EXPERIMENT NO. 8

NAME : PRANAV POL CLASS : D15A ROLL NO. : 42

**Aim:** Create a Jenkins CICD Pipeline with SonarQube / GitLab Integration to perform a static analysis of the code to detect bugs, code smells, and security vulnerabilities on a sample Web /Java / Python application.

**Theory:**

**Static Application Security Testing (SAST)**

SAST is a testing methodology that analyzes source code to find security vulnerabilities, making applications less susceptible to attacks. It scans the application before the code is compiled, also known as white-box testing.

**Problems SAST Solves**

- **Early Detection**: Identifies vulnerabilities early in the SDLC, allowing developers to resolve issues without breaking builds or passing vulnerabilities to the final release.
- **Real-Time Feedback**: Provides developers with immediate feedback as they code, helping them fix issues before moving to the next phase.
- **Graphical Representations**: Offers visual aids to navigate code, pinpointing exact locations of vulnerabilities and providing guidance on fixes.
- **Regular Scanning**: Should be run regularly, such as during daily/monthly builds, code check-ins, or code releases.

**Importance of SAST**

- **Resource Efficiency**: Developers outnumber security staff, making it challenging to perform manual code reviews. SAST tools can analyze 100% of the codebase quickly.
- **Speed**: Can scan millions of lines of code in minutes, identifying critical vulnerabilities like buffer overflows, SQL injection, and cross-site scripting with high confidence.

**CI/CD Pipeline**

A CI/CD pipeline refers to the Continuous Integration/Continuous Delivery pipeline, which is the backbone of the DevOps approach. It involves a series of tasks connected in sequence to facilitate quick software releases. The pipeline is responsible for building code, running tests, and deploying new software versions.

**SonarQube**

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality. It performs static code analysis, providing detailed reports on bugs, code smells, vulnerabilities, and code duplications. It supports over 25 major programming languages and can be extended with various plugins.

**Benefits of SonarQube**

- **Sustainability**: Reduces complexity, vulnerabilities, and code duplications, optimizing application lifespan.
- **Increased Productivity**: Lowers maintenance costs and risks, reducing the need for extensive code changes.
- **Quality Code**: Ensures code quality control is an integral part of software development.
- **Error Detection**: Automatically detects errors and alerts developers to fix them before output submission.
- **Consistency**: Identifies code criteria breaches, enhancing overall code quality.
- **Business Scaling**: Supports scaling without restrictions.

**Implementation:**

## Prerequisites

1. **Jenkins** installed on your machine.
2. **Docker** installed to run SonarQube.
3. **SonarQube** installed via Docker.

**1. Set Up Jenkins**

- Open Jenkins Dashboard on `localhost:8080` or your configured port.
- Install the necessary plugins:
    - **SonarQube Scanner Plugin**

## 2. Run SonarQube in Docker

Run the following command to start SonarQube in a Docker container:

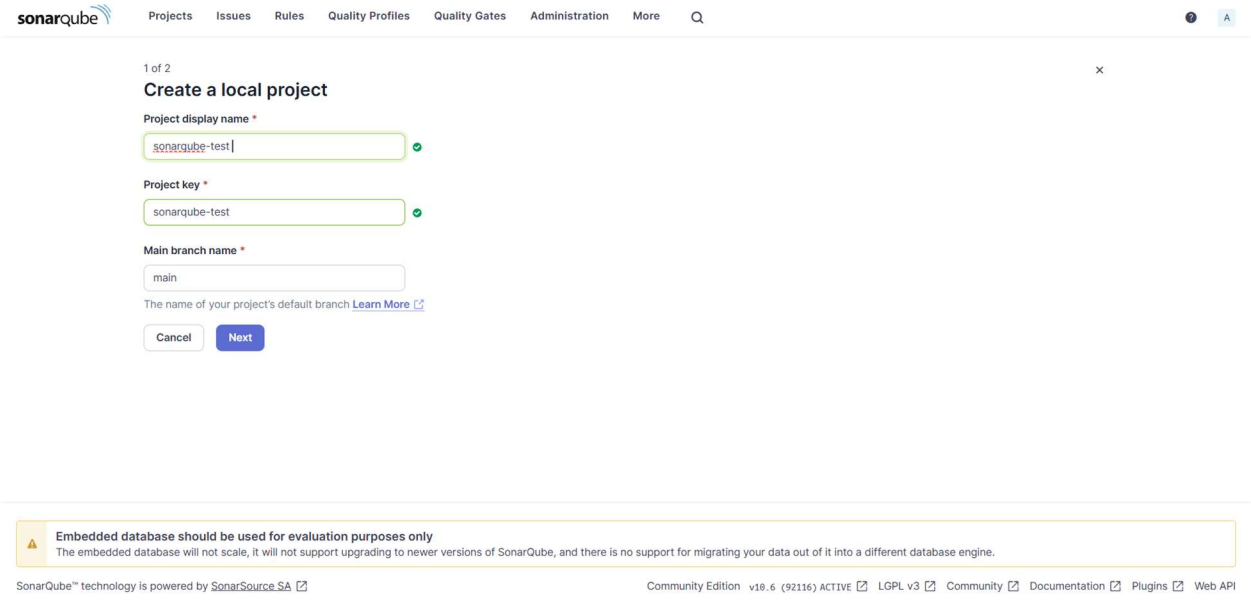
command :
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest

- Check SonarQube status at `http://localhost:9000`.
- Login with your credentials:

```
C:\Users\sbpol>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
7478e0ac0f23: Pull complete
90a925ab929a: Pull complete
7d9a34308537: Pull complete
80338217a4ab: Pull complete
1a5fd5c7e184: Pull complete
7b87d6fa783d: Pull complete
bd819c9b5ead: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:72e9feec71242af83faf65f95a40d5e3bb2822a6c3b2cda8568790f3d31aecde
Status: Downloaded newer image for sonarqube:latest
2f213117ce50f08304d681a60dc0e2a4dd6c3c8e46f5725be7fb40fd0d48bb5d
```

## 3. Create a Project in SonarQube

- Go to **Projects** > **Create Project**.
- Name the project (e.g., `sonarqube-test`).

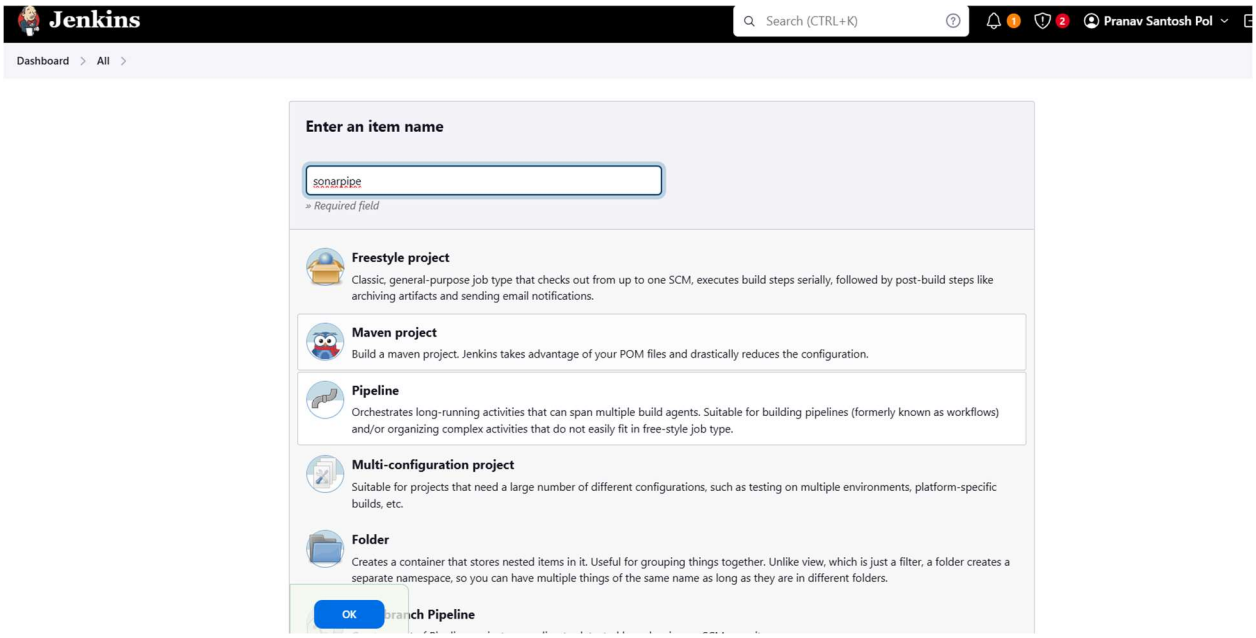


### 4. Generate SonarQube Token

- Go to **My Account** > **Security** > **Generate Tokens**.
- Copy the generated token for later use.

### 5. Create a Jenkins Pipeline

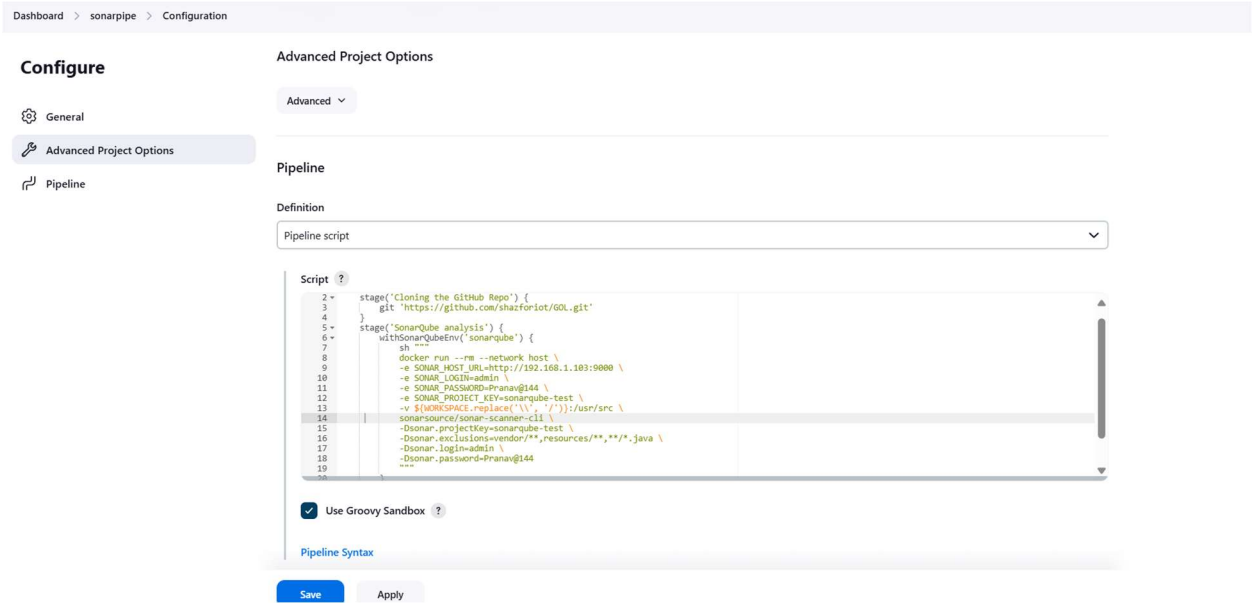
- Go to Jenkins Dashboard, click **New Item**, and select **Pipeline**.




### 6. Under Pipeline Script, enter the following script:

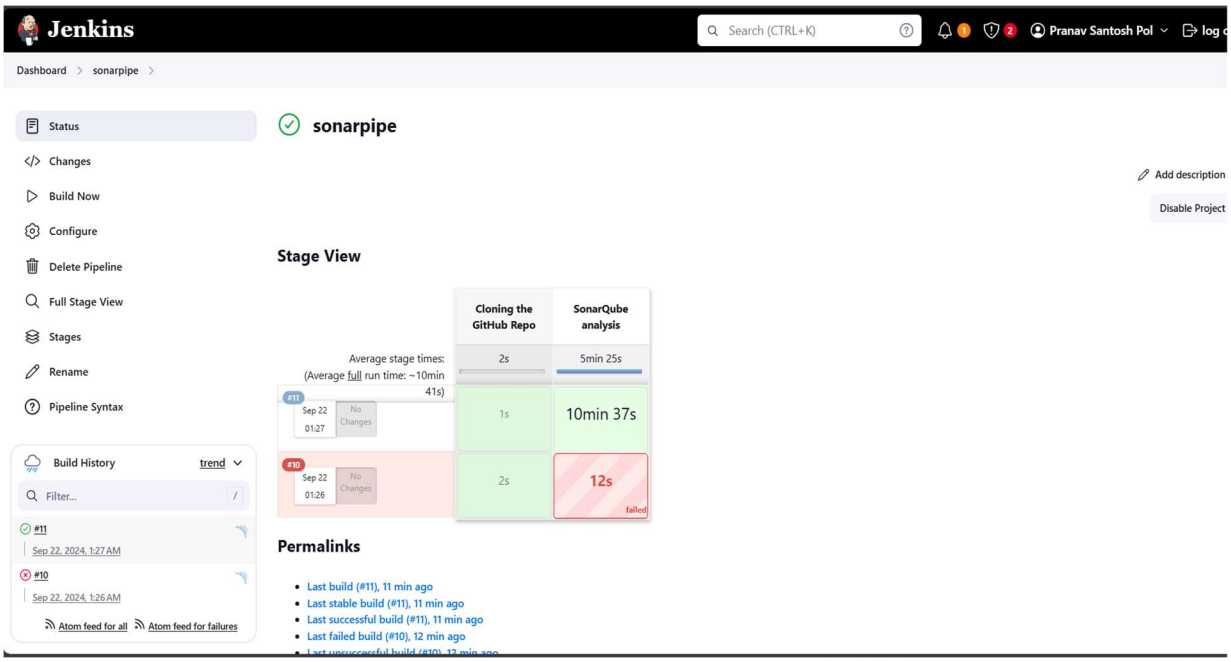
```
docker network create sonarnet

node {
    stage('Cloning the GitHub Repo') {
        git 'https://github.com/shazforiot/GOL.git'
    }
    stage('SonarQube analysis') {
        withSonarQubeEnv('sonarqube') {
            sh """
            docker run --rm --network host \
            -e SONAR_HOST_URL=http://<ip_address>:9000 \
            -e SONAR_LOGIN=admin \
            -e SONAR_PASSWORD=<Sonarqube_password> \
            -e SONAR_PROJECT_KEY=sonarqube-test \
            -v ${WORKSPACE.replace('\\', '/')}:/usr/src \
            sonarsource/sonar-scanner-cli \
            -Dsonar.projectKey=sonarqube-test \
```

```
            -Dsonar.exclusions=vendor/**,resources/**,**/*.java \
            -Dsonar.login=admin \
            -Dsonar.password=<Sonarqube_password>
            """
        }
    }
}
```



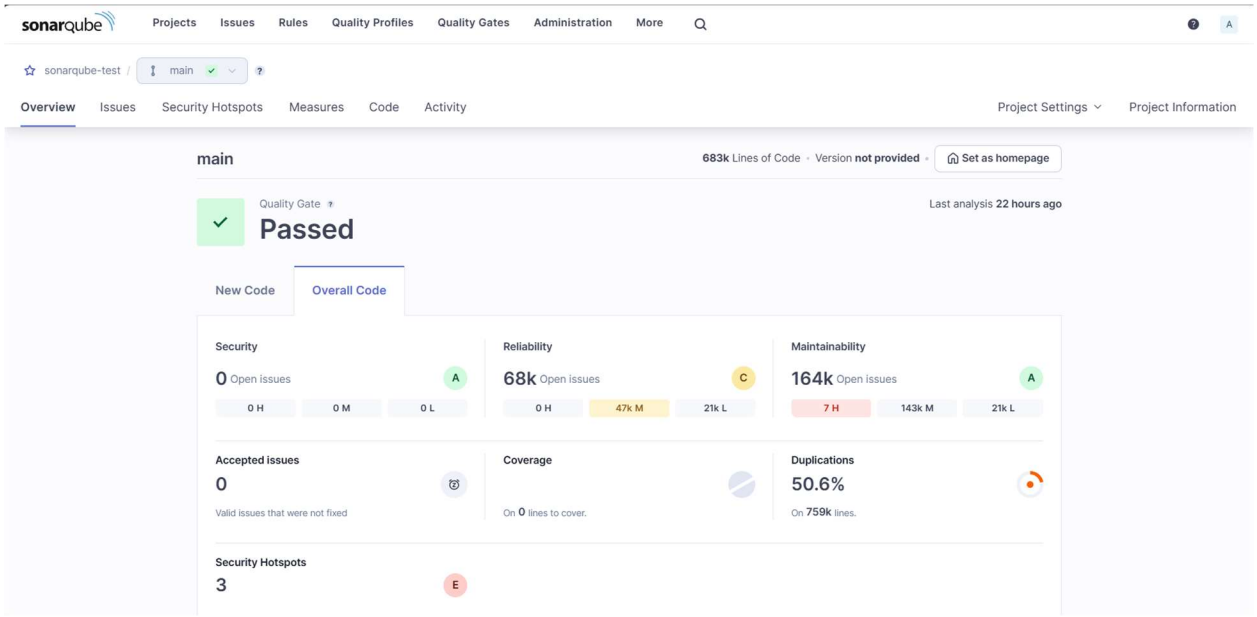

## 7. Run the Pipeline

- Save the pipeline and click **Build Now.**
- Monitor the console output for any errors.



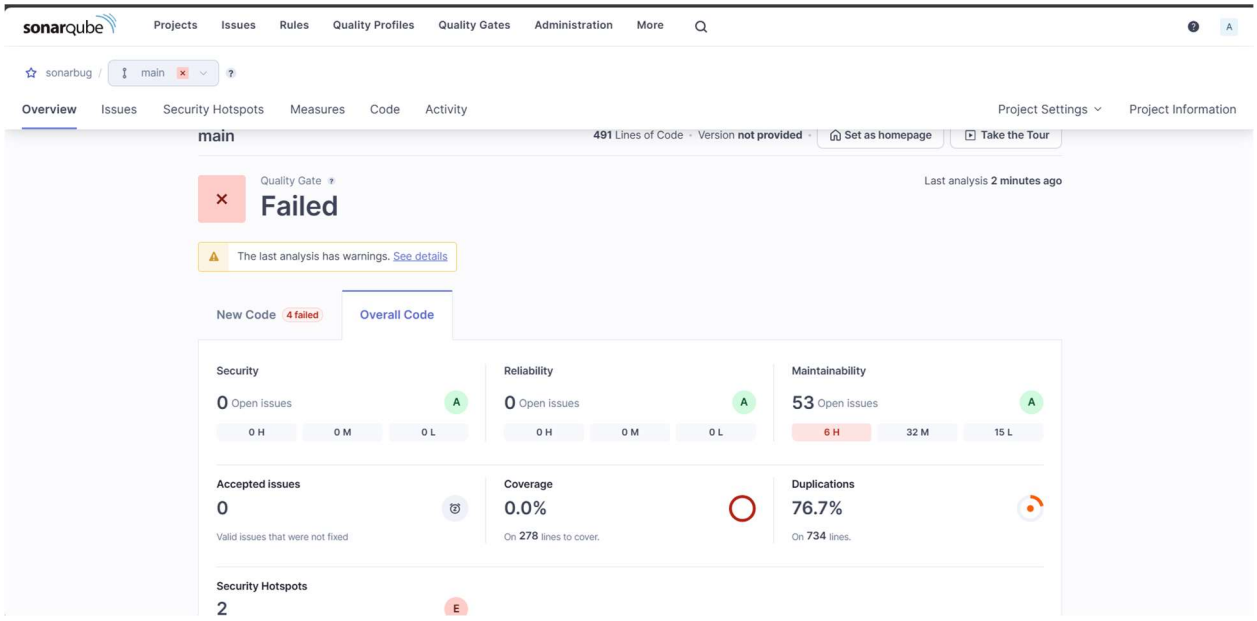
## 9. Check SonarQube for Analysis Results

- Go to your SonarQube dashboard and check the project for issues such as bugs, code smells, and security vulnerabilities.

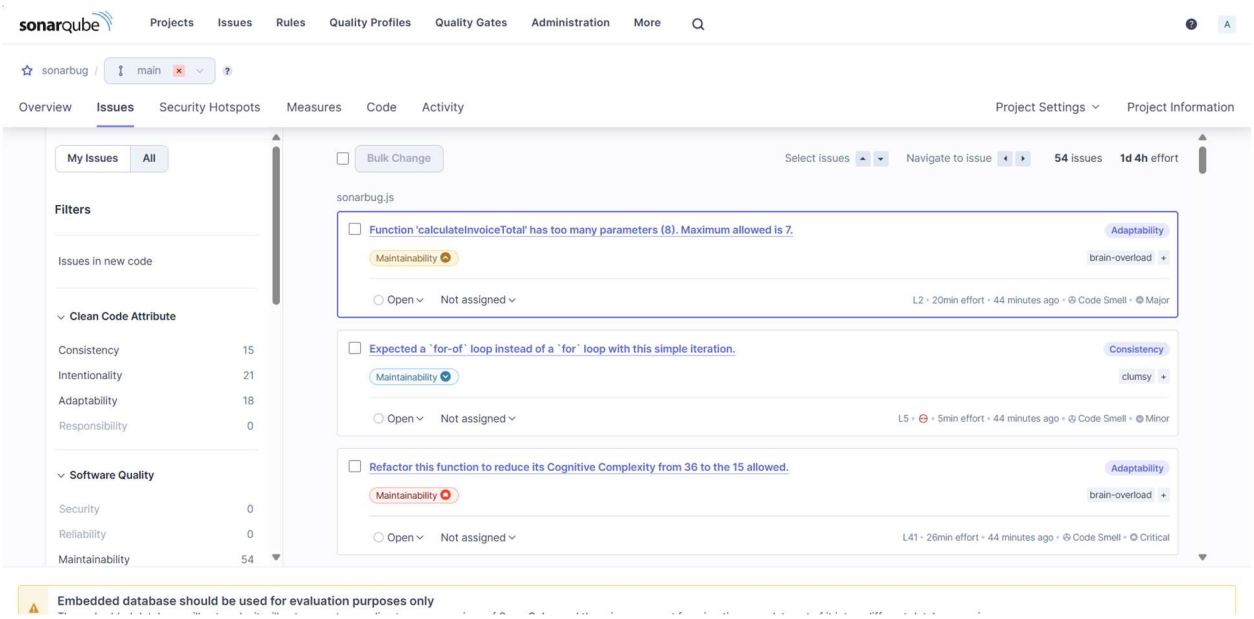



## 10. Checking SonarQube for Analysis Results of a Code File with Bugs , Code Smells, Security Vulnerabilities, Cyclomatic Complexities and Duplicates .

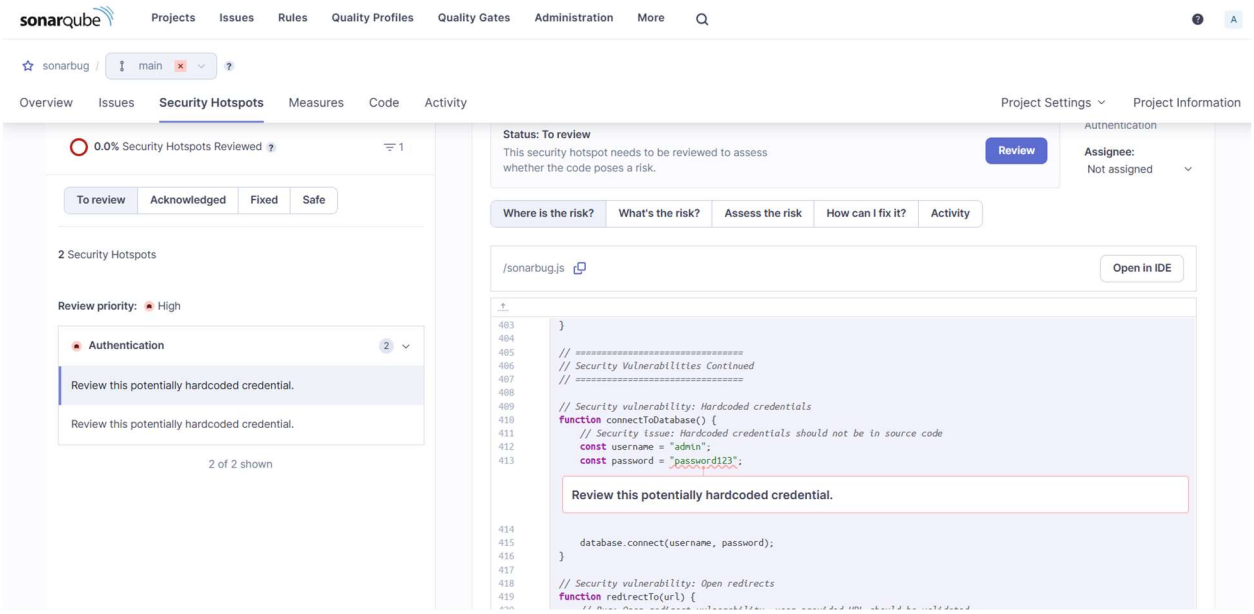
- Overview -



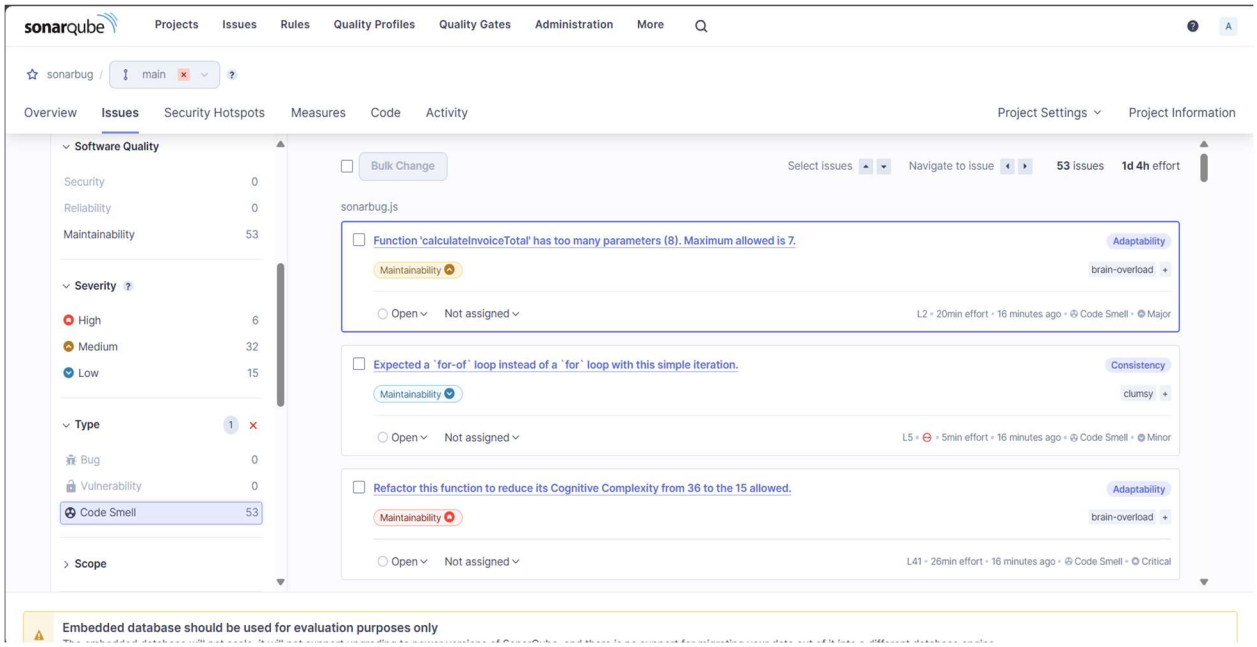
- Issues -

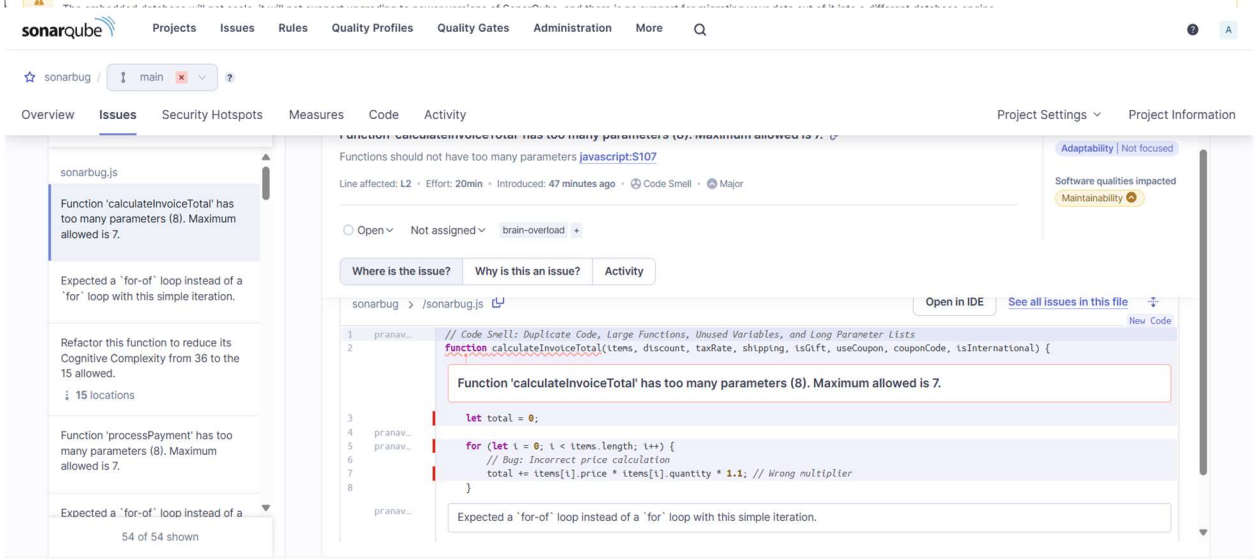



- Security Hotspot (Security Vulnerabilities) -

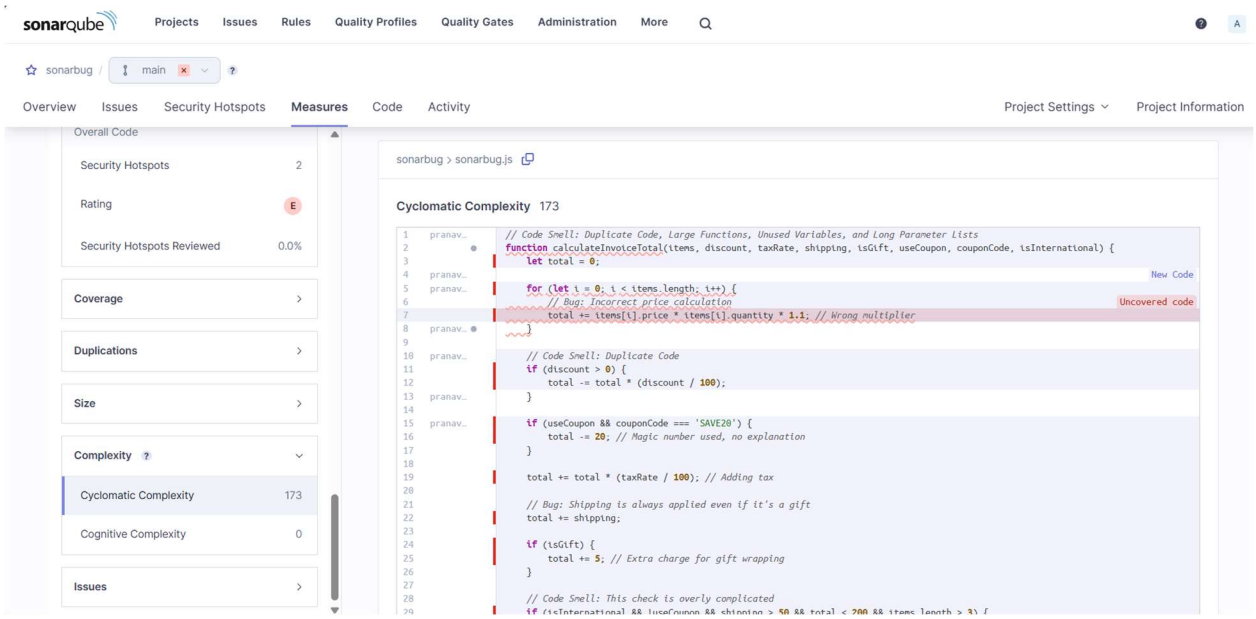


- Codesmells -

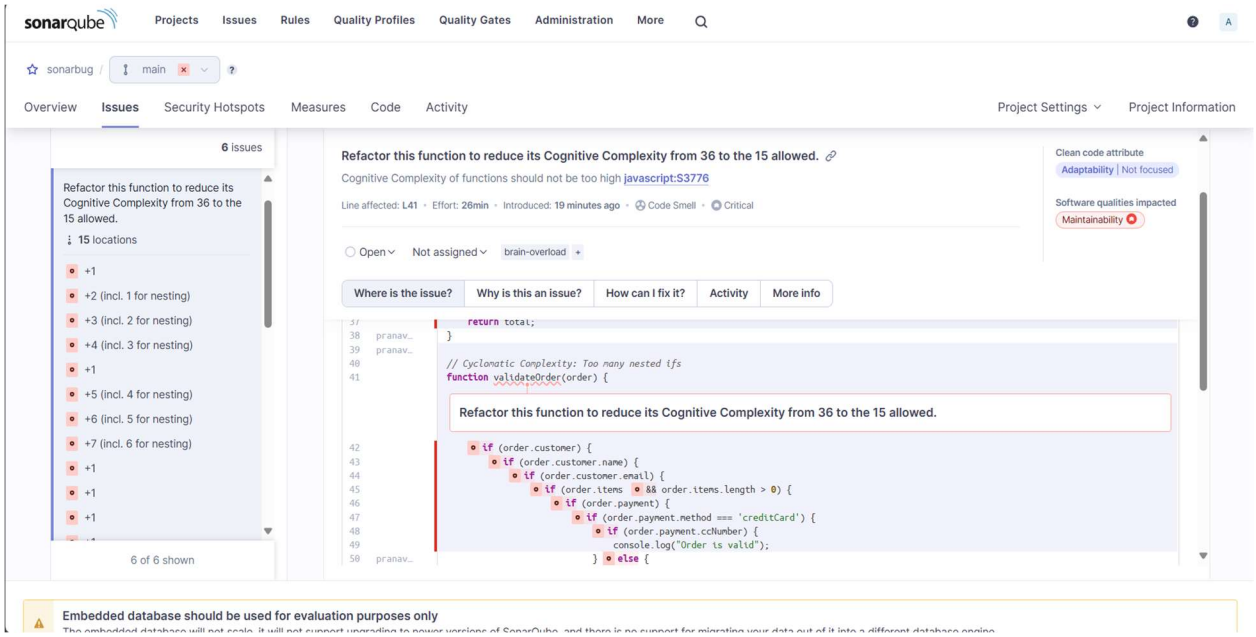

sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
sonarbug / main
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
Software Quality
Security 0
Reliability 0
Maintainability 53
Severity
High 6
Medium 32
Low 15
Type
Bug 0
Vulnerability 0
Code Smell 53
Scope
Bulk Change
Select issues
Navigate to issue
53 issues
1d 4h effort
sonarbug.js
Function 'calculateInvoiceTotal' has too many parameters (8). Maximum allowed is 7.
Adaptability
Maintainability
brain-overload
Open
Not assigned
L2 • 20min effort • 16 minutes ago • Code Smell • Major
Expected a `for-of` loop instead of a `for` loop with this simple iteration.
Consistency
clumsy
L5 • 5min effort • 16 minutes ago • Code Smell • Minor
Refactor this function to reduce its Cognitive Complexity from 36 to the 15 allowed.
L41 • 26min effort • 16 minutes ago • Code Smell • Critical
Embedded database should be used for evaluation purposes only


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
sonarbug / main
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
sonarbug.js
Function 'calculateInvoiceTotal' has too many parameters (8). Maximum allowed is 7.
Expected a `for-of` loop instead of a `for` loop with this simple iteration.
Refactor this function to reduce its Cognitive Complexity from 36 to the 15 allowed.
15 locations
Function 'processPayment' has too many parameters (8). Maximum allowed is 7.
Expected a `for-of` loop instead of a
54 of 54 shown
Functions should not have too many parameters javascript:S107
Line affected: L2 • Effort: 20min • Introduced: 47 minutes ago • Code Smell • Major
Adaptability | Not focused
Software qualities impacted
Maintainability
Open
Not assigned
brain-overload
Where is the issue?
Why is this an issue?
Activity
sonarbug > /sonarbug.js
Open in IDE
See all issues in this file
New Code
// Code Smell: Duplicate Code, Large Functions, Unused Variables, and Long Parameter Lists
function calculateInvoiceTotal(items, discount, taxRate, shipping, isGift, useCoupon, couponCode, isInternational) {
Function 'calculateInvoiceTotal' has too many parameters (8). Maximum allowed is 7.
let total = 0;
for (let i = 0; i < items.length; i++) {
// Bug: Incorrect price calculation
total += items[i].price * items[i].quantity * 1.1; // Wrong multiplier
}
Expected a `for-of` loop instead of a `for` loop with this simple iteration.
Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

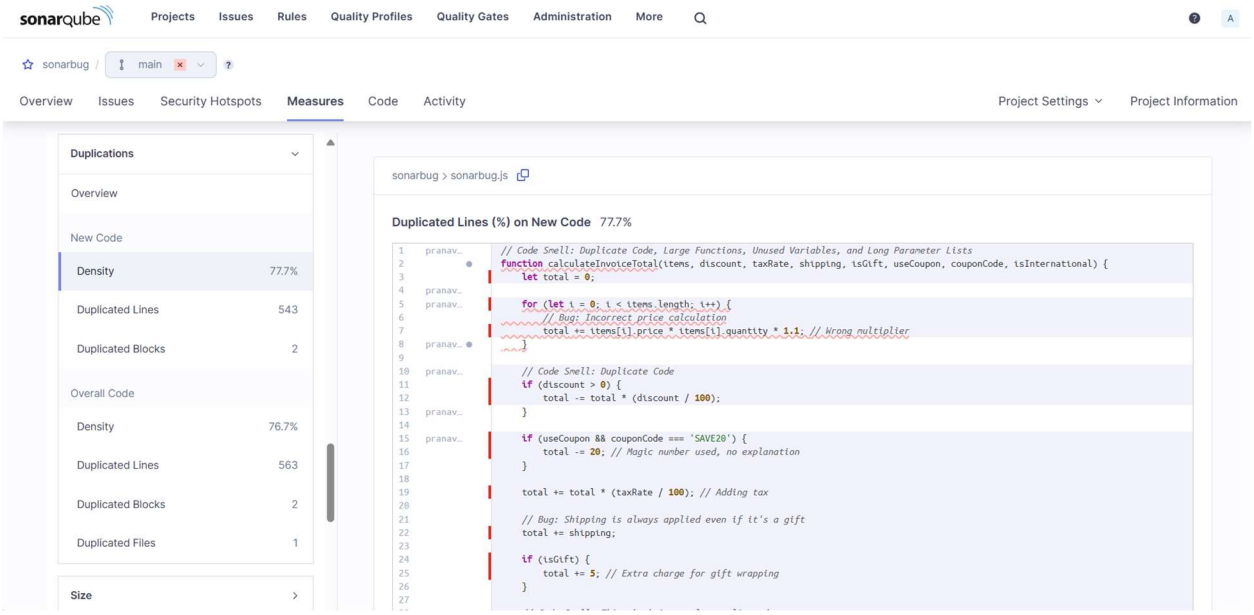
- Cyclomatic Complexity -




- Cognitive Complexity -



- Duplications -




**Conclusion:**

In this experiment, we performed a static analysis of the code to detect bugs, code smells, and security vulnerabilities on our sample codes.