SOURCE CODE

```
import tkinter as tk
from tkinter import *
from tkinter import filedialog, messagebox, ttk
from PIL import Image, ImageTk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tensorflow as tf, customtkinter as ctk, matplotlib.pyplot as plt,
sounddevice as sd, wavio, time, pygame, spotipy, webbrowser, os, requests,
librosa, numpy as np, threading
from spotipy.oauth2 import SpotifyClientCredentials
from datetime import datetime
from tkinter import font

pygame.mixer.init()

model = tf.keras.models.load_model('emotion_recognition_model.h5')
EMOTIONS = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
IMAGE_DIR = 'C:/Users/annap/project/static/'

# Spotify API credentials
client_id = 'aaaa058c985d4a0a8c4748f3e2a141ad'
client_secret = '011e082ed763413ab497194745df9a10'
redirect_uri = 'http://localhost:8888/callback'  # Set your redirect URI

sp =
spotipy.Spotify(client_credentials_manager=SpotifyClientCredentials(client_id
=client_id, client_secret=client_secret))

# Global variables
```

```
current_audio_file = None
recorded_audio_file = 'recording.wav'
image_label = None
emotion_text_label = None
invoice_text = None
waveform_canvas = None
invoice_text_widget = None  # Define invoice_text_widget as a global variable
language_entry = None
playlists = []
show_waveform_button = None
show_invoice_button = None
menu_fg_color = "#000000"  # Black color for menubar and menu items
men_fg_color = "#ffffff"
content_frame=None
predicted_emotion=None
upload_frame=None
emotion_game_map = {
    "happy": [("Snake", "http://slither.com/io"),
              ("fighting", "https://krunker.io/")],
    "sad": [("Guess the word", "https://skribbl.io/"),
            ("Tetris Effect",
"https://store.steampowered.com/app/1003590/Tetris_Effect_Connected/")],
    "angry": [("shellshock", "https://shellshock.io/"),
              ("DOOM", "https://store.steampowered.com/app/379720/DOOM/")],
    "disgust": [("Minecraft", "https://www.minecraft.net/en-us"),
            ("Overcooked! 2",
"https://store.steampowered.com/app/728880/Overcooked_2/")],
    "surprise": [("Minecraft", "https://www.minecraft.net/en-us"),
            ("Guess the word", "https://skribbl.io/"),],
```

```
    "fear": [("Snake", "http://slither.com/io"),
             ("Overcooked! 2",
"https://store.steampowered.com/app/728880/Overcooked_2/")],
    "neutral": [("Minecraft", "https://www.minecraft.net/en-us"),
             ("Overcooked! 2",
"https://store.steampowered.com/app/728880/Overcooked_2/")]
}

# Function to open the URL
def open_game_link(url):
    webbrowser.open(url)

# Function to show information message
def show_info(message):
    messagebox.showinfo("Information", message)

# Function to show error message
def show_error(message):
    messagebox.showerror("Error", message)

# Function to display recommended games
def display_game_recommendations(emotion):
    recommendations = emotion_game_map.get(emotion, [])
    for widget in game_recommendation_frame.winfo_children():
        widget.destroy()
    if recommendations:
        for game, url in recommendations:
            link = tk.Label(game_recommendation_frame, text=game, fg="blue",
cursor="hand2", width=38, heigh=4)
```

```
            link.pack(pady=5)
            link.bind("<Button-1>", lambda e, url=url: open_game_link(url))
    else:
        no_recommendation = tk.Label(game_recommendation_frame, text="No
game recommendations available.")
        no_recommendation.pack()
# Function to show waveform
def show_waveform():
    global current_audio_file, recorded_audio_file
    file_path = current_audio_file if current_audio_file else recorded_audio_file
    feature, audio, sample_rate = extract_features(file_path)
    display_waveform_window(audio, sample_rate)
# Function to display waveform in a separate window
def display_waveform_window(audio, sample_rate):
    global waveform_canvas

    try:
        # Create a new Toplevel window for waveform display
        waveform_window = tk.Toplevel()
        waveform_window.title("Waveform Display")
        waveform_window.geometry("800x600")

        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 6))
        ax1.plot(audio)
        ax1.set_title('Raw Waveform')
        ax1.set_xlabel('Sample')
        ax1.set_ylabel('Amplitude')

        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
```

```
        img = librosa.display.specshow(mfccs, x_axis='time', ax=ax2)
        fig.colorbar(img, ax=ax2)
        ax2.set_title('MFCCs')
        ax2.set_xlabel('Time')
        ax2.set_ylabel('MFCC Coefficient')

        # Draw the canvas on the Toplevel window
        waveform_canvas = FigureCanvasTkAgg(fig, master=waveform_window)
        waveform_canvas.draw()
        waveform_canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH,
expand=True)

        # Create a button to close the waveform window
        close_button = ctk.CTkButton(waveform_window, text="Close
Waveform", command=waveform_window.destroy)
        close_button.place(x=10, y=10)

    except Exception as e:
        show_error(f"Failed to display waveform: {e}")

def close_waveform():
    global waveform_canvas

    if waveform_canvas:
        waveform_canvas.get_tk_widget().destroy()
        waveform_canvas = None
    else:
        show_info("Waveform window is not currently open.")
```

```
# Function to show invoice
def show_invoice():
    global current_audio_file, recorded_audio_file
    file_path = current_audio_file if current_audio_file else recorded_audio_file
    feature, audio, sample_rate = extract_features(file_path)
    predicted_emotion =
EMOTIONS[np.argmax(model.predict(np.expand_dims(feature, axis=0)))]
    generate_invoice_text(file_path, audio, sample_rate, predicted_emotion)

# Function to generate invoice text
def generate_invoice_text(file_path, audio, sample_rate, emotion):
    global content_frame
    unique_id = str(int(time.time()))
    current_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Extract specific details for the invoice
    amplitude = np.max(audio)
    pitch = np.mean(librosa.core.pitch_tuning(librosa.core.piptrack(y=audio,
sr=sample_rate)[0]))
    duration = librosa.core.get_duration(y=audio, sr=sample_rate)

    # Create a frame to hold the invoice information
    global invoice_text_widget
    if invoice_text_widget:
        invoice_text_widget.destroy()

    invoice_frame = tk.Frame(content_frame, bd=10, relief=tk.GROOVE)
    invoice_frame.place(x=1300, y=200)
```

```
    # Constructing header text with formatted variables
    header_text = f"""
    Invoice

    Invoice Number: {unique_id}
    Date and Time: {current_date}

    Description of Services:
    ------------------------------------------------
    Item: Audio Emotion Detection
    Description: Processing and predicting emotion
    ------------------------------------------------
    """

    header_label = tk.Label(invoice_frame, text=header_text, anchor="w",
justify=tk.LEFT, font=("Arial", 14, "bold"))
    header_label.grid(row=0, column=0, pady=10, sticky="w")

    # Define labels and values for each detail
    details = [
        ("Amplitude:", f"{amplitude:.2f}"),
        ("Pitch:", f"{pitch:.2f}"),
        ("Frequency:", f"{duration:.2f} seconds"),
        ("Predicted Emotion:", emotion.capitalize())
    ]

    # Create and place details in a grid layout
    for i, (label_text, value_text) in enumerate(details):
```

```
        label = tk.Label(invoice_frame, text=label_text, anchor="w",
justify=tk.LEFT)
        label.grid(row=i+1, column=0, sticky="w")

        value_label = tk.Label(invoice_frame, text=value_text, anchor="w",
justify=tk.LEFT)
        value_label.grid(row=i+1, column=1, sticky="w")

    # Add a button to overlay on the invoice frame
    overlay_button = ctk.CTkButton(invoice_frame, text="close",
command=close_invoice)
    overlay_button.grid(row=len(details)+2, column=0, pady=10, sticky="w")  #
Adjust position as needed

    invoice_text_widget = invoice_frame

def close_invoice():
    global invoice_text_widget

    if invoice_text_widget:
        invoice_text_widget.destroy()
        invoice_text_widget = None
    else:
        show_info("Invoice display is not currently open.")

def display_image(emotion):
    global image_label, emotion_text_label, show_waveform_button,
show_invoice_button, content_frame, playlist_listbox, language_combobox,
upload_frame
```

```
    image_path = IMAGE_DIR + emotion + ".jpeg"
    image = Image.open(image_path)
    image = image.resize((300, 300), Image.LANCZOS)
    photo = ImageTk.PhotoImage(image)
    if image_label is None:
        image_label = tk.Label(upload_frame, image=photo)
        image_label.image = photo
        image_label.place(x=700, y=210)
    else:
        image_label.config(image=photo)
        image_label.image = photo
    if emotion_text_label is None:
        emotion_text_label = ctk.CTkLabel(upload_frame, text=f"Predicted
Emotion: {emotion.capitalize()}", font=("Arial", 16), bg_color=men_fg_color,
text_color=menu_fg_color)
        emotion_text_label.place(x=580, y=130)
    else:
        emotion_text_label.configure(text=f"Predicted Emotion:
{emotion.capitalize()}")

    # Ensure buttons are created and placed horizontally under the emotion image
    if show_waveform_button is None:
        show_waveform_button = ctk.CTkButton(upload_frame, text="Show
Waveform", command=show_waveform, font=("Arial", 12))
    show_waveform_button.place(x=690, y=420)  # Adjust the x, y coordinates
as needed

    if show_invoice_button is None:
```

```
        show_invoice_button = ctk.CTkButton(upload_frame, text="Show
Invoice", command=show_invoice, font=("Arial", 12))
    show_invoice_button.place(x=540, y=420)  # Adjust the x, y coordinates as
needed
    playlist_label = ctk.CTkLabel(upload_frame, text="Recommended
Playlists:", font=("Arial", 14, "bold"))
    playlist_label.place(x=120, y=180)

    language_label = ctk.CTkLabel(upload_frame, text="Select Language:",
font=("Arial", 14, "bold"))
    language_label.place(x=120, y=150)

    language_combobox = ctk.CTkComboBox(upload_frame, values=["Telugu",
"Hindi", "English", "Spanish", "French", "German", "Italian"], font=("Arial",
12))
    language_combobox.place(x=250, y=150)
    language_combobox.set("Telugu")

    scrollbar = tk.Scrollbar(upload_frame, orient=tk.VERTICAL)
    playlist_listbox = tk.Listbox(upload_frame, yscrollcommand=scrollbar.set,
font=("Arial", 12), width=38, height=10)
    scrollbar.configure(command=playlist_listbox.yview)
    for playlist in playlists:
        playlist_listbox.insert(tk.END, playlist['name'])
    playlist_listbox.place(x=150, y=270)
    scrollbar.place(x=500, y=270)
    playlist_listbox.bind("<Double-1>", open_playlist_url)
    i=Image.open("C:/Users/annap/project/static/refresh.jpg")
    ctk_image = ctk.CTkImage(light_image=i, size=(30, 30))
```

```
    refresh_button = ctk.CTkButton(upload_frame, image=ctk_image, text="",
command=refresh_playlists, font=("Arial", 12), width=30, height=30,
fg_color="white")
    refresh_button.place(x=340, y=220)
    # Enable buttons after displaying the emotion image
    show_waveform_button.configure(state=tk.NORMAL)
    show_invoice_button.configure(state=tk.NORMAL)
def get_spotify_playlists(emotion):
    try:
        selected_language = language_combobox.get()
        query = f"{emotion} {selected_language}"
        results = sp.search(q=query, type='playlist', limit=5)
        playlists = []

        for playlist in results['playlists']['items']:
            playlists.append({
                'name': playlist['name'],
                'url': playlist['external_urls']['spotify']
            })

        return playlists

    except Exception as e:
        print(f"Error retrieving playlists: {str(e)}")
        return []
# Function to extract features from an audio file
def extract_features(file_name):
    try:
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
```

```
        mfccs = np.mean(librosa.feature.mfcc(y=audio, sr=sample_rate,
n_mfcc=40).T, axis=0)
        return mfccs, audio, sample_rate
    except Exception as e:
        show_error(f"Error encountered while parsing file: {file_name}, error:
{e}")
        return None, None, None
# Function to predict emotion
def predict_emotion():
    global invoice_text, playlists, playlist_listbox, predicted_emotion
    try:
        file_path = current_audio_file if current_audio_file else
recorded_audio_file
        feature, audio, sample_rate = extract_features(file_path)
        if feature is not None:
            feature = np.expand_dims(feature, axis=0)
            prediction = model.predict(feature)
            predicted_emotion = EMOTIONS[np.argmax(prediction)]
            display_image(predicted_emotion)
            refresh_playlists()
            display_game_recommendations(predicted_emotion)
            show_waveform_button.configure(state=tk.NORMAL)  # Enable show
waveform button
            show_invoice_button.configure(state=tk.NORMAL)  # Enable show
invoice button
            return predicted_emotion  # Return the predicted emotion
        else:
            show_error("Failed to extract features from the selected file.")
            return None
```

```
    except Exception as e:
        show_error(f"Failed to predict emotion: {e}")
        return None

def refresh_playlists():
    global playlists, playlist_listbox, predicted_emotion
    #predicted_emotion = predict_emotion()
    if predicted_emotion:
        playlists = get_spotify_playlists(predicted_emotion)
        playlist_listbox.delete(0, tk.END)
        for playlist in playlists:
            playlist_listbox.insert(tk.END, playlist['name'])
# Function to open the playlist URL
def open_playlist_url(event):
    global playlists, current_audio_file, recorded_audio_file
    selection_index = playlist_listbox.curselection()
    if selection_index:
        url = playlists[selection_index[0]]['url']
        webbrowser.open_new(url)

# Function to select an audio file
def select_file():
    global current_audio_file, audio_file_label
    file_path = filedialog.askopenfilename(filetypes=[("WAV files", "*.wav")])
    if file_path:
        current_audio_file = file_path
        audio_file_label.configure(text=f"{file_path.split('/')[-1]}")

# Function to play audio
```

```
def play_audio():
    global current_audio_file, recorded_audio_file
    file_path = current_audio_file if current_audio_file else recorded_audio_file
    try:
        pygame.mixer.music.load(file_path)
        pygame.mixer.music.play()
    except Exception as e:
        show_error(f"Error playing audio: {e}")

# Function to record audio
def record_audio():
    global recorded_audio_file
    duration = 10
    samplerate = 44100
    show_info("Recording will start now. Please speak into the microphone.")
    recording = sd.rec(int(samplerate * duration), samplerate=samplerate,
channels=2)
    sd.wait()
    wavio.write(recorded_audio_file, recording, samplerate, sampwidth=2)
    show_info("Recording finished.")
    audio_file_label.configure(text="Recorded File: recording.wav")
    current_audio_file = None
    # Predict emotion automatically after recording
    refresh_playlists()



import customtkinter as ctk
from PIL import Image, ImageTk
import tkinter as tk
```

```
from tkinter import font

def open_main_gui():
    global root, middle_frame, main_frame, upload_frame

    middle_frame.pack_forget()  # Hide the middle frame
    main_frame.pack(fill="both", expand=True)  # Show the main frame

    root.title("Emotion Recognition")

root.geometry(f"{root.winfo_screenwidth()}x{root.winfo_screenheight()}+0+0
")

    # Load images once
    menu_items_top = [
        ("Home", "C:/Users/annap/project/static/home.jpeg"),
        ("Abstract", "C:/Users/annap/project/static/abstract.jpeg"),
        ("About Us", "C:/Users/annap/project/static/About.jpeg"),
        ("Project", "C:/Users/annap/project/static/project.jpg"),
        ("Help", "C:/Users/annap/project/static/settings.jpeg"),

    ]
    menu_items_bottom = [
        ("Conclusion", "C:/Users/annap/project/static/conclusion.jpeg"),
        ("Exit", "C:/Users/annap/project/static/exit.jpg"),
    ]

    frame = ctk.CTkFrame(main_frame, fg_color=menu_fg_color)
    frame.pack(fill="y", side="left")
```

```
    profile_image = ctk.CTkLabel(frame, text="🔵", font=("Arial", 40),
fg_color=menu_fg_color, text_color="white")
    profile_image.pack(pady=20)
    app_name = ctk.CTkLabel(frame, text="EmotiVoice", font=("Arial", 20),
fg_color=menu_fg_color, text_color="white")
    app_name.pack(pady=10)

    labels = []
    active_text = ctk.StringVar()
    content_frame = ctk.CTkFrame(main_frame, fg_color=men_fg_color)
    content_frame.pack(fill="both", expand=True, padx=20, pady=20)

    def on_menu_click(item):
        for label in labels:
            label[1].configure(fg_color=menu_fg_color, text_color="white")
        item[1].configure(fg_color="grey", text_color="black", width=70)
        active_text.set(item[1].cget("text"))

        page_name = item[1].cget("text")
        for widget in content_frame.winfo_children():
            widget.destroy()

        if page_name == "Home":
            label = ctk.CTkLabel(content_frame, text="Home", justify="left",
font=("Times New Roman", 28, "bold"))
            label.place(x=10, y=40)
            custom_font = ctk.CTkFont(family="Times New Roman", size=18)
            label = ctk.CTkLabel(content_frame, text="", font=custom_font,
justify="left", anchor="nw")
```

```
        label.pack(pady=80, padx=40, fill='both', expand=True)
        tk_font = font.Font(family="Times New Roman", size=20)

        def auto_type(text, idx=0, current_text=""):
            try:
                if idx < len(text):
                    new_char = text[idx]
                    current_line = current_text.split('\n')[-1]
                    current_line_width = tk_font.measure(current_line + new_char)
                    if current_line_width > label.winfo_width():
                        current_text += '\n'
                    current_text += new_char
                    label.configure(text=current_text)
                    idx += 1
                    content_frame.after(100, auto_type, text, idx, current_text)
            except tk.TclError:
                # Handle TclError if label or other widgets are destroyed
prematurely
                pass

        auto_type("""Welcome to EmotiVoice!
                This innovative project harnesses the power of emotion
recognition from voice to enhance your daily experiences. By analyzing the
emotional nuances in your speech, EmotiVoice can accurately detect your mood
and emotional state. Whether you're feeling joyful, melancholic, energetic, or
relaxed, EmotiVoice uses this information to recommend personalized playlists
that perfectly match your current mood. Moreover, it suggests engaging games
tailored to lift your spirits or help you unwind, ensuring an immersive and
emotionally responsive user experience. Dive into a world where your emotions
```

```
guide your entertainment choices, making every interaction with EmotiVoice
uniquely attuned to your feelings.""")

    elif page_name == "Abstract":
        ctk.CTkLabel(content_frame, text="Abstract", font=("Times New
Roman", 28, "bold")).pack(pady=40)
        abstract_text = """
        Emotion plays a crucial role in human communication, significantly
impacting our interactions and decision-making processes. This project explores
the innovative integration of Speech Emotion Recognition (SER) technology
with personalized entertainment recommendations. By analyzing vocal inputs,
the system identifies the speaker's emotional state using advanced machine
learning algorithms. Once the emotion is detected, the system recommends
tailored music playlists and games that align with the user's current mood. This
fusion of SER with personalized entertainment aims to enhance user
experiences by providing emotionally congruent content, fostering emotional
well-being, and potentially offering therapeutic benefits. Our approach
leverages a robust dataset for training the SER model and employs collaborative
filtering techniques for recommendation, ensuring relevance and
personalization. This project not only showcases the potential of emotion-aware
technologies but also paves the way for more empathetic and responsive digital
environments.
        """
        text_widget = ctk.CTkLabel(content_frame, text=abstract_text.strip(),
wraplength=800, justify="left", font=("Times New Roman", 20))
        text_widget.pack(pady=20, padx=10)
        image_path = "C:/Users/annap/project/static/ab.jpg"
        original_image = Image.open(image_path)
        resized_image = original_image.resize((400, 300), Image.LANCZOS)
```

```
            photo = ImageTk.PhotoImage(resized_image)
            label = ctk.CTkLabel(content_frame, image=photo, text="")
            label.image = photo
            label.pack()

        elif page_name == "Project":
            global current_audio_file, game_recommendation_frame, upload_frame
            ctk.CTkLabel(content_frame, text="Project", font=("Arial",
20)).pack(pady=20)
            upload_frame = ctk.CTkFrame(content_frame, width=1250, height=650,
fg_color="white", border_color="black", border_width=2)
            upload_frame.pack(pady=20, padx=20)
            upload_frame.pack_propagate(False)

            j = Image.open("C:/Users/annap/project/static/upload.jpg")
            ctk_image1 = ctk.CTkImage(light_image=j, size=(30, 30))
            upload_button = ctk.CTkButton(upload_frame, text="",
image=ctk_image1, command=select_file, fg_color="white", width=20)
            upload_button.place(x=720, y=10)

            predict_button = ctk.CTkButton(upload_frame, text="predict",
command=predict_emotion)
            predict_button.place(x=550, y=70)

            k = Image.open("C:/Users/annap/project/static/play.png")
            ctk_image2 = ctk.CTkImage(light_image=k, size=(30, 30))
            play_button = ctk.CTkButton(upload_frame, text="",
image=ctk_image2, command=play_audio, fg_color="white", width=20)
            play_button.place(y=60, x=770)
```

```
        l = Image.open("C:/Users/annap/project/static/record.png")
        ctk_image3 = ctk.CTkImage(light_image=l, size=(30, 30))
        button_record = ctk.CTkButton(upload_frame, text="",
image=ctk_image3, command=record_audio, fg_color="white", width=20)
        button_record.place(y=10, x=770)

        global audio_file_label
        audio_file_label = ctk.CTkLabel(upload_frame, text="No files
selected", text_color="black")
        audio_file_label.pack(pady=20)

        game_recommendation_frame = tk.Frame(content_frame, bd=10)
        game_recommendation_frame.place(x=1250, y=650)

    elif page_name == "Conclusion":
        ctk.CTkLabel(content_frame, text="Conclusion", font=("Times New
Roman", 28, "bold")).pack(pady=40)
        conclusion_text = """
        The integration of Speech Emotion Recognition (SER) technology with
personalized playlist and game recommendations marks a significant
advancement in creating emotionally intelligent digital environments. By
accurately analyzing vocal inputs to detect user emotions, our system offers
tailored content that aligns with the user's current mood, thereby enhancing the
overall user experience and contributing to emotional well-being. The
combination of robust machine learning algorithms for emotion detection and
collaborative filtering techniques for recommendation ensures that the content
provided is both relevant and personalized, addressing the unique emotional
states and preferences of each user.This project demonstrates the transformative
```

potential of emotion-aware technologies in various applications, from entertainment to mental health support. By offering emotionally congruent content, such systems can enhance user engagement, aid in stress relief, and support mood regulation. Future developments could expand the range of emotions recognized and further refine recommendation algorithms, deepening personalization and relevance. Our work underscores the importance of integrating empathy into technology design, paving the way for more responsive and human-centered digital experiences.

"""

text_widget = ctk.CTkLabel(content_frame, text=conclusion_text.strip(), wraplength=800, justify="left", font=("Times New Roman", 20))

text_widget.pack(pady=20, padx=10)

image_path = "C:/Users/annap/project/static/refresh.jpg"

original_image = Image.open(image_path)

resized_image = original_image.resize((400, 300), Image.LANCZOS)

photo = ImageTk.PhotoImage(resized_image)

label = ctk.CTkLabel(content_frame, image=photo, text="")

label.image = photo

label.pack()

elif page_name == "About Us":

ctk.CTkLabel(content_frame, text="About Us", font=("Times New Roman", 28, "bold")).pack(pady=40)

about_us_text = """

At EmotiVoice, we are passionate about creating technology that understands and responds to human emotions. Our multidisciplinary team brings together expertise in machine learning, speech processing, and user experience design to develop innovative solutions that enhance everyday

interactions. We believe in the power of empathy-driven technology to improve well-being and foster deeper connections between people and their digital environments. Our mission is to push the boundaries of what's possible in emotion recognition and personalized recommendations, making technology more intuitive, responsive, and attuned to the emotional nuances of human communication.

"""

text_widget = ctk.CTkLabel(content_frame, text=about_us_text.strip(), wraplength=800, justify="left", font=("Times New Roman", 20))

text_widget.pack(pady=20, padx=10)

image_path = "C:/Users/annap/project/static/refresh.jpg"

original_image = Image.open(image_path)

resized_image = original_image.resize((400, 300), Image.LANCZOS)

photo = ImageTk.PhotoImage(resized_image)

label = ctk.CTkLabel(content_frame, image=photo, text="")

label.image = photo

label.pack()

elif page_name == "Help":

ctk.CTkLabel(content_frame, text="Help", font=("Arial", 20)).pack(pady=20)

Define colors and styles

bg_color = "#000080"

card_color = "#FFFFFF"

form_color = "#99D0F0"

Set the background color

#content_frame.configure(bg=bg_color)

Create frames for the contact details with specified width and height in the constructor

Create the contact form frame with increased width and height in the constructor

```
        form_frame = ctk.CTkFrame(content_frame, fg_color=bg_color, width=600, height=300)
        form_frame.place(x=500, y=200)
```

Add form fields

```
        ctk.CTkLabel(form_frame, text="Contact Us", fg_color=bg_color, text_color="white", font=("Arial", 28)).pack(pady=30)
        name_entry = ctk.CTkEntry(form_frame, placeholder_text="Enter your Name", width=300)
        name_entry.pack(padx=20,pady=5)
        email_entry = ctk.CTkEntry(form_frame, placeholder_text="Enter a valid email address", width=300)
        email_entry.pack(padx=70,pady=5)
        message_entry = ctk.CTkTextbox(form_frame, height=150, width=300)
        message_entry.pack(padx=50,pady=15)
```

Function to clear the form entries and show "Submitted"

```
        def clear_form():
            name_entry.delete(0, ctk.END)
            email_entry.delete(0, ctk.END)
            message_entry.delete("1.0", ctk.END)
            submitted_label.pack(pady=5)  # Show "Submitted" message
```

```
# Add submit button and bind it to clear_form function
        submit_button = ctk.CTkButton(form_frame, text="SUBMIT",
command=clear_form)
        submit_button.pack(pady=10)

# Add hidden label for "Submitted" message
        submitted_label = ctk.CTkLabel(form_frame, text="Submitted",
fg_color=bg_color, font=("Arial", 16), text_color="white")

        frame_office = ctk.CTkFrame(content_frame, fg_color=form_color,
width=150, height=120)
        frame_phone = ctk.CTkFrame(content_frame, fg_color=form_color,
width=150, height=120)
        frame_fax = ctk.CTkFrame(content_frame, fg_color=form_color,
width=150, height=120)
        frame_email = ctk.CTkFrame(content_frame, fg_color=form_color,
width=150, height=120)

# Place the frames
        frame_office.place(x=380, y=100)
        frame_phone.place(x=550, y=100)
        frame_fax.place(x=730, y=100)
        frame_email.place(x=910, y=100)

        def load_image(file_path, width, height):
            image = Image.open(file_path)
            image = image.resize((width, height), Image.LANCZOS)
            return ImageTk.PhotoImage(image)
        office_image_path = "C:/Users/annap/OneDrive/Desktop/location.jpeg"
```

```
        phone_image_path = "C:/Users/annap/OneDrive/Desktop/contact.jpeg"
        fax_image_path = "C:/Users/annap/OneDrive/Desktop/fax.jpeg"
        email_image_path = "C:/Users/annap/OneDrive/Desktop/mail.png"

        office_image = load_image(office_image_path, 50, 50)
        phone_image = load_image(phone_image_path, 50,50)
        fax_image = load_image(fax_image_path, 50, 50)
        email_image = load_image(email_image_path, 50, 50)

# Add labels and images to the frames
        ctk.CTkLabel(frame_office, image=office_image,text="",
justify="center", fg_color=form_color).place(x=50, y=5)
        ctk.CTkLabel(frame_phone, image=phone_image,text="",
justify="center", fg_color=form_color).place(x=60, y=5)
        ctk.CTkLabel(frame_fax, image=fax_image,text="", justify="center",
fg_color=form_color).place(x=55, y=20)
        ctk.CTkLabel(frame_email, image=email_image,text="",
justify="center", fg_color=form_color).place(x=58, y=20)


# Add labels to the frames
        ctk.CTkLabel(frame_office, text="OUR MAIN OFFICE",
justify="center", fg_color=form_color, font=("Arial", 14,
"bold")).place(x=10,y=50)
        ctk.CTkLabel(frame_office, text="SoHo 94 Broadway St\nNew York,
NY 1001", justify="center", fg_color=form_color).place(x=10,y=70)
        ctk.CTkLabel(frame_phone, text="PHONE NUMBER",
justify="center", fg_color=form_color, font=("Arial", 14,
"bold")).place(x=20,y=45)
```

```
            ctk.CTkLabel(frame_phone, text="234-9876-5400\n888-0123-4567
\n(Toll Free)", justify="center", fg_color=form_color).place(x=30,y=70)
            ctk.CTkLabel(frame_fax, text="FAX", justify="center",
fg_color=form_color, font=("Arial", 14, "bold")).place(x=60,y=60)
            ctk.CTkLabel(frame_fax, text="1-234-567-8900", justify="center",
fg_color=form_color).place(x=30,y=80)
            ctk.CTkLabel(frame_email, text="EMAIL", justify="center",
fg_color=form_color, font=("Arial", 14, "bold")).place(x=60,y=60)
            ctk.CTkLabel(frame_email, text="hello@theme.com", justify="center",
fg_color=form_color).place(x=30,y=80)



        elif page_name == "Exit":
            root.destroy()

    def on_label_enter(event, label):
        if label.cget("text") != active_text.get():
            label.configure(fg_color=menu_fg_color, text_color="yellow")

    def on_label_leave(event, label):
        if label.cget("text") != active_text.get():
            label.configure(fg_color=menu_fg_color, text_color="white")

    def create_menu_item(menu_item):
        label = ctk.CTkLabel(frame, text=menu_item[0], fg_color=menu_fg_color,
text_color="white", font=("Arial", 20), width=140, height=60,
image=ctk.CTkImage(Image.open(menu_item[1]), size=(20, 20)),
compound="left", padx=10, anchor="w", corner_radius=0)
        label.pack(fill="both", pady=5)
```

```
        label.bind("<Button-1>", lambda e: on_menu_click((menu_item, label)))
        label.bind("<Enter>", lambda e: on_label_enter(e, label))
        label.bind("<Leave>", lambda e: on_label_leave(e, label))
        return (menu_item, label)

    for item in menu_items_top + menu_items_bottom:
        labels.append(create_menu_item(item))

    on_menu_click(labels[0])

def switch_to_middle_frame():
    main_frame.pack_forget()  # Hide the main frame
    middle_frame.pack(fill="both", expand=True)  # Show the middle frame

# Create main root window
root = ctk.CTk()
root.title("Emotion Recognition")
root.geometry(f"{root.winfo_screenwidth()}x{root.winfo_screenheight()}+0+0
")



# Create frames
middle_frame = ctk.CTkFrame(root)
main_frame = ctk.CTkFrame(root)

# Create initial middle window
middle_frame.pack(fill="both", expand=True)
def close():
    root.destroy()
```

```
image_path = "C:/Users/annap/project/static/bg.png"
img = Image.open(image_path)
photo = ctk.CTkImage(light_image=img, dark_image=img,
size=(root.winfo_screenwidth(), root.winfo_screenheight()))
label = ctk.CTkLabel(middle_frame, image=photo, text="")
label.image = photo
label.place(x=0, y=0, relwidth=1, relheight=1)
middle_button = ctk.CTkButton(middle_frame, text="Proceed",
command=open_main_gui)
middle_button.place(x=620,y=600)
exit_button = ctk.CTkButton(middle_frame, text="exit", command=close)
exit_button.place(x=770,y=600)

# Show the initial frame
middle_frame.pack(fill="both", expand=True)

root.mainloop()
```