Lint Report

Grado en Ingeniería Informática - Ingeniería del Software

Diseño y Pruebas II

Curso 2023 - 2024

Código de Grupo: C1-001		
Autor	**Correo**	**Rol**
José María Baquero Rodríguez	josbaqrod@alum.us.es	Desarrollador
Pedro Pablo Santos Domínguez	pedsandom@alum.us.es	Desarrollador
Guillermo Gómez Romero	guigomrom@alum.us.es	Manager
Ángel Neria Acal	angneraca@alum.us.es	Desarrollador, operador
Manuel Vélez López	manvellop2@alum.us.es	Desarrollador

Repositorio: https://github.com/DP2-2023-2024-C1-001/Acme-SF-D03

Índice de Contenidos

1. Resumen Ejecutivo

Objetivo:

El propósito de este informe de análisis Lint realizado a través de Eclipse es ofrecer una evaluación exhaustiva del código fuente del proyecto, identificando áreas críticas para mejorar la calidad del código y minimizar posibles problemas técnicos. Se han analizado los estándares de codificación pertinentes, identificado posibles violaciones o errores, y se ofrecen recomendaciones para optimizar el código y garantizar un desarrollo más eficiente y sólido del proyecto.

Roles y Responsabilidades:

Desarrollador: Responsable de escribir código, realizar pruebas informales y mantener altos estándares de calidad en el desarrollo del proyecto.

Operador: Encargado de configurar el entorno de despliegue, implementar la aplicación y garantizar su correcto funcionamiento en producción.

Todos los Roles: Tienen la responsabilidad compartida de escribir informes sobre el trabajo realizado y colaborar en la mejora continua del proceso de desarrollo.

Plan de Acción:

Configuración del Proyecto: Los desarrolladores establecen la configuración del proyecto en Eclipse, asegurando que se sigan las mejores prácticas de codificación y se apliquen las reglas de linting adecuadas.

Análisis del Código: Se ejecuta el análisis Lint en el código fuente del proyecto utilizando las herramientas integradas en Eclipse.

Identificación y Corrección de Problemas: Se identifican posibles violaciones de estándares de codificación, errores o malas prácticas en el código, y se toman medidas correctivas para abordarlos.

Commit y Push: Una vez que se han corregido los problemas, se realiza un commit y push al repositorio remoto para mantener un registro de los cambios realizados.

Revisión del Código: Se crea una solicitud de revisión de código (pull request) y se asigna un revisor para validar los cambios realizados.

Aprobación y Merge: Después de que el revisor aprueba los cambios, se realiza el merge del código corregido con la rama principal del repositorio.

2. Control de Versiones

Fecha	Versión	Descripción
24/04/2024	V1.0	Creación inicial del documento.

3. Introducción

El siguiente informe de lint se centra en una evaluación exhaustiva de los resultados obtenidos del análisis estático realizado por SonarLint en nuestro proyecto. Como parte esencial de este proceso, se ha examinado cada uno de los "malos olores" reportados con el objetivo de identificar áreas de atención y tomar decisiones informadas para su corrección o justificación.

En síntesis, este informe de Lint se concentra en la evaluación exhaustiva de los problemas de código identificados, proporcionando recomendaciones y soluciones para su mejora. A través de una estructura clara y sistemática, se busca facilitar la toma de decisiones informadas y promover el avance del proyecto en cuestión.

La estructura de este documento comprende una presentación detallada de los "malos olores" detectados por SonarLint en nuestro proyecto, junto con una justificación de por qué se consideran inofensivos. En primer lugar, se proporcionará un listado exhaustivo de estos "malos olores", identificando cada uno de ellos y explicando su naturaleza. A continuación, se incluirán argumentos sólidos que respalden la conclusión de que estos problemas son inocuos en el contexto específico del proyecto. Esta estructura permite una comprensión completa de los desafíos identificados por SonarLint y las decisiones adoptadas para abordarlos o justificar su no intervención

4. Contenido

Listado de Bad Smells:

- ### Replace this assert with a proper check:

  Sugiere que se está utilizando un assert de manera incorrecta en el código. Los assert son utilizados típicamente para verificar suposiciones que el programador cree que siempre serán ciertas durante la ejecución del programa. Sin embargo, los assert no deben ser utilizados para la validación de entrada de usuario o para la verificación de condiciones que puedan fallar debido a situaciones normales durante la ejecución del programa.

  Para corregir este "bad smell", se debería reemplazar el assert con una verificación adecuada de la condición. Esto podría implicar utilizar una estructura de control como un if para verificar la condición y manejarla de manera apropiada si no se cumple. Además, dependiendo del contexto, se podrían tomar medidas adicionales como registrar un error, lanzar una excepción o proporcionar retroalimentación al usuario sobre la condición que

no se cumplió.Este bad smell se repite en las clases: **AdministratorBannerCreateService.java,AdministratorBannerDeleteService.java,AdministratorBannerShowService.java,AdministratorBannerUpdateService.java**

AdministratorObjetiveCreateService.java,AdministratorObjetiveListService.java,AdministratorObjetiveShowService.java

AdministratorRiskCreateService.java,AdministratorRiskDeleteService.java,AdministratorRiskListService.java,AdministratorRiskShowService.java,AdministratorRiskUpdateService.java

AdministratorSystemConfigurationUpdateService.java,AdministratorSystemConfigurationShowService.java

- ## Define a constant instead of duplicating this literal “variable” 4 times:

  Se nos indica que creemos una constante global en vez de usar el literal “variable”, este bad smell no llega a ser nocivo ya que se trata de una clase con pocas líneas de código y solo se usa un total de 4 veces, en el caso de que consideramos clases de mayor tamaño donde dicho literal formará parte de más usos, en este caso sería aconsejable crear dicha constante, ya que en el caso de tener que modificar dicha constante solo deberíamos actualizarla en un único sitio.Esté bad smell se repite en las clases:

  AdministratorBannerCreateService.java,AdministratorBannerUpdateService.java,AdministratorObjetiveCreateService.java,AdministratorRiskCreateService.java y AdministratorRiskUpdateService.java

- ## Use a StringBuilder instead:

  Aquí se nos indica el uso ineficiente de la concatenación de cadenas de caracteres, especialmente en bucles o situaciones donde se están realizando múltiples concatenaciones. Esto puede causar un impacto negativo en el rendimiento y la utilización de memoria, ya que cada concatenación crea un nuevo objeto de cadena.

  La solución recomendada es utilizar StringBuilder en lugar de la concatenación directa de cadenas. StringBuilder es una clase diseñada específicamente para construir cadenas de caracteres eficientemente, especialmente cuando hay una gran cantidad de concatenaciones o en bucles. StringBuilder minimiza la cantidad de objetos de cadena creados y optimiza el rendimiento de la construcción de cadenas.Esté bad smell se da en la clase: **AdministratorSystemConfigurationUpdateService.java**

5. Conclusión

En conclusión, este informe de análisis Lint ha sido una herramienta invaluable para identificar y abordar los problemas de calidad en nuestro proyecto. A través del exhaustivo análisis estático realizado por SonarLint, hemos identificado varios "malos olores" en nuestro código. Si bien algunos de estos "malos olores" requerían acciones correctivas, fueron considerados inofensivos o de bajo impacto en el contexto específico del proyecto.

Nuestro enfoque para abordar estos problemas ha sido sistemático y reflexivo. Si bien hemos identificado los "malos olores" reportados por SonarLint, hemos optado por no realizar cambios en el código, ya que no era necesario. Los desarrolladores han evaluado cuidadosamente las recomendaciones proporcionadas por SonarLint y han concluido que no era preciso realizar modificaciones en este momento. A pesar de no realizar cambios en el código, hemos establecido un plan de acción claro para estar preparados en caso de que sea necesario en el futuro. Esto incluye la configuración del proyecto, la corrección de problemas específicos y la revisión del código.

En última instancia, este informe subraya la importancia de utilizar herramientas de análisis estático como SonarLint para mejorar continuamente la calidad del código y fortalecer el proceso de desarrollo de software. Al adoptar un enfoque proactivo para identificar y abordar los problemas de calidad, podemos garantizar un desarrollo más eficiente y sólido del proyecto, beneficiando a todos los involucrados en su desarrollo y mantenimiento.

6. Bibliografía

Intencionadamente en blanco.