Singapore ——————————— Sydney

Boarding Pass

Flightship

We ship flights and feelings.

Passenger Name:
Vinny Valeria

Flight:
SEBPT19

Seat:
2A

Gate:
S 25

Date:
June 13, 2025

SEBPT19 Project 2 – General Assembly Singapore 2025

LDR

Singapore ---------- Sydney

Project Objective

Flightship is a travel support app designed to help people in long-distance relationships plan and track their flights more meaningfully.

By combining practical tools—like flight search, saved itineraries, weather risk indicators, and location mapping—with emotional features such as personal messages, Flightship makes long-distance travel feel more connected and intentional.

Features

- Search flights by form inputs

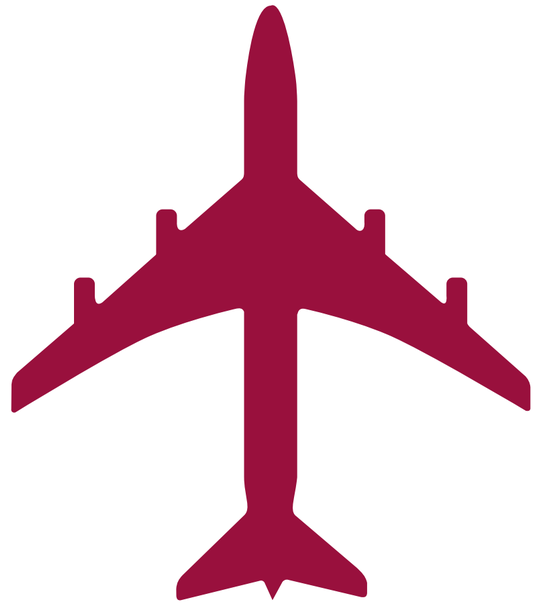
  (origin, destination, date, booking number)
- Save selected flights
- View saved flights in a dedicated page
- See weather conditions (rain/snow/wind) with icons
- Add custom messages to saved flights

Tech Stack

- React (with Vite)
- Chakra UI v3 (UI library)
- React Router
- Leaflet (map)
- Tomorrow.io (weather)
- Airtable (backend DB)
- Postman + fetch (API testing and calls)

Coding Approach

- ✅ Built around APIs: Flight → Map → Weather → Message
- ✅ Used Postman to test APIs
- ✅ Fallback to mock JSON data during development to avoid hitting rate limits
- ✅ Final debugging with real API calls
- ✅ State management with useState, React Router for routing, Chakra for styling
- ✅ Cache with useRef and useContext



Singapore

Sydney

Demo Time

Flight Search Handling

Singapore ———— Sydney

```jsx
FlightSearch.jsx
components > Flights > FlightSearch.jsx > FlightSearch
const FlightSearch = ({ newFlightSearch, setNewFlightSearch, fetch }) => {
    const handleSubmit = async (event) => {
        // stopping the default form submission action
        event.preventDefault();

        // prevent multiple submissions
        if (isSubmitting) return;

        // clear any previous error
        setSubmitError(null);

        // do not store empty input or whitespace
        if (
            !newFlightSearch.bookingId.trim() ||
            !newFlightSearch.apiKey.trim() ||
            !newFlightSearch.departure.trim() ||
            !newFlightSearch.arrival.trim() ||
            !newFlightSearch.departureDate.trim()
        ) {
            setSubmitError("Please fill in all required fields");
            return;
        }

        // set the current submit state after previous checks
        setIsSubmitting(true);

        try {
            // fetch data on submit - await the result
            const fetchData = await fetch(newFlightSearch);

            if (fetchData.success) {
                // reset show suggestion - on success
                setShowSuggestions({
                    departure: true,
                    arrival: true,
                });
            } else {
                // set the error message received
                setSubmitError(fetchData.error);
            }
        } catch (err) {
            // handle unexpected errors
            setSubmitError(err.message || "An unexpected error occurred");
        } finally {
            // once all cases are handled
            setIsSubmitting(false);
        }
    };
```

- Stops the default form behavior (e.g., page reload on submit).
- Prevents multiple submissions while a fetch is already in progress.
- Clears any previous error message before starting the next submission.
- All fields are set to required.
  User will not be able to submit the form without completing the fields.

  Checks if any required field is empty or just whitespace.
  If so, it sets an error message and stops the function early.
- Sets a flag to show that the form is currently being submitted, blocking resubmissions until done.
- Calls flight API with fetch (props) passing in lifted state (newFlightSearch) from App.js.
- If the fetch is successful, it resets the suggestion view.
  If there is an error, it shows the error message.

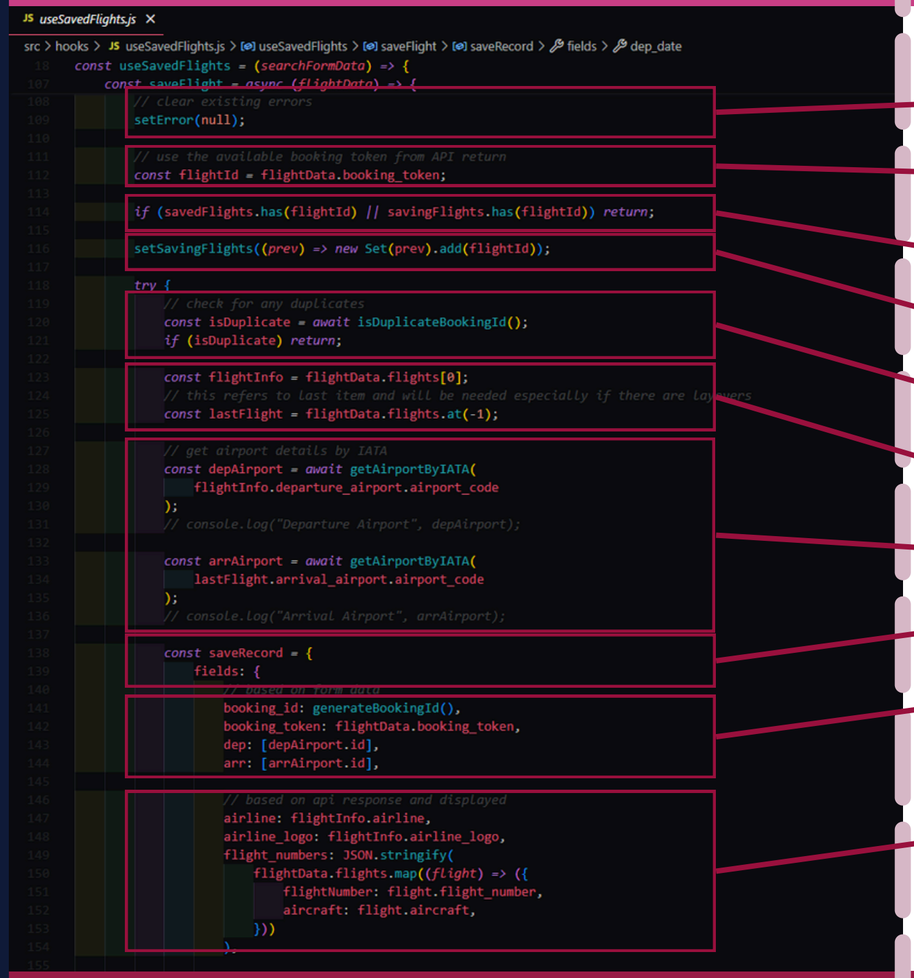
  Both success and error message are custom message created for fetch props.
- Catches unexpected fetch errors (e.g., network issue) and displays a message.
- Resets the isSubmitting flag so the form can be used again.

Demo Time - FlightSearch.jsx

LDR

Saving to Airtable (1/2)

Singapore ———— Sydney



useSavedFlights.js

src > hooks > useSavedFlights.js > useSavedFlights > saveFlight > saveRecord > fields > dep_date

```
const useSavedFlights = (searchFormData) => {
    const saveFlight = async (flightData) => {
        // clear existing errors
        setError(null);

        // use the available booking token from API return
        const flightId = flightData.booking_token;

        if (savedFlights.has(flightId) || savingFlights.has(flightId)) return;

        setSavingFlights((prev) => new Set(prev).add(flightId));

        try {
            // check for any duplicates
            const isDuplicate = await isDuplicateBookingId();
            if (isDuplicate) return;

            const flightInfo = flightData.flights[0];
            // this refers to last item and will be needed especially if there are layovers
            const lastFlight = flightData.flights.at(-1);

            // get airport details by IATA
            const depAirport = await getAirportByIATA(
                flightInfo.departure_airport.airport_code
            );
            // console.log("Departure Airport", depAirport);

            const arrAirport = await getAirportByIATA(
                lastFlight.arrival_airport.airport_code
            );
            // console.log("Arrival Airport", arrAirport);

            const saveRecord = {
                fields: {
                    // based on form data
                    booking_id: generateBookingId(),
                    booking_token: flightData.booking_token,
                    dep: [depAirport.id],
                    arr: [arrAirport.id],

                    // based on api response and displayed
                    airline: flightInfo.airline,
                    airline_logo: flightInfo.airline_logo,
                    flight_numbers: JSON.stringify(
                        flightData.flights.map((flight) => ({
                            flightNumber: flight.flight_number,
                            aircraft: flight.aircraft,
                        }))
                    ),
```

Resets any previously set error state to null so users see a fresh state when saving a new flight.

Retrieves the booking_token from the flight data.
You use this as a unique ID to prevent saving duplicates.

If this flight was already saved (savedFlights) or is currently being saved (savingFlights), exit early to prevent redundancy or race conditions.

Adds the current flightId to the savingFlights state to indicate it is in progress.

Calls your custom function isDuplicateBookingId() to see if the flight already exists in Airtable. If it does, abort saving.

Grabs the first and last flight segments. This is important when layovers are involved.

Fetches full airport data (e.g., Airtable record ID) for departure and arrival using IATA codes.

Constructs a fields object matching Airtable schema.

Values are based on form data. For booking_id, generateBookId() will concatenate the booking number from for with departure and arrival IATA code. Example : ABC123_SIN_SYD

Values are based on flight info from api response.

Storing these will be helpful for displaying savedFlights based on Airtable fetch.

Demo Time - useSavedFlights.js

LDR

Saving to Airtable (1/2)

Singapore ———— ✈ ———— Sydney

useSavedFlights.js

src > hooks > useSavedFlights.js > useSavedFlights > deleteFlight

```
const useSavedFlights = (searchFormData) => {
    const saveFlight = async (flightData) => {
        const saveRecord = {
            fields: {
                hasLayover: flightData.layovers !== null,
                layover_details: flightData.layovers
                    ? JSON.stringify(
                        flightData.layovers.map((layover) => ({
                            duration: layover.duration_label,
                            city: layover.city,
                            airportName: layover.airport_name,
                            airportCode: layover.airport_code,
                        }))
                    )
                    : null,

                // additional metadata - for record tracking purpose
                saved_date: new Date().toISOString(),
                data_source: "Flight API",
            },
        };

        const url = `${AIRTABLE_URL}/${BASE_ID}/${TABLE_ID}`;

        const res = await fetch(url, {
            method: "POST",
            headers: {
                Authorization: `Bearer ${TOKEN}`,
                "Content-Type": "application/json",
            },
            body: JSON.stringify(saveRecord),
        });

        if (!res.ok) {
            const errorData = await res.json();
            console.error("Airtable Error:", errorData);
            throw new Error(`Airtable Error: ${errorData.error?.message}`);
        }

        setSavedFlights((prev) => new Set(prev).add(flightId));
    } catch (err) {
        setError(err.message);
    } finally {
        setSavingFlights((prev) => {
            const updated = new Set(prev);
            updated.delete(flightId);
            return updated;
        });
    }
};
```

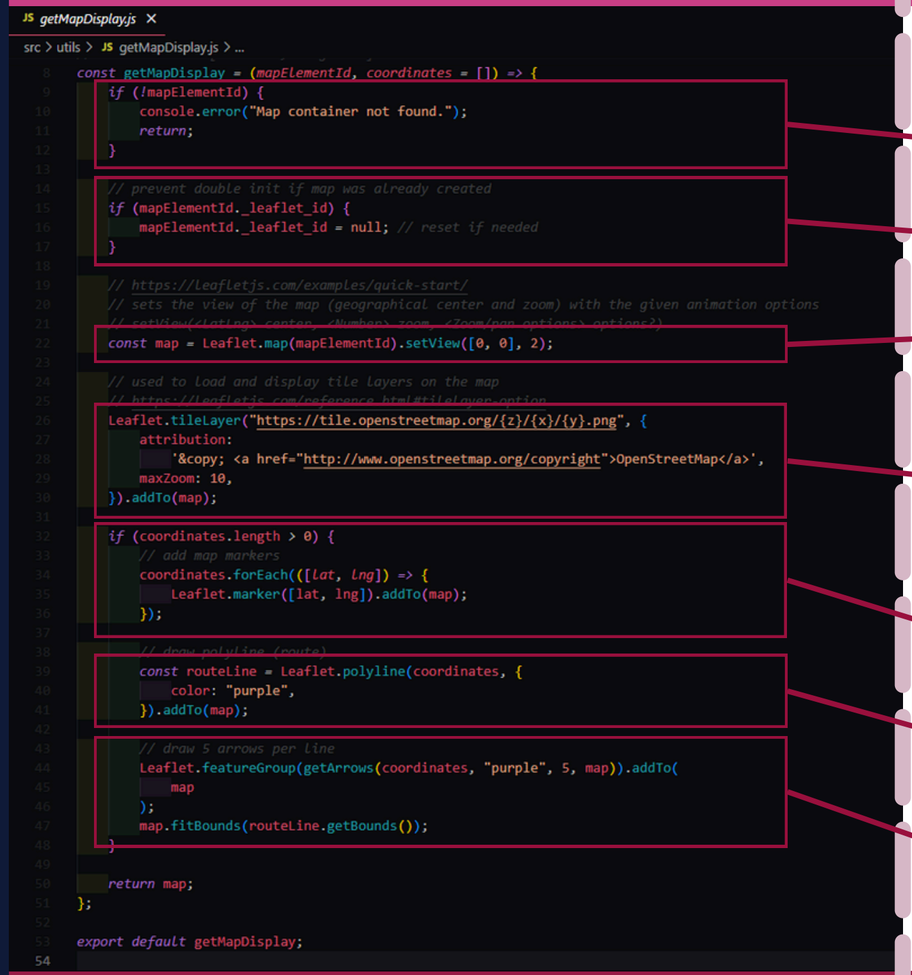
- Includes layover metadata: count, presence, and detailed JSON if available.
- Adds a timestamp and source label for auditing or future debugging.
- Constructs the Airtable API endpoint dynamically.
  All constants are declared at top of the file.
- Sends a POST request with auth headers and your saveRecord payload.
  TOKEN is imported from .env file created.
- If the API returns an error, extract and throw it to be caught below (catch).
- Adds this flight ID to the savedFlights set so it will not be saved again.
- Catches any errors during the save process and sets the error state.
- Removes the flightId from savingFlights to signal completion, regardless of success or failure.

Demo Time - useSavedFlights.js

LDR

Leaflet Map Integration

Singapore ———— Sydney



```
src > utils > getMapDisplay.js > ...
const getMapDisplay = (mapElementId, coordinates = []) => {
    if (!mapElementId) {
        console.error("Map container not found.");
        return;
    }

    // prevent double init if map was already created
    if (mapElementId._leaflet_id) {
        mapElementId._leaflet_id = null; // reset if needed
    }

    // https://leafletjs.com/examples/quick-start/
    // sets the view of the map (geographical center and zoom) with the given animation options
    // setView(<LatLng> center, <Number> zoom, <Zoom/pan options> options?)
    const map = Leaflet.map(mapElementId).setView([0, 0], 2);

    // used to load and display tile layers on the map
    // https://leafletjs.com/reference.html#tilelayer-option
    Leaflet.tileLayer("https://tile.openstreetmap.org/{z}/{x}/{y}.png", {
        attribution:
            '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>',
        maxZoom: 10,
    }).addTo(map);

    if (coordinates.length > 0) {
        // add map markers
        coordinates.forEach(([lat, lng]) => {
            Leaflet.marker([lat, lng]).addTo(map);
        });

        // draw polyline (route)
        const routeLine = Leaflet.polyline(coordinates, {
            color: "purple",
        }).addTo(map);

        // draw 5 arrows per line
        Leaflet.featureGroup(getArrows(coordinates, "purple", 5, map)).addTo(
            map
        );
        map.fitBounds(routeLine.getBounds());
    }

    return map;
};

export default getMapDisplay;
```

- Imports the Leaflet library (for interactive maps).
- Imports a custom helper getArrows (which displays on the polyline in the map)

If mapElementId is falsy (e.g. null or undefined), it logs an error and exits.
This is the id you set on your map div element.

Leaflet attaches _leaflet_id to DOM elements to identify them.
If it already exists, this line resets it to avoid map re-initialization errors.

Creates a new map inside the element.
setView([0, 0], 2) centers the map at the equator (lat 0, lng 0) with zoom level 2 (world view).

Loads OpenStreetMap tiles as the base layer (this is copied from Leaflet tutorial).
Make sure to include add attribution as required by OpenStreetMap's license.

Set maxZoom: 10 prevents zooming in too far.

At the end, adds the tile layer to the map.

If coordinates exist, the following block will render markers, a route line, and directional arrows.

The rendering will be done through loops at each latitude and longtitude pair.

Creates a polyline that connects between markers and set color to purple.

Call custom helper getArrows passing in coordinates, color, frequency and map instance.
This is to be wrapped in featureGroup to work.

The last line is required to automatically zooms and panst the map to fit the bound of the routes.

Demo Time - getMapDisplay.js

LDR

Weather Forecast Logic

Singapore ———— ✈ ———— Sydney

WeatherProvider.jsx M

src > contexts > WeatherProvider.jsx > ...

You, 1 second ago | 2 authors (vinnyvaleria and one other)

```jsx
// src/contexts/WeatherProvider.jsx

import analyseWeather from "@/utils/analyseWeather";

import * as realtimeWeatherService from "@/services/realtimeWeatherService";
// import data from "@/data/weather.json";

import { createContext, useRef } from "react";

// create context for saved weather state
const WeatherContext = createContext();

const WeatherProvider = ({ children }) => {
    // store any fetch data of the same location (to reduce api calls)
    const cache = useRef({});

    const getWeather = async (location) => {
        if (!location) return null;

        if (cache.current[location]) {
            return cache.current[location];
        }

        // fetch weather data based on the location passed in
        const data = await realtimeWeatherService.show(location);
        // console.log("Weather data of ", location, ":", data);

        // get the return of how likely it will rain, snow or windy
        const processed = analyseWeather(data);
        // console.log("Weather analyse result :", processed);

        // include raw data in the storage (cache)
        cache.current[location] = { ...processed, raw: data };
        // console.log("Current weather cache :", cache.current);

        return cache.current[location];
    };

    return (
        <WeatherContext.Provider value={{ getWeather }}>
            {children}
        </WeatherContext.Provider>
    );
};

export default WeatherProvider;
export { WeatherContext };
```

realtimeWeatherService handles the API call to Tomorrow.io,
analyseWeather is a utility function that processes raw data to check for rain, snow, or wind.

This context allows components across the app to access getWeather logic without prop drilling.
Similarly used in savedFlightsProvider.

Initializes an empty cache object using useRef.
Stores weather results keyed by location (e.g., "SIN"), preventing repeated API calls for the same airport.

Checks if a valid location is passed.

If weather data for this location is already fetched, it just returns that without calling the API again.

If not cached, fetches real-time weather data from Tomorrow.io using realtimeWeatherService.

Store the result in the cache, including both the processed and raw data.
Then return it for use in other components.

Wrapped in provider, exposing getWeather through context.
Allows any child component to call getWeather(location) and receive clean, preprocessed, and possibly cached weather data.

Demo Time - WeatherProvider.jsx

LDR

Weather Icon Rendering

Singapore ———— ✈ ———— Sydney

WeatherFlex.jsx

src > components > Weather > WeatherFlex.jsx > ...

vinnyvaleria, 4 hours ago | 1 author (vinnyvaleria)

```jsx
// src/components/Weather/WeatherFlex.jsx

import { Spinner, Flex, Status } from "@chakra-ui/react";

import { FaCloudRain, FaRegSnowflake, FaWind } from "react-icons/fa";

const WeatherFlex = (data) => {
    return (
        <>
            {data.loading ? (
                <Spinner size="sm" />
            ) : data.error === "" ? (
                <Status.Root colorPalette="red">
                    <Status.Indicator />
                    Fail weather fetch
                </Status.Root>
            ) : (
                <Flex gap={2} color="lightblue">
                    <FaCloudRain
                        size="20px"
                        style={{ opacity: data.isRainLikely ? 1 : 0.2 }}
                    />
                    <FaRegSnowflake
                        size="20px"
                        opacity={data.isSnowLikely ? 1 : 0.2}
                    />
                    <FaWind size="20px" opacity={data.isWindy ? 1 : 0.2} />
                </Flex>
            )}
        </>
    );
};

export default WeatherFlex;
```

Imports Chakra components and react-icons.
Searched from https://react-icons.github.io/react-icons/icons/lu/

Uses a ternary chain to determine what to display based on state.

When there is no loading and no error, the component displays a horizontal row of three weather icons—rain, snow, and wind—using a Chakra UI Flex layout.

Each icon is visually styled based on the weather condition it represents: the rain icon is fully visible if isRainLikely is true, and faded with 0.2 opacity if false. The snow and wind icons follow the same logic using their respective condition booleans, isSnowLikely and isWindy.

The criteria of these 3 weather info is taken from analyseWeather.js

- isRainLikely: values.precipitationProbability > 50 || values.rainIntensity > 0.2,
- isSnowLikely: values.snowIntensity > 0,
- isWindy: values.windGust > 12,

where values is the response from the realtimeWeatherService

The gap={2} property adds spacing between the icons, and the entire row is styled with a light blue color to indicate weather-related content.

Demo Time - WeatherFlex.jsx

LDR

Key Takeaways

✅ How to integrate multiple APIs into one seamless flow.

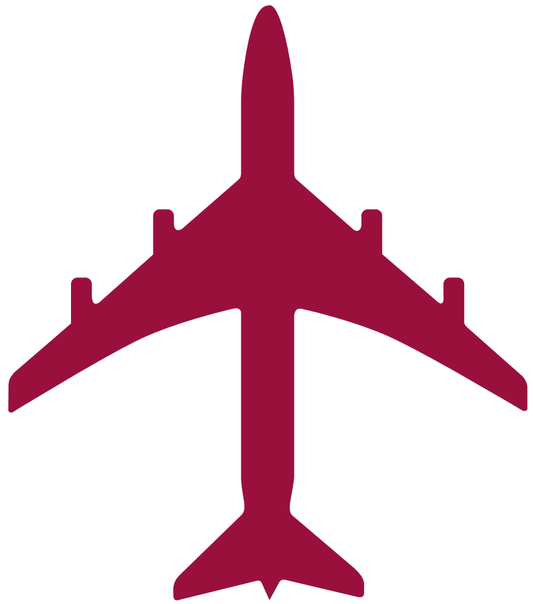
✅ Relational data structuring with Airtable.

✅ Chakra UI for clean, responsive design.

✅ Development under API rate limits using fallback JSON.

✅ Learned to manage technical scope and simplify logic under constraints.

✅ Reflected on time management: balancing coding, debugging, and planning while juggling work commitments taught me the importance of early milestones and tighter iteration loops.



In conclusion: I have no idea how I pulled this off either. But here we are.

— Thank you!