Lint Report

VIRGINIA MESA PEREZ
Repositorio: https://github.com/DP2-C1-021/Acme-ANS-C2
Student 2

ÍNDICE

Resumen Ejecutivo

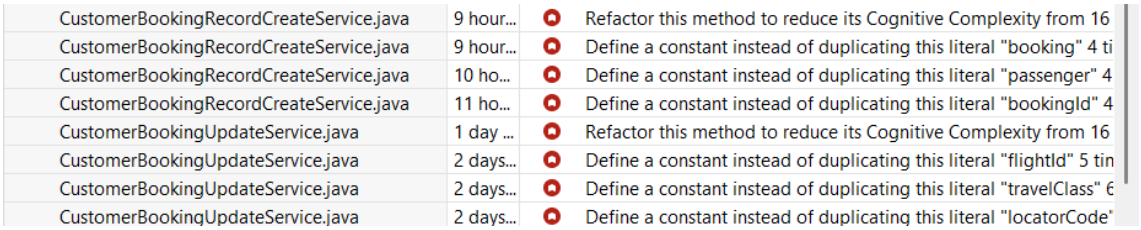
Este es el proyecto del grupo C2.021 sobre Acme ANS, compañía ficticia especializada en organizar y controlar las operaciones de los aeropuertos y cuya logística de vuelos se gestiona con un WIS.

Para el correcto funcionamiento del sistema se debe escribir un reporte de SonarLint donde se detallen los olores encontrados tras el análisis.

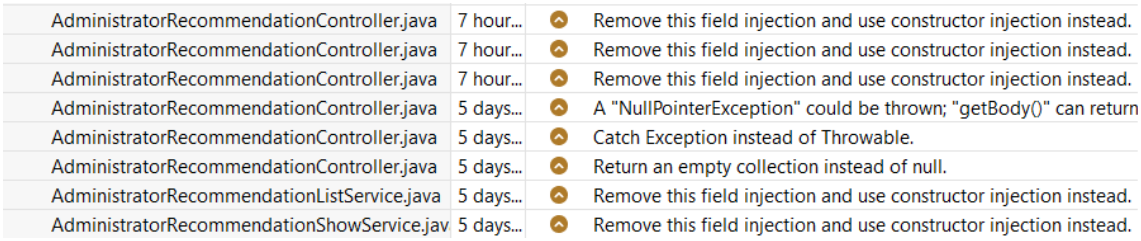
Introducción

En este reporte se expondrán los olores encontrados tras ejecutar el análisis de sonarLint en mi proyecto, teniendo en cuenta los resultados para los requisitos individuales del estudiante 2, tanto obligatorios como opcionales.

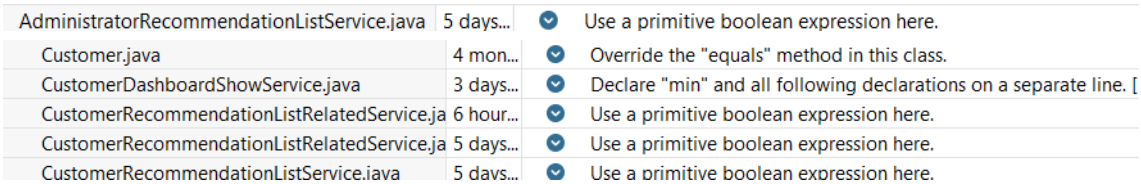
Contenidos



CustomerBookingRecordCreateService.java	9 hour...	Refactor this method to reduce its Cognitive Complexity from 16
CustomerBookingRecordCreateService.java	9 hour...	Define a constant instead of duplicating this literal "booking" 4 ti
CustomerBookingRecordCreateService.java	10 ho...	Define a constant instead of duplicating this literal "passenger" 4
CustomerBookingRecordCreateService.java	11 ho...	Define a constant instead of duplicating this literal "bookingId" 4
CustomerBookingUpdateService.java	1 day ...	Refactor this method to reduce its Cognitive Complexity from 16
CustomerBookingUpdateService.java	2 days...	Define a constant instead of duplicating this literal "flightId" 5 tin
CustomerBookingUpdateService.java	2 days...	Define a constant instead of duplicating this literal "travelClass" 6
CustomerBookingUpdateService.java	2 days...	Define a constant instead of duplicating this literal "locatorCode"



AdministratorRecommendationController.java	7 hour...	Remove this field injection and use constructor injection instead.
AdministratorRecommendationController.java	7 hour...	Remove this field injection and use constructor injection instead.
AdministratorRecommendationController.java	7 hour...	Remove this field injection and use constructor injection instead.
AdministratorRecommendationController.java	5 days...	A "NullPointerException" could be thrown; "getBody()" can return
AdministratorRecommendationController.java	5 days...	Catch Exception instead of Throwable.
AdministratorRecommendationController.java	5 days...	Return an empty collection instead of null.
AdministratorRecommendationListService.java	5 days...	Remove this field injection and use constructor injection instead.
AdministratorRecommendationShowService.jav	5 days...	Remove this field injection and use constructor injection instead.



AdministratorRecommendationListService.java	5 days...	Use a primitive boolean expression here.
Customer.java	4 mon...	Override the "equals" method in this class.
CustomerDashboardShowService.java	3 days...	Declare "min" and all following declarations on a separate line. [
CustomerRecommendationListRelatedService.ja	6 hour...	Use a primitive boolean expression here.
CustomerRecommendationListRelatedService.ja	5 days...	Use a primitive boolean expression here.
CustomerRecommendationListService.java	5 days...	Use a primitive boolean expression here.

Durante la revisión con SonarLint del código correspondiente a mis funcionalidades (Customer y Recommendation para administrador), se detectaron varios *bad smells*. A continuación, expongo los más relevantes y explico las razones por las cuales he decidido no realizar modificaciones en el código.

1. Complejidad Cognitiva excesiva

En las clases CustomerBookingRecordCreateService.java y CustomerBookingUpdateService.java, se detectó una complejidad cognitiva superior a lo recomendable. Si bien reducirla podría mejorar la legibilidad, estos métodos ya están organizados en bloques claros y cambiar su estructura podría afectar a la trazabilidad del flujo de trabajo que he diseñado. Considero que los métodos siguen siendo comprensibles y mantenibles.

2. Duplicación de literales

SonarLint ha reportado múltiples literales duplicados, como "booking", "passenger", "bookingId", "flightId", "locatorCode" o "travelClass". Aunque es cierto que declarar estos valores como constantes globales mejoraría la mantenibilidad en caso de cambios futuros, he optado por dejarlos en su forma literal ya que su aparición está contenida y limitada a una clase o método específico, lo que no compromete la claridad del código.

3. Uso de @Autowired en atributos

En varias clases del paquete AdministratorRecommendation, como AdministratorRecommendationController.java o AdministratorRecommendationListService.java, SonarLint ha señalado el uso de @Autowired directamente en atributos, recomendando usar inyección por constructor. Aunque la sugerencia es válida, he decidido mantener el patrón mostrado en los ejemplos docentes, como Acme-Jobs, para mantener la coherencia estructural del proyecto.

4. Excepciones genéricas

En AdministratorRecommendationController.java, se señala que capturar Throwable no es recomendable y se propone usar Exception en su lugar. Estoy de acuerdo con esta recomendación, pero el código base del que partimos ya realizaba esta captura de manera genérica y cambiarlo habría requerido rehacer parte del manejo de errores. Tomé nota de ello para futuras refactorizaciones.

5. Expresiones booleanas primitivas

En varias clases relacionadas con listas de recomendaciones de clientes y administradores (CustomerRecommendationListService.java, etc.), se señala que se podrían simplificar expresiones booleanas. Estos casos son mínimos y no afectan la lógica ni la legibilidad del código, por lo que no he considerado necesario modificarlo.

6. Ordenación de miembros y métodos

Se sugiere reorganizar métodos y miembros en clases como CustomerDashboardShowService.java y Customer.java para cumplir con convenciones de estilo. En este caso, se ha decidido no hacer cambios, dado que no afecta la funcionalidad ni genera errores de compilación, y seguir el orden actual facilita el trabajo colaborativo.

Conclusiones

Como conclusión, el reporte Lint es muy útil para poder visualizar bad smells en el código que puedan suponer problemas de mantenimiento o introducción de bugs en el futuro. Si bien aquí no le di demasiado uso al reporte como tal, sí considero que tiene mucha importancia (tanto la herramienta como el reporte) a la hora de llevar a cabo un proyecto software, pues la presencia de bad smells es determinante para un desarrollo en el que hacer cambios no suponga una rotura de la aplicación.