|  |
| --- |
| Metadata\_Framework  Version 1.0  Code analysis |

|  |
| --- |
| **By: default**  **2023-08-11** |

# Content

[Content 1](#_Toc79573867)

[Introduction 2](#_Toc79573868)

[Configuration 2](#_Toc79573869)

[Synthesis 3](#_Toc79573870)

[Analysis Status 3](#_Toc79573871)

[Quality gate status 3](#_Toc79573872)

[Metrics 3](#_Toc79573873)

[Tests 3](#_Toc79573874)

[Detailed technical debt 3](#_Toc79573875)

[Metrics Range 5](#_Toc79573876)

[Volume 5](#_Toc79573877)

[Issues 6](#_Toc79573878)

[Charts 6](#_Toc79573879)

[Issues count by severity and type 8](#_Toc79573880)

[Issues List 8](#_Toc79573881)

[Security Hotspots 9](#_Toc79573882)

[Security hotspots count by category and priority 9](#_Toc79573883)

[Security hotspots List 9](#_Toc79573884)

# Introduction

This document contains results of the code analysis of Metadata\_Framework.

# Configuration

* Quality Profiles
  + Names: Sonar way [Python];
  + Files: AYeVFt0viG7mriH640Ur.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** |



## Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Coverage | Duplication | Comment  density | Median number of lines of code per file | Adherence to coding standard |
| 12.4 % | **18.5 %** | **16.0 %** | **1310.0** | **99.6 %** |

## Tests

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

## Detailed technical debt

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Maintainability | Total |
| - | - | 1d 2h 10min | 1d 2h 10min |

## Metrics Range

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cyclomatic  Complexity | Cognitive  Complexity | Lines of code per file | Comment  density (%) | Coverage | Duplication (%) |
| Min | 1.0 | 0.0 | 87.0 | 0.3 | 0.0 | 0.0 |
| Max | 196.0 | 629.0 | 8745.0 | 50.0 | 47.3 | 40.6 |

## Volume

|  |  |
| --- | --- |
| Language | Number |
| Python | 8745 |
| Total | 8745 |

# 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 | 7 | 58 | 13 | 0 |

## Issues List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| String literals should not be duplicated | Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences. On the other hand, constants can be referenced from many places, but only need to be updated in a single place. Noncompliant Code Example With the default threshold of 3: def run(): prepare("this is a duplicate") # Noncompliant - "this is a duplicate" is duplicated 3 times execute("this is a duplicate") release("this is a duplicate") Compliant Solution ACTION\_1 = "action1" def run(): prepare(ACTION\_1) execute(ACTION\_1) release(ACTION\_1) Exceptions No issue will be raised on: duplicated string in decorators strings with less than 5 characters strings with only letters, numbers and underscores @app.route("/api/users/", methods=['GET', 'POST', 'PUT']) def users(): pass @app.route("/api/projects/", methods=['GET', 'POST', 'PUT']) # Compliant def projects(): pass | CODE\_SMELL | CRITICAL | 4 |
| Cognitive Complexity of functions should not be too high | Cognitive Complexity is a measure of how hard the control flow of a function is to understand. Functions with high Cognitive Complexity will be difficult to maintain. See Cognitive Complexity | CODE\_SMELL | CRITICAL | 3 |
| Bare "raise" statements should only be used in "except" blocks | A bare raise statement, i.e. a raise with no exception provided, will re-raise the last active exception in the current scope. If the "raise" statement is not in an except or finally block, no exception is active and a RuntimeError is raised instead. If the bare raise statement is in a function called in an except statement, the exception caught by the except will be raised. This works but is hard to understand and maintain. Nothing indicates in the parent except that the exception will be reraised, and nothing prevents a developer from calling the function in another context. Note also that using a bare raise in a finally block only works when an exception is active, i.e. when an exception from the try block is not caught or when an exception is raised by an except block. It will fail in any other case and should not be relied upon. This code smell is handled by rule S5704. This rule raises an exception when a bare raise statement is not in an except or finally block. Noncompliant Code Example raise # Noncompliant def foo(): raise # Noncompliant try: raise # Noncompliant except ValueError as e: handle\_error() except: raise else: raise # Noncompliant finally: raise def handle\_error(): raise # Noncompliant. This works but is hard to understand. Compliant Solution raise ValueError() def foo(): raise ValueError() try: raise ValueError() except: raise else: raise ValueError() finally: raise See Python Documentation - The raise statement | CODE\_SMELL | CRITICAL | 6 |
| 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 | 47 |
| Assertions should not fail or succeed unconditionally | Assertions are meant to detect when code behaves as expected. An assertion which fails or succeeds all the time should be fixed. This rule raises an issue when an assertion method is given parameters which will make it succeed or fail all the time. It covers three cases: an assert statement or a unittest’s assertTrue or assertFalse method is called with a value which will be always True or always False. a unittest’s assertIsNotNone or assertIsNone method is called with a value which will be always None or never None. a unittest’s assertIsNot or assertIs method is called with a literal expression creating a new object every time (ex: [1, 2, 3]). Noncompliant Code Example import unittest class MyTestCase(unittest.TestCase): def expect\_fail1(self): assert False def expect\_fail2(self): self.assertTrue(False) # Noncompliant. This assertion always fails. def expect\_not\_none(self): self.assertIsNotNone(round(1.5)) # Noncompliant. This assertion always succeeds because "round" returns a number, not None. def helper\_compare(param): self.assertIs(param, [1, 2, 3]) # Noncompliant. This assertion always fails because [1, 2, 3] creates a new object. Compliant Solution import unittest class MyTestCase(unittest.TestCase): def expect\_fail(self): self.fail("This is expected") def expect\_not\_none(self): self.assertNotEqual(round(1.5), 0) def helper\_compare(param): self.assertEqual(param, [1, 2, 3]) See Python documentation - the unittest module Python documentation - the assert statement | CODE\_SMELL | MAJOR | 11 |
| Local variable and function parameter names should comply with a naming convention | Shared naming conventions allow teams to collaborate effectively. This rule raises an issue when a local variable or function parameter name does not match the provided regular expression. Exceptions Loop counters are ignored by this rule. for i in range(limit): # Compliant print(i) | CODE\_SMELL | MINOR | 7 |

# 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 | 0 | 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 | 6 | 0 | 0 |
| Cross-Site Request Forgery (CSRF) | 0 | 0 | 0 |
| Open Redirect | 0 | 0 | 0 |
| SQL Injection | 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 |
| Log Injection | Configuring loggers is security-sensitive | LOW | CRITICAL | 6 |