Project Overview

Aim:

- Enhance flight navigation systems by developing a software solution.

Key Challenges Addressed:

- Adverse weather conditions
- Unavailable GPS signals
- Various environmental factors

Objectives

1. Integration of Real-time Data:

- Identify optimal flight paths.
- Provide risk assessments.
- Suggest alternative routes.

2. Leveraging Advanced Algorithms:

- Improve safety in flight navigation.
- Enhance efficiency and reliability.

3. Health Metrics Tracker:

- Based on flight sensor data.
- Offer insights into the aircraft's operational status.
- Ensure proactive risk mitigation.

API Documentation

- This documentation provides comprehensive details about the APIs involved in the Flight Route Planner web application, including user guides for pilots/control centers, technical documentation for development and support teams, and an Architecture Requirements Document (ARD) outlining the system architecture and design principles.

User Guide for Pilots / Control Center

1. Flight Plan Search

Endpoint:

GET / http://api.flightplandatabase.com/search/plans?fromICAO=VABB&toICAO=VIDP

Description

- This API endpoint searches for flight plans between two airports specified by their ICAO codes.

Parameters

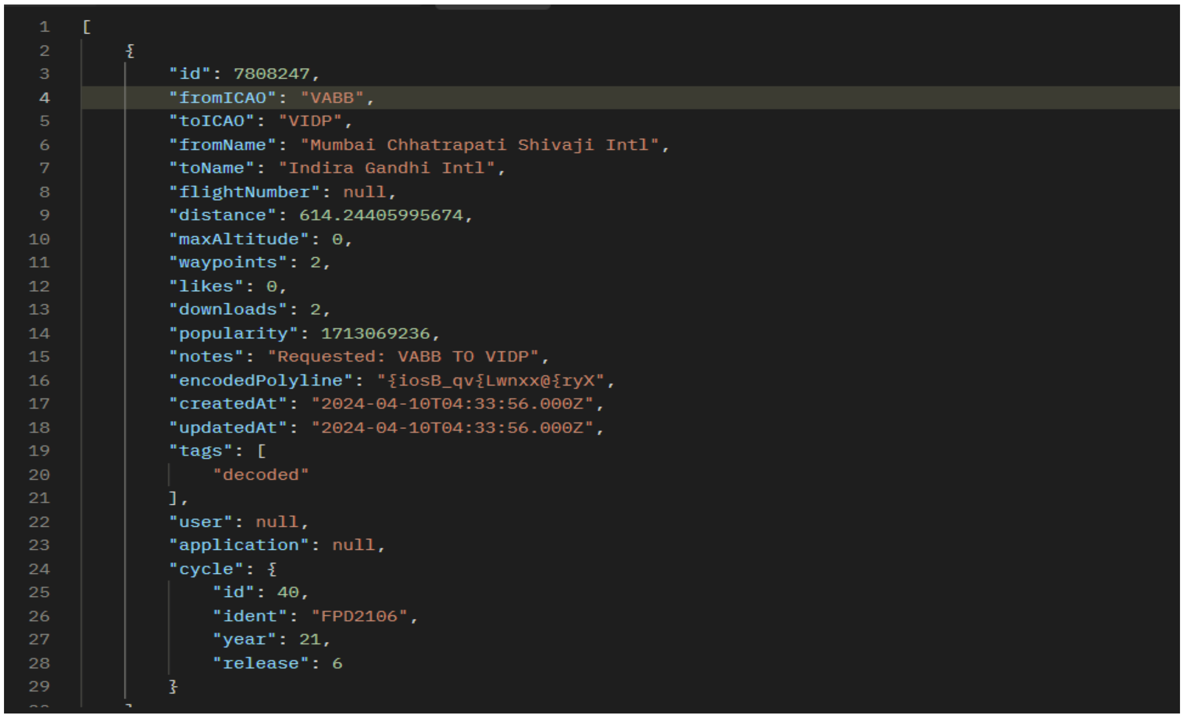
- **from ICAO** (required): The ICAO code of the departure airport.
- **to ICAO** (required): The ICAO code of the destination airport.

Response

The response is in XML format and typically includes:

- **planId:** Unique identifier for the flight plan.
- **route**: The route details.
- **distance**: Total distance of the flight.
- **duration**: Estimated flight duration.
- **waypoints**: List of waypoints in the flight plan.

Example :



```
[
    {
        "id": 7808247,
        "fromICAO": "VABB",
        "toICAO": "VIDP",
        "fromName": "Mumbai Chhatrapati Shivaji Intl",
        "toName": "Indira Gandhi Intl",
        "flightNumber": null,
        "distance": 614.24405995674,
        "maxAltitude": 0,
        "waypoints": 2,
        "likes": 0,
        "downloads": 2,
        "popularity": 1713069236,
        "notes": "Requested: VABB TO VIDP",
        "encodedPolyline": "{iosB_qv{Lwnxx@{ryX",
        "createdAt": "2024-04-10T04:33:56.000Z",
        "updatedAt": "2024-04-10T04:33:56.000Z",
        "tags": [
            "decoded"
        ],
        "user": null,
        "application": null,
        "cycle": {
            "id": 40,
            "ident": "FPD2106",
            "year": 21,
            "release": 6
        }
```

OpenWeatherAPI

Description

- This API endpoint provides current weather data for a specified location based on latitude and longitude.

Parameters

- **lat** (required): Latitude of the location (e.g., 35).
- **lon** (required): Longitude of the location (e.g., 139).
- **appid** (required): Your unique API key(e.g.,4c9c36bb42745e8c0226baf21bf2a3).

Example Request

GEThttps://api.openweathermap.org/data/2.5/weather?lat=35&lon=139&appid=4c9c36bb42745e8c0226baf21b583

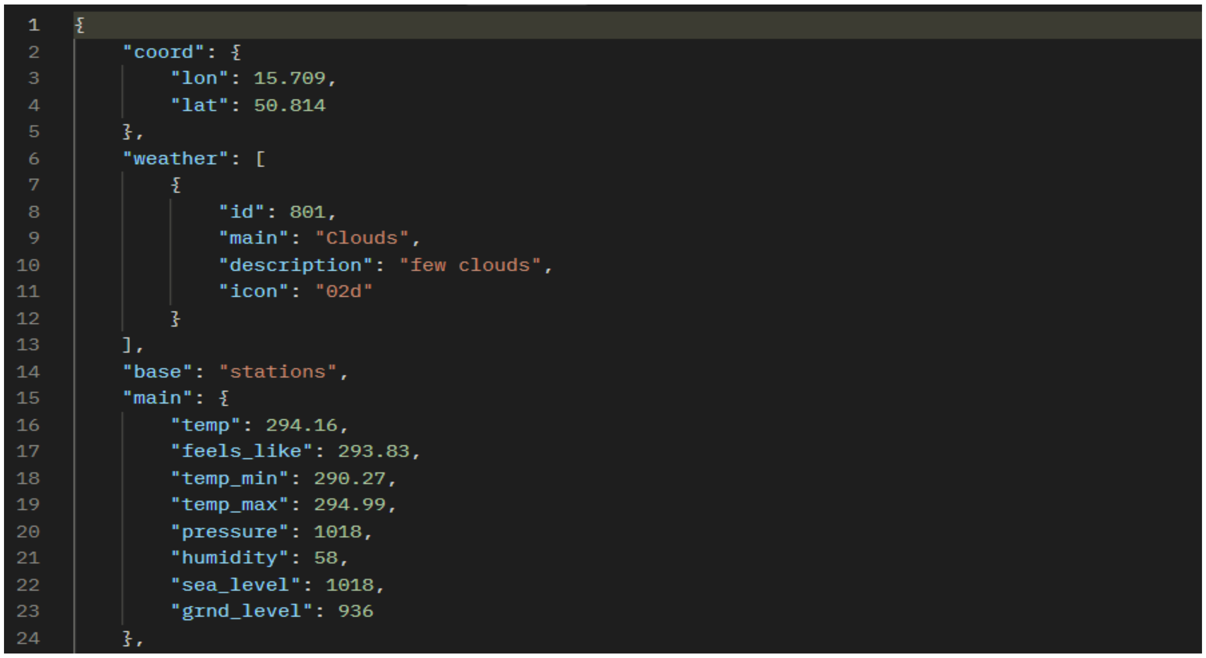
Response

The response is in JSON format and includes the following key elements:

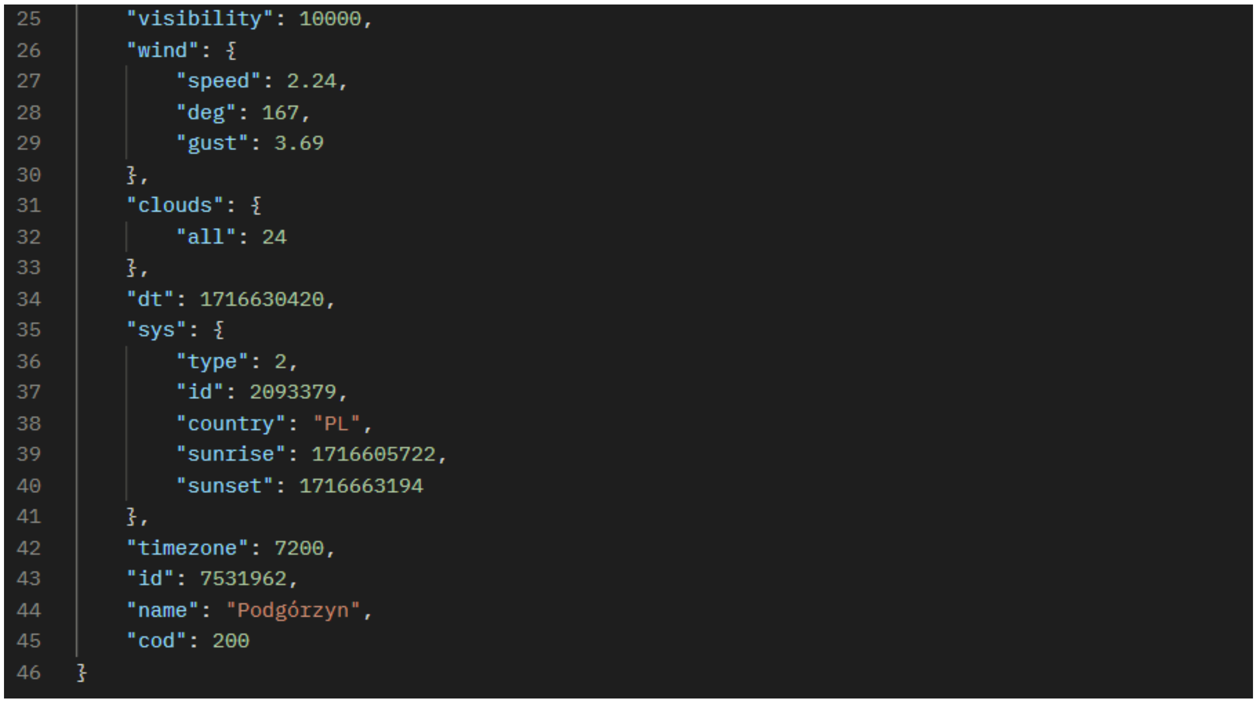
- **coord:** Coordinates of the location.
- **lon:** Longitude.
- **lat:** Latitude.
- **weather:** Weather conditions.
- **id:** Weather condition ID.
- **main:** Group of weather parameters (Rain, Snow, etc.).
- **description:** Weather condition description.
- **icon:** Weather icon ID.
- **base:** Internal parameter.
- **main:** Main weather data.
- **temp:** Temperature.
- **feels_like:** Perceived temperature.
- **temp_min:** Minimum temperature at the moment.
- **temp_max:** Maximum temperature at the moment.
- **pressure:** Atmospheric pressure.
- **humidity:** Humidity percentage.

- **visibility:** Visibility distance.
- **wind:** Wind data.
- **speed:** Wind speed.
- **deg:** Wind direction.
- **clouds:** Cloudiness percentage.
- **dt:** Time of data calculation (Unix timestamp).
- **sys:** System data.
- **type:** Internal parameter.
- **id:** Internal parameter.
- **country:** Country code.
- **sunrise:** Sunrise time (Unix timestamp).
- **sunset:** Sunset time (Unix timestamp).
- **timezone:** Shift in seconds from UTC.
- **id:** City ID.
- **name:** City name.
- **cod:** Internal parameter.

Example Response :



```json
{
    "coord": {
        "lon": 15.709,
        "lat": 50.814
    },
    "weather": [
        {
            "id": 801,
            "main": "Clouds",
            "description": "few clouds",
            "icon": "02d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 294.16,
        "feels_like": 293.83,
        "temp_min": 290.27,
        "temp_max": 294.99,
        "pressure": 1018,
        "humidity": 58,
        "sea_level": 1018,
        "grnd_level": 936
    },
```



```
    "visibility": 10000,
    "wind": {
        "speed": 2.24,
        "deg": 167,
        "gust": 3.69
    },
    "clouds": {
        "all": 24
    },
    "dt": 1716630420,
    "sys": {
        "type": 2,
        "id": 2093379,
        "country": "PL",
        "sunrise": 1716605722,
        "sunset": 1716663194
    },
    "timezone": 7200,
    "id": 7531962,
    "name": "Podgórzyn",
    "cod": 200
}
```

Flight Fuel Consumption :

Description

- This API endpoint calculates fuel requirements for a specified aircraft over a given distance.

Parameters

- **aircraft (required):** The unique identifier for the aircraft (e.g., 60006b).
- **distance (required):** The distance of the flight in nautical miles (e.g., 389).

Example Request

GET: https://despouy.ca/flight-fuel-api/q/?aircraft=60006b&distance=389

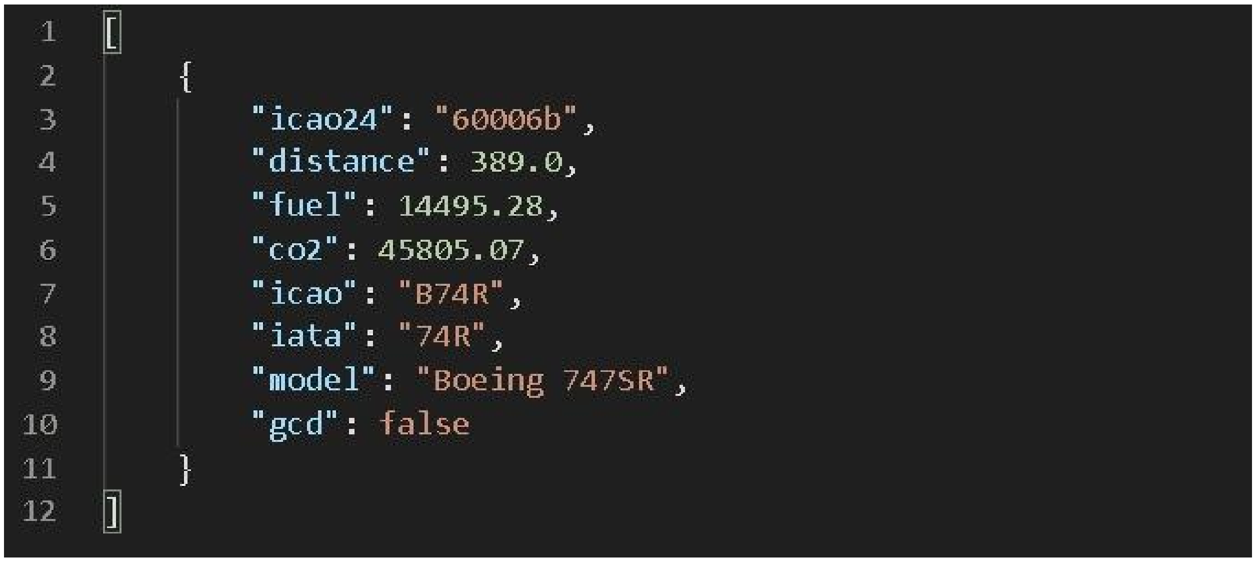
Response

The response is in JSON format and typically includes:

- **aircraft:** The identifier of the aircraft.
- **distance:** The distance of the flight.
- **fuel required:** The amount of fuel required for the flight.

- **units:** The units of the fuel required (e.g., liters, gallons).

Example Response :



```
[
    {
        "icao24": "60006b",
        "distance": 389.0,
        "fuel": 14495.28,
        "co2": 45805.07,
        "icao": "B74R",
        "iata": "74R",
        "model": "Boeing 747SR",
        "gcd": false
    }
]
```

Data Flow

Flight Plan Search:

- **User Input:** The user provides input such as departure and destination airports.
- **Frontend Validation:** The front end validates the user input to ensure it meets the required format and criteria.
- **API Request:** The validated user input is sent as parameters in an API request to the flight plan search endpoint.
- **Backend Processing:** The backend receives the API request and processes it by querying the flight plan database or external flight planning APIs.
- **Response Handling:** The backend receives the response from the flight plan database or external API, and formats it appropriately.
- **Data Presentation:** The formatted flight plan data is sent back to the front end for presentation to the user.

Weather Data Retrieval:

- **User Input:** The user provides the location for which weather data is required.

- **Frontend Validation:** Similar to flight plan search, the frontend validates the user input for the location.
- **API Request:** The validated location data is sent as parameters in an API request to the weather data retrieval endpoint.
- **Backend Processing:** The backend receives the API request and processes it by querying the weather database or external weather APIs.
- **Response Handling:** The backend receives the response from the weather database or external API, and formats it appropriately.
- **Data Presentation:** The formatted weather data is sent back to the frontend for presentation to the user.

Route Visualization:

- **Flight Plan Selection:** The user selects a specific flight plan from the available options.
- **Frontend Processing:** The frontend processes the selected flight plan data and extracts relevant route information.
- **Map Integration:** The front end integrates with mapping libraries or APIs (e.g., Google Maps) to display the route visually.
- **Route Rendering:** The front end renders the flight route on the map, including waypoints and other relevant details.
- **User Interaction:** The user can interact with the route visualization, such as zooming, panning, or viewing additional information about waypoints.
- **Real-Time Updates (Optional):** If supported, the frontend may update the route visualization in real time based on changes in flight status or weather conditions.

Architecture Requirements Document (ARD)

1. Introduction

Purpose:

The purpose of this document is to outline the architecture and design principles of the Flight Route Planner web application.

Scope:

The scope includes a detailed description of the system architecture, API interactions, data flow, and design principles necessary to build and maintain the Flight Route Planner application.

2. Architecture Components

Client-side :

React: Think of React as a toolbox for building parts of a webpage. It helps in creating interactive and dynamic user interfaces.

Components: These are like building blocks of a webpage. Each component does a specific job, like handling user input or displaying information.

- **FlightDetail.js:** Displays detailed information about a selected flight, such as times, airline, and other specifics.
- **FlightPathMap.js:** Shows the flight route on a map, helping users visualize the journey.
- **FlightSelection.js:** Allows users to choose a flight from a list of available options, handling user input and displaying flight choices.
- **FlightWeatherFaults.js:** Shows weather-related issues or alerts for a selected flight, informing users about potential delays or hazards.
- **FuelData.js:** Provides information about the fuel requirements for a flight, such as the amount needed.

Server-side :

- **Node.js:** Imagine Node.js as the engine that runs the backend of a website. It allows us to use JavaScript to handle server-side tasks.

APIs: These are like special assistants that help our website get information from other places on the internet.

- **FlightPlanDatabase API:** This API helps us get details about flight plans, like routes and waypoints.
- **Open Weather API:** This API provides weather information for specific locations, helping us show weather forecasts on our website.

3. Design Principles

- **Modularity:** The system is like a collection of Lego blocks, where each block (or component) has a specific job. This makes it easy to change or add new parts without affecting the whole system.
- **Scalability:** Think of the system as a playground that can grow as more kids join. It can handle more users or data without slowing down or breaking.
- **Reliability:** Just like a sturdy bridge, the system is built to stay strong even if something goes wrong. It keeps working even if there are bumps along the way.
- **Security:** Imagine the system as a locked treasure chest. Data traveling between users and the system is kept safe from prying eyes, like a secret code that only the sender and receiver understand.
- **Performance:** Picture the system as a fast car on a highway. It's designed to be quick and efficient, delivering information without making users wait too long.

4. System Requirements

Frontend :

- React.js
- Map visualization library (e.g., Leaflet, Mapbox)

Backend:

- Node.js

External APIs :

- FlightPlanDatabase API
- Open Weather API
- Flight Fuel Consumption API

5. Deployment and Maintenance

Deployment:

- Use CI/CD pipelines for automated testing and deployment.
- Deploy on a scalable cloud platform Vercel.
- Deployed Link : https://airbus-challenge-fronted.vercel.app/

Maintenance:

- Regularly update API keys and endpoints.
- Monitor application performance and error logs.
- Implement automated backups for critical data.

This comprehensive documentation aims to provide clarity on the system architecture, API interactions, and user guide to ensure efficient implementation and usage of the Flight Route Planner web application.