9530

St. MOTHER THERESAENGINEERING COLLEGE

COMPUTER SCIENCE ENGINEERING

NM-ID: 268CBE2006014D0837B47E5D639E631A

REG NO: 953023104012

DATE: 14-09-2025

Completed the project named as

Phase 2

FRONT END TECHNOLOGY

LIVE WEATHER DASHBOARD

SUBMITTED BY,

M. ANTRO PRATHIK SAM

9344879821

Phase 2 – Solution Design & Architecture

Tech Stack Selection

- **Frontend: React.js (with hooks for state management) and TailwindCSS for styling. React is widely used for building interactive dashboards. We may also use UI libraries (Bootstrap/Tailwind) for layout.**

- **APIs: OpenWeatherMap API (free tier) for weather data. (As noted in tutorials, the OpenWeatherMap API provides current and forecast weather data via simple REST calls.)**

- **Backend (optional): Node.js with Express (if needed to proxy API calls or manage API keys securely). For basic apps, we can call the API directly from React, but a lightweight Express server can cache requests or handle heavy loads.**

- **State & Data Storage: React Context or simple Hooks (useState , useEffect) for state; localStorage for saving search history (as in typical weather dashboard examples).**

- **Development Tools: Axios or Fetch API for HTTP requests; GitHub for code; Vite or Create React App for bootstrapping the project.**

UI Structure / API Schema Design

- **UI Components:**
- *SearchBar* – input field and button to submit city name.
- *CurrentWeather* – displays city name, date, condition icon, and metrics (temp, humidity, wind, UV).
- *ForecastList* – container showing 5 *ForecastCard* components.
- *ForecastCard* – shows one day's date, icon, and temperature/humidity.
- *HistoryList* – sidebar or dropdown of previously searched cities (clickable).
- *Header/Footer* – page header with app title (e.g. "Live Weather Dashboard")

- **API Data Schema (example):**

```
{
"city": "New York",
"current": {
                    "date": "2025-09-13",
                     "icon": "10d",
                     "temp": 28.5,
                     "humidity": 65,
                     "wind": 5.2,
                     "uvIndex": 7.1
 },
  "forecast": [
       {"date": "2025-09-14", "icon": "10d", "temp": 26.3,
"humidity": 60},
]
}
```

(In React, we might shape the API response into a similar object or directly use the OpenWeatherMap response fields.)

Data Handling Approach

Data flows from the weather APIs into the UI components. We will fetch JSON data asynchronously (using fetch or Axios in React's useEffect) whenever a user searches or selects a city. The returned data will be stored in React state (e.g. weatherData) which triggers re-render of components. For example, once current weather JSON is received, we update state and React updates the *CurrentWeather* card with the new values. Search history is stored in localStorage so it persists across sessions (as in similar apps). UV index values will be interpreted (e.g., shown in color) by the component rendering. Any loading state (e.g. spinner) will be managed while data is being fetched.

Component / Module Diagram

We break the system into modules:

- **WeatherService** – a module (or context) that contains functions to call the weather API (e.g. getCurrentWeather(city) , getForecast(city)).

- **Components Tree:** The root App component includes a SearchBar and a container that either shows CurrentWeather + ForecastList or prompts for input. Data flows via props or context from top-level state to child components.

- **State Management:** Weather data and search history may reside in a React Context or in the top-level App state. Components subscribe to this state.

- **Diagram Sketch (conceptual):** *[Not shown]* A UML-like component diagram would show App ->SearchBar , App -> CurrentWeather , App -> ForecastList , and a WeatherService module connected to external OpenWeatherMap API.

Basic Flow Diagram

1. **User Enters City:** The user types a city name into SearchBar and clicks “Search”.
2. **Fetch Weather Data:** The app calls the WeatherService API functions:
3. getCurrentWeather(city) -> calls OpenWeatherMap /weather .
4. getForecast(city) -> calls OpenWeatherMap /forecast .
5. **State Update:** On receiving data, the app updates React state: currentWeather = response1 , forecast = response2 .
6. **Render UI:** React re-renders: CurrentWeather component displays the new weather (temperature, icon, etc.), and ForecastList updates with day cards.
7. **Save History:** The city name is added to search history state and stored in localStorage.
8. **User Interaction:** If the user clicks a past city in history, steps 2–4 repeat for that city (fetch or retrieve from cache), updating the display.

This completes the design and architecture for our Live Weather Dashboard project. All data paths and interactions are thus defined to meet the requirements.

References: Authoritative examples and documentation were used to inform design choices

Currentweatherdata-OpenWeatherMap
https://openweathermap.org/current

OneCallAPI3.0-OpenWeatherMap
https://openweathermap.org/api/one-call-3

GitHub - sylviaprabudy/weather-dashboard: A simple web application that allows users to search for a city to get the current weather and 5 day forecast. Cities that users previously looked up will be saved in their local storage. Retrieve weather data from https://openweathermap.org/

https://github.com/sylviaprabudy/weather-dashboard

Build a Weather Dashboard: Your First API Project with React – DEVCommunity

https://dev.to/syawqy/build-a-weather-dashboard-your-first-api

project-with-react-4j7h

Create a simple weather app using Node.js, Express, and React | by Maison Moa | Medium

https://medium.com/@maison.moa/create-a-simple-weather-app
using-node-js-express-and-react-54105094647a