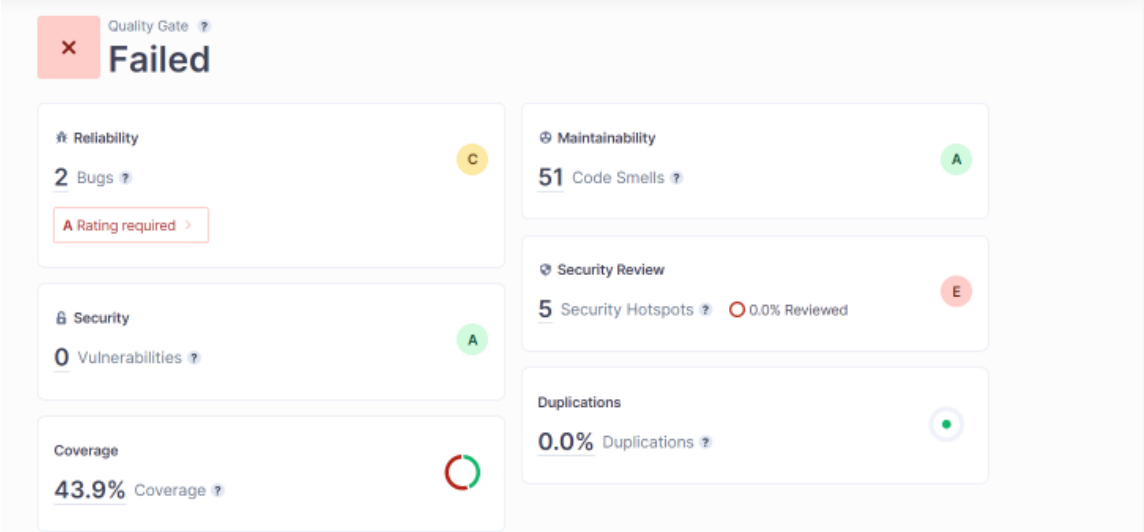
INFORME DE CALIDAD Nº2 – SPRINT 02

Autor: Mario Martín Pérez

ANÁLISIS 4 NOVIEMBRE 2022

CAPTURA



INCIDENCIAS

El análisis realizado no pasa los criterios de calidad esperados por la organización, debido la calificación de la sección *Reliability*. En este caso, para cumplir con las expectativas de la organización, sería necesario obtener una calificación A en dicho apartado. Esto puede lograrse solucionando los bugs presentes en la aplicación, los cuales se mencionan a continuación:

- **Bug en PromocionesArrayAdapter**: Esta clase, hace uso del método *toLowerCase*. No obstante, no utiliza el valor de la conversión realizada en ningún momento. Por lo tanto, no tiene sentido utilizar dicha función, ya que además si realmente fuera necesaria pero no se hiciera uso de su resultado, se estaría dando lugar a un comportamiento erróneo del código programado. En base a estos motivos, se considera el bug con una severidad *Major* que, si bien no llega a ser crítica, sería recomendable igualmente ponerle solución lo antes posible.
- **Bug en Promoción**: En esta clase, contamos con un método *equals* definido, pero no un método *hashCode*. Por convención del lenguaje Java, siempre que se defina uno de estos dos métodos, debería definirse también el restante, de forma que devuelvan valores coherentes entre sí. Es decir, si *equals* afirmara que dos objetos son iguales,

hashCode debería retornar para estos también el mismo valor entero. Dado que puede no suponer ningún fallo si no se usa el método, el bug se categoriza con una severidad menor. Aún así, es recomendable su subsanación, para lograr cumplir así con los criterios esperados por la organización.

En cuanto a problemas de mantenibilidad se refiere, si bien hay un mayor número de *code smells* a revisar, seguimos cumpliendo con el rango esperado. No obstante, a continuación, se ofrece un análisis de los diferentes problemas encontrados de mayor a menor severidad, con el fin de no postergar su solución, ya que esto podría ser más complicado según se aplace.

BLOCKER:

- **Clase EliminarPromotionUITest**: El test ejecuta acciones que prueban la interfaz gráfica. Sin embargo, no se realiza ningún tipo de aserción, lo que hace que el test carezca de utilidad, ya que no se logra comprobar el correcto funcionamiento. Urge su arreglo, ya que en caso de haber un error en aquello que se prueba, no lo estaríamos detectando.

En total, acumulamos 10 minutos de deuda técnica en este nivel de severidad.

CRITICAL:

- **Clase ListaPromocionesView:** La clase cuenta con un método *init* vacío que no realiza ninguna acción. Dado que parece existir únicamente por inercia y no cumple con ningún objetivo, es mejor eliminarlo para evitar posibles comportamientos anómalos en el futuro y facilitar la mantenibilidad.

En total, acumulamos 5 minutos de deuda técnica en este nivel de severidad.

MAJOR:

- **Clase GasolineraDetailPresenter:** Contiene un atributo privado del cuál no se hace uso, por lo que su eliminación resultará útil de cara a la mantenibilidad del código en el futuro.
- **Clase ListaPromocionesView:** Por lo general, se acumulan posibles mejoras, tales como el uso de funciones lambda, así como la eliminación de asignación a variables de valores que no se utilizan. No darán lugar a errores, pero pueden afectar a la productividad y mantenibilidad.
- **Clase PromocionesArrayAdapter**: El *code smell* detectado, radica en la no utilización del valor retornado por la función *toLowerCase*, tal y como se ha comentado previamente en la descripción de los bugs encontrados.
- **Clase GasolinerasService**: Se anidan expresiones, lo que dificulta la legibilidad del código y no se parametriza la llamada a *CallbackAdapter*, lo que, en caso de despistes, puede dar lugar a código inseguro y que pueda fallar.
- **Clases de test para ListaPromocionesPresenter**: En varias ocasiones se utilizan comparaciones dentro de *assertTrue*, lo cuál puede sustituirse directamente por el uso de *assertEquals*.

En total, se acumula un total de 1 hora y 3 minutos de deuda técnica en este nivel de severidad.

MINOR:

- **Clase GasolineraDetailPresenter**: Cuenta con *imports* no usados o atributos que únicamente se usan de forma local en un método. No afectan apenas a la mantenibilidad ni provocan errores, pero su modificación ayuda a mejorar la calidad del código de nuestro producto.
- **Clase GasolineraDetailView**: Cuenta con una amplia cantidad de *imports* no usados. De nuevo, no suponen ninguna amenaza, pero eliminarlos favorece la calidad del código y además no requieren apenas tiempo de esfuerzo.
- **Clase IGasolineraDetailContract**: Posee el mismo problema de uso de *imports.*
- **Clase IListaPromocionesContract:** Posee el mismo problema de uso de *imports*.
- **Clase ListaPromocionesView:** Posee el mismo problema de uso de *imports*.
- **Clase PromocionesArrayAdapter:** Posee el mismo problema de uso de *imports.*
- **Clase ListaPromocionesPresenter:** Posee el mismo problema de uso de *imports*. Además, se usa una estructura *if-else* con varios *returns*, la cual puede simplificarse en una línea de código favoreciendo la legibilidad. También se recomienda el uso de métodos más característicos de Java como *isEmpty,* en lugar de comparar un tamaño igual a cero manualmente. De nuevo, nada de esto supone ningún riesgo, pero da margen para aumentar la calidad del código implementado.
- **Clase Promoción**: Posee el mismo problema de uso de *imports*. También ocurre el mismo problema comentado con anterioridad con respecto a la no definición de un método *hashCode*, existiendo un método *equals*.
- **Otros errores menores**: Repartidos por más clases, se encuentran también más problemas con el uso de *imports* no útiles, uso de booleanos redundantes, o la posibilidad de utilizar expresiones que se adhieran más al estándar establecido para Java. Nuevamente, esto no afecta al funcionamiento del código, pero permite facilitar su mantenimiento y entendimiento.

En total se acumula 1 hora y 32 minutos de deuda técnica, fácilmente reducible, ya que la mayoría proviene del mal uso de *imports*.

INFO:

No son fallos en la funcionalidad de la aplicación ni problemas de calidad, pero su solución siempre resulta positiva. Principalmente, se trata de anotaciones TODO o del uso de algunos métodos obsoletos.

Acumulan un total de 37 minutos de deuda técnica.

Finalmente, en cuanto a puntos problemáticos de seguridad se refiere, no se han detectado nuevos problemas, salvo los ya existentes relacionados con el uso de preferencias en Android. Dado que apenas se aplican para guardar selecciones de ítems como los filtros, no se requiere tener en cuenta ningún tipo de encriptación, por lo que no se contemplan modificaciones en este apartado.

Para terminar con el informe de las incidencias detectadas, se adjunta a continuación un resumen de la deuda técnica acumulada por cada clase hasta el momento:

- ListaPromocionesView: 1 hora y 10 minutos.
- GasolineraDetailView: 16 minutos.
- GasolineraDetailPresenter: 14 minutos.
- PromocionesArrayAdapter: 14 minutos.
- CallRunnable: 10 minutos.
- EliminarPromotionUITest: 10 minutos.
- GasolinerasRepository: 10 minutos.
- GasolinerasService: 10 minutos.
- IListaPromocionesContract: 6 minutos.
- IListaPromocionesPresenter: 6 minutos.
- ListaPromocionesPresenterITest: 4 minutos.
- ListaPromocionesPresenter: 4 minutos.
- GasolinerasServiceConstants: 2 minutos.
- IGasolineraDetailContract: 2 minutos.
- MarcaDao: 2 minutos.
- Promocion: 2 minutos.

Total: 3 horas y 2 minutos.

PLAN DE ACCIÓN

1) Arreglar el bug de la clase PromocionesArrayAdapter, reduciendo la deuda técnica en 10 minutos.
2) Arreglar el bug de la clase Promoción, logrando así finalmente cumplir con los requerimientos de calidad de la organización y reduciendo la deuda técnica otros 15 minutos.

A continuación, se detallan tareas complementarias para aumentar la calidad del código, aunque sin ser imprescindibles, ya que a partir de este punto se cumpliría con los requisitos de calidad estipulados.

3) Solucionar el *code smell* de severidad *blocker* en la clase EliminarPromotionUITest, que aporta 10 minutos de deuda técnica y puede llevar a no detectar fallos en el test de interfaz para eliminar promociones.
4) Solucionar el *code smell* de severidad *critical* en la clase ListaPromocionesView, que aporta 5 minutos de deuda técnica. De esta forma, logramos evitar los problemas más relevantes y sin un esfuerzo en tiempo considerable.
5) Solucionar los *code smells* de severidad *major*, en especial de la clase ListaPromocionesView, ya que acumula una deuda técnica elevada con respecto al resto de elementos de la aplicación y que sería conveniente disminuir cuanto antes. Además, a estas alturas del proceso, habremos conseguido eliminar ya todos los *code smells* relativos a algunas clases, especialmente las de prueba.
6) Solucionar los *code smells* de severidad *minor*. En este punto, se hará principal hincapié en la limpieza de *imports,* que permitirá reducir la deuda en torno a 46 minutos, de una forma bastante rápida.
7) Finalmente, si se dispone del tiempo suficiente, se corregirán los errores relacionados con el uso de métodos obsoletos o la utilización de valores asignados a variables no utilizadas, pudiendo reducir la deuda técnica aún más de media hora.