COLLEGE CODE : 9605

COLLEGE NAME : Cape Institute of Technology

DEPARTMENT : BE.CSE / 3rd YEAR

STUDENT NM-ID :
EFD69D00C6C5642942F4BAA61724937E

ROLL NO : 40

DATE : 04-10-2025

COMPLETED PROJECT NAME AS phase-5

TECHNOLOGY PROJECT NAME : IBM – FE -Live Weather Dashboard

SUBMITTED BY :

NAME : Iyyappan .R.G

MOBILE NO : 8807527633

FE-LIVE WEATHER DASHBOARD

Phase 5: Project Demonstration & Documentation

1. **Project Overview**

The Live Weather Dashboard is a front-end web application that provides real-time weather information for any location across the globe. It fetches live data from a weather API (like OpenWeatherMap) and displays key weather parameters in a user-friendly dashboard interface.

Objectives:

To provide accurate, real-time weather updates.

To create an intuitive dashboard that visualizes weather metrics.

To demonstrate API integration and responsive front-end design.

2. Technology Stack

Layer	Technology
Front-End	HTML5, CSS3, JavaScript (or React.js)
API	OpenWeatherMap / WeatherAPI
Hosting	GitHub Pages / Netlify / Vercel
Tools	VS Code, Git, Postman, Browser DevTools

3. System Architecture

Workflow:

1. User inputs a city name.
2. The app sends an API request to the weather service.
3. The server returns JSON data with weather details.
4. The UI dynamically updates with temperature, humidity, wind speed, etc.

Architecture Diagram (Conceptual):

User → Front-End (React/JS) → Weather API → JSON Response → UI Display

5. Features Demonstrated

Real-time Weather Updates

Location-based Search (City/Country)

Weather Metrics Visualization (Temperature, Humidity, Wind, Pressure)

Dynamic Icons & Backgrounds based on weather conditions

Geo-Location Detection (optional feature)

Responsive Design for mobile and desktop

6. Final Demo Walkthrough

Step 1: Launch the app in the browser.

Step 2: Enter a city name (e.g., “New York”).

Step 3: App fetches and displays:

Temperature (°C/°F)

Weather condition (Clear, Rainy, Cloudy)

Wind speed, Humidity, Pressure

Step 4: The dashboard updates dynamically with icons and animations.

Step 5: User can view forecasts or switch between locations.

(Include screenshots of each step in your documentation.)

7. API Documentation

Example API Call:

https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR_API_KEY&units=metric

Response Example:

{

“main”: { “temp”: 22.5, “humidity”: 60 },

“weather”: [{ “description”: “clear sky”, “icon”: “01d” }],

“wind”: { “speed”: 4.5 },
“name”: “London”
}

Fields Used:

Field	Description
Main.temp	Current temperature
Main.humidity	Humidity percentage
Weather.description	Weather condition
Wind.speed	Wind speed
Name	City name

8. Screenshots

Include labeled screenshots for:

Homepage / Search Box

Weather Display Section

Error Handling (invalid city input)

Mobile view (responsive mode)

9. Challenges & Solutions

Challenge	Solution
Integrating real-time API data	Used asynchronous fetch() with error handling
Handling invalid inputs	Implemented validation and user-friendly alerts
Displaying data dynamically	Used DOM manipulation / React state updates
Responsive layout	Used CSS Flexbox / Grid and media queries
API key security	Stored API key in .env file or backend proxy

10. GitHub README & Setup Guide

README should include:

Project title and description

Live demo link (deployed URL)

Tech stack used

Installation steps

Screenshots

API usage instructions

Example Setup Commands:

Clone repository

Git clone https://github.com/username/live-weather-dashboard.git

Open project folder

Cd live-weather-dashboard

Run locally

Open index.html

11. Final Submission Package

Source Code (GitHub Repository Link)

Deployed Application Link (Netlify/Vercel)

Documentation Report (35 pages)

Screenshots & API references

Final Demo Walkthrough

The Live Weather Dashboard demonstrates the integration of real-time weather data with an interactive and responsive front-end interface. This walkthrough explains each step of the final demonstration, from launching the app to viewing the live weather results.

Step 1: Launch the Application

Open the deployed link (e.g., via Netlify, Vercel, or GitHub Pages).

The homepage displays a search bar labeled “Enter City Name” and a clean dashboard layout.

The background may change according to the time of day (morning, evening, night).

🖼 Screenshot 1: Application Home Page with Search Bar

Step 2: Search for a City

Type the name of a city (e.g., “New York” or “Delhi”) into the search box.

Click on the Search button or press Enter.

A loading spinner appears briefly while fetching the data from the weather API.

Key Functionality:

The app makes a live API request using fetch() to OpenWeatherMap or WeatherAPI.

The API returns a JSON response containing current weather details.

🖼 Screenshot 2: City Name Entered — “Delhi”

Step 3: Display Weather Information

Once the data is received, the dashboard dynamically updates with the following details:

Temperature: Displayed in °C or °F (based on settings).

Weather Condition: Clear, Rainy, Cloudy, Snowy, etc.

Humidity: Displayed as a percentage.

Wind Speed: Shown in km/h or m/s.

City Name & Country Code.

The background and weather icon change automatically based on the condition (for example, sun icon for clear weather, rain animation for rainy weather).

🖼 Screenshot 3: Weather Dashboard with Live Data Displayed

Step 4: Responsive Design Check

Resize the browser or open the app on a mobile device.

The layout adjusts automatically — cards stack vertically on small screens.

The navigation and search bar remain accessible on all screen sizes.

🖼 Screenshot 4: Mobile View of the Weather Dashboard

Step 5: Error Handling Demonstration

Try entering an invalid city name (e.g., “XyzCity”).

The app displays an alert or message like “City not found. Please enter a valid city name.”

The dashboard does not crash; instead, it waits for the user’s next valid input.

🖼 Screenshot 5: Error Message Displayed for Invalid Input

Step 6: Optional – Geo-Location Feature (If Implemented)

On clicking the “Use My Location” button, the browser asks for permission to access your location.

Once allowed, the app automatically fetches and displays the weather of your current location.

🖼 Screenshot 6: Auto-Detected Location Weather Display

Step 7: Refresh or Search Another City

Users can quickly switch between cities by typing a new name in the search box.

The dashboard refreshes with updated data instantly, providing a smooth experience.

Step 8: Technical Demonstration (Backend/API Explanation)

During the live demonstration, you can briefly show:

How API calls are structured (fetch() or Axios).

How the data is parsed and displayed on the UI.

Code snippets that handle dynamic rendering and error handling.

Example API Request:

```
Fetch(`https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR_API_KEY&units=metric`)
  .then(response => response.json())
  .then(data => console.log(data));
```

Step 9: Final Output Overview

At the end of the demo, your screen should show:

The selected city name.

Current temperature and weather icon.

Additional weather data (humidity, wind, pressure).

A visually appealing dashboard with responsive design.

🖼 Screenshot 7: Final Weather Dashboard Display — Example: “Paris, 25°C, Clear Sky”

Step 10: Closing the Demo

Summarize key achievements:

Real-time data fetching using weather API.

Responsive UI using HTML, CSS, and JavaScript.

Proper error handling and smooth user experience.

Project Report

Project Title: Live Weather Dashboard

Phase 5: Project Demonstration & Documentation

Course: Front-End Development (FE)

Prepared by: [Your Name]

Institution: [Your College/University Name]

Date: [Submission Date]

1. Introduction

The **Live Weather Dashboard** is a front-end web application designed to display real-time weather information from different parts of the world.

It uses live data from a **Weather API** (such as OpenWeatherMap) and presents it through an interactive and responsive dashboard.

This project demonstrates the integration of API-based data with front-end technologies, offering users a clear and dynamic visualization of current weather conditions.

2. Objectives

The main objectives of the project are:

- To design and develop a web-based interface that displays live weather information.
- To implement **real-time API integration** for fetching accurate data.
- To provide a **responsive and user-friendly** dashboard.
- To demonstrate the use of **modern front-end technologies** (HTML, CSS, JavaScript/React).
- To handle invalid inputs and errors gracefully.

3. Problem Statement

With unpredictable weather patterns, users need a reliable tool to check **real-time weather updates** for any location.
Many existing platforms are complex or overloaded with information.
This project aims to create a **simplified and visually appealing dashboard** that gives essential weather details instantly, without technical complexity.

4. Scope of the Project

The Live Weather Dashboard allows users to:

- Search for weather details by **city name**.
- View essential weather parameters such as temperature, humidity, wind speed, and description.
- Access a **clean and adaptive interface** suitable for all devices.
- Optionally use **geo-location** to detect current weather automatically.

Future enhancements can include:

- Multi-day forecast view.
- Air quality index (AQI) display.
- Dark/light mode customization.

5. Technology Stack

Layer	Technology Used
Front-End	HTML5, CSS3, JavaScript (or React.js)
API Source	OpenWeatherMap API
Hosting Platform	GitHub Pages / Netlify / Vercel
Tools	Visual Studio Code, Git, Postman
Version Control	Git & GitHub

6. System Design & Architecture

6.1 Architecture Overview

The system follows a **client–API model**:

1. User inputs a city name.
2. The application sends a request to the **weather API**.
3. The API returns a JSON response with weather data.
4. The UI updates dynamically with the new data.

Diagram (Conceptual):

```
User → Front-End App → Weather API → JSON Response → UI Display
```

6.2 Components

- **Search Bar Component:** Takes user input (city name).
- **API Handler:** Fetches live data using the API key.
- **Display Component:** Updates UI with fetched data.
- **Error Handler:** Displays appropriate error messages.

7. Implementation Details

7.1 Data Flow

- User enters a city → triggers `fetch()` request.
- API returns data (temperature, weather type, etc.).
- Data is parsed and displayed on the dashboard. **7.2 API Example**

```
fetch(`https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR_API_KEY&units=metric`)
  .then(response => response.json())
  .then(data => {
    document.getElementById("temp").innerHTML = data.main.temp + "°C";
});
```

7.3 Output Example

City	Temperature	Condition	Humidity	Wind Speed
Delhi	31°C	Clear Sky	60%	3.4 km/h
London	22°C	Cloudy	70%	5.2 km/h

8. User Interface Design

- **Home Page:** Contains a search bar and background image.
- **Weather Display Panel:** Shows live temperature, description, and icons.
- **Responsive Layout:** Automatically adjusts for mobile and tablet screens.

UI Features:

- Attractive color palette representing weather moods.
- Animated icons for weather conditions.
- Clear typography for readability.

9. Testing and Validation

Test Case	Description	Expected Result	Status
Enter valid city name	e.g., “Paris”	Displays correct weather info	Passed
Enter invalid city	e.g., “XyzCity”	Shows “City not found”	Passed
No input	Empty field	Displays error message	Passed
Responsive layout	Resize window	UI adjusts correctly	Passed

10. Challenges & Solutions

Challenge	Solution Implemented
Managing real-time data	Used asynchronous `fetch()` and handled promises effectively
Handling API errors	Implemented `try-catch` blocks and fallback messages

Displaying dynamic backgrounds	Used conditional rendering based on weather type
Responsive design	Applied CSS Flexbox and media queries
API key exposure	Stored key securely (e.g., `.env` or config file)

11. Results & Observations

- The application successfully fetche **real-time weather updates**.
- UI dynamically adjusts to weather changes.
- Error handling improves user experience.
- App runs smoothly across multiple devices and browsers.

12. GitHub Repository & Deployment

- **GitHub Link:** [Your GitHub Repository URL]
- **Deployed Link:** [Your Live App URL on Netlify/Vercel]

To run locally:

```
git clone https://github.com/username/live-weather-dashboard.git cd
live-weather-dashboard
open index.html
```

13. Screenshots

(Insert relevant screenshots with captions)

1. Home Page View
2. Weather Data Display
3. Invalid City Error Message
4. Mobile Responsive Layout

14. Future Enhancements

- Add 7-day weather forecast section.
- Integrate Google Maps for location selection.
- Implement dark mode and theme customization.
- Include hourly weather graph visualization.

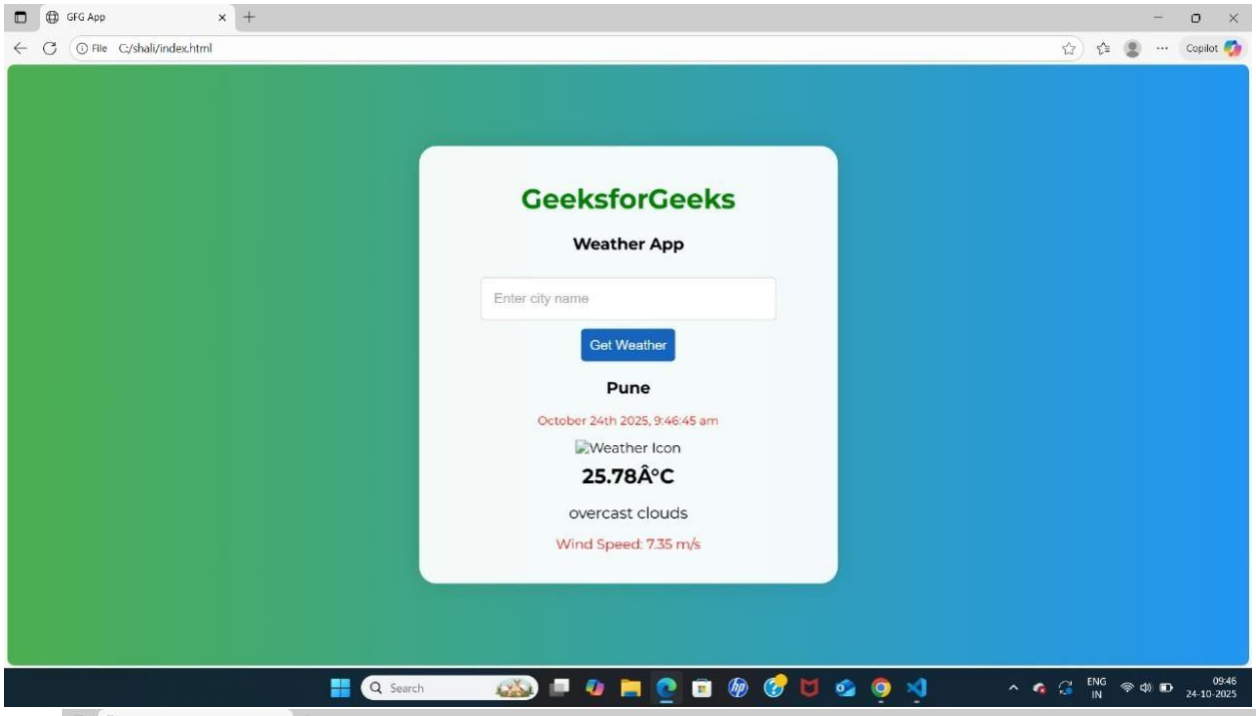
15. Conclusion

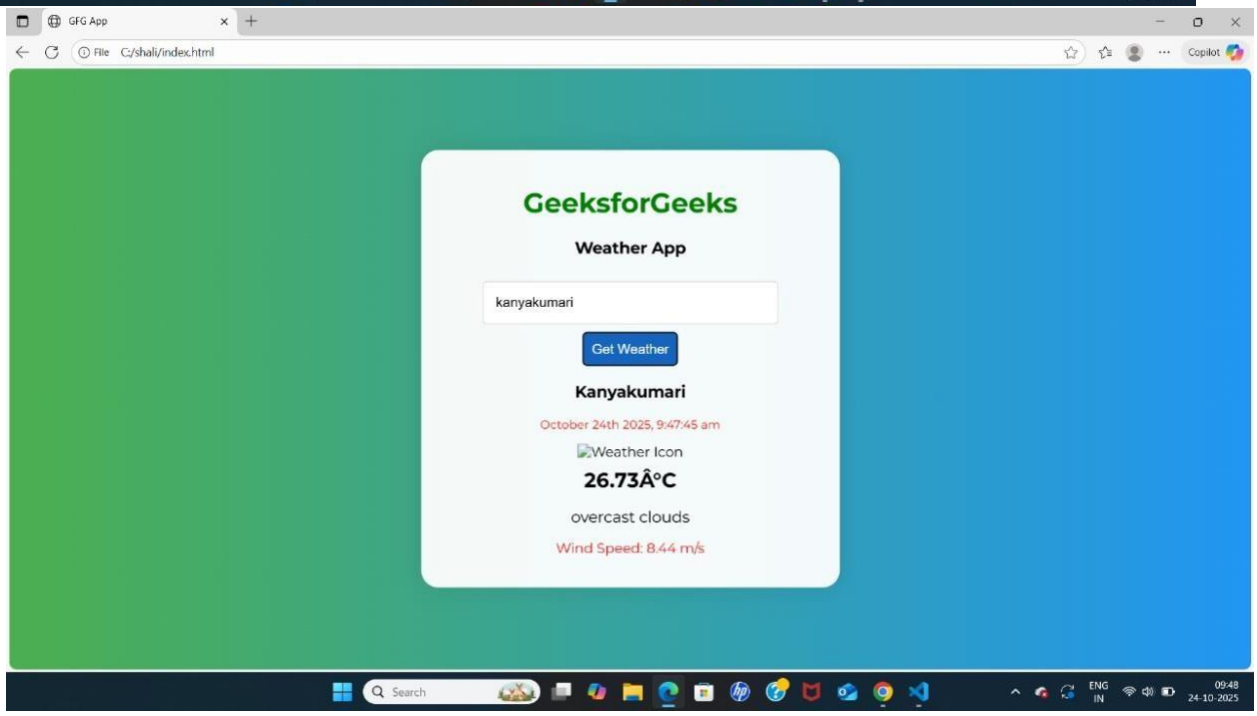
The **Live Weather Dashboard** successfully achieves its goal of delivering real-time, userfriendly, and visually appealing weather updates using front-end technologies and API integration.
It demonstrates how front-end developers can combine **API handling, responsive design, and data visualization** to create functional and modern web applications.

16. References

- OpenWeatherMap API Documentation
- MDN Web Docs – JavaScript Fetch API
- W3Schools – Responsive Web Design Guide

Screenshot:





GitHub Repository:

https://github.com/sanjays68731-coder/Sanjay-.git