Gremio Calidad

Manual SonarQube

Contenido

1. Introducción

En este documento se describen los pasos a seguir para ejecutar el análisis de SonarQube para los diferentes lenguajes de programación soportados en la herramienta.
La herramienta la podrá ocupar cualquier empleado que tenga un usuario y contraseña para poder ingresar y ver los proyectos que tenga dados de alta, poniendo en práctica las buenas prácticas de programación.
Es importante para las personas que nunca han trabajado con esta herramienta ya que se explica cómo se integra desde cero en los proyectos que sean necesarios.

2. Objetivo

Incentivar a los desarrolladores el uso e implementación de SonarQube, para el aseguramiento de la calidad de software, adoptando buenas prácticas de desarrollo desde el inicio de ciclo del software, basándose en los estándares convencionales de programación establecidos para el grupo, en distintos lenguajes de programación utilizados actualmente.

2.1. ¿Qué es SonarQube?

Es una herramienta que permite realizar análisis estático de código fuente de manera automática, buscando patrones con errores, malas prácticas o incidentes. Además, realiza un cálculo de la deuda técnica y cobertura de pruebas unitarias.

2.1.1. Beneficios de implementar SonarQube

Nos permite verificar dentro del código:

- Detección de malas prácticas que afectan el performance y mal funcionamiento de la aplicación en producción.
- Vulnerabilidades conocidas de seguridad.
- Adecuación a estándares y convenciones de código establecidas en el grupo.
- Detección de código duplicado.
- Falta de pruebas unitarias, falta de comentarios.
- Código spaghetti, complejidad ciclomática, alto acoplamiento.

2.2. Categoría de Reglas

Hay cuatro tipos de problemas:

1. **bug**: Un punto de fallo real o potencial en su software.
2. **Vulnerabilidad**: un punto en su código que está abierto a ataques.
3. **CodeSmell**: un problema de mantenimiento que hace que su código sea confuso y difícil de mantener.

4. **Security Hotspot:** resalta un fragmento de código sensible a la seguridad que permite mejorar la solidez de una aplicación.

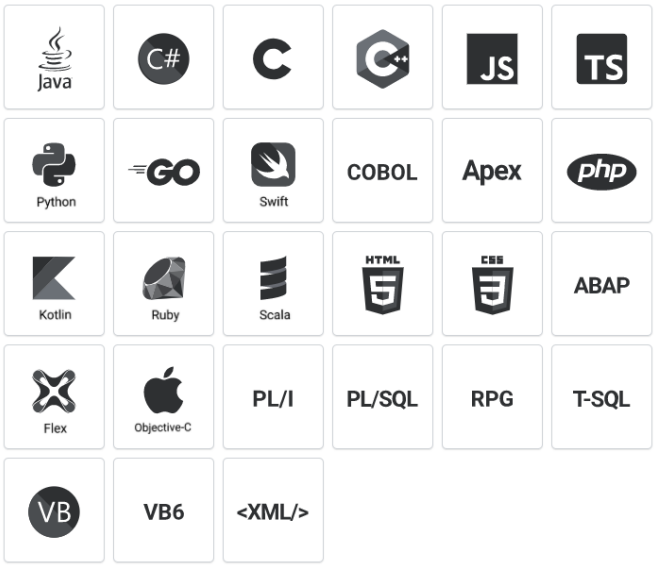
2.3. Gravedad de los Errores

Cada problema tiene una de cinco severidades:

1. **Blocker** **(Grave)**
   Es un error con una alta probabilidad de afectar el comportamiento de la aplicación en producción, puede afectar tanto el performance y la seguridad de la aplicación.
2. **Critical**
   O bien un error con una baja probabilidad de impactar el comportamiento de la aplicación en la producción o un problema que representa una falla de seguridad (Ej.: bloque catch vacío, inyección SQL, etc.).
3. **Major**
   Defecto de calidad que puede afectar la productividad del desarrollador. (Ej. bloques duplicados, parámetros no utilizados, etc).
4. **Minor**
   Defecto de calidad que puede afectar ligeramente la productividad del desarrollador.
5. **Info**
   Ni un error ni un defecto de calidad, solo un hallazgo.

2.4. Lenguajes de programación soportados

Sonar cuenta con más de 20 lenguajes de programación. Cada analizador proporciona numerosas reglas para detectar problemas de calidad generales y específicos del lenguaje.



2.5. Quality Profile

Son colecciones de reglas que se revisan durante el análisis. Para cada tecnología hay un quality Profile predeterminado, pero podemos tener varios para un mismo lenguaje.
En caso de que un proyecto analizado no tenga asignado explícitamente uno, se analizara con el perfil configurado como predeterminado para dicho lenguaje.

2.6. Quality Gate

Es el conjunto de condiciones que el proyecto debe de cumplir antes de que pueda ser lanzado a producción.

2.7. Formas de utilizar SonarQube

Existen 3 formas de realizar análisis de SonarQube:

1. **Local**, mediante el IDE con el que codifica el usuario (Soportado en Eclipse, Intellij, Visual Studio, Visual Studio Code).
2. **Línea de comando**, desde power Shell o cmd.
3. **Jenkins(otros),** se configura para lanzar en automático el análisis.

3. ¿Cómo se realiza el análisis de un proyecto en SonarQube?

El análisis se puede llevar a cabo mediante un SO **Windows** o **Linux** y existen dos formas de realizarlo:

3.1. Análisis por línea de ejecución

Se realiza el análisis por medio de una línea de ejecución, y así poder subir el resultado a los servidores de SonarQube, para eso se necesita:

1. Tener creado un proyecto en el Portal de SonarQube
   a) Validar la entrada al portal de SonarQube, ver Anexo 01 – Ingresar a la página SonarQube.
   b) Buscar y configurar el proyecto dentro de SonarQube, ver Anexo 02 – Configuración de proyecto en SonarQube.
2. Al ya haber realizado con anterioridad un análisis de algún proyecto, pero se extravió el Token, ver Anexo 03 – Recuperación de token SonarQube**.**

3.1.1. Ejemplo por línea de ejecución

Para realizar el análisis con la línea de ejecución es necesario:

1. Tener el token y la línea de ejecución que se obtuvo al configurar el proyecto en SonarQube, ver Anexo 02 – Configuración de proyecto en SonarQube.
2. Al tener la línea de ejecución es necesario realizar unas modificaciones como: (Ver Anexo 04 – Descripción de Línea de ejecución SonarQube).
   a. Cambiar los parámetros necesarios a la línea de ejecución.
3. Al tener listos los **pasos 1 y 2**, posteriormente seguir los pasos de Anexo 05 – Implementación de la línea de ejecución de SonarQube.

3.2. Análisis de un Plugin en IDE

Se instala un plugin en el IDE de un lenguaje de programación, en el cual se instará SonarLint que nos ayudará al análisis del código como se vaya agregando código. Este análisis no se sube al servidor de SonarQube para eso es necesario hacer un análisis por línea de comandos. (Ir a Análisis por línea de ejecución).

Los ejemplos de instalación de **PLUGIN** en un **IDE** son:
(Control + Clic derecho para ver los manuales/ Sobre los nombres).

- **Eclipse.**
- **Intelli J.**
- **Visual Studio (ASPX .NET, C#, C, C++).**
- **Visual Studio Code.**

4. Pruebas Unitarias

Las pruebas unitarias son una forma de comprobar que un fragmento de código funciona correctamente, teniendo con ellas varias ventajas como:

- Demuestran que la lógica del código está en buen estado y que funcionará en todos los casos.
- Aumentan la legibilidad del código y ayudan a los desarrolladores a entender el código base, lo que facilita hacer cambios más rápidamente.
- Los test unitarios bien realizados sirven como documentación del proyecto.
- Se realizan en pocos milisegundos, por lo que podrás realizar cientos de ellas en muy poco tiempo.

Las pruebas unitarias varían acorde a los frameworks utilizados para estas. A continuación, se describen las configuraciones a realizar según el framework utilizado:

4.1. Jest con React

Cuando se construye una aplicación con React, mediante el comando npx se agrega por defecto Jest, si no es así, se debe agregar de forma manual dicho framework para el testing.
Una vez agregado Jest al proyecto se debe modificar el archivo “package.json”, ubicado en la carpeta raíz del proyecto en cuestión
Las modificaciones se realizarán el apartado “scripts” del archivo antes mencionado:

1. **Sustituir la línea "test": "react-scripts test" por "test": "react-scripts test \"--watchAll=false\"". Esto es para que, al ejecutar esta instrucción se lleven a cabo todas las pruebas unitarias existentes en el proyecto.**

2. **Agregar "updateSnapShot": "react-scripts test \"-u\"". La cual será utilizada para actualizar los snapshots existentes.**

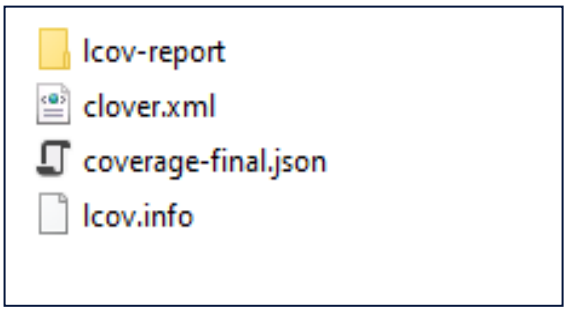
3. **Agregar la sentencia "coverage": "react-scripts test \"--coverage\" \"--watchAll=false\"". Esto ejecutará todas las pruebas existentes en el proyecto generando el archivo de cobertura de código (coverage) de dichas pruebas.**

Ejemplo:

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test \"--watchAll=false\"",
    "coverage": "react-scripts test \"--coverage\" \"--watchAll=false\"",
    "updateSnapShot": "react-scripts test \"-u\"",
    "eject": "react-scripts eject"
  }
```

Al ejecutar el script del punto 3, este creará el directorio “coverage” (en caso de no existir) en la carpeta raíz del proyecto. Este nuevo directorio se compone de los elementos siguientes:

- **Directorio "lcov-report":** El cual contiene el reporte de covertura en formato HTML (ejecutar index.html).
- **clover.xml:** Reporte de covertura en formato xml.
- **coverage-final.json:** Reporte de covertura en formato json.
- **lcov.info:** Reporte de covertura lcov.



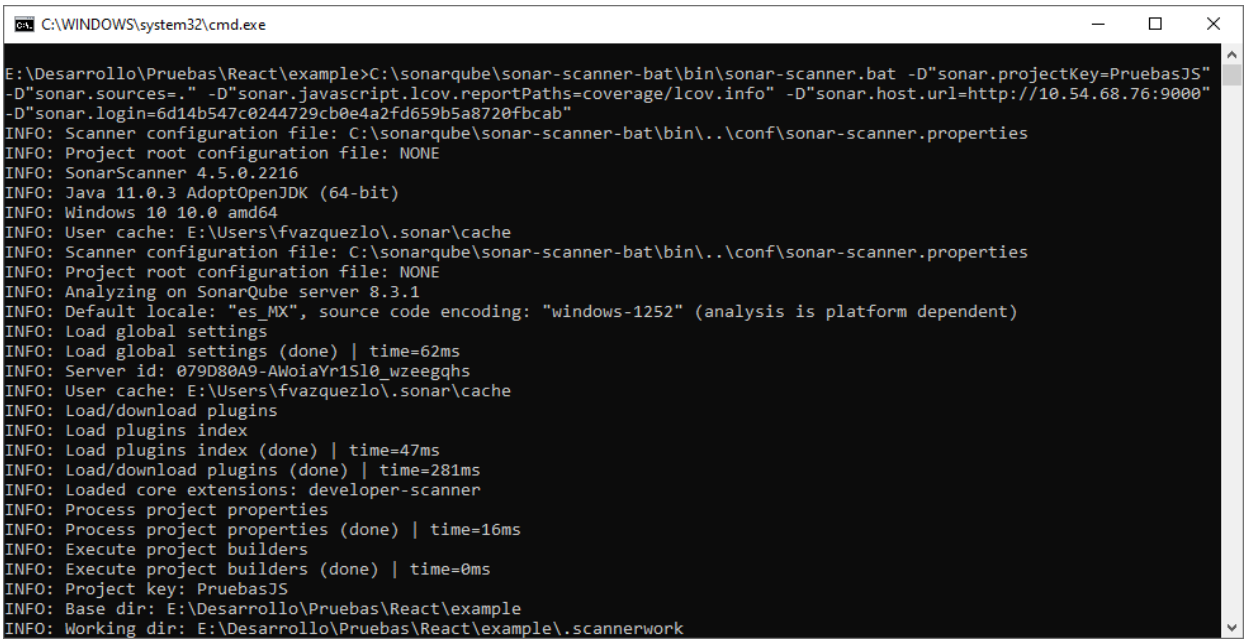
Para enviar el reporte de cobertura de código a SonarQube se debe anexar el parámetro "**sonar.javascript.lcov.reportPaths**" a al Anexo 05 – Implementación de la línea de ejecución de SonarQube, el cual recibe como parámetro la ruta del archivo lcov.info. Adicional se excluyen del coverage los archivos correspondientes a las pruebas unitarias, mediante el parámetro "**sonar.coverage.exclusions**".
Si en la terminal se encuentra situado en la carpeta raíz del proyecto, el valor de **sonar.javascript.lcov.reportPaths** debe ser: coverage/lcov.info, referencia relativa de que se analizará el código del directorio en el que se esté situado, caso contario especificar la ubicación completa de donde se encuentre dicho archivo, ejemplo: **E:\Desarrollo\Pruebas\React\example\coverage\lcov.info.** Y para **sonar.coverage.exclusions** se indica el directorio que contiene las pruebas unitarias o bien por la extensión de estos. Ejemplo: ****\src__tests__***** excluye todos los archivos dentro de cualquier directorio con el nombre __**tests**__; y ***test.js** excluye cualquier archivo que termine con **test.js**.

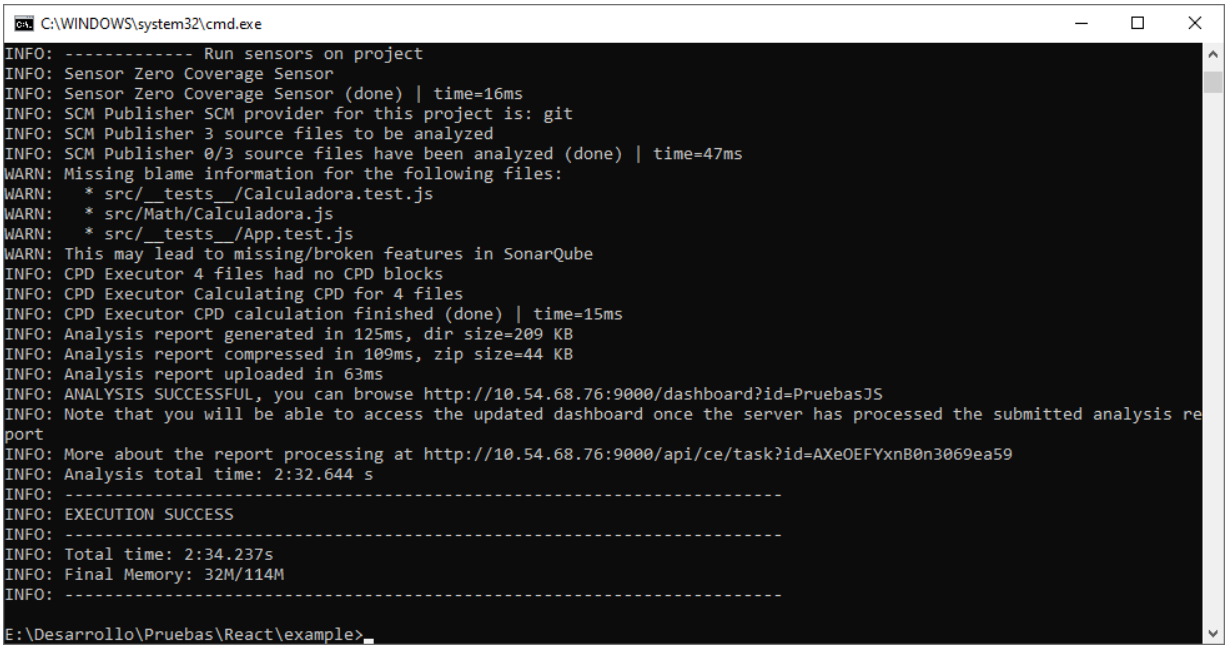
4.1.1.1. Ejemplo

```
Windows:
C:\sonarqube\sonar-scanner-bat\bin\sonar-scanner.bat -D"sonar.projectKey=PruebasJS" -
D"sonar.sources=."         -D"sonar.coverage.exclusions=**\src\__tests__\**\*,*test.js,**/*Tests.js" -
D"sonar.javascript.lcov.reportPaths=coverage\lcov.info" -D"sonar.host.url=http://10.54.68.51:9000" -
D"sonar.login=9b071bfb1b553f380989d8db5088f8f80a503b5"
```

```
Linux:
/home/qacentral/Scanner_Sonar/sonar-TS_JS/bin/sonar-scanner \
  -Dsonar.projectKey=PruebasJS \
  -Dsonar.sources=. \
  -Dsonar.host.url= http://10.54.68.51:9000 \
  -Dsonar.login=9b071bfb1b553f380989d8db5088f8f80a503b5 \
  -Dsonar.coverage.exclusions=**/src/__tests__/**/*,*test.js,**/*Tests.js \
  -Dsonar.javascript.lcov.reportPaths=coverage/lcov.info \
```

Ejecución en terminal:





Una vez terminado el análisis, ver [Ejemplo de Coverage de SonarQuebe](#).

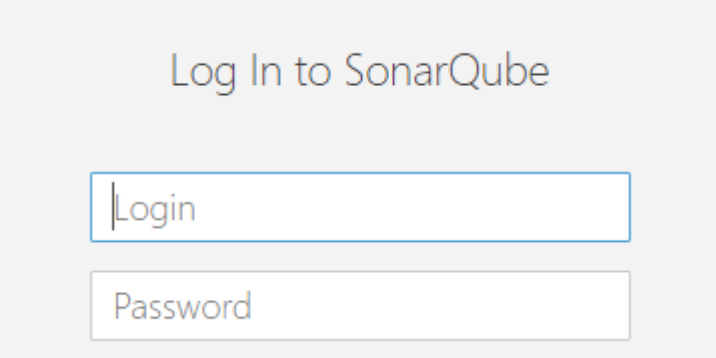
5. Anexos

5.1. Anexo 01 – Ingresar a la página SonarQube

SonarQuebe es una herramienta que permite realizar un análisis estático en el código, para ello es necesario:

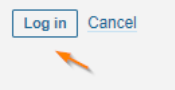
1. Validar el acceso a la página **http://10.54.68.51:9000/.**
2. Para poder loguearse, debes tener el usuario y contraseña que fue proporcionado por el equipo de QA. (Ver imagen **Anexo1.0**)
   a) Al no contar con ello, solicitarlo al correo de pruebascentrales@elektra.com.mx **.**

Una vez que ya tenemos nuestro usuario generado, vamos a entrar con nuestro número de empleado y contraseña proporcionada por el equipo de QA:



Anexo1.0: Login de SonarQube

Damos clic en el botón Log in:

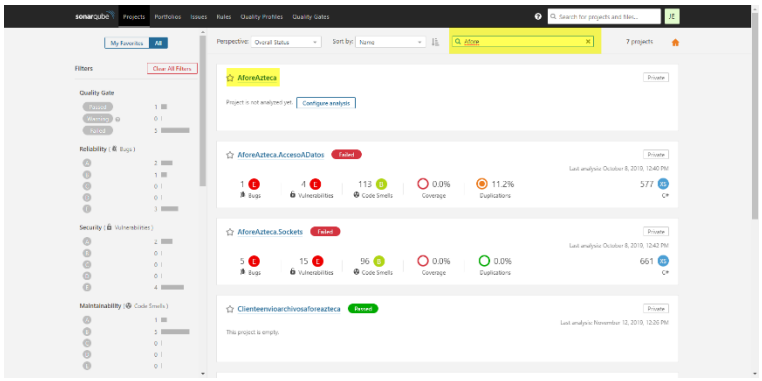


Y automáticamente vamos a ingresar al dashboard general de sonar, en este dashboard vamos a encontrar el listado de portafolios, aplicaciones y proyectos a los cuales tenemos permiso.

5.2. Anexo 02 – Configuración de proyecto en SonarQube

La configuración para un proyecto en SonarQuebe, solo se realiza una vez.

1. Buscar el proyecto que se realizará el análisis. (Ver imagen **Anexo2.0**).

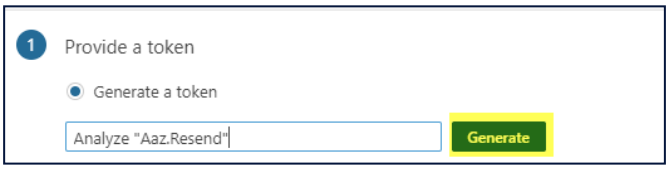


Anexo2.0: Buscar proyecto en SonarQube

2. Se escribe una descripción clave y posteriormente se da clic en botón **Generar token**. (Ver imagen **Anexo2.1**).

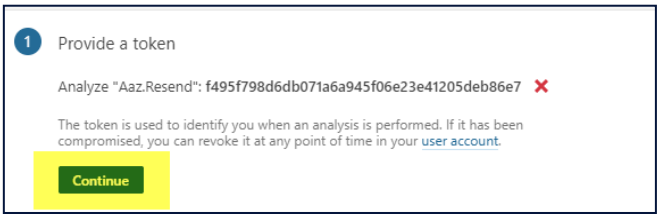
El token que se genera es importante respaldarlo ya que si se llega a perder no se puede volver a generar el mismo token y esto ocasionara que se pierda la secuencia de los análisis anteriores. Este se puede usar para todos los proyectos que se tengan asignados.

a) Al extraviar el token ir a Anexo 03 – Recuperación de token SonarQube.



Anexo2.1: Generar token.

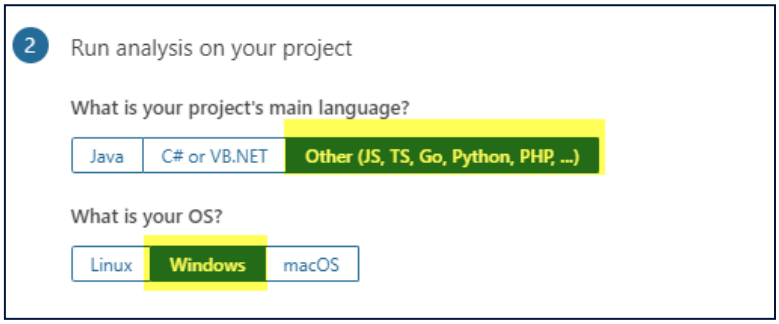
3. Dar clic en **Continuar** para seguir con la configuración. (Ver imagen **Anexo2.2**).



Anexo2.2: Finalizar la creación de token.

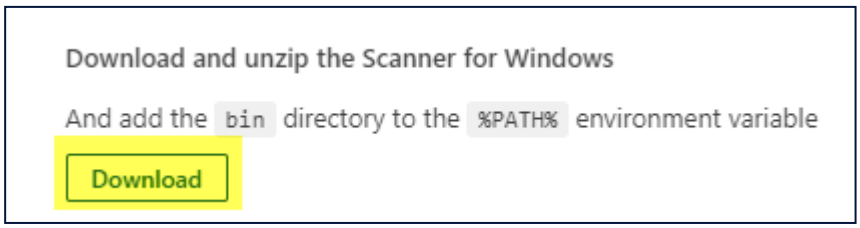
4. Para la configuración del proyecto se debe tener en cuenta: (Ver imagen **Anexo2.3**).
   - **a)** El lenguaje que se está utilizando en el proyecto.
     - **i.** Java.
     - **ii.** C# o VB.NET.
     - **iii.** Otros (JS, TS, GO, PYTON, PHP, ETC…).
   - **b)** El sistema operativo donde se encuentra.

i. Linux.
ii. Windows.
iii. macOS.



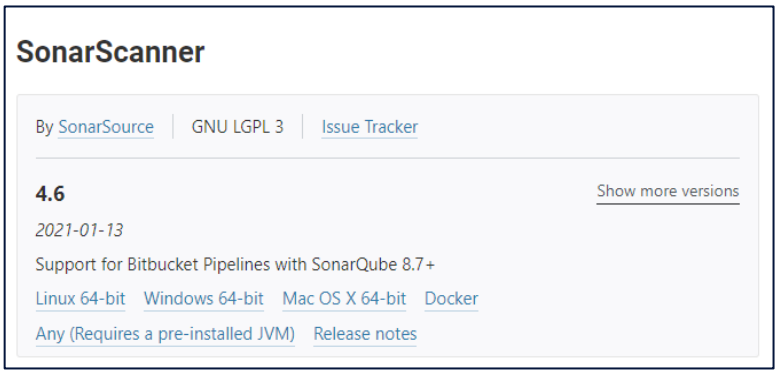
Anexo2.3: Ejemplo de proyecto para JAVA y JAVA-Script con S.O. Windows

5. Posteriormente se deberá descargar **Sonar Scanner,** que dependerá de la selección que se haya hecho en el **paso 4**. (Ver imagen **Anexo2.4**).



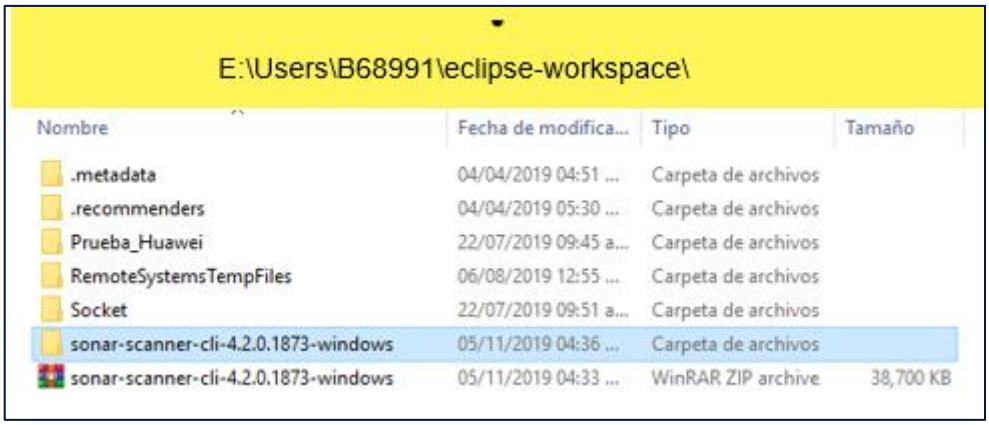
Anexo2.4: Descarga Sonar Scanner

i. También se puede descargar **Sonar Scanner**, desde la liga: https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/ , en la cual se deberá elegir el sistema operativo que se necesite. (Ver imagen **Anexo2.5**).



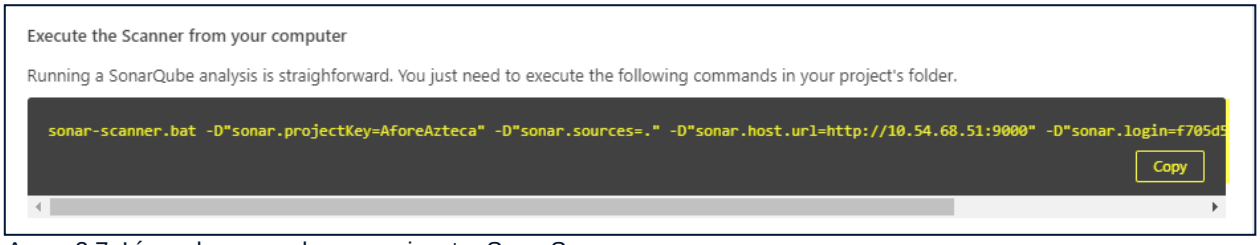
Anexo2.5: Pagina para descargar Sonar Scanner

b) Al concluir con la descarga de **Sonar Scanner** identificándolo ya que es un .ZIP, el cual se recomienda descomprimir en una carpeta nueva y que se encuentre en **C:** o bien en **E:** ya que no se moverá al iniciar con el análisis de los proyectos. (Ver imagen **Anexo2.6**).



Anexo2.6: Descomprimir Sonar Scanner, en un lugar seguro.

6. En la parte inferior de la página de **SonarQube** muestra en una línea de ejecución, para **Sonar Scanner**, la cual se debe de guardar en un .txt ya que será ocupada para todos los proyectos que se analizarán. (Ver imagen **Anexo2.7**).
   a) La línea de ejecución solo se le cambiará:
   (Ver Anexo 04 – Descripción de Línea de ejecución SonarQube)
   i. Ubicación de donde se encuentra el proyecto.
   ii. El nombre del proyecto
   iii. Servidor donde se enviará el análisis de SonarQube.

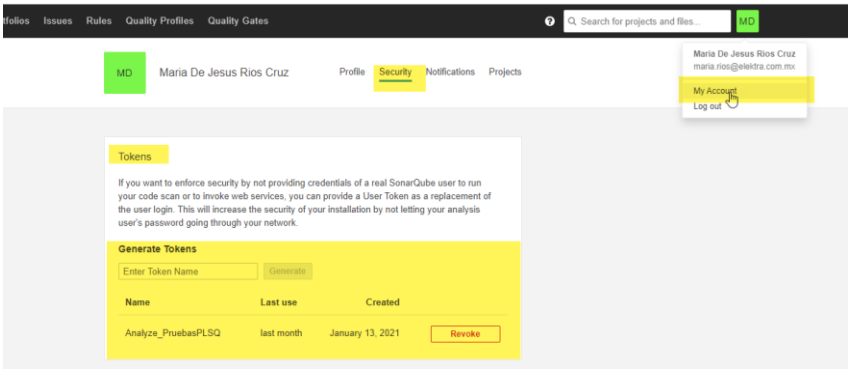


Anexo2.7: Línea de comandos para ejecutar SonarScanner

5.3. Anexo 03 – Recuperación de token SonarQube

Ingresar a la página de SonarQube, vamos a **My Account,** este lo vamos a encontrar en la parte superior derecha del portal.

Posteriormente se da clic en **Security,** se abrirá una pestaña llamada **Token,** como siguiente en la parte inferior podrás generar un nuevo token o bien ver los que has creado. (Ver **imagen Anexo3.0**).



Anexo3.0: Generar un nuevo token

5.4. Anexo 04 – Descripción de Línea de ejecución SonarQube

La línea de ejecución la proporciona la herramienta de SonarQube, después de configurar nuestro tipo de proyecto a analizar.
La línea funciona para ejecutar el análisis del proyecto desde consola, cada una **es** diferente y depende por el lenguaje de programación en que se realizará el análisis, lo que no cambia es la manera de ejecutarla en CMD es la misma, se pude realizar el análisis desde un SO**:**

- **Windows (**Ver Windows:**)**
- **Linux (**Ver Linux:**)**

Descripción de parámetros que conforma una línea para ejecución

1. **[PathSonarScanner]:** Es la ubicación del archivo sonar-scanner**.**

   Ejemplo:
   - **Windows: "C:\sonarqube\sonar-scanner-bat\bin\\sonar-scanner.bat"**
   - **Linux: /home/qacentral/Scanner_Sonar/sonar-TS_JS/bin/sonar-scanner**

2. **[PathMSBuildVS]:** Ruta donde se encuentra MSBuild de Visual Studio
   - **Ejemplo: "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Current\Bin\MsBuild.exe"**

3. **[ProjectKey]:** hace referencia al id del proyecto (project key) del proyecto a analizar**.** Para obtenerlo ver ¿Cómo se obtiene el Project Key?.

   - **Ejemplo: PruebasJS**

4. **[PathProyecto]:** es la ubicación de la carpeta raíz del proyecto a analizar**.**

   Si en la terminal se encuentra situado en la carpeta raíz del proyecto, el valor de este parámetro debe ser un punto (.), referencia relativa de que se analizará el código del directorio en el que se esté situado. Caso contario especificar la ubicación completa del proyecto, ejemplo: E:\Desarrollo\Pruebas\React\example.

5. **[URLProyecto**]: hace referencia a la ruta donde se encuentra el proyecto.
   - **Ejemplo: C:\ProyectoAnalizar**

6. **[ServidorSonarQube**]: hace referencia al servidor de SonarQube.

   - **Ejemplo: http://10.54.68.51:9000**

6.- [Token]: Token vinculado al usuario que generará el análisis.

- **Ejemplo: 9b071bfb1b553f380989d8db5088f8f80a503b5**

Ejemplo de línea para ejecución:

5.4.1. Windows:

Nota: Sustituir las descripciones de los parámetros que se encuentran en […]. Descripción de parámetros que conforma una línea para ejecución.

5.4.1.1. Línea de ejecución de Java

```
Windows JAVA:
"[PathSonarScanner]"-D"sonar.projectKey= [ProjectKey]" -D"sonar.sources=[PathProyecto]" -
D"sonar.java.binaries=[URLProyecto]" -D"sonar.host.url=[ServidorSonarQube] " -D"sonar.login=[Token]"
```

5.4.1.2. Línea de ejecución de BD

```
Windows Base de Datos:
"[PathSonarScanner]"-D"sonar.projectKey= [ProjectKey]" -D"sonar.sources=[PathProyecto]" -D-
D"sonar.host.url=[ServidorSonarQube] " -D"sonar.login=[Token]"
```

5.4.1.3. Línea de ejecución .NET

```
Windows .NET:
[PathSonarScanner] begin /k:"[ProjectKey]" /d:sonar.host.url="[ServidorSonarQube]"
/d:sonar.login="[Token]"
[PathMSBuildVS] /t: Rebuild
[PathSonarScanner] end /d:sonar.login="[Token]"
```

5.4.2. Linux:

Nota: Sustituir las descripciones de los parámetros que se encuentran en […] Descripción de parámetros que conforma una línea para ejecución.

5.4.2.1. Línea de ejecución de JAVA

```
Linux JAVA:
[PathSonarScanner] \
  -Dsonar.projectKey=[ProjectKey] \
  -Dsonar.sources=[PathProyecto] \
  -Dsonar.java.binaries=[URLProyecto] \
  -Dsonar.host.url=[ServidorSonarQube] \
  -Dsonar.login=[Token] \
```

5.4.2.2. Línea de ejecución de ANGULAR

```
Linux ANGULAR:
[PathSonarScanner] \
  -Dsonar.projectKey=[ProjectKey] \
  -Dsonar.sources=[PathProyecto] \
  -Dsonar.host.url=[ServidorSonarQube] \
  -Dsonar.login=[Token] \
```

5.4.2.3. Línea de ejecución de REACT

```
Linux REACT:
[PathSonarScanner] \
  -Dsonar.projectKey=[ProjectKey] \
  -Dsonar.sources=[PathProyecto] \
  -Dsonar.host.url=[ServidorSonarQube] \
  -Dsonar.login=[Token] \
```

5.5. Anexo 05 – Implementación de la línea de ejecución de SonarQube

Para ejecutar la línea de ejecución, y poder ver nuestro análisis en SonarQube es necesario:

1. Al tener el token y la línea de ejecución que se obtuvo al configurar el proyecto en SonarQube, ver Anexo 02 – Configuración de proyecto en SonarQube
2. Al tener la línea de ejecución es necesario cambiar los parámetros se indica en Anexo 04 – Descripción de Línea de ejecución SonarQube.

5.5.1. Windows con línea de ejecución de Java

1. Se modifica la línea de ejecución sustituyendo los datos que corresponde, teniendo como resultado: (Ver Anexo 04 – Descripción de Línea de ejecución SonarQube).

```
Windows JAVA:
“C:\sonarqube\sonar-scanner-bat\bin\sonar-scanner.bat” -D"sonar.projectKey=AforeAzteca" -
D"sonar.sources=." -D"sonar.java.binaries=C:\ruta del proyecto" -
D"sonar.host.url=http://10.54.68.51:9000" -D"sonar.login=f705d5f0d97881b0d1cdabed6cae8d2beec0ecd5"
```

2. Al tener lista la línea de ejecución, posteriormente abrir la consola CMD en modo administrador. (Ver Imagen 5.1.0).

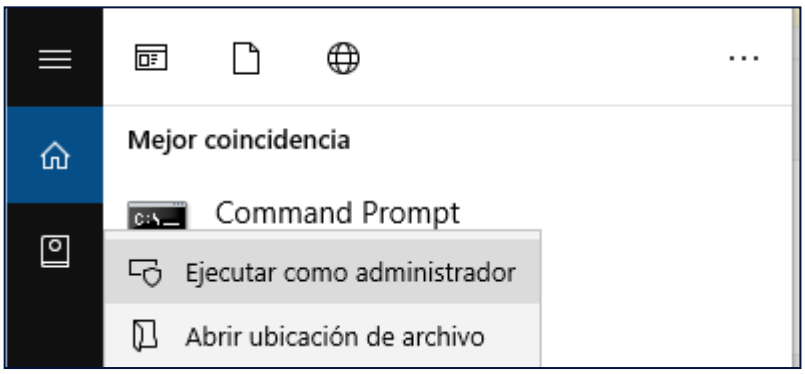


Imagen 5.1.0: Abrir cmd como administrador.

3. Debe posicionarse en la ruta donde se encuentra el proyecto a analizar. (Ver Imagen 5.1.1).
4. Al estar ubicado donde está el proyecto, debe de copiar la línea de ejecución que se configuro

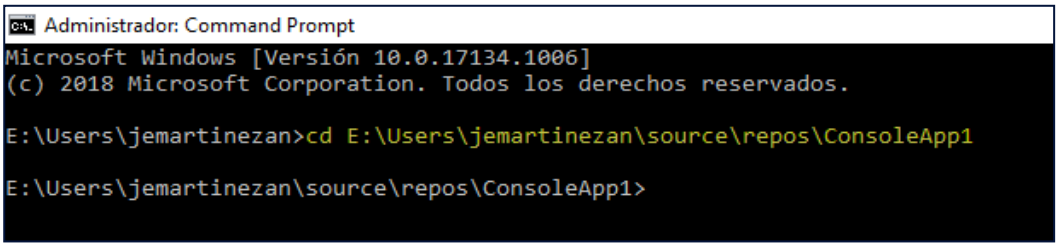


Imagen 5.1.1: CMD se posiciona en la carpeta donde está el proyecto

con anterioridad. (Ver Anexo 04 – Descripción de Línea de ejecución SonarQube).

5. Posteriormente dar clic en la tecla ENTER para poder ejecutar la línea copiada. (Ver Imagen 5.1.2).

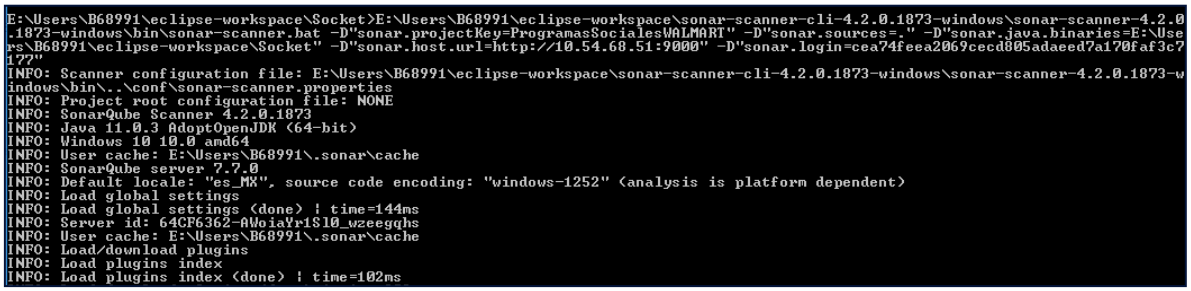


Imagen 5.1.2: CMD se posiciona en la carpeta donde está el proyecto

6. Posteriormente al concluir, en CMD se vera la leyenda "**EXECUTION SUCCESS**". Indicando que el análisis ha concluido. (Ver Imagen 5.1.3).

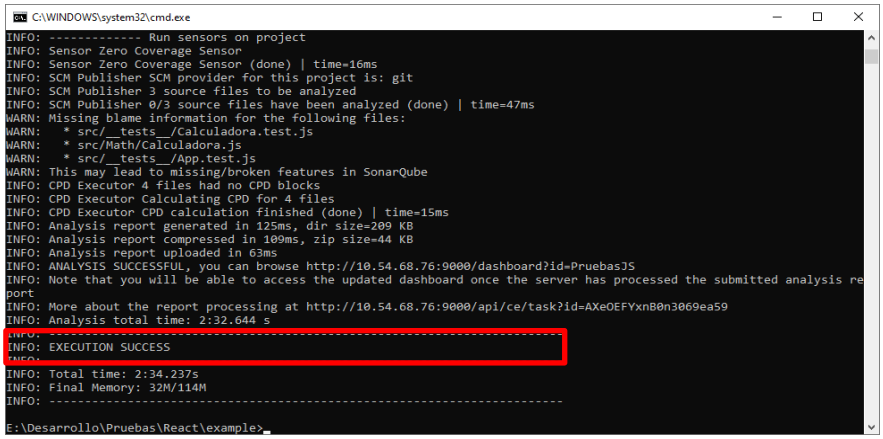


Imagen 5.1.3: Se ha finalizado el análisis de SonarQube

7. Posteriormente debe ir a la página de SonarQue, donde se encontrará el resultado del análisis que se acaba de realizar. (Ver ejemplos de Ejemplo de Coverage de SonarQuebe).

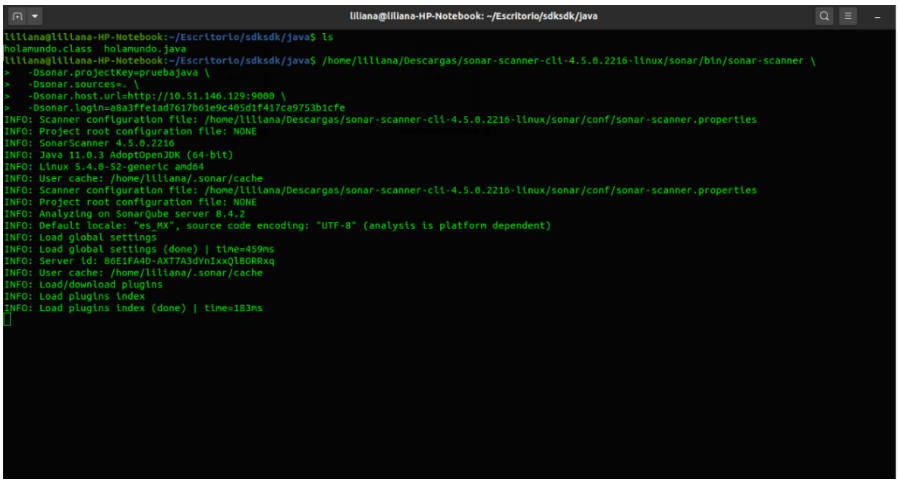
5.5.2. Linux con línea de ejecución de Java

1. Se copia previamente las líneas obtenidas en la página de SonarQube Anexo 04 – Descripción de Línea de ejecución SonarQube

```
Linux REACT:
/home/qacentral/Scanner_Sonar/sonar-TS_JS/bin/sonar-scanner \
  -Dsonar.projectKey=PruebasJS \
  -Dsonar.sources=. \
  -Dsonar.host.url=http://10.54.68.51:9000 \
  -Dsonar.login=9b071bfb1b553f380989d8db5088f8f80a503b5 \
```

2. Se completa la línea de ejecución del proyecto al que vamos analizar, cambia conforme al lenguaje que vamos analizar posteriormente la ejecución por medio de la terminal es la misma:

3. Posteriormente al haber completado la línea de ejecución, Se abre la terminal y se posiciona donde se encuentra el proyecto y se copia la línea de ejecución que hicimos, posteriormente se da clic en la tecla "**ENTER**".



4. Y se espera hasta que se concluye la ejecución, mostrando un mensaje de "**EXECUTION SUCESS**".
5. Posteriormente debe ir a la página de SonarQue, donde se encontrará el resultado del análisis que se acaba de realizar. (Ver Ejemplo de Coverage de SonarQuebe).

6. Instalación de PLUGIN en IDE

6.1. Eclipse

1. Instalación de Plugin

a) Se ingresa al IDE de Eclipse

b) Se busca el menú *Help* >> *Eclipse Marketplace...* (Ver **Imagen #1**).

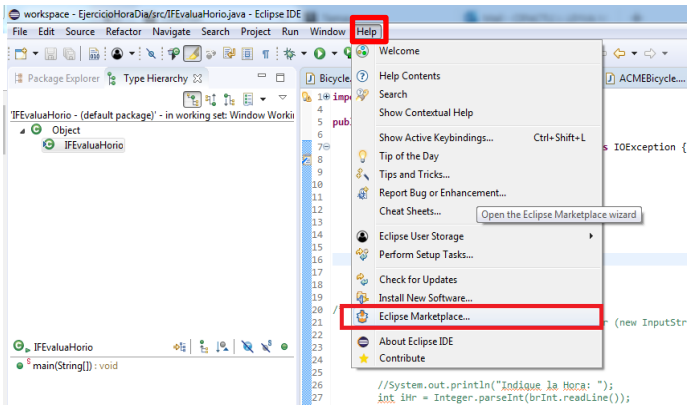


Imagen #1: Abrir la tienda del IDE Eclipse.

c) Posteriormente se abrirá la pestaña de *Marketplace,* dirigirse a la pestaña de *Serch*.

d) En el campo de **Find** escribir: *SonarLint*, y dar clic en *Install*. (Ver **Imagen #2**).

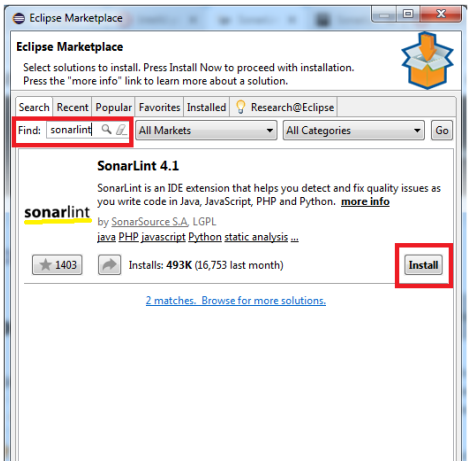


Imagen #2: Instalación de SonarLint.

2. Configuración de Plugin para realizar el análisis

a) Se dará clic derecho sobre el proyecto que vamos analizar: **Poryect >> SonarLint >> Bind to SonarQube ir SonarCloud**. (Ver **Imagen #3**).

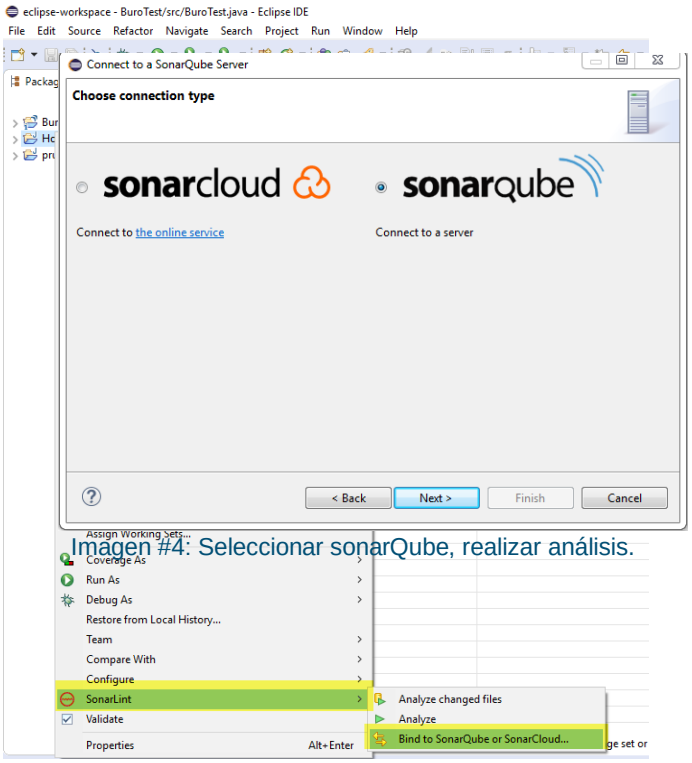


Imagen #4: Seleccionar sonarQube, realizar análisis.

Imagen #3: Configuración del proyecto para SonarLint

b) Se abrirá una nueva ventana, se debe seleccionar **SonarQube** dar clic en **Next.** (Ver **Imagen #4**).

c) Posteriormente mostrará una nueva pestaña, y se deberá ingresar la URL del servidor de SonarQube. Posteriormente se le da clic en Next**.**

(i) La URL se proporcionará en caso de no contar con la información **http://10.54.68.51:9000/.** (Ver **Imagen #5**).

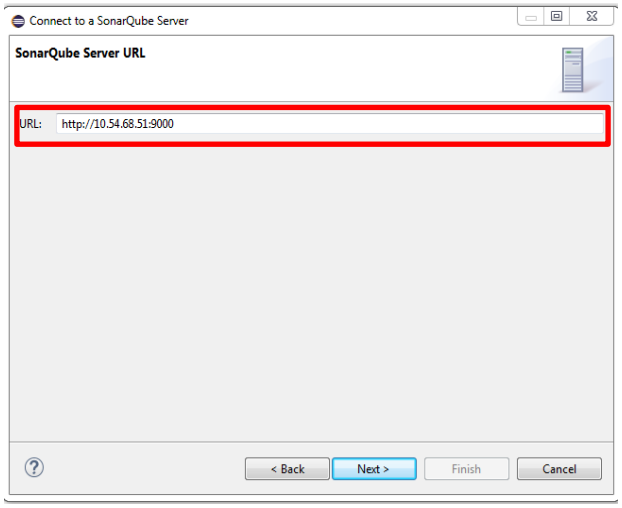


Imagen #5: Agregar URL del servidor.

d) Posteriormente se mostrará una pestaña, en la cual se seleccionará **User + Password**.

e) Se deberá colocar el usuario y contraseña que proporciono el equipo de QA. Al no contar con ello, solicitarlo al correo a pruebascentrales@elektra.com.mx. (Ver **Imagen #6**).

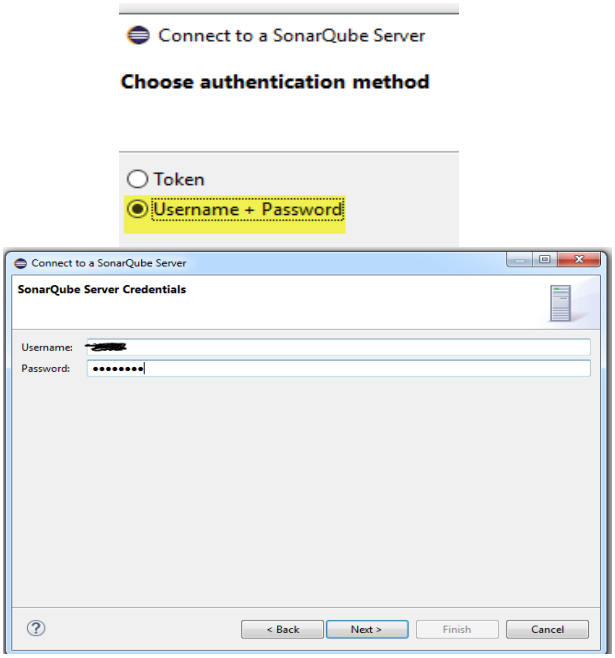


Imagen #6: Agregar Usuario y password.

f) En la siguiente pantalla es donde se colocará el **nombre de conexión**, el nombre debe ser como lo vamos a identificar más adelante.

g) Posteriormente dar clic en **FINISH.** (Ver **Imagen #7**).

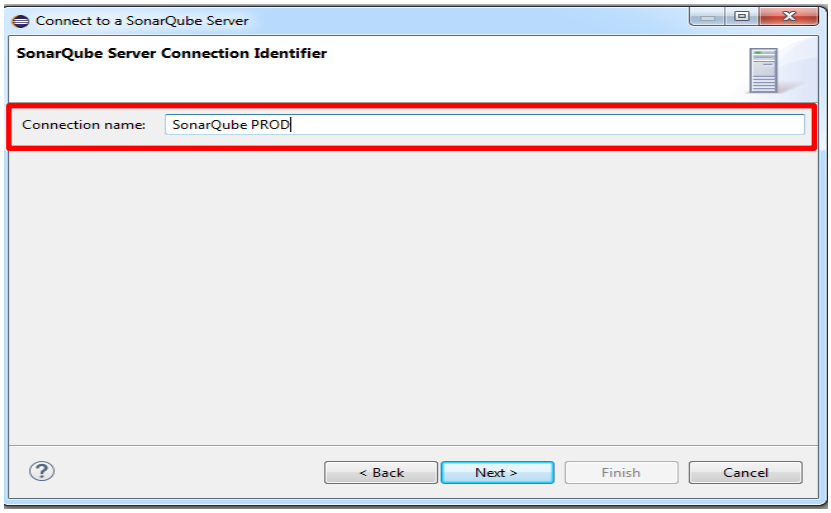


Imagen #7: Agregar un nombre de conexión.

h) Para agregar un proyecto, que se va analizar por Default. (Ver **Imagen #8**).

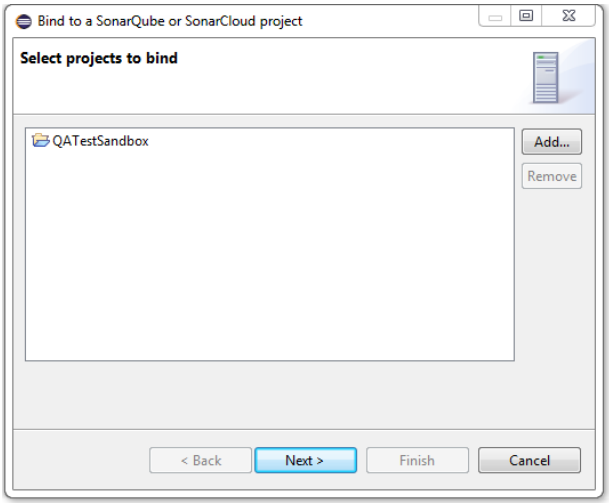


Imagen #8: Agregar un proyecto para el análisis SonarLint.

i) Indicar el nombre del proyecto y dar clic en **FINISH.** (Ver **Imagen #9**).

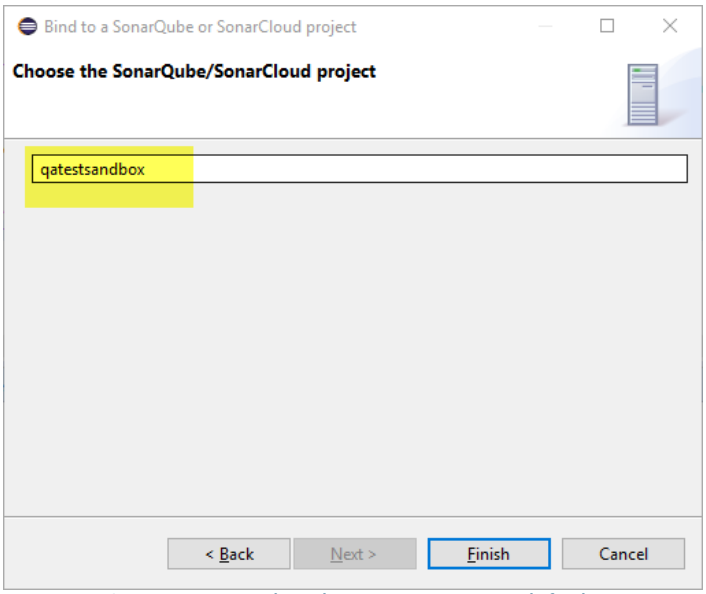


Imagen #9: Agregar nombre de un proyecto por default.

3. Ejecución de análisis con el Plugin IDE Eclipse

a) Se da clic derecho sobre el **proyecto a analizar >> Sonar Lint >> Analyze**. (Ver **Imagen #10**).

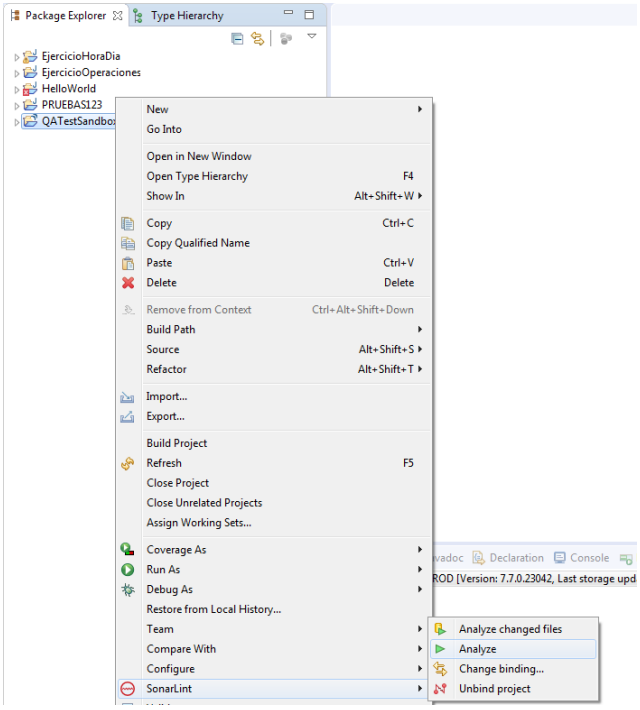


Imagen #10: Analizar el proyecto por SonarLint.

b) En la parte inferior de la pantalla, se abrirá una venta llamada **SonarLint Report**, ahí se podrán observar las incidencias que se estan presentando despues de analizar el proyecto. (Ver **Imagen #11**).

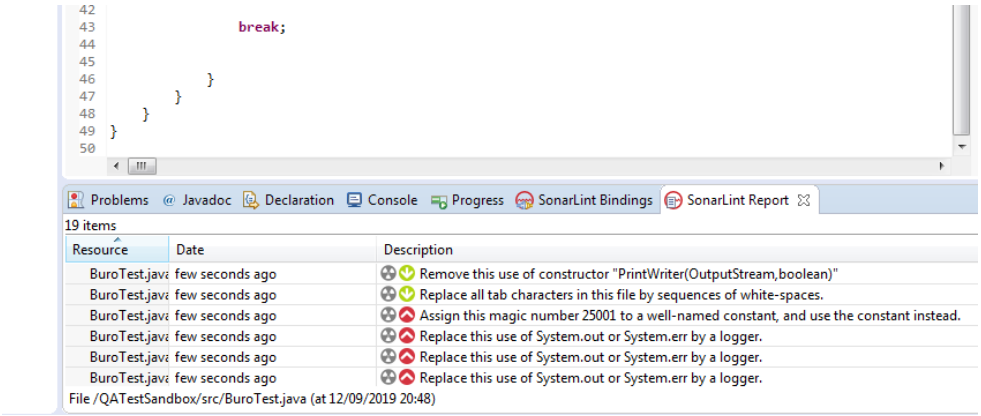


Imagen #11: Resultado de análisis de SonarLint en IDE.

i. En la imagen se muestra el resultado de las incidencias, las cuales se le pueden dar doble clic para que se muestre un mejor detalle, estas se definen como:

1. **Flecha verde**: Se encuentra en un impacto menor.
2. **Flecha Roja:** Se encuentra en un impacto crítico.

6.2. Intelli J

Para comenzar con la configuración de SonarLint es necesario:

- Tener la última versión actualizada de Intelli J.
- Tener JDK 1.8 (8) u 11 (sólo esos 2 están soportados).
- Tener configuradas apropiadamente el JAVA_HOME y Path en variables de sistema.

1. Instalación del PLUGIN

a) Abrir **IDE intelliJ**. (Ver **Imagen 7.2.0**).

Imagen 7.2.0: Iniciando IDE.

b) Posteriormente se empezará la configuración, Ir a la pestaña **File->Settings.** (Ver **Imagen 7.2.1**).

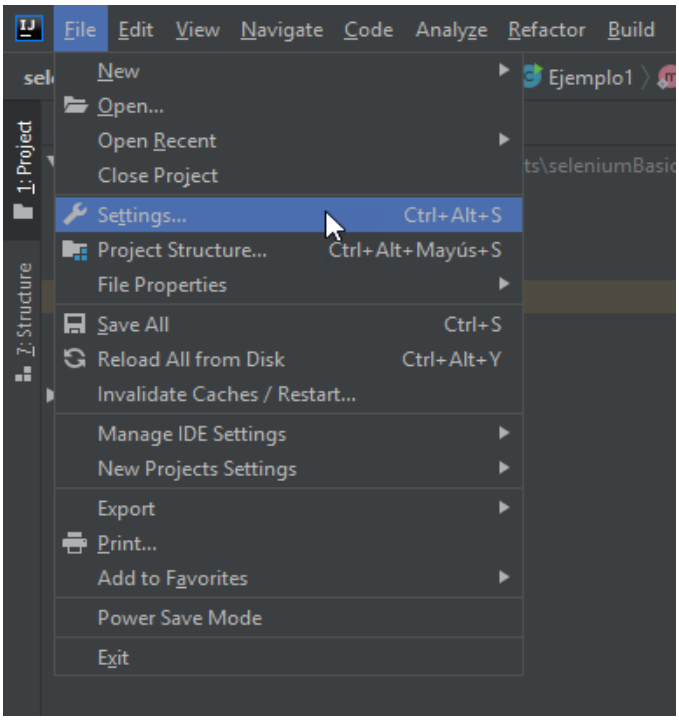


Imagen 7.2.1: Búsqueda de file -> herramientas.

c) Seleccionar la opción **Plugins->Marketplace** y buscar el **pluginSonarLint**. (Ver **Imagen 7.2.2**).

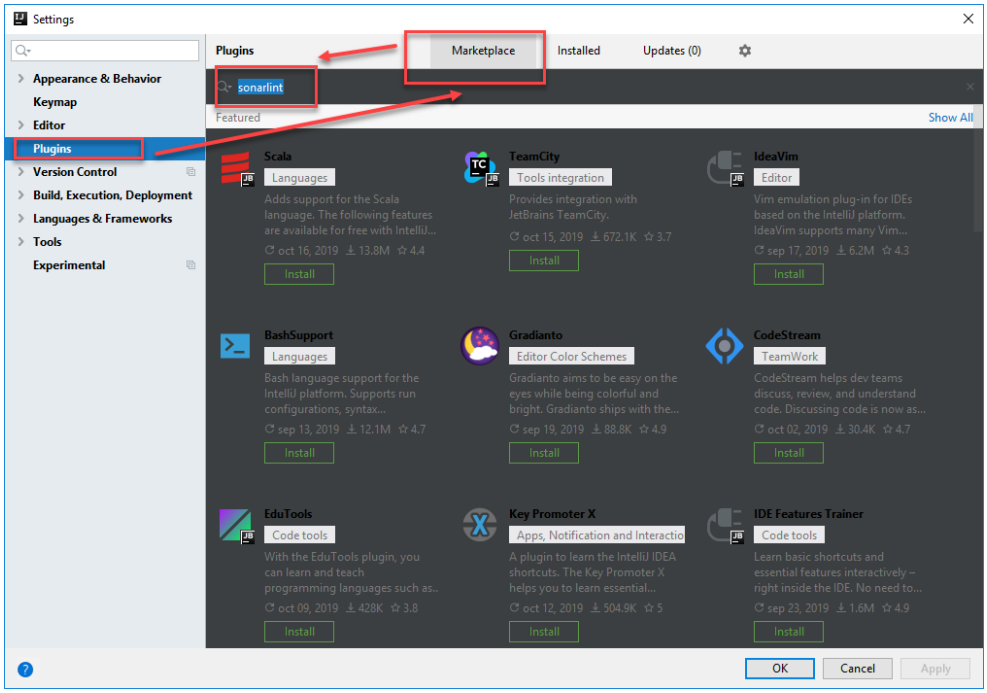


Imagen 7.2.2: Tienda de Intelli J

d) Una vez que la búsqueda sea satisfactoria se procede con la instalación del plugin. (Ver **Imagen 7.2.3**).

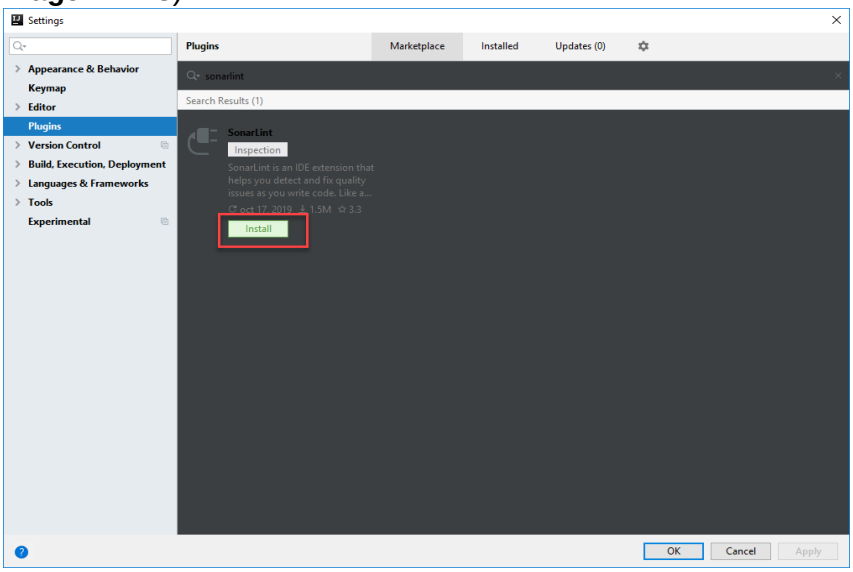


Imagen 7.2.3: Instalación de SonarLint

e) Aparecerá la siguiente ventana y damos clic en **Accept.** (Ver **Imagen 7.2.4**).

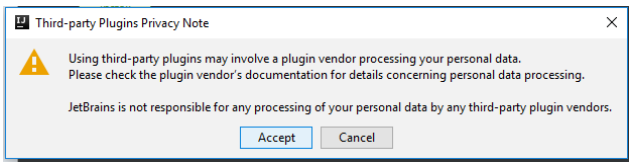


Imagen 7.2.4: Aceptación de mensaje de instalación.

f) Una vez finalizada la instalación solicitará **reiniciar el IDE**. (Ver **Imagen 7.2.5**).

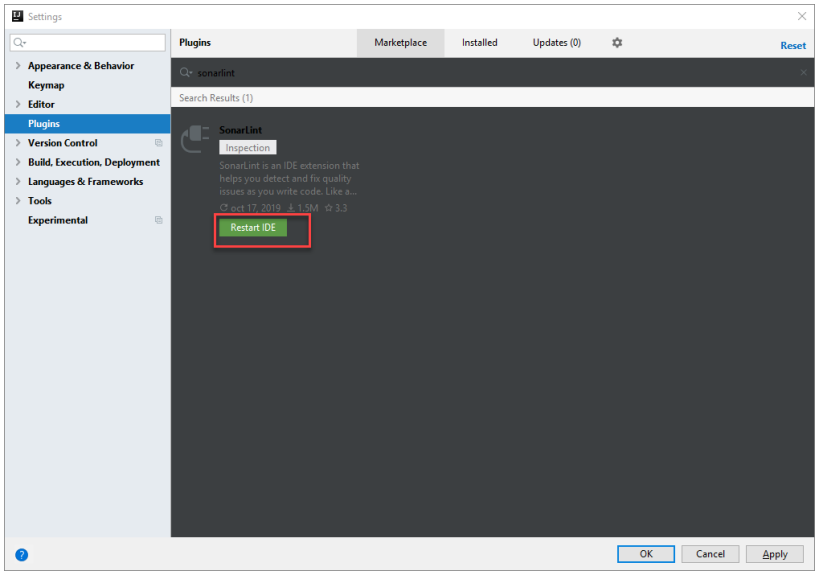


Imagen 7.2.5: Finalización de la instalación.

g) Una vez que se haya reiniciado, nos ubicamos nuevamente en la pestaña **File->Settings.** (Ver **Imagen 7.2.6**).

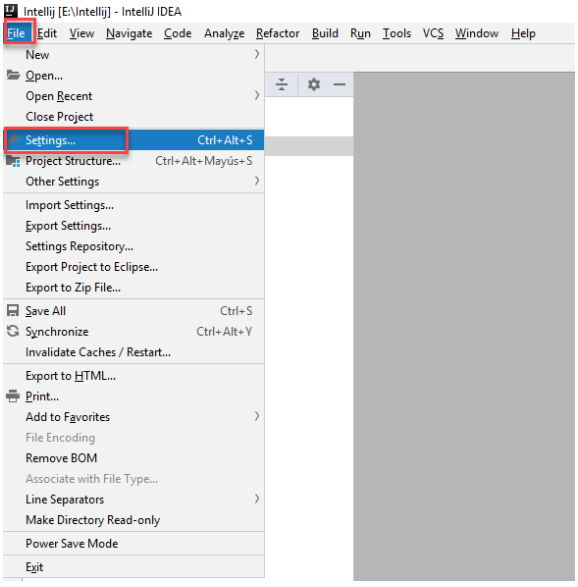


Imagen 7.2.6: Ir al sub menú settings

h) En la opción de **Other Settings** nos aparecen 2 nuevas opciones de **SonarLint**, selecciona la opción remarcada y damos clic en el símbolo señalado. (Ver **Imagen 7.2.7**).

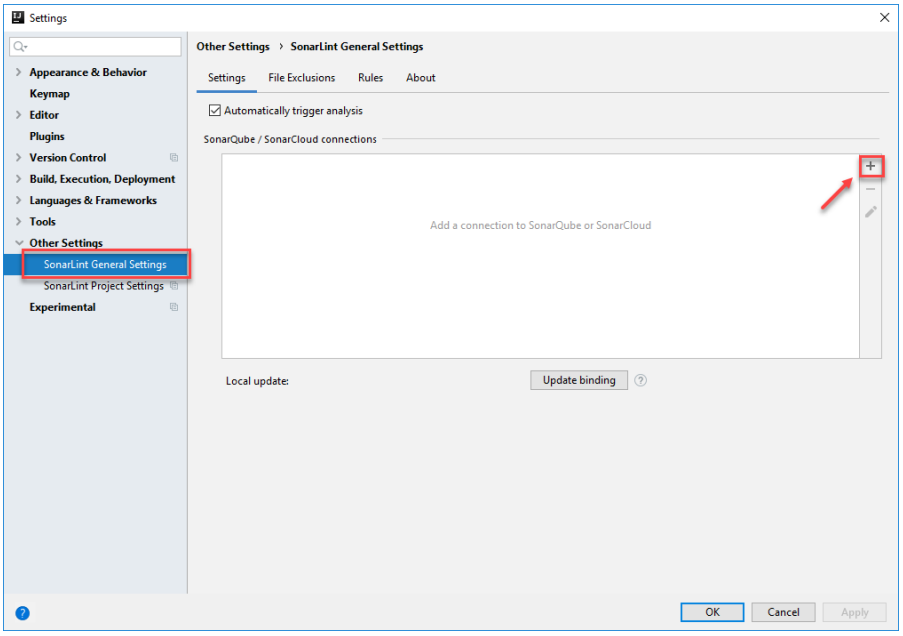


Imagen 7.2.7: Ir a la opción Other Settings

i) Llenamos los campos solicitados y se **selecciona la opción de SonarQube**.
Y en el campo inferior derecho se coloca la URL SonarQube Producción http://10.54.68.51:9000.
Posteriormente se da clic en **NEXT**. (Ver **Imagen 7.2.8**).

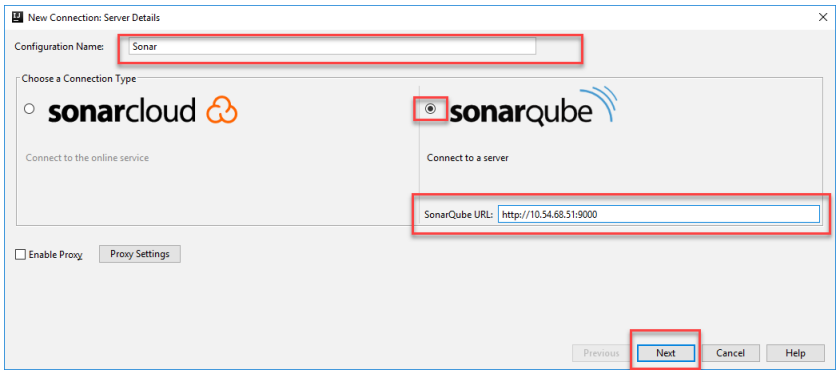


Imagen 7.2.8: Llenar los datos solicitados

j) Posteriormente muestra una pestaña donde se debe colocar el usuario y contraseña que proporciono el equipo de QA. (Ver **Imagen 7.2.9**).
Al no contar con ello, solicitarlo al correo de pruebascentrales@elektra.com.mx .

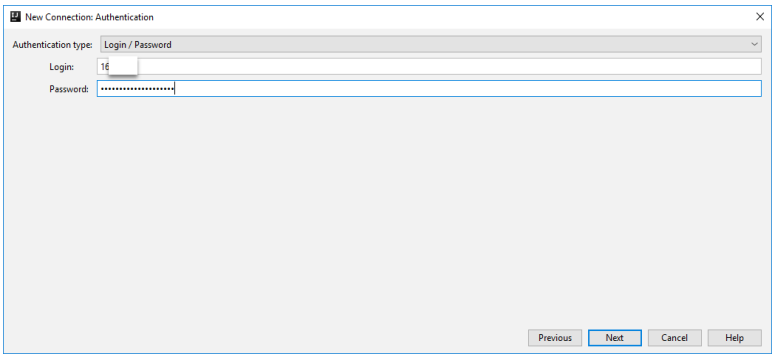


Imagen 7.2.9: Agregar Usuario y contraseña

k) Finalizamos la configuración del servidor de Sonar. (Ver **Imagen 7.2.10**).

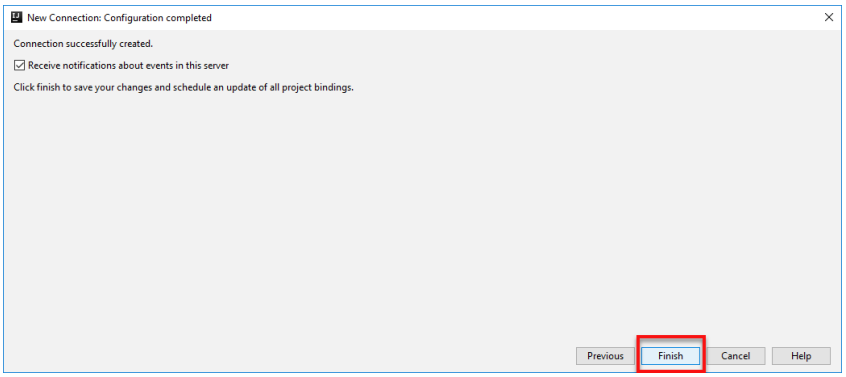


Imagen 7.2.10: Se termina la configuración

l) Posteriormente Seleccionamos la opción **SonarLint Project Settings** y en **Connection** seleccionamos la opción configurada anteriormente, y se habilitara la opción **Search in list.** (Ver **Imagen 7.2.11**).

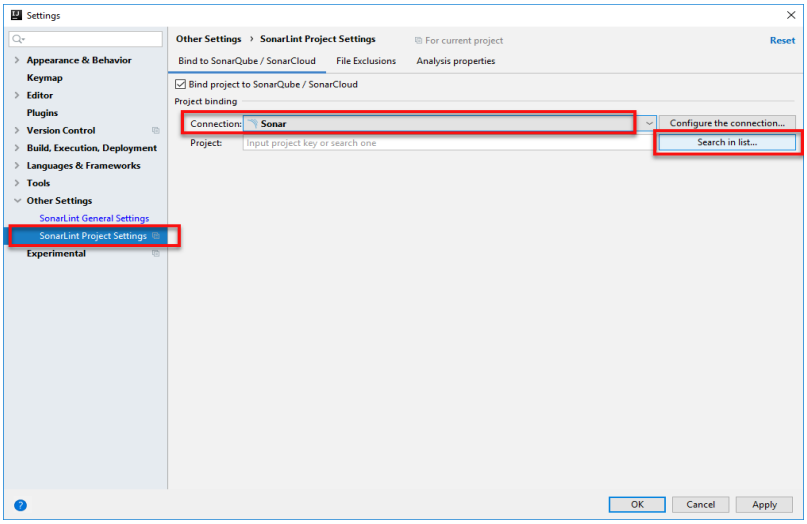


Imagen 7.2.11: Se conecta a la configuración antes realizada

m) Posteriormente aparecerá esta ventana con una lista de proyectos a los cuales tenemos acceso, seleccionamos el proyecto a analizar y damos clic en **OK**. (Ver **Imagen 7.2.12**).

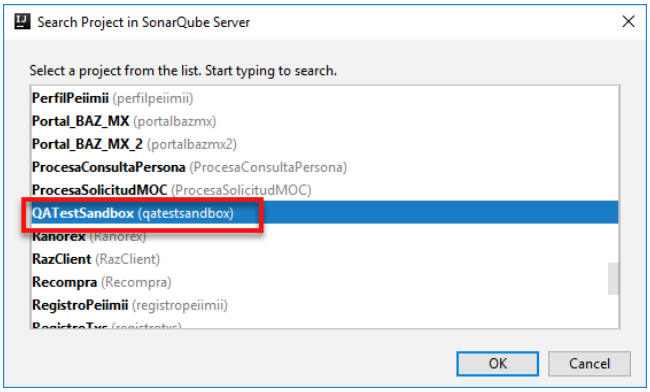


Imagen 7.2.12: Selección de un proyecto.

2. Ejecución de análisis

a) Una vez finalizada la configuración, iniciamos con el análisis, damos click dererecho sobre el proyecto a analizar y seleccionamos **SonarLint->Analyze with SonarLint**, comenzará a hacer el análisis local. (Ver **Imagen 7.2.13**).

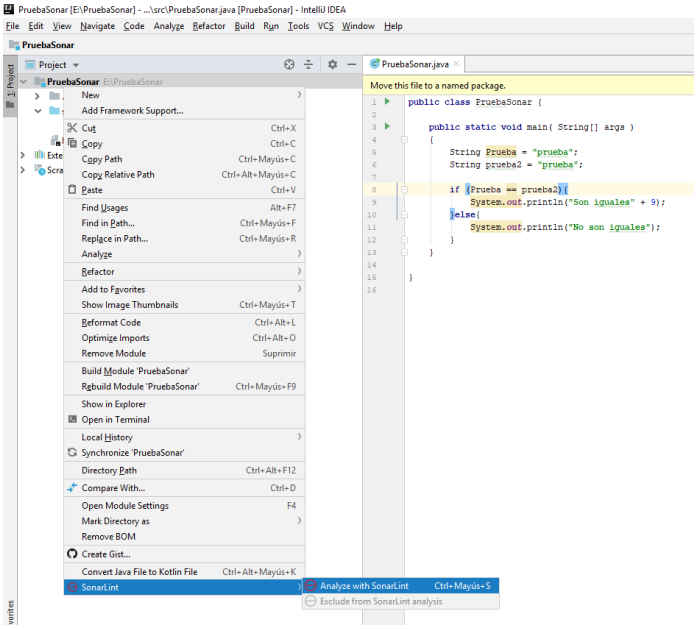


Imagen 7.2.13: Se selecciona SonarLint.

b) Aparecerá la siguiente ventana damos clic en **Proceed.** (Ver **Imagen 7.2.14**).

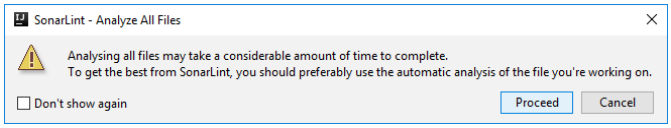


Imagen 7.2.14: Aceptar para comenzar el análisis de SonarLint.

c) Una vez finalizado el análisis en la parte inferior aparecerán los diferentes **blockers o codesmells** a mejorar**.** (Ver **Imagen 7.2.15**).

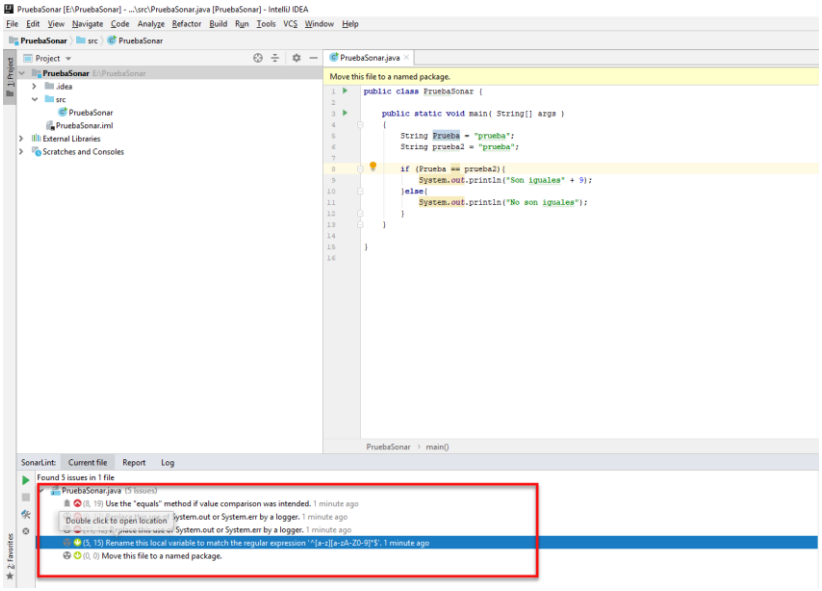


Imagen 7.2.15: Resultado del análisis de SonarLint.

6.3. Visual Studio (ASPX .NET, C#, C, C++)

Para la instalación del plugin en .net, seguiremos los siguientes pasos:

1. Se descargará SonarLint desde el menú **herramientas >> Extensiones y Actualizaciones**. (Ver **Imagen 7.3.0**).

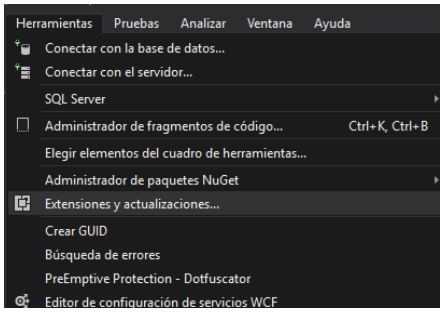


Imagen 7.3.0: Abrir el menú para descargar SonarLint.

2. Instalar SonarLint y reinciar Visual Estudio. (Ver **Imagen 7.3.1**).

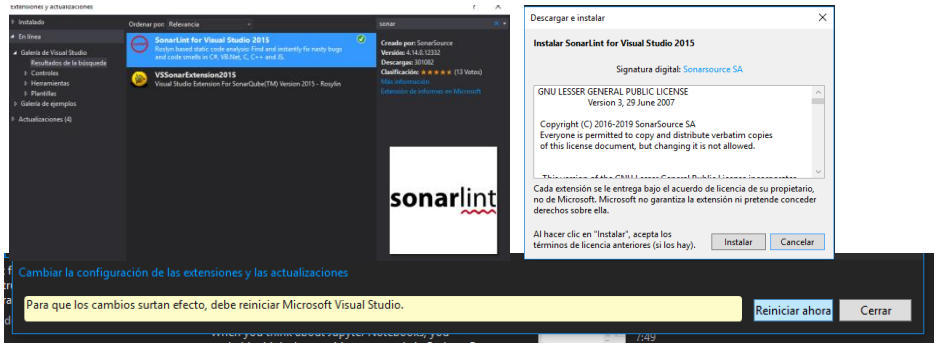


Imagen 7.3.1: Buscar e instalar SonarLint.

3. En el menú **Analizar**, se posiciona en la pestaña de **Manage SonarQube Connections** para realizar la conexión con **SonarQube** y poder obtener las reglas. (Ver **Imagen 7.3.2**).

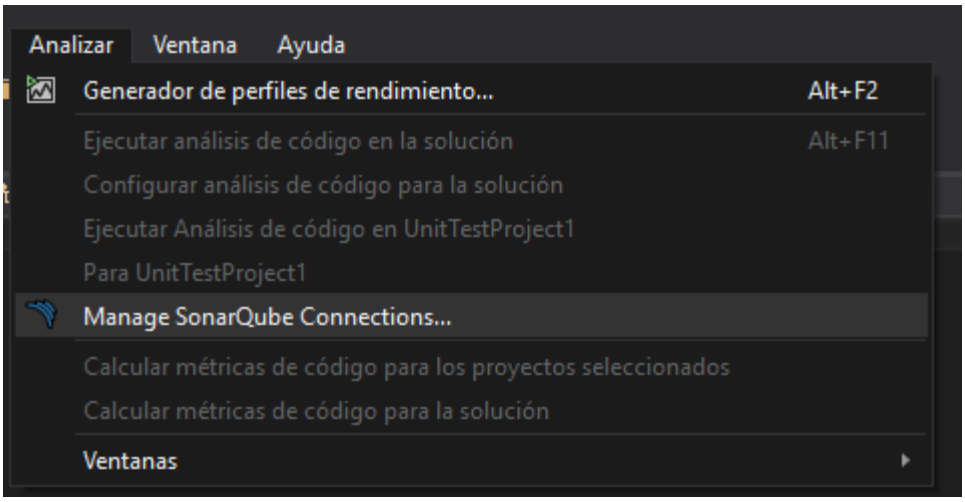


Imagen 7.3.2: Configuración de SonarLint.

4. Ingresamos usuario y password enviados previamente por el equipo de QA ya que son necesarios para realizar la conexión a sonarqube. (Ver **Imagen 7.3.3**).
    a. **URL: http://10.54.68.51:9000**
    b. Al no contar con ello, solicitarlo al correo de pruebascentrales@elektra.com.mx .

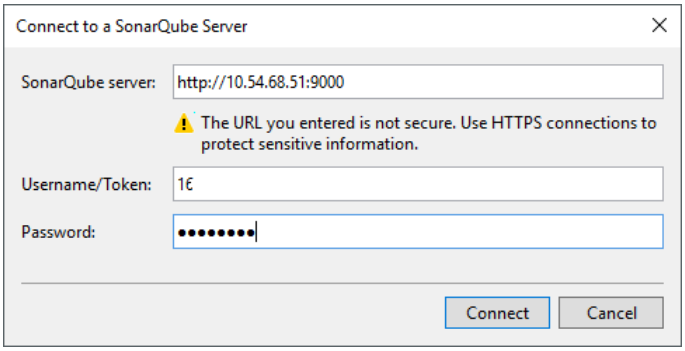


Imagen 7.3.3: Configuración de usuario y contraseña para SonarLint.

5. En la ventada de **Team Explorer** vamos a ver la siguiente pantalla la cual indica la conexión realizada con éxito y los proyectos asignados en SonarQube. (Ver **Imagen 7.3.4**).

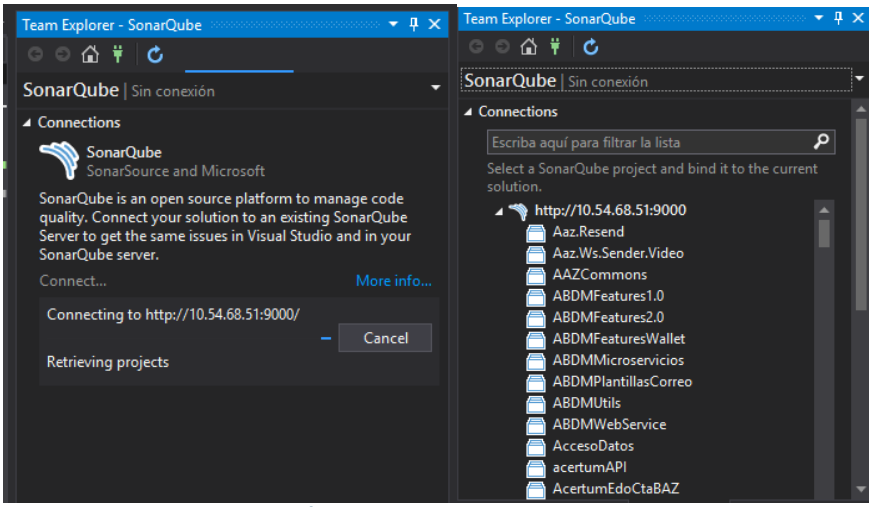


Imagen 7.3.4: Ver la conexión de SonarLint.

6. Para ejecutar un análisis local damos nos posicionamos en nuestro proyecto, clic derecho y en el siguiente menú Análisis y Limpieza de código y ejecutar limpieza de código. (Ver **Imagen 7.3.5**).

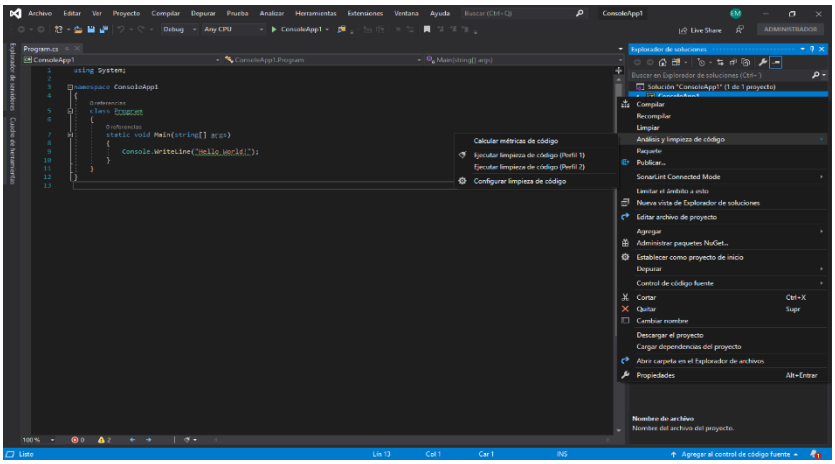


Imagen 7.3.5: Ejecutar SonarLint.

7. En la pestaña de Advertencias nos muestra lo analizado por SonarQube, la letra S son derivados de SonarQube y la C directamente de Visual Studio. (Ver **Imagen 7.3.5**).

Ejemplo: S1118

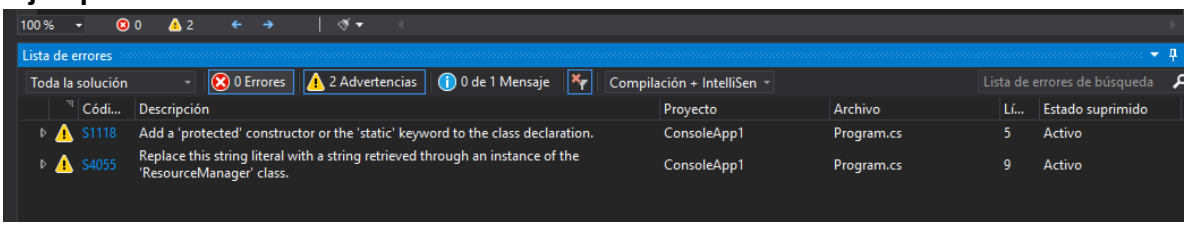


Imagen 7.3.5: Resultado al ejecutar SonarLint.

6.4. Visual Studio Code

Para poder instalar el **PLUGIN** es necesario contar con:

1. La última versión actualizada de Visual Studio Code.
2. JDK 1.8 (8) u 11 (sólo esos 2 están soportados).
3. Tener configuradas apropiadamente el JAVA_HOME y Path en variables de sistema.

Posteriormente seguir los pasos:

1) Abrimos nuestro **Visual Studio Code**, y nos dirigimos a la ventana de extensiones.

2) En el menú lateral se selecciona "**Buscar extensiones en MarketPlace**" y posteriormente escribimos "**SonarLint**". (Ver **Imagen 7.4.0**).

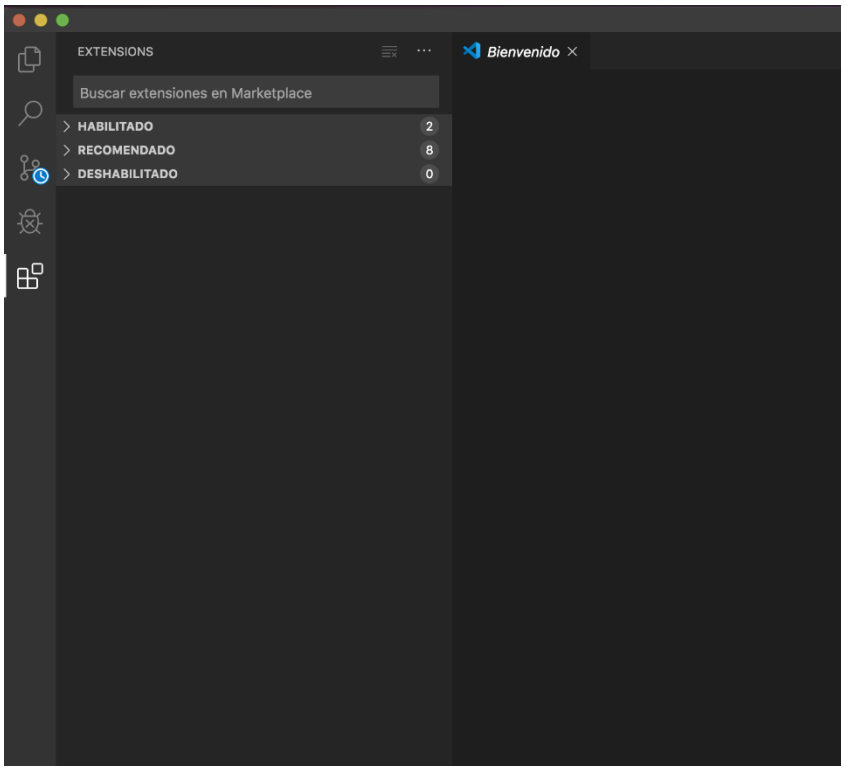


Imagen 7.4.0: Se busca SonarLint.

3) Aparecerá el **PLUGIN** y damos clic en **Instalar**. (Ver **Imagen 7.4.1**).

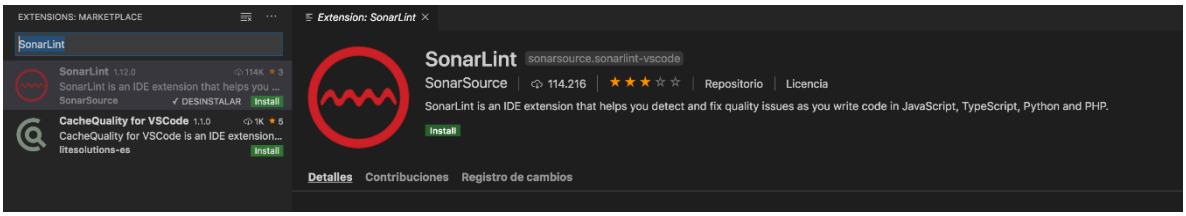


Imagen 7.4.1: Se instala SonarLint.

4) En nuestra barra de menú inferior, en el apartado de **Problemas**, aparecerán las observaciones del análisis del código. (Ver **Imagen 7.4.2**).

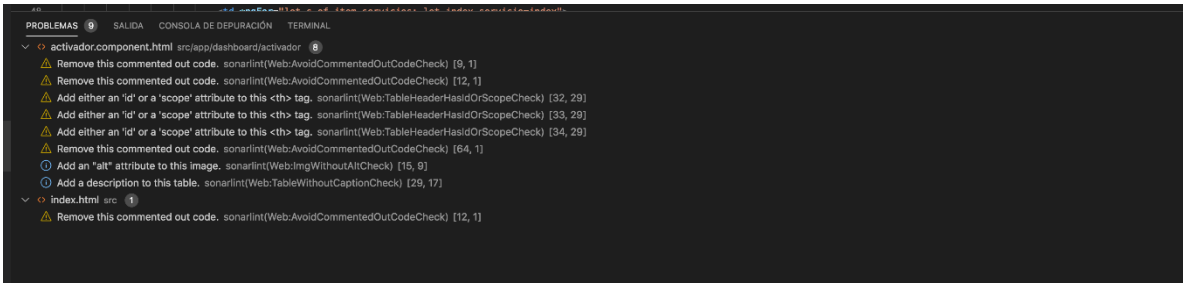


Imagen 7.4.2: Observaciones del análisis de SonarLint.

5) Nos dirigimos nuevamente a nuestro apartado de **PLUGINS**, y en el menú de **Habilitado**, Se da clic en el icono de engrane que tiene nuestro SonarLint en la parte inferior derecha. Abrirá un menú y se dará clic en **Configure las opciones de extensión**. (Ver **Imagen 7.4.3**).

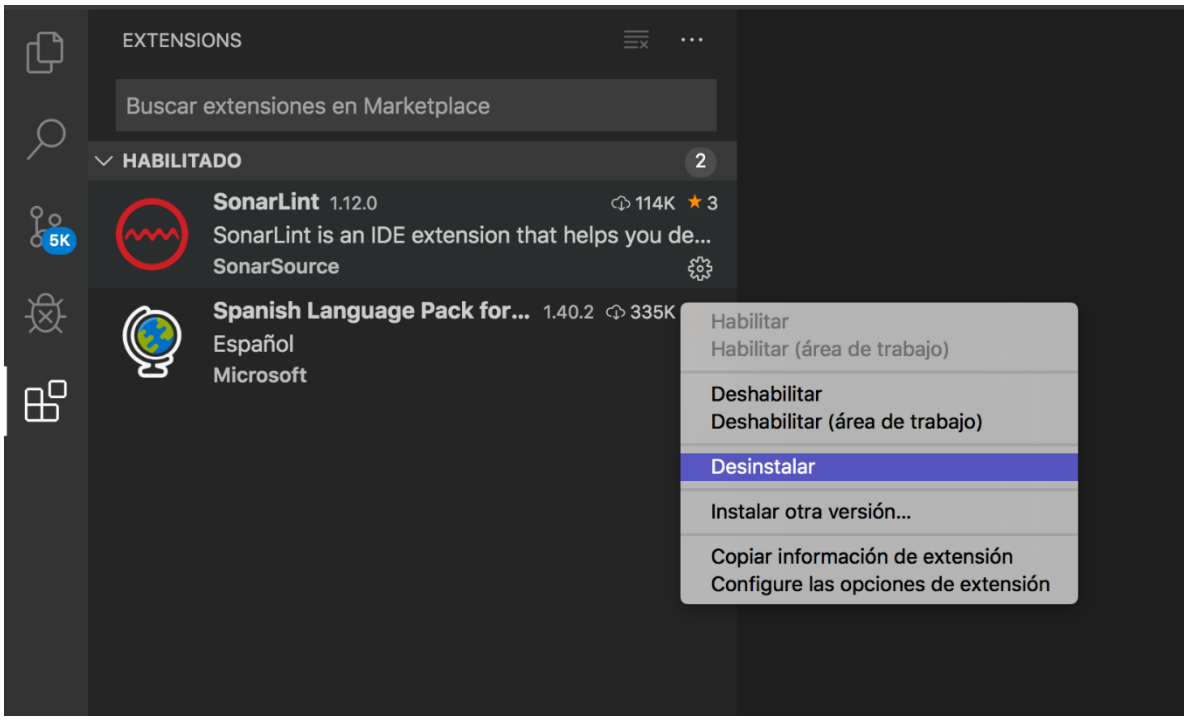


Imagen 7.4.3: Configuración de SonarLint.

6) Dentro del menú, se encontrarán las configuraciones que se harán del servidor de SonarQube. Para preparar la configuración se da clic en **Editar en settings.json**. (Ver **Imagen 7.4.4**).

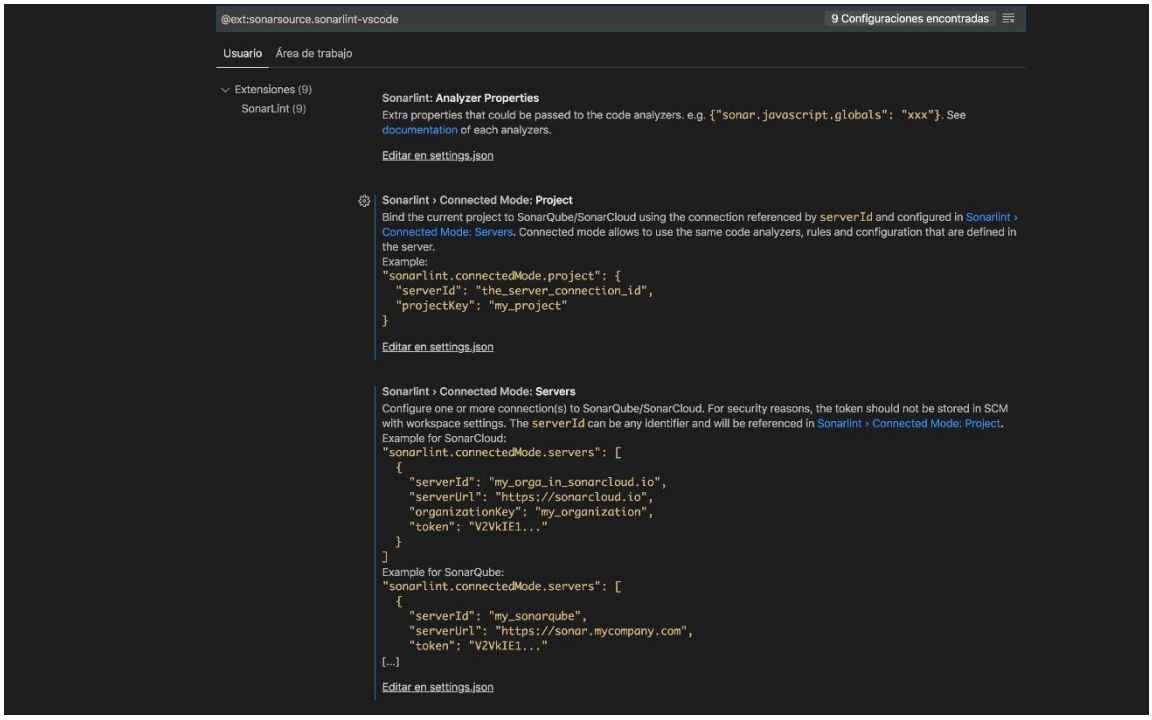


Imagen 7.4.4: Configuración de SonarLint.

7) Dentro del **settings.json**, vamos a cargar dos configuraciones, el **connectedMode.servers** y **connectedMode.project**: (Ver **Imagen 7.4.5**).
8) El **sonarlint.connectedMode.servers** configura la conexión al SonarQube requiriendo los siguientes parámetros:
    a) **serverId**: Identificador único para hacer referencia al proyecto SonarQube al cual nos conectaremos.
    b) **serverUrl**: Dirección URL de nuestro SonarQube el cual puede ser:
        (a) **SonarQube Producción: http://10.54.68.51:9000**
    c) **token:** Es un código único generado dentro de SonarQube.
9) El **sonarlint.connectedMode.project** sirve para crear un identificador único de nuestro proyecto en SonarQube para instanciarlo dentro de Visual Studio Code requiriendo los siguientes parámetros:
    a) **serverId**: Nombre para identificar nuestro proyecto dentro de nuestra configuración de **sonarlint.connectedMode.servers.**

b) **projectKey**: Llave de nuestro proyecto en SonarQube.

```
{
    "window.zoomLevel": 1,

    "sonarlint.connectedMode.servers": [
        {
          "serverId": "Web_PortalPM",
          "serverUrl": "http://10.54.68.51:9000",
          "token": "b83567e46b30d8c0a2996ba51a3f2431439fb7fd"
        }
    ],
    "sonarlint.connectedMode.project": {
          "serverId": "Web_PortalPM",
          "projectKey": "WebPortalPM"
        }

}

```

Imagen 7.4.5: Configuración de SonarLint.

10) Una vez configurado el **settings.json**, se ejecuta la combinación de teclado **Command+Shift+P** en Visual Studio Code. En la parte superior encontraremos un campo de texto.

a) Se buscará **SonarLint**, y se seleccionará la opción de **Update all bindings to SonarQube/SonarCloud**. (Ver **Imagen 7.4.6**).

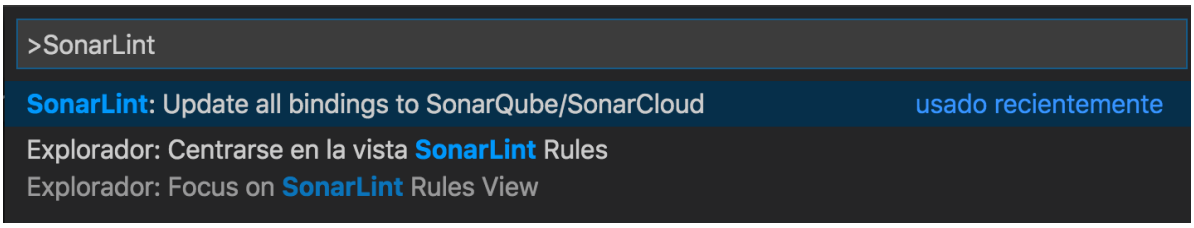
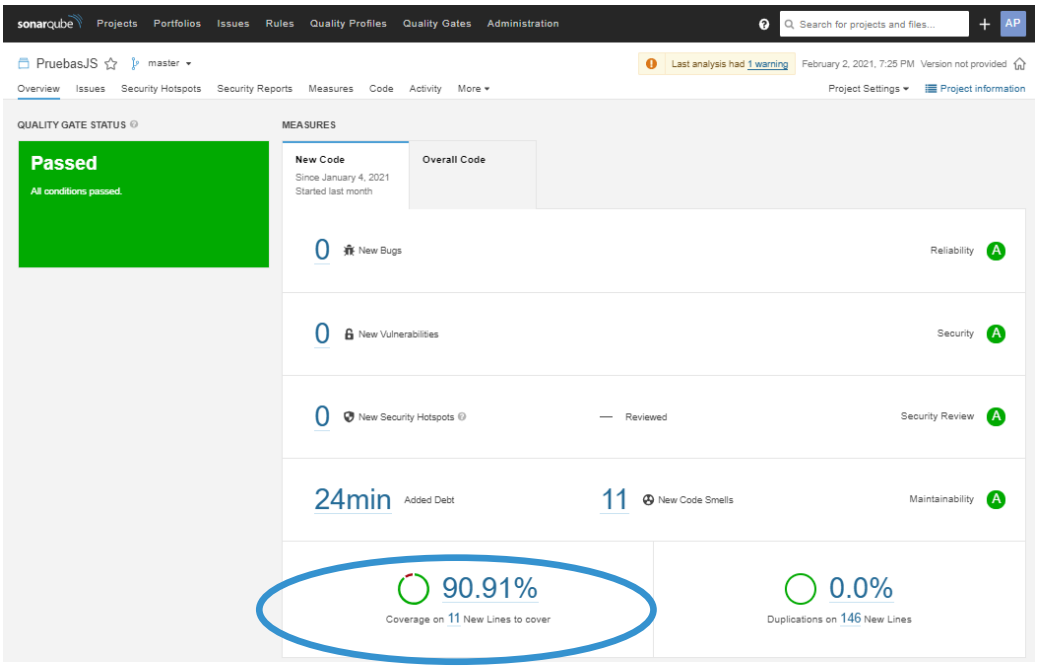


Imagen 7.4.5: Configuración de SonarLint.

11) Al finalizar, volvemos a nuestro explorador de proyecto, y nos aparecerá la lista de reglas de SonarLint.

7. Ejemplo de Coverage de SonarQuebe

Coverage es un medidor de cobertura de líneas de código, al tener un informe de cobertura de código tiene ventajas, ya que nos permite saber en todo momento si nuestras pruebas unitarias están cubriendo la mayor parte de nuestro código o no.
Una vez ejecutadas las líneas de ejecución (Ver Anexo 05 – Implementación de la línea de ejecución de SonarQube), muestra en la página de SonarQube un reporte de cobertura, con el porcentaje de cobertura que tenga el proyecto.



8. Preguntas frecuentes

8.1. ¿Cómo puedo cambiar mi contraseña en SonarQube?

Por motivos de seguridad se sugiere que después del primer ingreso a la herramienta la contraseña debe cambiarse.
Para cambiar la contraseña nos vamos a ir a My Account, este lo vamos a encontrar en la parte superior derecha del portal: (Ver **Imagen 9.1.0**).

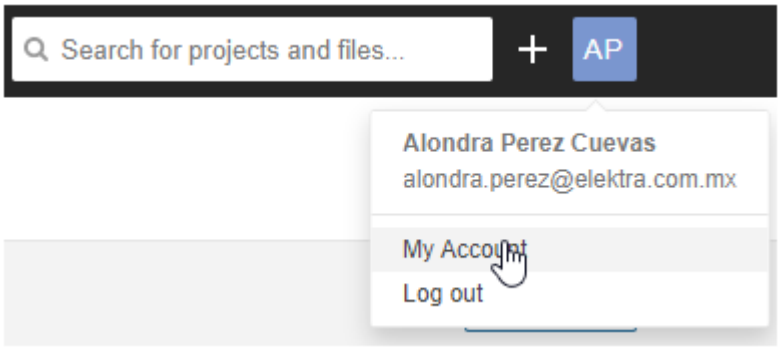


Imagen 9.1.0: Pestaña para ver mi cuenta.

Ahora daremos clic en el apartado Security y ahí se va a mostrar el formulario para cambiar la contraseña: (Ver **Imagen 9.1.1**).

La contraseña deberá ser mínimo de 8 caracteres y que sea fácil de recordar por el usuario.

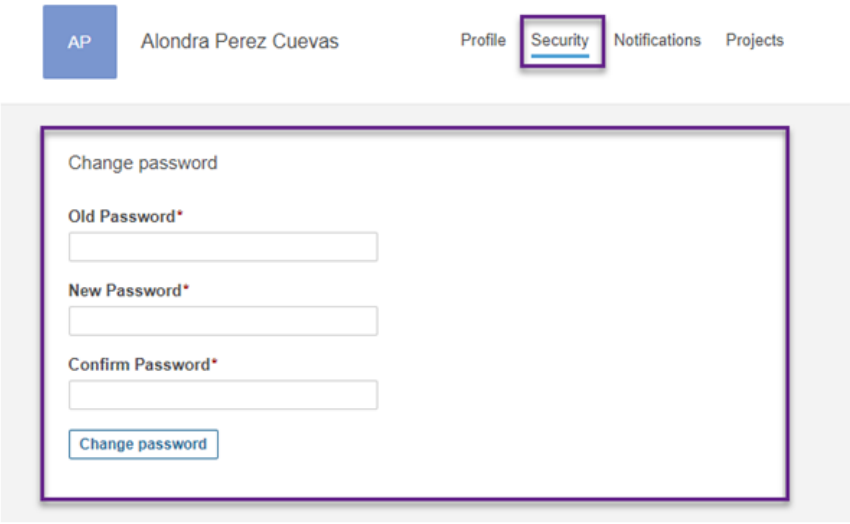


Imagen 9.1.1: Pestaña para cambiar mi contraseña

8.2. ¿Cuál es la vista del proyecto?

Cada proyecto dispone de una pantalla con el resumen del mismo, la cual se actualizará después de cada análisis. Desde esta vista podemos obtener información referente a los issues detectados por sonar. (Ver **Imagen 9.2.0**).

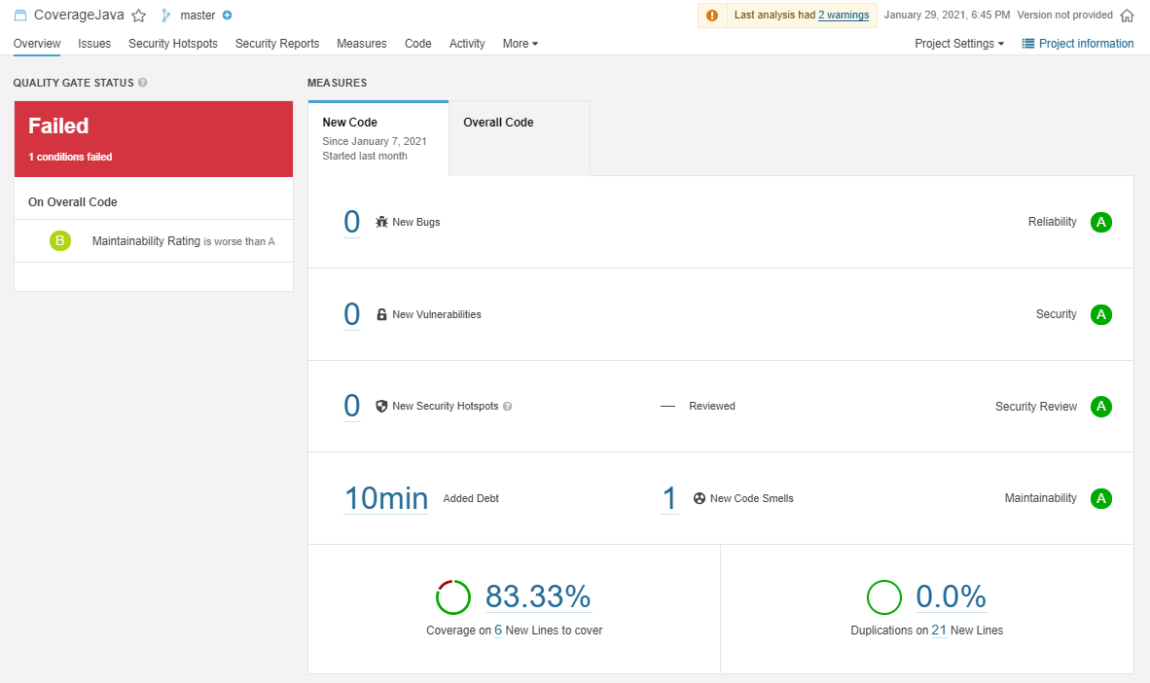


Imagen 9.2.0: Vista de un proyecto

8.3. ¿Cuál es la vista de issues?

Desde esta vista podemos gestionar y obtener más detalles de los issues pendientes. (Ver **Imagen 9.3.0**).

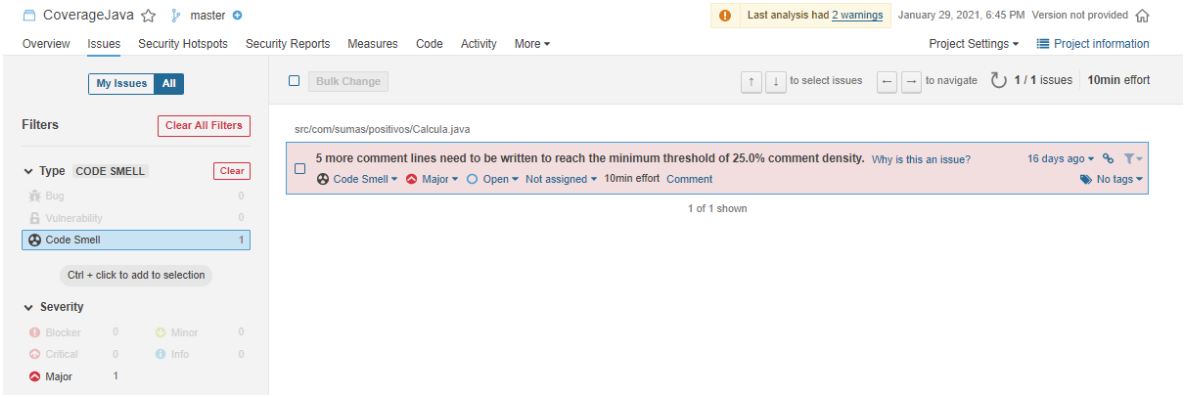


Imagen 9.3.0: Vista de Issues.

Pulsando en cada uno de ellos podemos ver la línea exacta donde se encuentra dicho issue, una breve descripción de la misma e incluso nos ofrece una posible solución para solventarla: (Ver **Imagen 9.3.1**).

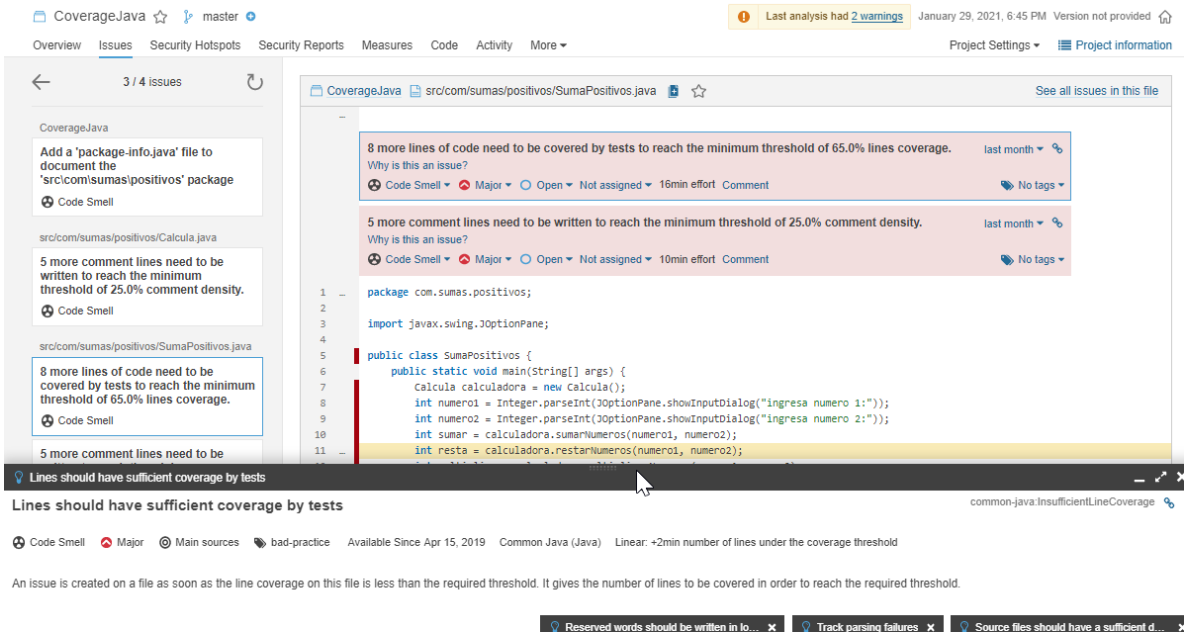


Imagen 9.3.1: Vista con detalle de Issues.

8.4. ¿Cuál es la vista de actividad?

Desde aquí podemos ver un historial de ejecuciones y sus resultados. Este apartado es muy útil para sacar métricas y podemos ir observando cómo van mejorando ciertos aspectos como: número de issues, cobertura de código, código duplicado, etc. (Ver **Imagen 9.4.0**).

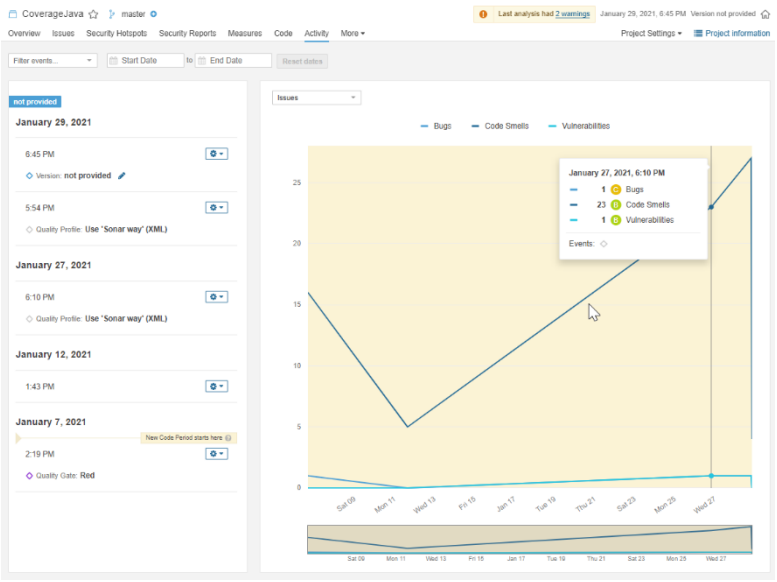


Imagen 9.4.0: Vista de historial de análisis.

8.5. ¿Cómo se obtiene el Project Key?

El ProjectKey es el identificador único de tu proyecto de sonar, para obtenerlo nos dirigimos al dashboard del proyecto y en la parte superior derecha se encuentra un menú de hamburguesa que dice **Project information**, ahí damos clic y se mostrará información general de nuestro proyecto, hasta abajo encontramos el ProjectKey. (Ver **Imagen 9.5.0**).

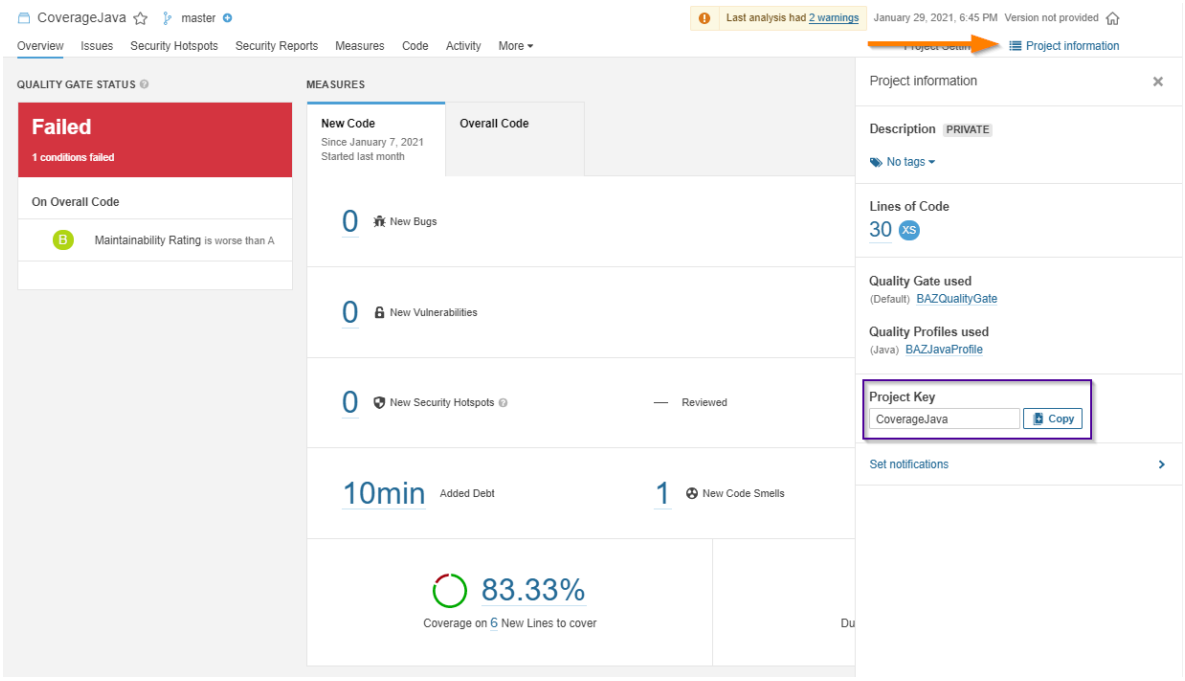


Imagen 9.5.0: Vista para obtener Project Key.

9. Reportes

9.1. Reporte ejecutivo

Este informe en PDF contiene la información más importante para tu proyecto en una sola página. Incluye tanto las métricas generales como las métricas sobre el código nuevo.
Este reporte lo vamos a encontrar en la vista **More → Reporting**: (Ver **Imagen 10.1.0**).

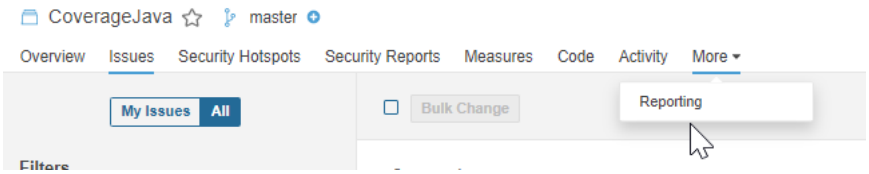


Imagen 10.1.0: Vista para generar un reporte.

Ahí se va a mostrar el informe y lo vamos a descargar: (Ver **Imagen 10.1.1**).

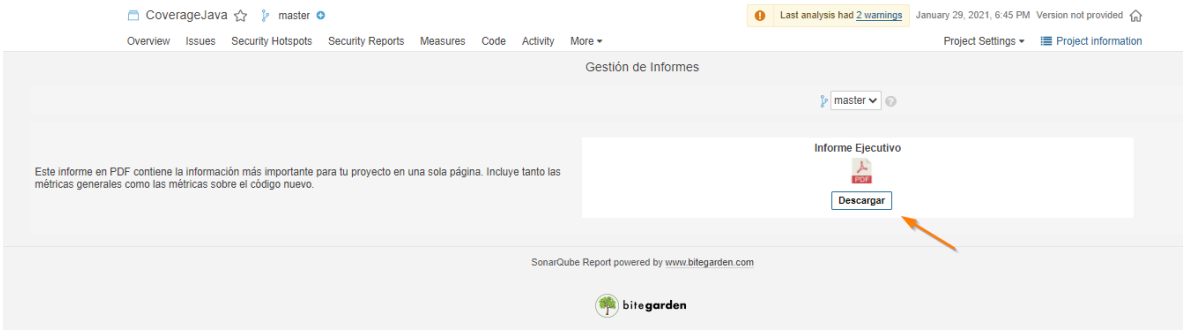


Imagen 10.1.1: Vista para generar un reporte.

Una vez que descarguemos dicho informe vamos a poder visualizar el contenido del mismo: (Ver **Imagen 10.1.2**).

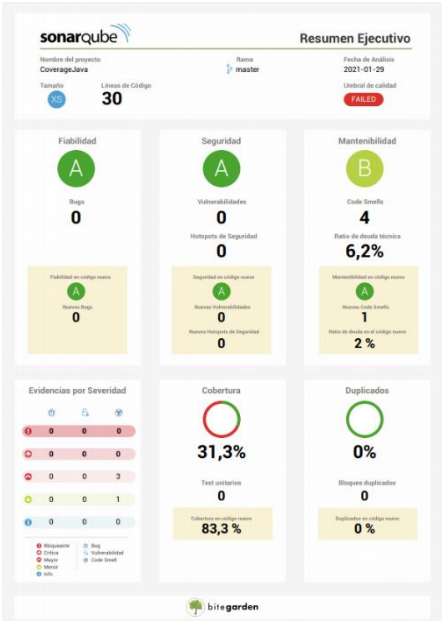


Imagen 10.1.2: Vista de un reporte.