## 

**COLLEGE CODE: 8203**

**COLLEGE: AVC COLLEGE OF ENGINEERING**

**DEPARTMENT: INFORMATION TECHNOLOGY**

**STUDENT NM-ID: au820323205075**

**ROLL NO: 23IT75**

**DATE:**

**Completed the project named as**

**Phase\_\_ TECHNOLOGY PROJECT**

**NAME:**  **Weather Dashboard**

**SUBMITTED BY,**

**NAME: PRADEEP KUMAR R**

**MOBILE NO: 6379897924**

## Tech Stack Selection – Weather Dashboard

**1. Frontend (User Interface)**

* **HTML, CSS, JavaScript** → Simple, responsive UI for city input & results.
* (Optional) **React.js** → For modern, dynamic, and mobile-friendly interface.

*Reason:* Lightweight, easy to use, responsive across devices.

**2. Backend (Server Logic)**

* **Node.js** → Handles multiple requests efficiently with non-blocking I/O.
* **Express.js** → Simplifies route handling and API integration.

*Reason:* Scalable, fast, widely used for REST APIs.

**3. API Integration**

* **Axios** → HTTP client to fetch data from Open Weather API.
* **Open Weather API** → Reliable, real-time weather information.

*Reason:* Simple, fast, and flexible for external API calls.

**4. Caching Layer**

* **Redis** → Stores frequently requested city weather data.

*Reason:* Improves performance and reduces API usage costs.

**5. Tools & Utilities**

* **Git/GitHub** → Version control & collaboration.
* **Figma** → For designing wireframes & UI mockups
* **Postman** → API testing.

## Ui Structure & API Schema Design

**UI Structure – Weather Dashboard**

**Homepage / Dashboard:**

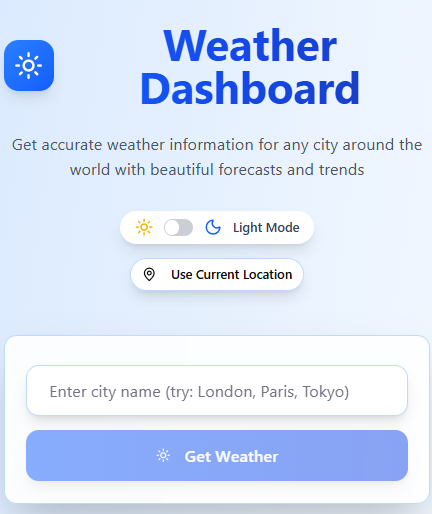
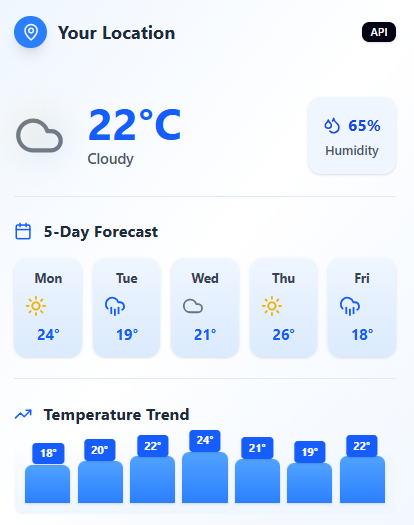
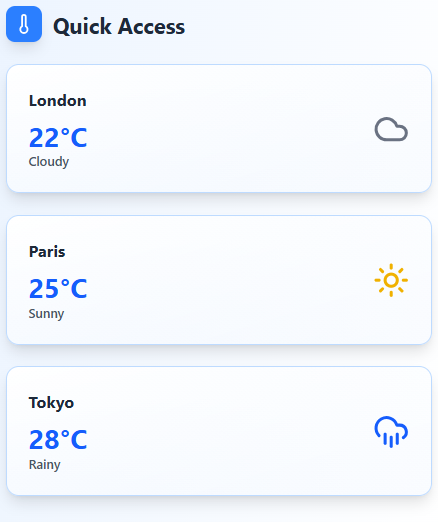
* **Header** → “Weather Dashboard”
* **Search Section**

Input box: *Enter City Name*

* + Button: *Get Weather*
* **Weather Result Card**
  + City Name (bold)
  + Temperature (°C)
  + Humidity (%)
  + Weather Condition (with icon ☀️☁️🌧️)
  + Source Label (API / Cache)
* **Error Message Section**
  + Red text: “City not found” / “API unavailable”

**Additional (Optional):**

* 5-Day Forecast Section (small horizontal cards).
* Theme Toggle (Light/Dark).
* Current Location Weather (if GPS allowed).

**API Schema – Weather Dashboard**

1. **Endpoint: /weather**

**Method:** GET  
 **Request Params:**

* city (string, required)

**Success Response (API Source):**

**JSON**

{

"source": "api",

"data": {

"city": "London",

"temperature": 25,

"humidity": 60,

"condition": "clear sky"

}

**}**

**Success Response (Cache Source):**

**JSON**

**{**

"source": "cache",

"data": {

"city": "London",

"temperature": 25,

"humidity": 60,

"condition": "clear sky"

**}**

**}**

**Error Response (Invalid City):**

**JSON**

**{**"error": "City not found"

**}**

1. **Endpoint:** /forecast **(optional)**

**Method:** GET **Request Params:**

* city (string, required)

**Response:**

**JSON**

**{**

"city": "London",

"forecast": [

{"date": "2025-09-21", "temp": 24, "condition": "sunny"},

{"date": "2025-09-22", "temp": 22, "condition": "rain"}

**]**

**}**

1. **Endpoint: /health**

**Method:** GET

**Response:**

**JSON**

**{**

**"status": "ok"**

**}**

## Data Handling – Weather Dashboard

**1.User Input Handling**

* User enters a city name in the search box.
* Frontend validates (empty input / invalid characters not allowed).
* Request sent to backend with query: /weather? city=London.

**2. Backend Data Handling**

* **Step 1: Cache Check**
  + Backend checks Redis cache first.
  + If city found → return cached data (faster response).
* **Step 2: API Fetch**
  + If not in cache, backend calls Open Weather API using Axios.
  + Weather data (temperature, humidity, condition) is received in JSON format.
* **Step 3: Data Transformation**
  + Raw API response → simplified JSON object**.**
  + **Example:**

**{**

**"city": "London",**

**"temperature": 25,**

**"humidity": 60,**

**"condition": "clear sky"**

**}**

* **Step 4: Cache Update**
  + The transformed data is stored in Redis with a TTL

**3.Frontend Data Handling**

* Receives cleaned JSON from backend.
* Displays City, Temperature, Humidity, Condition on the UI card.
* If error → shows message:
  + “City not found” (Invalid city)
  + “Service unavailable” (API down).

**4.Error Data Handling**

* Invalid city → return error JSON:

**{**

"error": "City not found"

**}**

* API failure → return fallback error JSON:

**{**

"error": "Failed to fetch weather data"

**}**

* Frontend shows red warning text instead of crashing.

**5. Optional / Advanced Handling**

* Logging: Store user search history in DB (optional).
* Forecast Data: Handle multiple days weather array.
* User Location: Handle GPS permission and auto-fetch weather.

## Components of Weather Dashboard

**1. Frontend (User Interface)**

* Input Box → Enter city name
* Get Weather Button → Trigger request
* Result Card → Show city, temperature, humidity, condition, source
* Error Message Section → Display invalid city / API down
* (Optional) Forecast Section, Theme Toggle

**2.Backend (Application Layer – Node.js + Express)**

* **Route Handlers**
  + /weather → Fetch current weather
  + /forecast → Get 5-day forecast (optional)
  + /health → Check server status
* **Controller Module** → Handles logic between frontend & API
* **Error Handling Module** → Returns user-friendly error messages

**3. External API Integration**

* **Open Weather API**
  + Provides real-time weather data
  + Communicates via Axios

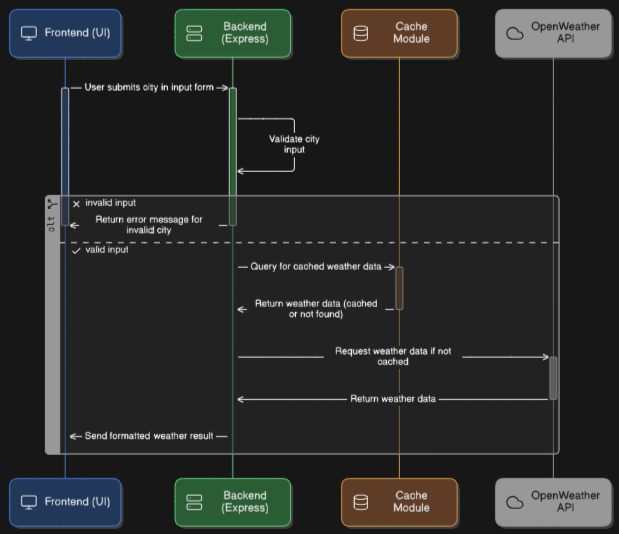
**4. Caching Layer**

* **Redis**
  + Stores frequently queried city weather
  + Reduces repeated API calls
  + TTL (expiry time) for fresh data

**5.Utilities & Tools**

* **Axios** → API requests
* **Postman** → Testing APIs
* **Git/GitHub** → Version control
* **Figma** → Wireframes

**Module Diagram**

****

## Basic Flow diagram

