Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Sistemas de Informação
Trabalho de Conclusão de Curso – Artigo

Uma ferramenta para análise de qualidade de código fonte utilizando Inteligência Artificial

Rafael Reis de Oliveira[1]
Centro Universitário Academia, Juiz de Fora, MG

Daves Marcio Silva Martins[2]
Centro Universitário Academia, Juiz de Fora, MG

Linha de pesquisa: Engenharia de Software

Resumo

A qualidade do código transcende sua mera funcionalidade, influenciando diretamente a eficiência operacional e a sustentabilidade econômica de projetos de software. Nesse contexto, a inteligência artificial desempenha um papel revolucionário, impactando diversos setores. Na saúde, por exemplo, a IA melhora diagnósticos e personaliza tratamentos, enquanto no setor financeiro, ela analisa riscos, identifica fraudes e automatiza transações. Na logística, a tecnologia otimiza processos de entrega e gerenciamento de suprimentos. No desenvolvimento de software, a integração de IA representa um avanço significativo, impulsionando inovação, eficiência e economia em projetos e equipes. Na esfera da qualidade de software, a inteligência artificial transforma a análise e revisão de código. Ferramentas baseadas em IA são capazes de identificar padrões complexos, detectar vulnerabilidades e sugerir melhorias, o que aumenta a eficiência operacional e reduz os custos de

[1] Discente do Curso de Sistemas de Informação do Centro Universitário Academia - UniAcademia. E-mail: rafaeloliveira63@yahoo.com

[2] Docente do Curso de Sistemas de Informação do Centro Universitário Academia - UniAcademia. Orientador.

desenvolvimento. Isso, por sua vez, eleva a qualidade e viabilidade econômica do software, evidenciando o impacto significativo da IA neste campo. Este artigo apresenta a ferramenta AI Code Analyzer, investigando como a inteligência artificial pode ser aplicada para analisar o código, identificar defeitos e propor melhorias substanciais. Integrando-se de forma ágil e assertiva aos ambientes de desenvolvimento e aos repositórios de código-fonte, e utilizando análises aprofundadas de métricas de qualidade, a ferramenta oferece uma abordagem proativa e eficaz. Essa integração permite a identificação e resolução precoce de problemas, facilitando a manutenção contínua da integridade e eficiência do código. O estudo enfatiza o impacto transformador da IA na gestão da qualidade do código, destacando sua importância crucial na promoção da melhoria contínua e no sucesso sustentável dos projetos de desenvolvimento de software.

Palavras-chave: Inteligência Artificial. Desenvolvimento de Software. Qualidade de Software.

ABSTRACT

The quality of code transcends its mere functionality, directly influencing operational efficiency and the economic sustainability of software projects. In this context, artificial intelligence plays a revolutionary role, impacting various sectors. In healthcare, for example, AI improves diagnostics and personalizes treatments, while in the financial sector, it analyzes risks, identifies fraud, and automates transactions. In logistics, the technology optimizes delivery processes and supply management. In software development, the integration of AI represents a significant advance, driving innovation, efficiency, and cost-effectiveness in projects and teams. In the realm of software quality, artificial intelligence transforms code analysis and review. AI-based tools can identify complex patterns, detecting vulnerabilities, and suggesting improvements, which enhance operational efficiency and reduce development costs. This, in turn, raises the quality and economic viability of the software, highlighting the significant impact of AI in this field. This article introduces the AI Code Analyzer tool, investigating how artificial intelligence can be applied to analyze code, identify defects, and propose substantial improvements. By integrating swiftly and assertively into development

environments and code repositories, and utilizing in-depth analysis of quality metrics, the tool offers a proactive and effective approach. This integration allows for early identification and resolution of issues, facilitating the continuous maintenance of code integrity and efficiency. The study emphasizes the transformative impact of AI on code quality management, highlighting its crucial importance in promoting continuous improvement and the sustainable success of software development projects.

1. INTRODUÇÃO

Estudos recentes indicam que a qualidade do código-fonte é crucial para o sucesso de um projeto de desenvolvimento de software, impactando diretamente aspectos como legibilidade, manutenção, escalabilidade e desempenho. Um código de alta qualidade não apenas facilita a colaboração entre desenvolvedores, mas também melhora a detecção e correção de erros, reduzindo significativamente os custos de manutenção (Fitzgerald e Stol, 2020). Além disso, a complexidade do desenvolvimento de software moderno exige códigos claros e modulares, que suportem mudanças sem comprometer a estabilidade do sistema (Capretz e Ahmed, 2021). Projetos com códigos de baixa qualidade frequentemente enfrentam desafios como manutenção complexa, dificuldades ao adicionar novas funcionalidades e maior propensão a bugs (Zheng e Zhou, 2022). Em 'Code Complete', Steve McConnell (2004) enfatiza que a alta qualidade do código facilita as modificações futuras, reduz o custo de manutenção e diminui o número de erros, todos essenciais para o sucesso a longo prazo de qualquer software. O autor argumenta que a clareza e a modularidade do código são fundamentais em ambientes que exigem adaptação rápida e confiável a novas exigências sem comprometer a estabilidade do sistema. Neste contexto, a análise contínua da qualidade do código é essencial para mitigar riscos e garantir o sucesso em ambientes dinâmicos e competitivos.

Este estudo explora abordagens inovadoras para avaliar a qualidade do código, empregando modelos de inteligência artificial. A ferramenta *AI Code Analyzer* foi desenvolvida para demonstrar como as técnicas de Inteligência Artificial podem aprimorar a compreensão da qualidade do código. Ela oferece insights valiosos e propõe melhorias efetivas. Adicionalmente, a pesquisa enfatiza a relevância de se adotar métricas de qualidade do código, cujos detalhes serão explorados em capítulos

posteriores.

Por fim, a integração de métricas de desenvolvimento em tempo real e a análise completa de repositórios de código-fonte visa superar as limitações das abordagens tradicionais, proporcionando avanços significativos na eficácia do desenvolvimento de software. A análise detalhada dos métodos, resultados e conclusões deste estudo será apresentada nos capítulos subsequentes.

Este artigo foi dividido da seguinte forma. Após uma introdução detalhada na seção 1 sobre a importância da qualidade do código fonte em projetos de desenvolvimento de software e o papel das ferramentas de análise de código, o documento se desdobra em diversas seções que abordam temas específicos. A seção 2, "Referencial Teórico", apresenta as bases teóricas que sustentam a análise de qualidade do código e discute métricas e ferramentas relevantes no campo. Segue-se a descrição da "Ferramenta AI Code Analyzer" na seção 3, onde são explorados os detalhes técnicos e a funcionalidade da ferramenta desenvolvida. Posteriormente, a seção 4, "Resultados e Utilização" ilustra os resultados práticos obtidos com a aplicação da ferramenta em cenários reais de desenvolvimento, ressaltando a eficácia dela. Finalmente, na seção 5 o artigo se conclui com as "Considerações Finais", onde são destacadas as contribuições do estudo para o campo da engenharia de software e são sugeridos caminhos para futuras pesquisas.

2. REFERENCIAL TEÓRICO

No contexto do desenvolvimento de software, a qualidade do código não apenas dita sua funcionalidade imediata, mas também afeta profundamente sua sustentabilidade e adaptabilidade a longo prazo. Isso envolve uma série de características essenciais, incluindo a legibilidade, eficiência, reusabilidade, testabilidade e manutenibilidade. Métricas como essas são vitais para que o código seja facilmente compreendido e modificado por outros desenvolvedores, o que facilita o processo de manutenção e adaptação a novas necessidades.

Para avaliar a qualidade do código de maneira objetiva, foram desenvolvidas diversas métricas. Entre as ferramentas mais notáveis que automatizam essa análise,

destacam-se o *SonarQube*[3] *e* o *CodeClimate*[4]*. O SonarQube* oferece uma visão contínua e multidimensional da qualidade, apontando problemas de segurança, bugs e code smells ao longo do desenvolvimento do software. Por sua vez, *CodeClimate* se integra aos fluxos de trabalho de CI/CD, fornecendo feedback automatizado para refinamento da qualidade e manutenibilidade do código.

Outro exemplo prático é a ferramenta *PHP Analyzer*[5]*,* desenvolvida por Vicente e Sirqueira (2022), que adota uma abordagem de análise estática do código-fonte para identificar funções vulneráveis e depreciadas que podem impactar aplicações web. Tipos de ferramentas como essas é crucial para o processo de manutenção e evolução do software, auxiliando na detecção de problemas antes que se tornem críticos. A implementação de ferramentas como estas, especializadas para a análise de métricas de qualidade pode significar elevar a qualidade do software, culminando em produtos mais robustos, eficientes e de fácil manutenção.

A integração de técnicas de aprendizado profundo na análise de código tem revolucionado a forma como as métricas de qualidade são avaliadas. Ferramentas como *DeepCode*[6] e Tabnine[7], que utilizam algoritmos de aprendizado de máquina, demonstram uma capacidade notável de identificar nuances complexas no código que muitas vezes escapam à detecção humana. Essas ferramentas oferecem sugestões de correções e melhorias com base em vastos repositórios de dados e práticas recomendadas de codificação, elevando o padrão de qualidade do software desenvolvido.

Além disso, a emergência de assistentes de programação baseados em IA,

---

[3] SonarQube. Disponível em: <https://docs.sonarsource.com/sonarqube/latest/>. Acesso em: 16 de maio de 2024.

[4] CodeClimate. Disponível em: <https://docs.codeclimate.com/docs/getting-started-with-code-climate>. Acesso em: 16 de maio de 2024.

[5] PHP Analyzer. Disponível em: <https://seer.uniacademia.edu.br/index.php/cesi/article/view/2544>. Acesso em: 16 de maio de 2024.

[6] DeepCode. Disponível em: <https://snyk.io/pt-BR/platform/deepcode-ai/>. Acesso em: 16 de maio de 2024.

[7] Tabnine. Disponível em: <https://www.tabnine.com/>. Acesso em: 16 de maio de 2024.

como o GitHub Copilot[8], que utiliza o modelo de linguagem GPT-3[9] representa um marco na assistência ao desenvolvimento de software. Essas ferramentas não apenas sugerem trechos de código em tempo real mas também aprendem dinamicamente com as interações do usuário, proporcionando uma experiência de codificação mais intuitiva e eficiente.

A integração de tecnologias de Inteligência Artificial com práticas tradicionais de desenvolvimento de software, como revisões de código, está marcando o início de uma nova era na engenharia de software. Estas ferramentas avançadas vão além da mera detecção de erros; elas atuam como conselheiras, guiando os desenvolvedores na adoção de soluções ótimas e incentivando uma cultura de excelência e inovação contínua. Neste contexto dinâmico, conceitos clássicos de engenharia de software como a “Complexidade Ciclomática” de McCabe (1976), que avalia a complexidade do código ao medir o número de caminhos de execução possíveis, e a análise de “code smells” de Kent Beck (1999) e Martin Fowler (1999) , que identifica padrões problemáticos que podem não ser imediatamente visíveis como bugs, são reavaliados. Esses conceitos são essenciais para antecipar e mitigar riscos que podem comprometer a adaptabilidade e a eficiência do software no longo prazo. Através desta sinergia entre IA e práticas estabelecidas, desenvolvedores podem alcançar uma compreensão mais profunda e uma resposta mais eficaz aos desafios do desenvolvimento moderno de software.

Essas iniciativas de IA não apenas aceleram o processo de desenvolvimento, mas também elevam a barra para a qualidade do software, promovendo práticas de codificação mais robustas e seguras. A integração de IA na análise de código está se tornando uma prática padrão na indústria, reconhecida por sua eficácia em melhorar significativamente a manutenibilidade e a eficiência dos projetos de software.

A busca incessante pela melhoria da qualidade do código torna-se fundamental para o desenvolvimento de software que não apenas atenda às demandas atuais de forma eficaz, mas também se mantenha sustentável ao longo do tempo. A adoção do

[8] GitHub Copilot: Disponível em: <https://docs.github.com/en/copilot/>. Acesso em: 16 de maio de 2024.

[9] GPT-3. Disponível em: <https://platform.openai.com/docs/models/gpt-3-5-turbo/>. Acesso em: 16 de maio de 2024

uso de ferramentas analíticas avançadas, aliada a uma estratégia orientada por métricas precisas, estabelece uma metodologia sólida para alcançar uma qualidade de software superior.

Na próxima seção, apresentaremos conceitos relacionados à construção da ferramenta *AI Code Analyzer,* além de explorar as tecnologias utilizadas em seu desenvolvimento.

3. AI CODE ANALYZER

A ferramenta AI Code Analyzer foi desenvolvida com o objetivo de auxiliar na adoção de melhores práticas de codificação, provando ser extremamente útil nos processos de revisão e análise de código. Para seu desenvolvimento, foram empregadas diversas tecnologias modernas e métodos de análise, descritos a seguir:

3.1 JavaScript e React

A interface da aplicação foi desenvolvida utilizando o *JavaScript*[10], juntamente com o framework *React*[11], assegurando uma experiência de usuário fluida e altamente responsiva. O React, conhecido por seu desempenho eficiente e abordagem baseada em componentes, facilita o gerenciamento do estado da aplicação e a renderização dinâmica de interfaces. Essa combinação otimiza tanto a performance quanto a usabilidade, garantindo uma navegação ágil e intuitiva.

3.2 Firebase

No desenvolvimento da aplicação, utilizamos componentes essenciais do *Firebase*[12] para garantir segurança e eficiência em tempo real. O *Firebase Authentication*[13] foi adotado para gerenciar todas as sessões dos usuários,

[10] JavaScript. Disponível em: <https://www.w3schools.com/jsrEF/default.asp/>. Acesso em: 16 de maio de 2024.

[11] JavaScript. Disponível em: <https://www.w3schools.com/jsrEF/default.asp/>. Acesso em: 16 de maio de 2024.

[12] Firebase. Disponível em: <https://firebase.google.com/docs?hl=pt-br/>
Acesso em: 16 de maio de 2024.

[13] Firebase Authentication. Disponível em: <https://firebase.google.com/docs/auth?hl=pt-br/>. Acesso em: 16 de maio de 2024.

assegurando que cada interação com a aplicação seja não apenas segura, mas também personalizada.

Por outro lado, o *Firebase Realtime Database*[14] desempenha um papel fundamental como banco de dados em nuvem da aplicação. Esse serviço foi utilizado para o armazenamento e gestão de dados salvos pelo usuário, proporcionando uma plataforma robusta para manipulação de dados em tempo real.

3.3 Google Gemini

O *Google Gemini*[15] representa um marco na evolução das inteligências artificiais multimodais generativas. Lançado em dezembro de 2023, Gemini é o sucessor dos avançados modelos LaMDA e PaLM 2, surgindo como uma resposta inovadora ao *GPT-4*[16] da OpenAI, *Gemini* se destaca por sua capacidade de processar e integrar diversos tipos de dados, como texto, áudio, imagem, vídeo e código. A acessibilidade do Gemini é ampliada por sua API dedicada, permitindo uma implementação eficiente e versátil em várias aplicações. Para este projeto, será utilizada a versão "gemini-1.0-pro" do modelo.

3.4 Integração com GitHub

A ferramenta *AI Code Analyzer* foi projetada para se integrar de forma eficiente com o GitHub[17], uma das plataformas de hospedagem de código-fonte mais utilizadas por desenvolvedores ao redor do mundo. Essa integração permite aos usuários analisar repositórios inteiros diretamente da interface do *AI Code Analyzer*, sem a necessidade de baixar manualmente os arquivos de código.

3.5 Análise de Métricas de Qualidade

O *AI Code Analyzer* utiliza a API do Google Gemini para integrar as funcionalidades de análise de código. Através desta integração, a ferramenta envia prompts à API, que contêm tanto o código quanto as métricas detalhadamente

---

[14] Firebase Realtime Database. Disponível em: <https://firebase.google.com/docs/database?hl=pt-br/>.. Acesso em: 16 de maio de 2024.

[15] Google Gemini. Disponível em: <https://ai.google.dev/gemini-api/docs?hl=pt-br/>. Acesso em: 16 de maio de 2024.

[16] GPT-4. Disponível em: <https://openai.com/index/gpt-4/>. Acesso em: 16 de maio de 2024.

[17] GitHub: Disponível em: <https://github.com/> . Acesso em: 16 de maio de 2024.

especificadas. Esta integração facilita uma avaliação detalhada e eficaz do código, permitindo que a ferramenta entregue insights valiosos baseados nas métricas requeridas.

As métricas especificadas no prompt abrangem aspectos cruciais do código, tais como estrutura e lógica, complexidade ciclomática, detecção de code smells, análise de acoplamento e aderência a padrões de design e melhores práticas. A inteligência artificial examina essas métricas da seguinte maneira:

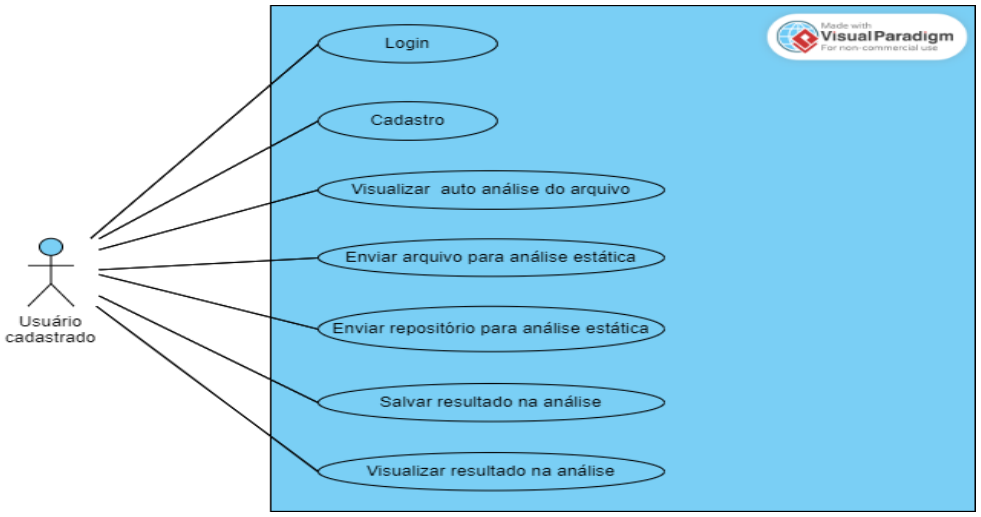
- **Estrutura e Lógica:** A IA avalia a organização do código, identificando os principais fluxos de controle e avaliando a clareza da estrutura lógica. Ela verifica se o código é estruturado de forma que facilite a compreensão e a manutenção, sugerindo melhorias na organização ou na documentação se necessário.
- **Complexidade Ciclomática:** A IA calcula a complexidade ciclomática, que mede o número de caminhos de execução possíveis no código. Um valor alto pode indicar que o código é difícil de testar e manter. Após o cálculo e a entrega dos resultados, fornece recomendações para simplificar o código, como decompor funções complexas em subfunções menores.
- **Code Smells:** A IA detecta padrões problemáticos no código que podem indicar uma necessidade de refatoração. Ela aponta esses code smells e sugere maneiras específicas de refinar o código para melhorar sua legibilidade, eficiência e manutenibilidade.
- **Acoplamento:** A análise de acoplamento é realizada para entender como diferentes módulos e componentes do código estão interligados. A IA avalia se o acoplamento é excessivamente rígido, o que pode comprometer a manutenibilidade, e sugere estratégias para alcançar um acoplamento mais solto e modular.
- **Padrões de Design e Melhores Práticas:** Esta métrica avalia a adesão do código aos padrões de design estabelecidos e às melhores práticas de codificação. A utilização adequada de padrões de design pode facilitar tanto o desenvolvimento quanto a manutenção do software. A ferramenta de IA revisa o uso de padrões como Singleton, Factory, Observer, entre outros, e verifica a adesão às melhores práticas, como a programação orientada a

objetos e princípios SOLID[18].

Após a análise das métricas descritas acima, a IA devolve os resultados em um formato estruturado, acompanhados de exemplos de código e sugestões detalhadas. Esses resultados ajudam os desenvolvedores a compreender as áreas que necessitam de melhoria e a implementar as mudanças sugeridas de maneira eficiente, elevando a qualidade geral do software.

O uso das tecnologias e métodos de análise descritos, aprimoraram as funcionalidades da ferramenta, resultando em um desenvolvimento otimizado e uma aplicação robusta e fácil de usar. A figura 1 mostra um diagrama de caso de uso, que ajuda a entender a estrutura e funcionamento do sistema.

Figura 1: Diagrama de caso de uso.



Fonte: Elaboração própria.

O diagrama descreve as funcionalidades do sistema AI Code Analyzer, incluindo login e cadastro do usuário, visualização e envio de arquivos e repositórios para análise, além de salvar e visualizar os resultados das análises.

No próximo capítulo, serão explorados os resultados e a utilização da ferramenta.

[18] SOLID Principles. Disponível em: <https://en.wikipedia.org/wiki/SOLID/>. Acesso em: 04 de junho de 2024.

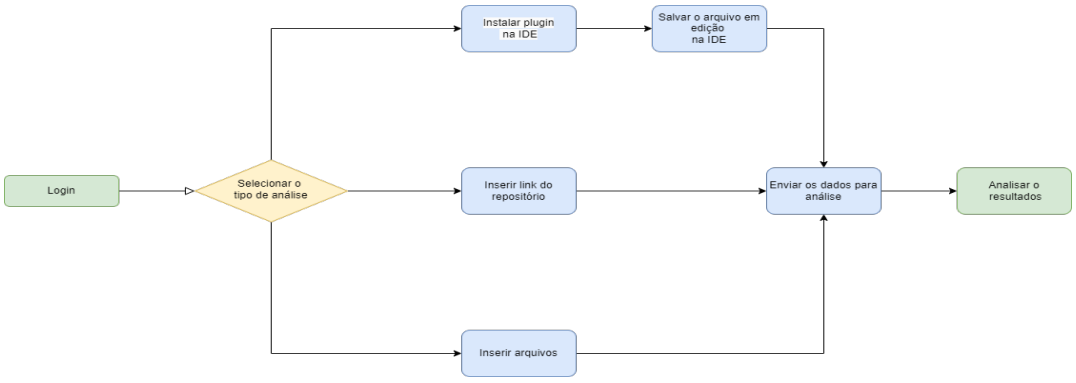
4. RESULTADOS E UTILIZAÇÃO

Este capítulo expõe os resultados alcançados e a aplicação prática da ferramenta *AI Code Analyzer*, que foi especialmente desenvolvida para integrar-se como um plugin no ambiente de desenvolvimento integrado (IDE) *Visual Studio Code*[19]. Essa integração permite a análise em tempo real do código à medida que é desenvolvido, além de oferecer a capacidade de avaliar arquivos de código inseridos manualmente.

Ao longo deste capítulo, o uso da ferramenta será detalhado em cenários reais de desenvolvimento de software. Serão apresentados casos específicos onde a implementação da *AI Code Analyzer* proporcionou melhorias notáveis e insights valiosos para a otimização da qualidade do código. Esses exemplos ilustram como a ferramenta contribuiu efetivamente para a elevação dos padrões de desenvolvimento e a correção de falhas de forma proativa, reforçando a importância de tecnologias de inteligência artificial na manutenção e na escrita de código limpo e eficiente.

A figura 2 irá demonstrar o fluxograma detalhado que descreve a jornada do usuário ao utilizar o *AI Code Analyzer* integrado ao Visual Studio Code. O diagrama ilustra desde a autenticação do usuário até a realização e revisão das análises de código, demonstrando as etapas de interação disponíveis na ferramenta.

Figura 2: Fluxograma de uso da ferramenta.



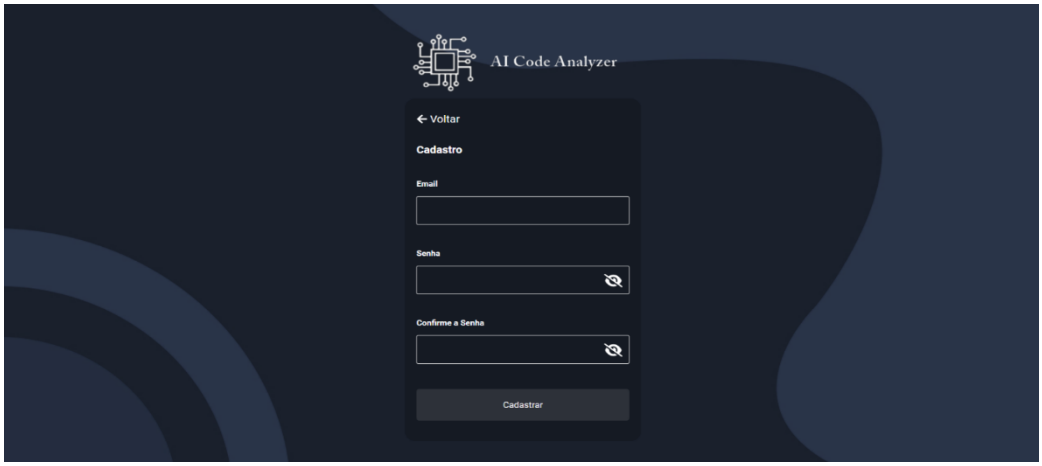
Fonte: Elaboração própria.

De acordo com o fluxograma contido na figura 2, para dar início a utilização do *AI Code Analyzer* e fazer o login, será necessário criar o cadastro pela própria

[19] Visual Studio Code : <https://code.visualstudio.com/docs>. Acesso em: 16 de maio de 2024.

plataforma informando um e-mail e senha, como demonstra a figura 3.

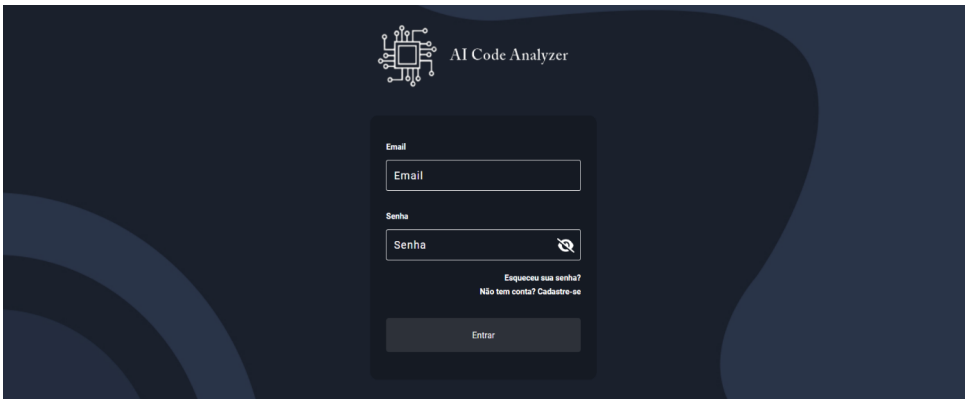
Figura 3: *AI Code Analyzer* – Cadastro.



Fonte. Elaboração própria.

Após o cadastro, o login poderá ser realizado na tela inicial informando o e-mail e senha cadastrado, como demonstra a figura 4.

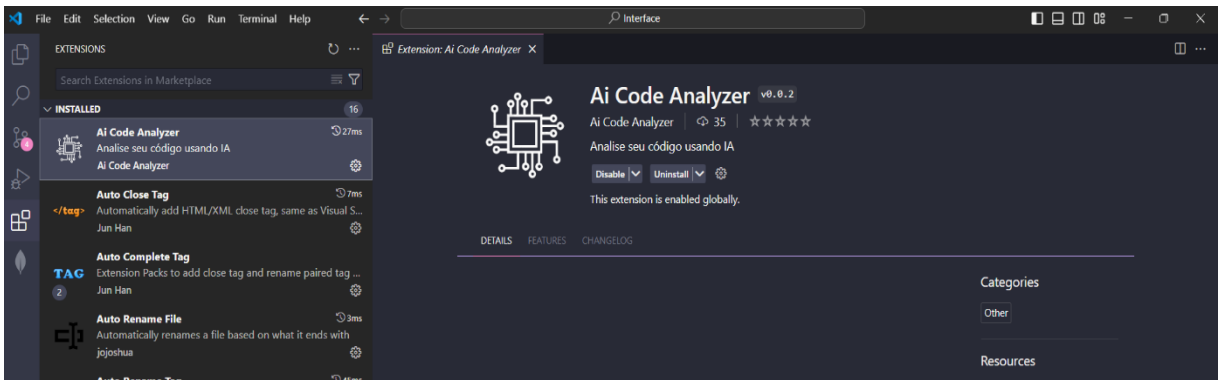
Figura 4: *AI Code Analyzer* - Login.



Fonte. Elaboração própria.

Após o login, para dar início a opção de autoanálise será necessário instalar o plugin desenvolvido e disponível para fazer a integração com a aplicação. O plugin está disponível diretamente no marketplace da IDE como demonstra a figura 5, facilitando o acesso e a instalação. Após a instalação, a IDE já irá se conectar à ferramenta automaticamente.

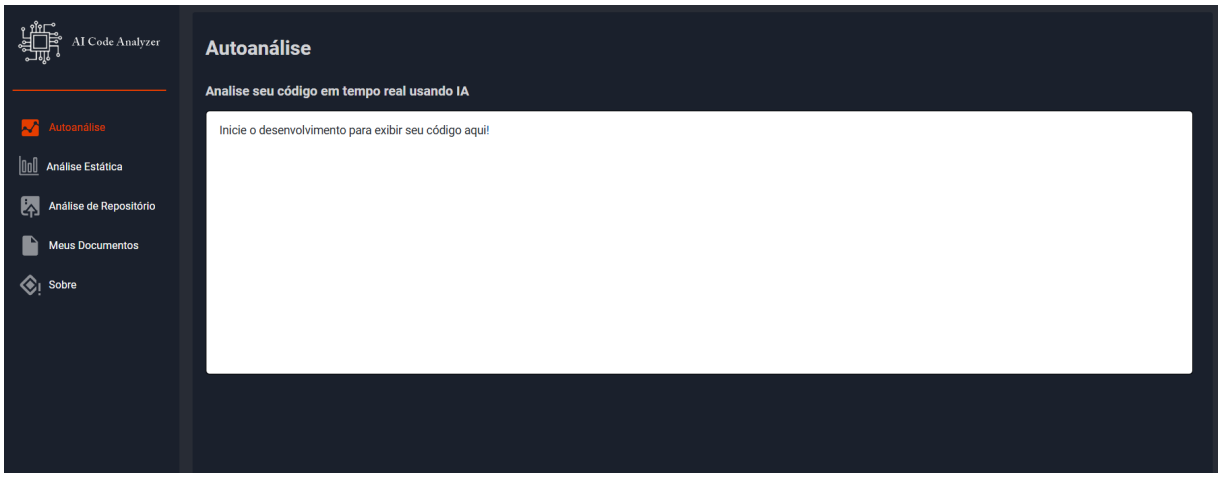
Figura 5: Plugin *AI Code Analyzer.*



Fonte. Elaboração própria.

Com o plugin já instalado e após o processo de autenticação seguiremos para tela inicial onde encontraremos as opções no menu lateral para a seleção do tipo de análise desejada. Quando selecionado o menu "Autoanálise", sempre que um arquivo é salvo na IDE, a análise de código é acionada automaticamente, garantindo uma verificação contínua e em tempo real; A Figura 6 demonstra a tela inicial e o menu de autoanálise.

Figura 6: Tela inicial – Autoanálise.



Fonte. Elaboração própria.

Para demonstrar a usabilidade da análise de código presente na ferramenta, utilizaremos como exemplo o trecho que código na linguagem Java[20] demonstrado na figura 7. O código Java apresentado define uma calculadora simples com operações

[20] Java. Disponível em: <https://docs.oracle.com/javase/tutorial/index.html/>. Acesso em: 16 de maio de 2024.

de adição, subtração, multiplicação e divisão. A classe `Calculator` possui um método `calculate` que usa um `switch` para realizar a operação especificada pelo usuário. O programa principal permite múltiplas operações em um loop, lendo entradas do usuário e exibindo os resultados. Ele também verifica divisão por zero e operações inválidas, exibindo mensagens de erro apropriadas. O loop continua até que o usuário opte por sair.

Figura 7: Código java.

```java
import java.util.Scanner;

public class Calculator {

    public int calculate(int a, int b, String operation) {
        switch (operation) {
            case "add":
                return a + b;
            case "subtract":
                return a - b;
            case "multiply":
                return a * b;
            case "divide":
                if (b == 0) {
                    System.out.println("Cannot divide by zero!");
                    return 0;
                } else {
                    return a / b;
                }
            default:
                System.out.println("Invalid operation!");
                return 0;
        }
    }


    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Calculator calculator = new Calculator();

        while (true) {
            System.out.println("Enter first number:");
            int a = scanner.nextInt();
            System.out.println("Enter second number:");
            int b = scanner.nextInt();
            System.out.println("Enter operation (add, subtract, multiply, divide):");
            String operation = scanner.next();

            int result = calculator.calculate(a, b, operation);
            System.out.println("Result: " + result);

            System.out.println("Do you want to continue? (yes/no)");
            if (scanner.next().equals("no")) {
                break;
            }
        }
        scanner.close();
    }
}
```

Fonte. Elaboração própria.

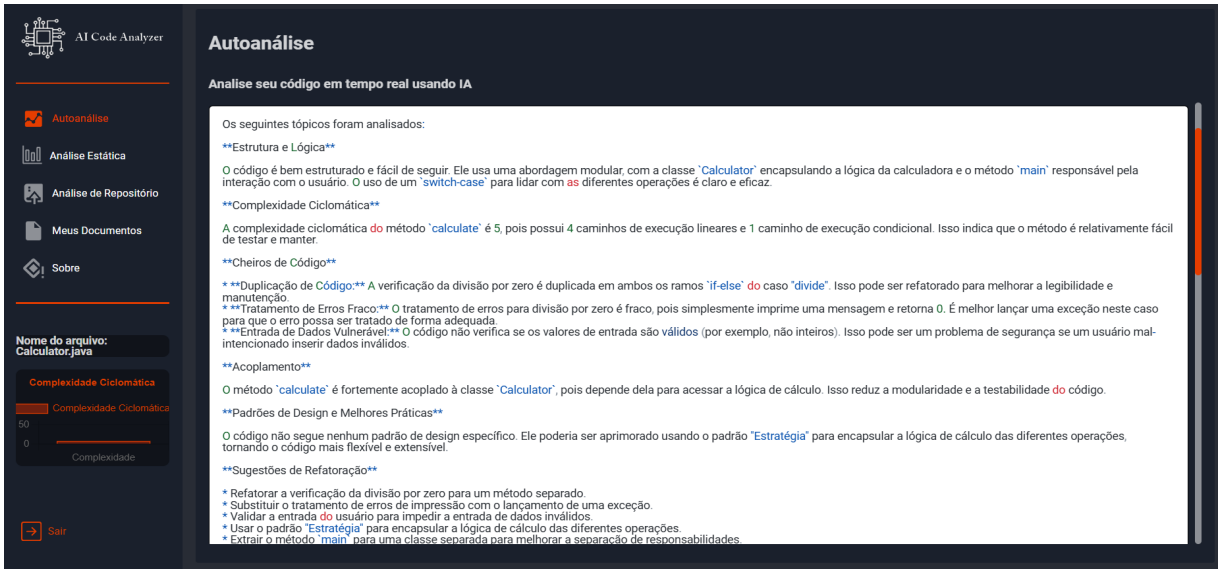
Para calcular a “Complexidade Ciclomática” do código apresentado, utilizamos a fórmula de McCabe (1976), M = E - N + 2P, onde "E" representa o número de arestas no grafo de fluxo de controle, "N" é o número de nós, e "P" refere-se ao número de componentes conexos (usualmente 1 para a maioria dos programas). No método 'calculate', temos um nó de entrada, seis nós de decisão (cada um correspondente a um 'case' e o 'default'), e um nó de saída, totalizando 8 nós. O método principal adiciona mais 4 nós, alcançando um total de 12 nós. Com 15 arestas, a “Complexidade

Ciclomática” do código é 5.

A seguir, utilizaremos a ferramenta *AI Code Analyzer* para realizar uma autoanálise do código, permitindo uma comparação direta dos resultados obtidos com a análise de “Complexidade Ciclomática” feita pela ferramenta e os resultados do método manual demonstrado anteriormente.

Para um código, após a realização das edições e salvamento do arquivo de código na IDE, o AI Code Analyzer procede de forma autônoma com a análise do código. Esta operação envolve o uso de modelos de inteligência artificial para vasculhar o código em busca de padrões e métricas pré-definidas no prompt[21]. Concluída a análise, a ferramenta exibe os resultados em uma interface intuitiva e detalhada, como ilustrado na Figura 8.

Figura 8: Resultado da análise de código - Gemini.



Fonte. Elaboração própria.

Na Figura 8, observa-se o relatório completo da análise de código fornecido pelo *AI Code Analyzer*, incluindo os resultados da análise de “Complexidade Ciclomática”, que indicam um valor de 5. Esse resultado confirma a eficácia da metodologia de análise adotada pela ferramenta. A interface do *AI Code Analyzer* também estrutura as informações em tópicos distintos, abrangendo desde a estrutura

[21] Prompt.Disponível em: <https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results/>. Acesso em: 16 de maio de 2024.

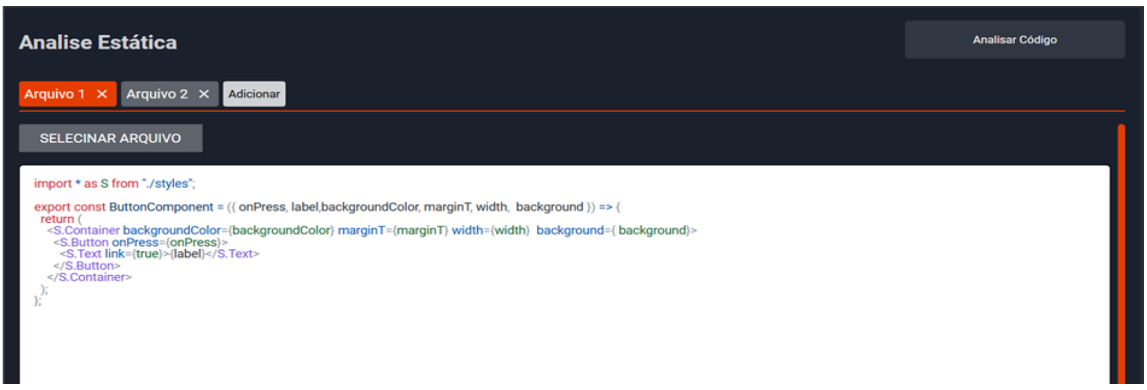
e lógica do código até a “Complexidade Ciclomática”, identificação de 'code smells', acoplamento, padrões de design e práticas recomendadas. Cada seção do relatório proporciona uma análise detalhada e crítica, destacando áreas específicas para melhoria e oferecendo insights valiosos para a otimização do código.

Na interface, um destaque especial é dado para a barra lateral esquerda, que exibe um indicador gráfico referentes à “Complexidade Ciclomática”. Esse indicador permite uma rápida compreensão visual do nível de complexidade, facilitando a interpretação dos dados pelo usuário. Isso possibilita uma tomada de decisão mais rápida e fundamentada sobre potenciais ações de refatoração ou revisão. Além disso, os resultados mostram que o cálculo da “Complexidade Ciclomática”, realizado pela IA é consistente com o cálculo manual previamente feito, comprovando a precisão e utilidade da ferramenta.

A análise estática, conforme ilustrado nas Figuras 9 e 10, resultará em insights semelhantes aos da autoanálise, tendo como diferencial a capacidade de processar múltiplos arquivos simultaneamente e salvar o resultado da análise. Esta abordagem amplia o escopo de análise e oferece uma visão mais abrangente dos dados.

Utilizaremos como exemplo o código de um componente de botão construído em JavaScript usando o React e Styled-components[22], onde as propriedades como cor de fundo, margem e largura são passadas e aplicadas dinamicamente.

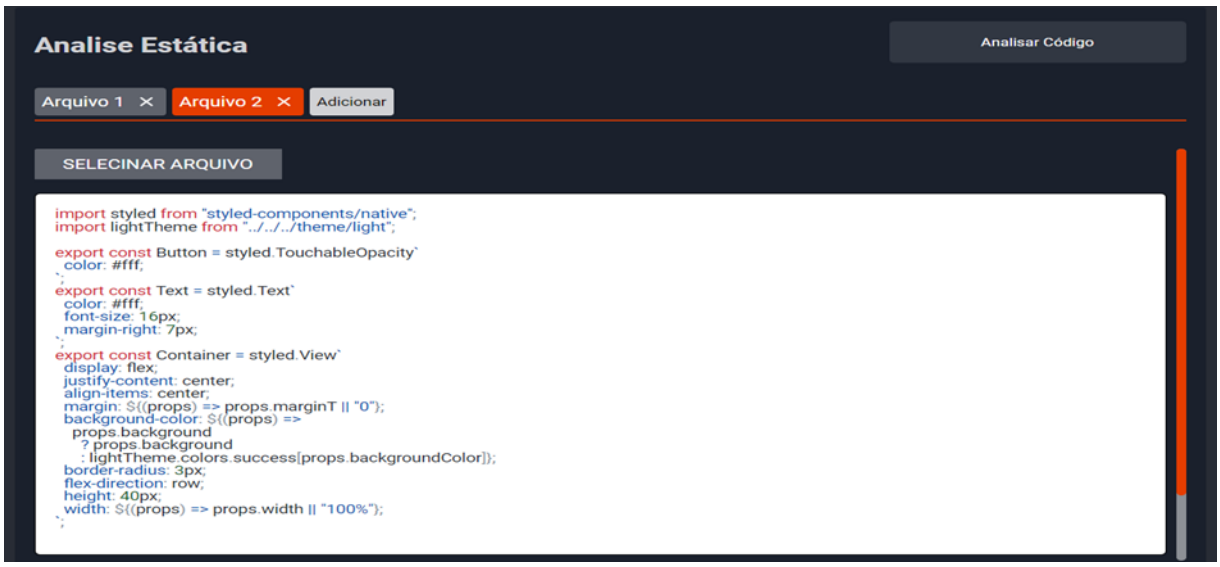
Figura 9: Exemplo de código em React.



Fonte: Elaboração própria.

Figura 10: Exemplo de código em React.

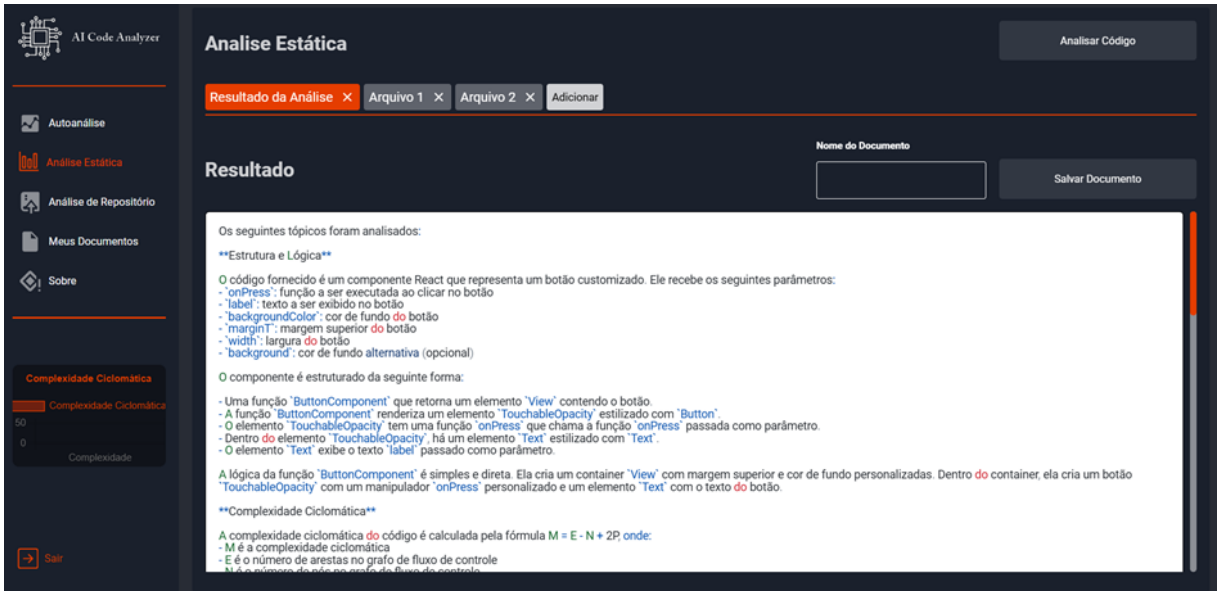
[22] Styled-components. Disponível em <https://styled-components.com/docs/>. Acesso em: 16 de maio de 2024.



Fonte: Elaboração própria.

Após a inserção dos arquivos, ao selecionar a opção “Analisar Código”, os dados serão processados pela inteligência artificial. Os resultados da análise serão então exibidos conforme ilustrado na Figura 11.

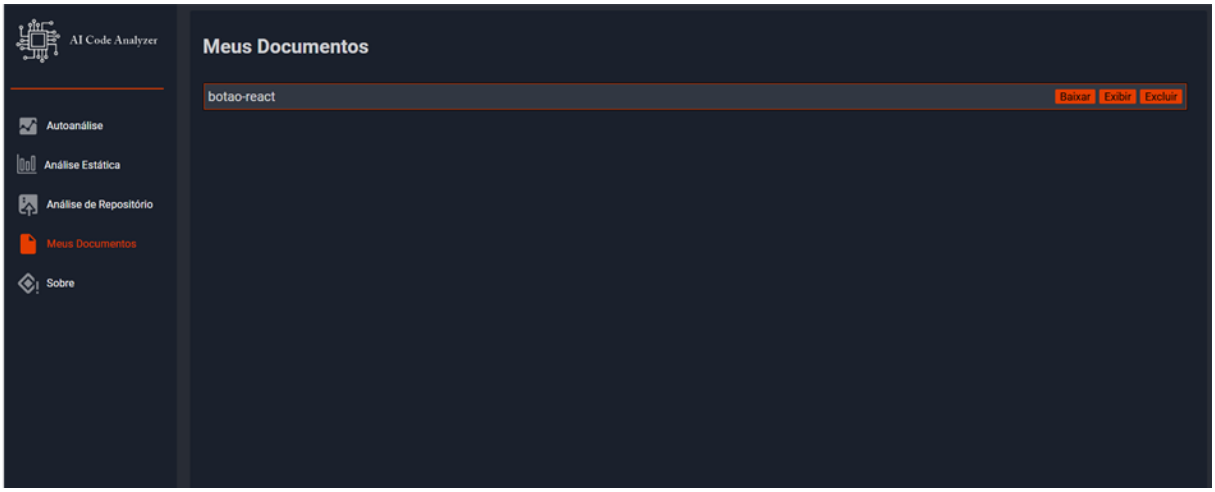
Figura 11: Resultados da análise estática.



Fonte: Elaboração própria.

Também é possível salvar o resultado gerado, que ficará disponível na aba "Meus Documentos". Nesta seção, os usuários podem visualizar, excluir ou baixar o arquivo em formato PDF. A Figura 12 demonstrará a sessão "Meus Documentos".

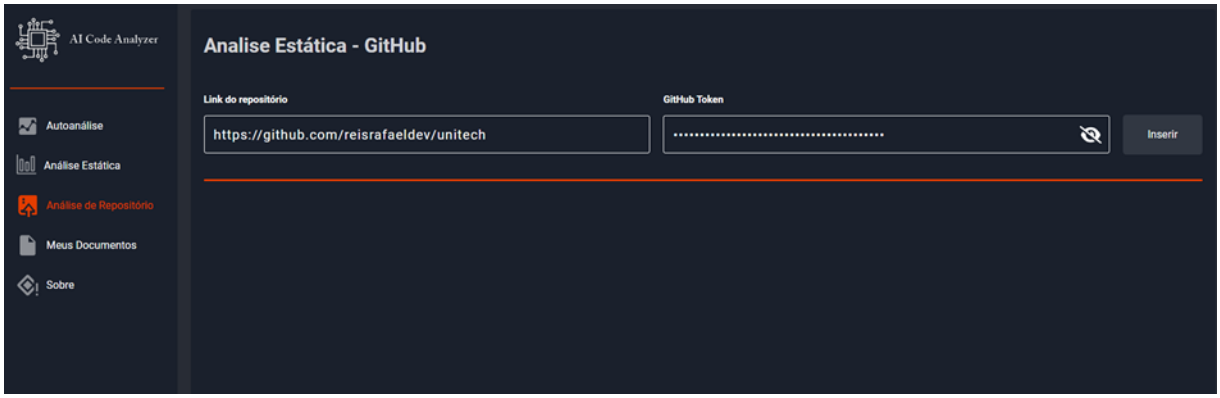
Figura 12: Meus Documentos.



Fonte: Elaboração própria.

Por fim, a ferramenta *AI Code Analyzer* introduz uma funcionalidade avançada que permite a análise direta de repositórios hospedados no GitHub. Para utilizar essa capacidade, os usuários devem fornecer o URL do repositório desejado e um token de acesso pessoal. Este token garante uma interação segura e autorizada com o repositório, conforme demonstrado na figura 13.

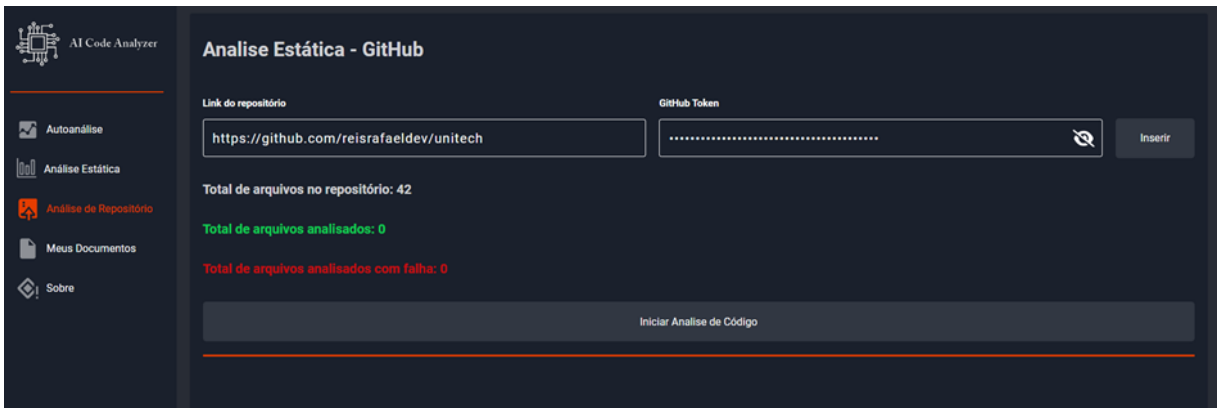
Figura 13: Credenciais GitHub.



Fonte: Elaboração própria.

Adiante, após inserir as credenciais corretamente, conforme demonstrado na figura 14, será exibida uma legenda indicando o total de arquivos no repositório, o número de arquivos analisados e o total de arquivos com falhas na análise.

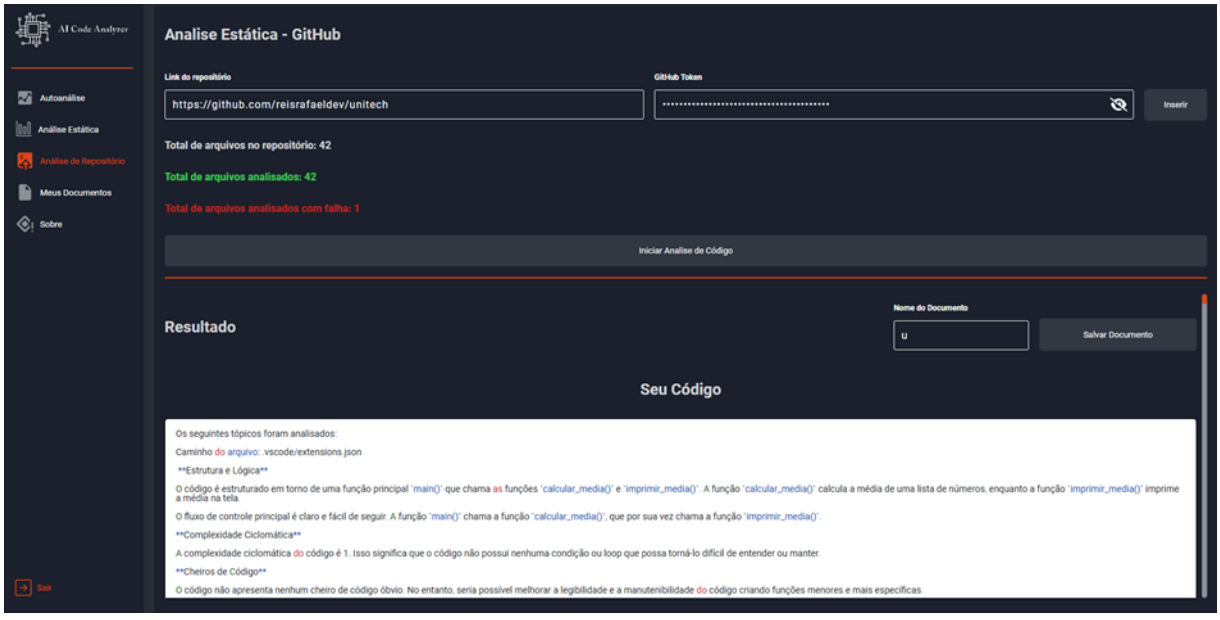
Figura 14: Dados iniciais do repositório.



Fonte: Elaboração própria.

Ao clicar em “Iniciar Análise de Código”, as informações na legenda sobre o total de arquivos no repositório e o número de arquivos analisados serão atualizadas dinamicamente à medida que a análise avança. Uma vez concluída a análise de todo o repositório, os resultados serão exibidos conforme ilustrado na figura 15. Além disso, esses resultados poderão ser armazenados na seção “Meus Documentos”, onde os usuários têm a opção de visualizá-los novamente, excluí-los ou salvá-los em formato PDF no dispositivo local.

Figura 15: Resultado da análise de repositório.



Fonte: Elaboração própria.

O código-fonte da ferramenta pode ser encontrado no seguinte endereço do GitHub: https://github.com/reisrafaeldev/ai-code-analyzer.git. Para utilizar a ferramenta, acesse: https://aicode-analyzer.vercel.app. Além disso, um vídeo de apresentação está disponível pelo link: https://youtu.be/ArkJmBp9YOY.

No próximo capítulo abordaremos as considerações finais e conclusão deste trabalho.

5. CONSIDERAÇÕES FINAIS

Este estudo confirmou a eficácia da integração da inteligência artificial na análise da qualidade do código-fonte, evidenciada pelo desenvolvimento e aplicação da ferramenta *AI Code Analyzer*. A capacidade desta tecnologia de identificar padrões e sugerir melhorias proativamente revoluciona o processo de desenvolvimento de software, facilitando uma manutenção mais ágil e promovendo a melhoria contínua da qualidade do código.

A implementação de ferramentas baseadas em IA, como o *AI Code Analyzer*, transcende a simples otimização dos processos de desenvolvimento; ela eleva o padrão de qualidade do software produzido. As métricas de qualidade exploradas revelam como a ferramenta se integra de maneira eficaz aos ambientes de desenvolvimento, proporcionando economia significativa de recursos e otimização do tempo de desenvolvimento.

Olhando para o futuro, é crucial que continuemos a evoluir essas tecnologias. O desenvolvimento de ferramentas mais avançadas e a exploração de novas métricas e metodologias são essenciais para assegurar a relevância e eficiência da análise de código em um ambiente de software em constante mudança. As experiências acumuladas e os êxitos obtidos com o *AI Code Analyzer* formam uma base sólida para futuros avanços no campo da engenharia de software.

Em conclusão, a análise de código com IA representa não apenas uma ferramenta técnica, mas uma estratégia fundamental que suporta a sustentabilidade e adaptabilidade a longo prazo dos softwares no competitivo mercado atual. Assim, é fortemente recomendada a adoção dessas tecnologias avançadas para não apenas satisfazer as demandas contemporâneas, mas também para nos preparar de forma mais eficiente para os desafios futuros.

Referências

McCONNELL, S. **Code Complete**: A Practical Handbook of Software Construction. 2ª ed. Redmond: Microsoft Press, 2004.

FITZGERALD, B.; STOL, K-J**. Software development practices and quality**: trends

and implications. *Journal of Software Engineering*, v. 36, n. 3, p. 456-470, 2020.

CAPRETZ, L. F.; AHMED, F. **Modular code architecture**: Necessity in modern software development. *International Journal of Software and Systems Modeling*, v. 25, n. 2, p. 335-350, 2021.

ZHENG, Y.; ZHOU, M. **Challenges in low-quality software projects**: An industrial survey. *Software Quality Journal*, v. 30, n. 1, p. 123-142, 2022.

MCCABE, Thomas J. A complexity measure. **IEEE Transactions on software Engineering**, n. 4, p. 308-320, 1976. Disponível em: <https://ieeexplore.ieee.org/abstract/document/1702388>. Acesso em: 15 de maio de 2024.

CHIDAMBER, Shyam R.; DARCY, David P.; KEMERER, Chris F. Managerial use of metrics for object-oriented software: An exploratory analysis. **IEEE Transactions on software Engineering**, v. 24, n. 8, p. 629-639, 1998. Disponível em: <https://ieeexplore.ieee.org/abstract/document/707698/>. Acesso em: 15 de maio de 2024.

BECK, Kent; FOWLER, Martin; BECK, Grandma. Bad smells in code. **Refactoring: Improving the design of existing code**, v. 1, n. 1999, p. 75-88, 1999. Disponível em: <http://www-public.tem-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/BeckFowler99.pdf>. Acesso em: 15 de maio de 2024.

VICENTE, Jonas Antônio Gomes; SIRQUEIRA, Tassio Ferenzini Martins. Uma Ferramenta de Análise Estática de Código Fonte para Aplicações Web PHP. **Caderno de Estudos em Sistemas de Informação**, v. 7, n. 1, 2022. Disponível em: <https://seer.uniacademia.edu.br/index.php/cesi/article/view/2544>. Acesso em: 15 de maio de 2024.

HU, Xing et al. Deep code comment generation. In: **Proceedings of the 26th conference on program comprehension**. 2018. p. 200-210. Disponível em: <https://dl.acm.org/doi/abs/10.1145/3196321.3196334>. Acesso em: 15 de maio de 2024.

BOURQUE, Pierre; FAIRLEY, RJNICS. Swebok. **Nd: IEEE Computer society**, 2004. Disponível em: <http://publicationslist.org/data/p.bourque/ref-796/915.pdf/>. Acesso em: 15 de maio de 2024.