Agora cada comando no código está comentado, explicando seu propósito e funcionamento.

ncluir os comantários: # Filename: generate\_summary.py # Date: 2024-06-30 # Root Project Folder: AutoNoCode # Command: python C:/AutoNoCode/scripts/generate\_summary.py

Here is the updated code that maintains the original functionality while ensuring each generated .txt file does not exceed 5MB:Limitar as mensagens de depuração a uma por fase e não repetir nomes de arquivos processados.corrigir a sintaxe no final do script para garantir que ele execute corretamente.Mensagens de Depuração Detalhadas adicionar uma mensagem inicial para confirmar que o script foi executado. Mensagens de Depuração limitadas a 10, Além disso, vamos garantir que o código seja compatível com o MyPy, adicionando as devidas anotações de tipo. navegue corretamente para o diretório especificado Gerar para copiar e funcionar ao colar no terminal. Os códigos fornecidos precisam de algumas melhorias para garantir que estejam de acordo com as anotações de tipo exigidas pelo mypy e boas práticas de codificação..Colocar o patch correto no terminal , primeiro move depois executa. Os comandos juntos. Evitar o diretório raiz. Gerar arquivos para serem executados no terminal do vs code. Se inteirar das seguintes bibliotecas. Ferramentas de análise de código. Pylint, MyPy e Pyright.

Encapsular sempre que possível,

# Path: automate\_typing.py # Filename: automate\_typing.py # Date: 2024-07-01 # Root Project Folder: AutoNoCode

add type annotations for all fixture references in the test file

Código para ps

Encapsular

Resumo dos commandos

S C:\AutoNoCode> # Listar a estrutura do diretório e copiá-la para a área de transferência

>> tree /F /A C:\AutoNoCode | clip

>>

PS C:\AutoNoCode>

PS C:\AutoNoCode> # Listar a estrutura do diretório e copiá-la para a área de transferência

>> Get-ChildItem -Recurse -File | ForEach-Object { $\_.FullName.Replace((Get-Location).Path, '') } | clip

# Filename: generate\_summary.py

# Date: 2024-06-30

# Root Project Folder: AutoNoCode

# Command: python C:/AutoNoCode/scripts/generate\_summary.py

import os

import glob

import codecs

from datetime import datetime

from itertools import islice

from typing import Dict, Any, List, Tuple

# Configurações ajustadas

MAX\_LINES\_PER\_FILE = 27000   # Máximo de linhas a serem incluídas por arquivo

MAX\_LINES\_PER\_PART = 3000    # Máximo de linhas por arquivo de resumo

MAX\_PARTS = 4

TRUNCATED\_TEXT = '...[TRUNCATED]...'

# Arquivos e diretórios importantes

IMPORTANT\_FILES = ['settings.py', 'Dockerfile', 'requirements.txt', 'package.json', 'README.md', 'main.py', 'app.py']

IMPORTANT\_DIRS = ['backend', 'frontend', 'scripts']

# Descrição do projeto

PROJECT\_DESCRIPTION = """

AutoNoCode é uma plataforma avançada que automatiza a criação de vídeos usando técnicas de IA. Este projeto inclui

módulos backend e frontend, além de integrações com serviços como o n8n para automação de fluxos de trabalho.

O objetivo principal é simplificar a criação de conteúdo visual de alta qualidade através de ferramentas automatizadas.

"""

class ProjectSummarizer:

    def \_\_init\_\_(self, root\_dir: str) -> None:

        self.root\_dir = root\_dir

        self.summary\_dir = os.path.join(root\_dir, 'scripts', 'summary')

        self.base\_summary\_file\_path = os.path.join(self.summary\_dir, 'summary')

        self.part\_counter = 1

        self.line\_counter = 0

        self.current\_part\_path = f"{self.base\_summary\_file\_path}\_part\_{self.part\_counter}.txt"

        self.included\_files = set()

    def collect\_project\_info(self) -> Dict[str, Any]:

        print("Coletando informações do projeto...")

        info = {

            "project\_name": os.path.basename(self.root\_dir),

            "date\_generated": datetime.now().strftime('%Y-%m-%d %H:%M:%S'),

            "total\_files": 0,

            "total\_dirs": 0,

        }

        for \_, dirs, files in os.walk(self.root\_dir):

            info["total\_dirs"] += len(dirs)

            info["total\_files"] += len(files)

        return info

    def get\_project\_structure(self) -> str:

        print("Gerando estrutura do projeto...")

        structure = []

        for root, dirs, files in os.walk(self.root\_dir):

            level = root.replace(self.root\_dir, '').count(os.sep)

            indent = ' ' \* 4 \* level

            structure.append(f"{indent}{os.path.basename(root)}/")

            subindent = ' ' \* 4 \* (level + 1)

            for f in files:

                structure.append(f"{subindent}{f}")

        return '\n'.join(structure)

    @staticmethod

    def read\_file\_content(file\_path: str, max\_lines: int) -> str:

        if not os.path.exists(file\_path):

            return f"{file\_path} not found."

        try:

            with codecs.open(file\_path, 'r', encoding='utf-8', errors='replace') as f:

                lines = list(islice(f, max\_lines))

                content = ''.join(lines)

                if next(f, None) is not None:

                    content += f"\n{TRUNCATED\_TEXT}"

                return content

        except Exception as e:

            return f"Failed to read {file\_path}: {e}"

    @staticmethod

    def get\_file\_modification\_time(file\_path: str) -> float:

        return os.path.getmtime(file\_path)

    def write\_to\_current\_part(self, text: str) -> None:

        if self.line\_counter + text.count('\n') > MAX\_LINES\_PER\_PART:

            self.part\_counter += 1

            self.line\_counter = 0

            self.current\_part\_path = f"{self.base\_summary\_file\_path}\_part\_{self.part\_counter}.txt"

        with open(self.current\_part\_path, 'a', encoding='utf-8') as f:

            f.write(text)

        self.line\_counter += text.count('\n')

    def generate\_summary\_with\_content(self) -> None:

        print("Gerando resumo do conteúdo do projeto...")

        latest\_files: Dict[str, Tuple[str, float]] = {}

        for root, dirs, files in os.walk(self.root\_dir):

            if not any(important\_dir in root for important\_dir in IMPORTANT\_DIRS):

                continue

            for file in files:

                file\_path = os.path.join(root, file)

                extension = os.path.splitext(file)[1]

                if extension in ['.py', '.js']:

                    mod\_time = self.get\_file\_modification\_time(file\_path)

                    if file not in latest\_files or mod\_time > latest\_files[file][1]:

                        latest\_files[file] = (file\_path, mod\_time)

        for file, (file\_path, \_) in latest\_files.items():

            if file\_path not in self.included\_files:

                self.included\_files.add(file\_path)

                content = self.read\_file\_content(file\_path, MAX\_LINES\_PER\_FILE)

                self.write\_to\_current\_part(f"{file}\n")

                if content != f"{file\_path} not found.":

                    self.write\_to\_current\_part(f"{'=' \* 80}\n")

                    self.write\_to\_current\_part(f"{content}\n")

                    self.write\_to\_current\_part(f"{'=' \* 80}\n\n")

    def create\_project\_summary\_header(self, project\_info: Dict[str, Any]) -> str:

        header = f"""

{PROJECT\_DESCRIPTION}

Project Name: {project\_info['project\_name']}

Date Generated: {project\_info['date\_generated']}

Total Files: {project\_info['total\_files']}

Total Directories: {project\_info['total\_dirs']}

{'=' \* 80}

Project Structure:

{'=' \* 80}

{self.get\_project\_structure()}

{'=' \* 80}

Directory Structure and File Contents:

{'=' \* 80}

"""

        return header

    def remove\_excess\_files(self) -> None:

        print("Removendo arquivos de resumo excedentes...")

        all\_files = sorted(glob.glob(f"{self.base\_summary\_file\_path}\_part\_\*.txt"))

        if len(all\_files) > MAX\_PARTS:

            for file in all\_files[MAX\_PARTS:]:

                try:

                    os.remove(file)

                except OSError as e:

                    print(f"Error removing file {file}: {e}")

    def run(self) -> None:

        print("Iniciando geração de resumo do projeto...")

        os.makedirs(self.summary\_dir, exist\_ok=True)

        project\_info = self.collect\_project\_info()

        header = self.create\_project\_summary\_header(project\_info)

        self.write\_to\_current\_part(header)

        self.generate\_summary\_with\_content()

        self.remove\_excess\_files()

        print("Geração de resumo do projeto concluída.")

def main() -> None:

    print("Executando generate\_summary.py...")

    project\_dir = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

    summarizer = ProjectSummarizer(project\_dir)

    summarizer.run()

    print("Execução de generate\_summary.py concluída.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()