Mise en place CI/CD via des github actions

Diffusion : OpenClassrooms – Mentor : Cesare de Padura - Evaluateur :

Version	Date	Auteur	Motif
1		Christophe Pierrès	Création du document
2			

Contenu ***Page***

1. Vue générale 7

1.1. Contexte 7
1.2. Pourquoi les github actions ? 7
1.3. Travail effectué, décrit en 2 temps 7
1.3.1. Partie 1 : validation et contrôle qualité avec backend-ci.yml et frontend-ci.yml 8
1.3.1.1. Diagramme de séquence simplifié (partie 1) 8
1.3.1.2. Deux workflows de « github actions » séparés (un par couche) 9
Détail du workflow frontend 10
Détail des steps d'un job (par exemple frontend-sonar) 10
Détail du workflow backend 11
Détail des steps du job backend-tests 11
Détail des steps du job backend-sonar 12
Détail des steps du job backend-docker 13
Détail job annexe « backend-coverage » généré par step « JaCoCo Code Coverage Report » 13
Détail d'une Pull Request 14
Process d'une Pull request 15
1.3.1.3. Deux projets SonarQube séparés (un par couche également) 15
1.3.2. Partie 2 : complèment avec gestion des release (Release-please) 16
1.3.2.1. Diagramme de séquence intégrant les release-please (partie 2) 16
1.3.2.2. Principe de fonctionnement de release-please 17
1.3.2.3. Point de départ (exemple avec frontend) 17
Dernière version de releases/tags 17
PR de release générée par release-please suite à un précédente validation de PR 17
1.3.2.4. Création d'une feature 18
Convertir le push de la branche en PR 18
Checks terminés avec commentaire ajouté pour l'image docker « pr » (mail expédié) 20
1.3.2.5. L'approbateur valide/merge la PR 21
1.3.2.6. frontend-ci est automatiquement déclenché sur le main puis Release Please Frontend 22
1.3.2.7. Release-please a automatiquement adapté la version dans la PR de release 22
Détail de la PR de release (en statut autorelease :pending) ; vue CHANGELOG.md 23
Une confirmation est demandée (possibilité de modifier le message de commit par défaut si besoin) 25
Nouveau cycle de frontend-ci et de Release Please Frontend sur main26
Noter qu'il n'y a plus de PR de release (une nouvelle sera générée par Release Please Frontend lors de la première PR avec libellé de commit conventionnel) 26
Tag et release créés 27
La PR de release a généré un commit pour bumper la version et mettre à jour CHANGELOG dans main (autorelease : tagged) 27
Commentaires pour documenter où se trouvent la release et les images docker 28
Fichiers modifiés par le commit de release généré par la PR de release 30
Images docker créées pour bobapp-frontend 30
1.3.2.8. Détail des jobs du workflow release-please-frontend.yml 32
1.3.3. Reste à faire 32
1.3.3.1. Nettoyage des images pr-* une fois que la PR de feature ou fix est fermée/mergée 32
1.3.3.2. Boucle de rétroaction avec les utilisateurs de bobapp 32

2. Protection du dépôt Github 34

2.1. Vue d'ensemble de la stratégie de protection 34
2.2. Partie 1 : Le Fichier CODEOWNERS 34

2.2.1. Configuration ... 34
2.2.2. Fonctionnement avec les Pull Requests ... 35
2.2.3. Avantages de CODEOWNERS ... 35
2.2.4. Limitations de CODEOWNERS ... 35
2.3. Partie 2 : Protection de Branches ... 36
2.3.1. Comment configurer les protections de branches ... 36
2.3.2. Configurations effectuées ... 36
2.3.2.1. Pour la branche `main` ... 36
2.3.2.2. Pour les branches `workflow-test-*` ... 36
2.4. Partie 3 : Complémentarité des Mécanismes ... 37
2.4.1. Comment CODEOWNERS et la protection de branches se complètent ... 37
2.4.2. Scénarios de protection ... 37
2.5. Partie 4 : Considérations de Sécurité Spécifiques ... 37
2.5.1. Sécurisation des branches `workflow-test-*` ... 37
2.5.2. Risques et mitigations ... 38
2.6. Partie 5 : Recommandations d'évolutions pour BobApp ... 38
2.7. Conclusion ... 39

3. Architecture et structure ... 40

3.1. Architecture de la solution avec SonarQube community edition ... 40
3.1.1. Adressage SonarQube ... 40
3.2. Description générale des workflows de github actions et projets SonarQube ... 41
3.2.1. Structure de base d'un workflow de github actions ... 41
3.2.2. Structure des Workflows de github actions ... 41
3.2.2.1. Backend CI Workflow (`.github/workflows/backend-ci.yml`) ... 41
3.2.2.2. Frontend CI Workflow (`.github/workflows/frontend-ci.yml`) ... 42
3.2.3. Déclencheurs des Workflows ... 42
3.2.4. Structure des Projets SonarQube correspondants ... 43
3.2.5. Avantages de la Séparation en deux Workflows ... 43
3.2.6. Avantages Spécifiques de la Séparation des Projets SonarQube ... 44
3.2.1. Attention particulière au respect de l'environnement précis de chacun des projets 45
3.2.1.1. Pourquoi respecter les environnements spécifiques est crucial ... 45
3.2.1.2. Impact sur la structuration des jobs et le partage d'artefacts ... 46
3.2.1.3. Ce qu'il ne faudrait pas faire dans un souci de simplification ... 46
Ne pas compiler et tester le backend avec JDK 17 simplement parce qu'il est requis par SonarQube ... 47
Ne pas simplifier le workflow frontend de manière similaire ... 47
3.2.1.4. Conclusion ... 47
3.3. Fichiers docker-compose présents dans le projet ... 48
3.3.1. Vue d'ensemble ... 48
3.3.2. docker-compose.yml ... 48
3.3.3. docker-compose-prebuilt.yml ... 49
3.3.4. docker-compose-sonar.yml ... 49
3.3.5. Informations générales ... 50

4. Contrôle qualité avec SonarQube ... 51

4.1. Intérêt des Quality Gates de SonarQube ... 51
4.1.1. Qu'est-ce qu'une Quality Gate ? ... 51
4.1.2. Avantages des Quality Gates ... 51
4.2. Métriques clés des Quality Gates ... 51
4.2.1. Taux de couverture (Coverage) ... 51
4.2.2. New Blocker severity Issues (Nouveaux problèmes bloquants) ... 52
4.2.3. Autres métriques importantes ... 52
4.3. Quality Gates recommandées pour BobApp ... 53
4.3.1. Stratégie différenciée et équilibrée ... 53
4.3.1.1. Avantages spécifiques de cette approche pour le projet BobApp ... 53
4.3.1.2. Stratégie d'évolution des seuils ... 54
4.3.1.3. Conclusion ... 54
4.3.2. Quality Gates définies pour le projet Backend (Java/Spring Boot) ... 54
4.3.2.1. Justification des métriques et seuils spécifiques ... 54
4.3.2.2. Explications détaillées sur les métriques pour le backend ... 56
Maintainability Rating (Notation de Maintenabilité) ... 56

Reliability Rating (Notation de Fiabilité) ... 56
Security Rating (Notation de Sécurité) ... 57
4.3.2.3. Pourquoi exiger des Ratings A dans les Quality Gates ? ... 57
4.3.2.4. Application pratique dans notre workflow CI/CD ... 58
4.3.3. Quality Gates définies pour le projet Frontend (Angular) ... 58
4.3.3.1. Justification des métriques et seuils spécifiques pour le Frontend ... 58
Spécificités des Quality Gates pour le Frontend Angular ... 60
Différences clés avec les Quality Gates du Backend ... 60
Métriques spécifiques au Frontend qui pourraient être ajoutées ... 60
Stratégie d'évolution des Quality Gates Frontend ... 61
Conclusion ... 61
4.3.3.2. Explications détaillées sur les métriques pour le frontend ... 61
New Blocker severity Issues is greater than 0 ERROR ... 61
New High severity Issues is greater than 2 ERROR ... 62
New Medium et Low severity Issues is greater than 10 WARNING ... 62
4.3.3.3. Pourquoi « Medium et Low severity Issues » n'a pas été choisi pour le Backend 62
4.4. Quality Profiles dans SonarQube ... 63
4.4.1. Avantages de la séparation des projets pour les Quality Profiles ... 63
4.4.2. Quality Profiles par défaut ... 63
4.4.3. Évolution future des Quality Profiles ... 64
4.4.4. Conclusion ... 64
4.5. Différences entre « New Code » et « Overall Code » dans SonarQube ... 64
4.5.1. Définition des concepts ... 65
4.5.1.1. New Code ... 65
4.5.1.2. Overall Code ... 65
4.5.1.3. Différences fondamentales d'approche ... 65
4.5.2. Pourquoi cette distinction est importante ... 65
4.5.2.1. Approche « Clean as You Code" ... 65
4.5.2.2. Gestion efficace de la dette technique ... 66
4.5.2.3. Application différenciée des Quality Gates ... 66
4.5.3. Application dans les workflows CI/CD de BobApp ... 66
4.5.3.1. Configuration actuelle ... 66
4.5.3.2. Visualisation des résultats ... 66
4.5.4. Stratégies recommandées ... 67
4.5.4.1. Quality Gates différenciées ... 67
4.5.4.2. Visualisation et suivi ... 67
4.5.4.3. Intégration dans le processus de développement ... 67
4.5.4.4. Exemple concret pour BobApp ... 67
Situation actuelle hypothétique ... 67
Interprétation ... 67
Actions recommandées ... 68
Conclusion ... 68

5. Installation et configuration de SonarQube pour définir deux projets correspondants à nos deux workflows ci-cd ... 69

5.1. Créer le projet pour le backend : clé cpierres_P10cicd-backend ... 70
5.1.1. Permettre l'accès non authentifié pour la démonstration ... 74
5.2. Créer le projet pour le frontend : clé cpierres_P10cicd-frontend ... 77
5.3. Rendre publique ce dépôt ... 78
5.4. Configurer les Quality gates ... 79
5.4.1. Mise en place des Quality Gates pour le backend ... 79
5.4.1.1. Procédure pour créer des Quality Gates ... 81
5.4.1.2. Quality Gates on Overall Code pour le backend ... 83
5.4.1.3. Quality Gates on New Code pour le backend ... 86
5.4.1.4. Application de la quality gates du backend sur cpierres_P10cici-backend 89
5.4.1.5. Permissions de modification ... 89
5.4.2. Mise en place des Quality Gates pour le frontend ... 91
5.4.2.1. Quality Gates on Overall Code pour le frontend ... 91
5.4.2.2. Quality Gates on New Code pour le frontend ... 91
5.4.2.1. Application de la quality gates du frontend sur la clé de projet cpierres_P10cici-frontend 91

5.4.3. Vue des 2 projets SonarQube (un pour chaque couche/workflow) 92
5.1. Intégration du plugin SonarQube dans l'IDE IntelliJ 93
5.1. Possibilité pour le développeur d'exécuter les workflows sur le réseau local sur sa branche (avant la PR) 93
5.1.1. sonar-local-analyze-backend.bat 94
5.1.2. sonar-local-analyze-frontend.bat 94

6. Exploitation des workflows 95

6.1. Test du workflow avec modif uniquement sur le back 95
6.1.1. Création d'une PR sur le back par cpierres59 (autre user que propriétaire)95
6.1.1.1. Dans onglet Actions, cliquer sur le commit qui a déclenché le workflow afin d'afficher le détail (et récupérer les artefacts au besoin) 97
Optimisation du workflow pour parallèlisation du job backend-sonar et du job backend-docker 98
6.1.1.2. Cliquer sur le job backend-tests afin d'afficher les steps et leur détail100
6.1.1.3. Cliquer sur le job backend-sonar afin d'afficher les steps et leur détail101
Affichage du rapport SonarQube directement depuis le step Lien SonarQube Backend 102
Afficher les Issues classés par sévérité 103
Afficher les Security Hotspots 104
6.1.1.4. Cliquer sur le job backend-docker afin d'afficher les steps et leur détail104
6.1.2. L'approbateur cpierres approuve la PR (malgré les avertissements dans Overall Code) 105
6.1.3. Le collaborateur cpierres59 lit le commentaire d'approbation du reviewer puis merge sa PR 108
6.1.4. Cas avec erreur lors de la compilation ou dans les tests 112
6.1. Test du workflow avec modif uniquement sur le front 113
6.1.1. Lien SonarQube Frontend 116
6.2. Test du workflow avec modif à la fois sur le back et le front (remise en place du code d'origine) 116
6.2.1. Les commentaires de PR pour la génération des images apparaissent bien pour les 2 couches 118
6.2.2. Dans écran Actions, affichage des workflows exécutés parallèlement ... 119
6.2.3. Affichage des rapports sonar du back en cliquant sur les liens affichés dans job backend-sonar 119
6.2.3.1. Rapport du backend sur New code 119
6.2.3.2. Rapport du backend sur Overall code 120
6.2.3.3. Interprétation des résultats 121
Explication de chaque point 121
Coverage 39,2% (Failed; requis ≥ 70%) 121
Reliability: 1 Open issue - Rating D 121
Maintainability: 8 Open issues - Rating A 123
Security Hotspots: 2 — Rating E 123
Priorisation des actions 124
6.2.1. Affichage des rapports sonar du back en cliquant sur les liens affichés dans job backend-sonar 125
6.2.1.1. Rapport du frontend sur New code 125
6.2.1.2. Rapport du frontend sur Overall code 126

7. Analyse des retours utilisateurs 127

7.1. Actions prioritaires 127

8. Annexes 129

8.1. Protections du dépôt (copies d'écran) 129
8.1.1. Protéger la branche main en obligeant les développeurs à passer par des Pull Requests 129
8.1.1.1. Définir une ruleset de branche 129
Cocher également Require status checks to pass 130
8.1.2. Protéger les branches workflow-test-* 132
8.1.3. Création du fichier .github/CODEOWNERS 134
8.1.3.1. Contenu du fichier 135
8.1.3.2. Optionnel : inviter un collaborateur (cpierres59) directement sur ce repo135

Propriétaire du compte cpierres envoi une invitation à cpierres59 .. 136
Le collaborateur reçoit un email pour accepter l'invitation 136
L'écran suivant apparait pour Accepter l'invitation 137
Le collaborateur peut désormais cloner le repository 137
Le propriétaire voit le collaborateur inscrit 138
cpierres59 crée un autre projet dans intelliJ avec un répertoire de clonage différent de la session cpierres 138
8.1. Exemples de scénarios pour tester les workflows 139
8.1.1. Tests pour le backend 139
8.1.1.1. Test d'échec de compilation 139
8.1.1.2. Test d'échec des tests unitaires 139
Exécution du workflow (non souhaitable : container ne devrait pas être généré) 140
Correctif du workflow backend-cy.yml 140
8.2. Gérer les PullRequest (PR) 141
8.2.1. Créer une branche de modification par un collaborateur contributeur (cpierres59) 141
8.2.2. Création de la Pull Request par le contributeur 141
8.2.3. Fermer cette PR mal adressée avec un commentaire 145
8.2.4. Re-créer la Pull Request en se référant à la base de votre propre dépôt !145
8.2.5. Traiter la PR par un reviewer 148
8.2.5.1. Rôle Reviewer 150
Actions typiques 150
8.2.5.2. Rôle Assignee 150
Actions typiques 150
8.2.5.3. Comparaison pratique des rôles 150
8.2.6. cpierres59 vérifie ses PR dans IntelliJ 151
8.2.6.1. Copier l'url de la PR vers le browser connecté sur cpierres59 152
8.2.7. Cpierres59 effectue une modification sur la branche et la pousse 153
8.2.7.1. Soumettre cette revue : 155
8.2.8. Cpierres reçoit un email l'informant que cpierres59 a inscrit un commentaire dans sa review 155
8.2.9. Mais pour cpierres, la request view n'est pas terminée par cpierres59 :. 156
8.2.10. cpierres59 approuve son changement de PR 156
8.2.11. Cpierres voir que la PR est terminée et propose le squash and merge ou rebase 157
8.2.12. Cpierres rebase de la branche vers main pour clôre la PR 158
8.2.13. Afficher les tests qui se sont exécutés 159
8.2.13.1. Détail de l'exécution 159
8.2.13.2. Affichage de l'artefact du backend (rapport de couverture dézippé) .. 160
8.3. Build vers dockerhub 161
8.3.1. Créer un token dans DockerHub 161
8.3.2. Configurer un secret d'accès à DockerHub dans les paramètres du dépôt GitHub 163
8.4. Exemples de Blocker severity Issues et High severity Issues dans SonarQube 164
8.4.1. Couche Backend (Java/Spring Boot) 164
8.4.1.1. Blocker severity Issues (Backend) 164
8.4.1.2. High severity Issues (Backend) 165
8.4.2. Couche Frontend (Angular) 165
8.4.2.1. Blocker severity Issues (Frontend) 165
8.4.2.2. High severity Issues (Frontend) 166
8.4.3. Différences clés entre Blocker et High severity Issues 167
8.4.3.1. Blocker severity Issues 167
8.4.3.2. High severity Issues 167
8.5. Exemples de Faux Positifs pour des Blocking severity Issues dans Angular 167
8.5.1. Utilisation de innerHTML dans les templates Angular 167
8.5.2. Utilisation de setTimeout ou setInterval sans nettoyage explicite 168
8.5.3. Accès direct au DOM via ElementRef 168
8.5.4. Utilisation de eval() dans les expressions dynamiques 169
8.5.5. Références circulaires dans les services Angular 169
8.5.6. Pourquoi une tolérance est nécessaire pour le frontend 169

1. Vue générale

1.1. Contexte

BobApp est une application open source de « blague du jour » lancée il y a 3 ans. Le trafic augmente chaque jour et le développeur initial (Bob) peine à:

- gérer les nombreux bogues remontés par les utilisateurs et à les informer des correctifs;

- exécuter des déploiements manuels coûteux (valider les PR, récupérer le code, lancer les tests, builder, vérifier, déployer via FTP);

Pour rationnaliser et stabiliser le flux de développement, Bob nous a confié la gestion dans un nouveau repository GitHub, le temps de valider la plateforme CI/CD.

Portée de ce document :

- Définir un cadre de contribution (Pull Requests obligatoires; pas de push direct sur main);

- Décrire le pipeline CI/CD (tests, qualité, build, images Docker);

- Intégrer SonarQube et proposer des KPI/Quality Gates;

Limite actuelle de ce projet :

- on s'arrête au push des images vers Docker Hub (pas de déploiement final).

- on ne gère pas le monitoring (surveillance de disponibilité).

1.2. Pourquoi les github actions ?

La solution de CI-CD github actions est adaptée à ce projet Open source car :

- le repository git est stocké sur github

- approche populaire et simple de type : « feature branches depuis main + validation par PR » (pas de branches develop et release)

  - une branche principale (main) protégée,

  - des branches de fonctionnalité/correctifs courtes créées depuis main,

  - fusion via Pull Request avec revue et CI obligatoires,

  - versionnage via tags/releases.

- Aligné avec l'outillage GitHub (branch protection, required checks, CODEOWNERS)

1.3. Travail effectué, décrit en 2 temps

Le projet étant stocké dans Github, l'objectif est de constituer un workflow de CI-CD simple mais complet grâce aux Github actions.

Ayant déjà une expérience CI-CD, notamment avec Jenkins, j'ai souhaité couvrir la quasi-totalité du workflow, avec versionnage des couches backend et frontend, et versionnage des images docker associées.

Pour raison pédagogique, je vais effectuer la présentation en deux temps :

1) Premiers workflows développés : backend-ci.yml et frontend-ci.yml

Un workflow dédié à chaque couche backend et frontend, dont le rôle essentiel est d'assurer la qualité avec les jobs suivants :

- compiler, exécuter les tests (bloquant),

- contrôler la qualité avec sonarQube (non bloquant),

- produire une image docker lors de la soumission de la PR (pas de gestion de version, juste lastest dans la version intiale).

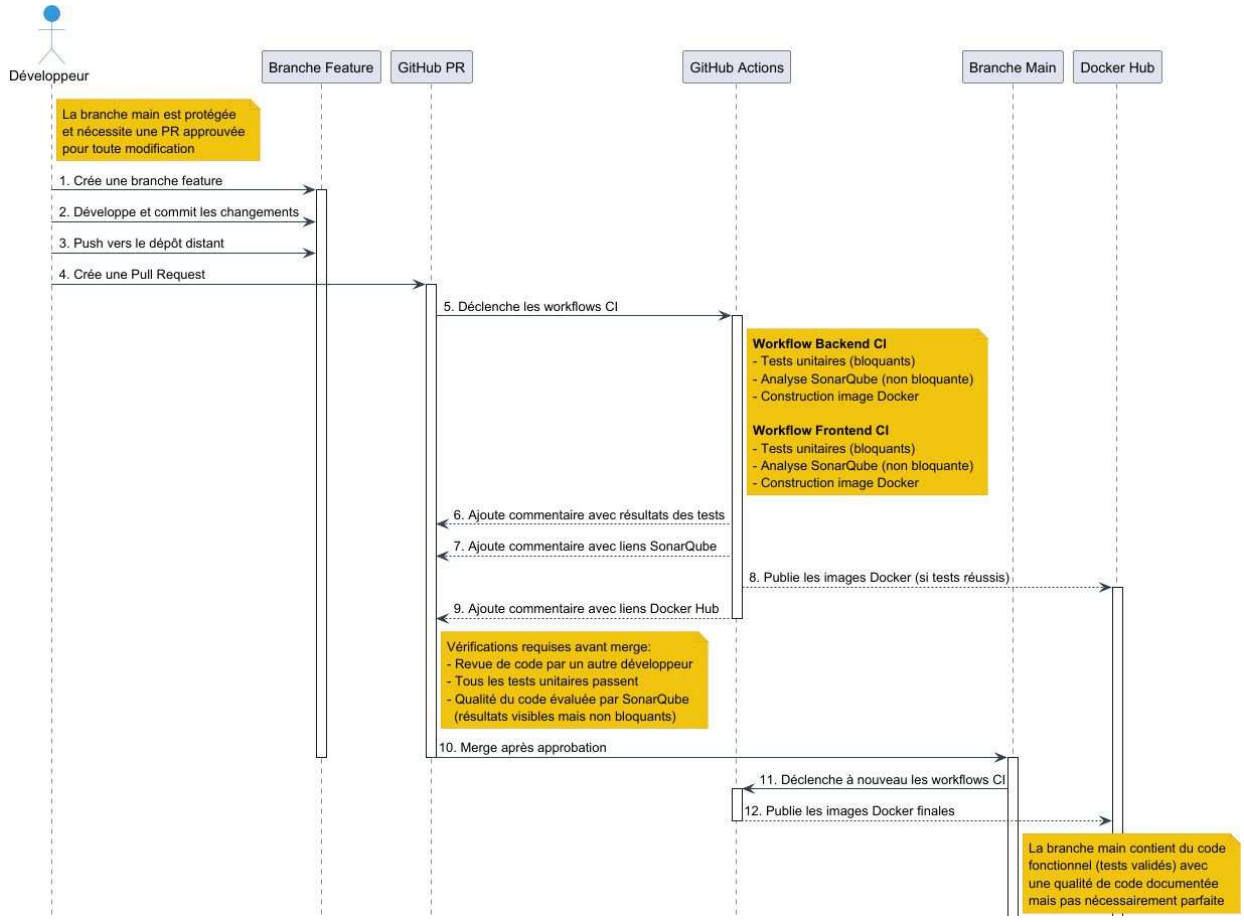
2) Workflows complémentaires de gestion de releases/tags dédiés également à chaque couche :

- release-please-backend.yml
- release-please-frontend.yml

Note : je vous prie de m'excuser car, par manque de temps, certaines copies écrans liées à la partie 1 n'ont pas été actualisées en fonction des évolutions de la partie 2 de gestion des releases.

1.3.1. Partie 1 : validation et contrôle qualité avec backend-ci.yml et frontend-ci.yml

1.3.1.1. Diagramme de séquence simplifié (partie 1)



L'approche proposée vise à maintenir un équilibre entre qualité du code et vélocité de développement, en utilisant SonarQube comme guide plutôt qu'un obstacle. L'approche choisie se veut « non punitive ».

Bien entendu, cette approche, qui tient compte du fait qu'on récupère du code legacy, peut évoluer avec la maturité de la maitrise de la qualité. Ainsi, à moyen terme, lorsque le code existant sera assaini, il serait possible de rendre le non respect qualitatif de SonarQube <u>bloquant</u> pour le merge de la PR dans la branche principale ainsi que la production des images Docker.

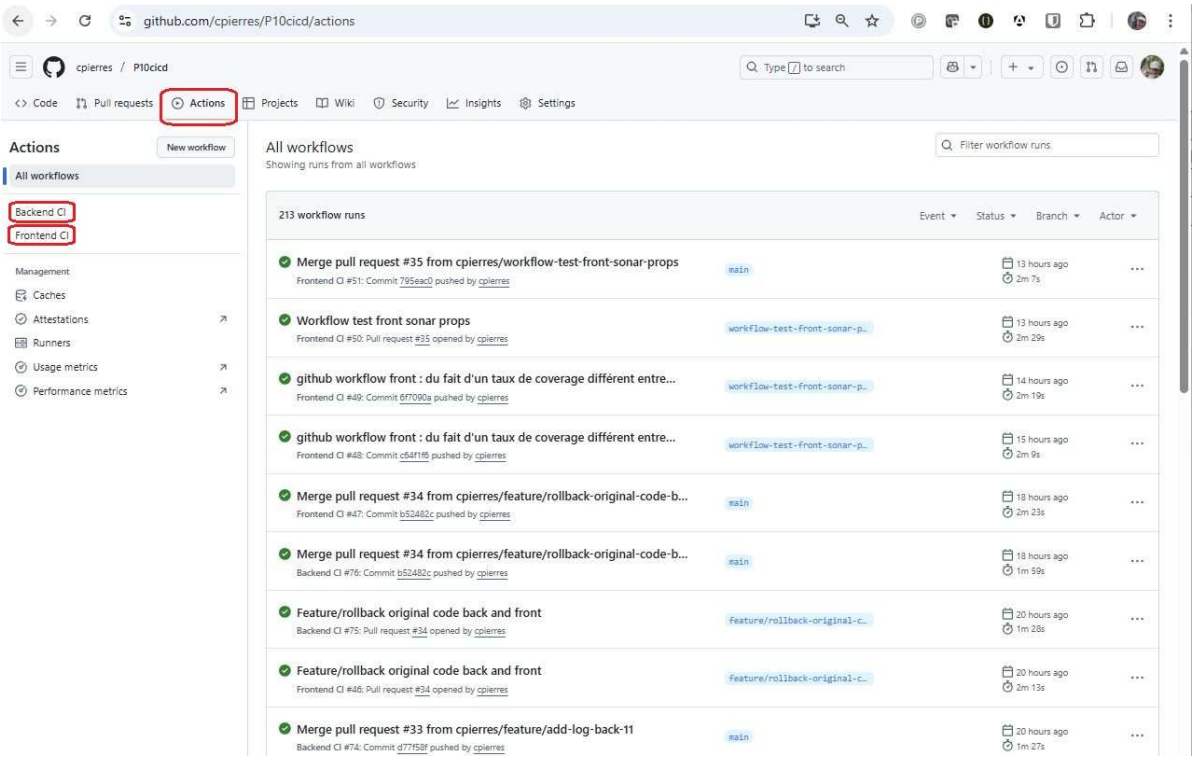
1. **Distinction entre qualité critique et non-critique** :
   - Les tests unitaires vérifient le fonctionnement correct du code (critique)
   - SonarQube évalue la qualité du code et les bonnes pratiques (important mais non bloquant)

2. **Flexibilité dans le processus de développement** :
   - Permet de déployer des versions fonctionnelles même avec des dettes techniques
   - Donne de la visibilité sur la qualité sans bloquer le cycle de livraison

3. **Approche pragmatique** :
   - Les problèmes de qualité peuvent être adressés progressivement
   - Les équipes peuvent prioriser les corrections sans bloquer les fonctionnalités

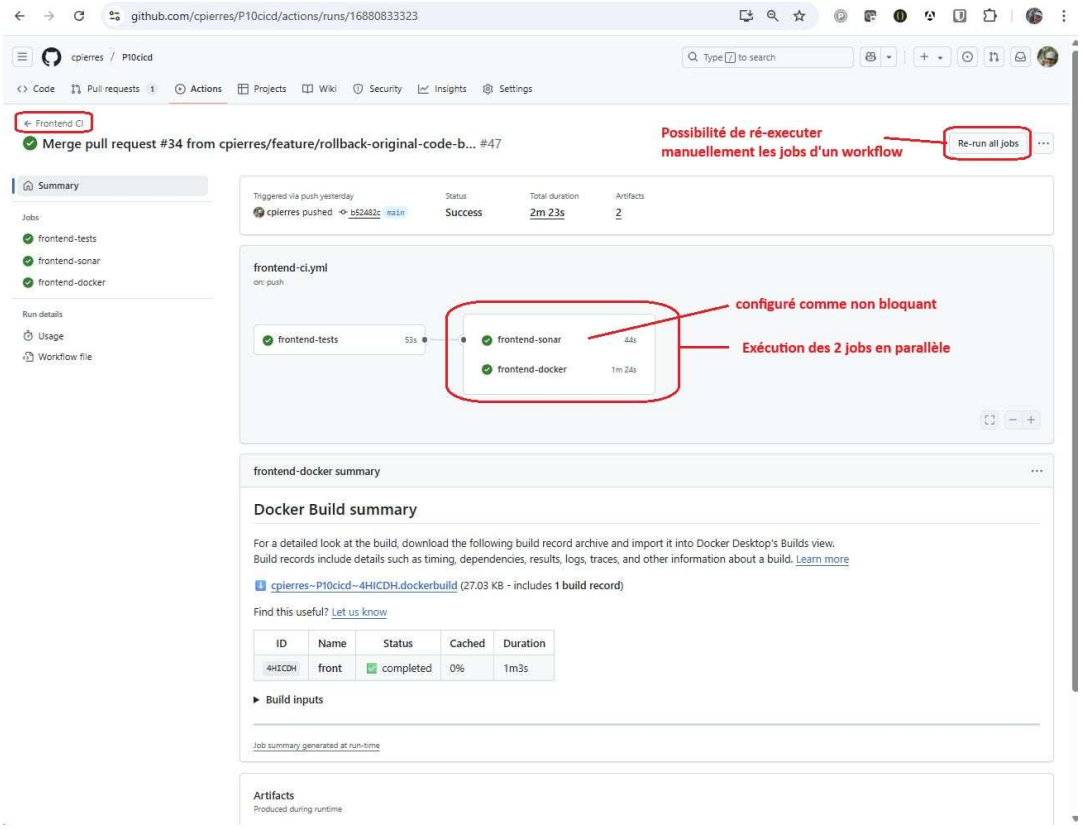
1.3.1.2. Deux workflows de « github actions » séparés (un par couche)

Chacun de ces workflows n'est activé que si un fichier est modifié dans la couche en question. Si des fichiers sont modifiés dans les deux couches simultanément, alors ces deux workflows s'exécutent en parallèle.



Détail du workflow frontend

Au sein de chaque workflow, on a défini 3 jobs principaux :



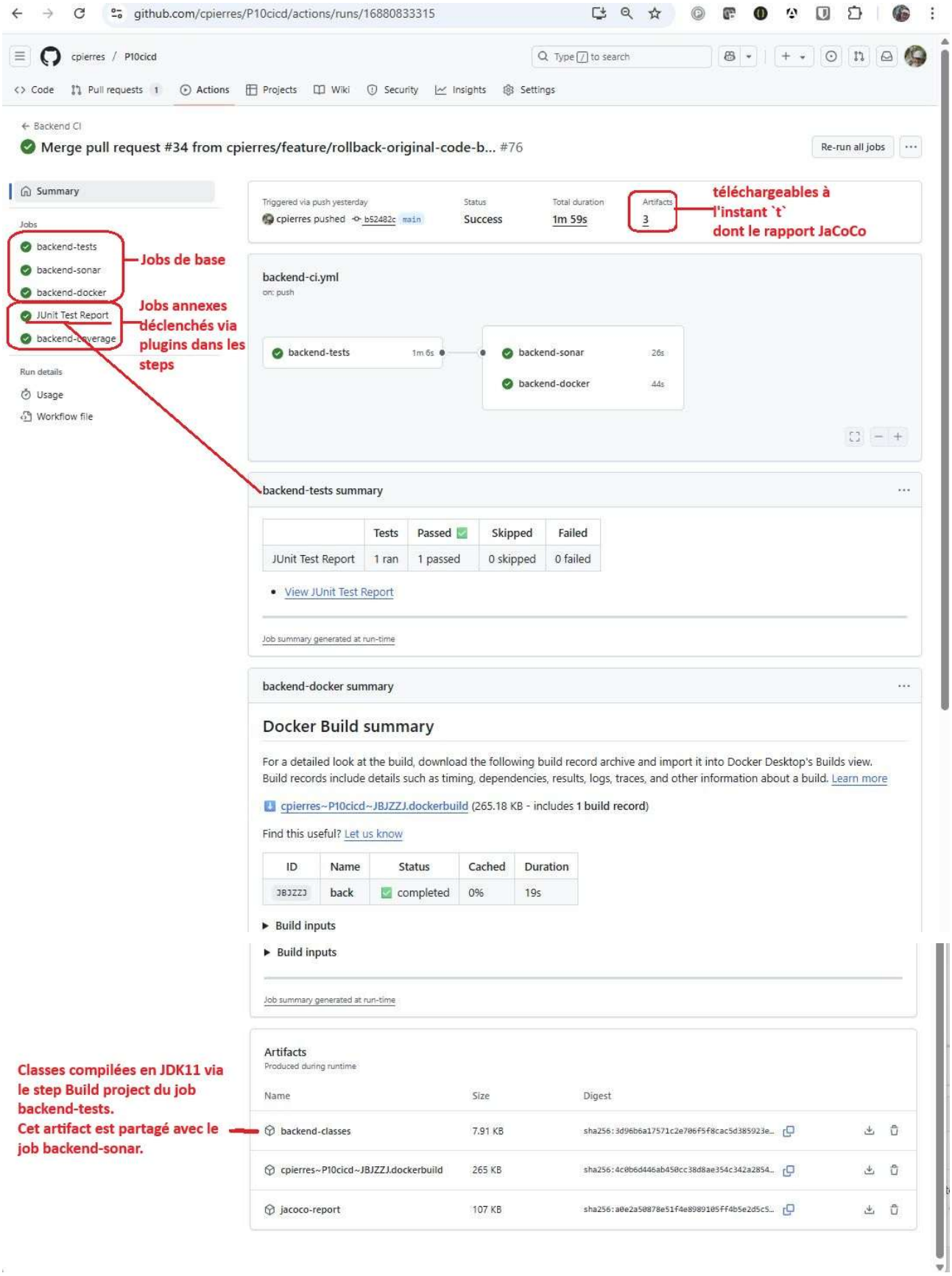
Détail des steps d'un job (par exemple frontend-sonar)

Les « steps » du job sont affichés avec possibilité de consulter les logs de détail.

Si vous cliquez sur ce lien pour avoir le résultat Sonar, s'identifier avec : `ocrGuest` et `ocrGuest123!`

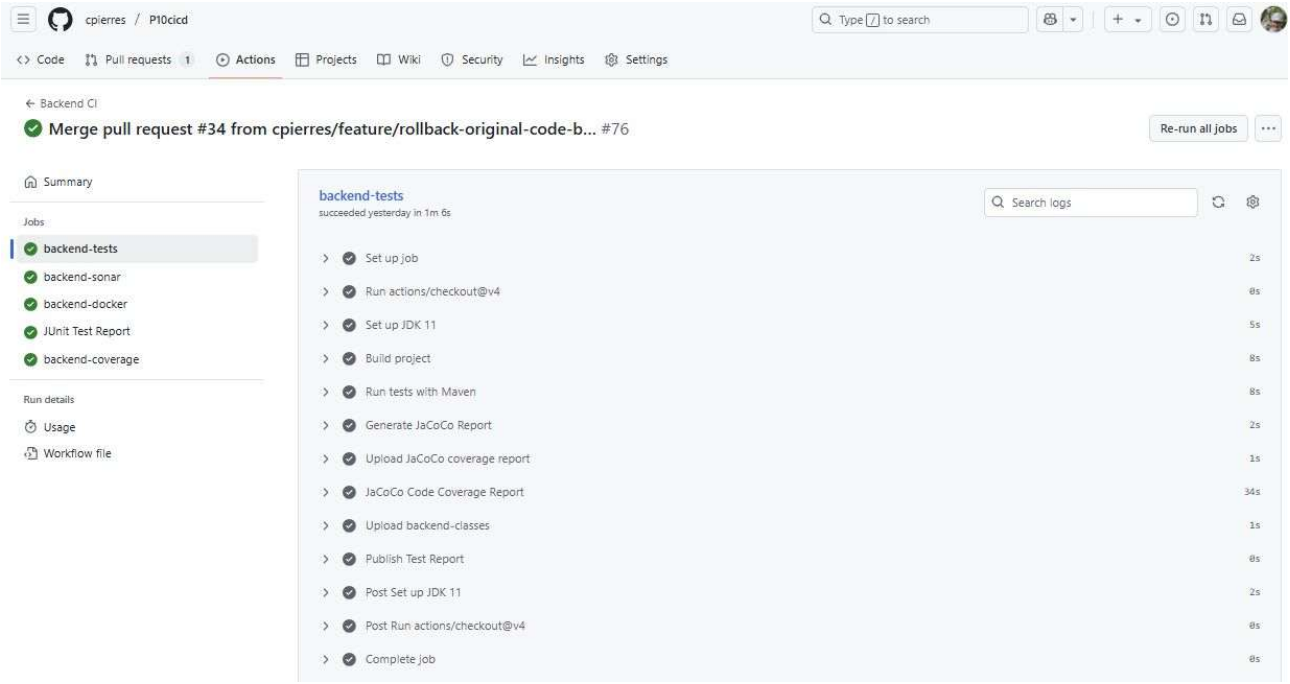
Détail du workflow backend

Même structure de jobs (mais steps différents) :



Détail des steps du job backend-tests

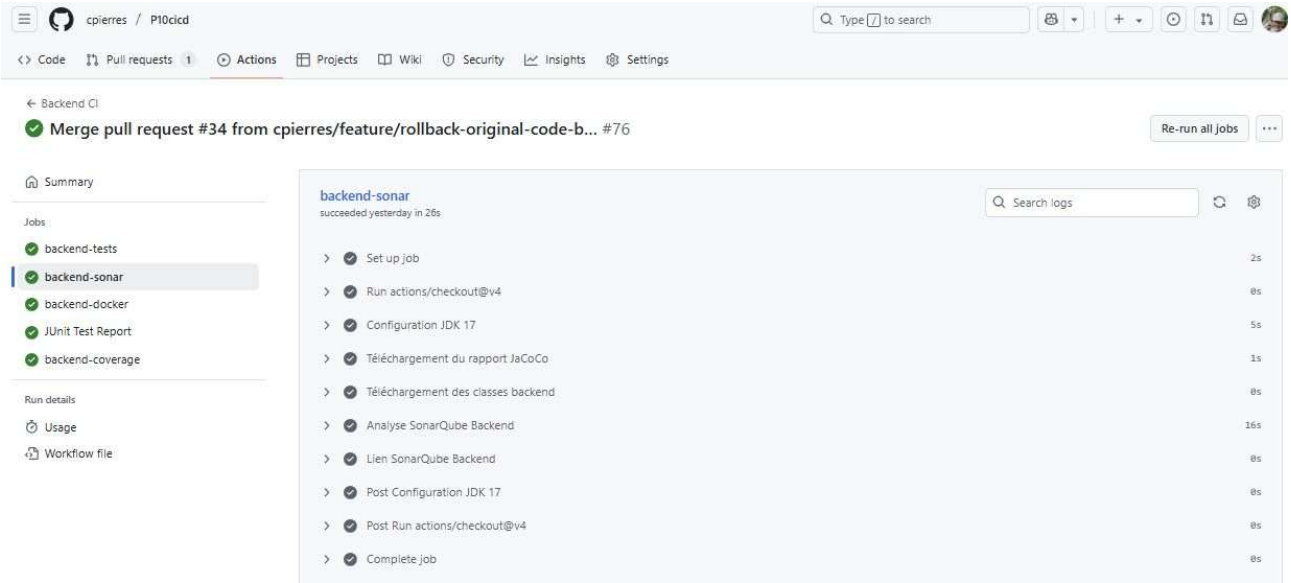
https://github.com/cpierres/P10cicd/actions/runs/16880833315/job/47816008256



<u>Très important</u> : respect de l'environnement du projet à savoir JDK11. Un artefact des classes compilées est produit pour être partagé avec le job Sonar.

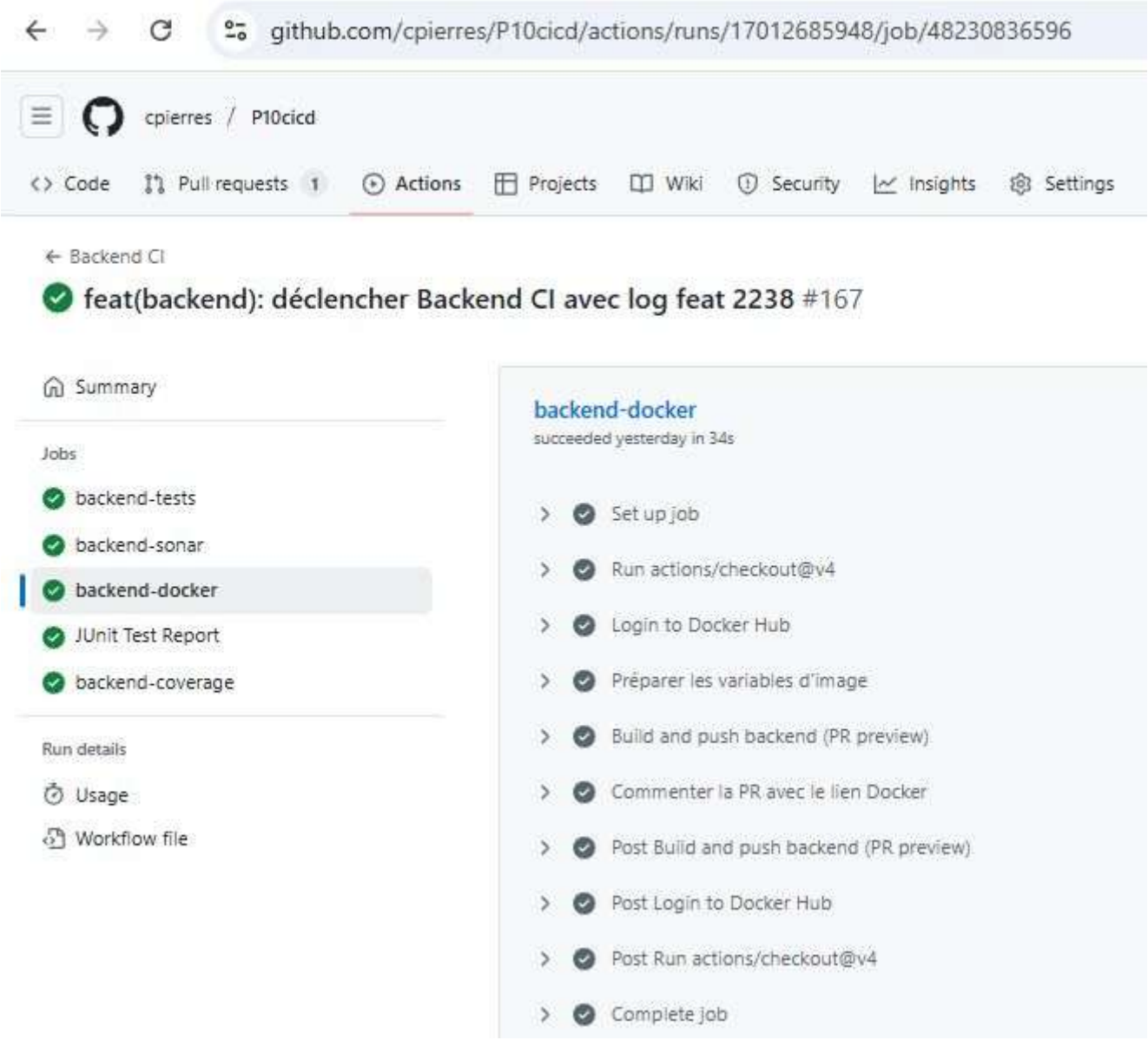
<u>Détail des steps du job backend-sonar</u>

https://github.com/cpierres/P10cicd/actions/runs/16880833315/job/47816098832



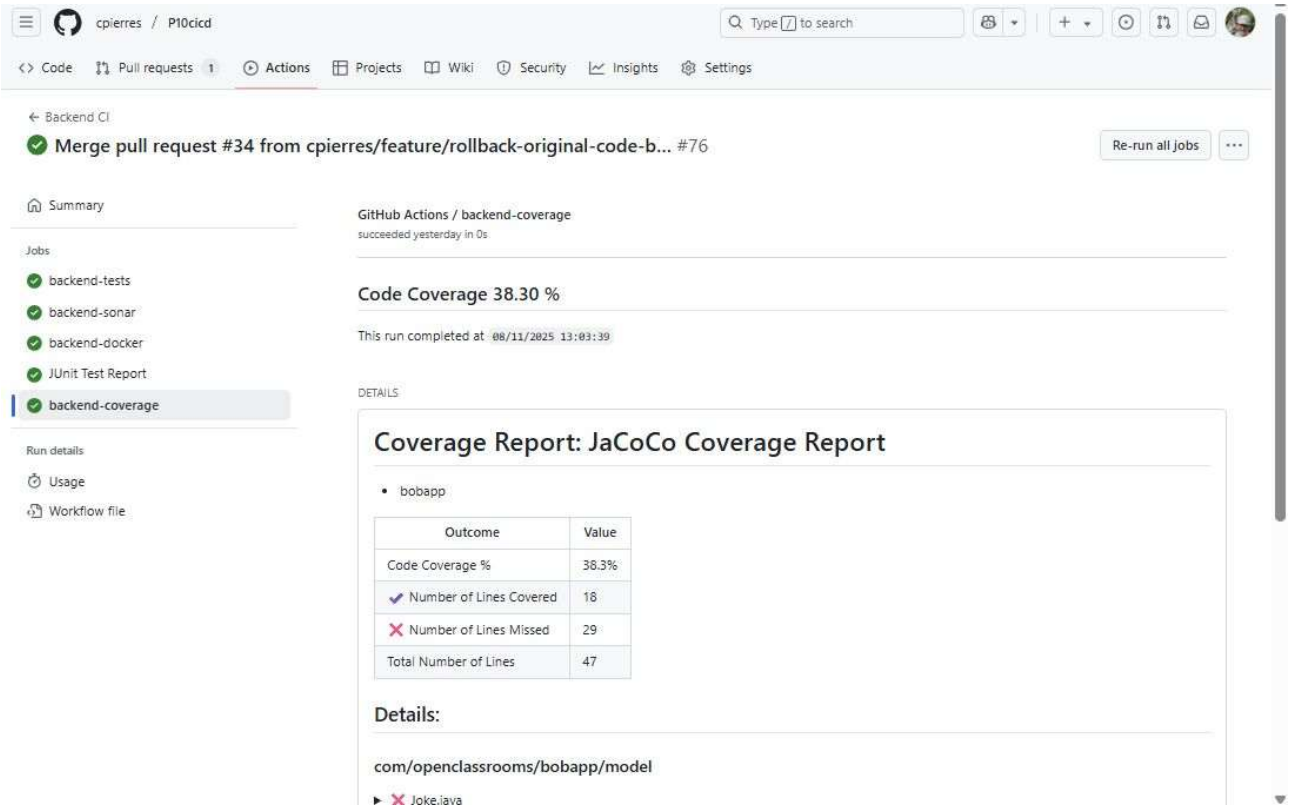
Détail des steps du job backend-docker

https://github.com/cpierres/P10cicd/actions/runs/17012685948/job/48230836596



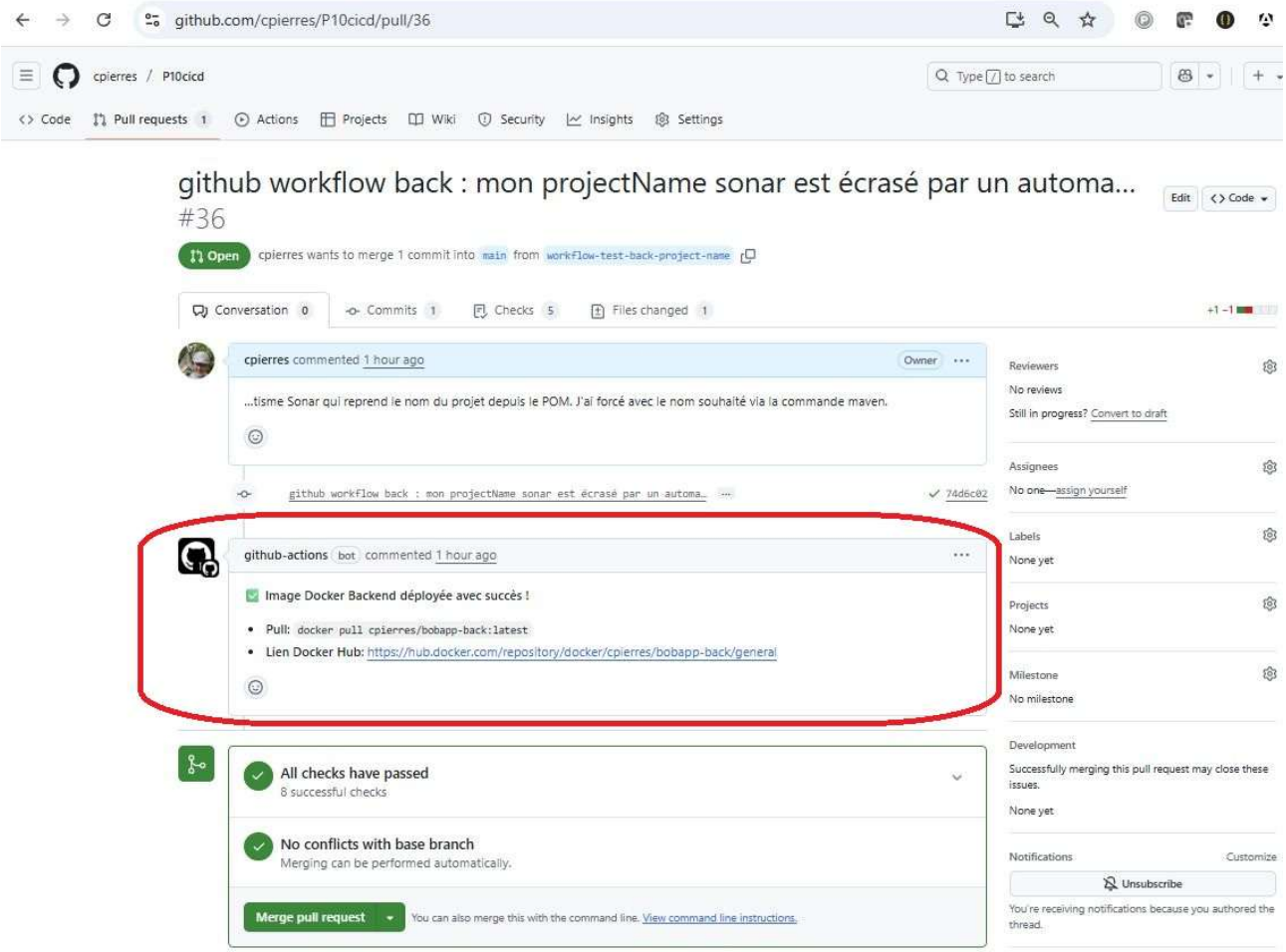
Détail job annexe « backend-coverage » généré par step « JaCoCo Code Coverage Report »

Le plugin `PavanMudigonda/jacoco-reporter` a des fonctionnalités intéressantes, dont la possibilité d'afficher directement le taux de couverture, ce qui évite d'avoir à télécharger l'artifact :



Détail d'une Pull Request

Le workflow ajoute des Commentaires sur la Pull request. L'exemple suivant est un commentaire généré par le job backend-docker pour informer de la bonne génération de l'image docker :



Ces commentaires ont été améliorés (cf. partie 2).

Process d'une Pull request

Ceci est décrit dans la partie 2 d'une manière complète

1.3.1.3. Deux projets SonarQube séparés (un par couche également)

J'ai préféré exposer mon propre SonarQube (version community) sur internet plutôt qu'utiliser SonarCloud, ceci afin de mieux controler l'environnement, sans limitation (Quality Gates) :

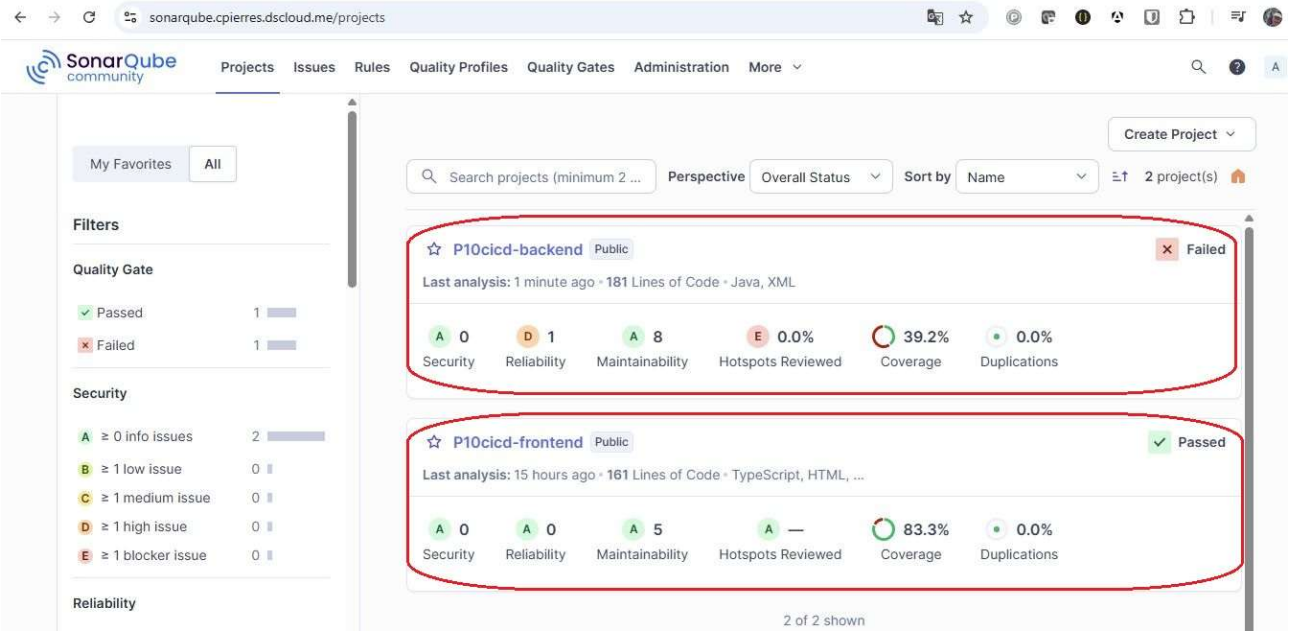
https://sonarqube.cpierres.dscloud.me

Pour une raison que je pourrai vous préciser sur demande, je ne pouvais pas utiliser l'accès anonyme (considéré comme deprecatead dans la dernière version).

Par conséquent, veuillez utiliser le user suivant pour la première connexion :

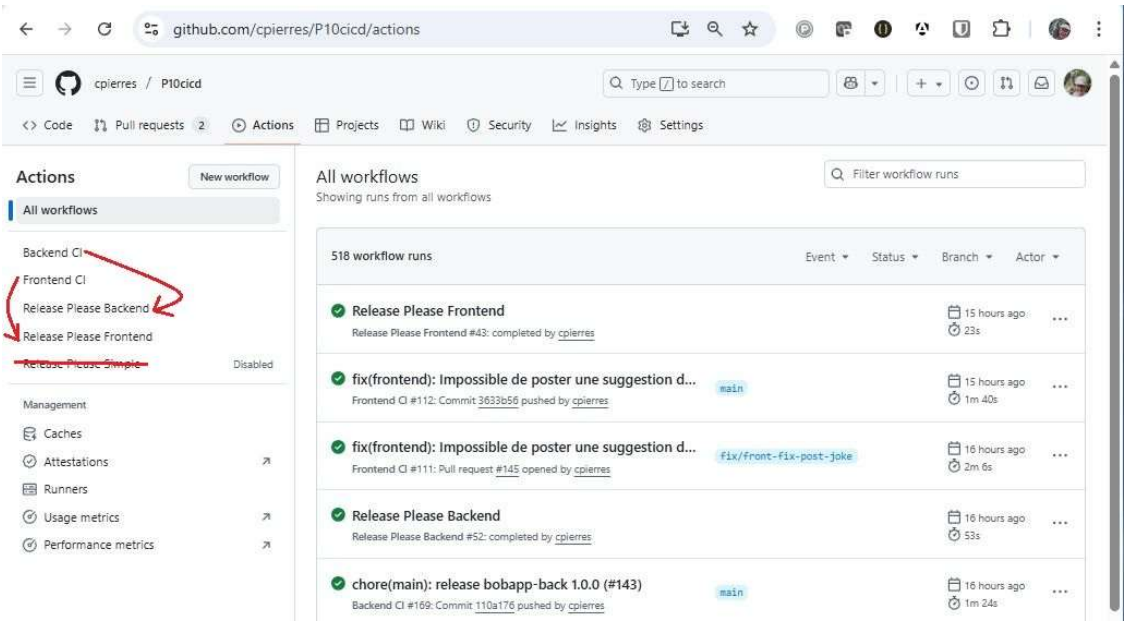
login : `ocrGuest`

password : `ocrGuest123!`

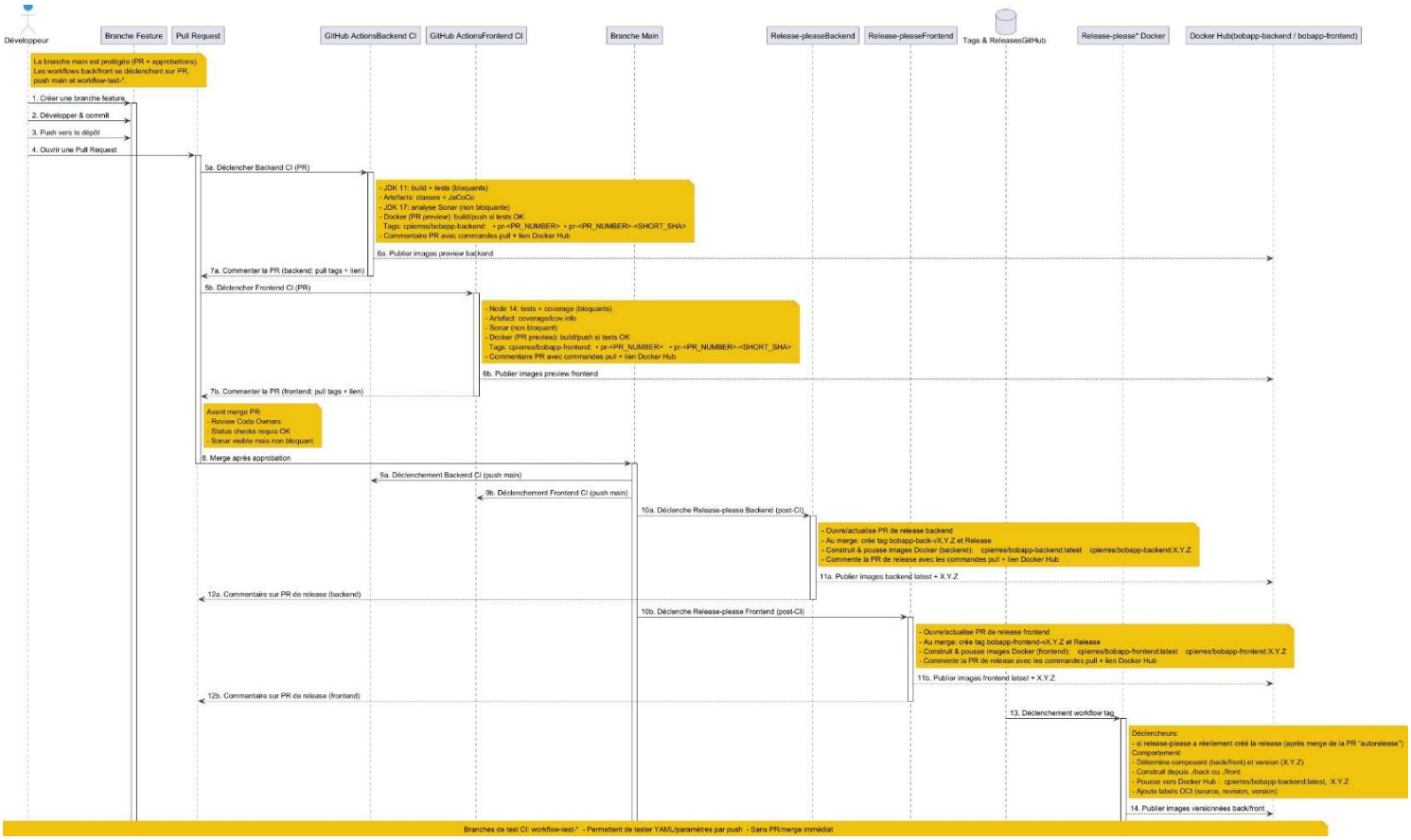


1.3.2. Partie 2 : complèment avec gestion des release (Release-please)

La gestion des releases est la continuité de chacun des workflows précédents, mais la responsabilité étant différente, j'ai créé des workflows distincts (release-please-backend.yml et release-please-frontend.yml), qui s'exécutent en cas de réussite de leur workflow initial respectif :



1.3.2.1. Diagramme de séquence intégrant les release-please (partie 2)

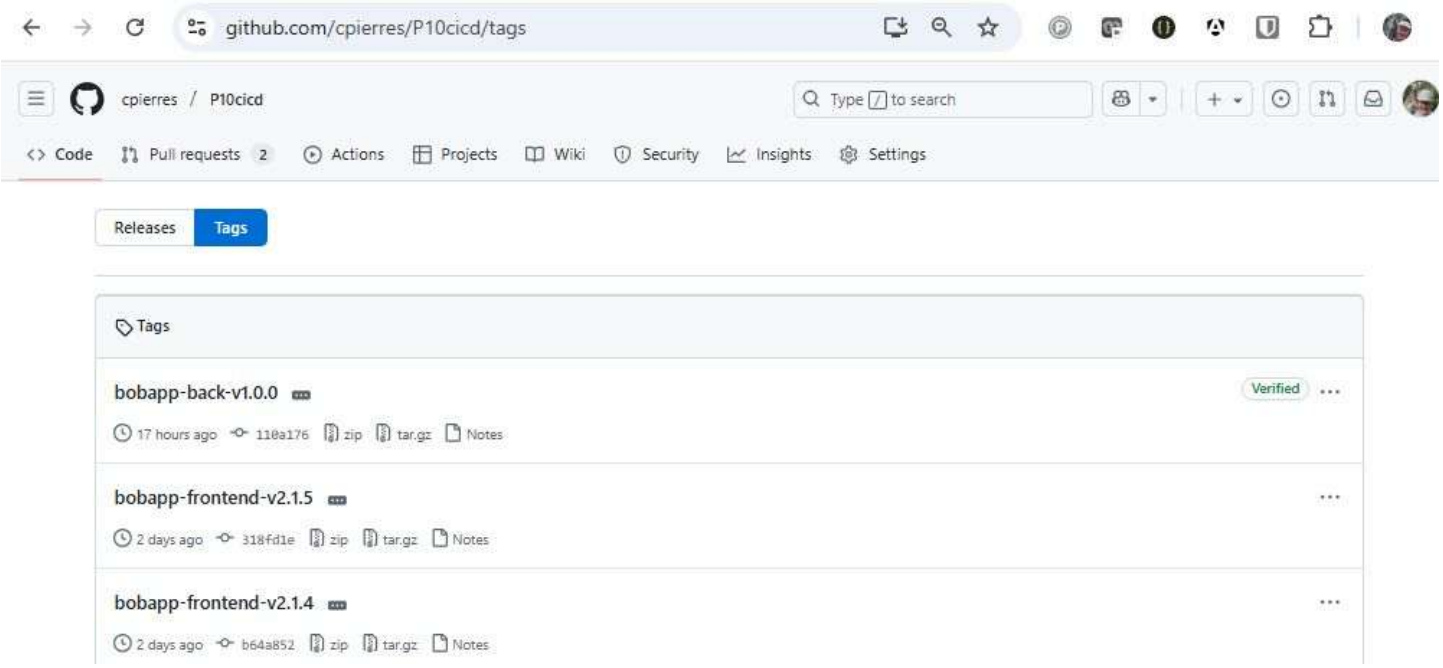


1.3.2.2. Principe de fonctionnement de release-please

- Analyse des commits depuis le dernier tag, au format Conventional Commits, pour détecter les changements (feat, fix, breaking).
- Détermination automatique du bump sémantique selon les types de commits.
- Ouverture d'une « PR de release » qui:
  - met à jour les fichiers de version (manifest/versions),
  - génère/actualise le CHANGELOG (important pour le feedback pour les utilisateurs qui ont signalé des bugs),
  - propose la nouvelle version.
- PR idempotente et auto-actualisée : si de nouveaux commits arrivent, la PR est régénérée avec la bonne version/changelog.
- Au merge de la PR de release: création du tag et de la GitHub Release correspondante, avec notes de version.
- Intégration GitHub Actions: déclenchement manuel, planifié ou sur push; fonctionne avec GITHUB_TOKEN, permissions write.
- Sécurité du flux: pas de tag/release « surprise » - la publication passe par une PR révisable.
- Prévisible et traçable: tout passe par une PR, un changelog généré et des tags versionnés.

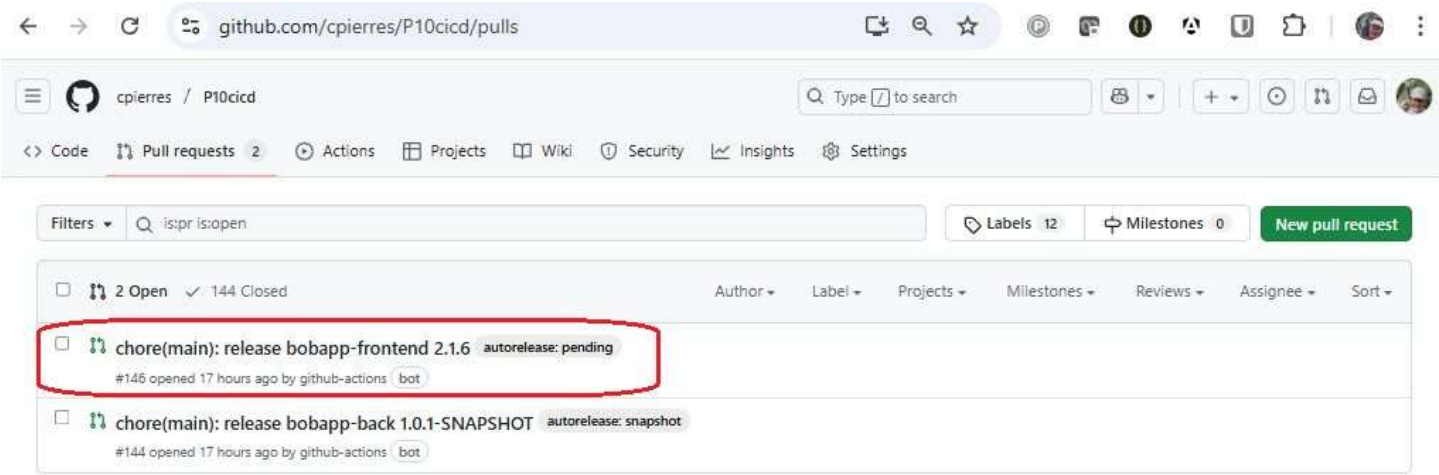
1.3.2.3. Point de départ (exemple avec frontend)

Dernière version de releases/tags



PR de release générée par release-please suite à un précédente validation de PR

Ci-dessous, une PR de release pour le frontend a déjà été générée « automatiquement » par release-please suite à un précédente validation de PR, concernant un fix relevé par un de nos utilisateurs. Du coup, comme il s'agit d'un correctif/patch, cela a défini dans un premier temps une release avec un incrément « mineur » (dernier chiffre+1) :



Note : le préfixe de libellé de commit « chore » assure que ce commit de « maintenance/ménage » ne sera pas pris en compte dans le CHANGELOG.md.

1.3.2.4. Création d'une feature

Pour la circonstance, je vais créer une PR de feature que je vais valider moi-même (ce qui n'est pas conventionnel j'en conviens). Pour des exemples « plus réalistes avec collaborateurs », voir la section « Exploitation des workflows » à la page 95.

Branche : `feat/front-feat-add-tchat`

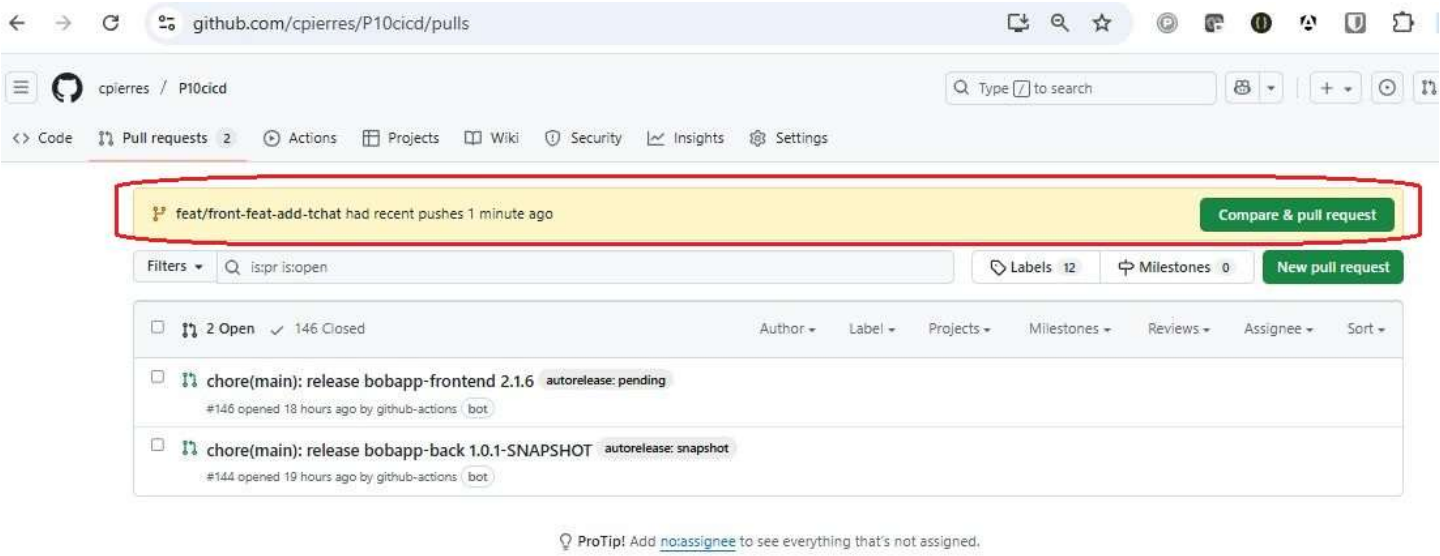
Libellé du commit : `feat(front): Ajout possibilité de tchatter entre blagueurs`

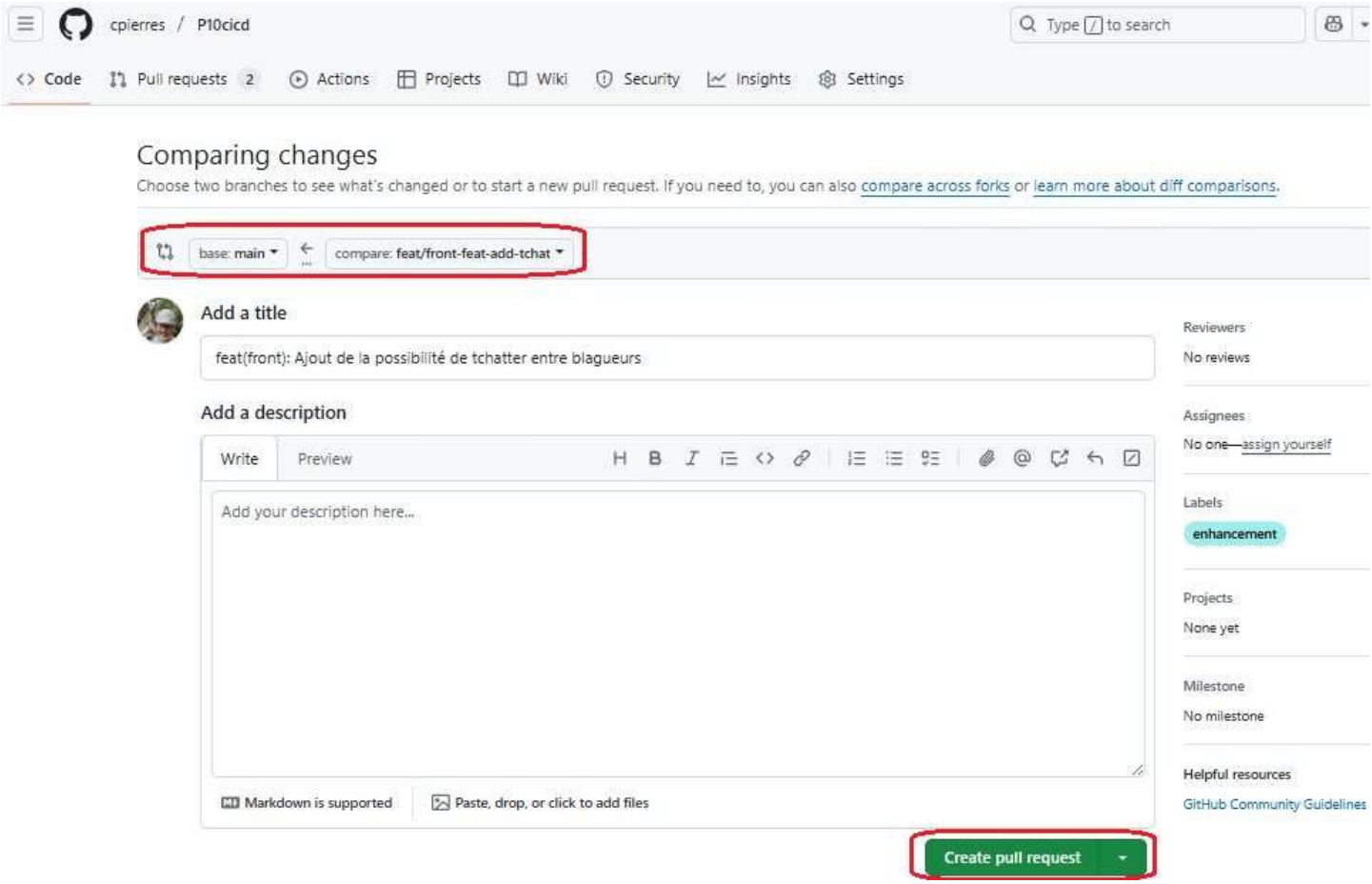
<u>Oops désolé d'une erreur</u> : en tant que propriétaire du dépôt, j'ai eu la possibilité de faire l'erreur de pusher directement sur `main` le commit `feat(front): Ajout possibilité de tchatter entre blagueurs`. Du coup, il n'a pas été analysé par mon workflow release-please-frontend ! <u>Ne pas en tenir compte dans le log svp</u>. Je renomme un peu mon commit que je vais pousser sur la branche :

Libellé du commit : `feat(front): Ajout de la possibilité de tchatter entre blagueurs`

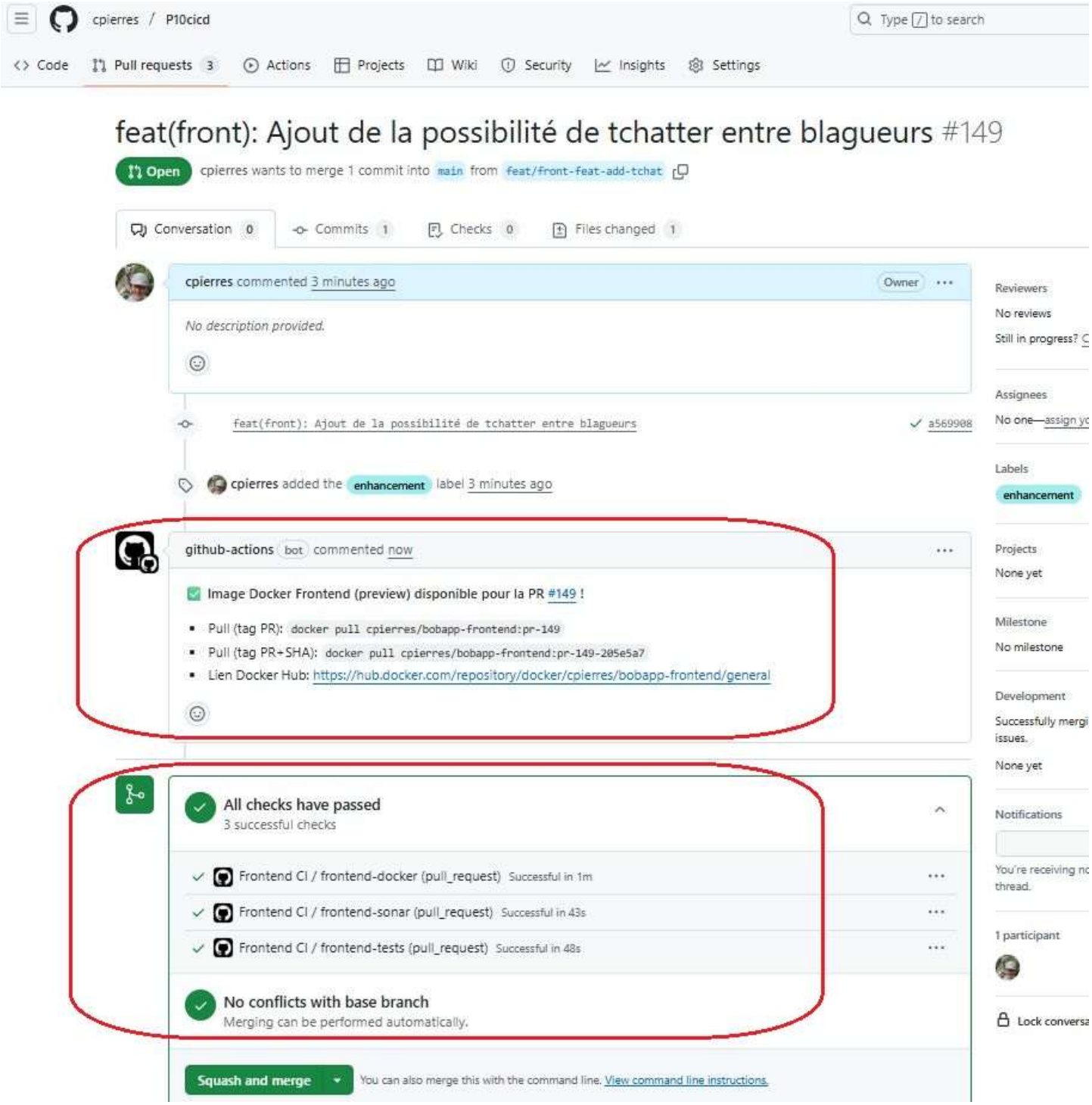
<u>Convertir le push de la branche en PR</u>

Dans l'écran des **Pull requests** :




cpierres / P10cicd
Type / to search
Code
Pull requests 2
Actions
Projects
Wiki
Security
Insights
Settings
Comparing changes
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.
base: main
compare: feat/front-feat-add-tchat
Add a title
feat(front): Ajout de la possibilité de tchatter entre blagueurs
Add a description
Write
Preview
Add your description here...
Markdown is supported
Paste, drop, or click to add files
Create pull request
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
enhancement
Projects
None yet
Milestone
No milestone
Helpful resources
GitHub Community Guidelines

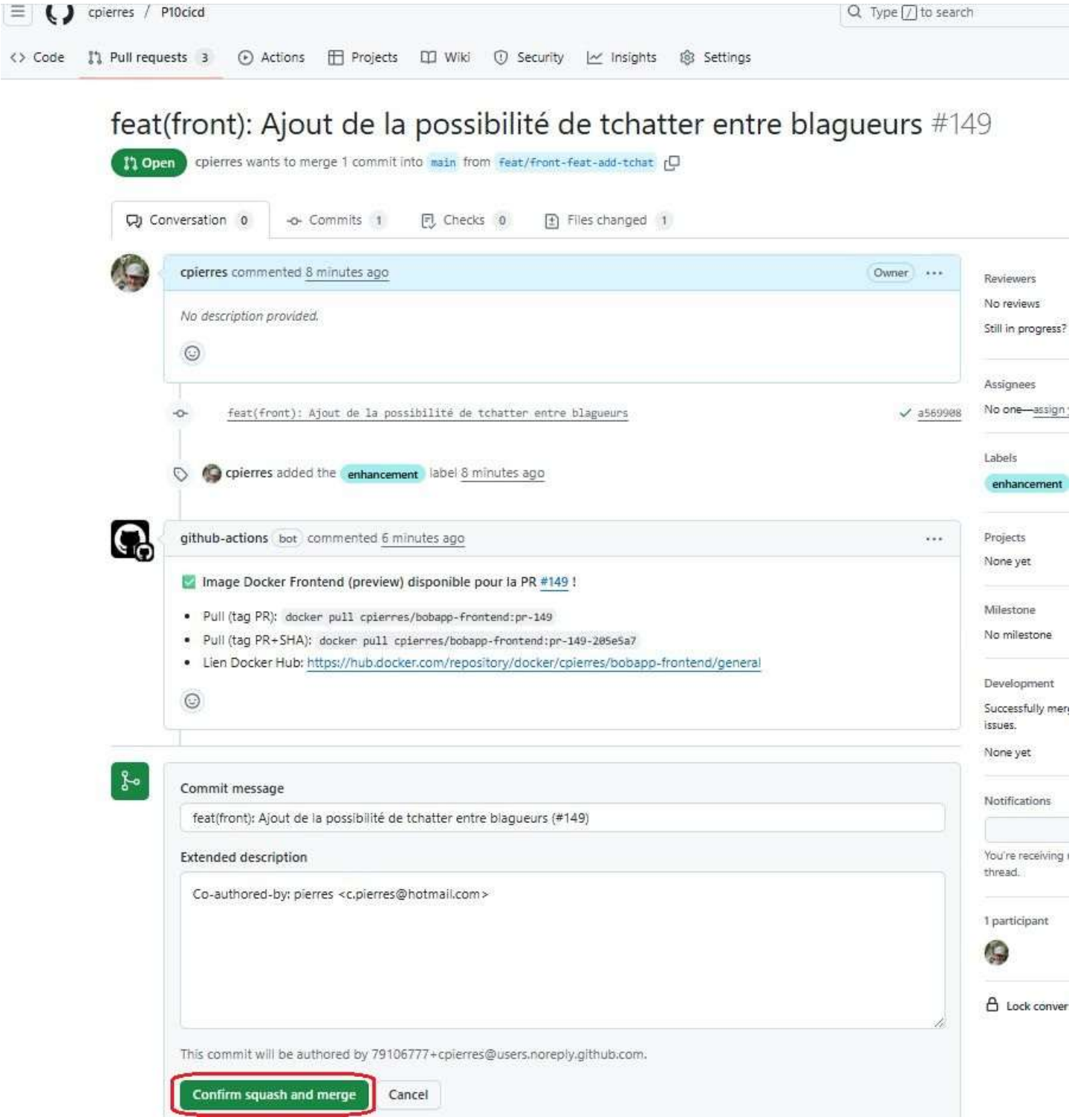
Checks terminés avec commentaire ajouté pour l'image docker « pr » (mail expédié)



1.3.2.5. L'approbateur valide/merge la PR

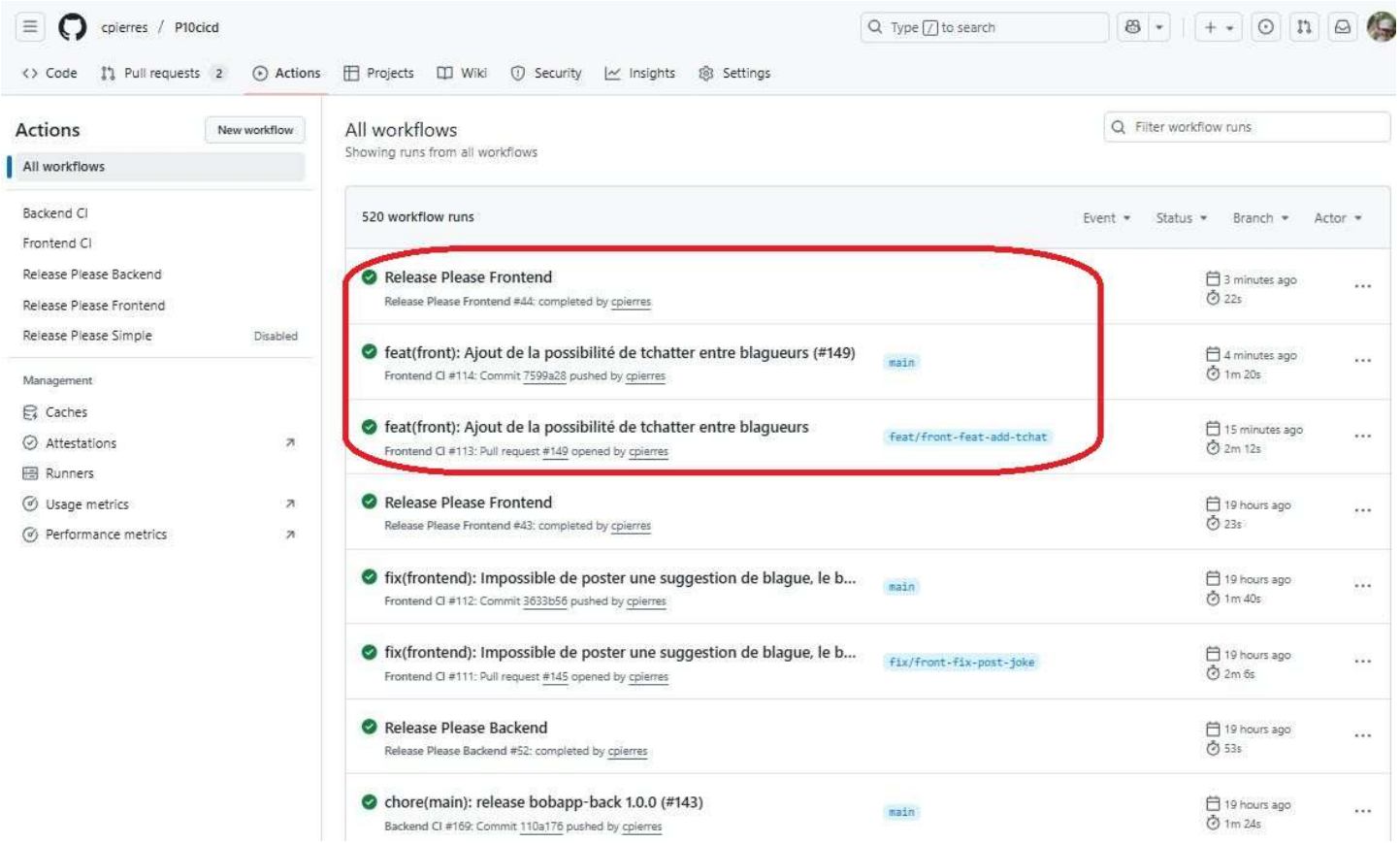
Lorsque l'approbateur revoit la PR, si correcte, il clique sur l'un des options de merge (**Squash and merge** par exemple).

Ensuite, le message du commit lui est suggéré (suffixé par le numéro de la PR) et il peut confirmer :



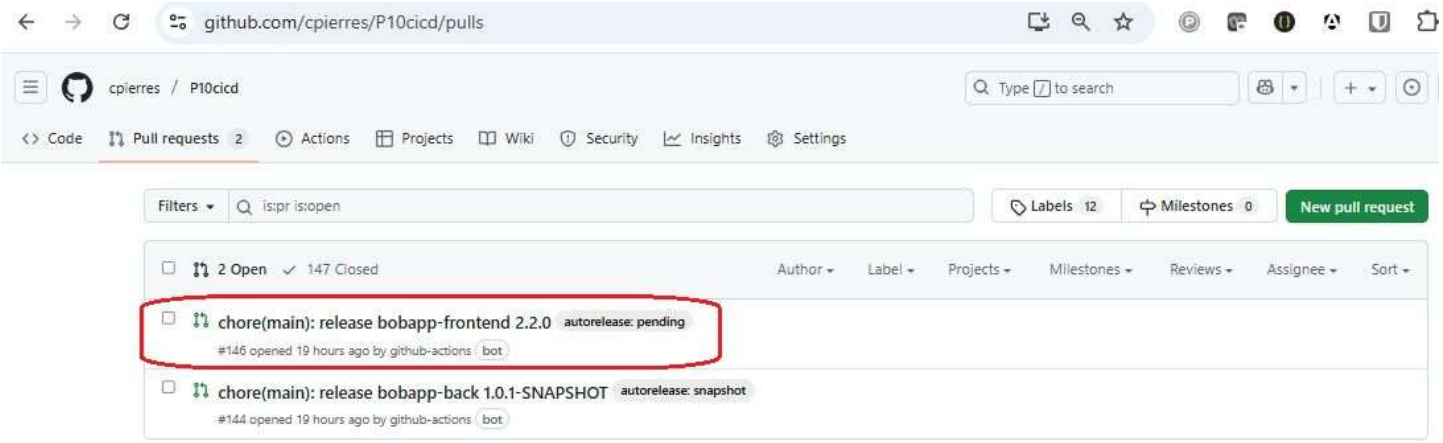
La PR est alors mergée vers main et fermée.

1.3.2.6. frontend-ci est automatiquement déclenché sur le main puis Release Please Frontend



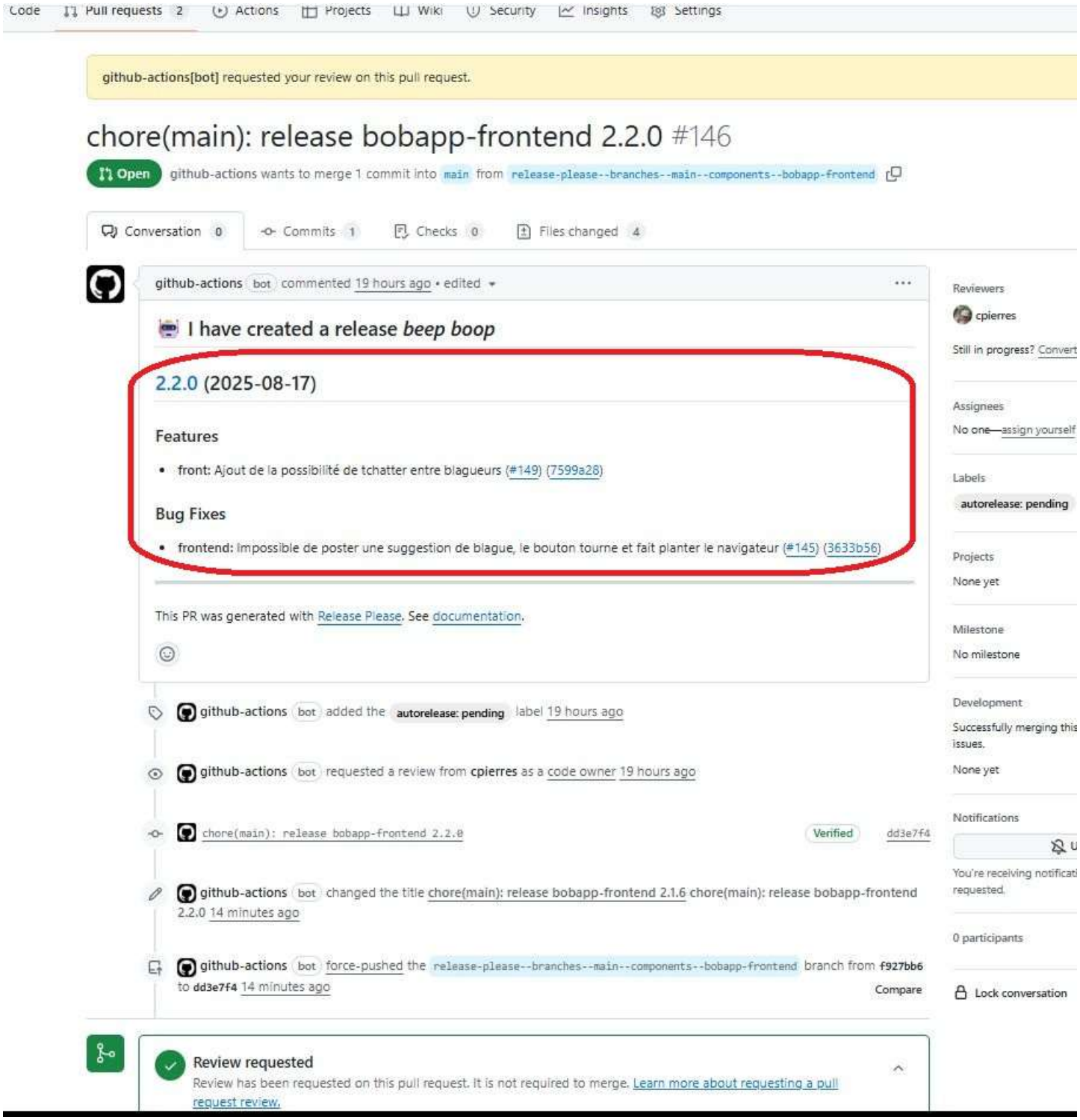
1.3.2.7. Release-please a automatiquement adapté la version dans la PR de release

La version précédente était 2.1.6 et est devenue 2.2.0 puisqu'on ajouté une feature (on repart à 0 pour le numéro mineur de patch) :



Détail de la PR de release (en statut autorelease :pending) ; vue CHANGELOG.md

L'utilisateur ayant rapporté le bug bloquant à traiter prioritairement sera ravi de constater qu'il est résolu !



Vous avez compris qu'on peut ajouter autant de PR qu'on veut, puis viendra un moment où le code owner se décidera à générer la release. Pour cela, il devra merger la PR de release :

Open chore(main): release bobapp-frontend 2.2.0 #146

github-actions wants to merge 1 commit into main from release-please--branches-...

github-actions bot added the autorelease: pending label 19 hours ago

github-actions bot requested a review from cpierres as a code owner 19 hours ago

chore(main): release bobapp-frontend 2.2.0 Verified dd3e7f4

github-actions bot changed the title chore(main): release bobapp-frontend 2.1.6 chore(main): release bobapp-frontend 2.2.0 21 minutes ago

github-actions bot force-pushed the release-please--branches--main--components--bobapp-frontend branch from f927bb6 to dd3e7f4 21 minutes ago Compare

Review requested

Review has been requested on this pull request. It is not required to merge. Learn more about requesting a pull request review.

1 pending review

cpierres was requested for review as a codeowner

No conflicts with base branch

Merging can be performed automatically.

Squash and merge You can also merge this with the command line. View command line instructions.

Add a comment

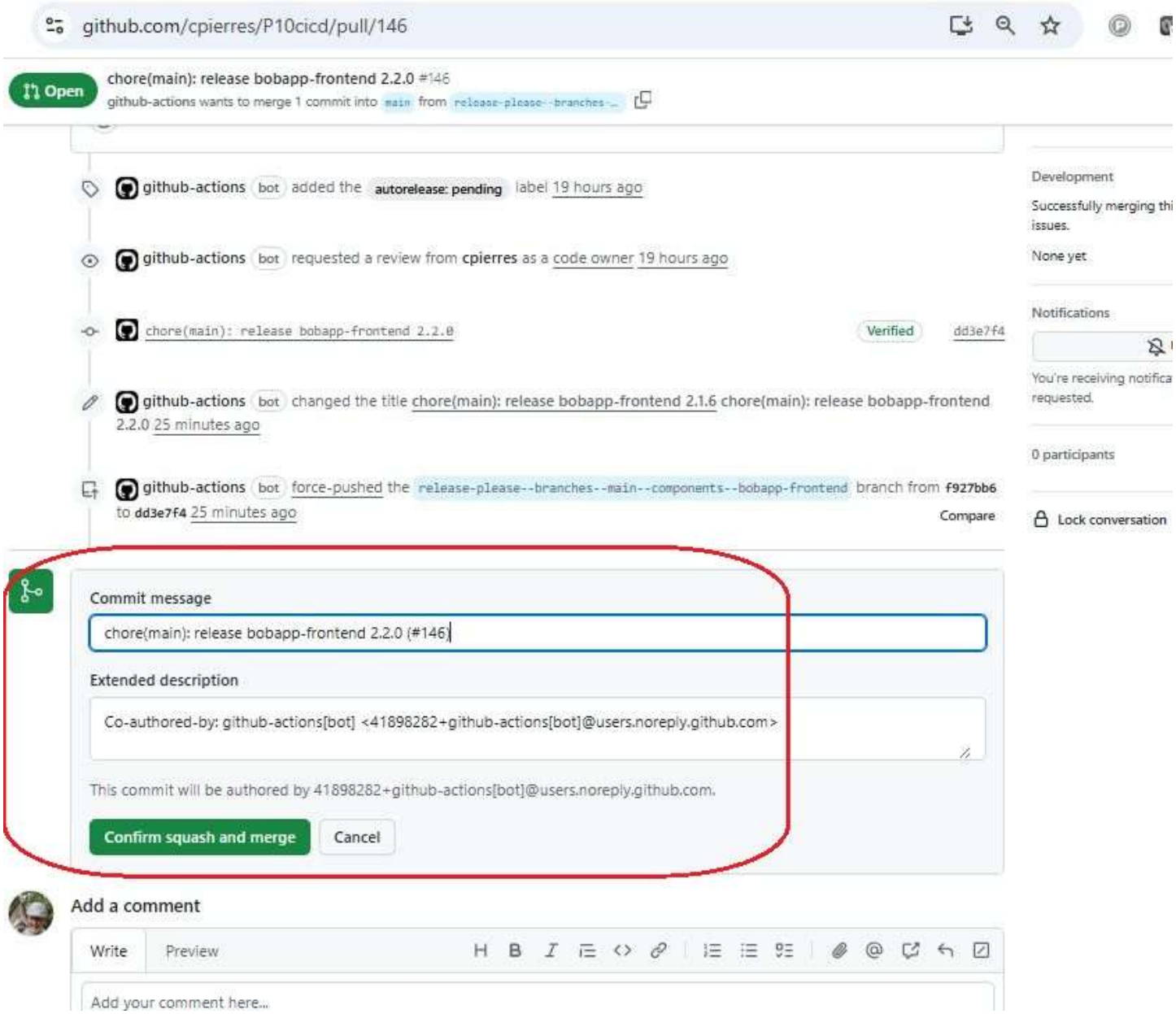
Write Preview

Add your comment here...

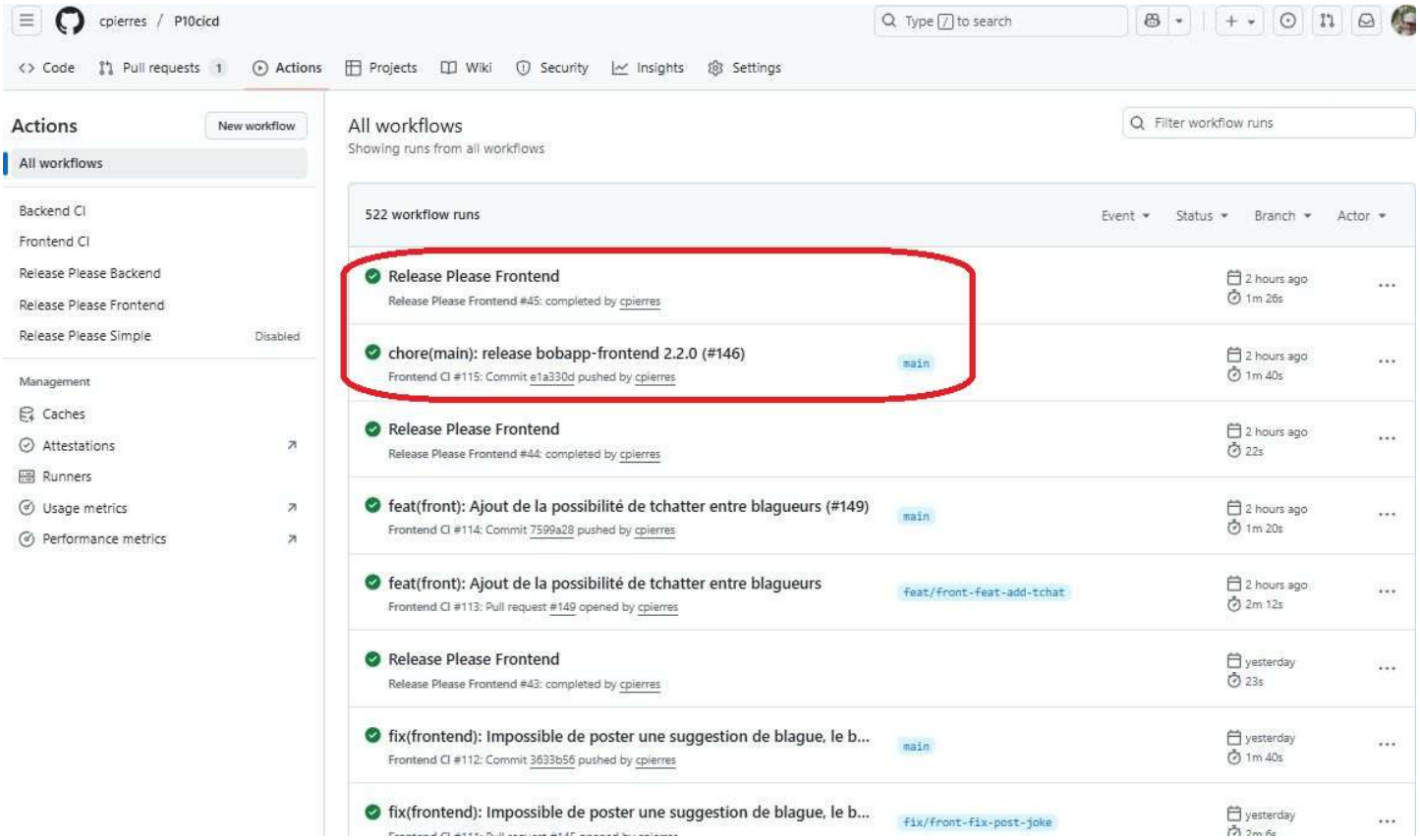
Markdown is supported Paste, drop, or click to add files

Close pull request Comment

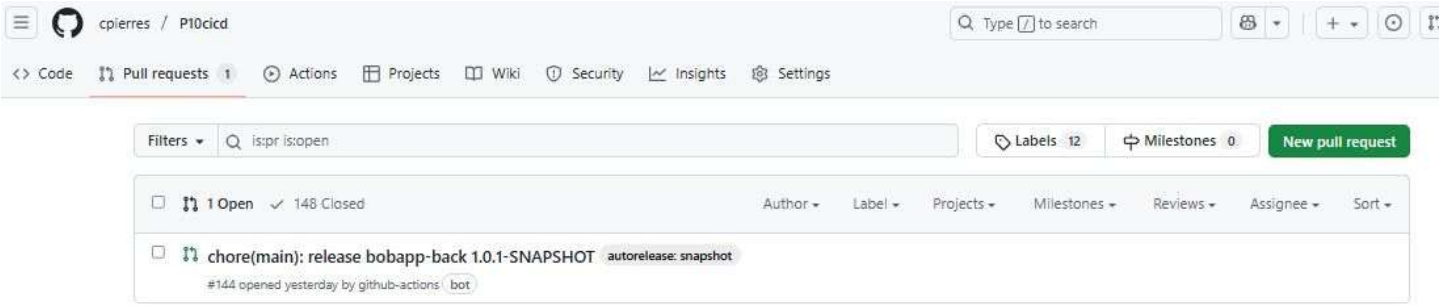
Une confirmation est demandée (possibilité de modifier le message de commit par défaut si besoin)



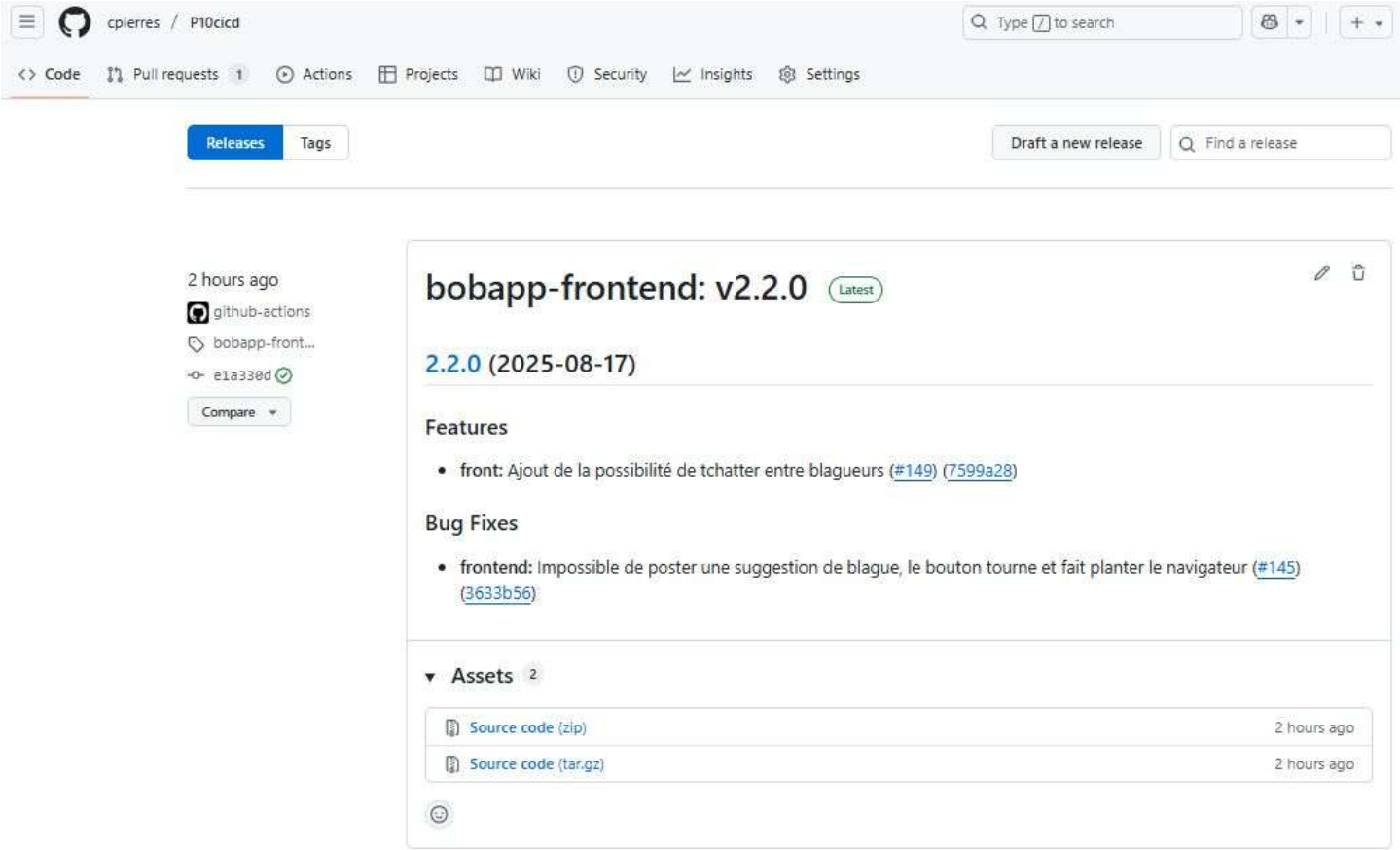
Nouveau cycle de frontend-ci et de Release Please Frontend sur main



Noter qu'il n'y a plus de PR de release (une nouvelle sera générée par Release Please Frontend lors de la première PR avec libellé de commit conventionnel)



Tag et release créés

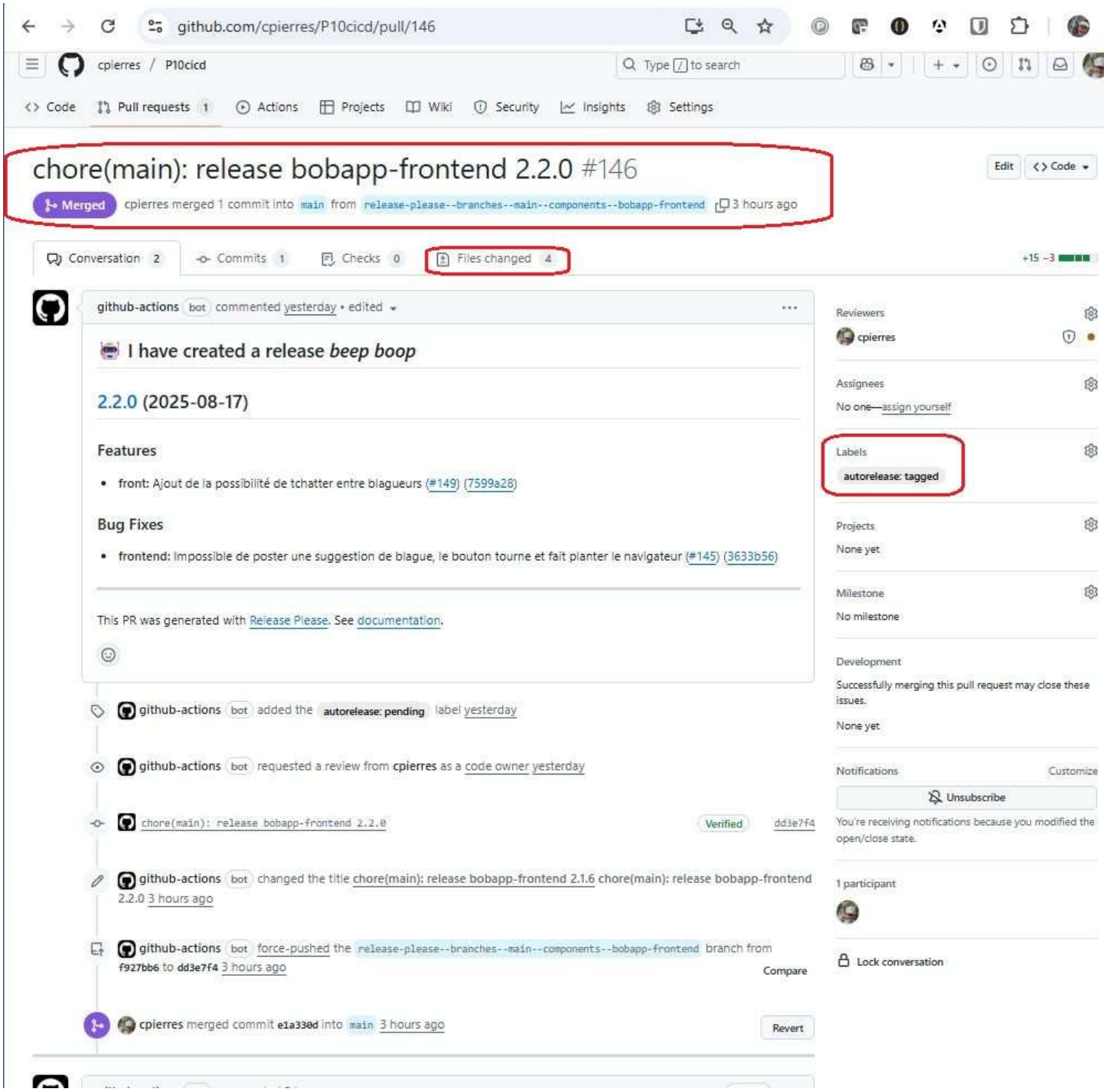


La PR de release a généré un commit pour bumper la version et mettre à jour CHANGELOG dans main (autorelease : tagged)

Bien penser à faire un update de votre `main` en local pour récupérer ce commit de bump+CHANGELOG.md.

Dans les Pull Request fermées, cliquer sur la PR de release suivante :

https://github.com/cpierres/P10cicd/pull/146



Commentaires pour documenter où se trouvent la release et les images docker

Des mails sont envoyés dans le même temps.

Merged

chore(main): release bobapp-frontend 2.2.0 #146

cpierres merged 1 commit into `main` from `release-please--branches-...` 3 hours ago

`chore(main): release bobapp-frontend 2.2.0` Verified dd3e7f4

github-actions bot changed the title chore(main): release bobapp-frontend 2.1.6 chore(main): release bobapp-frontend 2.2.0 3 hours ago

github-actions bot force-pushed the `release-please--branches--main--components--bobapp-frontend` branch from `f927bb6` to `dd3e7f4` 3 hours ago Compare

cpierres merged commit `e1a330d` into `main` 3 hours ago Revert

github-actions bot commented 3 hours ago Author

Release is at https://github.com/cpierres/P10cicd/releases/tag/bobapp-frontend-v2.2.0

github-actions bot added **autorelease: tagged** and removed **autorelease: pending** labels 3 hours ago

github-actions bot commented 3 hours ago Author

Images Docker Frontend publiées pour la release 2.2.0

- Pull (latest): `docker pull cpierres/bobapp-frontend:latest`
- Pull (version): `docker pull cpierres/bobapp-frontend:2.2.0`
- Lien Docker Hub: https://hub.docker.com/repository/docker/cpierres/bobapp-frontend/general

Pull request successfully merged and closed

You're all set — the `release-please--branches--main--components--bobapp-frontend` branch can be safely deleted.

Delete branch

Fichiers modifiés par le commit de release généré par la PR de release

cpierres / P10cicd

Code | Pull requests 1 | Actions | Projects | Wiki | Security | Insights | Settings

chore(main): release bobapp-frontend 2.2.0 #146

Merged cpierres merged 1 commit into main from release-please--branches--main--components--bobapp-frontend 3 hours ago

Conversation 2 | Commits 1 | Checks 0 | Files changed 4 +15 −3

Changes from all commits | File filter | Conversations | 0 / 4 files viewed | Ask Copilot | Review in codespace | Review changes

Filter changed files

- .github
  - .release-please-manifest-front...
- front
  - CHANGELOG.md
  - package-lock.json
  - package.json

.github/.release-please-manifest-frontend.json

```
@@ -1,3 +1,3 @@
 {
-  "front": "2.1.5"
+  "front": "2.2.0"
 }
```

front/CHANGELOG.md

```
@@ -1,5 +1,17 @@
 # Changelog

+ ## [2.2.0](https://github.com/cpierres/P10cicd/compare/bobapp-frontend-v2.1.5...bobapp-frontend-v2.2.0) (2025-08-17)
+
+
+ ### Features
+
+ * **front:** Ajout de la possibilité de tchatter entre blagueurs ([#149](https://github.com/cpierres/P10cicd/issues/149)) ([7599a28](https://github.com/cpierres/P10cicd/commit/7599a2896c0869d5aae443f50dc7cdda13992735))
+
+
+ ### Bug Fixes
+
+ * **frontend:** Impossible de poster une suggestion de blague, le bouton tourne et fait planter le navigateur ([#145](https://github.com/cpierres/P10cicd/issues/145)) ([3633b56](https://github.com/cpierres/P10cicd/commit/3633b5689544919bc7e9c24ec076d19e28bc7c6b))
+
 ## [2.1.5](https://github.com/cpierres/P10cicd/compare/bobapp-frontend-v2.1.4...bobapp-frontend-v2.1.5) (2025-08-15)
```

front/package-lock.json

Load diff

Some generated files are not rendered by default. Learn more about how customized files appear on GitHub.

Images docker créées pour bobapp-frontend

https://hub.docker.com/repository/docker/cpierres/bobapp-frontend/general

hub.docker.com/repository/docker/cpierres/bobapp-frontend/general

hub | Explore | My Hub | Search Docker Hub | CtrlK

cpierres
Docker Personal

- Repositories
- Collaborations
- Settings
  - Default privacy
  - Notifications
- Billing
- Usage
  - Pulls
  - Storage

Repositories / bobapp-frontend / General

Using 1 of 1 private repositories. Get more

cpierres/bobapp-frontend

Last pushed about 3 hours ago • Repository size: 467.2 MB

Add a description

Add a category

Docker commands

Public view

To push a new tag to this repository:

```
docker push cpierres/bobapp-frontend:tag name
```

General | Tags | Image Management BETA | Collaborators | Webhooks | Settings

Tags

DOCKER SCOUT INACTIVE
Activate

This repository contains 12 tag(s).

Tag	OS	Type	Pulled	Pushed
2.2.0		Image	less than 1 day	about 3 hours
latest		Image	less than 1 day	about 3 hours
pr-149-205e5a7		Image	less than 1 day	about 3 hours
pr-149		Image	less than 1 day	about 3 hours
pr-145-e9a0e35		Image	less than 1 day	about 22 hours

See all

buildcloud

Build with
Docker Build Cloud

Accelerate image build times with access to cloud-based builders and shared cache.

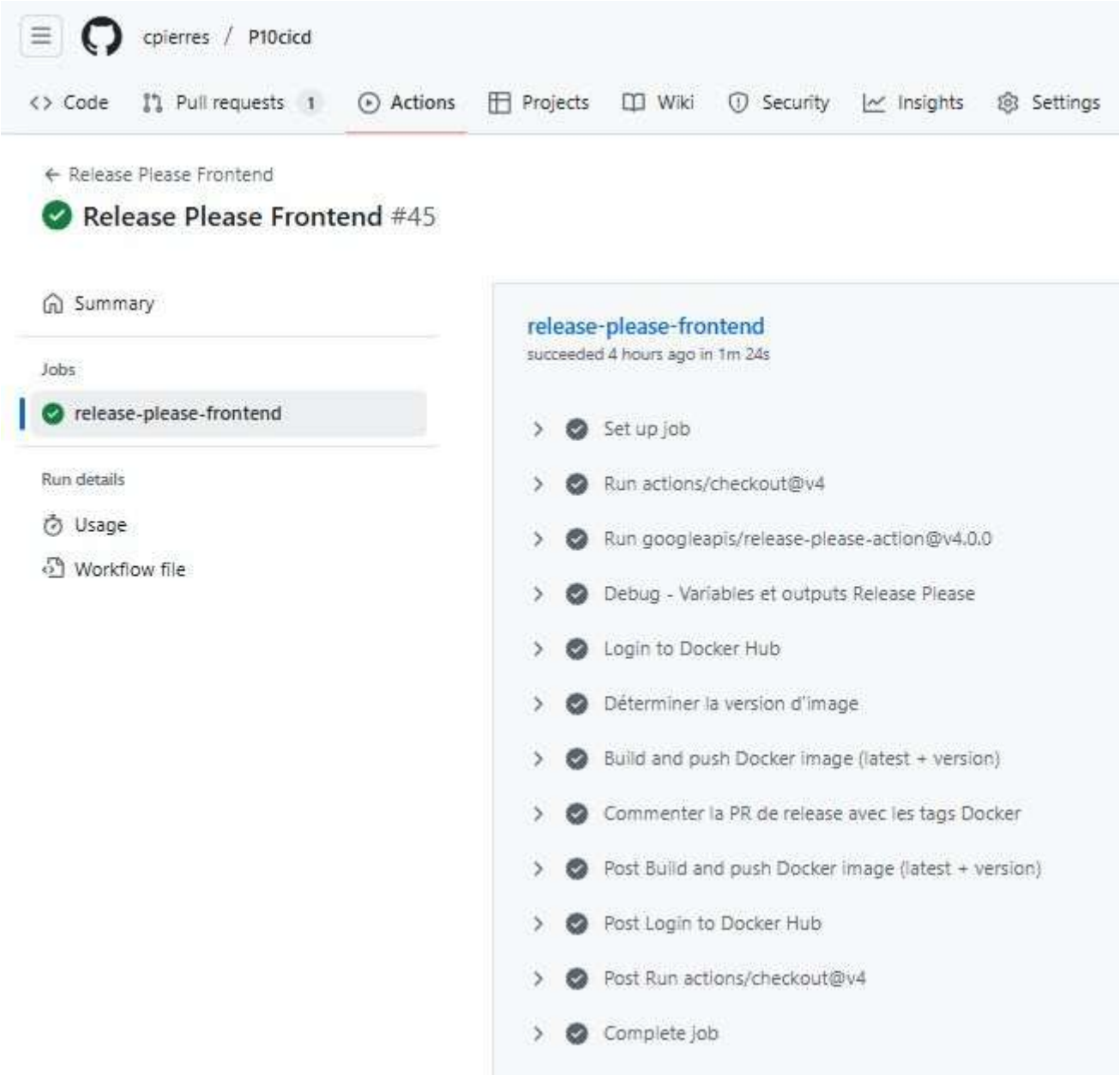
Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.

Get faster builds through shared caching across your team, native multi-platform support, and encrypted data transfer - all without managing infrastructure.

Go to Docker Build Cloud →

1.3.2.8. Détail des jobs du workflow release-please-frontend.yml

Les jobs de release-please-backend.yml sont similaires.



1.3.3. Reste à faire

1.3.3.1. Nettoyage des images pr-* une fois que la PR de feature ou fix est fermée/mergée

Les images dockers de pull request de validation doivent être supprimées dès que la PR est mergée.

1.3.3.2. Boucle de rétroaction avec les utilisateurs de bobapp

- créer une boucle de communication avec les utilisateurs ayant rapporté des bugs. Github possède des outils pour cela.

- créer un bouton/lien dans le front de bobapp afin de permettre à l'utilisateur de qualifier les issues (via un issue template)
- lors de la prise en compte du bug, envoyer à l'utilisateur un accusé de réception

- Nous avons déjà produit le changelog.md grâce à release-please, ce qui est déjà une bonne étape.

- Informer l'utilisateur de la résolution (fermer la boucle) via un webhook slack ou discord. Ou encore via un flux RSS auxquels les utilisateurs peuvent s'abonner.

Voir la section « Analyse des retours utilisateurs » / « Actions prioritaires » à la page 127.

2. Protection du dépôt Github

Ces sections expliquent les mécanismes de protection mis en place pour le dépôt BobApp, en se concentrant sur deux aspects complémentaires : le fichier CODEOWNERS et les règles de protection de branches.

Il s'agit principalement de protéger la branche `main`.

Pour des informations plus pratiques, se reporter à l'annexe : « Protections du dépôt (copies d'écran) » à la page 129.

Dans un but pratique, un pattern de branches `workflow-test-*` a également été ajouté.

Les objectifs sont de permettre au propriétaire du dépôt (et administrateurs) de :

- **tester les workflows rapidement par de simples push** sur ces branches dédiées, **sans nécessiter d'ouvrir/merger une PR**.

- Expérimenter des **évolutions du pipeline** sans impacter la branche `main`.

- Valider des **modifications de configuration** (p.ex. changement de version JDK/Node, options Maven/NPM, réglages JaCoCo/Karma, flags SonarQube) et observer les effets sur les jobs suivants (tests, sonar, docker) avant toute mise en production.

Bénéfices pratiques :

- Cycle d'itération plus **rapide** pour déboguer le YAML et phaser des changements CI.

- **Isolation** claire : aucune exécution sur `main` tant que la branche de test n'est pas fusionnée.

Bonnes pratiques d'utilisation :

- Préfixer de manière explicite :
  - `workflow-test-back-…` pour le backend
  - `workflow-test-front-…` pour le frontend
- Nommer la suite du nom avec la thématique du test.
- Nettoyer régulièrement les branches de test devenues obsolètes (note : je les ai gardées pour info)

2.1. Vue d'ensemble de la stratégie de protection

La sécurité et l'intégrité du code source sont assurées par une approche à deux niveaux :

1. fichier **CODEOWNERS** : Définit qui doit approuver les modifications via Pull Requests
2. **Protection de branches** : Contrôle comment les modifications peuvent être apportées aux branches importantes

Ces deux mécanismes fonctionnent en tandem pour créer un système de protection robuste garantissant que toutes les modifications sont correctement examinées et testées avant d'être intégrées au code de production.

2.2. Partie 1 : Le Fichier CODEOWNERS

Le fichier CODEOWNERS est une fonctionnalité de GitHub permettant de définir automatiquement le ou les propriétaires (reviewers) de différentes parties du code source. Lorsqu'une Pull Request (PR) est créée, GitHub assigne automatiquement les personnes ou équipes désignées comme propriétaires des fichiers modifiés en tant que reviewers.

2.2.1. Configuration

Le fichier CODEOWNERS de BobApp (situé dans `.github/CODEOWNERS`) est structuré comme suit :

```
# Le fichier CODEOWNERS lui-même ne peut être modifié que par l'administrateur
CODEOWNERS @cpierres

# Définit cpierres comme reviewer pour tous les fichiers du projet
* @cpierres
```

2.2.2. Fonctionnement avec les Pull Requests

1. Lorsqu'une PR est créée, GitHub analyse les fichiers modifiés
2. Pour chaque fichier, GitHub identifie les propriétaires définis dans CODEOWNERS
3. Ces propriétaires sont automatiquement ajoutés comme reviewers requis
4. La PR ne peut pas être fusionnée tant que les reviewers requis n'ont pas approuvé les modifications

2.2.3. Avantages de CODEOWNERS

1. **Gouvernance centralisée** :
   - Cohérence dans les revues de code
   - Contrôle strict sur toutes les modifications
   - Vision globale de l'évolution du projet

2. **Automatisation des assignations** :
   - Élimine le besoin d'assigner manuellement les reviewers
   - Garantit que les bonnes personnes examinent le code pertinent

3. **Documentation des responsabilités** :
   - Clarté sur qui est responsable de quelles parties du code
   - Référence utile pour les nouveaux contributeurs

4. **Intégration avec les workflows CI/CD** :
   - Complète la structure des workflows GitHub Actions séparés (backend/frontend)
   - Maintient une gouvernance unifiée sur ces workflows séparés

2.2.4. Limitations de CODEOWNERS

1. **Ne protège pas contre les push directs** :
   - S'applique uniquement aux Pull Requests
   - N'empêche pas les push directs sur les branches

2. **Ne protège pas directement les branches** :
   - Ne peut pas empêcher les modifications sur des branches spécifiques
   - Doit être complété par des règles de protection de branches

2.3. Partie 2 : Protection de Branches

Les règles de protection de branches sont des paramètres configurables dans GitHub qui contrôlent comment les modifications peuvent être apportées à des branches spécifiques. Elles permettent d'imposer certaines exigences avant qu'une branche puisse être modifiée ou fusionnée.

Vous trouverez ci-dessous les directives principales. Le détail et les copies d'écran se trouvent en annexe à la page 129.

2.3.1. Comment configurer les protections de branches

1. Accéder aux paramètres du dépôt GitHub
2. Sélectionner **Branches** dans le menu latéral
3. Cliquer sur **Add branch protection rule**
4. Définir le modèle de branche (ex: `main` ou `workflow-test-*`)
5. Configurer les options de protection souhaitées
6. Sauvegarder les règles

2.3.2. Configurations effectuées

2.3.2.1. Pour la branche `main`

- **Require a pull request before merging** :
  - Empêche les push directs sur la branche principale
  - Sous-option : « Require approvals » (au moins 1)
  - Sous-option : « Dismiss stale pull request approvals when new commits are pushed"

- **Require status checks to pass before merging** :
  - Exige que les workflows CI/CD réussissent
  - Checks spécifiques : « backend-tests", « frontend-tests"
  - Option : « Require branches to be up to date before merging"

- **Require conversation resolution before merging** :
  - Exige que tous les commentaires dans les discussions soient résolus

- **Include administrators** :
  - Applique ces règles même aux administrateurs du dépôt

2.3.2.2. Pour les branches `workflow-test-*`

Ce pattern de branches sert à tester les workflow des github actions. Ce pattern a été spécifié dans les workflows comme vous le constaterez prochainement. Pour la procédure détaillée, lire annexe « Protéger les branches workflow-test-* » à la page 132.

- **Require a pull request before merging**

- **Restrict creations/updates/deletions** + gestion des **droits de bypass** pour admin (je serai ainsi le seul à pouvoir mettre à jour cette branche « sensible » en tant que propriétaire du dépôt)

2.4. Partie 3 : Complémentarité des Mécanismes

2.4.1. Comment CODEOWNERS et la protection de branches se complètent

Aspect	CODEOWNERS	Protection de Branches
Champ d'application	Fichiers spécifiques	Branches entières
Moment d'action	Lors de la création de PR	Lors des push et des fusions
Type de protection	Gouvernance (qui approuve)	Technique (comment modifier)
Contournement possible	Push directs	Aucun (si bien configuré)

2.4.2. Scénarios de protection

1. **Modification via Pull Request** :
   - CODEOWNERS assigne automatiquement @cpierres comme reviewer
   - La protection de branches exige que la PR soit approuvée
   - Les checks CI/CD doivent réussir
   - Résultat : Modification sécurisée et validée

2. **Tentative de push direct sur `main`** :
   - CODEOWNERS n'intervient pas
   - La protection de branches bloque le push
   - Résultat : Modification rejetée

3. **Modification des workflows GitHub Actions** :
   - CODEOWNERS assigne @cpierres comme reviewer
   - La protection de branches exige une PR approuvée
   - Résultat : Changements critiques toujours vérifiés

2.5. Partie 4 : Considérations de Sécurité Spécifiques

2.5.1. Sécurisation des branches `workflow-test-*`

Les branches commençant par `workflow-test-*` méritent une attention particulière car :

1. Elles peuvent déclencher l'exécution des workflows GitHub Actions
2. Ces workflows ont accès aux secrets du dépôt
3. Elles pourraient potentiellement être utilisées pour exécuter du code malveillant

2.5.2. Risques et mitigations

Risque	Mitigation par CODEOWNERS	Mitigation par Protection de Branches
Modification des workflows	Revue obligatoire	Blocage des push directs
Accès aux secrets	Revue du code utilisant les secrets	Limitation de qui peut créer/modifier ces branches
Exécution de code arbitraire	Revue du code exécuté	Vérifications de statut obligatoires

2.6. Partie 5 : Recommandations d'évolutions pour BobApp

1. **Configuration CODEOWNERS** :
   - Maintenir la structure actuelle avec @cpierres comme propriétaire global
   - Envisager d'ajouter des propriétaires spécifiques si l'équipe s'agrandit

2. **Revue régulière** :
   - Auditer périodiquement les configurations de protection
   - Ajuster les règles en fonction de l'évolution du projet (qui s'occupe de quelle couche par exemple)

Avantages de cette approche combinée

1. **Sécurité renforcée** :
   - Protection complète contre les modifications non autorisées
   - Réduction des risques liés aux secrets et aux workflows

2. **Qualité du code améliorée** :
   - Revue systématique du code
   - Vérifications automatisées via CI/CD

3. **Gouvernance claire** :
   - Responsabilités bien définies
   - Processus de modification transparent

4. **Flexibilité maintenue** :
   - Workflows séparés pour le backend et le frontend
   - Possibilité d'ajuster les protections selon les besoins

2.7. Conclusion

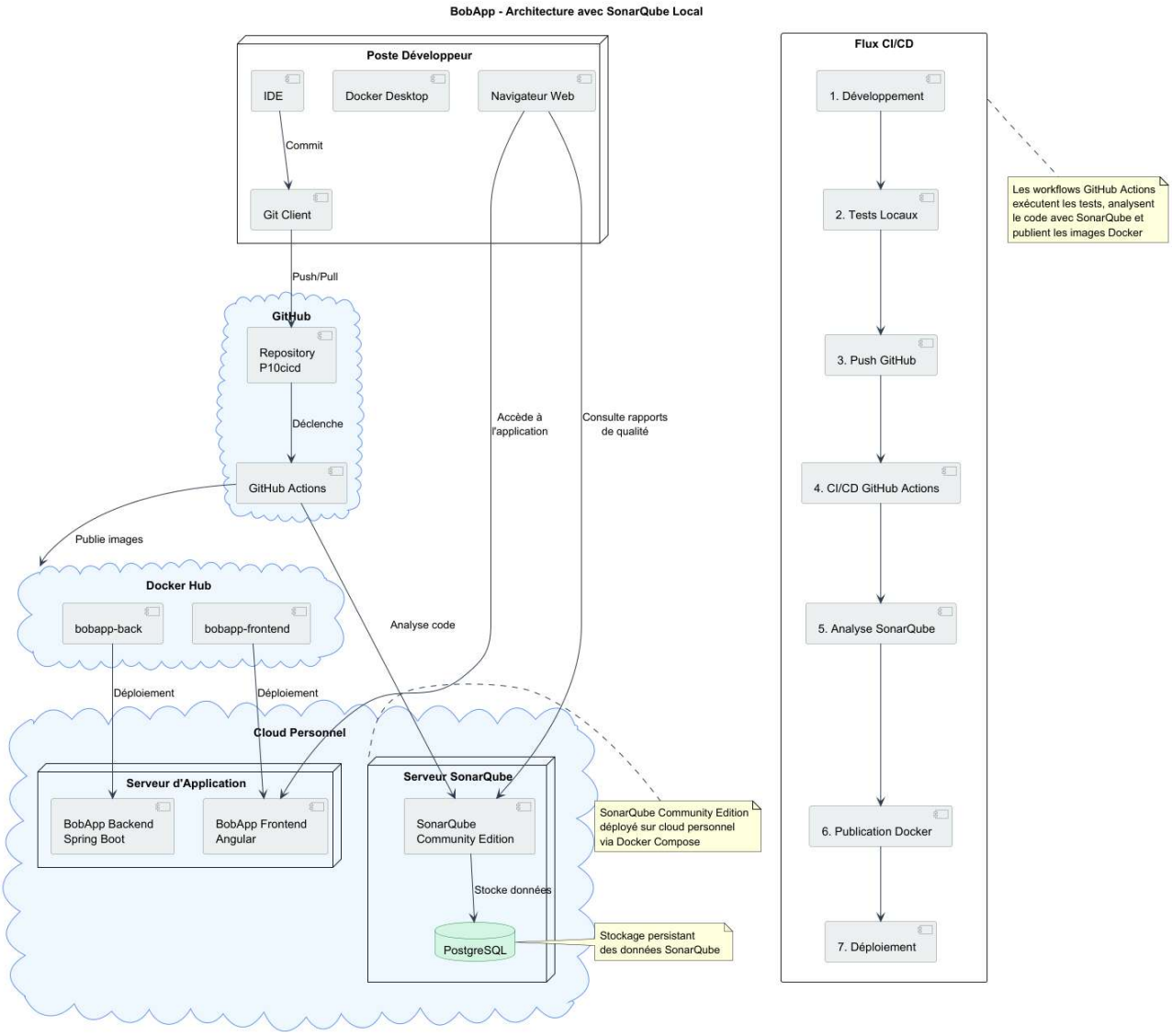
La combinaison du fichier CODEOWNERS et des règles de protection de branches offre une stratégie de protection complète pour le dépôt BobApp. Cette approche à deux niveaux garantit que :

1. Toutes les modifications sont correctement examinées (via CODEOWNERS)
2. Les branches critiques sont protégées contre les modifications directes (via la protection de branches)
3. Les workflows CI/CD sont sécurisés et ne peuvent être modifiés sans validation
4. La qualité et l'intégrité du code sont maintenues tout au long du cycle de développement

Cette stratégie s'aligne parfaitement avec la structure modulaire de l'application BobApp et ses workflows GitHub Actions séparés pour le backend et le frontend, tout en maintenant une gouvernance centralisée et sécurisée.

3. Architecture et structure

3.1. Architecture de la solution avec SonarQube community edition



3.1.1. Adressage SonarQube

J'ai exposé sur internet mon propre SonarQube (version comunity) plutôt qu'utiliser SonarCloud, ceci afin de mieux controler l'environnement, sans limitation (Quality Gates) :

https://sonarqube.cpierres.dscloud.me

Pour une raison que je pourrai vous indiquer sur demande, je ne pouvais pas utiliser l'accès anonyme.

Par conséquent, pour afficher les contrôles, veuillez utiliser le user suivant :

login : `ocrGuest`

password : `ocrGuest123!`

A la fin des jobs Sonar, vous trouverez les liens internet.

Deux projets sonarqube ont été créés, dédiés aux deux couches backend et frontend.

3.2. Description générale des workflows de github actions et projets SonarQube

Les workflows sont suffisamment commentés pour expliquer leur fonctionnement détaillé.

Les sections suivantes expliquent la structure et les avantages des workflows GitHub Actions et des projets SonarQube mis en place pour le projet BobApp, une application full-stack composée d'un backend Spring Boot et d'un frontend Angular.

3.2.1. Structure de base d'un workflow de github actions

Un fichier de workflow GitHub Actions suit cette structure générale :

```
name: Nom du Workflow

on:
  # Déclencheurs du workflow

jobs:
  job1:
    # Configuration du premier job
    runs-on: type-de-machine

    steps:
      # Liste des étapes à exécuter

  job2:
    # Configuration du deuxième job
    needs: [job1]
    runs-on: type-de-machine

    steps:
      # Liste des étapes à exécuter
```

3.2.2. Structure des Workflows de github actions

Le projet utilise deux workflows distincts :

- Backend CI Workflow (`.github/workflows/backend-ci.yml`)
- Frontend CI Workflow (`.github/workflows/frontend-ci.yml`)

Note : les workflows complémentaire de production de release sont décrits dans la première section de vue générale.

3.2.2.1. Backend CI Workflow (`.github/workflows/backend-ci.yml`)

Ce workflow est dédié à l'intégration continue du backend Spring Boot.

Il comprend trois jobs principaux, chacun de ces jobs incluant des steps :

1. **backend-tests** :
   - Exécute sur Ubuntu latest
   - Configure Java 11 (version requise par le projet)
   - Compile le code et exécute les tests unitaires
   - Génère des rapports de couverture de code avec JaCoCo
   - Publie les résultats des tests JUnit
   - Stocke les artefacts (rapports et classes compilées) pour les jobs suivants

2. **backend-sonar** :
   - Dépend de la réussite du job backend-tests
   - Configure Java 17 (requis pour SonarQube)
   - Récupère les artefacts générés par le job précédent

- Effectue l'analyse de qualité du code avec SonarQube
- Fournit un lien vers le rapport SonarQube
- Continue même en cas d'échec (`continue-on-error: true`)

3. **backend-docker** :
- S'exécute si les tests ont réussi, indépendamment du résultat de SonarQube
- Construit et publie l'image Docker du backend sur Docker Hub
- Ajoute un commentaire sur la PR avec les informations de l'image Docker (si applicable)

3.2.2.2. Frontend CI Workflow (`.github/workflows/frontend-ci.yml`)

Ce workflow est dédié à l'intégration continue du frontend Angular et suit une structure similaire :

1. **frontend-tests** :
- Exécute sur Ubuntu latest
- Configure Node.js 14 (version requise par le projet Angular)
- Installe les dépendances avec `npm ci`
- Exécute les tests Angular en mode headless avec couverture de code
- Stocke les rapports de couverture comme artefacts

2. **frontend-sonar** :
- Dépend de la réussite du job frontend-tests
- Récupère les artefacts de couverture générés précédemment
- Effectue l'analyse de qualité du code avec SonarQube
- Fournit un lien vers le rapport SonarQube
- Continue même en cas d'échec (`continue-on-error: true`)

3. **frontend-docker** :
- S'exécute si les tests ont réussi, indépendamment du résultat de SonarQube
- Configure Docker Buildx
- Construit et publie l'image Docker du frontend sur Docker Hub
- Ajoute un commentaire sur la PR avec les informations de l'image Docker (si applicable)

3.2.3. Déclencheurs des Workflows

Les deux workflows sont déclenchés dans les situations suivantes :
- Push sur la branche `main`
- Push sur les branches commençant par `workflow-test-*`(pour valider au plus tôt les tests sur les workflows eux-mêmes)
- Pull requests vers la branche `main`
- Déclenchement manuel via `workflow_dispatch`

Chaque workflow ne s'exécute que lorsque des fichiers pertinents sont modifiés :

- Le workflow backend se déclenche uniquement pour les modifications dans le dossier `back/` ou du fichier de workflow lui-même

- Le workflow frontend se déclenche uniquement pour les modifications dans le dossier `front/` ou du fichier de workflow lui-même

3.2.4. Structure des Projets SonarQube correspondants

De manière similaire aux workflows GitHub Actions, l'analyse de qualité du code a été divisée en deux projets SonarQube distincts :

1. **Projet Backend** :
   - Clé du projet : `cpierres_P10cicd-backend`
   - Analyse le code Java du backend Spring Boot
   - Utilise un token d'authentification spécifique (SONARQUBE_TOKEN_BACKEND)
   - URL du tableau de bord : https://sonarqube.dscloud.me/dashboard?id=cpierres_P10cicd-backend

2. **Projet Frontend** :
   - Clé du projet : `cpierres_P10cicd-frontend`
   - Analyse le code TypeScript/JavaScript du frontend Angular
   - Utilise un token d'authentification spécifique (SONARQUBE_TOKEN_FRONTEND)
   - URL du tableau de bord : https://sonarqube.dscloud.me/dashboard?id=cpierres_P10cicd-frontend

Ces projets ont été créés manuellement dans l'instance SonarQube hébergée à l'adresse https://sonarqube.dscloud.me.

3.2.5. Avantages de la Séparation en deux Workflows

1. **Efficacité des ressources** :
   - Seule la partie concernée par les modifications est testée et déployée
   - Évite d'exécuter des tests inutiles (par exemple, ne pas tester le frontend si seul le backend a été modifié)
   - Réduit le temps d'exécution global et la consommation de ressources GitHub Actions

2. **Parallélisation** :
   - Les workflows peuvent s'exécuter en parallèle si les deux parties sont modifiées simultanément
   - Réduit le temps d'attente pour les développeurs

3. **Isolation des problèmes** :
   - Un échec dans une partie n'affecte pas l'autre
   - Facilite l'identification et la résolution des problèmes
   - Permet de continuer à déployer une partie même si l'autre rencontre des problèmes

4. **Spécialisation des environnements** :
   - Chaque workflow utilise l'environnement adapté à sa technologie (Java 11 pour le backend, Node.js 14 pour le frontend)

- Permet d'optimiser les configurations spécifiques à chaque stack technique

5. **Maintenance simplifiée** :
   - Les workflows sont plus courts et plus faciles à comprendre
   - Les modifications d'un workflow n'affectent pas l'autre
   - Facilite l'évolution indépendante des processus CI/CD pour chaque partie

6. **Gestion des dépendances** :
   - Utilisation de caches spécifiques (Maven pour le backend, npm pour le frontend)
   - Optimisation des temps de téléchargement et d'installation des dépendances

7. **Flexibilité des déploiements** :
   - Possibilité de déployer indépendamment le backend et le frontend
   - Adapté à une architecture de microservices ou de services découplés

3.2.6. Avantages Spécifiques de la Séparation des Projets SonarQube

1. **Analyse adaptée aux technologies** :
   - Chaque projet SonarQube peut être configuré avec des règles d'analyse spécifiques à sa technologie
   - Les seuils de qualité peuvent être ajustés indépendamment pour le backend et le frontend
   - Utilisation de plugins spécifiques à chaque langage (Java pour le backend, TypeScript/JavaScript pour le frontend)

2. **Visualisation claire des métriques** :
   - Tableaux de bord distincts pour le backend et le frontend
   - Évite la dilution des métriques (par exemple, une faible couverture dans une partie ne masque pas une bonne couverture dans l'autre)
   - Facilite la comparaison des métriques entre les différentes parties du projet

3. **Gestion des accès et des permissions** :
   - Possibilité d'attribuer des droits d'accès différents selon les équipes (équipe backend vs équipe frontend)
   - Tokens d'authentification distincts pour chaque projet, limitant l'impact en cas de compromission

4. **Suivi indépendant de la dette technique** :
   - Chaque équipe peut suivre et gérer sa propre dette technique
   - Les objectifs de qualité peuvent être définis et suivis séparément

5. **Optimisation des analyses** :
   - Analyses plus rapides car traitant moins de code à la fois
   - Réduction de la consommation de ressources sur le serveur SonarQube
   - Possibilité de planifier des analyses à des fréquences différentes selon les besoins

6. **Historique et évolution distincts** :

- Suivi indépendant de l'évolution de la qualité pour chaque partie
- Les modifications majeures dans une partie n'affectent pas l'historique de l'autre

7. **Intégration ciblée avec les workflows CI/CD** :

- Chaque projet SonarQube s'intègre naturellement avec son workflow CI/CD correspondant
- Les résultats d'analyse sont plus pertinents car directement liés aux modifications en cours

Cette approche de séparation des workflows et des projets SonarQube reflète la nature modulaire de l'application BobApps.

3.2.1. Attention particulière au respect de l'environnement précis de chacun des projets

Dans la structuration des workflows, Une attention particulière a été portée pour bien respecter l'environnement fourni par chaque projet.

3.2.1.1. Pourquoi respecter les environnements spécifiques est crucial

Le respect des environnements spécifiques pour chaque projet (JDK 11 pour le backend Java et Node 14 pour le frontend Angular) est fondamental pour plusieurs raisons essentielles :

1. **Compatibilité et comportement prévisible** :

- Le backend Spring Boot 2.6.1 est conçu pour fonctionner avec Java 11. L'utilisation d'une version différente pourrait introduire des incompatibilités subtiles ou des comportements inattendus.
- Angular 14.2.0 est optimisé pour Node 14. Des versions plus récentes ou plus anciennes peuvent causer des problèmes de compatibilité avec certaines dépendances ou fonctionnalités.

2. **Reproduction fidèle de l'environnement de production** :

- Les workflows CI/CD doivent reproduire aussi fidèlement que possible l'environnement de production pour garantir que les tests reflètent le comportement réel de l'application.
- Si le code est compilé avec JDK 17 mais déployé avec JDK 11, des problèmes peuvent survenir uniquement en production.

3. **Validation de la compatibilité des dépendances** :

- Certaines bibliothèques peuvent avoir des comportements différents selon la version de JDK ou de Node.js utilisée.
- Tester avec la version cible garantit que toutes les dépendances fonctionnent correctement ensemble.

4. **Cohérence entre les environnements de développement et CI/CD** :

- Les développeurs utilisent JDK 11 et Node 14 localement, donc les workflows CI/CD doivent utiliser les mêmes versions pour éviter le syndrome « ça marche sur ma machine ».

3.2.1.2. Impact sur la structuration des jobs et le partage d'artefacts

Cette exigence de respecter des environnements spécifiques a directement influencé la conception des workflows GitHub Actions :

1. **Séparation des jobs par préoccupation** :

- Dans le workflow backend, un job `backend-tests` utilise JDK 11 pour la compilation et les tests, tandis qu'un job distinct `backend-sonar` utilise JDK 17 pour l'analyse SonarQube.
- De même, le workflow frontend sépare les tests (avec Node 14) de l'analyse SonarQube.

2. **Partage d'artefacts entre jobs** :

- Les classes compilées avec JDK 11 sont partagées via des artefacts pour éviter de recompiler avec JDK17:

```
  - name: Upload backend-classes
    uses: actions/upload-artifact@v4
    with:
      name: backend-classes
      path: |
        back/target/classes
        back/target/test-classes
```

- Les rapports de couverture générés pendant les tests sont également partagés :

```
  - name: Upload JaCoCo coverage report
    uses: actions/upload-artifact@v4
    with:
      name: jacoco-report
      path: back/target/site/jacoco/
```

- Ces artefacts sont ensuite téléchargés dans le job SonarQube :

```
  - name: Téléchargement des classes backend
    uses: actions/download-artifact@v4
    with:
      name: backend-classes
      path: back/target/
```

3. **Optimisation du cache Maven** :

- Le cache Maven est partagé entre les jobs via le paramètre `cache-dependency-path: '/pom.xml'` pour éviter de télécharger plusieurs fois les dépendances.

4. **Exécution conditionnelle des jobs** :

- Le job Docker dépend des résultats des tests mais pas nécessairement de SonarQube, permettant de continuer le déploiement même si l'analyse de qualité échoue.

3.2.1.3. Ce qu'il ne faudrait pas faire dans un souci de simplification

Ce qui doit être éviter, c'est cela :

```
test:
  needs: build
  runs-on: ubuntu-latest
  defaults:
    run:
      working-directory: ./back

  steps:
  - name: Checkout
    uses: actions/checkout@v4

  - name: Set up JDK 17
    uses: actions/setup-java@v4
    with:
      java-version: '17'
      distribution: 'temurin'
      cache: maven
```

```
  - name: Run tests
    run: mvn clean test

  - name: JaCoCo coverage report
    uses: actions/upload-artifact@v4
    with:
      name: coverage_backend
      path: back/target/site/jacoco

  - name: Build and analyze
    env:
      SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
    run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=
cpierres_P10cicd-backend
```

Très mauvais : dans un job `test` unique, on mélange des steps de build et tests du projet cible avec le contrôle qualité sonar en utilisant du coup un JDK qui est obligatoire pour Sonar mais pas au projet !

Ne pas compiler et tester le backend avec JDK 17 simplement parce qu'il est requis par SonarQube

Utiliser JDK 17 pour la compilation et les tests du backend (qui requiert JDK 11) dans un job unique serait problématique pour plusieurs raisons :

- **Risque de faux positifs/négatifs dans les tests** : Le code pourrait passer les tests avec JDK 17 mais échouer en production avec JDK 11, ou inversement.

- **Utilisation de fonctionnalités non disponibles** : JDK 17 offre des fonctionnalités et des API qui n'existent pas dans JDK 11. Le code pourrait accidentellement utiliser ces fonctionnalités et compiler avec JDK 17, mais échouer en production.

- **Comportements différents du compilateur** : Le compilateur Java 17 peut être plus strict ou plus permissif sur certains aspects que Java 11, masquant potentiellement des problèmes qui apparaîtraient en production.

- **Optimisations spécifiques à la version** : Les optimisations JIT et autres améliorations de performance diffèrent entre JDK 11 et 17, pouvant masquer des problèmes de performance.

Ne pas simplifier le workflow frontend de manière similaire

Pour le frontend, utiliser une version de Node.js différente de celle recommandée pour Angular 14 présenterait des risques similaires :

- **Incompatibilités avec les dépendances** : Certains packages npm peuvent avoir des comportements différents ou des bugs selon la version de Node.js.

- **Différences dans le moteur JavaScript** : Les différentes versions de Node.js utilisent différentes versions de V8, ce qui peut affecter l'exécution du code.

- **Problèmes avec les outils de build** : Angular CLI et ses dépendances peuvent avoir des comportements différents selon la version de Node.js.

3.2.1.4. Conclusion

La séparation des jobs par environnement spécifique et le partage d'artefacts entre eux représentent une approche robuste qui garantit :

1. La compilation et les tests dans l'environnement cible exact (JDK 11, Node 14)
2. L'analyse de qualité du code avec les outils requis (SonarQube avec JDK 17)
3. Une meilleure isolation des problèmes et identification des erreurs
4. Une reproduction fidèle de l'environnement de production

Cette approche, bien que semblant plus complexe au premier abord, offre en réalité une plus grande fiabilité et prévisibilité du pipeline CI/CD, réduisant les risques de problèmes en production et facilitant le diagnostic des erreurs.

3.3. Fichiers docker-compose présents dans le projet

Cette section complémente les explications déjà données dans le README.md en expliquant les différents fichiers Docker Compose présents dans le projet BobApp, leur configuration et leur intérêt dans le cadre du développement et du déploiement de l'application.

3.3.1. Vue d'ensemble

Le projet BobApp utilise trois fichiers Docker Compose distincts, chacun ayant un rôle spécifique :

1. **docker-compose.yml** - Configuration principale pour le développement local
2. **docker-compose-prebuilt.yml** - Configuration utilisant les images précompilées depuis Docker Hub
3. **docker-compose-sonar.yml** - Configuration pour l'analyse de qualité du code avec SonarQube

3.3.2. docker-compose.yml

Objectif : Construire et exécuter l'application complète (frontend et backend) à partir du code source local.

Intérêt :

- Permet aux développeurs de tester l'application complète dans un environnement conteneurisé
- Construit les images Docker à partir du code source local (utile pendant le développement)
- Crée un réseau dédié pour la communication entre les services

Configuration :

- **bobapp-back** : Service backend Spring Boot
  - Construit l'image à partir du contexte `./back`
  - Expose le port 8080
  - Redémarre automatiquement sauf en cas d'arrêt manuel

- **bobapp-front** : Service frontend Angular
  - Construit l'image à partir du contexte `./front`
  - Expose le port 80
  - Dépend du service backend
  - Redémarre automatiquement sauf en cas d'arrêt manuel

- **Réseau** : Crée un réseau nommé `bobapp-network` pour la communication entre les services

Commandes d'utilisation :

```
# Démarrer avec un nom de projet spécifique
docker-compose -p cpierres-bobapp up -d

# Arrêter les services
docker-compose -p cpierres-bobapp down
```

3.3.3. docker-compose-prebuilt.yml

Objectif : Déployer rapidement l'application en utilisant des images précompilées depuis Docker Hub.

Intérêt :

- Accélère le déploiement en évitant l'étape de construction des images
- Permet de tester les images exactes qui ont été validées par le pipeline CI/CD
- Facilite le déploiement dans des environnements sans outils de build

Configuration :

- **bobapp-back** : Service backend Spring Boot
  - Utilise l'image précompilée `cpierres/bobapp-back:latest`
  - Configuration réseau et ports identiques à docker-compose.yml

- **bobapp-front** : Service frontend Angular
  - Utilise l'image précompilée `cpierres/bobapp-frontend:latest`
  - Configuration réseau et ports identiques à docker-compose.yml

Commandes d'utilisation :

```
# Démarrer avec les images précompilées
docker-compose -f docker-compose-prebuilt.yml -p cpierres-bobapp-prebuilt up -d

# Arrêter les services
docker-compose -f docker-compose-prebuilt.yml -p cpierres-bobapp-prebuilt down
```

3.3.4. docker-compose-sonar.yml

Objectif : Fournir un environnement d'analyse de qualité du code avec SonarQube. N’a servi qu’à préparer le serveur SonarQube interne déployé sur internet (via reverse proxy).

Intérêt :

- Permet d'analyser la qualité du code source localement
- Facilite l'identification des problèmes de code avant l'intégration
- Offre une interface web pour visualiser les rapports d'analyse

Configuration :

- **sonarqube** : Service SonarQube Community Edition
  - Utilise l'image `sonarqube:25.7.0.110598-community`
  - Expose le port 9000 pour l'interface web

- Configure la connexion à la base de données PostgreSQL
- Utilise des volumes persistants pour les données, logs et extensions

- **db** : Service PostgreSQL pour SonarQube
- Utilise l'image `postgres:15`
- Configure les identifiants et la base de données pour SonarQube
- Utilise un volume persistant pour les données

Commandes d'utilisation :

```
# Démarrer SonarQube
docker-compose -f docker-compose-sonar.yml -p sonar up -d

# Arrêter SonarQube
docker-compose -f docker-compose-sonar.yml -p sonar down
```

3.3.5. Informations générales

1. **Isolation des environnements** : L'utilisation du paramètre `-p` (nom de projet) permet d'isoler complètement les différentes stacks et d'éviter tout conflit entre elles, même si elles sont exécutées simultanément.

2. **Volumes persistants** : Les volumes définis dans docker-compose-sonar.yml assurent la persistance des données entre les redémarrages. On pourrait aussi donner la possibilité d'externaliser les volumes sur la machine hôte.

3. **Dépendances entre services** : La directive `depends_on` garantit que les services démarrent dans le bon ordre.

4. **Réseaux dédiés** : Chaque stack utilise son propre réseau pour isoler les communications.

5. **Politique de redémarrage** : La configuration `restart: unless-stopped` assure que les services redémarrent automatiquement en cas de panne, sauf si l'utilisateur les arrête explicitement.

Cette organisation en plusieurs fichiers Docker Compose permet une flexibilité dans le développement, les tests et le déploiement de l'application BobApp, tout en maintenant une séparation claire des préoccupations pour chaque environnement.

4. Contrôle qualité avec SonarQube

4.1. Intérêt des Quality Gates de SonarQube

4.1.1. Qu'est-ce qu'une Quality Gate ?

Une Quality Gate dans SonarQube est un ensemble de conditions de qualité que le code doit satisfaire pour être considéré comme « prêt à être livré ». Elle agit comme un garde-fou qui empêche l'intégration de code ne respectant pas les standards de qualité définis.

4.1.2. Avantages des Quality Gates

1. **Maintien de la qualité du code** :
   - Établit un seuil minimal de qualité pour tout le code intégré
   - Prévient la dégradation progressive de la qualité (dette technique)
   - Garantit la cohérence des standards de qualité dans le temps

2. **Détection précoce des problèmes** :
   - Identifie les problèmes avant qu'ils n'atteignent la production
   - Réduit le coût de correction des bugs (plus tôt = moins cher)
   - Évite l'accumulation de problèmes difficiles à résoudre

3. **Objectivité dans l'évaluation du code** :
   - Fournit des métriques objectives et quantifiables
   - Élimine les désaccords subjectifs sur la qualité du code
   - Permet de suivre l'évolution de la qualité au fil du temps

4. **Automatisation du processus de qualité** :
   - S'intègre parfaitement aux workflows CI/CD
   - Réduit le besoin de revues manuelles pour les aspects basiques
   - Libère du temps pour se concentrer sur des aspects plus complexes

5. **Éducation des développeurs** :
   - Sensibilise l'équipe aux bonnes pratiques
   - Fournit un feedback immédiat sur la qualité du code
   - Encourage l'amélioration continue des compétences

4.2. Métriques clés des Quality Gates

4.2.1. Taux de couverture (Coverage)

Le taux de couverture mesure le pourcentage de code exécuté pendant les tests automatisés.

Pourquoi c'est important :

- **Fiabilité** : Un code bien testé est généralement plus fiable
- **Confiance dans les refactorisations** : Les tests permettent de détecter rapidement les régressions
- **Documentation vivante** : Les tests illustrent le comportement attendu du code

Recommandation : Un taux de couverture minimum de 70% est un bon point de départ, offrant un équilibre entre effort et bénéfice. Pour les parties critiques du code, une couverture plus élevée (80-90%) peut être envisagée.

4.2.2. New Blocker severity Issues (Nouveaux problèmes bloquants)

Les problèmes bloquants sont des défauts critiques qui présentent un risque élevé pour l'application.

Exemples de problèmes bloquants :

- Failles de sécurité critiques (injections SQL, XSS, etc.)
- Fuites de mémoire
- Exceptions non gérées dans des chemins critiques
- Code mort ou inaccessible
- Utilisation de méthodes dépréciées critiques

Pourquoi c'est important :

- **Sécurité** : Prévient l'introduction de vulnérabilités
- **Stabilité** : Évite les crashs et comportements imprévisibles
- **Maintenabilité** : Empêche l'accumulation de code problématique

Recommandation : La tolérance pour les nouveaux problèmes bloquants devrait être zéro. Tout nouveau problème bloquant devrait faire échouer la Quality Gate.

4.2.3. Autres métriques importantes

1. **Duplication de code** :
   - Mesure le pourcentage de code dupliqué
   - Recommandation : Limiter à 5% maximum

2. **Dette technique** :
   - Mesure le temps estimé pour corriger tous les problèmes
   - Recommandation : Limiter l'ajout de nouvelle dette technique

3. **Complexité cognitive** :
   - Mesure la difficulté à comprendre le code
   - Recommandation : Limiter la complexité des nouvelles méthodes

4. **Bugs et Vulnérabilités** :

- Détecte les problèmes potentiels de sécurité et de fiabilité
- Recommandation : Zéro nouveau bug ou vulnérabilité de gravité élevée

4.3. Quality Gates recommandées pour BobApp

4.3.1. Stratégie différenciée et équilibrée

Nous allons établir des critères distincts pour le Nouveau Code et le Code Global, en adoptant une approche stratégique et équilibrée pour la gestion de la qualité du code pour ce projet BobApp.

La section « Différences entre « **New Code** » et « **Overall Code** » dans SonarQube » à la page 64 peut être lue en préalable.

Voici une justification détaillée de nos choix.

L'approche différenciée entre Nouveau Code et Code Global s'appuie sur plusieurs principes fondamentaux :

1. **Amélioration progressive** : En étant plus strict avec le nouveau code tout en étant plus tolérant avec le code existant, nous permettons une amélioration graduelle de la qualité globale sans bloquer le développement.

2. **Réalisme opérationnel** : Reconnaître que la correction immédiate de tous les problèmes dans une base de code existante est souvent irréaliste et pourrait paralyser le développement.

3. **Responsabilisation des développeurs** : Les développeurs sont tenus à des standards élevés pour leur nouveau code, ce qui encourage l'apprentissage et l'amélioration continue.

4. **Visibilité double** : Maintenir une vue sur la qualité du nouveau code ET sur l'état global du projet permet de suivre à la fois les améliorations immédiates et les tendances à long terme.

4.3.1.1. Avantages spécifiques de cette approche pour le projet BobApp

1. **Adaptation à la maturité du projet** :

- Si BobApp est un projet existant avec du code legacy, cette approche permet de continuer le développement tout en améliorant progressivement la qualité.
- Si BobApp est un nouveau projet, ces standards établissent dès le départ une base solide tout en reconnaissant que des compromis peuvent être nécessaires dans certaines phases.

2. **Équilibre entre rigueur et pragmatisme** :

- Les seuils ERROR pour le nouveau code garantissent que la qualité ne se dégrade pas.
- Les seuils WARNING pour le code global permettent une visibilité sur les problèmes sans bloquer le pipeline.

3. **Priorisation intelligente** :

- La sévérité ERROR sur la sécurité pour le code global force l'équipe à traiter immédiatement les vulnérabilités.
- La tolérance plus élevée pour les problèmes de maintenabilité permet de prioriser les corrections.

4. **Alignement avec les workflows CI/CD** :

- Cette configuration s'intègre parfaitement avec la séparation des workflows backend/frontend.
- Les jobs peuvent continuer même si les analyses SonarQube du code global génèrent des warnings.

4.3.1.2. Stratégie d'évolution des seuils

Cette configuration n'est pas statique et devrait évoluer avec la maturité du projet :

1. **Phase initiale** (actuelle) : Différenciation marquée entre nouveau code et code global.

2. **Phase intermédiaire** : À mesure que la qualité globale s'améliore, les seuils pour le code global peuvent être progressivement resserrés.

3. **Phase mature** : À terme, les seuils pour le nouveau code et le code global pourraient converger, signalant une base de code uniformément de haute qualité.

4.3.1.3. Conclusion

L'approche différenciée des Quality Gates pour le Nouveau Code et le Code Global représente une stratégie équilibrée qui :

- Garantit que la qualité du code s'améliore progressivement
- Reconnaît les réalités pratiques du développement logiciel
- Établit des priorités claires pour la correction des problèmes
- S'aligne avec la philosophie DevOps d'amélioration continue

Cette configuration permet à l'équipe BobApp de maintenir un rythme de développement soutenu tout en améliorant constamment la qualité du code, créant ainsi un cercle vertueux où chaque nouvelle contribution renforce la robustesse et la maintenabilité de l'application.

4.3.2. Quality Gates définies pour le projet Backend (Java/Spring Boot)

4.3.2.1. Justification des métriques et seuils spécifiques

Métrique	Nouveau Code	Code Global	Justification
Coverage	80% (ERROR)	70% (WARNING)	**Nouveau Code**: Le seuil de 80% représente un équilibre entre rigueur et faisabilité. Il est suffisamment élevé pour garantir que la majorité du code est testée, mais pas trop contraignant pour bloquer le développement. **Code Global**: Le seuil de 70% reconnaît la réalité du code legacy qui peut avoir été développé sans tests suffisants. C'est un niveau minimal acceptable qui signale un besoin d'amélioration sans bloquer le pipeline.
Issues	0	Non défini	**Nouveau Code**: politique de "zéro défaut" pour le nouveau code. Encourage les développeurs à

Métrique	Nouveau Code	Code Global	Justification
			résoudre tous les problèmes avant de fusionner leur code. Évite l'accumulation progressive de dette technique **Code Global**: pour le code global (le « code d'avant » le contrôle qualité), on est moins sévère et plus fin dans les catégories de Issues.
Blocker severity Issues	Non défini	5 (WARNING)	**Code Global**: La limite de 5 reconnaît que certains problèmes bloquants peuvent exister dans le code legacy et nécessiter une planification pour être corrigés. Exemples en annexe page 164.
High severity Issues (anciennement Critical Issues)	Non défini	10 (WARNING)	**Nouveau Code**: Aucun nouveau problème critique ne devrait être introduit, conformément au principe de « ne pas aggraver la situation ». Voir Issues ci-dessus. **Code Global**: La limite de 10 permet de prioriser la correction des problèmes critiques existants sans bloquer le développement. Exemples en annexe page 165.
Medium et Low severity Issues (anciennement Code Smells)	Non défini	Non défini	**Nouveau Code**: Une petite tolérance pour les Medium et Low severity Issues reconnaît que certaines pratiques peuvent être justifiées dans des contextes spécifiques. **Code Global**: Non défini car déjà couvert implicitement par le Maintainability Rating.
Duplicated Lines	5% (WARNING)	10% (WARNING)	**Nouveau Code**: Un faible niveau de duplication est toléré pour le nouveau code, reconnaissant que certaines duplications peuvent être intentionnelles **Code Global**: Un seuil plus élevé pour le code existant permet une correction progressive de la duplication.
Maintainability Rating	A (ERROR)	B (WARNING)	**Nouveau Code**: Exiger un rating A garantit que le nouveau code est hautement maintenable **Code Global**: Un rating B est acceptable pour le code existant, signalant un niveau raisonnable de maintenabilité tout en permettant des améliorations progressives.
Reliability Rating	A (ERROR)	B (WARNING)	Un rating A garantit que le nouveau code ne contient pas de bugs connus. **Code Global** : Un rating B tolère quelques bugs mineurs dans le code existant, qui peuvent être adressés progressivement
Security Hotspots Reviewed	100%	Non défini	**Nouveau Code** : Ce critère vérifie que tous les hotspots de sécurité identifiés dans le nouveau code ont été examinés par un développeur. Ce critère est fortement recommandé selon la poltique « Clean as your code » de SonarQube.
Security Rating	A (ERROR)	A (ERROR)	**Les deux**: Aucun compromis sur la sécurité, même pour le code existant. Les vulnérabilités de sécurité

Métrique	Nouveau Code	Code Global	Justification
			doivent être traitées immédiatement, quelle que soit leur ancienneté.

4.3.2.2. Explications détaillées sur les métriques pour le backend

Explication des Ratings pour Maintainability, Reliability et Security dans SonarQube.

Dans SonarQube, les « Ratings » (notations) sont des indicateurs de qualité qui évaluent différents aspects du code sur une échelle de A (excellent) à E (très problématique). Les trois ratings mentionnés – Maintainability, Reliability et Security - sont des métriques fondamentales permettant d'évaluer objectivement la qualité globale d'un projet.

Maintainability Rating (Notation de Maintenabilité)

La notation de maintenabilité évalue la facilité avec laquelle le code peut être modifié, compris et maintenu par les développeurs.

Comment est-elle calculée ?

- Elle est directement liée au ratio de la « dette technique » par rapport au temps de développement

- La dette technique représente le temps estimé nécessaire pour corriger tous les « Medium et Low severity Issues » (mauvaises pratiques de code)

- Un rating A signifie que le ratio de dette technique est inférieur à 5% du temps de développement

Échelle de notation :

- **A** : Ratio de dette technique < 5%

- **B** : Ratio entre 6% et 10%

- **C** : Ratio entre 11% et 20%

- **D** : Ratio entre 21% et 50%

- **E** : Ratio > 50%

Exemples de problèmes affectant cette notation :

- Code dupliqué

- Méthodes trop complexes ou trop longues

- Trop de paramètres dans les fonctions

- Classes trop grandes

- Commentaires insuffisants ou obsolètes

Reliability Rating (Notation de Fiabilité)

La notation de fiabilité évalue la probabilité que le code contienne des bugs ou des défauts qui pourraient provoquer des comportements incorrects en production.

Comment est-elle calculée ?

- Elle est basée sur la sévérité du bug le plus grave détecté dans le code

- Un seul bug bloquant suffit pour obtenir la note E

Échelle de notation :

- **A** : Aucun bug
- **B** : Au moins un bug mineur
- **C** : Au moins un bug majeur
- **D** : Au moins un bug critique
- **E** : Au moins un bug bloquant

Exemples de problèmes affectant cette notation :

- Déréférencement de pointeur null
- Fuites de ressources
- Logique incorrecte dans les conditions
- Problèmes de concurrence
- Exceptions non gérées

Security Rating (Notation de Sécurité)

La notation de sécurité évalue la présence de vulnérabilités de sécurité dans le code qui pourraient être exploitées par des attaquants.

Comment est-elle calculée ?

- Elle est basée sur la sévérité de la vulnérabilité la plus grave détectée
- Une seule vulnérabilité critique suffit pour obtenir la note E

Échelle de notation :

- **A** : Aucune vulnérabilité
- **B** : Au moins une vulnérabilité mineure
- **C** : Au moins une vulnérabilité majeure
- **D** : Au moins une vulnérabilité critique
- **E** : Au moins une vulnérabilité bloquante

Exemples de problèmes affectant cette notation :

- Injections SQL
- Cross-Site Scripting (XSS)
- Failles CSRF (Cross-Site Request Forgery)
- Mauvaise gestion des secrets et mots de passe
- Utilisation d'algorithmes cryptographiques faibles

4.3.2.3. Pourquoi exiger des Ratings A dans les Quality Gates ?

Dans le contexte des Quality Gates de SonarQube, exiger des ratings A pour ces trois métriques sur le nouveau code signifie que :

1. **Standard élevé pour le nouveau code** : Tout nouveau code ajouté doit être de haute qualité, même si le code existant peut contenir des problèmes.

2. **Prévention de la dégradation** : Cette approche empêche l'accumulation progressive de problèmes (principe du « ne pas casser les vitres").

3. **Amélioration progressive** : Au fil du temps, à mesure que le code ancien est remplacé, la qualité globale du projet s'améliore.

4. **Objectivité** : Ces ratings fournissent des critères objectifs et mesurables pour évaluer la qualité du code.

4.3.2.4. Application pratique dans notre workflow CI/CD

Dans les workflows GitHub Actions de votre projet BobApp, l'intégration de ces ratings comme conditions de Quality Gate permet de :

1. **Fournir un feedback immédiat** : Les développeurs reçoivent rapidement des informations sur les problèmes à corriger.

2. **Maintenir des standards cohérents** : Les mêmes critères de qualité sont appliqués à tous les développeurs et toutes les parties du code.

3. **Faciliter la revue de code** : Les revues peuvent se concentrer sur des aspects plus subtils, les problèmes évidents étant déjà détectés automatiquement.

En exigeant des ratings A pour la maintenabilité, la fiabilité et la sécurité, on établit un standard élevé qui garantit que votre code reste propre, fiable et sécurisé au fil du temps, tout en réduisant le coût de maintenance à long terme.

4.3.3. Quality Gates définies pour le projet Frontend (Angular)

Voici notre proposition avec approche différenciée entre le Nouveau Code et le Code Global.

4.3.3.1. Justification des métriques et seuils spécifiques pour le Frontend

Métrique	Nouveau Code	Code Global	Justification
Coverage	80% (ERROR)	70% (WARNING)	**Nouveau Code**: Le seuil de 80% est identique à celui du backend pour maintenir une cohérence dans les standards de qualité entre les deux parties de l'application. Ce niveau garantit une bonne couverture des composants et services Angular **Code Global**: Le seuil plus bas de 70% pour le code global reconnaît la difficulté de tester certains aspects du code frontend, notamment les templates Angular et les interactions UI complexes.
Issues	0	Non défini	**Nouveau Code**: politique de "zéro défaut" pour le nouveau code dans les dernières versions d'Angular. Encourage les développeurs à résoudre tous les problèmes avant de fusionner leur code. Évite l'accumulation progressive de dette technique. SonarQue promet de pouvoir tenir ce seuil à zéro grâce à ses conseils actualisés. A vérifier dans la

Métrique	Nouveau Code	Code Global	Justification
			pratique car plus difficile dans le front d'avoir ce zéro défaut. Possibilité de mettre un comment NOSONAR pour non prise en compte si besoin. On conserve tout de même les autres severity issues. **Code Global**: pour le code global (le « code d'avant » le contrôle qualité), on est moins sévère et plus fin dans les catégories de Issues. Cf. les 3 prochaines sections.
Blocker severity Issues	0 (ERROR)	3 (WARNING)	**Nouveau Code**: Tolérance zéro pour les problèmes bloquants dans le nouveau code, conformément à la philosophie « Clean as You Code ». **Code Global**: La limite de 3 (légèrement moins stricte que pour le backend) reconnaît que les frameworks frontend peuvent générer des faux positifs pour des problèmes bloquants. Exemples de Faux Positifs pour des Blocking severity Issues dans Angular en annexe page 167.
High severity Issues	2 (ERROR)	15 (WARNING)	**Nouveau Code**: Une légère tolérance (2) pour les problèmes critiques dans le nouveau code frontend, car certains patterns Angular légitimes peuvent être signalés comme critiques par SonarQube. **Code Global**: La limite plus élevée pour le code existant permet une correction progressive des problèmes critiques hérités. Exemples en annexe page 165.
Medium et Low severity Issues	10 (WARNING)	100 (WARNING)	**Nouveau Code**: Le seuil de 10 (plus élevé que pour le backend) reconnaît que le code TypeScript/Angular génère naturellement plus de Medium et Low severity Issues en raison des patterns spécifiques au framework. **Code Global**: La limite de 100 pour le code global reconnaît la réalité des applications Angular qui peuvent accumuler des Medium et Low severity Issues, tout en encourageant leur réduction progressive.
Duplicated Lines	3% (WARNING)	8% (WARNING)	**Nouveau Code**: Un seuil plus strict que pour le backend (3% vs 5%) encourage la création de composants réutilisables, un principe fondamental d'Angular. **Code Global**: Le seuil de 8% pour le code existant permet une correction progressive de la duplication.
Maintainability Rating	A (ERROR)	B (WARNING)	**Nouveau Code**: Exiger un rating A garantit que le nouveau code est hautement maintenable, avec une structure de composants claire et des services bien définis. **Code Global**: Un rating B est acceptable pour le code existant, reconnaissant la complexité inhérente aux applications Angular matures.
Reliability Rating	A (ERROR)	B (WARNING)	**Nouveau Code** : Un rating A garantit que le nouveau code ne contient pas de bugs connus.

Métrique	Nouveau Code	Code Global	Justification
			Code Global : Un rating B tolère quelques bugs mineurs dans le code existant, qui peuvent être adressés progressivement
Security Rating	A (ERROR)	A (ERROR)	**Les deux**: Aucun compromis sur la sécurité, même pour le code existant. Les vulnérabilités de sécurité dans le frontend (XSS, CSRF, etc.) doivent être traitées immédiatement, quelle que soit leur ancienneté.

Spécificités des Quality Gates pour le Frontend Angular

Le développement frontend avec Angular présente des défis spécifiques qui justifient certains ajustements dans les Quality Gates :

1. Complexité des templates : Les templates Angular peuvent être difficiles à tester de manière exhaustive, ce qui explique la tolérance légèrement plus élevée pour la couverture de code global.

2. Duplication fonctionnelle vs structurelle : Angular encourage la réutilisation des composants, mais certaines duplications peuvent être nécessaires pour des raisons de performance ou d'isolation. Le seuil de duplication est donc adapté en conséquence.

3. Faux positifs spécifiques au framework : SonarQube peut signaler certains patterns Angular légitimes comme des problèmes, d'où la tolérance légèrement plus élevée pour les problèmes critiques.

4. Évolution rapide des bonnes pratiques : Les bonnes pratiques Angular évoluent rapidement avec chaque nouvelle version du framework, ce qui peut générer des Medium et Low severity Issues temporaires pendant les migrations.

Différences clés avec les Quality Gates du Backend

Aspect	Frontend	Backend	Justification
High severity Issues	New : 2 Overall 15	New : 0 Overall : 10	Le frontend a une tolérance légèrement plus élevée en raison des faux positifs spécifiques à Angular.
Medium et Low severity Issues	10 : New Overall : 100	New : 5 Overall : Non défini	Le code TypeScript/Angular génère naturellement plus de Medium et Low severity Issues que Java.
Duplicated Lines	New : 3% Overall : 8%	New : 5% Overall : 10%	Seuil plus strict pour encourager la réutilisation des composants Angular.

Métriques spécifiques au Frontend qui pourraient être ajoutées

Pour une analyse encore plus précise du code frontend, ces métriques supplémentaires pourraient être envisagées à l'avenir :

1. Complexité cognitive des composants : Limiter la complexité des composants Angular pour améliorer la maintenabilité.

2. Nombre de dépendances par composant/service : Encourager l'injection de dépendances minimale pour réduire le couplage.

3. Taille des bundles JavaScript : Surveiller la taille des bundles générés pour optimiser les performances.

Stratégie d'évolution des Quality Gates Frontend

Comme pour le backend, cette configuration n'est pas statique et devrait évoluer avec la maturité du projet :

1. Phase initiale (actuelle) : Différenciation marquée entre nouveau code et code global.

2. Phase intermédiaire : À mesure que la qualité globale s'améliore, les seuils pour le code global peuvent être progressivement resserrés.

3. Phase mature : À terme, les seuils pour le nouveau code et le code global pourraient converger, signalant une base de code uniformément de haute qualité.

Conclusion

Cette approche différenciée des Quality Gates pour le frontend Angular offre un équilibre entre rigueur et pragmatisme, adaptée aux spécificités du développement frontend moderne. En maintenant des standards élevés pour le nouveau code tout en reconnaissant les défis du code existant, cette configuration permet :

1. D'améliorer progressivement la qualité du code frontend
2. De maintenir un rythme de développement soutenu
3. D'adapter les critères de qualité aux spécificités d'Angular
4. De maintenir une cohérence avec l'approche adoptée pour le backend

Cette stratégie s'inscrit parfaitement dans la philosophie DevOps du projet BobApp, favorisant l'amélioration continue tout en reconnaissant les réalités pratiques du développement logiciel.

4.3.3.2. Explications détaillées sur les métriques pour le frontend

Dans le cadre de notre stratégie d'assurance qualité pour le projet BobApp, nous avons défini plusieurs quality gates spécifiques pour le frontend Angular. Voici l'explication détaillée des trois quality gates mentionnées :

New Blocker severity Issues is greater than 0 ERROR

Cette quality gate vérifie qu'aucun nouveau problème bloquant n'est introduit dans le code frontend.

Fonctionnement :

- Un « Blocker Issue » est un problème critique qui présente un risque élevé pour l'application
- La sévérité « ERROR » signifie que le build échouera si cette condition n'est pas respectée
- Le seuil « 0 » indique une tolérance zéro pour ce type de problème

Justification :

- Les problèmes bloquants dans le frontend peuvent compromettre gravement l'expérience utilisateur
- Exemples typiques : failles de sécurité XSS, fuites de mémoire importantes, ou code qui pourrait causer des crashs du navigateur
- Cette règle stricte garantit qu'aucun problème critique ne sera déployé en production

New High severity Issues is greater than 2 ERROR

Cette quality gate limite le nombre de nouveaux problèmes critiques à un maximum de 2 dans le code frontend.

Fonctionnement :

- Un « Critical Issue » est un problème sérieux mais légèrement moins grave qu'un problème bloquant
- La sévérité « ERROR » signifie que le build échouera si plus de 2 problèmes critiques sont introduits
- Le seuil « 2 » offre une légère tolérance, reconnaissant la complexité du développement frontend

Justification :

Une tolérance limitée (2) a été choisie pour le frontend car :

- Le développement Angular peut parfois nécessiter des compromis techniques qui génèrent des alertes
- Certaines pratiques courantes en développement frontend peuvent être signalées comme critiques par SonarQube
- Cette flexibilité permet de ne pas bloquer le développement pour des problèmes qui peuvent être adressés ultérieurement

New Medium et Low severity Issues is greater than 10 WARNING

Cette quality gate surveille l'introduction de « Medium et Low severity Issues » (mauvaises pratiques de code) avec un seuil de tolérance plus élevé.

Fonctionnement :

- Un « Code Smell » est un problème de maintenabilité qui n'affecte pas directement le fonctionnement
- La sévérité « WARNING » signifie que le build ne sera pas bloqué, mais une alerte sera générée
- Le seuil « 10 » permet une certaine flexibilité tout en maintenant la qualité globale

Justification :

Le seuil plus élevé (10) pour le frontend est justifié par :

- La nature du code TypeScript/Angular qui peut générer plus de Medium et Low severity Issues en raison des patterns spécifiques au framework
- Les templates Angular et les fichiers de style qui peuvent déclencher des alertes de code smell
- L'évolution rapide des bonnes pratiques frontend qui peut rendre certaines règles moins pertinentes

4.3.3.3. Pourquoi « Medium et Low severity Issues » n'a pas été choisi pour le Backend

La quality gate « New Medium et Low severity Issues » n'a pas été explicitement configurée pour le backend pour plusieurs raisons :

1. **Différence de nature du code** :
   - Le code Java/Spring Boot du backend est généralement plus structuré et suit des patterns bien établis
   - Les règles de qualité pour Java sont plus matures et standardisées que pour TypeScript/Angular

2. **Couverture implicite par d'autres métriques** :
   - Le backend utilise la métrique « Maintainability Rating on New Code is worse than A » qui englobe déjà indirectement les Medium et Low severity Issues

- Cette métrique est calculée principalement à partir du ratio de dette technique, qui est lui-même basé sur les Medium et Low severity Issues

3. **Approche plus stricte pour le backend** :

- Le backend gère la logique métier critique et l'accès aux données
- Une approche plus stricte est appliquée au backend car les problèmes à ce niveau peuvent avoir des conséquences plus graves
- L'exigence d'un rating A pour la maintenabilité impose déjà une limite stricte sur les Medium et Low severity Issues

4. **Différence de cycle de développement** :

- Le code backend évolue généralement moins rapidement que le frontend
- Les standards de qualité pour le code Java sont plus stables dans le temps
- Cette stabilité permet d'appliquer des règles plus strictes sans risquer de bloquer fréquemment le développement

En résumé, pour le frontend, nous avons opté pour une approche plus granulaire avec des seuils spécifiques pour différents types de problèmes, reconnaissant la nature plus dynamique et évolutive du développement frontend. Pour le backend, nous avons privilégié une approche basée sur les ratings globaux (A pour Maintainability et Security) qui englobent implicitement les différents types de problèmes, y compris les Medium et Low severity Issues.

4.4. Quality Profiles dans SonarQube

Un Quality Profile est un ensemble de règles d'analyse de code qui sont appliquées lors de l'analyse SonarQube. Contrairement aux Quality Gates qui définissent des seuils sur les métriques, les Quality Profiles définissent quelles règles sont activées et avec quelle sévérité.

4.4.1. Avantages de la séparation des projets pour les Quality Profiles

La séparation du projet BobApp en deux projets SonarQube distincts (backend et frontend) offre plusieurs avantages pour la gestion des Quality Profiles :

1. **Règles spécifiques au langage** :

- Le backend (Java) et le frontend (TypeScript/JavaScript) nécessitent des ensembles de règles complètement différents
- Chaque projet peut utiliser le Quality Profile le plus adapté à sa technologie

2. **Personnalisation ciblée** :

- Les règles peuvent être ajustées indépendamment pour chaque partie de l'application
- Possibilité d'activer des règles spécifiques à Spring Boot pour le backend
- Possibilité d'activer des règles spécifiques à Angular pour le frontend

3. **Évolution indépendante** :

- Les standards de qualité peuvent évoluer à des rythmes différents pour le backend et le frontend
- Facilite l'adoption progressive de nouvelles règles sans impacter l'autre partie

4.4.2. Quality Profiles par défaut

Pour l'instant, le projet BobApp utilise les Quality Profiles par défaut de SonarQube :

- **Backend** : « Sonar way » pour Java
- **Frontend** : « Sonar way » pour TypeScript et JavaScript

Ces profils par défaut sont déterminés automatiquement par SonarQube en fonction des langages détectés dans le code source. SonarQube maintient ces profils et les met à jour régulièrement avec les meilleures pratiques et les nouvelles règles.

4.4.3. Évolution future des Quality Profiles

À mesure que le projet BobApp évoluera, il pourrait être judicieux de créer des Quality Profiles personnalisés basés sur les profils « Sonar way » par défaut, avec des ajustements spécifiques :

1. **Pour le backend** :
   - Ajout de règles spécifiques à Spring Boot
   - Personnalisation des règles pour les tests JUnit
   - Ajustement des règles de documentation (Javadoc)

2. **Pour le frontend** :
   - Ajout de règles spécifiques à Angular
   - Personnalisation des règles pour les tests Jasmine/Karma
   - Ajustement des règles de gestion d'état et de cycle de vie des composants

4.4.4. Conclusion

L'intégration des Quality Gates SonarQube dans le processus CI/CD de BobApp complète parfaitement la stratégie de séparation des workflows GitHub Actions. En définissant des critères de qualité spécifiques pour le backend et le frontend, tout en maintenant un seuil minimum commun de 70% de couverture de code, cette approche garantit :

1. **Cohérence de la qualité** à travers les différentes parties de l'application
2. **Détection précoce des problèmes** avant qu'ils n'atteignent la production
3. **Amélioration continue** grâce à des métriques objectives et quantifiables
4. **Flexibilité** pour adapter les critères de qualité aux spécificités de chaque technologie

Cette stratégie s'inscrit dans la continuité de l'approche modulaire adoptée pour les workflows CI/CD et les projets SonarQube, tout en maintenant une gouvernance unifiée de la qualité du code.

4.5. Différences entre « New Code » et « Overall Code » dans SonarQube

Cette section explique la distinction fondamentale que fait SonarQube entre le « New Code » (nouveau code) et l'"Overall Code » (code global), et comment cette distinction influence l'application des Quality Gates et la gestion de la dette technique dans le projet BobApp.

4.5.1. Définition des concepts

4.5.1.1. New Code

Le « New Code » dans SonarQube désigne le code qui a été ajouté ou modifié depuis une certaine référence temporelle. Cette référence peut être :

- **Depuis la dernière version** : Code modifié depuis la dernière release
- **Depuis une période spécifique** : Par exemple, les 30 derniers jours
- **Depuis une analyse précise** : Une date ou une analyse spécifique
- **Depuis une branche de référence** : Généralement la branche principale (main)

Dans le contexte des Pull Requests, le « New Code » correspond aux modifications introduites par la PR elle-même.

4.5.1.2. Overall Code

L'Overall Code représente l'intégralité du code source du projet, incluant à la fois le code historique et le nouveau code. Il s'agit de l'état complet du projet à un moment donné.

4.5.1.3. Différences fondamentales d'approche

Aspect	New Code	Overall Code
Portée	Limité aux changements récents	Ensemble du projet
Focus	Prévention de nouvelle dette	État global de la qualité
Objectif	Maintenir une qualité élevée pour les nouveaux développements	Évaluer la santé globale du projet
Métrique	Plus stricte, souvent 0 tolérance pour les problèmes critiques	Peut tolérer une certaine quantité de problèmes
Visualisation dans SonarQube	Onglet « New Code » dans le tableau de bord	Onglet « Overall » dans le tableau de bord

4.5.2. Pourquoi cette distinction est importante

4.5.2.1. Approche « Clean as You Code"

SonarQube promeut une philosophie appelée « Clean as You Code » (Nettoyez en codant) qui <u>se concentre sur le maintien de la qualité du nouveau code</u> plutôt que sur la correction immédiate de toute la dette technique existante. Cette approche présente plusieurs avantages :

- **Réalisme** : Il est souvent irréaliste de corriger immédiatement tous les problèmes dans une base de code existante
- **Progressivité** : La qualité globale s'améliore naturellement au fil du temps
- **Efficacité** : Les développeurs se concentrent sur la qualité de leur travail actuel
- **Motivation** : Évite le découragement face à une montagne de dette technique

4.5.2.2. Gestion efficace de la dette technique

La distinction New Code/Overall Code permet de :

- **Éviter l'accumulation** de nouvelle dette technique
- **Prioriser les corrections** en se concentrant d'abord sur les problèmes critiques dans le nouveau code
- **Planifier la réduction** progressive de la dette existante
- **Mesurer l'amélioration** de la qualité au fil du temps

4.5.2.3. Application différenciée des Quality Gates

Dans notre configuration pour BobApp, nous avons défini des Quality Gates qui s'appliquent principalement au nouveau code :

- Maintainability Rating on New Code is worse than A
- Security Rating on New Code is worse than A

Cette approche garantit que :

- Le nouveau code respecte des standards élevés
- La qualité globale s'améliore progressivement
- Les développeurs reçoivent un feedback immédiat sur leur travail

4.5.3. Application dans les workflows CI/CD de BobApp

4.5.3.1. Configuration actuelle

Dans nos workflows GitHub Actions, l'analyse SonarQube est configurée pour examiner à la fois le nouveau code et le code global :

1. **Pour le backend** :
```
run: mvn -B sonar:sonar -DskipTests=true -Dsonar.projectKey=cpierres_P10cicd-backend
```

2. **Pour le frontend** :
```
uses: SonarSource/sonarqube-scan-action@v5.3.0
```

4.5.3.2. Visualisation des résultats

Dans l'interface SonarQube, les résultats sont accessibles via différentes URL :

- **New Code** :
`https://sonarqube.cpierres.dscloud.me/dashboard?id=cpierres_P10cicd-backend&codeScope=new`
- **Overall Code** :
`https://sonarqube.cpierres.dscloud.me/dashboard?id=cpierres_P10cicd-backend&codeScope=overall`

Notez que dans le lien SonarQube du workflow backend, nous utilisons déjà le paramètre `codeScope=overall` :
```
echo « Rapport SonarQube Backend: https://sonarqube.cpierres.dscloud.me/dashboard?id=cpierres_P10cicd-backend&codeScope=overall"
```

4.5.4. Stratégies recommandées

4.5.4.1. Quality Gates différenciées

Pour une gestion optimale de la qualité, nous recommandons :

- **Quality Gates strictes pour le New Code** : Maintenir les critères actuels avec des seuils stricts
- **Quality Gates plus souples pour l'Overall Code** : Définir des seuils réalistes qui peuvent être atteints progressivement

4.5.4.2. Visualisation et suivi

- **Surveiller principalement le New Code** pour les développements quotidiens
- **Examiner périodiquement l'Overall Code** pour suivre l'amélioration globale
- **Définir des objectifs d'amélioration** pour l'Overall Code sur des périodes définies

4.5.4.3. Intégration dans le processus de développement

- **Pull Requests** : Appliquer strictement les Quality Gates sur le New Code
- **Revues de code** : Utiliser les rapports SonarQube comme support
- **Sprints dédiés** : Planifier périodiquement des sprints de « nettoyage » pour réduire la dette technique dans l'Overall Code

4.5.4.4. Exemple concret pour BobApp

Situation actuelle hypothétique

Imaginons que l'analyse SonarQube de BobApp montre :

Métrique	New Code	Overall Code
Coverage	85%	62%
Bugs	0	24
Vulnerabilities	0	7
Medium et Low severity Issues	3	156
Duplications	1.2%	8.5%

Interprétation

Cette situation illustre parfaitement l'approche « Clean as You Code » :

- Le nouveau code est de haute qualité (85% de couverture, aucun bug ni vulnérabilité)
- Le code global contient une dette technique historique
- La stratégie fonctionne car la qualité s'améliore progressivement

Actions recommandées

1. **Continuer à maintenir** la qualité du nouveau code
2. **Planifier la réduction** de la dette technique existante
3. **Prioriser la correction** des vulnérabilités et bugs critiques dans le code global
4. **Suivre l'évolution** des métriques globales au fil du temps

Conclusion

La distinction entre « New Code » et « Overall Code » est un concept fondamental dans la gestion de la qualité avec SonarQube. En concentrant les efforts sur le maintien d'une qualité élevée pour le nouveau code tout en planifiant la réduction progressive de la dette technique existante, BobApp peut atteindre et maintenir un haut niveau de qualité de manière réaliste et durable.

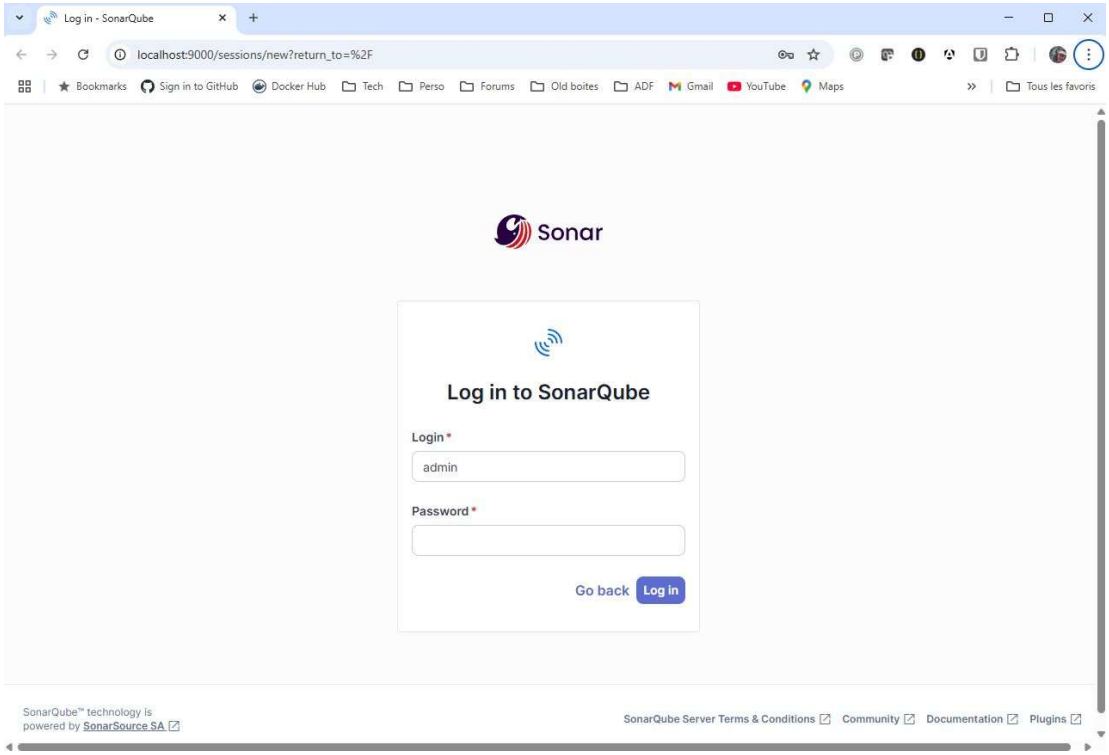
Cette approche s'aligne parfaitement avec la structure modulaire de l'application et la séparation des workflows CI/CD pour le backend et le frontend, permettant une gestion fine et adaptée de la qualité pour chaque composant du système.

5. Installation et configuration de SonarQube pour définir deux projets correspondants à nos deux workflows ci-cd

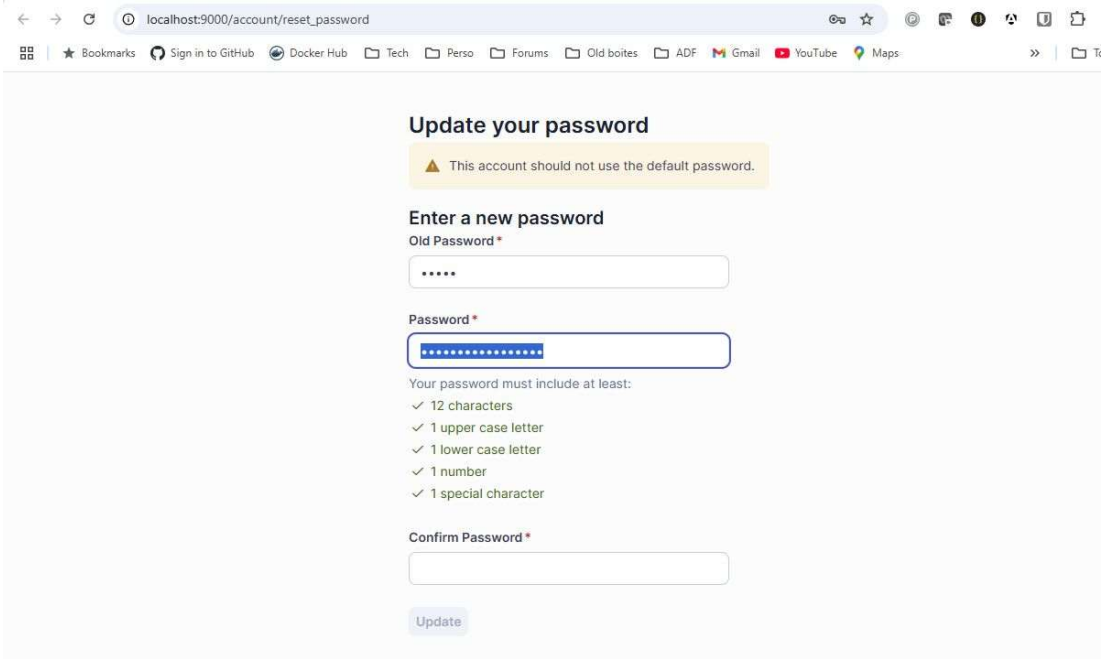
L'installation du stack a été effectuée via le docker-compose-sonar.yml.

Le stack accessible sur le réseau local en `http://localhost :9000` est également exposé sur internet avec l'url : `https://sonarqube.cpierres.dscloud.me` (avec un certificat valide).

Lors de la première connexion, se connecter avec `admin/admin`

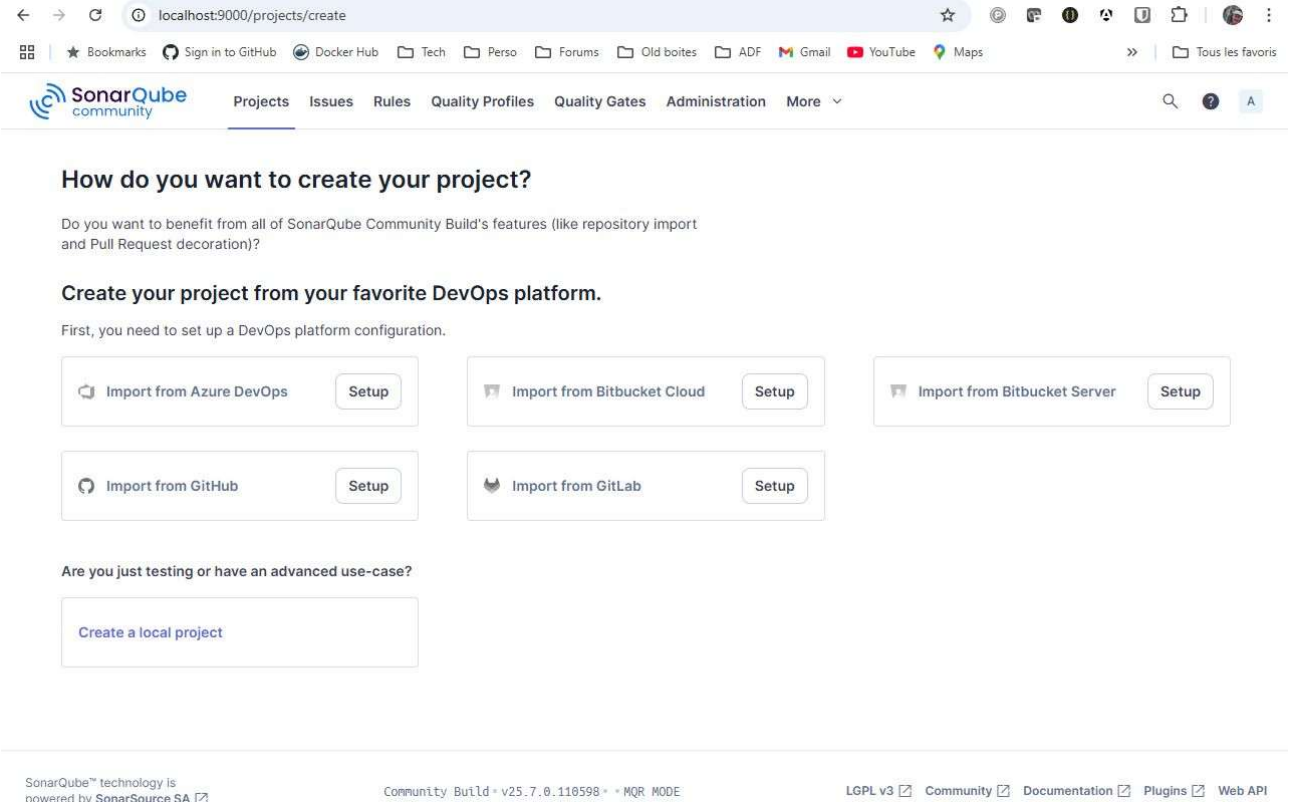


Changer le mot de passe :

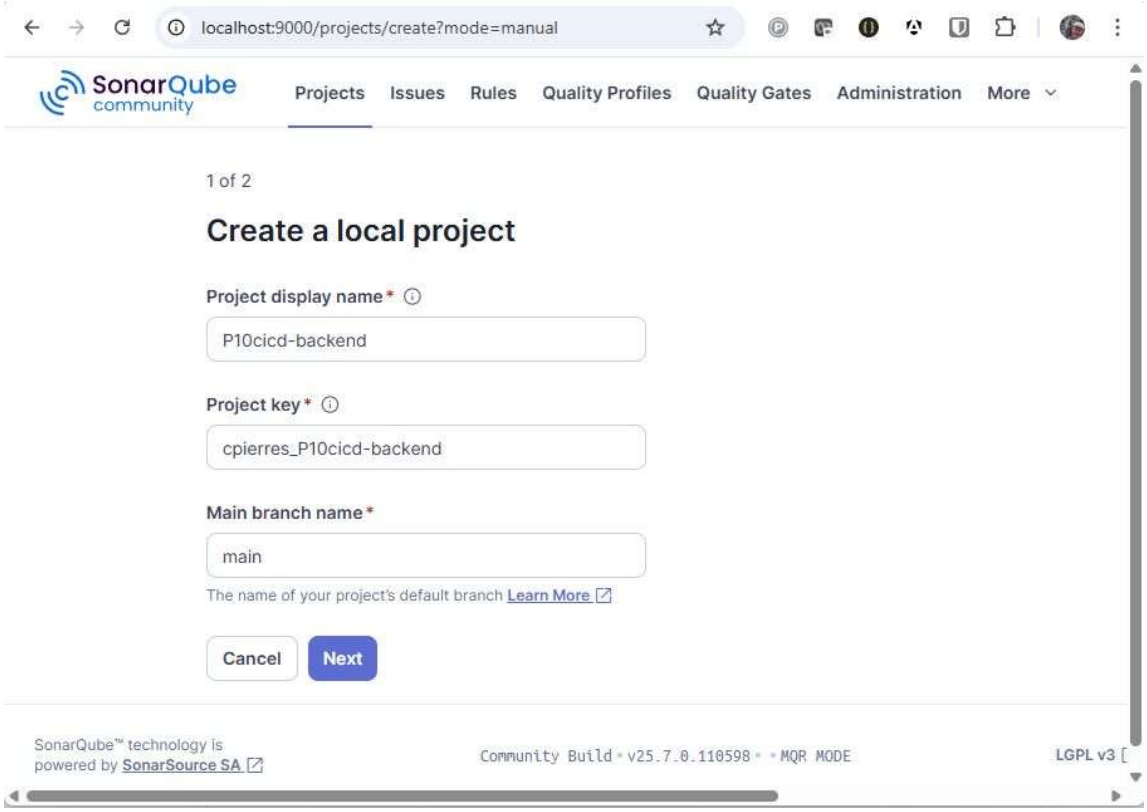


5.1. Créer le projet pour le backend : clé cpierres_P10cicd-backend

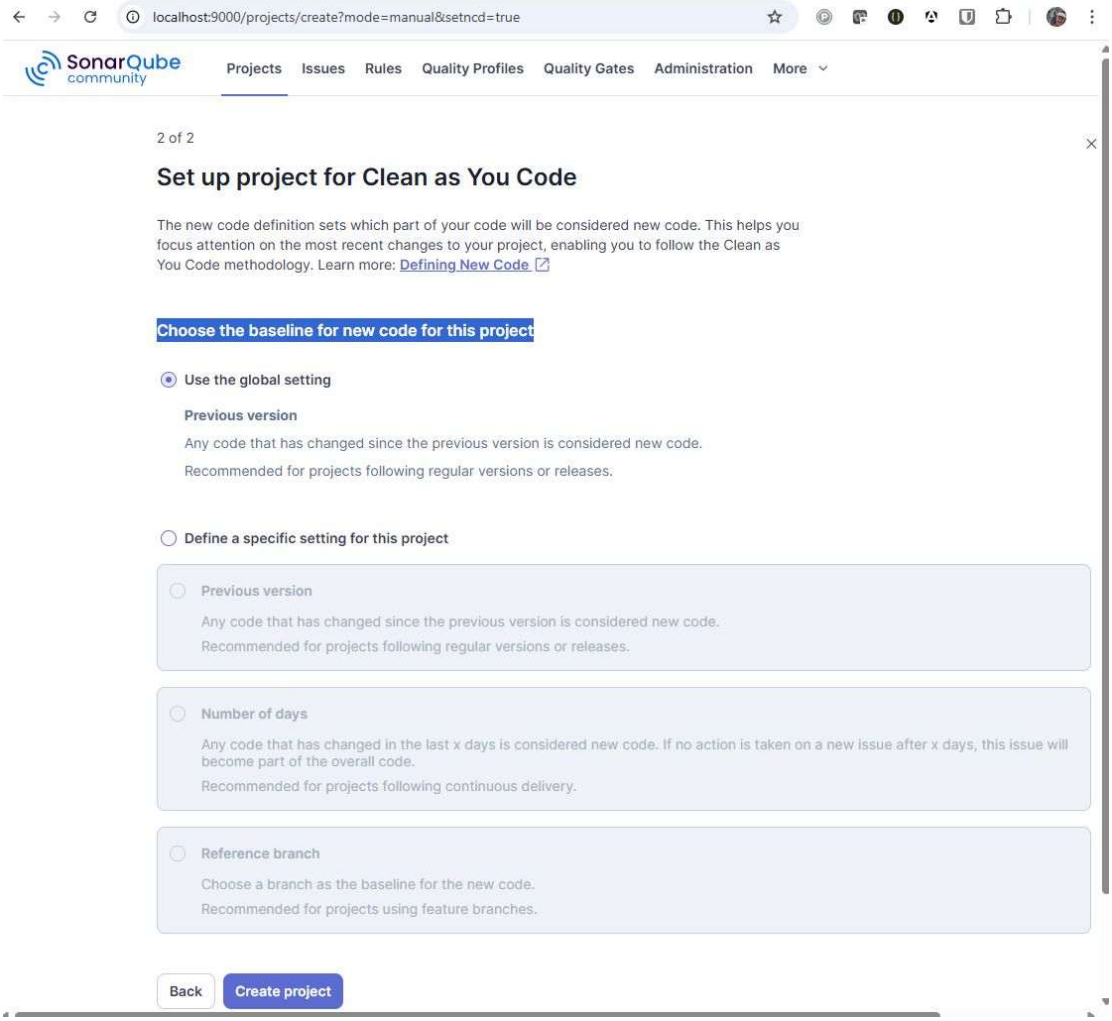
Si vous devez définir un projet standard pour Github (un projet sonar=un repository github), il faudrait choisir **Import from GitHub**. Cependant, comme nous souhaitons installer 2 projets sonar pointant sur chaque couche de notre repository github, cela nécessite de choisir **Create a local projet** :



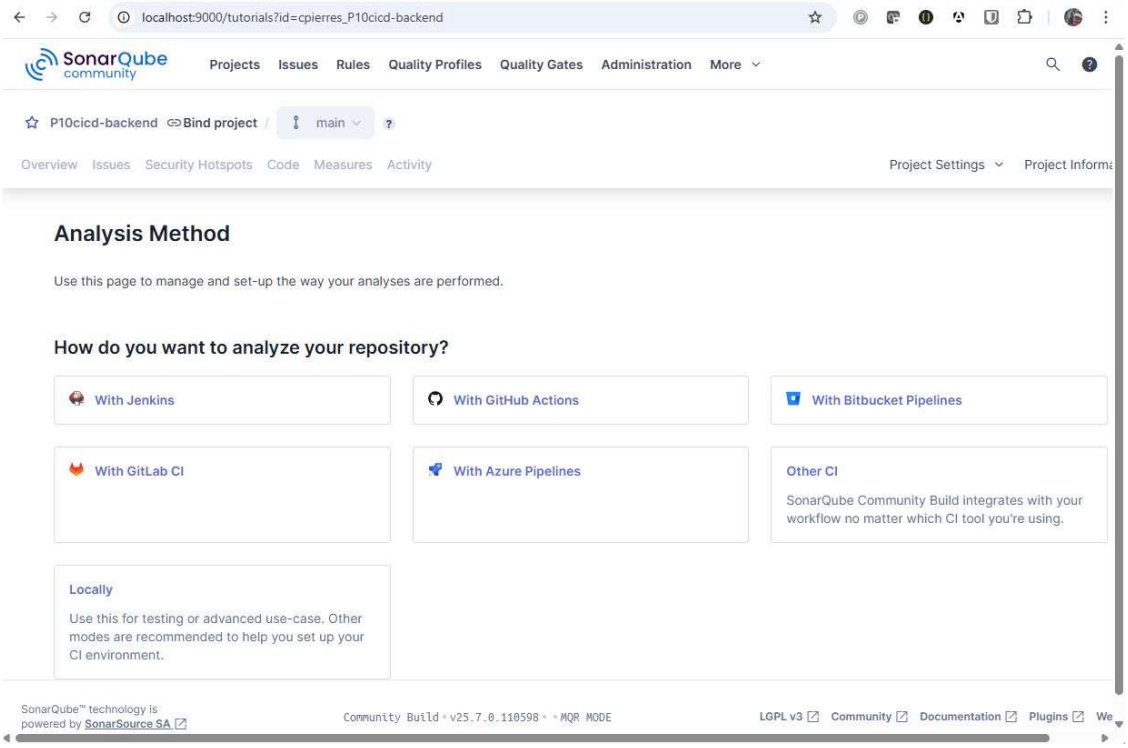
Renseigner la clé de projet `cpierres_P10cicd-backend` définie dans backend-ci.yml :



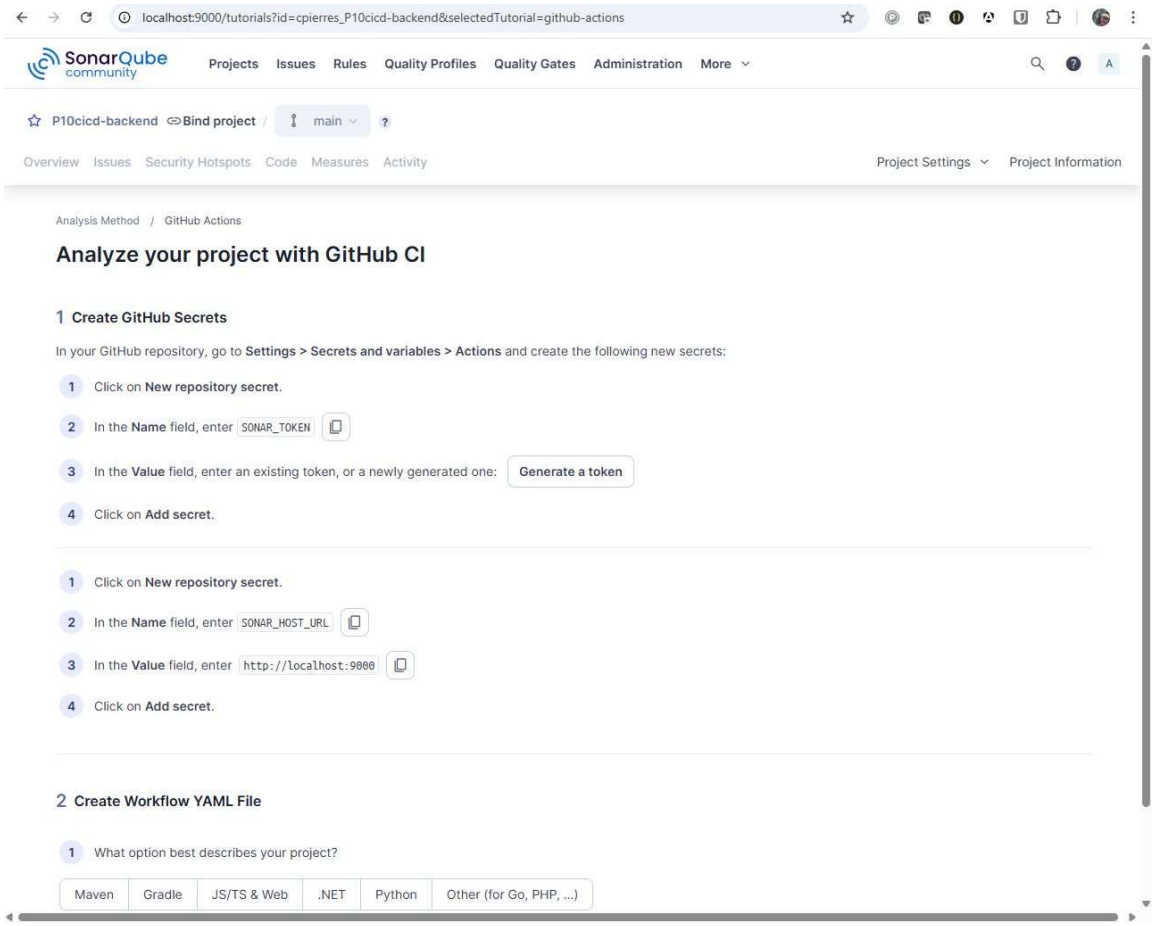
Choisir la ligne de base pour le nouveau code de ce projet :



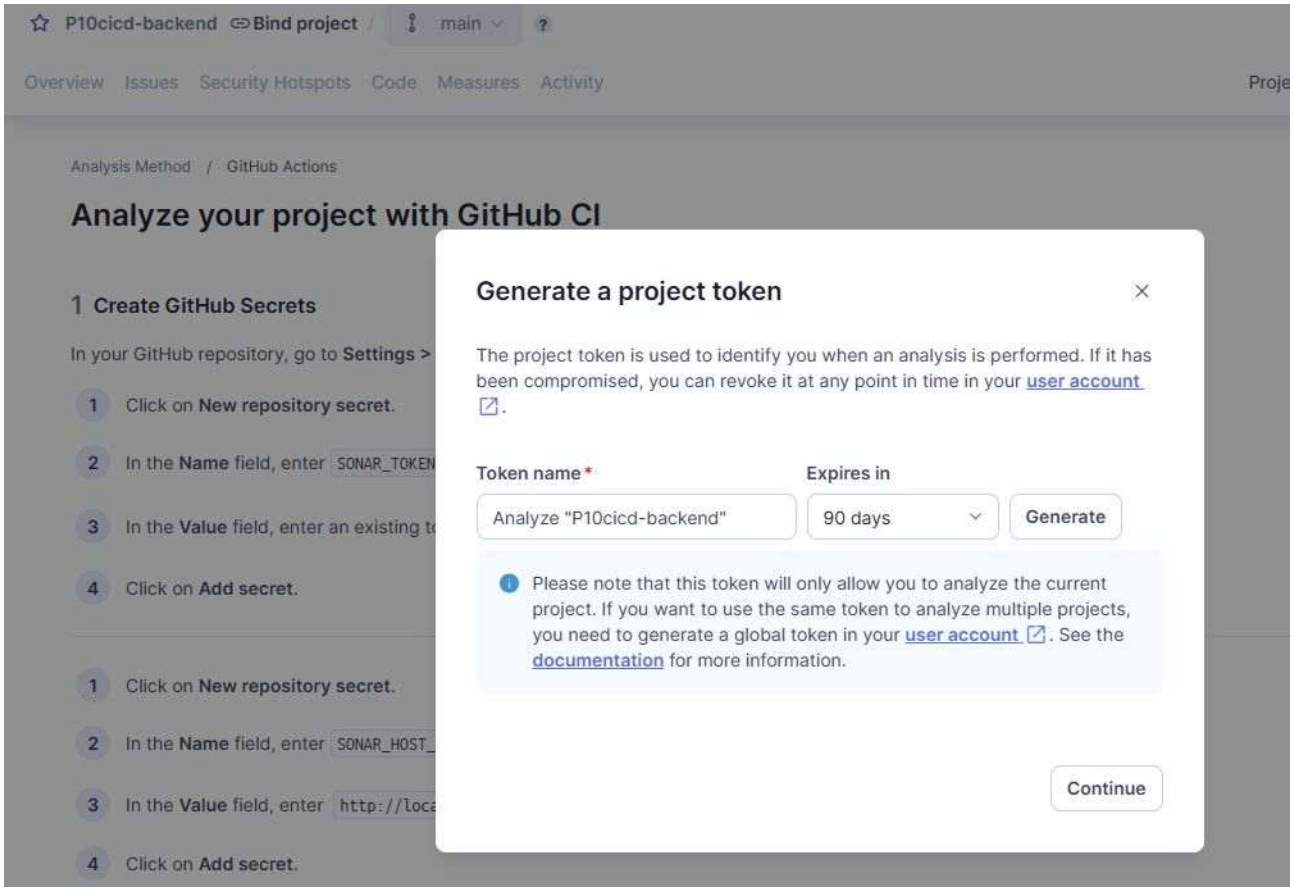
Choisir **GitHub actions** :



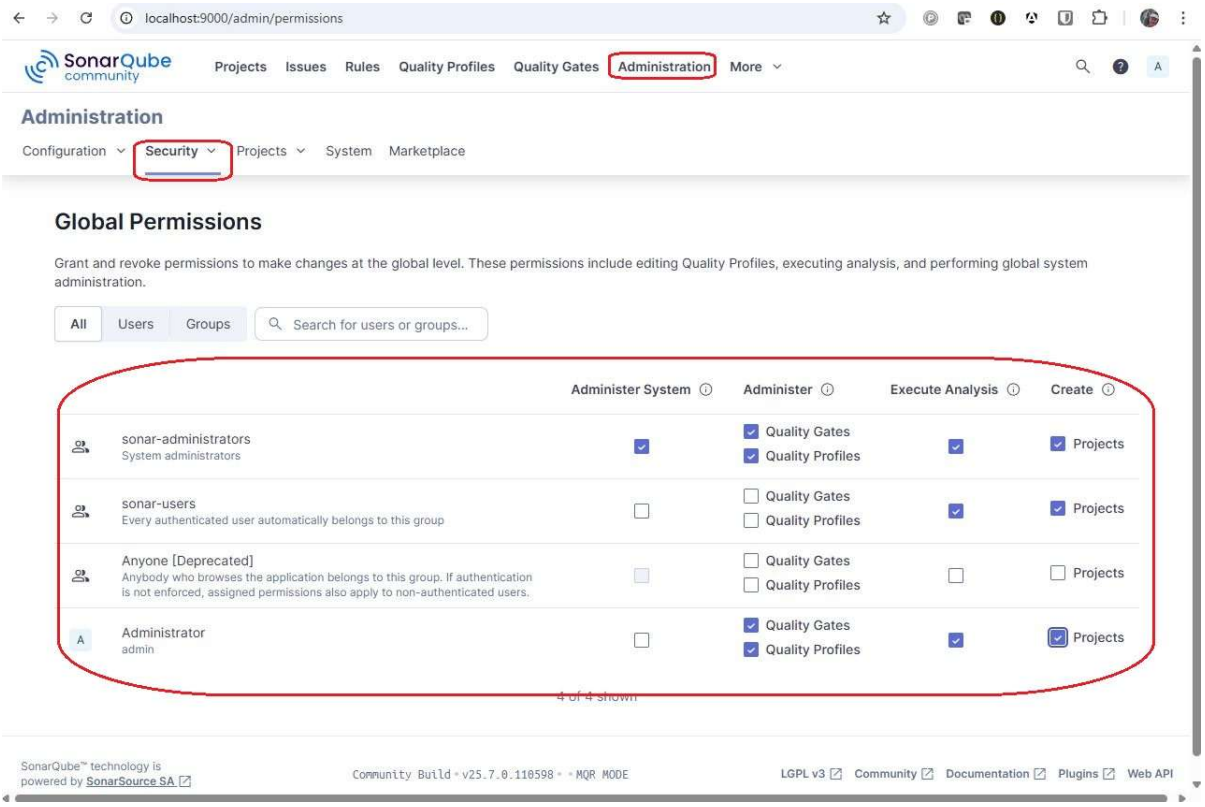
Suivre les instructions (au lieu de `SONAR_TOKEN`, vous indiquerez `SONARQUBE_TOKEN_BACKEND`) :



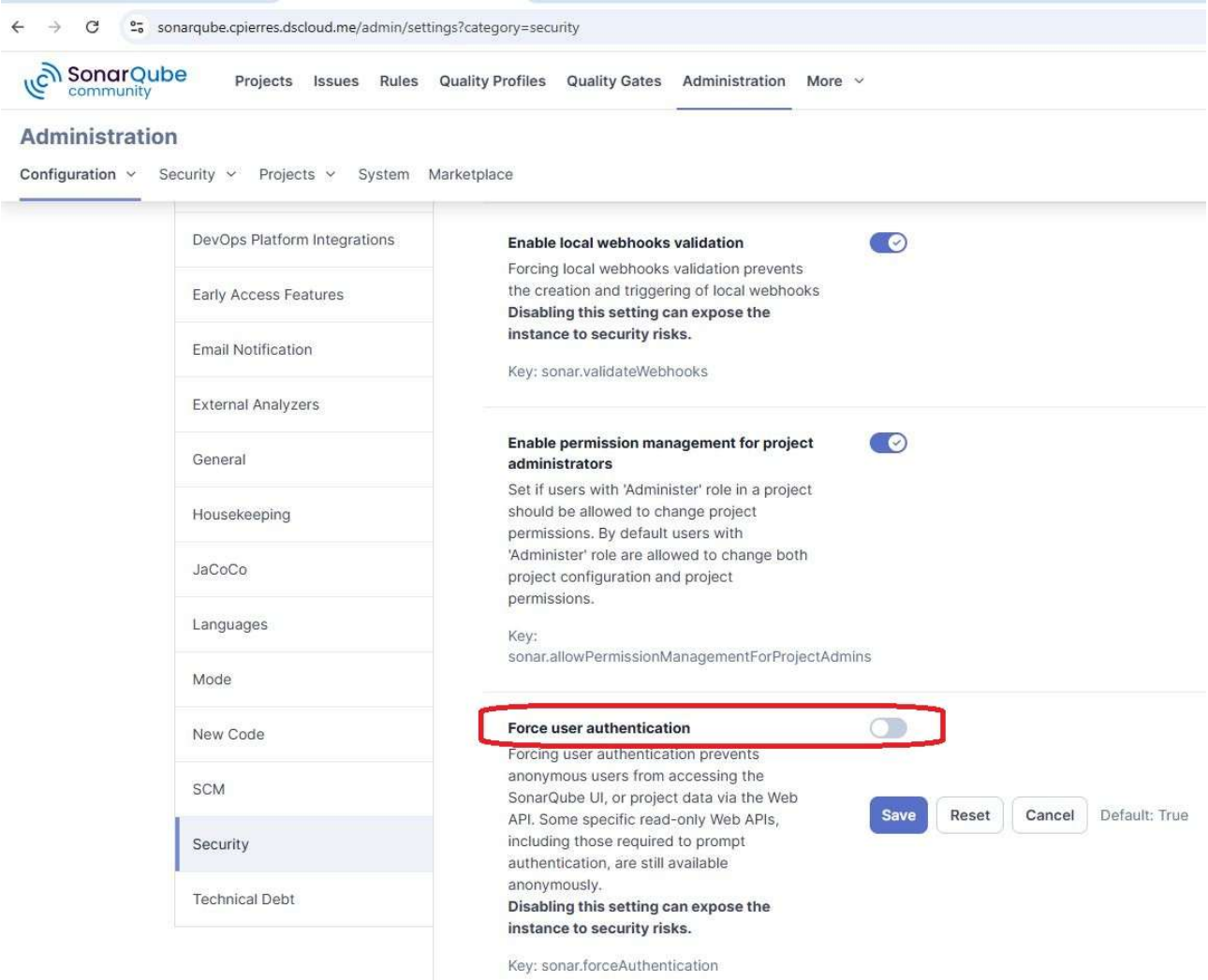
Cliquer sur **Generate a token** (à bien noter dans un registre et le reporter dans github) :



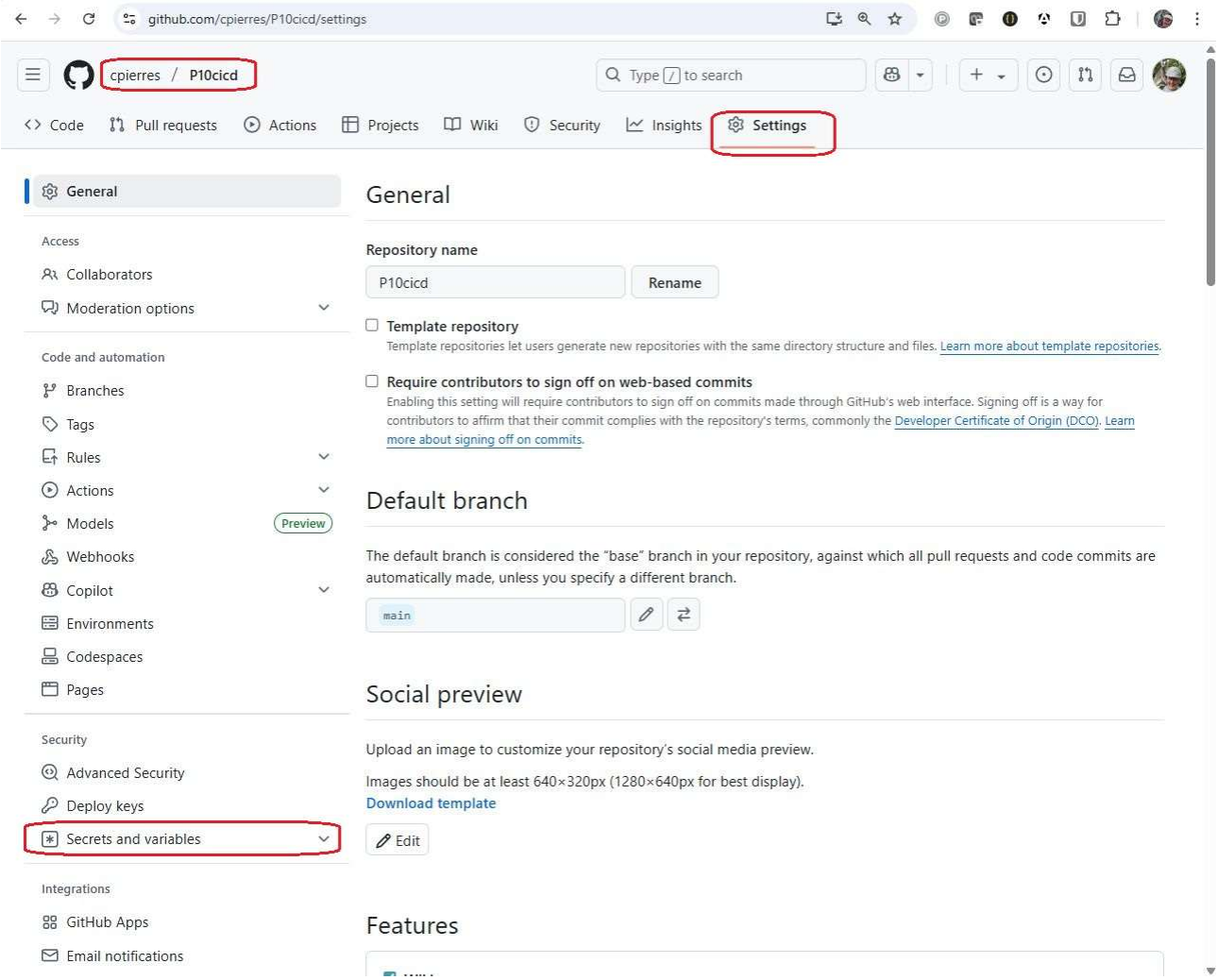
Définir les permissions :



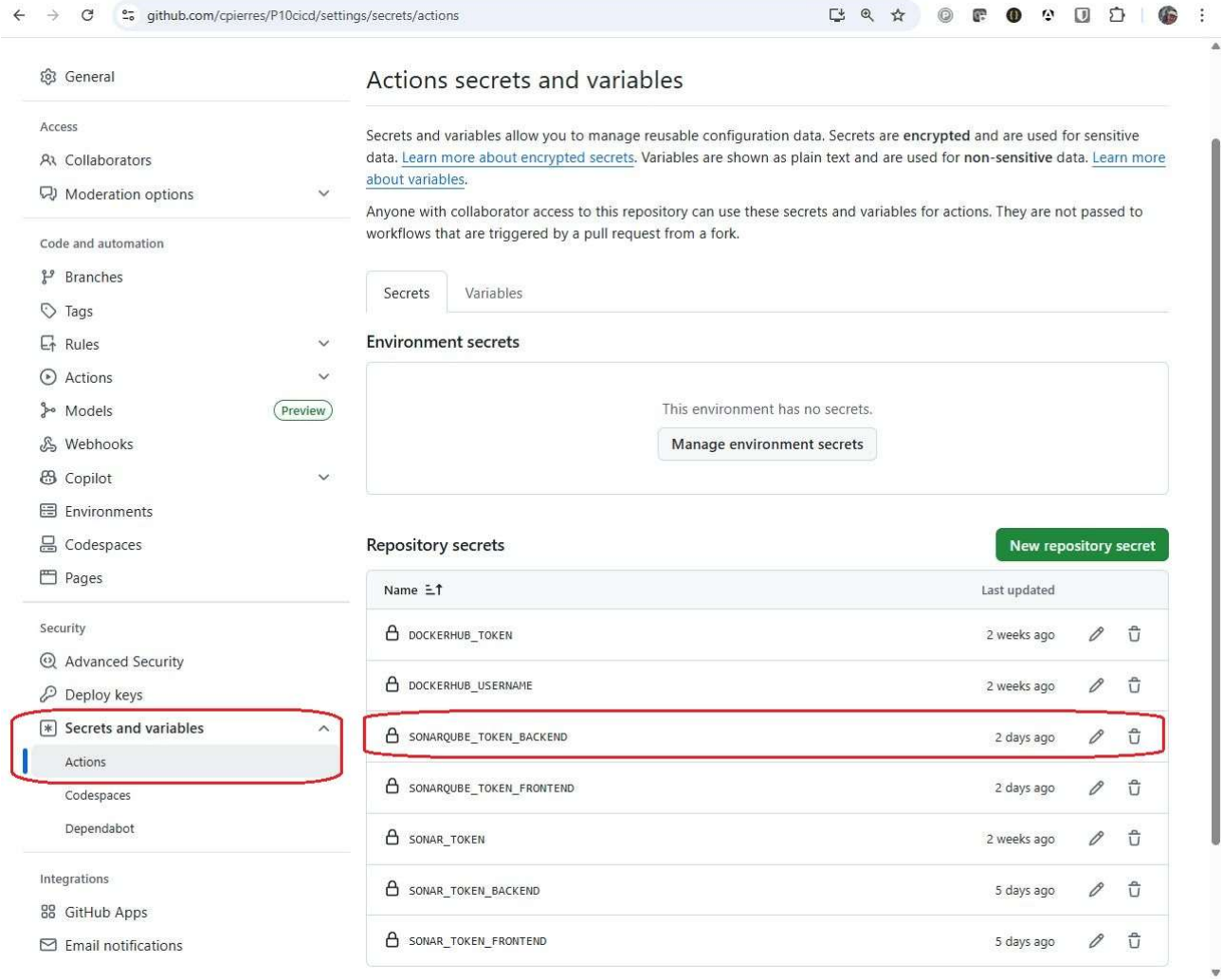
5.1.1. Permettre l'accès non authentifié pour la démonstration



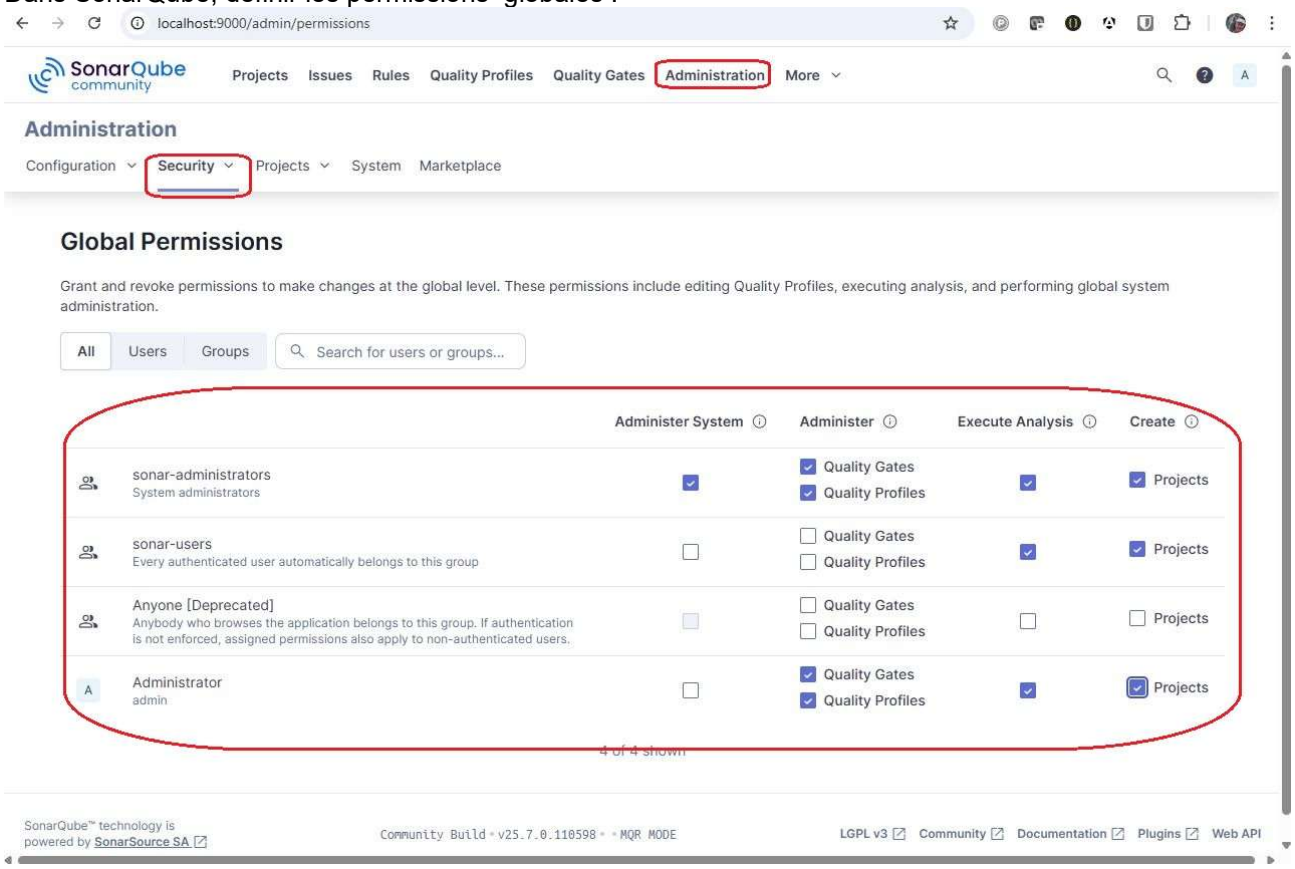
Dans les **Settings** du repository github :



Choisir **Actions** puis coller le token dans la variable adéquate :

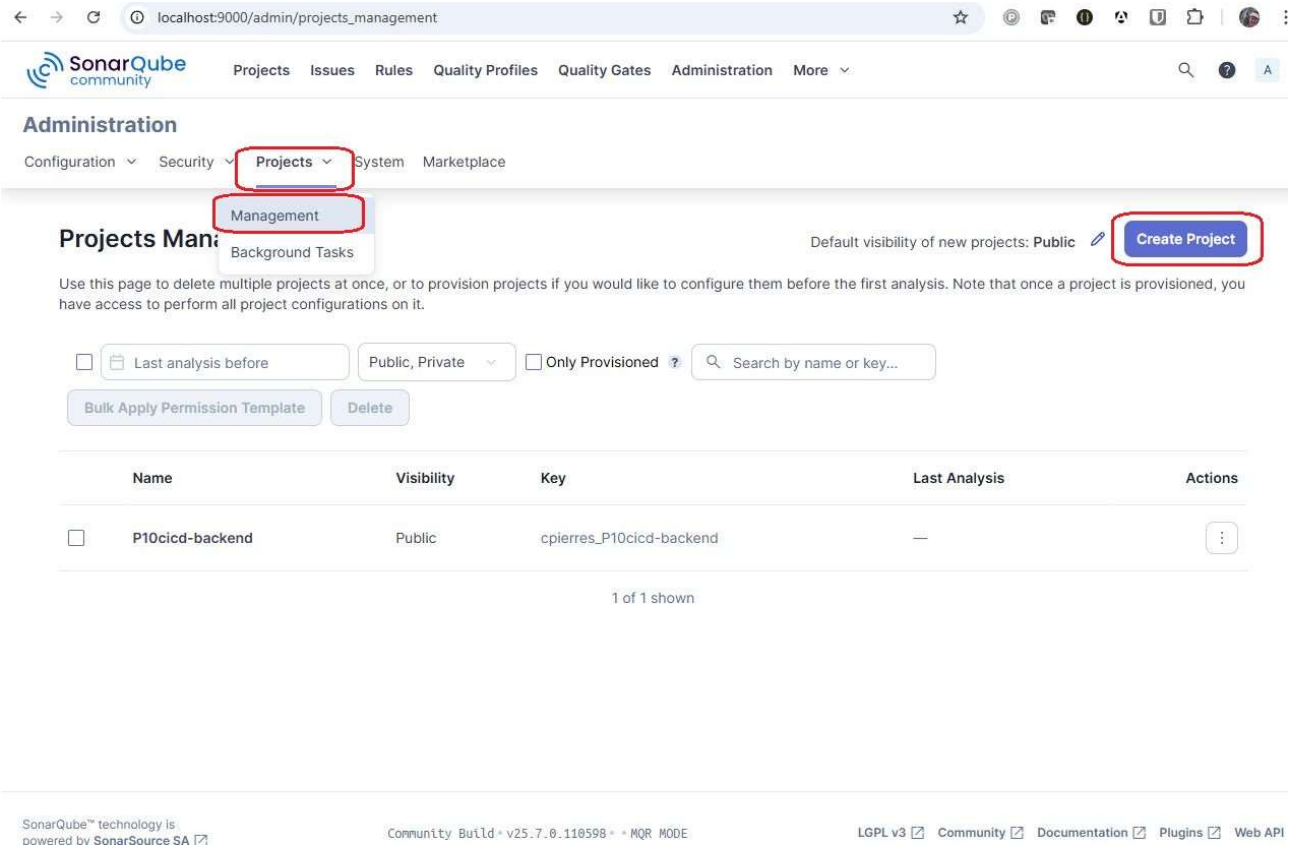


Dans SonarQube, définir les permissions globales :

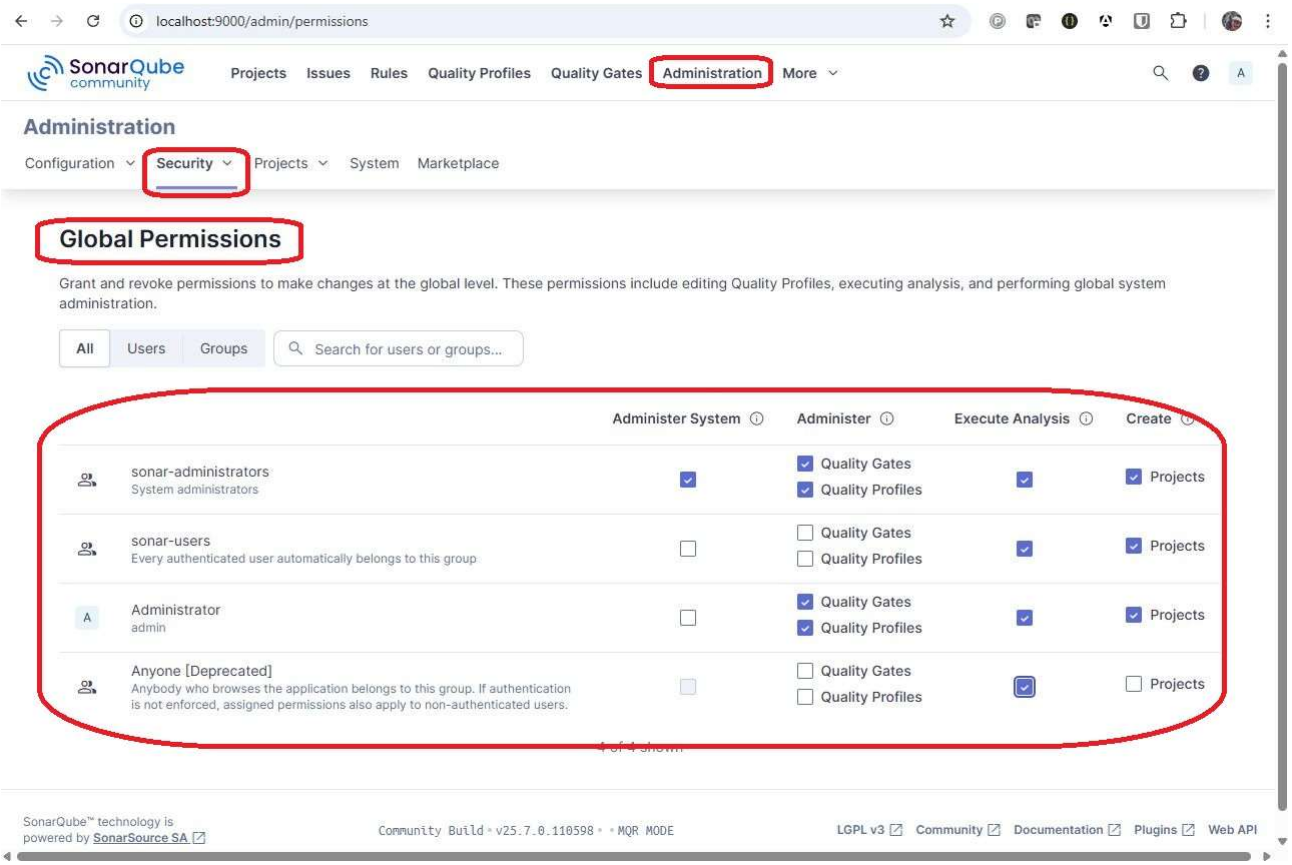


5.2. Créer le projet pour le frontend : clé cpierres_P10cicd-frontend

Même procédure que pour le backend évidemment.



Résultat :



5.3. *Rendre publique ce dépôt*

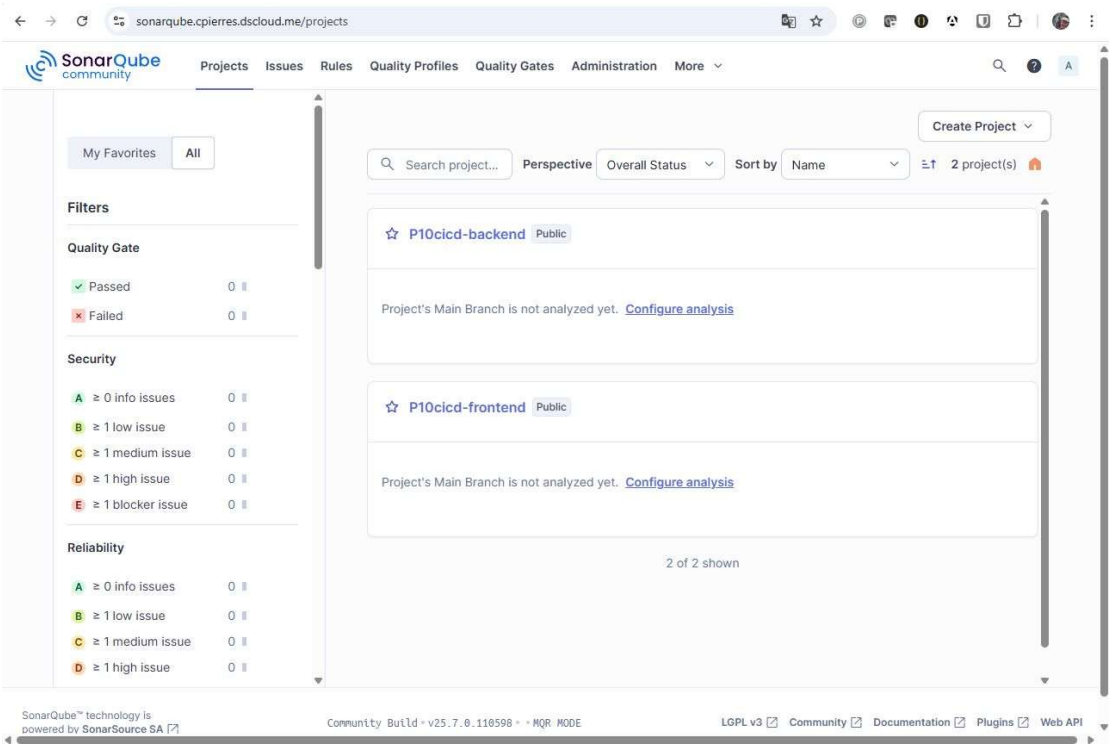
Via un reverse proxy de mon cloud personnel, l'adresse est :

https://sonarqube.cpierres.dscloud.me

5.4. Configurer les Quality gates

Nous voilà avec les 2 projets créés manuellement pour chacune des couches. Aucune analyse n'a encore eu lieu. Inutile de cliquer sur **Configure analysis** du projet backend car ceci a déjà été fait.

Cliquer sur **Quality Gates** :



5.4.1. Mise en place des Quality Gates pour le backend

5.4.1.1. La justification du choix des Avantages spécifiques de cette approche pour le projet BobApp

1. **Adaptation à la maturité du projet** :

- Si BobApp est un projet existant avec du code legacy, cette approche permet de continuer le développement tout en améliorant progressivement la qualité.
- Si BobApp est un nouveau projet, ces standards établissent dès le départ une base solide tout en reconnaissant que des compromis peuvent être nécessaires dans certaines phases.

2. **Équilibre entre rigueur et pragmatisme** :

- Les seuils ERROR pour le nouveau code garantissent que la qualité ne se dégrade pas.
- Les seuils WARNING pour le code global permettent une visibilité sur les problèmes sans bloquer le pipeline.

3. **Priorisation intelligente** :

- La sévérité ERROR sur la sécurité pour le code global force l'équipe à traiter immédiatement les vulnérabilités.
- La tolérance plus élevée pour les problèmes de maintenabilité permet de prioriser les corrections.

4. **Alignement avec les workflows CI/CD** :

- Cette configuration s'intègre parfaitement avec la séparation des workflows backend/frontend.
- Les jobs peuvent continuer même si les analyses SonarQube du code global génèrent des warnings.

5.4.1.2. Stratégie d'évolution des seuils

Cette configuration n'est pas statique et devrait évoluer avec la maturité du projet :

1. **Phase initiale** (actuelle) : Différenciation marquée entre nouveau code et code global.

2. **Phase intermédiaire** : À mesure que la qualité globale s'améliore, les seuils pour le code global peuvent être progressivement resserrés.

3. **Phase mature** : À terme, les seuils pour le nouveau code et le code global pourraient converger, signalant une base de code uniformément de haute qualité.

5.4.1.3. Conclusion

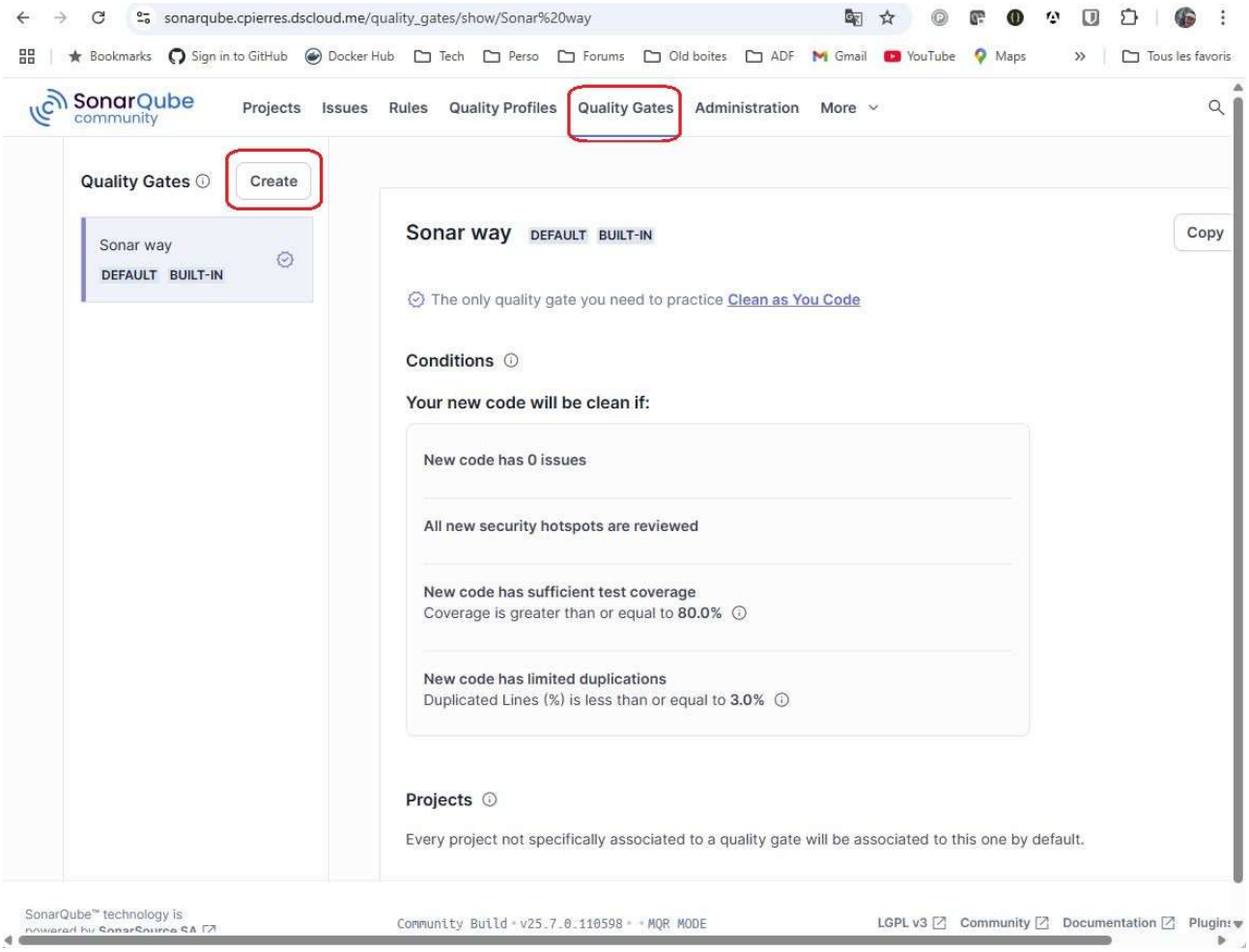
L'approche différenciée des Quality Gates pour le Nouveau Code et le Code Global représente une stratégie équilibrée qui :

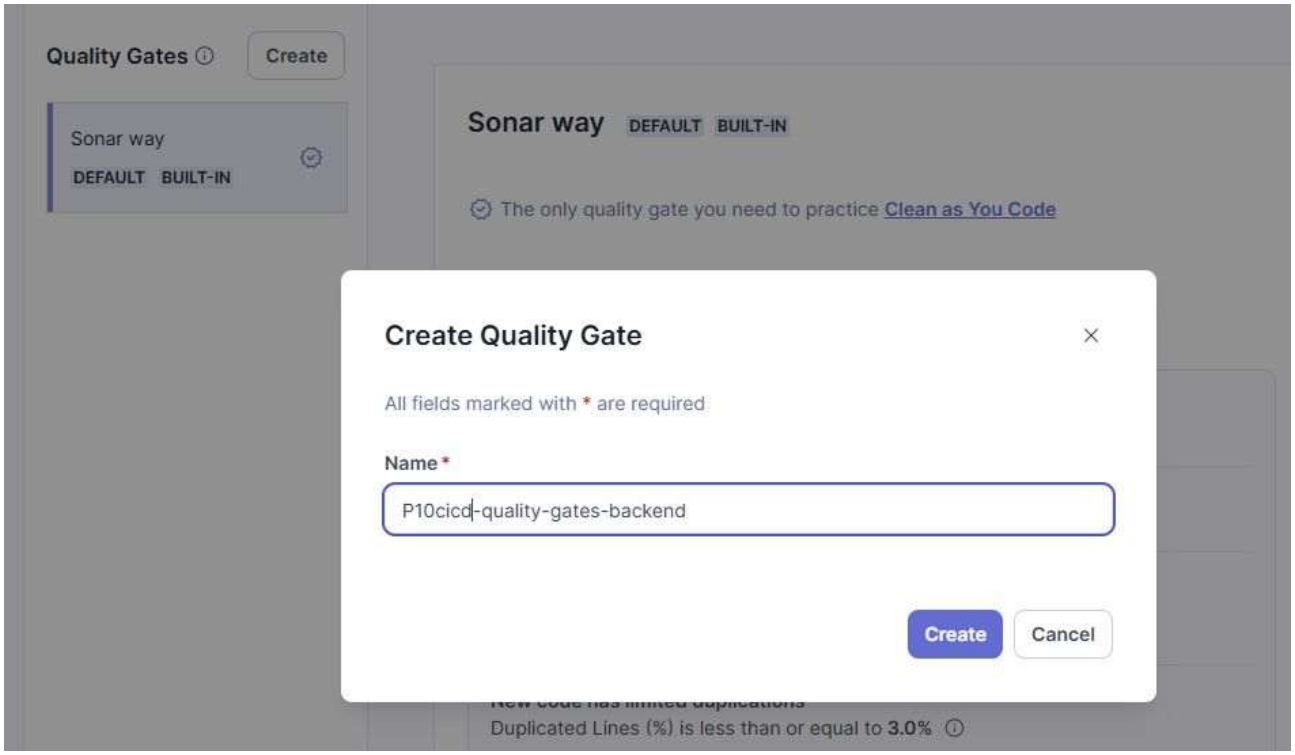
- Garantit que la qualité du code s'améliore progressivement
- Reconnaît les réalités pratiques du développement logiciel
- Établit des priorités claires pour la correction des problèmes
- S'aligne avec la philosophie DevOps d'amélioration continue

Cette configuration permet à l'équipe BobApp de maintenir un rythme de développement soutenu tout en améliorant constamment la qualité du code, créant ainsi un cercle vertueux où chaque nouvelle contribution renforce la robustesse et la maintenabilité de l'application.

Quality Gates définies pour le projet Backend (Java/Spring Boot) se trouve en page 53.

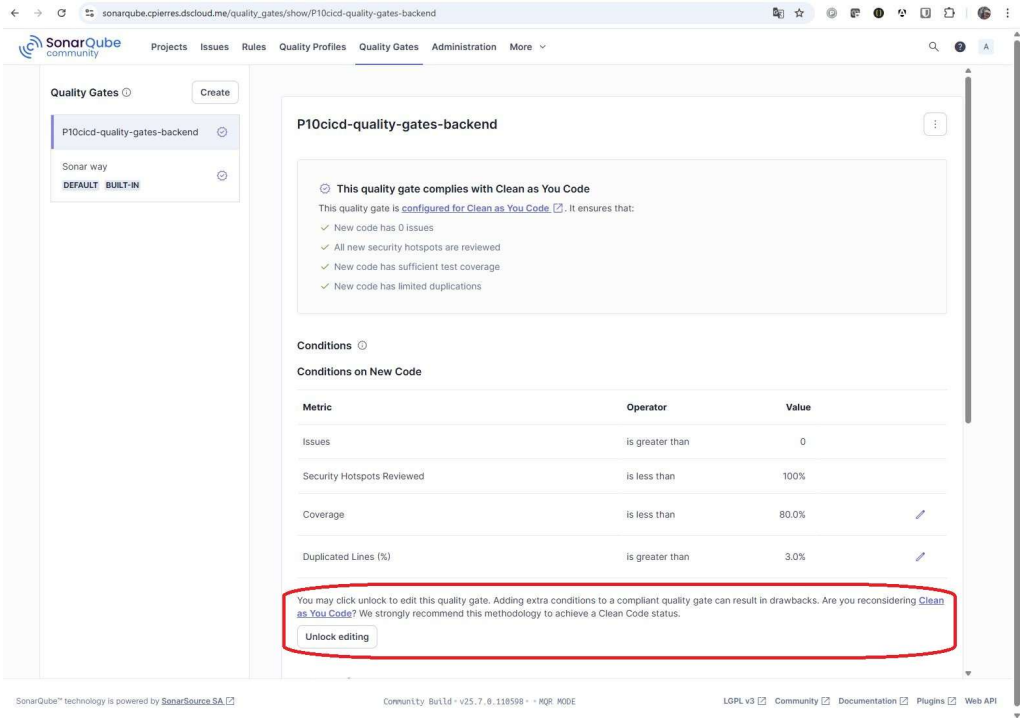
5.4.1.4. Procédure pour créer des Quality Gates

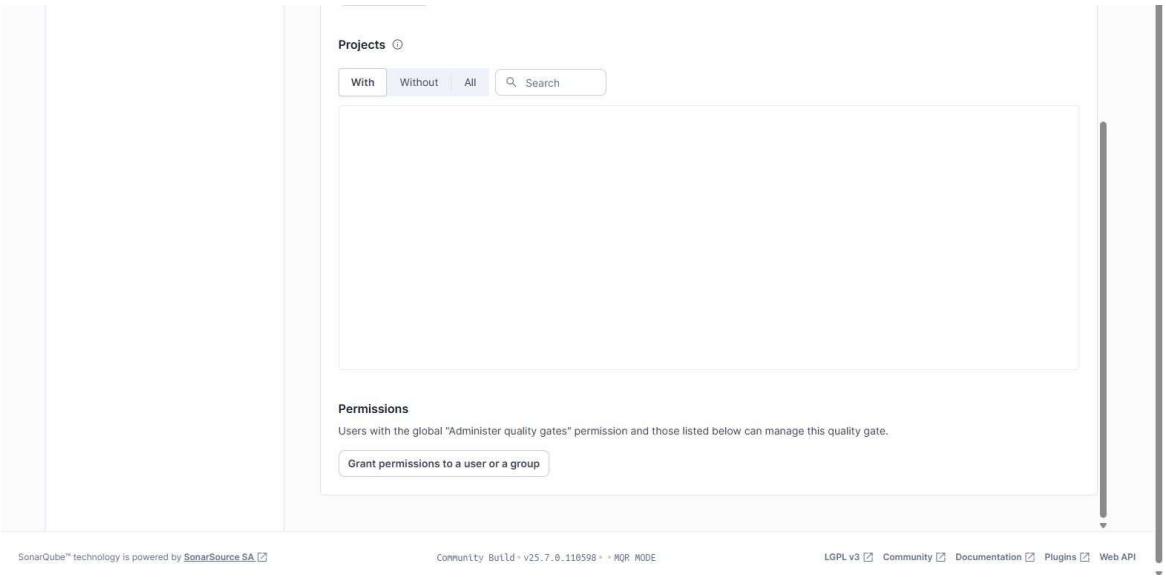




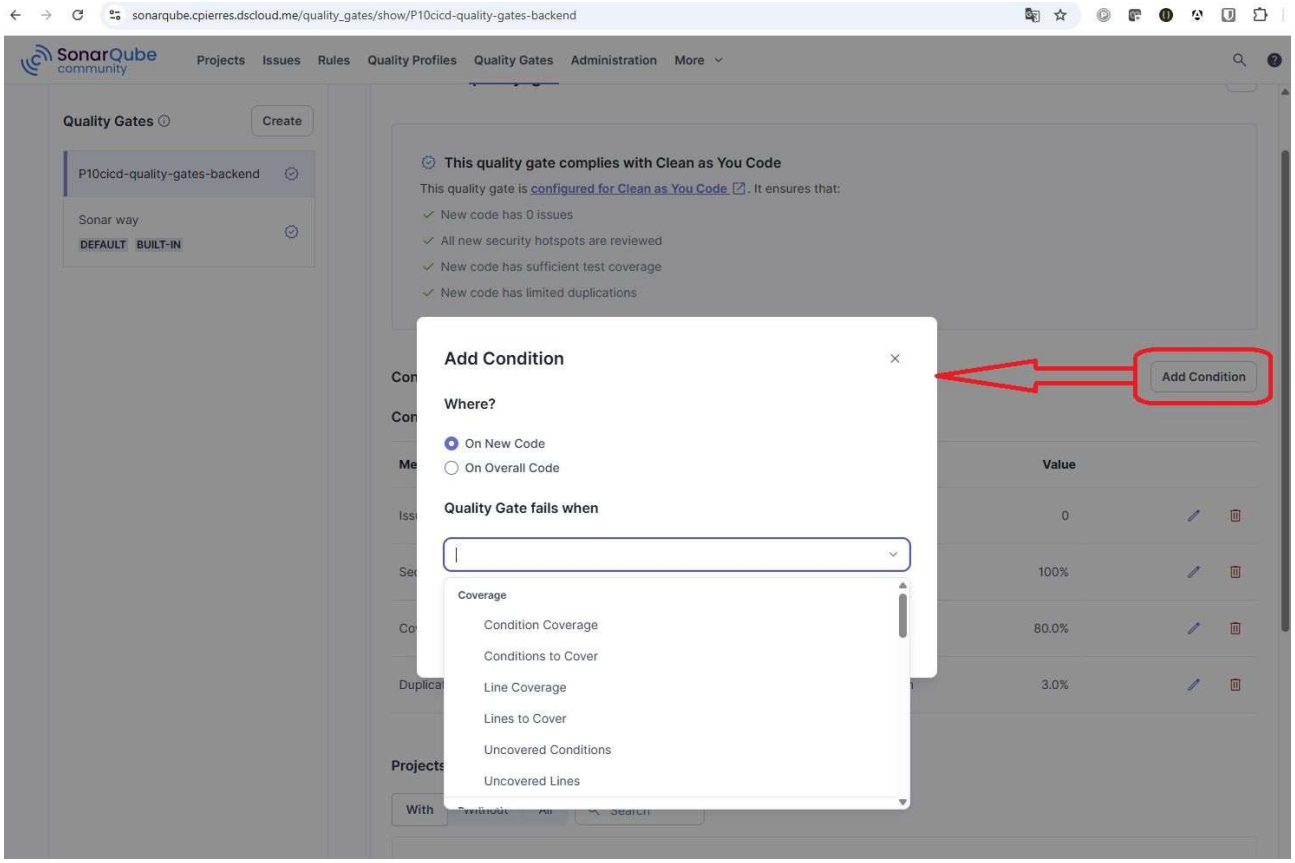
Ce Quality Gates à définir vient avec quelques critères préféfinis pour le nouveau code. Notre but est de personnaliser ce nouveau Quality Gates.

Cliquer sur **Unlock editing** :





Le bouton **Unlock editing** a affiché le bouton **Add condition** :



5.4.1.5. Quality Gates on Overall Code pour le backend

5.4.1.6. Pour rappel, la justification du choix des Avantages spécifiques de cette approche pour le projet BobApp

1. **Adaptation à la maturité du projet** :

- Si BobApp est un projet existant avec du code legacy, cette approche permet de continuer le développement tout en améliorant progressivement la qualité.
- Si BobApp est un nouveau projet, ces standards établissent dès le départ une base solide tout en reconnaissant que des compromis peuvent être nécessaires dans certaines phases.

2. **Équilibre entre rigueur et pragmatisme** :

- Les seuils ERROR pour le nouveau code garantissent que la qualité ne se dégrade pas.
- Les seuils WARNING pour le code global permettent une visibilité sur les problèmes sans bloquer le pipeline.

3. **Priorisation intelligente** :

- La sévérité ERROR sur la sécurité pour le code global force l'équipe à traiter immédiatement les vulnérabilités.
- La tolérance plus élevée pour les problèmes de maintenabilité permet de prioriser les corrections.

4. **Alignement avec les workflows CI/CD** :

- Cette configuration s'intègre parfaitement avec la séparation des workflows backend/frontend.
- Les jobs peuvent continuer même si les analyses SonarQube du code global génèrent des warnings.

5.4.1.7. Stratégie d'évolution des seuils

Cette configuration n'est pas statique et devrait évoluer avec la maturité du projet :

1. **Phase initiale** (actuelle) : Différenciation marquée entre nouveau code et code global.

2. **Phase intermédiaire** : À mesure que la qualité globale s'améliore, les seuils pour le code global peuvent être progressivement resserrés.

3. **Phase mature** : À terme, les seuils pour le nouveau code et le code global pourraient converger, signalant une base de code uniformément de haute qualité.

5.4.1.8. Conclusion

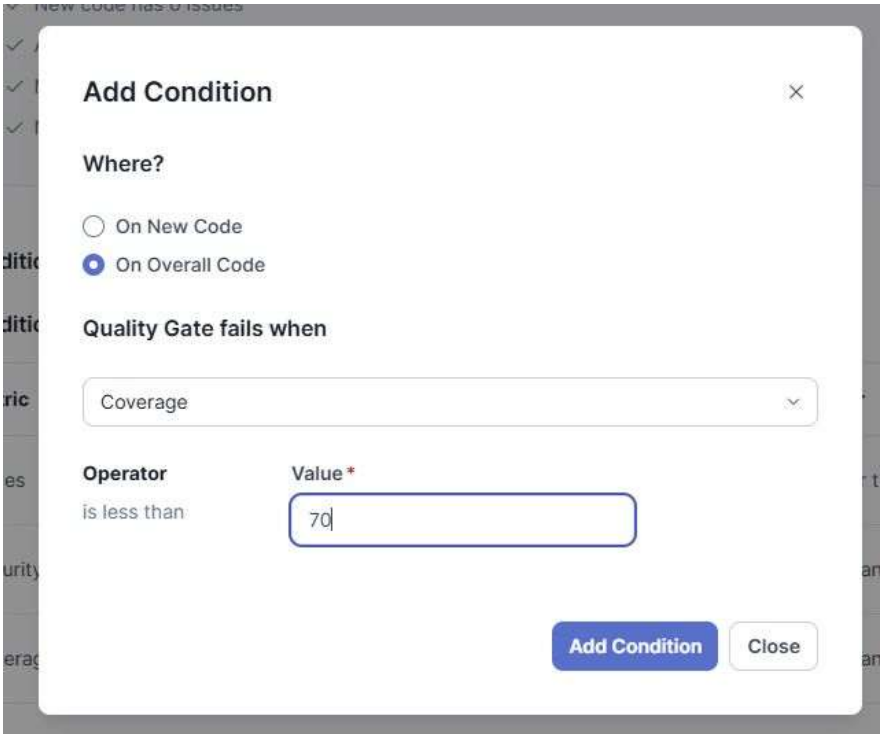
L'approche différenciée des Quality Gates pour le Nouveau Code et le Code Global représente une stratégie équilibrée qui :

- Garantit que la qualité du code s'améliore progressivement
- Reconnaît les réalités pratiques du développement logiciel
- Établit des priorités claires pour la correction des problèmes
- S'aligne avec la philosophie DevOps d'amélioration continue

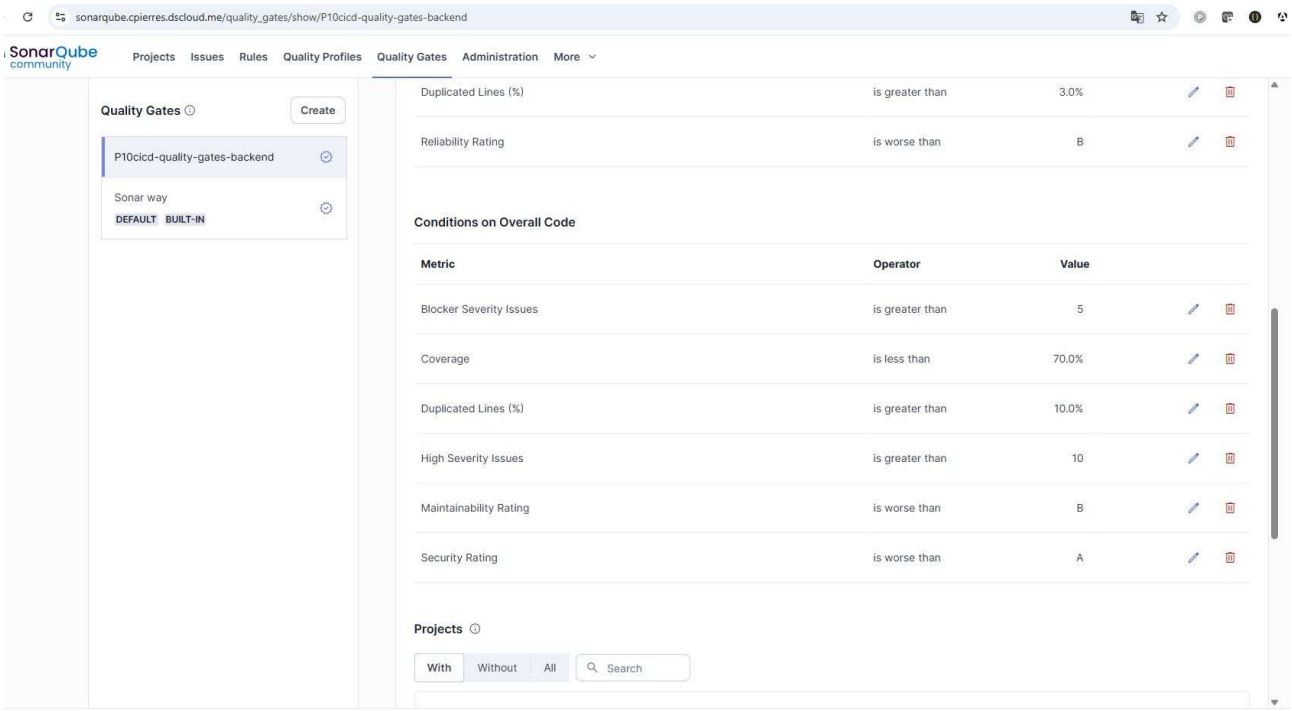
Cette configuration permet à l'équipe BobApp de maintenir un rythme de développement soutenu tout en améliorant constamment la qualité du code, créant ainsi un cercle vertueux où chaque nouvelle contribution renforce la robustesse et la maintenabilité de l'application.

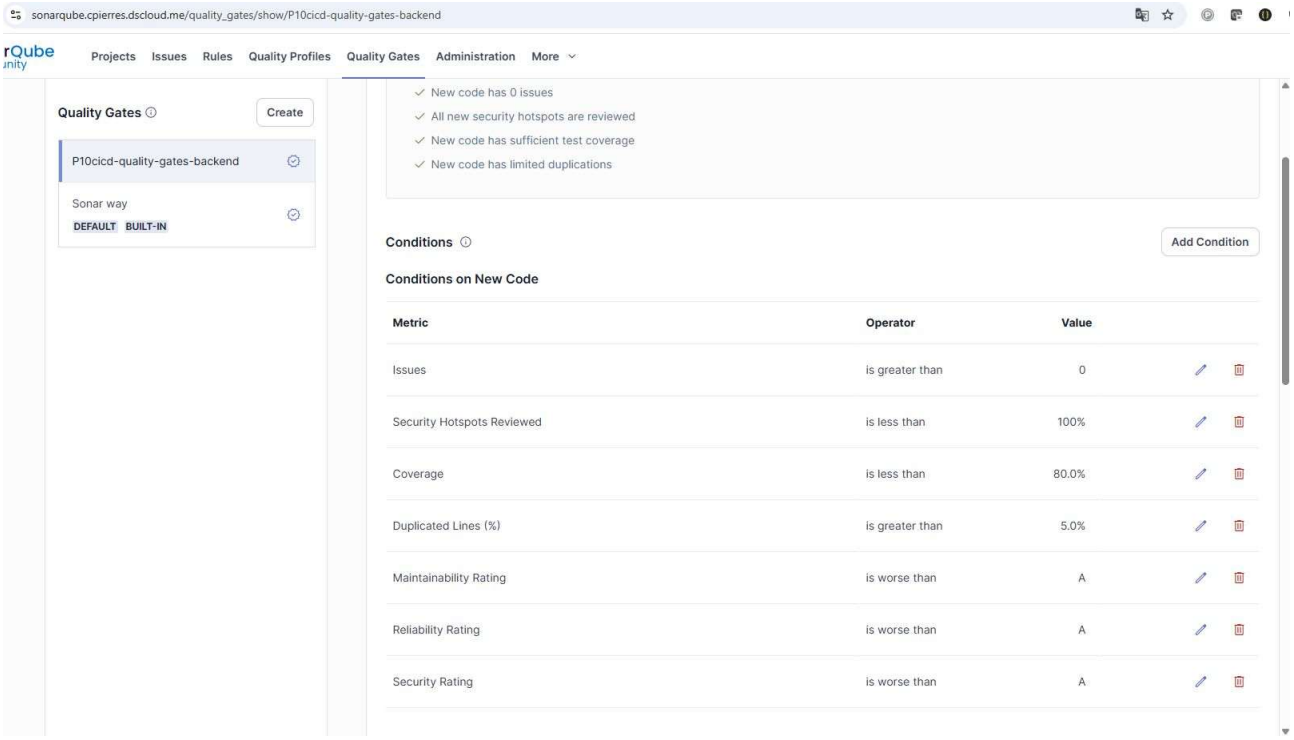
Quality Gates définies pour le projet Backend (Java/Spring Boot) se trouve en page 53. Jusqu'alors les règles prédéfinies se déclenchent uniquement sur le Nouveau code.

Dans cette section, nous allons définir nos règles sur le code global/existant (Overall code).



Voici nos conditions pour le Overall Code (code global) terminées pour le backend :





5.4.1.9. Quality Gates on New Code pour le backend

5.4.1.10. Pour rappel, la justification du choix des Avantages spécifiques de cette approche pour le projet BobApp

1. **Adaptation à la maturité du projet** :

- Si BobApp est un projet existant avec du code legacy, cette approche permet de continuer le développement tout en améliorant progressivement la qualité.
- Si BobApp est un nouveau projet, ces standards établissent dès le départ une base solide tout en reconnaissant que des compromis peuvent être nécessaires dans certaines phases.

2. **Équilibre entre rigueur et pragmatisme** :

- Les seuils ERROR pour le nouveau code garantissent que la qualité ne se dégrade pas.
- Les seuils WARNING pour le code global permettent une visibilité sur les problèmes sans bloquer le pipeline.

3. **Priorisation intelligente** :

- La sévérité ERROR sur la sécurité pour le code global force l'équipe à traiter immédiatement les vulnérabilités.
- La tolérance plus élevée pour les problèmes de maintenabilité permet de prioriser les corrections.

4. **Alignement avec les workflows CI/CD** :

- Cette configuration s'intègre parfaitement avec la séparation des workflows backend/frontend.
- Les jobs peuvent continuer même si les analyses SonarQube du code global génèrent des warnings.

5.4.1.11. Stratégie d'évolution des seuils

Cette configuration n'est pas statique et devrait évoluer avec la maturité du projet :

1. **Phase initiale** (actuelle) : Différenciation marquée entre nouveau code et code global.

2. **Phase intermédiaire** : À mesure que la qualité globale s'améliore, les seuils pour le code global peuvent être progressivement resserrés.

3. **Phase mature** : À terme, les seuils pour le nouveau code et le code global pourraient converger, signalant une base de code uniformément de haute qualité.

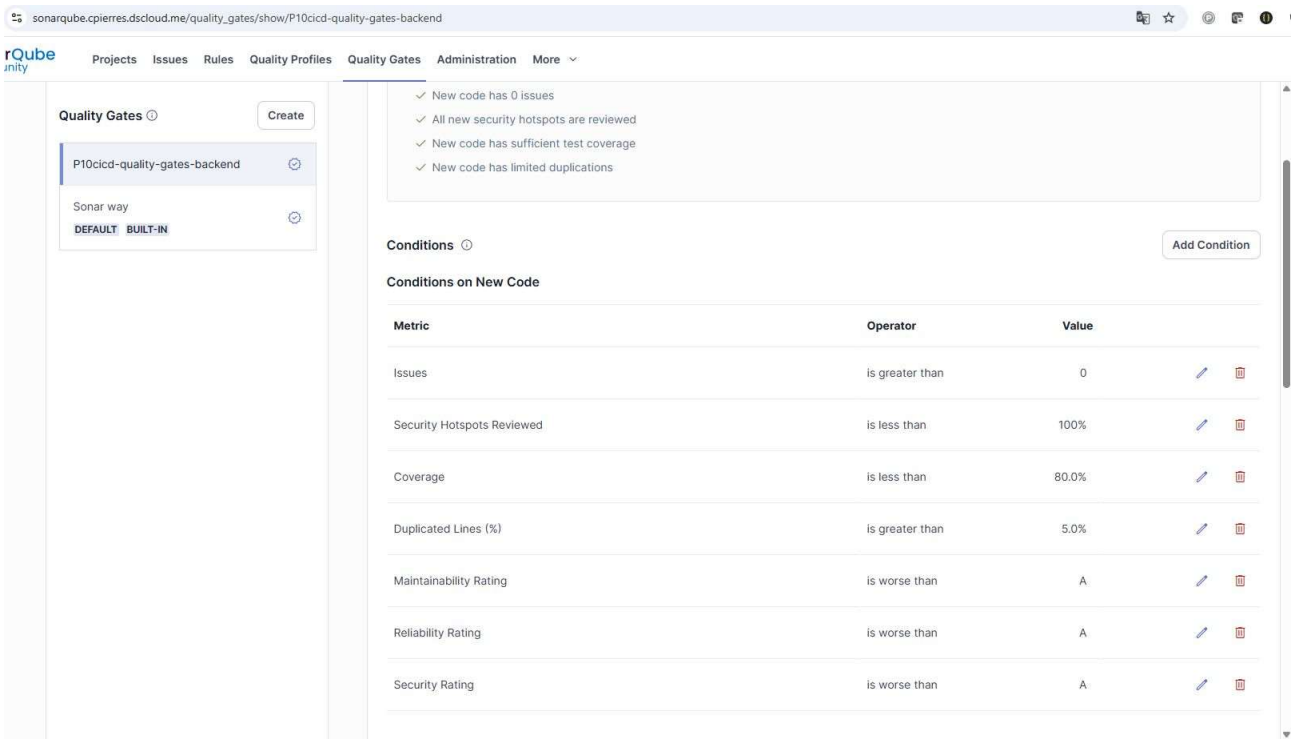
5.4.1.12. Conclusion

L'approche différenciée des Quality Gates pour le Nouveau Code et le Code Global représente une stratégie équilibrée qui :

- Garantit que la qualité du code s'améliore progressivement
- Reconnaît les réalités pratiques du développement logiciel
- Établit des priorités claires pour la correction des problèmes
- S'aligne avec la philosophie DevOps d'amélioration continue

Cette configuration permet à l'équipe BobApp de maintenir un rythme de développement soutenu tout en améliorant constamment la qualité du code, créant ainsi un cercle vertueux où chaque nouvelle contribution renforce la robustesse et la maintenabilité de l'application.

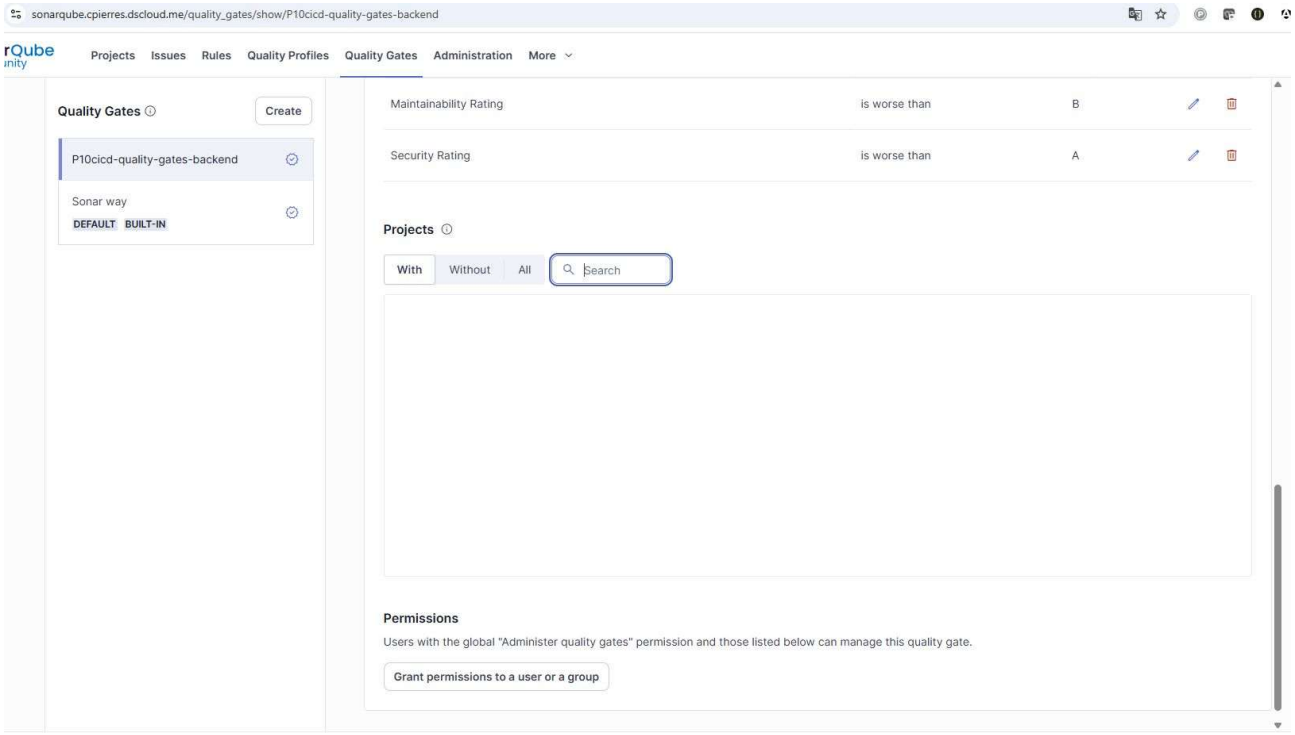
Quality Gates définies pour le projet Backend (Java/Spring Boot) se trouve en page 53. Les règles prédéfinies sur le Nouveau code doivent être modifiées pour certaines et complétées.



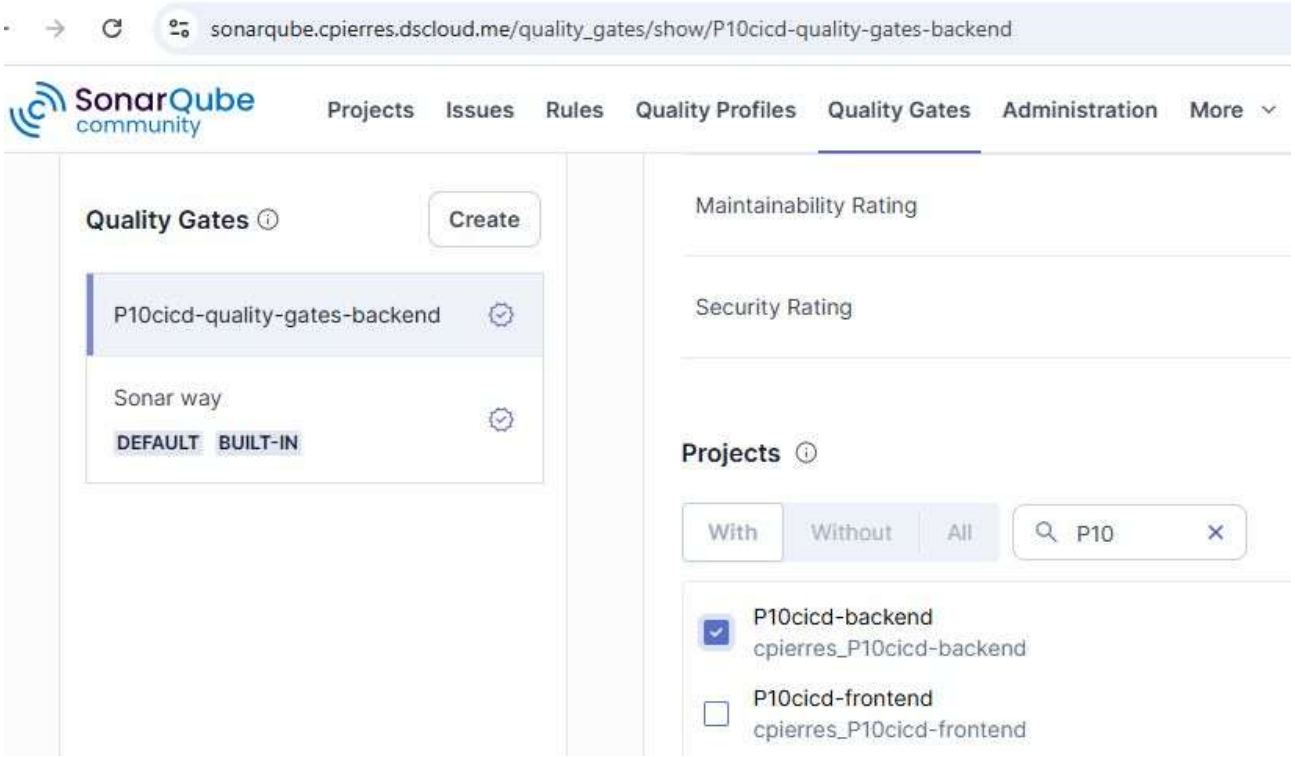
5.4.1.13. Application de la quality gates du backend sur cpierres_P10cici-backend

Il nous reste à choisir le (ou les) projets sur le(s)quel(s) appliquer notre paramétrage.

Dans la zone **Search**, saisir `P10` :

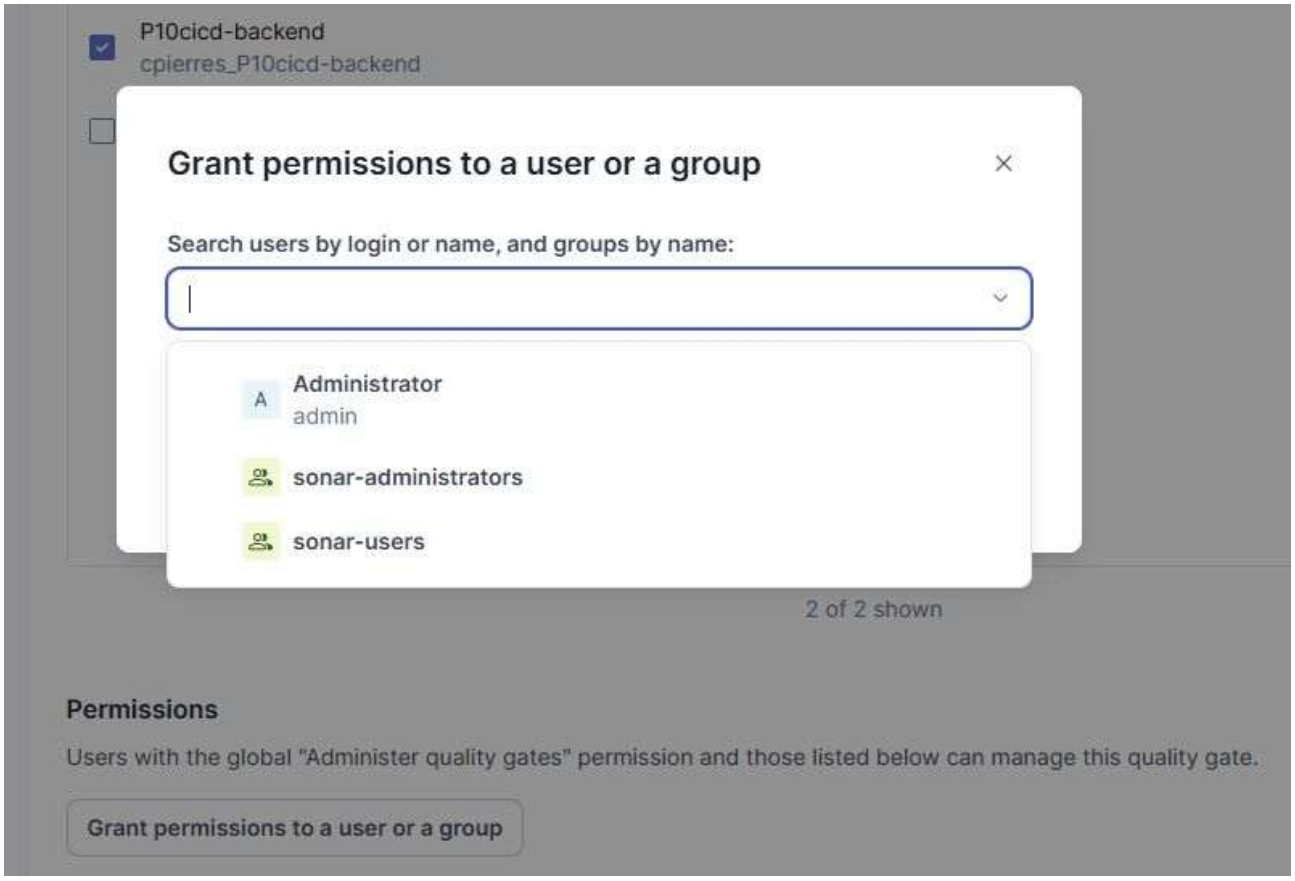


puis choisir la clé de projet `cpierres_P10cicd-backend` :



5.4.1.14. Permissions de modification

En principe, la modification devrait être restreinte au groupe des administrateurs de Sonar :

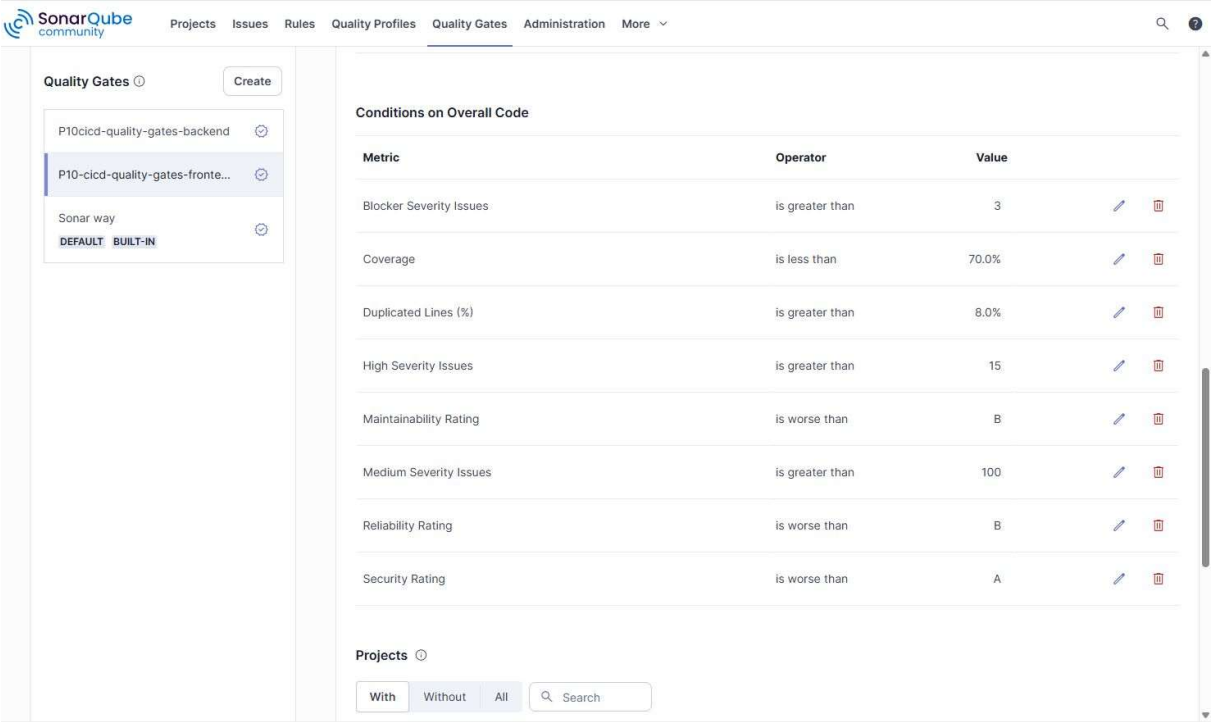

P10cicd-backend
cpierres_P10cicd-backend
Grant permissions to a user or a group
Search users by login or name, and groups by name:
A
Administrator
admin
sonar-administrators
sonar-users
2 of 2 shown
Permissions
Users with the global "Administer quality gates" permission and those listed below can manage this quality gate.
Grant permissions to a user or a group

5.4.2. Mise en place des Quality Gates pour le frontend

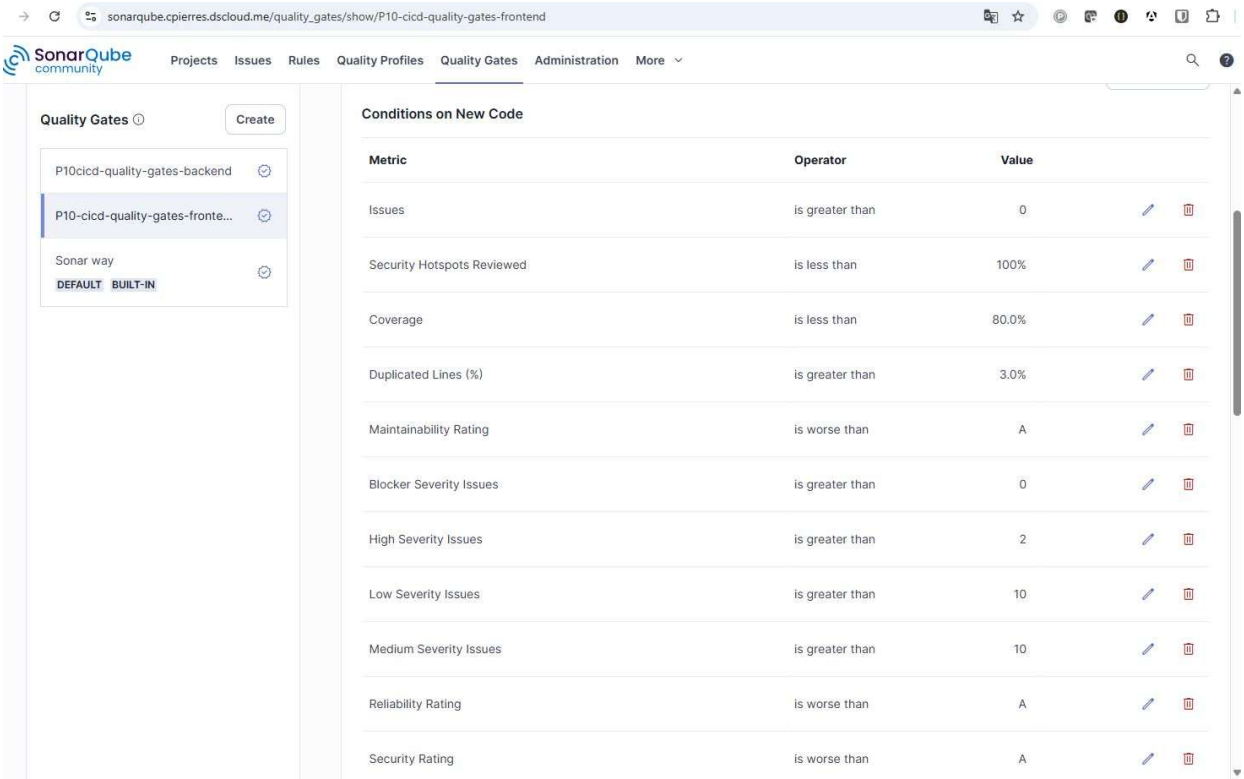
La justification du choix des Quality Gates définies pour le projet Frontend (Angular) se trouve en page 58.

La procédure est désormais connue.

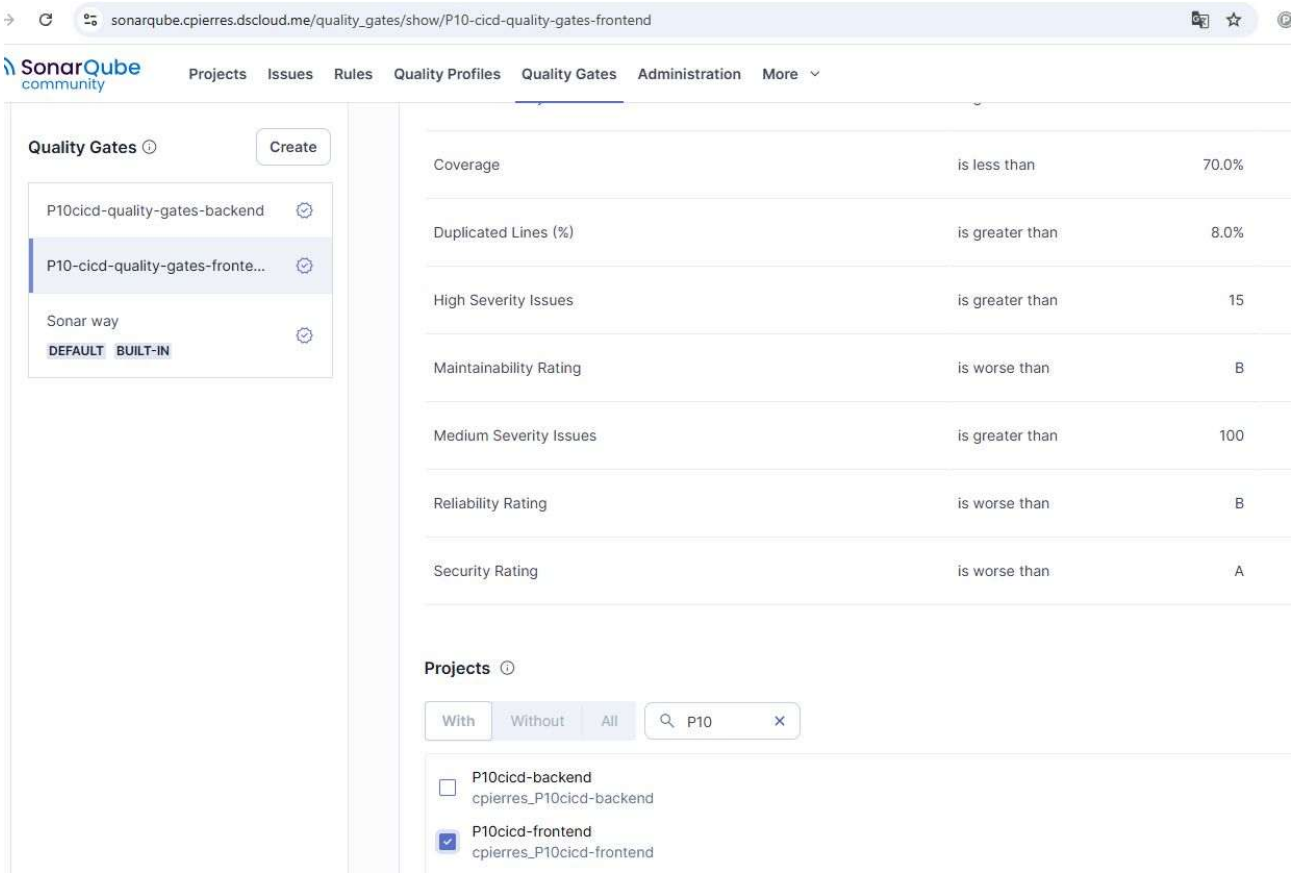
5.4.2.1. Quality Gates on Overall Code pour le frontend



5.4.2.2. Quality Gates on New Code pour le frontend

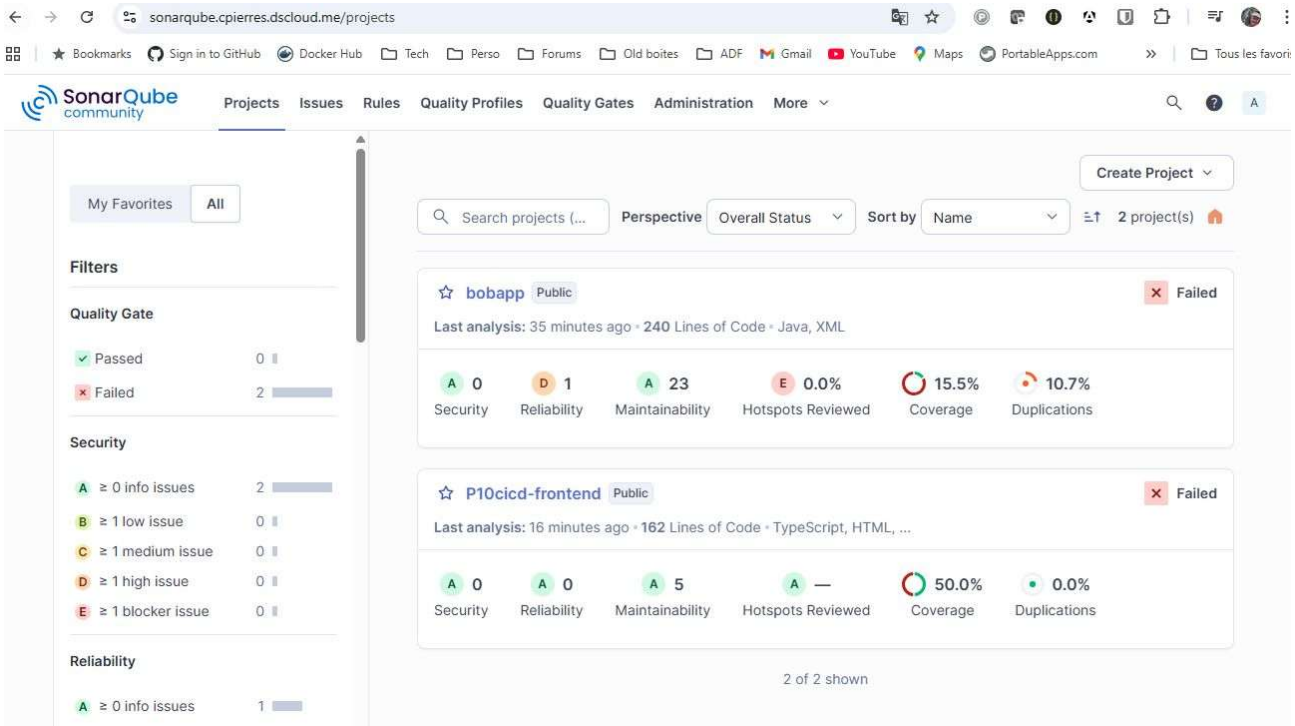


5.4.2.1. Application de la quality gates du frontend sur la clé de projet cpierres_P10cici-frontend



5.4.3. Vue des 2 projets SonarQube (un pour chaque couche/workflow)

Note dans ce premier affichage, les résultats du back sont mauvais car j'ai introduit volontairement des erreurs (code dupliqué, code mort, code complexe) afin de tester/solliciter les QualityGates :



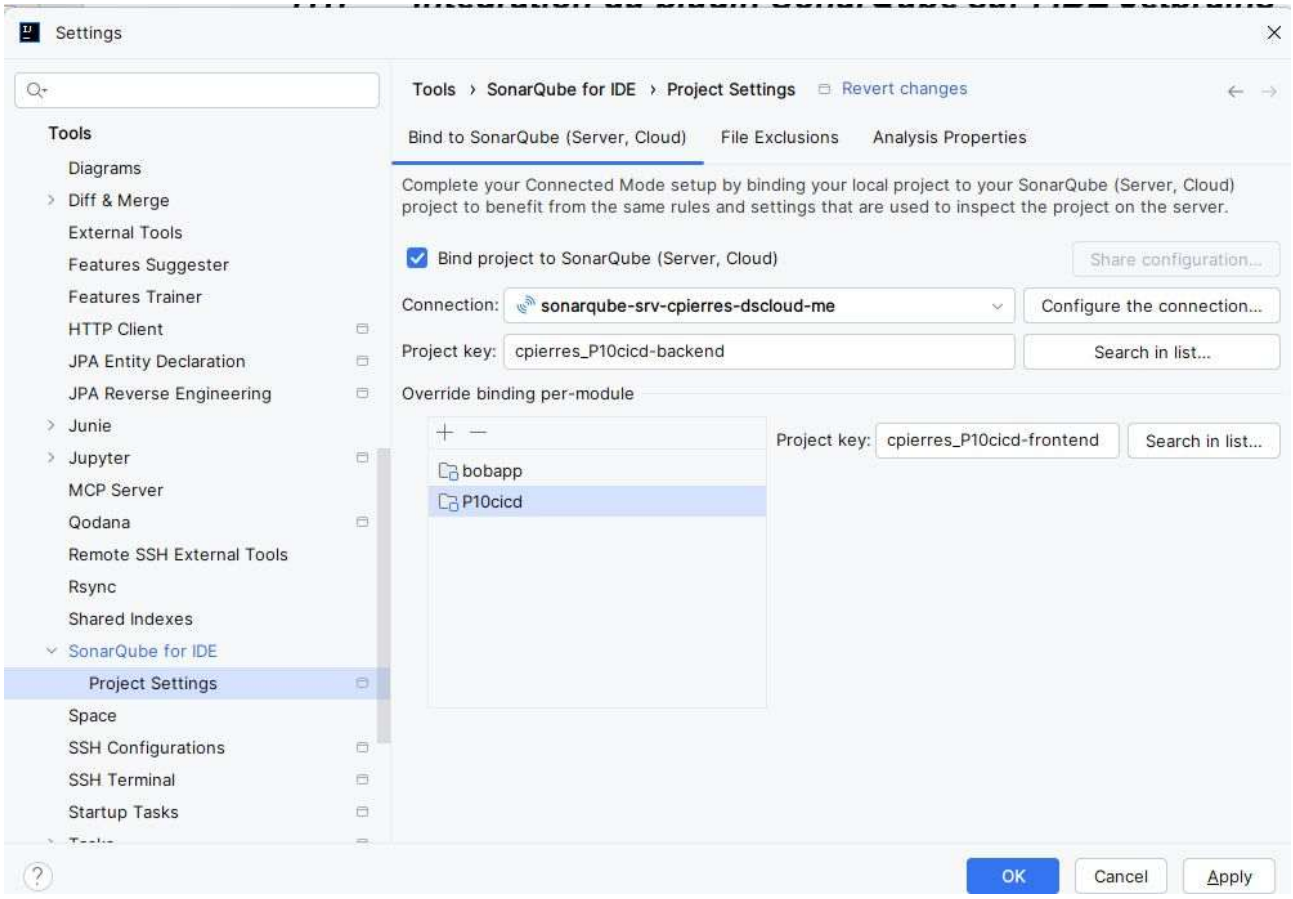
Dans ce second affichage, le code d'origine a été rétabli afin d'avoir la vue « telle quelle » (on est supposé mettre la CI/CD en place, et non pas ajouter du code) :

5.1. Intégration du plugin SonarQube dans l'IDE IntelliJ

Dès lors que l'IDE découvre un fichier sonar-project.properties dans chacun des projets, il propose d'installer le plugin et de se connecter à votre serveur SonarQube.

Les contrôles qualité se déclencheront mais n'enrichiront pas la base de données. Cela permet au développeur d'être alerté au plus tôt avant de faire un commit.

Pour bénéficier de l'analyse séparée entre le back et le front, il faut générer dans SonarQube un token pour l'IDE et surtout bien mapper les deux projets back et front pour que chaque projet bénéficie de ses Quality gates personnalisées :



5.1. Possibilité pour le développeur d'exécuter les workflows sur le réseau local sur sa branche (avant la PR)

Sur une branche non référencée dans le workflow, celui-ci ne s'exécutera pas automatiquement lors du push dans github.

Aussi il est intéressant de pouvoir déclencher les workflows pour chaque couche sur le code qui n'a pas été commité.

Contrairement au plugin de l'IDE, les contrôles effectués par les batchs seront enregistrés dans la base de données de notre instance de SonarQube (exposée sur internet mais également disponible sur le réseau local).

Cela permettra ainsi de suivre les efforts des développeurs pour corriger leurs problèmes avant de les soumettre à une Pull Request.

Deux batchs (pour windows) ont été créés à la racine pour cela :

- Pour le projet back : `sonar-local-analyze-backend.bat`

- Pour le projet front : `sonar-local-analyze-frontend.bat`

Attention : les token `SONAR_TOKEN_LOCAL_BACKEND` et `SONAR_TOKEN_LOCAL_FRONTEND` doivent être enregistrées dans les variables d'environnement des développeurs.

Ces deux batchs, prévus pour windows, devront être adaptés pour l'OS de votre IDE si nécessaire.

5.1.1. sonar-local-analyze-backend.bat

```
echo off
REM call set-env.bat
cd back
mvn clean verify sonar:sonar "-Dsonar.projectKey=cpierres_P10cicd-backend" "-Dsonar.projectName=P10cicd-backend"
"-Dsonar.host.url=http://localhost:9000" -Dsonar.token=%SONAR_TOKEN_LOCAL_BACKEND%
```

5.1.2. sonar-local-analyze-frontend.bat

Contenu :

```
@echo off
cd front
REM npm test -- --no-watch --no-progress --browsers=ChromeHeadless --code-coverage
npm run test-coverage
sonar-scanner "-Dsonar.host.url=http://localhost:9000" -Dsonar.token=%SONAR_TOKEN_LOCAL_FRONTEND% -
Dsonar.projectKey=cpierres_P10cicd-frontend
```

6. Exploitation des workflows

A noter que j'ai introduit différentes anomalies dans le code (code dupliqué, code complexe, code mort) afin de déclencher volontairement les alertes de SonarQube.

Nous allons procéder en 3 étapes :

- Test du workflow avec modif uniquement sur le back
- Test du workflow avec modif uniquement sur le front
- Test du workflow avec modif à la fois sur le back et le front (remise en place du code d'origine)

Dans le test final (test avec modifications des deux couches simultanément), je rétablirai le code « standard » qui était présent afin que vous puissiez vérifié les valeurs que vous attendiez, puis je donnerai une interprétation de ces résultats comme attendu.

Pour info, j'avais commencé mes tests avec SonarCloud ; c'est pourquoi, vous ne trouverez pas d'analyses anciennes dans le SonarQube Community, car plus récemment installé.

6.1. Test du workflow avec modif uniquement sur le back

Dans ces sections, nous montrons le cycle complet de création de PR puis de validation par le reviewer.

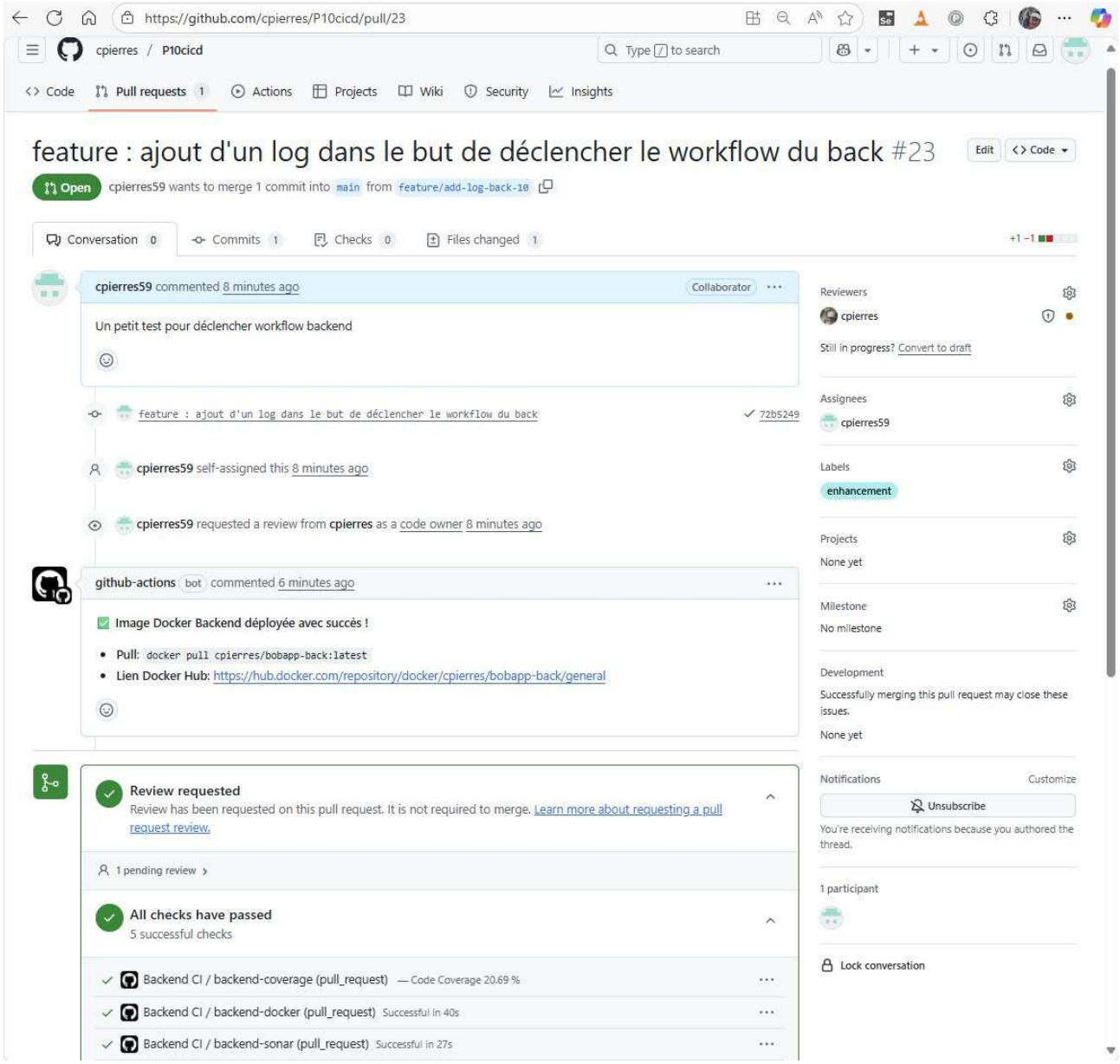
6.1.1. Création d'une PR sur le back par cpierres59 (autre user que propriétaire)

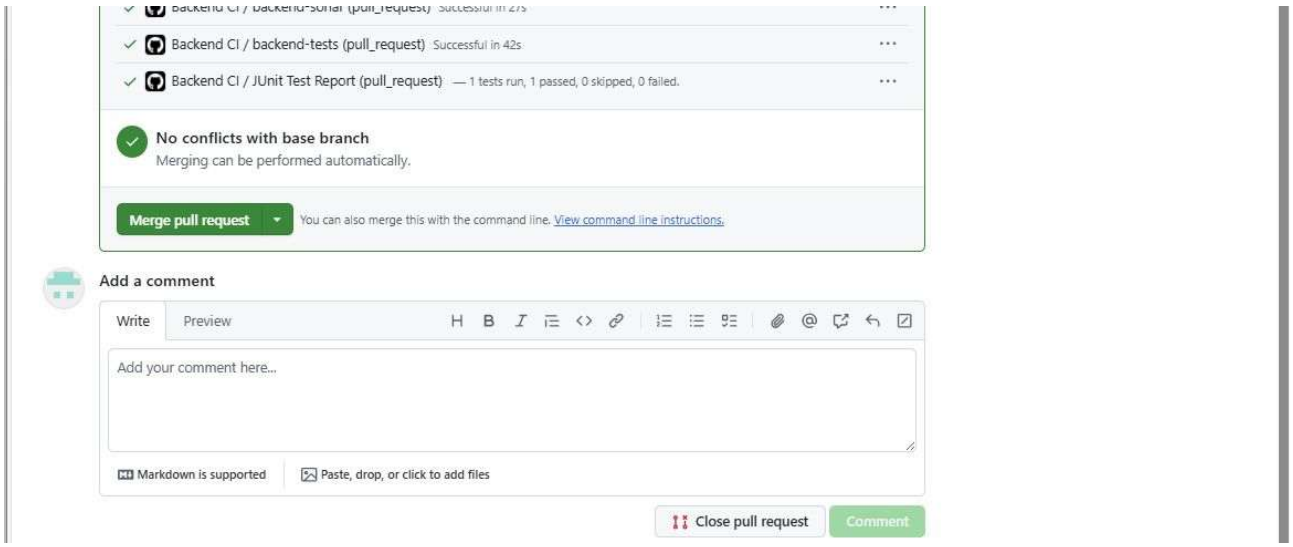
Le collaborateur cpierres59 a pushé un commit sur le backend puis a généré la PR.

Seul le workflow Backend CI s'est déclenché (pas celui du frontend puisqu'aucun fichier n'a été modifié dans celui-ci).

On constate dans le cadre des **Checks** que les jobs et steps du backend se sont déclenchés. Ensuite un step du job docker a généré un commentaire sur la PR pour indiquer la disponibilité de l'image docker déployée.

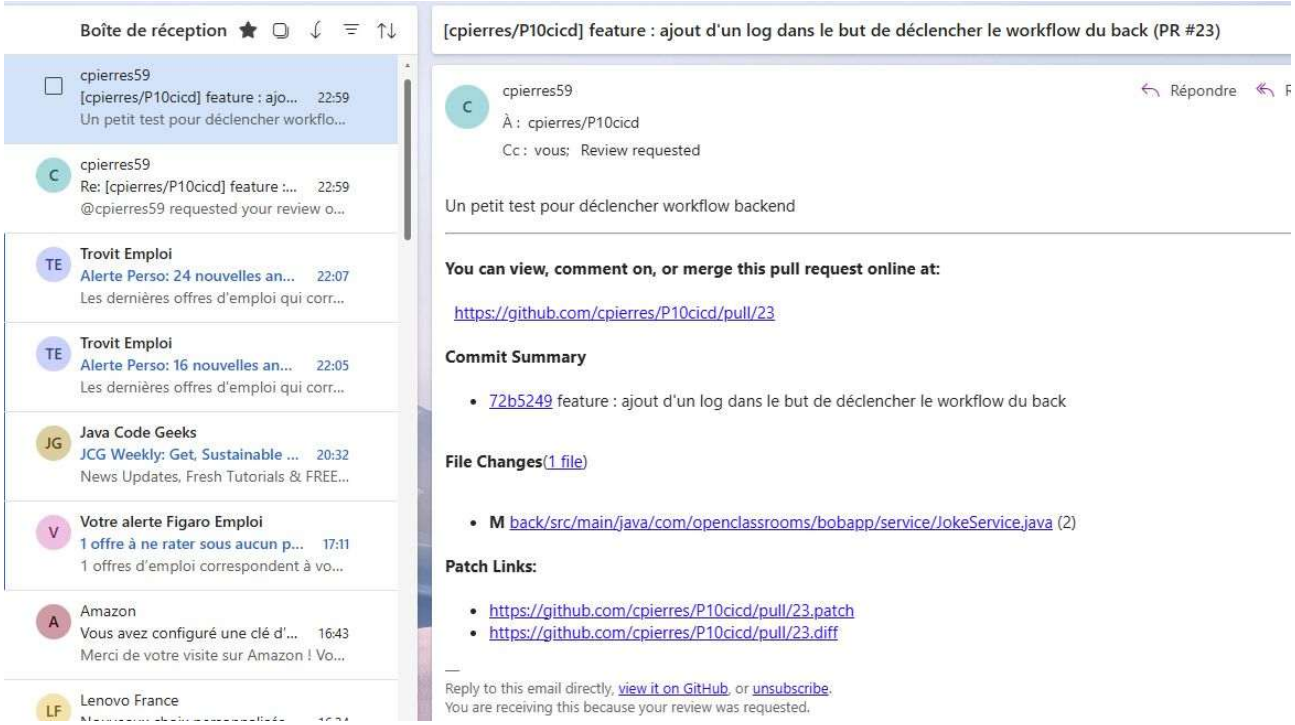
Une review a été demandé à cpierres (qui est le codeowner). Il va recevoir deux emails pour être prévenu.





✅ A noter que grâce à nos protections de dépôt, ce collaborateur ne pourrait pas merger cette PR vers `main` sans au moins une approbation d'un reviewer. Une erreur apparaitrait lors d'une tentative.

Le reviewer cpierres reçoit deux emails pour gérer la PR :



6.1.1.1. Dans onglet Actions, cliquer sur le commit qui a déclenché le workflow afin d'afficher le détail (et récupérer les artefacts au besoin)

Nous voyons dans la partie centrale les 3 jobs principaux qui se sont déclenchés :
backend-tests => backend-sonar => backend-docker.

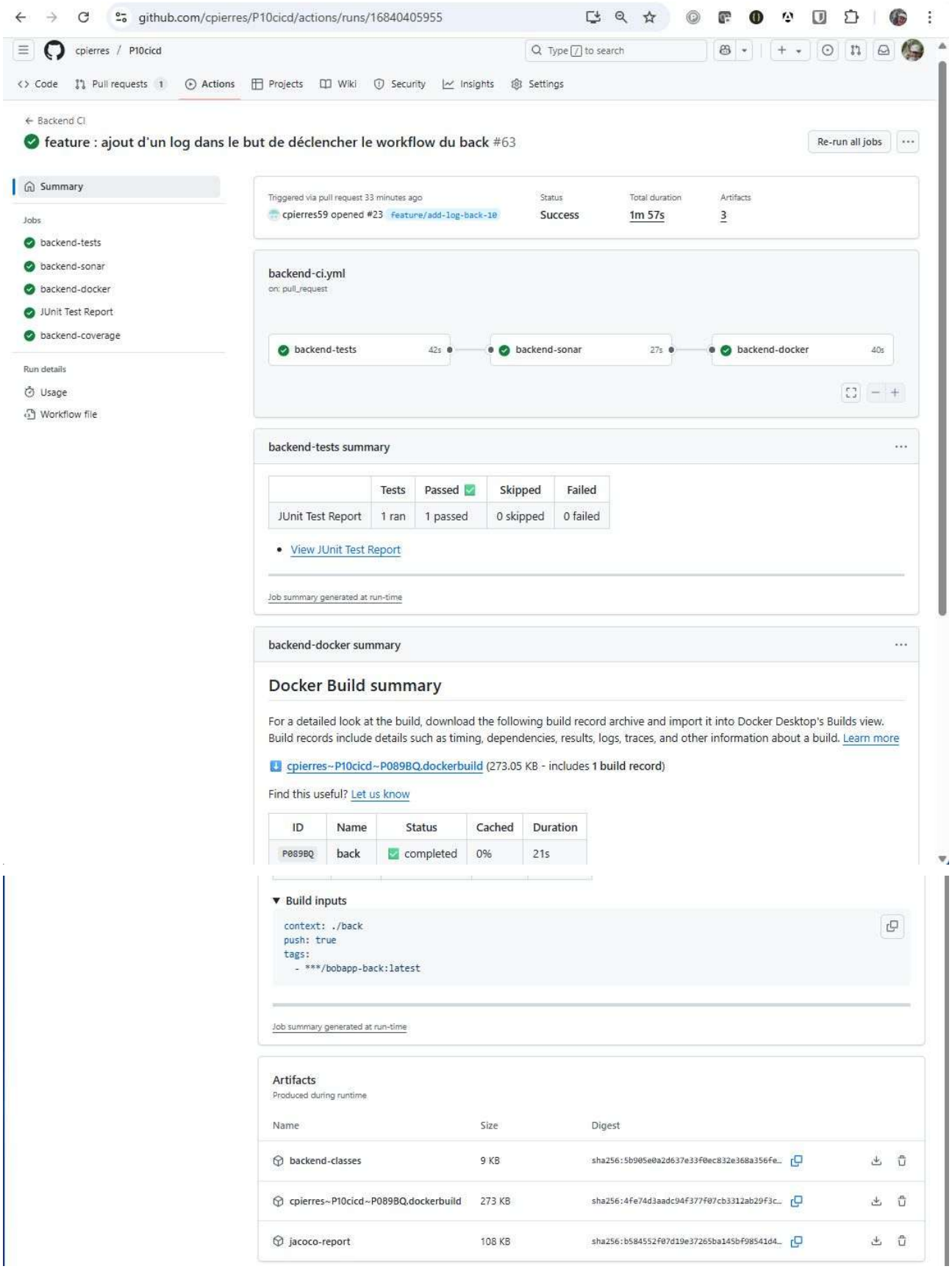
Certains plugins spécifiés dans des steps ont aussi déclenché leur propre job secondaire que l'on peut voir sur la gauche (JUnit Test report et backend-coverage).

Enfin, tout en bas, on peut voir <u>les artefacts générés</u> :

- les classes du backend qui ont été transmises pour analyse au job sonar (ce dernier est séparé du job des tests du fait du JDK 11 requis par le back)

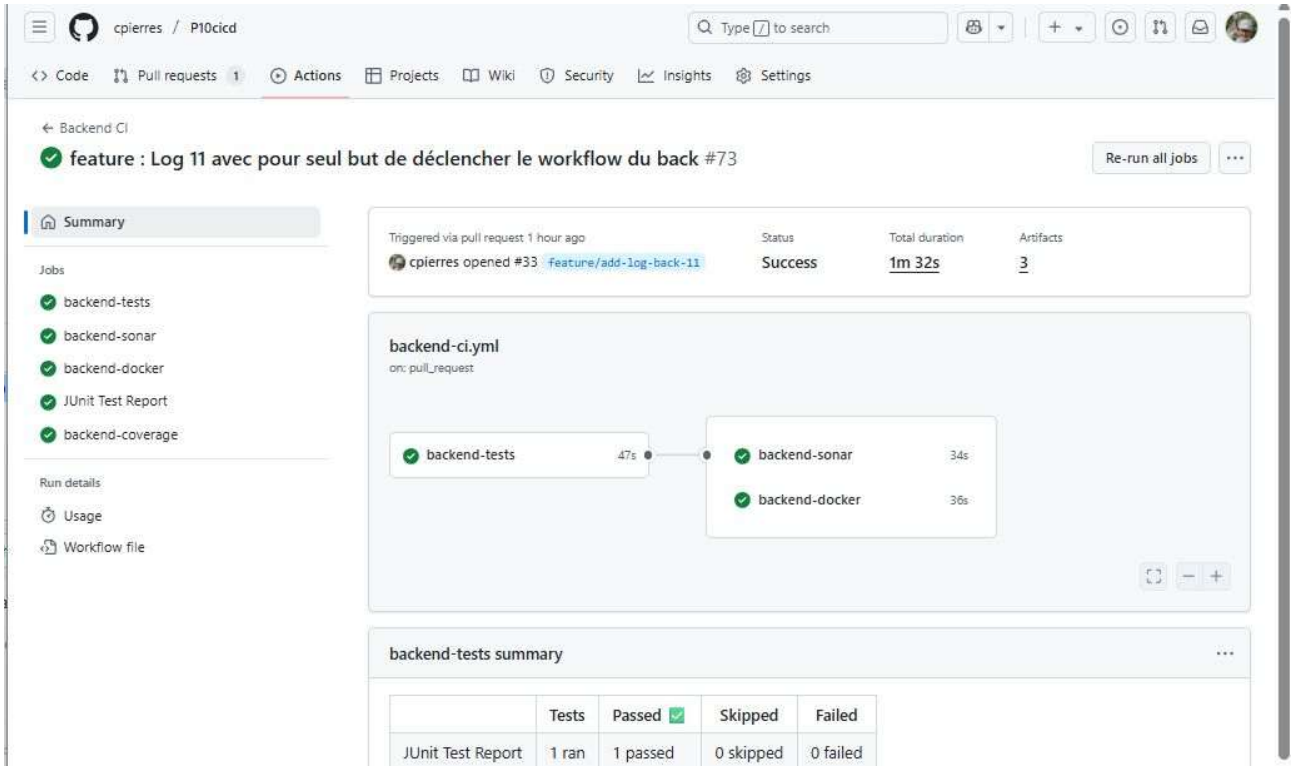
- le rapport jacoco de couverture des tests

- dockerbuild (rappel un commentaire indique dans la PR le lien direct vers l'image dans docker hub)

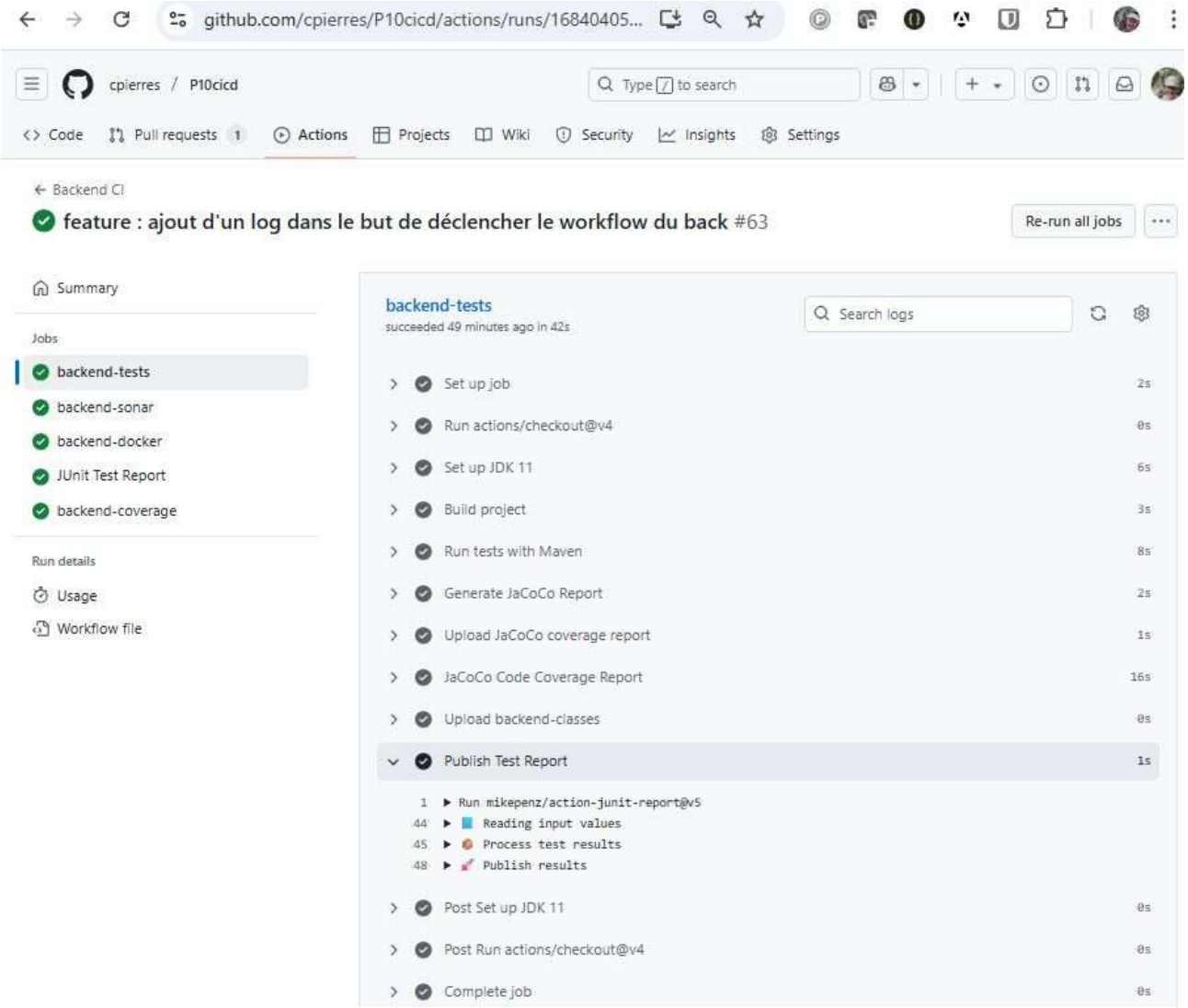


Optimisation du workflow pour parallèlisation du job backend-sonar et du job backend-docker

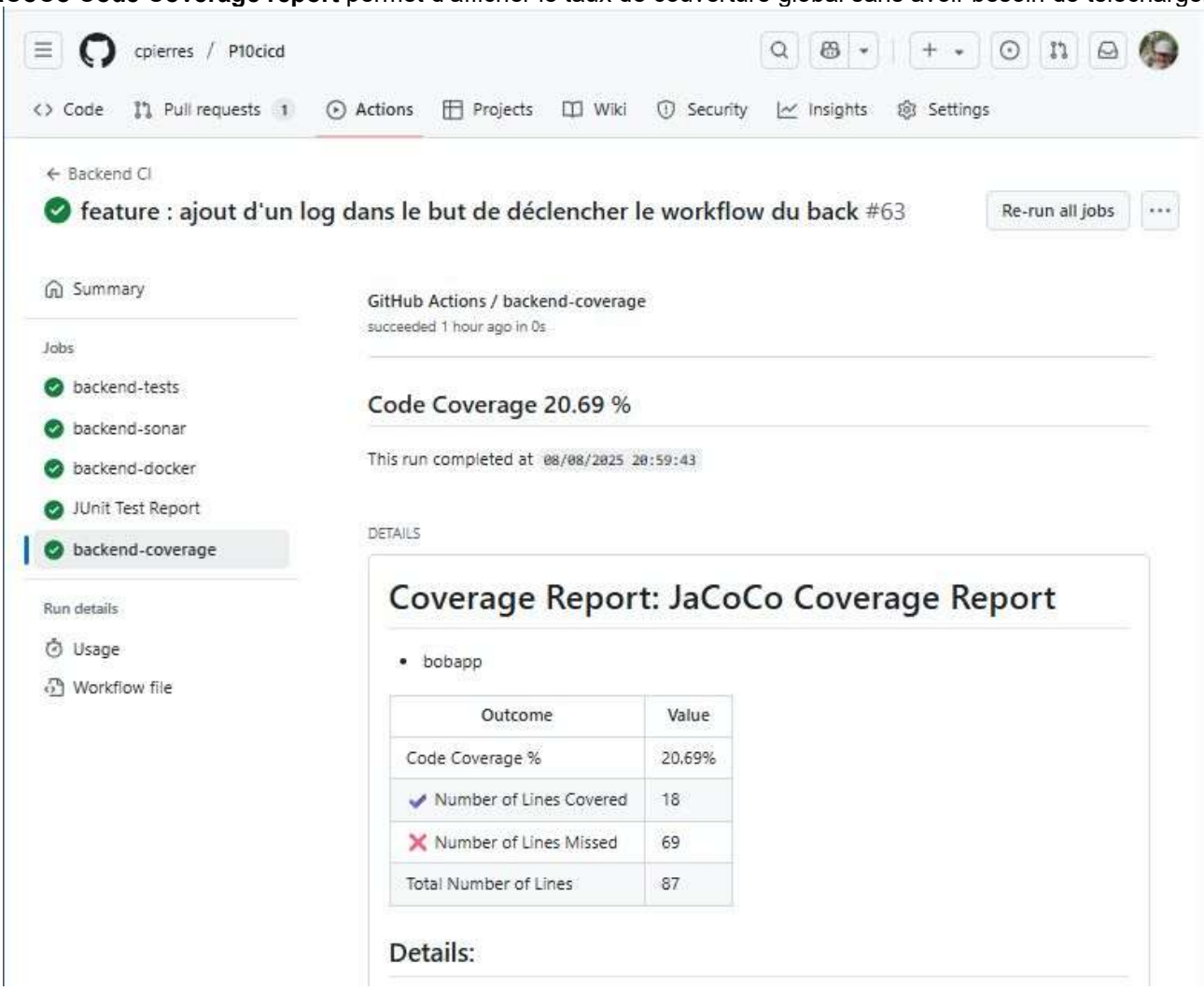
backend-ci.yml et frontend-ci.yml ont été optimisés afin de permettre aux jobs sonar et docker de s'exécuter en parallèle. Pour rappel, le job sonar n'est pas bloquant pour la production de l'image docker.


cpierres / P10cicd
Type / to search
Code
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings
Backend CI
feature : Log 11 avec pour seul but de déclencher le workflow du back #73
Re-run all jobs
Summary
Jobs
backend-tests
backend-sonar
backend-docker
JUnit Test Report
backend-coverage
Run details
Usage
Workflow file
Triggered via pull request 1 hour ago
cpierres opened #33 feature/add-log-back-11
Status
Success
Total duration
1m 32s
Artifacts
3
backend-ci.yml
on: pull_request
backend-tests 47s
backend-sonar 34s
backend-docker 36s
backend-tests summary
Tests
Passed
Skipped
Failed
JUnit Test Report
1 ran
1 passed
0 skipped
0 failed

6.1.1.2. Cliquer sur le job backend-tests afin d'afficher les steps et leur détail



Le step **JaCoCo Code Coverage report** permet d'afficher le taux de couverture global sans avoir besoin de télécharger l'artefact :



Le step **Publish Test Report** a déclenché l'action-junit-report pour afficher le résultat des tests sur l'écran détail, sans avoir besoin de télécharger l'artefact.

6.1.1.3. Cliquer sur le job backend-sonar afin d'afficher les steps et leur détail

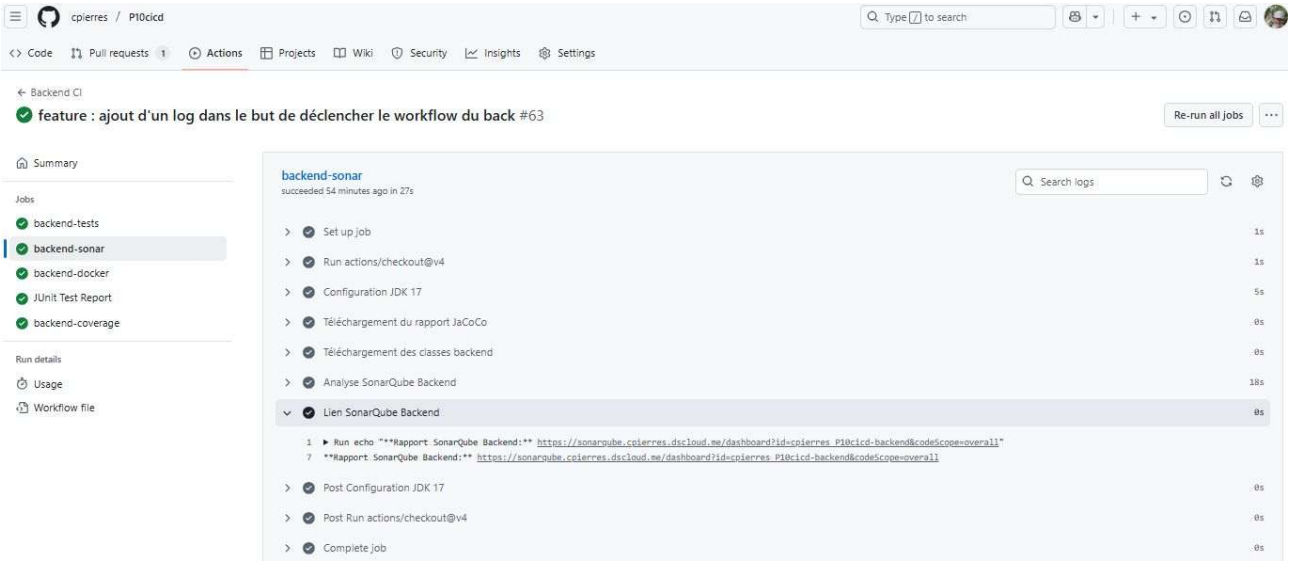
Pour son fonctionnement, Sonar a besoin du JDK 17.

On récupère du job des tests tout ce dont a besoin SonarQube pour faire son analyse :

- le rapport Jacoco

- ainsi que les classes compilés avec le JDK11 du job précédent (très important de séparer les tests du contrôle qualité pour respecter la contrainte du JDK de production et éviter toute sorte de différences)

A la fin un step génère le lien pour afficher directement le rapport SonarQube du backend.

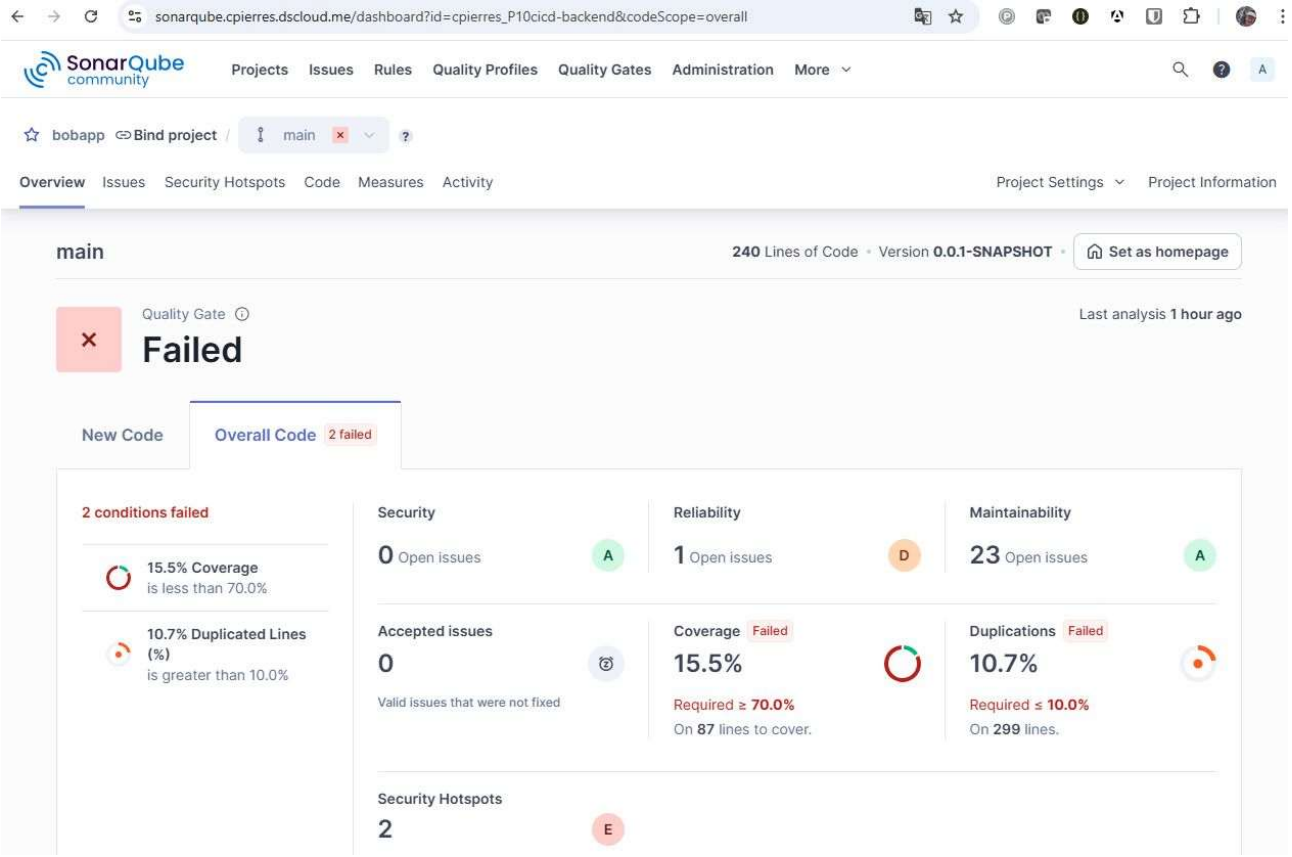


<u>Affichage du rapport SonarQube directement depuis le step Lien SonarQube Backend</u>

Rappel : je n'ai pu utiliser l'accès anonyme. Veuillez utiliser :

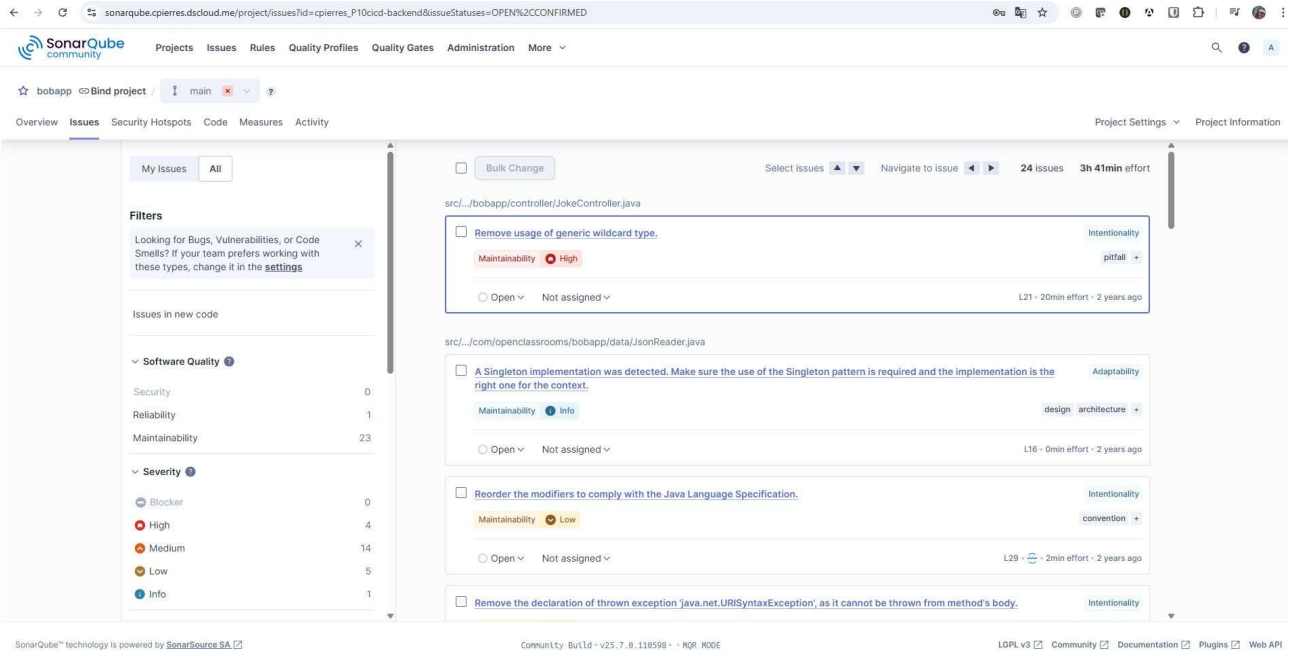
login : `ocrGuest`

password : `ocrGuest123!`

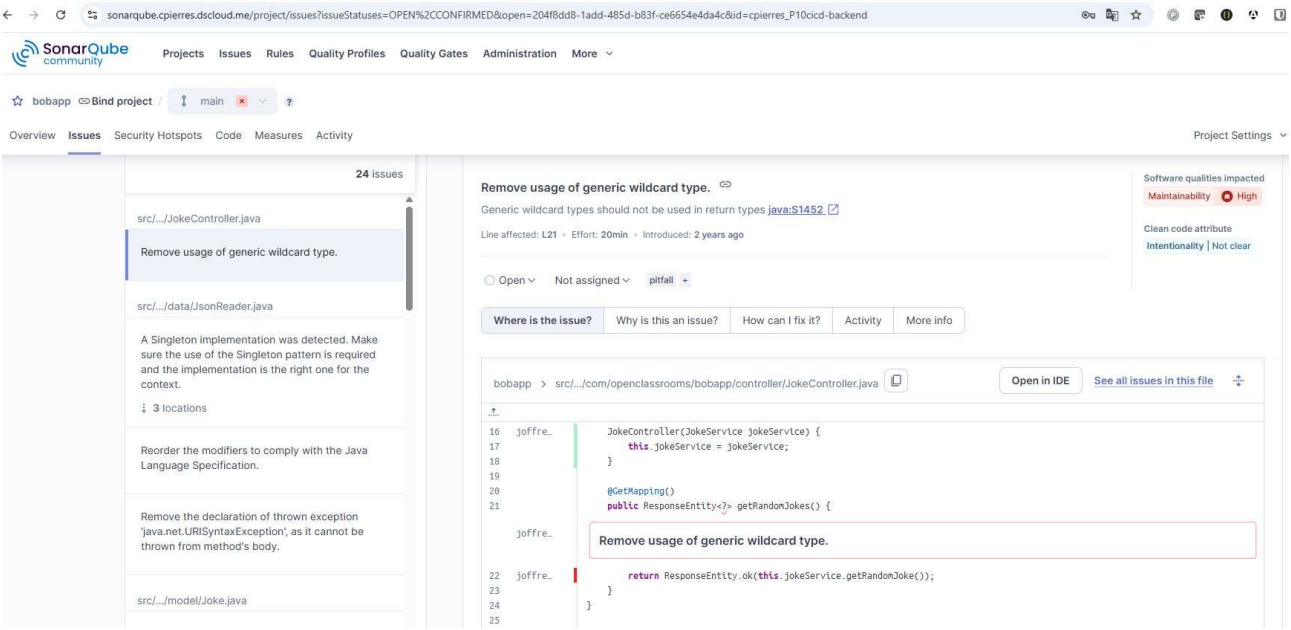


Malgré des résultats SonarQube en berne, on a néanmoins choisi de produire l'image docker (pas bloquant). Pour rappel, j'ai volontairement introduit du mauvais code (duplication, complexité) pour solliciter les indices.

Afficher les Issues classés par sévérité

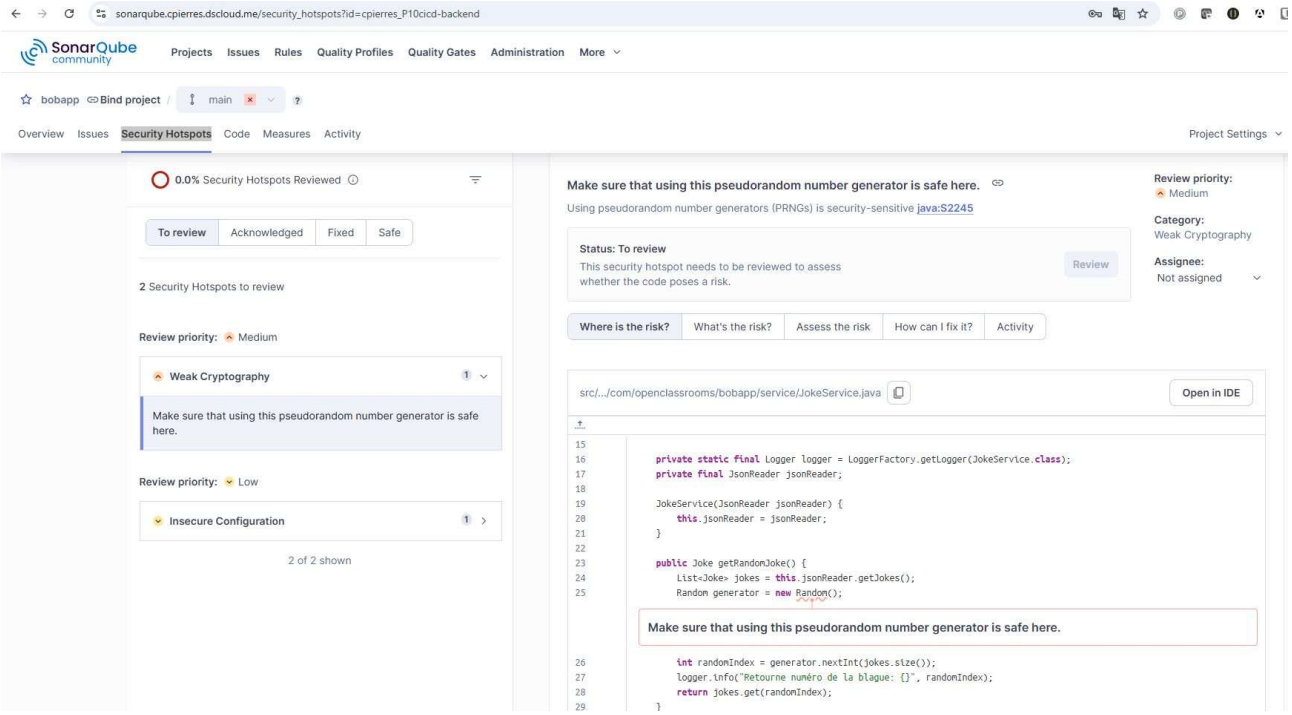


Cliquer sur le détail de **Remove usage of generic wildcard type** :



✅ Comme le plugin de IntelliJ a été configuré, vous pouvez ouvrir le code directement dans l'IDE pour faire la correction.

Afficher les Security Hotspots



6.1.1.4. Cliquer sur le job backend-docker afin d'afficher les steps et leur détail

On peut voir le step de génération du commentaire sur la PR pour indiquer le lien de l'image vers dockerhub :

cpierres / P10cicd

Type / to search

Code | Pull requests 1 | Actions | Projects | Wiki | Security | Insights | Settings

← Backend CI

feature : ajout d'un log dans le but de déclencher le workflow du back #63

Re-run all jobs

Summary

Jobs

- backend-tests
- backend-sonar
- backend-docker
- JUnit Test Report
- backend-coverage

Run details

- Usage
- Workflow file

backend-docker
succeeded 1 hour ago in 40s

Search logs

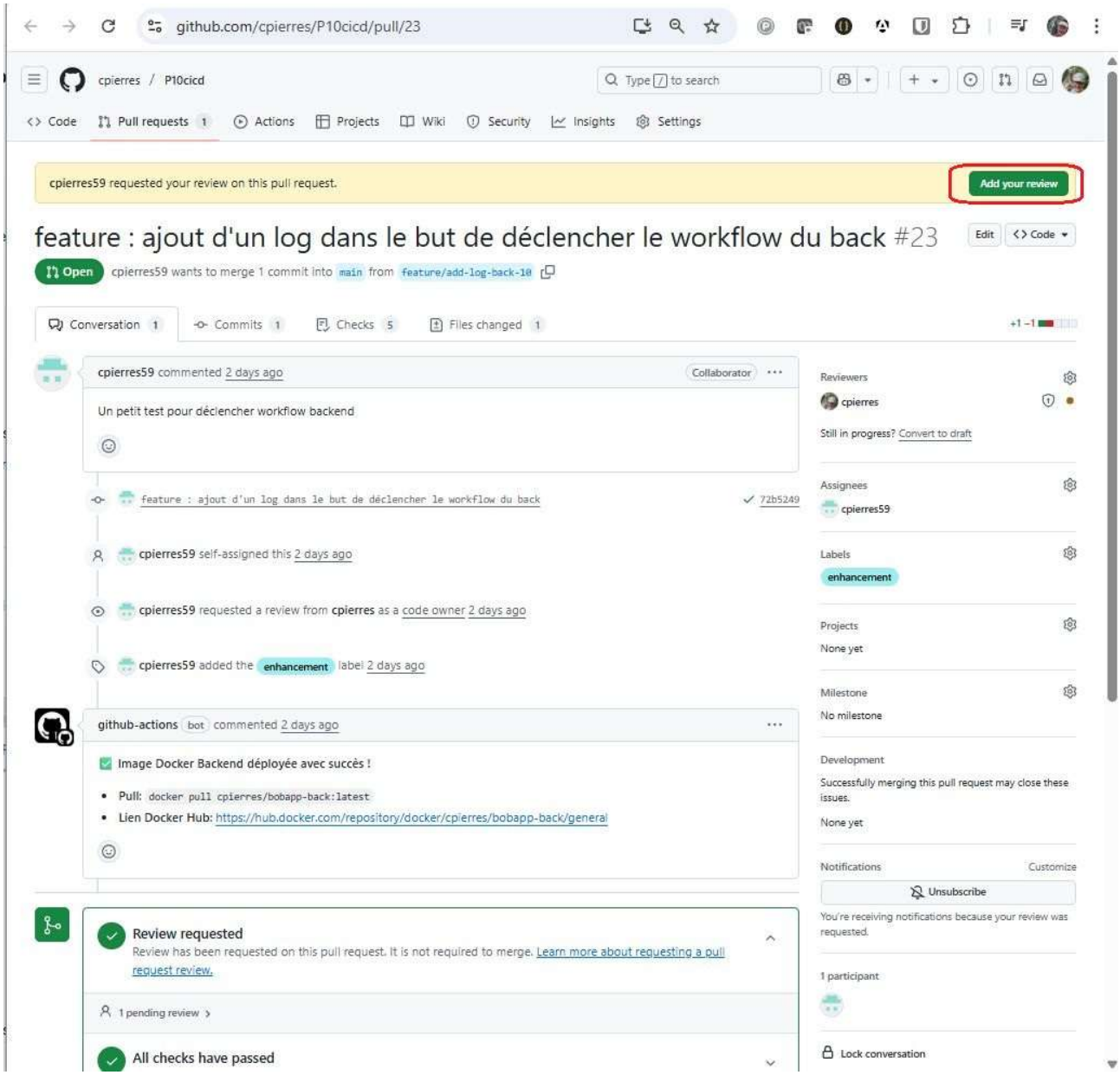
- Set up job 1
- Run actions/checkout@v4 1
- Login to Docker Hub 1
- Build and push backend 32
- Commenter la PR avec le lien Docker 1

```
1  ▼ Run actions/github-script@v7
2    with:
3      github-token: ***
4      script: const { issue: { number: issue_number }, repo: { owner, repo } } = context;
5    github.rest.issues.createComment({
6      owner,
7      repo,
8      issue_number,
9      body: `✅ **Image Docker Backend déployée avec succès !**
10     - **Pull**: \`docker pull ***/bobapp-back:latest\`
11     - **Lien Docker Hub**: [https://hub.docker.com/repository/docker/***/bobapp-back/general]
(https://hub.docker.com/repository/docker/***/bobapp-back/general)
12     `
13   });
14
15     debug: false
16     user-agent: actions/github-script
17     result-encoding: json
18     retries: 0
19     retry-exempt-status-codes: 400,401,403,404,422
```

- Post Build and push backend 1
- Post Login to Docker Hub 0
- Post Run actions/checkout@v4 0
- Complete job 0

Attention : pour une raison de sécurité le docker_username n'apparait pas dans les logs. Cependant, il apparaitra bien dans le commentaire.

6.1.2. L'approbateur cpierres approuve la PR (malgré les avertissements dans Overall Code)


github.com/cpierres/P10cicd/pull/23
cpierres / P10cicd
Type / to search
Code
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings
cpierres59 requested your review on this pull request.
Add your review
feature : ajout d'un log dans le but de déclencher le workflow du back #23
Edit
Code
Open
cpierres59 wants to merge 1 commit into main from feature/add-log-back-10
Conversation 1
Commits 1
Checks 5
Files changed 1
+1 −1
cpierres59 commented 2 days ago
Collaborator
Un petit test pour déclencher workflow backend
feature : ajout d'un log dans le but de déclencher le workflow du back
72b5249
cpierres59 self-assigned this 2 days ago
cpierres59 requested a review from cpierres as a code owner 2 days ago
cpierres59 added the enhancement label 2 days ago
github-actions bot commented 2 days ago
Image Docker Backend déployée avec succès !
Pull: docker pull cpierres/bobapp-back:latest
Lien Docker Hub: https://hub.docker.com/repository/docker/cpierres/bobapp-back/general
Review requested
Review has been requested on this pull request. It is not required to merge. Learn more about requesting a pull request review.
1 pending review
All checks have passed
Reviewers
cpierres
Still in progress? Convert to draft
Assignees
cpierres59
Labels
enhancement
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because your review was requested.
1 participant
Lock conversation

cpierres / P10cicd

Code | Pull requests 1 | Actions | Projects | Wiki | Security | Insights | Settings

feature : ajout d'un log dans le but de déclencher le workflow du back #23

Open cpierres59 wants to merge 1 commit into `main` from `feature/add-log-back-10`

Conversation 1 | Commits 1 | Checks 5 | Files changed 1

back/src/main/java/com/openclassrooms/bobapp/service/JokeService.

```
@@ -31,7 +31,7 @@ public Joke getRandomJoke() {
31  31      // méthode avec plusieurs branches conditionnelles et sans test pc
32  32      public String complexMethodWithoutTests(int value, String input) {
33  33          logger.debug("Valeur: {}, Texte: {}", value, input);
34    -         logger.debug("Ajout d'un log 9 pour déclencher le test du work
    34  +       logger.debug("Ajout d'un log 10 pour déclencher le test du wor
35  35          StringBuilder result = new StringBuilder();
36  36
37  37          if (value > 100) {
```

Finish your review

Write | Preview

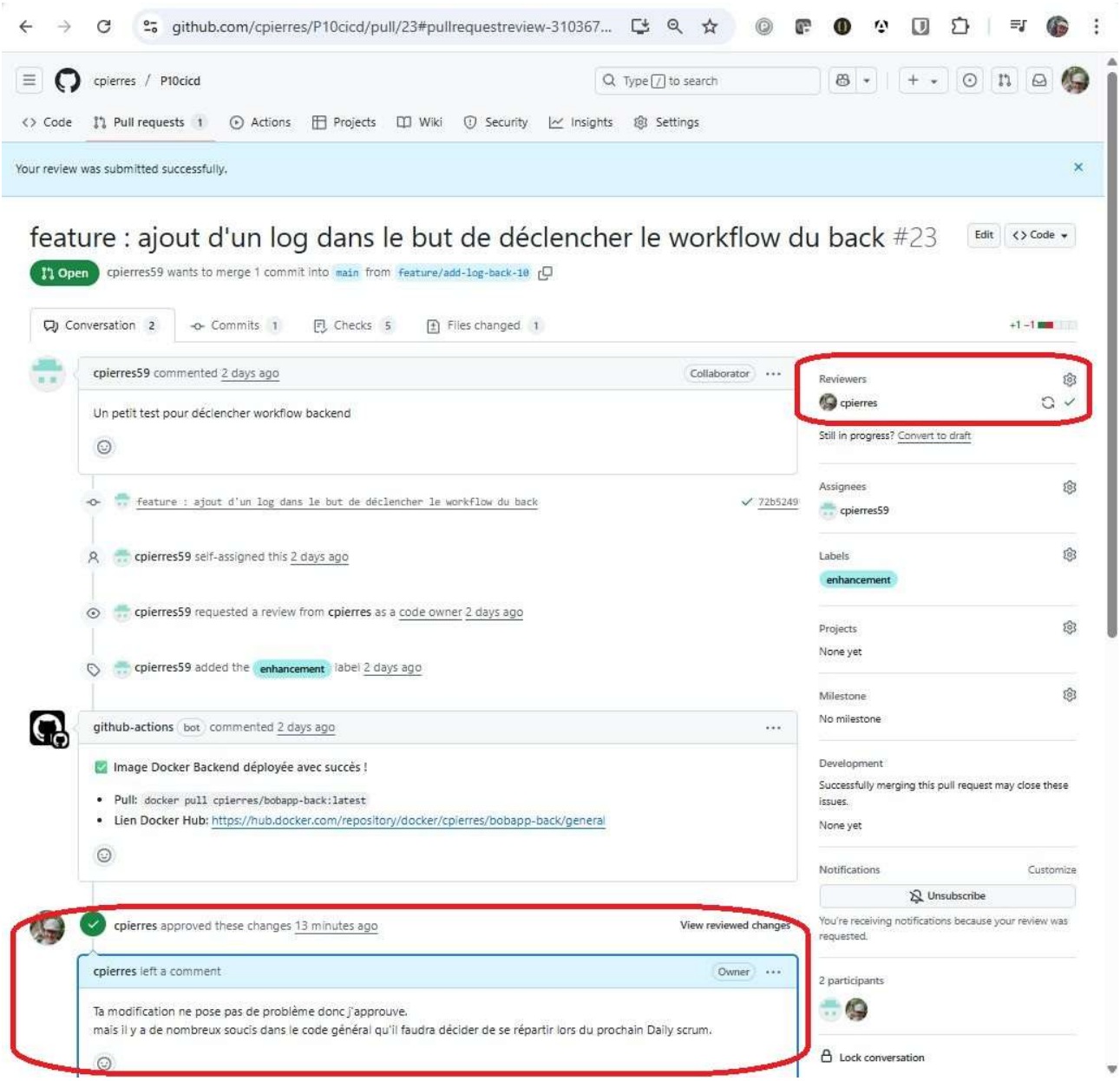
Ta modification ne pose pas de problème donc j'approuve.
mais il y a de nombreux soucis dans le code général qu'il faudra décider de se répartir lors du prochain Daily scrum.

Markdown is supported | Paste, drop, or click to add files

- Comment — Submit general feedback without explicit approval.
- Approve — Submit feedback and approve merging these changes.
- Request changes — Submit feedback that must be addressed before merging.

Submit review

6.1.3. Le collaborateur cpierres59 lit le commentaire d'approbation du reviewer puis merge sa PR



https://github.com/cpierres/P10cicd/pull/23

Open **feature : ajout d'un log dans le but de déclencher le workflow du back** #23
cpierres59 wants to merge 1 commit into `main` from `feature/add-log-back-10`

Image Docker Backend déployée avec succès !

- Pull: `docker pull cpierres/bobapp-back:latest`
- Lien Docker Hub: https://hub.docker.com/repository/docker/cpierres/bobapp-back/general

cpierres approved these changes 19 minutes ago — View reviewed changes

cpierres left a comment — Owner

Ta modification ne pose pas de problème donc j'approuve.
mais il y a de nombreux soucis dans le code général qu'il faudra décider de se répartir lors du prochain Daily scrum.

Commit message

Merge pull request #23 from cpierres/feature/add-log-back-10

Extended description

feature : ajout d'un log dans le but de déclencher le workflow du back

This commit will be authored by 80390224+cpierres59@users.noreply.github.com.

Confirm merge | Cancel

Add a comment

Write | Preview

Add your comment here...

Markdown is supported | Paste, drop, or click to add files

Close pull request | Comment

Development

Successfully merging this pull request may close these issues.

None yet

Notifications — Customize

Unsubscribe

You're receiving notifications because you authored the thread.

2 participants

Lock conversation

Open **feature : ajout d'un log dans le but de déclencher le workflow du back** #23

cpierres59 wants to merge 1 commit into `main` from `feature/add-log-back-10`

✅ Image Docker Backend déployée avec succès !

- Pull: `docker pull cpierres/bobapp-back:latest`
- Lien Docker Hub: https://hub.docker.com/repository/docker/cpierres/bobapp-back/general

cpierres approved these changes 17 minutes ago — View reviewed changes

cpierres left a comment — Owner

Ta modification ne pose pas de problème donc j'approuve.
mais il y a de nombreux soucis dans le code général qu'il faudra décider de se répartir lors du prochain Daily scrum.

Changes reviewed
1 approving review by reviewers with write access.

✓ 1 approval

All checks have passed
5 successful checks

No conflicts with base branch
Merging can be performed automatically.

Merge pull request — You can also merge this with the command line. View command line instructions.

Add a comment

Write | Preview

Add your comment here...

Markdown is supported — Paste, drop, or click to add files

Close pull request — Comment

Development

Successfully merging this pull request may close these issues.

None yet

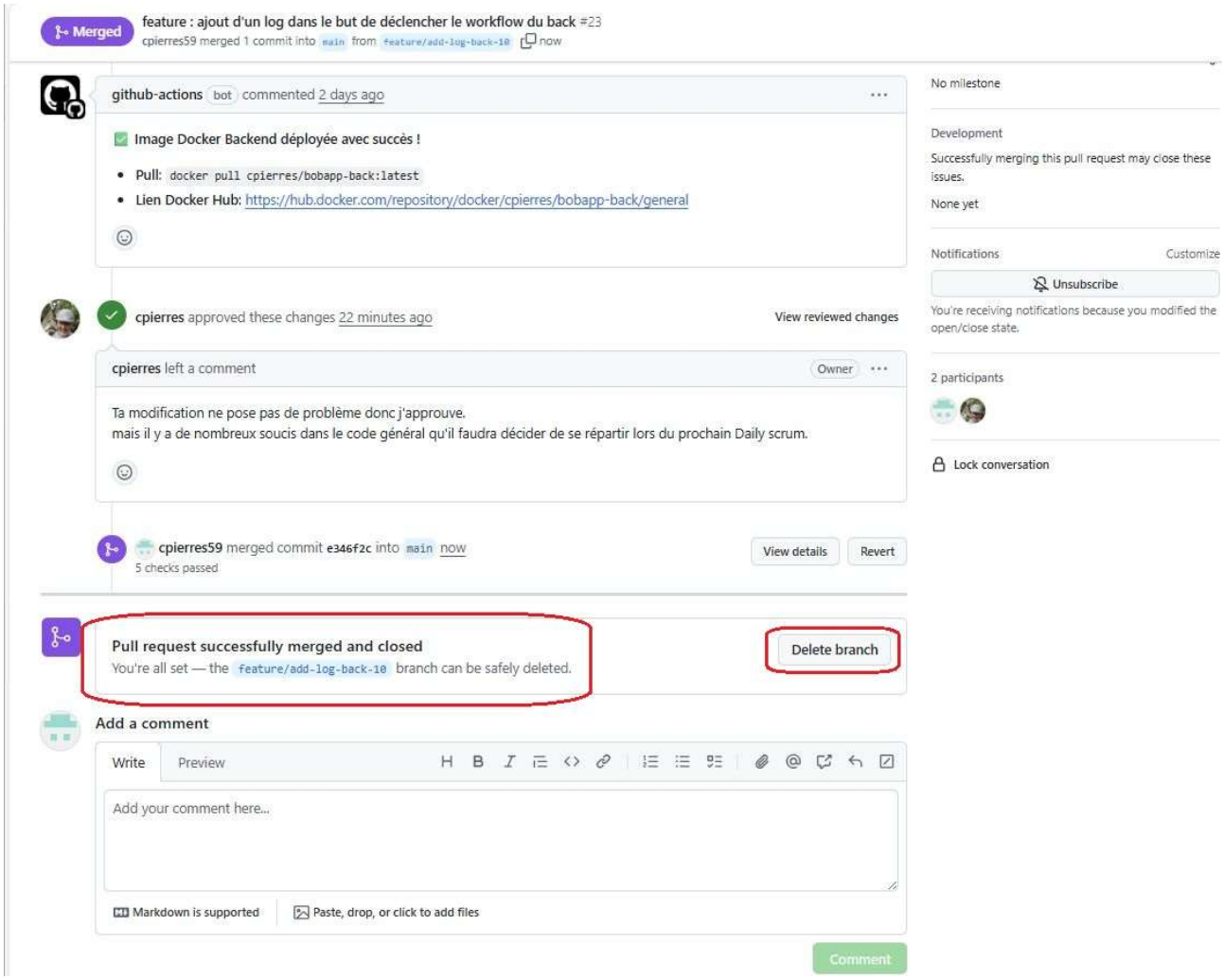
Notifications — Customize

Unsubscribe

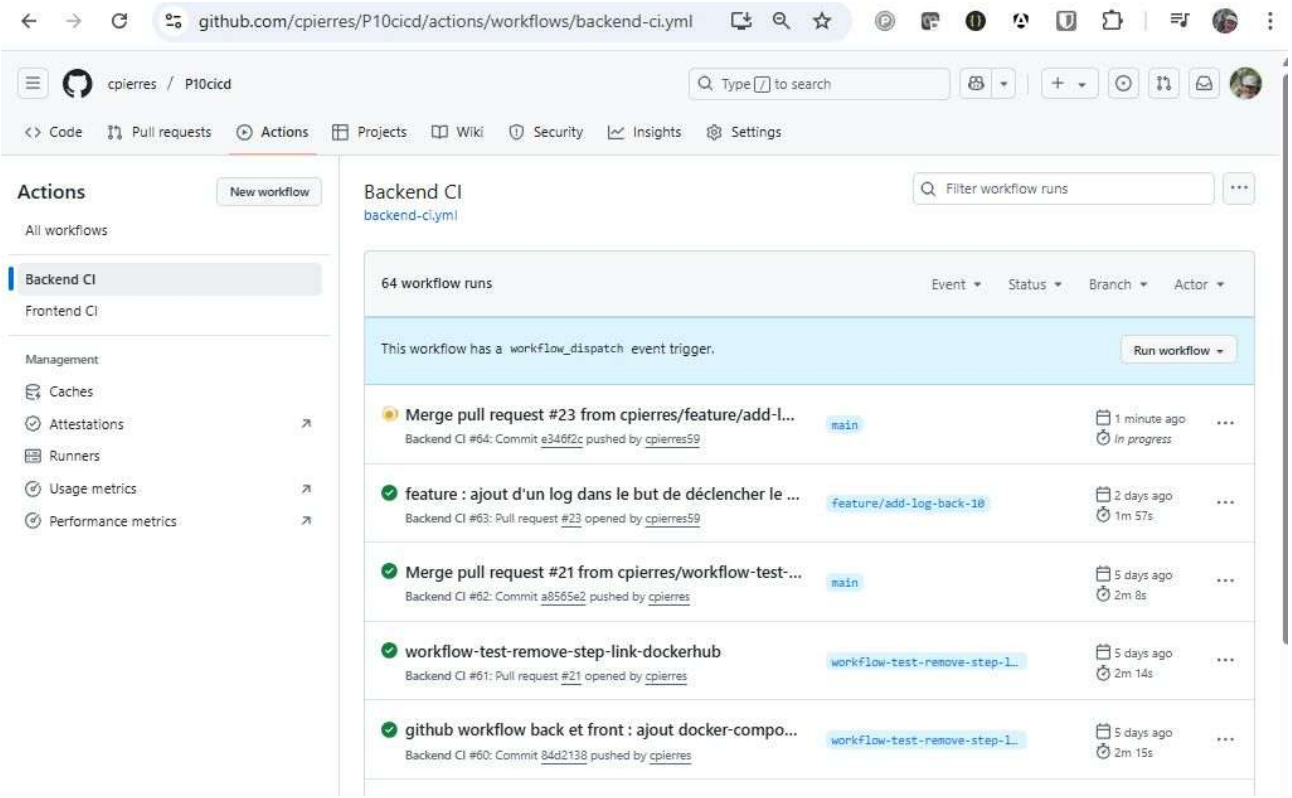
You're receiving notifications because you authored the thread.

2 participants

Lock conversation



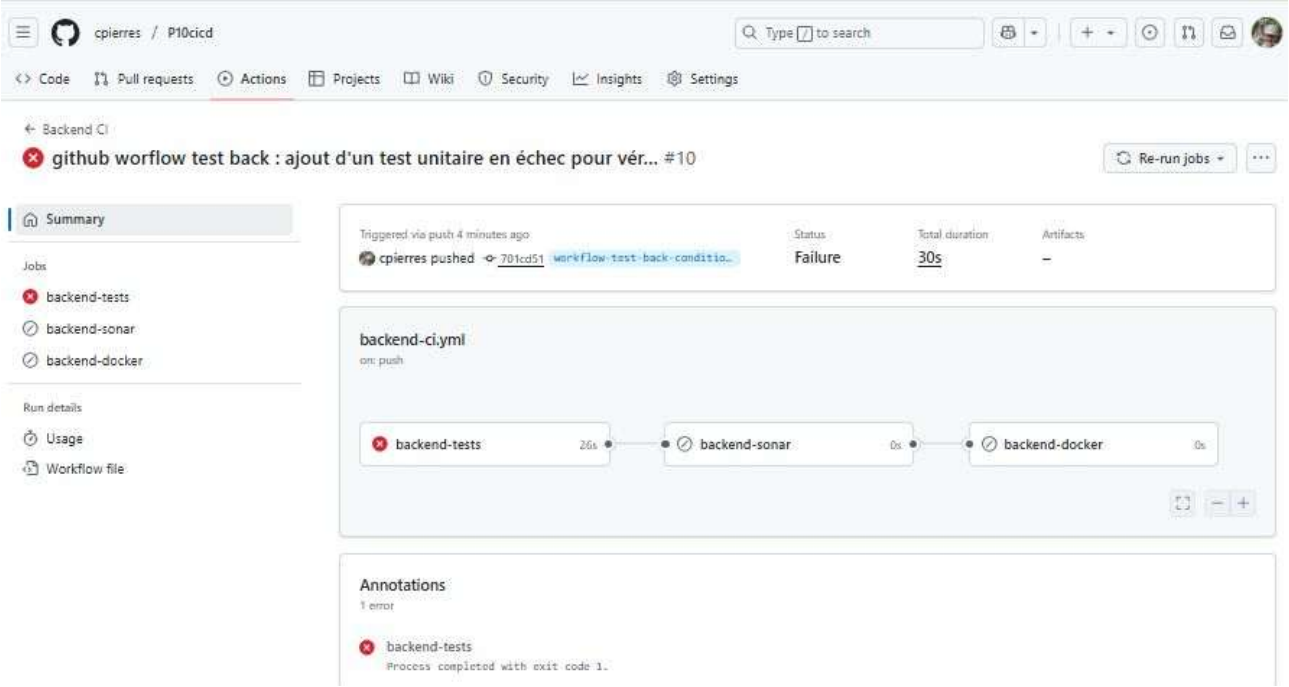
Un dernier cycle de contrôle se produit au moment du merge :



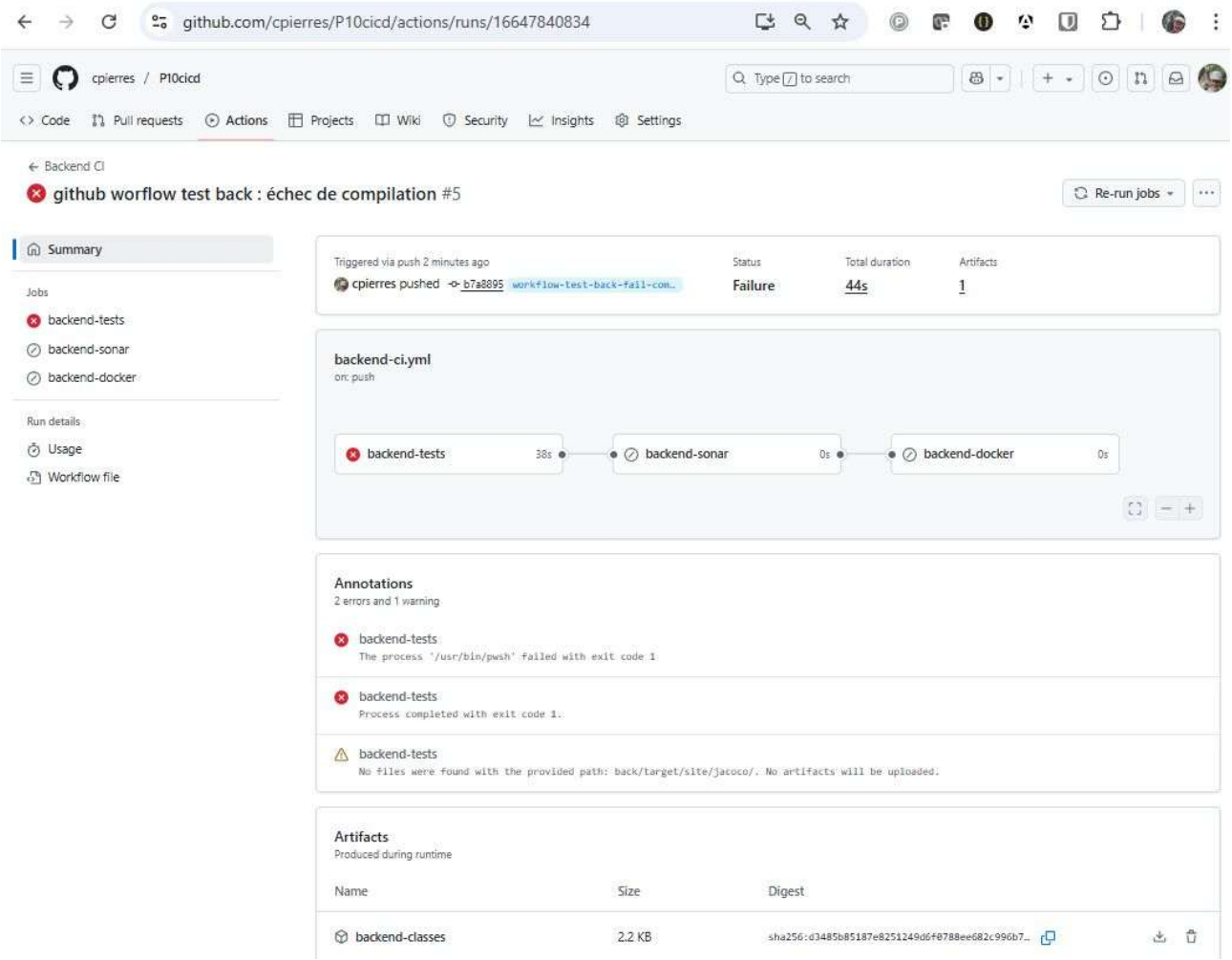
6.1.4. Cas avec erreur lors de la compilation ou dans les tests

Si une erreur se produit lors de la compilation ou les tests du backend, les prochains jobs ne sont pas exécutés.

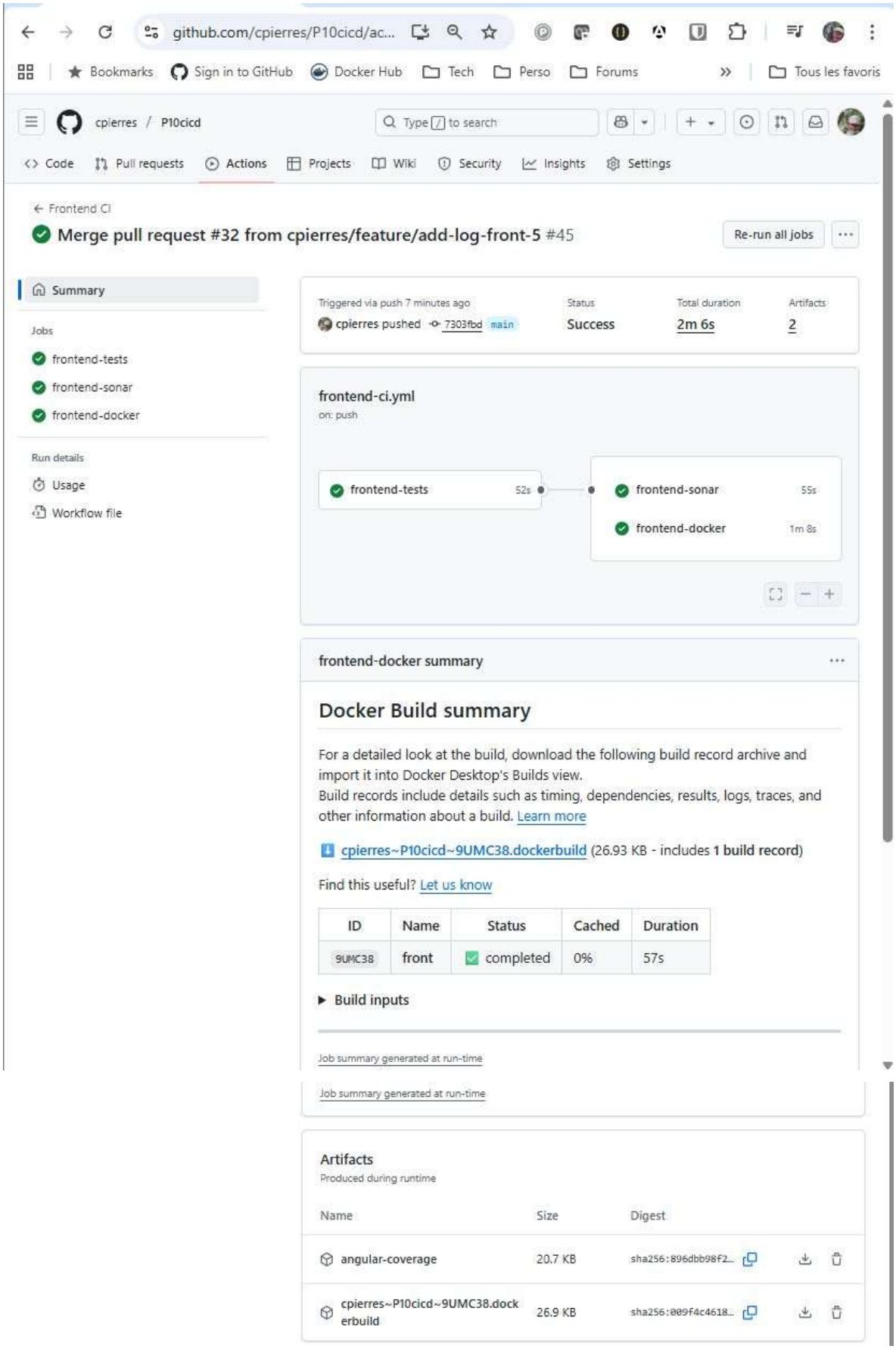
Cas d'une erreur dans les tests :



Cas d'une erreur de compilation :

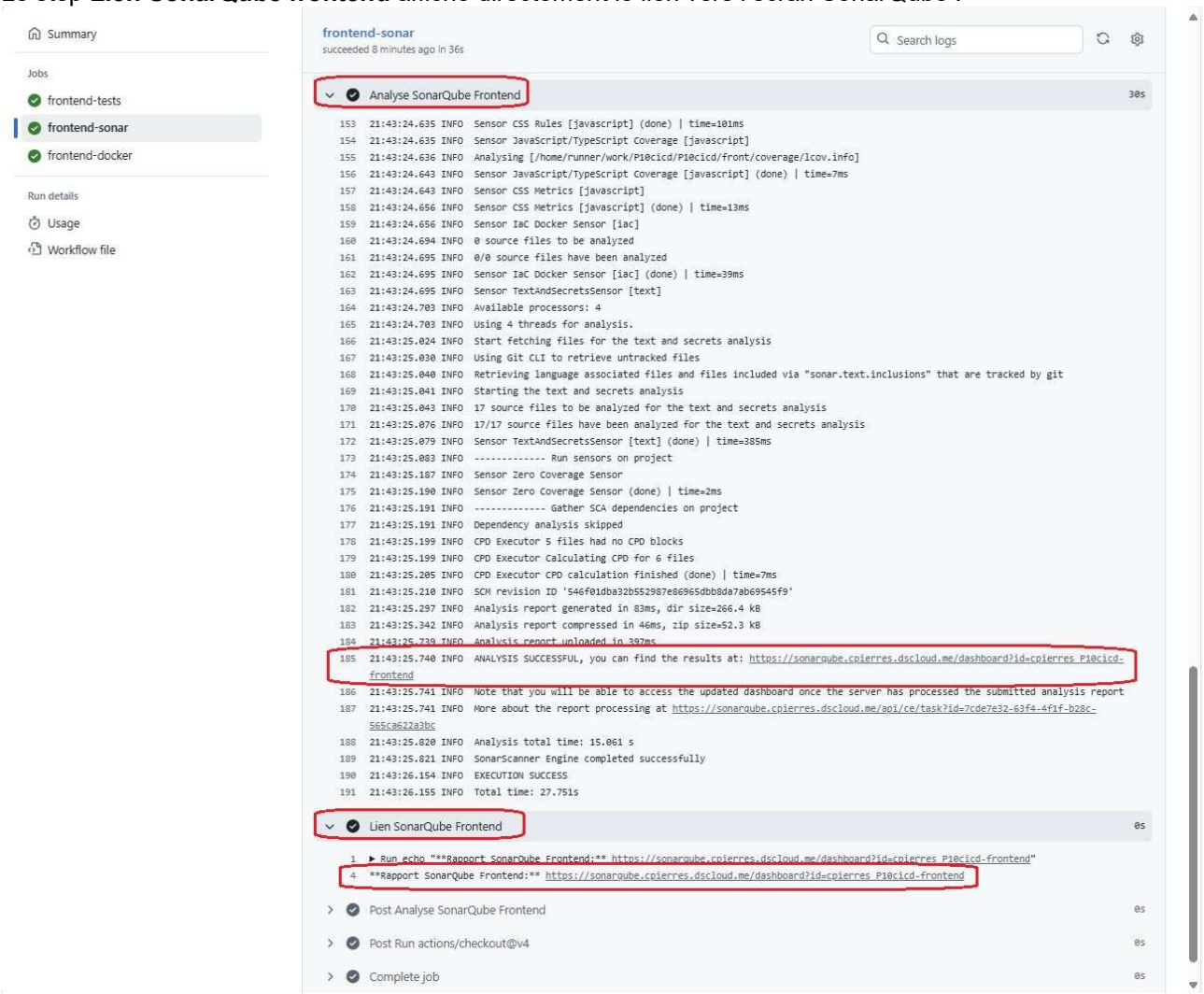


6.1. Test du workflow avec modif uniquement sur le front

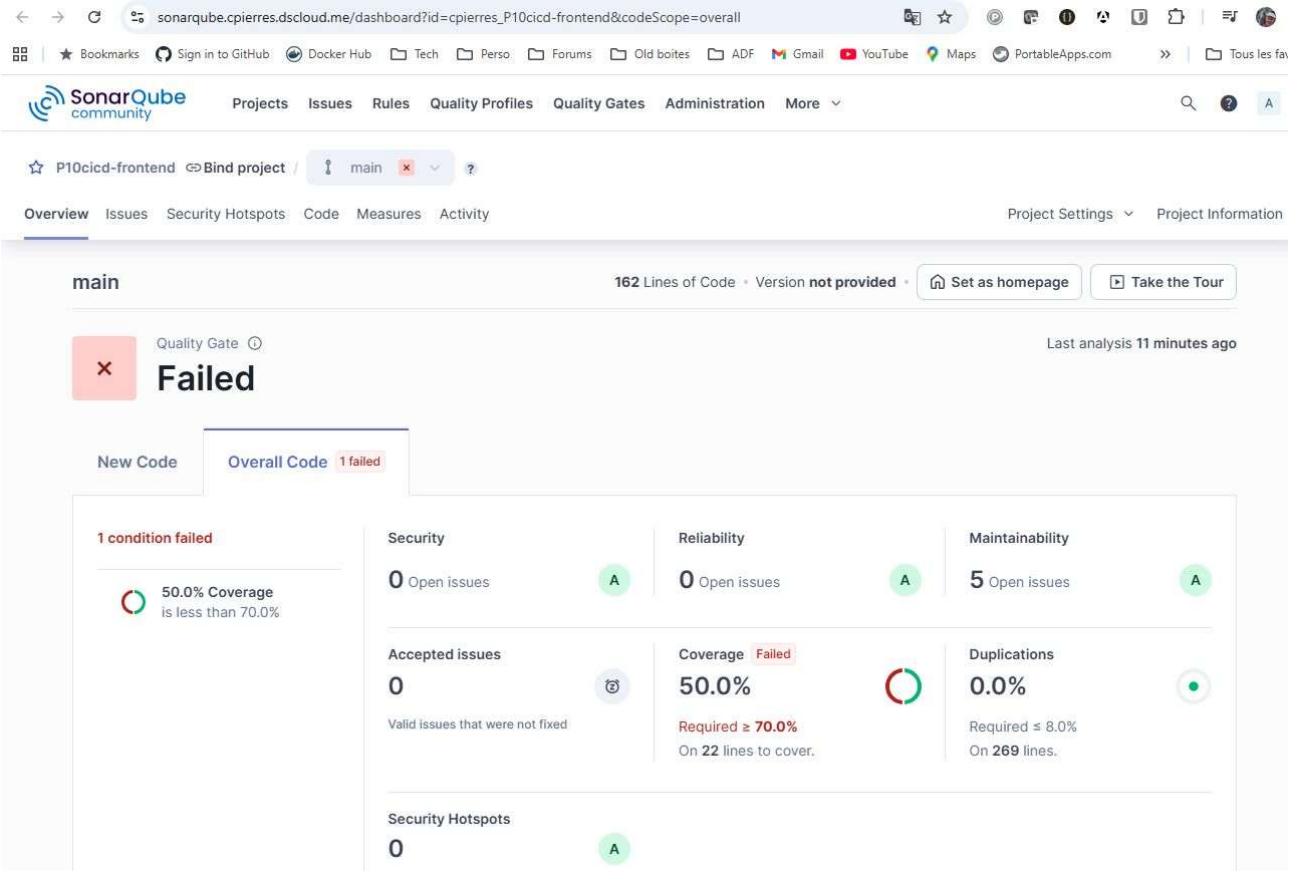

github.com/cpierres/P10cicd/ac...
Bookmarks
Sign in to GitHub
Docker Hub
Tech
Perso
Forums
Tous les favoris
cpierres / P10cicd
Type / to search
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Frontend CI
Merge pull request #32 from cpierres/feature/add-log-front-5 #45
Re-run all jobs
Summary
Jobs
frontend-tests
frontend-sonar
frontend-docker
Run details
Usage
Workflow file
Triggered via push 7 minutes ago
cpierres pushed 7303fbd main
Status
Success
Total duration
2m 6s
Artifacts
2
frontend-ci.yml
on: push
frontend-tests 52s
frontend-sonar 55s
frontend-docker 1m 8s
frontend-docker summary
Docker Build summary
For a detailed look at the build, download the following build record archive and import it into Docker Desktop's Builds view.
Build records include details such as timing, dependencies, results, logs, traces, and other information about a build. Learn more
cpierres~P10cicd~9UMC38.dockerbuild (26.93 KB - includes 1 build record)
Find this useful? Let us know
ID
Name
Status
Cached
Duration
9UMC38
front
completed
0%
57s
Build inputs
Job summary generated at run-time
Job summary generated at run-time
Artifacts
Produced during runtime
Name
Size
Digest
angular-coverage
20.7 KB
sha256:896dbb98F2...
cpierres~P10cicd~9UMC38.dockerbuild
26.9 KB
sha256:009f4c4618...

Le step de **l'Analyse SonarQube frontend** affiche le lien de l'écran SonarQube en bas du log.

Le step **Lien SonarQube frontend** affiche directement le lien vers l'écran SonarQube :



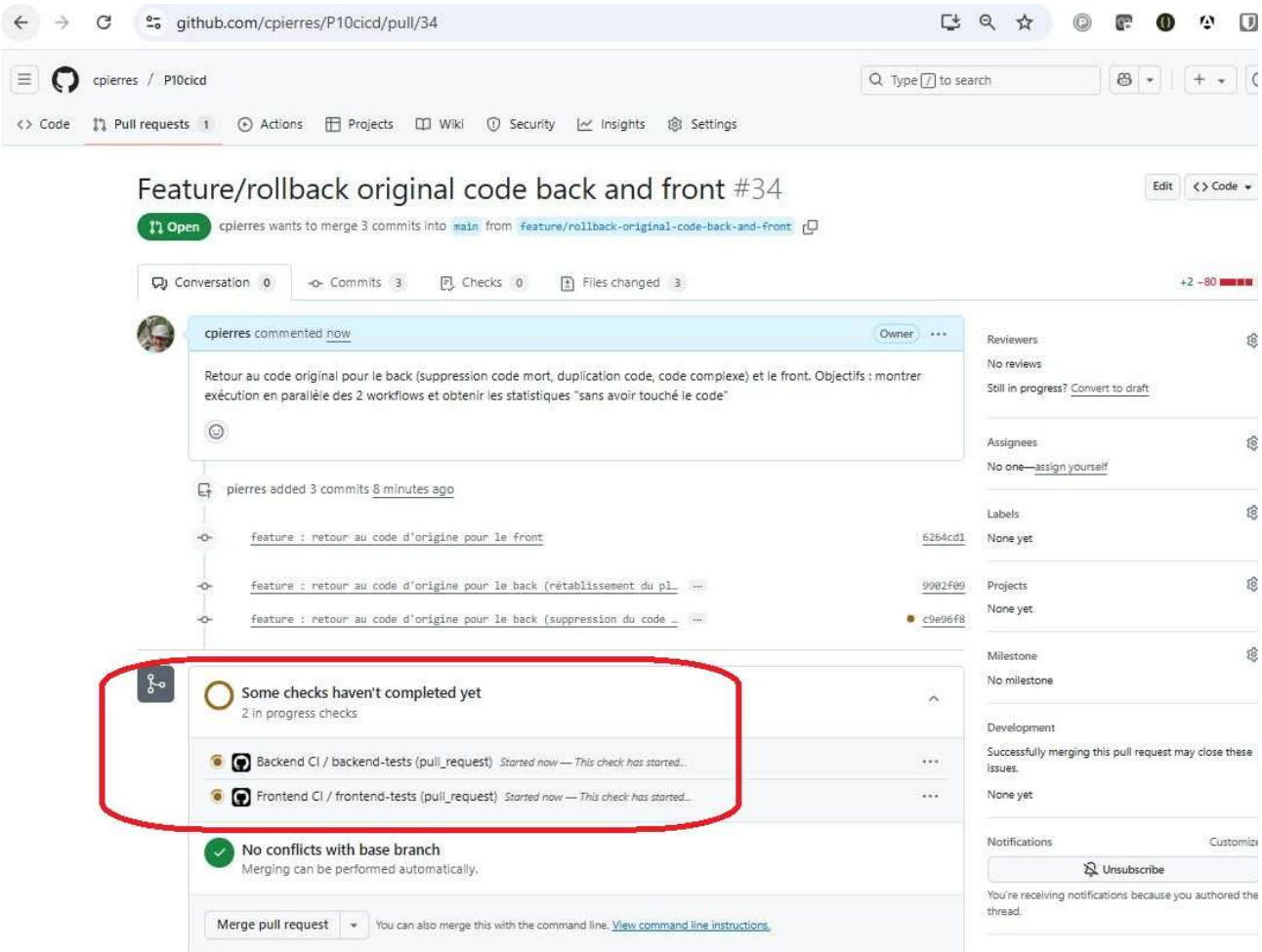
6.1.1. Lien SonarQube Frontend



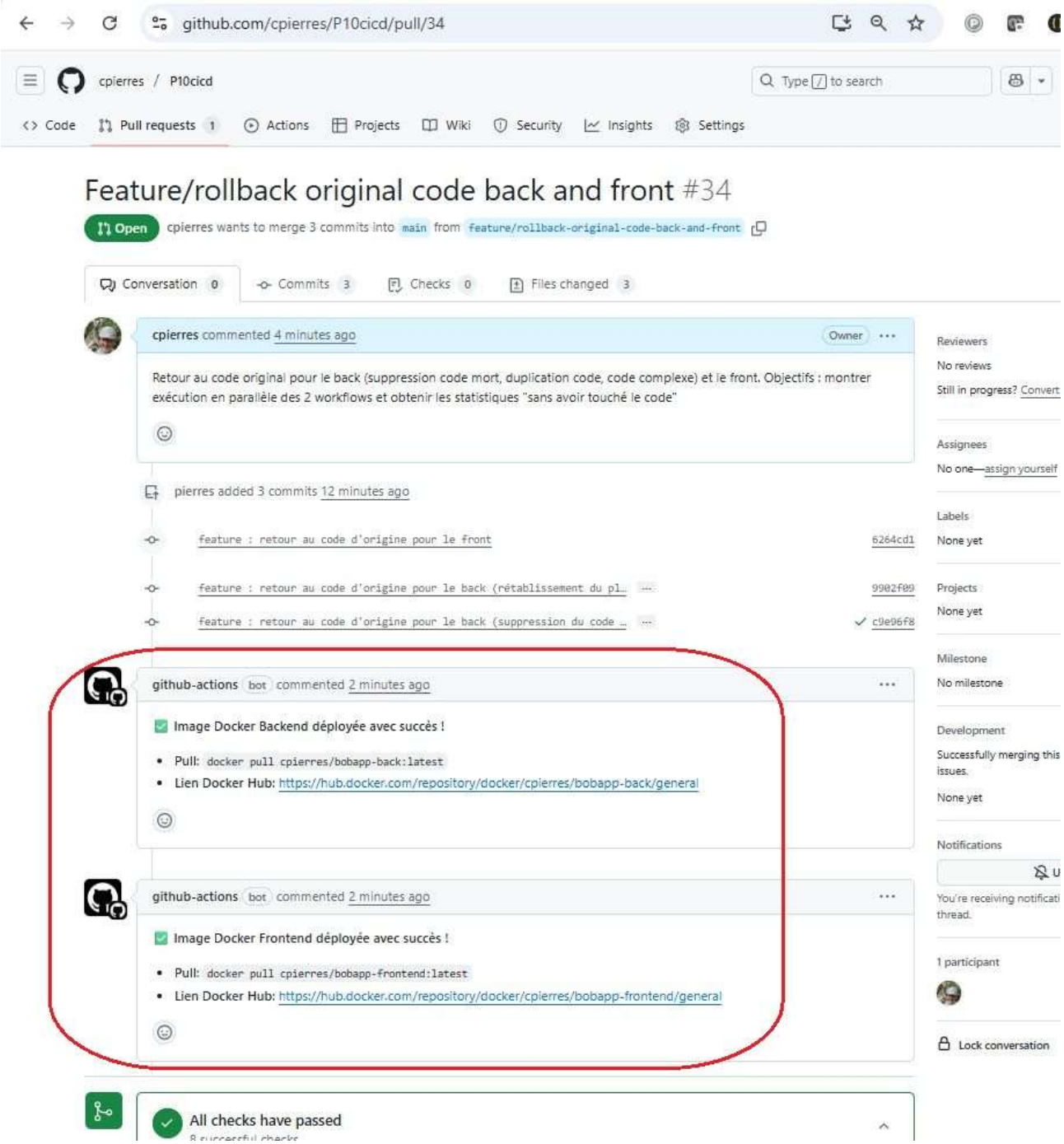
6.2. Test du workflow avec modif à la fois sur le back et le front (remise en place du code d'origine)

La PR suivante montre des commits effectués simultanément dans les deux couches front et back.

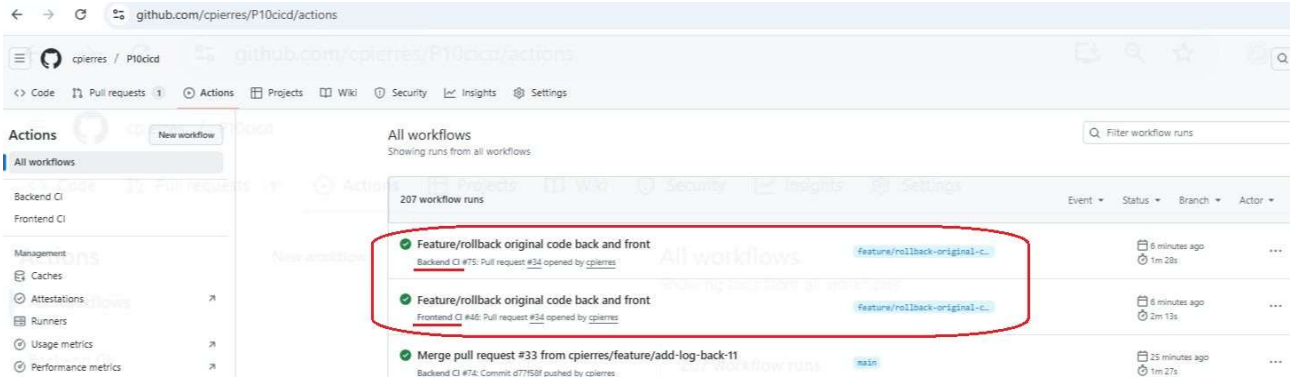
Du coup, les workflows se déclenchent en parallèle :


github.com/cpierres/P10cicd/pull/34
cpierres / P10cicd
Type / to search
Code
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings
Feature/rollback original code back and front #34
Edit
Code
Open
cpierres wants to merge 3 commits into main from feature/rollback-original-code-back-and-front
Conversation 0
Commits 3
Checks 0
Files changed 3
+2 −80
cpierres commented now
Owner
Retour au code original pour le back (suppression code mort, duplication code, code complexe) et le front. Objectifs : montrer exécution en parallèle des 2 workflows et obtenir les statistiques "sans avoir touché le code"
pierres added 3 commits 8 minutes ago
feature : retour au code d'origine pour le front
6264cd1
feature : retour au code d'origine pour le back (rétablissement du pl...
9902f09
feature : retour au code d'origine pour le back (suppression du code ...
c9e96f8
Some checks haven't completed yet
2 in progress checks
Backend CI / backend-tests (pull_request) Started now — This check has started...
Frontend CI / frontend-tests (pull_request) Started now — This check has started...
No conflicts with base branch
Merging can be performed automatically.
Merge pull request
You can also merge this with the command line. View command line instructions.
Reviewers
No reviews
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Unsubscribe
You're receiving notifications because you authored the thread.

6.2.1. Les commentaires de PR pour la génération des images apparaissent bien pour les 2 couches

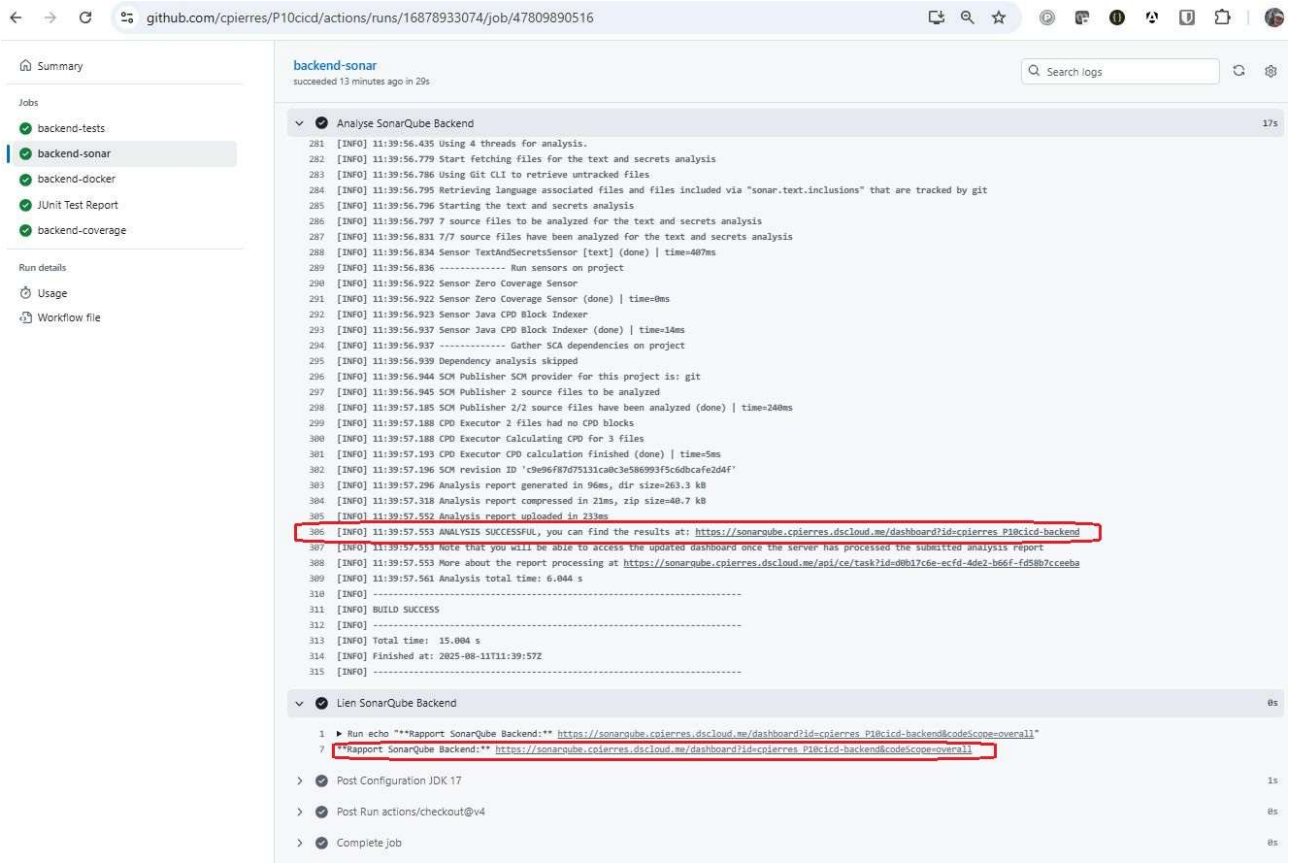


6.2.2. Dans écran Actions, affichage des workflows exécutés parallèlement



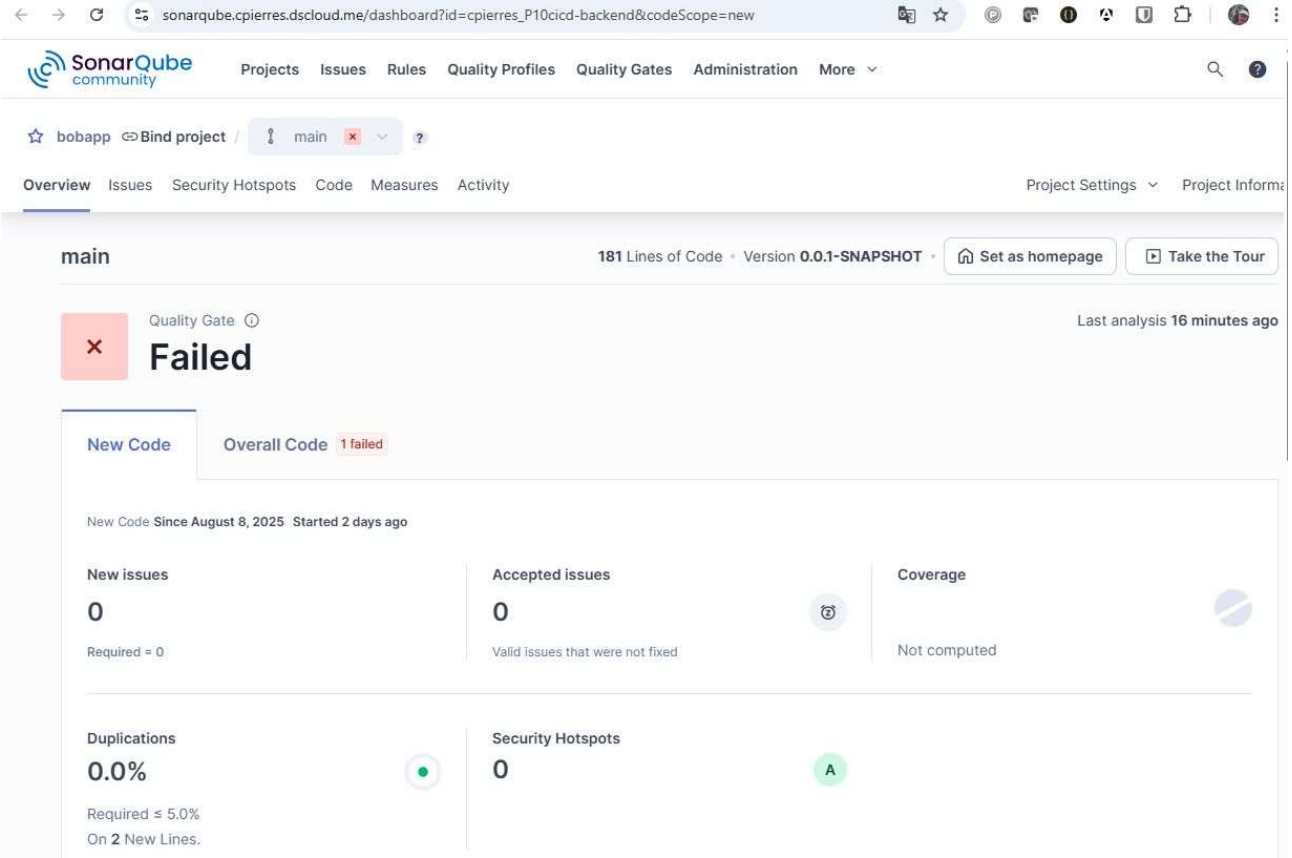
6.2.3. Affichage des rapports sonar du back en cliquant sur les liens affichés dans job backend-sonar

Cliquer sur l'un de ces deux liens :

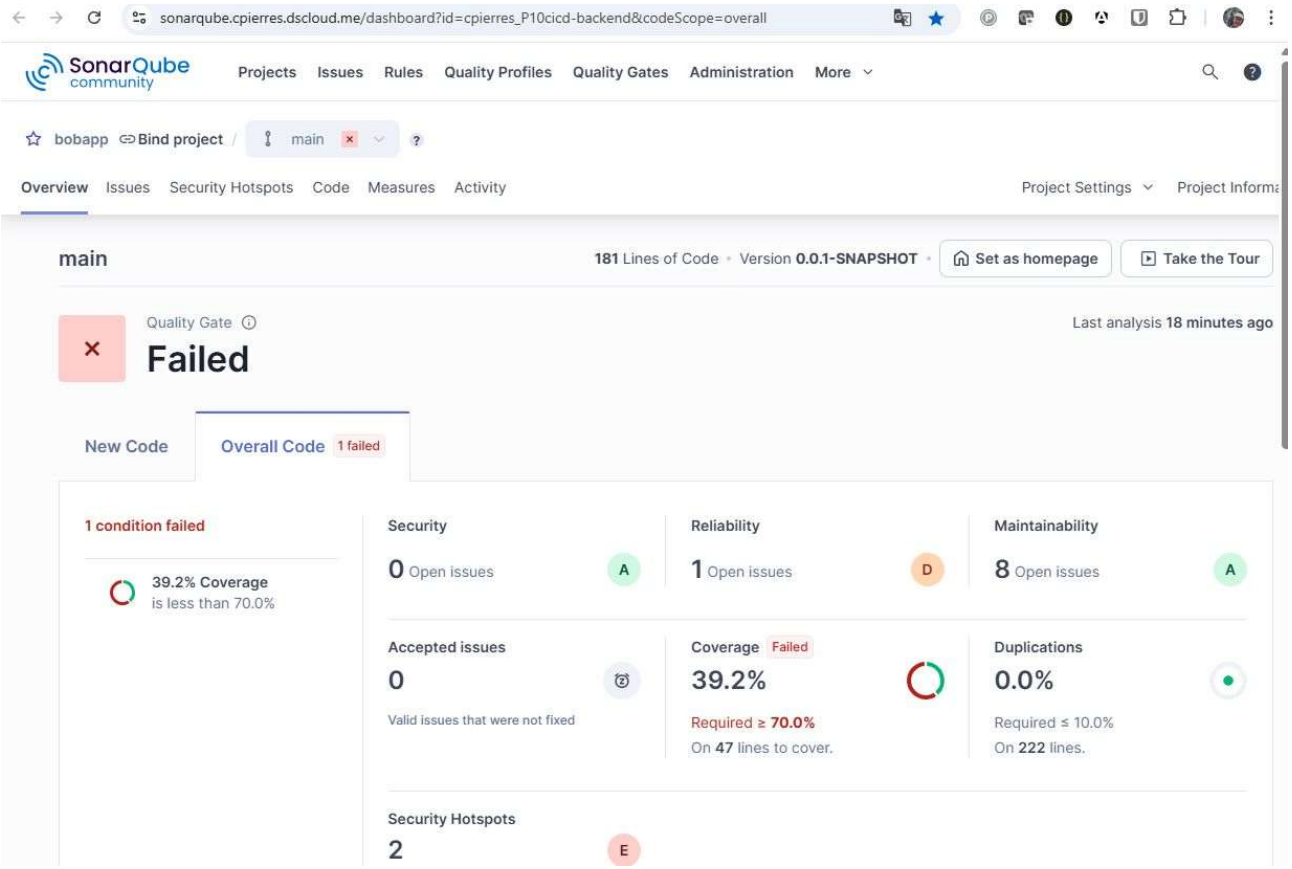


6.2.3.1. Rapport du backend sur New code

Aucun problème sur le nouveau code commité (seul l'overall est failed) :



6.2.3.2. Rapport du backend sur Overall code



6.2.3.3. Interprétation des résultats

L'état "Overall" actuel met en évidence trois leviers à haut rendement:

(1) corriger l'unique bug de fiabilité,

(2) revoir les 2 hotspots de sécurité,

(3) ajouter quelques tests unitaires pour dépasser 70% de couverture.

Ces actions rapides permettent de viser des ratings A/A/A et un coverage conforme.

Les Quality Gates restent strictes sur le New Code (Clean as You Code) pour éviter toute régression de qualité; pour l'Overall, nous choisissons soit de maintenir 70% immédiatement (périmètre réduit à couvrir), soit de placer un WARNING provisoire à 50% et de planifier un sprint d'augmentation de la couverture.

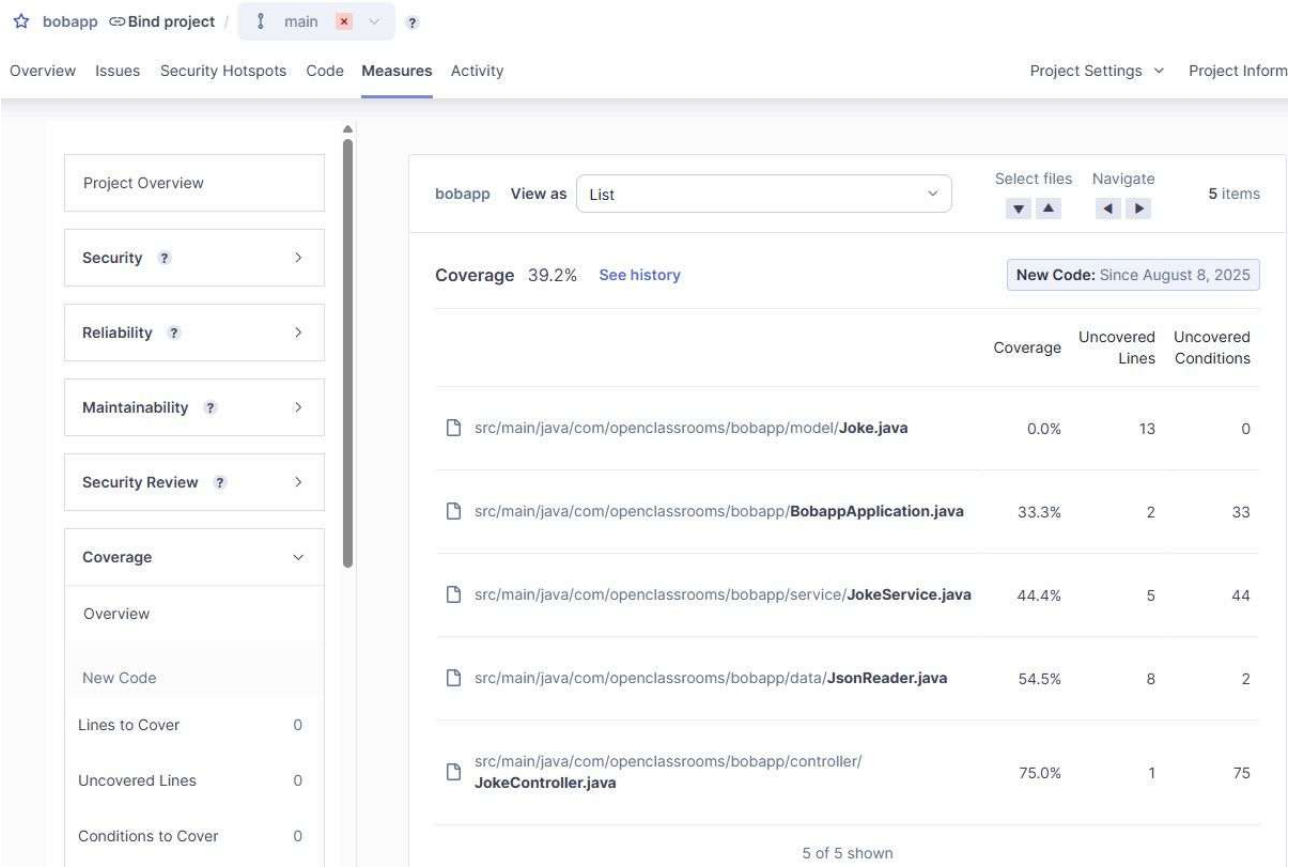
Explication de chaque point

Coverage 39,2% (Failed; requis ≥ 70%)

Part des lignes exécutées par les tests automatisés.

- Interprétation: sur 47 lignes "à couvrir", environ 18–19 sont couvertes et 28–29 ne le sont pas. Avec un petit périmètre, quelques tests ciblés peuvent faire rapidement passer le seuil.

- Impact: faible couverture = risque de régressions non détectées, confiance limitée lors des refactorings.

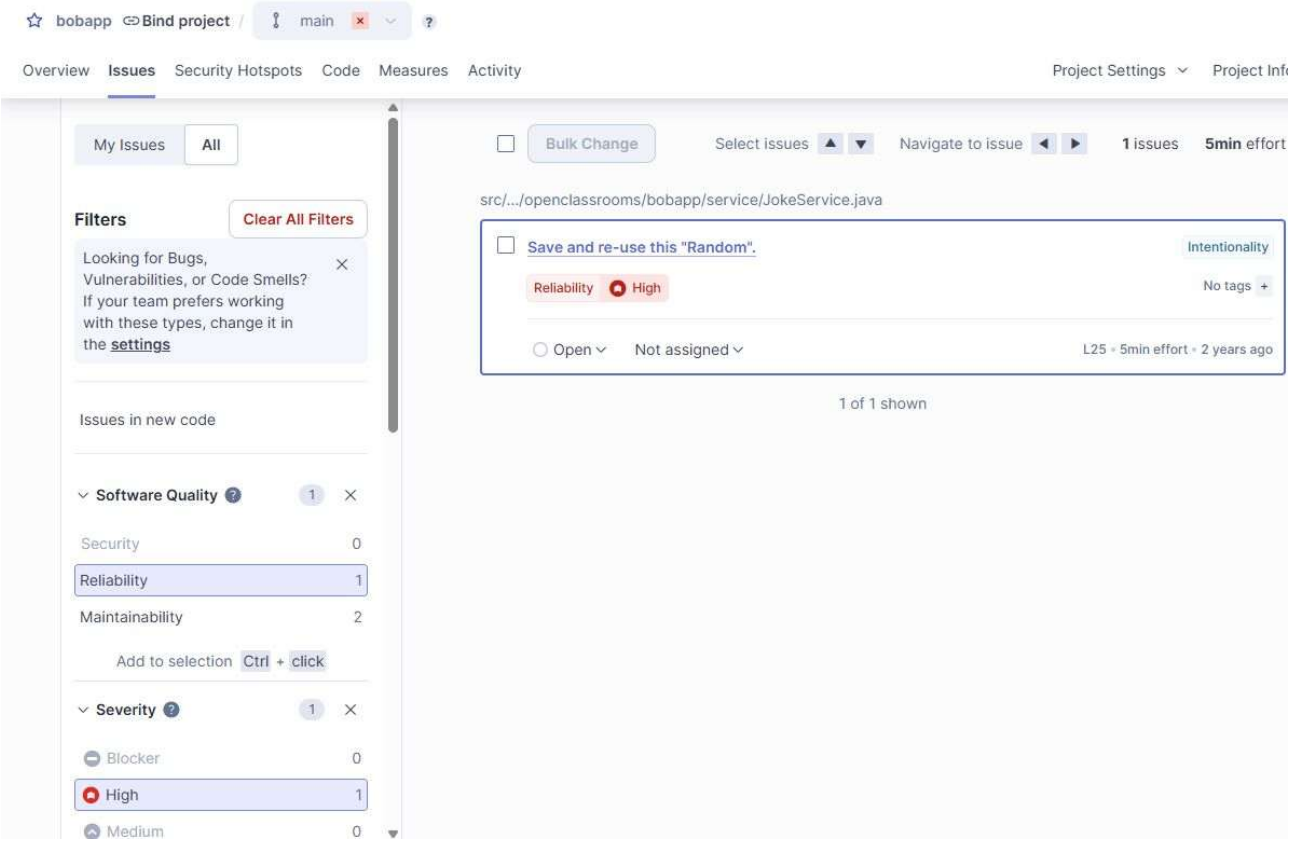


Reliability: 1 Open issue - Rating D

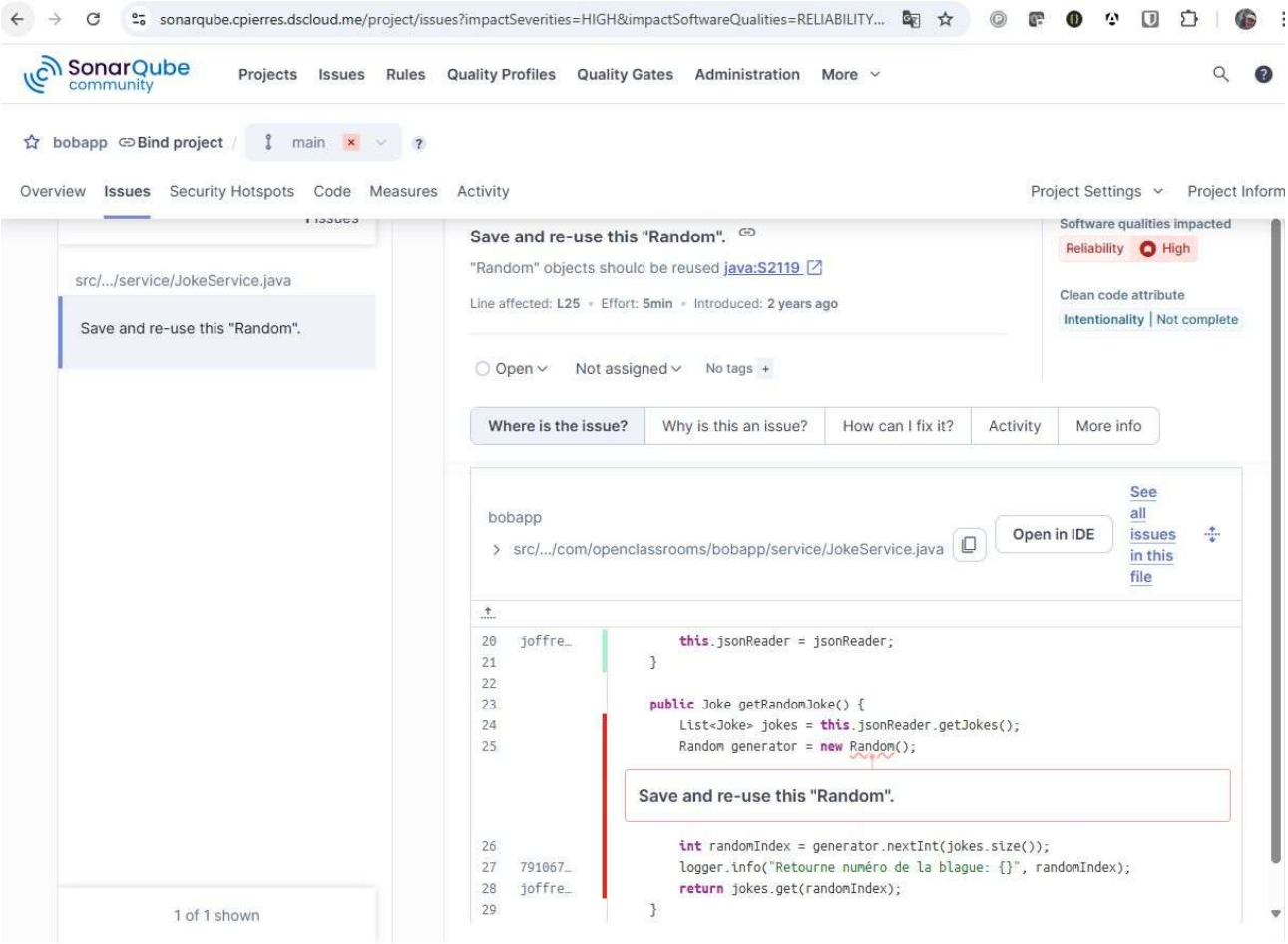
La fiabilité est dégradée par la présence de bugs (issues de type "Bug"). La notation (A → E) est principalement dictée par la gravité du bug le plus sérieux.

- Interprétation: un unique bug, d'une gravité suffisante, tire la note à D. Une correction unique peut suffire à remonter à A.

Cliquer sur le chiffre 1 pour voir l'issue :



Le problème est lié à l'utilisation de random :

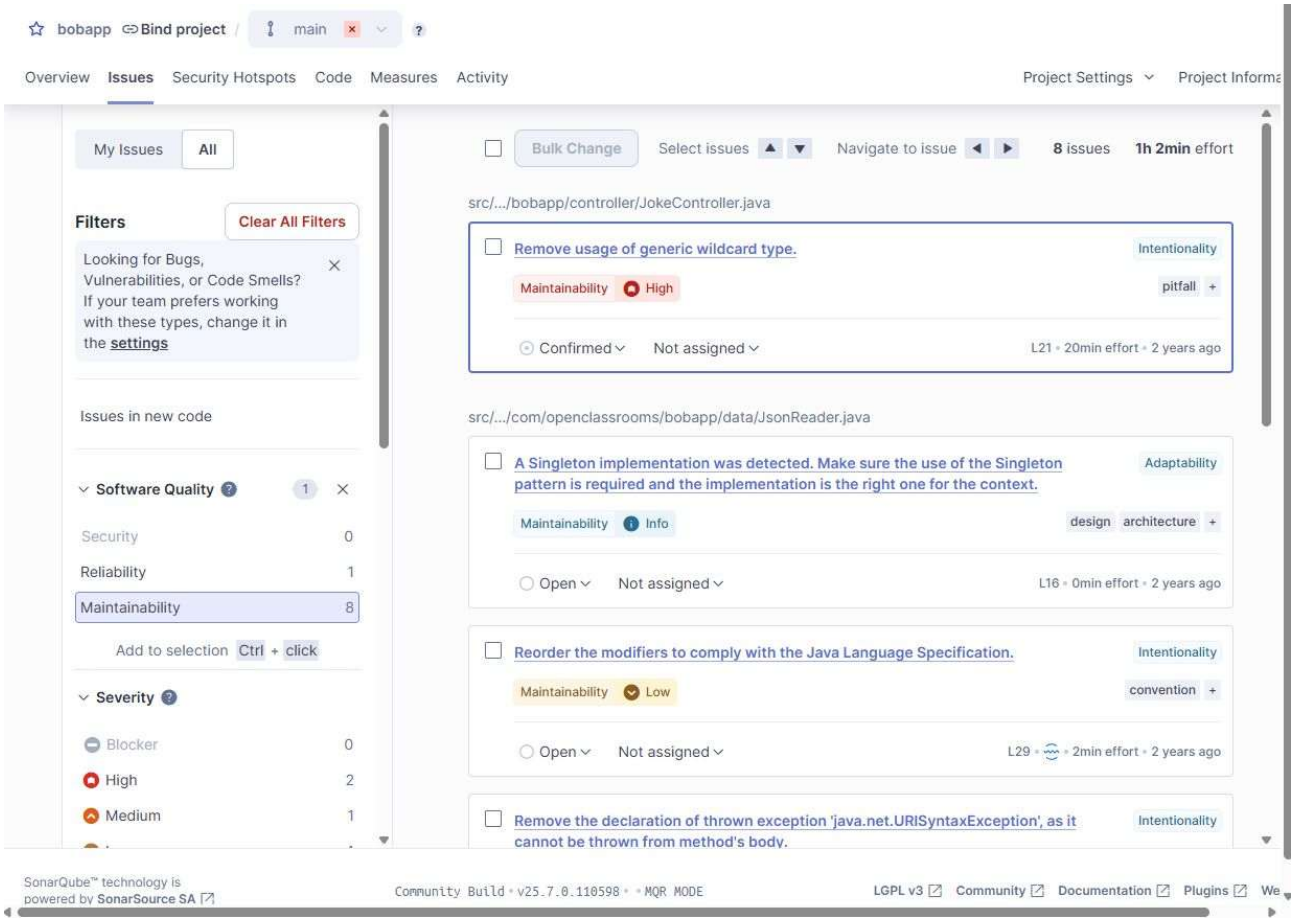


Maintainability: 8 Open issues - Rating A

La maintenabilité mesure la “dette technique” (code smells × effort de remediation) rapportée à la taille du code.

- Interprétation: malgré 8 issues, le ratio de dette < 5% ⇒ Rating A. Ces issues sont donc de faible impact relatif.

Les voici :



Security Hotspots: 2 — Rating E

Points de vigilance sécurité nécessitant une revue humaine (ce ne sont pas des vulnérabilités avérées).

- Interprétation: non revus, ils dégradent fortement la note (E). Une revue rapide (et correction si nécessaire) peut faire remonter la note à A.

- Exemples fréquents: logs sensibles, choix crypto, configuration d’en-têtes de sécurité.

Voici les deux problèmes relevés dans bobApp :

bobapp Bind project / main

Overview Issues Security Hotspots Code Measures Activity Project Settings Project Inf

0.0% Security Hotspots Reviewed

To review | Acknowledged | Fixed | Safe

2 Security Hotspots to review

Review priority: Medium

Weak Cryptography 1

Make sure that using this pseudorandom number generator is safe here.

Review priority: Low

Insecure Configuration 1

Make sure this debug feature is deactivated before delivering the code in production.

2 of 2 shown

sensitive java:S4507

Insecure Configuration

Assignee: Not assigned

Status: To review

This security hotspot needs to be reviewed to assess whether the code poses a risk.

Review

Where is the risk? | What's the risk? | Assess the risk | How can I fix it? | Activity

src/.../com/openclassrooms/bobapp/data/JsonReader.java Open in IDE

```
14
15      @Repository
16      public class JsonReader {
17          private final ObjectMapper mapper = new ObjectMapper();
18          private JsonNode jsonFile;
19
20          private JsonReader() {
21              try {
22                  this.getJsonFile();
23              } catch (IOException | URISyntaxException e) {
24                  e.printStackTrace();
```

Make sure this debug feature is deactivated before delivering the code in production.

```
25              }
26          }
27
28          private static class SingletonHolder {
29              private final static JsonReader instance = new JsonReader();
30          }
```

Priorisation des actions

1) Corriger la Reliability (D → A)

- Ouvrir l'onglet Issues (type "Bug"), corriger l'unique bug
- Commit/push puis relancer l'analyse ⇒ la note peut rebasculer à A immédiatement.

2) Revoir les 2 Security Hotspots (E → A)

- Ouvrir les Hotspots, examiner le code.
- Si le code est sûr, marquer "Reviewed" avec justification; sinon corriger puis reviewer.
- Objectif: "Security Hotspots Reviewed = 100%".

3) Remonter la couverture vers 70%

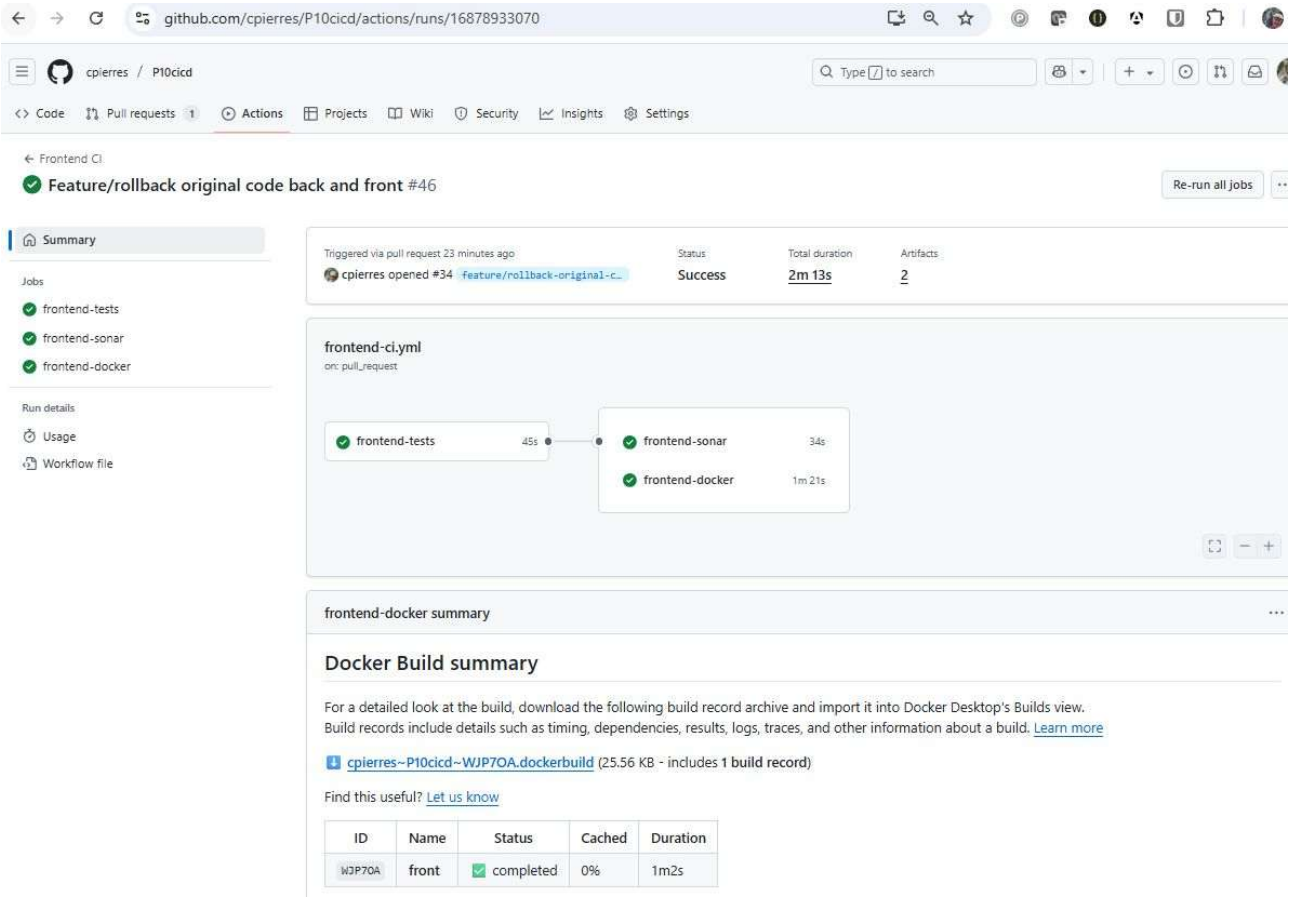
- Ajouter des tests unitaires ciblés sur les chemins non couverts et branches conditionnelles.
- Cibles typiques: service métier principal (cas nominal + cas limites), contrôleur REST (200/404), parsing/chargement de données (fichier JSON embarqué), gestion d'erreurs.
- 10–15 lignes supplémentaires couvertes peuvent suffire à dépasser 70%.

4) Maintenabilité: trier les 8 code smells; corriger 2 ou 3 à fort levier.

- Tester les chemins d'erreurs et scénarios négatifs pour améliorer la robustesse.

6.2.1. Affichage des rapports sonar du back en cliquant sur les liens affichés dans job backend-sonar

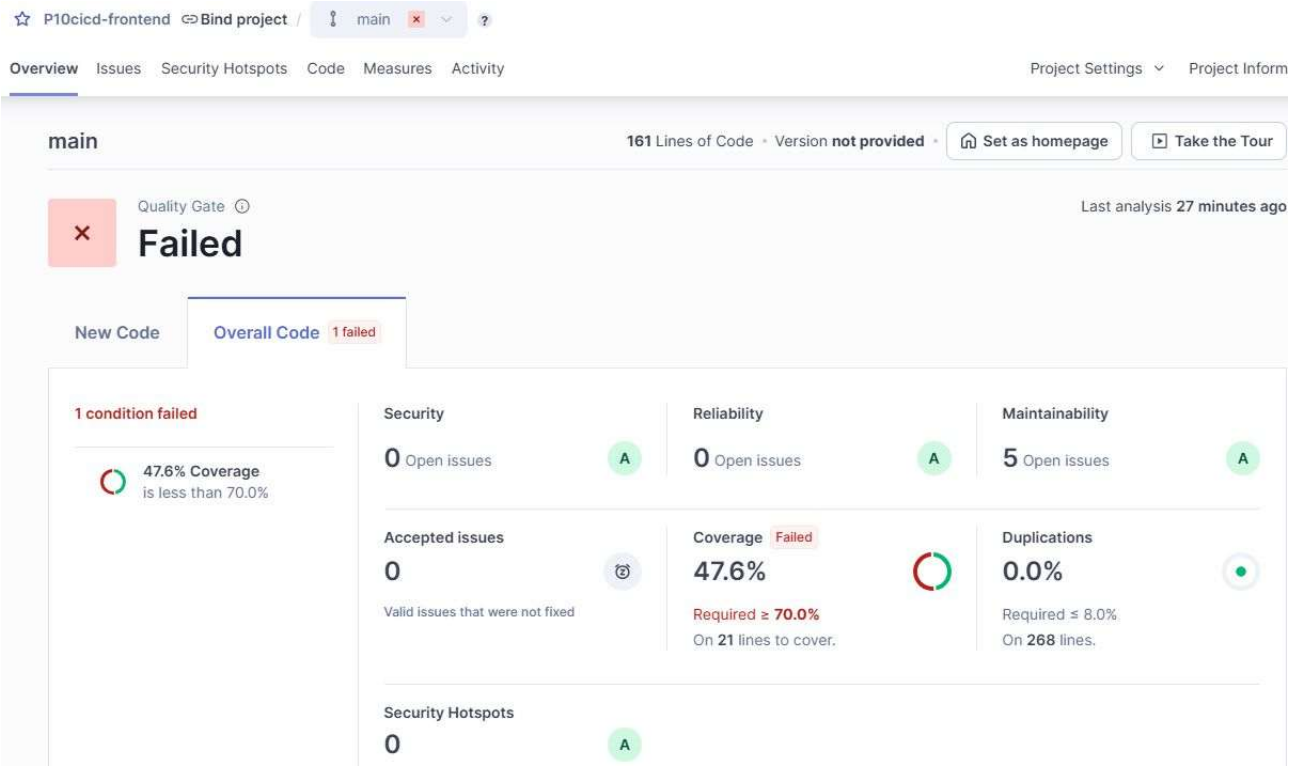
Aller sur l'exécution de workflow du front et faire la même opération que pour le back afin d'afficher les rapports Sonar (cliquer sur le job sonar pour voir ses steps) :



6.2.1.1. Rapport du frontend sur New code

Aucun problème signalé.

6.2.1.2. Rapport du frontend sur Overall code



7. Analyse des retours utilisateurs

1. **Avis utilisateur 1**: « Impossible de poster une suggestion de blague, le bouton tourne et fait planter mon navigateur »
   - **Problème**: La fonctionnalité de soumission de blagues provoque des blocages du navigateur.
   - **Priorité**: Élevée - Il s'agit d'une fonctionnalité critique orientée utilisateur.
   - **Branche** : front-fix-post-joke

2. **Avis utilisateur 2**: « Bug sur la publication de vidéo toujours présent après deux semaines »
   - **Problème**: Bug non résolu avec la fonctionnalité de publication de vidéo.
   - **Priorité**: Moyenne - La fonctionnalité ne fonctionne pas mais ne provoque pas de plantages système.

3. **Avis utilisateur 3**: « Ne reçoit pas de contenu depuis une semaine, pas de réponse à l'email »
   - **Problème**: Perturbation potentielle du service backend et support client médiocre.
   - **Priorité**: Élevée - Problème de disponibilité du service.

4. **Avis utilisateur 4**: « Site retiré des favoris »
   - **Problème**: Insatisfaction générale conduisant à l'abandon des utilisateurs.
   - **Priorité**: Moyenne - Indique des problèmes plus larges avec l'expérience utilisateur.

7.1. Actions prioritaires

1. **Corriger la fonctionnalité de soumission de blagues (avis 1)**: Cela semble provoquer des plantages de navigateur et est mentionné dans plusieurs plaintes d'utilisateurs.

2. **Mettre en place des tests complets** : Configurer des tests automatisés pour tous les flux utilisateurs critiques afin de détecter les problèmes avant qu'ils n'atteignent la production.

3. **Améliorer la gestion des erreurs**: De nombreux problèmes semblent liés à une mauvaise gestion des erreurs, provoquant le gel de l'application ou des échecs silencieux.

4. **Mettre en place une surveillance**: Implémenter une surveillance de l'application pour détecter les perturbations de service avant que les utilisateurs ne les signalent.

5. **Établir une boucle de rétroaction**: Créer un système de notification pour suivre et répondre rapidement aux problèmes des utilisateurs. Informer du changelog du dépôt github pour suivre les fix et features.

Lire la section « Boucle de rétroaction avec les utilisateurs de bobapp » à la page 32 (reste à faire).

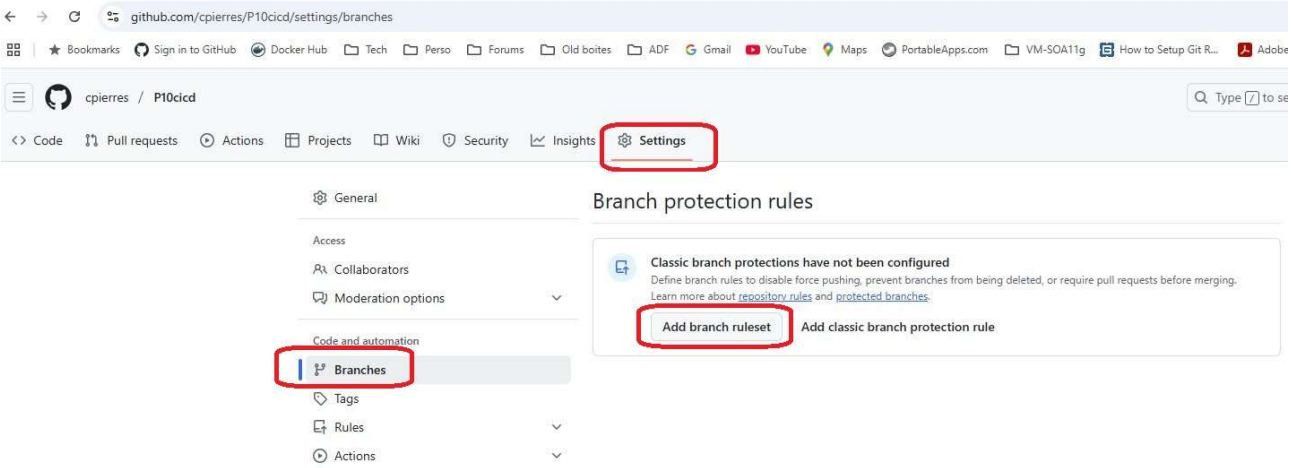
8. Annexes

8.1. Protections du dépôt (copies d'écran)

8.1.1. Protéger la branche main en obligeant les développeurs à passer par des Pull Requests

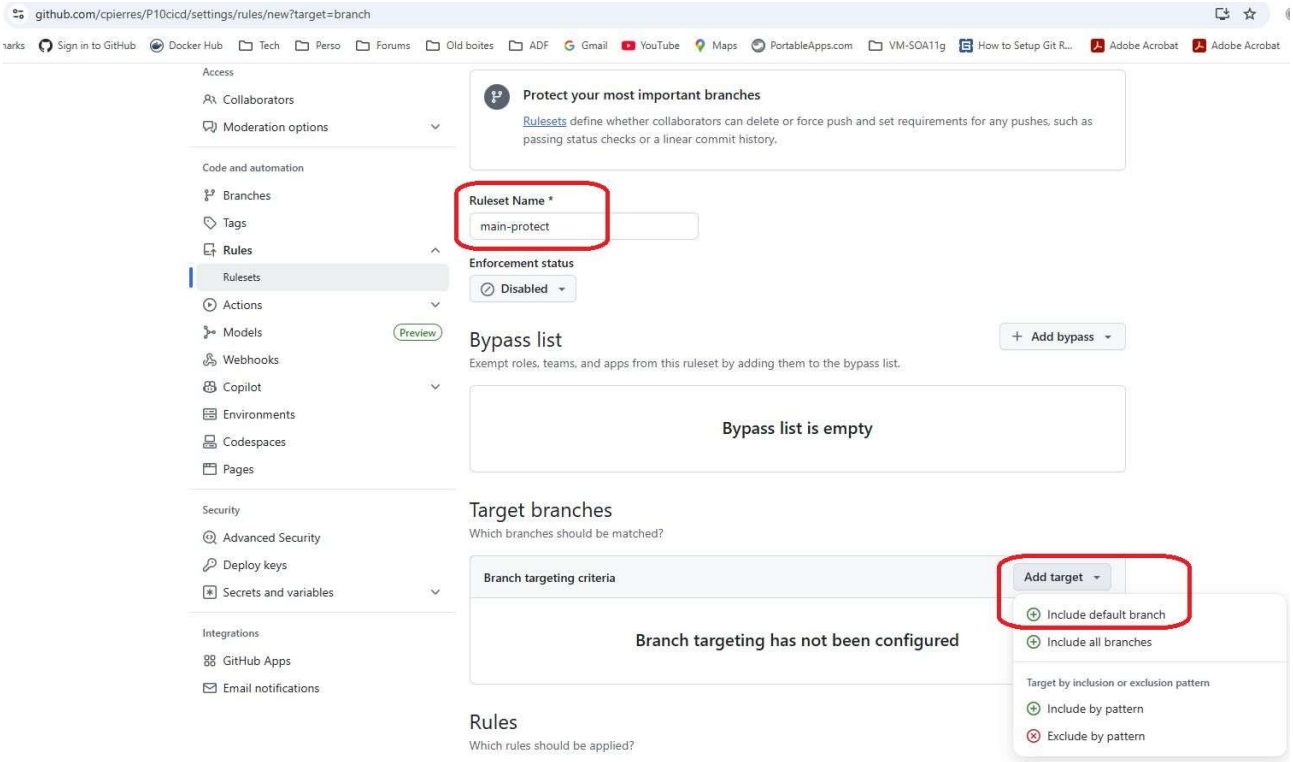
8.1.1.1. Définir une ruleset de branche

Seul le propriétaire (ou les collaborateurs habilités) pourront effectuer le merge vers la branche par défaut.



Ruleset Name : `main-protect`

Target branches : default (`main`)

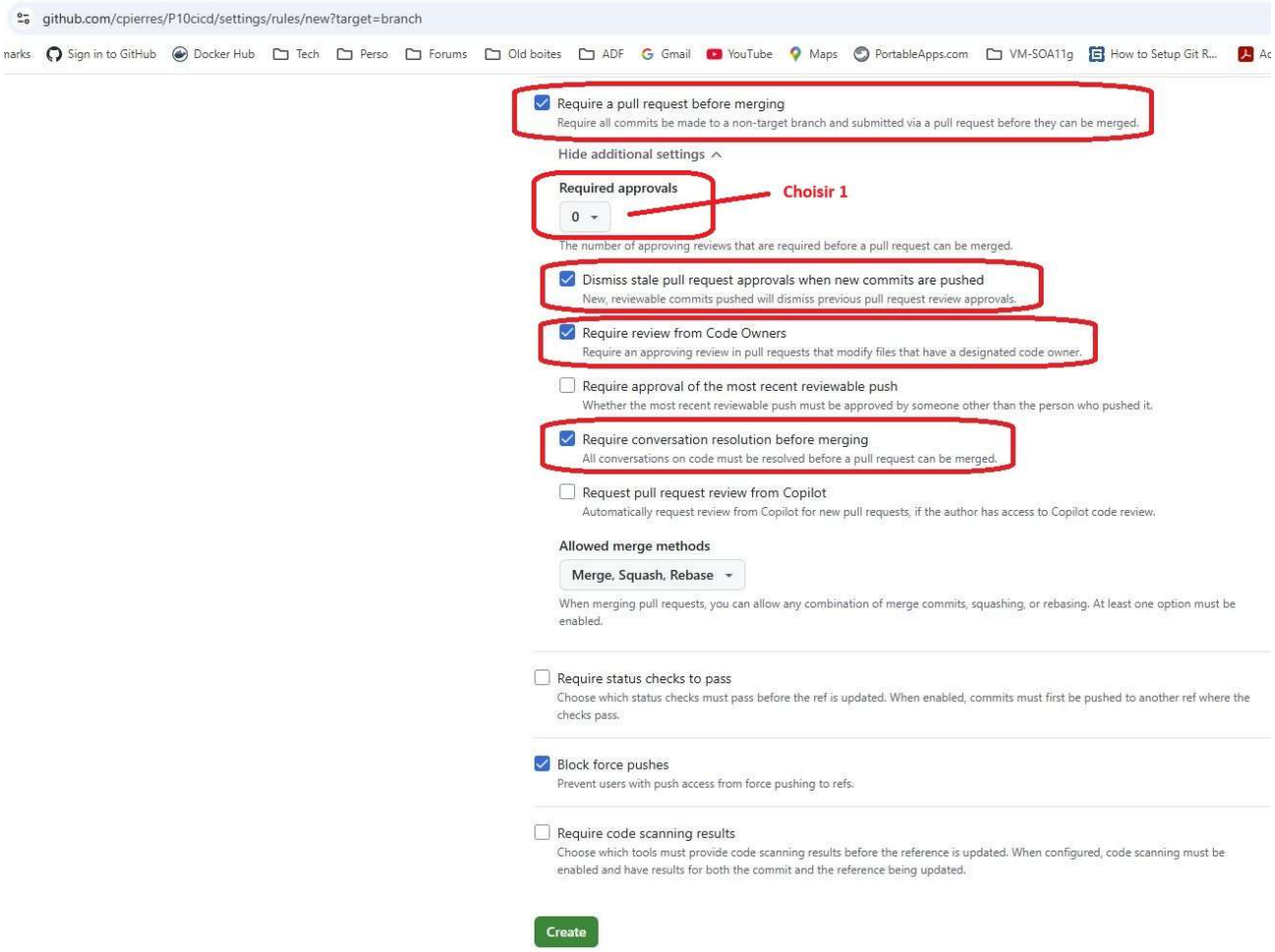


Activer les options suivantes :

✓ Require a pull request before merging

- ✓ Dismiss stale pull request approvals when new commits are pushed
- ✓ Require pull request reviews before merging

✓ Require review from Code Owners
✓ Require conversation resolution before merging

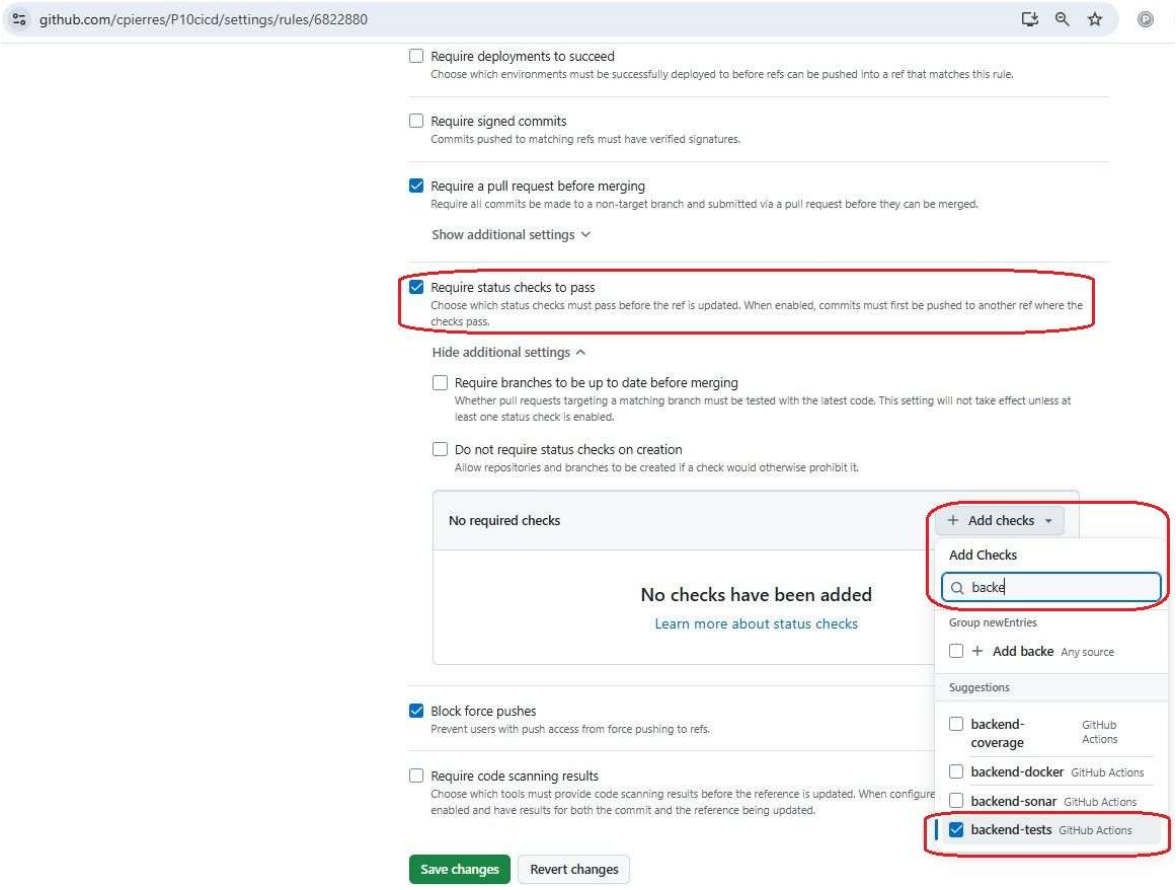


Important : **Required approvals** doit être à 1.

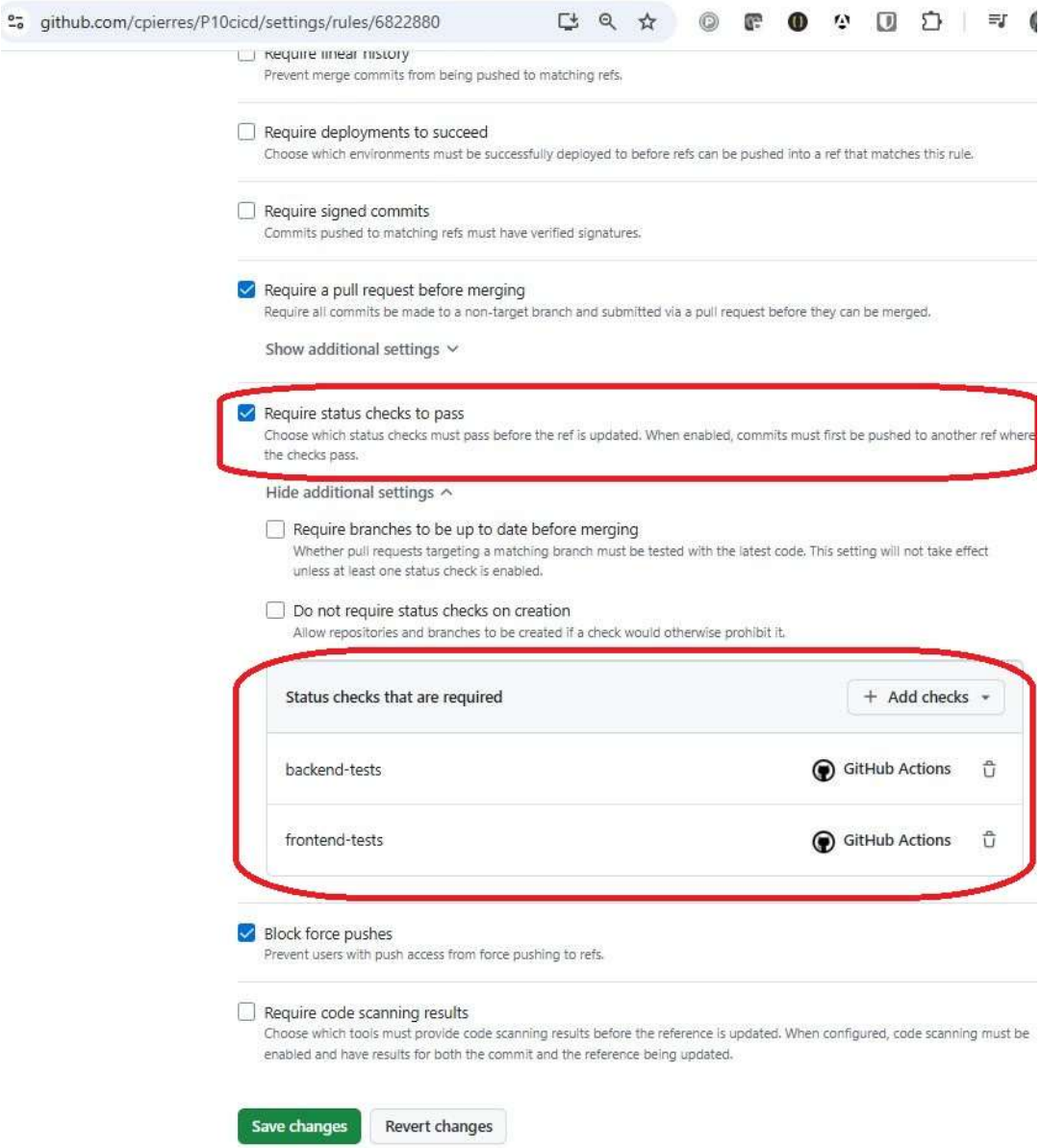
<u>Cocher également Require status checks to pass</u>

Lors de la sélection des checks de github actions, ne choisir que les tests (backend-tests et frontend-tests) car pour l'instant nous ne considérons pas qu'un mauvais résultat sonar doit bloquer le merge.

Ainsi, la PR vers main ne pourra être fusionnée que si les tests passent.

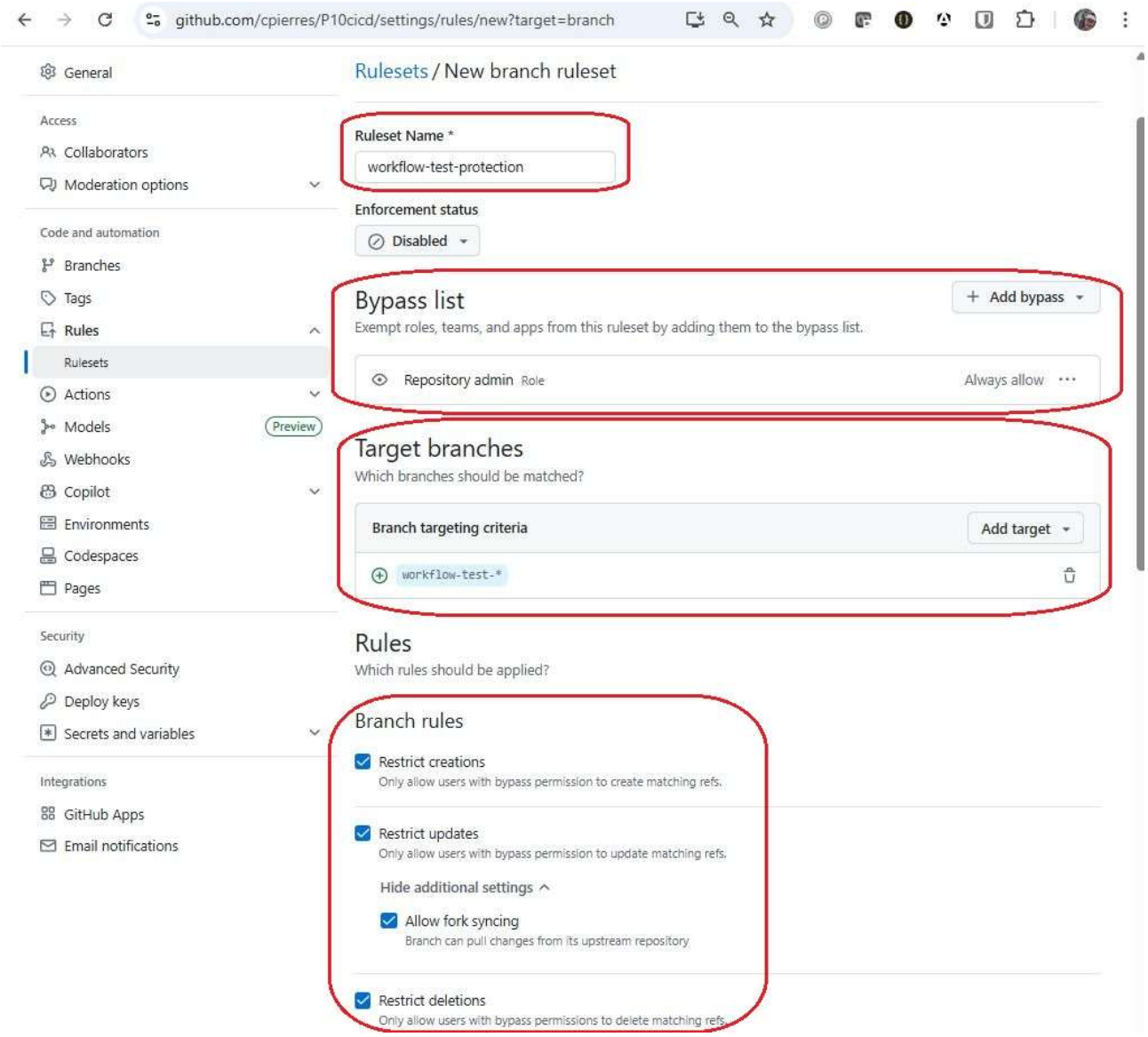


Vous devez n’avoir que ces deux checks :

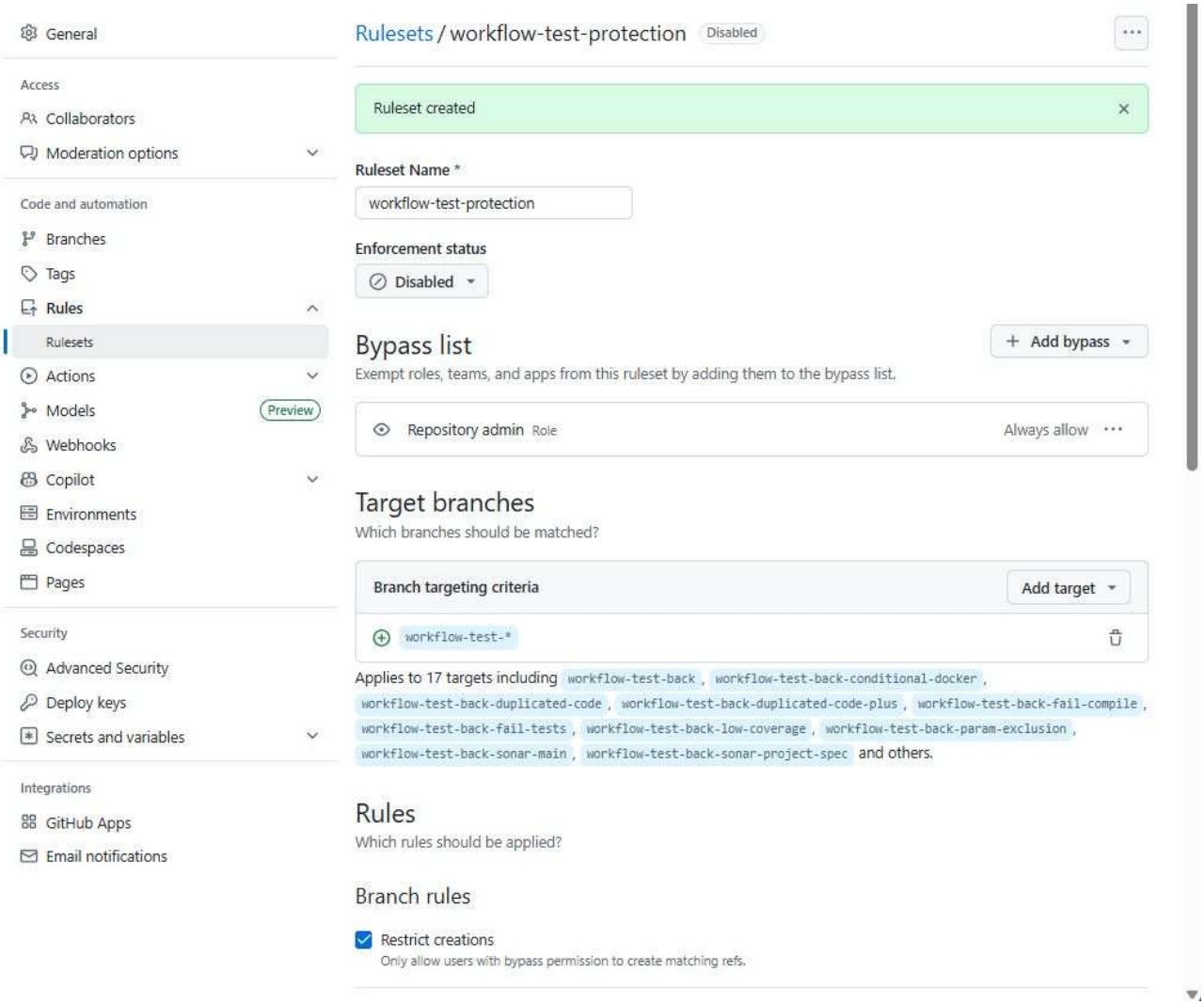


*8.1.2. Protéger les branches workflow-test-**

Pour que l'admin (ou propriétaire du dépôt) soit le seul à pouvoir pousser directement dans ce pattern de branches :



Quelques branches respectant le pattern sont affichées :



8.1.3. Création du fichier .github/CODEOWNERS

L'objectif de ce fichier sert plusieurs objectifs importants :

1. Attribution des reviewers

Le fichier CODEOWNERS permet de désigner automatiquement des personnes ou des équipes comme reviewers lorsqu'une pull request modifie certains fichiers. Cela garantit que les personnes les plus compétentes ou responsables d'une partie spécifique du code examinent les modifications qui y sont apportées.

2. Définition claire des responsabilités

En documentant qui est responsable de quelles parties du code, le fichier CODEOWNERS clarifie les responsabilités au sein d'une équipe ou d'un projet. Cela aide à savoir à qui s'adresser pour des questions ou des problèmes concernant des parties spécifiques du code. On peut désigner des répertoires pour chaque reviewer (exemple : le ou les responsables des PR du backend ainsi que ceux ou celles pour le frontend)s

3. Garantie de la qualité du code

En s'assurant que les modifications sont toujours examinées par les personnes appropriées, le fichier CODEOWNERS contribue à maintenir la qualité du code et à prévenir l'introduction de bugs ou de problèmes de sécurité.

4. Facilitation du processus de revue

Le processus de revue de code est simplifié car les reviewers sont automatiquement assignés (en fonction des répertoires par exemple), éliminant ainsi la nécessité de rechercher manuellement qui devrait examiner une pull request.

5. Documentation versionnable des propriétaires du code

Contrairement à une configuration dans l'interface utilisateur, le fichier CODEOWNERS est versionné avec le code source, ce qui permet de suivre l'évolution des responsabilités au fil du temps et de conserver un historique des changements.

6. Granularité des attributions

Le fichier CODEOWNERS permet une grande flexibilité dans l'attribution des responsabilités, permettant de spécifier différents propriétaires pour différents fichiers ou répertoires, en utilisant des patterns similaires à ceux de .gitignore.

7. Protection du code critique

Pour les parties critiques du code, le fichier CODEOWNERS est utilisé en conjonction avec les règles de protection de branche pour exiger l'approbation des propriétaires désignés avant que les modifications puissent être fusionnées.

8. Auto-protection

Le fichier CODEOWNERS peut se protéger lui-même en désignant des personnes spécifiques comme propriétaires du fichier CODEOWNERS, comme on peut le voir dans la configuration que nous avons définie.

8.1.3.1. Contenu du fichier

```
# Le fichier CODEOWNERS lui-même ne peut être modifié que par l'administrateur
CODEOWNERS @cpierres

# Définit cpierres comme reviewer pour tous les fichiers du projet
* @cpierres
```

L’avantage d’avoir un seul est que la PR sera automatiquement associée à cpierres pour review.

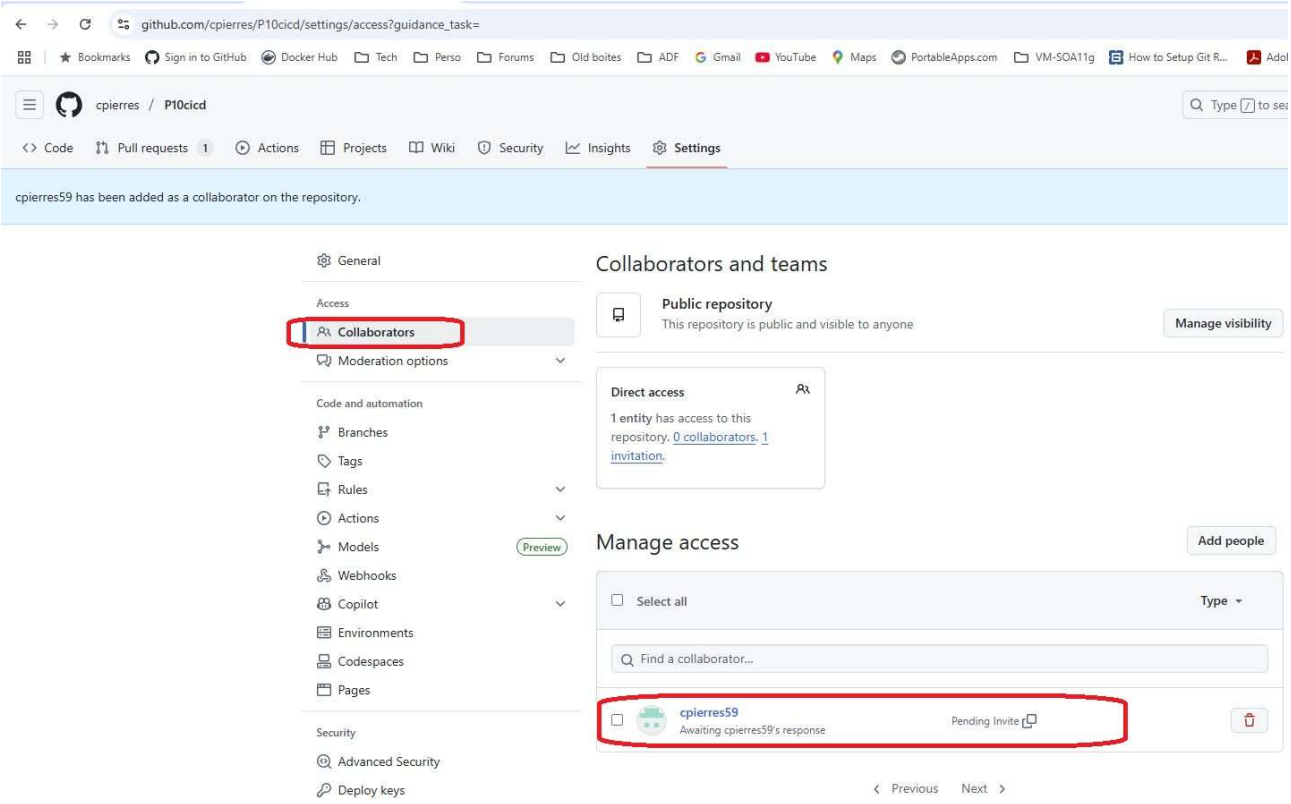
Note : il serait possible de définir plusieurs approbateurs si le projet devient plus gros et également de les spécialiser au besoin par répertoire (responsable/reviewer des PR du backend et responsable des PR du frontend).

8.1.3.2. Optionnel : inviter un collaborateur (cpierres59) directement sur ce repo

On ne souhaite pas que les PR soient envoyées directement au dépôt parent (OpenClassrooms) mais qu’elles puissent être gérées localement sur le dépôt forké par une équipe de 2 personnes (le propriétaire cpierres du dépôt forké et un collaborateur cpierres59).

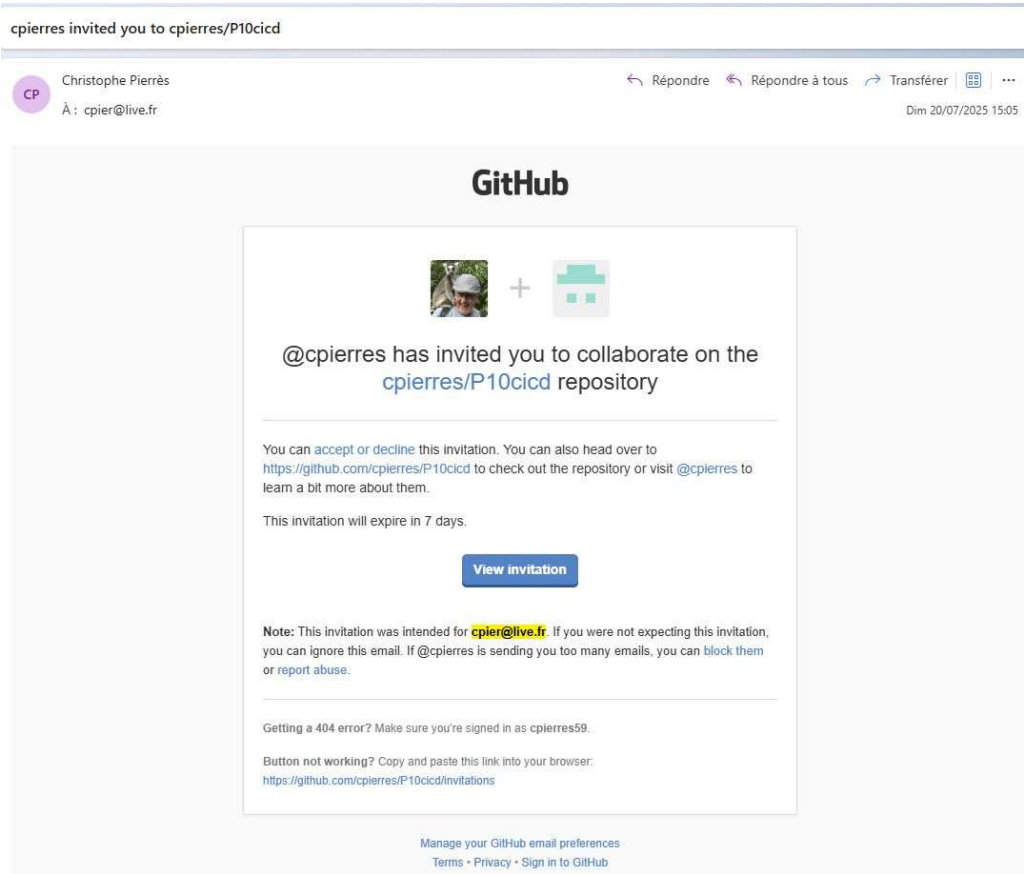
Pour l’exemple, j’ai créé un second compte cpierres59, lequel pourra effectuer directement des pushs sur ce repo. C’est un contributeur faisant partie de l’équipe.

Propriétaire du compte cpierres envoi une invitation à cpierres59

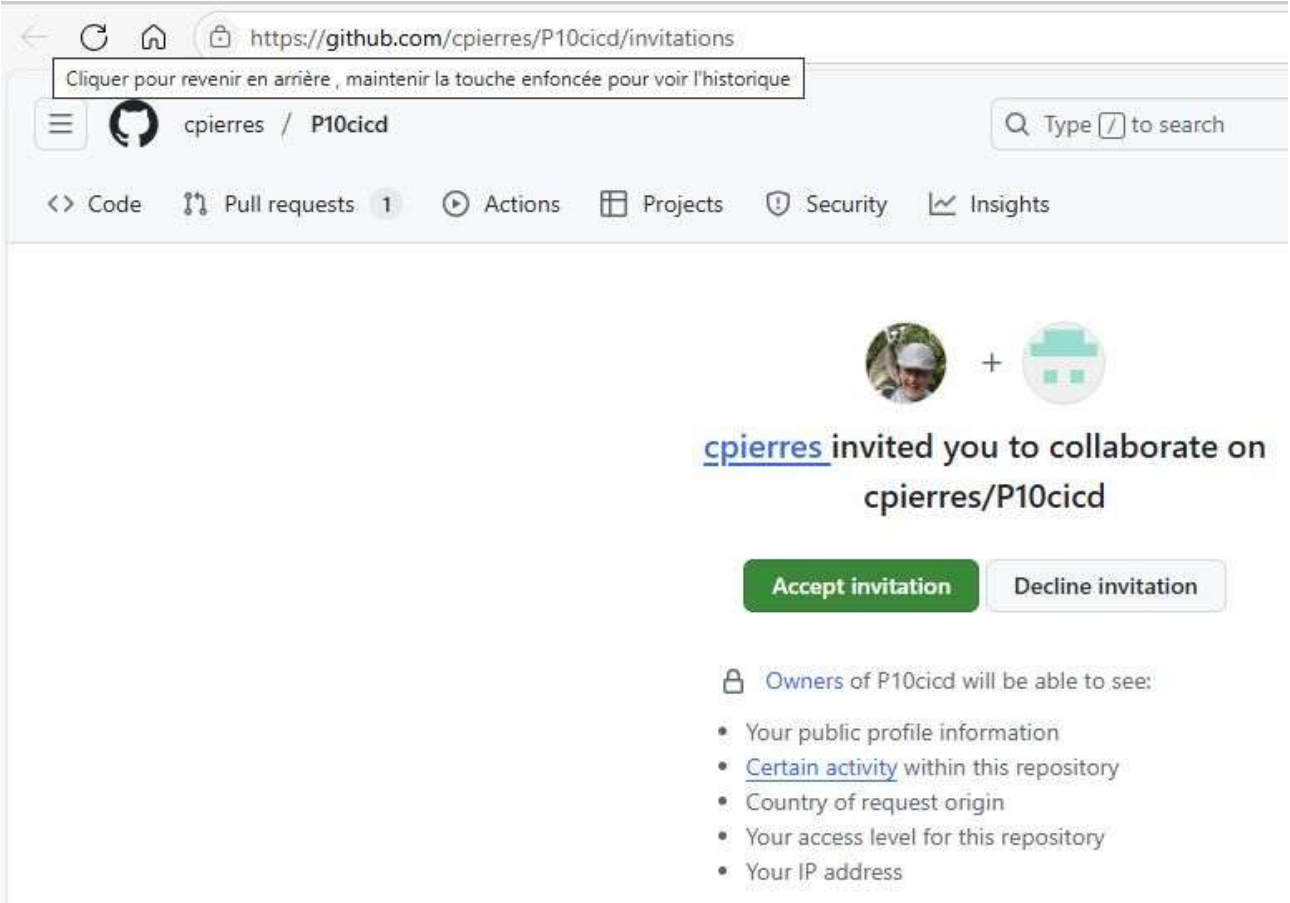


Le collaborateur reçoit un email pour accepter l'invitation

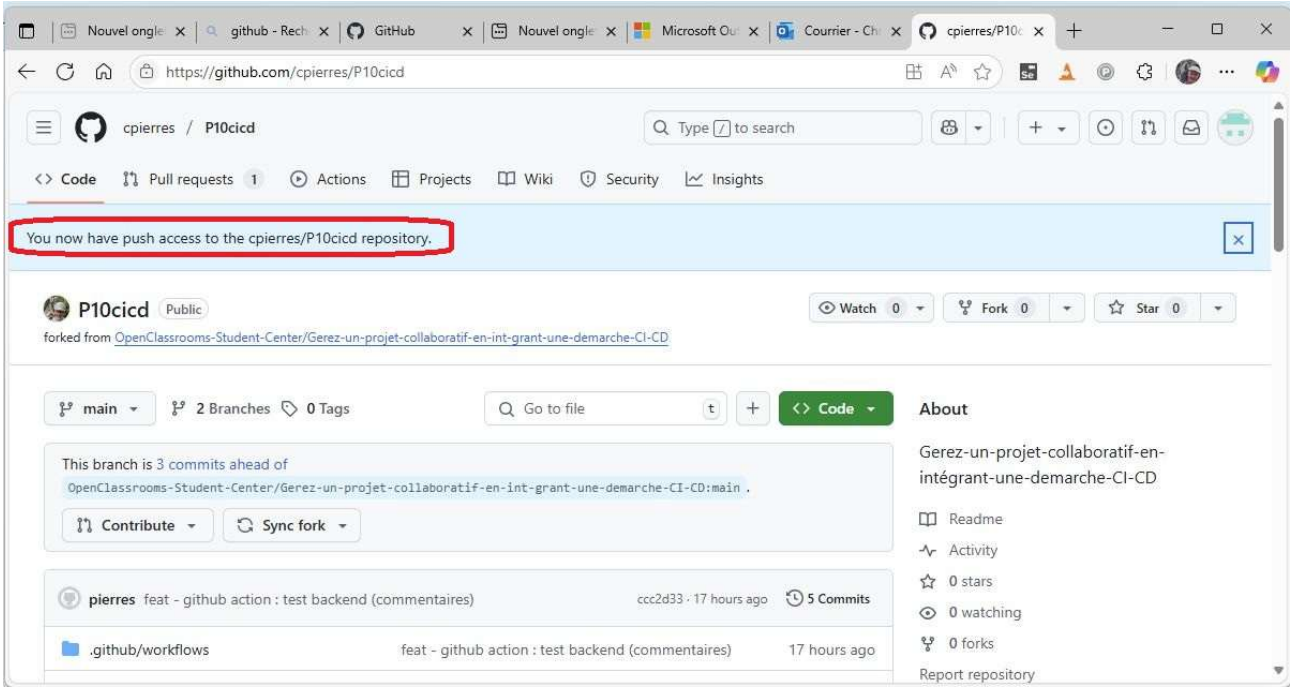
Note : utiliser un browser différent de la session du propriétaire.



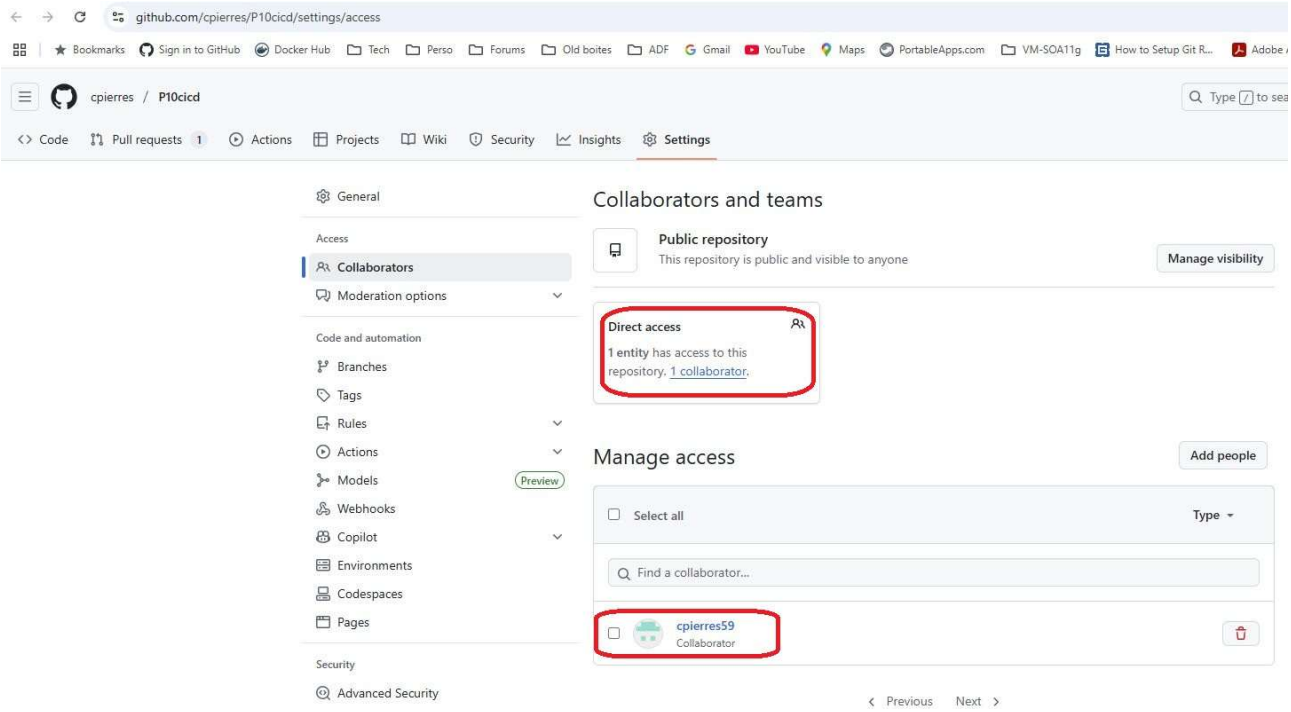
L'écran suivant apparait pour Accepter l'invitation



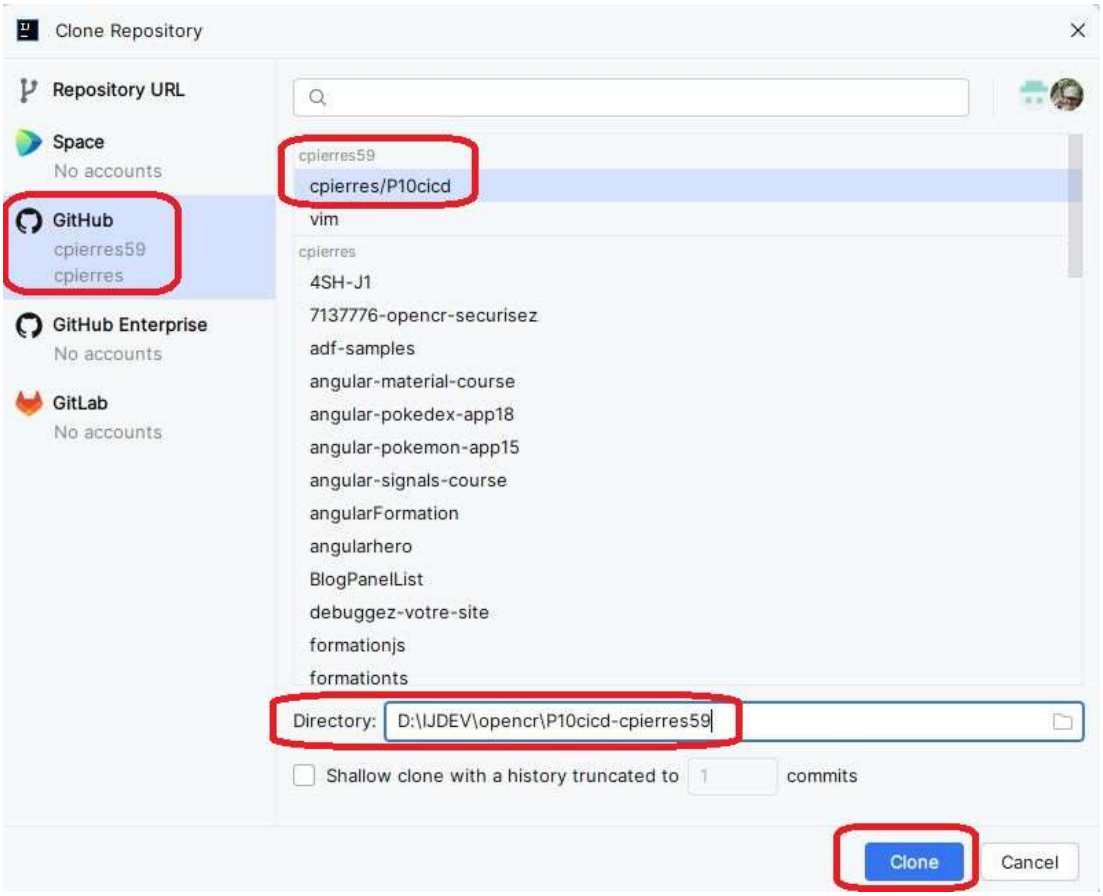
Le collaborateur peut désormais cloner le repository



Le propriétaire voit le collaborateur inscrit



cpierres59 crée un autre projet dans intelliJ avec un répertoire de clonage différent de la session cpierres



Définir le user spécifique à utiliser pour ce dépôt local :

```
D:\IJDEV\opencr\P10cicd-cpierres59 git:[main]
git config user.name "cpierres59"

D:\IJDEV\opencr\P10cicd-cpierres59 git:[main]
git config user.email "cpier@live.fr"

D:\IJDEV\opencr\P10cicd-cpierres59 git:[main]
```

8.1. Exemples de scénarios pour tester les workflows

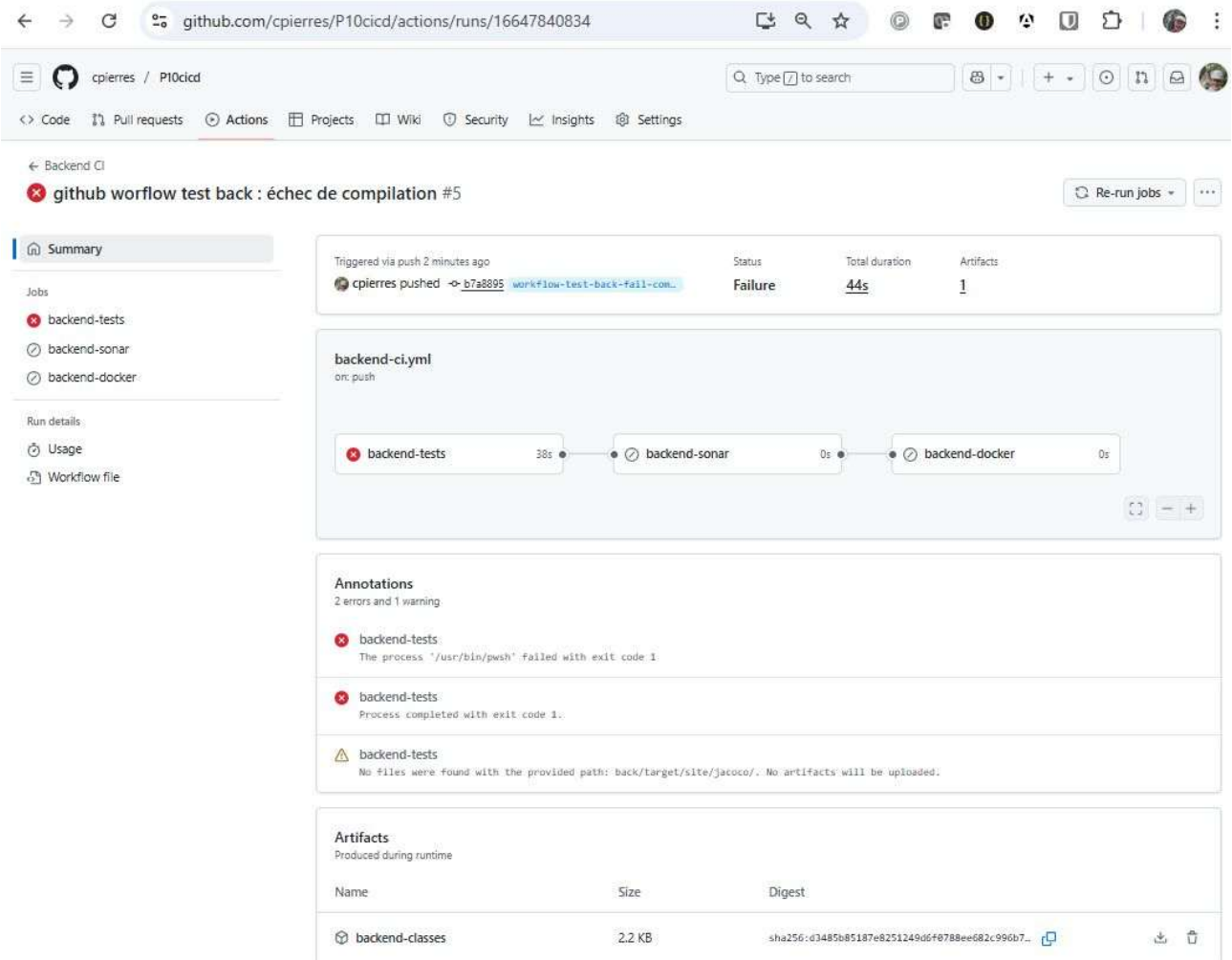
Ci-dessous une série de tests qui s'exécute dans des branches spécifiques pour tester le comportement des workflows sans impacter le code existant.

8.1.1. Tests pour le backend

8.1.1.1. Test d'échec de compilation

Branche `workflow-test-back-fail-compile`.

Le workflow s'interrompt, comme attendu.



8.1.1.2. Test d'échec des tests unitaires

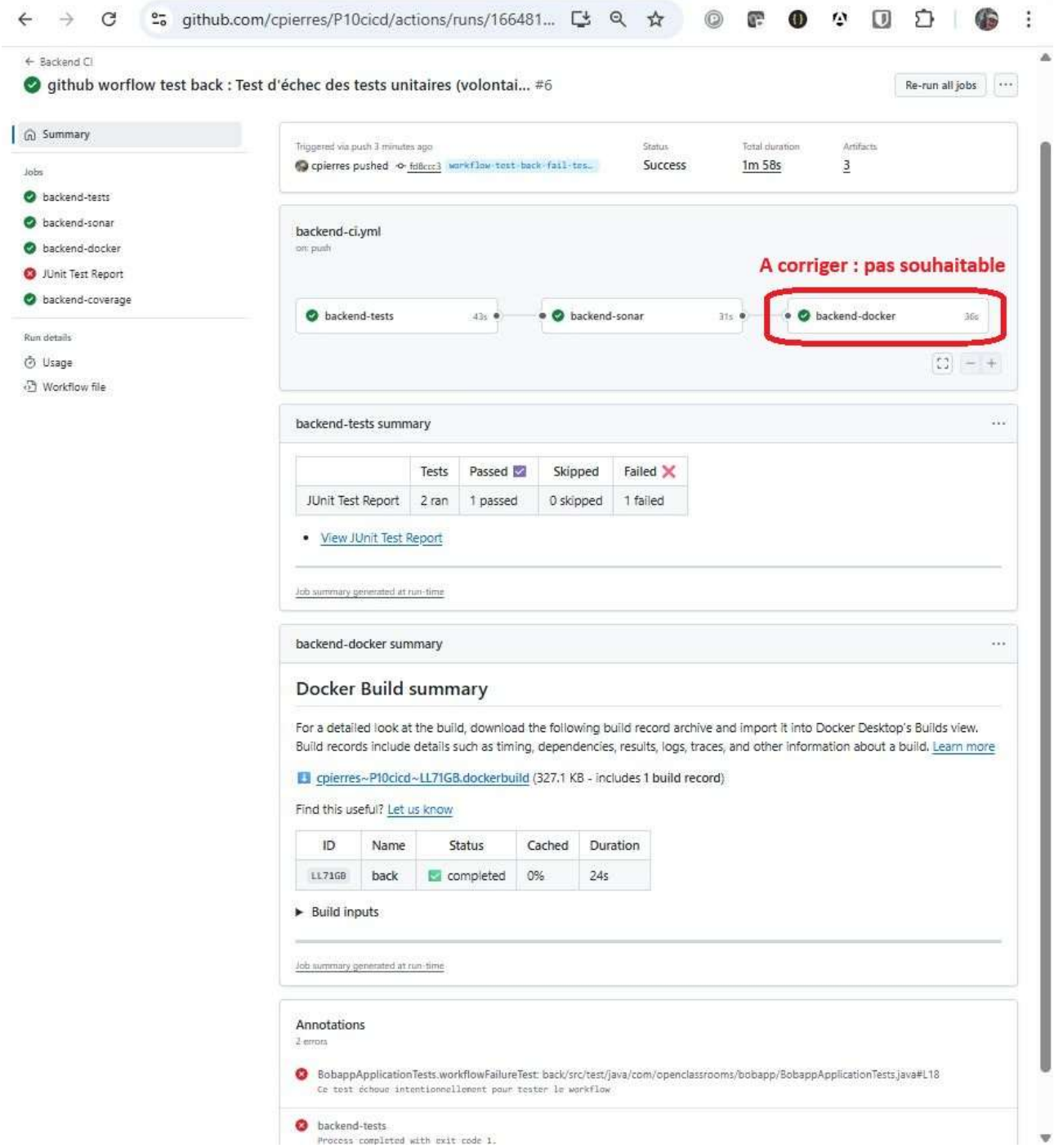
Dans une branche workflow-test-back-fail-tests , j'ai ajouté ce test à un test existant :

```
@Test
```

```
void workflowFailureTest() {
    // Assertion qui échouera toujours
    fail("Ce test échoue intentionnellement pour tester le workflow");
}
```

Exécution du workflow (non souhaitable : container ne devrait pas être généré)

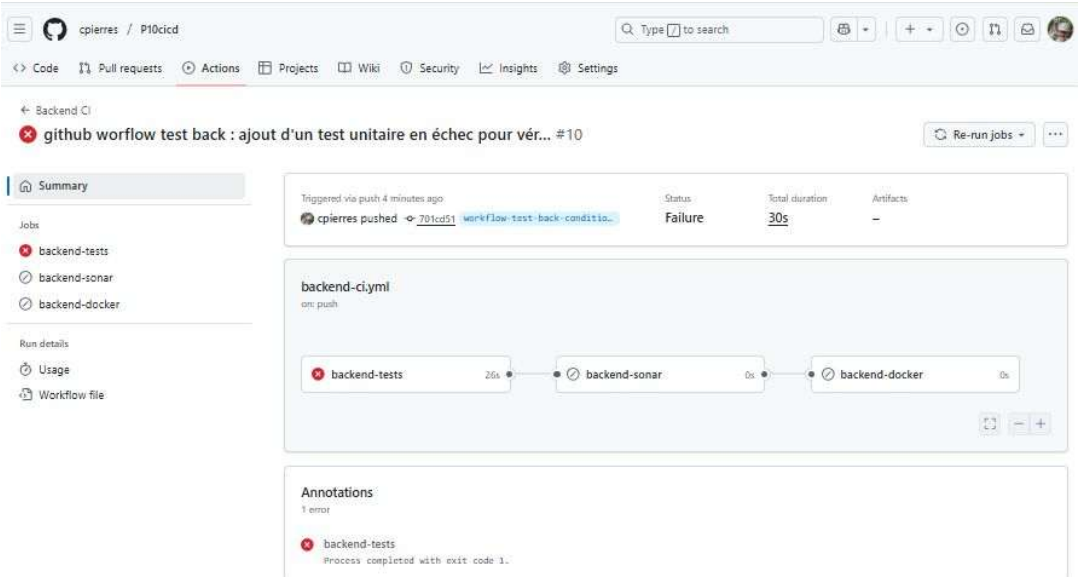
One doit pas exécuter les contrôles qualité sonar ni la génération du container. Je vais améliorer mon workflow.



Correctif du workflow backend-cy.yml

Correctif effectué dans la branche `workflow-test-back-conditional-docker`.

Le workflow se comporte désormais comme attendu :



8.2. Gérer les PullRequest (PR)

Déclencher l'exécution du rapport (via un push vers main par le propriétaire ou une PR approuvée)

8.2.1. Créer une branche de modification par un collaborateur contributeur (cpierres59)

Pour la démonstration, nous allons effectuer une modification simple consistant à afficher un log dans la classe JokeService.

Pour créer une PR, nous devons créer une branche :

```
git checkout -b feature/add-service-log
```

Commiter et pousser les modifications :

```
git add back/src/main/java/com/openclassrooms/bobapp/service/JokeService.java
git commit -m « feat: ajout log lors du retour d'une blague aléatoire »
git push origin feature/add-service-log
```

8.2.2. Création de la Pull Request par le contributeur

Aller sur GitHub

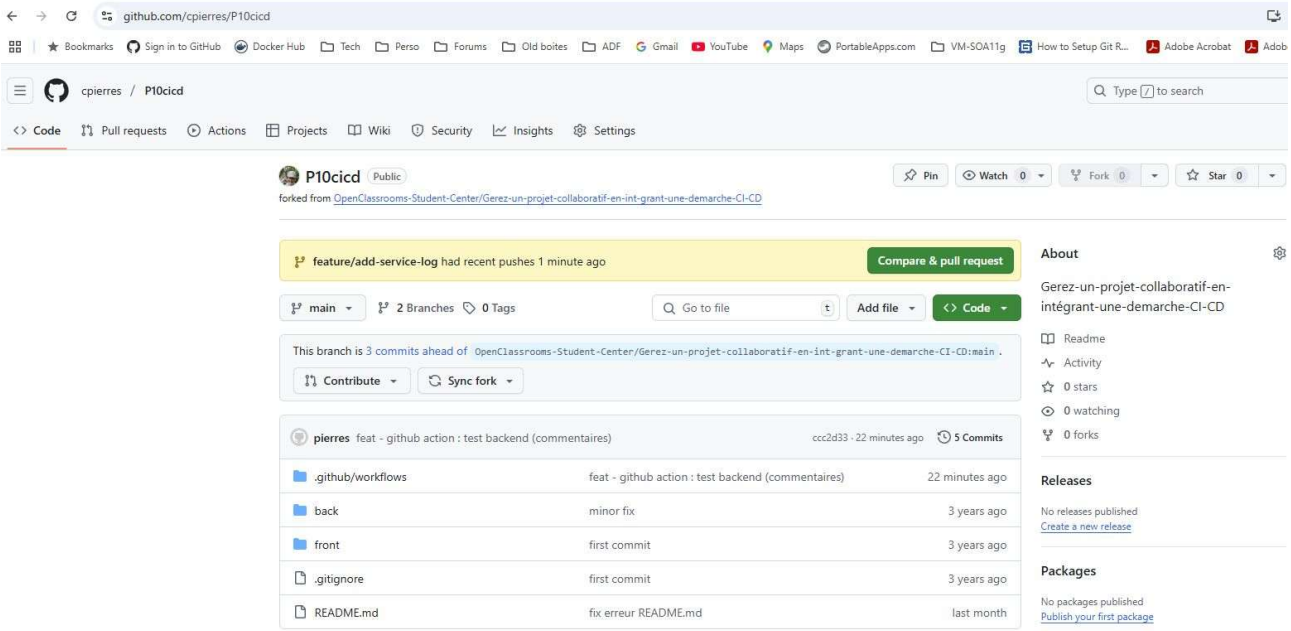
Créer une Pull Request depuis la branche `feature/add-service-log` vers main

Titre : « Feature: Add logging in JokeService"

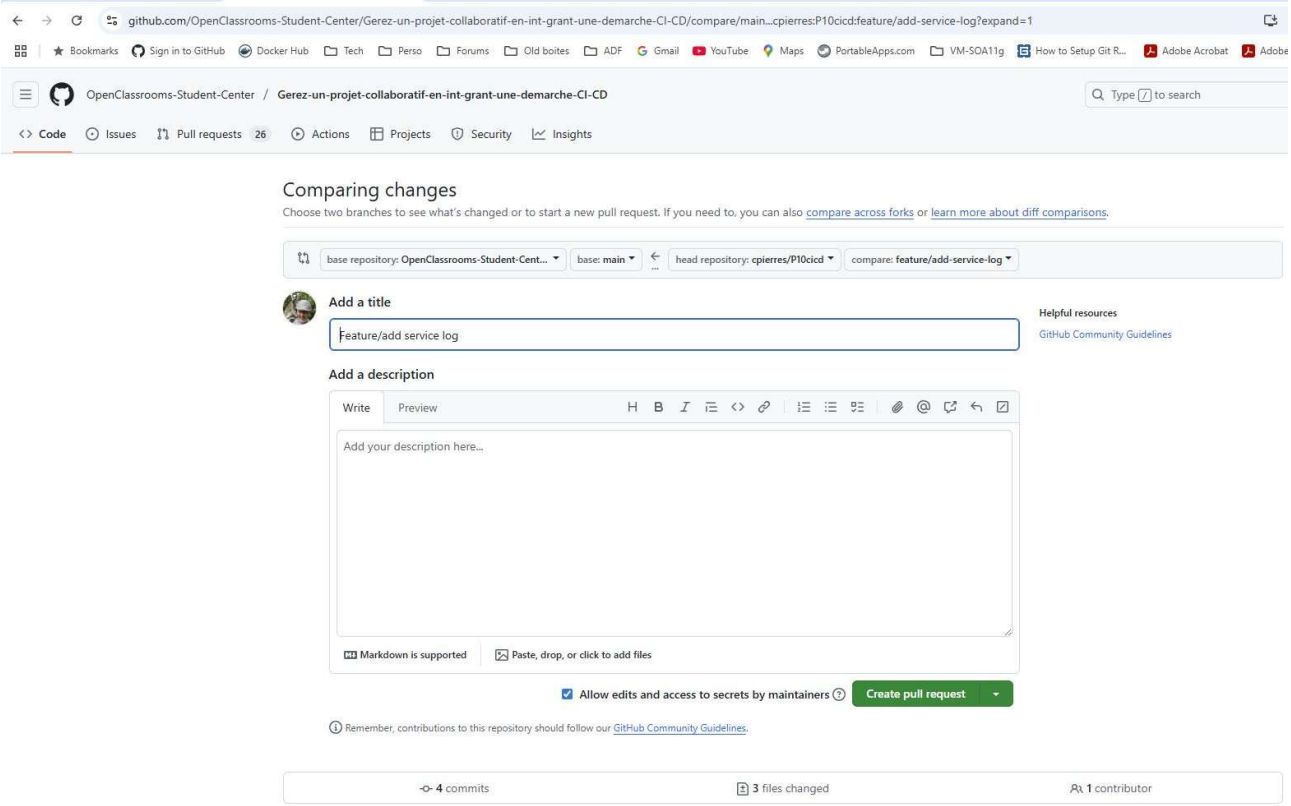
Description suggérée :

Pour créer une Pull Request (PR) sur GitHub après avoir poussé la branche, voici les étapes :

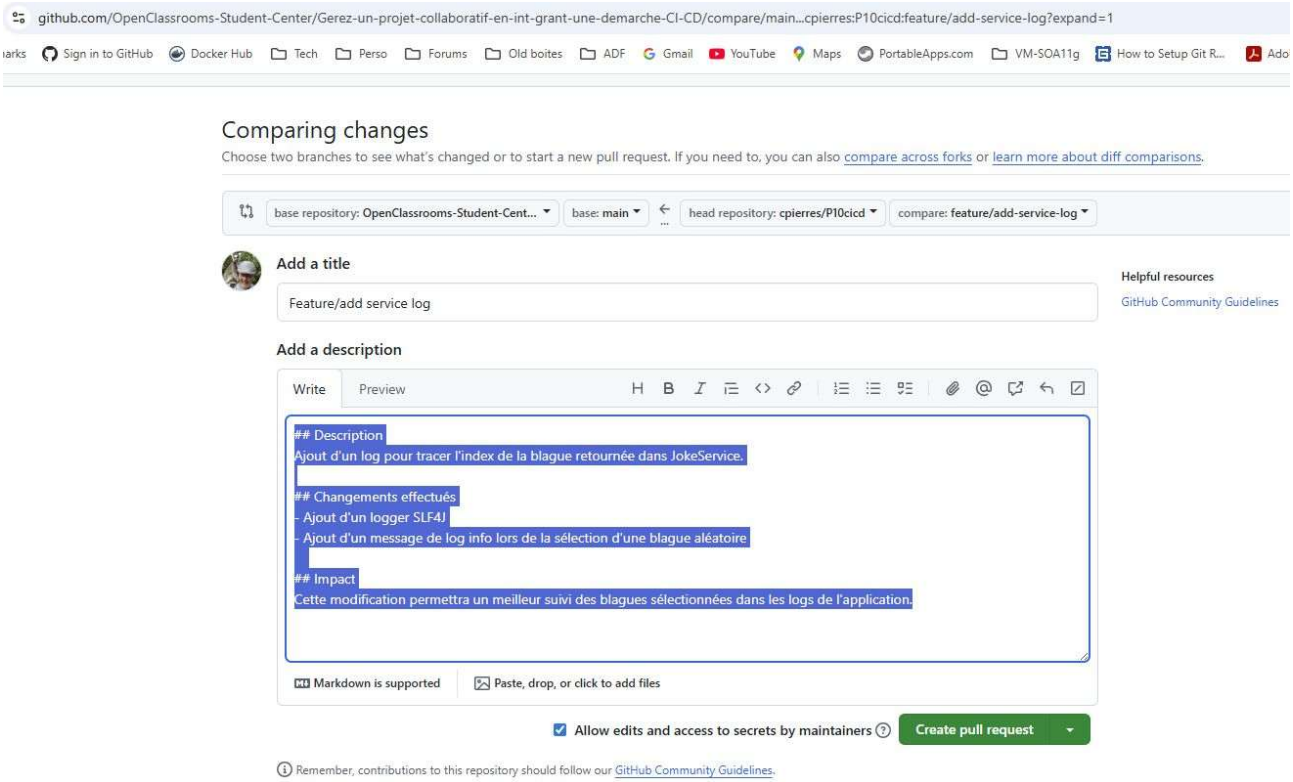
1. Depuis l'interface GitHub
   - Aller sur la page du repository
   - Un bandeau jaune apparaîtra :

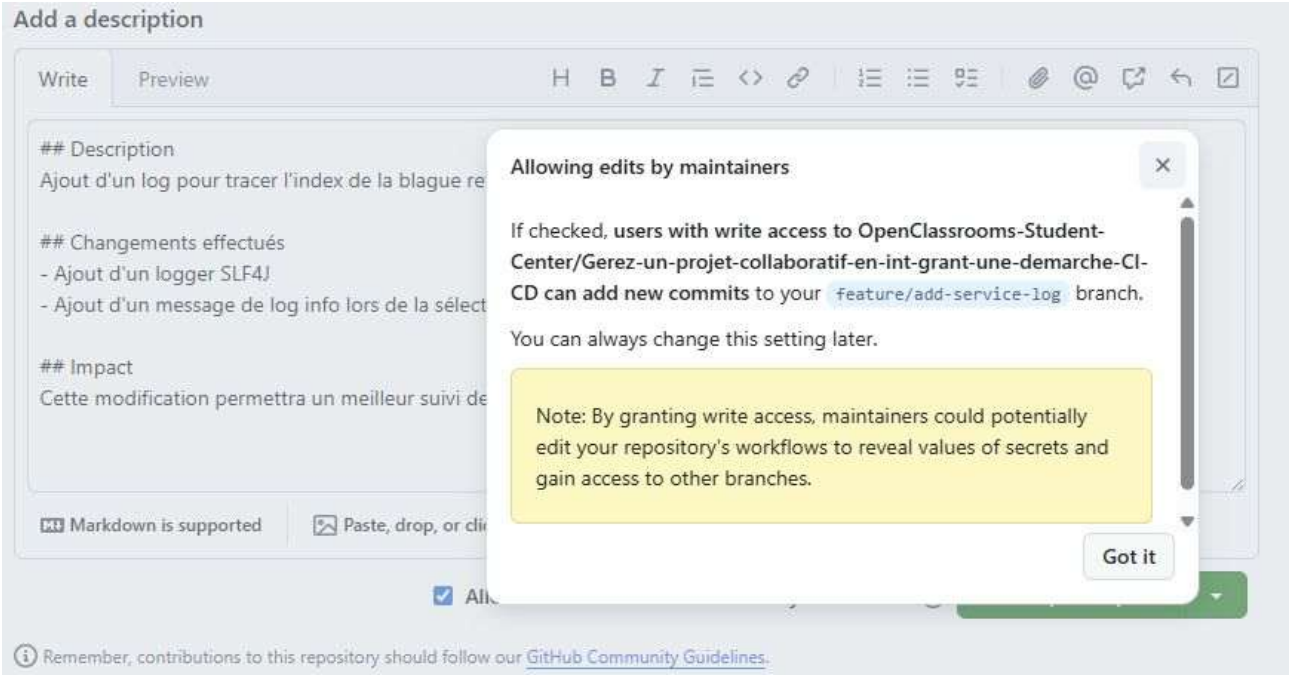


- Cliquer sur « Compare & pull request » (attention de bien bien choisir le **base repository** sous peine d'adresser la PR au projet parent plutôt qu'à votre dépôt forké et à votre équipe !)

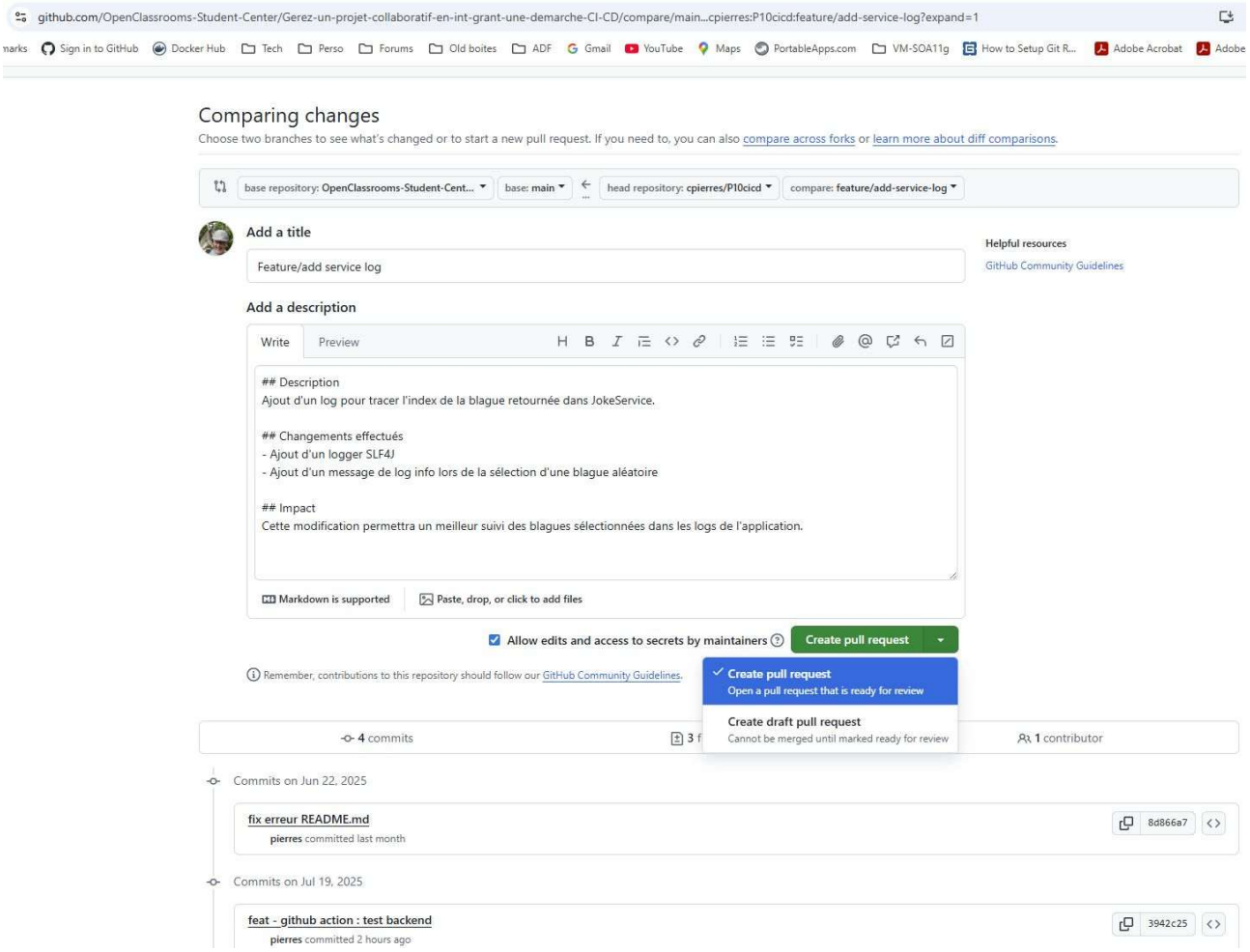


2. Remplir les informations de la PR

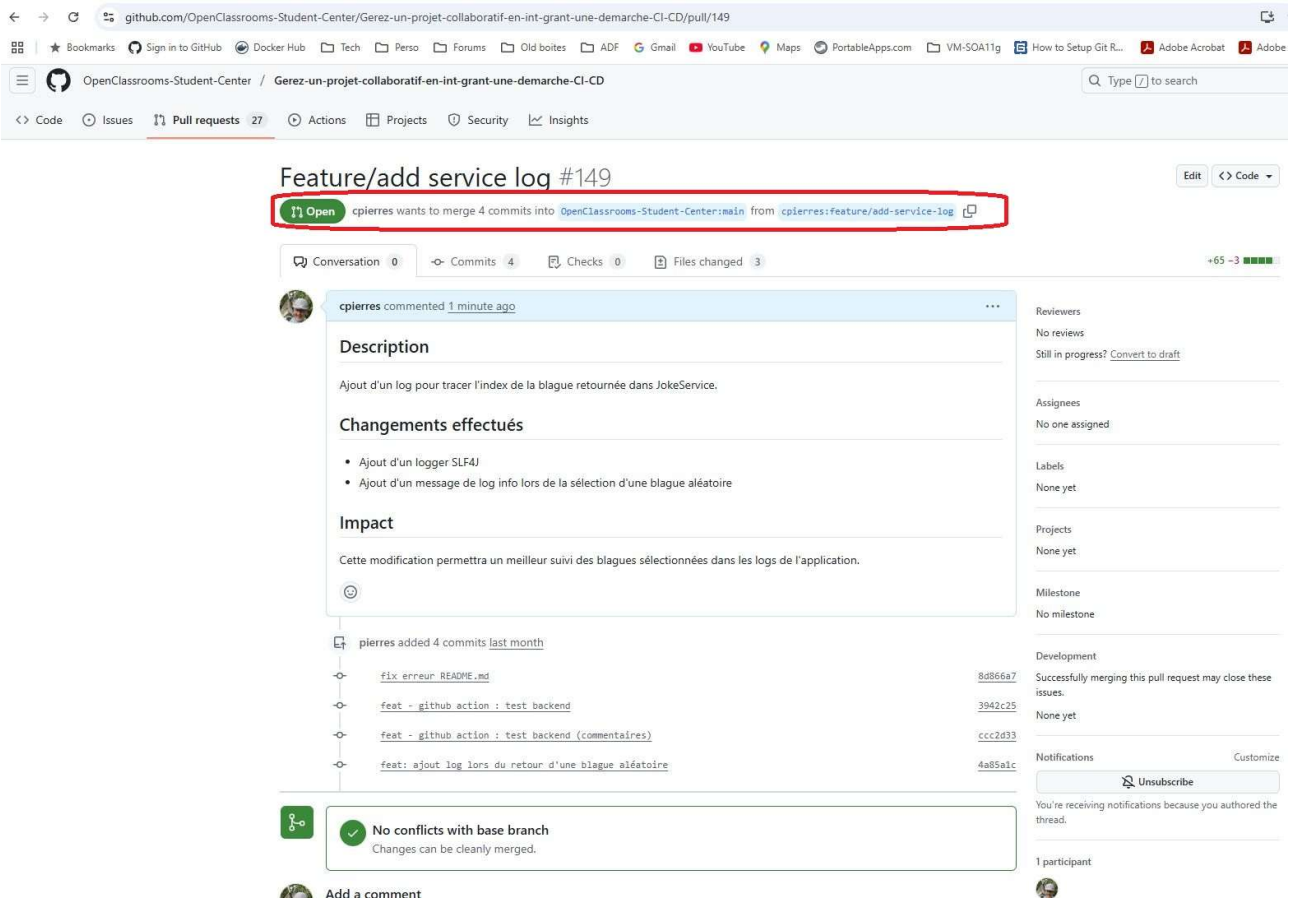




Cliquer sur Create pull request :



La PR est créée :

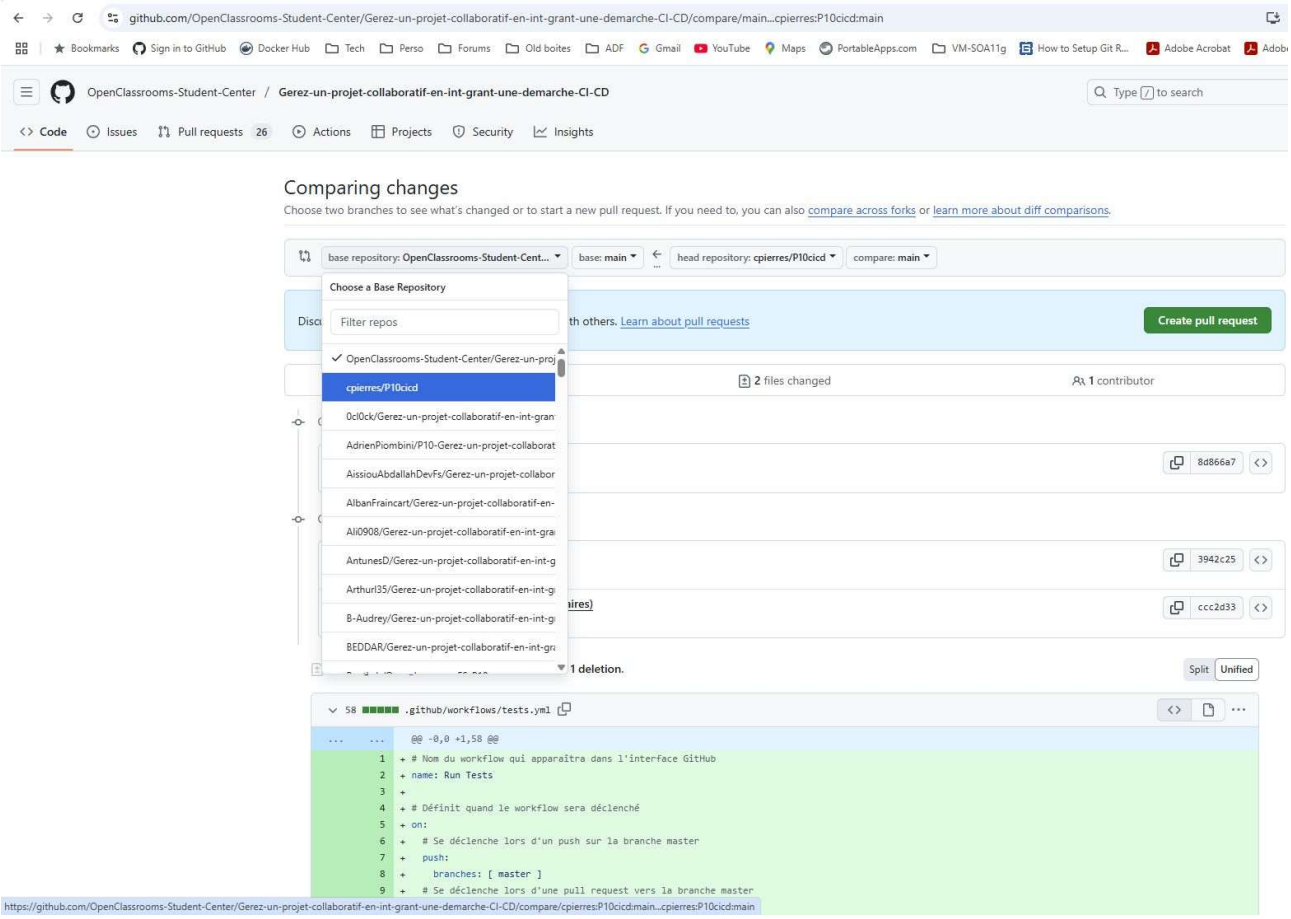


Erreur ! La PR a été affectée au projet Parent au lieu de notre propre dépôt.

8.2.3. Fermer cette PR mal adressée avec un commentaire

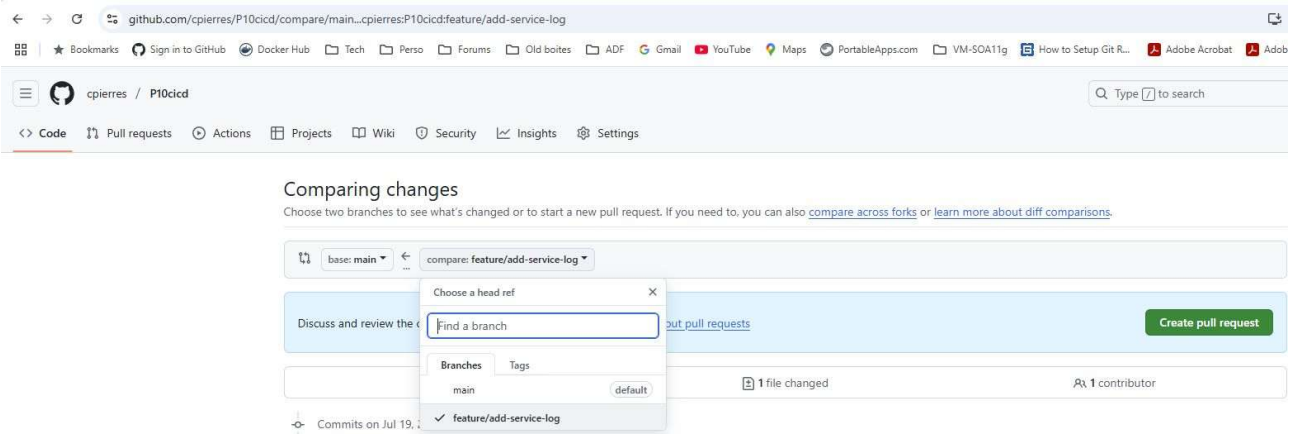
Ajouter un commentaire lié à l'erreur.

8.2.4. Re-créer la Pull Request en se référant à la base de votre propre dépôt !

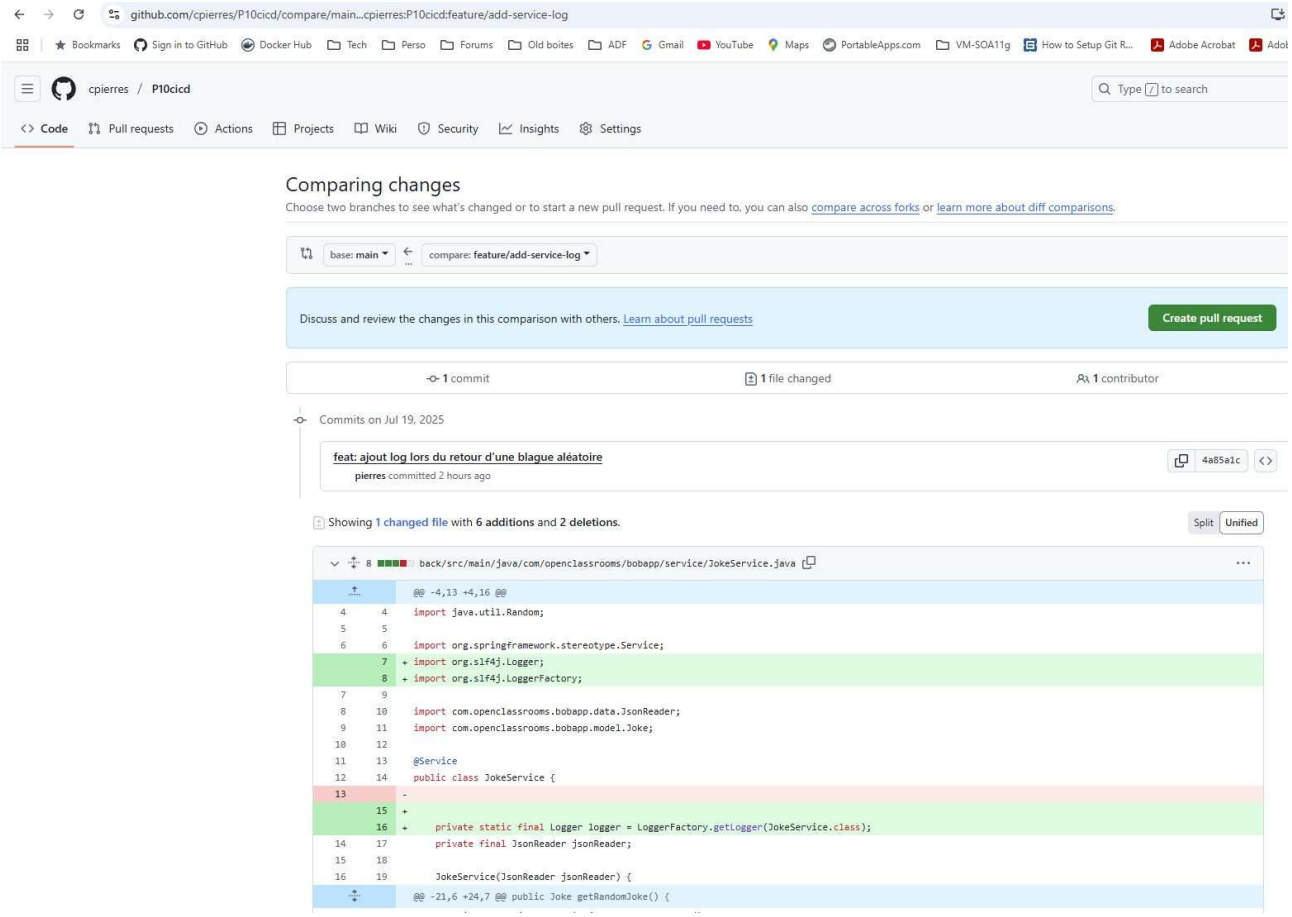


Lorsqu'on sélectionne cpierres/P10cicd, on revient à notre repository.

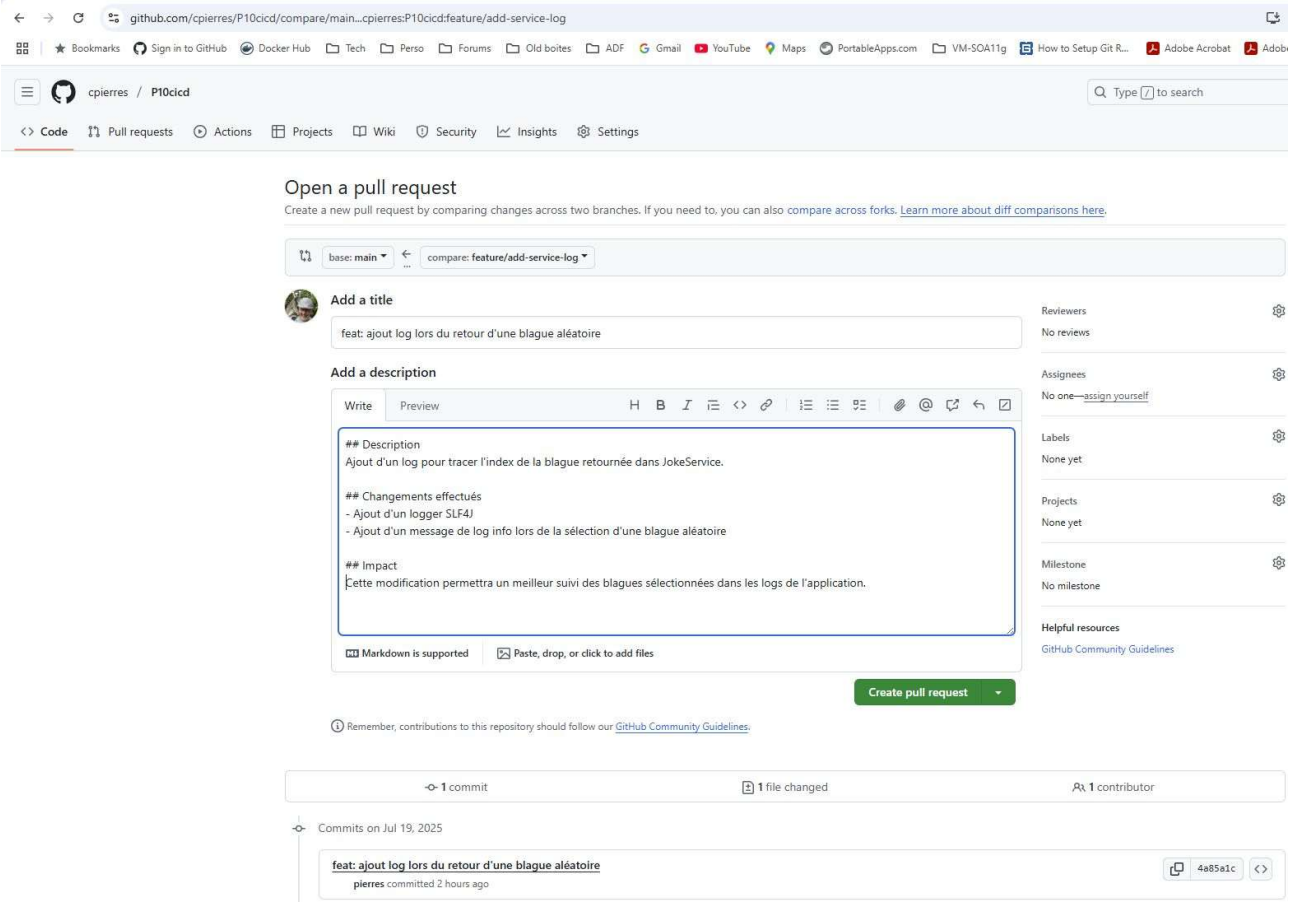
Comparer avec notre branche feature/add-service-log :



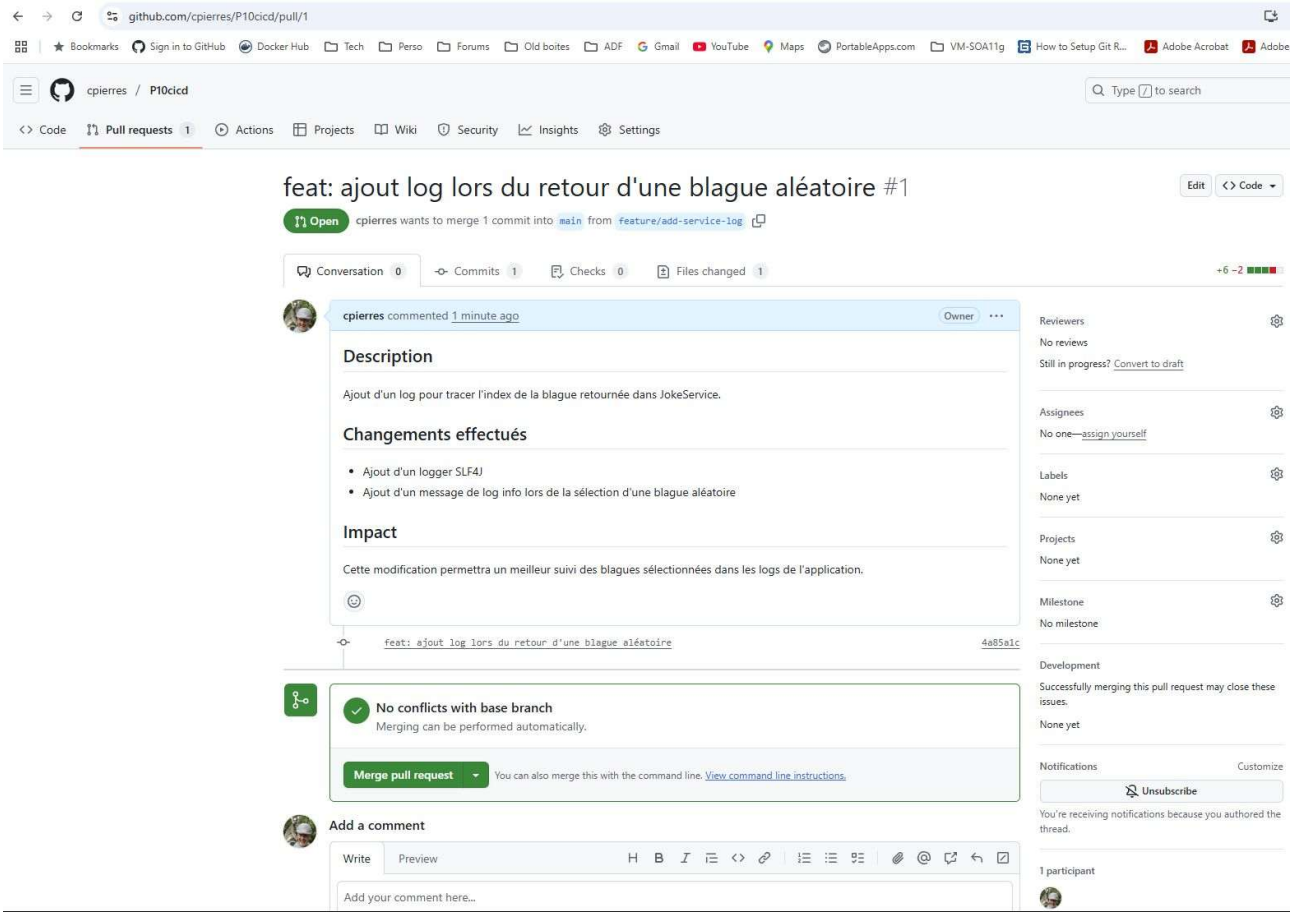
Ce qui donne :



Cliquer sur Create pull request et compléter la description :

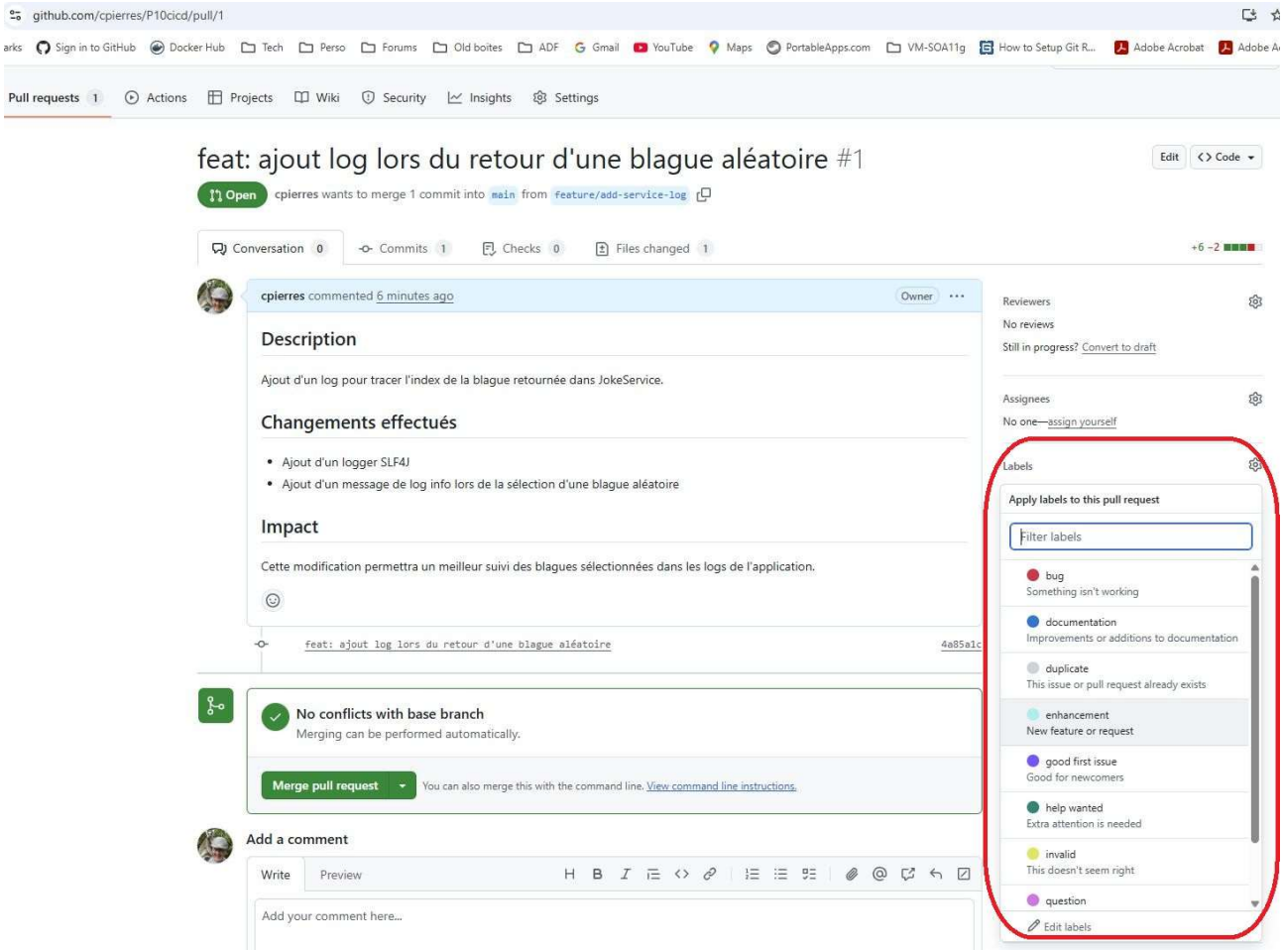


Cliquer à nouveau sur Create pull request :

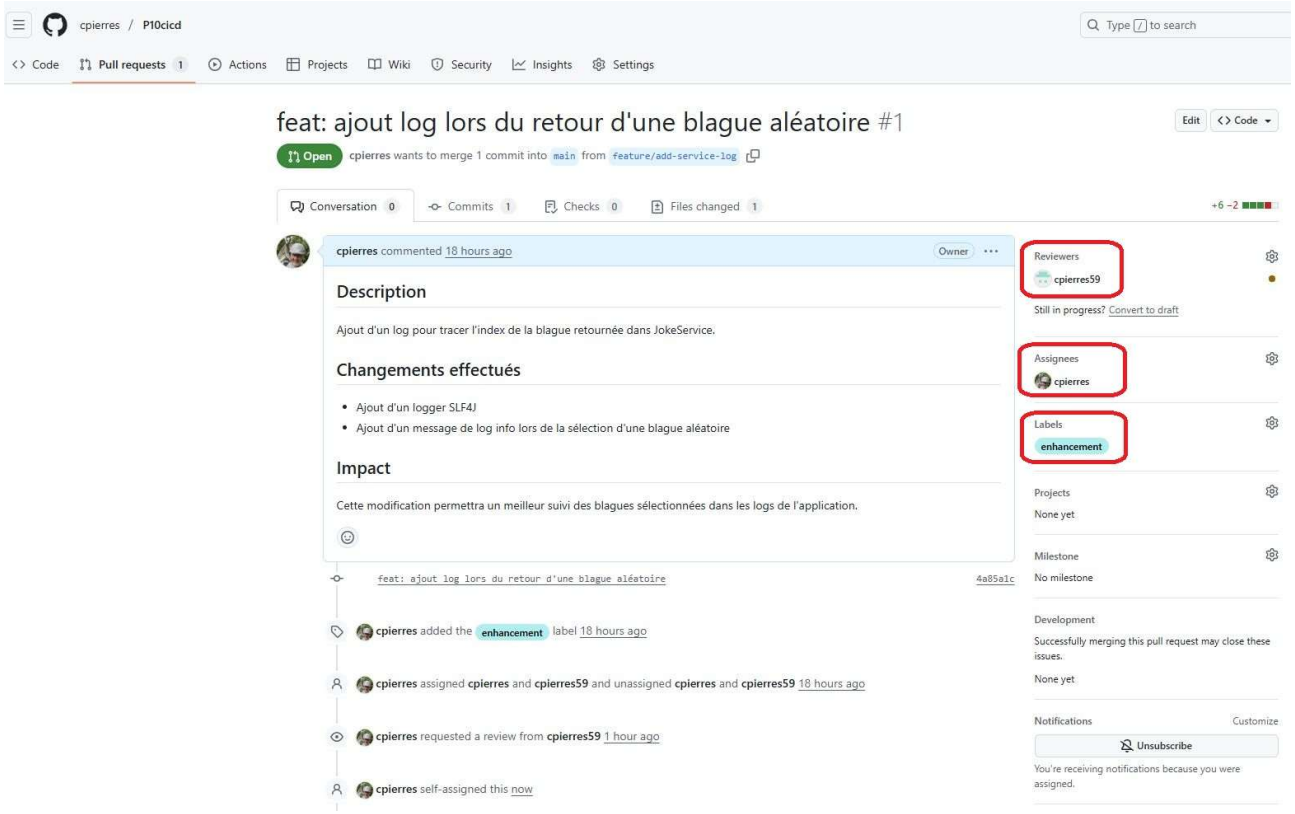


8.2.5. Traiter la PR par un reviewer

A l'étape précédente, nous aurions dû attribuer des labels et un Assignee (je fais cela maintenant) :



Le reviewer est assigné à cpierres59 ainsi que le assignee (capacité à faire le merge vers la branche main)



8.2.5.1. Rôle Reviewer

- Examine le code ligne par ligne
- Commente les modifications
- Suggère des améliorations
- Approuve ou demande des changements
- Se concentre sur la qualité technique du code

Actions typiques

- Lire et analyser les changements de code
- Laisser des commentaires sur des lignes spécifiques
- Suggérer des modifications
- Approuver la PR
- Demander des changements
- Simplement commenter

8.2.5.2. Rôle Assignee

- Responsable global de la Pull Request
- Gestionnaire du processus de la PR
- Décideur final sur le merge
- Coordonne le processus de review
- Propriétaire de la PR jusqu'à sa fermeture

Actions typiques

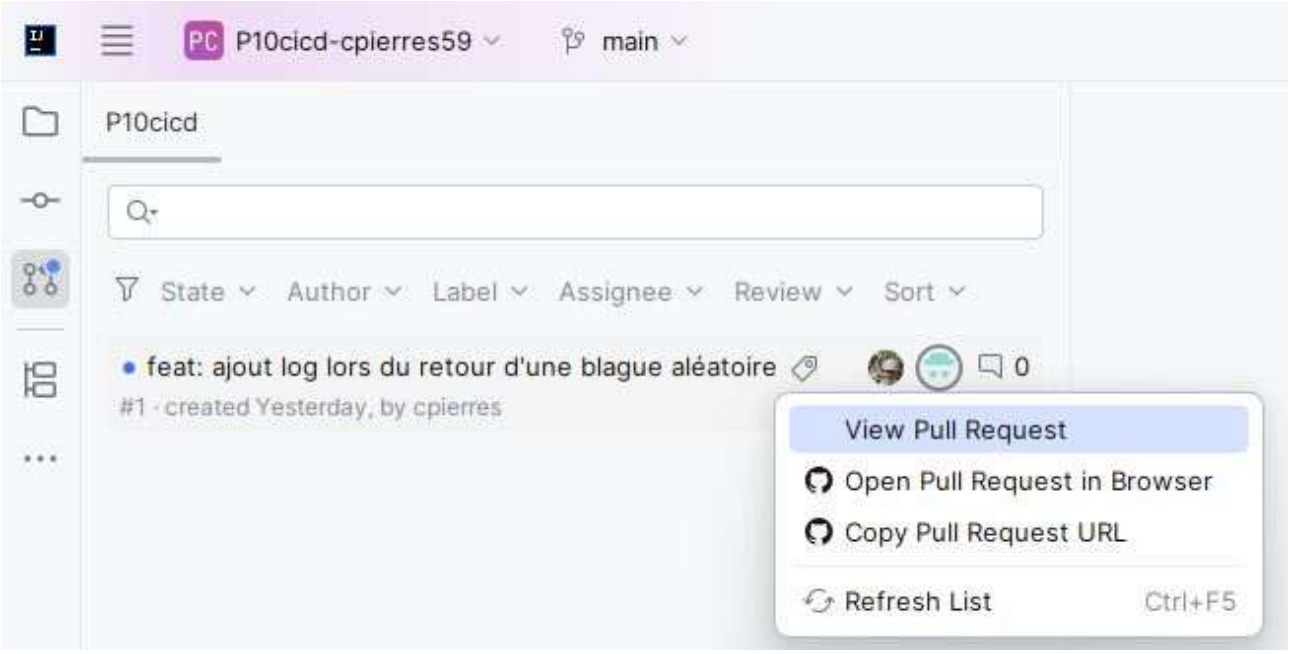
- Assigner des reviewers
- Suivre l'avancement de la PR
- Merger la PR une fois approuvée
- Fermer la PR si nécessaire
- Résoudre les conflits de merge
- Communiquer avec l'auteur et les reviewers

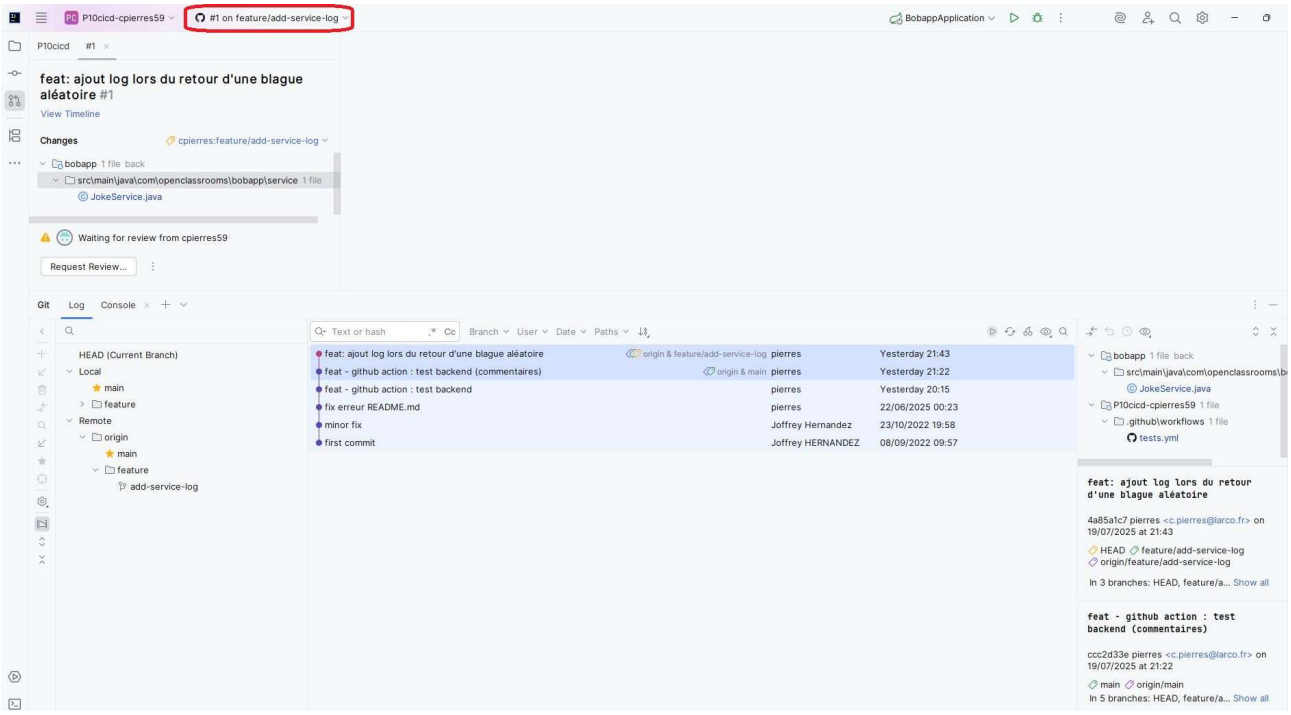
8.2.5.3. Comparaison pratique des rôles

Aspect	Reviewer	Assignee
Nombre	Peut être multiple	Généralement unique
Focus	Qualité du code	Gestion de la PR
Action principale	Review + Approve/Request changes	Merge/Close
Responsabilité	Technique	Processus

Moment d'intervention	Pendant la review	Tout au long du processus

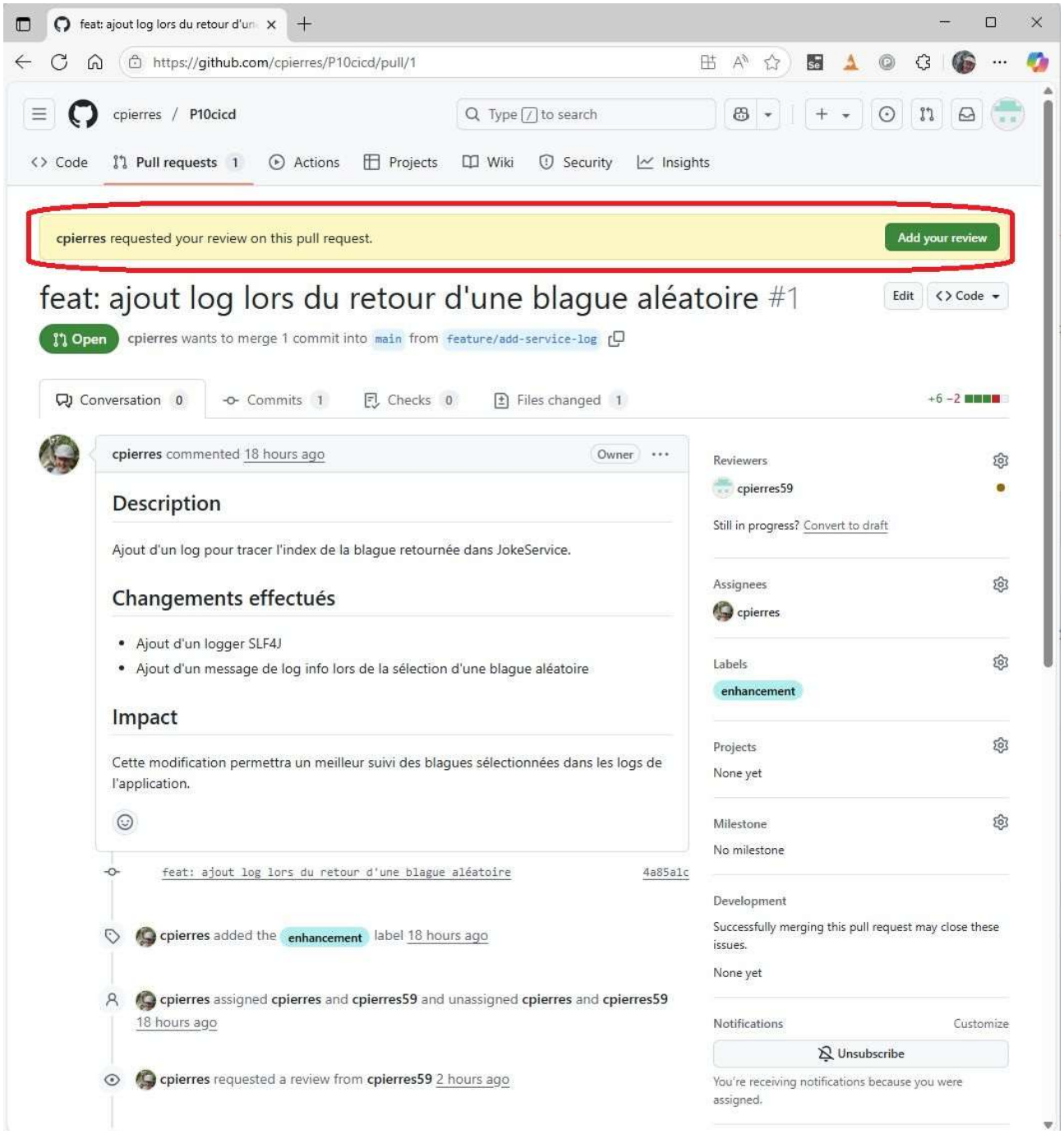
8.2.6. cpierres59 vérifie ses PR dans IntelliJ





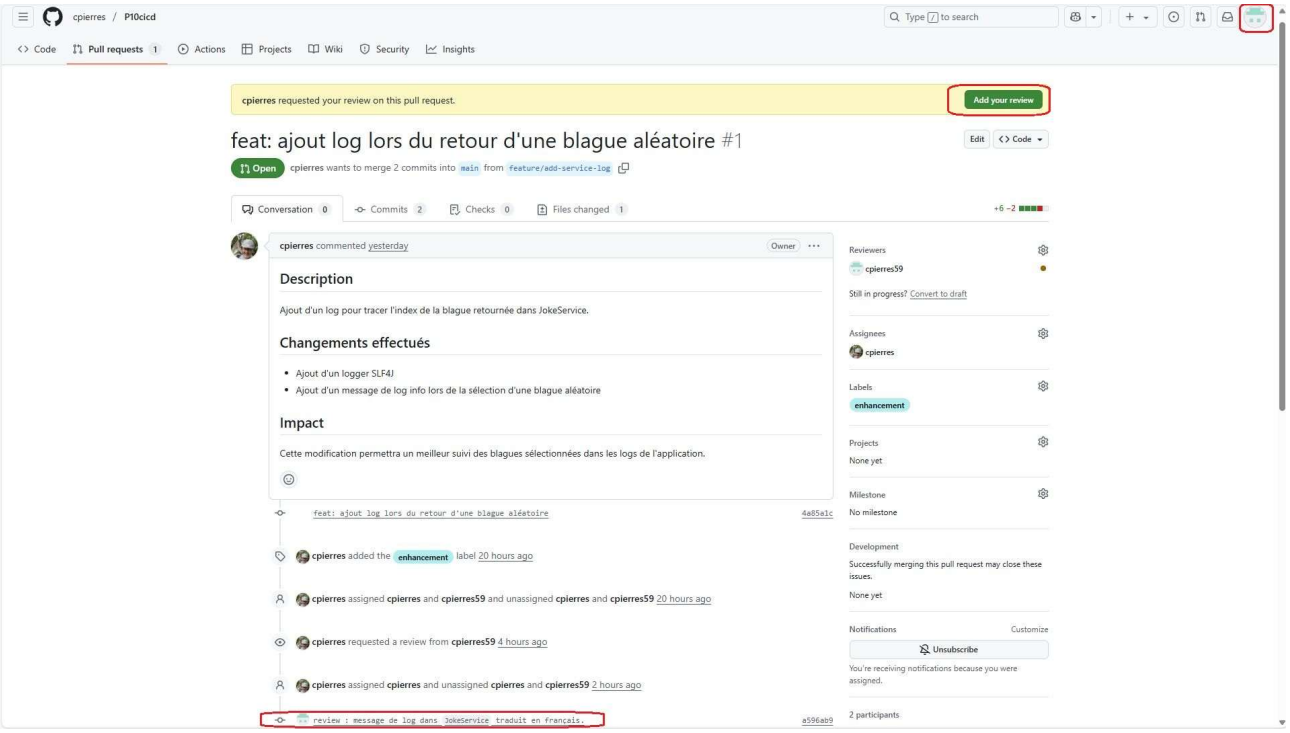
8.2.6.1. Copier l'url de la PR vers le browser connecté sur cpierres59

Ou bien configurer le bon browser pour le projet.

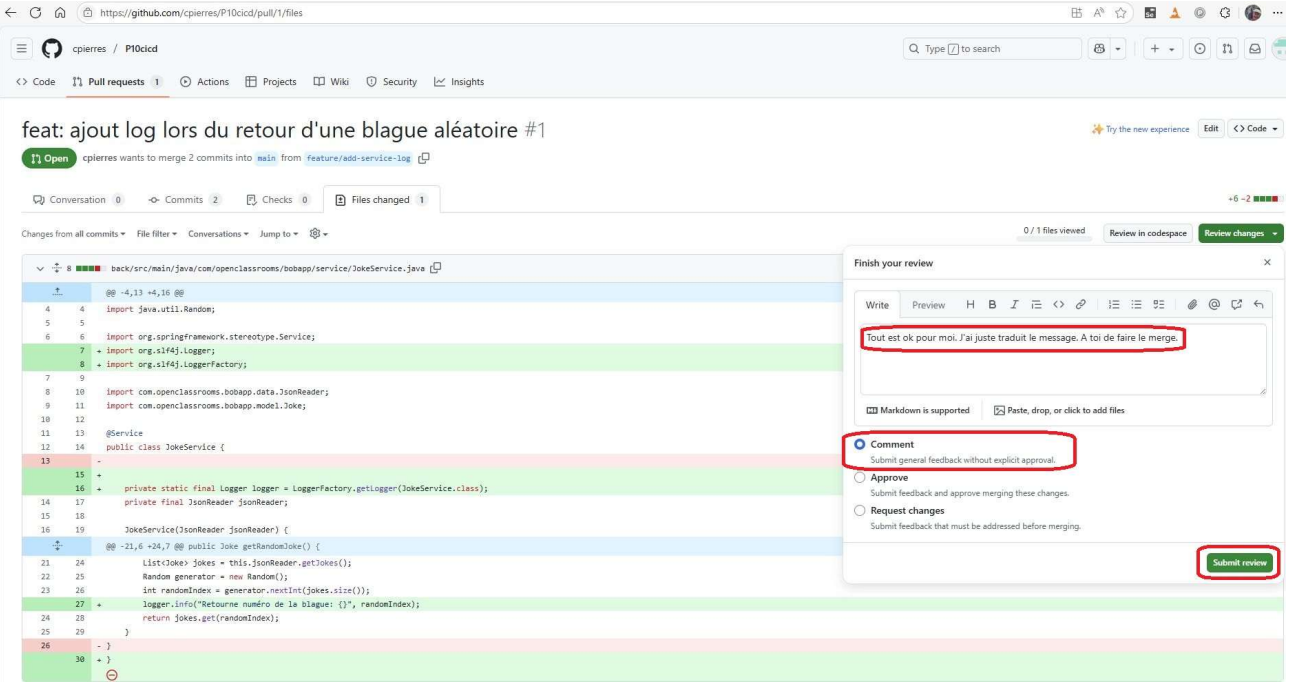


8.2.7. Cpierres59 effectue une modification sur la branche et la pousse

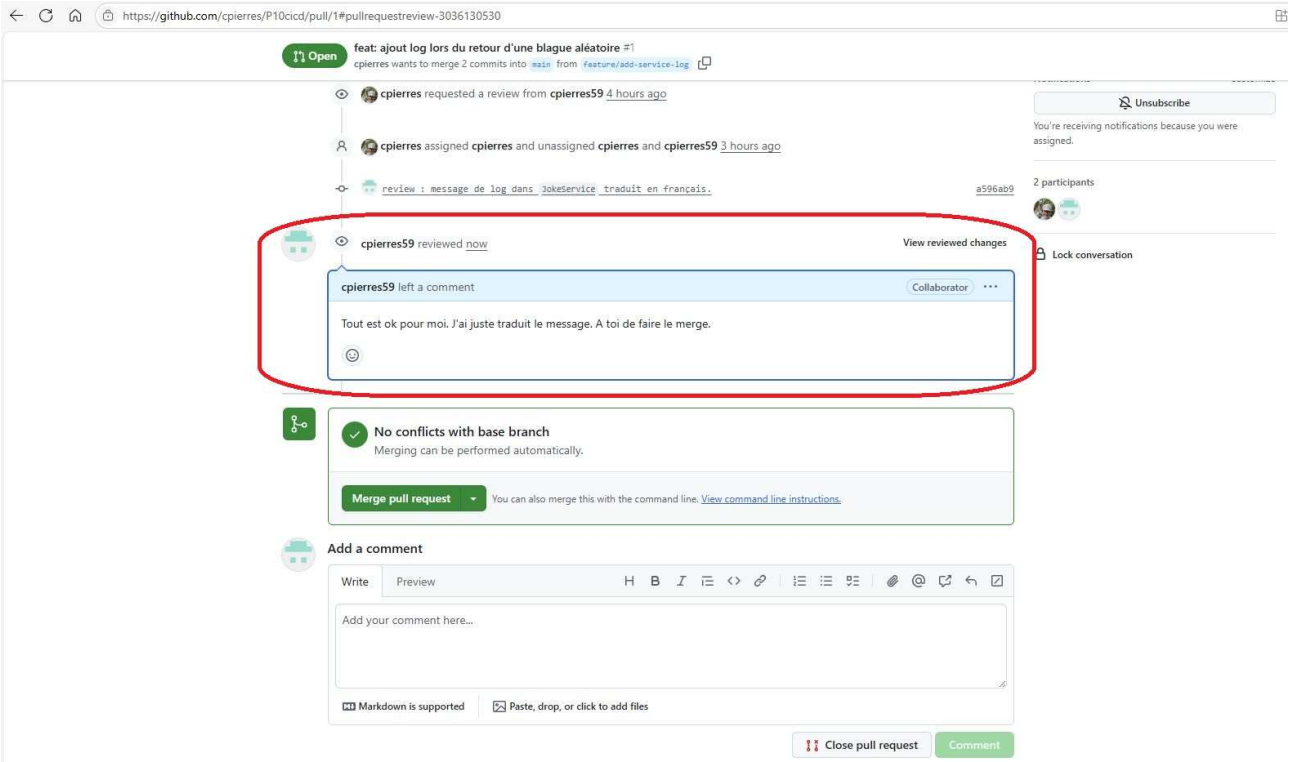
```
remote: Permission to cpierres/P10cicd.git denied to cpierres59.
fatal: unable to access 'https://github.com/cpierres/P10cicd.git/': The requested URL returned error: 403
```



Cpierres59 va ajouter un commentaire pour donner le go à cpierres pour qu'il se charge du contrôle final puis le merge vers main, qui lui revient en tant qu'Assignee. Pour cela, cliquer sur Add your review :

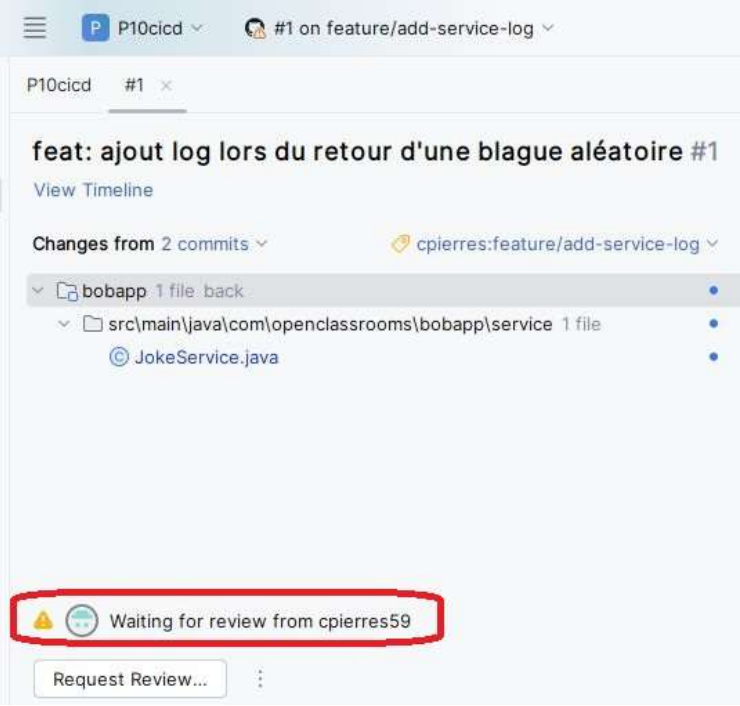


8.2.7.1. Soumettre cette revue :

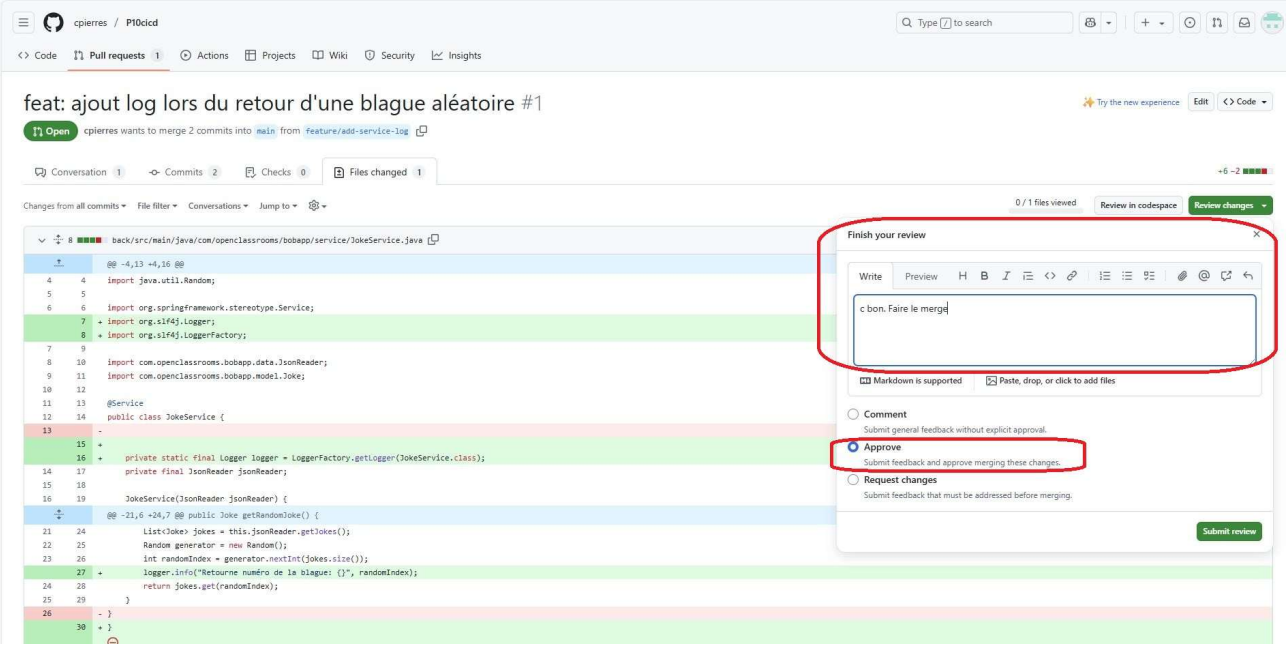


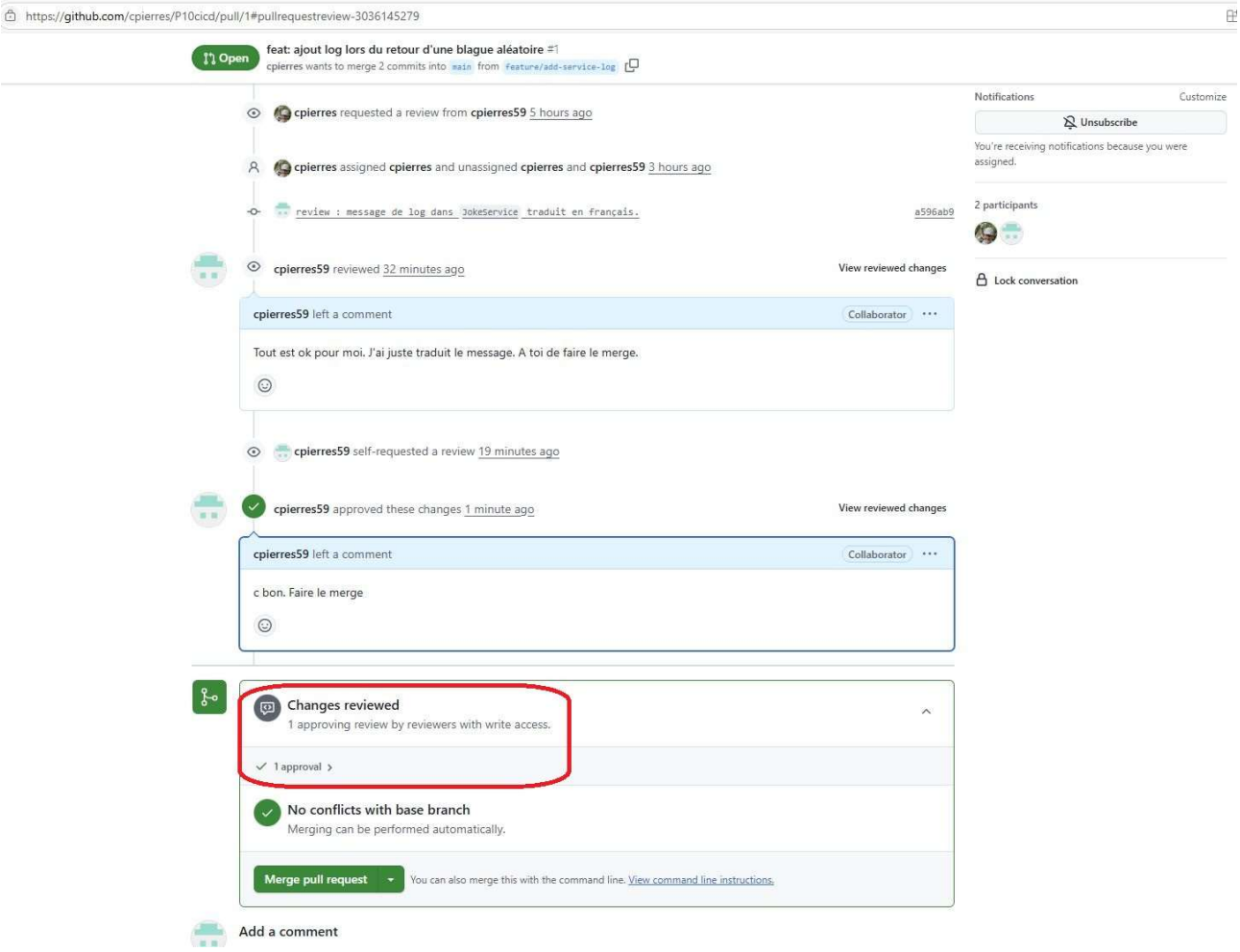
8.2.8. Cpierres reçoit un email l'informant que cpierres59 a inscrit un commentaire dans sa review

8.2.9. Mais pour cpierres, la request view n'est pas terminée par cpierres59 :

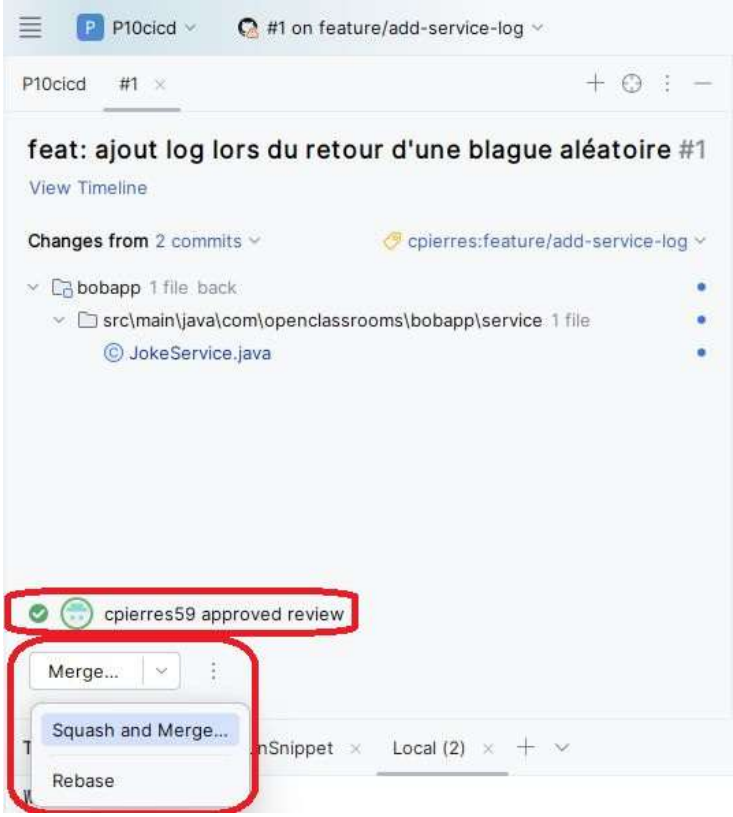


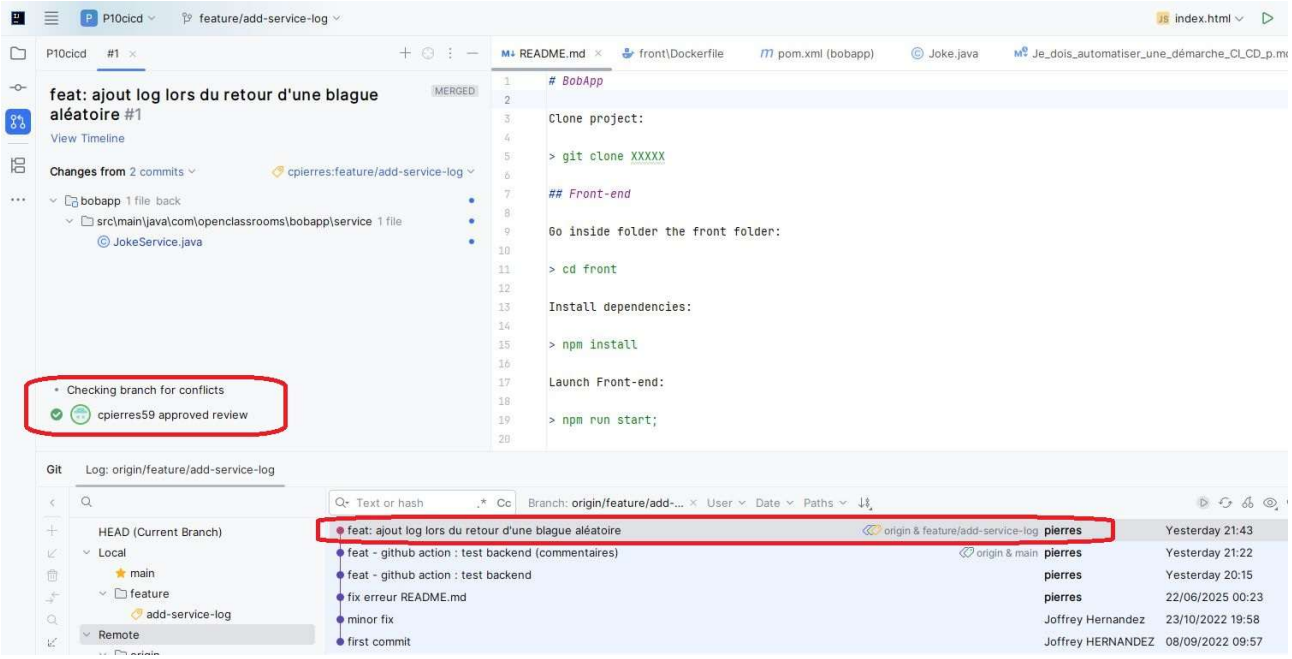
8.2.10. cpierres59 approuve son changement de PR



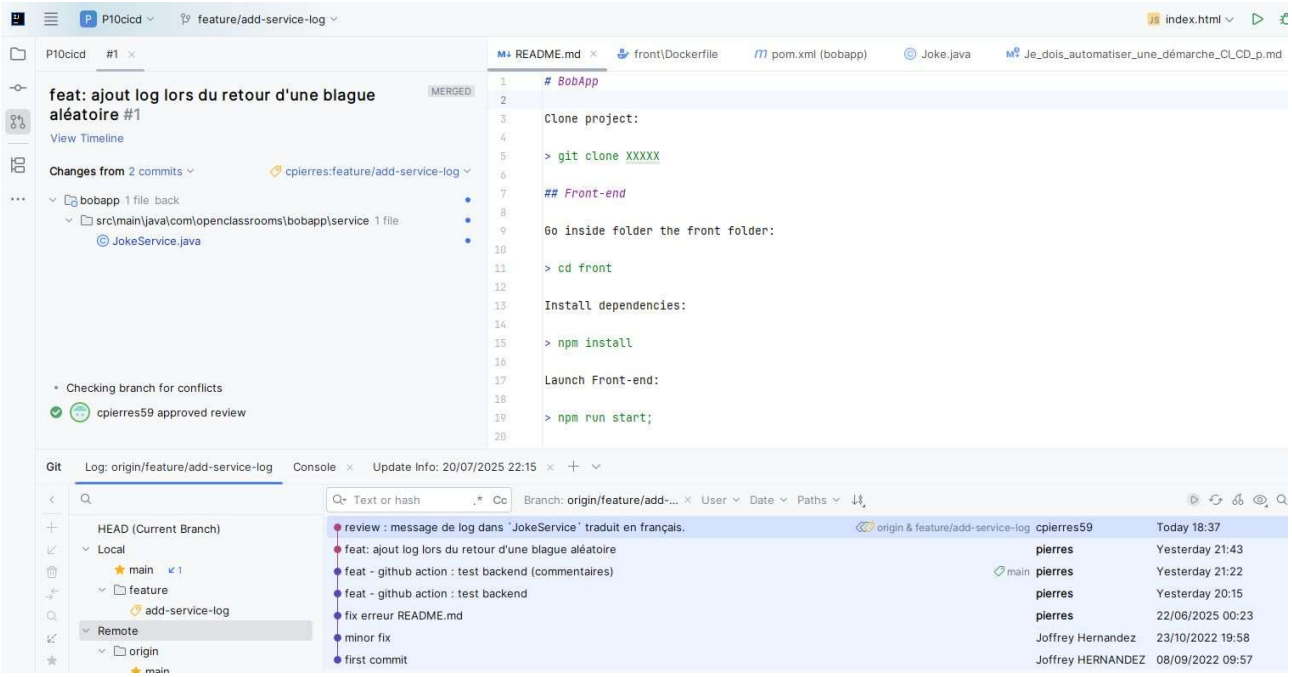


8.2.11. Cpierres voir que la PR est terminée et propose le squash and merge ou rebase



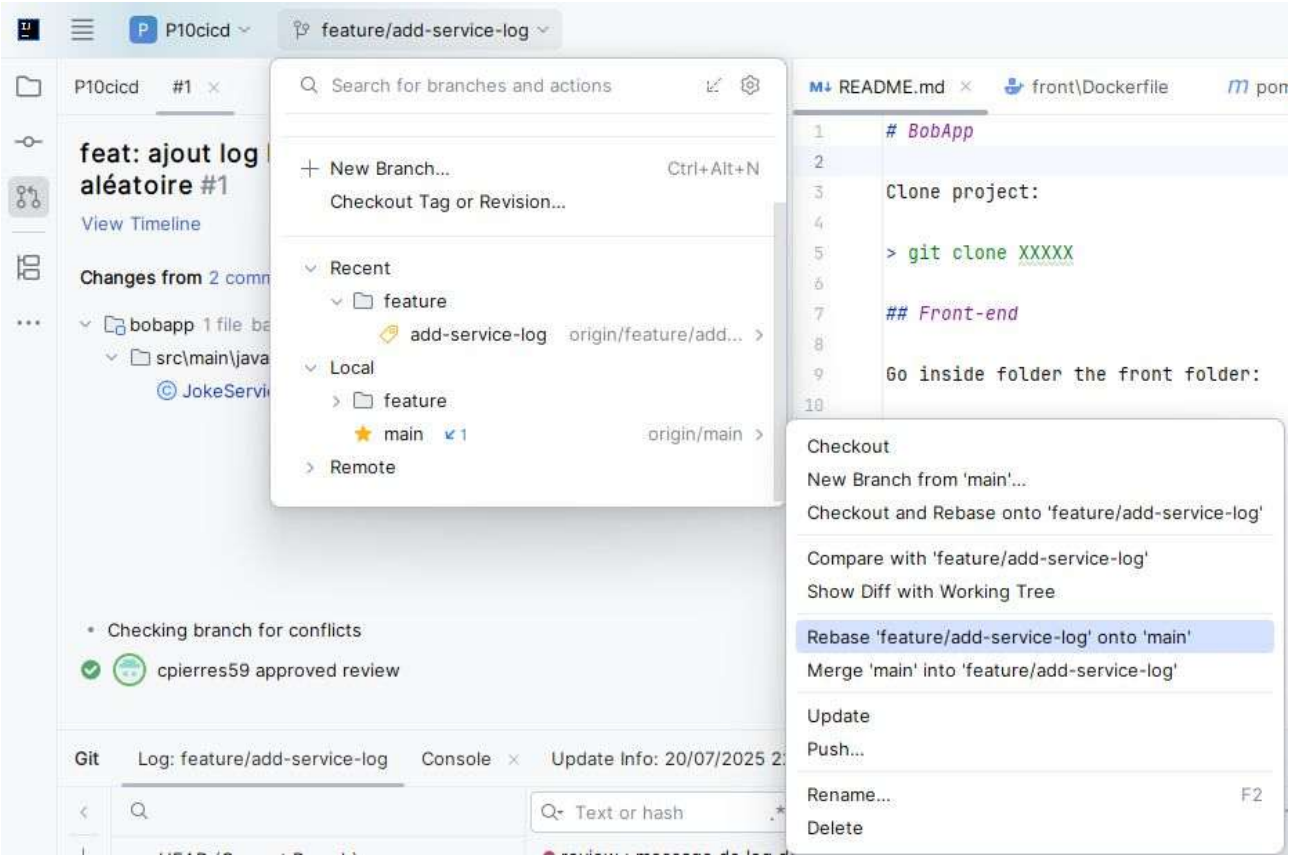


Push de la branche (avec rebase) :



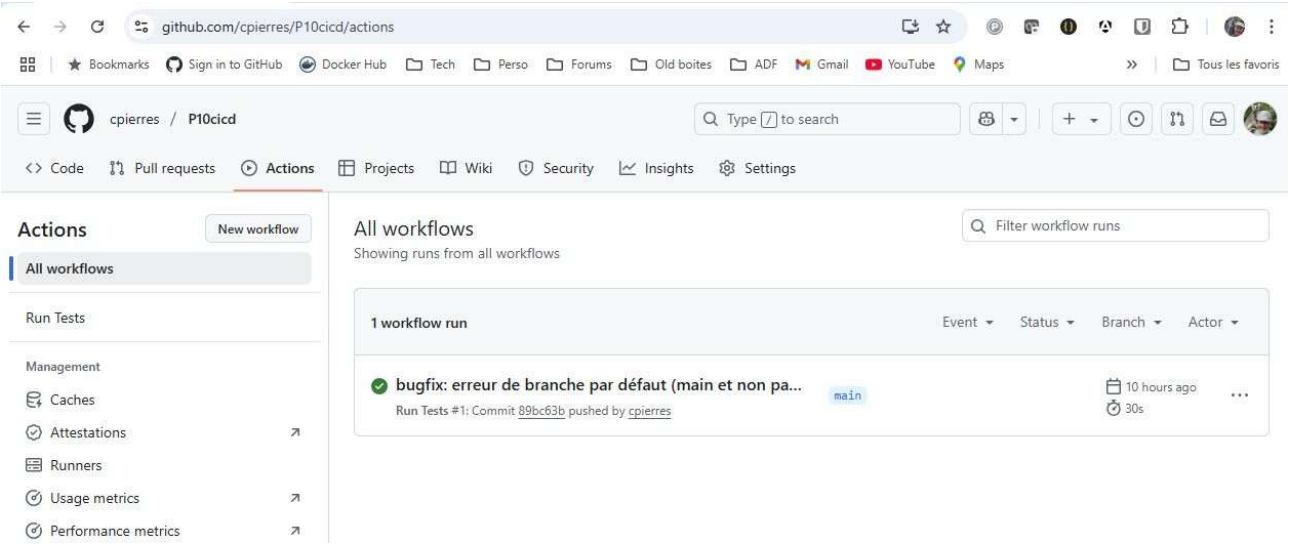
8.2.12. Cpierres rebase de la branche vers main pour clôre la PR

Q-est-ce la bonne démarche ?



8.2.13. Afficher les tests qui se sont exécutés

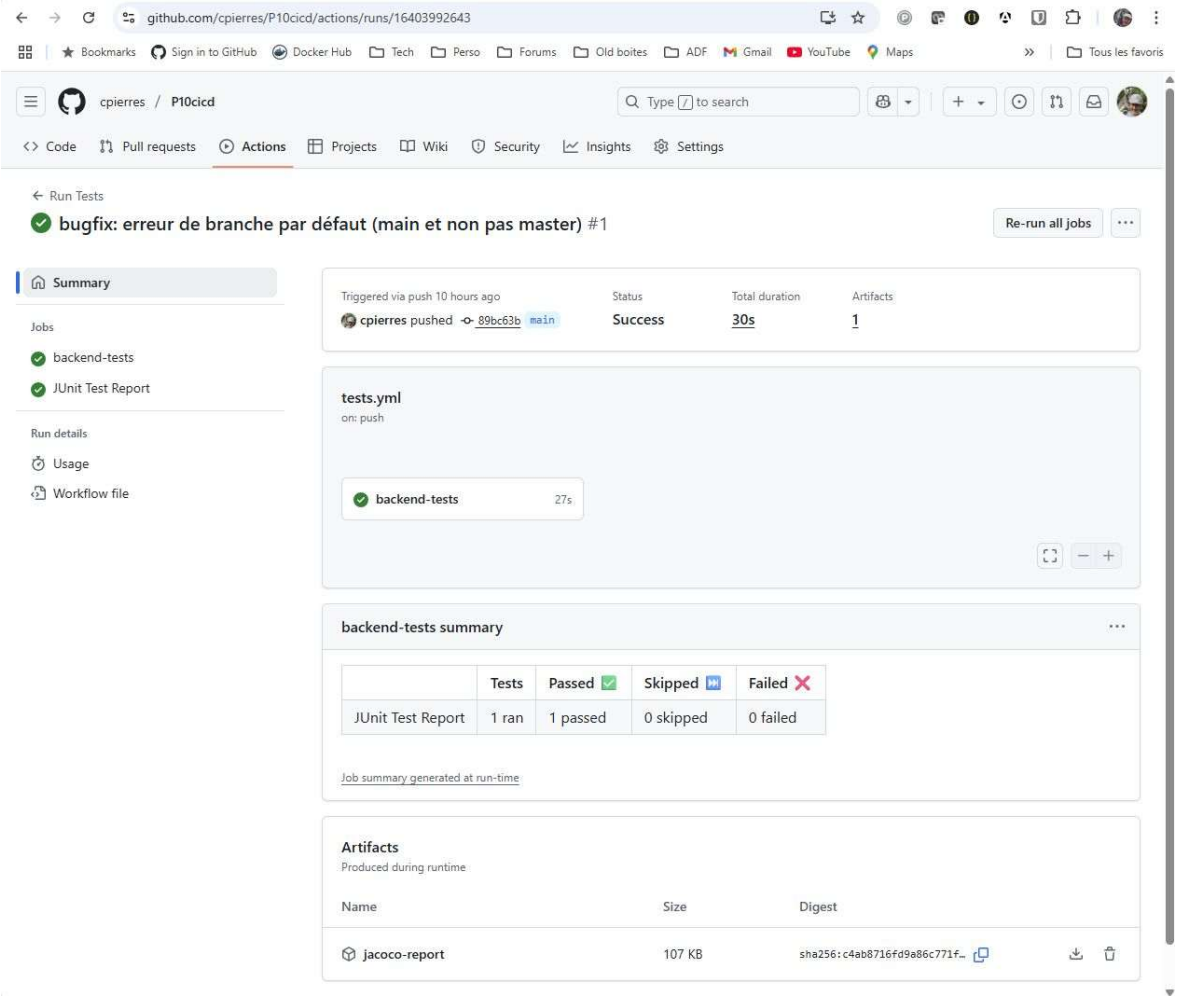
Exécution des tests lors d'un commit :



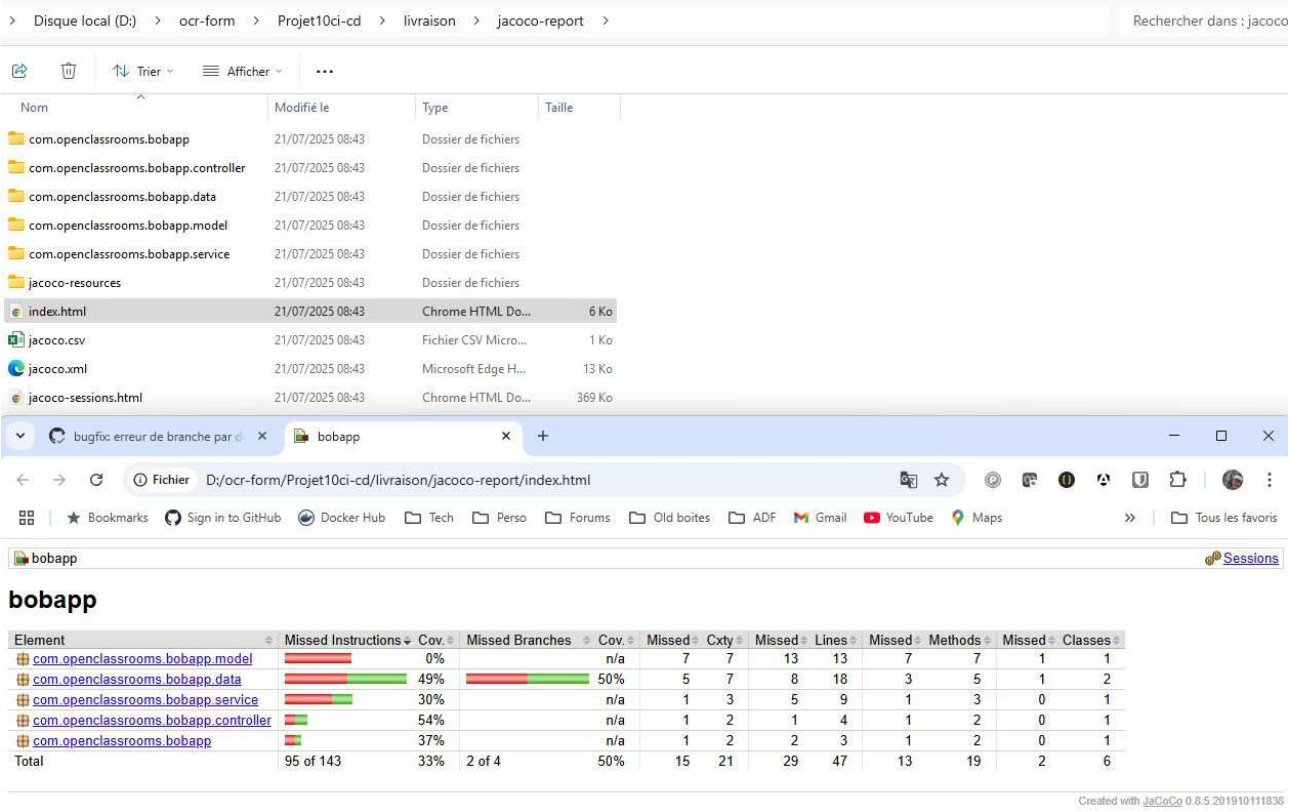
Note : le workflow ne s'est pas déclenché lors de la PR précédente car j'avais fait une erreur dans le nom de la branche par défaut.

8.2.13.1. Détail de l'exécution

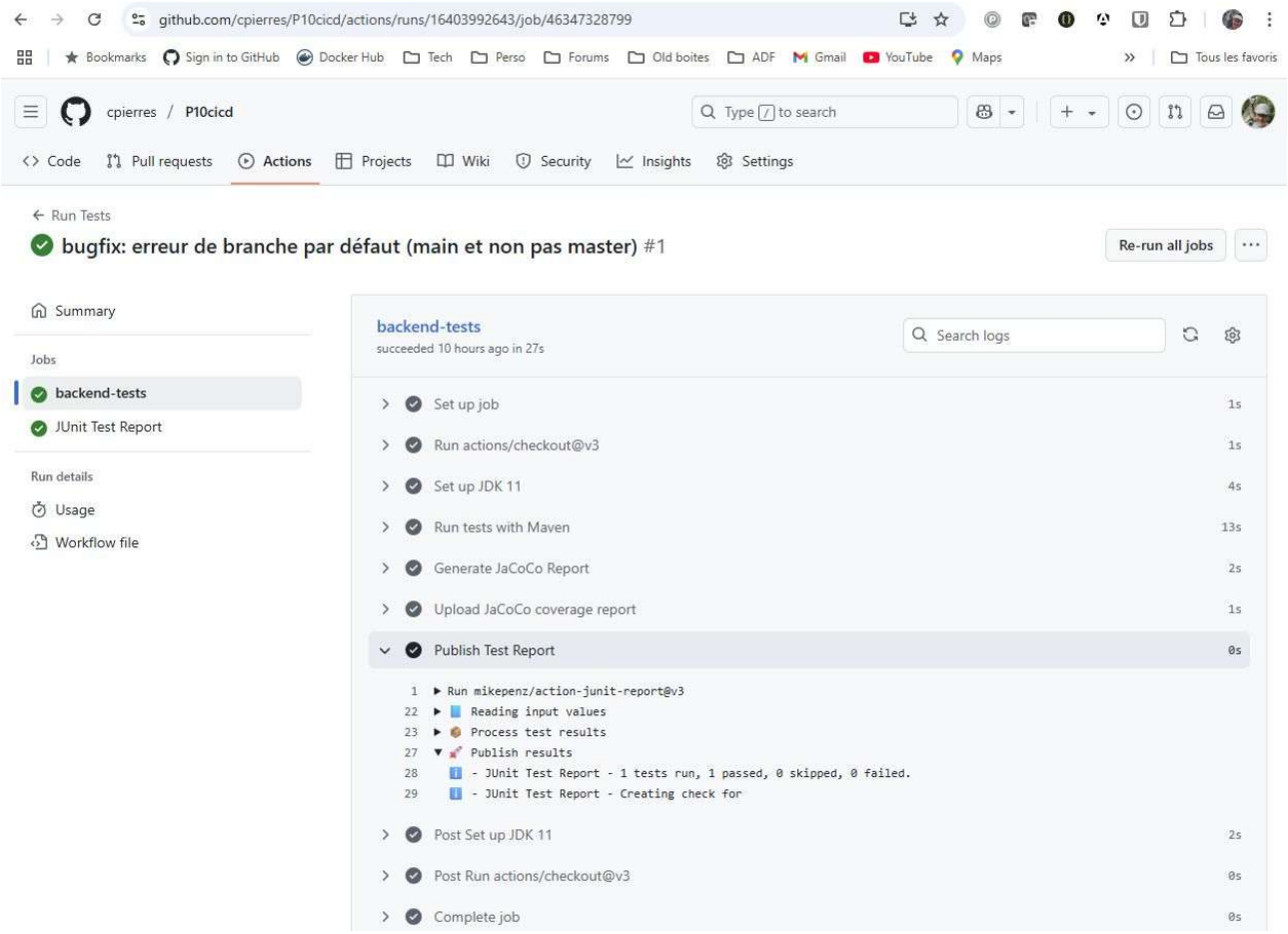
Constater la possibilité de télécharger le résultat jacoco du test, sous forme de zip :



8.2.13.2. Affichage de l'artefact du backend (rapport de couverture dézippé)



En cliquant sur backend-tests, vous affichez le déroulé du test avec les détails qui peuvent être dépliés :



8.3. Build vers dockerhub

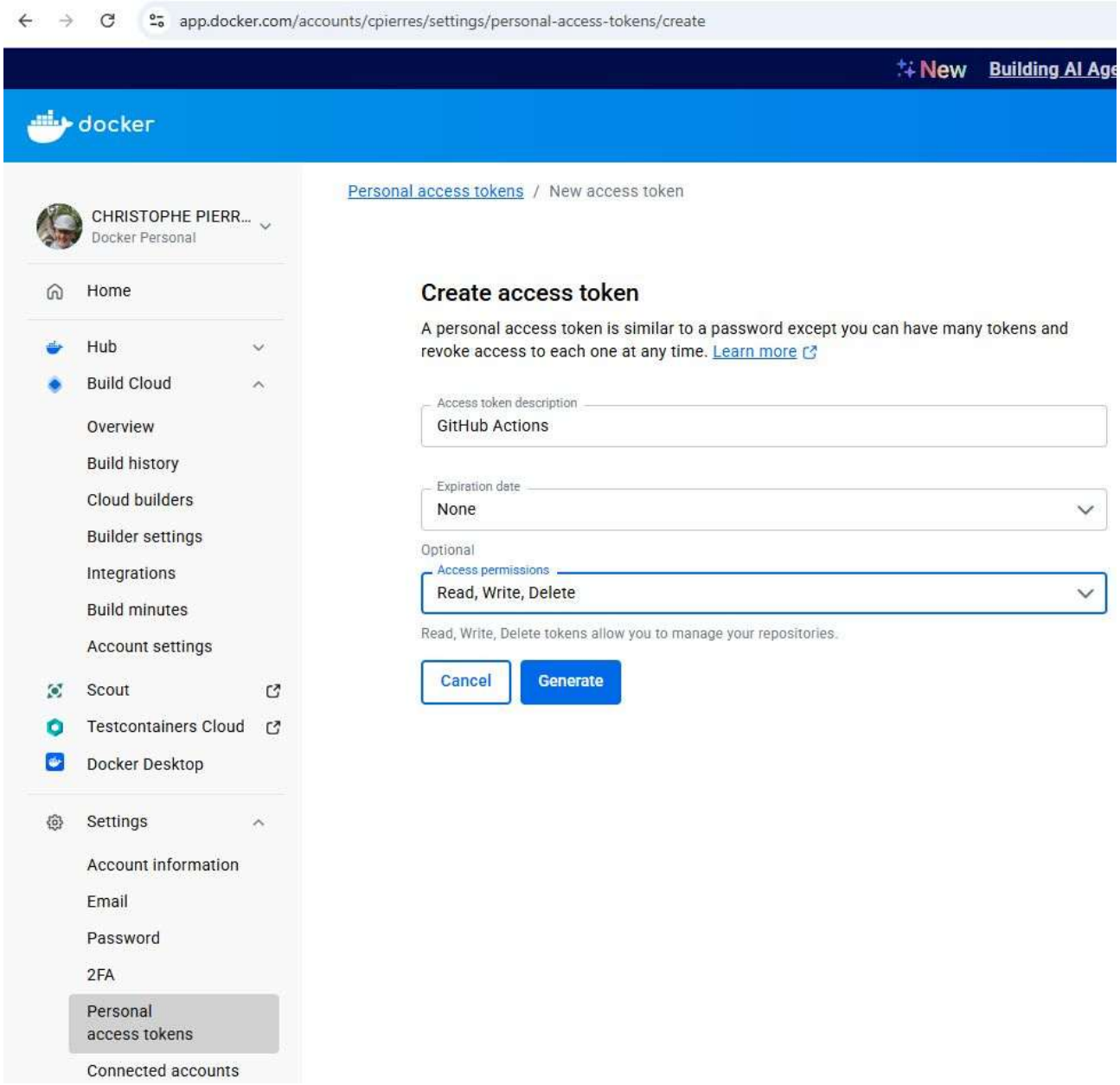
8.3.1. Créer un token dans DockerHub

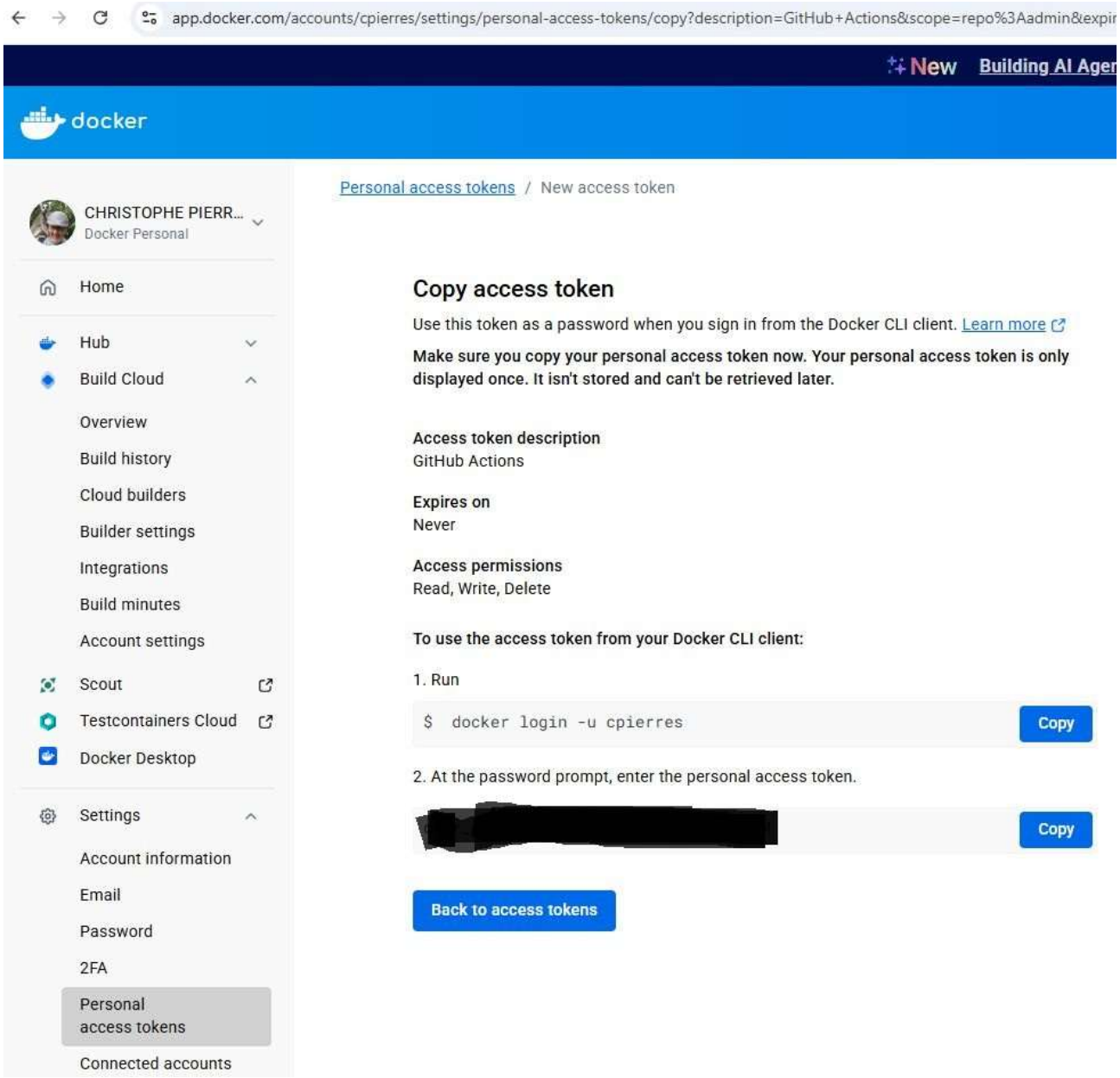
Pour créer un token Docker Hub :

1. Se connecter à [Docker Hub](https://hub.docker.com/)
2. Cliquer sur son avatar puis **Account Settings**
3. Accéder à « Personal access tokens »
4. Cliquer sur « Generate Token"
5. Donner un nom au token (par exemple « GitHub Actions")
6. Sélectionnez les permissions appropriées : Read&Write

7. Copier immédiatement le token généré (vous ne pourrez plus le voir après avoir quitté cette page)

Ce token sera configuré dans GitHub pour permettre aux GitHub Actions de pousser nos images vers DockerHub.





8.3.2. Configurer un secret d'accès à DockerHub dans les paramètres du dépôt GitHub

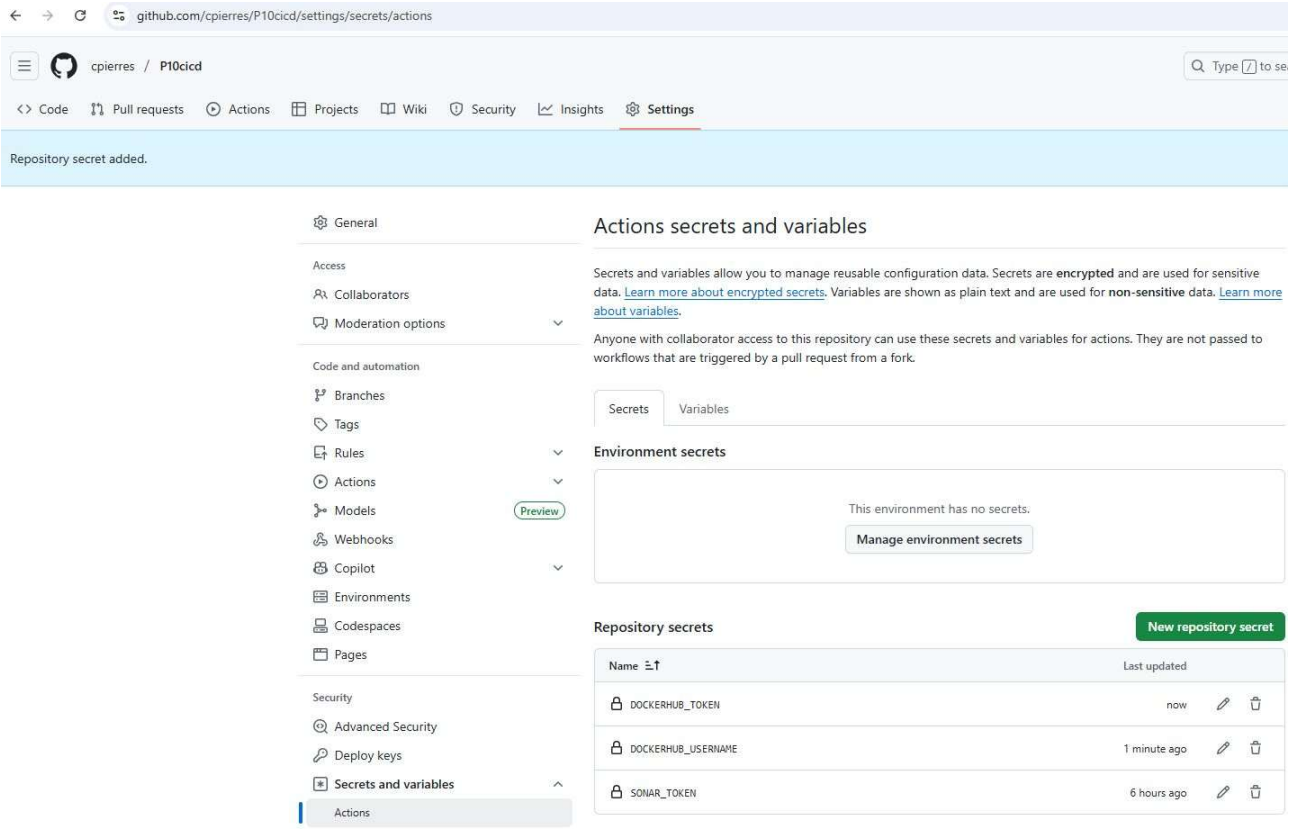
1. Accéder au dépôt GitHub
2. Cliquer sur l'onglet « **Settings** »
3. Dans le menu latéral, cliquez sur **Secrets and variables** puis **Actions**
4. Cliquer sur « New repository secret"
5. Ajouter les deux secrets suivants :

- Nom : `DOCKERHUB_USERNAME`

  Valeur : `nom utilisateur Docker Hub`

- Nom : `DOCKERHUB_TOKEN`

  Valeur : votre token d'accès Docker Hub (pas votre mot de passe)



8.4. *Exemples de Blocker severity Issues et High severity Issues dans SonarQube*

Dans le contexte des Quality Gates SonarQube pour votre projet BobApp, voici des exemples concrets de problèmes bloquants (Blocker severity Issues) et critiques (High severity Issues) pour chaque couche de l'application.

8.4.1. *Couche Backend (Java/Spring Boot)*

8.4.1.1. Blocker severity Issues (Backend)

1. **Injection SQL non paramétrée** :

```
// Exemple de code vulnérable
String query = "SELECT * FROM users WHERE username = '" + userInput + "'";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

Pourquoi c'est bloquant : Permet des attaques par injection SQL qui peuvent compromettre toute la base de données.

2. **Déréférencement certain d'une référence null** :

```
public void processUser(User user) {
    String username = user.getUsername(); // user peut être null
    int length = username.length(); // NullPointerException si user est null
}
```

Pourquoi c'est bloquant : Provoquera un crash de l'application en production.

3. **Fuite de ressources critiques** :

```
public void readFile(String path) throws IOException {
    FileInputStream fis = new FileInputStream(path);
    // Utilisation de fis sans jamais le fermer (pas de try-with-resources)
}
```

Pourquoi c'est bloquant : Peut épuiser les ressources système et causer des défaillances.

4. **Utilisation de méthodes cryptographiques obsolètes** :

```
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] hash = md.digest(password.getBytes());
```

Pourquoi c'est bloquant : MD5 est considéré comme non sécurisé et facilement cassable.

8.4.1.2. High severity Issues (Backend)

1. **Catch d'exception trop générique** :

```
try {
    // Code complexe
} catch (Exception e) {
    // Silence l'exception ou log générique
}
```

Pourquoi c'est critique : Masque potentiellement des problèmes graves et rend le débogage difficile.

2. **Exposition de données sensibles dans les logs** :

```
logger.info("User authentication attempt with password: " + password);
```

Pourquoi c'est critique : Expose des informations sensibles dans les fichiers de logs.

3. **Méthodes avec complexité cyclomatique élevée** :

```
public boolean validateUserInput(String input) {
    // Méthode avec plus de 15 branches conditionnelles
    if (...) {
        if (...) {
            // Imbrication profonde de conditions
        } else if (...) {
            // ...
        }
    } else if (...) {
        // ...
    }
    // etc.
}
```

Pourquoi c'est critique : Code difficile à maintenir, tester et comprendre.

4. **Utilisation de @Autowired sur des champs** (au lieu de l'injection par constructeur) :

```
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;
    // Pas de constructeur avec injection
}
```

Pourquoi c'est critique : Rend le code difficile à tester et crée des dépendances cachées.

8.4.2. Couche Frontend (Angular)

8.4.2.1. Blocker severity Issues (Frontend)

1. **Vulnérabilité XSS (Cross-Site Scripting)** :

```
// Dans un composant Angular
@Component({
  template: `<div [innerHTML]="userContent"></div>`
})
export class UserContentComponent {
```

```
  userContent = '';
  // Affichage direct de contenu utilisateur sans assainissement
}
```

Pourquoi c'est bloquant : Permet l'exécution de scripts malveillants dans le navigateur des utilisateurs.

2. **Utilisation de eval() avec des entrées utilisateur** :

```
function calculateUserFormula(formula: string) {
  return eval(formula); // formula pourrait contenir du code malveillant
}
```

Pourquoi c'est bloquant : Permet l'exécution de code arbitraire fourni par l'utilisateur.

3. **Fuites de mémoire dans les Observables** :

```
export class DataComponent implements OnInit {
  private subscription: Subscription;

  ngOnInit() {
    this.subscription = this.dataService.getData().subscribe(data => {
      // Traitement des données
    });
  }
  // Pas de ngOnDestroy pour désabonner
}
```

Pourquoi c'est bloquant : Provoque des fuites de mémoire qui peuvent dégrader les performances de l'application.

4. **Stockage de secrets en clair dans le code** :

```
const API_KEY = 'a1b2c3d4e5f6g7h8i9j0';

export class ApiService {
  private apiUrl = 'https://api.example.com?key=' + API_KEY;
  // ...
}
```

Pourquoi c'est bloquant : Expose des informations sensibles qui peuvent être extraites du bundle JavaScript.

8.4.2.2. High severity Issues (Frontend)

1. **Utilisation de any excessif** :

```
export class UserService {
  getUser(): any {
    return this.http.get('api/user') as any;
  }

  processData(data: any): any {
    // ...
  }
}
```

Pourquoi c'est critique : Contourne le système de typage TypeScript, augmentant le risque d'erreurs.

2. **Détection de changements inefficace** :

```
@Component({
  template: `<div *ngFor="let item of items; trackBy: trackById">{{item.name}}</div>`
})
export class ListComponent {
  items = [];
  // Pas de fonction trackBy implémentée
}
```

Pourquoi c'est critique : Cause des problèmes de performance lors du rendu de grandes listes.

3. **Appels HTTP sans gestion d'erreur** :

```
loadData() {
```

```
    this.http.get('/api/data').subscribe(
      (data) => { this.data = data; }
      // Pas de gestion d'erreur
    );
  }
```

Pourquoi c'est critique : L'application peut se retrouver dans un état incohérent en cas d'échec de l'API.

4. **Composants surchargés avec trop de responsabilités** :

```
@Component({
  template: `...` // Template très complexe
})
export class MegaComponent {
  // Des dizaines de propriétés et méthodes
  // Gestion de l'état, logique métier, appels API, manipulation DOM, etc.
}
```

Pourquoi c'est critique : Viole le principe de responsabilité unique, rendant le code difficile à maintenir.

8.4.3. Différences clés entre Blocker et High severity Issues

8.4.3.1. Blocker severity Issues

- Représentent des problèmes qui doivent être corrigés immédiatement
- Peuvent causer des crashs, des failles de sécurité exploitables ou des comportements incorrects graves
- Ont généralement un impact direct sur la production
- Justifient l'arrêt du pipeline CI/CD (ERROR dans les Quality Gates)

8.4.3.2. High severity Issues

- Représentent des problèmes sérieux mais légèrement moins urgents
- Affectent la maintenabilité, la performance ou la sécurité de manière significative
- Peuvent ne pas causer de problèmes immédiats mais représentent un risque élevé
- Devraient être corrigés rapidement, mais peuvent parfois tolérer un délai court

Ces exemples illustrent pourquoi il est judicieux de configurer des Quality Gates avec une tolérance zéro pour les Blocker severity Issues dans le nouveau code, tout en permettant une légère tolérance pour les High severity Issues, particulièrement dans le frontend où certains patterns peuvent générer des faux positifs.

8.5. Exemples de Faux Positifs pour des Blocking severity Issues dans Angular

Dans le contexte des Quality Gates SonarQube, nous avons défini une tolérance de 3 problèmes bloquants pour le code global du frontend Angular, contrairement au backend où nous avons une tolérance zéro. Cette différence s'explique par la présence de faux positifs spécifiques à Angular que SonarQube peut signaler comme des problèmes bloquants alors qu'ils sont en réalité des patterns légitimes ou des limitations inhérentes au framework. Voici des exemples concrets :

8.5.1. Utilisation de innerHTML dans les templates Angular

Exemple de code flaggé comme bloquant :

```
@Component({
  template: `<div [innerHTML]="sanitizedContent"></div>`
})
export class ContentComponent {
  sanitizedContent: SafeHtml;

  constructor(private sanitizer: DomSanitizer) {
```

```
    // Le contenu est correctement assaini
    this.sanitizedContent = this.sanitizer.bypassSecurityTrustHtml(someContent);
  }
}
```

Pourquoi c'est un faux positif :

SonarQube détecte l'utilisation de `innerHTML` comme une vulnérabilité XSS bloquante, même lorsque le contenu est correctement assaini via le service `DomSanitizer` d'Angular. Angular dispose de mécanismes intégrés pour gérer ce risque, mais SonarQube ne peut pas toujours détecter que ces mécanismes sont correctement utilisés.

8.5.2. Utilisation de setTimeout ou setInterval sans nettoyage explicite

Exemple de code flaggé comme bloquant :

```
@Component({
  selector: 'app-timer',
  template: '<div>{{counter}}</div>'
})
export class TimerComponent implements OnInit {
  counter = 0;
  private timerId: any;

  ngOnInit() {
    this.timerId = setInterval(() => {
      this.counter++;
    }, 1000);
  }

  ngOnDestroy() {
    clearInterval(this.timerId); // Le nettoyage est bien fait ici
  }
}
```

Pourquoi c'est un faux positif :

SonarQube peut signaler une fuite de ressources bloquante car il ne reconnaît pas toujours que le nettoyage est effectué dans la méthode `ngOnDestroy`. Il peut également ne pas comprendre le cycle de vie des composants Angular et comment ils sont détruits.

8.5.3. Accès direct au DOM via ElementRef

Exemple de code flaggé comme bloquant :

```
@Component({
  selector: 'app-focus',
  template: '<input #inputField>'
})
export class FocusComponent implements AfterViewInit {
  @ViewChild('inputField') inputElement: ElementRef;

  constructor(private el: ElementRef) {}

  ngAfterViewInit() {
    // Manipulation directe du DOM, parfois nécessaire
    this.inputElement.nativeElement.focus();
  }
}
```

Pourquoi c'est un faux positif :

SonarQube considère souvent l'accès direct au DOM comme une pratique dangereuse (ce qui est généralement vrai), mais dans Angular, certaines manipulations DOM via `ElementRef` sont parfois nécessaires et sécurisées dans le contexte du framework. Angular s'assure que ces manipulations sont effectuées dans un environnement contrôlé.

8.5.4. Utilisation de eval() dans les expressions dynamiques

Exemple de code flaggé comme bloquant :

```
// Dans un service qui utilise une bibliothèque d'expressions mathématiques
@Injectable()
export class CalculatorService {
  evaluateExpression(expression: string): number {
    // Utilisation d'une bibliothèque sécurisée qui utilise eval() en interne
    return mathExpressionLibrary.calculate(expression);
  }
}
```

Pourquoi c'est un faux positif :

Si une bibliothèque tierce utilise `eval()` en interne mais de manière sécurisée (avec validation et assainissement), SonarQube peut toujours le signaler comme un problème bloquant. Dans certains cas, comme les calculatrices ou les évaluateurs d'expressions, l'utilisation contrôlée d'`eval()` peut être légitime.

8.5.5. Références circulaires dans les services Angular

Exemple de code flaggé comme bloquant :

```
@Injectable({
  providedIn: 'root'
})
export class ServiceA {
  constructor(private serviceB: ServiceB) {}
}

@Injectable({
  providedIn: 'root'
})
export class ServiceB {
  constructor(private serviceA: ServiceA) {}
}
```

Pourquoi c'est un faux positif :

SonarQube peut signaler cette dépendance circulaire comme un problème bloquant, mais Angular a des mécanismes pour gérer ce cas (comme l'injection forward). Dans certaines architectures complexes, ces dépendances circulaires peuvent être inévitables et gérées correctement.

8.5.6. Pourquoi une tolérance est nécessaire pour le frontend

Ces faux positifs justifient une tolérance plus élevée pour les problèmes bloquants dans le code frontend pour plusieurs raisons :

1. **Patterns spécifiques au framework** : Angular utilise des patterns qui peuvent sembler risqués pour des analyseurs statiques génériques mais sont sécurisés dans le contexte du framework.

2. **Bibliothèques tierces** : Les applications Angular dépendent souvent de bibliothèques tierces qui peuvent contenir des patterns flaggés comme problématiques par SonarQube.

3. **Limitations de l'analyse statique** : SonarQube a des limitations dans sa compréhension du cycle de vie et des mécanismes de sécurité d'Angular.

4. **Compromis UX/sécurité** : Certains patterns considérés comme risqués peuvent être nécessaires pour offrir une expérience utilisateur optimale.

Cette tolérance de 3 problèmes bloquants pour le code global du frontend permet à l'équipe de développement de maintenir un équilibre entre la qualité du code et la flexibilité nécessaire pour développer des applications Angular modernes et performantes.