LABORATORY REPORT

Application Development Lab
(CS33002)

B.Tech Program in ECSc

Submitted By

Name:- Bhairav Ganguly

Roll No: 2230246

Kalinga Institute of Industrial Technology
(Deemed to be University)
Bhubaneswar, India

Spring 2024-2025

Table of Content

Exp No.	Title	Date of Experiment	Date of Submission	Remarks
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				
10.	Backend report of OCR PDF & Chat	02/04/25	05/04/25	

Experiment Number	10
Experiment Title	Backend report of OCR PDF & Chat
Date of Experiment	02/04/25
Date of Submission	05/04/25

1. Objective:- The objective of this lab experiment is to create a Python program that converts handwritten notes and digital text from PDF files into digital text, enhancing accessibility. Additionally, it allows users to interact with the PDF content through natural language processing for efficient information retrieval and user engagement.

2. Procedure:- The project involved using Python as the core programming language to integrate Google Vision API for optical character recognition (OCR) to extract text from handwritten notes and digital text PDFs. For the interactive chat functionality with the PDF content, the Groq API Llama 3-8b model was employed to implement natural language processing. The frontend of the application was developed using Vanilla JavaScript, HTML, and CSS, further styled with Tailwind CSS to ensure a user-friendly and responsive interface.

3. Code:-

App.py:

```
import os
import io
import json
from flask import Flask, render_template, request, jsonify, redirect, url_for, session
from werkzeug.utils import secure_filename
from google.cloud import vision
from PyPDF2 import PdfReader
from pdf2image import convert_from_bytes
import groq
from dotenv import load_dotenv
from flask_socketio import SocketIO, emit

load_dotenv()
```

```
app = Flask(__name__)
app.secret_key = os.getenv('FLASK_SECRET_KEY', 'supersecretkey')
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'pdf'}
socketio = SocketIO(app)


vision_client = vision.ImageAnnotatorClient()
groq_client = groq.Client(api_key=os.getenv("GROQ_API_KEY"))

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in
app.config['ALLOWED_EXTENSIONS']

def extract_text_from_pdf(pdf_path):
    text = ""
    with open(pdf_path, 'rb') as file:
        pdf_reader = PdfReader(file)
        for page in pdf_reader.pages:
            text += page.extract_text() or ""
    return text

def perform_ocr_on_pdf(pdf_path):
    images = convert_from_bytes(open(pdf_path, 'rb').read())
    full_text = ""

    for image in images:
        byte_arr = io.BytesIO()
        image.save(byte_arr, format='PNG')
        byte_arr = byte_arr.getvalue()

        image = vision.Image(content=byte_arr)
        response = vision_client.text_detection(image=image)
        texts = response.text_annotations

        if texts:
            full_text += texts[0].description + "\n\n"

    return full_text

@app.route('/', methods=['GET', 'POST'])
```

```
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)

        file = request.files['file']
        if file.filename == '':
            return redirect(request.url)

        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
            file.save(filepath)


            extracted_text = extract_text_from_pdf(filepath)
            if len(extracted_text.strip()) < 100:
                extracted_text = perform_ocr_on_pdf(filepath)

            session['pdf_text'] = extracted_text
            return redirect(url_for('chat'))

    return render_template('upload.html')

@app.route('/chat')
def chat():
    if 'pdf_text' not in session:
        return redirect(url_for('upload_file'))
    return render_template('chat.html')

@socketio.on('send_message')
def handle_message(data):
    user_message = data['message']
    pdf_text = session.get('pdf_text', '')

    try:
        prompt = f"""
        DOCUMENT CONTENT:
        {pdf_text[:15000]}

        USER QUESTION: {user_message}
```

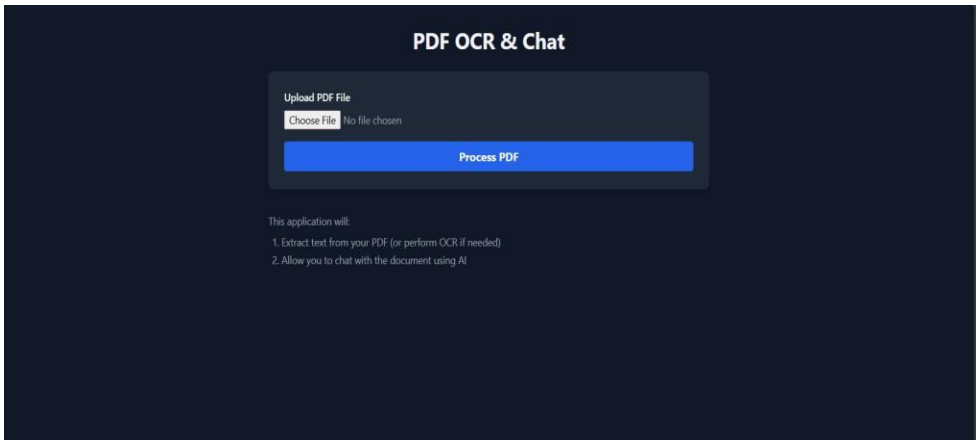
```
        Please provide a detailed answer based on the document content. Format your response with:
        - Clear bullet points (start each with '- ')
        - One blank line between points
        - Concise yet informative points
        - If no relevant info, state that clearly
        """

        chat_completion =
            groq_client.chat.completions.create( messages=[
                {
                    "role": "system",
                    "content": "You are a helpful AI assistant that provides detailed answers about document contents. Format responses with clear bullet points using hyphens and proper spacing."
                },
                {
                    "role": "user",
                    "content": prompt
                }
            ],
            model="llama3-8b-8192",
            temperature=0.2,
            max_tokens=2048
        )

        response = chat_completion.choices[0].message.content

        html_response = response.replace('\n- ', '<br>- ').replace('\n\n', '<br><br>')
        emit('receive_message', {'message': html_response})
    except Exception as e:
        emit('receive_message', {'message': f"Error: {str(e)}"})

def format_response(text):
    """Convert plain text response to formatted HTML with proper point spacing"""

    lines = text.split('\n')
    formatted_lines = []

    for line in lines:
        line = line.strip()
```

```
        if line.startswith('- '):
            formatted_lines.append(f'<li>{line[2:]}</li>')
        elif line == '':
            continue
        else:
            formatted_lines.append(f'<p>{line}</p>')

    formatted_text = '\n'.join(formatted_lines)

    if any('<li>' in line for line in formatted_lines):
        formatted_text = formatted_text.replace('<li>', '<ul><li>', 1)
        formatted_text = formatted_text[::-1].replace('>il/<', '>lu/<', 1)[::-1]

    return formatted_text

if __name__ == '__main__':
    os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
    socketio.run(app, debug=True)
```

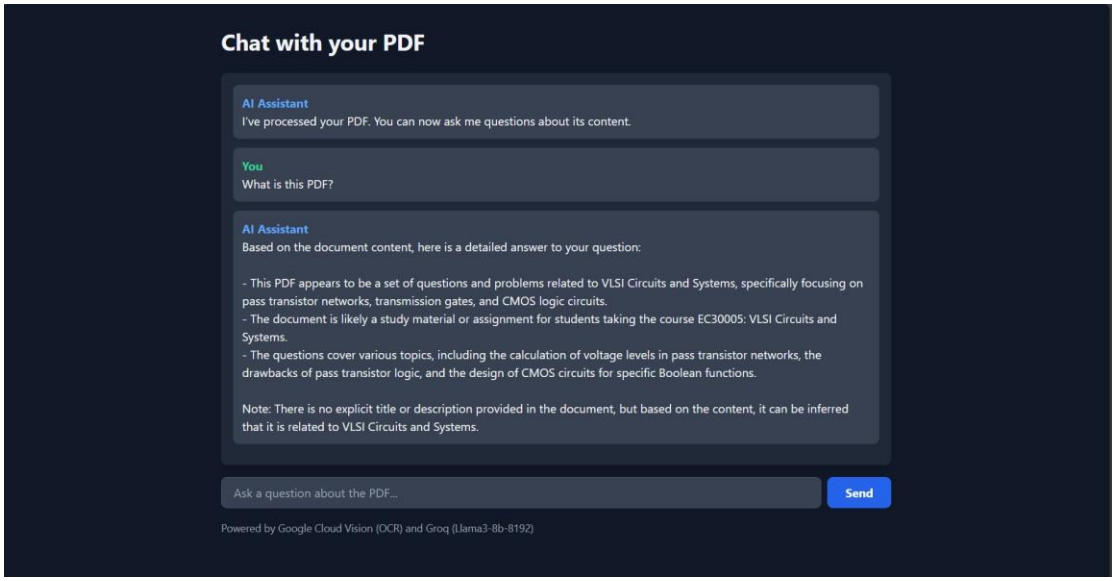
GitHub Repo link: https://github.com/Bhairavg7/AD_LAB_OPEN_END.git

4. Output:-

Landing Page:



Chat with PDF page:



5. Conclusion: In this experiment, we successfully integrated Python flask backed with frontend Using Vanilla JavaScript, HTML, and CSS, the structure and functionality were implemented effectively, while Tailwind CSS was utilized to streamline styling and create a visually appealing and adaptable user interface. This cohesive combination ensured a seamless experience, enabling users to interact with the application effortlessly.

Bhairav Ganguly

(Name of the Student)

Signature of the Lab Coordinator

(Name of the Coordinator)