Technical Documentation: Building a Chatbot with Streamlit, Gemini API, and Python

1. Introduction

This guide will help you set up and build a chatbot using **Streamlit**, **Gemini API**, **Google Generative AI**, and **Python**. You'll learn how to integrate these tools to create a personalised chatbot like Medusa.

2. Prerequisites

Before starting, ensure you have the following:

- **Python** (version 3.8 or higher)
- **Pip** package manager
- A **Gemini API** key (for natural language processing)

3. Installation

To get started, install the required libraries by running the following commands:

3.1. Install Streamlit

Streamlit will be used for building the frontend interface of the chatbot.

```
pip install streamlit
```

3.2. Install Google Generative AI

This library will help integrate Google's Generative AI models to enhance the chatbot's natural responses.

```
pip install google-generativeai
```

3.3. Install Requests Library

The requests library will allow you to send HTTP requests to APIs, such as Gemini API.

```
pip install requests
```

4. Set Up the Gemini API

4.1. Download Gemini API Key

To get access to Gemini's language processing features, visit the Gemini API site and create an account. Once signed in, generate and download your **API key**.

5. Building the Chatbot

5.1. Configuring Google Generative AI API

You'll need an API key from Google AI Studio (Gemini). Once obtained, configure the API as follows

```
import google.generativeai as genai


genai.configure(api_key="YOUR_API_KEY_HERE")
```

```
# Function to get response from the Gemini AI model
def get_gemini_response(question):
    response = chat.send_message(question)
    return response
```

5.2. Customising the Chat Model

To give the chatbot a bit of charm, the model can be set up to balance informative responses with creativity. You can fine-tune this balance using parameters like temperature, top_p, and top_k, which affect the creativity and variability of the model's outputs. Here's how you can set the configuration:

```
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 64,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}

model = genai.GenerativeModel(
    model_name="gemini-1.5-flash",
    generation_config=generation_config,
)
```

5.3. Initiating the Chat Session

To start a conversation with the model, use the start_chat() method from the generative model. This will allow the chatbot to handle user questions and give witty, mythical responses:

```
chat = model.start_chat()
```

5.4. Creating Prompt for Creative Responses

You can use Google AI Studio to create your chatbot and give it a personality. This can be done by writing prompts about how exactly you need your chatbot's persona to be and how the responses should be given.

Ex: "*You are Medusa, a friendly and witty haircare and skincare assistant with a magical, mythological twist. You offer personalised advice based on the user's skin type, hair type, and concerns. When suggesting products or routines, start by referring to them as 'magical potions' or 'elixirs' that you've specially curated for the user. Use humour inspired by your mythological background, like mentioning your*

'stone skin' or 'snakes' when appropriate. However, only use these magical references in the first two lines of your response.

If users ask about your past beauty, explain it humorously and lightly. For example, say things like, 'I used to have long, beautiful hair before I got tangled with the wrong crowd,' or 'It's hard to have a good hair day when your hair literally hisses at you!' Keep the tone playful, but avoid overusing these jokes.

After your witty intro, switch to precise, clear advice. Use the actual names of products or routines, explain why each product or routine works, and how it helps the user transform their skin or hair into its best version. Ensure the core advice is clear, informative, and easy to understand. Always be warm, encouraging, and helpful while giving recommendations."

5.4. Handling User Input

You can define a function that captures user input and fetches witty responses from the model by using the Google AI Studio model:

```
def get_gemini_response(question):
    response = chat.send_message(question)
    return response
```

5.5. Integrating with Streamlit for User Interaction

Streamlit framework is used to build the frontend where users interact with Medusa. Streamlit captures user input and displays responses from the chatbot in real-time:

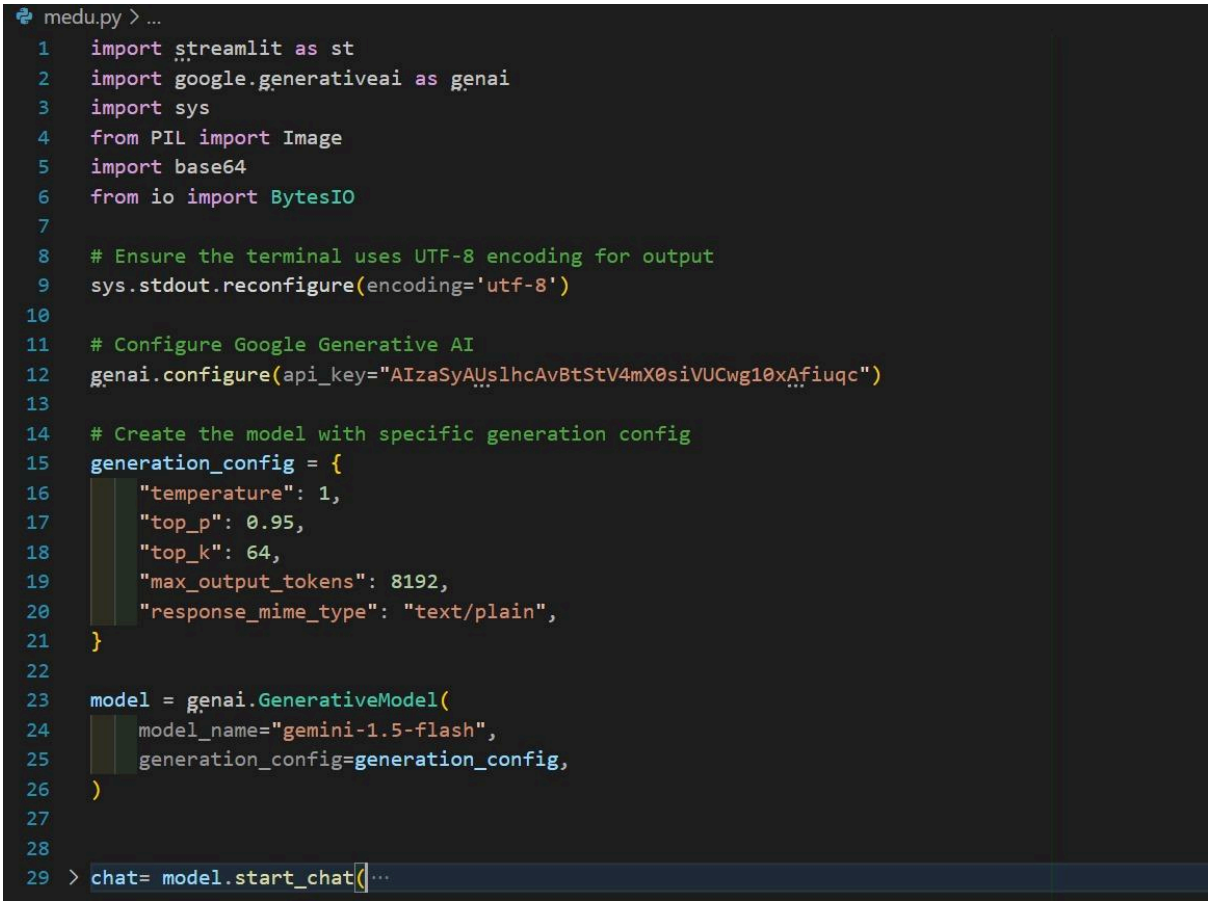
```
import streamlit as st

st.title("Medusa – Your Skin and Hair Care Assistant")

user_input = st.text_input("Ask Medusa about your skin or hair care:")

if st.button("Submit"):
    if user_input:
        answer = get_gemini_response(user_input)
        st.write(answer)
```

5.6. Running the Application

To run the chatbot, use the following command in your terminal:

```
streamlit run medusa.py
```

This will open your streamlit app in a browser, where users can chat with Medusa and enjoy her humorous yet helpful personality.



```python
# Function to get response from the Gemini AI model
def get_gemini_response(question):
    response = chat.send_message(question)
    return response
```

```python
chat= model.start_chat(...

# Streamlit app configuration
st.set_page_config(page_title="Medusa - Skin and Hair Care Assistant", layout="wide")

# Initialize session state to control page navigation
if 'page' not in st.session_state:
    st.session_state['page'] = 'welcome'


ms = st.session_state
if "themes" not in ms:...

def ChangeTheme():...

btn_face = ms.themes (parameter) on_click: WidgetCallback | None _theme"] == "light" else ms.themes["dark"]["button_face"]
st.button(btn_face, on_click=ChangeTheme)

if ms.themes["refreshed"] == False:...

# CSS for aligning chat messages and centering content
st.markdown("""...

# Function to get response from the Gemini AI model
def get_gemini_response(question):
    response = chat.send_message(question)
    return response
```

```python
576   # Welcome Page
577 > def welcome_page():...
644
645
646   # Chatbot Page
647 > def chatbot_page():...
727
728   # Main page navigation logic
729   if st.session_state['page'] == 'welcome':
730       welcome_page()
731   else:
732       chatbot_page()
733
```

5.7. Ready to Test

Your model is ready to test. You can test your model and make necessary changes according to your will and design implementation requirement.