DETAILS DU WORKFLOW

Détail des étapes de CI/CD mises en place

Le détail de la pipeline CI/CD via les GitHub Actions comprend les étapes suivantes pour chaque push ou pull request (sur la branche MAIN) :

- Tests Frontend
  - Checkout sur la branche
  - Installation de Node.js
  - Installation de ChromeHeadless
  - Installation des dépendances du projet
  - Build du front
  - Lancement des tests
  - Archivage des rapports et couverture de code
  - Mise à disposition du rapport de tests dans github
  - Affichage de l'URL pour récupérer le rapport

- Tests Backend
  - Checkout sur la branche
  - Configuration de l'environnement avec installation de Java 11
  - Build du back
  - Lancement des tests
  - Génération de rapports de tests
  - Archivage des rapports et couverture de code
  - Mise à disposition du rapport de tests dans github
  - Affichage de l'URL pour récupérer le rapport

- Analyses SonarQube (back)
  - Récupère l'historique git depuis 0
  - Ajout et configure Java 11
  - Lance le build avec test
  - Lance le scan Sonar

- Analyses SonarQube (front)
  - Récupère l'historique git depuis 0

    - Récupère le rapport de couverture
    - Lance le scan Sonar

- Build et publication Docker front
    - Login sur docker hub
    - Build du front
    - Si branche main : Push de l'image sur Docker Hub avec tag basé sur le SHA du commit.

- Build et publication Docker front
    - Login sur docker hub
    - Build du front
    - Si branche main :Push de l'image sur Docker Hub avec tag basé sur le SHA du commit.

Par ailleurs, des règles sont mises en place sur Git hub pour compléter ces pipelines :

- Contrôle qualité de PR
    - Les pull requests sont bloquées si les jobs CI/CD échouent, ce qui garantit une intégration uniquement si les tests et l'analyse qualité passent avec succès.

KPIS PROPOSES

Voici 3 indicateurs de performance pertinents pour controler la qualité de l'application :

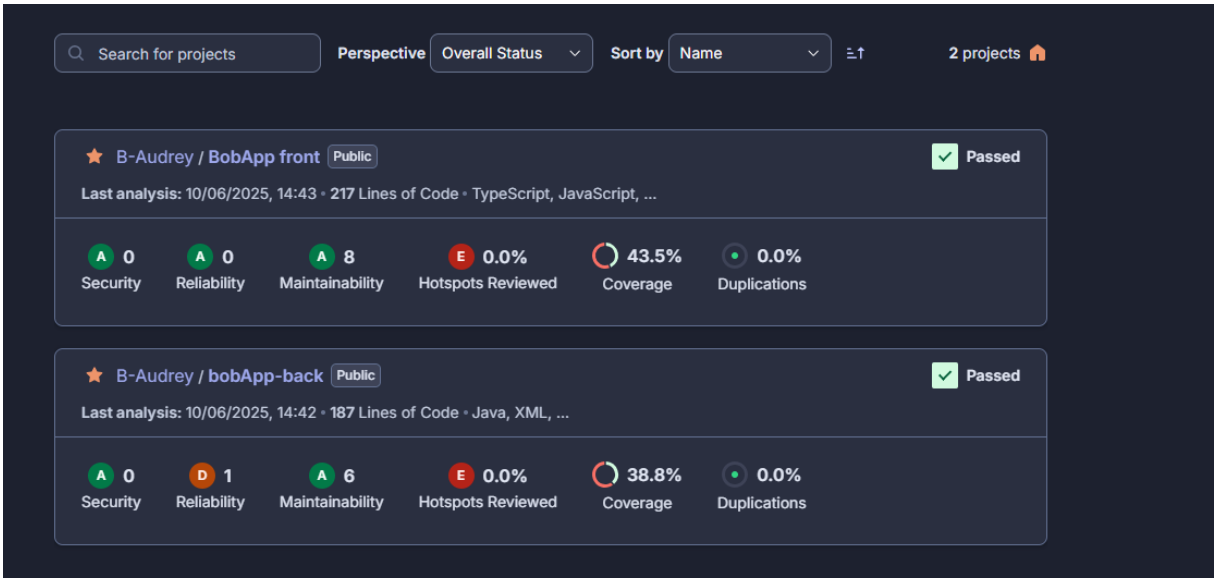
KPI	Seuil à atteindre	Mesure	Valeur actuelle Front	Valeur actuelle back
Taux de couverture des tests	70%	SonarQube	38.8	43.5
Problème de sécurité critique	0	SonarQube	0	0
Nombre de pull request en cours	5 maximum	GitHub	x	x

Ces KPI sont quantifiables et mesurables sur l'interface SonarCloud et GitHub.

- Le taux de couverture de chaque projet sera à maintenir par la personne en charge de la feature afin de ne pas régresser**. L'idéal serait d'atteindre 80% de taux de couverture**. Cependant le projet étant Open Source et ouvert aux pull request, une proposition minimal à 70% me parait être cohérent afin de ne pas limiter les développeurs tout gardant un seuil minimal acceptable.
- **Les failles de sécurité critique sont à bannir**. Aucune introduction de faille de sécurité ne doit apparaitre. Lorsqu'une faille de sécurité apparait, une analyse plus poussée doit être réalisée pour trouver une solution. Ce KPI reste à surveiller afin de garantir l'intégrité de l'application pour les utilisateurs.
- L'exécution des pipelines de CI/CD s'exécute pour les tests et pour Sonar sur les branches de pull request. Il est possible de se fixer un nombre maximal de pull request à traiter afin de ne pas se laisser submerger et d'éviter que les contributeurs prennent des retard sur leurs branches de travail. Un maximum de 5 branches en parallèle dans un premier temps serait cohérent afin d'éviter l'encombrement, préserver la qualité du code et la pertinence des features puis merger le code dans des délais raisonnables. Cela permettra aussi de prioriser les fix de bug avant l'ajout de nouvelles features pour stabiliser l'application et la pérenniser. **L'onglet Insight** nous permets de voir le nombre de pull request actuellement ouvertes sur le projet. En cas de dépassement, il faudra envisager des actions pour peut être limiter, créer une file d'attente ou bien accélérer les rythmes et temps a passer sur le projet.

ANALYSE DES METRIQUES OBTENUES

Voici les métriques obtenues lors que l'analyse du code sur la branche main.



L'analyse est acceptable à date. Néanmoins, certaines actions vont devoir être mises en place afin de stabiliser l'application.

Le **code coverage** est trop faible côté front et côté back, il va être nécessaire de prioriser la rédaction de tests afin d'augmenter le taux de couverture et ainsi limiter, voir corriger les bugs de l'application.

Le code actuel est **maintenable**, **sécurisé** et ne présente pas de **duplications**. La quantité de code étant limité, il faudra surveiller cette métrique au fil de temps.

L'analyse révèle un potentiel **bug** côté backend sur le fichier JokeSerice.java, il faudra donc traiter ce bug potentiel connu afin d'éviter de détériorer le code.

Aucun **code sensible** ne demande à être examiné à ce jour.

Métrique	**Définition**	Valeur actuelle	Analyse
Code coverage	Pourcentage de lignes de code couvertes par des tests automatisés.	Back : 38.8% Front : 43.5%	Ces valeurs sont insuffisantes et révèle des potentiels bugs non couverts par des tests
Maintainability	Mesure la facilité à comprendre, modifier et faire évoluer le code. Calculée en fonction de la taille de la dette technique.	Back : 6 - A Front 8 - A	Ces deux notes sont classées A et sont donc bonnes, la projet étant petit, il faut être vigilent sur la constance de cette note
Security	Nombre de vulnérabilités identifiées (failles potentielles d'exploitation).	0 - A	Aucune faille de sécurité n'a été trouvée, ce qui est un bon point et devra être maintenu.
Reliability	Nombre de bugs identifiés.	Back : 1 - D Front : 0 - A	Un bug potentiel est détecté côté backend. Une correction devra t être envisagée.
Duplications	Pourcentage de code dupliqué dans le projet.	0%	Aucun code n'est actuellement dupliqué. Cette métrique est intéressante sur un projet open source où le projet n'est pas maîtrisé par tous ceux qui y participent.
Hotspots	Portions de code sensibles qui méritent une revue manuelle (non forcément vulnérables).	0%	

CONCLUSION ET PLAN D'ACTION PRIORISÉ

Retour utilisateurs

Les avis clients sont extrêmement négatifs et traduisent des problèmes de qualité produit :

- Blocages critiques de l'interface (UX non fonctionnelle).
- Bugs non résolus malgré des signalements.
- Inactivité perçue par les utilisateurs qui ont tenté de faire remonter des problèmes.

À la suite de ces constat et à l'analyse des métriques obtenues sur la qualité du code, les indicateurs montrent que le manque de tests nuit à la qualité de l'app. Les retours utilisateurs confirment cette urgence.

Voici donc les 3 priorités recommandées :

Priorité	Description	Action
1	Corriger les bugs bloquants	Identifier et corriger les problèmes bloquants signalés : 1. Le navigateur de l'utilisateur qui ne répond pas 2. Le post d'une suggestion impossible 3. Le post de vidéo qui bug
2	Améliorer le code coverage	Viser les 80% sur la base de code existante pour partir d'une base de code stable et limiter les futures dégradations
3	Intégrer une gestion des tickets	Ajouter une solution permettant le suivi des rapport de bugs dans l'application par les utilisateurs pour maintenir le lien avec les utilisateurs.

Ces priorités traduisent les objectifs suivants :

- Résoudre les bugs bloquants pour garder les utilisateurs actuels
- Stabiliser l'application en augmentant la couverture de tests
- Améliorer l'image auprès des utilisateurs en communiquant davantage