import streamlit as st

from openai import OpenAI

import torch

from diffusers import StableDiffusionPipeline

from PIL import Image

import base64

import io

# App Configuration

st.set\_page\_config(

    page\_title="LogoSynth AI - AI Logo Generator",

    page\_icon="✨",

    layout="wide"

)

# CSS

st.markdown("""

<style>

    :root {

        --primary: #6d28d9;

        --primary-light: #8b5cf6;

        --primary-dark: #5b21b6;

        --secondary: #ec4899;

        --accent-1: #06b6d4;

        --accent-2: #f59e0b;

        --accent-3: #10b981;

        --dark: #0f172a;

        --light: #f8fafc;

        --gray: #64748b;

        --bg: #0e1117;

        --card-bg: #1e293b;

        --border: #334155;

    }

    body {

        background-color: var(--bg);

        color: var(--light);

        font-family: 'Poppins', 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

        line-height: 1.6;

    }

    .stApp {

        background-color: var(--bg);

    }

    .header {

        background: linear-gradient(120deg, var(--primary), var(--secondary), var(--accent-1));

        background-size: 200% 200%;

        animation: gradient-shift 10s ease infinite;

        color: white;

        padding: 4rem 0;

        text-align: center;

        position: relative;

        overflow: hidden;

        box-shadow: 0 4px 20px rgba(0, 0, 0, 0.2);

        border-bottom-left-radius: 30px;

        border-bottom-right-radius: 30px;

        margin-bottom: 3rem;

    }

    @keyframes gradient-shift {

        0% {background-position: 0% 50%;}

        50% {background-position: 100% 50%;}

        100% {background-position: 0% 50%;}

    }

    .logo-title {

        font-size: 3.5rem;

        font-weight: 800;

        margin-bottom: 1rem;

        text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.2);

        letter-spacing: -1px;

    }

    .logo-title span {

        color: var(--accent-2);

        background: white;

        padding: 0 10px;

        border-radius: 8px;

        color: var(--primary);

    }

    .tagline {

        font-size: 1.4rem;

        margin-bottom: 2rem;

        max-width: 800px;

        margin-left: auto;

        margin-right: auto;

    }

    .generator-card {

        background: linear-gradient(135deg, var(--card-bg), #1a2333);

        border-radius: 20px;

        padding: 2.5rem;

        box-shadow: 0 15px 35px rgba(0, 0, 0, 0.1);

        margin-bottom: 3rem;

        position: relative;

        overflow: hidden;

        border: 1px solid var(--border);

    }

    .generator-title {

        display: flex;

        align-items: center;

        margin-bottom: 2rem;

    }

    .generator-title h2 {

        font-size: 2rem;

        font-weight: 700;

        margin-left: 15px;

        background: linear-gradient(90deg, var(--primary), var(--secondary));

        -webkit-background-clip: text;

        background-clip: text;

        -webkit-text-fill-color: transparent;

        color: transparent;

    }

    .generator-title .icon {

        font-size: 2.5rem;

        color: var(--primary);

    }

    .stTextInput>div>div>input, .stSelectbox>div>div>select, .stTextArea>div>div>textarea {

        background-color: var(--card-bg);

        color: var(--light);

        border-color: var(--border);

        border-radius: 10px;

        padding: 0.9rem 1.2rem;

        font-size: 1rem;

        transition: all 0.3s;

        box-shadow: 0 2px 5px rgba(0, 0, 0, 0.05);

    }

    .stTextInput>div>div>input:focus, .stSelectbox>div>div>select:focus, .stTextArea>div>div>textarea:focus {

        outline: none;

        border-color: var(--primary-light);

        box-shadow: 0 0 0 3px rgba(139, 92, 246, 0.2);

    }

    .stSelectbox>div>div>select {

        appearance: none;

        background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 24 24' fill='none' stroke='%238b5cf6' stroke-width='2' stroke-linecap='round' stroke-linejoin='round'%3e%3cpolyline points='6 9 12 15 18 9'%3e%3c/polyline%3e%3c/svg%3e");

        background-repeat: no-repeat;

        background-position: right 1rem center;

        background-size: 1.2em;

        padding-right: 3rem;

    }

    .stButton>button {

        background: linear-gradient(90deg, var(--primary), var(--secondary));

        color: white;

        border: none;

        padding: 1rem 2rem;

        border-radius: 10px;

        font-size: 1.1rem;

        font-weight: 600;

        cursor: pointer;

        transition: all 0.3s;

        box-shadow: 0 4px 15px rgba(109, 40, 217, 0.4);

        width: 100%;

    }

    .stButton>button:hover {

        transform: translateY(-2px);

        box-shadow: 0 6px 18px rgba(109, 40, 217, 0.5);

    }

    .stButton>button:active {

        transform: translateY(0);

    }

    .logo-card {

        background-color: var(--card-bg);

        border-radius: 16px;

        padding: 15px;

        border: 1px solid var(--border);

        margin-bottom: 20px;

        box-shadow: 0 10px 25px rgba(0, 0, 0, 0.08);

        transition: transform 0.3s, box-shadow 0.3s;

    }

    .logo-card:hover {

        transform: translateY(-10px);

        box-shadow: 0 15px 35px rgba(0, 0, 0, 0.12);

    }

    .spinner {

        width: 70px;

        height: 70px;

        border-radius: 50%;

        background: conic-gradient(transparent 50%, var(--primary));

        -webkit-mask: radial-gradient(farthest-side, transparent calc(100% - 10px), #fff 0);

        mask: radial-gradient(farthest-side, transparent calc(100% - 10px), #fff 0);

        animation: spin 1s infinite linear;

        margin: 0 auto 1rem;

    }

    @keyframes spin {

        to { transform: rotate(360deg); }

    }

    .footer {

        background-color: var(--dark);

        color: white;

        padding: 2rem 0;

        border-top-left-radius: 30px;

        border-top-right-radius: 30px;

        margin-top: 3rem;

    }

    .footer-bottom {

        text-align: center;

        padding-top: 2rem;

        border-top: 1px solid rgba(255, 255, 255, 0.1);

        font-size: 0.9rem;

        color: rgba(255, 255, 255, 0.5);

    }

    @media (max-width: 768px) {

        .logo-title {

            font-size: 2.5rem;

        }

        .tagline {

            font-size: 1.1rem;

        }

    }

</style>

""", unsafe\_allow\_html=True)

# Initialize OpenAI client

client = OpenAI(

    base\_url="https://openrouter.ai/api/v1",

    api\_key="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",  # Replace with your actual API key

)

# Initialize Stable Diffusion pipeline

@st.cache\_resource

def load\_model():

    return StableDiffusionPipeline.from\_pretrained(

        "runwayml/stable-diffusion-v1-5",

        torch\_dtype=torch.float16,

        safety\_checker=None

    ).to("cuda")

pipe = load\_model()

def refine\_description(raw\_description, brand\_name, industry, style, color\_scheme):

    # First validate all inputs exist and are strings

    if not all([isinstance(x, str) and x.strip() for x in [raw\_description, brand\_name, industry, style, color\_scheme]]):

        return raw\_description  # Return original if any parameter is invalid

    try:

        response = client.chat.completions.create(

            model="deepseek/deepseek-chat-v3-0324:free",

            messages=[

                {"role": "system", "content": "Refine this logo description..."},

                {"role": "user", "content": raw\_description}

            ]

        )

        # Simple, robust response extraction

        return getattr(getattr(getattr(response, 'choices', [{}])[0], 'message', {}), 'content', raw\_description)

    except Exception:

        return raw\_description  # Silent fallback to original

def create\_prompt(refined\_description, brand\_name, industry, style, color\_scheme):

    # Basic validation

    if not isinstance(refined\_description, str) or not refined\_description.strip():

        refined\_description = f"{brand\_name} {industry} logo"

    try:

        response = client.chat.completions.create(

            model="deepseek/deepseek-chat-v3-0324:free",

            messages=[

                {"role": "system", "content": "Create logo prompt..."},

                {"role": "user", "content": refined\_description}

            ]

        )

        # Simple response extraction with fallback

        return getattr(getattr(getattr(response, 'choices', [{}])[0], 'message', {}), 'content',

                      f"Professional {style} logo for {brand\_name} with {color\_scheme} colors")

    except Exception:

        return f"Professional {style} logo for {brand\_name} with {color\_scheme} colors"

# Function to generate logos

def generate\_logos(final\_prompt):

    logos = []

    with st.spinner("Generating unique logo concepts..."):

        for i in range(3):  # Generate 3 variants

            image = pipe(

                final\_prompt,

                num\_inference\_steps=50,

                guidance\_scale=7.5

            ).images[0]

            buffered = io.BytesIO()

            image.save(buffered, format="PNG")

            img\_bytes = base64.b64encode(buffered.getvalue()).decode()

            logos.append(img\_bytes)

    return logos

# App Header

st.markdown("""

<div class="header">

    <div class="container">

        <div class="header-content">

            <div class="logo-title">Logo<span>Synth AI</span></div>

            <div class="tagline">Where Imagination Meets Algorithm</div>

        </div>

    </div>

</div>

""", unsafe\_allow\_html=True)

# Main content container

with st.container():

    # Generator card

    st.markdown("""

    <div class="generator-card">

        <div class="generator-title">

            <div class="icon">✨</div>

            <h2>Create Your Dream Logo</h2>

        </div>

    """, unsafe\_allow\_html=True)

    # Input form

    with st.form("logo\_form"):

        col1, col2 = st.columns(2)

        with col1:

            brand\_name = st.text\_input("Brand Name", placeholder="Your company name")

        with col2:

            industry = st.selectbox("Industry", [

                "Technology", "Food & Beverage", "Health & Wellness",

                "Finance", "Education", "Fashion", "Creative Arts",

                "Sports", "Real Estate", "Other"

            ])

        # Additional style options

        style = st.selectbox("Style", [

            "Modern & Minimalist", "Vintage & Retro", "Playful & Colorful",

            "Geometric", "Handcrafted", "Luxury", "Futuristic",

            "Organic", "Abstract", "Corporate"

        ])

        # Color scheme selection

        color\_scheme = st.selectbox("Color Scheme", [

            "Cool Blues", "Warm Reds & Oranges", "Nature Greens",

            "Vibrant Rainbow", "Monochromatic", "Pastel Tones",

            "Neon Brights", "Earth Tones", "Metallic Shades",

            "Dark & Moody"

        ])

        # Description field

        description = st.text\_area(

            "Describe your ideal logo",

            placeholder="e.g., I want a fox mascot logo with geometric shapes that looks modern but friendly...",

            height=100

        )

        generate\_col, regenerate\_col = st.columns(2)

        with generate\_col:

            generate\_clicked = st.form\_submit\_button("✨ Generate Logos", use\_container\_width=True)

        with regenerate\_col:

            regenerate\_clicked = st.form\_submit\_button("🔄 Regenerate", use\_container\_width=True,

                                                     disabled=not st.session\_state.get('logos'))

    st.markdown("</div>", unsafe\_allow\_html=True)  # Close generator-card

# In the generation logic section, update the code to properly handle color\_scheme:

# In your generation logic:

if generate\_clicked or regenerate\_clicked:

    if not brand\_name or not description:

        st.warning("Please fill in both brand name and description")

    else:

        with st.spinner("Creating your logos..."):

            # Get all inputs with defaults

            brand = str(brand\_name) if brand\_name else "Brand"

            desc = str(description) if description else "Logo"

            industry\_type = str(industry) if industry else "Business"

            logo\_style = str(style) if style else "Modern"

            colors = str(color\_scheme) if color\_scheme else "vibrant colors"

            # Safe generation process

            try:

                refined = refine\_description(desc, brand, industry\_type, logo\_style, colors)

                prompt = create\_prompt(refined, brand, industry\_type, logo\_style, colors)

                if prompt:  # Only generate if we got a valid prompt

                    st.session\_state.logos = generate\_logos(prompt)

                    st.session\_state.final\_prompt = prompt

                    st.session\_state.refined\_description = refined

                else:

                    st.warning("Couldn't create prompt - using default")

                    st.session\_state.logos = generate\_logos(

                        f"Professional {logo\_style} logo for {brand} with {colors}"

                    )

            except Exception as e:

                st.error("Logo generation encountered an issue")

                st.session\_state.logos = generate\_logos(

                    f"Simple {logo\_style} logo for {brand}"

                )

# Display results

if 'logos' in st.session\_state:

    st.subheader("Your Generated Logos")

    cols = st.columns(3)

    for idx, img\_bytes in enumerate(st.session\_state.logos):

        with cols[idx]:

            st.markdown(f'<div class="logo-card">', unsafe\_allow\_html=True)

            st.image(f"data:image/png;base64,{img\_bytes}", use\_container\_width=True)

            st.download\_button(

                label="Download Logo",

                data=base64.b64decode(img\_bytes),

                file\_name=f"{brand\_name.replace(' ', '\_')}\_logo\_{idx+1}.png",

                mime="image/png",

                use\_container\_width=True

            )

            st.markdown(f'</div>', unsafe\_allow\_html=True)

# Simplified Footer

st.markdown("""

<div class="footer">

    <div class="container">

        <div class="footer-bottom">

            <p>Made by Parth Chauhan, Aryan Makwana, Sumit Solanki</p>

        </div>

    </div>

</div>

""", unsafe\_allow\_html=True)