 A black and white logo

AI-generated content may be incorrect.

**COLLEGE CODE:**8203

**COLLEGE NAME:**AVC COLLEGE OF ENGINEERING

**DEPARTMENT NAME:** CSE

**STUDENT NM ID:**

5029EE305301B2FA1CF4AA1F1A50DB0D

**REG NO:** 820323104061

**DATE:**15-09-2025

PHASE 2 - SOLUTION DESIGN AND ARCHITECTURE

**PROJECT NAME:**WEATHER DASHBOARD

**SUBMITTED BY :**

S.MEENATCHI SUNDARI

**MOBILE NO :**9943928868

Designing a **Weather Dashboard** with 4 pages requires a scalable, maintainable, and user-friendly **Solution Design and Architecture**. Here's a comprehensive breakdown covering:

* Page-wise layout
* Frontend and backend architecture
* APIs & integrations
* Database (if applicable)
* Technologies
* Security & performance considerations

| **Page** | **Description** |
| --- | --- |
| **1.Dashoard** | Displays current weather, temperature, humidity, wind speed, etc. |
| **2. Forecast** | Shows 5- or 7-day weather forecast with temperature trends, icons, etc. |
| **3.Radar map** | Displays interactive weather radar with precipitation, storms, etc. |
| **4. Settings** | Allows users to choose location, units (C/F), theme, language, etc. |

**Solution Architecture**

**1. Frontend (Client-side)**

* **Framework**: React / Vue / Angular (React recommended)
* **Routing**: React Router (or Vue Router)
* **State Management**: Redux / Context API / Pinia (Vue)
* **Components**:
  + WeatherCard, ForecastGraph, RadarViewer, SettingsForm
  + Navbar & Sidebar for navigation
* **Design**: Responsive (Tailwind CSS / Material UI / Bootstrap)

**2. Backend (API layer)**

* **Framework**: Node.js with Express / Python Flask / FastAPI
* **Endpoints**:
  + /api/weather/current?location=...
  + /api/weather/forecast?location=...
  + /api/weather/radar?location=...
  + /api/user/settings (GET/POST/PUT)
* **Authentication (if needed)**: JWT / OAuth2 (optional for user prefs)

**3. Third-Party API Integration**

* **Weather APIs**:
  + OpenWeatherMap
  + WeatherAPI.com
  + Climacell / Tomorrow.io
* **Maps/Radar APIs**:
  + Mapbox + RainViewer
  + Leaflet + OpenWeatherMap radar overlays

**4. Database (optional, for settings)**

* **Type**: NoSQL (MongoDB) or SQL (PostgreSQL)
* **Tables/Collections**:
  + users: { id, location, units, theme, language }

**Detailed Page Design & Flow**

**1. Dashboard Page**

* **Components**:
  + Current temperature, city, date/time
  + Weather icon (sunny, rain, etc.)
  + Stats: wind, humidity, pressure
* **Data Source**: GET /api/weather/current
* **Updates**: Auto-refresh every 10 min

**2. Forecast Page**

* **Components**:
  + Line/Bar chart of temp trends
  + Cards for each day (icon, highs/lows)
* **Data Source**: GET /api/weather/forecast
* **Library**: Chart.js / Recharts

**3. Radar Map Page**

* **Components**:
  + Interactive map (zoom/pan)
  + Toggle layers: radar, clouds, wind
* **Libraries**:
  + Leaflet.js or Mapbox GL
  + Overlay from Weather API
* **Data Source**: External radar tiles

**4. Settings Page**

* **Components**:
  + Input: location (autocomplete with Mapbox or Google Places)
  + Units (°C/°F), theme (dark/light), language
  + Save button
* **Data Source**: GET/POST /api/user/settings

**Tech Stack Summary**

| **Layer** | **Technology** |
| --- | --- |
| **Frontend** | React + Tailwind CSS / Material UI |
| **Backend** | Node.js + Express or FastAPI |
| **APIs** | OpenWeatherMap, Mapbox, Radar overlays |
| **Database** | MongoDB / PostgreSQL |
| **Authentication (optional)** | JWT / OAuth2 |
| **Deployment** | Vercel / Netlify (frontend), Render / Railway / AWS (backend) |

**Performance & UX Considerations**

* **Caching**:
  + Cache API responses (e.g. Redis, in-memory) for rate-limited APIs

**Error handling**:

* + Graceful fallbacks on API failure
* **Loading states**:
  + Skeleton loaders while fetching weather
* **Responsive design**:
  + Works well on mobile/tablets
* **Accessibility**:
  + Keyboard navigation, contrast, ARIA labels

**Security Considerations**

* Avoid exposing API keys on the frontend
* Use a backend proxy for weather API calls
* Implement rate limiting on backend endpoints
* Sanitize user inputs in settings
* Optional: Auth system for saving preferences across devices

**Folder Structure (React + Express Example)**

weather-dashboard/

├── client/ # React frontend

│ ├── public/

│ ├── src/

│ │ ├── pages/

│ │ │ ├── Dashboard.jsx

│ │ │ ├── Forecast.jsx

│ │ │ ├── RadarMap.jsx

│ │ │ └── Settings.jsx

│ │ ├── components/

│ │ ├── App.jsx

│ │ └── index.js

├── server/ # Backend API

│ ├── routes/

│ │ ├── weather.js

│ │ └── user.js

│ ├── services/

│ ├── app.js

│ └── config.js

├── .env

├── package.json

└── README.md