Phase 6: User Interface Development

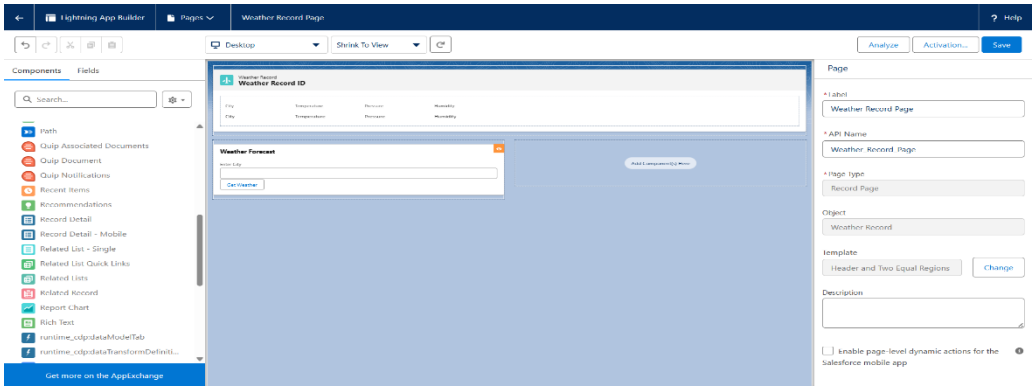
1.Lightning App Builder

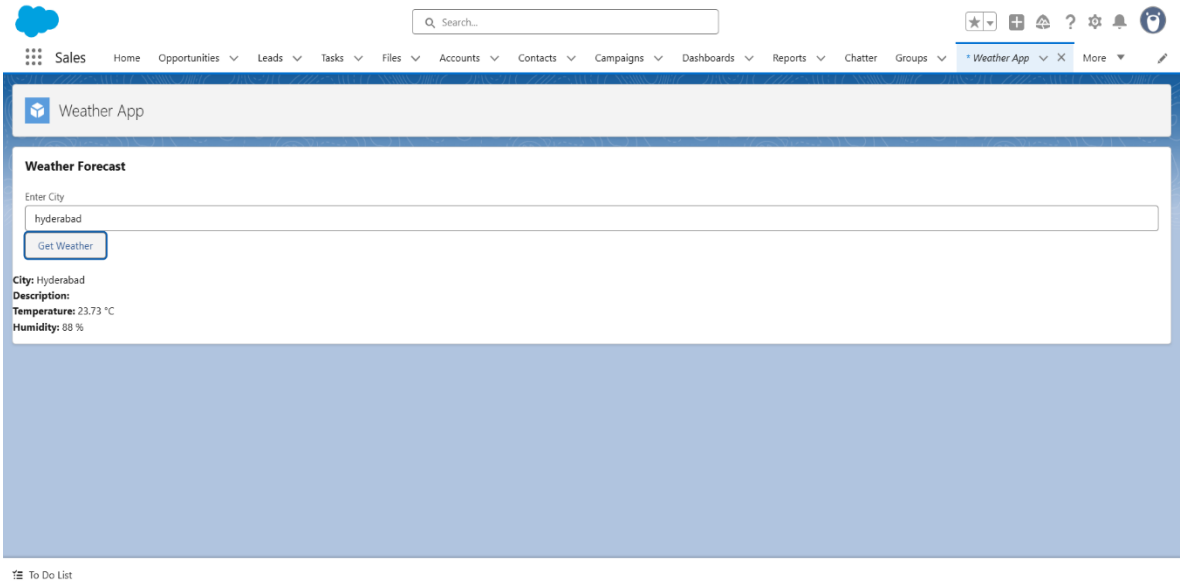
Purpose

- **Lightning App Builder** is a **point-and-click tool in Salesforce** that allows admins and developers to create and customize pages without writing any code.
- It supports Home Pages, Record Pages, and App Pages**,** enabling users to tailor the interface to business needs**.**
- With drag-and-drop components, it reduces dependency on developers and accelerates deployment of user-friendly pages.

Key Features

1. **Drag-and-Drop Interface**
   - Allows placing components (standard, custom, or LWC) anywhere on the page.
   - Users can preview the layout in real time.
2. **Multiple Page Types**
   - **Home Page:** Displays key information on login, dashboards, or quick actions.
   - **Record Page:** Customizes the layout of object records, showing fields, related lists, and LWCs.
   - **App Page:** Used for custom applications like a Weather Dashboard with multiple components.
3. **Responsive Layouts**
   - Choose from 1-column, 2-column, or 3-column layouts.
   - Adjust component width and placement for desktop and mobile devices.
4. **Component Types Supported**
   - **Standard Components:** Reports, Tasks, Chatter feeds, Related Lists.
   - **Custom Components:** Created via Lightning Web Components or Aura Components.
   - **Dynamic Components:** Components that display based on filters, user roles, or record data.
   -





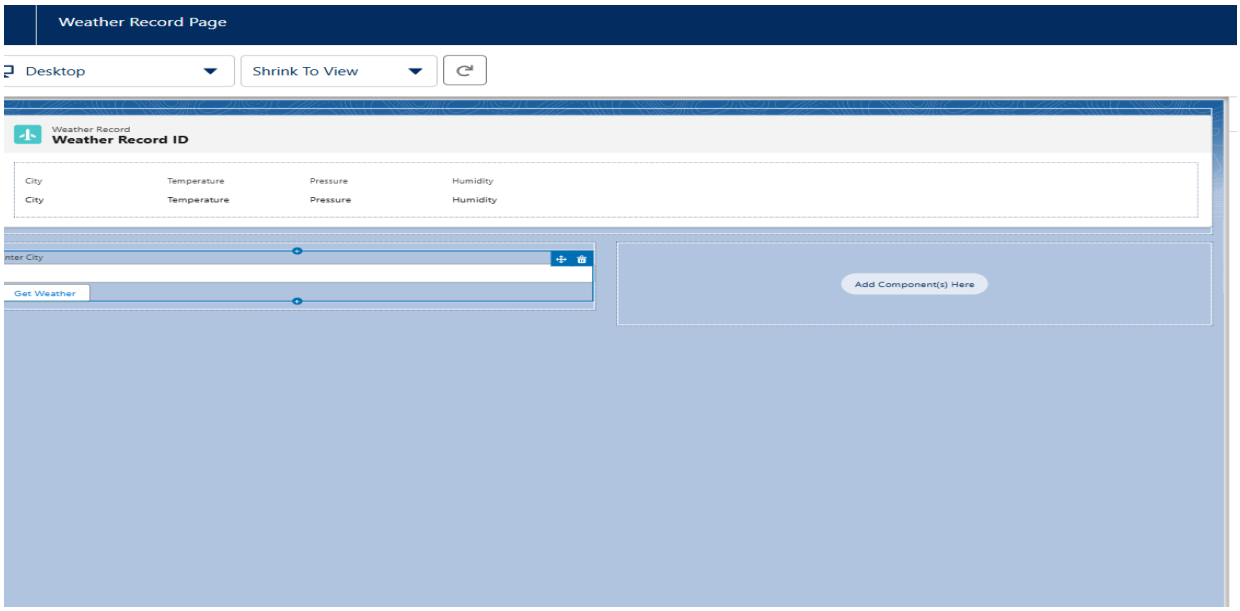
2. Record Pages

Purpose

- **Record Pages** in Salesforce allow you to **customize the layout of individual records** of any object, whether standard (Account, Contact) or custom (Weather_Record__c).
- They control **which fields, components, and related lists** appear for different users, profiles, and record types.
- Improves **data visibility and user experience** by displaying only relevant information.

Key Features

1. **Custom Layouts by Object & Record Type**
    - Different layouts for different profiles or record types.
    - Example: Weather_Record__c might show additional fields for "Admin" users but fewer fields for standard users.
2. **Component-Based Design**
    - Use **Lightning App Builder** to drag components like:
        - Standard components: Related lists, Highlights Panel, Tabs.
        - Custom components: WeatherApp LWC, charts, or dynamic dashboards.
3. **Dynamic Visibility Rules**
    - Components can appear or hide based on:
        - Field values (e.g., Status__c = "Confirmed")
        - User profile or role
        - Record type
4. **Mobile & Desktop Optimization**
    - Record Pages can be designed to work seamlessly on mobile Salesforce app.



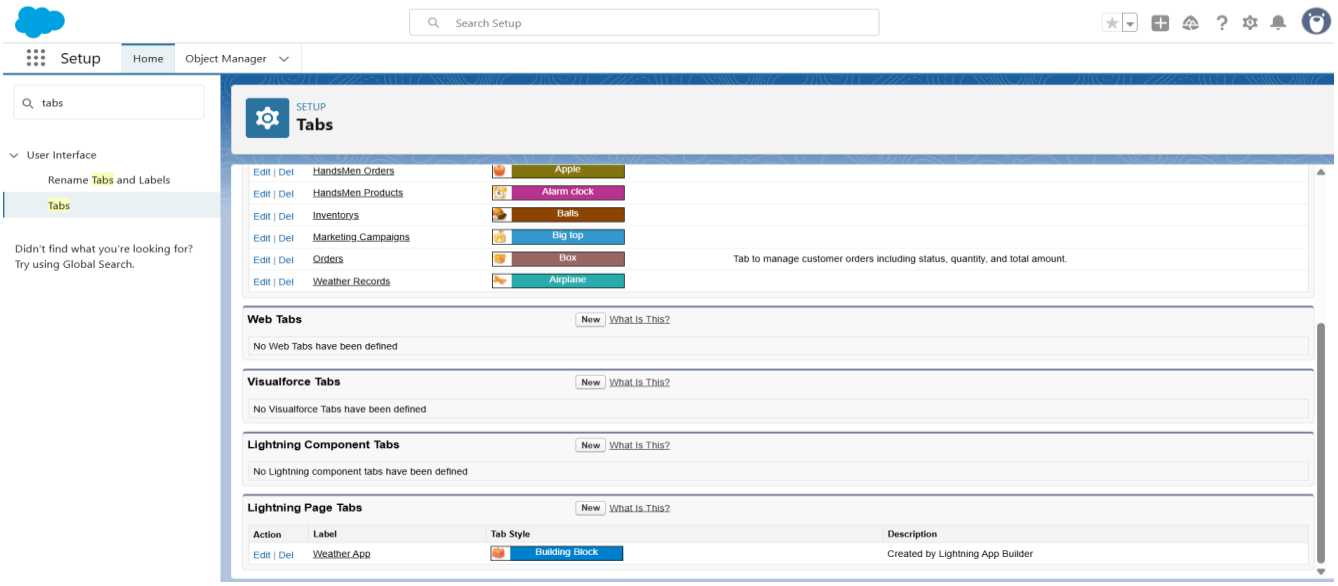
3. Tabs

Purpose

- **Tabs** allow users to **navigate between different objects, apps, or Lightning Pages** efficiently.
- They help **organize Salesforce UI** and provide quick access to key data.

Key Features

1. **Custom Object Tabs**
   - Link to custom objects like **Weather Records**, **Cities**, etc.
   - Appear in App navigation bar.
2. **Lightning Page Tabs**
   - Link directly to Lightning Pages (e.g., **Weather App Page**).
   - Can be used in App navigation or Utility Bar.
3. **Web Tabs**
   - Embed external web content or dashboards inside Salesforce.
4. **Visualforce & Lightning Component Tabs**
   - Display Visualforce pages or standalone Lightning Components as tabs.
5. **App-Specific Tabs**
   - Tabs can be added to **specific apps** via **App Manager → Edit → Navigation Items**.
   - Order of tabs can be customized for usability.



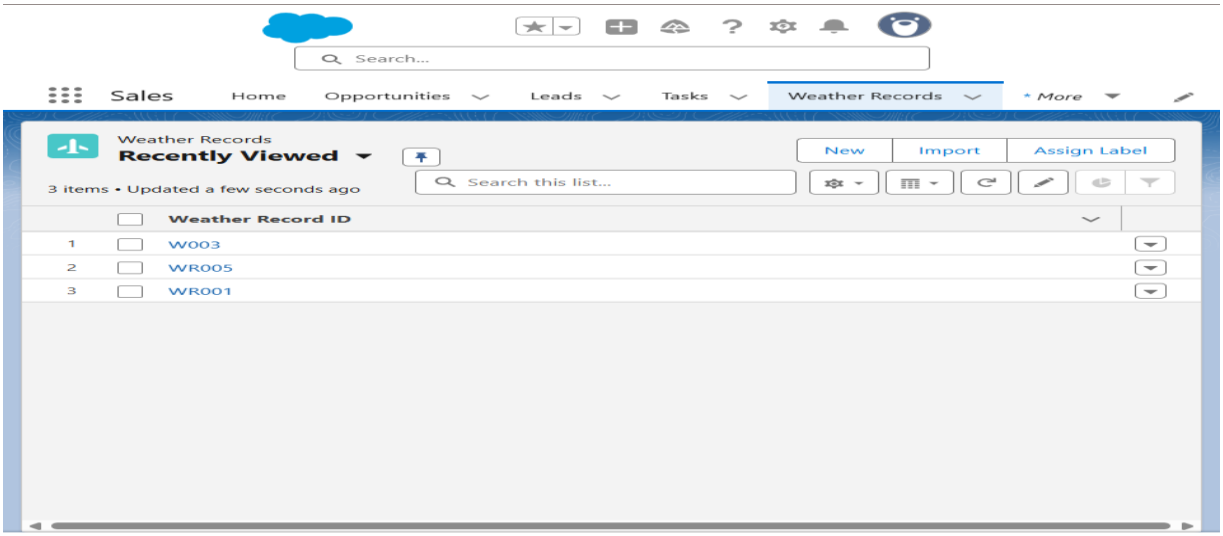
4.Home Page Layouts

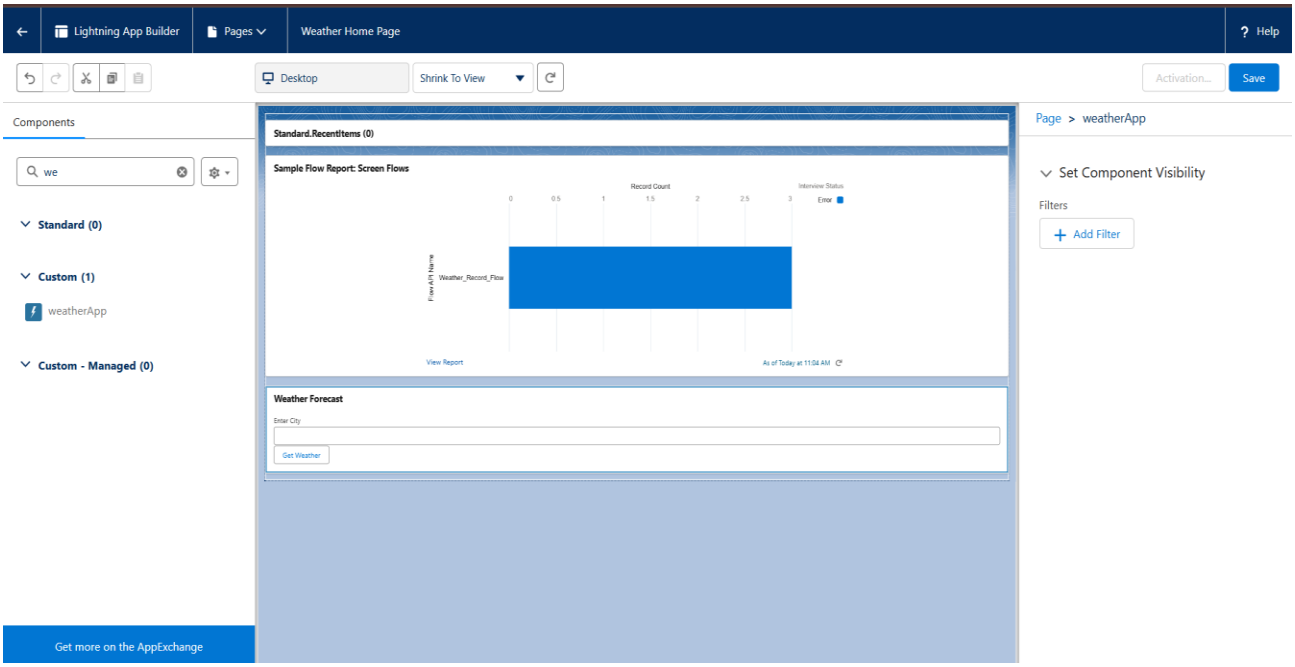
Overview

Home Page Layouts control **what users see immediately after logging into Salesforce**. They allow organizations to highlight key information, reports, tasks, or custom components like the Weather App for fast access. Properly designed layouts improve **user productivity and data visibility**.

Key Features

1. **Custom Layouts**
   - Choose from 1-column, 2-column, or 3-column templates.
   - Adjust components for desktop and mobile users.
2. **Drag-and-Drop Components**
   - Standard Components: Reports, Tasks, News, Recent Items.
   - Custom Components: Lightning Web Components like WeatherApp, charts, dashboards.
3. **Dynamic Visibility**
   - Show or hide components based on:
     - Profile
     - Role
     - Record type or field values
4. **Quick Access to Information**
   - Displays real-time or summarized data for immediate decision-making.
   - Reduces the need for users to navigate multiple pages.





5. Utility Bar

Overview

The **Utility Bar** is a persistent toolbar at the bottom of Salesforce apps. It provides quick access to essential tools and components, improving workflow efficiency without leaving the current page.

Key Features

1. **Persistent Access**
    - Available on all pages in the app.
    - Users can open utilities like calculators, WeatherApp, Tasks, or Notes while navigating other pages.
2. **Custom Components**
    - Drag Lightning Web Components or standard components into the Utility Bar.
    - Example: WeatherApp can show live weather in a small, collapsible panel.
3. **Configurable Properties**
    - Label, icon, panel size (small, medium, large).
    - Visibility rules based on profiles or roles.
    - Auto-collapse or dock options for better UI.
4. **Enhances Productivity**
    - Provides tools at users' fingertips without navigating away.
    - Supports multitasking, e.g., viewing weather while editing records.

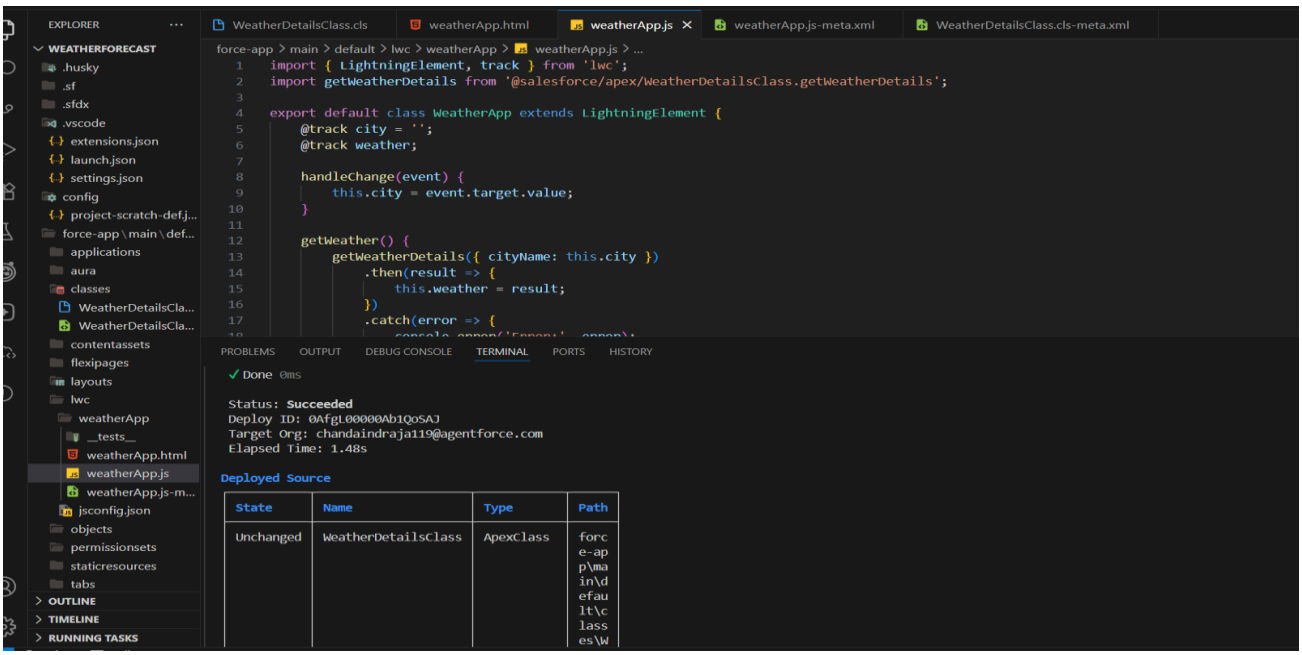
6. Lightning Web Components (LWC)

Overview

- Lightning Web Components (LWC) is **Salesforce's modern UI framework** for building fast, reusable, and interactive components.
- Based on **modern web standards** (HTML, JavaScript, CSS).
- LWCs can run on **Salesforce desktop and mobile**, integrate with **Apex**, and interact with other components.

Key Features

1. **Component-Based Architecture**
    - Modular, reusable, and encapsulated.
    - Each LWC consists of **HTML, JS, CSS, and meta XML**.
2. **Reactive Data Binding**
    - Uses **@track** and **@api** decorators to bind data between JS and HTML.
3. **Integration with Salesforce Data**
    - Can call **Apex methods** to fetch or update records.
    - Can use **Wire Adapters** for reactive Salesforce data.
4. **Performance & Efficiency**
    - Runs client-side with minimal server requests.
    - Lightweight and optimized for Salesforce Lightning Experience.

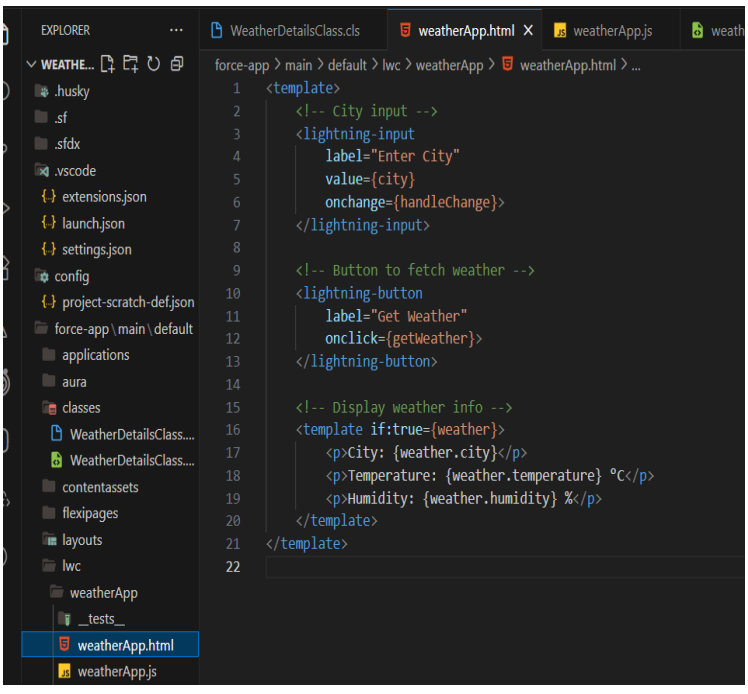



```
import { LightningElement, track } from 'lwc';
import getWeatherDetails from '@salesforce/apex/WeatherDetailsClass.getWeatherDetails';

export default class WeatherApp extends LightningElement {
    @track city = '';
    @track weather;

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

    getWeather() {
        getWeatherDetails({ cityName: this.city })
            .then(result => {
                this.weather = result;
            })
            .catch(error => {
                console.error('Error:', error);
            });
    }
}
```



7. Apex with LWC

Overview

- LWCs can communicate with **server-side Apex** to handle complex logic or access Salesforce data.
- Apex methods are called using **@AuraEnabled** decorators.

Key Features

1. **Imperative Calls**
   - JavaScript calls Apex on-demand (e.g., after user clicks "Get Weather").
2. **Reactive Calls with @wire**
   - Automatically fetches Salesforce data and updates the UI when records change.
3. **Data Processing**
   - Apex handles filtering, calculations, or API integration, then returns data to LWC.

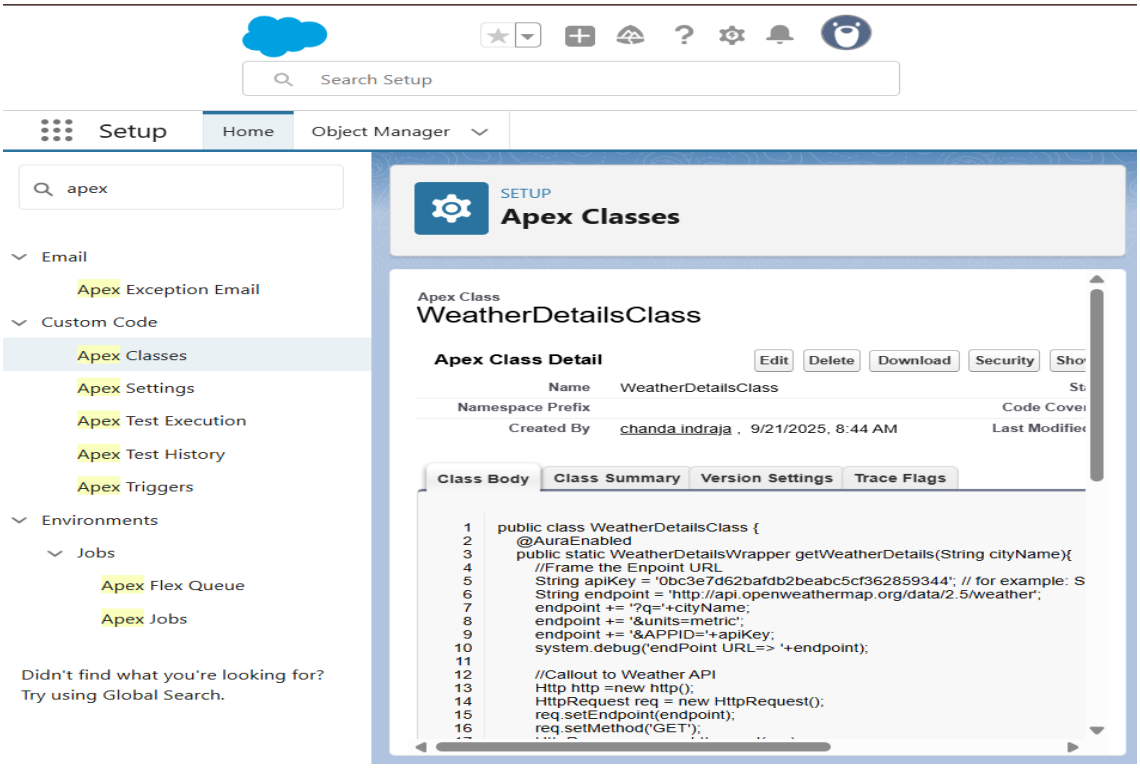
Step-by-Step Implementation

1. **Create Apex Class**
   - Example: `WeatherDetailsClass` with `@AuraEnabled(cacheable=true)` method.
   - Handles fetching weather details from API or Salesforce records.
2. **Call Apex from LWC**
   - **Imperatively** in JS:

```
getWeatherDetails({ cityName: this.city })
.then(result => { this.weather = result; })
.catch(error => { console.error(error); });
```

- **Reactively** with `@wire` decorator for automatic updates.

3. **Handle Success & Errors**
   - Update component state on success.
   - Use `catch()` for errors or display **toast messages**.
4. **Deploy Apex Class and LWC**
   - Deploy both together to Salesforce.
5. **Test Component**
   - Place LWC on a Lightning Page → verify Apex data is fetched and displayed.



8. Events in LWC

Overview

- **Events in LWC** allow components to communicate **within the component hierarchy**.
- Two main types:
    1. **Standard DOM Events** – e.g., `click`, `change`.
    2. **Custom Events** – e.g., sending weather data from a child component to a parent component.

Key Features

1. **CustomEvent**
    - Send structured data using `detail` object.
    - Example: `{ detail: { city: 'Mumbai', temperature: 25 } }`.
2. **Event Bubbling**
    - Events propagate up the DOM to parent components.
3. **Component Interaction**
    - Enables reusable child components that inform parent components about user actions.

. 9.Wire Adapters

Overview

- **Wire Adapters** in LWC provide a **reactive connection** between Salesforce data and your Lightning Web Component.
- Data fetched via wire adapters automatically **updates the component when underlying Salesforce data changes**.
- Ideal for **read-only or reactive UI elements**, such as displaying Weather Records in real-time.

Key Features

1. **Reactive Data Binding**
   - Automatically updates the component when data changes in Salesforce.
2. **Integration with Apex or Standard Objects**
   - Use `@wire` to call **Apex methods** or **standard Lightning Data Service adapters** like `getRecord` or `getListUi`.
3. **Declarative Syntax**
   - Simplifies code; no need to explicitly handle server requests or callbacks.

10. Imperative Apex Calls – Detailed Documentation

Overview

- **Imperative Apex Calls** in LWC are used to **call Apex methods on-demand**, typically triggered by user actions like button clicks.
- Unlike `@wire`, which is reactive, imperative calls are **explicit and controlled by the component**.

Key Features

1. **On-Demand Execution**
   - Fetch or process data when required, e.g., after clicking "Get Weather".
2. **Supports Promises**
   - Uses `.then()` and `.catch()` to handle success and errors.
3. **Flexibility**
   - Suitable for dynamic operations that depend on user input.