Utilizando SonarQube para melhoria da qualidade do sistema Web EducAPI: Um Relato de Experiência

Drayon Richard Magno Duarte Silva[1]

Departamento de Ciências Exatas (DCX) –
Universidade Federal da Paraíba (UFPB) - Campus IV
Cep 58297-000 – Rio Tinto – PB – Brasil

drayon.richard@dcx.ufpb.br

Abstract. *Considering that there is a high demand for qualified professionals in the area of Information Technology and not so many qualified professionals available in the market, some companies have reinforced practices in training inexperienced people to fill the remaining vacancies in the area and the use of tools to seek to guarantee that the quality of their products can be guaranteed. Examples of these tools are the static analysis tools that help to identify and solve code structuring problems, duplication, common security problems, among others. This work aims to illustrate the use of an automatic code review tool, SonarQube, in an existing Web system (EducAPI) and analyze the main lessons learned in this process.*

Resumo. *Considerando que há uma alta demanda por profissionais qualificados na área de Tecnologia da Informação e pouca mão de obra qualificada disponível no mercado, algumas empresas têm reforçado práticas como o treinamento de pessoas inexperientes para suprir as vagas remanescentes da área e o uso de ferramentas para buscar garantir que a qualidade de seus produtos possa ser garantida. Exemplos dessas ferramentas são as ferramentas de análise estática que ajudam a identificar e resolver problemas de estruturação do código, duplicidade, problemas comuns de segurança, dentre outros. Este trabalho visa ilustrar a utilização de uma ferramenta automática de revisão de código, que é o SonarQube, em um sistema Web existente (o EducAPI) e analisar as principais lições aprendidas neste processo.*

1. Introdução

A pandemia da Covid-19 espalhou-se rapidamente, o que causou uma série de mudanças no mundo devido à adoção de medidas sanitárias de isolamento social. Com as pessoas passando mais tempo em casa, ocorreu uma aceleração da adesão ao mundo virtual e houve um grande crescimento no mercado de TI. Com a alta demanda nas empresas por mão de obra especializada, surgiram muitos empregos no mercado de tecnologia e faltam profissionais qualificados disponíveis. A Figura 1 ilustra o aumento dessa demanda. Conforme ressalta a [Brasscom[2] 2021]:

> Com apenas 53 mil pessoas formadas por ano em cursos de perfil tecnológico e uma demanda média anual de 159 mil profissionais de

[1] Trabalho de conclusão de curso, sob orientação da professora Ayla Débora Dantas de Souza Rebouças submetido ao Curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação (CCAE) da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de LICENCIADO EM CIÊNCIA DA COMPUTAÇÃO.

[2] Associação das Empresas de Tecnologia da Informação e Comunicação (TIC) e de Tecnologias Digitais

Tecnologia da Informação e Comunicação, o Brasil tem um grande desafio pela frente.

Figura 1: Demanda por novos profissionais de tecnologia 2021-2025

Fonte: Brasscom[3]

Com essa grande demanda por profissionais, as empresas têm reforçado práticas como contratar profissionais inexperientes com desejo de aprender e capacitá-los para preencher vagas em aberto. A falta de experiência vinda desses profissionais pode gerar uma queda na qualidade do código e consequentemente do software; o que pode tornar as entregas menos confiáveis, gerar baixa manutenibilidade do software e maior carga de trabalho. Uma das formas de evitar perdas de qualidade de software por adotar tal estratégia, é abordando a exploração de ferramentas que promovem a qualidade de código durante a etapa de desenvolvimento de software gerando uma maior qualidade do software, sendo uma delas o SonarQube[4]. Essas ferramentas trazem tarefas de qualidade e melhoria para o momento inicial de sua implementação. Como consequência, pode-se gerar software de maior qualidade, melhor manutenção e diminuir a carga de trabalho na fase de manutenção, uma vez que códigos de qualidade tendem a trazer tais resultados.

Considerando isso, este trabalho tem como objetivo geral identificar de que forma o SonarQube pode identificar melhorias em serviços Web existentes e relatar a experiência de utilizar essa ferramenta em um sistema Web, identificando os possíveis ganhos e desafios nesse processo. Para isso, será utilizado o sistema EducAPI, um sistema Web desenvolvido em Java com o intuito de apoiar o desenvolvimento de aplicativos e sistemas para alfabetização:

[3] Estudo da Brasscom, aponta demanda de 797 mil profissionais de tecnologia até 2025. Disponível em: https://brasscom.org.br/estudo-da-brasscom-aponta-demanda-de-797-mil-profissionais-de-tecnologia-ate-2025/. Acesso em: 26/06/2022

[4] SonarQube. Disponível em: https://www.sonarqube.org/. Acesso em: 17 de abril. de 2022.

> O sistema EducAPI é um sistema que gerencia uma base de dados multimídia colaborativa dinâmica, onde os usuários podem cadastrar e consultar desafios (ou palavras) agrupadas em contextos (ou temas)[...] A versão atual da API REST do EducAPI está disponível em https://github.com/a4s-ufpb/EducAPI e foi desenvolvida utilizando o framework Spring Boot e a linguagem de programação Java [Ludgério et al. 2021].

Inicialmente foram analisados artigos e livros sobre a área de melhoria da qualidade de software, com foco em ferramentas de análise estática de código fonte. Posteriormente, foi selecionada a ferramenta de análise estática a utilizar, que no caso foi o SonarQube, bem como o sistema onde ela seria explorada, que foi o EducAPI. Através do serviço de computação Google *cloud* em sua versão gratuita, foi criada e executada uma instância do Sistema Operacional Ubuntu/Linux onde foi preparado o ambiente para receber o SonarQube. A partir deste ponto foi feita a integração ao repositório git do EducAPI, a fim de identificar e documentar os passos necessários para que tal integração ocorra com êxito. Então, foram gerados com essa ferramenta os relatórios de análise de código com o intuito de identificar Bugs e possíveis vulnerabilidades que expõem o software a comportamentos não desejados, duplicações de código e outras sugestões de melhoria da estrutura do código. Tais relatórios foram então analisados, bem como todo o processo de uso do SonarQube no EducAPI para identificar e compartilhar neste documento lições aprendidas, conclusões e propostas de trabalhos futuros.

São planejados para este trabalho os seguintes objetivos específicos: i) Identificar os passos utilizados para preparar o ambiente SonarQube; ii) Identificar os passos para integrar o SonarQube ao repositório do sistema EducAPI; iii) Analisar correções apontadas pelo sonar relativas ao sistema EducAPI; iv) Relatar a experiência da utilização da ferramenta SonarQube para melhoria da qualidade de software com base em seu uso do EducAPI e as principais lições aprendidas.

As demais seções deste artigo estão organizadas conforme descrito a seguir: a Seção 2 é dedicada à apresentação de tecnologias e conceitos abordados neste trabalho para melhor compreendê-lo; a Seção 3 apresenta o sistema EducAPI; a Seção 4 descreve os passos para uso do SonarQube e a integração com EducAPI; a Seção 5 apresenta a análise da utilização do SonarQube; a Seção 6 apresenta as lições aprendidas durante a realização deste trabalho; e, por fim, a Seção 7 apresenta as conclusões e sugestões de trabalho futuros.

2. Qualidade de Software e Ferramentas Utilizadas

Nesta seção serão apresentados conceitos utilizados neste trabalho como: Qualidade de Software, Análise Estática de Código, Boas Práticas de Programação. Serão apresentados também alguns detalhes sobre ferramentas utilizadas neste trabalho: SonarQube, Lint, Google Cloud, repositório Git, onde foi aplicado o SonarQube para realização da análise estática.

2.1 Qualidade no Desenvolvimento de Software e Análise Estática

Na etapa de desenvolvimento de software, há diversos desafios a serem superados. Um deles é produzir um código limpo. O código limpo tornou-se uma cultura de desenvolvimento de software que traz técnicas que buscam facilitar a escrita e leitura de um código, uma vez que a produção de códigos limpos traz vantagens como por exemplo, um código de fácil entendimento e consequentemente de fácil manutenibilidade. Fowler (1999) destaca que

qualquer um consegue escrever códigos que um computador entenda, mas só bons programadores escrevem código que humanos podem entender.

Existem diversas ferramentas que ajudam o desenvolvedor a manter um código limpo e coerente. Entretanto, é crucial para o desenvolvedor ter um bom entendimento da codificação limpa e também de ferramentas de análise estática, pois estas permitem a verificação de boas práticas de codificação. Conforme destacado por Fowler (1999), "*aprender a criar códigos limpos é uma tarefa árdua e requer mais do que o simples conhecimento dos princípios e padrões.*". A não adoção de boas práticas de desenvolvimento pode se tornar uma grande "dor de cabeça" futuramente, tornando a manutenção do código desgastante até para quem o desenvolveu.

Essas boas práticas ajudam a construir um ambiente de desenvolvimento de software mais seguro onde se espera um processo de construção de software de qualidade. Embora sabendo que a qualidade de software envolve várias questões, este trabalho se concentrará em qualidade de código e em ferramentas utilizadas para este propósito. Segundo Fitzpatrick (1996), a qualidade de Software é o quanto um conjunto de funcionalidades desejáveis são incorporadas a um produto de forma a melhorar seu desempenho ao longo do tempo..

Uma das estratégias para garantir qualidade de código é a análise estática. A análise estática é a inspeção do software sem a necessidade de haver uma execução desse código, ou seja, uma inspeção estática do código, o que difere da dinâmica que necessita executar o código. A programação por pares[5], pode ser considerada uma análise estática [Gomes 2007], pois há uma revisão do código fonte sem o executar. Porém, a terminologia análise estática é usada para identificar ferramentas de software automatizadas para esta tarefa. Nesse processo, conforme destaca Medeiros (2017), podem ser utilizadas diversas técnicas para detectar partes do código problemáticas e que precisam ser melhoradas ou corrigidas. Existem quatro técnicas principais para isso: análise de fluxo de dados, análise de fluxo de controle, *taint analysis* e análise léxica [Owasp 2017 apud Medeiros 2017].

Para Gomes (2007) o uso de ferramentas de análise estática previne erros como: Bugs de segurança comuns, problemas no *design* e problemas em manter um único padrão de codificação.

2.2 Ferramentas Utilizadas

Aqui serão apresentadas as ferramentas de software utilizadas no presente trabalho. A principal delas é o SonarQube. A plataforma SonarQube foi escolhida para a geração da análise estática pois de acordo com Shelajev et al. (2014), entre as ferramentas analisadas, é a mais popular com 20% do uso. Esta ferramenta faz uma análise no código fonte com base nas métricas preestabelecidas gerando um relatório que aponta automaticamente bugs, vulnerabilidades, gargalos de desempenho, sugestões de refatoração de código e *code smells,* por exemplo. Pode-se aproveitar os benefícios do Sonar através da adoção junto com práticas de integração contínua (IC). Um princípio de IC é inspeção contínua, que inclui análise estática de código-fonte, entre outros tipos de avaliações, em cada alteração do software [Marcilio et al, 2019].

[5] Programação por pares, consiste na prática onde 2 programadores trabalham em um computador, enquanto 1 programador codifica o outro faz a revisão do código escrito. [LIMA 2013]

O SonarQube pode apoiar esse processo fazendo uma análise estática sempre que houver uma atualização no código fonte, assim mantendo o relatório atualizado de acordo com o código mais atual possível. Para isso, o Sonar precisa estar integrado a alguma ferramenta de controle de versão. Uma das ferramentas mais utilizadas para isso é a ferramenta Git[6] que é um sistema de controle de versão não centralizado para o controle de versões de qualquer tipo de arquivo.

Uma das primeiras ferramentas de análise estática que surgiram foi o Lint. Ela surgiu logo quando foi provado que a revisão de código era uma das peças-chave para manter produtividade e qualidade do desenvolvimento até a entrega do produto [Chelf; Chou, 2007]. Este trabalho aborda a ferramenta SonarLint[7], que é usada como plugin em IDEs que fazem a geração da análise estática mesmo que o código ainda não esteja integrado com algum repositório de controle de versão. Desta forma, há um feedback em tempo real do código fonte para o programador.

Por último, uma outra ferramenta também utilizada neste trabalho foi o Google *Cloud Computing,* que é uma infraestrutura de nuvem que oferece flexibilidade na configuração da máquina, escolhendo a quantidade de vCPUs, armazenamento, memória ou GPU. Também é possível configurar a rede e definir o sistema operacional. O *Cloud Computing* é responsável por gerenciar a instância do sistema operacional Ubuntu/Linux que por sua vez é responsável por hospedar o Sonar.

3. O Sistema EducAPI

O EducAPI foi escolhido por ser um serviço Web colaborativo *open source*, desenvolvido por egressos dos cursos de Licenciatura em Ciência da Computação e Bacharelado em Sistemas de Informação do campus IV da UFPB e por se tratar de um sistema que não passou por análise do SonarQube. O EducAPI é desenvolvido e mantido pelo o projeto de pesquisa Apps4Society[8], coordenado pela professora Ayla Débora Dantas de Souza Rebouças. É um sistema que tem como objetivo apoiar o processo de alfabetização utilizando recursos multimídia cadastrados de forma colaborativa. O sistema faz o gerenciamento de uma base de dados multimídia, onde os usuários desse sistema podem criar temas (contextos) e suas palavras associadas (desafios) associando-os a recursos multimídias (imagens, áudios e vídeos), e ou consultar a base de dados através de aplicativos ou softwares educacionais que vão consumir os dados. A Figura 2 ilustra alguns elementos do sistema.

[6] Git, Disponível: https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27git.27_name.3F. Acesso em: 17 de abril de 2022.
[7] SonarLint. Disponível: https://community.sonarsource.com/t/frequently-asked-questions/7204. Acesso em: 17 de abril. de 2022.
[8] Apps4Soeciety, Disponível em: https://apps4society.dcx.ufpb.br/. Acesso em 6 de junho de 2022.

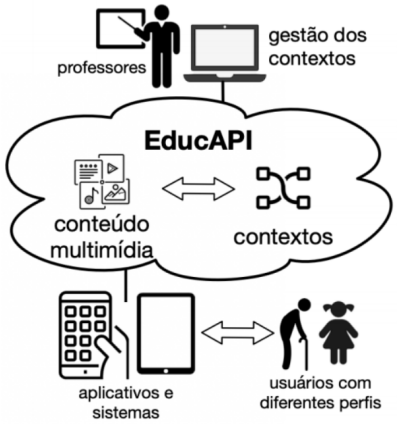


Figura 2: Ilustração EducAPI
Fonte: https://apps4society.dcx.ufpb.br/?page_id=942

O sistema do EducAPI é desenvolvido na linguagem Java utilizando o *framework* Spring[9], e os serviços do sistema como cadastro de usuário, login, cadastro de contextos e seus desafios estão disponíveis através de uma API REST [Masse 2011]. Seu código pode ser acessado no repositório disponível em: https://github.com/a4s-ufpb/EducAPI.

4. Processo de Utilização do SonarQube

O SonarQube é uma plataforma web de gerenciamento de qualidade contínua de código. Segundo Baldassarre (2020):

> O SonarQube verifica a conformidade do código em relação a um conjunto de regras de codificação (ou seja, problemas técnicos). Se o código violar alguma das regras classificadas, o SonarQube o considera uma violação ou uma dívida técnica. Além disso, define dívida técnica como o tempo necessário (ou seja, tempo de remediação) para corrigir as regras violadas.

Durante a etapa de desenvolvimento de software, a execução de ferramentas de análise estática nas IDE 's se faz necessária, pois os desenvolvedores enfrentam diversos desafios e há situações em que o desenvolvedor necessita quebrar algumas das boas práticas ("leis") como por exemplo o DRY (*Don't Repeat Yourself*). Segundo Hunt & Thomas (1999), códigos não devem ser duplicados, ou seja, um código deve ser declarado apenas uma vez, evitando assim que quando houver necessidade de manutenção no código, não seja necessário modificar outras partes do sistema, evitando bugs e retrabalho desnecessário. Mesmo sabendo o que deve fazer, sem executar a análise estática, podem ser gerados diversos cenários onde é gerado um código com problema ou que poderia ser melhor, seguindo algumas boas práticas. Quando se faz a análise estática, pode-se identificar *Code Smells*, que segundo Fowler são códigos que não foram escritos da melhor maneira e indicam algum problema na estrutura do código fonte e que pode ser facilmente corrigido com refatoração. Refatoração, por sua vez, é definida como uma melhoria de código sem alteração da sua funcionalidade.

O uso de análise estática é muito comum em projetos que utilizam processos baseados em metodologias ágeis com integração contínua. Nesses projetos são utilizados, geralmente, repositórios para controle de versões de códigos, e que são construídos e acessados por diferentes desenvolvedores. Conforme destacado por Meyer (2014) "*A integração contínua é mais do que um conjunto de práticas, é uma mentalidade que tem uma coisa em mente:*

[9] Spring, Disponível em: https://spring.io/. Acesso em 06/06/2022.

aumentar o valor do cliente." A exploração da inspeção de código automática garante que princípios de boas práticas de programação sejam seguidos, ajudando a reconhecer bugs mais frequentemente e vulnerabilidades.

O SonarQube é uma ferramenta que permite a inspeção de código automática do software. Neste trabalho, esta ferramenta será utilizada para inspeção do código do EducAPI, detalhando o passo a passo para a configuração da integração do Repositório com a plataforma e o processo de geração e análise dos relatórios.

O SonarQube foi implantado dentro de um servidor *Cloud* do Google em sua versão *Community Edition 9.4* e configurado junto a uma instância de banco de dados SQL, estando ambos rodando no mesmo servidor. Após o SonarQube estar devidamente configurado, iniciou-se o serviço, sendo possível acessar a *Interface Web* do usuário por meio do navegador no endereço: http://35.199.79.5:9000/.

No primeiro acesso ao SonarQube, se faz a exigência da troca de senha do usuário administrador. Depois disso é possível integrar um repositório com o SonarQube.

Para este trabalho foi criada uma cópia do repositório Git do EducAPI por meio da operação "fork". Esta cópia estava hospedada na plataforma GitHub. Por meio do "fork" se faz uma cópia completa de um repositório no seu estado corrente e faz-se uma cópia não só dos arquivos de código, mas de toda a ramificação git até o momento para um novo repositório distinto, que no caso foi o seguinte: https://github.com/drayonrichard/EducAPI. O GitHub é uma das plataformas que possui uma configuração nativa com o SonarQube, tornando mais fácil a configuração da integração entre ambos. Além disso, o SonarQube possui uma documentação robusta dentro da própria instância da plataforma, permitindo ao usuário explorar o SonarQube confortavelmente. Seguindo a documentação do Sonar, é possível configurar um servidor rapidamente. Porém, foi sentida uma certa dificuldade na hora da integração entre o SonarQube e o GitHub, conforme detalhado adiante.

4.1 Passos para configuração do SonarQube com o EducAPI

O detalhamento da configuração inicial do SonarQube é de extrema importância para evitar alguns problemas que poderiam ser gerados durante a implantação da plataforma. Inicialmente foram levantados alguns pré-requisitos para a implementação do SonarQube conforme a sua documentação. Alguns dos requisitos identificados foram: i) Ter uma máquina com no mínimo 2GB disponíveis somente para o Sonar; ii) Ter espaço em disco suficiente para análises, onde o valor suficiente depende da quantidade de código a ser analisada; iii) Usar um sistema baseado em 64 bits; iv) Ter a versão Java 11 ou 17 instalada na máquina; v) Ter um dos seguintes bancos de dados com suas respectivas versões postgreSQL v9.6 à v13, MSSQL v12 à v15, Oracle v19C, v18C, v12C, vXE Editions. vi) ter um navegador para acessar a interface web.

Com os pré-requisitos em mãos, foi reservada uma máquina virtual com 1vCPU, 4 GB de RAM e 32GB de ROM rodando o sistema operacional Ubuntu 18.04 bionic na plataforma do Google Cloud.

Uma vez que se obteve Hardware compatível para implementação do SonarQube, passou-se para a segunda etapa, a de configuração da infraestrutura do sistema a ser analisado. Com a máquina virtual inicializada, inicialmente foi necessário rodar alguns comandos para configuração do ambiente:

- Entrar no modo de Super Usuário;

```
sudo su -
```

- Atualizar o sistema operacional;

```
apt-get update
```

- Instalar o Java 11 (na versão ubuntu utilizada o JDK default é o Java 11);

```
apt-get install unzip software-properties-common wget default-jdk
```

- Instalar o postgreSQL em sua versão com funcionalidades estendidas por ter maior familiaridade com essa versão;

```
apt-get install postgresql postgresql-contrib
```

- Acessar o postgres como superusuário;

```
su - postgres
```

- Acessar o shell do postgres;

```
psql
```

- Criar um usuário para o postgres;

```
CREATE USER sonarqube WITH PASSWORD 'Dray0n-S0n@r001';
```

- Criar um banco de dados e configurar o usuário criado acima como dono;

```
CREATE DATABASE sonarqube OWNER sonarqube;
```

- Dar os privilégios do banco ao usuário;

```
GRANT ALL PRIVILEGES ON DATABASE sonarqube TO sonarqube;
```

- É necessário sair do shell do postgres e reiniciar a máquina

```
\q

sudo reboot
```

Agora com o ambiente devidamente configurado para implementação do SonarQube, foi necessário baixar e configurar o mesmo, utilizando os seguintes comandos:

- Entrar no modo de Super Usuário;

```
sudo su -
```

- Criar uma pasta dentro de /downloads com “-p” que garante a criação sem erros

```
mkdir /downloads/sonarqube -p
```

- Acessar a pasta recém criada

```
cd /downloads/sonarqube
```

- Realizar o download do SonarQube versão 9.4 compactado em zip

```
wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.4.0.54424.zip
```

- Descompactar o SonarQube

```
unzip sonarqube-9.4.0.54424.zip
```

- Mover a pasta descompactada para diretório "/opt/sonarqube"

```
mv sonarqube-9.4.0.54424 /opt/sonarqube
```

- Criar um usuário sonarqube que será responsável por gerenciar o SonarQube e mudar o dono dos arquivos em "/opt/sonarqube" para o usuário "sonarqube" pertencente ao grupo “sonarqube”

```
adduser --system --no-create-home --group --disabled-login sonarqube

chown -R sonarqube:sonarqube /opt/sonarqube
```

Agora a etapa de instalação foi concluída e resta configurar o SonarQube. Vale ressaltar que a configuração pode variar de acordo com a necessidade. As configurações usadas neste trabalho foram as seguintes:

- Através do editor de texto “vi” foi acessado o arquivo sonar.sh e modificada a propriedade RUN_AS_USER do arquivo para o usuário "sonarqube". Depois disso, foram salvas as alterações no arquivo.

```
mv sonarqube-9.4.0.54424 /opt/sonarqube

RUN_AS_USER=sonarqube
```

- Continuando com o “vi” foi acessado o arquivo sonar.properties, adicionando e salvando algumas propriedades no arquivo, conforme ilustrado a seguir. As propriedades jdbc definem as credenciais do banco de dados e as propriedades web estão ligadas ao servidor web Sonar.

```
vi /opt/sonarqube/conf/sonar.properties

sonar.jdbc.username=sonarqube
sonar.jdbc.password=Dray0n-S0n@r001
sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube
sonar.web.javaAdditionalOpts=-server
sonar.web.host=0.0.0.0
```

- Ainda com o "vi" foi também acessado o arquivo 99-sonarqube.conf, para definição das propriedades que delimitam o número de arquivos e threads que o usuário executando o sonar está liberado para abrir.

```
vi /etc/security/limits.d/99-sonarqube.conf


sonarqube   -   nofile   65536
sonarqube   -   nproc    4096
```

- Por último, ainda pelo "vi", foi editado o arquivo sysctl.conf, adicionando as propriedades a seguir. O "vm.max_map_count" define a quantidade máxima de memória virtual e "fs.file-max" define a quantidade máxima de arquivos abertos.

```
vi /etc/sysctl.conf


vm.max_map_count=262144
fs.file-max=65536
```

- Após salvar os arquivos editados, é necessário reiniciar a máquina

```
sudo reboot
```

Depois destes passos, o SonarQube e seu ambiente de execução foram devidamente configurados. É necessário também, porém, iniciar o serviço do SonarQube. Para isso, basta seguir os próximos passos:

- Iniciar serviço do Sonar

```
/opt/sonarqube/bin/linux-x86-64/sonar.sh start
```

- Acompanhar o log de execução do Sonar

```
tail -f /opt/sonarqube/logs/sonar.log
```

Depois de configurar e executar o serviço do sonar, é possível acessar sua interface web através de um navegador na porta 9000 da máquina. A tela de login é carregada e as

credenciais requeridas no primeiro acesso por padrão são o valor **admin** para o Login e para o Password, conforme ilustrado pela Figura 3.

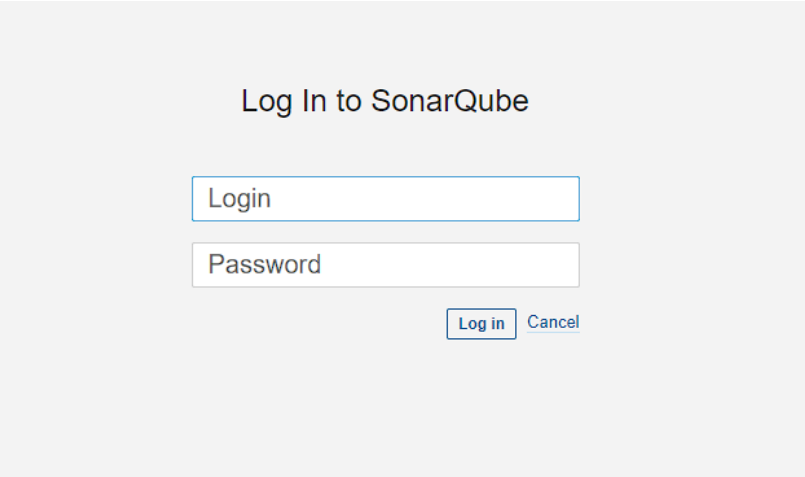


Figura 3: Login SonarQube
Fonte: Autor

Após o login no primeiro acesso, é requerido ao usuário administrador que mude a senha padrão para fins de segurança, conforme ilustrado pela Figura 4. Logo após a troca da senha, o usuário é redirecionado para a tela de criação dos projetos, ilustrada pela Figura 5.

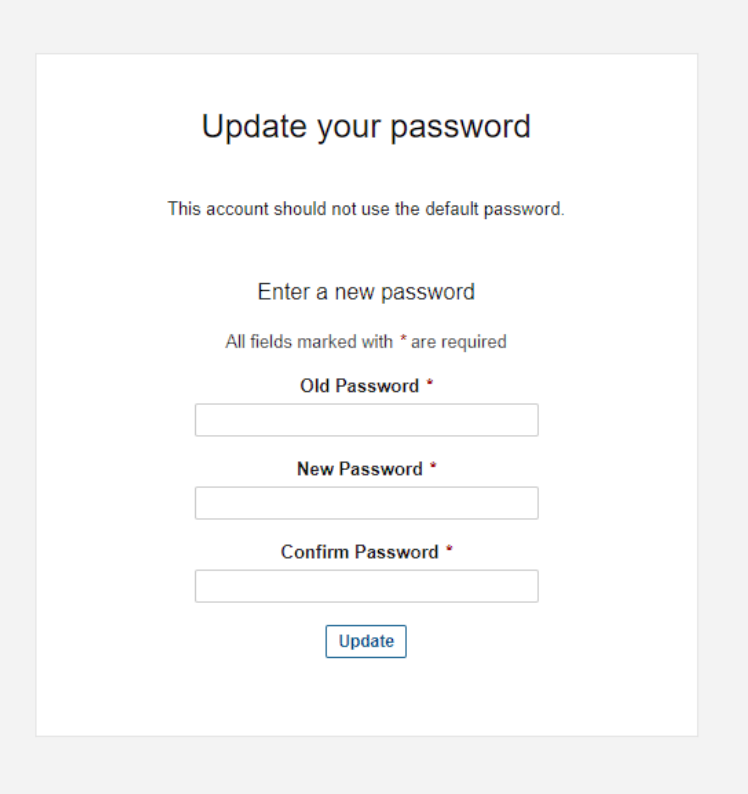


Figura 4: Atualização de senha do SonarQube
Fonte: Autor

Na tela de criação de projetos, é necessário configurar uma plataforma (como Azure DevOps, GitHub, GitLab) ou realizar uma configuração manual para poder criar a integração entre o projeto a ser analisado e o SonarQube.

Neste trabalho foi utilizado o GitHub para a integração. Logo, ao clicar na imagem "From GitHub" será aberta uma janela *pop-up* de formulário para configurar a plataforma GitHub. O formulário é bem descritivo quanto às informações necessárias para configuração, conforme ilustrado pela Figura 5.

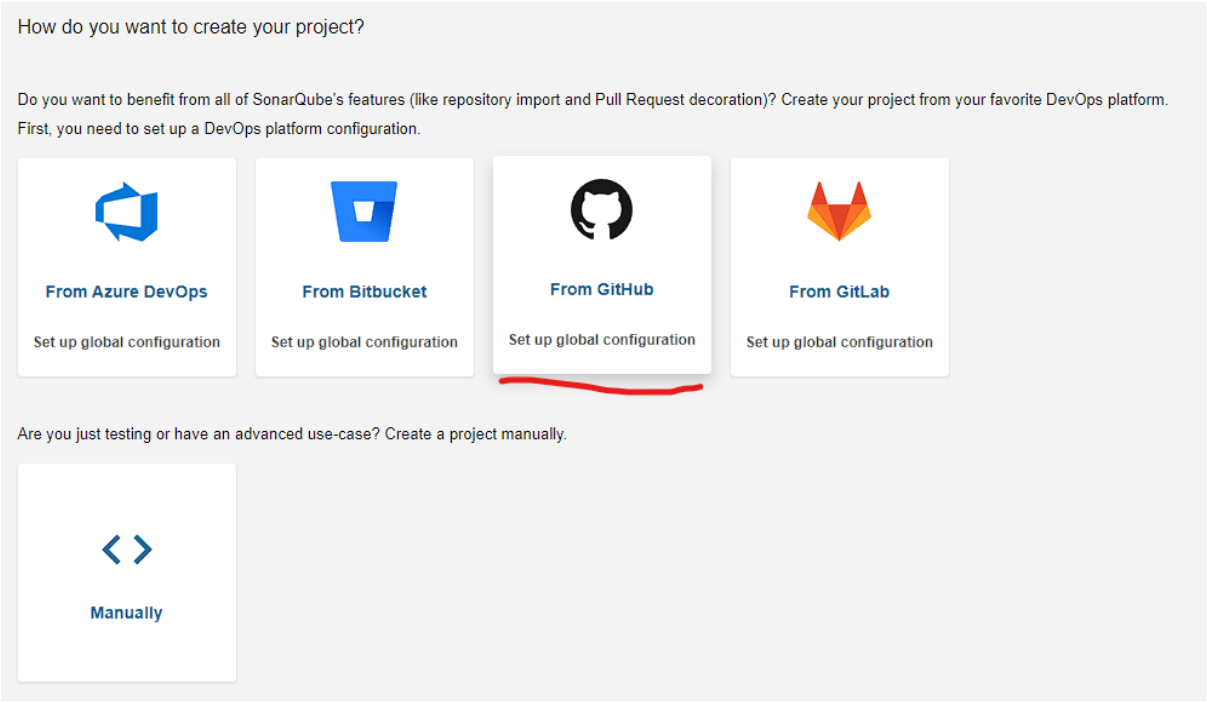


Figura 5: Integração do SonarQube
Fonte: Autor

Para o preenchimento dessas informações é necessário ir até a plataforma do GitHub, como destacado na Figura 6.

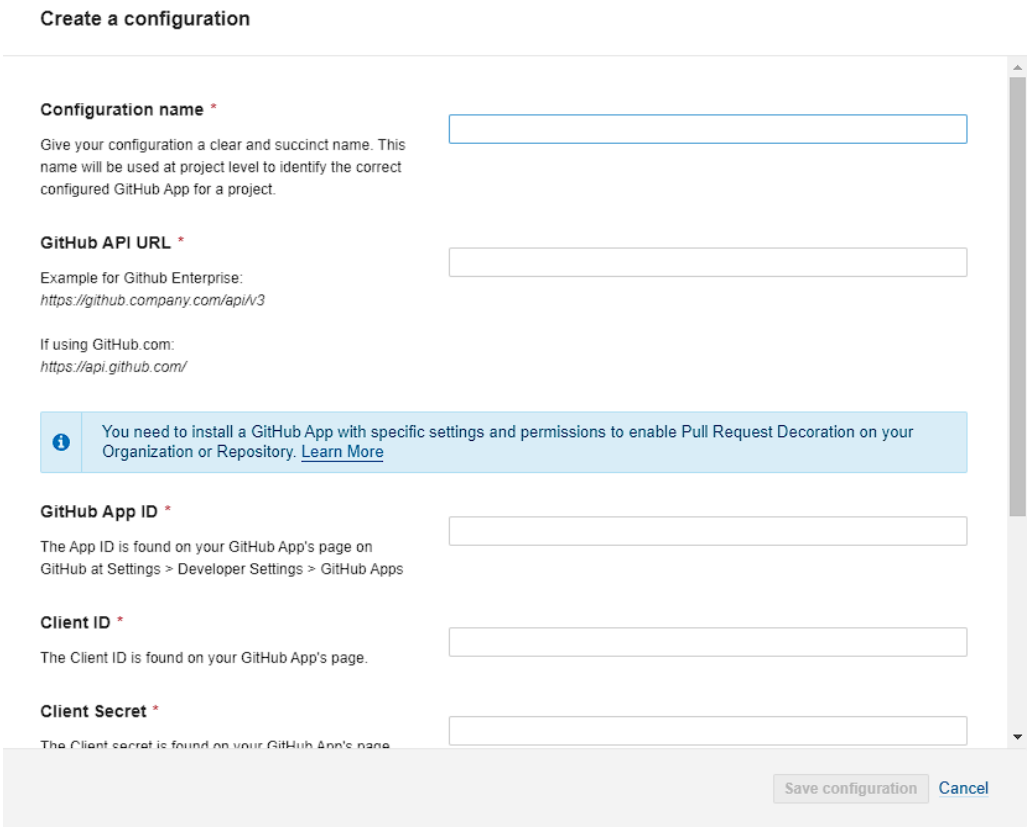


Figura 6: Configuração do GitHub no SonarQube
Fonte: Autor

Será possível ver que o SonarQube possui uma documentação para explicar como gerar ou pegar essas informações no GitHub. A seguir é descrito o passo a passo do preenchimento das informações:

- Na plataforma GitHub, depois de já feito o login, vá até as configurações da conta ou da sua organização;
- Na barra lateral da esquerda (*sidebar* à esquerda) clique em *Developer settings* (ver Figura 7)*;*

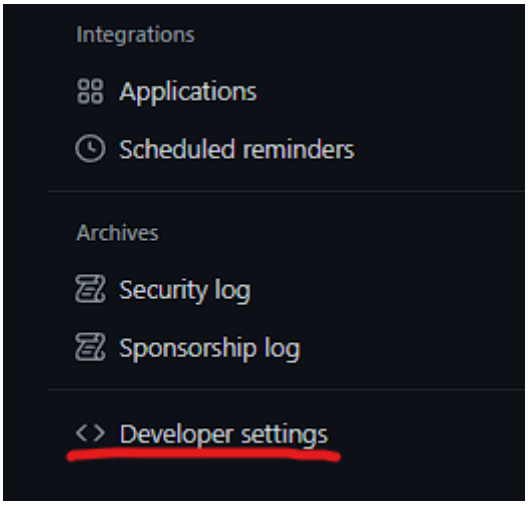


Figura 7: Barra lateral do GitHub

Fonte: Autor

- Agora vá até ***Github Apps*** e clique em ***New Github App*** (ver Figura 8)

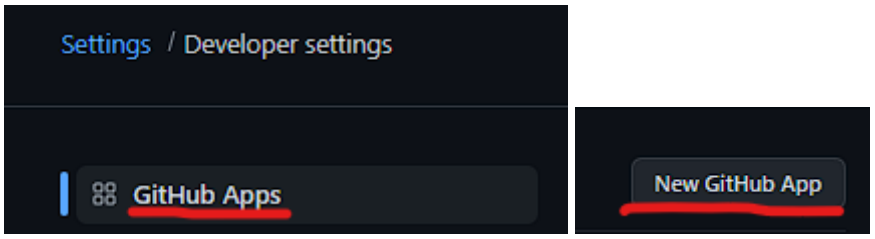


Figura 8: Criação de um novo *GitHub App*
Fonte: Autor

- Você será redirecionado para a página de criação do GitHub App, onde poderá fazer uma configuração mínima para criação do app. Neste trabalho foram utilizadas as seguintes configurações:
    - GitHub App name: **sonarIntegrationTesting**
    - Homepage URL: **https://www.sonarqube.org/**
    - Callback URL: **http://35.199.79.5:9000/projects/**
    - Expire user authorization tokens: **marcado**
    - Webhook: **Desativado**
    - Repository permissions:

Permissão	Nível de acesso
Checks	Read & write
Metadata	Read-only
Pull Requests	Read & write
Commit statuses	Read-only

    - Where can this Github App be installed?: **Only on this account**
- Com o app criado, é necessário gerar o *client secrets* (segredo do cliente) e a *private key* (chave privada) (ver Figura 9 e Figura 10).

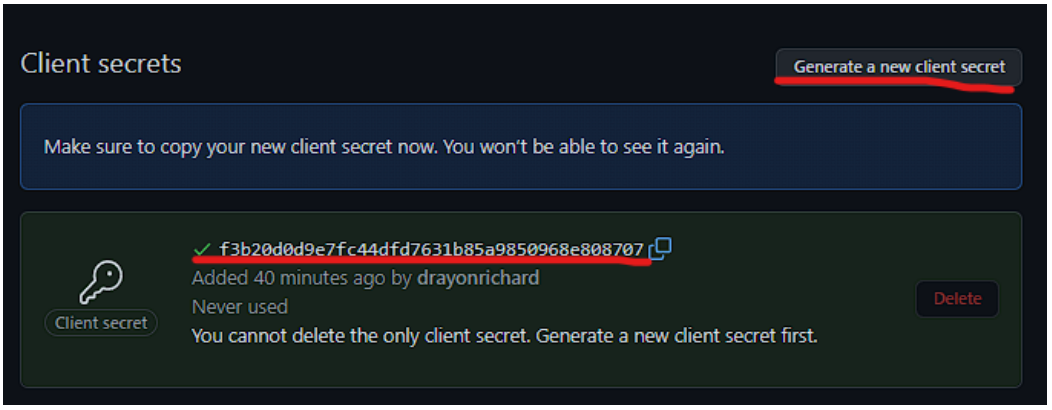


Figura 9: Segredo do cliente do *GitHub App*
Fonte: Autor

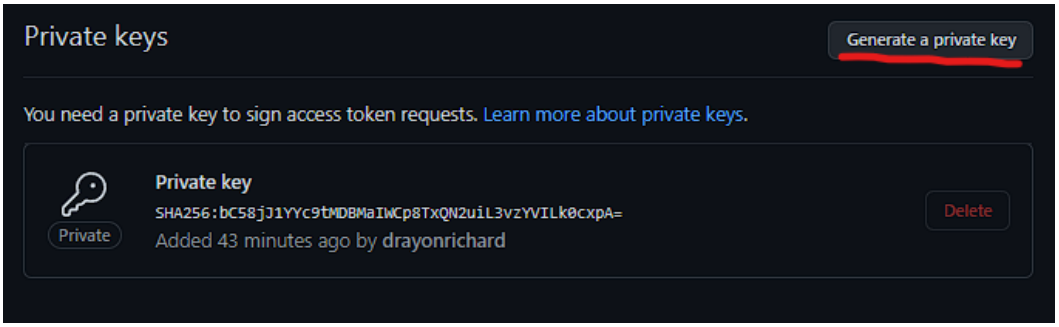


Figura 10: Chave privada do *GitHub App*
Fonte: Autor

- Será feito o download de arquivo .pem contendo a chave privada. Após abrir o arquivo, você poderá ver a chave, conforme ilustrado pelas Figuras 11 e 12.

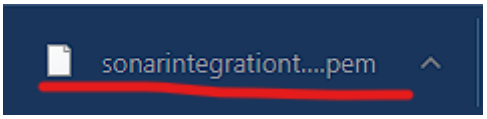


Figura 11: Download de arquivo
Fonte: Autor

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAt6JRmTQ8FsVbBky73p/F+cfDNrnu5So+ew22EYvjqcQQOynv
9zyynQbbHB9MV9I2Wf8/nmrCsiJJ0xW3RX2H6Hwc3YuKxD0eDBaO81F7JfkLGW6/
dxJLwREckMIXUa7PzEbqAT7/45LlNbuAB+lmH6XWPTpXicCXjS4/IfGy4QIXClBp
74FMMrTPig15ILsf8ma+fcfahTAupaiVOukKxYMDy9+uJ2kzR5fYmsd1n8Xz5CA7
aUZLs9CR0d4NsINvmpYm4jO1k70+Jd5B14HNqqLiUvVggoOOAYDi3W05Fbd4t15K
XOkqwXoIaZhl8EXXGl/6QCGs2Eoa6OT0uq+i6QIDAQABAoIBAHg96BJaWrJKXjT8
z9ORaRAKVvAhxaT3CVyH+Nz6bYN/J2WmK0QH0ajykUyTzeh/dClv8pdnrr73psF6
Kbxt2y1G/5utwJqIroEZ0FbQ1sNhlHv9wY9VQpdT/hFnld9diSFomddhCitYx49s
jAQiN0Wn0GXdKu6/Vj0SuDIg0pF0MFmok41ug4zGgOE2KDswHAHWg7ZLuzHPDXM0
EgxVWjmtjD5RpC0+ZhYgxZDzHdJt5kKg8jiWQsZmRmyHuT1EJnhVEPrdG7tvqMUy
PcetUK41H4i/60nw0kh3ULoHPzgr8f8URzoJfGWO9Ky6k7P5psH7q8GEhuXhXbn3
A7ulFwECgYEA8QwcwScG/dJ46/U+PTaRBBm+xbK8vpKcfoqL2p7LCEP7MvVOr7Bo
JUmvP7DyrXulqJdFSWWzNGGOSlNehWAGCPloYwFIuRxhZRRV0/gxmOYrrE80AhlR
SpPI1zD9tnNUl+uJnngzzJdlmuJfgO17wmG9qVJnfbZuy6pIJZmVuuECgYEAwwZ4
cmgxeQcmYp0T2mxqGaOtynvP6n1n6T4UR7wQ/Vbxmn4iNevfN05fdPj6yKUaafjS
u1AFNEAWZ1Ve6IDNddO0L2oGGJo/vjHLaqU7aHY8CME2enrOVsXvHM365mlogKxL
kC52XnNRHTQvpr689tqWYvYc+KV5JIXzQf+HMQkCgYEA0+JZxE+SDbvdH0zMEPLO
NlSLJS0gTESD25fdp6a9wd5ACWMob7cZVO2YZtwb9fMigze1Z/dFt12pvhvaklL9
+xWBdMXELzOLSVa+uz0IG+b3rNdSWCX5Fx0hBkfCPYbTbVdV+T6gCc9rOOKUILig
eRWBqj0NS6v5aH2ePa7rtwECgYAWZhzZTvw3oUmYU+Ae2wrIjRL9yYCUtviDQl0D
7CarYhv3mXsGZGPLr1WpUptDBitldm2Fk+g1GxM3pQd0fq6aVMlCFNREaRrfFkr9
oaKUGdhFTSObp5UhRDR1Q3cIlZyqAp/fHGb97BKp3v+GECwV0f8puIiVSjQCrNvG
fbmiOQKBgAKh3/YpCh/ylbjvbTOAZ0LexR7Zu9k7YOYxTXZffAOxNb+WcsAtikbI
zho1Lle3K7MVIEZ3BGnA9pK5OSOFHnd4711n6aBuR7zE7DMx1eeJFwRPRKhRSWzL
FIkCKNKujI26PubeJjwtNRRtGyGqaBWtnv/ZW8n4/u/WnFUp1kDo
-----END RSA PRIVATE KEY-----
```


Figura 12: Conteúdo do arquivo
Fonte: Autor

- Para que o SonarQube possa ler o repositório desejado, é necessária a instalação do ***GitHub App*** recém criado e escolher qual nível de acesso aquele App terá sobre seus repositórios, conforme ilustrado pelas Figuras 13 e 14.

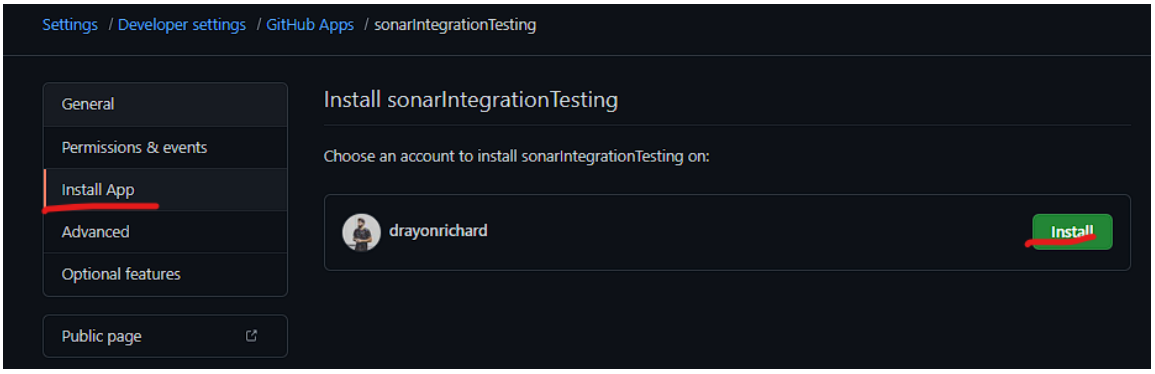


Figura 13: Página de instalação no GitHub
Fonte: Autor

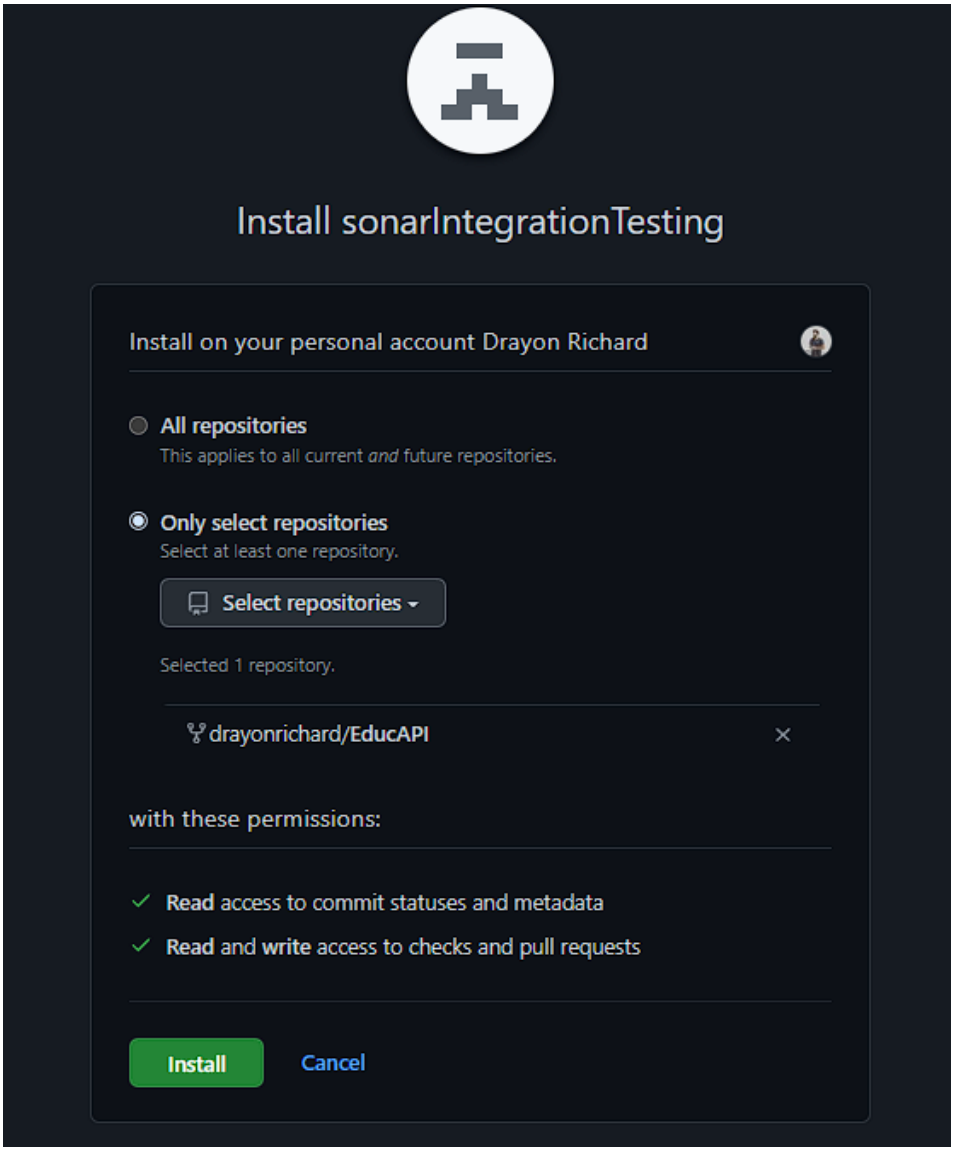


Figura 14: Instalação do *GitHub App* no GitHub
Fonte: Autor

- Depois de realizar os passos anteriores, será possível realizar a integração do SonarQube com a plataforma GitHub. Para isso, volte ao SonarQube e preencha as informações do formulário de criação visto anteriormente (Figura 6):
    - Configuration name: **testandoCI**
    - Github API URL: **https://api.github.com/**
    - Github App ID: **207409**
    - Client ID: **Iv1.4d9e55b8268e0be4**
    - Client Secret: **f3b20d0d9e7fc44dfd7631b85a9850968e808707**
    - Private Key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAt6JRmTQ8FsVbBky73p/F+cfDNrnu5So+ew22EYvjqcQQOynv
9zyynQbbHB9MV9I2Wf8/nmrCsiJJ0xW3RX2H6Hwc3YuKxD0eDBaO81F7JfkLGW6/
dxJLwREckMIXUa7PzEbqAT7/45LlNbuAB+lmH6XWPTpXicCXjS4/IfGy4QIXClBp
74FMMrTPig15Ilsf8ma+fcfahTAupaiVOukKxYMDy9+uJ2kzR5fYmsd1n8Xz5CA7
aUZLs9CR0d4NsINvmpYm4jO1k70+Jd5B14HNqqLiUvVggoOOAYDi3W05Fbd4t15K
XokqwXoIaZhl8EXXGl/6QCGs2Eoa6OT0uq+i6QIDAQABAoIBAHg96BJaWrJKXjT8
z9ORaRAKVvAhxaT3CVyH+Nz6bYN/J2WmK0QH0ajykUyTzeh/dClv8pdnrr73psF6
Kbxt2y1G/5utwJqIroEZ0FbQ1sNhlHv9wY9VQpdT/hFnld9diSFomddhCitYx49s
jAQiN0Wn0GxdKu6/Vj0SuDIg0pF0Mfmok41ug4zGgOE2KDswHAHWg7ZLuzHPDXM0
EgxVWjmtjD5RpC0+ZhYgxZDzHdJt5kKg8jiWQsZmRmyHuT1EJnhVEPrdG7tvqMUy
PcetUK41H4i/60nw0kh3UloHPzgr8f8URzoJfGWO9Ky6k7P5psH7q8GehuXhXbn3
A7ulFwECgYEA8QwcwScG/dJ46/U+PTaRBBm+xbK8vpKcfoqL2p7LCEP7MvVOr7Bo
JumvP7DyrXulqJdFSWWzNGGOSlNehWAGCPloYwFIuRxhZRRV0/gxmOYrrE80AhlR
SpPI1zD9tnNUl+uJnngzzJdlmuJfgO17wmG9qVJnfbZuy6pIJZmVuuECgYEAwwZ4
cmgxeQcmYp0T2mxqGaOtynvP6n1n6T4UR7wQ/Vbxmn4iNevfN05fdPj6yKUaafjS
u1AFNEAWZ1Ve6IDNddO0L2oGGJo/vjHLaqU7aHY8CME2enrOVsXvHM365mlogKxL
kC52XnNRHTQvpr689tqWYvYc+KV5JIXzQf+HMQkCgYEA0+JzxE+SDbvdH0zMEPLO
NlSLJS0gTESD25fdp6a9wd5ACWMob7cZVO2YZtwb9fMigze1Z/dFt12pvhvaklL9
+xWBdMXELzOLSVa+uz0IG+b3rNdSWCX5Fx0hBkfCPYbTbVdV+T6gCc9rOOKUILig
eRWBqj0NS6v5aH2ePa7rtwECgYAWZhzZTvw3oUmYU+Ae2wrIjRL9yYCUtviDQl0D
7CarYhv3mXsGZGPLr1WpUptDBitldm2Fk+g1GxM3pQd0fq6aVMlCFNREaRrfFkr9
oaKUGdhFTSObp5UhRDR1Q3cIlZyqAp/fHGb97BKp3v+GECwV0f8puIiVSjQCrNvG
fbmiOQKBgAKh3/YpCh/ylbjvbTOAZ0LexR7Zu9k7YOYxTXZffAOxNb+WcsAtikbI
zho1Lle3K7MVIEZ3BgnA9pK5OSOFHnd4711n6aBuR7zE7DMx1eeJFwRPRKhRSWzL
FIkCKNKujI26PubeJjwtNRRtGyGqaBWtnv/ZW8n4/u/WnFUp1kDo
-----END RSA PRIVATE KEY-----
```

- Ao criar a configuração você será desconectado da conta do SonarQube e ao conectar novamente, deverá clicar na imagem do GitHub como fosse fazer a configuração e será redirecionado para uma página do GitHub pedindo autorização de acesso. Ao autorizar, o Sonar estará configurado com a conta ou organização.

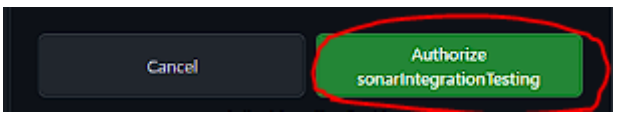


Figura 15: Confirmação de autorização
Fonte: Autor

A partir desse ponto o SonarQube está integrado com o GitHub e pode ler os repositórios da conta ou organização configurada. No caso deste trabalho, foi feita a conexão

com o *fork* do projeto do EducAPI na conta pessoal do autor. No SonarQube, na aba *Projects*, será feita a importação do projeto para posteriormente gerar a execução da análise do Sonar como mostrado nas Figuras 15, 16, 17, 18, 19 e 20:

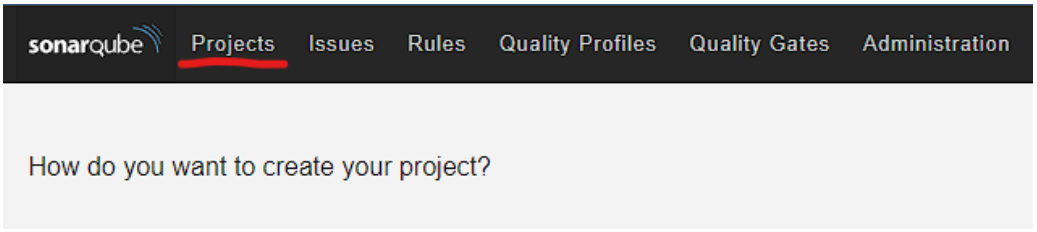


Figura 15: Aba de projeto no SonarQube
Fonte: Autor

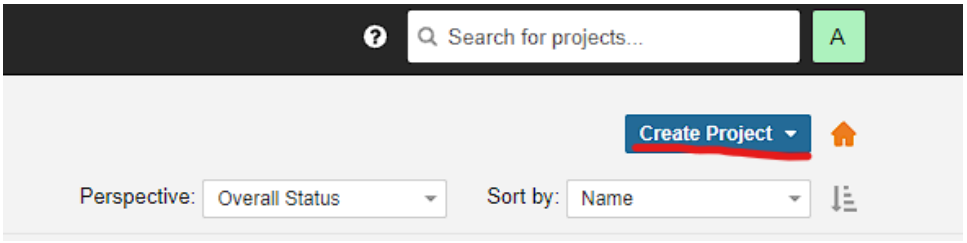


Figura 16: Criar um novo projeto para análise no SonarQube
Fonte: Autor

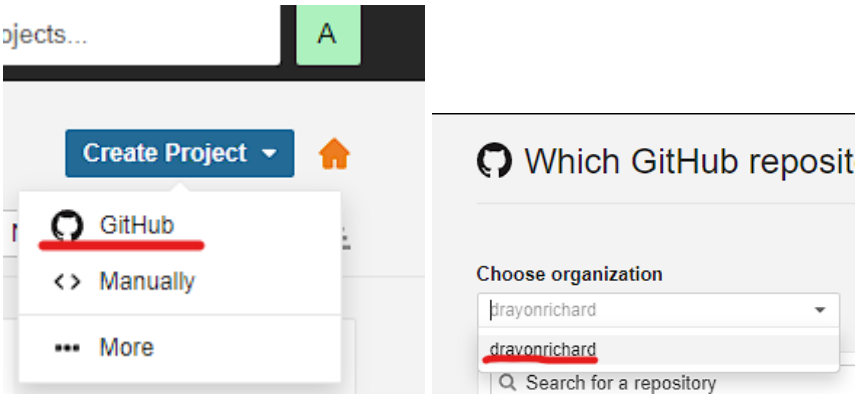


Figura 17: Seleção da plataforma e conta/organização no SonarQube
Fonte: Autor

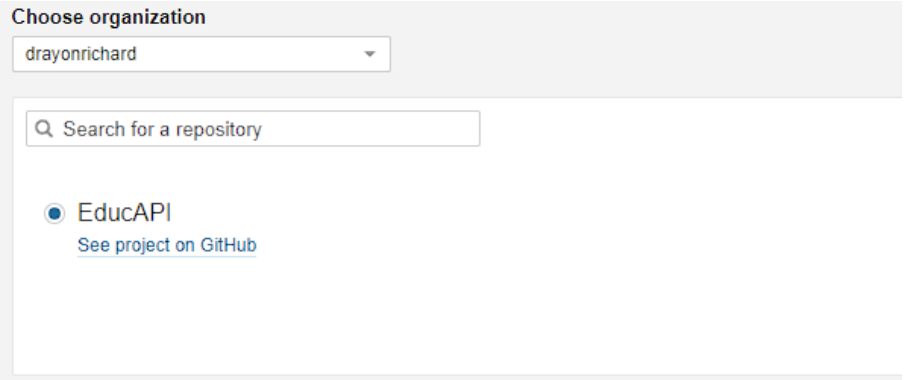


Figura 18: Opções de repositório no SonarQube
Fonte: Autor

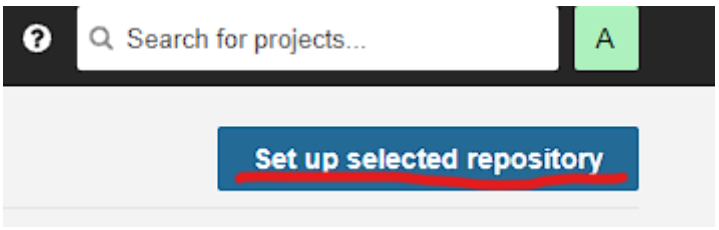


Figura 19: Seleção do repositório a ser analisado no SonarQube
Fonte: Autor

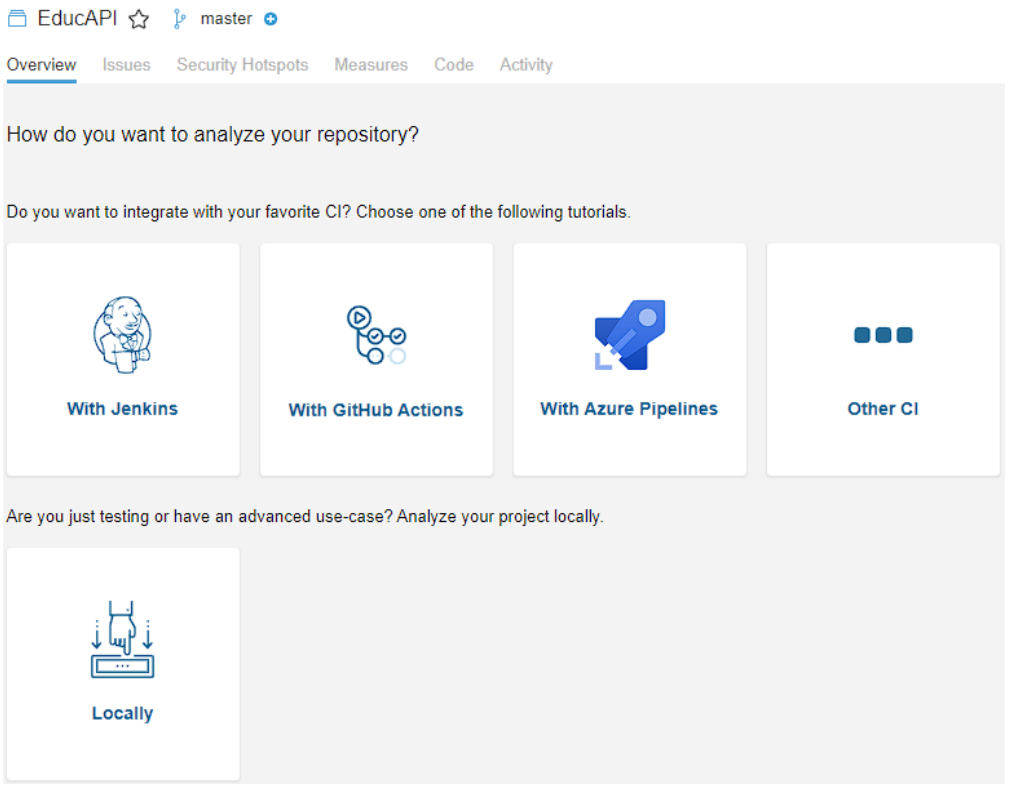


Figura 20: Janela de opções de integração contínua no SonarQube
Fonte: Autor

Por fim, é necessário escolher uma forma de integração contínua (CI) do projeto. Neste trabalho foi usado o ***GitHub Actions*****.** Ao clicar na imagem com nome ***With GitHub Actions,*** será aberta uma página do SonarQube detalhando como configurar os segredos do repositório e o fluxo de execução (*workflow,* que é um *script* ou roteiro de execução*)*. Obs: Para integração com GitHub é necessário que o servidor do SonarQube esteja acessível pela internet, conforme mostrado no exemplo da Figura 21.

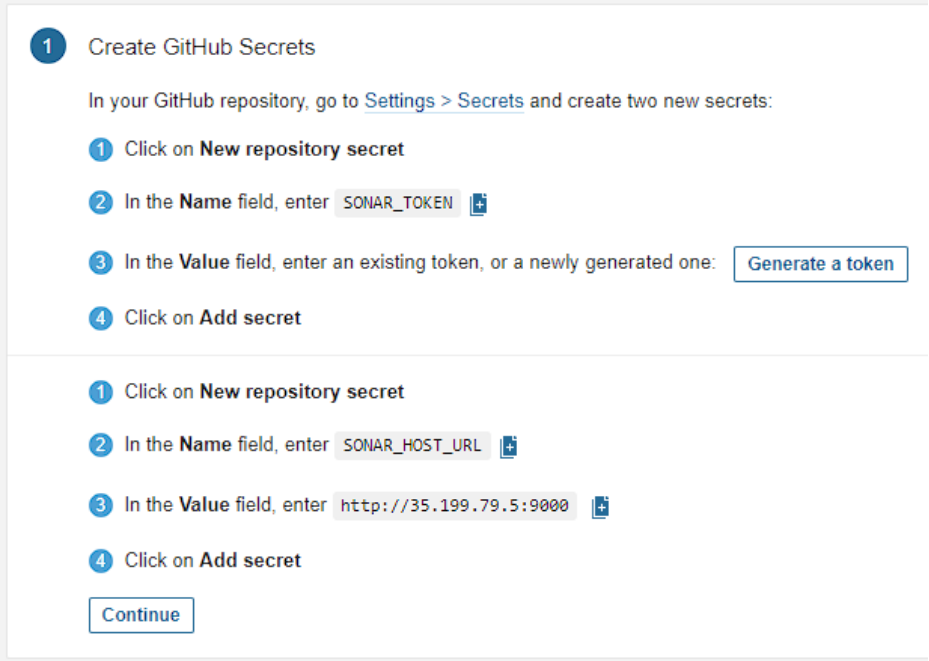


Figura 21: Instalação do *GitHub App* no GitHub
Fonte: Autor

- Indo até o repositório do GitHub na aba de configurações, no *sidebar* (barra lateral) ir em *Secrets* e depois em *Actions*, conforme as Figuras 22 e 23:

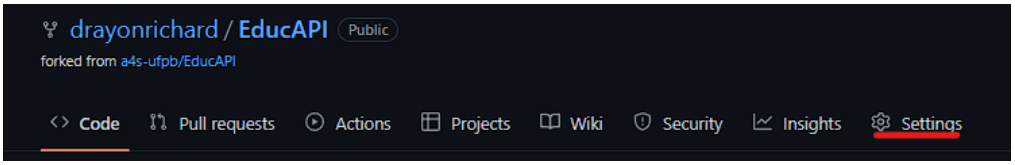


Figura 22: Configuração do repositório no GitHub
Fonte: Autor

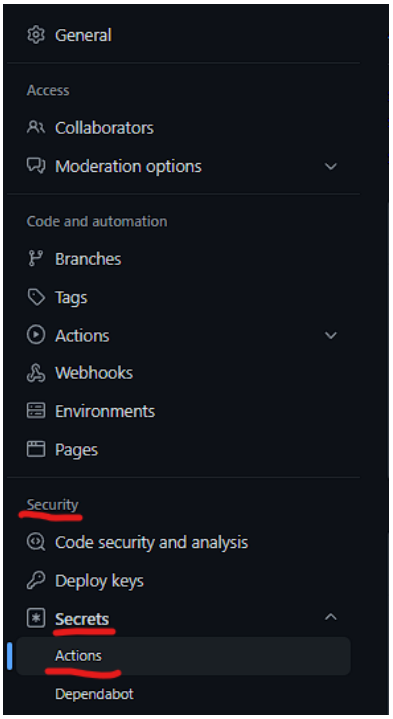


Figura 23: Acessando os segredos do repositório no GitHub
Fonte: Autor

- Na janela de *Actions secrets*, clique em *New repository secret*, conforme a Figura 24 e crie os segredos de acordo com as informações disponibilizadas pelo SonarQube de acordo com a Figura 21. Depois que os segredos forem criados, será exibida uma tela semelhante à mostrada na Figura 25.

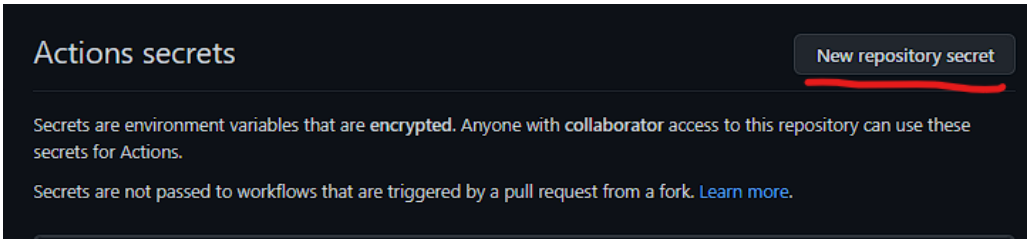


Figura 24: Criando novos segredos de repositório no GitHub
Fonte: Autor

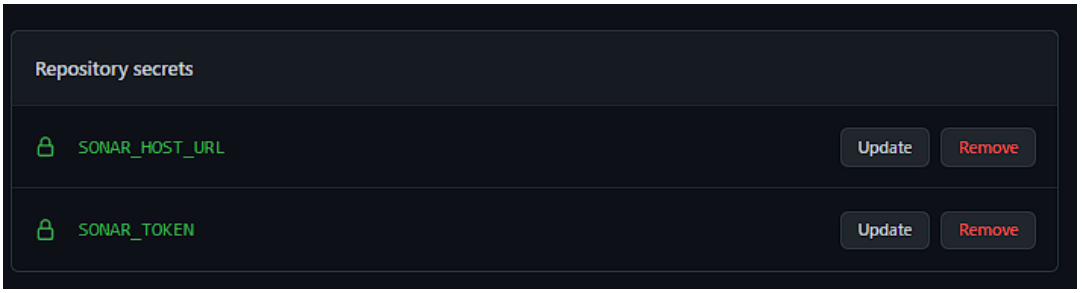


Figura 25: Segredos criados no repositório do EducAPI no GitHub
Fonte: Autor

- Ao finalizar a configuração dos segredos (*Secrets)* do repositório, clique em *continue* na aba de configuração do SonarQube. Dessa forma, será realizado o Passo 2 conforme ilustrado na Figura 26. Neste passo será feito o detalhamento da criação do *workflow* (Fluxo de trabalho), onde deve ser informada qual a ferramenta utilizada para construção do projeto (*build)* e o local do repositório de onde deve ser criado. Um exemplo de ferramenta para *build* de projetos Java é o Maven.

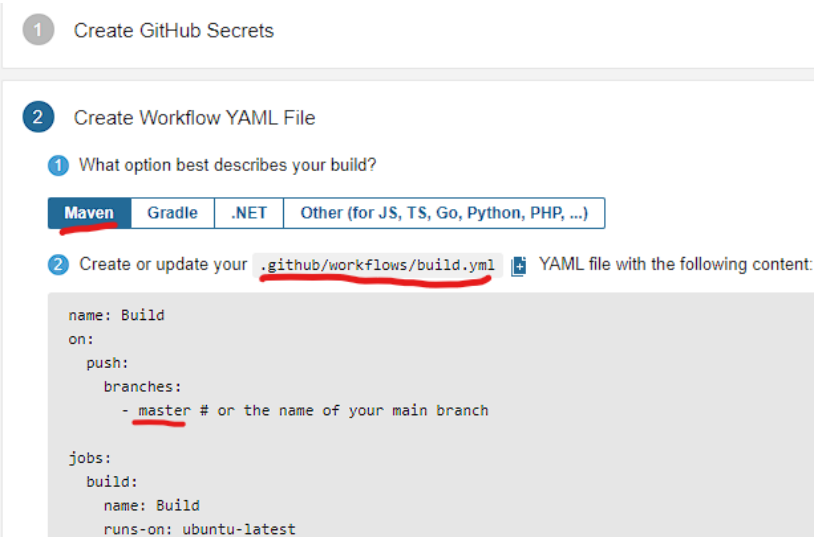


Figura 26: Passo 2 criação do *workflow*
Fonte: Autor

- Depois clique no ícone de cópia, conforme ilustrado na Figura 27;

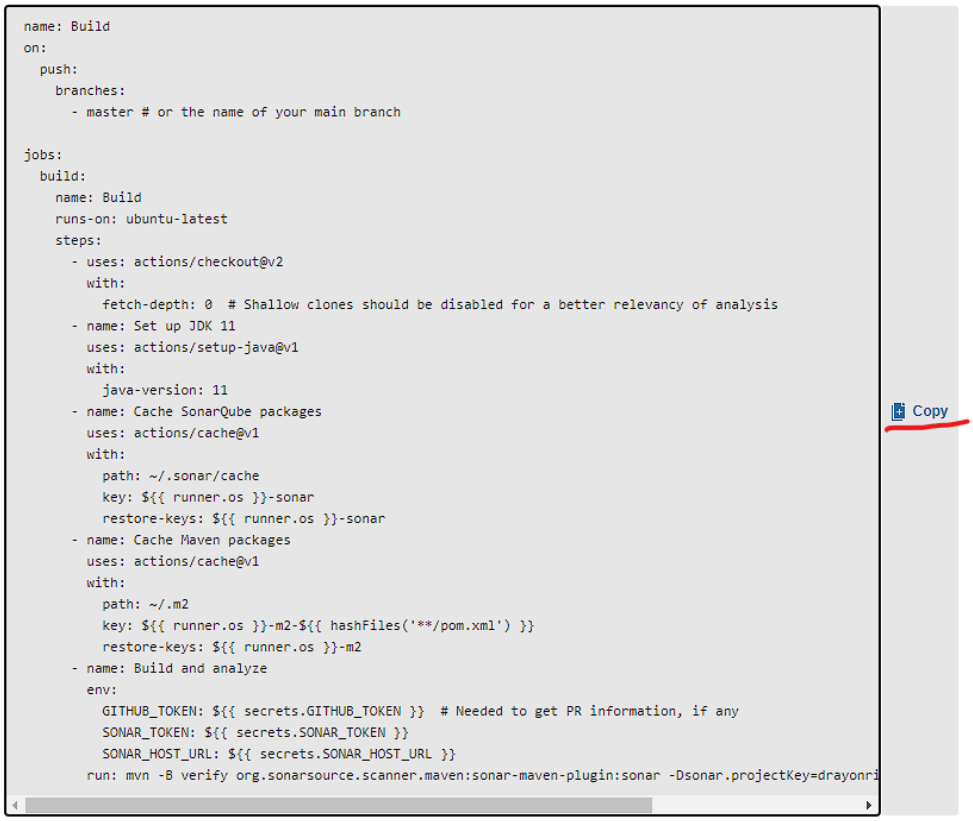


Figura 27: Script do workflow
Fonte: Autor

- Vá até o repositório do GitHub na aba Actions, clique em *New workflow* e depois clique no *link set up a workflow yourself*, conforme as Figuras 28 e 29;

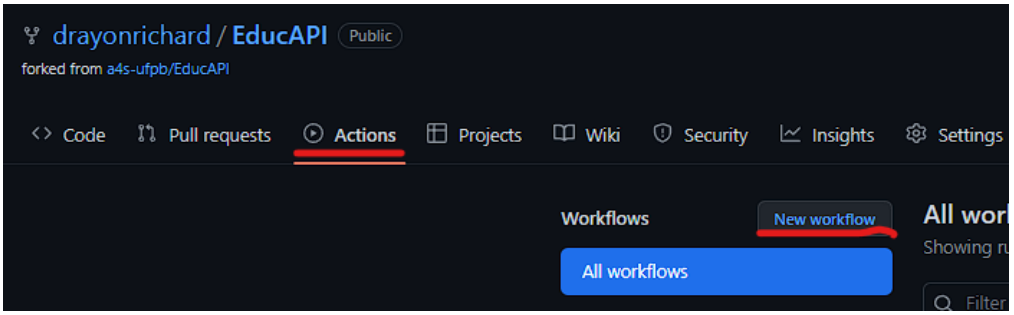


Figura 28: Janela da criação do *workflow* no GitHub
Fonte: Autor

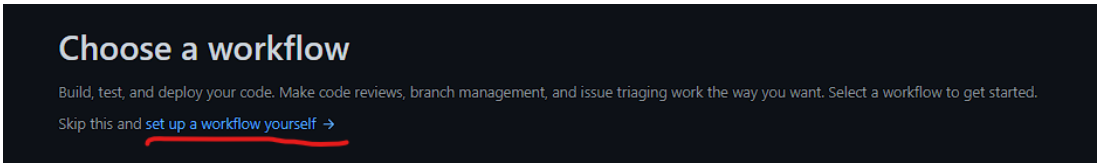


Figura 29: Link de criação manual do *workflow* no GitHub
Fonte: Autor

- Será aberto um arquivo para edição, onde deve ser colocada a configuração copiada. O arquivo pode ser renomeado e enviado ao repositório por meio da operação *commit* do

Git. Depois disso, o repositório estará devidamente configurado. Na configuração se especifica uma ramificação escolhida de versão ou *branch*. No exemplo da Figura 30 se usa “master”.

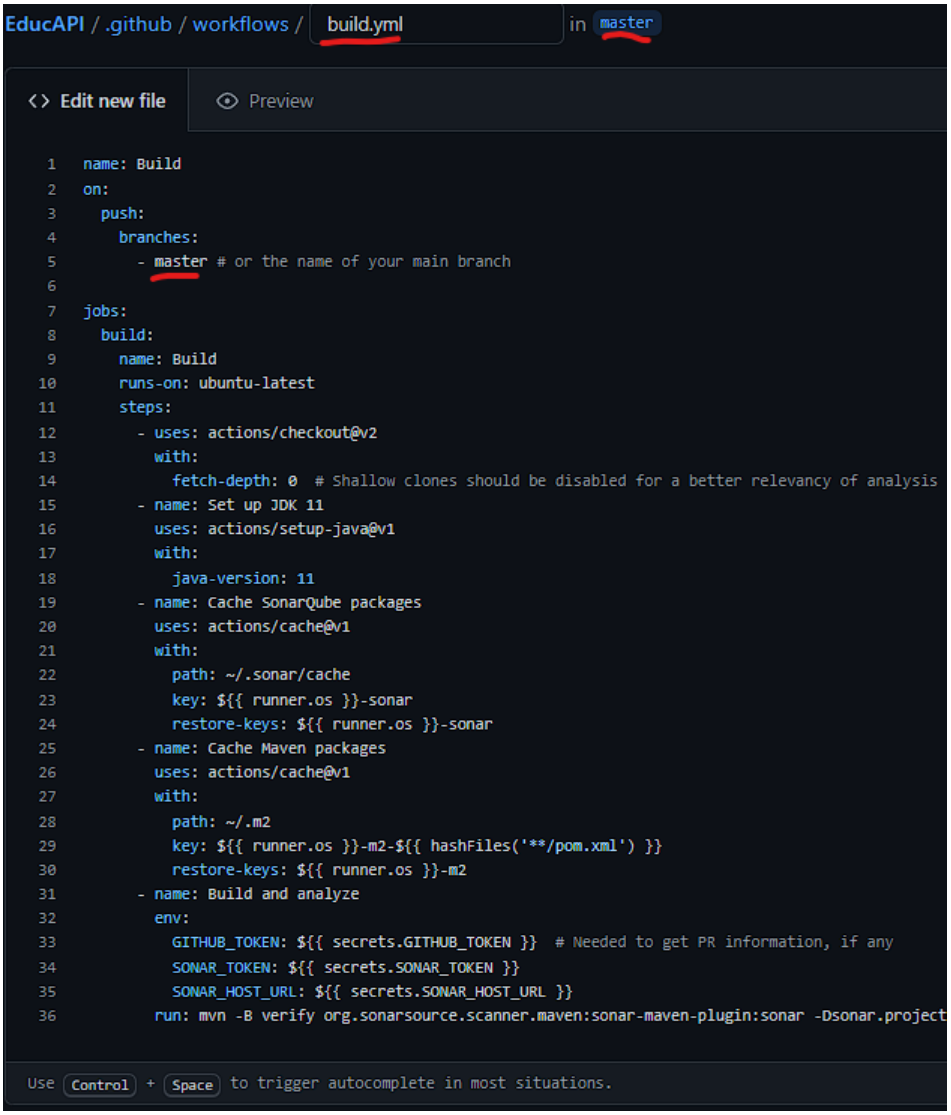


Figura 30: Editor de texto com script do *workflow* no GitHub
Fonte: Autor

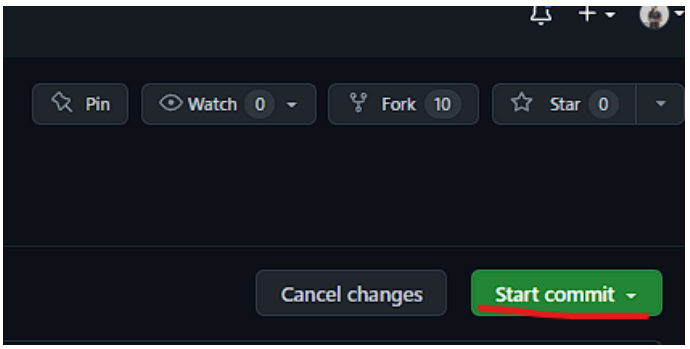


Figura 31: Botão *commit* no GitHub
Fonte: Autor

Agora o projeto está integrado e quando houver alguma alteração (*commit*) na *branch* de análise ou se for forçada a execução da construção do projeto (*build*) via *GitHub Actions,* o Sonar irá gerar um novo relatório a partir do estado do repositório corrente configurado.

4.2 Resultados obtidos no uso do SonarQube

Antes de iniciar a exploração dos resultados gerados pelo SonarQube, é necessário o entendimento do funcionamento da geração de análise de código. O SonarQube utiliza nas suas análises a ferramenta SonarLint, que pode ser utilizada por cada desenvolvedor em suas IDEs (ambientes integrados de desenvolvimento, como IntelliJ, VisualStudio Code, etc). O SonarLint é uma versão Sonar da ferramenta Lint. Sendo um analisador estático de código, ele necessita de uma execução de análise manual por parte do desenvolvedor para a geração das tarefas reportadas. Como destacado na Seção 4, há diversos cenários que podem acontecer quando há vários desenvolvedores trabalhando em um repositório hospedado em uma plataforma e a cada *commit* gerado, pode ser feita nova análise do código pelo Sonar. Para além da análise estática do SonarLint, o SonarQube pode fazer também uma análise dinâmica de código, executando testes automáticos ou verificando cobertura dos testes, por exemplo, mas este não foi o foco deste trabalho. Após realizar análises, o SonarQube pode gerar feedbacks para o desenvolvedor, através da interface web da ferramenta ou por e-mails enviados através da plataforma onde o repositório está hospedado. O SonarLint também pode fornecer este feedback na IDE de cada desenvolvedor. Este processo de verificações está ilustrado pela Figura 32, onde se mostra uma visão geral do uso do SonarQube.

Figura 32: Fluxo de análise do SonarQube
Fonte: SonarQube

Na plataforma web do SonarQube é possível visualizar o relatório gerado pelo SonarQube. Como resultado da inspeção, o SonarQube cria tarefas (*issues*) a serem resolvidas para possíveis correções de Bugs, Vulnerabilidades, *Security Hotspots* (Pontos do código sensíveis que podem oferecer um problema de segurança), *Code Smells*, Duplicação de código e cobertura dos testes unitários. Essas tarefas possuem uma classificação que é definida pelas métricas. O estudo das métricas não será abordado neste trabalho, mas é importante destacar que os pontos notificados recebem classificações como *Blocker (A)*, *Critical* (B), *Major* (C), *Minor* (D), *Info* (E) . Além de cada tarefa possuir sua classificação, é criado um *overview (*visão geral*)* das classificações chamado de *Quality Gate,* e as métricas devem atender aos valores definidos. Cada projeto deve ter um e por *default* (padrão) o Sonar

agrega um *Quality Gate* padrão a todos os novos projetos, mas é uma funcionalidade que pode ser gerenciada para cada projeto.

Métrica	Operador	Valor
Cobertura	é menor que	80%
Porcentagem de linhas Duplicadas	é maior que	3%
Classificação de Manutenibilidade	é pior que	A
Classificação de Confiabilidade	é pior que	A
Pontos de acesso de Segurança revisados	é menor que	100%
Classificação de Segurança	é pior que	A

***Quality Gate* (*default*).**

Fonte: SonarQube

Cada conjunto de métricas possui um *Quality Gate* que é utilizado para realizar uma classificação geral do projeto, que é um indicativo de extrema importância para informar os desenvolvedores quanto à "saúde" do projeto. Caso não atenda aos valores, o projeto terá indicativo de falha em vermelho e caso atenda aos valores, terá um indicativo de que passou (em verde) no *Quality Gate* do projeto. A partir desse ponto os desenvolvedores vão analisar se o projeto está pronto para ser entregue ou se irá necessitar de uma atenção especial de acordo com a análise. Antes de prosseguir, vale ressaltar que o SonarQube gera dois tipos de relatórios, um relatório do *commit* mais recente e outro relatório geral de todo o projeto.

Com a configuração e integração com o *fork* do repositório do EducAPI via GitHub, o SonarQube está disponível para gerar a execução da inspeção de código automática. A versão *Community* (gratuita) do SonarQube é a versão básica e aberta (*open*). Apesar de básica, é uma versão que supre muitas das necessidades na inspeção de código. Existem outras versões do SonarQube que são pagas, porém esse trabalho não tem o intuito de comparar versões e seus recursos.

A inspeção automática do código do projeto EducAPI, utilizado para análise neste trabalho, pode ser visualizada nas Figuras 33 e 34:

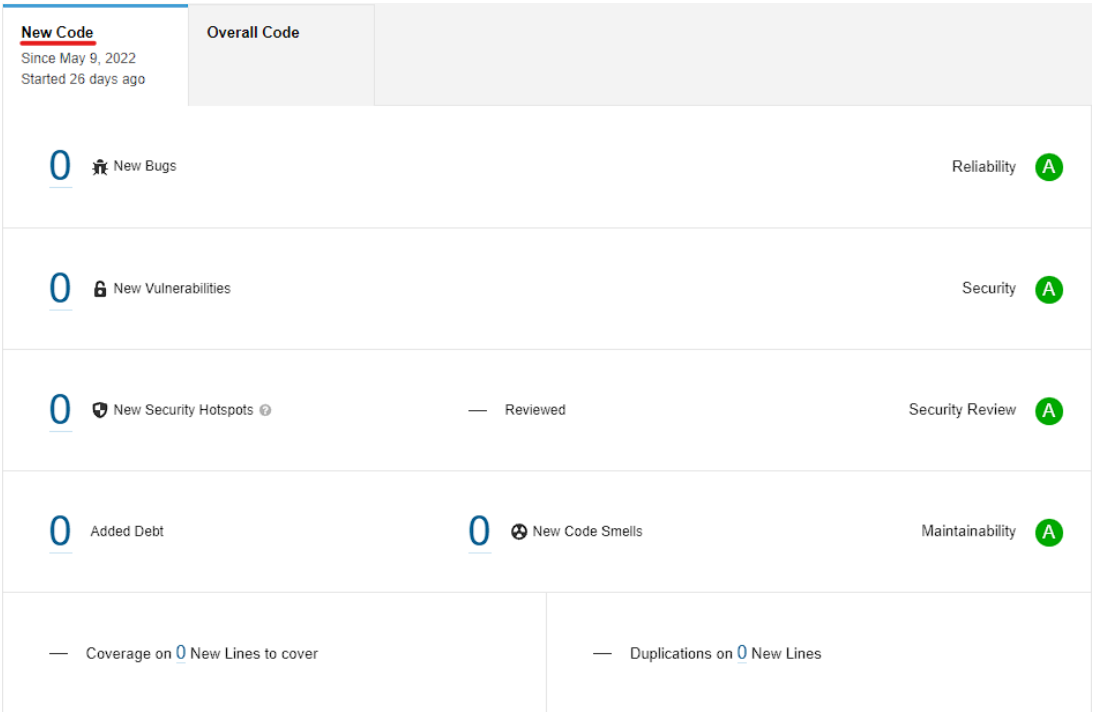


Figura 33: Relatório gerado do último *commit*
Fonte: Autor

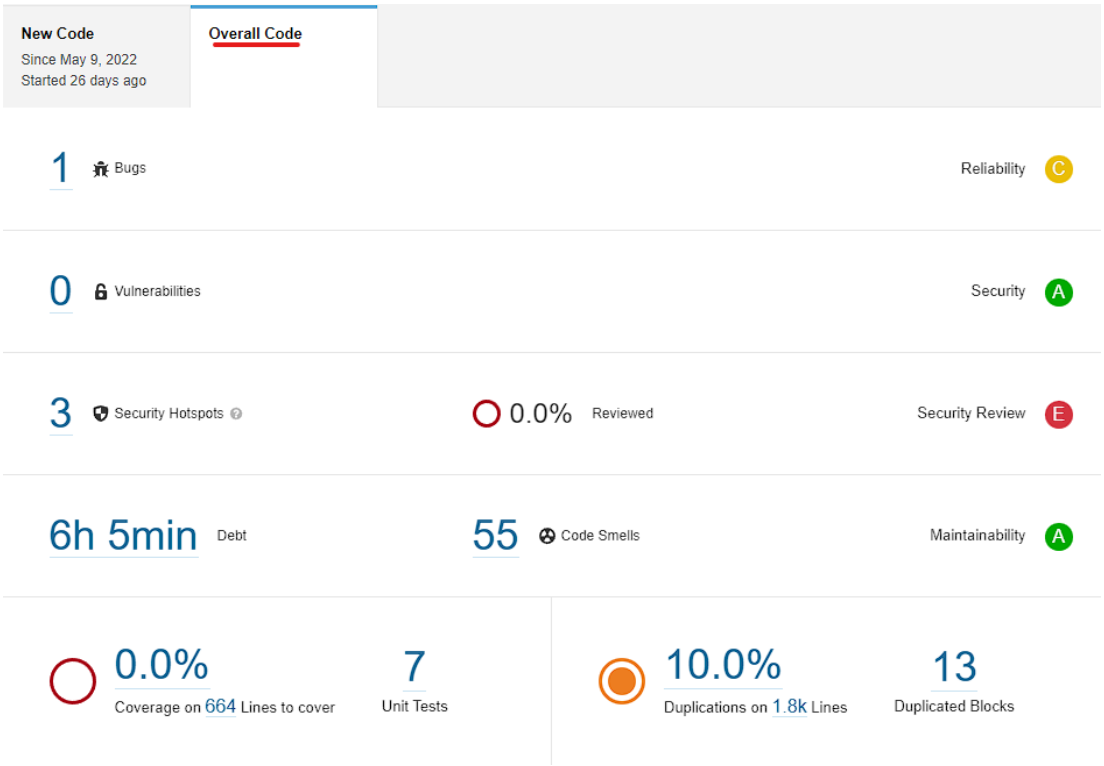


Figura 34: Relatório geral do projeto
Fonte: Autor

Como visto na Figura 33, o *commit* mais recente não gerou nenhuma quebra de boas práticas. Porém, no relatório geral, o Sonar possui tarefas a serem analisadas para possíveis correções: i) 1 tarefa de Bug com classificação C; ii) 3 tarefas de pontos de segurança a serem

revisados de nível E; iii) 55 tarefas de Code Smells com classificação A; iv) 7 testes unitários com 0% de cobertura em 664 linhas; v) 10% de código duplicado.

5. Análise da Utilização do SonarQube

Nesta seção será feita uma análise dos resultados obtidos através da inspeção de código do SonarQube. A análise será feita com base no conhecimento e experiência do autor com o uso da plataforma SonarQube e visa observar se as sugestões de melhoria propostas para o EducAPI são de fato interessantes. A intenção do autor é compartilhar a experiência com outros desenvolvedores de forma que possam entender melhor a etapa de análise de tarefas geradas pela plataforma SonarQube. Para isso, o autor escolheu um projeto *open source* intitulado EducAPI e cujo código fonte está público e acessível, permitindo assim que se possa tentar replicar os passos aqui listados. Além disso, um outro objetivo é também contribuir para a melhoria da qualidade do EducAPI, um sistema desenvolvido por alunos egressos dos cursos de Licenciatura em Ciência da Computação e Bacharelado em Sistemas de Informação da UFPB - Campus IV, e que continua sendo evoluído em projetos de ensino e extensão. A análise dos testes unitários será desconsiderada pela questão do SonarQube não estar reconhecendo ainda os testes unitários que foram implementados e pelo foco deste trabalho em análise estática. Sendo assim, foi desconsiderada a análise dinâmica que demandaria execução dos testes, mas esse é um possível trabalho futuro e que demanda ajustes de configuração. Outro ponto a destacar é que não serão analisadas todas as tarefas, pois a intenção é relatar a experiência do autor com a plataforma para melhor familiarização de futuros leitores e usuários.

5.1 Análise de Tarefas de Bugs

O relatório do Sonar encontrou 1 tarefa de bug a ser analisada. O sonar mostra a possível linha defeituosa, descreve exatamente qual seria o problema e dá exemplo de como resolver aquele tipo de tarefa proposta pelo sonar, conforme ilustrado pela Figura 35.

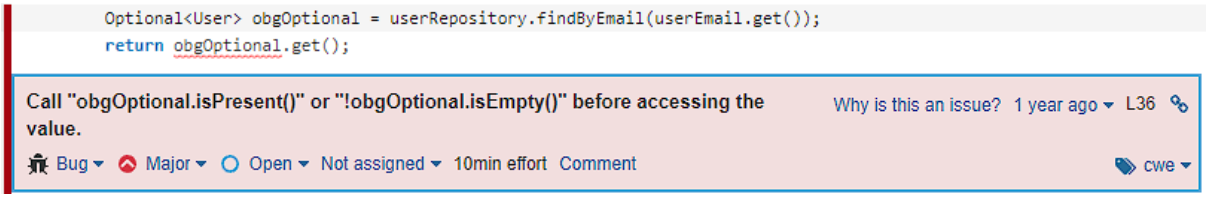


Figura 35: Linha que apresentou Bug
Fonte: Autor

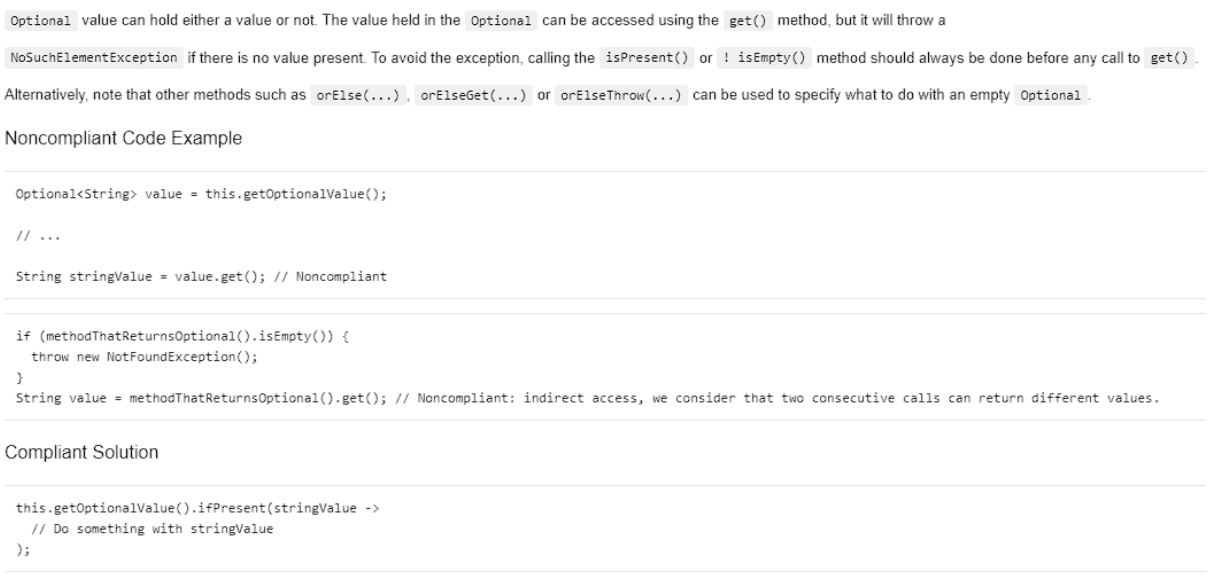


Figura 36: Documentação suporte para a tarefa da Figura 35
Fonte: Autor

A imagem está relatando um bug de possível erro de ponteiro nulo (*Null Pointer*). Analisando, é necessária tal alteração pois ao tentar recuperar a resposta do banco, pode acontecer de o objeto vir vazio e gerar um comportamento inesperado do sistema.

Sendo assim, o sistema sugere que seja adicionada uma validação para verificar se o objeto não veio nulo (Figura 36). Logo, a correção desse bug é relevante para ajudar a proteger o sistema de comportamentos inesperados.

5.2 Análise de Tarefas *Security Hotspots*

O relatório mostrou 3 tarefas *Security Hotspots* (Segurança de ponto de acesso) para serem revisadas. O sonar identificou o uso do *CrossOrigin* em 3 *endpoints* da API. Esse site de acesso a *endpoints* levanta um sinal vermelho no SonarQube, pois pode haver reutilização de sessões ativas. O Sonar sugere que seja feita uma revisão e que se adicione um comentário explicando se aquele ponto de acesso detectado é seguro.

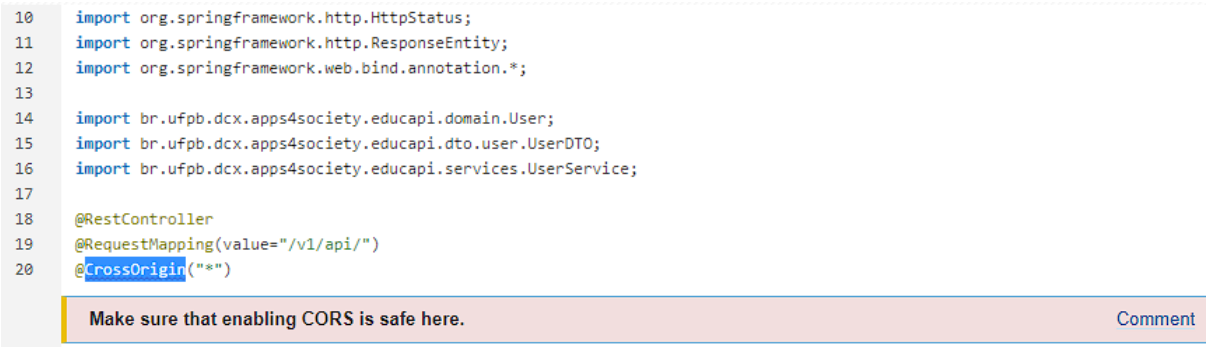


Figura 37: Linha para revisão de *Hotspot* (Ponto de acesso)
Fonte: Autor

O *Crossorigin* permite que outro domínio fora do domínio ao qual pertence o recurso tenha acesso a recursos restritos da API. Pela proposta da API, esse hotspot é algo esperado. Logo, não se trata de um vulnerabilidade e sim um comportamento esperado do sistema. De

qualquer forma, cabe uma análise mais profunda sobre o uso da anotação *CrossOrigin* no EducAPI.

5.3 Análise de Tarefas de *Code Smells*

Esta seção está reservada para a análise de *code smells*. Como há 55 tarefas de *code smells*, será feita uma análise para cada classificação. De acordo com o relatório, existem as seguintes classificações de *Code Smells*, conforme ilustrado na Figura 38: *Blocker, Critical, Major, Minor* e *Info*.

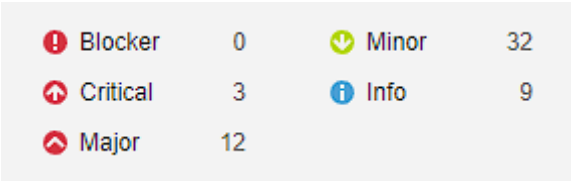


Figura 38: Visão geral dos *code smells* e suas classificações
Fonte: Autor

5.3.1 *Code Smell Critical*

O *Code Smell Critical* apontado no sistema é crítico pois a anotação *@Autowired* diz que as dependências devem ser resolvidas quando a classe for instanciada. No entanto, isso pode causar a inicialização incorreta dos objetos (*Beans*) ou até mesmo fazer o contexto procurar os objetos em locais errados. As Figuras 39 e 40 ilustram um dos relatórios de *Code Smell Critical.*

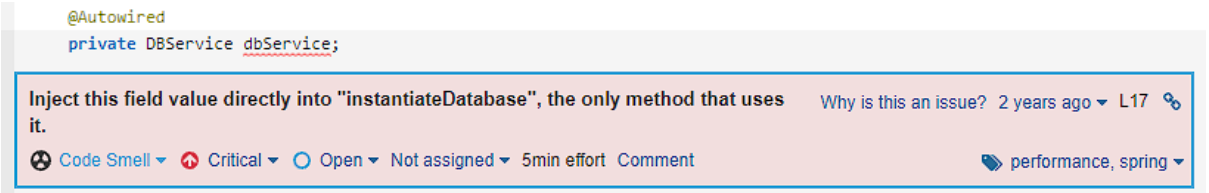


Figura 39: Linha de *code smell critical*
Fonte: Autor

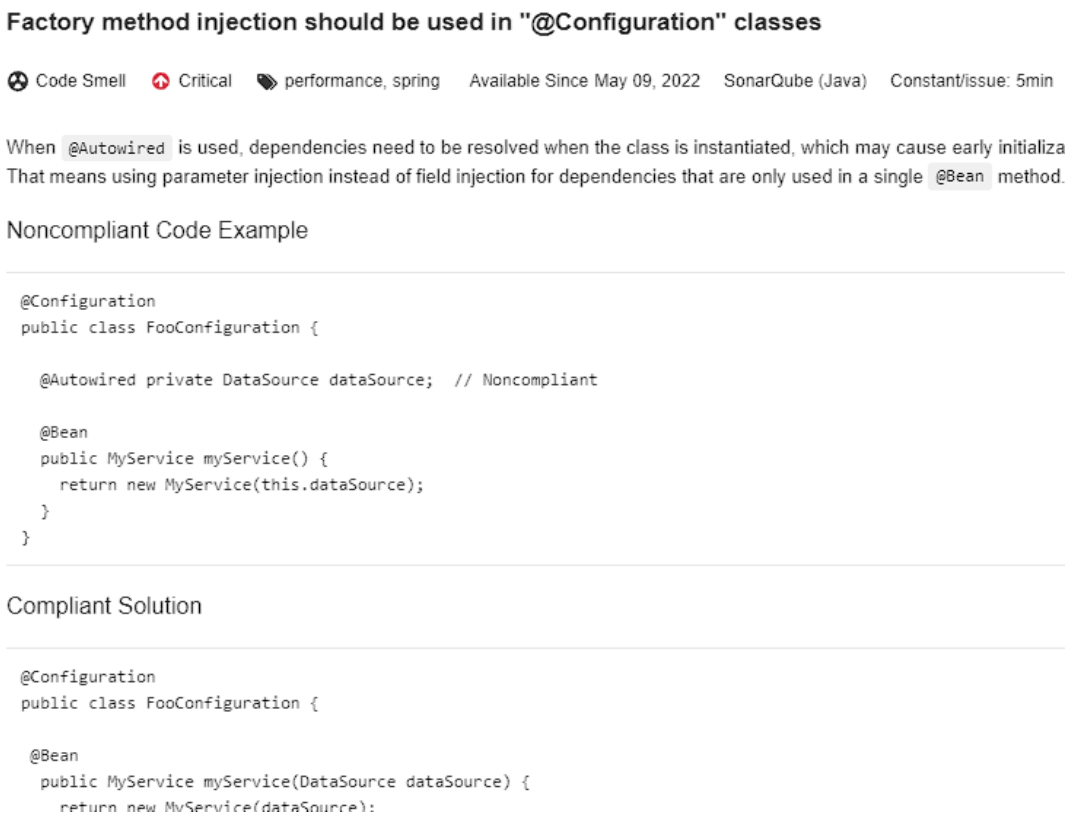


Figura 40: Documentação suporte para a tarefa da Figura 39
Fonte: Autor

O *crossorigin*, permite que outro domínio fora do domínio ao qual pertence o recurso tenha acesso a recursos restritos da API. A melhoria de código proposta pelo Sonar é criar um método com a anotação *@Bean* responsável por fazer a injeção (ou seja, usar diretamente uma dependência) por parâmetro e substituir a injeção por variável.

Após a análise ver que há ganhos de performance ao injetar a dependência por parâmetro com essa mudança, as dependências serão resolvidas o mais tardar possível, gerando uma inicialização mais limpa e rápida das dependências.

5.3.2 *Code Smell Major*

O *Code Smell Major* é uma tarefa sobre classes utilitárias. Essa regra diz que classes utilitárias estáticas não devem ser instanciadas, mesmo sendo classes abstratas. Pede-se para o desenvolvedor a criação de um construtor privado para a classe utilitária, conforme ilustram as Figuras 41 e 42. Na experiência usando a ferramenta no EducAPI, esse foi um dos *Code Smells* mais recorrentes.

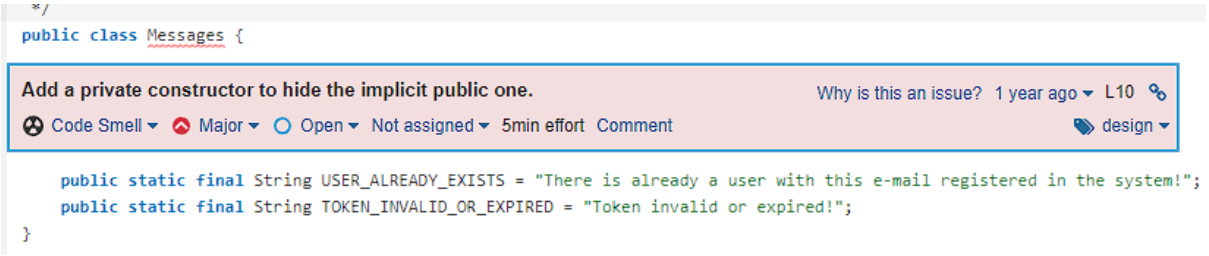


Figura 41: Linha de *code smell major*
Fonte: Autor

Figura 42: Documentação suporte para a tarefa da Figura 41
Fonte: Autor

Ao analisar, seguindo a cultura das boas práticas, é realmente uma boa prática criar construtores privados para classes utilitárias, impedindo que o construtor padrão da classe seja chamado. Para além do SonarQube, marcar a classe utilitária como *final* também impede a classe utilitária de ser estendida sendo essa a melhor prática.

5.3.3 *Code Smell Minor*

A partir da experiência de uso do SonarQube no EducAPI, esse foi o *Code Smell* mais recorrente nas tarefas do SonarQube. Esse *Code Smell* é gerado a partir de quando um desenvolvedor esquece de fazer a remoção de *imports* não mais usados. Essa regra do SonarQube afirma que manter *imports* não utilizados dificulta a leitura do código e confunde outros desenvolvedores. As Figuras 43 e 44 ilustram um relatório indicando um problema deste tipo.

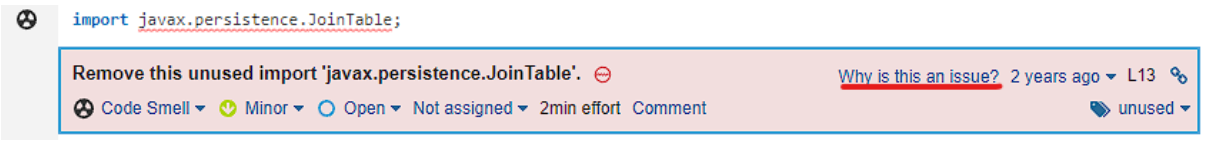
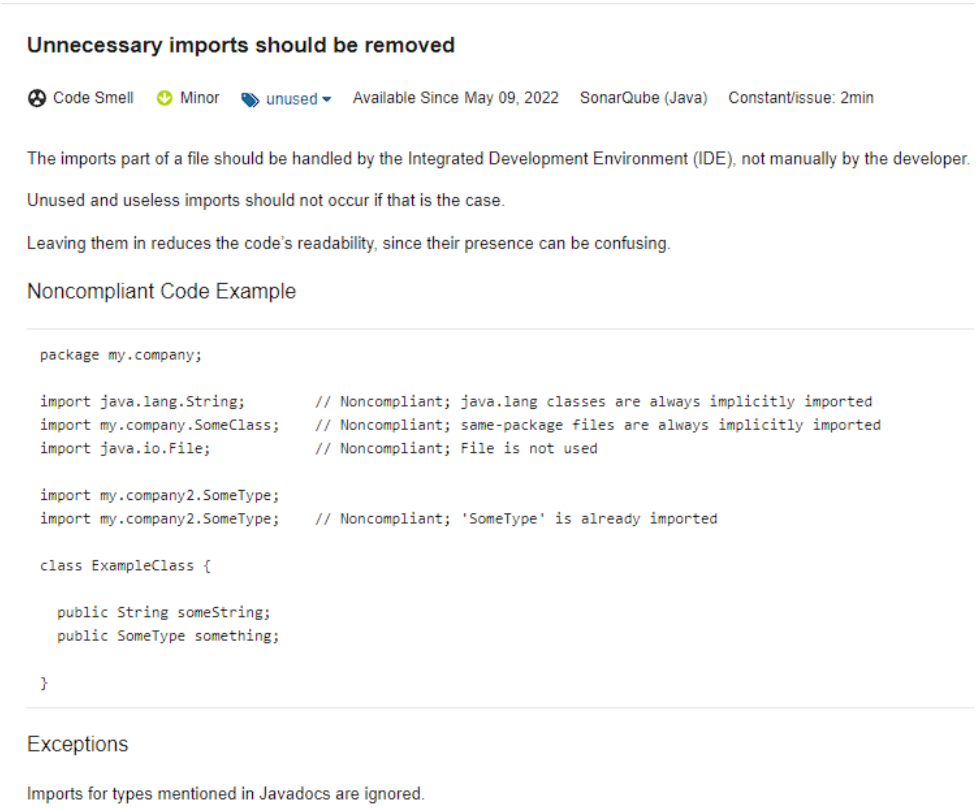


Figura 43: Linha de *code smell minor*
Fonte: Autor



Unnecessary imports should be removed

Code Smell · Minor · unused · Available Since May 09, 2022 · SonarQube (Java) · Constant/issue: 2min

The imports part of a file should be handled by the Integrated Development Environment (IDE), not manually by the developer.

Unused and useless imports should not occur if that is the case.

Leaving them in reduces the code's readability, since their presence can be confusing.

Noncompliant Code Example

```
package my.company;

import java.lang.String;        // Noncompliant; java.lang classes are always implicitly imported
import my.company.SomeClass;    // Noncompliant; same-package files are always implicitly imported
import java.io.File;            // Noncompliant; File is not used

import my.company2.SomeType;
import my.company2.SomeType;    // Noncompliant; 'SomeType' is already imported

class ExampleClass {

  public String someString;
  public SomeType something;

}
```

Exceptions

Imports for types mentioned in Javadocs are ignored.

Figura 44: Documentação suporte para a tarefa da Figura 43

Fonte: Autor

Ao analisar, tem-se que é de extrema importância manter o código o mais legível possível para que o código seja compreendido por qualquer desenvolvedor de maneira rápida. Sendo assim, seguindo as boas práticas, se faz necessária a remoção dos *imports* de classes não utilizadas para garantir a legibilidade do código.

5.3.4 *Code Smell Info*

O *Code Smell Info* é uma tarefa que não quebra nenhuma das boas práticas, porém é uma forma de tornar o código mais legível, sendo essa uma tarefa complementar. Esse *Code Smell* está relacionado à especificação da visibilidade das classes de testes unitários utilizando JUnit 5. A regra fala que o JUnit 5 é mais tolerante a visibilidades e que usar a visibilidade padrão ao invés de `public` tornaria a classe de teste mais legível. As Figuras 45 e 46 ilustram um exemplo de tarefa desse tipo com relação à classe `JWTServiceTest`.

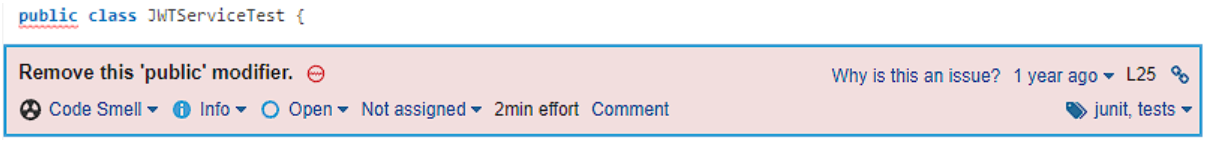


Figura 45: Linha de *code smell info*
Fonte: Autor

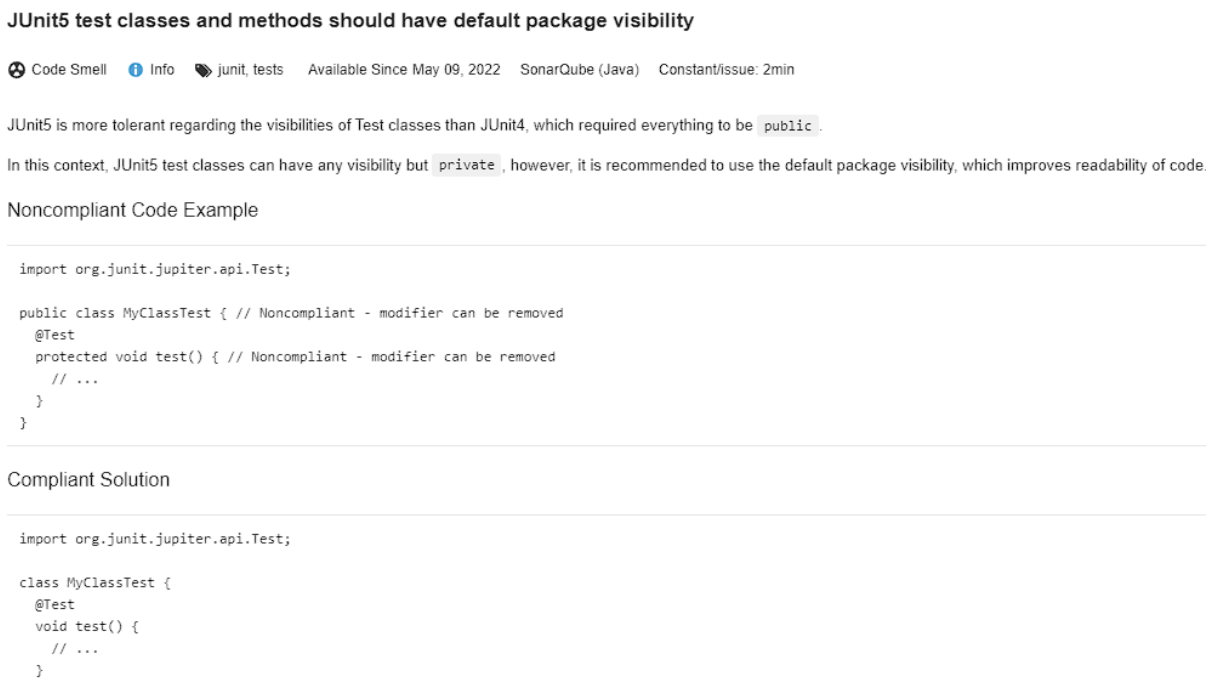


Figura 46: Documentação suporte para a tarefa da Figura 45
Fonte: Autor

Ao analisar, sabe-se que com a adoção do Sonar a um projeto, se faz necessário seguir ao máximo suas recomendações de tarefas, salvo tarefas que o desenvolvedor julgue serem um comportamento correto do código. No caso em particular do EducAPI, a substituição da visibilidade pública pela visibilidade padrão é uma boa prática a ser adotada.

5.4 Análise de Tarefas de *Duplications*

A duplicidade de código é algo que pode impactar um projeto prejudicando a manutenção do código, uma vez que as alterações precisaram ser efetuadas em diversos locais do código se o código problemático estiver repetido. Essa regra do SonarQube é justamente para que se evite que algo desse tipo aconteça, mas cabe ao desenvolvedor analisar a situação e ver se aquela duplicação é necessária ou se realmente é necessária uma refatoração.

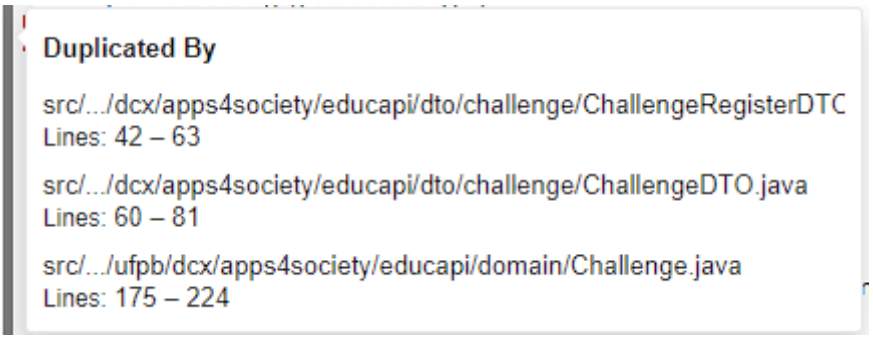


Figura 47: Linhas de código duplicadas
Fonte: Autor

Em um dos blocos de duplicação que o Sonar aponta para o EducAPI, pode-se ver que a duplicação de código está em DTOs e *Models*. A partir da experiência do autor com o desenvolvimento de sistemas REST em Java, é comum ter blocos de duplicação em classes DTOs e Models, pois muitas vezes isso é necessário para isolar objetos e trabalhar de uma

forma mais limpa e intuitiva na manipulação dos mesmos. Deste modo, não foi considerado necessário fazer essa refatoração com base na análise feita.

6. Lições Aprendidas

Aqui serão detalhadas as principais lições aprendidas durante a etapa de desenvolvimento do presente trabalho. Houve várias dificuldades durante a elaboração do trabalho, mas superados com cautela e destreza.

Do ponto de vista técnico houve bastante aprendizado e aqui listo os principais: i) Durante a etapa de configuração da máquina e do ambiente não haviam sido considerados os pré-requisitos de instalação do SonarQube, havendo assim muita dificuldade em entender porque a máquina estava demorando tanto para levantar o servidor do SonarQube. Foram feitas algumas tentativas visando tentar reservar menos memória, mas por fim foi necessário ter uma máquina mais robusta para conseguir rodar o SonarQube sem erros; ii) A integração do SonarQube com GitHub foi feita seguindo a própria documentação do Sonar, porém a documentação em relação à configuração de integração não é tão clara, deixando brechas para um bom entendimento, o que tornou complexa a configuração do *GitHub App* com o SonarQube. No campo para preencher a URL do GitHub API, interpretou-se de uma maneira errada e estava-se tentando usar a URL da API do repositório e da conta do GitHub sem sucesso e não estava compreensível o porquê, havendo verificações sobre se alguma configuração poderia estar errada, mas ao tentar usar a URL de Exemplo funcionou; iii) A partir do SonarQube é possível fazer análise de projetos e verificar vulnerabilidades rapidamente; iv) O *fork* do GitHub apesar do conhecimento da ferramenta mostrou-se extremamente útil principalmente para criar softwares baseando-se em projetos já criados ou dar continuidade a um projeto descontinuado. v) O *GitHub App* foi uma descoberta de funcionalidade que impressionou pois sua aplicação mostrou-se fácil e bastante útil.

7. Conclusões e Trabalhos Futuros

Esse trabalho teve como objetivo principal identificar como o SonarQube poderia melhorar a qualidade de código do sistema EducAPI. Através da integração do SonarQube com o repositório no GitHub do projeto EducAPI, foram definidos os passos para esta integração e foram geradas tarefas na plataforma do SonarQube com possíveis melhorias para o projeto. A partir desse ponto foram analisadas tarefas por tipo, destacando a importância da resolução da tarefa proposta pelo SonarQube. Neste trabalho foram identificadas melhorias no código fonte do EducAPI e que são relevantes para a melhoria do sistema, além de práticas de integração do SonarQube com projetos e que podem ser exploradas em outros projetos seguindo passos semelhantes. Acreditamos assim, que o objetivo geral do trabalho foi alcançado.

As principais contribuições deste trabalho são as seguintes: i) Criação de material instrucional sobre a configuração do ambiente SonarQube no sistema operacional Ubuntu/Linux; ii) Criação de material instrucional sobre a configuração da plataforma SonarQube com a plataforma GitHub; iii) Criação de material instrucional sobre configuração de integração da plataforma SonarQube a novos e antigos projetos; iv) Detalhamento de propostas de melhorias na qualidade de software do projeto EducAPI; v) Análise dos tipos de tarefas geradas pela plataforma SonarQube e que poderiam ou não gerar alterações no sistema EducAPI, de acordo com sua relevância.

Possíveis trabalhos futuros são a correção no EducAPI dos apontamentos feitos pelo SonarQube e a correção da configuração para permitir análise dos testes unitários no SonarQube para que seja possível visualizar a cobertura de testes na plataforma. Outro possível trabalho futuro é a aplicação do SonarQube em outros projetos com base nos passos mapeados neste trabalho e também realizar estudos aprofundados das métricas do SonarQube e de sua importância.

Referências

Baldassarre, Maria Teresa et al (2020), On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube, Information and Software Technology, Volume 128, 2020, 106377, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2020.106377.

Chelf, Ben; Chou, Andy. (2007). The next generation of static analysis.

Fitzpatrick, Ronan. (1996) Definitions and strategic issues. Software quality.

Fowler, Martin. (1999). Refactoring: Improving the Design of Existing Code. AddisonWesley Professional, 1st edition.

Gomes, Ivo; Morgado, Pedro; Gomes, Tiago; Moreira, Rodrigo. (2007). An overview on the Static Code Analysis approach in Software Development.

Hunt, A.; Thomas, D. (1999) The Evils of Duplication. The Pragmatic Programmer.

Lima, Vagner Carlos Marcolino (2013). Programação em par: investigando sua eficácia perante tarefas de modelagem e construção de software. 2013. 122 f. Dissertação (Mestrado em Computação Aplicada) – Universidade Tecnológica Federal do Paraná, Curitiba.

Ludgerio, R.; Amorim, L. V. ; Dantas, Emerson; Rebouças, Ayla Dantas (2021). O sistema colaborativo EducAPI e sua avaliação como ferramenta para apoiar a alfabetização. REVISTA TECNOLOGIAS NA EDUCAÇÃO, v. 35, p. 1-18.

M. Meyer, "Continuous Integration and Its Tools," in IEEE Software, vol. 31, no. 3, pp. 14-16, May-June 2014, doi: 10.1109/MS.2014.58.

Marcilio, Diego; Bonifácio, Rodrigo; Monteiro, Eduardo; Canedo, Edna; Luz, Welder and Pinto, Gustavo (2019). "Are Static Analysis Violations Really Fixed? A Closer Look at Realistic Usage of SonarQube," *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 209-219, doi: 10.1109/ICPC.2019.00040.

Masse, Mark (2011). REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. O'Reilly Media, Inc.

Medeiros, João Eduardo Ribeiro de (2017). Estudo comparativo de ferramentas de análise estática de código / João Eduardo Ribeiro de Medeiros.

Plösch, Reinhold; Neumüller, Cornelia (2020). Does Static Analysis Help Software Engineering Students? In Proceedings of the 2020 9th International Conference on Educational and Information Technology (ICEIT 2020). Association for Computing Machinery, New York, NY, USA, 247–253, DOI: https://doi.org/10.1145/3383923.3383957

Pressman, R.S. (2001). “Software Engineering: A Practitioner’s Approach”, Fifth Edition, McGraw Hill.

Saarimaki, Nyyti et al (2019), "On the Accuracy of SonarQube Technical Debt Remediation Time," 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2019, pp. 317-324, doi: 10.1109/SEAA.2019.00055.

Shanin, Mojtaba; Ali Babar, Muhammad; Zhu, Liming (2017). "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," in IEEE Access, vol. 5, pp. 3909-3943, doi: 10.1109/ACCESS.2017.2685629.

Shelajev, Oleg; Muuga, Sigmar; Maple, Simon; White, Oliver. The Wise Developers’ Guide to Static Code Analysis featuring FindBugs, Checkstyle, PMD, Coverity and SonarQube. Disponível em: <http://zeroturnaround.com/rebellabs/developers-guide-static-code-analysis-findbugscheckstyle-pmd-coverity-sonarqube/>.