**PHASE-4:**

**The chatbot by integrating it into a web app using Flask.**

**Introduction:**

Integrating Your Chatbot into a Flask Web AppIn this next phase of our project, we embark on a fascinating journey to enhance the accessibility and user-friendliness of your chatbot. By integrating your chatbot into a web application powered by Flask, we are poised to provide an engaging and interactive experience for users across the web.

**1. Set up a Flask App:**

Start by creating a new Flask application or using an existing one. You can install Flask using pip if you haven't already.

python

from flask import Flask, render\_template, request, jsonify

app = Flask(\_\_name\_\_)

**2. Create a Chat Interface:**

Design the HTML template for your chat interface. You'll likely have an input box for user messages and a display area for the chat history.

html

<!DOCTYPE html>

<html>

<head>

<title>Chatbot</title>

</head>

<body>

<div id="chat-container">

<div id="chat"></div>

<input type="text" id="user-input" placeholder="Type a message...">

<button id="send">Send</button>

</div>

</body>

</html>

**3. Create a Route for the Chatbot:**

Create a route that will handle user requests and communicate with the chatbot. For example:

python

@app.route('/')

def chatbot\_page():

# Fetch user input from the request

user\_input = request.args.get('user\_input')

# Use the chatbot to generate a response

bot\_response = chatbot\_function(user\_input)

return render\_template('chatbot.html', user\_input=user\_input, bot\_response=bot\_response)

**4. Add User Interface:**

Integrate your chatbot into the HTML templates. You can use JavaScript for handling user input and displaying chat responses.

**5. Handle User Input:**

Set up a route to receive user input from the web interface. You can use the `request` object to get user messages.

**6. Integrate Chatbot Logic:**

Implement the logic for your chatbot. You can use Python to process user input and generate chatbot responses.

**7. Return Responses:**

Send chatbot responses back to the web interface for display. You can use JavaScript to update the chat interface with the responses.

**8. Testing:**

Test your web app locally to ensure everything is working as expected

**Ex:**

def clean\_text(text):

text = unicode\_to\_ascii(text.lower().strip())

text = re.sub(r"i'm", "i am", text)

text = re.sub(r"\r", "", text)

text = re.sub(r"he's", "he is", text)

text = re.sub(r"she's", "she is", text)

text = re.sub(r"it's", "it is", text)

text = re.sub(r"that's", "that is", text)

text = re.sub(r"what's", "that is", text)

text = re.sub(r"where's", "where is", text)

text = re.sub(r"how's", "how is", text)

text = re.sub(r"\'ll", " will", text)

text = re.sub(r"\'ve", " have", text)

text = re.sub(r"\'re", " are", text)

text = re.sub(r"\'d", " would", text)

text = re.sub(r"\'re", " are", text)

text = re.sub(r"won't", "will not", text)

text = re.sub(r"can't", "cannot", text)

text = re.sub(r"n't", " not", text)

text = re.sub(r"n'", "ng", text)

text = re.sub(r"'bout", "about", text)

text = re.sub(r"'til", "until", text)

text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", text)

text = text.translate(str.maketrans('', '', string.punctuation))

text = re.sub("(\\W)"," ",text)

text = re.sub('\S\*\d\S\*\s\*','', text)

text = "<sos> " + text + " <eos>"

return text.

**9. Deployment:**

When you're satisfied with the functionality, you can deploy your Flask app on a web server so that it's accessible online. Popular platforms for deployment include Heroku, AWS, and GCP.

Here's a basic example of what the structure of your Flask app might look like:

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():

user\_input = request.form['user\_input']

# Process user\_input and get chatbot response

chatbot\_response = get\_chatbot\_response(user\_input)

return chatbot\_response

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**10. Create HTML Template**:

Create an HTML template (e.g., `chatbot.html`) to display the chat interface and show user input and chatbot responses.

html

<!DOCTYPE html>

<html>

<head>

<title>Chatbot Web App</title>

</head>

<body>

<h1>Chatbot</h1>

<div>

<p><strong>User:</strong> {{ user\_input }}</p>

<p><strong>Chatbot:</strong> {{ bot\_response }}</p>

</div>

<form action="/" method="GET">

<input type="text" name="user\_input">

<input type="submit" value="Send">

</form>

</body>

</html>

1. **Run Your Flask App:**

Add the following lines at the end of your `app.py` to run the app:

python

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

**12. Run Your Flask App:**

Run your Flask app by executing `python app.py` in your terminal.

**13. Access Your Chatbot Web App:**

Open a web browser and go to `http://localhost:5000` (by default) to interact with your chatbot through the web interface.

def home():

return render\_template('index.html')

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

def chat():

user\_input = request.form['user\_input']

# Process user\_input and get chatbot response

chatbot\_response = get\_chatbot\_response(user\_input)

return chatbot\_response

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Conclusion:**

This concludes Phase 4 of your project, where you successfully integrated your chatbot into a web app using Flask. Your chatbot is now accessible via a user-friendly interface, and you can continue to improve and expand its capabilities as needed.