Back end – Python code

```
import os

import cv2

import easyocr

import threading

import logging

import shutil

import numpy as np

import time

from flask import Flask, render_template, request, jsonify, send_file, session, Response

from flask_socketio import SocketIO, emit

from werkzeug.utils import secure_filename

from googletrans import Translator

from gtts import gTTS

from langdetect import detect

from multiprocessing import Pool

from tempfile import NamedTemporaryFile

import uuid

app = Flask(__name__)

app.config["SECRET_KEY"] = str(uuid.uuid4())

app.config["UPLOAD_FOLDER"] = "uploads"

app.config["EXPORT_FOLDER"] = "exports"

os.makedirs(app.config["UPLOAD_FOLDER"], exist_ok=True)

os.makedirs(app.config["EXPORT_FOLDER"], exist_ok=True)

socketio = SocketIO(app, async_mode="eventlet")

logging.basicConfig(level=logging.INFO, filename="app.log", format="%(asctime)s - %(levelname)s -
%(message)s")

logger = logging.getLogger(__name__)

try:

    reader = easyocr.Reader(["en"], gpu=True)

    logger.info("Initialized EasyOCR with GPU")

except Exception as e:

    logger.warning(f"Failed to initialize EasyOCR with GPU: {str(e)}. Falling back to CPU")

    reader = easyocr.Reader(["en"], gpu=False)

translator = Translator()

live_ocr_running = False
```

```
live_ocr_lock = threading.Lock()
def allowed_file(filename):
    return "." in filename and filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS
def cleanup_uploads():
    try:
        shutil.rmtree(app.config["UPLOAD_FOLDER"], ignore_errors=True)
        os.makedirs(app.config["UPLOAD_FOLDER"])
        logger.info("Cleaned up uploads folder")
    except Exception as e:
        logger.error(f"Failed to clean up uploads: {str(e)}")
def safe_remove(file_path, retries=3, delay=0.5):
    for attempt in range(retries):
        try:
            os.remove(file_path)
            logger.debug(f"Successfully deleted {file_path}")
            return
        except PermissionError as e:
            logger.warning(f"Attempt {attempt + 1}/{retries} to delete {file_path} failed: {str(e)}")
            time.sleep(delay)
        except Exception as e:
            logger.error(f"Failed to delete {file_path}: {str(e)}")
            break
    logger.error(f"Could not delete {file_path} after {retries} attempts")
@app.route("/")
def index():
    try:
        return render_template("index.html")
    except Exception as e:
        logger.error(f"Error rendering index: {str(e)}")
        return jsonify({"error": "Failed to load page"}), 500
def process_frame(frame):
    try:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        results = reader.readtext(gray)
        return "\n".join([res[1] for res in results])
    except Exception as e:
        logger.error(f"Error processing frame: {str(e)}")
        return ""
@app.route("/upload_image", methods=["POST"])
def upload_image():
    try:
        files = request.files.getlist("file")
        if not files:
            logger.warning("No files uploaded in /upload_image")
            return jsonify({"error": "No files uploaded"}), 400
        extracted_texts = []
        annotated_images = []
        temp_files = []
        for file in files:
            if not file or not allowed_file(file.filename):
                logger.warning(f"Invalid file type: {file.filename}")
                return jsonify({"error": f"Invalid file type for {file.filename}. Allowed: png, jpg, jpeg"}), 400
            if file.content_length > MAX_FILE_SIZE:
                logger.warning(f"File too large: {file.filename}")
                return jsonify({"error": f"File {file.filename} too large. Max 10MB"}), 400
            temp_file = NamedTemporaryFile(delete=False, suffix=".png")
            temp_file_path = temp_file.name
            temp_files.append(temp_file_path)
            try:
                file.save(temp_file_path)
                temp_file.close()
                img = cv2.imread(temp_file_path)
                if img is None:
                    logger.error(f"Invalid image file: {file.filename}")
                    return jsonify({"error": f"Invalid image file: {file.filename}"}), 400
                gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                results = reader.readtext(gray)
                extracted_text = "\n".join([res[1] for res in results])
                for res in results:
                    (top_left, top_right, bottom_right, bottom_left), text = res[0], res[1]
                    top_left = tuple(map(int, top_left))
                    bottom_right = tuple(map(int, bottom_right))
                    cv2.rectangle(img, top_left, bottom_right, (0, 255, 0), 2)
                    cv2.putText(img, text, top_left, cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
                annotated_path = os.path.join(app.config["EXPORT_FOLDER"],
f"annotated_{secure_filename(file.filename)}")
                cv2.imwrite(annotated_path, img)
                extracted_texts.append(extracted_text)
                annotated_images.append(annotated_path)
            finally:
                safe_remove(temp_file_path)
        extracted_text = "\n\n".join(extracted_texts)
        session["last_extracted_text"] = extracted_text
        try:
            session["last_lang_code"] = detect(extracted_text) if extracted_text else "en"
        except:
            session["last_lang_code"] = "en"
        logger.info(f"Processed {len(files)} images")
        return jsonify({"text": extracted_text, "annotated_images": annotated_images})
    except Exception as e:
        logger.error(f"Error in upload_image: {str(e)}")
        for temp_file in temp_files:
            safe_remove(temp_file)
        return jsonify({"error": f"Failed to process images: {str(e)}"}), 500
@app.route("/upload_video", methods=["POST"])
def upload_video():
    try:
        file = request.files.get("file")
        if not file or file.filename == "":
            logger.warning("No file uploaded in /upload_video")
            return jsonify({"error": "No file uploaded"}), 400
        if not allowed_file(file.filename):
            logger.warning(f"Invalid file type: {file.filename}")
            return jsonify({"error": "Invalid file type. Allowed: mp4, avi"}), 400
        if file.content_length > MAX_FILE_SIZE:
            logger.warning(f"File too large: {file.filename}")
            return jsonify({"error": "File too large. Max 10MB"}), 400
        temp_file = NamedTemporaryFile(delete=False, suffix=".mp4")
```

```
temp_file_path = temp_file.name
try:
    file.save(temp_file_path)
    temp_file.close()
    cap = cv2.VideoCapture(temp_file_path)
    if not cap.isOpened():
        logger.error(f"Invalid video file: {file.filename}")
        return jsonify({"error": "Invalid video file"}), 400
    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_interval = max(1, int(fps / 2))
    extracted_texts = []
    frame_count = 0
    frames_to_process = []
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        if frame_count % frame_interval == 0:
            frames_to_process.append(frame)
        frame_count += 1
    try:
        with Pool(processes=4) as pool:
            results = pool.map(process_frame, frames_to_process)
            extracted_texts.extend([r for r in results if r])
    except Exception as e:
        logger.error(f"Multiprocessing error: {str(e)}")
        extracted_texts.extend([process_frame(f) for f in frames_to_process])
    cap.release()
    extracted_text = "\n".join(extracted_texts)
    session["last_extracted_text"] = extracted_text
    try:
        session["last_lang_code"] = detect(extracted_text) if extracted_text else "en"
    except:
        session["last_lang_code"] = "en"
    logger.info(f"Processed video: {file.filename}")
    return jsonify({"text": extracted_text})
```

```
        finally:
            safe_remove(temp_file_path)
    except Exception as e:
        logger.error(f"Error in upload_video: {str(e)}")
        safe_remove(temp_file_path)
        return jsonify({"error": f"Failed to process video: {str(e)}"}), 500
def generate_frames():
    global live_ocr_running
    try:
        cap = cv2.VideoCapture(0)
        if not cap.isOpened():
            logger.error("Failed to open webcam")
            return
        while live_ocr_running and cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            ret, buffer = cv2.imencode(".jpg", frame)
            frame = buffer.tobytes()
            yield (b"--frame\r\n" b"Content-Type: image/jpeg\r\n\r\n" + frame + b"\r\n")
        cap.release()
    except Exception as e:
        logger.error(f"Error in generate_frames: {str(e)}")
def live_ocr_stream():
    global live_ocr_running
    try:
        cap = cv2.VideoCapture(0)
        if not cap.isOpened():
            logger.error("Failed to open webcam")
            return
        with live_ocr_lock:
            while live_ocr_running and cap.isOpened():
                ret, frame = cap.read()
                if not ret:
                    break
                gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                results = reader.readtext(gray)
                text = "\n".join([res[1] for res in results])
                socketio.emit("live_text", {"text": text})
        cap.release()
    except Exception as e:
        logger.error(f"Error in live_ocr_stream: {str(e)}")
@app.route("/video_feed")
def video_feed():
    try:
        return Response(generate_frames(), mimetype="multipart/x-mixed-replace; boundary=frame")
    except Exception as e:
        logger.error(f"Error in video_feed: {str(e)}")
        return jsonify({"error": "Failed to stream video"}), 500
@socketio.on("start_live_ocr"
def start_live_ocr():
    global live_ocr_running
    try:
        with live_ocr_lock:
            if not live_ocr_running:
                live_ocr_running = True
                threading.Thread(target=live_ocr_stream, daemon=True).start()
                logger.info("Started live OCR")
    except Exception as e:
        logger.error(f"Error starting live OCR: {str(e)}")
        socketio.emit("error", {"error": "Failed to start live OCR"})
@socketio.on("stop_live_ocr")
def stop_live_ocr():
    global live_ocr_running
    try:
        with live_ocr_lock:
            live_ocr_running = False
        logger.info("Stopped live OCR")
    except Exception as e:
        logger.error(f"Error stopping live OCR: {str(e)}")
        socketio.emit("error", {"error": "Failed to stop live OCR"})
@app.route("/export_txt", methods=["GET"])
```

```
def export_txt():
    try:
        extracted_text = session.get("last_extracted_text", "")
        if not extracted_text:
            logger.warning("No text to export")
            return jsonify({"error": "No text to export"}), 400
        file_path = os.path.join(app.config["EXPORT_FOLDER"], "extracted_text.txt")
        with open(file_path, "w", encoding="utf-8") as f:
            f.write(extracted_text)
        response = send_file(file_path, as_attachment=True)
        try:
            safe_remove(file_path)
        except:
            logger.warning(f"Failed to delete {file_path}")
        logger.info("Exported text to TXT")
        return response
    except Exception as e:
        logger.error(f"Error in export_txt: {str(e)}")
        return jsonify({"error": f"Failed to export text: {str(e)}"}), 500
@app.route("/translate_text", methods=["POST"])
def translate_text():
    try:
        target_lang = request.json.get("lang", "es")
        # Supported languages: en (English), ta (Tamil), es (Spanish), fr (French), de (German), hi (Hindi), zh
(Chinese), ja (Japanese)
        extracted_text = session.get("last_extracted_text", "")
        if not extracted_text:
            logger.warning("No text to translate")
            return jsonify({"error": "No text to translate"}), 400
        translated = translator.translate(extracted_text, dest=target_lang)
        session["last_extracted_text"] = translated.text
        session["last_lang_code"] = target_lang
        logger.info(f"Translated text to {target_lang}")
        return jsonify({"text": translated.text})
    except Exception as e:
        logger.error(f"Error in translate_text: {str(e)}")
        return jsonify({"error": f"Failed to translate text: {str(e)}"}), 500
@app.route("/speak_text", methods=["GET"])
def speak_text():
    try:
        extracted_text = session.get("last_extracted_text", "")
        lang_code = session.get("last_lang_code", "en")
        if not extracted_text:
            logger.warning("No text to convert to speech")
            return jsonify({"error": "No text to convert to speech"}), 400
        output_path = os.path.join(app.config["EXPORT_FOLDER"], f"speech_{uuid.uuid4().hex}.mp3")
        tts = gTTS(text=extracted_text, lang=lang_code)
        tts.save(output_path)
        audio_url = f"/exports/{os.path.basename(output_path)}"
        logger.info(f"Generated speech audio: {output_path}")
        return jsonify({"audio_url": audio_url})
    except Exception as e:
        logger.error(f"Error in speak_text: {str(e)}")
        return jsonify({"error": f"Failed to generate speech: {str(e)}"}), 500
@app.route("/exports/<filename>")
def serve_exported_file(filename):
    try:
        file_path = os.path.join(app.config["EXPORT_FOLDER"], filename)
        if not os.path.exists(file_path):
            logger.warning(f"File not found: {file_path}")
            return jsonify({"error": "File not found"}), 404
        response = send_file(file_path, mimetype="audio/mpeg")
        threading.Timer(60, lambda: safe_remove(file_path)).start()
        logger.info(f"Served file: {file_path}")
        return response
    except Exception as e:
        logger.error(f"Error serving file {filename}: {str(e)}")
        return jsonify({"error": f"Failed to serve file: {str(e)}"}), 500
if __name__ == "__main__":
    try:
        cleanup_uploads()
        socketio.run(app, debug=True)
    except KeyboardInterrupt:
        logger.info("Shutting down server")
    except Exception as e:
        logger.error(f"Error running server: {str(e)}")
    finally:
        cleanup_uploads()
```

Front end – HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>OCR Flask App</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
  <script src="https://cdn.socket.io/4.4.1/socket.io.min.js"></script>
  <script src="{{ url_for('static', filename='script.js') }}" defer></script>
</head>
<body>
<div class="container">
  <h1>TEXT EXTRACTION APP 🗣️💬🌐</h1>
  <div class="options">
    <label for="imageUpload" class="button">📷 Upload Image(s)</label>
    <input type="file" id="imageUpload" accept="image/*" multiple hidden />
    <label for="videoUpload" class="button">🎥 Upload Video</label>
    <input type="file" id="videoUpload" accept="video/*" hidden />
    <button class="button" id="startLive">🔴 Start Live OCR</button>
    <button class="button" id="stopLive">🛑 Stop Live OCR</button>
    <select id="languageSelect" class="dropdown">
      <option value="en">English</option>
      <option value="ta">Tamil</option>
      <option value="es">Spanish</option>
      <option value="fr">French</option>
      <option value="de">German</option>
      <option value="hi">Hindi</option>
      <option value="zh">Chinese</option>
      <option value="ja">Japanese</option>
    </select>
    <button class="button" id="translate">🌍 Translate Text</button>
    <button class="button" id="speak">🔊 Text to Speech</button>
    <button class="button" id="exportTxt">📤 Export to TXT</button>
  </div>
  <div class="output">
    <p id="notification"></p>
    <textarea id="outputText" placeholder="Extracted text will appear here..." readonly></textarea>
    <div id="audioContainer" class="audio-container hidden">
      <audio id="audioPlayer" controls></audio>
      <button class="button" id="toggleAudio">⏯️ Pause</button>
    </div>
    <div id="annotatedImages"></div>
  </div>
</div>
<div id="livePopup" class="popup hidden">
  <div id="popupHeader">📡 Live OCR Feed (Drag me)</div>
  <video id="liveVideo" src="/video_feed" autoplay muted></video>
  <textarea id="liveText" readonly></textarea>
</div>
</body>
</html>
```

Front end – CSS

```
body {
  background: linear-gradient(45deg, #141E30, #243B55);
  font-family: 'Segoe UI', sans-serif;
  color: white;
  text-align: center;
  margin: 0;
  padding: 0;
}
.container {
  padding: 30px;
}
```

```
h1 {
  margin-bottom: 30px;
  font-size: 32px;
}
.options {
  display: flex;
  flex-direction: column;
  gap: 15px;
  align-items: center;
}
.button {
  background: #007acc;
  color: white;
  padding: 15px 30px;
  font-size: 16px;
  border: none;
  border-radius: 10px;
  cursor: pointer;
  transition: all 0.3s ease-in-out;
}
.button:hover {
  background-color: #005f99;
  transform: scale(1.05);
}
.output {
  margin-top: 30px;
}
textarea {
  width: 80%;
  height: 200px;
  border: none;
  border-radius: 8px;
  background: #2f2f2f;
  color: white;
  font-size: 16px;
  padding: 10px;
```

```
}
#notification {
  font-size: 16px;
  color: #f0f0f0;
  margin-bottom: 10px;
}
.popup {
  position: fixed;
  top: 20px;
  right: 20px;
  width: 300px;
  background-color: #1f1f1f;
  border: 2px solid #007acc;
  border-radius: 10px;
  z-index: 1000;
  padding: 10px;
  box-shadow: 0 0 15px #007acc;
  cursor: move;
}
.popup.hidden {
  display: none;
}
.popup textarea {
  width: 100%;
  height: 120px;
  margin-top: 10px;
  background-color: #2f2f2f;
  color: white;
  border: none;
  border-radius: 8px;
  padding: 8px;
}
.popup video {
  width: 100%;
  height: auto;
  border-radius: 8px;
```

```
}
.dropdown {
  padding: 10px;
  font-size: 16px;
  border-radius: 10px;
  border: none;
  background-color: #007acc;
  color: white;
}
#annotatedImages img {
  max-width: 80%;
  margin: 20px auto;
  display: block;
  border-radius: 8px;
}
.audio-container {
  margin-top: 15px;
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 10px;
}
.audio-container.hidden {
  display: none;
}
#audioPlayer {
  width: 80%;
  background: #2f2f2f;
  border-radius: 8px;
}
#toggleAudio {
  padding: 10px 20px;
  font-size: 14px;
}
```

Front end – Java Script

```
const imageUpload = document.getElementById("imageUpload");
const videoUpload = document.getElementById("videoUpload");
const outputText = document.getElementById("outputText");
const notification = document.getElementById("notification");
const languageSelect = document.getElementById("languageSelect");
const annotatedImages = document.getElementById("annotatedImages");
const audioPlayer = document.getElementById("audioPlayer");
const toggleAudio = document.getElementById("toggleAudio");
const audioContainer = document.getElementById("audioContainer");
let socket = null;
let selectedLang = languageSelect.value;

languageSelect.addEventListener("change", () => {
  selectedLang = languageSelect.value;
});

imageUpload.addEventListener("change", () => {
  if (imageUpload.files.length > 0) {
    notification.textContent = "📤 Uploading images... Working on text extraction...";
    let formData = new FormData();
    for (let file of imageUpload.files) {
      formData.append("file", file);
    }

    fetch("/upload_image", { method: "POST", body: formData })
      .then(res => {
        if (!res.ok) throw new Error(`Server error: ${res.status}`);
        return res.json();
      })
      .then(data => {
        if (data.error) throw new Error(data.error);
        outputText.value = data.text || "No text found.";
        annotatedImages.innerHTML = "";
        if (data.annotated_images) {
          data.annotated_images.forEach(src => {
            const img = document.createElement("img");
            img.src = src;
```

```
                annotatedImages.appendChild(img);
              });
            }
            notification.textContent = `✅ Text extracted from ${imageUpload.files.length} image(s).`;
            // Hide audio controls when new content is loaded
            audioContainer.classList.add("hidden");
          })
          .catch(err => {
            notification.textContent = `❌ ${err.message}`;
            console.error(err);
          });
      }
    });
    videoUpload.addEventListener("change", () => {
      if (videoUpload.files.length > 0) {
        notification.textContent = "📤 Uploading video... Working on text extraction...";
        let formData = new FormData();
        formData.append("file", videoUpload.files[0]);
        fetch("/upload_video", { method: "POST", body: formData })
          .then(res => {
            if (!res.ok) throw new Error(`Server error: ${res.status}`);
            return res.json();
          })
          .then(data => {
            if (data.error) throw new Error(data.error);
            outputText.value = data.text || "No text found.";
            annotatedImages.innerHTML = "";
            notification.textContent = "✅ Text extracted from video.";
            audioContainer.classList.add("hidden");
          })
          .catch(err => {
            notification.textContent = `❌ ${err.message}`;
            console.error(err);
          });
      }
```

```
document.getElementById("startLive").addEventListener("click", () => {
  notification.textContent = "📸 Starting live OCR...";
  socket = io.connect(location.origin);
  socket.emit("start_live_ocr");
  document.getElementById("livePopup").classList.remove("hidden");
  annotatedImages.innerHTML = "";
  audioContainer.classList.add("hidden");
  socket.on("live_text", (data) => {
    document.getElementById("liveText").value = data.text;
    outputText.value = data.text;
    notification.textContent = " 📡 Receiving live OCR data...";
  });
  socket.on("error", (data) => {
    notification.textContent = `❌ ${data.error}`;
    document.getElementById("livePopup").classList.add("hidden");
    socket.disconnect();
  });
.getElementById("livePopup").classList.add("hidden");
  notification.textContent = "🛑 Live OCR stopped.";
  audioContainer.classList.add("hidden");
});
document.getElementById("exportTxt").addEventListener("click", () => {
document.getElementById("translate").addEventListener("click", () => {
  notification.textContent = "🌍 Translating text...";
  fetch("/translate_text", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ lang: selectedLang })
  })
  .then(res => {
    if (!res.ok) throw new Error(`Server error: ${res.status}`);
    return res.json();
  })
  .then(data => {
    if (data.error) throw new Error(data.error);
```

```
        outputText.value = data.text;
        annotatedImages.innerHTML = "";
        notification.textContent = "✅ Translation complete.";
        audioContainer.classList.add("hidden");
    })
    .catch(err => {
        notification.textContent = `❌ ${err.message}`;
        console.error(err);
    });
});
document.getElementById("speak").addEventListener("click", () => {
    notification.textContent = "🔊 Converting to speech...";
    fetch("/speak_text")
        .then(response => {
            if (!response.ok) throw new Error(`Server error: ${response.status}`);
            return response.json();
        })
        .then(data => {
            if (data.error) throw new Error(data.error);
            audioPlayer.src = data.audio_url;
            audioContainer.classList.remove("hidden");
            audioPlayer.play();
            toggleAudio.textContent = "⏯️ Pause";
            notification.textContent = "🎧 Playing speech audio.";
        })
        .catch(err => {
            notification.textContent = `❌ ${err.message}`;
            console.error(err);
            audioContainer.classList.add("hidden");
        });
});
toggleAudio.addEventListener("click", () => {
    if (audioPlayer.paused) {
        audioPlayer.play();
        toggleAudio.textContent = "⏯️ Pause";
    } else {
        audioPlayer.pause();
        toggleAudio.textContent = "⏯️ Play";
    }
});
const livePopup = document.getElementById("livePopup");
const popupHeader = document.getElementById("popupHeader");
let offsetX, offsetY;
popupHeader.addEventListener("mousedown", (e) => {
    offsetX = e.clientX - livePopup.offsetLeft;
    offsetY = e.clientY - livePopup.offsetTop;
    document.addEventListener("mousemove", movePopup);
    document.addEventListener("mouseup", () => {
        document.removeEventListener("mousemove", movePopup);
    });
});
function movePopup(e) {
    livePopup.style.left = `${e.clientX - offsetX}px`;
    livePopup.style.top = `${e.clientY - offsetY}px`;
}
```