Nishanth Raju

[nraju2024@fau.edu](mailto:nraju2024@fau.edu)

App URL: <https://project-3-ks5zqe362q-ue.a.run.app/>

GCP Project Name: conversational AI project

GCP Project ID: leafy-unity-436523-a9

# Introduction

This project is a web-based application that allows users to interact with Google Cloud’s vertexAI API. The primary goal is to provide a user interface where users can:

1. Upload audio and have it transcribed into text.
2. Implement sentiment analysis on that.

# Architecture

### **High-Level Architecture**:

* **Frontend**: A simple HTML interface to handle audio recording and uploading.
* **Backend**: A Flask application that interfaces with Google Cloud’s VertexAI api for transcription generation and sentiment analysis.
* **Google Cloud Services**:
  + VertexAI api
* **File Storage**: Files (audio) are recorded and uploaded by the user to the cloud.

# Implementation:

### **Frontend**:

* The page includes a button for recording, uploading the audio file and analysis.

**Backend**:

* The backend is a Flask web application hosted on Google Cloud.
* The audio files are recorded and uploaded by the user from their machine to the vertexAI api.

### **Google Cloud API**:

* The application communicates with Google’s API via service account credentials. The application uses these credentials to authenticate and interact with the API without requiring API keys in the frontend

# Pros and Cons

### **Pros**:

* **Scalability**: The application uses Google Cloud, which means the APIs can handle substantial traffic and workloads with high reliability.
* **Integration**: Seamless integration of api within a single platform.
* **Simplicity**: The app is easy to use, with a clean UI for recording audio and generating text/audio files.

### **Cons**:

* It is expensive to maintain over a long period of time.

# Application instructions:

* + **Uploading Audio**:
    - Click on the "Start recording" button to record and upload audio file.

# A screenshot of a computer Description automatically generated

* Press the “stop recording” button and the transcription and sentiment analysis will be shown on the webpage.

A screen shot of a computer

Description automatically generated

# Lessons learned

* + **API Integration**: learned how to integrate cloud APIs such as Google’s vertexAI api into a web application.
  + **HTML and Flask**: I gained a deeper understanding of using html for handling client-side interactions (e.g.,uploading audio) and Flask for building the backend.
  + **Debugging Web Applications**: I learned valuable skills in debugging python and Flask applications, particularly dealing with file uploads and API calls.

**App.py:**

import os

import base64

from flask import Flask, request, Response, render\_template

import vertexai

from vertexai.generative\_models import GenerativeModel, Part, SafetySetting

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

PROJECT\_ID = "leafy-unity-436523-a9"

LOCATION = "us-east1"

vertexai.init(project=PROJECT\_ID, location=LOCATION)

MODEL\_NAME = "gemini-1.5-flash-002"

GENERATION\_CONFIG = {

    "max\_output\_tokens": 8192,

    "temperature": 1,

    "top\_p": 0.95,

}

SAFETY\_SETTINGS = [

    SafetySetting(

        category=SafetySetting.HarmCategory.HARM\_CATEGORY\_HATE\_SPEECH,

        threshold=SafetySetting.HarmBlockThreshold.OFF,

    ),

    SafetySetting(

        category=SafetySetting.HarmCategory.HARM\_CATEGORY\_DANGEROUS\_CONTENT,

        threshold=SafetySetting.HarmBlockThreshold.OFF,

    ),

    SafetySetting(

        category=SafetySetting.HarmCategory.HARM\_CATEGORY\_SEXUALLY\_EXPLICIT,

        threshold=SafetySetting.HarmBlockThreshold.OFF,

    ),

    SafetySetting(

        category=SafetySetting.HarmCategory.HARM\_CATEGORY\_HARASSMENT,

        threshold=SafetySetting.HarmBlockThreshold.OFF,

    ),

]

@app.route("/")

def index():

    return render\_template("index.html")

@app.route("/analyze-audio", methods=["POST"])

def analyze\_audio():

    try:

        if "audioFile" not in request.files:

            return Response("No audio file provided", status=400, mimetype="text/plain")

        audio\_file = request.files["audioFile"]

        if audio\_file.filename == "":

            return Response("No selected file", status=400, mimetype="text/plain")

        file\_path = os.path.join("uploads", audio\_file.filename)

        os.makedirs("uploads", exist\_ok=True)

        audio\_file.save(file\_path)

        with open(file\_path, "rb") as f:

            audio\_base64 = base64.b64encode(f.read()).decode("utf-8")

        model = GenerativeModel(MODEL\_NAME)

        audio\_part = Part.from\_data(

            mime\_type="audio/wav", data=base64.b64decode(audio\_base64)

        )

        prompt = "transcribe and provide a sentiment analysis"

        responses = model.generate\_content(

            [audio\_part, prompt],

            generation\_config=GENERATION\_CONFIG,

            safety\_settings=SAFETY\_SETTINGS,

            stream=True,

        )

        response\_text = "".join(response.text for response in responses)

        os.remove(file\_path)

        return Response(response\_text, mimetype="text/plain")

    except Exception as e:

        return Response(f"Error: {e}", status=500, mimetype="text/plain")

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

    app.run(host="0.0.0.0", port=8080, debug=True)

**Index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Audio Analysis</title>

</head>

<body>

    <h1>Audio Transcription and Sentiment Analysis</h1>

    <button id="recordButton">Start Recording</button>

    <p id="statusMessage">Click the button to start recording.</p>

    <form id="uploadForm" action="/analyze-audio" method="post" enctype="multipart/form-data" style="display: none;">

        <input type="file" name="audioFile" id="audioFile" hidden>

        <button type="submit">Analyze</button>

    </form>

    <script>

        let mediaRecorder;

        let audioChunks = [];

        document.getElementById("recordButton").addEventListener("click", async () => {

            const recordButton = document.getElementById("recordButton");

            const statusMessage = document.getElementById("statusMessage");

            if (mediaRecorder && mediaRecorder.state === "recording") {

                mediaRecorder.stop();

                recordButton.textContent = "Start Recording";

                statusMessage.textContent = "Recording stopped. Uploading...";

            } else {

                // Request access to the microphone

                const stream = await navigator.mediaDevices.getUserMedia({ audio: true });

                mediaRecorder = new MediaRecorder(stream);

                mediaRecorder.ondataavailable = (event) => {

                    audioChunks.push(event.data);

                    if (mediaRecorder.state === "inactive") {

                        const audioBlob = new Blob(audioChunks, { type: "audio/wav" });

                        const formData = new FormData();

                        formData.append("audioFile", audioBlob, "recording.wav");

                        // Upload the audio blob via a POST request

                        fetch("/analyze-audio", {

                            method: "POST",

                            body: formData,

                        })

                            .then((response) => response.text())

                            .then((data) => {

                                statusMessage.textContent = `Analysis Complete: ${data}`;

                            })

                            .catch((error) => {

                                statusMessage.textContent = `Error: ${error.message}`;

                            });

                    }

                };

                // Start recording

                mediaRecorder.start();

                audioChunks = [];

                recordButton.textContent = "Stop Recording";

                statusMessage.textContent = "Recording in progress...";

            }

        });

    </script>

</body>

</html>

**Requirements.txt:**

Flask

requests

debugpy # Required for debugging.

google-cloud-speech

google-cloud-texttospeech

google-cloud-language

google-cloud-aiplatform

google-ai-generativelanguage

google-generativeai

shapely