**Detailed Design Document**

|  |  |
| --- | --- |
| **Created By** |  |
| **Status** |  |
| **Version** |  |
| **Date** |  |

**Approvers**

|  |  |  |
| --- | --- | --- |
| **Name** | **Project Role** | **Date of Approval** |
|  |  |  |
|  |  |  |
|  |  |  |

**Document Information**

|  |  |
| --- | --- |
| **Location** |  |

History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Modified By** | **Changes** |
| 1.0 |  | Draft Version |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Document Review and Sign-off

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Role** | **Action** | **Version** | **Date Reviewed** |
|  |  |  |  |  |
|  |  |  |  |  |

# Contents

[1 Contents 3](#_Toc91253007)

[1. SonarQube Introduction 4](#_Toc91253008)

[1.1. Context 4](#_Toc91253009)

[1.2. Purpose of Sign-Off 4](#_Toc91253010)

[1.3. In Scope 4](#_Toc91253011)

[2 SonarQube Design and Details 5](#_Toc91253012)

[2.1 Continuous Quality - CICD 5](#_Toc91253013)

[2.2 Bugs, Code Smell, & Security 5](#_Toc91253014)

[2.3 Developing the Repository 5](#_Toc91253015)

[2.4 Decision Making 5](#_Toc91253016)

[3 SonarQube Configuration 6](#_Toc91253017)

[3.1 Java SE Requirement 6](#_Toc91253018)

[3.1.1 SonarQube Server Installation 7](#_Toc91253019)

[4 Running SonarQube Project 9](#_Toc91253020)

[4.1 SonarQube Project Creation 9](#_Toc91253021)

[4.2 SonarQube Project Evaluation 9](#_Toc91253022)

[4.2.1 SonarQube Project Token 9](#_Toc91253023)

[4.2.2 SonarQube Project Scanner 10](#_Toc91253024)

[4.2.3 Executing the Scanner 11](#_Toc91253025)

[5 SonarQube Dashboard Results 12](#_Toc91253026)

# SonarQube Introduction

## Context

SonarQube is an open source, automatic code review tool to detect bugs, vulnerabilities, and code smells in your code. It can integrate with your existing workflow to enable continuous code inspection across your project branches and pull requests.

To ensure overall quality of the code and to ensure that we are not lacking on the quality of the code, we need a standardized policy for code improvement. When multiple developers are working on the same project, it's good to have a code review. SonarQube is a tool through which we can evaluate our code.

The objective of the document is to provide the below-mentioned details

* A brief overview of the SonarQube application in the current scope.
* Sequence diagram(s) for key use case(s)
* Description of the components used in SonarQube.

## Purpose of Sign-Off

Document sign-off indicates that the signing authority agrees with the adopted and stated Detailed Design Document.

## Intended Audience

The document is intended for the use of the following audience.

|  |  |
| --- | --- |
| Audience | Objectives |
| Architect Team | To review, provide feedback and approve the proposed detailed design for the platform. |
| Development Team | To align the actual implementation of the platform with the guidance provided in this document. |
| Test Team | To understand the detailed level architecture and design of the platform and identify the testable non-functional requirements for preparing test cases. |
| Support Team | To understand the system design of the platform to enable support. |
| Technical Operations Team | To understand the detailed system design of the platform as an input towards the preparation for operational testing. |
| Infrastructure Team | To understand the overall solution and its impact on infrastructure design. |
| Security Team | To understand security aspects of the platform and interaction between components, from security perspective. |

## In Scope

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tool | Environment | Source | Target | Dashboard | Result |
| SonarQube | On-Premise | .Net 5.0 | MSBuild | Browser based (localhost) | Code Smells |

# SonarQube Design and Details

## Continuous Quality - CICD

SonarQube is easy to pair with a Continuous Integration and Deployment (CICD) platform. It introduces the notion of Continuous Quality, which is easy to digest in the context of CICD pipelines. Such a pipeline would pass the code through SonarQube in an automated fashion to ensure Continuous Quality. Qualitative inspections provide not only insights into the health of the source code, but also the ability to highlight potential new risks. SonarQube also detects vulnerabilities that extend beyond the domain of code design.

## Bugs, Code Smell, & Security

All the below metrics can be found in the SonarQube dashboard.

1. **Reliability**: Covered by bug detection.
2. **Security**: Covered by the detection of points of weaknesses, and problems related specifically to the security of the code.
3. **Maintainability**: Inferred based on the following two factors:
4. **Code Smell:** Determined by the code’s conformity to best practices.
5. **Technical Debt**: An approximation of the time required to understand the code-base.
6. **Coverage:** A measure of the rate of code covered by tests.
7. **Duplication:** A measure of the rate of code that is repeated across the code-base.
8. **Size**: A set of statistics about the code-base such as: number of files, functions, classes etc.
9. **Complexity**: A measure of the cyclomatic complexity of control flow in the code.

## Developing the Repository

SonarQube offers two major ways to adapt the standards and requirement levels for each project.

* **Quality Profiles:** This feature allows you to define the standards and best practices for each programming language. While there are several preset industry standards such as PSR-2 for PHP users, SonarQube’s community has also contributed various other quality standards. It is quite possible to extend Quality Profiles by adding additional rules to define custom standards.
* **Quality Gates:** Quality Gates define a set of conditions to be met for code quality to be considered sufficient. They can be applied universally or on a case-by-case basis. For example, a Quality Gate could mandate that all new code must include at least 80% test coverage, or that there should be no diagnosed security issues.

The combination of Quality Profiles and Quality Gates allow you to define the high-level expectations of code quality within an organization. In general, more rules in profiles and more conditions in gates indicate a higher expectation of quality.

## Decision Making

SonarQube’s ability to produce several key metrics and offer a way to customize Quality Profiles and Quality Gates are essential assets for decision-making. Developers, tech leads, and managers can all benefit from such assets when it comes to making both technical and product related decisions. For example, a high visibility application with some technical debt can be rewarded with a sprint dedicated to refactoring to reduce the debt.

# SonarQube Configuration

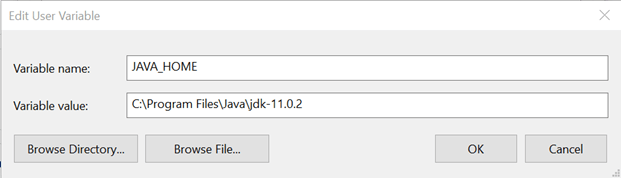
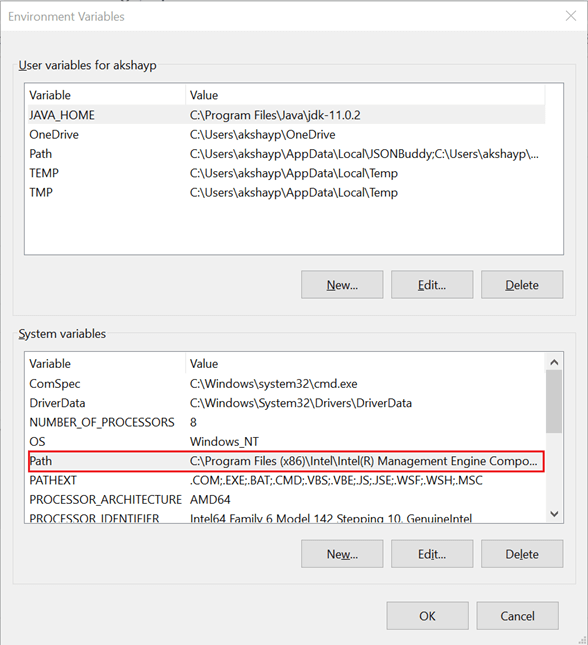
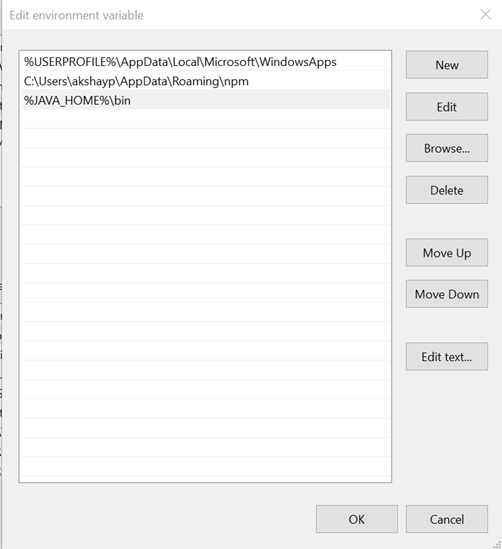
**Environment** : On-Prem

SonarQube is meant to be integrated with on-premise solutions.

## Java SE Requirement

In order to run SonarQube, we need to install JAVA in our local system.

* Steps followed for configuration of the 'PATH' system variable under environment variables:

1. Go to Control Panel > System > Advanced System Settings, it will open the System Properties window.
2. Click on the "Environment Variables" button.
3. Click on the "View" button under User Variables.
4. Give the variable name as 'JAVA\_HOME'.
5. The variable value will be your JDK path where you installed JAVA.  
   
6. Select path variable under system variable and click on the "Edit" button.  
   
7. Add a new path as mentioned below and click the "OK" button.  
   

### SonarQube Server Installation

SonarQube files are used from: https://www.sonarqube.org/downloads/

Extracted files of the SonarQube are used for the configuration of the Sonar Server.

On the extracted path of the SonarQube file we run StartSonar.bat using CMD in Jump Server and the SonarQube server is started locally.

A picture containing background pattern

Description automatically generated

In order to verify the same, we browse http://localhost:9000 in the browser and you able to see the landing page of SonarQube.  
  
Graphical user interface, text, application

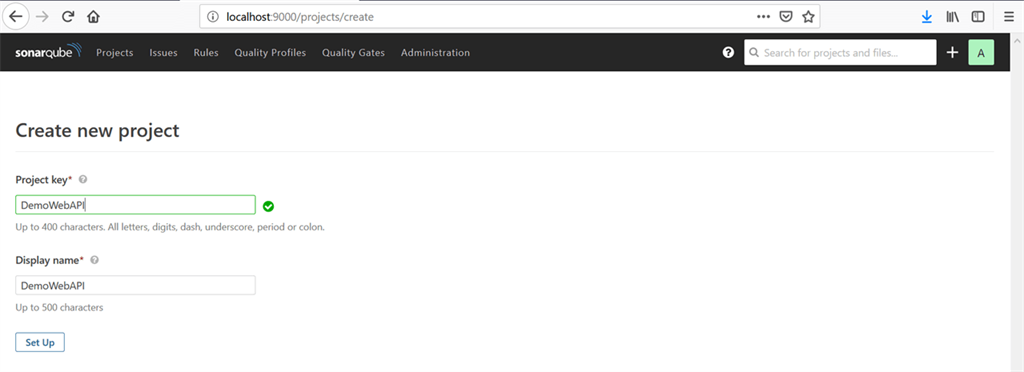
Description automatically generated

# Running SonarQube Project

This sections provides the steps and methods for running the SonarQube project in localhost to analyze the code smells in the given repository.

## SonarQube Project Creation

Creation of the project is done through the SonarQube dashboard providing the project key and display name.

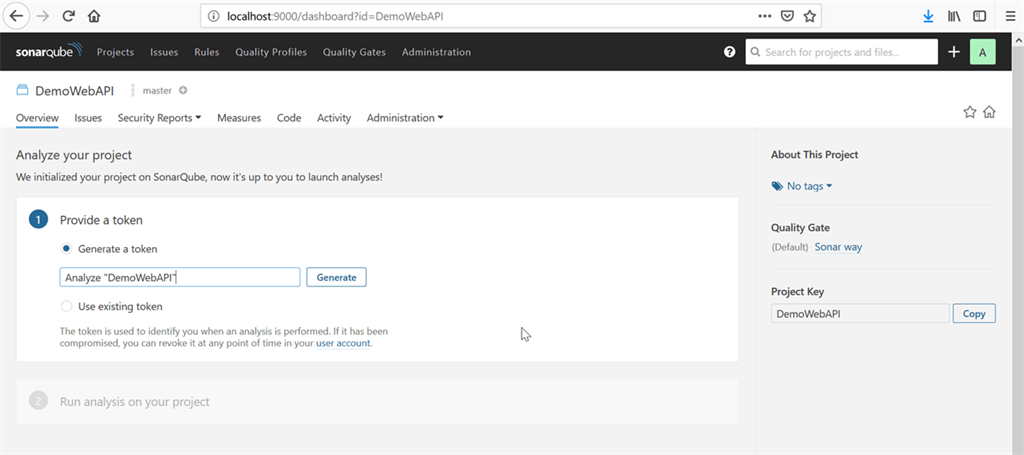


## SonarQube Project Evaluation

We have our repository cloned directly in the Jump Server within Visual Studio which saves the entire solution files inside the source folder of user profile.

We use the downloaded scanner for MSBuild for .Net 5.0 to evaluate the code using the Command Prompt.

### SonarQube Project Token

Once the project is set up, a token is generated as shown on the below example.

### SonarQube Project Scanner

On the further steps in SonarQube project, we have .Net selection for project analysis from where we need to download and extract the scanner for MSBuild.

Graphical user interface, text, application, email

Description automatically generated

### Executing the Scanner

Running a SonarQube analysis is straighforward. We execute the following commands at the root of the solution.

1. dotnet sonarscanner begin /k:"Test2" /d:sonar.host.url="http://localhost:9000" /d:sonar.login="88313c03bb77946e73452e7dfb516a39a954adab"
2. dotnet build
3. dotnet sonarscanner end /d:sonar.login="88313c03bb77946e73452e7dfb516a39a954adab"   
   (SonarQube project token)

Once the above command are ran successfully using the CMD, we get Build Successful message on the CMD and the dashboard page reloads automatically.

Text

Description automatically generated

# SonarQube Dashboard Results

After the build is successful with the SonarScan and MSBuild, we finally get the results in the Sonar Dashboard which shows all the details as shown below:

Graphical user interface, application, chat or text message

Description automatically generated