# Import necessary libraries

import wikipediaapi # For fetching data from Wikipedia

from gtts import gTTS # For text-to-speech conversion

from flask import Flask, render\_template, request, session # Flask framework for web server and session management

import os # For handling file system operations

import speech\_recognition as sr # For voice recognition from microphone input

import openai # For integrating OpenAI's GPT-3 (ChatGPT)

import pygame # For playing audio files (generated by gTTS)

import traceback # For error tracking and logging

from langdetect import detect # For detecting the input language

from googletrans import Translator # For translating between different languages

from nltk import word\_tokenize # For processing and tokenizing text

import spacy # Natural Language Processing (NLP) library for improving understanding of queries

import cv2 # OpenCV for computer vision, used to capture video from a camera

import random # For generating random emotions (simple demo)

# Initialize Flask app

app = Flask(\_\_name\_\_)

app.secret\_key = "your\_secret\_key" # Secret key for session management

# Set your OpenAI GPT-3 API key

api\_key = "YOUR\_API\_KEY" # Replace with your OpenAI API key

# Set a custom user agent for Wikipedia requests

HEADERS = {'User-Agent': 'YourUserAgent/1.0 (YourEmail@example.com)'}

wiki\_wiki = wikipediaapi.Wikipedia('en', extract\_format=wikipediaapi.ExtractFormat.WIKI, headers=HEADERS)

translator = Translator()

# Load NLP model (SpaCy)

nlp = spacy.load("en\_core\_web\_sm")

# Initialize OpenCV for computer vision

camera = cv2.VideoCapture(0) # Initialize the camera (0 refers to the first connected camera)

# Function to detect emotion (simple random emotion for demo)

def detect\_emotion():

emotions = ['Happy', 'Sad', 'Neutral', 'Confused']

return random.choice(emotions)

# Function to get a limited response from Wikipedia

def get\_limited\_wikipedia\_response(topic, max\_words=200):

page = wiki\_wiki.page(topic)

if page.exists():

summary = page.text

words = word\_tokenize(summary)

if len(words) > max\_words:

return ' '.join(words[:max\_words])

return summary

return "I couldn't find any information on that topic."

# Function to get an answer from ChatGPT using OpenAI API (with session memory)

def get\_gpt3\_answer(question):

openai.api\_key = api\_key

prompt = "The following is a conversation between a teacher and a student. The teacher is helpful and informative.\n"

# Adding previous questions and answers to the context for conversational memory

if 'conversation' in session:

prompt += session['conversation']

prompt += f"Student: {question}\nTeacher:"

response = openai.Completion.create(

engine="davinci",

prompt=prompt,

max\_tokens=150

)

answer = response['choices'][0]['text'].strip()

# Storing the conversation for memory

session['conversation'] = session.get('conversation', '') + f"Student: {question}\nTeacher: {answer}\n"

return answer

# Function to speak text using gTTS and play audio with pygame

def speak(text, language='en'):

tts = gTTS(text, lang=language)

tts.save("response.mp3")

play\_audio("response.mp3")

# Function to play an audio file using pygame

def play\_audio(file\_path):

pygame.mixer.init()

pygame.mixer.music.load(file\_path)

pygame.mixer.music.play()

while pygame.mixer.music.get\_busy():

pygame.time.Clock().tick(10)

# Function to translate the text to a specific language

def translate\_text(text, lang):

translation = translator.translate(text, dest=lang)

return translation.text

# Function to detect language of the input

def detect\_language(text):

return detect(text)

# Route to render the main webpage

@app.route('/')

def index():

session.clear() # Clear the session on a new start

return render\_template('combined.html')

# Route for Wikipedia search with multilingual support

@app.route('/test\_wikipedia\_search', methods=['POST'])

def test\_wikipedia\_search():

try:

topic = request.form['topic']

lang = detect\_language(topic) # Detect the input language

# Translate topic to English if necessary

if lang != 'en':

topic = translate\_text(topic, 'en')

topic\_answer = get\_limited\_wikipedia\_response(topic, max\_words=200)

if topic\_answer:

translated\_answer = translate\_text(topic\_answer, lang) # Translate back to user's language

speak(translated\_answer, lang)

return render\_template('combined.html', result=translated\_answer)

except Exception as e:

traceback.print\_exc()

return f"Internal Server Error: {e}", 500

# Route for voice-based ChatGPT interaction

@app.route('/test\_chatgpt\_voice', methods=['POST'])

def test\_chatgpt\_voice():

try:

recognizer = sr.Recognizer()

microphone = sr.Microphone()

with microphone as source:

recognizer.adjust\_for\_ambient\_noise(source, duration=1)

audio = recognizer.listen(source, timeout=15) # Listen for up to 15 seconds

try:

question = recognizer.recognize\_google(audio, show\_all=False).lower()

answer = get\_gpt3\_answer(question)

speak(answer)

except sr.UnknownValueError:

return "I couldn't understand the audio. Please try again."

except sr.RequestError as e:

return f"Speech recognition request error: {e}"

return render\_template('combined.html', result=answer)

except Exception as e:

traceback.print\_exc()

return f"Internal Server Error: {e}", 500

# Route for text-based ChatGPT interaction with conversational memory

@app.route('/test\_chatgpt\_text', methods=['POST'])

def test\_chatgpt\_text():

try:

question = request.form['question']

answer = get\_gpt3\_answer(question)

# Emotion detection for fun (can be expanded)

emotion = detect\_emotion()

speak(f"Your answer is {answer}. You seem {emotion}!", 'en')

return render\_template('combined.html', result=answer)

except Exception as e:

traceback.print\_exc()

return f"Internal Server Error: {e}", 500

# Start the Flask app

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

app.run(host='localhost', port=5000, debug=True)