Sabih Uddin Qureshi

221531

Operating Systems Lab

Semester Project

File Management System

Introduction

The Python File Manager is a GUI-based application that provides users with an intuitive interface for managing their files and directories. It leverages the **customtkinter** library for the graphical interface and includes features like file navigation, opening, renaming, moving, copying, and deleting files or directories. The application also supports quick access to common locations and a search functionality.

Features

- **Navigation**: Navigate through directories and open files.
- **Quick Access**: Common locations like Documents, Videos, Pictures, and Downloads are easily accessible.
- **File Operations**: Rename, move, copy, and delete files and directories.
- **Search**: Search for files or directories by name.
- **Dynamic Path Display**: Current path is displayed and updated as the user navigates.

Libraries & Modules

1. os Library

The **os** module in Python provides a way of using operating system-dependent functionality like reading or writing to the file system. In the File Manager application, the **os** module is used for:

- Navigating directories (**os.getcwd()**, **os.listdir()**, **os.path.join()**, **os.path.isdir()**)
- Managing paths (**os.path.expanduser()**, **os.path.basename()**)
- Performing file operations (like **os.unlink()** for deleting files)

2. shutil Library

The **shutil** module offers a number of high-level operations on files and collections of files. In the File Manager, it is used for:

- Copying files and directories (**shutil.copy()**, **shutil.copytree()**)
- Moving files and directories (**shutil.move()**)
- Removing directories (**shutil.rmtree()**)

3. send2trash Library

The **send2trash** module sends files and directories to the trash or recycle bin instead of permanently deleting them. This is safer because it allows recovery of mistakenly deleted files. The typical usage in this application would be replacing permanent deletion with sending items to the trash:

- **send2trash.send2trash(selected_path)**

4. customtkinter (ctk) Library

customtkinter is a library that extends the standard **tkinter** library to provide more modern and customizable GUI elements. It is used for:

- Creating a modern-looking interface with custom themes (**ctk.set_appearance_mode()**, **ctk.set_default_color_theme()**)
- Custom widgets like **CTk**, **CTkFrame**, **CTkLabel**, **CTkButton** which are used to build the application's interface.

5. tkinter Library

tkinter is the standard GUI toolkit in Python, which **customtkinter** extends. It is used for:

- Creating dialogs and message boxes (**tkinter.filedialog**, **tkinter.messagebox**, **tkinter.simpledialog**)
- Basic widgets like **Listbox**, which are used to display lists of files and directories.

Installing Dependencies:

Open your terminal and run the following commands:

```
pip install send2trash
pip install customtkinter
```

Code Structure

The code is structured into a single class, **FileManager**, which inherits from **ctk.CTk**. This class encapsulates all the functionality and the GUI elements.

```
class FileManager(ctk.CTk):
    def __init__(self):
        super().__init__()
        ...
        self.create_widgets()
    ...
    def create_widgets(self):
        ...
    def populate_listbox(self):
        ...
    def populate_locations(self):
        ...
    def on_location_selected(self, event):
        ...
    def on_item_selected(self, event):
        ...
    def open_item(self):
        ...
    def rename_item(self):
        ...
    def move_item(self):
        ...
    def copy_item(self):
        ...
    def delete_item(self):
        ...
    def search_item(self):
        ...
    def search_recursive(self, directory, query):
        ...
    def go_back(self):
        ...
    def add_to_quick_access(self):
        ...
```

Class and Methods

FileManager:

__init__(self)

- Initializes the main window, sets up the appearance, defines common locations, and creates the UI widgets.

create_widgets(self)

- Creates and packs the top, main, left, right, and bottom frames.
- Initializes listboxes and buttons, and binds events.

populate_listbox(self)

- Populates the main file listbox with the contents of the current directory.

populate_locations(self)

- Populates the locations listbox with common locations.

on_location_selected(self, event)

- Handles double-click events on the locations listbox to navigate to the selected location.

on_item_selected(self, event)

- Handles double-click events on the main file listbox to open directories or files.

open_item(self)

- Opens the selected item in the default application or navigates into a directory.

rename_item(self)

- Prompts the user to enter a new name and renames the selected item.

move_item(self)

- Opens a dialog to select a new directory and moves the selected item to this directory.

copy_item(self)

- Opens a dialog to select a new directory and copies the selected item to this directory.

delete_item(self)

- Deletes the selected item after user confirmation.

search_item(self)

- Prompts the user to enter a search query and lists all matching items in the current directory and subdirectories.

search_recursive(self, directory, query)

- Recursively searches for items matching the query within a directory.

go_back(self)

- Navigates back to the parent directory.

add_to_quick_access(self)

- Adds the selected directory to the quick access list.

User Interface

The user interface is designed with a dark theme for a modern look, using the **customtkinter** library. The main window includes:

- **Top Frame**: Displays the current path.
- **Left Frame**: Lists common locations for quick access.
- **Right Frame**: Displays the contents of the current directory.
- **Bottom Frame**: Contains buttons for various file operations.

Usage/Operation:

1. **Navigate**: Double-click on items in the list to navigate into directories or open files.
2. **Quick Access**: Double-click on common locations to quickly navigate to those directories.
3. **File Operations**:
    - Select an item and click on "Open" to open it.
    - Click on "Rename" to rename the selected item.
    - Click on "Move" or "Copy" to move or copy the selected item to another directory.
    - Click on "Delete" to delete the selected item.
4. **Search**: Click on "Search" to search for items by name within the current directory and subdirectories.
5. **Back**: Click on "Back" to navigate to the parent directory.
6. **Add to Quick Access**: Select a directory and click "Add to Quick Access" to add it to the quick access list.

Error Handling

- **PermissionError**: Displays an error message if access to a directory is denied.
- **File Operations**: Confirms with the user before performing actions like deletion to prevent accidental data loss.
- **No Selection**: Warns the user if no item is selected for operations that require a selection.

Usage of customtkinter:

customtkinter is a modern and customizable library for creating graphical user interfaces (GUIs) in Python. It extends the capabilities of the standard **tkinter** library by providing additional features and improved aesthetics. **customtkinter** is particularly useful for developers looking to create more visually appealing and user-friendly applications with minimal effort.

Reasons for Choosing customtkinter:

- Modern Look and Feel
- Dark Mode Support
- Customizability
- Enhanced Widgets
- Simplified Theming

Final Result: