🌦️ Weather Forecast Project

🗀 GitHub: irfan-khan12/Weather-Forecast

Phase 1: Problem Understanding & Industry Analysis

Objective:

Weather plays a critical role in planning across multiple domains like agriculture, tourism, logistics, and daily life. The aim of this project is to build a **Salesforce-integrated weather application** that fetches **live weather data** using external APIs and presents it to the user within the Salesforce platform.

Industry Relevance:

- **Agriculture**: Timely forecasts help plan irrigation and harvest.
- **Logistics**: Avoid delays caused by bad weather.
- **Travel**: Helps tourists and travelers plan trips.
- **Urban planning**: Helps in data-driven decisions for city events or infrastructure.

Phase 2: Org Setup & Configuration

Salesforce Setup:

- Salesforce Developer Edition Org created.
- Remote Site Settings enabled to allow API callouts.
- **Named Credentials** configured for secure authentication.
- **Custom Tabs** and **Lightning Pages** created for UI.

Configuration:

- OpenWeatherMap API key stored securely.
- Enabled necessary permissions for Apex callouts.

Phase 3: Data Modeling & Relationships

Custom Objects:

- City__c: Stores user-input locations.
- Weather_Report__c: Stores response from the weather API.

Relationships:

- **Lookup relationship**: Weather_Report__c looks up to City__c, allowing multiple weather reports per city.

Sample Object Fields:

City__c

- Name (Text)

- Country__c (Text)

Weather_Report__c

- Temperature__c (Number)

- Humidity__c (Percent)

- Weather_Condition__c (Text)

- Date_Time__c (Date/Time)

Phase 4: Process Automation (Admin)

What We Automated:

- A **flow** triggers when a user creates a City__c record, calling an **Apex class** to fetch weather.
- Scheduled flow to refresh weather every 3 or 6 hours.

Flow Logic:

1. User enters city → triggers Flow
2. Flow calls Apex Action (getWeather)
3. Response saved into Weather_Report__c

Example (Flow Call):

```
@InvocableMethod
public static void fetchWeather(List<Id> cityIds) {
  // Call the Weather API and update Weather_Report__c
}
```

Phase 5: Apex Programming (Developer)

Key Classes:

1. WeatherAPIController.cls

```
public with sharing class WeatherAPIController {
    public static String API_KEY = 'Your_API_Key';

    public static WeatherData fetchWeather(String cityName) {
        String endpoint = 'https://api.openweathermap.org/data/2.5/weather?q=' + cityName + '&appid=' + API_KEY + '&units=metric';

        HttpRequest req = new HttpRequest();
        req.setEndpoint(endpoint);
        req.setMethod('GET');

        Http http = new Http();
        HttpResponse res = http.send(req);

        if(res.getStatusCode() == 200) {
            return (WeatherData)JSON.deserialize(res.getBody(), WeatherData.class);
        } else {
            throw new CalloutException('Failed to fetch weather.');
        }
    }

    public class WeatherData {
        public Main main;
        public List<Weather> weather;
```

```
    public class Main {
        public Decimal temp;
        public Decimal humidity;
    }
    public class Weather {
        public String description;
    }
  }
}
```

2. WeatherTriggerHandler.cls

```
public class WeatherTriggerHandler {
  public static void handleCityInsert(List<City__c> newCities) {
    for(City__c city : newCities) {
      WeatherAPIController.WeatherData data =
WeatherAPIController.fetchWeather(city.Name);

      Weather_Report__c report = new Weather_Report__c(
        City__c = city.Id,
        Temperature__c = data.main.temp,
        Humidity__c = data.main.humidity,
        Weather_Condition__c = data.weather[0].description
      );
      insert report;
    }
  }
}
```

Phase 6: User Interface Development

UI Tools:

- **Lightning Web Component (LWC)** used to create a dynamic weather display.
- **Input form** for city name, output panel for live weather.

Sample UI Code (HTML):

```
<template>
  <lightning-input label="Enter City" value={cityName}
onchange={handleChange}></lightning-input>
  <lightning-button label="Get Weather" onclick={getWeather}></lightning-button>

  <template if:true={weatherData}>
    <p>Temperature: {weatherData.main.temp} °C</p>
    <p>Humidity: {weatherData.main.humidity}%</p>
    <p>Condition: {weatherData.weather[0].description}</p>
  </template>
</template>
```

Sample JS Controller:

```
import { LightningElement, track } from 'lwc';
import getWeather from '@salesforce/apex/WeatherAPIController.fetchWeather';

export default class WeatherComponent extends LightningElement {
  @track cityName = '';
  @track weatherData;

  handleChange(event) {
    this.cityName = event.target.value;
  }

  getWeather() {
```

```
    getWeather({ cityName: this.cityName })
      .then(result => {
        this.weatherData = result;
      })
      .catch(error => {
        console.error(error);
      });
  }
}
```

Phase 7: Integration & External Access

🔗 **Integration Type:**

- **REST API Integration** with OpenWeatherMap using Apex HTTP callouts.

🔐 **Secure External Access:**

- Configured **Named Credentials** for storing the base URL securely.
- Remote Site Settings added to whitelist the endpoint.

Phase 8: Data Management & Deployment

🗂 **Deployment:**

- Used **Change Sets** to migrate components from Sandbox to Production.
- All Apex classes had >75% test coverage.

🗑 **Data Handling:**

- Weather reports are updated or deleted after 48–72 hours to manage data volume.

Deployment Tools:

- Salesforce CLI (optional)
- GitHub for source control

Phase 9: Reporting, Dashboards & Security Review

Dashboards:

- "Top 5 Cities by Weather Requests"
- "Avg. Temperature by City"

Security Compliance:

- CRUD/FLS respected in all Apex queries.
- Users assigned Permission Sets for:
    - Read/Create access on City__c and Weather_Report__c
    - API callout rights (via profile)

Phase 10: Final Presentation & Demo Day

What Was Shown:

- Live weather fetched from OpenWeatherMap API
- Seamless Salesforce UI for input/output
- Automated data storage and updates
- Live chart with weather trends using Salesforce Reports

✅ **Results:**

- Fully functional project
- Scalable structure
- Clear API integration
- Secure and compliant