Planesoptter Helper

an all-in-one information website for plane spotters

This is the **report/document** for the individual project of the course ***Microservice Architecture***. As required, the report is structured to include ***Project Overview(1.)***, ***Application Features & Functions(2.)***, ***Development Tools & Frameworks & Libraries(3.)***, ***Component Functions & Implementation(4.)*** , ***Deployment & Configuration(5.)***, and ***API Ref & Schema(6.)***.

Do note that this report/document contains **everything about the project except the project source code itself**. No other files will be given separately for ***Configuration and deployment description***, ***JSON Schema, database script, and XML Schema***, or ***the documents of the Web APIs***.

1. Project Overview

PlaneSpotter Helper is a web-based application specifically designed for aviation enthusiasts known as plane spotters. The primary goal of this project is to provide users with a comprehensive, one-stop platform that delivers detailed information about airports and flights based on the selected airport. This tool aims to enhance the experience of plane spotters by centralizing relevant data into a single, easily accessible location.

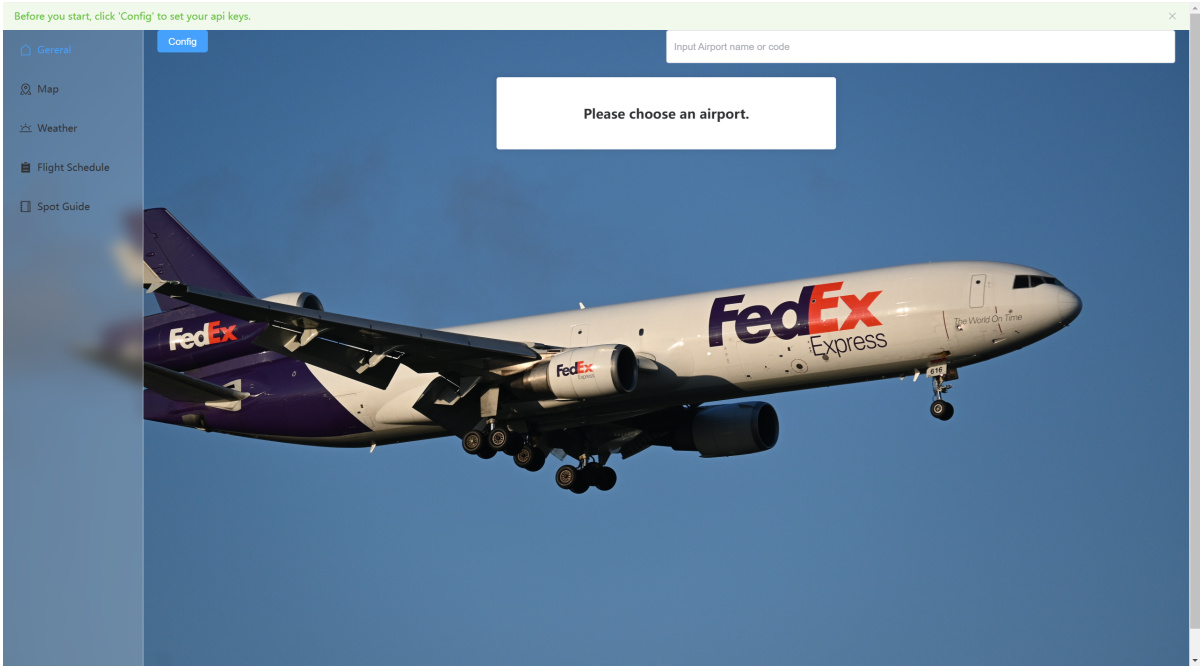
Plane spotting often requires enthusiasts to have up-to-date knowledge of flight schedules, aircraft types, weather conditions, and airport logistics. This project addresses these needs by aggregating data from various sources to display real-time flight information, airport details, and other related content. Users can explore specifics such as arrival and departure times, delays, gate information, aircraft models, and even weather forecasts that might affect plane spotting.

The goal of **PlaneSpotter Helper** is to streamline the information-gathering process, allowing users to quickly access accurate and comprehensive data about their chosen airports and flights. By centralizing this information, the project aims to save users time and effort, eliminating the need to visit multiple websites or apps to gather the necessary data for their spotting activities.

This application is particularly useful for hobbyists who enjoy tracking and photographing aircraft, providing them with the tools they need to plan their visits to airports effectively. Additionally, it serves as a valuable resource for individuals interested in aviation logistics, air traffic patterns, and the operational aspects of airports. The platform's user-friendly design ensures that both beginners and experienced plane spotters can efficiently utilize its features to enhance their plane-watching experience.

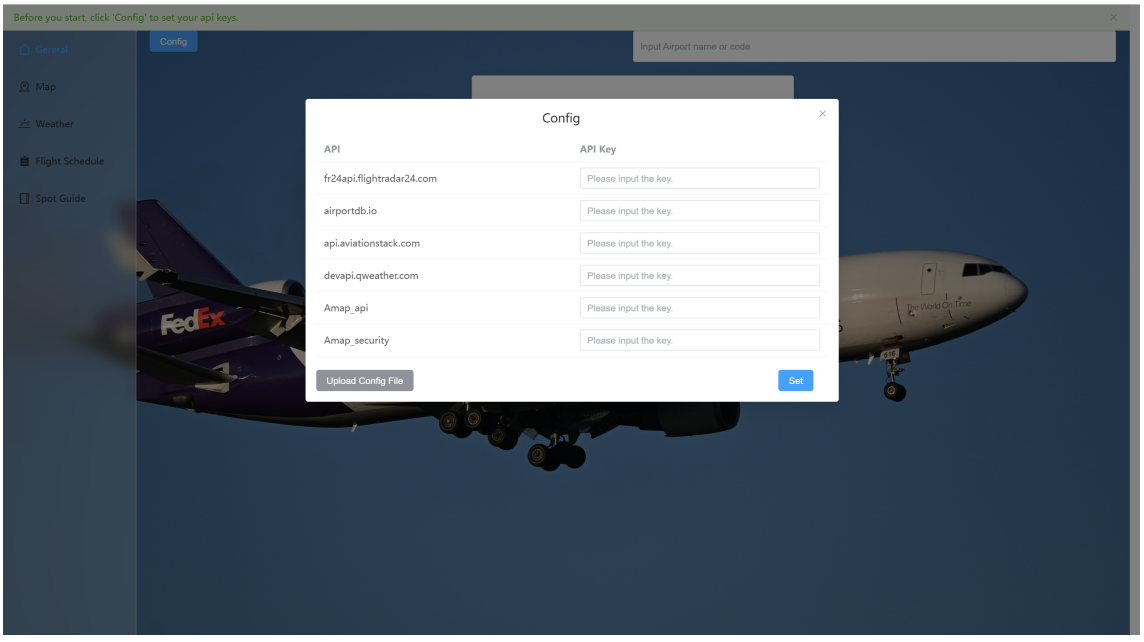
2. Application Features & Functions

`PlaneSpotter Helper` offers several key features: Airport Autocomplete, Flight Information Display, Weather Forecast and Sunrise/Sunset Times, Real-time Flight Tracking, Airport Spotting Guide, Detailed Airport Information, and Airport Satellite and Radar Map Display.



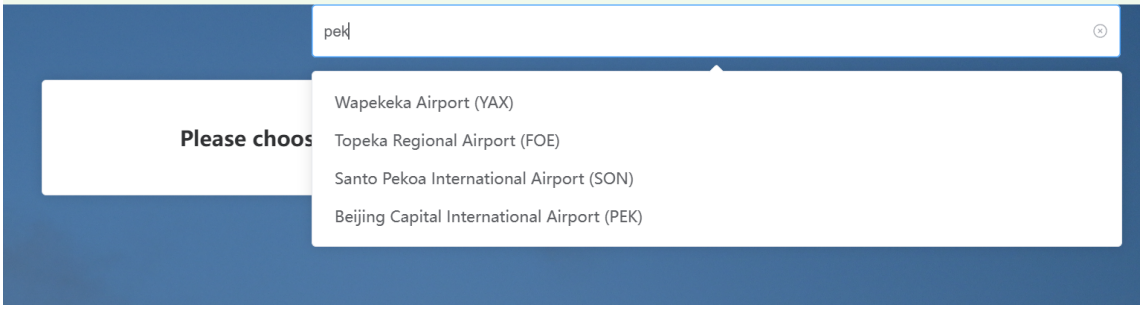
- **Easy API Setup**

  Users will be instructed to set the API keys needed to use this website when accessing. The setup process will be easy with both manual input and config file upload options.



- **Airport Autocomplete**

  Users can type an airport name or IATA code in the search box, and the system will provide autocomplete suggestions. This allows users to quickly select an airport and load relevant information associated with that airport.

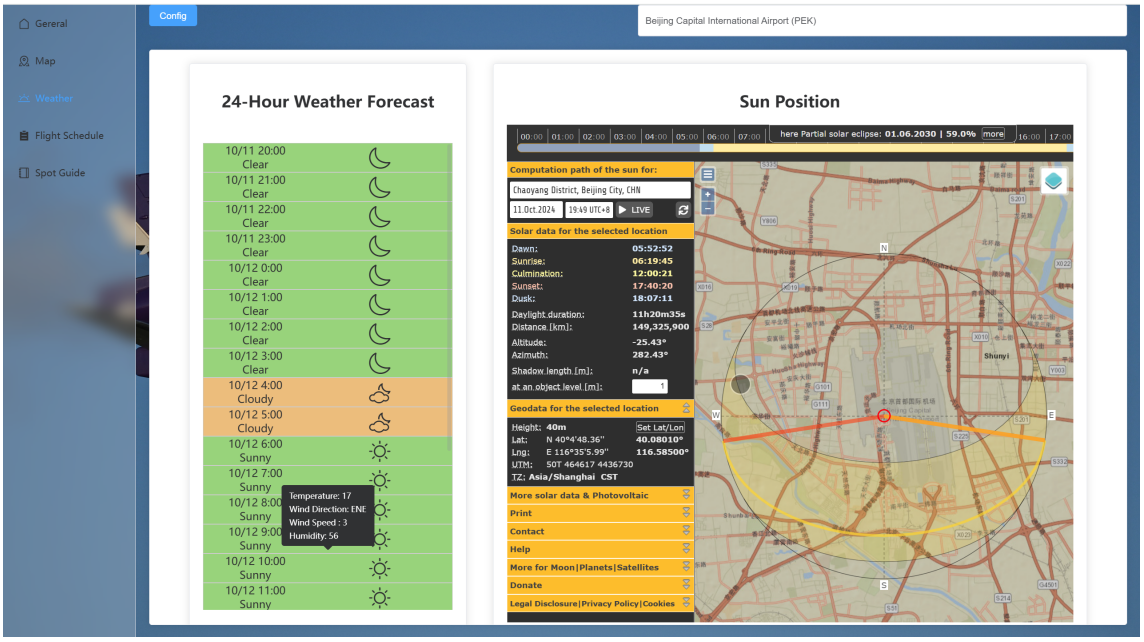


- **Integrated Flight Information Display**

  Displays the list of outbound and inbound flights for the selected airport, including details such as flight number, destination, and estimated arrival/departure times. Users can filter the flight list based on the flight code or/and plane types to quickly find flights of interest. Also, **the traditional flight details are also coupled with the hourly weather data presented in color** to help plane spotters to evaluate the sightseeing condition quickly.

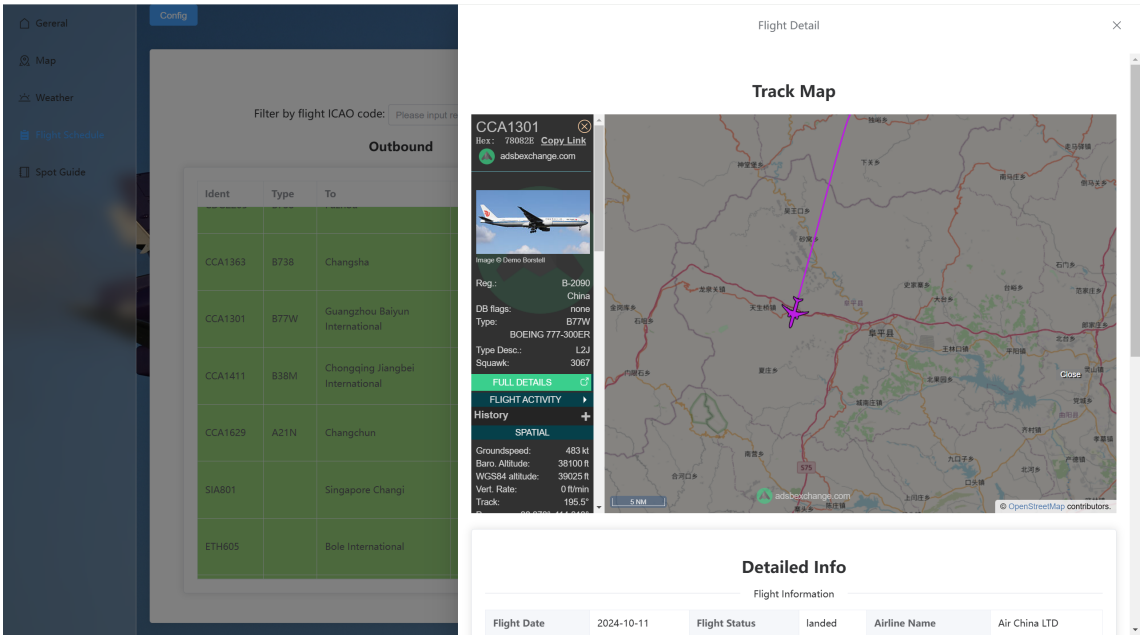
- **Weather Forecast and Sunrise/Sunset Times**

  Shows real-time weather conditions for the selected airport by hour, including temperature, humidity, and wind speed, along with sunrise and sunset times to help users assess lighting conditions and environmental factors.



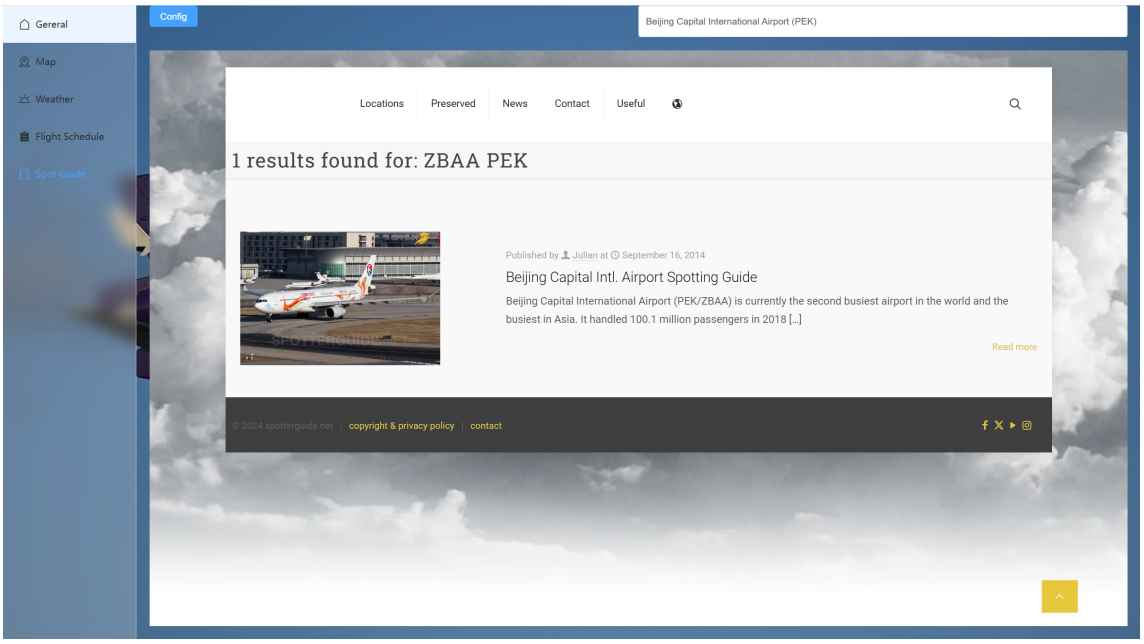
- **Real-time Flight Tracking**

  Provides real-time tracking of flight paths and positions around the airport using radar maps, allowing users to visually monitor aircraft movements and current airspace activity.



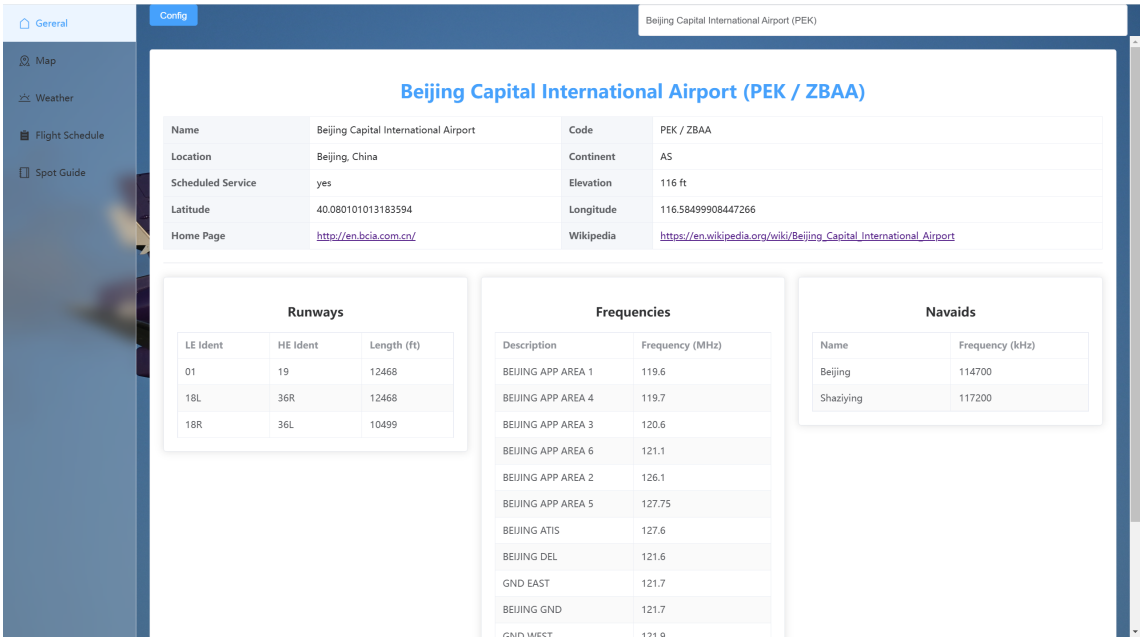
- **Airport Spotting Guide**

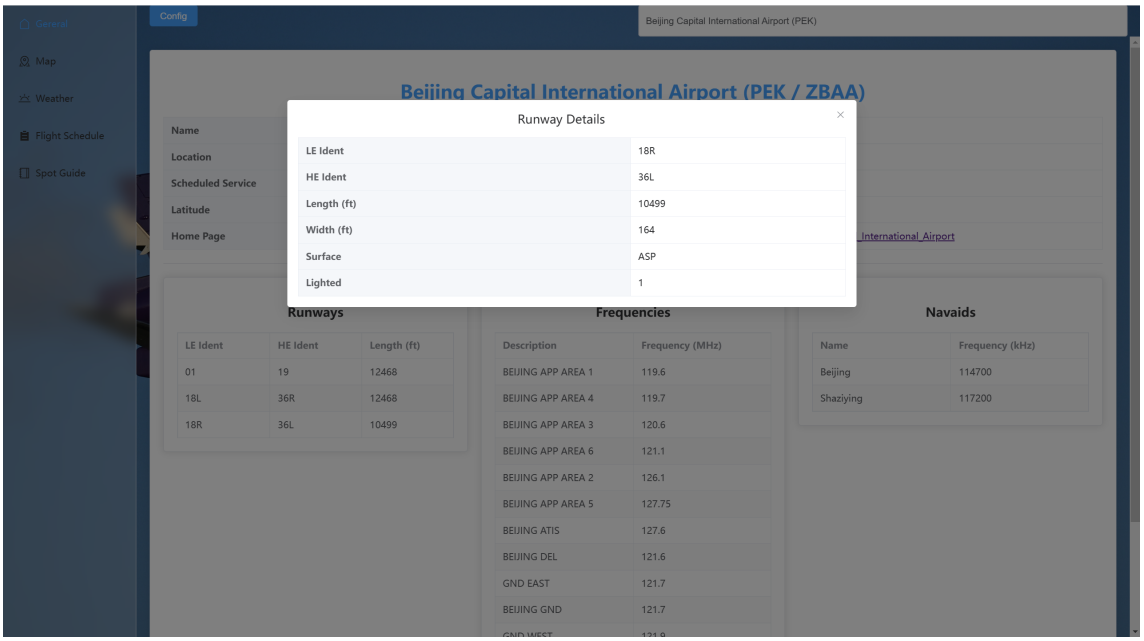
  Offers detailed information on the best spotting locations around the airport, helping plane spotters find ideal positions for viewing and photographing aircraft, with insights into each spot's features and accessibility.



- **Detailed Airport Information**

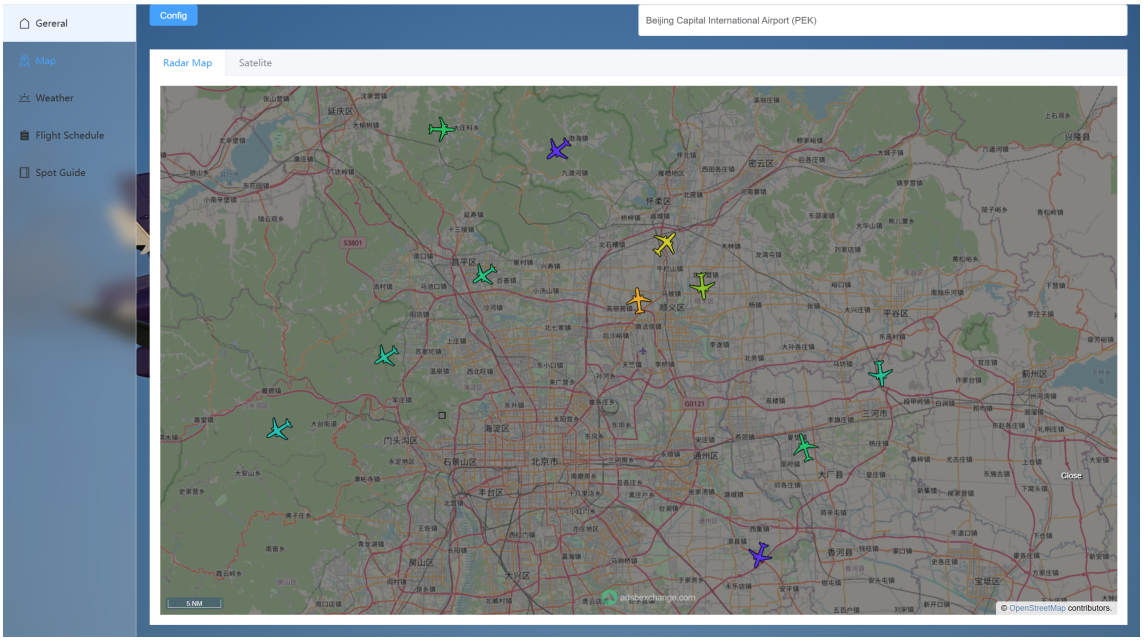
  Displays comprehensive details about the airport, including its name, IATA/ICAO codes, location, facilities, and operational data. Users can gain a full understanding of the airport's basic and advanced features.

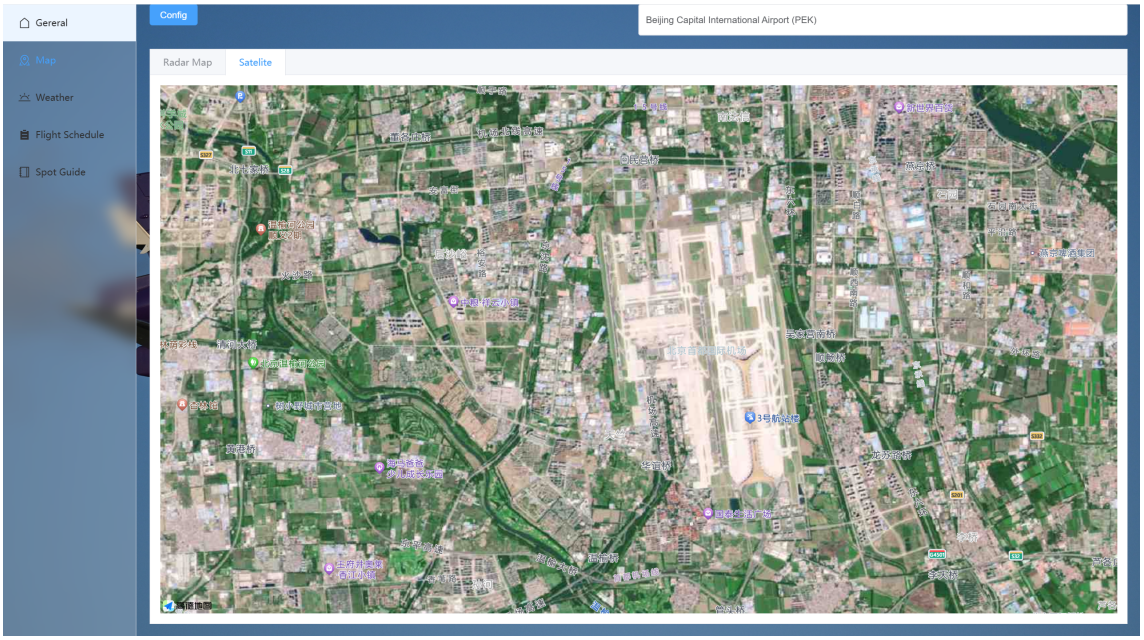




- **Airport Satellite and Radar Map Display**

  Provides high-resolution satellite imagery of the airport and its surrounding areas, showcasing detailed terrain and infrastructure. The radar map displays real-time and flight activity near the airport, enabling users to monitor airspace operations effectively.





This application provides comprehensive support and convenience to plane spotters and travelers by integrating detailed airport and flight information, enhancing the user experience with interactive and efficient airport data visualization.

3. Development Tools & Frameworks & Libraries

In this project, a variety of modern front-end frameworks, libraries, and tools are utilized to enhance development efficiency, improve user experience, and provide comprehensive functionality. Below is a detailed description of the main technologies used in the project:

1. **Core Framework**
    - **Vue 3**: The project uses Vue 3 as its core framework. Vue 3 is a progressive JavaScript framework known for its reactive data binding and component-based architecture, making it ideal for building user interfaces and single-page applications (SPA).
2. **UI Component Library**
    - **Element Plus**: Element Plus is an advanced UI component library based on Vue 3. It offers a rich set of components (e.g., forms, buttons, dialogs) that help developers quickly create consistent and visually appealing user interfaces.
    - **Element Plus Icons**: This library provides a diverse collection of icons, integrated with Element Plus, to enhance visual representation in the application.
3. **Data Fetching and API Communication**
    - **Axios**: Axios is used for making API requests and fetching data. It is a Promise-based HTTP client that interacts with backend services to retrieve dynamic data like weather conditions and flight information.
4. **Map Integration**
    - **Vue AMap**: The project integrates the Vue AMap library to support the AMap (Gaode Map) API. It enables embedding interactive maps within the application, providing geolocation and navigation services, with secure API key management using the `api_keys.json` file.

5. **Other Functional Libraries**
    - **SunCalc**: SunCalc is used to calculate the position of the sun and related parameters, such as azimuth and altitude angles. This library helps in displaying weather and time-related data.
    - **QWeather Icons**: This library provides weather icons, offering a visually intuitive way to display weather-related data in the application.
6. **Project Build and Dependency Management**
    - **Vite**: As a build tool, it is commonly employed in Vue projects to speed up the development process and optimize the bundle size in production.

4. Component Functions & Implementation

(1) Home.vue

Description:

`Home.vue` is the main page of the application, organized with a container layout that includes a sidebar, a top navigation bar, and the main content display area. When no airport is selected, the main content area shows a hint card prompting the user to choose an airport. Once an airport is selected via the `TopBar` component, the page dynamically displays the detailed information of the selected airport. The overall design of the homepage is intended to guide the user to select an airport first before proceeding to view related information.

Implementation:

For implementation, `Home.vue` utilizes Vue's Composition API along with the `Element Plus` UI library to organize the page structure. The `Menu` component in the sidebar allows users to click and select different pages. The top `TopBar` component supports airport name autocomplete, and when an airport is selected, it triggers the `airportSelect` method, passing the airport code to the `AirportInfo` component. The main content area uses the value of `airport_code` to decide what content to display. If no airport is selected, it displays a hint card that says, "Please choose an airport," ensuring that no airport-related information is shown until an airport is selected.

(2) TopBar.vue

Element Position:

`TopBar.vue` is located at the top section of the webpage, forming part of the global navigation in the `el-header` section of the page layout. It directs the user to input airport names and assists with airport selection through an autocomplete feature.

Description:

The primary function of the `TopBar.vue` component is to provide airport name autocomplete and selection feature and API key setup.
In `TopBar.Vue,` Search.vue `and` Config.vue `are used as the core child components.` TopBar.vue `mainly handles the overall layout and event management, while the specific airport search functionality is implemented within` Search.vue `, and the API setup is implemented in` Config.vue `.

When the user inputs an airport name or code in `Search.vue`, it performs the autocomplete feature using data from the local `airports.json` file and displays the matching results to the user. After the user selects an airport in `Search.vue`, it triggers a selection event that passes the selected airport data back to `TopBar.vue`, which then forwards this data to the parent component `Home.vue` to update the airport details displayed on the page.

`Config.vue` implements a popup display that allows users to provide their API keys. The setup process will be easy with both manual input and config file upload options, meaning it can be done by either typing in the keys one by one or by uploading a config file to fill the fields in one step.

Implementation:

`TopBar.vue` utilizes the `Search.vue` child component to handle user input, using Vue's `el-autocomplete` component for the autocomplete functionality. The data comes from a local `airports.json` file, and when the user types into the input box, `Search.vue` automatically invokes the `filterAirports` method to filter the list of airports in real time. Once the user selects an airport, the `selectAirport` method passes the selected airport information back to the parent component to update the page display.

```
const filterAirports = (search, cb) => {
  const query = search.toLowerCase();
  const results = airportsData.filter(
    airport =>
      airport.name.toLowerCase().includes(query) ||
      airport.iata_code.toLowerCase().includes(query)
  ).map(
    airport => {
      airport.value = `${airport.name} (${airport.iata_code})`;
      return airport;
    }
  )
  console.log("Results:", results);
  cb(results);
};
```

(3) Menu.vue

Element Position:

The `Menu.vue` component is positioned on the left sidebar of the page and serves as the core navigation element of the application, guiding users through different informational pages. It provides menu items that enable users to quickly access and switch to the desired feature pages.

Description:

The primary responsibility of `Menu.vue` is to provide navigational functionality within the user interface. It utilizes the `el-aside` component to render a vertical menu containing multiple menu items, each of which maps to a specific page or feature module. When a user clicks on a menu item, the component triggers the page-switching logic, updating the user interface to display content corresponding to the selected menu item. `Menu.vue` dynamically changes the current selection by listening to the user click events and passes the selected page information to `Home.vue` to update the central display area. We deployed a frosted glass texture to enhance its appearance.

Implementation:

`Menu.vue` uses the `select_page` method to respond to user interactions with the menu, transmitting the identifier of the selected menu item to the parent component `Home.vue`. This allows `Home.vue` to update the main display area based on the menu selection. The design follows a unidirectional data flow approach, ensuring efficient and controlled data transmission between components.

```
const emit = defineEmits(['select-page']);
const select_page = (v) => {
    emit('select-page', v);
};
```

(4) AirportInfo.vue

Element Position:

`AirportInfo.vue` is located in the main content area of the page, serving as the central hub for displaying information. This component aggregates and presents detailed information related to the airport selected by the user, relying on inputs from `Menu.vue` and `TopBar.vue` to dynamically update its content.

Description:

The primary function of `AirportInfo.vue` is to dynamically organize and present information related to the airport based on user interactions. It integrates multiple sub-components, such as flight data, weather conditions, and map displays, which update their content according to the selected airport information. The design of this component allows it to switch displayed content as the user navigates through different pages using

`Menu.vue`, such as toggling between flight details, weather information, or airport guides. This approach provides users with a comprehensive interface for viewing detailed airport information.

Implementation:

In terms of implementation, `AirportInfo.vue` leverages Vue's dynamic component loading and conditional rendering techniques to determine the displayed content based on the `airport_code` and `current_page` parameters passed from the parent component. Each sub-component (such as `FlightData.vue`, `WeatherSun.vue`, etc.) uses the Composition API to receive airport data and dynamically update its presentation according to user selections. The data flow between components follows a unidirectional design, with information passing from `TopBar.vue` and `Menu.vue` to `AirportInfo.vue`, which then coordinates the update logic of its sub-components, ensuring data consistency and control.

(4.1) FlightData.vue

Key Functions & Features:

- **Flight Information Display**: Clearly shows all outbound and inbound flights of the airport, including flight number, destination/origin, and estimated arrival/departure times.
- **Flight Filtering**: Users can filter the flight list by entering flight numbers or aircraft types to quickly find flights of interest.
- **Dynamic Color Indication**: Flight list rows change colors dynamically based on weather conditions, helping users quickly determine if the weather will be good to do plane spotting **for that specific flight**.
- **Flight Detail Popup**: Clicking on any flight brings up a detail window showing real-time flight position(if available), flight path, and weather details.

Description:

The primary function of `FlightData.vue` is to show all the information related to the flight coming or leaving the specific airport. It collects this information, combining it with weather information and flight tracking information to help users know what to expect. It presents outbound and inbound flight information in two tables, displaying details such as flight number, aircraft type, destination/origin, registration number, and departure/arrival times. It allows users to filter flights based on regular expressions for flight number and aircraft type, making it easier to pinpoint specific flights. The component dynamically sets the background color of table rows based on weather conditions related to the flight's scheduled time, giving users a visual indication of how the weather will be for the plane spotting of that specific flight.

We are trying to make the page as simple and understandable as possible, but to know more about a specific flight, users can click on a flight row and triggers a drawer-style popup displaying detailed information about the flight, including real-time flight tracking and weather conditions.

Implementation:

The implementation of `FlightData.vue` focuses on a seamless user experience and dynamic data display:

1. **Flight Data Retrieval**: It uses the Axios library to fetch real-time flight data from the AviationStack API, calling the `fetchFlights` method when the component is mounted. For development and testing purposes, the component also supports loading mock data from local files (`arrival.json` and `departure.json`).

```
// Method: Get flight info
const fetchFlights = async () => {

  if (props.useDemoData) {
    outboundFlights.value = demo_response_dep.data;
    inboundFlights.value = demo_response_arr.data;
    return;
  }

  try {
    const response_arr = await
axios.get(`http://api.aviationstack.com/v1/flights`, {
      params: {
        access_key: key,
        arr_icao: props.icaoCode,
      },
    });

    const response_dep = await
axios.get(`http://api.aviationstack.com/v1/flights`, {
      params: {
        access_key: key,
        dep_icao: props.icaoCode,
      },
    });

    outboundFlights.value = response_dep.data.data;
    inboundFlights.value = response_arr.data.data;
  } catch (error) {
    console.error('Flight info fetch fail:', error);
  }
};
```

2. **Dynamic Row Styling**: The component applies dynamic background colors to table rows through the `rowStyle` method, which is determined by weather conditions calculated via the `chooseWeatherByHour` method. This color coding provides users with a visual cue about potential weather impacts on flights.

```
const chooseWeatherByHour = (time) => {
  if (props.useDemoData) {
    var now = new Date();
    now.setHours(now.getHours() + Math.floor(Math.random() * (22 - 1 + 1) +
1));
    var t = timeByHour(now);
    var re = hourlyWeather.value[t];
    return re;
  }
  return hourlyWeather.value[timeByHour(time)];
};
```

3. **Filtering Functionality**: The filtering feature utilizes regular expressions to refine the flight list based on user input for flight numbers and aircraft types. The filtered flight information is immediately reflected in the displayed tables.

```
// Filter flight, get/choose weather and time.
const filteredOutboundFlights = computed(() => {
  return outboundFlights.value.filter((flight) => {
    const flightMatch = flightNumberRegex.value
      ? (flight.flight && new RegExp(flightNumberRegex.value,
'i').test(flight.flight.icao))
      : true;
    const aircraftMatch = aircraftTypeRegex.value
      ? (flight.aircraft && new RegExp(aircraftTypeRegex.value,
'i').test(flight.aircraft.icao))
      : true;
    return flightMatch && aircraftMatch;
  }).map((v) => {
    v.display_time = chooseTime(v.departure, true);
    v.related_local_time = chooseTime(v.departure, false);
    v.weather = chooseWeatherByHour(v.related_local_time);
    return v;
  })
    ;
});

const filteredInboundFlights = computed(() => {
  return inboundFlights.value.filter((flight) => {
    const flightMatch = flightNumberRegex.value
      ? (flight.flight && new RegExp(flightNumberRegex.value,
'i').test(flight.flight.icao))
      : true;
    const aircraftMatch = aircraftTypeRegex.value
      ? (flight.aircraft && new RegExp(aircraftTypeRegex.value,
'i').test(flight.aircraft.icao))
      : true;
    return flightMatch && aircraftMatch;
  }).map((v) => {
    v.display_time = chooseTime(v.arrival, true);
    v.related_local_time = chooseTime(v.arrival, false);
```

```
    v.weather = chooseWeatherByHour(v.related_local_time);
    return v;
  });
});
```

4. **Click-to-Detail Drawer Popup**: When a user clicks on any flight row, the component invokes the `display_flight` method, opening a drawer-style popup to show comprehensive flight details, including the flight path and associated weather conditions. This is facilitated by the `FlightDetail.vue` sub-component, ensuring a robust and interactive data presentation.

Data Source:

The data sources used by `FlightData.vue` are fetched using **AviationStack API**, which provides lists of detailed information on flights related to the specific airport.

(4.2) WeatherSun.vue

Key Functions & Features:

- **Real-time Weather Information**: Shows up-to-date weather conditions at the airport, including temperature, humidity, and wind speed to help users evaluate travel conditions.
- **Sun Movement Display**: Accurately displays sunrise and sunset times at the airport location, as well as the sun movement track.
- **Weather Icons Display**: Display icons based on current weather conditions, making changes in the weather easily noticeable.
- **Detailed Weather View**: Users can hover or click on icons to see more detailed weather forecasts.

Description:

The primary functions of `WeatherSun.vue` include:

1. **Real-time Weather Display**: Using the `Weather.Vue child component, it displays real-time weather conditions based on the airport's geographic coordinates. This component handles the retrieval and presentation of meteorological data such as temperature, humidity, and wind speed.
2. **Sunrise and Sunset Times**: It employs the `SunInfo.Vue child component to refer to SunCalc.org with the right parameters including time and location. This allows users to predict sun movement near the airport with little effort.
3. **Data Communication**: The component uses the `send_weather` method to transmit the obtained weather data to the parent component or other relevant components, ensuring that weather information is correctly updated and utilized throughout the application.

Implementation:

In its implementation, `WeatherSun.vue` relies on coordinate information passed via `props` to initialize and update weather data. Through the `Weather.Vue child component, it fetches data from a weather service API based on geographic coordinates or uses mock data when in demo mode.` SunInfo.vue `uses the same coordinate data to refer to suncalc.org using ***iframe***. The component also defines a` send-weather` event to broadcast the latest weather information to other components whenever the data is updated.

(4.3) Map.vue

Key Functions & Features:

- **Real-time Flight Tracking**: Displays radar maps around the airport to track flight paths and positions in real-time, helping users understand air traffic conditions.
- **High-resolution Satellite Imagery**: Offers detailed satellite views of the airport and surrounding areas, allowing users to zoom in on the terrain and infrastructure.
- **Multi-view Switching**: Users can easily toggle between "Radar Map" and "Satellite Map" to access different perspectives of the airport.

Description:

The primary functions of `Map.vue` include **Radar Map Display** and **Satellite Imagery Display**. It uses the `RadarMap.vue` child component to show real-time radar imagery around the airport, allowing users to track current flight movements and weather conditions in the airspace. It utilizes the `SatelliteMap.vue` child component to render satellite images of the airport, giving users a detailed view of the airport's terrain and infrastructure. It provides a user-friendly tab-based interface that allows switching between "Radar Map" and "Satellite" views for quick access to different map types.

Implementation:

In its implementation, `Map.vue` uses Vue's `el-tabs` component to structure the different map displays, utilizing `RadarMap.vue` and `SatelliteMap.vue` to load radar and satellite images, respectively. The component dynamically sets the map's center position based on the provided airport information, ensuring that the displayed map content corresponds accurately to the selected airport.

RadarMap.vue

`RadarMap.vue` utilizes **ADS-B Exchange** by embedding an ***iframe*** to display the radar map. It dynamically generates the URL based on the airport's ICAO code, ensuring the displayed content aligns with the selected airport.

SatelliteMap.vue

`SatelliteMap.vue` uses **Amap** by rendering satellite images through the `el-amap` component and the `el-amap-layer-satellite` layer. The component dynamically sets the map's center based on the provided coordinates, ensuring the satellite imagery precisely corresponds to the selected airport's location.

Data Source:

The data sources used by `Map.vue` are fetched from **ADS-B Exchange** and **Amap**. We don't use the traditional REST API for the reason that they don't recommend us doing that.

(4.4) SpotGuide.vue

Key Functions & Features:

Airport Spotting Guide: Provides information on the best spotting locations around the airport to help plane spotters choose ideal viewing and photography spots.

Description:

The primary function of `SpotGuide.vue` is to provide information on airport spotting locations by embedding an ***iframe*** from Spotterguide, displaying the best viewing spots associated with the selected airport.

Implementation:

In its implementation, `SpotGuide.vue` uses the `getUrl` method to generate a Spotterguide URL based on the provided airport object, formatted as `https://www.spotterguide.net/?s=ICAO_CODE+IATA_CODE`. When the user selects a different airport, the embedded ***iframe*** automatically updates to display the spotting guide for that airport.

```
<template>
  <div class="flex-container">
    <iframe :src="getUrl(airport)" width="100%" height="100%"
frameborder="0" allowfullscreen
      sandbox="allow-same-origin allow-forms allow-scripts"></iframe>
  </div>
</template>

<script setup>
import { ref, onMounted, watch } from 'vue';

const props = defineProps({
  airport: {
    type: Object,
    required: true,
  },
});
```

```
const getUrl = (airport) => {
  return `https://www.spotterguide.net/?s=${airport.icao}+${airport.iata}`;
};

</script>


<style>
.flex-container {
  display: flex;
  width: 100%;
  height: 100%;
}

.flex-container iframe {
  flex: 1;
  border: none;
}
</style>
```

Data Source:

The data sources used by `Map.vue` are fetched from **SpotterGuide**. We don't use the traditional REST API because it does not exist.

(4.5) AirportDetails.vue

Key Functions & Features:

- **Basic Airport Information**: Displays essential airport details like IATA/ICAO codes, name, location, and operating hours, giving users a quick overview of the airport's core data.
- **Facility Information**: Includes detailed descriptions of airport facilities and services such as terminals, runways, and lounges, helping users plan their visit.

Description:

The primary functions of `AirportDetails.vue` include:

1. **Basic Airport Information Display**: It shows fundamental data about the airport, including its IATA and ICAO codes, name, and location.
2. **Fetching Additional Information**: Retrieves extended information about the airport via external API calls and presents it in a structured format to the user.
3. **Error Handling and User Feedback**: Displays error messages in case of data retrieval failures, helping users understand the cause of the issue.

Implementation:

`AirportDetails.vue` fetches airport data by calling APIs from Flightradar24 and AirportDB. Initially, it retrieves basic information using the airport code from Flightradar24, followed by a request to AirportDB for extended details like geographical location and facility information. The `fetchAirportInfo` method is designed to execute automatically when the component is mounted and to be re-invoked upon data updates. The retrieved data is stored in the reactive variable `airportInfo` and displayed using the `DetailTable.vue` child component in a tabular format.

Data Source:

The data sources used by `AirportDetails.vue` include:

1. **Flightradar24 API**: Supplies basic airport information, such as IATA and ICAO codes, airport name, and current flight count.
2. **AirportDB API**: Provides detailed extended information, including the airport's geographic coordinates, facility details, and other supplementary data.

5. Deployment & Configuration

As a mashup website based on the available backend service from other providers, this project is designed to be a purely static web page that requires **zero** backend support from its developers. This is designed on purpose so it can be easily deployed on many cheap or even free ***website hosting platforms***. For now, we use vercel.com to host our website, which is located in a private Github repository, and let the platform do the rest. The website domain is https://planesoptter-helper.vercel.app/.

6. API Ref & Schema

The following will include all the services the project needs. It is sorted by the service provider. For each service/API used, the following information will be provided.

- The official website of the service provider, linked to the name of the provider.
- The online doc related to the service/API used, **if available**, linked to the name of the service name.
- How to send a request for the service, possibly shown with ***example headers, parameters bodies, and JSON Schema if applicable.***
- What to expect from the provider, usually shown with ***example responses JSON Schema, if applicable.***

flightradar24.com

Airport Info Light

Request

URL:

```
https://fr24api.flightradar24.com/api/static/airports/${iata_code}/light
```

Headers:

```
{
  'Authorization': 'Bearer $api_key',
  'Accept-Version': 'v1',
  'Accept': 'application/json',
}
```

Response Example

```
{"name":"Beijing Capital International Airport","iata":"PEK","icao":"ZBAA"}
```

spotterguide.net

Spot

Request

URL:

```
https://www.spotterguide.net/?s=${airport.icao}+${airport.iata}
```

Response

Integrated using ***iframe***.

aviationstack.com

API DOC TO BE ADDED

Airport Flight Info

Request

URL:

```
http://api.aviationstack.com/v1/flights
```

Param:

- ***access_key***
- ***arr_icao*** or ***dep_icao***

Response Example


```
{
    "pagination": {
        "limit": 100,
        "offset": 0,
        "count": 100,
        "total": 2807
    },
    "data": [
        {
            "flight_date": "2024-09-28",
            "flight_status": "landed",
            "departure": {
                "airport": "Sydney Kingsford Smith Airport",
                "timezone": "Australia\/Sydney",
                "iata": "SYD",
                "icao": "YSSY",
                "terminal": "1",
                "gate": "61",
                "delay": 20,
                "scheduled": "2024-09-28T09:25:00+00:00",
                "estimated": "2024-09-28T09:25:00+00:00",
                "actual": "2024-09-28T09:45:00+00:00",
                "estimated_runway": "2024-09-28T09:45:00+00:00",
                "actual_runway": "2024-09-28T09:45:00+00:00"
            },
            "arrival": {
                "airport": "Los Angeles International",
                "timezone": "America\/Los_Angeles",
                "iata": "LAX",
                "icao": "KLAX",
                "terminal": "B",
                "gate": "131",
                "baggage": null,
                "delay": 21,
                "scheduled": "2024-09-28T06:10:00+00:00",
                "estimated": "2024-09-28T06:10:00+00:00",
```

```
            "actual": "2024-09-28T06:08:00+00:00",
            "estimated_runway": "2024-09-28T06:08:00+00:00",
            "actual_runway": "2024-09-28T06:08:00+00:00"
        },
        "airline": {
            "name": "Virgin Atlantic",
            "iata": "VS",
            "icao": "VIR"
        },
        "flight": {
            "number": "3822",
            "iata": "VS3822",
            "icao": "VIR3822",
            "codeshared": {
                "airline_name": "delta air lines",
                "airline_iata": "dl",
                "airline_icao": "dal",
                "flight_number": "40",
                "flight_iata": "dl40",
                "flight_icao": "dal40"
            }
        },
        "aircraft": null,
        "live": null
    },
    {
        "flight_date": "2024-09-28",
        "flight_status": "landed",
        "departure": {
            "airport": "Sydney Kingsford Smith Airport",
            "timezone": "Australia\/Sydney",
            "iata": "SYD",
            "icao": "YSSY",
            "terminal": "1",
            "gate": "61",
            "delay": 20,
            "scheduled": "2024-09-28T09:25:00+00:00",
            "estimated": "2024-09-28T09:25:00+00:00",
            "actual": "2024-09-28T09:45:00+00:00",
            "estimated_runway": "2024-09-28T09:45:00+00:00",
            "actual_runway": "2024-09-28T09:45:00+00:00"
        },
        "arrival": {
            "airport": "Los Angeles International",
            "timezone": "America\/Los_Angeles",
            "iata": "LAX",
            "icao": "KLAX",
            "terminal": "B",
            "gate": "131",
            "baggage": null,
            "delay": 21,
            "scheduled": "2024-09-28T06:10:00+00:00",
```

```
                "estimated": "2024-09-28T06:10:00+00:00",
                "actual": "2024-09-28T06:08:00+00:00",
                "estimated_runway": "2024-09-28T06:08:00+00:00",
                "actual_runway": "2024-09-28T06:08:00+00:00"
            },
            "airline": {
                "name": "KLM",
                "iata": "KL",
                "icao": "KLM"
            },
            "flight": {
                "number": "7209",
                "iata": "KL7209",
                "icao": "KLM7209",
                "codeshared": {
                    "airline_name": "delta air lines",
                    "airline_iata": "dl",
                    "airline_icao": "dal",
                    "flight_number": "40",
                    "flight_iata": "dl40",
                    "flight_icao": "dal40"
                }
            },
            "aircraft": null,
            "live": null
        }
    ]
}
```


airportdb.io

Airport Info Detailed

Request

URL:

```
https://airportdb.io/api/v1/airport/${icao_code}
```

Params:

- ***apiToken***

Response Example


```
{
    "ident": "ZBAA",
    "type": "large_airport",
    "name": "Beijing Capital International Airport",
```

```
    "latitude_deg": 40.080101013183594,
    "longitude_deg": 116.58499908447266,
    "elevation_ft": "116",
    "continent": "AS",
    "iso_country": "CN",
    "iso_region": "CN-11",
    "municipality": "Beijing",
    "scheduled_service": "yes",
    "gps_code": "ZBAA",
    "iata_code": "PEK",
    "local_code": "",
    "home_link": "http://en.bcia.com.cn/",
    "wikipedia_link":
"https://en.wikipedia.org/wiki/Beijing_Capital_International_Airport",
    "keywords": "BJS, Bejing, Peking, Olympics",
    "icao_code": "ZBAA",
    "runways": [
        {
            "id": "269343",
            "airport_ref": "27188",
            "airport_ident": "ZBAA",
            "length_ft": "12468",
            "width_ft": "197",
            "surface": "concrete",
            "lighted": "1",
            "closed": "0",
            "le_ident": "01",
            "le_latitude_deg": "40.0589",
            "le_longitude_deg": "116.618",
            "le_elevation_ft": "94",
            "le_heading_degT": "353",
            "le_displaced_threshold_ft": "",
            "he_ident": "19",
            "he_latitude_deg": "40.0928",
            "he_longitude_deg": "116.612",
            "he_elevation_ft": "84",
            "he_heading_degT": "173",
            "he_displaced_threshold_ft": "",
            "he_ils": {
                "freq": 108.9,
                "course": 179
            },
            "le_ils": {
                "freq": 108.5,
                "course": 359
            }
        }
    ],
    "freqs": [
        {
            "id": "51227",
            "airport_ref": "27188",
```

```
            "airport_ident": "ZBAA",
            "type": "TWR",
            "description": "BEIJING  WEST TWR",
            "frequency_mhz": "124.3"
        }
    ],
    "country": {
        "id": "302627",
        "code": "CN",
        "name": "China",
        "continent": "AS",
        "wikipedia_link": "https://en.wikipedia.org/wiki/China",
        "keywords": "中国的机场"
    },
    "region": {
        "id": "303422",
        "code": "CN-11",
        "local_code": "11",
        "name": "Beijing Municipality",
        "continent": "AS",
        "iso_country": "CN",
        "wikipedia_link": "https://en.wikipedia.org/wiki/Beijing",
        "keywords": ""
    },
    "navaids": [
        {
            "id": "94153",
            "filename": "Shaziying_VOR-DME_CN",
            "ident": "SZY",
            "name": "Shaziying",
            "type": "VOR-DME",
            "frequency_khz": "117200",
            "latitude_deg": "40.10169982910156",
            "longitude_deg": "116.46199798583984",
            "elevation_ft": "17",
            "iso_country": "CN",
            "dme_frequency_khz": "117200",
            "dme_channel": "119X",
            "dme_latitude_deg": "",
            "dme_longitude_deg": "",
            "dme_elevation_ft": "",
            "slaved_variation_deg": "",
            "magnetic_variation_deg": "-6.221",
            "usageType": "TERMINAL",
            "power": "HIGH",
            "associated_airport": "ZBAA"
        }
    ],
    "station": {
        "icao_code": "ZBAA",
        "distance": 0
    }
```

```
}
```

suncalc.org

Sun Movement Map

API DOC TO BE ADDED

Request

URL:

```
https://suncalc.org/#/${lat},${lon},${zoom}/${date}/${time}/${objectlevel}/
${maptype}
```

Response

Integrated using ***iframe***.

qweather.com

Hourly Weather

Request

URL:

```
https://devapi.qweather.com/v7/grid-weather/24h
```

Params

- ***lang***
- ***location***
- ***key***

Response Example


```
{
    "code": "200",
    "updateTime": "2024-10-06T16:00+08:00",
    "fxLink": "https://www.qweather.com",
    "hourly": [
        {
            "fxTime": "2024-10-06T10:00+00:00",
            "temp": "17",
            "icon": "102",
            "text": "Few Clouds",
```

```
            "wind360": "107",
            "windDir": "ESE",
            "windScale": "2",
            "windSpeed": "8",
            "humidity": "81",
            "precip": "0.06",
            "pressure": "1018",
            "cloud": "100",
            "dew": "14"
        },
        {
            "fxTime": "2024-10-07T08:00+00:00",
            "temp": "20",
            "icon": "100",
            "text": "Sunny",
            "wind360": "318",
            "windDir": "NW",
            "windScale": "1",
            "windSpeed": "2",
            "humidity": "34",
            "precip": "0.0",
            "pressure": "1019",
            "cloud": "0",
            "dew": "4"
        },
        {
            "fxTime": "2024-10-07T09:00+00:00",
            "temp": "19",
            "icon": "100",
            "text": "Sunny",
            "wind360": "127",
            "windDir": "SE",
            "windScale": "1",
            "windSpeed": "3",
            "humidity": "37",
            "precip": "0.0",
            "pressure": "1019",
            "cloud": "0",
            "dew": "4"
        }
    ],
    "refer": {
        "sources": [
            "QWeather"
        ],
        "license": [
            "CC BY-SA 4.0"
        ]
    }
}
```

adsbexchange.com

Airport/Flight Radar Map

Request

URL:

```
https://globe.adsbexchange.com/
```

Params

- ***airport***(airport code) or ***icao***(flight code)
- ***hideSidebar***
- ***hideButtons***
- ***zoom***

Response

Integrated using ***iframe***.