**Document-OCR**

Start: Start of the algorithm.

Input: Take the file path of the PDF as input.

Perform OCR:

Use a Java library (such as Tesseract OCR or Apache PDFBox) to perform OCR on the PDF file.

Extract text from the PDF and store it in a string variable.

Save as Word Document:

Use a Java library (such as Apache POI) to create a new Word document (.docx).

Write the extracted text to the Word document.

Save the Word document.

End: End of the algorithm.

Algorithm-

import org.apache.pdfbox.pdmodel.PDDocument;

import org.apache.pdfbox.text.PDFTextStripper;

import org.apache.poi.xwpf.usermodel.XWPFDocument;

import org.apache.poi.xwpf.usermodel.XWPFParagraph;

import org.apache.poi.xwpf.usermodel.XWPFRun;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class PDFToWordConverter {

public static void main(String[] args) {

String inputPdfFilePath = "ocr.pdf";

String outputWordFilePath = "output.docx";

try {

String extractedText = performOCR(inputPdfFilePath);

saveToWordDocument(extractedText, outputWordFilePath);

System.out.println("Text extracted from PDF and saved to Word document successfully!");

} catch (IOException e) {

e.printStackTrace();

}

}

private static String performOCR(String inputPdfFilePath) throws IOException {

try (PDDocument document = PDDocument.load(new File(inputPdfFilePath))) {

PDFTextStripper pdfStripper = new PDFTextStripper();

return pdfStripper.getText(document);

}

}

private static void saveToWordDocument(String extractedText, String outputWordFilePath) throws IOException {

try (XWPFDocument document = new XWPFDocument()) {

XWPFParagraph paragraph = document.createParagraph();

XWPFRun run = paragraph.createRun();

run.setText(extractedText);

try (FileOutputStream out = new FileOutputStream(new File(outputWordFilePath))) {

document.write(out);

}

}

}

}

//Bonus Part

import React, { useState } from 'react';

import axios from 'axios';

function OCRTranslatorApp() {

const [selectedFile, setSelectedFile] = useState(null);

const [ocrText, setOcrText] = useState('');

const [translatedText, setTranslatedText] = useState('');

const [loadingOCR, setLoadingOCR] = useState(false);

const [loadingTranslate, setLoadingTranslate] = useState(false);

const handleFileChange = (event) => {

setSelectedFile(event.target.files[0]);

};

const handleOCR = async () => {

setLoadingOCR(true);

const formData = new FormData();

formData.append('pdf', selectedFile);

try {

const response = await axios.post('/perform-ocr', formData, {

headers: {

'Content-Type': 'multipart/form-data',

},

});

setOcrText(response.data);

// Reset translated text

setTranslatedText('');

} catch (error) {

console.error('Error performing OCR:', error);

} finally {

setLoadingOCR(false);

}

};

const handleTranslate = async () => {

setLoadingTranslate(true);

// Assuming you have a translation API endpoint

const response = await axios.post('/translate', { text: ocrText });

setTranslatedText(response.data);

setLoadingTranslate(false);

};

return (

<div>

<h1>OCR & Translator App</h1>

<div>

<input type="file" accept=".pdf" onChange={handleFileChange} />

<button onClick={handleOCR} disabled={!selectedFile || loadingOCR}>

Perform OCR

</button>

{loadingOCR && <p>Loading OCR...</p>}

</div>

{ocrText && (

<div>

<h2>Original Text (OCR)</h2>

<p>{ocrText}</p>

<button onClick={handleTranslate} disabled={loadingTranslate}>

Translate

</button>

{loadingTranslate && <p>Loading Translation...</p>}

</div>

)}

{translatedText && (

<div>

<h2>Translated Text</h2>

<p>{translatedText}</p>

</div>

)}

</div>

);

}

export default OCRTranslatorApp;