Name: Hrishikesh Kumbhar

Div: D15A

Roll no: 32

Sub: Advanced DevOps

Experiment No: 8

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.

Steps to create a Jenkins CI/CD Pipeline and use SonarQube to perform SAST

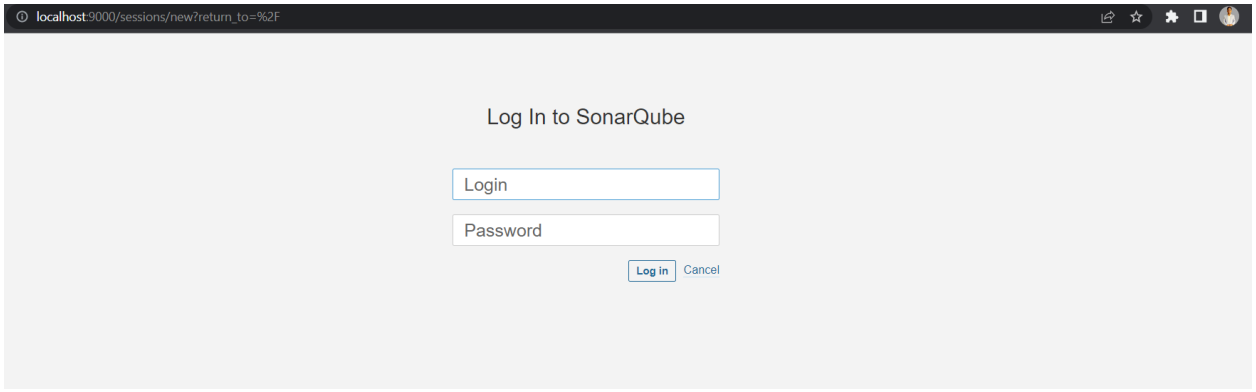
Step 1. Open up Jenkins Dashboard on localhost, port 8080 or whichever port it is at for you.

Step 2. Run SonarQube in a Docker container using this command -

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

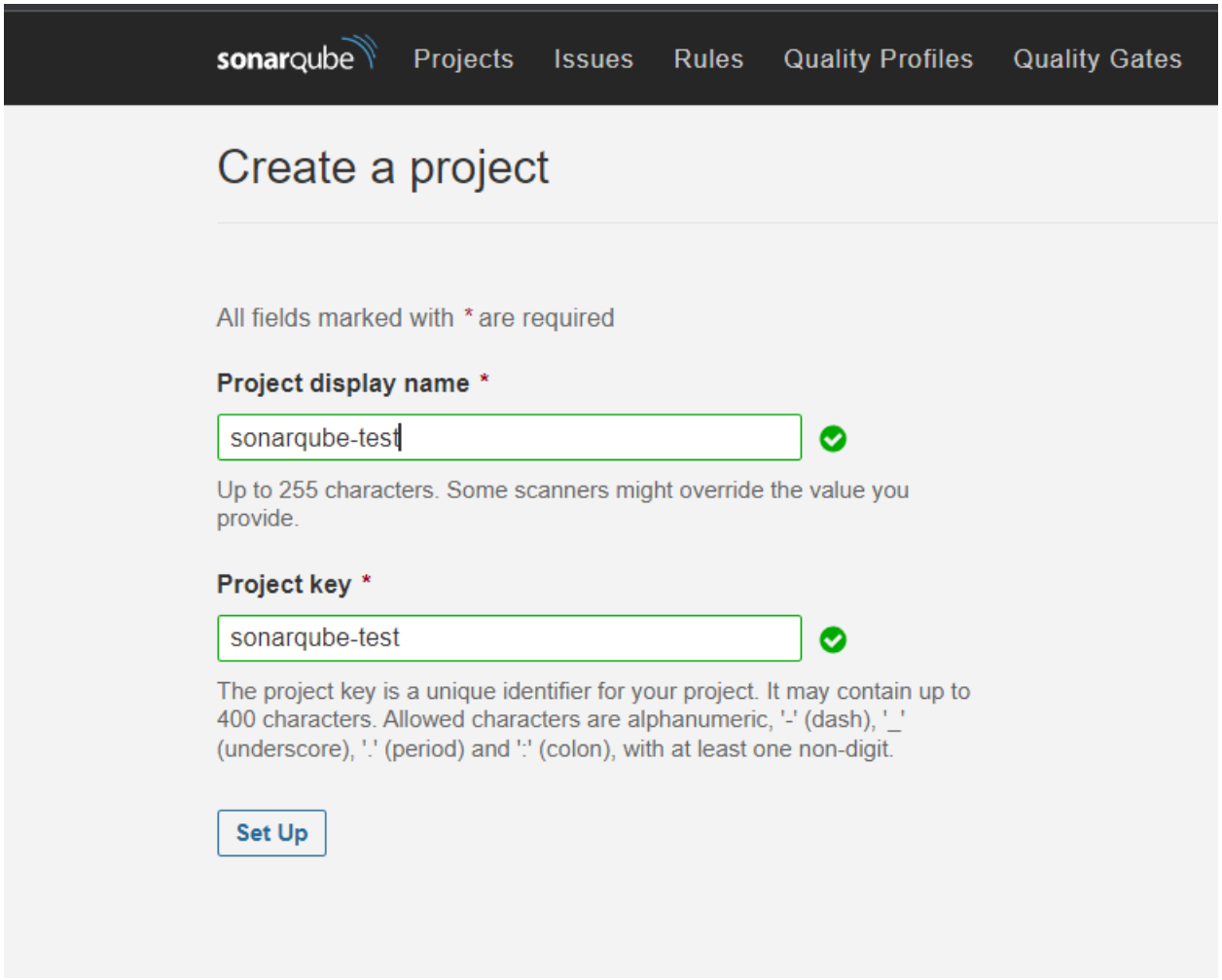
```
C:\Users\acer>docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
fac04ce1b97fadf955ddf9cc9c4f041b2ccac89196d5f93a0bf8199a71c404d2

C:\Users\acer>
```

Step 3. Once the container is up and running, you can check the status of SonarQube at localhost port 9000.

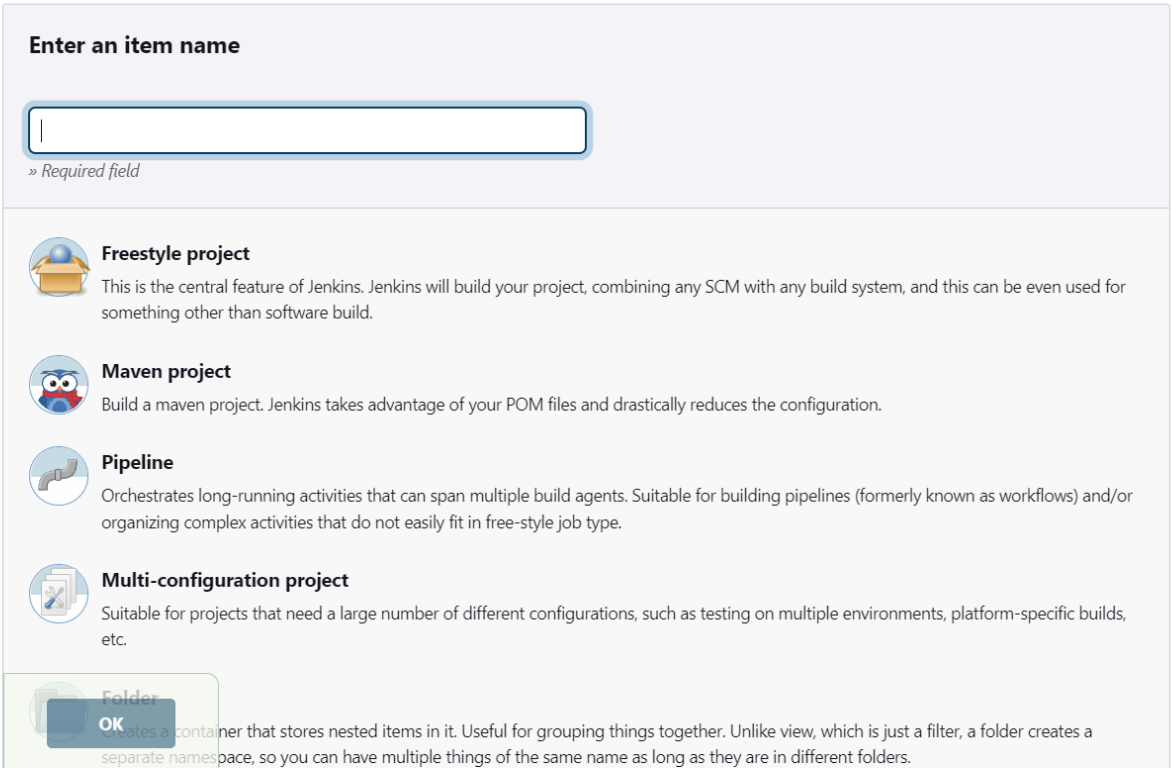


Step 4. Login to SonarQube

Step 5. Create a manual project in SonarQube with the name sonarqube-test



Step 6. Create a New Item in Jenkins, choose Pipeline.



Step 7. Under Pipeline Script, enter the following -

```
node{
    stage('cloning the github repo'){
        git 'https://github.com/shazforiot/GOL.git'
    }
    stage('Sonarqube analysis'){
        withSonarQubeEnv('sonarqube') {
        sh
"/c/ProgramData/Jenkins/.jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube/bin//sonar-scanner \
        -D sonar.login=admin \
        -D sonar.password=hrishi156 \
        -D sonar.projectKey=sonarqube-test \
        -D sonar.exclusions=vendor/**,resources/**,**/*.java \
        -D sonar.host.url=http://127.0.0.1:9000/"
            }
    }
```

```
}
```

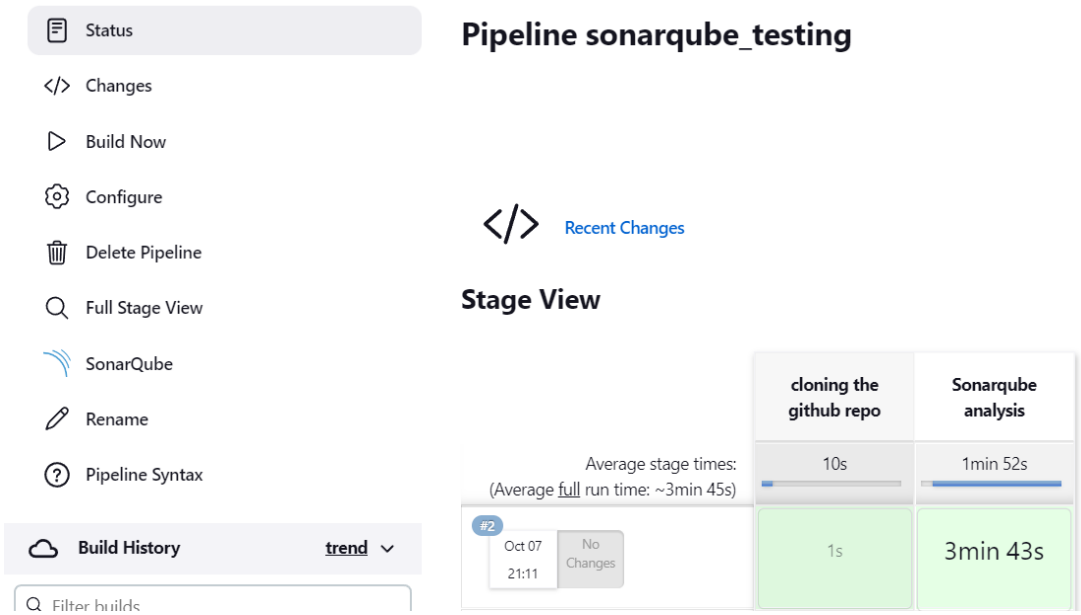
Pipeline script

Script ?

```
node{
    stage('cloning the github repo'){
        git 'https://github.com/shazforiot/GOL.git'
    }
    stage('Sonarqube analysis'){
        withSonarQubeEnv('sonarqube') {
        sh "/c/ProgramData/Jenkins/.jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/sonarqube/bin//sonar-scanner \
        -D sonar.login=admin \
        -D sonar.password=hrishi156 \
        -D sonar.projectKey=sonarqube-test \
        -D sonar.exclusions=vendor/**,resources/**,**/*.java \
        -D sonar.host.url=http://127.0.0.1:9000/"
            }
    }
}
```

It is a java sample project which has a lot of repetitions and issues that will be detected by SonarQube.

Step 8. Run The Build.

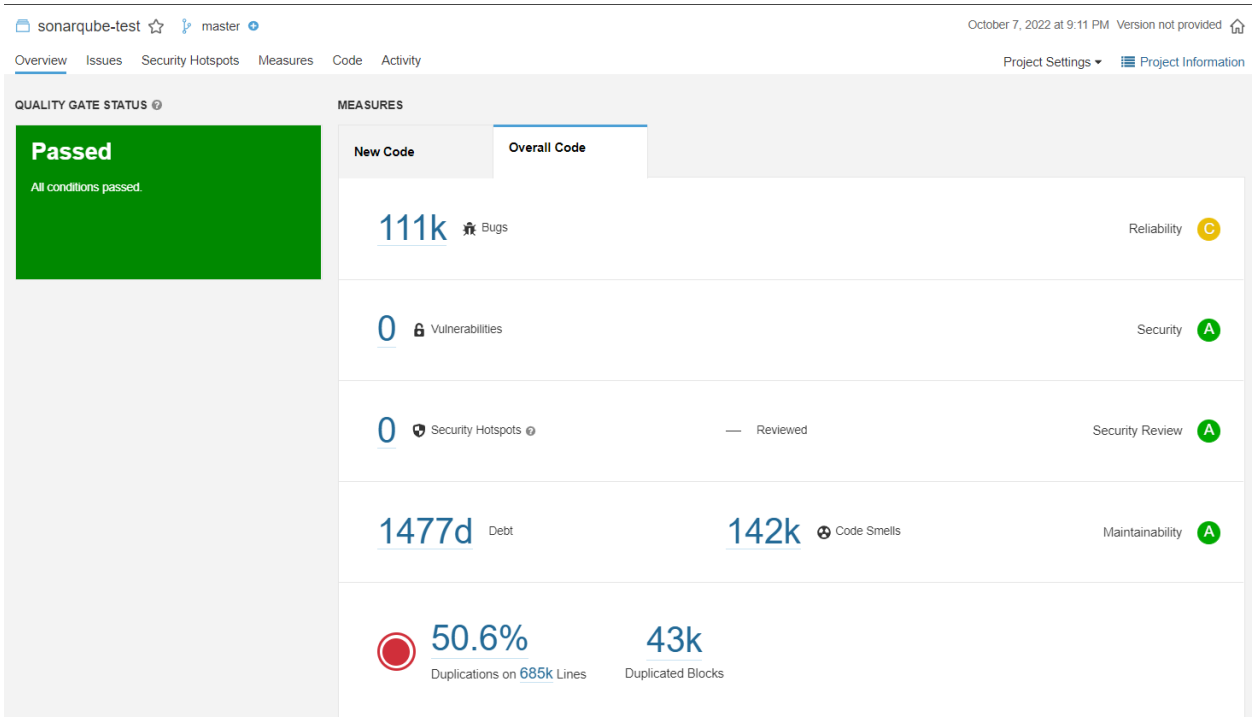


Step 9. Check the console output once the build is complete.

Skipping 3,970 KB.. Full Log

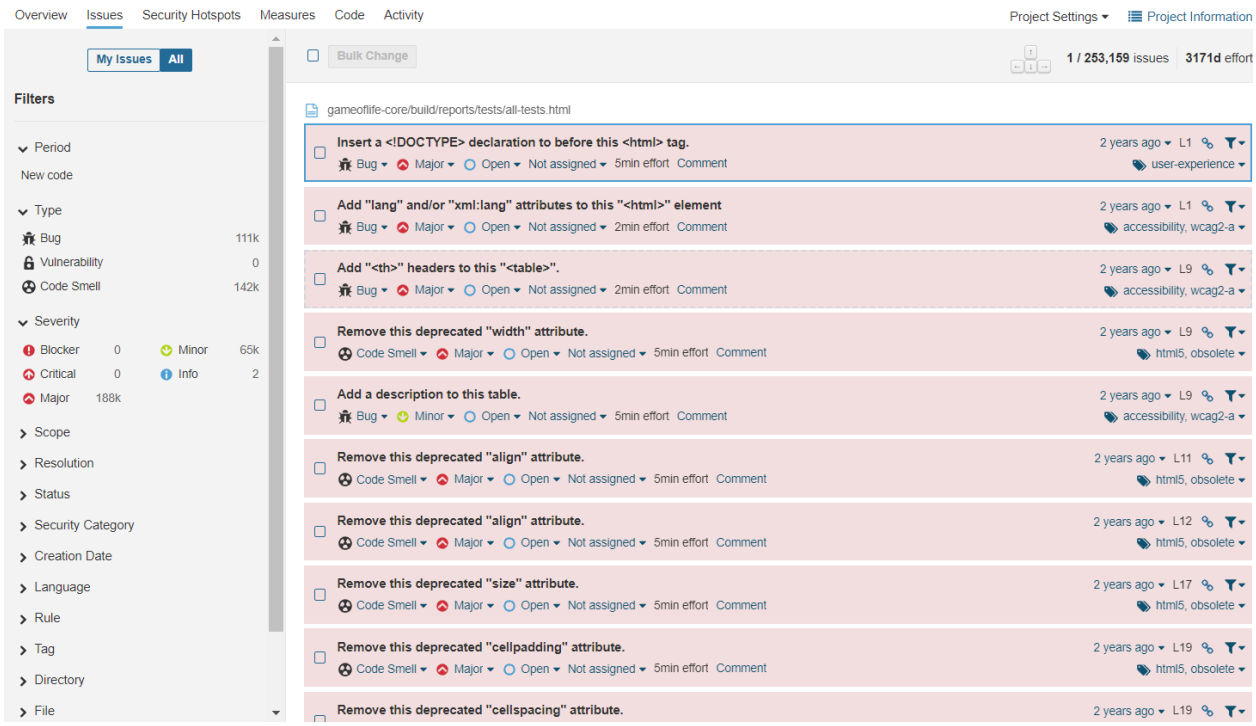
```
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 230. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 240. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 32. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 227. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 230. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 40. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 41. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/modifiers/BeanShellPreProcessorBeanInfo.html for block at line 17. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/gui/util/TextAreaCellRenderer.html for block at line 296. Keep only the first 100 references.
WARN: Too many duplication references on file gameoflife-web/tools/jmeter/docs/api/org/apache/jmeter/gui/util/TextAreaCellRenderer.html for block at line 75. Keep only the first 100 references.
INFO: CPD Executor CPD calculation finished (done) | time=68375ms
INFO: Analysis report generated in 1479ms, dir size=129.8 MB
INFO: Analysis report compressed in 8051ms, zip size=29.8 MB
INFO: Analysis report uploaded in 647ms
INFO: ANALYSIS SUCCESSFUL, you can find the results at: http://127.0.0.1:9000/dashboard?id=sonarqube-test
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
INFO: More about the report processing at http://127.0.0.1:9000/api/ce/task?id=AYOzHqoyc4o3ZFQUtxSH
INFO: Analysis total time: 3:39.680 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 3:41.441s
INFO: Final Memory: 16M/74M
INFO: ------------------------------------------------------------------------
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

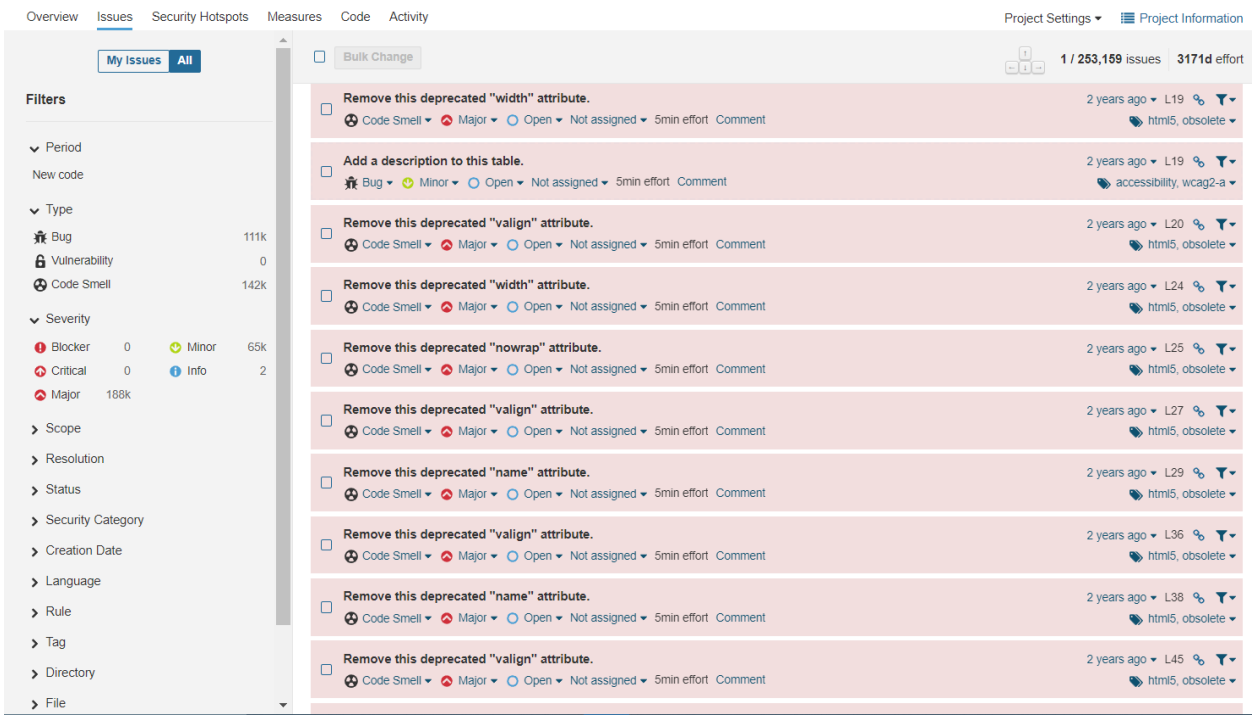
Step 10. After that, check the project in SonarQube.



Under different tabs, check all different issues with the code.

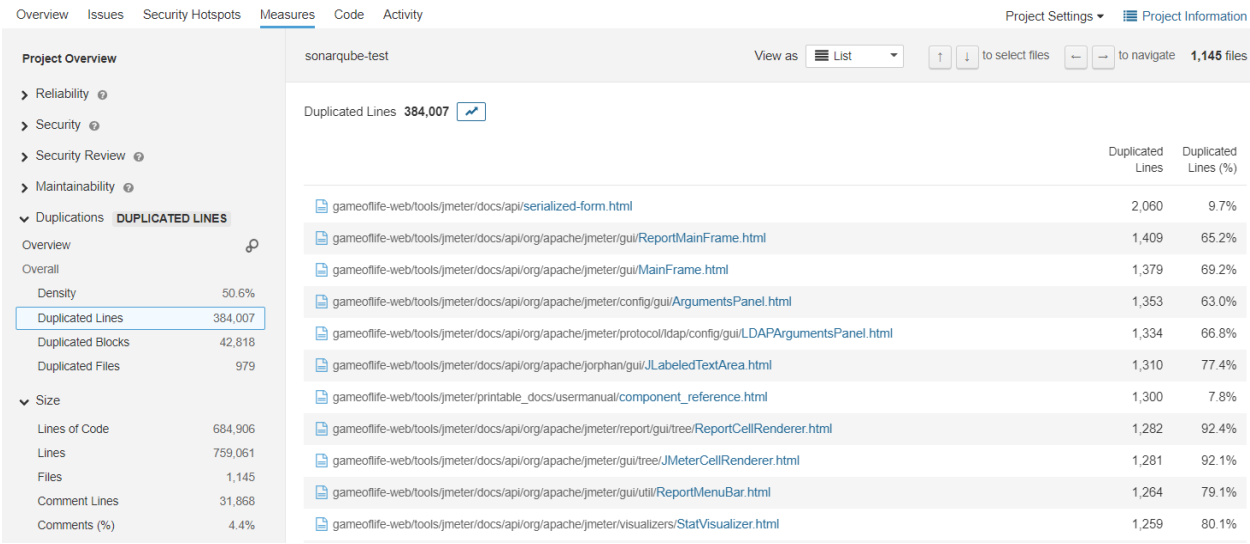
Step 11. Code Problems -



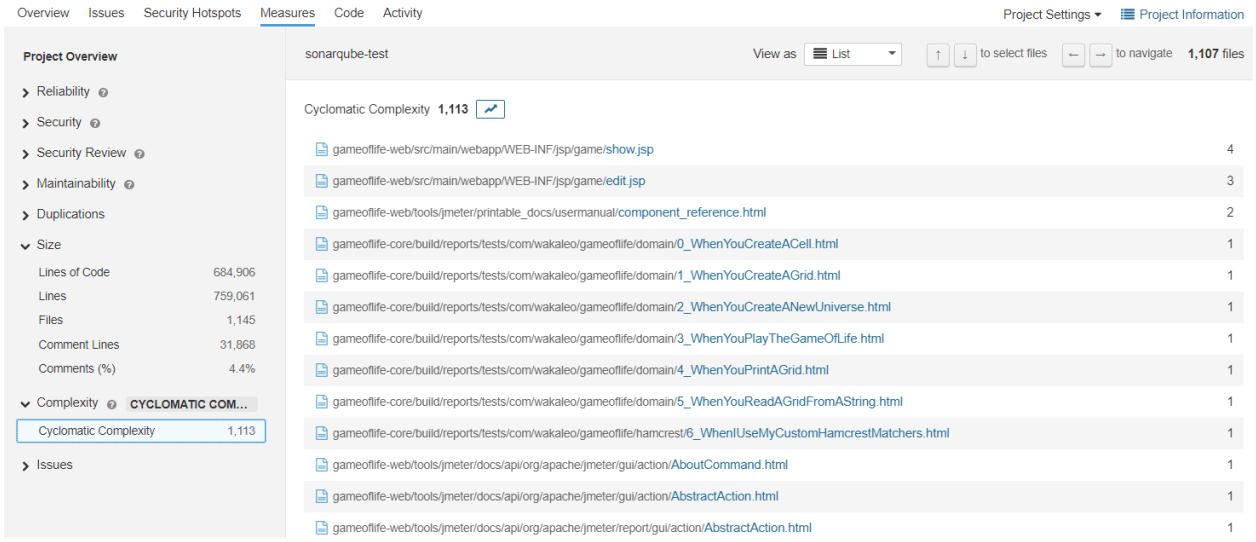


Bugs and Code Smells

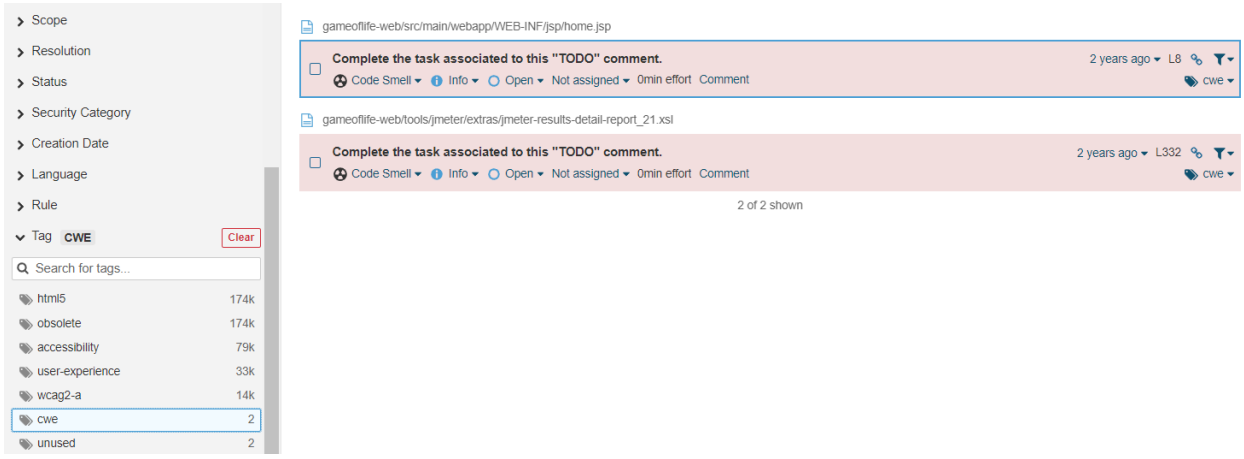
Duplicates



Cyclomatic Complexities



Unfinished TODOs



In this way, we have created a CI/CD Pipeline with Jenkins and integrated it with SonarQube to find issues in the code like bugs, code smells, duplicates, cyclomatic complexities, etc.

Conclusion:

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