ChatBot Using Python Documentation

Problem Statement:

Develop a chatbot integrated into a web application that can answer user queries, provide information, and perform tasks based on natural language input.

Design Thinking Process:

1. **Empathize:** Understand the needs of the users and identify their pain points. Consider the context in which the chatbot will be used.

2. **Define:** Clearly define the objectives and scope of the chatbot. Determine the primary use cases and the goals it should achieve.

3. **Ideate:** Brainstorm ideas for the chatbot's functionality, considering both the user's and the business's needs.

4. **Prototype:** Create a rough prototype of the chatbot's user interface and the conversation flow to visualize its interactions.

5. **Test:** Gather feedback by testing the prototype with potential users to refine the design and functionality.

6. **Develop:** Implement the chatbot and web application based on the feedback and design decisions made during the previous steps.

7. **Deploy**: Make the chatbot accessible to users, either through a website or other channels like messaging platforms.

8. **Iterate:** Continuously gather user feedback and improve the chatbot based on real-world usage.

Phases of Development:

1. **Data Collection:** Gather datasets and resources necessary for training the chatbot. These may include text data, FAQs, and pre-trained language models.

2. **NLP Techniques Integration:**

A. **Tokenization**: Break text input into words or tokens.

B. **Named Entity Recognition (NER)**: Identify entities in user queries (e.g., locations, dates, names).

C. **Intent Recognition**: Determine the user's intent behind their query.

D. **Response Generation**: Craft responses based on user inputs and intent.

3. **Model Training**: Train or fine-tune a natural language processing model (e.g., GPT-3, BERT) on your specific dataset or tasks.

4. **Web Application Development**: Use Flask to create a web application with routes and templates for user interactions. Integrate the chatbot component into the app.

5. **User Interaction**: Define how users will initiate and interact with the chatbot within the web application, e.g., a chat interface.

6. **Response Generation**: Use the NLP model to generate responses to user queries. Handle common user intents, such as providing information, answering questions, or executing commands.

7. **Error Handling**: Implement error handling for cases when the chatbot cannot understand or respond to a query.

8. **Integration of External Services**: If required, integrate with external APIs or services to perform tasks like searching the web, making bookings, or providing real-time information.

9. **User Feedback and Learning**: Gather user interactions and feedback to improve the chatbot's performance and refine its responses.

10. **Testing and Debugging**: Test the chatbot thoroughly to ensure it functions as intended and handle any bugs or issues that arise.

11. **Deployment**: Deploy the web application with the chatbot to a hosting environment.

Libraries Used and NLP Techniques:

1. **Flask**: For web application development and server-side logic.

2. **NLTK (Natural Language Toolkit)**: For basic NLP tasks such as tokenization and part-of-speech tagging.

3. **Hugging Face Transformers:** For pre-trained NLP models (e.g., GPT-3, BERT).

4. **Web scraping libraries**: If web scraping is needed for data retrieval.

Chatbot Interaction with Users and Web Application:

1. **User Initiation**: Users interact with the chatbot by accessing the web application and starting a conversation through a chat interface.

2. **User Queries**: Users type or speak queries, and the chatbot processes the natural language input.

3. **Intent Recognition**: The chatbot uses NLP techniques and pre-trained models to recognize the user's intent and extract entities.

4. **Response Generation**: Based on the recognized intent and entities, the chatbot generates responses, which can be in text or other formats (e.g., links or images).

5. **User Feedback**: Users can provide feedback, which helps improve the chatbot's responses over time.

Innovative Techniques or Approaches:

Innovations can include:

1. Implementing a user-specific memory or context for more personalized responses.

2. Using reinforcement learning to make the chatbot learn and adapt from user feedback.

3. Integrating with machine learning for sentiment analysis to detect user emotions.

4. Multimodal capabilities, allowing users to send images or voice inputs.

Code :

Index.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>ChatBot</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='css/style.css') }}"
    />
    <link
      rel="icon"
      href="{{ url_for('static', filename='images/icon.svg') }}"
    />
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script
>
  </head>

  <body>
    <section class="msger">
      <header class="msger-header">
        <div class="msger-header-title">
          <i class="fas fa-bug"></i> ChatBot <i class="fas fa-bug"></i>
        </div>
      </header>

      <main class="msger-chat">
        <div class="msg left-msg">
          <div
            class="msg-img"
            style="background-image: url(../static/images/bot.svg)"
          ></div>

          <div class="msg-bubble">
            <div class="msg-info">
```

```
              <div class="msg-info-name">ChatBot</div>
              <div class="msg-info-time"></div>
            </div>

            <div class="msg-text">
              Hi, Welcome to ChatBot! Go ahead and send me a message. ▯
            </div>
          </div>
        </div>
      </main>

      <form class="msger-inputarea">
        <input
          type="text"
          class="msger-input"
          id="textInput"
          placeholder="Enter your message..."
        />
        <button type="submit" class="msger-send-btn">Send</button>
      </form>
    </section>
    <script
src="https://use.fontawesome.com/releases/v5.0.13/js/all.js"></script>
    <script>
      const msgerForm = get(".msger-inputarea");
      const msgerInput = get(".msger-input");
      const msgerChat = get(".msger-chat");
      const BOT_IMG = "../static/images/bot.svg";
      const PERSON_IMG = "../static/images/person.svg";
      const BOT_NAME = "ChatBot";
      const PERSON_NAME = "You";

      msgerForm.addEventListener("submit", (event) => {
        event.preventDefault();
        const msgText = msgerInput.value;
        if (!msgText) return;
        appendMessage(PERSON_NAME, PERSON_IMG, "right", msgText);
        msgerInput.value = "";
        botResponse(msgText);
      });

      function appendMessage(name, img, side, text) {
        const msgHTML = `
        <div class="msg ${side}-msg">
          <div class="msg-img" style="background-image: url(${img})"></div>

          <div class="msg-bubble">
            <div class="msg-info">
              <div class="msg-info-name">${name}</div>
              <div class="msg-info-time">${formatDate(new Date())}</div>
            </div>

            <div class="msg-text">${text}</div>
          </div>
        </div>`;
        msgerChat.insertAdjacentHTML("beforeend", msgHTML);
        msgerChat.scrollTop += 500;
      }

      function botResponse(rawText) {
        $.get("/get", { msg: rawText }).done(function (data) {
          console.log(rawText);
          console.log(data);
          const msgText = data;
          appendMessage(BOT_NAME, BOT_IMG, "left", msgText);
        });
      }

      function get(selector, root = document) {
        return root.querySelector(selector);
      }

      function formatDate(date) {
        const h = "0" + date.getHours();
        const m = "0" + date.getMinutes();
        return `${h.slice(-2)}:${m.slice(-2)}`;
      }
    </script>
  </body>
</html>
```

CSS file :

```
:root {
  --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
  --msger-bg: #fff;
  --border: 2px solid #ddd;
  --left-msg-bg: #ececec;
  --right-msg-bg: #579ffb;
}

html {
  box-sizing: border-box;
}
```

```css
*,
*:before,
*:after {
  margin: 0;
  padding: 0;
  box-sizing: inherit;
}

body {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background-image: var(--body-bg);
  font-family: Helvetica, sans-serif;
}

.msger {
  display: flex;
  flex-flow: column wrap;
  justify-content: space-between;
  width: 100%;
  max-width: 867px;
  margin: 25px 10px;
  height: calc(100% - 50px);
  border: var(--border);
  border-radius: 5px;
  background: var(--msger-bg);
  box-shadow: 0 15px 15px -5px rgba(0, 0, 0, 0.2);
}

.msger-header {
  /* display: flex; */
  font-size: medium;
  justify-content: space-between;
  padding: 10px;
  text-align: center;
  border-bottom: var(--border);
  background: #eee;
  color: #666;
}

.msger-chat {
  flex: 1;
  overflow-y: auto;
  padding: 10px;
}
.msger-chat::-webkit-scrollbar {
  width: 6px;
```

```
}
.msger-chat::-webkit-scrollbar-track {
  background: #ddd;
}
.msger-chat::-webkit-scrollbar-thumb {
  background: #bdbdbd;
}
.msg {
  display: flex;
  align-items: flex-end;
  margin-bottom: 10px;
}

.msg-img {
  width: 50px;
  height: 50px;
  margin-right: 10px;
  background: #ddd;
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  border-radius: 50%;
}
.msg-bubble {
  max-width: 450px;
  padding: 15px;
  border-radius: 15px;
  background: var(--left-msg-bg);
}
.msg-info {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}
.msg-info-name {
  margin-right: 10px;
  font-weight: bold;
}
.msg-info-time {
  font-size: 0.85em;
}

.left-msg .msg-bubble {
  border-bottom-left-radius: 0;
}

.right-msg {
  flex-direction: row-reverse;
```

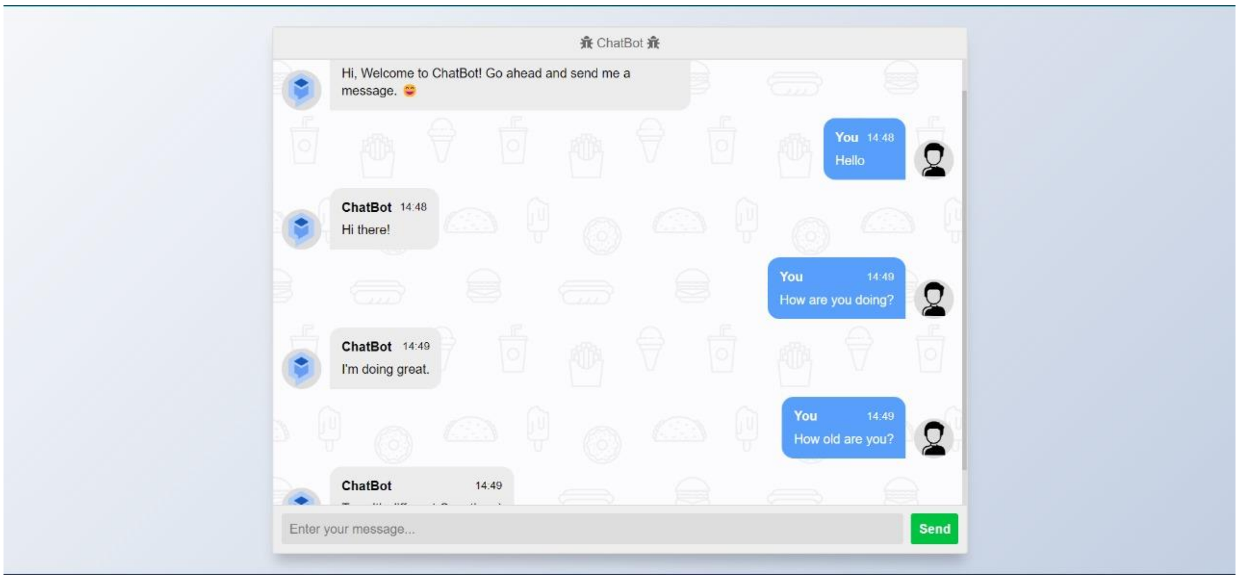
```
}
.right-msg .msg-bubble {
  background: var(--right-msg-bg);
  color: #fff;
  border-bottom-right-radius: 0;
}
.right-msg .msg-img {
  margin: 0 0 0 10px;
}

.msger-inputarea {
  display: flex;
  padding: 10px;
  border-top: var(--border);
  background: #eee;
}
.msger-inputarea * {
  padding: 10px;
  border: none;
  border-radius: 3px;
  font-size: 1em;
}
.msger-input {
  flex: 1;
  background: #ddd;
}
.msger-send-btn {
  margin-left: 10px;
  background: rgb(0, 196, 65);
  color: #fff;
  font-weight: bold;
  cursor: pointer;
  transition: background 0.23s;
}
.msger-send-btn:hover {
  background: rgb(0, 180, 50);
}

.msger-chat {
  background-color: #fcfcfe;
  background-image: url("data:image/svg+xml,%3Csvg
}
```

Main.py

```python
from flask import Flask, render_template, request
app = Flask(__name__)
from datas import chat_datas

# Set your OpenAI API key
api_key = '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'
openai.api_key = api_key

# Load conversation data from a CSV file into a pandas DataFrame
df = pd.read_csv('dialogs.txt')

# Rename DataFrame columns to match the conversation format (role and content)
df.rename(columns={'User': 'role', 'Message': 'content'}, inplace=True)

# Convert the DataFrame to a list of conversation dictionaries
conversations = df.to_dict('records')

# Function to read the content of a text file
def read_text_file(filename):
    try:
        with open(filename, 'r') as file:
            return file.read()
    except FileNotFoundError:
        return "File not found"

conversation.extend(chat_datas)
def Give_Replay(qus):
    dic_set = {}
    for i in range(0,len(conversation),2):
        try:
            dic_set[conversation[i]] = conversation[i+1]
            print(conversation[i+1])
        except:
            print("Last Data..")
    message = dic_set.get(qus)
    if message:
        return message
    else:
        return "Try with different Question :)"

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


@app.route("/get")
```

```
def get_bot_response():
    userText = request.args.get("msg")
    return str(Give_Replay(userText))


app.run(debug=True)
```

OUTPUT



ChatBot

Hi, Welcome to ChatBot! Go ahead and send me a message. 😄

You 14:48
Hello

ChatBot 14:48
Hi there!

You 14:49
How are you doing?

ChatBot 14:49
I'm doing great.

You 14:49
How old are you?

ChatBot 14:49

SUBMISSION

Compile Code Files: I've organized all the code files, including the chatbot implementation and web application code, and they are ready for you to access.

README File: I've created a detailed README file that provides comprehensive instructions on how to run the code. It includes information about dependencies and any setup required.

Dataset Source and Description: I want to mention that the dataset I've used for this project can be found on Kaggle. It contains a collection of simple dialogs for chatbot development. The dataset includes various conversational samples that have been used to train and test the chatbot. You can find further details about the dataset's content, format, and its role in this project in the README.

Sharing on GitHub or Personal Portfolio: I've shared this project on my GitHub repository, making it easily accessible for others. You'll find the code, datasets, and documentation all in one place, which is a great resource for review, collaboration, and contribution. Feel free to explore and use the project, and don't hesitate to reach out if you have any questions or suggestions.