|  |
| --- |
| backend  Version 0.0.1-SNAPSHOT  Code analysis |

|  |
| --- |
| **By: default**  **2024-11-30** |

# Content

[Content 1](#_Toc97156258)

[Introduction 2](#_Toc97156259)

[Configuration 2](#_Toc97156260)

[Synthesis 3](#_Toc97156261)

[Analysis Status 3](#_Toc97156262)

[Quality gate status 3](#_Toc97156263)

[Metrics 3](#_Toc97156264)

[Tests 3](#_Toc97156265)

[Detailed technical debt 3](#_Toc97156266)

[Metrics Range 5](#_Toc97156267)

[Volume 5](#_Toc97156268)

[Issues 6](#_Toc97156269)

[Charts 6](#_Toc97156270)

[Issues count by severity and type 8](#_Toc97156271)

[Issues List 8](#_Toc97156272)

[Security Hotspots 9](#_Toc97156273)

[Security hotspots count by category and priority 9](#_Toc97156274)

[Security hotspots List 9](#_Toc97156275)

# Introduction

This document contains results of the code analysis of backend.

Pingeso

# Configuration

* Quality Profiles
  + Names: Sonar way [Java]; Sonar way [XML];
  + Files: AZNrmCLoxUS2\_b0QqNN-.json; AZNrmCPSxUS2\_b0QqNXm.json;
* Quality Gate
  + Name: Sonar way
  + File: Sonar way.xml

# Synthesis

## Analysis Status

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Security Review | Maintainability |
| A.png | **A.png** | **E.png** | **A.png** |

## Quality gate status

|  |  |
| --- | --- |
| Quality Gate Status | **OK.png** |

|  |  |
| --- | --- |
| Metric | Value |
| Reliability Rating on New Code | OK |
| Security Rating on New Code | OK |
| Maintainability Rating on New Code | OK |

## Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Coverage | Duplication | Comment  density | Median number of lines of code per file | Adherence to coding standard |
| 100.0 % | **0.0 %** | **9.7 %** | **193.0** | **99.8 %** |

## Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total | Success Rate | Skipped | Errors | Failures |
| 18 | **100.0 %** | **0** | **0** | **0** |

## Detailed technical debt

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Maintainability | Total |
| - | - | 0d 0h 36min | 0d 0h 36min |

## Metrics Range

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cyclomatic  Complexity | Cognitive  Complexity | Lines of code per file | Comment  density (%) | Coverage | Duplication (%) |
| Min | 3.0 | 0.0 | 27.0 | 0.0 | 100.0 | 0.0 |
| Max | 56.0 | 39.0 | 343.0 | 20.5 | 100.0 | 0.0 |

## Volume

|  |  |
| --- | --- |
| Language | Number |
| Java | 343 |
| XML | 178 |
| Total | 521 |

# Issues

## Charts

## Issues count by severity and type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type / Severity | INFO | MINOR | MAJOR | CRITICAL | BLOCKER |
| BUG | 0 | 0 | 0 | 0 | 0 |
| VULNERABILITY | 0 | 0 | 0 | 0 | 0 |
| CODE\_SMELL | 0 | 2 | 0 | 2 | 0 |

## Issues List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| Try-with-resources should be used | Java 7 introduced the try-with-resources statement, which guarantees that the resource in question will be closed. Since the new syntax is closer to bullet-proof, it should be preferred over the older try/catch/finally version. This rule checks that close-able resources are opened in a try-with-resources statement. Note that this rule is automatically disabled when the project’s sonar.java.source is lower than 7. Noncompliant Code Example FileReader fr = null; BufferedReader br = null; try { fr = new FileReader(fileName); br = new BufferedReader(fr); return br.readLine(); } catch (...) { } finally { if (br != null) { try { br.close(); } catch(IOException e){...} } if (fr != null ) { try { br.close(); } catch(IOException e){...} } } Compliant Solution try ( FileReader fr = new FileReader(fileName); BufferedReader br = new BufferedReader(fr) ) { return br.readLine(); } catch (...) {} or try (BufferedReader br = new BufferedReader(new FileReader(fileName))) { // no need to name intermediate resources if you don't want to return br.readLine(); } catch (...) {} See CERT, ERR54-J. - Use a try-with-resources statement to safely handle closeable resources | CODE\_SMELL | CRITICAL | 1 |
| Cognitive Complexity of methods should not be too high | Cognitive Complexity is a measure of how hard the control flow of a method is to understand. Methods with high Cognitive Complexity will be difficult to maintain. Exceptions equals and hashCode methods are ignored because they might be automatically generated and might end up being difficult to understand, especially in presence of many fields. See Cognitive Complexity | CODE\_SMELL | CRITICAL | 1 |
| Strings should not be concatenated using '+' in a loop | Strings are immutable objects, so concatenation doesn’t simply add the new String to the end of the existing string. Instead, in each loop iteration, the first String is converted to an intermediate object type, the second string is appended, and then the intermediate object is converted back to a String. Further, performance of these intermediate operations degrades as the String gets longer. Therefore, the use of StringBuilder is preferred. Noncompliant Code Example String str = ""; for (int i = 0; i &lt; arrayOfStrings.length ; ++i) { str = str + arrayOfStrings[i]; } Compliant Solution StringBuilder bld = new StringBuilder(); for (int i = 0; i &lt; arrayOfStrings.length; ++i) { bld.append(arrayOfStrings[i]); } String str = bld.toString(); | CODE\_SMELL | MINOR | 1 |
| "StandardCharsets" constants should be preferred | JDK7 introduced the class java.nio.charset.StandardCharsets. It provides constants for all charsets that are guaranteed to be available on every implementation of the Java platform. ISO\_8859\_1 US\_ASCII UTF\_16 UTF\_16BE UTF\_16LE UTF\_8 These constants should be preferred to: the use of a String such as "UTF-8" which has the drawback of requiring the catch/throw of an UnsupportedEncodingException that will never actually happen the use of Guava’s Charsets class, which has been obsolete since JDK7 Noncompliant Code Example try { byte[] bytes = string.getBytes("UTF-8"); // Noncompliant; use a String instead of StandardCharsets.UTF\_8 } catch (UnsupportedEncodingException e) { throw new AssertionError(e); } // ... byte[] bytes = string.getBytes(Charsets.UTF\_8); // Noncompliant; Guava way obsolete since JDK7 Compliant Solution byte[] bytes = string.getBytes(StandardCharsets.UTF\_8) | CODE\_SMELL | MINOR | 1 |

# Security Hotspots

## Security hotspots count by category and priority

|  |  |  |  |
| --- | --- | --- | --- |
| Category / Priority | LOW | MEDIUM | HIGH |
| LDAP Injection | 0 | 0 | 0 |
| Object Injection | 0 | 0 | 0 |
| Server-Side Request Forgery (SSRF) | 0 | 0 | 0 |
| XML External Entity (XXE) | 0 | 0 | 0 |
| Insecure Configuration | 1 | 0 | 0 |
| XPath Injection | 0 | 0 | 0 |
| Authentication | 0 | 0 | 0 |
| Weak Cryptography | 0 | 0 | 0 |
| Denial of Service (DoS) | 0 | 0 | 0 |
| Log Injection | 0 | 0 | 0 |
| Cross-Site Request Forgery (CSRF) | 0 | 0 | 0 |
| Open Redirect | 0 | 0 | 0 |
| Permission | 0 | 0 | 0 |
| SQL Injection | 0 | 0 | 0 |
| Encryption of Sensitive Data | 0 | 0 | 0 |
| Traceability | 0 | 0 | 0 |
| Buffer Overflow | 0 | 0 | 0 |
| File Manipulation | 0 | 0 | 0 |
| Code Injection (RCE) | 0 | 0 | 0 |
| Cross-Site Scripting (XSS) | 0 | 0 | 0 |
| Command Injection | 0 | 0 | 0 |
| Path Traversal Injection | 0 | 0 | 0 |
| HTTP Response Splitting | 0 | 0 | 0 |
| Others | 0 | 0 | 0 |

## Security hotspots List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Name | Priority | Severity | Count |
| Insecure Configuration | Having a permissive Cross-Origin Resource Sharing policy is security-sensitive | LOW | MINOR | 1 |