

# TO DO LIST

1  
2  
3



## ToDo & Co

Documentation technique

Audit de qualité et de performance

Frédéric Vanmarcke



VmkDev

10 avril 2022

# Sommaire

<b>Introduction .....</b>	<b>3</b>
Contexte du projet .....	3
Objectifs du projet .....	3
Approche méthodologique .....	3
<b>Analyse préliminaire du projet .....</b>	<b>Erreur ! Signet non défini.</b>
Analyse technique .....	4
Environnement technique.....	4
Qualité du code.....	6
Analyse fonctionnelle.....	8
Fonctionnalités .....	8
Qualité .....	9
Synthèse : Inventaire de la dette technique .....	10
Conclusion de la dette technique.....	12
<b>Rapport d’Audit de qualité et de performances .....</b>	<b>13</b>
Qualité du code.....	13
Architecture .....	13
Documentation .....	13
SymfonyInsight .....	13
Tests : couverture du code de l’application.....	14
Performance de l’application.....	15
<b>Bilan et plan d’amélioration .....</b>	<b>18</b>

# Introduction

## Contexte du projet

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).

## Objectifs du projet

ToDo & Co a enfin réussi à lever des fonds pour permettre le développement de l'entreprise et surtout de l'application.

Dans ce cadre, ToDo & Co a confié l'analyse et l'amélioration de l'application à VmkDev, entreprise spécialisée dans le développement web.

Les objectifs définis sont les suivants :

- Identifier et de corriger les anomalies
- Implémenter de nouvelles fonctionnalités
- Implémenter des tests automatisés
- Analyser le projet grâce à des outils permettant d'avoir une vision d'ensemble de la qualité du code et des différents axes de performance de l'application
- Etablir un compte rendu de cet audit de qualité et de performance et proposer un plan d'amélioration

## Approche méthodologique

L'évaluation de la qualité et des performances techniques de l'application web sera réalisée en plusieurs étapes.

1. Un état des lieux de la **dette technique** de l'application sera réalisé par une analyse manuelle (fonctionnelle et technique) et grâce à des outils d'automatisation d'analyse de code et de performance.
2. Sur les bases de cette première analyse, des correctifs et améliorations seront apportés à l'application afin de réduire la dette technique et d'implémenter les nouvelles fonctionnalités demandées.
3. Finalement, un audit de qualité et de performance sera de nouveau réalisé afin d'évaluer la progression de l'application en termes de qualité et de performance après mise en place des différentes actions correctives.

La correction des anomalies et l'implémentation des nouvelles fonctionnalités seront réalisées en suivant la méthodologie de développement piloté par les tests unitaires et fonctionnels.

## Analyse préliminaire du projet

L'analyse préliminaire a pour objectif d'évaluer la dette technique de l'application avant la mise en place des actions correctives et l'implémentation des nouvelles fonctionnalités.

## Analyse technique

Cette analyse technique a été réalisée dès la récupération du projet initial sur [GitHub](#).

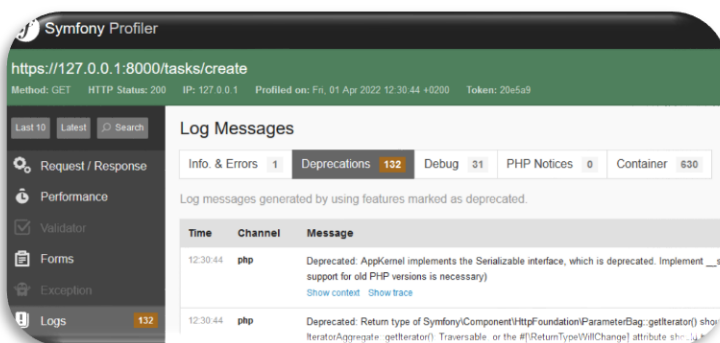
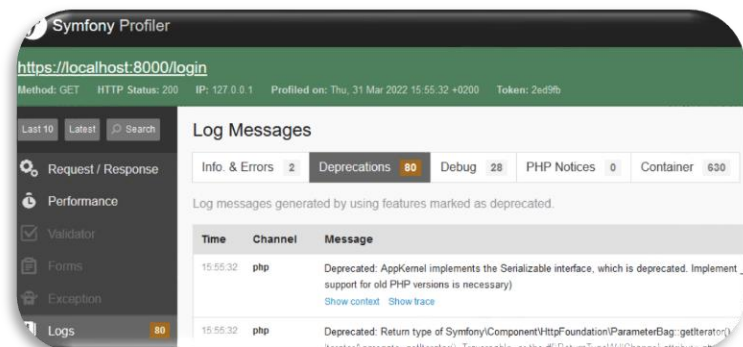
### Environnement technique

L'utilisation d'une **version stable et maintenue** du Framework est essentielle pour garantir la pérennité du développement de l'application. La dernière version actuellement maintenue étant la [version 6.0](#), les premières actions correctives seront de faire évoluer le projet initial vers cette version du Framework Symfony.



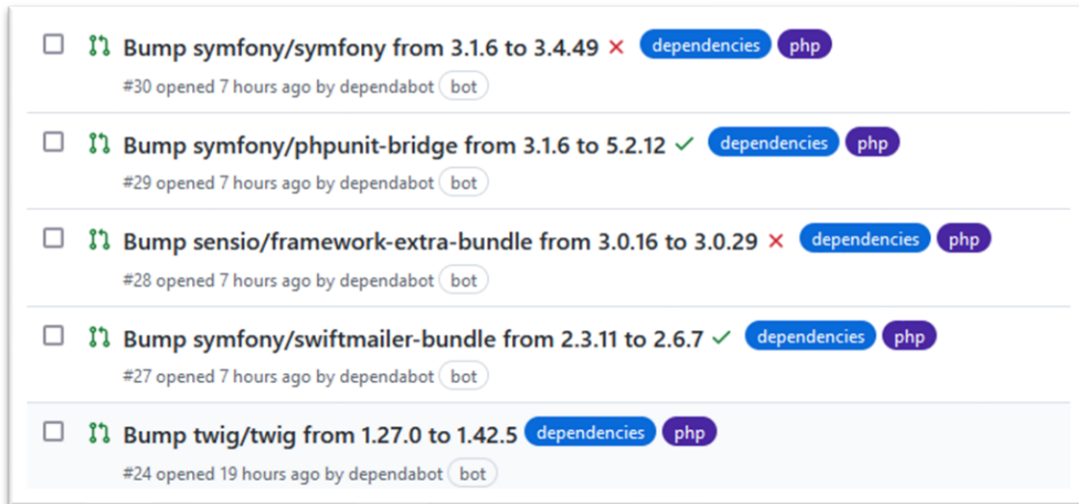
L'installation du projet initial en son état de MVP a permis de mettre en évidence un **environnement technique obsolète**, avec d'une part une version non maintenue depuis juillet 2017 du Framework Symfony ([version 3.1.10](#)) avec un PHP [version 5.5.9](#) donc la mise à jour remonte au 6 février 2014,

d'autre part l'analyse du Profiler Symfony a générés 80 Messages de fonctionnalités marquées comme obsolètes sur la route <https://localhost:8000/login>,

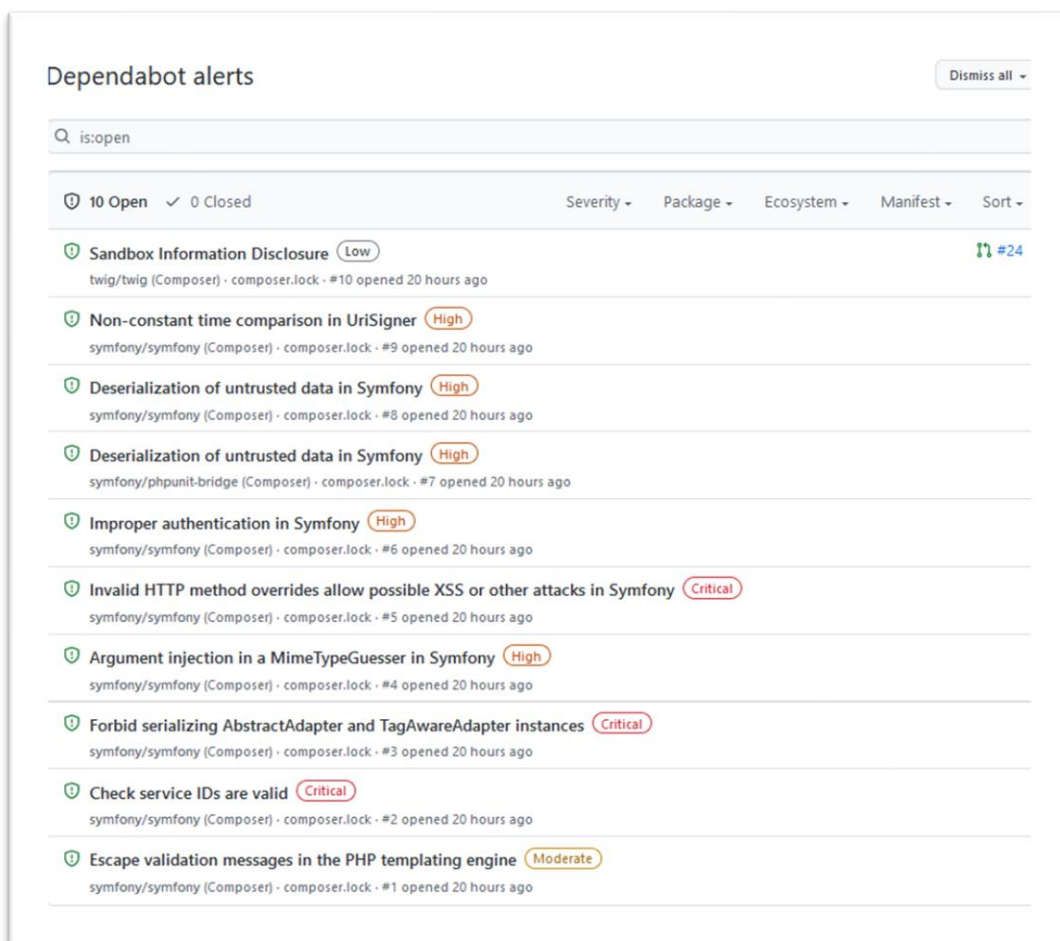


et 132 Messages de fonctionnalités marquées comme obsolètes sur la route : <https://127.0.0.1:8000/tasks/create>

La maintenabilité d'une application passe également par l'utilisation de composants à jour. Une vérification automatisée des versions des différents composants a été mise en place grâce à l'outil [Dependabot](#) intégré à la plateforme GitHub. Cet outil propose, par le biais de pull-request (PR), la mise à jour des différentes dépendances. Suite à l'analyse du projet initial par Dependabot (ci-dessous), 5 PRs ont été proposées afin de pallier ces dépréciations.



Les alertes Dependabot font apparaître des vulnérabilités de sécurité connues dans certains fichiers manifestes de dépendance. Les mises à jour de sécurité Dependabot maintiennent automatiquement l'application en mettant à jour les dépendances en réponse à ces alertes. Les mises à jour de version de Dependabot peuvent également aider à maintenir les dépendances à jour.



## Qualité du code

La qualité du code a été évaluée par une revue de code manuelle d'une part, et automatisée d'autre part. La qualité est un concept qui englobe la qualité du code, mais pas seulement. La qualité perçue par l'utilisateur de l'application ou encore la qualité perçue par les collaborateurs de l'entreprise sont des points essentiels à prendre en compte, ainsi que la qualité perçue par les personnes travaillant sur le projet.

### Analyse manuelle

Une 1ère lecture du code a permis d'observer une architecture de type MVC (**Model-View-Controller**) en adéquation avec les bonnes pratiques du Framework Symfony. On retrouve néanmoins une **architecture obsolète** due à une version ancienne de Symfony, avec notamment l'ensemble du code métier présent dans un sous-dossier **AppBundle** du dossier src. Dans les versions plus récentes, le code métier est situé directement dans le dossier src sous le **namespace App**.

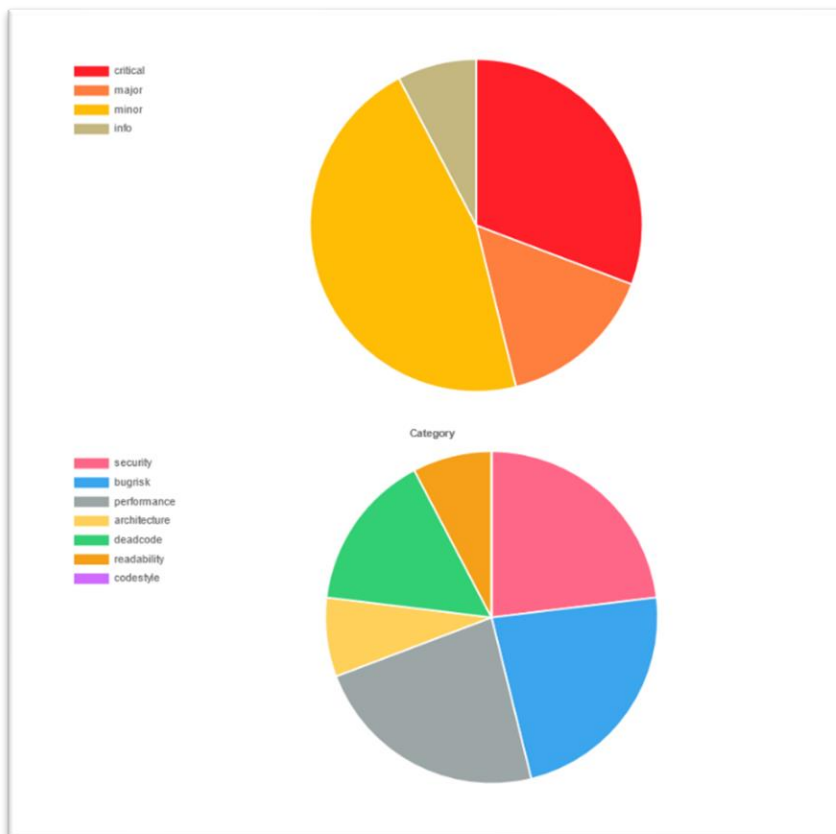
On retrouve également l'ensemble de la logique métier au sein des Controller. Nous améliorerons la maintenabilité et la compréhension du code en séparant d'avantage encore les couches métiers et fonctionnelles. Dans ce but, nous profiterons pleinement du Framework en mettant en place l'Injection de dépendances et supprimeront la nomenclature obsolète des appels aux container de services.

### Analyse automatisée

La qualité du code a été évaluée par l'outil d'automatisation **SymfonyInsight** qui permet de remonter différents problèmes pouvant être liés à la sécurité, la performance ou encore le style du code. L'utilisation de cet outil analytique a été associé au repository GitHub sur lequel seront apportées les modifications, afin de dresser une analyse récurrente pour chaque pull-request proposé et d'assurer le maintien d'un certain niveau de qualité au cours du développement. L'analyse **SymfonyInsight** préliminaire est présentée ci-dessous :

Cette analyse préliminaire, dévoile une note de **33/100** pour l'application à l'état initial, reflétant un certain nombre d'anomalies. Celles-ci, au nombre de 13, concernent principalement des problèmes mettant en péril la sécurité de l'application.

The screenshot displays the SymfonyInsight dashboard for a project. On the left sidebar, there's a progress indicator showing a score of 33/100 and a goal to reach the Platinum Medal in 2.8 days. Below this, a search bar and a severity filter are visible. The main area lists 13 suggestions, categorized by severity (Critical, Major, Minor, Info) and risk (Data leak, Productivity, Reliability, Reputation, Security, Uninsured). The suggestions include: 'The dependencies of your project could not be installed' (Critical, Uninsured), 'Your project must not rely on dependencies with known security issues' (Critical, Security), 'Your project must not expose sensitive infrastructure configuration' (Critical, Data leak), 'Your project must use a custom favicon instead of the default one' (Critical, Reputation), 'Your project should use Doctrine migrations' (Major, Reliability), 'Your project should not contain "FIXME" comments' (Major, Productivity), and 'Your project should not contain commented code' (Minor, Productivity). Each suggestion has a 'Read doc' and 'Ignore all' button. The bottom of the dashboard shows the last commit by Frédéric 16 days ago on the develop branch.



Le vérificateur a détecté 21 problèmes de sécurité dans le paquet Symfony/Symfony installé dans la version 3.1.6.0.

Des refactorisations permettant une diminution de la complexité de certains fichiers, ainsi que l'arrêt de l'utilisation de méthodes et variables non conseillées permettront entre autres de réduire significativement le nombre d'erreurs remontées.

### Tests : couverture du code de l'application

Un rapport de couverture du code de l'application initiale a été généré par le biais de PHPUnit et a révélé une absence totale de tests (rapport de couverture ci-dessous).

Code Coverage									
	Lines			Functions and Methods			Classes and Traits		
Total		0.00%	0 / 108		0.00%	0 / 37		0.00%	0 / 10
■ Controller		0.00%	0 / 55		0.00%	0 / 10		0.00%	0 / 4
■ DataFixtures		0.00%	0 / 23		0.00%	0 / 4		0.00%	0 / 2
■ Entity		0.00%	0 / 23		0.00%	0 / 21		0.00%	0 / 2
■ Form		0.00%	0 / 7		0.00%	0 / 2		0.00%	0 / 2
■ AppBundle.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 8.1.1 and PHPUnit 9.5.19 at Thu Mar 31 18:35:50 CEST 2022.

## Analyse fonctionnelle

L'analyse fonctionnelle a été réalisée après récupération du projet initial (version 3.1) sur GitHub et migrer à la version 3.4 de Symfony, sans altération majeure du code métier. En effet, l'incompatibilité de la version d'origine de Symfony rendait l'application inutilisable. La migration était donc nécessaire afin d'appréhender l'aspect fonctionnel.

### Fonctionnalités

Grâce aux schémas UML générés au début du projet, une évaluation des fonctionnalités initiales a permis de mettre en évidence quelques anomalies :

- Lien de l'en-tête vers la page d'accueil non fonctionnel.
- Les tâches, qu'elles soient terminées ou non, apparaissent lorsque l'on suit le lien "Consulter la liste des tâches à faire".
- Le lien "Consulter la liste des tâches terminées" n'est pas fonctionnel et aucune route n'est associée au sein des Controller.
- Lorsque l'on clique sur les boutons "Marque comme faite" ou "Marquer non terminée", le message de succès est bien affiché mais il est le même pour ces 2 actions : "Superbe ! La tâche a été marquée comme faite".
- Lorsque l'on crée un utilisateur avec un Nom d'utilisateur déjà présent dans la base de données, un message d'erreur "SQLSTATE [23000] : Integrity constraint violation : 1062 Duplicate entry 'test1' for key 'UNIQ\_8D93D649F85E0677'" apparaît. Cette information devrait être remontée lors de la validation du formulaire avec un message précisant qu'il ne peut y avoir plusieurs utilisateurs avec le même nom.
- Manque de contraintes de validation notamment sur le nombre de caractères minimum pour le mot de passe.

En termes de sécurité, l'utilisation du composant Security de Symfony permet de correctement restreindre l'accès à l'application aux utilisateurs enregistrés.



## Qualité

Pour cette analyse nous nous sommes focalisés sur la qualité globale de l'application perçue par les utilisateurs.

- Navigation utilisateur limitée, peu de liens entre les pages tels que “retour à la liste des tâches, liste des utilisateurs, lien vers la page des tâches terminée que sur la page d'accueil...).
- Visuel non attractif.
- Le titres des pages est toujours le même : “ To Do List app ”.
- Obligation de renseigner le mot de passe lors de l'édition d'un utilisateur.
- Tout le contenu de la tâche n'est pas visible si long descriptif.
- Pas de page d'accueil accessible par les utilisateurs non authentifiés : redirection automatique vers la page de login.
- Mélange d'anglais et français.
- Pages d'erreurs inexistantes.
- Pas de demande de confirmation de suppression des tâches.

## Synthèse : Inventaire de la dette technique

### Points d'amélioration identifiés par ToDo & Co

Une tâche doit être attachée à un utilisateur :

- ✓ Mise à jour de la structure de la base de données avec implémentation d'une relation entre les tables User et Task.
- ✓ Lors de la création de la tâche, l'utilisateur authentifié est défini comme l'auteur de la tâche.
- ✓ L'auteur ne peut pas être modifié lors de la modification d'une tâche.
- ✓ Rattacher les tâches déjà créées à un utilisateur anonyme.

Choisir un rôle pour l'utilisateur :

- ✓ Implémentation des rôles ROLE\_USER et ROLE\_ADMIN.
- ✓ Possibilité de modifier le rôle lors de l'édition d'un utilisateur.

Suppression des tâches par l'auteur seulement :

- ✓ Seul l'auteur d'une tâche est autorisé à la supprimer.
- ✓ Les tâches rattachées à un utilisateur "anonyme" ne peuvent être supprimées que par un administrateur.

Implémentation de tests automatisés :

- ✓ Implémentation des tests unitaires avec PHPUnit.
- ✓ Implémentation des tests fonctionnels avec PHPUnit.
- ✓ Etablir un rapport de couverture de code (>70%).

Points d'amélioration identifiés lors de l'état des lieux dont la mise en place est nécessaire au bon fonctionnement de l'application au regard des besoins explicités par ToDo & Co

Fichiers manquants :

- ✓ Il manque le fichier « jquery.js » dans le dossier : \public\js. Les liens dans le fichier « base.html.twig » seront simplement remplacés par les liens CDN de la version actuellement utilisée de Bootstrap ([v3.3.7](#)).
- ✓ Il n'y a pas de favicon.ico n'y de site.webmanifest Ils seront ajoutés dans le dossier public de l'application.

Obsolescence de la version du Framework Symfony :

- ✓ Nouvelle installation de Symfony dans sa version la plus récente.

Obsolescence des composants utilisés :

- ✓ Du fait de la nouvelle installation, les dépendances seront à jour dans leurs dernières versions.
- ✓ Mise en place du suivi par l'outil Dependabot sur GitHub pour assurer une application toujours à jour.

Code non documenté :

- ✓ Documenter le code afin de faciliter la compréhension et la maintenance par d'autres développeurs.

Analyse qualité SymfonyInsight note 33/100 :

- ✓ Obtenir une médaille d'argent au minimum.
- ✓ Mise en place d'un suivi permettant le maintien de ce statut.

Complexité du code :

- ✓ Extraire la logique des contrôleurs.

Implémentation de tests automatisés :

- ✓ Mise en place d'un outil d'intégration continue (**TravisCI**) permettant de lancer une suite de tests prédéfinis à chaque PR afin de vérifier l'intégrité de l'application avant de merger les modifications proposées.

Points d'amélioration identifiés lors de l'état des lieux ne mettant pas en péril le bon fonctionnement de l'application et dont la mise en place pourra faire l'objet d'amélioration continue de l'application

Qualité utilisateur :

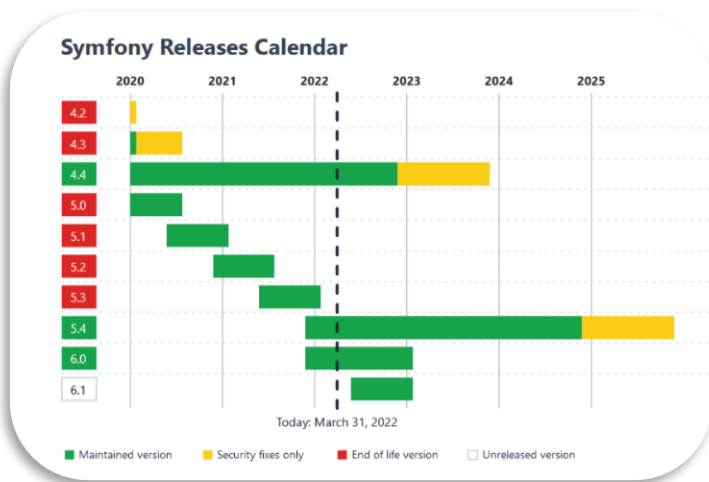
- ✓ Amélioration de l'aspect visuel général de l'application.
- ✓ Amélioration du responsive design.
- ✓ Amélioration de l'affichage des tâches (visibilité du descriptif).
- ✓ Amélioration de la navigation (ajout de lien entre les différentes pages).
- ✓ Implémentation d'un système de traduction.
- ✓ Création des pages d'erreurs.

Sécurité :

- ✓ Activation d'un compte utilisateur par email.
- ✓ Mise en place d'une demande de confirmation ainsi que d'une protection CSRF pour la suppression des tâches.

## Conclusion de la dette technique

Avec tous ces éléments, et au vu de la petite taille de l'application, nous estimons que cela prendrait beaucoup trop de temps pour faire toutes les migrations vers la dernière version de Symfony. Nous



préconisons donc la refonte totale du site afin de profiter des améliorations du Framework, et nous avons décidés de recréer une nouvelle installation de Symfony pour bénéficier des avantages de la version la plus récente, à savoir la dernière version stable 6.0 et PHP 8 en attendant de pouvoir migrer vers la prochaine LTS (**Long Term Support**) de Symfony (6.4).

# Rapport d'Audit de qualité et de performances

Suite à l'état des lieux de l'application ToDo & Co, de nouvelles fonctionnalités ainsi qu'un certain nombre d'actions correctives et d'améliorations ont été menées selon le plan d'action déterminé dans la partie précédente. A l'issue de ces améliorations, un audit de qualité et de performances a été réalisé et les résultats sont présentés ci-après.

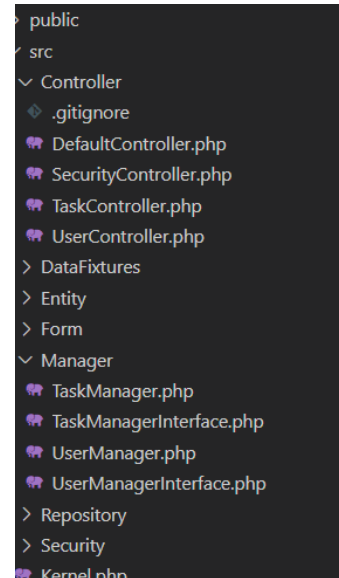
## Qualité du code

### Architecture

Du fait de la nouvelle installation de Symfony dans sa version la plus récente, l'architecture des fichiers a subi un changement conséquent.

Le code métier, initialement présent au sein d'un dossier AppBundle dans le dossier src est maintenant directement présent au sein de ce même répertoire qui correspond au namespace App\ . L'architecture MVC est conservée, mais la majorité de la logique initialement présente au sein des contrôleurs est maintenant répartie entre des **Managers** et leurs **Interfaces** (TaskManager, TaskManagerInterface et UserManager, UserManagerInterface), dont le rôle est d'effectuer les actions sur les données et faire le lien avec les Repository en charge de persister les données en base de données. Les contrôleurs ne conservent donc que la responsabilité de traiter la requête et de renvoyer une réponse adaptée.

Le code ainsi remodelé est amené à être plus maintenable et évolutif.



### Documentation

Afin d'améliorer la compréhension du code par des personnes extérieures, toutes les méthodes du code métier ont été documentées, avec une description du rôle de la méthode ainsi que le type des paramètres et de la réponse attendus.

















### SymfonyInsight

Alors que l'analyse préliminaire de SymfonyInsight indiquait une note de 33/100, les remaniements du code ont permis une réduction de la dette qualitative, avec l'obtention d'une note de 100/100 avec une médaille de platine. Grace à la nouvelle installation de Symfony, il n'y a plus aucune anomalie. Comme évoqué précédemment, cette analyse a été intégrée à GitHub et est donc réalisée à chaque Pull Request, afin de maintenir ce niveau de qualité au cours des futurs développements.



## Tests : couverture du code de l'application

L'analyse préliminaire ayant mis en évidence une absence totale de tests pour l'application, la 1ère étape du développement a été de mettre en place des tests unitaires, fonctionnels et d'intégration pour les fonctionnalités. Après la réécriture du code, la correction des anomalies et l'implémentation des nouvelles fonctionnalités, les tests ont donc été implémentés après l'écriture du code métier.

	Code Coverage					
	Lines			Functions and Methods		
Total		85.87%	158 / 184		94.37%	67 / 71
■ Controller		100.00%	58 / 58		100.00%	13 / 13
■ DataFixtures		0.00%	0 / 24		0.00%	0 / 3
■ Entity		100.00%	49 / 49		100.00%	29 / 29
■ Form		100.00%	11 / 11		100.00%	4 / 4
■ Manager		100.00%	20 / 20		100.00%	11 / 11
■ Repository		100.00%	10 / 10		100.00%	7 / 7
■ Security		83.33%	10 / 12		75.00%	3 / 4
Kernel.php		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 8.1.1](#) and [PHPUnit 9.5.20](#) at Sat Apr 16 10:10:06 CEST 2022.

Un taux de couverture de 70% était demandé pour ce projet, une couverture > 85% a été implémentée afin de couvrir la majorité du code métier de l'application et de s'assurer que les différentes fonctionnalités ne seront pas altérées par les développements futurs.

En ce sens, un outil d'intégration continue a été intégré à GitHub (**Travis CI**) et permet de vérifier l'ensemble des tests à chaque nouveau PR ou commit.

```
$ composer install
$ php bin/console doctrine:database:create --env=test
$ php bin/console doctrine:schema:update --force --env=test
$ php bin/console doctrine:fixtures:load -n --env=test
$ php bin/phpunit
PHPUnit 9.5.20 Standards
Testing
.....
38 / 38 (100%)

Time: 00:03.649, Memory: 66.50 MB

The command "php bin/phpunit" exited with 0.

Done. Your build exited with 0.
```

## Performance de l'application

Les améliorations en termes de performance de l'application ont été mises en évidence grâce à l'outil d'analyse Blackfire. Une analyse initiale de l'application a été réalisée juste après une première mise à jour vers la version 3.4 de Symfony, l'application étant non fonctionnelle en son état initial.

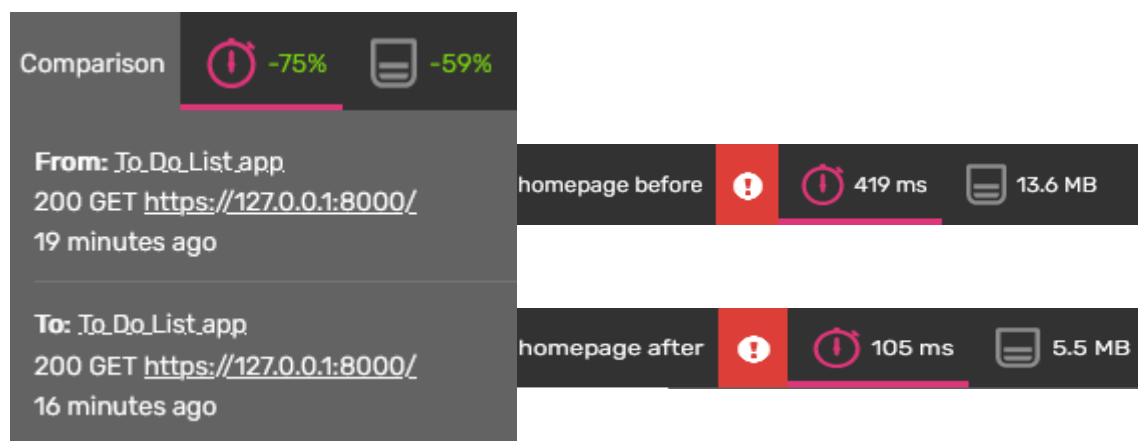
Un comparatif a été réalisé sur 6 routes de l'application :

- ✓ La page d'accueil (/)
- ✓ La page login (/login)
- ✓ La page liste des utilisateurs (/users)
- ✓ La page création utilisateur (/users/create)
- ✓ La page des tâches (/tasks pour la version initiale et /tasks/todo pour la version finale)
- ✓ La page créer une tâche (/tasks/create)

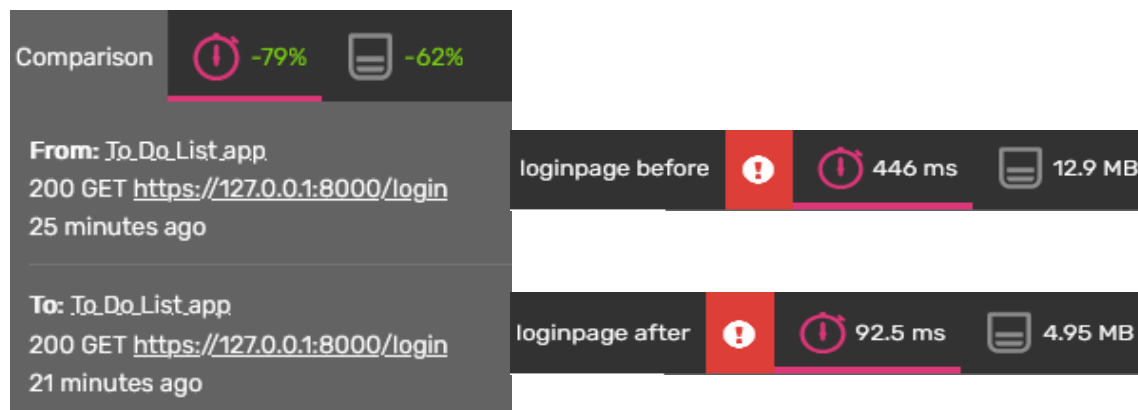
Les figures ci-dessous illustrent les comparatifs des analyses Blackfire réalisées dans l'application d'origine (**before**) et après réécriture et amélioration dans la nouvelle version de Symfony (**after**), en termes de performance grâce à deux mesures : le temps de chargement de la page et la mémoire allouée.

Comme on peut le constater, le temps de chargement des différentes pages (requêtes GET) a été en moyenne diminué de 75% grâce aux optimisations, passant d'un temps de chargement d'environ 420ms à un temps de chargement plus ou moins égale à 100ms, confirmant une meilleure réactivité de l'application à l'issue des améliorations.

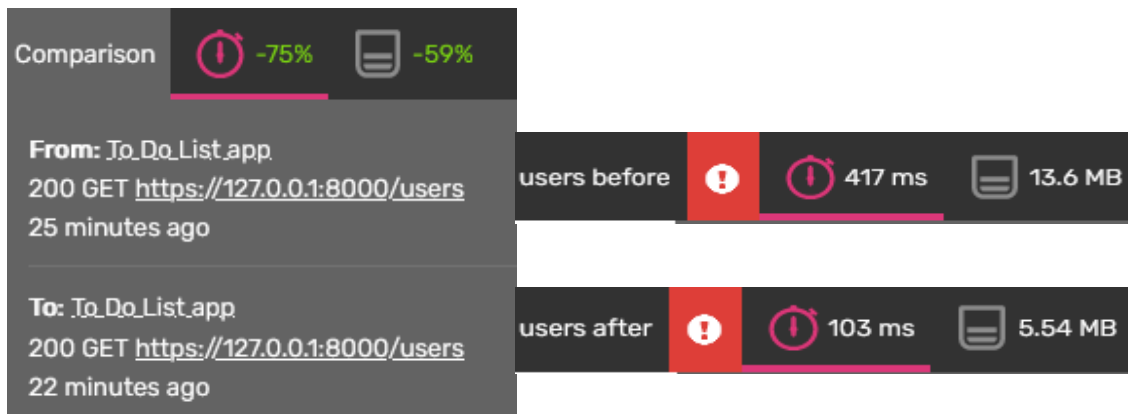
Route : <https://127.0.0.1:8000/>



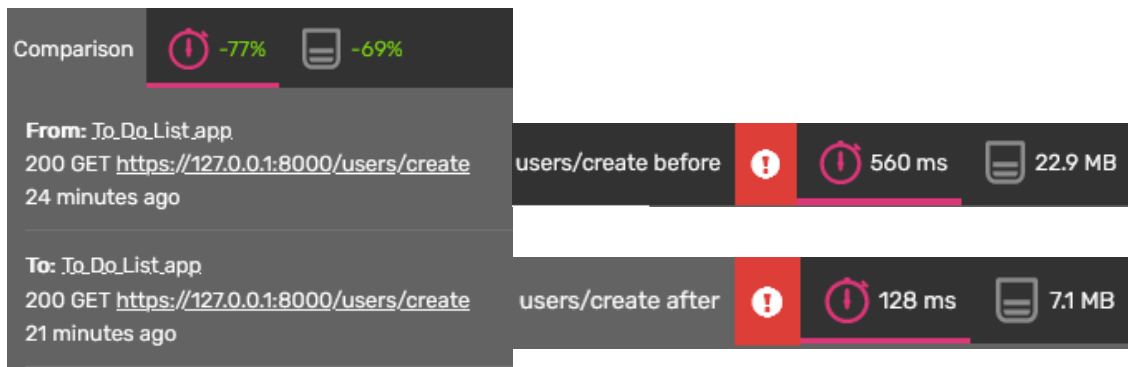
Route : <https://127.0.0.1:8000/login>



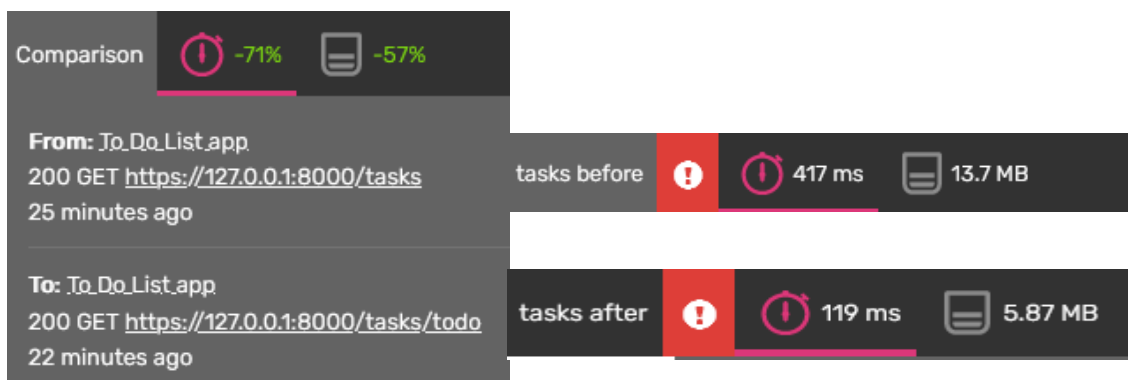
Route : <https://127.0.0.1:8000/users>



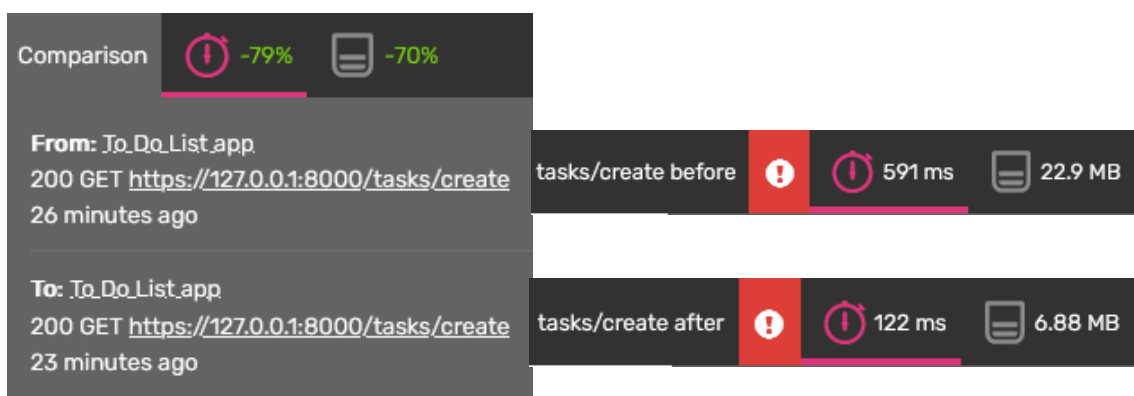
Route : <https://127.0.0.1:8000/users/create>



Route : <https://127.0.0.1:8000/tasks>



Route : <https://127.0.0.1:8000/tasks/create>





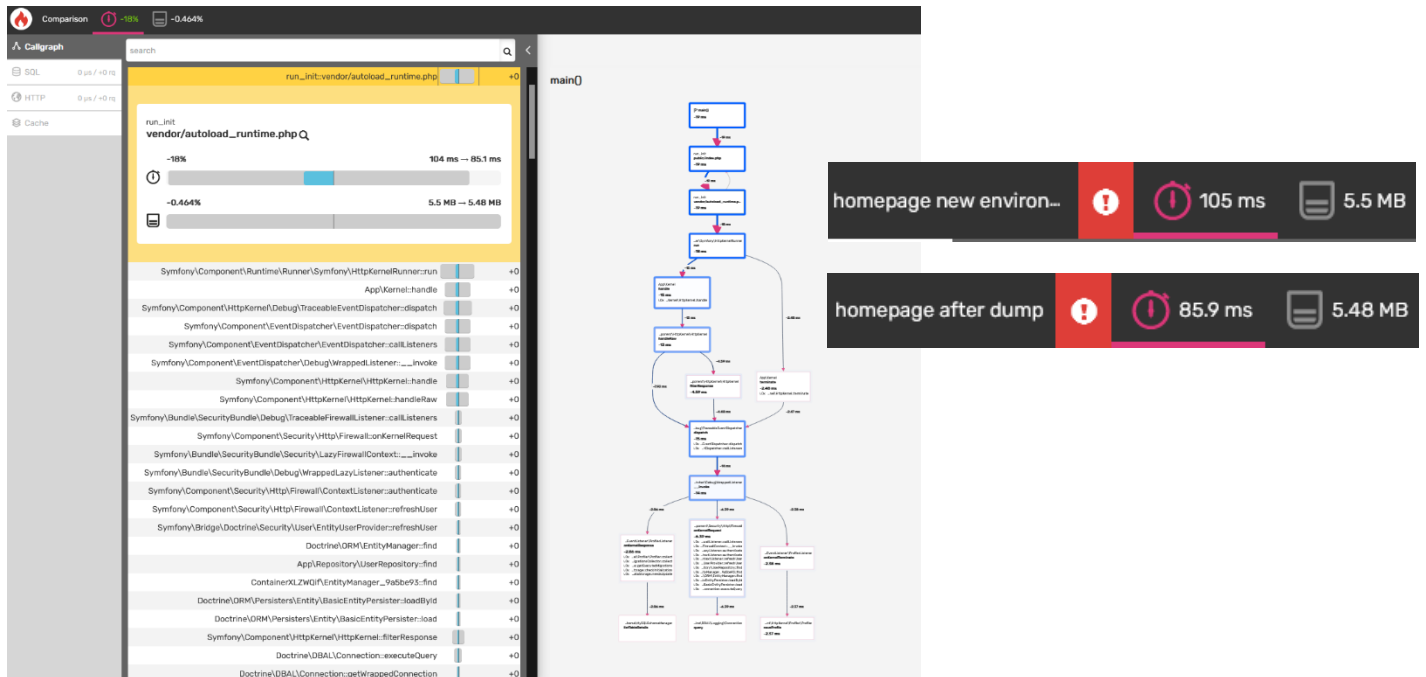
En poussant l'analyse comparative Blackfire grâce aux graphiques de comparaison fournis, il ressort que la principale raison de l'amélioration des performances en termes de rapidité provient de la diminution du nombre d'appels des fonctions de l'autoloader de Composer. Cette amélioration semble être en lien avec la mise à jour vers la version la plus récente du Framework.

Nous avons après cette première comparaison, optimisé l'autoloader grâce à la commande :

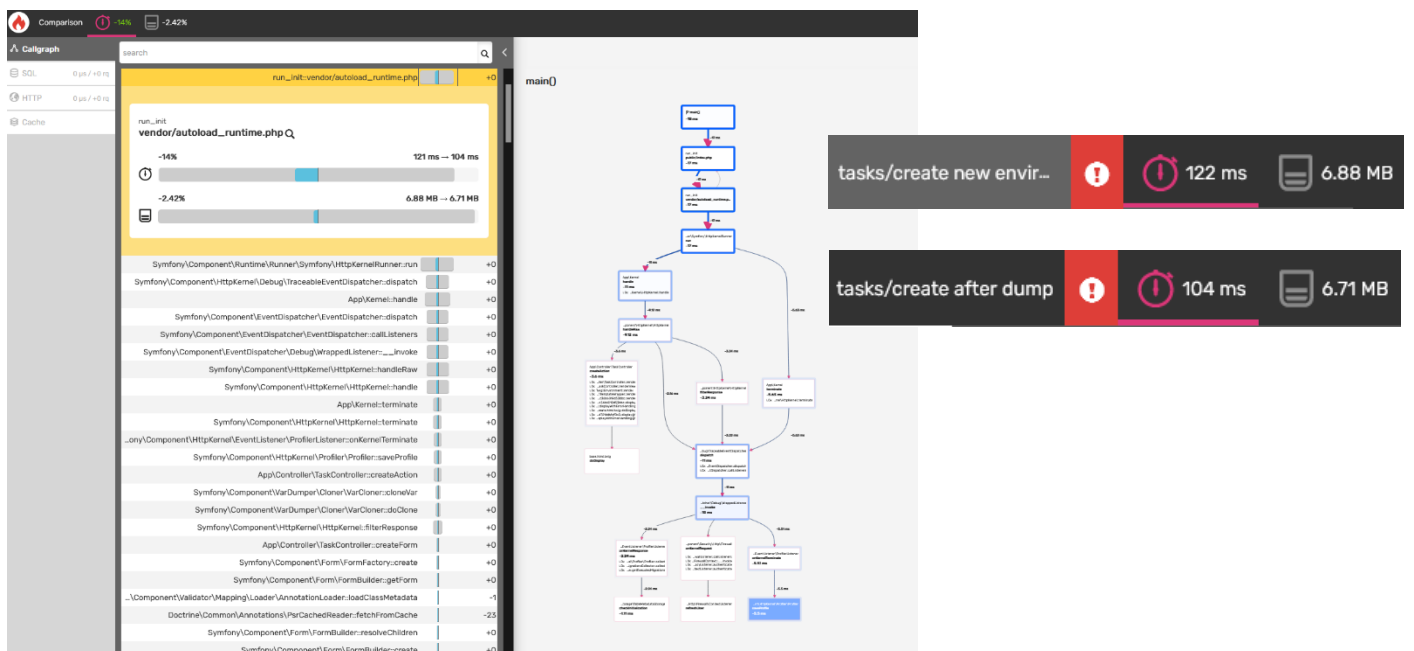
➤ `composer dump-autoload -o`

Nous avons pu ainsi gagner encore quelques millisecondes sur l'affichage des pages :

Route : <https://127.0.0.1:8000>



Route : <https://127.0.0.1:8000/tasks/create>



## Bilan et plan d'amélioration

A l'initialisation de ce projet d'amélioration de l'application ToDo & Co, l'état des lieux de la dette technique dressé dans la 1ère partie de ce document a permis d'identifier un certain nombre d'anomalies, comprenant celles explicitement spécifiées par le client mais également d'autres anomalies révélées par une analyse technique et fonctionnelle.

Après implémentation de correctifs et implémentation des nouvelles fonctionnalités demandées par le client, l'audit de qualité et de performance a mis en évidence une amélioration de la qualité du code (**SymfonyInsight**), du taux de couverture du code par des tests unitaires et fonctionnels (**PHPUnit**) et de la maintenance (outils d'amélioration continue **TravisCI**, **Dependabot**), ainsi que de la performance (**Blackfire**).

Néanmoins, la qualité d'une application ne se résume pas à la qualité du code mais prend en compte également la qualité perçue par les utilisateurs. Or un certain nombre de pistes d'améliorations pourraient grandement améliorer la qualité de l'application perçue par les utilisateurs.

Afin d'orienter les futurs développements, une proposition de plan d'action regroupant plusieurs pistes d'améliorations est proposée ci-dessous ainsi que sur le projet [Kanban Improvement plan of ToDo&Co](#).

Améliorations globales :

- Upgrader Bootstrap dans sa version la plus récente.
- Améliorer l'esthétique et le responsive design de l'application.

Fonctionnalités :

- Donner la possibilité à l'Administrateur de supprimer un utilisateur, avec une demande de confirmation avant la suppression. Deux choix devront être possible pour les éventuelles tâches liées à cet utilisateur. Soit les tâches seront conservées avec un statut anonyme, soit elles seront supprimées en même temps que l'utilisateur.
- Faire en sorte que l'administrateur ne soit pas obligé de renseigner à chaque fois le mot de passe lors de la modification d'un utilisateur.
- Ajouter une date de fin souhaitée à une tâche et afficher une alerte lorsque cette date arrive à échéance.
- Mettre en place un système d'activation par e-mail pour la création de nouveaux utilisateurs.
- Ajouter un système de pagination en cas de nombre élevé de tâches.
- Améliorer la sécurité de l'application en ajoutant une confirmation de suppression des tâches.
- Créer une page profil afin de donner la possibilité à un utilisateur de modifier ses informations et son mot de passe même s'il n'est pas administrateur.

Performance :

- Dans la même perspective d'amélioration continue, il serait intéressant de mettre en place des tests de performance grâce à l'outil Blackfire, afin de vérifier la performance dans le temps et s'assurer de l'absence d'introduction de bug de performance au cours des développements.