PS17 – Informe de calidad: Historia de usuario “Consultar próximas llegadas”

Resumen

El presente documento recoge el análisis de calidad asociado a la historia de usuario #241952 – “Consultar próximas llegadas”, así como la descripción del proceso de corrección realizado.

Document ID:	PS17/00/2017-QR006-US241952-ConsultarPróximasLlegadas
Departamento:	Calidad y Auditoría, dentro del proyecto integrado
Tipo:	Análisis de calidad
Privacidad:	CONFIDENCIAL
Estado:	ENTREGABLE
Versión:	1.0.3
Fecha:	29/11/2017
Autores:	Solar Iglesias, Fernando
Revisores:	Martínez Vila, Javier; Cerezo Fernández, Elsa

HISTORIAL DE CAMBIOS

Versión	Fecha	Cambio	Responsable
1.0.0	21/11/2017	Creación del documento.	Solar Iglesias, Fernando
1.0.1	29/11/2017	Redacción del informe.	Solar Iglesias, Fernando
1.0.2	29/11/2017	Revisión gramatical y ortográfica.	Martínez Vila, Javier
1.0.3	29/11/2017	Revisión final.	Cerezo Fernández, Elsa

1. Introducción

El presente documento contiene el proceso de análisis de calidad llevado a cabo para la historia de usuario **#241952 – Consultar próximas llegadas**.

2. Desarrollo del proceso de análisis de calidad

2.1. Análisis inicial

Para efectuar el análisis de calidad de la historia de usuario indicada como objetivo, y realizar las correcciones pertinentes, se ha seguido el proceso que se describe a continuación.

En primer lugar, se procedió a subir el código del proyecto al servidor *SonarCloud*, cuyo análisis inicial mostró los siguientes resultados:

- **Deuda técnica**: un día.
- **Medidas e *issues***:
  - Un bug de criticidad menor.
  - Sesenta y cinco code smells de diversa criticidad.
- **Calificaciones:**
  - Bugs: B.
  - Vulnerabilities: A.
  - Code smells: A.

Una vez analizados los resultados obtenidos, se procedió a realizar las correcciones más convenientes según la relación entre mejora, esfuerzo y tiempo, para aumentar la calidad del código desarrollado.

El plan de acción seguido ha sido el de dar prioridad, en primer lugar, al *bug*, ya que hacía que la calificación fuese "B" en lugar de "A" y, en segundo lugar, a los code smells. Dentro de estos últimos, se han priorizado los de severidad *major*, posteriormente los de severidad *minor* y, en último lugar, los de severidad *info*.

Dentro del orden de prioridades citado en el párrafo anterior, se ha tenido en cuenta la proporción óptima entre el coste temporal de la resolución de cada *issue*, y el impacto en la calidad del mismo, de manera tal que algunos *code smells* de severidad *info* no han sido tratados debido al tiempo necesario para solventarlos y su bajo impacto en la calidad del código final.

2.2. *Issues* y acciones correctivas efectuadas

A continuación, se adjunta un listado con los *issues* que, según el criterio previamente explicado, resultaban más relevantes y, por tanto, han sido tenidos en cuenta de manera prioritaria al ejecutar acciones correctivas. Se han unificado aquellos *issues* y medidas adoptadas de igual procedencia/tratamiento.

2.2.1. Bugs

Issue	Severidad	Solución adoptada
Clase ListEstimacionesAdapter.java, línea cincuenta y tres. Es conveniente realizar un *cast* al realizar la división empleando un *float* (actualmente se divide entre sesenta con formato *Integer*).	Minor	Se ha realizado el *cast* sustituyendo el número anteriormente tratado como *Integer* por su equivalente en *float*.

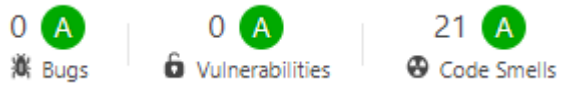
2.2.2. Code Smells

Issue	Severidad	Solución adoptada
Clase ParserJSON.java, líneas cincuenta y nueve y ciento veintiuno. La complejidad cognitiva de los métodos en dichas líneas es excesivamente elevada.	Critical	Se ha sustituido los bloques if/else if con multicondición de variable por su equivalente empleando switch.
Clases ParserJSON.java (línea ciento cuarenta y ocho) y LoadDataAsync.java (líneas sesenta y cuatro y ciento nueve). Se repite en varias ocasiones el mismo valor literal.	Critical	Se han definido constantes con el valor de dichos literales para evitar redundancias.
Clases Database.java, ListEstimacionesPresenter.java, LoadDataAsync.java, EstimacionesFragment.java y ListEstimacionesAdapter.java. Existen comentarios en el código que contienen código funcional.	Major	Se han eliminado las líneas de código que contenían código funcional.
Clase Database.java, línea quinientos cincuenta y uno. Se crea una variable y se le asigna un valor realizando una llamada a un método, y dicha variable no se utiliza.	Major	Se ha eliminado la variable.
Clase ListEstimacionesPresenter.java, líneas veintidós y veintiséis. Existen dos variables que no se utilizan (“ERROR” y “remoteFetchEstimaciones”).	Major	Se han eliminado las variables.
Clase EstimacionesFragment.java, línea ochenta y ocho. Existe una variable (“toast1”) que no se utiliza.	Major	Se ha eliminado la variable.
Clase ListEstimacionesAdapter.java. No debe utilizarse Math.round si no es necesario.	Major	Se ha eliminado la llamada al método.

Clase Estimacion.java, ListEstimacionesPresenter.java, EstimacionesFragment.java, GruposFragment.java y ParadasFragment.java. Existen imports que no se utilizan.	Minor	Se han eliminado los imports no utilizados.
Clase EstimacionesFragment.java, línea ciento cuarenta y siete. El nombre del método no cumple con los estándares sintácticos establecidos.	Minor	Se ha renombrado el método para que cumpla con las reglas predefinidas.
Clase GruposFragment.java, línea sesenta. Existe una variable local que no se utiliza ("i").	Minor	Se ha eliminado la variable.
Clase EstimacionesFragment.java, línea setenta y dos. No es necesario inicializar un ArrayList indicando dos veces su tipo de objetos (operador diamante).	Informativo	Se ha eliminado la segunda mención al tipo de objeto.

2.3. Análisis final, *issues* pendientes y conclusiones

La corrección de todos los *issues* citados anteriormente ha permitido que finalmente se obtengan los resultados siguientes:



- Siete horas de deuda técnica.
- Veintiún *code smells*, de los cuales cinco son de severidad *major* y dieciséis son de severidad *info*.

Puede apreciarse que el número de *issues* ha descendido considerablemente con respecto a la versión inicial. De igual manera, puede remarcarse que la deuda técnica ha sufrido un descenso de gran importancia, pues ha pasado de un día a siete horas. También se han mejorado las calificaciones y reducido, hasta casi eliminar, los errores más importantes y con mayor impacto en la mantenibilidad de la aplicación. Por tanto, puede valorarse el impacto de las medidas correctivas efectuadas como muy positivo.

Se listan a continuación los *issues*, que han quedado por el momento sin tratar debido al bajo impacto en la calidad del producto y el alto esfuerzo temporal que suponen, con el fin de que en revisiones posteriores sean considerados.

Issue	Severidad	Tipo
En varias clases, existen bloques de código duplicados. Este *issue* no se ha tratado debido a que, a pesar de que podría resultar relevante, precisa de una profunda restructuración del código implementado que no es posible en el punto actual del proyecto.	Major	Code smell
La llamada a *Thread.sleep(3000)* no debería realizarse al programar un test. Sin embargo, viene generado y recomendado por la propia herramienta de pruebas utilizada. Este *issue* no se ha tratado debido a la complejidad que supone implementar una solución alternativa.	Info	Code smell
El uso de *progress dialog* se considera obsoleto (*deprecated*). Este *issue* no se ha tratado debido a que se ha considerado de muy baja relevancia.	Info	Code smell

3. Sumario

En este documento se ha recogido el proceso seguido, los pasos analíticos y las acciones correctivas llevadas a cabo para mejorar la calidad del código asociado a la historia de usuario "Consultar próximas estimaciones". Además, se ha observado como la aplicación de estas acciones ha permitido incrementar de manera positiva las calificaciones en los distintos apartados vistos.