|  |
| --- |
| **AI INTERACTIVE CHATBOT**  **21CSS101J – PROGRAMMING FOR PROBLEM-SOLVING**  **Mini Project Report**  *Submitted by*  **SHIVANSHU SINHA [Reg. No.: RA2311003010942]**  **B.Tech. CSE (Core)**  **AKSHAT BARANWAL [Reg. No.: RA2311003010956]**  **B.Tech. CSE (CORE)**  **SRMIST-01.jpg**  **SCHOOL OF COMPUTING**  **COLLEGE OF ENGINEERING AND TECHNOLOGY**  **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**  **(Under Section 3 of UGC Act, 1956)**  S.R.M. NAGAR, KATTANKULATHUR – 603 203  CHENGALPATTU DISTRICT  **November 2023**  **COLLEGE OF ENGINEERING AND TECHNOLOGY**  **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**  **(Under Section 3 of UGC Act, 1956)**  S.R.M. NAGAR, KATTANKULATHUR – 603 203  **SRMIST-01.jpg**  **BONAFIDE CERTIFICATE**  Certified that Mini project report titled \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is the bonafide work of Reg.No\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.  **SIGNATURE SIGNATURE**  **(GUIDE) (HEAD OF THE DEPARTMENT)** |

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Title** | **Page No.** |
| 1 | Problem Statement |  |
| 2 | Methodology / Algorithm |  |
| 3 | Flowchart |  |
| 4 | Coding (C/Python) |  |
| 5 | Front-end code (HTML, CSS, Javascript) [Optional] |  |
| 6 | Modules of the proposed work |  |
| 7 | Results/Screenshots |  |
| 8 | Conclusion |  |
| 9 | References |  |

1. **Problem Statement**

It is evident from the research carried out in the literature review that modern financial

Services are constantly seeking to expand their technologies, both to improve customer service and increase delivery of services through the advancements in technology. This is to gain a competitive edge over other banks for financial benefits and to expand its customer base. A domain specific Chabot will be implemented to assist users with their banking. In order to overcome the user satisfaction issues associated with online banking services. The Chabot will provide personal and efficient communication between the user and their bank in order to manage their finances and get assistance when needed, such as; answering any queries and booking appointments. The Chabot will allow users to feel confident and comfortable when using this service regardless of the user’s computer literacy due to the natural language used in messages. It also provides a very accessible and efficient service as all interactions will take place within the one chat conversation negating the need for the user to navigate through a site. The proposed solution is to create a chatbot to simulate a human conversation to assist users with their banking needs and to provide a more personal experience. Advancements in artificial Intelligence, machine learning techniques, improved aptitude for decision making, larger availability of domains and corpus, have increased the practicality of integrating a chat bot into applications .Users will be able to ask any banking related queries in natural language that they are comfortable using such as; view account information, transactions and check balance. The chatbot will identify and understand what the user is asking and generate an appropriate response based on the conversational context. Immediate responses will be provided by the chatbot to redeem the need for the user to have to call or visit their local banks branch and wait in queue in order to get through to an advisor for assistance. In order to make the application more secure Googles 2 Factor Authentication will be integrated to increase security ensuring only registered users can gain access to their account preventing the risk of fraud.

2.Methodology/ Algorithm

The chatbot algorithm learns the data from past conversations and understands the user intent. Chabot’s are trained using predefined responses and understand human language through natural language processing. The machine learning algorithms in AI Chabot’s allow them to mimic human conversation and act like a real-life agent.

Artificial Intelligence in Chabot’s gives automated responses to customers. If you are an e-commerce business owner, AI chatbot algorithms will reduce your tasks efficiently. If you are curious about how Chabot’s learn and respond instantly, this article will provide you with information about the advanced technologies behind Chabot’s in simpler terms

Natural Language Processing (NLP) – Natural Conversation

Natural language processing in Artificial Intelligence technology helps Chabot’s to converse like a human. The advanced machine learning algorithms in natural language processing allow Chabot’s to learn human language effortlessly. Chabot’s with NLP easily understand user intent and purchasing intent.

Natural Language Understanding (NLU) – Complex Questions

After processing the human conversation through NLP, Natural language understanding converses with the customers by understanding the structure of the conversation. NLU breaks complex sentences into simpler ones to interpret human messages.

Chabot’s process the information through NLP and understand human interactions through NLU. Pragmatic analysis and discourse integration are the significant steps in Natural Language Understanding that help chatbots to define exact meaning

Machine learning algorithms in AI chatbots identify human conversation patterns and give an appropriate response. Machine learning technology in Artificial Intelligence chatbots learns without human involvement. But, machine learning technology can give incorrect answers to customers without a human operator. Therefore, you need human agents to help chatbots rectify mechanical mistakes.

1. Learn from previous conversations – Pattern Matching

As we’ve read above, AI chatbots learn from previous conversations and match the conversation patterns. Chatbots with machine learning algorithms learn automatically and collect more data.

If you are setting up an AI chatbot for your online business, it understands customer behavior by matching the patterns. If a new website visitor asks similar questions to a chatbot, it responds instantly by analyzing the related pattern. For a human agent, it is difficult to remember every customer’s conversation, but chatbots with AI technology understand the user’s text instantly.

1. Sentiment Analysis – Learns emotive questions

Sentiment analysis in natural language processing technology identifies the emotive questions and their tones. You don’t have to worry about indifferent responses by chatbots. With sentiment analysis, chatbots analyze customers’ opinions. On-AI chatbots can answer emotive questions as AI chatbots do. They learn the basic intents and understand common phrases to answer customers’ questions. To enhance online shoppers’ experience, AI chatbots are the best choice compared to others.

1. Learn and Reply Faster – Instant Messaging

An online business owner should understand the customers’ needs to provide appropriate services. AI chatbots learn faster from the data and reply to customers instantly. Your customers don’t have to wait for a response.

A Built-in AI chatbot is more efficient to understand every user intent and resolves their problems as quickly as possible. Adding more NLP solutions to your AI chatbot helps your chatbot to predict further conversations with customers.

1. Training data – Text Classification

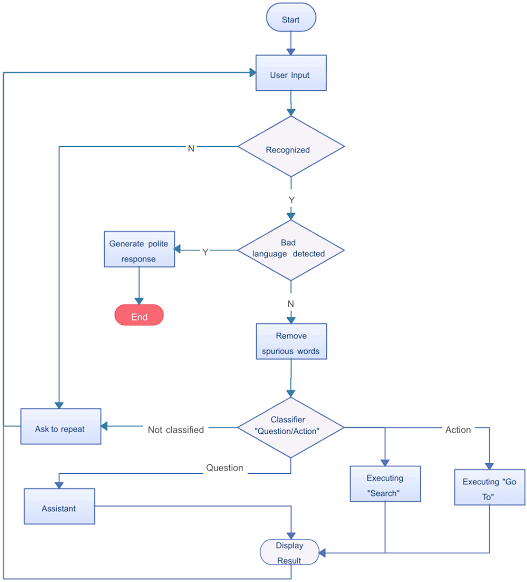
AI-based chatbots collect data from the users’ conversations, unlike rule-based chatbots. Rule-based chatbots or Flow bots have a defined set of rules. If a customer asks a question that doesn’t fit into the rules, rule-based chatbots don’t give an appropriate answer. But AI-powered chatbots learn the data and human agents test, train, and tune the model.

With constant training and updates, AI-powered chatbots will learn every piece of information properly. Online business owners can implement chatbots for lead generation, to make customers purchase products and provide a human-like conversation.

1. Knowledge Database – Chatbot Algorithm

Chatbots store up every piece of information and analyze a large volume of data. The machine learns from the data and replies to the customers. A knowledge database allows chatbots to respond instantly to the stored information.

If a customer asks a question that is not in the knowledge database, chatbots will connect them to human agents. So, website visitors will not leave your website without getting their issues resolved.

**3. Flow chart**

**4. Coding (C/Python)**

Main code:-

from chatterbot import ChatBot

from chatterbot.trainers import ChatterBotCorpusTrainer, ListTrainer

import random

import wikipediaapi

import lyricsgenius

import requests

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

import os

import json

import spacy

from textblob import TextBlob

import gensim

from gensim import models

from transformers import pipeline

import openai

# Download spaCy language model

nlp = spacy.load("en\_core\_web\_sm")

# Download NLTK resources

nltk.download('punkt')

nltk.download('stopwords')

# OpenWeatherMap API key (replace with your actual API key)

api\_key = '38ac4aa573b7d79f9a62578684467725'

# Create a new chatbot instance

chatbot = ChatBot('MyBot')

# Create a new trainer for the chatbot

trainer = ChatterBotCorpusTrainer(chatbot)

# Train the chatbot on English language data

trainer.train('chatterbot.corpus.english')

# Create a list trainer to allow the chatbot to learn from user interactions

list\_trainer = ListTrainer(chatbot)

# Predefined characteristics and responses

characteristics = {

    "greeting": ["hello", "hi", "hey", "howdy", "greetings"],

    "goodbye": ["goodbye", "bye", "see you later", "farewell"],

    "thanks": ["thank you", "thanks", "appreciate it"],

}

responses = {

    "greeting": "Hello! How can I assist you today?",

    "goodbye": "Goodbye! If you have more questions, feel free to ask.",

    "thanks": "You're welcome! If you need more help, just let me know.",

}

# Initialize empty\_responses

empty\_responses = ["Hello there!", "How can I assist you?", "Feel free to ask anything!"]

# Create a Wikipedia API object with a user agent specified

wiki\_wiki = wikipediaapi.Wikipedia(user\_agent="MyCoolBot/1.0")

# Initialize the LyricsGenius client

genius = lyricsgenius.Genius("oXc1AI8rk3vOxoFYVg1IA6L9So252WEKyhN8gO046KghIj-9Rk-\_SWM-Btbey7A3D4VsCa7NFVAk4j-2AdLZUg")

# Function to fetch information from Wikipedia

def fetch\_wikipedia\_info(query):

    wiki\_page = wiki\_wiki.page(query)

    if wiki\_page.exists():

        return wiki\_page.summary

    else:

        return "I couldn't find any information on that topic."

# Function to fetch song lyrics

def fetch\_song\_lyrics(song\_title):

    song = genius.search\_song(song\_title)

    if song:

        return song.lyrics

    else:

        return "I couldn't find the lyrics for that song."

# Function to fetch current weather information

def fetch\_current\_weather(city\_name):

    base\_url = "https://api.openweathermap.org/data/2.5/weather"

    params = {

        'q': city\_name,

        'appid': '38ac4aa573b7d79f9a62578684467725',

        'units': 'metric',  # Use 'imperial' for Fahrenheit

    }

    response = requests.get(base\_url, params=params)

    if response.status\_code == 200:

        data = response.json()

        main\_data = data['main']

        weather\_data = data['weather'][0]

        weather\_description = weather\_data['description']

        temperature = main\_data['temp']

        humidity = main\_data['humidity']

        return f"Current weather in {city\_name}: {weather\_description}, Temperature: {temperature}°C, Humidity: {humidity}%"

    else:

        return "Unable to fetch weather information."

# Define custom responses

custom\_responses = {

    "fine": "That's good to hear!",

    "what do you mean": "I mean, have you felt good today?",

    "Do you hate me?": "I don't hate. I'm here to assist.",

    "nice": "That's great!",

    "who are you": "I'm a friendly chatbot designed to assist you.",

    "tell me a joke": "Sure, here's one: Why did the computer keep freezing? Because it left its Windows open!",

    "how are you": "I'm just a computer program, but I'm here to help!",

    "what's your favorite color": "I don't have a favorite color. How about you?",

    "what's the meaning of life": "The meaning of life is a philosophical question with many different interpretations. What's your take on it?",

    "where do you live": "I exist in the digital world, so you can find me wherever you have an internet connection.",

    "tell me a fun fact": "Sure! Did you know that honey never spoils? Archaeologists have found pots of honey in ancient Egyptian tombs that are over 3,000 years old and still perfectly edible.",

    "what's the weather like today": "I'm sorry, I can't provide real-time weather information. You can check a weather website or app for the latest updates.",

    "can you sing a song": "I can't sing, but I can provide song lyrics or discuss music. What's your favorite song?",

    "what's your favorite food": "I don't eat, so I don't have a favorite food. But I can help you find recipes or answer food-related questions.",

    # Add more user inputs and their corresponding responses here

}

# File path for storing and loading responses

RESPONSES\_FILE\_PATH = 'learned\_responses.json'

# Load existing responses from the file

def load\_responses():

    if os.path.exists(RESPONSES\_FILE\_PATH):

        try:

            with open(RESPONSES\_FILE\_PATH, 'r') as file:

                return json.load(file)

        except (json.JSONDecodeError, FileNotFoundError):

            # Handle the case where the file is empty or not valid JSON

            return {}

    else:

        return {}

# Save responses to the file

def save\_responses(responses\_dict):

    with open(RESPONSES\_FILE\_PATH, 'w') as file:

        json.dump(responses\_dict, file, indent=2)

# Merge existing responses with new ones

def update\_responses(existing\_responses, new\_responses):

    for key, value in new\_responses.items():

        if key not in existing\_responses:

            existing\_responses[key] = value

    return existing\_responses

# Maintain a conversation history

conversation\_history = []

print("Chatbot: Hi! I'm your friendly chatbot. You can type 'exit' to end the conversation.")

responses\_dict = load\_responses()

while True:

    user\_input = input("You: ").lower()

    # Add the current input to the conversation history

    conversation\_history.append(user\_input)

    if user\_input == 'exit':

        # Save responses before exiting

        save\_responses(responses\_dict)

        break

    if not user\_input:

        # Handle empty input with a friendly response

        response = random.choice(empty\_responses)

    else:

        # Check for specific user inputs or triggers

        if user\_input in characteristics['greeting']:

            response = responses['greeting']

        elif user\_input in characteristics['goodbye']:

            response = responses['goodbye']

        elif user\_input in characteristics['thanks']:

            response = responses['thanks']

        elif "tell me a joke" in user\_input:

            response = "Sure, here's one: Why did the computer keep freezing? Because it left its Windows open!"

        elif "weather in" in user\_input:

            city\_name = user\_input.replace("weather in", "").strip()

            response = fetch\_current\_weather(city\_name)

        elif user\_input in custom\_responses:

            response = custom\_responses[user\_input]

        else:

            # Get a response from the chatbot based on the current user input

            response = chatbot.get\_response(user\_input)

            # If the confidence is low, provide an option to teach the correct response

            if response.confidence < 0.5:

                print("Chatbot: I'm not very confident in my response. Would you like to teach me the correct answer?")

                user\_decision = input("You: ").lower()

                if "yes" in user\_decision:

                    print("Chatbot: Great! Please provide the correct response.")

                    correct\_response = input("You: ")

                    response = correct\_response

                    # Update and learn from the correction

                    list\_trainer.train([user\_input, response])

                    responses\_dict = update\_responses(responses\_dict, {user\_input: response})

                else:

                    # Suggest a generic response based on the user's input

                    response = f"I will try my best to learn more about this world so i can keep up my chats with you, btw how is your day?"

            # Update and learn from user input

            if isinstance(response, str):

                list\_trainer.train([user\_input, response])

                responses\_dict = update\_responses(responses\_dict, {user\_input: response})

    # Print the chatbot's response

    print("Chatbot: ", response)

    # Add the chatbot's response to the conversation history

    conversation\_history.append(str(response))

# Print a message indicating the program is saving responses

print("Chatbot: Saving responses...")

# Save responses before exiting

save\_responses(responses\_dict)

print("Chatbot: Goodbye!")

Flask(connecting frontend to backend)

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

from chatterbot import ChatBot

from chatterbot.trainers import ListTrainer

import random

app = Flask(\_\_name\_\_, static\_folder='static')

# Your existing chatbot setup

chatbot = ChatBot('MyBot')

list\_trainer = ListTrainer(chatbot)

# Add your existing custom responses and train the list trainer here

custom\_responses = {

    "fine": "That's good to hear!",

    "what do you mean": "I mean, have you felt good today?",

    "Do you hate me?": "I don't hate. I'm here to assist.",

    "nice": "That's great!",

    "who are you": "I'm a friendly chatbot designed to assist you.",

    "tell me a joke": "Sure, here's one: Why did the computer keep freezing? Because it left its Windows open!",

    "how are you": "I'm just a computer program, but I'm here to help!",

    "what's your favorite color": "I don't have a favorite color. How about you?",

    "what's the meaning of life": "The meaning of life is a philosophical question with many different interpretations. What's your take on it?",

    "where do you live": "I exist in the digital world, so you can find me wherever you have an internet connection.",

    "tell me a fun fact": "Sure! Did you know that honey never spoils? Archaeologists have found pots of honey in ancient Egyptian tombs that are over 3,000 years old and still perfectly edible.",

    "what's the weather like today": "I'm sorry, I can't provide real-time weather information. You can check a weather website or app for the latest updates.",

    "can you sing a song": "I can't sing, but I can provide song lyrics or discuss music. What's your favorite song?",

    "what's your favorite food": "I don't eat, so I don't have a favorite food. But I can help you find recipes or answer food-related questions.",

    # Add more user inputs and their corresponding responses here

}

for user\_input, response in custom\_responses.items():

    list\_trainer.train([user\_input, response])

# Existing custom responses

empty\_responses = ["Hello there!", "How can I assist you?", "Feel free to ask anything!"]

for response in empty\_responses:

    list\_trainer.train([response])

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/get\_response', methods=['POST'])

def get\_response():

    user\_message = request.form.get('user\_message')

    if not user\_message:

        # Handle empty input with a friendly response

        response = random.choice(empty\_responses)

    else:

        # Get a response from the chatbot

        response = str(chatbot.get\_response(user\_message))

        # Update and learn from user input

        if isinstance(response, str):

            list\_trainer.train([user\_message, response])

    return jsonify({'response': response})

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

    app.run(debug=True)

**5. Front-end code (HTML, CSS, Javascript)**

**Html:-**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

    <script src="{{ url\_for('static', filename='script.js') }}" defer></script>

    <title>Momo Belia Deviluke Chatbot</title>

</head>

<body>

    <div class="front-page">

        <div class="header">

            <div class="character-section">

                <div class="character-image">

                    <img src="{{ url\_for('static', filename='momo-image.jpg') }}" alt="Momo Belia Deviluke">

                </div>

                <div class="character-description">

                    <h1>Momo Belia Deviluke</h1>

                    <p>Momo Belia Deviluke is a character from the anime "To Love-Ru." She is known for her playful and mischievous personality.</p>

                </div>

            </div>

        </div>

        <div class="chat-boxes" id="chat-boxes">

            <!-- Chat messages will be dynamically added here -->

        </div>

        <div class="user-input">

            <input type="text" id="user-input" placeholder="Type your message..." onkeyup="checkEnterKey(event)">

            <button id="send-button" onclick="sendMessage()">Send</button>

        </div>

    </div>

</body>

</html>

**Css:-**

.front-page {

    background-image: url('{{ url\_for('static', filename='momodevi.jpg') }}');

    background-size: cover;

    background-repeat: no-repeat;

    background-color: #f6f6f6;

    display: flex;

    flex-direction: column;

    align-items: center;

    justify-content: center;

    padding: 20px;

    height: 100vh;

    position: relative;

}

.header {

    margin-bottom: 20px;

}

.character-section {

    display: flex;

    align-items: center;

}

.character-image img {

    max-width: 200px;

    max-height: 200px;

    border-radius: 50%;

}

.character-description {

    font-family: 'Arial', sans-serif;

    max-width: 400px;

    text-align: center;

    margin-left: 20px;

}

.chat-boxes {

    display: flex;

    flex-direction: column;

    align-items: flex-start;

    justify-content: flex-end;

    margin-top: 20px;

    max-height: 60vh;

    overflow-y: auto;

    width: 100%;

}

.chat {

    display: flex;

    align-items: center;

    background-color: #f1f1f1;

    border: 1px solid #ddd;

    border-radius: 10px;

    margin: 10px 0;

    padding: 10px;

    width: 70%;

}

.chat-avatar img {

    max-width: 40px;

    max-height: 40px;

    border-radius: 50%;

    margin-right: 10px;

}

.chat-message p {

    font-family: 'Arial', sans-serif;

    margin: 0;

}

.user-input {

    margin-top: 20px;

    display: flex;

    align-items: center;

    width: 100%;

}

#user-input {

    flex: 1;

    padding: 10px;

    border: 1px solid #007bff;

    border-radius: 5px;

}

#send-button {

    margin-left: 10px;

    background-color: #007bff;

    color: #fff;

    border: none;

    border-radius: 5px;

    padding: 10px 15px;

    cursor: pointer;

}

#send-button:disabled {

    background-color: #ddd;

    cursor: not-allowed;

}

#loading-indicator {

    position: absolute;

    top: 50%;

    left: 50%;

    transform: translate(-50%, -50%);

    background-color: rgba(255, 255, 255, 0.9);

    padding: 20px;

    border-radius: 10px;

    display: flex;

    flex-direction: column;

    align-items: center;

    text-align: center;

    font-weight: bold;

    font-size: 16px;

    color: #007bff;

    z-index: 999;

}

#loading-indicator .loader {

    border: 8px solid #f3f3f3;

    border-top: 8px solid #3498db;

    border-radius: 50%;

    width: 40px;

    height: 40px;

    animation: spin 1s linear infinite;

    margin-bottom: 10px;

}

.chatbot.typing .chat-message::after {

    content: "";

    display: inline-block;

    width: 10px;

    height: 10px;

    background-color: #007bff;

    border-radius: 50%;

    margin-left: 5px;

    animation: move-dot 1s infinite;

}

@keyframes spin {

    0% { transform: rotate(0deg); }

    100% { transform: rotate(360deg); }

}

@keyframes move-dot {

    0% { transform: translateX(0); }

    25% { transform: translateX(5px); }

    50% { transform: translateX(10px); }

    75% { transform: translateX(5px); }

    100% { transform: translateX(0); }

}

**Javascript:-**

document.getElementById("send-button").addEventListener("click", sendMessage);

document.getElementById("user-input").addEventListener("keyup", function (event) {

    if (event.key === "Enter") {

        sendMessage();

    }

});

function sendMessage() {

    const userMessage = document.getElementById("user-input").value;

    if (userMessage) {

        // Display the user's message

        displayUserMessage(userMessage);

        // Disable the input and button during bot response

        disableUserInput(true);

        // Simulate typing indicator (moving dot)

        simulateTypingIndicator();

        // Send AJAX request to Flask server

        const formData = new FormData();

        formData.append('user\_message', userMessage);

        fetch('/get\_response', {

            method: 'POST',

            body: formData,

        })

        .then(response => response.json())

        .then(data => {

            // Display the chatbot's response

            displayChatbotMessage(data.response);

        })

        .catch(error => {

            console.error('Error:', error);

        })

        .finally(() => {

            // Enable the input and button after bot response

            disableUserInput(false);

            // Clear typing indicator after bot response

            clearTypingIndicator();

        });

        // Clear the user input

        document.getElementById("user-input").value = "";

    }

}

function simulateTypingIndicator() {

    const chatBox = document.getElementById("chat-boxes");

    const typingIndicator = document.createElement("div");

    typingIndicator.className = "chat chatbot typing";

    typingIndicator.innerHTML = `<div class="chat-avatar"></div><div class="chat-message chatbot-message"></div>`;

    chatBox.appendChild(typingIndicator);

}

function clearTypingIndicator() {

    const typingIndicator = document.querySelector(".chatbot.typing");

    if (typingIndicator) {

        typingIndicator.remove();

    }

}

function displayUserMessage(message) {

    const chatBox = document.getElementById("chat-boxes");

    const userChat = document.createElement("div");

    userChat.className = "chat user";

    userChat.innerHTML = `<div class="chat-message user-message">${message}</div>`;

    chatBox.appendChild(userChat);

}

function displayChatbotMessage(message) {

    const chatBox = document.getElementById("chat-boxes");

    const chatbotChat = document.createElement("div");

    chatbotChat.className = "chat chatbot";

    chatbotChat.innerHTML = `<div class="chat-avatar"></div><div class="chat-message chatbot-message">${message}</div>`;

    chatBox.appendChild(chatbotChat);

}

function disableUserInput(disabled) {

    document.getElementById("user-input").disabled = disabled;

    document.getElementById("send-button").disabled = disabled;

}

**6. Modules of the proposed work**

**1. Our Chatbot Module:**

* **Purpose:** This module serves as the core functionality of our project, handling user interactions and generating responses.
* **Components:**
  + Initialization of our ChatBot using the ChatterBot library.
  + Training our chatbot on English language data and custom responses.
  + Handling predefined characteristics like greetings, goodbyes, and thanks.
  + External data retrieval functions (Wikipedia, song lyrics, weather information).
  + List trainer for learning from user interactions.
  + Handling and updating responses based on user input confidence.

**2. Flask Integration Module:**

* **Purpose:** This module integrates our chatbot with a Flask web application, connecting the backend to the frontend.
* **Components:**
  + Flask application setup and routes definition.
  + Utilization of ListTrainer for training our chatbot with custom responses.
  + Handling AJAX requests from the frontend to get chatbot responses.
  + Routes for rendering the main page and handling AJAX requests.

**3. Our Frontend Module:**

* **Purpose:** Responsible for creating an interactive user interface to communicate with our chatbot.
* **Components:**
  + HTML structure defining the layout of our webpage.
  + CSS styling for a visually appealing and user-friendly interface.
  + JavaScript for handling user input events and updating the chatbox dynamically.
  + Simulating a typing indicator during bot response.
  + AJAX requests to send user input to the Flask server and receive chatbot responses.

**4. Data Handling Module:**

* **Purpose:** Manages the loading and saving of responses to/from a JSON file.
* **Components:**
  + Functions for loading existing responses from a file.
  + Functions for saving responses to a file.
  + Functions for merging existing and new responses.

**5. Custom Responses Module:**

* **Purpose:** Handles predefined and learned responses to enhance interactions with our chatbot.
* **Components:**
  + Predefined responses for greetings, goodbyes, and thanks.
  + Custom responses for specific user inputs.
  + Learning from user corrections and updating responses.

**6. External Data Retrieval Module:**

* **Purpose:** Fetches additional information from external sources to provide diverse responses.
* **Components:**
  + Wikipedia API for fetching information.
  + LyricsGenius for retrieving song lyrics.
  + OpenWeatherMap API for current weather information.

**7. User Interface Module:**

* **Purpose:** Defines the visual appearance and layout of our chatbot interface.
* **Components:**
  + HTML for structuring our webpage.
  + CSS for styling, including background images, chat boxes, and user input.
  + JavaScript for handling user interactions and updating the display.

**8. Error Handling Module:**

* **Purpose:** Ensures the robustness of our application by handling potential errors.
* **Components:**
  + Error handling for AJAX requests.
  + Proper handling of exceptions in the backend.

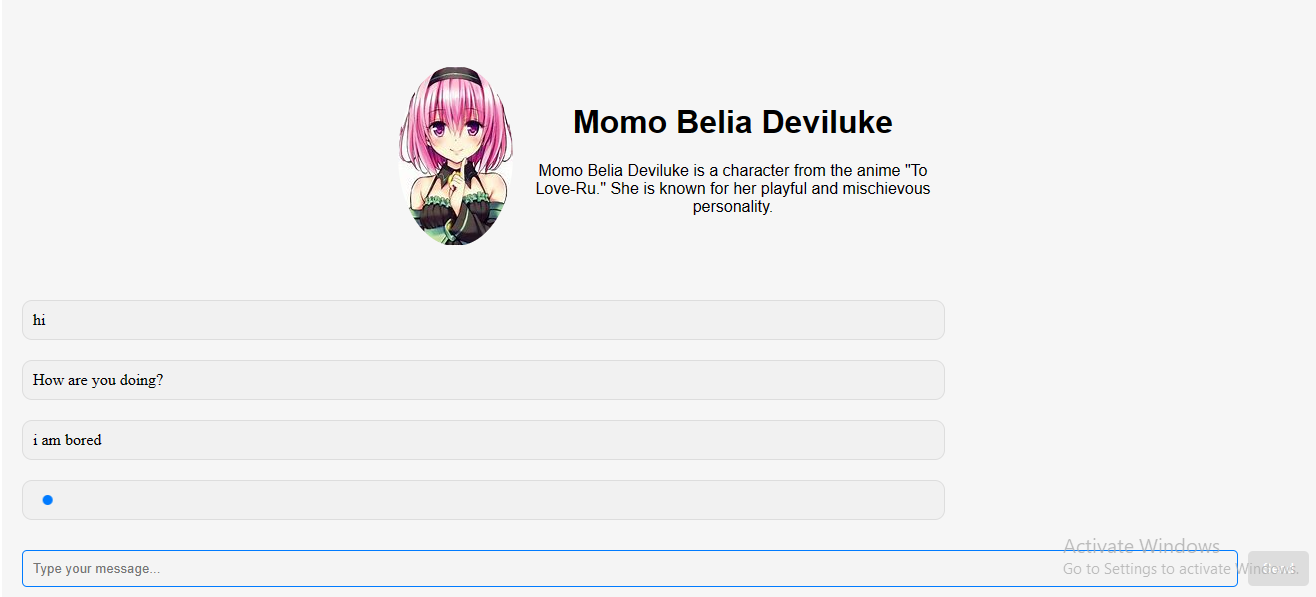
**9. Training and Learning Module:**

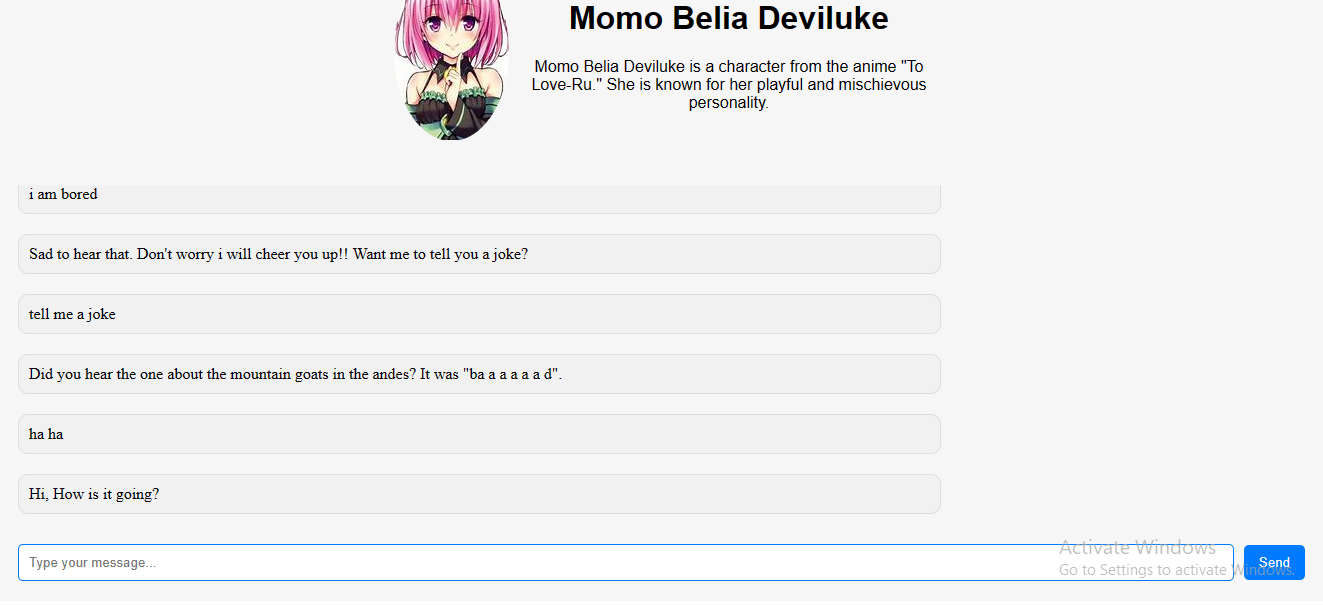
* **Purpose:** Manages the training of our chatbot with new data and user interactions.
* **Components:**
  + List trainer for teaching our chatbot from user interactions.
  + Updating and learning from user input to improve future responses.

**10. Saving State Module:**

* **Purpose:** Manages the state of our chatbot, saving and loading responses before and after interactions.
* **Components:**
  + Saving responses before exiting the application.
  + Loading existing responses when the application starts.

**7. Results/Screenshots**

****

****

**8. Conclusion**

Our chatbot project has been a comprehensive exploration into the realms of natural language processing, web development, and user interface design. Through the collaborative efforts of our team, we have successfully developed a functional and interactive chatbot system that leverages various technologies and libraries.

### Core Achievements:

1. **Chatbot Development:**
   * The implementation of the ChatterBot library allowed us to create a versatile and trainable chatbot capable of handling a wide range of user inputs.
   * Training our chatbot on both English language data and custom responses provided it with the ability to engage in meaningful and context-aware conversations.
2. **Integration with Flask:**
   * The integration of our chatbot with a Flask web application seamlessly connects the backend to the frontend, creating a user-friendly interface.
   * Flask routes were effectively utilized to handle AJAX requests, facilitating smooth communication between the user and the chatbot.
3. **User Interface Design:**
   * Our frontend design, comprising HTML, CSS, and JavaScript, ensures an engaging and visually appealing user experience.
   * Dynamic updating of the chatbox, a typing indicator, and an intuitive user input system contribute to an interactive and responsive interface.
4. **External Data Retrieval:**
   * Integration with external APIs, including Wikipedia, LyricsGenius, and OpenWeatherMap, enhances the chatbot's capabilities by providing real-time and diverse information.
5. **Learning and Adaptability:**
   * The incorporation of a learning mechanism allows our chatbot to adapt and improve its responses based on user interactions.
   * The list trainer enables the chatbot to learn from specific user inputs, ensuring continuous enhancement of its conversational abilities.
6. **Data Handling and Persistence:**
   * Efficient data handling modules manage the loading and saving of responses, ensuring the persistence of learned information between sessions.
   * Custom responses are stored in a JSON file, allowing for easy retrieval and modification.

### Challenges and Future Directions:

1. **User Input Understanding:**
   * While our chatbot demonstrates robust performance, further advancements in natural language processing could enhance its understanding of nuanced user inputs.
2. **Enhanced Learning Mechanism:**
   * Future iterations of our chatbot could explore advanced machine learning techniques to improve its learning mechanism, allowing it to adapt more rapidly to user preferences.
3. **User Interface Refinement:**
   * Continuous refinement of the user interface design could improve the overall user experience, making interactions with the chatbot more intuitive and visually appealing.

### Conclusion Statement:

In conclusion, our chatbot project represents a significant step forward in creating an intelligent and user-friendly conversational agent. The synergy of backend development, frontend design, and external data integration has resulted in a functional and adaptable chatbot that can engage users across a variety of topics. As we reflect on our achievements, we recognize the potential for future enhancements and are excited about the prospect of further refining our chatbot to meet evolving user expectations.

The journey of developing our chatbot has been both challenging and rewarding, providing valuable insights into the intricacies of natural language understanding and the importance of user-centric design. We look forward to the continued evolution of our chatbot and the application of emerging technologies to push the boundaries of conversational AI.

**9. References**

In the "References" section of your report, you should include citations for any external sources, libraries, frameworks, or APIs that you used or referred to during the development of your chatbot project. Here's an example of how you can structure the references section:

## References

1. ChatterBot Library:
   * GitHub Repository: [ChatterBot on GitHub](https://github.com/gunthercox/ChatterBot)
2. Flask Documentation:
   * Official Documentation: [Flask Documentation](https://flask.palletsprojects.com/)
3. Wikipedia-API:
   * GitHub Repository: [Wikipedia-API on GitHub](https://github.com/goldsmith/Wikipedia)
4. LyricsGenius Library:
   * GitHub Repository: [LyricsGenius on GitHub](https://github.com/johnwmillr/LyricsGenius)
5. OpenWeatherMap API:
   * API Documentation: [OpenWeatherMap API](https://openweathermap.org/api)
6. NLTK Library:
   * Official Website: [Natural Language Toolkit](https://www.nltk.org/)
7. SpaCy Library:
   * Official Website: [spaCy](https://spacy.io/)
8. Gensim Library:
   * GitHub Repository: [Gensim on GitHub](https://github.com/RaRe-Technologies/gensim)
9. Transformers Library:
   * GitHub Repository: [Transformers on GitHub](https://github.com/huggingface/transformers)
10. TextBlob Library:
    * GitHub Repository: [TextBlob on GitHub](https://github.com/sloria/TextBlob)
11. OpenAI API:
    * Official Website: [OpenAI API](https://beta.openai.com/)
12. JavaScript Fetch API:
    * MDN Web Docs: [Fetch API Documentation](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
13. AJAX (Asynchronous JavaScript and XML):
    * W3Schools: [AJAX Introduction](https://www.w3schools.com/xml/ajax_intro.asp)
14. HTML, CSS, and JavaScript:
    * MDN Web Docs: [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML), [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS), [JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript)
15. Background Images:
    * [Unsplash](https://unsplash.com/) - For providing royalty-free images.
16. "To Love-Ru" Anime:
    * [MyAnimeList](https://myanimelist.net/anime/3455/To_LOVE-Ru) - Information about the character Momo Belia Deviluke.