|  |
| --- |
| Minh  Version 38  Code analysis |

|  |
| --- |
| **By: default**  **2022-11-09** |

# 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 Minh.

# Configuration

* Quality Profiles
  + Names: Sonar way [CSS]; Sonar way [Python]; Sonar way [HTML];
  + Files: AYRWmepCrqufXX4VQRN8.json; AYRWme28rqufXX4VQRe-.json; AYRWmfJQrqufXX4VQSFN.json;
* Quality Gate
  + Name: Sonar way
  + File: Sonar way.xml

# Synthesis

## Analysis Status

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Security Review | Maintainability |
| E.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 |
| 0.0 % | **23.2 %** | **3.3 %** | **44.0** | **97.9 %** |

## Tests

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

## Detailed technical debt

|  |  |  |  |
| --- | --- | --- | --- |
| Reliability | Security | Maintainability | Total |
| 0d 6h 53min | - | 10d 3h 40min | 11d 2h 33min |

## 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 | 1309.0 | 2763.0 | 17382.0 | 100.0 | 0.0 | 83.3 |

## Volume

|  |  |
| --- | --- |
| Language | Number |
| CSS | 1537 |
| Python | 8054 |
| HTML | 7791 |
| Total | 17382 |

# Issues

## Charts

## Issues count by severity and type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type / Severity | INFO | MINOR | MAJOR | CRITICAL | BLOCKER |
| BUG | 0 | 120 | 174 | 1 | 1 |
| VULNERABILITY | 0 | 0 | 0 | 0 | 0 |
| CODE\_SMELL | 0 | 119 | 89 | 224 | 0 |

## Issues List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| Instance and class methods should have at least one positional parameter | Every instance method is expected to have at least one positional parameter. This parameter will reference the object instance on which the method is called. Calling an instance method which doesn't have at least one parameter will raise a "TypeError". By convention, this first parameter is usually named "self". Class methods, i.e. methods annotated with @classmethod, also require at least one parameter. The only differences is that it will receive the class itself instead of a class instance. By convention, this first parameter is usually named "cls". Note that \_\_new\_\_ and \_\_init\_subclass\_\_ take a class as first argument even thought they are not decorated with @classmethod. This rule raises an issue when an instance of class method does not have at least one positional parameter. Noncompliant Code Example class MyClass: def instance\_method(): # Noncompliant. "self" parameter is missing. print("instance\_method") @classmethod def class\_method(): # Noncompliant. "cls" parameter is missing. print("class\_method") Compliant Solution class MyClass: def instance\_method(self): print("instance\_method") @classmethod def class\_method(cls): print("class\_method") @staticmethod def static\_method(): print("static\_method") See Python documentation - Method Objects | BUG | BLOCKER | 1 |
| Shorthand properties that override related longhand properties should be avoided | A shorthand property defined after a longhand property will completely override the value defined in the longhand property making the longhand one useless. The code should be refactored to consider the longhand property or to remove it completely. Noncompliant Code Example a { padding-left: 10px; padding: 20px; /\* Noncompliant; padding is overriding padding-left making it useless \*/ } Compliant Solution a { padding: 10px; /\* Compliant; padding is defining a general behaviour and padding-left, just after, is precising the left case \*/ padding-left: 20px; } See Mozilla Web Technology for Developers - Shorthand properties | BUG | CRITICAL | 1 |
| Tables should have headers |  | BUG | MAJOR | 13 |
| "<th>" tags should have "id" or "scope" attributes |  | BUG | MAJOR | 155 |
| 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 | 1 |
| 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 | 4 |
| All code should be reachable | Jump statements (return, break, continue, and raise) move control flow out of the current code block. So any statements that come after a jump are dead code. Noncompliant Code Example def fun(a): i = 10 return i + a # Noncompliant i += 1 # this is never executed Compliant Solution def fun(a): i = 10 return i + a See MITRE, CWE-561 - Dead Code CERT, MSC56-J. - Detect and remove superfluous code and values CERT, MSC12-C. - Detect and remove code that has no effect or is never executed | BUG | MAJOR | 1 |
| "<strong>" and "<em>" tags should be used |  | BUG | MINOR | 69 |
| Image, area and button with image tags should have an "alt" attribute |  | BUG | MINOR | 8 |
| "<table>" tags should have a description |  | BUG | MINOR | 43 |
| Functions and methods should not be empty | There are several reasons for a function or a method not to have a body: It is an unintentional omission, and should be fixed to prevent an unexpected behavior in production. It is not yet, or never will be, supported. In this case an exception should be thrown. The method is an intentionally-blank override. In this case a nested comment should explain the reason for the blank override. Noncompliant Code Example def myfunc1(foo="Noncompliant"): pass class MyClass: def mymethod1(self, foo="Noncompliant"): pass Compliant Solution def myfunc1(): pass # comment explaining why this function is empty def myfunc2(): raise NotImplementedError() def myfunc3(): """ Docstring explaining why this function is empty. """ class MyClass: def mymethod1(self): pass # comment explaining why this function is empty def mymethod2(self): raise NotImplementedError() def mymethod3(self): """ Docstring explaining why this method is empty. Note that this is not recommended for classes which are meant to be subclassed. """ Exceptions No issue will be raised when the empty method is abstract and meant to be overriden in a subclass, i.e. it is decorated with abc.abstractmethod, abc.abstractstaticmethod, abc.abstractclassmethod or abc.abstractproperty. Note however that these methods should normally have a docstring explaining how subclasses should implement these methods. import abc class MyAbstractClass(abc.ABC): @abc.abstractproperty def myproperty(self): pass @abc.abstractclassmethod def myclassmethod(cls): pass @abc.abstractmethod def mymethod(self): pass @abc.abstractstaticmethod def mystaticmethod(): pass | CODE\_SMELL | CRITICAL | 2 |
| 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 | 131 |
| 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 | 47 |
| "SystemExit" should be re-raised | SystemExit&nbsp;is raised when sys.exit() is called. This exception is expected to propagate up until the application stops. It is ok to catch it when a clean-up is necessary but it should be raised again immediately. A bare except: statement, i.e. an except without any exception class, is equivalent to except BaseException. Both statements will catch every exception, including SystemExit. It is recommended to catch instead a specific exception. If it is not possible, the exception should be raised again. Note that it is also a good idea to reraise the KeyboardInterrupt exception. This rule raises an issue when a bare except:, an except BaseException or an except SystemExit don't reraise the exception caught. Noncompliant Code Example try: open("foo.txt", "r") except SystemExit: # Noncompliant pass except KeyboardInterrupt: # No issue raised but be careful when you do this pass try: open("bar.txt", "r") except BaseException: # Noncompliant pass except: # Noncompliant pass Compliant Solution try: open("foo.txt", "r") except SystemExit: # clean-up raise except KeyboardInterrupt: # clean-up raise try: open("bar.txt", "r") except BaseException as e: # clean-up raise e except: # Noncompliant # clean-up raise # or use a more specific exception try: open("bar.txt", "r") except FileNotFoundError: # process the exception See PEP 352 - Required Superclass for Exceptions Python Documentation - Built-in exceptions Python Documentation - The try statement MITRE, CWE-391 - Unchecked Error Condition | CODE\_SMELL | CRITICAL | 44 |
| 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 |
| 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 |
| Collapsible "if" statements should be merged | Merging collapsible if statements increases the code's readability. Noncompliant Code Example if condition1: if condition2: # ... Compliant Solution if condition1 and condition2: # ... | CODE\_SMELL | MAJOR | 2 |
| Nested blocks of code should not be left empty | Most of the time a block of code is empty when a piece of code is really missing. So such empty block must be either filled or removed. Noncompliant Code Example for i in range(3): pass Exceptions When a block contains a comment, this block is not considered to be empty. | CODE\_SMELL | MAJOR | 7 |
| Redundant pairs of parentheses should be removed | The use of parentheses, even those not required to enforce a desired order of operations, can clarify the intent behind a piece of code. But redundant pairs of parentheses could be misleading, and should be removed. Noncompliant Code Example return ((3)) # Noncompliant return ((x + 1)) # Noncompliant x = ((y / 2)) + 1 # Noncompliant Compliant Solution return 3 return (3) return x + 1 return (x + 1) x = y / 2 + 1 x = (y / 2) + 1 | CODE\_SMELL | MAJOR | 1 |
| 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 | 67 |
| 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 def func(a, b, compute): i = a + b # Noncompliant; calculation result not used before value is overwritten i = compute() # Noncompliant; the value is not used before leaving the function Compliant Solution def func(a, b, compute): i = a + b i += compute() return i Exceptions This rule ignores initializations to -1, 0, 1, None, True, False and "". No issue will be raised on unpacked variables. See MITRE, CWE-563 - Assignment to Variable without Use ('Unused Variable') CERT, MSC13-C. - Detect and remove unused values CERT, MSC56-J. - Detect and remove superfluous code and values | CODE\_SMELL | MAJOR | 2 |
| Two branches in a conditional structure should not have exactly the same implementation |  | CODE\_SMELL | MAJOR | 1 |
| Builtins should not be shadowed by local variables | Shadowing a builtin makes your code more difficult to read and maintain. It may also be a source of bugs as you can reference the builtin by mistake. It is sometimes ok to shadow a builtin to improve the readability of a public API or to support multiple versions of a library. In these cases the value is higher than the maintainability cost. Just be careful when you do it. It is not ok to shadow builtins with variables which are local to a function or method. These variables are not public and can be easily renamed, thus reducing the confusion and making the code less error-prone. This rule raises an issue when the name of a local variable matches the name of a builtin. Noncompliant Code Example def a\_function(): int = 42 # Noncompliant; int is a builtin Compliant Solution def a\_function(): value = 42 See Python documentation - Built-in Types Python documentation - Built-in Functions | CODE\_SMELL | MAJOR | 4 |
| Class names should comply with a naming convention | Shared coding conventions allow teams to collaborate effectively. This rule allows to check that all class names match a provided regular expression. The default regular expression is based on PEP-8 standard. It allows "CapWords" convention and "snake\_case" in lowercase. The "snake\_case" convention is accepted by PEP-8 when the class is primarily used as a callable (ex: decorator, context manager, etc...). However the "CapWords" convention is recommended in every case. Noncompliant Code Example With default provided regular expression ^\_?([A-Z\_][a-zA-Z0-9]\*|[a-z\_][a-z0-9\_]\*)$: class myClass: # Noncompliant ... class my\_CONTEXT\_manager: # Noncompliant def \_\_enter\_\_(self): pass def \_\_exit\_\_(self, type, value, traceback): pass Compliant Solution class MyClass: ... class my\_context\_manager: def \_\_enter\_\_(self): pass def \_\_exit\_\_(self, type, value, traceback): pass | CODE\_SMELL | MINOR | 1 |
| 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 | 26 |
| Unused local variables should be removed | If a local variable 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 is used for. Noncompliant Code Example def hello(name): message = "Hello " + name # Noncompliant print(name) for i in range(10): foo() Compliant Solution def hello(name): message = "Hello " + name print(message) for \_ in range(10): foo() Exceptions \_ as well as tuples will not raise an issue for this rule. The following examples are compliant: for \_ in range(10): do\_something() username, login, password = auth do\_something\_else(username, login) | CODE\_SMELL | MINOR | 78 |
| "pass" should not be used needlessly | The use of a pass statement where it's not required by the syntax is pure cruft and should be removed. Noncompliant Code Example def \_\_init\_\_(self, log="", who="", date=0, files=[]): self.log = log self.files = files self.who = who self.date = date pass # Noncompliant def lookup(): pass # Compliant; method can't be empty Compliant Solution def \_\_init\_\_(self, log="", who="", date=0, files=[]): self.log = log self.files = files self.who = who self.date = date def lookup(): pass | CODE\_SMELL | MINOR | 14 |

# 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 | 5 |
| Weak Cryptography | 0 | 1 | 0 |
| Denial of Service (DoS) | 0 | 0 | 0 |
| Log Injection | 0 | 0 | 0 |
| Cross-Site Request Forgery (CSRF) | 0 | 0 | 7 |
| 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 | 83 | 0 | 0 |

## Security hotspots List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Category | Name | Priority | Severity | Count |
| Cross-Site Request Forgery (CSRF) | Allowing both safe and unsafe HTTP methods is security-sensitive | HIGH | MINOR | 7 |
| Weak Cryptography | Using pseudorandom number generators (PRNGs) is security-sensitive | MEDIUM | CRITICAL | 1 |
| Authentication | Hard-coded credentials are security-sensitive | HIGH | BLOCKER | 5 |
| Insecure Configuration | Delivering code in production with debug features activated is security-sensitive | LOW | MINOR | 1 |
| Others | Using clear-text protocols is security-sensitive | LOW | CRITICAL | 60 |
| Others | Disabling resource integrity features is security-sensitive | LOW | MINOR | 20 |
| Others | Using hardcoded IP addresses is security-sensitive | LOW | MINOR | 3 |