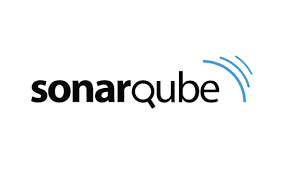
Sonar

Comandos y tips



Simple Code | Javier

Contenido

[Introducción 2](#_Toc84014609)

[Requisitos Instalación Local 2](#_Toc84014610)

[Configure la variable del sistema 'PATH' en variables de entorno 2](#_Toc84014611)

[Instalación del servidor SonarQube 2](#_Toc84014612)

[Configuración y conexión de Sonar con SqlServer 2](#_Toc84014613)

[Instalación con Docker 5](#_Toc84014614)

[Customizar la imagen de sonar con Docker 5](#_Toc84014615)

[Interpretación de las Métricas 6](#_Toc84014616)

[Issues, tipos y severidades 6](#_Toc84014617)

[Reglas de codificación 6](#_Toc84014618)

[Reglas de Calidad 6](#_Toc84014619)

[Duplicación de código 7](#_Toc84014620)

[Complejidad ciclo matica 7](#_Toc84014621)

[Resumen de la sección 7](#_Toc84014622)

[Inspección y Conceptos de calidad 7](#_Toc84014623)

[Deuda Técnica 7](#_Toc84014624)

[SQALE (Software Quality Assessment based on Lifecycle Expectations) 8](#_Toc84014625)

[Deuda Técnica y SQALE Rating 8](#_Toc84014626)

[Compuerta de Calidad 10](#_Toc84014627)

[Análisis de calidad para Angular 10](#_Toc84014628)

[Análisis de calidad con Node 10](#_Toc84014629)

[Comandos Para Net Core 10](#_Toc84014630)

[Comandos para la ejecución del proyecto con sonar: 10](#_Toc84014631)

[Test Api Net Core 11](#_Toc84014632)

[Referencias 13](#_Toc84014633)

# Introducción

Cuando se trabaja en equipo, y están distintos desarrolladores en el mismo proyecto, es bueno tener una revisión del código. SonarQube es una herramienta a través de la cual podemos evaluar nuestro código.

Los tipos de evaluaciones son basadas a reglas de calidad que por defecto contienen ciertos valores y que se pueden modificar o ampliar con plugins (pagos o gratuitos).

Sonar soporta distintos lenguajes, por ello es posible concretar el análisis sobre distintos tipos de tecnologías.

# Requisitos Instalación Local

Para ejecutar SonarQube, necesitamos instalar JAVA 11+ en nuestro sistema local. Consulte el enlace a continuación para descargar el instalador de JAVA e instalar JAVA. <https://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

## Configure la variable del sistema 'PATH' en variables de entorno

Vaya a Panel de control> Sistema> Configuración avanzada del sistema, se abrirá la ventana Propiedades del sistema.

Haga clic en el botón "Variables de entorno".

Haga clic en el botón "Ver" en Variables de usuario.

Dé el nombre de la variable como 'JAVA\_HOME'.

El valor de la variable será su ruta JDK donde instaló JAVA.

# Instalación del servidor SonarQube

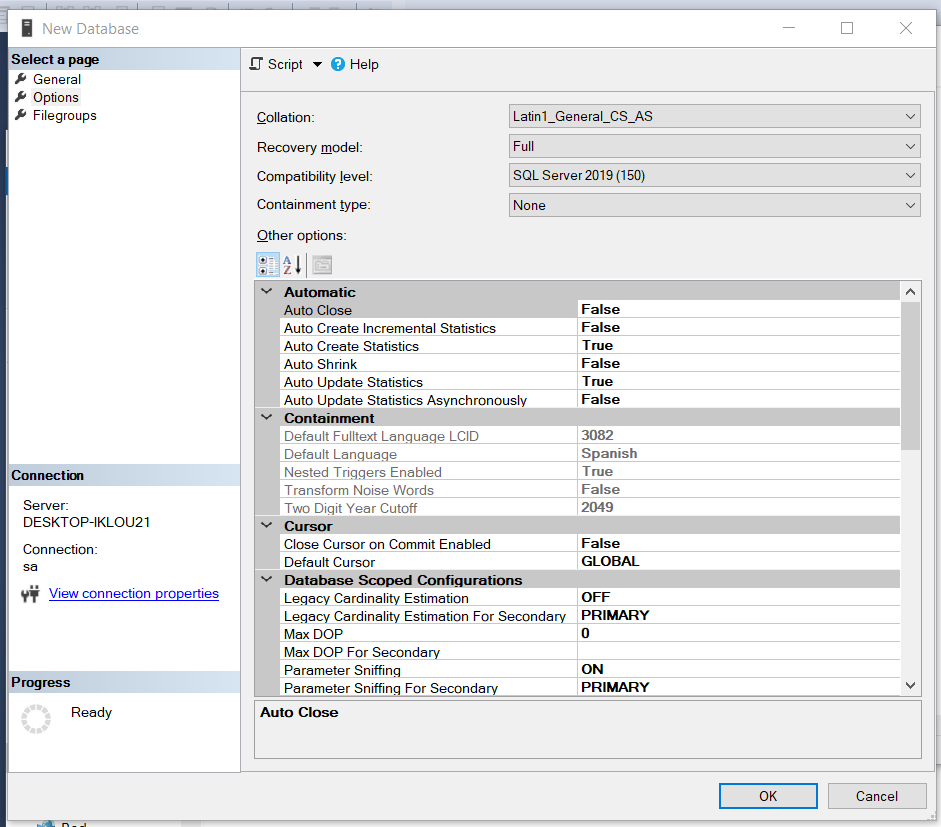
Descarga SonarQube desde <https://www.sonarqube.org/downloads/>

Extraerlo en una de tus unidades locales como D: \ sonarqube-7.6.

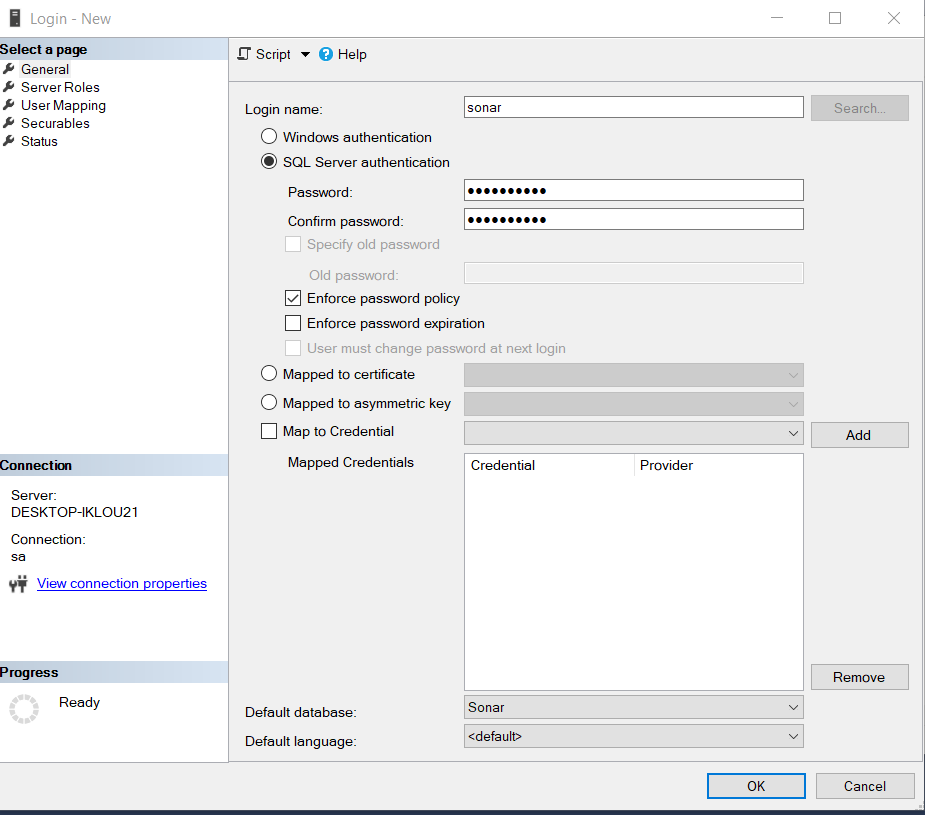
Ejecute el símbolo del sistema como administrador. Vaya a la ruta extraída y ejecute StartSonar.bat.

## Configuración y conexión de Sonar con SqlServer

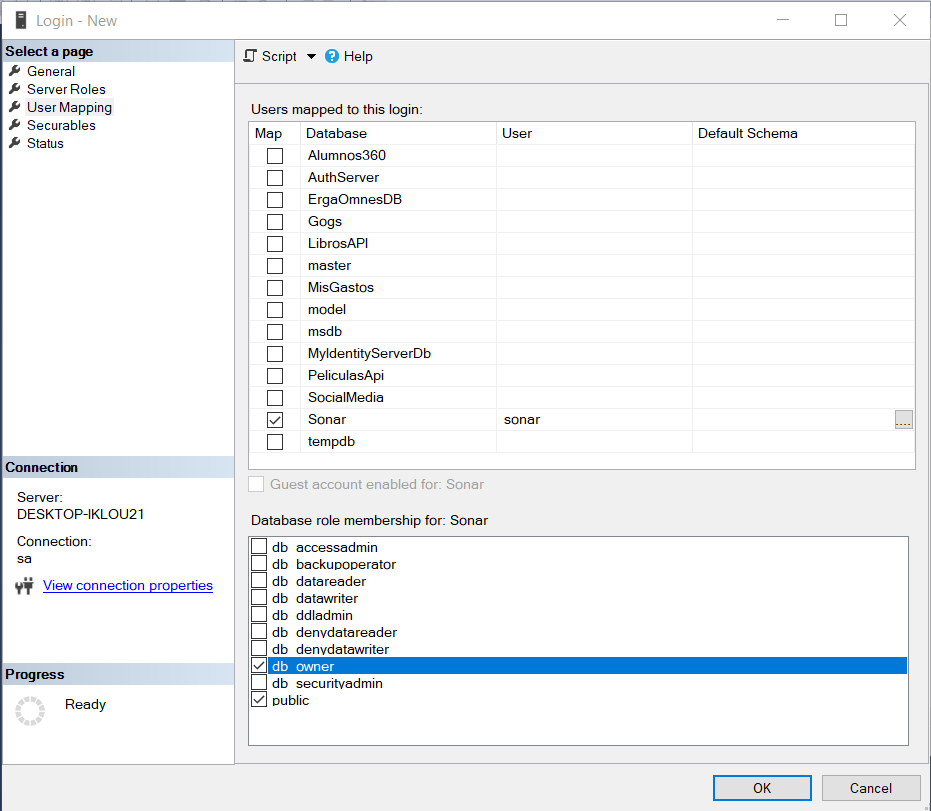
1. Crear una base de datos con el mismo nombre mencionado en el archivo “sonar.properties” de la carpeta conf., en mi caso Sonar.
2. Una vez creada la base de datos, ir a Options y cambiar la Collation: default a 🡪 Collation Latin1\_General\_CS\_AS, esto hace que SqlServer / Case-Sesitive / Accent-Sensitive



1. Crear un usuario para el login con la BD de sonar



1. Además, hay que indicar a sql que el nuevo usuario sonar, va a ser el dueño de la BD



1. Ahora hay que ubicarse en la carpeta: sonarqube-xx\conf

Abrir el archivo sonar.properties, con un editor de texto de su preferencia, para descomentar las líneas respectivas a la

1. Ejecutar desde la cmd en la ruta: C:\Windows\System32\cmd.exe, el siguiente comando: .\StartSonat.bat

Si surgen problemas de ejecución y en el final de línea: **wrapper | <-- Wrapper Stopped**

El motivo podría ser el port numberde sonarQube O el de la instancia de elasticSearch utilizada por sonarQube (tuve un problema similar antes), por lo que el paso para cambiar ambos / uno de esos puertos es:

Vaya a esta carpeta: sonarqube-xx \ conf

Abra este archivo: sonar.properties

Para el puerto de sonarQube:

Buscar: # sonar.web.port

Cambie el valor de 9000 a otro puerto, como 9123; y anule el comentario de la línea (elimine # al principio)sonar.web.port=9123

Para el puerto de instancia de elasticSearch de sonarQube:

Buscar: # sonar.search.port

cambie esta línea a sonar.search.port=0 (esto significa que buscará cualquier puerto disponible y lo vinculará)

Guarda tus cambios

Vuelve a empezar tu sonarqube

Acceso al servidor con el nuevo puerto sonarQube especificado: http: // localhost: 9123

1. Para instalar sonar como un servicio hay que ejecutar como administrador el archivo bat:

<https://docs.sonarqube.org/latest/setup/install-server/>

# Instalación con Docker

docker run -d --name sonarqube --name sonarqube -e SONAR\_ES\_BOOTSTRAP\_CHECKS\_DISABLE=true

-p 9000:9000 sonarqube:8.5.1-community

Banderas de Docker utilizadas:

-d: ditache, para no bloquear la consola con el contenedor

-name: ponerle un nombre al contenedor

-p 9000:9000 : es para mapear el puerto del host con el del contenedor, el puerto del lado izquierdo

es el de la compu local, el de la derecha el del contenedor

-t: es un tag

docker exec -t <id del contenedor> bash: para ver el contenido de la imagen en el contenedor

pwd: para ver la ruta del contenedor

docker images: lista las imagenes locales que tenemos

## Customizar la imagen de sonar con Docker

1. Tener un archivo sonar.properties donde se setearan las configuraciones que se requiera personalizar, por ejemplo las conexiones a la BD de sonar, el puerto que se quiere exponer el sonar web, etc.
2. Crear un archivo docker file con los siguientes pasos:

FROM sonarqube:8.5.1-community

COPY sonar.properties /opt/sonarqube/conf/

EXPOSE 9090

1. Para construir la image: docker image build -t sonarqube-custom .

# Interpretación de las Métricas

## Issues, tipos y severidades

En el apartado de los issues se exponen todos los problemas encontrados, para ello se dividen en los siguientes aparatados:

Tipos

1. **Bug:** Son problemas que pueden hacer que software eventualmente falle
2. **Vulnerability:** Problemas de seguridad donde el software está expuesto a ataques
3. **Code Smell:** código difícil de mantener, poco claro y fuera de los standars de programación

Nota: en todos los casos sonar nos señala cual es el problema y también brinda información de cómo solucionarlo.

**Severity**

Las severidades de los issues son importantes porque nos permiten plantear estrategias de como atacar los problemas a ajustar. Las severidades se dividen por 5 tipos:

1. Blocker: Son problemas con alta probabilidad de impactar en un software en producción. Ejemplo contraseña hardcord.
2. Critical: Tiene pocas probabilidades de impactar en un software en producción, por ejemplo una función vacia.
3. Mijor: Tiene un impacto alto en la productividad del desarrollador, ejemplo una lógica compleja
4. Minor: Tiene un impacto bajo en la productividad del desarrollador, ejemplo corregir ciertos tags html.
5. Info: No son bugs, no son defectos de calidad, son hallazgos de calidad, por ejemplo ajustes de estilos css.

**Security HotSpots**

En el apartado de Security HotSpots, son secciones de código con problemas de seguridad que se requiere una evaluación más detallada para ver si son falsos positivos o si realmente hay correcciones a realizar.

## Reglas de codificación

En SonarQube una regla de codificación es un conjunto de patrones que definen el estilo y la consistencia del código dado un determinado lenguaje, por ejemplo, en c# se usa stylecop y en java se usa checkstyle. La ventaja de usa sonar, todo el equipo tendrá la misma medición de calidad con respecto al lenguaje que se esté trabajando.

## Reglas de Calidad

## Duplicación de código

Sonar es capaz de detectar duplicación de código en tres niveles:

1. duplicación en el mismo archivo
2. duplicación entre archivos del mismo proyecto
3. duplicación entre proyectos

También hay un calculo de la densidad del código en porcentaje, lo ideal sería tener un porcentaje lo mas cercano al cero posible.

## Complejidad ciclomatica

La complejidad ciclomatica es medida en números, entonces nos podemos preguntar: ¿Cuál es el número que deberíamos considerar como aceptable?

Para ello existen 4 rangos:

1-10: Sin riesgo

11-20: Moderado

21-50: Alto

+50: Reactorizar, ya no se considera ni testeable, es mejor refactorizar.

## Resumen de la sección

LOC (Lines Of Code) es una métrica importante porque es la base sobre la cual se calculan otras métricas como por ejemplo proporción de coberturas de pruebas.

Los **issues** se clasifican por su tipo y severidad.

En SonarQube una regla de codificación es un conjunto de patrones que definen estilo y la consistencia que un código fuente debería tener en determinado lenguaje.

Sobre las métricas de pruebas se puede decir que como desarrolladores debemos considerar el tiempo y el esfuerzo necesarios para construir la prueba versus agregado en densidad de cobertura que esta va a ofrecer.

El método de refactorización que NO aporta valor a la hora de reducir el código duplicado es el reemplazar condicional con polimorfismo.

# Inspección y Conceptos de calidad

## Deuda Técnica

Sonar realiza un análisis de lo que considera deuda técnica y expone para cada archivo una estimación de lo que llevara el ajuste.

Pero si miramos la metrica de “Debt ratio”, es un porcentaje entre la deuda técnica total acumulada vs el total de deuda técnica posible (el peor escenario). Entonces puede suceder que un archivo que tiene una estimación mayor en tiempo tenga un porcentaje bajo por su baja calificación.

## SQALE (Software Quality Assessment based on Lifecycle Expectations)

Sonar asigna una nota al ratio de la deuda técnica, la cual se conoce como sqale, para obtener una medición: rápida, breve y visual de cual es la calidad del código del proyecto basados en atributos no funcionales: la mantenibilidad, la confiabilidad y la seguridad. La deuda técnica es un atributo de la mantenibilidad.

## Deuda Técnica y SQALE Rating

Empiezo por el concepto de «Deuda Técnica» (Technical Debit). No quisiera extenderme mucho más de lo necesario y entrar en debates de qué se puede considerar o no «Deuda Técnica». El concepto de «Deuda Técnica» se basa

en una metáfora expuesta por Ward Cunningham con la que se transmite en términos financieros la necesidad refactorizar para mantenerse en unos niveles de calidad técnica aceptables. Se ha incurrido previamente en una deuda por

la que actualmente se están pagando intereses; es decir, la refactorización vendría a ser un modo de amortización de la deuda. Un buen resumen para entender el concepto de «Deuda Técnica» lo tenemos en

este artículo de Martin Fowler; y si seguimos un punto de vista estricto se puede considerar que un mál código no estaría considerado como «Deuda Técnica»,

es simplemente un mal código, este razonamiento lo podemos encontrar en este post del blog de Robert C. Martin.

En el siguiente vídeo tenemos al propio autor de la metáfora, Ward Cunningham, hablando sobre este tema:

Y en este artículo hay una reflexión, comentarios, debate, y más analogías sobre qué es la «Deuda Técnica».

Esto esta muy bien, pero ¿Cómo Sonarqube distingue lo que sería «Deuda Técnica» de lo que simplemente es mal código? Pues efectivamente, no puede. Si nos fijamos, los valores de la métrica de «Deuda Técnica» están expresados en unidades de tiempo. Así que la

interpretación de lo que Sonarqube considera como «Deuda Técnica» es el tiempo que se tendría que invertir para corregir esa carencia, ya sea porque es un mal código o porque es una deuda técnica asumida previamente.

A efectos prácticos, nos ayuda a entender que cuanta menor «Deuda Técnica» tenga un proyecto, podemos considerar que está en un mejor estado de salud. Y al tener asignado un peso temporal, puede servir para hacerse una idea del

por qué se tarda tanto en incluir ciertos cambios y si es necesario asignar recursos para amortizar la deuda, refactorizar, o incluso si podemos considerar la «banca rota» del proyecto, no seguir invirtiendo en nuevas.

funcionalidades y como responsables pedir un presupuesto para que éstas sirvan como embrión de otro proyecto en el que la inversión de su desarrollo no implique seguir acumulando deuda.

Ahora voy a ver qué es eso de SQALE Rating. Bien, pues lo primero es ¿de dónde viene el nombre «SQALE»? su origen es Software Quality Assessment based on Lifecycle Expectations y que viene de un método

con el mismo nombre que tiene por objetivo asegurar la calidad del código fuente. Podemos ver en qué consiste con más detalle en www.sqale.org.

Si vemos un poco el detalle de cómo se calcula, y cómo Sonarqube nos lo muestra, vemos que está íntimamente relacionado con la «Deuda Técnica», más concretamente con el ratio de la «Deuda Técnica» (documentación en Sonarqube).

Buscándole un sentido práctico, sirve para transmitir de forma rápida, breve y visual la calidad del proyecto al responsable dándole una categoría entre la A y la E siguiendo la siguiente escala:

A: Ratio de «Deuda Técnica» menor al 10%. Podemos considerar que el proyecto está sano y no habría que hacer nada especial.

B: Ratio de «Deuda Técnica» entre el 10% y el 20%. Se puede considerar que está en unos parámetros aceptables, pero hay que seguir de checkeo periódico para vigilar que no empeora. Se puede empezar a plantearse tomar medidas preventivas

C: Ratio de «Deuda Técnica» entre el 21% y el 50%. Entramos en unos valores en los que la salud ya no es buena, hay que empezar a tomar medidas correctivas. Si bien el carácter de urgencia puede venir marcado por la desviación de los datos respecto a los valores aceptables.

D: Ratio de «Deuda Técnica» entre el 51% y el 100%. Es urgente tomar medidas que lleven de vuelta el proyecto a zonas más saludables.

E: Ratio de «Deuda Técnica» superior al 100%. Se puede decir que hemos entrado en la zona de rescate, hay que tomar una decisión entre apostar por un rescate, con todos los esfuerzos que esto significa de tiempo y dinero, o bien dejar que el proyecto entre en banca rota para buscar otro camino fuera del mismo.

Esta categorización del SQALE Rating por letras y colores, es muy similar a la que tenemos en la certificación energética tan de moda hoy en día. Donde el color aporta

la información de lo bien construido que está el proyecto y el tamaño de la barra el esfuerzo que habría que hacer para llevarlo a una zona más confortable.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Así que ya sabes que gran parte del peso de los criterios de calidad de un proyecto para saber si está en un buen estado de salud o no recaen en el «Ratio de Deuda Técnica».

Pero ¿cómo se calcula realmente esta métrica? Bueno no es algo sencillo explicar, aunque se puede resumir en varios cálculos que comparan el esfuerzo de

corregir las carencias detectadas frente a volver a construirlo desde cero. Para más información sobre cómo se calculan estos valores se pueden consultar estos enlaces:

SQALE – Technical Debt Ratio,

Tenchincal Debt.

# Compuerta de Calidad

Sonar nos asiste en analizar el código basado a criterios de aceptación no funcionales, los de calidad.

La compuerta de calidad en sonar es un conjunto de condiciones booleanas definidas previamente, para aprobar o no si el proyecto esta listo para el despliegue a producción. No todas las compuertas de calidad aplican a todos los proyectos.

Se puede crear compuertas de calidad personalizadas para cada proyecto.

# Análisis de calidad para Angular

## Configuraciones para integrar Sonar con Angular

**Paso 1:** ejecuta Sonarqube localmente

La forma más fácil y rápida de hacer que Sonarqube funcione localmente es ejecutarlo en un contenedor docker, mediante la siguiente secuencia de comandos:

docker run -d --name sonarqube -p 9000:9000 sonarqube:latest

Una vez que el contenedor esté en funcionamiento, deberíamos poder acceder a sonarqube con la siguiente URL e iniciar sesión con las credenciales predeterminadas de administrador/administrador.

http://localhost:9000

Nota: No se apresure a acceder a la URL, descubra que no funciona y elimine el contenedor de la ventana acoplable. Por lo general, toma unos segundos poner en funcionamiento a Sonarqube.

Hay dos formas diferentes de adjuntar un proyecto Angular a la instancia de sonda.

Conectándolo desde la consola del sonar.

Ejecutar el escáner de sonda desde el proyecto a escanear.

Aquí me ocuparé del segundo enfoque.

**Paso 2:** Configurar el proyecto Angular

Hay escáneres específicos para diferentes herramientas de compilación, pero para la aplicación basada en Angular (Typescript) debemos usar el paquete base sonar-scanner npm. Vamos a agregarlo a nuestra aplicación Angular.

npm i sonar-scanner --save-dev

Luego, debemos agregar el archivo de propiedades (sonar-project.properties) mencionado a continuación en la raíz de la aplicación. Esto define la instancia de la sonda, la ruta del archivo de origen, las extensiones del archivo de prueba y los archivos de informe. La explicación de todas las propiedades posibles se puede encontrar en este enlace .

**sonar-project.properties:**

|  |  |
| --- | --- |
| sonar.login=admin | |
| sonar.password=admin |  | |
| sonar.projectKey=my-angular-app |  | |
| sonar.sourceEncoding=UTF-8 |  | |
| sonar.sources=src |  | |
| sonar.tests=src |  | |
| sonar.exclusions=\*\*/node\_modules/\*\* |  | |
| sonar.test.inclusions=\*\*/\*.spec.ts |  | |
| sonar.typescript.lcov.reportPaths=coverage/lcov.info |  | |
| sonar.testExecutionReportPaths=reports/ut\_report.xml |  | |
|  |  | |

Si bien la mayoría de las propiedades son obvias, se agregarán algunos detalles para algunas de ellas.

sonar.source: Esto se refiere a la ruta donde residen nuestros archivos fuente.

sonar.tests: Esto se refiere a la ruta donde residen nuestros archivos de prueba. Esto ayudará a escanear los informes de ejecución.

sonar.test.inclusions: Esto se refiere al patrón de extensión de archivo para los archivos de prueba y asegura que nuestros archivos de prueba estén incluidos para el análisis. Este [enlace](https://docs.sonarqube.org/latest/project-administration/narrowing-the-focus/) de documentación de la sonda tiene detalles adicionales sobre cómo seleccionar los archivos que se incluirán y excluirán para el escaneo.

sonar.typescript.lcov.reportPaths: Esto se refiere al archivo lcov.info (informe de cobertura de código) creado por complementos de karma de terceros. Asegúrese de que las siguientes propiedades en karma.conf.js estén configuradas correctamente para que el informe de cobertura se cree bajo la raíz de la aplicación angular.

**dir**: require(**'path'**).join(\_\_dirname, **'coverage'**),

Nota: El valor predeterminado será ../coverage, que creará el informe fuera de la carpeta raíz de la aplicación Angular.

sonar.testExecutionReportPaths: Esto se refiere al archivo de informe de ejecución de prueba creado nuevamente por complementos de karma de terceros. Sonarqube no tiene soporte directo para escanear el informe de ejecución de la prueba, y esto se puede lograr mediante la biblioteca npm de código abierto [karma-sonarqube-unit-reporter](https://github.com/tornaia/karma-sonarqube-unit-reporter) .

Los siguientes pasos rápidos agregarán este reportero a nuestra aplicación. Comencemos agregando la biblioteca npm a nuestra aplicación.

npm i karma-sonarqube-unit-reporter --save-dev

Y continúe haciendo las siguientes adiciones en karma.conf.js para agregar este reportero.

|  |
| --- |
| plugins: [ |
| require('karma-sonarqube-unit-reporter') |  |
| ], |  |
| sonarQubeUnitReporter: { |  |
| sonarQubeVersion: 'LATEST', |  |
| outputFile: 'reports/ut\_report.xml', |  |
| overrideTestDescription: true, |  |
| testPaths: ['./src'], |  |
| testFilePattern: '.spec.ts', |  |
| useBrowserName: false |  |
| }, |  |
| reporters: ['sonarqubeUnit'], |  |

Y el paso final para configurar el proyecto Angular, agregue el escáner de sonda a los scripts en package.json.

|  |  |
| --- | --- |
| "scripts": { | |
| "sonar":"sonar-scanner" |  |
| } |  |

**Paso 3:** Escanea la aplicación

Dado que el escáner de sonda depende de los informes de cobertura y ejecución generados por los complementos de karma de terceros, creémoslos primero ejecutando los comandos angular-cli.

ng test --code-coverage --watch=false

Asegúrese de generar los archivos de informe, en ./coverage y ./reports.

Ahora vamos a ejecutar el escáner,

npm run sonar

El escáner realiza las siguientes acciones visibles junto con otras listas de acciones entre bastidores,

Crea un proyecto correspondiente a la aplicación escaneada en la instancia de sonarqube que se ejecuta en localhost:9000.

Escanea la aplicación y crea informes bajo el nombre del proyecto mencionado en la clave del proyecto (sonar-project.properties).

Escanea los informes de cobertura y ejecución y crea referencias para ellos en la consola de sonda.

Eso completa la configuración y ahora actualiza la consola de sonarqube para ver las actualizaciones,

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Fig. 2: Panel de control de Sonarqube para el proyecto my-angular-app

Conclusión

Me propuse escribir este artículo porque no pude encontrar una cuenta clara y sucinta que explicara los pasos necesarios a seguir para este proceso. Espero que este artículo te sea útil. ¡Gracias por leer y déjame saber tu opinión en los comentarios!

Referencia: <https://codeburst.io/setup-sonarqube-for-angular-application-locally-in-three-easy-steps-8f31e339ac19>

# Análisis de calidad con Node

# Comandos Para Net Core

Primero hay que instalar una herramienta nativa para proyectos net core mediante el siguiente comando:

**dotnet tool install --global dotnet-sonarscanner**

## Comandos para la ejecución del proyecto con sonar:

1.

dotnet C:\sonar-scanner-msbuild\SonarScanner.MSBuild.dll begin /k:"dotnet5.0-bookStore-api" /d:sonar.host.url="http://localhost:9123" /d:sonar.login="a86a76670b2cff50a3c2081ae30c6cf72b6a43ba"

2.

dotnet build D:\UADE-GOGS\dotNet5.0-bookStore-api\BookStore.sln

3.

dotnet C:\sonar-scanner-msbuild\SonarScanner.MSBuild.dll end /d:sonar.login="a86a76670b2cff50a3c2081ae30c6cf72b6a43ba"

Nota: El nombre del proyecto no debe contener espacios porque puede generar error en el comando “dotnet build”

Otro ejemplo para ejecutar desde el CLI de dotnet para net core:

1. dotnet sonarscanner begin /d:sonar.login=admin /d:sonar.password=admin /k:"dotnet-math-project" /d:sonar.host.url=http://localhost:9123
2. dotnet build D:\UADE-GOGS\sonarqubeDemo-master\source\Math\Math.sln
3. dotnet sonarscanner end /d:sonar.login=admin /d:sonar.password=bitnami

Para generar el reporte de covertura

dotnet test BookStore.API.Tests/BookStore.API.Tests.csproj /p:CollectCoverage=true /p:CoverletOutputFormat=opencover

## Test Api Net Core

Informe de cobertura de código

Este informe se utiliza para verificar si hemos cubierto el código completo del sistema bajo prueba en casos unitarios o no. es decir, nuestros casos unitarios pueden probar toda la funcionalidad proporcionada en el sistema bajo prueba (en este caso MathsService) o no.

El informe de cobertura de código está disponible directamente en Test Explorer en la edición Visual Studio Enterprise, pero como estoy usando la herramienta de cobertura de código de la edición de Visual Studio Community no forma parte de Test Explorer. Tendremos que generar un informe de cobertura de código usando la herramienta de línea de comandos y podemos ver el mismo en el navegador, ya que generaremos un informe basado en html.

Para ejecutar todos los comandos mencionados a continuación, abra la herramienta de línea de comandos de Visual Studio y cambie el directorio actual a la ruta del Proyecto de prueba xUnit y luego ejecute todos los comandos.

Para poder usar el informe de cobertura de código, necesitamos instalar la herramienta reportgenerator usando el comando como se muestra a continuación

**dotnet tool install -g dotnet-reportgenerator-globaltool**

Para capturar datos para el informe de cobertura de código, debemos ejecutar el siguiente comando desde la herramienta de línea de comandos. Este comando generará un archivo XML llamado cobertura.cobertura.xml en la ruta TestResults \ guid \.

**dotnet test --collect:"XPlat Code Coverage"**

Después de que se haya generado el archivo XML anterior, ejecute el siguiente comando para generar un informe de cobertura de código, en este comando debe especificar la ruta a la cobertura.cobertura.xml y también especificar el tipo de informe que se ha establecido en Html en el siguiente comando .

**reportgenerator "-reports:path\to\TestResults\guid\coverage.cobertura.xml" "-targetdir:coveragereport" -reporttypes:Html**

**reportgenerator "-reports:D:\UADE-GOGS\dotNet5.0-bookStore-api\tests\BookStore.Infrastructure.Tests\TestResults\950da2bf-260c-4468-8646-b985fc7a0983\coverage.cobertura.xml" "-targetdir:coveragereport" -reporttypes:Html**

Después de ejecutar todos los comandos anteriores, generará un informe en la carpeta de informe de cobertura, ya que hemos especificado este nombre de carpeta en el argumento targetdir. Ahora abra el archivo index.html generado en esa carpeta en cualquier navegador, debería poder ver el informe de cobertura del código como se muestra a continuación

Informe de cobertura de código

Podemos ver en el informe anterior que la cobertura de código es del 0% para todos los tipos distintos de MathsService, lo cual es obvio, ya que no hemos escrito ninguna prueba unitaria para estos tipos. Pero el problema es que, aunque hemos escrito una prueba unitaria para todas las funciones en el tipo MathsService, la cobertura del código para MathsService es del 86,6%, es decir, no es del 100%. Necesitamos profundizar y comprobar por qué la cobertura del código no es del 100% para MathsService.

Después de hacer clic en el tipo MathService en el informe, obtenemos los detalles a continuación

Detalles de cobertura de código

En la pantalla de arriba podemos ver que la cobertura del código para todas las funciones es del 100% excepto la función Dividir, cuya cobertura de sucursales es del 50% y la cobertura de la línea es del 66,6,6%. También desde los detalles de cobertura de línea podemos ver que la función Dividir arroja una excepción en caso de que x2 sea 0 y esta funcionalidad no se haya cubierto en pruebas unitarias debido a que la cobertura de código no es del 100%

Así que ahora vamos a escribir una prueba unitaria para esta excepción y luego volver a generar el informe de cobertura de código para verificar si la cobertura de código para MathsService es del 100% o no.

Agreguemos la siguiente función con el atributo [Fact] en la clase de prueba unitaria MathServiceTests que probará la excepción dividir por cero que lanza la función Dividir en caso de que el valor de x2 sea 0

# Referencias

<https://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

<https://www.c-sharpcorner.com/article/sonarqube-configuration-for-net-core-web-api/>

<http://oscarmoreno.com/quality-gates-sonarqube/>

Instalaciones y docu de Docker

<https://docs.docker.com/engine/install/ubuntu/>

<https://docs.sonarqube.org/latest/requirements/requirements/>

<https://hub.docker.com/_/sonarqube>

(repo para instalar sonar con kubernetes): <https://github.com/helm/charts/tree/master/stable/sonarqube>

Repos con ejemplos Net core con test y algunos con las configs para sonar

<https://github.com/sandeepananthaTS/sonarqubeDemo>