College Code :9607

College Name : Immanuel Arasar JJ College Of Engineering

Register No :960723104022

Department :B.E(CSE)

Student NM-ID :aut2396070044

Date :26.09.2025

Completed the project named as phase 4

M.Durga devi,B.Sundhareswari,T.Sivakalidevi

TECHNOLOGY PROJECT NAME:Live Weather Dashboard

SUBMITTED BY,

Name :T.Sivakalidevi

Mobile No :7604902967

Phase 4:Enhancement & Deployment

Content:

- ❖ Additional Features
- ❖ UI/UX Improvements
- ❖ API Enhancements
- ❖ Performance&security checks
- ❖ Testing of Enhancements
- ❖ Deployment (Netlify,Vercel,or Cloud Platform)

Live weather dashboard

Additional Features:

1.Real-Time Weather Alerts

Press bad weather notifications: Thunderstorms, Hurricanes, Blizzards, etc.

Warning Severity.

2.Interactive Radar Card

- Live precipitation (rain, snow, storm)
- Layer: Temperature, Cloud Coverage, Wind Speed/Direction, Air Quality

3. 7-14 Days Prediction

- Daily forecasts and expansion times
- Graphic view of temperature, humidity and wind trends

4.Store Follow-up

- Users can record and switch between several cities
- Ideal for travelers and distant teams

Intelligent Function

5.Weather-based Recommendations

- Take an umbrella or Wear sunscreen
- Custom- sible alerts about personal thresholds

6.AI powered insight

- Natural Language Summary: Today, 3°C is colder than strong winds.
- Predict weather trends.

Extended Data:

7.Air Quality Index (AQI)

- Real-time AQI with ventilation
- AQI based health recommendations

8.Pollen and Allergen Count

- Daily Pollen Prediction for Allergies
- Specific types (tree, grass, grass)

9.UV -index and sun effect timer

- Viven UV rating with sunburn risk levels
- Timer for reverse exposure time depending on the type of skin

Context Layer:

10.Events/Calendar Integration

- Provides the best time to plan your activities
- Google/Outlook Weather Calendar

11.Data History Expression

- Compare the current weather with the previous year
- Visual trends in climate monitoring

Ecological and Satellite Data:

12.Satellite Cloud Images

- High-resolution satellite impact / NOAA
- Temporal suturing mode

13.Tidal and sea data

- Ideal for coastal areas
- Wave condition, time, tide height

User Customization

14. play

- Select your favorite indicator (for example, humidity, dew point)

15.Subject Mode

- Clear/Dark Fashion
- Weather-reactive interface (e.g., rain-like animation when it rains)

Platform Function:

16.Vocal assistant integration

- Ask Alexa/Siri/Google: "What's the weather like this weekend?"

17.Autonomic mode

- Low/Latest coverage of weather data known for no internetzone

User interface improvements:

1.Pure and minimalist layout

- Use plenty of white space to avoid obstacles. Group information for sections or clear cards.
- Uses constant printing and hierarchy.

2.Reactive Design

- Please make sure your dashboard works perfectly on mobile, tablet, and desktop computers.
- Use flexible grids or CSS cards for layout.

3.Color and visual feedback coding

- Color code temperature beach (blue if it's cold, red if it's hot).
- Use visual signals for warnings.
- Predicts the stem or radial diagram of humidity, wind speed, or UV index.

UX improvements:

1. Navigation and control elements are intuitively understandable

- Easy access to locations with automatic filling. Save your favorite locations for quick switching.
- Simple and clean button/icon for switching expressions.

2. Actual comments

- Displays the skeleton or spinner when data is received. Provides important error messages.

3. Personalization options

- Let the user select their favorite unit (celsius/Fahrenheit, km/h/mph).

4. Context information

- It shows the "sensation" of temperature at actual temperature.
- Add a short and comfortable weather resume to your natural language.

5. Accessibility

- Use HTML semantic tags.
- Provides key navigation. Illumination of the ARIA label on the screen player.
- Supports ample color contrast.

Live Weather Panel API Improvements:

1. Some data and aggregation sources

- Integrate several weather APIs.
- Total data to increase accuracy and retirement if APIs are not possible.
- Implement logic to merge or organize the priorities of the best data from sources.

2. Cache and speed limits

- Response to the customer or server cache API .
- Define the validity period for the cache depending on your data freshness needs.
- Creates the start of a research entry to reduce customer-side speed limits or API calls.

3. Party and Volume Data

- Handles weather reception for several locations of API requests .
- It's useful for "favorites" or the city's masses.

4 .WebSocket/Server-Ente (SSE) Events for Real-Time Updates

- Go from surveys for important data to actual streaming updates.
- Low latency and reduce unnecessary demands.

5. User's final point API

- Let the user configure weather preferences.
- Provides individual responses according to user settings.

6. Historical and planned data

- Add the latest points in the history of weather data.
- Integrate prediction or automated learning analytics models into predictive trends.

7. Treatment and separation of errors

- Provides clear and structured errors in the API.
- If no API calls are made, implement the backup service mechanism.

8. Location and internationalization

- Supports several languages in the API response.
- Provides individual units and formats.

9. Metadata and context information

- Add additional data such as AQI, pollen levels, UV index, sunrise/sunset.
- Attach metadata such as data sources, time brands, and confidence ratings.

10. Improves safety

- Attach the API key using environment variables and proxy.
- Limit of API usage the speed to prevent abuse.

Example: Enhanced API Request with Caching (pseudo code)

```
async function fetchWeather(location) {
  const cacheKey = `weather-${location}`;
  const cached = localStorage.getItem(cacheKey);
  const cacheTime = localStorage.getItem(`${cacheKey}-time`);

  if (cached && Date.now() - cacheTime < 10 * 60 * 1000) { // 10 min cache
    return JSON.parse(cached);
  }

  const response = await
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${location}&appid=AP
I_KEY`);
  const data = await response.json();

  localStorage.setItem(cacheKey, JSON.stringify(data));
```

```
  localStorage.setItem(`${cacheKey}-time`, Date.now());

  return data;
}
```

PERFORMANCE CHECKS:

Focus: Fast load times, smooth interactivity, efficient API usage

1. Frontend Performance

Check	What to Look For	Tools
Initial Load Time	< 2–3s	Lighthouse, WebPageTest
Bundle Size	Keep < 250 KB gzipped	Webpack Bundle Analyzer
Time to First Byte (TTFB)	< 200ms	Chrome DevTools
Lazy Loading	Load images, charts, or components only when needed	React.lazy / Vue async components
Image Optimization	Use WebP, resize, compress	Image CDN (e.g., Cloudinary)
Efficient CSS/JS	Remove unused styles/scripts	PurgeCSS, Tree shaking
CDN Usage	Use CDN for static assets	Cloudflare, Vercel edge network

2. API Performance

Check	What to Look For	Tools
Response Time	< 500ms for API calls	Postman, k6
Caching	Cache weather responses (e.g., 10 min)	SWR, React Query, service workers
Debounced Search	Avoid API overuse on typing	Lodash debounce
Retry & Fallbacks	Handle failed API calls	Axios retry, exponential backoff

SECURITY CHECKS:

Focus: Secure API keys, user data, and client-server interaction

1. API Key Protection

Check	What to Do	Notes
Do NOT expose keys in frontend	Use a backend proxy or serverless function	Avoid direct fetch from React to OpenWeatherMap with keys in code
Use environment variables	`.env.production` and server-side secrets	Never commit `.env` files

2. HTTPS & Secure Headers

Check	What to Do	Tools
HTTPS enforced	Redirect all traffic to HTTPS	Vercel/Netlify do this by default
Security headers	Add CSP, X-Content-Type-Options, Referrer-Policy	Helmet (Node.js), Netlify headers
Check for mixed content	Ensure all API/image URLs are HTTPS	Chrome DevTools

3. User Input & Client Safety

Check	What to Do
Escape user input	Sanitize search/location fields
Rate limiting	Prevent abuse of search endpoint or weather API

4. Vulnerability Scanning

Tool	What It Checks
OWASP ZAP	Web app security scan (XSS, injections, etc.)
npm audit / yarn audit	Dependency vulnerabilities
Snyk	Continuous security for dependencies

Sample Tools & Scripts:

Lighthouse (Performance + Accessibility + SEO)

```
npx lighthouse https://your-dashboard-url.com –view
```

Bundle Analysis (Webpack)

```
npm install --save-dev webpack-bundle-analyzer
```

Add to `webpack.config.js`:

```
plugins: [new BundleAnalyzerPlugin()]
```

Final Checklist Summary:

Category	Checks
Performance	Load time < 2s, API latency < 500ms
Optimization	Lazy loading, image compression, code splitting
Caching	Client-side + CDN + API caching
Security	API key hiding, HTTPS, secure headers

Testing	Lighthouse, k6, OWASP ZAP, npm audit

Types of tests to be performed:

1. Unit test

- Try small pieces of isolated logic.
- Examples:Temperature conversion function
- API formatting -Data
- Component that displays the logic
- Tools: Jest, Moka, Vhest, Test Library

2. Integration test

- Try how components or modules work.
- Examples:API weather -service + collaboration
- Search position + map update
- Dark Mode Taggle influences the theme –context

3. End-to-end test (E2E)

- It simulates the interactions of real users from onion to the backend.
- Examples:
- Search for a city, view the updated time
- Passes between the stored position.

4. Performance test

- Try how quickly the data and processes of the dashboard.
- Examples:Charging time of the page.

5. Safety test

- Make sure the app is safe for vulnerabilities.
- Examples:The API key is hidden by the Frontend.
- HTTPS and safety heads kept.

- No Vulnerable NPM package.
- Tools: Owasp Zap, NPM Audit, SNYK.
- Sample Test Cases.

Unit Tests (Jest)

```
// temperatureUtils.test.ts
import { convertToFahrenheit } from './temperatureUtils';

test('converts Celsius to Fahrenheit', () => {
  expect(convertToFahrenheit(0)).toBe(32);
});
```

Component Test (React Testing Library)

```
// WeatherCard.test.tsx
import { render, screen } from '@testing-library/react';
import WeatherCard from './WeatherCard';

test('displays current temperature', () => {
  render(<WeatherCard temp={26} condition="Sunny" />);
  expect(screen.getByText(/26°C/i)).toBeInTheDocument();
});
```

E2E Test (Cypress)

```
describe('Weather Search Flow', () => {
  it('should search and display weather for a city', () => {
    cy.visit('/');
    cy.get('input[placeholder="Search location"]').type('London');
    cy.get('button[type="submit"]').click();
    cy.contains('London').should('be.visible');
    cy.contains(/°C/).should('exist');
  });
});
```

Performance Testing (Lighthouse CLI):

```
npx lighthouse https://your-dashboard-url.com --view
```

Check for:

- First contentful paint (FCP)
- Time to interactive (TTI)
- Cumulative layout shift (CLS)
- Largest contentful paint (LCP)

Security Testing:

Run vulnerability scan

```
npm audit fix
```

Use OWASP ZAP or Snyk

- Scan for XSS, insecure headers, dependency vulnerabilities

Organize Test Files (Best Practice):

```
src/
├── components/
│   └── WeatherCard.test.tsx
├── utils/
│   └── temperatureUtils.test.ts
cypress/
├── integration/
│   └── search-weather.spec.js
tests/
├── performance/
│   └── lighthouse-report.html
```

CI/CD Integration (Optional):

Add tests to your GitHub Actions or GitLab CI pipeline:

```
# .github/workflows/test.yml
name: Run Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - uses: actions/setup-node@v3
      with:
        node-version: '18'
    - run: npm install
    - run: npm test
```

Summary: What to Test After Enhancements

Area	What to Test
New Features	Work as expected & handle errors
Performance	Load speed, API latency, bundle size
UI/UX	Responsiveness, accessibility, layout
Security	No exposed secrets, headers secure
Regression	Existing features not broken

Quick Comparison

Feature	Netlify	Vercel	Cloud Platforms (AWS/GCP/Firebase)
Setup Simplicity	Very Easy	Very Easy	Medium–Hard
CI/CD Support	Built-in	Built-in	Manual or via GitHub Actions
Custom Domains	Free + Easy	Free + Easy	Possible, more setup
Environment Vars	UI-based	UI-based	Varies (console or CLI)
Cost (free tier)	Generous	Generous	Limited / usage-based
Best For	Static Sites	React/Next.js Sites	Complex/Enterprise setups

Vercel or Netlify for a typical weather dashboard:

Deploying on Netlify

1. Connect GitHub Repo

- Go to Netlify
- Click **"Add New Site" → "Import from Git"**
- Select your GitHub repo

2. Configure Build Settings

If you're using **React**, defaults are usually:

- **Build Command**: `npm run build`
- **Publish Directory**: `build/`

For **Vite**, use:

- **Build Command**: `npm run build`
- **Publish Directory**: `dist/`

3. Set Environment Variables

Go to **Site Settings → Environment Variables** and add:

```
REACT_APP_WEATHER_API_KEY=your_api_key
REACT_APP_BASE_URL=https://api.openweathermap.org
```

4. Deploy

Click **Deploy Site**, Netlify will:

- Pull your repo
- Run the build command
- Host it at `https://your-project-name.netlify.app`

Deploying on Vercel:

1. Connect GitHub Repo”

- Go to Vercel
- Click **"New Project"**, import your repo

2. Set Framework Preset

- Vercel detects React, Next.js, Vue, etc.
- Confirm **Build Command** and **Output Directory** (e.g., `build` or `dist`)

3. Add Environment Variables

- Go to **Project Settings → Environment Variables**

```
REACT_APP_WEATHER_API_KEY=your_api_key
REACT_APP_BASE_URL=https://api.openweathermap.org
```

4. Deploy

Click **Deploy**, and Vercel will:

- Automatically build your project
- Host it on a Vercel domain: `https://your-dashboard.vercel.app`

You also get:

- Preview URLs for pull requests
- -based deployments

Deploying on a Cloud Platform:

Firebase Hosting

Best for simple React/Vue apps

Steps:

1. **Install CLI**

```
npm install -g firebase-tools
```

2. **Login & Init**

```
firebase login
firebase init
```

- Select **Hosting**
- Set build directory (`build/` for CRA, `dist/` for Vite)

3. **Deploy**

```
npm run build

firebase deploy
```

AWS (S3 + CloudFront)

Best for production-grade static hosting with CDN

Steps:

1. Build your app:

```
npm run build
```

2. Upload to S3 bucket (static website enabled)
3. Use **CloudFront** for CDN + HTTPS
4. Manage DNS via **Route 53**

Tools: AWS CLI, Terraform (for automation)

Google Cloud (App Engine)

Best for dynamic apps or backend integration

Deployment Checklist:

Item	Check
Environment Variables	Set securely
`.env` in `.gitignore`	Don't expose API keys
CORS configuration	API supports frontend
Build succeeds locally	`npm run build` works
Fallback routing	For React Router (`_redirects` or rewrites)
HTTPS	Automatic with Netlify/Vercel

Bonus: Offline Mode / PWA (Optional):

- Add service worker for offline weather viewing
- Use `workbox` or CRA's built-in PWA support

Want hands-on help?

I can:

- Generate a `netlify.toml` or `vercel.json` config
- Set up Firebase CLI scripts
- Create custom `.env` and deployment workflow for GitHub Actions

GitHub:

https://github.com/sivakalidevi/T.sivakali