Overview of the Code of Surya’s Chatbot

This Streamlit web app integrates Google’s generative AI services to create a chatbot called **"Surya's AI Chatbot"**. The chatbot can provide conversational responses based on user input, and it includes date-related functionalities (like providing the current day, tomorrow’s date, etc.). Additionally, the app supports customization of the user interface (UI) by allowing changes to the background image, font styles, and font sizes. The core functionality of the chatbot uses Google's **Gemini-1.5 Flash** model to generate AI-based responses.

Imports and Dependencies

The code begins by importing the necessary libraries:

- streamlit as st: This is the main library for building the web application interface. It allows you to display text, create input forms, and add interactivity to the app.
- google.generativeai as genai: This is the library to interact with Google's generative AI. It allows the app to generate text responses based on the context provided by the user.
- datetime and timedelta: These modules are used to manipulate and calculate dates. The datetime.now() function retrieves the current date and time, and timedelta is used to add or subtract days from a given date.

Google Generative AI Configuration

To use Google’s generative AI model, the app requires an API key. The key surya_api_key is provided, which authenticates the app with Google’s AI services. The genai.configure() function is called to set up the API key for use throughout the application, ensuring that all interactions with the AI model are authenticated.

Model Initialization

The code initializes the generative AI model using the specific model identifier "gemini-1.5-flash". This model is designed to generate conversational responses based on the input provided. By assigning this model to the model variable, the app is ready to generate responses for users' queries.

Date Handling with get_date_info

The function get_date_info(user_input) is designed to handle user queries related to dates. It checks if the input contains keywords like "today", "tomorrow", "yesterday", "day", or "week", and based on these keywords, it provides the appropriate date-related information.

- For queries like "today", it will return the current date.
- For "tomorrow", it will calculate and return the date of the next day.
- For "yesterday", it returns the previous day's date.
- For queries like "day of the week" or "next Thursday", it will return the name of the day or the date of the upcoming Thursday, respectively.

This function ensures that the app can respond to common date-related questions and can be expanded to handle additional date queries if needed.

Setting a Custom Background Image

The set_background_image(url) function is used to set a custom background image for the Streamlit app. It takes a URL to an image as an argument, and the app applies this image as the background using inline CSS. The background image is set to cover the entire screen, positioned centrally, and it does not repeat, giving the app a visually appealing backdrop.

Customizing Font Style and Size

Two functions are responsible for customizing the font style and size in the app:

1. **set_font_style_and_size(font_family, font_size)**: This function allows you to change the font style and size for the entire app. You can pass the desired font_family (e.g., Arial) and font_size (in pixels) to modify how text appears throughout the app’s interface. This is done by applying custom CSS to the Streamlit app's main container.
2. **set_text_area_font_size(font_size)**: This function specifically changes the font size of the text areas used for user input. It allows you to modify the font size of any text area component within the app for better visibility and readability.

Both functions are customizable, enabling users to adjust the appearance of the app to their preferences.

Streamlit App Structure

The app is structured with the following key components:

1. **Sidebar Navigation**: The sidebar allows users to select the functionality they want. In this case, there is a single option: **"Surya's AI Chatbot"**. If the user selects this option, the corresponding interface for interacting with the AI chatbot is displayed.
2. **Conversation History**: The chatbot stores all interactions in st.session_state.conversation_history, ensuring that the conversation persists across different inputs. The conversation is displayed as a list of messages, with the user’s inputs prefixed by "You:", and the AI’s responses prefixed by "Surya's AI Response:".
3. **User Input**: The user can type their question into a text input box. When the user submits a query by clicking the "Submit" button, the app processes the query. If the query is related to a date, the app provides an appropriate response using the get_date_info() function. Otherwise, it sends the entire conversation history to the generative AI model to generate a relevant response.

4. **AI Response Generation**: If the query does not involve dates, the app uses the model.generate_content() function to generate a response from Google's AI model. The entire conversation history is passed to the model, and the response is appended to the conversation history. The response is displayed on the UI, and if the word "Subplots" appears, it is highlighted by making it bold.
5. **Message ID Tracking**: The app uses a message_id to track each user's input and ensure that each new input box has a unique identifier. This helps in rendering the conversation history and dynamically creating new input fields.

Error Handling

The AI response generation is wrapped in a try-except block to handle any errors that might occur during the generation process (e.g., network issues, API limits). If an error occurs, an error message is displayed to the user, and no AI response is generated.

Running the Streamlit App

Finally, the main() function is called to run the Streamlit app. It sets up the background image, handles the navigation and chatbot functionality, and dynamically updates the conversation history as the user interacts with the chatbot.

Conclusion

This code is a simple yet customizable implementation of a chatbot app using Google's generative AI and Streamlit. It combines AI-generated conversational responses with practical date handling features, while offering customization options like background images and font styles to enhance user experience.

Code:

```
import streamlit as st
import google.generativeai as genai
from datetime import datetime, timedelta

# Configure the API key for Google Generative AI
surya_api_key = "AIzaSyCu2susiqMRwXoSPdgreHNMcemLXMOFY4M"  # Your API key
genai.configure(api_key=surya_api_key)

# Initialize the generative model for Google AI
model = genai.GenerativeModel("gemini-1.5-flash")

# Function to get date-related responses for the AI Chat
def get_date_info(user_input):
  """Return date-related information based on user query."""
  today_date = datetime.now()

  if 'today' in user_input.lower():
     return f"Today's date is {today_date.strftime('%Y-%m-%d')}."

  if 'tomorrow' in user_input.lower():
     tomorrow = today_date + timedelta(days=1)
```

```
        return f"Tomorrow's date is {tomorrow.strftime('%Y-%m-%d')}."

    if 'yesterday' in user_input.lower():
        yesterday = today_date - timedelta(days=1)
        return f"Yesterday's date was {yesterday.strftime('%Y-%m-%d')}."

    if 'day' in user_input.lower() and 'week' in user_input.lower():
        return f"Today is {today_date.strftime('%A')}."

    if 'next thursday' in user_input.lower():
        current_day = today_date.weekday()
        days_until_thursday = (3 - current_day) if current_day <= 3 else (10 - current_day)
        next_thursday = today_date + timedelta(days=days_until_thursday)
        return f"The next Thursday will be on {next_thursday.strftime('%Y-%m-%d')}."

    return None


# Function to add background image for the main app
def set_background_image(url):
    st.markdown(
        f"""
        <style>
        .stApp {{
            background-image: url('{url}');
            background-size: cover;
            background-position: center;
            background-repeat: no-repeat;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )


# Function to set custom font style and size in code
def set_font_style_and_size(font_family, font_size):
    st.markdown(
        f"""
        <style>
        .stApp {{
            font-family: '{font_family}';
            font-size: {font_size}px;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )


# Function to set font size for text_area
def set_text_area_font_size(font_size):
    st.markdown(
        f"""
        <style>
        .stTextArea textarea {{
            font-size: {font_size}px;
        }}
        </style>
        """,
        unsafe_allow_html=True
    )


# Streamlit app
def main():
    # Set background image for main app

set_background_image("https://t3.ftcdn.net/jpg/06/27/85/70/360_F_627857071_bREuJgK1MTU
5YcTPTfgJPOYZ86F1l421.jpg")
```

```
  # Dynamic font size slider for code output and text_area

  # font_size = st.slider("Select Font Size for Text Area and Code", min_value=10,
max_value=50, value=18)  # Dynamic font size



  # Apply the selected font size to code blocks and text_area

  #set_font_style_and_size("Arial", font_size)

  #set_text_area_font_size(font_size)



  # Sidebar for navigation between functionalities

  option = st.sidebar.selectbox("Choose functionality", ("Surya's AI Chatbot",))



  # AI Chat

  if option == "Surya's AI Chatbot":

      st.header("Surya's AI Chatbot")

      st.write("This is a chatbot created by Surya Vishal")



      # Initialize conversation history in session state

      if "conversation_history" not in st.session_state:

          st.session_state.conversation_history = []

      # Initialize message_id to track unique question inputs

      if "message_id" not in st.session_state:

          st.session_state.message_id = 0
```

```
    # Display the conversation history
    for message in st.session_state.conversation_history:
        if message.startswith("You:"):
            st.markdown(f"**{message}**")
        else:
            st.markdown(f"{message}")

    # Input and button for AI Chat
    user_input = st.text_input(
        "**Enter your question:**",
        key=f"user_input_{st.session_state.message_id}",
        placeholder="Type your question here..."
    )

    if st.button("Submit") and user_input:
        # Add user input to the conversation history
        st.session_state.conversation_history.append(f"You: {user_input}")

        # Check for date-related responses
        date_response = get_date_info(user_input)
        if date_response:
            st.session_state.conversation_history.append(f"**Surya's AI Response:**
{date_response}")
```

```python
        else:

            # Generate a response using the AI model

            try:

                context = "\n".join(st.session_state.conversation_history)

                response = model.generate_content(f"{context}\nAI:")

                response_text = response.text  # Adjust this if necessary


                # Highlight "Subplots" if mentioned

                if "Subplots" in response_text:

                    response_text = response_text.replace("Subplots", "**Subplots**")


                st.session_state.conversation_history.append(f"**Surya's AI Response:** {response_text}")

            except Exception as e:

                st.error(f"Error generating response: {e}")

                return


        # Increment message_id to generate a new input box for next question

        st.session_state.message_id += 1


if __name__ == "__main__":

    main()
```