Document explicatif – CI/CD BobApp

1) Introduction

Ton application BobApp propose chaque jour une blague aléatoire à tes utilisateurs. Elle repose sur une architecture classique avec un back-end développé en Java (Spring Boot) et un front-end en Angular.

Le succès grandissant de l'application entraîne une augmentation du nombre d'utilisateurs, et donc de bugs signalés, que tu passes beaucoup de temps à reproduire et à corriger.

À cela s'ajoutent des déploiements entièrement manuels, longs et fastidieux, qui t'empêchent de te concentrer sur l'ajout de nouvelles fonctionnalités.

Dans ce contexte, je te propose d'automatiser les principales tâches manuelles et répétitives grâce à une pipeline CI/CD avec GitHub Actions ce qui te permettra de gagner un temps précieux.

2) Contexte et objectifs du CI/CD

La CI/CD (Continuous Integration / Continuous Delivery) est une approche qui vise à automatiser les tâches essentielles du développement logiciel, comme:

- l'exécution des tests automatisés,
- le contrôle qualité via des outils comme SonarCloud,
- la génération de rapports de couverture de tests,
- la création et la livraison continue des artefacts, comme les images Docker.

Pour répondre à tes besoins, j'ai défini les objectifs suivants :

- Automatiser l'exécution des tests à chaque pull request vers la branche main, afin de détecter rapidement les régressions.
- Mettre en place une analyse de la qualité du code (bugs, duplications...) avec SonarCloud.
- Générer et centraliser les rapports de couverture afin d'évaluer le niveau de test du projet.
- Créer une image Docker pour chaque service (front et back) et les publier sur Docker Hub.
- Empêcher la création des images Docker si une étape échoue, garantissant ainsi un pipeline de qualité strictement validé.

Cette mise en place te permettra :

- De savoir instantanément si un changement casse quelque chose.
- D'améliorer progressivement la qualité de ton code, sans y passer des heures.
- D'automatiser la création des images Docker, prêtes à être déployées.
- D'encourager d'autres développeurs à contribuer, car tout est automatisé et fiable.

3) Mise en place des GitHub Actions

Pour ce projet la mise en place du pipeline a été décomposée en 4.

- CI-CD (fichier ci-cd.yml)

  → Ce workflow est le **workflow principal**. Il orchestre et appelle l'ensemble des workflows détaillés ci-dessous. Il est déclenché sur chaque pull request ou push sur la branche main.

- Intégration continue back-end (fichier ci-back.yml) :

  → Ce workflow est composé de deux jobs distincts nommés « back » et « sonar-analysis ».

  - Pour le job «**back**», il effectue les actions suivantes :
    - Il clone le code du projet dans l'environnement du workflow
    - Installe et configure Java 11 (par le biais d'une Github action)
    - Exécute les tests unitaires du back-end (mvn clean verify)
    - Génère le rapport de couverture (**Jacoco**)
    - Upload le rapport en tant qu'artefact afin de le rendre disponible sur Github
  - Le job «**sonar**-**analysis**» quant à lui, s'exécute uniquement si les tests passent. Il utilise SonarCloud pour vérifier la qualité du code (bugs, duplication, couverture, etc.)

- Intégration continue front-end (fichier ci-front.yml)

  → Ce workflow composé de deux jobs distincts nommés « front » et « sonar-analysis ».

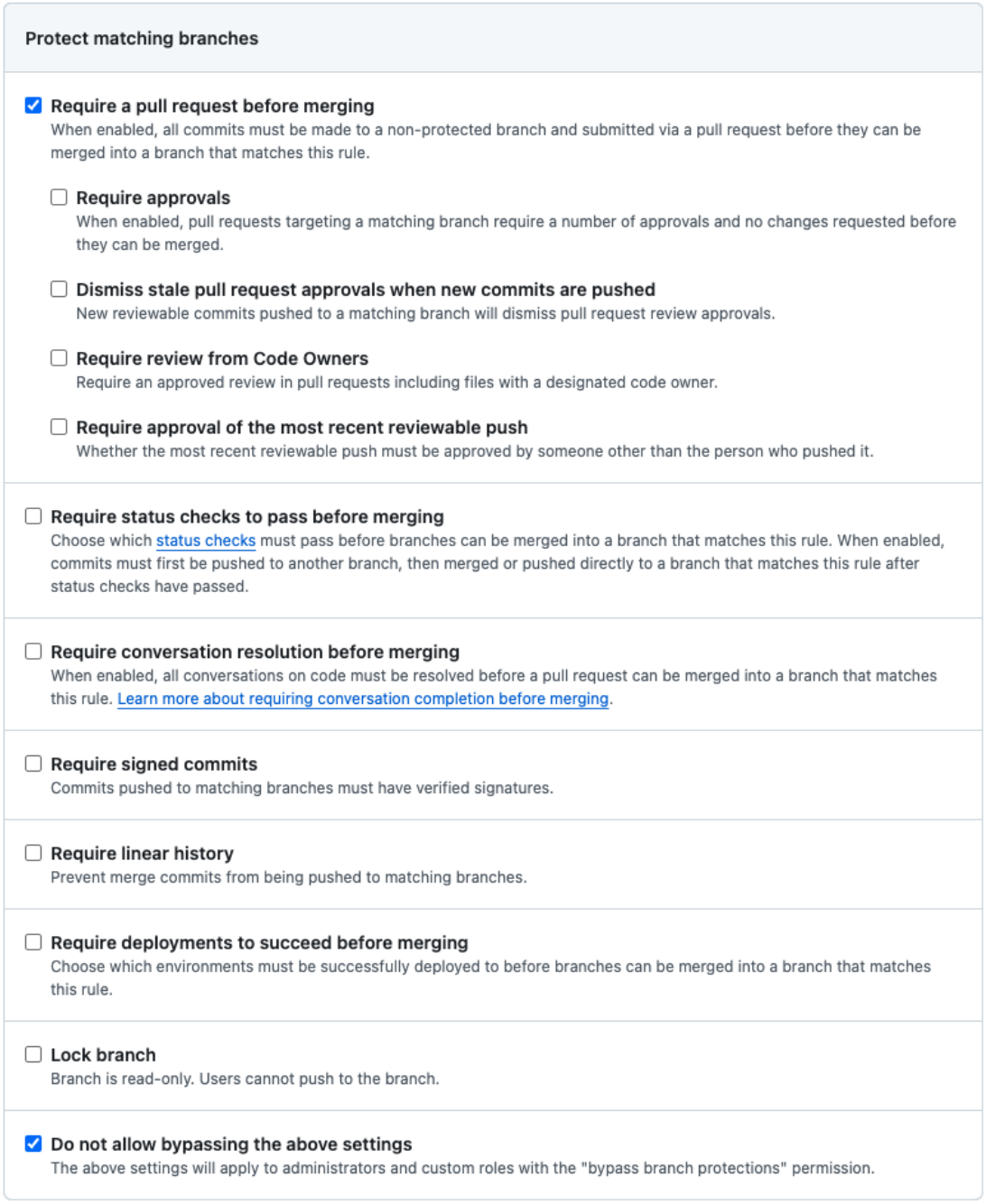
  - Pour le job « **front** », il effectue les tâches suivantes :
    - Il clone le code du projet dans l'environnement du workflow
    - Installe et configure Node.js
    - Installe les dépendances du projet
    - Exécute les tests unitaires et génère rapport de couverture
    - Upload le rapport de couverture de tests du front-end Angular en tant qu'artefact pour le rendre disponible sur Github
  - Comme pour le back, le job «**sonar**-**analysis**» quant à lui, s'exécute uniquement si les tests passent. Il utilise SonarCloud pour vérifier la qualité du code (bugs, duplication, couverture, etc.)

- Docker (fichier docker-publish.yml)

  → Il effectue les tâches suivantes :

  - Se connecte à Docker Hub
  - Construit les images docker pour le front et le back-end
  - Pousse les images taguées « latest » sur Docker Hub
  - Ne s'exécute que si les tests et la quality gate passent

➢ En complément de ces workflows, des règles de protection ont été mises en place sur la branche main :



➢ **<u>En résumé :</u>**

La branche main est protégée : aucun commit ne peut y être poussé directement.
Toute modification du code doit passer par une branche de développement, comme « develop », via une pull request.

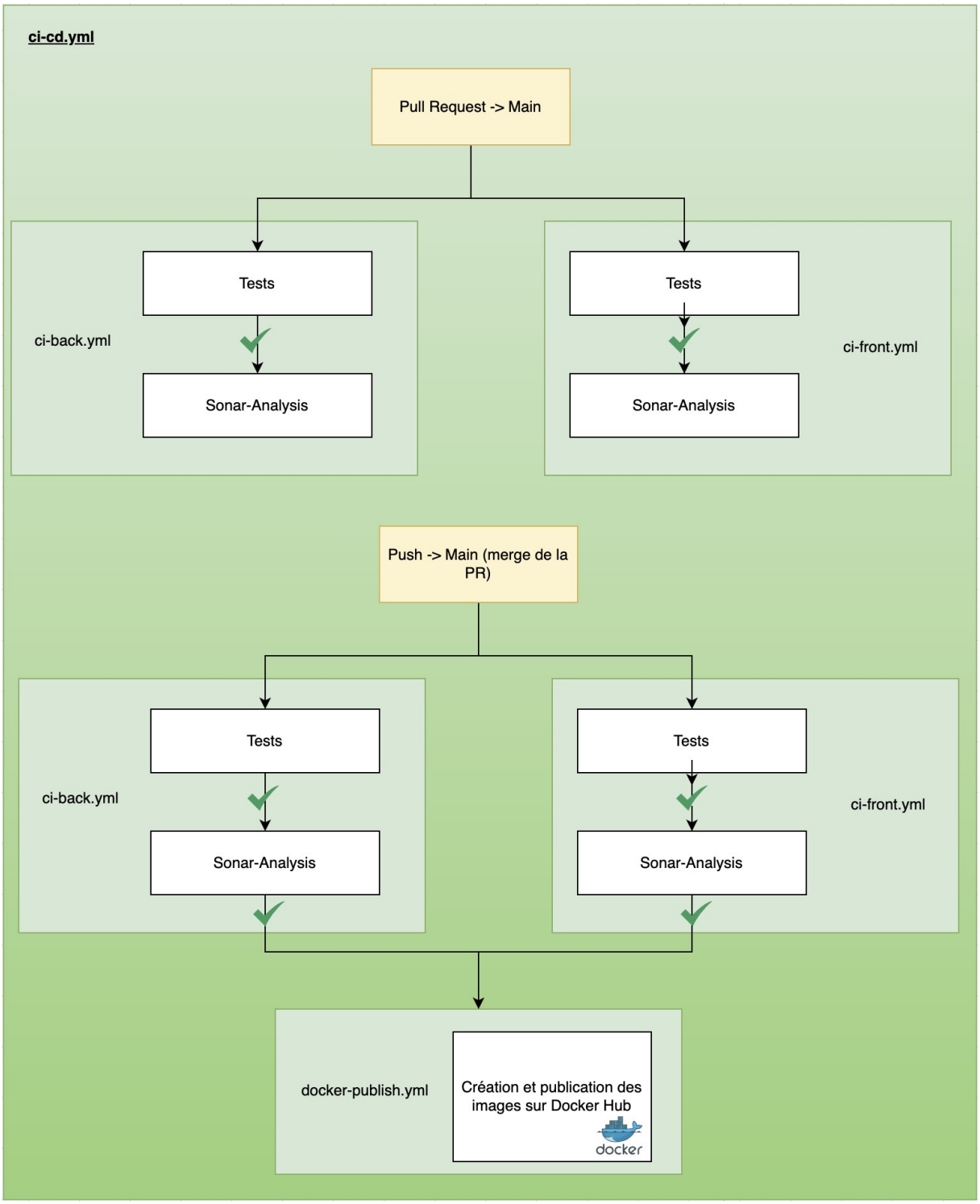
Lors de la création d'une pull request vers main, le workflow principal ci-cd.yml est déclenché. Ce dernier orchestre l'exécution des workflows « ci-back » et « ci-front », qui lancent respectivement les tests unitaires du back-end et du front-end, ainsi qu'une analyse de code via SonarCloud.

Une fois la pull request fusionnée dans main, le workflow principal est relancé automatiquement.
Il exécute à nouveau les vérifications CI. Si toutes les étapes sont validées avec succès, le workflow « docker-publish » est déclenché.

Celui-ci construit les images Docker pour le front-end et le back-end, puis les pousse sur Docker Hub.

Ce processus garantit un haut niveau de fiabilité avant chaque mise en production, tout en automatisant le cycle de développement.

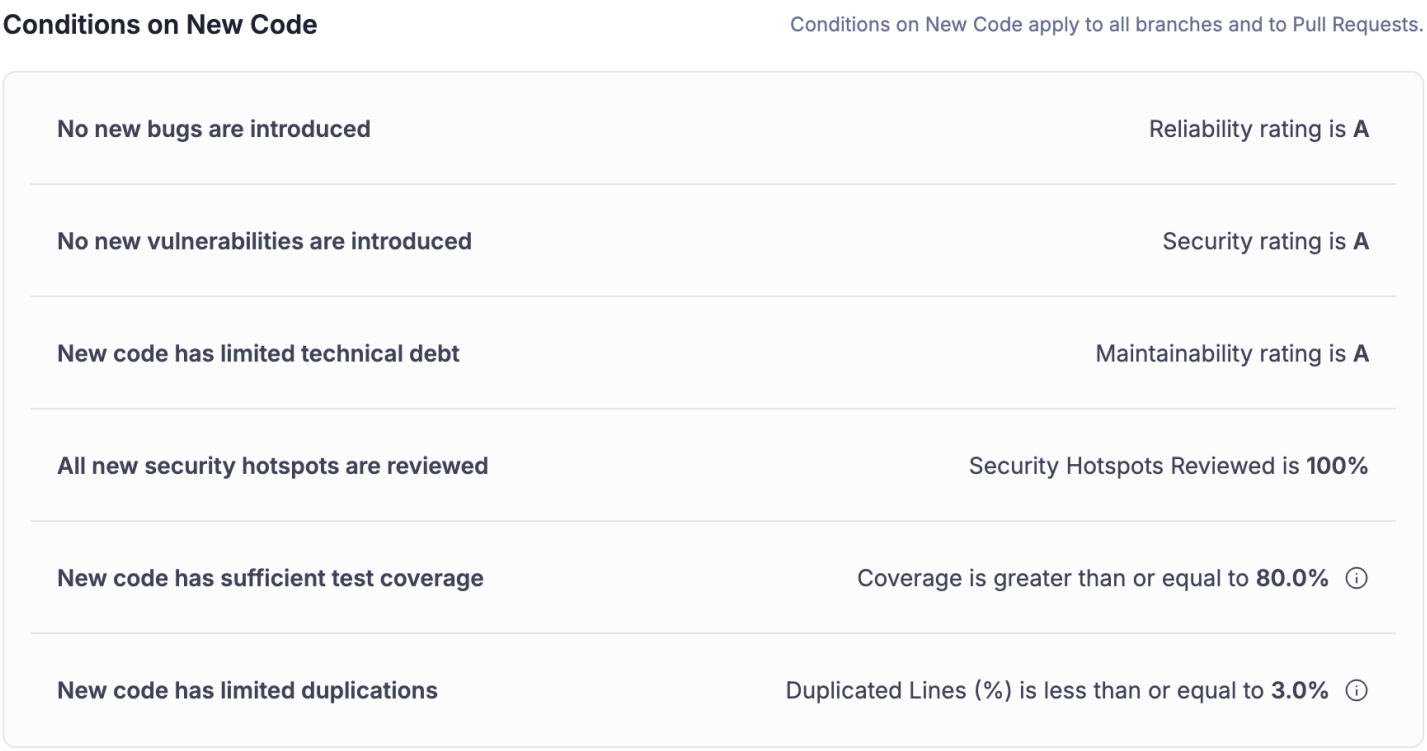
Pipeline CI/CD



4) Indicateurs de Qualité (KPI) pour le projet

Pour assurer un bon niveau de qualité logicielle durant le projet, j'ai défini un ensemble d'indicateurs clés de performance (KPI) basés sur les règles de SonarCloud.
J'ai choisi d'utiliser la Quality Gate par défaut, qui se concentre uniquement sur le nouveau code, c'est-à-dire le code ajouté récemment.

Ce choix permet de s'assurer que le code reste propre au fil des développements, sans avoir à corriger immédiatement tout le code déjà existant. C'est une approche souvent utilisée en entreprise, car elle favorise une amélioration continue de la qualité sans freiner l'avancement du projet.



A propos de ces indicateurs :

- Reliability (Fiabilité) :

  Le nouveau code ne doit pas contenir de bugs critiques ou bloquants. Le niveau attendu est un **rating A**, ce qui signifie qu'aucun bug grave n'a été introduit.

- Security (Sécurité) :

  Aucune vulnérabilité critique ne doit être ajoutée dans le nouveau code. Là aussi, un niveau A est requis pour valider cet indicateur.

- Maintainability (Maintenabilité) :

  Le nouveau code doit être propre et clair. Un rating A est attendu. SonarCloud détecte notamment ce qu'on appelle des **«code smells»** : ce sont des mauvaises pratiques ou des choix de conception qui, sans provoquer d'erreurs immédiates, rendent le code plus complexe à comprendre, modifier ou faire évoluer.

- Security Hotspot (Hotspots de sécurité) :

  Si des points de vigilance liés à la sécurité on été détectés par SonarCloud, ils doivent être examinés par les développeurs et résolus afin d'obtenir un taux de revue de 100%.

- Coverage (Taux de couverture) :

  Le taux de couverture des tests automatisés sur le nouveau code doit être d'au moins 80%.

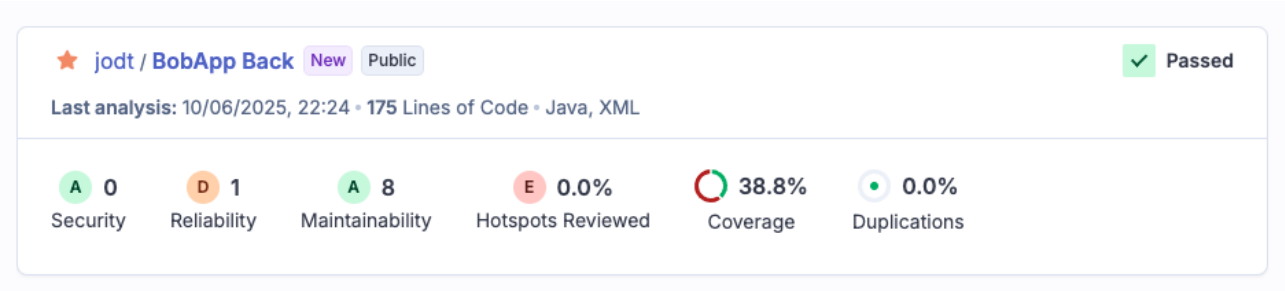
- Duplicated Lines (Lignes dupliquées) :

  Dans le nouveau code, le pourcentage de lignes dupliquées ne doit pas dépasser 3%.

Actuellement, ces indicateurs s'appliquent uniquement au nouveau code, ce qui permet d'assurer une qualité progressive sans ralentir le projet. Mais à terme, l'idée est que tout le code du projet respecte aussi ces mêmes critères de qualité.

5) Analyse des métriques

➢ Pour le back-end :

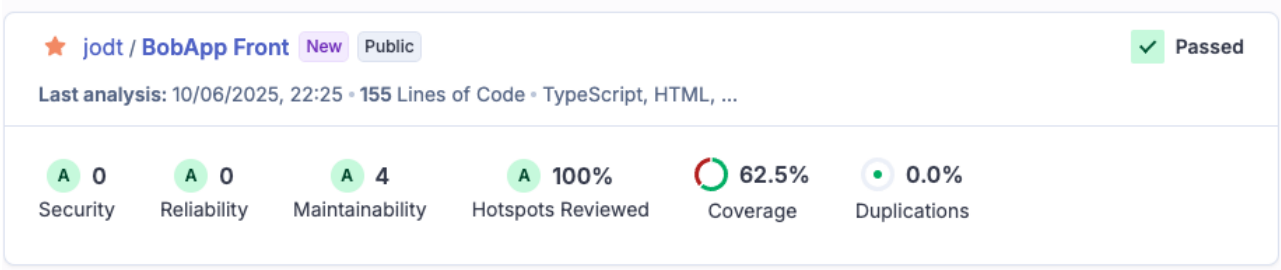


L'analyse de SonarCloud montre qu'à ce stade du projet, on a **1 bug identifié** (Reliability), **8 code smells** (des mauvaises pratiques qui n'empêchent pas le programme de fonctionner mais le rendent moins propre), et une **couverture de tests unitaires de 38,8 %**.

Même si la Quality Gate se base uniquement sur le nouveau code, il serait intéressant de commencer à améliorer le code existant petit à petit. Voici un ordre de priorité que je propose :

1. Vérifier les Security Hotspots, car ce sont des zones sensibles qui pourraient présenter un risque de sécurité si elles ne sont pas bien maîtrisées.
2. Corriger le bug détecté pour éviter tout comportement imprévu à l'exécution.
3. Traiter progressivement les code smells, afin d'avoir un code plus clair, plus propre et plus facile à maintenir.
4. Ajouter des tests au fur et à mesure, pour augmenter la couverture de code, avec un objectif d'environ 80 % à terme.

➢ Pour le front-end :



Pour le front-end, tous les indicateurs sont au vert, sauf le taux de couverture des tests unitaires, qui est actuellement de 62,5 %.
Ce n'est pas urgent, mais il serait intéressant d'ajouter progressivement quelques tests lors des développements futurs pour approcher les 80 %.
On pourrait aussi en profiter pour corriger les quelques code smells détectés, afin de garder un code propre et maintenable.

6) FeedBack des utilisateurs

La note générale de 2 sur 5, montre un mécontentement général des utilisateurs. Leurs commentaires font état de bugs rencontrés dans l'application qui dégradent l'expérience des utilisateurs. Ils se plaignent également d'un manque de réactivité dans la résolution des bugs signalés et d'un manque de réponses à leurs sollicitations. Il me semble donc **prioritaire** de résoudre ces bugs fonctionnels afin d'éviter de perdre de nouveaux utilisateurs. Une fois ces bugs fixés, on pourra réduire la dette technique progressivement.

7) Conclusion

Voilà donc le CI/CD mis en place pour ton application. En automatisant les tests et la mise à disposition des images Docker prêtes à être déployées, tu pourras consacrer plus de temps au développement de nouvelles fonctionnalités et à la correction des bugs fonctionnels.

Grâce à SonarCloud, tu disposes d'une vision claire de la qualité et de la stabilité du code, ce qui te permettra d'anticiper les problèmes avant qu'ils n'arrivent en production.

La mise à disposition des rapports de couverture de tests sur Github te permettra de garder un œil sur les différents composants qui ne sont pas ou peu couverts par les tests, et d'ajouter progressivement les tests manquants.

Ces différentes actions mises en place contribueront à la stabilité de l'application et inciteront peut-être de nouveaux développeurs à participer au projet.

Enfin, selon moi, la priorité doit être donnée à la correction des bugs fonctionnels remontés par les utilisateurs, avant de s'attaquer à la réduction de la dette technique.