

Project Title

Weather Information Center

Submitted By:

A Prasanna

Email Id : 21jg1a0501.prasanna@gvpcew.ac.in

College : Gayatri Vidya Parishad College Of
Engineering For Women

1. Project Overview

The Weather Information Center is a real-time weather information application built on Salesforce using Lightning Web Components (LWC), Apex, and the OpenWeather API. Users enter a city name and receive instant weather details, such as temperature, humidity, pressure, and more. The application features a simple, interactive interface and is configured to work across various Salesforce pages, including Home, App, Record, and Community pages. This project showcases seamless external API integration in Salesforce, providing a practical tool for real-time weather insights within the Salesforce ecosystem.

2. Objectives

1. Provide Real-Time Weather Information: Enable users to access up-to-date weather details for any city directly within Salesforce.
2. Seamless API Integration: Integrate the OpenWeather API with Salesforce using Apex for external data retrieval.
3. User-Friendly Interface: Create an intuitive Lightning Web Component (LWC) UI for easy input and display of weather data.
4. Multi-Context Usability: Configure the component for use across various Salesforce pages (Home, App, Record, and Community).
5. Error Handling and Data Validation: Ensure robust handling of API errors and user input validation for a reliable user experience.

3. Salesforce Key Features and Concepts Utilized

1. Lightning Web Components (LWC): Used to build a responsive and user-friendly interface for capturing city input and displaying weather data.
2. Apex Callouts: Apex is leveraged to make HTTP requests to the OpenWeather API, retrieve weather information, and process API responses.
3. @AuraEnabled Apex Methods: Enables Apex methods to be accessible from the LWC, allowing seamless communication between front-end components and back-end logic.
4. Metadata Configuration (LightningComponentBundle XML): Configures the component's visibility across different Salesforce contexts (Home, App, Record, and Community pages).
5. Reactive Property Binding: Uses `@track` decorators in JavaScript to enable real-time updates in the component UI based on user input or API responses.

6. Custom Wrapper Class: An Apex wrapper class structures weather data into an organized format, facilitating easy display and usage in the LWC.
7. Conditional Rendering: Uses `template lwc:if` in HTML to control the display of weather information only when data is available.
8. Error Handling and Logging: Includes error capture in both Apex and LWC for debugging and to handle issues like invalid city names or API call failures.

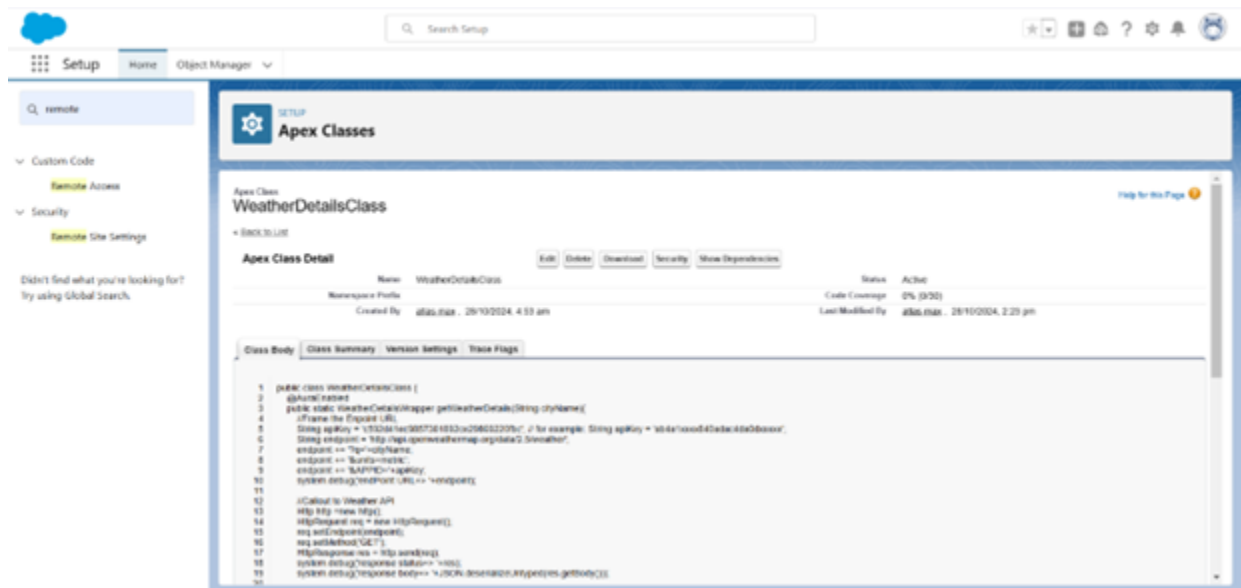
4.Detailed Steps to Solution Design

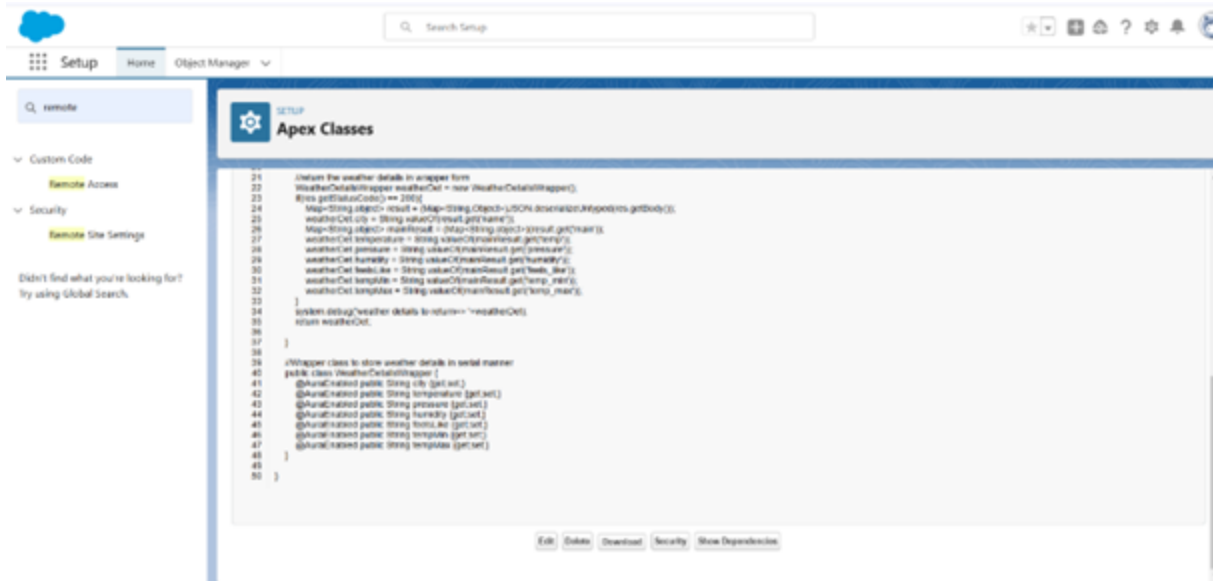
Step1.

Create Apex Class for API Integration (WeatherDetailsClass)

- 1.*Define getWeatherDetails Method:
 - The Apex method getWeatherDetails uses @AuraEnabled to expose it to LWC.
 - Construct the OpenWeather API endpoint dynamically, appending the cityName and API key parameters for the weather query.
- 2.Send HTTP Request to OpenWeather API:
 - Use Http and HttpRequest classes to initiate a GET request to the weather API.

- Capture the response in `HttpResponse`, checking the status to confirm a successful response.
- 3.Deserialize JSON Response:
 - Parse the JSON response into a `Map<String, Object>` and retrieve relevant weather details.
 - Populate the fields of `WeatherDetailsWrapper` (e.g., temperature, humidity, pressure) from the parsed data.
- 4.Return Data Using Wrapper Class:
 - Define a `WeatherDetailsWrapper` class to organize weather details and return them to the LWC.
 - This wrapper simplifies the transfer of structured weather data ts



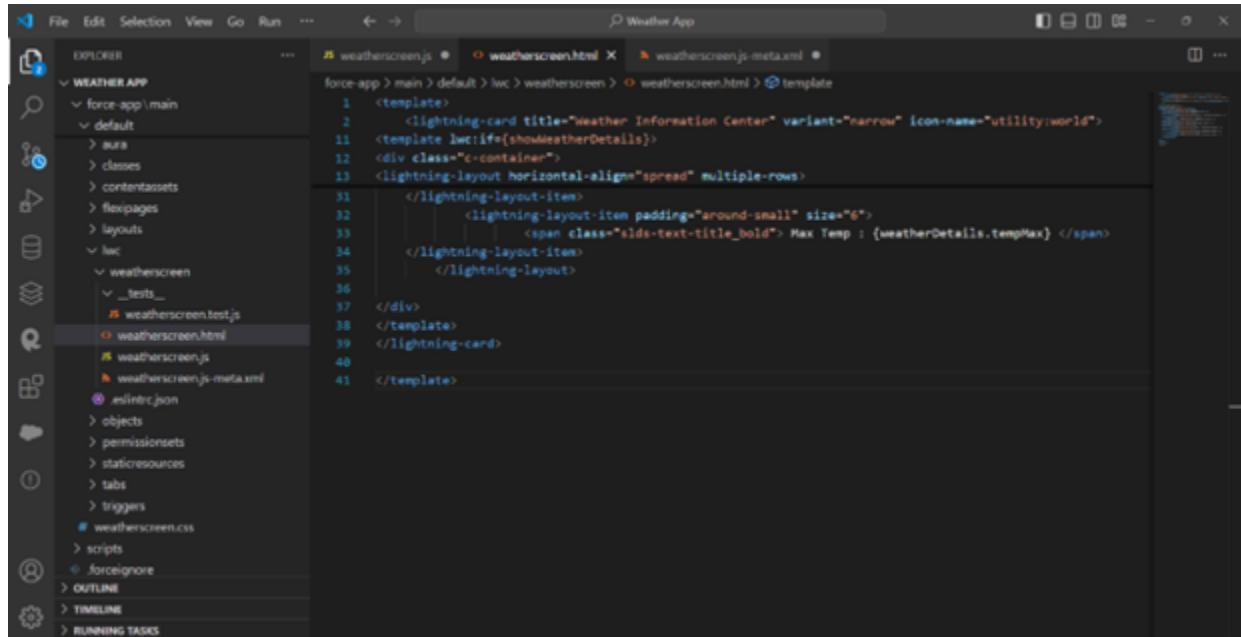


Step2.

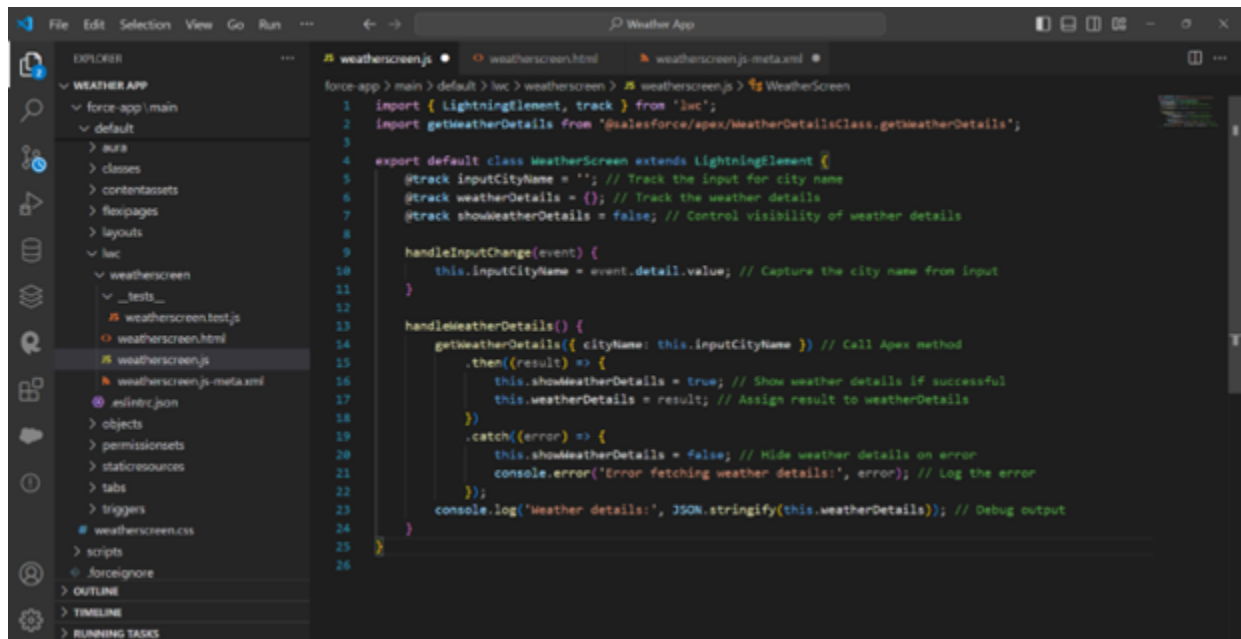
Design the LWC HTML Template

- 1.Create a Structured Card Layout:
 - Use <lightning-card> to display the component with a title (“Weather Information Center”) and an icon (utility:world) for a user-friendly, structured layout.
- 2.Add User Input and Button Elements:
 - City Input: Use <lightning-input> with a placeholder for city name entry and an onchange handler to capture real-time input.
 - Fetch Weather Button: Add <lightning-button> labeled “Get Weather Details” with an onclick handler to initiate the weather fetch process.
- 3.Display Weather Data Using Conditional Rendering:
 - Use <template lwc:if={showWeatherDetails}> to conditionally display weather information based on data availability.

- Use a responsive layout with `<lightning-layout>` and `<lightning-layout-item>` for a grid display of weather details (city name, temperature, humidity, etc.)



```
force-app > main > default > lwc > weatherscreen > weatherscreen.html > template
1 <template>
2   <lightning-card title="Weather Information Center" variant="narrow" icon-name="utility:world">
11    <template lwc:if={showWeatherDetails}>
12      <div class="c-container">
13        <lightning-layout horizontal-align="spread" multiple-rows>
31          </lightning-layout-item>
32          <lightning-layout-item padding="around-small" size="6">
33            <span class="slds-text-title_bold"> Max Temp : {weatherDetails.tempMax}</span>
34          </lightning-layout-item>
35        </lightning-layout>
36      </div>
37    </template>
38  </lightning-card>
40 </template>
```



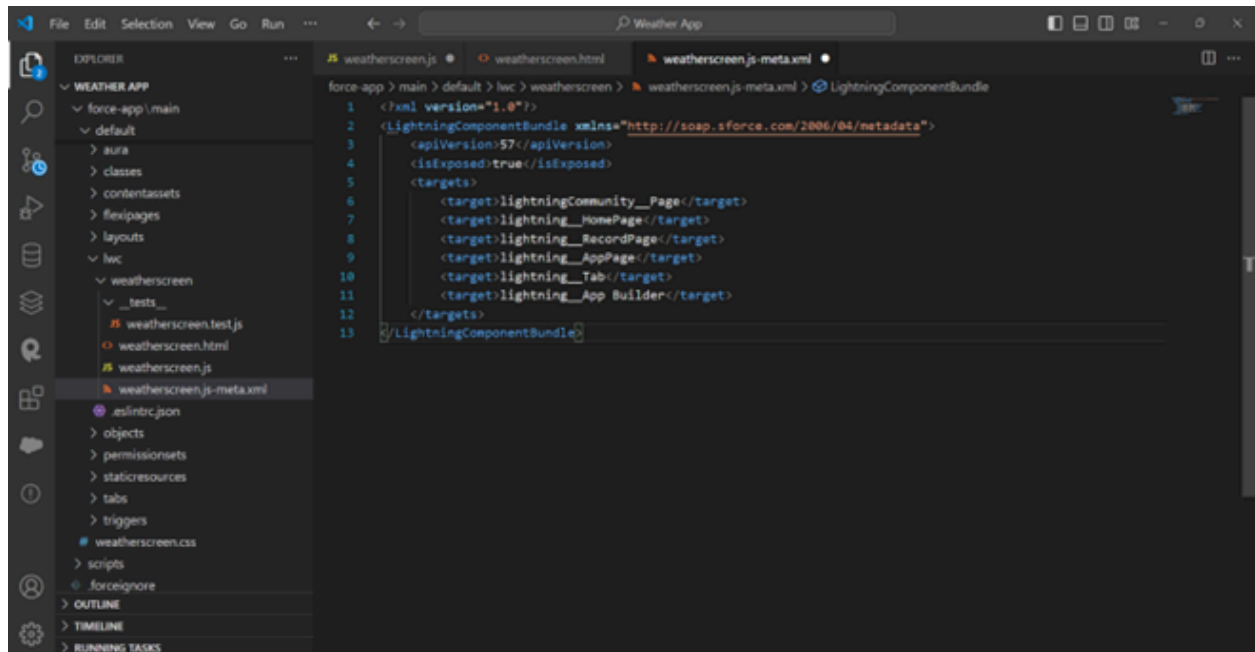
```
force-app > main > default > lwc > weatherscreen > weatherscreen.js > WeatherScreen
1 import { LightningElement, track } from 'lwc';
2 import getWeatherDetails from '@salesforce/apex/WeatherDetailsClass.getWeatherDetails';
3
4 export default class WeatherScreen extends LightningElement {
5   @track inputCityName = ''; // Track the input for city name
6   @track weatherDetails = {}; // Track the weather details
7   @track showWeatherDetails = false; // Control visibility of weather details
8
9   handleInputChange(event) {
10     this.inputCityName = event.detail.value; // Capture the city name from input
11   }
12
13   handleWeatherDetails() {
14     getWeatherDetails({ cityName: this.inputCityName }) // Call Apex method
15     .then((result) => {
16       this.showWeatherDetails = true; // Show weather details if successful
17       this.weatherDetails = result; // Assign result to weatherDetails
18     })
19     .catch((error) => {
20       this.showWeatherDetails = false; // Hide weather details on error
21       console.error('Error fetching weather details:', error); // Log the error
22     });
23     console.log('Weather details:', JSON.stringify(this.weatherDetails)); // Debug output
24   }
25 }
26
```

Step 3.

Implement JavaScript Logic in LWC Controller (WeatherScreen.js)

- 1.Track Variables with @track Decorator:
 - Use @track for inputCityName (to store city input), weatherDetails (to store fetched data), and showWeatherDetails (to control data visibility in the UI).
- 2.Handle User Input (handleInputChange Method):
 - Define handleInputChange to update inputCityName with the user's input from event.detail.value.
 - This ensures that the city name is updated in real time without requiring a page reload.
- 3.Handle Weather Fetch Action (handleWeatherDetails Method):
 - Define handleWeatherDetails to call the Apex method getWeatherDetails asynchronously, using this.inputCityName as the parameter.
 - Implement .then() to handle successful API responses and update weatherDetails with returned data, setting showWeatherDetails to true.

- Use .catch() to manage errors, logging issues to the console and setting showWeatherDetails to false in case of failures.

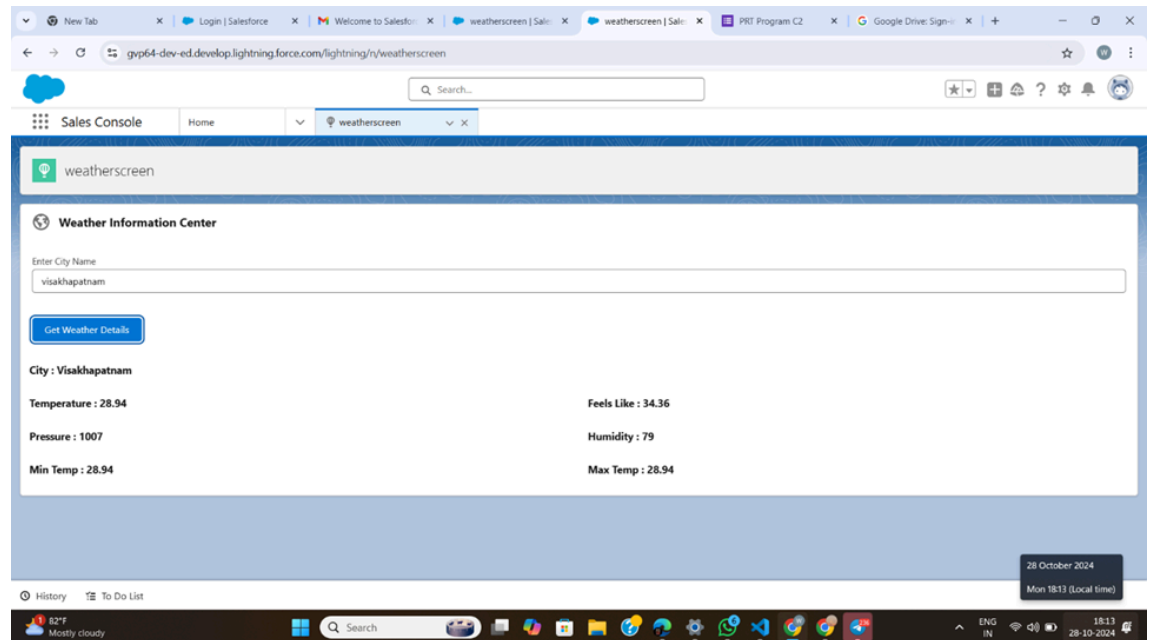


Step4.

Metadata Configuration and Component Exposure

- Define Component Metadata (XML):
 - Set up the LightningComponentBundle XML file to define apiVersion, set isExposed to true, and specify supported Salesforce targets such as lightning__HomePage, lightning__AppPage, lightningCommunity__Page, and others.

- This allows the component to be accessible on multiple pages and enhances flexibility.



5. Testing and Validation

The ****Weather Information Center**** output screen provides a clean, user-friendly interface that delivers real-time weather data for a specified city. The component's title and icon ("Weather Information Center" with a world icon) immediately set the context, while an input field labeled "Enter City Name" allows users to type in their desired city, with the option to click the "Get Weather Details" button to fetch data. Upon a successful API call, the component dynamically displays weather details, including the city name, current temperature, "feels like" temperature, atmospheric pressure, humidity, and minimum and maximum temperatures, all structured within a responsive grid layout for readability across devices. The layout ensures a clear presentation with `lightning-layout` rows and columns for a well-organized look. If an error occurs (such as an invalid city name or API issue), the data display remains hidden, while console logging aids in debugging. This design allows users to gain a quick, comprehensive view of the weather for any city, enhancing their Salesforce experience with relevant, real-time data.

6. Summary

Summary of Achievements

The Weather Information Center project has successfully delivered a robust solution for real-time weather information retrieval and display using Salesforce technologies. Key accomplishments include the development of a custom Apex class that integrates with the OpenWeather API, allowing users to input a city name and fetch corresponding weather data

seamlessly. The Lightning Web Component (LWC) is designed to provide an intuitive user interface, featuring an input field for city names and a button that triggers the weather data retrieval process. Upon successful API calls, the component dynamically displays essential weather details such as temperature, humidity, and atmospheric pressure, organized in a responsive layout for optimal user experience. Additionally, the implementation of effective error handling enhances the component's reliability by gracefully managing potential issues with API calls. The project is fully integrated into Salesforce environments, ensuring accessibility and usability across various platforms. Extensive testing and detailed user documentation have also been completed, reinforcing the solution's reliability and ease of use, ultimately showcasing the effective application of Salesforce tools for real-time data management and user engagement.

THANK YOU !!!