St. Francis Institute of Technology, Mumbai-400 103

Department of Information Technology

A.Y. 2023-2024

Class: TE-ITA/B, Semester: V

Subject: **Advanced DevOps Lab**

Experiment – 10: To test code (python files) with SonarQube and observe, analyse the results.

1. Aim: To perform analysis of the code to detect bugs, code smells, security vulnerabilities on a Python application.

2. Objectives: After study of this experiment, the student will be able to

● Analys the source code.

3. Lab objective mapped :ITL504.4: To identify and remediate application vulnerabilities earlier and help integrate security in the development process using SAST Techniques.

4. Prerequisite: Fundamentals of Software Testing

5. Requirements: PC and Internet, SonarQube, Sonar Scanner, Java JDK

6. Pre-Experiment Exercise:

Brief Theory:

SonarQube is a static analysis code tool. It basically goes through developers' code and identifies errors at the early stage. It is an open-source static testing analysis software. It is used by developers to manage source code quality and consistency. Some of the code quality checks are:

● Potential Bugs

● Code defects to design inefficiencies — Identifies the code which is not compatible with the design structure of the application.

● Code duplication — Code duplications take a lot of memory. The tool can identify those things.

● Lack of Test Coverage — There maybe we are not enough tests written to application. The tool can identify those things.

● Excess complexity — Tool can identify a much more simple may to complex code segments.

Features of SonarQube

- **It can work in 25 different languages**. (Java, .NET, JavaScript, COBOL, PHP, Python, C++, Ruby, Kotlin and Scala)
- **Identify tricky issues.**

Detect Bugs — SonarQube can detect tricky bugs or can raise on pieces of code that it thinks is faulty.

Code Smells — Code smells are the characteristics of a code that indicates that there might be a problem caused by the code in the future. But smells aren't necessarily bad, sometimes they are how the functionality works and there is nothing that can be done about it. This is something called best practices.

Security Vulnerability — SonarQube can detect security issues that code may face. As an example If a developer forgets to close an open a SQL database OR If important details like username and password have been directly written in the code. Then SonarQube can identify these things. Because leaving SQL database open can cause issues in the source code and you definitely do not want to write username and password directly in the code. You should inject them.

Activate Rules Needed — You can create and maintain different sets of rules that are specific to particular projects, these are known as **Quality Profiles**. This means a team or project should follow specific rules. Then we can create a Quality profile in SonarQube.

Execution Path — Whenever there is Data flow in your program, and there is a lot of involvement between the different Modules. SonarQube can figure out if there are any tricky bugs in these execution paths. When a company works on an application there obviously have a code pipeline a data flow in the program. SonarQube when it integrated to Jenkins or any deployment tool it works by itself it keeps looking on errors and bugs. Sometimes SonarQube identifies these tricky bugs in these pathways. Suppose an error that depends on Module that is way back in the code pipeline or way back in the data flow in the program then can figure out the integration error that happens between these.

- **Enhanced Workflow (Ensure Better CI/CD)**

Automated Code Analysis — Keep working in the background from the development phase itself, monitoring and identify errors. SopnarQube can be automated by integrating with the deployment tool or integration tool and it will keep working on the background and it finds all the errors, the Code Smells, Technical Dept by itself.

Get access through Webhooks and API — To initiate tests do not need to come to SonarQube directly, we can do that through an API call. You do not need to install SonarQube directly. You can just use APIs and call them.

Integrate GitHub — It can be directly integrated with your choice of version control software. You can find errors as well as the version of the code you are using.

Analyze branches and Decorate pull requests — It gives us a branch Level analysis. As an example, it does not just analyze the master branch it also analyzes the other branches, identifying any errors.

- **Built-in methodology**

Discovery Memory Leaks — It can show the memory leaks in your application if the application has a tendency to fail or go out of memory. This generally will happen slowly happen over a period of time.

Good Visualizer — It has a good way visualizing, it gives simple overviews of the overall health of the code. After the code has been developed a proper record of how the core is been performing created by SonarQube and it will be presenting on the Dashboard. So the team Lead or the Developer himself can go through it.

Enforces a quality gate — It can enforce a quality gate, you can tell SonarQube based on your requirements and practices what code is wrong and what is correct.

Digs into issues — If it shows that there is a problem SonarQube allows you to go and directly check it out from the summary report or from one code file to another. In the SonarQube summary dashboard, you can see furthermore details of the errors bu just clicking on the error.

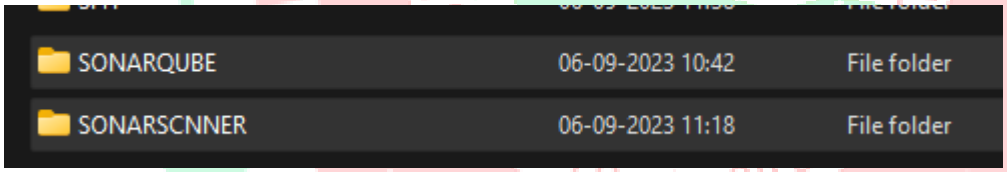
Plugins for IDEs — It has a plugin called "SonarLint" which helps SonarQube to integrate itself with an IDE. Which means there is no need to install the whole SonarQube package.

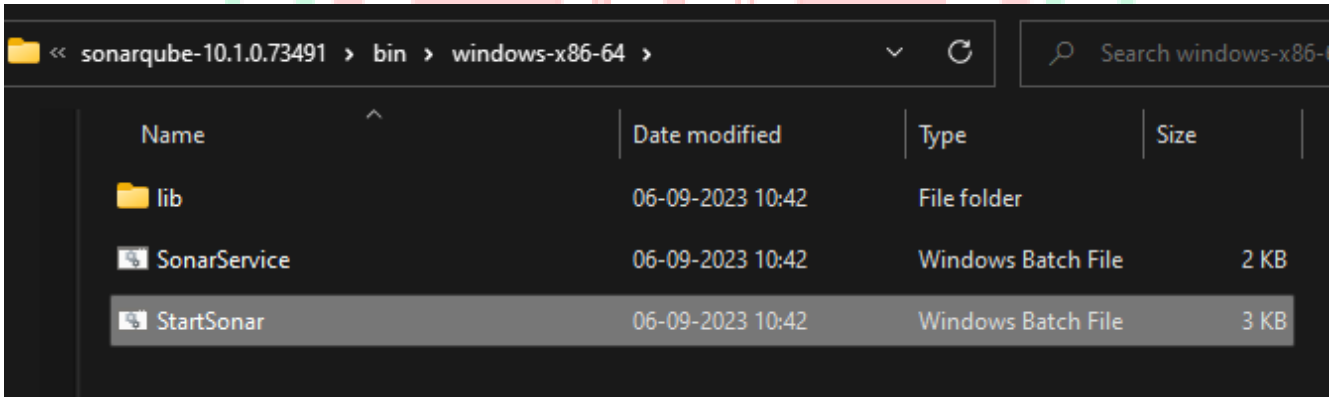
Concept	Definition
Clean Code	Code whose attributes make your software reliable, secure, and maintainable. See Clean Code for more details.
Bug	An issue that represents something wrong in the code. If this has not broken yet, it will, and will probably break at the worst possible moment. This needs to be fixed as soon as possible.
Code smell	A maintainability-related issue in the code. Leaving it as-is means that at best, developers maintaining the code will have a harder time than they should when making changes. At worst, they'll be so confused by the state of the code that they'll introduce additional errors as they make changes.
Cost	See Remediation cost.
Debt	See Technical debt.
Issue	When a piece of code does not comply with a rule, an issue is logged on the snapshot. An issue can be logged on a source file or a unit test file.
Measure	The value of a metric for a given file or project at a given time. For example, 125 lines of code on class MyClass or, the density of duplicated lines = 30.5% on project myProject, can be considered a measure.
Metric	A type of measurement. Metrics can have varying values, or measures, over time. Examples: number of lines of code, complexity, etc. A metric may be either *qualitative* (for example, the density of duplicated lines, line coverage by tests, etc.) or *quantitative* (for example, the number of lines of code, the complexity, etc.)
New code definition	A changeset or period that you're keeping a close watch on for the introduction of new problems in the code. Ideally, this is since the previous_version, but if you don't use a Maven-like versioning scheme, you may need to set a time period such as *21 days since a specific analysis* or use a reference branch. See Defining new code for more details.
Quality profile	A set of rules. Each snapshot is based on a single quality profile. See also Quality profiles.

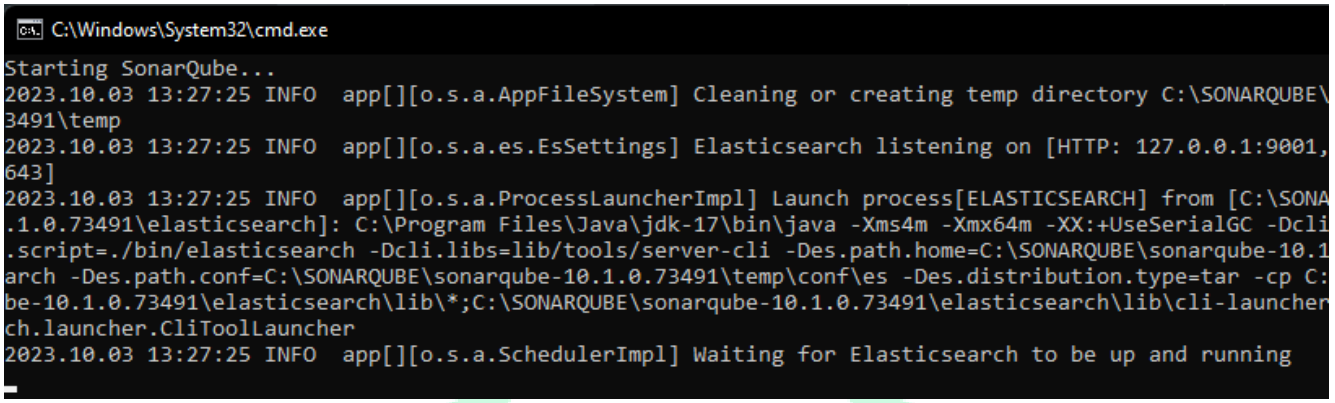
Rule	A coding standard or practice that should be followed. Not complying with coding rules can lead to issues and hotspots. Adherence to rules can be used to measure the quality of code files or unit tests.
Remediation cost	The estimated time required to fix vulnerability and reliability Issues.
Snapshot	A set of measures and issues on a given project at a given time. A snapshot is generated for each analysis.
Security hotspot	Security-sensitive pieces of code that need to be manually reviewed. Upon review, you'll either find that there is no threat or that there is vulnerable code that needs to be fixed.
Technical debt	The estimated time required to fix all maintainability issues and code smells.
Vulnerability	A security-related issue that represents a backdoor for attackers. See also Security-related rules.

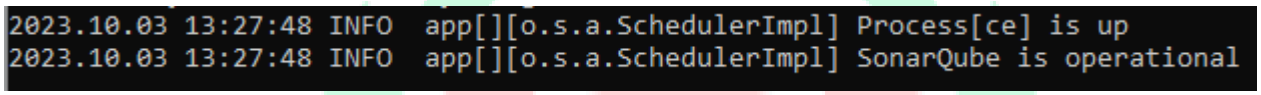
7. Laboratory Exercise

Step 1: StartSonar server from SonarQube folder

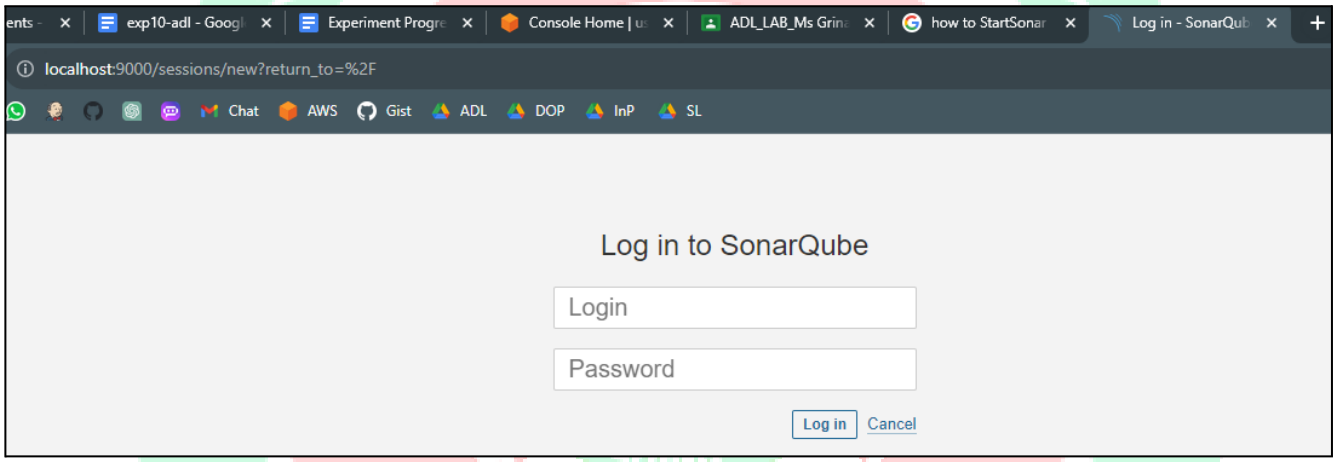




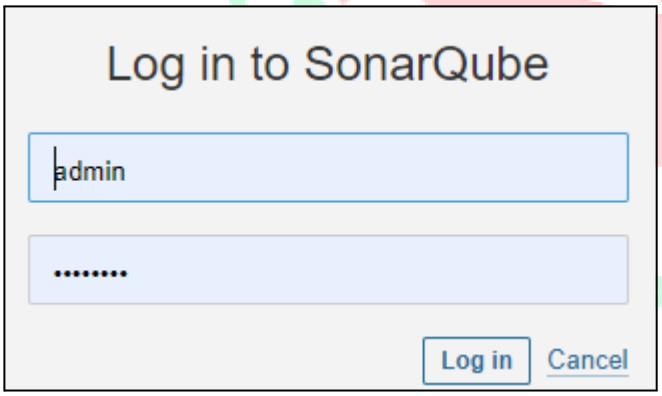




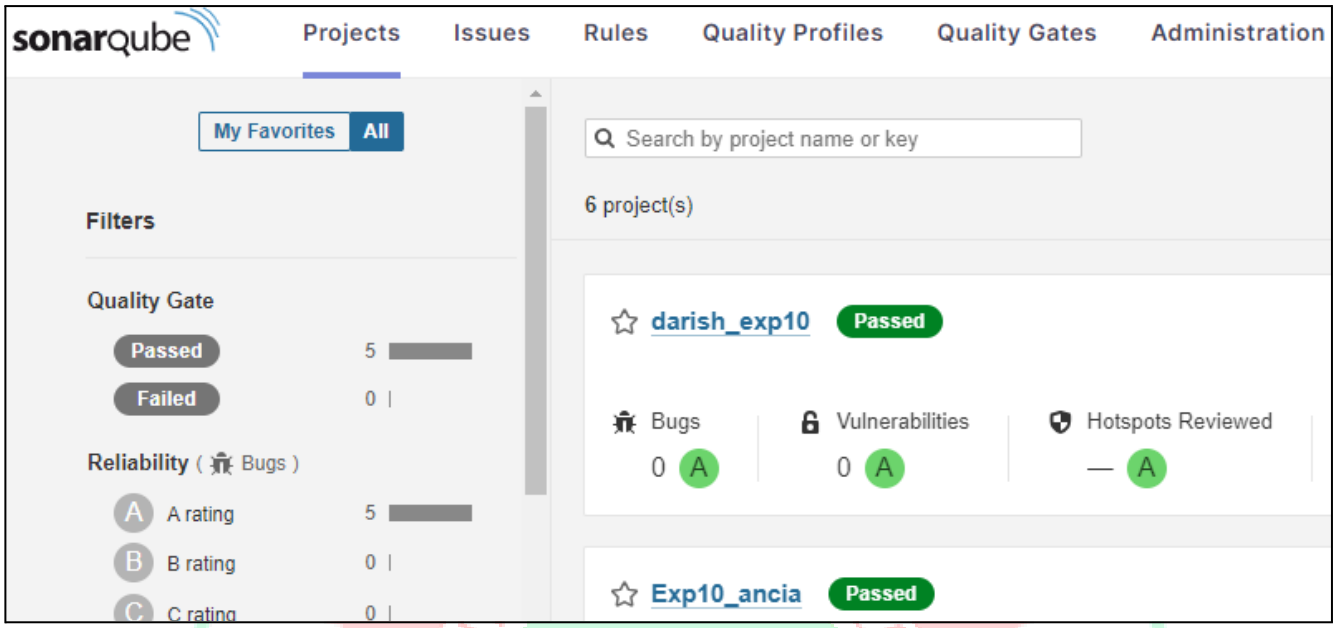
Step 2: : on browser chk https://localhost:9000



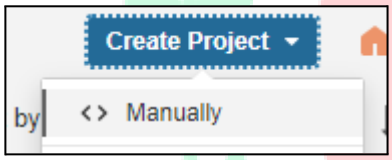
Step 3: By default Username and Password is admin. You can change the password here.

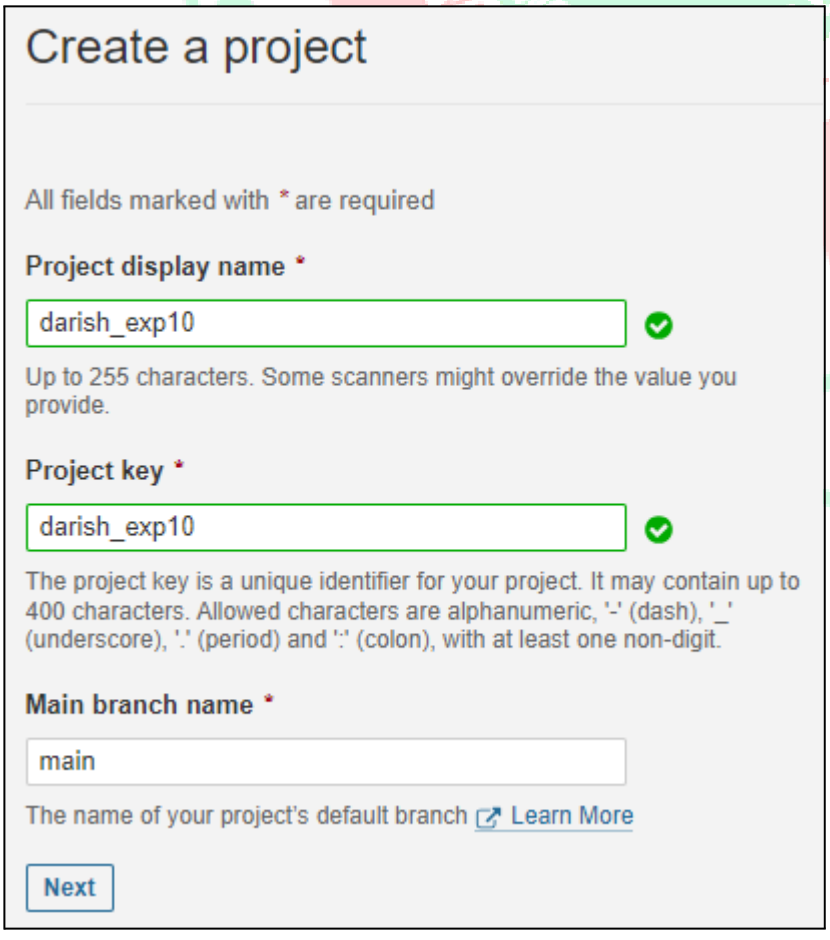


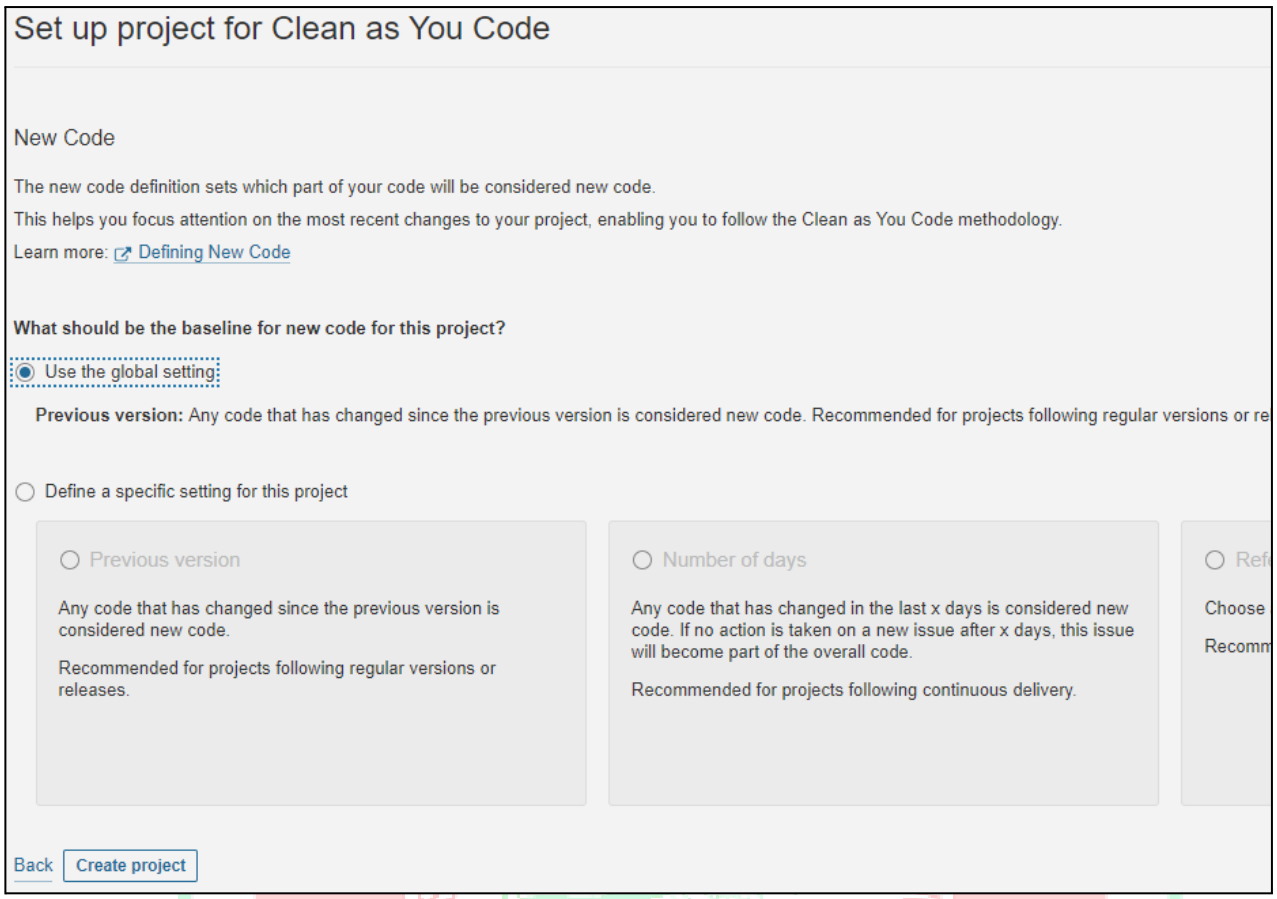
Step 4: Login with new password.



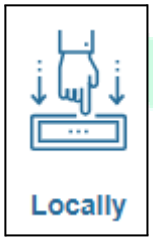
Step 5: Click on Manually. Create a Project with name

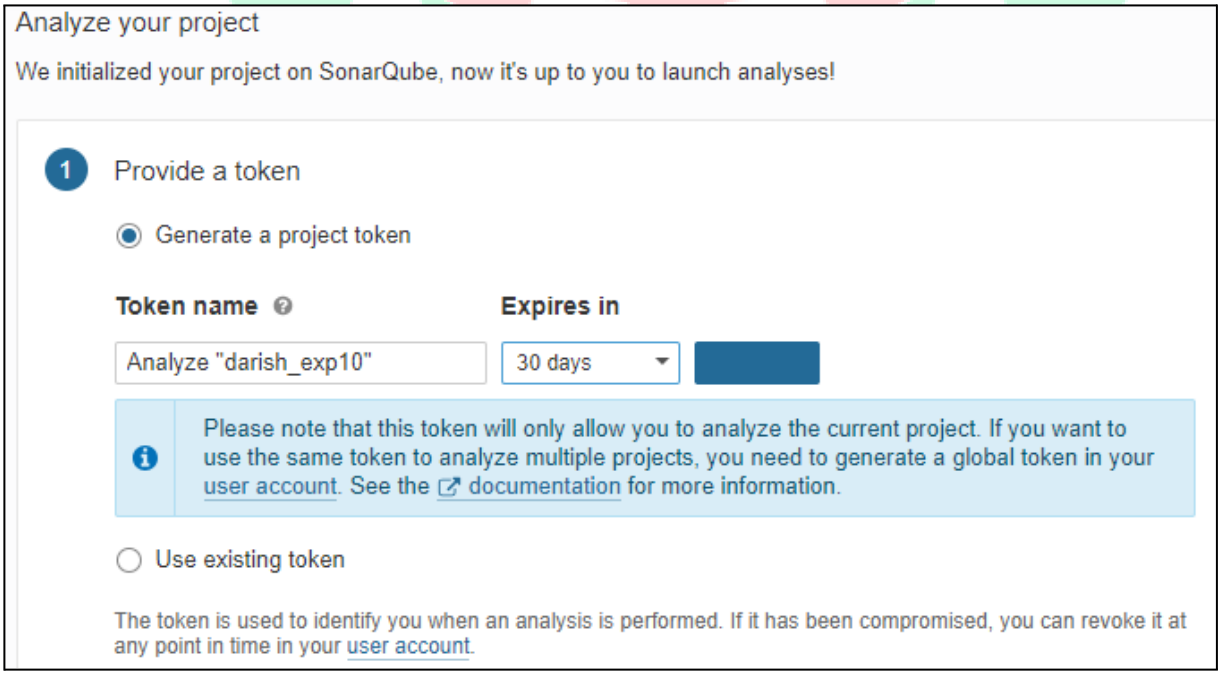




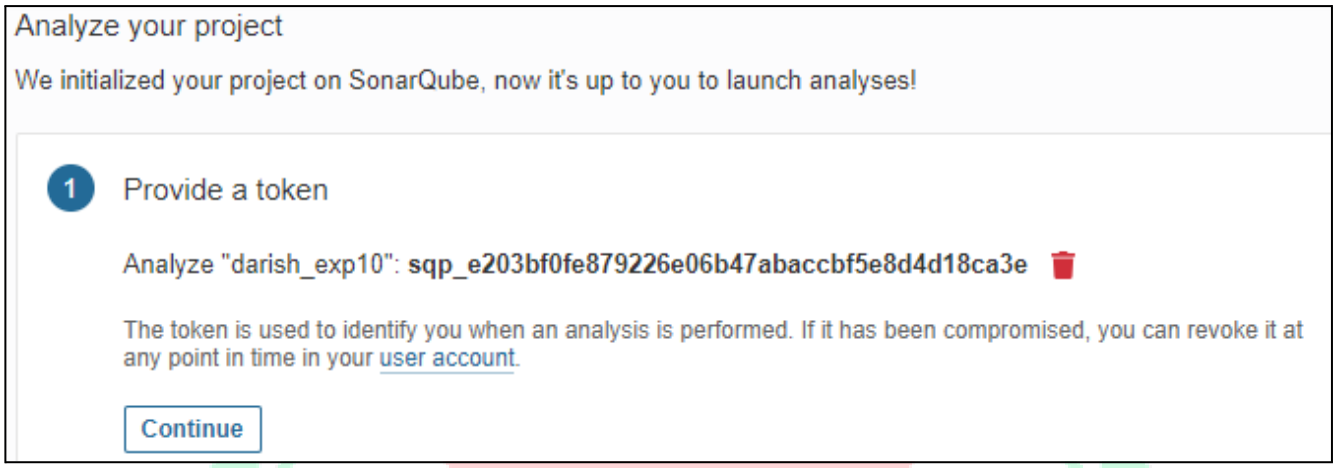


Step 6: Click on Locally and given token name

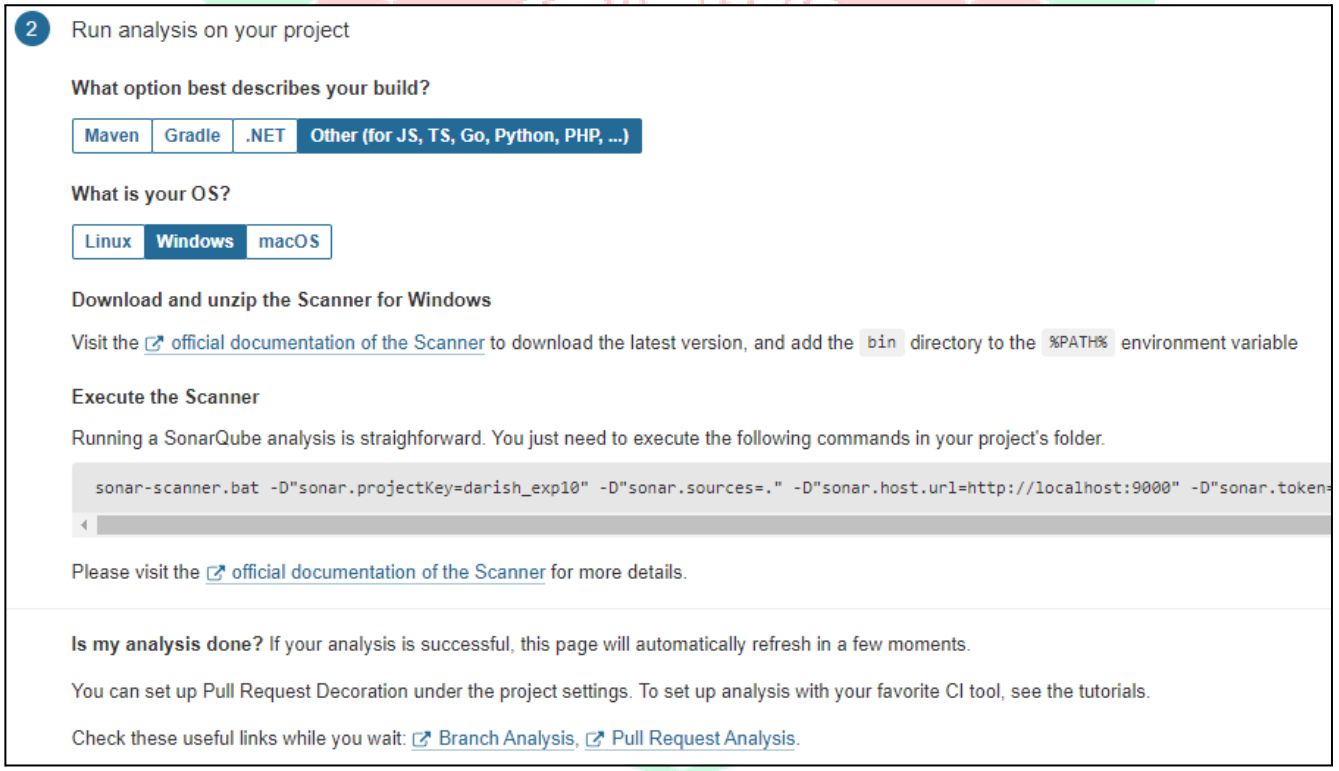




Step 7: Give token name and continue

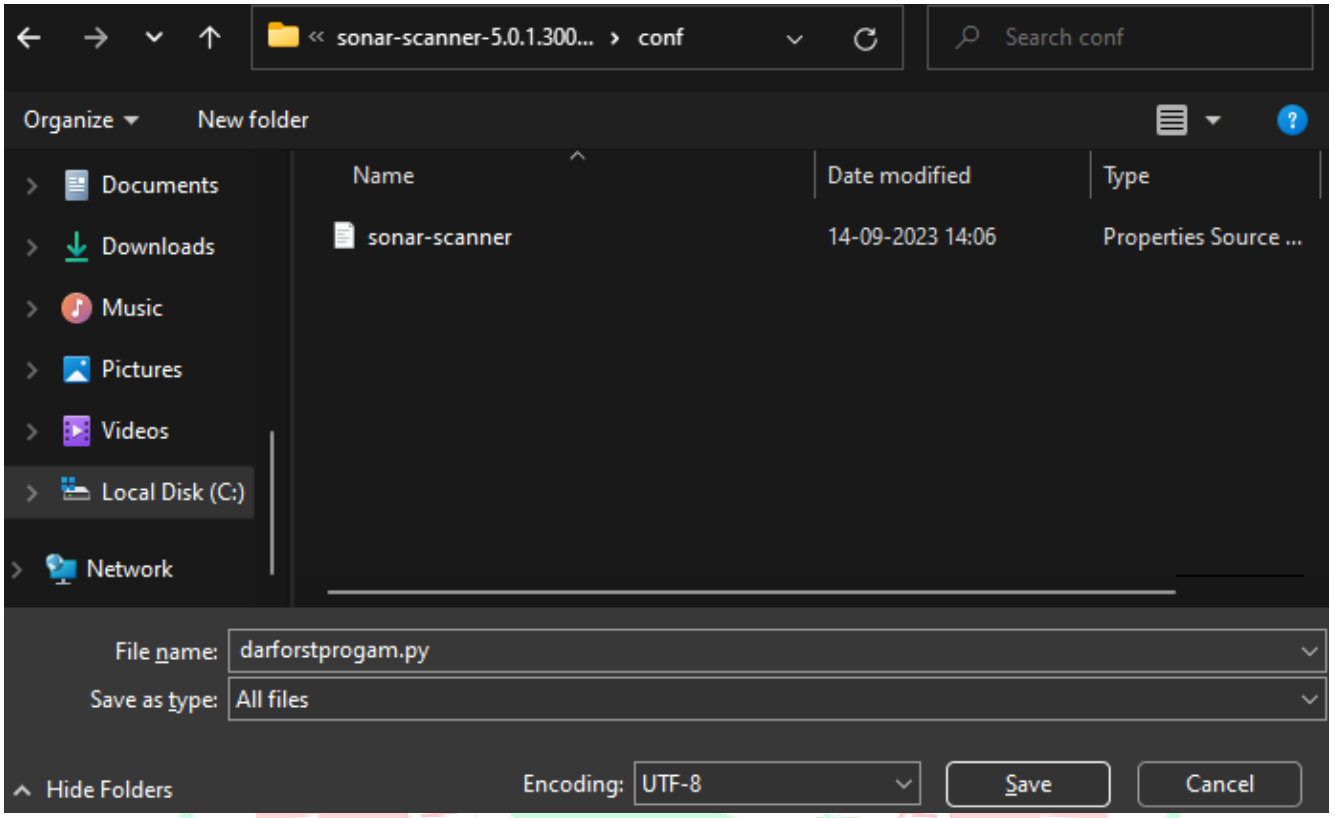


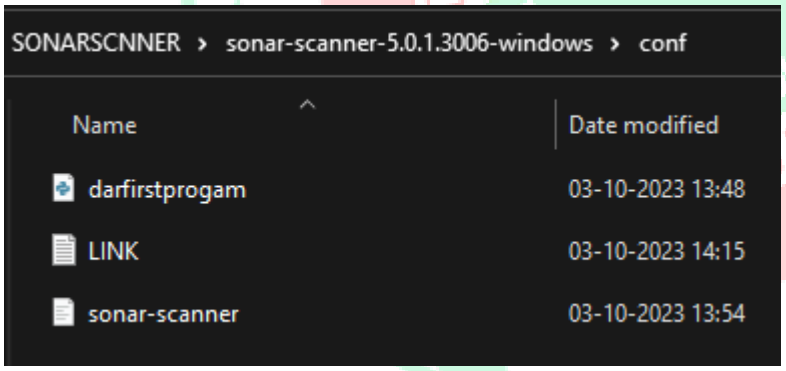
Step 8: Select what type of project you want to test



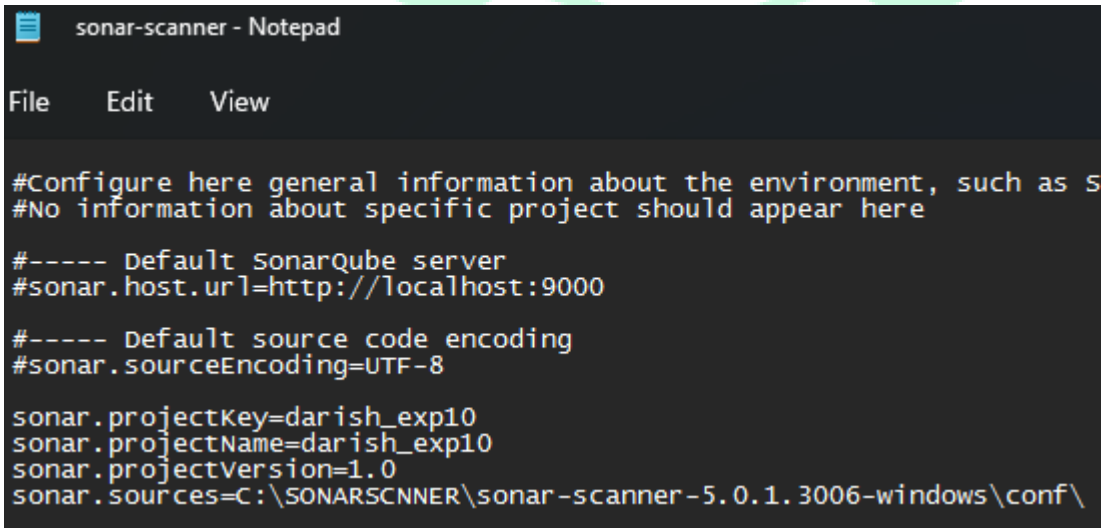
Step 9: Click on Other and select OS as Windows

Step 10: Create a folder to keep python scripts





Step 11: Start configuration of the scanner.



Add following lines to C:/SonarScanner/conf/Sonar-Scanner.properties

```
#No information about specific project should appear here
#----- Default SonarQube server
#sonar.host.url=http://localhost:9000
#----- Default source code encoding
#sonar.sourceEncoding=UTF-8
sonar.projectKey=darish_exp10
sonar.projectName=darish_exp10
sonar.projectVersion=1.0
sonar.sources=C:\SONARSCNNER\sonar-scanner-5.0.1.3006-windows\conf\
```

Step 12: Open new command prompt to Run and scan python files

C:/SFITJobs/PY Scripts >sonar-scanner.bat -D"sonar.projectKey=darish_exp10" -D"sonar.sources=."

-D"sonar.host.url=http://localhost:9000" -D" sonar.token=

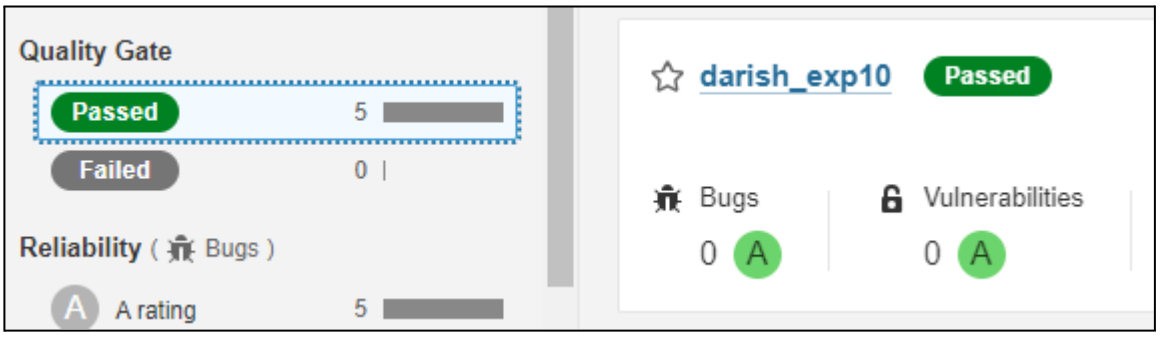
sqp_e203bf0fe879226e06b47abaccbf5e8d4d18ca3e"

```
C:\SONARSCNNER\sonar-scanner-5.0.1.3006-windows\conf>sonar-scanner.bat -D"sonar.projectKey=darish_exp
s=." -D"sonar.host.url=http://localhost:9000" -D"sonar.token=sqp_e203bf0fe879226e06b47abaccbf5e8d4d1
INFO: Scanner configuration file: C:\SONARSCNNER\sonar-scanner-5.0.1.3006-windows\bin\..\conf\sonar-
INFO: Project root configuration file: NONE
INFO: SonarScanner 5.0.1.3006
INFO: Java 17.0.7 Eclipse Adoptium (64-bit)
INFO: Windows 11 10.0 amd64
INFO: User cache: C:\Users\Student\.sonar\cache
INFO: Analyzing on SonarQube server 10.1.0.73491
INFO: Default locale: "en_IN", source code encoding: "windows-1252" (analysis is platform dependent)
INFO: Load global settings
INFO: Load global settings (done) | time=55ms
```

```
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=AYr0pfgGRGEHtkuKE
INFO: Analysis total time: 5.209 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 6.388s
INFO: Final Memory: 21M/80M
INFO: ------------------------------------------------------------------------

C:\SONARSCNNER\sonar-scanner-5.0.1.3006-windows\conf>
```

Step 13: SonarQube command will give output on dashboard. Command Prompt is showing Execution Success and dashboard will give Quality Gate Status : Passed



8. Post-Experiments Exercise

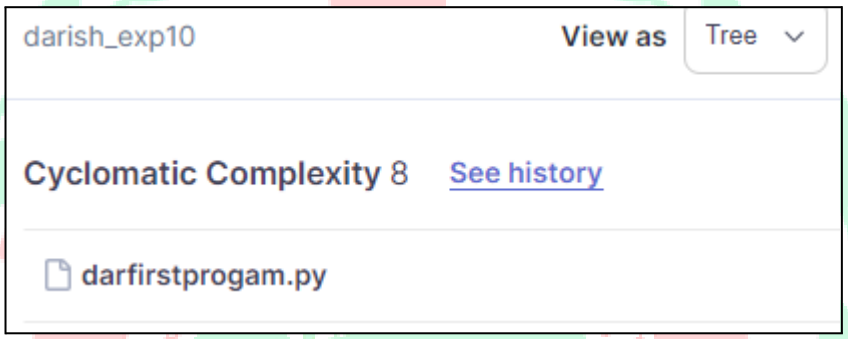
A. Extended Theory:(write in hand)

- what is Code smell?
- list characteristics of good quality code.

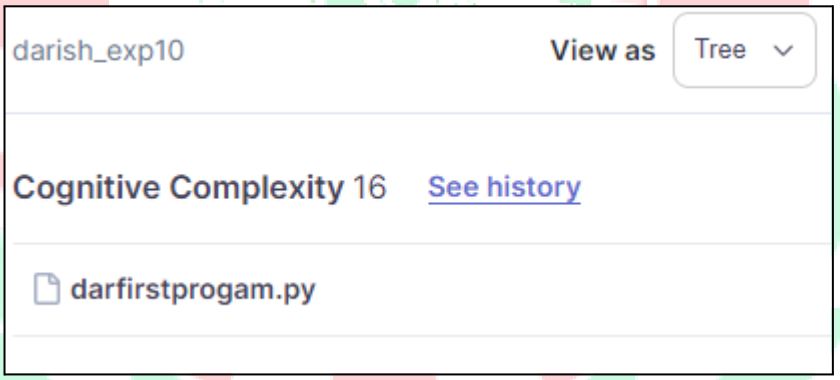
B. Questions:(attach SS)

- complexity measure of the code

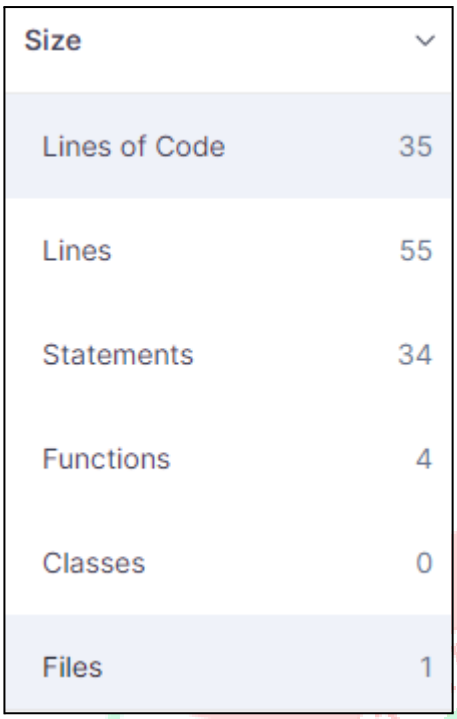
Cyclomatic complexity

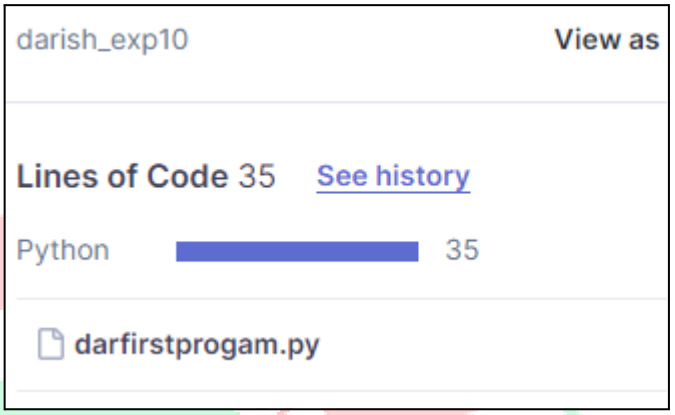


Cognitive complexity

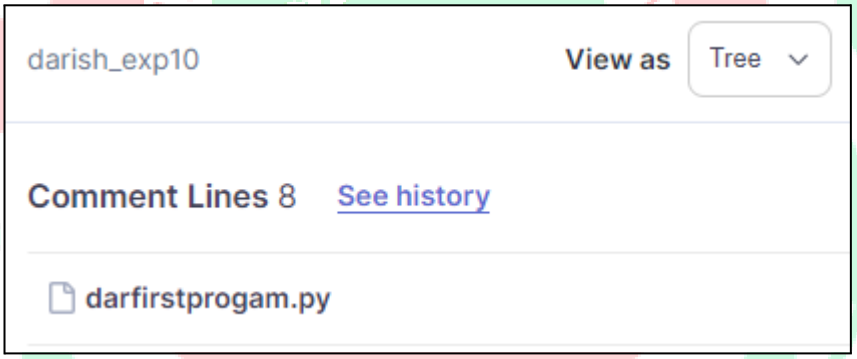


- size of the code





- no of comments



C. Conclusion:(write in hand)

1. Write what was performed in the experiment
2. Mention few applications of what was studied.
3. Write the significance of the studied topic

D. References:

A. https://medium.com/swlh/sonarqube-part-2-features-of-sonarqube-installation-and-some-practice-on-sonarqube-d523ae9a998a

B. https://docs.sonarqube.org/

C.https://www.codeusingjava.com/interview/sonar
