Google Genai audio analyzer

App URL: https://genai-432569728156.us-east1.run.app/

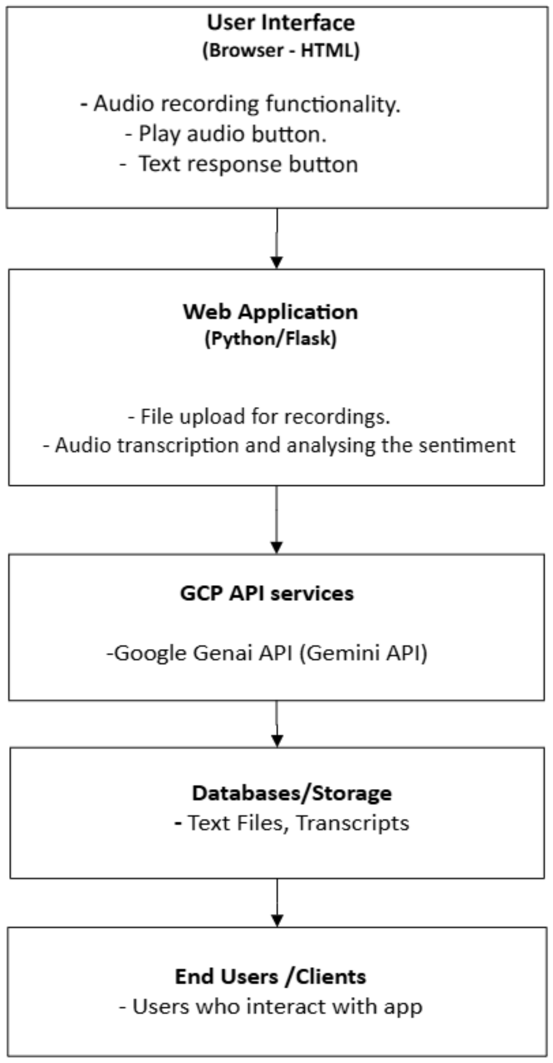
Introduction:

The application leverages Google's generative AI capabilities to create a seamless workflow for transcribing audio and performing sentiment analysis. By integrating multimodal LLMs and other cloud services, the app provides users with both textual and audio outputs for uploaded audio recordings. This project aims to streamline the process of interpreting audio inputs and extracting meaningful insights, improving accessibility and user interaction.

Goals and Objectives:

- Develop a web application utilizing advanced AI models for audio transcription and sentiment analysis.
- Enable users to upload or record audio and receive insights in both text and audio formats.
- Integrate transcription, sentiment analysis, and text-to-speech conversion for improved accessibility.
- Ensure efficient file management and deliver a user-friendly interface for seamless interaction.

Architecture:


User Interface
(Browser - HTML)
- Audio recording functionality.
- Play audio button.
- Text response button
Web Application
(Python/Flask)
- File upload for recordings.
- Audio transcription and analysing the sentiment
GCP API services
-Google Genai API (Gemini API)
Databases/Storage
- Text Files, Transcripts
End Users /Clients
- Users who interact with app

Components:

1. **Frontend**:
    - **HTML/JS Interface**: The user interface built using HTML and JavaScript enables audio recording, uploading, and interaction with the generated outputs.
    - **Dynamic Content Handling**: Uses JavaScript to fetch and display content, including transcripts and audio files.

2. **Backend**:

- **Flask Framework**: A lightweight Python web framework that serves as the backbone for routing, file processing, and API calls.
- **File Management**: Uploaded and generated files are stored in designated directories for efficient access.

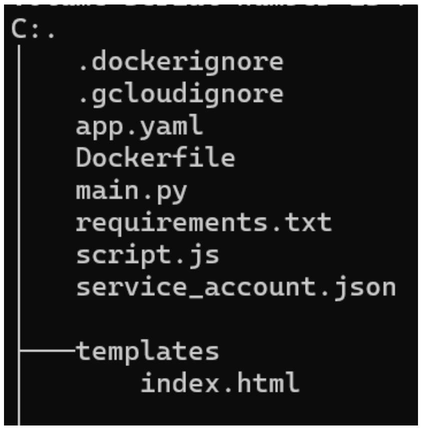
3. **Google Cloud Services**:
   - **Google Generative AI API (Gemini)**: Handles transcription and sentiment analysis in a single API call, reducing latency and simplifying workflows.
   - **Text-to-Speech API**: Converts the analyzed text and sentiment into audio files for user playback.
4. **Storage**:
   - **Directories**: Audio and processed files are stored in temporary directories (/uploads and /tts) for streamlined file access.

Workflow:

1. User uploads or records an audio file.
2. The file is securely uploaded to the server and processed through Google's Gen AI API.
3. Transcribed text and sentiment analysis are saved to a .txt file.
4. The text output is synthesized into audio using Google's TTS API.
5. Both the text and audio results are made available for download or playback on the user interface.

Implementation

File structure:



```
C:.
│   .dockerignore
│   .gcloudignore
│   app.yaml
│   Dockerfile
│   main.py
│   requirements.txt
│   script.js
│   service_account.json
│
├───templates
│       index.html
│
```

1. User Interface (HTML/JavaScript)

Code:

HTML Location: The main HTML page index.html is stored in the /templates/ directory, following Flask's template rendering system.

Features:

HTML:

Sentiment Generation: Users can input audio and generate sentiment of them.

- It dynamically displays generated text files via Flask's templating engine ({% for file in files %})..

Web Application (Flask/Python)

Code Structure:

Location: The main application logic resides in app.py, which contains the server-side routes for handling HTTP requests.

Dependencies:

Flask: To create routes and render the HTML templates.

google-generativeai: For transcription and text sentiment generation.

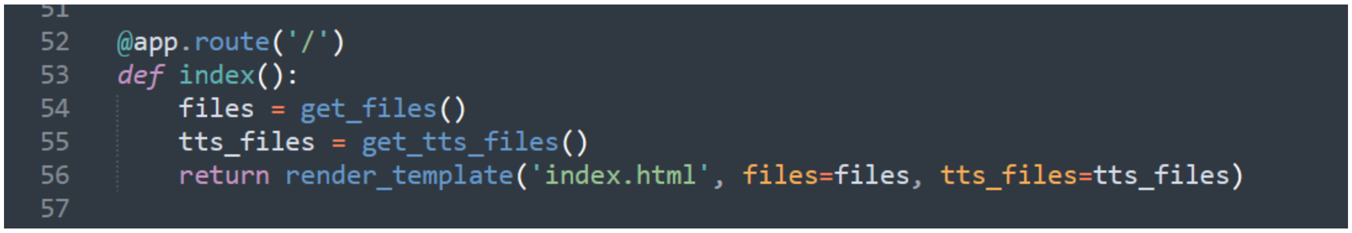
Werkzeug: For handling file uploads securely.

os, uuid, and datetime: For file handling, creating unique file names, and timestamping.

Key Flask Routes:

Home Route (/):

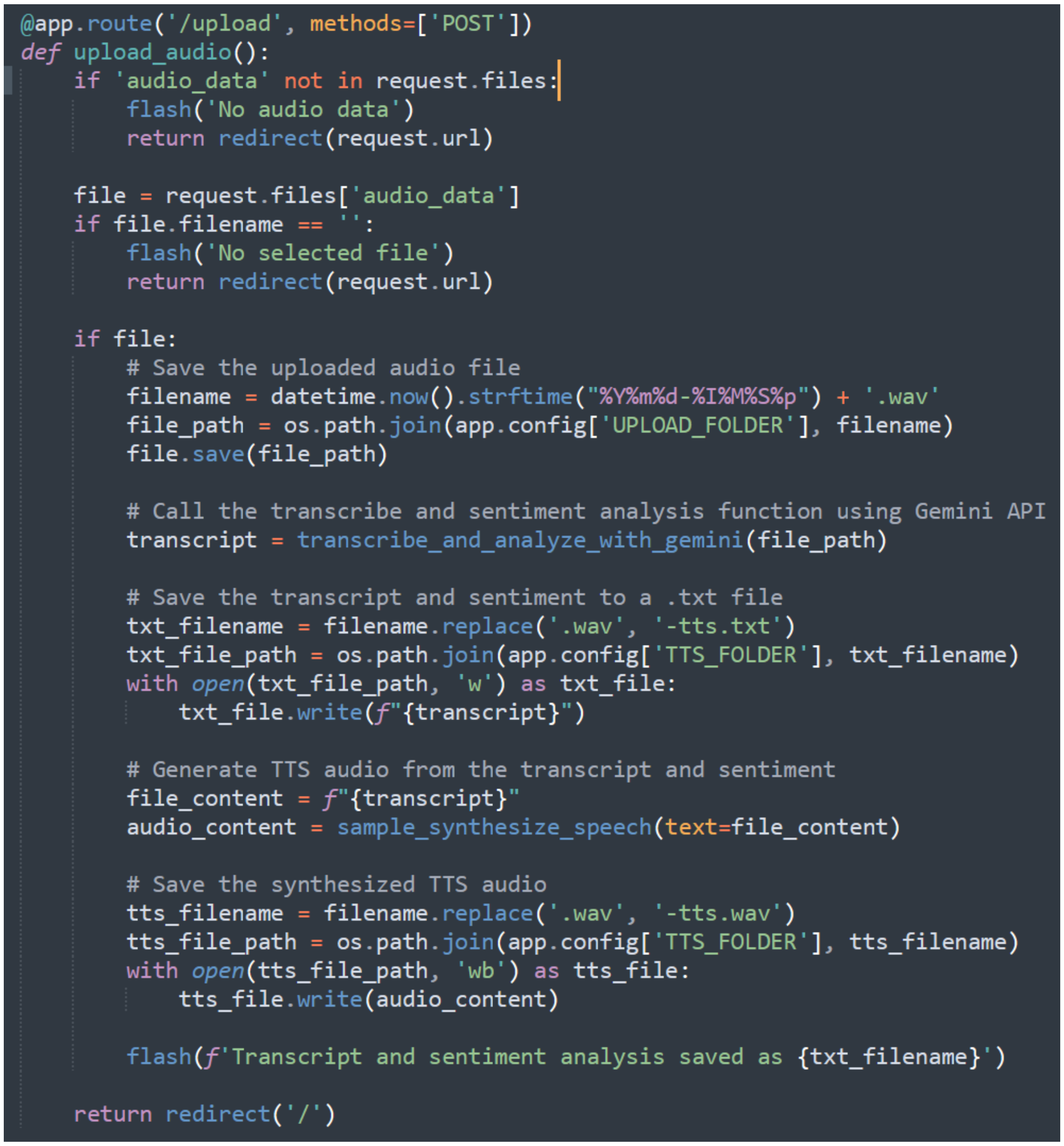
Renders the HTML page and dynamically loads uploaded and generated file.



```python
@app.route('/')
def index():
    files = get_files()
    tts_files = get_tts_files()
    return render_template('index.html', files=files, tts_files=tts_files)
```

Audio Upload Route (/upload):

- The /upload route processes user-uploaded audio files.
- Uploaded audio files are saved with unique timestamp-based filenames in the designated upload directory.
- The audio file is sent to the Google Gen AI API for transcription and sentiment analysis.
- The resulting text and sentiment are saved in a .txt file in the TTS directory.
- The text is converted into speech using the Google Text-to-Speech API, generating a .wav file.
- Both the .txt and .wav files are stored and made accessible to the user.
- Flash messages inform the user about the successful processing of their file .



```python
@app.route('/upload', methods=['POST'])
def upload_audio():
    if 'audio_data' not in request.files:
        flash('No audio data')
        return redirect(request.url)

    file = request.files['audio_data']
    if file.filename == '':
        flash('No selected file')
        return redirect(request.url)

    if file:
        # Save the uploaded audio file
        filename = datetime.now().strftime("%Y%m%d-%I%M%S%p") + '.wav'
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        # Call the transcribe and sentiment analysis function using Gemini API
        transcript = transcribe_and_analyze_with_gemini(file_path)

        # Save the transcript and sentiment to a .txt file
        txt_filename = filename.replace('.wav', '-tts.txt')
        txt_file_path = os.path.join(app.config['TTS_FOLDER'], txt_filename)
        with open(txt_file_path, 'w') as txt_file:
            txt_file.write(f"{transcript}")

        # Generate TTS audio from the transcript and sentiment
        file_content = f"{transcript}"
        audio_content = sample_synthesize_speech(text=file_content)

        # Save the synthesized TTS audio
        tts_filename = filename.replace('.wav', '-tts.wav')
        tts_file_path = os.path.join(app.config['TTS_FOLDER'], tts_filename)
        with open(tts_file_path, 'wb') as tts_file:
            tts_file.write(audio_content)

        flash(f'Transcript and sentiment analysis saved as {txt_filename}')

    return redirect('/')
```

-

GCP API Services :

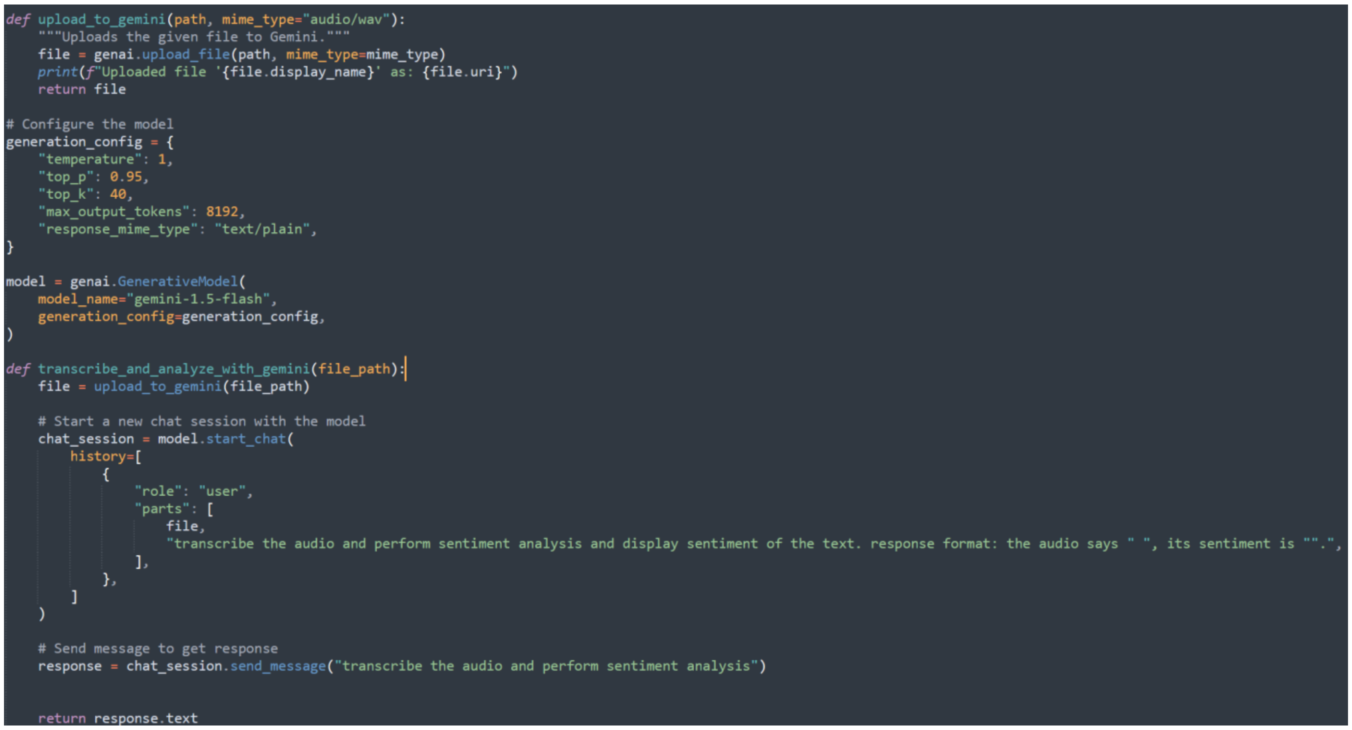
Upload Audio to Gemini:
The “upload_to_gemini” function uploads the audio file to the Gemini API, logs the file details, and returns the uploaded file object.

Configure the Model:
The generative model (gemini-1.5-flash) is set up with parameters to control output style, size, and format.

Transcription and Sentiment Analysis:

- The “transcribe_and_analyze_with_gemini” function uploads the audio file using upload_to_gemini.
- Starts a chat session with the model, instructing it to transcribe the audio and analyze its sentiment.
- Sends the request and retrieves the model’s response.

Result:
The response includes both the audio transcription and its sentiment, which is returned as a single text output.



```python
def upload_to_gemini(path, mime_type="audio/wav"):
    """Uploads the given file to Gemini."""
    file = genai.upload_file(path, mime_type=mime_type)
    print(f"Uploaded file '{file.display_name}' as: {file.uri}")
    return file

# Configure the model
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 40,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}

model = genai.GenerativeModel(
    model_name="gemini-1.5-flash",
    generation_config=generation_config,
)

def transcribe_and_analyze_with_gemini(file_path):
    file = upload_to_gemini(file_path)

    # Start a new chat session with the model
    chat_session = model.start_chat(
        history=[
            {
                "role": "user",
                "parts": [
                    file,
                    "transcribe the audio and perform sentiment analysis and display sentiment of the text. response format: the audio says " ", its sentiment is ""."",
                ],
            },
        ]
    )

    # Send message to get response
    response = chat_session.send_message("transcribe the audio and perform sentiment analysis")


    return response.text
```

Configuration:

Google Cloud Credentials: The credentials file (service_account.json) is loaded via the os.environ to authenticate API requests.

- Google Genai API key is taken from google ai studio

File and Storage Management

Storage Locations:

Generated sentiment of the text: The generated sentiment of the text are stored as .txt files in the 'uploads' folder and 'tts' folder.

Running the Application:

- The application can be started by running the following command on the command prompt:
- **Cmd** : python app.py

Pros and Cons:

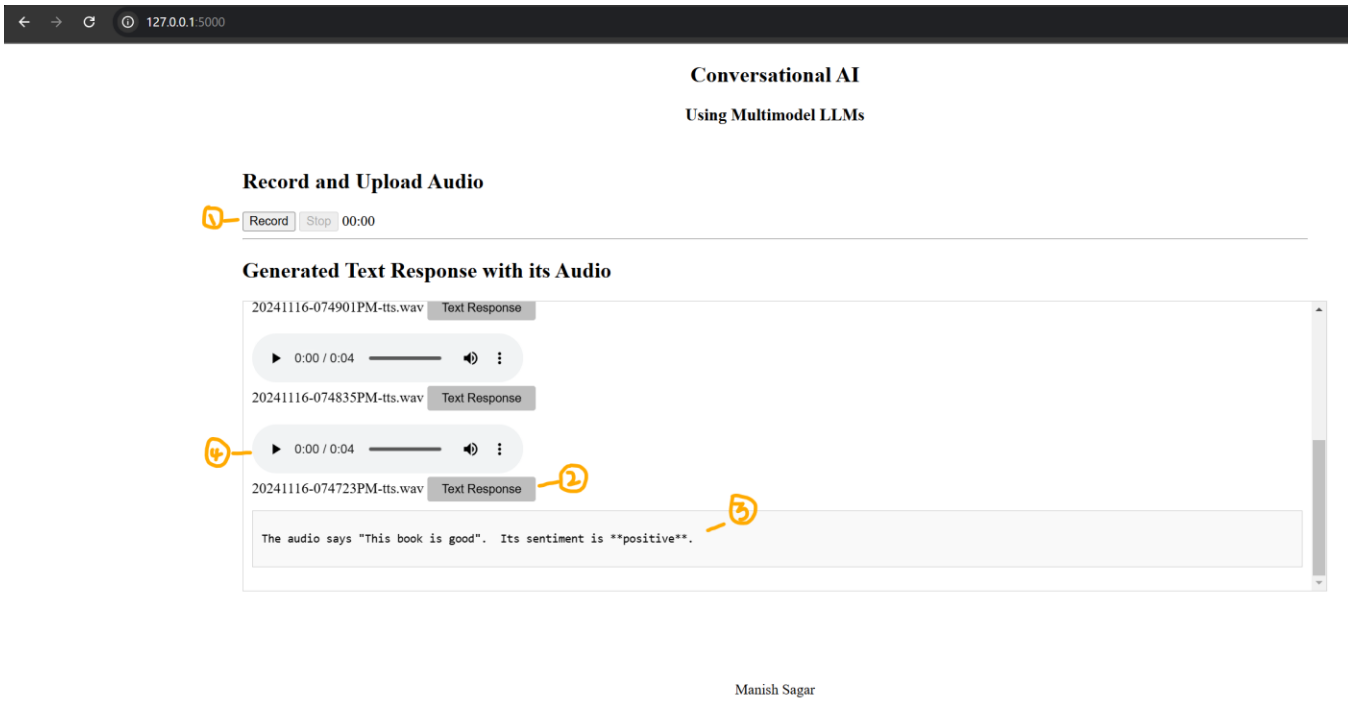
Pros:

- Integrating advanced APIs for transcription and sentiment analysis into a single workflow improved efficiency and reduced the need for multiple systems.
- The intuitive design made the application easy to navigate, ensuring accessibility for users with varying levels of technical expertise.
- Adding text-to-speech functionality enhanced inclusivity, making the application usable for individuals with visual impairments or reading difficultie

Cons:

- Relying on third-party APIs introduces risks such as service outages, changes in API structure, or cost implications for extended use.
- Setting up credentials and integrating multiple cloud services requires careful management and may pose challenges for less experienced developers.

Application instructions:



Conversational AI

Using Multimodel LLMs

Record and Upload Audio

Generated Text Response with its Audio

20241116-074901PM-tts.wav

0:00 / 0:04

20241116-074835PM-tts.wav

0:00 / 0:04

20241116-074723PM-tts.wav

```
The audio says "This book is good".  Its sentiment is **positive**.
```

Manish Sagar

Steps:

1. Use the record button to record the audio, once you record the audio press on the stop button. Now this audio is sent to the genai API for transcription and sentiment analysis.
2. To view the text response click on the "Text Response" button.
3. The text response is shown as shown in the image above.
4. To listen to the text response click on the play button as shown in the image above.

Design decisions and Lessons learned:

- In developing the application, several key design decisions were made that focused on using advanced technology like Google APIs for transcription and sentiment analysis, which simplified the workflow with one call.
- Focused on creating an intuitive interface that allows users to interact with the app effortlessly. Prioritizing simplicity and clarity in design enhances usability and ensures a positive user experience.
- Configured APIs and services with secure credentials and scalable configurations. Secure integration of external services is critical for reliability and protecting sensitive data.
- Added features to make the application more accessible, such as converting text results to audio. Accessibility considerations make the application more inclusive and improve usability for a broader audience.

Appendix:

- **App.py**

Code:

```
from datetime import datetime


from flask import Flask, render_template, request, redirect, url_for, send_file,
send_from_directory, flash

from werkzeug.utils import secure_filename

import uuid

from google.cloud import texttospeech_v1

import google.generativeai as genai

import io

import os


credential_path = r"service_account.json"

os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = credential_path


app = Flask(__name__)

app.secret_key = '11477asdf'


genai.configure(api_key=os.environ["GEMINI_API_KEY"])


client = texttospeech_v1.TextToSpeechClient()


# Configure upload folder

UPLOAD_FOLDER = '/tmp/uploads'

TTS_FOLDER = '/tmp/tts'
```

```
# Ensure the directories exist
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(TTS_FOLDER, exist_ok=True)

ALLOWED_EXTENSIONS = {'wav'}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['TTS_FOLDER'] = TTS_FOLDER

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

def allowed_file(filename):
  return '.' in filename and \
      filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def get_files(folder):
  files = [f for f in os.listdir(folder) if allowed_file(f)]
  files.sort(reverse=True)
  return files

@app.route('/')
def index():
  files = get_files(app.config['UPLOAD_FOLDER'])
  tts_files = get_files(app.config['TTS_FOLDER'])
  return render_template('index.html', files=files, tts_files=tts_files)
```

```
def upload_to_gemini(path, mime_type="audio/wav"):
    """Uploads the given file to Gemini."""
    file = genai.upload_file(path, mime_type=mime_type)
    print(f"Uploaded file '{file.display_name}' as: {file.uri}")
    return file


# Configure the model
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 40,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}


model = genai.GenerativeModel(
    model_name="gemini-1.5-flash",
    generation_config=generation_config,
)


def transcribe_and_analyze_with_gemini(file_path):
    file = upload_to_gemini(file_path)


    # Start a new chat session with the model
    chat_session = model.start_chat(
        history=[
            {
```

```
            "role": "user",
            "parts": [
              file,
              "transcribe the audio and perform sentiment analysis and display sentiment of the text.
response format: the audio says " ", its sentiment is "".",
            ],
          },
        ]
    )

    # Send message to get response
    response = chat_session.send_message("transcribe the audio and perform sentiment analysis")



    return response.text


def sample_synthesize_speech(text=None, ssml=None):
    input = texttospeech_v1.SynthesisInput()
    if ssml:
      input.ssml = ssml
    else:
      input.text = text

    voice = texttospeech_v1.VoiceSelectionParams()
    voice.language_code = "en-US"
    # voice.ssml_gender = "MALE"
    audio_config = texttospeech_v1.AudioConfig()
    audio_config.audio_encoding = "LINEAR16"

    request = texttospeech_v1.SynthesizeSpeechRequest(
        input=input,
        voice=voice,
        audio_config=audio_config,
    )

    response = client.synthesize_speech(request=request)

    return response.audio_content

@app.route('/upload', methods=['POST'])
def upload_audio():
    if 'audio_data' not in request.files:
        flash('No audio data')
        return redirect(request.url)

    file = request.files['audio_data']
    if file.filename == '':
        flash('No selected file')
        return redirect(request.url)

    if file:
        # Save the uploaded audio file
        filename = datetime.now().strftime("%Y%m%d-%I%M%S%p") + '.wav'
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        # Call the transcribe and sentiment analysis function using Gemini API
        transcript = transcribe_and_analyze_with_gemini(file_path)

        # Save the transcript and sentiment to a .txt file
        txt_filename = filename.replace('.wav', '-tts.txt')
        txt_file_path = os.path.join(app.config['TTS_FOLDER'], txt_filename)
        with open(txt_file_path, 'w') as txt_file:
            txt_file.write(f"{transcript}")

        # Generate TTS audio from the transcript and sentiment
        file_content = f"{transcript}"
        audio_content = sample_synthesize_speech(text=file_content)

        # Save the synthesized TTS audio
        tts_filename = filename.replace('.wav', '-tts.wav')
        tts_file_path = os.path.join(app.config['TTS_FOLDER'], tts_filename)
        with open(tts_file_path, 'wb') as tts_file:
            tts_file.write(audio_content)

        flash(f'Transcript and sentiment analysis saved as {txt_filename}')

    return redirect('/')

@app.route('/tts/<filename>')
```

```
def uploaded_tts_file(filename):
    return send_from_directory(app.config['TTS_FOLDER'], filename)


@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)


@app.route('/script.js',methods=['GET'])
def scripts_js():
    return send_file('./script.js')


if __name__ == '__main__':
    app.run(debug=True)
```

Index.html

Code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Audio Recorder</title>
    <style>
        /* Style the footer */
        footer {
            text-align: center;
            margin-top: 90px;
        }

        /* Style for scroller in Record and Upload section */
        .scroll-container {
          max-height: 400px;
          overflow-y: auto;
          padding-right: 10px;
        }

        .text-content {
          margin-top: 10px;
          padding: 10px;
          border: 1px solid #ccc;
          background-color: #f9f9f9;
          display: none;  /* Initially hide the content */
        }

    </style>
</head>
<body>

    <div class="container">
        <h2 style="text-align: center; margin-top: 20px;">Conversational AI</h2>
        <h3 style="text-align: center; margin-top: 20px;">Using Multimodel LLMs</h3>

        <div style="display: flex; justify-content: center;">
          <table style="width:100%; max-width: 1200px; border-collapse: collapse;">
            <tbody>
              <tr>
                <td style="width:50%; padding: 10px;">
            <h2>Record and Upload Audio</h2>
            <button id="record">Record</button>
            <button id="stop">Stop</button>
            <span id="timer">00:00</span>
            <audio id="audio"></audio>
            <form id="uploadForm" method="POST" enctype="multipart/form-data">
              <input type="hidden" name="audio_data" id="audioData">
            </form>
            <script src="script.js"></script>
            <hr>
            <h2>Generated Text Response with its Audio</h2>
            <div class="scroll-container" style="width: 100%; max-height: 300px; overflow-y:
auto; border: 1px solid #ccc; padding: 10px;">
              <ul style="list-style-type: none; padding: 0;">
                {% for tts_file in tts_files %}
                <li style="margin-bottom: 15px;">
                  <!-- Audio player for each TTS audio file -->
                  <audio controls>
                    <source src="{{ url_for('uploaded_tts_file', filename=tts_file) }}"
type="audio/wav">
                    Your browser does not support the audio element.
                  </audio><br>
                  <!-- Display the file name -->
                  {{ tts_file }}


                  <button onclick="fetchTextContent('{{ url_for('uploaded_tts_file',
filename=tts_file.replace('.wav', '.txt')) }}', this)"
```

```
                                    style="padding: 6px 16px; background-color: #BEBEBE; color: black;
border: none; border-radius: 3px;">
                                  Text Response
                                </button>

                                <!-- Text content container that will display the transcript and sentiment below
the button -->
                                <div class="text-content" id="content-{{ tts_file.replace('.wav', '') }}"></div>
                              </li>
                              {% else %}
                              <p>No Wav files generated yet.</p>
                              {% endfor %}
                            </ul>
                          </div>
                        </td>
                      </tr>
                    </tbody>
                  </table>
                </div>
              </div>

              <footer>
                <p>Manish Sagar</p>
              </footer>

              <script>
                // Function to fetch and toggle the visibility of the content of the .txt file directly below the
button
```

```
      function fetchTextContent(url, button) {
        const contentDiv = button.nextElementSibling;  // Get the associated content div

        if (contentDiv.style.display === 'block') {
          // If content is already displayed, hide it
          contentDiv.style.display = 'none';
        } else {
          // Otherwise, fetch the content and display it
          fetch(url)
            .then(response => response.text())
            .then(data => {
              contentDiv.innerHTML = `<pre>${data}</pre>`;  // Insert the content
              contentDiv.style.display = 'block';  // Show the content
            })
            .catch(error => {
              console.error('Error fetching text content:', error);
              contentDiv.innerHTML = `<p style="color: red;">Failed to load text content.</p>`;
              contentDiv.style.display = 'block';  // Show the error message
            });
        }
      }
    </script>

</body>
</html>
```

Script.js

Code:

```
const recordButton = document.getElementById('record');
const stopButton = document.getElementById('stop');
const audioElement = document.getElementById('audio');
const uploadForm = document.getElementById('uploadForm');
const audioDataInput = document.getElementById('audioData');
const timerDisplay = document.getElementById('timer');

let mediaRecorder;
let audioChunks = [];
let startTime;

function formatTime(time) {
  const minutes = Math.floor(time / 60);
  const seconds = Math.floor(time % 60);
  return `${minutes}:${seconds.toString().padStart(2, '0')}`;
}

recordButton.addEventListener('click', () => {
  navigator.mediaDevices.getUserMedia({ audio: true })
    .then(stream => {
      mediaRecorder = new MediaRecorder(stream);
      mediaRecorder.start();

      startTime = Date.now();
      let timerInterval = setInterval(() => {
        const elapsedTime = Math.floor((Date.now() - startTime) / 1000);
        timerDisplay.textContent = formatTime(elapsedTime);
      }, 1000);

      mediaRecorder.ondataavailable = e => {
        audioChunks.push(e.data);
      };

      mediaRecorder.onstop = () => {
        const audioBlob = new Blob(audioChunks, { type: 'audio/wav' });
        const formData = new FormData();
        formData.append('audio_data', audioBlob, 'recorded_audio.wav');

        fetch('/upload', {
          method: 'POST',
          body: formData
        })
        .then(response => {
          if (!response.ok) {
            throw new Error('Network response was not ok');
          }
          location.reload(); // Force refresh

          return response.text();
        })
        .then(data => {
          console.log('Audio uploaded successfully:', data);
          // Redirect to playback page or display success message
```

```
        })
        .catch(error => {
          console.error('Error uploading audio:', error);
        });
      };
    })
    .catch(error => {
      console.error('Error accessing microphone:', error);
    });

  recordButton.disabled = true;
  stopButton.disabled = false;
});

stopButton.addEventListener('click', () => {
  if (mediaRecorder) {
    mediaRecorder.stop();
  }

  recordButton.disabled = false;
  stopButton.disabled = true;
});

// Initially disable the stop button
stopButton.disabled = true;
```

- **app.yaml**

Code:

```
runtime: python39

entrypoint: gunicorn -b :$PORT main:app
```

- **Docker File**

```
FROM python:3.9.17-bookworm

ENV PYTHONUNBUFFERED True

ENV APP_HOME /convai
WORKDIR $APP_HOME
COPY . ./

RUN pip install --no-cache-dir --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt

CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0 main:app
```

- **requirements.txt**

```
Flask==2.2.2

google-api-core==2.19.1

google-auth==2.32.0

google-cloud-aiplatform==1.60.0

google-cloud-bigquery==3.25.0

google-cloud-core==2.4.1

google-cloud-resource-manager==1.12.4

google-cloud-storage==2.18.0

google-cloud-texttospeech==2.17.2

google-generativeai==0.8.3

werkzeug==2.2.2

gunicorn
```