

# RAPPORT QUALITE ET PERFORMANCE

Todo and Co



Florian LEBOUL  
Openclassroom Projet 8

## Table des matières

I-	Suivi de qualité .....	2
II-	Etude de la dette technique.....	4
III-	Suivi des performances .....	5

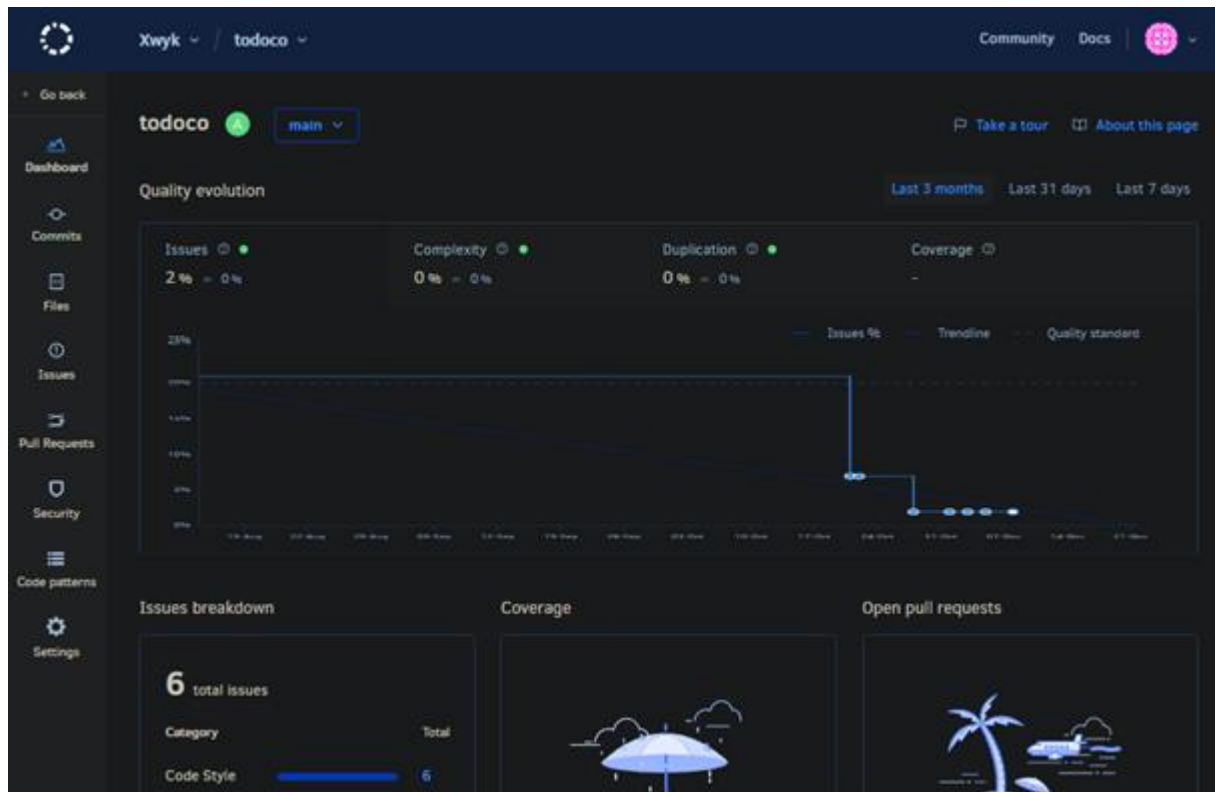
## I- Suivi de qualité

Deux outils ont été mis en place afin d'assurer le suivi de la qualité du code

Codacy

Analyse :

- Les vulnérabilités
- Les mauvaises pratiques
- Le code inutilisé
- La duplication de code

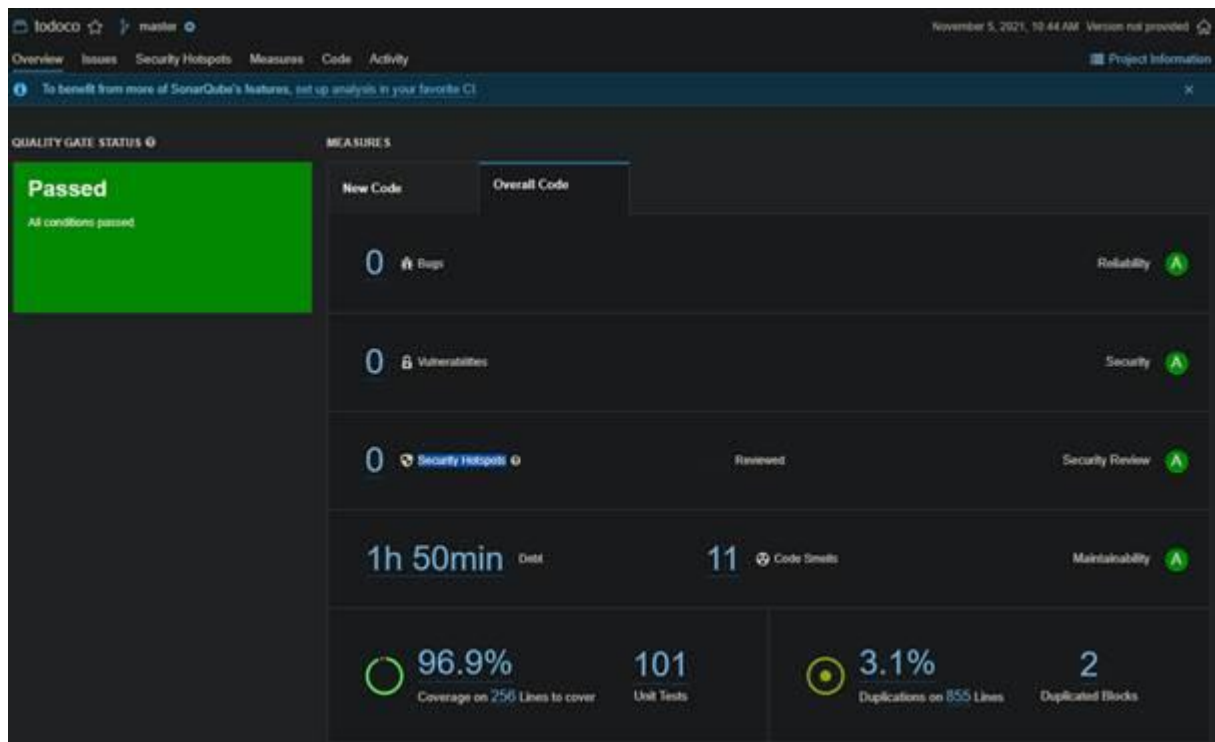


SonarQube (<https://www.sonarqube.org/>)

Analyse :

- La fiabilité du code (recherche de bugs)
- Les vulnérabilités
- Les « Security Hotspots », morceaux de code potentiellement risqués point de vue sécurité, nécessitant une vérification de la part du développeur
- Les mauvaises pratiques présentes dans le code

L'outil permet également l'interprétation des résultats PHPUnit, ainsi que la vérification de duplication de code afin de centraliser les informations sur un unique dashboard.









Pour ces deux outils, les standards stipulés dans le CONTRIBUTING.md sont analysés (PSR1-2-4)

En local sur le projet, installés via Composer, PHPStan et PHP-cs-fixer ont également permis de suivre la qualité du code, et des mauvaises pratiques de syntaxe lors du développement. PHP-cs-fixer est inclus dans la CI github, et termine l'action si la sortie est en erreur. Il est donc important de le lancer avant de pusher les modifications sur le repo.

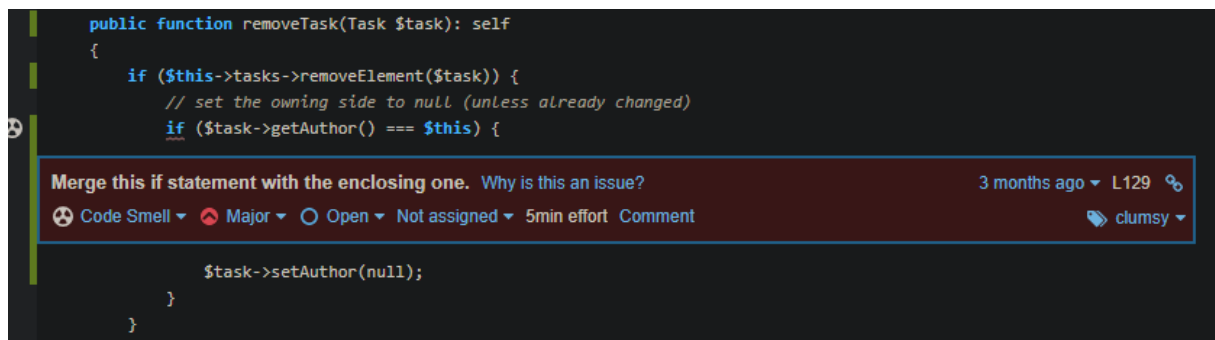
## II- Etude de la dette technique

La dette technique représente les bugs présents dans l'application, liés ou non au non-respect des règles établies.

Ici, la dette technique est estimée par l'outil de suivi de qualité SonarQube à 1h50.

Technical Debt 1h 50min 	
 src/DataFixtures/TaskFixtures.php	30min
 src/Security/Voter/UserVoter.php	30min
 src/Security/Voter/TaskVoter.php	25min
 src/Tests/XwykWebTestCaseInterface.php	20min
 src/Entity/User.php	5min

Sont donc répertoriées toutes les modifications à effectuer afin de réduire la dette. Par exemple :



On peut ici voir qu'une des modifications recommandée par l'outil est de rassembler les deux vérifications en une seule.

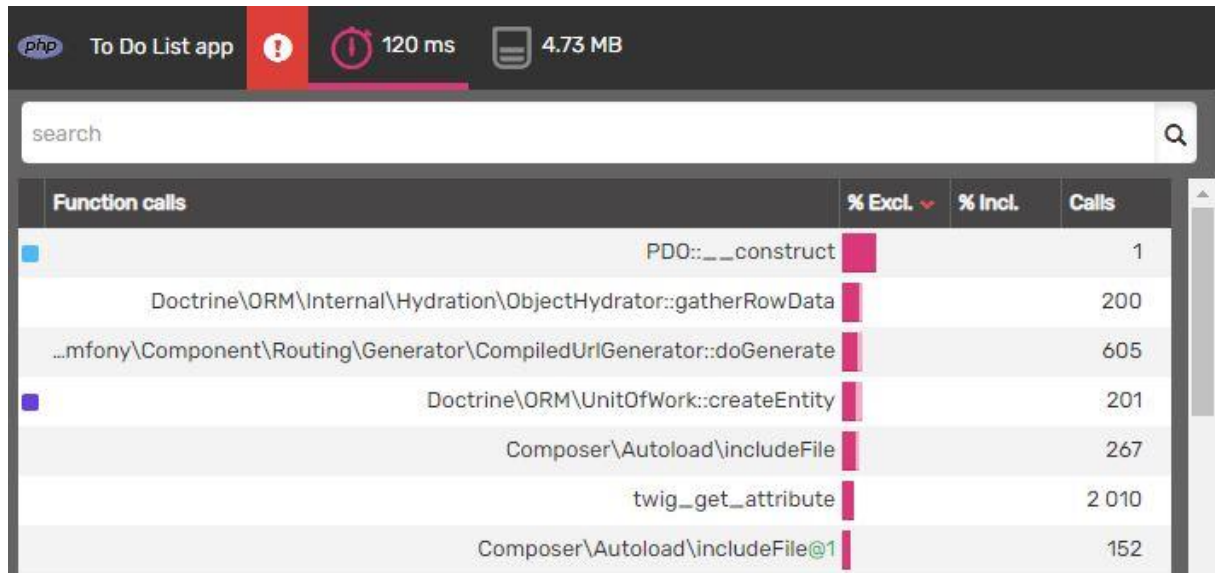
La réduction de la dette technique de l'application d'une manière générale passe par :

- L'établissement de règles de codage (ici, respect des PSR1-2-4 et best practices pour Symfony 5.3)
- Le suivi de la qualité du code au fur et à mesure du projet
- Le développement de tests unitaires et fonctionnels afin de valider chaque modification de code

### III- Suivi des performances

Le suivi des performances de l'application a été effectué avec l'outil externe BlackFire.

Ci-dessous, le résultat d'une analyse de la page de liste des tâches d'un utilisateur admin (admin1), ayant ses propres tâches (au nombre de 100), ainsi que les tâches anonymes (100 également).

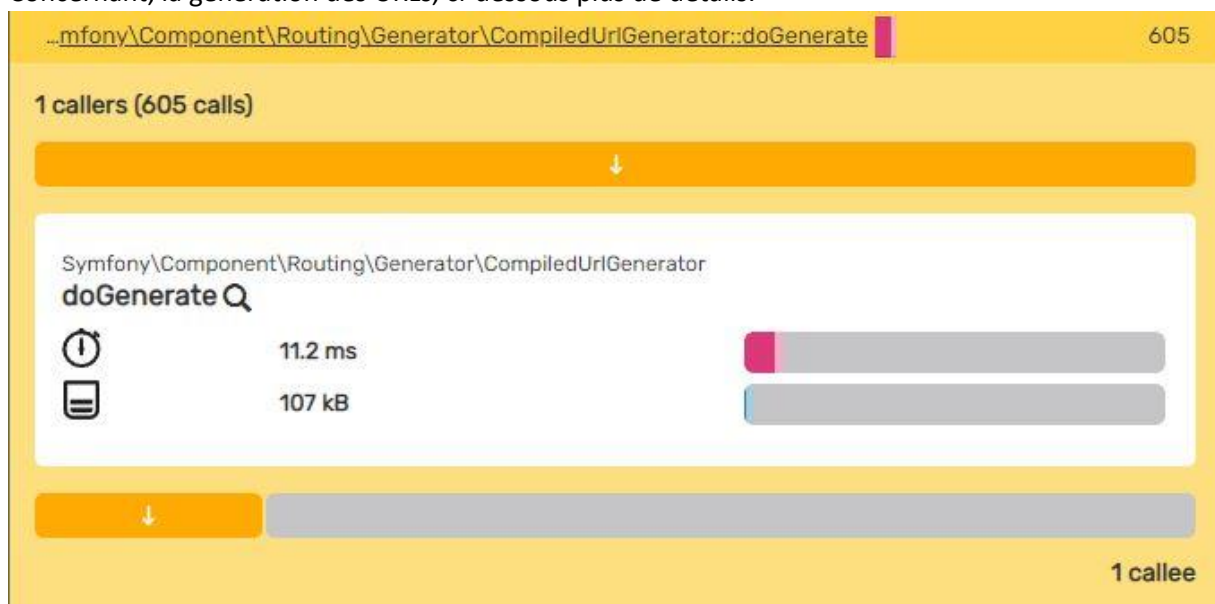


The screenshot shows the BlackFire interface for the 'To Do List app'. At the top, it displays a total time of 120 ms and a memory usage of 4.73 MB. Below this is a search bar. The main table lists function calls with columns for 'Function calls', '% Excl.', '% Incl.', and 'Calls'. The data is as follows:

Function calls	% Excl.	% Incl.	Calls
PDO::__construct			1
Doctrine\ORM\Internal\Hydration\ObjectHydrator::gatherRowData			200
...mfony\Component\Routing\Generator\CompiledUrlGenerator::doGenerate			605
Doctrine\ORM\UnitOfWork::createEntity			201
Composer\Autoload\includeFile			267
twig_get_attribute			2 010
Composer\Autoload\includeFile@1			152

Au niveau performances, outre la première ligne, PDO ::\_construct, dépendante de la configuration du serveur, on peut observer que les trois premières sources de lenteur sont la création et l'hydratation des objets dans le code, en retour des requêtes sur la base de données, ainsi que la génération des URL.

Concernant, la génération des URLs, ci-dessous plus de détails.



The screenshot shows the detailed view of the 'doGenerate' function in the BlackFire tool. It indicates that this function is called 605 times. Below this, it shows the function signature 'Symfony\Component\Routing\Generator\CompiledUrlGenerator doGenerate' with a search icon. Performance metrics are listed: 11.2 ms for execution time and 107 kB for memory usage. A progress bar shows the function's relative performance. At the bottom, it indicates '1 callee'.

Function	Time	Memory
Symfony\Component\Routing\Generator\CompiledUrlGenerator doGenerate	11.2 ms	107 kB

On peut donc constater que la seule génération d'URLs occupe 10% du temps total de la requête. Ceci s'explique par le fonctionnement du moteur Twig. Pour chaque objet Tâche à afficher (200 au total), il va rechercher par le routeur la route à utiliser pour les liens /edit, /toggle et /delete. Le nombre de requêtes pour ces seules fonctionnalités va donc être de 200 \* 3 soit 600.

Un axe d'amélioration pourrait consister à récupérer les URLs une seule fois depuis le contrôleur, ces dernières n'ayant comme seule modification que l'ID de la tâche, et de les passer au moteur Twig (par exemple sous forme de tableau, afin de ne pas relancer une recherche pour chaque élément.

605 requêtes sont actuellement exécutées, en suivant cette modification, seules 8 requêtes (les 5 non liées aux tâches et les 3 à effectuer dans le contrôleur) resteraient.

En second lieu, on peut observer le temps nécessaire à la création de nos objets par ORM :



Il ressort que la seule création de nos objets Tâche, et leur hydratation occupe 20% du temps total de la requête.

Afin de réduire ces temps, il pourrait être envisageable de travailler avec un résultat de requête sous forme de tableau. Les informations nécessaires à l'affichage d'une tâche n'étant que chaîne de caractères (titre et description), entier (id) et booléen (état), elles peuvent être passées au moteur Twig sous forme de tableau de données.

L'exploitation de données sous forme de tableau permet un accès plus rapide aux informations, et est bien moins chronophage que la création complète d'une entité pour chaque élément, suivie de son hydratation.

En conclusion, il apparaît donc qu'en apportant les deux modifications citées plus haut, le temps de la requête pourrait être réduit d'environ 30ms, soit 25 à 30% du temps total.