Sindhu K | Technical Architect

HEXAWARE TECHNOLOGIES  SIRUSERI

Nov 2024

FLYEASY PERSONAL FLIGHT ASSISTANT

Gen AI - Project

**Contents**

[**1.** **Project Overview** 1](#_Toc181890156)

[**2.** **Project Goals** 2](#_Toc181890157)

[**3.** **Data Sources** 3](#_Toc181890158)

[**4.** **Design** 3](#_Toc181890159)

[**Architecture** 3](#_Toc181890160)

[**Technologies Used** 4](#_Toc181890161)

[**5.** **Code Structure** 4](#_Toc181890162)

[**Directory Layout** 4](#_Toc181890163)

[**Modules and Functions** 5](#_Toc181890164)

[**6.** **TESTCASES** 5](#_Toc181890165)

[**7.** **Challenges Faced** 6](#_Toc181890166)

[**Technical Challenges** 6](#_Toc181890167)

[**Data Challenges** 7](#_Toc181890168)

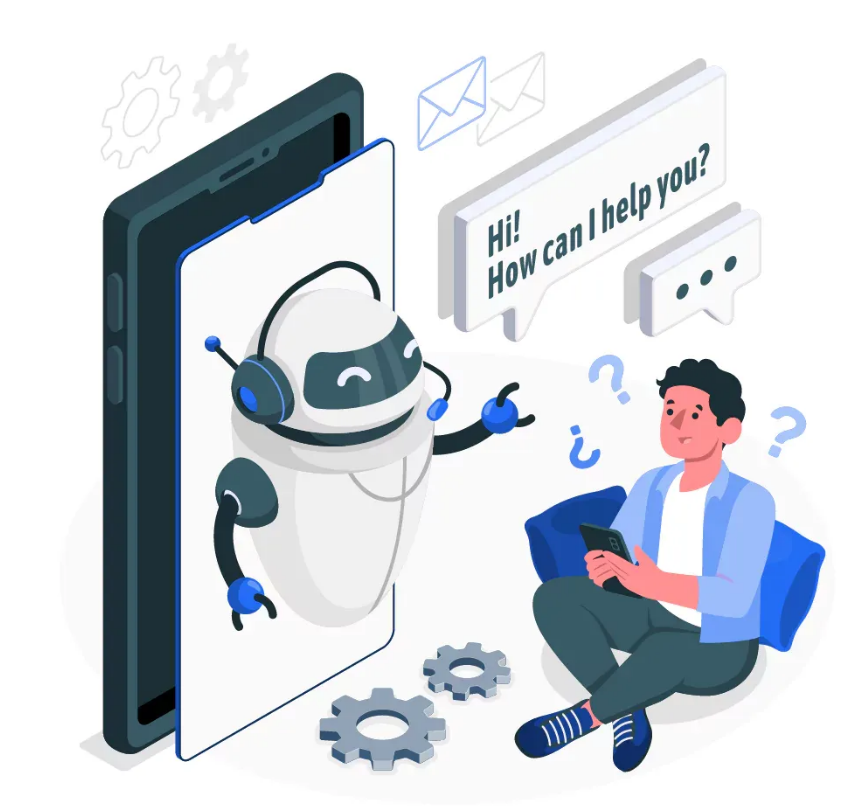
[**Operational Challenges** 7](#_Toc181890169)

[**8.** **Conclusion** 7](#_Toc181890170)

# **Project Overview**

Welcome to FlyEasy: Your Personal Flight AI Assistant

Revolutionizing Your Travel Experience!



Meet FlyEasy, the intelligent and friendly AI assistant designed to make your travel seamless and stress-free. With FlyEasy, managing your travel plans and getting real-time flight support has never been easier. Explore our four exclusive services:

* Discover Flight Information
* Book Your Flight
* Lodge a Complaint
* Know the Weather Information

# **Project Goals**

The primary objectives of the project are:

Create a UI for chatbot conversation:

* To provide users with real-time flight information.
* To assist users in booking flights seamlessly.
* To enable users to lodge complaints efficiently.
* To provide weather information for specific locations.

A screenshot of a computer

Description automatically generated

# **Data Sources**

The project utilizes several data sources:

* **Azure OpenAI gpt-4o-mini**: For generating responses and managing conversations.
* **OpenWeatherMap API**: For fetching weather information.
* **AviationStack API**: For fetching flight information.
* **JSON File**: For getting the flight IATA code based on location

# **Technologies Used**

* **Programming Language**: Python
* **IDE**: VS Code
* **Frameworks and Libraries**:
  + Streamlit for the web interface
  + Azure OpenAI's gpt-4o-mini for natural language processing
  + Requests for API calls
  + dotenv for managing environment variables

# **Design**

## **Architecture**

1. **User Interface (Streamlit)**
   * Streamlit will be used to create an interactive web application where users can input their queries.
   * The interface will have sections for weather information and aviation data.
2. **Natural Language Processing (Azure OpenAI)**
   * Azure OpenAI will be used to process user inputs and determine whether the query is related to weather or aviation.
   * Based on the processed input, the appropriate API (OpenWeatherMap or AviationStack) will be called to fetch the required data.
3. **APIs Integration**
   * **OpenWeatherMap API**: Used to fetch current weather data.
   * **AviationStack API**: Used to fetch real-time flight information.
4. **Additional Techniques**
   * **Few-shot learning:** Enhances model performance with minimal training examples.
   * **Chain of thoughts:** Improves reasoning by breaking down complex queries into smaller steps.
   * **Moderation Check:** Ensures responses are appropriate and within guidelines.

# **Process Flow**

A diagram of a diagram

Description automatically generated

# **Code Structure**

## **Directory Layout**

* **main.py**: The main script that initializes the Streamlit interface, handles user inputs, and manages conversations.
* **functions.py**: Contains helper functions for making API calls and processing data.
* **.env**: Contains environmental variables to store the required API Keys

## **Modules and Functions**

|  |  |  |
| --- | --- | --- |
| Function Name | Objective | Functionality |
| load\_airports\_data(filepath="FlightDetails.json") | Load airport data from a JSON file. | Opens the specified JSON file. Reads and parses the JSON data. Returns the parsed data. The data has mapping of location and IATA codes. |
| get\_iata\_code(city\_name, airports\_data) | Retrieve the IATA code for a given city. | Iterates through the FlightDetails JSON data. Compares the city name (case-insensitive) with the provided city name. Checks if the IATA code is not null or empty. Returns the IATA code if found, otherwise returns None. |
| fetch\_flight\_details(api\_key, origin, destination, travel\_date) | Fetch flight details between specified origin and destination. | Constructs the API URL and parameters. Makes an API request to fetch flight data. Filters flights based on the travel date and status. Sorts the flights by cost. Returns the top 5 flight details including flight name, code, datetime, cities, and cost. |
| get\_flight\_info(loc\_origin, loc\_destination, travel\_date=datetime.now().strftime("%Y-%m-%d")) | Gather flight information for the provided origin, destination, and travel date. | Loads airport data from a JSON file. Retrieves the IATA codes for the origin and destination cities. Fetches flight details using the IATA codes and travel date. Returns the flight data as JSON. |
| fetch\_flight\_details\_using\_flightcode(api\_key, flight\_code) | Fetch flight details based on the flight code. | Constructs the API URL and parameters. Makes an API request to fetch flight data based on the flight code. Retrieves and returns the flight details including flight name, code, datetime, cities, and status. |
| book\_flight(name, passport\_num, flight\_code) | Gather flight booking information for the provided customer name, passport number, and flight code. | Fetches flight details using the flight code. Appends the passenger’s name and passport number to the flight details. Adds a dummy message that flight has been booked. Returns the booking information as JSON. |
| file\_complaint(customer\_name, customer\_email, issue\_desc) | Register a dummy complaint for the provided customer name, email, and description of the issue. | Create a complaint dictionary with the information provided. Adds a dummy status indicating the complaint was successfully filed. Returns the complaint information as JSON. |
| get\_location\_details(location) | Fetch location details from an external API for the provided location. | Constructs the API URL and parameters. Makes an API request to fetch location data. Parses and returns the location data. |
| get\_weather\_details(lat, lon) | Collect real-time weather information from an external API for the latitude and longitude provided. | Constructs the API URL and parameters. Makes an API request to fetch weather data. Parses and returns the weather data. |
| get\_weather\_report(location) | Gather the weather report for the provided location. | Fetches location details using the provided location name. Extracts latitude and longitude from the location data. Fetches weather details using the latitude and longitude. Constructs a weather report dictionary. Returns the weather report as JSON. |

# **TESTCASES**

There are seven test cases captured as stated below:

## **Warm Welcome:**

Ensure the system responds appropriately to a normal greeting.

A screenshot of a chat

Description automatically generated

## **Flight Finder:**

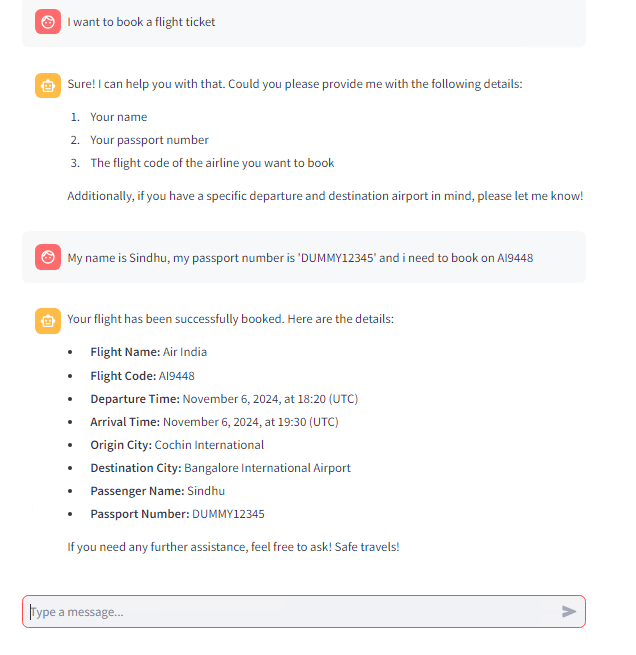
Request real-time flight information using the AviationStack API.

A screenshot of a chat

Description automatically generated

## **Flight Booking:**

Request real-time flight information and simulate booking the flight with a dummy message.



## **Complaint Box:**

File a complaint and verify the system's response.

A screenshot of a computer screen

Description automatically generated

## **Safety First:**

Perform a moderation check to flag and handle harmful conversations.

A screenshot of a computer

Description automatically generated

## **Weather Watch:**

Get real-time weather information using the OpenWeatherMap API.

A screenshot of a computer

Description automatically generated

## **Goodbye:**

End the conversation and ensure the interaction closes gracefully.

A screenshot of a computer

Description automatically generated

# **Challenges Faced**

* **Integration with APIs**: Ensuring seamless integration with external APIs for fetching weather and flight information. Included more than one APIs to get the relevant information. Used data dumps for the IATA code mapping as unable to find a free api.
* **Handling User Inputs**: Accurately processing and understanding user inputs to provide relevant responses based on function calling.
* **Data Availability**: Ensuring real-time availability of flight and weather data from external APIs.
* **Data Quality**: Handling inconsistencies and errors in data fetched from external sources as it hallucinates sometimes.
* **Resource Limitations**: Using Free APIs have limited usage so have to create multiple accounts to use the free subscriptions.
* **User Experience**: Designing an intuitive and user-friendly interface for seamless interaction.Even a small error will collapse the whole application.

# **SOURCE CODE**

# **Conclusion**

The "FlyEasy" personal flight assistant successfully provides users with real-time flight support, booking services, complaint lodging, and weather information. Future improvements could include expanding the range of services offered, enhancing the user interface, and improving the accuracy of responses.

A cartoon character with a blue text

Description automatically generated