***Project 1: Speech-to-Text & Text-to-Speech Web Application***

**Student Name: Naveen Reddy Janamma**  
Email: njanmma2023@fau.edu

**GCP Project Name**: **Project1-Speech-to-text**  
**GCP Project ID**: Project1-Speech-to-text (Link)

**Cloud Run Service Name**: speech-to-text-app  
**App URL**: <https://speech-to-text-170363883300.us-central1.run.app/>

***Introduction***

This is a web-based speech-to-text project, which runs on Google Cloud Platform (GCP). It does conversion from audio files to text and vice versa by using Google's Speech-to-Text API.

Text-to-Speech: Whatever text is provided by the user, the application will convert it to an audio file with the help of Google Text-to-Speech API. *Speech-to-Text*: Such an audio file will have been submitted for the application to transcribe it into text and show that text using Google Speech-to-Text API.

The application will have a web interface where users can do the following with Python Flask interface: *Record and upload audio*: The application must then transcribe the audio using Google's Speech-to-Text API. From text to audio conversion: The application must generate an audio file from the text based on Google's Speech-to-Text API. Both audio and text files will be saved to the server's local file system. Furthermore, the application has been developed in the Docker containerization pattern and deployed on Cloud Run serverlessly, which means it can be easily scaled and used.

***System Architecture***

The software setup includes the following components:

* Web application (Frontend): To provide users with an interface to input text for text-to-speech conversion, and upload an audio version for transcription.
* Backend API (Python/Flask): To process text-to-speech and transcription requests coming from the users along with returning the results.
* Local Storage: Storage space for audio files uploaded by the users. (Optional if local file storage should be used.
* Google Speech-to-Text API: Transcribing uploaded audio files into text.
* Google Text-to-Speech API: Converting text to audio.
* Cloud Run: Manages the backend application, automatically scaling based on traffic.

***Component Details***

* Cloud Storage: Provides temporary storage for the audio files that were uploaded before they are processed by the back-end.
* Flask Web Server: It is a Flask application that provides routes related to different front-end user requests; the requests include uploading the audio, converting the text into speech, and revealing the transcribed text in front of the users.
* Google Speech-to-Text API: Converts speech to text and is able to handle different audio formats and languages.
* Google Text-to-Speech API: These APIs are used to generate speech from a written text in the language of the choice.
* Cloud Run: It is a cloud service that provides a ready-to-scale and serverless environment for the back-end. The system scales based on traffic and charges only for the resources that are consumed.

***Implementation***

***Application Structure:***

The structure of the application follows a simple Python Flask application pattern.

* Frontend: The frontend is in basic HTML and JavaScript, where the user interacts with the app to either upload an audio to be transcribed or provides text to be synthesized.
* Backend: Flask is used for the backend where the application handles routes for both functionalities (speech-to-text and text-to-speech).

***Code and Configuration:***

***Dockerfile:***

The Dockerfile used for creating the Docker container is as follows:

FROM python:3.9-slim

WORKDIR /app

COPY . /app

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

COPY project1-speech-to-text.json /app/project1-speech-to-text.json

CMD ["python", "app.py"]

# Expose port 8080 (Cloud Run's default port)

EXPOSE 8080

*Flask API****:***

import os

from google.cloud import speech, texttospeech

import librosa

import soundfile as sf

import noisereduce as nr

from werkzeug.utils import secure\_filename

from flask import request,Flask, session, request, render\_template, send\_from\_directory, jsonify, redirect

from datetime import datetime

import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

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

app.secret\_key = 'AbcdEfgh1234'

# Configure upload folder and allowed extensions

UPLOAD\_FOLDER = 'uploads'

ALLOWED\_EXTENSIONS = {'wav', 'mp3'}

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

# Google Cloud credentials and API clients -

# from google.oauth2 import service\_account

# credentials\_path = os.getenv("GOOGLE\_APPLICATION\_CREDENTIALS", "./project1-speech-to-text.json")

# credentials = service\_account.Credentials.from\_service\_account\_file(credentials\_path)

# using IAM for deployement

from google.auth import default

from google.cloud import speech

credentials, project = default()

client = speech.SpeechClient(credentials=credentials)

speech\_client = speech.SpeechClient(credentials=credentials)

text\_to\_speech\_client = texttospeech.TextToSpeechClient(credentials=credentials)

# Helper function to check allowed file extensions

def allowed\_file(filename):

    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

# Index route

@app.route('/')

def index():

    if 'audio\_files' not in session:

        session['audio\_files'] = []

    if 'text\_files' not in session:

        session['text\_files'] = []

    return render\_template('index.html', audio\_files=session['audio\_files'], text\_files=session['text\_files'])

# Route to upload and transcribe audio

@app.route('/upload', methods=['POST'])

def upload\_audio():

    if 'audio\_data' not in request.files:

        return 'No file part', 400

    file = request.files['audio\_data']

    if file and allowed\_file(file.filename):

        filename = secure\_filename(f"{datetime.now().strftime('%Y%m%d-%I%M%S%p')}.mp3")

        file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        # Save uploaded file

        try:

            file.save(file\_path)

        except Exception as e:

            return jsonify({'error': f'Failed to save file: {e}'}), 500

        # # Perform noise reduction using librosa

        # try:

        #     # Load the audio file using librosa

        #     audio, sample\_rate = librosa.load(file\_path, sr=None)

        #     noise\_sample = audio[:sample\_rate]

        #     # Apply noise reduction (simple example using Spectral Subtraction)

        #     noise\_reduced\_audio = nr.reduce\_noise(y=audio, sr=sample\_rate, y\_noise=noise\_sample)

        #     # Save the noise-reduced audio back to the file path

        #     sf.write(file\_path, noise\_reduced\_audio, sample\_rate)

        # except Exception as e:

        #     return jsonify({'error': f'Noise reduction failed: {e}'}), 500

        # Perform speech-to-text

        try:

            with open(file\_path, 'rb') as audio\_file:

                content = audio\_file.read()

            audio = speech.RecognitionAudio(content=content)

            config = speech.RecognitionConfig(

                encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,

                language\_code="en-US"

            )

            response = speech\_client.recognize(config=config, audio=audio)

            # Save transcription

            transcription = "\n".join(result.alternatives[0].transcript for result in response.results)

            transcription\_file = f"{os.path.splitext(filename)[0]}.txt"

            transcription\_path = os.path.join(app.config['UPLOAD\_FOLDER'], transcription\_file)

            with open(transcription\_path, 'w') as f:

                f.write(transcription)

            # Update session

            session['audio\_files'].append(filename)

            session['text\_files'].append(transcription\_file)

            return render\_template('index.html', audio\_files=session['audio\_files'], text\_files=session['text\_files'])

        except Exception as e:

            return jsonify({'error': f'Speech-to-text failed: {e}'}), 500

    return jsonify({'error': 'Invalid file type'}), 400

# Route to convert text to speech

@app.route('/text\_to\_speech', methods=['POST'])

def text\_to\_speech():

    text = request.form.get('text')

    if not text:

        return 'No text provided', 400

    # Convert text to speech

    input\_text = texttospeech.SynthesisInput(text=text)

    voice = texttospeech.VoiceSelectionParams(

        language\_code="en-US",

        ssml\_gender=texttospeech.SsmlVoiceGender.NEUTRAL

    )

    audio\_config = texttospeech.AudioConfig(audio\_encoding=texttospeech.AudioEncoding.LINEAR16)

    try:

        response = text\_to\_speech\_client.synthesize\_speech(input=input\_text, voice=voice, audio\_config=audio\_config)

        filename = secure\_filename(f"tts\_{datetime.now().strftime('%Y%m%d-%I%M%S%p')}.wav")

        file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

        with open(file\_path, 'wb') as out:

            out.write(response.audio\_content)

        # Generate  transcription (or extract any relevant info if needed)

        transcription\_file = f"{os.path.splitext(filename)[0]}.txt"

        transcription\_path = os.path.join(app.config['UPLOAD\_FOLDER'], transcription\_file)

        with open(transcription\_path, 'w') as f:

            f.write(f"Text-to-speech transcription for file: {filename}\n")

            f.write(f"Original text: {text}")

        # Update session

        session['audio\_files'].append(filename)

        session['text\_files'].append(transcription\_file)

        return render\_template('index.html', audio\_files=session['audio\_files'], text\_files=session['text\_files'])

    except Exception as e:

        return jsonify({'error': f'Text-to-speech failed: {e}'}), 500

    return redirect('/')

# Route to serve uploaded files

@app.route('/uploads/<filename>')

def uploaded\_file(filename):

    if filename.endswith('.wav'):

        return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename, as\_attachment=False, mimetype='audio/wav')

    elif filename.endswith('.mp3'):

        return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename, as\_attachment=False, mimetype='audio/mpeg')

    else:

        return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename)

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

def convert\_to\_text():

    # Assuming the last uploaded file is the recorded audio

    if not session.get('audio\_files'):

        return jsonify({'error': 'No audio file to transcribe'}), 400

    audio\_file = session['audio\_files'][-1]  # Get the last uploaded file

    file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], audio\_file)

    try:

        with open(file\_path, 'rb') as audio\_file\_data:

            content = audio\_file\_data.read()

        audio = speech.RecognitionAudio(content=content)

        config = speech.RecognitionConfig(

            encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,

            language\_code="en-US"

        )

        response = speech\_client.recognize(config=config, audio=audio)

        # Extract transcription from response

        transcription = "\n".join(result.alternatives[0].transcript for result in response.results)

                # Create transcription file

        transcription\_file = f"{os.path.splitext(audio\_file)[0]}.txt"

        transcription\_path = os.path.join(app.config['UPLOAD\_FOLDER'], transcription\_file)

        with open(transcription\_path, 'w') as f:

            f.write(transcription)

        # Update session

        session['text\_files'].append(transcription\_file)

        # Return transcription as JSON

        return jsonify({'transcription': transcription})

    except Exception as e:

        return jsonify({'error': f'Error processing audio: {e}'}), 500

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=int(os.environ.get("PORT", 8080)))

*Frontend (HTML):*

The HTML file provides two forms: one for uploading audio files for transcription and another for text input to generate speech.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Speech-to-Text and Text-to-Speech</title>

    <style>

        body {

            font-family: sans-serif;

            background-color: #f0f0f0;

            display: flex;

            justify-content: center;

            align-items: center;

            min-height: 100vh;

        }

        .container {

            background-color: #fff;

            padding: 40px;

            border-radius: 5px;

            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

            width: 600px;

        }

        h1 {

            text-align: center;

            margin-bottom: 30px;

        }

        h2 {

            margin-top: 20px;

        }

        button {

            padding: 12px 20px;

            border: none;

            border-radius: 3px;

            background-color: #4CAF50;

            color: #fff;

            font-size: 16px;

            cursor: pointer;

        }

        button:hover {

            background-color: rgb(202, 134, 44);

        }

        textarea {

            width: 100%;

            padding: 10px;

            border: 1px solid #ccc;

            border-radius: 3px;

            resize: vertical;

        }

        ul {

            list-style: none;

            padding: 0;

        }

        li {

            margin-bottom: 5px;

        }

        #progressBar {

            width: 0;

            height: 20px;

            background-color: green;

            margin-top: 10px;

        }

        #timer {

            font-size: 20px;

            margin-top: 10px;

        }

        input[type="file"] {

            margin-top: 10px;

        }

        .transcription {

            margin-top: 20px;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Speech-to-Text and Text-to-Speech Application</h1>

        <!-- Section for Text-to-Speech (Enter text to convert to speech) -->

        <h2>Text-to-Speech</h2>

        <form action="/text\_to\_speech" method="POST">

            <textarea name="text" rows="4" cols="50" placeholder="Enter text here"></textarea><br>

            <button type="submit">Convert to Audio</button>

        </form>

        <!-- Section for uploading audio to transcribe -->

        <h2>Upload Audio and Transcribe</h2>

        <form action="/upload" method="POST" enctype="multipart/form-data">

            <input type="file" name="audio\_data" accept="audio/\*" required><br>

            <button type="submit">Upload and Transcribe</button>

        </form>

        <!-- Section to record audio -->

        <h2>Record Audio</h2>

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

        <button id="stop" disabled>Stop Recording</button>

        <p id="timer">0:00</p>

        <div id="progressBar"></div>

        <audio id="audio" controls></audio>

        <button id="convertToText" disabled>Convert to Text</button>

        <!-- Section to display uploaded audio files -->

        <h2>Uploaded Audio Files</h2>

        <ul>

            {% for file in audio\_files %}

                <li><a href="/uploads/{{ file }}" target="\_blank">{{ file }}</a></li>

            {% endfor %}

        </ul>

        <!-- Section to display transcriptions -->

        <h2>Transcriptions</h2>

        <ul>

            {% for file in text\_files %}

                <li><a href="/uploads/{{ file }}" target="\_blank">{{ file }}</a></li>

            {% endfor %}

        </ul>

        <!-- Display the transcription for the uploaded audio -->

        {% if transcription %}

            <div class="transcription">

                <h3>Transcription:</h3>

                <p>{{ transcription }}</p>

            </div>

        {% endif %}

    </div>

    <script src="/static/script.js"></script>

</body>

</html>

*Pros and Cons*

*Pros:*

* Interactive Web Interface: Allows users to easily convert text to speech and transcribe audio with just a few clicks.
* Serverless Deployment: By using Google Cloud Run, the app is highly scalable and cost-effective.
* Integration of Google APIs: Utilizes Google's robust speech and text APIs for reliable, accurate conversions.

*Cons:*

* Latency: Depending on the size of the audio or text, there may be some delay in generating the output, particularly when large files are uploaded.
* Dependency on Google APIs: If the Google Speech-to-Text or Text-to-Speech services experience downtime, the app will be impacted.

*Problems Encountered and Solutions*

*File Handling:*

* Problem: Ensuring secure and reliable storage of uploaded audio files was crucial. Malicious file names could potentially cause security vulnerabilities or system instability.
* Solution: The secure\_filename function from the werkzeug library was utilized to sanitize uploaded file names, preventing potential security risks and ensuring proper file storage.

*Latency in API Calls:*

* Problem: API calls to Google Cloud services (Speech-to-Text and Text-to-Speech) can introduce latency, particularly with larger audio files or heavy traffic. This latency could impact the user experience.
* Solution: Strategies to mitigate latency included:   
  Efficient File Handling: Optimizing file handling processes on the server to minimize delays in reading and processing audio data. Asynchronous Processing: Implementing asynchronous tasks for long-running API calls to avoid blocking the main application thread.

*Noise Handling in Audio Files:*

* Problem: Background noise in audio recordings significantly impacted the accuracy of speech-to-text transcription. The Google Cloud Speech-to-Text API may struggle to accurately recognize speech in noisy environments.
* Solution:  
  Noise Reduction: Before sending audio to the Speech-to-Text API, noise reduction techniques were implemented. Libraries like librosa were used to process audio and remove background noise. Audio Preprocessing: Preprocessing steps, such as noise reduction and potential audio normalization, were incorporated to improve the quality of the audio input and enhance the accuracy of transcription. Text-to-Speech Enhancement: For Text-to-Speech, ensuring high-quality and well-formatted input text contributed to generating clearer and more natural-sounding audio output.  
  Improvements.  
  Noise Handling Workflow: The integration of noise reduction techniques into the audio processing pipeline significantly improved the overall accuracy and reliability of speech-to-text transcriptions. This preprocessing step is now a crucial component of the application's workflow.

*Application Instructions*

* 1. Text to Speech: Enter your text in the text box and submit the form to generate an audio file.
* 2. Speech to Text: Upload an audio file (in WAV or MP3 format) to receive a transcription.
* The generated audio file and transcription text will be saved on the server and can be accessed via the provided links.

*Lessons Learned*

* API Integration: I gained valuable experience working with Google's Speech-to-Text and Text-to-Speech APIs.
* Web Development with Flask: I developed skills in building and deploying web applications using Flask, as well as integrating them with external APIs.
* File Management: I learned how to manage file uploads and securely store files on the server.

*Appendix*

*Dockerfile:* # (as shown above)

*Flask Application (app.py):* # (as shown above)

*Frontend (HTML):* # (as shown above)

*requirements.txt:*

Flask

google-cloud-speech

google-cloud-texttospeech

Werkzeug

librosa

noisereduce

*script.js:*

// Select DOM elements

const recordButton = document.getElementById('record');

const stopButton = document.getElementById('stop');

const convertToTextButton = document.getElementById('convertToText');

const audioElement = document.getElementById('audio');

const textToSpeechButton = document.getElementById('textToSpeech');

const textInput = document.getElementById('textInput');

const progressBar = document.getElementById('progressBar');

const timerDisplay = document.getElementById('timer');

let mediaRecorder;

let audioChunks = [];

let startTime;

let timerInterval;

// Format time for the timer

function formatTime(time) {

    const minutes = Math.floor(time / 60);

    const seconds = Math.floor(time % 60);

    return `${minutes}:${seconds.toString().padStart(2, '0')}`;

}

// Request microphone permissions and start recording

async function startRecording() {

    try {

        // Request microphone permission

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

        const audioContext = new (window.AudioContext || window.webkitAudioContext)();

        const source = audioContext.createMediaStreamSource(stream);

        // Create and apply lowpass filter for noise reduction

        const filter = audioContext.createBiquadFilter();

        filter.type = "lowpass";

        filter.frequency.value = 5000;

        source.connect(filter);

        filter.connect(audioContext.destination);

        mediaRecorder = new MediaRecorder(stream);

        mediaRecorder.start();

        startTime = Date.now();

        let elapsedTime = 0;

        timerInterval = setInterval(() => {

            elapsedTime = Math.floor((Date.now() - startTime) / 1000);

            timerDisplay.textContent = formatTime(elapsedTime);

            // Update progress bar

            const progress = Math.min(elapsedTime / 60, 1); // 1 minute max recording

            progressBar.style.width = `${progress \* 100}%`;

        }, 1000);

        mediaRecorder.ondataavailable = e => {

            audioChunks.push(e.data);

        };

        mediaRecorder.onstop = () => {

            clearInterval(timerInterval); // Stop the timer

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

            const formData = new FormData();

            formData.append('audio\_data', audioBlob, 'recorded\_audio.wav');

            // Upload the audio file to the server

            fetch('/upload', {

                method: 'POST',

                body: formData

            })

            .then(response => {

                if (!response.ok) {

                    throw new Error('Network response was not ok');

                }

                location.reload(); // Refresh to show updated files

            })

            .catch(error => {

                console.error('Error uploading audio:', error);

                alert('There was an issue uploading the audio. Please try again.');

            });

        };

    } catch (error) {

        // Handle errors, e.g., permission denied or microphone access issues

        if (error.name === 'PermissionDeniedError') {

            alert('Permission to access the microphone was denied. Please allow microphone access and try again.');

        } else {

            console.error('Error accessing microphone:', error);

            alert('Unable to access your microphone. Please check your permissions.');

        }

    }

}

// Start recording when the record button is clicked

recordButton.addEventListener('click', () => {

    startRecording();

    recordButton.disabled = true;

    stopButton.disabled = false;

    convertToTextButton.disabled = true;

});

// Stop recording when the stop button is clicked

stopButton.addEventListener('click', () => {

    if (mediaRecorder) {

        mediaRecorder.stop();

    }

    recordButton.disabled = false;

    stopButton.disabled = true;

    convertToTextButton.disabled = false;

});

// Handle text-to-speech conversion

textToSpeechButton.addEventListener('click', () => {

    const text = textInput.value.trim();

    if (!text) {

        alert("Please enter some text.");

        return;

    }

    // Show loading state

    textToSpeechButton.disabled = true;

    textToSpeechButton.textContent = 'Converting...';

    fetch('/text\_to\_speech', {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json',

        },

        body: JSON.stringify({ text: text })

    })

    .then(response => response.blob())

    .then(audioBlob => {

        // Play the generated audio

        const audioUrl = URL.createObjectURL(audioBlob);

        console.log('Audio URL:', audioUrl);

        audioElement.src = audioUrl;

        audioElement.play();

    })

    .catch(error => {

        console.error('Error during text-to-speech conversion:', error);

    })

    .finally(() => {

        // Reset button text and state

        textToSpeechButton.disabled = false;

        textToSpeechButton.textContent = 'Convert to Audio';

    });

});

// Handle convert to text (transcribe recorded audio)

convertToTextButton.addEventListener('click', () => {

    fetch('/convert\_to\_text', {

        method: 'POST',

    })

    .then(response => response.json())

    .then(data => {

        if (data.transcription) {

            alert('Transcription: ' + data.transcription);

        } else {

            alert('No transcription available.');

        }

    })

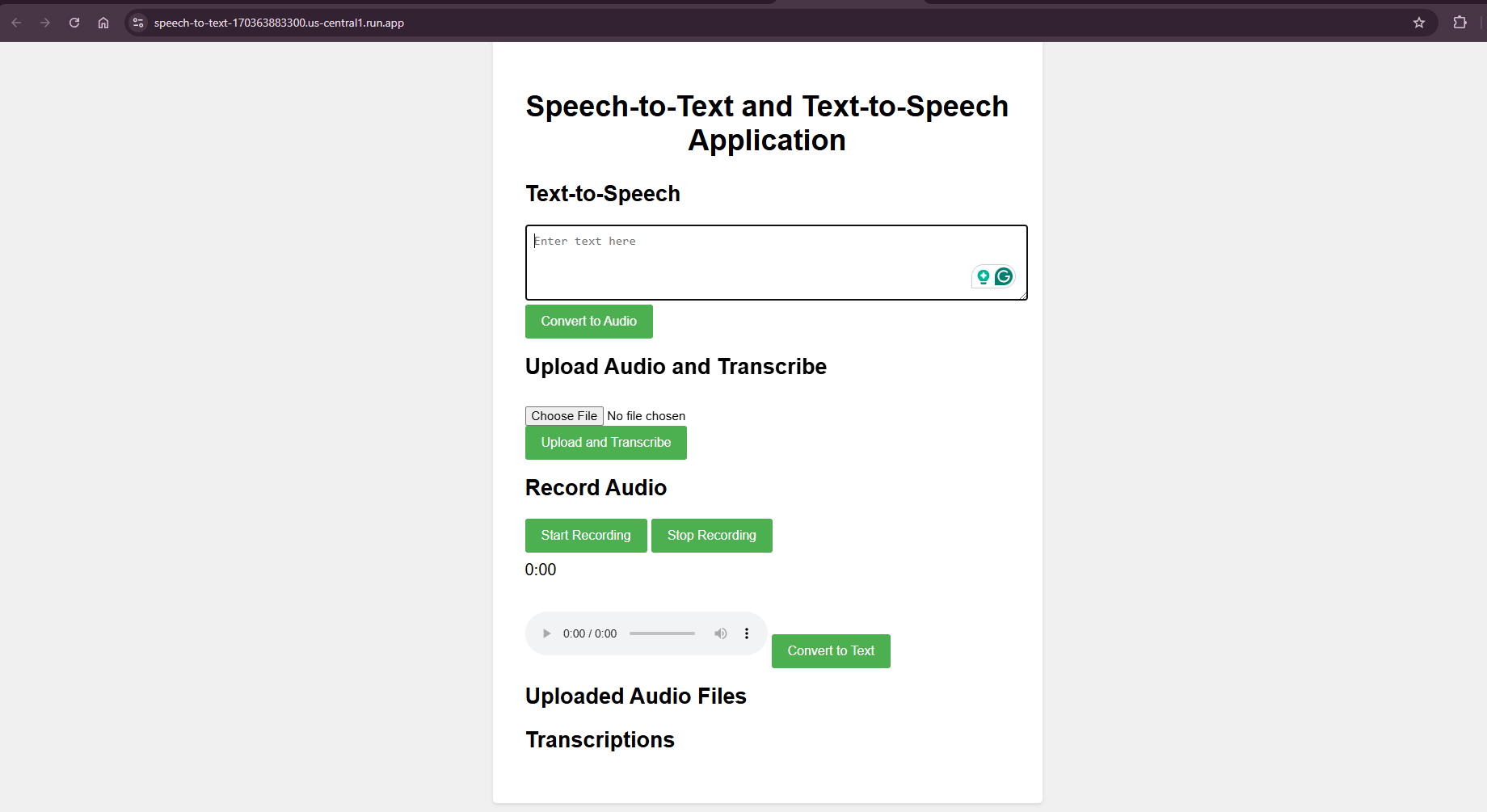
    .catch(error => {

        console.error('Error during transcription:', error);

        alert('There was an issue with transcription. Please try again.');

    });

});

*APP ScreenShot  
*