Audit de code & performance

Avant propos

Cet audit de code se base sur l'application ToDoList de l'entreprise ToDo & Co le but étant d'évaluer la dette technique de l'application, puis d'évaluer la qualité du code et de performance de l'application après modification des anomalies et améliorations effectués.

Qualité du code

Pour évaluer la dette technique de l'application, je me suis appuyé sur plusieurs outils tels que Scrutinizer, Codacy, PHPUnit et mon expérience personnelle.

En terme de structure d'application, ToDoList est dans les standards de Symfony et il n'y a pas de problème à notifier de ce côté.
Cependant, il y a quelques point d'améliorations à prendre en compte tels que :

- La version de Symfony 3.1 utilisée n'est plus maintenue depuis Juillet 2017. Cela risque d'entraîner des dépréciations de méthodes et un risque de maintenabilité dans le temps.
  https://symfony.com/roadmap?version=3.1#checker

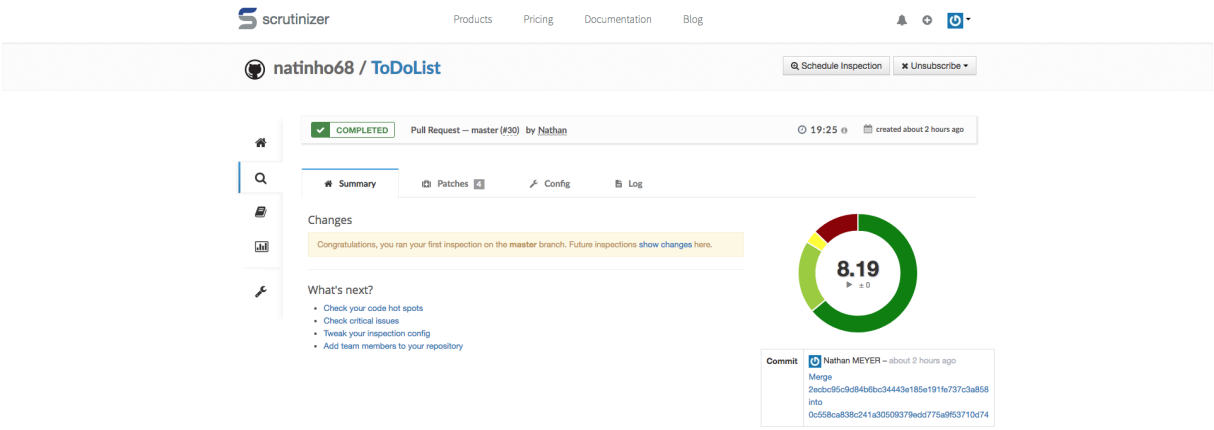
- La validation des formulaires se fait directement via $form->isValid(). Cette méthode fonctionnera mais va lever un Warning. Il faut également ajouter la méthode $form->isSubmitted()
  https://symfony.com/doc/3.4/form/without_class.html

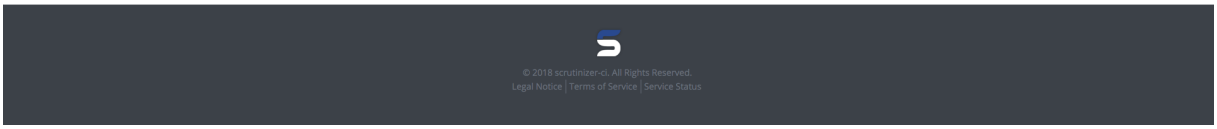
- De part mon expérience sur Symfony, il y a des critères de sécurité qu'il faut vérifier et qui peuvent être risqués si on me modifie pas ces violations. C'est le cas pour cette application, je parle notamment de :
  - L'ajout d'une description dans le composer.json
  - De changer le token secret qui peut entrainer des failles de sécurité
  - Le nom des sessions cookies devrait être renommées

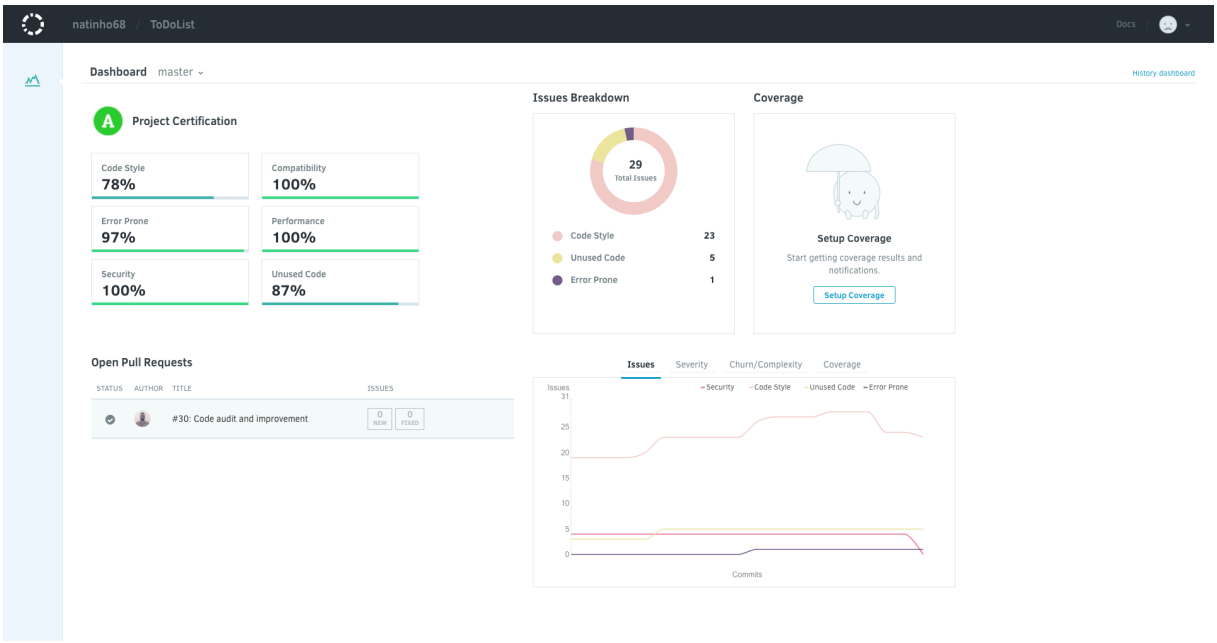
- On peut aussi noter qu'il faudrait avoir les différentes librairies utilisées dans l'application, à jour, au risque d'être face à des problèmes de sécurités et/ou de dépréciations, conflits…

Un des points les plus important à noter sur la dette technique de l'application initiale est la non utilisation des tests unitaires et fonctionnels. Le taux de couverture est de 0%. Il faudrait être en mesure de pouvoir tester les points d'entrées et de sorties et d'ajouter un outil d'intégration continue tel que Travis ou Jenkins.
Cela permettrait, une fois les test implémentés, de vérifier très rapidement si à chaque commit, les tests sont bien passés.

De manière générale, la qualité de code de l'application est plutôt bonne comme le montre les résultats de Scrutinizer et Codacy







Performance

Il est demandé d'établir un rapport de performance via BlackFire une fois que les modifications ont été apportées.
Selon plusieurs études, la fidélisation des utilisateurs d'une application web dépend en grande partie de la qualité et de la performance de l'application. Sur ce point, l'expérience utilisateur est très importante et on peut noter 3 axes à surveiller particulièrement :

- La rapidité d'affichage des pages d'un site web (40% des visiteurs abandonnent la navigation si le temps de réponse dépasse 3 secondes)
- Le temps nécessaire au téléchargement des éléments d'une page
- La fluidité du navigateur dans la manipulation des éléments d'une page

Il est également bon de rappeler qu'il ne s'agit pas là des seuls éléments de performances sur lesquels il faut se pencher. La performance d'une application web dépend effectivement du temps de réponses, mais également de la disponibilité du système, de la robustesse de celui-ci, et de sa capacité à monter en charge.

Certaines de ces données ne sont pas testables compte tenu de l'environnement local sur lequel les tests ont été effectués.

Toute fois, voici le tableau des résultats obtenus sur les requêtages d'URL, qu'il faudra peut être comparé sur un environnement de production :

URL	**Temps**	**Mémoire**
/tasks/edit	309 ms	13.8 MB
/tasks/create	300 ms	13.8 MB
/tasks	248 ms	11.1 MB
/users/edit	318 ms	14.1 MB
/users/create	317 ms	14 MB
/	222 ms	10.8 MB
/login	159 ms	7.17 MB

Compte tenu de ce qui a été dit précédemment, ces temps d'exécution et la mémoire allouée à ceux-ci sont tout à fait correct.

Même si ces résultats sont bons, nous pouvons les améliorer assez rapidement en utilisant, comme le suggère BlackFire :

- Le système de cache de Symfony
- Le système de cache de Doctrine

Il y a une partie sur laquelle la performance de notre application a une influence, ce sont les éléments du front. Des images lourdes, leurs traitements peuvent faire augmenter le temps de chargement de votre page et donc faire diminuer les performances de votre application. Le cache y remédierait, un système de LazyLoading également.
Etant donné que l'application est vouée à évoluer, quantifier tous les éléments du front qui pourraient être amélioré serait fastidieux.

Cependant on peut noter quelques améliorations qui pourrait permettre à l'application de se charger plus rapidement comme :

- L'utilisation d'un CDN pour charger Boostrap et jQuery
- Minifier les fichiers JS et CSS et les regrouper en 1 seul fichier
- Utiliser un système de lazy loading pour les images

Pour conclure

La version de Symfony a été upgradé vers la version 3.4 qui est la version stable du moment.
Pour le reste des améliorations à mettre en place, ce sont des pratiques qui peuvent être facilement établies et qui concerne principalement la qualité du code de l'application.
Pour la partie performance, la mise en place d'un cache permettrait de gagner encore un peu de temps sur

l'exécution des requêtes. Le point le plus bloquant sur cette application était la non utilisation de tests. Cela a été corrigé, en même temps que les demandes proposées dans le cahier des charges pour passer d'une couverture de 0% à 100%. Un outil d'intégration continue a également été mis en place (Travis).