Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática

**Lint Report**



Grado en Ingeniería Informática – Ingeniería del Software  
Diseño y Pruebas 2.

Curso 2024 – 2025

|  |  |
| --- | --- |
| **Fecha** | **Versión** |
| 23/05/2025 | v1.0.0 |

|  |  |  |
| --- | --- | --- |
| **Grupo de prácticas: C1.005** | | |
| **Autores por orden alfabético** | **Rol** | **Descripción del rol** |
| Artero Bellido Manuel – manartbel@alum.us.es | Developer | Persona encargada de desarrollar el código. |
| Calderón Rodríguez, Manuel María -mancalrod@alum.us.es | Operador | Encargado de las tareas de campo, de las instalaciones y del mantenimiento de los sistemas de la empresa. |
| González Benito, Claudio – clagonben@alum.us.es | Project Manager | Persona encargada de tomar decisiones de diseño y vigilar el correcto desarrollo |
| Márquez Gutiérrez, José Manuel – josmargut@alum.us.es | Tester | Persona encargada de realizar pruebas sobre el código. |
| Ramos Vargas, Alba – albramvar1@alum.us.es | Developer | Persona encargada de desarrollar el código. |

**Control de Versiones**

|  |  |  |
| --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** |
| 23/05/2025 | v1.0.0 | Desarrollo de la primera versión. |
|  |  |  |
|  |  |  |
|  |  |  |

Repositorio: <https://github.com/Manuelgithuv/Acme-ANS-D04>

Índice de contenido

[**1.** **Resumen ejecutivo** 2](#_Toc198907746)

[**2.** **Introducción** 3](#_Toc198907747)

[**3.** **Contenido** 4](#_Toc198907748)

[**4.** **Conclusiones** 7](#_Toc198907749)

# **Resumen ejecutivo**

El propósito de este documento es asegurar la transparencia en el mantenimiento del código, demostrar nuestra comprensión de las alertas generadas por SonarLint y justificar las decisiones tomadas respecto a la necesidad de corregir o no cada uno de estos problemas. Con ello, buscamos mantener un equilibrio entre la calidad del código y la eficiencia en el desarrollo, evitando cambios innecesarios que no aporten un valor relevante al proyecto.

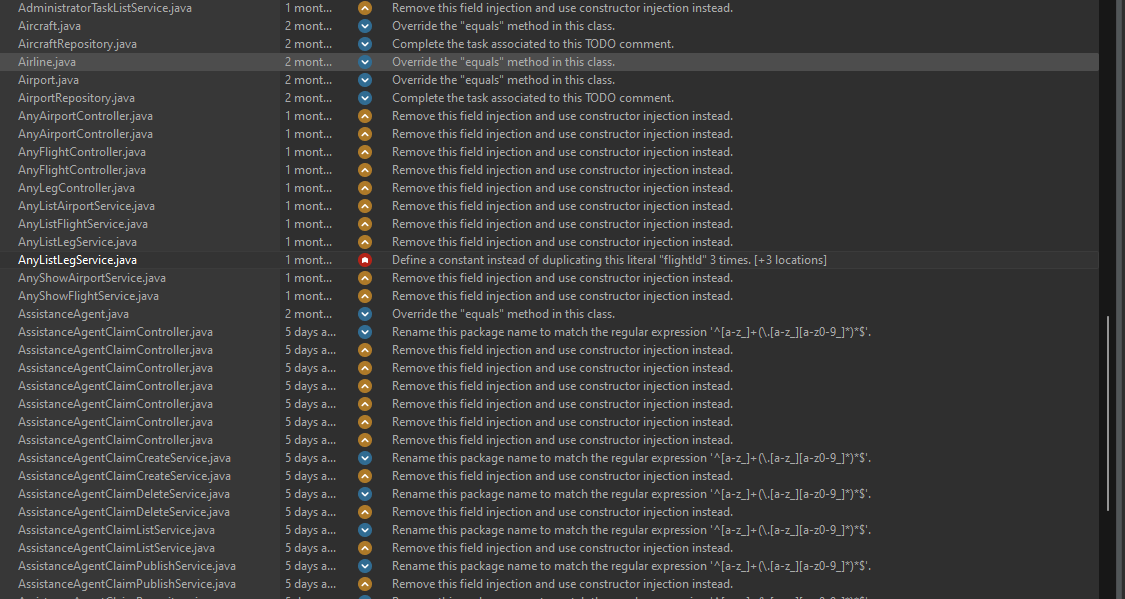
El principal objetivo de esta entrega es probar de manera formal el funcionamiento correcto de nuestra aplicación, grabando para ello las pruebas necesarias sobre distintas operaciones legales e ilegales en las funcionalidades implementadas en la anterior entrega.

# **Introducción**

Este documento presenta una lista de los “bad smells” (códigos problemáticos o antipatrón) identificados por SonarLint en nuestro proyecto. SonarLint es una herramienta de análisis estático que detecta posibles errores, problemas de calidad y malas prácticas en el código. Sin embargo, no todos los problemas detectados necesariamente afectan la funcionalidad o el rendimiento de nuestro software de manera significativa.

En las siguientes secciones, se enumeran los “bad smells” reportados, proporcionando una descripción detallada de cada uno, así como la ubicación específica en el código. Además, se ofrece una justificación para cada caso, explicando por qué consideramos que estos problemas son inocuos y no requieren intervención inmediata.

# **Contenido**



|  |  |  |  |
| --- | --- | --- | --- |
| **Clase** | **Línea** | **Descripción** | **Justificación** |
| AnyListLegService | 18 | Remove this field injection and use constructor injection instead | Este “bad smell” es debido a inyectar dependencias directamente sin usar el constructor de la clase. No se puede quitar para que el framework use la dependencia inyectada correctamente. |
| AnyListLegService | 30 | Define a constant instead of duplicating this literal "flightId" 3 times. | Este “bad smell” surge porque al copiar y pegar la misma cadena se introducen duplicaciones que elevan el riesgo de errores (p. ej. tipográficos) y desincronización en cambios posteriores. Al usar una constante, se evita la redundancia y se asegura que cualquier modificación se propague correctamente en todo el código. No se ha arreglado por que pertenece a un entregable extra. |
| AnyListFlightService | 16 | Remove this field injection and use constructor injection instead | Este “bad smell” es debido a inyectar dependencias directamente sin usar el constructor de la clase. No se puede quitar para que el framework use la dependencia inyectada correctamente. |
| AnyShowFlightService | 15 | Remove this field injection and use constructor injection instead | Este “bad smell” es debido a inyectar dependencias directamente sin usar el constructor de la clase. No se puede quitar para que el framework use la dependencia inyectada correctamente. |
| AnyFlightController | 15 | Remove this field injection and use constructor injection instead | Este “bad smell” es debido a inyectar dependencias directamente sin usar el constructor de la clase. No se puede quitar para que el framework use la dependencia inyectada correctamente. |
| AnyFlightController | 18 | Remove this field injection and use constructor injection instead | Este “bad smell” es debido a inyectar dependencias directamente sin usar el constructor de la clase. No se puede quitar para que el framework use la dependencia inyectada correctamente. |
| AnyLegController | 15 | Remove this field injection and use constructor injection instead | Este “bad smell” es debido a inyectar dependencias directamente sin usar el constructor de la clase. No se puede quitar para que el framework use la dependencia inyectada correctamente. |

**Advertencia**:

Todos los “code smells” no pertenecientes a clases implementadas por el estudiante 1 no han sido analizados, puesto que no son responsabilidad de este la mala implementación de código o él no uso de buenas prácticas, por lo que este solo puede garantizar que para la realización de su código se ha usado el feedback recibido en clase y las buenas prácticas comentadas en estas.

# **Conclusiones**

En conclusión, aunque el código no está completamente libre de “bad smells”, es importante destacar que muchos de estos no son el resultado de malas prácticas o errores de nuestra parte. Las líneas de código que generan estas alertas son, en su mayoría, necesarias para el correcto funcionamiento del framework que estamos utilizando, y se han implementado siguiendo las especificaciones y directrices proporcionadas por los docentes en clase.

La decisión de mantener estos “bad smells” está respaldada por una evaluación cuidadosa de su impacto en la funcionalidad y rendimiento de la aplicación. En muchos casos, las modificaciones sugeridas por SonarLint podrían comprometer la estabilidad o el comportamiento esperado del software, se ha optado por priorizar la adherencia a los requisitos del framework y las prácticas recomendadas.

El compromiso con la calidad del código sigue siendo una prioridad, por lo que siempre que es posible se revisa la calidad de este de manera constante. La presencia de estos “code smells” no afecta negativamente la eficiencia o la fiabilidad de la aplicación, y se han tomado las medidas necesarias para asegurar que cualquier posible riesgo esté adecuadamente mitigado.

1. **Bibliografía**

Intencionalmente en blanco