GlowBot GPT Module Code Archive

# Beginner Skincare Kit Generator (generateBeginnerKit.js)

// 📁 gpt/generateBeginnerKit.js  
module.exports = async function generateBeginnerKit(openai) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare educator introducing beginners to affordable, effective routines." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce a beginner skincare kit. Keep it under 30 words.`),  
 gptPrompt(`List exactly 5 essential skincare products that belong in a beginner’s kit. Use this format:\n\n1. Gentle Cleanser – Removes dirt and oil without drying.`),  
 gptPrompt(`In 1 sentence, encourage beginners to stick with their routine and stay consistent. Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "beginnerKit",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Demo Script Generator (generateDemoScript.js)

// 📁 gpt/generateDemoScript.js  
module.exports = async function generateDemoScript(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a creative, engaging skincare content writer focused on UGC scripts." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, write a punchy intro for a demo video featuring "${product}". Keep it under 30 words.`),  
 gptPrompt(`Write a short 100 word demo script for "${product}". It should walk the viewer through how to use it and what it feels like, in a confident and casual tone.`),  
 gptPrompt(`In 1 sentence, write a confident outro that encourages viewers to try "${product}". Keep it under 20 words.`)  
  
 ]);  
  
 return {  
 type: "demoScript",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Drugstore Dupe Generator (generateDrugstoreDupe.js)

// 📁 gpt/generateDrugstoreDupe.js  
module.exports = async function generateDrugstoreDupe(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare expert revealing affordable alternatives to luxury products." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce a great drugstore dupe for a high-end product. Keep it under 30 words.`),  
 gptPrompt(`Describe how "${product}" is a great drugstore dupe for a more expensive product in 110 words or less. Compare the key ingredients, performance, and results.`),  
 gptPrompt(`In 1 sentence, encourage readers to try "${product}" and save money. Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "drugstoreDupe",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Dry Skin Product List Generator (generateDrySkinList.js)

// 📁 gpt/generateDrySkinList.js  
module.exports = async function generateDrySkinList(openai) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare expert known for recommending top-tier product picks for specific skin concerns." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce a list of the top skincare products for dry skin. Keep it under 30 words.`),  
 gptPrompt(`List the top 5 skincare products for dry skin. For each, include the name and a short benefit in this format:   
 1. CeraVe Moisturizing Cream – Deeply hydrating and packed with ceramides.  
 Ensure you provide exactly 5 products.`),  
 gptPrompt(`In 1 sentence, encourage readers to try a product and maintain consistency. Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "drySkinList",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Influencer Caption Generator (generateInfluencerCaption.js)

// 📁 gpt/generateInfluencerCaption.js  
module.exports = async function generateInfluencerCaption(openai, product, tone = "influencer") {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare influencer who writes captions that convert and engage." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce an influencer caption for "${product}". Keep it under 30 words.`),  
 gptPrompt(`Write a short, punchy influencer-style caption for "${product}" in a ${tone} tone. Include 1-2 emojis and a light call to action.`),  
 gptPrompt(`In 1 sentence, write a short outro that encourages followers to try "${product}". Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "influencerCaption",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Full Content Output Generator (generateOriginal (Full Content Output).js)

// 📁 gpt/generateFullOriginal.js  
module.exports = async function generateFullOriginal(openai, product, tone = "engaging") {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare content expert." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.8  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [  
 intro,  
 productDescription,  
 demoScript,  
 problemSolution,  
 personalReview,  
 socialCaptions,  
 hashtagSets,  
 trendingHashtags,  
 outro  
 ] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, write a hook-style intro for "${product}" in a ${tone} tone. Keep it under 30 words.`),  
 gptPrompt(`Write a short 100 word and helpful product description for "${product}" in an ${tone} tone.`),  
 gptPrompt(`Create a 2-3 sentence skincare demo script for "${product}" that feels casual and confident.`),  
 gptPrompt(`Describe a common skincare problem and how "${product}" solves it. Keep it under 4 sentences.`),  
 gptPrompt(`Write a 100 word personal-sounding review of "${product}" by someone who used it daily for 2 weeks.`),  
 gptPrompt(`Give me 3 social media captions for a post about "${product}". Include 1-2 emojis in each.`),  
 gptPrompt(`Give me 3 sets of hashtags for skincare posts. Each set should include 5 hashtags relevant to "${product}".`),  
 gptPrompt(`Give me 5 trending skincare hashtags for 2024.`),  
 gptPrompt(`In 1 sentence, write a confident outro that encourages people to try "${product}". Keep it under 20 words.`)  
  
 ]);  
  
 return {  
 type: "original",  
 content: "Full content bundle",  
 intro,  
 productDescription,  
 demoScript,  
 problemSolution,  
 personalReview,  
 socialCaptions,  
 hashtagSets,  
 trendingHashtags,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Personal Review Generator (generatePersonalReview.js)

// 📁 gpt/generatePersonalReview.js  
module.exports = async function generatePersonalReview(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare content creator writing authentic and helpful product reviews." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce your review for "${product}". Keep it under 20 words.`),  
 gptPrompt(`Write a detailed, authentic-sounding 100 word or less personal review of "${product}". Include how long it was used, what results were seen, and how it felt during use.`),  
 gptPrompt(`In 1 sentence, conclude your review and recommend who would benefit from using "${product}". Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "personalReview",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Product Comparison Generator (generateProductComparison.js)

const { enrichPromptWithData } = require('../utils/dataEnricher');  
  
module.exports = async function generateProductComparison(openai, product, scraperData) {  
 async function gptPrompt(prompt, contextData = '') {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 {   
 role: "system",   
 content: "You are a skincare expert who explains product comparisons using verified data and community insights."   
 },  
 {   
 role: "user",   
 content: contextData + '\n\n' + prompt   
 }  
 ],  
 temperature: 0.8  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const enrichedData = await enrichPromptWithData(product, scraperData);  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(  
 `In 1-2 short sentences, introduce a product comparison for "${product}". Keep it under 20 words.`,  
 enrichedData.contextualPrompt  
 ),  
 gptPrompt(  
 `Compare "${product}" to its most discussed alternative based on the provided data. Focus on verified prices, ingredients, and community feedback.`,  
 enrichedData.contextualPrompt  
 ),  
 gptPrompt(  
 `Give a data-backed recommendation about "${product}" based on the price point and community sentiment.`,  
 enrichedData.contextualPrompt  
 )  
 ]);  
  
 return {  
 type: "productComparison",  
 intro,  
 content,  
 outro,  
 metadata: {  
 price: enrichedData.productDetails.price,  
 rating: enrichedData.productDetails.rating,  
 communityInsights: enrichedData.communityInsights.length  
 },  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Pros and Cons Generator (generateProsAndCons.js)

// 📁 gpt/generateProsAndCons.js  
module.exports = async function generateProsAndCons(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a balanced skincare reviewer who helps people make informed decisions." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.8  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce a pros and cons breakdown for "${product}". Keep it under 30 words.`),  
 gptPrompt(`List 3 pros and 3 cons of "${product}". Be concise with 100 words or less, but insightful, and maintain a balanced tone.`),  
 gptPrompt(`In 1 sentence, wrap up the pros and cons and recommend who should try "${product}". Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "prosAndCons",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Routine Example Generator (generateRoutineExample.js)

// 📁 gpt/generateRoutineExample.js  
module.exports = async function generateRoutineExample(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare educator building sample daily routines with product integrations." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce a simple skincare routine with "${product}". Keep it under 30 words.`),  
 gptPrompt(`Create a basic 4-step skincare routine in under 125 words (AM or PM) that includes "${product}". For each step, include the step name and a short note, e.g.,\n\n1. Cleanser – Gently removes dirt and oil.`),  
 gptPrompt(`In 1 sentence, encourage consistency in the routine and remind viewers of long-term benefits. Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "routineExample",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Surprise Me Generator (generateSurpriseMe.js)

// 📁 gpt/generateSurpriseMe.js  
module.exports = async function generateSurpriseMe(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a creative skincare copywriter known for surprising and delighting the audience." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.95  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`Write a playful, mysterious 30 words intro that teases a surprise piece of content about "${product}".`),  
 gptPrompt(`Generate a fun and unexpected 100 word or less piece of skincare content about "${product}". It could be a weird tip, a story, or a bold claim — surprise the reader in a positive way.`),  
 gptPrompt(`End with a quirky or clever 20 word outro that makes the viewer smile or want to share the post.`)  
 ]);  
  
 return {  
 type: "surpriseMe",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# TikTok Breakdown Generator (generateTikTokBreakdown.js)

// 📁 gpt/generateTikTokBreakdown.js  
module.exports = async function generateTikTokBreakdown(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare creator breaking down trending TikTok content with insights and attitude." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.9  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce why "${product}" is trending on TikTok. Keep it under 30 words.`),  
 gptPrompt(`Break down why "${product}" is blowing up on TikTok in 125 words or less. Mention key creators, viral claims, and results people are seeing.`),  
 gptPrompt(`In 1 sentence, encourage viewers to try "${product}" and share their thoughts. Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "tiktokBreakdown",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Top 5 Under $25 Generator (generateTop5Under25.js)

// 📁 gpt/generateTop5Under25.js  
module.exports = async function generateTop5Under25(openai) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a savvy skincare content creator helping people find effective products on a budget." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce a list of skincare products under $25. Keep it under 30 words.`),  
 gptPrompt(`List 5 great skincare products that cost less than $25. For each, include the product name and a short benefit using this format:\n\n1. The Ordinary Niacinamide 10% – Helps reduce oil and visibly smooth skin.`),  
 gptPrompt(`In 1 sentence, motivate readers to buy affordable products. Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "top5Under25",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Trending Products List Generator (generateTrendingProducts.js)

// 📁 gpt/generateTrendingProducts.js  
module.exports = async function generateTrendingProducts(openai) {  
 const prompt = `Give me a list of 5 skincare products that are currently trending on TikTok or Amazon.   
Only return the product names, numbered 1 through 5. No extra text.`;  
  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare market analyst tracking trending beauty products." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.7  
 });  
  
 const text = response.choices[0].message.content.trim();  
 const products = text  
 .split('\n')  
 .map(line => line.replace(/^\d+\.\s\*/, '').trim())  
 .filter(Boolean);  
  
 return products;  
};

# Why I Switched Generator (generateWhyISwitched.js)

// 📁 gpt/generateWhyISwitched.js  
module.exports = async function generateWhyISwitched(openai, product) {  
 async function gptPrompt(prompt) {  
 const response = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [  
 { role: "system", content: "You are a skincare storyteller writing persuasive switch-style content." },  
 { role: "user", content: prompt }  
 ],  
 temperature: 0.85  
 });  
 return response.choices[0].message.content.trim();  
 }  
  
 const [intro, content, outro] = await Promise.all([  
 gptPrompt(`In 1–2 short sentences, introduce a story about switching to "${product}". Keep it under 30 words.`),  
 gptPrompt(`Write a clear 100 word explanation of why the person switched to "${product}". Describe the differences they noticed, the improvements it made, and how it felt.`),  
 gptPrompt(`In 1 sentence, recommend others try "${product}" and explain why. Keep it under 20 words.`)  
 ]);  
  
 return {  
 type: "whyISwitched",  
 intro,  
 content,  
 outro,  
 watermark: "generated-by-glowbot-v1"  
 };  
};

# Main Server Logic (index.js) (index.js)

const express = require('express');  
const OpenAI = require('openai');  
const path = require('path');  
  
// Import content generation modules  
const generateContent = require('./gpt/generateDrySkinList');  
const generateInfluencerCaption = require('./gpt/generateInfluencerCaption');  
const generateProductComparison = require('./gpt/generateProductComparison');  
const generateRoutineExample = require('./gpt/generateRoutineExample');  
const generateProsAndCons = require('./gpt/generateProsAndCons');  
const generateTop5Under25 = require('./gpt/generateTop5Under25');  
const generateWhyISwitched = require('./gpt/generateWhyISwitched');  
const generateBeginnerKit = require('./gpt/generateBeginnerKit');  
const generateDrugstoreDupe = require('./gpt/generateDrugstoreDupe');  
const generateTikTokBreakdown = require('./gpt/generateTikTokBreakdown');  
const generateSurpriseMe = require('./gpt/generateSurpriseMe');  
const generateDemoScript = require('./gpt/generateDemoScript');  
const generatePersonalReview = require('./gpt/generatePersonalReview');  
const generateTrendingProducts = require('./gpt/generateTrendingProducts');  
  
// Import Instagram and Twitter Scrapers  
const { getTikTokTrending } = require('./scrapers/tikTokScraper');  
const { getRedditTrending } = require('./scrapers/redditScraper');  
const { getInstagramTrending } = require('./scrapers/instagramScraper');  
const getGoogleTrends = require('./scrapers/googleTrendsScraper');  
  
  
const app = express();  
app.use(express.json());  
app.use(express.static('public'));  
  
const openai = new OpenAI({ apiKey: process.env.OPENAI\_API\_KEY });  
console.log("OpenAI API Status:", process.env.OPENAI\_API\_KEY ? "✅ API Key Present" : "❌ Missing API Key");  
  
const { getYouTubeTrending } = require('./scrapers/youtubeScraper');  
  
app.get('/youtube-trending', async (req, res) => {  
 const results = await getYouTubeTrending();  
 res.json({ videos: results });  
});  
  
const getAmazonTrending = require('./scrapers/amazonTrendingScraper');  
  
app.get('/amazon-trending', async (req, res) => {  
 const products = await getAmazonTrending();  
 res.json({ products });  
});  
  
// Route to handle dynamic trending products from TikTok, Reddit, Instagram, and Twitter  
app.get('/dynamic-trending', async (req, res) => {  
 try {  
 const [tiktok, reddit, instagram, youtube, google, amazon] = await Promise.all([  
 getTikTokTrending(),  
 getRedditTrending(),  
 getInstagramTrending(),  
 getYouTubeTrending(),  
 getGoogleTrends(),  
 getAmazonTrending()  
 ]);  
  
 console.log('📥 Scraped Reddit Data:', reddit);  
 console.log('📥 Scraped YouTube Data:', youtube);  
 // optional if already present  
  
 const allTrends = [  
 ...tiktok,  
 ...reddit,  
 ...instagram,  
 ...youtube,  
 ...google,  
 ...amazon  
 ];  
  
 const allText = allTrends.map(t =>  
 typeof t === 'string' ? t : t.title || t.caption || t.name || JSON.stringify(t)  
 ).join('\n');  
  
 const prompt = `  
Here is a raw dump of trending skincare content:  
  
${allText}  
  
Your task is to extract \*\*exactly 6\*\* trending skincare products from this list. Format as a JSON array of product objects like this:  
  
[  
 { "title": "Product Name", "link": "https://example.com/product" },  
 ...  
]  
  
✅ Only include real product names (no hashtags or slogans).  
✅ If a link is missing, omit it. Do not guess.  
✅ Keep it clean and concise — no extra text outside the JSON.  
`.trim();  
  
 const completion = await openai.chat.completions.create({  
 model: 'gpt-4',  
 messages: [{ role: 'user', content: prompt }],  
 temperature: 0.5  
 });  
  
 const gptReply = completion.choices?.[0]?.message?.content?.trim() || '[]';  
 const jsonStart = gptReply.indexOf('[');  
 const cleanedProducts = JSON.parse(gptReply.slice(jsonStart));  
  
 res.json({ products: cleanedProducts });  
 } catch (err) {  
 console.error('❌ GPT-cleaned trending error:', err);  
 res.status(500).json({ products: [] });  
 }  
});  
  
  
  
// Route to generate content  
app.post('/generate', async (req, res) => {  
 try {  
 const { product, templateType, tone = "engaging" } = req.body;  
 let result;  
  
 // Fetch relevant data for the product  
 const [amazonData, redditData, instagramData] = await Promise.all([  
 getAmazonTrending(),  
 getRedditTrending(),  
 getInstagramTrending()  
 ]);  
  
 const scraperData = {  
 amazonData: amazonData.find(p => p.title.toLowerCase().includes(product.toLowerCase())) || {},  
 redditData: redditData.filter(p => p.title.toLowerCase().includes(product.toLowerCase())),  
 instagramData: instagramData.filter(p => p.caption?.toLowerCase().includes(product.toLowerCase()))  
 };  
  
 switch (templateType) {  
 case 'original':  
 const generateOriginal = require('./gpt/generateOriginal (Full Content Output)');  
 result = await generateOriginal(openai, product, tone);  
 break;  
 case 'drySkinList':  
 result = await generateContent(openai);  
 break;  
 case 'influencerCaption':  
 result = await generateInfluencerCaption(openai, product);  
 break;  
 case 'productComparison':  
 result = await generateProductComparison(openai, product);  
 break;  
 case 'routineExample':  
 result = await generateRoutineExample(openai, product);  
 break;  
 case 'prosAndCons':  
 result = await generateProsAndCons(openai, product);  
 break;  
 case 'top5Under25':  
 result = await generateTop5Under25(openai);  
 break;  
 case 'whyISwitched':  
 result = await generateWhyISwitched(openai, product);  
 break;  
 case 'beginnerKit':  
 result = await generateBeginnerKit(openai);  
 break;  
 case 'drugstoreDupe':  
 result = await generateDrugstoreDupe(openai, product);  
 break;  
 case 'tiktokBreakdown':  
 result = await generateTikTokBreakdown(openai, product);  
 break;  
 case 'surpriseMe':  
 result = await generateSurpriseMe(openai, product);  
 break;  
 case 'demoScript':  
 result = await generateDemoScript(openai, product);  
 break;  
 case 'personalReview':  
 result = await generatePersonalReview(openai, product);  
 break;  
 default:  
 return res.status(400).json({ error: 'Invalid template type' });  
 }  
  
 res.json({ result });  
 } catch (err) {  
 console.error('Generation error:', err);  
 res.status(500).json({ error: err.message });  
 }  
});  
  
// Serve main HTML file  
app.get('/', (req, res) => {  
 res.sendFile(path.join(\_\_dirname, 'public/index.html'));  
});  
  
  
  
app.get('/trend-digest', async (req, res) => {  
 try {  
 const [tiktok, reddit, instagram, youtube, google, amazon] = await Promise.all([  
 getTikTokTrending(),  
 getRedditTrending(),  
 getInstagramTrending(),  
 getYouTubeTrending(),  
 getGoogleTrends(),  
 getAmazonTrending()  
 ]);  
  
 const combinedTrends = [  
 ...tiktok.map(item => `- TikTok: ${item}`),  
 ...reddit.map(item => `- Reddit: ${item}`),  
 ...instagram.map(item => `- Instagram: ${item.title || item.caption}`),  
 ...youtube.map(item => `- YouTube: ${item.title}`),  
 ...google.map(item => `- Google: ${item.title}`),  
 ...amazon.map(item => `- Amazon: ${item.title}`)  
 ]  
 .slice(0, 15)  
 .join('\n');  
  
 const prompt = `  
You are an AI trained to generate content ideas for skincare creators based on social media trends.  
  
Here are recent trending topics:  
${combinedTrends}  
  
Please return the following as a JSON object with three fields:  
  
{  
 "viralHooks": ["idea 1", "idea 2", "idea 3"],  
 "videoScript": ["line 1", "line 2", "line 3"],   
 "creatorInsight": "short final insight"  
}  
  
1. Generate 3 separate Viral Hook Ideas for TikTok or YouTube Shorts based on current skincare trends.  
2. Create a short 150-word video script outline an AI could read as a voiceover.  
3. Provide a final takeaway or insight for creators.  
  
Respond with only the raw JSON object.`.trim();  
  
 const completion = await openai.chat.completions.create({  
 model: "gpt-4",  
 messages: [{ role: "user", content: prompt }],  
 temperature: 0.7  
 });  
  
 const raw = completion.choices?.[0]?.message?.content?.trim();  
 const jsonStart = raw.indexOf('{');  
 const json = raw.slice(jsonStart);  
 const parsed = JSON.parse(json);  
  
 res.json(parsed);  
 } catch (err) {  
 console.error("Trend digest error:", err);  
 res.json({  
 viralHooks: [  
 "💡 3 Skincare Mistakes You're Still Making",  
 "Why This $12 Serum is Outselling Big Brands",  
 "Glow Up in 30 Seconds a Day"  
 ],  
 videoScript: [  
 "Today, we're fixing 3 common skincare mistakes — fast.",  
 "From product layering to SPF fails, here's what you're missing.",  
 "And wait till you see the glow from this budget serum."  
 ],  
 creatorInsight: "Trend-forward content works, but credibility wins. Pair viral hooks with helpful insights to grow faster."  
 });  
 }  
});  
  
  
app.get('/scraper-health', async (req, res) => {  
 const statuses = {};  
  
 async function checkSource(label, getDataFn) {  
 const start = Date.now();  
 try {  
 const data = await getDataFn();  
 const time = Date.now() - start;  
 const isAI = data?.some(item => item.caption?.includes('Simulated') || item.caption?.includes('AI'));  
 const count = data?.length || 0;  
  
 console.log(`${label} Scraper Output (${count} items, ${time}ms):`, data);  
  
 if (!count) return `⚠️ No Data`;  
  
 const sourceType = isAI ? 'AI Generated' : 'Active';  
 return `✅ ${sourceType} (${count} items, ${time}ms)`;  
 } catch (err) {  
 console.error(`❌ ${label} health check error:`, err);  
 return '❌ Error';  
 }  
 }  
  
 async function getSourceStatus(label, getDataFn) {  
 const start = Date.now();  
 try {  
 const data = await getDataFn();  
 const time = Date.now() - start;  
 const count = data?.length || 0;  
 const isAI =   
 label === '📸 Instagram' ||   
 (data?.some && data.some(item =>   
 item.caption?.includes('AI') ||   
 item.link === '#' ||   
 item.title?.includes('Fallback')  
 ));  
  
 if (!count) return '⚠️ No Data';  
 return `✅ ${isAI ? 'AI Generated' : 'Active'} (${count} items, ${time}ms)`;  
 } catch (err) {  
 console.error(`❌ ${label} health check error:`, err);  
 return '❌ Error';  
 }  
 }  
  
 statuses.tiktok = await getSourceStatus('🎵 TikTok', getTikTokTrending);  
 statuses.instagram = await getSourceStatus('📸 Instagram', getInstagramTrending);  
 statuses.reddit = await getSourceStatus('🔴 Reddit', getRedditTrending);  
 statuses.google = await getSourceStatus('🟢 Google Trends', require('./scrapers/googleTrendsScraper'));  
 statuses.amazon = await getSourceStatus('🟠 Amazon', require('./scrapers/amazonTrendingScraper'));  
 statuses.youtube = await getSourceStatus('🔵 YouTube', require('./scrapers/youtubeScraper').getYouTubeTrending);  
  
 // YouTube (with caching)  
 try {  
 if (!global.youtubeHealthCache || Date.now() - global.youtubeHealthLastCheck > 5 \* 60 \* 1000) {  
 const start = Date.now();  
 const { getYouTubeTrending } = require('./scrapers/youtubeScraper');  
 const data = await getYouTubeTrending();  
 const time = Date.now() - start;  
 const isAI = data?.some(item => item.caption?.includes('Simulated') || item.caption?.includes('AI'));  
 const count = data?.length || 0;  
  
 console.log(`🔵 YouTube Scraper Output (${count} items, ${time}ms):`, data);  
  
 global.youtubeHealthCache = count > 0  
 ? `✅ ${isAI ? 'AI Generated' : 'Active'} (${count} items, ${time}ms)`  
 : '⚠️ No Data';  
 global.youtubeHealthLastCheck = Date.now();  
 }  
  
 statuses.youtube = global.youtubeHealthCache;  
 } catch (err) {  
 console.error('❌ YouTube health check error:', err);  
 statuses.youtube = '❌ Error';  
 }  
  
 res.json(statuses);  
});  
  
  
  
// Start server  
const PORT = process.env.PORT || 3000;  
  
  
app.listen(PORT, '0.0.0.0', () => {  
 const used = process.memoryUsage();  
 console.log(`🚀 Server running on port ${PORT}`);  
 console.log('📊 Memory Usage:', {  
 heapTotal: `${Math.round(used.heapTotal / 1024 / 1024 \* 100) / 100} MB`,  
 heapUsed: `${Math.round(used.heapUsed / 1024 / 1024 \* 100) / 100} MB`,  
 rss: `${Math.round(used.rss / 1024 / 1024 \* 100) / 100} MB`,  
 });  
});  
  
const promptFactory = require('./utils/promptFactory'); // Add near your other imports  
  
// Route to handle batch generation of trending content  
app.post('/generate-all', async (req, res) => {  
 try {  
 const tone = req.body.tone || '';  
 const [tiktok, reddit, instagram] = await Promise.all([  
 getTikTokTrending(),  
 getRedditTrending(),  
 getInstagramTrending(),  
 ]);  
  
 const trends = [...tiktok, ...reddit, ...instagram];  
 const prompts = trends.map(trend => promptFactory(trend, tone));  
  
 const results = [];  
  
 for (const p of prompts) {  
 const response = await fetch(`http://localhost:${PORT}/generate`, {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 product: p.product,  
 templateType: p.templateType,  
 tone: p.tone,  
 }),  
 });  
  
 const result = await response.json();  
 results.push({ trend: p.product, result });  
 }  
  
 res.json({ results });  
 } catch (err) {  
 console.error('❌ Batch generation failed:', err);  
 res.status(500).json({ error: 'Batch generation failed' });  
 }  
});  
  
console.log('✅ Server fully initialized');

# Project Configuration (package.json) (package.json)

{  
 "name": "nodejs",  
 "version": "1.0.0",  
 "description": "",  
 "main": "index.js",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "keywords": [],  
 "author": "",  
 "license": "ISC",  
 "dependencies": {  
 "@types/node": "^22.13.11",  
 "autoprefixer": "^10.4.21",  
 "axios": "^1.9.0",  
 "body-parser": "^2.2.0",  
 "cheerio": "^1.0.0",  
 "compression": "^1.8.0",  
 "daisyui": "^5.0.35",  
 "dotenv": "^16.5.0",  
 "express": "^5.1.0",  
 "express-rate-limit": "^7.5.0",  
 "google-trends-api": "^4.9.2",  
 "helmet": "^8.1.0",  
 "openai": "^4.97.0",  
 "postcss": "^8.5.3",  
 "puppeteer": "^24.8.2",  
 "snoowrap": "^1.23.0",  
 "tailwindcss": "^4.1.5",  
 "twitter": "^1.1.0",  
 "undici": "^7.8.0",  
 "xml2js": "^0.6.2"  
 }  
}

# Frontend Logic (script.js) (script.js)

const trendingContainer = document.getElementById('trending-products');  
const trendDigestBox = document.getElementById('trend-digest-box');  
  
// Call loadTrendingProducts when the page loads  
document.addEventListener('DOMContentLoaded', () => {  
 loadTrendingProducts();  
 loadTrendDigest();  
 updateScraperHealth(); // Call once on page load  
});  
  
console.log('🔥 Fetching from /dynamic-trending...');  
function loadTrendingProducts() {  
 trendingContainer.innerHTML = `  
 <div class="bg-white p-4 rounded-lg shadow-sm w-full">  
 <div class="animate-pulse">  
 <p class="text-lg">⏳ Loading trending products...</p>  
 <p class="text-sm text-base-content/70">This may take a few seconds</p>  
 </div>  
 </div>`;  
  
 fetch('/dynamic-trending')  
 .then(res => res.json())  
 .then(data => {  
 console.log('🧪 Trending data:', data.products);  
 trendingContainer.innerHTML = '';  
  
 (data.products || []).forEach(product => {  
 let title;  
  
 // Handle string format and object format  
 if (typeof product === 'string') {  
 title = product;  
 } else if (typeof product === 'object') {  
 title = product.title || product.caption || product.name || product.product || null;  
 }  
  
 if (!title) {  
 console.warn('⚠️ Skipped invalid product:', product);  
 return;  
 }  
  
 const element = document.createElement('button');  
 element.className = 'px-4 py-2 rounded-full bg-blue-50 text-blue-800 hover:bg-blue-100 transition-colors text-sm whitespace-nowrap m-1';  
  
 element.onclick = () => {  
 document.getElementById('product-input').value = title;  
 window.scrollTo({ top: document.getElementById('product-form').offsetTop, behavior: 'smooth' });  
 };  
  
 element.textContent = title;  
 trendingContainer.appendChild(element);  
 });  
  
 loadTrendDigest();  
 })  
 .catch(err => {  
 console.error('❌ Failed to load trending products:', err.message || err);  
 trendingContainer.innerHTML = '<div class="alert alert-error">Failed to load trending products</div>';  
 });  
}  
  
async function updateScraperHealth() {  
 const healthContainer = document.getElementById('scraper-health');  
 if (!healthContainer) return;  
  
 try {  
 const res = await fetch('/scraper-health');  
 const statuses = await res.json();  
  
 // Log only on first load  
 if (!window.hasLoggedScraperHealth) {  
 console.log('Scraper Health Status:', statuses);  
 window.hasLoggedScraperHealth = true;  
 }  
  
 healthContainer.innerHTML = Object.entries(statuses)  
 .map(([scraper, status]) => `  
 <div class="flex items-center gap-2 p-2">  
 <span class="font-medium">${scraper}:</span>  
 <span>${status}</span>  
 </div>  
 `).join('');  
 } catch (err) {  
 console.error('❌ Failed to load scraper health:', err);  
 healthContainer.innerHTML = '<div class="alert alert-error">Failed to load scraper health</div>';  
 }  
}  
  
// Corrected scraper health updater  
function updateScraperHealth() {  
 fetch('/scraper-health')  
 .then(res => res.json())  
 .then(data => {  
 console.log('Scraper Health Status:', data);  
 const container = document.getElementById('scraper-health');  
 if (container) {  
 container.innerHTML = `  
 <div class="mt-4 text-left text-sm leading-6">  
 <strong class="block mb-1">🧠 Scraper Health</strong>  
 <ul class="list-disc list-inside text-gray-700 space-y-1">  
 <li>tiktok: ${data.tiktok || 'N/A'}</li>  
 <li>instagram: ${data.instagram || 'N/A'}</li>  
 <li>reddit: ${data.reddit || 'N/A'}</li>  
 <li>google: ${data.google || 'N/A'}</li>  
 <li>youtube: ${data.youtube || 'N/A'}</li>  
 <li>amazon: ${data.amazon || 'N/A'}</li>  
 </ul>  
 </div>  
 `;  
 }  
 })  
 .catch(err => {  
 console.error('❌ Failed to load scraper health:', err);  
 });  
}  
  
// Start checking scraper health  
updateScraperHealth();  
  
async function loadTrendDigest() {  
 if (!trendDigestBox) return;  
 trendDigestBox.innerHTML = "Loading AI Trend Digest...";  
  
 fetch('/trend-digest')  
 .then(res => res.json())  
 .then(data => {  
 if (!data.viralHooks || !data.videoScript || !data.creatorInsight) {  
 trendDigestBox.innerHTML = '<p class="text-gray-500">No trend digest available.</p>';  
 return;  
 }  
  
 trendDigestBox.innerHTML = `  
 <div class="space-y-6">  
 <div class="bg-white p-4 rounded-lg shadow-sm">  
 <h3 class="font-bold text-rose-700 mb-2">🎯 Viral Hook Ideas</h3>  
 <ul class="list-disc pl-5 space-y-1 text-gray-700">  
 ${data.viralHooks.map(item => `<li>${item}</li>`).join('')}  
 </ul>  
 </div>  
 <div class="bg-white p-4 rounded-lg shadow-sm">  
 <h3 class="font-bold text-rose-700 mb-2">📝 Video Script Outline</h3>  
 <ul class="list-disc pl-5 space-y-1 text-gray-700">  
 ${data.videoScript.map(line => `<li>${line}</li>`).join('')}  
 </ul>  
 </div>  
 <div class="bg-white p-4 rounded-lg shadow-sm">  
 <h3 class="font-bold text-rose-700 mb-2">💡 Creator Insight</h3>  
 <p class="text-gray-700">${data.creatorInsight}</p>  
 </div>  
 </div>  
 `;  
 })  
 .catch((err) => {  
 console.error('Trend Digest Error:', err);  
 trendDigestBox.innerHTML = "❌ Failed to load AI Trend Digest. Check console for details.";  
 });  
}  
  
  
const form = document.getElementById('product-form');  
const resultsContainer = document.getElementById('results-container');  
const resultsDiv = document.getElementById('results');  
const productInput = document.getElementById('product-input');  
  
// Character Counter  
const updateCharCount = () => {  
 const count = productInput.value.length;  
 const maxLength = 100;  
 productInput.parentElement.querySelector('.char-count').innerHTML = `${count}/${maxLength} characters`;  
};  
const charCounter = document.createElement('span');  
charCounter.className = 'char-count text-sm text-base-content/70 mt-1';  
productInput.parentElement.appendChild(charCounter);  
updateCharCount();  
productInput.addEventListener('input', updateCharCount);  
  
// Keyboard Shortcut: Ctrl+Enter or Cmd+Enter  
document.addEventListener('keydown', (e) => {  
 if ((e.ctrlKey || e.metaKey) && e.key === 'Enter') {  
 if (!form.querySelector('button[type="submit"]').disabled) {  
 form.dispatchEvent(new Event('submit'));  
 }  
 }  
});  
  
// Submission Handler  
form.addEventListener('submit', async (e) => {  
 e.preventDefault();  
 const submitBtn = form.querySelector('button[type="submit"]');  
 const inputs = form.querySelectorAll('input, select');  
  
 submitBtn.disabled = true;  
 inputs.forEach(input => input.disabled = true);  
 submitBtn.innerHTML = '<span class="loading loading-spinner"></span> Generating...';  
  
 resultsDiv.innerHTML = `  
 <div class="animate-pulse">  
 <p class="text-lg">⏳ Crafting your content...</p>  
 <p class="text-sm text-base-content/70">This may take a few seconds</p>  
 </div>  
 `;  
 resultsContainer.style.display = 'block';  
 resultsContainer.scrollIntoView({ behavior: 'smooth' });  
  
 const product = document.getElementById('product-input').value;  
 const affiliate = document.getElementById('affiliate-input').value;  
 const tone = document.getElementById('tone-select').value;  
 const templateSelect = document.getElementById('template-select');  
 let templateType = templateSelect.value;  
  
 const templateMap = {  
 "comparison": "productComparison",  
 "prosCons": "prosAndCons",  
 "random": "surpriseMe",  
 "whySwitched": "whyISwitched",  
 "drugstoreDupes": "drugstoreDupe"  
 };  
 if (templateMap[templateType]) {  
 templateType = templateMap[templateType];  
 }  
  
 try {  
 const res = await fetch('/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ product, affiliate, tone, templateType }),  
 });  
  
 const data = await res.json();  
 if (!res.ok || !data.result || !data.result.content) {  
 const errorMsg = data.error || 'Invalid response from AI';  
 throw new Error(errorMsg);  
 }  
  
 showToast('success', 'Content generated successfully!');  
 displayResults(data);  
 } catch (err) {  
 console.error('❌ Generation error:', err);  
 showToast('error', 'Something went wrong: ' + err.message);  
 resultsDiv.innerHTML = `<div class="alert alert-error">❌ Error: ${err.message}</div>`;  
 } finally {  
 submitBtn.disabled = false;  
 inputs.forEach(input => input.disabled = false);  
 submitBtn.innerHTML = 'Generate Content';  
 }  
});  
  
// ⏱️ Helper function to estimate video duration  
function estimateVideoDuration(text) {  
 const wordCount = text.trim().split(/\s+/).length;  
 return Math.ceil(wordCount / 2.5); // 2.5 words/sec  
}  
  
// 🔧 Patch displayResults to include estimated time  
function displayResults(data, { append = false } = {}) {  
 const resultsDiv = document.getElementById('results');  
 const wrapper = document.createElement('div');  
 wrapper.id = "content-wrapper";  
 wrapper.className = "space-y-6";  
  
 let totalText = '';  
  
 if (data.result?.type === "original") {  
 const sections = [  
 { id: 'intro', title: '✨ Introduction', content: data.result.intro },  
 { id: 'desc', title: '🧴 Product Description', content: data.result.productDescription },  
 { id: 'demo', title: '🎥 Demo Script', content: data.result.demoScript },  
 { id: 'problem', title: '❓ Problem/Solution', content: data.result.problemSolution },  
 { id: 'review', title: '🗣️ Personal Review', content: data.result.personalReview },  
 { id: 'captions', title: '💬 Social Captions', content: data.result.socialCaptions },  
 { id: 'hashtags', title: '🏷️ Hashtag Sets', content: data.result.hashtagSets },  
 { id: 'trending', title: '📈 Trending Skincare Hashtags', content: data.result.trendingHashtags },  
 { id: 'outro', title: '💡 Conclusion', content: data.result.outro }  
 ];  
  
 sections.forEach(({ id, title, content }) => {  
 if (content) {  
 totalText += content + ' ';  
 const div = document.createElement('div');  
 div.className = 'bg-base-100 rounded-lg p-4 border';  
 div.innerHTML = `  
 <h3 class="font-bold mb-2">${title}</h3>  
 <div id="${id}"><pre class="whitespace-pre-wrap">${content}</pre></div>  
 <div class="flex justify-end mt-2">  
 <button class="copy-btn tooltip" data-tip="Copy section" onclick="copyToClipboard('${id}')">  
 📋 Copy Section  
 </button>  
 </div>  
 `;  
 wrapper.appendChild(div);  
 }  
 });  
 } else {  
 const sections = [  
 { id: 'intro', title: '✨ Introduction', content: data.result.intro },  
 { id: 'content', title: '📝 Content', content: data.result.content },  
 { id: 'outro', title: '💡 Conclusion', content: data.result.outro }  
 ];  
  
 sections.forEach(({ id, title, content }) => {  
 if (content) {  
 totalText += content + ' ';  
 const div = document.createElement('div');  
 div.className = 'bg-base-100 rounded-lg p-4 border';  
 div.innerHTML = `  
 <h3 class="font-bold mb-2">${title}</h3>  
 <div id="${id}"><pre class="whitespace-pre-wrap">${content}</pre></div>  
 <div class="flex justify-end mt-2">  
 <button class="copy-btn tooltip" data-tip="Copy section" onclick="copyToClipboard('${id}')">  
 📋 Copy Section  
 </button>  
 </div>  
 `;  
 wrapper.appendChild(div);  
 }  
 });  
 }  
  
 const estimatedTime = estimateVideoDuration(totalText);  
 const timeDiv = document.createElement('p');  
 timeDiv.className = 'text-center text-sm text-gray-500 italic mt-2';  
 timeDiv.textContent = `🎬 Estimated Video Length: ${estimatedTime} seconds`;  
 wrapper.appendChild(timeDiv);  
  
 wrapper.innerHTML += `  
 <div class="flex justify-center mt-6">  
 <button class="btn btn-primary" onclick="copyAll()">  
 📋 Copy All Content  
 </button>  
 </div>  
 `;  
  
 if (!append) resultsDiv.innerHTML = '';  
 resultsDiv.appendChild(wrapper);  
 document.getElementById('results-container').style.display = 'block';  
}  
  
  
function copyToClipboard(id) {  
 const text = document.getElementById(id).innerText;  
 navigator.clipboard.writeText(text).then(() => {  
 alert(`✅ Copied section: ${id}`);  
 });  
}  
  
function copyAll() {  
 const preTags = document.querySelectorAll('#results pre');  
 let allText = '';  
  
 preTags.forEach(pre => {  
 allText += pre.innerText.trim() + '\n\n';  
 });  
  
 navigator.clipboard.writeText(allText.trim()).then(() => {  
 showToast('success', 'Copied all content!');  
 });  
}  
  
function showToast(type, message) {  
 const toast = document.createElement('div');  
 toast.className = `alert alert-${type} fixed bottom-4 right-4 max-w-sm fade-in z-50`;  
 toast.innerHTML = message;  
 document.body.appendChild(toast);  
  
 setTimeout(() => {  
 toast.style.opacity = '0';  
 setTimeout(() => toast.remove(), 500);  
 }, 3000);  
}  
  
// 🔁 Generate content for all trending products in batch (with progress bar)  
document.getElementById('generate-all-btn').addEventListener('click', async () => {  
 const tone = document.getElementById('tone-select').value;  
 showToast('info', 'Batch generation started...');  
  
 // Show progress UI  
 const progressWrapper = document.getElementById('batch-progress');  
 const progressBar = document.getElementById('progress-bar');  
 const progressCount = document.getElementById('progress-count');  
 const progressTotal = document.getElementById('progress-total');  
  
 progressWrapper.classList.remove('hidden');  
 progressBar.value = 0;  
 progressCount.innerText = '0';  
 progressTotal.innerText = '...';  
  
 try {  
 const res = await fetch('/dynamic-trending');  
 const trendData = await res.json();  
 const trends = trendData.products || [];  
  
 const total = trends.length;  
 let completed = 0;  
 progressTotal.innerText = total;  
  
 const results = [];  
  
 for (const trend of trends) {  
 const trendRes = await fetch('/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 product: trend.title,  
 templateType: detectTemplateFromText(trend.title + ' ' + trend.caption),  
 tone  
 })  
 });  
  
 const result = await trendRes.json();  
 results.push({ trend: trend.title, result });  
  
 completed++;  
 progressCount.innerText = completed;  
 progressBar.value = Math.floor((completed / total) \* 100);  
 }  
  
 showToast('success', 'Batch complete!');  
  
 // Display each result using your normal UI renderer  
 document.getElementById('results').innerHTML = '';  
 results.forEach(r => {  
 if (r.result?.result) {  
 displayResults(r.result, { append: true });  
 }  
 });  
 } catch (err) {  
 console.error('❌ Batch generation error:', err);  
 showToast('error', 'Batch generation failed.');  
 } finally {  
 progressWrapper.classList.add('hidden');  
 }  
});  
  
// 💡 Helper to infer template type from text  
function copyTrendSection(id) {  
 const text = document.getElementById(id).innerText;  
 navigator.clipboard.writeText(text).then(() => {  
 showToast('success', 'Section copied to clipboard!');  
 });  
}  
  
function copyAllTrendSections() {  
 const sections = ['viral-hooks', 'video-script', 'creator-insight'];  
 const allText = sections  
 .map(id => {  
 const el = document.getElementById(id);  
 return el ? el.innerText : '';  
 })  
 .join('\n\n');  
  
 navigator.clipboard.writeText(allText).then(() => {  
 showToast('success', 'All sections copied to clipboard!');  
 });  
}  
  
function detectTemplateFromText(text) {  
 const lower = text.toLowerCase();  
 if (lower.includes('dupe') || lower.includes('alternative')) return 'drugstoreDupe';  
 if (lower.includes('routine')) return 'routineExample';  
 if (lower.includes('top 5') || lower.includes('best')) return 'drySkinList';  
 if (lower.includes('switch') || lower.includes('why i switched')) return 'whyISwitched';  
 if (lower.includes('review')) return 'personalReview';  
 if (lower.includes('compare') || lower.includes('vs')) return 'productComparison';  
 if (lower.includes('demo') || lower.includes('script')) return 'demoScript';  
 if (lower.includes('caption')) return 'influencerCaption';  
 return 'surpriseMe';  
}

# Frontend Layout (index.html) (index.html)

<!DOCTYPE html>  
<html lang="en" data-theme="light">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0"/>  
 <title>GlowBot Skincare Content Generator</title>  
 <link href="styles.css" rel="stylesheet">  
 <script src="https://cdn.tailwindcss.com"></script>  
</head>  
<body class="min-h-screen bg-base-100 p-4 md:p-8">  
 <div class="max-w-4xl mx-auto">  
 <header class="mb-8">  
 <h1 class="text-4xl md:text-5xl font-bold text-center text-rose-700 leading-tight">  
 GlowBot Skincare Content Generator 💸  
 </h1>  
 <p class="text-center text-lg md:text-xl text-gray-700 mt-3">  
 Generate viral-ready product blurbs, captions, and hashtags in seconds. Ideal for TikTok, Reels, and affiliate marketing.  
 </p>  
 </header>  
  
 <section class="text-center mb-8">  
 <h2 class="text-2xl md:text-3xl font-bold text-rose-700 mb-2">🔥 Trending Skincare Products</h2>  
 <div id="trending-products" class="flex flex-wrap justify-center gap-2 mt-4"></div>  
 </section>  
  
 <div class="text-center my-4">  
 <button id="generate-all-btn" class="btn btn-secondary">⚡ Generate Content for All Trending Products</button>  
 </div>  
  
 <section class="text-center mb-8">  
 <h2 class="text-2xl md:text-3xl font-bold text-rose-700 mb-2 flex items-center justify-center gap-2">  
 🧴 Skincare Content Generator  
 </h2>  
 </section>  
  
 <form id="product-form" class="card bg-white p-6 shadow-sm mb-8">  
 <div class="form-control w-full">  
 <label class="label" for="product-input">  
 <span class="label-text font-medium">Amazon Product Name or URL:</span>  
 </label>  
 <input type="text" id="product-input" required placeholder="e.g. CeraVe Moisturizing Cream" class="input input-bordered w-full border-black" />  
 </div>  
  
 <div class="form-control w-full mt-4">  
 <label class="label" for="affiliate-input">  
 <span class="label-text font-medium">Amazon Affiliate Link (optional):</span>  
 </label>  
 <input type="text" id="affiliate-input" placeholder="e.g. https://amzn.to/xyz123" class="input input-bordered w-full border-black" />  
 </div>  
  
 <div class="form-control w-full mt-4">  
 <label class="label" for="tone-select">  
 <span class="label-text font-medium">Content Tone:</span>  
 </label>  
 <select id="tone-select" class="select select-bordered w-full border-black">  
 <option value="">Default</option>  
 <option value="fun">Fun</option>  
 <option value="influencer">Influencer</option>  
 <option value="clinical">Clinical</option>  
 <option value="snarky">Snarky</option>  
 </select>  
 </div>  
  
 <div class="form-control w-full mt-4">  
 <label class="label" for="template-select">  
 <span class="label-text font-medium">Content Template Type:</span>  
 </label>  
 <select id="template-select" class="select select-bordered w-full border-black">  
 <optgroup label="🛒 Amazon Product Focused">  
 <option value="original">Original (Full Content Output)</option>  
 <option value="drySkinList">Top 5 for Skin Type</option>  
 <option value="routineExample">Routine Example</option>  
 <option value="productComparison">Product Comparison</option>  
 <option value="influencerCaption">Influencer Caption</option>  
 <option value="personalReview">Personal Review</option>  
 <option value="demoScript">Demo Script</option>  
 <option value="prosAndCons">Pros & Cons</option>  
 <option value="whyISwitched">Why I Switched</option>  
 </optgroup>  
 <optgroup label="💡 General / Trend-Based">  
 <option value="top5Under25">Top 5 Under $25</option>  
 <option value="beginnerKit">Beginner Kit</option>  
 <option value="drugstoreDupe">Drugstore Dupe</option>  
 <option value="tiktokBreakdown">TikTok Breakdown</option>  
 <option value="surpriseMe">Surprise Me!</option>  
 </optgroup>  
 </select>  
</div>  
  
  
 <button type="submit" class="btn btn-primary mt-6 w-full">Generate Content (Ctrl/Cmd + Enter)</button>  
 </form>  
  
 <!-- Batch Progress UI -->  
 <div id="batch-progress" class="hidden mb-6">  
 <div class="text-sm text-gray-700 mb-1">  
 Generating batch content... <span id="progress-count">0</span> / <span id="progress-total">0</span>  
 </div>  
 <progress id="progress-bar" class="progress progress-rose w-full h-4" value="0" max="100"></progress>  
 </div>  
  
 <div id="results-container" class="mt-8 hidden">  
 <div id="results" class="card bg-white p-6 shadow-sm">  
 <div class="prose max-w-none">  
 <div id="content-wrapper"></div>  
 </div>  
 <div class="text-center mt-6">  
 <button class="btn btn-outline" onclick="summarizeToVideoScript()">  
 🎬 Summarize as 60-sec Video Script  
 </button>  
 </div>  
 <div id="video-script-output" class="mt-4 text-left whitespace-pre-wrap text-sm hidden"></div>  
 </div>  
 </div>  
  
 <section class="text-center mb-12">  
 <h2 class="text-2xl md:text-3xl font-bold text-rose-700 mb-2 flex items-center justify-center gap-2">  
 📈 AI Trend Digest  
 <button onclick="loadTrendDigest()" class="text-sm text-rose-500 hover:text-rose-700 ml-2 flex items-center">  
 🔁 <span class="underline">Refresh</span>  
 </button>  
 </h2>  
 <div id="trend-digest-box" class="text-left max-w-2xl mx-auto mt-4"></div>  
 </section>  
  
 <footer class="bg-rose-50 p-6 mt-12 text-sm text-center">  
 <p class="text-gray-700">  
 <strong>Affiliate Disclosure</strong><br>  
 As an Amazon Associate, I earn from qualifying purchases. This site may contain affiliate links, which means I may earn a commission if you click a link and make a purchase at no extra cost to you.  
 </p>  
 <p class="mt-4 text-gray-700">  
 <strong>Contact</strong><br>  
 Email: <a href="mailto:sportsplaya11997@gmail.com" class="text-rose-700 hover:underline">sportsplaya11997@gmail.com</a>  
 </p>  
 <p class="mt-4 text-gray-700">  
 <a href="/privacy-policy.html" class="text-rose-700 hover:underline" target="\_blank">Privacy Policy</a>  
 </p>  
 <p class="mt-4 text-gray-700">  
 <a href="/about.html" class="text-rose-700 hover:underline" target="\_blank">About</a>  
 </p>  
 </footer>  
 </div>  
  
 <script src="script.js"></script>  
</body>  
</html>

# Amazon Trending Scraper (amazonTrendingScraper.js)

// scrapers/amazonTrendingScraper.js  
const axios = require('axios');  
const cheerio = require('cheerio');  
  
const SCRAPER\_API\_KEY = process.env.SCRAPER\_API\_KEY;  
  
async function getAmazonTrending() {  
 const targetURL = 'https://www.amazon.com/Best-Sellers-Beauty/zgbs/beauty';  
 const scraperUrl = `http://api.scraperapi.com?api\_key=${SCRAPER\_API\_KEY}&url=${encodeURIComponent(targetURL)}`;  
  
 try {  
 const res = await axios.get(scraperUrl);  
 const $ = cheerio.load(res.data);  
  
 const products = [];  
  
 $('div.zg-grid-general-faceout').each((i, el) => {  
 if (i < 5) {  
 const title =  
 $(el).find('.\_cDEzb\_p13n-sc-css-line-clamp-3\_g3dy1').text().trim() ||  
 $(el).find('.\_cDEzb\_p13n-sc-css-line-clamp-2\_EWgCb').text().trim();  
 const link = 'https://www.amazon.com' + $(el).find('a.a-link-normal').attr('href');  
 products.push({ title, link });  
 }  
 });  
  
 return products;  
 } catch (err) {  
 console.error('❌ Error scraping Amazon trending:', err.message);  
 return [  
 { title: 'Fallback Product 1', link: 'https://www.amazon.com' },  
 { title: 'Fallback Product 2', link: 'https://www.amazon.com' }  
 ];  
 }  
}  
  
module.exports = getAmazonTrending;

# Google Trends Scraper (googleTrendsScraper.js)

const fetch = require('node-fetch');  
  
async function getGoogleTrends() {  
 try {  
 const url = `https://api.rss2json.com/v1/api.json?rss\_url=https://trends.google.com/trends/trendingsearches/daily/rss?geo=US`;  
 const response = await fetch(url);  
 const json = await response.json();  
  
 const items = json.items || [];  
  
 return items.slice(0, 5).map(item => ({  
 title: item.title,  
 pubDate: item.pubDate,  
 link: item.link,  
 }));  
 } catch (error) {  
 console.error('❌ Google Trends Scraper Error:', error.message);  
 return [  
 { title: 'Fallback Trend 1', pubDate: '', link: '' },  
 { title: 'Fallback Trend 2', pubDate: '', link: '' },  
 ];  
 }  
}  
  
module.exports = getGoogleTrends;

# Instagram Scraper (instagramScraper.js)

// ✅ Instagram Scraper with GPT-generated fallback trends  
async function getInstagramTrending() {  
 try {  
 const OpenAI = require('openai');  
 const openai = new OpenAI({ apiKey: process.env.OPENAI\_API\_KEY });  
  
 const completion = await openai.chat.completions.create({  
 model: 'gpt-4',  
 messages: [  
 {  
 role: 'system',  
 content: 'You are a skincare trend expert. Provide 5 trending skincare products from Instagram formatted as JSON: [{ title, link, caption }]. Use placeholder # for links.'  
 },  
 {  
 role: 'user',  
 content: 'What skincare products are trending on Instagram right now?'  
 }  
 ],  
 temperature: 0.7,  
 });  
  
 const raw = completion.choices[0].message.content.trim();  
 const jsonStart = raw.indexOf('[');  
 const json = raw.slice(jsonStart);  
 return JSON.parse(json);  
 } catch (err) {  
 const errorReason = 'Using AI-generated content because Instagram API requires Meta Business account and has strict access limitations';  
 console.error('❌ Instagram Scraper Error:', {  
 timestamp: new Date().toISOString(),  
 endpoint: 'GPT-4 Instagram Trends',  
 requestStatus: err.status || 'unknown',  
 quotaExceeded: err.message.includes('quota'),  
 rateLimited: err.message.includes('429'),  
 message: err.message,  
 type: err.name,  
 stack: err.stack?.split('\n')[0],  
 openaiError: err.response?.data?.error || null  
 });  
 console.log("📍 Error occurred in:", \_\_filename);  
 console.log(`⚠️ ${errorReason}`);  
 return [  
 { title: "Glow Recipe Watermelon Toner", link: "#", caption: "Hydration + Glow in one swipe" },  
 { title: "CeraVe Moisturizing Cream", link: "#", caption: "TikTok’s favorite for barrier repair" },  
 { title: "The Ordinary Niacinamide", link: "#", caption: "Affordable pore shrinker" },  
 { title: "Laneige Lip Sleeping Mask", link: "#", caption: "Overnight lip revival" },  
 { title: "Tatcha Dewy Skin Cream", link: "#", caption: "Luxe glow for dry skin days" }  
 ];  
 }  
}  
  
module.exports = { getInstagramTrending };

# Reddit Scraper (redditScraper.js)

const axios = require('axios');  
const qs = require('qs');  
  
const REDDIT\_CLIENT\_ID = process.env.REDDIT\_CLIENT\_ID;  
const REDDIT\_SECRET = process.env.REDDIT\_SECRET;  
const REDDIT\_USERNAME = process.env.REDDIT\_USERNAME;  
const REDDIT\_PASSWORD = process.env.REDDIT\_PASSWORD;  
const REDDIT\_USER\_AGENT = 'Glowbot Skincare by Then-Bodybuilder4539';  
async function getRedditAccessToken() {  
 const auth = Buffer.from(`${REDDIT\_CLIENT\_ID}:${REDDIT\_SECRET}`).toString('base64');  
  
 try {  
 const response = await axios.post(  
 'https://www.reddit.com/api/v1/access\_token',  
 qs.stringify({  
 grant\_type: 'password',  
 username: REDDIT\_USERNAME,  
 password: REDDIT\_PASSWORD,  
 }),  
 {  
 headers: {  
 Authorization: `Basic ${auth}`,  
 'Content-Type': 'application/x-www-form-urlencoded',  
 'User-Agent': REDDIT\_USER\_AGENT,  
 },  
 }  
 );  
  
 return response.data.access\_token;  
 } catch (error) {  
 console.error('⚠️ Reddit Token Error:', error.response?.data || error.message);  
 return null;  
 }  
}  
  
async function getRedditTrending() {  
 const token = await getRedditAccessToken();  
 if (!token) {  
 throw new Error('Failed to obtain Reddit access token');  
 }  
  
 try {  
 const response = await axios.get(  
 'https://oauth.reddit.com/r/SkincareAddiction/hot?limit=10',  
 {  
 headers: {  
 Authorization: `Bearer ${token}`,  
 'User-Agent': REDDIT\_USER\_AGENT,  
 },  
 }  
 );  
  
 const posts = response.data.data.children;  
  
 return posts  
 .filter(p => p.data && !p.data.stickied)  
 .map(p => ({  
 title: p.data.title,  
 link: `https://reddit.com${p.data.permalink}`,  
 caption: 'Trending Reddit skincare discussion',  
 }));  
 } catch (err) {  
 console.error('⚠️ Reddit fetch error:', err.response?.data || err.message);  
 return [];  
 }  
}  
  
module.exports = { getRedditTrending };

# TikTok Scraper (tikTokScraper.js)

const axios = require('axios');  
const cheerio = require('cheerio');  
  
const SCRAPER\_API\_KEY = process.env.SCRAPER\_API\_KEY;  
  
const hashtags = [  
 'skincare',  
 'skincareroutine',  
 'acnetips',  
 'dryskin',  
 'glowup',  
 'skincaretips',  
 'skincareproducts',  
];  
  
function getRandomHashtag() {  
 const randomIndex = Math.floor(Math.random() \* hashtags.length);  
 return hashtags[randomIndex];  
}  
  
async function getTikTokTrending() {  
 if (!SCRAPER\_API\_KEY) {  
 console.error('⚠️ SCRAPER\_API\_KEY not found in environment variables');  
 return [];  
 }  
  
 const hashtag = getRandomHashtag();  
 // Try TikTok discovery instead of hashtag for more reliable scraping  
 const targetURL = `https://www.tiktok.com/discover/${hashtag}`;  
 const scraperUrl = `http://api.scraperapi.com?api\_key=${SCRAPER\_API\_KEY}&url=${encodeURIComponent(targetURL)}&render=true`;  
  
  
 try {  
 console.log(`🔍 Attempting to scrape TikTok discovery page: /discover/${hashtag}`);  
  
 const maxRetries = 3;  
 const timeout = 60000;  
 let attempt = 0;  
 let response;  
  
 while (attempt < maxRetries) {  
 try {  
 response = await axios.get(scraperUrl, {  
 headers: {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 Chrome/122.0.0.0 Safari/537.36'  
 },  
 timeout: timeout  
 });  
  
 if (response.data) {  
 // Log raw HTML preview for debugging  
 console.log('📃 TikTok raw HTML preview:\n', response.data.slice(0, 500));  
 break;  
 }  
 } catch (err) {  
 attempt++;  
 if (attempt === maxRetries) throw err;  
 console.log(`⏳ Retry attempt ${attempt} for hashtag #${hashtag}`);  
 await new Promise(resolve => setTimeout(resolve, 2000));  
 }  
 }  
  
 if (!response?.data) {  
 throw new Error('Empty response from scraper API');  
 }  
  
 const $ = cheerio.load(response.data);  
 const posts = [];  
  
 const selectors = [  
 'div[data-e2e="search-video-item"]',  
 'div[data-e2e="video-item"]',  
 '.video-feed-item',  
 'div[data-e2e="browse-video-item"]' // added for discover  
 ];  
  
 for (const selector of selectors) {  
 $(selector).each((i, el) => {  
 const title =  
 $(el).find('a').attr('title') ||  
 $(el).find('[data-e2e="video-desc"]').text() ||  
 `Trending TikTok on #${hashtag}`;  
  
 const link = $(el).find('a').attr('href');  
  
 if (link) {  
 posts.push({  
 title: title.trim(),  
 link: link.startsWith('http') ? link : `https://www.tiktok.com${link}`,  
 caption: `Trending TikTok skincare video from #${hashtag}`  
 });  
 }  
 });  
  
 if (posts.length > 0) break;  
 }  
  
 if (posts.length === 0) {  
 console.warn('⚠️ No TikTok posts found with any selector');  
 return [];  
 }  
  
 console.log(`✅ Successfully scraped ${posts.length} TikTok posts`);  
 return posts.slice(0, 6);  
  
 } catch (err) {  
 console.error('❌ TikTok scraper error:', {  
 message: err.message,  
 status: err.response?.status,  
 statusText: err.response?.statusText,  
 hashtag,  
 timestamp: new Date().toISOString()  
 });  
 return [];  
 }  
}  
  
module.exports = { getTikTokTrending };

# YouTube Scraper (youtubeScraper.js)

const fetch = require('node-fetch');  
  
const YOUTUBE\_API\_KEY = process.env.YOUTUBE\_API\_KEY;  
const MAX\_RESULTS = 10;  
  
async function getYouTubeTrending() {  
 try {  
 const query = 'skincare routine OR review OR unboxing OR product demo';  
  
 const url = `https://www.googleapis.com/youtube/v3/search?key=${YOUTUBE\_API\_KEY}&part=snippet&type=video&q=${encodeURIComponent(query)}&order=date&maxResults=${MAX\_RESULTS}`;  
  
 const response = await fetch(url);  
 const data = await response.json();  
  
 const videos = (data.items || [])  
 .filter(item =>  
 item?.snippet?.title &&  
 /skin|beauty|routine|glow|acne|hydration/i.test(item.snippet.title)  
 )  
 .map(item => ({  
 title: item.snippet.title,  
 url: `https://www.youtube.com/watch?v=${item.id.videoId}`,  
 publishedAt: item.snippet.publishedAt  
 }));  
  
 return videos;  
 } catch (error) {  
 console.error('❌ YouTube Scraper Error:', error.message);  
 return [];  
 }  
}  
  
module.exports = { getYouTubeTrending };