Query_Assistant

March 25, 2025

[ ]:
```
# pip install fastapi
# pip install uvicorn
# pip install python-multipart
# pip install opencv-python
# pip install pygame
# pip install google.generativeai
# pip install SpeechRecognition
# pip install gtts
```

[ ]:
```
import os
import sys
import threading
import time
from io import BytesIO
from PIL import Image
import base64
import cv2
import json
import pygame
import google.generativeai as genai
from google.generativeai.types import HarmCategory, HarmBlockThreshold
import tkinter as tk
from tkinter import filedialog, scrolledtext, ttk
import speech_recognition as sr
from gtts import gTTS
```

[ ]:
```
# GEMINI_API_KEY=""
```

[ ]:
```
class GeminiLiveApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Gemini Live Image Assistant")
        self.root.geometry("1050x700")
        self.setup_ui()

        # self.api_key = os.environ.get("GEMINI_API_KEY", "")
        self.api_key="GEMINI_API_KEY"
        if not self.api_key:
            self.log("Please set GEMINI_API_KEY environment variable")
            return

        genai.configure(api_key=self.api_key)

        self.model = genai.GenerativeModel(
            model_name="gemini-1.5-pro",
            generation_config={
                "temperature": 0.4,
                "top_p": 0.95,
                "top_k": 0,
            },
            safety_settings={
                HarmCategory.HARM_CATEGORY_HATE_SPEECH: HarmBlockThreshold.
↪BLOCK_MEDIUM_AND_ABOVE,
                HarmCategory.HARM_CATEGORY_HARASSMENT: HarmBlockThreshold.
↪BLOCK_MEDIUM_AND_ABOVE,
                HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT:␣
↪HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
                HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT:␣
↪HarmBlockThreshold.BLOCK_MEDIUM_AND_ABOVE,
            }
        )

        self.session = None
        self.image_path = None
        self.image_data = None
        self.recognizer = sr.Recognizer()
        self.speaking = False
        self.listening = False

    def setup_ui(self):
        top_frame = ttk.Frame(self.root)
        top_frame.pack(fill=tk.X, padx=10, pady=10)

        middle_frame = ttk.Frame(self.root)
        middle_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        bottom_frame = ttk.Frame(self.root)
        bottom_frame.pack(fill=tk.X, padx=20, pady=10)

        ttk.Button(top_frame, text="Upload Image", command=self.load_image).
↪pack(side=tk.LEFT, padx=5)
        ttk.Button(top_frame, text="Capture from Camera", command=self.
↪capture_image).pack(side=tk.LEFT, padx=5)
```

```
        self.image_label = ttk.Label(top_frame, text="No image selected")
        self.image_label.pack(side=tk.LEFT, padx=20)
        ttk.Button(top_frame, text="Start New Session", command=self.
↪start_session).pack(side=tk.RIGHT, padx=5)

        self.chat_display = scrolledtext.ScrolledText(middle_frame, wrap=tk.
↪WORD, state='disabled', height=20)
        self.chat_display.pack(fill=tk.BOTH, expand=True)

        self.user_input = ttk.Entry(bottom_frame, width=60)
        self.user_input.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=5)
        self.user_input.bind("<Return>", lambda event: self.send_message())

        send_button = ttk.Button(bottom_frame, text="Send", command=self.
↪send_message)
        send_button.pack(side=tk.LEFT, padx=3)

        voice_button = ttk.Button(bottom_frame, text="Speak", width=5,␣
↪command=self.toggle_voice_input)
        voice_button.pack(side=tk.LEFT, padx=10)

        self.status_label = ttk.Label(self.root, text="Ready")
        self.status_label.pack(side=tk.BOTTOM, pady=5)

    def load_image(self):
        file_path = filedialog.askopenfilename(filetypes=[
            ("Image files", "*.jpg *.jpeg *.png *.bmp *.gif")
        ])
        if file_path:
            self.image_path = file_path
            self.image_label.config(text=f"Image: {os.path.
↪basename(file_path)}")
            self.log(f"Loaded image: {os.path.basename(file_path)}. Press START␣
↪NEW SESSION")

            try:
                with open(file_path, "rb") as image_file:
                    self.image_data = image_file.read()
            except Exception as e:
                self.log(f"Error loading image: {str(e)}")
                self.image_data = None

    def capture_image(self):
        self.log("Initializing camera...")
        try:
            cap = cv2.VideoCapture(0)
            if not cap.isOpened():
                self.log("Could not open camera")
                return

            ret, frame = cap.read()
            if not ret:
                self.log("Failed to capture image")
                cap.release()
                return

            temp_file = "temp_capture.jpg"
            cv2.imwrite(temp_file, frame)
            cap.release()

            self.image_path = temp_file
            self.image_label.config(text=f"Image: Camera Capture")
            self.log("Image captured from camera")

            with open(temp_file, "rb") as image_file:
                self.image_data = image_file.read()

        except Exception as e:
            self.log(f"Camera error: {str(e)}")
            self.image_data = None

    def start_session(self):
        if not self.api_key:
            self.log("Please set API key first")
            return

        if not self.image_data:
            self.log("Please select or capture an image first")
            return

        try:
            self.session = self.model.start_chat(history=[])
            self.log("New session started")

            image_parts = [
                {
                    "inline_data": {
                        "mime_type": "image/jpeg",
                        "data": base64.b64encode(self.image_data).
↪decode("utf-8")
                    }
                },
                {
                    "text": "This is the image I want to discuss. Please␣
↪acknowledge receipt but wait for my specific questions."
                }
            ]

            response = self.session.send_message(image_parts)
            self.display_message("You", "Image uploaded", "blue")
            self.display_message("Gemini", response.text, "green")

        except Exception as e:
            self.log(f"Error starting session: {str(e)}")

    def send_message(self):
        user_text = self.user_input.get().strip()
        if not user_text:
            return

        if not self.session:
            self.log("Please start a session first")
            return

        self.user_input.delete(0, tk.END)
        self.display_message("You", user_text, "blue")

        self.status_label.config(text="Processing...")

        threading.Thread(target=self.process_message, args=(user_text,),␣
↪daemon=True).start()

    def process_message(self, user_text):
        try:
            response = self.session.send_message(user_text)
            full_response = response.text
            displayed_text = ""

            for i in range(len(full_response) + 1):
                if i < len(full_response):
                    displayed_text = full_response[:i+1]
                    self.root.after(0, lambda text=displayed_text: self.
↪update_response(text))
                    time.sleep(0.01)  # Controls the "typing" speed

            if self.speaking:
                self.speak_text(full_response)

            self.status_label.config(text="Ready")
        except Exception as e:
            self.log(f"Error processing message: {str(e)}")
            self.status_label.config(text="Error")

    def update_response(self, text):
        self.chat_display.config(state='normal')
        content = self.chat_display.get(1.0, tk.END)
        if "Gemini:" in content:
            last_idx = content.rfind("Gemini:")
            self.chat_display.delete(f"1.0 + {last_idx}c", tk.END)
            self.chat_display.insert(tk.END, f"Gemini: {text}\n\n")
        else:
            self.chat_display.insert(tk.END, f"Gemini: {text}\n\n")
        self.chat_display.config(state='disabled')
        self.chat_display.see(tk.END)

    def display_message(self, sender, message, color):
        self.chat_display.config(state='normal')
        self.chat_display.insert(tk.END, f"{sender}: ", color)
        self.chat_display.insert(tk.END, f"{message}\n\n")
        self.chat_display.config(state='disabled')
        self.chat_display.see(tk.END)

    def log(self, message):
        print(message)
        self.status_label.config(text=message)

    def toggle_voice_input(self):
        if self.listening:
            self.listening = False
            self.status_label.config(text="Voice input stopped")
        else:
            self.listening = True
            self.status_label.config(text="Listening...")
            threading.Thread(target=self.listen_for_speech, daemon=True).start()

    def listen_for_speech(self):
        with sr.Microphone() as source:
            self.recognizer.adjust_for_ambient_noise(source)
            try:
                audio = self.recognizer.listen(source, timeout=5)
                self.status_label.config(text="Processing speech...")
                try:
                    text = self.recognizer.recognize_google(audio)
                    self.root.after(0, lambda: self.user_input.insert(0, text))
                    self.root.after(100, self.send_message)
                except sr.UnknownValueError:
                    self.status_label.config(text="Could not understand audio")
                except sr.RequestError:
                    self.status_label.config(text="Speech service unavailable")
            except Exception as e:
                self.status_label.config(text=f"Error: {str(e)}")

            self.listening = False

    def speak_text(self, text):
        try:
            tts = gTTS(text=text, lang='en')
            audio_file = "response.mp3"
            tts.save(audio_file)

            pygame.mixer.init()
            pygame.mixer.music.load(audio_file)
            pygame.mixer.music.play()
            while pygame.mixer.music.get_busy():
                pygame.time.Clock().tick(10)

            os.remove(audio_file)
        except Exception as e:
            self.log(f"Error in text-to-speech: {str(e)}")
```

[ ]:
```
if __name__ == "__main__":
    root = tk.Tk()
    app = GeminiLiveApp(root)
    root.mainloop()
```

[ ]:
```
```