ESPE UNIVERSIDAD DE LAS FUERZAS ARMADAS INNOVACIÓN PARA LA EXCELENCIA	INFORME DE PRÁCTICA DE LABORATORIO

DEPARTAMENTO:	Ciencias de la Computación	**CARRERA:**	Software		
ASIGNATURA:	Pruebas de software	**NIVEL:**	6to.	**FECHA:**	10/05/2025
DOCENTE:	Ing. Luis Castillo	**PRÁCTICA N°:**	1	**CALIFICACIÓN:**	

Evaluación de software mediante SonarQube

Ruben Benavides

RESUMEN

Durante esta práctica de laboratorio se instaló y configuró SonarQube utilizando un contenedor Docker para evaluar la calidad de distintos proyectos de software. Se analizaron aplicaciones desarrolladas en Java, Python, PHP y Angular, identificando bugs, vulnerabilidades, problemas de mantenibilidad y duplicación de código. Tras un análisis inicial, se aplicaron acciones de refactorización basadas en los reportes generados, lo que permitió reducir significativamente la cantidad de incidencias detectadas. Este ejercicio permitió familiarizarse con el uso práctico de SonarQube, comprender sus características clave, como seguridad, fiabilidad, mantenibilidad y duplicación, y validar su utilidad como herramienta de apoyo en la mejora continua del desarrollo de software.

Palabras Claves: SonaQube, análisis, refactorización.

1. INTRODUCCIÓN:

En esta actividad, se manejó la herramienta SonarQube, la cual nos permite evaluar la calidad del código de un proyecto. Esta es una herramienta de revisión automática de código para detectar bugs, vulnerabilidades y code smells, además realiza revisiones de código duplicado y podemos obtener la deuda técnica del mismo.

2. OBJETIVO(S):

2.1 Instalar la herramienta SonarQube.
2.2 Manejar la herramienta para evaluar proyectos de software.
2.3 Refactorizar el código para mejorar eliminar bugs y vulnerabilidades de impacto.

3. MARCO TEÓRICO:

SonarQube es una herramienta de código abierto que se dedica al aseguramiento de la calidad del código. Permite detectar errores, inseguridades y code smells en distintos y variados lenguajes de programación. Bajo esta ventaja, funciona como un entorno de desarrollo para que los equipos de software mantengan estándares de calidad y seguridad mediante el análisis automático y eficiente de proyectos, generando informes que destacan problemas críticos y sugerencias de mejora. Además, SonarQube tiene integraciones con herramientas como Jenkins, GitHub o Azure DevOps.

Para desplegar un servidor de SonarQube, el cuál permite aprovechar su funciones, existen distintas opciones: Docker, Kuberneters y de forma local con un ZIP. Docker es una plataforma que permite construir, compartir, ejecutar y verificar aplicaciones o servicios en cualquier lugar, sin la necesidad de realizar configuraciones desde cero. A través de las llamadas imágenes, que posee las configuraciones necesarias para correr las aplicaciones. Entonces, Docker crea contenedores en la máquina host del usuario que son las que se ejecutan de manera segura, rápida y totalmente equipadas con lo necesario.

Docker sirve, entonces, para realizar testing, entornos de desarrollo, o incluso desplegar aplicaciones para producción, lo que lo ha llevado a integrarse en distintas plataformas, entornos y aplicaciones como Visual Studio Code, CircleCI y GitHub. Docker se puede utilizar mediante la interfaz de comandos de línea o con la interfaz gráfica

Docker Desktop. Dependiendo de las imágenes, al momento de ejecutar un contenedor, se pueden proveer diferentes configuraciones según la aplicación y su naturaleza, sin embargo, una configuración esencial de todos es que a los contenedores se les puede indicar específicamente el puerto donde se aloja en la máquina host la aplicación.

El uso de la imagen de sonarqube requiere de un sistema operativo Linux (x64, AArch64), Windows (x64) o macOS (x64, AArch64), RAM de mínimo 4GB, procesador de mínimo 64 bits con arquitectura amd64 o Apple SIlicon arm64 y 30GB en disco. Además, en software requiere de uno de los siguientes navegadores: Edge, Firefox, Chrome o Safari, y de Java, específicamente Oracle JRE o un OpenJDK y Java 17 en adelante. Java es un lenguaje de programación y el JDK es un software que incluye el intérprete Java, clases Java y herramientas de desarrollo Java. Finalmente, la imagen de SonarQube no es suficiente para empezar a evaluar proyectos de software. SonarQube utiliza SonnarScanner CLI para ejecutarse.

4. DESCRIPCIÓN DEL PROCEDIMIENTO:

Para la práctica se utilizaron los requerimientos recomendados para una instalación de escala más grande, ya que los requisitos mínimos son solo para instalaciones pequeñas.

Reactivos

No aplica.

Insumos

- Una PC con Linux
- SonarQube
- JDK 24
- Acceso a Internet

Equipos

Ubuntu 20.04.2 LTS, Procesador AMD Ryzen™ 7 7435HS × 16, 16GB RAM, 1TB en disco, NVIDIA GeForce RTX™ 4050 Laptop GPU.

Procedimiento

Descargar e instalar SonarQube

1. Se instaló la imagen oficial de docker mediante **docker pull sonarqube** en la terminal.
2. Se ejecutó el contenedor mediante **docker run --name sonarqube -p 9000:9000 sonarqube**.
3. Se accedió a la interfaz web proporcionada por sonarqube en **http://localhost:9000**
4. Se cambió la contraseña predeterminada **admin** por una nueva.
5. Se revisó la plataforma que se puede observar en la Figura 1.

Configurar SonarScanner

Para proyectos de categoría Other

1. Se descargó el zip que contiene los archivos para ejecutar SonarScanner.
2. Se descomprimió el zip.
3. Se añadió la carpeta bin a las variables de entorno.

Para proyectos de categoría JS/TS & Web

1. Se descargó el paquete @sonar\scan para npm mediante npm install -g @sonar/scan
2. Posteriormente, se utilizó el comando proporcionado por sonarqube al momento de realizar el análisis.

Análisis de Proyectos

1. Se crearon 4 proyectos para evaluar. La creación de un proyecto que analiza las aplicaciones se puede visualizar en las figuras 2, 3, 4 y 5. El proceso consiste en crear un nuevo proyecto, darle un nombre, una clave que sirve para generar tokens, y la rama que debe ser analizada. Posterior a eso, se debe configurar

la línea de base en la que el scaner va a ejecutarse, es decir, todo el código o a partir de cierto punto. Luego, se puede hacer diferentes tipos de análisis
   a. CMU - Proyecto desarrollado en HTML, CSS, Javascript y php.
   b. ActCont - Proyecto desarrollado en Java con Springboot y JavaFX.
   c. gogoApp - Proyecto desarrollado en Angular (Typescript y HTML).
   d. ajedrez - Proyecto desarrollado en Python.
2. Se ejecutó el comando dado por la herramienta SonarQube para el análisis del proyecto según su lenguaje y entorno de construcción como se puede visualizar en la figura 6.
3. Se revisó y se solucionó posibles problemas al momento de ejecutar un análisis exitoso.

Interpretación de los Resultados

Se ingresó a la plataforma y se revisó las vulnerabilidades, nivel de peligro y bugs encontrados.

CMUTest

Se aplicó un análisis para proyectos php como se puede observar en la figura 7, y los resultados indican que el código:

- No tiene ningún riesgo en la seguridad, presentando 0 incidencias.
- Tiene una fiabilidad con problemas de severidad media presentando 7 incidencias
- Tiene una mantenibilidad con severidad baja presentando 49 incidencias.
- Tiene un 13.5% de duplicación en el código.

ActCont

Se aplicó un análisis para proyectos Other como se puede observar en la figura 8, y los resultados indican que el código:

- No tiene ningún riesgo en la seguridad, presentando 0 incidencias.
- Tiene una fiabilidad con problemas de severidad alta presentando 36 incidencias
- Tiene una mantenibilidad con severidad baja presentando 299 incidencias.
- Tiene un 5.9% de duplicación en el código.
- Tiene puntos críticos de seguridad presentando 13 incidencias.

gogoApp

Se aplicó un análisis para proyectos JS/TS como se puede observar en la figura 9, y los resultados indican que el código:

- No tiene ningún riesgo en la seguridad, presentando 0 incidencias.
- Tiene una fiabilidad con problemas de severidad media presentando 19 incidencias
- Tiene una mantenibilidad con severidad baja presentando 36 incidencias.
- Tiene un 4.8% de duplicación en el código.

ajedrez

Se aplicó un análisis para proyectos Python como se puede observar en la figura 9, y los resultados indican que el código:

- No tiene ningún riesgo en la seguridad, presentando 0 incidencias.
- Tiene una fiabilidad con problemas de severidad alta presentando 1 incidencia.
- Tiene una mantenibilidad con severidad muy baja presentando 4 incidencias.

5. ANÁLISIS DE RESULTADOS:

Refactorización de código

En un análisis inicial, se obtuvo los datos que se pueden observar en la tabla 1, que representan las estadísticas anteriores a una refactorización de código presentando afecciones en la seguridad, mantenibilidad, duplicación y puntos críticos.

Proyecto	Características Afectadas	Incidencias	Problemas a solucionar
CMU	Fiabilidad	7	Algunos archivos html, dentro de la etiqueta html, no tenían especificado el valor del atributo lang. Esto es importante ya que describe el lenguaje por defecto del documento html.
	Mantenibilidad	49	Algunos identificadores css se repetían, esto afecta a la mantenibilidad ya que en caso de tener el mismo nombre, los navegadores dan preferencia al último, dificultando así cambios.
ActCont	Seguridad	1	La contraseña de la base de datos está expuesta directamente en el application.properties. Esto es una vulnerabilidad directa a la seguridad de la base de datos.
	Fiabilidad	36	Existe una estructura condicional que no realiza nada. Existe una unidad en un archivo css que no funciona. Existe un identificador css que no se reconoce. Estos problemas pueden generar un desentendimiento en el código.
	Puntos críticos de seguridad	13	Existen múltiples manejos de excepciones que imprimen toda la estructura e información del código, lo que compromete el código.
gogoApp	Fiabilidad	19	Una etiqueta <label> debe envolver un control o usar el atributo htmlFor para asociarse a uno. Si no tiene texto ni está vinculado a un control, puede causar mal entendimiento del código e interfaz confusas.
	Mantenibilidad	36	Existen archivos sin contenido y clases que nunca se reasignan, generando confusión al momento de modificar el código.
ajedrez	Fiabilidad	1	Una función que recibe 3 parámetros sólo está recibiendo 2, esto puede afectar en el entendimiento general del código.
	Mantenibilidad	4	Se está importando todos los módulos pertenecientes al paquete pygame.locals, esto genera una carga y poco mantenimiento al no ser capaz de utilizar solo lo necesario y específico.

Tabla 1. Análisis de resultados iniciales.

Tras una refactorización de código en los proyectos, se obtuvo los datos que se pueden observar en la tabla 2, que representan las nuevas estadísticas con una explicación acerca de lo realizado en los proyectos.

Proyecto	Características	Incidencias	Solución (Ver figuras 10, 11, 12 y 13)

	Afectadas		
CMU	Fiabilidad	5	Se especificó el valor ¨es¨ para el atributo lang de la etiqueta html en los archivos, logrando obtener una severidad de bajo impacto.
	Mantenibilidad	43	Se quitó los duplicados en los identificadores css y se unió en uno solo.
ActCont	Seguridad	1	Se utilizaron variables de entorno del sistema para no exponer las credenciales.
	Fiabilidad	30	Se eliminaron los bloques no implementados y se quitaron las unidades no reconocidas. Sin embargo, en lo que es identificadores no reconocidos, no es posible quitarlos, ya que esto es debido al uso de javafx, este paquete gráfico usa css, pero personalizado, y este si entiende sus identificadores.
	Puntos críticos de seguridad	4	Se intercambió la impresión de pila de error, por un mensaje personalizado que no expone directamente la estructura del código.
gogoApp	Fiabilidad	1	Se envolvieron los labels en inputs y también se les dio un valor de texto faltante.
	Mantenibilidad	28	Se eliminaron los archivos sin uso y se marcó algunas clases como **readonly** para manejar mejor su propósito.
ajedrez	Fiabilidad	0	Se añadió un parámetro por defecto en la función con el valor **None.**
	Mantenibilidad	3	Se importó solo los paquetes necesarios.

Tabla 2. Análisis de resultados tras refactorización de código.

6. GRÁFICOS O FOTOGRAFÍAS:

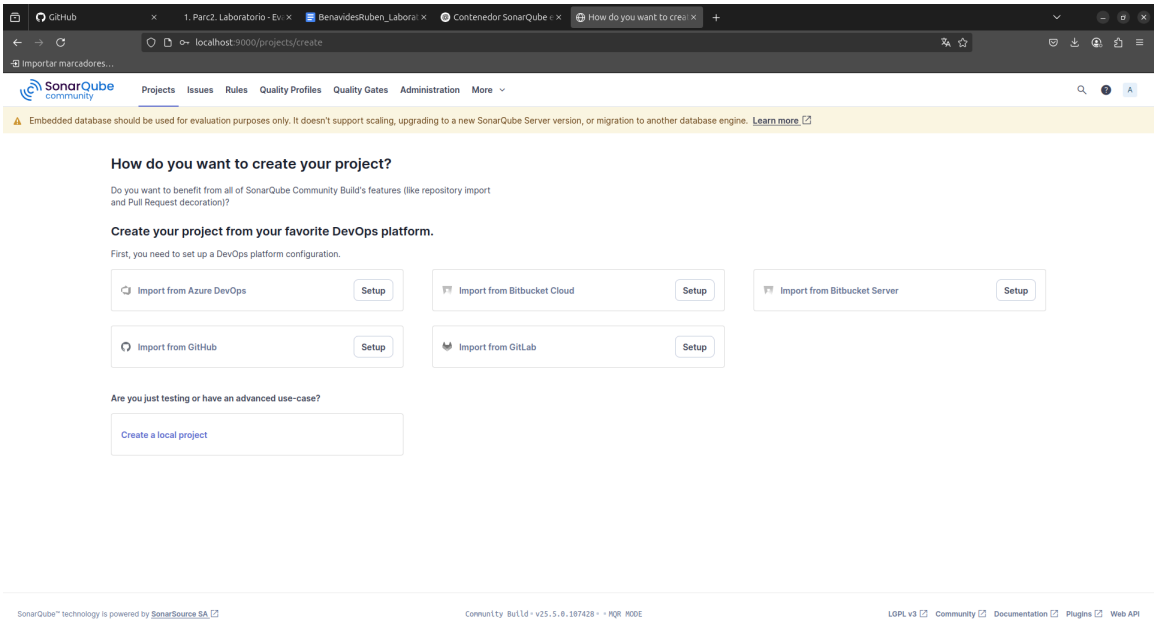

Fig. 1. Plataforma Sonarqube.

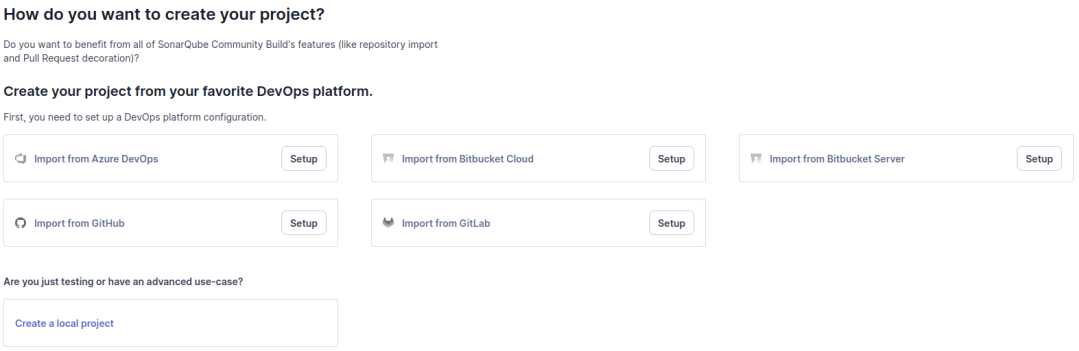

Fig. 2. Creación de un proyecto en SonarQube.

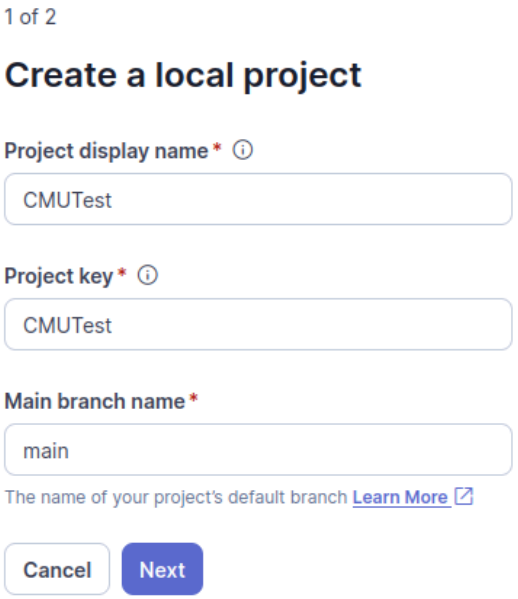

Fig. 3. Parámetros de creación de un proyecto en SonarQube.

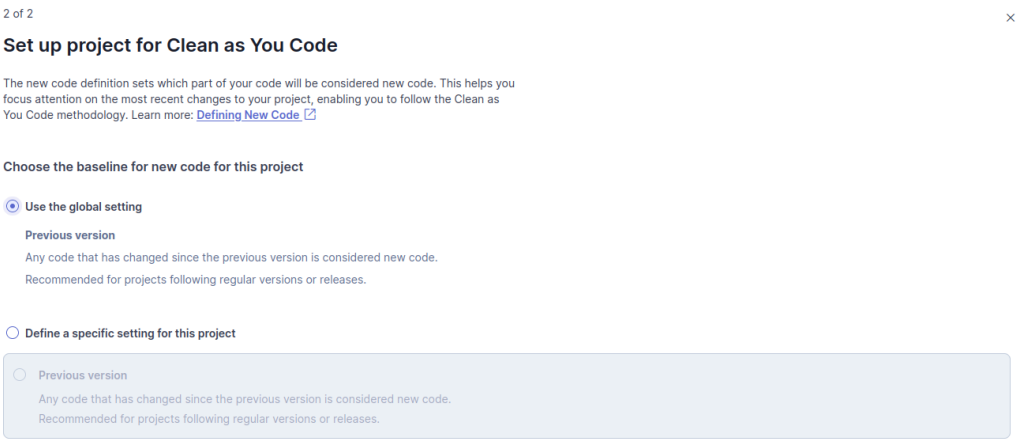

Fig. 4. Configuración de la línea de base de evaluación del proyecto.

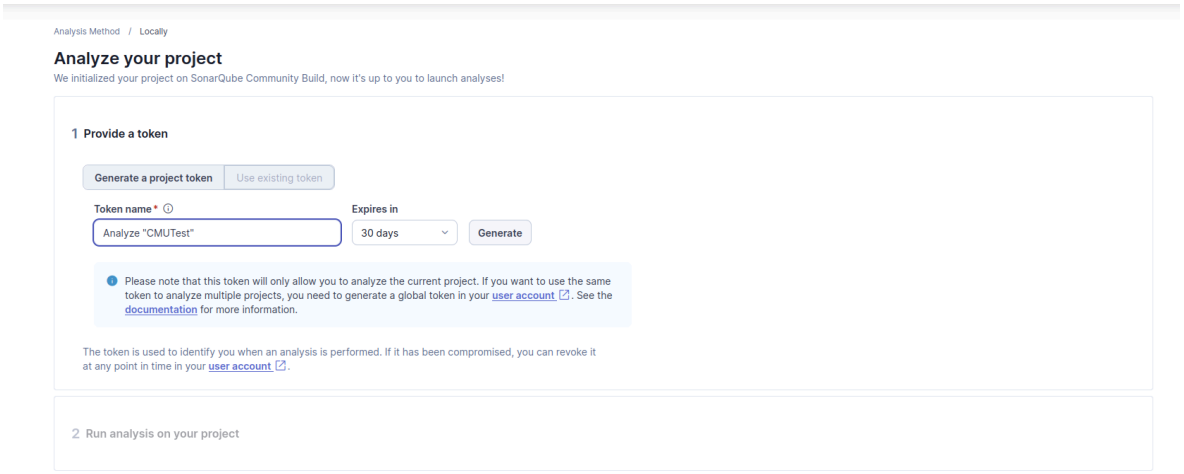

Fig. 5. Generación de token para analizar los proyectos.

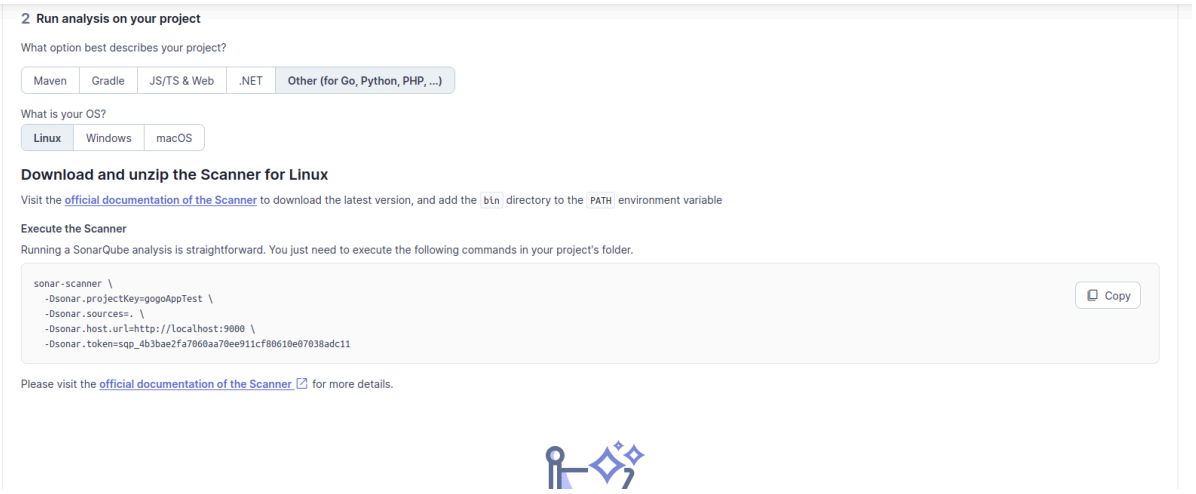

Fig. 6. Métodos de análisis.

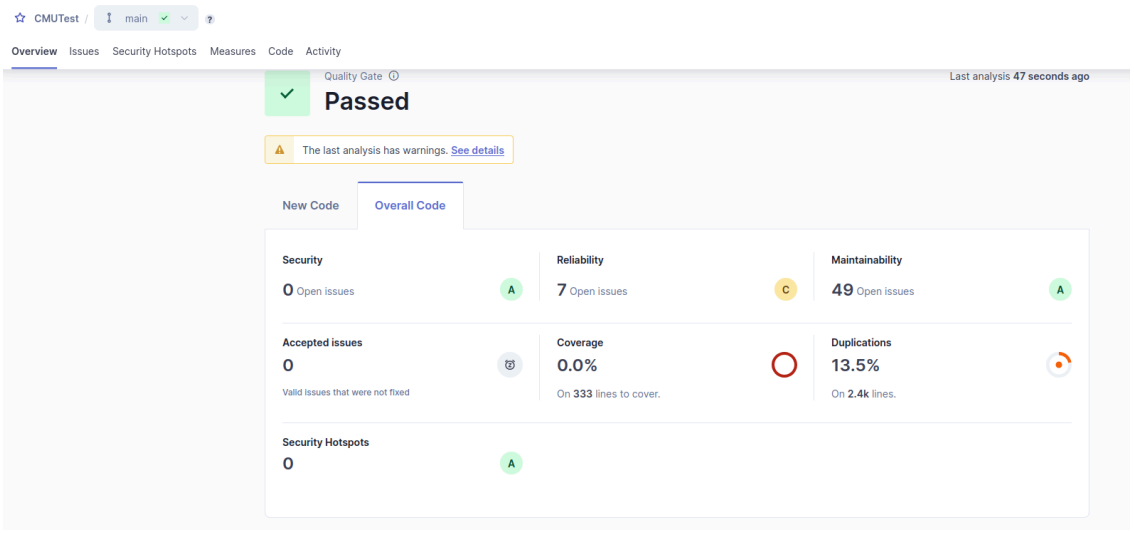

Fig. 7. Análisis inicial del proyecto CMU.

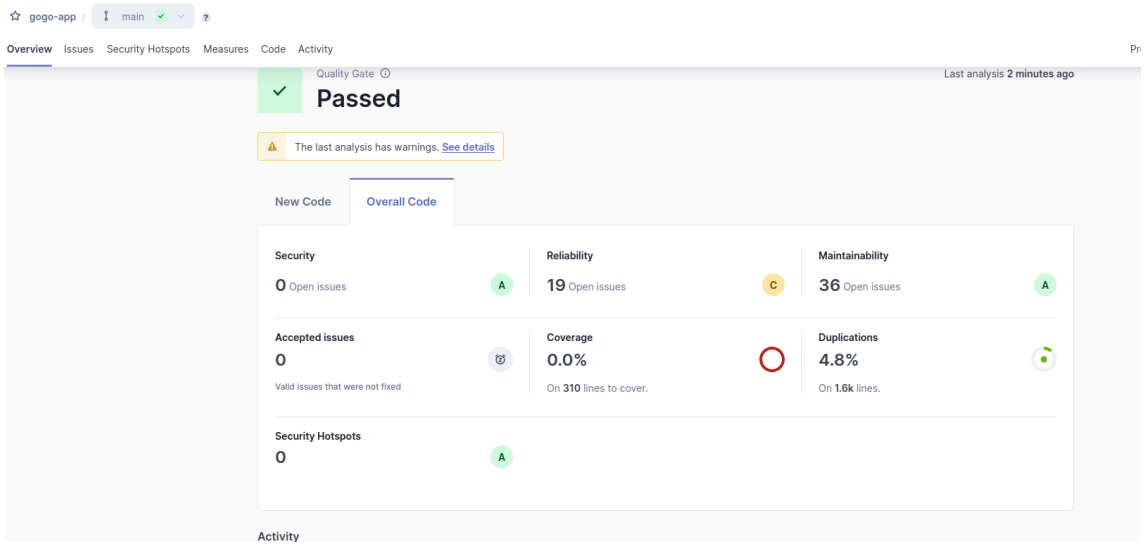

Fig. 9. Análisis inicial del proyecto gogoApp

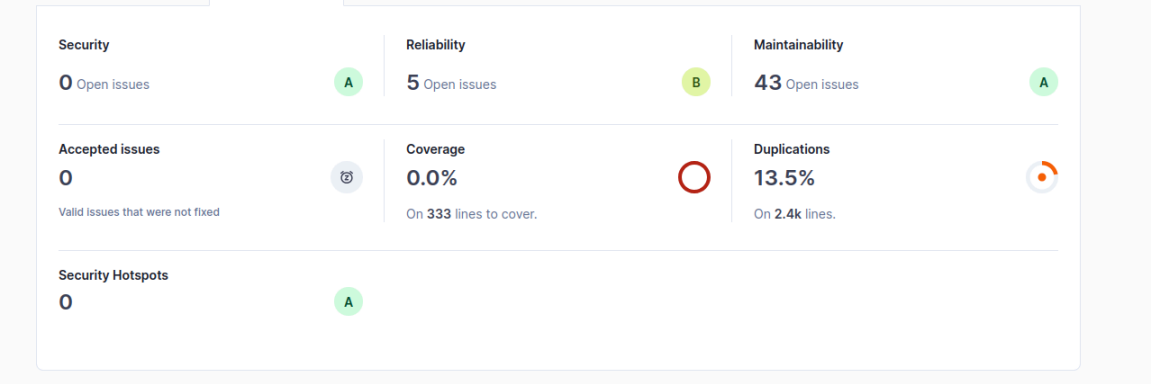

Fig. 10. Análisis tras la refactorización de código del proyecto CMU.

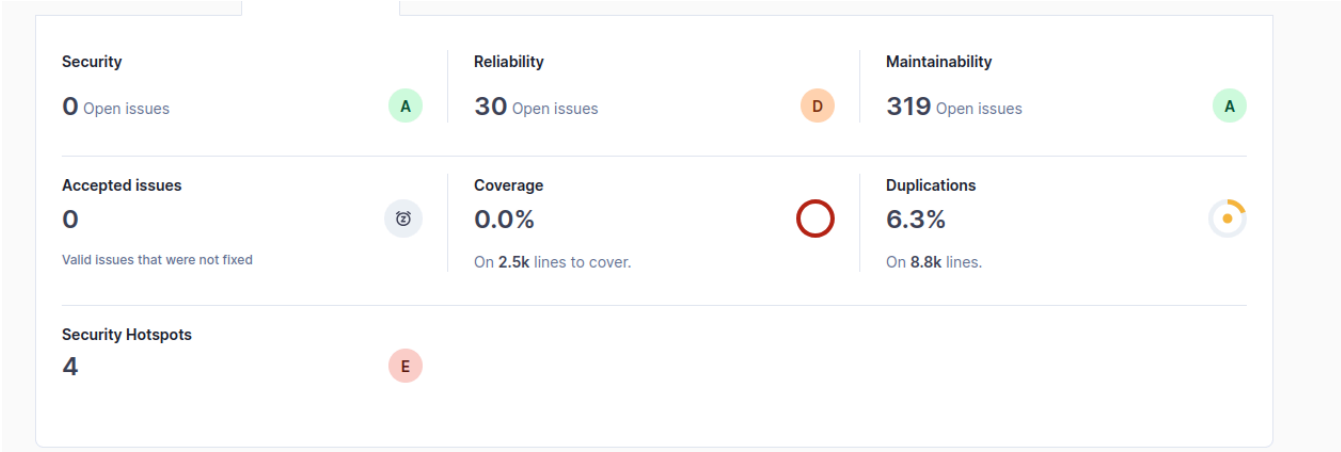

Fig. 11. Análisis tras la refactorización de código del proyecto ActCont.

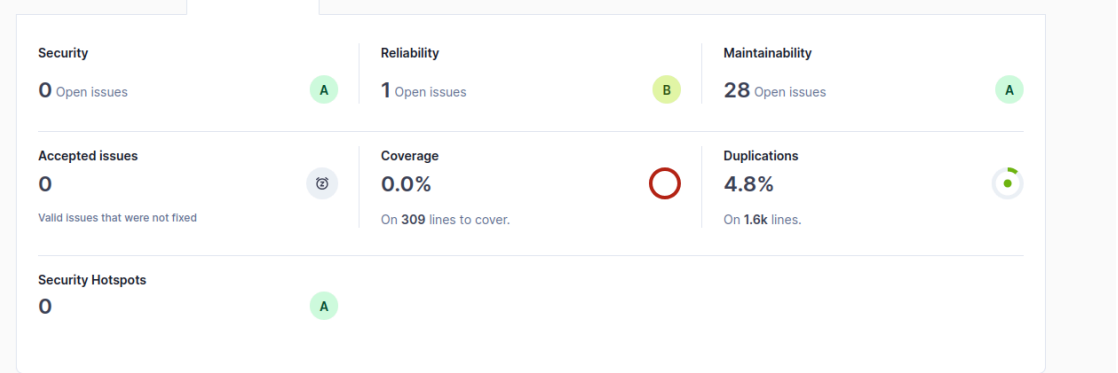

Fig. 12. Análisis tras la refactorización de código del proyecto gogoApp.

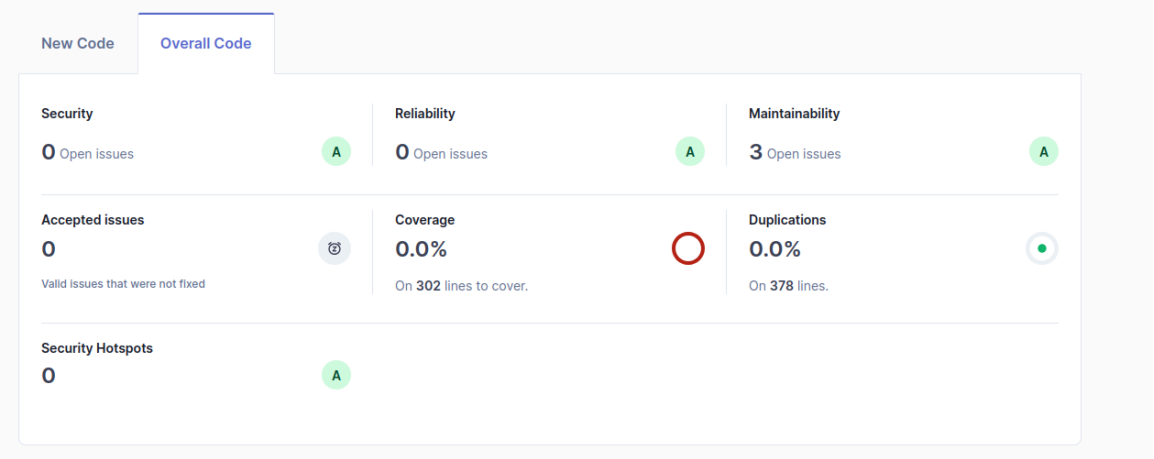

Fig. 13. Análisis tras la refactorización de código del proyecto ajedrez.

7. DISCUSIÓN:

A través de los análisis ejecutados por SonarQube fue posible encontrar defectos, vulnerabilidades y código sucio en los proyectos mediante categorización por impacto. Esta evaluación permitió identificar problemas de fiabilidad, seguridad, mantenibilidad y duplicación en distintos lenguajes y frameworks. El uso de esta herramienta proporcionó una visión clara del estado técnico del software, facilitando la priorización de tareas de refactorización según la severidad de los hallazgos.

Se evidenció cómo ciertas malas prácticas, como el uso innecesario de imports globales o la exposición de credenciales, pueden comprometer la seguridad o dificultar el mantenimiento. También se observó cómo la estructura del código, aunque funcional, puede degradar la calidad si no se siguen estándares de programación. Tras aplicar mejoras sugeridas por la herramienta, se logró reducir significativamente los indicadores negativos en todos los proyectos, confirmando que la integración continua con análisis estático de código es una práctica eficaz para mantener la calidad del software. Este laboratorio permitió contrastar teoría y práctica, y comprender cómo SonarQube actúa como una herramienta de soporte y gestión de la calidad del software,

8. CONCLUSIONES:

- Se logró instalar y configurar exitosamente la herramienta SonarQube, integrándose con SonarScanner para realizar análisis estáticos de proyectos de software desarrollados en distintos lenguajes.
- Los análisis iniciales permitieron detectar errores comunes que afectan directamente a la seguridad, mantenibilidad y fiabilidad del software.
- La refactorización guiada por los reportes de SonarQube demostró ser efectiva para reducir el número de incidencias y mejorar las métricas de calidad del código.

- SonarQube se presenta como una herramienta esencial en entornos de desarrollo profesional, al permitir monitorear y mejorar continuamente la calidad del software.

9. BIBLIOGRAFÍA:

[1] SonarSource, “Better Code & Better Software | Ultimate Security and Quality,” [En línea]. Available: https://www.sonarsource.com/. [Accedido el: 10 de mayo de 2025].

[2] SonarSource, “Try out SonarQube Server | Documentation,” [En línea]. Available: https://docs.sonarsource.com/sonarqube-server/latest/try-out-sonarqube/. [Accedido el: 10 de mayo de 2025].

[3] Docker Inc., “Get started - Docker Docs,” [En línea]. Available: https://docs.docker.com/get-started/. [Accedido el: 10 de mayo de 2025].