ESCUELA POLITÉCNICA NACIONAL

ISWD853 Desarrollo de software seguro

NOMBRES ESTUDIANTES:	Fredviner Bailón Jorge Dueñas Marco Márquez Milton Pastor Jonathan Poaquiza
FECHA:	03/08/2025

TEMA: Informe de Análisis Estático – SonarQube

Proyecto: Aplicación Python (Flask)

Fecha del análisis: 2025-08-03

Quality Gate: PASSED

Contenido

1. Descripción General del Proyecto

La aplicación analizada es un sistema desarrollado en Python usando el framework Flask, con funcionalidades relacionadas a operaciones bancarias y manejo de usuarios. El proyecto incluye manejo de autenticación con JWT, ejecución de consultas SQL, y una estructura que integra lógica de negocio y acceso a datos.

El análisis se ha realizado con SonarQube, herramienta de análisis estático de código que permite identificar vulnerabilidades, problemas de mantenibilidad, duplicación de código y cobertura de pruebas.

2. Objetivos y Alcance del Análisis

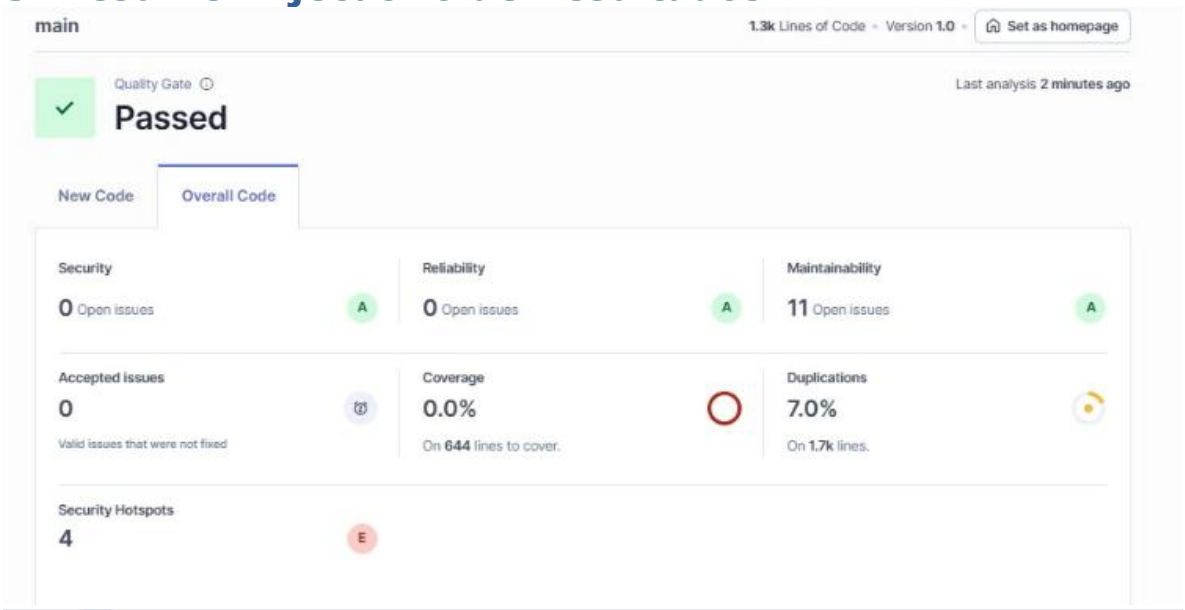
Objetivos:

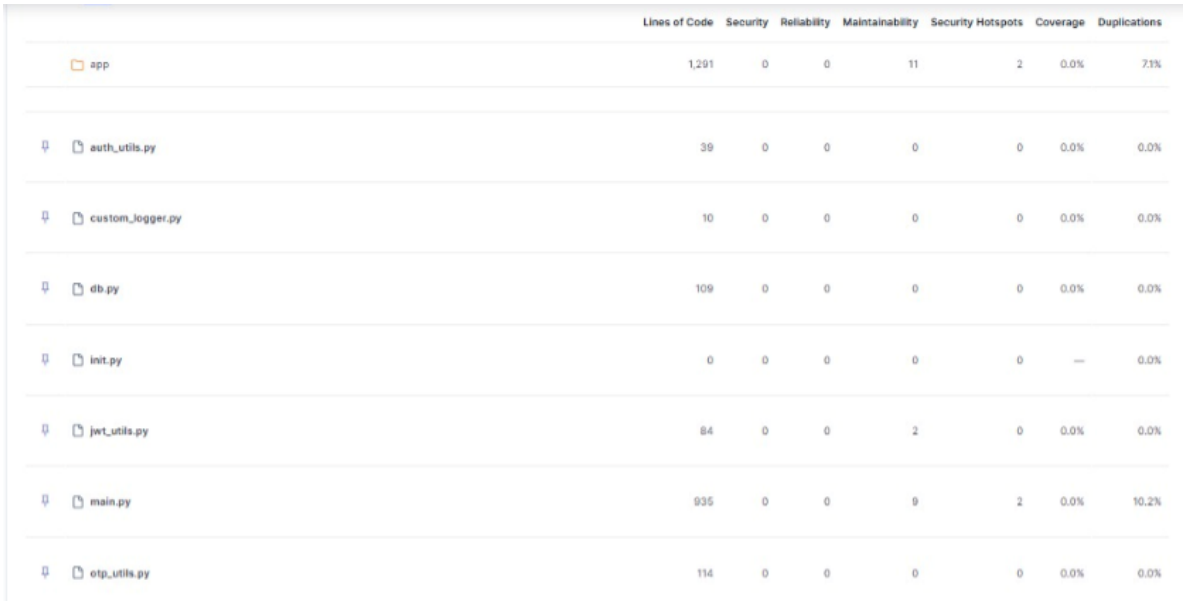
- Identificar vulnerabilidades y hotspots de seguridad que puedan comprometer la aplicación.
- Detectar problemas de mantenibilidad y duplicación de código.
- Evaluar la cobertura de pruebas unitarias.
- Proporcionar recomendaciones para mejorar la calidad del código y reducir riesgos.

Alcance:

- Análisis completo de todos los módulos fuente del proyecto.
- Evaluación de Security, Reliability, Maintainability, Coverage y Duplications.
- Revisión de Security Hotspots detectados.
- No se incluye ejecución dinámica de pruebas (solo análisis estático).

3. Resumen Ejecutivo de Resultados





	Lines of Code	Security	Reliability	Maintainability	Security Hotspots	Coverage	Duplications
app	1,291	0	0	11	2	0.0%	7.1%
auth_utils.py	39	0	0	0	0	0.0%	0.0%
custom_logger.py	10	0	0	0	0	0.0%	0.0%
db.py	109	0	0	0	0	0.0%	0.0%
init.py	0	0	0	0	0	—	0.0%
jwt_utils.py	84	0	0	2	0	0.0%	0.0%
main.py	935	0	0	9	2	0.0%	10.2%
otp_utils.py	114	0	0	0	0	0.0%	0.0%

El análisis estático realizado con SonarQube sobre el código fuente del proyecto arroja un balance positivo en aspectos críticos como seguridad y fiabilidad, ya que no se detectaron vulnerabilidades abiertas ni errores que puedan interrumpir la ejecución del sistema.

Sin embargo, se identificaron áreas importantes de mejora:

- Cobertura de pruebas: Actualmente es del 0% sobre 644 líneas relevantes, lo que implica que no se cuenta con pruebas automatizadas que validen el correcto funcionamiento del código. Esto aumenta la probabilidad de que errores no detectados lleguen a producción.

- Duplicación de código: Se detecta un 7% de líneas duplicadas en el total analizado (1.7k líneas). Esto afecta negativamente la mantenibilidad, ya que cualquier cambio en un fragmento duplicado debe repetirse en múltiples lugares, incrementando el riesgo de inconsistencias.
- Security Hotspots: Se encontraron 4 configuraciones potencialmente riesgosas, que, si bien no son vulnerabilidades confirmadas, requieren atención para evitar problemas futuros.

El proyecto presenta un buen estado en seguridad y fiabilidad, pero necesita mejoras urgentes en cobertura de pruebas y reducción de código duplicado.

4. Detalle de Issues (Maintainability)

Archivo	Línea	Tipo	Severidad	Descripción	Esfuerzo estimado
app/jwt_utils.py	42	Error Handling	High	Especificar clase de excepción	5 min
app/jwt_utils.py	52	Error Handling	High	Especificar clase de excepción	5 min
app/main.py	7	Importación	High	Importar solo nombres necesarios	5 min
app/main.py	8	Importación	High	Importar solo nombres necesarios	5 min
app/main.py	50	Diseño	High	Literal 'CajeroPass123!' repetido x3	6 min
app/main.py	213	Diseño	High	Literal "Credenciales inválidas" repetido x3	6 min
app/main.py	677	Diseño	High	Literal "El monto debe ser mayor que cero" repetido x4	8 min
app/main.py	700	Diseño	High	Literal "Cuenta no encontrada" repetido x4	8 min
app/main.py	752	Diseño	High	Query repetida x7	14 min
app/main.py	888	Diseño	High	Query repetida x3	6 min
app/main.py	991	Diseño	High	Query repetida x3	6 min

Se identificaron 11 issues de alta prioridad, todos clasificados dentro de la categoría de mantenibilidad. Los hallazgos más relevantes son:

- Duplicación de literales y consultas SQL

1. Mensajes de error y consultas SQL repetidas en múltiples lugares del código (app/main.py).
2. Esto provoca que cualquier cambio deba replicarse manualmente, aumentando el riesgo de errores y reduciendo la eficiencia del mantenimiento.

- Manejo de excepciones genéricas

1. En app/jwt_utils.py se detecta captura de errores sin especificar el tipo de excepción.
2. Esto puede ocultar fallos importantes y dificultar la depuración.

- Importaciones innecesarias

1. En app/main.py se importan módulos completos en lugar de funciones específicas.
2. Esto reduce la claridad del código y puede incrementar el consumo de recursos.

Estos problemas no interrumpen la ejecución del sistema, pero afectan la claridad, mantenibilidad y escalabilidad del código. Una refactorización temprana reducirá el esfuerzo de mantenimiento a largo plazo.

5. Hotspots de Seguridad

Categoría	Ubicación	Descripción	Riesgo Potencial	Acción Recomendada
CSRF	app = Flask(__name__)	Protección CSRF deshabilitada	Ejecución de solicitudes no autorizadas	Habilitar CSRF
Permisos	FROM python:3.10-slim	Imagen Docker usa usuario root	Escalamiento de privilegios	Configurar usuario no root
Insecure Config	app.run(..., debug=True)	Modo debug activo	Fuga de información	Desactivar en producción
Otros	RUN apt-get update && apt-get install ...	Instalación de paquetes recomendados por defecto	Inclusión de paquetes innecesarios	Usar --no-install-recommends

Se identificaron 4 puntos de atención en seguridad que, si bien no representan vulnerabilidades confirmadas, requieren validación manual y posibles ajustes para asegurar que la aplicación sea segura en producción:

- Protección CSRF deshabilitada

Riesgo de ataques donde un usuario podría ejecutar acciones no autorizadas sin saberlo.

- Ejecución con usuario root en Docker

Si un atacante obtiene acceso, tendría control total del sistema.

- Modo debug activo en producción

Puede exponer información sensible a usuarios externos.

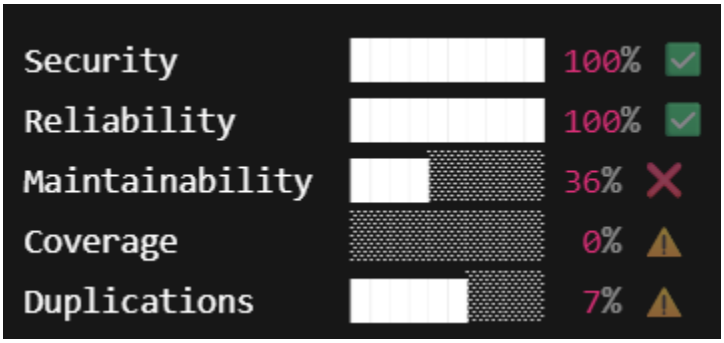
- Instalación automática de paquetes recomendados en Docker

Puede incluir componentes innecesarios que amplíen la superficie de ataque.

Aunque actualmente no se ha identificado explotación activa de estas configuraciones, ignorar estos puntos puede derivar en vulnerabilidades críticas en el futuro.

6. Métricas Relevantes

Cobertura: 0.0% (Debe incrementarse al 70% mínimo recomendado).
Duplicación de código: 7.0% (Objetivo: <3%).
Tiempo estimado de corrección de issues: 1h 23min.



7. Recomendaciones Finales

Mantenibilidad

- Centralizar literales y queries en un archivo de constantes.
- Mejorar manejo de excepciones especificando clases concretas.
- Reducir imports innecesarios.

Seguridad

- Activar protección CSRF en formularios.
- Configurar usuario no root en Docker.
- Desactivar modo debug antes de producción.
- Usar --no-install-recommends en instalaciones de paquetes.

Cobertura de Código

- Implementar pruebas unitarias con pytest.
- Integrar medición de cobertura en pipeline CI/CD.

Conclusiones

Estado general del proyecto

El análisis realizado indica que el sistema cuenta con un código estable y sin vulnerabilidades críticas detectadas, lo que demuestra un buen nivel de seguridad en los aspectos esenciales. Sin embargo, el hallazgo de 4 hotspots de seguridad evidencia que existen configuraciones y prácticas que requieren revisión antes de un despliegue en producción.

Cobertura de pruebas insuficiente

La ausencia total de pruebas automatizadas (0% de cobertura) representa uno de los puntos más críticos detectados. Esto implica que no existe un mecanismo automatizado para validar el correcto funcionamiento del sistema después de cambios o actualizaciones, lo que aumenta el riesgo de introducir fallos en producción.

Mantenibilidad afectada por código duplicado

La detección de 7% de código duplicado y 11 issues de alta prioridad relacionadas principalmente con duplicación de literales y consultas SQL, manejo genérico de excepciones e importaciones ineficientes, refleja la necesidad de una refactorización que facilite el mantenimiento y reduzca el esfuerzo de futuras modificaciones.

Seguridad y configuración en entornos de despliegue

Los security hotspots identificados, como la ejecución de la aplicación en modo debug, el uso de usuario root en Docker y la ausencia de protección CSRF, no representan vulnerabilidades activas en este momento, pero deben ser corregidos para cumplir con las mejores prácticas de seguridad en entornos productivos.

Oportunidad de mejora continua

Implementar pruebas unitarias, centralizar valores y consultas, optimizar las importaciones y aplicar buenas prácticas de seguridad permitirá elevar el estándar de calidad del proyecto, reducir riesgos operativos y facilitar su evolución futura.