// Developed by Manjistha Bidkar

// Performs the following :

// - Initial scan to detect whether text is typed or handwritten

// - Final optimized OCR pass with tuned config for detected type

import Tesseract from 'tesseract.js';

import path from 'path';

import { PreprocessMode } from './preprocess';

// Enum to tag OCR content type

export enum OCRMode {

  TYPED = 'TYPED',

  HANDWRITTEN = 'HANDWRITTEN'

}

// Heuristic: check if likely typed

function isProbablyTyped(text: string): boolean {

  const lines = text.split('\n').filter(line => line.trim() !== '');

  const avgLineLength = lines.reduce((sum, l) => sum + l.length, 0) / (lines.length || 1);

  const punctuationCount = (text.match(/[.,;:!?]/g) || []).length;

  return avgLineLength > 40 && punctuationCount > 5;

}

export async function extractTextFromImage(imagePath: string): Promise<{ text: string, mode: PreprocessMode }> {

  const langPath = path.join(\_\_dirname, '../../tessdata\_best');

  console.log(`[OCR] Running initial scan to detect content type...`);

  // Light scan using generic config

  const lightScan = await Tesseract.recognize(imagePath, 'eng', {

    langPath,

    logger: (m: { status: string; progress: number }) => {

      if (m.status) console.log(`[Initial]: ${m.status} - ${Math.round((m.progress || 0) \* 100)}%`);

    },

    tessedit\_pageseg\_mode: '11',

    preserve\_interword\_spaces: '1'

  } as any);

  const initialText = lightScan.data.text;

  // Use heuristic to determine text type

  const isTyped = isProbablyTyped(initialText);

  const mode = isTyped ? PreprocessMode.TYPED : PreprocessMode.HANDWRITTEN;

  console.log(`[OCR] Detected mode: ${mode}`);

  // Configure Tesseract settings for final scan

  const finalConfig: Record<string, string> = {

    tessedit\_pageseg\_mode: isTyped ? '6' : '11',

    preserve\_interword\_spaces: '1'

  };

  // Restrict character set for typed content

  if (isTyped) {

    finalConfig.tessedit\_char\_whitelist =

      'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.,:;-()[]{}!?\'" ';

  }

  // Final OCR scan with tuned settings

  const finalScan = await Tesseract.recognize(imagePath, 'eng', {

    langPath,

    logger: (m: { status: string; progress: number }) => {

      if (m.status) console.log(`[Final OCR]: ${m.status} - ${Math.round((m.progress || 0) \* 100)}%`);

    },

    ...(finalConfig as any)

  });

  return { text: finalScan.data.text.trim(), mode };

}

// src/utils/preprocess.ts

// Dynamically adjusts sharp filters based on whether text is typed or handwritten to improve OCR results

import sharp from 'sharp';

export enum PreprocessMode {

  TYPED = 'TYPED',

  HANDWRITTEN = 'HANDWRITTEN'

}

export async function preprocessImage(

  inputPath: string,

  outputPath: string,

  mode: PreprocessMode

) {

  const image = sharp(inputPath).grayscale();

  // For handwritten text: enhance contrast, sharpen edges, binarize, enlarge

  if (mode === PreprocessMode.HANDWRITTEN) {

    await image

      .normalize()

      .sharpen({ sigma: 1 })

      .threshold(140)

      .resize({ width: 1800 })

      .jpeg({ quality: 100 })

      .toFile(outputPath);

  } else {

    // For typed text: just resize and save

    await image

      .resize({ width: 1500 })

      .jpeg({ quality: 100 })

      .toFile(outputPath);

  }

}

// Developed by Manjistha Bidkar

// This Express server accepts image uploads, detects whether the content is typed or handwritten,

// preprocesses the image accordingly, extracts text using Tesseract OCR, and returns cleaned text for topic matching.

import express from 'express';

import multer from 'multer';

import path from 'path';

import fs from 'fs';

import { preprocessImage, PreprocessMode } from './utils/preprocess';

import { extractTextFromImage } from './utils/ocr';

const app = express();

const PORT = 3000;

// Directory to store uploaded and processed images

const IMAGE\_DIR = path.join(\_\_dirname, '../images');

if (!fs.existsSync(IMAGE\_DIR)) fs.mkdirSync(IMAGE\_DIR);

// Multer setup for file upload

const upload = multer({ dest: IMAGE\_DIR });

// Text cleanup function

function cleanText(text: string): string {

  return text

    .replace(/[^\x20-\x7E\n]/g, '')     // Remove non-ASCII

    .replace(/[^\w\s\n]/g, '')          // Remove punctuation

    .replace(/\b\w{1,2}\b/g, '')        // Remove very short words like "j", "oo" if needed

    .replace(/\s{2,}/g, ' ')            // Collapse extra spaces

    .replace(/\n+/g, ' ')               // Remove all newlines

    .toLowerCase()

    .trim();

  }

app.post('/extract-text', upload.single('image'), async (req, res) => {

  try {

    if (!req.file) {

      res.status(400).send('No file uploaded');

      return;

    }

    const originalPath = req.file.path;

    const processedPath = path.join(IMAGE\_DIR, `processed-${Date.now()}.jpg`);

    // Step 1: Initial scan to detect mode

    const { text: initialText, mode } = await extractTextFromImage(originalPath);

    // Step 2: Preprocess based on mode

    await preprocessImage(originalPath, processedPath, mode);

    // Step 3: Final OCR

    const { text: finalText } = await extractTextFromImage(processedPath);

    // Step 4: Clean text

    const cleanedText = cleanText(finalText);

    // Step 5: Cleanup

    fs.unlinkSync(originalPath);

    fs.unlinkSync(processedPath);

    res.json({ text: cleanedText });

  } catch (err) {

    console.error('OCR error:', err);

    res.status(500).send('Error processing image');

  }

});

app.listen(PORT, () => {

  console.log(`Server running at http://localhost:${PORT}`);

});

{

  "compilerOptions": {

    "target": "ES2020",

    "module": "CommonJS",

    "esModuleInterop": true,

    "strict": true,

    "outDir": "dist",

    "baseUrl": "./",

    "resolveJsonModule": true

  },

  "include": ["src"]

}

// src/utils/topicMatcher.ts

// Developed by Manjistha Bidkar

import \* as XLSX from 'xlsx';

import Fuse from 'fuse.js';

import { cleanText } from './textCleaner';

export function loadConceptsFromExcel(filePath: string): string[] {

  const workbook = XLSX.readFile(filePath);

  const sheet = workbook.Sheets[workbook.SheetNames[0]];

  const data = XLSX.utils.sheet\_to\_json<{ Concept: string }>(sheet);

  return data

    .map(row => row.Concept?.trim().toLowerCase())

    .filter((v): v is string => !!v);

}

export function identifyConcepts(text: string, concepts: string[]): string[] {

  const cleanedText = cleanText(text).toLowerCase();

  const matched = new Set<string>();

  for (const concept of concepts) {

    if (cleanedText.includes(concept)) {

      matched.add(concept);

    }

  }

  const remaining = concepts.filter(c => !matched.has(c));

  const fuse = new Fuse(remaining, {

    includeScore: true,

    threshold: 0.03,

    minMatchCharLength: 4,

  });

  const words = cleanedText.split(/\s+/);

  const phrases: string[] = [...words]; // add single words

  for (let i = 0; i < words.length - 1; i++) {

    phrases.push(`${words[i]} ${words[i + 1]}`);

    if (i < words.length - 2) {

      phrases.push(`${words[i]} ${words[i + 1]} ${words[i + 2]}`);

    }

  }

  for (const phrase of phrases) {

    const results = fuse.search(phrase);

    for (const result of results) {

      if (result.score !== undefined && result.score < 0.03) {

        matched.add(result.item);

      }

    }

  }

  return Array.from(matched);

}

// src/utils/textCleaner.ts

// Developed by Manjistha Bidkar

/\*\*

 \* Cleans the input OCR text by:

 \* - Removing non-ASCII characters

 \* - Removing punctuation

 \* - Removing very short words (1–2 letters)

 \* - Collapsing extra whitespace and newlines

 \* - Converting to lowercase

 \*/

export function cleanText(text: string): string {

  return text

    .replace(/[^\x20-\x7E\n]/g, '')     // Remove non-ASCII

    .replace(/[^\w\s\n]/g, '')          // Remove punctuation

    .replace(/\b\w{1,2}\b/g, '')        // Remove very short words

    .replace(/\s{2,}/g, ' ')            // Collapse extra spaces

    .replace(/\n+/g, ' ')               // Remove newlines

    .toLowerCase()

    .trim();

}