AI BASED PRODUCT PROMOTER

CODING

app.py

```
from flask import Flask, render_template, request, jsonify, send_from_directory
from models.caption_generator import generate_caption
from models.text_generator import generate_marketing_text
from models.image_generator import generate_image
import os
from werkzeug.utils import secure_filename
import logging

app = Flask(__name__)

# Set up logging
logging.basicConfig(level=logging.DEBUG)

UPLOAD_FOLDER = 'uploads'
STATIC_FOLDER = 'static'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/generate-caption', methods=['POST'])
def generate_caption_route():
    try:
        image = request.files.get('image')
        if not image:
            return jsonify({'error': 'No image uploaded'}), 400

        filename = secure_filename(image.filename)
        filepath = os.path.join(UPLOAD_FOLDER, filename)
        image.save(filepath)

        caption = generate_caption(filepath)
        web_path = filepath.replace("\\", "/")  # Ensure browser-safe path
        return jsonify({'caption': caption, 'path': web_path})
    except Exception as e:
        logging.error(f"Caption error: {e}")
        return jsonify({'error': str(e)}), 500

@app.route('/generate-text', methods=['POST'])
def generate_text_route():
    try:
        data = request.get_json()
        caption = data.get('caption')
        if not caption:
            return jsonify({'error': 'No caption provided'}), 400

        text = generate_marketing_text(caption)
        return jsonify({'text': text})
    except Exception as e:
        logging.error(f"Text generation error: {e}")
        return jsonify({'error': str(e)}), 500

@app.route('/generate-image', methods=['POST'])
def generate_image_route():
    try:
        data = request.get_json()
        input_path = data.get('input_path')
        if not input_path:
            return jsonify({'error': 'No input image path provided'}), 400

        filename = generate_image(input_image_path=input_path)
        rel_path = filename.replace('static/', '', 1)
        return jsonify({'image_url': f'/static/{rel_path}'})
    except Exception as e:
        logging.error(f"Image enhancement error: {e}")
        return jsonify({'error': str(e)}), 500

@app.route('/static/<path:filename>')
def static_files(filename):
    return send_from_directory(STATIC_FOLDER, filename)

if __name__ == '__main__':
    app.run(debug=True)
```

Caption_Generation

```
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
import torch

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model =
BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-ba
se")

def generate_caption(image_path):
    image = Image.open(image_path).convert('RGB')
    inputs = processor(image, return_tensors="pt")
    out = model.generate(**inputs)
    return processor.decode(out[0], skip_special_tokens=True)
```

Image_Generation

```
from PIL import Image
from realesrgan import RealESRGANer
from basicsr.archs.srvgg_arch import SRVGGNetCompact
import torch
import os
import uuid
import numpy as np

STATIC_FOLDER = 'static'
os.makedirs(STATIC_FOLDER, exist_ok=True)

# Device setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Load model architecture
model = SRVGGNetCompact(
    num_in_ch=3, num_out_ch=3, num_feat=64,
    num_conv=32, upscale=4, act_type='prelu'
)
model.to(device)

# Initialize RealESRGANer with tiling enabled
upsampler = RealESRGANer(
    scale=4,
    model_path='weights/realesr-general-x4v3.pth',
    model=model,
    tile=128,  # Enables tiling to avoid memory issues
    tile_pad=10,
    pre_pad=0,
    half=not torch.cuda.is_available()
)

def generate_image(prompt=None, input_image_path=None):
    try:
        if not input_image_path:
            raise ValueError("Input image path is required for enhancement.")

        # Load and convert image
        image = Image.open(input_image_path).convert("RGB")

        # Optional: Resize very large images to avoid OOM
        MAX_RES = 2048
        if max(image.size) > MAX_RES:
            image.thumbnail((MAX_RES, MAX_RES), Image.Resampling.LANCZOS)

        image_np = np.array(image)

        # Enhance image
        output, _ = upsampler.enhance(image_np)

        # Convert output back to PIL and save
        output_image = Image.fromarray(output)
        filename = f"{uuid.uuid4()}.jpg"
        save_path = os.path.join(STATIC_FOLDER, filename)
        output_image.save(save_path)
        return f"static/{filename}"

    except Exception as e:
        print(f"❌ Image enhancement error: {str(e)}")
        return None
```

Text_Generation

```
import requests
import os
from dotenv import load_dotenv

load_dotenv()

GROQ_API_KEY = os.getenv("GROQ_API_KEY")
GROQ_API_URL = "https://api.groq.com/openai/v1/chat/completions"

def generate_marketing_text(caption):
    headers = {
        "Authorization": f"Bearer {GROQ_API_KEY}",
        "Content-Type": "application/json"
    }

    prompt = (
        f"You are a professional marketing assistant. Based on the following image
caption: "
        f"\"{caption}\", generate well-structured HTML content that includes:\n\n"
        f"<b>Product Description:</b> (Wrap in <p> tag)\n"
        f"<b>Slogans:</b> (List using <ul><li>)\n"
        f"<b>Social Media Captions:</b> (List using <ul><li>)\n"
        f"<b>Hashtags:</b> (List using <ul><li>)\n\n"
        f"Ensure there is clear spacing between sections and headings are bold. Do not
wrap the entire response in <html> or <body> tags."
    )

    data = {
        "model": "llama3-70b-8192",
        "messages": [{"role": "user", "content": prompt}],
        "temperature": 0.7
    }

    try:
        response = requests.post(GROQ_API_URL, headers=headers, json=data)
        result = response.json()
        if 'choices' in result:
            return result['choices'][0]['message']['content'].strip()
        else:
            print("Groq API Error:", response.text)
            return "Error generating text."
    except Exception as e:
        print("❌ Request failed:", str(e))
        return "Error generating text."
```