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 request**s — 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.

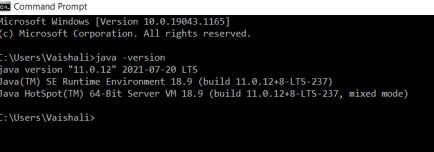
**Digs into issues**— If it shows that there is a problem SonarQube allows you to go and directly check it out from the summary report or from one code file to another. In the SonarQube summary dashboard, you can see furthermore details of the errors bu just clicking on the error.

**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.

1. **Laboratory Exercise**

Installation Steps

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

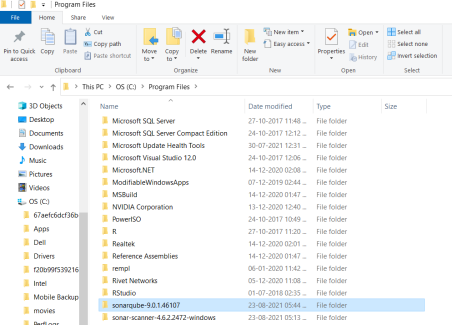
****

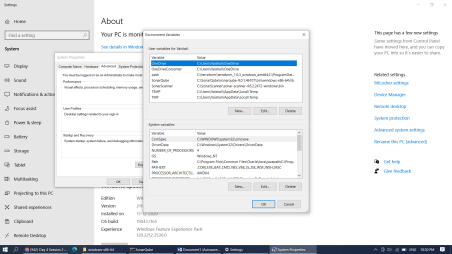
**Step B:** Download **SonarQube** from https://www.sonarqube.org/downloads/

SFIT, Department of Information Technology|2021

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

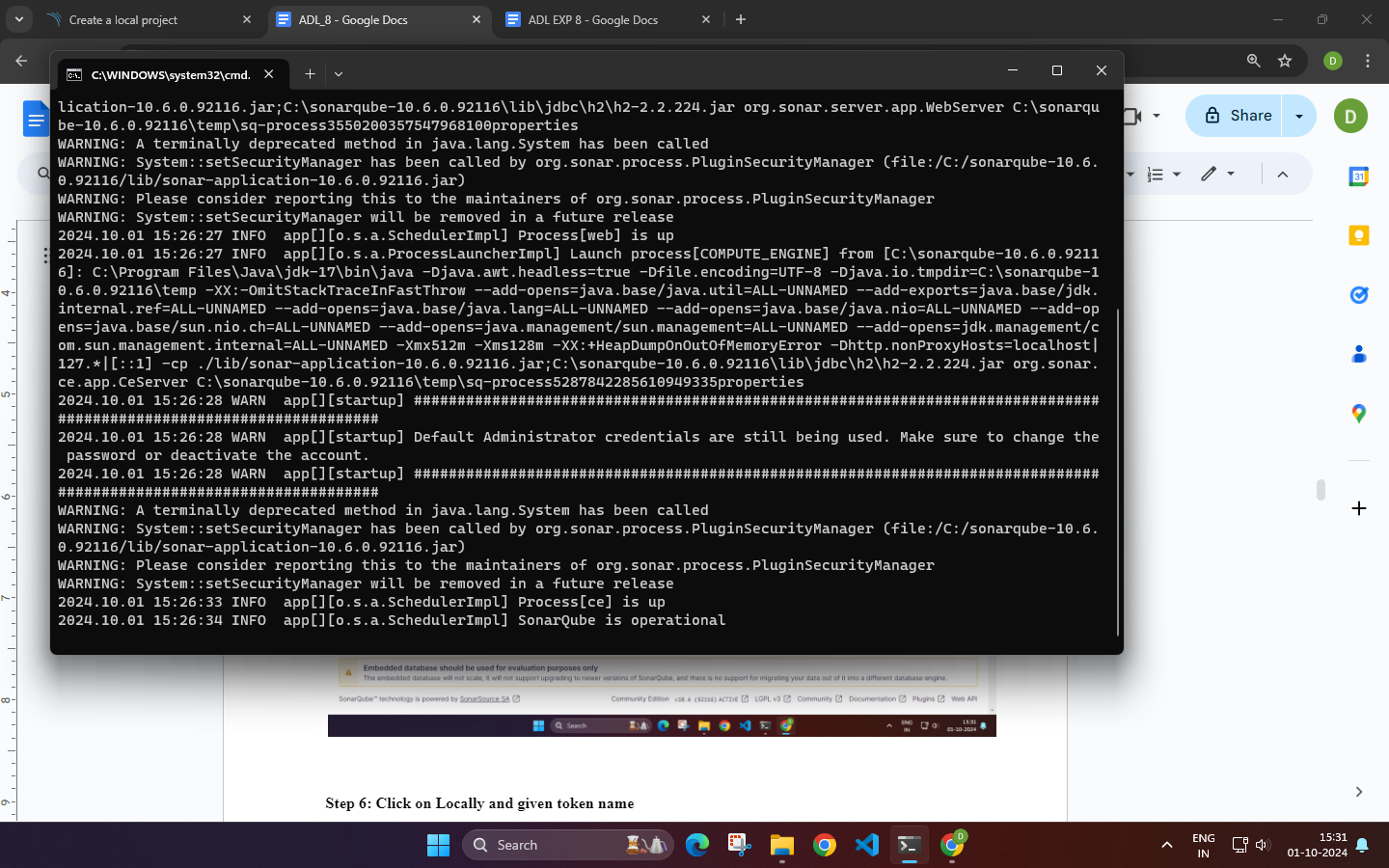
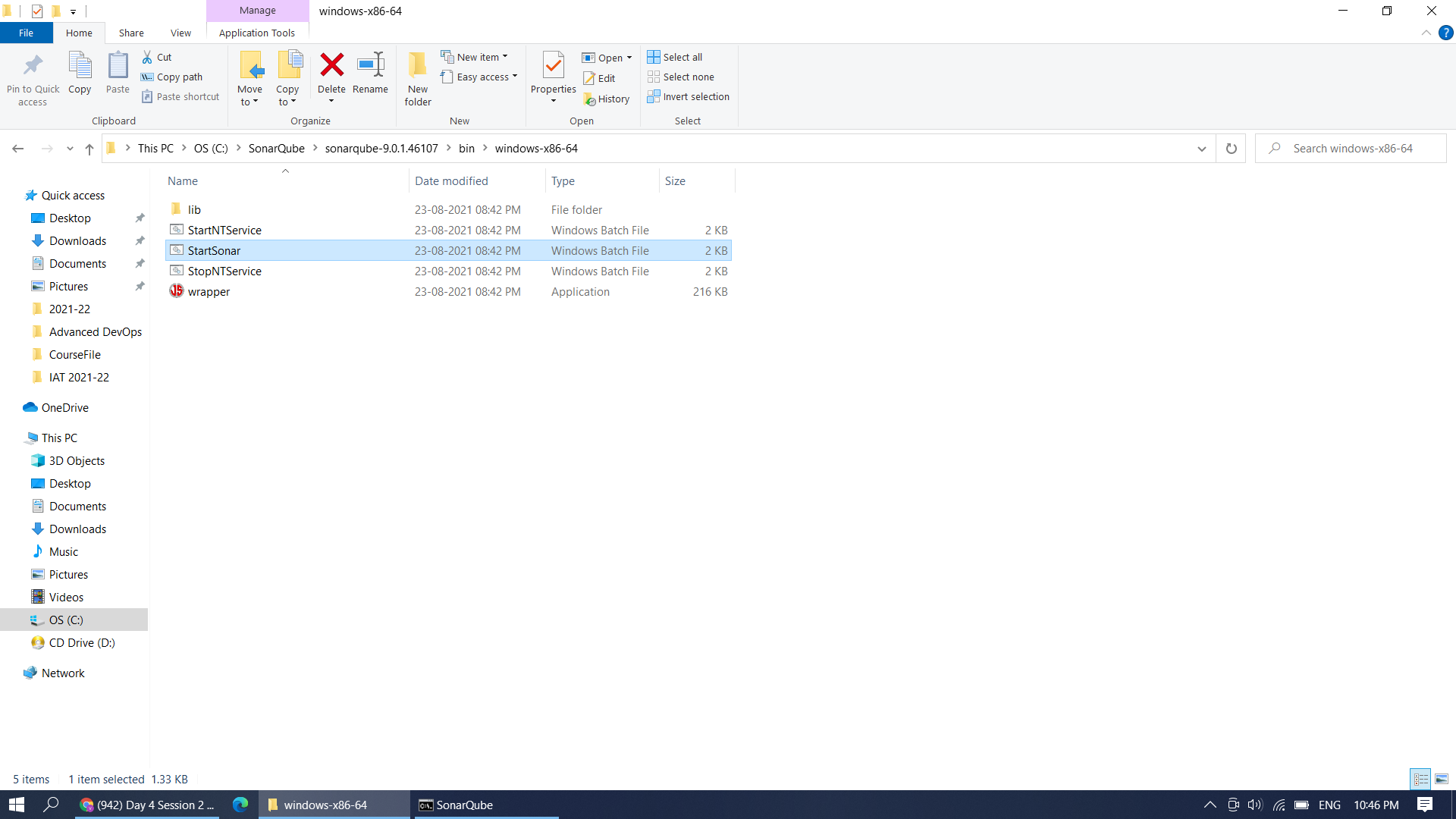
**Step D: Extract Zip files in C:/Program Files/SonarQube folder and C:/Program Files /SonarScanner folder**

****

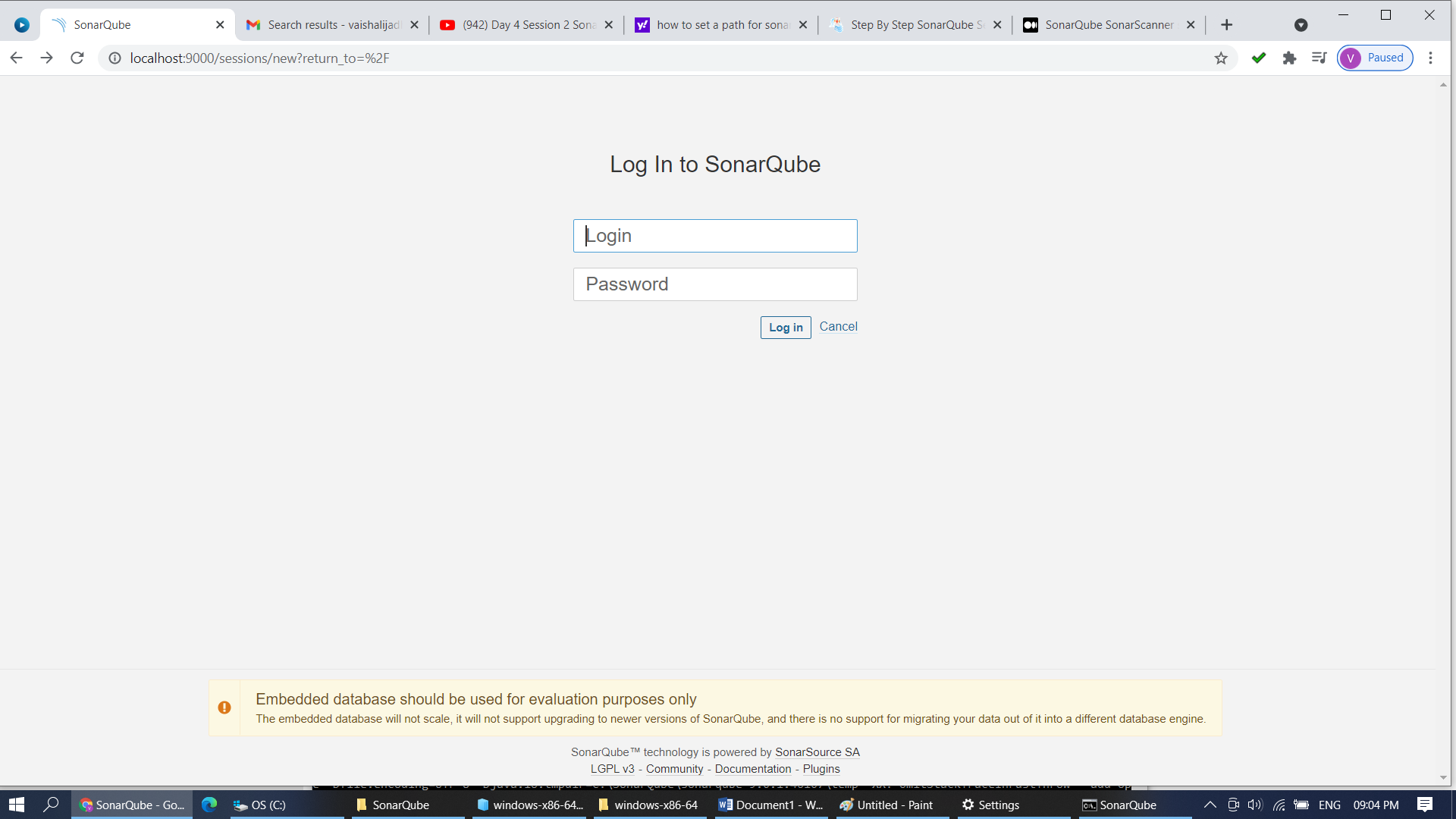


We installed and configured SonarQube and SonarScanner.

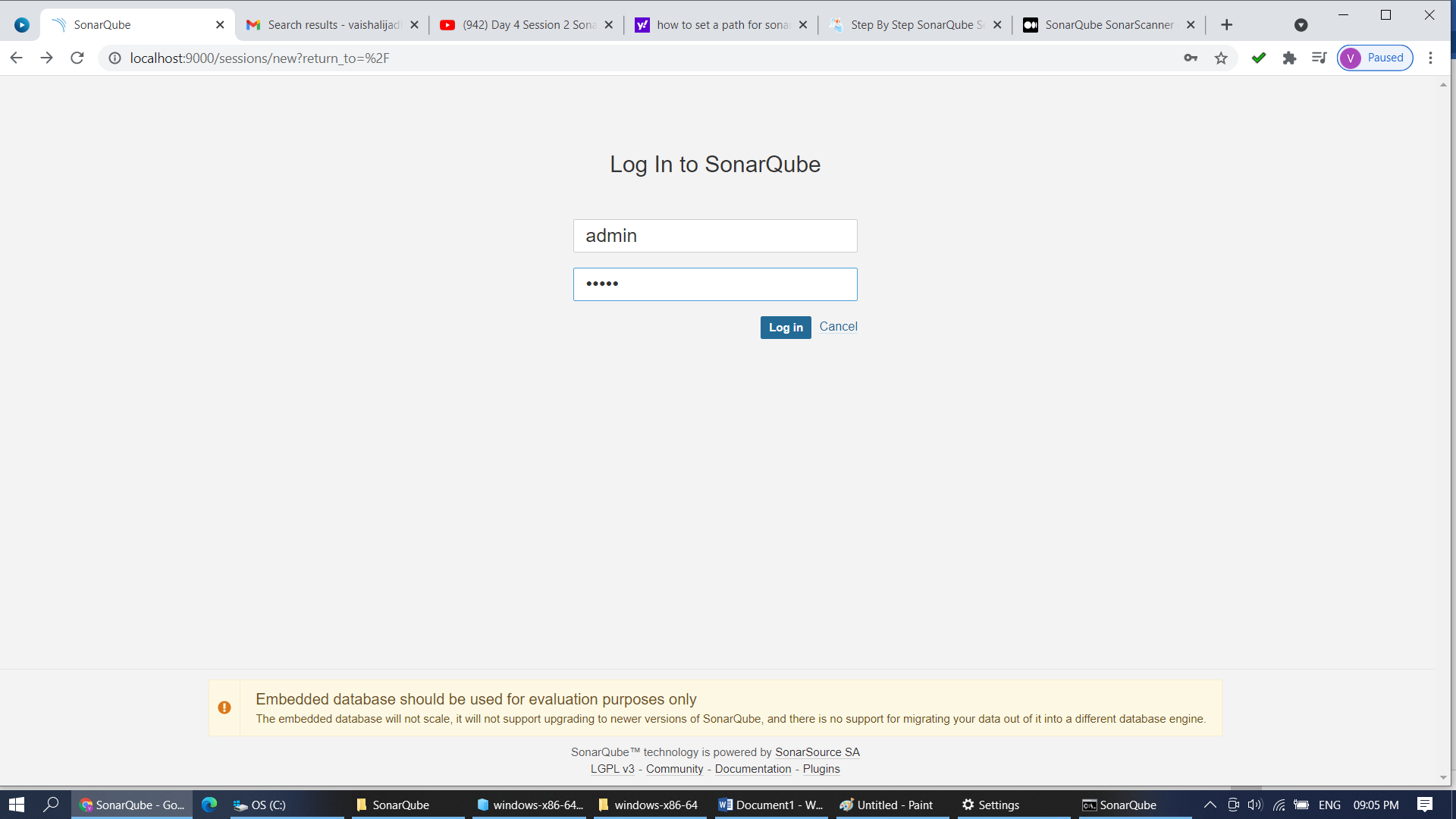
**Step 1: StartSonar server from SonarQube folder**



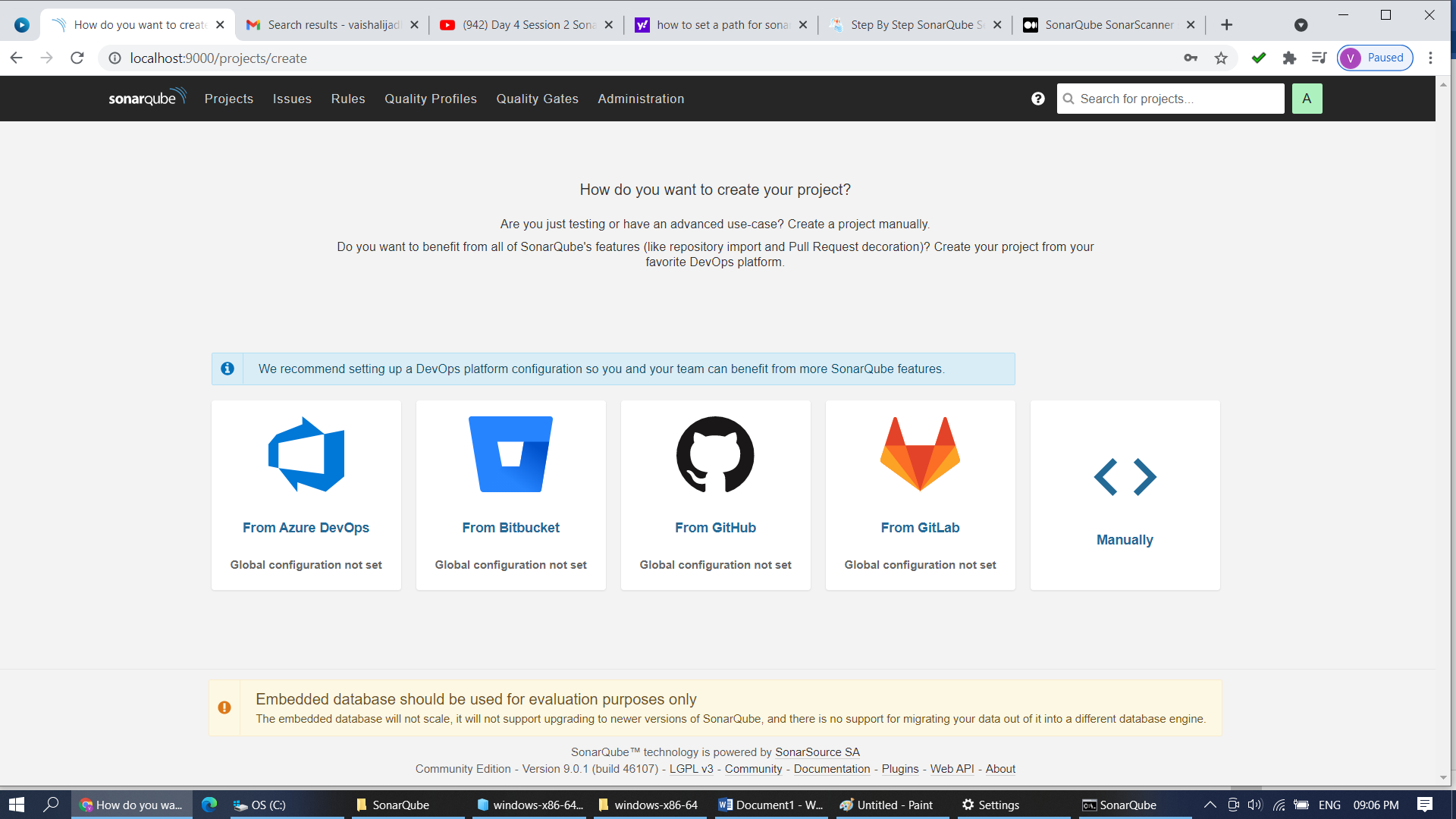
**Step 2: : on browser chk** [**https://localhost:9000**](https://localhost:9000)



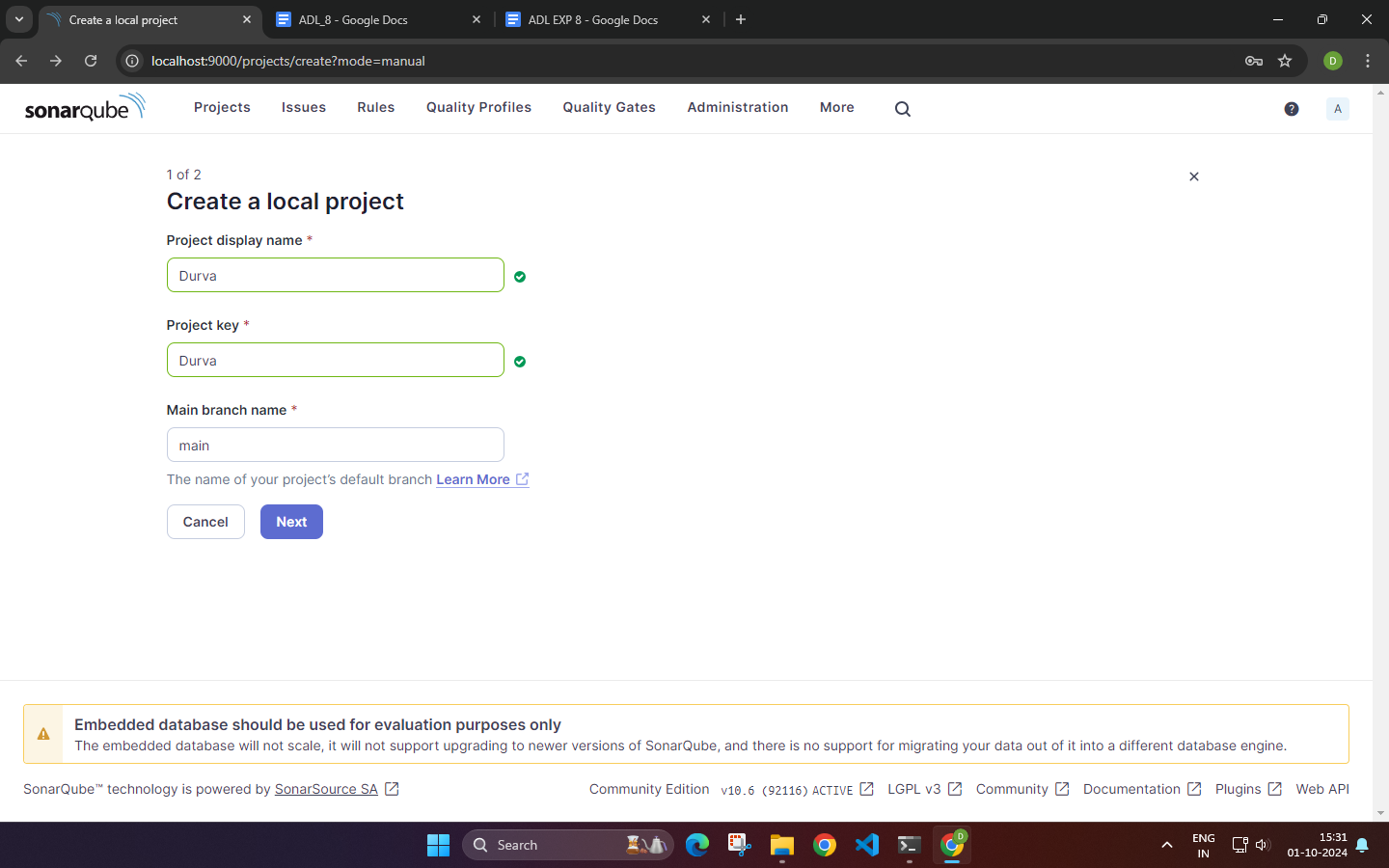
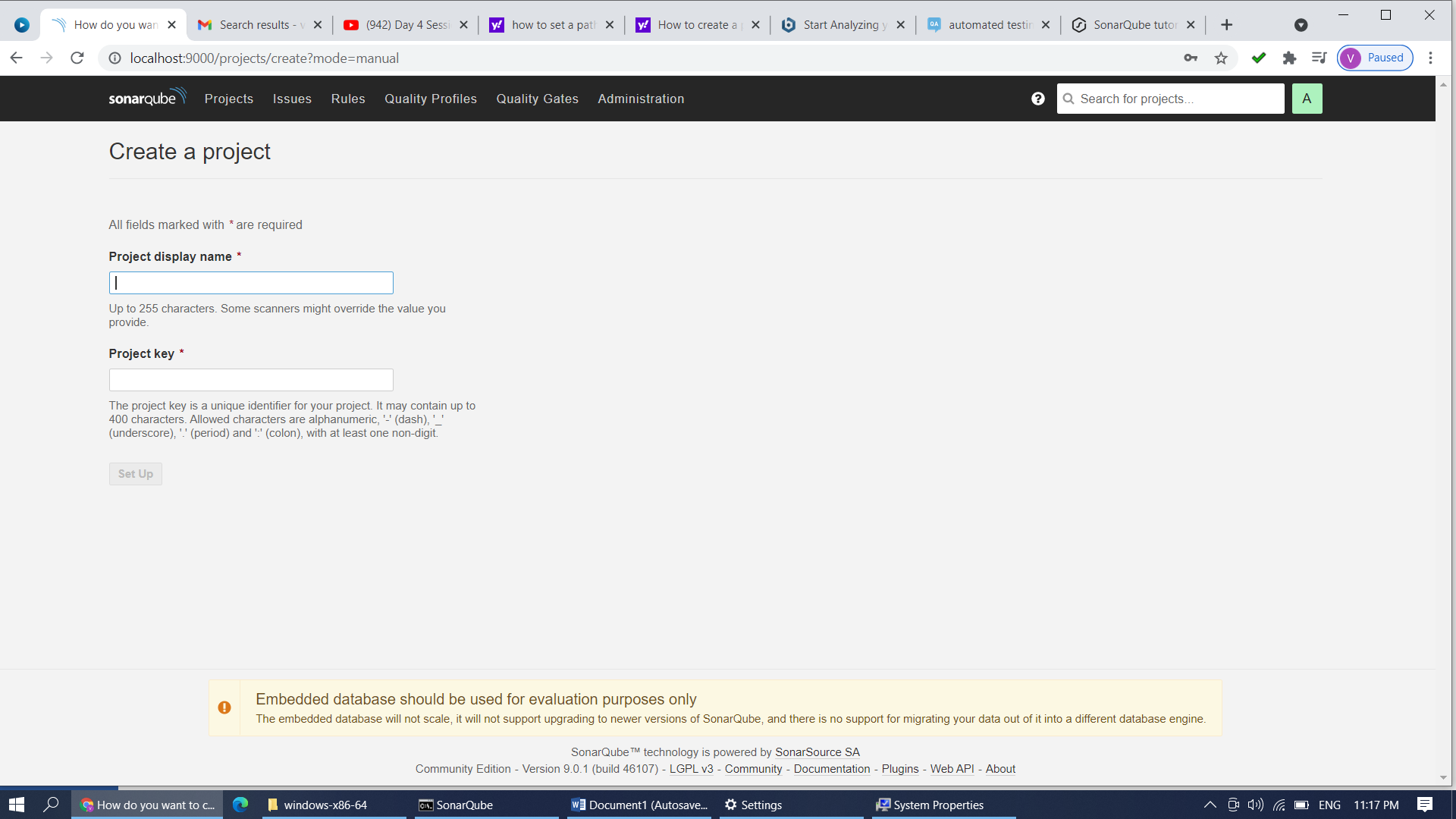
**Step 3: By default Username and Password is admin. You can change the password here.**



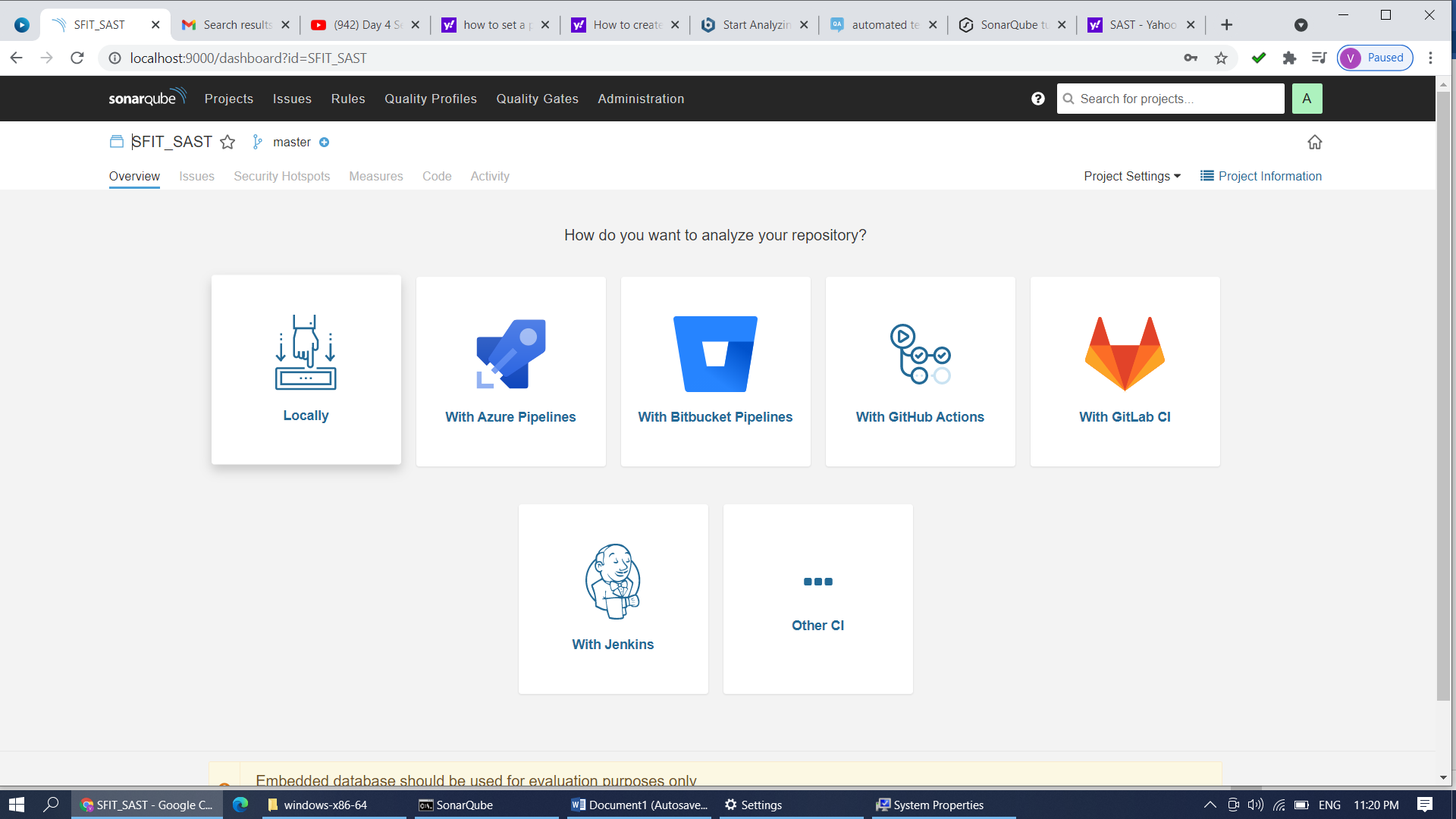
**Step 4: Login with new password.** 



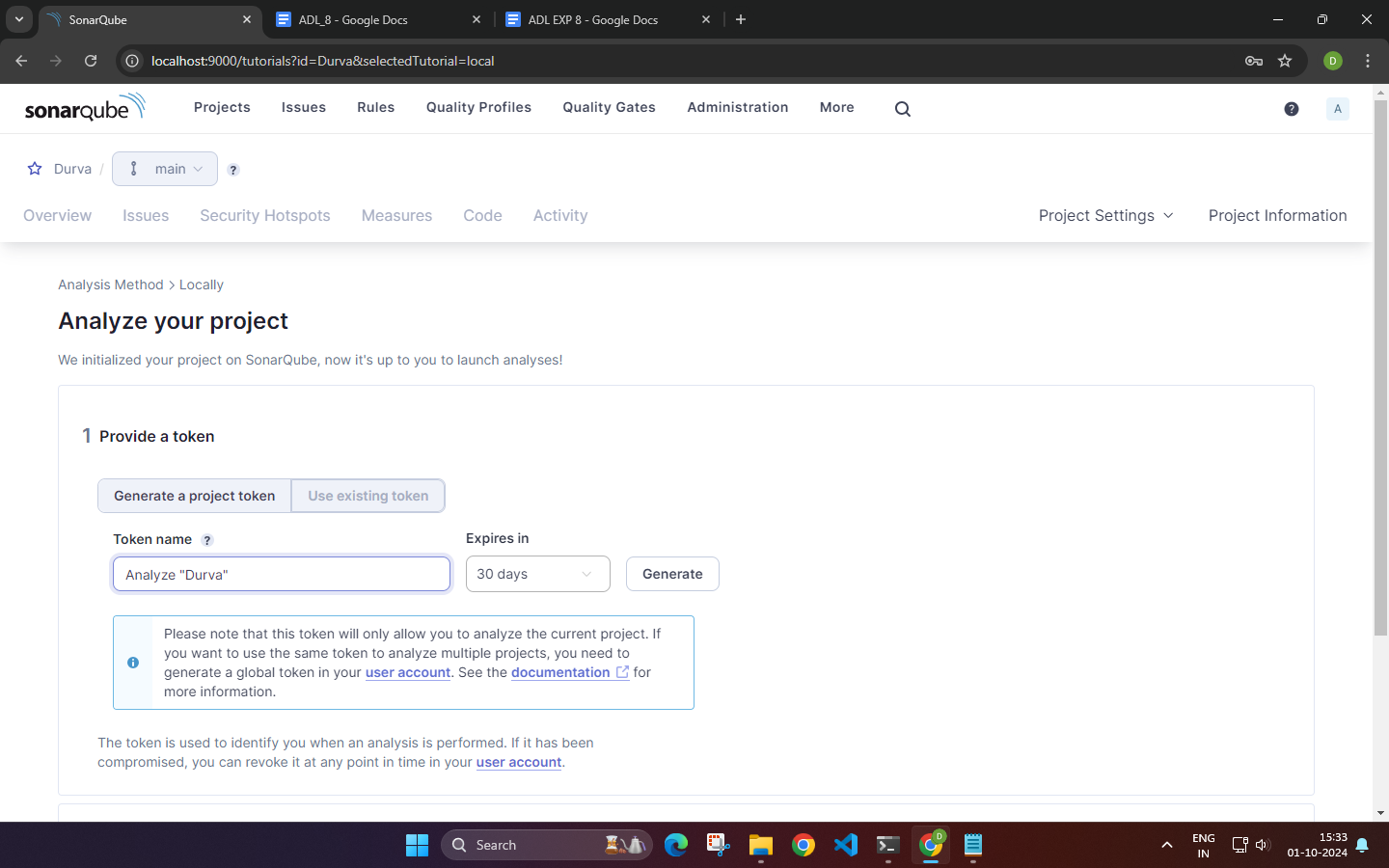
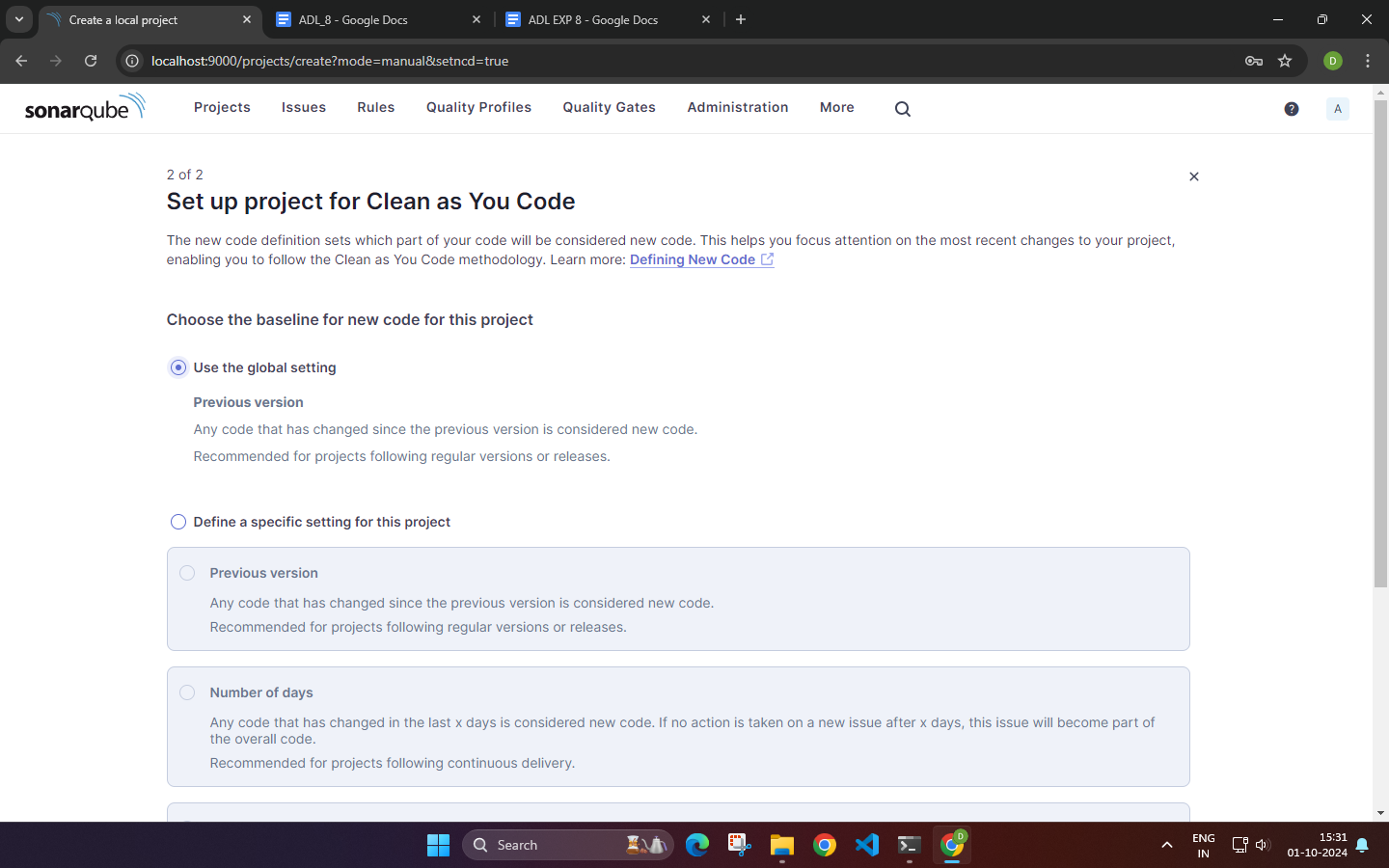
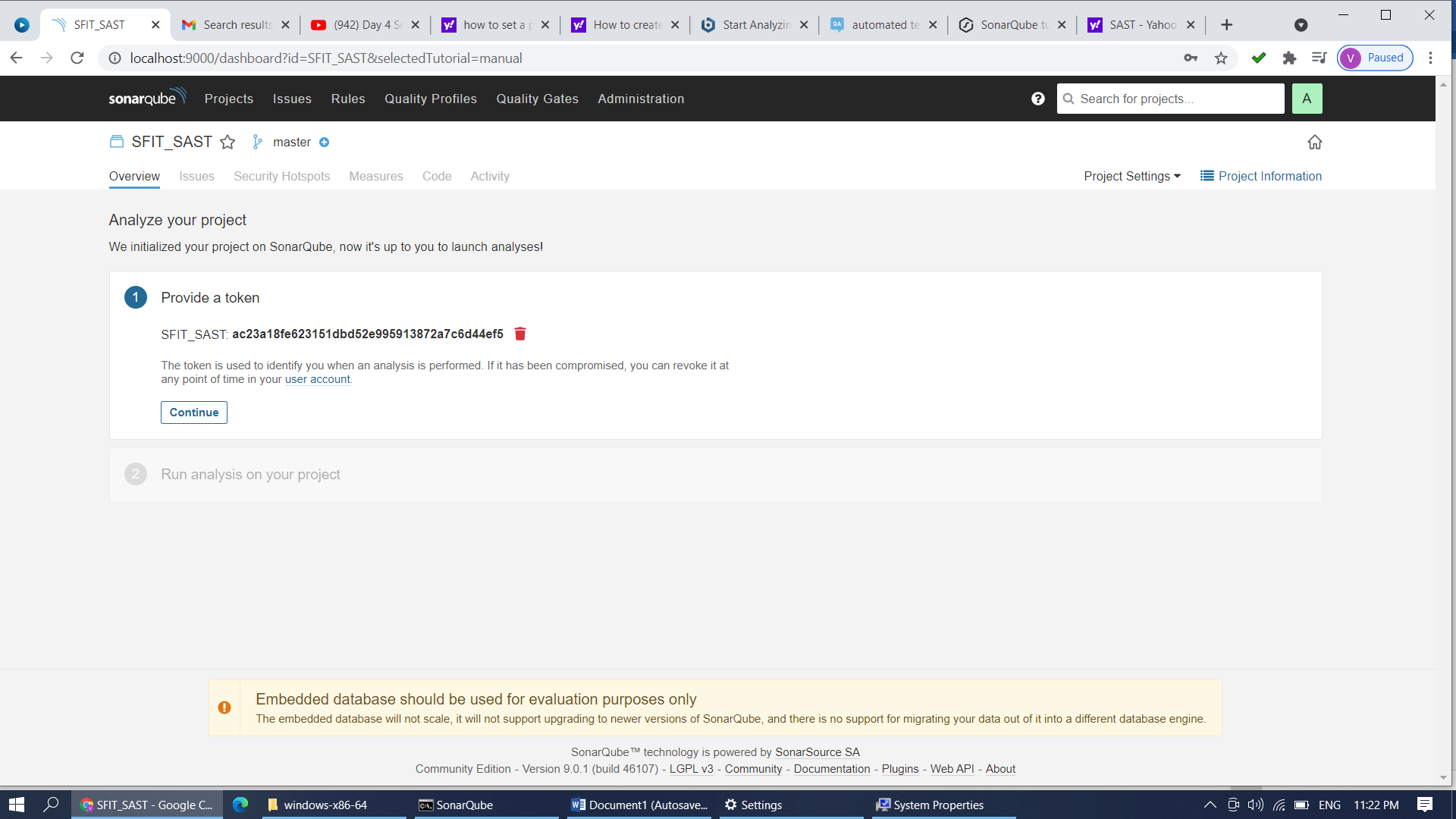
**Step 5: Click on Manually. Create a Project named SFIT\_SAST**



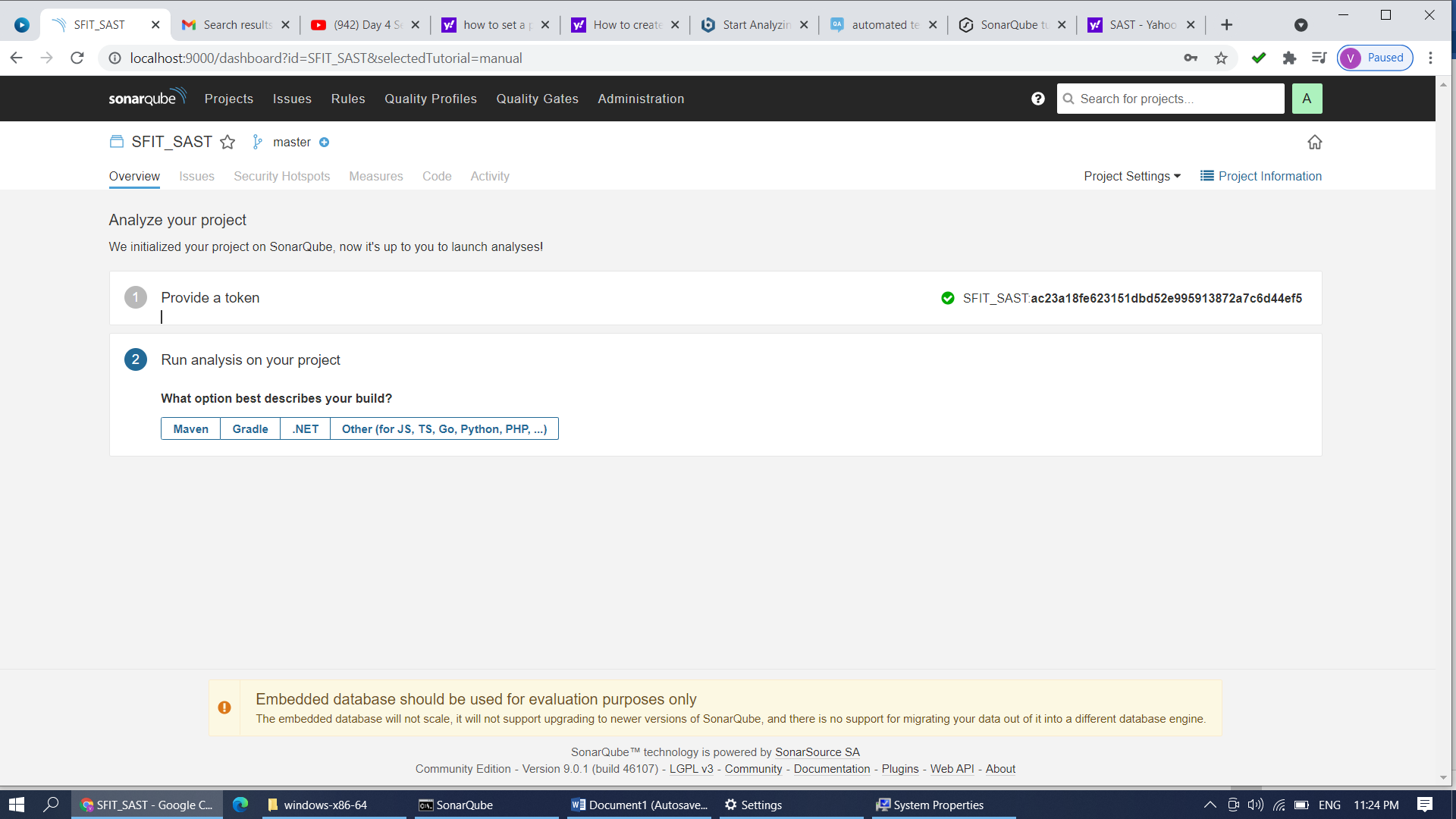
**Step 6: Click on Locally and given token name**



**Step 7: Give token name SFIT\_SAST and continue**



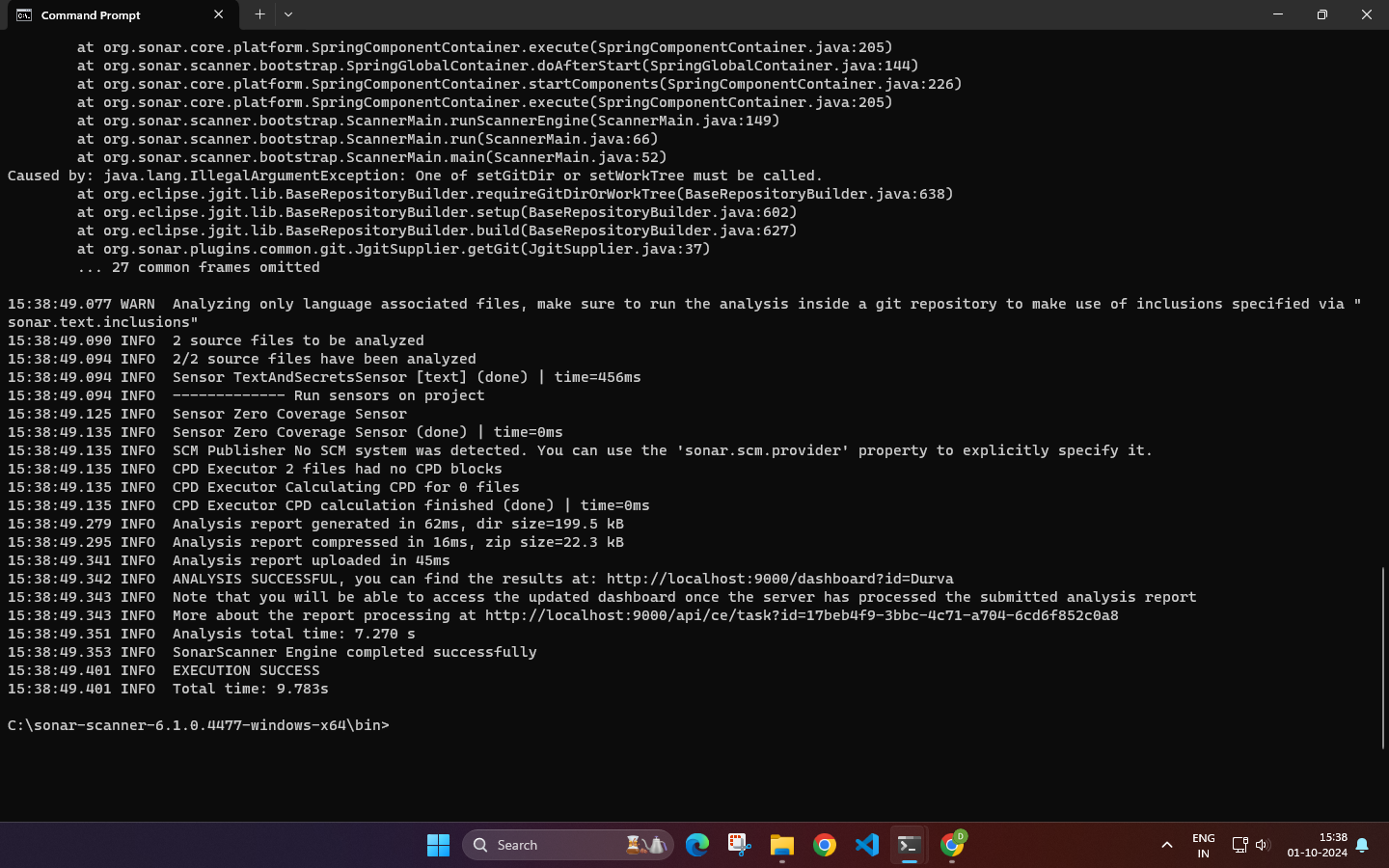
**Step 8: Select what type of project you want to test**



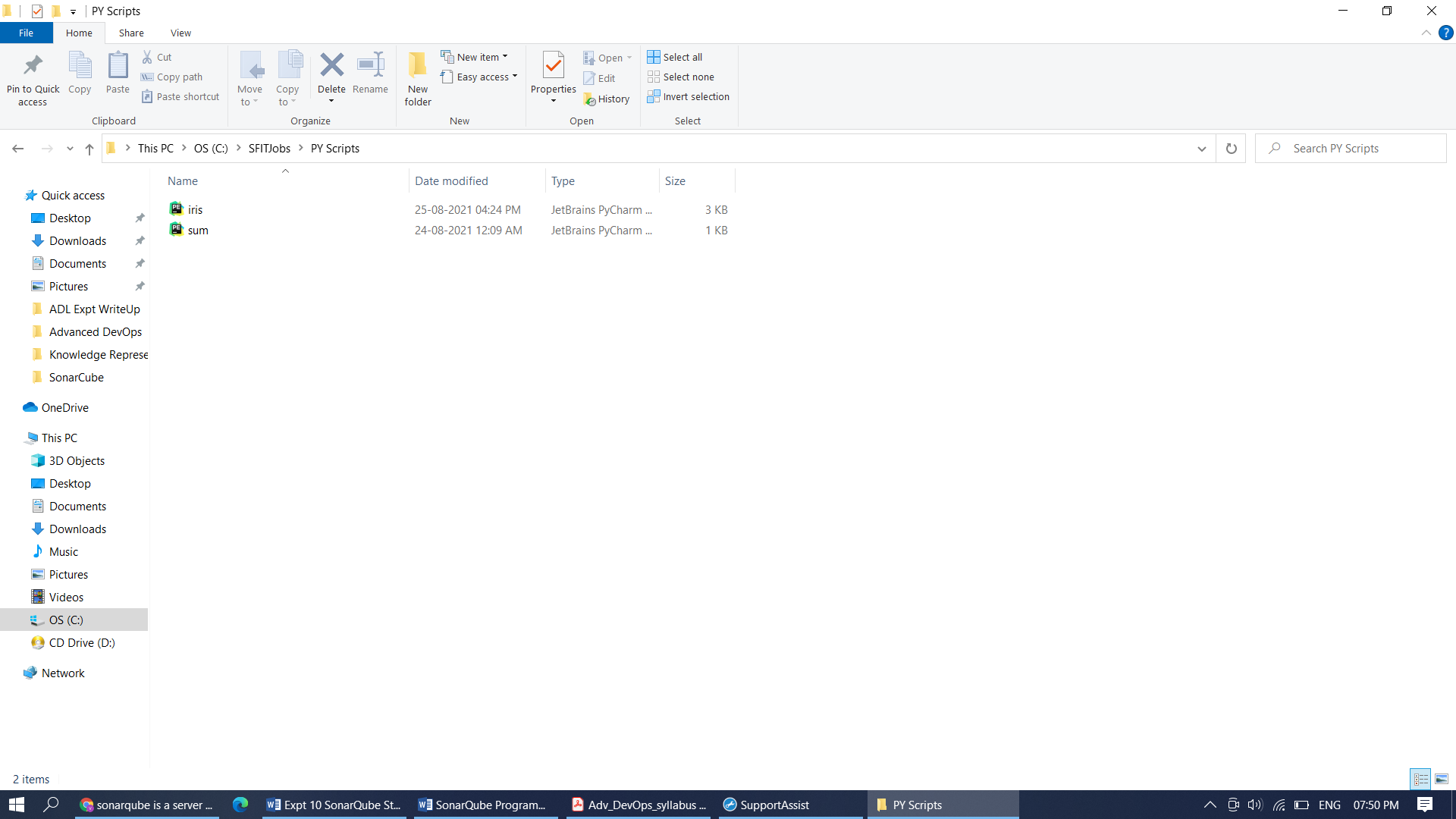
**Step 9: Click on Other and select OS as Windows**



**Step 10: Copy that command …Open new command prompt and paste that command after configuration of scanner.**

****

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



**Start configuration of the scanner. Add following lines to C:/SonarScanner/conf/Sonar-Scanner.properties**…Open that properties file **with Notepad** and do the following changes.

#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=SFIT\_SAST3**

**sonar.projectName=SFIT\_SAST3**

**sonar.projectVersion=1.0**

**#sonar.projectBaseDir=C:\SFITJobs**

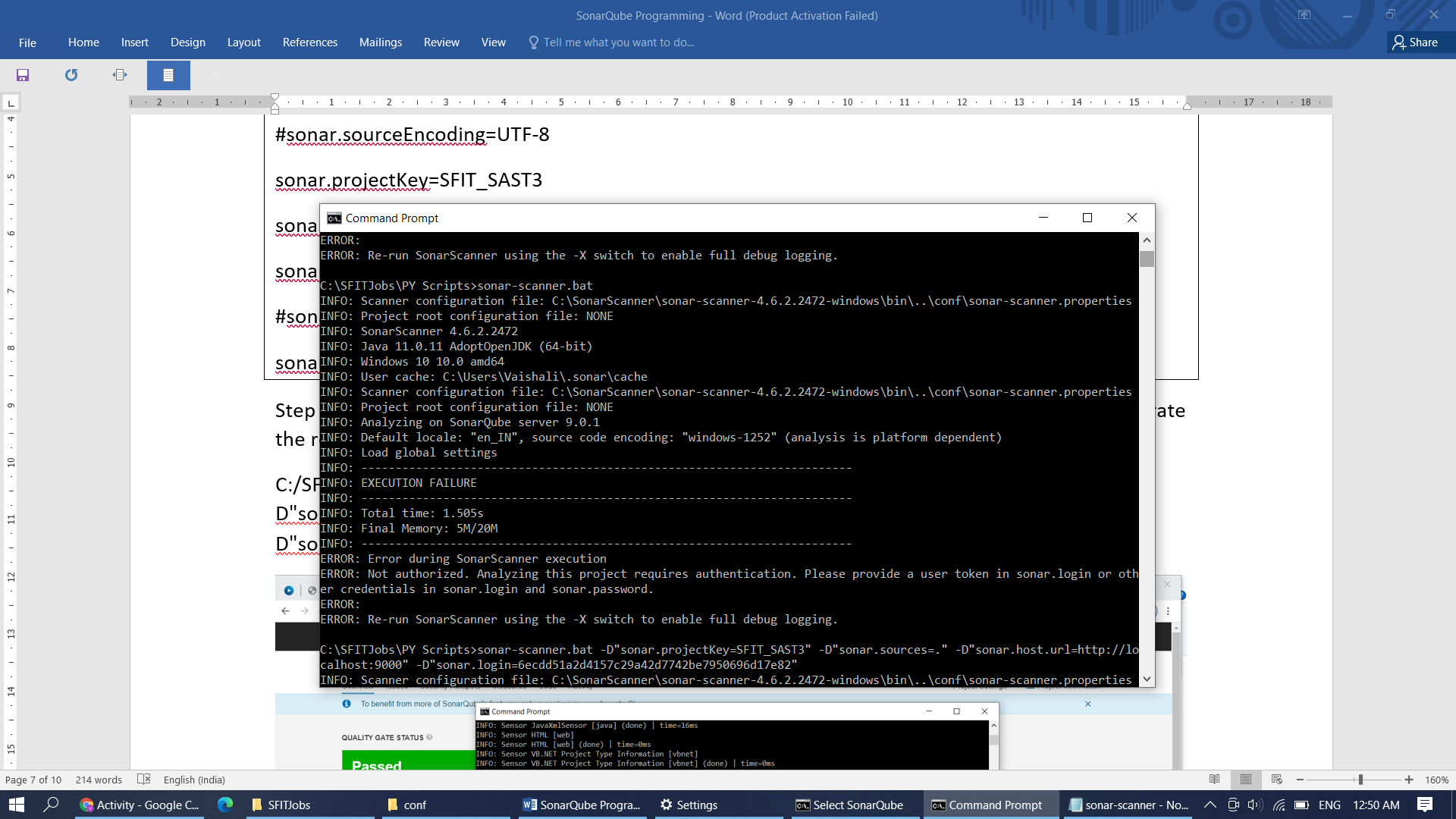
**sonar.sources=C:\SFITJobs\PY Scripts**

Step 12: Save the above file. Open new command prompt. Go to C:/SFITJobs/Py Scripts and Run the copied command (from step 10) from dashboard on it. It will generate the report.

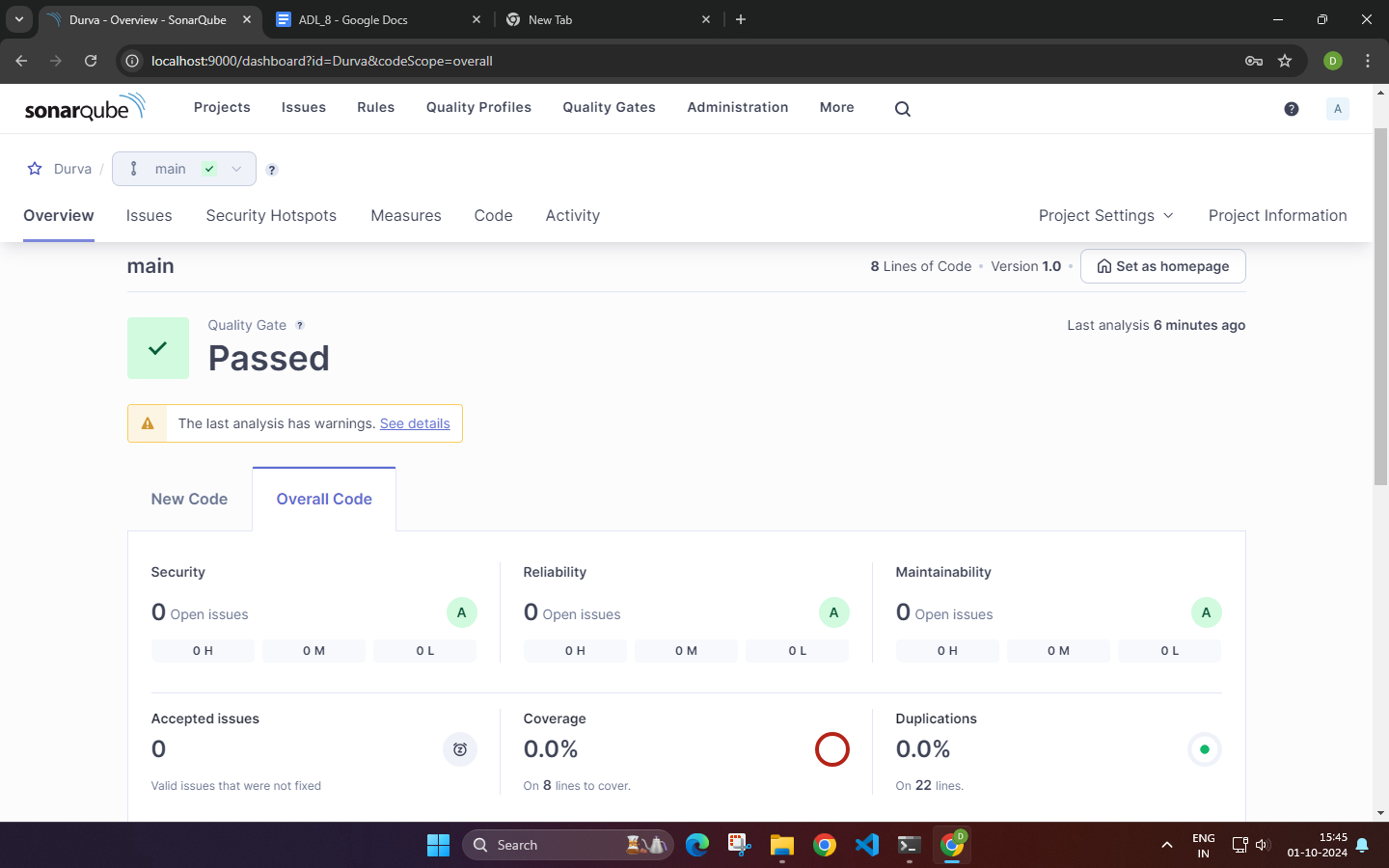
C:/SFITJobs/PY Scripts >sonar-scanner.bat -D"sonar.projectKey=SFIT\_SAST" -D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -D"sonar.login=edabde9219cd89080688d93f3ff57ee8ba4caaf4"

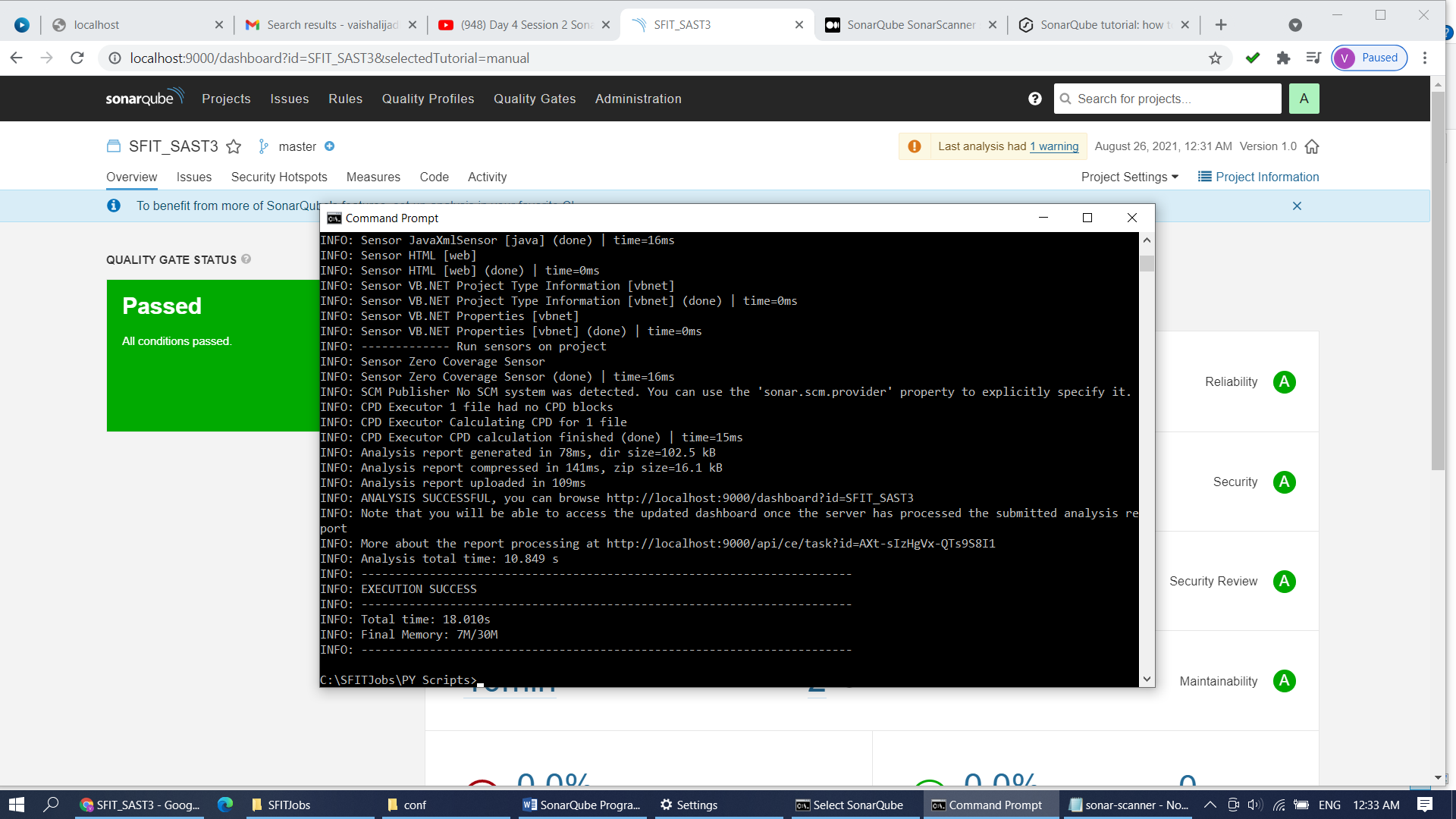
If Scanner is not get configured properly you may get an error.

Error File Screenshot: (Execution Failure)

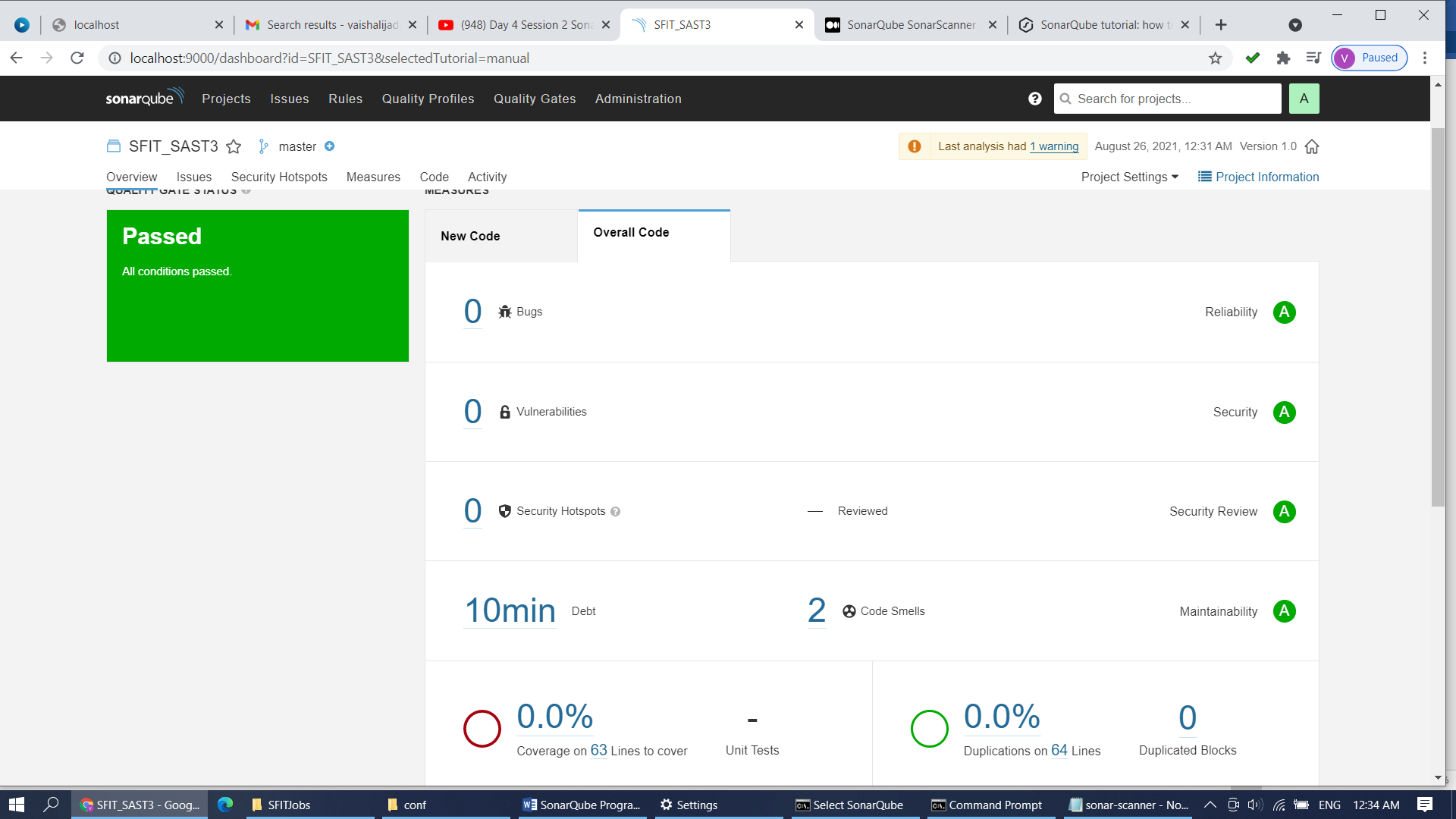


If no error in Sonar Scanner and SonarQube configuration then copied command will give output on dashboard. Command Prompt is showing Execution Success and dashboard will give Quality Gate Status : Passed



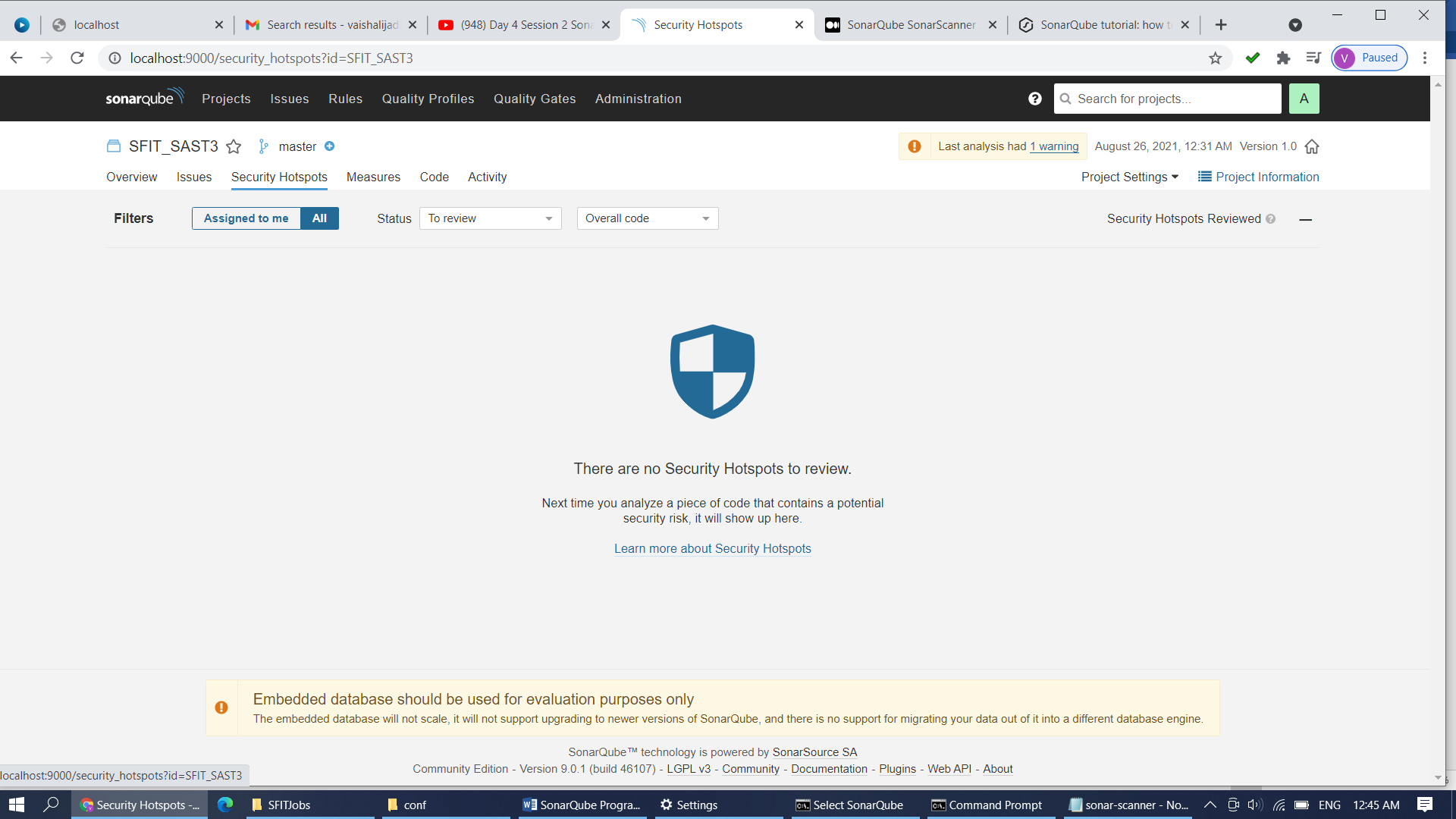


Check each tab i.e. Overview,issues, security hot spots, measures, code and activity.

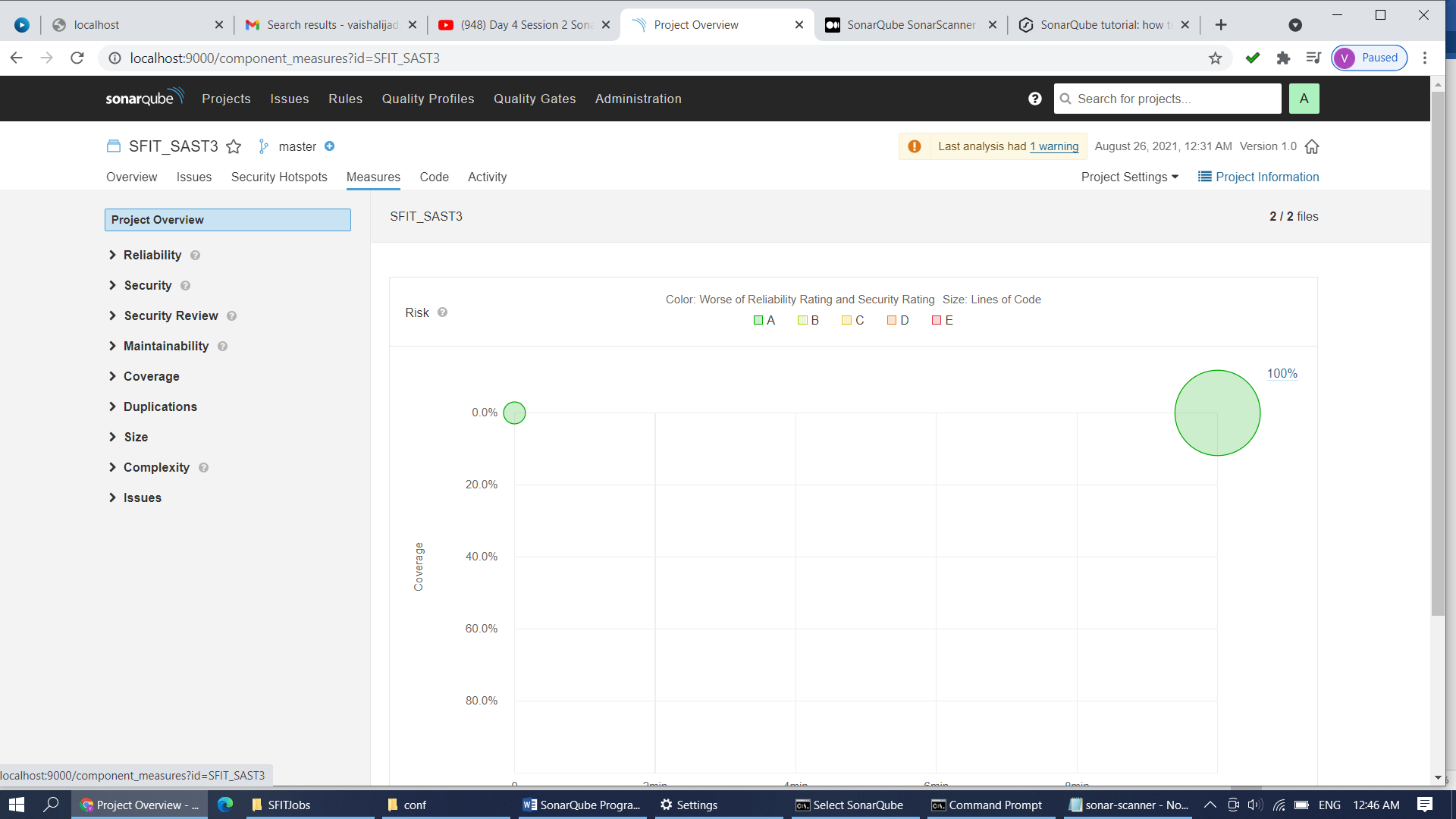


Issues tab give 2 code smells in IRIs.py file. Click on it and check the unnecessary code in the file. Iris.py Sonar Scanner is recommending to remove these unnecessary import lines from the code file iris.py.

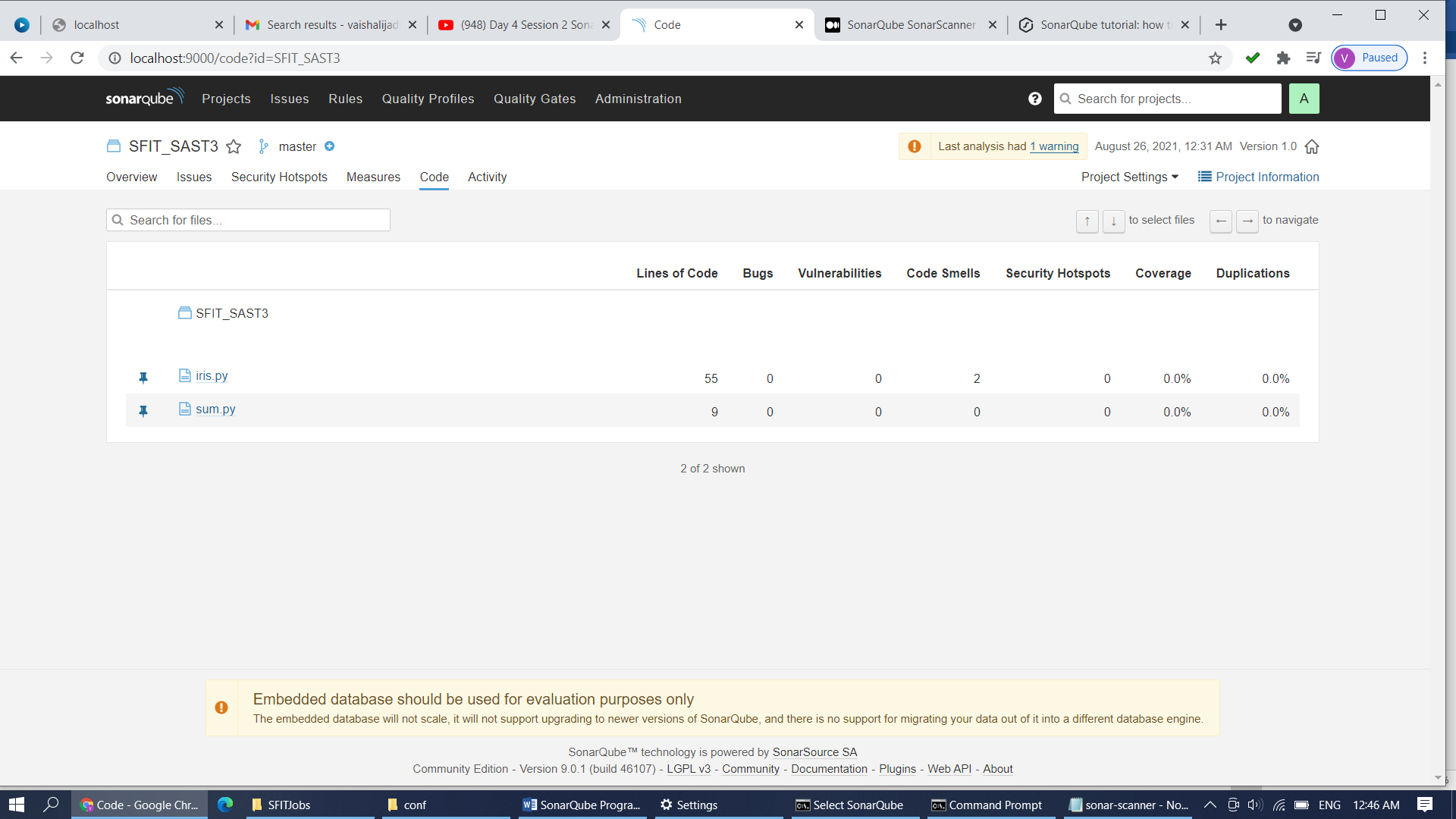
No security hotspots are seen in PY Scripts folder.



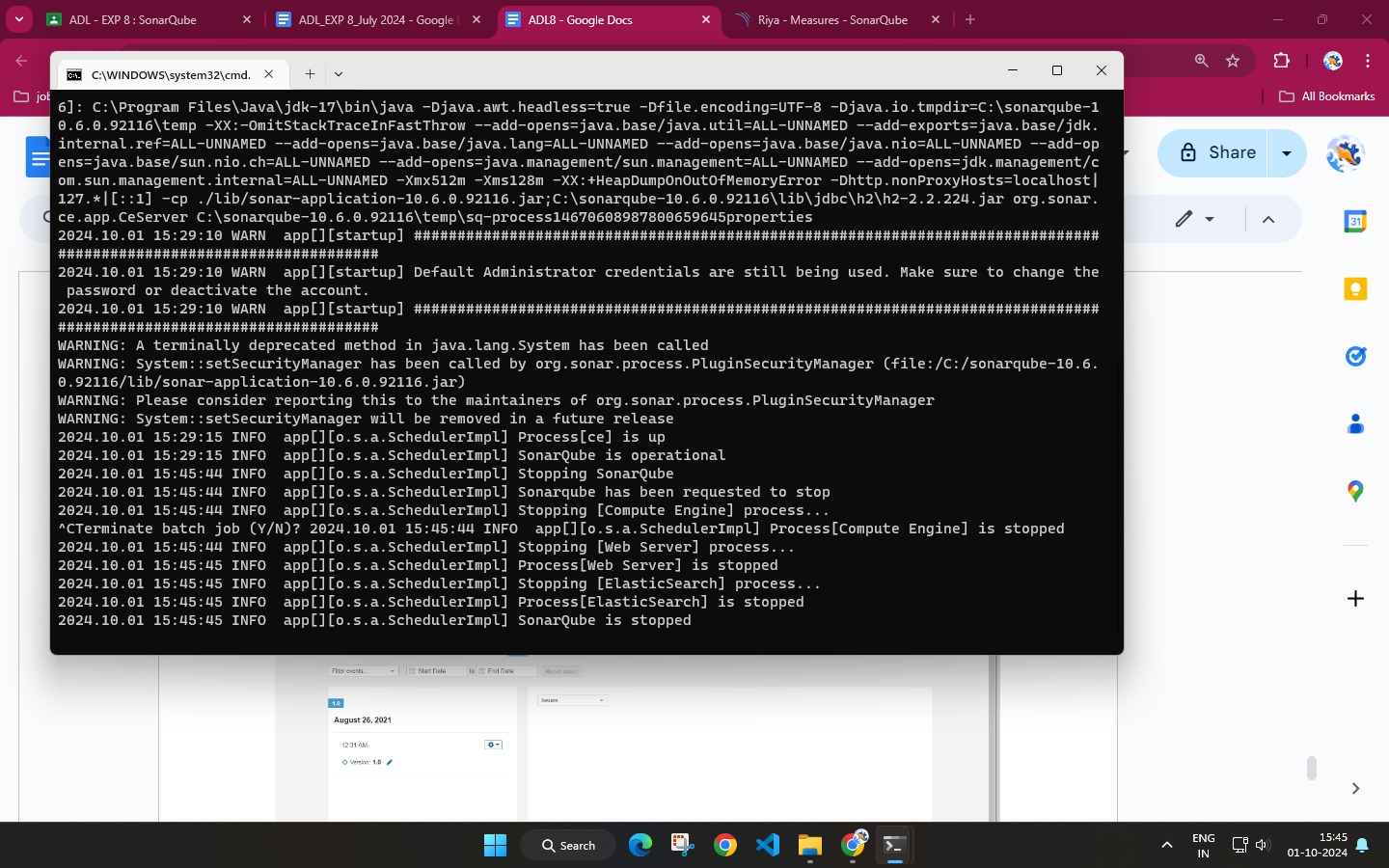
Risk is 0% while deploying PY Scripts folder.



Py Scripts Folder contains two files iris.py and sum.py …iris.py contains 2 code smells.



Activity tab will give scan and test activity details. i.e date, time ,version of scanned file .



1. **Post-Experiments Exercise:**
2. **Extended Theory:**

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

1. **Questions:**(soft copy)

**Q.1 List characteristics of good quality code.**

**ANS:**

1. **Clarity**: Code should be easy to understand, with clear logic, structure, and intent.
2. **Simplicity**: The code should solve the problem in the simplest way, avoiding unnecessary complexity.
3. **Modularity**: Functions and components should be modular, promoting reuse and easy debugging.
4. **Efficiency**: Code should be optimized for performance and resource usage without sacrificing readability.
5. **Robustness**: Good code handles edge cases and unexpected inputs gracefully without crashing.
6. **Scalability**: The code should be able to accommodate future growth in terms of data, users, or functionality.
7. **Consistency**: Consistent coding style, naming conventions, and structure across the codebase to make collaboration smoother.
8. **Testability**: The code should be written in a way that makes it easy to write tests, ensuring high reliability.
9. **Security**: Secure coding practices should be followed to prevent vulnerabilities and ensure data safety.
10. **Documentation**: Proper comments and documentation to explain why certain decisions were made and how the code works.

Good quality code is essential for building reliable, maintainable, and scalable software. It ensures that the codebase is efficient, secure, easy to read, and adaptable to future changes, ultimately leading to successful long-term project outcomes.

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

**ANS:**

1. **Code Efficiency**: The time complexity and space complexity of the code, ensuring optimal performance.
2. **Modularity**: How well the code is divided into distinct, reusable, and manageable pieces or functions.
3. **Error Handling**: How gracefully the code handles exceptions and unexpected situations, minimizing crashes or failures.
4. **Security**: The extent to which the code protects against unauthorized access, data breaches, and vulnerabilities.
5. **Test Coverage**: The percentage of the code that is covered by automated tests, ensuring that every functionality is validated.
6. **Code Duplication**: Minimal or no duplication of code, ensuring that similar functionality is encapsulated in functions or modules.
7. **Complexity**: Keeping the code as simple as possible by avoiding unnecessary nested conditions or loops.
8. **Consistency in Style**: Maintaining uniformity in the code structure, naming conventions, and formatting.
9. **Adherence to Standards**: Following industry standards and best practices in coding, design patterns, and architecture.
10. **Documentation Quality**: The clarity and completeness of in-line comments and external documentation to explain the purpose and usage of the code.

By focusing on these traits, higher-quality code can be achieved, ensuring the system remains scalable, secure, and maintainable over time.

**C. Conclusion:**(To be written in hand)

Write the significance of the topic studied in the experiment.

1. **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>

---------------------------------