|  |
| --- |
| SonarTest  Version 0.0.1  Code analysis |

|  |
| --- |
| **By: default**  **2020-08-06** |

# Content

[Content 1](#_Toc7772423)

[Introduction 2](#_Toc7772424)

[Configuration 2](#_Toc7772425)

[Synthesis 2](#_Toc7772426)

[Metrics 2](#_Toc7772427)

[Volume 3](#_Toc7772428)

[Issues count by severity and type 3](#_Toc7772429)

[Charts 3](#_Toc7772430)

[Issues 3](#_Toc7772431)

# Introduction

This document contains results of the code analysis of SonarTest.

# Configuration

* Quality Profiles
  + Names: Sonar way [CSS]; Sonar way [JavaScript];
  + Files: AXO9OmHBMhI8b8VPHXKI.json; AXO9OmPAMhI8b8VPHXSK.json;
* Quality Gate
  + Name: Sonar way
  + File: Sonar way.xml

# Synthesis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Quality Gate | Reliability | Security | Maintainability | Coverage | Duplication |
| OK | **E** | **A** | **A** | **0.0 %** | **1.4 %** |

# Metrics

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cyclomatic  Complexity | Cognitive  Complexity | Lines of code per file | Comment  density (%) | Coverage | Duplication (%) |
| Min | 0.0 | 0.0 | 5.0 | 0.0 | 0.0 | 0.0 |
| Max | 679.0 | 362.0 | 3713.0 | 45.9 | XX-MAXCOVERAGE-XX | 86.7 |

# Volume

|  |  |
| --- | --- |
| Language | Number |
| CSS | 158 |
| JavaScript | 3555 |
| Total | 3713 |

# Issues count by severity and type

|  |  |  |
| --- | --- | --- |
| Type | Severity | Number |
| VULNERABILITY | BLOCKER | 0 |
| VULNERABILITY | CRITICAL | 0 |
| VULNERABILITY | MAJOR | 0 |
| VULNERABILITY | MINOR | 0 |
| VULNERABILITY | INFO | 0 |
| BUG | BLOCKER | 1 |
| BUG | CRITICAL | 0 |
| BUG | MAJOR | 35 |
| BUG | MINOR | 13 |
| BUG | INFO | 0 |
| CODE\_SMELL | BLOCKER | 0 |
| CODE\_SMELL | CRITICAL | 0 |
| CODE\_SMELL | MAJOR | 12 |
| CODE\_SMELL | MINOR | 4 |
| CODE\_SMELL | INFO | 0 |
| SECURITY\_HOTSPOT | BLOCKER | 0 |
| SECURITY\_HOTSPOT | CRITICAL | 0 |
| SECURITY\_HOTSPOT | MAJOR | 0 |
| SECURITY\_HOTSPOT | MINOR | 0 |
| SECURITY\_HOTSPOT | INFO | 0 |

# Charts

# Issues

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| Callbacks of array methods should have return statements | Arrays in JavaScript have several methods for filtering, mapping or folding that require a callback. Not having a return statement in such a callback function is most likely a mistake. This rule applies for the following methods of an array: Array.from Array.prototype.every Array.prototype.filter Array.prototype.find Array.prototype.findIndex Array.prototype.map Array.prototype.reduce Array.prototype.reduceRight Array.prototype.some Array.prototype.sort Noncompliant Code Example var merged = arr.reduce(function(a, b) { a.concat(b); }); // Noncompliant: No return statement Compliant Solution var merged = arr.reduce(function(a, b) { return a.concat(b); }); | BUG | BLOCKER | 1 |
| Font declarations should contain at least one generic font family | If none of the font names defined in a font or font-family declaration are available on the browser of the user, the browser will display the text using its default font. It's recommended to always define a generic font family for each declaration of font or font-family to get a less degraded situation than relying on the default browser font. All browsers should implement a list of generic font matching these families: Serif, Sans-serif, cursive, fantasy, Monospace. Noncompliant Code Example a { font-family: Helvetica, Arial, Verdana, Tahoma; /\* Noncompliant; there is no generic font family in the list \*/ } Compliant Solution a { font-family: Helvetica, Arial, Verdana, Tahoma, sans-serif; } See CSS Specification - Generic font families | BUG | MAJOR | 28 |
| Properties should not be duplicated | CSS allows duplicate property names but only the last instance of a duplicated name determines the actual value that will be used for it. Therefore, changing values of other occurrences of a duplicated name will have no effect and may cause misunderstandings and bugs. This rule ignores $sass, @less, and var(--custom-property) variable syntaxes. Noncompliant Code Example a { color: pink; background: orange; color: orange } Compliant Solution a { color: pink; background: orange } | BUG | MAJOR | 6 |
| Strict equality operators should not be used with dissimilar types | Comparing dissimilar types using the strict equality operators === and !== will always return the same value, respectively false and true, because no type conversion is done before the comparison. Thus, such comparisons can only be bugs. Noncompliant Code Example var a = 8; var b = "8"; if (a === b) { // Noncompliant; always false // ... } Compliant Solution var a = 8; var b = "8"; if (a == b) { // ... } or var a = 8; var b = "8"; if (a === Number(b)) { // ... } | BUG | MAJOR | 1 |
| "<strong>" and "<em>" tags should be used |  | BUG | MINOR | 6 |
| Image, area and button with image tags should have an "alt" attribute |  | BUG | MINOR | 7 |
| Sections of code should not be commented out | Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required. | CODE\_SMELL | MAJOR | 1 |
| 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 | 3 |
| 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 | 4 |
| CSS files should not be empty | This rule raises an issue when a CSS file is empty (ie: containing only spaces). | CODE\_SMELL | MAJOR | 2 |
| Variables should not be shadowed | Overriding or shadowing a variable declared in an outer scope can strongly impact the readability, and therefore the maintainability, of a piece of code. Further, it could lead maintainers to introduce bugs because they think they're using one variable but are really using another. See CERT, DCL01-C. - Do not reuse variable names in subscopes CERT, DCL51-J. - Do not shadow or obscure identifiers in subscopes | CODE\_SMELL | MAJOR | 1 |
| Array-mutating methods should not be used misleadingly | Many of JavaScript's Array methods return an altered version of the array while leaving the source array intact. reverse and sort do not fall into this category. Instead, they alter the source array in addition to returning the altered version, which is likely not what was intended. This rule raises an issue when the return values of these methods are assigned, which could lead maintainers to overlook the fact that the original value is altered. Noncompliant Code Example var b = a.reverse(); // Noncompliant var d = c.sort(); // Noncompliant Compliant Solution var b = [...a].reverse(); // de-structure and create a new array, so reverse doesn't impact 'a' a.reverse(); c.sort(); // this sorts array in place | 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 ones 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. Noncompliant Code Example if (expression) { return true; } else { return false; } Compliant Solution return expression; or return !!expression; | CODE\_SMELL | MINOR | 1 |
| "switch" statements should have at least 3 "case" clauses | switch statements are useful when there are many different cases depending on the value of the same expression. For just one or two cases however, the code will be more readable with if statements. Noncompliant Code Example switch (variable) { case 0: doSomething(); break; default: doSomethingElse(); break; } Compliant Solution if (variable == 0) { doSomething(); } else { doSomethingElse(); } | CODE\_SMELL | MINOR | 1 |
| Default export names and file names should match | By convention, a file that exports only one class, function, or constant should be named for that class, function or constant. Anything else may confuse maintainers. Noncompliant Code Example // file path: myclass.js -- Noncompliant class MyClass { // ... } export default MyClass; Compliant Solution // file path: MyClass.js class MyClass { // ... } export default MyClass; Exceptions Case, underscores ( \_ ) and dashes (-) are ignored from the name comparison. | CODE\_SMELL | MINOR | 2 |