ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES

Rapport d'analyse de qualité de code SonarQube.

Introduction

Ce rapport présente les résultats de l'analyse de qualité de code réalisée avec SonarQube. L'objectif principal est d'identifier les problèmes critiques, les vulnérabilités potentielles, et les aspects du code qui pourraient être améliorés afin d'assurer un code robuste, sécurisé et maintenable. Le projet analysé est :

Application de gestion de projet

Présentation du projet

Le projet analysé est une application développée avec **Spring Boot**, un framework Java largement utilisé pour créer des applications backend robustes et évolutives.

- **Nom du projet** : application de gestion de projet
- **Technologies utilisées** :
  - **Langage** : Java
  - **Framework** : Spring Boot
  - **Base de données** : MySQL
  - **Tests unitaires** : JUnit 5

Objectif de l'analyse

L'analyse vise à évaluer la **qualité du code** en utilisant **SonarQube**, un outil de référence pour identifier :

- **Bugs** : Détecter les erreurs critiques qui pourraient provoquer des dysfonctionnements.
- **Vulnérabilités** : Identifier des failles de sécurité potentielles.
- **Code Smells** : Repérer des mauvaises pratiques de codage.
- **Couverture de code** : Mesurer l'efficacité des tests grâce à JaCoCo.

Outils utilises

1. **SonarQube** : Analyse statique du code et suivi des métriques de qualité.
2. **JaCoCo** : Génération de rapports de couverture de tests unitaires.
3. **Maven** : Gestionnaire de build pour exécuter les tests et l'analyse.

Structure du rapport

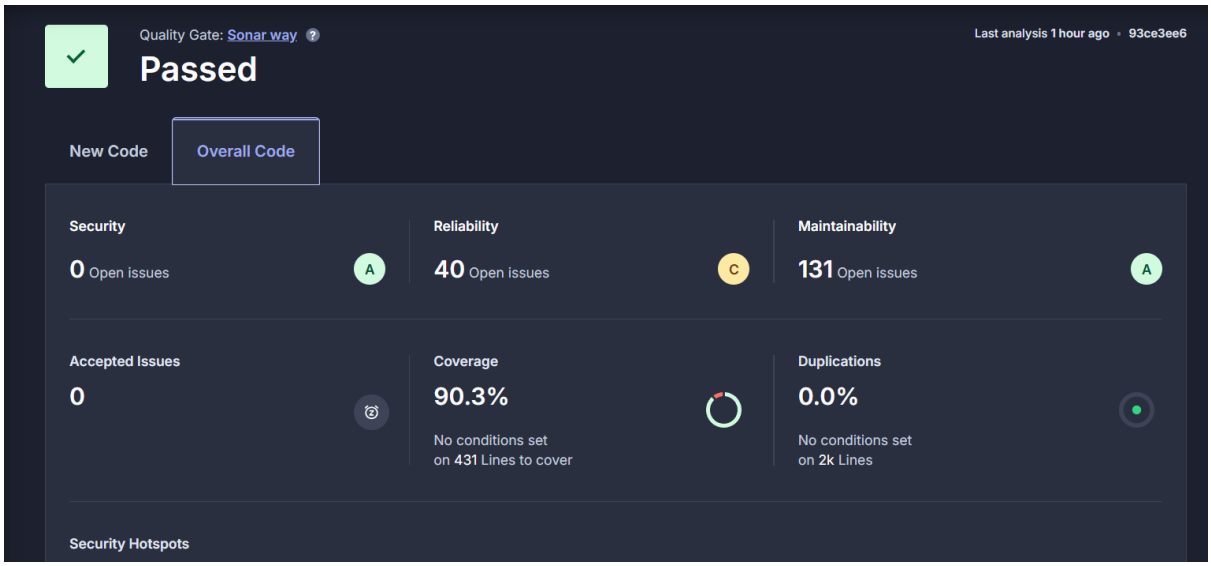
Le rapport est structuré comme suit :

1. Résultats de l'analyse statique du code.
2. Couverture des tests unitaires.
3. Recommandations pour améliorer la qualité du code.

Résultats de l'analyse statique du code.

Résultats de l'analyse statique du code

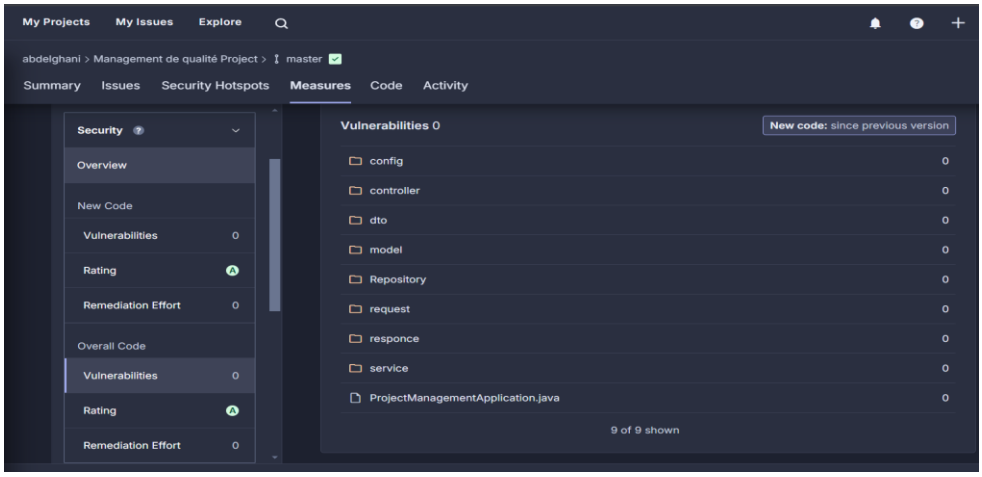
L’analyse statique du code a été réalisée en utilisant **SonarQube**, qui a permis d’évaluer différents aspects de la qualité du code source de notre projet **Spring Boot**. Cette analyse a permis d’identifier plusieurs problèmes .



Vulnérabilités

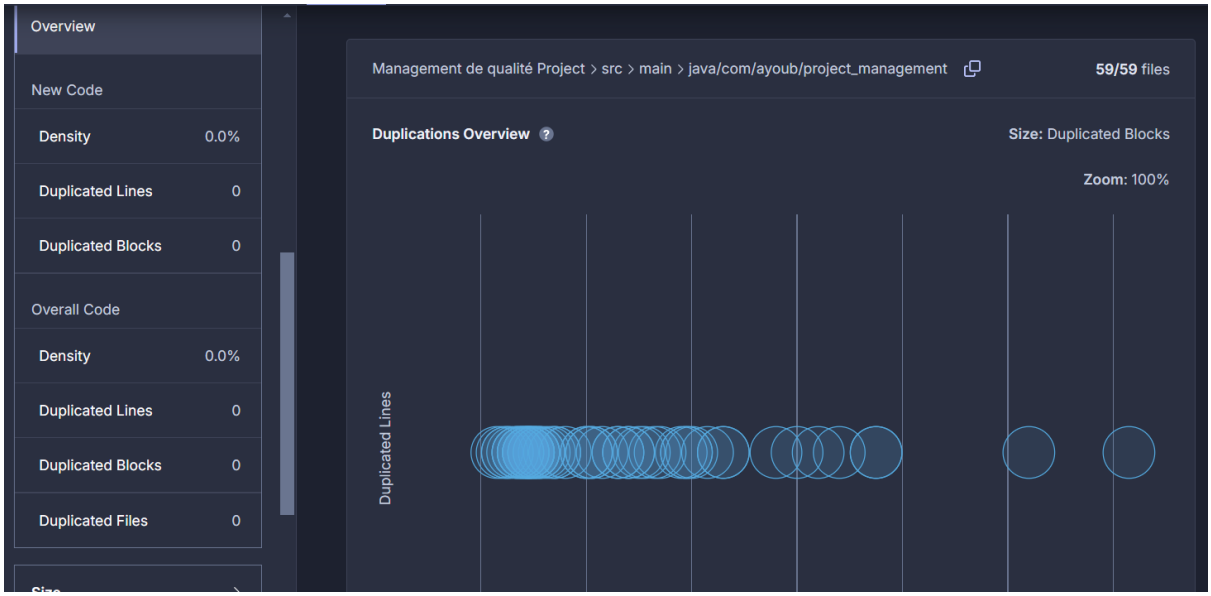
Les **vulnérabilités** sont des failles de sécurité dans le code qui pourraient être exploitées par un attaquant. L’analyse a permis d’identifier des vulnérabilités potentielles dans l’application. Ces vulnérabilités doivent être corrigées pour assurer la sécurité de l’application.

Nombre total de vulnérabilités : 0



Duplications

Les duplications de code désignent les segments identiques ou très similaires dans différentes parties du projet. Les duplications augmentent la complexité, rendent le code plus difficile à maintenir, et augmentent les risques d'erreurs lors des modifications. SonarQube permet de détecter et de mesurer ces duplications. **Pourcentage de duplication : 0%**

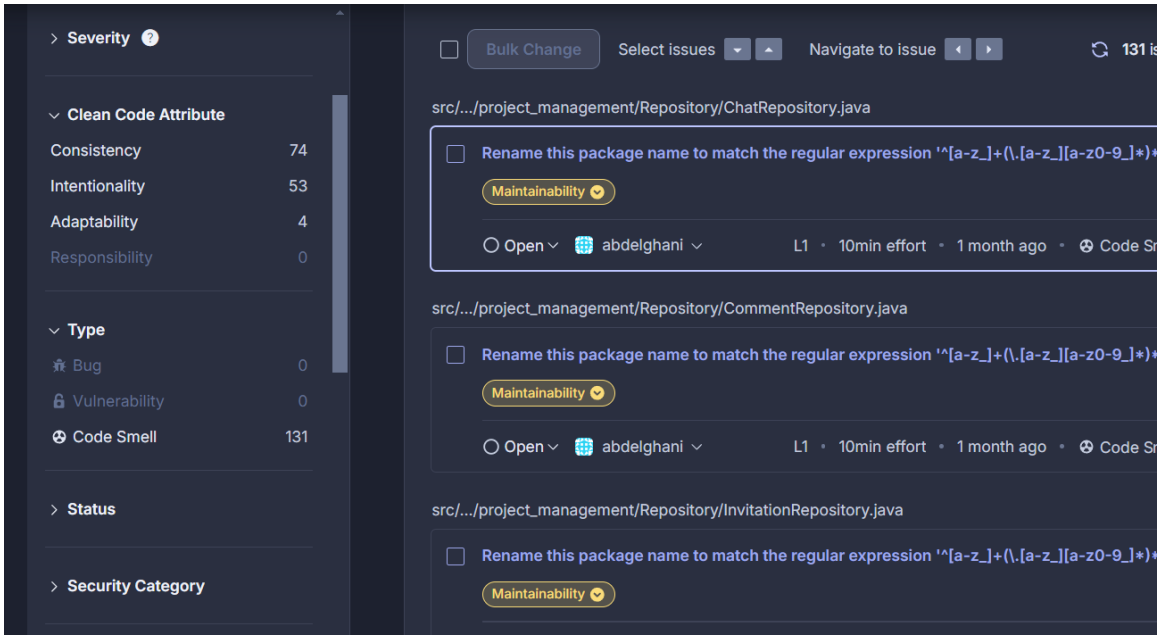


Code Smells

Les **Code Smells** sont des mauvaises pratiques ou des problèmes dans le code qui ne sont pas nécessairement des bugs, mais qui peuvent nuire à la lisibilité, la maintenance et la performance de l'application. L'analyse a détecté plusieurs problèmes de ce type.

Nombre total de Code Smells :131

Nombre total de bugs :0

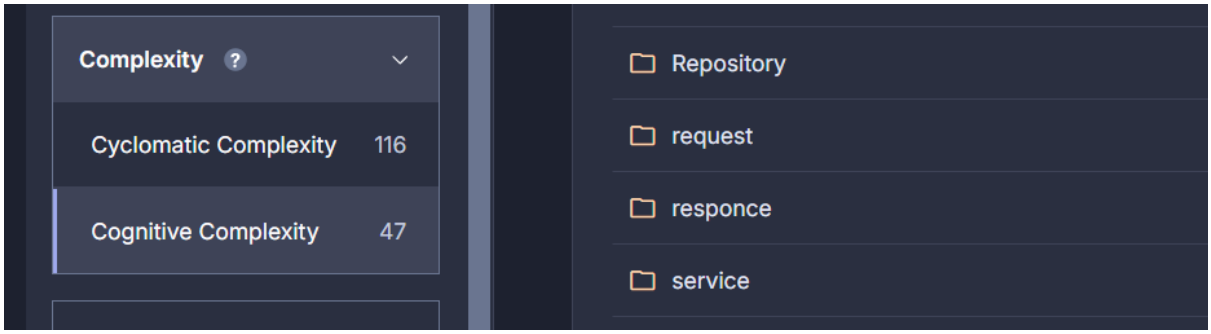


Complexité Cyclomatique : La Complexité Cyclomatique mesure le nombre de chemins indépendants dans un morceau de code (par exemple, à travers les conditions if, switch, for, etc.). Une complexité élevée signifie que le code est plus difficile à tester et à maintenir.

Score total de Complexité Cyclomatique : 116

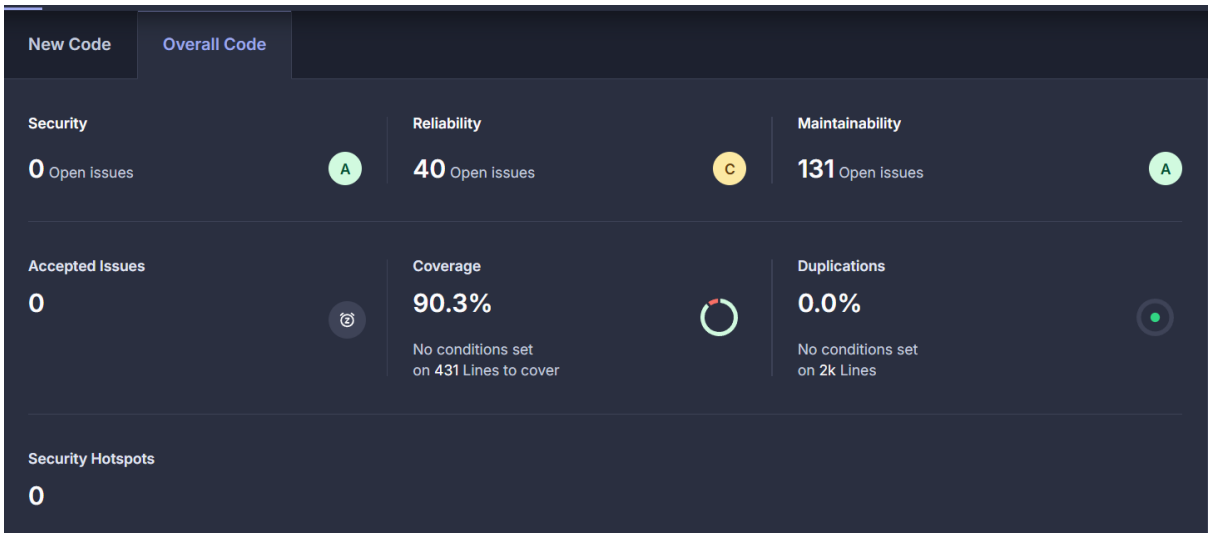
Complexité Cognitive : La Complexité Cognitive mesure à quel point le code est difficile à comprendre par un développeur humain. Contrairement à la complexité cyclomatique, elle tient compte de la lisibilité et des structures imbriquées.

Score total de Complexité Cognitive : **47**



Couverture des tests unitaires

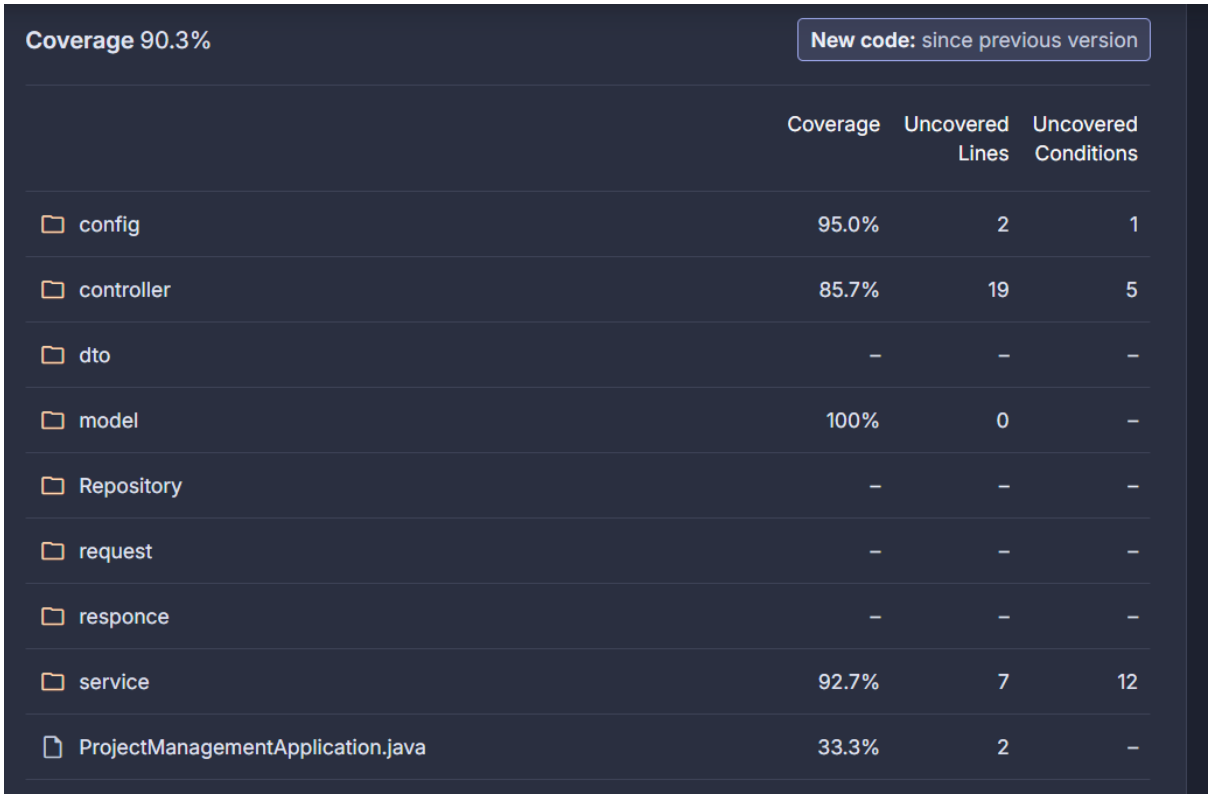
La couverture des tests unitaires pour le projet a été mesurée à l'aide de JaCoCo et analysée dans SonarQube. Avec une couverture globale de **90.3%**, le projet démontre une très bonne qualité de tests. Cependant, certaines parties nécessitent encore des améliorations.



Couverture globale des lignes : 90,3%

Packages avec des lacunes :

- `controller` : 85.7% de couverture, avec **19 lignes** et **5 conditions** non couvertes.
- `service` : 92.7% de couverture, avec **7 lignes** et **12 conditions** non couvertes.
- `ProjectManagementApplication.java` : 33.3% de couverture, nécessitant une attention particulière.



Conclusion

Avec une couverture de **90.3%**, le projet atteint un bon niveau de qualité. Des efforts ciblés sur les classes `controller`, `service`, et `ProjectManagementApplication.java` permettront d'atteindre une couverture encore meilleure, assurant une robustesse et une maintenabilité accrues.