COLLEGE CODE : 9238

COLLEGE NAME: Mangayarkarasi College of Engineering

DEPARTMENT: CSE

STUDENT NM-ID: 6EFDED058087CA09FB543E1463BF2088

ROLL NO.: 923823104006

DATE: 08-09-2025

Completed the project named as Phase 3 – MVP Implementation

FRONT END TECHNOLOGY

PROJECT NAME: LIVE WEATHER DASHBOARD

SUBMITTED BY,

NAME: ASMATH FOUZIYA M
MOBILE NO.: 9994164941

LIVE WEATHER DASHBOARD

Phase 3 – MVP Implementation

1. Project Setup

The project setup phase is foundational to the success of the Live Weather Dashboard MVP. A properly configured environment ensures smooth development and makes future iterations easier to manage. This phase involves selecting the technology stack, preparing the development environment, and organizing the project structure.

Choosing the Technology Stack

For the development of the Live Weather Dashboard, selecting the appropriate tech stack is crucial to ensure smooth performance, scalability, and flexibility in building the dashboard's core features. The following technologies are recommended:

1. **Frontend**: React.js is a popular JavaScript library for building user interfaces. It allows developers to create fast, interactive UIs with reusable components. React's state management capabilities will allow easy handling of dynamic weather data.
2. **Backend**: While an MVP might not require a full-fledged server-side infrastructure, it is still recommended to have a backend framework like Node.js with Express.js. This will allow handling API requests to external weather data services and processing any necessary logic before sending data to the frontend.
3. **API for Weather Data**: The OpenWeatherMap API is an excellent choice to fetch real-time weather information. It provides details on temperature, humidity, pressure, and other weather metrics. This external service will serve as the backbone for the weather data displayed on the dashboard.
4. **Database**: During the MVP stage, using **local state** (via React's built-in state management like `useState`) can be enough to hold the data temporarily during the session. However, as the product grows, incorporating a NoSQL database such as **MongoDB** or a cloud database service like **Firebase** may become necessary to store user preferences, favorites, or historical weather data.
5. **Version Control**: Git and GitHub are essential tools for managing code, collaboration, and maintaining version history. A Git repository will enable continuous updates and smooth team collaboration, and GitHub will act as the centralized location for storing and reviewing the code.

Development Environment Setup

To begin development, developers need to set up a proper local environment. Here are the steps to set it up:

1. Install **Node.js** and **npm** (Node package manager) to enable dependency management and project configuration.
2. Install **React** using `create-react-app` or manually set up Webpack if greater customization is needed for the build process.

3. Install essential dependencies like **Axios** for API calls, **React Router** for routing (if needed), and **Redux** (optional) for more complex state management.
4. Initialize a Git repository to start version control and push the project to GitHub for collaboration.

Project Structure

The project should have a simple, modular folder structure to manage components, actions, and utilities efficiently. The following folder structure is recommended:

1. **/public**: Contains the `index.html` file and static assets like icons.
2. **/src**: Holds all React components and logic.
    1. **/components**: For reusable UI elements such as WeatherCard, Header, etc.
    2. **/assets**: For images and icons used in the app.
    3. **/redux** (optional): For global state management if using Redux.
    4. **/utils**: For utility functions such as API calls.
    5. **App.js**: The main entry point of the React app.
3. **package.json**: Contains the project dependencies and scripts for running the app.

With this foundational setup, the project is ready for feature development.

2. Core Features Implementation

Once the project is set up, the next step is to implement the core features that define the MVP of the Live Weather Dashboard. These features should be minimal but functional enough to demonstrate the potential value of the product to users.

Weather Data Display

The primary function of the Live Weather Dashboard is to fetch and display real-time weather data. The core feature here is integrating the OpenWeatherMap API to retrieve weather data based on the user's input (city, location, or other parameters).

The weather data displayed on the dashboard will include:

1. **Current Temperature**: In both Celsius and Fahrenheit.
2. **Weather Condition**: A brief description such as “Clear,” “Rainy,” or “Cloudy.”
3. **Humidity and Wind Speed**: Essential weather parameters to provide a more complete view of the weather.
4. **Weather Icons**: Icons representing different weather conditions (sun, cloud, rain, etc.).

User Search Functionality

To allow users to query weather for different locations, the dashboard will feature a search bar. When users input a city name or location and submit it, an API call will be made to retrieve the weather data for that location.

This feature should be simple but responsive, updating the weather data dynamically without needing to reload the page. The user should also be able to see an error message if an invalid location is entered (e.g., "City not found").

UI/UX Design

While the MVP focuses on functionality, the user interface (UI) and user experience (UX) should still be clean and intuitive. The design should allow users to interact with the dashboard easily. Here are some key considerations for the MVP:

1. **Responsive Layout**: The dashboard should be responsive, adapting to different screen sizes such as mobile phones, tablets, and desktops.
2. **Minimalist Design**: For the MVP, the design should focus on simplicity, with a few essential UI components such as the weather display, search bar, and error messages.
3. **Easy Navigation**: The user should have no trouble navigating the dashboard. They should quickly understand how to search for weather data and view results.

Error Handling and Feedback

Error handling is a crucial aspect of ensuring a good user experience. For example, if the user enters an invalid city name, the app should return an error message that guides them to correct the input. Similarly, if the API fails or the weather data cannot be retrieved, the app should display a user-friendly error message rather than a blank screen.

3. Data Storage (Local State/Database)

For an MVP, it's important to decide how data will be stored and managed. In the case of the Live Weather Dashboard, two primary options are available: local state and database storage.

Local State Management

Since the MVP focuses on delivering real-time weather data from an external API, **local state** in the frontend is sufficient for temporarily storing weather data. React’s `useState` hook allows you to manage this state within the components.

The weather data retrieved from the Open Weather Map API will be stored in the local state, and each time a user searches for a new city, the state will be updated with the new data. This allows the app to dynamically update the UI without requiring a page reload.

For managing complex states (e.g., weather data for multiple cities), a global state management library like **Redux** can be used. Redux will help store weather data globally and allow different components to access the same state without prop drilling.

Persistent Data Storage

While local state is sufficient for most of the data during the MVP stage, additional features may require persistent storage, such as storing user preferences, historical weather data, or favorite cities.

1. **Firebase**: If you decide to add features like saving favorite locations or storing user-specific data (e.g., recent searches), **Firebase** is an excellent solution. Firebase offers easy-to-use real-time database functionality, and its Firestore database can handle these use cases without the need for a complex backend.
2. **Mongo DB** : If you need a more traditional NoSQL database and plan to scale, **Mongo DB** can be an option. It offers flexibility and scalability for storing various types of data.

State Persistence

In scenarios where the user may want to return to the dashboard later, storing data in **localStorage** or **sessionStorage** can persist the state even after the page is refreshed or the user closes the browser. This ensures a seamless experience, where the user doesn't have to re-enter their search query every time they return to the dashboard.

4. Testing Core Features

Testing is a vital part of the development process, especially when building an MVP. Ensuring the core features function as intended is necessary for the success of the Live Weather Dashboard.

Unit Testing

Unit testing involves testing individual components of the app to ensure that they behave correctly. For example, you can write tests for components like the **WeatherCard**, which displays weather data, to verify that the correct data is rendered.

Testing libraries like **Jest** and **React Testing Library** are popular tools for testing React components. These libraries allow you to simulate user interactions and verify that the app works as expected.

Integration Testing

Integration tests ensure that multiple components and parts of the application work together as expected. For example, you can test the integration between the search functionality and the weather data display. You can check if entering a city name in the search bar correctly triggers the API request and updates the UI with the new weather information.

End-to-End Testing

End-to-end (E2E) testing simulates a real user's journey through the application. Tools like **Cypress** or **Puppeteer** are used to write E2E tests, verifying the entire flow of the application, from entering a city in the search bar to viewing the updated weather information.

Manual Testing

While automated testing is essential, manual testing is also important to catch edge cases and verify the app’s overall functionality. This could involve testing the app on different devices and browsers, ensuring that it is responsive and works as expected under different network conditions.

5. Version Control (GitHub)

Version control is critical for managing code changes and collaborating with other developers.