Chatbot Project Documentation

Phase 5: Project Documentation & Submission :

Table of Contents :

1. Introduction :

This project is aimed at developing a web-based chatbot using the Flask framework. The chatbot interacts with users and provides responses based on a dataset of question-answer pairs. This documentation provides an overview of the project's development and usage.

2. Problem Statement :

The problem is to create a chatbot that can engage in conversations with users and provide meaningful responses. The chatbot should be able to handle user input and generate responses based on the provided dataset.

3. Design Thinking Process :

Our design thinking process involves the following steps:

- Define the problem statement and project goals.
- Choose the appropriate technology stack (Flask, Python, etc.).

- Develop the chatbot logic for generating responses.
- Create a user interface for interacting with the chatbot.
- Test and iterate on the chatbot's performance.

4. Project Phases :

The project was divided into the following phases:

- Data collection: Loading question-answer pairs from "dialogs.txt."
- Flask application setup: Creating a web application using Flask.
- Chatbot logic: Implementing the chatbot logic to generate responses.
- Web interface: Developing a simple HTML interface for users to interact with the chatbot.

5. Libraries and Tools :

- Flask: A micro web framework for Python.
- Pandas: Used for data manipulation and loading the dataset.
- HTML and JavaScript: For creating the web interface.

6. NLP Integration :

The project does not heavily rely on Natural Language Processing (NLP) techniques as it provides responses based on predefined question-answer pairs. However, you can extend the chatbot's functionality with NLP techniques for more advanced interactions.

7. Chatbot Interaction :

The chatbot interacts with users through a web interface. Users can type messages, and the chatbot responds with answers based on the dataset.

8. Innovative Approaches :

To ensure that responses are not repeated, the chatbot keeps track of used responses and provides unique responses from the dataset. If all responses are used, it resets the list.

9. Code Files :

The code for the Flask application, including the chatbot logic and web interface, is available in the "app.py" file. The HTML code for the web interface is provided in the "index.html" file.

10. README Instructions :

Please refer to the README file for instructions on how to run the code and any necessary dependencies.

11. Dataset Description :

The dataset used for this project is stored in the "dialogs.txt" file. It consists of question-answer pairs separated by tabs ("\t"). The dataset is used by the chatbot to generate responses.

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

Python code for integrating it into a web app using Flask :

```
# Import necessary libraries
from flask import Flask, render_template, request, jsonify

app = Flask("Chatbot")

# Load data from "dialogs.txt" file
data = []

with open("dialogs.txt", "r", encoding="utf-8") as file:
    for line in file:
        parts = line.strip().split('\t')
        if len(parts) == 2:
            question, answer = parts
            data.append({"question": question, "answer": answer})

response_index = 0
```

```
def get_chatbot_response(user_input):
    global response_index
    if response_index < len(data):
        response = data[response_index]["answer"]
        response_index += 1
    else:
        response = "I have no more responses."
    return response

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

@app.route('/chat', methods=['POST'])
def chat():
    try:
        user_input = request.form['user_input']
        response = get_chatbot_response(user_input)
        return jsonify({"response": response})
    except Exception as e:
        return jsonify({"error": str(e)})

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

HTML code for integrating it into a web app using Flask :

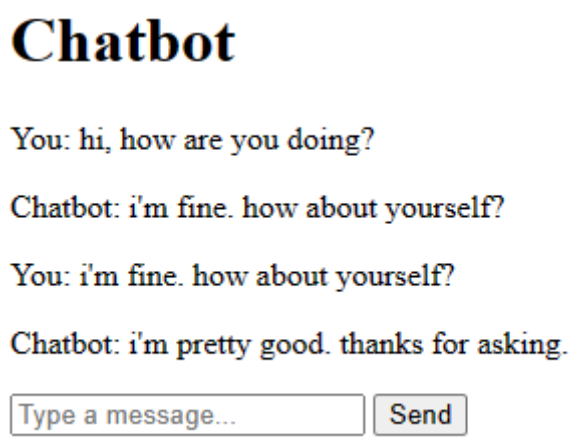
```
<!DOCTYPE html>
<html>
<head>
    <title>Chatbot</title>
</head>
<body>
    <h1>Chatbot</h1>
    <div id="chatbox">
        <!-- Chatbot responses will be displayed here -->
    </div>
    <input type="text" id="user_input" placeholder="Type a message..." />
    <button onclick="sendMessage()">Send</button>
    <script>
```

```
        function sendMessage() {
            var user_input = document.getElementById("user_input").value;
            document.getElementById("user_input").value = "";
            document.getElementById("chatbox").innerHTML += "<p>You: " +
user_input + "</p>";

            fetch('/chat', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ user_input: user_input })
            })
            .then(response => response.text())
            .then(response => {
                displayChatbotResponse(response);
            })
            .catch(error => {
                console.error(error);
            });
        }

        function displayChatbotResponse(response) {
            document.getElementById("chatbox").innerHTML += "<p>Chatbot: " +
response + "</p>";
        }
    </script>
</body>
</html>
```

Output :



Conclusion :

this chatbot project built on Flask offers a user-friendly interface for engaging in conversations. The dataset-driven responses can be customized, making it adaptable for a range of applications. With sequential responses, users receive unique interactions. This project provides a simple foundation for more advanced chatbots and showcases Flask's versatility for web-based chat applications. As chatbots continue to evolve, this project serves as a starting point for those exploring their potential applications and capabilities.