import streamlit as st

import google.generativeai as genai

import speech\_recognition as sr

import pyaudio

import datetime

import uuid

import io

import base64

from reportlab.lib.pagesizes import letter

from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle

from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle

from reportlab.lib import colors

from streamlit\_option\_menu import option\_menu

import os

import tempfile

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from email.mime.application import MIMEApplication

from supabase import create\_client, Client

from pydub import AudioSegment

# Supabase Configuration

supabase\_url = "https://kyuubcusqtbczcexsire.supabase.co"

supabase\_key = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Imt5dXViY3VzcXRiY3pjZXhzaXJlIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NDI5MTk0NjIsImV4cCI6MjA1ODQ5NTQ2Mn0.rntfn9r-1VzKmL0OFG2Y-\_JrjFycsuQX525UtTKGPUU"

supabase: Client = create\_client(supabase\_url, supabase\_key)

# SMTP Configuration

SMTP\_SERVER = "smtp.gmail.com"

SMTP\_PORT = 587

SMTP\_EMAIL = "madhurisabale09@gmail.com"

SMTP\_PASSWORD = "argigfabzyogwnyj"

# Initialize Gemini AI

genai.configure(api\_key="AIzaSyAWQiJDabf7L2Jh28vjg\_qBz2LNeoeXSQ4")

model = genai.GenerativeModel('gemini-1.5-flash')

# Function to generate 5 tips using Gemini based on the category

def generate\_cybercrime\_tips(category):

prompt = f"""

You are a cybersecurity expert tasked with providing practical and meaningful advice. Based on the cybercrime category '{category}', generate exactly 5 concise, actionable tips to help individuals avoid falling victim to this type of crime. Ensure the tips are specific to the category, easy to understand, and useful for the general public. Format the response as a numbered list (1-5) with no additional explanations or introductions beyond the tips themselves.

Category: {category}

"""

response = get\_gemini\_response(prompt)

return response if response else "1. Be cautious online.\n2. Use strong passwords.\n3. Avoid suspicious links.\n4. Keep software updated.\n5. Report suspicious activity."

# Function to send confirmation email with tips

def send\_confirmation\_email(to\_email, ticket\_id, category):

try:

tips = generate\_cybercrime\_tips(category)

msg = MIMEMultipart()

msg['From'] = SMTP\_EMAIL

msg['To'] = to\_email

msg['Subject'] = f"CyberGuard AI - Complaint Confirmation (Ticket ID: {ticket\_id})"

body = f"""

Dear User,

Thank you for submitting your complaint to CyberGuard AI - National Cyber Crime Reporting Portal.

Your complaint has been successfully registered with the following details:

Ticket ID: {ticket\_id}

Date Filed: {datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")}

Status: Under Investigation

Category: {category}

You can track the status of your complaint using the Ticket ID on our portal under 'Track Complaint'.

For any further assistance, please contact us at cybercrime@nic.in or call 1930.

\*5 Tips to Avoid {category} Crimes:\*

{tips}

Regards,

CyberGuard AI Team

National Cyber Crime Reporting Portal

"""

msg.attach(MIMEText(body, 'plain'))

with smtplib.SMTP(SMTP\_SERVER, SMTP\_PORT) as server:

server.starttls()

server.login(SMTP\_EMAIL, SMTP\_PASSWORD)

server.send\_message(msg)

return True

except Exception as e:

st.error(f"Failed to send confirmation email: {e}")

return False

# TTS function

def speak\_text(text, lang\_code):

try:

js\_code = f"""

<script>

var utterance = new SpeechSynthesisUtterance("{text}");

utterance.lang = "{lang\_code}";

utterance.pitch = {st.session\_state.voice\_pitch};

utterance.rate = {st.session\_state.voice\_rate};

window.speechSynthesis.speak(utterance);

</script>

"""

st.components.v1.html(js\_code, height=0)

except Exception as e:

st.error(f"Error in speech synthesis: {e}")

# Improved STT function with retries and fallback

def recognize\_speech(language\_code):

recognizer = sr.Recognizer()

retries = 3

for attempt in range(retries):

with sr.Microphone() as source:

st.info(f"🎙 Listening in {st.session\_state.selected\_language}... (Attempt {attempt + 1}/{retries})")

recognizer.adjust\_for\_ambient\_noise(source, duration=2)

try:

audio = recognizer.listen(source, timeout=5, phrase\_time\_limit=10)

with tempfile.NamedTemporaryFile(delete=False, suffix=".wav") as temp\_file:

temp\_file.write(audio.get\_wav\_data())

temp\_path = temp\_file.name

try:

text = recognizer.recognize\_google(audio, language=language\_code)

os.unlink(temp\_path)

return text

except sr.UnknownValueError:

st.warning("Google STT failed, trying Gemini transcription...")

text = get\_gemini\_response("Transcribe this audio:", temp\_path, "audio/wav")

os.unlink(temp\_path)

return text if text else None

except sr.WaitTimeoutError:

st.error("🎙 No speech detected. Please speak louder or closer to the mic.")

except sr.RequestError as e:

st.error(f"🎙 Speech recognition error: {e}")

except Exception as e:

st.error(f"Unexpected error: {e}")

if attempt < retries - 1:

st.info("Retrying...")

st.error("🎙 Failed to recognize speech after multiple attempts. Please try typing or check your microphone.")

return None

# Gemini response function with improved error handling

def get\_gemini\_response(prompt, file\_path=None, mime\_type=None):

try:

if file\_path and mime\_type:

with open(file\_path, "rb") as file:

file\_content = file.read()

response = model.generate\_content([prompt, {"mime\_type": mime\_type, "data": file\_content}])

else:

response = model.generate\_content(prompt)

return response.text.strip()

except Exception as e:

st.error(f"Gemini API error: {e}")

return None

# Translation function with caching

@st.cache\_data

def translate\_text(text, source\_lang, target\_lang):

if source\_lang == target\_lang or not text:

return text

try:

prompt = f"""

You are a precise language translator. Translate the following '{source\_lang}' text into '{target\_lang}' and return ONLY the translated text, nothing else—no explanations, no breakdowns, no notes. Use the appropriate script for the target language.

Text to translate: {text}

"""

response = get\_gemini\_response(prompt)

return response

except Exception as e:

st.error(f"Translation error: {e}")

return text

# Extract information using Gemini with improved prompting

def extract\_info\_from\_response(question, response):

prompt = f"""

You are an advanced information extraction system. Given the following question and user response, extract ONLY the relevant information as a plain string. Do not use brackets, '[Not Provided]', or any extra text—just the extracted value. If the response is incomplete or unclear, return only what can be confidently extracted; otherwise, return an empty string ('').

Question: {question}

Response: {response}

Extract:

- For "What is your full name and contact phone number?": the full name and phone number as a single string (e.g., "pruthviraj 544434")

- For "What is your email address?": the email address (e.g., "aakash@gmail.com")

- For "When did the incident occur?": the date and time (e.g., "12-03-2025 14:30")

- For "Can you describe what happened in detail?": the full description as provided

- For "Do you have any evidence...?": the evidence description as provided

- For yes/no questions: "yes" or "no" (lowercase); if unclear, return an empty string ('')

"""

result = get\_gemini\_response(prompt)

return result if result else ""

# Initialize session state

def init\_session\_state():

defaults = {

'chat\_history': [],

'form\_data': {},

'form\_data\_translated': {},

'complaint\_tickets': {},

'questions\_index': 0,

'chatbot\_active': False,

'speech\_input': '',

'last\_spoken\_index': -1,

'voice\_enabled': True,

'voice\_pitch': 1.0,

'voice\_rate': 1.0,

'selected\_language': "English",

'authenticated': False,

'current\_page': 'signin'

}

for key, value in defaults.items():

if key not in st.session\_state:

st.session\_state[key] = value

init\_session\_state()

# Language mappings

languages = {

"Hindi": "hi-IN", "Konkani": "kok-IN", "Kannada": "kn-IN", "Dogri": "doi-IN",

"Bodo": "brx-IN", "Urdu": "ur-IN", "Tamil": "ta-IN", "Kashmiri": "ks-IN",

"Assamese": "as-IN", "Bengali": "bn-IN", "Marathi": "mr-IN", "Sindhi": "sd-IN",

"Maithili": "mai-IN", "Punjabi": "pa-IN", "Malayalam": "ml-IN", "Manipuri": "mni-IN",

"Telugu": "te-IN", "Sanskrit": "sa-IN", "Nepali": "ne-IN", "Santali": "sat-IN",

"Gujarati": "gu-IN", "Odia": "or-IN", "English": "en-IN"

}

tts\_lang\_codes = {k: v.split('-')[0] + '-' + v.split('-')[1].upper() for k, v in languages.items()}

native\_commands = {

"English": {"next": "next", "back": "back", "submit": "submit", "repeat": "repeat"},

"Hindi": {"next": "अगला", "back": "पीछे", "submit": "जमा करें", "repeat": "दोहराएं"},

"Tamil": {"next": "அடுத்து", "back": "பின்னால்", "submit": "சமர்ப்பி", "repeat": "மீண்டும்"},

"Telugu": {"next": "తదుపరి", "back": "వెనక్కి", "submit": "సమర్పించు", "repeat": "పునరావృతం"},

"Kannada": {"next": "ಮುಂದಿನ", "back": "ಹಿಂದೆ", "submit": "ಸಲ್ಲಿಸು", "repeat": "ಪುನರಾವರ್ತನೆ"},

"Malayalam": {"next": "അടുത്തത്", "back": "പിന്നോട്ട്", "submit": "സമർപ്പിക്കുക", "repeat": "ആവർത്തിക്കുക"},

"Marathi": {"next": "पुढील", "back": "मागे", "submit": "सादर करा", "repeat": "पुनरावृत्ती"},

"Bengali": {"next": "পরবর্তী", "back": "পিছনে", "submit": "জমা দিন", "repeat": "পুনরাবৃত্তি"},

"Gujarati": {"next": "આગળ", "back": "પાછળ", "submit": "સબમિટ કરો", "repeat": "પુનરાવર્તન"},

"Punjabi": {"next": "ਅਗਲਾ", "back": "ਪਿੱਛੇ", "submit": "ਜਮ੍ਹਾ ਕਰੋ", "repeat": "ਦੁਹਰਾਓ"},

"Dogri": {"next": "अगला", "back": "पिछे", "submit": "जमा करो", "repeat": "दुहराओ"},

}

# Updated Complaint Questionnaire

form\_filling\_questions = [

{"field": "name\_phone", "question": {"English": "What is your full name and contact phone number?"}, "required": True},

{"field": "email", "question": {"English": "What is your email address?"}, "required": True},

{"field": "incident\_date", "question": {"English": "When did the incident occur? (Please provide the date and approximate time.)"}, "required": True},

{"field": "threat\_harass\_women\_children", "question": {"English": "Have you or someone you know received threatening or harassing messages online specifically targeting women or children? Please answer with yes or no."}, "required": False, "type": "yes\_no"},

{"field": "financial\_scam", "question": {"English": "Have you experienced unauthorized financial transactions, phishing scams, or deceptive financial offers via online communications? Please answer with yes or no."}, "required": False, "type": "yes\_no"},

{"field": "malware\_ransomware", "question": {"English": "Has your computer system or network been compromised by malware, ransomware, or unauthorized access? Please answer with yes or no."}, "required": False, "type": "yes\_no"},

{"field": "illegal\_trafficking", "question": {"English": "Have you encountered online platforms or content facilitating illegal trafficking or sale of goods/services? Please answer with yes or no."}, "required": False, "type": "yes\_no"},

{"field": "incident\_description", "question": {"English": "Can you describe what happened in detail?"}, "required": True},

{"field": "evidence", "question": {"English": "Do you have any evidence such as screenshots, emails, or messages that support your report? Please describe and upload if available."}, "required": False, "type": "text\_and\_upload"},

]

# UI Config

st.set\_page\_config(

page\_title="CyberGuard AI - National Cyber Crime Reporting Portal",

page\_icon="🛡",

layout="wide",

initial\_sidebar\_state="expanded"

)

# Custom CSS (unchanged)

st.markdown(

"""

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">

<style>

.stApp {

background-color: #f5f7fa;

margin: 0 !important;

padding: 0 !important;

}

.header-home {

background: linear-gradient(90deg, #0047AB 0%, #184C78 100%);

color: white;

padding: 1rem;

text-align: center;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

margin: 0 0 20px 0 !important;

font-size: 0.9em;

}

.content-card {

background-color: white;

padding: 20px;

border-radius: 10px;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);

margin: 0 0 20px 0 !important;

}

.dashboard-card {

background-color: white;

padding: 15px;

border-radius: 10px;

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);

text-align: center;

margin: 5px;

}

.stButton>button {

background-color: #0047AB;

color: white;

border: none;

border-radius: 5px;

padding: 0.5rem 1rem;

font-weight: 500;

}

.stButton>button:hover {

background-color: #003087;

border: none;

}

.stTextInput>div>div>input, .stTextArea>div>div>textarea {

border-radius: 5px;

border: 1px solid #E0E0E0;

}

.status-badge {

padding: 5px 10px;

border-radius: 15px;

font-weight: 500;

display: inline-block;

}

.status-pending {

background-color: #FFF9C4;

color: #F57F17;

}

.status-active {

background-color: #E3F2FD;

color: #0D47A1;

}

.status-resolved {

background-color: #E8F5E9;

color: #1B5E20;

}

.chat-message {

padding: 1rem;

border-radius: 15px;

margin-bottom: 10px;

max-width: 80%;

display: inline-block;

}

.user-message {

background-color: #E3F2FD;

float: right;

clear: both;

border-bottom-right-radius: 5px;

}

.bot-message {

background-color: #F5F5F5;

float: left;

clear: both;

border-bottom-left-radius: 5px;

}

.chat-container {

height: calc(80vh - 200px);

overflow-y: auto;

padding: 20px;

display: flex;

flex-direction: column;

}

.helpline-badge {

background-color: #E91E63;

color: white;

padding: 5px 10px;

border-radius: 30px;

font-weight: bold;

margin: 5px;

display: inline-block;

}

.footer {

text-align: center;

padding: 20px;

color: #666;

font-size: 0.8rem;

border-top: 1px solid #eee;

margin-top: 30px;

}

.stat-counter {

font-size: 2rem;

font-weight: bold;

color: #0047AB;

}

.big-icon {

font-size: 100px;

color: #0047AB;

text-align: center;

margin-bottom: 20px;

}

.auth-container {

max-width: 400px;

margin: 50px auto;

text-align: center;

}

.auth-icon {

font-size: 60px;

color: #0047AB;

margin-bottom: 20px;

}

.auth-button {

width: 100%;

margin-top: 10px;

}

@media (max-width: 768px) {

.chat-message {

max-width: 90%;

}

}

#MainMenu {visibility: hidden;}

footer {visibility: hidden;}

header {visibility: hidden;}

.progress-bar {

background-color: #E0E0E0;

border-radius: 5px;

height: 20px;

}

.progress-fill {

background-color: #0047AB;

height: 100%;

border-radius: 5px;

}

div[data-testid="stFileUploader"] {

width: 100%;

max-width: 500px;

margin-top: 10px;

}

div[data-testid="stFileUploader"] > div > div {

width: 100%;

}

</style>

""",

unsafe\_allow\_html=True

)

# Authentication Functions

def register\_user(username, password, email):

response = supabase.table('users').select('username').eq('username', username).execute()

if response.data:

return False, "Username already exists."

try:

supabase.table('users').insert({

'username': username,

'password': password, # In production, hash the password

'email': email

}).execute()

return True, "Registered successfully! Please sign in."

except Exception as e:

return False, f"Registration failed: {str(e)}"

def sign\_in\_user(username, password):

response = supabase.table('users').select('\*').eq('username', username).eq('password', password).execute()

if response.data:

st.session\_state.authenticated = True

st.session\_state.current\_page = 'dashboard'

return True, "Signed in successfully!"

return False, "Invalid credentials."

# Sign In Page

def sign\_in\_page():

st.markdown(

"""

<div class="auth-container">

<i class="fas fa-sign-in-alt auth-icon"></i>

<h2>Sign In</h2>

</div>

""",

unsafe\_allow\_html=True

)

with st.form(key='signin\_form', clear\_on\_submit=True):

username = st.text\_input("Username", key="signin\_username")

password = st.text\_input("Password", type="password", key="signin\_password")

submit\_button = st.form\_submit\_button(label="Sign In")

if submit\_button:

success, message = sign\_in\_user(username, password)

if success:

st.success(message)

st.rerun()

else:

st.error(message)

st.markdown(

"""

<div class="auth-container">

<p>Not registered yet? <a href="#" id="register\_link">Register here</a></p>

</div>

""",

unsafe\_allow\_html=True

)

if st.button("Go to Register", key="to\_register"):

st.session\_state.current\_page = 'register'

st.rerun()

# Register Page

def register\_page():

st.markdown(

"""

<div class="auth-container">

<i class="fas fa-user-plus auth-icon"></i>

<h2>Register</h2>

</div>

""",

unsafe\_allow\_html=True

)

with st.form(key='register\_form', clear\_on\_submit=True):

username = st.text\_input("Username", key="register\_username")

password = st.text\_input("Password", type="password", key="register\_password")

email = st.text\_input("Email", key="register\_email")

submit\_button = st.form\_submit\_button(label="Register")

if submit\_button:

success, message = register\_user(username, password, email)

if success:

st.success(message)

st.session\_state.current\_page = 'signin'

st.rerun()

else:

st.error(message)

st.markdown(

"""

<div class="auth-container">

<p>Already registered? <a href="#" id="signin\_link">Sign in here</a></p>

</div>

""",

unsafe\_allow\_html=True

)

if st.button("Go to Sign In", key="to\_signin"):

st.session\_state.current\_page = 'signin'

st.rerun()

# Supabase Integration for Complaint Storage

def save\_to\_supabase(data, translated\_data):

ticket\_id = f"CYBER-{uuid.uuid4().hex[:8].upper()}"

complaint\_data = {

"ticket\_id": ticket\_id,

"data": data,

"translated\_data": translated\_data,

"status": "Under Investigation",

"date\_filed": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

"last\_updated": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

}

try:

supabase.table('complaints').insert(complaint\_data).execute()

email = translated\_data.get('email', '')

category = translated\_data.get('category', 'Other')

if email and send\_confirmation\_email(email, ticket\_id, category):

st.success(f"✅ Confirmation email with safety tips sent to {email}!")

else:

st.warning("Complaint saved, but email confirmation failed.")

return ticket\_id

except Exception as e:

st.error(f"Failed to save to Supabase: {e}")

return None

def fetch\_complaint\_from\_supabase(ticket\_id):

try:

response = supabase.table('complaints').select('\*').eq('ticket\_id', ticket\_id).execute()

if response.data:

return response.data[0]

return None

except Exception as e:

st.error(f"Failed to fetch from Supabase: {e}")

return None

# Generate PDF

def generate\_complaint\_pdf(data):

buffer = io.BytesIO()

doc = SimpleDocTemplate(buffer, pagesize=letter)

styles = getSampleStyleSheet()

story = []

title\_style = ParagraphStyle('Title', parent=styles['Heading1'], alignment=1, spaceAfter=12)

story.append(Paragraph("CYBER CRIME COMPLAINT REPORT", title\_style))

story.append(Spacer(1, 12))

complaint\_data = [

["Ticket Number", data.get('ticket\_id', '')],

["Date Filed", data.get('date\_filed', '')],

["Name and Phone", data.get('name\_phone', '')],

["Email Address", data.get('email', '')],

["Incident Date", data.get('incident\_date', '')],

["Threat/Harassment to Women/Children", data.get('threat\_harass\_women\_children', 'N/A')],

["Financial Scam", data.get('financial\_scam', 'N/A')],

["Malware/Ransomware", data.get('malware\_ransomware', 'N/A')],

["Illegal Trafficking", data.get('illegal\_trafficking', 'N/A')],

["Incident Description", data.get('incident\_description', '')],

["Evidence Description", data.get('evidence', '')],

["Category", data.get('category', 'N/A')],

["Category Explanation", data.get('category\_explanation', 'N/A')],

["Status", data.get('status', '')],

]

table = Table(complaint\_data, colWidths=[150, 400])

table.setStyle(TableStyle([

('BACKGROUND', (0, 0), (-1, 0), colors.grey),

('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),

('ALIGN', (0, 0), (-1, -1), 'LEFT'),

('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),

('FONTSIZE', (0, 0), (-1, 0), 14),

('BOTTOMPADDING', (0, 0), (-1, 0), 12),

('BACKGROUND', (0, 1), (-1, -1), colors.white),

('TEXTCOLOR', (0, 1), (-1, -1), colors.black),

('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),

('FONTSIZE', (0, 1), (-1, -1), 12),

('GRID', (0, 0), (-1, -1), 1, colors.black),

('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),

]))

story.append(table)

story.append(Spacer(1, 12))

footer\_style = ParagraphStyle('Footer', parent=styles['Normal'], fontSize=10, textColor=colors.grey, alignment=1)

story.append(Paragraph("This is an auto-generated document from CyberGuard AI Portal", footer\_style))

doc.build(story)

buffer.seek(0)

return buffer

# Get question text with caching

@st.cache\_data

def get\_question\_text(question\_dict, lang):

if lang in question\_dict:

return question\_dict[lang]

english\_text = question\_dict.get("English", "")

translated\_text = translate\_text(english\_text, "English", lang)

question\_dict[lang] = translated\_text

return translated\_text

# Analyze Image with Gemini

def analyze\_image(image\_path):

prompt = "Analyze this image and describe its content relevant to a cybercrime complaint."

analysis = get\_gemini\_response(prompt, image\_path, "image/jpeg")

return analysis if analysis else "No significant content detected in the image."

# Check if audio has sound

def has\_sound(audio\_path):

try:

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

return audio.dBFS > -60 # Threshold for detecting sound

except Exception as e:

st.error(f"Audio analysis error: {e}")

return False

# Enhanced Categorization with Advanced Prompting

def categorize\_complaint(data):

complaint\_text = "\n".join([f"{k}: {v}" for k, v in data.items() if k not in ['evidence\_files', 'category', 'category\_explanation']])

image\_analyses = []

if 'evidence\_files' in data:

for evidence in data['evidence\_files']:

if evidence['name'].endswith(('.jpg', '.jpeg', '.png')):

with tempfile.NamedTemporaryFile(delete=False, suffix=".jpg") as temp\_file:

temp\_file.write(base64.b64decode(evidence['content']))

temp\_path = temp\_file.name

analysis = analyze\_image(temp\_path)

image\_analyses.append(f"Image {evidence['name']}: {analysis}")

os.unlink(temp\_path)

prompt = f"""

You are an expert cybercrime analyst with advanced knowledge in digital forensics, behavioral analysis, and legal frameworks. Your task is to categorize the following complaint into one of these categories:

- Cyber Harassment

- Financial Fraud

- System Security

- Illegal Activities

- Other

Provide a precise and detailed explanation (6-10 sentences) justifying the chosen category, adhering to these enhanced steps:

1. Analyze each Yes/No response, assigning weighted relevance to potential categories based on severity and specificity.

2. Perform a semantic analysis of the incident description, identifying key phrases, intent, and contextual clues with high accuracy.

3. Integrate image analysis (if available) as corroborative evidence, assessing its relevance to the complaint narrative.

4. Cross-reference the combined data against known cybercrime patterns and typologies for consistency.

5. Resolve ambiguities by prioritizing the most specific and impactful evidence, avoiding generic assumptions.

6. Conclude with a clear, evidence-based rationale for the selected category, ensuring alignment with legal definitions.

Respond in this format:

Category: [category name]

Explanation: [detailed explanation]

Complaint details:

{complaint\_text}

Image Analysis (if any):

{"; ".join(image\_analyses) if image\_analyses else "No images provided."}

"""

response = get\_gemini\_response(prompt)

try:

category\_line, explanation\_line = response.split('\n', 1)

category = category\_line.split(": ")[1]

explanation = explanation\_line.split(": ")[1]

valid\_categories = ["Cyber Harassment", "Financial Fraud", "System Security", "Illegal Activities", "Other"]

return category if category in valid\_categories else "Other", explanation

except Exception:

return "Other", "Failed to categorize due to an error in processing the response. Please ensure all details are complete and retry."

# Process chatbot input with clean data storage

def process\_chatbot\_input(user\_input, audio\_file, current\_question):

lang = st.session\_state.selected\_language

commands = native\_commands.get(lang, native\_commands["English"])

question\_text = get\_question\_text(current\_question['question'], "English")

if audio\_file:

with tempfile.NamedTemporaryFile(delete=False, suffix=".wav") as temp\_file:

temp\_file.write(audio\_file.read())

temp\_path = temp\_file.name

if has\_sound(temp\_path):

user\_input = get\_gemini\_response("Transcribe this audio:", temp\_path, "audio/wav")

if not user\_input:

user\_input = "No recognizable speech in audio."

else:

os.unlink(temp\_path)

st.warning("Uploaded audio has no sound. Please upload a valid audio file.")

return "No sound detected in audio."

os.unlink(temp\_path)

st.session\_state.speech\_input = user\_input

if user\_input.lower() in [commands["next"], "next"]:

st.session\_state.questions\_index += 1

return None

elif user\_input.lower() in [commands["back"], "back"]:

st.session\_state.questions\_index = max(0, st.session\_state.questions\_index - 1)

return None

elif user\_input.lower() in [commands["submit"], "submit"]:

st.session\_state.chatbot\_active = False

category, explanation = categorize\_complaint(st.session\_state.form\_data\_translated)

st.session\_state.form\_data['category'] = category

st.session\_state.form\_data['category\_explanation'] = explanation

st.session\_state.form\_data\_translated['category'] = category

st.session\_state.form\_data\_translated['category\_explanation'] = explanation

return f"All details collected. Complaint categorized as: {category}. Explanation: {explanation}"

elif user\_input.lower() in [commands["repeat"], "repeat"]:

return get\_question\_text(current\_question['question'], lang)

extracted\_value = extract\_info\_from\_response(question\_text, user\_input)

if current\_question.get('type') == "yes\_no":

if extracted\_value in ["yes", "no"]:

st.session\_state.form\_data[current\_question['field']] = extracted\_value

st.session\_state.form\_data\_translated[current\_question['field']] = extracted\_value

st.session\_state.questions\_index += 1

else:

return translate\_text("Please respond with 'yes' or 'no'.", "English", lang)

elif current\_question.get('type') == "text\_and\_upload":

st.session\_state.form\_data[current\_question['field']] = extracted\_value

st.session\_state.form\_data\_translated[current\_question['field']] = translate\_text(extracted\_value, lang, "English")

uploaded\_files = st.file\_uploader("Upload evidence here", accept\_multiple\_files=True, key=f"upload\_{st.session\_state.questions\_index}")

if uploaded\_files:

st.session\_state.form\_data['evidence\_files'] = [{"name": f.name, "content": base64.b64encode(f.read()).decode('utf-8')} for f in uploaded\_files]

st.session\_state.questions\_index += 1

else:

st.session\_state.form\_data[current\_question['field']] = extracted\_value

st.session\_state.form\_data\_translated[current\_question['field']] = translate\_text(extracted\_value, lang, "English")

st.session\_state.questions\_index += 1

if st.session\_state.questions\_index >= len(form\_filling\_questions):

st.session\_state.chatbot\_active = False

category, explanation = categorize\_complaint(st.session\_state.form\_data\_translated)

st.session\_state.form\_data['category'] = category

st.session\_state.form\_data['category\_explanation'] = explanation

st.session\_state.form\_data\_translated['category'] = category

st.session\_state.form\_data\_translated['category\_explanation'] = explanation

return f"Thank you for providing all the information. Complaint categorized as: {category}. Explanation: {explanation}"

return None

# Display chat message

def display\_chat\_message(message, is\_user=False):

message\_class = "user-message" if is\_user else "bot-message"

st.markdown(f'<div class="chat-message {message\_class}">{message}</div>', unsafe\_allow\_html=True)

# Main Dashboard

def dashboard():

with st.sidebar:

selected = option\_menu(

"Main Menu",

["Home", "Register Complaint", "Track Complaint", "Contact Us"],

icons=['house', 'file-alt', 'search', 'envelope'],

menu\_icon="shield-lock",

default\_index=0,

styles={

"container": {"padding": "5px", "background-color": "#f5f7fa"},

"icon": {"color": "#0047AB", "font-size": "25px"},

"nav-link": {"font-size": "16px", "text-align": "left", "margin": "0px", "--hover-color": "#eee"},

"nav-link-selected": {"background-color": "#0047AB"},

}

)

if st.button("Sign Out", key="signout"):

st.session\_state.authenticated = False

st.session\_state.current\_page = 'signin'

st.rerun()

language\_code = languages.get(st.session\_state.selected\_language, languages["English"])

tts\_lang = tts\_lang\_codes.get(st.session\_state.selected\_language, tts\_lang\_codes["English"])

if selected == "Home":

st.markdown(

"""

<div class="header-home">

<h4>CyberGuard AI - National Cyber Crime Reporting Portal</h4>

<p>File and Track Cyber Crime Complaints with Ease | Powered by AI</p>

<div class="helpline-badge">National Cyber Crime Helpline: 1930</div>

</div>

""",

unsafe\_allow\_html=True

)

st.markdown('<i class="fas fa-shield-alt big-icon"></i>', unsafe\_allow\_html=True)

st.markdown(

"""

<div class="content-card">

<h2>Welcome to CyberGuard AI</h2>

<p>India's premier portal for reporting and tracking cyber crimes, powered by Artificial Intelligence. Our mission is to provide a secure, efficient, and user-friendly platform to combat cyber threats.</p>

</div>

""",

unsafe\_allow\_html=True

)

col1, col2, col3 = st.columns(3)

with col1:

st.markdown(

"""

<div class="dashboard-card">

<h4>Total Complaints</h4>

<div class="stat-counter">{}</div>

</div>

""".format(len(st.session\_state.complaint\_tickets)),

unsafe\_allow\_html=True

)

with col2:

st.markdown(

"""

<div class="dashboard-card">

<h4>Resolved Cases</h4>

<div class="stat-counter">{}</div>

</div>

""".format(sum(1 for ticket in st.session\_state.complaint\_tickets.values() if ticket.get('status') == "Resolved")),

unsafe\_allow\_html=True

)

with col3:

st.markdown(

"""

<div class="dashboard-card">

<h3>Active Cases</h3>

<div class="stat-counter">{}</div>

</div>

""".format(sum(1 for ticket in st.session\_state.complaint\_tickets.values() if ticket.get('status') == "Under Investigation")),

unsafe\_allow\_html=True

)

elif selected == "Register Complaint":

st.session\_state.selected\_language = st.selectbox(

"Select Language / भाषा चुनें",

list(languages.keys()),

index=list(languages.keys()).index("English"),

key="language\_selector"

)

language\_code = languages[st.session\_state.selected\_language]

tts\_lang = tts\_lang\_codes[st.session\_state.selected\_language]

st.markdown('<i class="fas fa-file-alt big-icon"></i>', unsafe\_allow\_html=True)

st.markdown(

"""

<div class="content-card">

<h4>File a New Complaint</h4>

<p>Please provide the necessary details to register your complaint. You can use the AI chatbot to fill the form automatically or fill it manually.</p>

</div>

""",

unsafe\_allow\_html=True

)

with st.expander("Voice Settings"):

st.session\_state.voice\_enabled = st.checkbox("Enable Voice", value=True)

st.session\_state.voice\_pitch = st.slider("Pitch", 0.5, 2.0, 1.0)

st.session\_state.voice\_rate = st.slider("Speed", 0.5, 2.0, 1.0)

use\_chatbot = st.checkbox("Use AI Chatbot to Fill Form", value=False)

if use\_chatbot:

st.session\_state.chatbot\_active = True

st.markdown('<div class="chat-container">', unsafe\_allow\_html=True)

progress = st.session\_state.questions\_index / len(form\_filling\_questions) \* 100

st.markdown(f'<div class="progress-bar"><div class="progress-fill" style="width:{progress}%"></div></div>', unsafe\_allow\_html=True)

for chat in st.session\_state.chat\_history:

display\_chat\_message(chat['message'], chat['is\_user'])

if st.session\_state.chatbot\_active and st.session\_state.questions\_index < len(form\_filling\_questions):

current\_question = form\_filling\_questions[st.session\_state.questions\_index]

with st.spinner("Loading question..."):

q\_text = get\_question\_text(current\_question['question'], st.session\_state.selected\_language)

if st.session\_state.voice\_enabled and st.session\_state.questions\_index > st.session\_state.last\_spoken\_index:

st.info(f"🔊 Speaking in {st.session\_state.selected\_language}...")

speak\_text(q\_text, tts\_lang)

st.session\_state.last\_spoken\_index = st.session\_state.questions\_index

display\_chat\_message(q\_text)

col1, col2 = st.columns([4, 1])

with col1:

user\_input = st.text\_input(

"Your response",

value=st.session\_state.speech\_input,

key=f"chat\_input\_{st.session\_state.questions\_index}"

)

with col2:

if st.button("🎙", key=f"mic\_{st.session\_state.questions\_index}"):

with st.spinner("Processing speech..."):

speech\_text = recognize\_speech(language\_code)

if speech\_text:

st.session\_state.speech\_input = speech\_text

st.rerun()

audio\_file = st.file\_uploader(

"Upload Audio Response",

type=["wav", "mp3"],

key=f"audio\_{st.session\_state.questions\_index}",

help="Upload an audio file as your response (max 5MB)",

label\_visibility="visible"

)

if user\_input or audio\_file:

with st.spinner("Processing your response..."):

final\_input = user\_input if user\_input else ""

if audio\_file:

final\_input = process\_chatbot\_input("", audio\_file, current\_question) or final\_input

if final\_input:

display\_chat\_message(final\_input, is\_user=True)

st.session\_state.chat\_history.append({"message": q\_text, "is\_user": False})

st.session\_state.chat\_history.append({"message": final\_input, "is\_user": True})

response = process\_chatbot\_input(final\_input, None, current\_question)

if response:

display\_chat\_message(response)

st.session\_state.speech\_input = ""

st.rerun()

st.markdown('</div>', unsafe\_allow\_html=True)

if not st.session\_state.chatbot\_active:

st.success("✅ Complaint data collected successfully.")

# Removed Collected Details display

st.markdown(f"\*Category Assigned:\* {st.session\_state.form\_data.get('category', 'N/A')}")

st.markdown(f"\*Explanation:\* {st.session\_state.form\_data.get('category\_explanation', 'N/A')}")

if st.button("Submit Complaint", key="chatbot\_submit"):

with st.spinner("Submitting your complaint..."):

ticket\_id = save\_to\_supabase(st.session\_state.form\_data, st.session\_state.form\_data\_translated)

if ticket\_id:

st.session\_state.form\_data['ticket\_id'] = ticket\_id

st.session\_state.form\_data\_translated['ticket\_id'] = ticket\_id

st.session\_state.form\_data\_translated.update({

"status": "Under Investigation",

"date\_filed": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

})

pdf\_buffer = generate\_complaint\_pdf(st.session\_state.form\_data\_translated)

st.success(f"✅ Complaint filed successfully! Your ticket ID is: {ticket\_id}")

st.download\_button(

label="Download Complaint PDF",

data=pdf\_buffer,

file\_name=f"Complaint\_{ticket\_id}.pdf",

mime="application/pdf"

)

st.session\_state.form\_data = {}

st.session\_state.form\_data\_translated = {}

st.session\_state.chat\_history = []

st.session\_state.questions\_index = 0

if not use\_chatbot or not st.session\_state.chatbot\_active:

with st.form(key='complaint\_form'):

col1, col2 = st.columns(2)

with col1:

name\_phone = st.text\_input("Full Name and Phone Number", value=st.session\_state.form\_data.get('name\_phone', ''))

email = st.text\_input("Email Address", value=st.session\_state.form\_data.get('email', ''))

incident\_date = st.text\_input("Incident Date & Time (e.g., DD-MM-YYYY HH:MM)", value=st.session\_state.form\_data.get('incident\_date', ''))

threat\_harass = st.selectbox("Threat/Harassment to Women/Children", ["No", "Yes"],

index=0 if st.session\_state.form\_data.get('threat\_harass\_women\_children', 'no') == 'no' else 1)

financial\_scam = st.selectbox("Financial Scam", ["No", "Yes"],

index=0 if st.session\_state.form\_data.get('financial\_scam', 'no') == 'no' else 1)

with col2:

malware = st.selectbox("Malware/Ransomware", ["No", "Yes"],

index=0 if st.session\_state.form\_data.get('malware\_ransomware', 'no') == 'no' else 1)

illegal\_trafficking = st.selectbox("Illegal Trafficking", ["No", "Yes"],

index=0 if st.session\_state.form\_data.get('illegal\_trafficking', 'no') == 'no' else 1)

incident\_description = st.text\_area("Incident Description", value=st.session\_state.form\_data.get('incident\_description', ''), height=200)

evidence = st.text\_area("Evidence Description (e.g., screenshots, emails, messages)", value=st.session\_state.form\_data.get('evidence', ''))

evidence\_files = st.file\_uploader("Upload Evidence (Screenshots, Documents, etc.)", accept\_multiple\_files=True, type=['jpg', 'png', 'pdf', 'docx'])

submit\_button = st.form\_submit\_button(label='Submit Complaint')

if submit\_button:

with st.spinner("Submitting your complaint..."):

required\_fields = {q['field']: q['question']['English'] for q in form\_filling\_questions if q['required']}

form\_values = {

"name\_phone": name\_phone,

"email": email,

"incident\_date": incident\_date,

"incident\_description": incident\_description

}

missing\_fields = [field for field, value in form\_values.items() if field in required\_fields and not value]

if missing\_fields:

st.error(f"Please fill in all required fields: {', '.join(required\_fields[field] for field in missing\_fields)}")

else:

complaint\_data = {

"name\_phone": name\_phone,

"email": email,

"incident\_date": incident\_date,

"threat\_harass\_women\_children": threat\_harass.lower(),

"financial\_scam": financial\_scam.lower(),

"malware\_ransomware": malware.lower(),

"illegal\_trafficking": illegal\_trafficking.lower(),

"incident\_description": incident\_description,

"evidence": evidence,

"evidence\_files": []

}

if evidence\_files:

complaint\_data["evidence\_files"] = [{"name": f.name, "content": base64.b64encode(f.read()).decode('utf-8')} for f in evidence\_files]

category, explanation = categorize\_complaint(complaint\_data)

complaint\_data["category"] = category

complaint\_data["category\_explanation"] = explanation

translated\_data = {k: translate\_text(v, st.session\_state.selected\_language, "English") for k, v in complaint\_data.items()}

ticket\_id = save\_to\_supabase(complaint\_data, translated\_data)

if ticket\_id:

translated\_data["ticket\_id"] = ticket\_id

translated\_data.update({

"status": "Under Investigation",

"date\_filed": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

})

pdf\_buffer = generate\_complaint\_pdf(translated\_data)

st.success(f"✅ Complaint filed successfully! Your ticket ID is: {ticket\_id}")

# Removed Collected Details display

st.markdown(f"\*Category Assigned:\* {category}")

st.markdown(f"\*Explanation:\* {explanation}")

st.download\_button(

label="Download Complaint PDF",

data=pdf\_buffer,

file\_name=f"Complaint\_{ticket\_id}.pdf",

mime="application/pdf"

)

st.session\_state.form\_data = {}

st.session\_state.form\_data\_translated = {}

st.session\_state.chat\_history = []

st.session\_state.questions\_index = 0

elif selected == "Track Complaint":

st.markdown('<i class="fas fa-search big-icon"></i>', unsafe\_allow\_html=True)

st.markdown(

"""

<div class="content-card">

<h2>Track Your Complaint</h2>

<p>Enter your ticket ID to check the status of your complaint.</p>

</div>

""",

unsafe\_allow\_html=True

)

ticket\_id = st.text\_input("Enter Ticket ID (e.g., CYBER-XXXXXXXX)", "").upper()

if ticket\_id:

ticket\_data = fetch\_complaint\_from\_supabase(ticket\_id)

if ticket\_data:

status\_class = "status-pending" if ticket\_data['status'] == "Under Investigation" else "status-resolved"

st.markdown(

f"""

<div class="content-card">

<h3>Complaint Status</h3>

<p><strong>Ticket ID:</strong> {ticket\_id}</p>

<p><strong>Status:</strong> <span class="status-badge {status\_class}">{ticket\_data['status']}</span></p>

<p><strong>Date Filed:</strong> {ticket\_data['date\_filed']}</p>

<p><strong>Last Updated:</strong> {ticket\_data['last\_updated']}</p>

<p><strong>Category:</strong> {ticket\_data['translated\_data'].get('category', 'N/A')}</p>

<p><strong>Category Explanation:</strong> {ticket\_data['translated\_data'].get('category\_explanation', 'N/A')}</p>

</div>

""",

unsafe\_allow\_html=True

)

pdf\_buffer = generate\_complaint\_pdf(ticket\_data['translated\_data'])

st.download\_button(

label="Download Complaint PDF",

data=pdf\_buffer,

file\_name=f"Complaint\_{ticket\_id}.pdf",

mime="application/pdf"

)

else:

st.error("❌ Invalid Ticket ID. Please check and try again.")

elif selected == "Contact Us":

st.markdown('<i class="fas fa-envelope big-icon"></i>', unsafe\_allow\_html=True)

st.markdown(

"""

<div class="content-card">

<h2>Contact Us</h2>

<p>For immediate assistance, please use the following contact information:</p>

<div class="helpline-badge">National Cyber Crime Helpline: 1930</div>

<p><strong>Email:</strong> cybercrime@nic.in</p>

<p><strong>Website:</strong> <a href="https://cybercrime.gov.in" target="\_blank">cybercrime.gov.in</a></p>

<p><strong>Address:</strong> Ministry of Home Affairs, Cyber Crime Wing, North Block, New Delhi - 110001</p>

</div>

""",

unsafe\_allow\_html=True

)

st.markdown(

"""

<div class="footer">

<p>© 2023 CyberGuard AI | All Rights Reserved | Powered by Streamlit & Google Gemini AI</p>

<p>Disclaimer: This is a demo application. For actual complaints, please visit the official portal at cybercrime.gov.in</p>

</div>

""",

unsafe\_allow\_html=True

)

# Main App Logic

if not st.session\_state.authenticated:

if st.session\_state.current\_page == 'signin':

sign\_in\_page()

elif st.session\_state.current\_page == 'register':

register\_page()

else:

dashboard()