Paweł Wolski 234014

Tworzenie i Utrzymanie Czystego Kodu

Metryki jakości kodu – SonarQube

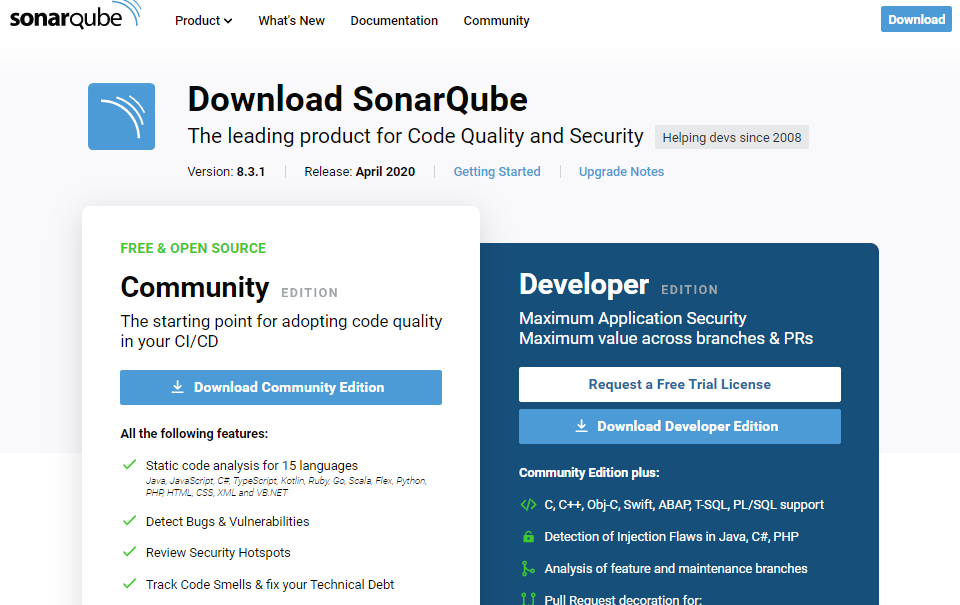
Laboratorium 6

1. **Wstęp**

Celem ćwiczenia było zapoznanie z narzędziem SonarQube, służącym do analizowania kodu pod względem statystycznym. Dzięki niemu można przejrzeć kod pod względem błędów, optymalności bądź luk bezpieczeństwa. Do przetestowania danego kodu została użyta aplikacja kalkulatora, przygotowana i przetestowana za pomocą wtyczki do mavena JaCoCo.

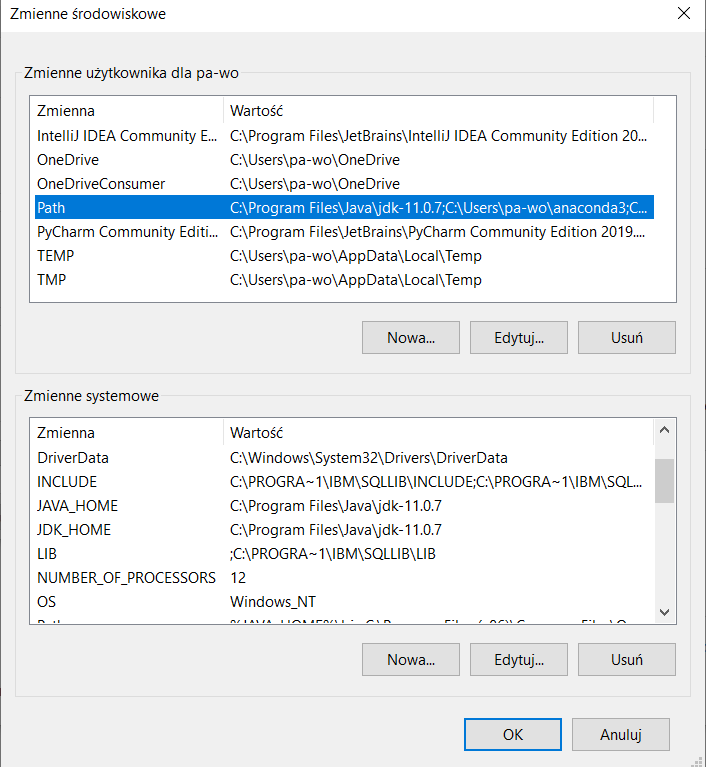
1. **Ćwiczenie**
   1. **Instalacja środowiska**

Ćwiczenie należało zacząć od pobrania narzędzia SonarQube ze strony <https://www.sonarqube.org/downloads/> widocznej na Rys. 2.1.1



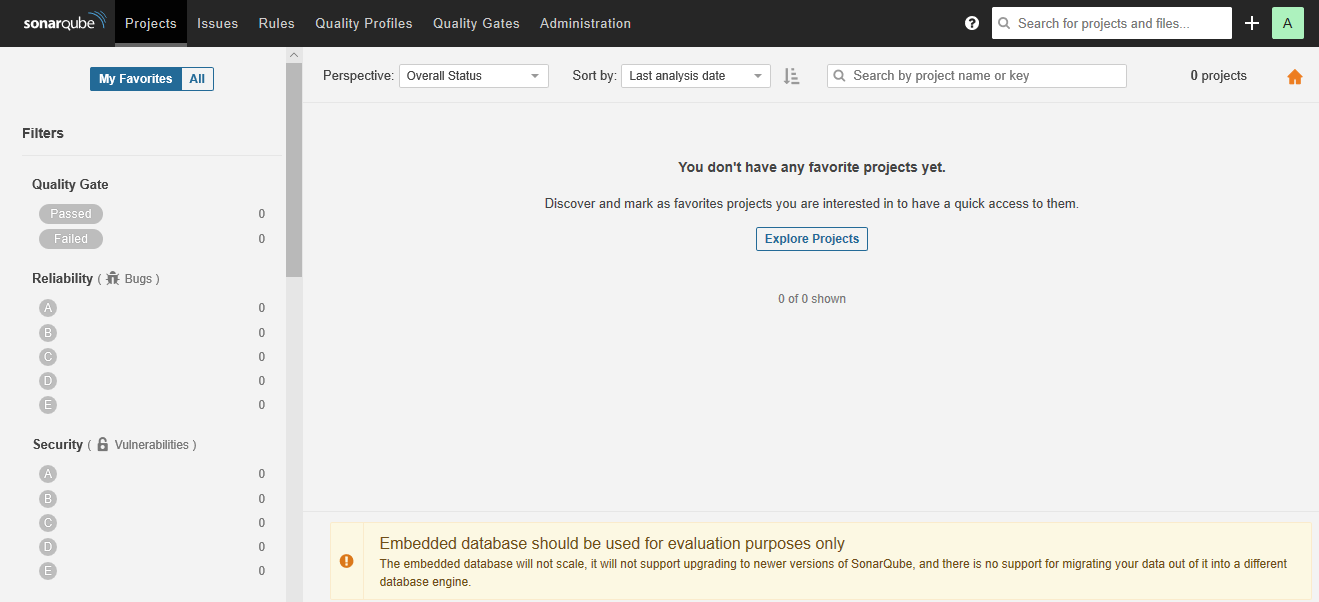
Rys. 2.1.1 Strona główna SonarCube

Następnie trzeba było skonfigurować zainstalowaną na naszych komputerach Javę do wersji 11, gdyż SonarQube obsługuje tą właśnie wersję. Aby prawidłowo ustawić wersję Javy, należało skonfigurować zmienne środowiskowe systemu. Konfiguracja Javy została przedstawiona na Rys. 2.1.2



Rys. 2.1.2 Konfiguracja zmiennych środowiskowych dla Javy 11

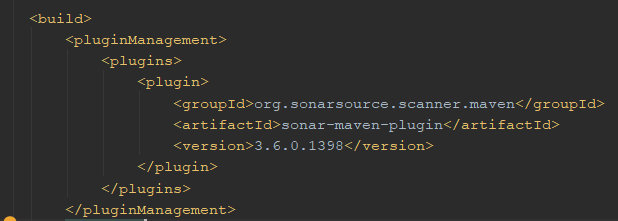
Kolejnym krokiem była instalacja SonarQube, konfiguracja oraz zalogowanie się do http://localhost:9000 jako System Administrator z danymi (login=admin, password=admin). Rezultat pomyślnego zalogowania został przedstawiony na Rys. 2.1.3.



Rys. 2.1.3 Strona startowa SonarQube w localhost:9000

* 1. **Nowy projekt**

Przed wykonaniem analizy, do istniejącego projektu trzeba było wprowadzić konfigurację pom.xml aby możliwe było wygenerowanie raportu:

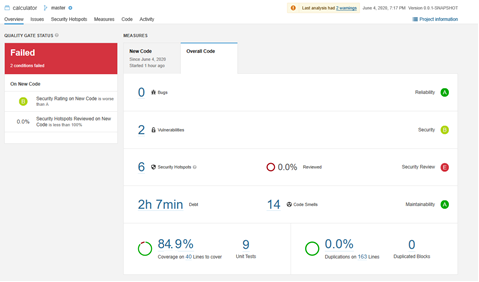


Aby rozpocząć analizę projektu, należało stworzyć nowy projekt w SonarQube i przekleić sekwencję uruchomieniową do Intellij, bo zbudowaniu projektu:

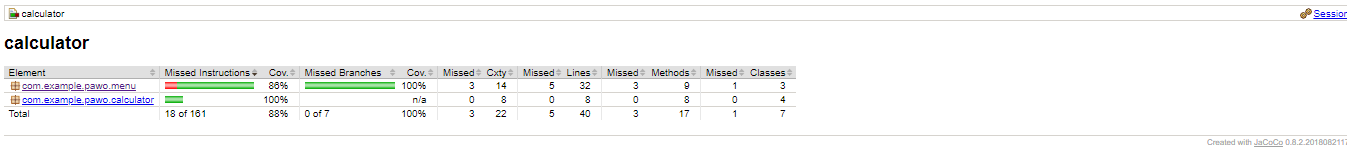
*mvn sonar:sonar -Dsonar.projectKey=lab6Calcu -Dsonar.host.url=http://localhost:9000 -Dsonar.login=e1f649e8940e06ed499293a6dcb4fed6f3648f1c -X*

* 1. **Uruchomienie**

Po uruchomieniu analizy, pojawiła się strona SonarQube z analizą kodu widoczna na Rys. 2.3.1. Można zauważyć, że dany projekt nie przeszedł bram jakości. Jest duże narażenie na bezpieczeństwo projektu oraz 14 możliwych popraw kodu jest sugerowane. Pokrycie kodu jest porównywalne do tego wykrytego przez JaCoCo. (Rys.2.3.2).



Rys. 2.3.1 Analiza kodu za pomocą SonarQube



Rys. 2.3.2 Analiza kodu za pomocą JaCoCo.

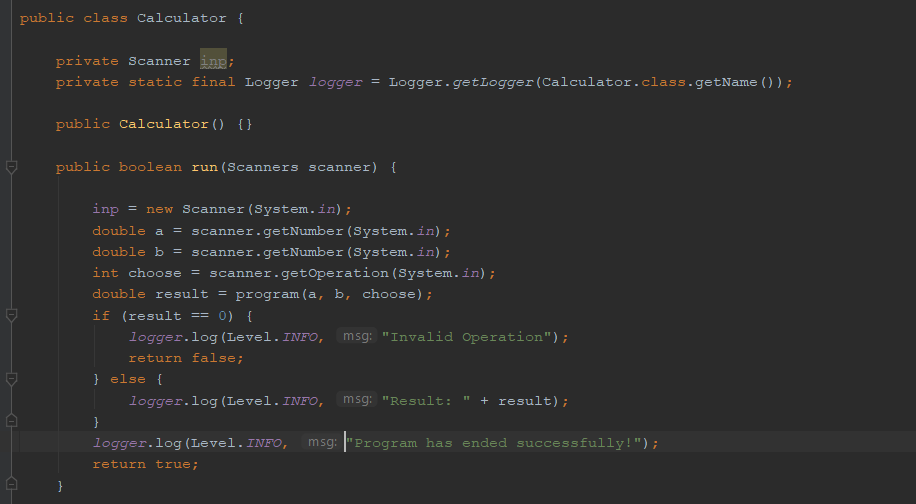
Aplikacja ponadto wykryła wiele błędów, które powinny zostać poprawione w celu lepszej optymalizacji kodu. Błędy zostały przedstawione na Rys. 2.3.3.



Rys. 2.3.3. Błędy wykryte przez SonarQube

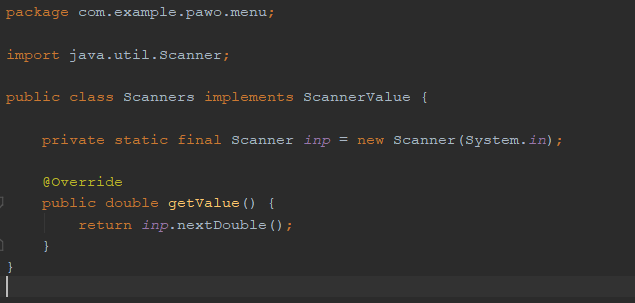
* 1. **Poprawa błędów**

Poprawę błędów rozpocząłem od usunięcie klasy Sytemowej do wyświetlania komunikatów i zamienienie jej na klasę Logger, która jest już w pełni przetestowana. Rezultat można zobaczyć na Rys. 2.4.1.



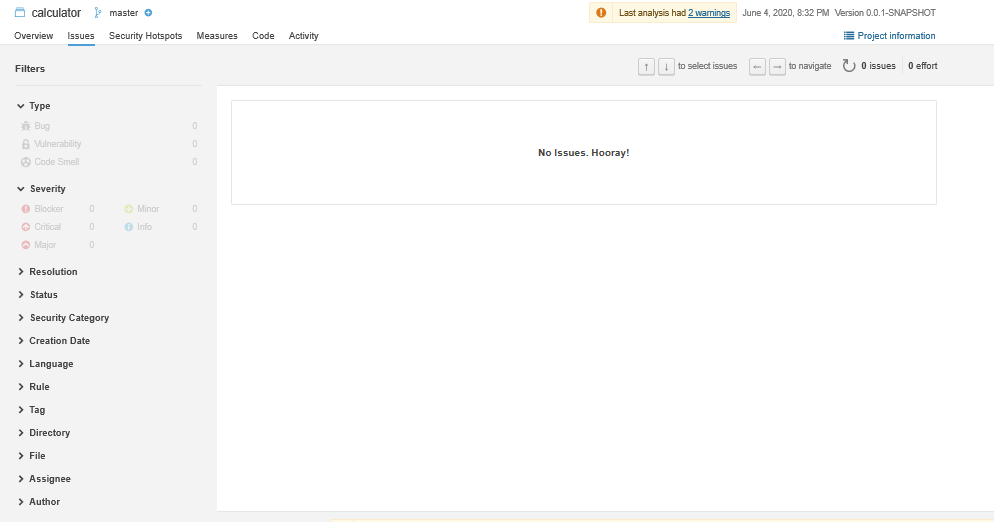
Rys. 2.4.1 Zamiana klasy System.out.println na Logger.log

Kolejnym etapem było usunięcie nieużywanych zmiennych oraz parametrów i zamiana klasy Scanner na klasę finalną. Ponadto Wyodrębniłem klasę Scanner jako abstrakcję w interfejsie, aby nie była testowana, gdyż Scanner to pakiet, który zawiera Java i został już przetestowany, więc pokrycie go testami nie jest konieczne. Rezultat poprawy oraz odwrócenia zależności został przedstawony na Rys. 2.4.2.



Rys. 2.4.2. Odseparowanie klasy Scanner od głównej części oprogramowania

Po zastosowaniu takich poprawek w kodzie, można zauważyć, że stał się optymalny oraz potencjalne błędy zostały usunięte z kodu. Rezultat został przedstawiony na Rys. 2.4.3.



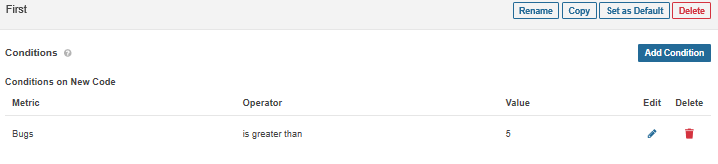
Rys. 2.4.3. Brak błędów w SonarQube

* 1. **Bramy jakości**

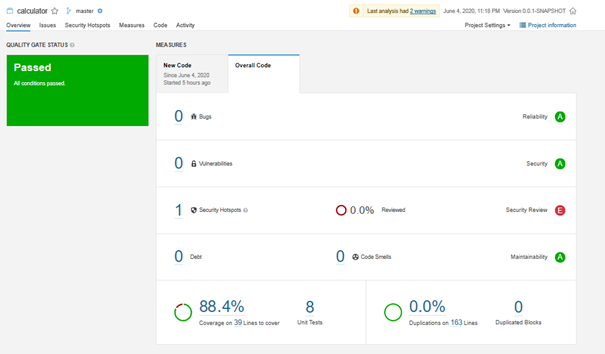
Bramy jakości są wykorzystywane do sprawdzenia czy standardy w kodzie są przestrzegane. Są definiowane jako zestaw miar progowych określonych w danym projekcie. Wstępne dane sprawdzają czy pokrycie programu kodem jest większe niż 80 % i zostały przedstawione na Rys. 2.5.1, ale można zdefiniować własne standardy i zobaczyć wynik aplikacji. W Aplikacji została zdefiniowana nowa brama jakości, sprawdzająca czy występuje więcej niż 5 bugów widoczna na Rys. 2.5.2. Po poprawieniu błędów w kodzie można zauważyć, że dany kod przeszedł przez bramy jakości oraz pokrycie kodu testami wzrosło o około 4 %. Ponadto wszystkie bramy jakości przeszły co można zobaczyć jako końcowy wynik działania narzędzia statystycznego na Rys. 2.5.3.



Rys. 2.5.1 Domyślne bramy jakości w SonarQube



Rys. 2.5.2 Dodanie nowej bramy jakości w SonarQube



Rys. 2.5.3 Końcowy efekt działania programu

1. **Wnioski**

SonarQube to bardzo przydatne i wydajne narzędzie do analizy kodu pod wieloma względami. Wykrywa wszelkie błędy, poprawia czytelność kodu oraz bezpieczeństwo. Aplikacja jest prosta w użyciu, rozbudowana oraz jest w stanie przetwarzać kod w wielu językach. To narzędzie umożliwia stałe monitorowanie kodu oraz poprawę jego implementacji na bieżąco. Nadaje się również podczas współdzielenia kodu z innymi osobami. Dodanie bram jakości w kodzie pozwala na patrzenie na jego jakość pod wieloma względami.

1. **Bibliografia**
2. <https://www.baeldung.com/sonar-qube>
3. <https://docs.sonarqube.org/latest/setup/get-started-2-minutes/>
4. <https://www.swtestacademy.com/sonarqube-tutorial/>