète et ne plus maintenue augmente le risque de vulnérabilité. Il en est de même pour celle de PHP qui est 5.5.9

Une migration vers une version à jour et maintenue est recommandée. Pour cela nous recommandons la version LTS 4.4 ou la version 5.

2.1.2. Analyse du code

Aucune analyse du code n'avait été effectuée avant. Nous avons procédé à une analyse du code pour ce projet de base et voici le résultat obtenu avec **codacy** et qui est accessible [ici](#):



2.1.3. Test unitaires et fonctionnels

The screenshot shows a terminal window with a file explorer on the left displaying the project structure. The terminal output shows the command to run PHPUnit tests, which resulted in 'No tests executed!'.

```
PS C:\_TodoList_Old\TodoList_Old> vendor/bin/phpunit
PHPUnit 8.5.18 by Sebastian Bergmann and contributors.

No tests executed!
PS C:\_TodoList_Old\TodoList_Old>
```

On remarque qu'il n'y avait aucune couverture de code pour l'application de base.

2.1.4. Tests manuels

En testant manuellement les pages de notre application on relève les défauts suivants à améliorer :

- Le bouton « Consulter les tâches terminées » non fonctionnel ;
- Le lien « To Do List app » ne dirige nulle part ;
- Le bouton « Créer un utilisateur » ne devrait pas s'afficher sur la page de connexion ;
- Des pages d'erreurs (403/404/500) non personnalisées.

2.2. Test sur l'application améliorée

2.2.1. Security

Ici il s'agit de procéder à la vérification des vulnérabilités des packages installés pour Symfony avec la commande :

```
php bin/console security : check
```

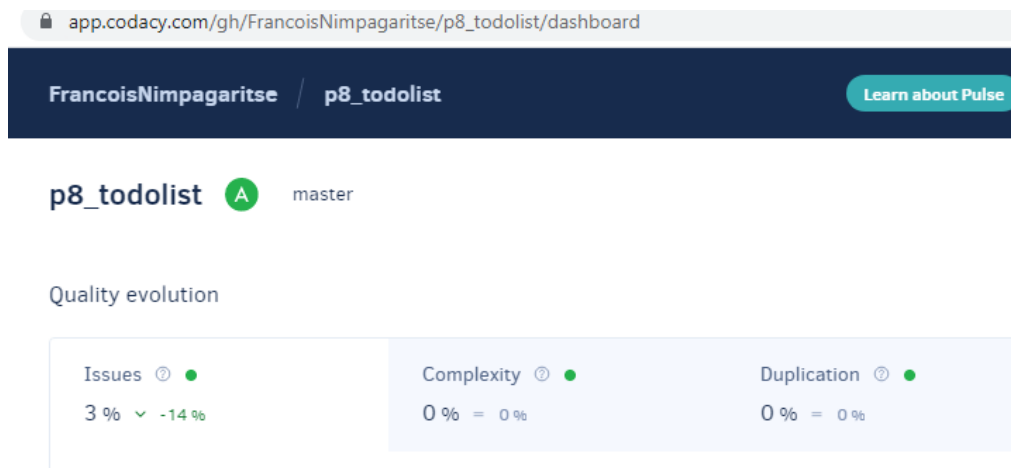
```
Symfony Security Check Report
=====

No packages have known vulnerabilities.

Note that this checker can only detect vulnerabilities that are referenced in the security advisories database.
Execute this command regularly to check the newly discovered vulnerabilities.
PS C:\_P8TodoList52\todolist>
```















2.2.2. Analyse du code

Analyse du code obtenu avec codacy et qui est accessible [ici](#):



2.2.3. Test unitaires et fonctionnels

Nous avons mis en place les tests unitaires et fonctionnels pour notre application améliorée comme le montre cette image :

	Code Coverage					
	Lines			Functions and Methods		
Total		85.90%	134 / 156		86.00%	43 / 50
Controller		98.39%	61 / 62		90.91%	10 / 11
Entity		92.31%	48 / 52		90.32%	28 / 31
Form		100.00%	10 / 10		100.00%	2 / 2
Repository		100.00%	4 / 4		100.00%	2 / 2
Security		84.62%	11 / 13		50.00%	1 / 2
Kernel.php		0.00%	0 / 15		0.00%	0 / 2

Legend

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

















































Generated by [php-code-coverage 7.0.14](#) using [PHP 7.2.8](#) with [Xdebug 3.0.4](#) and [PHPUnit 8.5.17](#) at Tue Jul 20 20:53:37 CEST 2021.

Ce rapport est accessible à ce [lien](#)

3. Les tests de performance

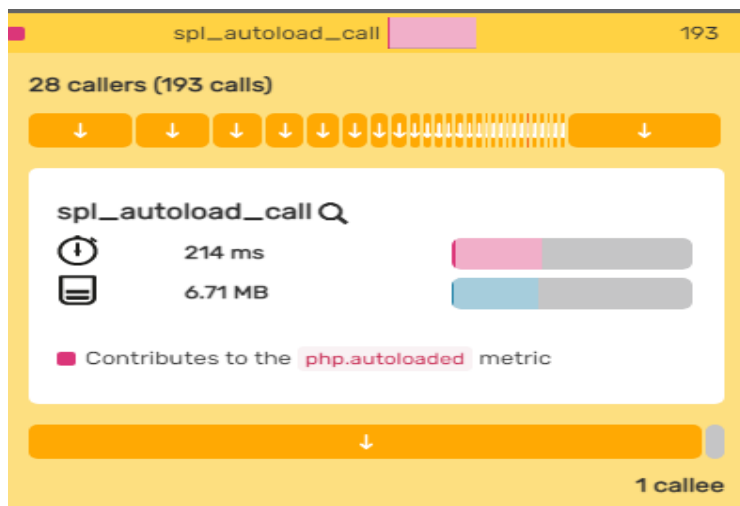
Toutes nos analyses de performance ont été réalisées en utilisant l'outil **Blackfire**.

Les tests portent sur les routes suivantes de l'application sur la version de base et puis sur la version améliorée et montrent une nette amélioration de performance en faisant une migration vers les versions à jour.

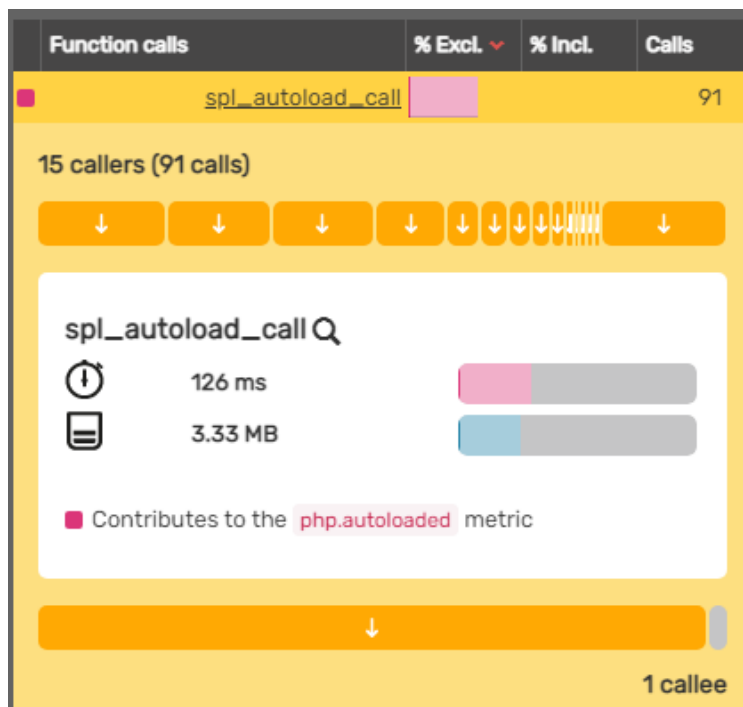
Adresse testée	V1	V2	Comparaison
	Temps d'exécution & mémoire consommée	Temps d'exécution & mémoire consommée	
/login	To Do List app  693 ms  14.3 MB	To Do List app  388 ms  12.4 MB	Comparison  -44%  -13%
/	To Do List app  788 ms  16.9 MB	To Do List app  445 ms  14.6 MB	Comparison  -43%  -13%
/users	To Do List app  834 ms  16.9 MB	To Do List app  447 ms  14.7 MB	Comparison  -46%  -13%
/users/create	To Do List app  863 ms  18.3 MB	To Do List app  585 ms  19.2 MB	Comparison  -32%  +4.79%
/users/{id}/edit	To Do List app  1.01 s  20.5 MB	To Do List app  612 ms  19.1 MB	Comparison  -40%  -7.03%
/tasks	To Do List app  978 ms  17 MB	To Do List app  458 ms  14.8 MB	Comparison  -53%  -13%
/tasks/create	To Do List app  1 000 ms  20.7 MB	To Do List app  571 ms  18.8 MB	Comparison  -43%  -9.53%
/tasks/{id}/edit	To Do List app  1.01 s  20.8 MB	To Do List app  578 ms  19.1 MB	Comparison  -39%  -8.2%

On a un gain de temps d'exécution de 42% en moyenne et une économie de 9% en moyenne de mémoire consommée.

En observant le graphe d'appels, nous remarquons qu'il est indispensable d'optimiser le chargement de classes.



On pourra le faire avec la commande : « `composer dump-autoload -o` » . Comme on le voit le temps d'exécution et la mémoire consommée sont sensiblement améliorés.



4. Conclusion

Pour l'application améliorée, des tests couvrant à plus de 85% l'application ont été mis en place ainsi qu'un système d'intégration continue avec **Travis CI**. Ce qui permet de s'assurer qu'en continuant à modifier le code de notre projet on a les bons éléments pour nous garantir que notre application ne subit pas de régression.

Les tests de performance quant à eux nous indiquent les endroits où le code a besoin d'être amélioré et ainsi obtenir une application optimisée.

Les différentes recommandations sont indiquées dans un fichier dédié.