

BobApp : Workflow CI/CD

Sommaire

Étapes mises en œuvre dans les GitHub Actions

1. Déclencheurs du workflow
2. Jobs
 - backend-testing-and-sonarcloud-analysis
 - frontend-testing-and-sonarcloud-analysis
 - docker-build-and-push

KPI Proposés

1. Couverture des tests (Code Coverage)
2. Duplication du code

Analyse des métriques et des avis utilisateurs

1. Métriques
 - Backend
 - Frontend
2. Analyse des avis utilisateurs et des métriques obtenues
 - Avis utilisateurs pertinents et priorités
 - Priorité des métriques à améliorer

Étapes mises en œuvre dans les GitHub Actions

1. Déclencheurs du workflow

Backend

- **Événements push :** Le workflow est déclenché automatiquement à chaque push de modifications sur la branche main, à condition que ces changements affectent les fichiers situés dans les dossiers back/** ou .github/workflows/**.
- **Événements pull_request :** Le workflow se déclenche aussi pour les événements de pull_request vers la branche main, incluant l'ouverture, la synchronisation et la réouverture des pull requests. Cela garantit que tout nouveau code proposé pour intégration dans main est rigoureusement testé avant fusion.

Frontend

- **Événements push** : Le workflow est déclenché automatiquement à chaque push de modifications sur la branche main, à condition que ces changements affectent les fichiers situés dans les dossiers front/** ou .github/workflows/**.
- **Événements pull_request** : Le workflow se déclenche aussi pour les événements de pull_request vers la branche main, incluant l'ouverture, la synchronisation et la réouverture des pull requests. Cela garantit que tout nouveau code proposé pour intégration dans main est rigoureusement testé avant fusion.

2. Jobs

Backend

1. backend-run-tests-for-protection-rule

- **Objectif** : Vérifier que les tests passent pour pouvoir merge les branches
- **Étapes** :
 - **Checkout Repository** : Récupère le code source du dépôt. La profondeur de clonage est définie à 0 pour obtenir l'historique complet, nécessaire pour certains outils de vérification.
 - **Set up Java** : Configure l'environnement Java avec la version 11, utilisant la distribution Temurin.
 - **Install Dependencies and Run Tests** : Utilise les commandes Maven pour installer les dépendances nécessaires et exécuter les tests.

2. backend-testing-and-sonarcloud-analysis

- **Objectif** : Valider les tests sur une pull request et analyser avec SonarCloud.
- **Étapes** :

- **Checkout Repository** : Récupère le code source du dépôt. La profondeur de clonage est définie à 0 pour obtenir l'historique complet, nécessaire pour certains outils de vérification.
- **Set up Java** : Configure l'environnement Java avec la version 11, utilisant la distribution Temurin.
- **Install Dependencies and Run Coverage** : Utilise les commandes Maven pour installer les dépendances nécessaires et exécuter les tests, incluant la mesure de la couverture du code. Un rapport de couverture est généré automatiquement à l'aide de JaCoCo.
- **Upload Code Coverage Report** : Télécharge le rapport de couverture généré (jacoco.xml) en tant que artifact pour une consultation ultérieure.
- **SonarCloud Scan** : Effectue une analyse de qualité du code à l'aide de SonarCloud.

3. docker-build-and-push

- **Objectif** : Construire et déployer l'image Docker.
- **Dépendance** : Nécessite la réussite du job backend-testing-and-sonarcloud-analysis.
- **Condition** : Le job s'exécute uniquement pour les pull requests vers la branche main.
- **Étapes** :
 - **Checkout Repository** : Récupère une fois de plus le code source du dépôt.
 - **Log in to Docker Hub** : Se connecte à Docker Hub en utilisant les informations d'identification stockées en tant que secrets. Cela permet d'authentifier le déploiement de l'image.
 - **Build and Push Docker Image** : Construit l'image Docker à partir du répertoire ./back et lui attribue un tag. Ensuite, elle pousse l'image vers Docker Hub, ce qui rend

l'application prête à être déployée sur des serveurs ou des environnements de production.

Frontend

1. frontend-run-tests-for-protection-rule

- **Objectif** : Vérifier que les tests passent pour pouvoir merge les branches
- **Étapes** :
 - **Checkout Repository** : Récupère le code source du dépôt. La profondeur de clonage est définie à 0 pour obtenir l'historique complet, nécessaire pour certains outils de vérification.
 - **Set up Node.js** : Configure l'environnement Node.js avec la version 16.
 - **Install Dependencies and Run Test** : Utilise les commandes NPM pour installer les dépendances et lancer les tests.

2. frontend-testing-and-sonarcloud-analysis

- **Objectif** : Valider les tests sur une pull request et analyser avec SonarCloud.
- **Étapes** :
 - **Checkout Repository** : Récupère le code source du dépôt. La profondeur de clonage est définie à 0 pour obtenir l'historique complet, nécessaire pour certains outils de vérification.
 - **Set up Node.js** : Configure l'environnement Node.js avec la version 16.
 - **Install Dependencies and Run Coverage** : Utilise les commandes NPM pour installer les dépendances et lancer les tests, avec une analyse de la couverture du code. Un rapport de couverture est produit via Karma.

- **Upload Code Coverage Report** : Télécharge le rapport de couverture généré (lcov.info) en tant que artifact pour une consultation ultérieure.
- **SonarCloud Scan** : Effectue une analyse de qualité du code à l'aide de SonarCloud.

3. docker-build-and-push

- **Objectif** : Construire et déployer l'image Docker.
 - **Dépendance** : Nécessite la réussite du job frontend-testing-and-sonarcloud-analysis.
 - **Condition** : Le job s'exécute uniquement pour les pull requests vers la branche main.
 - **Étapes** :
 - **Checkout Repository** : Récupère une fois de plus le code source du dépôt.
 - **Log in to Docker Hub** : Se connecte à Docker Hub en utilisant les informations d'identification stockées en tant que secrets. Cela permet d'authentifier le déploiement de l'image.
 - **Build and Push Docker Image** : Construit l'image Docker à partir du répertoire ./front et lui attribue un tag. Ensuite, elle pousse l'image vers Docker Hub, ce qui rend l'application prête à être déployée sur des serveurs ou des environnements de production.
-

KPI Proposés

1. Couverture des tests (Code Coverage)

- **Seuil proposé** : 80 %
- **Explication** : Un taux de couverture de 80% permet de garantir que l'essentiel du code est testé, ce qui réduit les risques de bugs en

production. Ce seuil est un compromis reconnu entre une couverture élevée et une efficacité de développement raisonnable.

2. Duplication du code

- **Seuil proposé : 3%**
- **Explication :** Un taux de duplication de code de 3% ou moins aide à maintenir un code propre et facile à maintenir. Un code dupliqué peut entraîner une augmentation des efforts de maintenance et des risques de bugs.

Il est possible de modifier les KPI mentionnés ci-dessus directement dans Sonar. Voici quelques images pour illustrer cela.

sonarcloud

bobapp-front

PUBLIC

Overview

Main Branch

Pull Requests 2

Branches 2

Information

Administration

My Projects

My Issues

Explore

Clara Bernadou | Azuryah > bobapp-front > Overview

B

bobapp-front

No tags

Last analysis 07 Nov 2024 109 Lines of Code

TypeScript HTML CSS

Main Branch Status

Quality Gate 7

Passed

General Settings

Analysis Method

Quality Profiles

Quality Gate

New Code

Links

Permissions

Background Tasks

Update Key

Webhooks

Deletion

Enjoy your sparkling clean code!

See Full Analysis

Main Branch Evolution since 8 days ago

4 Issues

Issues Coverage Duplications

Issues New Code

See full history

Latest Activity

https://sonarcloud.io/project/quality_gate?id=clarabernadou-bobapp-front

sonarcloud

bobapp-front

PUBLIC

Overview

Main Branch

Pull Requests 2

Branches 2

Information

Administration

My Projects

My Issues

Explore

Clara Bernadou | Azuryah > bobapp-front > Administration / Quality Gates

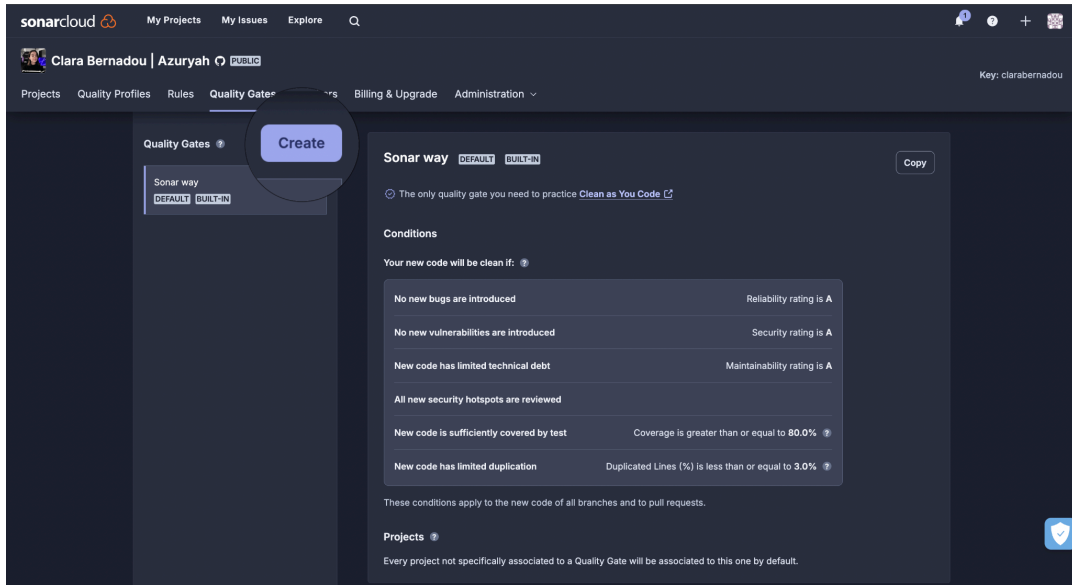
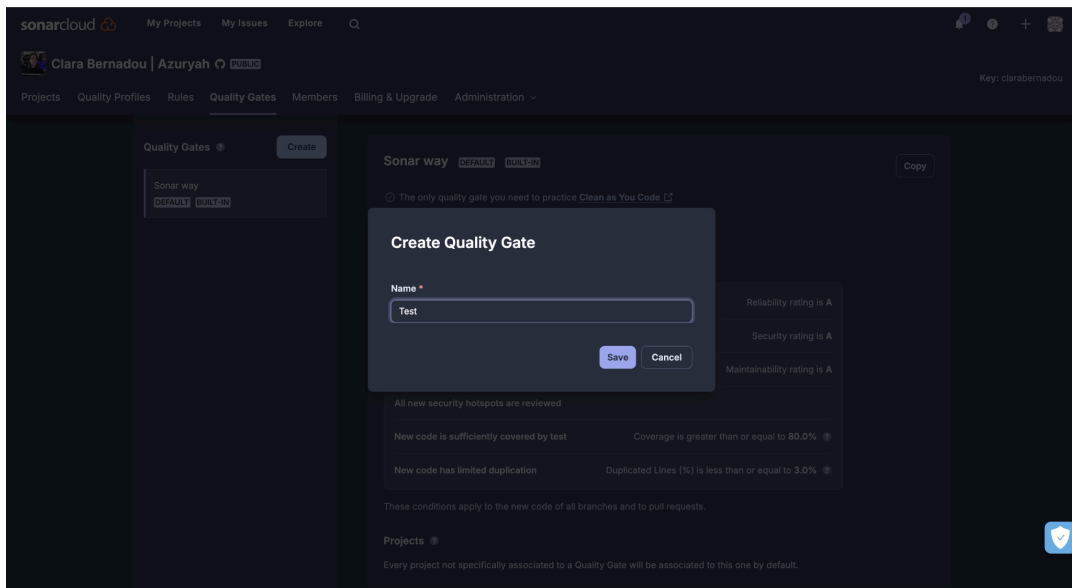
Quality Gate 7

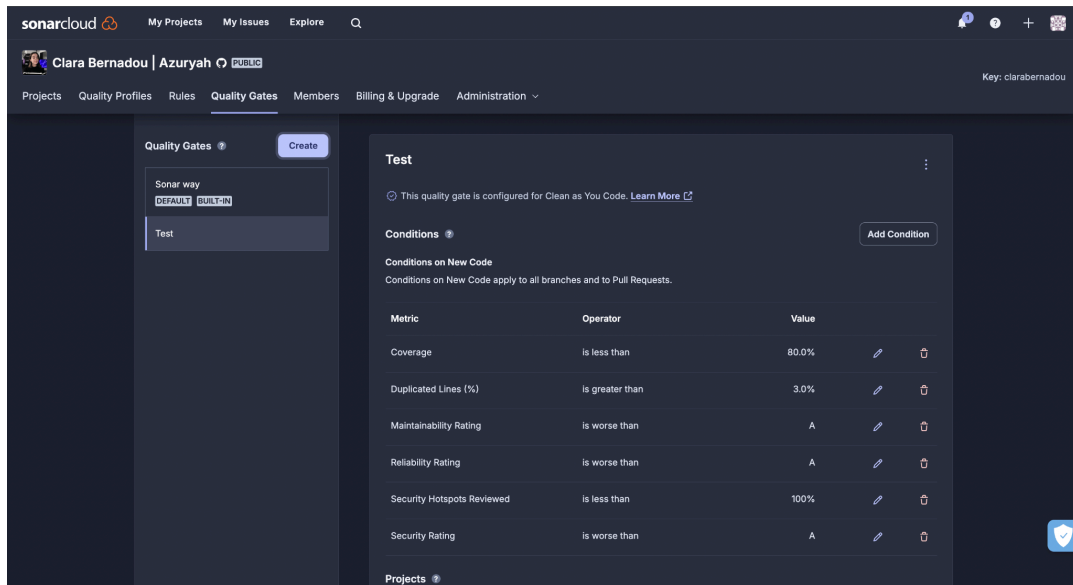
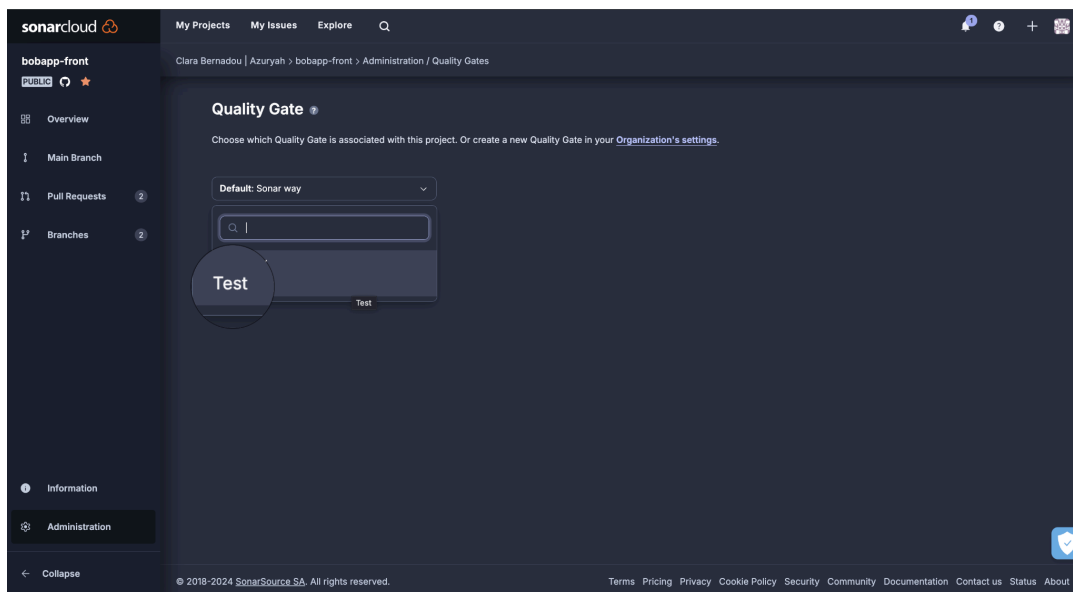
Choose which Quality Gate is associated with this project. Or create a new Quality Gate in your [Organization's settings](#).

Default: Sonar way

Terms Pricing Privacy Cookie Policy Security Community Documentation Contact us Status About

https://sonarcloud.io/organizations/clarabernadou/quality_gates





Analyse des métriques et des avis utilisateurs

1. Métriques

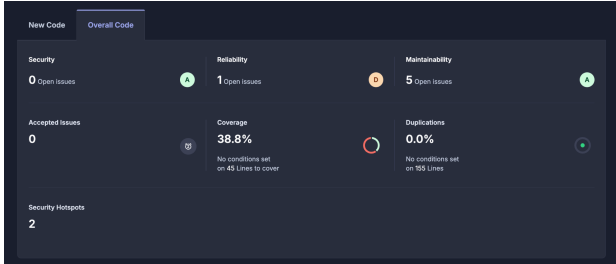
Backend

a. Rapport de couverture

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Cvty	Missed	Lines	Missed	Methods	Missed	Classes
com.oenrclassrooms.bobapo.model		0%		n/a	7	7	13	13	7	7	1	1
com.oenrclassrooms.bobapo.data		49%		50%	5	7	8	18	3	5	1	2
com.oenrclassrooms.bobapo.service		25%		n/a	1	2	4	7	1	2	0	1
com.oenrclassrooms.bobapo.controller		54%		n/a	1	2	1	4	1	2	0	1
com.oenrclassrooms.bobapo		37%		n/a	1	2	2	3	1	2	0	1
Total		90 of 134	32% 2 of 4	50%	15	20	28	45	13	18	2	6

/	Model	Data	Service	Controller	Global	Total
Pourcentage	0%	49%	25%	54%	37%	32%
Commentaire	La couverture est inexistante, ce qui signifie que ce package n'est pas testé. Il est important d'ajouter des tests pour améliorer la fiabilité du modèle.	La couverture est faible, presque la moitié du code n'est pas testée. Des tests supplémentaires sont nécessaires pour garantir la stabilité des opérations sur les données.	Ce niveau de couverture est très bas pour les services, où des tests sont essentiels pour s'assurer que la logique métier fonctionne correctement.	La couverture est modérée, mais il serait bénéfique d'augmenter ce score pour garantir que les contrôleurs répondent comme prévu.	Cette couverture est faible. Des tests globaux sur ce package principal aideraient à assurer une meilleure fiabilité.	La couverture de code totale est faible. Il est essentiel d'augmenter la couverture des tests, en particulier pour les packages model, service, et data, afin de garantir que le code est bien testé et que les fonctionnalités principales sont fiables.

b. SonarCloud



/	Sécurité	Fiabilité	Maintenabilité	Problèmes acceptés	Couverture de code	Duplications de code
Note	A	D	A	0	38.8%	0.0%
Commentaire	Le code ne présente aucune faille de sécurité détectée, assurant une bonne protection des données.	La fiabilité nécessite des améliorations, avec potentiellement des bugs ou du code sujet aux erreurs.	Le code est bien structuré et facile à maintenir, facilitant les évolutions futures.	Aucun hotspot de sécurité n'a été examiné, ce qui pourrait laisser des vulnérabilités non identifiées.	Le taux de couverture est relativement bas. Cela laisse une grande partie du code non testée, augmentant les risques d'erreurs en production.	Absence de duplication dans le code, ce qui améliore la maintenabilité et réduit les risques de bugs.

Frontend

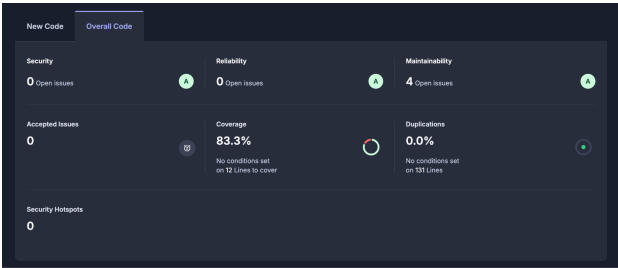
a. Coverage



/	Global	Controller	Total
Pourcentage	60%	87.5%	76.92%
Commentaire	La couverture pour les branches est complète, mais les fonctions et les lignes ont encore des parties	Ce répertoire a une excellente couverture, ce qui indique que les services sont bien testés. Peu	La couverture globale est assez bonne, avec un total de 76.92% pour les statements et une couverture

	non testées. Il est recommandé d'ajouter des tests pour couvrir davantage de fonctions et lignes dans ce répertoire.	d'améliorations sont nécessaires ici, sauf pour atteindre une couverture de 100% des statements.	complète pour les branches. Cependant, les fonctions restent partiellement couvertes (57.14%) et il reste des lignes non testées (83.33%). Il est recommandé de concentrer les efforts sur le répertoire app pour améliorer la couverture des statements et des fonctions.
--	--	--	--

b. SonarCloud



/	Sécurité	Fiabilité	Maintenabilité	Problèmes acceptés	Couverture de code	Duplications de code
Note	A	A	A	0	83.3%	0.0%

Commentaire	Aucun problème de sécurité est détecté, garantissant la sûreté de l'application côté client.	Le code est fiable avec peu de risques de bugs, assurant une bonne robustesse pour les utilisateurs.	Le code est bien structuré et facilement modifiable, facilitant les évolutions futures.	Tous les hotspots de sécurité ont été examinés, garantissant une protection renforcée contre les vulnérabilités potentielles.	Une bonne couverture de tests qui assure la qualité et réduit les risques de régression. Toutefois, viser 100% pourrait encore renforcer la robustesse.	Aucun code dupliqué, ce qui réduit la complexité et les risques d'incohérence.
-------------	--	--	---	---	---	--

2. Analyse des avis utilisateurs et des métriques obtenues



NOTES ET AVIS →

2,0

☆☆☆☆☆

Je mets une étoile car je ne peux pas en mettre zéro ! Impossible de poster une suggestion de blague, le bouton tourne et fait planter mon navigateur !

☆☆☆☆☆

#BobApp j'ai remonté un bug sur le post de vidéo il y a deux semaines et il est encore présent ! Les dees vous faites quoi ????

☆☆☆☆☆

Ca fait une semaine que je ne reçois plus rien, j'ai envoyé un email il y a 5 jours mais toujours pas de nouvelles...

☆☆☆☆☆

J'ai supprimé ce site de mes favoris ce matin, dommage, vraiment dommage.

Avis utilisateurs pertinents et priorités

Problèmes de performance lors de l'ajout de suggestions :

- **Explication** : Un utilisateur signale qu'il est impossible de poster une suggestion de blague, une fonctionnalité pourtant attendue. Après vérification, il apparaît que ni l'application web ni le code ne contiennent cette fonctionnalité. Il peut s'agir d'une régression, d'un retrait volontaire, ou d'une fonctionnalité prévue mais jamais implémentée.
- **Priorité** : Élevée

- **Solution** : Vérifier l'historique des versions afin de déterminer si la fonctionnalité a déjà été développée puis retirée, ou si elle n'a jamais été implémentée. Si cette fonctionnalité est attendue et jugée nécessaire, procéder à son implémentation ou réimplémentation. Une fois développée, il est essentiel de mettre en place des tests unitaires et d'intégration pour garantir son bon fonctionnement et éviter toute régression future.
-

Bug persistant concernant le lecteur vidéo :

- **Explication** : Un utilisateur indique avoir signalé un bug sur le post vidéo il y a deux semaines, mais il est toujours présent. Après vérification, il apparaît que ni l'application web ni le code ne contiennent cette fonctionnalité. Il peut s'agir également d'une régression, d'un retrait volontaire, ou d'une fonctionnalité prévue mais jamais implémentée.
 - **Priorité** : Élevée
 - **Solution** : Vérifier l'historique des versions afin de déterminer si la fonctionnalité a déjà été développée puis retirée, ou si elle n'a jamais été implémentée. Si cette fonctionnalité est attendue et jugée nécessaire, procéder à son implémentation ou réimplémentation. Une fois développée, il est essentiel de mettre en place des tests unitaires et d'intégration pour garantir son bon fonctionnement et éviter toute régression future.
-

Support client lent ou inactif :

- **Explication** : Un utilisateur mentionne un problème de non-réception d'e-mails et déplore l'absence de réponse malgré plusieurs relances (une semaine sans nouvelles).
- **Priorité** : Moyenne
- **Solution** : Améliorer les processus internes du support client, par exemple en mettant en place un système de réponse

automatique pour informer les utilisateurs de la prise en charge de leur demande. Augmenter la fréquence des vérifications de tickets non résolus pour garantir une réactivité accrue. Former l'équipe pour mieux gérer les délais de réponse, surtout pour les cas techniques urgents.

Perte de confiance et désinstallation de l'application :

- **Explication** : Un utilisateur a décidé de supprimer BobApp de ses favoris, exprimant une grande déception face à l'application.
 - **Priorité** : Moyenne
 - **Solution** : Envoyer un sondage ou un e-mail de suivi aux utilisateurs pour comprendre leurs frustrations. Mettre en place un plan d'amélioration basé sur les retours d'expérience, incluant des mises à jour pour corriger les principales plaintes. Communiquer les améliorations apportées pour reconquérir la confiance des utilisateurs et favoriser la réinstallation de l'application.
-

Priorité des métriques à améliorer

Couverture de code insuffisante en backend (38,8%) :

- **Explication** : Le faible niveau de couverture de code augmente les risques de bugs non détectés en production. Il est essentiel d'augmenter cette couverture pour garantir une meilleure stabilité et fiabilité du système.
 - **Priorité** : Élevée
 - **Solution** : Augmenter le nombre de tests unitaires et d'intégration, en ciblant prioritairement les parties critiques du code. Utiliser le rapport de couverture JaCoCo pour identifier les zones non testées et développer des tests supplémentaires.
-

Fiabilité (Note D) :

- **Explication** : La note de fiabilité du backend est alarmante, suggérant des problèmes de performance et de stabilité qui doivent être traités en urgence.
 - **Priorité** : Élevée
 - **Solution** : Réaliser une analyse approfondie des zones à risque identifiées, en particulier celles qui causent des instabilités. Mettre en place une revue de code plus stricte, corriger les points de vulnérabilité et améliorer la gestion des erreurs. Tester le système avec des charges de travail élevées pour évaluer et ajuster la performance.
-

Points d'accès sécurisés (2) :

- **Explication** : Il est crucial d'augmenter le nombre de points d'accès sécurisés pour protéger l'application contre les menaces potentielles.
- **Priorité** : Moyenne
- **Solution** :
 - **Valider les entrées** : Assurez-vous que toutes les requêtes envoyées au backend, même pour des actions simples comme changer de blague, sont validées. Cela inclut la vérification des formats de données et la gestion des erreurs d'entrée pour éviter les injections de code ou les attaques par script.
 - **Configurer les en-têtes de sécurité** : Utiliser des en-têtes HTTP comme Content-Security-Policy, X-Content-Type-Options, et X-Frame-Options pour limiter les risques d'attaques comme le clickjacking ou les injections de scripts.
 - **Limiter les requêtes** : Ajouter une restriction sur la fréquence de changement de blague (par exemple, un

limiteur de requêtes) pour éviter les abus, comme un spam de requêtes au serveur.