|  |
| --- |
| **Lint**  **Report** |
| **Ángela Lopez Oliva** [**(anglopoli1@alum.us.es)**](mailto:(anglopoli1@alum.us.es) |
| **26-05-2025**  **Grupo: C1.030 Repositorio:**  **https://github.com/JuanAntonioMorenoMoguel/Acme-ANS-D04** |

**1**



**Tabla de Contenidos**

[Resumen Ejecutivo 3](#_bookmark0)

[Historial de Versiones 4](#_bookmark1)

[Introducción 5](#_bookmark2)

[Contenidos 6](#_bookmark3)

[Conclusiones 6](#_bookmark3)

[Bibliografía 10](#_bookmark4)

# Resumen Ejecutivo

Este informe tiene como objetivo evaluar los *bad smells* detectados en el proyecto Acme-ANS (versión D04), concluyendo que no comprometen la funcionalidad ni la mantenibilidad del sistema.

# Historial de Versiones

|  |  |  |
| --- | --- | --- |
| Versión | Fecha | Descripción de los Cambios |
| 1.0 | 26/05/2025 | Creación y revisión del documento para el entregable D04 |

# Introducción

Este análisis recoge los *bad smells* detectados por SonarLint en la versión D04 del proyecto Acme-ANS. Tras examinar cuidadosamente las advertencias generadas, se ha determinado que muchas de ellas no suponen un riesgo real dentro del contexto académico. Las observaciones no afectan negativamente ni al funcionamiento ni al mantenimiento del sistema, y en su mayoría responden a decisiones de diseño deliberadas que priorizan la claridad y simplicidad del código. En consecuencia, se considera que no es necesario aplicar cambios en esta etapa.

# Contenido

* **Uso de excepciones genéricas y falta de validaciones específicas:** En ciertos servicios, como los relacionados con la gestión de cursos, se manejan excepciones genéricas sin diferenciar los tipos de error. Aunque este enfoque simplifica el código en un entorno académico, en un contexto de producción sería más adecuado emplear excepciones específicas para mejorar el diagnóstico de errores
* **Ausencia de validación frente a posibles nulos:** Algunos servicios no comprueban si ciertos valores (como una respuesta externa) son nulos antes de ser utilizados. Aunque esto podría derivar en excepciones, el comportamiento del sistema es predecible en este entorno controlado. No obstante, añadir comprobaciones podría mejorar la robustez general del código.
* **Métodos definidos sin implementación explícita:** Se han identificado métodos vacíos que no incluyen comentarios justificativos. Aunque estos métodos no son necesarios en la lógica actual, sería útil documentar la razón de su existencia para mejorar la comprensión del código a largo plazo.
* **Inyección directa de campos sin constructor:** En lugar de aplicar la inyección por constructor, se ha optado por la asignación directa de campos para mayor simplicidad. Esta elección, aunque no la más óptima desde el punto de vista de la inmutabilidad, es válida dentro del contexto y alcance del proyecto.
* **Declaración innecesaria de valores booleanos:** Se han detectado literales booleanos redundantes que no afectan la lógica del programa. Su eliminación mejoraría la legibilidad del código.
* **Advertencias sobre serialización no aplicables:** Se reportan algunos atributos que no son transient o serializables en clases que no persisten directamente en base de datos. Dado que se trata de datos temporales o no persistentes, estas advertencias no suponen un riesgo real.

# Conclusiones

El análisis realizado sobre los *bad smells* detectados refleja que, si bien existen ciertas prácticas que se desvían de las recomendaciones generales de estilo y diseño, ninguna de ellas compromete la funcionalidad ni la estabilidad del sistema en su contexto actual. La mayoría de las advertencias responden a decisiones conscientes orientadas a la simplicidad, legibilidad o integración con servicios externos.

Dado el alcance académico del proyecto y la naturaleza controlada de su ejecución, no se considera necesario aplicar correcciones inmediatas. No obstante, se proponen algunas mejoras que podrían adoptarse en futuras versiones para favorecer la escalabilidad y la mantenibilidad a largo plazo.

# Bibliografía

Intencionalmente en blanco.