Projeto Final - Avaliação A3

GERENCIADOR DE AQUIVOS

Relatório técnico apresentado na UC Organização de Computadores e Sistemas Operacionais pelo Prof. MSc Flávio Henrique da Silva.

Curitiba

2025

INTEGRANTES DO GRUPO

Roger Oliveira, RA: 172320937;

Eduardo Plotchkacz Drozdz, RA: 172311203;

Leandro Guerra, RA: 172317494.

SUMÁRIO

1 INTRODUÇÃO

Este relatório apresenta o desenvolvimento de um sistema de gerenciamento de arquivos, proposto como projeto final da disciplina de Organização de Computadores e Sistemas Operacionais. O objetivo principal é consolidar os conhecimentos adquiridos ao longo do semestre por meio da criação de uma aplicação funcional que interage diretamente com o sistema de arquivos do sistema operacional, oferecendo funcionalidades essenciais como abrir, copiar, mover, renomear e excluir arquivos e pastas, bem como registrar logs das ações realizadas.

2 DESENVOLVIMENTO

O projeto foi desenvolvido com base nos conceitos estudados na UC, utilizando a linguagem de programação Python. A aplicação foi construída de forma modular, dividida entre a lógica de backend e a interface com o usuário.

2.1 Divisão das Tarefas

Roger Oliveira: Desenvolvimento da lógica de backend, responsável pelas funções de manipulação de arquivos.

Eduardo Plotchkacz Drozdz: Implementação da interface gráfica e integração com o backend.

Leandro Guerra: Elaboração do relatório técnico, apoio na codificação e validação das funcionalidades implementadas.

2.2 Estrutura do Projeto

O projeto do Gerenciador de Arquivos utiliza diversas bibliotecas Python, tanto padrão quanto de terceiros, para construir sua funcionalidade. Abaixo estão as principais bibliotecas e como elas são empregadas no sistema:

-customtkinter:

É a principal biblioteca utilizada para a construção da interface gráfica do usuário (GUI). Ela estende o tkinter padrão, fornecendo widgets modernos e personalizáveis, além de temas escuros e claros.

Permite criar a janela principal (CTk), botões (CTkButton), rótulos (CTkLabel), campos de entrada (CTkEntry), e barras de rolagem (CTkScrollbar), definindo a aparência e o comportamento visual da aplicação. É fundamental para a experiência do usuário.

-tkinter.ttk:

Parte da biblioteca padrão tkinter, ttk (Themed Tkinter) fornece acesso a widgets temáticos que se parecem e se comportam de forma mais nativa em diferentes sistemas operacionais. É usado em conjunto com customtkinter para elementos como o Treeview.

Utilizado para criar as visualizações em árvore (Treeview) que exibem a estrutura de diretórios na barra lateral e a lista de arquivos na área principal. Permite estilizar esses componentes para se integrarem ao tema do customtkinter.

-os (Built-in):

Módulo padrão do Python que fornece uma maneira de interagir com o sistema operacional. É amplamente utilizado para operações de caminho de arquivo e diretório.

Empregado em FileOperations para renomear e excluir arquivos (os.rename, os.remove), verificar se um caminho é um arquivo ou diretório (os.path.isfile, os.path.isdir), e manipular caminhos (os.path.join, os.path.dirname). Também é usado para abrir arquivos com o programa padrão do sistema (os.startfile no Windows, os.system para xdg-open ou open em outros sistemas).

-shutil (Built-in):

Módulo padrão do Python que oferece operações de arquivo de alto nível, como cópia, movimentação e exclusão de árvores de diretórios.

Utilizado em FileOperations para copiar arquivos (shutil.copy2), mover arquivos/diretórios (shutil.move) e remover diretórios recursivamente (shutil.rmtree). Simplifica a manipulação de arquivos complexa.

-logging (Built-in):

Módulo padrão do Python que fornece um sistema flexível para emitir mensagens de log de aplicações.

É a base da classe AppLogger. Permite configurar diferentes níveis de log (INFO, WARNING, ERROR, DEBUG, CRITICAL), direcionar logs para arquivos e/ou console, e formatar as mensagens de log. Essencial para rastrear o comportamento da aplicação e depurar problemas.

-psutil:

Biblioteca de terceiros para obter informações sobre processos em execução e utilização do sistema (CPU, memória, discos, rede).

Empregado na classe DriveDetector para listar todas as partições de disco montadas (psutil.disk_partitions()). Isso permite que a aplicação identifique e exiba as unidades de armazenamento disponíveis na barra lateral da GUI.

-**pywin32** (Apenas para Windows):

Um conjunto de extensões Python para Windows que fornece acesso a muitas APIs do Windows.

Utilizado como detecção de drives USB de forma mais confiável e manipulação de atalhos. Sua presença no requirements.txt foi usado para funcionalidades específicas do Windows que não são cobertas por os ou psutil de forma genérica.

-**tkinter.filedialog** (Built-in):

Módulo padrão do tkinter que fornece caixas de diálogo comuns para seleção de arquivos e diretórios.

Utilizado na GUI para permitir que o usuário selecione arquivos ou diretórios de origem e destino para operações como copiar, mover e abrir. Por exemplo, filedialog.askopenfilename() e filedialog.askdirectory().

-**tkinter.messagebox** (Built-in):

Módulo padrão do tkinter que fornece caixas de diálogo para exibir mensagens de informação, aviso, erro ou para solicitar confirmação do usuário.

Empregado na GUI para exibir mensagens de status (sucesso/erro) e para solicitar confirmação do usuário antes de realizar operações destrutivas, como a exclusão de arquivos.

Essas bibliotecas trabalham em conjunto para fornecer a funcionalidade completa do Gerenciador de Arquivos, desde a interface visual até as operações de baixo nível no sistema de arquivos e o registro de eventos.

2.3 Explicação da Aplicação/Software

O Gerenciador de Arquivos é uma aplicação de desktop com interface gráfica (GUI) que permite ao usuário navegar, visualizar e manipular arquivos e diretórios de

forma intuitiva. Sua arquitetura é dividida em duas partes principais: Frontend (GUI) e Backend (Operações de Arquivo, Detecção de Drives, Log).

1. Inicialização (main.py): O ponto de entrada da aplicação é o script main.py. Ele é responsável por:

-Configurar o modo de aparência (padrão do sistema) e o tema de cor (blue) da interface usando a biblioteca customtkinter.

-Criar uma instância da classe FileManagerGUI, que representa a janela principal da aplicação.

-Iniciar o loop principal da aplicação (app.mainloop()), que mantém a janela ativa e responsiva a eventos do usuário, como cliques e digitação.

2. Interface Gráfica (frontend/gui.py - Classe FileManagerGUI): A GUI é construída com customtkinter e tkinter.ttk, oferecendo uma experiência de usuário moderna. As principais funcionalidades e componentes são:

-Configuração da Janela: A janela principal é centralizada na tela e aplica um tema escuro com fontes personalizadas para uma aparência aprimorada.

-Barra Lateral (Treeview de Diretórios):

-Exibe uma árvore de diretórios, incluindo "Acesso Rápido" (para pastas comuns como Downloads, Documentos) e "Este Computador" (listando unidades montadas).

-A detecção de unidades é realizada pela classe DriveDetector do backend.

-Ao clicar em um diretório na barra lateral, a função load_directory() é acionada para exibir o conteúdo desse diretório na área principal.

A árvore de diretórios é populada dinamicamente: apenas os diretórios de nível superior são carregados inicialmente, e os subdiretórios são carregados sob demanda quando o usuário expande um nó.

Área Principal (Lista de Arquivos):

Exibe o conteúdo do diretório atual em uma tabela (file_tree), mostrando nome, tipo, tamanho e data de modificação de arquivos e pastas.

Permite navegar para o diretório pai e digitar caminhos diretamente para navegação rápida.

Um clique duplo em um item abre o diretório (se for pasta) ou tenta abrir o arquivo (se for arquivo).

Botões de Operação: Botões como "Copiar", "Mover", "Renomear", "Excluir" e "Abrir" chamam funções correspondentes que interagem com a classe FileOperations do backend para executar as ações no sistema de arquivos. Diálogos de seleção de arquivo/diretório e caixas de mensagem de confirmação são utilizados para interação com o usuário.

Mensagens de Status: Um rótulo na parte inferior da janela exibe mensagens de sucesso ou erro para o usuário, fornecendo feedback sobre as operações.

3. Operações de Backend: O backend é composto por três classes principais que lidam com a lógica de negócio e interagem com o sistema operacional:

-FileOperations (backend/file_operations.py): É o núcleo das operações de arquivo, fornecendo métodos para copiar, mover, renomear e excluir arquivos/diretórios. Todas as operações incluem tratamento de erros (permissão negada, arquivo não encontrado, etc.) e registram os eventos no sistema de log.

-DriveDetector (backend/drive_detector.py): Responsável por identificar e listar as unidades de disco montadas no sistema, utilizando a biblioteca psutil. Isso é essencial para popular a barra lateral da GUI com os pontos de montagem dos drives.

-AppLogger (backend/logger.py): Fornece uma interface consistente para registrar eventos e erros em um arquivo de log e no console. É utilizada por todas as classes de backend para registrar suas atividades, facilitando o rastreamento e a depuração.

2.4 Orientações de execução da Aplicação/Software

Para executar a aplicação do Gerenciador de Arquivos, siga os passos abaixo:

1.Clonar o Repositório: Abra um terminal ou prompt de comando e clone o repositório do GitHub para o seu ambiente local:

2.Navegar até o Diretório do Projeto: Entre no diretório do projeto clonado:

3.Instalar Dependências: O projeto utiliza algumas bibliotecas externas que precisam ser instaladas. As dependências estão listadas no arquivo requirements.txt. (recomenda-se criar um ambiente virtual para isolar as dependências do projeto)

-customtkinter: Para a construção da interface gráfica moderna.

-psutil: Para a detecção de unidades de disco.

-pywin32 (apenas para Windows): Para funcionalidades específicas do sistema operacional Windows, como detecção de drives de forma mais robusta.

4.Executar a Aplicação: Após instalar as dependências, você pode iniciar a aplicação executando o script principal:

2.5 Repositório

https://github.com/RogerOliveira1/Gerenciador-de-arquivos-python/tree/main

3 CONCLUSÃO

O desenvolvimento do Gerenciador de Arquivos permitiu a criação de uma aplicação funcional, clara e bem organizada, capaz de realizar operações essenciais no sistema de arquivos de forma prática e eficiente. A separação entre as partes visuais e operacionais do sistema tornou o código mais compreensível e fácil de manter.

A interface gráfica oferece uma experiência agradável ao usuário, enquanto o backend garante que as ações sejam realizadas com segurança, registrando todas as atividades e lidando com possíveis erros de forma adequada. As bibliotecas utilizadas foram escolhidas com critério, contribuindo para a estabilidade e a compatibilidade do sistema em diferentes ambientes.

De modo geral, o projeto alcançou seus objetivos, reunindo conceitos importantes da área de sistemas operacionais e organização de computadores em uma solução aplicável e de fácil utilização.

REFERÊNCIAS

TANENBAUM, Andrew S.; BOS, Herbert. Modern Operating Systems. 4. ed. Pearson, 2015.

SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg. Fundamentos de Sistemas Operacionais. 9. ed. Rio de Janeiro: LTC, 2018.

PYTHON SOFTWARE FOUNDATION. Python 3.12 Documentation. Disponível em: https://docs.python.org/3.12/. Acesso em: 12 jun. 2025.

PYTHON SOFTWARE FOUNDATION. Logging HOWTO. Disponível em: https://docs.python.org/3/howto/logging.html. Acesso em: 12 jun. 2025.

TKDOCS. Tkinter GUI Programming by Example. Disponível em: https://tkdocs.com/. Acesso em: 12 jun. 2025.

MICROSOFT DOCS. pywin32 library for Windows extensions. Disponível em: https://learn.microsoft.com/en-us/windows/win32/. Acesso em: 12 jun. 2025.

GITHUB. Gerenciador de Arquivos Python. Disponível em: https://github.com/RogerOliveira1/Gerenciador-de-arquivos-python. Acesso em: 12 jun. 2025.

FOWLER, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002

OPENAI. ChatGPT (versão GPT-4). Assistente de inteligência artificial utilizado como apoio na revisão estrutural do relatório técnico e correção de erros no código-fonte. Disponível em: https://chat.openai.com/. Acesso em: 12 jun. 2025.