**MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION, MUMBAI**

*MICRO PROJECT REPORT ON*   
***CHATBOT USING PYTHON***

*FOR*

***DATA STRUCTURES USING PYTHON (313306)***

***Semester III(K-Scheme)***

Submitted By :

**Anand Jangle(24922)**

**Parth Jadhav(24921)**

**Emmanuel J.(24923)**

Guided By:

**Mrs. Monali Chandwadkar**

Department of

**Artificial Intelligence And Machine Learning (AN)**

**Agnel Polytechnic ,Vashi**

**(Academic Year 2025-2026)**

****

**AGNEL POLYTECHNIC VASHI,**

**NAVI MUMBAI – 400703**

**CERTIFICATE**

**This is to certify that Mr. Anand Jangle Roll no. 24922 of *Third Semester Diploma in Artificial Intelligence and Machine Learning of Institute Agnel Polytechnic, Vashi* (0423) has attained micro project on the topic Chatbot Using Python satisfactory in course Data Structure Using Python (313306) for the academic year 2025-26 as prescribed in curriculum.**

**Place: Vashi Enrollment No.: 24111370132**

**Date: 18 sept 2025 Exam Seat No:**

**Subject Teacher HOD Principal**

SEAL

****

**AGNEL POLYTECHNIC VASHI,**

**NAVI MUMBAI – 400703**

**CERTIFICATE**

**This is to certify that Mr. Parth Jadhav Roll no. 24921 of *Third Semester Diploma in Artificial Intelligence and Machine Learning of Institute Agnel Polytechnic, Vashi* (0423) has attained micro project on the topic Chatbot Using Python satisfactory in course Data Structure Using Python (313306) for the academic year 2025-26 as prescribed in curriculum.**

**Place: Vashi Enrollment No.: 24111370131**

**Date: 18 sept 2025 Exam Seat No:**

**Subject Teacher HOD Principal**

SEAL

****

**AGNEL POLYTECHNIC VASHI,**

**NAVI MUMBAI – 400703**

**CERTIFICATE**

**This is to certify that Mr. Emanneul J. Roll no. 24923 of *Third Semester Diploma in Artificial Intelligence and Machine Learning of Institute Agnel Polytechnic, Vashi* (0423) has attained micro project on the topic Chatbot Using Python satisfactory in course Data Structure Using Python (313306) for the academic year 2025-26 as prescribed in curriculum.**

**Place: Vashi Enrollment No.: 24111370133**

**Date: 18 sept 2025 Exam Seat No:**

**Subject Teacher HOD Principal**

SEAL

**INDEX / TABLE OF CONTENTS**

1.CoverPage……………………………………………………………………………….1

2.Certificate………………………………………………………………………………..2

3. Acknowledgement…………………………………………………………………….. 3

4.Index…………………………………………………………………………………….4

5.Abstract/Summary………………………………………………………………………5

6.Introduction……………………………………………………………………………...6

7.ProblemStatement……………………………………………………………………….7

8.Methodology/WorkingPrinciple…………………………………………………………8

9.Flowchart………………………………………………………………………………..9

10.Components/ToolsRequired…………………………………………………………...10

11.WorkingProcedure……………………………………………………………………..11

12.Advantages&Limitations………………………………………………………………12

13.Applications……………………………………………………………………………13

14.Conclusion…………………………………………………………………………… 14

15.References……………………………………………………………………………..15

**ACKNOWLEDGEMENT**

It is indeed a matter of great pleasure and proud privilege to be able to present this project on The Chatbot Using Python completion of project work is a milestone in student’s life and its execution is inevitable in the hands of guide. We are highly indebted to our project guide **Prof.** **Monali Chandwadkar** for her invaluable guidance and appreciation for giving form and substance

to this report. It is due to his enduring efforts, patience and enthusiasm, which has given a sense of direction to this project and ultimately made it a success.

We would also like to express our deep regards and gratitude to our **HOD Prof. Sonali Sherigar** and our **Principal Mrs. Saly Antony.**

We wish to thank the non – teaching staff and our friends who have helped us all the time in our way or the other. Lastly, we thank almighty, our parents, our family and friends for their constant encouragement.

**ABSTRACT / SUMMARY**

This microproject presents a web-based chatbot application built using Flask as the backend framework and Google’s Gemini Generative AI model as the conversational engine.

The project aims to simplify human–computer interaction by providing an intelligent and responsive chatbot interface that can handle user queries in natural language.

Traditional systems often lack flexibility and require rule-based inputs, while this chatbot leverages advanced AI to generate meaningful responses dynamically.

The application integrates a lightweight HTML-based frontend served through Flask, and secure communication is established with the Gemini API using environment variables to protect sensitive credentials.

The main features include a user-friendly chat interface, AI-driven responses, real-time communication, error handling, and cross-platform compatibility.

The outcome of this project is a functional AI chatbot that demonstrates how modern generative AI models can be integrated into small-scale web applications, making it useful for educational, experimental, or customer service purposes.

**INTRODUCTION**

**Background**

Conversational AI has become an essential part of modern applications, from customer support systems to personal assistants. With advancements in generative AI, chatbots can now deliver human-like, context-aware responses.

**Importance**

This project showcases how cutting-edge AI can be combined with simple web frameworks to create a practical chatbot. It demonstrates secure API handling, real-time communication, and integration of AI into a scalable web service.

**Scope**

The project is limited to a simple chatbot accessible through a web interface. It can be extended to support multi-language conversations, authentication, or integration with databases for advanced use cases.

**Objectives**

* Build a functional AI chatbot using Flask and Gemini AI.
* Securely integrate API keys with .env files.
* Provide a simple yet interactive web-based user interface.
* Demonstrate error handling and robustness in AI-powered applications.

**PROBLEM STATEMENT**

In today's fast-paced digital world, people expect quick and intelligent responses to their queries — whether it's through customer service, personal assistants, or educational tools. Traditional rule-based chatbots have limitations as they rely on pre-defined scripts and decision trees, which often fail when faced with complex or unexpected user input. These chatbots cannot truly “understand” language or context the way humans do, leading to poor user experiences and reduced satisfaction.

The rise of large language models (LLMs) like OpenAI’s GPT and Google’s Gemini has revolutionized how machines can interpret and generate human language. This project addresses the need for a conversational system that utilizes this new generation of AI to provide dynamic, natural-sounding responses. By integrating the Google Gemini API with a Flask web server, we aim to create a responsive and intelligent chatbot that anyone can access through a simple web interface. This chatbot can assist users in real-time, offering a much more engaging and useful experience than traditional bots.

**METHODOLOGY / WORKING PRINCIPLE**

This chatbot was developed using the **Flask web framework** for the backend and **Google’s Generative AI API (Gemini)** to generate natural language responses. The Gemini API allows for real-time content generation based on user prompts. The model used*,* **gemini-1.5-flash***,* is optimized for speed and cost-efficiency, making it ideal for a lightweight web application.

The process involves:

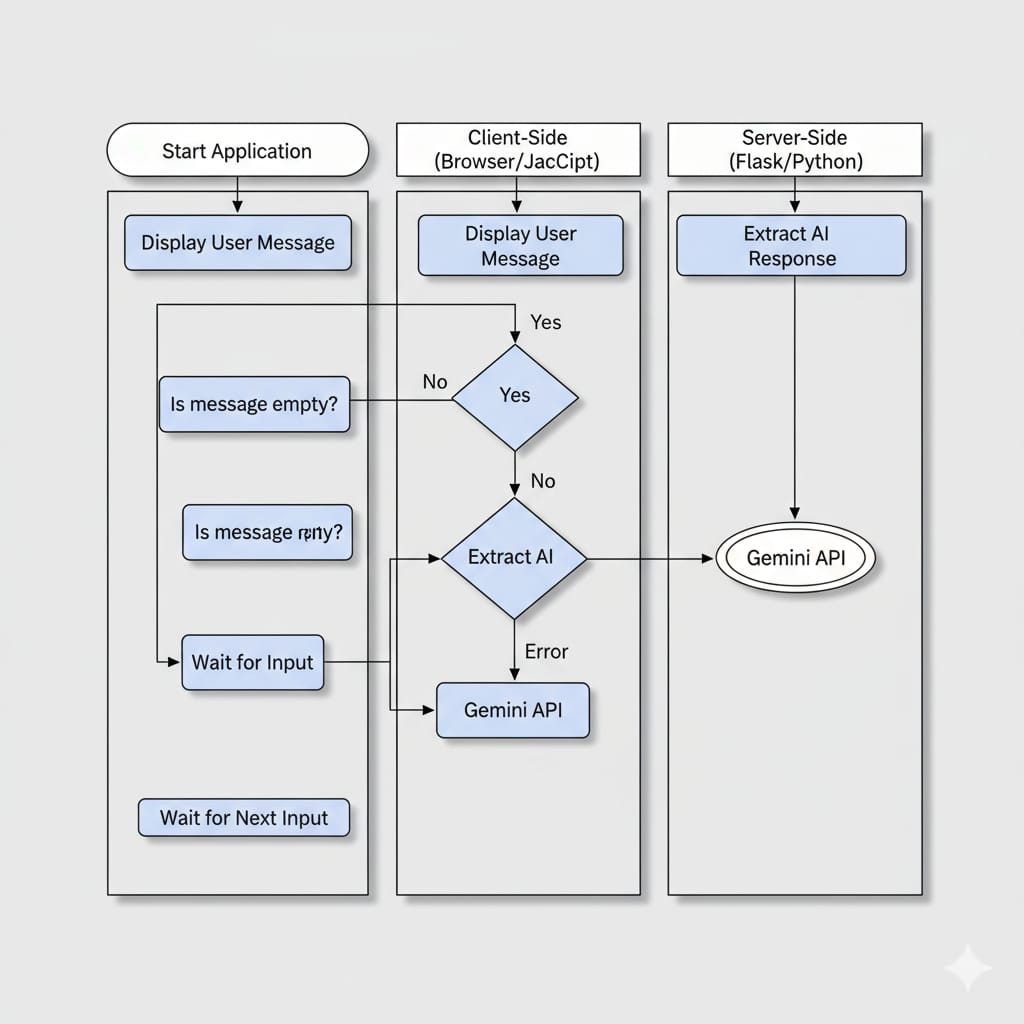
1. Setting up the Flask server to handle HTTP routes.
2. Configuring API access via environment variables.
3. Creating a chat endpoint (/chat) that accepts JSON-formatted POST requests.
4. Sending user input to Gemini’s generative model.
5. Receiving and displaying the AI’s response in the browser through a simple HTML frontend.

This setup ensures separation of concerns, easy maintainability, and the potential to expand features like persistent sessions, chat history, or authentication in future versions.

To build this project, we selected technologies that are lightweight, scalable, and easy to integrate. The **Flask framework** was used to build the backend server, which handles incoming requests, communicates with the Gemini API, and returns responses to the frontend. For the AI component, we used the **Google Generative AI SDK**, which allows access to models like gemini-1.5-flash. This particular model is optimized for quick response times and is ideal for use cases like chatbots, where speed is essential.

The overall working principle is simple but powerful. A user inputs a message through a web-based form. This message is sent to the Flask backend using a POST request. The server then uses the Gemini API to generate a response based on the user’s input. Once the response is received, it is sent back to the frontend, where it is displayed to the user. This process occurs in real-time, creating a smooth and interactive experience. Error handling mechanisms are built in to ensure the system responds appropriately to invalid inputs, server issues, or problems with the API.

**FLOWCHART**



**COMPONENTS/TOOLS REQUIRED**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | |  | | --- | |  |  |  | | --- | | **Component/Tool** | | **Description** |
| 1. | Python (v3.9) | |  | | --- | |  |  |  | | --- | | Programming language for backend development | |
| 2. | Flask | |  | | --- | |  |  |  | | --- | | Micro web framework used to build the backend server | |
| 3. | Google Generative AI SDK | Python SDK to interact with Gemini models |
| 4. | Gemini API Key | |  | | --- | |  |  |  | | --- | | Authenticates API requests to Google’s AI platform | |
| 5. | HTML/CSS | Frontend technologies for creating the user interface |
| 6. | Postman / cURL | Tools to test API endpoints |
| 7. | Terminal / Command Line | For running the Flask server and setting environment variables |
| 8. | Text Editor / IDE | VS Code, PyCharm, or any IDE to write and manage code |

**WORKING**

1. **Initialization:** First, the necessary environment variables (like GEMINI\_API\_KEY) are set. The Flask server is started, which begins listening on http://127.0.0.1:5000.
2. **User Input via Frontend**: A simple HTML page (html\_template.html) allows the user to type a message. When the message is submitted, it is sent as a JSON payload to the Flask server's /chat route via a POST request.
3. **Message Processing:** The Flask server receives the request, validates the JSON structure, and ensures that the user has provided a non-empty message. If validation passes, the message is passed to the generate\_content() method of the Gemini model.
4. **AI Response Generation:** The Gemini model uses its natural language understanding to generate a response. This could be an answer, a creative suggestion, or an informative reply, depending on the input.
5. **Response Display:** The generated response is returned in JSON format and shown to the user on the web page. The user can then continue the conversation by sending additional messages.
6. **Error Handling**: If something goes wrong — such as an invalid message or an issue with the Gemini API — meaningful error messages are returned to the frontend.

**Code**

1. Backend

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

import os

import sys

from flask\_cors import CORS

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

CORS(app)

GEMINI\_API\_KEY = os.getenv('GEMINI\_API\_KEY', 'AIzaSyDpZTqY4\_93qsWKzrI-WlzQig2DiN04Lkk')

genai.configure(api\_key=GEMINI\_API\_KEY)

model\_name = 'gemini-2.0-flash-exp'

print(f"Attempting to initialize model: {model\_name}")

try:

model = genai.GenerativeModel(

model\_name,

generation\_config={

"max\_output\_tokens": 2048,

"temperature": 0.9,

"top\_p": 1,

"top\_k": 40

}

)

print("\nTesting model connection...")

test\_prompt = "Hello, this is a test connection. Please respond with 'Connection successful!'"

test\_response = model.generate\_content(test\_prompt)

if hasattr(test\_response, 'text'):

print(f"Model initialized successfully!")

except Exception as e:

print(f"\nError initializing Gemini model: {str(e)}")

@app.route('/')

def index():

return render\_template('html\_template.html')

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

def chat():

try:

if not GEMINI\_API\_KEY or GEMINI\_API\_KEY == 'your-api-key-here':

return jsonify({

'error': 'API key not properly configured. Please set the GEMINI\_API\_KEY environment variable.',

'status': 'error'

}), 500

if not request.is\_json:

return jsonify({'error': 'Request must be JSON', 'status': 'error'}), 400

data = request.get\_json()

user\_message = data.get('message', '')

if not user\_message:

return jsonify({'error': 'No message provided', 'status': 'error'}), 400

try:

response = model.generate\_content(user\_message)

if not response or not hasattr(response, 'text'):

error\_details = {

'error': 'Invalid response from AI model',

'status': 'error',

'response\_attrs': dir(response) if hasattr(response, '\_\_dict\_\_') else 'No attributes'

}

print(f"Invalid response details: {error\_details}")

return jsonify(error\_details), 500

ai\_response = response.text

return jsonify({

'response': ai\_response,

'status': 'success'

})

except Exception as genai\_error:

print(f"Gemini API Error: {str(genai\_error)}")

return jsonify({

'error': f'Error from Gemini API: {str(genai\_error)}',

'status': 'error'

}), 500

except Exception as e:

print(f"Server Error: {str(e)}")

return jsonify({

'error': f'Server error: {str(e)}',

'status': 'error'

}), 500

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

print("🚀 Starting Flask chatbot server...")

print("📱 Open http://127.0.0.1:5000/ in your browser")

app.run(debug=True, host='127.0.0.1', port=5000)

2. Frontend (using HTML):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>AI Chatbot - Powered by Gemini</title>

<style>

:root {

--bg-primary: #0f0f0f;

--bg-secondary: #1a1a1a;

--bg-chat: #111111;

--bg-user-bubble: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

--bg-bot-bubble: #1e1e1e;

--text-primary: #ffffff;

--text-secondary: #a0a0a0;

--text-user: #ffffff;

--text-bot: #e5e5e5;

--border: #2a2a2a;

--border-light: #333333;

--accent: #667eea;

--accent-hover: #5a6fd8;

--shadow: 0 4px 20px rgba(0, 0, 0, 0.3);

--shadow-light: 0 2px 10px rgba(0, 0, 0, 0.2);

--shadow-bubble: 0 2px 12px rgba(0, 0, 0, 0.15);

--border-radius: 20px;

--border-radius-small: 12px;

}

[data-theme="light"] {

--bg-primary: #f8fafc;

--bg-secondary: #ffffff;

--bg-chat: #f8fafc;

--bg-user-bubble: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

--bg-bot-bubble: #ffffff;

--text-primary: #1a1a1a;

--text-secondary: #6b7280;

--text-user: #ffffff;

--text-bot: #374151;

--border: #e5e7eb;

--border-light: #d1d5db;

--shadow: 0 4px 20px rgba(0, 0, 0, 0.08);

--shadow-light: 0 2px 10px rgba(0, 0, 0, 0.05);

--shadow-bubble: 0 2px 12px rgba(0, 0, 0, 0.08);

}

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Inter', sans-serif;

background: var(--bg-primary);

color: var(--text-primary);

height: 100vh;

display: flex;

flex-direction: column;

line-height: 1.6;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

overflow: hidden;

}

.chat-container {

display: flex;

flex-direction: column;

height: 100vh;

max-width: 100%;

margin: 0 auto;

background: var(--bg-chat);

position: relative;

width: 100%;

}

.chat-header {

background: var(--bg-secondary);

color: var(--text-primary);

padding: 16px 24px;

border-bottom: 1px solid(var(--border));

position: sticky;

top: 0;

z-index: 100;

display: flex;

justify-content: center;

align-items: center;

min-height: 70px;

box-shadow: var(--shadow-light);

backdrop-filter: blur(20px);

-webkit-backdrop-filter: blur(20px);

}

.chat-header h1 {

font-size: 20px;

font-weight: 700;

margin: 0;

letter-spacing: -0.02em;

background: linear-gradient(135deg, var(--accent), #764ba2);

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

background-clip: text;

}

.chat-messages {

flex: 1;

overflow-y: auto;

overflow-x: hidden;

padding: 20px 16px;

display: flex;

flex-direction: column;

gap: 16px;

background: var(--bg-chat);

scroll-behavior: smooth;

max-width: 1200px;

margin: 0 auto;

width: 100%;

box-sizing: border-box;

}

.message {

display: flex;

width: 100%;

padding: 0 8px;

box-sizing: border-box;

animation: messageSlide 0.4s cubic-bezier(0.25, 0.46, 0.45, 0.94);

}

@keyframes messageSlide {

from {

opacity: 0;

transform: translateY(20px) scale(0.95);

}

to {

opacity: 1;

transform: translateY(0) scale(1);

}

}

.user-message {

justify-content: flex-end;

}

.bot-message {

justify-content: flex-start;

}

.message-bubble {

max-width: calc(100% - 60px);

padding: 16px 20px;

border-radius: var(--border-radius);

position: relative;

box-shadow: var(--shadow-bubble);

word-wrap: break-word;

backdrop-filter: blur(10px);

-webkit-backdrop-filter: blur(10px);

box-sizing: border-box;

}

.user-message .message-bubble {

background: var(--bg-user-bubble);

color: var(--text-user);

border-bottom-right-radius: 8px;

margin-left: auto;

margin-right: 0;

}

.bot-message .message-bubble {

background: var(--bg-bot-bubble);

color: var(--text-bot);

border: 1px solid(var(--border-light));

border-bottom-left-radius: 8px;

margin-left: 0;

margin-right: auto;

}

.message-text {

font-size: 15px;

line-height: 1.6;

word-wrap: break-word;

min-width: 0;

}

.message-text p {

margin: 0 0 8px 0;

}

.message-text p:last-child {

margin-bottom: 0;

}

.message-text strong {

font-weight: 600;

}

.message-text em {

font-style: italic;

}

.message-text code {

background-color: rgba(255, 255, 255, 0.1);

padding: 2px 6px;

border-radius: 6px;

font-family: 'SF Mono', Monaco, 'Cascadia Code', 'Roboto Mono', Consolas, 'Courier New', monospace;

font-size: 13px;

font-weight: 500;

}

.user-message .message-text code {

background-color: rgba(255, 255, 255, 0.2);

}

.message-avatar {

width: 32px;

height: 32px;

border-radius: 50%;

display: flex;

align-items: center;

justify-content: center;

flex-shrink: 0;

font-weight: 600;

font-size: 14px;

color: white;

position: absolute;

box-shadow: var(--shadow-bubble);

}

.user-message .message-avatar {

background: linear-gradient(135deg, #ff6b6b, #ee5a24);

right: -40px;

top: 8px;

}

.bot-message .message-avatar {

background: linear-gradient(135deg, var(--accent), #764ba2);

left: -40px;

top: 8px;

}

.chat-input-container {

padding: 20px 24px 24px;

background: var(--bg-primary);

border-top: 1px solid(var(--border));

backdrop-filter: blur(20px);

-webkit-backdrop-filter: blur(20px);

}

.chat-input-wrapper {

position: relative;

max-width: 800px;

margin: 0 auto;

width: 100%;

background: var(--bg-secondary);

border-radius: 24px;

border: 2px solid(var(--border));

box-shadow: var(--shadow);

transition: all 0.3s cubic-bezier(0.25, 0.46, 0.45, 0.94);

overflow: hidden;

}

.chat-input-wrapper:focus-within {

border-color: var(--accent);

box-shadow: 0 0 0 4px rgba(102, 126, 234, 0.1), var(--shadow);

transform: translateY(-2px);

}

.chat-input {

width: 100%;

padding: 18px 24px;

padding-right: 70px;

border: none;

border-radius: 24px;

font-size: 16px;

line-height: 1.6;

background: transparent;

color: var(--text-primary);

outline: none;

resize: none;

min-height: 24px;

max-height: 160px;

font-family: inherit;

font-weight: 400;

transition: all 0.2s ease;

}

.chat-input::placeholder {

color: var(--text-secondary);

opacity: 1;

font-weight: 400;

}

.send-button {

position: absolute;

right: 8px;

top: 50%;

transform: translateY(-50%);

background: linear-gradient(135deg, var(--accent), #764ba2);

color: white;

border: none;

border-radius: 16px;

width: 48px;

height: 48px;

display: flex;

align-items: center;

justify-content: center;

cursor: pointer;

transition: all 0.3s cubic-bezier(0.25, 0.46, 0.45, 0.94);

opacity: 1;

box-shadow: 0 4px 12px rgba(102, 126, 234, 0.3);

}

.send-button:hover {

transform: translateY(-50%) scale(1.05);

box-shadow: 0 6px 20px rgba(102, 126, 234, 0.4);

}

.send-button:active {

transform: translateY(-50%) scale(0.95);

}

.send-button:disabled {

opacity: 0.5;

cursor: not-allowed;

transform: translateY(-50%);

}

.send-icon {

width: 20px;

height: 20px;

}

.typing-indicator {

display: flex;

width: 100%;

justify-content: flex-start;

animation: messageSlide 0.4s cubic-bezier(0.25, 0.46, 0.45, 0.94);

}

.typing-bubble {

background: var(--bg-bot-bubble);

border: 1px solid(var(--border-light));

border-radius: var(--border-radius);

border-bottom-left-radius: 8px;

padding: 16px 20px;

margin-right: 40px;

box-shadow: var(--shadow-bubble);

position: relative;

}

.typing-dots {

display: flex;

gap: 6px;

align-items: center;

padding: 4px 0;

}

.typing-dots span {

width: 8px;

height: 8px;

background: var(--text-secondary);

border-radius: 50%;

animation: typingBounce 1.4s ease-in-out infinite;

}

.typing-dots span:nth-child(1) {

animation-delay: 0s;

}

.typing-dots span:nth-child(2) {

animation-delay: 0.2s;

}

.typing-dots span:nth-child(3) {

animation-delay: 0.4s;

}

@keyframes typingBounce {

0%, 60%, 100% {

opacity: 0.3;

transform: translateY(0);

}

30% {

opacity: 1;

transform: translateY(-8px);

}

}

.error-message .message-bubble {

background: linear-gradient(135deg, #ff6b6b, #ee5a24);

color: white;

}

/\* Scrollbar styling \*/

.chat-messages::-webkit-scrollbar {

width: 6px;

}

.chat-messages::-webkit-scrollbar-track {

background: transparent;

}

.chat-messages::-webkit-scrollbar-thumb {

background: var(--border);

border-radius: 3px;

}

.chat-messages::-webkit-scrollbar-thumb:hover {

background: var(--text-secondary);

}

/\* Theme toggle \*/

.theme-toggle {

position: fixed;

top: 20px;

right: 20px;

background: var(--bg-secondary);

border: 2px solid(var(--border));

border-radius: 12px;

width: 48px;

height: 48px;

display: flex;

align-items: center;

justify-content: center;

cursor: pointer;

box-shadow: var(--shadow);

z-index: 1000;

color: var(--text-primary);

transition: all 0.3s cubic-bezier(0.25, 0.46, 0.45, 0.94);

}

.theme-toggle:hover {

transform: scale(1.05);

box-shadow: var(--shadow-light);

}

/\* Welcome message \*/

.welcome-message {

text-align: center;

color: var(--text-secondary);

padding: 60px 24px;

font-size: 18px;

font-weight: 500;

background: linear-gradient(135deg, var(--accent), #764ba2);

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

background-clip: text;

}

/\* Responsive design \*/

@media (max-width: 768px) {

.chat-messages {

padding: 16px 20px;

}

.chat-header {

padding: 12px 20px;

min-height: 60px;

}

.chat-header h1 {

font-size: 18px;

}

.chat-messages {

padding: 16px;

gap: 12px;

}

.message-bubble {

max-width: 85%;

padding: 14px 16px;

}

.user-message .message-bubble {

margin-left: 20px;

}

.bot-message .message-bubble {

margin-right: 20px;

}

.message-avatar {

width: 28px;

height: 28px;

}

.user-message .message-avatar {

right: 0;

margin-right: 8px;

}

.bot-message .message-avatar {

left: 0;

margin-left: 8px;

}

.chat-input-container {

padding: 16px 20px 20px;

}

.chat-input {

padding: 16px 20px;

padding-right: 65px;

font-size: 16px;

}

.send-button {

width: 44px;

height: 44px;

}

.theme-toggle {

top: 16px;

right: 16px;

width: 44px;

height: 44px;

}

.welcome-message {

padding: 40px 20px;

font-size: 16px;

}

}

@media (max-width: 480px) {

.chat-messages {

padding: 12px 8px;

}

.message {

padding: 0 4px;

}

.message-bubble {

max-width: 90%;

padding: 12px 14px;

}

.chat-input {

padding: 14px 18px;

padding-right: 60px;

}

.send-button {

width: 40px;

height: 40px;

}

}

</style>

</head>

<body>

<div class="theme-toggle" onclick="toggleTheme()">

<svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24 24" fill="none" stroke="currentColor"

stroke-width="2" stroke-linecap="round" stroke-linejoin="round">

<circle cx="12" cy="12" r="5" />

<line x1="12" y1="1" x2="12" y2="3" />

<line x1="12" y1="21" x2="12" y2="23" />

<line x1="4.22" y1="4.22" x2="5.64" y2="5.64" />

<line x1="18.36" y1="18.36" x2="19.78" y2="19.78" />

<line x1="1" y1="12" x2="3" y2="12" />

<line x1="21" y1="12" x2="23" y2="12" />

<line x1="4.22" y1="19.78" x2="5.64" y2="18.36" />

<line x1="18.36" y1="5.64" x2="19.78" y2="4.22" />

</svg>

</div>

<div class="chat-container">

<div class="chat-header">

<h1>CHATBOT</h1>

</div>

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

<div class="welcome-message">How can I help you today?</div>

</div>

<div class="chat-input-container">

<form class="chat-input-form" id="chat-form">

<div class="chat-input-wrapper">

<textarea id="user-input" class="chat-input" placeholder="How are you..." rows="1" required></textarea>

<button type="submit" class="send-button" id="send-button">

<svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24 24" fill="none"

stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"

class="send-icon">

<line x1="22" y1="2" x2="11" y2="13"></line>

<polygon points="22 2 15 22 11 13 2 9 22 2"></polygon>

</svg>

</button>

</div>

</form>

</div>

</div>

<script>

// Theme toggle functionality

function toggleTheme() {

const body = document.body;

const isLight = body.hasAttribute('data-theme');

if (isLight) {

body.removeAttribute('data-theme');

localStorage.setItem('theme', 'dark');

} else {

body.setAttribute('data-theme', 'light');

localStorage.setItem('theme', 'light');

}

}

// Load saved theme

function loadTheme() {

const savedTheme = localStorage.getItem('theme');

if (savedTheme === 'light') {

document.body.setAttribute('data-theme', 'light');

}

}

class ChatBot {

constructor() {

this.chatMessages = document.getElementById('chat-messages');

this.messageInput = document.getElementById('user-input');

this.sendButton = document.getElementById('send-button');

this.chatForm = document.getElementById('chat-form');

this.initializeEventListeners();

this.autoResizeTextarea();

this.messageInput.focus();

this.removeWelcomeMessage();

}

removeWelcomeMessage() {

// Remove welcome message when first message is sent

this.shouldRemoveWelcome = true;

}

initializeEventListeners() {

this.chatForm.addEventListener('submit', (e) => {

e.preventDefault();

this.sendMessage();

});

this.messageInput.addEventListener('keypress', (e) => {

if (e.key === 'Enter' && !e.shiftKey) {

e.preventDefault();

this.sendMessage();

}

});

}

async sendMessage() {

const message = this.messageInput.value.trim();

if (!message) return;

// Remove welcome message on first interaction

if (this.shouldRemoveWelcome) {

const welcomeMsg = document.querySelector('.welcome-message');

if (welcomeMsg) {

welcomeMsg.remove();

}

this.shouldRemoveWelcome = false;

}

// Add user message to chat

this.addMessage(message, 'user');

this.messageInput.value = '';

this.setLoading(true);

try {

// Send message to backend

const response = await fetch('http://localhost:5000/chat', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ message: message })

});

const data = await response.json();

if (response.ok && data.status === 'success') {

this.addMessage(data.response, 'bot');

} else {

// Show more detailed error message from the server

const errorMsg = data.error || 'Sorry, something went wrong.';

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

this.addMessage(`Error: ${errorMsg}`, 'bot', true);

}

} catch (error) {

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

this.addMessage('Sorry, I couldn\'t connect to the server. Please try again.', 'bot', true);

} finally {

this.setLoading(false);

}

}

addMessage(text, sender, isError = false) {

const messageDiv = document.createElement('div');

messageDiv.className = `message ${sender}-message ${isError ? 'error-message' : ''}`;

const messageBubble = document.createElement('div');

messageBubble.className = 'message-bubble';

const messageText = document.createElement('div');

messageText.className = 'message-text';

messageText.innerHTML = this.formatMessage(text);

const avatar = document.createElement('div');

avatar.className = 'message-avatar';

avatar.innerHTML = sender === 'user'

? '<svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"><path d="M20 21v-2a4 4 0 0 0-4-4H8a4 4 0 0 0-4 4v2"></path><circle cx="12" cy="7" r="4"></circle></svg>'

: '<svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0 24 24" fill="currentColor"><path d="M12 2C6.48 2 2 6.48 2 12s4.48 10 10 10 10-4.48 10-10S17.52 2 12 2zm-5 9l1.5 1.5L13 8l4 4-6 6-4-4z"/></svg>';

messageBubble.appendChild(messageText);

messageBubble.appendChild(avatar);

messageDiv.appendChild(messageBubble);

this.chatMessages.appendChild(messageDiv);

this.scrollToBottom();

return messageDiv;

}

formatMessage(text) {

// Convert markdown-like formatting to HTML

return text

.replace(/\\*\\*(.\*?)\\*\\*/g, '<strong>$1</strong>') // Bold

.replace(/\\*(.\*?)\\*/g, '<em>$1</em>') // Italic

.replace(/`(.\*?)`/g, '<code>$1</code>') // Inline code

.replace(/\n/g, '<br>'); // Line breaks

}

autoResizeTextarea() {

const textarea = this.messageInput;

textarea.addEventListener('input', function () {

this.style.height = 'auto';

this.style.height = Math.min(this.scrollHeight, 160) + 'px';

});

}

setLoading(isLoading) {

if (isLoading) {

this.sendButton.disabled = true;

this.messageInput.disabled = true;

this.addTypingIndicator();

} else {

this.sendButton.disabled = false;

this.messageInput.disabled = false;

this.removeTypingIndicator();

this.messageInput.focus();

}

}

addTypingIndicator() {

const typingDiv = document.createElement('div');

typingDiv.className = 'typing-indicator';

typingDiv.id = 'typingIndicator';

const typingBubble = document.createElement('div');

typingBubble.className = 'typing-bubble';

const avatar = document.createElement('div');

avatar.className = 'message-avatar';

avatar.innerHTML = '<svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" viewBox="0 0 24 24" fill="currentColor"><pathid="M12 2C6.48 2 2 6.48 2 12s4.48 10 10 10 10-4.48 10-10S17.52 2 12 2zm-5 9l1.5 1.5L13 8l4 4-6 6-4-4z"/></svg>';

const dotsContainer = document.createElement('div');

dotsContainer.innerHTML = `<div class="typing-dots">

<span></span>

<span></span>

<span></span>

</div>`;

typingBubble.appendChild(dotsContainer);

typingBubble.appendChild(avatar);

typingDiv.appendChild(typingBubble);

this.chatMessages.appendChild(typingDiv);

this.scrollToBottom();

}

removeTypingIndicator() {

const typingIndicator = document.getElementById('typingIndicator');

if (typingIndicator) {

typingIndicator.remove();

}

}

scrollToBottom() {

setTimeout(() => {

this.chatMessages.scrollTop = this.chatMessages.scrollHeight;

}, 100);

}

}

// Initialize chatbot when page loads

document.addEventListener('DOMContentLoaded', () => {

loadTheme();

new ChatBot();

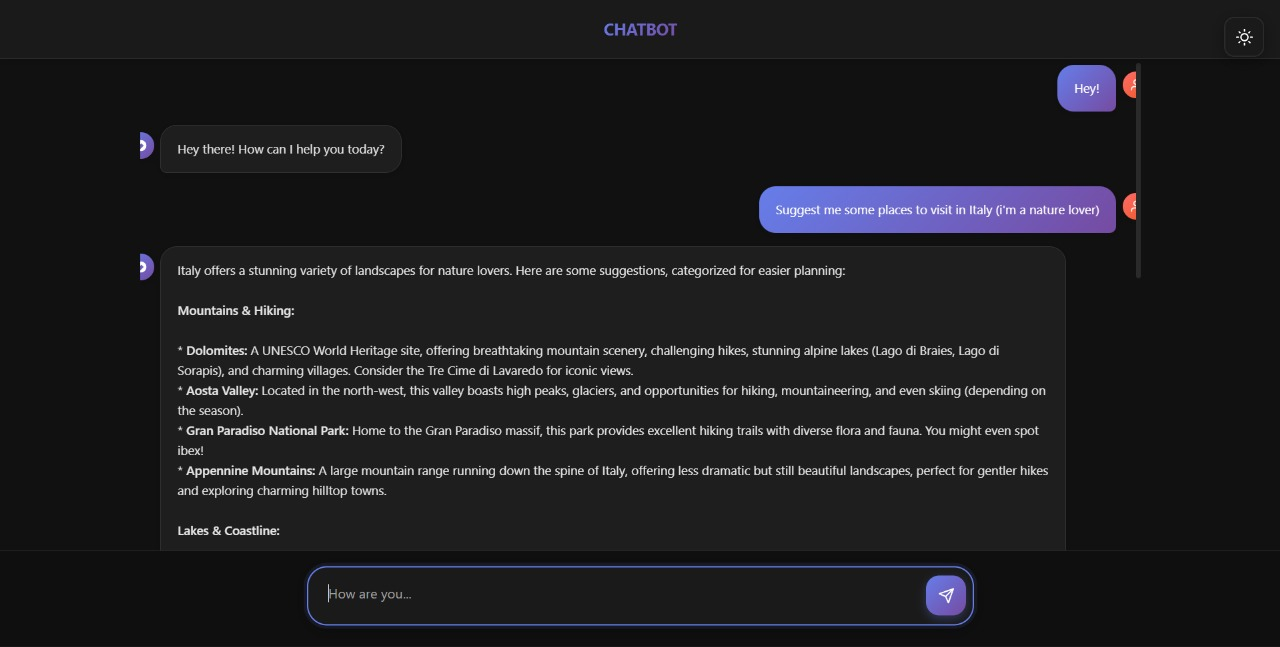
});

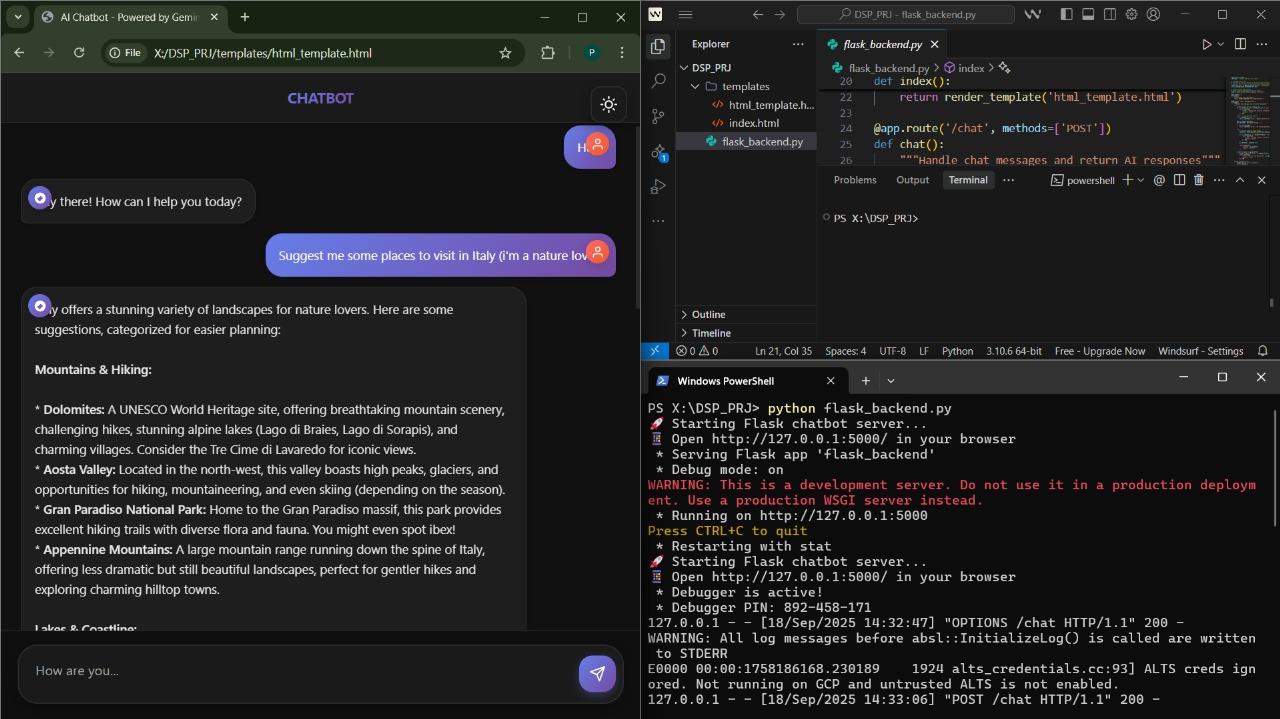
</script>

</body>

</html>

**OUTPUT**

****

****

**ADVANTAGES AND LIMITATIONS**

**Advantages:**

* **Intelligent Conversation**: The chatbot can handle open-ended questions, provide detailed answers, and even engage in creative or educational dialogues.
* **Scalable Architecture**: Built with modular components, making it easy to add new features like chat history, authentication, or voice support.
* **Easy Integration**: Can be integrated into websites, mobile apps, or desktop platforms with minimal configuration.

**Limitations:**

* **No Memory or Context**: Since we are not using start\_chat() or session tracking, the chatbot treats each message independently.
* **API Dependency**: The system is dependent on the availability and limits of the Gemini API, including request quotas and internet access.
* **Security Risks**: If not properly managed, hardcoding the API key can expose the system to misuse. Proper security protocols must be followed in deployment.

**APPLICATIONS**

This chatbot project has multiple practical applications across different sectors:

* **Education**: Can act as a tutor or study assistant, answering questions about various subjects in real time.
* **Customer Support**: Automate responses to frequent queries in e-commerce, banking, and SaaS platforms.
* **Personal Assistant**: Could be extended to manage tasks like scheduling, reminders, or answering factual questions.
* **Healthcare**: Used as an initial point of contact to answer common medical questions (with proper disclaimers).
* **Enterprise HR**: Can automate internal queries related to HR policies, leave management, and onboarding.

These applications show the versatility of AI chatbots and their ability to improve user experience while reducing operational costs.

**CONCLUSION**

This project successfully demonstrates the implementation of a modern AI chatbot using Google’s Gemini generative model integrated with a Flask web server. It highlights how recent advancements in AI can be made accessible through simple web technologies. The chatbot is capable of understanding user input and responding in a meaningful and contextually relevant way, showcasing the power of generative AI in everyday applications.

Through this project, valuable insights were gained into API integration, server-client communication, and error handling. The hands-on experience also emphasized the importance of secure coding practices and scalable design. In the future, this system could be extended to include features like persistent chat history, support for multiple languages, voice input/output, and integration with other APIs for extended functionality. With minor modifications, it can serve in professional, academic, or personal use cases.

**REFERENCES/BIBLIOGRAPHY**

1. **Google Generative AI Documentation** – https://ai.google.dev/
2. **Flask Official Documentation** – https://flask.palletsprojects.com/en/latest/
3. **Python Official Docs** – https://docs.python.org/3/
4. **Google MakerSuite (API Key)** – <https://makersuite.google.com/app/apikey>
5. **Stack Overflow** – <https://stackoverflow.com/>
6. **Real Python – Flask Tutorials** – https://realpython.com/tutorials/flask/
7. **Postman API Testing Tool** – https://www.postman.com/
8. **OpenAI Blog on LLMs and Chatbots** – <https://openai.com/blog>