Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Laboratorio de Desarrollo y Herramientas.

Informe 3:

herramientas de calidad del producto software y documentación- SonarQube, Maven y Doxygen

Marcelo Daniel Choque Mamani

(alu0101074986@ull.edu.es)

2. ÍNDICE

3. INTRODUCCIÓN

Esta práctica tiene como objetivo analizar y documentar un proyecto software proporcionado en el aula virtual, usando SonarQube [1], Maven [2] y Doxygen [3]. SonarQube es una herramienta de código abierto para la revisión y evaluación de la calidad del código, con capacidades como análisis de código estático, detección de errores y de código duplicado, y evaluación de la complejidad. Además, soporta numerosos lenguajes de programación.

Por su parte, Maven facilita la construcción y configuración de proyectos Java, proporcionando una estructura uniforme. El archivo pom.xml [4] es clave en Maven, ya que contiene la configuración del proyecto, como dependencias y plugins.

Finalmente, Doxygen genera documentación a partir de archivos fuente en diversos lenguajes, incluyendo Java, y permite crear gráficos que representan las relaciones entre elementos de código. A continuación, configuraremos algunos plugins de Maven para realizar el análisis y detectar problemas de seguridad en el proyecto dado.

4. DESARROLLO

4.1 Plugin de Maven

Para esta práctica, he implementado los siguientes plugins compatibles con Maven:

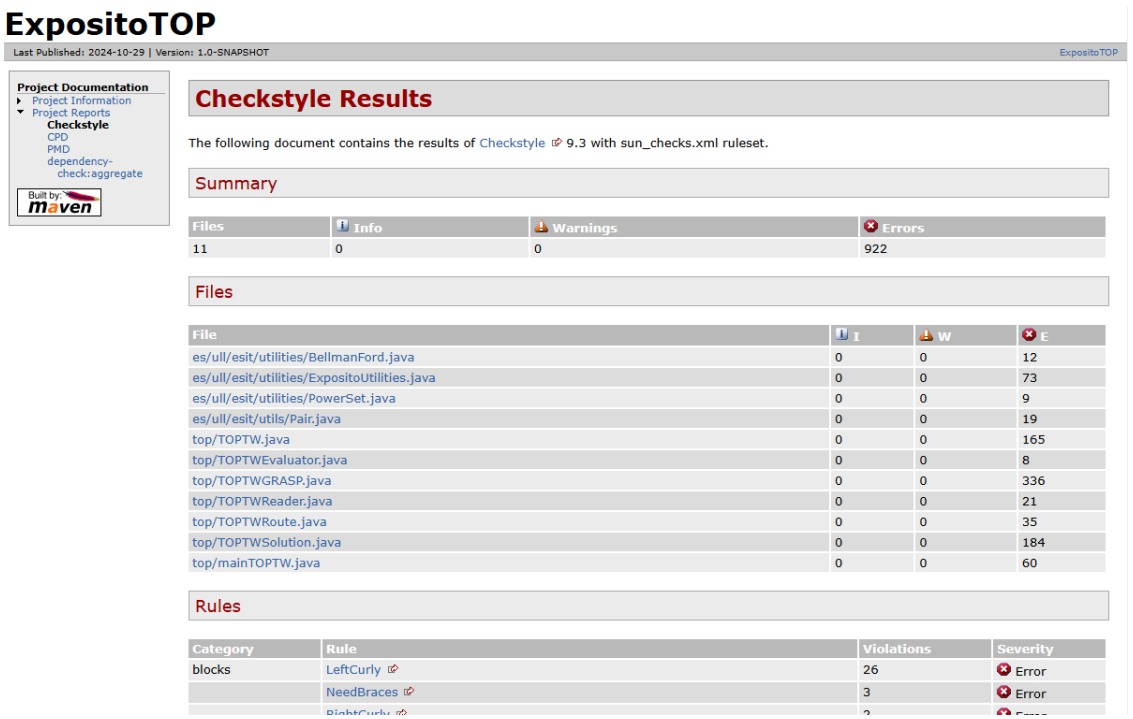
- Checkstyle [5]: es una herramienta que facilita la escritura de código Java bajo un estilo consistente y definido. Al integrarse con Maven, el plugin Checkstyle analiza el código en busca de advertencias y errores relacionados con el estilo elegido, generando un informe detallado que muestra las infracciones y recomendaciones para mejorar la legibilidad y consistencia del código.
- PMD [6]: se trata de una herramienta de análisis de código orientada a detectar defectos comunes, como variables declaradas pero no utilizadas o estructuras de código propensas a errores. Con Maven, el plugin PMD ejecuta esta herramienta automáticamente y produce un informe de resultados, lo cual permite identificar y corregir problemas antes de que se conviertan en fallos en producción.
- OWASP Dependency-Check [7]: es una herramienta de seguridad que examina las dependencias de un proyecto en busca de vulnerabilidades conocidas, proporcionando información crítica para mantener la seguridad del software. Con Maven, el plugin Dependency-Check verifica las dependencias y genera un informe detallado sobre cualquier vulnerabilidad identificada, lo cual permite tomar decisiones informadas sobre las actualizaciones necesarias para mejorar la seguridad del proyecto.

4.2. Análisis del proyecto

Para llevar a cabo el análisis del proyecto, me he ayudado de los plugins de Maven mencionados en el apartado anterior y de la herramienta SonarQube. Cabe destacar que, antes de poder analizar el código, he creado y configurado el archivo pom.xml, fundamental para poder utilizar Maven sobre el proyecto, y también he trasladado todas las subcarpetas del proyecto a un directorio src. A continuación se detalla el proceso realizado para analizar el proyecto con las diferentes herramientas mencionadas:

Checkstyle:

- En primer lugar, he ejecutado la orden “mvn checkstyle:checkstyle” en el directorio del proyecto.
- Después de ejecutar el plugin Checkstyle, se generó un informe en el subdirectorio target/site del proyecto. Como muestra la Figura 1, el análisis se realizó siguiendo las reglas definidas en el archivo sun_check.xml. Los resultados indican que el proyecto contiene 11 archivos con un total de 922 infracciones de estilo, lo que evidencia que el código no se ajusta a las normas establecidas.



ExpositoTOP

Last Published: 2024-10-29 | Version: 1.0-SNAPSHOT — ExpositoTOP

Project Documentation
- Project Information
- Project Reports
  - Checkstyle
  - CPD
  - PMD
  - dependency-check:aggregate

Built by: maven

Checkstyle Results

The following document contains the results of Checkstyle 9.3 with sun_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
11	0	0	922

Files

File	I	W	E
es/ull/esit/utilities/BellmanFord.java	0	0	12
es/ull/esit/utilities/ExpositoUtilities.java	0	0	73
es/ull/esit/utilities/PowerSet.java	0	0	9
es/ull/esit/utils/Pair.java	0	0	19
top/TOPTW.java	0	0	165
top/TOPTWEvaluator.java	0	0	8
top/TOPTWGRASP.java	0	0	336
top/TOPTWReader.java	0	0	21
top/TOPTWRoute.java	0	0	35
top/TOPTWSolution.java	0	0	184
top/mainTOPTW.java	0	0	60

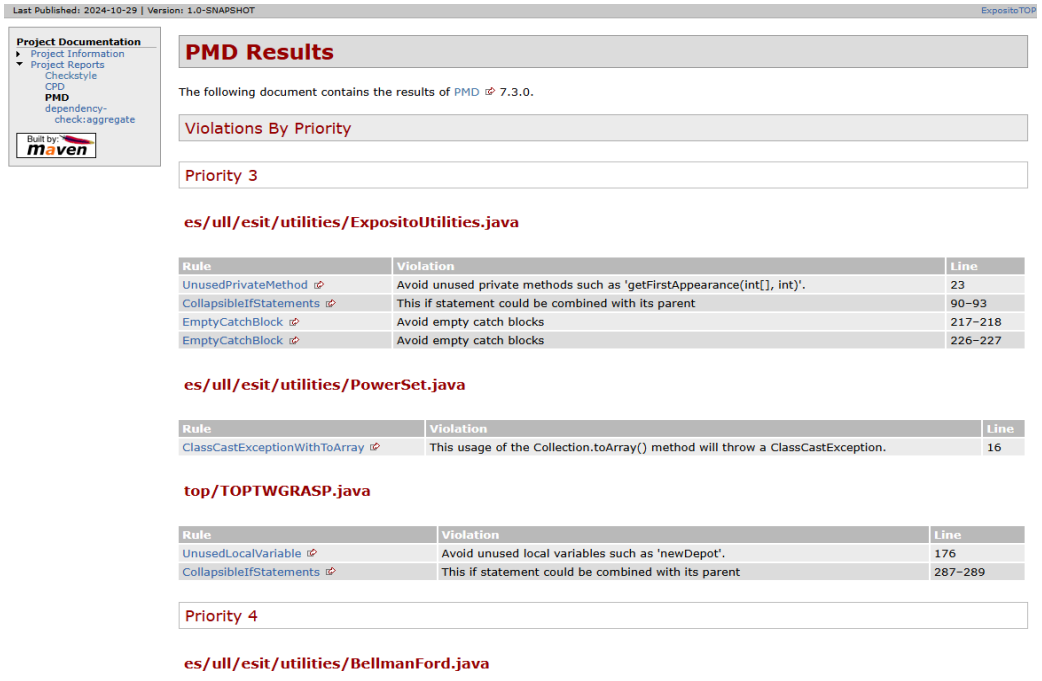
Rules

Category	Rule	Violations	Severity
blocks	LeftCurly	26	Error
	NeedBraces	3	Error
	RightCurly	2	Error

[Imagen 1: Resultados del checkstyle]

PMD:

- Después de analizar el código con Checkstyle, ejecuté el comando *mvn pmd:pmd* para lanzar el plugin PMD. Esto generó un informe en formato pmd.html, donde se detallan los defectos detectados en el código.
- En la imagen 2 se muestran los resultados de PMD. En ExpositoUtilities.java, se identificaron métodos privados sin uso, bloques if simplificables y bloques catch vacíos. En PowerSet.java, se señala un posible ClassCastException al usar toArray(). Este informe facilita la detección de malas prácticas y defectos para mejorar la calidad del código.



Last Published: 2024-10-29 | Version: 1.0-SNAPSHOT — ExpositoTOP

Project Documentation
- Project Information
- Project Reports
  - Checkstyle
  - CPD
  - PMD
  - dependency-check:aggregate

Built by: maven

PMD Results

The following document contains the results of PMD 7.3.0.

Violations By Priority

Priority 3

es/ull/esit/utilities/ExpositoUtilities.java

Rule	Violation	Line
UnusedPrivateMethod	Avoid unused private methods such as 'getFirstAppearance(int[], int)'.	23
CollapsibleIfStatements	This if statement could be combined with its parent	90–93
EmptyCatchBlock	Avoid empty catch blocks	217–218
EmptyCatchBlock	Avoid empty catch blocks	226–227

es/ull/esit/utilities/PowerSet.java

Rule	Violation	Line
ClassCastExceptionWithToArray	This usage of the Collection.toArray() method will throw a ClassCastException.	16

top/TOPTWGRASP.java

Rule	Violation	Line
UnusedLocalVariable	Avoid unused local variables such as 'newDepot'.	176
CollapsibleIfStatements	This if statement could be combined with its parent	287–289

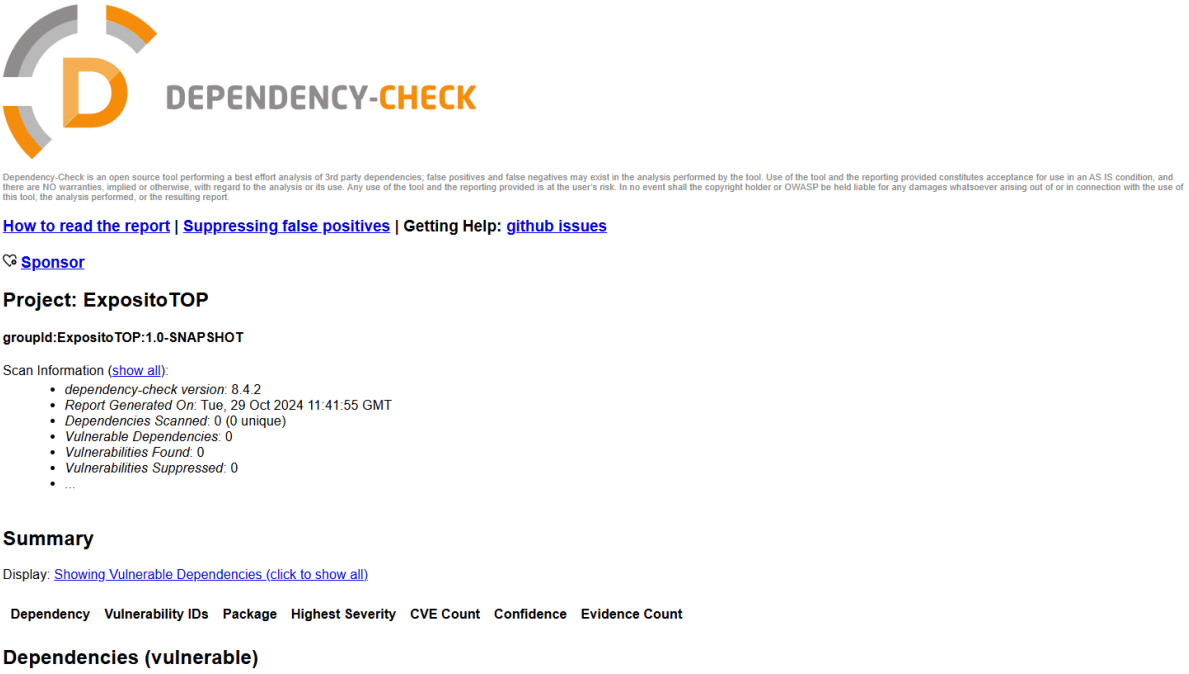
Priority 4

es/ull/esit/utilities/BellmanFord.java

[Imagen 2: Resultado generado por PMD]

OWASP Dependency-Check:

- La imagen muestra el informe generado por OWASP Dependency-Check. En el informe generado, se indica que no se detectaron dependencias vulnerables en el proyecto. En el resumen, se observa que el escaneo analizó cero dependencias y, por lo tanto, no encontró ninguna vulnerabilidad. Esto sugiere que, en términos de seguridad de dependencias, el proyecto está libre de riesgos conocidos.



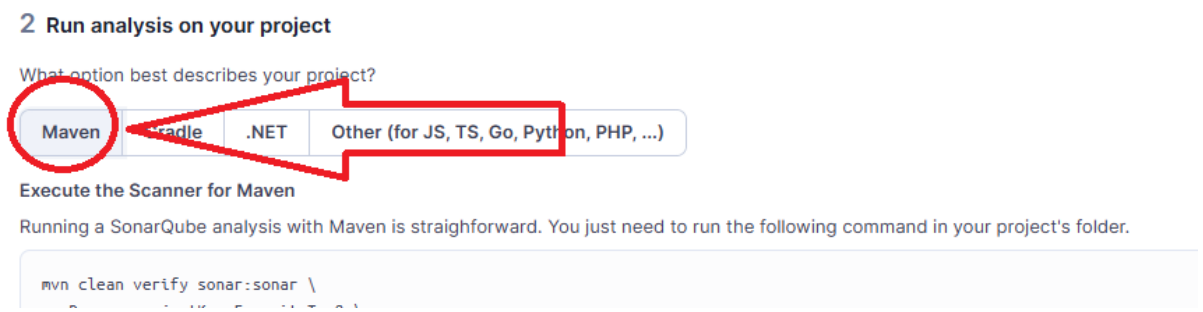
[Imagen 3: Resultado generado por OWASP Dependency-Check]

4.3 SonarQube

Iniciamos el análisis del proyecto reflejado en la imagen 4.

[Imagen 4: Ejecutando análisis con SonarQube]

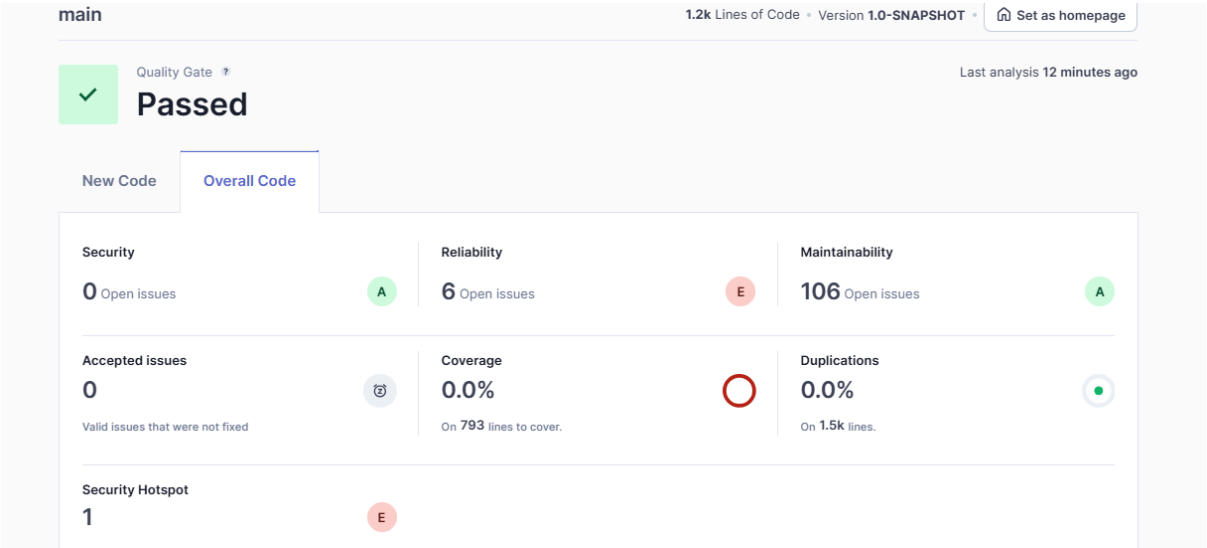
Después de analizar el proyecto con los plugins instalados, también utilicé SonarQube. Para ello, inicié la herramienta desde una terminal y accedí a localhost:9000 en mi navegador. Desde la interfaz de SonarQube, creé un nuevo proyecto llamado "ExpositoTOP" y lo configuré para ejecutar el análisis mediante Maven, como se muestra en la Figura 5.



[Imagen 5: Seleccionando maven para analizar proyecto]

Nos hemos de asegurar de establecer que el proyecto se construye con maven como se señala en la imagen 5 antes de realizar el análisis.

Finalizado el análisis realizado por SonarQube, hemos obtenido los siguientes resultados reflejados en la imagen 6:

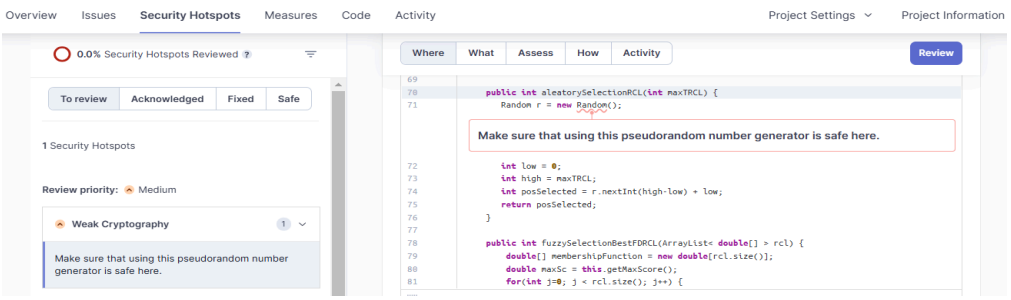


[Imagen 6: Resultado del análisis de SonarQube]

La Figura 6, muestra el resultado del análisis de SonarQube para el proyecto "ExpositoTOP", que en el apartado de seguridad no se encontraron problemas abiertos, mientras que en fiabilidad se identificaron 6 problemas y en mantenibilidad hay 106 problemas abiertos. Además, se detectó un "Security Hotspot" que requiere revisión. En general, el análisis refleja una buena seguridad, aunque existen áreas de mejora en fiabilidad y mantenibilidad, por lo tanto, ha pasado el "Quality Gate" y cumple con los estándares de calidad básicos

4.3.1 Security hotspots

En la imagen 7, se muestra un "Security Hotspot" detectado por SonarQube, que señala una posible vulnerabilidad en el uso de un generador de números pseudorrandom (Random). SonarQube recomienda revisar este código, ya que el uso de generadores de números predecibles puede representar un riesgo de seguridad en aplicaciones donde la imprevisibilidad es crucial. Si el software genera valores que pueden preverse, un atacante podría anticiparse a estos valores para explotar el sistema o acceder a datos sensibles.



[Imagen 7: Security hotspot]

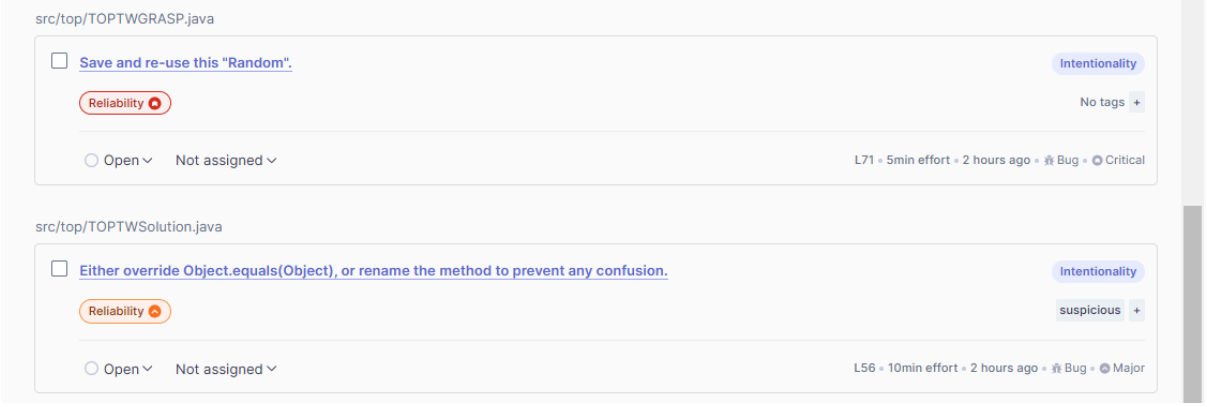
Específicamente, SonarQube desaconseja el uso de java.util.Random y java.lang.Math.random() en contextos de seguridad crítica. En su lugar, se recomienda utilizar java.security.SecureRandom, que genera números aleatorios criptográficamente seguros. En este caso, la línea de código problemática podría ser reemplazada por:

SecureRandom r = new SecureRandom();

4.3.2 Bugs

Este proyecto tiene un total de 6 bugs, algunos de ellos se muestran en la imagen 8, procederemos a arreglarlos.

El primer problema identificado se refiere a la línea de código resaltada en la Imagen 7. SonarQube detecta que generar un nuevo número aleatorio cada vez que se ejecuta el método public int *aleatorySelectionRCL(int)* es ineficiente y, dependiendo del JDK utilizado, podría producir resultados poco aleatorios. Por ello, sugiere declarar esta variable como una variable de clase, como se muestra en la Imagen 9.



[Imagen 8: Bug 1]

Compliant solution

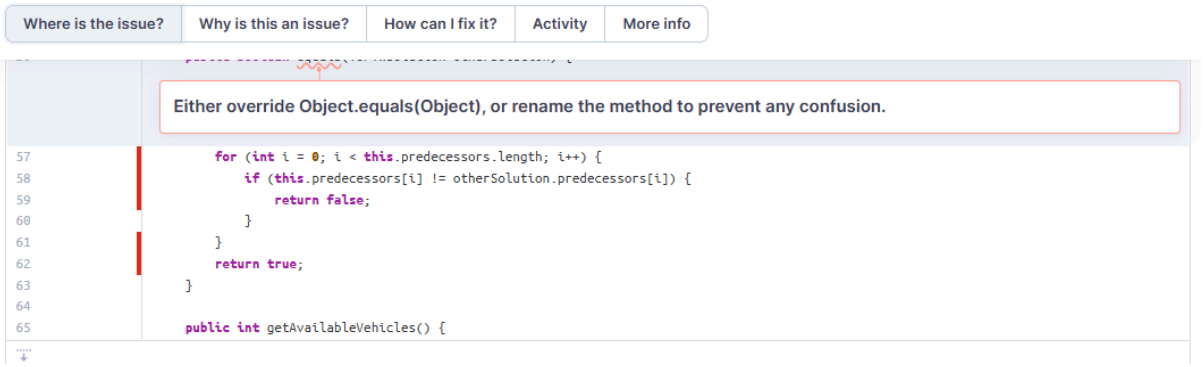
```
class MyClass {
    private Random random = new Random();  // Compliant

    public void doSomethingCommon() {
      int rValue = this.random.nextInt();
    }
}
```

[Imagen 9: Solución al bug 1]

El segundo problema identificado se encuentra en la línea 56 del archivo *TOPTWSolution.java* del proyecto. Como muestra la Imagen 10, SonarQube detectó un error relacionado con el uso del método equals().

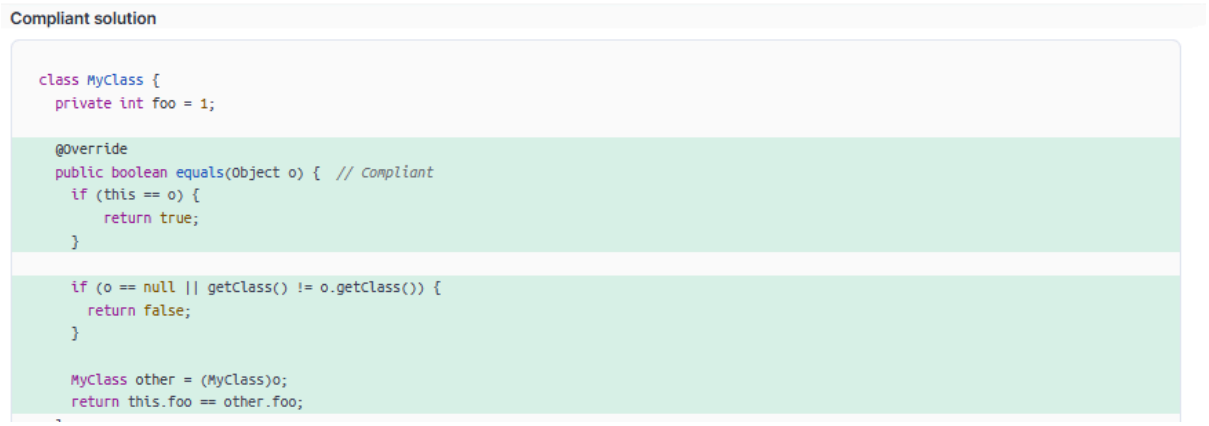
En Java, el método *Object.equals()* se utiliza para comparar objetos y suele sobrescribirse en las clases para definir un criterio de igualdad personalizado. La implementación predeterminada de *equals()* en Object compara las referencias de memoria, es decir, verifica si ambos objetos son la misma instancia. SonarQube recomienda que el nombre

equals se use exclusivamente para sobrescribir *Object.equals(Object)*, evitando así confusiones. Por esta razón, SonarQube detectó este error. Para resolverlo, he cambiado el nombre del método a *isEqual.*



[Imagen 10: Bug 2]

En la imagen 11, SonarQube nos da una sugerencia para solucionar el problema.



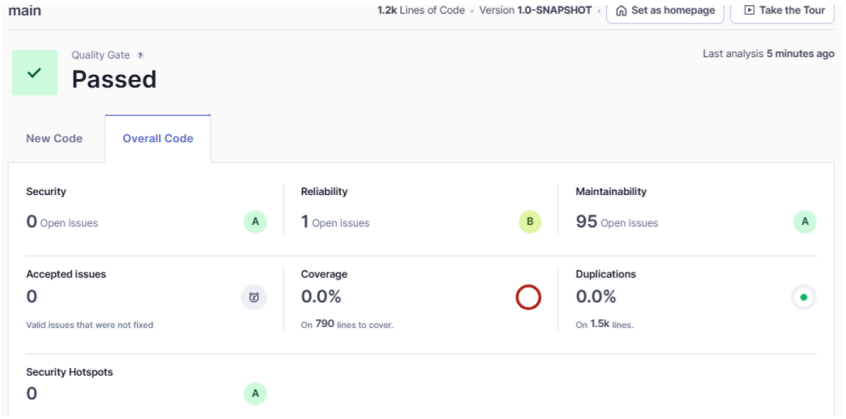
[Imagen 11: Solución al bug 2]

Corregimos otros errores Imagen 12 y 13.

[Imagen 12: bug 3]

[Imagen 13: bug 4]

Finalmente, SonarQube confirma que el proyecto ExpositoTop ha pasado el Quality Gate con éxito. El código se destaca por su alta seguridad, sin problemas abiertos, y una excelente fiabilidad con solo un detalle menor. Además, no presenta duplicaciones, lo cual contribuye a la eficiencia y claridad del proyecto. En general, el análisis muestra que el proyecto cumple con los estándares de calidad y está bien estructurado en términos de seguridad y organización del código, por otro lado ya no hay problemas de seguridad.

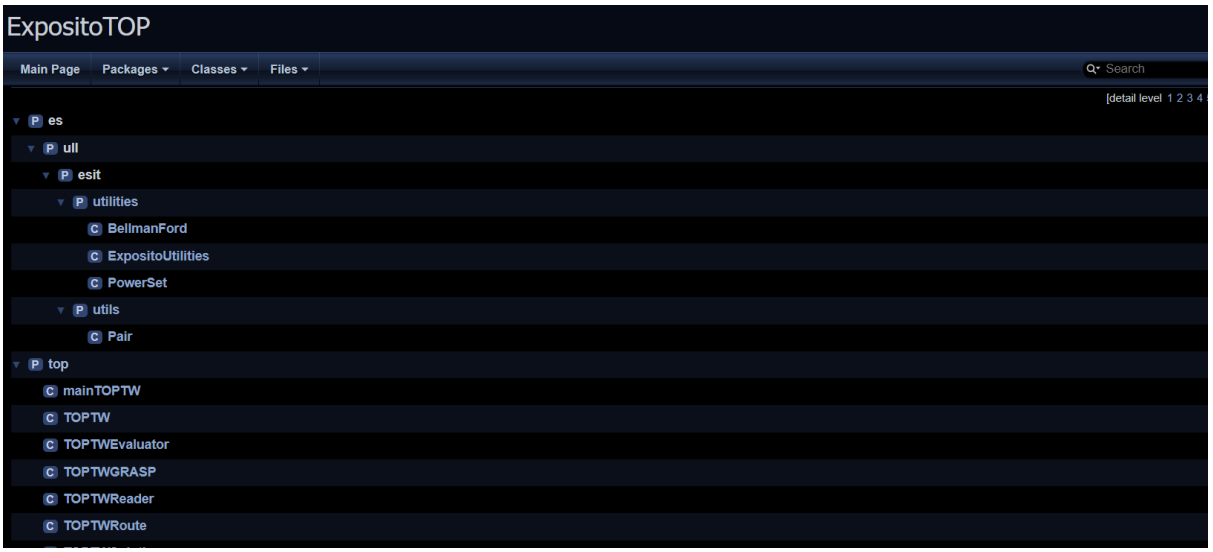


[Imagen 14: Resultado final del análisis de SonarQube]

5. Doxygen

Por último, configuramos el fichero Doxyfile y generamos la documentación con ayuda de doxygen ruta: html/index.html

```
C:\Users\danie\Documents\carpetaIntelligence\ExpositoTOP\ExpositoTOP>doxygen Doxyfile
Doxygen version used: 1.12.0 (c73f5d30f9e8b1df5ba15a1d064ff2067cbb8267)
Searching for include files...
Searching for example files...
Searching for images...
Searching for dot files...
Searching for msc files...
Searching for dia files...
Searching for files to exclude
Searching INPUT for files to process...
Searching for files in directory C:/Users/danie/Documents/carpetaIntelligence/ExpositoTOP/ExpositoTOP/src
```



6. Conclusión

En conclusión, el análisis de calidad del proyecto ExpositoTop demuestra un código bien estructurado, seguro y confiable, cumpliendo con los estándares del Quality Gate de SonarQube. La integración de Maven ha facilitado la gestión de dependencias y la automatización de tareas, asegurando una configuración consistente y eficiente. Además, el uso de Doxygen ha permitido generar documentación detallada, lo que contribuye a la claridad y mantenimiento del proyecto. La ausencia de vulnerabilidades y duplicaciones refleja el compromiso con las buenas prácticas de desarrollo, proporcionando una base sólida para el futuro del proyecto.

7. Bibliografía

[1] SonarQube

[2] Maven

[3] Doxygen

[4] Pom.xml

[5] Checkstyle

[6] PMD

[7] OWASP Dependency-Check