St. Francis Institute of Technology, Mumbai-400 103
Department of Information Technology

A.Y. 2024-2025
Class: TE-ITA/B, Semester: V

Subject: **Advanced DevOps Lab**

Experiment – 8: To test python files with SonarQube and observe 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
   - Understand static analysis of the code.
3. **Outcomes:** After study of this experiment, the student will be able to
   - Generate static analysis of the code with code smells, bugs, security vulnerabilities.
4. **Prerequisite:** Fundamentals of Software Testing
5. **Requirements:** PC and Internet
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.

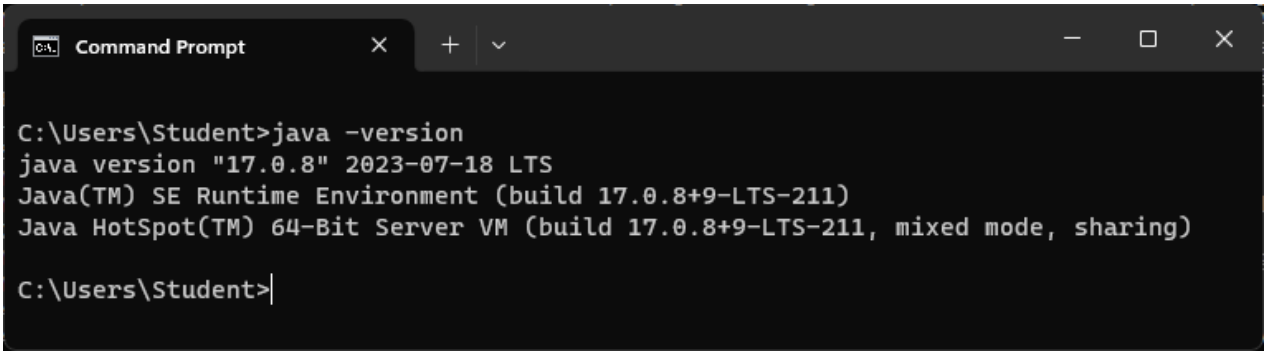
Digs into issues — If it shows that there is a problem SonarQube allows you to go and directly check it out from the summary report or from one code file to another. In the SonarQube summary dashboard, you can see furthermore details of the errors bu just clicking on the error.

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.

7. Laboratory Exercise

Installation Steps

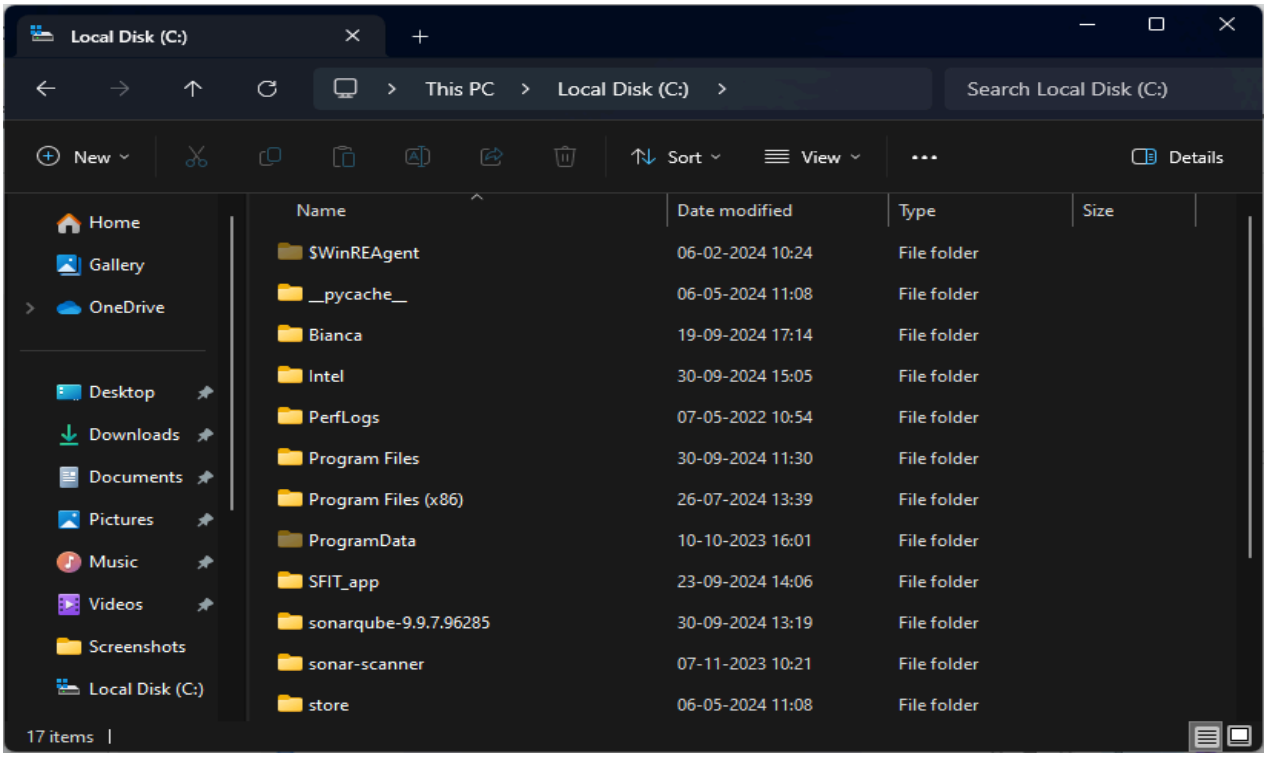
Step A: Install Java 1.11.0.11 or upgrade Java to min. jdk1.11

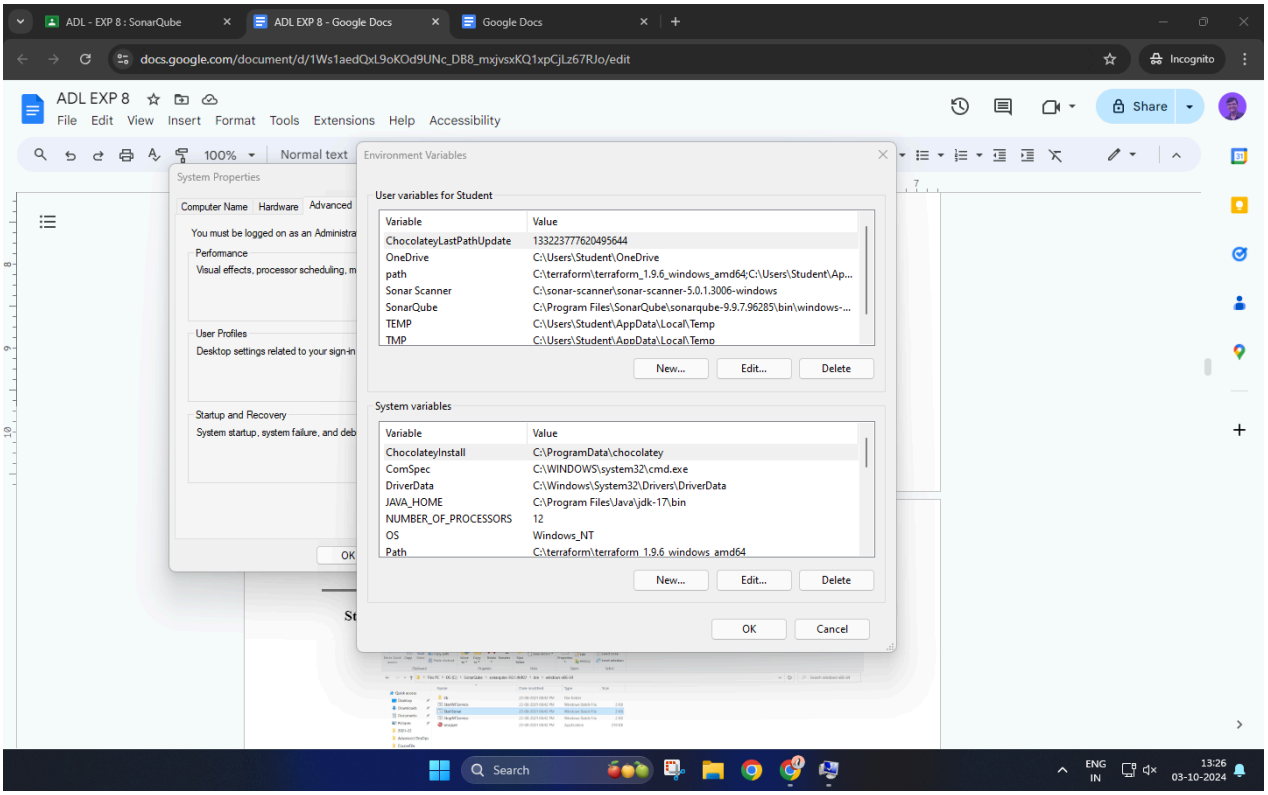


Step B: Download SonarQube from https://www.sonarqube.org/downloads/

Step C : Download SonarScanner from https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/

Step D: Extract Zip files in C:/Program Files/SonarQube folder and C:/Program Files /SonarScanner folder

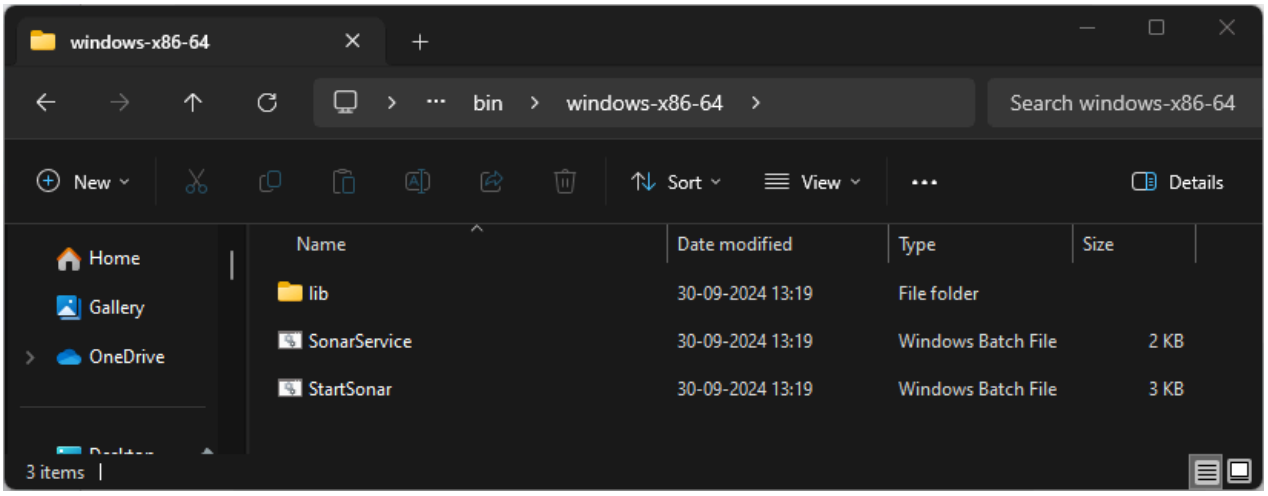


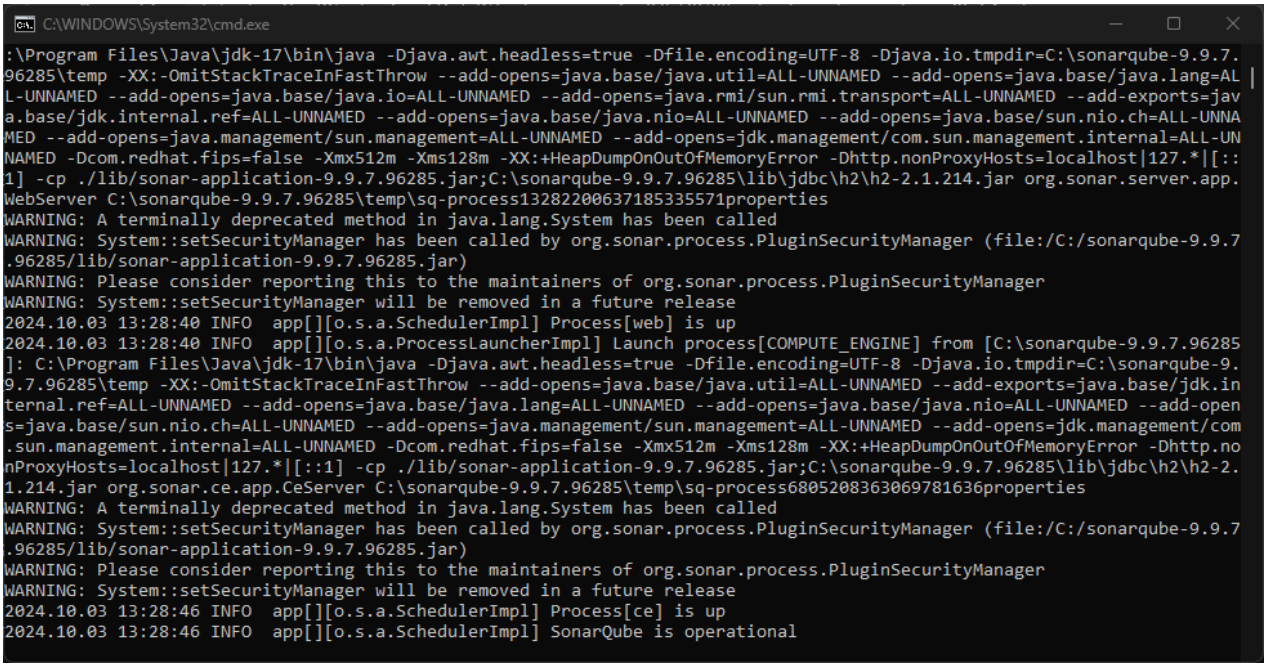


We installed and configured SonarQube and SonarScanner.

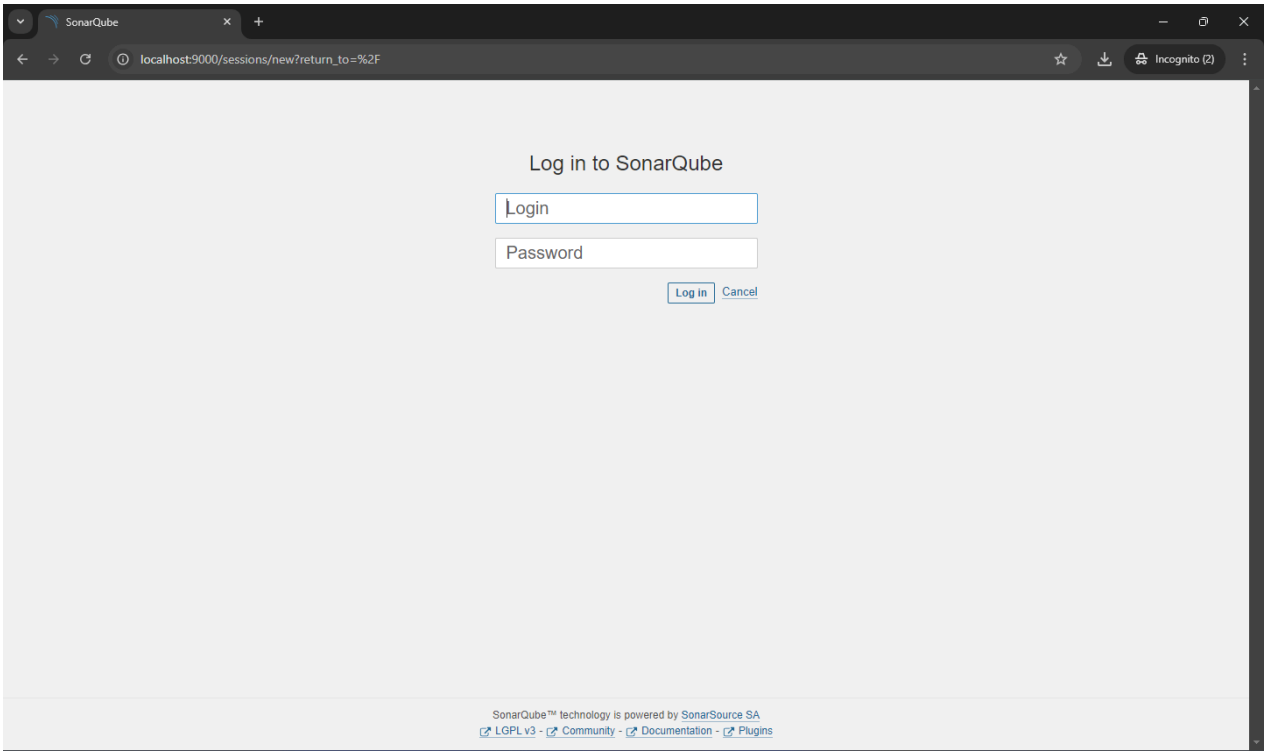
SonarQube Setup

Step 1: StartSonar server from SonarQube folder

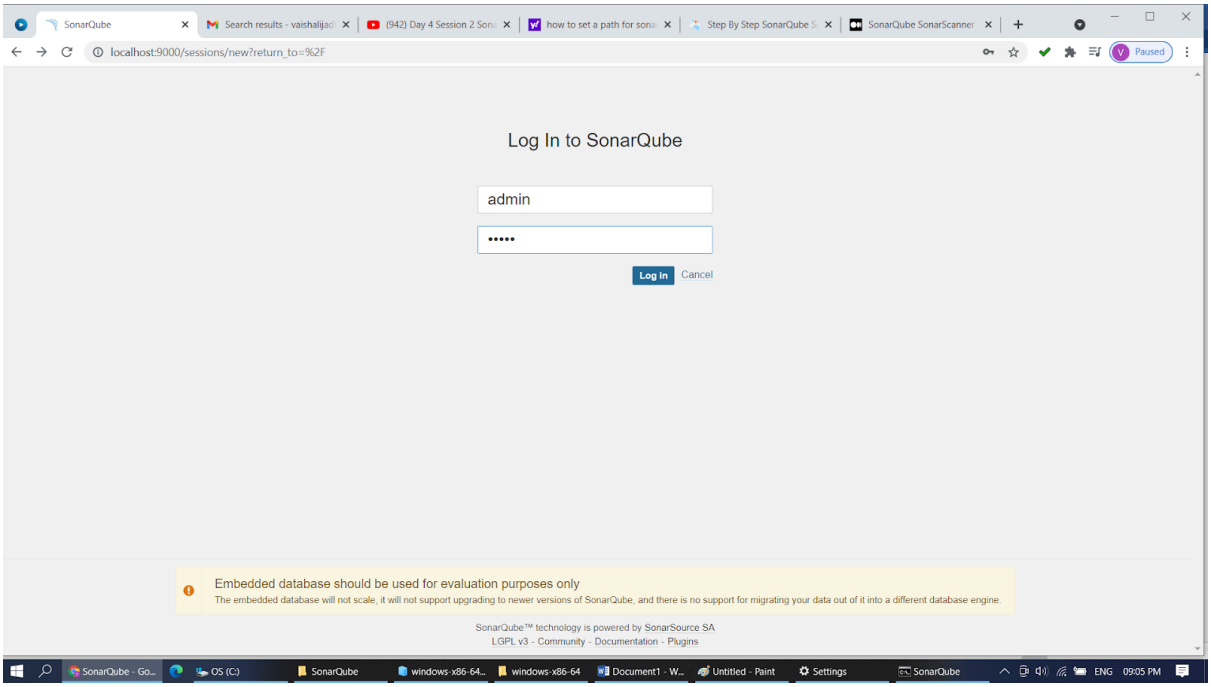




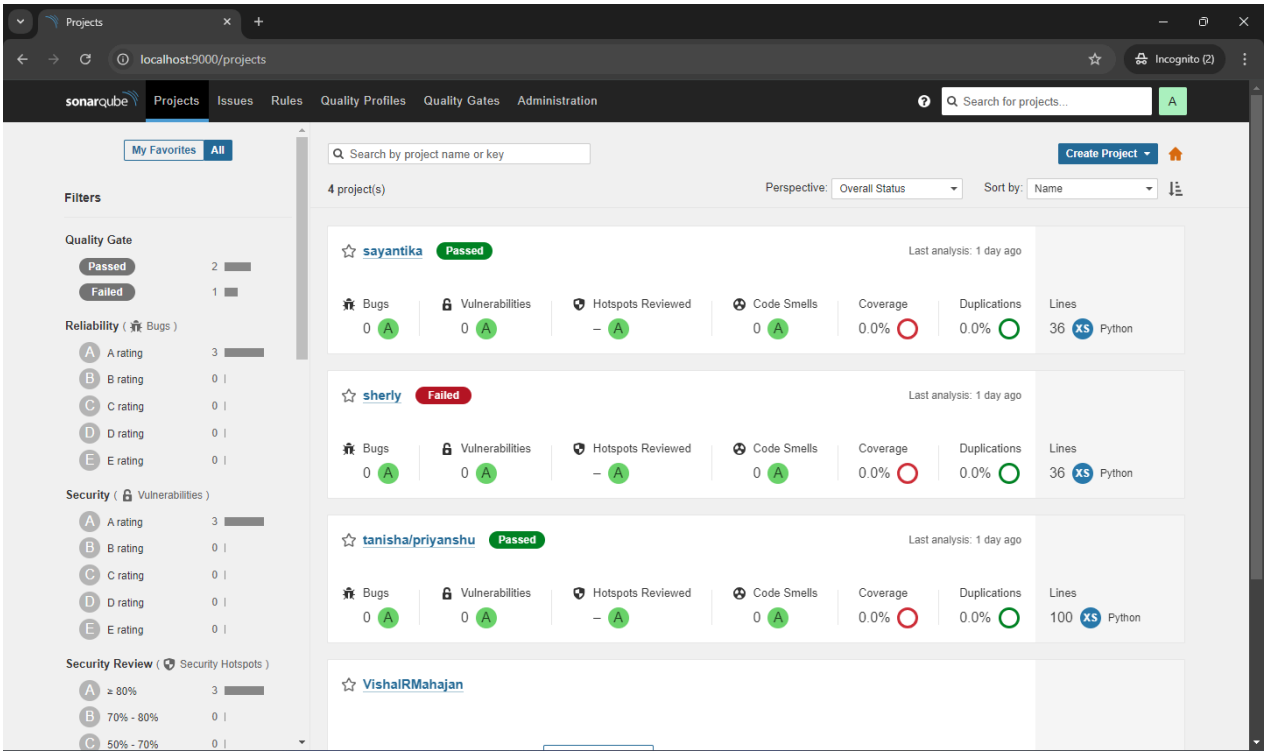
Step 2: On browser Open localhost:9000 or 127.0.0.1:9000



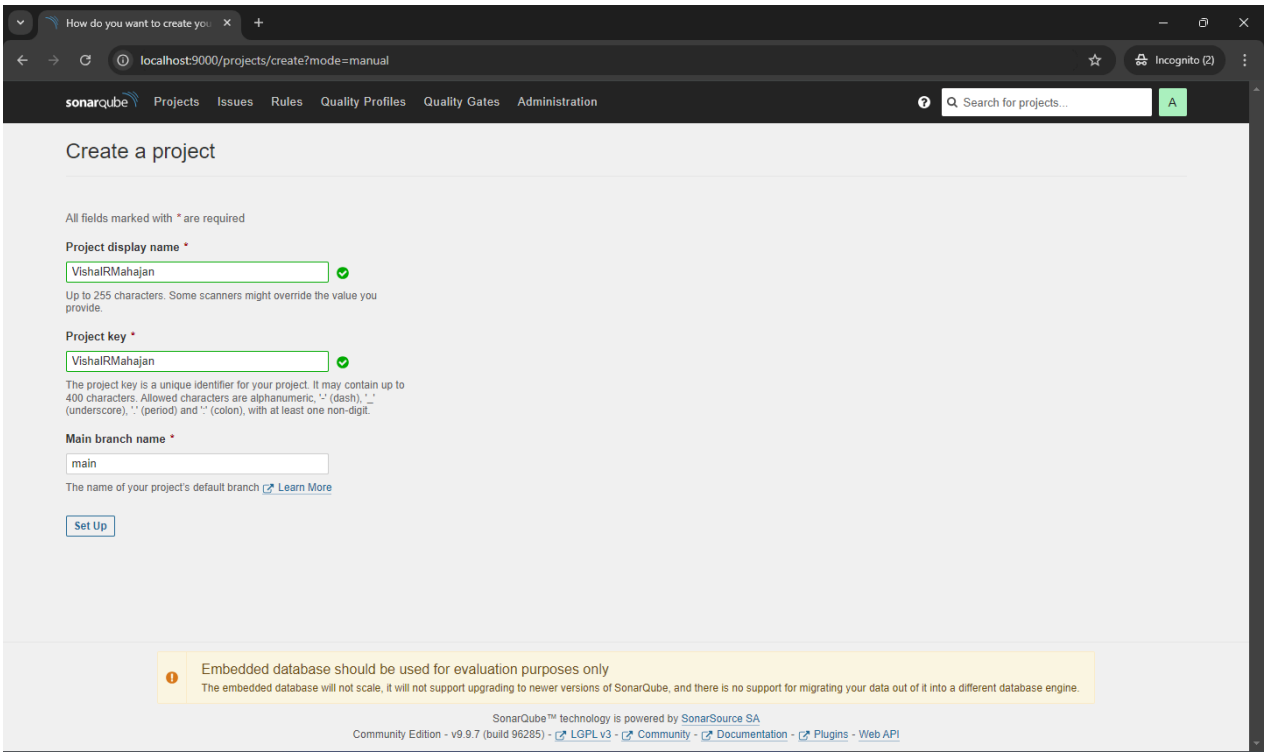
Step 3: By default Username and Password is **admin**. You can change the password here.



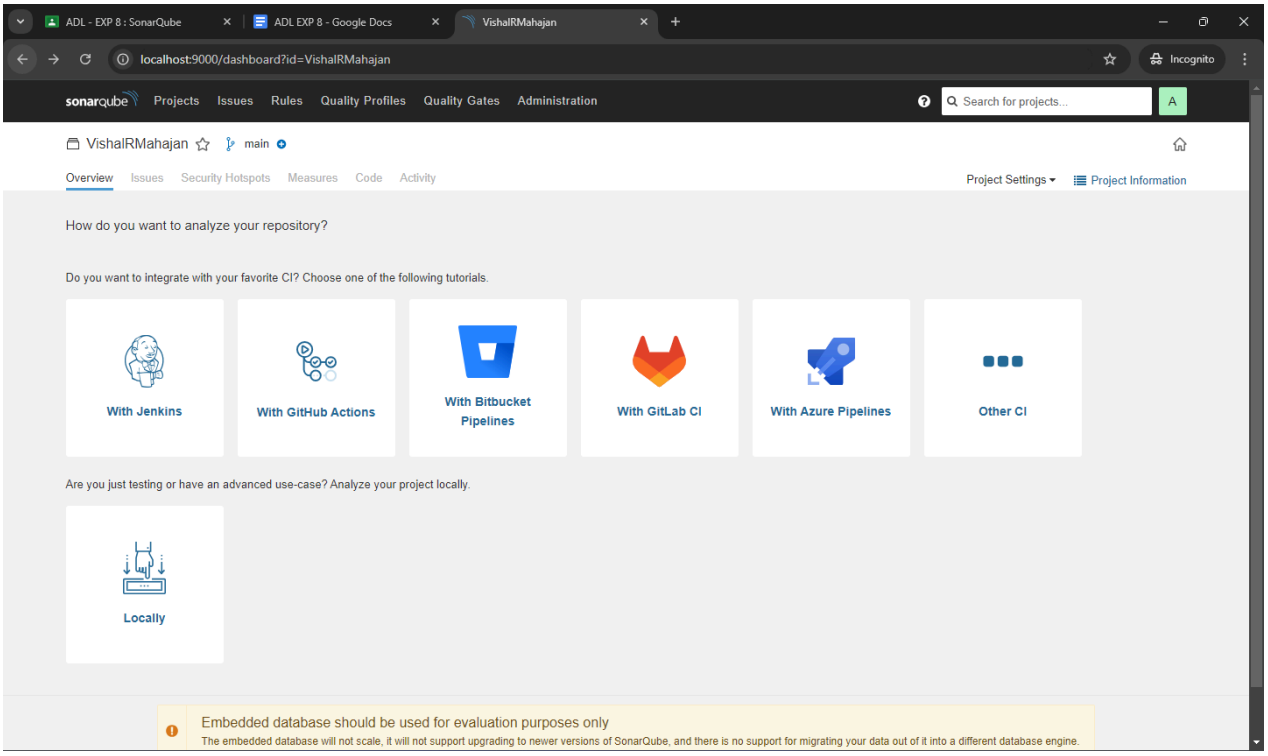
Step 4: Login with new password.



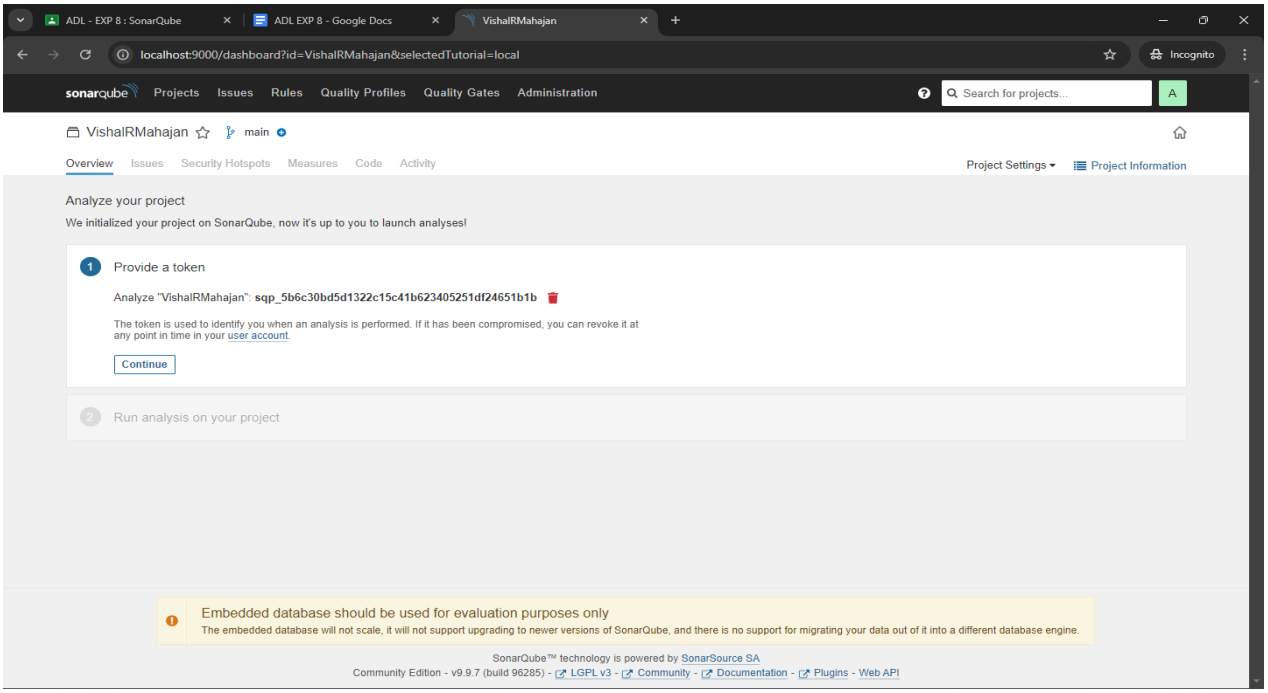
Step 5: Click on Manually. Create a Project named VishalRMahajan



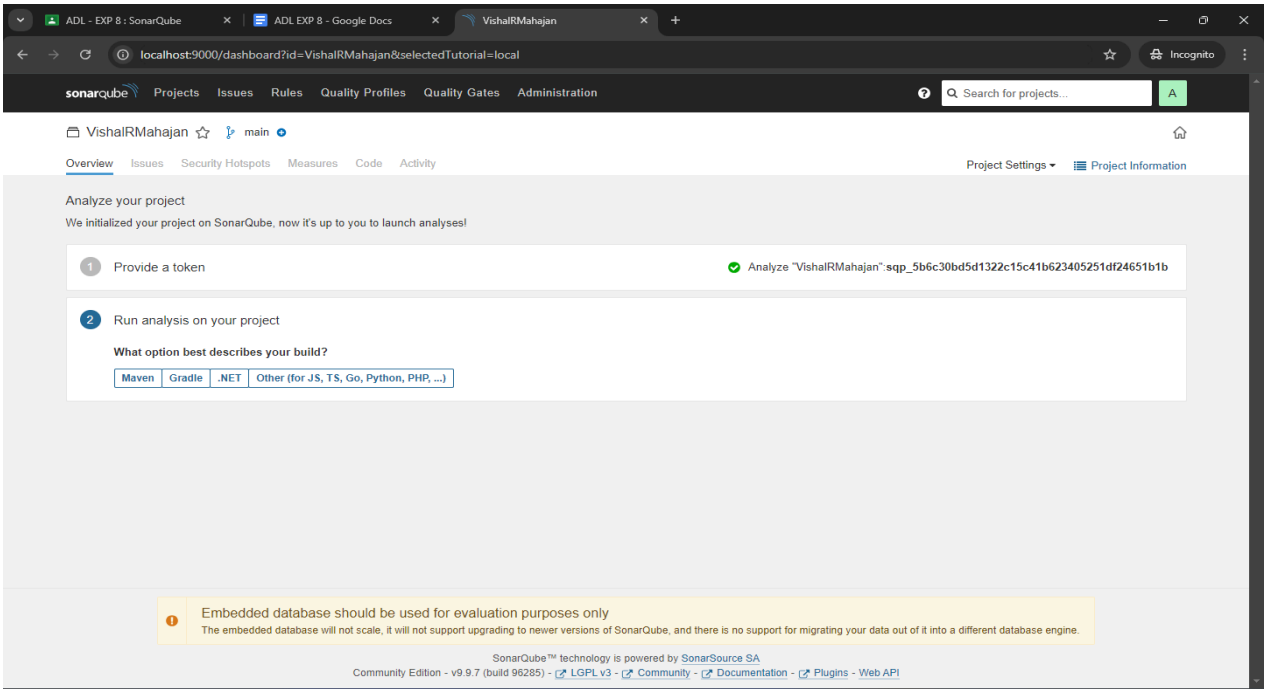
Step 6: Click on Locally and given token name



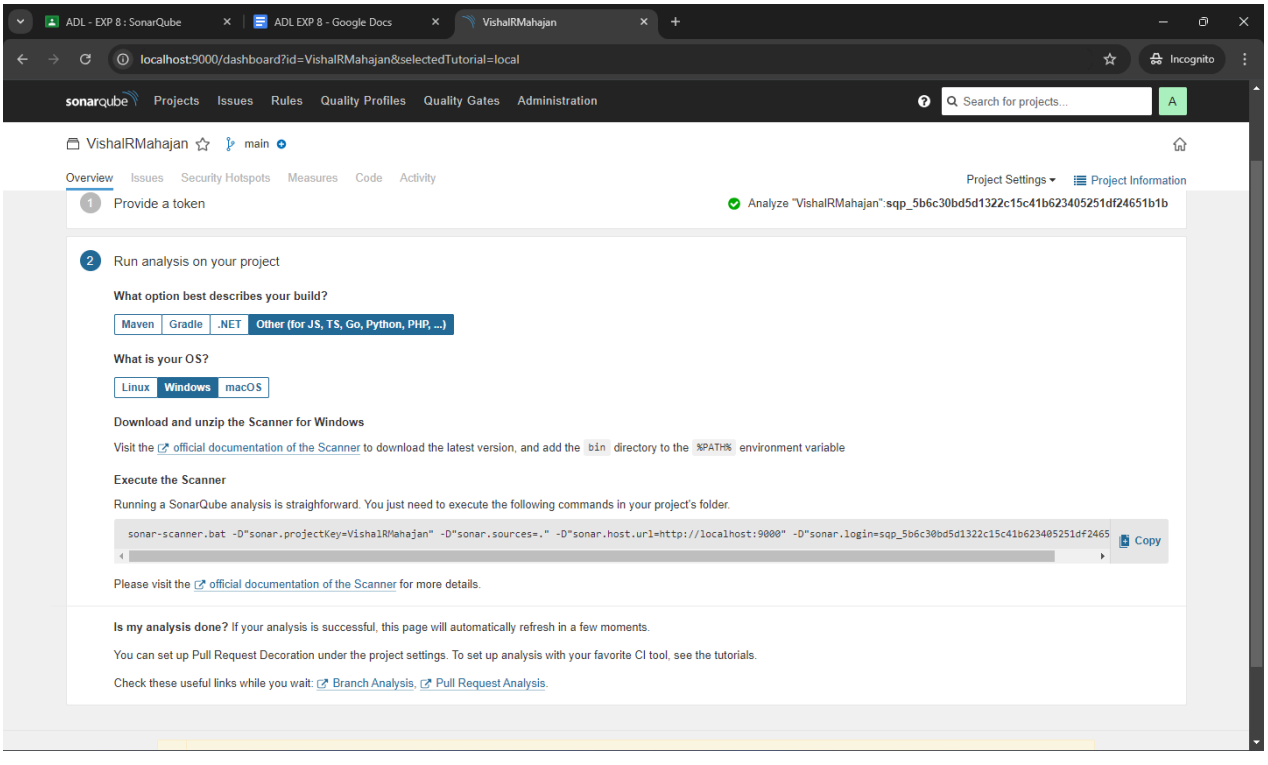
Step 7: Give token name VishalRMahajan and Click on Continue



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



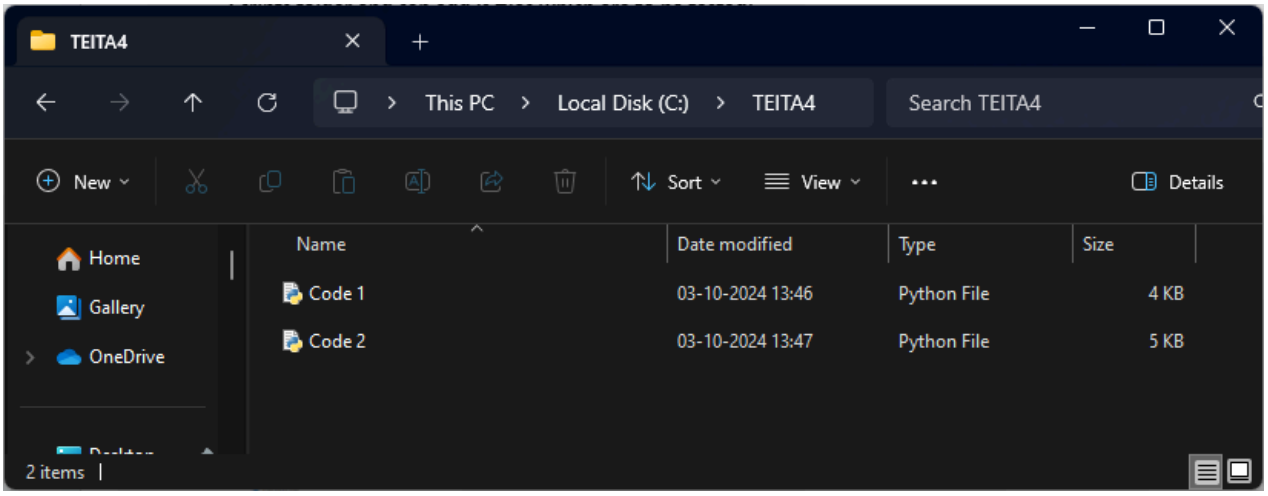
Step 9: Click on "Other" and choose "OS" as Windows. Keep this window open, as we will need it for the client setup.



Step 10: Copy the command and save it somewhere, as we will need it later.

sonar-scanner.bat -D"sonar.projectKey=VishalRMahajan" -D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -D"sonar.login=sqp_5b6c30bd5d1322c15c41b623405251df24651b1b"

Step 11: Create a folder in C:/TEITA4 . Add python programs in to it (you can create Java Scripts folder and can add js files which are to be tested)



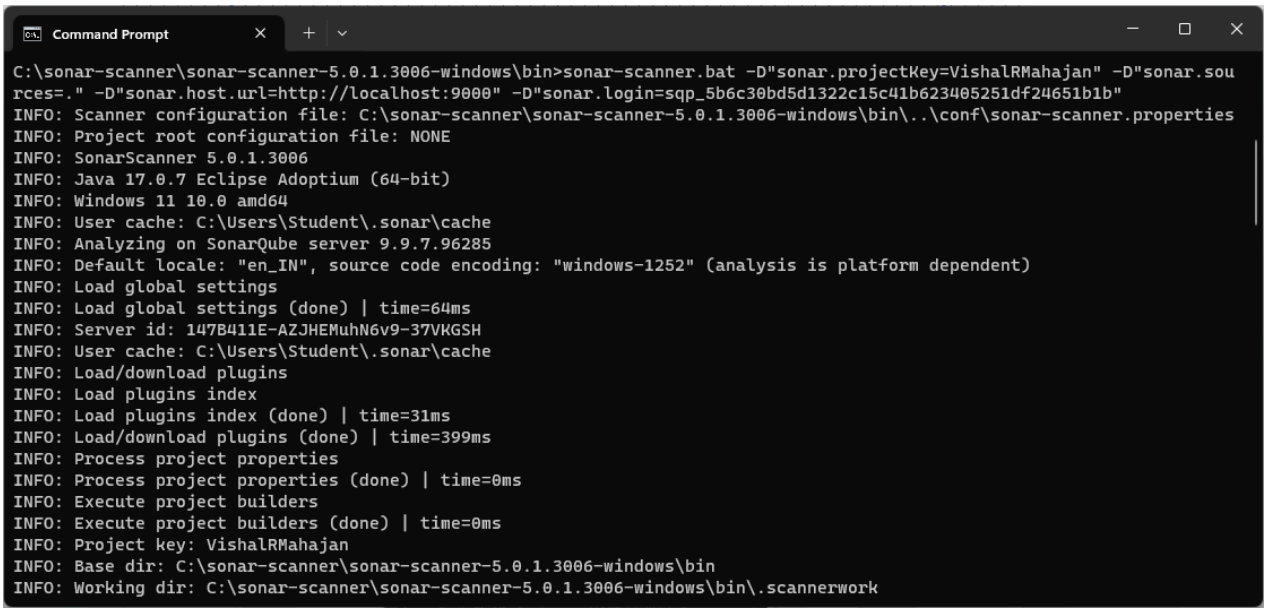
Sonar Scanner (Client) Setup

Step 12 : Open the config folder in the Sonar Scanner directory. Inside, you'll find the sonar-scanner.properties file. Open this file using Notepad or VS Code and make the following edits:

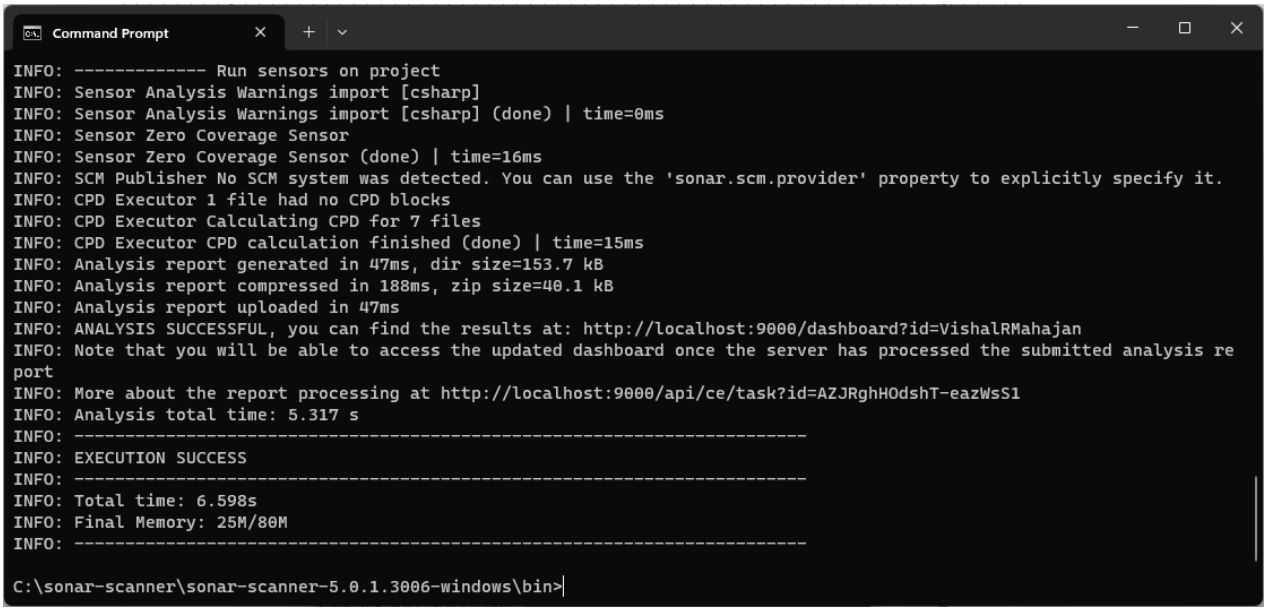
```
#Configure here general information about the environment, such as SonarQube server connection details for example

#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
```

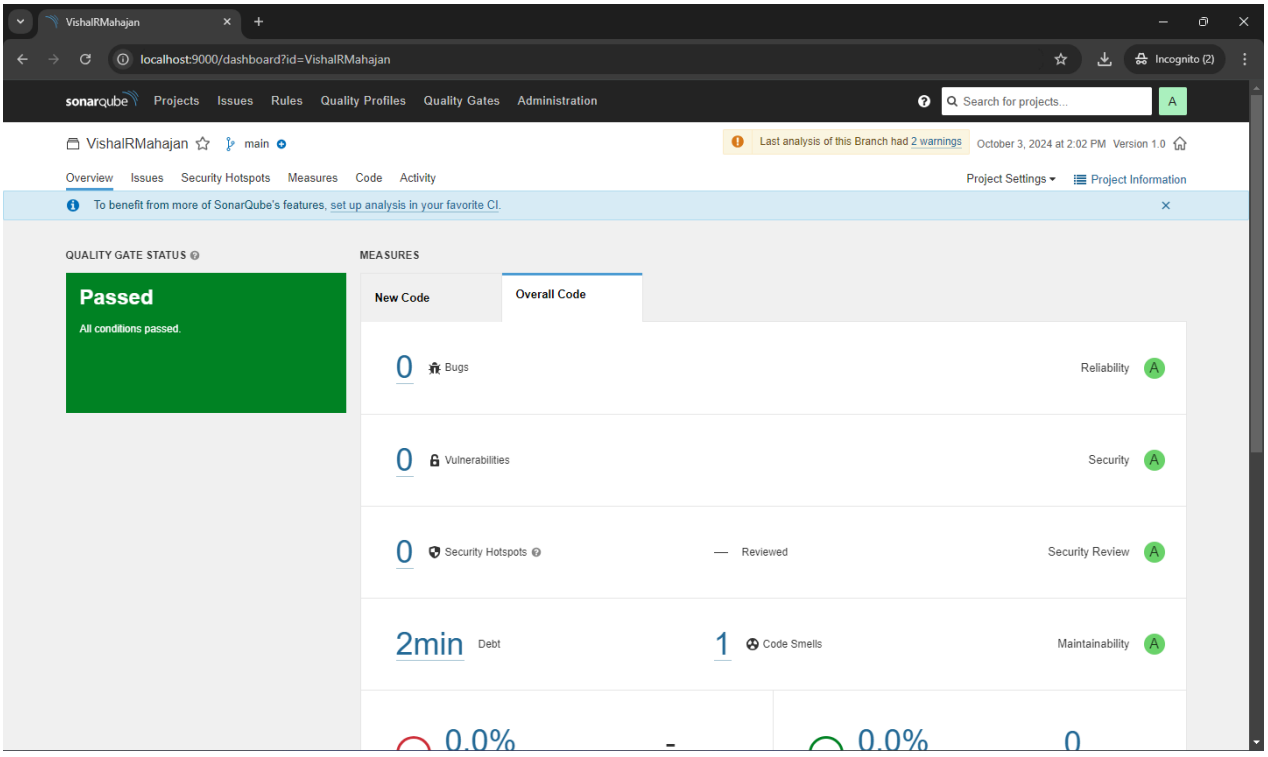
```
sonar.projectKey=VishalRMahajan

sonar.projectName=VishalRMahajan

sonar.projectVersion=1.0

sonar.sources=C:\sonar-scanner\sonar-scanner-5.0.1.3006-windows\bin
```

Step 13 : Save the changes to the file. Next, move the folder you created in Step 11 to the bin folder of the Sonar Scanner. Open a new Command Prompt and run the command copied from (Step 10) the dashboard. This will generate the report.

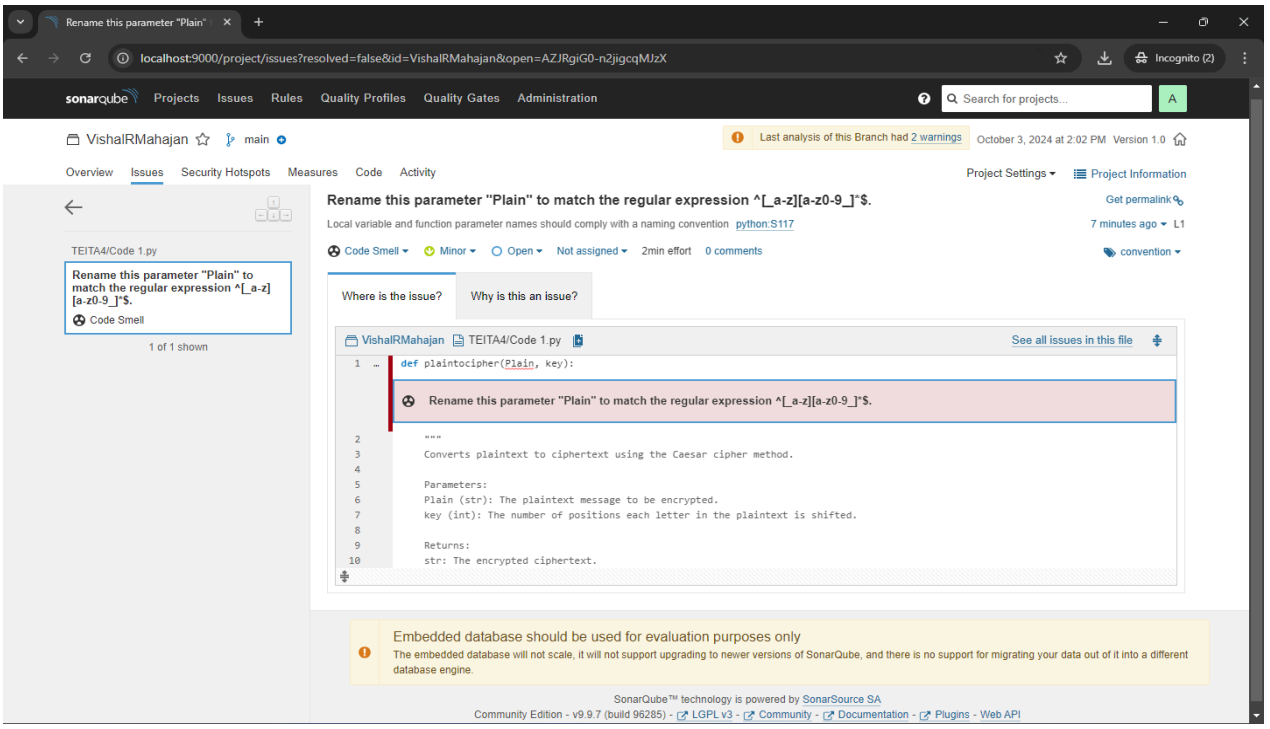


If there are no errors in the Sonar Scanner and SonarQube configuration, the copied command will produce output on the dashboard. The Command Prompt will display "*Execution Success*," and the dashboard will show "*Quality Gate Status: Passed*."

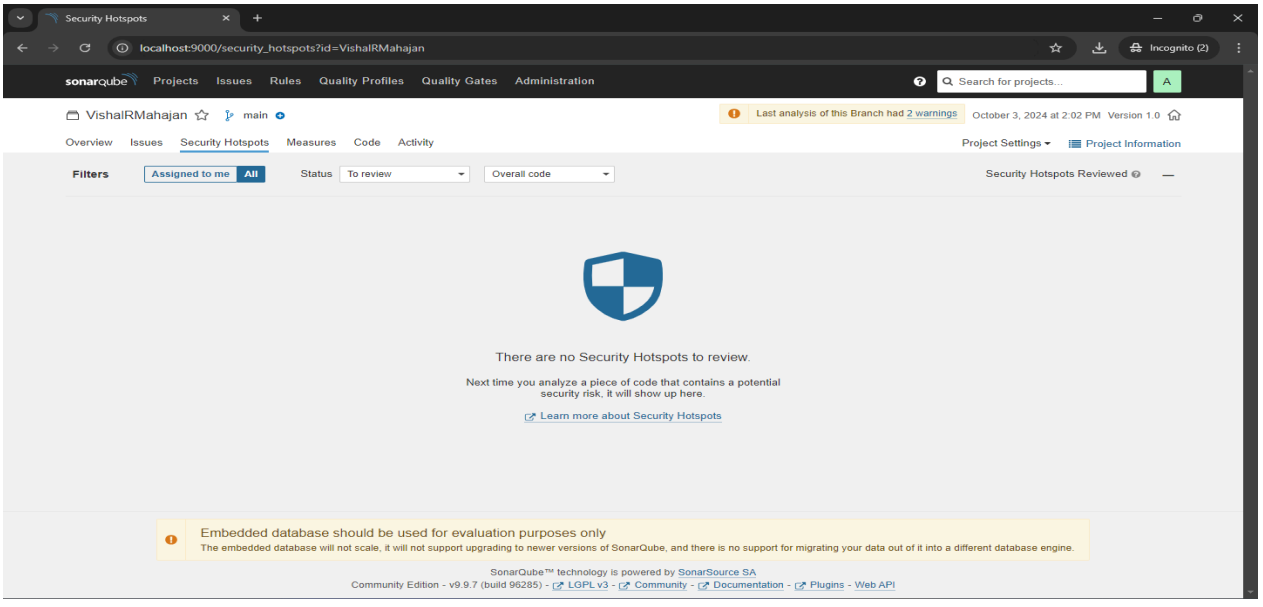




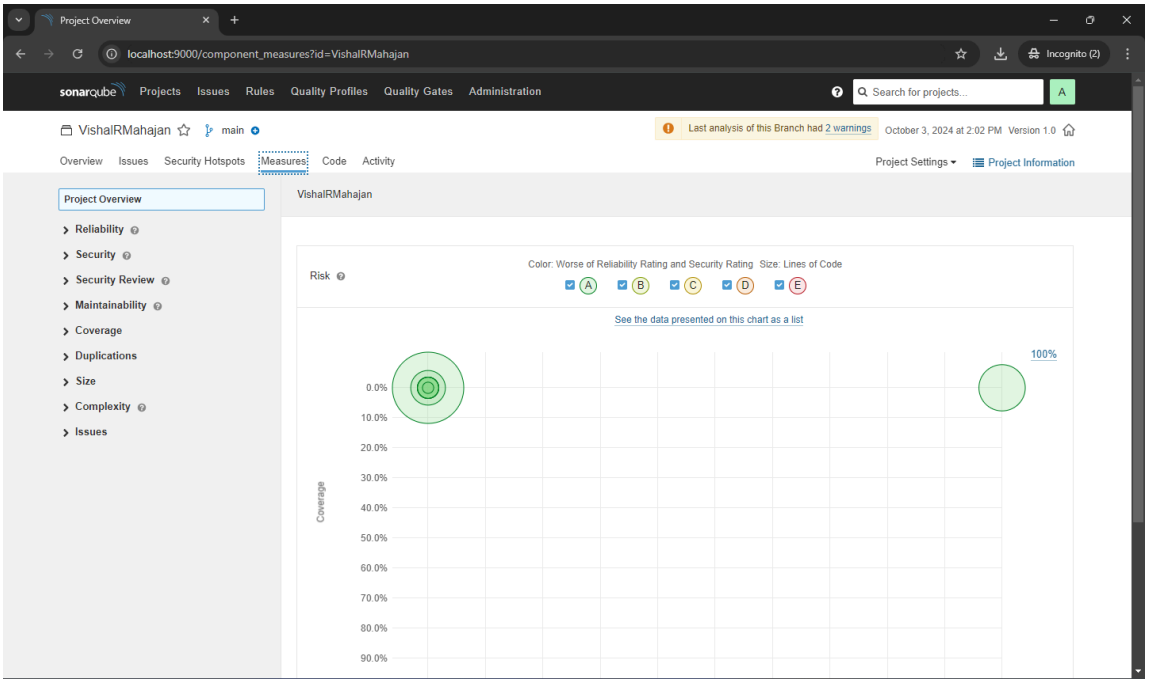
ISSUES: We have one issue: "*Rename this parameter 'Plain' to match the regular expression ^[a-z][a-z0-9]*$.*". This means that the parameter name "Plain" does not conform to the specified naming convention defined by the regular expression. According to this rule, parameter names should start with a lowercase letter or an underscore and can only contain lowercase letters, numbers, and underscores.



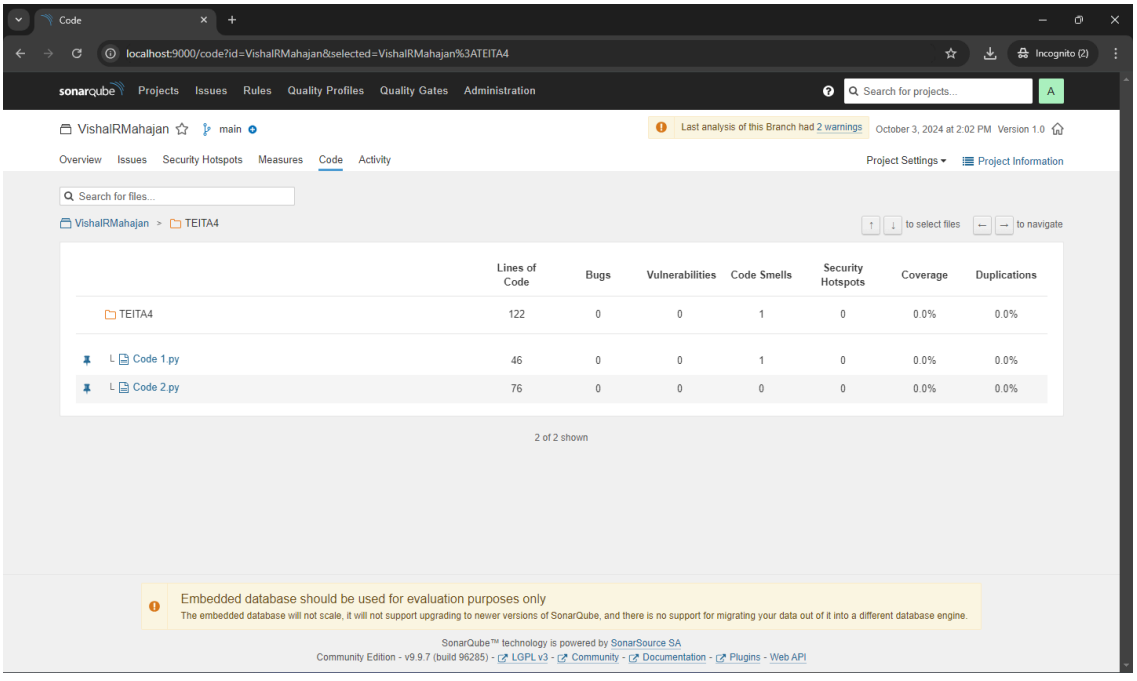
Security Hotspots: There are no Security Hotspots to review.



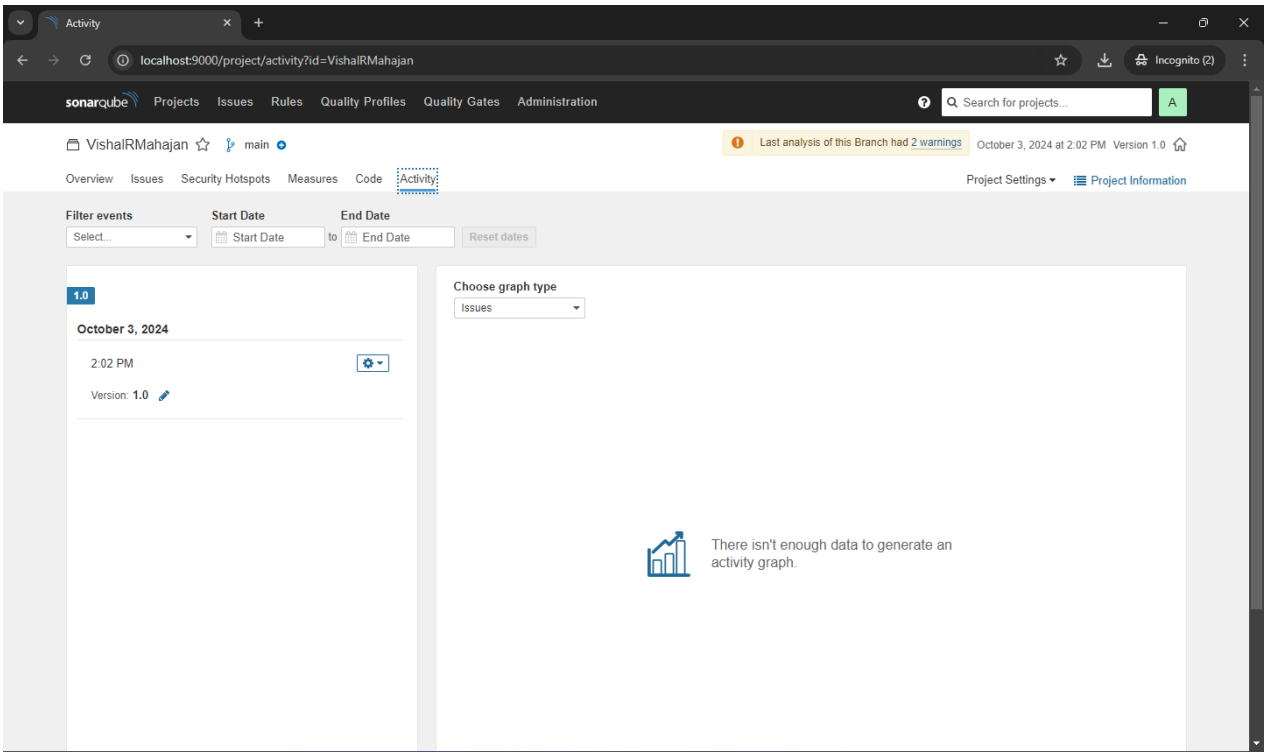
Measure: Risk is at 0%, indicating there are no identified vulnerabilities or security issues in the codebase, reflecting a clean status.



Code: The TEITA4 folder contains two Python scripts: Code 1 and Code 2. Code 1 has one code smell, while Code 2 has none. All other categories, including Bugs, Vulnerabilities, Security Hotspots, Coverage, and Duplications, are at zero for both scripts.



Activity : This tab provides detailed information about scan and test activities. It displays the date and time of each scan, along with the version of the scanned files.



8. Post-Experiments Exercise

A. Extended Theory:

- What is Code smell? (To be written in hand)

B. Questions:(soft copy)

Q.1 List characteristics of good quality code.

Q.2 Explain in short key traits to measure for higher quality

C. Conclusion:(To be written in hand)

Write the significance of the topic studied in the experiment.

D. References:

- https://medium.com/swlh/sonarqube-part-2-features-of-sonarqube-installation-and-some-practice-on-sonarqube-d523ae9a998a
- https://docs.sonarqube.org/latest/
- https://www.codeusingjava.com/interview/sonar

Q.1 List Characteristics of Good Quality Code:

1. **Readability**:
   - Code should be easy to read and understand by others, including yourself in the future. Proper indentation, naming conventions, and clear comments enhance readability.
2. **Maintainability**:
   - Code should be structured so it can be easily updated or modified in the future without introducing errors. This includes clear modularization and adherence to SOLID principles.
3. **Efficiency**:
   - Efficient use of resources (memory, CPU) without unnecessary complexity. The code should perform well in terms of time and space complexity.
4. **Reusability**:
   - Code should be designed in a way that it can be reused in different parts of the program or even in different projects, reducing redundancy.
5. **Testability**:
   - Code should be easily testable. Functions and modules should be designed to facilitate unit testing, with minimal dependencies on external systems.
6. **Consistency**:
   - Consistent use of coding standards (naming conventions, indentation style, etc.) throughout the codebase helps maintain quality.
7. **Modularity**:
   - Breaking the program into smaller, self-contained modules makes the code more understandable and maintainable. Each module should have a single responsibility.
8. **Robustness**:
   - Code should handle unexpected situations (e.g., incorrect inputs, null values) gracefully, with proper error handling and validation.
9. **Scalability**:
   - Good quality code should be able to handle increased load without requiring significant refactoring.
10. **Security**:
    - Code should be written with security in mind, preventing vulnerabilities like SQL injections, buffer overflows, or cross-site scripting.

Q.2 Explain in short Key Traits to Measure for Higher Quality Code:

1. **Code Coverage**:
   - Percentage of code covered by automated tests. Higher coverage means more of the code is tested and less prone to untested bugs.
2. **Cyclomatic Complexity**:
   - Measures the complexity of a program by counting the number of linearly independent paths through the code. Lower complexity indicates more straightforward, easier-to-maintain code.
3. **Code Smells**:
   - Identifying poor coding practices, such as duplicated code, long methods, or overcomplicated logic, which indicate a need for refactoring.
4. **Number of Defects**:
   - Count of bugs or issues found in the code. Fewer defects generally mean higher code quality.
5. **Performance Metrics**:
   - The code's runtime efficiency, memory usage, and response times, which help measure how well it performs under different conditions.
6. **Adherence to Coding Standards**:
   - Whether the code adheres to established conventions or guidelines within the team or organization, ensuring consistency across the codebase.
7. **Code Documentation**:
   - Well-documented code improves clarity, making it easier for others to understand its purpose and usage, especially for complex algorithms or APIs.
8. **Duplication Metrics**:
   - Reducing duplicate code to ensure maintainability and prevent inconsistent updates. Lower duplication results in more maintainable code.
9. **Test Pass Rate**:
   - The percentage of unit tests, integration tests, and end-to-end tests that pass successfully. A higher pass rate indicates more reliable code.
10. **Scalability**:
    - The ability of the code to handle increasing amounts of work or its potential to be extended without requiring significant rework.