**AI OCR Assignment: Business Card Data Extraction**

**Project Overview**

Objective: The project aims to develop an application that allows users to upload images of business cards and extract relevant information such as name, address, phone number, email, and company details using Optical Character Recognition (OCR) technology. The extracted information is displayed in a structured format on the webpage, enhancing accessibility and usability.

**Approach**

**1. Technology Stack:**

**Frontend:**

ReactJS: For building the user interface.

Tailwind CSS: For styling the application.

Axios: For making HTTP requests to the backend.

**Backend:**

Node.js: For server-side logic.

Express.js: For handling HTTP requests and routing.

Multer: For handling file uploads.

**OCR Processing:**

Python: For the OCR processing script.

OpenCV: For image preprocessing (grayscale conversion, noise reduction).

Pytesseract: For text extraction from images.

**2. Process Flow:**

Image Upload: Users upload an image of a business card via the frontend interface.

Preprocessing: The backend server receives the image and preprocesses it to enhance OCR accuracy.

OCR Processing: The preprocessed image is passed to a Python script that uses Tesseract OCR to extract text.

Data Parsing: The extracted text is parsed to identify and categorize relevant fields (e.g., name, address, phone,etc…).

Result Display: The parsed information and images are sent back to the frontend and displayed in a structured format.

**Error Handling**

**1. File Upload Errors:**

Invalid File Type: The server validates the file type and size before processing. If the file is invalid, an appropriate error message is returned.

File Save Errors: If there is an error saving the uploaded file, the server returns an error message.

**2. OCR Processing Errors:**

Image Preprocessing Errors: If there is an issue during image preprocessing (e.g., file read errors), the server logs the error and returns a corresponding error message.

Text Extraction Errors: If OCR fails to extract text, the error is caught, logged, and an error message is returned to the user.

**3. Network Errors:**

Server Unavailability: The frontend handles network errors gracefully by displaying user-friendly messages when the server is unreachable.

**SOURCE CODE:**

**Ocr.py**

import cv2

import pytesseract

import sys

import json

import re

import os

import numpy as np  # Ensure numpy is imported

pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

#preprocessing the image

def preprocess\_image(image\_path):

    try:

        image = cv2.imread(image\_path)

        gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

        gray = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)

        gray = cv2.fastNlMeansDenoising(gray, None, 30, 7, 21)

        kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])

        gray = cv2.filter2D(gray, -1, kernel)

        return image, gray

    except Exception as e:

        print(json.dumps({"error": f"Preprocess error: {str(e)}"}))

        sys.exit(1)

#Extract the from the image

def extract\_text(image):

    try:

        custom\_config = r'--oem 3 --psm 6'

        text = pytesseract.image\_to\_string(image, config=custom\_config)

        return text

    except Exception as e:

        print(json.dumps({"error": f"Extract text error: {str(e)}"}))

        sys.exit(1)

#parsing the text

def parse\_text(text):

    result = {

        'Name': None,

        'Address': None,

        'Phone': None,

        # 'Mobile': None,

        'Company': None,

        'Job': None,

        'Email': None,

        'Web': None

    }

    patterns = {

        'Phone': r'(?:(?:\+?\d{1,4}[-.\s])?(?:\(?\d{1,5}\)?[-.\s]?)?\d{1,4}[-.\s]?\d{1,4}[-.\s]?\d{1,9})',

        # 'Mobile': r'(?:(?:\+?\d{1,4}[-.\s])?(?:\(?\d{1,5}\)?[-.\s]?)?\d{1,4}[-.\s]?\d{1,4}[-.\s]?\d{1,9})',

        'Email': r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}',

        'Web': r'(?:http://|https://|www\.)[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(?:/[^\s]\*)?',

    }

    lines = text.split('\n')

    for line in lines:

        line = line.strip()

        if re.search(patterns['Phone'], line):

            result['Phone'] = line

        # elif re.search(patterns['Mobile'], line):

        #     result['Mobile'] = line

        elif re.search(patterns['Email'], line):

            result['Email'] = line

        elif re.search(patterns['Web'], line):

            result['Web'] = line

        elif result['Name'] is None:

            result['Name'] = line

        elif result['Address'] is None:

            if result['Company'] is None:

                result['Address'] = line

            else:

                result['Address'] += ' ' + line

        elif result['Company'] is None:

            result['Company'] = line

    return result

def main(image\_path):

    try:

        image, gray\_image = preprocess\_image(image\_path)

        text = extract\_text(gray\_image)

        result = parse\_text(text)

        gray\_image\_path = os.path.splitext(image\_path)[0] + '\_gray.png'

        cv2.imwrite(gray\_image\_path, gray\_image)

        output = {

            'ocr\_result': result,

            'original\_image\_path': image\_path,

            'grayscale\_image\_path': gray\_image\_path

        }

        print(json.dumps(output))

    except Exception as e:

        print(json.dumps({"error": f"Main error: {str(e)}"}))

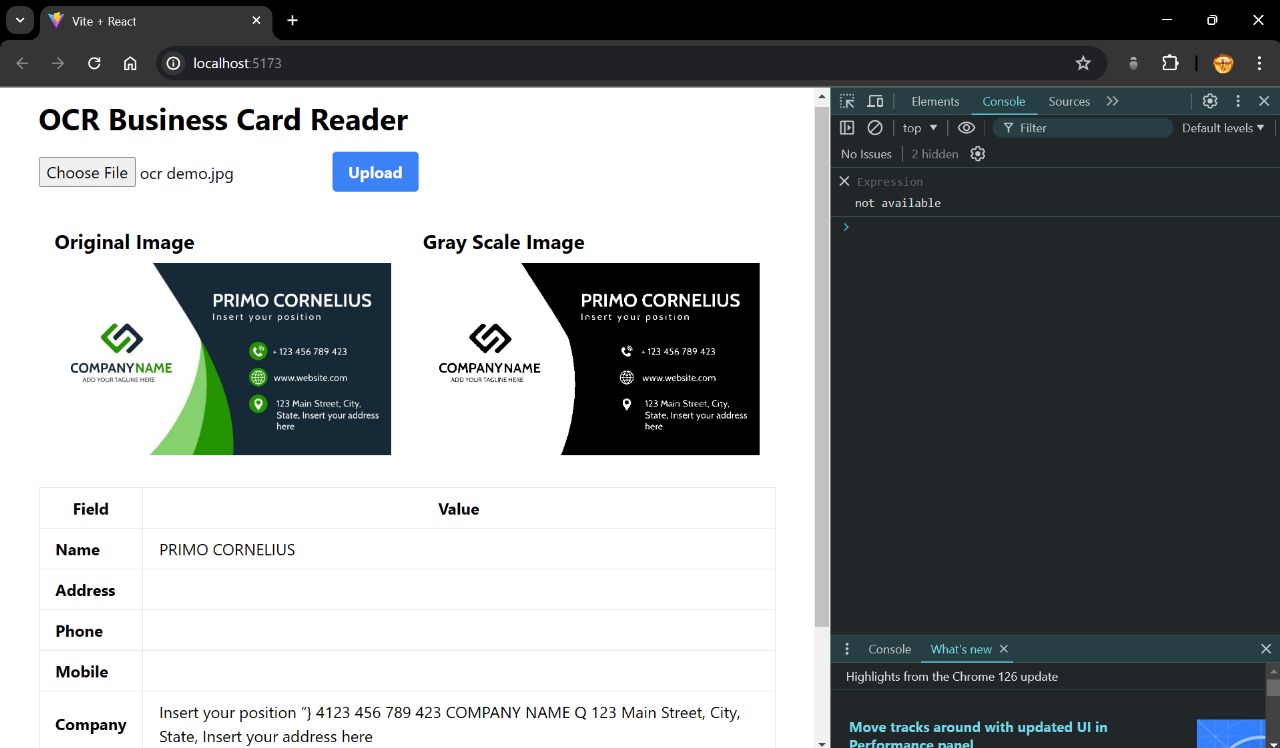
        sys.exit(1)

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

    main(sys.argv[1])

**OUTPUT:**

**BEFORE** **IMPROVING** **ACCURACY**:



**AFTER OPTIMIZED THE** **CODE**:

