ToDo&Co

Documentation technique
Audit de qualité et de performance

Auteur: Charlotte SAURY
Dernière mise à jour: 27/10/2020
Version: 1.0

Sommaire

Introduction

Contexte du projet

ToDo&Co est une startup dont le cœur de métier est une application permettant de gérer ses tâches quotidiennes. Cette application a dans un 1er temps été développée à l'état de concept viable, ou MVP (Minimum Viable Product), afin de démontrer une preuve de concept aux potentiels investisseurs.

Objectifs du projet

Ayant récemment obtenu une levée de fond pour le développement de l'entreprise, le projet m'a été confié afin de prendre en charge l'amélioration de l'application.
En ce sens, 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 notre application web sera réalisée en plusieurs étapes.

1. Dans un 1er temps, 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 dans un second temps 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 (ou TDD pour Test Driven Development) consistant à concevoir pas à pas et de façon itérative et incrémentale, en écrivant chaque test avant d'écrire le code source et en remaniant le code continuellement.

Analyse préliminaire du projet

Cette 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

L'analyse technique a été réalisée dès la récupération du projet initial sur GitHub.

➔ Environnement technique

Dans un 1er temps, 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 2018 du framework Symfony (**version 3.1.6**), et d'autre part une dépréciation de plusieurs composants du framework.

Or, l'utilisation d'une **version stable** du framework, dite **LTS (Long Term Support)**, est essentielle pour garantir la pérennité du développement de l'application. La dernière version LTS actuellement maintenue étant la version 4.4, les premières actions correctives ont visé à faire évoluer le projet initial vers cette version du framework Symfony. Afin de profiter des améliorations du framework, nous avons également effectué les modifications pour bénéficier des avantages de la version la plus récente, à savoir la dernière version stable 5.1 en réalisant les différentes mises à jour mineures en attendant de pouvoir migrer vers la prochaine LTS (5.4).

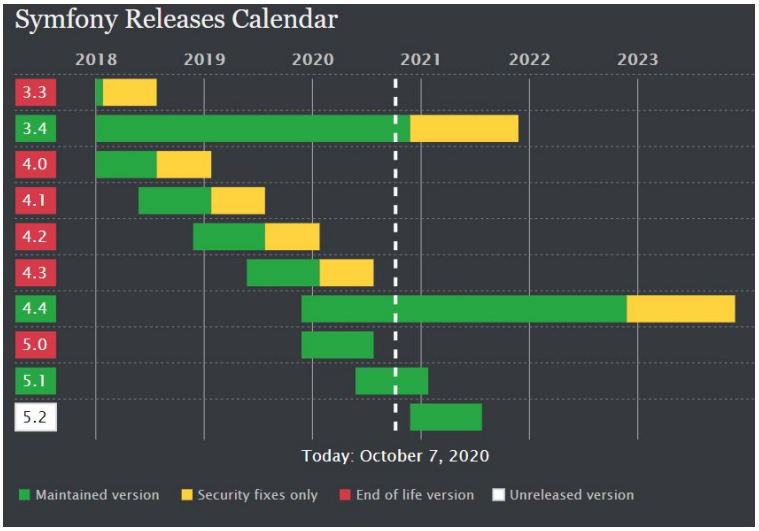


Figure 1: Calendrier de parution des dernières versions de Symfony. (source: https://symfony.com/releases)

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**, nouvellement 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), 6 PRs ont été proposées afin de pallier ces dépréciations.

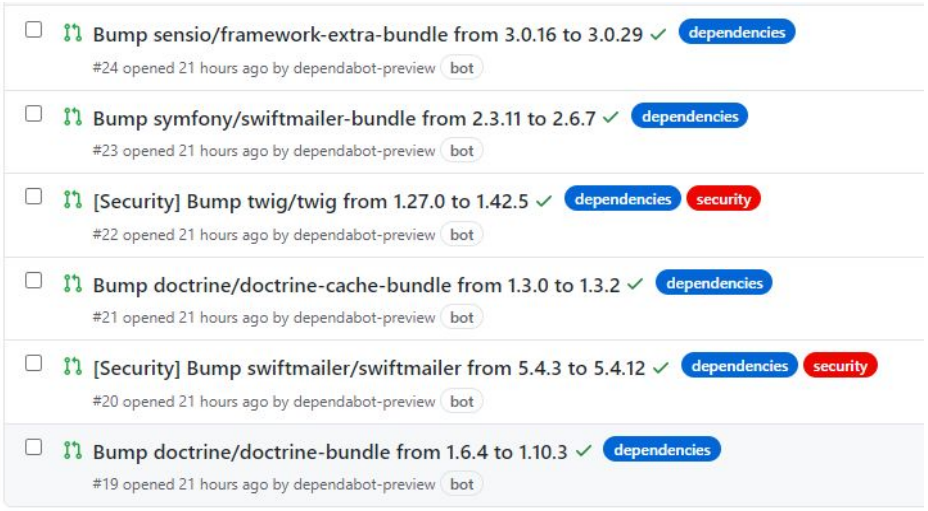


*Figure 2: Pull-requests correctives proposées par **Dependabot** pour pallier la dépréciation des composants SF.*

➔ 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 controllers, ce qui sera amélioré par la création de services (Manager) afin d'assurer une meilleure maintenabilité et compréhension du code. Finalement, on retrouve au sein de ces controllers l'appel au container de services via une nomenclature obsolète, ce qui sera simplifié par le système d'**injection de dépendances** (cf figure suivante: implémentation de l'injection de dépendances pour le service AuthenticationUtils).

```php
/**
 * @Route("/login", name="login")
 */
public function loginAction(Request $request, AuthenticationUtils $authenticationUtils)
{
    $error = $authenticationUtils->getLastAuthenticationError();
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render('security/login.html.twig', array(
        'last_username' => $lastUsername,
        'error'         => $error,
    ));
}
```

Figure 3: Implémentation de l'injection de dépendances, absente de la version initiale du framework et mise en place lors de la mise à jour vers la version 5.1 de Symfony

Analyse automatisée

La qualité du code a été évaluée par l'outil d'automatisation **Codacy** 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 codacy préliminaire est présentée ci-dessous:

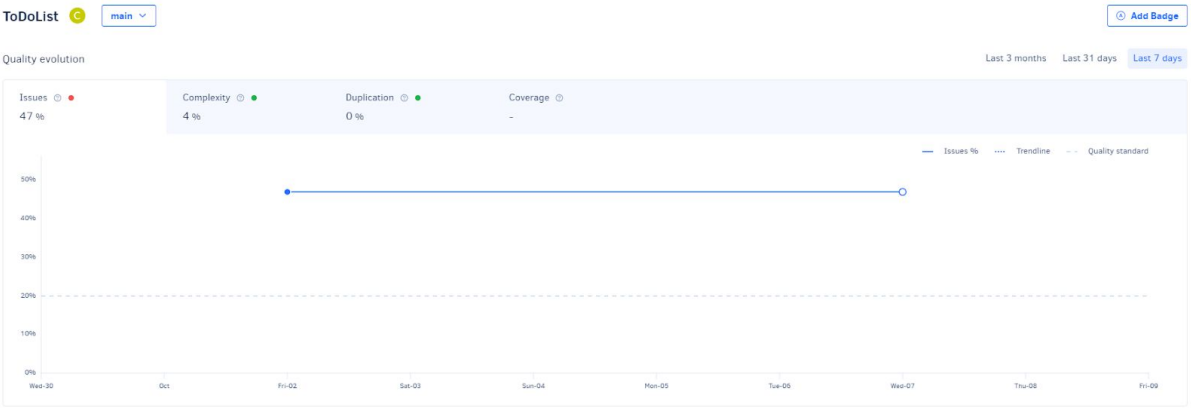


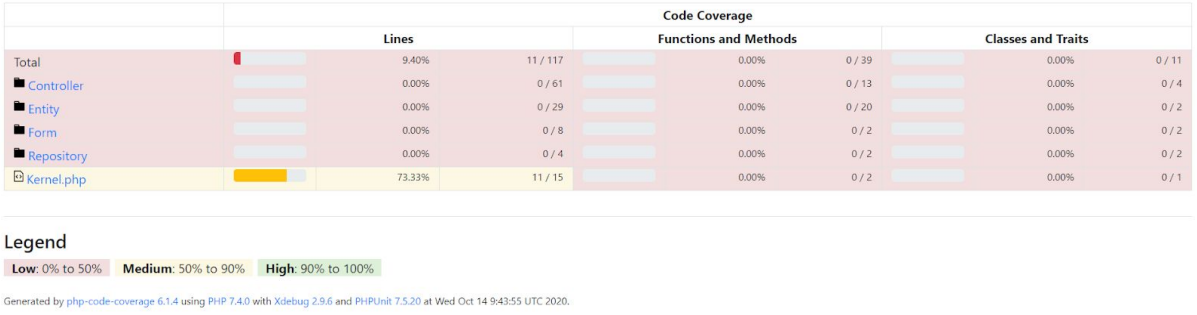
Figure 4: Analyse Codacy initiale.

L'analyse Codacy préliminaire, dont le récapitulatif est présenté ci-dessous, dévoile une note de **grade C** pour l'application à l'état initial, reflétant un certain nombre d'anomalies. Celles-ci, au nombre de 74, concernent principalement le style du code ainsi que des problèmes mettant en péril la sécurité de l'application. 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	9.40%	11 / 117		0.00%	0 / 39		0.00%	0 / 11
Controller	0.00%	0 / 61		0.00%	0 / 13		0.00%	0 / 4
Entity	0.00%	0 / 29		0.00%	0 / 20		0.00%	0 / 2
Form	0.00%	0 / 8		0.00%	0 / 2		0.00%	0 / 2
Repository	0.00%	0 / 4		0.00%	0 / 2		0.00%	0 / 2
Kernel.php	73.33%	11 / 15		0.00%	0 / 2		0.00%	0 / 1

Figure 5: Rapport de couverture du code de l'application initiale par les tests (PHPUnit)

Analyse fonctionnelle

L'analyse fonctionnelle a été réalisée après récupération du projet initial sur GitHub et migration à la version 3.3 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, et au vu de la petite taille de l'application, 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;
- Toutes les tâches apparaissent (terminées/non terminées) lorsque l'on suit le lien "Consulter la liste des tâches à faire";
- Lien "Consulter la liste des tâches terminées" non fonctionnel et aucune route associée au sein des controllers;
- Lorsque l'on clique sur les boutons "Marque comme faite" ou "Marquer comme non terminée", un message de succès est bien affiché mais toujours le même message "Superbe! La tâche a été marquée comme faite";
- Manque de contraintes de validation notamment pour le mot de passe

Certaines de ces anomalies, comme les liens non fonctionnels et la présence de toutes les tâches dans la liste des tâches non terminées ont été confirmées lors de la mise en place des tests fonctionnels avant de mettre en place les actions d'amélioration.
En termes de sécurité, l'utilisation du composant Security de Symfony permet de correctement restreindre l'accès à l'application aux utilisateurs enregistrés.

➔ Qualité

Cette analyse s'est ici focalisée 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
- Pas de titres des pages
- 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 standards
- Pas de confirmation de suppression des tâches

Synthèse: Inventaire de la dette technique

Points d'amélioration	Plan d'action
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 l'édition de la 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 pas 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	

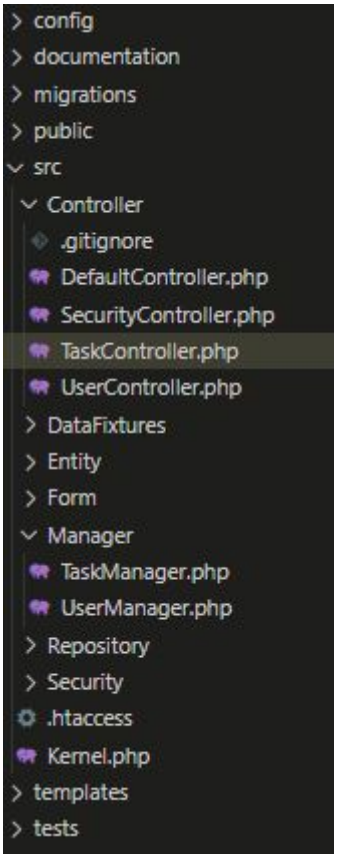
Obsolescence de la version du framework Symfony	- Migration vers version du framework LTS la plus récente
Obsolescence des composants utilisés	- Mise à jour des dépendances - Identification des dépendances qui ne sont plus supportées - Mise en place du suivi par l'outil Dependabot sur GitHub pour assurer 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 Codacy Grade C	- Obtenir un grade A en réduisant le nombre d'erreurs. - 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 - Amélioration 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

Rapport d'Audit de qualité et de performances

Suite à l'état des lieux de l'application ToDo&C, 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-dessous.

Qualité du code

Architecture



Afin de respecter l'évolution de l'architecture des fichiers des versions récentes de Symfony, un remaniement conséquent a été réalisé.
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 manageurs (TaskManager et UserManager), 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.

Figure 6: Architecture du code de l'application en sa version améliorée

Documentation

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

Codacy

Alors que l'analyse préliminaire de Codacy indiquait un grade C, les remaniements du code ont permis une réduction de la dette qualitative, avec l'obtention d'un badge grade A, et la correction d'un certain nombre d'anomalies. En effet, sur les 74 anomalies repérées initialement, seules 6 anomalies mineures persistent. 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 existantes attendues. L'implémentation de ces tests a également permis de confirmer certaines anomalies observées lors de l'analyse fonctionnelle manuelle, comme la présence de l'ensemble des tâches faites et non faites sur la liste des tâches à faire ou encore l'invalidité des liens vers la page d'accueil et vers la page des tâches terminées. Après la correction des anomalies et la vérification que l'ensemble des tests étaient positifs, l'implémentation des nouvelles fonctionnalités a été réalisée par la méthode TDD. Les tests ont donc été implémentés avant l'écriture du code métier.

	Code Coverage					
	Lines			Functions and Methods		
Total	97.39%	224 / 230		93.94%	62 / 66	
Controller	98.31%	58 / 59		92.86%	13 / 14	
DataFixtures	100.00%	58 / 58		100.00%	7 / 7	
Entity	94.12%	48 / 51		92.86%	26 / 28	
Form	100.00%	16 / 16		100.00%	3 / 3	
Manager	100.00%	28 / 28		100.00%	8 / 8	
Repository	100.00%	4 / 4		100.00%	2 / 2	
Security	85.71%	12 / 14		75.00%	3 / 4	

Legend

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

Figure 7: Couverture du code de l'application par les tests PHPUnit.

Alors qu'un taux de couverture de 70% était demandé pour ce projet, une couverture > 90% 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 nouvelle PR ou commit (figure ci-dessous).

```
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

Testing Project Test Suite
.....................................................           56 / 56 (100%)

Time: 44.96 seconds, Memory: 88.50 MB

OK (56 tests, 201 assertions)
The command "php bin/phpunit" exited with 0.


Done. Your build exited with 0.
```

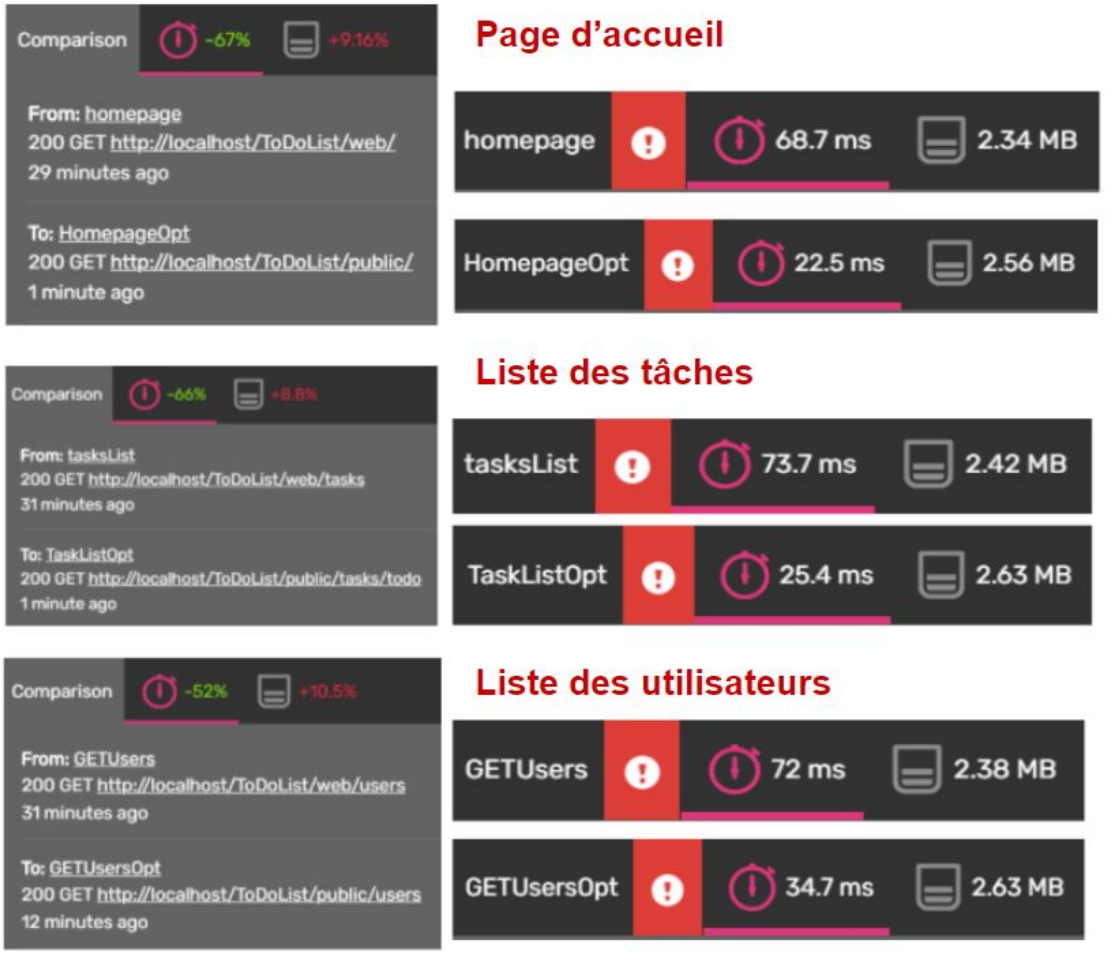
Figure 8: Exemple de build réalisé avec succès par Travis CI lors d'une pull-request.

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 3.3 de SF, l'application étant non fonctionnelle en son état initial.
Une comparatif a été réalisé sur 3 routes de l'application en GET et une route en POST:

- la page d'accueil (/)
- la page des tâches (/tasks pour la version initiale et /tasks/todo pour la version finale) avec 10 tâches
- la liste des utilisateurs (/users) avec 10 utilisateurs
- Action login en POST

La figure ci-dessous illustre les comparatifs des analyses Blackfire réalisées avant et après amélioration, 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 au minimum divisé par 2 grâce aux optimisations, passant d'un temps de chargement d'environ 70ms à un temps de chargement inférieur à 35ms, confirmant une meilleure réactivité de l'application à l'issue des améliorations.



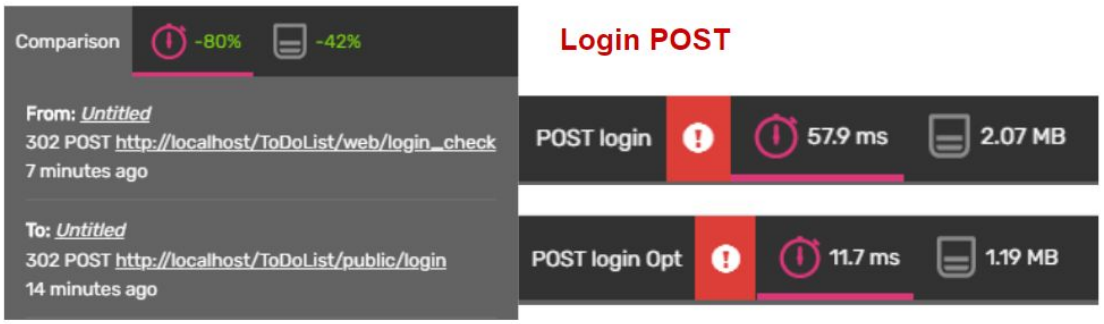


Figure 9: Comparatif de l'application avant et après améliorations en termes de performance, grâce à la mesure du temps de chargement de la page et de la mémoire allouée par l'outil d'analyse Blackfire.

En revanche, cette analyse révèle une légère augmentation de l'ordre de 10% de la mémoire allouée, ce qui reste néanmoins négligeable. Le choix de l'optimisation du temps de chargement de la page reste en effet à privilégier car directement en rapport avec le client, alors que cette légère augmentation de la mémoire impacte peu les performances au vu de la taille des serveurs actuels. De la même façon, une amélioration significative de la performance a été observée pour la requête POST de login, avec une diminution de 80% du temps de chargement et de 42% de mémoire allouée.

En poussant l'analyse comparative Blackfire grâce aux graphiques de comparaison fournis (figures ci-dessous), 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 file_exists et includeFile de l'autoloader de Composer. Cette amélioration semble être en lien avec la génération d'un autoloader optimisé grâce à la commande "composer dump-autoload -o", plus qu'avec la mise à jour vers une version récente du framework.

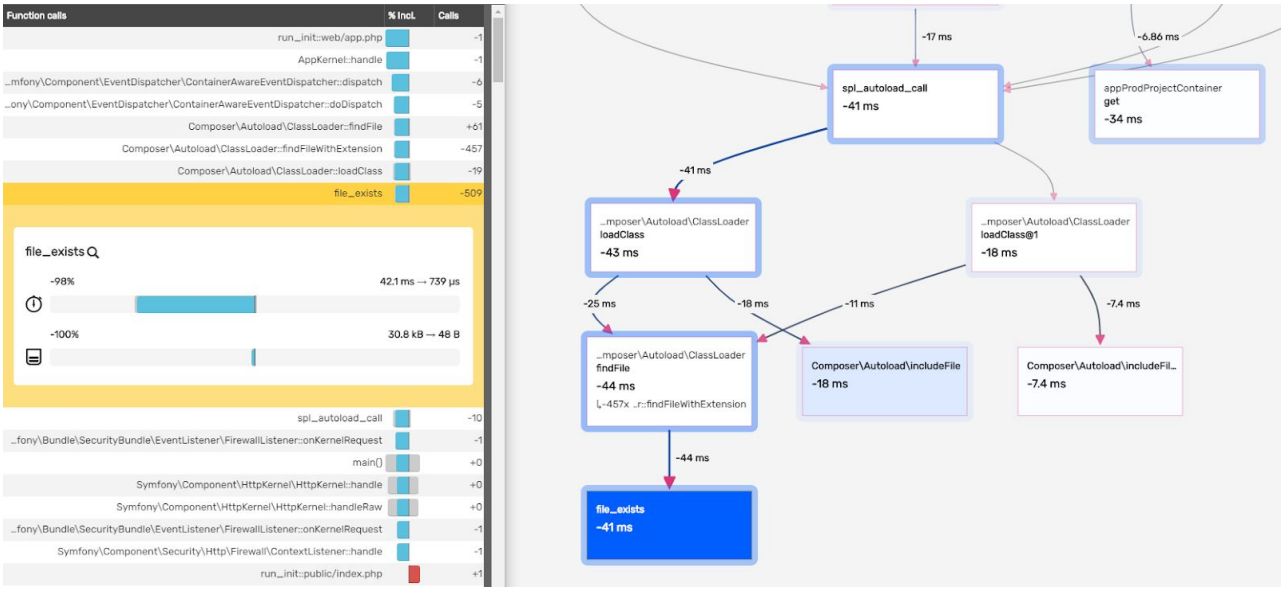


Figure 10: Comparatif des fonctions appelées au cours du chargement de la page /users avant et après améliorations.

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 (codacy), 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 améliorations est proposée ci-dessous ainsi que sur le projet Kanban ToDo&Co Improvment Plan.

Améliorations globales:

- Améliorer l'esthétique de l'application et le responsive design
- Traduction: mettre en place un système de traduction pour l'ensemble des messages d'erreurs et message flash, qui aujourd'hui ne sont pas homogènes

Fonctionnalités:

- Améliorer la gestion des tâches: Pagination, filtres, ajout d'un délai de réalisation à respecter avec envoi de notifications, possibilité de commenter les tâches, ajout d'un indice de progression de la réalisation de la tâche...
- Ajouter la possibilité de supprimer un utilisateur, dans la limite d'avoir toujours au moins un compte administrateur actif
- Améliorer la sécurité de l'application en ajoutant une confirmation de suppression des tâches, ainsi qu'une protection CSRF
- Améliorer la sécurité de l'application en ajoutant une vérification de l'email lors de l'inscription et en laissant la possibilité à l'utilisateur de modifier 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