Create a chatbot in Python

Phase 4

Introduction:

Building a chatbot and integrating it into a web app using Flask is a common and practical application of chatbot technology. Flask is a lightweight Python web framework that's well-suited for creating web applications, and you can easily integrate a chatbot into a Flask-based web app.

1. Install Required Libraries:

Make sure you have Flask and any other necessary libraries installed. You may want to use a chatbot framework or library, like ChatterBot or Rasa, to simplify chatbot development. Install them using pip:

```
bash

pip install Flask

pip install chatterbot
```

2. Create a Flask Web App:

Create a Flask web app by creating a Python file, e.g., `app.py`. Here's a basic example:

```
python

from flask import Flask, render_template, request


app = Flask(__name__)


@app.route('/')

def index():

    return render_template('index.html')
```

```
if __name__ == '__main__':
    app.run()
```

3. Create HTML Template:

Create an HTML template for your chat interface. You can use this template to collect user input and display chatbot responses. Save this template as `index.html` in a `templates` directory in your project folder.

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Chatbot Example</title>
</head>
<body>
    <h1>Chatbot Example</h1>
    <div id="chatbox">
        <div id="chatlog">
            <!-- Chat messages will appear here -->
        </div>
        <input type="text" id="user_input" placeholder="Type your message...">
        <button id="send">Send</button>
    </div>
</body>
</html>
```

4. Implement Chatbot Logic:

Implement your chatbot logic in Python. You can use ChatterBot, Rasa, or any other chatbot framework of your choice. Define a function in your Flask app that handles the chatbot interaction, taking user input and returning bot responses.

5. Handle User Input:

Add JavaScript to your HTML template to handle user input and display chatbot responses. You can use AJAX or WebSocket to communicate with your Flask app. Here's a simple example using jQuery:

```
html
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    $(document).ready(function () {
        $("#send").click(function () {
            var user_input = $("#user_input").val();
            $("#chatlog").append("<p>User: " + user_input + "</p>");
            $("#user_input").val("");
            $.ajax({
                type: "POST",
                url: "/get_response",
                data: JSON.stringify({ user_input: user_input }),
                contentType: "application/json; charset=utf-8",
                dataType: "json",
                success: function (data) {
                    $("#chatlog").append("<p>Bot: " + data.bot_response + "</p>");
                }
            });
        });
    });
</script>
```

6. Create a Flask Route for Chatbot Interaction:

In your Flask app, create a route to handle chatbot interactions. This route should take user input, process it, and return the chatbot's response.

```
python

from flask import request, jsonify


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

def get_response():

    user_input = request.json['user_input']

    # Process user_input and get the chatbot's response

    bot_response = get_chatbot_response(user_input)

    return jsonify({'bot_response': bot_response})
```

7. Run the Flask App:

Run your Flask app using the command `python app.py`. You can access your chatbot web app at `http://localhost:5000` in your web browser.

8. Test and Refine:

Test your chatbot and make any necessary refinements to improve its functionality and user experience.

These are the basic steps to integrate a chatbot into a Flask web app. Depending on your chatbot's complexity, you may need to add features like natural language processing, user authentication, and more to create a fully functional and secure chat application.

Program:

```
import pickle
from flask import Flask, request, jsonify
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from spellchecker import SpellChecker  # You may need to install this library

app = Flask(__name)

# Load your dialog dataset
dataset = pd.read_csv('dialog.csv')

# Preprocess the dataset
tokenizer = Tokenizer()
tokenizer.fit_on_texts(dataset['a'])
total_words = len(tokenizer.word_index) + 1

# Tokenize and pad the sequences
input_sequences = []
for line in dataset['a']:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i + 1]
        input_sequences.append(n_gram_sequence)
```

```
max_sequence_length = max([len(x) for x in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length, padding='pre')

# Separate input and target sequences
X = input_sequences[:, :-1]
y = input_sequences[:, -1]

# Create and compile the model
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_length - 1))
model.add(LSTM(150))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Train the model with a specified number of epochs
model.fit(X, y, epochs=10)  # You can specify the number of epochs here

# Save the trained model to a pkl file
with open('model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)

# Initialize a spell checker
spell = SpellChecker()

@app.route('/chatbot', methods=['POST'])
def chatbot():
    user_input = request.json['user_input']  # Assuming you're receiving JSON input

    # Load the trained model from the pkl file
```

```
    with open('model.pkl', 'rb') as model_file:
        model = pickle.load(model_file)

    # Preprocess user input
    user_input = user_input.lower()  # Convert to lowercase
    user_input = spell.correction(user_input)  # Correct spelling
    input_sequence = tokenizer.texts_to_sequences([user_input])[0]
    input_sequence = pad_sequences([input_sequence], maxlen=max_sequence_length - 1, padding='pre')

    # Generate a response using the trained model
    response_sequence = []
    for _ in range(max_sequence_length - 1):
        predicted_word_index = model.predict_classes(input_sequence, verbose=0)
        predicted_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted_word_index:
                predicted_word = word
                break
        input_sequence = pad_sequences([input_sequence.tolist() + [predicted_word_index]],
maxlen=max_sequence_length - 1, padding='pre')
        response_sequence.append(predicted_word)

    response = ' '.join(response_sequence)

    return jsonify({'response': response})
if _name_ == '_main_':
    app.run(debug=True)
```

Project By:

NAME: Muralidharan.P

DEPT: CSE III YEAR

COLLEGE: Maha Bharathi Engineering College

GROUP: IBM -GROUP-5

REG NO:621421104037