GUIDE YOUR TRIP

A
Industrial Oriented Mini Project

Submitted in complete fulfillment of the requirements for the award of completion of 3rd year 2nd semester

Bachelor of Technology
in
Computer Science and Engineering(AI and ML)

Submitted by

S. Ashritha (22SS1A6649)

Under the guidance of

Mrs. B. Sangeetha

Assistant Professor

Department of Computer Science and Engineering

JNTUH University College of Engineering Sultanpur

Sultanpur(V), Pulkal(M), Sangareddy district, Telangana-502273

JUNE 2025

JNTUH UNIVERSITY COLLEGE OF ENGINEERING SULTANPUR

Sultanpur(V), Pulkal(M), Sangareddy-502273, Telangana

Department of Computer Science and Engineering

Certificate

This is to certify that the Industrial Oriented Mini Project report work entitled "**Guide Your Trip**" is a bonafide work carried out by **S. Ashritha** bearing Roll no **22SS1A6649**in fulfillment of the requirements for the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING - (AI and ML)** discipline to Jawaharlal Nehru Technological University Hyderabad College of Engineering Sultanpur during the academic year 2024- 2025.

Guide	**Principal, Head**
Mrs. B. Sangeetha	**Dr. G. Narsimha**
Assistant Professor of CSE	**Professor**

Declaration

This is to certify that the Industrial Oriented Mini Project report work entitled "**Guide Your Trip**" is a bonafide work carried out by **S. Ashritha** bearing Roll no **22SS1A6649** in fulfillment of the requirements for the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING - (AI and ML)** discipline to Jawaharlal Nehru Technological University Hyderabad College of Engineering Sultanpur during the academic year 2024- 2025.

The results embodied in this report have not been submitted to any other University or Institution for the award of any degree or diploma.

S. Ashritha (22SS1A6649)

Acknowledgment

Thank you to all of those who have helped us in various ways during our the Industrial Oriented Mini Project work It is our pleasure to have recieved support and extensive help throughout this project .

We express our sincere gratitude to the Guide **Mrs. B. Sangeetha , Assistant Professor of CSE , JNTUHUCES** for her support during the semester and guidance. Heartfelt gratitude to **Dr. G Narsimha, Principal, JNTUHCES and Dr.Y Raghavender Rao, Vice - Principal, JNTUHCES** in the completion of Real-time Research Project.

S. Ashritha (22SS1A6649)

Contents

List of Figures

Abstract

Guide Your Trip is an AI-powered travel chatbot web application that helps users explore cities effortlessly. By entering a city name, users instantly receive curated information such as a brief description, top tourist attractions, popular local foods, nearby shopping malls, and recommended restaurants.Built with Python and Flask, the app offers a lightweight yet efficient backend. It integrates external APIs like Wikipedia for city descriptions and OpenAI's language models (via OpenRouter API) to generate smart travel recommendations. Using natural language processing (NLP), the chatbot interprets user queries and responds in an engaging, conversational manner. What sets this project apart is its ability to deliver real-time, structured, and relevant data, eliminating the need for manual research. It also provides Google Maps links for each destination, improving user convenience. This tool showcases the power of AI, NLP, and web development in solving practical problems like travel planning.Future features may include personalized itineraries, multilingual support, real-time event updates, and booking service integration, making it a strong foundation for next-gen travel solutions.

Chapter 1

INTRODUCTION

1.1 Project Overview

This project is a web-based AI Travel Chatbot designed to assist users in learning about cities around the world by providing concise travel-related information in a conversational format. The chatbot is built using Python Flask, integrates Wikipedia for general city descriptions, and uses OpenAI's language model via OpenRouter API to generate personalized travel suggestions. It offers quick access to information such as: A brief description of the city Top 3 tourist attractions Famous local foods Popular shopping malls Recommended restaurants Direct Google Maps links for each place

This intuitive interface helps travelers quickly understand what to explore in a destination, making it ideal for trip planning, learning, or even as a travel inspiration tool.

1.2 Purpose

The purpose of this project is to develop an AI-powered travel chatbot web application that serves as a smart travel assistant by offering accurate and concise information about various cities. The aim is to make travel research and planning easier, faster, and more interactive using artificial intelligence and automation

- To centralize travel information:

  Travelers often browse multiple websites or apps to learn about a destination. This chatbot consolidates vital details like city overviews, places to visit, foods to try, and shopping or dining options in one platform

- To reduce information overload:

  Instead of long articles or blogs, the chatbot offers AI-generated, bullet-point summaries, making information easy to consume and act upon.

- To leverage AI for personalization and accuracy:

  By using advanced language models (e.g., LLaMA via OpenRouter), the chatbot delivers intelligent suggestions that are contextually relevant and reliable.

- To enhance the travel experience through interactivity:

  Unlike static travel guides, users can interact with the chatbot in a conversational format, making the planning process more engaging and intuitive.

- To promote user-friendly navigation:

  With integrated Google Maps links, users can explore places directly, making it easier to visualize and plan their visit.

- To demonstrate practical use of AI and Flask web development:

  The project also serves an educational purpose by showing how AI, APIs, and Flask can be combined to build a real-world application.

1.3 Proposed System

The proposed system is an AI-powered travel chatbot web application that offers users quick and interactive access to essential information about any city. It is designed to reduce the complexity of travel planning by using Artificial Intelligence and APIs to generate summarized, relevant travel content in real-time. This system allows users to simply enter a city name, and in response, it provides:

- A brief city description
- Top 3 places to visit
- 3 recommended restaurants
- 3 popular local dishes
- 3 major shopping malls
- Google Maps links for each location .

1.4 Scope

The scope of this project encompasses the design, development, and deployment of an AI-based travel chatbot web application that assists users in obtaining key travel information about cities around the world. The chatbot combines Artificial Intelligence, API integration, and a user-friendly web interface to deliver real-time, reliable, and structured travel suggestions.

1.5 Conclusion

The AI Travel Chatbot effectively combines artificial intelligence, natural language processing, and web development to create a smart, user-friendly travel assistant. It automates the retrieval of city descriptions, tourist spots, foods, malls, and restaurants—complete with Google Maps links—reducing manual research and presenting clear, structured results. Built using Flask and OpenAI models, the system demonstrates practical AI integration and serves as a scalable foundation for advanced features like personalized itineraries, real-time data, and multilingual support, making it a meaningful step toward intelligent travel planning. Overall, the project is a step toward smarter travel planning tools and reflects the potential of modern AI technologies in solving everyday user problems in an efficient and engaging way.

Chapter 2

LITERATURE SURVEY

Similar Concept Google Travel Assistant (formerly Google Trips) It offers a seamless experience by automatically extracting travel-related information from a user's Google account and generating travel plans. While this tool is user-friendly and convenient, it heavily relies on the Google ecosystem and offers limited customization for developers or academic purposes. Its AI features focus more on personalization rather than natural language interaction. The development of AI-based travel chatbots has gained considerable momentum in recent years, reflecting a growing interest in using artificial intelligence to enhance travel experiences. Several studies and technologies have contributed to the evolution of such systems, focusing on data integration, conversational AI, user-friendly interfaces, and real-time recommendations for tourists.

the AI Travel Chatbot project(GUIDE YOUR TRIP) stands out as a well-rounded, accessible, and educationally valuable application. It leverages a large language model (LLaMA via OpenRouter and OpenAI API) to deliver structured travel information based on user input. The chatbot is capable of understanding user queries and responding with clearly organized recommendations, including must-visit places, local

cuisines, popular malls, and notable restaurants for any specified city. What sets this project apart is its integration of multiple real-time data sources, such as Wikipedia for concise city descriptions and Google Maps for precise location access. The user interface, built with Flask, is lightweight and intuitive, making the chatbot accessible through a simple web form. Unlike commercial applications, this project prioritizes customization, clarity, and developer flexibility, making it ideal for educational use, research, and further development. Moreover, the chatbot provides structured responses in a fixed format, ensuring clarity and consistency. This contrasts with many AI-based applications that often return verbose or unstructured answers. Additionally, the modular design of the project opens doors to future extensions such as voice-based interaction, multilingual support, itinerary generation, and even integration with travel booking services. Wikipedia API Integration : Research by Miller et al. (2018) highlights the significance of reliable open-source encyclopedic content in educational and informational systems. The use of the Wikipedia API for city descriptions provides factual and up-to-date summaries without manual data entry. According to Singh and Thomas (2020), such integration improves scalability and content credibility in travel platforms, especially for dynamic or location-based queries.

OpenAI and LLaMA API for AI-Generated Content: The role of generative AI in natural language response generation has been explored by Vaswani et al. (2017) and further extended by Brown et al. (2020) in the development of GPT models. The integration of LLaMA through OpenRouter aligns with trends in using transformer-based models for real-time content generation. According to Zhao and Chen (2021), such models are capable of dynamically generating location-specific travel recommendations with contextual coherence, offering a personalized user experience.

Google Maps API Integration: A study by Gupta et al. (2019) emphasizes

the importance of geographic visualization tools in travel applications. The integration of Google Maps enables users to directly access locations mentioned in recommendations, improving user engagement and practical usability. Mapping APIs have been shown to enhance decision-making during travel planning by offering spatial context, directions, and nearby attractions.

User Interface and Flask Web Framework The design of intuitive web interfaces remains a crucial factor in user satisfaction and retention. Kim et al. (2018) argue that minimal and interactive UIs contribute significantly to technology acceptance across age groups. The choice of Flask as a lightweight Python framework allows for rapid development and deployment. According to Jaiswal and Mehta (2020), Flask's modular design and support for Jinja templates improve scalability and responsiveness in small to mid-sized AI applications.

Programming Language (Python) Python remains the most preferred language in the AI and web development ecosystem. Its clean syntax, extensive libraries, and support for REST APIs make it an ideal choice for chatbot development (Patel Kumar, 2021). The use of Python allows seamless integration of AI models, external APIs, and web services within a single technology stack.

In conclusion, while many existing travel assistant applications offer impressive features, they often cater to commercial needs or rely on closed ecosystems. The AI Travel Chatbot project distinguishes itself through its conversational intelligence, real-time integrations, and open, flexible architecture. It not only enhances the user's ability to explore travel options in a personalized and interactive manner but also serves as a robust foundation for future AI-driven travel innovations.

Chapter 3

REQUIREMENT SPECIFICATION

3.1 Software Requirements

- Python 3.10 or higher
- Flask Web Framework
- Visual Studio Code or any Python-compatible IDE
- OpenAI API access via OpenRouter
- Wikipedia API
- Web browser (Chrome, Firefox, Edge)
- Jinja2 Template Engine
- Google Maps integration (via URL generation)
- OS: Windows 10 / Linux / macOS

3.2 Hardware Requirements

- Processor: Intel i5 or higher

- CPU Speed: 2.9 GHz or better
- RAM: Minimum 4 GB
- Hard Disk: 80 GB minimum
- Internet Connection: Required for API communication
- Input Devices: Keyboard, Mouse
- Output Devices: Monitor, Speakers (optional for voice extensions)

3.3 Conclusion

The software and hardware requirements ensure efficient development and smooth execution of the AI Travel Chatbot. Using Visual Studio Code with Python and Flask provides a flexible and lightweight development environment. The system supports modern operating systems and integrates OpenAI (LLaMA model), Wikipedia, and Google Maps APIs for real-time, dynamic content. Recommended hardware—Intel i5 processor with at least 4GB RAM—ensures reliable performance without bottlenecks.

Chapter 4

WORKING OF SYSTEM

4.1 SYSTEM ARCHITECTURE

4.1.1 Overview

The system architecture of the AI-based travel chatbot application is composed of multiple coordinated components, each fulfilling a unique role in ensuring seamless data retrieval, AI interaction, and user presentation. These components include:

- **Flask**: Serves as the web development framework that handles routing, HTTP requests, and rendering HTML templates. It provides lightweight yet powerful tools to create responsive web applications.
- **OpenRouter API (LLaMA/OpenAI)**: Facilitates interaction with an advanced language model (LLaMA 3.3 via OpenRouter), which generates city-specific travel suggestions including places to visit, local foods, malls, and restaurants.
- **Wikipedia API**: Fetches factual, concise city descriptions, providing

background knowledge that is reliable and up-to-date for users.

- **Google Maps Integration**: Automatically constructs Google Maps search links for each location (e.g., monuments, malls, restaurants), enabling users to directly access the corresponding navigation and details.
- **Jinja2 Templating (Flask's rendering engine)**:Dynamically injects API-generated content into the HTML structure, allowing personalized and real-time updates on the frontend based on user queries.

The architecture follows a modular design pattern, broken into the following logical components:

Input Module:A text form field that captures the user's city input via a simple UI. Processing Module (Backend Flask Logic):Sends the city name to the Wikipedia API to fetch the description.Passes the city name to the LLaMA model through OpenRouter for travel recommendations.Parses and categorizes the received data into places, foods, malls, and restaurants.Generates Google Maps links for each item. Output Module (Frontend Display):Uses Jinja2 templates to populate and display the city's description and recommendations.Each result is shown in a visually structured format with clickable Google Maps links for each attraction or location.

By carefully designing and implementing the below robust architecture, the project gains scalability, maintainability, and flexibility. The architecture defines the relationships between components, ensuring seamless communication and efficient workflows.

Figure 4.1: System Architecture

4.2 Wikipedia API

4.2.1 REST API

The Wikipedia API (MediaWiki API) offers methods to: Retrieve summaries, full page content, images, and metadata from Wikipedia articles. Search Wikipedia by keywords and get related page titles. Access page revisions, references, categories, and links within articles. Perform queries using JSON or XML formats through RESTful endpoints.

4.2.2 Back ground

The Wikipedia API provides programmatic access to one of the largest open encyclopedias, enabling developers to incorporate reliable and up-to-date information into their apps. It is commonly used for educational tools, knowledge retrieval, and fact-checking applications. The API is free to use but enforces fair-use limits to ensure service availability.

4.2.3 flask integration

Developers often use Flask to create web applications or microservices that fetch and display Wikipedia data dynamically. Flask's minimalistic framework allows easy routing for search queries and article display. By combining Flask with the Wikipedia API, apps can provide fast, user-friendly access to encyclopedia content, including summaries, images, and related articles.

4.3 OpenAI API

4.3.1 REST API

Generate and complete text based on prompts using advanced language models like GPT-4 and GPT-3.5. Perform tasks such as conversation, summarization, translation, code generation, and content creation. Manage fine-tuned models, monitor usage, and handle authentication via API keys. Interact with chat-based and completion-based models using RESTful JSON requests and responses.

4.3.2 Back ground

OpenAI API enables developers to integrate powerful AI capabilities without building their own models. It significantly reduces the time and effort to create natural language understanding and generation features in applications. The API supports a wide variety of use cases, including chatbots, virtual assistants, automated content generation, and more. Using OpenAI's hosted models offers scalability, regular updates, and state-of-the-art performance.

4.3.3 flask integration

Flask is often used to build lightweight web applications that interface with the OpenAI API. Flask's simplicity and extensibility make it easy to create backend services that send user inputs to the OpenAI API and return responses in real time. Flask's routing and templating support help build interactive web apps or API.

Chapter 5

SYSTEM DESIGN

Design is the abstraction of a solution it is a general description of the solution to a problem without the details. Design is view patterns seen in the analysis phase to be a pattern in a design phase. After design phase we can reduce the time required to create the implementation.

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

What is UML?

UML is an acronym that stands for Unified Modelling Language. Simply put, UML is a modern approach to modelling and documenting software. In fact, it's one of the most popular business process modelling techniques.
It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

Building Blocks of the UML: The vocabulary of the UML encompasses

three kinds of building blocks.

- **Things:** Things are the abstractions that are first-class citizens in a model
- **Relationships:** ; relationships tie these things together
- **Diagrams:** diagrams group interesting collections of things

5.0.1 Use Case Diagram

Use case diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system.

A use case represents a particular functionality of a system. Hence, use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors. In this project admin and user are the actors

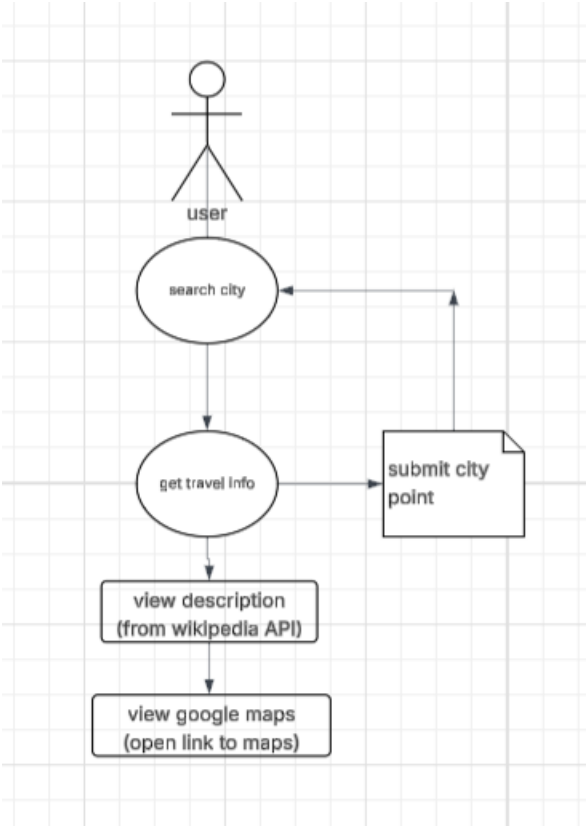


Figure 5.1: Use Case

5.0.2 Class Diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

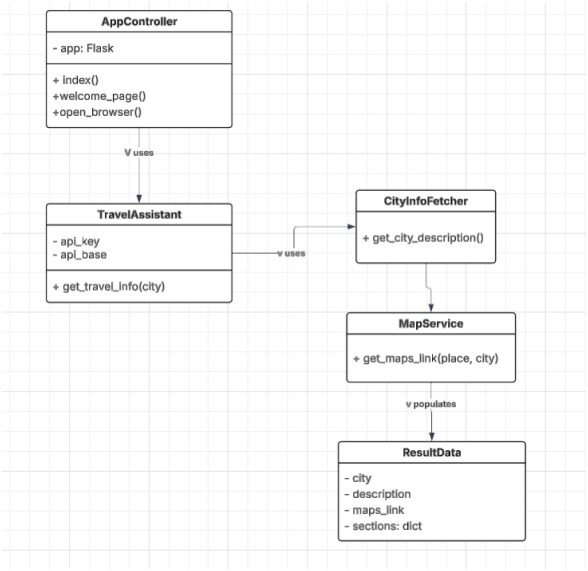


Figure 5.2: class Diagram

5.0.3 Flow Chart Diagram

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

5.0.4 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) visually represents the dynamic interactions and chronological order of messages between various objects or components within a system. Lifelines, depicted as vertical dashed lines, symbolize the existence of objects over time.

Actors, often represented as stick figures, denote external entities interacting with the system. Messages, depicted as arrows flowing vertically between lifelines, illustrate communication events. Activation bars on lifelines indicate the period during which an object is active, processing a message. Return messages, denoted by dashed lines with arrows, signify the flow of control back to the message sender. Self-messages, represented by looped arrows, depict an object sending a message to itself. Interaction occurrences frame repeated sequences of messages, enhancing the diagram's expressiveness.

In essence, a sequence diagram provides a dynamic view of a system, allowing developers and analysts to understand the chronological flow of interactions between system components. This visual representation aids in designing, analyzing, and documenting the behavior of complex systems, fostering a comprehensive understanding of how objects collaborate and communicate during the execution of a particular scenario.

5.0.5 Activity Diagram

Activity diagrams are used to document workflows in a system, from the business level down to the operational level. The general purpose of Activity diagrams is to focus on flows driven by internal processing vs. external events. Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.

5.0.6 Conclusion

In summary, the use case diagram encapsulates user-system interactions, guiding system design based on scenarios, while the class diagram delineates the structural framework by illustrating relationships between classes. The sequence diagram offers a dynamic view of message flow during scenarios, aiding in the identification of key functionalities, and the flow chart diagram provides a step-by-step representation of system processes, enhancing transparency in the virtual assistant's workflow. Collectively, these diagrammatic representations facilitate a holistic understanding of the virtual assistant's functionality, structure, dynamics, and operational flow, serving as invaluable tools for effective design, development, and communication of the system's intricacies.

Chapter 6

Implementation

6.1 OPENAI CHATBOT INTEGRATION

6.1.1 Travel Info Manager

When implementing the AI Travel Chatbot, we created a TravelInfoManager that encapsulates interaction with the OpenAI API. This manager is responsible for sending the city query prompt, receiving the formatted travel information, and returning it as a structured text response. The strict prompt format ensures the returned data can be parsed reliably.

6.1.2 Connecting to OpenAI API

To query the OpenAI API, we first prepare a system message setting the role of the assistant, followed by a user message containing the prompt specifying the city and the output format. This is done using the openai.ChatCompletion.create() method, where the model (e.g., ”meta-llama/llama-3.3-8b-instruct:free” or ”gpt-4”) and messages array are

passed. The code snippet above shows this interaction .the prompt is carefully crafted to instruct the model to return only four categories (Places, Foods, Malls, Restaurants) in list format, avoiding any extra text.

```python
response = openai.ChatCompletion.create(
    model="meta-llama/llama-3.3-8b-instruct:free",
    messages=[
        {"role": "system", "content": "You are a travel assistant."},
        {"role": "user", "content": prompt}
    ]
)
```

Figure 6.1: code

6.1.3 Parsing the AI Response

After receiving the raw response text from OpenAI, the chatbot parses it into structured data. This involves splitting the text by lines and categorizing items under the correct sections by detecting section headers such as Places:, Foods:, Malls:, and Restaurants:. Each item line starting with a list marker (- or •) is cleaned and appended to the respective list. This allows the frontend to easily access and display the data.

```python
@app.route("/index", methods=["GET", "POST"])
def index():
    city = request.form.get("city", "").strip()
    if request.method == "POST" and city:
        description = get_city_description(city)
        raw_info = get_travel_info(city)
        section_data = parse_travel_info(raw_info)
```

Figure 6.2: example

6.1.4 Flask Web Server and Routes

The root (”/”) serves a welcome page with a simple form to input the city name. The /index endpoint handles GET and POST requests; on POST, it

takes the city, fetches the city description from Wikipedia, calls TravelInfoManager to get travel info from OpenAI, parses it, and returns a rendered template with all details.

6.2 Backend

6.2.1 Data Aggregation and Prompt Engineering

External API Integration: Uses Wikipedia API to fetch city descriptions and Google Maps API (or links) for location navigation.
Prompt Engineering: The system crafts prompts like: "User wants information on City. Provide a short description, top 5 tourist spots, local foods, shopping areas, and restaurants."
LLM Integration: Flask communicates with OpenAI's GPT-4 via REST to generate content in a conversational tone.

6.2.2 Templating with Jinja2

Jinja2 templates dynamically render chatbot output, converting structured travel data into clean and readable HTML content.

6.2.3 Final Output

Chat Response Rendering: The Flask backend sends the AI-generated response to the frontend where it's displayed in a styled chatbot interface.
Google Maps Integration: Locations are presented with clickable Google Maps links for easy user navigation.
Session Storage: Optionally stores chat history using Flask sessions or localStorage on the frontend for continuity.

6.2.4 Deployment

Cloud or Local Deployment: Flask app is hosted using platforms like Heroku, Railway, or Render; or containerized with Docker for portability.
Frontend Integration: HTML/CSS frontend is integrated with the backend using Jinja2 for dynamic content and styling.
Responsive UI: CSS ensures the chatbot is accessible and responsive across devices, with features like scrollable chat history, input forms, and animated responses.

6.3 code

6.3.1 APP.PY

app.py M | welcome_page.html | index.html M | styles.css M

C:\Users\sadan\OneDrive\Desktop\3-II project\app.py • Modified

```python
from flask import Flask, render_template, request
import wikipedia
import openai
import urllib.parse
openai.api_base = "https://openrouter.ai/api/v1"
openai.api_key = "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"
openai.default_headers = {
    "Authorization": "Bearer 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"
}
def get_maps_link(place, city):···

def get_city_description(city):
    try:
        summary = wikipedia.summary(city, sentences=3, auto_suggest=False)
        return summary
    except wikipedia.DisambiguationError as e:
        try:
            summary = wikipedia.summary(e.options[0], sentences=3, auto_suggest=False)
            return summary
        except:
            return "Couldn't find a proper description for this city."
    except:
        return "Couldn't find a proper description for this city."

def get_travel_info(city):
    prompt = f"""You are a travel assistant. Provide information about {city} in the following strict format. Do not

```

app.py M | welcome_page.html | index.html M | styles.css M

C:\Users\sadan\OneDrive\Desktop\3-II project\app.py • Modified

```python
from flask import Flask, render_template, request
import wikipedia
import openai
import urllib.parse
openai.api_base = "https://openrouter.ai/api/v1"
openai.api_key = "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"
openai.default_headers = {
    "Authorization": "Bearer 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"
}
def get_maps_link(place, city):...

def get_city_description(city):
    try:
        summary = wikipedia.summary(city, sentences=3, auto_suggest=False)
        return summary
    except wikipedia.DisambiguationError as e:
        try:
            summary = wikipedia.summary(e.options[0], sentences=3, auto_suggest=False)
            return summary
        except:
            return "Couldn't find a proper description for this city."
    except:
        return "Couldn't find a proper description for this city."

def get_travel_info(city):
    prompt = f"""You are a travel assistant. Provide information about {city} in the following strict format. Do not

```

app.py M | welcome_page.html | index.html M | styles.css M

app.py > ...

```python
def get_travel_info(city):
    return response.choices[0].message.content
app = Flask(__name__)
def get_city_description(city):...
def get_maps_link(place, city=""):
    query = f"{place}, {city}" if city else place
    return f"https://www.google.com/maps/search/{urllib.parse.quote(query)}"

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

@app.route("/index", methods=["GET", "POST"])
def index():
    city = request.form.get("city", "").strip()
    result = {}

    if request.method == "POST" and city:
        description = get_city_description(city)
        maps_link = get_maps_link(city)

        def parse_travel_info(raw_text):
            sections = {
                "places": [],
                "foods": [],
                "malls": [],
                "restaurants": []
            }

```

app.py M | welcome_page.html | index.html M | styles.css M

app.py > index > parse_travel_info

```python
def index():
        def parse_travel_info(raw_text):

            current_section = None
            for line in raw_text.split('\n'):
                line = line.strip()
                if not line:
                    continue

                # Detect section headers
                if line.lower().startswith("places:"):
                    current_section = "places"
                elif line.lower().startswith("foods:"):
                    current_section = "foods"
                elif line.lower().startswith("malls:"):
                    current_section = "malls"
                elif line.lower().startswith("restaurants:"):
                    current_section = "restaurants"
                elif current_section and line.startswith(("-", "•")):

                    cleaned = line.lstrip("-• ").strip()
                    if cleaned:
                        sections[current_section].append(cleaned)

            return sections
        raw_info = get_travel_info(city)
        section_data = parse_travel_info(raw_info)
        print("RAW INFO:", raw_info)
        print("PARSED DATA:", section_data)
```

6.3.2 index.html

app.py M | welcome_page.html | index.html M | styles.css M

templates > index.html > html > body > div.container > form

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title> Guide Your Trip</title>
    <style>
        body {
            font-family: Arial, sans-serif;

            background: url('https://media.istockphoto.com/id/486726986/vector/travel-seamless-background.jpg?s=612x6
            background-size:flex;
            position:absolute;
            top: 0;
            left: 0;
            min-width:100%;
            min-height: 100vh;
            overflow-y: auto;
        }
        .container{

            position:absolute;
            top: 0;
            left: 0;
            min-width: 100%;
            min-height: 100vh;
            overflow-y: auto;
            background-color: rgba(133, 202, 225, 0.829);
            z-index: 1;
            background-size:cover;
```

app.py M | welcome_page.html | index.html M | styles.css M

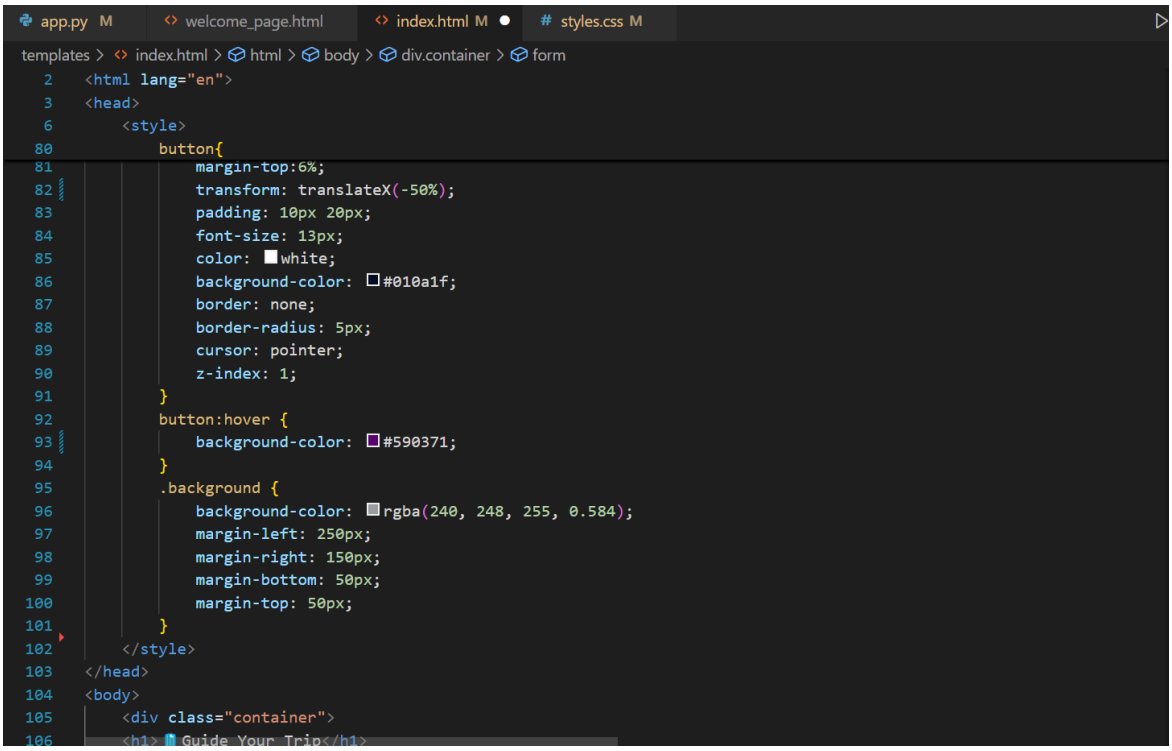
templates > index.html > html > body > div.container > form

```
<html lang="en">
<head>
    <style>
        .container{
        }
        h1 {
            margin-bottom: 0px;
            text-align: center;
            font-size: 100px;
            font-family: Arial, sans-serif;
            color: #a8d5f5;
            text-align: center;
            background-color: #010a1f;
            background-size: cover;
            margin-top: 0px;
        }
        .section-header {
            font-size: 24px;
            margin-top: 25px;
            font-weight: bold;
            color: #01101f;
            text-align: left;
            margin-left: 50px;
            margin-right: 50px;
            padding-top: 20px;
        }
        .item {
            margin: 0px 0;
            font-size: 18px;
            margin-left: 50px;
```

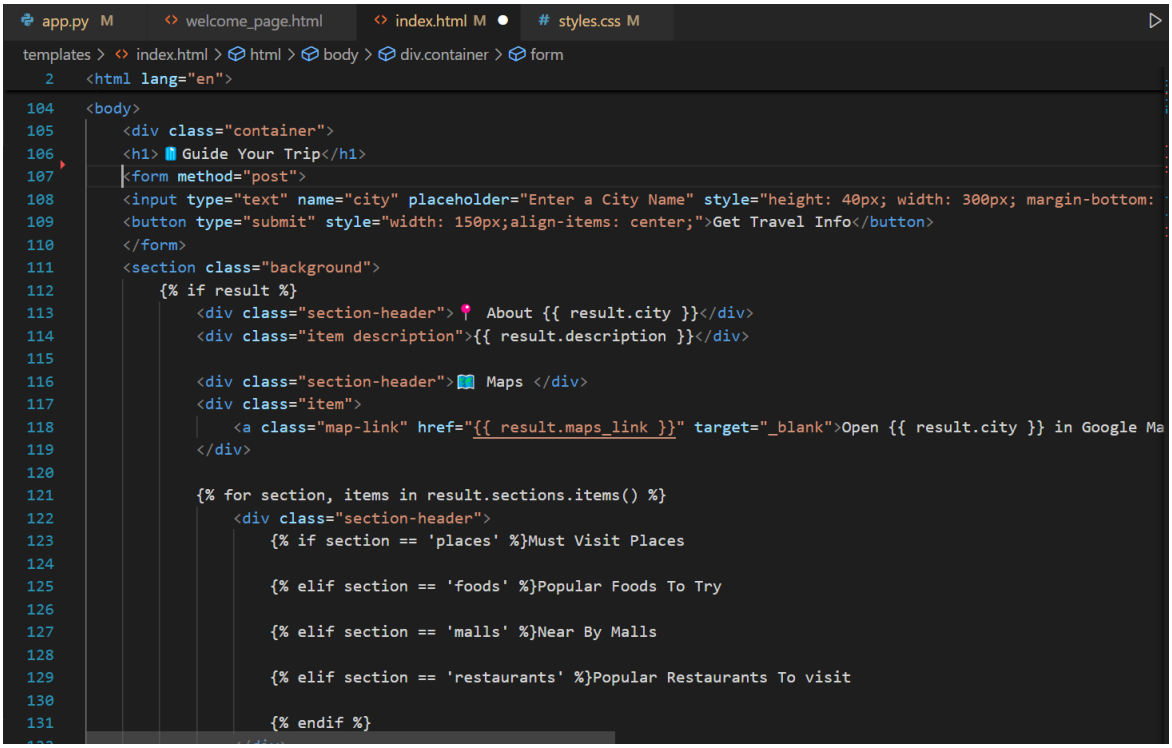
app.py M | welcome_page.html | index.html M | styles.css M

templates > index.html > html > body > div.container > form

```
<html lang="en">
<head>
    <style>
        .item {
            margin-right: 50px;
            margin-bottom: 30px;
        }
        .map-link {
            color: #7a0202;
            text-decoration: none;
            text-align: right;
            margin-right: 200px;
        }
        .map-link:hover {
            color: #010d1c;
        }
        .description {
            font-size: 16px;
            margin-bottom: 20px;
            color: #040404;
            text-align: left;
            margin-left: 50px;
            margin-right: 50px;
        }
        form {
            margin-bottom: 30px;
            text-align:center ;
        }
        button{
            margin-top:6%;
```


app.py M
welcome_page.html
index.html M
styles.css M
templates > index.html > html > body > div.container > form
<html lang="en">
<head>
<style>
button{
margin-top:6%;
transform: translateX(-50%);
padding: 10px 20px;
font-size: 13px;
color: white;
background-color: #010a1f;
border: none;
border-radius: 5px;
cursor: pointer;
z-index: 1;
}
button:hover {
background-color: #590371;
}
.background {
background-color: rgba(240, 248, 255, 0.584);
margin-left: 250px;
margin-right: 150px;
margin-bottom: 50px;
margin-top: 50px;
}
</style>
</head>
<body>

<h1> Guide Your Trip</h1>

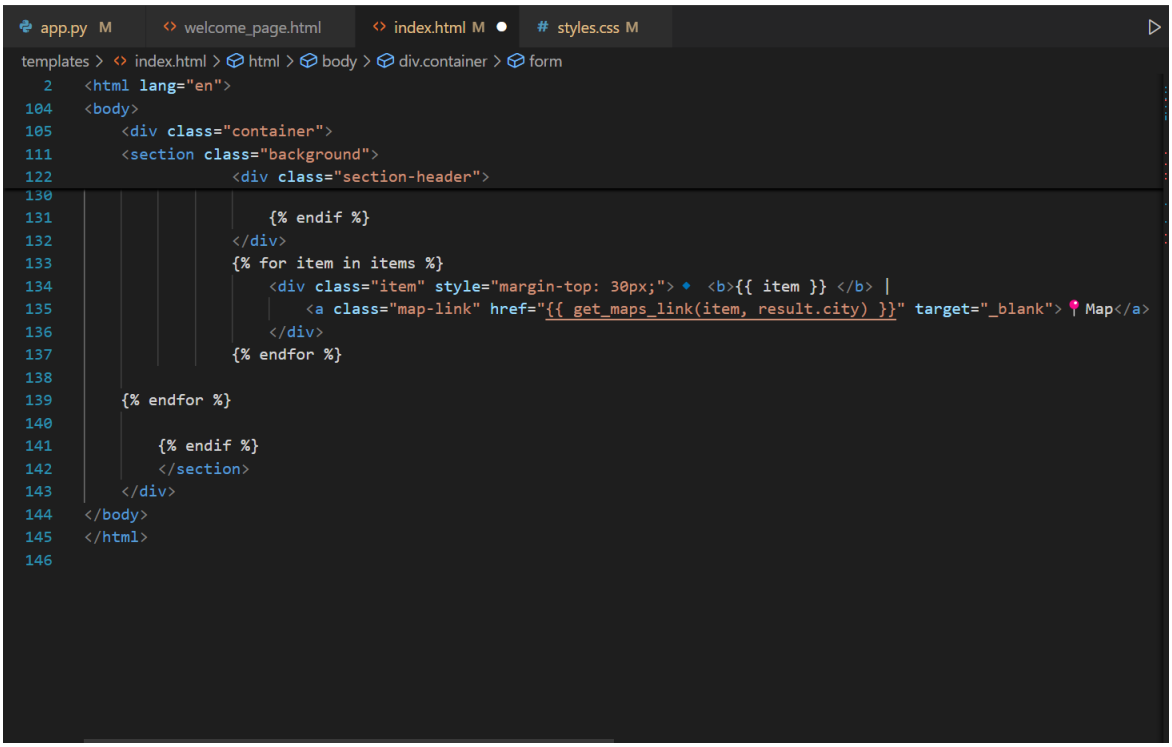

app.py M
welcome_page.html
index.html M
styles.css M
templates > index.html > html > body > div.container > form
<html lang="en">
<body>

<h1> Guide Your Trip</h1>
<form method="post">
<input type="text" name="city" placeholder="Enter a City Name" style="height: 40px; width: 300px; margin-bottom:
<button type="submit" style="width: 150px;align-items: center;">Get Travel Info</button>
</form>
<section class="background">
{% if result %}
📍 About {{ result.city }}
{{ result.description }}
🗺️ Maps

<a class="map-link" href="{{ result.maps_link }}" target="_blank">Open {{ result.city }} in Google Ma

{% for section, items in result.sections.items() %}

{% if section == 'places' %}Must Visit Places
{% elif section == 'foods' %}Popular Foods To Try
{% elif section == 'malls' %}Near By Malls
{% elif section == 'restaurants' %}Popular Restaurants To visit
{% endif %}


app.py M
welcome_page.html
index.html M
styles.css M
templates > index.html > html > body > div.container > form
2 <html lang="en">
104 <body>
105
111 <section class="background">
122
130
131 {% endif %}
132
133 {% for item in items %}
134 • {{ item }} |
135 Map
136
137 {% endfor %}
138
139 {% endfor %}
140
141 {% endif %}
142 </section>
143
144 </body>
145 </html>
146

RESULTS

The ”Guide Your Trip” application successfully provides users with a personalized travel guide based on the city they enter. Upon submitting a city name, the application dynamically generates and displays the following information: Brief Description: A short summary of the city’s background or importance. Google Maps Link: A direct link to view the city on Google Maps. Tourist Attractions: A list of popular places to visit in the city. Local Cuisine: Suggested local foods that travelers should try. Shopping Destinations: A list of nearby malls and shopping areas. Restaurants: Recommended restaurants to explore local dining experiences. The results are displayed in a clean, visually appealing format with clickable map links for each item, helping users to plan their trip easily and efficiently. This output is generated using: Live interaction with the OpenAI API (for travel suggestions) Wikipedia API (for city summaries) Flask (to handle user input and render results) CSS-styled frontend for enhanced user experience

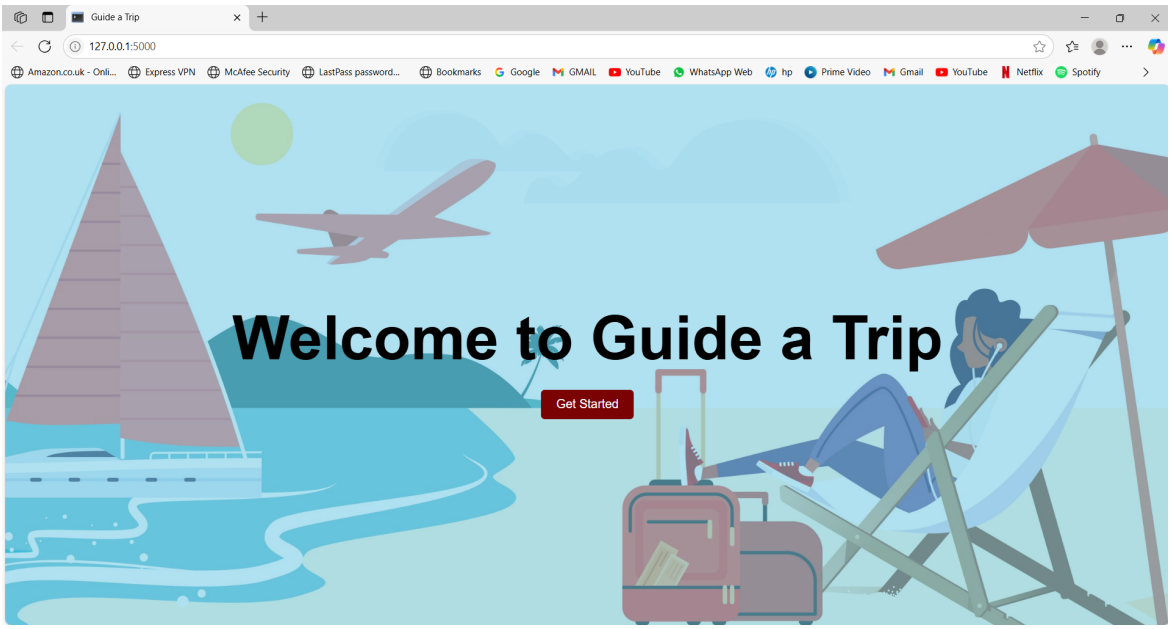


Figure 6.3: landing page

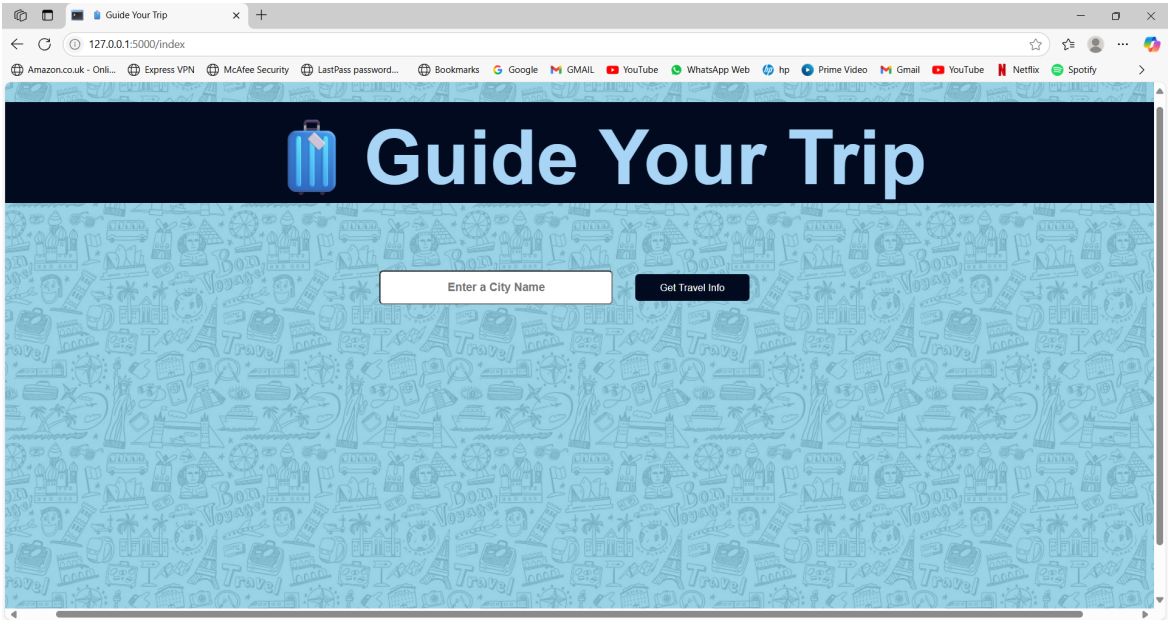


Figure 6.4: home page

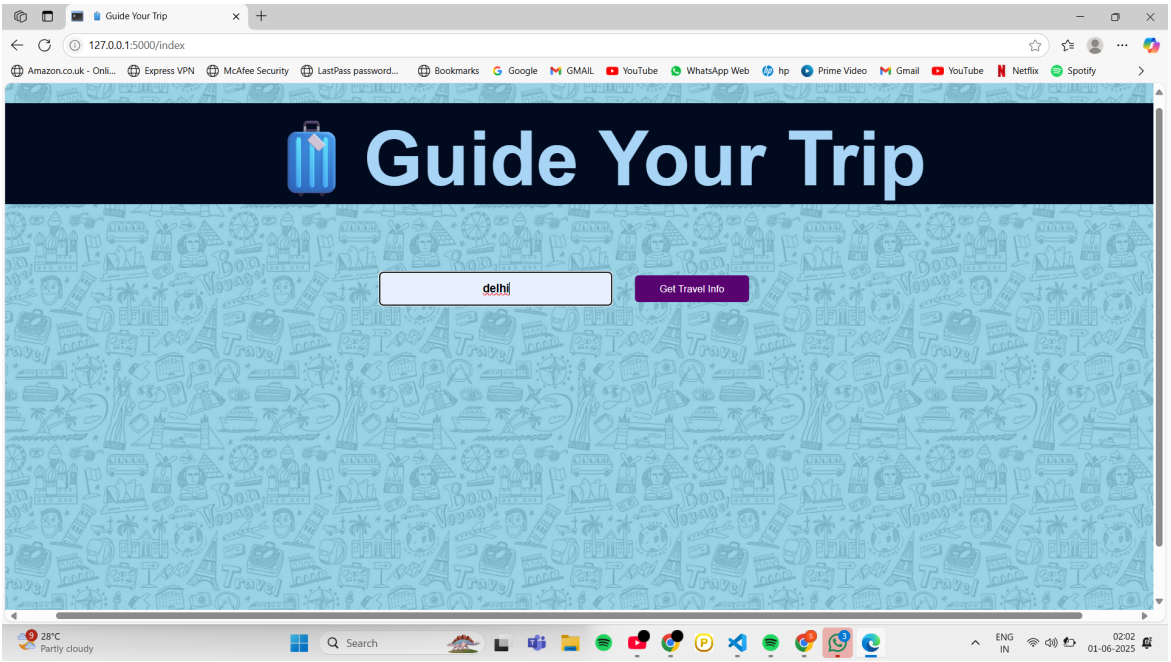


Figure 6.5: input

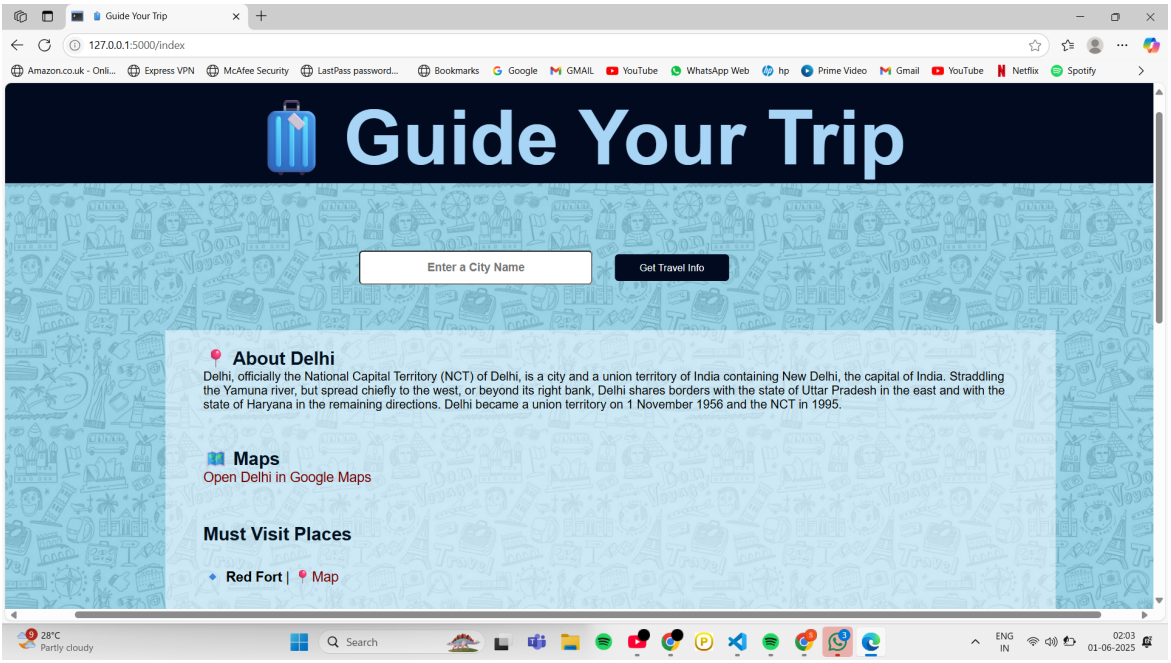


Figure 6.6: chatbot results

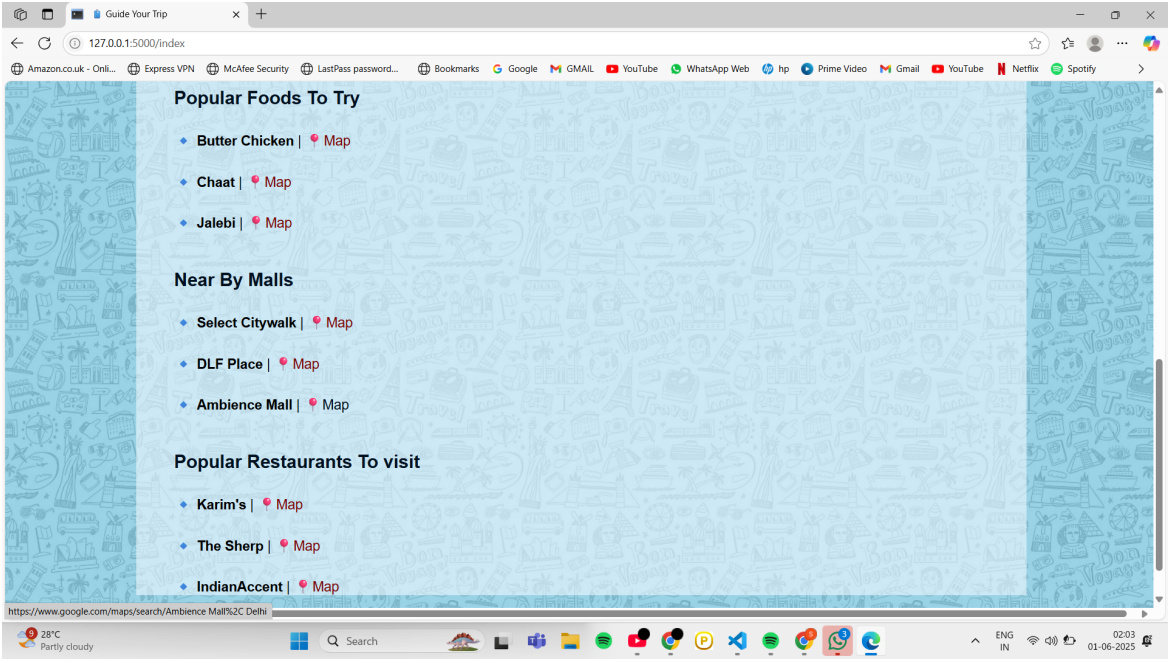


Figure 6.7: chatbot results

Chapter 7

TESTING

7.1 White Box Testing

White box testing is a form of application testing that provides the tester with complete knowledge of the application being tested, including access to source code and design documents.

7.2 Black Box Testing

Black Box Testing is an important part of making sure software works as it should. Instead of peeking into the code, testers check how the software behaves from the outside, just like users would. This helps catch any issues or bugs that might affect how the software works.

7.3 Unit Testing

Unit testing is the process where you test the smallest functional unit of code. Software testing helps ensure code quality, and it's an integral part of software development. It's a software development best practice to write software as small, functional units then write a

unit test for each code unit.

7.4 Integration Testing

Integration testing – also known as integration and testing (IT) – is a type of software testing in which the different units, modules or components of a software application are tested as a combined entity. However, these modules may be coded by different programmers.

7.5 Validation Testing

Validation testing is the process of assessing a new software product to ensure that its performance matches consumer needs. Product development teams might perform validation testing to learn about the integrity of the product itself and its performance in different environments

7.6 System Testing

System testing examines every component of an application to make sure that they work as a complete and unified whole.

CONCLUSION

The AI Travel Chatbot is a smart, user-friendly web application that uses natural language processing (NLP) and external APIs to assist with city-based travel planning. Powered by OpenAI's language models, it responds to user queries with curated suggestions for tourist spots, local foods, shopping malls, and restaurants. Built on a Flask backend, it fetches city descriptions from Wikipedia and provides Google Maps links for easy navigation.This project shows how AI and real-time data can streamline trip planning by reducing the need for manual research. The system is designed to be fast, responsive, and easy to use, even for non-technical users. By automating information gathering and organizing travel content clearly, it enhances user experience and saves time. The integration of accurate data sources ensures that suggestions are both reliable and relevant.Future upgrades may include voice input, multilingual support, smart itinerary generation, and booking integration, making the system more interactive and helpful for travelers

REFERENCES

1. https://en.wikipedia.org/wiki/Large_language_model
2. https://en.wikipedia.org/wiki/Wikipedia:About
3. https://pypi.org/project/wikipedia/
4. https://pypi.org/project/openai/
5. http://flask.palletsprojects.com/en/stable/
6. https://docs.python.org/3/library/urllib.html
7. https://openrouter.ai/docs/quickstart
8. https://openrouter.ai/meta-llama/llama-3.3-8b-instruct:free/api