ToDo & Co – TodoList

Audit de qualité du code & performance

Table des matières

1. Contexte 3
1.1 Présentation 3
1.2 Urgence de version 3
1.3 Note importante 3
2. Qualité du code 4
2.1 Analyses de code 4
2.1.1 Rapport Codacy 4
2. Performances 6
2.1 Analyse symfony/profiler 6
2.2 Optimisation 6
2.2.1 Knp_Pagantor_bundle 6
2.2.2 mise à jour de symfony. 7
2.3 Bilan 8
2.4 Pistes d'axes d'améliorations 9
2.4.1 OPCache 9
2.4.2 RealpathCache 9
2.4.3 Varnish 9
2.4.4 Hébergeur 9

1. Contexte

1.1 Présentation

L'application TodoList appartient à la jeune startup ToDO & Co. L'application a dû être développée à toute vitesse pour présenter le concept à de potentiels investisseurs. À la suite de la présentation, l'entreprise a réussi à lever des fonds pour le développement de l'application et son expansion.

1.2 Urgence de version

L'application a été mise à jour vers une version plus ressente par souci de maintenance, mais aussi de sécurité. Il faut savoir que la version 3.1 de symfony n'est plus maintenue, il était donc normal de faire une mise à jour avec urgence. La version 6.3 a été choisie pour une plus longue période de maintenance (https://symfony.com/releases/6.3).

Application avant modification

Symfony 3.1.10
php en 5.5.9
doctrine-bundle 1.6
doctrine-orm 2.5
database MySql

Application après modification

Symfony 6.3
php en 8.1
doctrine-bundle 2.10
doctrine-orm 2.16
knp-paginator-bundle 6.2

1.3 Note importante

Tous les tests et informations fournis dans ce document sont basés après le changement de versions de symfony.

2. Qualité du code

2.1 Analyses de code

Pour faire l'analyse du site je me suis appuyée sur Codacy, pour les erreurs de code et les bonnes pratiques.

2.1.1 Rapport Codacy

Selon Codacy, les statistiques de complexité et de duplication de code sont à 0 %, le nombre d'issues est de 61 %. La note globale du site est « b », ce qui est une note moyenne obtenable. Ce qui, dans la globalité, présente une bonne qualité de code. À savoir : le grade est

calculé sur le nombre d'issues pour 1k lignes de code.

Les différentes issues sont répertoriées par catégories, mais aussi par niveau, elles représentent les problèmes de code venant du site.

Info : Le type de problème le moins critique apparaîtra en bleu ; par exemple, les problèmes de style de code.

Avertissement : ce type de problème apparaîtra en jaune. Vous devez être prudent avec ceux-ci, ils sont basés sur les normes et les conventions du code.

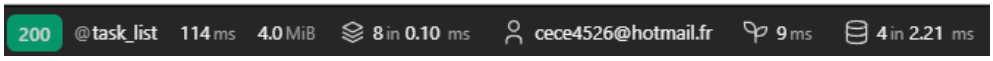
Erreur : les types de problèmes les plus dangereux s'affichent en rouge. Prenez le temps de les corriger, bien que le code puisse s'exécuter, ces problèmes montrent que le code est très susceptible de poser des problèmes. Ces problèmes sont sujets aux bogues et / ou peuvent avoir de graves problèmes de sécurité et de compatibilité.

Une configuration des exclusions de fichiers a été faite pour ne pas prendre en comptes les lignes de code généré par symfony et les différentes librairies utilisées.

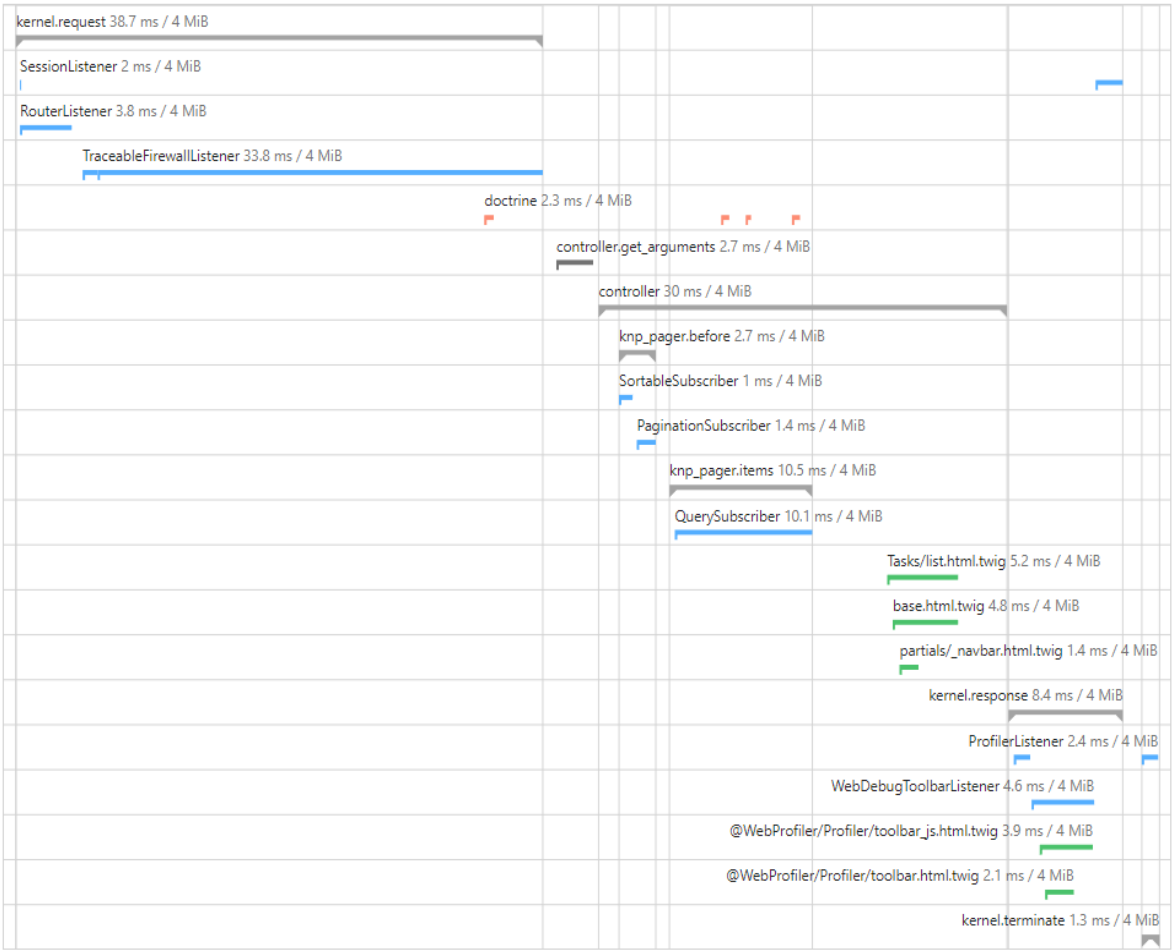
Dashboard codacy du projet : https://app.codacy.com/gh/cece4526/todolist/dashboard

2. Performances

2.1 Analyse symfony/profiler



L'analyse de symfony/profiler donne : le temps de chargement du site, la mémoire utiliser pour son chargement, les fonctions utilisées et son nombre de requêtes. Chaque fonction est détaillée avec son temps d'exécution et la mémoire utilisée. Un schéma est aussi consultable pour suivre le plan d'exécution du site, avec une couleur spécifique pour les fonctions les plus gourmande.

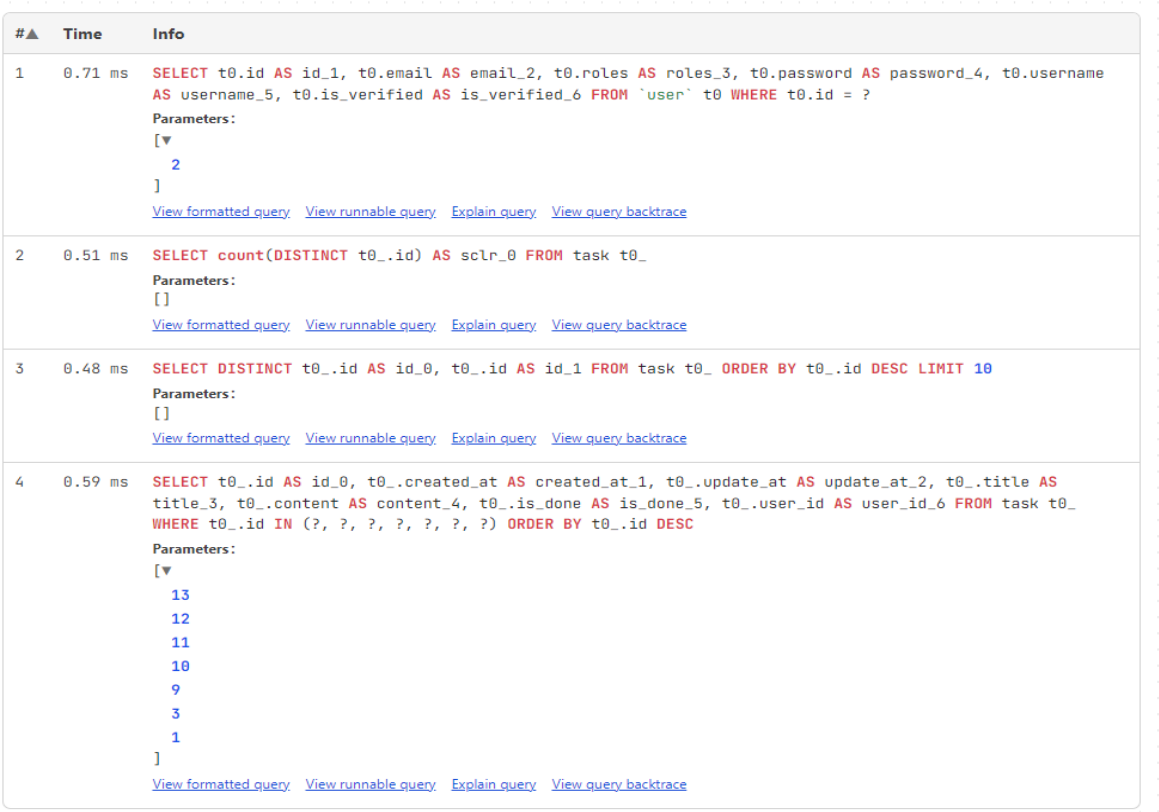


Lors de l'analyse j'ai pu constater de bon résultat pour l'affichage des taches au global.

2.2 Optimisation

2.2.1 Knp_Pagantor_bundle

L'utilisation du bundle knp-paginator-bundle permet l'optimisation de données reçu dans la requête suivant le nombre limite que l'on souhaite afficher (là ces 6 taches) ce qui permet d'optimiser la demande a la base de donné.



#▲	Time	Info
1	0.71 ms	`SELECT t0.id AS id_1, t0.email AS email_2, t0.roles AS roles_3, t0.password AS password_4, t0.username AS username_5, t0.is_verified AS is_verified_6 FROM user t0 WHERE t0.id = ?` **Parameters:** [▼ 2] View formatted query View runnable query Explain query View query backtrace
2	0.51 ms	`SELECT count(DISTINCT t0_.id) AS sclr_0 FROM task t0_` **Parameters:** [] View formatted query View runnable query Explain query View query backtrace
3	0.48 ms	`SELECT DISTINCT t0_.id AS id_0, t0_.id AS id_1 FROM task t0_ ORDER BY t0_.id DESC LIMIT 10` **Parameters:** [] View formatted query View runnable query Explain query View query backtrace
4	0.59 ms	`SELECT t0_.id AS id_0, t0_.created_at AS created_at_1, t0_.update_at AS update_at_2, t0_.title AS title_3, t0_.content AS content_4, t0_.is_done AS is_done_5, t0_.user_id AS user_id_6 FROM task t0_ WHERE t0_.id IN (?, ?, ?, ?, ?, ?, ?) ORDER BY t0_.id DESC` **Parameters:** [▼ 13 12 11 10 9 3 1] View formatted query View runnable query Explain query View query backtrace

2.2.2 mise à jour de symfony.

Le fait de passer a une version plus récente de symfony permet de bénéficier des optimisation, de la suppression d'éventuelles failles de sécurité et l'ajout de nouvel fonctionnalités.

2.3 Bilan

J’ai préféré partir sur une base nouvelle de symfony 6.3 pour le projet en y calçant les fonctionnalités déjà présentes dans le projet initial en symfony 3.1, tout en les optimisant avec les nouvel fonction et bundle disponible dans symfony.

J’ai aussi ajouté des bundles comme le mailer de symfony et le vérifie email bundle de symfonycasts pour une vérification de l’email des utilisateurs a l’inscription.

Certains bundles disponibles dans la version 3.1 comme ceux de sensio qui sont déprécier dans les versions 4 de symfony et remplacer par les mises à jour des bundles doctrine et celui du maker de doctrine.

Je pense que mon choix de partir sur une nouvelle base de symfony et d’y implantés petit a petit les fonctions de l’ancienne application ma permis d’éviter pas mal de bug et de perte de temps sur ce projet car mon environnement local est déjà configuré pour ces versions-là.

2.4 Pistes d'axes d'améliorations

Différentes optimisations simples à faire peuvent être mis en place, comme l'activation d'OPCache, la modification de realpathCache, l'utilisation de varnish. Mais bien entendu, l'optimisation des performances passe par une configuration du serveur optimale.

2.4.1 OPCache

« OPcache améliore les performances de PHP en stockant le bytecode des scripts pré-compilés en mémoire partagée, faisant ainsi qu'il n'est plus nécessaire à PHP de charger et d'analyser les scripts à chaque demande. » (src :https://fr.docs.wp-rocket.me/article/675-quest-ce-que-opcache)

Liens pour optimisation par OPCache : https://www.ekino.fr/articles/php-comment-configurer-utiliser-et-surveiller-opcache

2.4.2 RealpathCache

« RealpathCache, lorsqu'un chemin relatif est transformé en son chemin réel et absolu, PHP met en cache le résultat pour améliorer les performances. Les applications qui ouvrent de nombreux fichiers PHP, tels que les projets symfony sont beaucoup impactés par ces options-là »

Liens pour optimisation avec RealpathCache: https://symfony.com/doc/current/performance.html

2.4.3 Varnish

« Varnish est un accélérateur HTTP open source puissant, capable de servir rapidement du contenu mis en cache et de prendre en charge Edge Side Include. Étant donné que le cache de Symfony utilise les en-têtes de cache HTTP standard, le proxy inverse Symfony peut être remplacé par tout autre proxy inverse. »

Liens pour Varnish : https://symfony.com/doc/current/http_cache/varnish.html

2.4.4 Hébergeur

Le choix de l'hébergeur est crucial pour proposer un site réactif et optimal, autant sur la connexion que sur la configuration.

Lien explicatif des différents choix possible : https://www.codeur.com/blog/comment-choisir-un-hebergeur-web/