PAN TYS MEL

PLAN DE MANTENIMIENTO SONARCLOUD

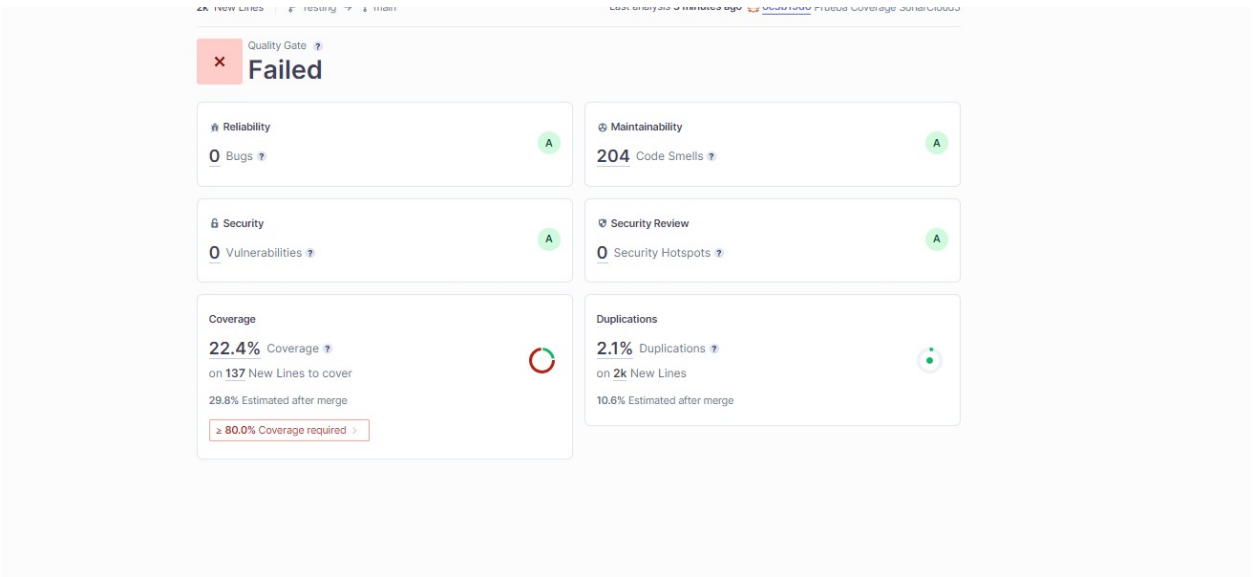
Introducción

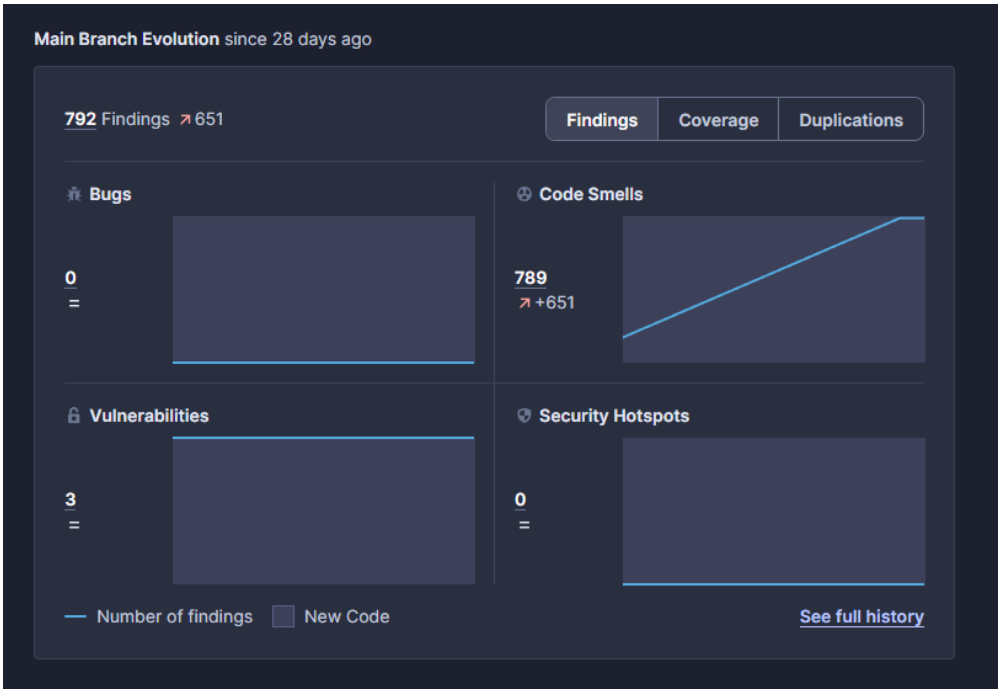
Nos enfocamos en garantizar la óptima funcionalidad, seguridad y confiabilidad de nuestro código al implementar un sólido Plan de Mantenimiento para nuestro programa en SonarCloud. Este plan incluye una evaluación completa de los medios de producción para mantener su rendimiento, optimizar los métodos de trabajo, lograr los objetivos y aprovechar al máximo los recursos disponibles.

La integridad y calidad del software, mejorando tanto la mantenibilidad como la confiabilidad, dependen del uso de SonarCloud, una herramienta de análisis estático de código basada en la nube.

Primer análisis

Primeramente partiremos de un **análisis inicial** (hecho de forma manual) y en base a los resultados mostrados tomaremos las medidas oportunas para resolver los problemas de cobertura, bugs, Code Smells, duplicaciones de código y vulnerabilidades.





Quality Gates

En el proceso de desarrollo se utilizan **Quality Gates** para la evaluación y garantización de la calidad del producto en cada etapa del ciclo de vida. Estos establecen algunos criterios y estándares que deben ser cumplidos antes de pasar a la siguiente etapa de desarrollo.

Idealmente, todos los proyectos utilizarán la misma puerta de calidad (Quality Gate), pero eso no siempre es práctico.

Los Quality Gates pueden abordar diversos aspectos de la calidad del software, como la funcionalidad, la seguridad, el rendimiento, la usabilidad, entre otros.

Mantenimiento preventivo

El **mantenimiento preventivo** de software se encarga (mediante las actividades planificadas realizadas en el código y la infraestructura de software) de prevenir problemas antes de que estos ocurran. Esto implica el uso de actualizaciones, parches, revisiones y otras medidas para garantizar el funcionamiento óptimo, la seguridad y la longevidad del software. Esto evita posibles fallas y mejora su rendimiento a lo largo del tiempo.

Para prevenir **malas prácticas** en código nos haremos cargo de estudiar las reglas de Java que se han establecido en sonarClub que están implicadas en la realización del código. Así como evitar el uso repetido de líneas de código, optimización del código, generación de comentarios…

Mantenimiento correctivo

El **mantenimiento correctivo** de software implica la corrección de errores, defectos o fallos encontrados en un programa **después de su implementación**. El objetivo es solucionar problemas identificados para restaurar el funcionamiento adecuado del software y minimizar impactos negativos en su desempeño. Este tipo de mantenimiento se lleva a cabo en respuesta a incidencias reportadas por usuarios o detectadas durante pruebas, garantizando que el software opere correctamente y de manera confiable.

A continuación, describiremos cómo abordaremos distintos departamentos, aplicando diversos tipos de correcciones. Posteriormente, destacaremos las mejoras en el código tras implementar estas acciones para presentar la evolución de las métricas de calidad de nuestro proyecto.

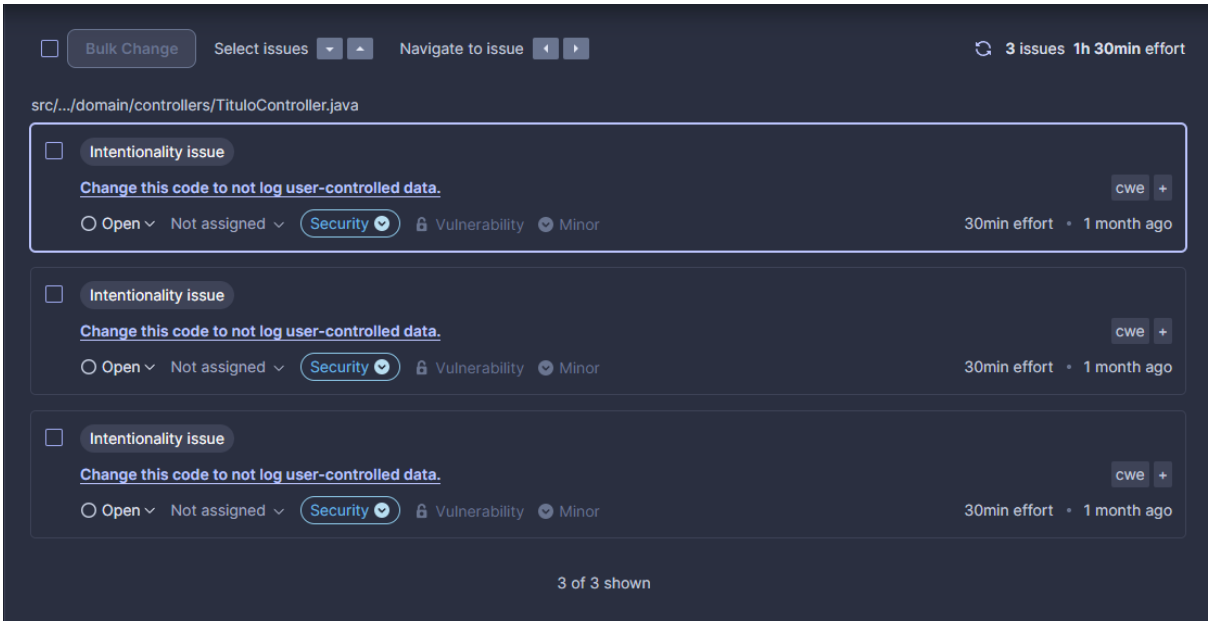
Para ejecutar eficazmente el plan de mantenimiento, debemos considerar la prioridad de los elementos a tratar. Entendemos que un error es crítico si:

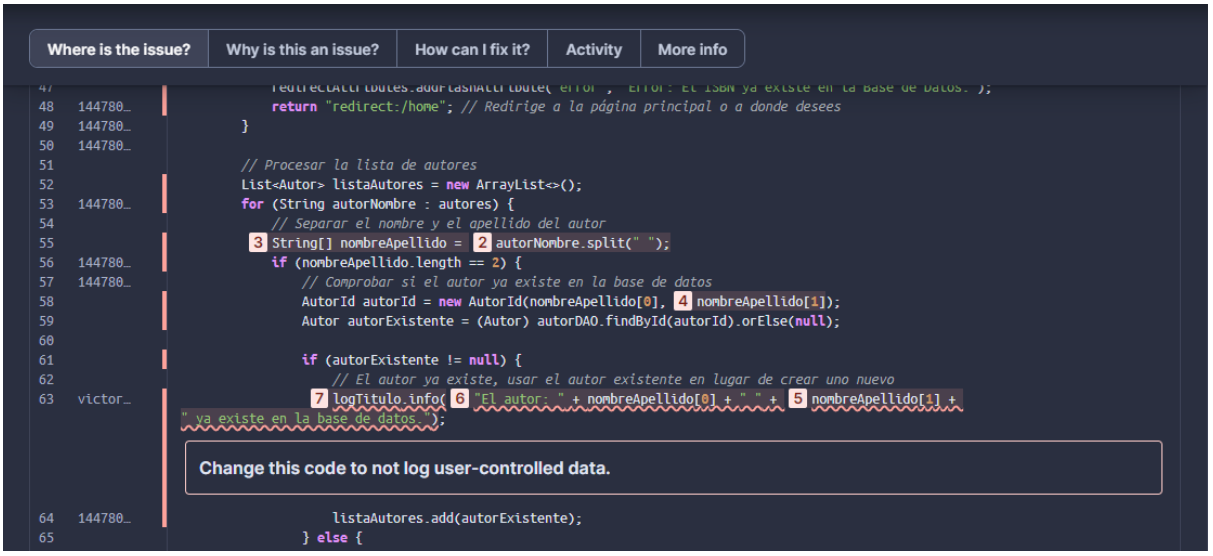
- **Pone en riesgo o compromete** la seguridad de la aplicación o del cliente.
- **Afecta a la eficacia** de la aplicación.
- **Afecta a la eficiencia** de la aplicación.

Ordenados por grado de criticidad:

1. Security (Vulnerabilities)

La sección de seguridad va a ser la primera que trataremos y llevaremos a cabo un tipo de mantenimiento correctivo. Los exploits de vulnerabilidades de seguridad nos muestran cuán vulnerable puede ser realmente el código de software.





Estas tres vulnerabilidades hacían referencia a un posible ataque por inyección con relación al funcionamiento y condición de los Loggers, la solución fue cambiar los diferentes parámetros que permitían la modificación de datos por parte del usuario.

2. *Reliability (Bugs)*

La cantidad de **bugs** o errores dentro del código que comprometiesen la confiabilidad del programa fue 0, esto debido a nuestra política de saneamiento y control que se fue aplicando desde el inicio para evitar la mayor cantidad de problemas a futuro.

Para cuando se realizó el primer análisis dentro de Sonar Cloud ya se habían solucionado todos los errores.

3. *Coverage*

La cobertura de código hace referencia a un mecanismo de medición de la calidad del software a través de la revisión de las pruebas. Específicamente, esta herramienta determina el rendimiento de las pruebas teniendo en cuenta su nivel de aplicación general.

El coverage mínimo al que nos centramos en llegar fue el 80% para asegurar una calidad notable del código.

4. *Maintainability(Code Smells)*

Los "code smells" son señales o indicios que pueden sugerir la presencia de problemas en el diseño o estructura de un código fuente. Estos problemas pueden no ser errores directos, pero indican áreas del código que podrían beneficiarse de una revisión y posible refactorización.

Empezaremos con los "*Code Smells"* que puedan afectar en mayor medida al funcionamiento del programa.

La prioridad es la siguiente: Blocker, Critical, Major, Minor.

- **Blocker:**

Se trata de un problema que tiene un **grandísimo impacto** en el aspecto de mantenibilidad del programa.

Track lack of copyright and license header.
Cada archivo fuente debe comenzar con un encabezado que indique la propiedad del archivo y la licencia que se debe utilizar para distribuir la aplicación.

Esta regla debe incluirse con el texto del encabezado que se espera al principio de cada archivo.

Este fue el único Code Smell de este tipo que observamos y su solución fue añadir una cabecera creada por nosotros dentro de la clase App.

- **Critical:**

Se trata de un problema que tiene un **gran impacto** en el aspecto de mantenibilidad del programa.

String literals should not be duplicated java:S1192

Los literales de cadena duplicados hacen que el proceso de refactorización sea complejo y propenso a errores, ya que cualquier cambio debería propagarse en todas las apariciones.

Para solucionarlo, debemos cambiar cadenas de caracteres repetidas y transformarlas en variables String.

- **Major:**

Hablamos de problemas que tienen un **impacto considerable** en la mantenibilidad del programa.

Lines should have sufficient coverage by tests common-java:InsufficientLineCoverage

Se crea un problema en un archivo tan pronto como la cobertura de línea en este archivo es menor que el umbral requerido. La solución está en proporcionar el número de líneas que se cubrirán para alcanzar el umbral requerido.

"java.time" classes should be used for dates and times java:S2143

Las antiguas clases de Fecha y Calendario siempre han sido confusas y difíciles de usar correctamente, particularmente en un contexto de subprocesos múltiples.

Magic numbers should not be used java:S109

Los números mágicos hacen que el código sea más complejo de entender, ya que requiere que el lector tenga conocimientos sobre el contexto global para comprender el número en sí. Su uso puede parecer obvio al escribir el código, pero puede no ser el caso para otro desarrollador o más adelante una vez que el contexto desaparezca. -1, 0 y 1 no se consideran números mágicos.

- **Minor:**

Se trata de un problema que tiene un **impacto menor** en el aspecto de mantenibilidad del programa.

Algunos tipos son:

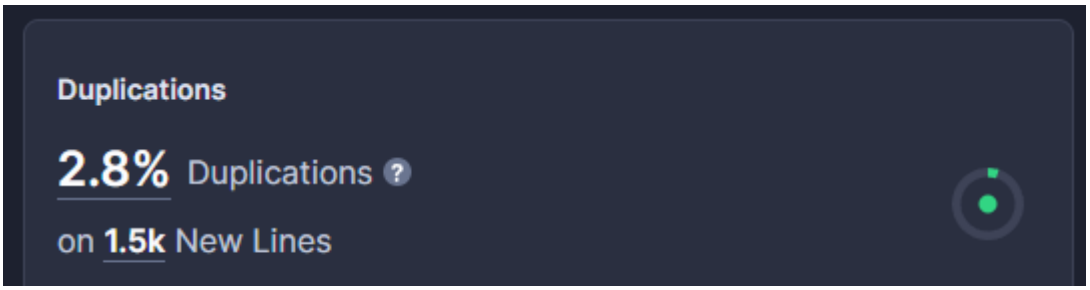
- Problemas de indentación en el código.
- Problemas con la declaración de variables locales.
- Problemas en la posición de ciertos comentarios.

Las soluciones a estos problemas son tribales.

5. Duplications

El término "duplicación de código" se refiere a la repetición del código fuente en uno o varios programas, ya sea propiedad o mantenidos por la misma entidad.

Esto suele indicar un estilo de programación deficiente, ya que un buen desarrollo se centra en la reutilización del código.



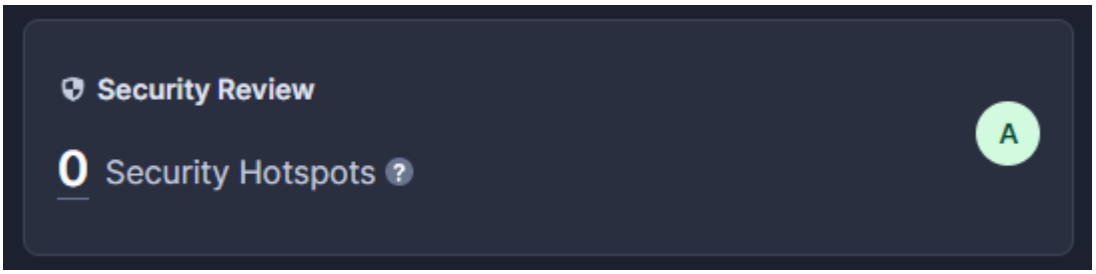
En nuestro caso se puede observar como la cantidad de código duplicado es ínfima en comparación con las líneas de código que hay:

La principal desventaja es el costoso mantenimiento del código. Los "code smell" abarcan la mayoría de los casos de duplicación, y los más graves se tratan después de los "code smell". Por ejemplo, al definir una variable final para el tipo de letra en lugar de repetir el String "Arial" varias veces, simplificamos el mantenimiento del código.

Aunque hay casos similares, al ser la solución la misma, consideramos redundante incluirlos en el plan de mantenimiento.

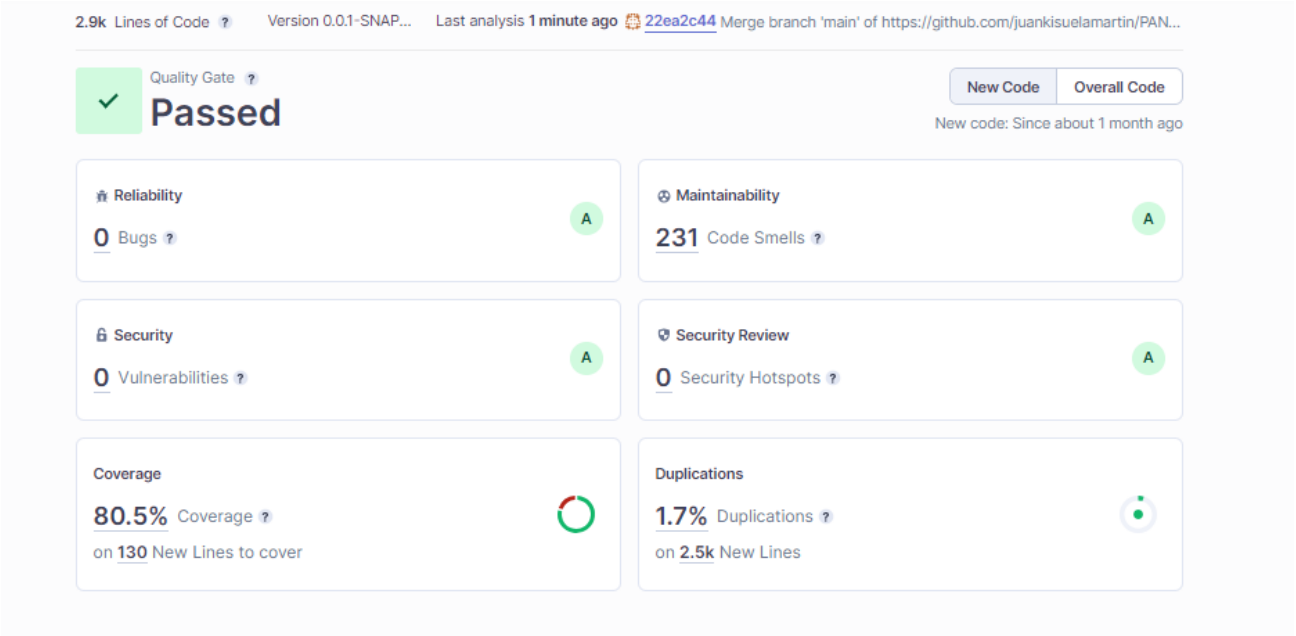
6. Security Review (Security Hotspots)

Un “Security Hotspot” destaca un fragmento de código sensible a la seguridad que el desarrollador debe revisar.



En nuestro caso una vez realizado el primer análisis no se encontraron Security Hotspot que solucionar.

Análisis final



Después de llevar a cabo el último análisis de código, es evidente que el proyecto ha superado el Quality Gate establecido, lo cual es un indicativo positivo en términos de calidad y mantenibilidad del software. Este análisis final se ha centrado en evaluar diferentes métricas y aspectos clave del código fuente, y los resultados obtenidos indican que el código cumple con los estándares definidos y sigue las mejores prácticas de desarrollo.

Principales conclusiones y observaciones:

- **Cumplimiento de Estándares:** Se ha observado un alto cumplimiento de los estándares de codificación y las convenciones definidas en las directrices del proyecto. Esto contribuye a la coherencia y legibilidad del código, facilitando su mantenimiento a largo plazo.

- **Complejidad Controlada:** Se ha logrado mantener la complejidad del código dentro de límites razonables, lo que favorece la comprensión y facilita la identificación de posibles problemas en el futuro.

- **Cobertura de pruebas:** La cobertura de pruebas ha sido satisfactoria, proporcionando una capa de seguridad para detectar posibles errores y garantizar la robustez del sistema. Es importante seguir monitoreando y mejorando la cobertura de pruebas a medida que el proyecto evoluciona.

- **Resolución de Problemas Críticos:** Los problemas críticos identificados en análisis anteriores han sido abordados y resueltos de manera efectiva, lo que ha contribuido a mejorar la calidad general del código.

- **Eficiencia en el Rendimiento:** Se ha prestado atención a la eficiencia en el rendimiento, y las optimizaciones necesarias se han implementado para garantizar una ejecución eficaz del software.