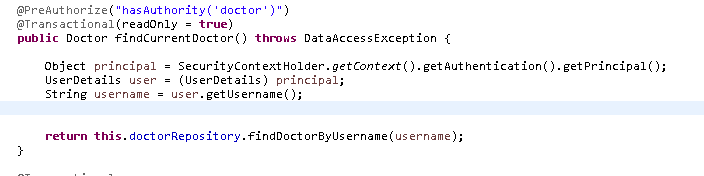
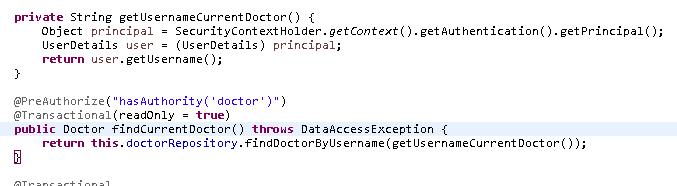
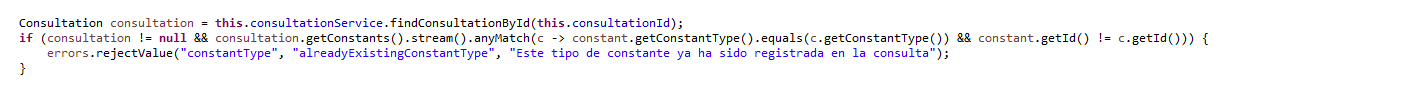
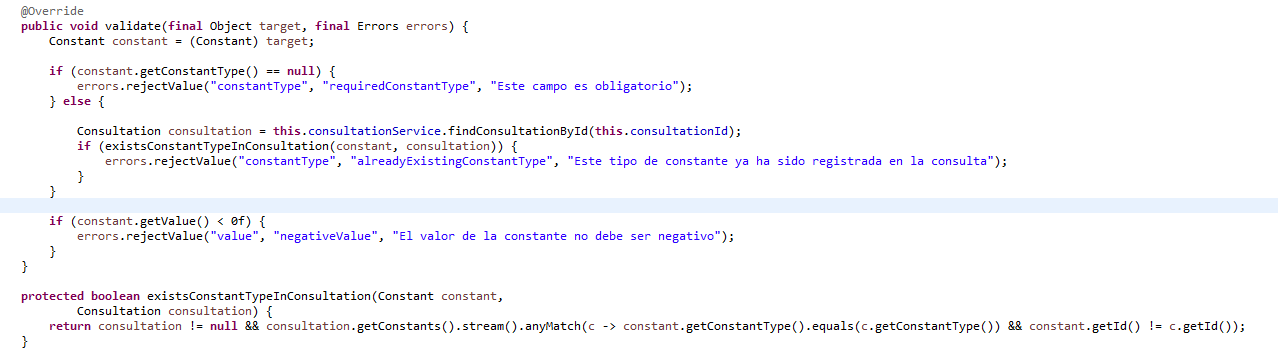
**Refactoring**

* Durante todo el desarrollo de la aplicación, por experiencias pasadas con otras asignaturas, siempre hemos tenido en cuenta la calidad de nuestro código, es decir, siempre hemos intentado seguir un standard a la hora de desarrollar tanto la aplicación como sus pruebas y siempre hemos intentado que sea claro y fácil de entender en numerosas revisiones. Por ello, hay pocas refactorizaciones que hayamos podido llevar a cabo.
* La mayoría de malos olores detectados tienen como solución la extracción de métodos compuestos y simplificación de condicionales:
* Extraemos ese fragmento de código relacionado con identificar a un usuario a un método aparte, quedando de la siguiente forma:
* También encontramos esta condición un poco compleja de entender, pudiendo simplificarla, de forma que quede más sencilla:
* Solución
* Como podemos observar, el predicado para comprobar si la contraseña es nula o está vacía (líneas 102 y 103) es excesivamente extenso. Para simplificarlo, hemos creado un método privado que realiza la misma función. Dicho método devuelve un booleano, por lo que se introducirá en la condición del bloque *if* directamente:
* Solución
* Encontramos la siguiente condición en el validador de Constant:

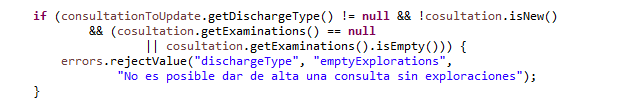


* Para solucionar la condición excesivamente larga, extraemos la condición a una función independiente que evalúe la condición:



Al realizar este cambio aumenta la facilidad para comprender el objetivo de la condición y su significado.

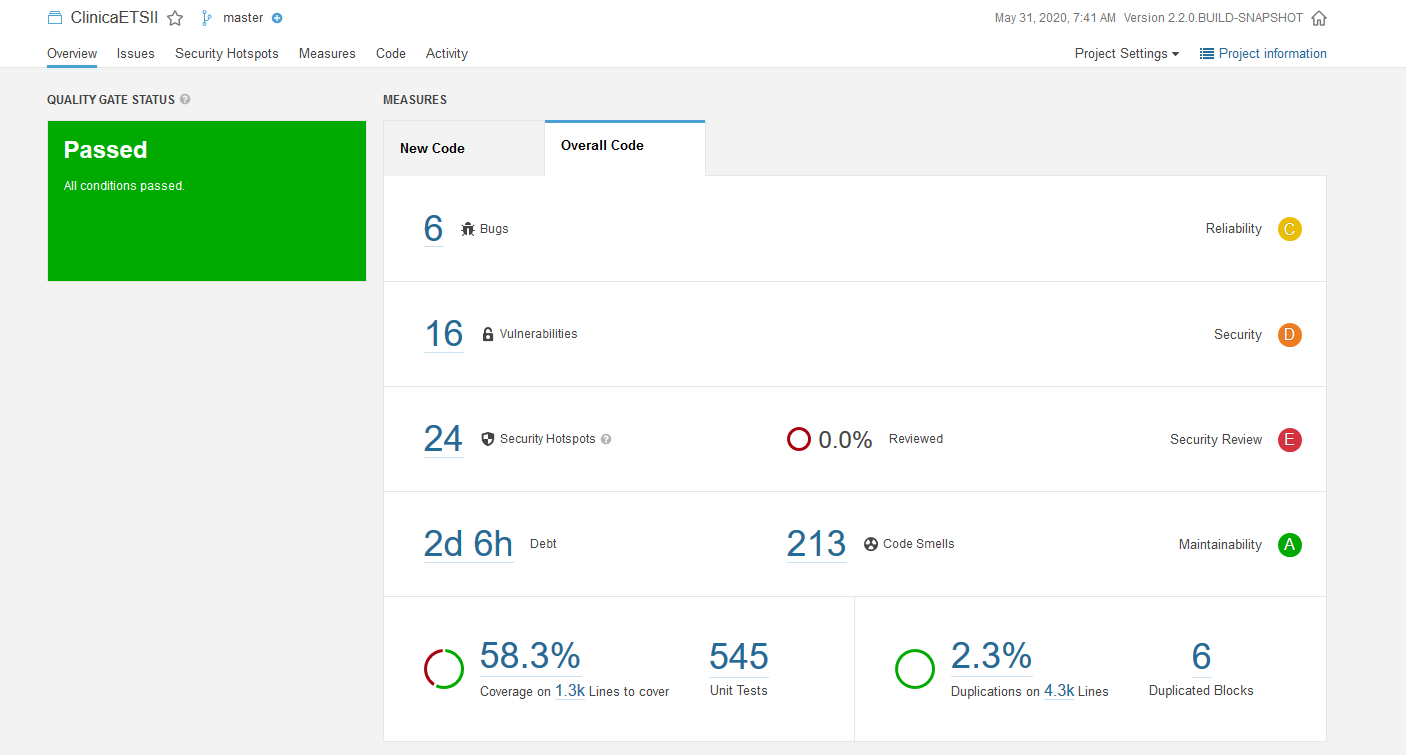
* Por otra parte, encontramos también otra condición excesivamente larga en el validador de Consultation:



* Para solucionar la condición excesivamente larga, extraemos la condición a una función independiente que evalúe la condición:



En este caso se ha utilizado un nombre más genérico al declarar la función puesto que en dicha condición se evalúa más de un atributo, aun así, el factor común es que esta condición solo se evalúa si ya existe la consulta con anterioridad y se trata de una actualización.

Para facilitar la búsqueda de Bad Smells se ha recurrido al uso de la herramienta SonarQube. 

A partir de este análisis se ha obtenido un resultado considerablemente bueno, debido a las siguientes razones:

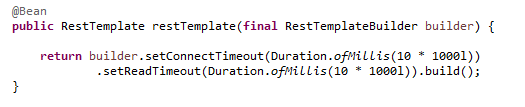
* Se han encontrado 6 bugs de los cuales, aquellos marcados con una cruz azul, pueden ser considerados falsos positivos debido a la implementación del código.



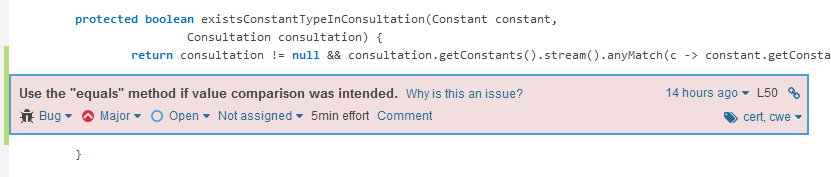
* A continuación, se procederá a corregir los dos bugs presentes en el código de la siguiente imagen, encontrado en SecurityConfiguration.



* Solución



Otro de los bugs es la recomendación del uso de equals en la siguiente condición de ConstantValidator:



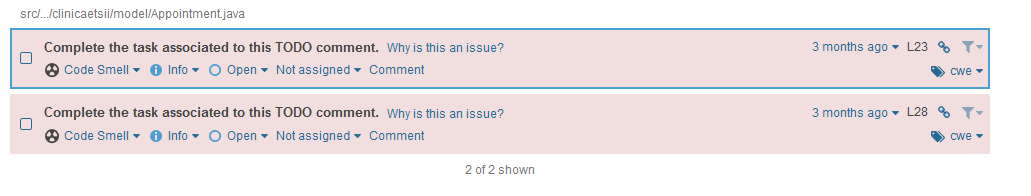


* Solución

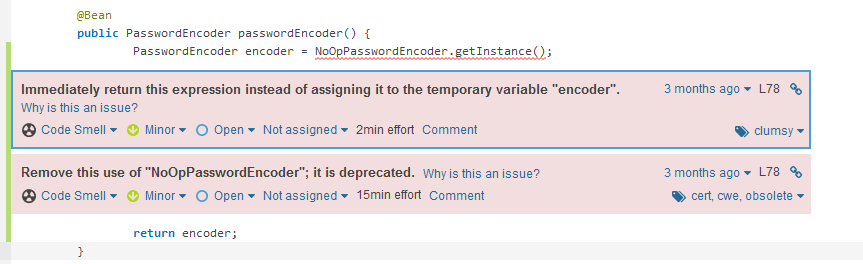


* Se han encontrado 16 vulnerabilidades, las cuales se deben principalmente al uso de objetos persistentes con la anotación @Entity como argumento de un método con @RequestMapping. De esto podemos deducir que el número de vulnerabilidades no se va a reducir, porque forma parte del patrón de programación utilizado a lo largo del proyecto.
* Se han encontrado 24 Hotspots, de los cuales 11 se deben al uso de anotaciones @PreAuthorize y @PostAuthorize y 12 al uso de patterns.
* Se han encontrado 213 Code Smells, en las cuales hay 5 Blockers, 19 Critical, 25 Mayors, 162 Minors y 2 Info.

Comenzaremos resolviendo los Info:



Se trata de dos comentarios TODO que describen 2 funcionalidades que habían sido planeadas para el código. Como estas ya han sido implementadas, se va a proceder borrando dichos comentarios.

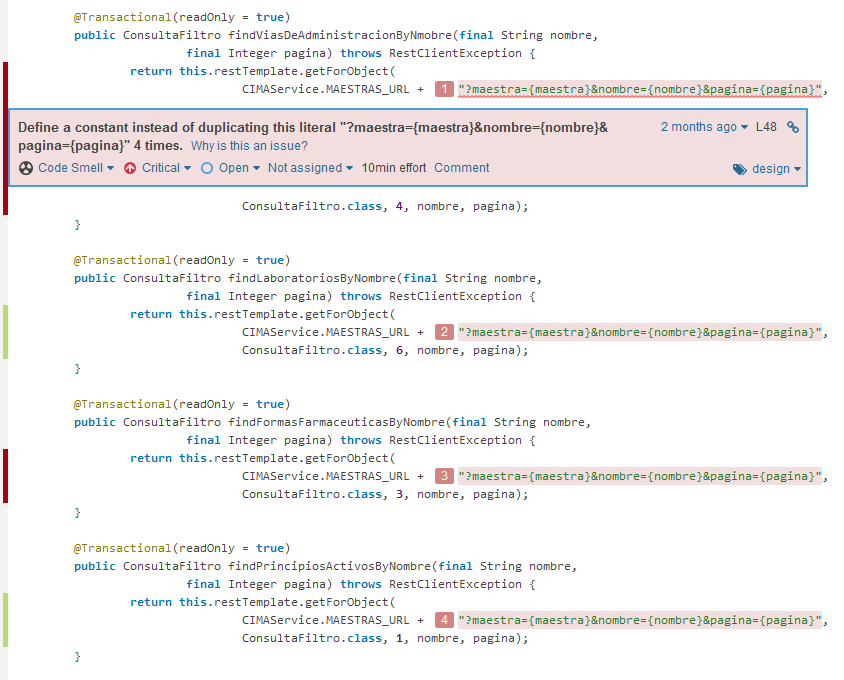
* Resolvemos uno de los Code Smells de severidad Minor:

El primero de los code smells señalados, situado en SecurityConfiguration, se resuelve devolviendo directamente el valor, en lugar de asignarlo a encoder, como se muestra en la siguiente imagen:

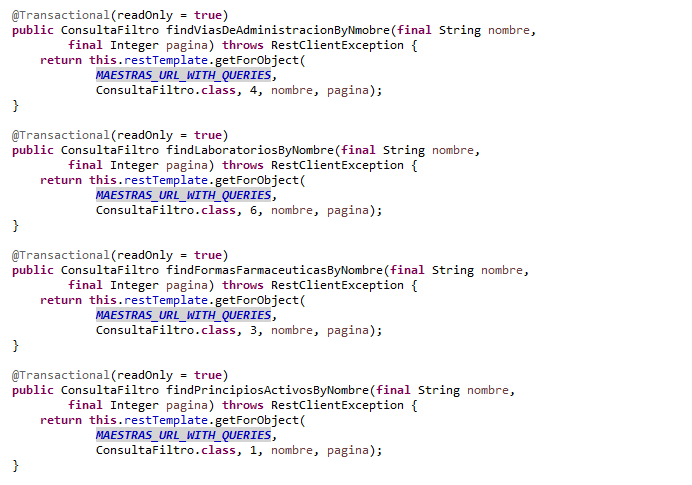
* Algunos de uno de los Code Smells de severidad Blocker son:



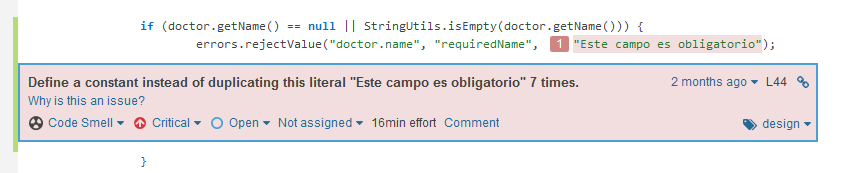
Los cuales se deben a que faltan algunos asserts o se encuentran incompletos. Su corrección es rápida, puesto que muchos de ellos se encuentras ausentes debido a despistes.

* Algunos de uno de los Code Smells de severidad Critical son:
* Solución



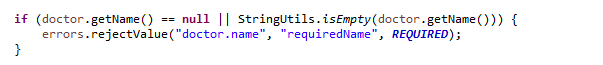


* En DoctorFormValidator, se ha encontrado:

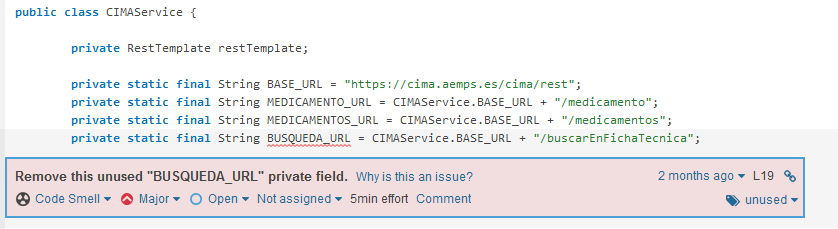


* Solución:



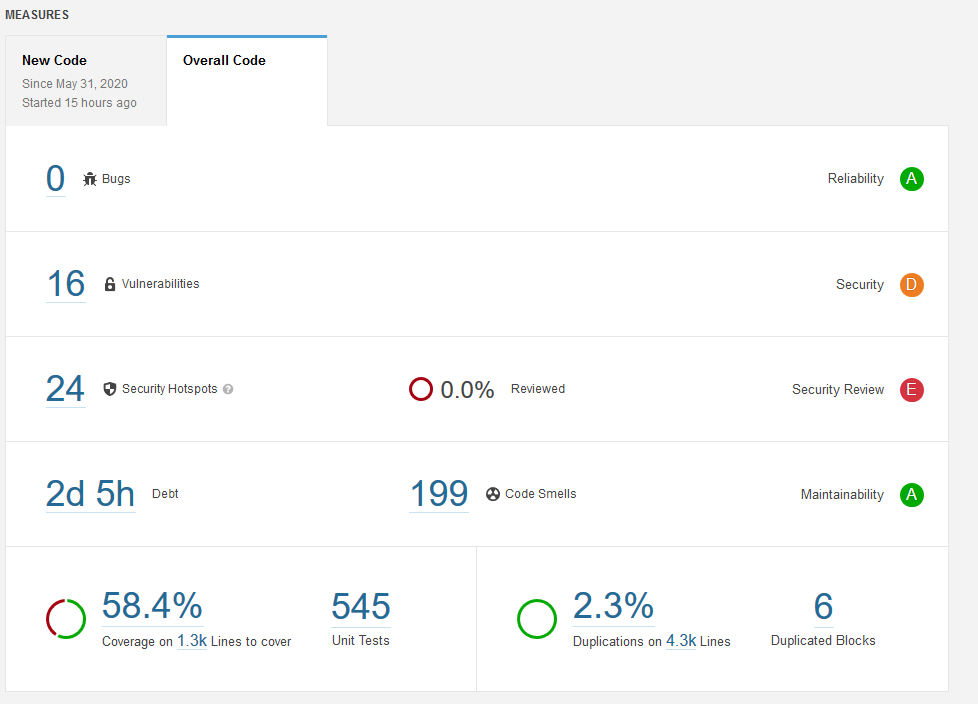


* Algunos de uno de los Code Smells de severidad Critical son:



Se soluciona borrando la constante no utilizada.

* Tras la resolución de algunos de los Code Smells encontrados, los resultados obtenidos del análisis son:



Tal como se esperaba, el número de vulnerabilidades se mantiene, pero se han reducido los bugs y el número de Code Smells a pasado de 213 a 199.