GitHub ACTIONS :

DEPLOIEMENT DE L’INTEGRATION CONTINU

Table des matières

1 Contexte

Ce document à pour but de retracer l'ensemble des modifications effectuées au sein du projet.

Nous aborderons dans les parties suivantes les chapitres relatifs à la mis en place d'une intégration continue au sein de GitHub, à l'implémentation des KPIs et enfin à une analyse des métriques obtenus.

2 Mise en place de l'intégration continu

Cette rédaction sera séparée en trois sous-chapitres à savoir l'implémentation de la création des images dockers, le déroulement des tests, ainsi que la configuration de SonarCloud au sein du projet.

2.1 Création des images Docker

Nous pouvons donc commencer avec la création des images Docker tributaires des deux composantes de notre application à savoir le front-end et le back-end.

Ces deux éléments nécessitent une configuration commune que nous allons traiter par la suite.

2.1.1 Configuration

Cette configuration est destiné à GitHub.

C'est cette dernière qui nous permettra de dérouler les générations d'images dans l'environnement attendu.

Pour permettre à notre fichier YAML de pouvoir téléverser sur le répertoire distant les images créées, il est requis de paramétrer deux **repository secret**. Ces deux éléments constituent chacun une variable globale à tous les fichiers YAML, qui contient une chaîne de caractère secrète telle qu'un token.

Pour cela, il est nécessaire de naviguer dans la configuration du projet c'est-à-dire de sélectionner l'onglet **Settings**, de cliquer sur **Secrets and variables** et enfin sur **Actions**.

Dans ce menu il est nécessaire de créer les deux variables ci-dessous pour permettre aux GitHub Actions d'utiliser le compte propriétaire du projet :

- **DOCKER_USERNAME** : variable contenant le nom d'utilisateur
- **DOCKER_PASSWORD** : variable contenant le mot de passe utilisateur

Ces deux variables sont relatives à DockerHub qui sera le répertoire distant stockant les images que nous aurons fabriqués.

Sa configuration est simple dans la mesure où il est possible de lier le compte GitHub avec le compte DockerHub.

De ces éléments de configuration communs il nous est désormais possible de traiter la création de nos deux images Docker dans la partie suivante.

2.1.2 Back-end & Front-end

Nous allons maintenant traiter de la création des images Docker du back-end et du front-end.

L'ensemble des traitements sont recensés dans les fichiers **docker-image-back-end** et **docker-image-front-end**. Nous allons fournir des explications quant aux fonctionnalités utilisés dans ces scripts.

2.1.2.1 Checkout

Cette fonctionnalité nous permet de récupérer le code de la branche référencée en tête de fichier à savoir ici **main**.

2.1.2.2 Setup buildx action

Cette action va créer et démarrer un conteneur nous permettant de fabriquer notre image.

2.1.2.3 Login action

Fonction utilisée pour se connecter au compte relié aux informations contenues dans les deux **actions secrets** configurés dans la partie 2.1.1.

2.1.2.4 Build push action

Code permettant de compiler et téléverser notre image fabriquée dans le répertoire distant de DockerHub.

Nous venons de voir l'ensemble des commandes utilisées dans la construction de nos images Docker.

Nous allons maintenant traiter des fichiers relatifs aux test du code.

2.2 Test du code

A travers ce chapitre nous allons aborder les différentes fonctionnalités utilisées pour mener à bien l'ensemble des tests sur nos deux composantes à savoir le back-end et le front-end.

Certains ont déjà été évoqués dans la partie 2.1.2, nous traiterons donc uniquement les fonctionnalités différentes.

2.2.1 Back

L'ensemble des traitements est recensé dans le fichier **test-docker-image-back-end**.

2.2.1.1 Setup java

Cette fonction nous permet d'installer dans le container une version particulière de java.

2.2.1.2 La fonctionnalité run

Cette fonctionnalité nous permet d'écrire et d'exécuter des commandes que l'on pourrait utiliser dans un terminal.

2.2.1.3 Jacoco report

Java Code Coverage (JaCoCo) nous permet de calculer le pourcentage de couverture de nos tests sur le code de l'application.

Nous venons de voir les nouvelles fonctionnalités côté back-end.

Nous allons maintenant nous intéresser à celle du front-end.

2.2.2 Front

L'ensemble des traitements est recensé dans le fichier **test-docker-image-front-end**.

2.2.2.1 Setup node

Cette appellation nous permet d'installer le gestionnaire de dépendances **node** dans le container.

Nous venons de voir l'ensemble des nouvelles fonctionnalités employées pour la mise en œuvre des tests back-end et front-end.

Nous allons désormais nous intéresser à l'implémentation de SonarCloud.

2.3 Implémentation de SonarCloud

Dans ce chapitre nous allons traiter l'implémentation de SonarCloud.

Tout d'abord SonarCloud est une plateforme en ligne analysant le code permettant d'en sortir des statistiques dont nous parlerons dans la partie 4.

2.3.1 Configuration

Il existe une configuration commune à nos deux composantes principales.

Pour interagir avec la plateforme en ligne nous allons avoir besoin de configurer un token d'accès.

De la même manière que DockerHub, SonarCloud nous permet de lier notre compte GitHub ce qui facilite la démarche d'implémentation.

Une fois cela fait, il est possible de générer notre token depuis l'interface de SonarCloud en cliquant sur l'**icône du compte utilisateur**, puis sur l'onglet **Security** et enfin dans la section **Generate token**, après avoir renseigné un nom tel que **GitHub actions** il est nécessaire de cliquer sur **Generate token**.

A ce stade il nous est possible de copier le token qui ne sera plus consultable à l'avenir.

De retour sur GitHub il est nécessaire de naviguer dans la configuration du projet c'est-à-dire de sélectionner l'onglet **Settings**, de cliquer sur **Secrets and variables** et enfin sur **Actions**.

La création de **SONAR_TOKEN** avec le contenu copié depuis SonarCloud est maintenant possible.

Nous venons de voir la configuration commune à nous deux composantes de l'application.

Nous allons maintenant aborder les fonctionnalités qu'elles arborent.

2.3.2 Back

L’ensemble des traitements est recensé dans le fichier **sonar cloud back-end analysis**.

2.3.2.1 Cache

Cette action permet de mettre en cache des dépendances dans le but d’améliorer le temps d’exécution du flux de travail, c’est-à-dire de l’ensemble des commandes à effectuer.

2.3.2.2 SonarCloud scan configuration

Pour permettre à la plateforme d’effectuer le scan il est nécessaire d’ajouter un fichier **sonar-project.properties** au sein du projet.

Ce dernier contient les éléments **sonar.organization** et **sonar.projectKey** qui sont deux variables nécessaires dans l’utilisation de SonarCloud scan dans le fichier **sonar cloud back-end analysis**.

Ces deux dernières correspondent respectivement au **nom de l’utilisateur en minuscule** et au **NomUtilisateur_NomDuProjet.**

2.3.3 Front

Côté front-end il n’y a qu’une seule nouvelle fonctionnalité.

2.3.3.1 SonarCloud github action

Cette action est complémentaire de celle du back-end puisqu’elle vient agrémenter les données précédemment analysées.

Nous venons de voir l’ensemble des nouvelles fonctionnalités employées.

Nous allons maintenant aborder les différents KPIs implémentés.

3 Les KPIs implémentés

Dans cette partie nous allons traiter les différents KPIs que l'on peut retrouver la gestion de configuration de l'application.

Nous parlerons dans un premier temps des KPIs JaCoCo et ensuite des KPI SonarCloud.

3.1 KPI JaCoCo

Tout d'abord un KPI est en français un indicateur clé de perfomance.

Parmis ces indicateurs on retrouve ceux que l'on a paramétré au sein de la configuration de JaCoCo dans le fichier **test-docker-image-back-end**.

Par défaut il dispose de deux indicateurs :

- Coverage-overall : paramètre qui tient compte de l'ensemble du code
- Coverage-changed-file : paramètre qui ne tient uniquement compte que des fichiers modifiés.

Ces deux éléments sont positionnés à 80 % dans le but d'obtenir une couverture convenable par nos tests pour le code de l'application.

Ces deux derniers ont été mis en place lors de la configuration JaCoCo.

Ce ne sont pas les seuls que l'on peut trouver dans l'environnement de l'application, SonarCloud en incorpore aussi et c'est ce que nous allons voir dans la partie suivante.

3.2 KPI SonarCloud

Via les deux scans effectués dans les deux fichiers relatifs à SonarCloud, la plateforme affiche différents éléments dont des KPIs tel que le **pourcentage de couverture** global de l'application ou encore le pourcentage des **aspects de sécurité** abordés.

Nous venons de voir quelques-uns des KPIs que l'on peux trouver sur l'application mais aussi sur SonarCloud.

Nous allons maintenant terminer avec la quatrième partie relative à l'analyse des métriques obtenus.

4 Analyse des métriques obtenus

Dans cette partie nous allons voir l'interface que nous permet de visualiser la plateforme SonarCloud.

Le premier onglet est le suivant :

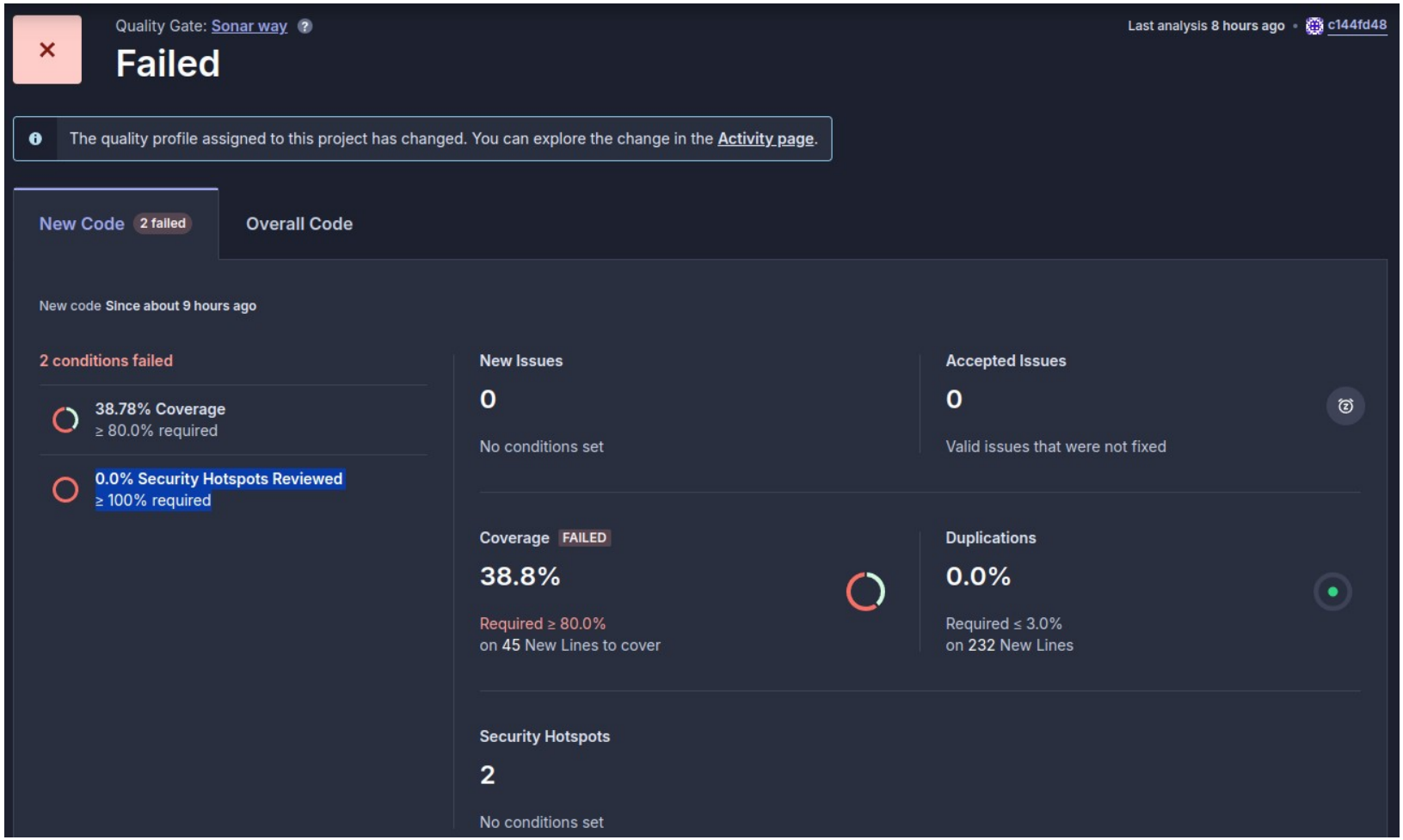

Figure 1: Interface SonarCloud new code

Voici le second onglet :

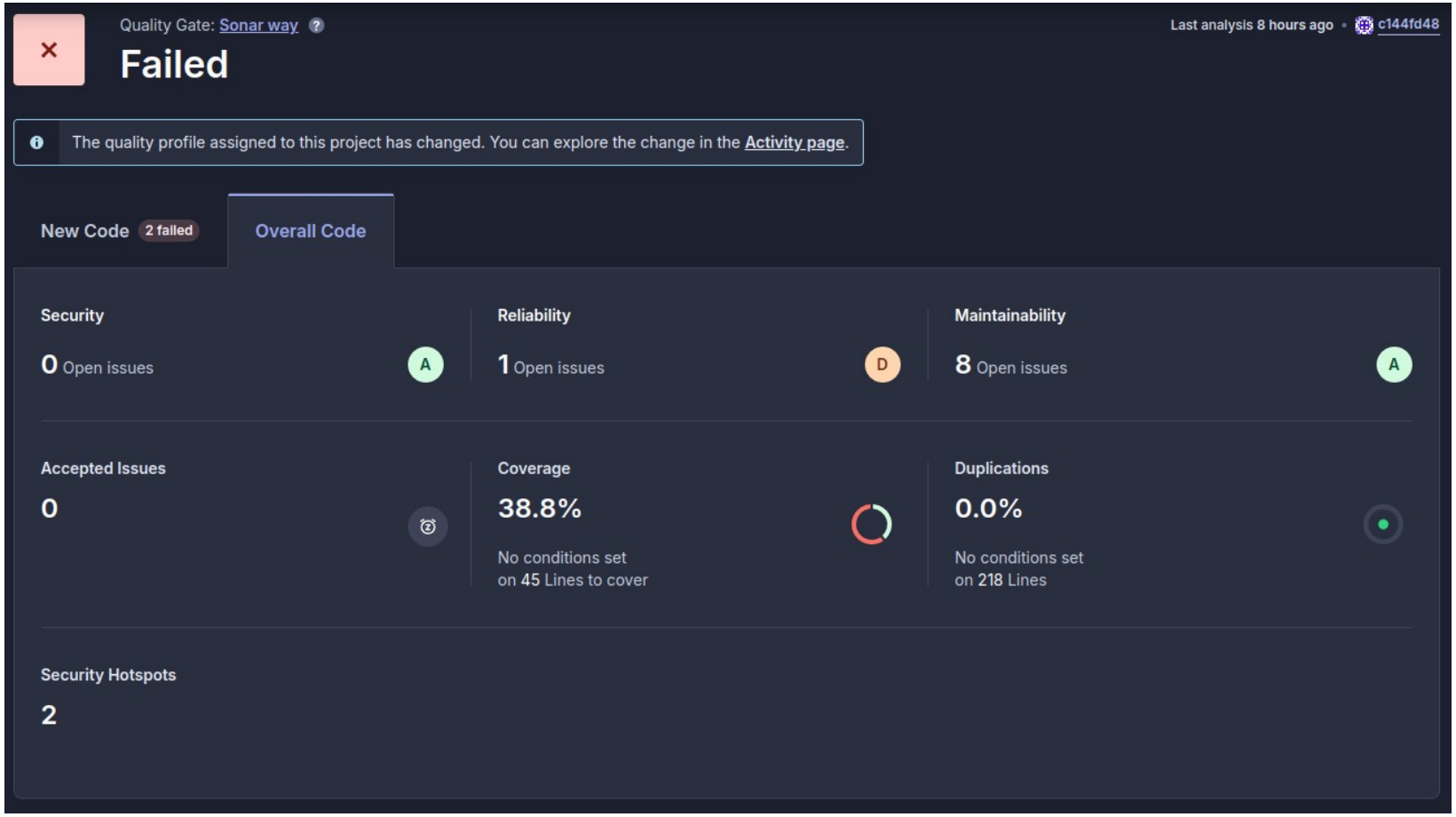

Figure 2: Interface SonarCloud overall code

Comme on peut voir sur la **Figure 1: Interface SonarCloud new code**, deux conditions n'ont pas été remplies.

Les éléments mis à disposition de l'utilisateur sont spécifiques aux morceaux de code qui ont évolué par rapport à l'application au moment de sa création. C'est le même paramètre que l'on retrouve dans la partie **KPI JaCoCo** avec coverage-changed-file.

Le coverage à 38,78 % est inférieur à la règle de 80 %. Ce pourcentage élevé permet d'atteindre un niveau de qualité de code convenable.

De plus deux **security hotspots** ne sont pas couverts. Ici pas de place pour le risque informatique puisque le pourcentage attendu est de 100 %.

La **Figure 2: Interface SonarCloud overall code** quant à elle nous permet de voir des éléments plus généraux.

On y retrouve les précédents évoqués dans le paragraphe ci-dessus mais aussi de nouveaux tels que **Security**, **Reliability** et **Maintainability**.

La première est noté **A** car il n'y a pas de ticket de sécurité ouvert pour le moment. Si on fait coïncider cette donnée avec le coverage des hotspots de sécurité, la note actuelle relève de la chance.

Le second est noté **D**, cela signifie qu'il y a au moins un ticket aillant un impact majeur sur une fonctionnalité de l'application.

Enfin, le troisième paramètre, noté **A**, contient l'ensemble des tickets ouverts aillant un impact moins important.

L'ensemble des ces métriques nous expose les différents manquements quant à la fiabilité de l'application, son pourcentage de coverage et sa sécurité.

La démarche CI mise en place au sein de GitHub permettra d'apporter des correctifs plus aisément et plus rapidement. Il sera de manière générale plus confortable d'effectuer des modifications au sein de l'application.