Universidad de Sevilla  
Escuela Técnica Superior de Ingeniería Informática

**Lint Report**



Grado en Ingeniería Informática – Ingeniería del Software  
DISEÑO Y PRUEBAS II

Curso 2024 – 2025

|  |  |
| --- | --- |
| **Fecha** | **Versión** |
| 03/07/2025 | V01r01 |

|  |  |
| --- | --- |
| **Grupo:** C2.006 | |
| **Autores** | |
| Castaño Juan, Cynthia | cyncasjua@alum.us.es |

**Repositorio GITHUB: https://github.com/rubpergar/Acme-ANS-C2.git**

**Índice de contenido**

[**1.** **Tabla de versiones** 2](#_Toc202409990)

[**2.** **Resumen ejecutivo** 2](#_Toc202409991)

[**3.** **Introducción** 2](#_Toc202409992)

[**4.** **Contenido** 2](#_Toc202409993)

[**4.1. Bad smells repetidos** 3](#_Toc202409994)

[**4.2.** **Bad smells en entidades y validadores** 3](#_Toc202409995)

[**4.3.** **Bad smells en servicios** 4](#_Toc202409996)

[**5.** **Conclusiones** 5](#_Toc202409997)

[**6. Bibliografía** 5](#_Toc202409998)

# **Tabla de versiones**

|  |  |  |
| --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** |
| 03/07/2025 | V01r01 | Documento inicial |
|  |  |  |
|  |  |  |

# **Resumen ejecutivo**

Este documento ofrece una visión general completa de pruebas estáticas que consisten en inspeccionar el código para detectar errores, concretamente para buscar los posibles "bad smells", es decir, fragmento de código fuente propenso a ocultar errores, detectados por el análisis del plug-in SonarLint en relación con los requisitos del Estudiante 1. Estos son identificados y serán analizados.

# **Introducción**

El objetivo de este documento es ofrecer un análisis de los code smells (malos olores de código) detectados en el proyecto, organizados según las distintas partes del estudiante 1 que aparecen en el proyecto. Un bad smell es un fragmento de código fuente que, aunque no sea necesariamente incorrecto, es propenso a ocultar errores. Cuantos más bad smells contiene un bloque de código, mayores son las probabilidades de que se oculte un fallo real.

Para cada bad smell identificado, se proporciona una justificación clara que explica por qué se considera inofensivo en el contexto específico en el que aparece.

En los casos en los que un mismo bad smell se repita en varias partes del código (por ejemplo, recomendaciones sobre la definición de constantes), se evaluará de forma conjunta, indicando con precisión las ubicaciones donde ha sido detectado.

# **Contenido**

## **4.1. Bad smells repetidos**

* “Remove this field injection and use constructor injection instead.”

Aparece este bad smell en todo lugar que contenga un @Autowired, lo cual ignoramos, ya que la metodología requiere usar esa anotación para inyectar cualquier repositorio o servicio.

* “Define a constant instead of duplicating this literal "{att}" x times. [+x locations] “

Siendo x el número de veces que aparece y att el atributo correspondiente en cada caso. Este bad smell aparece en numerosos servicios. La duplicación de "{att}" está vinculada a la gestión directa de datos en peticiones y respuestas (load, unbind…), donde la claridad del literal es crucial para evitar errores y vulnerabilidades. Por lo tanto, su duplicación en este contexto es justificada y aceptable.

## **Bad smells en entidades y validadores**

* “Merge this if statement with the enclosing one.” FlightValidator.java

Texto

El contenido generado por IA puede ser incorrecto.

Se unifica el if:

Texto

El contenido generado por IA puede ser incorrecto.

* “Override the "equals" method in this class.”

Este posible problema se detecta en todas las entidades y roles del proyecto, debido a que no se ha definido un método equals específico para ellas. No obstante, como todas heredan de AbstractEntity o AbstractRole, esta función ya está cubierta por el framework de manera automática.

## **Bad smells en servicios**

* “Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' and ensure that the list is unmodified.” ManagerLegPublishService.java

Texto

El contenido generado por IA puede ser incorrecto.

Se aplica la recomendación:

Texto

El contenido generado por IA puede ser incorrecto.

* Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation.

Aparece cada vez que hay algún método vacío, y aunque lo considero inofensivo, añadí un comentario a cada método.

# **Conclusiones**

Sonar’s Lint es una herramienta práctica y efectiva para identificar posibles problemas en el código sin necesidad de ejecutar pruebas o compilar el proyecto. Aunque no es infalible y se basa en patrones comunes para detectar errores, ofrece la ventaja de mostrar estos posibles problemas en tiempo real. En este caso, no se han encontrado malos olores significativos, pero siempre es útil revisar el análisis que proporciona para mantener la calidad del código.

# **6. Bibliografía**

Intencionalmente vacía