Voice-to-text

<!DOCTYPE html>

<html>

<head>

    <title>Real-time Voice Transcription</title>

</head>

<body>

    <button id="start-btn">Start Recording</button>

    <div id="transcript"></div>

    <script>

        const startBtn = document.getElementById('start-btn');

        const transcriptDiv = document.getElementById('transcript');

        startBtn.addEventListener('click', async () => {

            try {

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

                const recognition = new SpeechRecognition();

                recognition.continuous = true;

                recognition.interimResults = true;

                recognition.start(stream);

                recognition.onresult = (event) => {

                    const transcript = event.isFinal ? event.results[0][0].transcript : '';

                    transcriptDiv.textContent = transcript;

                };

            } catch (error) {

                console.error(error);

                transcriptDiv.textContent = 'Error accessing microphone';

            }

        });

    </script>

</body>

</html>

For app4

Voice -recorder

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Voice Recorder</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f4f4f4;

            margin: 0;

            padding: 0;

            display: flex;

            flex-direction: column;

            justify-content: center;

            align-items: center;

            height: 100vh;

        }

        .container {

            background-color: #fff;

            padding: 20px;

            border-radius: 10px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

            text-align: center;

            width: 90%;

            max-width: 800px;

        }

        button {

            background-color: #007bff;

            color: #fff;

            padding: 10px 20px;

            margin: 10px;

            border: none;

            border-radius: 5px;

            cursor: pointer;

        }

        button:hover {

            background-color: #0056b3;

        }

        #transcript {

            margin-top: 20px;

            padding: 10px;

            background-color: #f4f4f4;

            border: 1px solid #ddd;

            border-radius: 5px;

            white-space: pre-wrap; /\* Preserve line breaks \*/

            color: #333;

            min-height: 100px; /\* Ensure minimum height for the transcript box \*/

            text-align: left; /\* Align text to the left for better readability \*/

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Voice Recorder</h1>

        <button onclick="startRecording()">Start Recording</button>

        <button onclick="pauseRecording()">Pause Recording</button>

        <button onclick="stopRecording()">Stop Recording</button>

        <div id="transcript">Transcript will appear here...</div>

    </div>

    <script>

        let mediaRecorder;

        let audioChunks = [];

        let isRecording = false;

        function startRecording() {

            if (isRecording) return; // Prevent multiple start clicks

            navigator.mediaDevices.getUserMedia({ audio: true })

                .then(stream => {

                    mediaRecorder = new MediaRecorder(stream);

                    mediaRecorder.start();

                    isRecording = true;

                    mediaRecorder.ondataavailable = event => {

                        audioChunks.push(event.data);

                    };

                    mediaRecorder.onstop = () => {

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

                        const audioUrl = URL.createObjectURL(audioBlob);

                        const audio = new Audio(audioUrl);

                        audio.play();

                        const formData = new FormData();

                        formData.append('audio', audioBlob, 'recording.wav');

                        fetch('/transcribe', {

                            method: 'POST',

                            body: formData

                        })

                        .then(response => {

                            if (!response.ok) {

                                throw new Error(`Server error: ${response.statusText}`);

                            }

                            return response.json();

                        })

                        .then(data => {

                            if (data.transcription) {

                                document.getElementById('transcript').innerText = data.transcription;

                            } else if (data.error) {

                                document.getElementById('transcript').innerText = 'Error: ' + data.error;

                            } else {

                                document.getElementById('transcript').innerText = 'Could not transcribe audio.';

                            }

                        })

                        .catch(error => {

                            document.getElementById('transcript').innerText = 'Error: ' + error.message;

                        });

                    };

                })

                .catch(error => {

                    document.getElementById('transcript').innerText = 'Error accessing microphone: ' + error.message;

                });

        }

        function pauseRecording() {

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

                mediaRecorder.pause();

                document.getElementById('transcript').innerText = 'Recording paused...';

            }

        }

        function stopRecording() {

            if (mediaRecorder && isRecording) {

                mediaRecorder.stop();

                isRecording = false;

                document.getElementById('transcript').innerText = 'Recording stopped. Processing...';

            }

        }

    </script>

</body>

</html>

Отлично работает

Апп.пу

import os

import base64

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

from flask\_sqlalchemy import SQLAlchemy

import logging

from forms import ClientForm

from models import db, Client

import requests

import openai

from translator import transcribe\_audio, translate\_text

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

app.config.from\_object('config.Config')

db.init\_app(app)

# Set up logging

logging.basicConfig(level=logging.DEBUG)

@app.route('/')

def home():

    return render\_template('home.html')

@app.route('/voice-recorder')

def voice\_recorder():

    return render\_template('voice\_to\_text.html')

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

def transcribe\_audio\_route():

    if 'audio' not in request.files:

        return jsonify({"error": "No audio file"}), 400

    audio\_file = request.files['audio']

    audio\_base64 = base64.b64encode(audio\_file.read()).decode('utf-8')

    transcription = transcribe\_audio(audio\_base64)

    return jsonify({"transcription": transcription})

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

def translate\_text\_route():

    data = request.json

    text = data.get('text')

    target\_language = data.get('target\_language')

    translation = translate\_text(text, target\_language)

    if translation is None:

        return jsonify({"error": "Translation failed"}), 500

    return jsonify({"translation": translation})

@app.route('/visa')

def visa():

    return render\_template('visa.html')

@app.route('/visa/immigrant')

def immigrant\_visas():

    return render\_template('immigrant\_visas.html')

@app.route('/visa/non-immigrant')

def non\_immigrant\_visas():

    return render\_template('non\_immigrant\_visas.html')

@app.route('/client-info', methods=['GET', 'POST'])

def client\_info():

    form = ClientForm()

    if form.validate\_on\_submit():

        client = Client(

            full\_name=form.full\_name.data,

            alien\_registration\_number=form.alien\_registration\_number.data,

            uscis\_online\_account\_number=form.uscis\_online\_account\_number.data,

            date\_of\_birth=form.date\_of\_birth.data.strftime('%m/%d/%Y'),

            country\_of\_birth=form.country\_of\_birth.data,

            country\_of\_citizenship=form.country\_of\_citizenship.data,

            ssn=form.ssn.data,

            gender=form.gender.data,

            marital\_status=form.marital\_status.data,

            spouse\_full\_name=form.spouse\_full\_name.data,

            marriage\_date=form.marriage\_date.data,

            number\_of\_children=form.number\_of\_children.data,

            children\_names\_dobs=form.children\_names\_dobs.data,

            physical\_address=form.physical\_address.data,

            mailing\_address=form.mailing\_address.data,

            telephone\_number=form.telephone\_number.data,

            email\_address=form.email\_address.data,

            residential\_history=form.residential\_history.data,

            employment\_status=form.employment\_status.data,

            current\_employer\_name\_address=form.current\_employer\_name\_address.data,

            job\_title=form.job\_title.data,

            start\_date\_current\_job=form.start\_date\_current\_job.data,

            employment\_history=form.employment\_history.data,

            previous\_us\_visas=form.previous\_us\_visas.data,

            previous\_us\_entries=form.previous\_us\_entries.data,

            deportation\_history=form.deportation\_history.data,

            criminal\_history=form.criminal\_history.data,

            political\_social\_orgs=form.political\_social\_orgs.data,

            languages\_spoken=form.languages\_spoken.data,

            emergency\_contact=form.emergency\_contact.data,

            other\_citizenship=form.other\_citizenship.data,

            travel\_purpose=form.travel\_purpose.data,

            countries\_to\_visit=form.countries\_to\_visit.data,

            length\_of\_trip=form.length\_of\_trip.data,

            intended\_departure\_date=form.intended\_departure\_date.data,

            travel\_document\_type=form.travel\_document\_type.data,

            summary=form.summary.data,

        )

        db.session.add(client)

        db.session.commit()

        pdf\_file = create\_pdf(client)

        flash('Client information has been saved!', 'success')

        return send\_from\_directory(directory='client\_applications', path=pdf\_file, as\_attachment=True)

    return render\_template('client.html', form=form)

@app.route('/applications')

def applications():

    pdf\_path = 'C:/Users/Elena/Documents/GitHub/Imigration\_project\_new\_res/Addition\_information'

    pdf\_files = [f for f in os.listdir(pdf\_path) if f.endswith('.pdf')]

    return render\_template('applications.html', pdf\_files=pdf\_files, pdf\_path=pdf\_path)

@app.route('/view-pdf/<filename>')

def view\_pdf(filename):

    return send\_from\_directory('C:/Users/Elena/Documents/GitHub/Imigration\_project\_new\_res/Addition\_information', filename)

@app.route('/edit-pdf/<filename>')

def edit\_pdf(filename):

    return render\_template('edit\_pdf.html', filename=filename)

@app.route('/save-pdf', methods=['POST'])

def save\_pdf():

    data = request.json

    file\_name = data['fileName']

    pdf\_data = data['pdfData']

    save\_path = os.path.join('C:/Users/Elena/Documents/GitHub/Imigration\_project\_new\_res/pdf\_Application', file\_name)

    with open(save\_path, 'wb') as f:

        f.write(bytes.fromhex(pdf\_data))

    return jsonify({"message": "PDF saved successfully!"})

@app.route('/online-helper', methods=['GET', 'POST'])

def online\_helper():

    if request.method == 'POST':

        user\_info = request.form['user\_info']

        response = requests.post('http://localhost:5000/get-advice', json={'user\_info': user\_info})

        if response.status\_code == 200:

            try:

                response\_data = response.json()

                advice = response\_data.get('advice', 'No advice available')

            except requests.exceptions.JSONDecodeError as e:

                advice = f"Error decoding response: {str(e)}"

        else:

            advice = f"Error: Received status code {response.status\_code}"

        return render\_template('online\_helper\_result.html', advice=advice)

    return render\_template('online\_helper.html')

def create\_pdf(client):

    from reportlab.lib.pagesizes import letter

    from reportlab.pdfgen import canvas

    pdf\_file = os.path.join('client\_applications', f'{client.full\_name.replace(" ", "\_")}.pdf')

    c = canvas.Canvas(pdf\_file, pagesize=letter)

    c.drawString(100, 750, f"Full Name: {client.full\_name}")

    c.drawString(100, 735, f"Alien Registration Number: {client.alien\_registration\_number}")

    c.drawString(100, 720, f"USCIS Online Account Number: {client.uscis\_online\_account\_number}")

    c.drawString(100, 705, f"Date of Birth: {client.date\_of\_birth}")

    c.drawString(100, 690, f"Country of Birth: {client.country\_of\_birth}")

    c.drawString(100, 675, f"Country of Citizenship: {client.country\_of\_citizenship}")

    c.drawString(100, 660, f"U.S. Social Security Number: {client.ssn}")

    c.drawString(100, 645, f"Gender: {client.gender}")

    c.drawString(100, 630, f"Marital Status: {client.marital\_status}")

    c.drawString(100, 615, f"Number of Children: {client.number\_of\_children}")

    children\_list = eval(client.children\_names\_dobs) if client.children\_names\_dobs else []

    for i, child in enumerate(children\_list):

        c.drawString(100, 600 - i \* 15, f"Child {i+1} - Name: {child['first\_name']} {child['last\_name']}, Date of Birth: {child['date\_of\_birth']}")

    c.drawString(100, 540, f"Current Physical Address: {client.physical\_address}")

    c.drawString(100, 525, f"Mailing Address: {client.mailing\_address}")

    c.drawString(100, 510, f"Telephone Number: {client.telephone\_number}")

    c.drawString(100, 495, f"Email Address: {client.email\_address}")

    c.drawString(100, 480, f"Residential History: {client.residential\_history}")

    c.drawString(100, 465, f"Employment Status: {client.employment\_status}")

    c.drawString(100, 450, f"Current Employer Name and Address: {client.current\_employer\_name\_address}")

    c.drawString(100, 435, f"Job Title: {client.job\_title}")

    c.drawString(100, 420, f"Start Date of Current Job: {client.start\_date\_current\_job}")

    c.drawString(100, 405, f"Employment History: {client.employment\_history}")

    c.drawString(100, 390, f"Previous U.S. Visas: {client.previous\_us\_visas}")

    c.drawString(100, 375, f"Previous U.S. Entries: {client.previous\_us\_entries}")

    c.drawString(100, 360, f"Deportation History: {client.deportation\_history}")

    c.drawString(100, 345, f"Criminal History: {client.criminal\_history}")

    c.drawString(100, 330, f"Political or Social Organizations: {client.political\_social\_orgs}")

    c.drawString(100, 315, f"Languages Spoken: {client.languages\_spoken}")

    c.drawString(100, 300, f"Emergency Contact: {client.emergency\_contact}")

    c.drawString(100, 285, f"Other Countries of Citizenship: {client.other\_citizenship}")

    c.drawString(100, 270, f"Purpose of Travel: {client.travel\_purpose}")

    c.drawString(100, 255, f"Countries to Visit: {client.countries\_to\_visit}")

    c.drawString(100, 240, f"Expected Length of Trip: {client.length\_of\_trip}")

    c.drawString(100, 225, f"Date of Intended Departure: {client.intended\_departure\_date}")

    c.drawString(100, 210, f"Travel Document Type: {client.travel\_document\_type}")

    c.drawString(100, 195, f"Summary: {client.summary}")

    c.save()

    return pdf\_file

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

    with app.app\_context():

        db.create\_all()  # Create database tables

    app.run(debug=True)

Войс ту текст.пу

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Voice to Text and Translation</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f4f4f4;

            margin: 0;

            padding: 20px;

        }

        .container {

            background-color: #fff;

            padding: 20px;

            border-radius: 10px;

            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

            max-width: 800px;

            margin: auto;

        }

        h1 {

            color: #333;

        }

        .block {

            background-color: #007bff;

            color: #fff;

            padding: 10px;

            margin: 10px 0;

            border-radius: 5px;

            cursor: pointer;

            text-decoration: none;

            display: inline-block;

        }

        .block:hover {

            background-color: #0056b3;

        }

        .window {

            width: 100%;

            height: 150px;

            padding: 10px;

            border: 1px solid #ddd;

            border-radius: 5px;

            background-color: #f9f9f9;

            margin-top: 10px;

            overflow-y: auto;

        }

        select {

            margin: 10px 0;

            padding: 10px;

            border-radius: 5px;

            border: 1px solid #ddd;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Voice to Text and Translation</h1>

        <label for="spoken-language">Select Spoken Language:</label>

        <select id="spoken-language">

            <option value="en-US">English</option>

            <option value="bg-BG">Bulgarian</option>

            <option value="uk-UA">Ukrainian</option>

            <option value="es-ES">Spanish</option>

            <option value="pl-PL">Polish</option>

        </select>

        <label for="target-language">Select Target Language:</label>

        <select id="target-language">

            <option value="en">English</option>

            <option value="bg">Bulgarian</option>

            <option value="uk">Ukrainian</option>

            <option value="es">Spanish</option>

            <option value="pl">Polish</option>

        </select>

        <button class="block" onclick="startRecording()">Start Recording</button>

        <button class="block" onclick="pauseRecording()">Pause</button>

        <button class="block" onclick="stopRecording()">Stop</button>

        <h2>Transcription:</h2>

        <div id="transcription" class="window"></div>

        <h2>Translation:</h2>

        <div id="translation" class="window"></div>

    </div>

    <script>

        let mediaRecorder;

        let audioChunks = [];

        let isRecording = false;

        let recognition;

        function startRecording() {

            if (!isRecording) {

                const spokenLanguage = document.getElementById("spoken-language").value;

                navigator.mediaDevices.getUserMedia({ audio: true }).then(stream => {

                    mediaRecorder = new MediaRecorder(stream);

                    mediaRecorder.start();

                    isRecording = true;

                    mediaRecorder.addEventListener("dataavailable", event => {

                        audioChunks.push(event.data);

                    });

                    mediaRecorder.addEventListener("stop", () => {

                        const audioBlob = new Blob(audioChunks);

                        const formData = new FormData();

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

                        fetch("/transcribe", {

                            method: "POST",

                            body: formData

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

                          .then(data => {

                              document.getElementById("transcription").innerText = data.transcription;

                              // Automatically translate to the desired language

                              const targetLanguage = document.getElementById("target-language").value;

                              fetch("/translate", {

                                  method: "POST",

                                  headers: {

                                      "Content-Type": "application/json"

                                  },

                                  body: JSON.stringify({

                                      text: data.transcription,

                                      target\_language: targetLanguage

                                  })

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

                                .then(data => {

                                    document.getElementById("translation").innerText = data.translation;

                                }).catch(error => console.error("Error translating text:", error));

                          }).catch(error => console.error("Error transcribing audio:", error));

                    });

                });

                // Setup real-time speech recognition

                recognition = new (window.SpeechRecognition || window.webkitSpeechRecognition)();

                recognition.lang = spokenLanguage;

                recognition.interimResults = true;

                recognition.onresult = event => {

                    const transcript = Array.from(event.results)

                        .map(result => result[0])

                        .map(result => result.transcript)

                        .join('');

                    document.getElementById("transcription").innerText = transcript;

                    // Automatically translate to the desired language

                    const targetLanguage = document.getElementById("target-language").value;

                    fetch("/translate", {

                        method: "POST",

                        headers: {

                            "Content-Type": "application/json"

                        },

                        body: JSON.stringify({

                            text: transcript,

                            target\_language: targetLanguage

                        })

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

                      .then(data => {

                          document.getElementById("translation").innerText = data.translation;

                      }).catch(error => console.error("Error translating text:", error));

                };

                recognition.start();

            }

        }

        function pauseRecording() {

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

                mediaRecorder.pause();

                recognition.stop();

            }

        }

        function stopRecording() {

            if (isRecording) {

                mediaRecorder.stop();

                isRecording = false;

                audioChunks = [];

                recognition.stop();

            }

        }

    </script>

</body>

</html>

Транслетор.пу

import openai

import requests

import json

def transcribe\_audio(audio\_base64):

    openai.api\_key = 'your-openai-api-key'  # Replace with your OpenAI API key

    response = openai.Audio.transcribe(

        model="whisper-1",

        file=openai.File.create(file=audio\_base64, purpose='transcription'),

        language="auto"  # Automatically detect the language

    )

    return response['text']

def translate\_text(text, target\_language):

    api\_url = "https://translate.google.com/\_/TranslateWebserverUi/data/batchexecute"

    headers = {

        "Content-Type": "application/x-www-form-urlencoded;charset=UTF-8"

    }

    data = {

        "f.req": json.dumps([[[text, target\_language, "auto", "text"]]]),

        "at": "your-google-translate-api-key"  # Replace with your Google Translate API key

    }

    response = requests.post(api\_url, headers=headers, data=data)

    if response.status\_code == 200:

        try:

            result = json.loads(response.text[6:])[0][2]

            return result

        except Exception as e:

            print(f"Error parsing translation response: {e}")

            return None

    else:

        print(f"Translation request failed with status code {response.status\_code}")

        return None

import openai

import requests

import json

def transcribe\_audio(audio\_base64):

    openai.api\_key = 'your-openai-api-key'  # Replace with your OpenAI API key

    response = openai.Audio.transcribe(

        model="whisper-1",

        file=openai.File.create(file=audio\_base64, purpose='transcription'),

        language="auto"  # Automatically detect the language

    )

    return response['text']

def translate\_text(text, target\_language):

    api\_url = "https://translate.google.com/\_/TranslateWebserverUi/data/batchexecute"

    headers = {

        "Content-Type": "application/x-www-form-urlencoded;charset=UTF-8"

    }

    data = {

        "f.req": json.dumps([[[text, target\_language, "auto", "text"]]]),

        "at": "your-google-translate-api-key"  # Replace with your Google Translate API key

    }

    response = requests.post(api\_url, headers=headers, data=data)

    if response.status\_code == 200:

        try:

            result = json.loads(response.text[6:])[0][2]

            return result

        except Exception as e:

            print(f"Error parsing translation response: {e}")

            return None

    else:

        print(f"Translation request failed with status code {response.status\_code}")

        return None

Better version

App5.py

import os

from flask import Flask, render\_template, request, jsonify, send\_from\_directory, flash, url\_for

from flask\_sqlalchemy import SQLAlchemy

import logging

import speech\_recognition as sr

from pydub import AudioSegment

from forms import ClientForm

from models import db, Client  # Import db and Client

import openai  # Import the openai module

from translator import transcribe\_audio, translate\_text

import requests

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

app.config.from\_object('config.Config')

db.init\_app(app)  # Initialize db with the app

# Set up logging

logging.basicConfig(level=logging.DEBUG)

@app.route('/')

def home():

    return render\_template('home.html')

@app.route('/voice-recorder')

def voice\_recorder():

    return render\_template('voice\_to\_text.html')

@app.route('/visa')

def visa():

    return render\_template('visa.html')

@app.route('/visa/immigrant')

def immigrant\_visas():

    return render\_template('immigrant\_visas.html')

@app.route('/visa/non-immigrant')

def non\_immigrant\_visas():

    return render\_template('non\_immigrant\_visas.html')

@app.route('/client-info', methods=['GET', 'POST'])

def client\_info():

    form = ClientForm()

    if form.validate\_on\_submit():

        client = Client(

            full\_name=form.full\_name.data,

            alien\_registration\_number=form.alien\_registration\_number.data,

            uscis\_online\_account\_number=form.uscis\_online\_account\_number.data,

            date\_of\_birth=form.date\_of\_birth.data.strftime('%m/%d/%Y'),

            country\_of\_birth=form.country\_of\_birth.data,

            country\_of\_citizenship=form.country\_of\_citizenship.data,

            ssn=form.ssn.data,

            gender=form.gender.data,

            marital\_status=form.marital\_status.data,

            spouse\_full\_name=form.spouse\_full\_name.data,

            marriage\_date=form.marriage\_date.data,

            number\_of\_children=form.number\_of\_children.data,

            children\_names\_dobs=form.children\_names\_dobs.data,

            physical\_address=form.physical\_address.data,

            mailing\_address=form.mailing\_address.data,

            telephone\_number=form.telephone\_number.data,

            email\_address=form.email\_address.data,

            residential\_history=form.residential\_history.data,

            employment\_status=form.employment\_status.data,

            current\_employer\_name\_address=form.current\_employer\_name\_address.data,

            job\_title=form.job\_title.data,

            start\_date\_current\_job=form.start\_date\_current\_job.data,

            employment\_history=form.employment\_history.data,

            previous\_us\_visas=form.previous\_us\_visas.data,

            previous\_us\_entries=form.previous\_us\_entries.data,

            deportation\_history=form.deportation\_history.data,

            criminal\_history=form.criminal\_history.data,

            political\_social\_orgs=form.political\_social\_orgs.data,

            languages\_spoken=form.languages\_spoken.data,

            emergency\_contact=form.emergency\_contact.data,

            other\_citizenship=form.other\_citizenship.data,

            travel\_purpose=form.travel\_purpose.data,

            countries\_to\_visit=form.countries\_to\_visit.data,

            length\_of\_trip=form.length\_of\_trip.data,

            intended\_departure\_date=form.intended\_departure\_date.data,

            travel\_document\_type=form.travel\_document\_type.data,

            summary=form.summary.data,

        )

        db.session.add(client)

        db.session.commit()

        pdf\_file = create\_pdf(client)

        flash('Client information has been saved!', 'success')

        return send\_from\_directory(directory='client\_applications', path=pdf\_file, as\_attachment=True)

    return render\_template('client.html', form=form)

@app.route('/applications')

def applications():

    pdf\_path = 'C:/Users/Elena/Documents/GitHub/Imigration\_project\_new\_res/Addition\_information'

    pdf\_files = [f for f in os.listdir(pdf\_path) if f.endswith('.pdf')]

    return render\_template('applications.html', pdf\_files=pdf\_files, pdf\_path=pdf\_path)

@app.route('/view-pdf/<filename>')

def view\_pdf(filename):

    return send\_from\_directory('C:/Users/Elena/Documents/GitHub/Imigration\_project\_new\_res/Addition\_information', filename)

@app.route('/edit-pdf/<filename>')

def edit\_pdf(filename):

    return render\_template('edit\_pdf.html', filename=filename)

@app.route('/save-pdf', methods=['POST'])

def save\_pdf():

    data = request.json

    file\_name = data['fileName']

    pdf\_data = data['pdfData']

    save\_path = os.path.join('C:/Users/Elena/Documents/GitHub/Imigration\_project\_new\_res/pdf\_Application', file\_name)

    with open(save\_path, 'wb') as f:

        f.write(bytes.fromhex(pdf\_data))

    return jsonify({"message": "PDF saved successfully!"})

@app.route('/online-helper', methods=['GET', 'POST'])

def online\_helper():

    if request.method == 'POST':

        user\_info = request.form['user\_info']

        response = requests.post('http://localhost:5000/get-advice', json={'user\_info': user\_info})

        if response.status\_code == 200:

            try:

                response\_data = response.json()

                advice = response\_data.get('advice', 'No advice available')

            except requests.exceptions.JSONDecodeError as e:

                advice = f"Error decoding response: {str(e)}"

        else:

            advice = f"Error: Received status code {response.status\_code}"

        return render\_template('online\_helper\_result.html', advice=advice)

    return render\_template('online\_helper.html')

def create\_pdf(client):

    from reportlab.lib.pagesizes import letter

    from reportlab.pdfgen import canvas

    pdf\_file = os.path.join('client\_applications', f'{client.full\_name.replace(" ", "\_")}.pdf')

    c = canvas.Canvas(pdf\_file, pagesize=letter)

    c.drawString(100, 750, f"Full Name: {client.full\_name}")

    c.drawString(100, 735, f"Alien Registration Number: {client.alien\_registration\_number}")

    c.drawString(100, 720, f"USCIS Online Account Number: {client.uscis\_online\_account\_number}")

    c.drawString(100, 705, f"Date of Birth: {client.date\_of\_birth}")

    c.drawString(100, 690, f"Country of Birth: {client.country\_of\_birth}")

    c.drawString(100, 675, f"Country of Citizenship: {client.country\_of\_citizenship}")

    c.drawString(100, 660, f"U.S. Social Security Number: {client.ssn}")

    c.drawString(100, 645, f"Gender: {client.gender}")

    c.drawString(100, 630, f"Marital Status: {client.marital\_status}")

    c.drawString(100, 615, f"Number of Children: {client.number\_of\_children}")

    children\_list = eval(client.children\_names\_dobs) if client.children\_names\_dobs else []

    for i, child in enumerate(children\_list):

        c.drawString(100, 600 - i \* 15, f"Child {i+1} - Name: {child['first\_name']} {child['last\_name']}, Date of Birth: {child['date\_of\_birth']}")

    c.drawString(100, 540, f"Current Physical Address: {client.physical\_address}")

    c.drawString(100, 525, f"Mailing Address: {client.mailing\_address}")

    c.drawString(100, 510, f"Telephone Number: {client.telephone\_number}")

    c.drawString(100, 495, f"Email Address: {client.email\_address}")

    c.drawString(100, 480, f"Residential History: {client.residential\_history}")

    c.drawString(100, 465, f"Employment Status: {client.employment\_status}")

    c.drawString(100, 450, f"Current Employer Name and Address: {client.current\_employer\_name\_address}")

    c.drawString(100, 435, f"Job Title: {client.job\_title}")

    c.drawString(100, 420, f"Start Date of Current Job: {client.start\_date\_current\_job}")

    c.drawString(100, 405, f"Employment History: {client.employment\_history}")

    c.drawString(100, 390, f"Previous U.S. Visas: {client.previous\_us\_visas}")

    c.drawString(100, 375, f"Previous U.S. Entries: {client.previous\_us\_entries}")

    c.drawString(100, 360, f"Deportation History: {client.deportation\_history}")

    c.drawString(100, 345, f"Criminal History: {client.criminal\_history}")

    c.drawString(100, 330, f"Political or Social Organizations: {client.political\_social\_orgs}")

    c.drawString(100, 315, f"Languages Spoken: {client.languages\_spoken}")

    c.drawString(100, 300, f"Emergency Contact: {client.emergency\_contact}")

    c.drawString(100, 285, f"Other Countries of Citizenship: {client.other\_citizenship}")

    c.drawString(100, 270, f"Purpose of Travel: {client.travel\_purpose}")

    c.drawString(100, 255, f"Countries to Visit: {client.countries\_to\_visit}")

    c.drawString(100, 240, f"Expected Length of Trip: {client.length\_of\_trip}")

    c.drawString(100, 225, f"Date of Intended Departure: {client.intended\_departure\_date}")

    c.drawString(100, 210, f"Travel Document Type: {client.travel\_document\_type}")

    c.drawString(100, 195, f"Summary: {client.summary}")

    c.save()

    return pdf\_file

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

def transcribe\_audio\_route():

    if 'audio' not in request.files:

        return jsonify({"error": "No audio file"}), 400

    audio\_file = request.files['audio']

    recognizer = sr.Recognizer()

    try:

        # Convert audio file to WAV format

        audio = AudioSegment.from\_file(audio\_file)

        audio.export("temp.wav", format="wav")

        # Read the WAV file with speech\_recognition

        with sr.AudioFile("temp.wav") as source:

            audio\_data = recognizer.record(source)

            transcription = recognizer.recognize\_google(audio\_data, language="auto")

            return jsonify({"transcription": transcription})

    except sr.UnknownValueError:

        return jsonify({"error": "Could not understand audio"}), 400

    except sr.RequestError as e:

        return jsonify({"error": f"Could not request results from Google Speech Recognition service; {e}"}), 500

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

def translate\_text\_route():

    data = request.json

    text = data.get('text')

    target\_language = data.get('target\_language')

    if not text or not target\_language:

        return jsonify({"error": "Text and target language are required."}), 400

    try:

        translation = translate\_text(text, target\_language)

        return jsonify({"translation": translation})

    except Exception as e:

        logging.error(f"Error translating text: {e}")

        return jsonify({"error": str(e)}), 500

@app.route('/get-advice', methods=['POST'])

def get\_advice():

    user\_info = request.json.get('user\_info')

    try:

        response = openai.ChatCompletion.create(

            model="gpt-3.5-turbo",  # Replace with your fine-tuned model ID

            messages=[

                {"role": "system", "content": "You are a helpful assistant."},

                {"role": "user", "content": user\_info}

            ],

            max\_tokens=150

        )

        advice = response.choices[0].message['content'].strip()

        return jsonify({'advice': advice})

    except openai.error.OpenAIError as e:

        logging.error(f"OpenAI API error: {e}")

        return jsonify({'error': str(e)}), 500

    except Exception as e:

        logging.error(f"General error: {e}")

        return jsonify({'error': str(e)}), 500

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

    with app.app\_context():

        db.create\_all()  # Create database tables

    app.run(debug=True)

Transletor.py

import openai

import requests

import json

import os

from google.cloud import translate\_v2 as translate

from google.oauth2 import service\_account

# Set up Google Translate API credentials

credentials\_path = os.getenv('GOOGLE\_APPLICATION\_CREDENTIALS')

if credentials\_path is None:

    raise ValueError("The GOOGLE\_APPLICATION\_CREDENTIALS environment variable is not set.")

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

translate\_client = translate.Client(credentials=credentials)

def translate\_text(text, target\_language):

    translation = translate\_client.translate(text, target\_language=target\_language)

    return translation['translatedText']

def transcribe\_audio(audio\_base64):

    openai.api\_key = os.getenv('OPENAI\_API\_KEY')  # Replace with your OpenAI API key

    response = openai.Audio.transcribe(

        model="whisper-1",

        file=openai.File.create(file=audio\_base64, purpose='transcription'),

        language="auto"  # Automatically detect the language

    )

    return response['text']

application.html   
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>View Applications</title>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/pdf-lib/1.17.1/pdf-lib.min.js"></script>

</head>

<body>

    <h1>PDF Applications</h1>

    <ul>

        {% for pdf in pdf\_files %}

        <li>

            <a href="#" onclick="viewPdf('{{ pdf }}')">{{ pdf }}</a>

        </li>

        {% endfor %}

    </ul>

    <div id="pdfViewer" style="display: none;">

        <h2>Edit PDF: <span id="pdfFileName"></span></h2>

        <canvas id="pdfCanvas"></canvas>

        <button onclick="savePdf()">Save PDF</button>

    </div>

    <script>

        let pdfDoc = null;

        let pageNum = 1;

        let pdfFileName = '';

        function viewPdf(filename) {

            pdfFileName = filename;

            document.getElementById('pdfFileName').textContent = filename;

            document.getElementById('pdfViewer').style.display = 'block';

            fetch(`/view-pdf/${filename}`)

                .then(response => response.arrayBuffer())

                .then(data => {

                    return PDFLib.PDFDocument.load(data);

                })

                .then(pdf => {

                    pdfDoc = pdf;

                    renderPage(pageNum);

                });

        }

        function renderPage(num) {

            pdfDoc.getPage(num).then(page => {

                const viewport = page.getViewport({ scale: 1.5 });

                const canvas = document.getElementById('pdfCanvas');

                const context = canvas.getContext('2d');

                canvas.height = viewport.height;

                canvas.width = viewport.width;

                const renderContext = {

                    canvasContext: context,

                    viewport: viewport

                };

                page.render(renderContext);

            });

        }

        function savePdf() {

            pdfDoc.save().then(pdfBytes => {

                const hexString = Array.from(new Uint8Array(pdfBytes))

                    .map(b => b.toString(16).padStart(2, '0'))

                    .join('');

                fetch('/save-pdf', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json'

                    },

                    body: JSON.stringify({

                        fileName: pdfFileName,

                        pdfData: hexString

                    })

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

                .then(data => {

                    alert(data.message);

                });

            });

        }

    </script>

</body>

</html>

edit\_pdf.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Edit PDF</title>

</head>

<body>

    <h1>Edit PDF: {{ filename }}</h1>

    <embed src="{{ url\_for('view\_pdf', filename=filename) }}" width="100%" height="600px" type="application/pdf">

    <button onclick="savePdf()">Save PDF</button>

    <script>

        function savePdf() {

            const fileName = "{{ filename }}";

            const pdfData = ...; // Your code to get the edited PDF data

            fetch('/pdf/save-pdf', {

                method: 'POST',

                headers: {

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

                },

                body: JSON.stringify({ fileName, pdfData }),

            })

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

            .then(data => {

                alert(data.message);

            })

            .catch((error) => {

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

            });

        }

    </script>

</body>

</html>

**Introduction:**

"Hello everyone, today I am excited to present to you our comprehensive Immigration Consulting Services Application. This platform is designed to streamline the process of managing immigration-related tasks, including client information management, voice-to-text transcription, real-time language translation, and PDF form editing. Let's dive into the features of this application."

**Home Page:**

*Navigate to the home page of your application.*

"Here we have the home page of our application, which provides a clear and user-friendly interface for navigating through the various functionalities available."

**Client Information Management:**

*Click on the "Client Information" section.*

"This section allows immigration consultants to input and manage detailed client information. As you can see, the form captures all necessary details such as personal information, employment history, travel details, and more. Once the information is submitted, it is stored in our database, and a PDF summary of the client’s information is generated."

*Submit a sample form to demonstrate the PDF generation.*

**Voice Recorder and Real-time Translation:**

*Navigate to the "Voice Recorder" section.*

"Next, we have the voice recorder feature. This tool allows users to record their voice, which is then transcribed into text in real-time. Additionally, we support multiple languages including Ukrainian, English, Spanish, Bulgarian, and Polish."

*Demonstrate recording a voice message and show the real-time transcription.*

"Moreover, the transcribed text can be automatically translated into another language of your choice. This is particularly useful for clients who are more comfortable speaking in their native language."

*Show the translated text in the translation window.*

**PDF Applications:**

*Navigate to the "PDF Applications" section.*

"In the PDF Applications section, we provide a list of common immigration forms that can be edited directly within the application. Let’s select one of these forms."

*Click on a PDF form to open it.*

"Here you can see the selected PDF form. Users can view the form directly on this page and save their changes. This feature simplifies the process of filling out and managing immigration documents."

*Demonstrate editing and saving changes to the PDF form.*

**Online Helper:**

*Navigate to the "Online Helper" section.*

"Lastly, we have an online helper feature powered by OpenAI’s GPT-3.5-turbo model. Users can input their queries related to immigration services, and our AI assistant will provide relevant advice and information."

*Demonstrate by entering a sample query and showing the AI-generated advice.*

**Conclusion:**

"In conclusion, our Immigration Consulting Services Application offers a comprehensive set of tools to assist immigration consultants and their clients. From managing client information and generating PDF summaries to real-time voice-to-text transcription and translation, our platform aims to make the immigration process more efficient and accessible. Thank you for your attention, and I am happy to answer any questions you may have."