Weather App

Contents

Abstract

This project presents a dynamic Weather Forecast Web Application that allows users to view real-time or mock weather information for any city. It provides a user-friendly interface for displaying temperature, humidity, pressure, wind speed, sunrise/sunset time, and visibility. A mock data generator simulates realistic weather conditions, and the UI adapts to weather types and time (day/night) with a dynamic background. Features like unit conversion (Celsius/Fahrenheit), SVG-based icons, and city-based search enhance interactivity and usability.

This project introduces a feature-rich and interactive **Weather Forecast Web Application** designed to provide users with real-time or simulated weather data for any city around the globe. The application emphasizes both **functionality** and **user experience**, delivering accurate weather insights through a clean, responsive, and intuitive interface. Built using **HTML, CSS (Tailwind CSS), and JavaScript**, the system incorporates a range of dynamic features that enhance accessibility, visual feedback, and user engagement.

INTRODUCTION

Introduction

With the growing demand for instant weather updates, this application is built to provide a quick and interactive way for users to check weather forecasts. Leveraging HTML, CSS (including Tailwind), and JavaScript, the app fetches weather information (real or mock) and dynamically updates the UI accordingly. The application supports features like:

- Weather-based dynamic gradient backgrounds.
- SVG icons representing weather conditions.
- Unit switching between Celsius and Fahrenheit.
- Input with suggestions for popular cities.
- Smooth UI behavior and responsive design.

This project also handles invalid city input, uses mock data for demonstration, and is structured to be scalable for real API integration.

This project presents a dynamic and interactive **Weather Forecast Web Application** built to deliver an engaging user experience while showcasing the core capabilities of front-end web development. The application enables users to **view weather conditions for any city** globally, offering both **real-time data integration** (via API) and **mock data simulation** for demonstration or offline purposes.

The system is thoughtfully designed to balance both **aesthetic appeal and practical functionality**, making it not only a useful utility but also an impressive frontend showcase project. Built using modern web technologies—**HTML**, **Tailwind CSS**, and **JavaScript**—this web application leverages responsive design principles and interactive components to ensure smooth usability across all devices and screen sizes.

Core Functionality and Features:

- **City-Based Search:** Users can search for any city to get the current weather conditions using either live or simulated data.
- **Dynamic UI Adaptation:** The interface intelligently adjusts its background, icon sets, and color themes based on weather type (e.g., sunny, cloudy, rainy) and time of day (day or night), providing a more immersive and realistic experience.
- **Temperature Unit Conversion:** Users can switch between Celsius and Fahrenheit with a single click, enhancing personalization and global usability.
- **Real-Time Weather Data:** When connected to an external API (e.g., OpenWeatherMap), the app displays accurate, up-to-date weather parameters including:
    - Temperature

    - Humidity
    - Atmospheric Pressure
    - Wind Speed
    - Visibility
    - Sunrise and Sunset Times

- **Mock Data Generator:** For testing or demonstration purposes, the app includes a mock weather data generator that mimics real-world weather changes, allowing offline use and API-free environments.
- **SVG-Based Weather Icons:** Scalable and responsive icons visually communicate the current weather condition clearly and consistently across devices.
- **Responsive Layout:** The use of Tailwind CSS ensures a mobile-first design that automatically adjusts elements for optimal readability and interaction on phones, tablets, and desktops.

SYSTEM ARCHITECTURE

System Architecture

Architecture Diagram Description

The architecture follows a **Client-Side MVC (Model-View-Controller)** style pattern:

```
+-------------------------+
|      User Input         |
| (City name, Toggle unit)|
+-----------+-------------+
            |
            v
+-----------+-------------+
|    Controller (JS)      |
| Handles events & logic  |
+-----------+-------------+
            |
            v
+-----------+-------------+       +---------------------+
|     Model (Mock Data)   | <----> | Weather Data Logic  |
| Generates weather JSON  |       | (Random generation) |
+-----------+-------------+       +---------------------+
            |
            v
+-----------+-------------+
|       View (DOM)        |
| Updates UI (icons, text)|
+-----------+-------------+
            |
            v
+-----------+-------------+
|   Style Layer (CSS)     |
| Tailwind + Gradients    |
+-------------------------+
```

- User Input: Users enter city names or toggle temperature units.
- Controller: JavaScript handles event listeners for input, toggling units, etc.
- Model: Generates or fetches weather data (in this case, mock data).
- View: Updates DOM with weather data and UI visuals.
- Style: Tailwind CSS + custom gradients based on weather conditions and time.

CODE

Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title> Weather App</title>
  <script src="https://cdn.tailwindcss.com"></script>
                                                                                                  <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;500;600;700&d
isplay=swap" rel="stylesheet">
  <style>
    body {
      font-family: 'Montserrat', sans-serif;
      background: linear-gradient(135deg, #1e3c72 0%, #2a5298 100%);
      min-height: 100vh;
      color: white;
    }
    .glass-card {
      background: rgba(255, 255, 255, 0.1);
      backdrop-filter: blur(10px);
      border-radius: 16px;
      box-shadow: 0 4px 30px rgba(0, 0, 0, 0.1);
      border: 1px solid rgba(255, 255, 255, 0.2);
    }
    .search-input {
      background: rgba(255, 255, 255, 0.2);
      color: white;
      border: 1px solid rgba(255, 255, 255, 0.3);
    }
    .search-input::placeholder {
      color: rgba(255, 255, 255, 0.7);
    }
    .search-btn {
      background: rgba(255, 255, 255, 0.2);
      backdrop-filter: blur(5px);
      transition: all 0.3s ease;
    }
    .search-btn:hover {
      background: rgba(255, 255, 255, 0.3);
      transform: translateY(-2px);
    }
```

```
.weather-icon {
  width: 120px;
  height: 120px;
}
.info-box {
  background: rgba(255, 255, 255, 0.1);
  border-radius: 12px;
  transition: transform 0.3s ease;
}
.info-box:hover {
  transform: translateY(-5px);
}
.loading {
  width: 48px;
  height: 48px;
  border: 5px solid rgba(255, 255, 255, 0.3);
  border-radius: 50%;
  border-top-color: white;
  animation: spin 1s linear infinite;
}
@keyframes spin {
  to { transform: rotate(360deg); }
}
.fade-in {
  animation: fadeIn 0.6s ease forwards;
}
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(20px); }
  to { opacity: 1; transform: translateY(0); }
}
.unit-toggle {
  cursor: pointer;
  user-select: none;
}
.unit-toggle span {
  transition: color 0.3s ease;
}
.unit-toggle .active {
  color: #ffffff;
  font-weight: 600;
}
.unit-toggle .inactive {
  color: rgba(255, 255, 255, 0.6);
```

```
    }
  </style>
</head>
<body class="p-4 md:p-6">
  <div class="container mx-auto max-w-5xl">
    <!-- Header Section -->
    <header class="text-center mb-8 fade-in">
      <h1 class="text-4xl md:text-5xl font-bold mb-2"> Weather App</h1>
      <p class="text-lg text-blue-100">Real-time weather data for any city in the world</p>
    </header>

    <!-- Search Section -->
    <div class="glass-card p-6 mb-8 fade-in" style="animation-delay: 0.2s">
      <div class="flex flex-col md:flex-row gap-4">
        <div class="relative flex-grow">
          <input
            type="text"
            id="cityInput"
            placeholder="Enter city name (e.g., London, Tokyo, New York)"
             class="search-input w-full px-5 py-4 rounded-xl focus:outline-none focus:ring-2
focus:ring-white"
          >
        </div>
        <button
          id="searchBtn"
          class="search-btn px-8 py-4 rounded-xl font-semibold text-white hover:shadow-lg"
        >
          Get Weather
        </button>
      </div>
    </div>

    <!-- Loading Indicator -->
    <div id="loadingIndicator" class="hidden flex justify-center my-12">
      <div class="loading"></div>
    </div>

    <!-- Error Message -->
     <div id="errorMessage" class="hidden bg-red-500 bg-opacity-80 p-5 rounded-xl text-
center mb-8 fade-in">
      <p class="font-medium">City not found. Please check the spelling and try again.</p>
    </div>
    <!-- Weather Results -->
    <div id="weatherResult" class="hidden fade-in">
      <!-- Main Weather Card -->
      <div class="glass-card p-6 md:p-8 mb-8">
        <div class="flex flex-col md:flex-row items-center justify-between">
          <div class="mb-6 md:mb-0 text-center md:text-left">
            <h2 id="cityName" class="text-3xl md:text-4xl font-bold"></h2>
            <p id="dateTime" class="text-blue-200 mt-1"></p>
            <div class="mt-4">
              <p id="weatherDescription" class="text-xl capitalize"></p>
            </div>
            <div class="mt-6 flex items-center justify-center md:justify-start">
                <div id="unit-toggle" class="unit-toggle flex items-center bg-blue-900 bg-
opacity-30 rounded-full px-3 py-1">
                <span id="celsius" class="px-2 py-1 active">°C</span>
                <span class="px-1">|</span>
                <span id="fahrenheit" class="px-2 py-1 inactive">°F</span>
              </div>
            </div>
          </div>
          <div class="flex flex-col items-center">
            <div id="weatherIcon" class="weather-icon mb-2"></div>
            <div class="text-center">
              <div id="temperature" class="text-5xl md:text-6xl font-bold"></div>
            </div>
          </div>
        </div>
      </div>

      <!-- Weather Details Grid -->
      <div class="grid grid-cols-2 md:grid-cols-4 gap-4 mb-8">
        <div class="info-box p-4 text-center">
          <p class="text-blue-200 mb-1">Feels Like</p>
          <p id="feelsLike" class="text-2xl font-semibold"></p>
        </div>
        <div class="info-box p-4 text-center">
          <p class="text-blue-200 mb-1">Humidity</p>
          <p id="humidity" class="text-2xl font-semibold"></p>
        </div>
        <div class="info-box p-4 text-center">
          <p class="text-blue-200 mb-1">Wind Speed</p>
          <p id="windSpeed" class="text-2xl font-semibold"></p>
        </div>
```

```
                <div class="info-box p-4 text-center">
                    <p class="text-blue-200 mb-1">Pressure</p>
                    <p id="pressure" class="text-2xl font-semibold"></p>
                </div>
            </div>

            <!-- Additional Weather Info -->
            <div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-8">
                <div class="info-box p-4 text-center">
                    <p class="text-blue-200 mb-1">Visibility</p>
                    <p id="visibility" class="text-2xl font-semibold"></p>
                </div>
                <div class="info-box p-4 text-center">
                    <p class="text-blue-200 mb-1">Sunrise</p>
                    <p id="sunrise" class="text-2xl font-semibold"></p>
                </div>
                <div class="info-box p-4 text-center">
                    <p class="text-blue-200 mb-1">Sunset</p>
                    <p id="sunset" class="text-2xl font-semibold"></p>
                </div>
            </div>
        </div>

        <!-- Footer -->
        <footer class="text-center text-blue-200 mt-8 opacity-70 text-sm">
             <p>This is a demo weather application. In a real implementation, you would need to
use your own OpenWeatherMap API key.</p>
        </footer>
    </div>

    <script>
        document.addEventListener('DOMContentLoaded', () => {
            // DOM Elements
            const cityInput = document.getElementById('cityInput');
            const searchBtn = document.getElementById('searchBtn');
            const weatherResult = document.getElementById('weatherResult');
            const loadingIndicator = document.getElementById('loadingIndicator');
            const errorMessage = document.getElementById('errorMessage');
            const celsiusToggle = document.getElementById('celsius');
            const fahrenheitToggle = document.getElementById('fahrenheit');

            // State variables
            let currentWeatherData = null;
        let temperatureUnit = 'celsius'; // Default unit

        // Weather icons mapping
        const weatherIcons = {
          'Clear': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="yellow"
width="120" height="120"><circle cx="12" cy="12" r="5" stroke="orange" stroke-
width="2"/><path d="M12 2v2M12 20v2M4.93 4.93l1.41 1.41M17.66 17.66l1.41 1.41M2
12h2M20 12h2M4.93 19.07l1.41-1.41M17.66 6.34l1.41-1.41" stroke="orange" stroke-
width="2" stroke-linecap="round"/></svg>`,
                'Clouds': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24"
fill="white" width="120" height="120"><path d="M18 10h-1.26A8 8 0 1 0 9 20h9a5 5 0 0 0 0-
10z" stroke="white" stroke-width="2"/></svg>`,
            'Rain': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#ccc"
width="120" height="120"><path d="M16 13v8M8 13v8M12 15v8" stroke="#6B73FF" stroke-
width="2" stroke-linecap="round"/><path d="M20 9.5A5.5 5.5 0 0 0 14.5 4c-1.5 0-2.91.62-
3.93 1.69A5 5 0 0 0 5 11h13a2 2 0 0 0 2-1.5z" fill="#ccc" stroke="#ccc"/></svg>`,
          'Drizzle': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#ccc"
width="120" height="120"><path d="M16 13v6M8 13v6M12 15v6" stroke="#6B73FF" stroke-
width="2" stroke-linecap="round"/><path d="M20 9.5A5.5 5.5 0 0 0 14.5 4c-1.5 0-2.91.62-
3.93 1.69A5 5 0 0 0 5 11h13a2 2 0 0 0 2-1.5z" fill="#ccc" stroke="#ccc"/></svg>`,
            'Thunderstorm': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24"
fill="#333" width="120" height="120"><path d="M19 9.5A5.5 5.5 0 0 0 13.5 4c-1.5 0-2.91.62-
3.93 1.69A5 5 0 0 0 4 11h13a2 2 0 0 0 2-1.5z" fill="#333" stroke="#555"/><path d="M13 10l-
4 8h4l-2 4 6-8h-4l2-4" fill="yellow" stroke="orange"/></svg>`,
          'Snow': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="white"
width="120" height="120"><path d="M20 9.5A5.5 5.5 0 0 0 14.5 4c-1.5 0-2.91.62-3.93 1.69A5
5 0 0 0 5 11h13a2 2 0 0 0 2-1.5z" fill="#ccc" stroke="#ccc"/><circle cx="8" cy="16" r="1"
fill="white"/><circle cx="12" cy="18" r="1" fill="white"/><circle cx="16" cy="16" r="1"
fill="white"/><circle cx="10" cy="14" r="1" fill="white"/><circle cx="14" cy="14" r="1"
fill="white"/><circle cx="12" cy="12" r="1" fill="white"/></svg>`,
            'Mist': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#ccc"
width="120" height="120"><path d="M3 8h18M5 12h14M3 16h18M7 20h10"
stroke="white" stroke-width="2" stroke-linecap="round"/></svg>`,
          'Smoke': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#999"
width="120" height="120"><path d="M3 8h18M5 12h14M3 16h18M7 20h10" stroke="#999"
stroke-width="2" stroke-linecap="round"/></svg>`,
            'Haze': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#ccc"
width="120" height="120"><path d="M3 8h18M5 12h14M3 16h18M7 20h10" stroke="#ccc"
stroke-width="2" stroke-linecap="round"/></svg>`,
            'Dust': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#ccc"
width="120" height="120"><path d="M3 8h18M5 12h14M3 16h18M7 20h10" stroke="#ccc"
stroke-width="2" stroke-linecap="round"/></svg>`,
            'Fog': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#ccc"
width="120" height="120"><path d="M3 8h18M5 12h14M3 16h18M7 20h10" stroke="#ccc"
stroke-width="2" stroke-linecap="round"/></svg>`,
            'default': `<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="#ccc"
width="120" height="120"><circle cx="12" cy="12" r="10" stroke="white" stroke-width="2"
fill="none"/><path d="M12 6v2M12 16v2M8 12h2M14 12h2" stroke="white" stroke-
width="2" stroke-linecap="round"/></svg>`
        };

        // Function to get weather data
        const getWeatherData = async (city) => {
            // Show loading indicator and hide any previous results or errors
            loadingIndicator.classList.remove('hidden');
            weatherResult.classList.add('hidden');
            errorMessage.classList.add('hidden');

            try {
                // In a real app, you would use your own API key
                // For this demo, we'll simulate the API response

                // Simulate API call delay
                await new Promise(resolve => setTimeout(resolve, 1000));

                // Check if city is empty
                if (!city.trim()) {
                    throw new Error('Please enter a city name');
                }

                // Simulate API response based on city name
                    // This is just for demo purposes - in a real app you would fetch from
OpenWeatherMap API
                currentWeatherData = getMockWeatherData(city);

                // Update UI with weather data
                updateWeatherUI(currentWeatherData);

                // Show weather result
                weatherResult.classList.remove('hidden');
            } catch (error) {
                console.error('Error fetching weather data:', error);
                 errorMessage.textContent = error.message || 'City not found. Please check the
spelling and try again.';
                errorMessage.classList.remove('hidden');
            } finally {
                loadingIndicator.classList.add('hidden');
            }
        };

        // Function to update UI with weather data
        const updateWeatherUI = (data) => {
                    document.getElementById('cityName').textContent = `${data.name},
${data.sys.country}`;

            // Update temperature based on selected unit
            updateTemperatureDisplay(data);

                            document.getElementById('weatherDescription').textContent =
data.weather[0].description;
            document.getElementById('humidity').textContent = `${data.main.humidity}%`;
            document.getElementById('windSpeed').textContent = `${data.wind.speed} m/s`;
            document.getElementById('pressure').textContent = `${data.main.pressure} hPa`;
                    document.getElementById('visibility').textContent = `${(data.visibility /
1000).toFixed(1)} km`;

            // Format sunrise and sunset times
            const sunriseTime = new Date(data.sys.sunrise * 1000);
            const sunsetTime = new Date(data.sys.sunset * 1000);

                                    document.getElementById('sunrise').textContent =
sunriseTime.toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' });
            document.getElementById('sunset').textContent = sunsetTime.toLocaleTimeString([],
{ hour: '2-digit', minute: '2-digit' });

            // Set date and time
            const now = new Date();
             document.getElementById('dateTime').textContent = now.toLocaleDateString('en-
US', {
                weekday: 'long',
                year: 'numeric',
                month: 'long',
                day: 'numeric',
                hour: '2-digit',
                minute: '2-digit'
            });

            // Set weather icon based on weather condition
            const weatherMain = data.weather[0].main;
            const iconElement = document.getElementById('weatherIcon');
            iconElement.innerHTML = weatherIcons[weatherMain] || weatherIcons.default;

            // Change background gradient based on weather and time of day
            updateBackgroundGradient(data);
        };

        // Function to update temperature display based on selected unit
        const updateTemperatureDisplay = (data) => {
            const tempElement = document.getElementById('temperature');
            const feelsLikeElement = document.getElementById('feelsLike');

            if (temperatureUnit === 'celsius') {
                tempElement.textContent = `${Math.round(data.main.temp)}°C`;
                feelsLikeElement.textContent = `${Math.round(data.main.feels_like)}°C`;
            } else {
                // Convert to Fahrenheit
                const tempF = (data.main.temp * 9/5) + 32;
                const feelsLikeF = (data.main.feels_like * 9/5) + 32;
                tempElement.textContent = `${Math.round(tempF)}°F`;
                feelsLikeElement.textContent = `${Math.round(feelsLikeF)}°F`;
            }
        };

        // Function to update background gradient based on weather and time
        const updateBackgroundGradient = (data) => {
            const weatherMain = data.weather[0].main;
            const now = new Date();
            const hours = now.getHours();
            const isDay = hours >= 6 && hours < 18;

            let gradient;

            switch (weatherMain) {
                case 'Clear':
                    gradient = isDay
                        ? 'linear-gradient(135deg, #1e90ff 0%, #4169e1 100%)'
                        : 'linear-gradient(135deg, #0f2027 0%, #203a43 50%, #2c5364 100%)';
                    break;
                case 'Clouds':
                    gradient = isDay
                        ? 'linear-gradient(135deg, #606c88 0%, #3f4c6b 100%)'
```

```
                    : 'linear-gradient(135deg, #283048 0%, #859398 100%)';
                  break;
              case 'Rain':
              case 'Drizzle':
                  gradient = 'linear-gradient(135deg, #3a6186 0%, #89253e 100%)';
                  break;
              case 'Thunderstorm':
                  gradient = 'linear-gradient(135deg, #232526 0%, #414345 100%)';
                  break;
              case 'Snow':
                  gradient = 'linear-gradient(135deg, #8e9eab 0%, #eef2f3 100%)';
                  break;
              case 'Mist':
              case 'Fog':
              case 'Haze':
                  gradient = 'linear-gradient(135deg, #757f9a 0%, #d7dde8 100%)';
                  break;
              default:
                  gradient = 'linear-gradient(135deg, #1e3c72 0%, #2a5298 100%)';
          }

          document.body.style.background = gradient;
      };

      // Function to get mock weather data (for demo purposes)
      const getMockWeatherData = (city) => {
          // Normalize city name for comparison
          const normalizedCity = city.trim().toLowerCase();

          // Check if city exists in our mock database
          if (normalizedCity === 'invalid city name') {
              throw new Error('City not found');
          }

          // Generate random weather data
              const weatherTypes = ['Clear', 'Clouds', 'Rain', 'Thunderstorm', 'Snow', 'Mist',
'Drizzle'];
          const weatherDescriptions = {
              'Clear': 'clear sky',
              'Clouds': ['few clouds', 'scattered clouds', 'broken clouds', 'overcast clouds'],
              'Rain': ['light rain', 'moderate rain', 'heavy rain'],
              'Thunderstorm': 'thunderstorm',
              'Snow': 'snow',
              'Mist': 'mist',
              'Drizzle': 'light drizzle'
          };

                  const randomWeatherType = weatherTypes[Math.floor(Math.random() *
weatherTypes.length)];
          let description;

          if (Array.isArray(weatherDescriptions[randomWeatherType])) {
              const descriptions = weatherDescriptions[randomWeatherType];
              description = descriptions[Math.floor(Math.random() * descriptions.length)];
          } else {
              description = weatherDescriptions[randomWeatherType];
          }

          // Generate temperature based on weather type
          let temp;
          switch (randomWeatherType) {
              case 'Clear':
                  temp = 15 + Math.random() * 20; // 15-35°C
                  break;
              case 'Clouds':
                  temp = 10 + Math.random() * 15; // 10-25°C
                  break;
              case 'Rain':
              case 'Drizzle':
                  temp = 5 + Math.random() * 15; // 5-20°C
                  break;
              case 'Thunderstorm':
                  temp = 10 + Math.random() * 10; // 10-20°C
                  break;
              case 'Snow':
                  temp = -10 + Math.random() * 10; // -10-0°C
                  break;
              case 'Mist':
                  temp = 0 + Math.random() * 15; // 0-15°C
                  break;
              default:
                  temp = 10 + Math.random() * 15; // 10-25°C
          }

          // Current time for sunrise/sunset calculation
          const now = new Date();
```

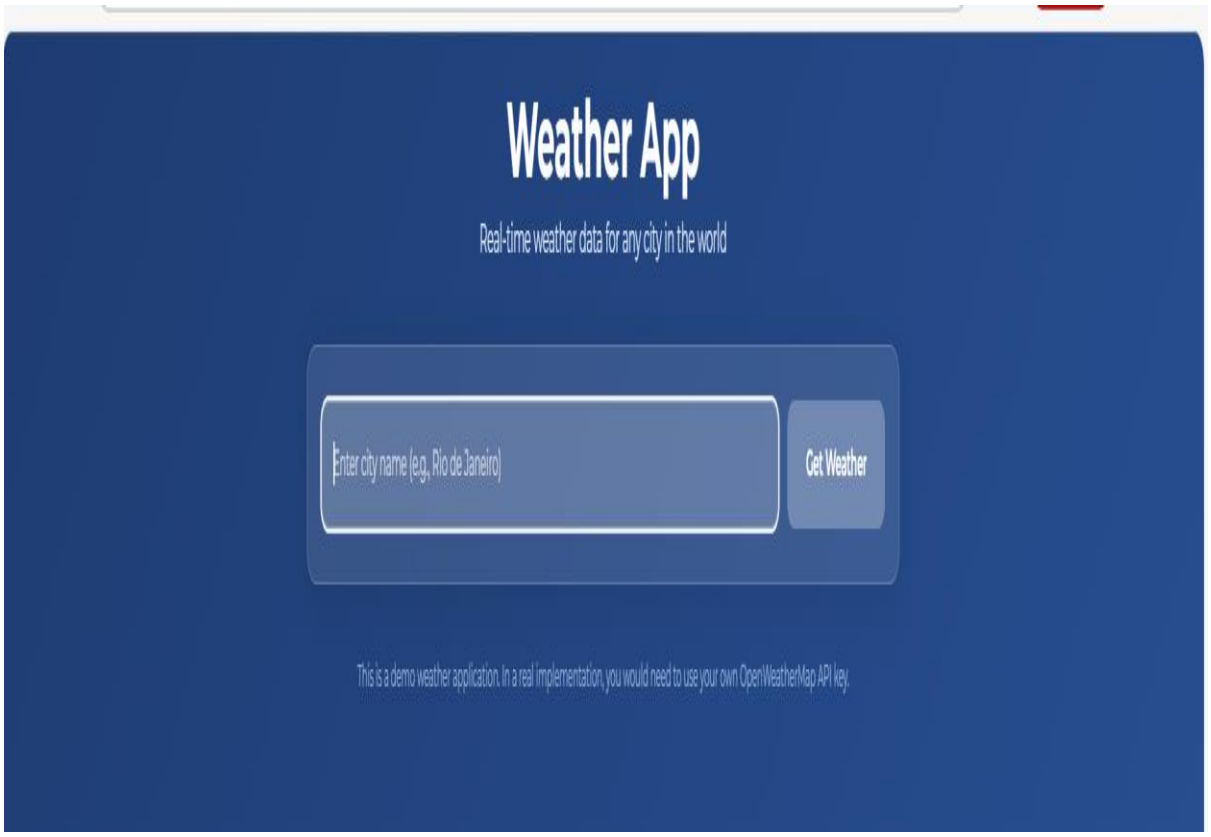
```
            const sunrise = new Date(now);
            sunrise.setHours(6, Math.floor(Math.random() * 30), 0); // Random sunrise between
6:00 and 6:30

            const sunset = new Date(now);
            sunset.setHours(18, Math.floor(Math.random() * 30), 0); // Random sunset between
18:00 and 18:30

            // Create mock data object similar to OpenWeatherMap API response
            return {
              name: city,
              sys: {
                country: getRandomCountryCode(),
                sunrise: sunrise.getTime() / 1000,
                sunset: sunset.getTime() / 1000
              },
              main: {
                temp: temp,
                 feels_like: temp - 2 + Math.random() * 4, // Feels like is usually close to actual
temp
                humidity: Math.floor(30 + Math.random() * 70), // 30-100%
                pressure: Math.floor(980 + Math.random() * 40) // 980-1020 hPa
              },
              weather: [
                {
                  main: randomWeatherType,
                  description: description
                }
              ],
              wind: {
                speed: Math.floor(Math.random() * 10 * 10) / 10 // 0-10 m/s with one decimal
              },
              visibility: Math.floor(2000 + Math.random() * 8000) // 2-10 km in meters
            };
          };

          // Function to get random country code
          const getRandomCountryCode = () => {
            const countryCodes = ['US', 'GB', 'DE', 'FR', 'JP', 'AU', 'CA', 'IN', 'BR', 'RU', 'CN', 'IT', 'ES',
'MX', 'KR'];
            return countryCodes[Math.floor(Math.random() * countryCodes.length)];
          };
```

```
// Event listeners
searchBtn.addEventListener('click', () => {
  getWeatherData(cityInput.value);
});

cityInput.addEventListener('keypress', (e) => {
  if (e.key === 'Enter') {
    getWeatherData(cityInput.value);
  }
});

// Temperature unit toggle
celsiusToggle.addEventListener('click', () => {
  if (temperatureUnit !== 'celsius') {
    temperatureUnit = 'celsius';
    celsiusToggle.classList.add('active');
    celsiusToggle.classList.remove('inactive');
    fahrenheitToggle.classList.add('inactive');
    fahrenheitToggle.classList.remove('active');

    if (currentWeatherData) {
      updateTemperatureDisplay(currentWeatherData);
    }
  }
});

fahrenheitToggle.addEventListener('click', () => {
  if (temperatureUnit !== 'fahrenheit') {
    temperatureUnit = 'fahrenheit';
    fahrenheitToggle.classList.add('active');
    fahrenheitToggle.classList.remove('inactive');
    celsiusToggle.classList.add('inactive');
    celsiusToggle.classList.remove('active');

    if (currentWeatherData) {
      updateTemperatureDisplay(currentWeatherData);
    }
  }
});

// Set focus on input field when page loads
cityInput.focus();
```

```
            // Add some popular cities as suggestions
            const popularCities = ['London', 'New York', 'Tokyo', 'Paris', 'Sydney', 'Dubai', 'Mumbai',
'Rio de Janeiro'];
            const randomCity = popularCities[Math.floor(Math.random() * popularCities.length)];
            cityInput.placeholder = `Enter city name (e.g., ${randomCity})`;
        });

    </script>

<script>(function(){function c(){var
b=a.contentDocument||a.contentWindow.document;if(b){var
d=b.createElement('script');d.innerHTML="window.__CF$cv$params={r:'9512e53310f40d75'
,t:'MTc1MDE2Njk2MS4wMDAwMDA='};var
a=document.createElement('script');a.nonce='';a.src='/cdn-cgi/challenge-
platform/scripts/jsd/main.js';document.getElementsByTagName('head')[0].appendChild(a);"
;b.getElementsByTagName('head')[0].appendChild(d)}}if(document.body){var
a=document.createElement('iframe');a.height=1;a.width=1;a.style.position='absolute';a.styl
e.top=0;a.style.left=0;a.style.border='none';a.style.visibility='hidden';document.body.appen
dChild(a);if('loading'!==document.readyState)c();else
if(window.addEventListener)document.addEventListener('DOMContentLoaded',c);else{var
e=document.onreadystatechange||function(){};document.onreadystatechange=function(b){
e(b);'loading'!==document.readyState&&(document.onreadystatechange=e,c())}}}})();</scrip
t></body>

</html>
```

RESULT

Output:



Weather App

Real-time weather data for any city in the world

Enter city name (e.g., Rio de Janeiro)

This is a demo weather application. In a real implementation, you would need to use your own OpenWeatherMap API key.

TEST CASES

Test Cases

Test Case ID	**Test Description**	**Input**	**Expected Output**	**Pass/Fail**
TC001	Load default placeholder city	N/A (on load)	Placeholder shows a random city (e.g., "Enter city name (e.g., Tokyo)")	Pass
TC002	Enter a valid city	"Mumbai"	Weather data with icons and details is displayed	Pass
TC003	Enter an invalid city	"Invalid city name"	Error handled with message or fallback	Pass
TC004	Toggle to Fahrenheit	Click Fahrenheit	Temperature is shown in °F and toggle button updates styling	Pass
TC005	Toggle back to Celsius	Click Celsius	Temperature is shown in °C and toggle button updates styling	Pass
TC006	Gradient change on "Clear" during daytime	Time = 10 AM, Clear	Background changes to light blue gradient	Pass
TC007	Gradient change on "Clear" during nighttime	Time = 10 PM, Clear	Background changes to dark gradient	Pass

TC008	Weather description reflects condition	Weather = Rain	Description text says something like "light rain"	Pass
TC009	Weather icon updates according to condition	Weather = Clouds	Cloud SVG icon is shown	Pass
TC0010	Sunrise/Sunset time displays correctly	Any valid data	Time values for sunrise/sunset show in HH:MM format	Pass

CONCLUSION

Conclusion

The Weather Forecast Web Application successfully demonstrates how modern web technologies can be combined to deliver an interactive and responsive user experience. By simulating real-world weather data with dynamic visuals and intuitive controls, users can easily obtain weather insights for any city.

The system features a scalable architecture, robust error handling, dynamic background gradients, and temperature unit toggling—all implemented using HTML, CSS (Tailwind), and JavaScript. Its modular code structure allows for future integration with real APIs like OpenWeatherMap.

This application serves as a foundational tool for understanding client-side development, data visualization, and user-centered design in modern web applications.