



Sonarqube

SonarQube

Como utilizar, criar regras e garantir código limpo

- **O que é o SonarQube?**

O SonarQube é uma plataforma de código aberto projetada para a análise contínua de código fonte, permitindo que equipes de desenvolvimento monitorem a qualidade do software e identifiquem problemas técnicos de forma proativa. Esta ferramenta não apenas promove a criação de um **código limpo**, ao fornecer métricas e feedbacks que incentivam a escrita de um código legível e sustentável, mas também reforça **boas práticas** de desenvolvimento. O SonarQube oferece uma gama abrangente de relatórios que ajudam a garantir que o código atenda a padrões de qualidade estabelecidos. Além disso, integra-se facilmente a ambientes de desenvolvimento e CI/CD, como o nosso **gitlab.qitech**, proporcionando uma visão clara da saúde do código em diferentes estágios do ciclo de vida do desenvolvimento de software.

- **Conceito de Código limpo?**

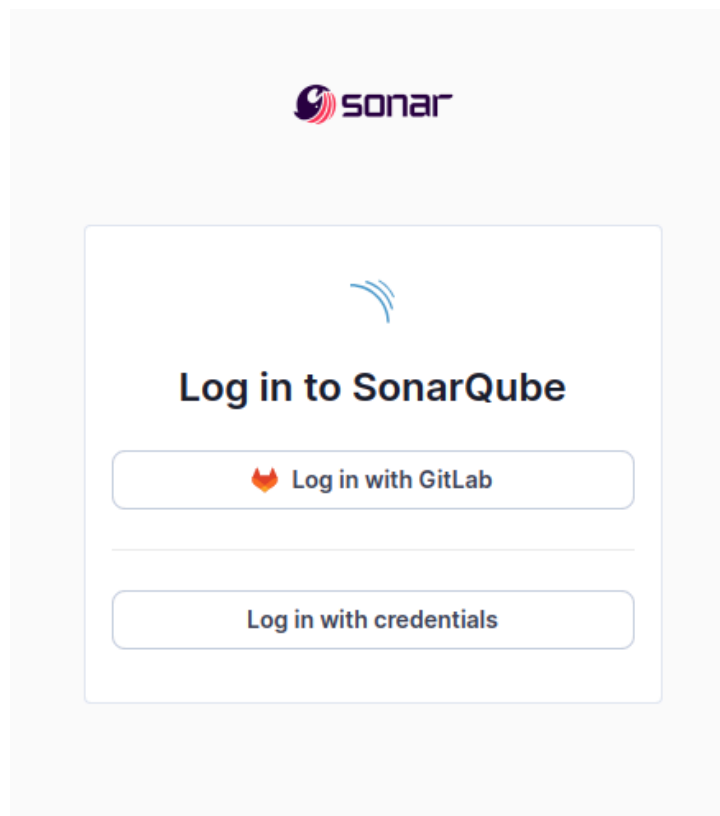
Código limpo refere-se a uma **filosofia de desenvolvimento** que prioriza a **legibilidade**, **simplicidade** e **manutenção** do código. Um código limpo é fácil de entender, o que facilita a colaboração entre desenvolvedores e a realização de alterações sem a introdução de novos erros. As características do código limpo incluem nomes de variáveis descritivos, estrutura organizada e a eliminação de redundâncias. Ao seguir esses princípios, as equipes promovem a criação de software mais robusto e sustentável, contribuindo para a **eficiência do processo de desenvolvimento** e para a satisfação dos stakeholders.

- **O que são boas práticas?**

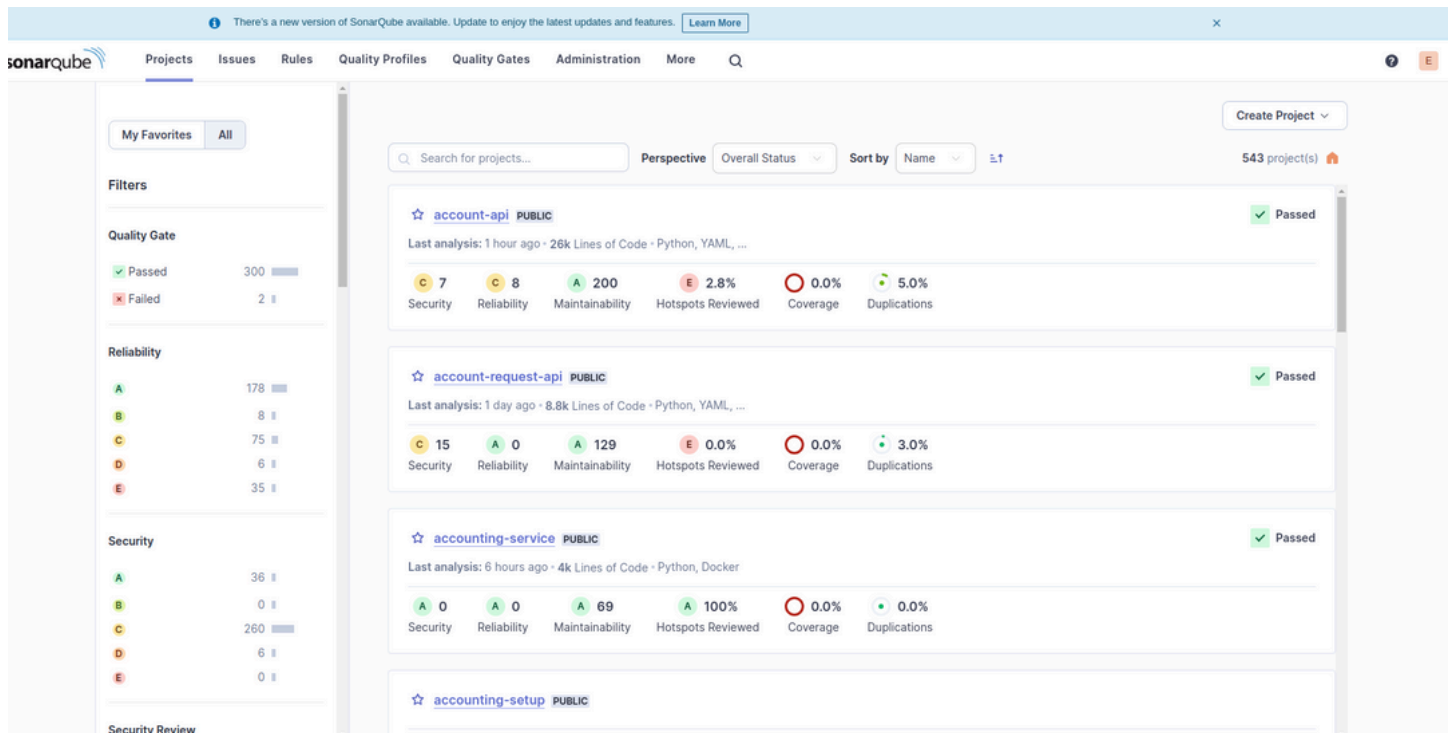
Boas práticas de desenvolvimento englobam um conjunto de princípios, diretrizes e padrões que visam **otimizar a criação de software de qualidade**. Essas práticas incluem o uso de testes automatizados, a adoção de **padrões** de arquitetura de código apropriados, a realização de **revisões** de código e a **documentação** adequada. A implementação dessas boas práticas não apenas melhora a qualidade do código, mas também facilita a comunicação e a colaboração entre os membros da equipe, reduzindo riscos e aumentando a previsibilidade no desenvolvimento de projetos. O SonarQube se torna um aliado valioso nesse contexto, uma vez que ajuda a monitorar e reforçar essas boas práticas, promovendo uma cultura de excelência e melhoria contínua no desenvolvimento de software.

Primeiros passos no SonarQube?

- **Login:**



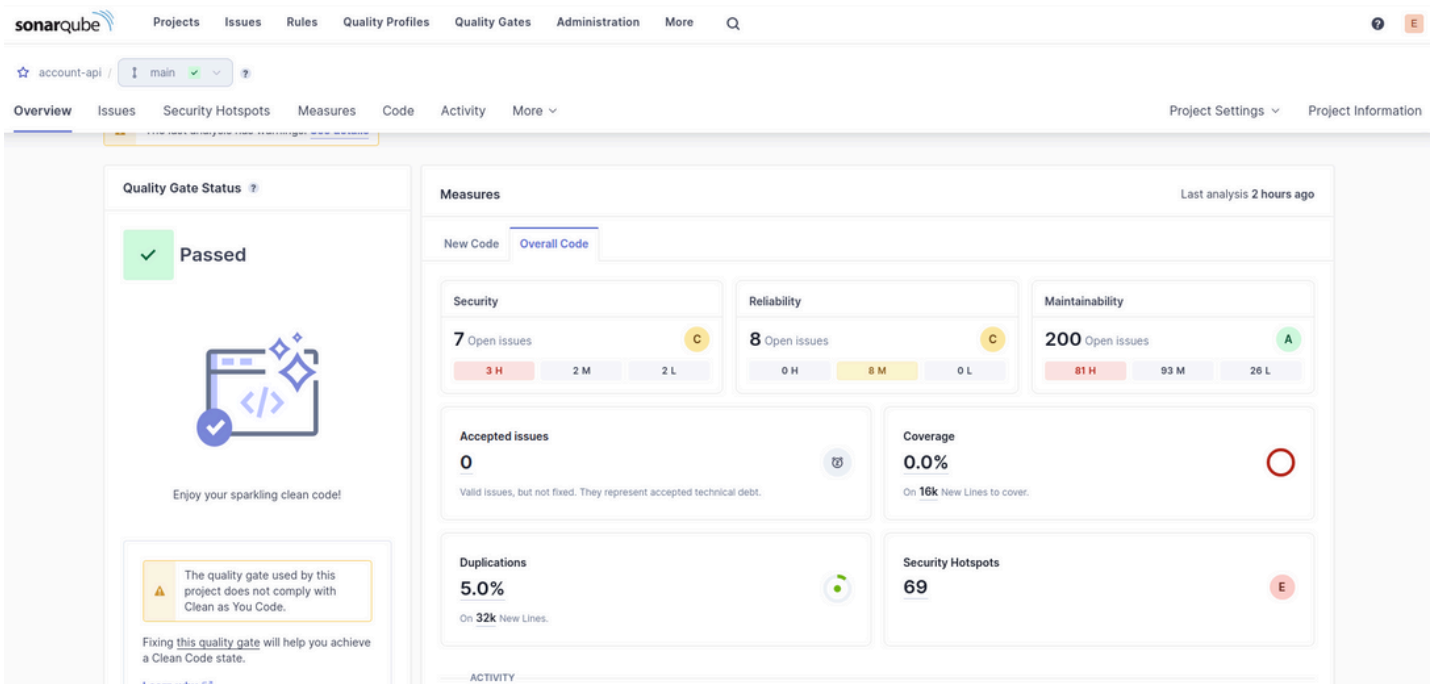
- **Tela inicial:**



A **tela inicial** do SonarQube apresenta uma visão consolidada dos **projetos do gitlab**, destacando informações essenciais como o status das análises e classificações de qualidade. No lado esquerdo, há filtros para visualizar projetos que passaram ou falharam no Quality Gate, além de métricas de confiabilidade e segurança. No painel central, cada projeto exibe detalhes como a última análise, linhas de código e notas em segurança, confiabilidade e manutenibilidade.

- **Projetos:**

Na seção de **projetos** do SonarQube, os desenvolvedores visualizam métricas essenciais como segurança, confiabilidade e manutenibilidade. Cada uma dessas métricas exibe problemas abertos categorizados em níveis de gravidade, auxiliando na priorização das correções.



- **Diferença entre New Code e Overall Code**

New Code

Overall Code

New Code

Código que foi adicionado ou modificado em um projeto.

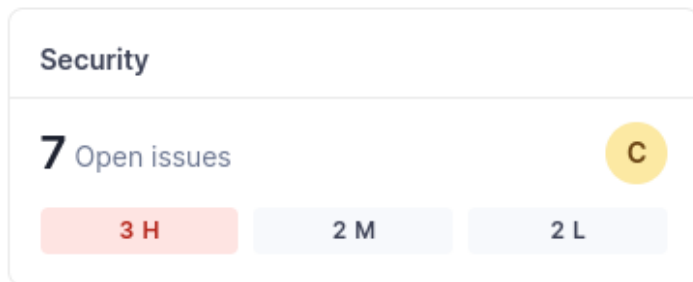
New Code

Overall Code

Overall Code

Abrange todo o código existente no repositório do projeto.

No SonarQube, as métricas de segurança, confiabilidade e manutenibilidade são fundamentais para avaliar a qualidade do código. Elas identificam vulnerabilidades, classificando os problemas como altos (H), médios (M) ou baixos (L), sendo H riscos críticos que precisam de correção imediata. As notas atribuídas podem variar entre A, B, C, D e E e variam para cada uma das áreas a seguir.



Security

A seção de segurança compara o código com erros de segurança comuns, como os listados no PCI DSS, OWASP Top 10 e CWE Top 25.



• Métricas - Security

- **A** = 0 vulnerabilidades
- **B** = pelo menos uma vulnerabilidade menor
- **C** = pelo menos uma vulnerabilidade maior
- **D** = pelo menos uma vulnerabilidade crítica
- **E** = pelo menos uma vulnerabilidade bloqueadora

Exemplo - Security



Filename: gunicorn:21.2.0 | Reference: CVE-2024-1135 | CVSS Score: 8.2 | Category: CWE-444 | Gunicorn fails to properly validate Transfer-Encoding headers, leading to HTTP Request Smuggling (HRS) vulnerabilities. By crafting requests with conflicting Transfer-Encoding headers, attackers can bypass security restrictions and access restricted endpoints. This issue is due to Gunicorn's handling of Transfer-Encoding headers, where it incorrectly processes requests with multiple, conflicting Transfer-Encoding headers, treating them as chunked regardless of the final encoding specified. This vulnerability allows for a range of attacks including cache poisoning, session manipulation, and data exposure. [↗](#)

Using Components with Known Vulnerabilities [OWASP:UsingComponentWithKnownVulnerability](#)

Introduced: 7 months ago • Vulnerability • Major

☐ Open ▾ Not assigned ▾ cwe ... +

Where is the issue? Why is this an issue? Activity

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

References:

- OWASP Top 10 2013-A9: [Using Components with Known Vulnerabilities](#)
- [Common Weakness Enumeration CWE-937](#)

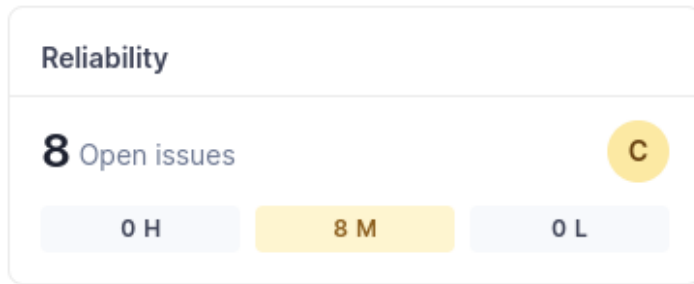
This issue was generated by [Dependency-Check](#)

Clean code attribute

Consistency | Not conventional

Software qualities impacted

Security ●



Reliability

A métrica de confiabilidade mede a probabilidade de o software funcionar sem falhas.



- **Métricas - Reliability**

- **A** = 0 Bugs
- **B** = pelo menos um Bug menor
- **C** = pelo menos um Bug maior
- **D** = pelo menos um Bug crítica
- **E** = pelo menos um Bug bloqueador

Exemplo - Reliability



Remove or correct this useless self-assignment. [python:S1656](#)

Variables should not be self-assigned [python:S1656](#)

Line affected: L141 • Effort: 3min • Introduced: 1 month ago • Bug • Major

Open

João Pedro Dias Nunes

No tags

Where is the issue?

Why is this an issue?

Activity

b3-operation-api > src/engines/assignment_cccb/registration_external_key_return_engine.py [Open in IDE](#) [See all issues in this file](#)

136 joao.n...

137

138

139

140

141

142 joao.n...

143

144

145 joao.n...

146

147

found_file_name = correspondences[0][0]

request_number = correspondences[0][1]

if filename == found_file_name:

request_number = request_number

self.logger.info(

f"File found. b3_return: {b3_return[0]}, Filename: {filename}"

)

break

except Exception as ex:

Uncovered code

Remove or correct this useless self-assignment.

Clean code attribute

Intentionality | Not logical

Software qualities impacted

Reliability

Remove or correct this useless self-assignment. [python:S1656](#)

Variables should not be self-assigned [python:S1656](#)

Line affected: L141 • Effort: 3min • Introduced: 1 month ago • Bug • Major

Open

João Pedro Dias Nunes

No tags

Where is the issue?

Why is this an issue?

Activity

There is no reason to re-assign a variable to itself. Either this statement is redundant and should be removed, or the re-assignment is a mistake and some other value or variable was intended for the assignment instead.

Noncompliant code example

name = name

Compliant solution

name = other.name

Clean code attribute

Intentionality | Not logical

Software qualities impacted

Reliability

Maintainability

200 Open issues

A

81 H

93 M

26 L

Maintainability

A seção de manutenibilidade avalia a facilidade com que o código pode ser alterado e melhorado.



• Métricas -Maintainability

- **A** = Se o custo de remediação outstanding for $\leq 5\%$ do tempo que já foi investido na aplicação.
- **B** = Se estiver entre 6% e 10%.
- **C** = Se estiver entre 11% e 20%.
- **D** = Se estiver entre 21% e 50%.
- **E** = Qualquer valor acima de 50%.

Exemplo - Maintainability

Remove this "raise" statement or move it inside an "except" block. [python:S5747](#)

Bare "raise" statements should only be used in "except" blocks [python:S5747](#)

Line affected: L1054 • Effort: 10min • Introduced: 1 month ago • Code Smell • Critical

Open ☐ Guilherme Marques ☐ error-handling ... +

Where is the issue? Why is this an issue? How can I fix it? Activity More info

account-api > src/modules/investment/retroactive_investment_module.py [Open in IDE](#) [See all issues in this file](#)

```
1049 guilhe... or current_available_yield_in.investment_movement_datetime.date()
1050                                     != available_yield_in_image.investment_movement_date
1051                                 ):
1052                                     continue
1053                                     if reference_date == start_day:
1054                                         raise
1055                                     else:
1056                                         self.__log_investment_movements(available_yield_in_image, current_available_yield_in)
1057                                         available_yield_in_image.original_investment_movement_id = current_available_yield_in.id
1058                                         self.__updating_original_in_from_image(available_yield_in_image, current_available_yield_in)
1059                                         break
1060
```

Uncovered code

Remove this "raise" statement or move it inside an "except" block.



Remove this "raise" statement or move it inside an "except" block. [python:S5747](#)

Bare "raise" statements should only be used in "except" blocks [python:S5747](#)

Line affected: L1054 • Effort: 10min • Introduced: 1 month ago • Code Smell • Critical

Open ☐ Guilherme Marques ☐ error-handling ... +

Where is the issue? Why is this an issue? How can I fix it? Activity More info

A bare `raise` statement, i.e. a `raise` with no exception provided, will re-raise the last active exception in the current scope:

```
def foo():
    try:
        ...
    except ValueError as e:
        raise # this will re-raise "e"
```

If the `raise` statement is not in an `except` or `finally` block, no exception is active and a `RuntimeError` is raised instead.

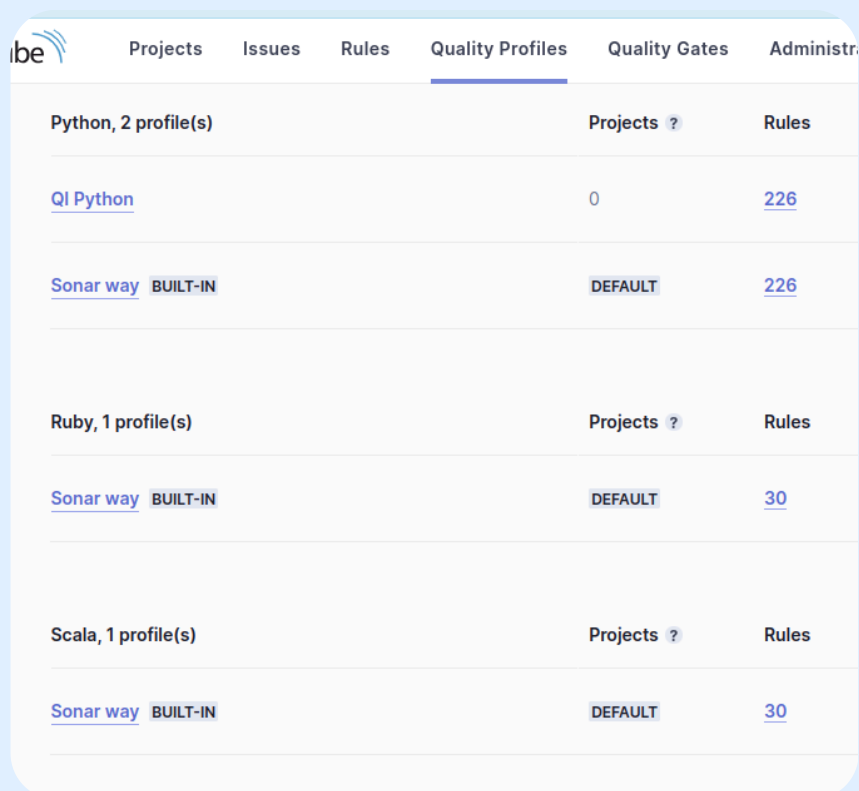

If the bare `raise` statement is in a function called in an `except` block, the exception caught by the `except` will be re-raised. However, this behavior is not reliable as nothing prevents a developer from calling the function from a different context.

Overall, having bare `raise` statements outside of `except` blocks is discouraged as it is hard to understand and maintain.

Notes

- **Quality Profiles:** Os Quality Profiles no SonarQube são coleções de regras padrão de qualidade agrupadas com base em critérios específicos, como linguagem de programação ou tipo de projeto, permitindo que as equipes personalize a análise de código de acordo com suas necessidades e padrões organizacionais. Esses perfis ajudam a garantir que todos os desenvolvedores sigam as mesmas diretrizes, promovendo consistência e melhorando a qualidade geral do código ao longo do tempo.

Exemplo - Quality Profiles

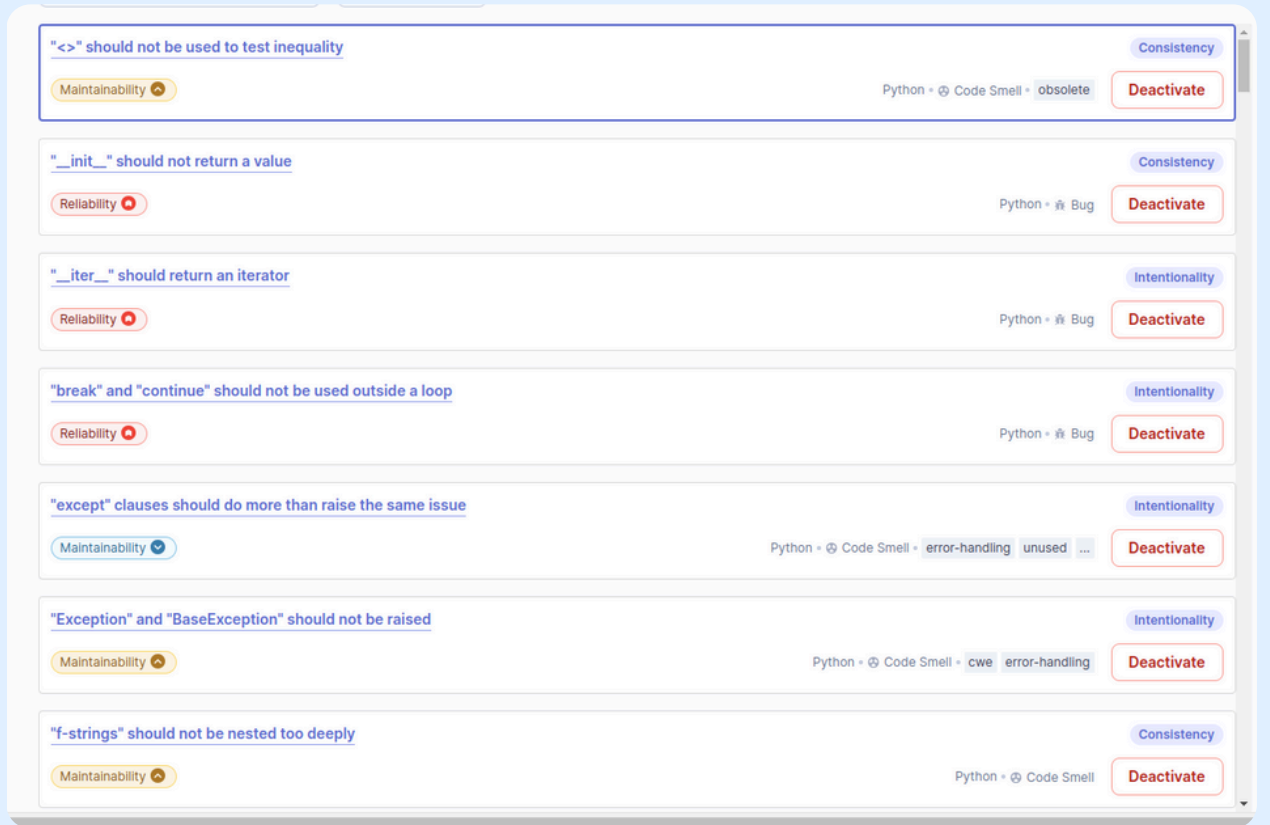


The screenshot shows the SonarQube interface with the 'Quality Profiles' tab selected. It displays a table of quality profiles for different languages. The table has columns for the language and profile name, the number of projects, and the number of rules. The profiles shown are for Python (2 profiles), Ruby (1 profile), and Scala (1 profile). Each language section lists the 'Sonar way' built-in profile as the default.

Python, 2 profile(s)		Projects ?	Rules
QI Python		0	226
Sonar way BUILT-IN		DEFAULT	226
Ruby, 1 profile(s)		Projects ?	Rules
Sonar way BUILT-IN		DEFAULT	30
Scala, 1 profile(s)		Projects ?	Rules
Sonar way BUILT-IN		DEFAULT	30

- **Rules:** As rules no SonarQube são diretrizes que definem os padrões de qualidade e as melhores práticas durante a análise do código, identificando problemas como bugs, vulnerabilidades e "code smells". Essas regras são cruciais para ajudar os desenvolvedores a detectarem e solucionarem falhas no código, assegurando que o software não apenas funcione, mas também mantenha altos padrões de qualidade e segurança.

Exemplo - rules



The screenshot displays the 'Custom Rules' configuration page in SonarQube. It features a list of seven rules, each with a title, a category tag, a severity indicator, a language/issue type path, and a 'Deactivate' button. The rules are as follows:

Rule Title	Category	Severity	Path	Action
"<>" should not be used to test inequality	Maintainability	Consistency	Python > Code Smell > obsolete	Deactivate
"__init__" should not return a value	Reliability	Consistency	Python > Bug	Deactivate
"__iter__" should return an iterator	Reliability	Intentionality	Python > Bug	Deactivate
"break" and "continue" should not be used outside a loop	Reliability	Intentionality	Python > Bug	Deactivate
"except" clauses should do more than raise the same issue	Maintainability	Intentionality	Python > Code Smell > error-handling > unused > ...	Deactivate
"Exception" and "BaseException" should not be raised	Maintainability	Intentionality	Python > Code Smell > cwe > error-handling	Deactivate
"f-strings" should not be nested too deeply	Maintainability	Consistency	Python > Code Smell	Deactivate

- **Custom Rules:** As custom rules no SonarQube permitem que as equipes de desenvolvimento criem regras personalizadas específicas para atender às particularidades de seus projetos e processos de desenvolvimento. Essa flexibilidade possibilita a inclusão de padrões internos ou requisitos específicos do setor, garantindo que a análise de código se alinhe de forma precisa com as necessidades da equipe, promovendo uma abordagem mais eficaz na manutenção da qualidade do software.

Criação de uma regra

→ Rules → Templates → show Templates Only → Create



Create Custom Rule

All fields marked with * are required

Name *

Key *

Category

Intentionality

Attribute

Complete

Software Quality *

☐ Security

☒ Reliability

☐ Maintainability

Severity *

None

High

None

Create Cancel

“

É possível criar uma regra utilizando código Java, no entanto, isso envolve uma complexidade considerável. Segue um exemplo de regra criada.

```

package org.sonarsource.plugins.example.rules;

import org.sonar.api.rule.RuleKey;
import org.sonar.api.rule.RuleStatus;
import org.sonar.api.rule.Severity;
import org.sonar.api.server.rule.RuleDescriptionSection;
import org.sonar.api.server.rule.RulesDefinition;

import static
    org.sonar.api.server.rule.RuleDescriptionSection.RuleDescriptionSectionKeys.HOW_TO_FIX_SECTION_KEY
    ;
import static
    org.sonar.api.server.rule.RuleDescriptionSection.RuleDescriptionSectionKeys.INTRODUCTION_SECTION_KEY
    ;
import static
    org.sonar.api.server.rule.RuleDescriptionSection.RuleDescriptionSectionKeys.ROOT_CAUSE_SECTION_KEY
    ;

public class JavaRulesDefinition implements RulesDefinition {

    public static final String REPOSITORY = "java-example";
    public static final String JAVA_LANGUAGE = "java";
    public static final RuleKey RULE_ON_LINE_1 = RuleKey.of(REPOSITORY,
"line1");

    @Override
    public void define(Context context) {
        NewRepository repository = context.createRepository
(REPOSITORY, JAVA_LANGUAGE).setName("My Custom Java Analyzer");

        var hibernate = new org.sonar.api.server.rule.Context("hibernate",
"Hibernate");
        var myBatis = new org.sonar.api.server.rule.Context("mybatis",
"MyBatis");

        NewRule x1Rule = repository.createRule(RULE_ON_LINE_1.rule())
            .setName("Stupid rule")
            .setHtmlDescription(
"Generates an issue on every line 1 of Java files")
            .addDescriptionSection(descriptionSection
(INTRODUCTION_SECTION_KEY, "This rule is not that stupid", null))
            .addDescriptionSection(descriptionSection(ROOT_CAUSE_SECTION_KEY,
"The root cause of this issue is this and that.", null))
            .addDescriptionSection(descriptionSection(HOW_TO_FIX_SECTION_KEY,
"To fix an issue reported by this rule when using Hibernate do this and
that."
, hibernate))
            .addDescriptionSection(descriptionSection(HOW_TO_FIX_SECTION_KEY,
"To fix an issue reported by this rule when using MyBatis do this and th
at."
, myBatis))
            // optional tags
            .setTags("style", "stupid")

            // optional status. Default value is READY.
            .setStatus(RuleStatus.BETA)

        // default severity when the rule is activated on a Quality profile. Def
        //ault value is MAJOR.
        .setSeverity(Severity.MINOR);

        x1Rule.setDebtRemediationFunction(x1Rule.debtRemediationFunctions().
linearWithOffset("1h", "30min"));

        // don't forget to call done() to finalize the definition
        repository.done();
    }

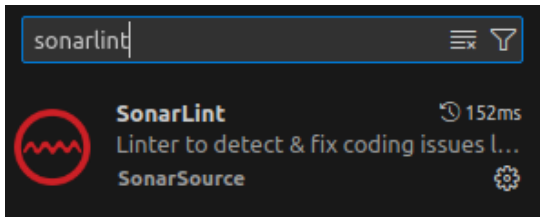
    private static RuleDescriptionSection descriptionSection(String
sectionKey, String htmlContent, org.sonar.api.server.rule.Context
context) {
        return RuleDescriptionSection.builder()
            .sectionKey(sectionKey)
            .htmlContent(htmlContent)
    }

    //Optional context - can be any framework or component for which you wan

```

```
    to create detailed description
    .context(context)
    .build();
  }
}
```

SonarLint



O **SonarLint** é uma ferramenta de análise de código que ajuda desenvolvedores a identificar e corrigir problemas de qualidade no código em [tempo real](#). Funciona como um plug-in para IDEs, incluindo **VS Code** e **intelliJ IDEA**, que são amplamente utilizados na empresa.

Basicamente essa ferramenta te mostra os erros que seriam mostrados pela interface do sonar de maneira interativa diretamente na IDE utilizada, facilitando a solução dos problemas.

Quality Gates

Um **Quality Gate** no SonarQube é um conjunto de condições que um projeto deve atender para ser considerado de boa qualidade. Essas condições avaliam diferentes aspectos do código, como a presença de bugs, vulnerabilidades, a cobertura de testes e a duplicação de código. O objetivo de um Quality Gate é garantir que o software atenda padrões específicos de qualidade antes de ser implantado.

Conditions on New Code		
Metric	Operator	Value
Issues	is greater than	0
Security Hotspots Reviewed	is less than	100%
Coverage	is less than	40.0%
Duplicated Lines (%)	is greater than	80.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Rating	is worse than	A

Excelente

Esse Quality Gate é o ideal e deve ser visado pelos projetos da Qitech. Para ser classificado como Excelente, o código deve ter:

- 0 Issues (nenhum problema aberto), Notas A ou melhores para manutenibilidade, confiabilidade e segurança.

Esse nível garante um código limpo, seguro e fácil de manter, refletindo nosso compromisso com a máxima qualidade.

Conditions on New Code		
Metric	Operator	Value
Issues	is greater than	15
Security Hotspots Reviewed	is less than	0.0%
Coverage	is less than	0.0%
Duplicated Lines (%)	is greater than	100%
Maintainability Rating	is worse than	C
Reliability Rating	is worse than	C
Security Rating	is worse than	C

Médio

Esse Quality Gate é um nível que ainda requer melhorias. Para ser classificado assim, o código deve ter no máximo:

- 15 Issues, Notas C ou melhores para manutenibilidade, confiabilidade e segurança.

Esse nível representa uma fase de transição, onde o código é mais vulnerável e permissível. Embora ainda apresente áreas para aprimoramento, é fundamental que a equipe trabalhe para garantir um código limpo, seguro e fácil de manter, refletindo nosso compromisso com a qualidade.

Conditions on New Code		
Metric	Operator	Value
Issues	is greater than	150
Security Hotspots Reviewed	is less than	0.0%
Coverage	is less than	0.0%
Duplicated Lines (%)	is greater than	100%
Maintainability Rating	is worse than	D
Reliability Rating	is worse than	D
Security Rating	is worse than	D

Baixo/Low

O Quality Gate QI - Baixo é o nível que quase não tem filtros para os projetos da Qitech.

- Mais de 150 Issues.

- Notas D ou piores para manutenibilidade, confiabilidade e segurança.

