Audit

Réalisé par Pierre Gaimard
Le 31/03/2021.

Introduction

Première partie

La première partie présente un état des lieux initial de l'application afin d'en évaluer la dette technique.

Seconde partie

La seconde présente les actions menées afin d'améliorer la qualité de l'application en s'appuyant sur cet état des lieux.

Troisième partie

La troisième partie présente une analyse réalisée au terme du développement.
Elle compare la version actuelle à la version initiale du projet sur deux axes: la qualité du code et la performance. Elle servira également de base de comparaison pour les évolutions futures du projet.

Table des matières

Etat des lieux initial de l'application **3**
- Installation, librairies et dépendances. 3
  - Symfony 3
  - Sécurité 4
  - Contrôleurs 4
  - Tests unitaires et fonctionnels 4
  - Front-End 4
  - Conclusion 4
- Audit de qualité du code 4
  - Issues 4
  - Dupplications 4
  - Conclusion 5
- Test fonctionnel de l'application 5
  - Sécurité 5
  - Navigation 5
  - Utilisateurs 5
  - Tâches 5
  - Conclusion 5

Actions menées **6**
- Mises à jour 6
  - Symfony et de ses dépendances 6
  - FrontEnd 6
- Qualité 6
- Correction les bugs fonctionnels relevés 6
- Anomalies et nouvelles fonctionnalités 7
  - Tâches & utilisateurs 7
  - Utilisateur 7
  - Autorisations 7
- Améliorations 7
  - Sécurité 7
  - Navigation 7
- Tests unitaires et fonctionnels 7
- Environnement de développement 7
  - Qualité du code 7
  - Intégration continue 7

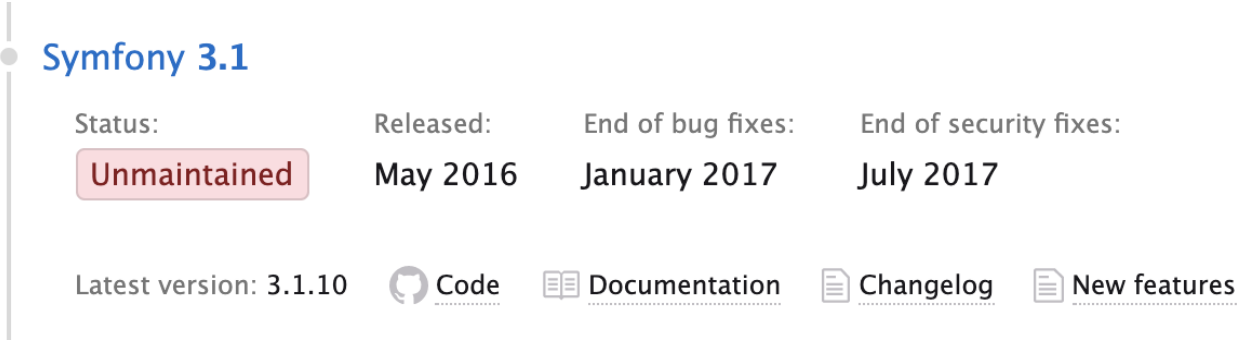
Analyse de qualité et de performance **9**
- Qualité du code 9
  - Analyse 9
  - Conclusion 10
- Performance 11
  - Code initial de l'application 11
  - Code actuel sans optimisation 11
  - Code actuel après optimisation 12
  - Comparaison entre le code initial et le code actuel. 12
  - Conclusion 12

Etat des lieux initial de l'application

Installation, librairies et dépendances.

Symfony

- L'application utilise la version 3.1 de Symfony qui n'est plus supportée.



- Les différentes dépendances sont également dans des versions anciennes.
- La librairie **symfony/swiftmailer-bundle** n'est pas utilisée dans le projet.
- Lors de l'installation des dépendances via Composer, différentes erreurs surviennent.
- Certaines corrections doivent être effectuées pour pouvoir tester l'application:
    - Mise à jour des dépendances via **composer update**
    - Nettoyage du cache
    - Remplacement de la classe **Sensio\Bundle\FrameworkExtraBundle\Configuration\Route** par la classe **Symfony\Component\Routing\Annotation\Route**

Un premier test de l'application révèle une liste **dépréciations**

Time	Channel	Message
19:13:49	php	Using the "Twig_Loader_Filesystem" class is deprecated since Twig version 2.7, use "Twig\Loader\FilesystemLoader" instead. Show stack trace
19:13:49	php	Using the "Twig_Extension_Profiler" class is deprecated since Twig version 2.7, use "Twig\Extension\ProfilerExtension" instead. Show stack trace
19:13:49	php	Using the "Twig_BaseNodeVisitor" class is deprecated since Twig version 2.7, use "Twig\NodeVisitor\AbstractNodeVisitor" instead. Show stack trace
19:13:49	php	Using the "Twig_Extension_Debug" class is deprecated since Twig version 2.7, use "Twig\Extension\DebugExtension" instead. Show stack trace
19:13:49	php	Using the "Twig_Extension_InitRuntimeInterface" class is deprecated since Twig version 2.7, use "Twig\Extension\InitRuntimeInterface" instead. Show stack trace
19:13:49	php	The Symfony\Bridge\Twig\Extension\FormExtension class implements Twig\Extension\InitRuntimeInterface that is deprecated since Twig 2.7, to be removed in 3.0 Show stack trace
19:13:49	php	Creating Doctrine\ORM\Mapping\UnderscoreNamingStrategy without making it number aware is deprecated and will be removed in Doctrine ORM 3.0. Show stack trace

Sécurité

Le système de gestion de la sécurité et de l'authentification de l'utilisateur n'est pas à jour, et ne respecte pas les best practices de la version LTS actuelle de Symfony.
L'encryptage des mots de passe "bcrypt" est trop faible.

Contrôleurs

Les contrôleurs n'implémentes pas la classe abstraite AbstractController

Tests unitaires et fonctionnels

Aucun test unitaire ou fonctionnel n'a été écrit.

Front-End

- La librairie jquery est absente

```
GET https://127.0.0.1:8002/js/jquery.js net::ERR_ABORTED 404
Uncaught Error: Bootstrap's JavaScript requires jQuery
    at bootstrap.min.js:6
```

- La librairie **respond.js** (https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js) n'est plus disponible sur le CDN.
- La version de bootstrap utilisée est très ancienne.

Conclusion

Un gros travail de mise à jour devra être effectué avant de faire évoluer l'application.

Audit de qualité du code

Une analyse **CodeClimate** du projet initial indique plusieurs choses :

Issues

2 issues **PHP Code Sniffer** liées à des lignes de code dépassant 120 caractères.

8 issues **PHP Mess Detector**: variables de moins de 3 caractères.

Dupplications

Aucune duplication de code.

Conclusion

La qualité du code est globalement bonne.

Test fonctionnel de l’application

Un premier test de l’application révèle un certain nombre de problèmes;

Sécurité

- Un accès à la création et à la gestion des utilisateurs est possible sans être authentifié.

Navigation

- Le lien dans la Navbar ne redirige pas vers la page d’accueil.
- Le lien vers la liste des tâches terminées ne fonctionne pas.
- Aucun bouton annuler n’est présent sur les formulaires.
- Il n’est pas possible de naviguer entre les différentes fonctionnalités de l’application.

Utilisateurs

- Pas de bouton supprimer pour les utilisateurs

Tâches

- La liste des tâches à faire affiche toutes les tâches
- La liste des tâches terminées est inaccessible
- Lorsqu’on marque une tâche comme non faite, le message d’alerte indique toujours que la tâche a été marquée comme faite.

Conclusion

Un travail de correction des fonctionnalités de l’application devra être effectué avant de la faire évoluer.

Actions menées

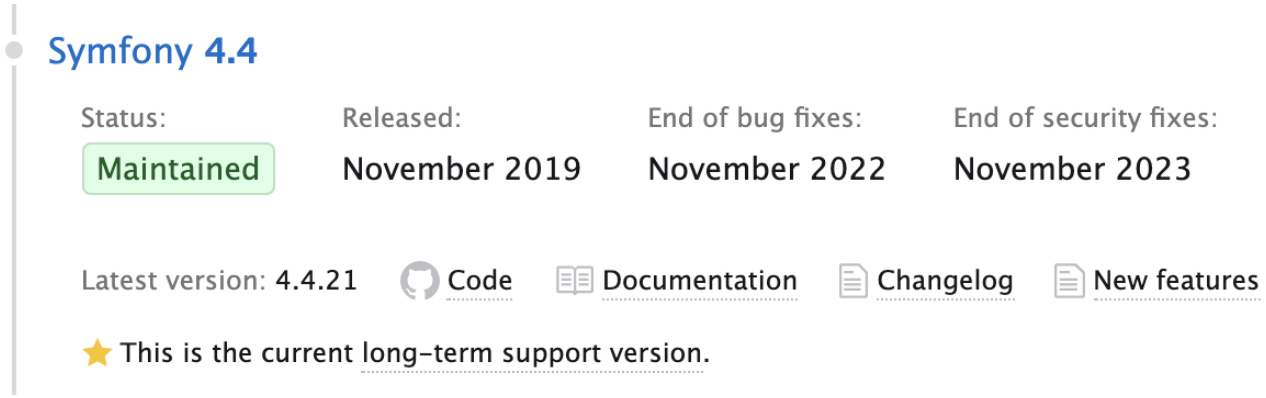
Suite à l'état des lieux, les actions suivantes ont été menées :

- Mises à jour de l'application et de ses dépendances
- Correction les différents points relevés dans l'état des lieux initial de l'application afin de réduire sa dette technique et de pérenniser ses futures évolutions.
- Correction des d'anomalies et implémentation les nouvelles fonctionnalités demandées
- Ajout d'une série d'améliorations qui me semblaient nécessaires
- Écriture d'une série de tests unitaires et fonctionnels afin de garantir le bon fonctionnement de l'application.
- Mise en place d'un environnement de développement facilitant l'analyse de la qualité du code et de la performance de l'application.

Mises à jour

Symfony et de ses dépendances

- Le framework a été passé dans la version LTS actuelle de Symfony: 4.4



- Les dépendances ont été supprimées et ré-installées via **Symfony flex**.
- Suppression de la librairie **symfony/swiftmailer-bundle** qui n'est pas utilisée dans le projet.

FrontEnd

- Mise à jour de Bootstrap vers la version 4.6
- Suppression des librairies qui n'existent plus.

Qualité

- Les différentes issues concernant la qualité du code relevées dans CodeClimate ont été fixées

Correction les bugs fonctionnels relevés

- Ajout d'un lien sur le "Brand" qui redirige vers la page d'accueil.
- Mise en place des pages "tâches à réaliser" et "tâches terminées"
- Correction des boutons d'accès aux tâches à traiter et terminées depuis la page d'accueil
- Ajout d'un bouton supprimer dans le listing des utilisateurs
- Correction du message lorsqu'une tâche est marquée comme non faite.

Anomalies et nouvelles fonctionnalités

Tâches & utilisateurs

- Ajout d'une propriété **owner** a l'entité **Task**.
- Rattachement d'une tâche à l'utilisateur connecté lors de sa création
- Attribution d'un utilisateur "anonyme" aux tâches n'ayant pas de propriétaire.

Utilisateur

- Ajout de l'attribut **Roles** à l'entité **User**.
- Ajout des roles ROLE_USER et ROLE_ADMIN
- Ajout du champ rôle sur les formulaires de création et de modification d'un utilisateur.

Autorisations

- Limitation de l'accès aux pages d'administration des utilisateurs aux utilisateurs ayant le rôle ROLE_ADMIN depuis la section **access_control** du fichier **config/package/security.yaml**
- Ajout d'un Voter permettant de réglementer la suppression d'une tâche.
    - Seul le propriétaire d'une tâche peut la supprimer
    - Les tâches liées à l'utilisateur anonyme ne peuvent être supprimées que par les utilisateurs ayant le rôle ROLE_ADMIN.

Améliorations

Sécurité

- Mise en place d'un guard authenticator pour gérer l'authentification des utilisateurs
- Ajout de contraintes de complexité de mot de passe utilisateur
- Utilisation du meilleur algorithme de cryptage de mots de passe disponible sur le serveur.
- Ajout de CSRF token sur les formulaires.

Navigation

- Ajout d'une Navbar permettant l'accès aux différentes fonctionnalités de l'application.
- Ajout de boutons annuler sur les formulaires de création/modification des utilisateurs et des tâches.

Tests unitaires et fonctionnels

Écriture d'une série de tests unitaires et fonctionnels afin de garantir le fonctionnement de l'application.

Environnement de développement

Qualité du code

Le repository a été connecté à CodeClimate et à SymfonyInsight afin d'alerter sur les éventuelles issues dans la qualité du code.

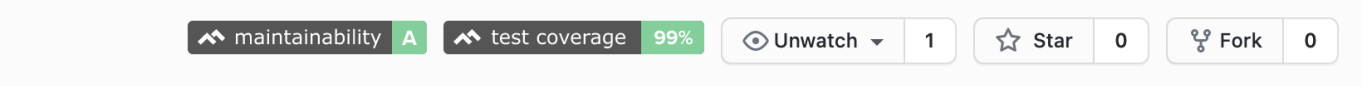
Intégration continue

Un système d'intégration continue a été mis en place via Travis-CI.

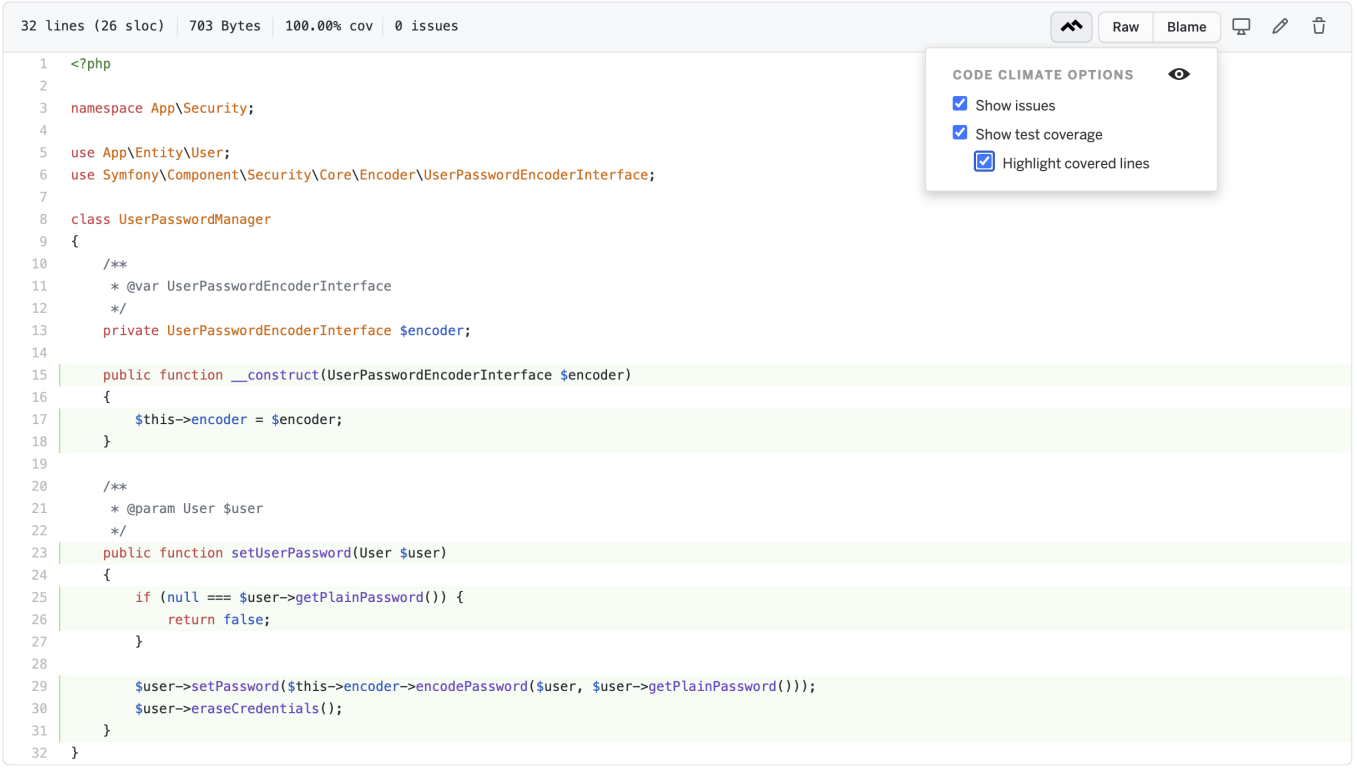
Celui-ci réalise plusieurs choses:

- Il exécute l'ensemble des tests unitaires et fonctionnels à chaque commit pour vérifier d'éventuelles erreurs dans le code.
- Le rapport de code-coverage est renvoyé vers CodeClimate pour garantir la couverture du code par les tests à un minimum de 70%.

  Note: CodeClimate étant connecté au Repository Github, les différentes issues et le code-coverage sont donc directement visible dans Github.



Apercu des indicateurs CodeClimate dans le Repository GitHub

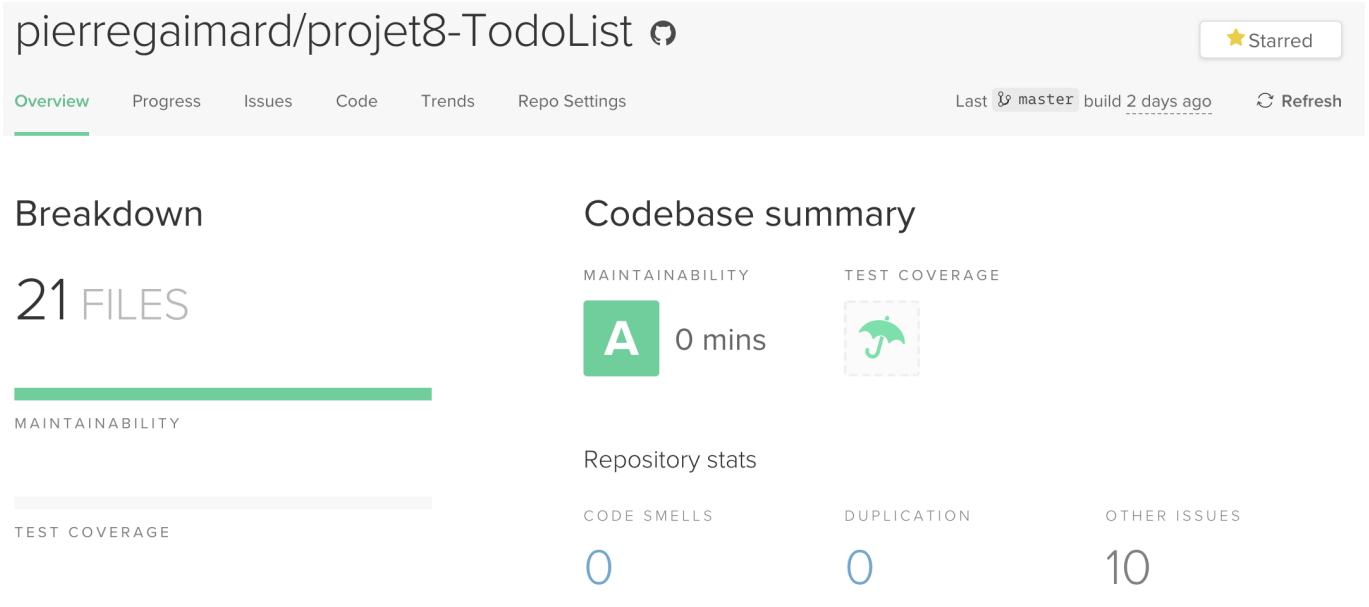


Apercu du code-coverage d'une classe dans le Repository GitHub

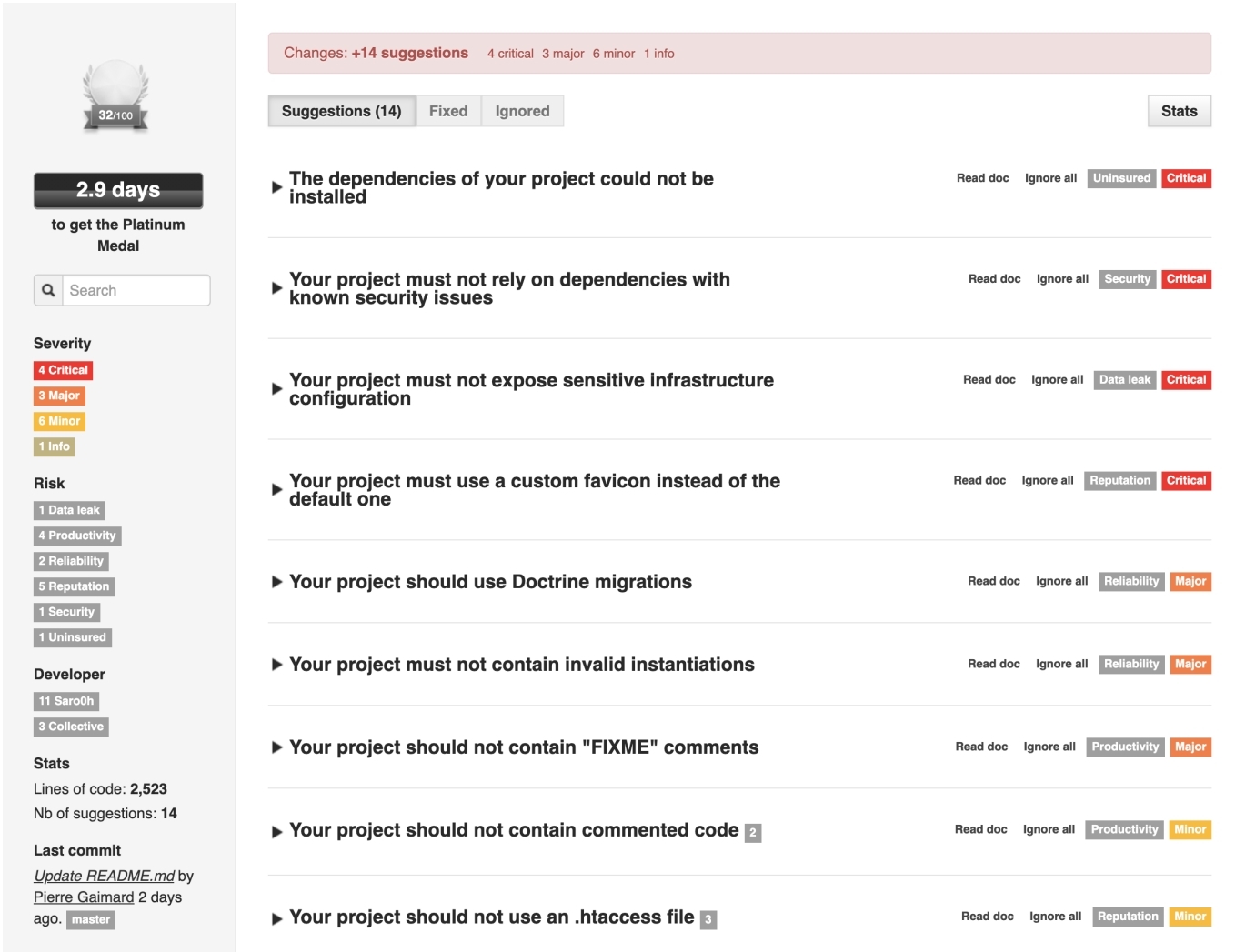
Analyse de qualité et de performance

Qualité du code

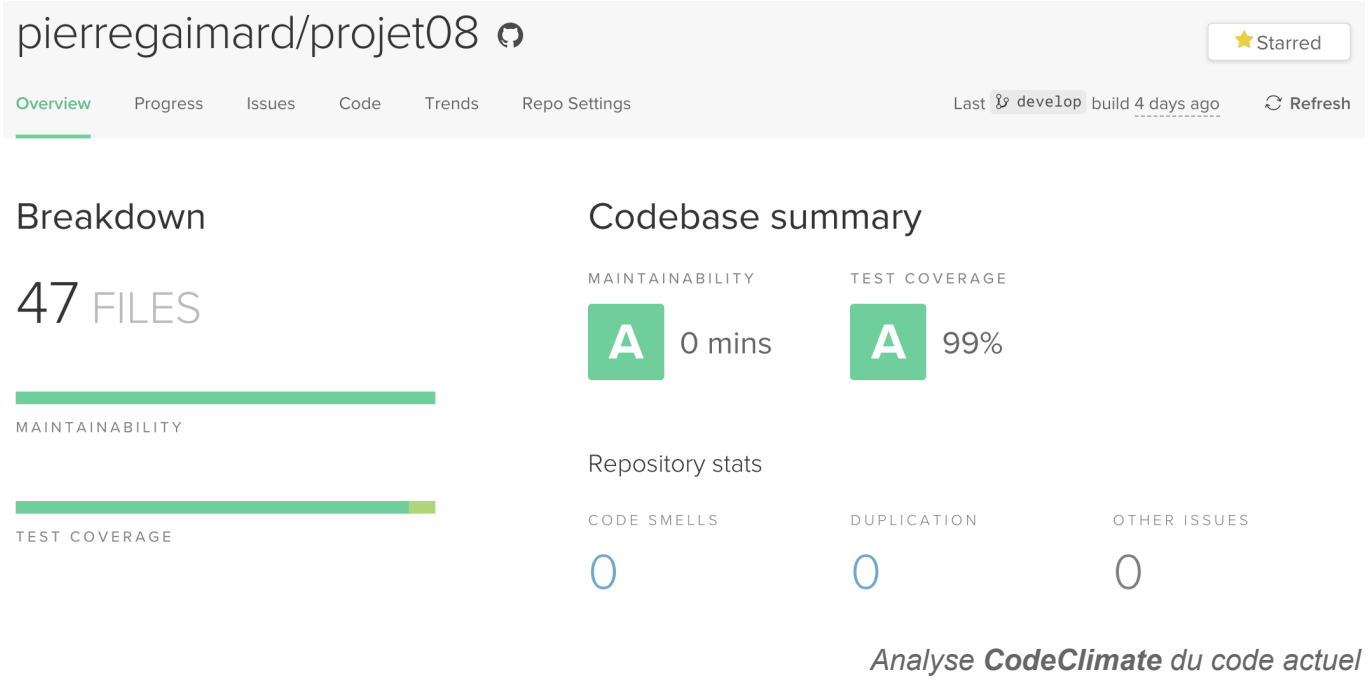
Analyse



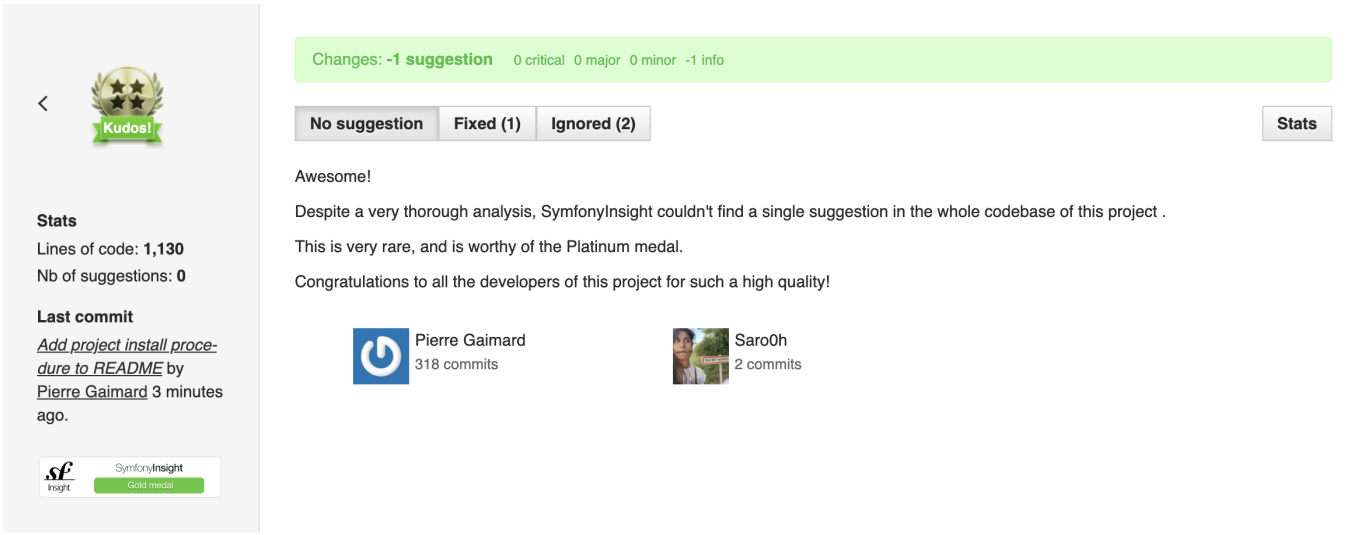
*Analyse **CodeClimate** du code initial*



*Analyse **SymfonyInsight** du code initial*



*Analyse **CodeClimate** du code actuel*



*Analyse **SymfonyInsight** du code actuel*

Conclusion

Ces rapports montrent que la dette technique de l'application a été fortement réduite.
L'ensemble des librairies sont à jour et les tests couvrent 99% du code. (Le 1% manquant correspond à la méthode logout() du SecurityController qui est obligatoire mais jamais appelée.)

Performance

Un audit de performance a été réalisé à l'aide de Blackfire.

Le test a été réalisé sur trois pages représentatrices de l'application:

- La page de login: **/login**
- La page des tâches à réaliser: **/tasks** pour le code initial et **/tasks/todo** pour le code actuel
- La liste des utilisateurs: **/users**

3 profils ont été générés pour chaque page:

- Un à partir du code initial de l'application
- Un à partir du code actuel sans optimisation
- Un après analyse et optimisation.

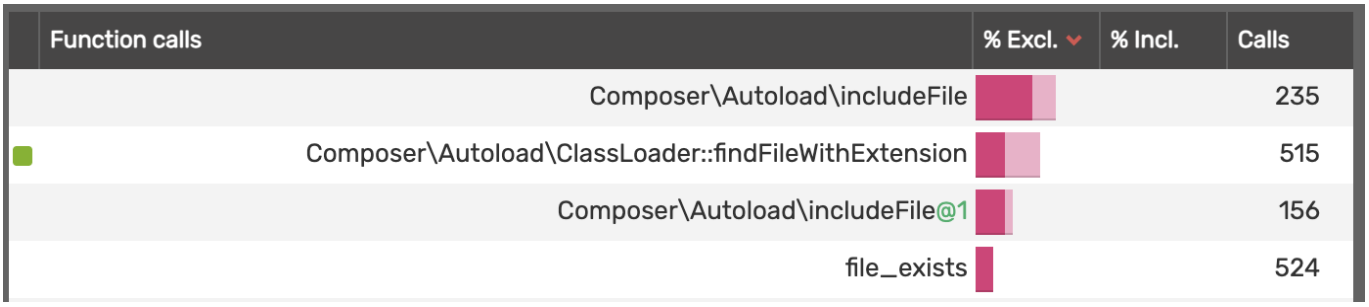
Code initial de l'application

Page	Temps de chargement	Mémoire consommée
/login	34 ms	1,78 Mo
/tasks - /tasks/todo	58,6 ms	2,97 Mo
/users	49,5 ms	2,83 Mo

Code actuel sans optimisation

Page	Temps de chargement	Mémoire consommée
/login	43,7 ms	2,45 Mo
/tasks - /tasks/todo	66,3 ms	3,56 Mo
/users	58,1 ms	3,30 Mo

On constate une augmentation du temps de chargement et de la mémoire consommée.
Suite à une analyse des graph Blackfire, je constate que ce ralentissement est principalement dû à l'utilisation de l'autoload de Composer.

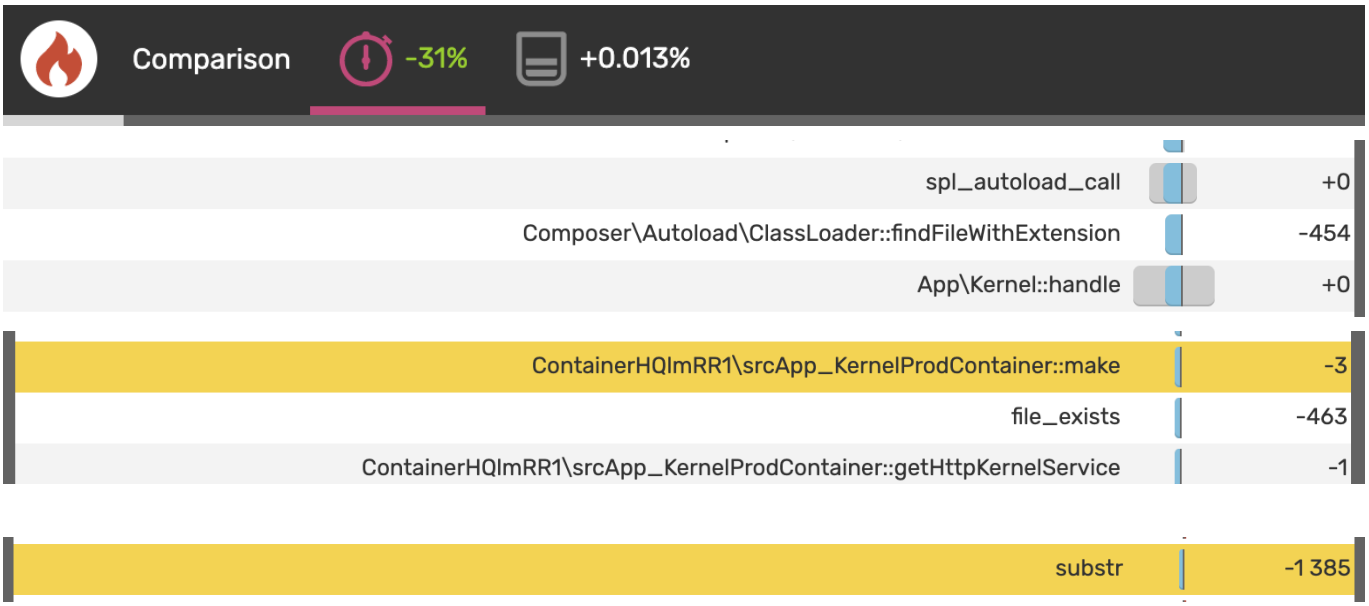


La documentation de Composer offre une solution à ce problème via la génération d'une "Class Map". Cela convertit les namespaces PSR-4 en ClassMap ce qui évite de faire appel au filesystem pour vérifier l'existence des classes.

Cette optimisation est faite via la commande **composer dump-autoload --optimize**.

En comparant le profil avant/après optimisation on constate un gain important de performance dans la rapidité d'affichage des pages sans consommation supplémentaire de mémoire significative.

En effet comme on peut le voir ci-dessous, la méthode **findFileWidthExtension()** de la classe **ClassLoader** ainsi que les méthodes PHP **file_exists** et **substr** ne sont plus appelées.



*Graph de comparaison de la page /**login** avant/après optimisation de l'autoload*

Code actuel après optimisation

Page	Temps de chargement	Mémoire consommée
/login	29,2 ms	2,45 Mo
/tasks - /tasks/todo	47,1 ms	3,56 Mo
/users	40 ms	3,3 Mo

Comparaison entre le code initial et le code actuel.

Page	Temps de chargement	Mémoire consommée
/login	-14%	+37, 8%
/tasks - /tasks/todo	-20%	+20.1%
/users	-19%	+16,5 %

Conclusion

Suite à cette optimisation, on constate une amélioration globale du temps de chargement. Par contre, la mémoire consommée est légèrement supérieure.

Je considère que cette optimisation est valable car le point le plus important est le temps de chargement de la page.

L’augmentation de la consommation de mémoire n’est pas trop problématique sachant que l’ajout de mémoire sur un serveur est peu coûteux en comparaison aux pertes potentielles générées par une application lente.