Resume Generator

Name : Nikunj Parida

Registration no. : 23BCE2004

Email : nikunj.parida2023@vitstudent.ac.in

Code :

```python
# resume_generator.py
import os
import streamlit as st
import google.generativeai as genai
from dotenv import load_dotenv
import pyperclip # For copy to clipboard functionality

# --- 1. API Key Loading ---
# Load environment variables from .env file in the current working directory
# (which should be the project root when running 'streamlit run')
load_dotenv()
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")

if not GOOGLE_API_KEY:
    st.error("GOOGLE_API_KEY not found in environment variables. "
             "Please create a .env file in the same directory as this script "
             "and add GOOGLE_API_KEY=\"YOUR_GEMINI_API_KEY_HERE\"")
    st.stop() # Stop the Streamlit app if API key is missing

# Configure the Google Generative AI API with your key
genai.configure(api_key=GOOGLE_API_KEY)

# --- 2. AI Generation Functions ---

def generate_resume_section(section_name, user_input, model_name="gemini-pro"):
    """
    Generates content for a specific resume section using Google Gemini API.
    Args:
        section_name (str): The name of the resume section (e.g., "Summary", "Experience").
        user_input (str): User-provided information/details for this section.
        model_name (str): The Gemini model to use. "gemini-pro" is generally good.

    Returns:
        str: Generated content for the resume section, or an error string.
    """
    model = genai.GenerativeModel(model_name)

    if not user_input or user_input.strip() == "":
        return f"No input provided for {section_name}. Skipping generation."

```

```python
    # --- Prompt Engineering for Resume Sections ---
    if section_name == "Summary":
        prompt = f"""
        You are an expert resume writer.
        Generate a concise and impactful professional summary for a resume.
        Focus on the user's key skills, experience, and career goals.
        Keep it to 3-4 strong sentences.

        User's core information: {user_input}

        Professional Summary:
        """
    elif section_name == "Experience":
        prompt = f"""
        You are an expert resume writer.
        Based on the following job details, generate 3-5 strong bullet points that highlight achievements, responsibilities, and impact.
        Use action verbs and quantify results where possible.

        Job Details: {user_input}

        Experience Bullet Points:
        """
    elif section_name == "Education":
        prompt = f"""
        You are an expert resume writer.
        Format the following educational background for a resume.
        Include degree, major, institution, and graduation year.
        If relevant, add honors or key coursework.

        Education Details: {user_input}

        Education Section:
        """
    elif section_name == "Skills":
        prompt = f"""
        You are an expert resume writer.
        Categorize and list the following skills for a resume.
        Group them logically (e.g., Programming Languages, Tools, Soft Skills).
        Use bullet points or a comma-separated list within categories.

        Raw Skills: {user_input}

        Skills Section:
        """
    else:
        return f"Invalid section name: {section_name}"

    try:
        response = model.generate_content(prompt)
        # Access the text content from the response
        return response.text.strip()
    except Exception as e:
        return f"Error generating {section_name}: {e}"
```

```
        return f"Error generating {section_name}: {e}"

def generate_full_resume_content(user_data):
    """
    Generates a full resume by combining generated sections.

    Args:
        user_data (dict): A dictionary containing user information for each section.
                          Example: {'Summary': '...', 'Experience': ['job1_str', 'job2_str'], ...}

    Returns:
        str: The complete resume in a structured text format (Markdown).
    """
    resume_parts = []
    has_content = False

    # Define a helper function for consistent formatting and error handling
    def add_section(section_title, section_key, input_data, is_list=False):
        nonlocal has_content # Indicate that has_content might be modified
        generated_text = ""

        if input_data:
            if is_list:
                temp_list_parts = []
                list_generated_any = False
                for item in input_data:
                    if item.strip(): # Only process non-empty items
                        result = generate_resume_section(section_key, item)
                        if not result.startswith("Error"):
                            # For experience, we add a placeholder for title/company
                            if section_key == "Experience":
                                temp_list_parts.append(f"### [Job Title], [Company Name] - [Start Date] - [End Date]\n{result}\n")
                            else: # For other list-based sections if you add them
                                temp_list_parts.append(result)
                            list_generated_any = True
                        else:
                            # Show error for specific item if generation failed
                            temp_list_parts.append(f"### [Job Title], [Company Name]\n_{result}_\n")
                if temp_list_parts:
                    resume_parts.append(f"## {section_title}\n" + "".join(temp_list_parts) + "\n")
                    has_content = has_content or list_generated_any
            else: # Not a list, single text area
                result = generate_resume_section(section_key, input_data)
                if not result.startswith("Error"):
                    generated_text = result
                    has_content = True
                else:
                    generated_text = f"_{result}_" # Indicate error for display

        if generated_text and not is_list: # Add section if it's not a list and has content
            resume_parts.append(f"## {section_title}\n{generated_text}\n")

    # Generate sections using the helper
    add_section("Professional Summary", "Summary", user_data.get('Summary', ''))
    add_section("Experience", "Experience", user_data.get('Experience', []), is_list=True) # Pass Experience as a list
    add_section("Education", "Education", user_data.get('Education', ''))
    add_section("Skills", "Skills", user_data.get('Skills', ''))

    if not has_content:
        return "Please provide more details to generate your resume, or there was an issue with API key/model generation."

    return "\n".join(resume_parts)
```

```python
# --- Streamlit Page Configuration ---
st.set_page_config(
    page_title="AI-Powered Resume Generator",
    page_icon="📄",
    layout="wide",
    initial_sidebar_state="expanded"
)

# --- Header ---
st.title("📄 AI-Powered Resume Generator")
st.markdown("Use Google's Generative AI (Gemini) to quickly draft your resume sections!")

# --- Input Form ---
st.header("Your Information")
st.write("Provide details for each section. The AI will expand and format them professionally.")

with st.form("resume_input_form"):
    st.subheader("1. Professional Summary (Overall Experience/Goals)")
    summary_input = st.text_area(
        "Describe your professional background, key skills, and career aspirations:",
        "Experienced software engineer with 5 years in Python and web development. Strong in building scalable applications and leading sma
        height=100,
        key="summary_input_area"
    )

    st.subheader("2. Experience (One job per box)")
    st.markdown("Enter details for each job role. Focus on responsibilities, achievements, and impact.")
```

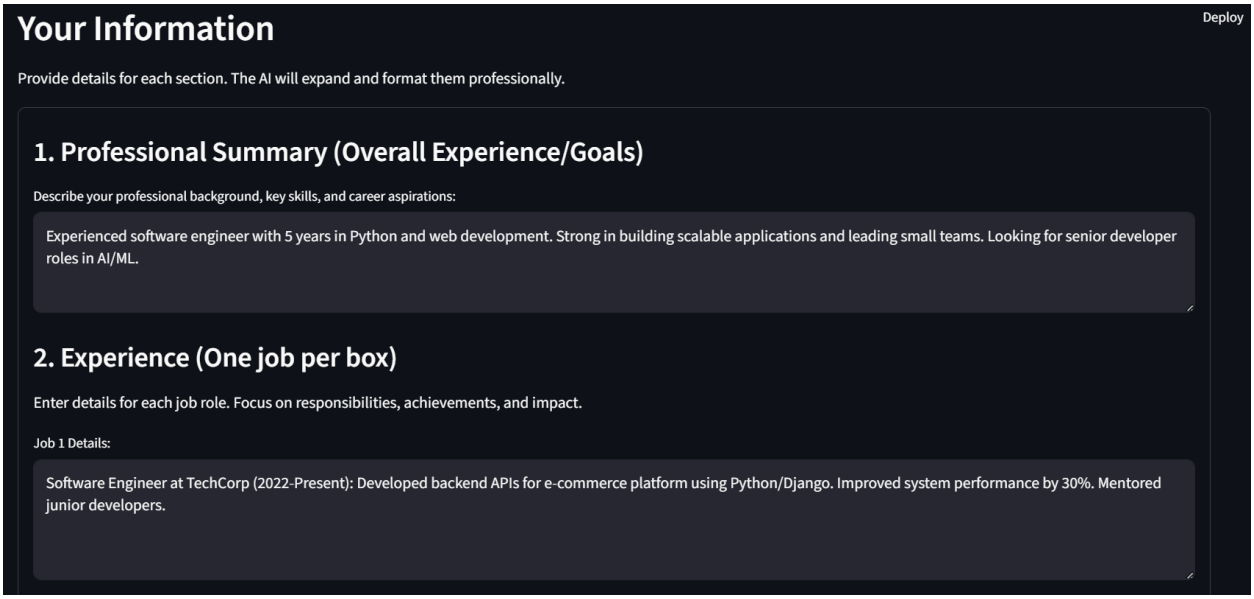
```python
    # Initialize a list of job inputs in session state if not already there
    if 'experience_inputs' not in st.session_state:
        st.session_state.experience_inputs = [
            "Software Engineer at TechCorp (2022-Present): Developed backend APIs for e-commerce platform using Python/Django. Improved sy
            "Junior Developer at InnovateX (2020-2022): Built frontend components with React. Assisted in database design (SQL). Contribut
        ]

    # Render all existing experience inputs
    for i, job_input in enumerate(st.session_state.experience_inputs):
        st.session_state.experience_inputs[i] = st.text_area(
            f"Job {i+1} Details:", job_input, height=120, key=f"job_details_{i}"
        )

    col1, col2 = st.columns(2)
    with col1:
        # Changed st.form_submit_button to st.button as per Streamlit best practices
        # for dynamic element additions within a form context that trigger rerun
        if st.form_submit_button("Add Another Job", help="Click to add another text box for a job."):
            st.session_state.experience_inputs.append("")
            st.rerun() # Rectified: Changed to st.rerun()
    with col2:
        if len(st.session_state.experience_inputs) > 1:
            if st.form_submit_button("Remove Last Job", help="Click to remove the last job text box."):
                st.session_state.experience_inputs.pop()
                st.rerun() # Rectified: Changed to st.rerun()
```

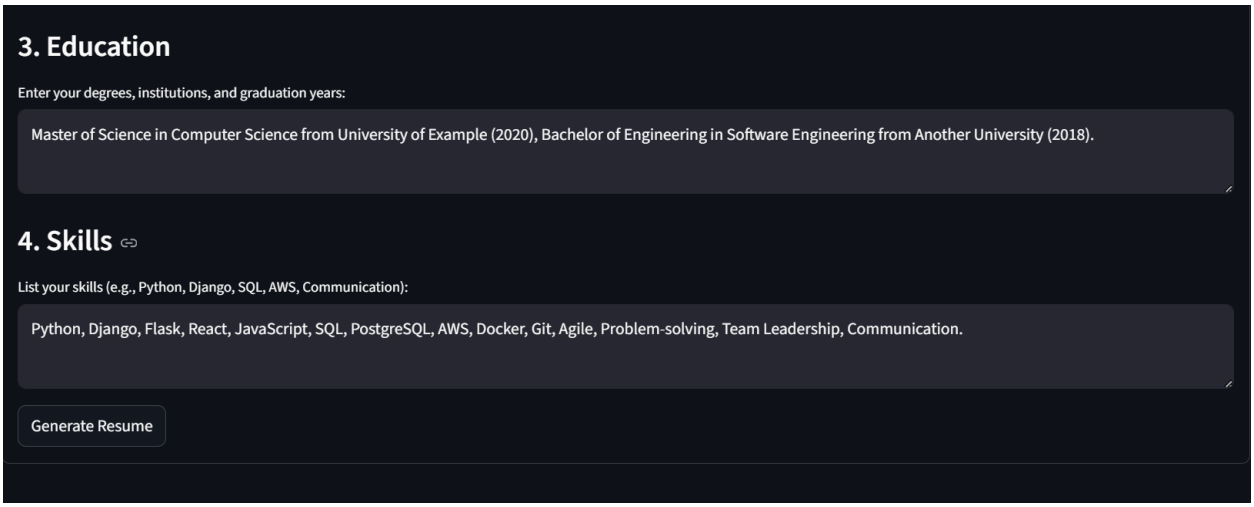
```python
    st.subheader("3. Education")
    education_input = st.text_area(
        "Enter your degrees, institutions, and graduation years:",
        "Master of Science in Computer Science from University of Example (2020), Bachelor of Engineering in Software Engineering from Anot
        height=80,
        key="education_input_area"
    )

    st.subheader("4. Skills")
    skills_input = st.text_area(
        "List your skills (e.g., Python, Django, SQL, AWS, Communication):",
        "Python, Django, Flask, React, JavaScript, SQL, PostgreSQL, AWS, Docker, Git, Agile, Problem-solving, Team Leadership, Communicatio
        height=80,
        key="skills_input_area"
    )
```

```python
    # Main submit button for the form
    generate_button = st.form_submit_button("Generate Resume")

    # If the generate button is clicked, process the data
    if generate_button:
        # Collect all user data
        user_data = {
            "Summary": summary_input,
            "Experience": [job for job in st.session_state.experience_inputs if job.strip() != ""], # Filter out empty job entries
            "Education": education_input,
            "Skills": skills_input,
        }

        with st.spinner("Generating your resume content... Please wait, this may take a few moments..."):
            generated_resume = generate_full_resume_content(user_data)

        # Store generated content in session state
        st.session_state.generated_resume_content = generated_resume

# --- Display Generated Resume ---
if 'generated_resume_content' in st.session_state and st.session_state.generated_resume_content:
    st.markdown("---") # Separator
    st.subheader("Generated Resume Content")

    # Display in a text area for easy copy
    st.text_area(
        "Your Resume (Markdown Format):",
```

```python
        st.session_state.generated_resume_content,
        height=600, # Make it tall enough to see content
        key="final_resume_display",
        help="Copy this content to a Markdown editor (like VS Code or Typora) to see formatted output."
    )

    # Row for copy and download buttons
    col_copy, col_download = st.columns([0.2, 0.8]) # Adjust column width for better button placement
    with col_copy:
        if st.button("Copy to Clipboard"):
            try:
                pyperclip.copy(st.session_state.generated_resume_content)
                st.success("Resume content copied to clipboard!")
            except pyperclip.PyperclipException as e:
                st.error(f"Could not copy to clipboard: {e}. Please copy manually.")
                st.info("On some Linux systems, you might need to install 'xclip' or 'xsel' for `pyperclip` to work.")

    with col_download:
        st.download_button(
            label="Download Resume (Markdown)",
            data=st.session_state.generated_resume_content,
            file_name="generated_resume.md",
            mime="text/markdown",
            help="Download the generated resume as a Markdown file."
        )

st.markdown("---")
st.markdown("Developed with 💔 using Streamlit & Google Generative AI.")
```

Output :


Your Information
Deploy
Provide details for each section. The AI will expand and format them professionally.
1. Professional Summary (Overall Experience/Goals)
Describe your professional background, key skills, and career aspirations:
Experienced software engineer with 5 years in Python and web development. Strong in building scalable applications and leading small teams. Looking for senior developer roles in AI/ML.
2. Experience (One job per box)
Enter details for each job role. Focus on responsibilities, achievements, and impact.
Job 1 Details:
Software Engineer at TechCorp (2022-Present): Developed backend APIs for e-commerce platform using Python/Django. Improved system performance by 30%. Mentored junior developers.


3. Education
Enter your degrees, institutions, and graduation years:
Master of Science in Computer Science from University of Example (2020), Bachelor of Engineering in Software Engineering from Another University (2018).
4. Skills
List your skills (e.g., Python, Django, SQL, AWS, Communication):
Python, Django, Flask, React, JavaScript, SQL, PostgreSQL, AWS, Docker, Git, Agile, Problem-solving, Team Leadership, Communication.
Generate Resume