OPERATING SYSTEMS PROJECT

F1+TF1

DR. K. VALLIDEVI

FILE EXPLORER

TEAM:

Mirdulaa Balaji 22BAI1284

Harshitha Balaji 22BAI1038

Sanjana Suresh 22BAI1439

Venna Divya Reddy 22BAI1476

TABLE OF CONTENTS

ABSTRACT

The "Python File Explorer" project presents the development and implementation of a versatile and intuitive file management system built entirely in Python. In response to the growing need for an efficient and user-friendly file handling tool, this project aims to provide a robust solution that allows users to navigate, manage, and interact with their files and directories seamlessly.

The file explorer offers a range of functionalities, including file browsing, manipulation, and search capabilities, all encapsulated within an intuitive graphical user interface (GUI). Leveraging Python's libraries and frameworks, the application focuses on simplicity, performance, and cross-platform compatibility.

This report details the objectives, methodologies, and design considerations undertaken during the development process. It discusses the architecture, key features, implementation strategies, and testing methodologies employed to ensure the reliability and functionality of the file explorer.

Furthermore, the report highlights challenges faced during development, future enhancements, and potential applications of the file explorer.

INTRODUCTION

In the realm of digital organization, the management of files and directories stands as an indispensable aspect of daily tasks for individuals and enterprises alike. The "Python-based File Explorer" project represents a concerted effort to address the complexities and demands of efficient file handling by leveraging the power and versatility of Python programming.

The exponential growth of data across diverse platforms necessitates a sophisticated yet user-friendly file management system. Existing solutions often fall short in providing a seamless user experience, leading to the conceptualization of a custom file explorer developed entirely in Python. This project endeavors to offer a solution that not only simplifies file navigation but also enriches it with functionalities that resonate with the contemporary user.

At its core, this project aims to develop a robust file explorer embedded with features such as comprehensive file browsing, versatile file manipulation options, and an intuitive graphical interface. Powered by Python's rich set of libraries and its innate flexibility, the File Explorer seeks to transcend the limitations of conventional file management tools.

This report delineates the intricate journey undertaken in conceptualizing, designing, and implementing the Python-based File Explorer. It delves into the underlying methodologies, architectural nuances, pivotal features, testing frameworks employed, and the evolution of the project from inception to fruition.

OBJECTIVE

- Create an interface that allows users to effortlessly navigate through directories, providing a clear and efficient view of file structures.
- Implement functionalities for managing files and folders, including options for creating, copying, moving, renaming, and deleting files.
- Design an intuitive graphical user interface (GUI) that prioritizes user-friendliness and accessibility, catering to diverse user needs and preferences.
- Ensure the file explorer's efficiency in handling various file types and sizes while maintaining reliability in executing file operations without compromising data integrity.
- Use the concept of journaling to maintain a history of operations done in the file explorer.
- Utilize file retrieval mechanism to recover deleted files

REQUIREMENTS

<u>SOFTWARE</u>

- Any file editor compatible with Python. The editor used here is Visual Studio Code along with necessary extensions Python and Pylance.
- Various Python libraries like os, tkinter and datetime.

<u>HARDWARE</u>

- Any device with windows 10/up or macOS.

FEATURES

1. **File Navigation:** Users can browse through directories and view files and folders within the chosen directory.
2. **File Operations:**
   - **Copy Files:** Users can create copies of selected files within the current directory.
   - **Move Files:** Files can be relocated from the current directory to another chosen directory.
   - **Delete Files:** Users can remove files from the current directory.
   - **Retrieve Deleted Files:** Deleted files can be restored from a designated "deleted_files" folder.
3. **Display Features:**
   - The program displays the current directory path.
   - Lists files and directories in the current path in a user-friendly manner using a graphical interface.
4. **Journal Functionality:**
   - Provides a log of file operations, such as move, copy, and delete actions, timestamped for reference.
5. **Directory Navigation:**
   - Enables users to move up one directory level to access parent directories.
6. **Deleted Files Management:**
   - Users can view files that have been previously deleted and perform actions like retrieval or permanent deletion from the "deleted_files" folder.
7. **Graphical User Interface (GUI):**
   - Implements a basic GUI using Tkinter, facilitating user interaction through buttons and lists.

DESIGN

1. **Tkinter GUI:**

   The program employs Tkinter to create a graphical interface for users to interact with the file manager. It includes buttons, labels, listboxes, and pop-up windows (Toplevel) for specific functionalities.

2. **Function-based Structure:**

   The script follows a function-based design, where various functions handle specific file operations, GUI updates, and interactions with the file system.

3. **Modularity:**

   Each functional aspect (e.g., file copying, moving, deleting, retrieval from deleted files) is encapsulated within separate functions, ensuring a modular and organized structure.

4. **Logging Mechanism:**

   The program logs file operations (copy, move, delete) into a log file (file_manager_log.txt) with timestamps, aiding in tracking and reviewing past actions.

5. **User Interface (UI):**

   The UI includes a main window (root) displaying the current directory path and a listbox presenting the files and directories within that path. Buttons for copy, delete, move, journal, and move-up actions are included in the UI. A separate window (Toplevel) manages deleted files, allowing users to retrieve or permanently delete them.

6. **File Operations Handling:**

   Functions such as copy_file(), delete_file(), move_file(), log_operation(), and retrieve_file() manage file operations by interacting with the operating system through Python's os and shutil libraries.

7. **Directory Navigation:**

   The navigate_directory() function enables users to double-click on directories to navigate within the file system.

8. **Logging System:**

   The log_operation() function writes file operations and associated paths with timestamps into a log file for later reference.

9. **Deleted Files Management:**

   Functions like refresh_deleted_files_list() and delete_permanently() handle operations related to managing files within the deleted_files folder.

IMPLEMENTATION

1. **Module Imports:**
   - The script starts by importing necessary modules such as os, tkinter, messagebox, filedialog, Text, and Scrollbar to enable file system operations and GUI creation.
2. **GUI Creation:**
   - Tkinter is used to create a graphical user interface featuring a main window (root) displaying the current directory path and a listbox presenting files and directories.
   - Buttons (Copy, Delete, Move, Journal, Move-Up) are added to perform respective file operations and actions.
3. **File Operations Handling:**
   - Functions such as copy_file(), delete_file(), move_file(), and retrieve_file() manage file operations by interacting with the operating system's file system using the os and shutil libraries.
4. **Directory Navigation:**
   - The navigate_directory() function enables users to double-click on directories within the listbox to navigate through the file system.
5. **Logging System:**
   - The log_operation() function writes file operations and associated paths with timestamps into a log file (file_manager_log.txt) for record-keeping.
6. **Deleted Files Management:**
   - Functions like refresh_deleted_files_list() and delete_permanently() handle operations related to managing files within the deleted_files folder, providing functionalities to retrieve or permanently delete files.

FUTURE ENHANCEMENTS

1. **Visual Improvements:**

   Introduce icons, themes, or visual cues to enhance the interface's aesthetic appeal and usability.

2. **Drag-and-Drop Functionality:**

   Implement drag-and-drop features for file/folder manipulation within the GUI.

3. **File Search:**

   Add a search feature to locate specific files or folders within the directory structure.

4. **File Attributes:**

   Display and allow users to modify file attributes such as permissions, ownership, and creation dates.

5. **Cross-Platform Compatibility:**

   Ensure the application works seamlessly across multiple operating systems.

6. **File Encryption:**

   Introduce encryption options for sensitive files.

7. **Cloud Integration:**

   Integrate with cloud storage platforms like Google Drive, Dropbox, etc.

8. **Version Control:**

   Incorporate basic version control features for file tracking and management.

9. **Accessibility Features:**

   Ensure compliance with accessibility standards for users with disabilities.

10. **Multi-Language Support:**

    Implement localization to support multiple languages.

11. **User Profiles:**

    Implement user accounts with personalized settings and access permissions.

12. **File Sharing:**

    Introduce sharing functionalities allowing users to share files or directories with others.

CONCLUSION

The development of the Python-based File Explorer stands as a testament to the power of Python in creating versatile and user-centric applications. This project embarked on the journey of addressing the intricate landscape of file management by providing a functional and foundational file management tool.

Throughout the development process, the project successfully achieved essential functionalities, offering users the ability to navigate directories, perform file operations, manage deleted files, and track actions through a logging system. Leveraging Python's capabilities, particularly the Tkinter library for GUI creation and the interaction with the operating system's file system, this file manager establishes a robust groundwork for future enhancements.

The extensibility of the codebase allows for seamless integration of new features and enhancements, aligning with the ever-evolving needs of users in the digital landscape. By fostering collaboration, incorporating user feedback, and prioritizing user experience, the Python-based File Explorer holds potential to become a comprehensive and indispensable tool for efficient file management.

REFERENCES - BIBLIOGRAPHY

https://data-flair.training/blogs/python-file-explorer-project/

https://www.geeksforgeeks.org/file-handling-python/?ref=lbp

https://www.geeksforgeeks.org/journaling-or-write-ahead-logging/

https://www.tutorialspoint.com/python/python_gui_programming.htm

https://www.pythontutorial.net/tkinter/

CODE-OUTPUT

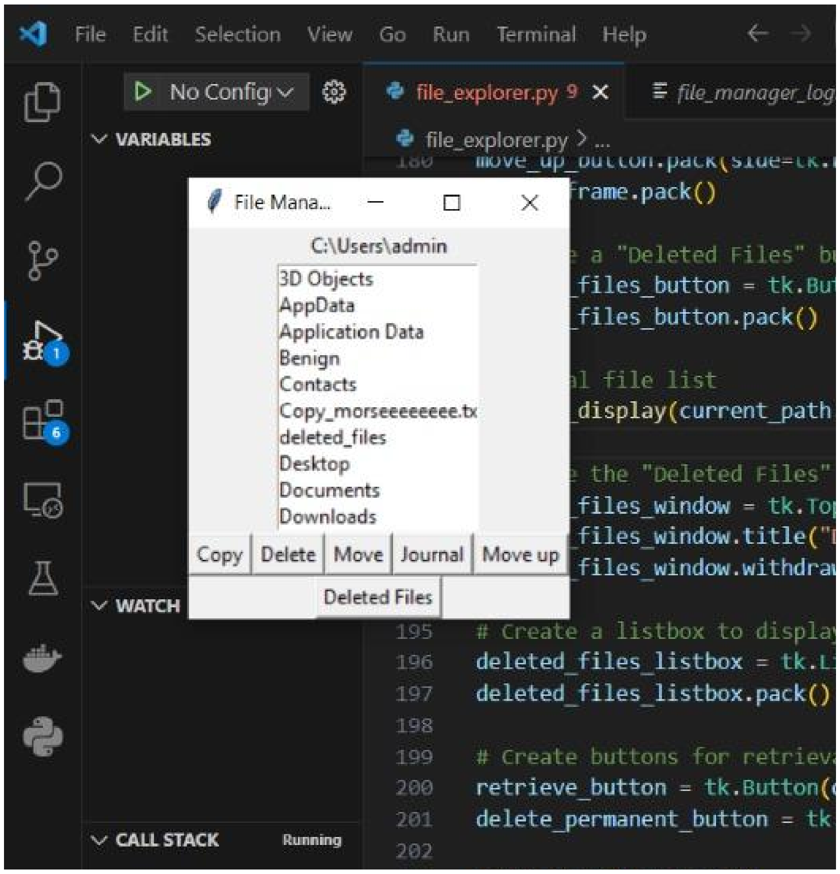
```
import os
import tkinter as tk
from tkinter import messagebox, filedialog, Text, Scrollbar
import datetime



# Define the log file path
log_file = "file_manager_log.txt"
# Define the "deleted_files" folder path
deleted_files_folder = r"C:\Users\admin\deleted_files"

# Function to list all the directories
def list_directory(path):
    f = []
    for files in os.listdir(path):
        if not files.startswith('.'):
            f.append(files)
    return f
def move_file():
    selected_indices = file_listbox.curselection()
    if selected_indices:
        selected_item = file_listbox.get(selected_indices[0])
        source_path = os.path.join(current_path.get(),
selected_item)

        # Prompt the user to choose the destination directory
        destination_dir = filedialog.askdirectory(title="Select
Destination Directory")
        if destination_dir:
            destination_path = os.path.join(destination_dir,
selected_item)
            try:
                os.rename(source_path, destination_path)
                refresh_display(current_path.get())
                log_operation("Move", source_path,
destination_path)
            except Exception as e:
                messagebox.showerror("Error", str(e))



# Function to view the contents when it is double-clicked
def navigate_directory(event):
    selected_item =
file_listbox.get(file_listbox.curselection())
    new_path = os.path.join(current_path.get(), selected_item)
    refresh_display(new_path)

# Function to copy a file
def copy_file():
    selected_indices = file_listbox.curselection()
    if selected_indices:
        selected_item = file_listbox.get(selected_indices[0])
        source_path = os.path.join(current_path.get(),
selected_item)
        destination_path = os.path.join(current_path.get(),
"Copy_" + selected_item)
        try:
            import shutil
            shutil.copy(source_path, destination_path)
            refresh_display(current_path.get())
            log_operation("Copy", source_path, destination_path)
        except Exception as e:
            messagebox.showerror("Error", str(e))
# Function to display the contents of a file in a new window
def show_file_content(file_path, content):
    file_content_window = tk.Toplevel(root)
    file_content_window.title(f"File Content -
{os.path.basename(file_path)}")

    text_widget = Text(file_content_window, wrap=tk.WORD)
    text_widget.pack(expand=tk.YES, fill=tk.BOTH)

    scrollbar = Scrollbar(file_content_window,
command=text_widget.yview)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
    text_widget.config(yscrollcommand=scrollbar.set)

    text_widget.insert(tk.END, content)
    text_widget.config(state=tk.DISABLED)

# Function to move a file to the "deleted_files" folder
def delete_file():
    selected_item =
file_listbox.get(file_listbox.curselection())
    file_path = os.path.join(current_path.get(), selected_item)
    deleted_file_path = os.path.join(deleted_files_folder,
selected_item)
    try:
        os.rename(file_path, deleted_file_path)
        refresh_display(current_path.get())
        log_operation("Delete", file_path)
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

```
# Function to refresh the display
def refresh_display(new_path):
    current_path.set(new_path)
    file_listbox.delete(0, tk.END) #delete all contents
    for item in list_directory(new_path):
        file_listbox.insert(tk.END, item)

# Function to log file operations
def log_operation(operation, *paths):
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
    with open(log_file, "a") as log:
        log.write(f"{timestamp} - {operation}: {' |
'.join(paths)}\n")

# Function to retrieve a file from the "deleted_files" folder
def retrieve_file():
    selected_indices = deleted_files_listbox.curselection()
    if selected_indices:
        selected_item =
deleted_files_listbox.get(selected_indices[0])
        source_path = os.path.join(deleted_files_folder,
selected_item)
        destination_path = os.path.join(current_path.get(),
selected_item)
        try:
            os.rename(source_path, destination_path)
            refresh_deleted_files_list()
        except Exception as e:
            messagebox.showerror("Error", str(e))
```

```
# Function to permanently delete a file from the "deleted_files"
folder
def delete_permanently():
    selected_indices = deleted_files_listbox.curselection()
    if selected_indices:
        selected_item =
deleted_files_listbox.get(selected_indices[0])
        file_path = os.path.join(deleted_files_folder,
selected_item)
        try:
            os.remove(file_path)
            refresh_deleted_files_list()
        except Exception as e:
            messagebox.showerror("Error", str(e))
# Function to open the journal window and display log file
operations
def open_journal_window():
    if hasattr(open_journal_window, 'journal_window') and
open_journal_window.journal_window:
        open_journal_window.journal_window.deiconify()
        journal_text.delete(1.0, tk.END)
        try:
            with open(log_file, "r") as log:
                log_contents = log.read()
            journal_text.insert(tk.END, log_contents)
        except Exception as e:
            journal_text.insert(tk.END, f"Error reading log
file: {str(e)}")
# Function to move up one directory
def move_up_directory():
    current_dir = os.path.abspath(current_path.get())
    parent_dir = os.path.dirname(current_dir)
    refresh_display(parent_dir)

# Function to refresh the deleted files list
def refresh_deleted_files_list():
    deleted_files_listbox.delete(0, tk.END)
    for item in list_directory(deleted_files_folder):
        deleted_files_listbox.insert(tk.END, item)

# Function to open the "Deleted Files" window
def open_deleted_files_window():
    deleted_files_window.deiconify()
    refresh_deleted_files_list()

# Create the main window
root = tk.Tk()
root.title("File Manager")

# Initialize the current path to the user's home directory
current_path = tk.StringVar()
current_path.set(os.path.expanduser("~"))

# Create a label to display the current path
path_label = tk.Label(root, textvariable=current_path)
path_label.pack()

# Create a listbox to display the files and folders
file_listbox = tk.Listbox(root, selectmode=tk.SINGLE)
file_listbox.pack()

# Bind double-click event to navigate_directory function
file_listbox.bind("<Double-1>", navigate_directory)
```

```
# Create buttons for file operations
button_frame = tk.Frame(root)
copy_button = tk.Button(button_frame, text="Copy",
command=copy_file)
delete_button = tk.Button(button_frame, text="Delete",
command=delete_file)
move_button = tk.Button(button_frame, text="Move",
command=move_file)
journal_button = tk.Button(button_frame, text="Journal",
command=open_journal_window)
move_up_button=tk.Button(button_frame,text="Move
up",command=move_up_directory);

copy_button.pack(side=tk.LEFT)
delete_button.pack(side=tk.LEFT)
move_button.pack(side=tk.LEFT)
journal_button.pack(side=tk.LEFT)
move_up_button.pack(side=tk.LEFT)
button_frame.pack()

# Create a "Deleted Files" button
deleted_files_button = tk.Button(root, text="Deleted Files",
command=open_deleted_files_window)
deleted_files_button.pack()

# Initial file list
refresh_display(current_path.get())

# Create the "Deleted Files" window
deleted_files_window = tk.Toplevel(root)
deleted_files_window.title("Deleted Files")
deleted_files_window.withdraw()
```

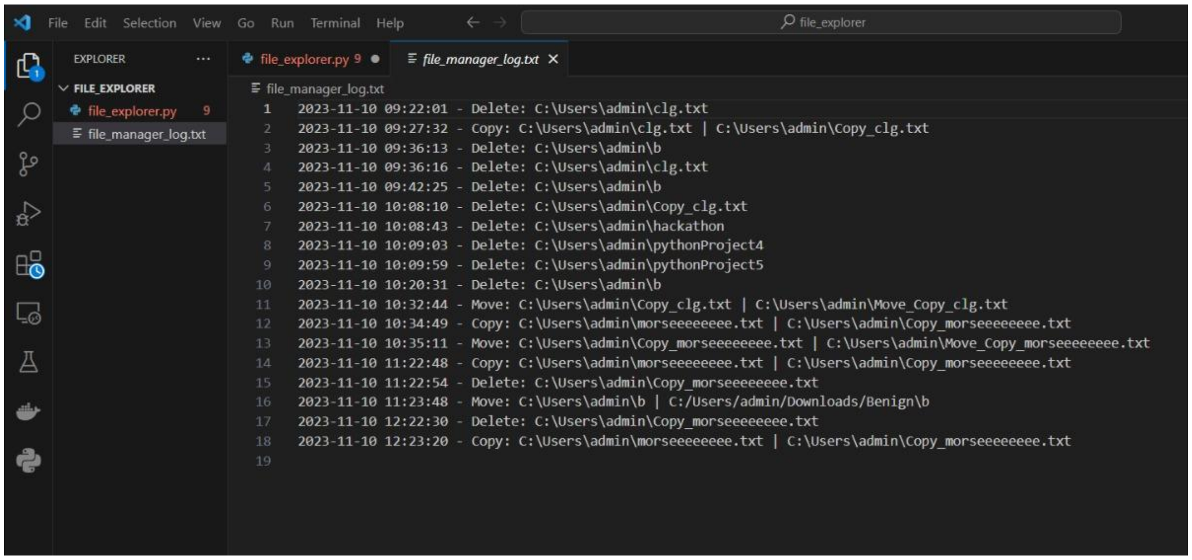
```
# Create a listbox to display deleted files
deleted_files_listbox = tk.Listbox(deleted_files_window,
selectmode=tk.SINGLE)
deleted_files_listbox.pack()

# Create buttons for retrieval and permanent deletion
retrieve_button = tk.Button(deleted_files_window,
text="Retrieve", command=retrieve_file)
delete_permanent_button = tk.Button(deleted_files_window,
text="Delete Permanently", command=delete_permanently)

retrieve_button.pack()
delete_permanent_button.pack()

# Function to refresh the deleted files list
def refresh_deleted_files_list():
    deleted_files_listbox.delete(0, tk.END)
    for item in list_directory(deleted_files_folder):
        deleted_files_listbox.insert(tk.END, item)

"""# Create a "Journal" button
journal_button = tk.Button(root, text="Journal",
command=open_journal_window)
journal_button.pack(side=tk.LEFT)"""

root.mainloop();
```



```
move_up_button.pack(side=tk.
frame.pack()

e a "Deleted Files" bu
_files_button = tk.But
_files_button.pack()

al file list
_display(current_path

e the "Deleted Files"
_files_window = tk.Top
_files_window.title("D
_files_window.withdraw

# Create a listbox to display
deleted_files_listbox = tk.Li
deleted_files_listbox.pack()

# Create buttons for retrieva
retrieve_button = tk.Button(
delete_permanent_button = tk
```

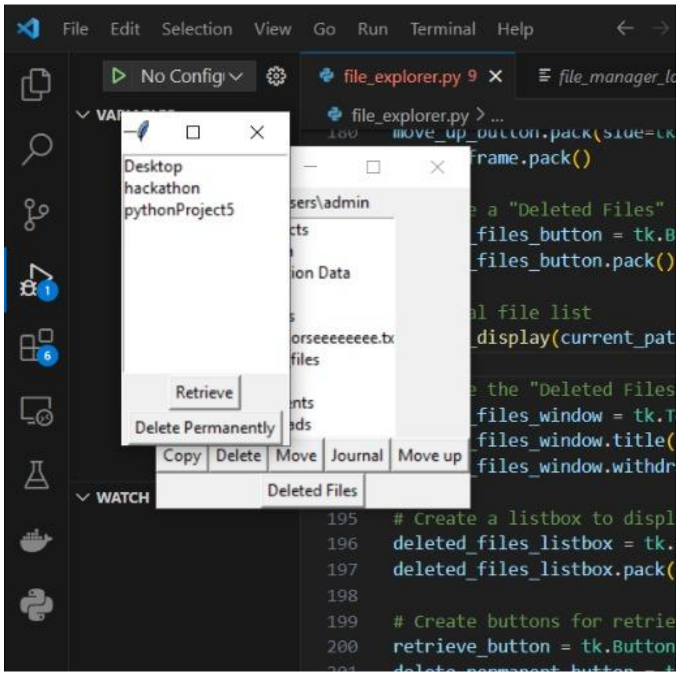
```
C:\Users\admin
3D Objects
AppData
Application Data
Benign
Contacts
Copy_morseeeeeeee.tx
deleted_files
Desktop
Documents
Downloads
```

Journaling



```
2023-11-10 09:22:01 - Delete: C:\Users\admin\clg.txt
2023-11-10 09:27:32 - Copy: C:\Users\admin\clg.txt | C:\Users\admin\Copy_clg.txt
2023-11-10 09:36:13 - Delete: C:\Users\admin\b
2023-11-10 09:36:16 - Delete: C:\Users\admin\clg.txt
2023-11-10 09:42:25 - Delete: C:\Users\admin\b
2023-11-10 10:08:10 - Delete: C:\Users\admin\Copy_clg.txt
2023-11-10 10:08:43 - Delete: C:\Users\admin\hackathon
2023-11-10 10:09:03 - Delete: C:\Users\admin\pythonProject4
2023-11-10 10:09:59 - Delete: C:\Users\admin\pythonProject5
2023-11-10 10:20:31 - Delete: C:\Users\admin\b
2023-11-10 10:32:44 - Move: C:\Users\admin\Copy_clg.txt | C:\Users\admin\Move_Copy_clg.txt
2023-11-10 10:34:49 - Copy: C:\Users\admin\morseeeeeeee.txt | C:\Users\admin\Copy_morseeeeeeee.txt
2023-11-10 10:35:11 - Move: C:\Users\admin\Copy_morseeeeeeee.txt | C:\Users\admin\Move_Copy_morseeeeeeee.txt
2023-11-10 11:22:48 - Copy: C:\Users\admin\morseeeeeeee.txt | C:\Users\admin\Copy_morseeeeeeee.txt
2023-11-10 11:22:54 - Delete: C:\Users\admin\Copy_morseeeeeeee.txt
2023-11-10 11:23:48 - Move: C:\Users\admin\b | C:/Users/admin/Downloads/Benign\b
2023-11-10 12:22:30 - Delete: C:\Users\admin\Copy_morseeeeeeee.txt
2023-11-10 12:23:20 - Copy: C:\Users\admin\morseeeeeeee.txt | C:\Users\admin\Copy_morseeeeeeee.txt
```

File retrieving



```
# Create a listbox to displ
deleted_files_listbox = tk.
deleted_files_listbox.pack(

# Create buttons for retrie
retrieve_button = tk.Button
```