FRAG.JETZT BACKEND

VERSION 1.0

CODE ANALYSIS

By: Administrator

2023-01-20

CONTENT

INTRODUCTION

This document contains results of the code analysis of frag.jetzt Backend.

CONFIGURATION

- Quality Profiles
  - Names: Sonar way [Java]; Sonar way [XML];
  - Files: AYW221OPfGQw7yIQA4ko.json; AYW221R2fGQw7yIQA4v0.json;
- Quality Gate
  - Name: frag.jetzt
  - File: frag.jetzt.xml

SYNTHESIS

ANALYSIS STATUS

Reliability	Security	Security Review	Maintainability
D	A	E	A

QUALITY GATE STATUS

Quality Gate Status	Failed

Metric	Value
Reliability Rating	ERROR (D is worse than A)
Security Rating	OK
Maintainability Rating	OK
Coverage	ERROR (1.1% is less than 80%)
Duplicated Lines (%)	ERROR (8.1% is greater than 3%)

METRICS

Coverage	Duplication	Comment density	Median number of lines of code per file	Adherence to coding standard
1.1 %	8.1 %	0.4 %	36.5	97.0 %

TESTS

Total	Success Rate	Skipped	Errors	Failures

3	**100.0 %**	**0**	**0**	**0**

DETAILED TECHNICAL DEBT

Reliability	Security	Maintainability	Total
1d 2h 35min	-	4d 3h 48min	**5d 6h 23min**

METRICS RANGE

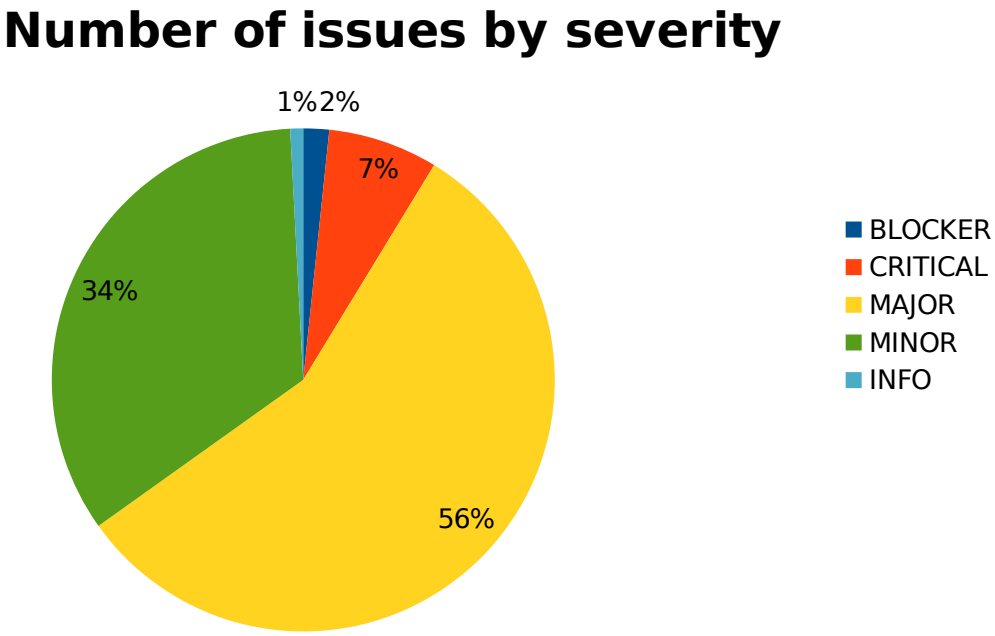
	Cyclomatic Complexity	Cognitive Complexity	Lines of code per file	Comment density (%)	Coverage	Duplication (%)
Min	0.0	0.0	3.0	0.0	0.0	0.0
Max	2695.0	688.0	11261.0	9.4	100.0	56.6

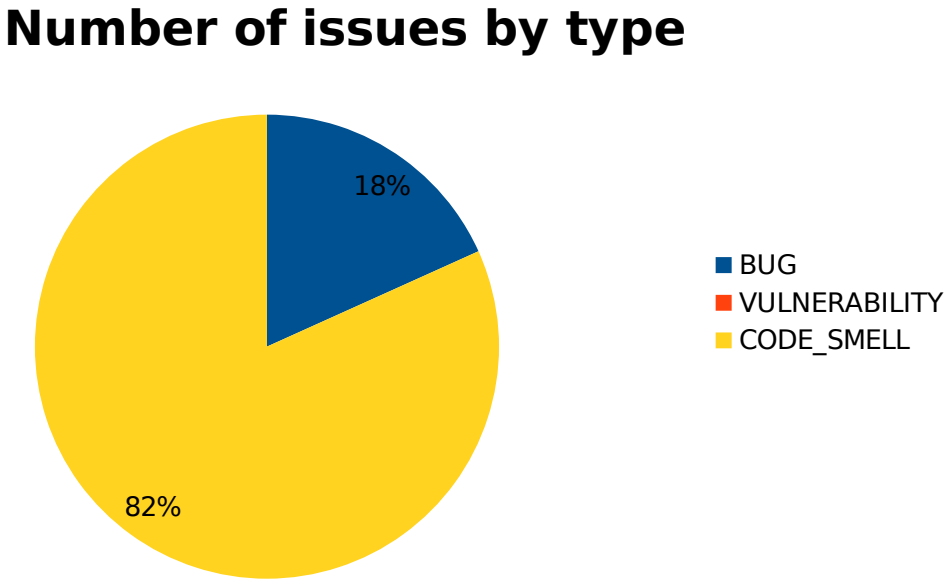
VOLUME

Language	Number
Java	11261
XML	169
Total	11430

ISSUES

CHARTS





ISSUES COUNT BY SEVERITY AND TYPE

Type / Severity	INFO	MINOR	MAJOR	CRITICAL	BLOCKER
BUG	0	41	2	1	0
VULNERABILITY	0	0	0	0	0
CODE_SMELL	2	41	134	16	4

ISSUES LIST

Name	Description	Type	Severity	Number
"Random" objects should be reused	Creating a new Random object each time a random value is needed is inefficient and may produce numbers which are not random depending on the JDK. For better efficiency and randomness, create a single Random, then store, and reuse it. The Random() constructor tries to set the seed with a distinct value every time. However there is no	BUG	CRITICAL	1

guarantee that the seed will be random or even uniformly distributed. Some JDK will use the current time as seed, which makes the generated numbers not random at all. This rule finds cases where a new Random is created each time a method is invoked and assigned to a local random variable. Noncompliant Code Example public void doSomethingCommon() { Random rand = new Random(); // Noncompliant; new instance created with each invocation int rValue = rand.nextInt(); //... Compliant Solution private Random rand = SecureRandom.getInstanceStrong(); // SecureRandom is preferred to Random public void doSomethingCommon() { int rValue = this.rand.nextInt(); //... Exceptions A class which uses a Random in its

	constructor or in a static main function and nowhere else will be ignored by this rule. See OWASP Top 10 2017 Category A6 - Security Misconfiguration			
Silly equality checks should not be made	Comparisons of dissimilar types will always return false. The comparison and all its dependent code can simply be removed. This includes: comparing an object with null comparing an object with an unrelated primitive (E.G. a string with an int) comparing unrelated classes comparing an unrelated class and interface comparing unrelated interface types comparing an array to a non-array comparing two arrays Specifically in the case of arrays, since arrays don't override Object.equals(), calling equals on two arrays is the same as comparing their addresses. This	BUG	MAJOR	1

means that array1.equals(array2) is equivalent to array1==array2. However, some developers might expect Array.equals(Object obj) to do more than a simple memory address comparison, comparing for instance the size and content of the two arrays. Instead, the == operator or Arrays.equals(array1, array2) should always be used with arrays.

Noncompliant Code Example

```
interface KitchenTool { ... }; interface Plant {...} public class Spatula implements KitchenTool { ... } public class Tree implements Plant { ...} //... Spatula spatula = new Spatula(); KitchenTool tool = spatula; KitchenTool [] tools = {tool}; Tree tree = new Tree(); Plant plant = tree; Tree [] trees = {tree};   if (spatula.equals(tree)) { // Noncompliant;
```

	unrelated classes // ... } else if (spatula.equals(plant)) { // Noncompliant; unrelated class and interface // ... } else if (tool.equals(plan t)) { // Noncompliant; unrelated interfaces // ... } else if (tool.equals(tool s)) { // Noncompliant; array & non-array // ... } else if (trees.equals(to ols)) { // Noncompliant; incompatible arrays // ... } else if (tree.equals(null)) { // Noncompliant // ... } See CERT, EXP02-J. - Do not use the Object.equals() method to compare two arrays			
Null pointers should not be dereferenced	A reference to null should never be dereferenced/ac cessed. Doing so will cause a NullPointerExcep tion to be thrown. At best, such an exception will cause abrupt program termination. At	BUG	MAJOR	1

worst, it could expose debugging information that would be useful to an attacker, or it could allow an attacker to bypass security measures. Note that when they are present, this rule takes advantage of @CheckForNull and @Nonnull annotations defined in JSR-305 to understand which values are and are not nullable except when @Nonnull is used on the parameter to equals, which by contract should always work with null. Noncompliant Code Example @CheckForNull String getName(){...} public boolean isNameEmpty() { return getName().length() == 0; // Noncompliant; the result of getName() could be null, but isn't null-checked } Connection conn = null; Statement stmt = null; try{ conn = DriverManager.getConnection(DB_URL,USER,PASS

```
);   stmt =
conn.createStat
ement();   // ...  }
catch(Exception
e)
{   e.printStackT
race(); }finally{
stmt.close();   //
Noncompliant;
stmt could be
null if an
exception was
thrown in the
try{} block
conn.close();  //
Noncompliant;
conn could be
null if an
exception was
thrown }
private void
merge(@Nonnull
Color firstColor,
@Nonnull Color
secondColor)
{...}  public
void
append(@Check
ForNull Color
color)
{     merge(curre
ntColor,
color);  //
Noncompliant;
color should be
null-checked
because
merge(...)
doesn't accept
nullable
parameters }
void paint(Color
color) {   if(color
== null)
{     System.out.
println("Unable
to apply color "
+
color.toString());
// Noncompliant;
NullPointerExcep
tion will be
thrown     return;
```

	} ... } See MITRE, CWE-476 - NULL Pointer Dereference CERT, EXP34-C. - Do not dereference null pointers CERT, EXP01-J. - Do not use a null in a case where an object is required			
"equals(Object obj)" and "hashCode()" should be overridden in pairs	According to the Java Language Specification, there is a contract between equals(Object) and hashCode(): If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result. It is not required that if two objects are unequal according to the equals(java.lang .Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer	BUG	MINOR	40

results for unequal objects may improve the performance of hashtables. In order to comply with this contract, those methods should be either both inherited, or both overridden. Noncompliant Code Example class MyClass { // Noncompliant - should also override "hashCode()" @Override public boolean equals(Object obj) { /* ... */ } } Compliant Solution class MyClass { // Compliant @Override public boolean equals(Object obj) { /* ... */ } @Override public int hashCode() { /* ... */ } } See MITRE, CWE-581 - Object Model Violation: Just One of Equals and Hashcode Defined CERT, MET09-J. - Classes that define an equals() method must also define a hashCode() method

---

Math operands should be cast before assignment