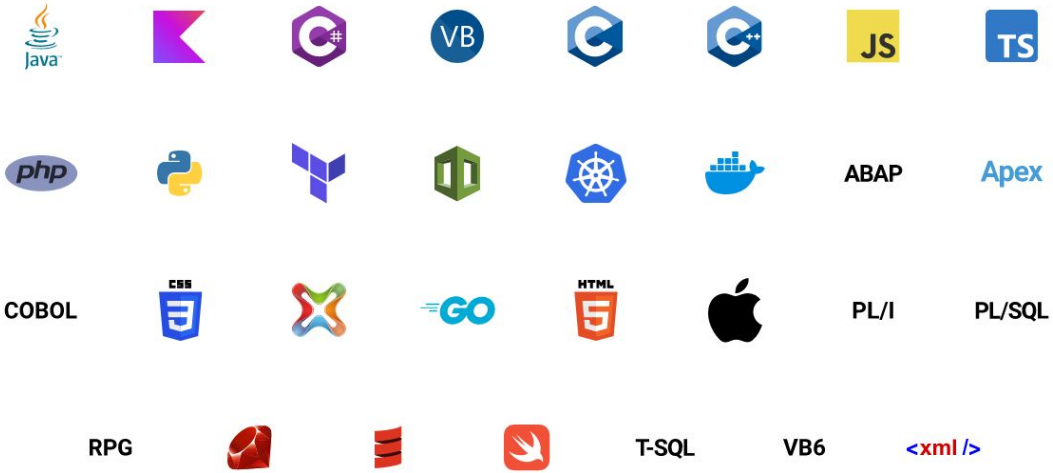
Kacper Machnik
Tomasz Madeja

Czym jest SonarQube?

SonarQube jest platformą open source opracowaną przez SonarSource do ciągłej kontroli jakości kodu w celu przeprowadzania automatycznych przeglądów ze statyczną analizą kodu w celu wykrycia błędów i zapachów kodu* w ponad 30 językach programowania i framework'ach SonarQube oferuje raporty dotyczące zduplikowanego kodu, standardów kodowania, testów jednostkowych, pokrycia kodu testami, złożoności kodu, komentarzy, błędów i zaleceń dotyczących bezpieczeństwa.



*Zapach kodu (ang. code smell)

Jak działa SonarQube?

Najprościej rzecz ujmując w wyniku przeprowadzonej analizy otrzymujemy ogólną miarę jakości oraz listę przypadków, w których zdefiniowane reguły zostały złamane. Jedną z najbardziej użytecznych z punktu widzenia programisty funkcjonalności opisywanej platformy jest Sonarlint – rozszerzenie dostępne dla wszystkich popularnych IDE – pozwalające na wykrywanie nieprawidłowości już w momencie pisania kodu. W celu integracji z IntelliJ IDEA instalujemy plugin Sonarlint.

Instalacja środowiska (dla systemu Windows):

1. Instalacja JDK 17 oraz dodanie do zmiennej środowiskowej PATH

   C:\Program Files\Java\jdk-17\bin

2. Instalacja Maven oraz dodanie do zmiennej środowiskowej PATH

   C:\$MAVEN_HOME$\apache-maven-3.9.5-bin\apache-maven-3.9.5\bin

3. Pobranie SonarQube: (https://www.sonarqube.org/downloads)
4. Włączenie serwera SonarQube

   C:\$SONARQUBE_HOME$\sonarqube-10.2.1.78527\bin\windows-x86-64\StartSonar.bat

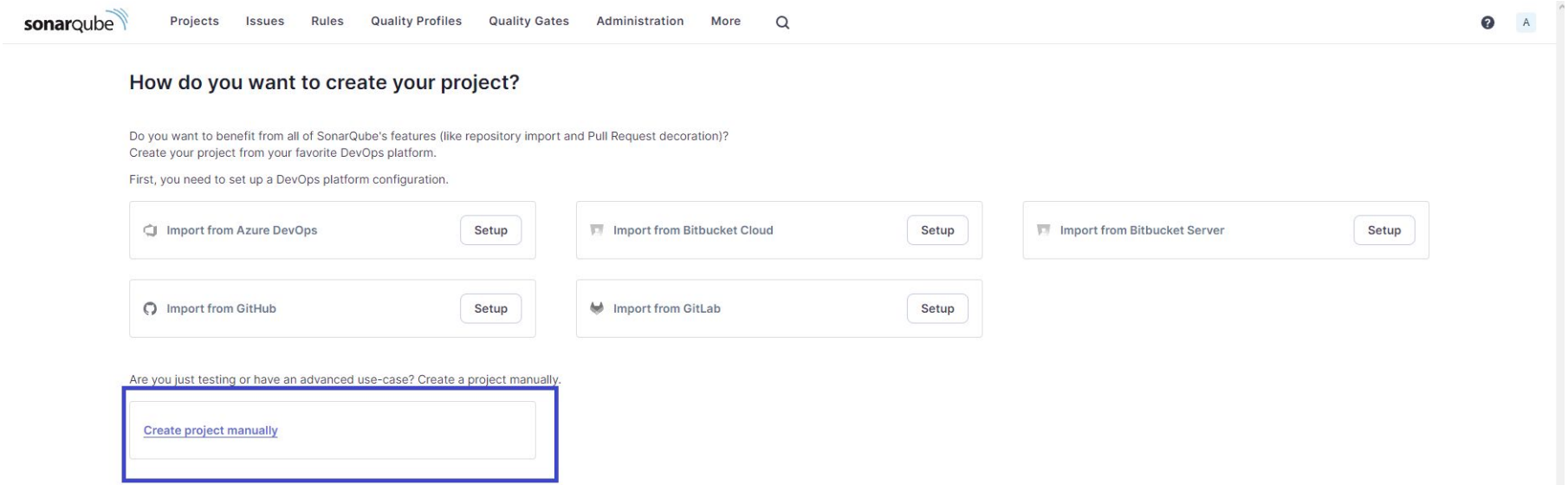
5. Pobranie Sonar Scanner (https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/)
6. Dodanie go do zmiennej środowiskowej PATH

   C:\$SONAR_SCANNER_HOME$\sonar-scanner-cli-5.0.1.3006-windows\sonar-scanner-5.0.1.3006-windows\bin

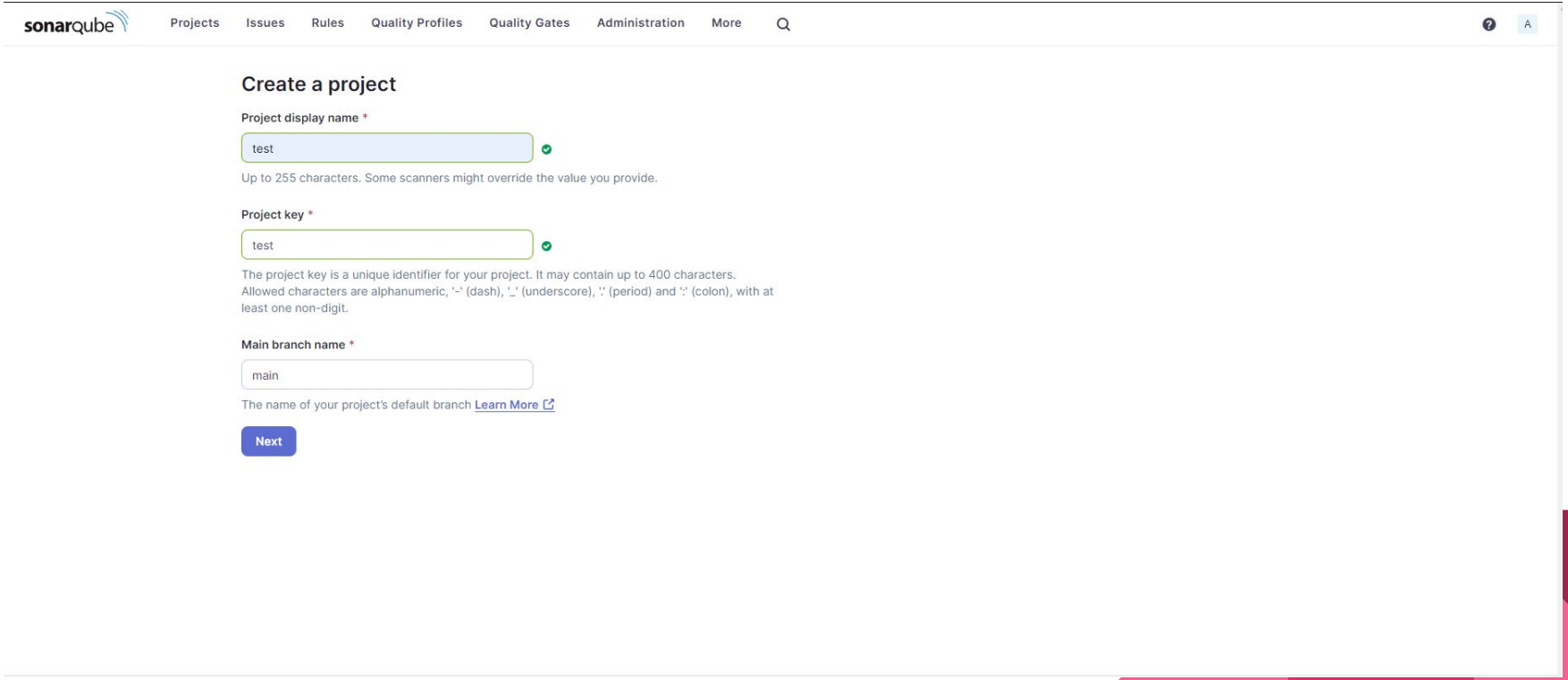
7. Logowanie na (http://localhost:9000) Login: admin | Password: admin
8. Zmiana hasła na inne

Generacja pierwszego raportu

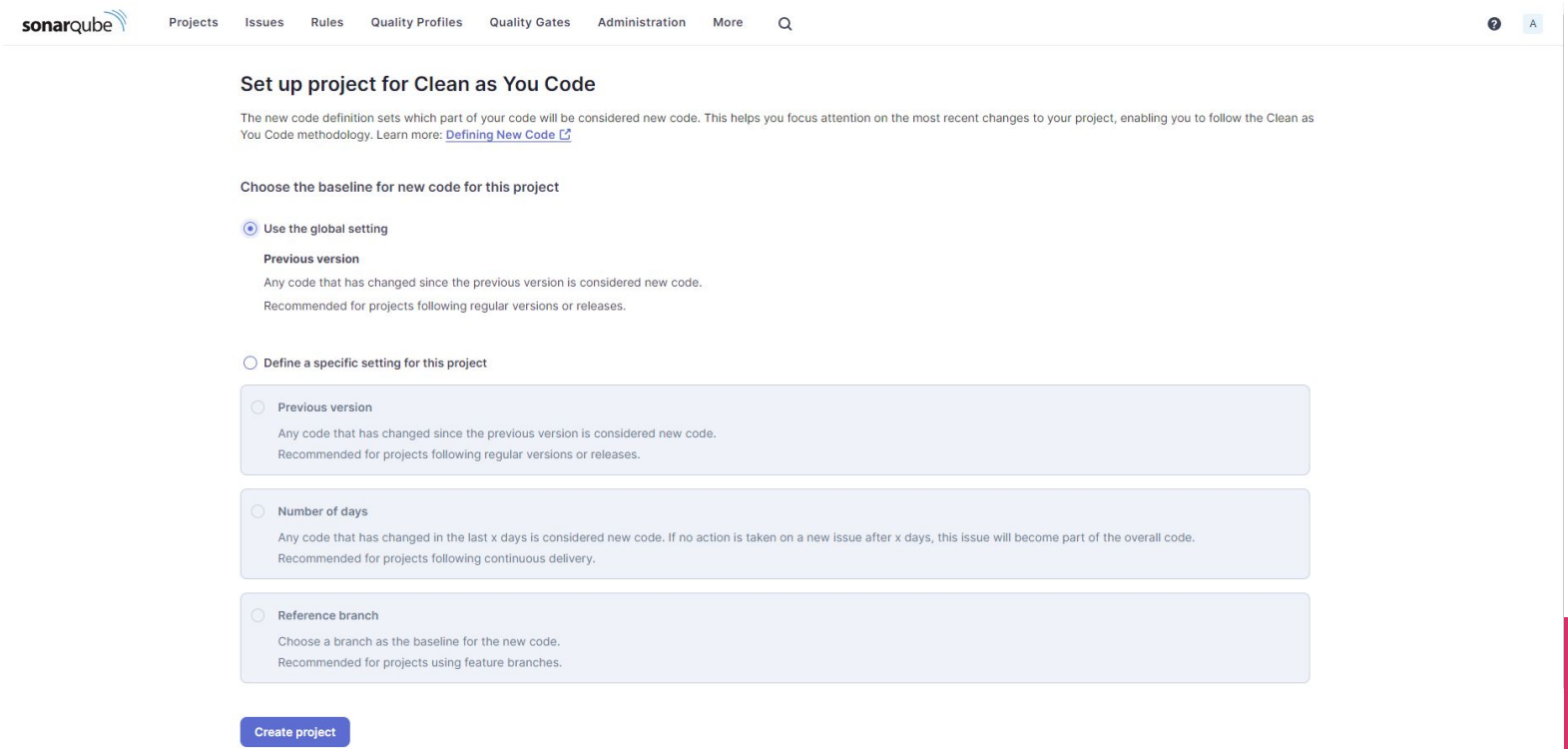
1. Tworzenie nowego projektu.



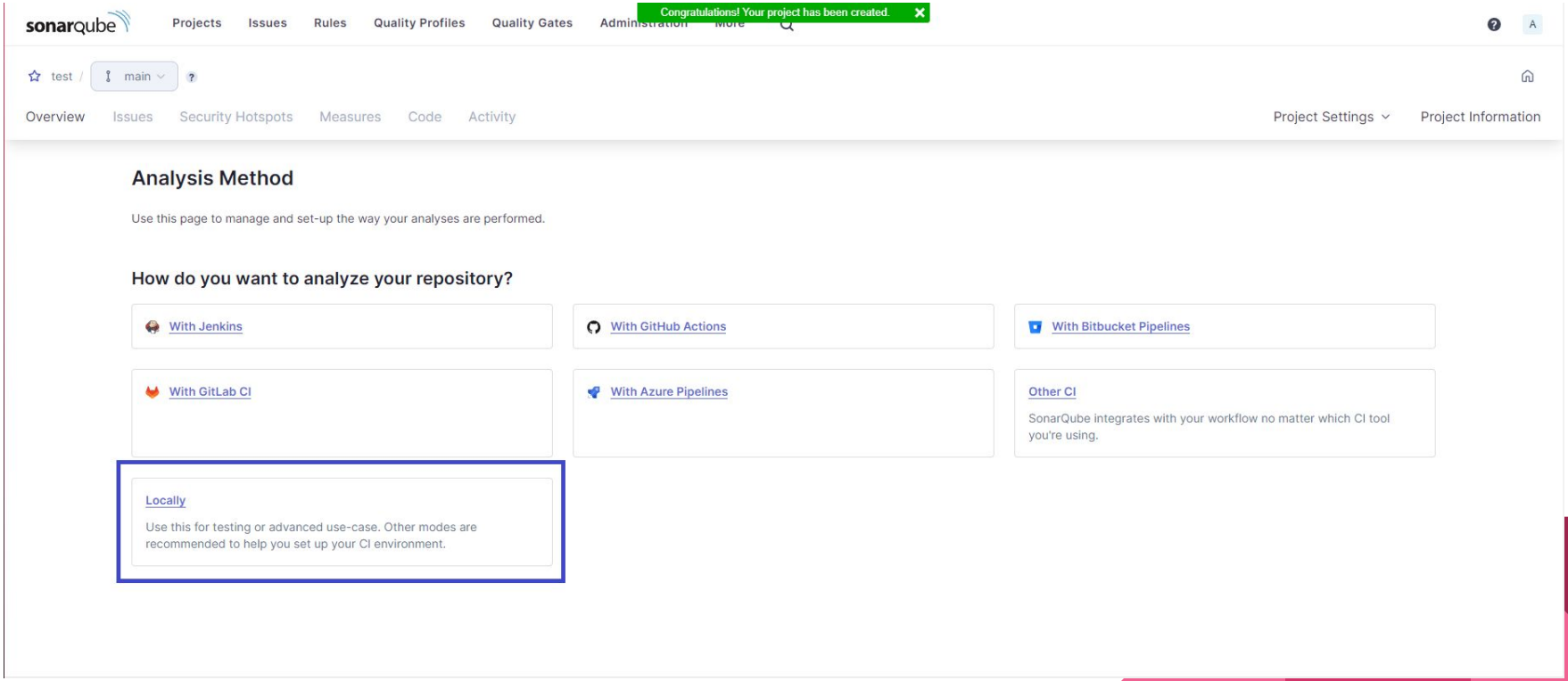
2. Nazwanie projektu oraz utworzenie klucza projektu



3. Wybór ustawień projektu

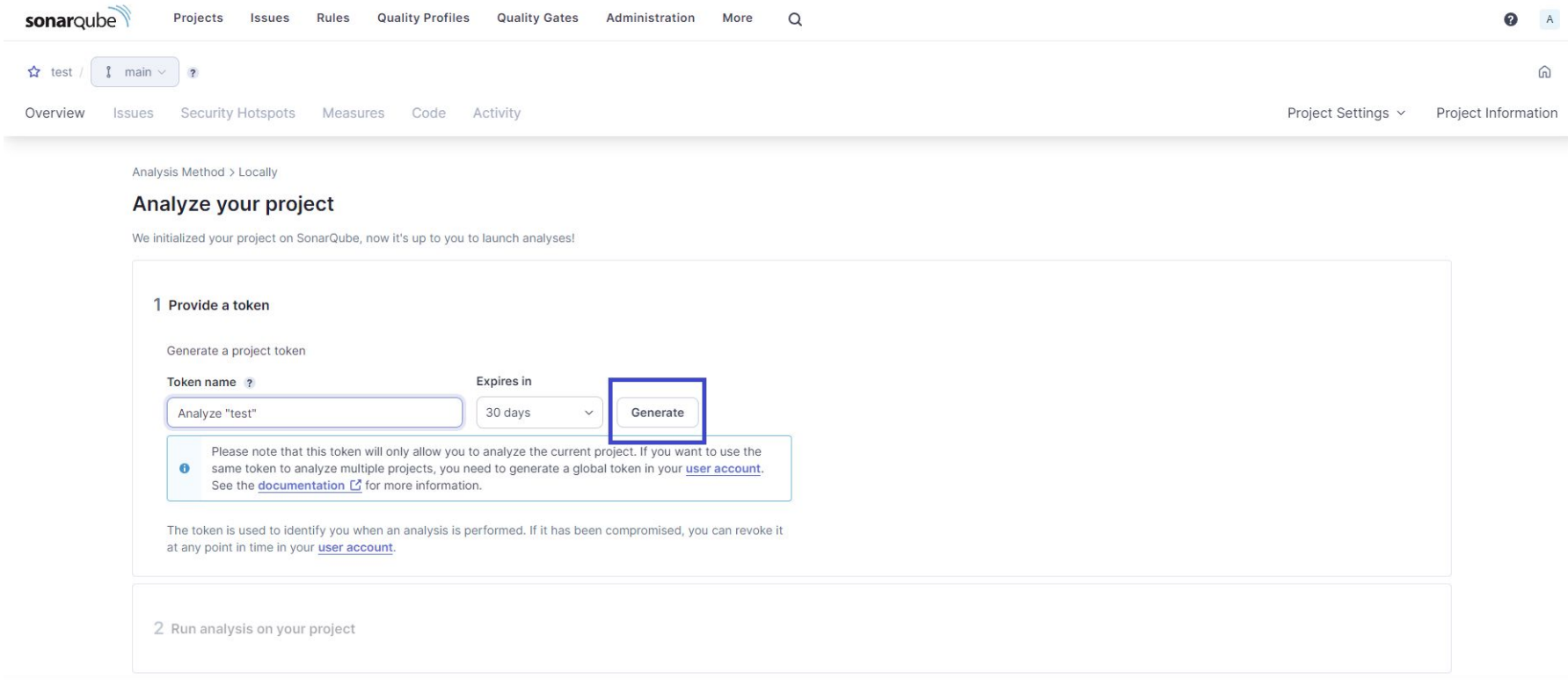


4. Wybór metody analizy projektu

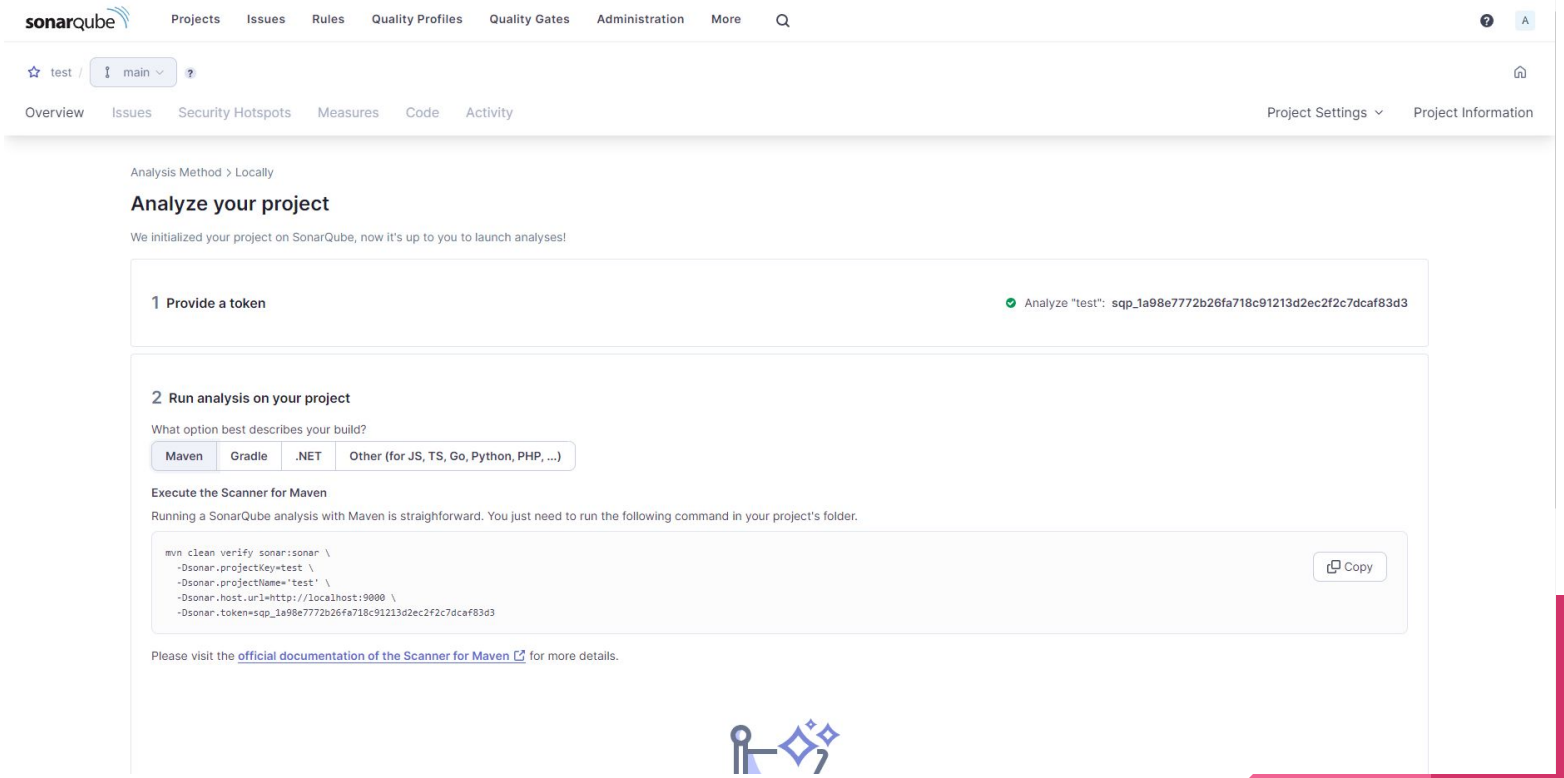


5. Generowanie tokenu

**możliwe jest również użycie wcześniej wygenerowanego tokenu*

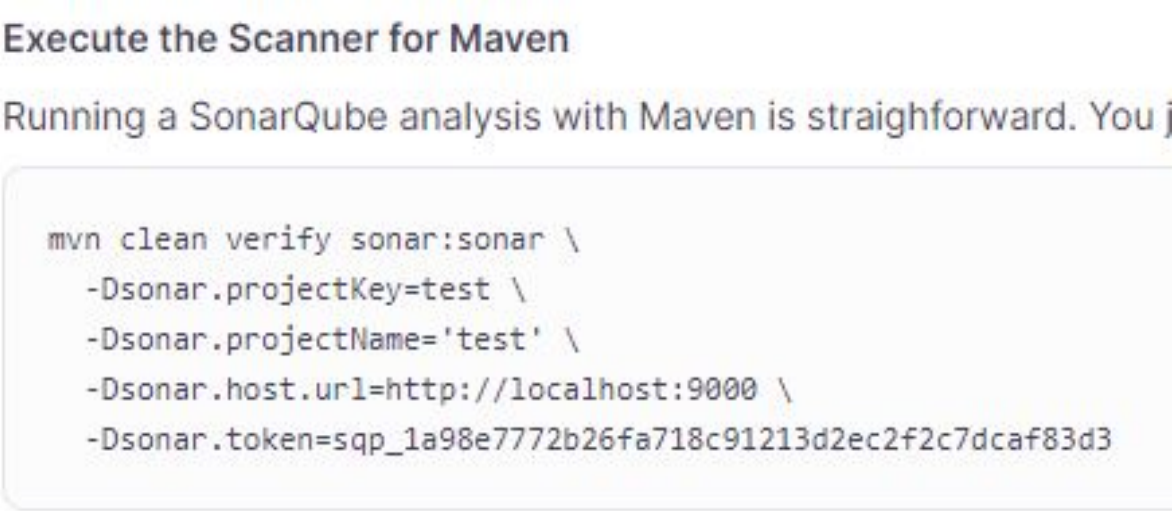


6. Generacja raportu przy użyciu Mavena



Uwaga!

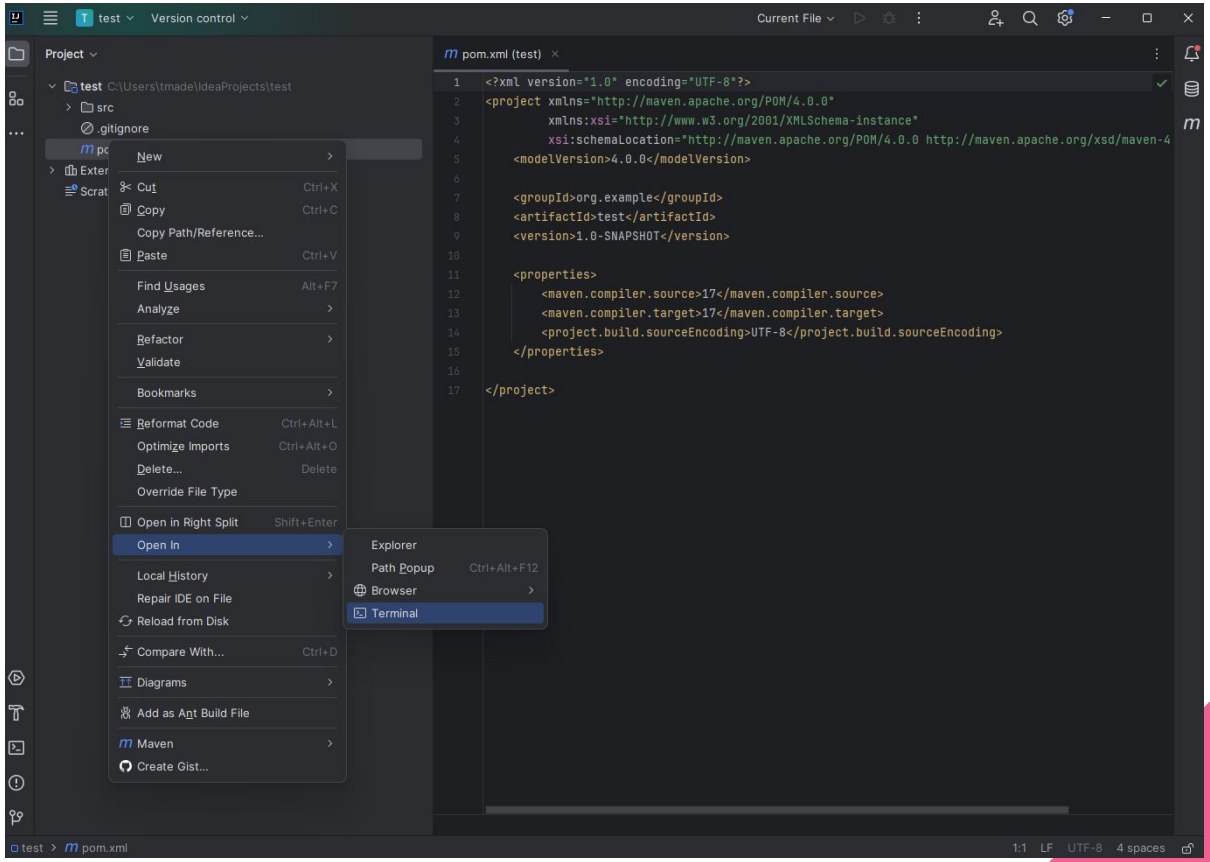
Aby komenda zadziałała prawidłowo musimy usunąć “\” oraz dodać spacje pomiędzy “-D”, a “sonar”



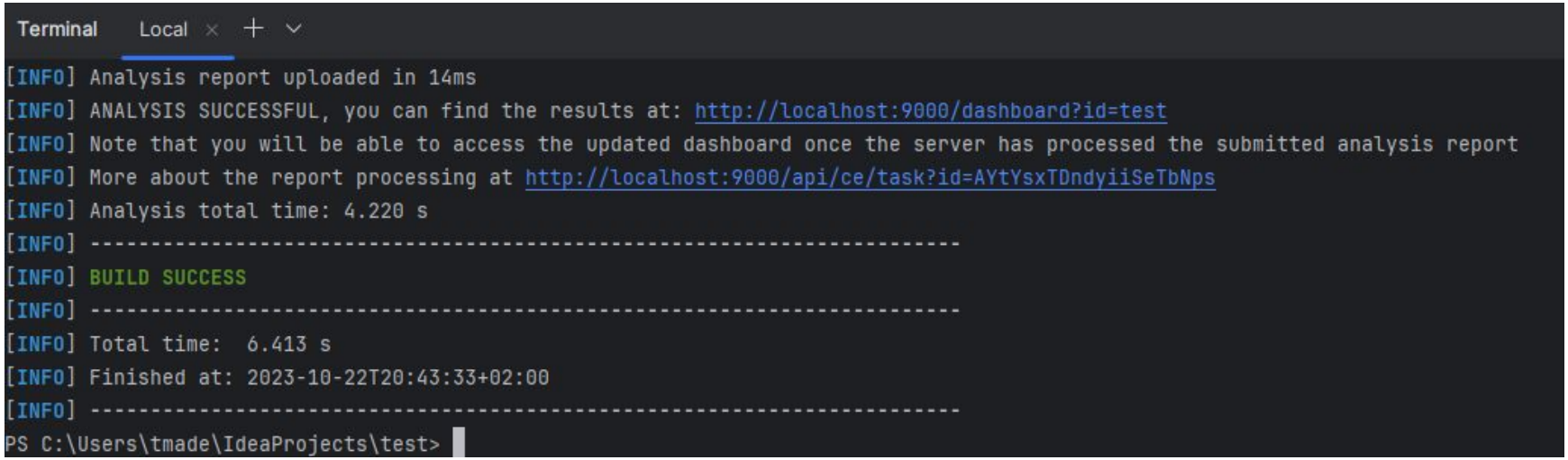
Prawidłowa komenda powinna wyglądać następująco:

mvn clean verify sonar:sonar -D sonar.projectKey=*YOUR_projectKey* -D sonar.projectName='*YOUR_projectName*' -D sonar.host.url=*YOUR_host.url* -D sonar.token=*YOUR_token*

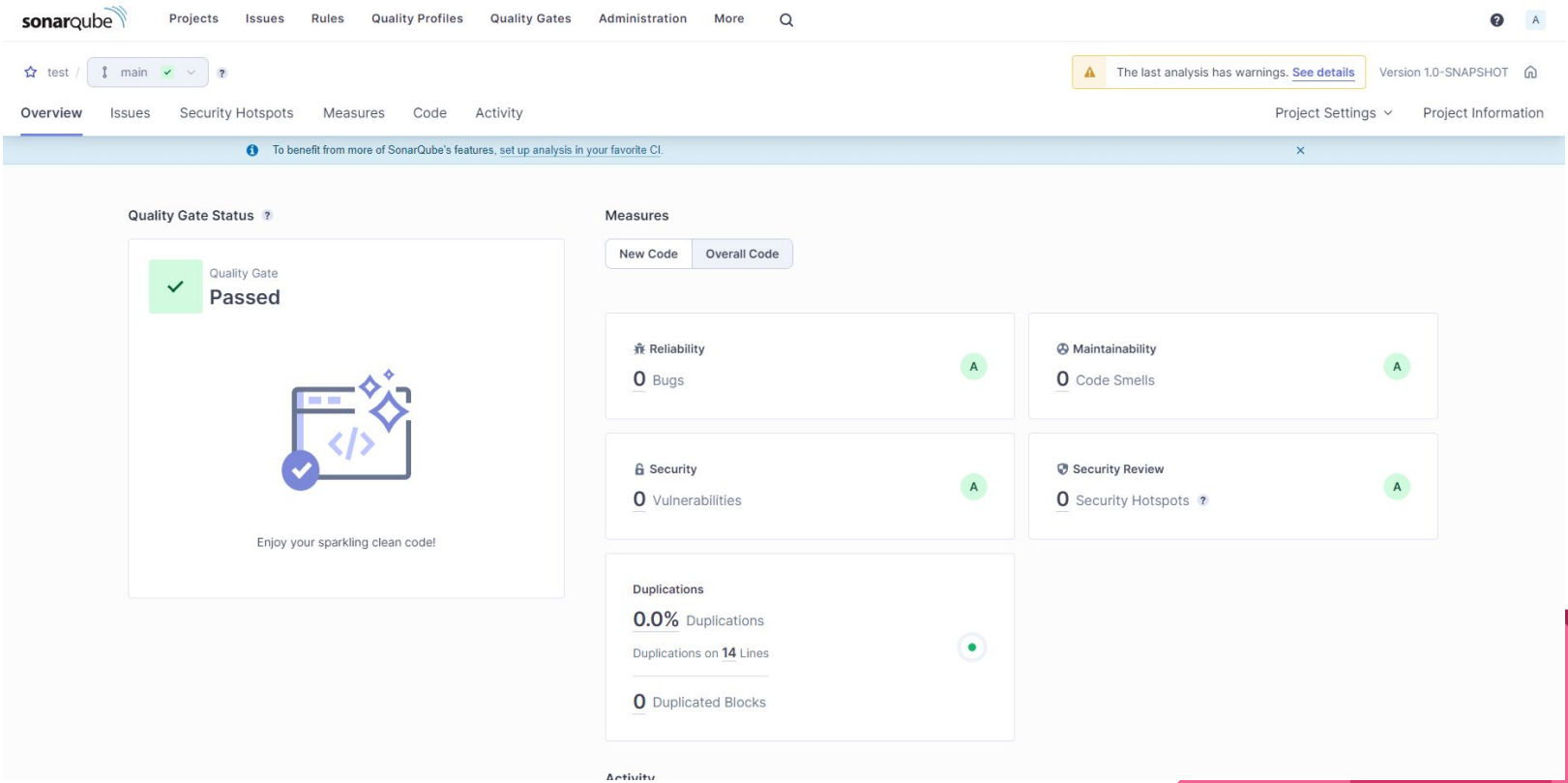
7. Wpisanie komendy w terminalu otwartym w projekcie



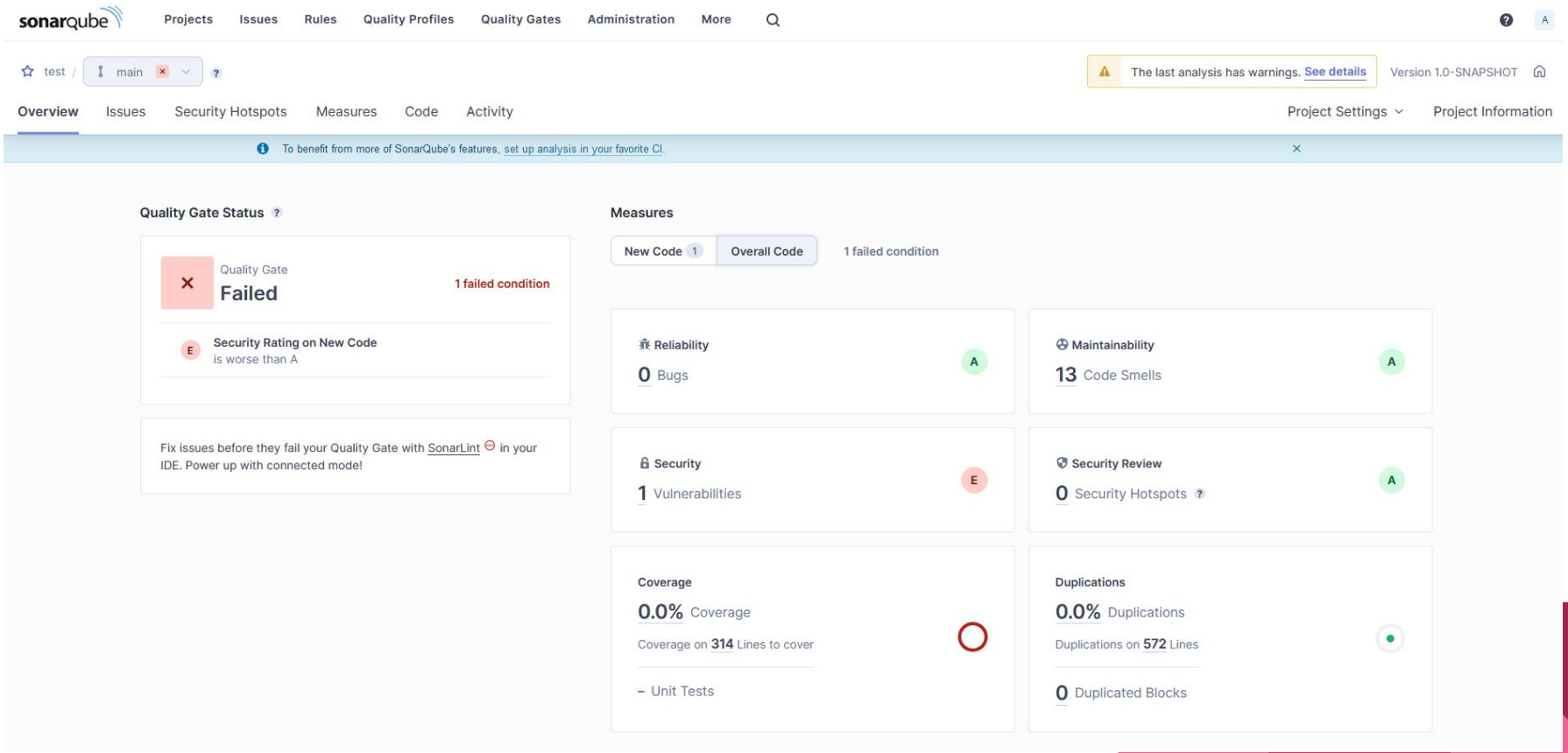
8. Jeżeli wszystkie kroki zostały wykonane poprawnie wywołane komendy powinno zakończyć się sukcesem



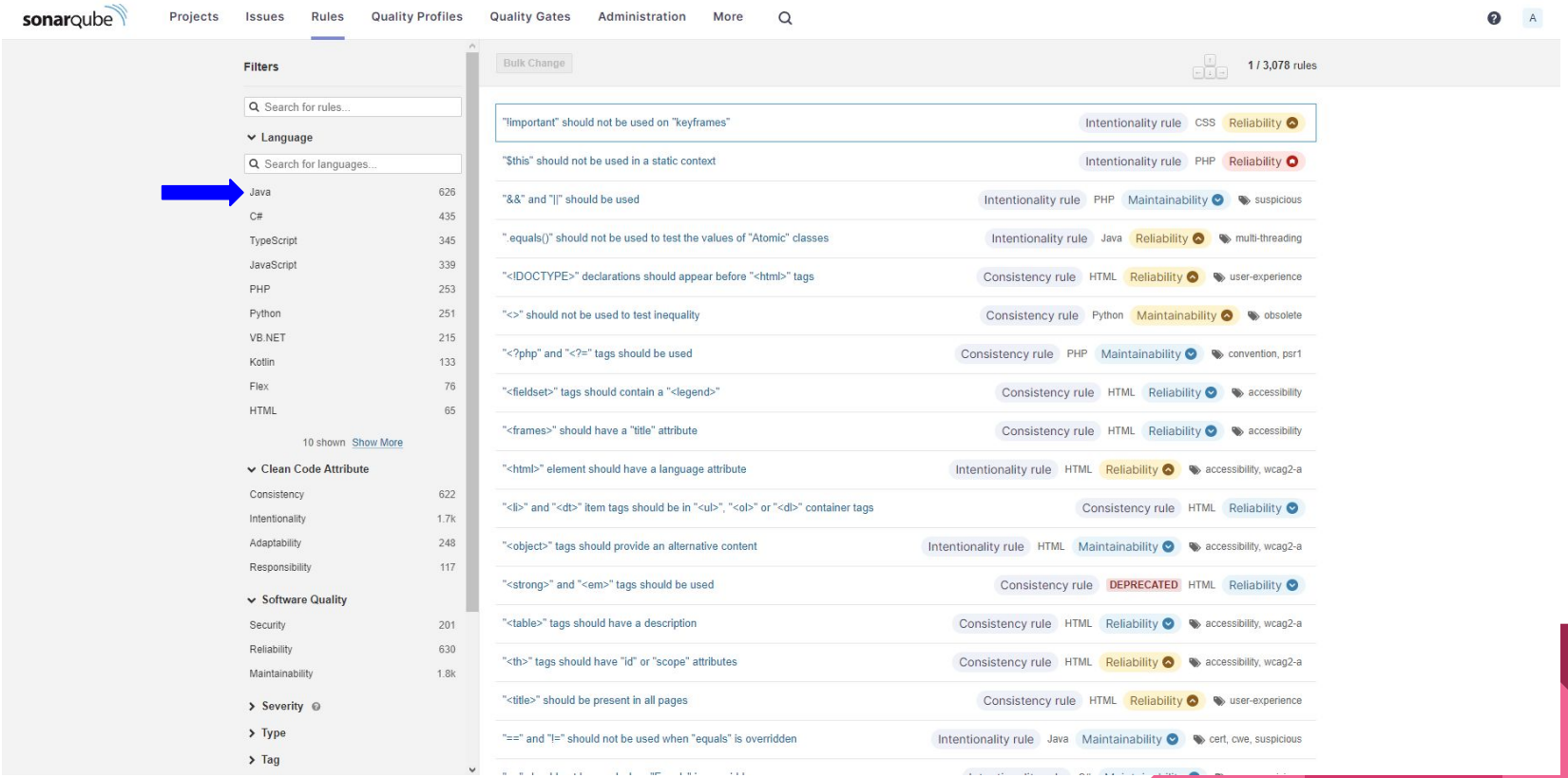
Raport


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
test / main
The last analysis has warnings. See details
Version 1.0-SNAPSHOT
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
To benefit from more of SonarQube's features, set up analysis in your favorite CI.
Quality Gate Status
Quality Gate
Passed
Enjoy your sparkling clean code!
Measures
New Code
Overall Code
Reliability
0 Bugs
A
Maintainability
0 Code Smells
A
Security
0 Vulnerabilities
A
Security Review
0 Security Hotspots
A
Duplications
0.0% Duplications
Duplications on 14 Lines
0 Duplicated Blocks
Activity

Raport


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
test / main
The last analysis has warnings. See details
Version 1.0-SNAPSHOT
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
To benefit from more of SonarQube's features, set up analysis in your favorite CI.
Quality Gate Status
Quality Gate
Failed
1 failed condition
Security Rating on New Code
is worse than A
Fix issues before they fail your Quality Gate with SonarLint in your IDE. Power up with connected mode!
Measures
New Code 1
Overall Code
1 failed condition
Reliability
0 Bugs
A
Maintainability
13 Code Smells
A
Security
1 Vulnerabilities
E
Security Review
0 Security Hotspots
A
Coverage
0.0% Coverage
Coverage on 314 Lines to cover
– Unit Tests
Duplications
0.0% Duplications
Duplications on 572 Lines
0 Duplicated Blocks

Zasady

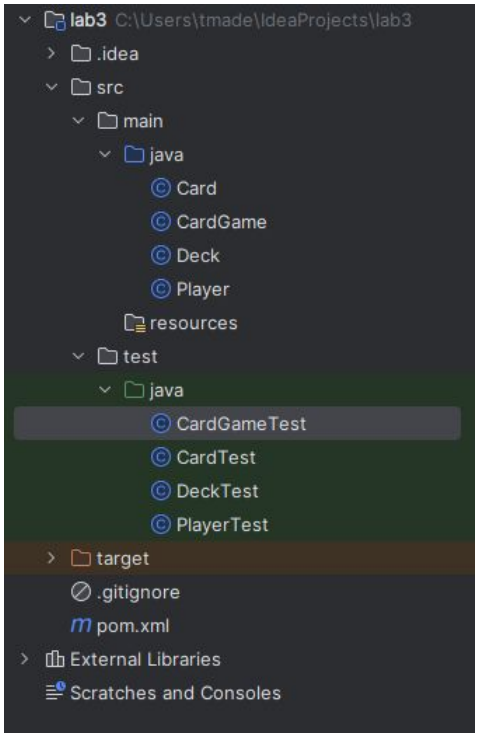

sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
Filters
Search for rules...
Language
Search for languages...
Java 626
C# 435
TypeScript 345
JavaScript 339
PHP 253
Python 251
VB.NET 215
Kotlin 133
Flex 76
HTML 65
10 shown Show More
Clean Code Attribute
Consistency 622
Intentionality 1.7k
Adaptability 248
Responsibility 117
Software Quality
Security 201
Reliability 630
Maintainability 1.8k
Severity
Type
Tag
Bulk Change
1 / 3,078 rules
"!important" should not be used on "keyframes" Intentionality rule CSS Reliability
"$this" should not be used in a static context Intentionality rule PHP Reliability
"&&" and "||" should be used Intentionality rule PHP Maintainability suspicious
".equals()" should not be used to test the values of "Atomic" classes Intentionality rule Java Reliability multi-threading
"<!DOCTYPE>" declarations should appear before "<html>" tags Consistency rule HTML Reliability user-experience
"<>" should not be used to test inequality Consistency rule Python Maintainability obsolete
"<?php" and "<?=" tags should be used Consistency rule PHP Maintainability convention, psr1
"<fieldset>" tags should contain a "<legend>" Consistency rule HTML Reliability accessibility
"<frames>" should have a "title" attribute Consistency rule HTML Reliability accessibility
"<html>" element should have a language attribute Intentionality rule HTML Reliability accessibility, wcag2-a
"<li>" and "<dt>" item tags should be in "<ul>", "<ol>" or "<dl>" container tags Consistency rule HTML Reliability
"<object>" tags should provide an alternative content Intentionality rule HTML Maintainability accessibility, wcag2-a
"" and "" tags should be used Consistency rule DEPRECATED HTML Reliability
"<table>" tags should have a description Consistency rule HTML Reliability accessibility, wcag2-a
"<th>" tags should have "id" or "scope" attributes Consistency rule HTML Reliability accessibility, wcag2-a
"<title>" should be present in all pages Consistency rule HTML Reliability user-experience
"==" and "!=" should not be used when "equals" is overridden Intentionality rule Java Maintainability cert, cwe, suspicious

Koncepty

Dzielą się na architekturalne i jakościowe. Do tych pierwszych należą: analizator, baza danych, oraz serwer. Natomiast do jakościowych między innymi należy: ”Clean Code”, “Code Smell” i “Bug”. Więcej na ich temat można przeczytać na: https://docs.sonarsource.com/sonarqube/latest/user-guide/concepts/

Pokrycie testami (Coverage)

1. Napisanie testów jednostkowych



Korzystamy z JUnita do napisania testów. Powinny się one znaleźć w zakładce src/test/java. Każda klasa ma odpowiadającą klasę testującą.

2. JaCoCo coverage tool

SonarQube nie generuje raportu pokrycia samodzielnie. Zamiast tego konfigurujemy narzędzie do tworzenia raportu w ramach procesu kompilacji. W przypadku projektów Java, SonarQube bezpośrednio obsługuje narzędzie pokrycia JaCoCo.

Dodanie JaCoCo do projektu Maven

```
<profile>
  <id>coverage</id>
  <build>
   <plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
     <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.8.7</version>
      <executions>
        <execution>
          <id>prepare-agent</id>
          <goals>
            <goal>prepare-agent</goal>
          </goals>
        </execution>
        <execution>
          <id>report</id>
          <goals>
            <goal>report</goal>
          </goals>
          <configuration>
            <formats>
              <format>XML</format>
            </formats>
          </configuration>
        </execution>
      </executions>
    </plugin>
    ...
   </plugins>
  </build>
</profile>
```

Do pliku pom.xml dodajemy odpowiedni profil*, według wzoru z dokumentacji: https://docs.sonarsource.com/sonarqube/9.9/analyzing-source-code/test-coverage/java-test-coverage/

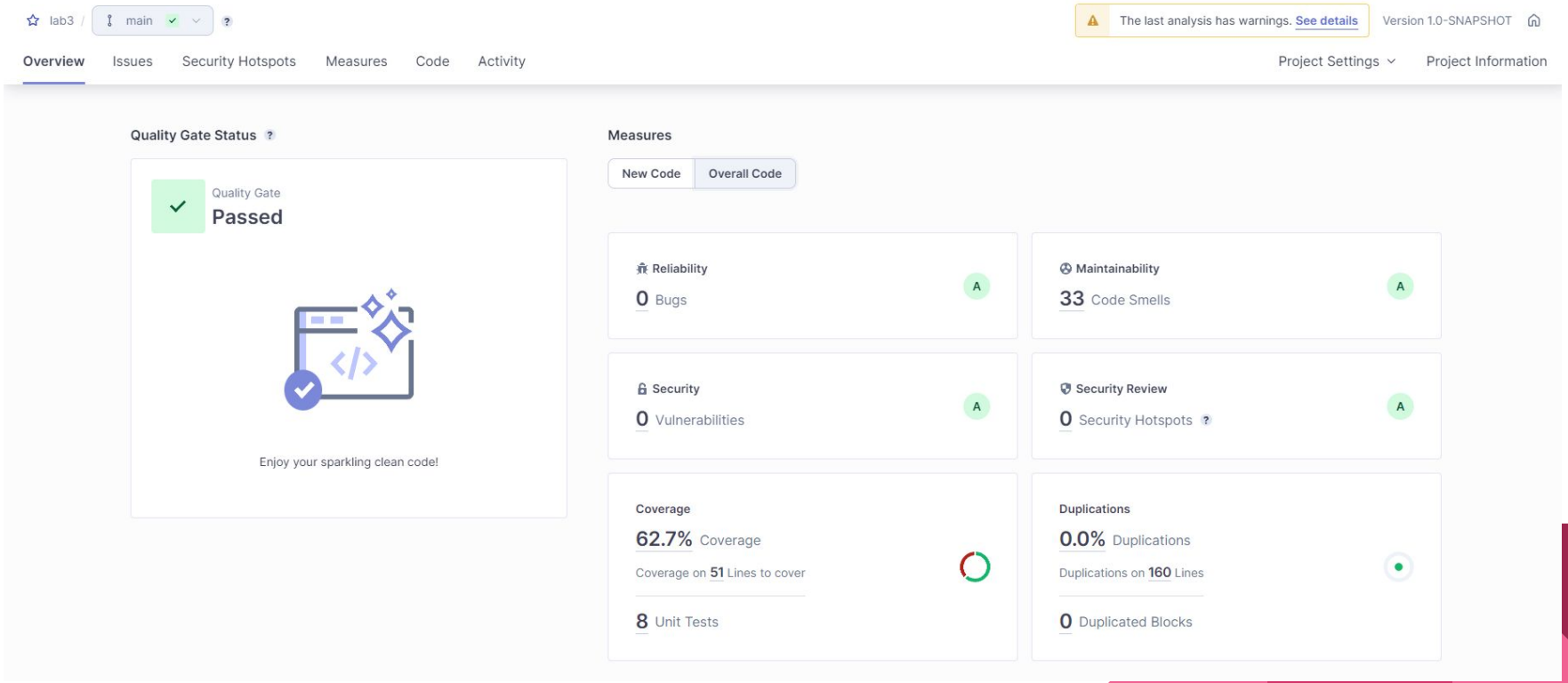
**Całość ma się znaleźć w sekcji <profiles></profiles>*

3. Generacja raportu

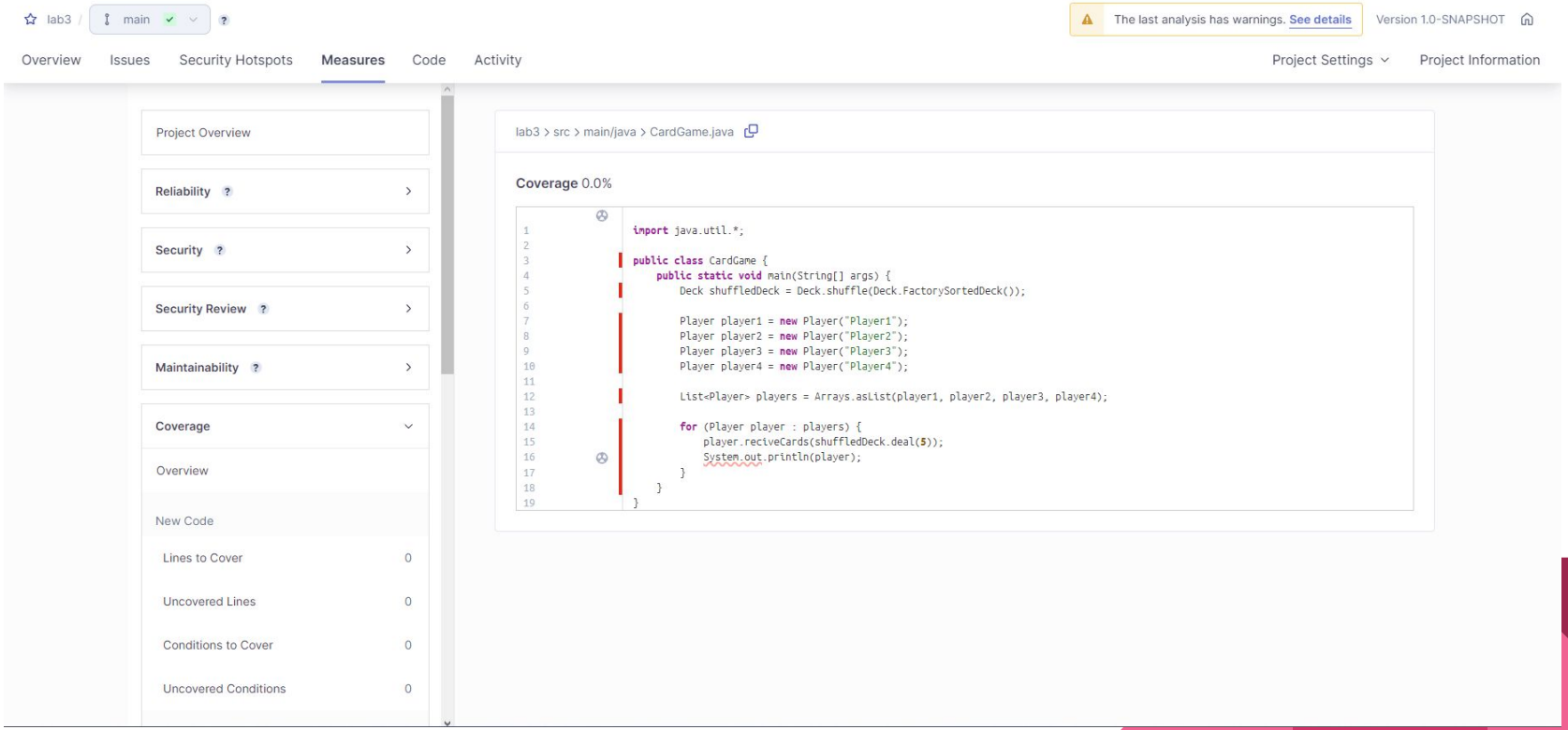
Aby wygenerować raport wykorzystujemy poprzednio podaną komendę, dopisując do niej ***-P coverage*** i wklejamy ją do terminala, powinna ona wyglądać następująco:

mvn clean verify sonar:sonar -D sonar.projectKey=*YOUR_projectKey* -D sonar.projectName='*YOUR_projectName*' -D sonar.host.url=*YOUR_host.url* -D sonar.token=*YOUR_token* *-P coverage*

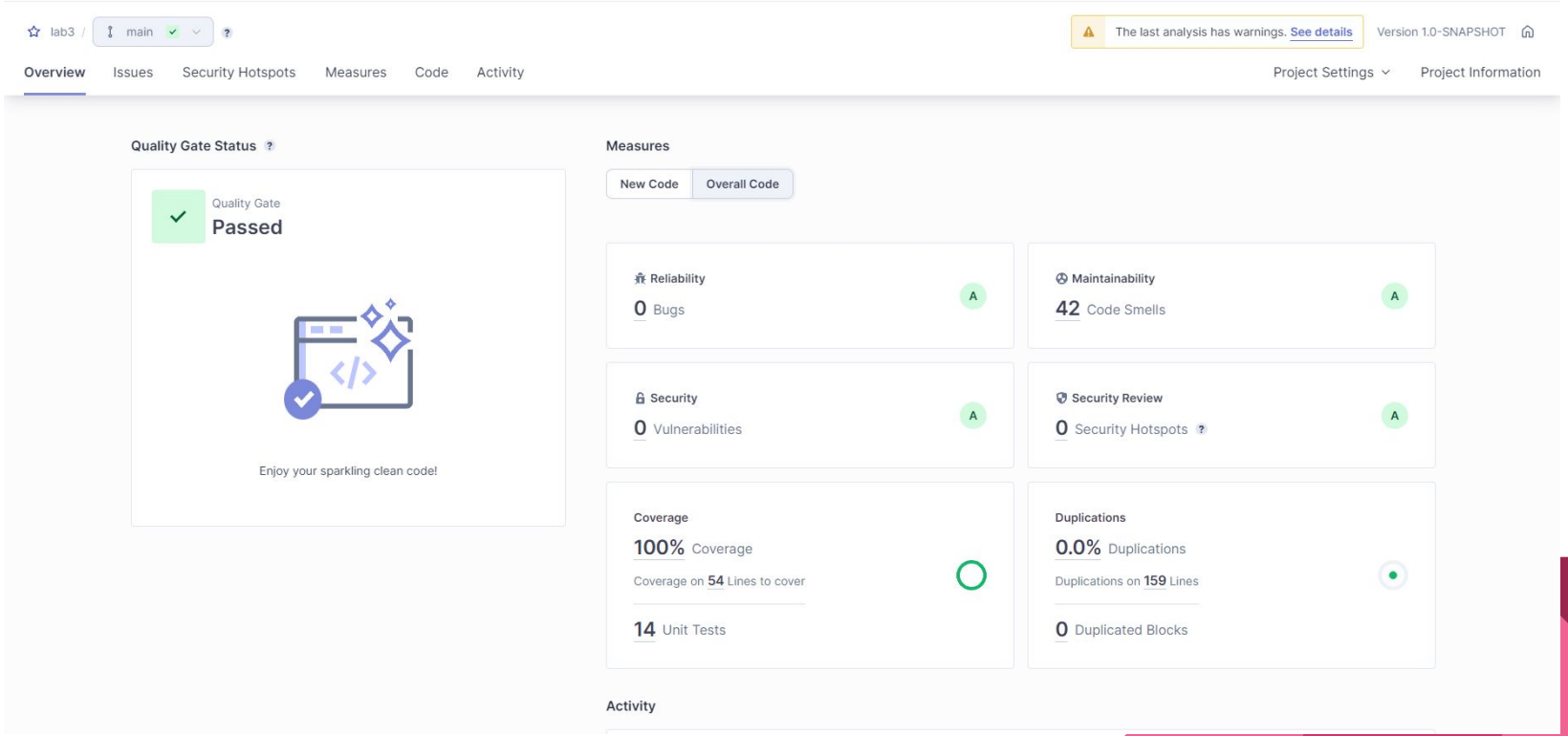
Raport po napisaniu części testów



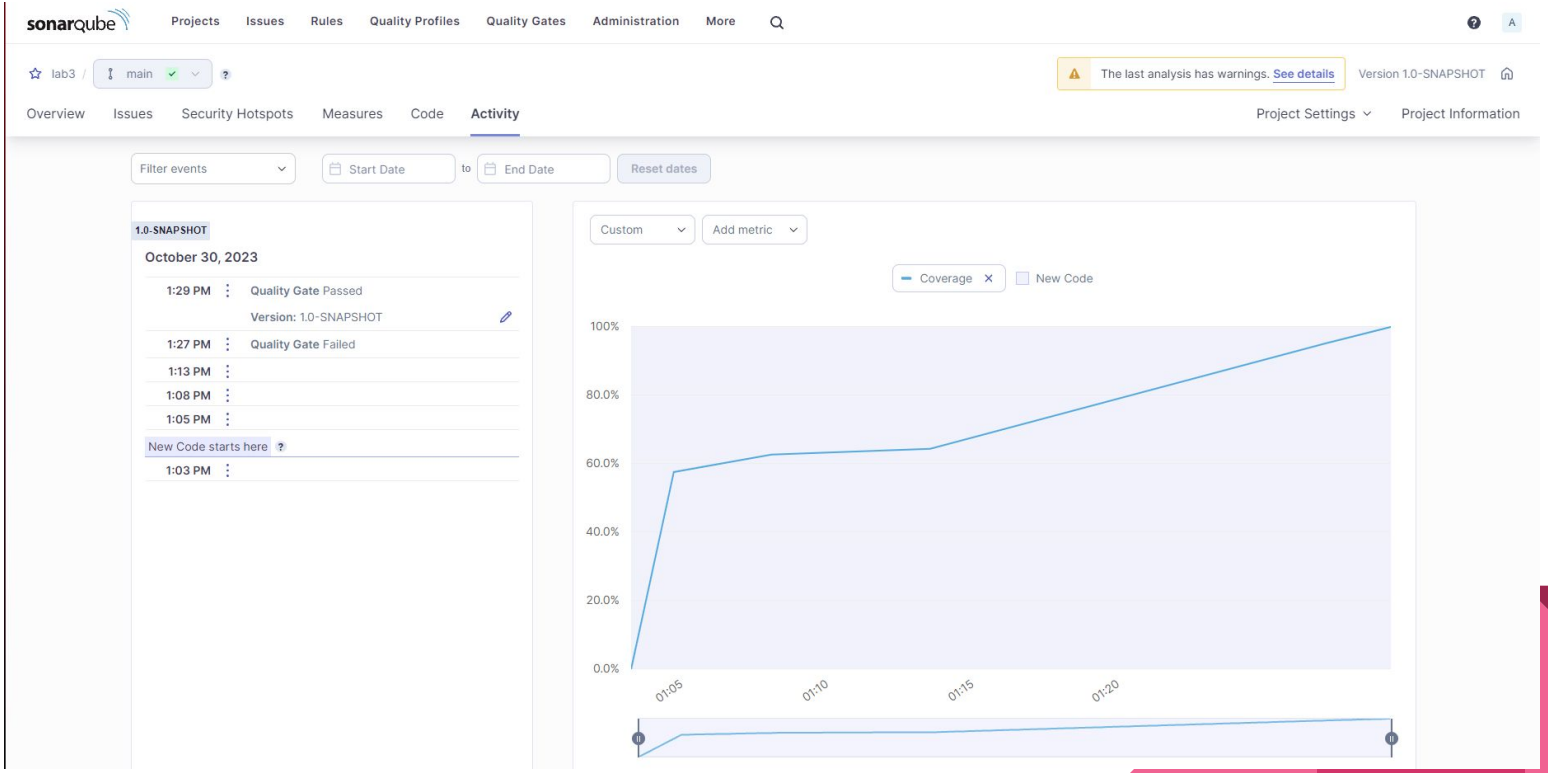
Strona pokazuje do jakich części kodu nie ma testów



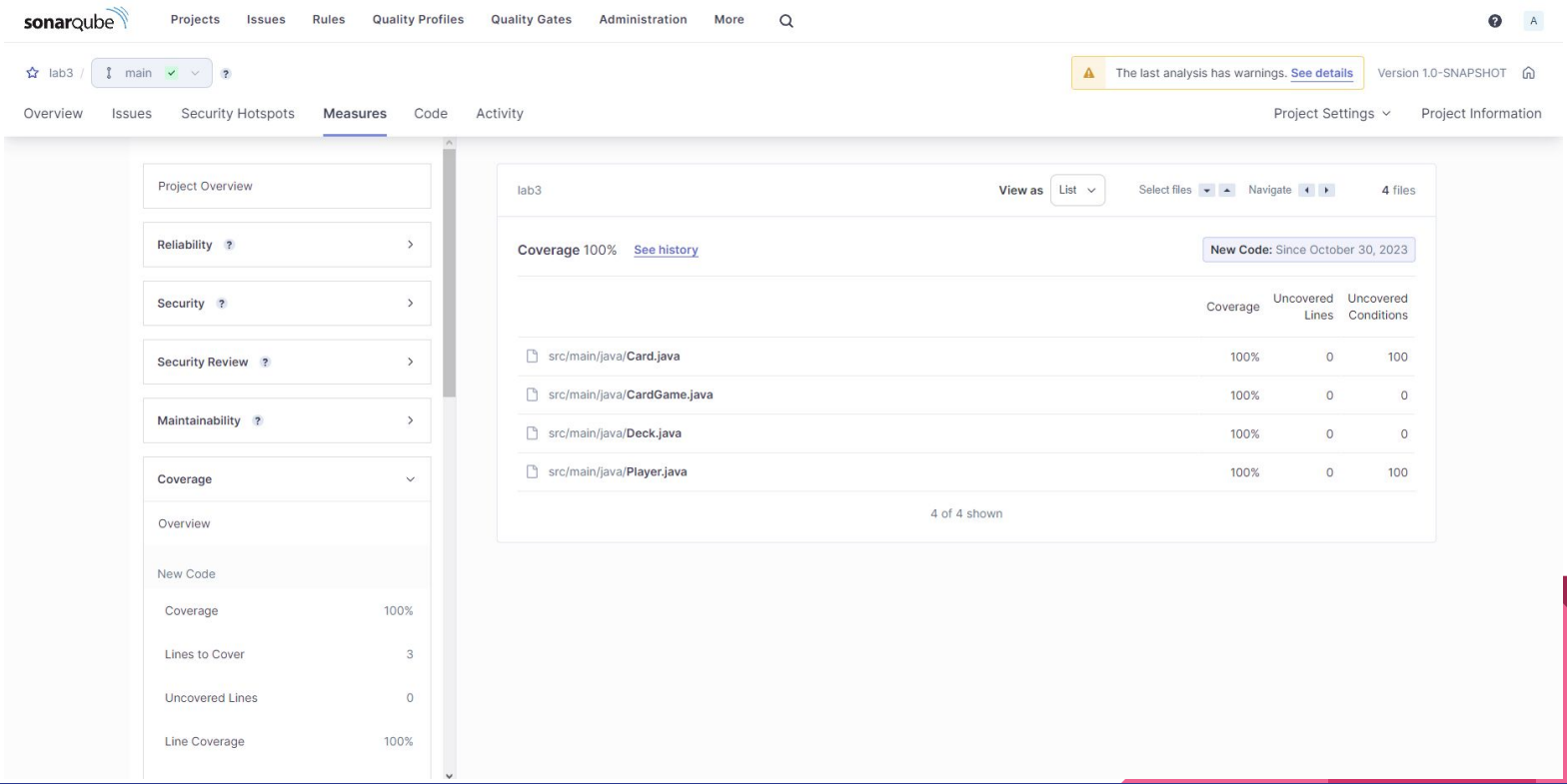
Po napisaniu testów do każdej funkcji



Wykres pokrycia testami w czasie



Stopień pokrycia testami danych klas



Przydatne linki

- https://docs.sonarsource.com/sonarqube/latest/user-guide/concepts/
- https://www.swtestacademy.com/sonarqube-tutorial/
- https://www.sonarsource.com/products/sonarqube/
- https://www.altkomsoftware.com/pl/blog/sonarqube-pierwsze-kroki/
- https://docs.sonarsource.com/sonarqube/latest/
- https://docs.sonarsource.com/sonarqube/9.9/analyzing-source-code/test-coverage/java-test-coverage/

Dziękujemy za uwagę! :)