import sounddevice as sd

import numpy as np

import wave

import os

import tkinter as tk

from tkinter import messagebox, filedialog

import threading

from scipy.fftpack import fft

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import time

from pathlib import Path

# Audio Recorder Class

class AudioRecorder:

def \_init\_(self, folder\_path="recordings"):

self.fs = 44100 # Sampling rate

self.recording = False

self.paused = False

self.frames = []

self.stream = None

self.folder\_path = 'E:/Projects/Python\_Audio\_Recorder/Recordings' # Placeholder for storing recording

self.start\_time = None

self.paused\_duration = 0 # To track the total pause duration

self.pause\_start\_time = None

# Ensure recordings folder exists

if not os.path.exists(self.folder\_path):

os.makedirs(self.folder\_path)

def start\_recording(self):

if not self.recording:

self.recording = True

self.frames = []

self.start\_time = time.time() # Start timer for duration

self.paused\_duration = 0 # Reset paused duration

self.stream = sd.InputStream(callback=self.audio\_callback, channels=1, samplerate=self.fs)

self.stream.start()

print("Recording started...")

def audio\_callback(self, indata, frames, time, status):

if self.recording and not self.paused:

self.frames.append(indata.copy())

def pause\_recording(self):

if self.recording:

if self.paused:

# Resuming recording

self.paused = False

if self.pause\_start\_time:

self.paused\_duration += time.time() - self.pause\_start\_time

print("Recording resumed.")

else:

# Pausing recording

self.paused = True

self.pause\_start\_time = time.time() # Mark the time when paused

print("Recording paused.")

def stop\_recording(self):

if self.recording:

self.recording = False

self.stream.stop()

self.stream.close()

print("Recording stopped.")

def reset\_recording(self):

if self.recording:

self.stop\_recording()

self.frames = []

print("Recording reset.")

def get\_duration(self):

if self.start\_time:

if self.paused:

return self.pause\_start\_time - self.start\_time - self.paused\_duration

else:

return time.time() - self.start\_time - self.paused\_duration

return 0

def save\_recording(self, filename):

if not self.frames:

messagebox.showwarning("Warning", "No recording to save!")

return

data = np.concatenate(self.frames, axis=0)

scaled\_data = np.int16(data / np.max(np.abs(data)) \* 32767)

filepath = os.path.join(self.folder\_path, filename)

with wave.open(filepath, 'wb') as wf:

wf.setnchannels(1)

wf.setsampwidth(2) # 16-bit audio

wf.setframerate(self.fs)

wf.writeframes(scaled\_data.tobytes())

messagebox.showinfo("Info", f"Recording saved as {filepath}")

def delete\_recording(self):

self.frames = []

messagebox.showinfo("Info", "Recording deleted.")

# GUI for the Audio Recorder

class RecorderApp:

def \_init\_(self, root):

self.root = root

self.root.title("Audio Recorder")

self.recorder = AudioRecorder()

# Main Layout

self.setup\_layout()

# Frequency Meter Setup

self.fig, self.ax = plt.subplots()

self.canvas = FigureCanvasTkAgg(self.fig, master=root)

self.canvas.get\_tk\_widget().pack()

# List all recordings on startup

self.refresh\_recordings()

# Update timer for recording duration

self.update\_duration()

def setup\_layout(self):

# Title

title = tk.Label(self.root, text="Python Audio Recorder", font=("Arial", 20))

title.pack(pady=10)

# Buttons

control\_frame = tk.Frame(self.root)

control\_frame.pack(pady=10)

self.record\_btn = tk.Button(control\_frame, text="Record", command=self.start\_recording)

self.record\_btn.grid(row=0, column=0, padx=5)

self.pause\_btn = tk.Button(control\_frame, text="Pause", command=self.pause\_recording)

self.pause\_btn.grid(row=0, column=1, padx=5)

self.stop\_btn = tk.Button(control\_frame, text="Stop", command=self.stop\_recording)

self.stop\_btn.grid(row=0, column=2, padx=5)

self.save\_btn = tk.Button(control\_frame, text="Save", command=self.save\_recording)

self.save\_btn.grid(row=0, column=3, padx=5)

self.reset\_btn = tk.Button(control\_frame, text="Reset", command=self.reset\_recording)

self.reset\_btn.grid(row=0, column=4, padx=5)

# Timer and Status Label

self.status\_label = tk.Label(self.root, text="Status: Idle | Duration: 0 s", font=("Arial", 14))

self.status\_label.pack(pady=10)

# Recordings List

self.recordings\_list = tk.Listbox(self.root, height=10)

self.recordings\_list.pack(pady=10)

# Play and Delete Buttons

list\_control\_frame = tk.Frame(self.root)

list\_control\_frame.pack(pady=5)

self.play\_btn = tk.Button(list\_control\_frame, text="Play", command=self.play\_recording)

self.play\_btn.grid(row=0, column=0, padx=5)

self.delete\_selected\_btn = tk.Button(list\_control\_frame, text="Delete", command=self.delete\_selected\_recording)

self.delete\_selected\_btn.grid(row=0, column=1, padx=5)

def start\_recording(self):

if not self.recorder.recording:

threading.Thread(target=self.recorder.start\_recording).start()

self.update\_frequency\_meter()

def pause\_recording(self):

self.recorder.pause\_recording()

def stop\_recording(self):

self.recorder.stop\_recording()

def reset\_recording(self):

self.recorder.reset\_recording()

self.status\_label.config(text="Status: Reset | Duration: 0 s")

def save\_recording(self):

if not self.recorder.frames:

messagebox.showwarning("Warning", "No recording to save!")

return

filename = filedialog.asksaveasfilename(defaultextension=".wav", filetypes=[("WAV files", "\*.wav")], initialdir=self.recorder.folder\_path)

if filename:

self.recorder.save\_recording(os.path.basename(filename))

self.refresh\_recordings()

def refresh\_recordings(self):

# Clear list

self.recordings\_list.delete(0, tk.END)

# List all saved recordings

recordings = os.listdir(self.recorder.folder\_path)

for rec in recordings:

if rec.endswith(".wav"):

self.recordings\_list.insert(tk.END, rec)

def play\_recording(self):

selected = self.recordings\_list.get(tk.ACTIVE)

if selected:

filepath = os.path.join(self.recorder.folder\_path, selected)

os.system(f"start {filepath}")

def delete\_selected\_recording(self):

selected = self.recordings\_list.get(tk.ACTIVE)

if selected:

filepath = os.path.join(self.recorder.folder\_path, selected)

os.remove(filepath)

self.refresh\_recordings()

messagebox.showinfo("Info", f"{selected} deleted.")

def update\_frequency\_meter(self):

if self.recorder.recording and not self.recorder.paused:

data = np.concatenate(self.recorder.frames, axis=0) if self.recorder.frames else np.zeros(1024)

fft\_data = fft(data)

self.ax.clear()

self.ax.plot(np.abs(fft\_data[:len(fft\_data)//2]))

self.canvas.draw()

if self.recorder.recording:

self.root.after(100, self.update\_frequency\_meter)

def update\_duration(self):

if self.recorder.recording:

duration = int(self.recorder.get\_duration())

status = "Paused" if self.recorder.paused else "Recording"

self.status\_label.config(text=f"Status: {status} | Duration: {duration} s")

else:

self.status\_label.config(text="Status: Idle | Duration: 0 s")

self.root.after(1000, self.update\_duration)

# Main

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

root = tk.Tk()

app = RecorderApp(root)

root.mainloop()

folder\_path = Path("E:\audioproject")

if folder\_path.exists():

for file in folder\_path.iterdir():

print(file.name)