

Análisis del Código Fuente y Métricas Asociadas Sprint 2

G6-63



Participantes:

Luis Cerrato Sánchez (Desarrollador)

Carmen Galván López (Desarrollador)

Ezequiel González Macho (Desarrollador)

Antonio Quijano Herrera (Desarrollador)

Juan Salado Jurado (Desarrollador)

Álvaro Úbeda Ruiz (Scrum máster y Desarrollador)

Tabla de cambios	3
Introducción	4
Métricas Sprint 2:	4
Métricas Sprint 3:	5
Métricas al finalizar el Sprint 3:	5
Descripción y análisis de potenciales bugs	6
Sprint 2	6
Sprint 3	7
Descripción y análisis de los tipos de “code smells”	8
Sprint 2	8
Sprint 3	13
Diferencias entre los resultados del Sprint 2 y el Sprint 3	16
Conclusión	16

1. Tabla de cambios

FECHA	Descripción
31/03/2022	Creación del documento.
01/04/2022	Añadido “Descripción y análisis de potenciales bugs”
02/04/2022	Añadido “Descripción y análisis de los tipos de code smells”
16/04/2022	Extendida sección “Descripción y análisis de potenciales bugs” con el análisis del Sprint 3 y añadido “Conclusión”
17/04/2022	Añadido “Métricas al finalizar el Sprint 3”

2. Introducción

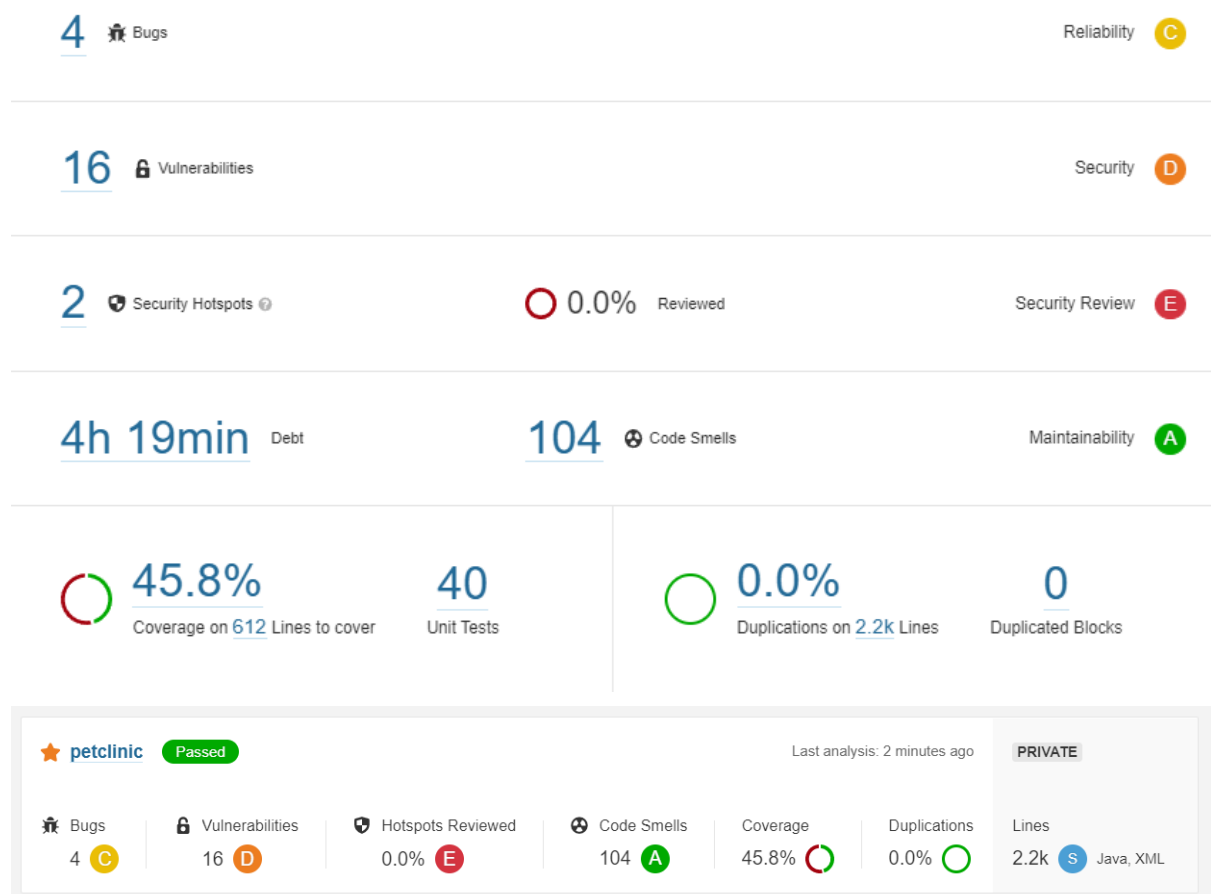
En el presente documento se pretende hacer un análisis del código de los sprints 2 y 3. En él se describirán en profundidad los “bugs” y “bad smells” proporcionados por la plataforma SonarQube.

Podríamos agrupar las issues que aparecen en tres grupos:

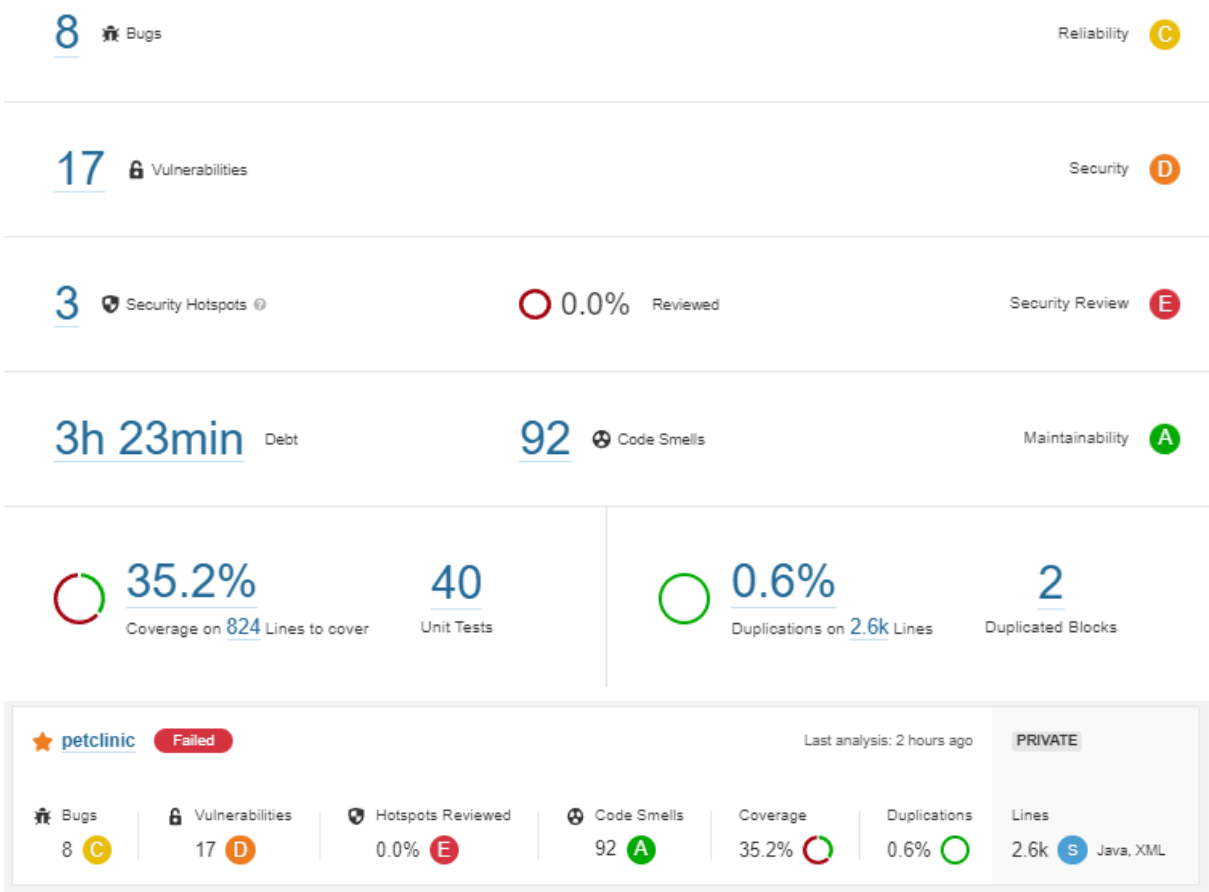
- **Code smells:** características del código que, aunque no impiden el correcto funcionamiento del programa, pueden indicar problemas potenciales y más profundos que afectan a la capacidad del mantenimiento del código. Su identificación puede ayudar a menguar la deuda técnica.
- **Bugs:** estos errores de código pueden impedir que el programa funcione según lo previsto. Afectan la fiabilidad del código
- **Vulnerabilidades:** son problemas en el código que podrían ser explotados por un mal actor para comprometer la seguridad de la aplicación.

A continuación se adjuntan las métricas proporcionadas por la plataforma:

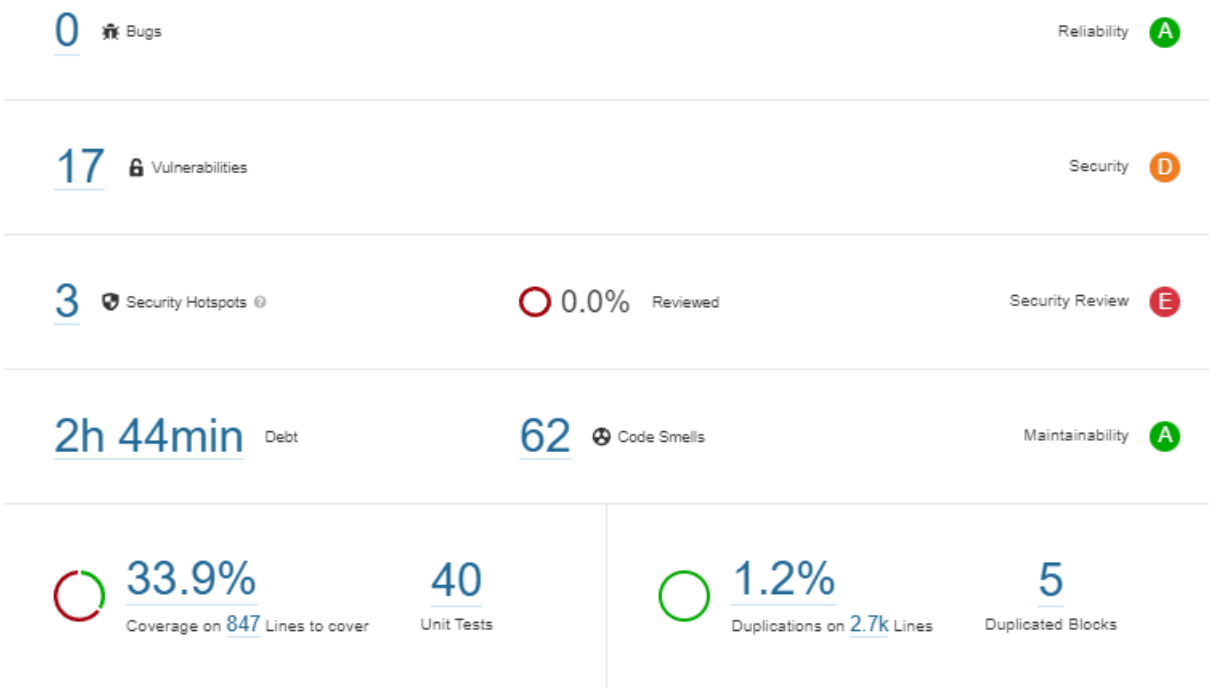
Métricas Sprint 2:

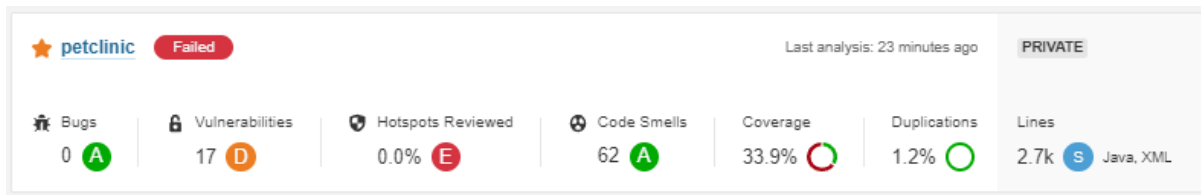


Métricas Sprint 3:



Métricas al finalizar el Sprint 3:





3. Descripción y análisis de potenciales bugs

Sprint 2

Bugs encontrados:

- **Comprobar si un tipo Optional está presente antes de usar su valor:**

La descripción proporcionada es “Llamar a “cause.isPresent()” antes de acceder al valor”. Esto podría ocasionar una excepción “NullPointerException” si se llega a intentar acceder al valor y éste resulta ser nulo.

La solución sería comprobar previamente si la variable “cause” está presente y en caso contrario tratar dicha excepción.

Ocurrencias:

```
List<Donation> donations = cause.get().getDonations().stream().collect(Collectors.toList());
```

Call "cause.isPresent()" before accessing the value. Why is this an issue? 7 days ago ▾ L60 🔗

🐞 Bug ▾ 🚨 Major ▾ 🔓 Open ▾ Not assigned ▾ 10min effort Comment 🔗 cwe ▾

(src/.../samples/petclinic/cause/CauseController.java)

```
Vet vet = this.vetService.findVetById(vetId).get();
```

Call "Optional#isPresent()" before accessing the value. Why is this an issue? 25 days ago ▾ L114 🔗

🐞 Bug ▾ 🚨 Major ▾ 🔓 Open ▾ Not assigned ▾ 10min effort Comment 🔗 cwe ▾

(src/.../samples/petclinic/vet/VetController.java)

- **Usar el método “equals” si se pretende comparar el valor**

La descripción proporcionada es “Usar el método “equals” si se pretende comparar su valor”. Al realizar una comparación con el operador “==” o “!=” se comprueba si los dos objetos apuntan exactamente a la misma dirección de la memoria, en cambio el método equals únicamente comprueba si sus valores coinciden.

La solución es simple, sustituir el operador “==” por el método “equals”.

En este caso el uso de “!=” es intencionado por lo tanto se marcará de este modo en la plataforma.

Ocurrencias:

```
if (compName.equals(name) && pet.getId()!=id) {
```

Use the "equals" method if value comparison was intended. Why is this an issue?

last month ▾ L148 🔗

🐞 Bug ▾ ⬆ Major ▾ 🔵 Open ▾ Not assigned ▾ 5min effort Comment

👤 cert, cwe ▾

(src/.../samples/petclinic/owner/Owner.java)

```
if (StringUtils.hasLength(pet.getName()) && (otherPet!= null && otherPet.getId()!=pet.getId())) {
```

Use the "equals" method if value comparison was intended. Why is this an issue?

last month ▾ L72 🔗

🐞 Bug ▾ ⬆ Major ▾ 🔵 Open ▾ Not assigned ▾ 5min effort Comment

👤 cert, cwe ▾

(src/.../samples/petclinic/pet/PetService.java)

Sprint 3

Trás finalizar las implementaciones del sprint 3 se procedió a arreglar los bugs encontrados en el código. Finalmente se realizó un nuevo análisis perteneciente al sprint 3.

Bugs encontrados:

- **Comprobar si un tipo Optional está presente antes de usar su valor:**

La descripción proporcionada es "Llamar a "cause.isPresent()" antes de acceder al valor". Esto podría ocasionar una excepción "NullPointerException" si se llega a intentar acceder al valor y éste resulta ser nulo.

La solución sería comprobar previamente si la variable "cause" está presente y en caso contrario tratar dicha excepción.

Ocurrencias:

sro/.../samples/petclinic/adoption/AdoptionRequestController.java	
<input type="checkbox"/> Call "Optional#isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	15 days ago L47 cwe
<input type="checkbox"/> Call "Optional#isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	15 days ago L67 cwe
<input type="checkbox"/> Call "Optional#isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	14 days ago L79 cwe
<input type="checkbox"/> Call "Optional#isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	15 days ago L118 cwe
<input type="checkbox"/> Call "Optional#isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	16 days ago L145 cwe
sro/.../samples/petclinic/adoption/AdoptionRequestService.java	
<input type="checkbox"/> Call "adoption.isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	19 days ago L61 cwe
<input type="checkbox"/> Call "Optional#isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	23 days ago L62 cwe
<input type="checkbox"/> Call "Optional#isPresent()" before accessing the value. Why is this an issue? Bug Major Open Not assigned 10min effort Comment	21 days ago L103 cwe

4. Descripción y análisis de los tipos de “code smells”

Sprint 2

Según la sección de SonarQube, issues o measure, podemos ver los code smells de nuestro proyecto.

Primeramente podemos observar el filtrado que podemos aplicarle a esos code smells del proyecto, clasificándolos en blocker, minor, critical, info y major, estos son los grados de importancia que toman.

Severity

Blocker

0

Minor

50

Critical

5

Info

40

Major

9

- Info

Existen 40 code smells con grado de severidad info. Representan información sobre algo cuya ausencia no afectase al correcto funcionamiento de la aplicación, solo que esté más limpio y fuera más eficiente o fácil de entender.

Todos son debido a los mismos dos fallos.

Nombre	<div> <input type="checkbox"/> Declare this local variable with "var" instead. Why is this an issue? </div> <div> Code Smell Info Open Not assigned 0min effort Comment </div>
Causas	Se produce porque hay varias variables que se declaran con el mismo nombre en distintas funciones. Lo detecta como code smell ya que esta situación puede llevar a confusión al no usar bien los nombres descriptivos.
Severidad del code smell	Dispensable
Solución	Se deberían utilizar distintos nombres para variables que no representan lo mismo, haría más rápido el entendimiento del código y sería más limpio.


Nombre	<div> <input type="checkbox"/> Remove this 'public' modifier. Why is this an issue? </div> <div> Code Smell Info Open Not assigned 2min effort Comment </div>
Causas	Informa que "public" modifier en las actuales versiones no es necesario de utilizar, ya que sin ello podría funcionar sin problemas.
Severidad del code smell	Dispensable
Solución	Para solucionarlo podríamos borrar la visibilidad " <i>public</i> " del método mencionado en el code smell, ya que no es necesario.

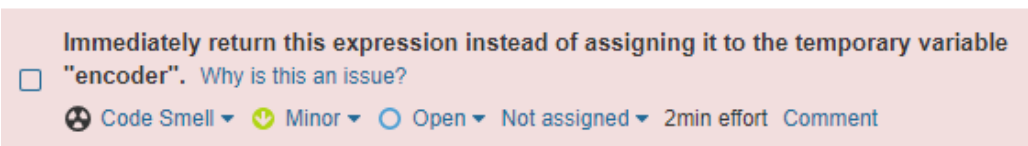
- Minor

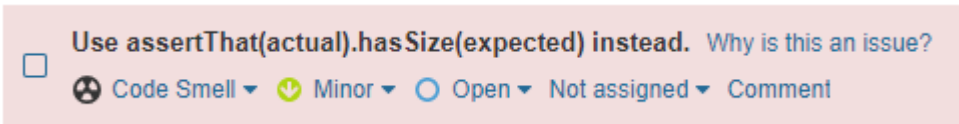
Existen 50 code smells con grado de severidad menor. Son code smells que no afectan al correcto funcionamiento del código, aunque son importantes para la buena legibilidad del mismo.

Muchos comparten el mismo error.

Nombre	<div> <input type="checkbox"/> Replace this lambda with a method reference. Why is this an issue? </div> <div> Code Smell Minor Open Not assigned 2min effort Comment </div>
Causas	Este code smell se da cuando utilizamos una solución que no explota las posibilidades del diseño orientado a objetos.
Severidad del code smell	Object-Orientation Abusers
Solución	Crear un método que realice esa operación.




Nombre	
Causas	Representan información sobre algo cuya ausencia no afectase al correcto funcionamiento de la aplicación, solo que esté más limpio y fuera más eficiente o fácil de entender.
Severidad del code smell	Dispensable (sobrante)
Solución	Eliminar ese import.




Nombre	
Causas	Ocurre al declarar una variable que se devolverá sin tener que realizar ningún tipo de filtro ni nada parecido, lo que hace que tengamos una línea más de código.
Severidad del code smell	Dispensable (sobrante)
Solución	En lugar de declarar una variable, retornarlo directamente.




Nombre	
Causas	Se produce porque el código escrito en los test, se podría simplificar más. Se podría conseguir un código más entendible y más limpio
Severidad del code smell	Dispensable (sobrante)
Solución	Sustituir el código por la alternativa indicada.




- Major




Contamos con un total de 9 code smells de severidad mayor. Estos errores suponen una alta gravedad ya que generan bastante deuda técnica.




Nombre	<input type="checkbox"/> Replace this use of System.out or System.err by a logger. Why is this an issue?  Code Smell ▾  Major ▾  Open ▾ Not assigned ▾ 10min effort Comment
Causas	Se puede simplificar el system.out sustituyéndolo por un logger, lo que haría que el código tuviese menos repeticiones y fuese más simple.
Severidad del code smell	Dispensable (sobrante) - Código duplicado
Solución	Sustituir por un logger, evitaría que tuviésemos que escribir un System.out cada vez que queramos mostrar un mensaje.

Nombre	<input type="checkbox"/> This block of commented-out lines of code should be removed. Why is this an issue?  Code Smell ▾  Major ▾  Open ▾ Not assigned ▾ 5min effort Comment
Causas	Los comentarios son útiles cuando explican por qué algo está implementado de una manera u otra, o explican algo complejo. Sin embargo, no sirve explicar lo que hace cada línea de código. Se prefiere cambiar el código para hacerlo más legible.
Severidad del code smell	Dispensable(sobrante) - Abundancia de comentarios
Solución	Eliminar el exceso de comentarios.

Nombre	<input type="checkbox"/> Remove this unused "userService" private field. Why is this an issue?  Code Smell ▾  Major ▾  Open ▾ Not assigned ▾ 5min effort Comment
Causas	Representa información sobre algo cuya ausencia no afectase al correcto funcionamiento de la aplicación, solo que esté más limpio y fuera más eficiente o fácil de entender. En este caso una variable que no se utiliza posteriormente.
Severidad del code smell	Dispensable (sobrante)
Solución	Eliminar la variable.

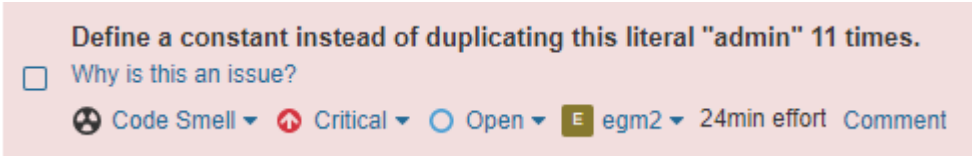
Nombre	<input type="checkbox"/> Add a private constructor to hide the implicit public one. Why is this an issue?  Code Smell ▾  Major ▾  Open ▾ Not assigned ▾ 5min effort Comment
Causas	El constructor debería ser privado.
Severidad del code smell	
Solución	Se debe definir al menos un constructor no público.

Nombre	<input type="checkbox"/> Rename field "vets" Why is this an issue?  Code Smell ▾  Major ▾  Open ▾ Not assigned ▾ 10min effort Comment
Causas	Java detecta que es confuso declarar un atributo de una clase cuyo nombre se parezca mucho a la clase en la que estamos trabajando. Las mejores prácticas dictan que cualquier campo o miembro con el mismo nombre que la clase adjunta debe cambiarse para que sea más descriptivo del aspecto particular de la clase que representa o tiene.
Severidad del code smell	Change preventers
Solución	Renombrar el atributo con otro nombre.

Nombre	<input type="checkbox"/> Define and throw a dedicated exception instead of using a generic one. Why is this an issue?  Code Smell ▾  Major ▾  Open ▾ Not assigned ▾ 20min effort Comment
Causas	El uso de excepciones genéricas evita que los métodos manejen excepciones verdaderas generadas por el sistema de manera diferente a los errores generados por la aplicación.
Severidad del code smell	
Solución	Definir un nuevo tipo de excepción

- Critical

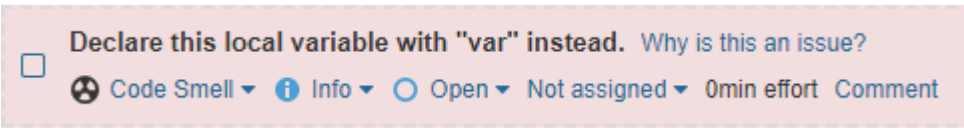
Existen 5 code smells con grado de severidad critical, donde todos ellos son del mismo tipo (duplicaciones). Estos errores suponen la máxima gravedad posible.

Nombre	
Causas	<p>Las cadenas de String duplicadas hacen que el proceso de refactorización sea propenso a errores, ya que debe asegurarse de actualizar todas las apariciones.</p> <p>Por otro lado, las constantes pueden referenciarse desde muchos lugares, pero solo necesitan actualizarse en un solo lugar.</p>
Severidad del code smell	Dispensable (sobrante)
Solución	Definir una constante.


Sprint 3

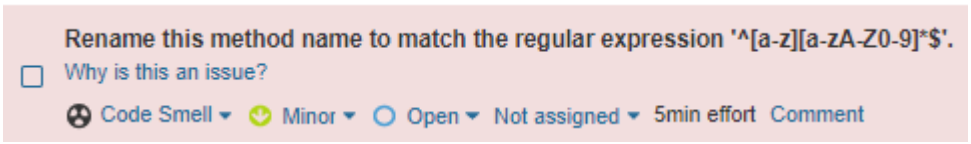
En el S3 nos encontramos con los siguientes code smells:

- Info

Nombre	
Causas	<p>Se produce porque hay varias variables que se declaran con el mismo nombre en distintas funciones. Lo detecta como code smell ya que esta situación puede llevar a confusión al no usar bien los nombres descriptivos.</p>
Severidad del code smell	Dispensable
Solución	<p>Se deberían utilizar distintos nombres para variables que no representan lo mismo, haría más rápido el entendimiento del código y sería más limpio.</p>

- Minor

Nombre	
Causas	Java 7 introdujo el operador de diamante (<>) para reducir la verbosidad del código genérico. Por ejemplo, en lugar de tener que declarar el tipo de una Lista tanto en su declaración como en su constructor, ahora puede simplificar la declaración del constructor con <>, y el compilador inferirá el tipo.
Severidad del code smell	Dispensable
Solución	Se podría reducir el código como por ejemplo en el siguiente caso "List<String> string = new ArrayList<String>();", se puede sustituir por "List<String> string = new ArrayList<>();", dejando solo <>.

Nombre	
Causas	Las convenciones de nomenclatura compartidas permiten que los equipos colaboren de manera eficiente. Esta regla comprueba que todos los nombres de métodos coincidan con una expresión regular proporcionada. El nombre de las funciones empiezan por mayúsculas y no sigue la regla.
Severidad del code smell	Dispensable
Solución	Cambiar los nombres para que sigan la expresión regular.

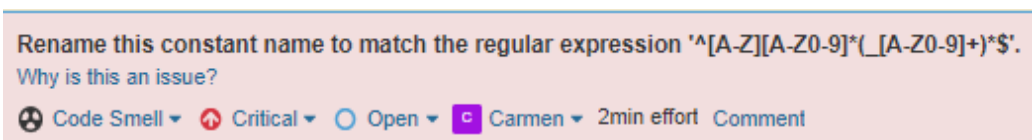
- Major

Nombre	<div> <input type="checkbox"/> 2 duplicated blocks of code must be removed. Why is this an issue? <div> Code Smell Major Open Not assigned 30min effort Comment </div> </div>
Causas	Hay dos bloques de código idénticos y sonar los identifica como código duplicado.
Severidad del code smell	Dispensable (sobrante) - Código duplicado
Solución	Abstraer en un método el código a utilizar.

Nombre	<div> <input type="checkbox"/> String contains no format specifiers. Why is this an issue? <div> Code Smell Major Open Carmen 10min effort Comment </div> </div>
Causas	Mal uso del Logger.
Severidad del code smell	
Solución	Seguir las pautas para el correcto funcionamiento de los logs, usando apropiadamente el formato de los string.

- Critical

Nombre	<div> <input type="checkbox"/> Define a constant instead of duplicating this literal "redirect:/vets" 3 times. Why is this an issue? <div> Code Smell Critical Open egm2 8min effort Comment </div> </div>
Causas	<p>Las cadenas de String duplicadas hacen que el proceso de refactorización sea propenso a errores, ya que debe asegurarse de actualizar todas las apariciones.</p> <p>Por otro lado, las constantes pueden referenciarse desde muchos lugares, pero solo necesitan actualizarse en un solo lugar.</p>
Severidad del code smell	Dispensable (sobrante)
Solución	Definir una constante.

Nombre	
Causas	Las convenciones de codificación compartidas permiten que los equipos colaboren de manera eficiente. Esta regla comprueba que todos los nombres constantes coincidan con una expresión regular proporcionada. Una constante no sigue la expresión regular, ya que está definida en minúsculas, y debe ir en mayúsculas.
Severidad del code smell	Dispensable (sobrante)
Solución	Definir correctamente la constante.

5. Diferencias entre los resultados del Sprint 2 y el Sprint 3

En este apartado, nos centraremos en comparar los resultados del análisis del código fuente del Sprint 2 con el código al finalizar el Sprint 3, para así demostrar cómo puede ayudarnos SonarQube a tener un mejor código.

Como podemos ver en el primer apartado, el primer aspecto en el que se ha mejorado con respecto al anterior Sprint es el de los bugs, eliminándolos por completo al final del Sprint 3.

En el apartado de vulnerabilidades contamos con una más respecto al segundo Sprint, esto es debido al incremento en cantidad del código, habiendo incrementado en solo 1.

En cuanto a la deuda técnica, es uno de los apartados en los que más hemos mejorado, hemos conseguido rebajar la deuda de 4 hora y 19 minutos en el Sprint 2, a tan solo 2 hora y 44 minutos al final del Sprint 3.

En el Sprint 2 contábamos con 104 malos olores, mientras que en el tercer Sprint, hemos conseguido reducir la cifra hasta los 62.

Lo que nos ha permitido mejorar los resultados del análisis es el propio análisis de SonarQube, ya que gracias a él hemos podido identificar todos los bugs y malos olores, entre otros, de nuestro código, para posteriormente poder arreglarlos.

6. Conclusión

Tras realizar el análisis del código perteneciente al sprint 2 y la implementación de la funcionalidad del sprint 2, se procedió a solucionar los bugs y bad smells proporcionados por SonarCloud.

Podemos ver que en cuanto a los code smells, son gran cantidad pero en cambio son poco variados y bastante repetidos debido a los malos hábitos a la hora de programar, que ensucian el código. Aún así son de fácil solución y no graves. Por otra parte, los bugs también son de fácil solución.

Finalmente, al crear la nueva release 2.0.0 se realizó un nuevo análisis para el Sprint 3 y se procedió a solucionar los nuevos bugs y bad smells.