COLLEGE CODE	:9233
COLLEGE NAME	:GOVERNMENT COLLEGE OF ENGINEERING,BODI
DEPARTMENT	: COMPUTER SCIENCE AND ENGINEERING
STUDENT NM-ID	: 35504C041D3AF39AD2A436E7DC10C4A5
ROLL NO	:23TRCS302
DATE	:26.09.2025

Completedtheprojectnamedas Phase3

TECHNOLOGYPROJECTNAME : WEATHER DASHBOARD

SUBMITTEDBY,

NAME	: PUSHPADHARANI G
MOBILENO	:7540083672

1. **Project Setup**

   A well-structured setup is crucial for building a reliable and scalable Weather Dashboard. This phase focuses on organizing the development environment, selecting the appropriate tech stack, and laying down the necessary configurations**.**

   - ***Backend Setup***
   - *Built RESTful API using Node.js/Express or Django to fetch and process weather data.*
   - *Integrated third-party weather APIs (e.g., OpenWeatherMap, WeatherAPI) for real-time weather info.*
   - ***Frontend Setup***
   - *Created a React.js project for a modular, responsive UI.*
   - *Configured routing (React Router) for Home, Location Search, Forecast Details, and Settings pages.*
   - *Used Axios to communicate with backend services and fetch weather data.*
   - *For quick prototyping, static HTML/CSS/JS pages are also available.*
   - ***Database Setup***
   - *Designed database schema with tables like:*
   - *Locations → (location_id, city_name, country, coordinates, user_id).*
   - *User Preferences → (user_id, preferred_units, favorite_locations, notification_settings).*
   - *Stored cached weather data for frequently accessed locations to reduce API calls and improve performance.*
   - ***API Documentation Setup***
   - *Integrated Swagger UI for documenting backend endpoints (weather fetch, location save, user prefs).*

- *Created a Postman collection for API testing.*
- *Environment Configuration*
- *Maintained different environments:*
- *Development → local DB, verbose logging, mock weather data.*
- *Testing → automated test mocks for APIs.*
- *Production → secure DB, optimized API keys, cloud deployment.*
- ***Version Control Setup***

  * Initialized GitHub repository with branching model

  * Configured .gitignore to exclude sensitive files like API keys.

2. Core Features Implementation

The MVP targets delivering essential weather dashboard functionality while enabling future improvements..

Weather Data Collection

- Integrates with multiple weather APIs (e.g., OpenWeatherMap, Weatherstack) to fetch real-time and forecast data.
- Validates API responses to ensure accuracy and handle errors gracefully.
- Supports location inputs by city name, GPS coordinates, or zip/postal codes.

Current Weather Display

- Shows key weather metrics: temperature, humidity, wind speed, UV index, and precipitation.
- Includes weather icons and descriptive text (e.g., “Partly Cloudy,” “Heavy Rain”).
- Automatically updates at configurable intervals (e.g., every 10 minutes).

Forecast Visualization

- Displays short-term forecasts (hourly for the next 24 hours).
- Provides extended forecasts (daily for up to 7-14 days).
- Interactive charts and graphs (temperature trends, precipitation probability).

Location Management

- Allows users to save multiple favorite locations.
- Supports automatic location detection via browser geolocation (optional).
- Users can switch between saved locations easily.

Alerts & Notifications

- Sends alerts for severe weather warnings (storms, heatwaves, frost).
- Configurable alert preferences by user location and weather conditions.
- Option for email or push notifications (optional MVP).

User Customization

- Choose temperature units (Celsius, Fahrenheit).
- Select themes for dashboard appearance (light, dark mode).
- Customize which weather parameters are shown on the main screen.

Data Caching & Offline Mode

- Caches last successful weather data to display during network outages.
- Provides approximate weather info with timestamp of last update.

Performance & API Rate Limiting

- Implements smart caching to reduce redundant API calls.
- Handles API rate limits by prioritizing critical data and queuing requests.

Admin Dashboard

- Monitors API usage and system health.
- Views user activity logs and manages alert settings.
- Configures available weather data providers and fallback options.

Security Features

- Secure API keys management on the backend.
- Rate limiting on user requests to prevent abuse.
- Data privacy compliance, no storing of sensitive user location info without consent.

3. Data Storage (Local State / Database)

Handling data efficiently is key for smooth performance and a great user experience.

- ***Frontend (Local State)***
- *React state and context store current weather and user preferences.*
- *Cached data stored in localStorage for offline access and faster reloads.*
- *Sensitive API keys never stored on the client.*
- ***Backend (Database)***

***** User data and preferences stored securely.

* Cached weather data stored with timestamps to prevent excessive API calls.

* Logs of user activity and API usage for monitoring and analytics.

4. Testing Core Features

Thorough testing ensures the app performs reliably across scenarios**.**

- ***Unit Testing***
- *Backend API endpoints tested with Mocha/Chai or Jest.*
- *Frontend React components tested with React Testing Library and Jest.*
- ***Integration Testing***
- *End-to-end tests using Cypress or Selenium covering search, favorite management, and notifications.*
- ***UI Testing***
  - Manual tests on desktop and mobile browsers for responsiveness and usability.
- ***Security Testing***

* Validation against common vulnerabilities like API key exposure or injection attacks.

5. Version Control (GitHub)

GitHub supports collaboration, version history, and deployment automation.

- *Repository Setup*
- *Repo: weather-dashboard*
- *Branching model:*

-
- *main → stable releases*
-
- *dev → active development*
-
- *feature/* → individual features like feature-map, feature-notifications*
- ***Commit Practices***
  - Frequent descriptive commits (e.g., “Added 7-day forecast charts”, “Implemented location caching”).
  - Followed conventional commit format.
- ***Pull Requests & Code Review***
  - Features merged through pull requests with peer reviews focusing on code quality and UX.
- ***CI/CD Integration***
- *GitHub Actions set up for automatic build, linting, and test runs on every push.*
- *Deployment pipelines configured for staging and production environments.*