<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Chat with PDF</title>

    <style>

        body {

            background-color: #121212;

            color: white;

            font-family: Arial, sans-serif;

            margin: 0;

            padding: 0;

        }

        .container {

            max-width: 800px;

            margin: 0 auto;

            padding: 20px;

        }

        .header {

            background-color: #1f3a6e;

            color: white;

            font-size: 30px;

            font-weight: bold;

            text-align: center;

            padding: 20px;

            border-radius: 10px;

            margin-bottom: 20px;

        }

        .chat-container {

            margin-top: 20px;

            height: 400px;

            overflow-y: auto;

            border: 1px solid #333;

            padding: 20px;

            border-radius: 10px;

            background-color: #1e1e1e;

        }

        .message-bubble {

            background-color: #1F3A6E;

            color: white;

            padding: 15px;

            margin: 10px 0;

            border-radius: 15px;

            max-width: 75%;

            margin-left: 10px;

            margin-right: auto;

            word-break: break-word;

        }

        .response-bubble {

            background-color: #333;

            color: white;

            padding: 15px;

            margin: 10px 0;

            border-radius: 15px;

            max-width: 75%;

            margin-left: auto;

            margin-right: 10px;

            position: relative;

            word-break: break-word;

        }

        .source-citation {

            font-size: 0.85em;

            margin-top: 8px;

            padding-top: 8px;

            border-top: 1px solid rgba(255, 255, 255, 0.2);

            color: #aaa;

        }

        .input-container {

            margin-top: 20px;

            display: flex;

            gap: 10px;

            align-items: center;

        }

        .input-container input {

            background-color: #333;

            color: white;

            padding: 12px;

            font-size: 16px;

            border-radius: 10px;

            border: none;

            flex-grow: 1;

        }

        .input-container button {

            background-color: #FF5722;

            color: white;

            padding: 12px 20px;

            font-size: 16px;

            border-radius: 20px;

            border: none;

            cursor: pointer;

            transition: background-color 0.3s;

        }

        .input-container button:hover {

            background-color: #e64a19;

        }

        .input-container button:disabled {

            background-color: #666;

            cursor: not-allowed;

        }

        .tts-toggle {

            background-color: #4CAF50;

            color: white;

            padding: 10px 15px;

            font-size: 16px;

            border-radius: 20px;

            border: none;

            cursor: pointer;

            display: flex;

            align-items: center;

            gap: 5px;

            transition: background-color 0.3s;

        }

        .tts-toggle.off {

            background-color: #f44336;

        }

        .tts-toggle:hover {

            opacity: 0.9;

        }

        .file-upload {

            background-color: #333;

            padding: 15px;

            border-radius: 10px;

            margin-top: 20px;

        }

        .file-upload button {

            background-color: #FF5722;

            color: white;

            padding: 12px;

            font-size: 14px;

            border-radius: 10px;

            border: none;

            cursor: pointer;

            transition: background-color 0.3s;

        }

        .file-upload button:hover {

            background-color: #e64a19;

        }

        footer {

            background-color: #1f3a6e;

            padding: 10px;

            text-align: center;

            color: white;

            margin-top: 20px;

            border-radius: 10px;

            font-size: 14px;

        }

        /\* Voice Input Styles \*/

        .input-wrapper {

            display: flex;

            align-items: center;

            flex-grow: 1;

            position: relative;

        }

        #question-input {

            padding-right: 40px;

            width: 100%;

            transition: background-color 0.3s;

        }

        #question-input:focus {

            outline: none;

            box-shadow: 0 0 0 2px #1f3a6e;

        }

        .voice-button {

            background: none;

            border: none;

            cursor: pointer;

            padding: 8px;

            color: #555;

            border-radius: 50%;

            margin-left: 8px;

            display: flex;

            align-items: center;

            justify-content: center;

            transition: all 0.3s ease;

            position: absolute;

            right: 8px;

        }

        .voice-button:hover {

            background-color: rgba(255, 255, 255, 0.1);

            color: #fff;

        }

        .voice-button.listening {

            color: #ff4444;

            background-color: rgba(255, 0, 0, 0.1);

            animation: pulse 1.5s infinite;

        }

        .voice-input {

            background-color: rgba(100, 200, 255, 0.2) !important;

            transition: background-color 0.5s ease;

        }

        .popup {

            display: none;

            position: fixed;

            z-index: 100;

            left: 0;

            top: 0;

            width: 100%;

            height: 100%;

            background-color: rgba(0,0,0,0.7);

        }

        .popup-content {

            background-color: #1f3a6e;

            margin: 15% auto;

            padding: 20px;

            border: 1px solid #333;

            width: 80%;

            max-width: 500px;

            border-radius: 8px;

            position: relative;

            box-shadow: 0 4px 8px rgba(0,0,0,0.2);

        }

        .popup-close-btn {

            position: absolute;

            right: 10px;

            top: 10px;

            font-size: 20px;

            cursor: pointer;

            color: #aaa;

        }

        .popup-close-btn:hover {

            color: #fff;

        }

        .stop-btn {

            margin-top: 10px;

            padding: 8px 15px;

            background-color: #f44336;

            color: white;

            border: none;

            border-radius: 5px;

            cursor: pointer;

            font-weight: bold;

            transition: background-color 0.3s;

        }

        .stop-btn:hover {

            background-color: #d32f2f;

        }

        @keyframes pulse {

            0% { transform: scale(1); }

            50% { transform: scale(1.1); }

            100% { transform: scale(1); }

        }

        /\* Typing indicator styles \*/

        .typing-indicator {

            display: inline-block;

            margin-left: 5px;

        }

        .typing-dot {

            display: inline-block;

            width: 8px;

            height: 8px;

            background-color: #aaa;

            border-radius: 50%;

            margin-right: 3px;

            animation: typingAnimation 1.4s infinite ease-in-out;

        }

        .typing-dot:nth-child(1) {

            animation-delay: 0s;

        }

        .typing-dot:nth-child(2) {

            animation-delay: 0.2s;

        }

        .typing-dot:nth-child(3) {

            animation-delay: 0.4s;

        }

        @keyframes typingAnimation {

            0%, 60%, 100% { transform: translateY(0); }

            30% { transform: translateY(-5px); }

        }

        /\* Audio controls styles \*/

        .audio-controls {

            display: flex;

            align-items: center;

            gap: 10px;

            margin-top: 10px;

            padding-top: 10px;

            border-top: 1px solid rgba(255, 255, 255, 0.1);

            user-select: none;

        }

        .audio-controls button {

            background: none;

            border: none;

            color: white;

            cursor: pointer;

            font-size: 1.2em;

            padding: 5px;

            border-radius: 50%;

            width: 36px;

            height: 36px;

            display: flex;

            align-items: center;

            justify-content: center;

            transition: background-color 0.2s;

        }

        .audio-controls button:hover {

            background-color: rgba(255, 255, 255, 0.1);

        }

        .audio-controls button:focus {

            outline: 2px solid #4CAF50;

            outline-offset: 2px;

        }

        .progress-container {

            flex-grow: 1;

            height: 4px;

            background-color: #333;

            border-radius: 2px;

            overflow: hidden;

            cursor: pointer;

        }

        .progress-bar {

            height: 100%;

            background-color: #4CAF50;

            width: 0%;

            transition: width 0.1s linear;

        }

        .time-display {

            font-size: 0.8em;

            color: #aaa;

            min-width: 100px;

            text-align: center;

        }

        /\* Mobile responsiveness \*/

        @media (max-width: 768px) {

            .container {

                padding: 10px;

            }

            .header {

                font-size: 24px;

                padding: 15px;

            }

            .chat-container {

                height: 300px;

                padding: 10px;

            }

            .input-container {

                flex-wrap: wrap;

            }

            .input-container input {

                width: 100%;

                margin-bottom: 10px;

            }

            .tts-toggle, #ask-button {

                flex: 1;

            }

            .message-bubble, .response-bubble {

                max-width: 85%;

            }

        }

        /\* Loading state for buttons \*/

        .button-loading {

            position: relative;

            pointer-events: none;

        }

        .button-loading::after {

            content: "";

            position: absolute;

            width: 16px;

            height: 16px;

            top: 0;

            left: 0;

            right: 0;

            bottom: 0;

            margin: auto;

            border: 3px solid transparent;

            border-top-color: #ffffff;

            border-radius: 50%;

            animation: button-loading-spinner 1s ease infinite;

        }

        @keyframes button-loading-spinner {

            from {

                transform: rotate(0turn);

            }

            to {

                transform: rotate(1turn);

            }

        }

    </style>

    <!-- Include marked.js for Markdown rendering -->

    <script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>

</head>

<body>

    <div class="container">

        <div class="header">

            Chat with PDF using Gemini 💁

        </div>

        <!-- Popup for messages -->

        <div id="popup" class="popup">

            <div class="popup-content">

                <span id="popup-close-btn" class="popup-close-btn">&times;</span>

                <p id="popup-message"></p>

                <div id="popup-buttons"></div>

            </div>

        </div>

        <div class="chat-container" id="chat-container">

            <!-- Chat messages will be appended here -->

        </div>

        <div class="input-container">

            <div class="input-wrapper">

                <input type="text" id="question-input" placeholder="Ask a Question based on the PDF content...">

                <button id="voice-button" class="voice-button" title="Voice Input">

                    <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round">

                        <path d="M12 1a3 3 0 0 0-3 3v8a3 3 0 0 0 6 0V4a3 3 0 0 0-3-3z"></path>

                        <path d="M19 10v2a7 7 0 0 1-14 0v-2"></path>

                        <line x1="12" y1="19" x2="12" y2="23"></line>

                        <line x1="8" y1="23" x2="16" y2="23"></line>

                    </svg>

                </button>

            </div>

            <button id="tts-toggle" class="tts-toggle" title="Toggle Text-to-Speech">

                🔊 TTS On

            </button>

            <button id="ask-button" onclick="askQuestion()">Ask</button>

        </div>

        <div class="file-upload">

            <input type="file" id="pdf-upload" multiple accept=".pdf">

            <button onclick="uploadFiles()">Submit & Process</button>

        </div>

        <footer>

            Powered by Gemini | Chatbot

        </footer>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.5.4/socket.io.js"></script>

    <script>

        const socket = io();

        const chatContainer = document.getElementById("chat-container");

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

        const popupMessage = document.getElementById("popup-message");

        const popupCloseBtn = document.getElementById("popup-close-btn");

        const popupButtons = document.getElementById("popup-buttons");

        const questionInput = document.getElementById("question-input");

        const voiceButton = document.getElementById("voice-button");

        const ttsToggle = document.getElementById("tts-toggle");

        const askButton = document.getElementById("ask-button");

        // State variables

        let ttsEnabled = true;

        let recognition = null;

        let isListening = false;

        let currentResponseBubble = null;

        let currentAudio = null;

        let isPlaying = false;

        let audioProgressInterval = null;

        let fullResponseText = '';

        // Initialize speech recognition

        function initSpeechRecognition() {

            const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;

            if (!SpeechRecognition) {

                showPopup("Speech recognition is not supported in your browser. Try Chrome or Edge.");

                return false;

            }

            try {

                recognition = new SpeechRecognition();

                recognition.continuous = false;

                recognition.interimResults = true;

                recognition.lang = navigator.language || 'en-US';

                recognition.onstart = () => {

                    isListening = true;

                    voiceButton.classList.add('listening');

                    showPopup("Listening... Click the microphone icon or press ESC to stop.", true);

                };

                recognition.onresult = (event) => {

                    const speechResult = event.results[event.results.length - 1];

                    const transcript = speechResult[0].transcript.trim();

                    if (transcript) {

                        questionInput.value = transcript;

                        questionInput.classList.add('voice-input');

                        setTimeout(() => {

                            questionInput.classList.remove('voice-input');

                        }, 1000);

                    }

                };

                recognition.onend = () => {

                    isListening = false;

                    voiceButton.classList.remove('listening');

                    closePopup();

                    // Auto-submit if text was captured

                    if (questionInput.value.trim()) {

                        askQuestion();

                    }

                };

                recognition.onerror = (event) => {

                    console.error('Speech recognition error:', event.error);

                    isListening = false;

                    voiceButton.classList.remove('listening');

                    switch(event.error) {

                        case 'no-speech':

                            showPopup("No speech detected. Try again.");

                            break;

                        case 'audio-capture':

                            showPopup("No microphone found. Check your device.");

                            break;

                        case 'not-allowed':

                            showPopup("Microphone access denied. Check permissions.");

                            break;

                        default:

                            showPopup("Error: " + event.error);

                    }

                };

                return true;

            } catch (error) {

                console.error('Failed to initialize speech recognition:', error);

                showPopup("Could not initialize speech recognition.");

                return false;

            }

        }

        // Toggle voice recognition

        function toggleVoiceRecognition() {

            if (!recognition) {

                if (!initSpeechRecognition()) {

                    return;

                }

            }

            if (isListening) {

                recognition.stop();

            } else {

                try {

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

                        .then(stream => {

                            stream.getTracks().forEach(track => track.stop());

                            recognition.start();

                        })

                        .catch(error => {

                            console.error("Microphone access error:", error);

                            showPopup("Could not access microphone. Check permissions.");

                        });

                } catch (error) {

                    console.error("Microphone access error:", error);

                    showPopup("Could not access microphone. Check permissions.");

                }

            }

        }

        // Stop speech recognition

        function stopListening() {

            if (recognition && isListening) {

                recognition.stop();

                isListening = false;

                voiceButton.classList.remove('listening');

                closePopup();

            }

        }

        // Toggle TTS functionality

        function toggleTTS() {

            ttsEnabled = !ttsEnabled;

            ttsToggle.textContent = ttsEnabled ? '🔊 TTS On' : '🔇 TTS Off';

            ttsToggle.classList.toggle('off', !ttsEnabled);

            showPopup(`Text-to-speech ${ttsEnabled ? 'enabled' : 'disabled'}`);

        }

        // Show popup message

        function showPopup(message, isListeningPopup = false) {

            popupMessage.textContent = message;

            popupButtons.innerHTML = '';

            if (isListeningPopup) {

                const stopBtn = document.createElement('button');

                stopBtn.textContent = 'Stop Listening';

                stopBtn.className = 'stop-btn';

                stopBtn.onclick = stopListening;

                popupButtons.appendChild(stopBtn);

            }

            popup.style.display = "block";

            if (!isListeningPopup) {

                setTimeout(() => {

                    closePopup();

                }, 3000);

            }

        }

        // Close popup

        function closePopup() {

            popup.style.display = "none";

        }

        // Handle socket events for streaming responses

        socket.on('response\_start', function(data) {

            const chatContainer = document.getElementById('chat-container');

            // Create new response bubble for streaming response

            currentResponseBubble = document.createElement('div');

            currentResponseBubble.className = 'response-bubble';

            // Add typing indicator

            const typingIndicator = document.createElement('div');

            typingIndicator.className = 'typing-indicator';

            typingIndicator.innerHTML = `

                <span class="typing-dot"></span>

                <span class="typing-dot"></span>

                <span class="typing-dot"></span>

            `;

            currentResponseBubble.appendChild(typingIndicator);

            // Store the full text for later

            fullResponseText = data.full\_text;

            // If there's audio data, prepare the audio controls

            if (data.audio) {

                currentAudio = new Audio(`data:audio/mp3;base64,${data.audio}`);

                currentAudio.onplay = function() {

                    isPlaying = true;

                    updateAudioControls();

                    startProgressTracking();

                };

                currentAudio.onpause = function() {

                    isPlaying = false;

                    updateAudioControls();

                    stopProgressTracking();

                };

                currentAudio.onended = function() {

                    isPlaying = false;

                    updateAudioControls();

                    stopProgressTracking();

                };

                // Auto-play if TTS is enabled

                if (ttsEnabled) {

                    currentAudio.play().catch(e => console.log("Audio play failed:", e));

                }

            }

            chatContainer.appendChild(currentResponseBubble);

            chatContainer.scrollTop = chatContainer.scrollHeight;

        });

        socket.on('response\_chunk', function(data) {

            if (!currentResponseBubble) return;

            // Remove typing indicator if it exists

            const typingIndicator = currentResponseBubble.querySelector('.typing-indicator');

            if (typingIndicator) {

                currentResponseBubble.removeChild(typingIndicator);

            }

            // Add the new word

            const wordSpan = document.createElement('span');

            wordSpan.textContent = data.word;

            currentResponseBubble.appendChild(wordSpan);

            // Scroll to bottom

            chatContainer.scrollTop = chatContainer.scrollHeight;

        });

        socket.on('response\_end', function(data) {

            if (!currentResponseBubble) return;

            // Add audio controls if we have audio

            if (currentAudio) {

                const audioControls = document.createElement('div');

                audioControls.className = 'audio-controls';

                audioControls.innerHTML = `

                    <button id="play-pause-btn">${isPlaying ? '⏸' : '▶'}</button>

                    <div class="progress-container" id="progress-container">

                        <div class="progress-bar" id="progress-bar"></div>

                    </div>

                    <span class="time-display" id="time-display">0:00 / 0:00</span>

                `;

                currentResponseBubble.appendChild(audioControls);

                // Add event listeners

                document.getElementById('play-pause-btn').addEventListener('click', toggleAudioPlayback);

                setupAudioSeeking();

                // Update time display

                updateTimeDisplay();

            }

            // Re-enable the ask button

            askButton.disabled = false;

            askButton.textContent = 'Ask';

            askButton.classList.remove('button-loading');

            currentResponseBubble = null;

        });

        socket.on('response\_error', function(data) {

            const chatContainer = document.getElementById('chat-container');

            const errorBubble = document.createElement('div');

            errorBubble.className = 'response-bubble';

            errorBubble.innerHTML = `<span style="color: #ff6b6b">${data.message}</span>`;

            chatContainer.appendChild(errorBubble);

            chatContainer.scrollTop = chatContainer.scrollHeight;

            // Re-enable the ask button if it was disabled

            askButton.disabled = false;

            askButton.textContent = 'Ask';

            askButton.classList.remove('button-loading');

        });

        // Audio control functions

        function toggleAudioPlayback() {

            if (!currentAudio) return;

            if (isPlaying) {

                currentAudio.pause();

            } else {

                currentAudio.play().catch(e => console.log("Audio play failed:", e));

            }

        }

        function updateAudioControls() {

            const playPauseBtn = document.getElementById('play-pause-btn');

            if (playPauseBtn) {

                playPauseBtn.textContent = isPlaying ? '⏸' : '▶';

            }

        }

        function startProgressTracking() {

            stopProgressTracking();

            audioProgressInterval = setInterval(updateProgress, 100);

        }

        function stopProgressTracking() {

            if (audioProgressInterval) {

                clearInterval(audioProgressInterval);

                audioProgressInterval = null;

            }

        }

        function updateProgress() {

            if (!currentAudio) return;

            const progressBar = document.getElementById('progress-bar');

            const timeDisplay = document.getElementById('time-display');

            if (progressBar && timeDisplay) {

                const progress = (currentAudio.currentTime / currentAudio.duration) \* 100;

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

                // Update time display

                const currentMinutes = Math.floor(currentAudio.currentTime / 60);

                const currentSeconds = Math.floor(currentAudio.currentTime % 60);

                const durationMinutes = Math.floor(currentAudio.duration / 60);

                const durationSeconds = Math.floor(currentAudio.duration % 60);

                timeDisplay.textContent =

                    `${currentMinutes}:${currentSeconds < 10 ? '0' : ''}${currentSeconds} / ` +

                    `${durationMinutes}:${durationSeconds < 10 ? '0' : ''}${durationSeconds}`;

            }

        }

        function updateTimeDisplay() {

            if (!currentAudio) return;

            const timeDisplay = document.getElementById('time-display');

            if (timeDisplay) {

                const durationMinutes = Math.floor(currentAudio.duration / 60);

                const durationSeconds = Math.floor(currentAudio.duration % 60);

                timeDisplay.textContent =

                    `0:00 / ${durationMinutes}:${durationSeconds < 10 ? '0' : ''}${durationSeconds}`;

            }

        }

        function setupAudioSeeking() {

            const progressContainer = document.getElementById('progress-container');

            if (!progressContainer) return;

            progressContainer.addEventListener('click', function(e) {

                if (!currentAudio) return;

                const rect = this.getBoundingClientRect();

                const pos = (e.clientX - rect.left) / rect.width;

                currentAudio.currentTime = pos \* currentAudio.duration;

                updateProgress();

            });

        }

        function askQuestion() {

            const question = questionInput.value;

            if (!question) return;

            // Disable the ask button while processing

            askButton.disabled = true;

            askButton.textContent = 'Processing...';

            askButton.classList.add('button-loading');

            const chatContainer = document.getElementById('chat-container');

            const userBubble = document.createElement('div');

            userBubble.className = 'message-bubble';

            userBubble.innerHTML = `<strong>You:</strong> ${question}`;

            chatContainer.appendChild(userBubble);

            socket.emit('ask\_question', {

                question: question,

                tts: ttsEnabled

            });

            questionInput.value = '';

        }

        function uploadFiles() {

            const files = document.getElementById('pdf-upload').files;

            if (files.length === 0) {

                showPopup("Please select a PDF file first.");

                return;

            }

            showPopup("Processing PDFs...");

            const formData = new FormData();

            for (let i = 0; i < files.length; i++) {

                formData.append('pdf\_docs', files[i]);

            }

            fetch('/upload', {

                method: 'POST',

                body: formData,

            })

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

            .then(data => {

                showPopup(data.message || "PDFs processed successfully!");

            })

            .catch(error => {

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

                showPopup("Error uploading PDFs.");

            });

        }

        function handleConnectionErrors() {

            socket.on('connect\_error', (error) => {

                showPopup("Connection error. Trying to reconnect...");

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

            });

            socket.on('reconnect\_attempt', () => {

                console.log('Attempting to reconnect...');

            });

            socket.on('reconnect', () => {

                showPopup("Reconnected successfully!");

            });

            socket.on('disconnect', (reason) => {

                if (reason === 'io server disconnect') {

                    showPopup("Disconnected by server. Please refresh the page.");

                } else {

                    showPopup("Disconnected. Trying to reconnect...");

                }

            });

        }

        // Event listeners

        voiceButton.addEventListener('click', toggleVoiceRecognition);

        ttsToggle.addEventListener('click', toggleTTS);

        popupCloseBtn.addEventListener('click', closePopup);

        questionInput.addEventListener('keypress', (e) => {

            if (e.key === 'Enter') {

                askQuestion();

            }

        });

        // Initialize when page loads

        document.addEventListener('DOMContentLoaded', function() {

            initSpeechRecognition();

            // Add keyboard shortcuts

            document.addEventListener('keydown', (e) => {

                // Escape to stop listening

                if (e.key === 'Escape' && isListening) {

                    stopListening();

                }

                // Space to play/pause audio when focused on audio controls

                if (e.key === ' ' && document.activeElement.id === 'play-pause-btn') {

                    e.preventDefault();

                    toggleAudioPlayback();

                }

            });

            // Close popup when clicking outside

            window.addEventListener('click', (event) => {

                if (event.target === popup) {

                    closePopup();

                }

            });

            // Focus the input field on page load

            questionInput.focus();

            // Handle connection errors

            handleConnectionErrors();

        });

    </script>

</body>

</html>

from flask import Flask, render\_template, request

from flask\_socketio import SocketIO, emit

from PyPDF2 import PdfReader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

import os

import google.generativeai as genai

from langchain\_google\_genai import GoogleGenerativeAIEmbeddings

from langchain.vectorstores import FAISS

from langchain\_google\_genai import ChatGoogleGenerativeAI

from langchain.chains.question\_answering import load\_qa\_chain

from langchain.prompts import PromptTemplate

from langchain.docstore.document import Document

from dotenv import load\_dotenv

import re

from gtts import gTTS

import tempfile

import base64

from io import BytesIO

# Load environment variables

load\_dotenv()

# Set up Google API key for Gemini

genai.configure(api\_key=os.getenv("GOOGLE\_API\_KEY"))

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

app.config['SECRET\_KEY'] = 'your\_secret\_key'

socketio = SocketIO(app)

# Function to extract text from PDF files with page tracking

def get\_pdf\_text(pdf\_docs):

    """Extract text from PDF documents with page number metadata."""

    documents\_with\_metadata = []

    for pdf in pdf\_docs:

        try:

            pdf\_reader = PdfReader(pdf)

            pdf\_name = os.path.basename(pdf.filename)

            for page\_num, page in enumerate(pdf\_reader.pages, start=1):

                page\_text = page.extract\_text() or ""

                if page\_text.strip():

                    documents\_with\_metadata.append({

                        "text": page\_text,

                        "metadata": {

                            "source": pdf\_name,

                            "page": page\_num

                        }

                    })

        except Exception as e:

            print(f"Error processing {pdf.filename}: {str(e)}")

    return documents\_with\_metadata

# Function to split text into chunks with metadata

def get\_text\_chunks(documents):

    """Split text into chunks while preserving page metadata."""

    text\_splitter = RecursiveCharacterTextSplitter(

        chunk\_size=10000,

        chunk\_overlap=1000,

        separators=["\n\n", "\n", " ", ""]

    )

    chunks = []

    for doc in documents:

        text\_chunks = text\_splitter.split\_text(doc["text"])

        for chunk in text\_chunks:

            chunks.append(Document(

                page\_content=chunk,

                metadata=doc["metadata"]

            ))

    return chunks

# Function to create vector store

def get\_vector\_store(text\_chunks):

    """Create and save FAISS vector store with metadata."""

    embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")

    vector\_store = FAISS.from\_documents(text\_chunks, embeddings)

    vector\_store.save\_local("faiss\_index")

    return vector\_store

# Function to create conversational chain

def get\_conversational\_chain():

    """Create QA chain with page reference instructions."""

    prompt\_template = """

    You are an expert PDF analyzer. Answer the question in detail using only the provided context.

    Always include the source document name and exact page number where the information was found.

    Format page references like this: [Source: filename.pdf, Page: X]

    If the answer isn't in the context, say: "I couldn't find this information in the documents."

    Context: {context}

    Question: {question}

    Detailed Answer:

    """

    model = ChatGoogleGenerativeAI(model="gemini-1.5-flash", temperature=0.3)

    prompt = PromptTemplate(template=prompt\_template, input\_variables=["context", "question"])

    chain = load\_qa\_chain(model, chain\_type="stuff", prompt=prompt)

    return chain

# Function to convert text to speech

def text\_to\_speech(text, lang='en'):

    """Convert text to speech and return as base64 encoded string."""

    try:

        tts = gTTS(text=text, lang=lang, slow=False)

        mp3\_fp = BytesIO()

        tts.write\_to\_fp(mp3\_fp)

        mp3\_fp.seek(0)

        audio\_data = base64.b64encode(mp3\_fp.read()).decode('utf-8')

        return audio\_data

    except Exception as e:

        print(f"TTS Error: {str(e)}")

        return None

# Add this new function for word-by-word streaming

def generate\_word\_stream(text):

    """Generator that yields text word by word with timing information."""

    words = text.split()

    for word in words:

        # Calculate delay based on word length (adjust as needed)

        delay = max(0.1, min(len(word) \* 0.05, 0.5))  # Between 0.1 and 0.5 seconds

        yield {

            'word': word + ' ',

            'delay': delay

        }

# Modify the handle\_question function

@socketio.on('ask\_question')

def handle\_question(data):

    """Handle questions with improved page reference tracking and TTS."""

    user\_question = data.get('question')

    tts\_enabled = data.get('tts', False)

    if not user\_question:

        emit('response', {'message': 'Please ask a question.'})

        return

    try:

        # Load vector store

        embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")

        if not os.path.exists("faiss\_index"):

            emit('response', {'message': 'Please upload and process PDFs first.'})

            return

        db = FAISS.load\_local("faiss\_index", embeddings, allow\_dangerous\_deserialization=True)

        docs = db.similarity\_search(user\_question, k=4)

        # Collect source documents and their page numbers

        sources\_with\_pages = []

        for doc in docs:

            if hasattr(doc, 'metadata'):

                source = doc.metadata.get('source', 'unknown')

                page = doc.metadata.get('page', 0)

                sources\_with\_pages.append(f"{source} (Page {page})")

        # Update prompt to encourage placing page references at the end

        prompt\_template = """

        You are an expert PDF analyzer. Answer the question in detail using only the provided context.

        DO NOT include source citations or page numbers within your main answer text.

        At the very end of your response, list all sources used in this format: [Source: filename.pdf, Page: X]

        Context: {context}

        Question: {question}

        Detailed Answer:

        """

        # Generate response

        model = ChatGoogleGenerativeAI(model="gemini-1.5-flash", temperature=0.3)

        prompt = PromptTemplate(template=prompt\_template, input\_variables=["context", "question"])

        chain = load\_qa\_chain(model, chain\_type="stuff", prompt=prompt)

        response = chain(

            {"input\_documents": docs, "question": user\_question},

            return\_only\_outputs=True

        )

        # Extract the response text

        response\_text = response.get("output\_text", "Sorry, I couldn't generate a response.")

        # Remove any inline citations that might have been generated despite instructions

        response\_text = re.sub(r'\[Source:.\*?\]', '', response\_text)

        # Add sources at the end if available, with proper formatting

        if sources\_with\_pages:

            # Remove any existing Sources section that might be in the response

            response\_text = re.sub(r'\n\nSources:.\*', '', response\_text)

            # Add a clean Sources section

            unique\_sources = list(set(sources\_with\_pages))

            response\_text += "\n\nSources:"

            for source in unique\_sources:

                response\_text += f"\n- {source}"

        # Generate complete response for TTS

        if tts\_enabled:

            # Clean text for TTS (remove citations, etc.)

            tts\_text = re.sub(r'\[Source:.\*?\]', '', response\_text)

            tts\_text = re.sub(r'\n\nSources:.\*', '', tts\_text)

            audio\_data = text\_to\_speech(tts\_text)

        else:

            audio\_data = None

        # Emit the initial response with audio and full text

        emit('response\_start', {

            'full\_text': response\_text,

            'audio': audio\_data

        })

        # Stream the response word by word

        for word\_chunk in generate\_word\_stream(response\_text):

            emit('response\_chunk', word\_chunk)

            socketio.sleep(word\_chunk['delay'])

        emit('response\_end', {'status': 'complete'})

    except Exception as e:

        emit('response', {'message': f"Error: {str(e)}"})

# Flask routes

@app.route('/')

def index():

    return render\_template('index.html')

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

def upload\_files():

    """Handle PDF upload and processing."""

    if 'pdf\_docs' not in request.files:

        return {'error': 'No files uploaded'}, 400

    files = request.files.getlist('pdf\_docs')

    if not files or files[0].filename == '':

        return {'error': 'No files selected'}, 400

    try:

        # Process PDFs

        documents = get\_pdf\_text(files)

        if not documents:

            return {'error': 'No text extracted from PDFs'}, 400

        # Create chunks with metadata

        text\_chunks = get\_text\_chunks(documents)

        # Create and save vector store

        get\_vector\_store(text\_chunks)

        return {'message': f'Processed {len(files)} PDF(s) successfully!'}, 200

    except Exception as e:

        return {'error': f'Processing failed: {str(e)}'}, 500

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

    socketio.run(app, debug=True)