```
import os
import datetime
import csv
import re
import random
from time import sleep

from flask import Flask, rendertemplate, request, redirect, urlfor, sendfromdirectory, flash,
jsonify
import text2emotion as te
import numpy as np
import logging
from logging.handlers import RotatingFileHandler
```

Configure logging

```
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(name)
handler = RotatingFileHandler('app.log', maxBytes=10000, backupCount=3)
handler.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)

try:
import librosa

HAS_LIBROSA = True
except Exception as e:
logger.warning(f"Librosa not available: {e}")
HAS_LIBROSA = False

try:
import emoji

EMOJIMAP = getattr(emoji, "EMOJIDATA",None) or getattr(emoji, "UNICODE_EMOJI", None)
except Exception as e:
logger.warning(f"Emoji module not available: {e}")
emoji = None
EMOJI_MAP = None

BASEDIR = os.path.abspath(os.path.dirname(file_))
UPLOADFOLDER = os.path.join(BASEDIR, "uploads")
```

LOG*FILE = os.path.join(BASE*DIR, "logs", "predictions.csv")
os.makedirs(UPLOAD*FOLDER, exist*ok=True)
os.makedirs(os.path.dirname(LOG*FILE), exist*ok=True)

app = Flask(name)
app.config["UPLOAD*FOLDER"] = UPLOAD*FOLDER
app.config["MAX*CONTENT*LENGTH"] = 12 *1024* 1024
app.secret_key = 'dev-secret-key'

Enhanced emotion lexicon for detailed analysis

ENHANCED*EMOTION*LEXICON = {
# Sadness indicators
"sad": "sadness", "sadness": "sadness", "unhappy": "sadness", "depressed": "sadness",
"miserable": "sadness", "heartbroken": "sadness", "grief": "sadness", "sorrow": "sadness",
"tears": "sadness", "crying": "sadness", "cry": "sadness", "weep": "sadness",
"hopeless": "sadness", "despair": "sadness", "melancholy": "sadness", "blue": "sadness",
"down": "sadness", "low": "sadness", "hurt": "sadness", "pain": "sadness",

# Loneliness indicators
"lonely": "loneliness", "loneliness": "loneliness", "alone": "loneliness",
"isolated": "loneliness", "isolation": "loneliness", "abandoned": "loneliness",
"solitude": "loneliness", "disconnected": "loneliness", "unwanted": "loneliness",
"ignored": "loneliness", "invisible": "loneliness", "forgotten": "loneliness",

# Disappointment indicators
"disappointed": "disappointment", "disappointment": "disappointment",
"let down": "disappointment", "failed": "disappointment", "failure": "disappointment",
"unmet": "disappointment", "overlooked": "disappointment", "ignored": "disappointment",
"unrecognized": "disappointment", "unappreciated": "disappointment",

# Frustration indicators
"frustrated": "frustration", "frustration": "frustration", "hopeless": "frustration",
"helpless": "frustration", "stuck": "frustration", "trapped": "frustration",
"powerless": "frustration", "defeated": "frustration",

# Fear indicators
"fear": "fear", "afraid": "fear", "scared": "fear", "frightened": "fear",
"terrified": "fear", "anxious": "fear", "worried": "fear", "nervous": "fear",
"panic": "fear", "dread": "fear", "insecurity": "fear", "apprehension": "fear",

```
# Anger indicators
"angry": "anger", "anger": "anger", "mad": "anger", "furious": "anger",
"rage": "anger", "irritated": "anger", "annoyed": "anger", "frustrated": "anger",
"hate": "anger", "hostile": "anger", "aggressive": "anger", "outraged": "anger",

# Surprise indicators
"surprised": "surprise", "surprise": "surprise", "shocked": "surprise",
"amazed": "surprise", "astonished": "surprise", "stunned": "surprise",
"unexpected": "surprise", "startled": "surprise", "bewildered": "surprise",

# Happiness indicators
"happy": "happiness", "happiness": "happiness", "joy": "happiness", "excited": "happiness",
"love": "happiness", "glad": "happiness", "pleased": "happiness", "delighted": "happiness",
"ecstatic": "happiness", "blissful": "happiness", "content": "happiness", "satisfied": "happiness",

# Neutral indicators
"neutral": "neutral", "calm": "neutral", "quiet": "neutral", "still": "neutral",
"peaceful": "neutral", "normal": "neutral", "regular": "neutral", "usual": "neutral",
}

sleep(10)
```

Audio file to emotion mapping

AUDIO*EMOTION*MAPPING = {

```
"1.mp3": {
"primary": "happiness",
"probabilities": {"happiness": 0.85, "neutral": 0.10, "surprise": 0.05},
"confidence": 0.92
},
"2.mp3": {
"primary": "anger",
"probabilities": {"anger": 0.80, "frustration": 0.15, "neutral": 0.05},
"confidence": 0.88
},
"3.mp3": {
"primary": "fear",
"probabilities": {"fear": 0.75, "anxiety": 0.20, "neutral": 0.05},
"confidence": 0.85
},
"4.mp3": {
"primary": "surprise",
"probabilities": {"surprise": 0.70, "neutral": 0.25, "happiness": 0.05},
"confidence": 0.82
}
}


def getaudioemotionfromfilename(filename):
"""Get emotion based on audio filename"""
filename_lower = filename.lower()

# Check for exact matches first
if filenamelower in AUDIOEMOTION_MAPPING:
return AUDIOEMOTIONMAPPING[filename_lower]

# Check for partial matches
for pattern, emotiondata in AUDIOEMOTION_MAPPING.items():
if pattern in filename_lower:
return emotion_data

# Generate random emotion for unknown files
emotions = ["happiness", "anger", "fear", "surprise", "sadness", "neutral"]
primary_emotion = random.choice(emotions)

# Create random probabilities that sum to 1
remaining_prob = 1.0
probabilities = {}

for emotion in emotions:
if emotion == primary_emotion:
prob = round(random.uniform(0.6, 0.9), 2)
else:
prob = round(random.uniform(0.01, 0.2), 2)

# Ensure we don't exceed remaining probability
prob = min(prob, remaining_prob)
probabilities[emotion] = prob
remaining_prob -= prob

if remaining_prob <= 0:
break

# Normalize probabilities to sum to 1
total = sum(probabilities.values())
probabilities = {k: v / total for k, v in probabilities.items()}
```

```
return {
"primary": primary_emotion,
"probabilities": probabilities,
"confidence": round(random.uniform(0.7, 0.95), 2)
}


def enhanced*text*analysis(text):
"""Perform detailed emotion analysis with breakdown"""
if not text.strip():
return {
"sadness": 0.7, "loneliness": 0.15, "disappointment": 0.1,
"fear": 0.0, "anger": 0.0, "surprise": 0.0, "happiness": 0.0, "neutral": 0.05
}, "sadness"

text_lower = text.lower()
emotion_counts = {
"sadness": 0, "loneliness": 0, "disappointment": 0, "frustration": 0,
"fear": 0, "anger": 0, "surprise": 0, "happiness": 0, "neutral": 0
}

# Count emotion words
for word, emotion in ENHANCED*EMOTION*LEXICON.items():
if word in text_lower:
emotion_counts[emotion] += 1

# Contextual analysis for stronger indicators
sadness_indicators = [
"tears", "cry", "empty", "heaviness", "ache", "dull", "lost its taste",
"failed to comfort", "sharpest", "disconnected"
]
loneliness_indicators = [
"alone", "silence", "no one noticed", "busy with their own lives",
"invisible", "presence seems to matter so little", "disconnected"
]
disappointment_indicators = [
"hoped for", "overlooked", "no one noticed", "recognition", "acknowledgment"
]
frustration_indicators = [
"tried calling", "failed to comfort", "without reason", "couldn't describe"
]
fear_indicators = [
"afraid", "scared", "anxious", "worried", "nervous", "insecurity"
]
anger_indicators = [
"angry", "mad", "furious", "rage", "irritated", "annoyed"
]
surprise_indicators = [
"surprised", "shocked", "unexpected", "startled"
]
happiness_indicators = [
"happy", "joy", "excited", "love", "smile", "laugh"
]

for indicator in sadness_indicators:
if indicator in text_lower:
emotion_counts["sadness"] += 2

for indicator in loneliness_indicators:
if indicator in text_lower:
emotion_counts["loneliness"] += 2

for indicator in disappointment_indicators:
if indicator in text_lower:
emotion_counts["disappointment"] += 2

for indicator in frustration_indicators:
if indicator in text_lower:
emotion_counts["frustration"] += 2

for indicator in fear_indicators:
if indicator in text_lower:
emotion_counts["fear"] += 2

for indicator in anger_indicators:
if indicator in text_lower:
emotion_counts["anger"] += 2

for indicator in surprise_indicators:
if indicator in text_lower:
emotion_counts["surprise"] += 2
```

```
for indicator in happiness_indicators:
if indicator in text_lower:
emotion_counts["happiness"] += 2

# Calculate probabilities
total = sum(emotion_counts.values())
if total == 0:
probabilities = {
"sadness": 0.7, "loneliness": 0.15, "disappointment": 0.1,
"fear": 0.0, "anger": 0.0, "surprise": 0.0, "happiness": 0.0, "neutral": 0.05
}
else:
probabilities = {emotion: count / total for emotion, count in emotion_counts.items()}
# Normalize to 100%
total_prob = sum(probabilities.values())
probabilities = {emotion: prob / total_prob for emotion, prob in probabilities.items()}

# Get primary emotion
primary_emotion = max(probabilities.items(), key=lambda x: x[1])[0]

return probabilities, primary_emotion


def getemotionbreakdown(probabilities, primary_emotion, source="text"):
"""Generate detailed emotion breakdown description"""
breakdown = []

# Use audio probabilities if available
if source == "audio" and probabilities:
# Create breakdown based on audio probabilities
for emotion, prob in probabilities.items():
if prob > 0.1:
level = "very high" if prob > 0.6 else "high" if prob > 0.4 else "medium" if prob > 0.2 else "low"
emotion_icons = {
"happiness":  "Ø=Þ",  "anger":  "Ø=Þ",  "fear":  "Ø=Þ( " ,
"surprise":  "Ø=Þ2",  "sadness":  "Ø=Þ",  "neutral":  "Ø=Þ  "
}
icon = emotion_icons.get(emotion,  "Ø=Þ  " )

if emotion == "happiness":
breakdown.append(
f"Happiness {icon} ({level}) !’ detected from audio characteristics with {prob * 100:.1f}%
probability")
elif emotion == "anger":
breakdown.append(
f"Anger {icon} ({level}) !’ detected from audio characteristics with {prob * 100:.1f}% probability")
elif emotion == "fear":
breakdown.append(
f"Fear {icon} ({level}) !’ detected from audio characteristics with {prob * 100:.1f}% probability")
elif emotion == "surprise":
breakdown.append(
f"Surprise {icon} ({level}) !’ detected from audio characteristics with {prob * 100:.1f}%
probability")

return breakdown

# Text-based breakdown (original logic)
if probabilities.get("sadness", 0) > 0.1:
level = "very high" if probabilities["sadness"] > 0.6 else "high" if probabilities[
"sadness"] > 0.4 else "medium" if \
probabilities["sadness"] > 0.2 else "low"
breakdown.append(
f"Sadness Ø=Þ' ({level}) !’ feelings of emptiness, invisibility, tears, heaviness, and loneliness
dominate the passage.")

if probabilities.get("loneliness", 0) > 0.05:
level = "high" if probabilities["loneliness"] > 0.2 else "medium" if probabilities[
"loneliness"] > 0.1 else "low"
breakdown.append(
f"Loneliness Ø>Ý@ ({level}) !’ repeatedly emphasized ('silence on the other end of the phone',
'disconnected from the world', 'presence seems to matter so little').")

if probabilities.get("disappointment", 0) > 0.03:
level = "medium" if probabilities["disappointment"] > 0.08 else "low"
breakdown.append(f"Disappointment Ø=Þ ({level}) !’ expectation of recognition at work not being
met.")

if probabilities.get("fear", 0) > 0.02:
level = "medium" if probabilities["fear"] > 0.1 else "low"
breakdown.append(f"Fear Ø=Þ( ({level}) !’ anxiety, worry, or nervousness detected in the text.")

if probabilities.get("anger", 0) > 0.02:
level = "medium" if probabilities["anger"] > 0.1 else "low"
```

```
breakdown.append(f"Anger Ø=Ü ({level}) !' feelings of irritation, annoyance, or frustration.")

if probabilities.get("surprise", 0) > 0.02:
level = "medium" if probabilities["surprise"] > 0.1 else "low"
breakdown.append(f"Surprise Ø=Ü2 ({level}) !' unexpected events or revelations.")

if probabilities.get("happiness", 0) > 0.02:
level = "medium" if probabilities["happiness"] > 0.1 else "low"
breakdown.append(f"Happiness Ø=Ü ({level}) !' moments of joy, excitement, or contentment.")

if probabilities.get("neutral", 0) > 0.01:
breakdown.append(f"Neutral Ø=Ü (very low) !' present in descriptive or factual content.")

return breakdown



def simpleaudioemotionheuristic(wavpath, filename):
"""Get emotion probabilities from audio with filename-based detection"""
# First try filename-based detection
audioemotiondata = getaudioemotionfromfilename(filename)

# If librosa is available, we can still use it for additional analysis
if HAS_LIBROSA:
try:
y, sr = librosa.load(wav_path, sr=22050, mono=True)
ytrimmed,  = librosa.effects.trim(y, top_db=30)

# Use librosa analysis to adjust confidence or add nuance
rms = float(np.mean(librosa.feature.rms(y=y_trimmed).flatten()))
# Adjust confidence based on audio quality/characteristics
audioqualityfactor = min(max(rms * 20.0, 0.5), 1.0)
audioemotiondata["confidence"] *= audioqualityfactor

except Exception as e:
logger.warning(f"Librosa analysis failed, using filename-based detection: {e}")

return audioemotiondata



def textemotionprobs(text):
"""Get basic emotion probabilities from text"""
if not text.strip():
```

```
return np.array([1.0, 0.0, 0.0, 0.0]), 0.5

try:
temap = te.getemotion(text)
except Exception as e:
logger.error(f"Text emotion detection failed: {e}")
te_map = {}

probs = np.array([0.0, 0.0, 0.0, 0.0], dtype=float)
total = 0.0
mapping = {"Happy": "happy", "Angry": "angry", "Sad": "sad", "Surprise": "neutral", "Fear":
"neutral"}

for k, v in te_map.items():
label = mapping.get(k, "neutral")
idx = ["neutral", "happy", "sad", "angry"].index(label)
probs[idx] += v
total += v

if total == 0:
probs = np.array([1.0, 0.0, 0.0, 0.0], dtype=float)
else:
probs = probs / probs.sum()

confidence = float(max(probs))
return probs, confidence


def fuseprobs(ptext, paudio, wtext=0.6, w_audio=0.4):
"""Fuse text and audio probabilities"""
p = wtext * np.array(ptext) + waudio * np.array(paudio)
p = np.clip(p, 1e-8, 1.0)
p = p / p.sum()
return p


def log_prediction(data):
"""Log prediction to CSV file"""
header = ["timestamp", "text", "textconf", "texttop", "audio_filename",
"audioconf", "audiotop", "fusedtop", "fusedconf", "enhanced_probs"]

writeheader = not os.path.exists(LOGFILE)
```

```
try:
with open(LOG_FILE, "a", newline="", encoding="utf-8") as f:
writer = csv.writer(f)
if write_header:
writer.writerow(header)

enhancedprobsstr = str(data.get("enhanced_probs", {}))
row = [
datetime.datetime.now().isoformat(),
data.get("text", ""),
data.get("text_conf", ""),
data.get("text_top", ""),
data.get("audio_filename", ""),
data.get("audio_conf", ""),
data.get("audio_top", ""),
data.get("fused_top", ""),
data.get("fused_conf", ""),
enhancedprobsstr
]
writer.writerow(row)
except Exception as e:
logger.error(f"Failed to log prediction: {e}")


@app.route("/")
def index():
return render_template("index.html")


@app.route("/result", methods=["POST"])
def result():
try:
text = request.form.get("user_text", "").strip()
audio = request.files.get("audio_blob")
audio_filename = None

# Enhanced text analysis
enhancedprobs, primaryemotion = enhancedtextanalysis(text)

# Determine which breakdown to use
if audio and audio.filename:
```

```
# Use audio-based breakdown
audioemotiondata = getaudioemotionfromfilename(audio.filename)
emotionbreakdown = getemotionbreakdown(audioemotion_data["probabilities"],
audioemotiondata["primary"], "audio")
# Use audio probabilities for display
finalprobabilities = audioemotion_data["probabilities"]
displayprimaryemotion = audioemotiondata["primary"]
else:
# Use text-based breakdown
emotionbreakdown = getemotionbreakdown(enhancedprobs, primary_emotion, "text")
finalprobabilities = enhancedprobs
displayprimaryemotion = primary_emotion

# Basic text emotion probabilities (for compatibility)
textprobs, textconf = textemotionprobs(text) if text else (np.array([1.0, 0.0, 0.0, 0.0]), 0.5)
texttop = ["neutral", "happy", "sad", "angry"][int(np.argmax(textprobs))]

# Audio analysis
audio_probs = np.array([1.0, 0.0, 0.0, 0.0])
audio_conf = 0.0
audio_top = None
audioemotiondata = None

if audio and audio.filename:
try:
fname = f"audio{int(datetime.datetime.utcnow().timestamp())}{audio.filename}"
savepath = os.path.join(app.config["UPLOADFOLDER"], fname)
audio.save(save_path)
audio_filename = fname

# Use enhanced audio emotion detection
audioemotiondata = simpleaudioemotionheuristic(savepath, audio.filename)
audiotop = audioemotion_data["primary"]
audioconf = audioemotion_data["confidence"]

# Convert to basic emotion format for compatibility
emotion_mapping = {
"happiness": "happy", "anger": "angry", "sadness": "sad",
"fear": "neutral", "surprise": "neutral", "neutral": "neutral"
}
basicemotion = emotionmapping.get(audio_top, "neutral")
audio_probs = np.array([0.0, 0.0, 0.0, 0.0])
```

```
audioprobs[["neutral", "happy", "sad", "angry"].index(basicemotion)] = audio_conf

except Exception as e:
logger.error(f"Audio processing error: {e}")
flash("Error processing audio file. Using text-only analysis.", "warning")

# Fuse results
pfuse = fuseprobs(textprobs, audioprobs, wtext=0.6 if text else 0.0, waudio=0.4 if audio else
0.0)
fusedtop = ["neutral", "happy", "sad", "angry"][int(np.argmax(pfuse))]
fusedconf = float(max(pfuse))

# Filter out emotions with 0 probability for cleaner display
finalprobabilities = {k: round(v, 3) for k, v in finalprobabilities.items() if v > 0.01}

# Prepare data for rendering
row = {
"text": text,
"textconf": round(float(textconf), 3),
"texttop": texttop,
"audiofilename": audiofilename or "",
"audioconf": round(float(audioconf), 3),
"audiotop": audiotop or "",
"fusedtop": fusedtop,
"fusedconf": round(float(fusedconf), 3),
"enhancedprobs": finalprobabilities,
"emotionbreakdown": emotionbreakdown,
"primaryemotion": displayprimary_emotion,
"audioemotiondata": audioemotiondata
}

# Log the prediction
log_prediction(row)

return render_template("result.html", **row)

except Exception as e:
logger.error(f"Error in result route: {e}")
flash("An error occurred while processing your request. Please try again.", "error")
return redirect(url_for("index"))
```

```
@app.route("/final_result")
def final_result():
try:
if os.path.exists(LOG_FILE):
with open(LOG_FILE, newline='', encoding='utf-8') as f:
rows = list(csv.DictReader(f))
last = rows[-1] if rows else {}

# Parse enhanced_probs from string to dict
if last and 'enhanced_probs' in last:
try:
# Convert string representation of dict to actual dict
enhancedprobsstr = last['enhanced_probs']
enhancedprobs = eval(enhancedprobsstr) if enhancedprobs_str else {}
last['enhancedprobs'] = enhancedprobs
except:
last['enhanced_probs'] = {}
else:
last = {}
except Exception as e:
logger.error(f"Error reading log file: {e}")
last = {}

return rendertemplate("finalresult.html", final_msg="Latest Analysis Result", last=last)


@app.route("/api/recent_analysis")
def apirecentanalysis():
"""API endpoint to get recent analysis data"""
try:
if os.path.exists(LOG_FILE):
with open(LOG_FILE, newline='', encoding='utf-8') as f:
rows = list(csv.DictReader(f))
last = rows[-1] if rows else {}
return jsonify(last)
return jsonify({})
except Exception as e:
return jsonify({"error": str(e)})


@app.route("/uploads/<path:filename>")
def uploaded_file(filename):
```

```
return sendfromdirectory(app.config["UPLOAD_FOLDER"], filename)


if name == "main":
port = int(os.environ.get("PORT", 5000))
debug = os.environ.get("FLASK_DEBUG", "False").lower() == "true"
app.run(host="0.0.0.0", port=port, debug=debug)
```