**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 2 - Solution Design & Architecture

**FRONT END TECHNOLOGY**

**PROJECT NAME:** LIVE WEATHER DASHBOARD

**SUBMITTED BY,**

NAME: ASMATH FOUZIYA M

MOBILE NO.: 9994164941

# LIVE WEATHER DASHBOARD

# Phase 2 – Solution Design & Architecture

## 1. Tech Stack Selection

To build a scalable and responsive live weather dashboard, the following technologies are selected:

### Frontend

1. React.js for component-based UI
2. Tailwind CSS or Bootstrap for styling
3. Axios for API calls

### Backend

1. Node.js with Express for RESTful API
2. Python with FastAPI (alternative for data-heavy processing)

### API Integration

1. OpenWeatherMap or WeatherAPI for real-time weather data
2. GeoLocation API for user-based location detection

### Database

1. MongoDB for storing user preferences and logs
2. Redis for caching frequent weather queries

### Hosting & Deployment

1. Vercel or Netlify for frontend
2. Heroku or AWS EC2 for backend services

### DevOps & Monitoring

1. GitHub Actions for CI/CD
2. Docker for containerization
3. Prometheus + Grafana for performance monitoring

## 2. UI Structure & API Schema Design

### UI Structure

The dashboard is divided into intuitive sections:

1. **Header**: Contains app name, search bar, and location toggle
2. **Main Panel**: Displays current weather, temperature, humidity, wind speed
3. **Forecast Section**: Hourly and 7-day forecast with icons
4. **Sidebar Widgets**: Favorite locations, alerts, and settings
5. **Footer**: Credits, API source, and contact info

### API Schema Design

**Endpoint**: `GET /weather?location={city}`

### Some Endpoints:

1. `GET /forecast?location={city}`
2. `POST /favorites`
3. `GET /alerts?location={city}`

## 3. Data Handling Approach

Efficient data handling ensures performance and reliability:

### Fetching Strategy

1. Real-time data fetched using scheduled polling
2. Webhooks for alert updates (if supported by API provider)

### Caching

1. Redis used to cache frequent queries
2. TTL (Time to Live) set based on forecast freshness

### Storage

1. MongoDB stores user preferences, search history, and logs
2. Weather logs used for analytics and trends

## Error Handling

1. Retry mechanism for failed API calls
2. Fallback to cached data during API downtime
3. Graceful UI degradation with user notifications

## Security Measures

1. API key encryption and rotation
2. HTTPS for secure data transmission
3. Input validation and rate limiting

# 4. Component / Module Diagram

## Frontend Components

1. **`SearchBar`:** Input for city/location
2. **`WeatherDisplay`:** Shows current weather
3. **`ForecastPanel`:** Hourly and daily forecast
4. **`MapWidget`:** Optional weather map integration
5. **`SettingsPanel`:** Theme, units, and preferences

## Backend Modules

1. `WeatherController`: Handles API requests
2. `ForecastService`: Processes forecast data
3. `APIClient`: Communicates with external weather APIs
4. `CacheManager`: Manages Redis caching
5. `UserPreferencesService`: Stores and retrieves user settings

## Database Collections

1. `Users`: Stores user profiles and preferences
2. `Locations`: Saved locations and search history
3. `WeatherLogs`: Historical weather data

## 5. Basic Flow Diagram

The flow of data and interaction is as follows:

### Code

```
User → UI → Backend →
Weather API
      ↓      ↓
   Display ← Cache ← DB
```

### Explanation

1. User enters a location in the UI
2. Request is sent to backend
3. Backend checks Redis cache
4. If not found, fetches from external API
5. Response is cached and stored in DB
6. UI updates with latest data