COT5390 Project 2 Chris Logan

COT 5930 Project 2

Student: Chris Logan
Email: loganc2023@fau.edu

GCP Project Name: cot5390project1
GCP Project ID: https://console.cloud.google.com/welcome/new?project=cot5390project1
Github Repo: https://github.com/christopherjlogan/cot5390_project1
App URL: https://cot5390project1.uc.r.appspot.com/
* Even those these links say “project1”, it contains the functionality for Project 2.

Assignment Instructions

Build upon your knowledge from project I and make the following adjustments: - Leverage the Language API in Google Cloud to evaluate the sentiment of the audio or text uploaded - Display whether the text has a positive/neutral/negative connotation

Provide a report of your application, architecture, code and design decisions, with a focus on what you learned.

Introduction

This project is a proof of concept for the uploading, recording and automated bi-directional conversion of speech and text from a web application. On project 2, additional functionality includes sentiment analysis.

Architecture

Project Planning

Project 1

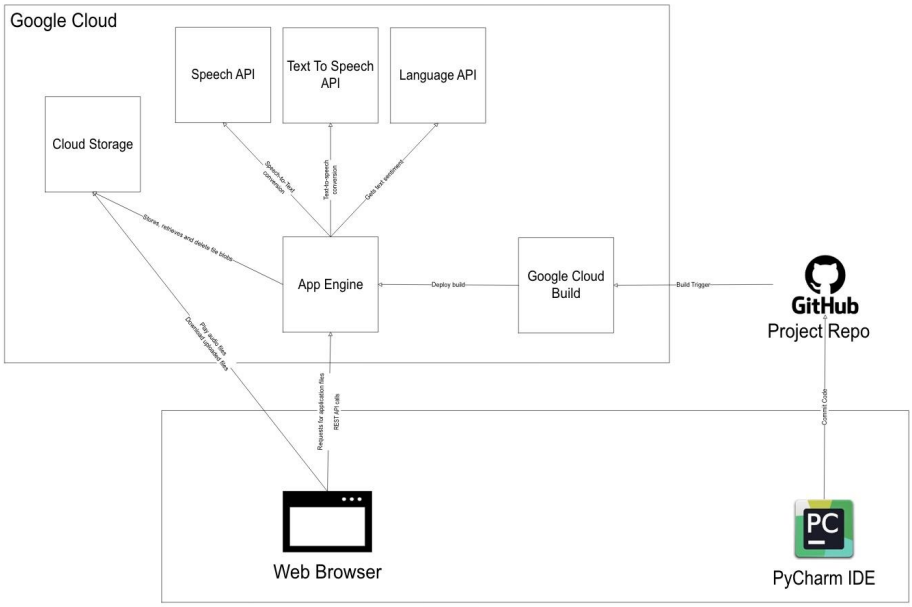
In the implementation of this project, the following steps were followed. During each of the steps, iterative coding and test took place. - Researched how to build web apps in Python - Created and ran basic Flask app on local dev machine - Added

file upload capability storing files on local dev machine - Added capability to display already uploaded files - Added speech recording capability - Researched Google Text-to-Speech API and how to integrate - Setup Google Cloud Project and IAM permissions - Added text-to-speech capability - Researched Google Speech API for speech-to-text - Added speech-to-text capability - Added language selection for conversion operations - Moved code to Github repo - Refactored code to use Google Cloud Storage for uploaded files - Setup Google App Engine and Admin - Setup Google Cloud Build with Trigger on Github Repo branch push - Tested and troubleshooted application running on Google App Engine

Project 2

In the implementation of Project 2, the following steps were followed. I followed the following steps: - Evaluated the shortcomings of project 1 architecture - Researched how to build Single Page Architecture (SPA) app using Python and JavaScript - Created and referenced JavaScript and Cascading Style Sheet (CSS) - Refactored HTML page and python app for SPA - Researched Language API usage for sentiment detection - Enabled Language API - Implemented sentiment detection API call - Added delete file API call - Enabled file deletion from web page

Solution Components



COT5390 Project2 Architecture.jpg

Implementation Details

Python Web Application

The application's user interface and back-end business logic is implemented in Python within a Flask app.
Dependencies: - Flask - for defining app endpoints and template generation - gunicorn and werkzeug - for running the Flask app - Google Cloud APIs - these APIs are discussed later in this section

Files:

```
- credentials
 - service-account.json (secret not stored in source code repo)
- static
 - img
  - negative.png
  - neutral.png
  - positive.png
  - sentiment-analysis.png
  - trash.png
 - app.js
 - styles.css
- templates
  - index.html
- README.md
- app.py
- app.yaml
- cloudbuild.yaml
- requirements.txt
```

Google App Engine

Runs the Python web application. Configured to give the service account access to deploy applications. ### Google Cloud Storage API Storing speech audio files. Google Cloud Storage is needed because Google App Engine cannot store persistently store files. Converts text into speech audio. Converts text into speech audio. Configured to give access to the Google Cloud project service account. Configured to give public access to the stored files since users are not authenticated. ### Google Speech API Converts speech audio files into text. Converts text into speech audio. Configured to give access to the Google Cloud project service account. ### Google Text-To-Speech API Converts text into speech audio. Configured to give access to the Google Cloud project service account. ### Google Language API Detects sentiment of provided text. Configured to give access to the Google Cloud project service account. ### Google Cloud Build Automatically builds and deploys the application to Google App Engine. Configured to trigger off a GitHut repo push. ### GitHub Stores source code. Configured as a public repo for sharing for grading.

Pros and Cons

Discuss what are the problems of this solution, assuming it needs to handle multiple users and scale as discussed in class. Discuss what are the advantages of this solution as implemented in this project. ### Pros 1. Using Single Page Architecture with REST APIs created a separation of concerns making UI and API more flexible. 2. Using Google Cloud Build with push triggers allows for continuous deployment of the code. 3. Using Google Cloud Storage makes the application ephemeral and therefore more fault tolerant. 4. Using Google App Engine, the application can be scaled since it is stateless.

Cons

1. The current architecture only supports a single user because all of the uploaded files are stored in a single cloud storage bucket without user segmentation.
2. The user interface is very basic and would not work well with many features.
3. No tests are implemented so testing the application required deploying and troubleshooting.
4. More effective error handling should be implemented
5. Text sentiment is not stored so it is not persistent after web page reload
6. It is confusing that the cloud project, repo and other resources are named as "project 1"

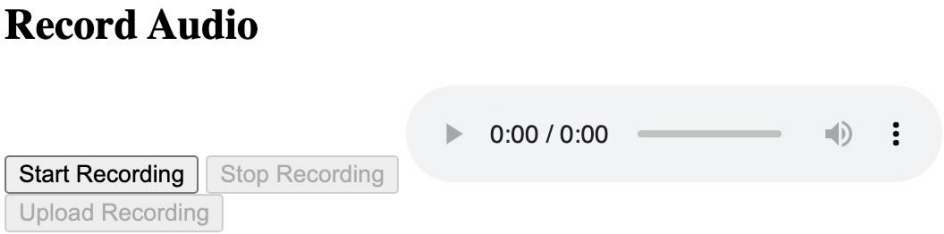
Problems Encountered and Solutions

1. Refactoring to SPA took significant refactoring of both front and back-ends of the application.
2. It took several iterations to understand how to use the Language API for sentiment detection

Application Instructions

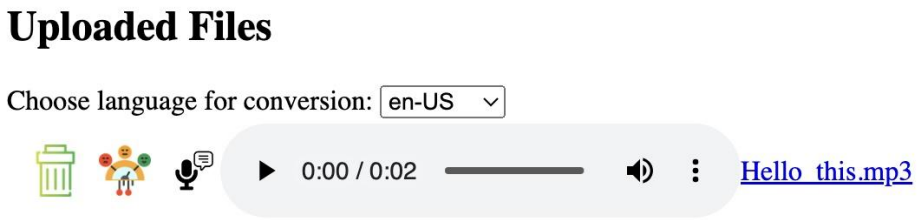
1. Uploading Speech Audio Files
    - To upload a file, click the "Choose File" button and select the audio file. Once selected, click the Upload button.

2. Recording Speech Audio
    - To record speech, click the "Start Recording" button
    - Once done speaking, click the "Stop Recording" button
    - Click the "Upload Recording" button to upload your recorded audio



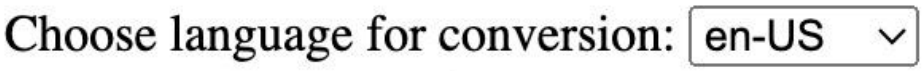
3. Playing Uploaded Speech Audio
    - Uploaded audio files are listed under the Uploaded Files section

- To play previously uploaded audio files, click the play button audio player control



4. Converting Text to Speech Audio File
   - To convert text to speech, type your message into the textbox and choose the language and gender for conversion
   - Click the “Convert to Speech” button
   - The converted text is stored as a file which can be downloaded in the Uploaded Files

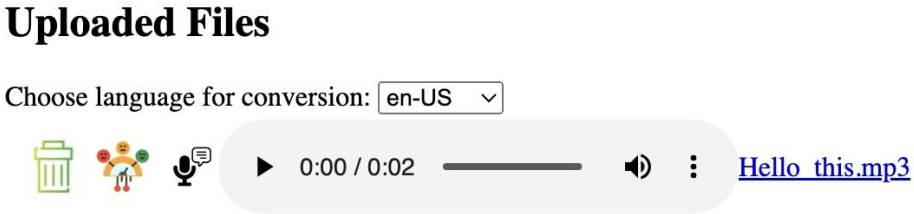
5. Converting Speech Audio File to Text
   - Select the target language:



   - Next to the desired audio file, click the  button
   - The converted text is stored as a file which can be downloaded in the Uploaded Files
6. Detecting Sentiment
   - Click the  icon next to the desired file
   - Based upon the detected sentiment, the icon will change according:

-  = POSITIVE
-  = NEGATIVE
-  = NEUTRAL

7. Download Audio File
   - To download an uploaded audio file, click the link on the name of the file to download it.



8. Download Text File
   - To download a text file, click the link on the name of the file to download it.

9. Delete Uploaded File
   - To delete a file, click the  icon next to the file
   - The list of files will refresh

Lessons Learned

1. An indicator is needed on the front-end to indicate that the REST API call is processing
2. How to use the JavaScript console in Chrome to troubleshoot API call issues

Code

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
        scale=1.0">
    <title>COT 5930 Project 2 - Chris Logan</title>
    <link rel="stylesheet" href="../static/styles.css">
    <meta http-equiv="Cache-Control" content="no-store, no-cache,
        must-revalidate">
    <meta http-equiv="Pragma" content="no-cache">
    <meta http-equiv="Expires" content="0">
</head>
<body>
 <h1>COT5390 Project 2 - Chris Logan</h1>
    <h2>Bi-Directional Text/Speech Conversion withe Sentiment
        Analysis</h2>
     <div id="overlay">
        <div id="spinner"></div>
    </div>
    <!-- Audio file upload form -->
    <h2>Upload An Audio File</h2>
    <input type="file" id="audioFileInput"
        accept=".mp3, .wav, .ogg, .m4a">
    <button id="uploadFileBtn">Upload</button>

    <hr>
    <h2>Record Audio</h2>
    <!-- Audio recorder interface -->
    <button id="startRecord">Start Recording</button>
    <button id="stopRecord" disabled>Stop Recording</button>
    <audio id="audioPlayback" controls></audio>
    <br>
    <button id="uploadRecord" disabled>Upload Recording</button>

    <hr>
    <h2>Convert Text to Speech</h2>
    <label for="language">Choose a language:</label>
    <select id="languageSelect"></select>

    <label for="gender">Choose a gender:</label>
    <select id="genderSelect" required>
        <option value="MALE">Male</option>
        <option value="FEMALE">Female</option>
    </select>
    <br>
    <textarea id="textToSpeechInput" rows="5" cols="40"
        placeholder="Enter text here" required></textarea>
```

```html
    <br>
    <button id="textToSpeechBtn">Convert to Speech</button>

    <hr>
    <h2>Uploaded Files</h2>
    <label for="language">Choose language for conversion:</label>
    <select id="languageSelectForSTT"></select>
    <div id="fileList">Loading files...</div>
    <script src="../static/app.js"></script>
</body>
</html>
```

app.js

```js
let uploadedFiles = []
let languages = []

const startRecordBtn = document.getElementById('startRecord');
const stopRecordBtn = document.getElementById('stopRecord');
const uploadRecordBtn = document.getElementById('uploadRecord');
const audioPlayback = document.getElementById('audioPlayback');
let mediaRecorder;
let audioChunks = [];
let audioBlob;

// Fetch list of uploaded files from API
async function loadUploadedFiles() {
    try {
        const response = await fetch('/api/files');
        if (!response.ok) {
            throw new Error('Failed to fetch files');
        }
        const data = JSON.parse(await response.text())
        uploadedFiles =
         data.message;  // Store the files into the local array

        displayFiles(uploadedFiles);
    } catch (error) {
        console.error('Error fetching the files:', error);
    }
}

function displayFiles(files) {
    const fileList = document.getElementById('fileList');
    fileList.innerHTML = '';  // Clear any existing list
    let textFiles = []
    let imageFiles = []
    let image_dir = '/static/img/'

    files.forEach(file => {
        if (file.endsWith('.txt')) {
            textFiles.push(file);
```

```
    } else {
        imageFiles.push(file);
    }
})

const table = document.createElement('table');
i=0

//Generating elements for image files
imageFiles.forEach(file => {
    const tablerow = document.createElement('tr');
    const tabledata = document.createElement('td');
    tabledata.style.display = 'flex';
    tabledata.style.alignItems = 'center';
    tabledata.style.marginBottom = '10px';

    // Create and configure the audio element
    const audioElement = document.createElement('audio');
    audioElement.controls = true;
    audioElement.src = file;

    // Extract the file name from the file URL
    const filename = file.substring(file.lastIndexOf('/') +
    1);

    const trashIcon = document.createElement('img');
    trashIcon.src = image_dir + 'trash.png';  // Replace with
    the actual path to your icon
    trashIcon.alt = 'Delete file';
    trashIcon.style.cursor = 'pointer';
    trashIcon.style.width = '30px';  // Adjust the size as
    needed
    trashIcon.style.marginLeft = '10px';
    trashIcon.onclick = () => {
        deleteFile(filename);
    };

    const sentimentIcon = document.createElement('img');
    sentimentIcon.src = image_dir + 'sentiment-
    analysis.png';  // Replace with the actual path to your
    icon
    sentimentIcon.alt = 'Analyze sentiment';
    sentimentIcon.style.cursor = 'pointer';
    sentimentIcon.style.width =
    '30px';  // Adjust the size as needed
    sentimentIcon.style.marginLeft = '10px';
    sentimentIcon.id = 'sentiment-icon-' + i;
    sentimentIcon.onclick = () => {
        analyzeSentiment(filename, sentimentIcon.id);
    };

    // Create the image that calls the conversion API
    const convertIcon = document.createElement('img');
```

```
        convertIcon.src = image_dir + 'speech-to-text.png';  //
        Replace with the actual path to your icon
        convertIcon.alt = 'Convert to text';
        convertIcon.style.cursor = 'pointer';
        convertIcon.style.width = '30px';  // Adjust the size as
        needed
        convertIcon.style.marginLeft = '10px';
        convertIcon.onclick = () => {
            convertAudioToText(filename);
        };

        const anchor = document.createElement('a');
        anchor.href = file;
        anchor.text = filename
        tabledata.append(trashIcon, sentimentIcon, convertIcon,
        audioElement, anchor);
        tablerow.appendChild(tabledata);
        table.appendChild(tablerow)
        i++
    });

    //Generating elements for text files
    textFiles.forEach(file => {
        const tablerow = document.createElement('tr');
        const tabledata = document.createElement('td');
        tabledata.style.display = 'flex';
        tabledata.style.alignItems = 'center';
        tabledata.style.marginBottom = '10px';

        const filename = file.substring(file.lastIndexOf('/') +
        1);

        const trashIcon = document.createElement('img');
        trashIcon.src = image_dir + 'trash.png';  // Replace with
        the actual path to your icon
        trashIcon.alt = 'Delete file';
        trashIcon.style.cursor = 'pointer';
        trashIcon.style.width = '30px';  // Adjust the size as
        needed
        trashIcon.style.marginLeft = '10px';
        trashIcon.onclick = () => {
            deleteFile(filename);
        };

        // Create the image for sentiment analysis
        const sentimentIcon = document.createElement('img');
        sentimentIcon.src = image_dir + 'sentiment-
        analysis.png';  // Replace with the actual path to your
        icon
        sentimentIcon.alt = 'Analyze sentiment';
        sentimentIcon.style.cursor = 'pointer';
        sentimentIcon.style.width =
        '30px';  // Adjust the size as needed
        sentimentIcon.style.marginLeft = '10px';
```

```
        sentimentIcon.id = 'sentiment-icon-' + i;
        sentimentIcon.onclick = () => {
            analyzeSentiment(filename, sentimentIcon.id);
        };

        const textLink = document.createElement('a');
        textLink.href = file;
        textLink.text = file.substring(file.lastIndexOf('/') + 1);
        textLink.target = "_new"

        tabledata.append(trashIcon, sentimentIcon, textLink);
        tablerow.appendChild(tabledata);
        table.appendChild(tablerow)
        i++
    })
    fileList.appendChild(table)
}

// Populate language options from API
async function loadLanguages() {
    try {
        const response = await fetch('/api/languages');
        if (!response.ok) {
            throw new Error('Failed to fetch files');
        }
        const data = JSON.parse(await response.text())
        languages = data.message;  // Store the files into the
        local array
        await populateLanguageSelect(languages);
    } catch (error) {
        console.error('Error fetching the files:', error);
    }
}

// Populate language dropdown dynamically for supported languages
function populateLanguageSelect(languages) {
    const languageSelect =
        document.getElementById('languageSelect');
    languages.forEach(language => {
        const option = document.createElement('option');
        option.value = language;
        option.textContent = language;
        languageSelect.appendChild(option)
    });

    const languageSelectForSTT =
        document.getElementById('languageSelectForSTT');
    languages.forEach(language => {
        const option = document.createElement('option');
        option.value = language;
        option.textContent = language;
        languageSelectForSTT.appendChild(option)
    });
```

```
}

// Start audio recording
startRecordBtn.addEventListener('click', async () => {
    const stream = await navigator.mediaDevices.getUserMedia({
        audio: true });
    mediaRecorder = new MediaRecorder(stream);
    mediaRecorder.start();

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

    mediaRecorder.onstop = () => {
        audioBlob = new Blob(audioChunks, { type: 'audio/wav' });
        audioChunks = [];
        const audioURL = URL.createObjectURL(audioBlob);
        audioPlayback.src = audioURL;
        uploadRecordBtn.disabled = false;
    };

    startRecordBtn.disabled = true;
    stopRecordBtn.disabled = false;
});

// Stop audio recording
stopRecordBtn.addEventListener('click', () => {
    mediaRecorder.stop();
    startRecordBtn.disabled = false;
    stopRecordBtn.disabled = true;
});

// Upload recorded audio
uploadRecordBtn.addEventListener('click', async () => {
    showLoadingOverlay()
    const formData = new FormData();
    const timestamp = new Date().toISOString().replace(/[-:.]/g,
        '');  // Generate a timestamp
    formData.append('file', audioBlob, `recording_$
        {timestamp}.wav`);

    const response = await fetch('/api/upload', {
        method: 'POST',
        body: formData
    });

    if (response.ok) {
        await loadUploadedFiles(); // Reload file list
    } else {
        alert('Request failed');
    }
    hideLoadingOverlay()
});
```

```
// Upload audio file
document.getElementById('uploadFileBtn').addEventListener('click',
        async () => {
    showLoadingOverlay()
    const fileInput = document.getElementById('audioFileInput');
    const file = fileInput.files[0];

    if (!file) {
        alert('Please select a file to upload');
        return;
    }

    const formData = new FormData();
    formData.append('file', file);

    const response = await fetch('/api/upload', {
        method: 'POST',
        body: formData
    });

    if (response.ok) {
        document.getElementById('audioFileInput').value = ''
        await loadUploadedFiles(); // Reload file list
    } else {
        alert('Request failed');
    }
    hideLoadingOverlay()
});

// Convert text to speech
document.getElementById('textToSpeechBtn').addEventListener('click',
        async () => {
    showLoadingOverlay()
    const text =
        document.getElementById('textToSpeechInput').value;
    const language =
        document.getElementById('languageSelect').value;
    const gender = document.getElementById('genderSelect').value;

    const response = await fetch('/api/text-to-speech', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ text, language, gender })
    });

    if (response.ok) {
        document.getElementById('textToSpeechInput').value = ''
        await loadUploadedFiles(); // Reload file list
    } else {
        alert('Request failed');
```

```
    }
    hideLoadingOverlay()
});

// Function to call the API for converting audio to text
async function convertAudioToText(filename) {
    showLoadingOverlay()
    let language =
        document.getElementById('languageSelectForSTT').value
    const response = await fetch('/api/speech-to-text', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ filename, language})
    });
    const result = await response.json();

    if (response.ok) {
        await loadUploadedFiles(); // Reload file list
    } else {
        alert('Request failed');
    }
    hideLoadingOverlay()
}

async function analyzeSentiment(filename, icon_id) {
    showLoadingOverlay()
    let language =
        document.getElementById('languageSelectForSTT').value
    const response = await fetch('/api/analyze-sentiment', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ filename})
    });
    const data = JSON.parse(await response.text())

    if (response.ok) {
        sentiment = data.sentiment
        //alert('Analyzed Sentiment: ' + sentiment)
        document.getElementById(icon_id).src='/static/img/' +
         sentiment + '.png'
    } else {
        alert('Request failed');
    }
    hideLoadingOverlay()
}

async function deleteFile(filename) {
    showLoadingOverlay()
    const response = await fetch('/api/delete-file', {
```

```
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ filename})
    });
    hideLoadingOverlay()
    loadUploadedFiles()
}

// Show the overlay
function showLoadingOverlay() {
    document.getElementById('overlay').style.display = 'flex';
}

// Hide the overlay
function hideLoadingOverlay() {
    document.getElementById('overlay').style.display = 'none';
}

// Initialize page
document.addEventListener('DOMContentLoaded', () => {
    loadUploadedFiles();
    loadLanguages();
});
```

styles.css

```
/* Full-screen overlay */
#overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0,
        0.5); /* Semi-transparent background */
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 9999; /* Ensure it appears above other content */
    display: none; /* Hidden by default */
}

/* Spinner graphic */
#spinner {
    border: 8px solid rgba(255, 255, 255, 0.3); /* Light border */
    border-top: 8px solid white; /* White top border */
    border-radius: 50%;
    width: 60px;
    height: 60px;
    animation: spin 1s linear infinite; /* Rotation animation */
```

```
}

/* Keyframes for the spin animation */
@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
```

app.py

```
import os
from flask import Flask, render_template, request, redirect,
        url_for, send_from_directory, session, jsonify, request
from google.cloud.language_v1 import LanguageServiceClient
from werkzeug.utils import secure_filename
from datetime import datetime
from typing import Sequence
from google.oauth2 import service_account
from google.cloud import storage, speech, texttospeech,
        language_v1

# Create a Flask app
app = Flask(__name__)
app.secret_key = 'COT5930'

SERVICE_ACCOUNT_FILE = "credentials/service-account.json"
# Check if the service account file exists
if os.path.exists(SERVICE_ACCOUNT_FILE):
    RUN_LOCALLY = True
    print("Service account file found, loading credentials...")
    credentials =
        service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE
    ttsclient =
        texttospeech.TextToSpeechClient(credentials=credentials)
    sttclient = speech.SpeechClient(credentials=credentials)
    gcsclient = storage.Client(credentials=credentials)
    langclient = LanguageServiceClient(credentials=credentials)
else:
    RUN_LOCALLY = False
    print("No service account file found, using Application
        Default Credentials (ADC)...")
    # Use Application Default Credentials (ADC)
    ttsclient = texttospeech.TextToSpeechClient()
    sttclient = speech.SpeechClient()
    gcsclient = storage.Client()
    langclient = LanguageServiceClient()

BUCKET_NAME = 'cot5390project1.appspot.com'
ALLOWED_EXTENSIONS = {'wav', 'mp3', 'ogg', 'm4a'}

# Function to check if file extension is allowed
def allowed_file(filename):
```

```
    return '.' in filename and filename.rsplit('.', 1)[1].lower()
        in ALLOWED_EXTENSIONS

def list_uploaded_files():
    bucket = gcsclient.get_bucket(BUCKET_NAME)  # Get the bucket
    blobs = bucket.list_blobs()
        # List files with the folder path prefix

    files = []
    for blob in blobs:
        # Exclude the folder itself from the list, only add files
        if not blob.name.endswith("/"):
            files.append(blob.public_url)
    return files

def upload_to_cloud_storage(file_content, filename):
    bucket = gcsclient.bucket(BUCKET_NAME)
    blob = bucket.blob(filename)
    blob.upload_from_string(file_content)
    return blob.public_url

def delete_from_cloud_storage(filename):
    bucket = gcsclient.bucket(BUCKET_NAME)
    blob = bucket.blob(filename)
    blob.delete()
    return True

def unique_languages_from_voices(voices:
        Sequence[texttospeech.Voice]):
    language_list = []
    for voice in voices:
        for language_code in voice.language_codes:
            if language_code not in language_list:  # Check for
        uniqueness
                language_list.append(language_code)
    return language_list

def list_languages():
    response = ttsclient.list_voices()
    languages = unique_languages_from_voices(response.voices)
    return languages

@app.route('/', methods=['GET'])
def home():
    return render_template('index.html')

# REST methods below
@app.route('/api/files', methods=['GET'])
def get_uploaded_files():
    files = list_uploaded_files()
    return jsonify({'message': files})

@app.route('/api/languages', methods=['GET'])
```

```python
def get_languages():
    return jsonify({'message': list_languages()})

@app.route('/api/upload', methods=['POST'])
def upload_audio():
    if 'file' not in request.files:
        return jsonify({'error': 'No file uploaded'}), 400

    file = request.files['file']
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        upload_to_cloud_storage(file.read(), filename)
        return jsonify({'message': 'File uploaded successfully',
         'filename': filename})
    return jsonify({'error': 'File not allowed'}), 400

@app.route('/api/text-to-speech', methods=['POST'])
def text_to_speech():
    data = request.get_json()
    text = data.get('text')
    language = data.get('language')
    gender = data.get('gender')
    response = generate_speech(text, language, gender)
    # Save the audio file
    timestamp = datetime.now().strftime("%Y%m%dT%H%M%S")
    filename = f"tts_{timestamp}_{language}_{gender}.mp3"
    upload_to_cloud_storage(response.audio_content, filename)
    return jsonify({'message': 'Text converted to speech
        successfully'})

def generate_speech(text_input, selected_language,
        selected_gender):
    synthesis_input = texttospeech.SynthesisInput(text=text_input)

    # Set the voice parameters, using the selected language
    voice = texttospeech.VoiceSelectionParams(
        language_code=selected_language,
        ssml_gender=selected_gender
    )

    # Select the audio format
    audio_config = texttospeech.AudioConfig(
        audio_encoding=texttospeech.AudioEncoding.MP3
    )

    # Perform the text-to-speech request
    response = ttsclient.synthesize_speech(
        input=synthesis_input, voice=voice,
        audio_config=audio_config
    )
    return response

@app.route('/api/speech-to-text', methods=['POST'])
```

```python
def speech_to_text():
    data = request.get_json()
    filename = data.get('filename')
    language = data.get('language')
    converted_text = convert_to_text(filename, language)
    timestamp = datetime.now().strftime("%Y%m%dT%H%M%S")
    filename = f"stt_{timestamp}_{language}.txt"
    upload_to_cloud_storage(converted_text, filename)
    return jsonify({'message': 'Speech converted to text
        successfully'})

@app.route('/api/delete-file', methods=['POST'])
def delete_file():
    data = request.get_json()
    filename = data.get('filename')
    delete_from_cloud_storage(filename)
    return jsonify({'message': 'File successfully deleted'})

@app.route('/api/analyze-sentiment', methods=['POST'])
def analyze_sentiment_from_file():
    data = request.get_json()
    filename = data.get('filename')
    # Get the file from the bucket
    text_to_analyze = ''
    if filename.endswith(".txt"):
        text_to_analyze = download_blob_as_text(BUCKET_NAME,
        filename)
    else:
        # If the file is audio, convert to text first
        text_to_analyze = convert_to_text(filename, "en")
    # Run through sentiment analysis
    document = language_v1.Document(
        content=text_to_analyze,
        type_=language_v1.Document.Type.PLAIN_TEXT,
        language='en'
    )
    sentiment =
        langclient.analyze_sentiment(document=document).document_sentiment.score
    text_sentiment = evaluate_sentiment_score(sentiment)
    print("Analyzing sentiment for", filename, "as", sentiment,
        "-",text_sentiment)
    return jsonify({'text': text_to_analyze,'sentiment':
        text_sentiment})

def evaluate_sentiment_score(score):
    if score > 0:
        return "positive"
    elif score < 0:
        return "negative"
    else:
        return "neutral"

def convert_to_text(filename, language):
```

```python
    audio_content = download_blob_as_bytes(BUCKET_NAME, filename)
    audio = speech.RecognitionAudio(content=audio_content)
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.MP3,  # Adjust based on your file type (MP3 assumed here)
        sample_rate_hertz=16000,
        language_code=language
    )
    response = sttclient.recognize(config=config, audio=audio)
    transcript = ""
    for result in response.results:
        transcript += result.alternatives[0].transcript
    return transcript

def download_blob_as_bytes(bucket_name, blob_name):
    print("Downloading blob as bytes", blob_name, "from bucket", bucket_name)
    bucket = gcsclient.get_bucket(bucket_name)
    blob = bucket.blob(blob_name)
    bytes = blob.download_as_bytes()
    return bytes

def download_blob_as_text(bucket_name, blob_name):
    print("Downloading blob as text", blob_name, "from bucket", bucket_name)
    bucket = gcsclient.get_bucket(bucket_name)
    blob = bucket.blob(blob_name)
    bytes = blob.download_as_text()
    return bytes

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

app.yaml

```yaml
runtime: python39  # Specify the Python runtime

entrypoint: gunicorn -b :$PORT app:app  # Use Gunicorn to run your Flask app

# Ensure the static directory is correctly mapped
handlers:
  - url: /static
    static_dir: static

  - url: /.*
    script: auto

# Optional: Environment variables
env_variables:
  FLASK_ENV: 'production'
```

cloudbuild.yaml

```yaml
steps:
  # Install dependencies
  - name: 'python:3.9-slim'
    id: 'Install dependencies'
    entrypoint: 'bash'
    args:
      - '-c'
      - |
        pip install --upgrade pip
        pip install -r requirements.txt

  # Deploy to Google App Engine
  - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
    id: 'Deploy to App Engine'
    entrypoint: 'bash'
    args:
      - '-c'
      - |
        gcloud app deploy app.yaml --quiet

options:
  logging: CLOUD_LOGGING_ONLY
```

requirements.txt

```
Flask==3.0.3
gunicorn
google-cloud-texttospeech
google-cloud-speech
google-cloud-storage
google-cloud-language
werkzeug~=3.0.4
protobuf~=5.28.2
```