**Informe de análisis de SonarLint**

* **Proyecto: Acme-ANS**
* **Repositorio:** <https://github.com/AngSanRui/Acme-ANS-C2>
* **Responsable: Ángel Sánchez Ruiz (angsanrui2@alum.us.es)**
* **Grupo: C2.056**

**Fecha: 28/06/2025**

**Índice:**

1 Resumen ejecutivo 2

2 Introducción 2

3 Resultados del análisis 2

3.1 Seguridad 2

3.2 Mantenibilidad 3

3.3 Código duplicado 4

4 Conclusiones 4

|  |  |  |
| --- | --- | --- |
| Versión | Fecha | Cambios |
| 1.0 | 28/06/2025 | Versión Inicial |
| 1.1 | 30/06/2025 | Resumen ejecutivo añadido |

**1-Resumen ejecutivo**

Este informe muestra un análisis de código en el que se ha usado SonarLint y SonarQube. El objetivo es detectar bad smells de mantenibilidad, código duplicado y algunos errores de seguridad (aunque como se mostrará más tarde, estos últimos son más bien advertencias).

Al principio se utilizó SonarLint integrado en Eclipse pero por la dificultad para interpretar los resultados se optó por desplegar un servidor local de SonarQube.

En total se detectaron 3 advertencias de seguridad, 223 incidencias de mantenibilidad, y un 18.7% de código duplicado.

**2-Introducción**

Sonarlint es una extensión de varios IDEs(entre ellos Eclipse) que ayuda a detectar y corregir problemas de calidad mientras se escribe código. Además, permite ejecutar análisis sobre proyectos que recopilan información que puede ser valiosa si se quieren aplicar patrones de diseño, si se quiere hacer el código más eficiente, o si se quiere hacer el código más fácil de mantener, entre otras utilidades.

He ejecutado primero un análisis de SonarLint sobre el proyecto Acme-ANS-C2, y al ver la pestaña con los resultados no consideraba que el resultado mostrado en la pestaña SonarLint Report fuera muy claro. Decidí usar SonarQube para obtener unos resultados en mejor formato.

SonarLint permite integrar SonarQube, que es un servidor que permite mostrar de forma sencilla una visión de la calidad del código base de tu proyecto. He descargado y desplegado localmente un servidor de SonarQube para el informe.

**3-Resultados del análisis**

Los resultados sólo han logrado mostrar información acerca de la seguridad del código, la mantenibilidad, y el código duplicado.

**3.1-Seguridad**

Lint ha detectado 3 issues en el apartado de seguridad:

* En src/main/resources/platform-development.properties nos indica que en la línea 15 nos aseguremos de que la contraseña de la base de datos se cambie y se elimine del código. Esto, en un contexto real, puede tener consecuencias importantes si no se atiende.

A screenshot of a computer

AI-generated content may be incorrect.

* En src/main/resources/platform-production.properties nos notifica de algo similar a lo anterior

A close up of a computer screen

AI-generated content may be incorrect.

* En src/main/resources/platform-testing.properties, de nuevo, la advertencia es la misma

A close up of a text

AI-generated content may be incorrect.

**3.2-Mantenibilidad**

De mantenibilidad, se nos notifican 223 issues.

De ellas, están clasificadas con severidad alta 64. Generalmente estas issues se pueden agrupar en:

* Definir una constante en vez de duplicar código. Muchos de estos casos se tratan de strings que se usan como parámetros a la hora de usar los datasets.
* Añadir comentarios explicando por qué una función está vacía. Generalmente estos casos se tratan de la función validate, que a veces no lleva código porque se aplican los validadores de la entidad.
* Refactorizar funciones para hacerlas más fáciles de entender. Estas funciones generalmente son del framework, no han sido creadas por los alumnos.

Las que tienen severidad media son 51. Todas son:

* Cambiar asserts por checks ya que los assert pueden ser desactivados en runtime en la JVM.

Finalmente las 108 de prioridad baja principalmente son:

* No declarar múltiples variables en una misma línea.
* Renombrar paquetes para que sigan la expresión regular **“**^[a-z\_]+(\.[a-z\_][a-z0-9\_]\*)\*$'”. Pienso que los nombres de los paquetes son bastantes descriptivos y siguen un formato lógico.
* Usar isEmpty() para chequear que colecciones estén vacías.
* Eliminar statements vacíos (como los hay en los métodos validate vacíos).
* Eliminar literales tipo boolean innecesarios.

**3.3-Código duplicado**

En un proyecto como este, hay estructuras de código que fácilmente se pueden repetir. Así, hemos obtenido que el proyecto tiene un 18.7% del código duplicado.

Tras echar un vistazo a los resultados, el código duplicado analizado es muy frecuente en los servicios, siendo el que más porcentaje tiene el TechnicianMaintenanceRecordUpdateService con un 77.6% del código repetido, que son 97 líneas. De los 67 items que se muestran, todos son servicios.

**4-Conclusiones**

El análisis de Lint ha mostrado que el proyecto tiene cosas que mejorar de cara al mantenimiento principalmente. A pesar de ser muchas, esas issues suelen repetirse bastante. No obstante, ya que el proyecto no es un sistema que va a emplearse en la vida real, estos problemas de mantenimiento no suponen algo demasiado grave, además que ya de por si los bad smells señalados generalmente o no son viables de cambiar(porque sean cosas del framework) o no presentan un problema real.

También, el código duplicado es bastante, sin embargo en mi opinión sería difícil reducir el porcentaje de forma sustancial.

En mi experiencia, el uso de SonarQube para este análisis ha sido acertado, ya que se han podido sacar conclusiones más claras y de forma más sencilla.