PHASE 4:DEVELOPMENT PART 2

Introduction:

The project's primary goal is to create a chatbot and make it accessible through a web interface. The chatbot is designed to facilitate text-based conversations with users, offering responses to their inquiries or prompts. To achieve this, we will use the Flask web framework to construct a web application that hosts the chatbot, bridging the gap between the user and the chatbot's functionalities.

Project Setup:

1. Setting up the Development Environment

Begin by ensuring that your development environment is prepared for the project. Verify the presence of essential tools and libraries, with a specific focus on Python and Flask. Install them if necessary to establish a suitable working environment**.**

2. Saving the Chatbot Code

The core of the project resides in a Python file, typically named chatbot.py. This file houses the chatbot's functionality, where it can interpret and respond to user messages. The chatbot should incorporate a function that receives user input and generates appropriate responses based on its internal logic.

Flask Integration:

3. Installing Flask in Google Colab

In the Google Colab environment, Flask must be installed to facilitate the creation of the web application. Use a straightforward pip installation command to add Flask to your project's dependencies.

4. Creating a Flask Web App

Within a Python file named app.py, commence the creation of a Flask web application. This application will serve as the central hub for handling user requests, communicating with the chatbot's logic, and providing a platform for the web interface.

5. Handling Routes and Endpoints

Define the routes and endpoints within the Flask app to orchestrate user interactions. These routes serve various purposes, including rendering the web interface and managing the chatbot interactions, such as receiving user messages and delivering chatbot responses.

Web Interface

6. Creating the Web Interface (HTML)

An interactive web interface is pivotal for enabling users to communicate with the chatbot. Create an HTML file (commonly named index.html) that serves as the user interface. This HTML file should incorporate input fields for user messages, a chat history display, and elements that create a user-friendly chat experience

Interaction with the Chatbot

7. Sending User Messages

Through the web interface, users have the capability to input their messages. These messages are then dispatched to the Flask app for processing and communication with the chatbot.

8. Chatbot Response

Within the Flask app, user messages are processed and delivered to the chatbot's logic. The chatbot generates responses based on the user's input and returns these responses to the Flask app for transmission to the web interface.

9. Displaying Responses

The web interface plays a critical role in conveying chatbot responses to users. Responses from the chatbot are dynamically displayed, fostering a conversational flow within the interface and creating a seamless user experience.

Running the Application

10. Starting the Flask App

With your Flask application's code in place, it is time to initiate the app. This step involves running your Flask app within your Google Colab environment or on your local machine, ensuring it is accessible and operational**.**

11. Accessing the Web Interface

To engage with the chatbot, users access the web interface by opening a web browser and navigating to the designated URL, typically http://localhost:5000. This is where they interact with the chatbot and initiate conversations.

Code for the chatbot by integrating it into a web app using flask:

```
!pip install flask

# Import necessary modules

from flask import Flask, render_template, request

from chatbot import generate_response  # Assuming you have this function

app = Flask(__name__)

# Route to handle chatbot logic

@app.route('/ask', methods=['POST'])

def ask():

    user_message = request.form['user_message']

    chatbot_response = generate_response(user_message)

    return chatbot_response

# Main route to serve the HTML page

@app.route('/')
```

```
def home():
    return render_template('index.html')


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Now, let's create the index.html file for the chat interface:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Chatbot Web Interface</title>
</head>
<body>
    <h1>Chatbot</h1>
    <div id="chatbox">
        <div id="chat"></div>
        <input type="text" id="user_input" placeholder="Type your message..."
autocomplete="off">
        <button onclick="sendMessage()">Send</button>
    </div>
    <script>
        function sendMessage() {
            const userMessage = document.getElementById('user_input').value;
            document.getElementById('chat').innerHTML +=
'<div><strong>You:</strong> ' + userMessage + '</div>';
            document.getElementById('user_input').value = '';

            fetch('/ask', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: 'user_message=' + userMessage
            })
            .then(response => response.text())
            .then(data => {
                document.getElementById('chat').innerHTML +=
'<div><strong>Chatbot:</strong> ' + data + '</div>';
            });
        }
    </script>
</body>
</html>
```

Conclusion:

This documentation elucidates the step-by-step process for constructing a chatbot and embedding it within a web application using Flask. It encompasses project setup, the creation of the web interface, and the orchestration of interactions between users and the chatbot. By adhering to these guidelines, you can create a web-based chatbot accessible to users, enhancing their conversational experiences.