LINT REPORT D03

INFORMACIÓN DE GRUPO

Group: C2.043

Repository: https://github.com/alepingar/Acme-Sf-C2.043

Realizado por:

Student #4
UVUS: alepingar
Contact: alepingar@alum.us.es

Miembros restantes:

Student #2
UVUS: alvmarmun1
Contact: alvmarmun1@alum.us.es

Date: Sevilla Febrero 04,07, 2024

TABLA DE CONTENIDO

Portada……1

Tabla de Contenido……2

Resumen Ejecutivo……3

Tabla de Revisiones……4

Introducción……5

Contenidos……6

Conclusiones……7

Bibliografía……8

RESUMEN EJECUTIVO

Este informe proporcionará una lista de problemas detectados por Sonar Lint en el proyecto, junto con explicaciones de por qué podrían no ser motivo de preocupación. Se utilizará un lenguaje claro y directo para asegurar una fácil comprensión y contribuir a la creación de un producto final de alta calidad.

TABLA DE REVISIONES

Revisión	Fecha	Descripción
1.0	2024-04-23	Introducción, resumen ejecutivo, contenidos, conclusiones y bibliografía

INTRODUCCIÓN

Utilizaré el plugin Sonar Lint de Eclipse para analizar todos los archivos que he creado como Estudiante 4 para mis tareas individuales. Específicamente, revisaremos el contenido de los paquetes Java llamados "entities.sponsorships", "features.sponsor", "features.any.sponsorship", y "features.authenticated.sponsor". Además, analizaremos el rol de Sponsor y el formulario SponsorDashboard. Los archivos de pobladores como Sponsorship, Invoice y Sponsor también serán analizados, junto con los archivos de vista (forms.jsp, list.jsp y archivos i18n) encontrados en las carpetas "views/sponsor", "views/any/sponsorship" y "views/authenticated/sponsor". Dado que la mayoría de los problemas de código son comunes a varios archivos, analizaré cada mal olor de código uno por uno en lugar de clase por clase.

CONTENIDOS

- Sobrescribe el método "equals" en esta clase: Esta sugerencia surge porque es una práctica común sobrescribir el método "equals" en las clases Java. Sin embargo, dado que no se utiliza en nuestra implementación y está comentado en la clase, no es necesario realizar ninguna corrección.

- Elimina el literal booleano innecesario: Este mal olor se debe a validaciones que verifican si una entidad está publicada: "object.isPublished() == false". Es cierto que se podría escribir simplemente "!object.isPublished()", pero dado que forma parte de una afirmación más amplia, considero que es más legible utilizar "== false". En cualquier caso, esto ocurre solo un par de veces y creo que vale la pena una expresión un poco más larga en aras de la comprensibilidad.

- Utiliza la sintaxis concisa de la clase de caracteres '\d' en lugar de '[0-9]': Esta recomendación se refiere al código, como atributos. Sin embargo, la implementación actual que utiliza '[0-9]' en la expresión regular es válida y argumentablemente más comprensible para representar rangos de números latinos. Por lo tanto, no es necesario realizar ninguna corrección.

- Renombra este nombre de paquete para que coincida con la expresión regular '^[a-z_]+(.[a-z_][a-z0-9_])$': Aunque hay una sugerencia para renombrar el paquete, el nombre actual cumple con el estándar del marco de trabajo y sigue lo enseñado por los profesores. Por lo tanto, no es necesaria ninguna corrección.

- Define una constante en lugar de duplicar este literal: Esta recomendación sugiere definir una constante en lugar de duplicar un valor literal varias veces en el código. Sin embargo, en este caso, duplicar el literal puede no ser un problema significativo. La justificación para su naturaleza inocua es que el literal se utiliza solo en unos pocos lugares (como máximo 2 o 3 veces).

- Reemplaza este assert con una verificación adecuada: El assert 'object != null;' está marcado, pero es la implementación recomendada tanto por los profesores como por el marco de trabajo. Por lo tanto, no es necesario realizar ninguna modificación.

CONCLUSIONES

En conclusión, aunque el análisis de código ha señalado varias áreas potenciales de mejora, tras un resumen más detenido, muchas de estas sugerencias se consideran inofensivas o incluso preferibles según el contexto específico del proyecto.

Por ejemplo, sugerencias como sobrescribir el método "equals", utilizar la sintaxis concisa de clases de caracteres y renombrar nombres de paquetes se consideran innecesarias dado que están alineadas con los estándares del proyecto y las prácticas enseñadas por los profesores.

Además, las decisiones de mantener ciertas estructuras de código, como usar comparaciones booleanas explícitas o duplicar literales, se justifican por su contribución a la legibilidad del código y es importante mencionar que solo ocurren un par de veces.

En resumen, si bien es esencial considerar y evaluar las sugerencias de código, también es igualmente importante sopesarlas con el contexto y los estándares del proyecto para determinar el curso de acción más apropiado. En este caso, las prácticas de código existentes parecen adecuadas para las necesidades del proyecto.

BIBLIOGRAFÍA

Intencionalmente en blanco.