app.py

```
from flask import Flask, render_template, request, jsonify, send_file
import os
import google.generativeai as genai
from io import BytesIO
from PIL import Image
from gtts import gTTS
from langdetect import detect
from deep_translator import GoogleTranslator
import tempfile
from flask_cors import CORS
import logging
import base64

app = Flask(__name__)
CORS(app)
```

⍰ Creates the Flask application.

⍰ Enables **CORS** (Cross-Origin Resource Sharing) so the app can accept requests from different domains (useful if frontend is separate).

```
app.config['UPLOAD_FOLDER'] = 'static/uploads'
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB max file size
```

⍰ Sets a folder to store uploaded files.

⍰ Limits file upload size to 16MB.

```
if not os.path.exists(app.config['UPLOAD_FOLDER']):
    os.makedirs(app.config['UPLOAD_FOLDER'])
```

- Creates the uploads folder if it doesn't exist.

```
# Configure Gemini API
genai.configure(api_key="AIzaSyD4i5vCeP-dl8QRDetOdVc5gpjRe7SNe5o")

generation_config = {
    "temperature": 0.4,
    "top_p": 0.8,
    "top_k": 40,
    "max_output_tokens": 2048,
}

# Define the available languages
indian_languages = {
    "Hindi": "hi",
    "English": "en",
    "Marathi": "mr",
    "Telugu": "te",
    "Tamil": "ta",
    "Bengali": "bn",
    "Gujarati": "gu",
    "Kannada": "kn",
    "Malayalam": "ml",
    "Odia": "or",
    "Punjabi": "pa",
    "Urdu": "ur"
}

#upload_to_gemini()
def upload_to_gemini(image_file):
    try:
        logger.debug("Starting upload_to_gemini function")
        logger.debug(f"Image file type: {type(image_file)}")

        # Save the uploaded file temporarily
        temp_path = os.path.join(app.config['UPLOAD_FOLDER'], 'temp_image.jpg')
        image_file.save(temp_path)

        # Open with PIL and convert to base64
        with Image.open(temp_path) as image:
            logger.debug(f"Image opened successfully: {image.format}, size: {image.size}")

            # Convert to RGB if necessary
            if image.mode in ('RGBA', 'LA') or (image.mode == 'P' and 'transparency' in
image.info):
                logger.debug("Converting image to RGB")
                image = image.convert('RGB')

            # Save to bytes
            image_bytes = BytesIO()
            image.save(image_bytes, format='JPEG', quality=95)
            image_bytes.seek(0)

            # Convert to base64
            image_base64 = base64.b64encode(image_bytes.getvalue()).decode('utf-8')
            logger.debug("Image converted to base64 successfully")

            # Clean up temp file
            os.remove(temp_path)

            # Create Gemini-compatible parts
        return [{"mime_type": "image/jpeg", "data": image_base64}]

    except Exception as e:
        logger.error(f"Error in upload_to_gemini: {str(e)}")
        logger.exception("Detailed error:")
        return None
```

Purpose: Convert uploaded image into a format accepted by Gemini.

Steps:

1. Saves the uploaded image temporarily.
2. Opens it using PIL (Python Imaging Library).
3. Converts image to JPEG format and base64 encodes it.
4. Returns the image as a list with MIME type and base64 data.

#detect_language

```
def detect_language(text):
    try:
        return detect(text)
    except Exception as e:
        return None
```

Purpose: Detect the language of the extracted text.

- Uses langdetect to guess the language of the input string.
- Returns a short language code like 'en' or 'hi'.

#translate_text

```
def translate_text(text, dest_language):
    try:
        return GoogleTranslator(source='auto', target=dest_language).translate(text)
    except Exception as e:
        return None
```

Purpose: Translate text to the desired language.

- Uses deep_translator to convert text from detected language to target language (like Hindi, Tamil, etc.).
- Automatically detects the source language.

#text_to_speech

```
def text_to_speech(text, language):
    try:
        tts = gTTS(text=text, lang=language, slow=False)
        temp_file = tempfile.NamedTemporaryFile(delete=False, suffix=".mp3")
        tts.save(temp_file.name)
        return temp_file.name
    except Exception as e:
        return None
```

Purpose: Convert translated text into speech (MP3 audio).

- Uses gTTS (Google Text-to-Speech).
- Saves the audio to a temporary file and returns the file path.

#upload.html (inside script tag)

```
document.getElementById('uploadForm').addEventListener('submit', async (e) => {
```

This line adds an event listener to the **form**. When the user clicks the "Process Image" button, it runs this function instead of the default form submission.

```
e.preventDefault();
```

This stops the form from reloading the page. Instead, it uses JavaScript to handle everything.

```
const formData = new FormData(e.target);
```

This gathers all the form inputs (language and image file) into a FormData object so we can send it to the server.

```
document.getElementById('loadingSpinner').style.display = 'block';

document.getElementById('resultSection').style.display = 'none';
```

It shows the **"Processing..." spinner** and hides the previous result (if any) while the server is working.

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

This sends the data to the /process route on the backend using fetch() in **POST** method.

```
const data = await response.json();
```

This reads the server's response and converts it to a JSON object so we can use it.

```
if (response.ok) {
```

If everything goes well...

```
document.getElementById('detectedLanguage').textContent = data.detected_language || 'Unknown';
```

It sets the **detected language** on the page (e.g., Hindi, Tamil).

```
document.getElementById('translatedText').textContent = data.translated_text;
```

It displays the **translated text**.

```
audioPlayer.src = data.audio_path + '?t=' + new Date().getTime();
```

Sets the audio file URL. The + '?t=' + new Date().getTime() is a trick to **prevent audio from being cached**.

```
downloadButton.href = '/download_audio';

downloadButton.style.display = 'block';

document.getElementById('resultSection').style.display = 'block';
```

It shows the **Download Audio** button and makes the result section visible.

```
alert(data.error || 'An error occurred while processing the image');
```

It shows an alert message with the error.

```
document.getElementById('loadingSpinner').style.display = 'none';
```

Once everything is done (whether success or error), the loading spinner is hidden.

#preview image

```
document.getElementById('file').addEventListener('change', (e) => {
    const preview = document.getElementById('preview');
    const previewContainer = document.getElementById('preview-container');
    const file = e.target.files[0];

    if (file) {
        const reader = new FileReader();
        reader.onload = function(e) {
            preview.src = e.target.result;
            previewContainer.style.display = 'block';
        }
        reader.readAsDataURL(file);
    } else {
        previewContainer.style.display = 'none';
    }
});
```

When the user selects an image file:

- It reads the image using **FileReader**
- Converts it into a temporary URL
- Sets the image preview’s src to show it
- Shows the image inside a preview box