Document Explicatif - CI/CD pour BobApp

Introduction

Ce document décrit les étapes que j'ai suivies pour mettre en place la pipeline CI/CD sur GitHub Actions pour le projet BobApp.

Il inclut également une proposition de KPIs pour assurer la qualité continue du projet ainsi qu'une analyse des métriques obtenues suite à l'exécution de la pipeline.

Enfin, je présente une synthèse des retours utilisateurs pour identifier les actions à mener en priorité.

Étapes des GitHub Actions

1. Job Backend

1.1 Checkout du Code

- Objectif : Récupérer le code source du projet à partir du repository GitHub.
- Étape : Utilisation de l'action actions/checkout@v3 pour cloner le repository afin de préparer l'exécution des étapes suivantes.

1.2 Setup Environnement Backend

- Objectif : Préparer l'environnement pour le backend, spécifiquement pour exécuter les tests et les builds sur une version Java compatible (Java 21 dans ce cas).
- Étape : Utilisation de l'action actions/setup-java@v3 avec la configuration de la version 21 du JDK.

1.3 Build et Exécution des Tests Backend

- Objectif : Compiler le backend et exécuter les tests pour vérifier que tout fonctionne correctement.
- Étape : Utilisation de la commande Maven mvn clean verify dans le répertoire ./back pour compiler et exécuter les tests.

1.4 Génération du Rapport de Couverture Backend

- Objectif : Générer un rapport de couverture de code à l'aide de JaCoCo.
- Étape : Exécution de la commande mvn jacoco:report pour générer le rapport de couverture.

1.5 Analyse SonarCloud Backend

- Objectif : Utiliser SonarCloud pour analyser la qualité du code du backend et détecter d'éventuelles failles ou problèmes.
- Étape : Exécution de la commande Maven sonar-maven-plugin avec les tokens appropriés pour effectuer une analyse SonarCloud.

1.6 Build et Push de l'Image Docker Backend

- Objectif : Créer une image Docker pour le backend et la pousser vers Docker Hub.
- Étape : Utilisation de Docker pour builder l'image, suivi d'une connexion à Docker Hub via docker/login-action@v3 et un push de l'image backend.

2. Job Frontend

2.1 Checkout du Code

- Objectif : Récupérer le code source du projet à partir du repository GitHub.
- Étape : Utilisation de l'action actions/checkout@v3 pour cloner le repository afin de préparer l'exécution des étapes suivantes.

2.2 Setup Environnement Frontend

- Objectif : Préparer l'environnement pour le frontend avec Node.js.
- Étape : Utilisation de l'action actions/setup-node@v4 pour configurer Node.js.

2.3 Exécution des Tests Frontend

- Objectif : Exécuter les tests du frontend en générant également un rapport de couverture de code.
- Étape : Exécution de la commande npm run test:prod -- --code-coverage pour générer les résultats des tests et la couverture.

2.4 Analyse SonarCloud Frontend

- Objectif : Analyser la qualité du code frontend avec SonarCloud.
- Étape : Utilisation de l'action SonarSource/sonarcloud-github-action@master avec les arguments spécifiques pour scanner le frontend.

2.5 Build et Push de l'Image Docker Frontend

- Objectif : Créer une image Docker pour le frontend et la pousser vers Docker Hub.
- Étape : Build de l'image Docker frontend et push vers Docker Hub.

Propositions de KPIs

1. Code Coverage (Couverture du Code)
    - Objectif : S'assurer que les tests couvrent une partie significative du code afin de réduire les risques de bugs non détectés.
    - Seuil Proposé : 80% de couverture de code pour commencer, avec un objectif à long terme de 90%.

2. Duplication de Code
    - Objectif : Maintenir une duplication de code proche de 0% pour améliorer la maintenabilité du projet et réduire le risque de bugs provenant de morceaux de code dupliqués.
    - Seuil Proposé : Duplication de code à un maximum de 3%, avec un objectif actuel de maintenir cette duplication à 0%, comme le montrent les résultats actuels.

Explication des Blocker Issues

Les "Blocker Issues" sont des problèmes critiques identifiés par SonarQube qui doivent être corrigés immédiatement.

Il y a 4 type de gravités pour une issue dans SonarQube : Blocker / Critical / Major / Minor

Les blocker issues peuvent rendre le système vulnérable, causer des dysfonctionnements graves, ou entraîner des dégradations de performance significatives.

En d'autres termes, une "Blocker Issue" est un problème bloquant qui doit être résolu avant tout autre développement pour éviter des impacts majeurs sur l'application.

Il est possible de configurer GitHub Actions pour empêcher le merge d'une pull request (PR) si des "Blocker Issues" sont détectées lors de l'analyse SonarQube. Cela permet de garantir que ces problèmes critiques sont corrigés avant que le code ne soit intégré à la branche principale, ce qui contribue à améliorer la stabilité et la sécurité de l'application.

Analyse des Métriques

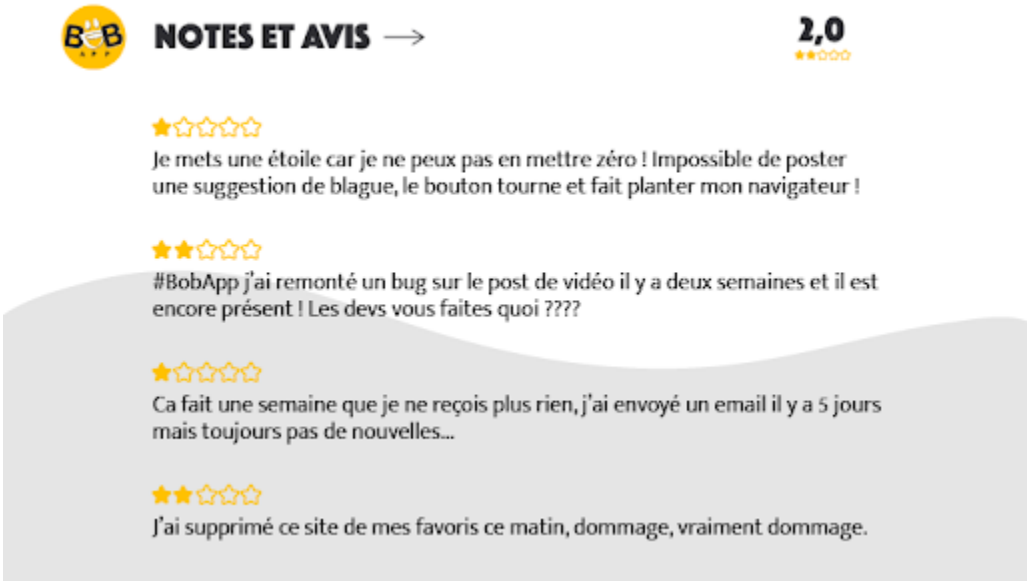
Après exécution de la pipeline CI/CD, voici les premières métriques obtenues :

- Couverture Backend : 38.8% (objectif de 80%)
- Couverture Frontend : 66.7% (objectif de 80%)
- Duplication de Code Backend : 0% (objectif de <3%)
- Duplication de Code Frontend : 0% (objectif de <3%)

Ces résultats montrent que la duplication de code est déjà optimisée et doit être maintenue à ce niveau.

Cependant, il y a encore du travail à faire pour améliorer la couverture des tests.

Analyse des Retours Utilisateurs



Les commentaires des utilisateurs montrent des problèmes récurrents sur l'application :

1. Bug du bouton de suggestion de blague : Ce problème semble provoquer des crashs dans le navigateur des utilisateurs. Priorité haute pour corriger ce bug.
2. Problème de réponse tardive des équipes de développement : Plusieurs utilisateurs se plaignent d'un manque de suivi après avoir signalé des bugs.
3. Dégradation de l'expérience utilisateur : Les utilisateurs expriment leur frustration face aux bugs non corrigés et à l'instabilité générale de l'application.

Recommandations

1. Correction immédiate des bugs critiques : Le problème du bouton de suggestion doit être résolu en priorité.

2. Augmenter la couverture des tests : Atteindre l'objectif de 80% de couverture minimum pour garantir une meilleure stabilité du code.

3. Maintenir la duplication de code à 0% : Travailler à maintenir cette bonne pratique, ce qui améliorera la maintenabilité du projet et réduira les risques d'erreurs.

4. Améliorer le suivi des retours utilisateurs : Mettre en place un système de suivi plus rigoureux pour s'assurer que les bugs signalés sont rapidement pris en charge.

5. Automatiser le processus de déploiement : La CI/CD mise en place permet déjà d'automatiser une partie du déploiement, mais une surveillance active des déploiements en production est nécessaire pour éviter des régressions.

Conclusion

La mise en place de cette CI/CD pipeline sur GitHub Actions permettra d'améliorer l'efficacité des déploiements et de la gestion des bugs sur BobApp.

Avec l'amélioration continue de la couverture des tests, le maintien de la duplication de code à 0%, et la correction des issues critiques, la qualité générale de l'application devrait s'améliorer, ce qui se reflétera dans de meilleurs avis utilisateurs.