

# Informe de Calidad II (Sprint 2)

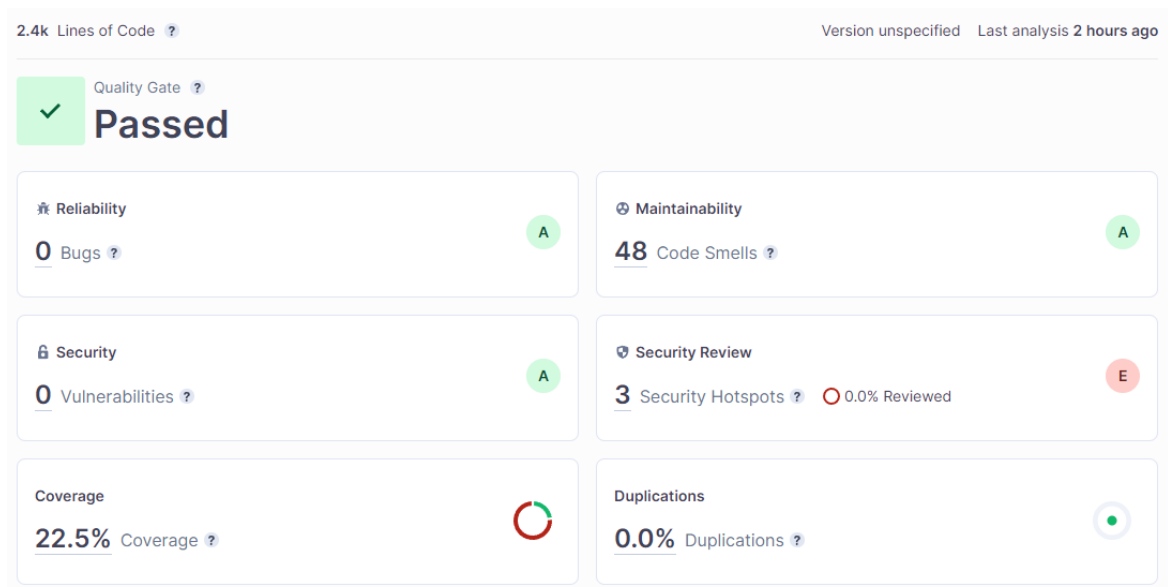
**Autores:** Marcos Fernández Alonso

## ANÁLISIS 1 DE NOVIEMBRE 2022

### CAPTURA



### VISTA RESUMIDA



### INCIDENCIAS

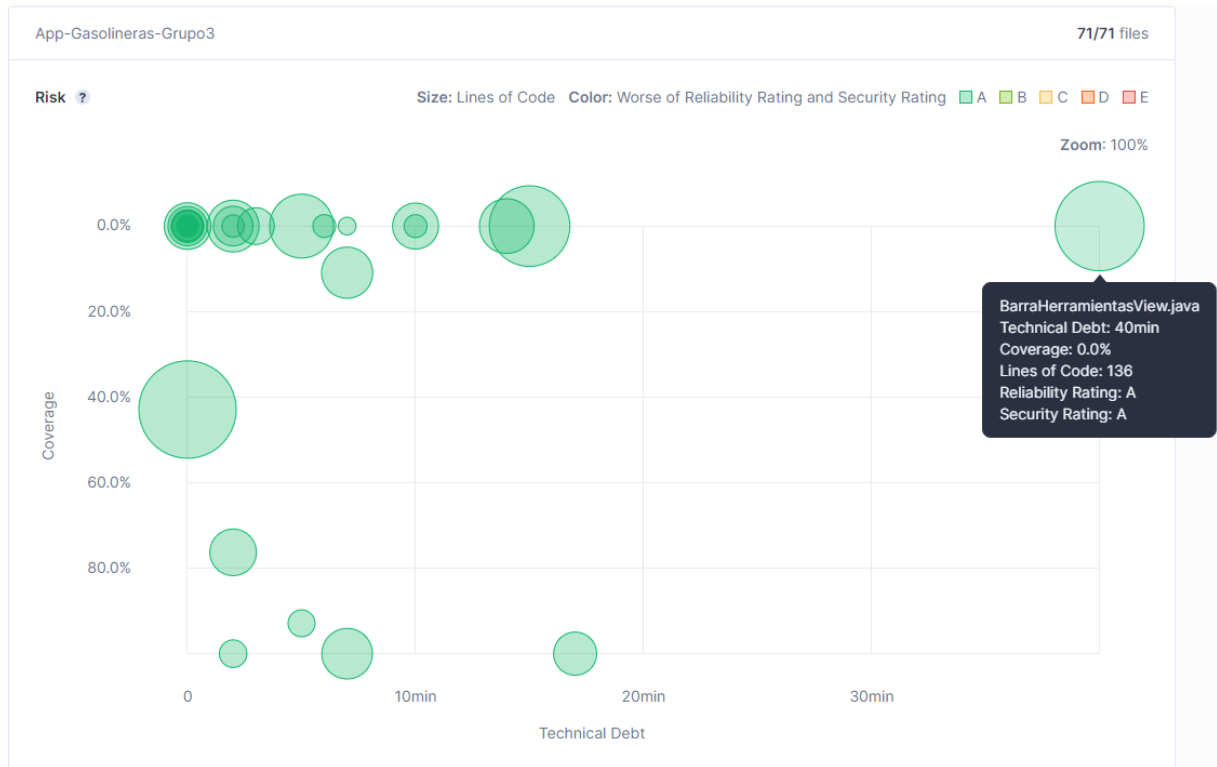
El análisis muestra que la versión actual de la aplicación si pasa los criterios de calidad de la organización. Esto se debe a que no ha encontrado ningún bug en el código, el número de code smells no es excesivamente alto respecto al número de líneas de código y tampoco ha encontrado ninguna vulnerabilidad en el apartado de seguridad, además no hay duplications y solo suspenden los security hotspots aunque tienen una prioridad baja.

Como ya se ha dicho previamente el número de code smells no es excesivo, pero hay que intentar mantenerlo al mínimo para aumentar tanto la calidad, como legibilidad y mantenibilidad de la aplicación. Afortunadamente solo hay un code smell bloqueante, tres críticos y uno mayor, siendo la gran mayoría de tipo minor e info lo que facilita solucionarlos.

# Informe de Calidad II (Sprint 2)

**Autores:** Marcos Fernández Alonso

Finalmente podemos comprobar que hay 3 security hotspots que nos indican posibles amenazas de seguridad las cuales habrá que analizar para determinar si son verdaderamente una amenaza. Y vemos que ya tenemos cubierto el 22.5% del código con nuestros tests.



Se observa que el total de deuda técnica es de 2h y 50 min lo que no está mal puesto que en el proyecto hay 2.4k líneas de código y más de cien horas invertidas. Aun así, si observamos un poco más en detalle, podemos ver que la clase BarraHerramientasView.java tiene una deuda técnica muy alta, esto se debe a que contiene varios code smell de los cuales uno es crítico, el cual se estipula que cambiarlo podría llevar unos 30 minutos, y otro blocker que también podría llevar más de lo normal. Sin embargo, la deuda técnica general del proyecto está repartida equitativamente.

## PLAN DE ACCIÓN

### 1. Analizar y resolver Security HotSpots

El primer security hotspot se viene arrastrando desde el anterior análisis de calidad y de momento se sigue considerando que no es necesario usar una base de datos encriptada para proteger los datos que estamos guardando.

Ahora han aparecido dos nuevos: el ACCESS\_COARSE\_LOCATION y ACCESS\_FINE\_LOCATION. Necesitamos acceder a la localización y en nuestro caso además buscamos que sea lo más precisa posible así que usamos el ACCESS\_FINE\_LOCATION para que utilice el gps también. Por lo tanto, al usar FINE\_LOCATION también tenemos que definir los permisos de COARSE\_LOCATION

### 2.Tratar el code smell de tipo blocker

# Informe de Calidad II (Sprint 2)

**Autores:** Marcos Fernández Alonso

En la vista de la barra de herramientas encontramos un code smell blocker dentro de un switch en el que se nos explica que la cláusula default siempre se ejecuta si no tenemos un return o un break dentro de los case, en este caso el fallo está en que el último case le falta su return, añadiéndolo solventamos el problema y reducimos la deuda técnica de la vista de la barra de herramientas. En 10 minutos

## 3. Solventar code smells críticos

Los dos primeros code smells críticos están en el MainView y son por usar ActivityCompact cuando llamamos a checkSelfPermission() y de acuerdo con sonar cloud deberíamos usar ContextCompact porque es un miembro estático y no debería ser llamado por tipos derivados como ActivityCompact. Esto reducirá la deuda técnica en 10 minutos.

En cuanto al code smell crítico de la vista de la barra de herramientas se arrastra desde el plan de pruebas anterior y se sigue manteniendo que debe ser ignorado dado que es imprescindible para la correcta funcionalidad.

## 4. Tratar code smell tipo mayor


Este code smell nos explica que las clases de utilidades son colecciones de métodos estáticos y que deberían tener un constructor privado puesto que nunca deberían ser instanciadas. Se soluciona añadiendo el constructor privado. Esto también reducirá la deuda técnica en 5 minutos.

## 5. Tratar code smells de tipo minor

La mayoría de code smells se encuentran en este apartado, en general se relacionan con unused imports, con mal uso de estructuras if else que contienen demasiados returns y con el seguimiento del convenio para nombrar variables y para determinar su visibilidad. Se han tratado la mayoría para hacer el código más legible y mantenible. El único code smell minor que no se ha tratado ha sido este:

```
class CallRunnable<T> implements Runnable {  
    private final Call<T> call;  
    public T response = null;  
}
```

 Make response a static final constant or non-public and provide accessors if needed. [Why is this an issue?](#) last month ▾ L19  

 Code Smell  Minor  Open Not assigned 10min effort No tags

Puesto que solo se llama internamente y considero que no tiene por qué ser algo problemático, además de que no ha sido implementado por nosotros. Solventar estos code smells reducirá a la mitad la deuda técnica.

## 6. Tratar los code smells de info

La mayoría son líneas de código que fueron usadas para depurar y en vez de ser borradas fueron comentadas así que no hubo problemas a la hora de resolverlos. Algunos de los que se han quedado

# Informe de Calidad II (Sprint 2)

**Autores:** Marcos Fernández Alonso

son TODO que aún hay que implementar, los code smells sobre funciones que no se usan que no vamos a borrar porque en un futuro podrían usarse

**Comentarios:** Puesto que no hay bugs que solucionar los cuales podrían llevar más tiempo me he decidido centrar en solucionar la mayoría de code smells y así reducir la deuda técnica lo máximo posible. También cabe recalcar que, a pesar de haber llevado a cabo el anterior plan de acción, algunos de los cambios se perdieron en algún merge que fue mal solucionado, principalmente volvieron a aparecer imports que no se usaban. Finalmente, la deuda técnica ha quedado en 50 min.