Lint Report

C2.017

https://github.com/vicgrabru/Acme-SF-D03

Victor Graván Bru
vicgrabru@alum.us.es
Carlos García Ortiz
cargarort3@alum.us.es
Alberto Escobar Sánchez
albescsan1@alum.us.es
Jorge Gómez de Tovar
jorgomde@alum.us.es
Francisco de Asís Rosso Ramírez
frarosram@alum.us.es

Tabla de contenidos

Resumen ejecutivo

El informe lint detalla el resultado analiza exhaustivamente la calidad del código mediante el software SonarLint, identificando áreas de mejora y posibles riesgos de seguridad en el código desarrollado. Utilizando técnicas avanzadas de análisis estático, dinámico y de comportamiento, el informe evalúa la legibilidad, mantenibilidad y eficiencia del código, así como la seguridad y fiabilidad de la aplicación resultante.

Tabla de revisiones

Revisión	Fecha	Descripción
1.0	22-06-2024	Primera versión del documento

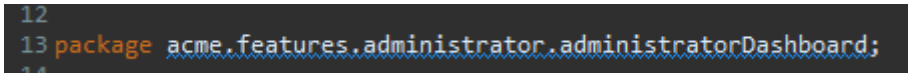
Introducción

En este informe, se detallan todos los “malos olores” detectados por SonarLint, y por qué no afectan realmente a la calidad del código, ni a la seguridad general del programa.

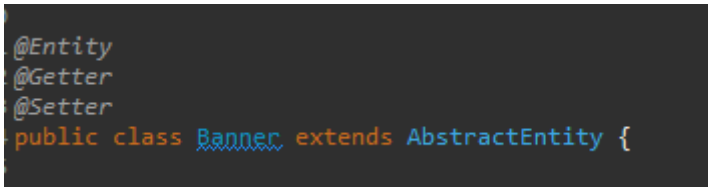
Contenido

Al analizar el proyecto con SonarLint, se detectan los siguientes malos olores:

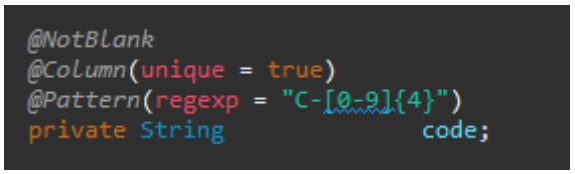
- **Nombres de algunos paquetes:** Se debería de utilizar una convención de nombres de los paquetes para facilitar la eficiencia de colaboración entre miembros del equipo. La convención sugerida por SonarLint implica no usar mayúsculas y por eso se marca como bad smell, pero como estamos usando camelCase para los paquetes el bad smell no es importante.



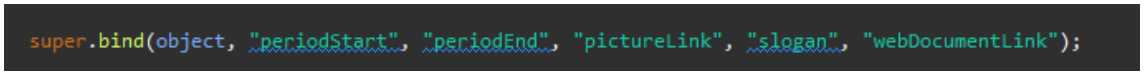
- **Subclase de una clase sin equals:** Las subclases que extienden de AbstractEntity son clasificadas como bad smell debido a que dicha clase padre contiene un método override equals, según SonarLint las subclases deberían poseer también un override equals. Sin embargo, el equals definido en AbstractEntity nos vale para su uso en el sistema así que no es necesario definir uno propio por clase.



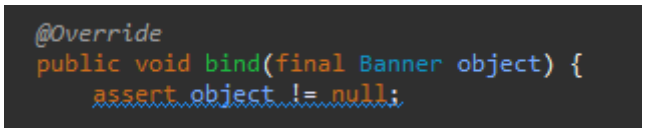
- **Uso de patrón "[0-9]":** El bad smell en cuestión consiste en que dicho patrón puede ser sustituido por el patrón "\\d" ya que se usan para exactamente lo mismo. Este bad smell no es grave ya que el patrón usado es válido.



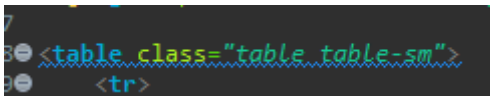
- **Repetición del mismo string:** En un mismo archivo se utiliza el mismo string múltiples veces, lo que podría solucionarse definiendo una variable con dicho string y usándola múltiples veces. Al no afectar de ninguna forma al código no se ha visto necesario corregirlo.



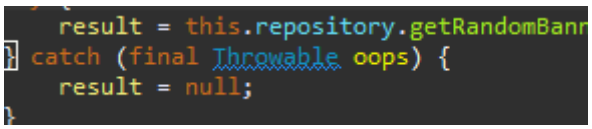
- **Asserts:** Los asserts marcados como bad smell validan parámetros de métodos públicos, cosa no recomendable. Además lanzan un error en lugar de una excepción. De todas formas este assert funciona de la forma que se desea así que no es necesario cambiarlo.



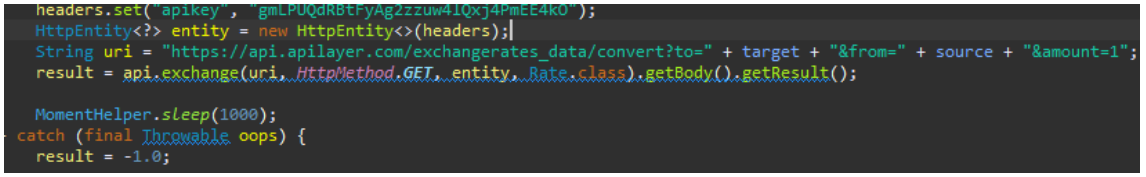
- **Descripción de Dashboard:** Se recomienda añadir una descripción al dashboard. Sin embargo, este mal olor no impacta de ninguna forma significativa al desempeño del propio dashboard por lo que no tiene una importancia elevada.



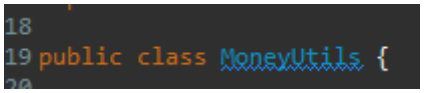
- **Throwable en try catch**: En el ControllerAdvice que selecciona un Banner al azar, SonarLint recomienda que utilicemos Exception en lugar de Throwable en el try-catch. Sin embargo, tal y como se ha enseñado en clase, y como aparece en las diapositivas y en el proyecto de ejemplo "Acme-Jobs", la clase correcta es Throwable. Este mal olor también se detecta en el repositorio en el que se hace la consulta a la api para el cambio de divisa. En ese caso se puede ignorar, ya que cuando se hacen peticiones a apis externas dentro de un try-catch, la clase correcta es Throwable.



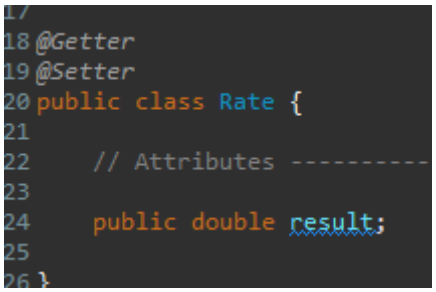
- **Excepción NullPointerException**: En el repositorio en el que se hace la petición a la api para el cambio de divisa, marca como mal olor que se está accediendo a un atributo del resultado de hacer una consulta a una api externa. Este resultado puede ser nulo, pero como está dentro de un try-catch se capturaría la excepción (SonarLint no lo detecta porque no se utiliza específicamente el tipo de excepción que lanzaría, aunque el tipo utilizado en el try-catch incluye ese tipo de excepción)



- **Añadir constructor privado**: Este mal olor salta porque creamos una clase sin crear explícitamente un constructor. Esto es irrelevante, ya que la clase no tiene atributos, y todos los métodos son estáticos, por lo que un objeto de esta clase sería inutil.



- **Atributo público en clase con getters y setters**: Este mal olor salta porque tenemos una clase con getters y setters, pero declaramos como público el único atributo que tiene. Este mal olor podemos ignorarlo, ya que esta clase es la que se utiliza para mapear la respuesta de la api de cambio de divisas, y es necesario que los atributos de la clase sean públicos.



Conclusión

En conclusión, es importante mantener un estándar de calidad a la hora de escribir código y producir software, para asegurar la legibilidad y mantenibilidad del mismo. Aunque SonarLint es una herramienta de gran utilidad para asegurar dicho estándar, no todo lo que marca como área de mejora tiene un efecto real sobre la calidad, siendo en ocasiones un compromiso necesario para el funcionamiento del programa.

Bibliografía

No aplica.