|  |
| --- |
| TWD\_GRUPO1  Version not provided  Code analysis |

|  |
| --- |
| **By: Administrator**  **2024-11-27** |

# 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 TWD\_GRUPO1.

# Configuration

* Quality Profiles
  + Names: Sonar way [CSS]; Sonar way [Java]; Sonar way [JavaScript]; Sonar way [HTML]; Sonar way [XML];
  + Files: AZLA0SfYUhnVQK0JYNUD.json; AZLA0S-KUhnVQK0JYONU.json; AZLA0SpzUhnVQK0JYNi7.json; AZLA0TDkUhnVQK0JYOVt.json; AZLA0TFcUhnVQK0JYOW8.json;
* Quality Gate
  + Name: Sonar way
  + File: Sonar way.xml

# Synthesis

## Analysis Status

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

## Quality gate status

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



## Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Coverage | Duplication | Comment  density | Median number of lines of code per file | Adherence to coding standard |
| 0.0 % | **7.7 %** | **9.4 %** | **56.5** | **99.3 %** |

## Tests

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

## Detailed technical debt

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Maintainability | Total |
| 0d 0h 10min | - | 0d 3h 43min | 0d 3h 53min |

## Metrics Range

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cyclomatic  Complexity | Cognitive  Complexity | Lines of code per file | Comment  density (%) | Coverage | Duplication (%) |
| Min | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Max | 460.0 | 141.0 | 7907.0 | 58.3 | 0.0 | 100.0 |

## Volume

|  |  |
| --- | --- |
| Language | Number |
| CSS | 4137 |
| Java | 862 |
| JavaScript | 3445 |
| HTML | 232 |
| XML | 97 |
| Total | 8773 |

# Issues

## Charts

## Issues count by severity and type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type / Severity | INFO | MINOR | MAJOR | CRITICAL | BLOCKER |
| BUG | 0 | 2 | 0 | 0 | 0 |
| VULNERABILITY | 0 | 0 | 0 | 0 | 0 |
| CODE\_SMELL | 0 | 8 | 35 | 0 | 0 |

## Issues List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| "<table>" tags should have a description |  | BUG | MINOR | 2 |
| Empty blocks should be removed | Leftover empty blocks are usually introduced by mistake. They are useless and prevent readability of the code. They should be removed or completed with real code. Noncompliant Code Example a { } Compliant Solution a { color: pink; } | CODE\_SMELL | MAJOR | 1 |
| Selectors should not be duplicated | Duplication of selectors might indicate a copy-paste mistake. The rule detects the following kinds of duplications: within a list of selectors in a single rule set for duplicated selectors in different rule sets within a single stylesheet. Noncompliant Code Example .foo, .bar, .foo { ... } /\* Noncompliant \*/ .class1 { ... } .class1 { ... } /\* Noncompliant \*/ Compliant Solution .foo, .bar { ... } .class1 { ... } .class2 { ... } | CODE\_SMELL | MAJOR | 3 |
| CSS files should not be empty | This rule raises an issue when a CSS file is empty (ie: containing only spaces). | CODE\_SMELL | MAJOR | 1 |
| Unused "private" fields should be removed | If a private field is declared but not used in the program, it can be considered dead code and should therefore be removed. This will improve maintainability because developers will not wonder what the variable is used for. Note that this rule does not take reflection into account, which means that issues will be raised on private fields that are only accessed using the reflection API. Noncompliant Code Example public class MyClass { private int foo = 42; public int compute(int a) { return a \* 42; } } Compliant Solution public class MyClass { public int compute(int a) { return a \* 42; } } Exceptions The rule admits 3 exceptions: Serialization id fields Annotated fields Fields from classes with native methods Serialization id fields The Java serialization runtime associates with each serializable class a version number, called serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. A serializable class can declare its own serialVersionUID explicitly by declaring a field named serialVersionUID that must be static, final, and of type long. By definition those serialVersionUID fields should not be reported by this rule: public class MyClass implements java.io.Serializable { private static final long serialVersionUID = 42L; } Annotated fields The unused field in this class will not be reported by the rule as it is annotated. public class MyClass { @SomeAnnotation private int unused; } Fields from classes with native methods The unused field in this class will not be reported by the rule as it might be used by native code. public class MyClass { private int unused = 42; private native static void doSomethingNative(); } | CODE\_SMELL | MAJOR | 14 |
| Unused assignments should be removed | A dead store happens when a local variable is assigned a value that is not read by any subsequent instruction. Calculating or retrieving a value only to then overwrite it or throw it away, could indicate a serious error in the code. Even if it’s not an error, it is at best a waste of resources. Therefore all calculated values should be used. Noncompliant Code Example i = a + b; // Noncompliant; calculation result not used before value is overwritten i = compute(); Compliant Solution i = a + b; i += compute(); Exceptions This rule ignores initializations to -1, 0, 1, undefined, [], {}, true, false and "". Variables that start with an underscore (e.g. '\_unused') are ignored. Assignment of null is ignored because it is sometimes used to help garbage collection Increment and decrement expressions are ignored because they are often used idiomatically instead of x+1 This rule also ignores variables declared with object destructuring using rest syntax (used to exclude some properties from object): let {a, b, ...rest} = obj; // 'a' and 'b' are ok doSomething(rest); let [x1, x2, x3] = arr; // but 'x1' is noncompliant, as omitting syntax can be used: "let [, x2, x3] = arr;" doSomething(x2, x3); See MITRE, CWE-563 - Assignment to Variable without Use ('Unused Variable') | CODE\_SMELL | MAJOR | 4 |
| Ternary operators should not be nested |  | CODE\_SMELL | MAJOR | 2 |
| No array index for keys in JSX list components |  | CODE\_SMELL | MAJOR | 9 |
| React Context Provider values should not have non-stable identities |  | CODE\_SMELL | MAJOR | 1 |
| Return of boolean expressions should not be wrapped into an "if-then-else" statement | Return of boolean literal statements wrapped into if-then-else flow should be simplified. Note that if the result of the expression is not a boolean but for instance an integer, then double negation should be used for proper conversion. if (expression) { return true; } else { return false; } or if (expression) { return true; } return false; Compliant Solution return expression; or return !!expression; | CODE\_SMELL | MINOR | 1 |
| Unnecessary imports should be removed | There’s no reason to import modules you don’t use; and every reason not to: doing so needlessly increases the load. Noncompliant Code Example import A from 'a'; // Noncompliant, A isn't used import { B1 } from 'b'; console.log(B1); Compliant Solution import { B1 } from 'b'; console.log(B1); | CODE\_SMELL | MINOR | 3 |
| Unused local variables and functions should be removed | If a local variable or a local function is declared but not used, it is dead code and should be removed. Doing so will improve maintainability because developers will not wonder what the variable or function is used for. Noncompliant Code Example function numberOfMinutes(hours) { var seconds = 0; // seconds is never used return hours \* 60; } Compliant Solution function numberOfMinutes(hours) { return hours \* 60; } | CODE\_SMELL | MINOR | 4 |

# 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 | 1 | 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 |
| Others | Disclosing fingerprints from web application technologies is security-sensitive | LOW | MINOR | 1 |