Análisis Estático de Código Fuente

El análisis estático de software es un tipo de análisis de software que se realiza sin ejecutar el programa.

Algunos beneficios de este tipo de análisis son:

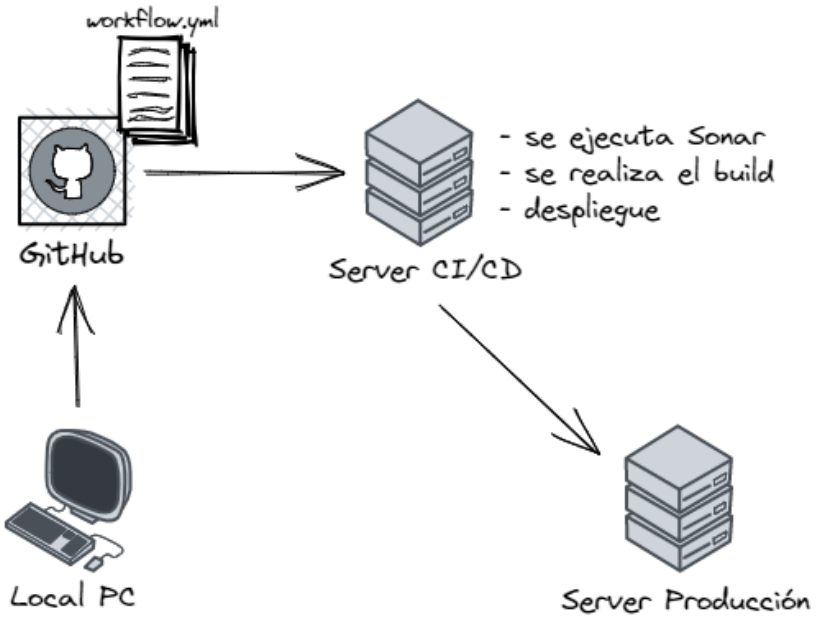
- Ayuda a la detección de bugs, vulnerabilidades, code smells.
- Mejorar la capacidad de mantenimiento del código.
- Posibilidad de identificar errores antes de realizar el despliegue a producción.

SonarQube

SonarQube es una herramienta para realizar análisis estático de código fuente. La misma nos sirve para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa. Permite a los equipos de desarrollo hacer seguimiento y detectar errores y vulnerabilidades de seguridad para mantener el código limpio.
SonarQube Incluye soporte para más de 20 lenguajes de programación, y detecta principalmente tres tipos de problemas:

- Bugs: error de codificación que rompe el código. debe corregirse de inmediato para poder ejecutar (esto ya debería ser detectado en el IDE, y no llegar hasta aquí).
- Vulnerabilidad: puntos de código abiertos a ataques o hacking.
- CodeSmells: fragmentos de código confuso que lo hacen difícil de mantener.

Un punto clave de esta herramientas es su integración con entornos de CI/CD (Jenkins, AWS, Github Actions, etc). Por ejemplo, si estamos usando un pipeline en Jenkins para construir, inspeccionar y desplegar el código, el paso de inspección continua, sería el que serviría de filtro para, si pasa el Quality Gate (o Umbrales de Calidad), desplegar, o, por el contrario, si no lo pasa, notificar del error y no desplegar hasta que se corrija, se suba de nuevo al repositorio de código, y empiece de nuevo el ciclo de despliegue continuo.



Quality Gate predeterminado de SonarQube

Coverage: Indica el porcentaje de código que está cubierto por pruebas unitarias. SonarQube establece en el Quality Gate predeterminado que la cobertura debe ser superior al 80%

Duplications: indica la cantidad de líneas que se repiten, en comparación con el total de líneas del proyecto. En este caso, se establece que este valor sea inferior al 3%.

Bugs: esta calificación se obtiene en relación a la severidad más alta de los bugs detectados. Para esta métrica, se determina que la calificación debe ser A, lo que significa que no haya ningún bug.

Vulnerabilities: se refiere también a la severidad más alta, pero de las vulnerabilidades encontradas. También debe ser A, lo que indica que no existiría ninguna vulnerabilidad en el código nuevo.

Hotspots Reviewed: son los fragmentos de código sensible a la seguridad que el desarrollador debe revisar. La principal diferencia entre un hotspot y una vulnerabilidad es la necesidad de una revisión antes de decidir si aplicar una corrección.

Code Smells: se mide el porcentaje de deuda técnica que existe en el código. La calificación debe ser A, qué vendría a ser que el porcentaje de deuda técnica es menor o igual al 5%.

Ejemplos de utilización de SonarQube

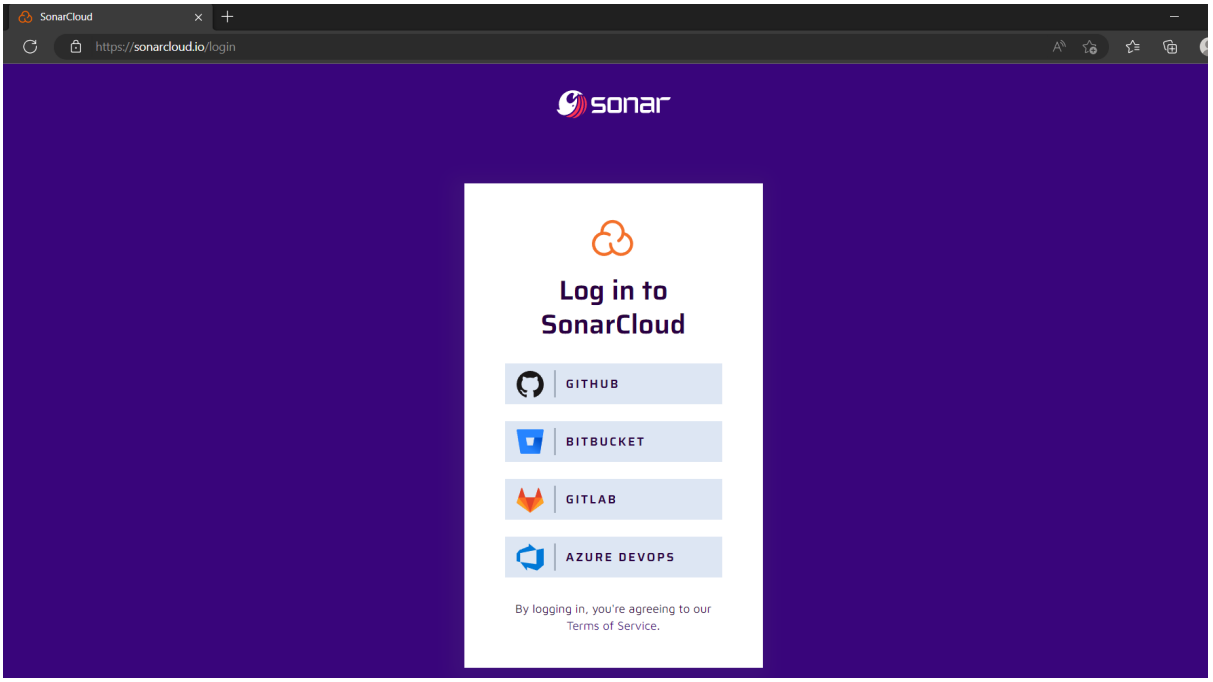
Para explicar y ejemplificar el funcionamiento de SonarQube analizaremos un desarrollo propio con archivos de ejemplo (test-project) con el objeto de ver, paso a paso, desde su instalación hasta un análisis general de sus funcionalidades principales.

Dos formas para utilizar SonarQube son:

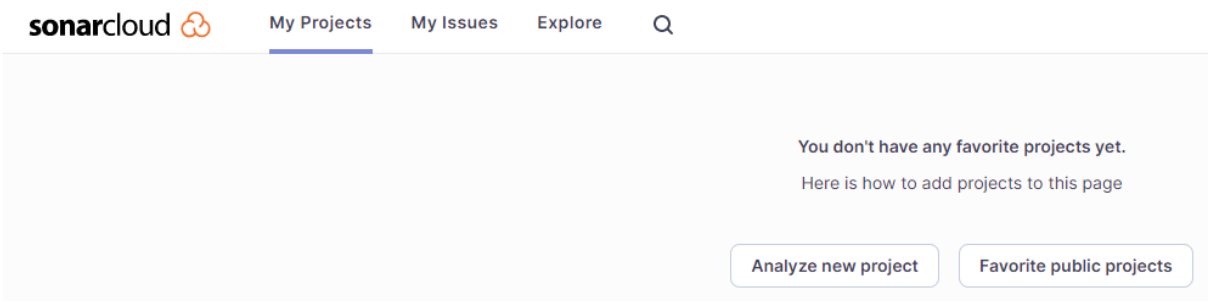
1. Desde SonarCloud (en este caso el código analizado es compartido en su plataforma a menos que se pague la versión premium).
2. Instalar SonarQube en un servidor propio.

SonarCloud online.

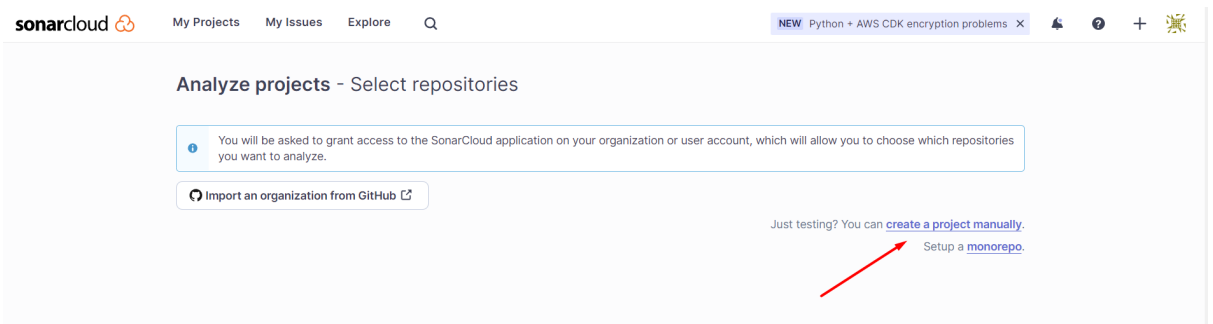
Dirigirse al sitio de SonarSloud (https://sonarcloud.io) y loguearse.



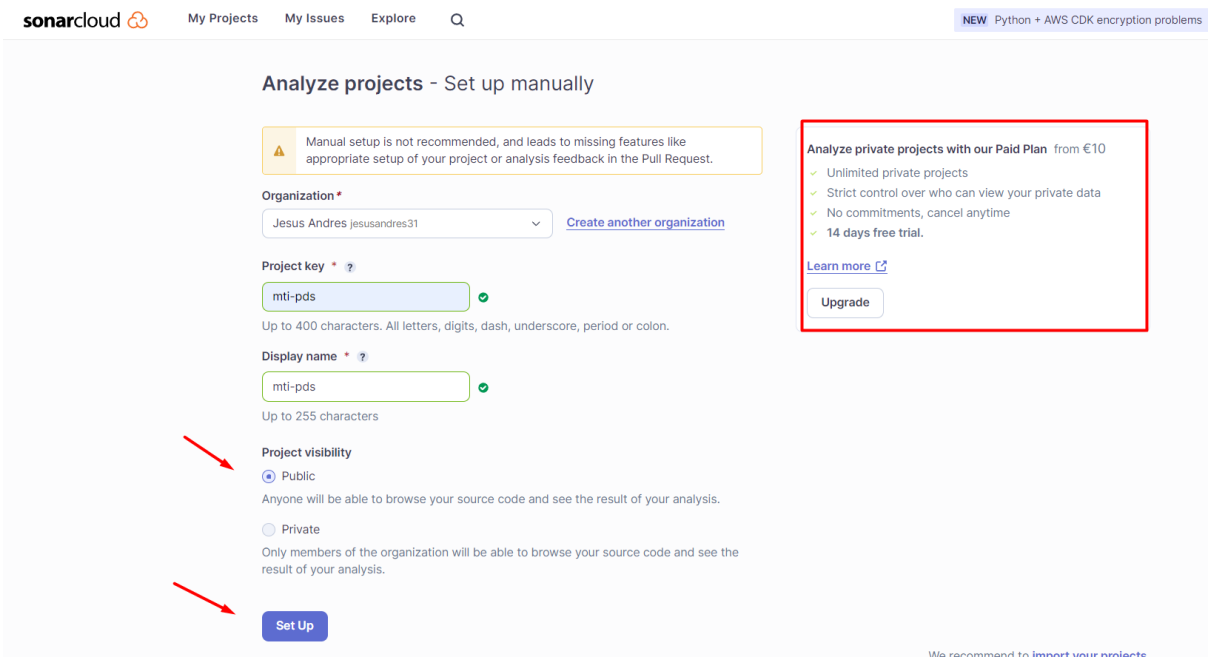
Dirigirse a la seccionar “My Projects” seleccionar “Analyze new project”:



Seleccionar “Crear proyecto manualmente”:

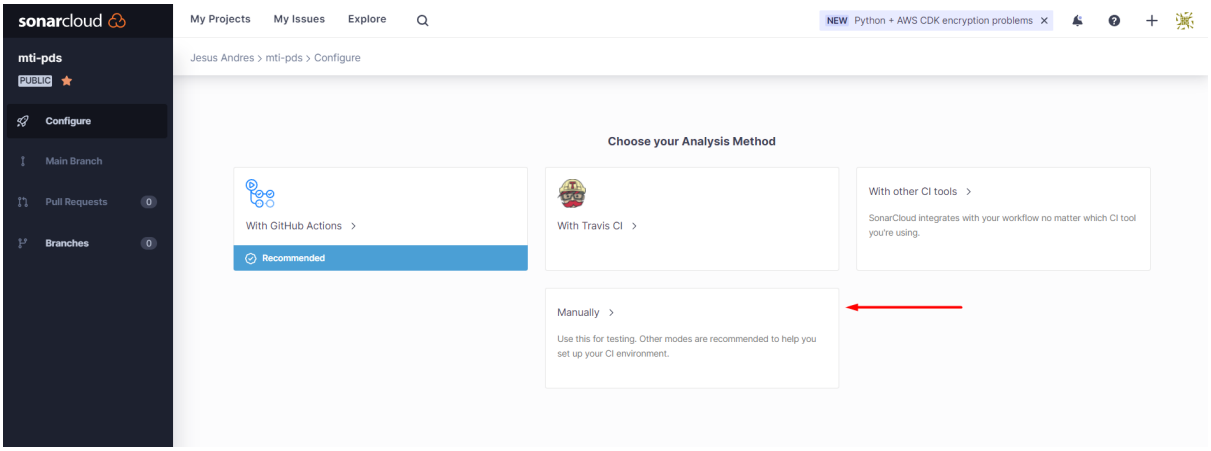


Cargamos los datos del formulario:



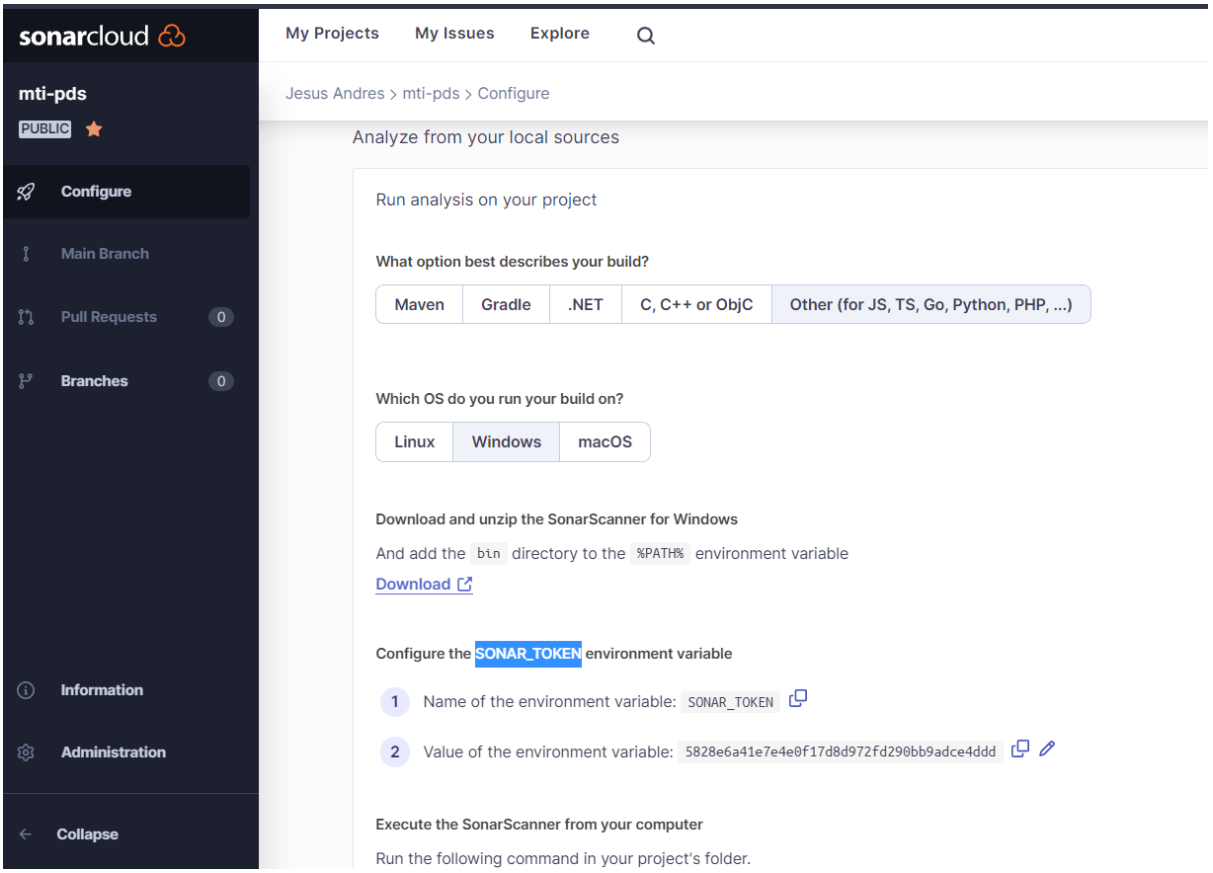
Como puede verse en Project visibility está inhabilitada la opción Private, con lo cual el código subido para analizar estará disponible para cualquiera que quiera ver los resultados.

Luego de pulsar el botón Set Up el formulario es enviado y la próxima pantalla que se muestra es la de configuración del proyecto dado de alta:

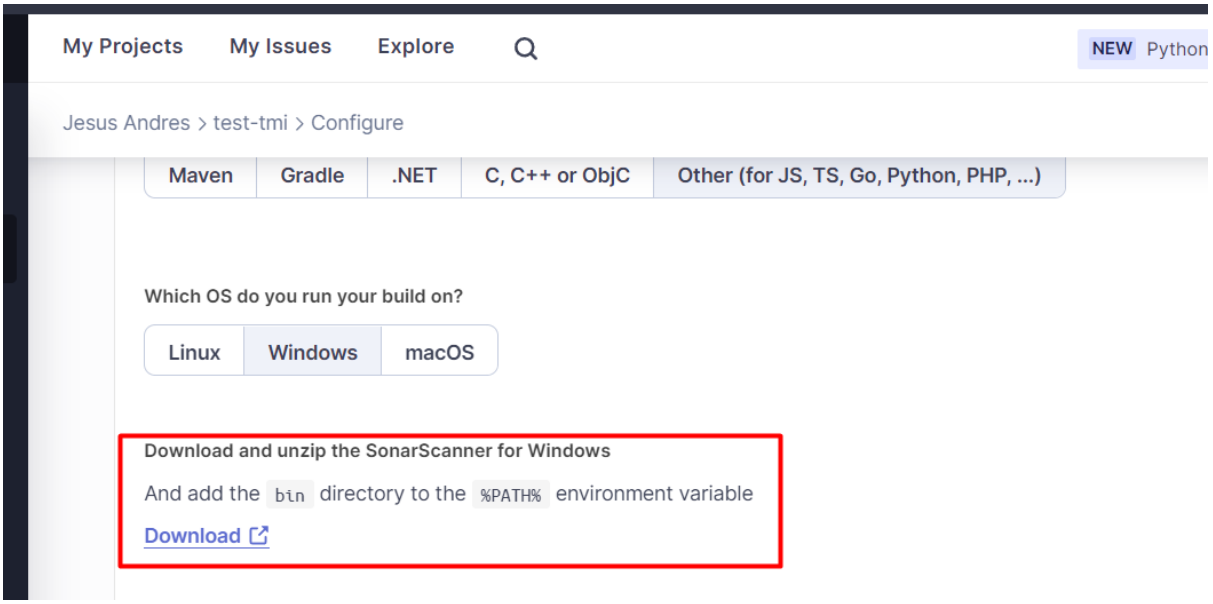


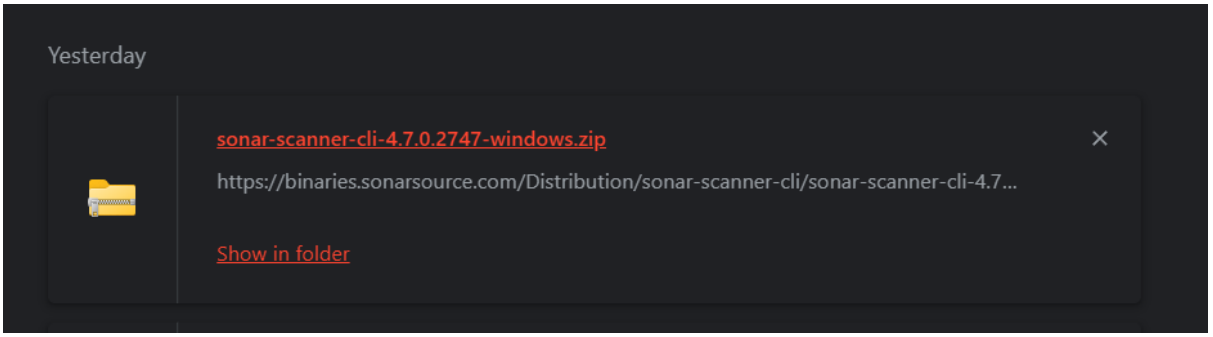
Como se mencionó antes SonarQube es muy utilizado en entornos de integración continua. Pero en este caso de prueba vamos a analizar el código manualmente.

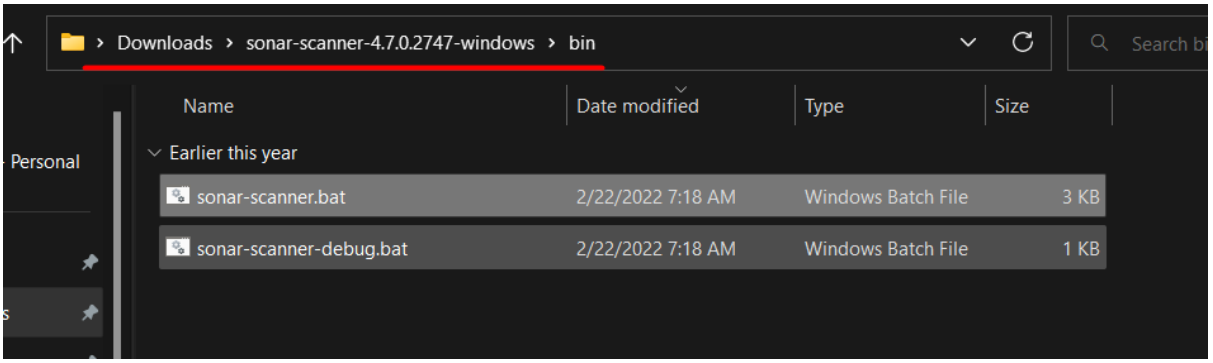
Seleccionamos la opción más acorde a nuestro lenguaje de programación y seleccionamos el sistema operativo donde se ejecutará SonarScanner.

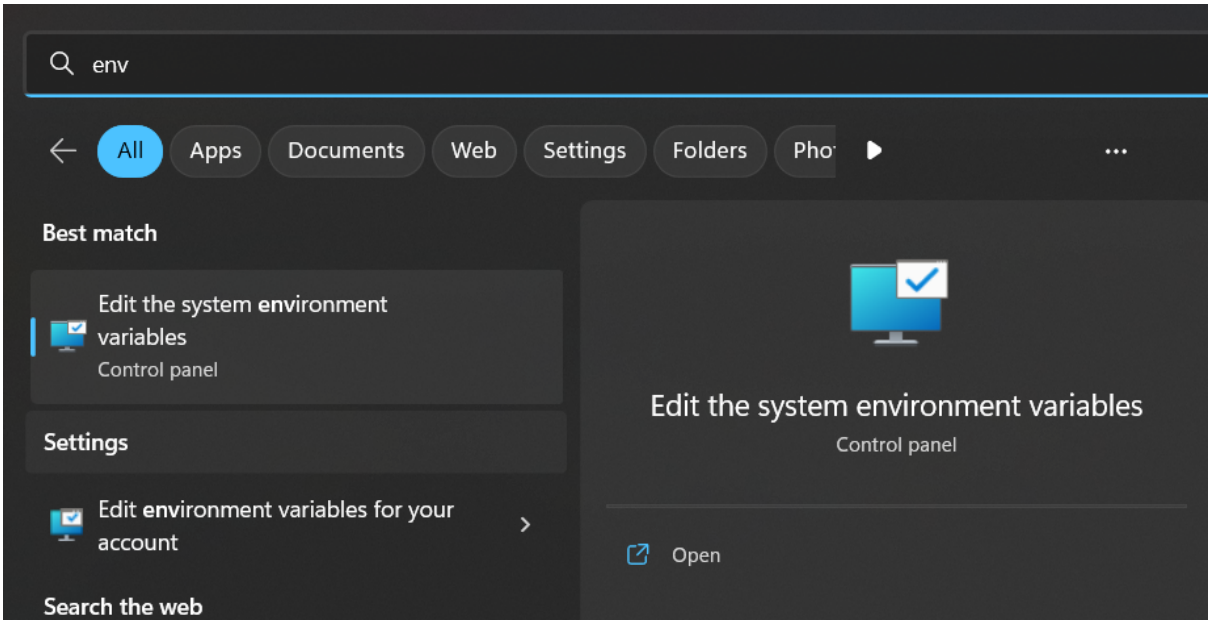


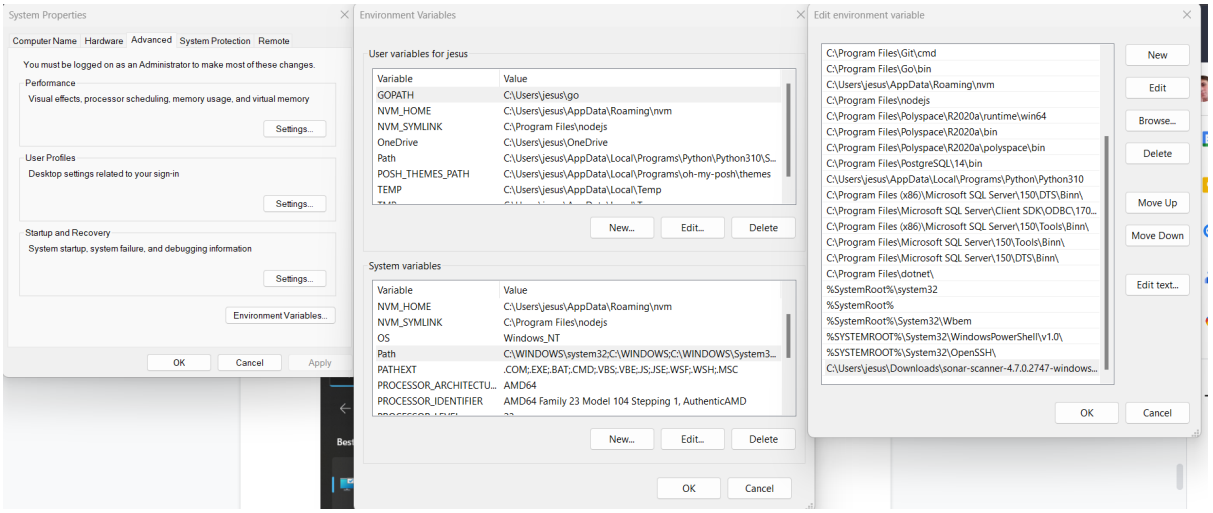
Descargamos SonarScanner y lo agregamos como variable de entorno.



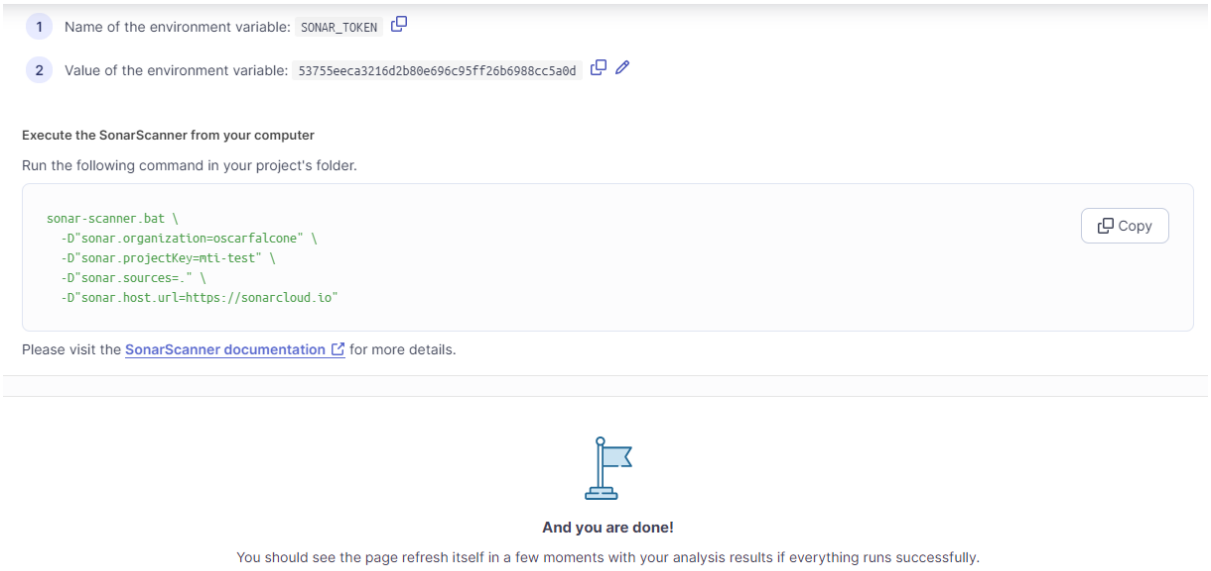

Yesterday
sonar-scanner-cli-4.7.0.2747-windows.zip
https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.7...
Show in folder

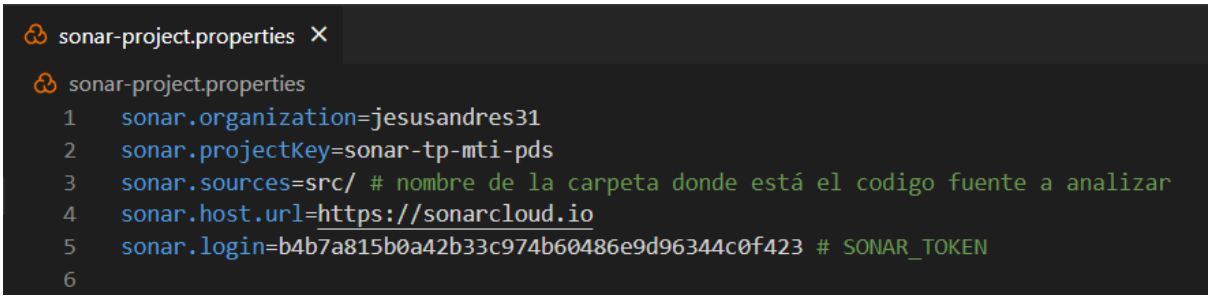

Downloads › sonar-scanner-4.7.0.2747-windows › bin
Search bi
Personal
Name
Date modified
Type
Size
Earlier this year
sonar-scanner.bat 2/22/2022 7:18 AM Windows Batch File 3 KB
sonar-scanner-debug.bat 2/22/2022 7:18 AM Windows Batch File 1 KB


env
All
Apps
Documents
Web
Settings
Folders
Best match
Edit the system environment variables
Control panel
Settings
Edit environment variables for your account
Search the web
Edit the system environment variables
Control panel
Open

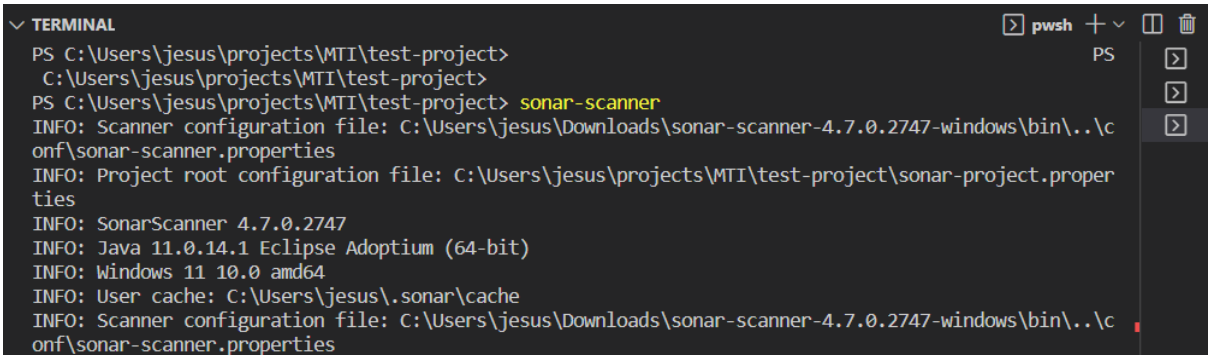

System Properties
Computer Name Hardware Advanced System Protection Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply
Environment Variables
User variables for jesus
Variable
Value
GOPATH C:\Users\jesus\go
NVM_HOME C:\Users\jesus\AppData\Roaming\nvm
NVM_SYMLINK C:\Program Files\nodejs
OneDrive C:\Users\jesus\OneDrive
Path C:\Users\jesus\AppData\Local\Programs\Python\Python310\S...
POSH_THEMES_PATH C:\Users\jesus\AppData\Local\Programs\oh-my-posh\themes
TEMP C:\Users\jesus\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
NVM_HOME C:\Users\jesus\AppData\Roaming\nvm
NVM_SYMLINK C:\Program Files\nodejs
OS Windows_NT
Path C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System3...
PATHEXT .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU... AMD64
PROCESSOR_IDENTIFIER AMD64 Family 23 Model 104 Stepping 1, AuthenticAMD
New...
Edit...
Delete
OK
Cancel
Edit environment variable
C:\Program Files\Git\cmd
C:\Program Files\Go\bin
C:\Users\jesus\AppData\Roaming\nvm
C:\Program Files\nodejs
C:\Program Files\Polyspace\R2020a\runtime\win64
C:\Program Files\Polyspace\R2020a\bin
C:\Program Files\Polyspace\R2020a\polyspace\bin
C:\Program Files\PostgreSQL\14\bin
C:\Users\jesus\AppData\Local\Programs\Python\Python310
C:\Program Files (x86)\Microsoft SQL Server\150\DTS\Binn\
C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\170...
C:\Program Files (x86)\Microsoft SQL Server\150\Tools\Binn\
C:\Program Files\Microsoft SQL Server\150\Tools\Binn\
C:\Program Files\Microsoft SQL Server\150\DTS\Binn\
C:\Program Files\dotnet\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Users\jesus\Downloads\sonar-scanner-4.7.0.2747-windows...
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel

Con los datos del SONAR_TOKEN y del comando autogenerado creamos un archivo llamado sonar-project.properties en la raíz de nuestro proyecto y copiamos esos datos.



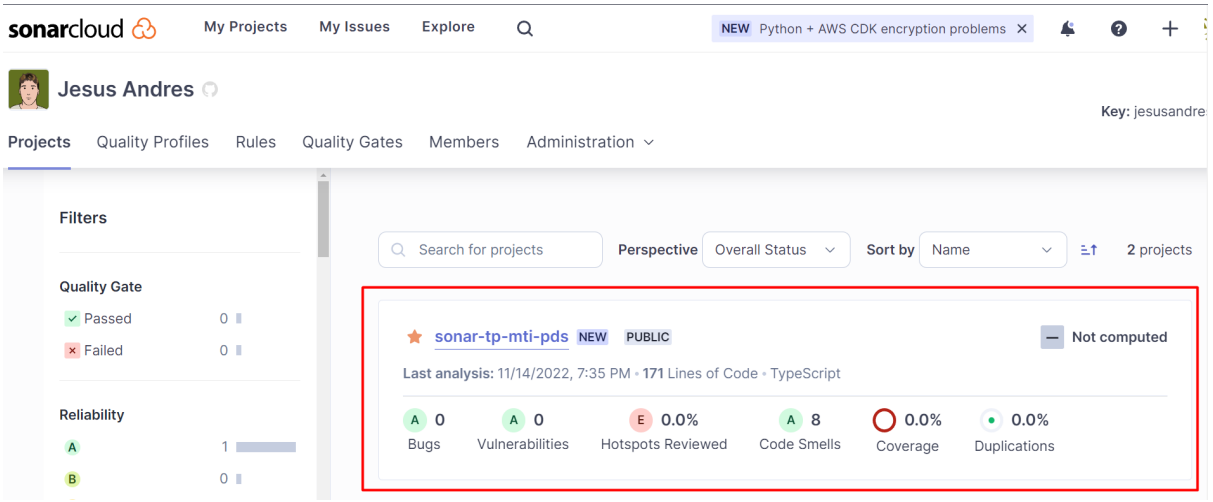


Ejecutamos sonar-scanner en la raíz de nuestro proyecto y esperamos a que termine de analizar.



```
TERMINAL                                                    pwsh
INFO: CPD Executor CPD calculation finished (done) | time=14ms
INFO: Analysis report generated in 436ms, dir size=218 KB
INFO: Analysis report compressed in 504ms, zip size=62 KB
INFO: Analysis report uploaded in 2981ms
INFO: ANALYSIS SUCCESSFUL, you can find the results at: https://sonarcloud.io/dashboard?id=sonar-tp-
mti-pds
INFO: Note that you will be able to access the updated dashboard once the server has processed the s
ubmitted analysis report
INFO: More about the report processing at https://sonarcloud.io/api/ce/task?id=AYR4SaDHxcnPmYz0Gd11
INFO: Analysis total time: 1:27.178 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 1:36.504s
INFO: Final Memory: 44M/154M
INFO: ------------------------------------------------------------------------
PS C:\Users\jesus\projects\MTI\test-project>
```

Podemos ver nuestro proyecto ya analizado en la interfaz de SonarCloud.

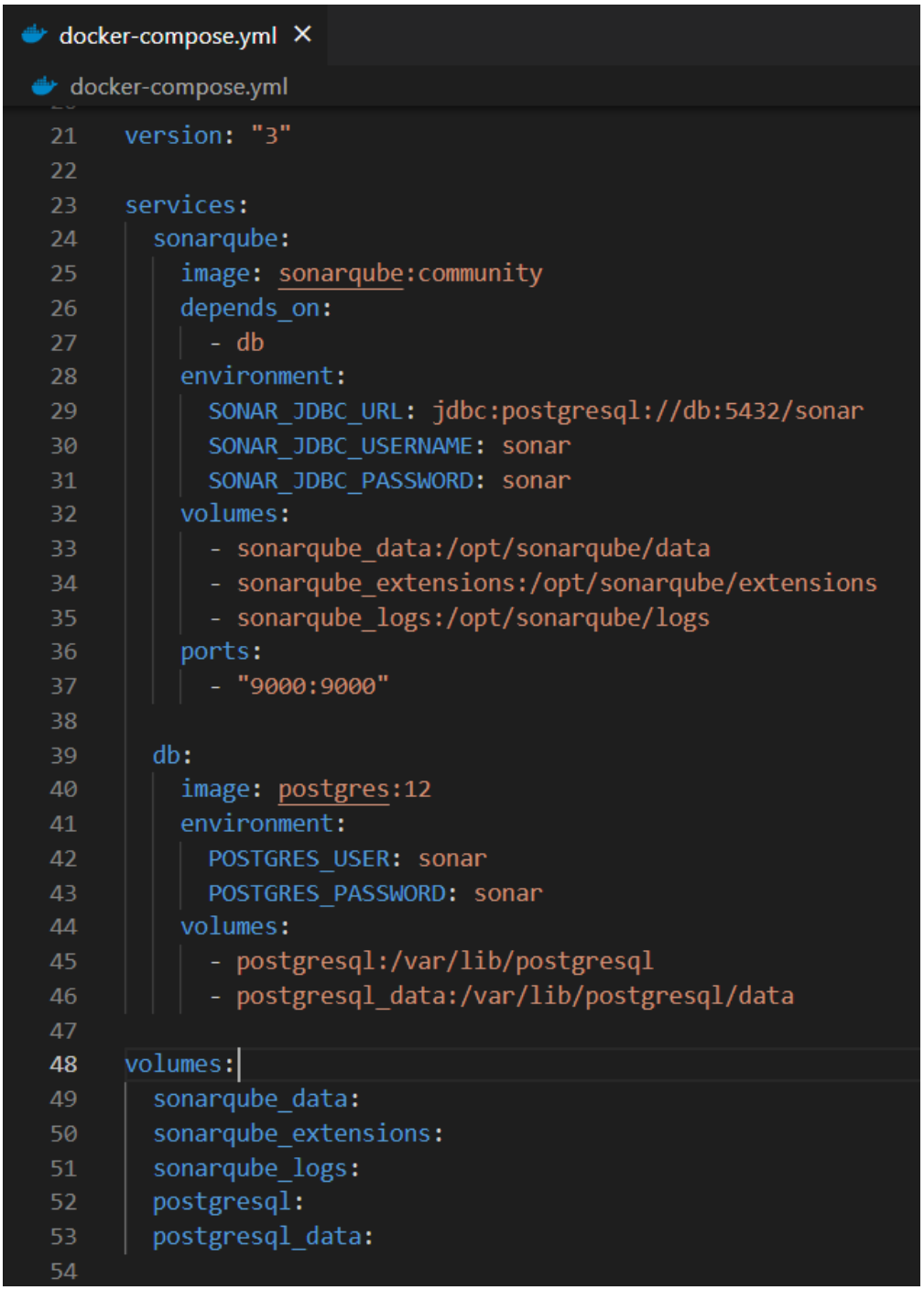


El análisis indica que se encontraron cero Bugs (calificación A) y cero Vulnerabilidades (calificación A).
Para los Hotspots Reviewed la calificación es E y está 0% cubierto, pero eso es debido a que no lo estamos usando (no hemos revisado ni un Hotspot). De cualquier forma, esto no es un impedimento para que el código analizado pase la prueba a menos que se configure lo contrario.
Se encontraron 8 Code Smells a ser factorizados.
El código está cubierto por tests en un 0% (no hemos creado tests para este código) y tiene 0% de duplicación de código.

SonarQube en un servidor

En este caso usaremos nuestra pc personal como servidor, pero lo ideal es instalar la herramienta en un servidor remoto real.

Preparamos un archivo docker-compose.yml que contiene la imagen oficial de SonarQube y una base de datos PostgreSQL. (https://github.com/jesusandres31/sonar-qube-docker)

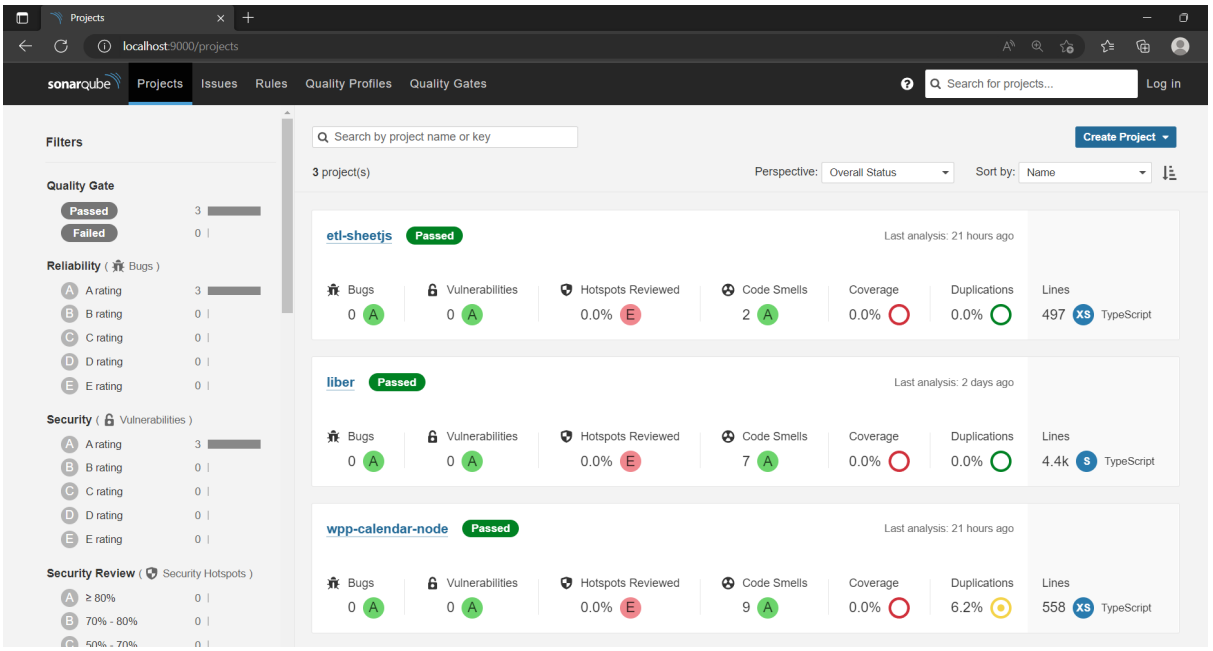


Levantamos los contenedores.

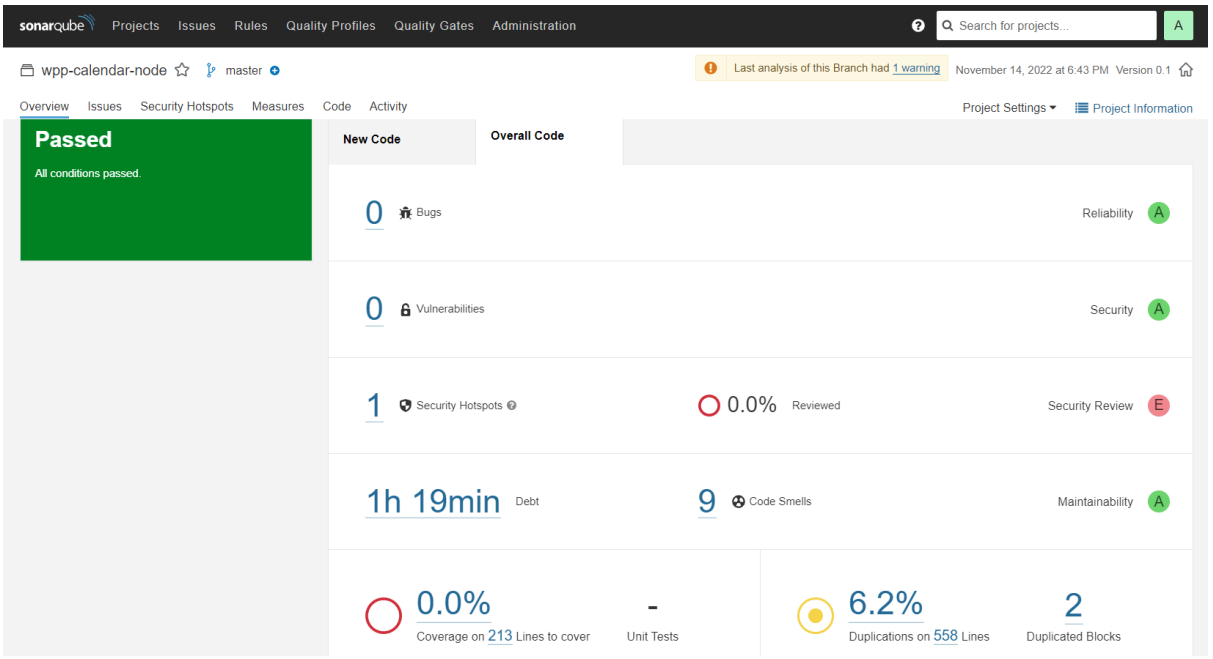
```
EXPLORER
TERMINAL                                                                    pwsh

PS C:\Users\jesus\projects\SonarQube>
docker-compose up -d
Starting sonarqube_db_1 ... done
Starting sonarqube_sonarqube_1 ... done
PS C:\Users\jesus\projects\SonarQube> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                    NAMES
4bea408fec03   sonarqube:community  "/opt/sonarqube/bin/…"   5 minutes ago    Up 2 seconds    0.0.0.0:9000->9000/tcp   sonarqube_sonarqube_1
20624954ee65   postgres:12          "docker-entrypoint.s…"   5 minutes ago    Up 3 seconds    5432/tcp                 sonarqube_db_1
```

Accedemos al servidor de SonarQube.

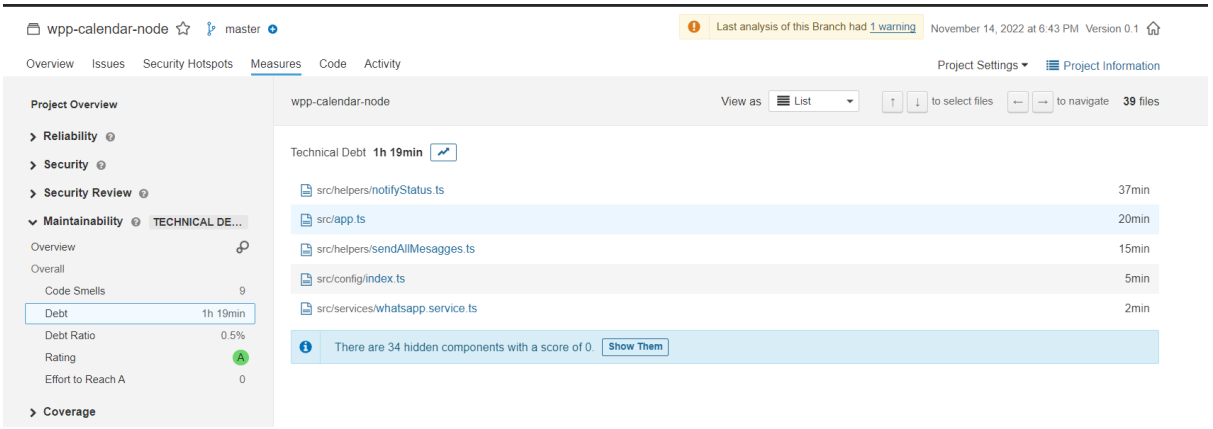


Podemos ver listados nuestros proyectos ya analizados o crear nuevos y analizarlos con SonnarScaner tal cual se mostró en la secuencia de pasos anterior.

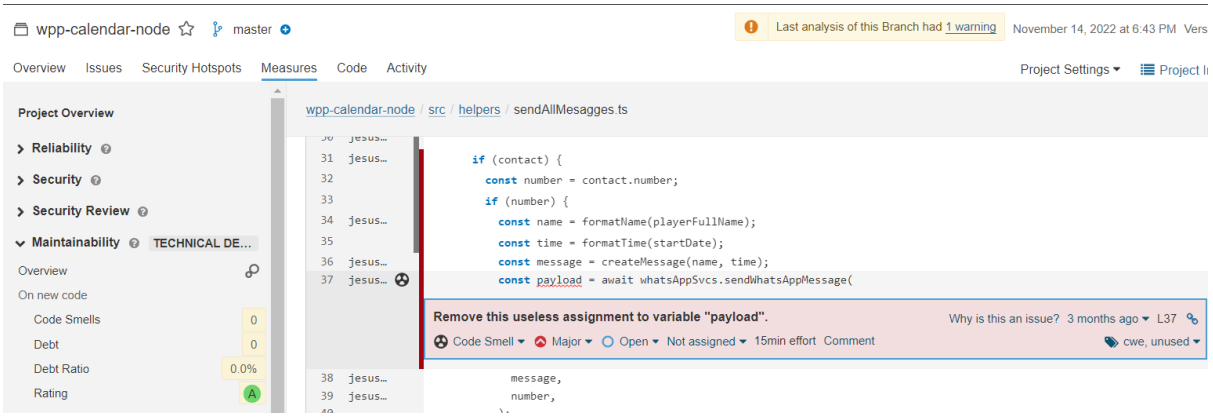


Si ingresamos a un proyecto vemos como Sonar lista los issues de nuestro código, y hasta estima cuánto tiempo vamos a tardar en solucionar estos issues mediante refactorización.

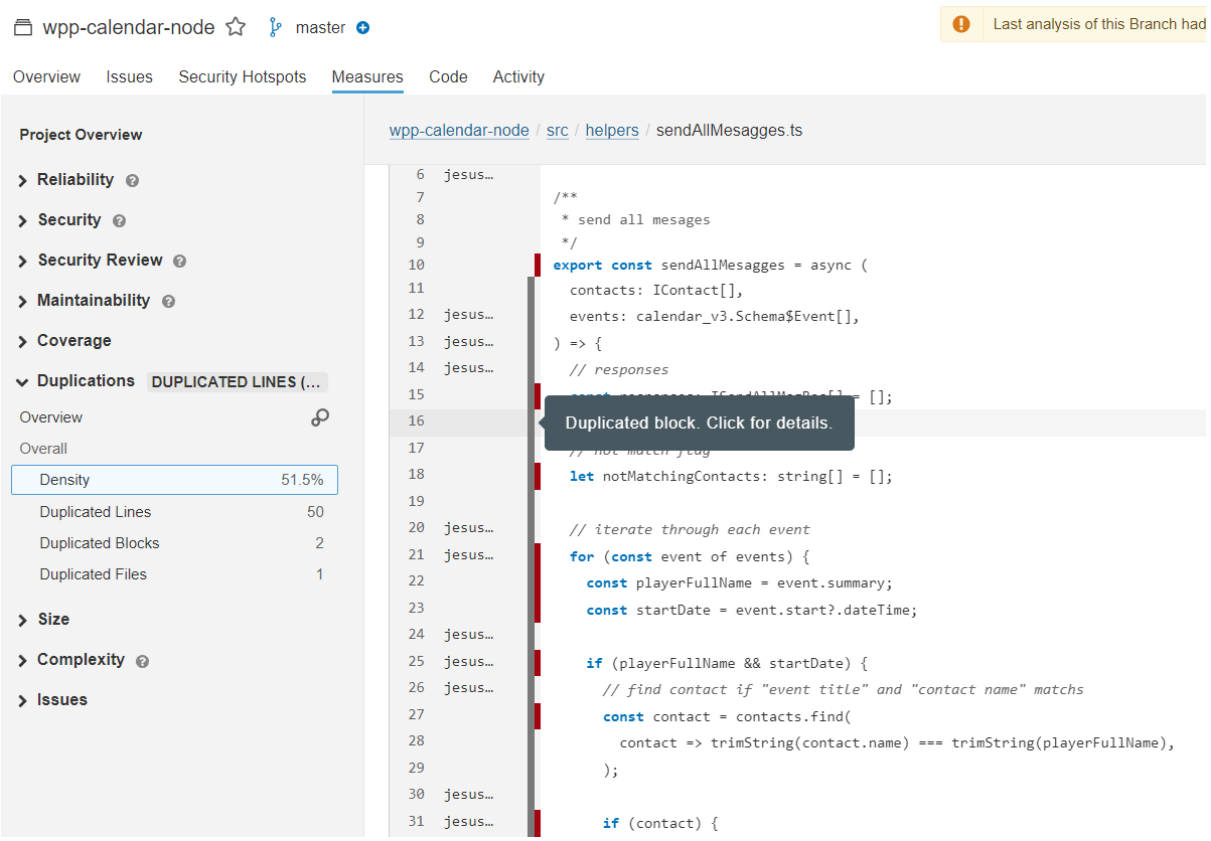
Si ingresamos por ejemplo a la sección de code smells podemos ver aquí listados los mismos.



Como ejemplo de un code smell, aquí Sonar nos sugiere remover esta asignación inútil de variable.



En este otro caso nos marca con una línea gris el código duplicado que podría ser refactorizado por ejemplo extrayendo ese código en una función; y que se utilice esa función en todos los lugares necesarios.


wpp-calendar-node
master
Last analysis of this Branch had
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Overview
Reliability
Security
Security Review
Maintainability
Coverage
Duplications
DUPLICATED LINES (...
Overview
Overall
Density 51.5%
Duplicated Lines 50
Duplicated Blocks 2
Duplicated Files 1
Size
Complexity
Issues
wpp-calendar-node / src / helpers / sendAllMesagges.ts
Duplicated block. Click for details.
/**
* send all mesages
*/
export const sendAllMesagges = async (
contacts: IContact[],
events: calendar_v3.Schema$Event[],
) => {
// responses
let notMatchingContacts: string[] = [];
// iterate through each event
for (const event of events) {
const playerFullName = event.summary;
const startDate = event.start?.dateTime;
if (playerFullName && startDate) {
// find contact if "event title" and "contact name" matchs
const contact = contacts.find(
contact => trimString(contact.name) === trimString(playerFullName),
);
if (contact) {

Lint

Cabe también mencionar la existencia de los Lints. Esta es una herramienta (plugin para el IDE) para el análisis estático de código en tiempo de codificación. Detecta errores de programación (vulnerabilidades y code smells) que escapan al habitual análisis sintáctico que hace el compilador.

Utilizando un lint podemos detectar en tiempo de desarrollo:

- Usos de variables antes de ser inicializadas o creadas.
- Variables/objetos instanciadas pero nunca utilizadas.
- Condiciones que no varían (siempre verdaderas o siempre falsas).
- Uso de diversas "malas prácticas".

La utilización de un lint en nuestro entorno de desarrollo nos ayuda a detectar este tipo de vulnerabilidades y code smells incluso antes de que estos lleguen a SonarQube.
Algunos ejemplos de estas herramientas son SonarLint (soporte para varios lenguajes) o ESLint (typescript o javascript).