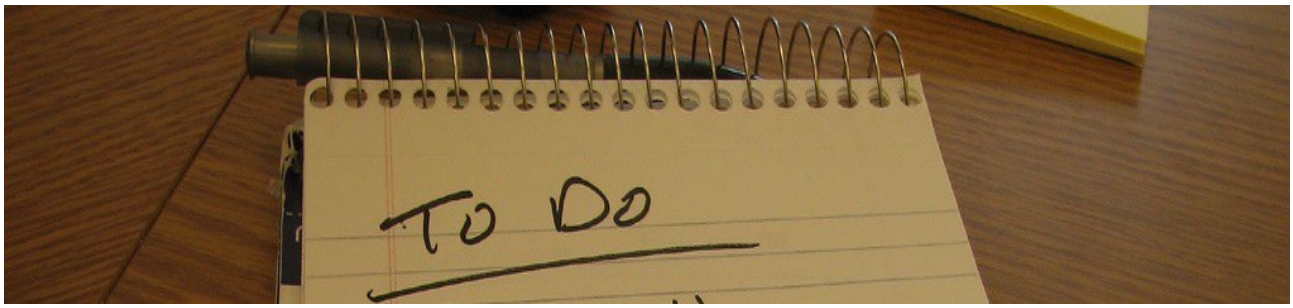


AUDIT DE QUALITE ET PERFORMANCE

Application Web ToDoList

Réalisé par Gallot Benjamin
Développeur d'Application Web PHP/Symfony

SOMMAIRE

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

Version MVP(Minimum Viable Product) :

L'application a été réalisée avec la version 3.1 du Framework Symfony. Cette version n'est plus maintenu par SensioLabs depuis juillet 2017. Vous pouvez voir sur leur site l'état de maintenance du Framework Symfony https://symfony.com/roadmap.

Version Améliorée:

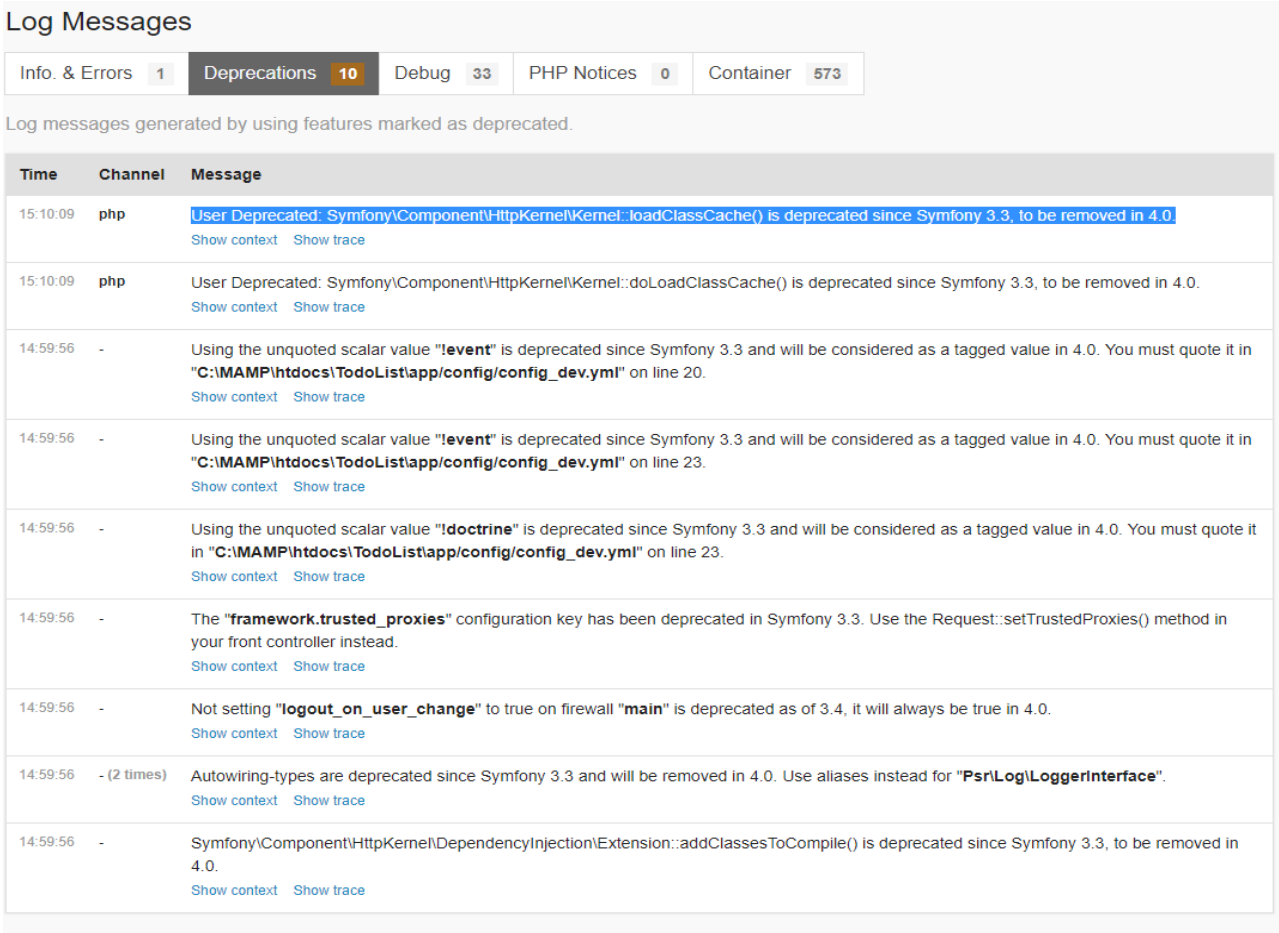
L'objectif est donc de mettre Symfony à jour vers la version 4.1. C'est la dernière version stable. Elle sera maintenue jusqu'à la fin du mois de juillet 2019. Il est recommandé par Symfony d'utiliser cette version pour les nouveaux projets. Comme votre souhait est de remettre à niveau votre application afin qu'elle puisse évoluer et satisfaire vos nouveaux clients. Nous avons fait le choix de la version 4.1 qui offre de meilleurs performances avec PHP 7.1 (Comparatif ici) et des évolutions possibles avec Symfony Flex.

Tout d'abord, nous vérifions 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.

La mise à jour se fait en deux étapes :

Nous mettons à jour le Framework vers la version stable 3.4 LTS(Long Term Support) . Nous vérifions avant 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.

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. Vous pouvez voir ci-dessous la liste des dépréciations corrigés.



Pour finir, dans le fichier composer.json nous pouvons rentrer `"symfony/symfony": "^4.1",` et lancer la commande `php composer.phar update symfony/symfony`. Votre framework Symfony va se mettre à jour.

Important : L'architecture de Symfony 3.4 est conservée, il est possible de migrer vers Symfony Flex dans le futur.

ETAT DES LIEUX ET CORRECTIONS

Dans cette partie, nous allons vous présenter un état des lieux de la version MVP après utilisation de l'application et analyse du code. Et dans un second temps, nous proposons un tableau comparatif des améliorations apportées à la dette technique de la version MVP.

ETAT DES LIEUX DE LA VERSION MVP

- Version 3.1 qui n'est plus maintenue depuis juillet 2017.
- Nous pouvons ajouter une tâche mais elle n'est rattachée à aucun utilisateur.
- Un utilisateur peut supprimer ou éditer une tâche qu'il n'a pas créé lui même.
- Lors de la création d'un utilisateur, le formulaire ne propose pas le choix d'un rôle.
- Tout utilisateur peut se connecter, accéder à la gestion des tâches et également à la gestion des utilisateurs.
- Il n'y a pas la possibilité de supprimer un utilisateur.
- Nous pouvons accéder à la liste des utilisateurs sans être identifiés.
- Absence de sécurité.
- Les fonctionnalités ne sont pas toutes accessibles par un bouton. Il est parfois nécessaire de taper l'url exact.
- La fonctionnalité de marquer une tâche comme faite ne marche pas correctement.
- Certaines contraintes de formulaires sont manquantes pour la création d'un utilisateur.
- Au niveau du code, il n'y a aucune couverture(ni tests fonctionnels, ni tests unitaires).
- Aucune fixtures fournies avec l'application.
- Duplications de code présent dans les controllers.
- Trop de logique métier présent dans les controllers.
- Le code n'est pas optimisé selon les principes solides.
- Présence de codes dépréciés ou inutiles (utilisation de l'outil Codacy pour analyser le code).
- Le code n'est pas documenté.
- Aucune documentation présente sur l'application (absence de diagrammes, fichier Readme.md non détaillé).

CORRECTIONS APPORTÉES AVEC LA VERSION AMÉLIORÉE :

	Dette technique de la version MVP	Correction apportée dans la version améliorée
Version	Version 3.1 du Framework Symfony.	Mise à jour vers la version 4.1 expliquée en détail dans la première partie.
Authentification	Les tâches ne sont pas liées aux utilisateurs.	Les tâches crées 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 `php bin/console demo:load anonymous`.
	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.
Tests	Aucune Couverture	Implementer tests avec PhpUnit Bridge. Couverture à 95% Utilise commande `vendor/bin/simple-phpunit –coverage-html cov\` 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.

Refactoring	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 crées.
	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 boutton supprimer dans view User list.html.twig. Création de tests fonctionnels pour cette action.
	Le code n'est pas documenté dans les entités.	Chaques 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 existente 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.
Documentation	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 POUR UN CODE DE QUALITE

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.

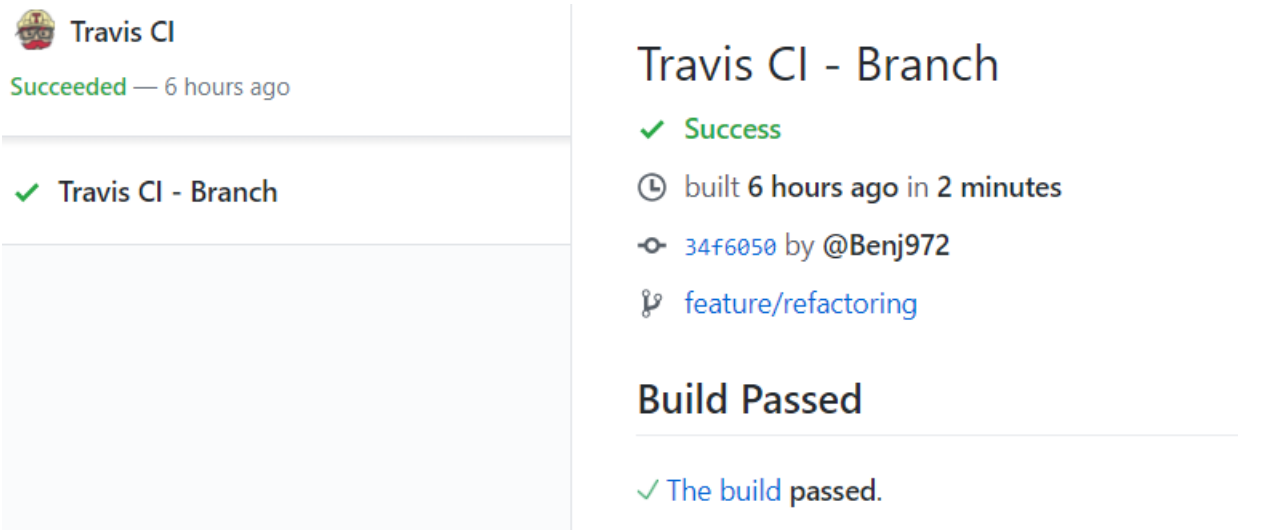
TESTS

Tout d'abord , il est un 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		97.19%	173 / 178		93.22%	55 / 59		86.67%	13 / 15
Command		100.00%	31 / 31		100.00%	5 / 5		100.00%	1 / 1
Controller		100.00%	50 / 50		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		100.00%	13 / 13		100.00%	4 / 4		100.00%	2 / 2
Security		90.00%	18 / 20		60.00%	3 / 5		0.00%	0 / 1
Service		100.00%	8 / 8		100.00%	3 / 3		100.00%	2 / 2
AppBundle.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

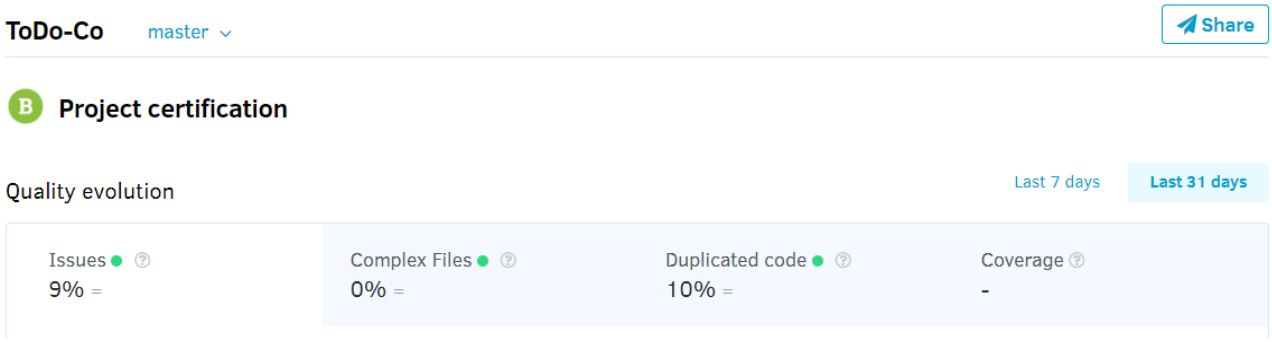
Lorsque que le taux de couverture est réalisé. Il faut ensuite utiliser l'outil Travis CI. C'est un outil d'intégration continue qui exécute automatiquement l'intégralité de nos tests unitaires et fonctionnels à chaque Pull Request.



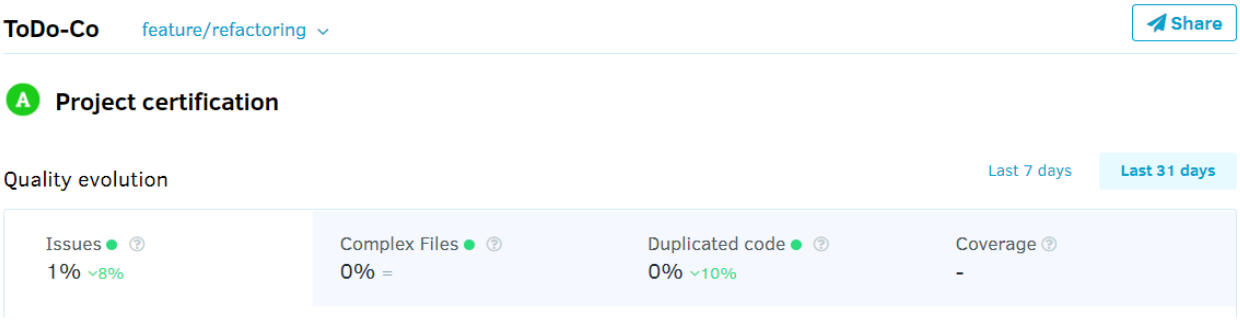
EVALUER LE CODE

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 MVP et ensuite à chaque création de nouvelles branches pour la version améliorée. Voici un exemple :

Analyse de la Version MVP :

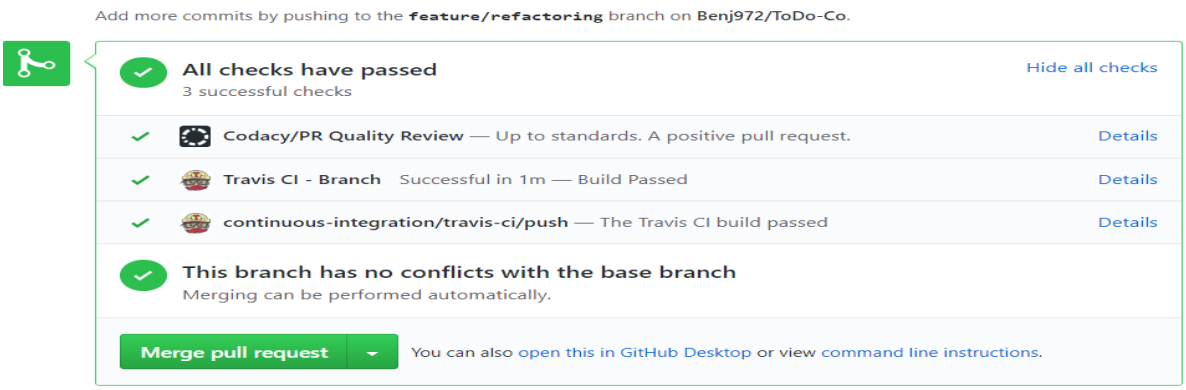


Analyse de la branche refactoring pour la version améliorée :



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 (analysé par Travis.ci et Codacy)



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 inférieur 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 équivalent. 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. SensioLabs préconise d'utiliser certains outils afin d'optimiser au mieux son application. https://symfony.com/doc/current/performance.html

Nous allons évoquer quelque points que nous avons retenu pour l'application ToDoList.

Configuration de PHP

Il est recommandé d'utiliser PHP7 ou supérieur qui est plus rapide que les versions antérieurs. Il faut configurer le fichier php.ini afin de tirer le meilleur de PHP.
Voici quelques propriétés importantes :

Désactiver l'utilisation des balises courtes «<? ?>».On doit ouvrir en écrivant explicitement «<?php»
short_open_tag = Off

temps maximum d'exécution après lequel un script sera arrêté
max_execution_time = 30

temps maximum qu'un script peut mettre pour recevoir des données input
max_input_time = 60

taille de mémoire qu'un script peut consommer
memory_limit = 512M

Il y a de nombreuses autres propriétés qui peuvent être configurées en fonction des besoins de l'application. Il faut retenir que PHP7 et Symfony4 utilisés ensemble renvoie le meilleur taux de performance (rapport de comparaison ici réalisé par PHP Benchmarks).

APCu

APCu est un cache de données qui permet de gérer le cache utilisateur. C'est une extension PHP qui reprend le code d'APC qui est moins utilisé au profit d'OPCache mais ce dernier ne gère pas le cache utilisateur d'où l'intérêt d'APCu.

Dans le fichier php.ini, il faut donc l'activer :

```
extension=php_apcu.dll
apc.enabled=1
apc.shm_size=32M
apc.ttl=7200
apc.enable_cli=1
apc.serializer=php
```

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*).
Configuration dans le php.ini :

Mémoire maximale que OPcache peut utiliser pour stocker des fichiers PHP compilés
opcache.memory_consumption=256
nombre maximal de fichiers pouvant être stockés dans le cache
opcache.max_accelerated_files=20000
désactiver la revalidation du script
opcache.validate_timestamps=0

PHP realpath cache

Lorsqu'un chemin relatif est transformé en son chemin réel et absolu, PHP met en cache le résultat pour améliorer les performances.
Configuration dans le php.ini :
Mémoire maximale allouée pour stocker les résultats
realpath_cache_size = 4096K
enregistrer les résultats pendant 10 minutes (600 secondes)
realpath_cache_ttl = 600

Autoloader

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 crée, 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%.

Rapport de performance avec optimisation(APCu, Opcache, PHPrealpath, Autoloader)

URL	TEMPS	MEMOIRE
/login	36.6ms	1.09MB
/	53.3ms	1.61MB
/tasks	58.7ms	1.71MB
/tasks/create	85.6ms	2.25MB
/tasks/{id}/edit	75.6ms	1.79MB
/users	75.1ms	1.66MB
/users/create	93.1ms	2.4MB
/users/{id}/edit	90.2ms	2.41MB

Nous constatons une nette amélioration du temps de réponse et de la consommation de mémoire. Vous devez donc configurer au mieux votre environnement en suivant les instructions de SensioLabs afin d'optimiser au mieux les performances de votre application lorsqu'elle sera déployée.

Pour finir, il y a également d'autres pistes intéressantes pour optimiser l'application en fonction de son évolution dans le futur. En voici quelques unes :

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.

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. Il également est important de suivre les recommandations d'optimisations pour la mise en production de votre application. A cela vous rajouter un code de qualité et votre application sera performante. Des points d'améliorations sont encore possibles comme par exemple un travail sur l'ergonomie du site avec le Framework d'interface Bootstrap.