Abstract

The ***PDF Search App*** is a Python-based desktop application designed to facilitate thesearch for specific keywords within PDF files located in a selected folder. Leveraging ***PyMuPDF*** for efficient PDF handling and ***Tkinter*** for a user-friendly GUI, the app allows users to easily select a folder, enter a search term, and initiate the search. Results are displayed in a clear tabular format, showing file paths and keyword counts. Users can directly open PDF files from the search results by double-clicking on the file path. The app features real-time progress updates, including a progress bar and loading animation, to keep users informed about the search status. Additionally, it provides error handling to notify users if no PDFs are found or if no keyword matches occur.

**The PDF Search *App aims to streamline the process of finding specific content within multiple PDF files, enhancing user productivity and efficiency.***

1. **Introduction**

**1.1Project Overview**

The *PDF Search App* is a *Python-based* desktop application.

Designed to facilitate the search for specific content or keywords within PDF files stored in a specified folder. This application features a user-friendly graphical interface, displays search results in a tabular format, and includes functionalities such as a progress bar, a loading animation, and the ability to open PDF files directly from the interface.

* 1. **Objectives**
* To develop a tool that can efficiently search for text within multiple PDF files.
* To provide a simple and intuitive GUI for users to interact with the application.
* To display search results in a clear and organized manner.
* To allow users to open PDF files directly from the search results.

**2. Technologies Used**

* ***Python***: The core programming language used for developing the application.
* ***PyMuPDF (fitz)*:** A library for handling and reading PDF files.
* ***Tkinte*r**: The standard Python interface to the Tk GUI toolkit.
* ***threading***: Python library used to handle background operations and improve UI responsiveness.
* ***re***: Regular expression library used for the search functionality.

**3. System Requirements**

* ***Operating System***: Windows, macOS, or Linux
* ***Python Version***: Python 3.6 or higher
* ***Libraries***:
  + *PyMuPDF*
  + *Tkinter*
  + *threading*
  + *re*
* **About PyMuPDF library**

PyMuPDF, also known as the fitz library, is a powerful Python binding for MuPDF, a lightweight PDF, XPS, and E-book viewer. It enables developers to interact with PDF documents and other supported formats programmatically. PyMuPDF provides extensive capabilities for reading, extracting, and manipulating PDF files. Users can extract text, images, and metadata from PDFs, allowing for robust document analysis and content retrieval. Additionally, the library supports various annotation types, form field manipulation, and document editing tasks like merging or splitting pages.

One of the key advantages of PyMuPDF is its performance efficiency; it is optimized for speed and low memory usage, making it suitable for processing large documents. Its API is intuitive and well-documented, facilitating ease of use for both beginners and experienced developers. Overall, PyMuPDF is a versatile tool for handling PDF documents in Python, offering a comprehensive set of features for a wide range of applications.

**4. Installation Instructions**

**4.1 Clone The Repository**

*git clone https://github.com/Robin-009/pdf-search-app.git*

*cd pdf-search-app*

**4.1 Install Required Libraries**

*pip install pymupdf tkinter*

5. File Directory

pdf-search-app/

│

├── gui/

│ └── app.py # Contains the GUI logic

│

├── pdf\_utils/

│ └── search.py # Contains the PDF search logic

│

├── main.py # Entry point of the application

│

└── README.md # Project documentation

6. Code Files

6.1 main.py

from gui.app import run\_app

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

    run\_app()

6.1 gui/app.py

import os

import tkinter as tk

from tkinter import filedialog, ttk, messagebox

from pdf\_utils.search import search\_pdfs\_in\_folder

import threading

import webbrowser

# FUnction to open the folder

def open\_folder(folder\_path\_var, folder\_path\_entry):

    folder\_path = filedialog.askdirectory()

    if folder\_path:

        folder\_path\_var.set(folder\_path)

        folder\_path\_entry.config(state=tk.NORMAL)

        folder\_path\_entry.delete(0, tk.END)

        folder\_path\_entry.insert(0, folder\_path)

        folder\_path\_entry.config(state=tk.DISABLED)

def search\_pdfs(search\_entry, folder\_path\_var, tree, progress\_var, progress\_label):

    folder\_path = folder\_path\_var.get()

    if not folder\_path:

        messagebox.showerror("Error", "Please select a folder first.")

        return

    search\_term = search\_entry.get()

    if not search\_term:

        messagebox.showerror("Error", "Please enter a search term.")

        return

    # Clear existing results

    for item in tree.get\_children():

        tree.delete(item)

    # Start a new thread for the search operation

    threading.Thread(target=perform\_search, args=(folder\_path, search\_term, tree, progress\_var, progress\_label)).start()

def perform\_search(folder\_path, search\_term, tree, progress\_var, progress\_label):

    results = search\_pdfs\_in\_folder(folder\_path, search\_term, progress\_callback=lambda idx, total: update\_progress(idx, total, progress\_var, progress\_label))

    if not results:

        progress\_label.after(0, lambda: messagebox.showinfo("No PDFs Found", "No PDF files containing the search term were found in the selected folder."))

        return

    # Insert new results

    for file\_path, count in results.items():

        tree.insert("", tk.END, values=(file\_path, count))

    # Hide progress bar when done

    progress\_var.set(0)

    progress\_label.after(0, lambda: progress\_label.config(text=""))

    # Updating the progress

def update\_progress(current, total, progress\_var, progress\_label):

    progress = (current / total) \* 100

    progress\_var.set(progress)

    progress\_label.after(0, lambda: progress\_label.config(text=f"Processing: {current} of {total} files"))

    # Open the file by double clicking (in the default view selected by the user)

def open\_file(event):

    selected\_item = tree.selection()[0]

    file\_path = tree.item(selected\_item, 'values')[0]

    if os.path.exists(file\_path):

        webbrowser.open\_new(file\_path)

    else:

        messagebox.showerror("Error", f"File not found: {file\_path}")

def run\_app():

    app = tk.Tk()

    app.title("PDF Folder Content Search")

    app.geometry("800x600")

    style = ttk.Style()

    style.configure("TLabel", font=("Arial", 12))

    style.configure("TButton", font=("Arial", 12))

    style.configure("TEntry", font=("Arial", 12))

    # Configure Treeview Style

    style.configure("Treeview.Heading",

                    font=("Arial", 12, "bold"),

                    background="#003366",

                    foreground="blue",

                    relief="flat")

    style.configure("Treeview",

                    font=("Arial", 12),

                    rowheight=25,

                    background="white",

                    foreground="black")

    style.map("Treeview", background=[("selected", "#003366")], foreground=[("selected", "white")])

    # Title label

    title\_label = tk.Label(app, text="PDF SEARCH APP", font=("Arial", 24, "bold"), fg="#003366")

    title\_label.pack(pady=20)

    frame = tk.Frame(app)

    frame.pack(pady=10)

    folder\_path\_var = tk.StringVar()

    search\_label = ttk.Label(frame, text="Search Term:")

    search\_label.grid(row=0, column=0, padx=10)

    search\_entry = ttk.Entry(frame, width=50)

    search\_entry.grid(row=0, column=1, padx=10)

    # Adjust the position of the Open Folder button

    folder\_path\_label = ttk.Label(frame, text="Selected Folder:")

    folder\_path\_label.grid(row=1, column=0, padx=10, pady=10)

    folder\_path\_entry = ttk.Entry(frame, width=50, textvariable=folder\_path\_var, state=tk.DISABLED)

    folder\_path\_entry.grid(row=1, column=1, padx=10, pady=10)

    open\_folder\_button = ttk.Button(frame, text="Open Folder", command=lambda: open\_folder(folder\_path\_var, folder\_path\_entry))

    open\_folder\_button.grid(row=1, column=2, padx=10, pady=10)

    search\_button = ttk.Button(frame, text="Search", command=lambda: search\_pdfs(search\_entry, folder\_path\_var, tree, progress\_var, progress\_label))

    search\_button.grid(row=0, column=2, padx=10)

    tree\_frame = tk.Frame(app)

    tree\_frame.pack(pady=20, fill=tk.BOTH, expand=True)

    columns = ("File Path", "Keyword Count")

    global tree

    tree = ttk.Treeview(tree\_frame, columns=columns, show="headings", selectmode="browse")

    tree.heading("File Path", text="File Path", anchor="center")

    tree.heading("Keyword Count", text="Keyword Count", anchor="center")

    tree.column("File Path", width=500, anchor="w")

    tree.column("Keyword Count", width=100, anchor="center")

    tree.grid(row=0, column=0, sticky='nsew')

    scrollbar = ttk.Scrollbar(tree\_frame, orient="vertical", command=tree.yview)

    tree.configure(yscroll=scrollbar.set)

    scrollbar.grid(row=0, column=1, sticky='ns')

    tree\_frame.grid\_rowconfigure(0, weight=1)

    tree\_frame.grid\_columnconfigure(0, weight=1)

    progress\_var = tk.DoubleVar()

    progress\_bar = ttk.Progressbar(app, variable=progress\_var, maximum=100)

    progress\_bar.pack(pady=10, padx=20, fill=tk.X)

    progress\_label = ttk.Label(app, text="")

    progress\_label.pack()

    # Bind the treeview to open file on double-click

    tree.bind("<Double-1>", open\_file)

    app.mainloop()

6.3 pdf\_utils/search.py

import os

import re

from pdf\_utils.extract import extract\_text\_from\_pdf

# Searches for a term in the text extracted from a PDF.

def search\_text\_in\_pdf(text\_list, search\_term, pattern):

    search\_results = {}

    for page\_num, page\_text in enumerate(text\_list):

        if pattern.search(page\_text):

            search\_results[page\_num + 1] = page\_text

    return search\_results

# Searches for a term in all PDF files within a specified folder and its subfolders.

def search\_pdfs\_in\_folder(folder\_path, search\_term, progress\_callback=None):

    results = {}

    pdf\_files = []

    # Define a regular expression pattern for the search term as a whole word

    pattern = re.compile(r'\b' + re.escape(search\_term) + r'\b', re.IGNORECASE)

    # Collect all PDF files in the specified folder and subfolders

    for root, \_, files in os.walk(folder\_path):

        for file in files:

            if file.lower().endswith('.pdf'):

                pdf\_files.append(os.path.join(root, file))

    total\_files = len(pdf\_files)

    # Process each PDF file

    for idx, file\_path in enumerate(pdf\_files):

        if progress\_callback:

            try:

                progress\_callback(idx + 1, total\_files)

            except Exception as e:

                print(f"Progress callback error: {e}")

        try:

            # Extract text from the PDF file

            text\_list = extract\_text\_from\_pdf(file\_path)

            # Search for the term in the extracted text

            search\_results = search\_text\_in\_pdf(text\_list, search\_term, pattern)

            # Count the occurrences of the search term as whole words

            if search\_results:

                count = sum(len(pattern.findall(page\_text)) for page\_text in text\_list)

                results[file\_path] = count

        except Exception as e:

            print(f"Error processing file {file\_path}: {e}")

    return results

6.4 pdf\_utils/extract.py

import fitz  # PyMuPDF

def extract\_text\_from\_pdf(pdf\_path):

    pdf\_document = fitz.open(pdf\_path)

    text = []

    for page\_num in range(len(pdf\_document)):

        page = pdf\_document.load\_page(page\_num)

        text.append(page.get\_text())

    return text

7. Features

 ***PDF Search***: Searches for a user-defined keyword within all PDF files in a selected folder.

 ***Graphical User Interface***: Provides an easy-to-use interface for users to interact with the application.

 ***Results Table***: Displays search results in a tabular format with file paths and keyword counts.

 ***File Opening***: Allows users to open PDF files directly by double-clicking the file path in the results table.

 **Progress *Bar***: Displays a progress bar and a loading animation while searching through the PDFs.

 ***Error Handling***: Notifies users if no PDFs are found in the selected folder or if no keyword matches are found.

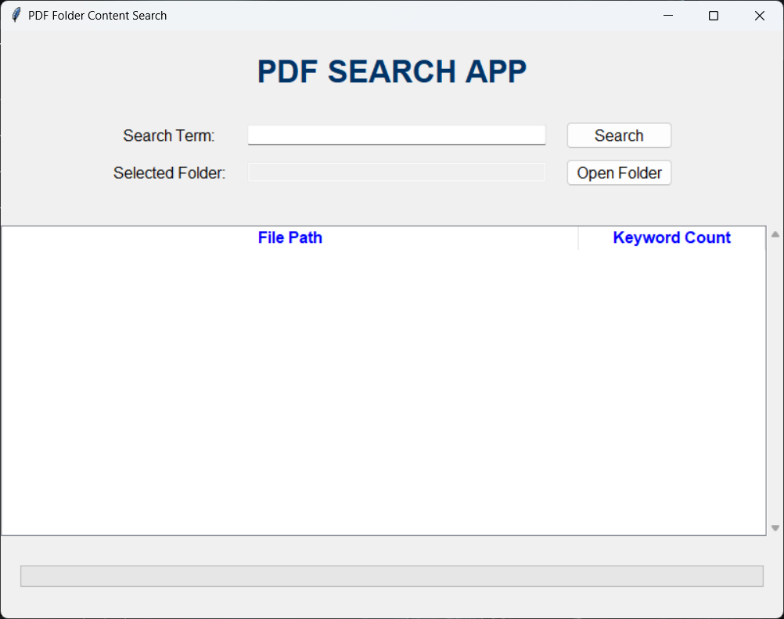
**8. Usage Instructions**

* + Run the application by executing main.py.

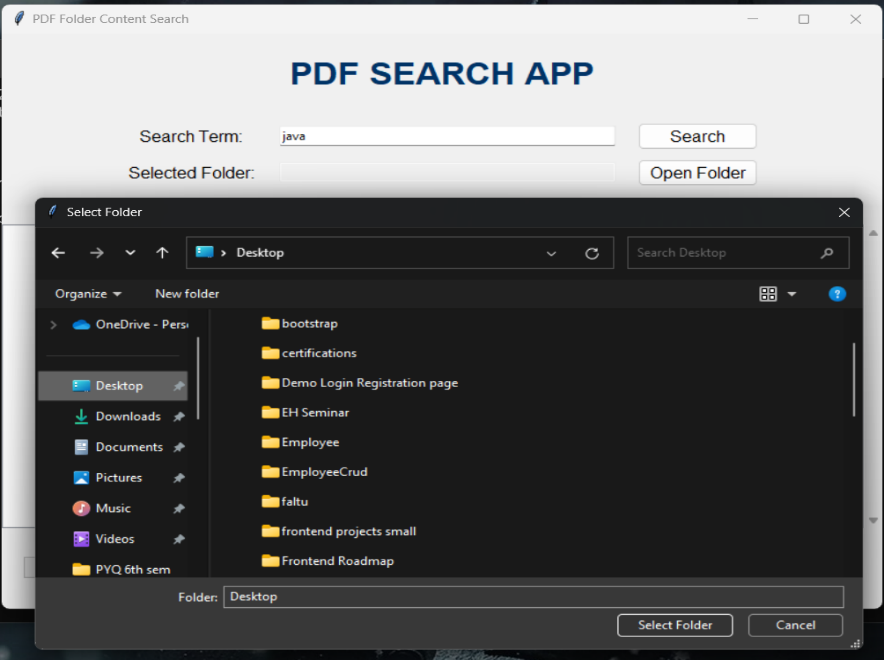
*python main.py*

* + Click the "***Open Folder***" button to select the folder containing your PDF files.
  + Enter the search term in the provided input box.
  + Click the "***Search***" button to start the search process.
  + View the search results in the table.
  + Double-click a file path in the table to open the corresponding PDF file.

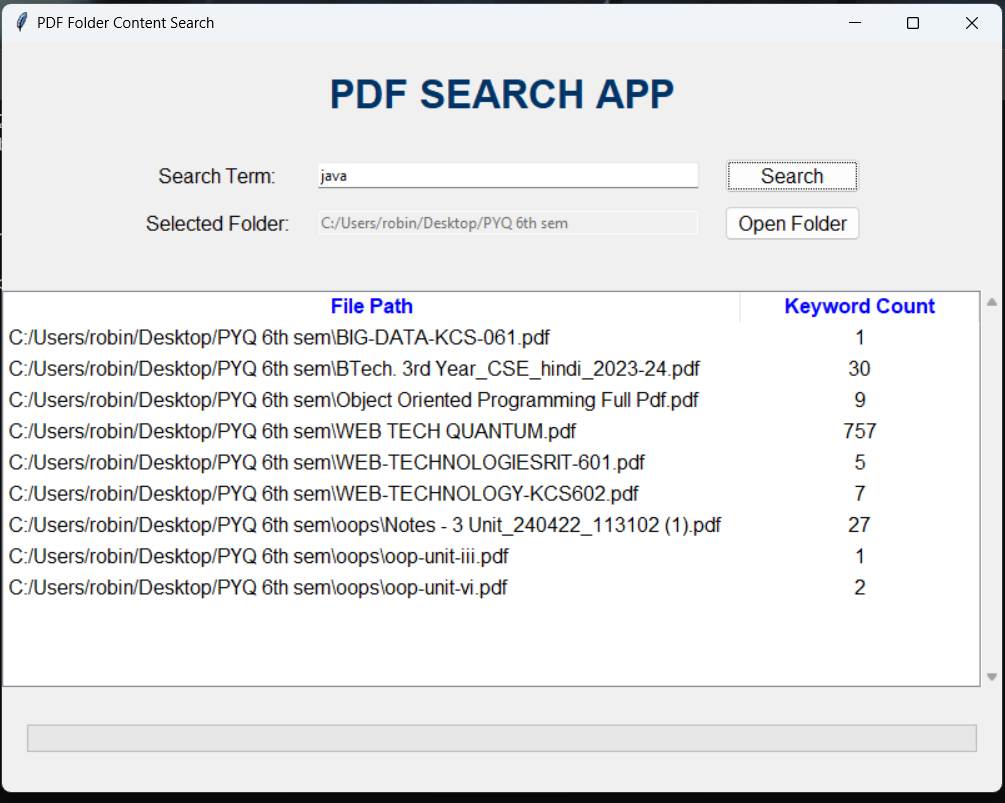
9. ScreenShots

****

1. **Interface of the APP**

****

**2. Search for the File by specifying the keyword and folder.**

****

**3. Output files are presented in tabular format.**

10. Future Scope

* ***Add Advanced Search Options***: Allow users to specify case sensitivity, search multiple keywords, etc.
* ***PDF Content Preview***: Show a preview of the text surrounding the keyword in the search results.
* ***Export Results***: Provide an option to export the search results to a CSV or Excel file.

11. Conclusion

The PDF Search App successfully demonstrates the practical application of Python in developing a user-friendly and efficient tool for searching specific content within PDF files. By integrating PyMuPDF for robust PDF handling and Tkinter for an intuitive graphical user interface, the app offers a seamless user experience. Users can easily select folders, enter search terms, and view results in a well-organized tabular format, with features such as real-time progress updates and direct file opening enhancing usability. The application's ability to handle large document sets quickly and efficiently underscores its utility in environments where document management and information retrieval are critical. The inclusion of error handling ensures reliability, making the app a valuable tool for both personal and professional use. Future enhancements could include advanced search options and export capabilities, further broadening its functionality. Overall, the PDF Search App represents a significant step towards simplifying and optimizing the process of keyword search within PDF documents.