Informe de análisis SonarLint – Student1

Número del grupo: C2-013

Repositorio: https://github.com/javpalgon/Acme-ANS-C2

Integrantes del grupo:

-Santia Bregu – sanbre@alum.us.es
-Raquel García Hortal – raqgarhor@alum.us.es

Fecha: 02/07/2025

1 CONTENIDO

2 RESUMEN EJECUTIVO

Este informe documenta el análisis realizado mediante SonarLint sobre el proyecto Acme-ANS-C2.
Se identificaron múltiples "bad smells" (malas prácticas o patrones de código sospechosos), de los cuales se han filtrado aquellos que pertenecen al framework Acme, conforme a las indicaciones.
En este documento se detallan los avisos reportados que, tras análisis, se consideran inocuos y no requieren corrección.
Cada entrada se acompaña de una justificación técnica que respalda su decisión. El objetivo de este informe es demostrar que el equipo ha aplicado un juicio crítico frente a las herramientas de análisis estático, corrigiendo solo cuando realmente se considera necesario y justificando cuando no.
Se espera que este documento sirva como referencia para la validación de calidad del código y pueda ser consultado por el profesorado para resolver dudas.

3 TABLA DE REVISIONES

Versión	Fecha	Descripción
1.0	03/07/2025	Versión inicial del informe
1.1	03/07/2025	Revisión y corrección de redacción

4 INTRODUCCIÓN

Durante el desarrollo del proyecto Acme-ANS-C2, se ha utilizado SonarLint como herramienta de análisis estático de código para detectar posibles defectos, malas prácticas o patrones inadecuados que puedan afectar a la calidad del software.
Este informe recoge aquellos "bad smells" que SonarLint ha detectado y que, tras una evaluación detallada por parte del equipo, se consideran inocuos.
Conforme a las indicaciones, no se incluyen los avisos generados sobre el framework Acme, y se han omitido los que fueron corregidos durante el desarrollo.
El informe se estructura en una tabla donde se exponen: la descripción del problema reportado, los archivos afectados y una justificación técnica sobre por qué no se considera necesario corregir dicho problema.
Finalmente, se incluyen las conclusiones generales del análisis y una sección de bibliografía.

5 CONTENIDO: JUSTIFICACIÓN DE BAD SMELLS

Regla / Descripción	Archivo(s)	Justificación
Remove this field injection and use constructor injection instead.	Varios (`Administrator*.java`, `Aircraft*.java`, `Airline*.java`)	Uso de @Autowired por simplicidad en un contexto controlado, sin impacto negativo en la testabilidad o ciclo de vida.
Replace this assert with a proper check.	AdministratorDashboardShowService.java, etc.	Uso interno para validaciones de desarrollo. No se ejecutan en producción por defecto.
Remove this useless assignment to local variable	AdministratorDashboardShowService.java, AircraftUpdateService.java	Asignaciones conservadas durante pruebas. No afectan lógica ni rendimiento.
This block of commented-out lines of code should be removed.	WeatherListService.java, etc.	Comentarios temporales como referencia de funcionalidades anteriores.
A NullPointerException could be thrown	WeatherCreateService.java	Control lógico previo garantiza que no se accede a variables nulas.
Add a nested comment explaining why this method is empty	CreateService.java, DisableService.java	Métodos vacíos por diseño, ya que forman parte del ciclo CRUD o plantillas abstractas.
Override the 'equals' method in this class.	Airline.java, Aircraft.java	No se requiere comparación lógica en estructuras de datos que lo necesiten.
Rename this field 'IATACode' to match...	Airline.java	Nombre semánticamente correcto; el cambio rompería legibilidad innecesariamente.
Define a constant instead of duplicating literal	AircraftCreateService.java, etc.	Duplicación leve y justificada por claridad del código en su contexto actual.
Replace negation and anyMatch with noneMatch	AirlineUpdateService.java	El uso de '!anyMatch()' se considera más expresivo y no afecta funcionalidad.

6 CONCLUSIONES

El análisis realizado con SonarLint ha permitido detectar numerosas advertencias dentro del proyecto.
Tras una revisión exhaustiva, muchas de ellas han sido corregidas durante el desarrollo, mientras que otras, como las recogidas en este informe, se han considerado inocuas.
Estas decisiones han sido respaldadas por criterios técnicos y experiencia previa, priorizando claridad, contexto y funcionalidad.
La aplicación rigurosa de herramientas automáticas como SonarLint, combinada con el juicio humano, permite mantener un equilibrio entre calidad técnica y pragmatismo en entornos reales de desarrollo académico.
El equipo reafirma su compromiso con las buenas prácticas, demostrando capacidad de análisis crítico y autonomía en la toma de decisiones respecto al mantenimiento del código.

7 BIBLIOGRAFÍA

Intentionally blank.