Document explicatif de l’application BobApp

SOMMAIRE

Documentation technique – Workflow CI/CD BobApp

L'intégration continue (CI) et le déploiement continu (CD) mis en place sur BobApp ont pour objectif de garantir la qualité du code, de sécuriser les évolutions de l'application et de simplifier le déploiement grâce à l'automatisation complète du cycle de build, de test, d'analyse de qualité et de publication des images Docker.

Lorsqu'une modification est poussée sur la branche principale ou lorsqu'une pull request est ouverte, le workflow orchestrateur se déclenche automatiquement. Ce workflow joue le rôle de chef d'orchestre et déclenche successivement trois sous-workflows, chacun ayant une responsabilité précise.

La première étape concerne l'exécution des tests unitaires et la génération des rapports de couverture de code, tant pour le backend que pour le frontend. Pour la partie backend, le job dédié s'exécute dans un environnement Ubuntu. Le code est d'abord récupéré via l'action officielle GitHub permettant de cloner le dépôt. Ensuite, l'environnement Java 11 est installé grâce à l'action setup-java afin de disposer de la version nécessaire pour compiler le projet Spring Boot. Les tests unitaires sont ensuite lancés avec Maven via la commande mvn clean test. Cette commande génère également le rapport de couverture JaCoCo, qui est ensuite publié en tant qu'artefact afin d'être consulté plus tard. Côté frontend, un autre job similaire est prévu. Après récupération du code, l'environnement Node.js version 18 est configuré. Les dépendances Angular sont installées via la commande npm install. Les tests unitaires Angular sont ensuite exécutés en mode headless via Chrome grâce à la commande npm run test avec les options permettant de désactiver le watch et de générer le rapport de couverture. Ce dernier est également archivé comme artefact. Cette étape de tests est cruciale, car elle permet de valider la non-régression du code et d'obtenir des indicateurs chiffrés sur la couverture de test.

Une fois la phase de tests et de couverture terminée, le workflow SonarQube est lancé. Il s'appuie sur SonarCloud afin de réaliser une analyse approfondie de la qualité du code des deux projets, backend et frontend. Ce workflow récupère d'abord l'ensemble du code source, en veillant à cloner l'historique complet du dépôt afin de permettre une analyse sur plusieurs commits. L'environnement Node.js est configuré à nouveau pour le frontend, puis les dépendances sont installées. Les tests unitaires du frontend sont relancés dans ce workflow, car SonarCloud se base sur les rapports de couverture générés pour établir ses métriques. L'environnement Java 11 est ensuite mis en place, et les dépendances Maven

sont mises en cache pour optimiser les temps d'exécution. Le projet backend est compilé et vérifié via Maven, ce qui exécute également les tests. Enfin, l'analyse SonarCloud est exécutée via l'action officielle SonarSource. Cette étape a pour objectif de mesurer la qualité du code au travers de nombreux indicateurs tels que la couverture, la dette technique, la complexité ou encore la présence de bugs et de vulnérabilités. L'analyse est ensuite consultable sur SonarCloud, permettant de vérifier si le projet respecte ou non les Quality Gates définies.

Lorsque l'analyse SonarCloud est terminée et validée, le workflow de publication des images Docker entre en jeu. Celui-ci se charge de construire et de publier les images Docker pour le backend et le frontend sur Docker Hub. Pour chaque partie de l'application, le code est d'abord récupéré. Ensuite, la commande docker build permet de construire une image Docker taguée avec le nom du projet et le tag latest. L'authentification à Docker Hub est réalisée de façon sécurisée en injectant le login et le mot de passe stockés sous forme de secrets GitHub. Une fois connecté, chaque image est poussée sur le registre Docker Hub grâce à la commande docker push. Cette étape garantit la disponibilité des conteneurs de l'application pour être déployés facilement sur n'importe quel environnement de production ou de test.

Ce processus CI/CD complet assure ainsi que tout changement apporté au code de BobApp est testé, vérifié pour sa qualité, mesuré en termes de couverture, et déployable de manière automatisée via des conteneurs Docker publiés sur Docker Hub. Cela réduit considérablement les risques liés aux erreurs manuelles, augmente la fiabilité du logiciel et libère du temps pour le développement de nouvelles fonctionnalités.

KPI n°1 Couverture de code minimale

La couverture de code est un indicateur essentiel qui mesure le pourcentage de lignes de code réellement exécutées lors des tests automatiques. Plus ce taux est élevé, plus on a de chances de détecter des anomalies avant la mise en production. Pour le projet BobApp, l'objectif est de fixer une couverture minimale à 80 %.

Actuellement, la couverture globale du projet atteint 48,1 %. Il est donc nécessaire d'augmenter significativement le nombre de tests, notamment côté backend où la couverture est de 47,4 %, contre 50,0 % pour le frontend. Cet objectif permettra de sécuriser davantage l'application et d'éviter que des bugs non détectés ne provoquent des incidents chez les utilisateurs.

KPI n°2 Limitation des problèmes de maintenabilité

Les "code smells" correspondent à des problèmes de maintenabilité détectés par SonarCloud. Il peut s'agir de portions de code trop complexes, redondantes ou difficiles à comprendre, ce qui peut ralentir la correction des bugs ou l'ajout de nouvelles fonctionnalités.

L'objectif fixé est de limiter le nombre de nouveaux "code smells" à un maximum de dix par analyse. Actuellement, onze problèmes de maintenabilité sont recensés sur le projet, répartis entre six sur le backend et cinq sur le frontend. Ce KPI a pour but de limiter l'augmentation de la dette technique, de préserver la lisibilité du code et de faciliter le travail des développeurs.

Analyse des métriques et des retours utilisateurs

Suite à l'exécution de la pipeline CI/CD sur BobApp, les métriques issues de SonarCloud permettent de dresser un état précis de la qualité actuelle du projet. La couverture globale atteint 48,1 %, ce qui reste insuffisant au regard de l'objectif de 80 %. La couverture est légèrement plus élevée sur le frontend (50,0 %) que sur le backend (47,4 %). Aucun problème critique de type "blocker issue" n'a été relevé sur la dernière analyse SonarCloud, ce qui est positif. La duplication de code est inexistante. Deux Security Hotspots ont cependant été identifiés et nécessitent une analyse approfondie pour s'assurer qu'ils ne constituent pas de vulnérabilités réelles. Enfin, la maintenabilité présente onze problèmes ouverts, confirmant la nécessité de poursuivre les efforts de refactoring pour simplifier le code et réduire sa complexité.

Notes et avis des utilisateurs

Les retours d'utilisateurs révèlent plusieurs dysfonctionnements importants, en cohérence avec certaines faiblesses techniques détectées. Deux problèmes ressortent particulièrement :

- Impossibilité de poster une suggestion de blague : le bouton dédié déclenche un chargement infini et fait planter le navigateur, bloquant ainsi une fonctionnalité clé de l'application.

- Bug persistant sur la publication de vidéos : signalé depuis plus de deux semaines et toujours non résolu, traduisant un manque de réactivité dans la correction des anomalies.

Problèmes prioritaires à résoudre

Les priorités identifiées suite à cette analyse sont les suivantes :

- Augmenter la couverture des tests pour atteindre progressivement le KPI fixé à 80 %, en se concentrant notamment sur le backend.

- Limiter l'apparition de nouveaux problèmes de maintenabilité (code smells) afin de garantir un code plus fiable et plus facile à maintenir.

- Corriger rapidement les bugs bloquants signalés par les utilisateurs, en particulier le bug sur la soumission de blagues et celui lié à la publication de vidéos.

- Analyser et résoudre les Security Hotspots identifiés afin de garantir la sécurité de l'application.