// server/index.js

// Run: cd server && npm install && OPENAI\_API\_KEY=sk-... node index.js

const express = require('express');

const axios = require('axios');

const cheerio = require('cheerio');

const cors = require('cors');

const helmet = require('helmet');

const tls = require('tls');

const dns = require('dns').promises;

const { Configuration, OpenAIApi } = require('openai');

const puppeteer = require('puppeteer');

const app = express();

app.use(cors());

app.use(express.json({ limit: '5mb' }));

app.use(helmet());

const OPENAI\_KEY = process.env.OPENAI\_API\_KEY || null;

let openaiClient = null;

if (OPENAI\_KEY) {

  const cfg = new Configuration({ apiKey: OPENAI\_KEY });

  openaiClient = new OpenAIApi(cfg);

}

function normalizeUrl(u) {

  if (!u) return null;

  u = u.trim();

  if (!/^https?:\/\//i.test(u)) u = 'https://' + u;

  try { new URL(u); return u; } catch (e) { return null; }

}

async function getSslInfo(hostname, port = 443) {

  return new Promise((resolve) => {

    const socket = tls.connect(port, hostname, { servername: hostname, rejectUnauthorized: false }, () => {

      const cert = socket.getPeerCertificate(true);

      if (!cert || Object.keys(cert).length === 0) {

        resolve({ valid: false });

        socket.end();

        return;

      }

      const now = new Date();

      const validFrom = new Date(cert.valid\_from);

      const validTo = new Date(cert.valid\_to);

      const valid = now >= validFrom && now <= validTo;

      resolve({ valid, validFrom: cert.valid\_from, validTo: cert.valid\_to, subject: cert.subject, issuer: cert.issuer });

      socket.end();

    });

    socket.setTimeout(5000, () => {

      resolve({ valid: false, error: 'timeout' });

      socket.destroy();

    });

    socket.on('error', (e) => {

      resolve({ valid: false, error: e.message });

    });

  });

}

function simpleLocalSummary(text, maxSentences = 3) {

  // extremely simple summarizer: take longest sentences

  const sents = text.match(/[^.!?]+[.!?]?/g) || [];

  const scored = sents.map(s => ({ s: s.trim(), score: s.length }));

  scored.sort((a,b) => b.score - a.score);

  return scored.slice(0, maxSentences).map(x => x.s).join(' ');

}

function computeSeoScore({ title, description, keywords, htmlLength }) {

  // naive scoring out of ~100

  let score = 0;

  if (title) {

    const ideal = 60;

    const diff = Math.abs((title.length || 0) - ideal);

    score += Math.max(0, Math.min(30, 30 - diff)); // up to 30

  }

  if (description) score += 30;

  if (keywords && keywords.length > 0) score += 10;

  if (htmlLength && htmlLength < 500000) score += 10;

  score = Math.round(score);

  return score;

}

app.get('/api/analyze', async (req, res) => {

  try {

    const rawUrl = req.query.url;

    const url = normalizeUrl(rawUrl);

    if (!url) return res.status(400).json({ error: 'Invalid URL' });

    const start = Date.now();

    // DNS lookup

    let ip = null;

    try {

      const hostname = new URL(url).hostname;

      const lookup = await dns.lookup(hostname);

      ip = lookup.address;

    } catch (e) {

      // ignore

    }

    // SSL info

    const isHttps = url.startsWith('https://');

    let ssl = null;

    if (isHttps) {

      try {

        const hostname = new URL(url).hostname;

        ssl = await getSslInfo(hostname);

      } catch (e) { ssl = { valid: false, error: e.message }; }

    }

    // Fetch HTML

    const axiosCfg = { method: 'get', url, responseType: 'text', timeout: 20000, headers: { 'User-Agent': 'WebDetector/1.0 (+https://example.com)' } };

    const resp = await axios(axiosCfg);

    const timing = Date.now() - start;

    const html = resp.data || '';

    const htmlLength = Buffer.byteLength(html, 'utf8');

    // parse

    const $ = cheerio.load(html);

    const title = ($('title').text() || '').trim();

    const description = $('meta[name="description"]').attr('content') || $('meta[property="og:description"]').attr('content') || null;

    const keywords = $('meta[name="keywords"]').attr('content') || null;

    // favicon

    let favicon = null;

    const relIcons = [ 'link[rel="icon"]', 'link[rel="shortcut icon"]', 'link[rel="apple-touch-icon"]' ];

    for (const sel of relIcons) {

      const href = $(sel).attr('href');

      if (href) { favicon = new URL(href, url).href; break; }

    }

    if (!favicon) {

      favicon = new URL('/favicon.ico', url).href;

    }

    // mixed content detection

    let mixedContent = false;

    if (isHttps) {

      $('img[src], script[src], link[href]').each((i, el) => {

        const src = $(el).attr('src') || $(el).attr('href');

        if (src && /^http:\/\//i.test(src)) mixedContent = true;

      });

    }

    // seo score

    const seoScore = computeSeoScore({ title, description, keywords, htmlLength });

    // local summary

    const textForSummary = $('body').text().replace(/\s+/g, ' ').slice(0, 20000);

    let summary = simpleLocalSummary(textForSummary);

    // optional OpenAI summary

    if (openaiClient) {

      try {

        const prompt = `Summarize the following web page content in 3 short sentences:\n\n${textForSummary}`;

        const respOA = await openaiClient.createChatCompletion({

          model: 'gpt-4o-mini',

          messages: [{ role: 'user', content: prompt }],

          max\_tokens: 200

        });

        if (respOA && respOA.data && respOA.data.choices && respOA.data.choices[0]) {

          summary = respOA.data.choices[0].message.content.trim();

        }

      } catch (e) {

        console.error('OpenAI summarization failed', e.message);

      }

    }

    // screenshot (puppeteer)

    let screenshot = null;

    try {

      const browser = await puppeteer.launch({ args: ['--no-sandbox', '--disable-setuid-sandbox'] });

      const page = await browser.newPage();

      await page.setViewport({ width: 1280, height: 800 });

      await page.goto(url, { waitUntil: 'networkidle2', timeout: 30000 });

      const buffer = await page.screenshot({ fullPage: false });

      screenshot = 'data:image/png;base64,' + buffer.toString('base64');

      await browser.close();

    } catch (e) {

      console.error('screenshot failed', e.message);

    }

    const out = {

      url,

      status: resp.status,

      timing,

      sizeBytes: htmlLength,

      title,

      description,

      keywords,

      favicon,

      ip,

      ssl,

      mixedContent,

      seoScore,

      summary,

      screenshot,

      headers: resp.headers

    };

    res.json(out);

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: err.message || 'internal error' });

  }

});

const PORT = process.env.PORT || 5173;

app.listen(PORT, () => console.log(`WebDetector server running on port ${PORT}`));