CHATBOT USING PYTHON

Phase-4 : Development part 2

Project Title: *chatbot using python*

OBJECTIVE:

In this part you will continue building your project.

Continue building the chatbot by integrating it into a web app using Flask.

DATASET LINK:

https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

DEVELOPEMENT PART:

To integrate Flask into a web app for building a chatbot, you'll need to follow these steps:

1. Set up a Flask project:

Start by creating a new Flask project. Install Flask using pip and create a new Python file, let's call it app.py, where you will define your Flask app.

2. Define routes:

In your app.py file, define the routes for your web app. You'll need at least two routes: one for the home page where the chatbot interface will be displayed, and another to handle the chatbot conversation.

Python:

```
from flask import Flask, render_template, request
```

```
app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/chat', methods=['POST'])
def chat():
    # Handle the chatbot conversation logic
    user_message = request.form['user_message']
    # Process user_message and generate chatbot response
    bot_response = "This is the chatbot response."

    return {'response': bot_response}
```

3. Create templates:

Create an index.html template file in a templates folder. This file will contain the HTML structure of your chatbot interface.

HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Chatbot</title>
</head>
<body>
```

```
    <h1>Chatbot Interface</h1>
    <div id="chat-container">
      <div id="chat-log"></div>
      <form id="chat-form">
        <input type="text" id="user-message" placeholder="Type your message..." autocomplete="off">
        <button type="submit">Send</button>
      </form>
    </div>

    <script src="{{ url_for('static', filename='script.js') }}"></script>
  </body>
  </html>
```

4. Create static files:

Create a static folder where you'll store the static files (CSS, JavaScript) for your web app. In this case, create a script.js file to handle the chatbot conversation.

JavaScript:

```
document.addEventListener('DOMContentLoaded', () => {
  const chatForm = document.querySelector('#chat-form');
  const chatLog = document.querySelector('#chat-log');
  const userMessageInput = document.querySelector('#user-message');

  chatForm.addEventListener('submit', (e) => {
    e.preventDefault();
    const userMessage = userMessageInput.value;
    appendMessage(userMessage, 'user');
    userMessageInput.value = '';

    fetch('/chat', {
      method: 'POST',
      body: JSON.stringify({ user_message: userMessage }),
      headers: {
        'Content-Type': 'application/json',
      },
    })
      .then(response => response.json())
      .then(data => {
        const botResponse = data.response;
        appendMessage(botResponse, 'bot');
      })
      .catch(error => console.error('Error:', error));
  });

  function appendMessage(message, sender) {
    const messageElement = document.createElement('div');
    messageElement.classList.add(sender);
    messageElement.innerText = message;
    chatLog.appendChild(messageElement);
  }
});
```

5. Run the Flask app:

Start the Flask development server using the command flask run. Open your web browser and navigate to http://localhost:5000 to see the chatbot interface.

Flask will handle the route for the home page and render the index.html template. When the user submits a message, the JavaScript code in script.js will capture the message, send it to the /chat route using an AJAX request, and display the response in the chat log.

6. Implement chatbot logic:

In the /chat route, process the user's message and generate the chatbot's response. You can use a natural language processing library like NLTK, SpaCy, or transformers to handle the conversation logic. Depending on your chatbot's complexity, you may need to integrate a machine learning model or use a pre-trained chatbot model.

Python:

```
# Example implementation using a simple rule-based chatbot
def chat():
    user_message = request.json['user_message']
    bot_response = get_bot_response(user_message)
    return {'response': bot_response}

def get_bot_response(user_message):
    # Implement your chatbot logic here
    # You can use if-else statements or more advanced techniques like machine learning models
    if user_message.lower() == 'hi':
        return 'Hello!'
    elif user_message.lower() == 'how are you?':
        return 'I am good, thank you!'
    else:
        return 'I am sorry, I did not understand that.'
```

# Note: This is a simple example for illustration purposes. You can make your chatbot more sophisticated based on your requirements.

7. Deploy your Flask app:

Once you have completed the integration and tested the chatbot locally, you can deploy your Flask app to a web server or a cloud platform like Heroku, AWS, or Google Cloud Platform. Refer to the respective documentation for deploying Flask apps.

By following these steps, you can integrate Flask into a web app for building a chatbot. Remember to handle any potential errors, sanitize user inputs, and continuously improve your chatbot's responses based on user feedback.