Audit Performance

Todo & Co - Todolist

Thomas Chauveau
OpenClassrooms - Développeur d’application - PHP / Symfony
Projet 8 - Améliorez une application existante de Todo & Co

30 octobre 2023

Sommaire

1.Introduction

1.1. Présentation

ToDo & Co est une startup qui a développé un MVP (Todolist) pour la gestion de tâches via une application Symfony. ToDo & Co veut faire évoluer la codebase de Todolist vers des technologies plus récentes et faciles à maintenir. ToDo & Co a donc fait appel à moi pour améliorer la qualité du code, ajouter de nouvelles fonctionnalités et corriger des anomalies. Je dois aussi implémenter des tests automatisés pour assurer la fiabilité de l'application et réaliser un audit de qualité du code et de performance.

1.2. Objectifs de l’audit

L'objectif principal de cet audit est d'évaluer et d'améliorer la qualité et les performances de l'application Symfony en question. Nous chercherons à identifier les goulets d'étranglement potentiels, les opportunités d'optimisation et les meilleures pratiques qui n'ont pas été suivies. L'audit fournira également des recommandations spécifiques pour améliorer la qualité du code et les performances de l'application.

1.3. Portée de l’audit

L'audit se concentrera sur les domaines suivants :

- **Cycles de Version de PHP et Symfony** : Une revue des versions actuellement utilisées pour PHP et Symfony, avec des recommandations sur les mises à jour nécessaires pour rester dans un cycle de support actif.
- **Performances de l'Application** : Utilisation du Profiler de Symfony pour évaluer les performances de l'application, notamment le temps de réponse, l'utilisation de la mémoire et l'efficacité des requêtes SQL.
- **Qualité du Code** : Évaluation de la qualité du code à l'aide d'outils d'analyse statique et de métriques de qualité du code. Les outils utilisés incluent PHPStan, PHP-CS-Fixer, PHPMetrics, et d'autres.

1.4. Rappel du cycle de versions

Que ce soit PHP ou Symfony, les deux utilisent un versionning dit ***Semantic Versionning***. C'est-à-dire que leurs numéros de versions sont notées ainsi : `MAJEUR.MINEUR.CORRECTIF`. (ex : PHP 8.2.11 ; Symfony 6.3.4)

1.4.1. PHP

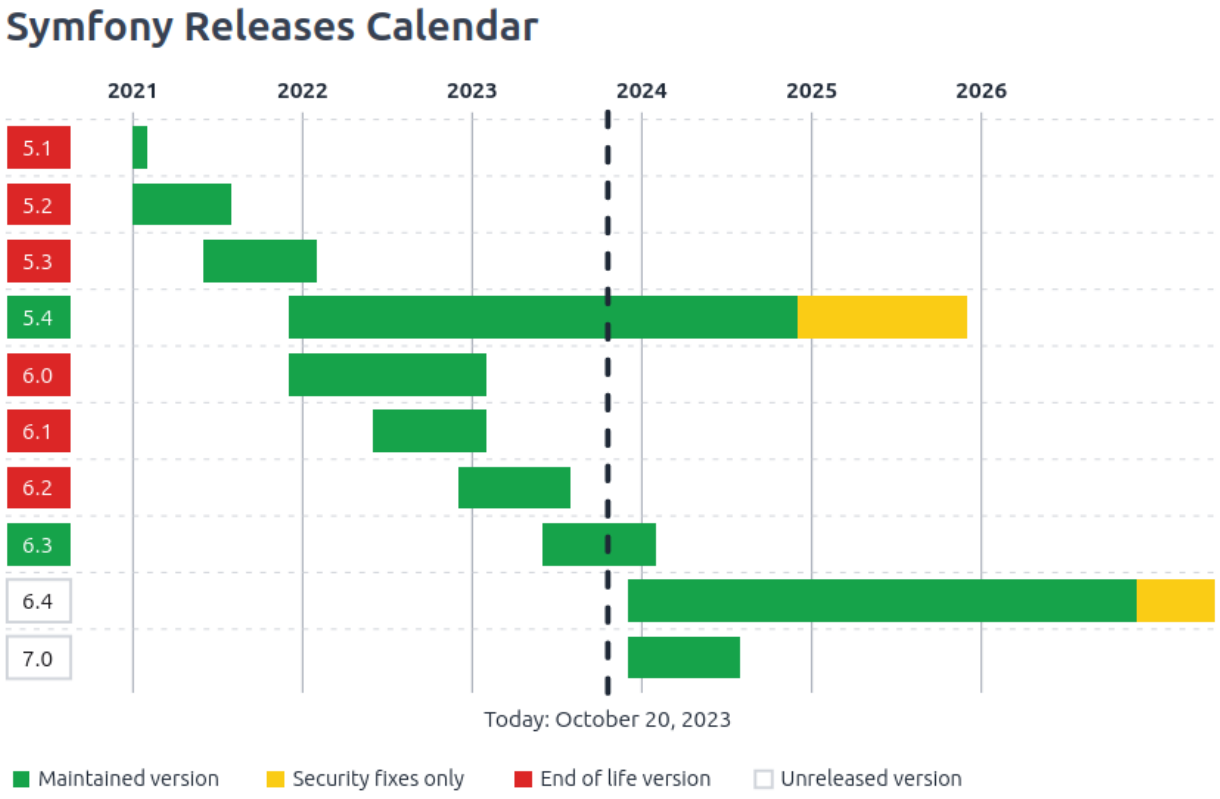
- **Support Actif** : Chaque version mineure de PHP reçoit un support actif pendant deux ans à partir de sa date de sortie initiale. Pendant cette période, des correctifs de bugs et des améliorations de sécurité sont régulièrement publiés.
- **Support de Sécurité** : Après la période de support actif, chaque version bénéficie d'une année supplémentaire de support de sécurité, pendant laquelle seuls les correctifs de sécurité sont fournis.
- **Fin de Vie (EOL)** : Après trois ans, la version atteint la fin de sa vie et ne reçoit plus de mises à jour, même pour des problèmes de sécurité.

1.4.2. Symfony

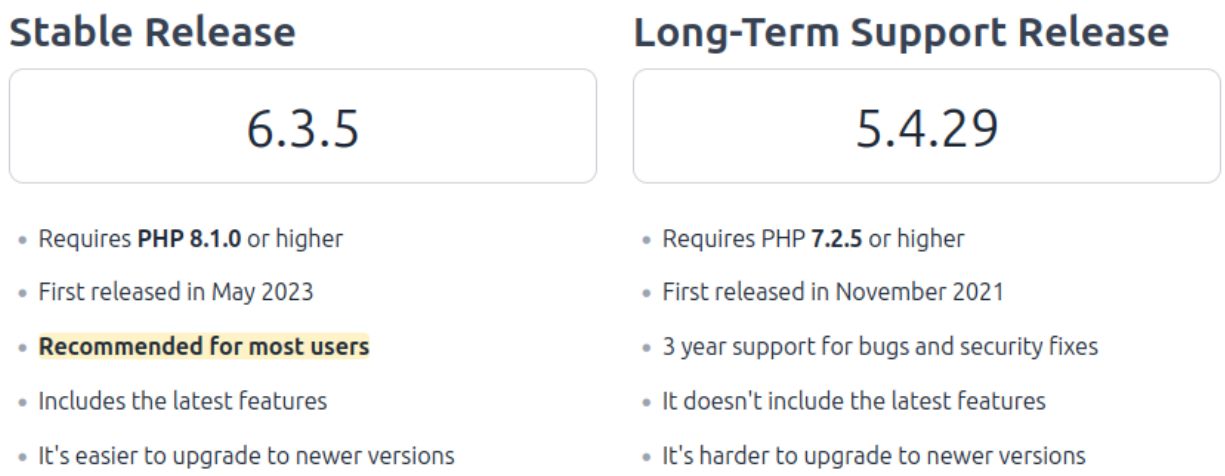
Symfony, bien que respectant le Semantic Versionning (SemVer), ajoute une notion de LTS (Long Term Support). Les versions LTS ont un support étendu.

- **Support Actif** : Chaque version mineure de Symfony reçoit un support actif pendant un an à partir de sa date de sortie initiale. Des correctifs de bugs sont régulièrement publiés pendant cette période.
- **Support de Sécurité** : Après la période de support actif, chaque version bénéficie de deux années supplémentaires de support de sécurité.
- **Versions LTS** : Les versions LTS de Symfony reçoivent un support total de trois ans et un support de sécurité de quatre ans.

1.4.3. Roadmap



Au moment de finaliser ce rapport, la version 6.3.5 de Symfony est celle qui est conseillée et est également étiquetée comme stable. Néanmoins, son cycle de support est restreint. Bien que la version LTS actuelle, la 5.4.29, aurait pu être une option, nous avons choisi de mettre à jour vers la dernière version stable pour minimiser la dette technique du MVP. Cette image illustre que cette version est actuellement la recommandée :



Plusieurs facteurs ont motivé cette décision. Par exemple, la version minimale de PHP requise est la 8.1.0. Étant donné que PHP 8.0 a apporté un ensemble significatif de nouvelles fonctionnalités et une amélioration notable des performances par rapport à PHP 7.4, il était logique d'opter pour la combinaison PHP 8.1+ et Symfony 6.3.x.
La mise à jour de Todolist est donc basée sur Symfony 6.3.5, et les résultats de l'audit final utiliserons PHP 8.2.11.
Les résultats de l'application de base sont, eux, basés sur Symfony 3.4.49 et PHP 7.2.34.

1.4.4. Notes et références

Pour en savoir plus sur le sujet, vous pouvez consulter ces pages :

- PHP 8 release annoucement [fr]
- PHP 8.1 release announcement [en]
- PHP 8.2 release announcement [en]
- Symfony releases [en]
- Symfony benchmark (5.4.2 - php 7.2 à php 8.1) [en]

1.5. Outils utilisés

Pour réaliser ces audits, plusieurs outils et méthodologies seront utilisés :

- **Profiler de Symfony** : Pour le benchmarking et l'analyse des performances.
- **PHPStan** : Pour l'analyse statique du code.
- **PHP-CS-Fixer** : Pour le formatage du code et la conformité aux standards.
- **PHPMetrics** : Pour obtenir des métriques détaillées sur la qualité du code.
- **PHPUnit** : Pour les tests unitaires et fonctionnels.

Chaque section de l'audit utilisera une combinaison de ces outils pour fournir une évaluation complète.

2. Audit des performances

Les prochains tableaux seront résalisés sous ces conditions :
Avant : Symfony 3.4 / PHP 7.4.2
Après : Symfony 6.3 / PHP 8.2.11

Les tests suivants ont tous été réalisés 3 fois. La valeur moyenne des 3 tests est la valeur gardée.

2.1. Temps de réponse par Requête

Le temps de réponse d'une application web est un facteur critique qui influence directement l'expérience utilisateur. Un temps de réponse plus court signifie une application plus réactive et donc une meilleure expérience utilisateur. Le tableau ci-dessous présente une comparaison du temps de réponse avant et après les optimisations pour différentes URL de l'application.

URL	Temps de réponse Avant (ms)	Temps de réponse Après (ms)	Amélioration (%)
/	57	25	-56.14
/login	147	53	-63.95
/tasks	107	60	-43.93
/tasks/create	114	25	-78.07
/tasks/{id}/edit	82	46	-43.9
/users	64	58	-9.38
/users/create	204	98	-51.96
/users/{id}/edit	102	34	-66.67

Nous observons des améliorations significatives sur toutes les routes. Par exemple, la route `/tasks/create` a vu une réduction de 78.07% du temps de réponse, passant de 114 ms à 25 ms.

2.2. Utilisation de la mémoire par requête

L'optimisation de l'utilisation de la mémoire est un autre aspect crucial pour améliorer les performances d'une application. Le tableau ci-dessous montre les changements dans l'utilisation de la mémoire pour différentes URL.

URL	Mémoire utiliée avant (MiB)	Mémoire utilisée après (MiB)	Amélioration (%)
/	2	2	0%
/login	6	2	-66.67%
/tasks	2	4	+100%
/tasks/create	4	4	0%
/tasks/{id}/edit	4	4	0%
/users	2	6	+200%
/users/create	4	4	0%
/users/{id}/edit	2	4	+100%

La plupart des routes ont vu une amélioration ou une stabilisation de l'utilisation de la mémoire. Par exemple, la route `/login` a vu une réduction de 66.97% de l'utilisation de la mémoire. Cependant, certaines routes comme `/tasks` et `/users` ont montré une augmentation de 100% et 200% respectivement, mais celà n'influe pas sur cette application, puisque le but premier reste d'être un MVP.

2.3. Requêtes SQL

Les requêtes SQL peuvent souvent être le goulot d'étranglement dans les performances d'une application web. Le tableau ci-dessous compare le temps d'exécution des requêtes SQL avant et après les optimisations.

Requête SQL	Temps exécution avant (ms)	Temps exécution après (ms)	Amélioration (%)
/	0.58	0.14	-75.86
/login	N/A	N/A	N/A
/tasks	0.61	0.36	-40.98
/tasks/create	0.51	0.24	-53.94
/tasks/{id}/edit	0.83	0.21	-74.7
/users	0.94	0.35	-62.77
/users/create	0.49	0.22	-55.1
/users/{id}/edit	1.82	0.20	-89.01

On observe des améliorations drastiques dans le temps d'exécution pour toutes les routes. Par exemple, la route `/users/{id}/edit` a vu une réduction de 89.01% du temps d'exécution. Cela indique que les optimisations apportées ont eu un impact positif sur la performance des requêtes SQL.

2.4. Analyse détaillée

Suite à la migration Symfony 3.4 / PHP 7.4.2 vers Symfony 6.3 / PHP 8.2.11, nous pouvons voir des améliorations notables, que ce soit au niveau des performances du temps de réponse, que des performances des requêtes SQL. La mesure concernant l'utilisation de mémoire ne permet pas d'affirmer que la migration a été bénéfique.

3. Qualité du code

3.1. Méthodologie

Dans cette partie, nous allons voir ce qui a été mis en place pour améliorer la qualité du code.

Nous allons donc voir les résultats des analyses réalisées avec PHPStan, PHP-CS-Fixer, PHPInsights, PHPMetrics et Rector. La dernière partie sera un résumé des tests réalisés avec PHPUnit.
Les tests (PHPUnit), ayant été implémentés après la migration, ne seront pas comparés entre l'application de base et sa mise à jour.

3.2. PHPStan

Nous pouvons observer que la majorité des erreurs relevées dans le projet initial sont dues au fait que le projet ne respectait pas totalement les normes PHP en utilisant les Type Declarations, introduits dès PHP 7.0 et améliorés à chaque version mineure de PHP.

Références :
Résultats PHPStan de base
Résultats PHPStan à jour

3.3. PHP-CS-Fixer

Le projet de base ne permet pas de soulever de potentielles erreurs (PHP-CS-Fixer v2).
Le projet à jour n'indique plus de problèmes à corriger.

3.4. PHPMetrics

Nous pouvons observer que globalement, le projet a beaucoup évolué et grandi, et que malgré certains points de comparaisons moins bons et plus de 540 lignes de code ajoutées, la complexité globale du projet a été améliorée, passant de 68.03 à 47.16.

Références :
Rapport du projet de base
Rapport du projet à jour

3.5. Rector

Au début de la migration, Rector a soulevé plusieurs problèmes, concernant 12 fichiers au total.
Après la migration, Rector ne soulève plus aucun problèmes.

Références:
Rapport du projet de base

3.6. PHPInsights

Le projet de base est trop ancien pour installer PHPInsights (requiert Symfony >= 4.2)
Voici les résultats du run sur la version mise à jour :
Code : 99%
Complexité : 91.3%
Architecture: 100%
Style: 98.8%

Références:
Rapport phpinsights

3.7. PHPUnit

Le projet de base ne contient pas de tests via PHPUnit ou autre suite de tests.
Le projet mis à jour contient 38 tests et 254 assertions.

Références :
résumé du run PHPUnit (coverage)
configuration PHPUnit

3.8. Résultats globaux

En raison de l'ancienneté de la version initiale de l'application, une comparaison directe entre les deux versions n'est pas entièrement faisable. Néanmoins, l'introduction de divers outils de qualité de code dans la version mise à jour offre une amélioration bienvenue de la qualité du projet.

Le code est maintenant en conformité avec les dernières normes et meilleures pratiques, tant du côté de PHP que de Symfony. De plus, la présence d'une grande suite de tests automatisés rend le projet entièrement testable, ajoutant une couche supplémentaire de fiabilité et de robustesse.

4. Conclusion

Ce rapport sert de témoignage à l'efficacité des mesures prises pour améliorer la qualité du code et la performance de l'application. Les efforts déployés ont non seulement rendu le projet conforme aux standards actuels, mais ont également préparé le terrain pour une maintenance et une évolution plus aisées à l'avenir. Le projet est désormais bien positionné pour répondre aux défis futurs, tout en maintenant un haut niveau de qualité et de performance.