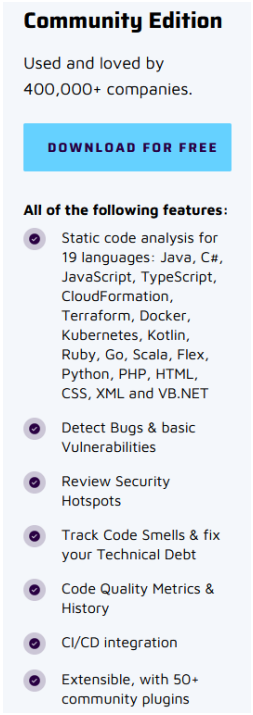
Metriken – SonarQube

von Florian Goldbach

SonarQube

SonarQube installieren

Zuerst downloaden wir die „SonarQube Community Edition“ von der offiziellen SonarQube Website: https://www.sonarsource.com/products/sonarqube/downloads/



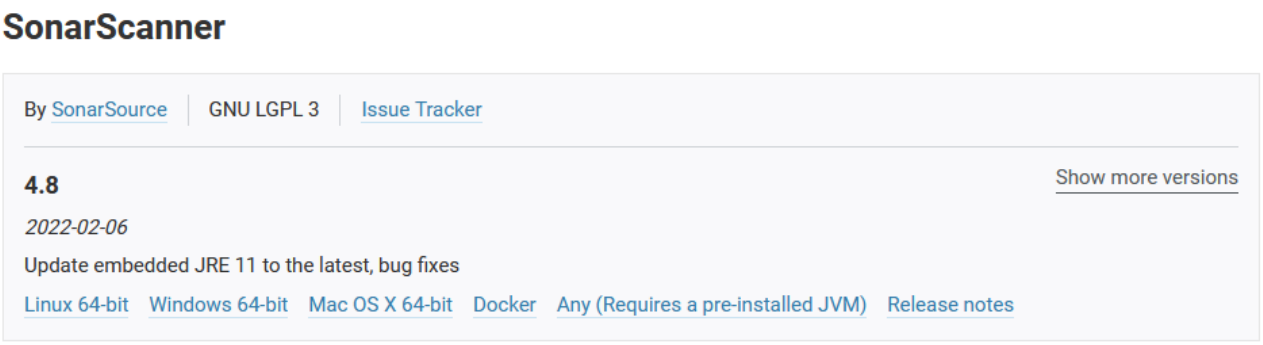
Wir extrahieren das zip file, bewegen es zu dem Ordner unserer Wahl und starten SonarQube (unterschiedlich, je nach Betriebssystem). Für Windows führen wir die StartSonar.bat-Datei in ..bin\windows-x86-xx aus.

Um SonarQube erfolgreich auszuführen benötigt man jdk-17. Ich hatte jdk-19 installiert und habe an dieser Stelle ca. 1 Stunde aufgewendet, damit mein System die jdk-17-Installation als Standard benutzt (Es war eine „versteckte“ Umgebungsvariable).

Sobald man den SonarQube Server mit StartSonar.bat gestartet hat, kann man nun über „http://localhost:9000“ im Webbrowser der Wahl auf das SonarQube interface zugreifen. Dort kann man sich mit „Username: admin“ und „Password: admin“ einloggen.

SonarQube Scanner installieren

Den SonarQube Scanner können wir hier herunterladen: https://docs.sonarqube.org/latest/analyzing-source-code/scanners/sonarscanner/



Wir extrahieren das zip-file an den Ort unserer Wahl und müssen als nächtes das ..\sonar-scanner\bin-Verzeichnis in unserer Path-Umgebungsvariable hinzufügen.

SonarQube Scanner konfigurieren

Im „root directory“ unseres Projektes, welches wir mit SonarQube analysieren möchen, erstellen wir eine Datei mit dem Namen: „**sonar-project.properties**“.

Ich werde hier das Beispielprojekt JUnitFeatureExploration (aus der letzten Einsendeaufgabe) benutzen und in einem Repository „SonarQubeFeatureExploration“ arbeiten. Link zum Repository: https://github.com/Florgol/SonarQubeFeatureExploration

Ich habe mich entschieden den „Test Coverage“ von Jacoco ebenfalls in SonarQube anzeigen zu lassen. SonarQube benötigt dafür einen Bericht im XML-Format. Diesen konnte ich hier in meinem build.gradle-file hinzufügen:

```
jacoco {
    toolVersion = "0.8.7"
}

jacocoTestReport {
    reports {
        xml.required.set(true)
        html.required.set(true)
    }
}
```

Für die richtige jacoco-Konfiguration habe ich ca. 1 Stunde aufgewendet. Doch nun habe ich den Bericht im XML-Format und kann mir diesen hoffentlich in SonarQube anzeigen lassen.

Hier die Konfiguration der **sonar-project.properties** Datei:

```
sonar.projectKey=JUnitFeatureExploration
sonar.projectName=JUnitFeatureExploration
sonar.projectVersion=1.0
sonar.sources=src/main
sonar.tests=src/test
sonar.language=java
sonar.coverage.jacoco.xmlReportPaths=build/reports/jacoco/test/jacocoTestReport.xml
```

„projectKey“ steht für einen einzigartigen Identifikator (einzigartig im scope „alle Projekte auf dem Server“)
Das Projekt wird im Interface mit „projectName“ angezeigt
„projektVersion“ – Man kann sich selbst eine Projektversionsnummer aussuchen
„sources“ – Alle Klassen des Projektes
„tests“ – Test-Klassen
„language“ – Projekt-Programmiersprache (SonarQube unterstützt über 20 verschiedene)
„coverage.jacoco.xmlReportPaths“ – der Test Coverage Bericht im XML-Format

Anmerkung 1: Man muss ebenfalls einen Token im SonerQube-Account generieren und diesen in der Konfigurationsdatei vermerken.

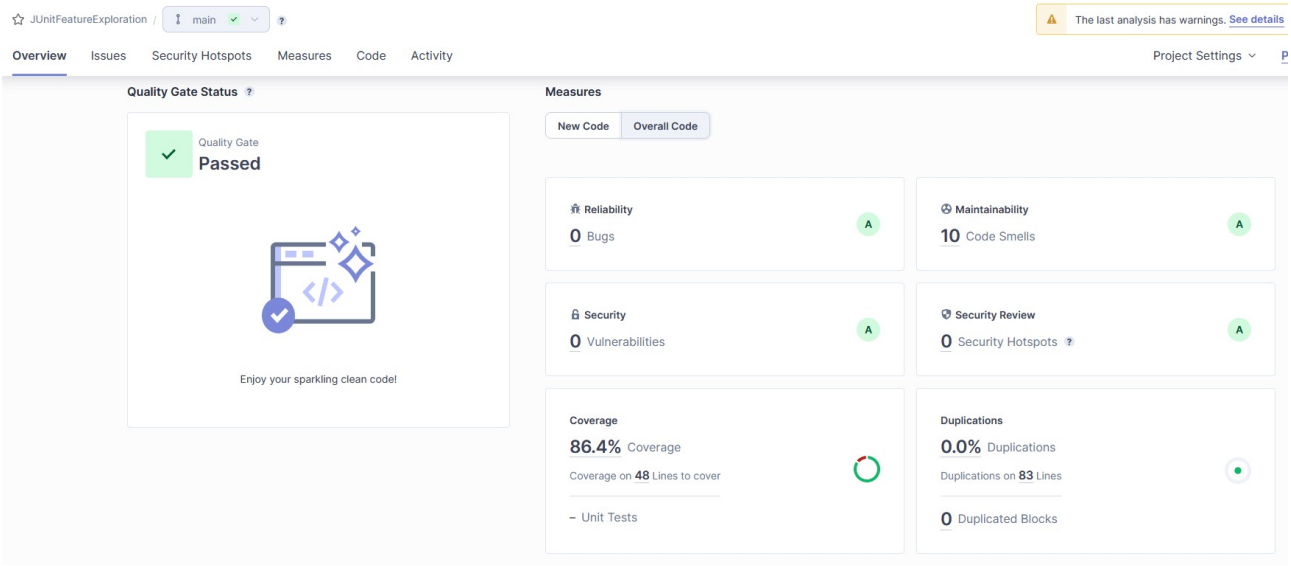
Anmerkung 2: Man muss ebenfalls den Ort (hier die Orte) der kompilierten Klassen in der sonar-project.properties-Datei angeben:

```
sonar.java.binaries=build/classes/java/main,build/classes/java/test
```

SonarQube ausführen und Bericht einsehen

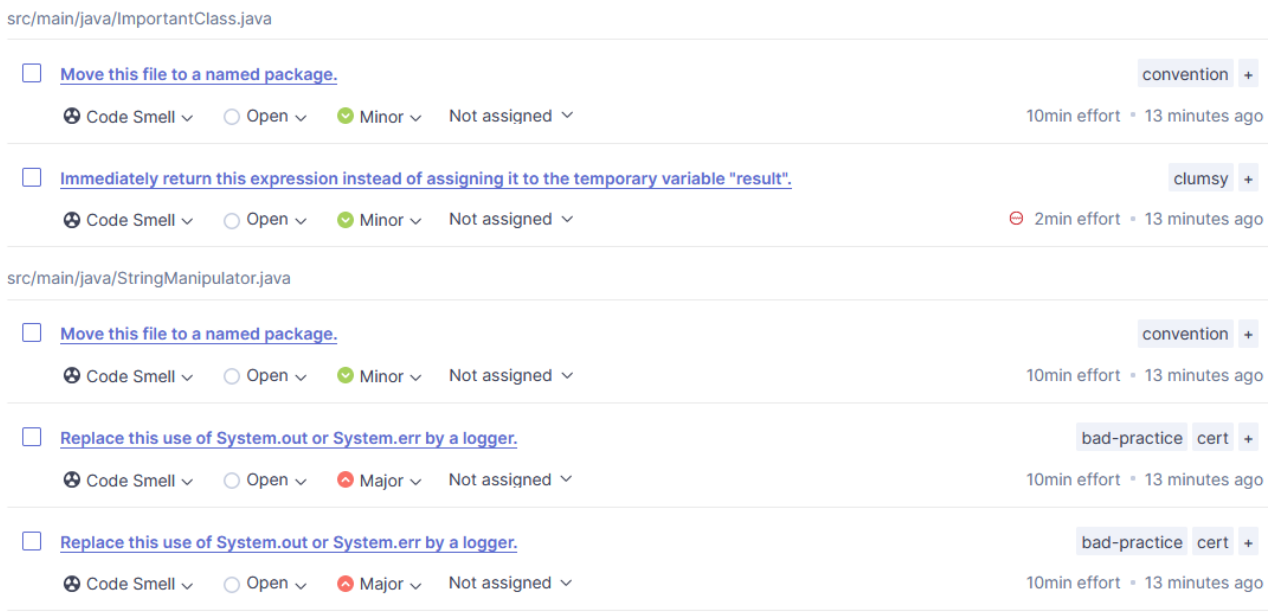
Wir navigieren zu unserem Projekt-Ordner (wo sich die sonar-project.properties-Datei befindet) und geben in der Shell unserer Wahl den Befehl „sonar-scanner“ ein.

In unserer Shell sehen wir dann, nach vielen ausgegebenen Informationen, einen Link zum Report unseres Projektes:



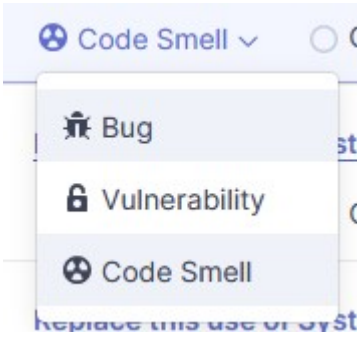
SonarQube zeigt einem an, ob man „Bugs“, „Vulnerabilities“, „Security Hotspots“, aber auch „Code Smells“ in seinem Projekt hat, wie viele es sind und wo sie sich befinden.

Wenn wir auf die 10 gefundenen „Code Smells“ klicken, sehen wir eine Auflistung. Für jeden gefundenen „Code Smell“ ist vermerkt was wir tun können um diesen zu beheben und auch eine Einschätzung, ob es sich um ein kleines oder großes „Problem“ handelt.

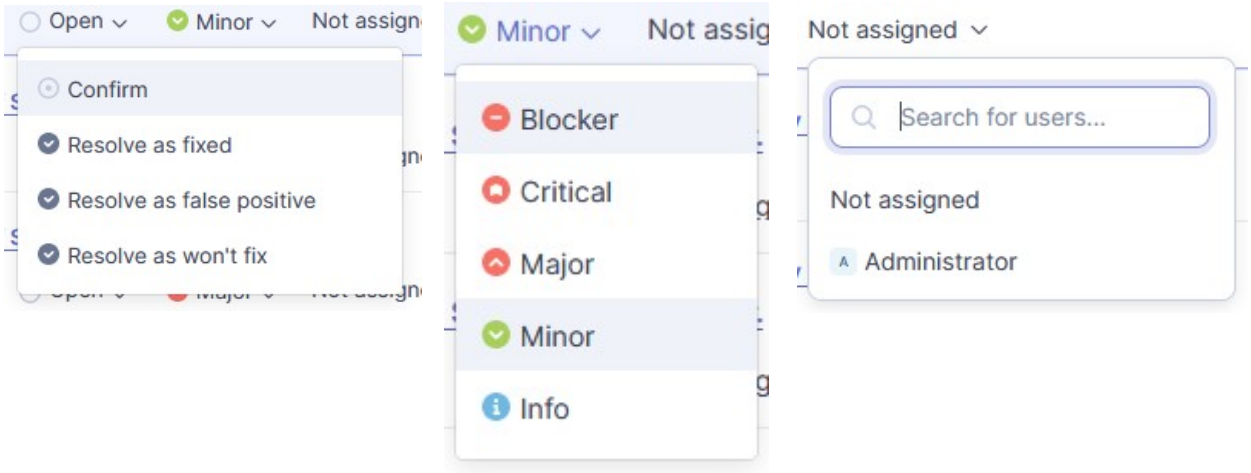


Ausserdem erhalten wir eine Einschätzung für den Zeitaufwand der Behebung und wir erhalten kleine Schlagwörter um den „Code Smell“ zu kategorisieren (wie z.B. „clumsy“, „bad-practice“ usw.).

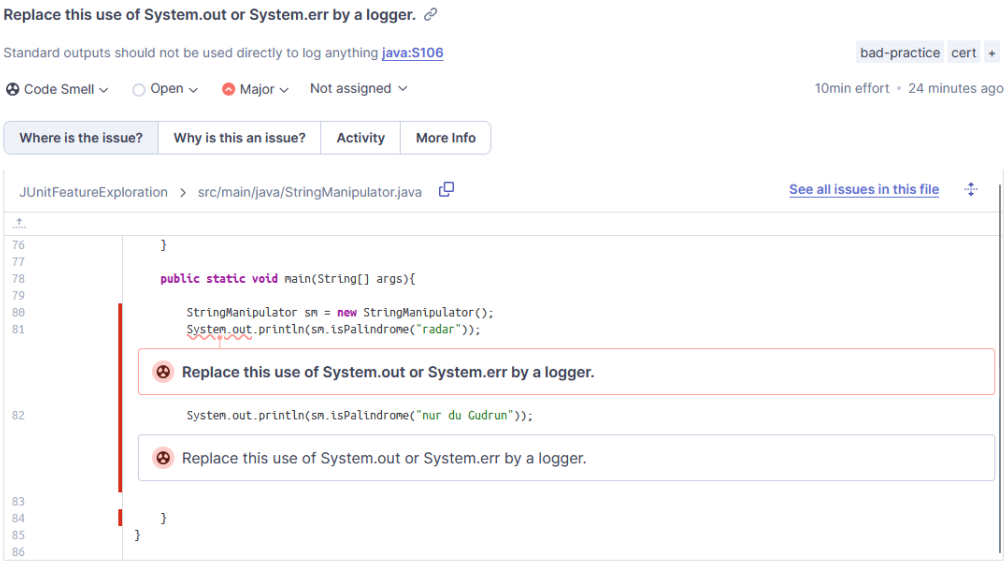
Wir haben die Möglichkeit die Kategorie „Code Smell“ anzupassen um z.B. festzustellen, dass es sich eigentlich um einen „Bug“, oder eine „Vulnerability“ handelt:



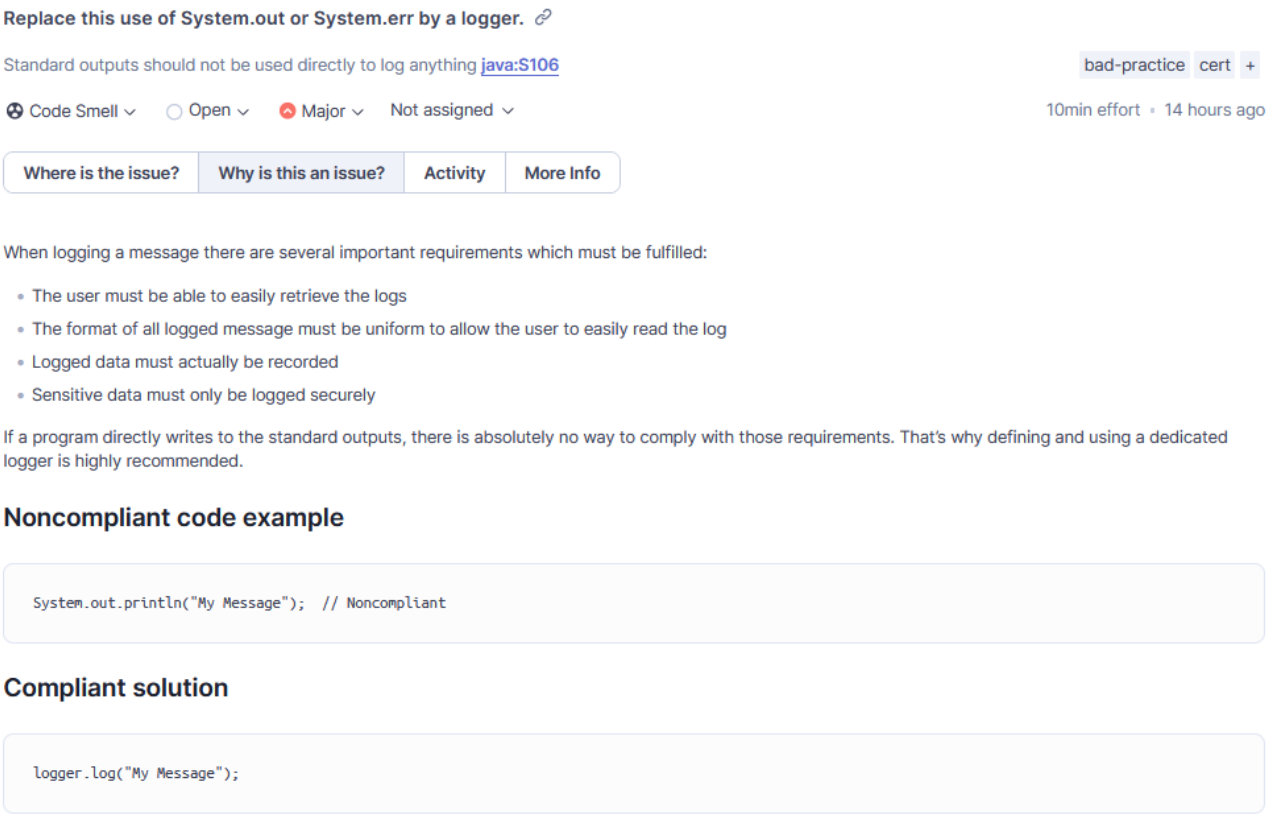
Und es bieten sich weitere Anpassungsmöglichkeiten (selbsterklärend):



Wenn wir hier auf einen „Code Smell“ klicken, dann erhalten wir eine detaillierte Anzeige, die angibt wo genau im Code was zu verändern ist:



Wenn wir auf „Why is this an issue?“ klicken, erhalten wir eine Erklärung warum wir diesen Teil im Code verändern sollten, und eine beispielhafte Lösung:



SonarQube bietet noch viele weitere Features, wie z.B. „Quality Gate“ (sieht man auch oben im Screenshot). Dabei handelt es sich um eine Menge von Bedingungen, die ein Projekt erfüllen muss, bevor es in Produktion gehen sollte. Diese Bedingungen sind z.B. mehr als 80% Test Coverage, keine „security vulnerabilities“, wenige „duplications“ usw. .