



Automatizando a documentação de aplicações

Trabalho de Conclusão de Curso

Caio Souza dos Reis

Antonio Carlos Souza

Instituto Federal da Bahia – IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador

Salvador, Bahia, Brasil
Novembro 2023

SUMÁRIO

1. Visão Geral

1.1. Declaração do Problema

1.2 Proposta de Solução de Software

1.3 Tecnologias Adotadas

1.3.1 Typescript

1.3.2 VSCode

1.3.3 Langchain

1.4 Trabalhos Relacionados

2. Requisitos

3. Design

3.1 Diagramas

3.1.1 De Classe

3.1.2. De casos de Uso

3.2 Visão Arquitetural

4. Testes de Software

4.1. Projeto de Testes

5. Implantação

5.1. Projeto de Implantação

6. Manual do Usuário

6.1 Instalação

6.2 Comandos

6.3 Configurações

Referências

Apêndices

Glossário, Siglas e Abreviações

1. Visão Geral

1.1. Declaração do Problema

O presente trabalho visa reduzir a quantidade de tempo investido na compreensão do código fonte de aplicações e na elaboração de documentação.

É comum que se gaste muito tempo lendo o código de uma aplicação quando há a necessidade de atualizá-la. O código já existente, muitas vezes escrito por outras pessoas ou há muito tempo atrás, precisa ser compreendido para que possa ser alterado. O tempo gasto nessa compreensão pode ser reduzido com uma boa documentação do projeto que seja capaz de explicar o que está sendo feito. O problema maior é que elaborar essa documentação e mantê-la atualizada demanda muito tempo.

A situação descrita acima, em que se gasta muito tempo para compreender um projeto já foi vivida pelo autor inúmeras vezes, seja por conta da pouca legibilidade de um código legado, escrito há muitos anos, seja porque se trata de um projeto elaborado em uma linguagem desconhecida. Uma das formas de minimizar esse problema surgiu com a popularização dos grandes modelos de linguagem (LLM - Large Language Models) que permitiram a geração automática de documentação capaz de explicar de forma didática, simples e em texto corrido, o fluxo lógico de um código. Isso tem potencial para facilitar e agilizar o trabalho de muitos engenheiros de software na compreensão e posterior produção de código.

1.2 Proposta de Solução de Software

Primeiramente utilizaremos no programa modelos de inteligência artificial já prontos e treinados, capazes de compreender e explicar código. Esses modelos serão acessados através de uma chamada a API da Openai utilizando comandos de uma extensão do editor de código VSCode.

Essa extensão é o programa elaborado como trabalho de conclusão de curso que será apresentado neste documento, e tem como nome Doc4me.

Doc4me tem a responsabilidade de passar parâmetros, prompts, arquivos e demais configurações para o modelo LLM, para então agregar as respostas do modelo e gerar arquivos em formato markdown com explicações dos códigos do projeto. Será realizada também a composição dos markdown gerados de modo a criar um arquivo maior resumindo e explicando todo o projeto.

Desse modo, o programador que precisa compreender melhor uma aplicação, basta executar a extensão e ler as explicações e resumo gerados pela inteligência artificial para ter uma noção sobre todo o projeto e dessa forma ter seu processo de leitura do código facilitado.

Além da explicação dos arquivos e do projeto, há funcionalidades complementares como a possibilidade de se fazer perguntas em linguagem natural a um arquivo específico do projeto e a possibilidade de calcular o valor em dólar das chamadas realizadas. Esse cálculo do custo em dólar das chamadas pode ser útil, principalmente quando estamos usando pela primeira vez a aplicação e estamos querendo estimar o custo de explicar o projeto todo, ou seja, pode-se gerar a documentação de apenas um diretório ou arquivo e calcular quanto custou esse processamento pontual, e com base nesse valor estimar o custo de executar a extensão para múltiplos diretórios.

1.3 Tecnologias Adotadas

1.3.1 Typescript

Só é possível escrever extensões para VSCode com Typescript ou Javascript, dessa forma o presente trabalho adotou Typescript por ser a linguagem sugerida pela documentação e apresentar as garantias de uma linguagem de tipagem estática. Além disso, a documentação para construção de extensões disponível para Typescript é mais completa do que a documentação correspondente para Javascript.

Typescript é uma linguagem cuja principal diferença para o Javascript é a possibilidade de declarar tipos para cada variável. O código escrito nessa linguagem então é transpilado para Javascript que aí sim passa a ser interpretado pelo navegador no frontend ou pelo motor de execução no backend (com node, deno, bun). É possível configurar o transpilador responsável por esse processo para converter o código de origem em versões diferentes da linguagem Javascript.

Por último, editores de código como VSCode conseguem ler as configurações do transpilador e avisar quaisquer incongruências no código durante a escrita dele. Isso ajuda o desenvolvedor a evitar erros que só seriam expostos em tempo de execução.

1.3.2 VSCode

O VSCode é um programa desenvolvido pela Microsoft e que desde 2018 se mostrou o editor de código mais utilizado pelos desenvolvedores de software. Sendo utilizado por 74% dos profissionais que responderam à pesquisa do StackOverflow no ano de 2022 (“Visual Studio Code”, 2023).

Dada tamanha popularidade, aparentou ser o melhor editor de códigos a ser escolhido para o desenvolvimento de uma extensão/plugin. Outro motivo que também auxiliou na escolha do VSCode como IDE a ser estendida com o programa apresentado foi a cuidadosa documentação voltada em auxiliar o desenvolvimento de extensões. Durante a elaboração do trabalho verificou-se que a documentação oficial possuía informações insuficientes, mas que foram encontradas em fóruns de dúvidas online.

O VSCode permite não só a escrita de códigos, como também a sua execução, depuração e leitura. A escrita é extremamente facilitada com funcionalidades conhecidas como autocompletar em que são exibidas sugestões de código que respeitam o contexto do programa. A leitura do que está sendo exibido na tela é facilitada pelo uso dos destaques de sintaxe (*syntax highlight*) que palavras são coloridas de acordo com a funcionalidade empregada no contexto da linguagem. A depuração permite o controle do ponto de execução e suspensão do programa, além da exibição dos valores das variáveis presentes em cada escopo.

1.3.3 Langchain

Langchain é uma biblioteca originalmente desenvolvida para Python, mas que possui recente versão para Javascript e Typescript.

Seu uso visou facilitar as interações com os modelos de inteligência artificial da Openai, principalmente com relação a formação de prompts de comando para o modelo e o encadeamento sucessivo de chamadas.

O encadeamento sucessivo de chamadas, que inclusive ajuda a dar nome à biblioteca, foi utilizado na hora de gerar explicações para arquivos de código com tamanhos maiores, compostos por uma quantidade de tokens acima do limite permitido para processamento pelo modelo de maior custo benefício, o gpt-3.5-turbo. Nesses casos, de arquivos grandes, foi realizada a quebra do arquivo em partes menores, que por sua vez são resumidas pelo modelo de inteligência artificial para só depois serem explicadas. Esse processamento em duas etapas é simplificado pela biblioteca e permite a geração de documentação para arquivos de textos de todos os tamanhos.

1.3.4 LLM - Large Language Model

Tecnologia utilizada para descrever a inteligência artificial criada a partir de grandes volumes de textos e capaz de responder ou complementar texto com base na compreensão gerada a partir dos textos utilizados no treinamento do modelo.

1.4 Trabalhos Relacionados

Algumas soluções similares a que está sendo proposta já estão disponíveis no mercado, mas cada uma delas possui pelo menos um limitador, de forma que não foi possível encontrar nenhuma que atendesse todos os requisitos e funcionalidades abaixo que são oferecidas pela presente solução:

- possibilidade de escolha do modelo de inteligência artificial a ser utilizado
- código fonte aberto
- capacidade de resumir todo o projeto em apenas um arquivo
- capacidade de documentar todos os arquivos de forma separada
- possibilidade de personalizar os prompts de comando
- possibilidade de personalizar quais linguagens podem ser analisadas
- possibilidade de personalizar quais diretórios podem ser ignorados
- possibilidade de personalizar o nome do diretório de destino contendo as documentações geradas
- possibilidade de explicar arquivos de apenas um diretório escolhido
- possibilidade de explicar apenas um arquivo
- possibilidade de responder qualquer pergunta sobre um arquivo específico
- possibilidade de calcular o custo das chamadas realizadas anteriormente
- possibilidade de visualizar a quantidade de tokens de cada arquivo do projeto e de tokens utilizados na explicação desses arquivos
- possibilidade de explicar o código do projeto em qualquer linguagem humana, como português ou inglês.

1.4.1 Github Copilot

[\[https://github.com/features/copilot\]](https://github.com/features/copilot)

Uma das mais famosas aplicações de inteligência artificial para programadores é o Copilot do Github. Infelizmente, ele não possui a funcionalidade de gerar documentação de forma automática. O que ele oferece são serviços de autocomplete e chat.

1.4.2 Docuwriter

[\[https://www.docuwriter.ai/\]](https://www.docuwriter.ai/)

O Docuwriter, apesar de possuir a funcionalidade para gerar documentação de arquivo de código, o preço cobrado por eles é proibitivo para muita gente. Na data da escrita deste trabalho (sete de novembro de 2023), o valor está em vinte e nove dólares por mês. Além disso, possui um limite na quantidade de arquivos gerados por mês.

1.4.3 Docify

[\[https://docify.ai4code.io/\]](https://docify.ai4code.io/)

O Docfy não gera documentação para um arquivo inteiro, mas gera documentação para cada função ou método presente no código em forma de docstrings. Isso limita a compreensão das relações entre as funções. Além disso, também possui uma mensalidade de alto custo no valor de 29 dólares por mês, assim como o Docuwriter.

2. Requisitos

Ao projetar e desenvolver um sistema, é imperativo compreender e definir claramente os requisitos que nortearão o processo. Esta seção aborda os requisitos funcionais e não funcionais do sistema, delineando as características específicas que o tornarão robusto, eficiente e alinhado às necessidades dos usuários.

2.1 Requisitos Funcionais

Requisitos funcionais são características e funcionalidades específicas que o sistema deve oferecer para atender às necessidades do usuário. Esses requisitos descrevem as ações que o sistema deve realizar.

- Como programador preciso ser capaz de executar um comando no editor de código e ter a documentação do meu projeto gerada automaticamente.
- Como programador desejo que a documentação do meu código seja em texto corrido contendo explicações detalhadas e compreensíveis de cada arquivo do projeto.
- Como programador desejo ter acesso a um arquivo só, contendo uma explicação de todo o projeto na linguagem de minha escolha, como português ou inglês.
- Como programador gostaria de poder fazer qualquer pergunta sobre o conteúdo de um arquivo do meu projeto.
- Como programador gostaria de poder calcular o custo financeiro das chamadas realizadas pela aplicação.
- Como programador gostaria de poder cadastrar minha chave da OpenAi para utilizar o serviço deles através da extensão
- Como programador gostaria de poder personalizar os prompts a serem enviados para a inteligência artificial
- Como programador gostaria de poder escolher o diretório de destino que armazenará os arquivos gerados pela extensão
- Como programador gostaria de adicionar e remover os diretórios que não devem ter seus arquivos explicados pelo programa
- Como programador gostaria de especificar quais linguagens de programação desejo que tenham seus códigos explicados.

2.2 Requisitos Não-Funcionais

Requisitos não funcionais são critérios que descrevem as características do sistema que não estão relacionadas diretamente às funcionalidades específicas, mas sim a características mais amplas de seu desenvolvimento, operação e tecnologia.

[RNF1] O programa deve ser escrito em Typescript.

[RNF2] O programa deve ser executado como extensão do VSCode.

[RNF3] O programa deve utilizar a biblioteca Langchain para facilitar a construção de prompts e encadeamento de chamadas em arquivos grandes.

[RNF4] O programa deve ser testado com testes automatizados capazes de garantir funcionamento e modularização.

3. Design

A etapa de design é crucial para a criação de sistemas robustos e eficientes. Nesta seção, abordaremos o uso do Unified Modeling Language (UML), uma linguagem padrão para modelagem de sistemas orientados a objetos. Desenvolvida para unificar métodos de modelagem, a UML proporciona uma notação gráfica consistente e abrangente, permitindo a representação precisa de diversos aspectos de um sistema de software.

3.1 Projeto UML

O Unified Modeling Language (UML) é uma linguagem padrão para modelagem de sistemas orientados a objetos. Desenvolvida para unificar diversos métodos de modelagem, a UML fornece uma notação gráfica consistente e abrangente para representar diferentes aspectos de um sistema de software. Os diagramas UML são ferramentas visuais que ajudam a comunicar, visualizar, especificar, construir e documentar os aspectos de um sistema.

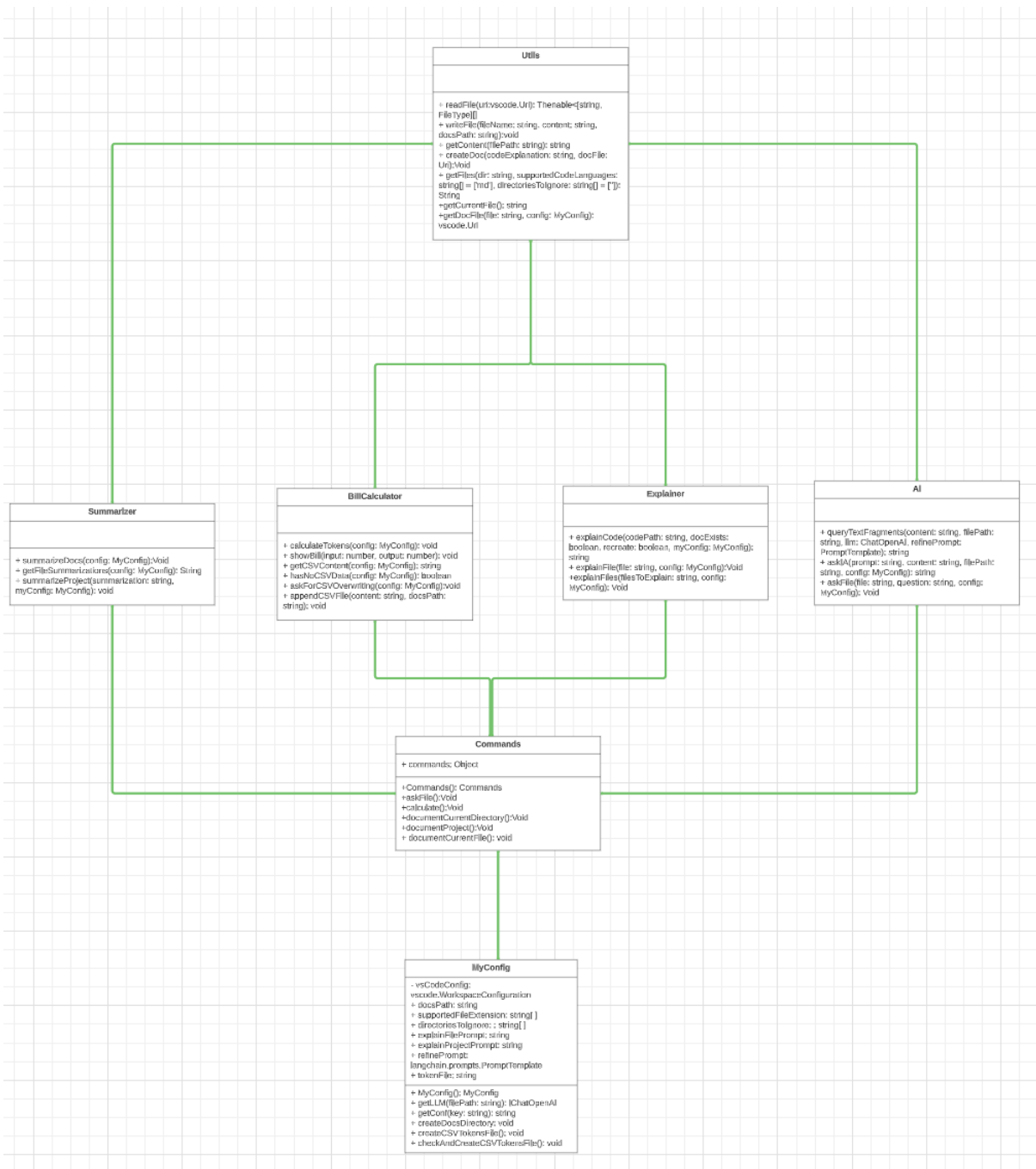
Existem vários tipos de diagramas UML, cada um com um propósito específico. A seguir será utilizado o diagrama de classe e o de casos de uso para descrever o projeto.

3.1.1 Diagrama de Classe

Diagrama de classe é uma representação visual da estrutura estática e das relações entre as classes de um sistema orientado a objetos. Ele ilustra as classes no sistema, seus atributos, métodos e as associações entre elas. As classes são representadas por retângulos, e as associações entre as classes são mostradas por linhas que conectam os retângulos. Além disso, o diagrama de classe pode incluir informações sobre herança, agregação, composição e outras relações entre as classes. Este tipo de diagrama é fundamental para entender a organização das classes em um sistema e é amplamente utilizado durante a fase de design de software.

https://lucid.app/lucidchart/6b14d05e-0b92-4c8c-95b7-ee6c365efd9f/edit?viewport_loc=-1536%2C-4201%2C5714%2C2890%2C0_0&invitationId=inv_8f9082be-93ce-4d61-b43c-efd130da2410

Figura 1



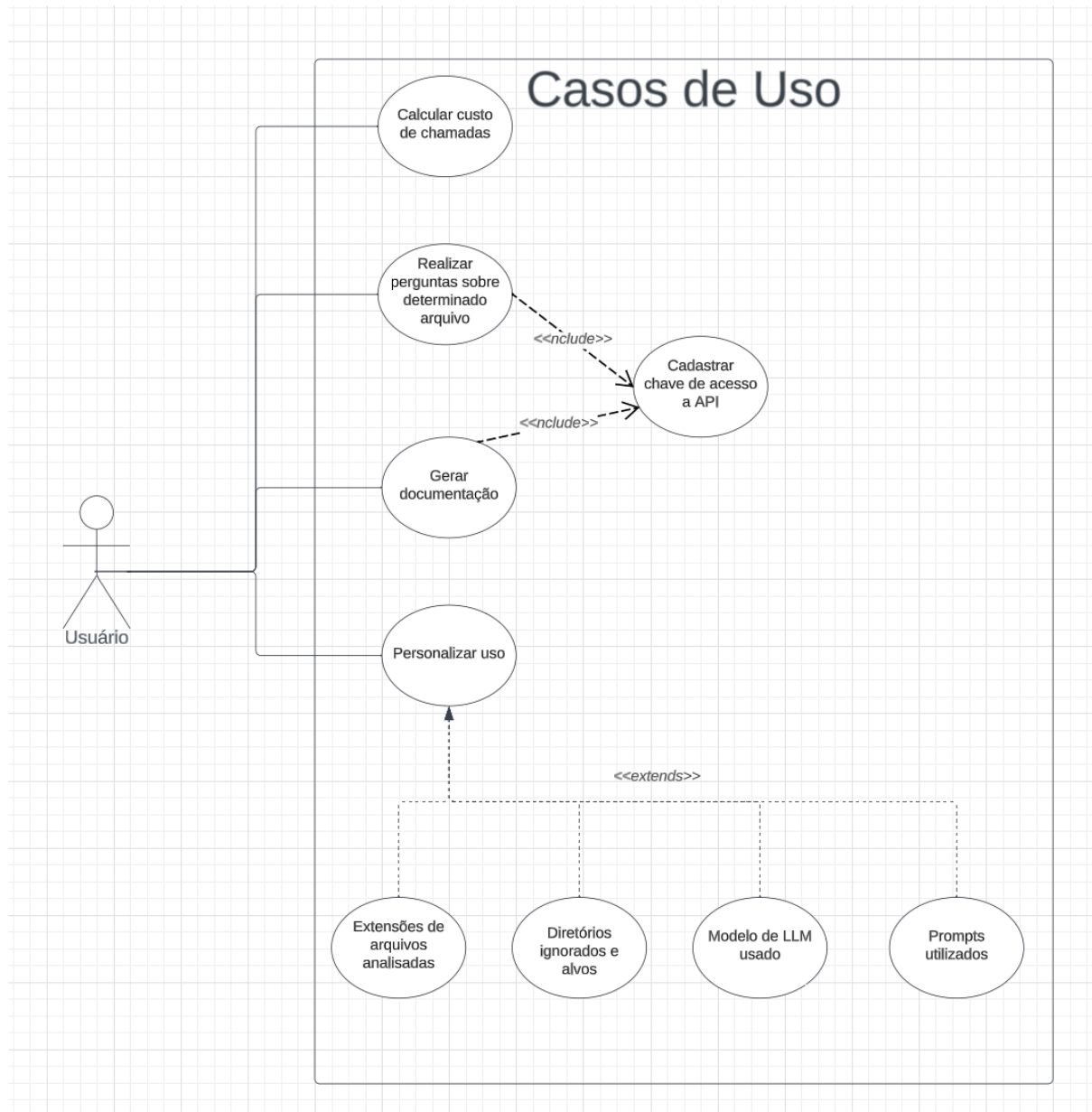
Fonte: Autor

3.1.2. Diagrama de Casos de Uso

Diagrama de casos de uso é uma representação gráfica que descreve as interações entre um sistema e seus atores externos (usuários, outros sistemas, etc.) em termos de cenários de uso. Ele mostra como os usuários interagem com o sistema para realizar determinadas funcionalidades. Os atores são representados por ícones externos ao sistema, enquanto os casos de uso são representados por elipses. As linhas conectam os atores aos casos de uso, indicando as interações entre eles. Este tipo de diagrama é especialmente útil para capturar e comunicar os requisitos funcionais do sistema de forma clara e compreensível, ajudando a definir as principais funcionalidades e interações esperadas.

https://lucid.app/lucidchart/ce0d625e-f55c-4ff0-b427-8f8b80dcef67/edit?viewport_loc=374%2C-684%2C1806%2C1581%2C0_0&invitationId=inv_f5d40ae6-b57e-4e12-ada2-2db32e0bba21

Figura 2



Fonte: Autor

3.2 Visão Arquitetural

O código foi escrito seguindo o paradigma Orientado a Objetos. Além disso, se encaixa no padrão arquitetural de Microkernel, em que é possível adicionar funcionalidades a um programa principal, nesse caso o VSCode é o kernel, enquanto Doc4me é um plugin/extensão.

A classe `MyConfig` é instanciada pela classe `Commands` que por sua vez se utiliza da técnica de injeção de dependência para transmitir as configurações da execução para todas as quatro classes responsáveis por executar os 5 comandos possíveis. Por sua vez, essas 4 classes se utilizam dos métodos estáticos da classe `Utils` para cumprir as suas funções. Os cinco comandos mencionados serão explicados com detalhes posteriormente.

4. Testes de Software

4.1. Projeto de Testes

Foram escritos testes unitários para garantir a correta execução das funcionalidades desenvolvidas. Os testes unitários estão descritos no arquivo `extension.test.ts` e podem ser executados rodando o comando `yarn test`.

Foi utilizado o *framework* de testes Javascript Mocha para realizar a execução dos testes. O *framework* Sinon foi utilizado para a construção de *stubs*, que são clones de um método. Esses clones podem ter seu retorno alterado, evitando-se a chamada do método clonado.

Cinco casos de teste foram escritos. Os dois primeiros testam o método `explainCode` da classe `Explainer`. O terceiro testa o método `getFileSummarizations` da classe `Summarizer`. O quarto teste garante o funcionamento do método `getFiles` da classe `Utils`. O quinto testa a `calculateTokens` da classe `BillCalculator`.

5. Implantação

5.1. Projeto de Implantação

O código do projeto pode ser encontrado nesse repositório público:
<https://github.com/caioreis123/doc4me>

As plataformas de hardware e software requeridas são bem abrangentes, tendo em vista que basta conexão com internet, a execução do editor de código VSCode e uma chave para utilização do serviço da OpenAI via API.

Em matéria de hardware o VSCode exige pelo menos 500MB de disco livre, 1.6GHz de processador e 1GB de memória RAM.

O programa pode ser executado a partir das seguintes configurações:

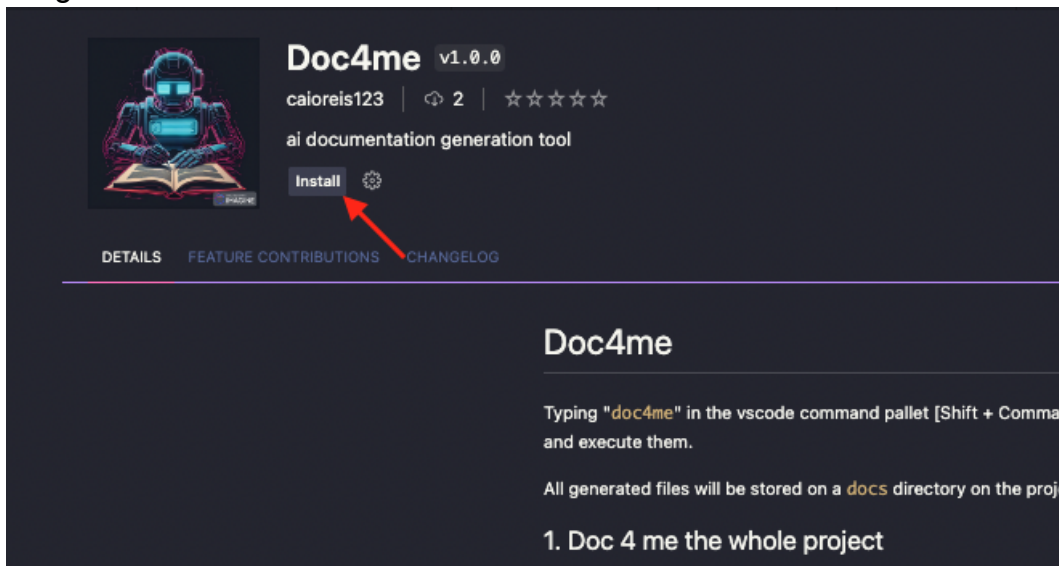
- Windows 10 (32 e 64 bits)
- MacOS com suporte de segurança da Apple
- Linux (Debian): Ubuntu Desktop 16.04, Debian 9
- Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 7, Fedora 34

6. Manual do Usuário

6.1 Instalação

1. Instale a extensão: procure por Doc4me no mercado de extensões do VSCode e instale.

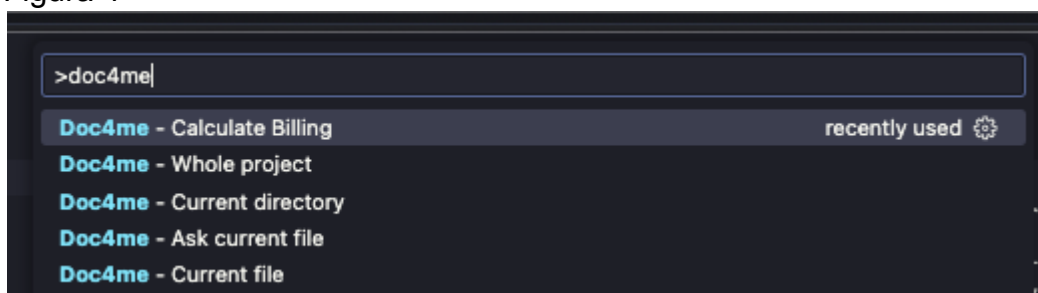
Figura 3



Fonte: Autor

2. Aperte o atalho *ctrl+shift+a* para rodar um comando no VSCode e digite o nome da extensão: "doc4me". Cinco opções vão aparecer. Cada uma representa um dos 5 comandos que a extensão é capaz de executar.

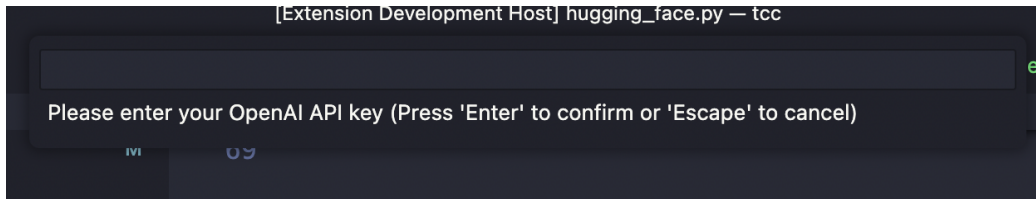
Figura 4



Fonte: Autor

3. Após a execução da extensão é solicitado que o usuário digite sua chave de acesso obtida com a Openai.

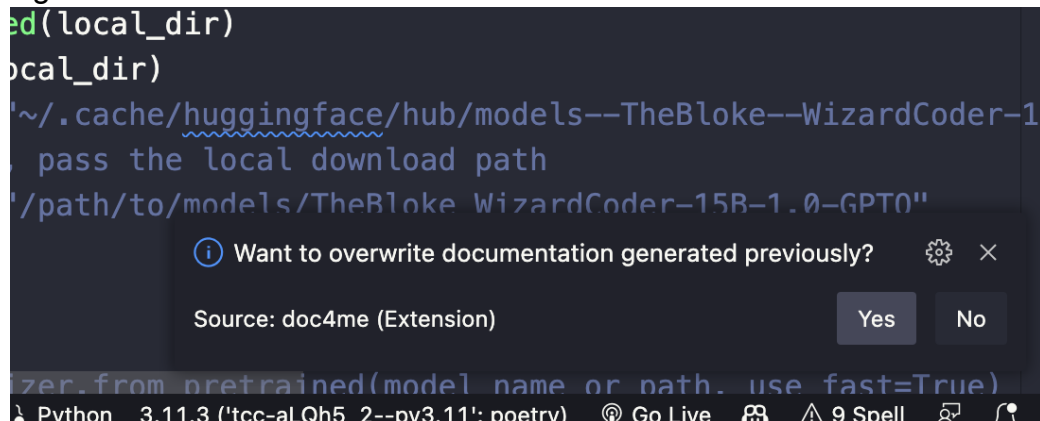
Figura 5



Fonte: Autor

4. Caso o programa já tenha sido executado anteriormente, será perguntado ao usuário se ele deseja remover os arquivos que já foram gerados e gerar novos.

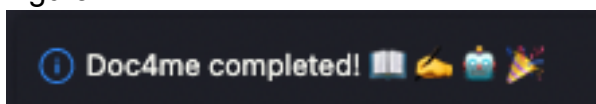
Figura 6



Fonte: Autor

5. Depois é só aguardar a execução do programa concluir com um aviso que aparecerá na tela.

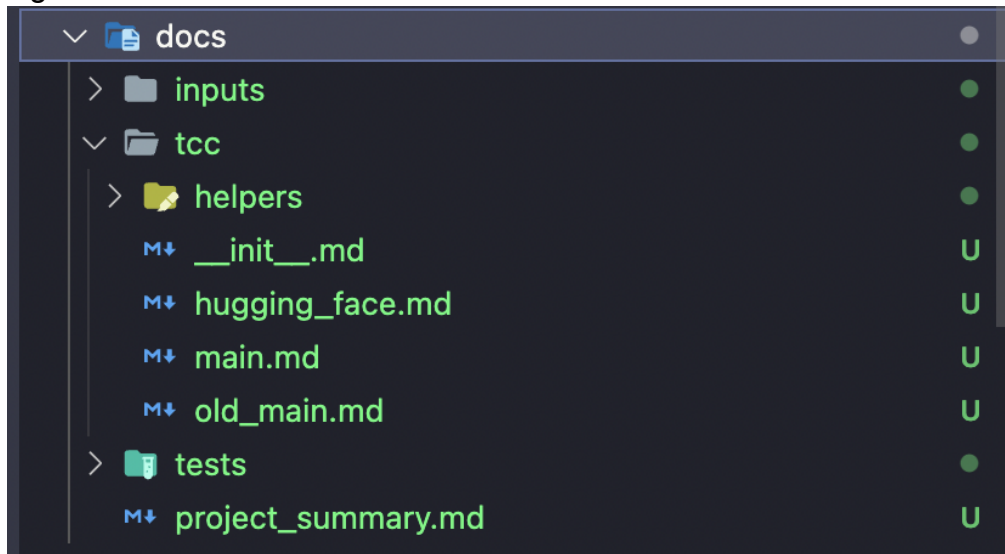
Figura 7



Fonte: Autor

6. Para consultar os arquivos de documentação gerados acesse o diretório "docs" presentes na raiz do projeto (é possível configurar a saída para outro diretório).

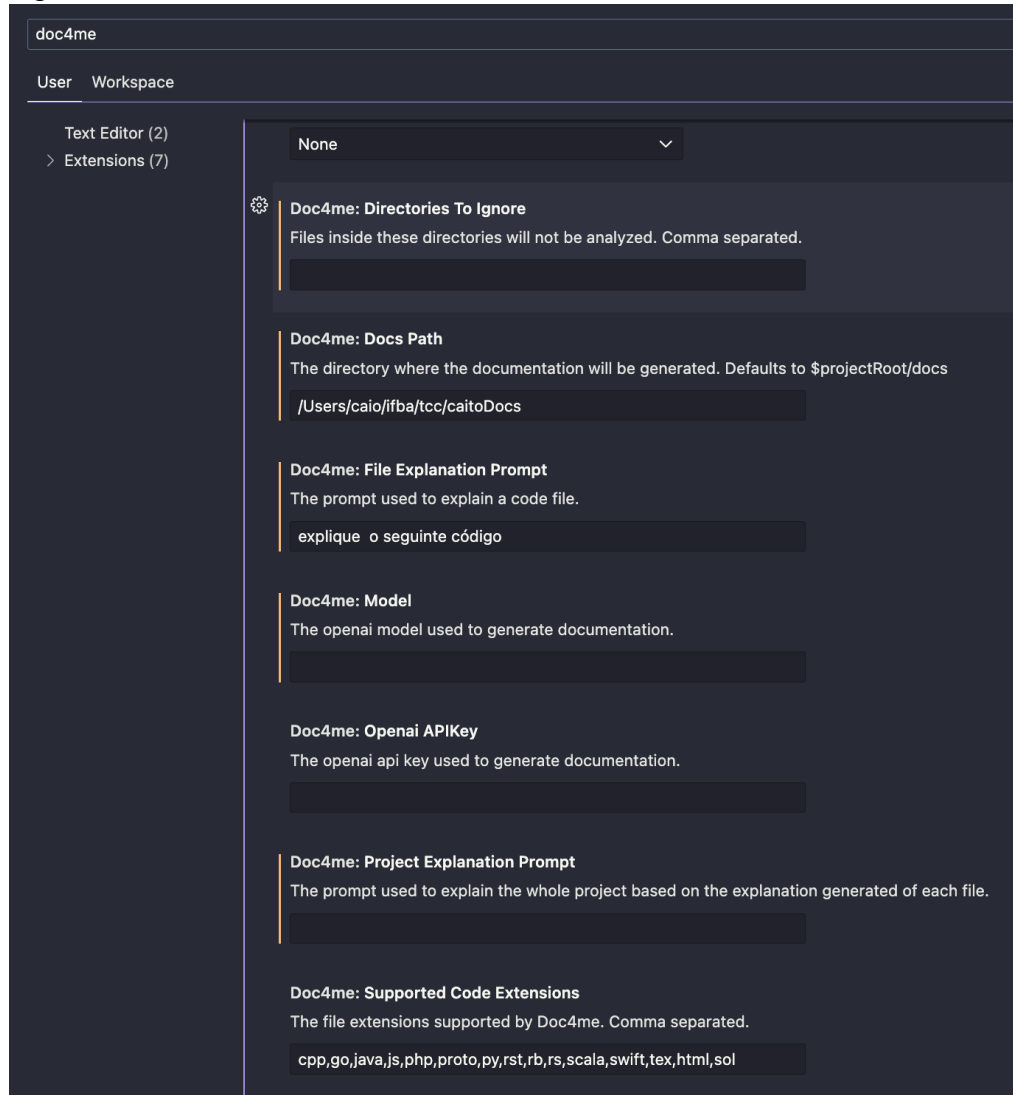
Figura 8



Fonte: Autor

7. Caso o usuário queira configurar a extensão e seus parâmetros basta abrir a aba de configurações do VSCode e pesquisar pelo nome da extensão. Lá será possível identificar quais as configurações presentes, seus valores padrão e as explicações de cada uma.

Figura 9



Fonte: Autor

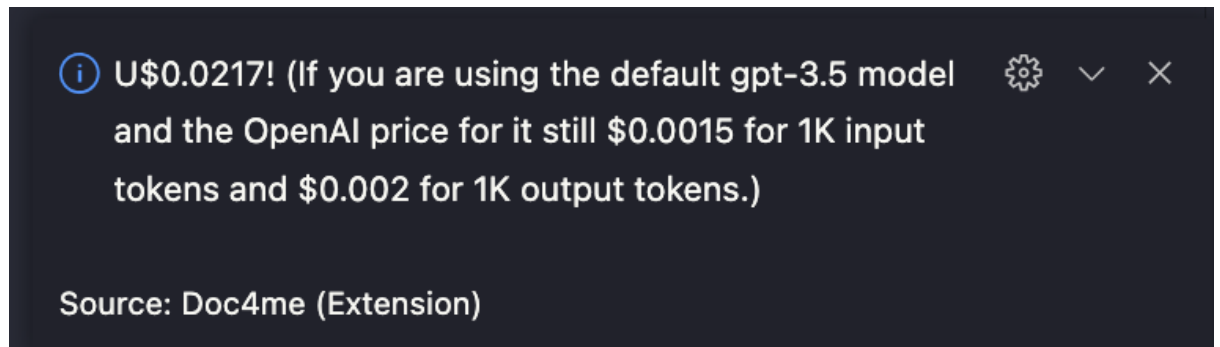
6.2 Comandos

Atualmente a extensão oferece 5 comandos, sendo eles:

1. **Doc4me - Whole project:** Será gerado um único arquivo markdown para cada arquivo de código em seu projeto lido pelo aplicativo (você pode alterar quais arquivos são lidos na configuração da extensão). Neste arquivo markdown, será escrita uma explicação em inglês (ou em outro idioma, já que o prompt é configurável) sobre o código. Um arquivo markdown contendo uma explicação sobre o projeto inteiro também será gerado no mesmo diretório de nome *docs* na raiz do projeto.
2. **Doc4me - Current directory:** Irá gerar documentação apenas para os arquivos armazenados no diretório do arquivo que o usuário está lendo no momento. O arquivo markdown que explica o projeto inteiro não será gerado.

3. **Doc4me - Current file:** Criará documentação apenas para o arquivo que o usuário está visualizando no momento em que o comando foi chamado.
4. **Doc4me - Ask current file:** Abre uma caixa de entrada no VSCode na qual o usuário poderá fazer qualquer pergunta sobre o conteúdo do arquivo que está atualmente visualizando. A resposta aparecerá em uma caixa de pop-up, mas também ficará armazenada em um arquivo chamado ask.txt dentro do diretório docs.
5. **Doc4me - Calculate Billing:** Exibe um popup com o valor em dólares do custo pelo uso da aplicação até então. Esse custo é calculado com base em uma planilha gerada no diretório docs com nome tokens.csv. Nessa planilha há a quantidade de tokens necessários para a explicação de cada arquivo. É um comando útil quando se possui um projeto grande e não quer rodar a explicação para o projeto todo sem antes ter uma estimativa do custo, dessa forma é possível rodar apenas para um diretório e calcular o custo daquele processamento pontual. O arquivo csv é editável, permitindo a remoção de determinados processamentos realizados, o que também traz flexibilidade para o usuário na hora de utilização da ferramenta.

Figura 10



Fonte: Autor

6.3 Configurações

Atualmente a extensão oferece 7 campos configuráveis, sendo eles:

1. **Directories to Ignore:** Em que é possível escrever os nomes das pastas que o usuário não deseja documentar. O padrão é: test,tests,docs,node_modules,dist,target,build,out,bin.
2. **Supported Code Extensions:** extensões de arquivos que poderão ser analisados pelo programa. O padrão é: cpp,go,java,js,php,proto,py,rst,rb,rs,scala,swift,tex,html,sol.
3. **Project Explanation Prompt:** Texto que será utilizado para a geração do arquivo contendo uma explicação resumida referente a todo o projeto. O padrão é: *Explain what this code project does, given the following explanations of each file.*
4. **File Explanation Prompt:** Texto que será passado para a inteligência artificial orientando a documentação de cada arquivo analisado. O padrão é: *Explain this code.*
5. **OpenAi Api Key:** chave de acesso aos serviços da api da OpenAi. Não há padrão, sendo necessário o cadastro no site da Openai.
6. **Model:** modelo de inteligência artificial que deve ser utilizado para o processamento. O padrão é o gpt-3.5-turbo

7. **Docs Path:** caminho absoluto do diretório que será utilizado para armazenar os arquivos gerados pela extensão. O padrão é a pasta *docs* na raiz do projeto.

Referências

The starting point for learning TypeScript. Disponível em: <<https://www.typescriptlang.org/docs/>>.

Extension API. Disponível em: <<https://code.visualstudio.com/api>>.

Sinon.JS - Standalone test fakes, spies, stubs and mocks for JavaScript. Works with any unit testing framework. Disponível em: <<https://sinonjs.org/>>.

OPENAI. OpenAI API. Disponível em: <<https://platform.openai.com/docs/introduction>>.

Visual Studio Code. Disponível em: <https://en.wikipedia.org/wiki/Visual_Studio_Code#:~:text=In%20the%202019%20Developer%20Survey>. Acesso em: 7 nov. 2023.

Introduction |  Langchain. Disponível em: <https://js.langchain.com/docs/get_started/introduction>. Acesso em: 7 nov. 2023.

Apêndices

Repositório com o código: <https://github.com/caioreis123/doc4me>

Vídeo explicando o uso da aplicação: <https://youtu.be/HJsLSPsMXfA>

Vídeo explicando o código que compõe a aplicação: https://youtu.be/EVSYFOY_TXI

Observação: No repositório do código é possível navegar por pastas com documentações geradas pela aplicação explicando o próprio projeto. Há documentação em inglês no diretório “docs” e português no diretório “pt-br-docs”.

Glossário, Siglas e Abreviações

CSV (Comma-Separated Values): Valores Separados por Vírgula é um formato de arquivo usado para armazenar e representar dados tabulares de forma simples e legível por máquina. Em um arquivo CSV, os dados são organizados em linhas e colunas, onde cada linha representa um registro ou entrada de dados e as colunas são separadas por vírgulas (,) ou outro delimitador, como ponto e vírgula (;).

Doc4me: nome da extensão apresentada como trabalho de conclusão de curso por esse documento.

IA (Inteligência Artificial): tecnologia capaz de realizar tarefas que normalmente exigiria inteligência humana e capaz de emular comportamentos como aprendizado e tomada de decisões.

LLM (Large Language Model): Modelos grandes de linguagem. Inteligência artificial utilizada para complementar a entrada do usuário a partir de uma grande massa de dados que foi utilizada no treinamento do modelo.

IDE (Integrated Development Environment): Ambiente de Desenvolvimento Integrado. É uma ferramenta de software que oferece um conjunto de recursos para auxiliar programadores e desenvolvedores no processo de escrita, teste e depuração de código de programação. Um IDE geralmente inclui um editor de código, ferramentas de compilação, depuração e muitos outros recursos para facilitar o desenvolvimento de software. Existem IDEs específicos para diferentes linguagens de programação e finalidades. Bem como IDEs que funcionam com múltiplas linguagens diferentes, como o VSCode.