

Analyse de la CI/CD BobApp

Introduction

Depuis le lancement de BobApp il y a trois ans, l'application rencontre un succès grandissant. Cependant, l'augmentation du nombre d'utilisateurs s'accompagne de difficultés de maintenance et de retours négatifs (bugs récurrents, problèmes de réactivité, etc.). Pour remédier à ces problèmes, la mise en place d'une pipeline CI/CD automatisée permettra de :

- **Valider automatiquement** les pull requests en s'assurant que les tests passent.
- **Générer et publier** les rapports de tests et de couverture.
- **Construire et déployer** les images Docker sur Docker Hub.
- **Analyser la qualité du code** via SonarCloud pour anticiper et corriger les problèmes (bugs, vulnérabilités, code smells).

Ce document décrit en détail l'architecture de la pipeline, les outils utilisés, les KPIs proposés, ainsi qu'une analyse préliminaire des métriques obtenues et des retours des utilisateurs.

Étapes de la Pipeline CI/CD

✓ Refactor/ci cd Frontend #56: Pull request #4 synchronize by jbpoujol	refactor/ci-cd
✓ Refactor/ci cd Backend #56: Pull request #4 synchronize by jbpoujol	refactor/ci-cd
✓ Refactor/ci cd SonarCloud Analysis #38: Pull request #4 synchronize by jbpoujol	refactor/ci-cd
✓ Refactor/ci cd Docker Publish Frontend #3: Pull request #4 synchronize by jbpoujol	refactor/ci-cd
✓ Refactor/ci cd Docker Publish Backend #3: Pull request #4 synchronize by jbpoujol	refactor/ci-cd

La pipeline s'appuie sur plusieurs workflows GitHub Actions, détaillés dans les fichiers YAML du projet. Voici les étapes principales :

Tests et Couverture de Code

Backend

Triggered via pull request 1 hour ago

jbpojoul synchronize #4 `refactor/ci-cd`

Status

Success

Total duration

39s

Artifacts

2

backend-tests.yml

on: pull_request

✓ Tests & Coverage

24s

⌵

−

+

Tests & Coverage summary

...

	Tests	Passed ✓	Skipped ⚠	Failed ✗
JUnit Test Report	1 ran	1 passed	0 skipped	0 failed

	Test	Result
JUnit Test Report	BobappApplicationTests.contextLoads	✓ pass

Job summary generated at run-time

- **Workflow** : backend-tests.yml
- **Processus** :
 1. Checkout du code depuis la branche principale.
 2. Installation et configuration du JDK 11 avec Maven.
 3. Exécution des tests via mvn clean verify et génération de rapports de couverture avec Jacoco.
 4. Upload des rapports d'artefacts (couverture et résultats des tests) pour consultation ultérieure.

Frontend

Triggered via pull request 1 hour ago

jbpujol synchronize #4 [refactor/ci-cd](#)

Status

Success

Total duration

59s

Artifacts

3

frontend-tests.yml

on: pull_request

✓ Tests & Coverage

48s

⌕

−

+

Tests & Coverage summary

...

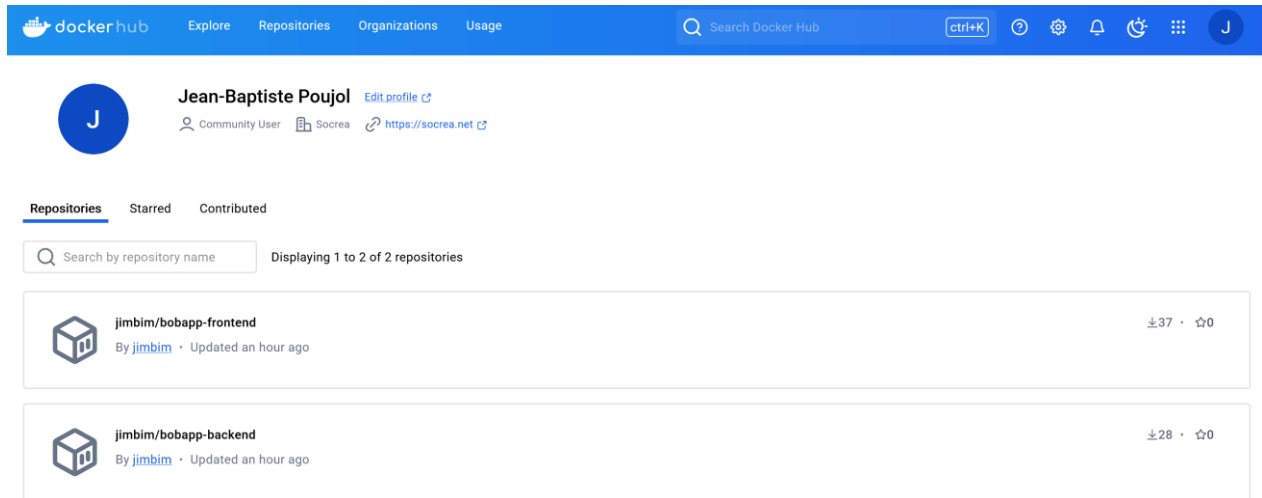
	Tests	Passed ✓	Skipped ⚠	Failed ✗
JUnit Test Report	2 ran	2 passed	0 skipped	0 failed

	Test	Result
JUnit Test Report	AppComponent.AppComponent should create the app	✓ pass
JUnit Test Report	JokesService.JokesService should be created	✓ pass

Job summary generated at run-time

- **Workflow** : frontend-tests.yml
- **Processus** :
 1. Checkout du code et installation de Node.js (version 16) ainsi que des dépendances via npm ci.
 2. Exécution des tests en mode headless avec génération de rapports de couverture via Karma.
 3. Upload des rapports de couverture et de tests pour suivi de la qualité.

Build et Publication des Images Docker



The screenshot shows the Docker Hub profile of Jean-Baptiste Poujol. The header includes the Docker Hub logo, navigation links (Explore, Repositories, Organizations, Usage), a search bar, and a user profile icon. The profile section shows the user's name, a community user status, and a link to their website. Below this, there are tabs for Repositories, Starred, and Contributed. A search bar for repositories is present, followed by a list of two repositories: jimbim/bobapp-frontend and jimbim/bobapp-backend. Each repository entry shows the Docker icon, the repository name, the user jimbim, the update time, and the number of downloads and stars.

Repositories | Starred | Contributed

Search by repository name | Displaying 1 to 2 of 2 repositories

Repository	By	Updated	Downloads	Stars
jimbim/bobapp-frontend	jimbim	Updated an hour ago	±37	☆0
jimbim/bobapp-backend	jimbim	Updated an hour ago	±28	☆0

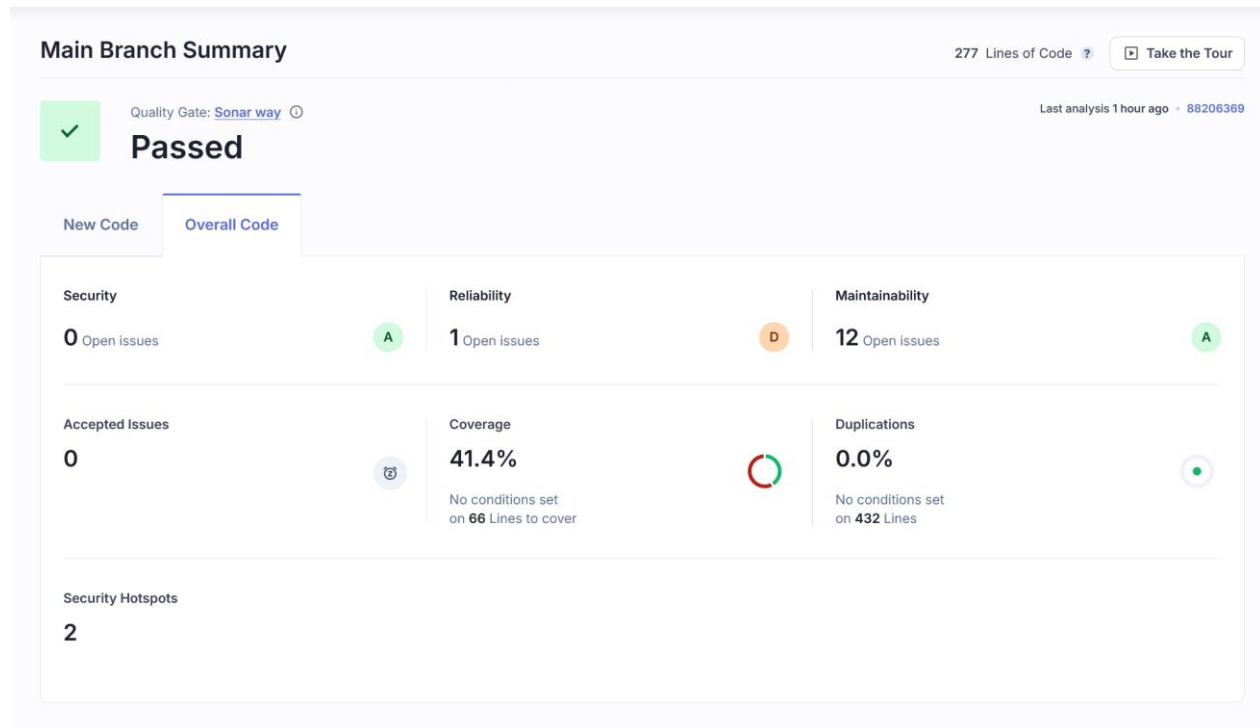
Backend

- **Workflow** : docker-publish-backend.yml
- **Processus** :
 1. Login sur Docker Hub.
 2. Extraction des métadonnées pour la génération des tags et labels.
 3. Build et push de l'image Docker du backend vers Docker Hub.

Frontend

- **Workflow** : docker-publish-frontend.yml
- **Processus** : Processus similaire à celui du backend pour l'image du frontend, incluant l'authentification, l'extraction des métadonnées et le push vers Docker Hub.

Analyse de la Qualité du Code avec SonarCloud



- **Workflow :** sonarcloud.yml
- **Processus :**
 1. Checkout complet du repository (fetch-depth: 0 pour une analyse complète).
 2. Configuration des environnements Java et Node.js pour exécuter les tests sur le backend et le frontend.
 3. Exécution des tests et collecte des rapports de couverture.
 4. Scan SonarCloud pour analyser la qualité du code, identifier les bugs, vulnérabilités, code smells, et issues bloquantes.
 5. Mise en cache des paquets Sonar pour accélérer les analyses ultérieures.

Outils Utilisés

- **GitHub Actions :** Orchestration et automatisation de l'ensemble des workflows (tests, build, déploiement, analyse).
- **Maven & JDK 11 :** Gestion et exécution des tests pour le backend (Spring Boot).

- **Node.js & npm** : Gestion des dépendances et exécution des tests pour le frontend (Angular).
- **Docker** : Construction et déploiement des conteneurs pour le back et le front.
- **SonarCloud** : Analyse continue de la qualité du code, identification des problèmes (bugs, vulnérabilités, code smells, etc.).

KPIs et Métriques

Proposition de KPIs

Pour garantir un niveau de qualité et de performance satisfaisant, il est proposé de suivre au moins les deux KPIs suivants :

Taux de Couverture du Code (Coverage) Minimum

Seuil proposé : Minimum 80% de couverture pour valider une pull request.

Ce KPI permet de s'assurer qu'une part significative du code est testée, réduisant ainsi le risque de régressions et de bugs non détectés.

Nombre de Nouvelles Issues Bloquantes (New Blocker Issues)

Seuil proposé : 0 nouvelle issue bloquante lors de chaque analyse SonarCloud.

L'objectif est de maintenir une qualité de code élevée en évitant l'introduction de problèmes critiques pouvant impacter la stabilité ou la sécurité de l'application.

Code smells

Seuil proposé : Par exemple, limiter le nombre de code smells à un maximum de 20 pour l'ensemble du projet, ou encore mieux, réduire leur ratio (par exemple, moins de 10 code smells pour 1 000 lignes de code).

L'objectif est ici d'améliorer la maintenabilité et réduire la dette technique

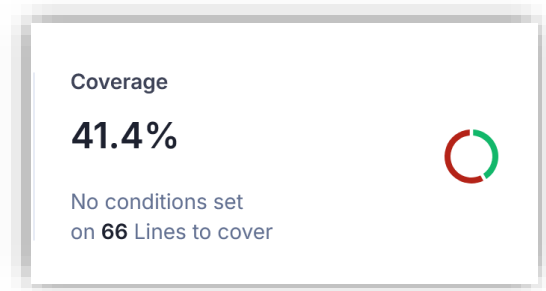
Sécurité hotspots.

Seuil proposé : L'objectif est d'avoir 0 Security Hotspots non traités lors de chaque analyse

Analyse des Métriques

Couverture du Code (Coverage)

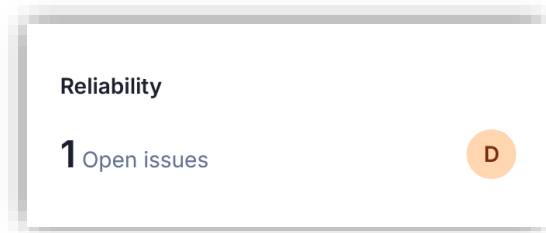
- Valeur actuelle : 41,4%
- Objectif : 80% minimum
- Écart : -38,6 points



La couverture de code reste bien en deçà de l'objectif. Les tests existants ne couvrent qu'une partie des classes et méthodes, exposant l'application à un risque élevé de régressions ou de bugs non détectés.

Fiabilité (Reliability)

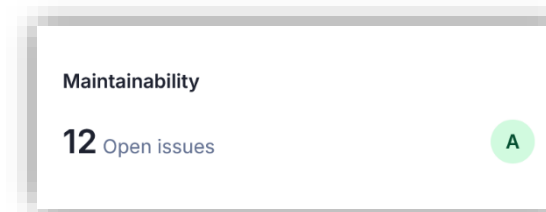
- Note : D (sur SonarCloud)
- Issues ouvertes : 1 (dont 1 critique)



Une seule issue critique peut avoir un impact significatif si elle n'est pas corrigée rapidement. Il faut s'assurer que cette issue critique ne se transforme pas en bug bloquant.

Maintenabilité (Maintainability)

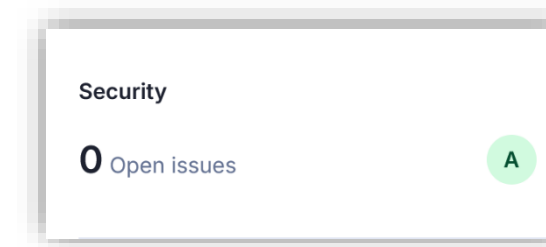
- Note : A
- Issues ouvertes : 12, essentiellement des code smells



Nécessité d'améliorer la maintenabilité du code. Les code smells identifiés (12) augmentent la dette technique et peuvent rendre le code difficile à faire évoluer.

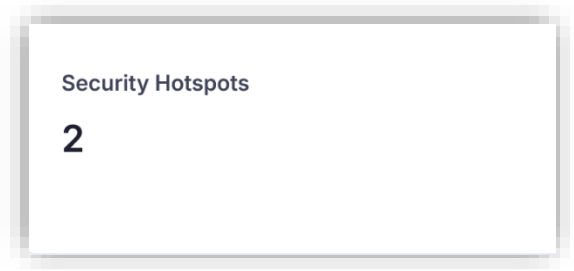
Sécurité (Security)

- Note : A
- Vulnérabilités : 0



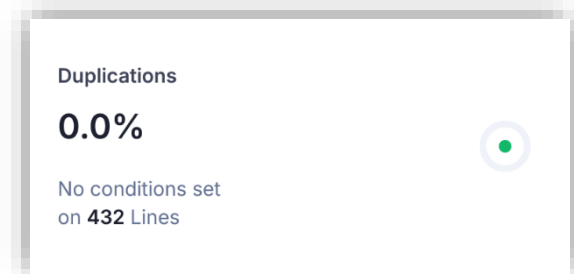
Security Hotspots : 2

- Aucune vulnérabilité directe, ce qui est positif.
- Toutefois, 2 security hotspots nécessitent une revue approfondie pour déterminer s'ils représentent un risque réel ou un faux positif.



Duplication : 0 %

- Acceptés (Issues fermées) : 0
- Lignes à couvrir : 66 pour le back, 432 pour le front (ou globalement 498 selon la vue).



Globalement, le projet passe la Quality Gate de Sonar (statut "Passed"), mais les indicateurs montrent plusieurs points d'attention, notamment la faible couverture de tests et la dette technique (code smells, hotspots).

Retours des Utilisateurs (Notes et Avis)

Les avis des utilisateurs récoltés en ligne mettent en lumière plusieurs problèmes d'ergonomie et de fonctionnalité :

Problème du bouton de suggestion de blague :

Un utilisateur signale que le bouton reste bloqué et provoque le plantage du navigateur, empêchant l'envoi de suggestions.

Bug sur le post de vidéo :

Un autre utilisateur indique que le bug remonté sur le post de vidéo persiste depuis plusieurs semaines.

Absence de notifications par e-mail :

Un utilisateur ne reçoit plus aucune notification par e-mail malgré plusieurs relances.

Expérience globale décevante :

Un dernier commentaire exprime une déception générale, poussant l'utilisateur à retirer BobApp de ses favoris.

Ces retours démontrent que, malgré un fonctionnement global acceptable, certaines fonctionnalités critiques posent problème et impactent négativement l'expérience utilisateur.

Recommandations

Pour améliorer la qualité de BobApp et répondre aux attentes des utilisateurs, les actions suivantes sont recommandées :

Correction des Bugs Frontaux :

- Bouton de suggestion de blague : Diagnostiquer et corriger le problème pour éviter le plantage du navigateur.
- Post de vidéo : Identifier et résoudre le bug récurrent signalé.
- Amélioration de la Communication par E-mail :
- Mettre en place un système de notifications plus fiable afin d'assurer une meilleure réactivité aux messages des utilisateurs.

Optimisation de la Qualité du Code :

- Renforcer les tests pour atteindre ou dépasser le seuil de 80% de couverture.
- Corriger les bugs et hotspots de sécurité identifiés par SonarCloud afin d'éliminer les issues bloquantes.
- Réduire le nombre de code smells afin de faciliter la maintenance et l'évolution du code.

Suivi Continu des KPIs :

Intégrer ces indicateurs dans le tableau de bord de suivi pour ajuster en continu les efforts d'amélioration et garantir la qualité lors de chaque nouvelle pull request.

Conclusion Finale

La mise en place de la CI/CD a permis d'identifier plusieurs points d'amélioration critiques dans l'application. Les métriques actuelles montrent un besoin urgent d'augmenter la couverture de tests et de résoudre les problèmes de qualité détectés.

Elle permettra également d'automatiser les validations et déploiements de BobApp tout en assurant une surveillance continue de la qualité du code.

Les KPIs proposés – un taux de couverture minimum de 80% et zéro nouvelle issue bloquante – constituent des objectifs ambitieux mais essentiels pour améliorer la stabilité et la sécurité de l'application. Par ailleurs, l'analyse des retours utilisateurs met en évidence des dysfonctionnements critiques (bouton de suggestion, post de vidéo, notifications e-mail) qui devront être traités en priorité pour regagner la confiance des utilisateurs.