Audit de code & performance

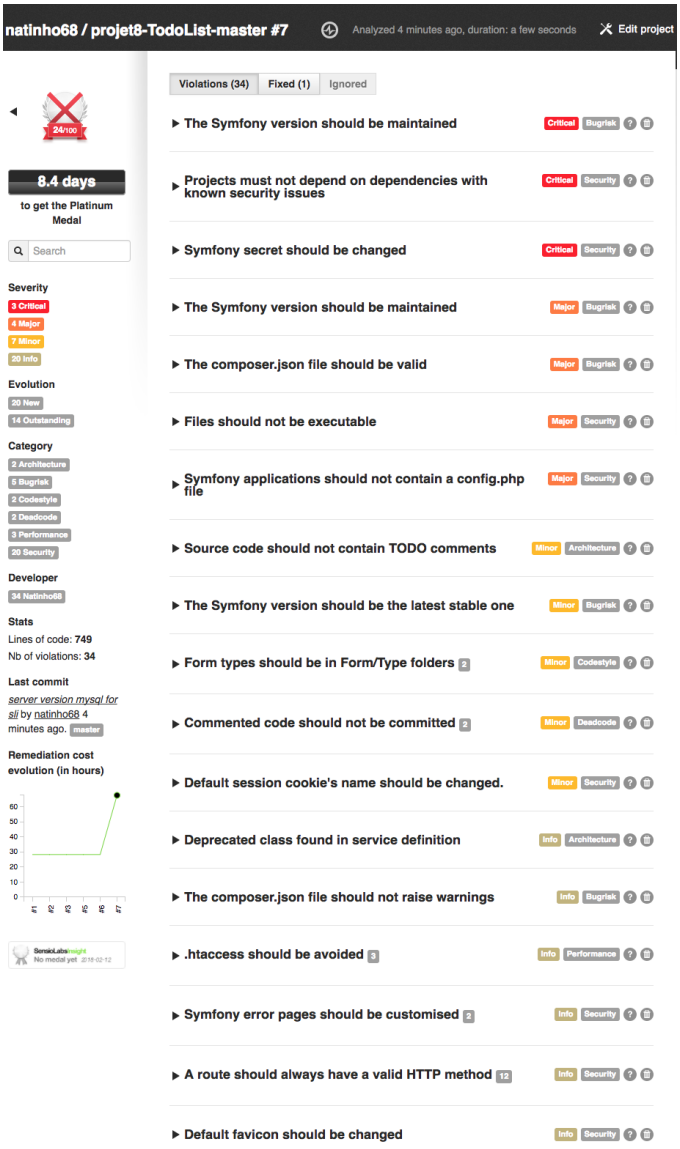
Avant propos

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.

I) QUALITÉ DU CODE ET DETTE TECHNIQUE

Pour évaluer la dette technique de l'application, je me suis appuyé sur plusieurs outils tels que Scrutinizer, Codacy, PHPUnit, SensioLabsInsight 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 points d'amélioration à prendre en compte tels que le démontre le rapport de SensioLabsInsight:



1) *Rapport SensioLabsInsight*

Le rapport ci dessus rapporte 19 points de violations à améliorer, en indiquant la gravité de celles-ci.

Les différents niveaux d'alertes sont :

a) Critical : Point très urgent, qui peut mettre en danger le bon fonctionnement de l'application et sa sécurité

Dans cette catégorie de violation nous avons 3 violations.

The Symfony version should be maintained : La version de Symfony 3.1 utilisée n'est plus maintenue depuis Aoüt 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

Projects must not depend on dependencies with known security issues : Cette erreur dépend de celle précédemment citée en rapport à la version de Symfony. Une mise à jour de la version de SF devrait supprimer cette alerte.

Symfony secret should be changed : La clé secrète est celle donnée par défaut. Pour supprimer cette violation il suffit de changer le token secret. Cette erreur peut entrainer des failles de sécurité.

b) Major : Point urgent, qui peut mettre en danger le bon fonctionnement de l'application et sa sécurité à moindre mesure que les violations « Critical »

Dans cette catégorie de violation nous avons également 3 violations.

The Symfony version should be maintained : Même point que pour l'alerte « Critical »

The composer.json file should be valid : Cette erreur indique que le fichier composer.json, devrait contenir un champ description

Files should not be executable : Certains fichiers ne devraient pas avoir la possibilité de s'exécuter. Il faut restreindre ces droits pour éviter des failles de sécurité

Symfony applications should not contain a config.php file : Le fichier config.php permet à votre projet d'être initialisé. Une fois le projet en production, il faut impérativement supprimer ce fichier au risque d'être face à une importante faille de sécurité.

c) Minor : Point mineur, portant sur le style du code, le code mort, le code inutilisé et la sécurité

Dans cette catégorie de violation nous avons 5 violations.

Source code should not contain TODO comments : Des portions de code contiennent des mentions TODO. Sur une application en production, il vaut mieux supprimer ce genre d'indications.

The Symfony version should be the latest stable one : Comme indiqué ici, non seulement la version de Symfony est dépréciée, mais celle-ci n'est même pas sur la version la plus stable.

Form types should be in Form/Type folders: Cette alerte est levée car les formulaires sont directement inclus dans le dossier /Form. Or les bonnes pratiques de Symfony recommande d'avoir une architecture pour les formulaires de cette manière, /Form/Type/LesFormulaires.

Commented code should not be committed : Une alerte levée car il y a du code commenté sur le projet.

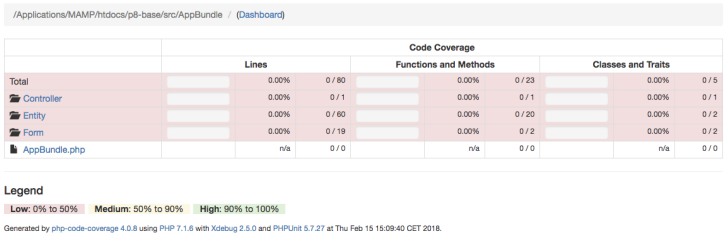
Default session cookie's name should be changed : Le nom des sessions cookies est celui par défaut, il faudrait le modifier pour contrer une éventuelle faille de sécurité.

d) Les infos

Les infos sont des alertes indicatives. Il est cependant bon d'essayer de les supprimer, des infos sur une version de SF peuvent très bien se transformer en point Critical au fur et à mesure de l'évolution du Framework ou des librairies.

2) Les tests

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.



/Applications/MAMP/htdocs/p8-base/src/AppBundle / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		0.00%	0 / 80		0.00%	0 / 23		0.00%	0 / 5
Controller		0.00%	0 / 1		0.00%	0 / 1		0.00%	0 / 1
Entity		0.00%	0 / 60		0.00%	0 / 20		0.00%	0 / 2
Form		0.00%	0 / 19		0.00%	0 / 2		0.00%	0 / 2
AppBundle.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

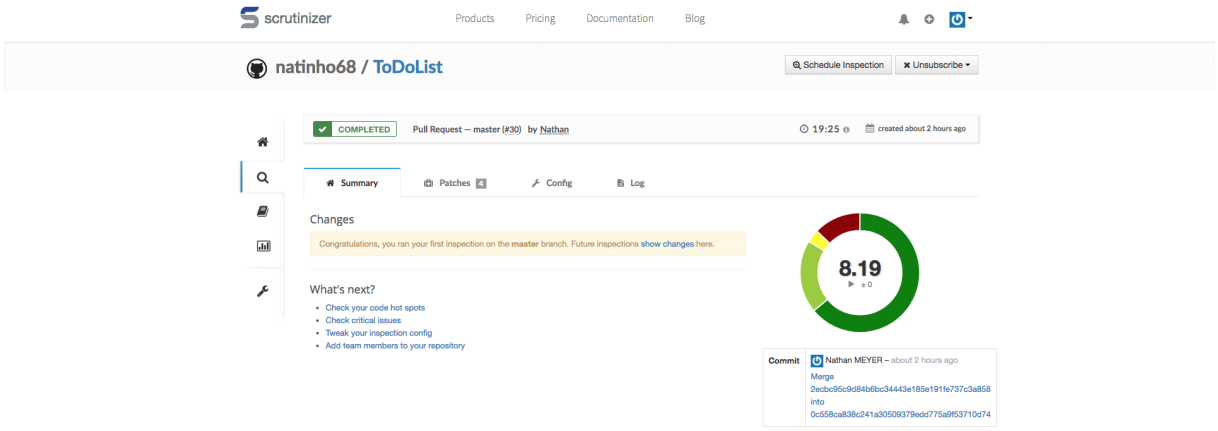
Legend

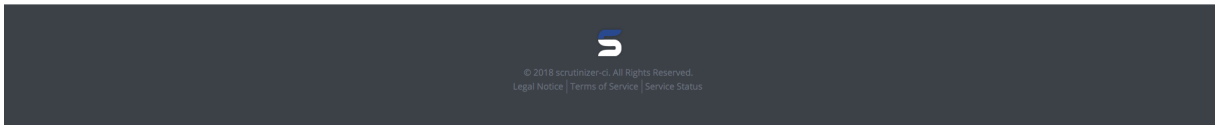
Low: 0% to 50% Medium: 50% to 90% High: 90% to 100%

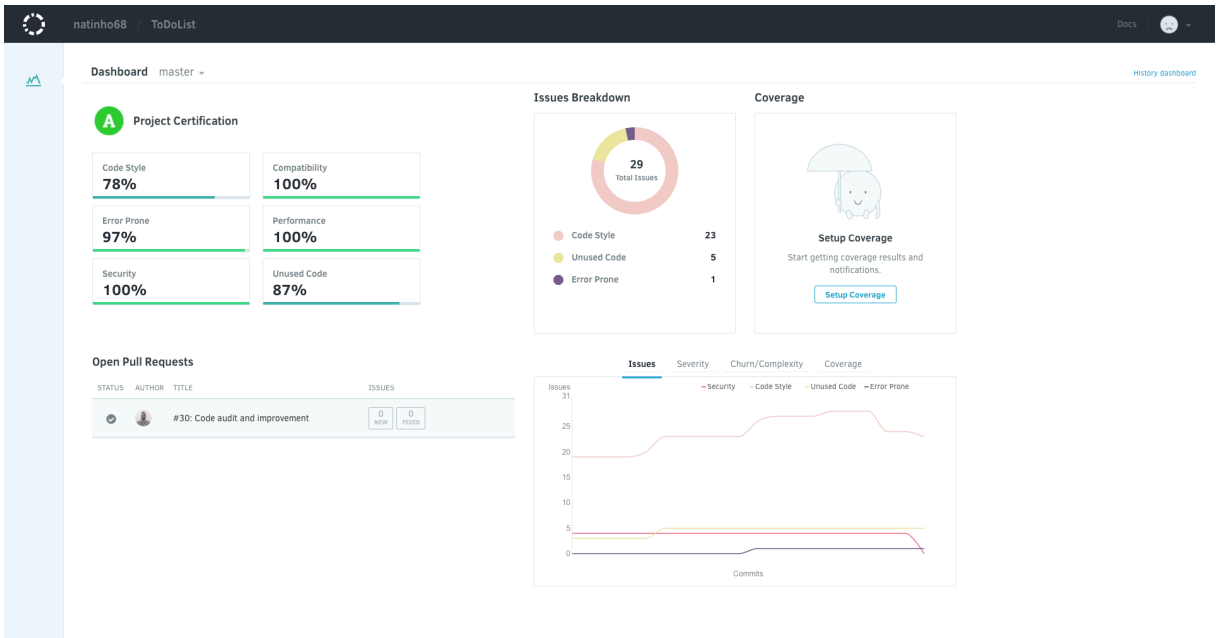
Generated by php-code-coverage 4.0.8 using PHP 7.1.6 with Xdebug 2.5.0 and PHPUnit 5.7.27 at Thu Feb 15 15:09:40 CET 2018.

3) Les autres outils

Sur un axe basé sur PHP et non Symfony en particulier, la qualité de code de l'application est plutôt bonne comme le montre les résultats de Scrutinizer et Codacy







II) 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 trois 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 réponse, mais également de la disponibilité du système, de la robustesse de celle-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.

Toutefois, voici le tableau des résultats obtenus sur les requêtages d'URL, qu'il faudra peut être comparer 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, ainsi qu'un système de LazyLoading.
L'application étant vouée à évoluer, quantifier tous les éléments du front qui pourraient être améliorés serait fastidieux.

Cependant on peut noter quelques améliorations qui pourraient 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

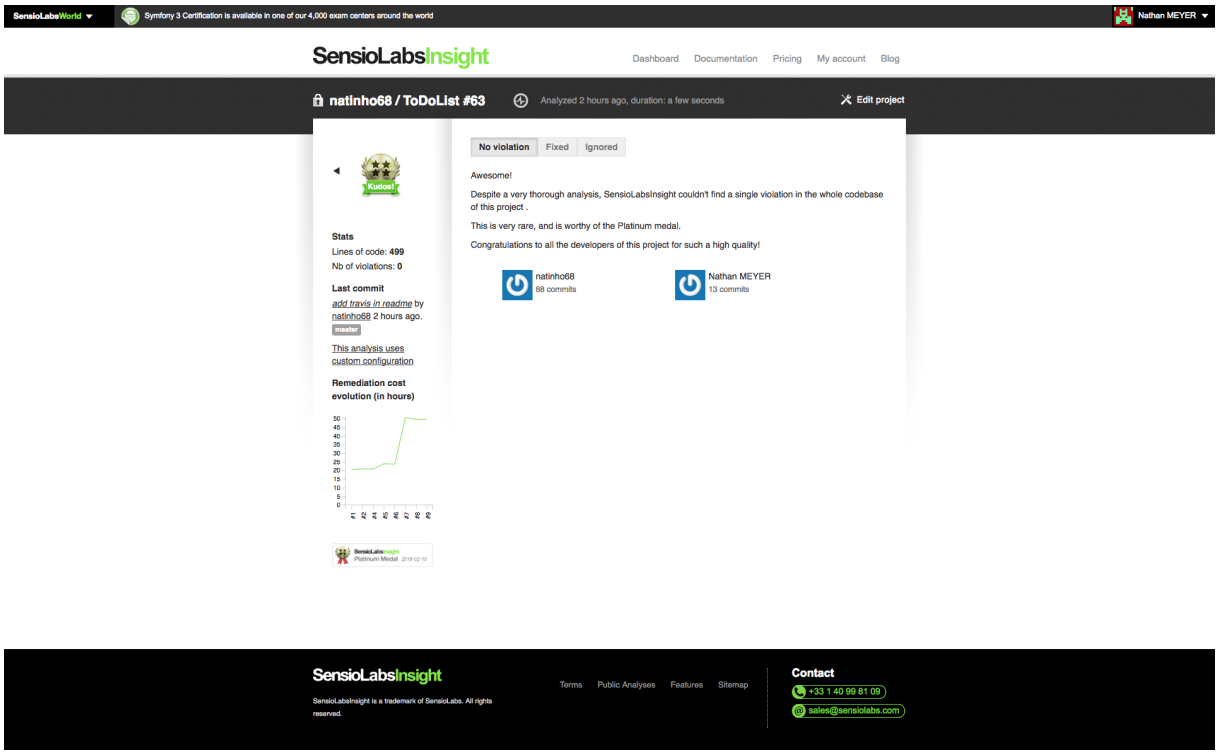
III) QUALITÉ ET PERFORMANCE APRÈS MODIFICATIONS

1) Qualité de code

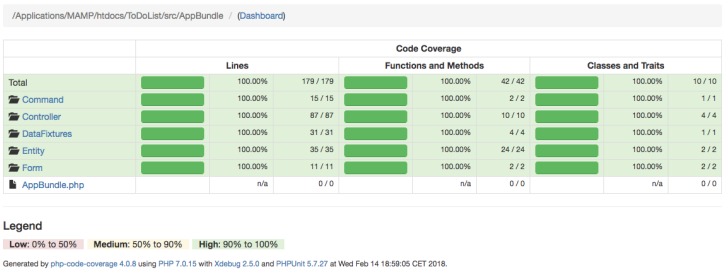
La qualité du code a été revue, ces points-ci ont été corrigés :

- La version de Symfony a été upgradé vers la version 3.4 qui est la version stable du moment.
- Les modifications demandées dans le cahier des charges ont bien été faites
- Toutes les violations du rapport de SensioLabsInsight ont été corrigées
- Les tests ont été créés et le code coverage est désormais à 100%
- Un outil d'intégration continu pour les tests et le déploiement a été mis en place, Travis

Rapport SensioLabsInsight



Code coverage de PHPUnit



2) Performance

Pour la partie performance, ces modifications ont été apportées:

- La mise en place du cache http de Symfony (à activer en production)
- Appel aux librairies JS et CSS via CDN

Grâce à BlackFire, nous pouvons comparer les métriques initiales avec celle de l'application finale avec le cache. Grâce à ces modifications, l'application est quasiment 12 fois plus rapide et requiert presque 10 fois moins de mémoire.

URL	**Temps**	**Mémoire**
/tasks/edit	24 ms	1.5 MB
/tasks/create	24 ms	1.5 MB
/tasks	24.2 ms	1.51 MB
/users/edit	24.6 ms	1.5 MB
/users/create	24.3 ms	1.5 MB
/	24.1 ms	1.5 MB
/login	24.2 ms	1.5 MB