Sales Force -Last Mile Admin Project

Project Name: SkyCast – Smart City Weather Information System

Problem Statement: Modern cities are increasingly affected unpredictable weather events such as heavy rainfall, heatwaves, air pollution, and storms. Citizens, city administrators, and emergency services often lack real-time, localized weather insights to make quick, data-driven decisions.

Current systems are either generic weather apps (that do not provide city-specific microclimate data) or disjointed dashboards with no integration into smart city infrastructure.

SkyCast aims to bridge this gap by building a **Smart City Weather Information System** that delivers hyperlocal weather forecasts, air quality data, and emergency alerts to citizens, businesses, and government agencies.

Phase 1: Problem Understanding & Industry Analysis

1.Requirement Gathering

- Collect **functional requirements**:
  - Real-time weather updates (temperature, rainfall, wind speed, humidity)
  - Air quality index monitoring
  - Smart city integration (IoT sensors, APIs from meteorological departments)
  - User-specific notifications (alerts for storms, floods, heatwaves)
- Collect **non-functional requirements**:
  - High availability and scalability
  - Data security and compliance (e.g., for IoT sensor data)
  - Multi-platform support (web, mobile, smart kiosks)

2. Stakeholder Analysis

- **Citizens** → Need accurate, real-time weather alerts for daily planning & safety
- **City Administrators** → Require dashboards for urban planning, traffic management, disaster preparedness
- **Emergency Services** → Need instant alerts for floods, storms, fire hazards, etc.
- **Businesses** (logistics, transport, retail, agriculture) → Rely on weather insights for operations
- **Meteorological Agencies** → Provide raw data & forecasts
- **Citizens & Businesses** – Need accurate, real-time weather and air quality information to plan daily activities, travel, and operations safely.

3. Business Process Mapping

AS-IS Process (Current Scenario):

- Citizens rely on generic weather apps → limited accuracy
- Administrators depend on scattered reports from different agencies
- No central platform for **real-time insights & alerts**

TO-BE Process (With SkyCast):

1. IoT weather sensors + external APIs collect live data
2. Data processed and stored in Salesforce/Cloud platform
3. SkyCast dashboard provides **real-time weather visualization**
4. Automated **alerts/notifications** sent to citizens & stakeholders
5. Integration with **smart city infrastructure** (traffic lights, public transport updates, emergency services)

4. Industry-specific Use Case Analysis

- **E-Governance (Smart Cities in India, Singapore, Dubai):** Real-time pollution monitoring dashboards
- **Transportation & Logistics:** Companies like Uber & DHL use weather insights to optimize routes
- **Agriculture:** Hyperlocal forecasts help farmers plan irrigation & crop protection
- **Telecom & Energy:** Companies adjust networks and grids based on weather impact

5. AppExchange Exploration

- **Salesforce AppExchange** already has apps for:
  - Weather Forecast APIs integration
  - IoT data connectors
  - Emergency management & notifications
- Gaps found:
  - No **city-level hyperlocal solution** combining weather, AQI, and alerts
- Decision:
  - Use available **weather data connectors** from AppExchange
  - Build **custom SkyCast dashboards, alerts, and integrations** for smart cities

Phase 2: Org Setup & Configuration

1. Salesforce Editions

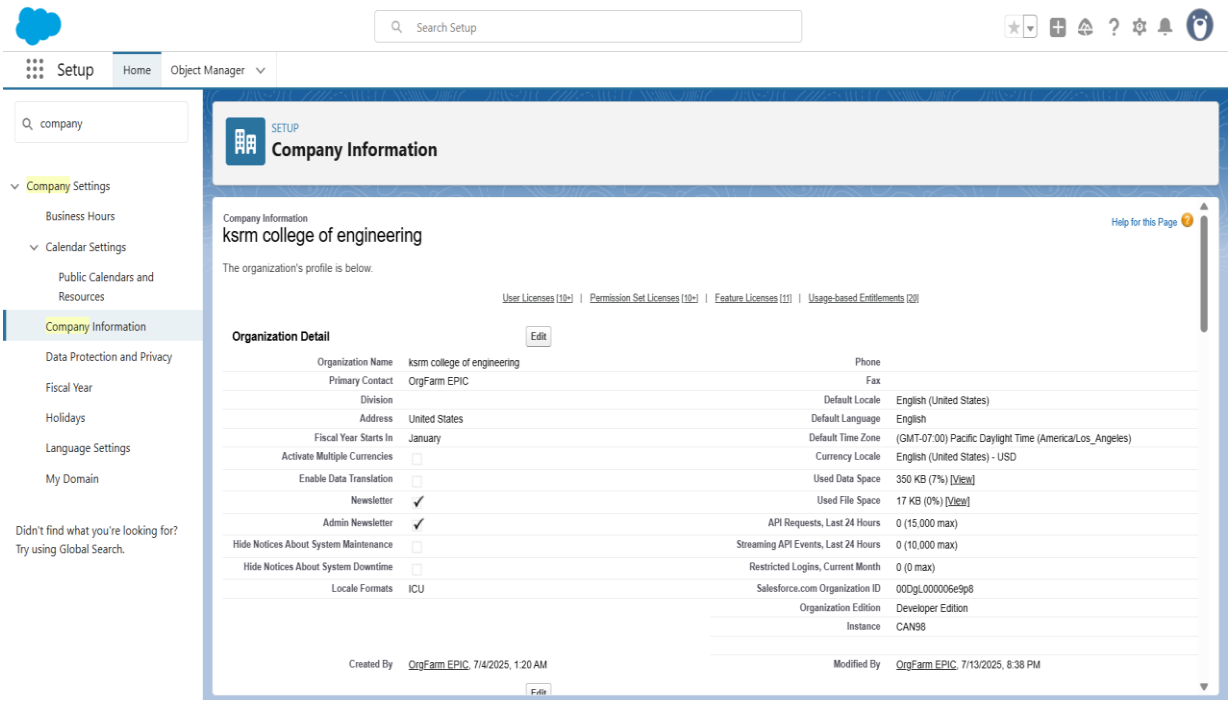
Choosing the right Salesforce Edition is a crucial step in configuring the SkyCast – Smart City Weather Information System. Each edition provides different levels of scalability, customization, and integration capabilities. After detailed evaluation, the Enterprise Edition was selected as the most suitable option.

- Evaluation of Editions: Professional, Enterprise, and Unlimited editions were compared for scalability, workflow automation, and API support.
- Selection Criteria: Considered factors like number of users, API limits, integration requirements, and cost feasibility.
- Final Choice: Enterprise Edition chosen for its balance of advanced automation and cost efficiency.
- Developer Org Usage: Developer Edition used for testing, training, and proof of concept activities.
- License Allocation: Licenses mapped for Admins, Standard Users, and API users.
- Justification: Enterprise Edition ensures scalability for future growth of the SkyCast system.

2. Company Profile Setup

The company profile setup establishes the foundational configuration for the Salesforce org. This ensures accurate localization, compliance, and brand identity within the system.

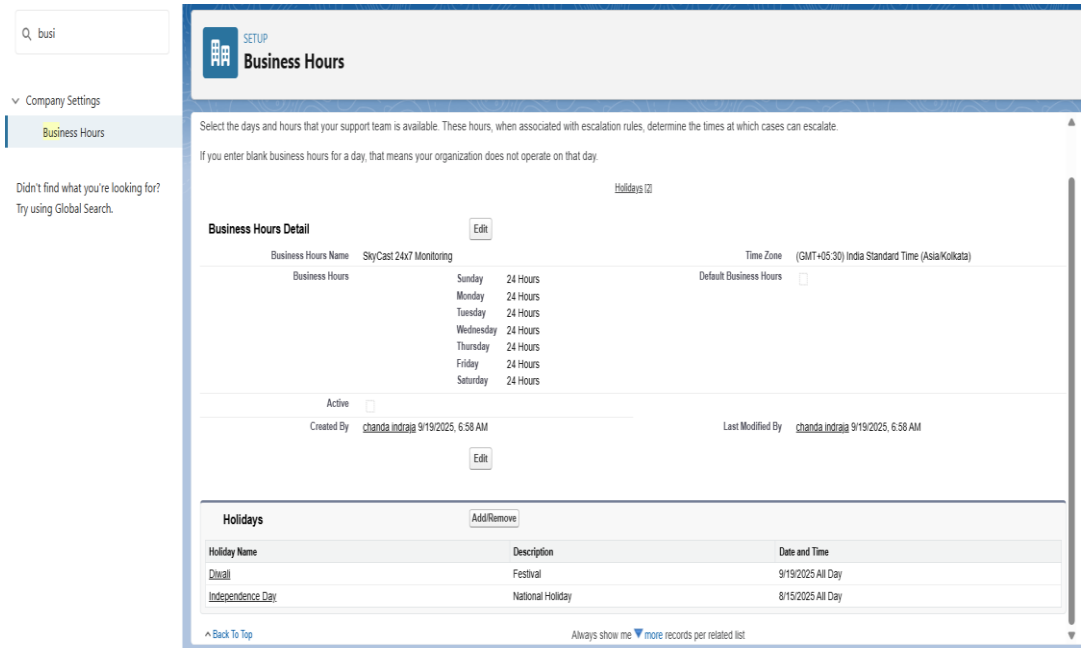
- Organization Details: Configured company name, address, and primary contact information.
- Locale & Time Zone: **Set to Indian Standard Time (IST) and English (India).**
- Default Currency: Enabled INR as the base currency and multi-currency support for USD/EUR.
- Branding: Uploaded company logo and configured email settings for notifications.
- Audit & Compliance: Verified organization information accuracy to prevent data/reporting issues.
- Governance: Ensured profile setup aligns with corporate policies.



3. Business Hours & Holidays

Business hours and holiday settings define availability for service-level agreements, escalation rules, and customer interactions. Proper configuration ensures accurate case handling and timely response to emergencies.

- **Standard Hours:** Defined 9 AM – 6 PM, Monday to Friday.
- **Holiday Calendar:** Configured key national holidays such as Republic Day, Independence Day, and Diwali.
- **Emergency Hours:** Created 24/7 service hours for City Administrators and Emergency Services.
- **Escalation Rules:** Linked escalation and case management with business hours and holidays.
- Testing: Verified that escalation pauses during defined holidays.
- **Regional Variations:** Added holidays for state-specific requirements.



4. Fiscal Year Settings

Feature	Configuration / Action	Impact / Notes
Standard Fiscal Year	April–March cycle (aligned with Indian financial year)	Ensures all reports and forecasts follow standard FY
Custom Fiscal Year	Evaluated but deferred	Maintains simplicity and avoids unnecessary complexity
Revenue & Forecast Alignment	Revenue, Opportunities, and ARR tracking aligned to fiscal year	Accurate revenue recognition and forecasting
Reporting & Analytics	Dashboards and reports configured with fiscal year filters	Consistent reporting and analytics across the org
Governance	Restricted changes to fiscal year	Prevents accidental changes; ensures long-term consistency

5. User Setup & Licenses

a) User Types & Roles

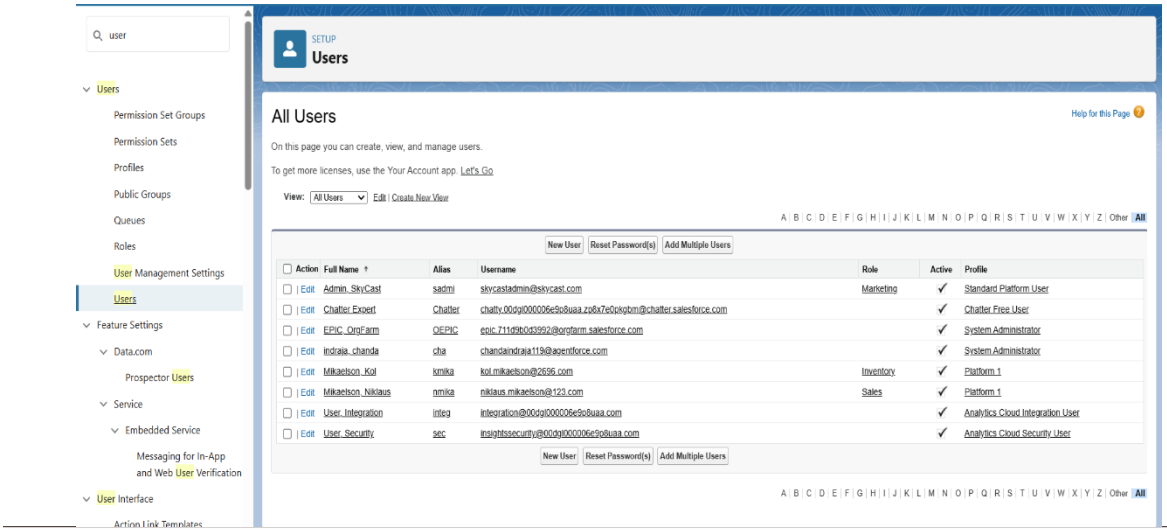
- **Admin Users:** Full system-level permissions for configuration and management.
- **Standard Users:** Role-based access tailored to responsibilities.
- **API Users:** Dedicated for IoT sensor and system integrations.

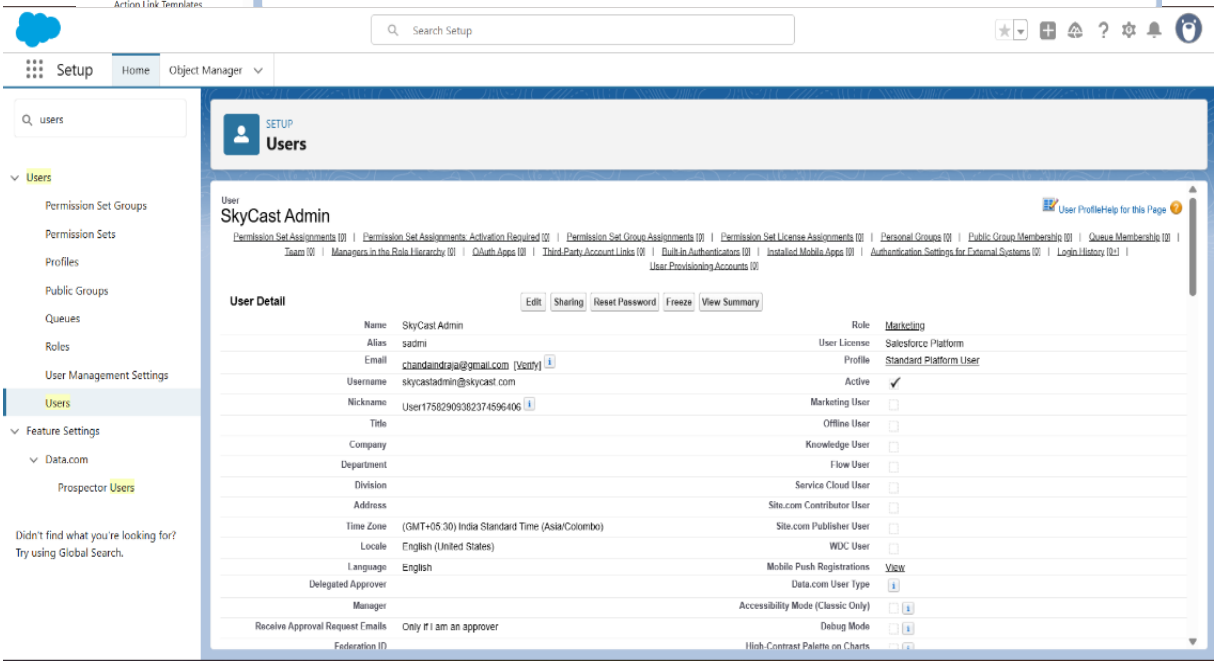
b) License Management

- Assigned based on job function to ensure compliance and cost-efficiency.
- Optimized allocation to prevent under- or over-licensing.

c) Security Measures

- **Multi-Factor Authentication (MFA):** Enforced for all privileged accounts.
- **Login IP Ranges:** Restricted to trusted locations to reduce security risks.





6. Profiles

- **Profile Details**

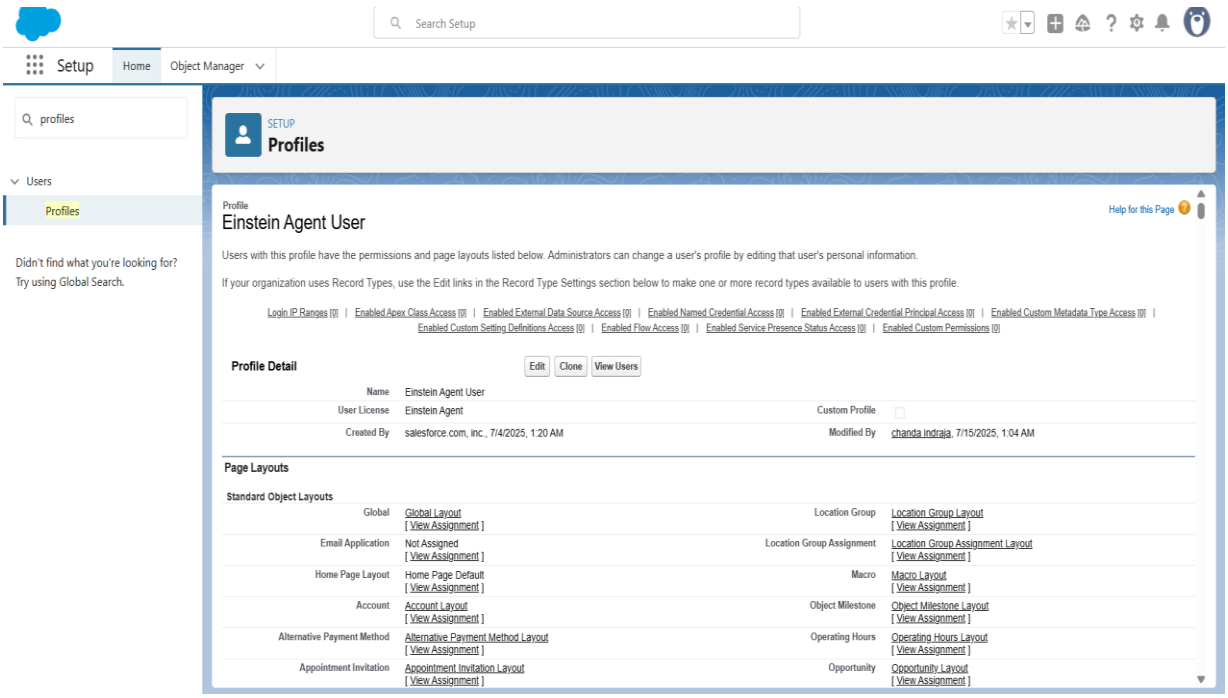
  - **Name:** Einstein Agent User
  - **User License:** Einstein Agent
  - Includes quick links to set **Login IP Ranges**, **Enabled Apex Class Access**, **External Data Source Access**, and other permissions.

- **Page Layout Assignments**

  - **Global Layout** – Default global page layout
  - **Home Page Default** – Home page configuration
  - **Account Layout** – Page layout for Account object
  - Additional layouts for Alternative Payment Method, Appointment Invitation, Location Group, Macro, Object Milestone, Operating Hours, and Opportunity.

- **Customization Options**

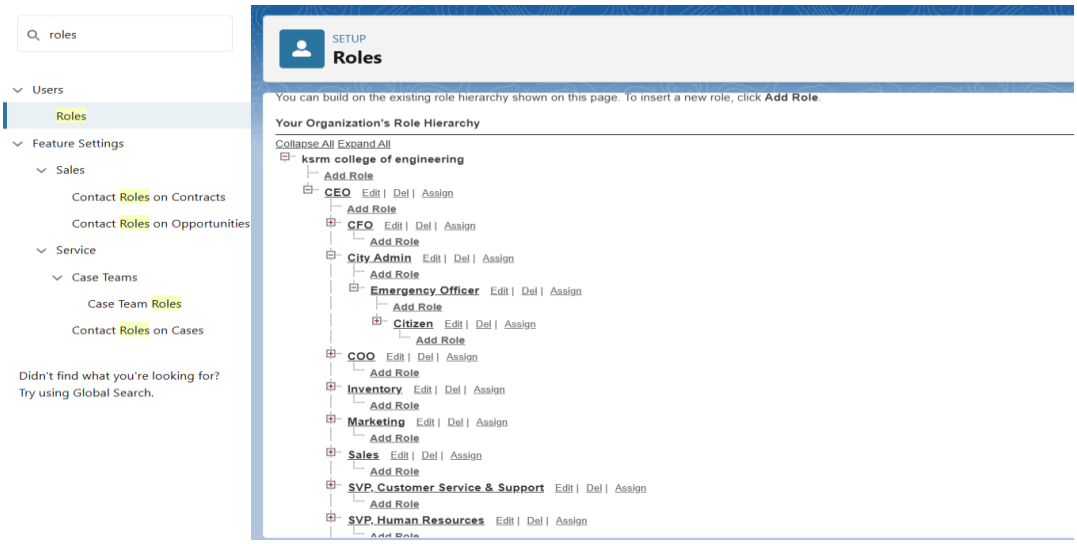
  - You can **Edit**, **Clone**, or **View Users** directly from the profile page.
  - Field-level security and tab visibility can be adjusted here to show only the relevant applications per profile.

- 

7. Roles

Roles control data visibility and hierarchical reporting structures in Salesforce. In SkyCast, they reflect the responsibilities of different smart city stakeholders.

- City Administrator: Top-level role with access to all records.
- Emergency Services: Role created with access to alerts and cases.
- Business Partners: Role with access to weather analytics and insights.
- Citizens: Limited access through community/portal roles.



8. Permission Sets

Objective

Provide selected users with **read-only access** to the custom object *Weather Data* without changing their base profile permissions.

Background

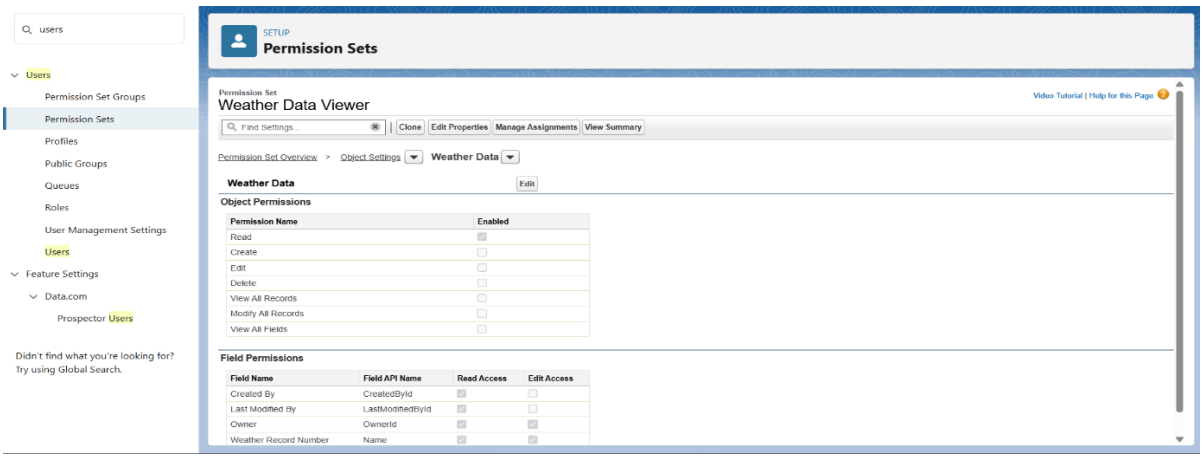
In Salesforce, **Permission Sets** extend a user's access beyond what their profile grants. **They are ideal when:**

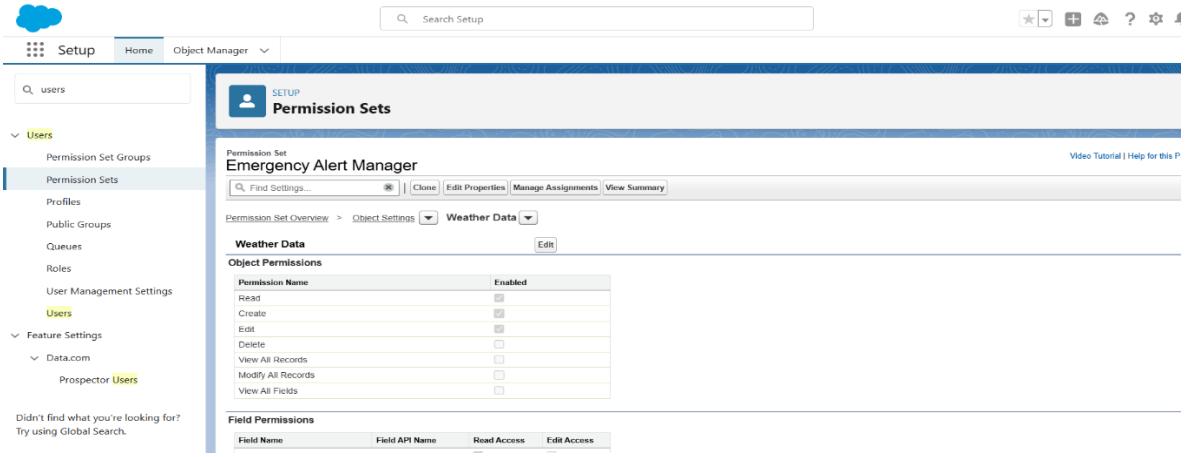
- Only certain users need extra permissions.
- You don't want to clone or modify profiles.

Configuration Details

Item	Description
Permission Set Name	Weather Data Viewer
Assigned Object	Weather Data (custom object)
Object Permissions	Read access only (Create, Edit, Delete disabled)
Field Permissions	Read access for all key fields: *Created By*, *LastModifiedBy*, *Owner*, *Weather Record Number*, *Name*
Assignments	Users who require visibility into weather records without edit rights

Implementation Steps



- 

9. Organization-Wide Defaults (OWD)

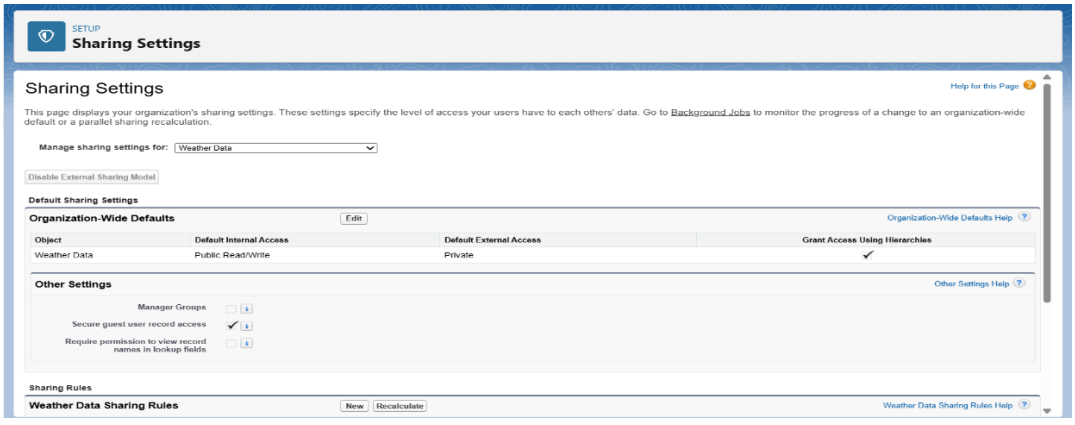
OWD settings define the baseline level of data access across the org. SkyCast required a balance between transparency of weather data and privacy for sensitive information.

- Accounts: Set to Private for confidentiality.
- Weather Data: Configured as Public Read-Only to allow transparency.
- Emergency Alerts: Access controlled via parent relationships.
- Cases: Configured with private visibility to protect sensitive reports.
- Grant Access Using Hierarchies: Enabled only where needed.
- Verification: Tested access using various user roles and profiles.

10. Sharing Rules

Sharing rules extend data access beyond OWD, ensuring that stakeholders receive the right insights without compromising security.

- Criteria-Based Sharing: Emergency alerts shared with Emergency Services.
- Owner-Based Sharing: Records shared upward through the role hierarchy.
- Public Groups: Configured for logistics, retail, and agriculture partners.
- Rule Testing: Conducted with multiple test records for accuracy.
- Audit Tracking: Documented all sharing rules for compliance.
- Quarterly Review: Established review cycle to keep rules updated.

- 

11. Login Access Policies

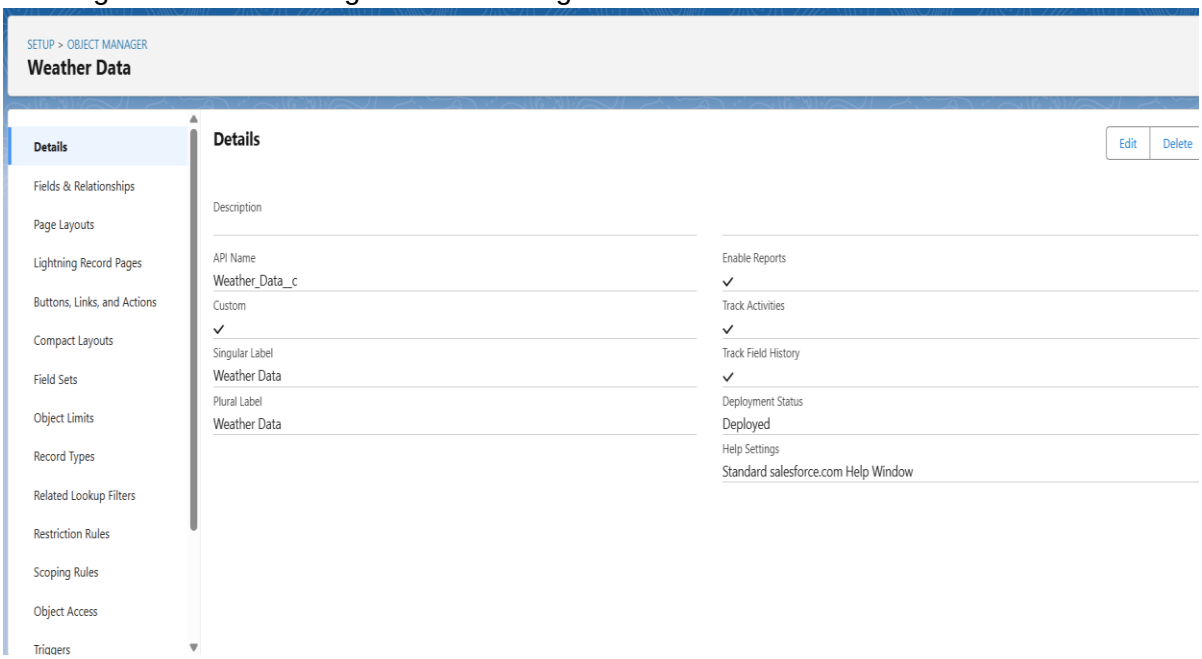
Login access policies provide additional security by defining conditions under which users can log in. This ensures strong security posture for SkyCast.

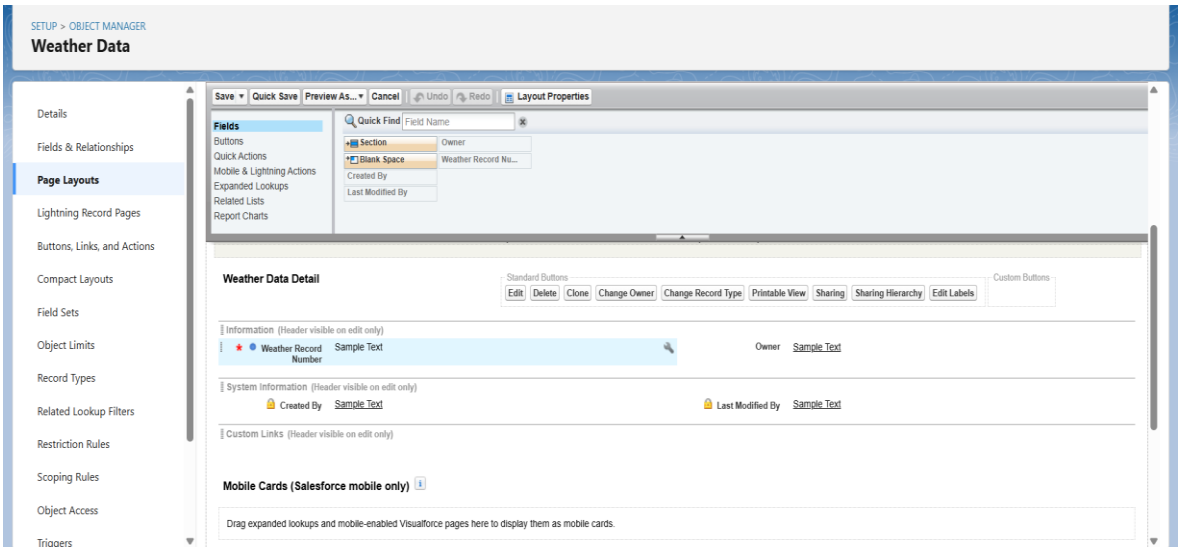
- IP Restrictions: Enforced trusted ranges for admins and API users.
- Login Hours: Limited non-admin users to business hours.
- MFA: Mandatory for all admins and privileged accounts.
- SSO Integration: Configured for corporate authentication.
- Login-As Policy: Enabled for admin troubleshooting with auditing.
- Monitoring: All login attempts logged and reviewed regularly.

12. Dev Org Setup

A developer org was established to test and validate configurations before moving to higher environments. This sandboxed approach ensures quality and reduces risks.

- Custom Objects: Created for Weather Data, Alerts, and IoT Readings.
- Page Layouts: Designed separately for different user groups.
- Validation Rules: Implemented to ensure data quality.
- Dashboards: Built sample weather monitoring dashboards.
- Debug Logs: Enabled to monitor integration behavior.
- Training: Used the Dev Org for user training and demonstrations.



- 

13. Sandbox Usage

Sandboxes were configured to ensure that development, testing, and deployment activities are isolated and safe. This protects the production environment.

- Developer Sandbox: Used for individual unit testing.
- Partial Sandbox: Contained sample data for integration testing.
- Full Sandbox: Created for UAT and performance testing.
- Refresh Strategy: Defined as monthly or after each release cycle.
- Data Masking: Applied to sensitive customer information.
- Deployment Testing: Verified deployment steps in sandbox before production.

14. Deployment Basics

Deployment planning was conducted to standardize movement of configurations across environments. This ensured smooth and error-free releases.

- Release Management: Defined process for Dev → UAT → Prod migration.
- Change Sets: Used for controlled metadata deployment.
- Version Control: GitHub maintained as the repository for metadata.
- Rollback Plans: Prepared for all major deployments.
- Naming Conventions: Implemented to standardize components.
- Documentation: Maintained logs for every deployment cycle.

Phase 3: Data Modeling & Relationships

This phase focuses on designing data structures, relationships, and layouts to manage weather information efficiently in Salesforce.

1. Standard & Custom Objects

Overview:

Salesforce provides Standard Objects (Account, Contact, User, etc.) and allows creation of Custom Objects to handle project-specific data.

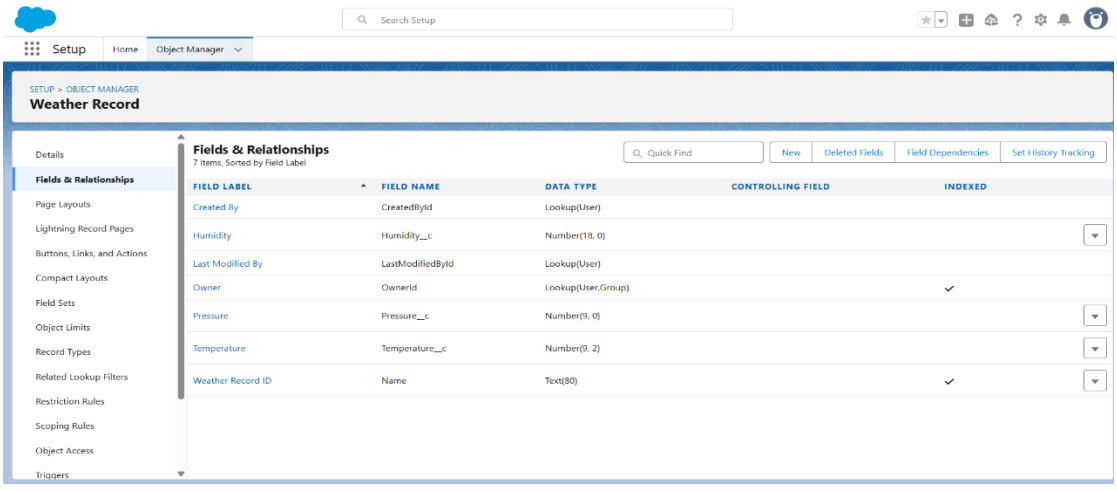
For this project, all weather-related objects are custom.

Objects Implemented:

Object Name Type Description
City__c Custom Stores city information including coordinates
Weather_Record__c Custom Stores weather data (Temperature, Humidity, Pressure, linked City)
Weather_Alert__c Custom Stores alerts for extreme weather conditions
Weather_Alert_City__c Custom Junction object to enable many-to-many relationship between Alerts and Cities
Optional: External_Weather_Data__x External Connects to OpenWeatherMap API to fetch live weather data

Implementation Notes:

All objects include standard audit fields: Created By, Last Modified By, Owner



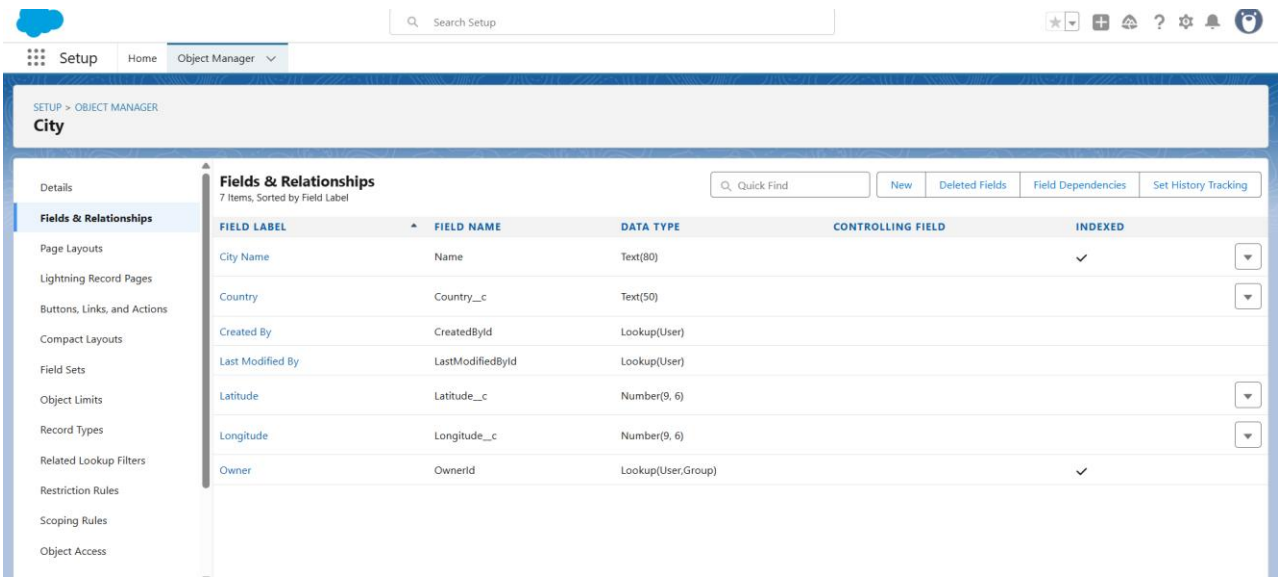
2. Fields

City__c Fields:

Field Name Type Description Real-Time Use Case
City Name Text Name of the city Identify city in Weather Records
Country__c Text Country of the city For geographical reporting
Latitude__c Number Latitude coordinate For mapping and external API calls

Longitude__c Number Longitude coordinate For mapping and external API calls

Weather_Record__c Fields:



Key Field Considerations

1. **Essential Data Capture** – The fields are carefully designed to store vital weather and city information such as City Name, Temperature, Humidity, Pressure, and Coordinates, ensuring the system can manage both geographical and climate-related data effectively.
2. **Practical Usage** – Every field supports a real-time business use case. For example, Temperature and Humidity fields drive dashboard insights, Precipitation fields help in flood alerts, and Timezone ensures weather records are logged accurately for each city.
3. **Data Accuracy & Integrity** – Relationships like Lookup and Master-Detail guarantee that every Weather Record is tied to the correct City, maintaining consistency and preventing duplicate or orphaned records.
4. **Scalability & Analytics** – Advanced fields such as Wind Speed, Visibility, and Condition are included to allow deeper analysis, support predictive weather models, and ensure the system can easily adapt to future requirements without major redesign

3.Record Types

Currently using default record type for all objects.Optional future enhancement: Different alert types (e.g., Rain Alert, Heatwave Alert) can use Record Types to control page layouts and picklist values.

- All objects (City__c, Weather_Record__c, Weather_Alert__c) currently use the **default record type**, ensuring consistent layouts and access to fields.

- Record Types allow different **page layouts and picklist values** based on record type, supporting tailored business processes.

- **Future enhancements** may include specialized alert types (e.g., Rain Alert, Heatwave Alert) with custom layouts and picklists to improve usability and reporting.

- Benefits include **better user experience**, reduced data entry errors, and **scalable reporting** for different alert categories.

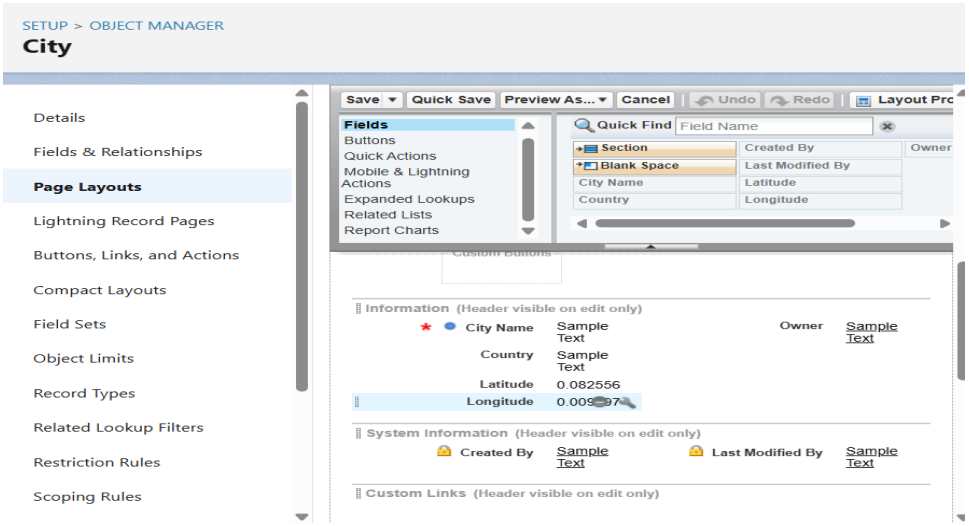
4.Page Layouts

City Page Layout:

- **Fields included:**
  - City Name
  - Country
  - Latitude
  - Longitude

Layout Design:

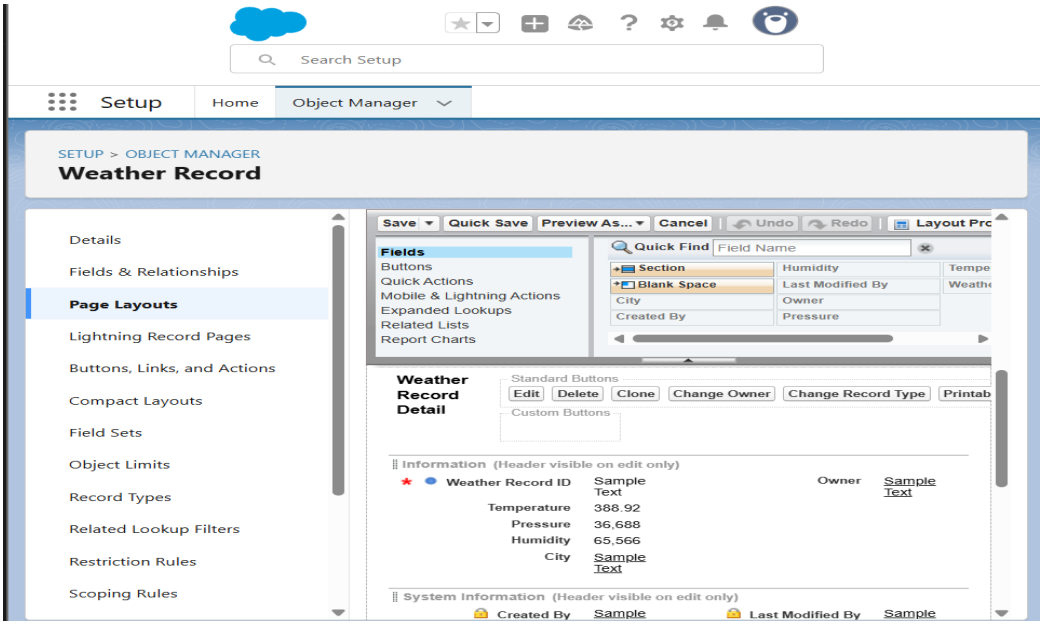
- Sections organized for **Quick View** and **System Info**.
- Includes **Related Lists** (Weather Alerts) for easy navigation.
  -



  -

Weather Record Page Layout:

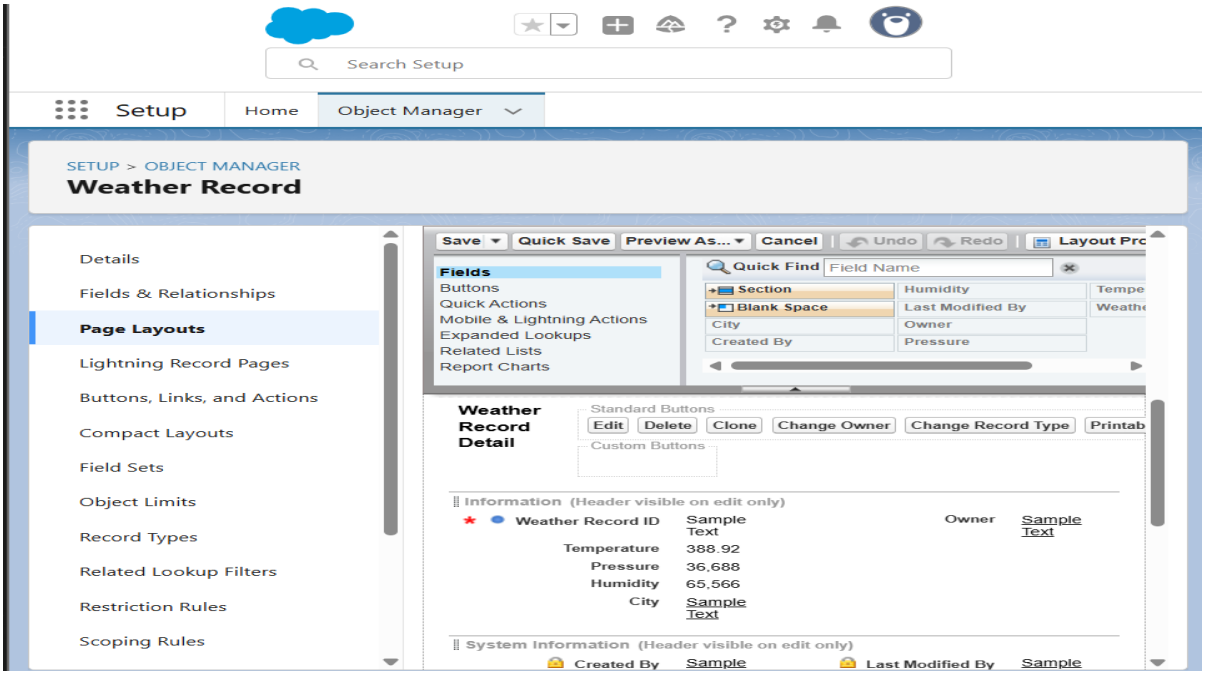
- **Fields included:**
  - City
  - Temperature
  - Humidity
  - Pressure
  - Weather_Record_ID
- **Layout Design:**
  - Sections organized for **Quick View** and **System Info**.
  - Includes **Related Lists** (Weather Alerts) for easy navigation.



5. Compact Layouts

- **Purpose:** Displays essential fields in the Lightning Highlights Panel for quick viewing.
- **City__c Fields Shown:** City Name, Country, Latitude, Longitude.
- **Weather_Record__c Fields Shown:** City, Temperature, Humidity, Weather_Record_ID.
- **Primary Assignment:** Each object has a primary compact layout assigned for consistent record previews.
- **User Benefit:** Enables quick access to critical information without opening the full record page.
- **Lightning Compatible:** Optimized for both desktop and mobile Lightning Experience.

Temperature



6.Schema Builder

Purpose:

- Provides a **visual overview of all objects and their relationships** in the project.
- Helps admins and developers **understand how data flows** between objects.

Relationships Implemented:

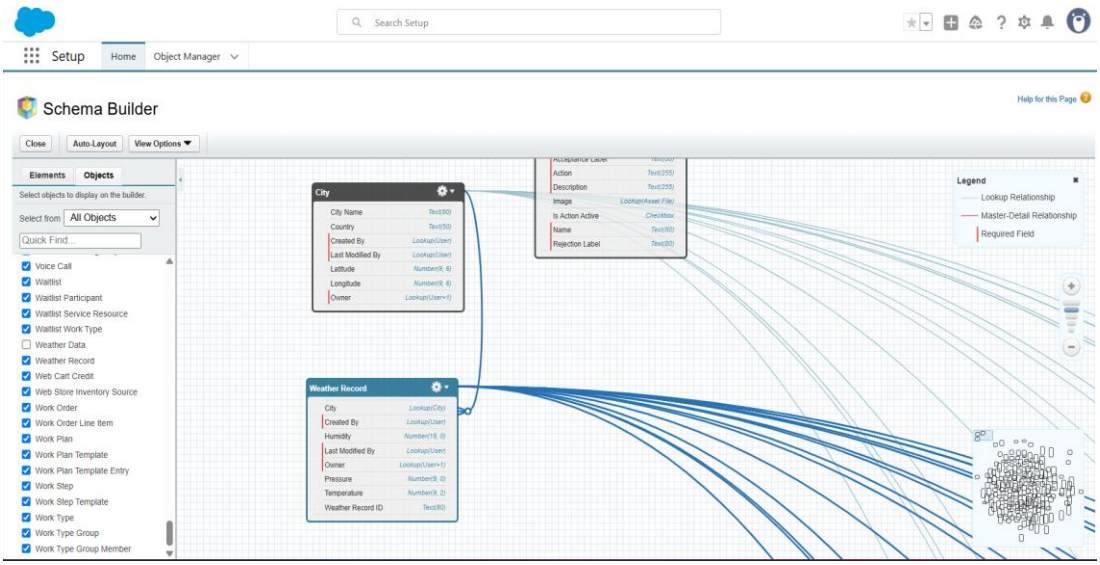
- **Weather_Record__c → City__c** (Lookup or Master-Detail) to link weather data with its city.
- **Weather_Alert_City__c → Weather_Alert__c** (Master-Detail) and → **City__c** (Master-Detail) to enable many-to-many alerts for multiple cities.

Layout & Organization:

- Objects are arranged for clarity: **City at the top**, Weather Records below, Alerts on the side.
- Drag-and-drop is used to organize the schema for **easy visualization and comprehension**.

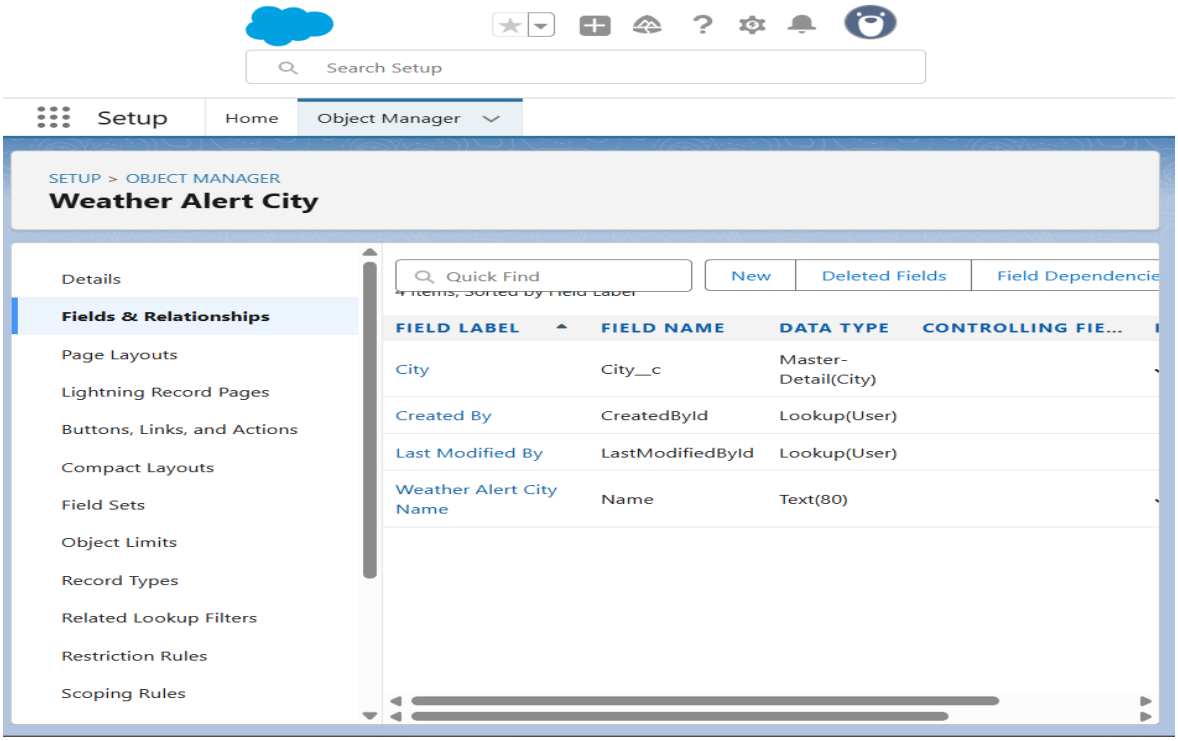
Benefits:

- Simplifies understanding of **object relationships** and data dependencies.
- Supports **reporting, dashboards, and future enhancements**.
- Provides a clear reference for **new team members or admins**

- 

7.Lookup vs Master-Detail vs Hierarchical Relationships

- **Lookup Relationship:**
  - Example: Weather_Record__c → City__c
  - Weather record exists independently of the city; deleting a city does not delete linked weather records.
  - Provides flexibility and optional linking between objects.
- **Master-Detail Relationship:**
  - Example: Weather_Record__c → City__c
  - Strong dependency; deleting the parent (City) deletes all child records.
  - Supports roll-up summary fields (e.g., average temperature per city).
- **Master-Detail (Junction):**
  - Example: Weather_Alert_City__c → Weather_Alert__c / City__c
  - Enables **many-to-many relationships**, allowing multiple alerts for multiple cities.
- **Hierarchical Relationship:**
  - Example: User → Manager
  - Not used in the Weather Project but useful for **approval processes and organizational hierarchy**.

- 

8.Junction Objects

- **Purpose:**
  - Enables **many-to-many relationships** between Weather Alerts and Cities.
  - Allows a single alert to be associated with multiple cities and vice versa.
- **Implementation:**
  - Custom object **Weather_Alert_City__c** acts as a junction.
  - Connected to **Weather_Alert__c** and **City__c** using Master-Detail relationships.
- **Real-Time Functionality:**
  - Multiple cities can be linked to a single alert.
  - Multiple alerts can be linked to a single city.
  - Ensures accurate tracking and reporting of alert coverage across locations.
- **Benefit:**
  - Supports **scalable alert management** across multiple cities.
  - Maintains **accurate relationships** between alerts and locations.
  - Facilitates **reporting, dashboards, and real-time monitoring** of alert coverage.

9. External Objects

External objects allow Salesforce to **access and display data from external systems** without storing it in Salesforce. In the Weather Project, the external object **External_Weather_Data__x** connects to a service like **OpenWeatherMap API** to fetch live weather data.

- Fields such as **Temperature, Humidity, and Pressure** are mapped from the external system to Salesforce.
- Data is displayed in **Lightning Record Pages or Dashboards**, providing **real-time insights**.
- This approach reduces **Salesforce storage usage** while keeping information **up-to-date**.

Key Benefits:

- Enables **dynamic and real-time monitoring** of weather conditions.
- Reduces internal storage requirements.
- Supports **analytics, dashboards, and decision-making** based on current weather data.

Phase 4 – Process Automation (Admin)

Process Automation in Salesforce streamlines business operations, reduces manual effort, and ensures data consistency. This phase focuses on building robust, real-time solutions using Salesforce's native automation tools.

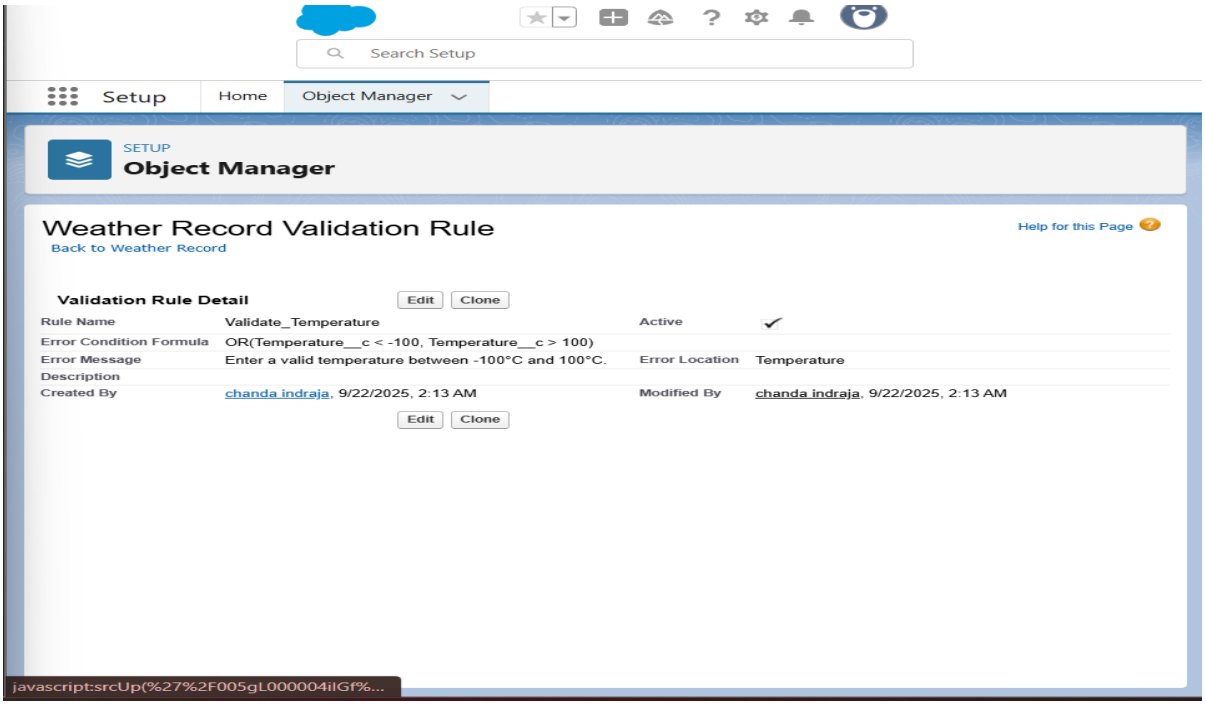
1. Validation Rules

Purpose
Validation Rules enforce data quality by preventing a record from being saved if it fails specified business criteria. They run whenever a record is created or updated—via the UI, API, Data Loader, or automation—ensuring that only complete and correct data enters Salesforce.

Key Benefits

- **Data Integrity:** Stops inaccurate or incomplete records before they impact reports or dashboards.
- **Compliance:** Enforces regulatory or internal policies (e.g., mandatory fields, date limits).
- **User Guidance:** Provides immediate, actionable error messages to end users.

Real-Time:
Scenario: Sales reps must set the Opportunity **Close Date** to today or a future date. Validation Formula: CloseDate < TODAY()

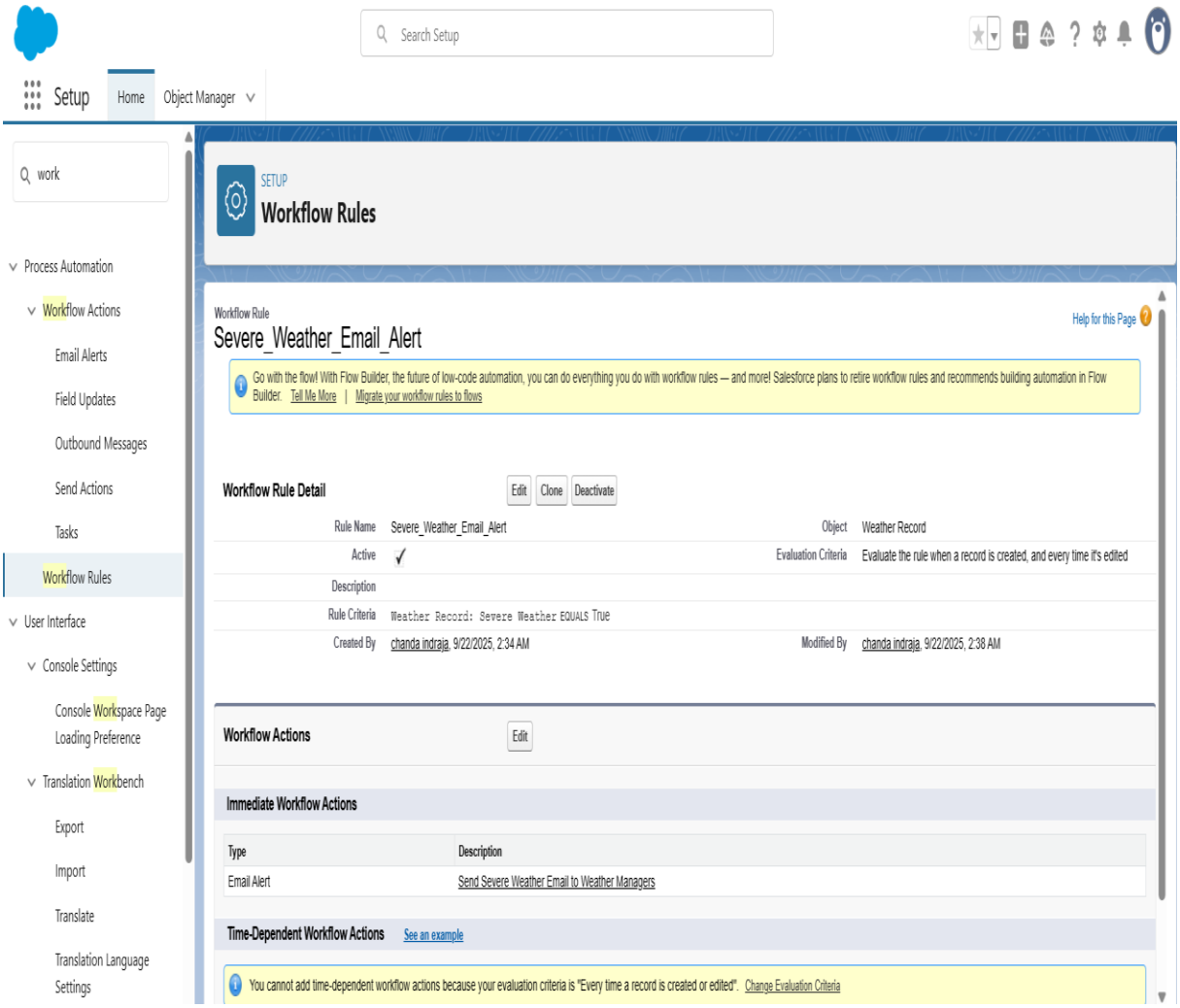


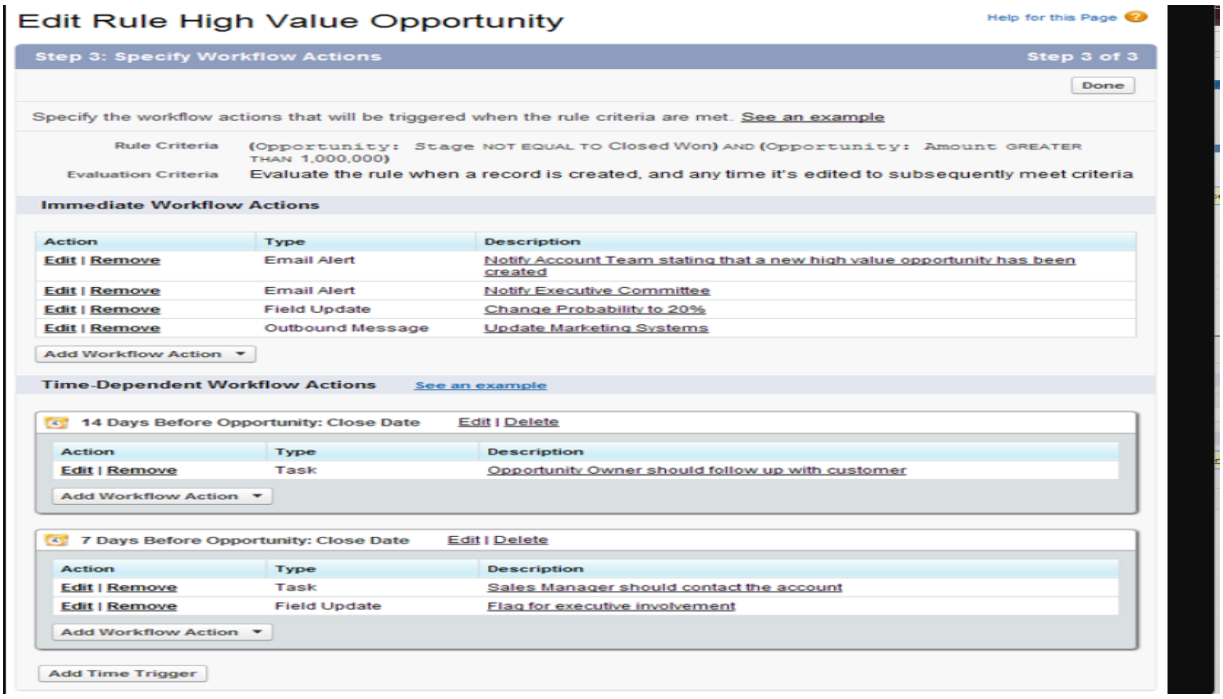
2. Workflow Rules

Purpose
Workflow Rules automate standard internal processes, allowing Salesforce to perform actions automatically when specific criteria are met.
They help reduce manual effort, improve response time, and ensure consistent business operations.

Key Benefits

- **Efficiency:** Automates routine tasks like sending emails or updating fields, freeing users to focus on high-value work.
- **Consistency:** Ensures that business rules are applied uniformly across all records.
- **Scalability:** Handles large volumes of records with minimal administrative overhead.





3. Process Builder

Purpose
Process Builder allows administrators to build complex, multi-step automation in Salesforce without code.
It supports multiple **if/then** branches and can call **Flows, Apex classes, or other processes**, enabling powerful end-to-end automation.

Real-Time Business Example – Rain Alert Level

Scenario: When a Weather Record indicates a high probability of rain, the system automatically sets the Alert Level to "High."

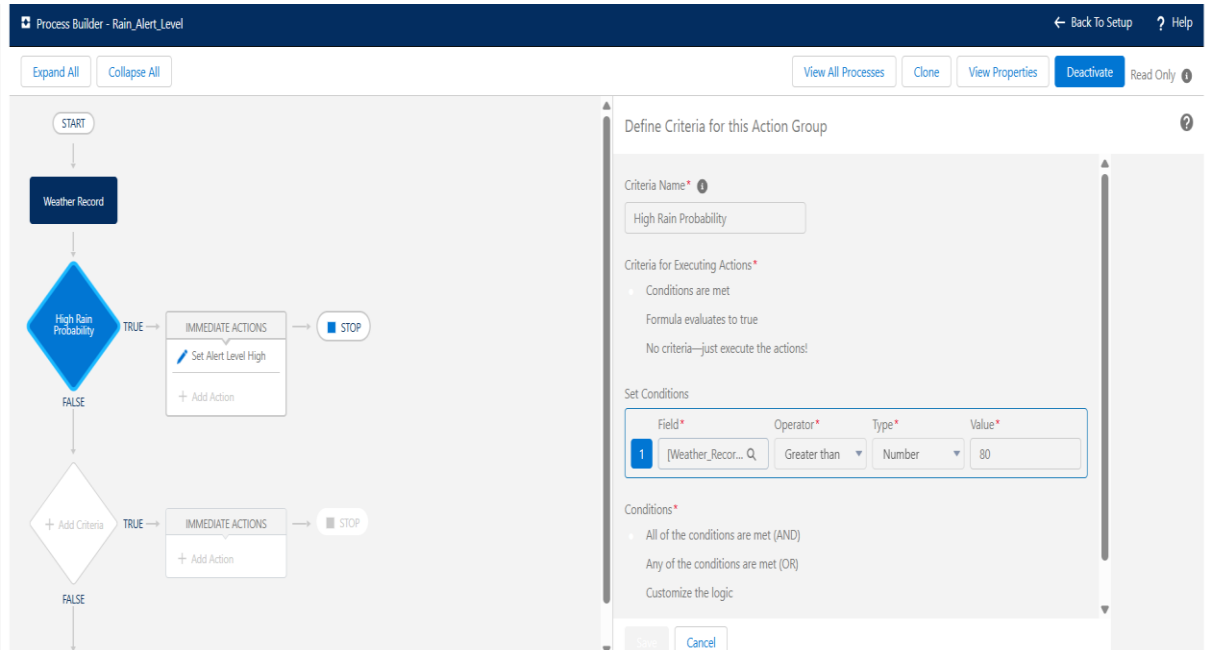
Key Configuration *:*

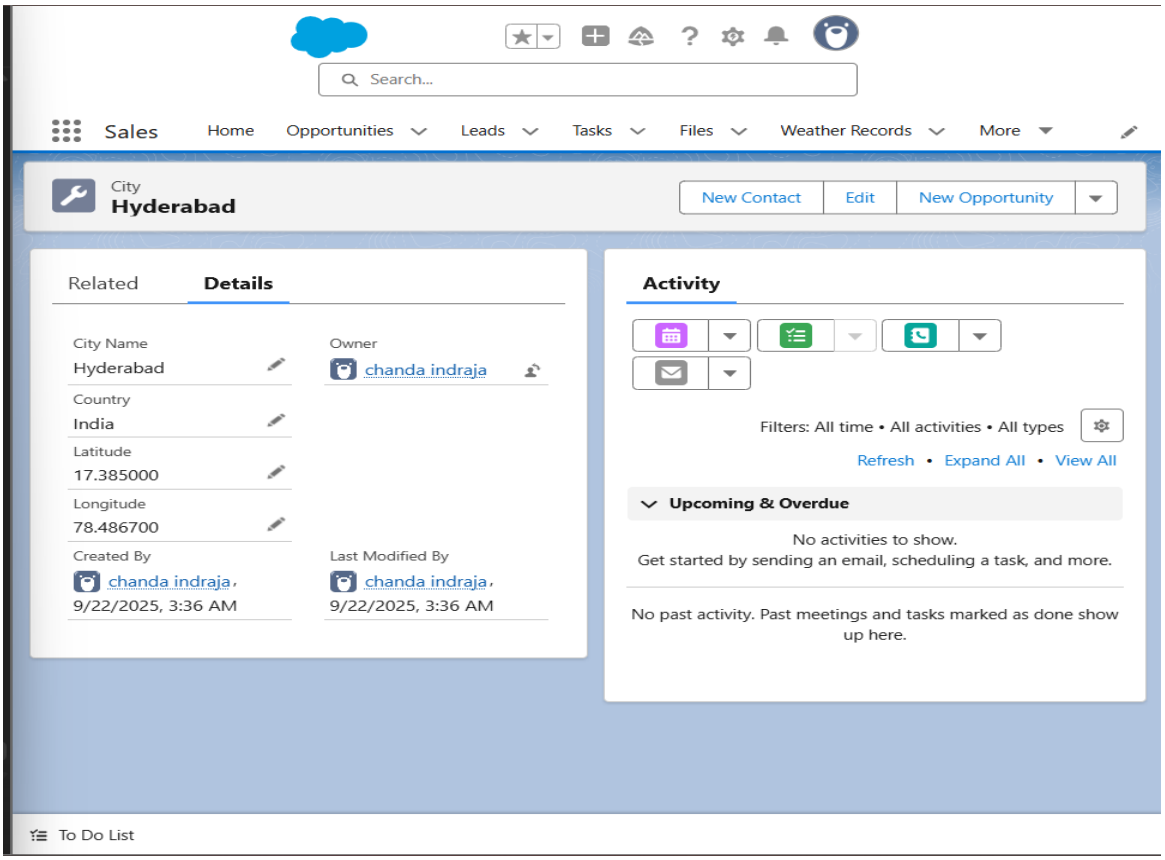
- **Object:** Weather Record
- **Criteria Name:** High Rain Probability
- **Condition:** `[Weather_Record__c].Rain_Probability__c > 80`
- **Immediate Action:** Update the Alert Level field to "High"

Core Features

- **Multiple Criteria Nodes:** Create separate decision branches for different conditions (e.g., Moderate or Low Rain Probability)

- **Immediate & Scheduled Actions:** Trigger field updates, email alerts, tasks, or invoke flows.
- **Integration Options:** Call an Apex class or launch an auto-launched flow for advanced logic.





4. Approval Process

Purpose
An Approval Process in Salesforce automates the routing of records to one or more users for review and sign-off.
It ensures that critical changes—such as pricing overrides, major discounts, or severe weather alerts—are properly authorized before finalization.

Real-Time Example – Severe Weather Approval

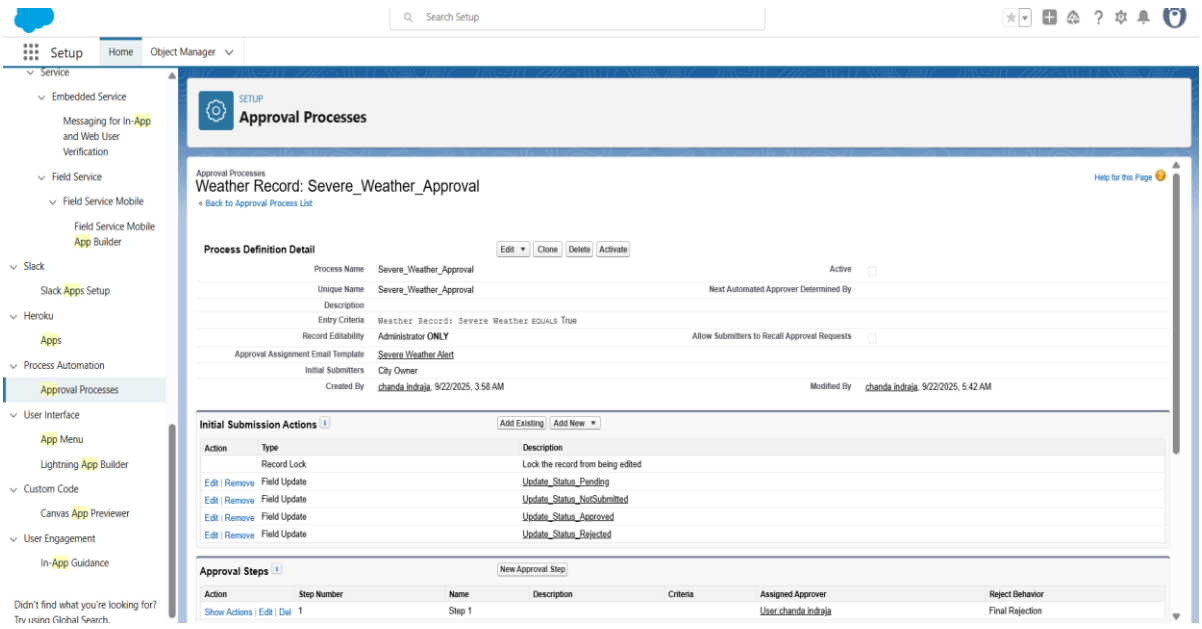
Business Requirement: When a Weather Record indicates **Severe Weather = TRUE**, it must be approved by the designated approver before any updates are allowed.

Key Configuration:

- **Process Name:** `Severe_Weather_Approval`
- **Entry Criteria:** `Weather_Record__c.Severe_Weather__c = TRUE`
- **Initial Submitter:** City Owner
- **Record Editability:** Administrator ONLY during approval
- **Approval Assignment Email Template:** *Severe Weather Alert*

Key Features

- **Record Locking:** Prevents edits while a record is under review.
- **Multi-Step Approvals:** Add sequential or parallel approval steps.
- **Automated Field Updates:** Update status fields (Pending, Approved, Rejected) automatically.
- **Notifications:** Email alerts to approvers and submitters.
-



5. Flow Builder (Screen, Record-Triggered, Scheduled, Auto-launched)

Purpose:

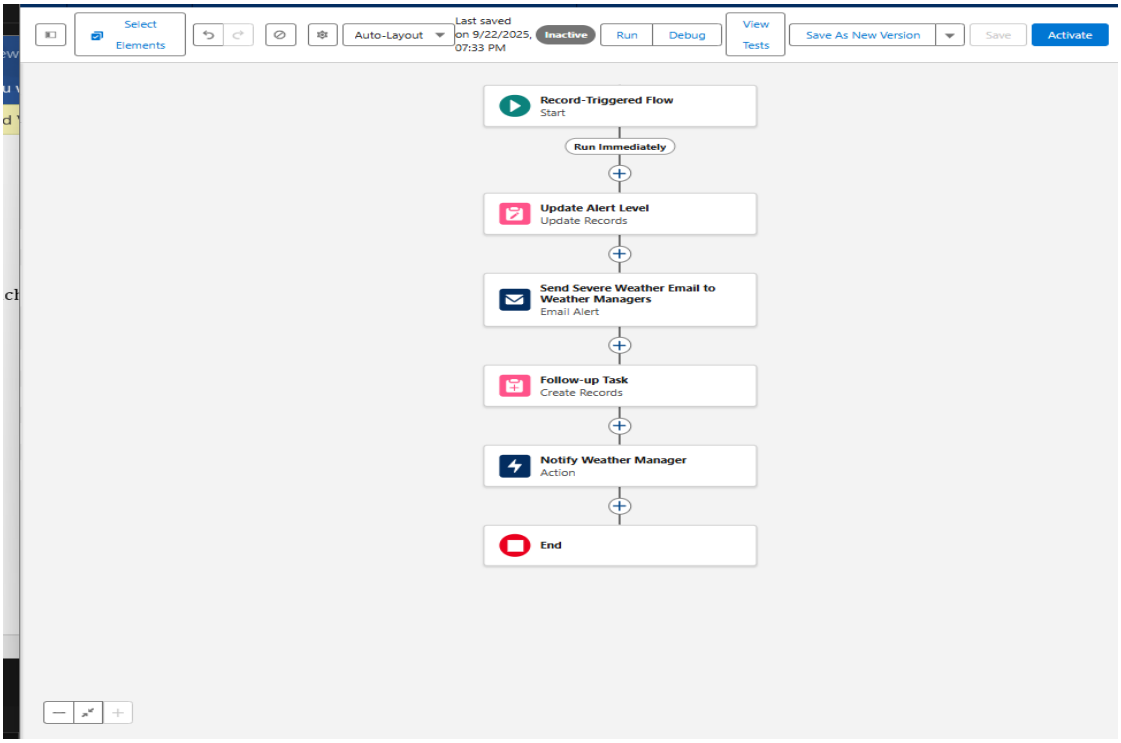
Flow Builder lets you automate business processes without code. It's flexible and can handle complex logic.

Types of Flows & Examples:

1. **Screen Flow** – Guides users step-by-step through a task.
   Example: Wizard to create a Case.
2. **Record-Triggered Flow** – Runs automatically when a record is created or updated.
   Example: Assign tasks when a new Lead is added.
3. **Scheduled Flow** – Runs at a specified time.
   Example: Send monthly renewal reminders.
4. **Auto-Launched Flow** – Runs in the background, often triggered by another process.
   Example: Update child records when a parent record changes..

Key Features:

- Automate tasks across Salesforce.
- Send **email notifications** to users.
- Update records, create new records, and perform calculations.
- Conditional logic and loops supported.



6.Email Alerts

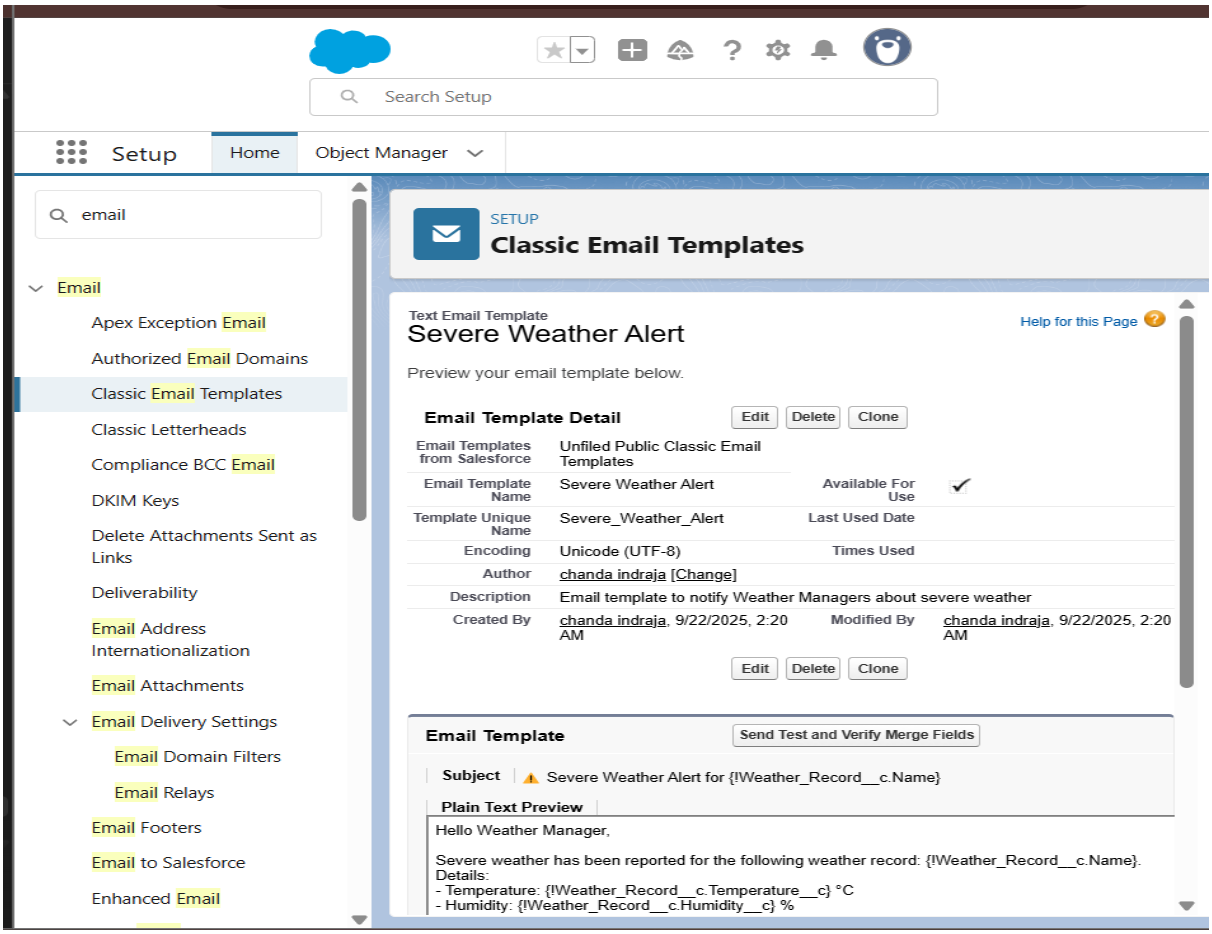
Purpose:
Automatically send email notifications based on specific actions or criteria, without manual intervention.

Key Features:

- Uses **Email Templates** (Classic or Lightning) to standardize messages.
- Can be triggered by **Workflow Rules**, **Process Builder**, or **Flow Builder**.
- Can notify multiple recipients: Users, Roles, Contacts, or email addresses.
- Supports **dynamic fields** to personalize emails (e.g., customer name, order number).
- Works with **approvals**, **case updates**, **lead assignment**, and more.

Real-Time Examples:

- Notify a customer when an order is shipped.
- Alert a manager when a high-value opportunity is created.
- Send reminders for overdue tasks or approvals.



7.Field Updates

Purpose:

- Maintain data accuracy and consistency.
- Save user time by reducing manual updates.
- Trigger changes that influence other automation (workflows, flows, approval processes).

Use Cases:

- **Opportunity Management:** Automatically update the **Opportunity Stage** to "Closed Lost" when the "Competitor" field is filled in.
- **Case Management:** Update a **Case Status** to "Escalated" when the case priority is set to "High."
- **Lead Qualification:** Mark a **Lead Source** field to "Web" if the lead was created from a web form.

Benefits:

- Eliminates human errors in repetitive updates.
- Provides real-time record changes.
- Improves reporting accuracy.

8.Tasks

Definition:

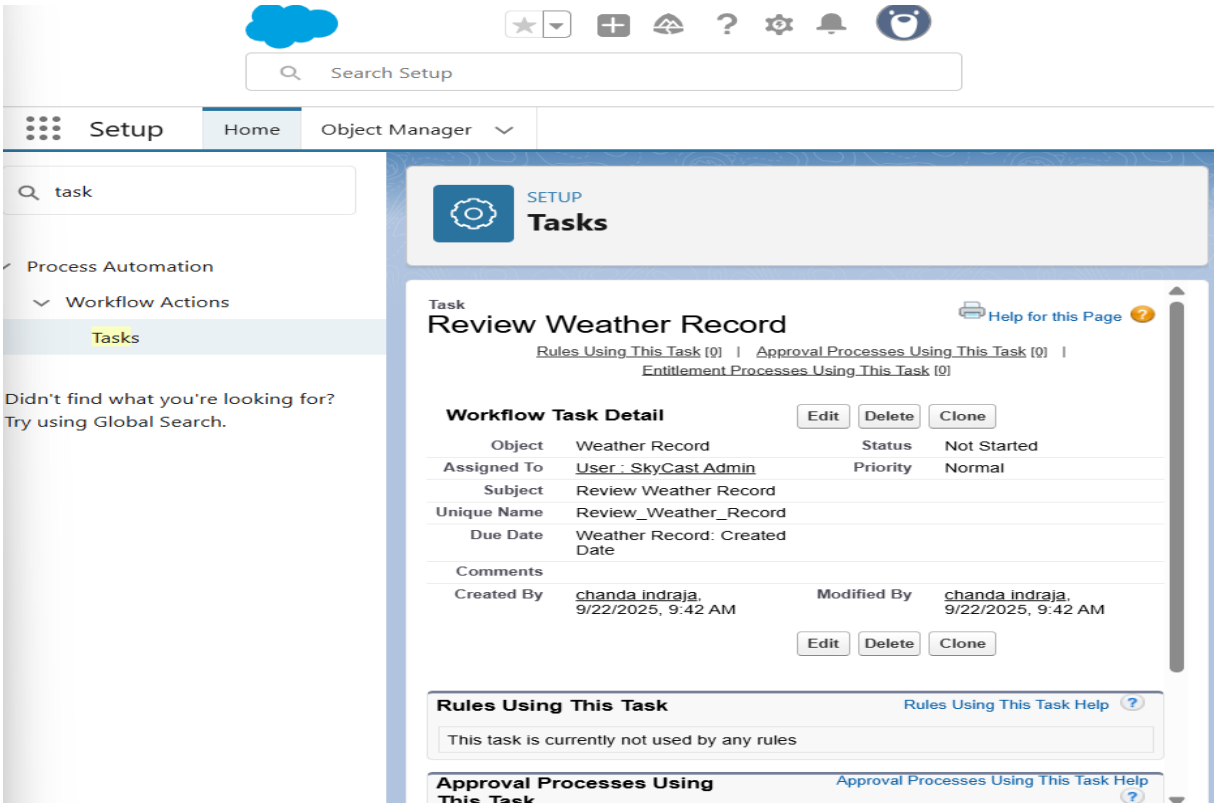
Tasks are to-do items that Salesforce can automatically create for users when a specific trigger condition is met. These tasks help guide sales and service teams by ensuring they follow up on critical actions.

Purpose:

- Assign follow-up actions to users at the right time.
- Keep sales cycles moving and avoid missed opportunities.
- Standardize customer engagement processes.

Use Cases:

- **Lead Follow-Up:** When a new high-value lead is created, automatically assign a task for the sales rep to call within 24 hours.
- **Renewals:** When an opportunity is 30 days away from its close date, assign a task for the account manager to reach out to the client.
- **Customer Support:** When a case is reopened, create a task for a support agent to review it immediately.
- Reduces the risk of missed follow-ups.



9.Custom Notifications

Purpose:

- Improve response time for urgent matters.
- Keep users informed without relying solely on emails.
- Provide flexibility in how users receive alerts (desktop, mobile).

Use Cases:

- **Sales Alerts:** Notify sales reps instantly when a “Hot Lead” is assigned to them.
- **Case Management:** Alert support managers when a VIP customer’s case is escalated.
- **Approvals:** Send a notification to a manager when a discount approval request is pending.

.

Phase 5: Apex Programming (Developer)

1.Classes & Objects

Classes in Apex

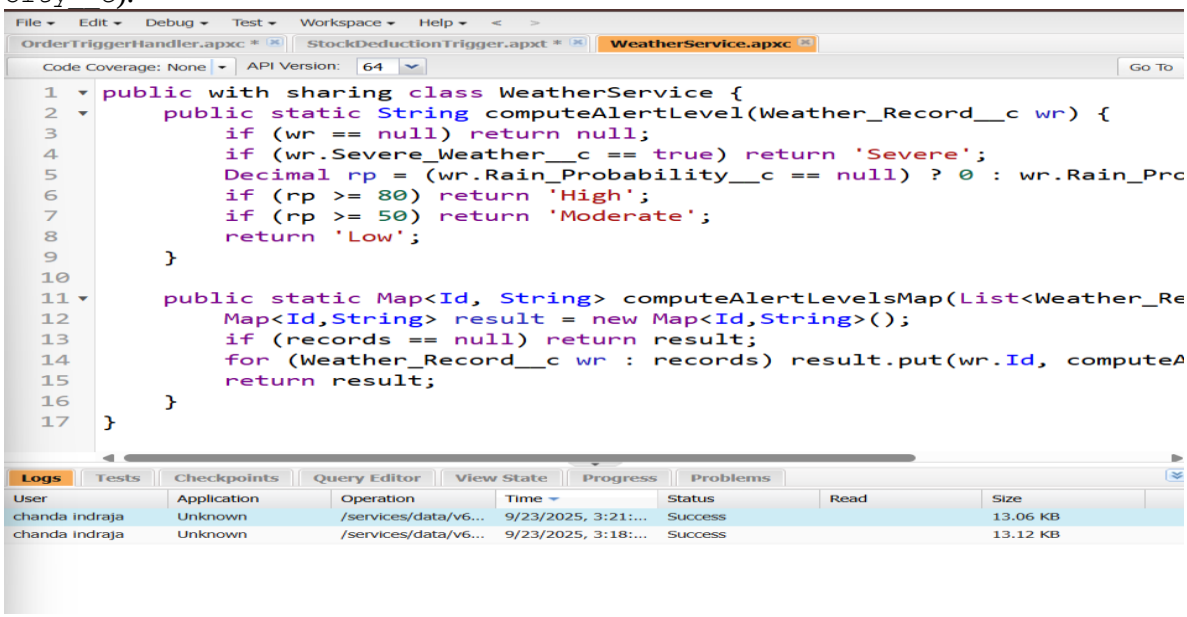
- **Definition:**
  Classes are blueprints or templates used to create objects in Salesforce. They encapsulate **business logic**, **data structures**, and **methods** that define the behavior of an object.
- **Purpose:**
  - Centralize code for easier maintenance and reuse.
  - Separate complex logic from triggers to follow **best practices**.
  - Facilitate **unit testing** of business logic independently of UI or triggers.

Key Components of a Class:

- **Properties (Variables):** Hold data (e.g., temperature, humidity).
- **Methods (Functions):** Contain logic to manipulate or fetch data.
- **Constructors:** Special methods used to initialize objects.
- **Access Modifiers:** `public`, `private`, `global` – control visibility.
- **Example – Weather Project Objects:**

Types of Objects:

- **Standard Objects:** Predefined by Salesforce (e.g., Account, Contact).
- **Custom Objects:** Created specifically for your project (e.g., `Weather_Record__c`, `City__c`).



2.Apex Triggers (before/after insert/update/delete)

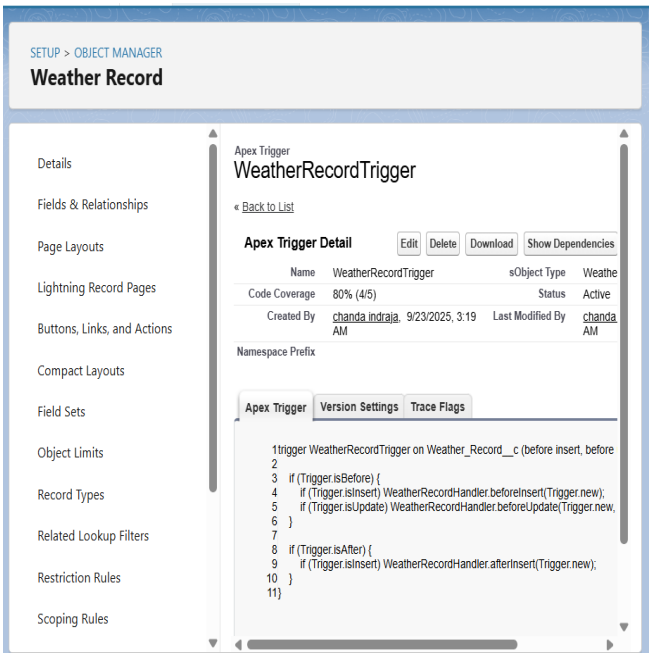
1. What is an Apex Trigger

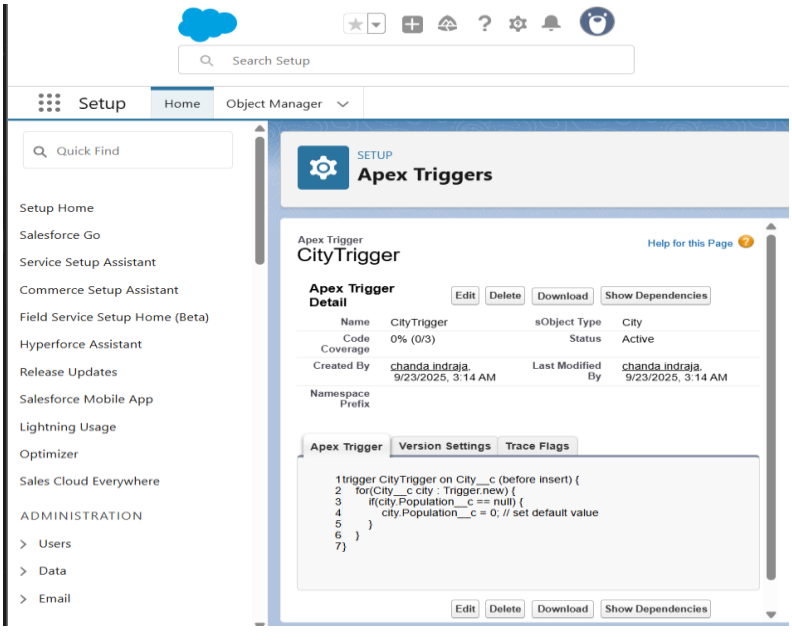
- **Definition:**
  An Apex Trigger is a piece of Apex code that executes **before or after a record is inserted, updated, deleted, or undeleted** in Salesforce.
- **Purpose:**
  - Automate business processes.
  - Perform operations that cannot be done with declarative tools like Workflow or Process Builder.
  - Ensure data integrity and enforce complex logic.

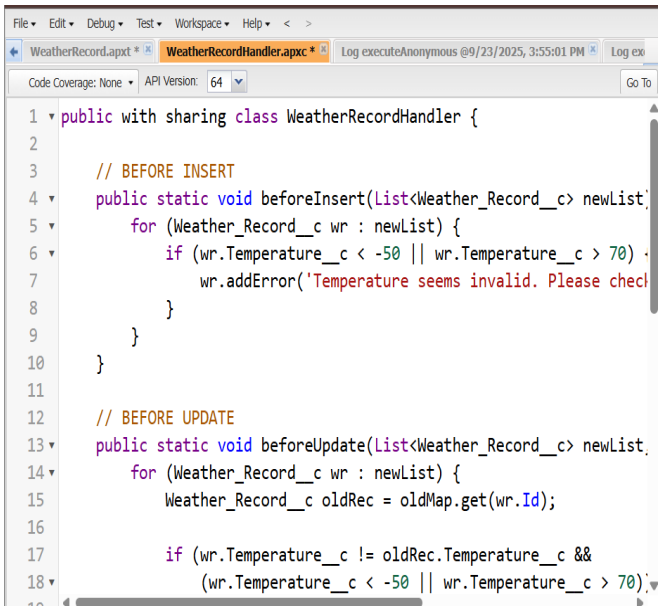
2. Trigger Events

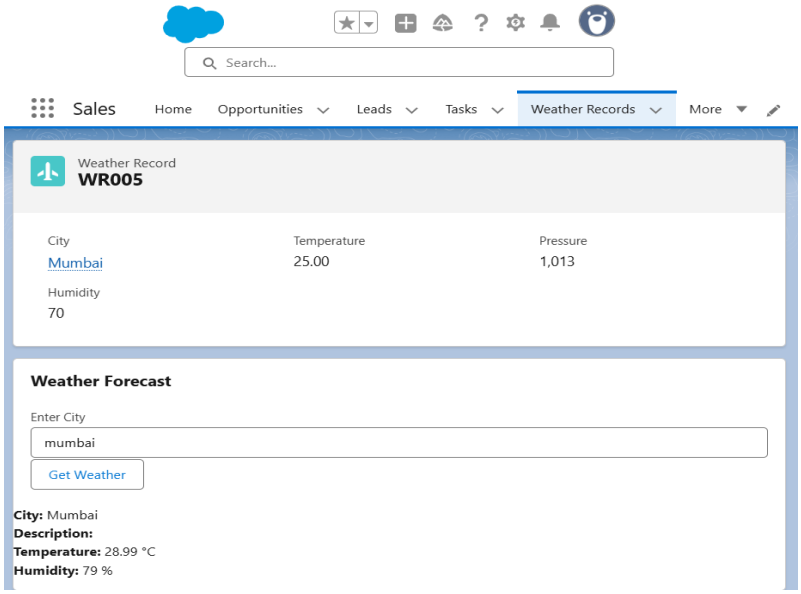
Triggers can be executed **before** or **after** DML (Data Manipulation Language) operations:

1. **Before Triggers:**
   - Executed **before** a record is saved to the database.
   - Used to **validate or modify field values** before insertion or update.
   - Example: Automatically setting default temperature for new weather records.
2. **After Triggers:**
   - Executed **after** a record is saved to the database.
   - Used for actions that require **record IDs** or interactions with other objects.
   - Example: Creating related Weather_Record__c after a City__c is added.









3.Trigger Design Pattern

1. What is a Trigger Design Pattern

- **Definition:**
  A Trigger Design Pattern is a **structured way to write Apex triggers** that ensures clean, maintainable, and scalable code.
- **Purpose:**
  - Avoid messy triggers with duplicated logic.
  - Handle **bulk operations** efficiently.
  - Separate trigger logic from business logic (delegated to handler classes).

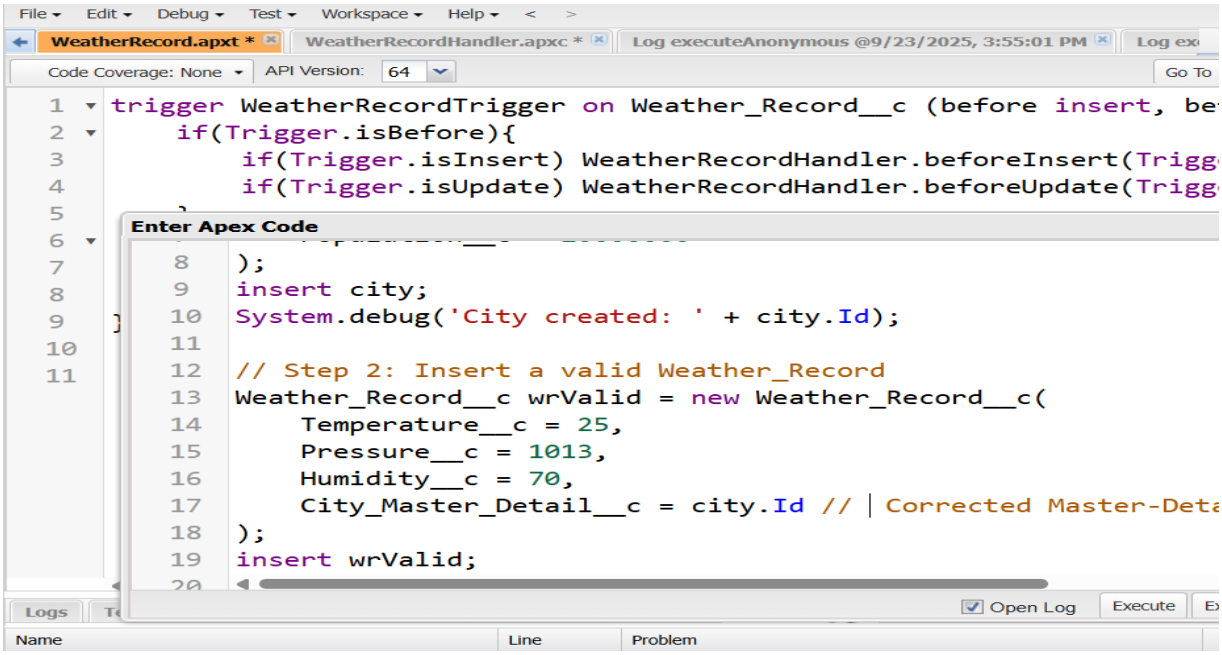
2. Key Components

1. **Trigger:**
   - The entry point that listens to events (insert, update, delete).
   - Should be **slim**; only calls the handler class.
2. **Handler Class:**
   - Contains the main business logic (create, update, delete related records).
   - Bulkified to handle multiple records at once.
3. **Utility or Service Classes (optional):**
   - For API calls, complex calculations, or reusable methods.

3. Benefits

- Easy to maintain and update.

- Single trigger per object.
- Supports bulk operations for multiple records.
- Reduces errors and improves readability.



4.SOQL & SOSL

1. SOQL (Salesforce Object Query Language)

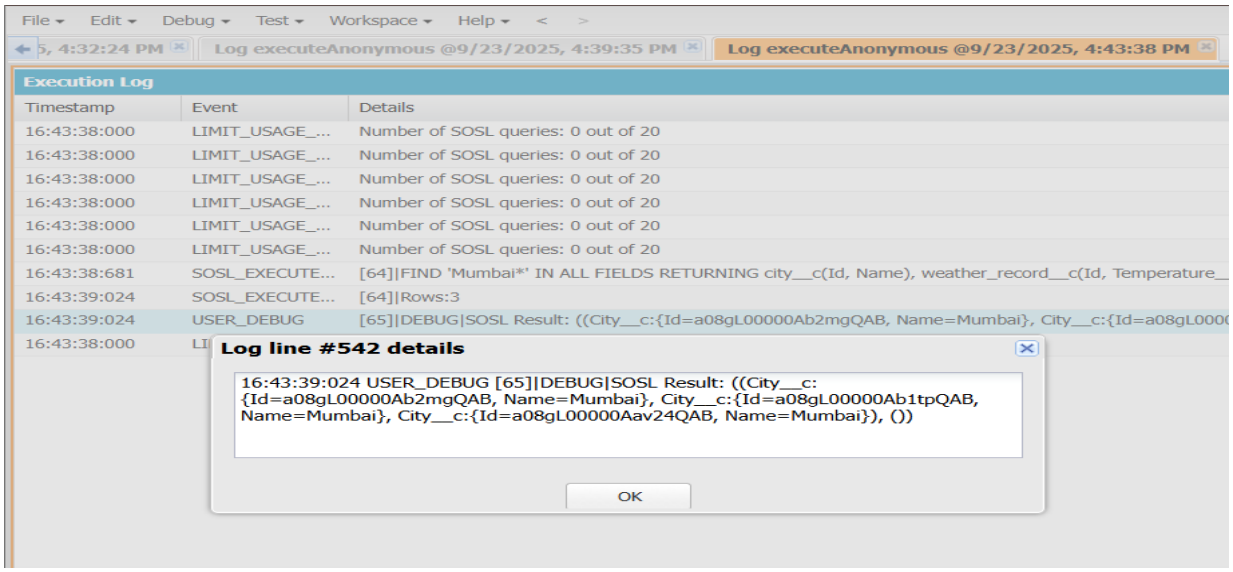
- **Definition:**
  SOQL is used to **query records from Salesforce objects**.
- **Purpose:**
  - Retrieve specific fields and records.
  - Filter and order data using `WHERE`, `ORDER BY`, `LIMIT`.
- **Example – Fetch Weather Records by City:**

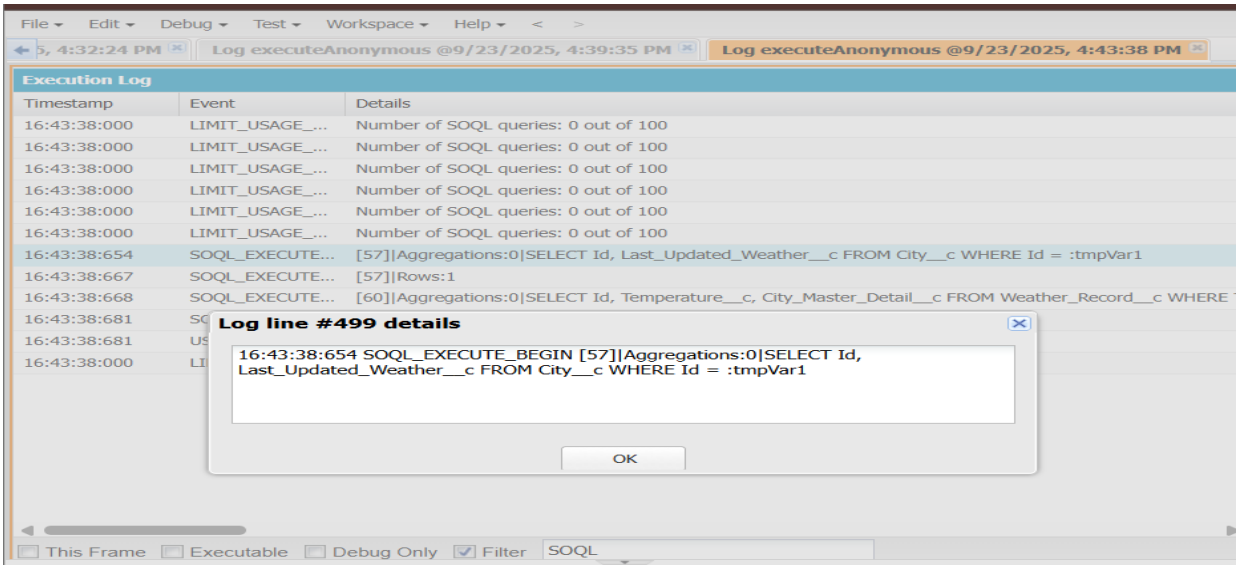
```
List<Weather_Record__c> records = [

 SELECT Id, Temperature__c, Humidity__c

 FROM Weather_Record__c

 WHERE City__c = :cityId

];
```

2. SOSL (Salesforce Object Search Language)

- **Definition:**
  SOSL is used to **search text, email, and phone fields** across multiple objects at once.
- **Purpose:**
  - Find records when you **don't know the exact field** where the data exists.
  - Useful for **search functionality** in Lightning Web Components (LWC) or Apex.
- **Example – Search Weather Records by Keyword:**

  List<List<SObject>> searchResults = [FIND 'rain*' IN ALL FIELDS RETURNING Weather_Record__c(Name, Description__c)];





5.Collections: List, Set, Map

Overview

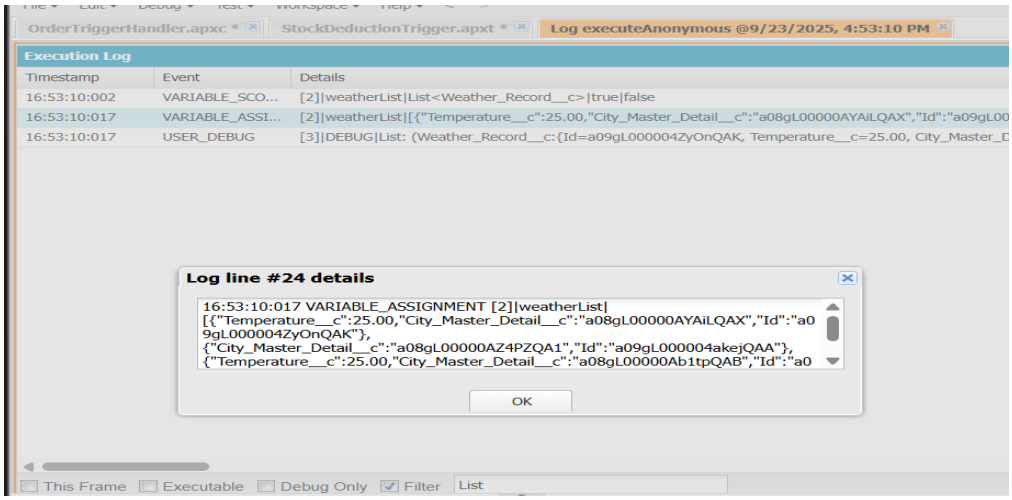
Collections in Apex are variables that can store multiple records or values together. They are widely used for **bulk data handling** in Salesforce.

Types of Collections

- **List**
  - Ordered collection.
  - Can contain duplicate values.
  - Index starts from 0.
  - Example use: Store multiple city names.

    List<String> cities = new List<String>{'Delhi', 'Mumbai', 'Chennai'};

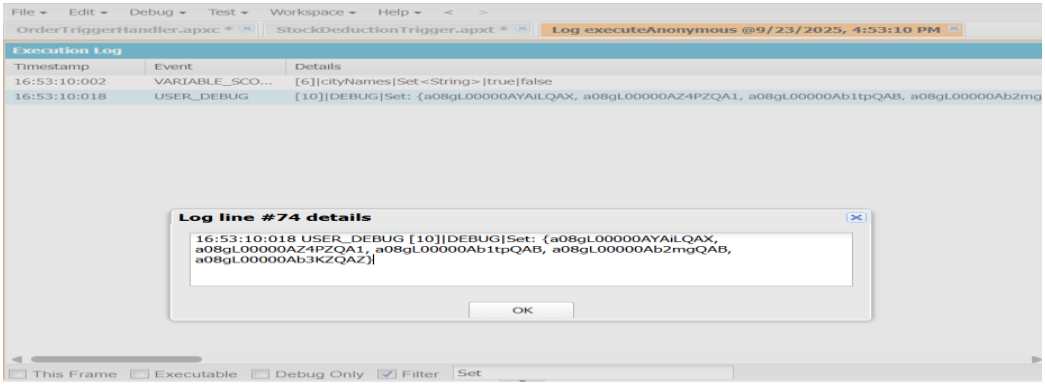
    System.debug(cities[0]); // Delhi



- **Set**
  - Unordered collection.
  - Does not allow duplicate values.
  - Example use: Store unique city names to avoid duplicates.

    Set<String> cities = new Set<String>{'Delhi', 'Mumbai', 'Delhi'};

    System.debug(cities.size());



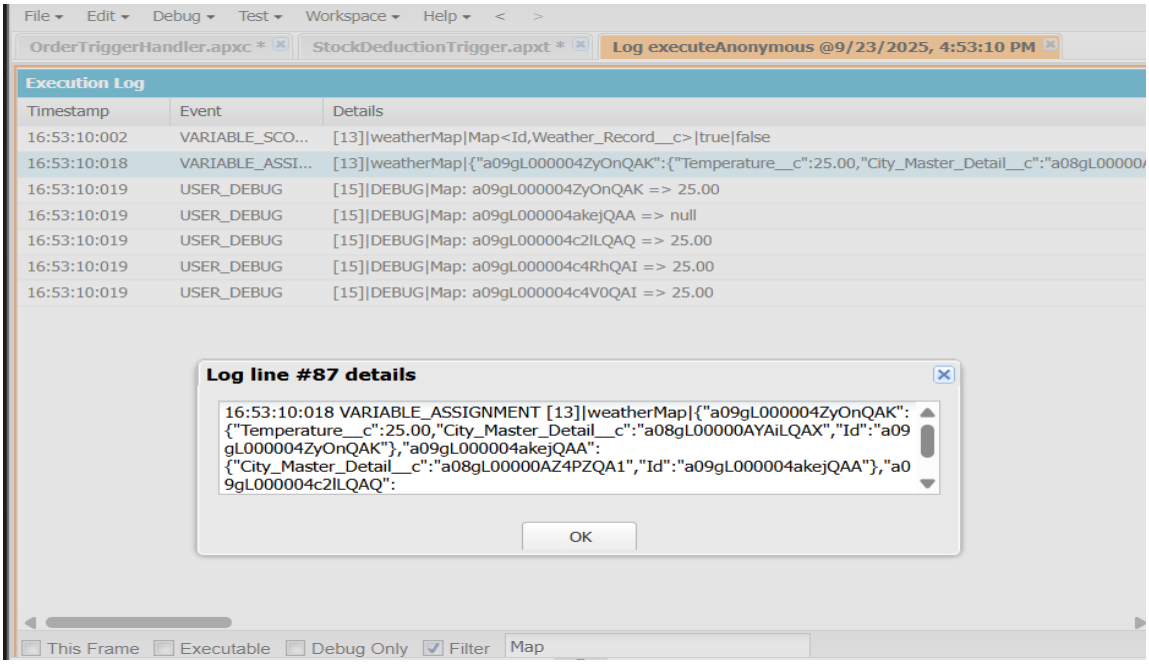
- **Map**
  - Collection of key–value pairs.
  - Each key is unique, but values can repeat.
  - Example use: Map city names to their temperature values.

    Map<String, Integer> cityTemp = new Map<String, Integer>();

    cityTemp.put('Delhi', 32);

    cityTemp.put('Mumbai', 29);

    System.debug(cityTemp.get('Delhi')); // 32



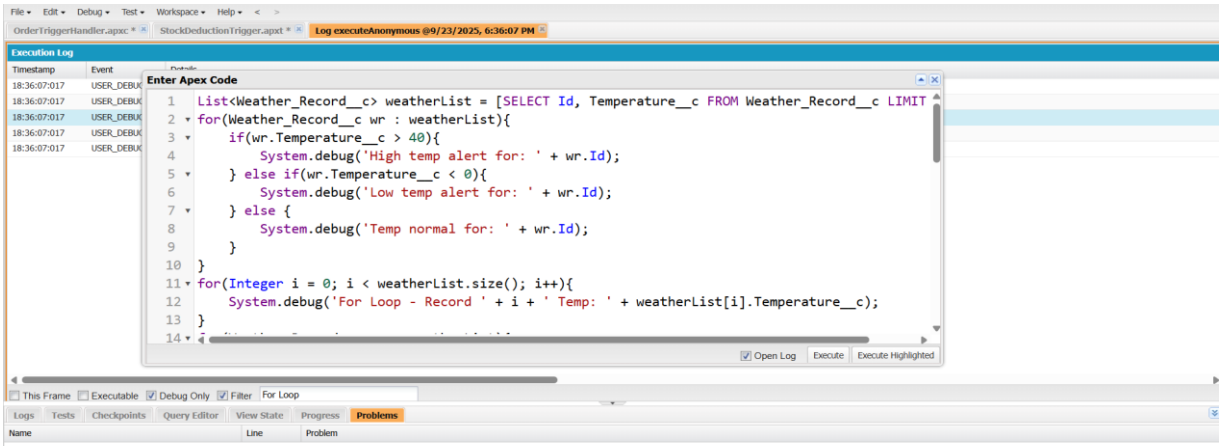
6.Control Statements

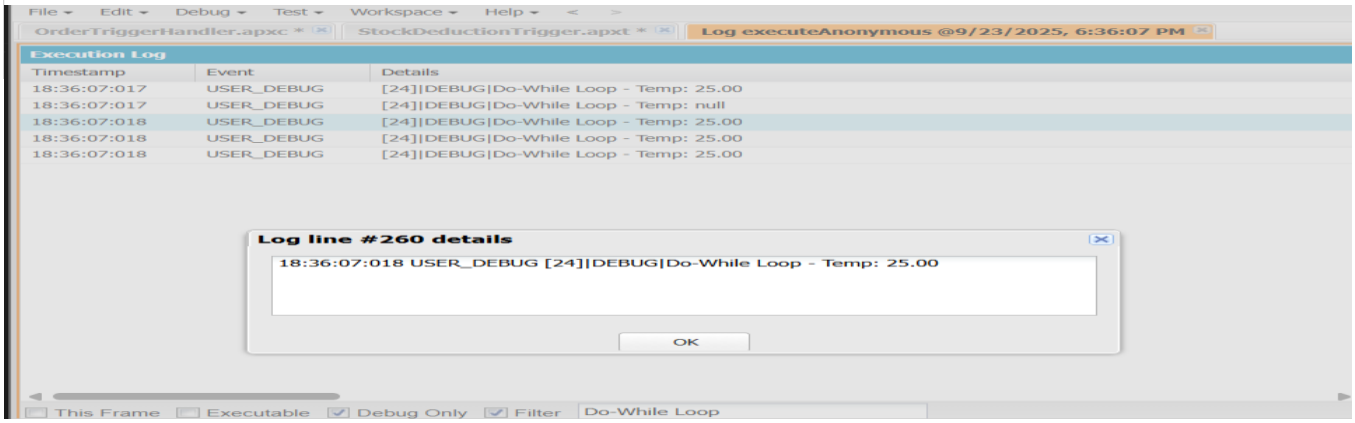
Overview

Control statements define the **flow of logic** in Apex programs.

Types

- **Conditional Statements**
  - `if-else` → Executes code based on conditions.
  - `switch` → Cleaner alternative to multiple if-else statements.
- **Looping Statements**
  - `for` → Iterates through records for a fixed number of times.
  - `for-each` → Simplified loop for collections.
  - `while` → Repeats as long as condition is true.
- **Break & Continue**
  - `break` → Immediately exits a loop.
  - `continue` → Skips current iteration and moves to next





7. Batch Apex

Overview

Batch Apex is used to process **large volumes of data** in manageable chunks, without hitting governor limits.

Why It's Needed

- Salesforce has strict **limits** on processing records.
- Batch Apex splits large jobs into smaller sets and processes them asynchronously.

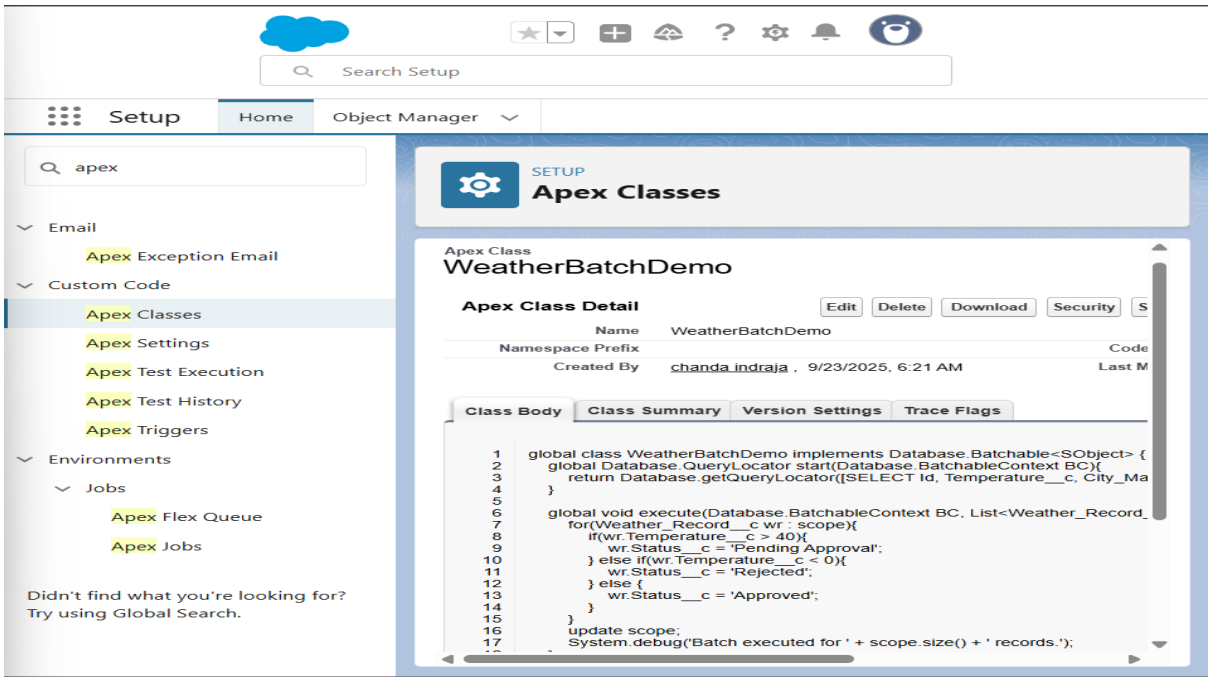
Structure

A Batch Apex class has three main methods:

1. **Start** → Collects the records to be processed.
2. **Execute** → Runs on each batch of records.
3. **Finish** → Performs final tasks once all batches are complete.

Benefits

- Handles millions of records safely.
- Runs in the background (asynchronous).
- Useful for scheduled jobs, data cleanup, and API-based data processing.



8.Queueable Apex

Overview

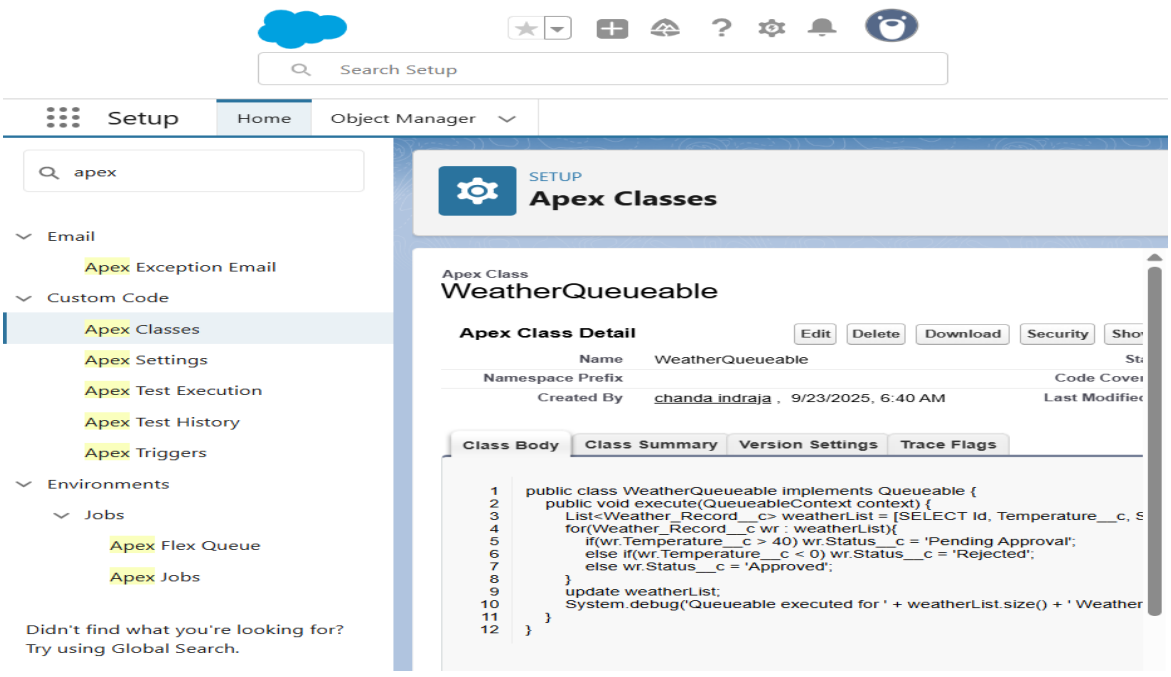
Queueable Apex is used to run **asynchronous jobs** in Salesforce. It is similar to `Future` methods but more powerful.

Key Features

- Allows **chaining** of jobs (one queueable can enqueue another).
- Supports **complex data types** like sObjects and custom classes as parameters.
- More flexible than `Future` methods.

Benefits

- Handles long-running processes without blocking users.
- Can process large datasets in the background.
- Easy to monitor in the **Apex Jobs** page.
- **Interface Used:** `System.Queueable`.
- Can process **more records than Future methods**, but still within governor limits.
- Can be **monitored, paused, and retried** from the **Apex Jobs** page.
- Supports **complex data types** as parameters (Lists, sObjects, Maps).
- Jobs can be **chained** → one job automatically starts another.
- Easier to test compared to `Future` methods.



9.Scheduled Apex

1. Purpose

- Automates tasks at **specific times or intervals**.
- Removes the need for manual execution of repetitive jobs.
- Useful for **maintenance, integration, and data refreshes**.

2. Features

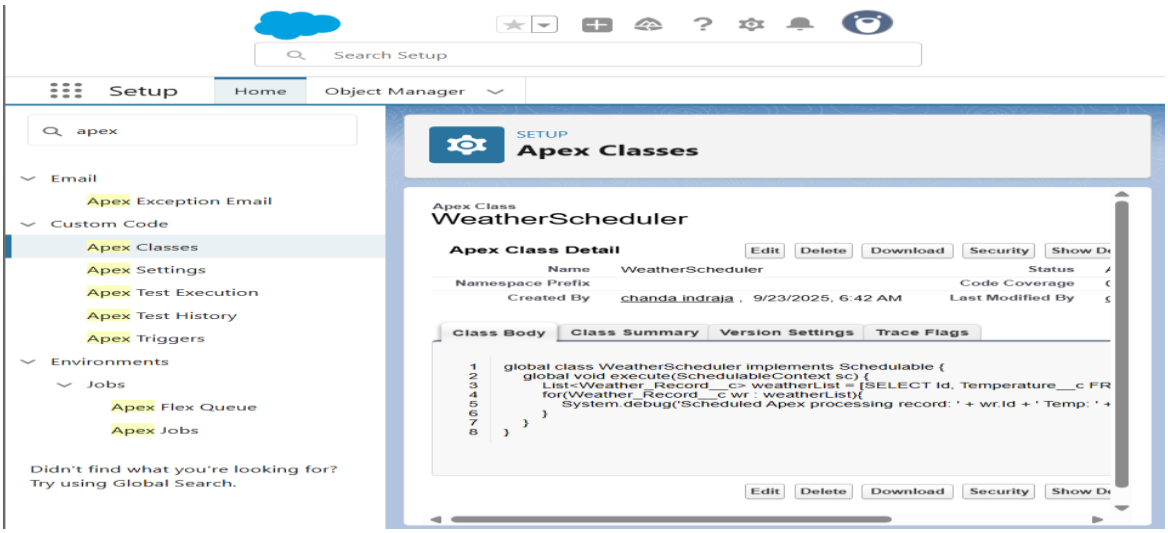
- Runs in the background (asynchronous execution).
- Can be scheduled via **Setup UI** or **System.schedule() method**.
- Uses **cron expressions** to define schedule (similar to UNIX cron jobs).
- Can call **Batch Apex** or **Queueable Apex** inside scheduled jobs.

3. Use Cases

- Daily or nightly updates (e.g., refresh weather data for all cities).
- Weekly or monthly reports (e.g., send summary of weather trends).
- Data cleanup jobs (e.g., remove old or duplicate weather records).
- Scheduled API callouts (e.g., fetch data from external weather API).

4. Cron Expression Format

- Structure: **Seconds Minutes Hours Day_of_Month Month Day_of_Week Year (optional)**
- Example:
  - `"0 0 0 * * ?"` → Runs daily at midnight.
  - `"0 0 12 ? * MON"` → Runs every Monday at 12 PM.



10.Future Methods

Overview

Future methods are used to run processes **asynchronously** in the background, especially for **callouts to external systems**.

Purpose

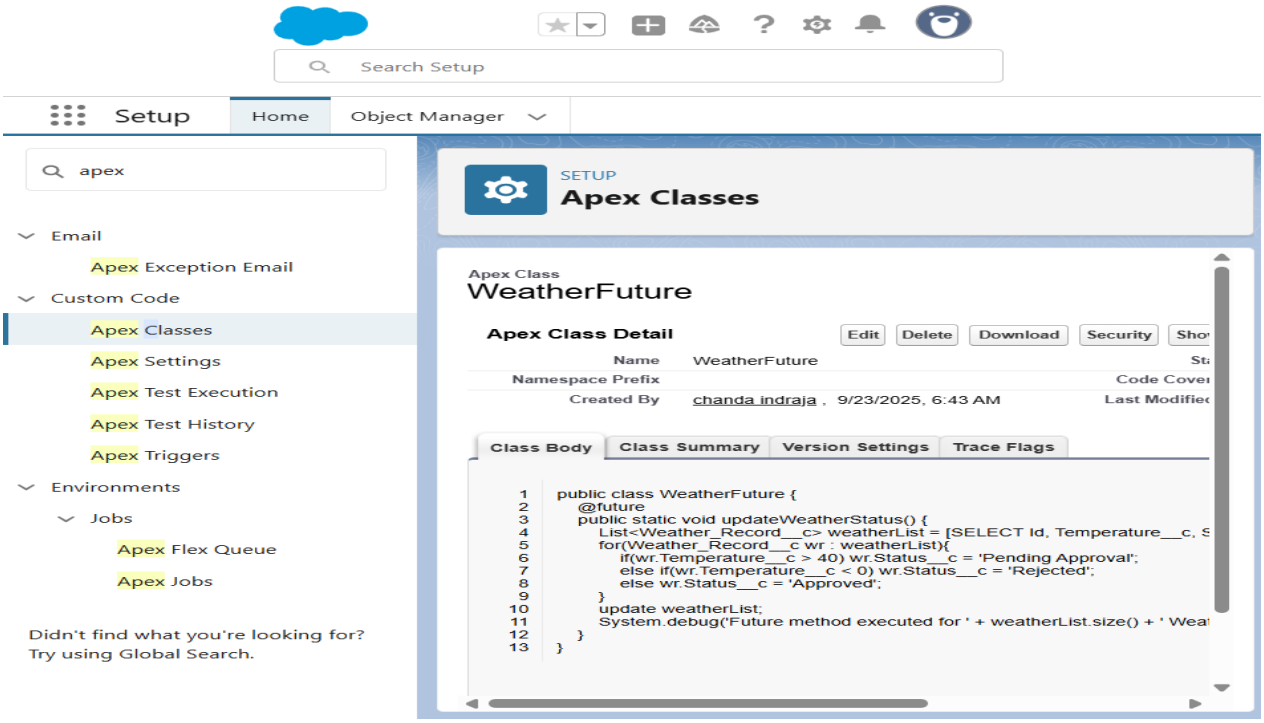
- Run operations **asynchronously** (in the background).
- Ideal for long-running tasks or external callouts.

Features

- Annotated with `@future`.
- Executes in a **separate thread** from the main transaction.
- Can be used for **callouts, email sending, and data updates**.

Use Cases

- Making **HTTP callouts** to external systems.
- Performing **resource-intensive calculations**.
- Allows integration with external APIs without blocking user actions.
- Helps avoid governor limits during synchronous execution.



11. Exception Handling

Overview

Exception handling in Apex is used to **catch errors gracefully** and prevent program crashes.

Types of Exceptions

- **System exceptions:** Thrown by Salesforce platform (e.g., NullPointerException, DmlException).
- **Custom exceptions:** Defined by developers for specific business scenarios.

Mechanism

- Uses `try`, `catch`, and `finally` blocks.
    - **try:** Code that may cause exception.
    - **catch:** Handles the exception.
    - **finally:** Always executes (cleanup).

Benefits

- Prevents application failures.
- Provides user-friendly error messages.
- Ensures smooth transaction rollbacks.

12.Test Classes

1. Purpose

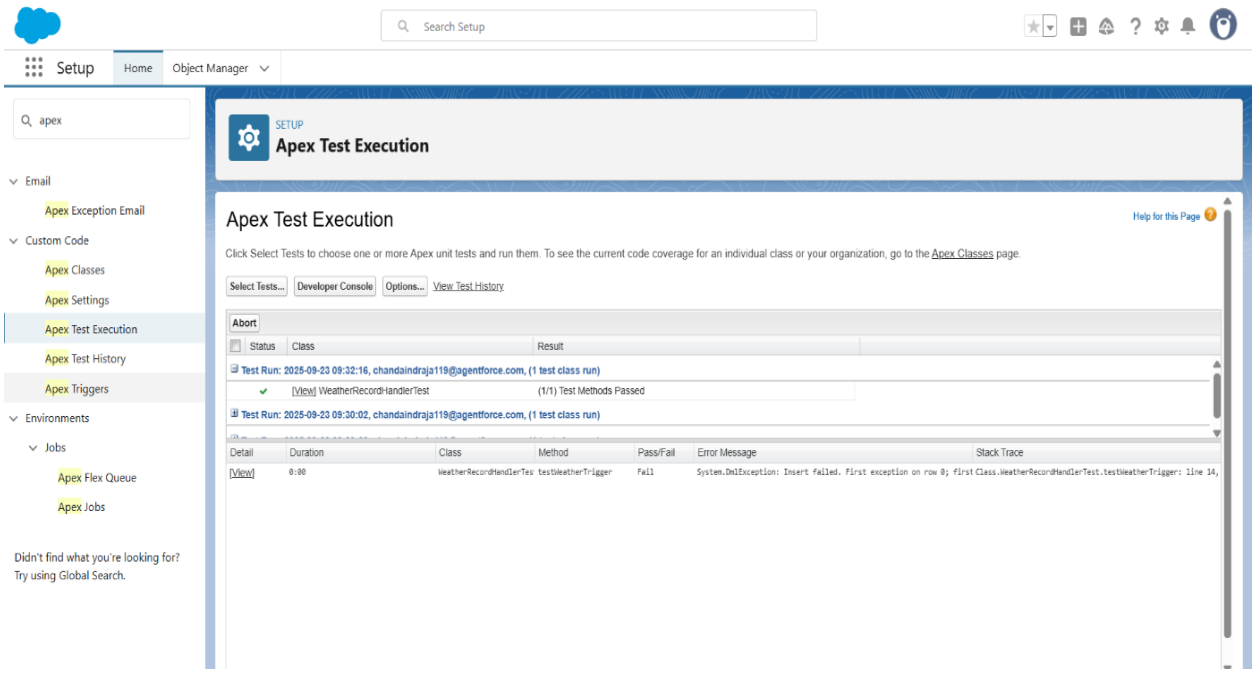
- Validate Apex code functionality.
- Ensure code is **deployable to production** (minimum 75% coverage required).
- Prevent bugs and unintended behavior in real-time usage.

2. Features

- Written using `@isTest` annotation.
- Run in a **test context** (no impact on org data).
- Support **positive, negative, and bulk testing**.

3. Use Cases

- Verifying **trigger logic** when records are inserted or updated.
- Testing **Batch, Queueable, and Future methods**.
- Ensuring **custom validations and exceptions** work as expected.



13.Asynchronous Processing

1. Purpose

- Run long-running or resource-intensive tasks **outside the main transaction**.
- Improves system performance and **avoids hitting governor limits**.

2. Types

- **Future Methods** → Lightweight background tasks.
- **Queueable Apex** → Supports job chaining and monitoring.
- **Batch Apex** → Processes millions of records in chunks.
- **Scheduled Apex** → Runs jobs at defined times.

3. Use Cases

- Making **API callouts** to weather services.
- Performing **large data updates** (e.g., refreshing weather history).
- Running **nightly or weekly scheduled jobs**.
- Sending **bulk notifications or emails**
- Helps handle **large data volumes efficiently**
- Provides **flexibility** in executing jobs.

Phase 6: User Interface Development

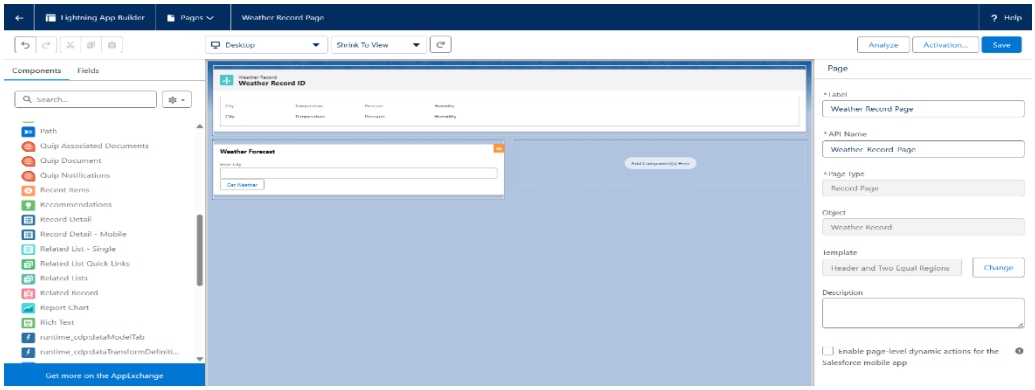
1.Lightning App Builder

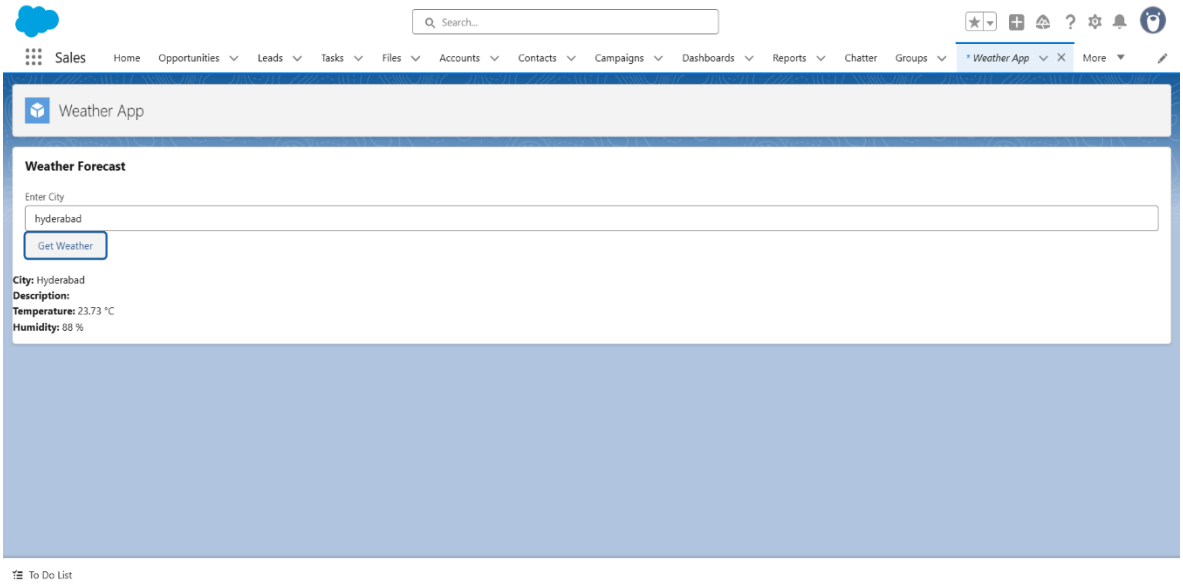
Purpose

- **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.
   -





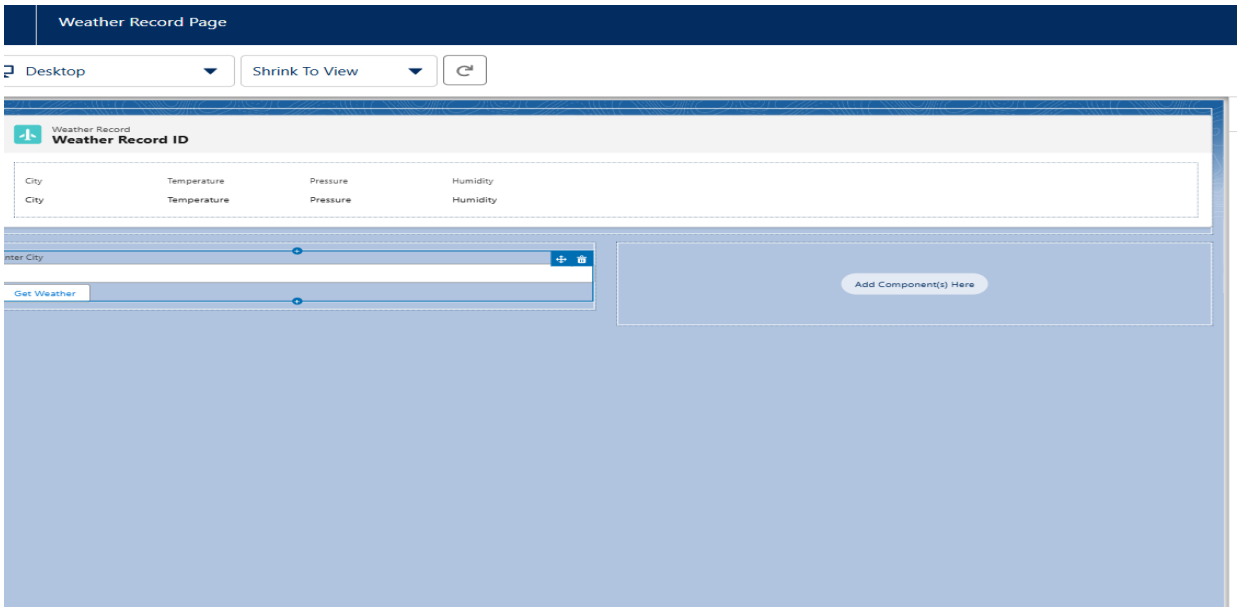
2. Record Pages

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.



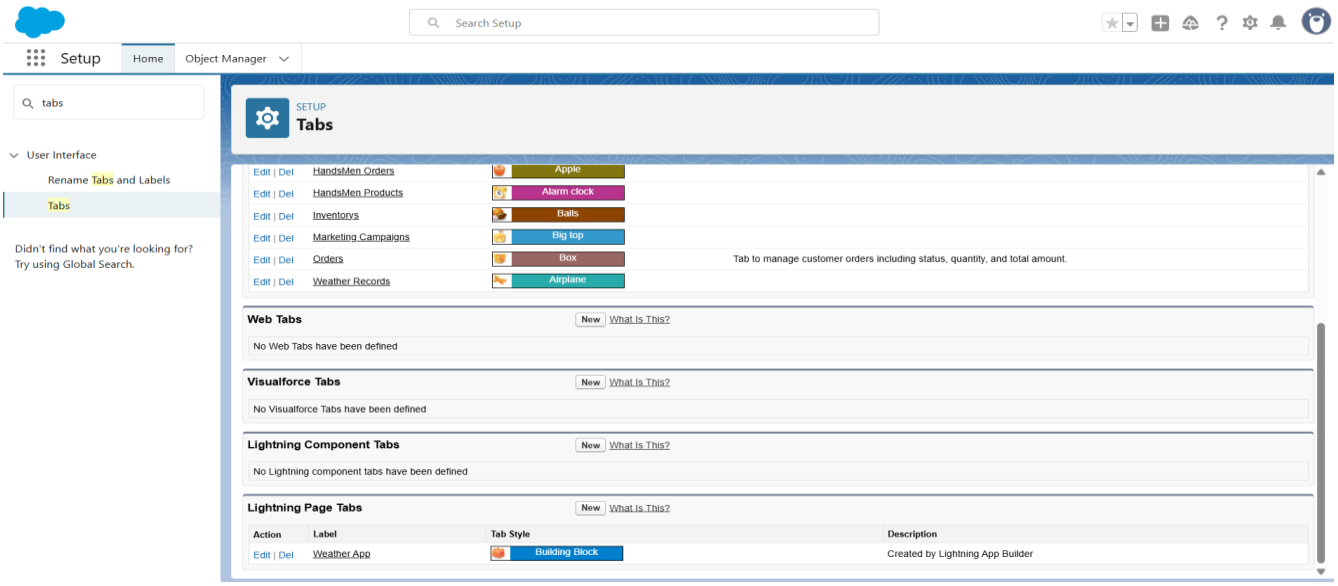
3. Tabs

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.



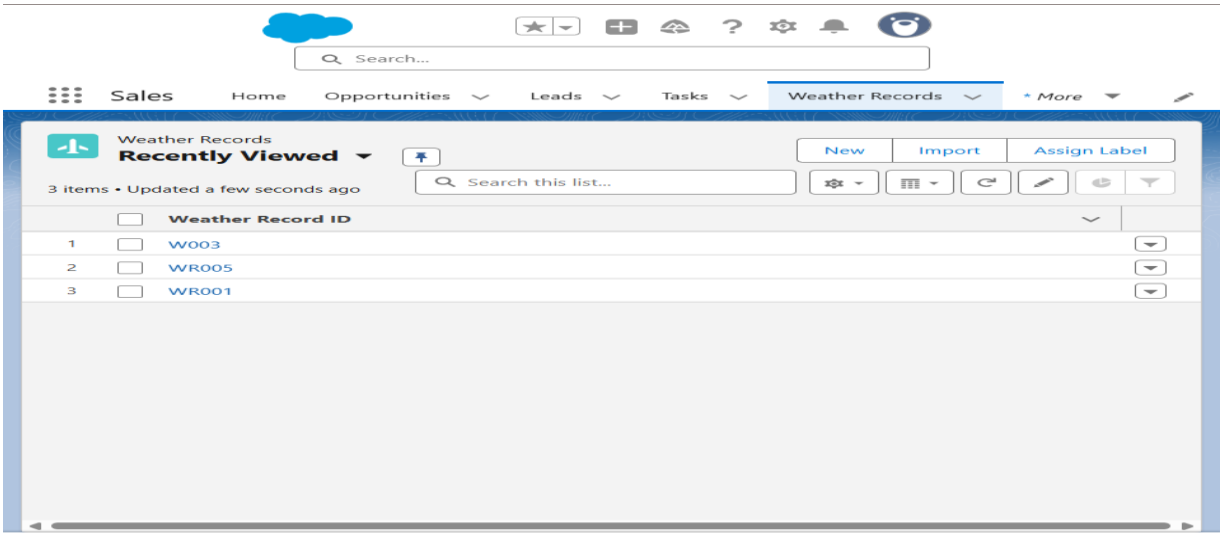
4.Home Page Layouts

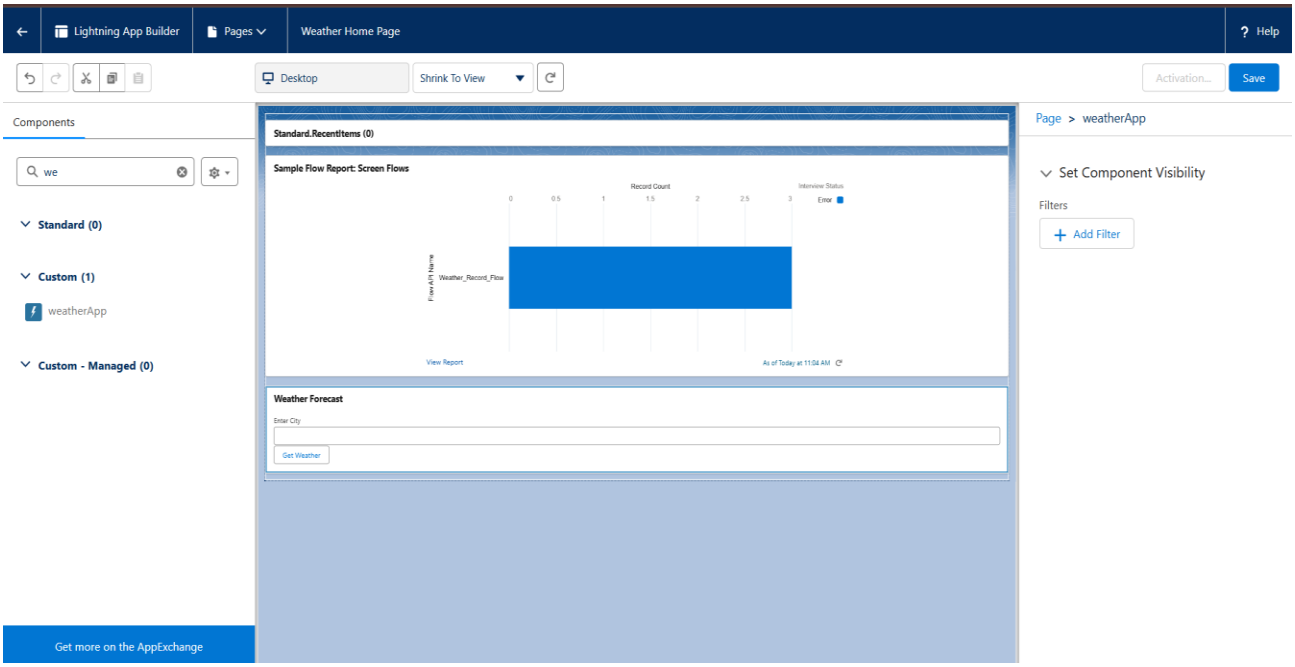
Overview

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.

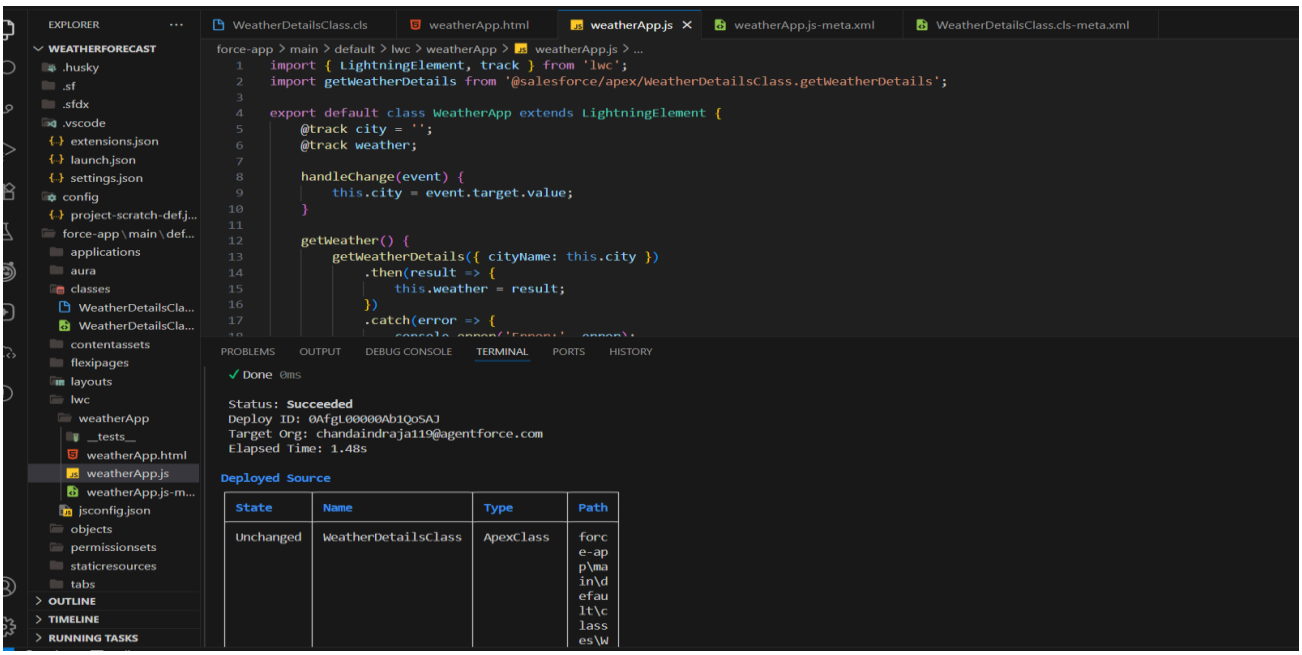
6. Lightning Web Components (LWC)

Overview

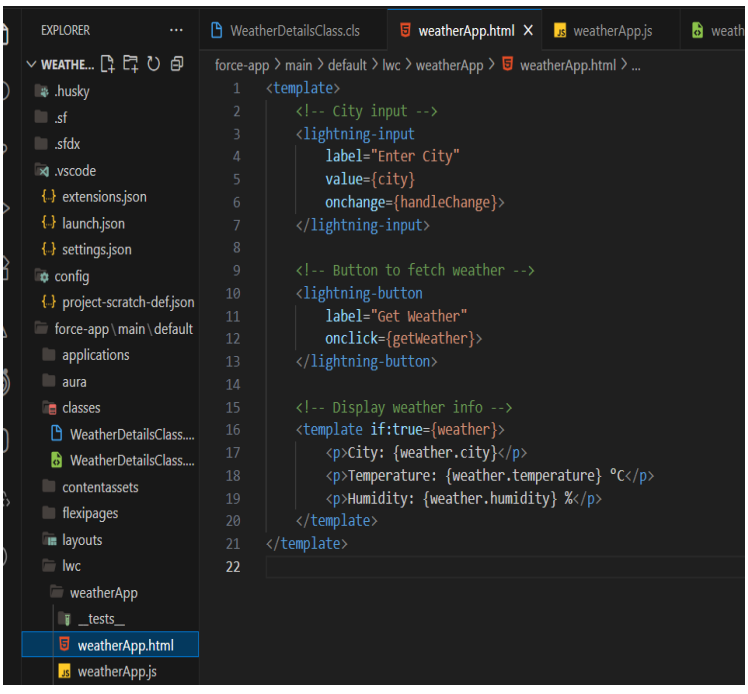
- Lightning Web Components (LWC) is **Salesforce's modern UI framework** for building fast, reusable, and interactive components.
- Based on **modern web standards** (HTML, JavaScript, CSS).
- LWCs can run on **Salesforce desktop and mobile**, integrate with **Apex**, and interact with other components.

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.

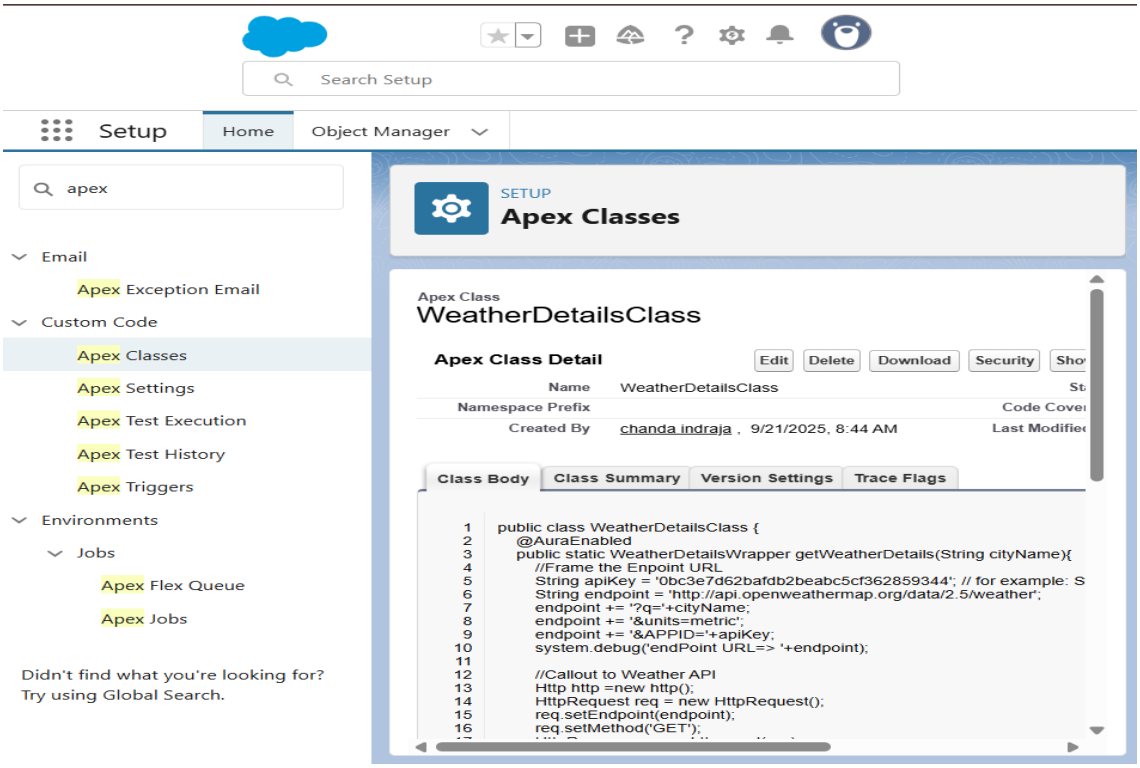
Step-by-Step Implementation

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.

Phase 7: Integration & External Access

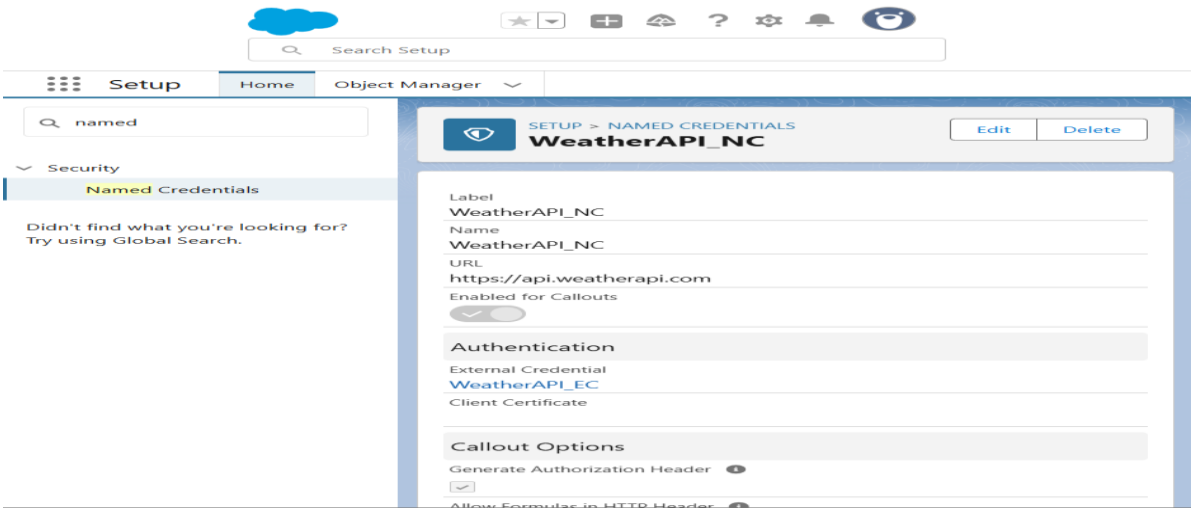
1. Named Credentials

Purpose

- Securely store authentication information for external APIs.
- Simplifies API callouts by avoiding hardcoding credentials in Apex code.
- Manages authentication protocols such as OAuth 2.0, Password Authentication, or Named Principal.
- Enhances security by controlling access centrally.

Key Components

- **Label & Name:** User-friendly label and unique API name for reference in Apex.
- **URL:** Base URL of the external service.
- **Identity Type:** Defines who is making the callout:
  - `Named Principal` – All users share the same authentication.
  - `Per User` – Each Salesforce user uses their own credentials.
- **Authentication Protocol:** Type of authentication used for the API:
  - `Password Authentication`
  - `OAuth 2.0`
  - `AWS Signature`
  - `JWT Bearer Token`
- **External Credential (Optional):** Can link to external credentials stored securely.
- **Callout Enabled:** Must be checked to allow callouts using this credential.
- **Usage in Apex**: HttpRequest req = new HttpRequest(); req.setEndpoint('callout:WeatherAPI_NC/v1/current.json?q=London'); req.setMethod('GET'); HttpResponse res = new Http().send(req); System.debug(res.getBody());



2. External Services

Purpose

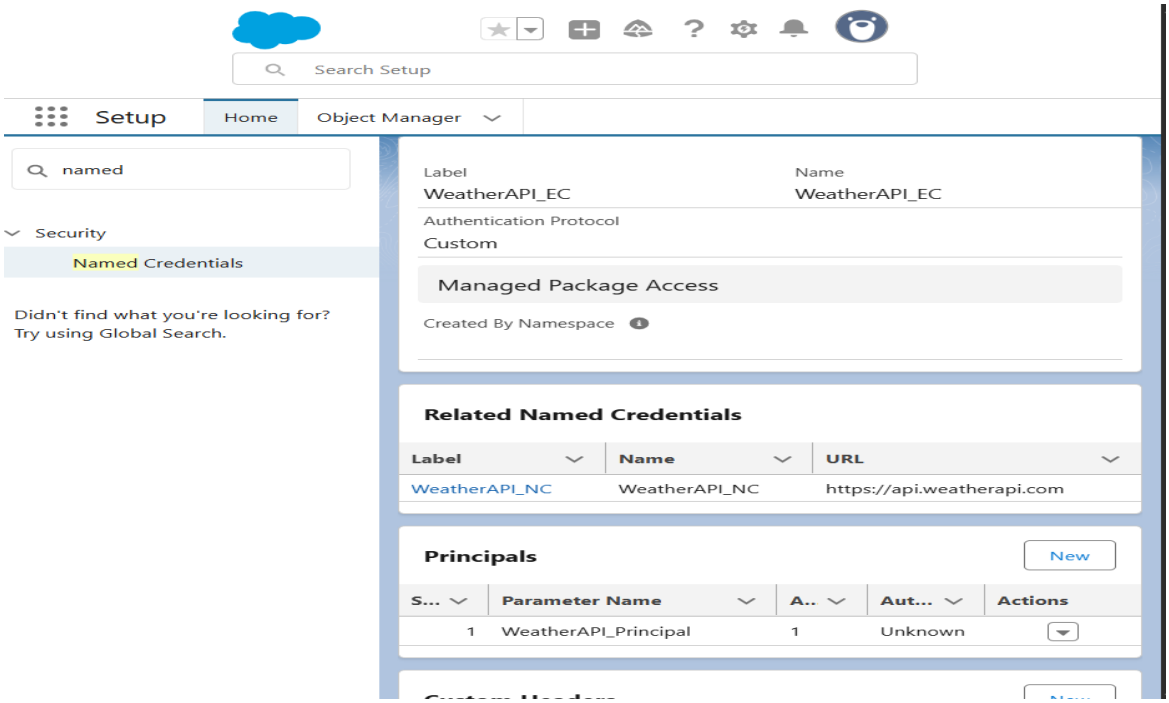
- Connect Salesforce to external APIs **declaratively**.
- Automatically generate **actions for Flows**.
- Simplifies integration for admins and non-developers.
- Supports both **REST and SOAP APIs**.

Key Features

- Declarative API integration with no Apex required.
- Uses **Swagger/OpenAPI schema** to define API structure.
- Actions are **automatically created** in Salesforce.
- Works with **Named Credentials** for secure authentication.
- Can be used in **Screen Flows, Scheduled Flows, and Record-Triggered Flows**.
- Supports reusability and governance of API calls.

Steps:

- Setup → External Services → New External Service
- Provide Name, Description, Service Schema (Swagger/OpenAPI URL)
- Select Named Credential
- Salesforce auto-generates actions usable in Flow



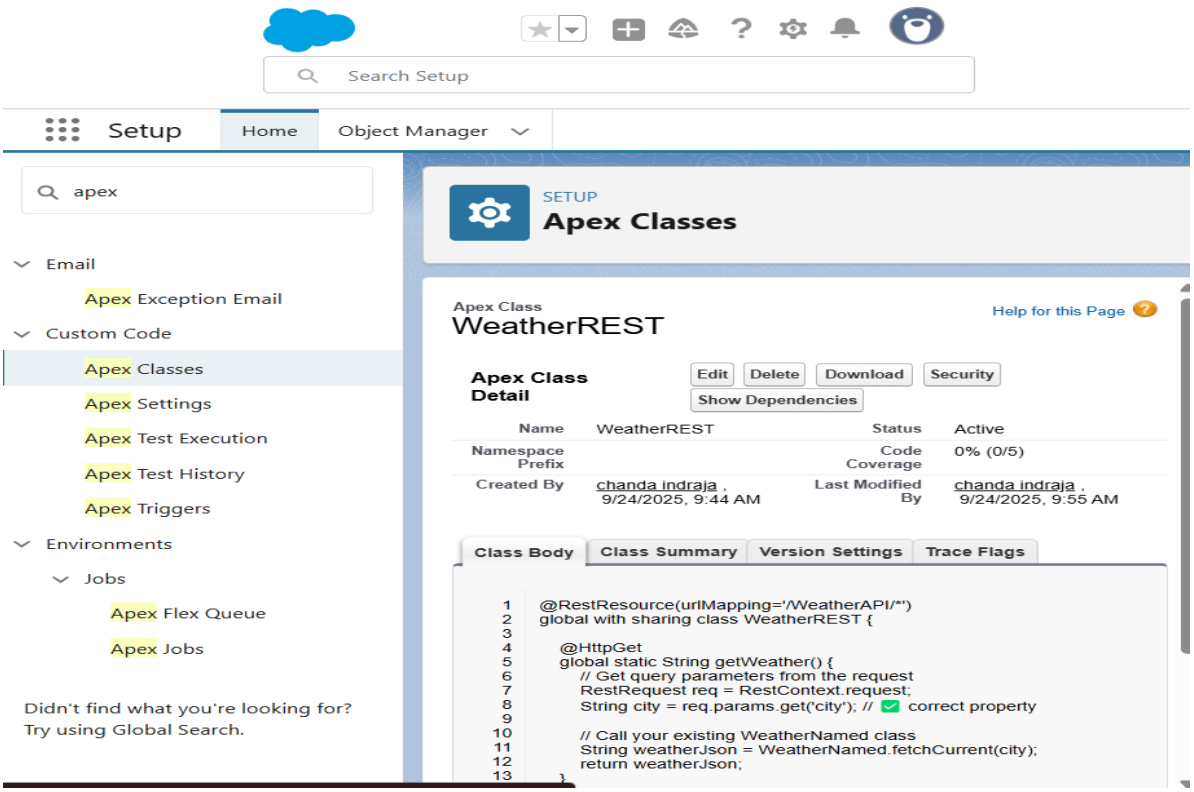
3. Web Services (REST/SOAP)

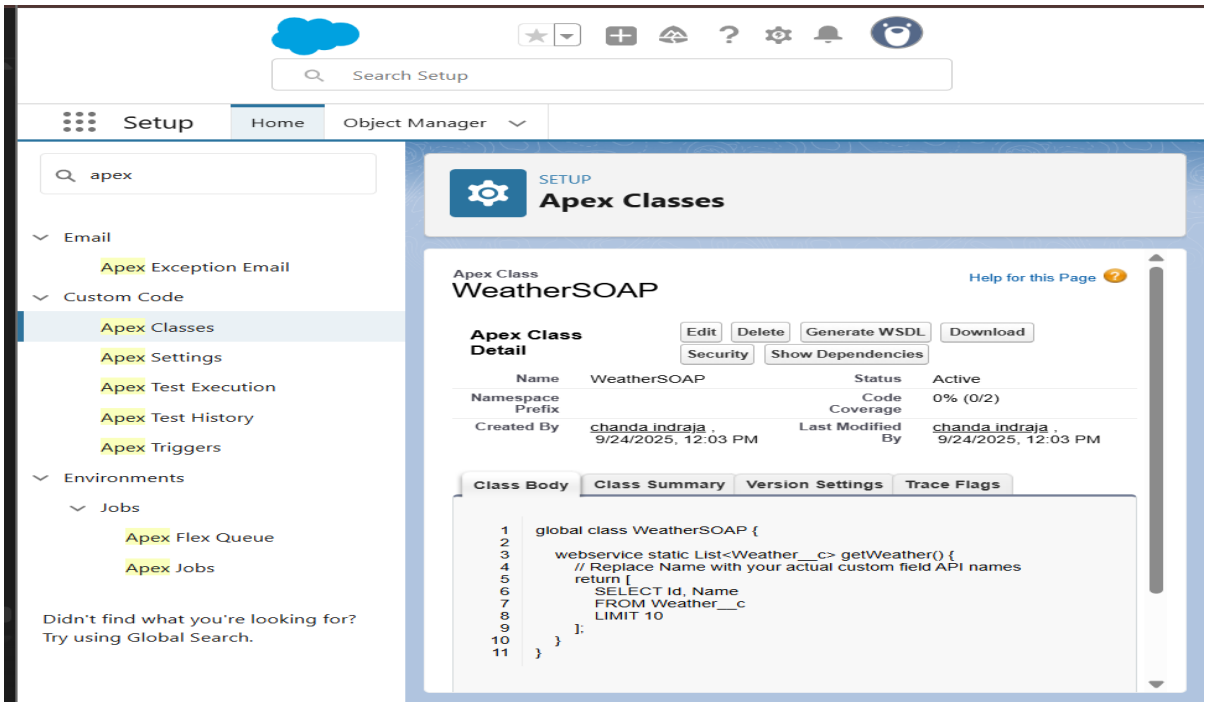
Purpose

- Expose Salesforce data to external systems.
- Consume external web services within Salesforce.
- Enable integration with external applications using standard protocols.
- Supports both **REST (lightweight)** and **SOAP (structured)** protocols.

Key Features

- **REST Services**
  - Lightweight, stateless, uses HTTP methods (GET, POST, PUT, DELETE).
  - Supports JSON and XML formats.
  - Easier to use for modern web and mobile apps.
- **SOAP Services**
  - Standardized protocol using XML.
  - Strongly typed operations, supports WSDL.
  - Preferred for enterprise-level integrations.





4. Callouts

Purpose

- Enable Salesforce to **communicate with external APIs** or web services.
- Retrieve or send data from/to external systems.
- Used in integrations, third-party services, or external data synchronization.

Key Features

- Supports **HTTP methods**: GET, POST, PUT, DELETE.
- Can use **Named Credentials** for secure authentication.
- Can also use **Remote Site Settings** for older setups.
- Supports **synchronous** (immediate response) and **asynchronous** (future methods, batch Apex) callouts.
- Can handle **REST and SOAP APIs**.
- Use **Named Credentials** instead of Remote Site Settings for security.
- Always handle exceptions with `try-catch`.

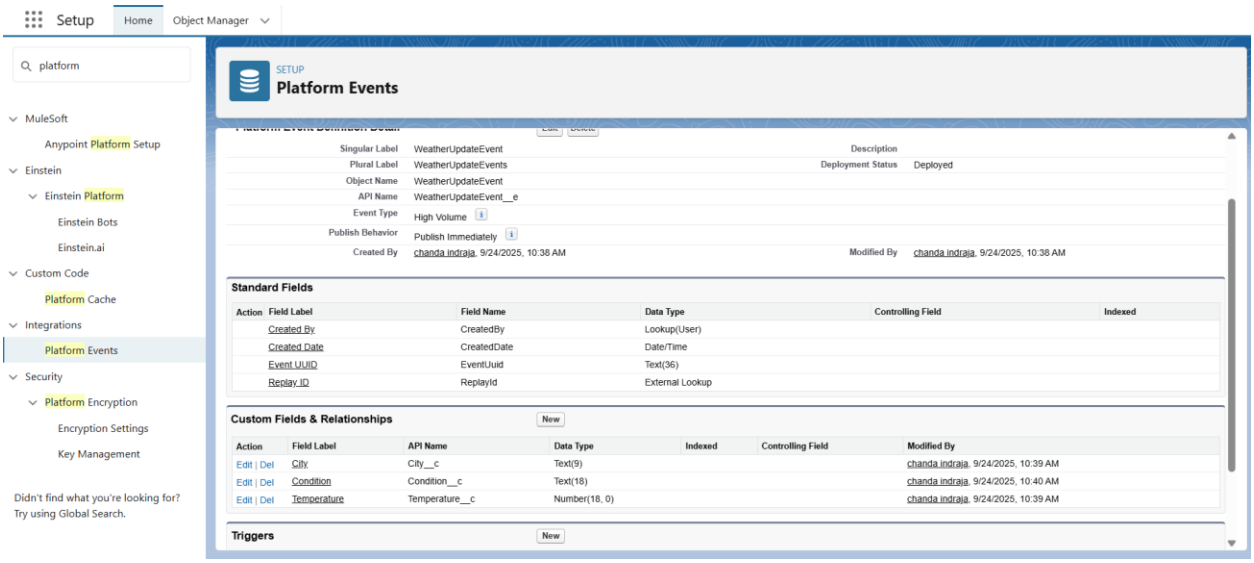
5. Platform Events in Salesforce

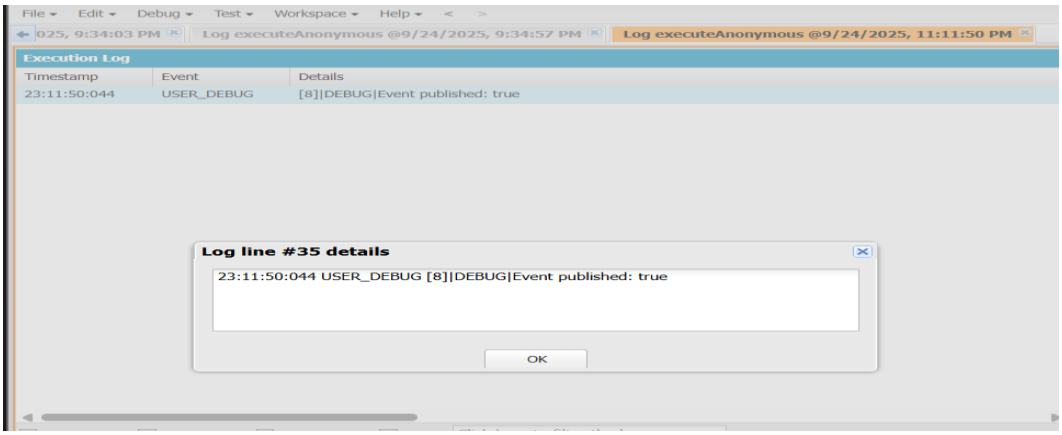
Purpose

- Enable **event-driven architecture** inside Salesforce.
- Send and receive custom notifications between Salesforce processes or external systems.
- Useful for real-time updates and decoupled integrations.

Key Features

- **Custom Event Objects** with fields like any other object.
- Events are **published** and **subscribed** asynchronously.
- Supports **Publish-Subscribe model**:
    - **Publisher**: Creates and publishes events.
    - **Subscriber**: Triggers, Flows, or external systems that react to events.
- Can be used in **Apex, Flows, and External Systems (Streaming API)**.
- Durable events: Can retain events for up to **24 hours**.





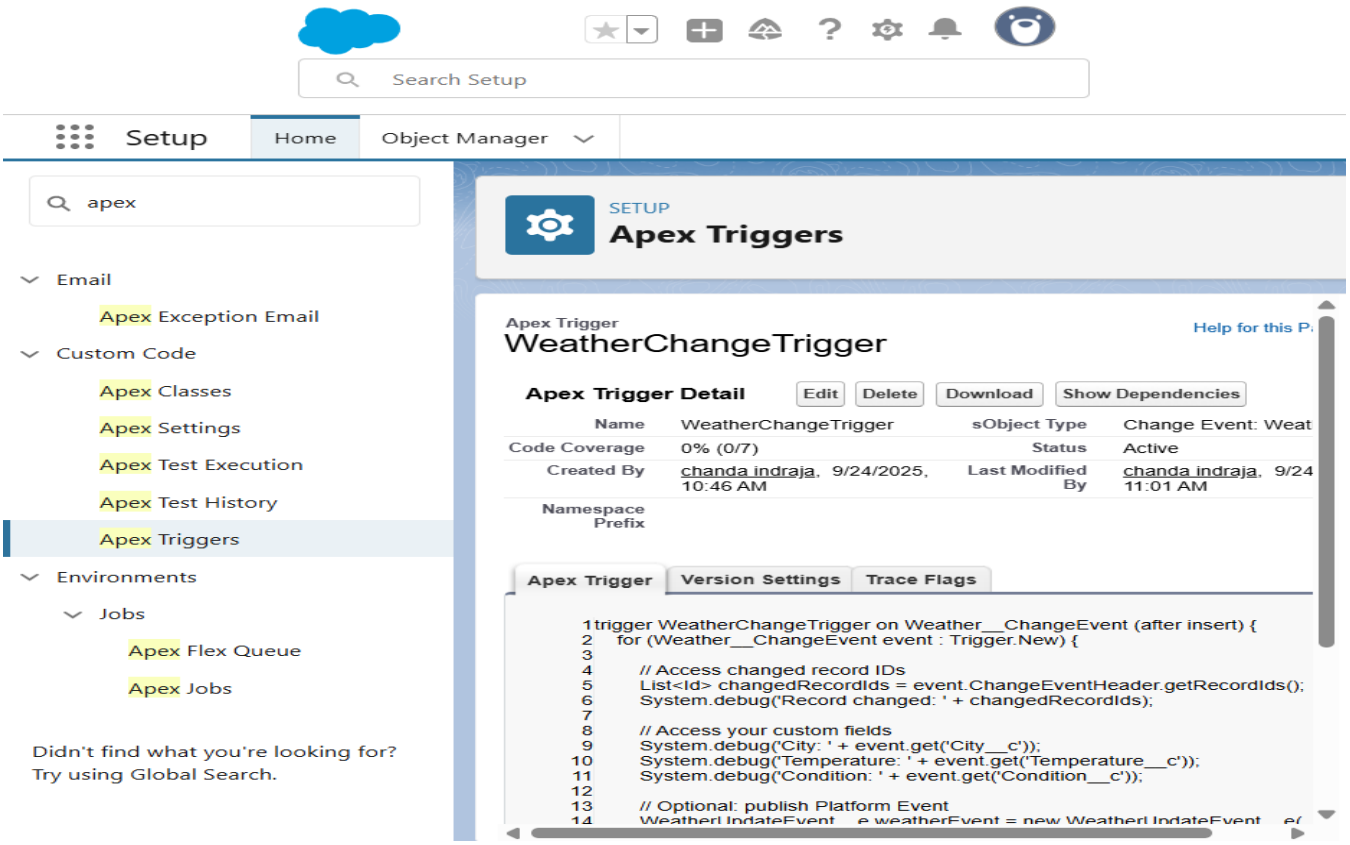
6. Change Data Capture (CDC)

Purpose

- Track **create, update, delete, and undelete events** on Salesforce records in real time.
- Enable **asynchronous integration** with internal or external systems.
- Helps in **synchronizing Salesforce data** with other systems.
- Works with **any standard or custom object**, including `Weather__c`.

Key Features

- Real-time tracking of **record changes**.
- Automatically generates **Change Event objects** for tracked objects.
- Supports **Apex Triggers, Flows, and External Subscribers**.
- Captures **all field changes** automatically.
- Can be integrated with **Streaming API** or external systems.
- Works with **standard objects** and **custom objects**.



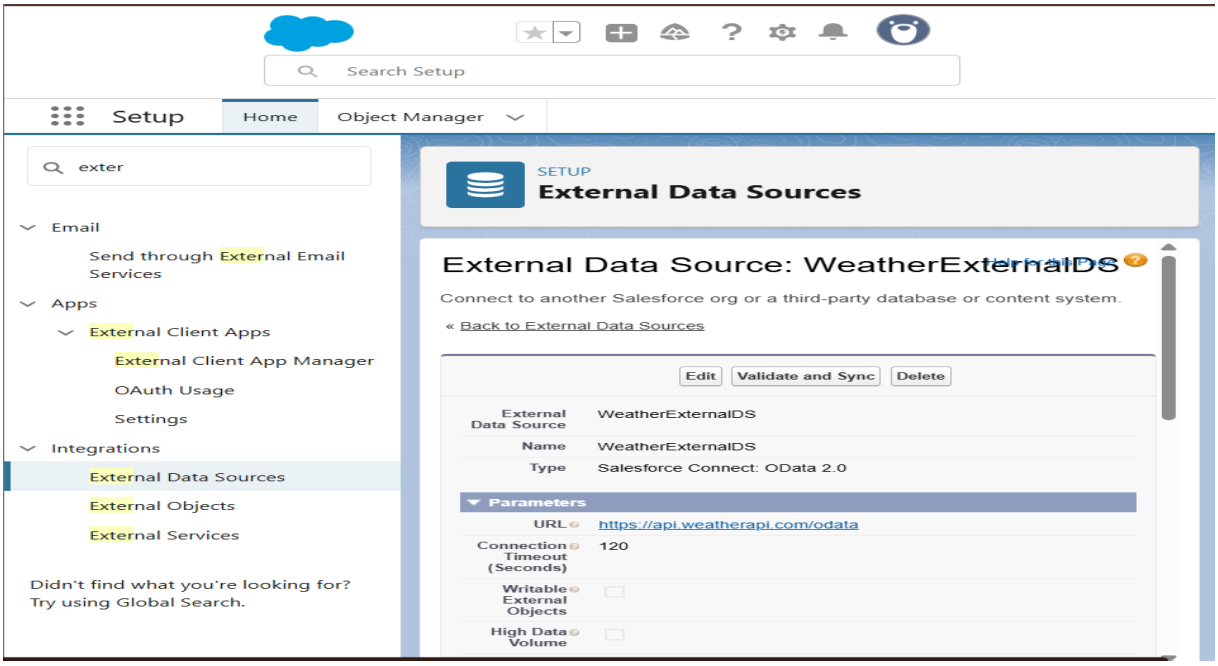
7. Salesforce Connect

Purpose

- Integrate external data sources directly into Salesforce without storing the data in Salesforce.
- Enables **real-time access** to external objects via **OData, REST, or custom adapters**.
- Ideal for **large datasets** or external systems where replication is not practical.

Key Features

- Supports **OData 2.0 and 4.0** protocols.
- Creates **External Objects** in Salesforce representing external tables.
- Provides **read, create, update, and delete** access depending on external system capabilities.
- Works with **Lightning, Visualforce, and Apex**.
- Can combine **external objects with standard Salesforce objects** using **relationship**



8. API Limits

Purpose

- Salesforce enforces **limits on API calls** to protect system resources.
- Helps track and manage usage for integrations and automation.

Key Points

- **API Request Limits** vary by **edition, license type, and user**.

- Types of API calls:
    - REST API
    - SOAP API
    - Bulk API
    - Streaming API
    - Tooling API
- Monitor usage via: **Setup → System Overview → API Usage**, or **REST `Limits` resource**.
- **Exceeded limits** return `REQUEST_LIMIT_EXCEEDED` errors.

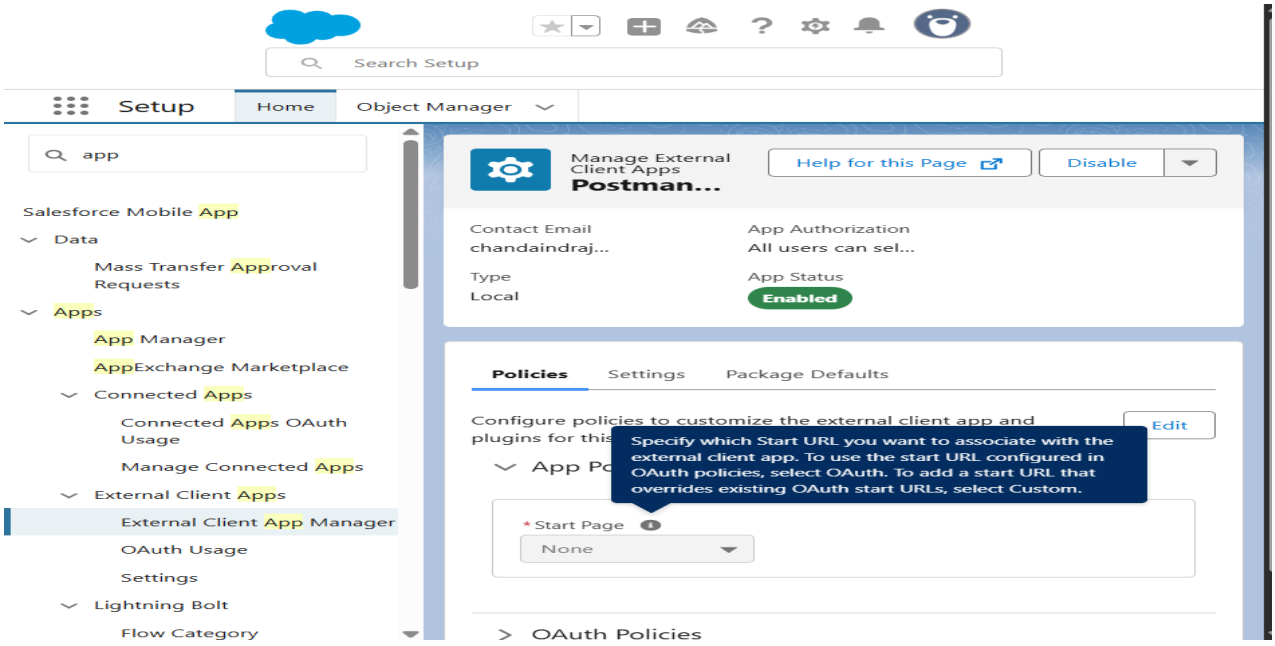
9. OAuth & Authentication

Purpose

- Secure integration with Salesforce or external systems.
- Provides **token-based authentication** without storing passwords.
- Supports **single sign-on (SSO) and API access**.

Key Features

- Supports **OAuth 2.0 flows**:
    - Authorization Code
    - Username-Password
    - JWT Bearer
    - Refresh Token
- Provides **access token** for API calls.
- Tokens can expire; refresh tokens allow re-authentication without user intervention**.**



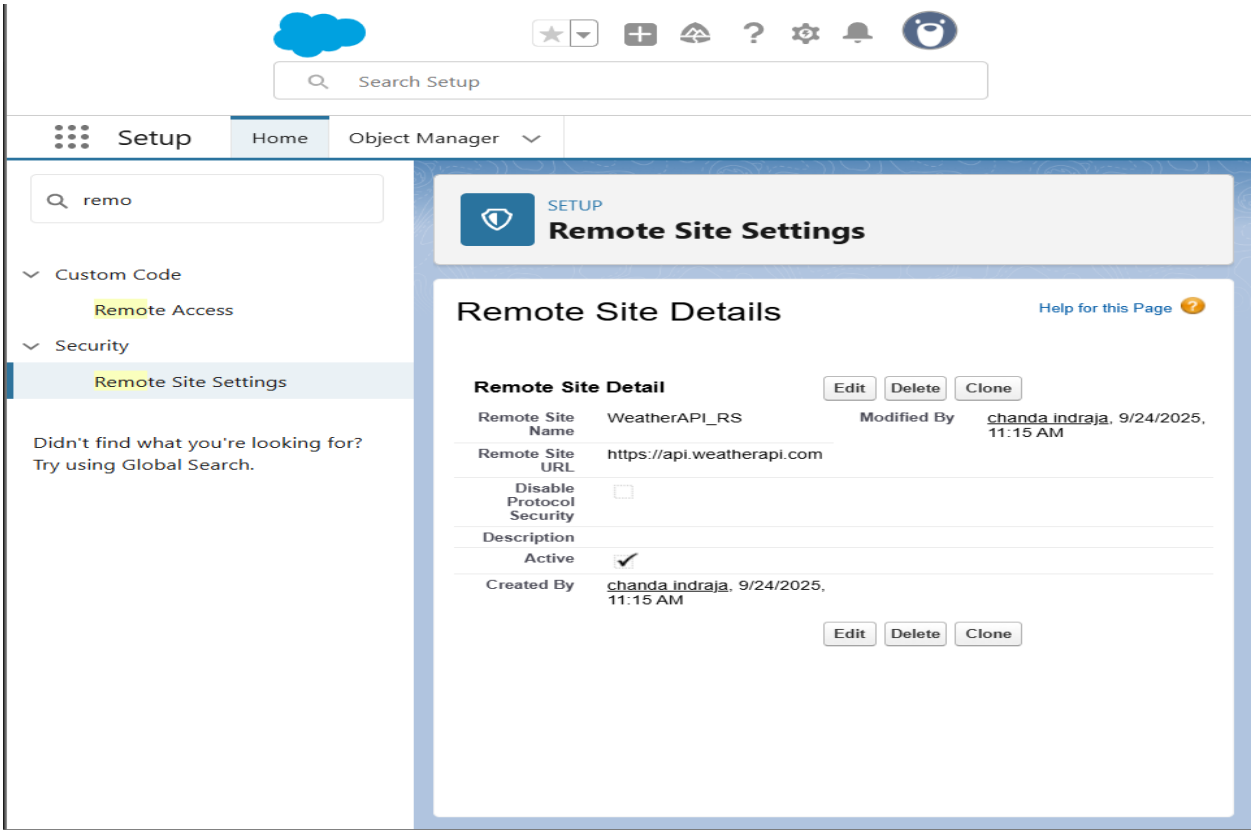
10.Remote Site Settings

Purpose

- Allows Salesforce to **perform HTTP or HTTPS callouts** to external systems.
- Required when using **Apex HTTP callouts** to URLs that are **not secured via Named Credentials**.
- Ensures **security by allowing only trusted URLs** to be accessed from Salesforce.

Key Features

- Lets Salesforce **call external web services or APIs**.
- Provides a **centralized place** to manage external URLs.
- Required for **REST/SOAP API integration**, webhooks, or external system data fetch.
- Works with **all Apex HTTP callouts**, including GET, POST, PUT, DELETE.
- Supports **HTTPS** endpoints for secure communication.
- Can be used in combination with **Named Credentials** for advanced authentication.

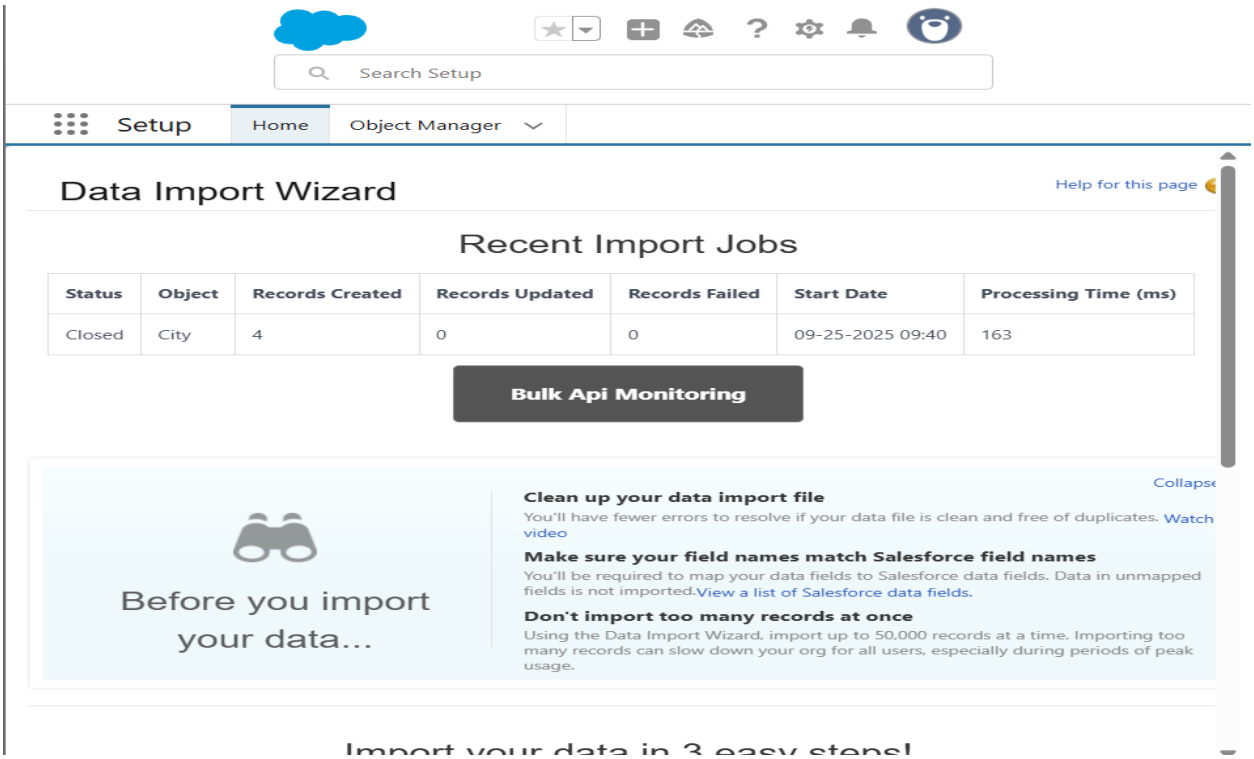


Phase 8: Data Management & Deployment

Effective data management and deployment strategies ensure that Salesforce data is clean, secure, and easy to move between environments (like sandbox and production).

1. Data Import Wizard

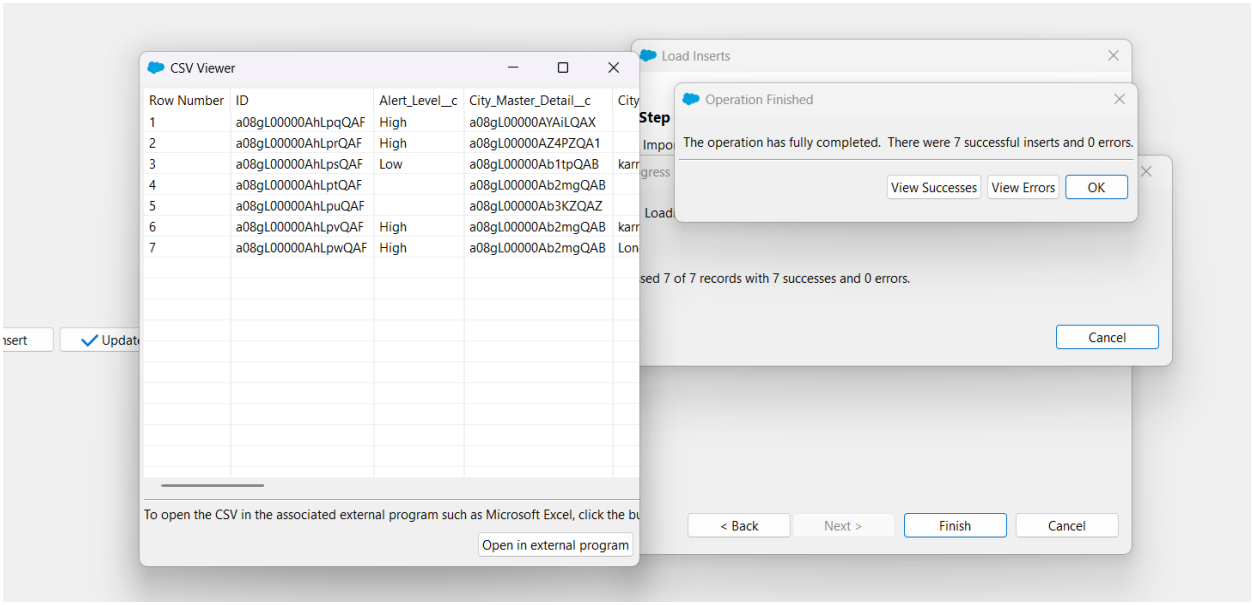
- **Purpose**: Easy-to-use tool for importing data (CSV files) directly into Salesforce.
- **Usage**: Available inside Salesforce Setup → Data Import Wizard.
- **Features**:
  - Import up to **50,000 records**.
  - Supports standard objects (Accounts, Contacts, Leads, Campaign Members) and custom objects.
  - Provides **field mapping** interface.
  - Can update existing records or insert new ones.
- **Best Use Case**: Small data loads by admins (no external tool needed).

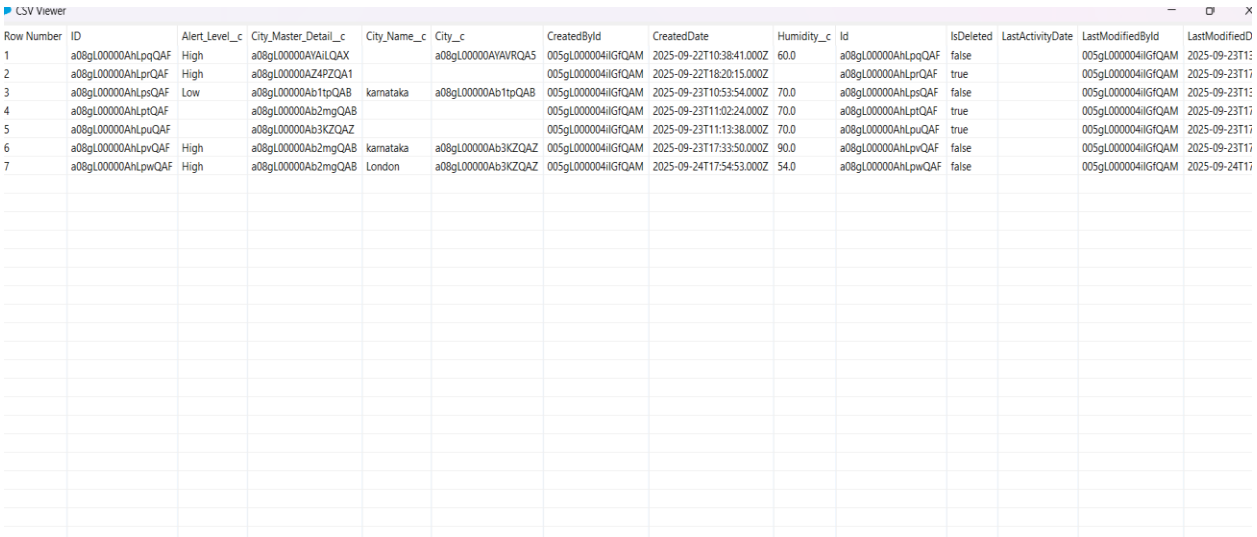


2. Data Loader

- **Purpose**: Advanced client application for **bulk import/export** of data.
- **Features**:
  - Handles **millions of records**.
  - Supports Insert, Update, Upsert, Delete, Hard Delete, Export.

    - Requires CSV files.
    - Available as **UI** and **Command-line**.
- **Best Use Case**: Large data migrations, scheduled automated data jobs.

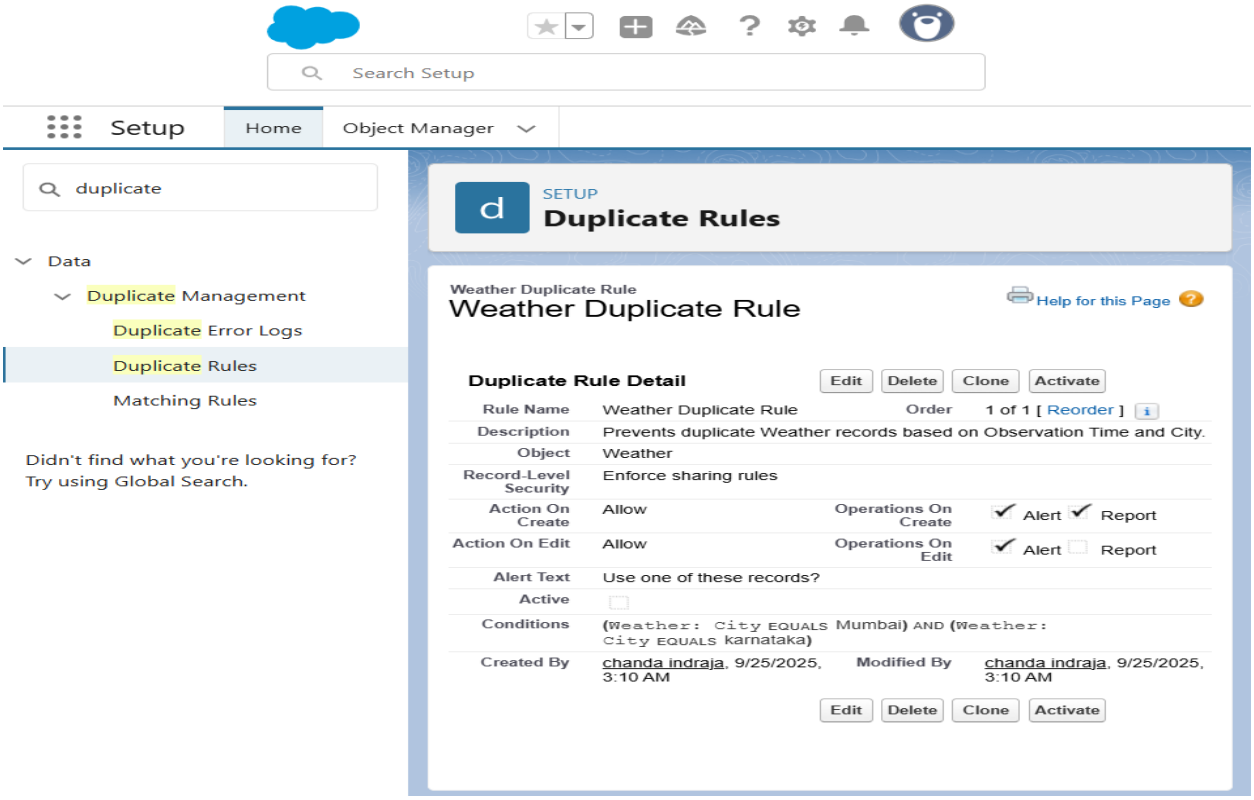


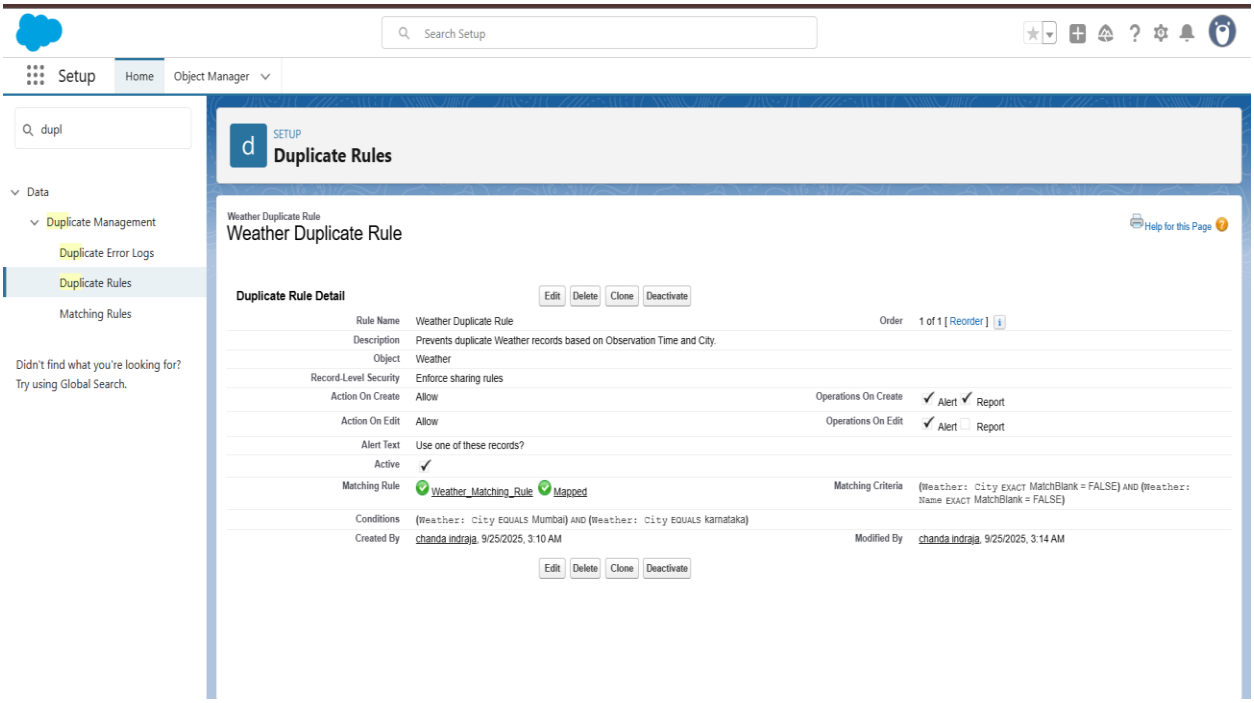


3. Duplicate Rules

- **Purpose**: Prevent or manage duplicate records in Salesforce.
- **Setup**: Setup → Duplicate Rules.
- **Features**:
    - Compare new/existing records to find duplicates.
    - Supports **blocking** or **allowing but alerting** duplicates.

    - Works with **Matching Rules** (e.g., match by Email, Phone, Name).
- **Best Use Case**: Keep clean data in Leads, Accounts, Contacts.





4. Data Export & Backup

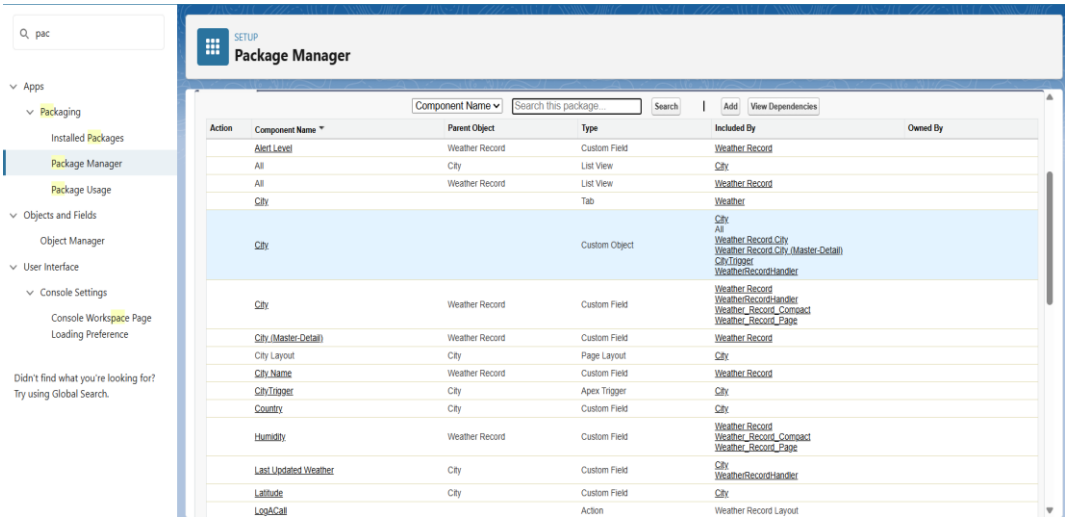
- **Purpose**: Protect against data loss by exporting Salesforce data.
- **Methods**:
  - **Data Export Wizard** → Manual or Scheduled CSV export of all objects.
  - **Data Loader Export** → Export selected objects or queries.
- **Features**:
  - Export can include **attachments, documents, images**.
  - Can schedule weekly or monthly backups.
- **Best Use Case**: Regular data backup for compliance & recovery.

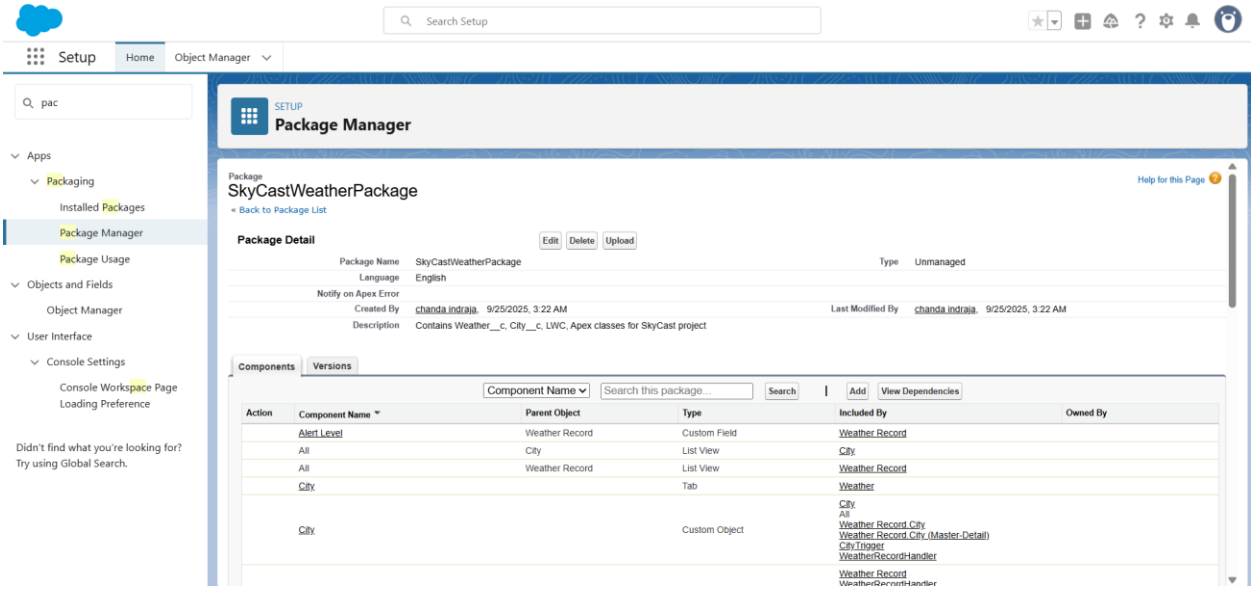
5. Change Sets

- **Purpose**: Deploy metadata (configuration/customizations) between Salesforce orgs (Sandbox → Production).
- **Features**:
  - Supports objects, fields, workflows, flows, validation rules, etc.
  - Point-and-click interface (no coding).
  - Requires **connected sandbox/production orgs**.
- **Limitations**: Cannot deploy some metadata types (e.g., standard picklist values).
- **Best Use Case**: Admin-friendly deployments.

6. Unmanaged vs Managed Packages

- **Unmanaged Package**:
  - Source code is visible.
  - Used for sharing open-source or sample apps.
  - No upgrade support.
- **Managed Package**:
  - Code is hidden (IP protected).
  - Supports **version upgrades**.
  - Used by ISVs (Independent Software Vendors) for AppExchange apps.
- **Best Use Case**:
  - *Unmanaged*: Internal projects & learning.
  - *Managed*: Commercial apps on AppExchange.





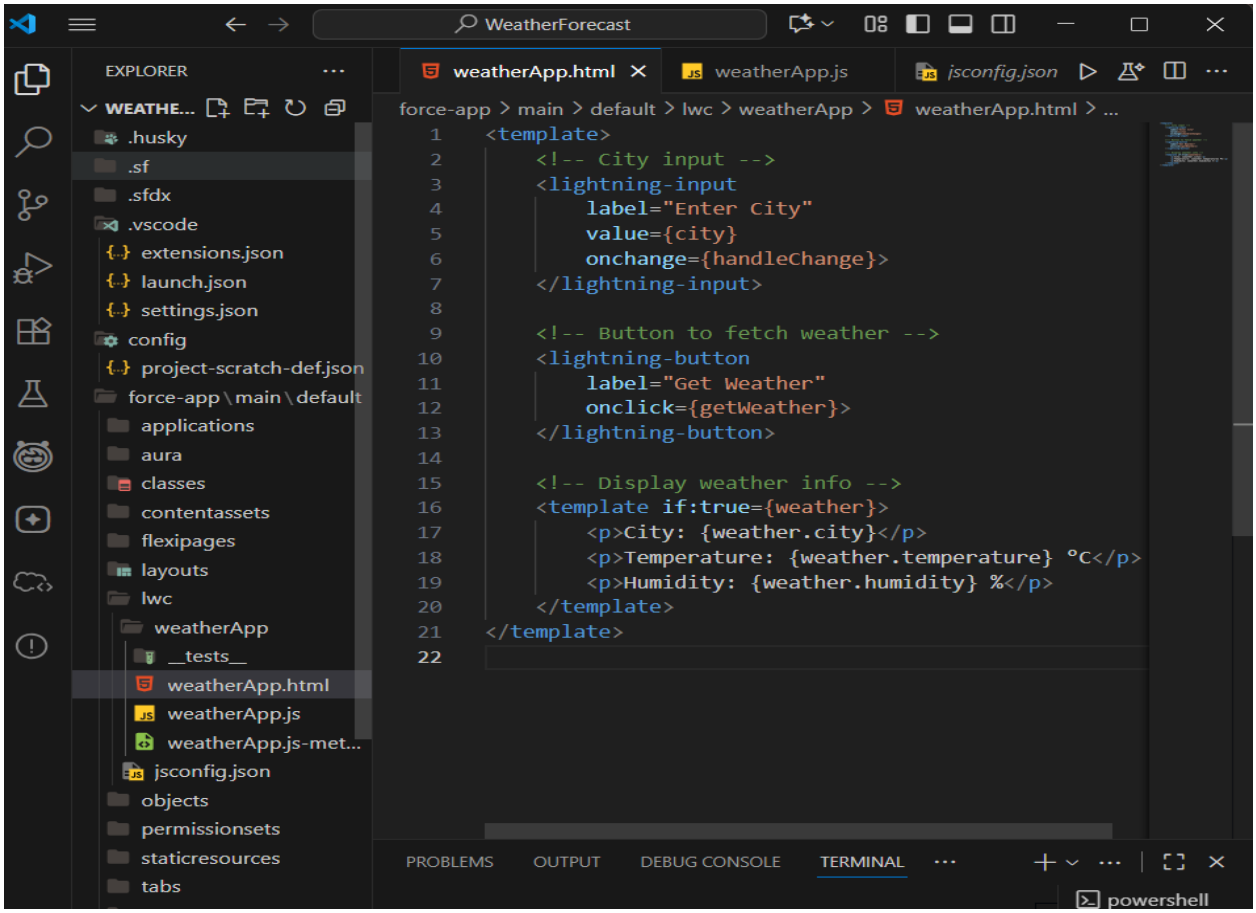
7. ANT Migration Tool

- **Purpose**: Command-line utility for automating metadata deployment.
- **Features**:
  - Based on **Apache ANT + ant-salesforce.jar**.
  - Supports scripted **retrieve & deploy**.
  - Uses `build.xml` + `package.xml` to define deployment.
- **Best Use Case**: Automated deployments in CI/CD pipelines.
- **Requirement**: Java & ANT installed locally.

8. VS Code & SFDX (Salesforce DX)

- **Purpose**: Modern development & deployment tool for Salesforce projects.
- **Features**:
  - Uses **Salesforce CLI (SFDX)** for source-driven development.
  - Supports Git-based version control.
  - Create & manage **scratch orgs** for testing.
  - Retrieve, deploy, and test metadata.
- **Best Use Case**: Developer-focused projects, CI/CD automation.
- **Setup**:

1. Install **VS Code**.
2. Install Salesforce **CLI (SFDX)**.
3. Install Salesforce **Extension Pack** in VS Code.
4. Authenticate org (`sfdx force:auth:web:login`).
5. Retrieve/deploy metadata via commands.



Phase 9: Reporting, Dashboards & Security Review

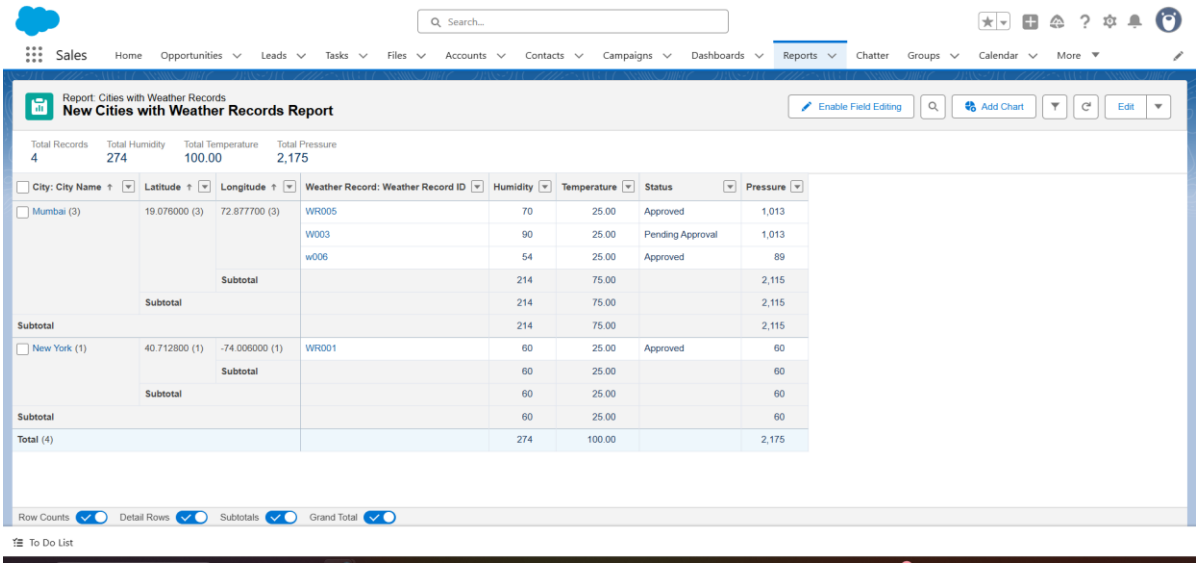
This phase focuses on analyzing business data through **Reports & Dashboards**, and ensuring **data security and compliance** using Salesforce's built-in security features.

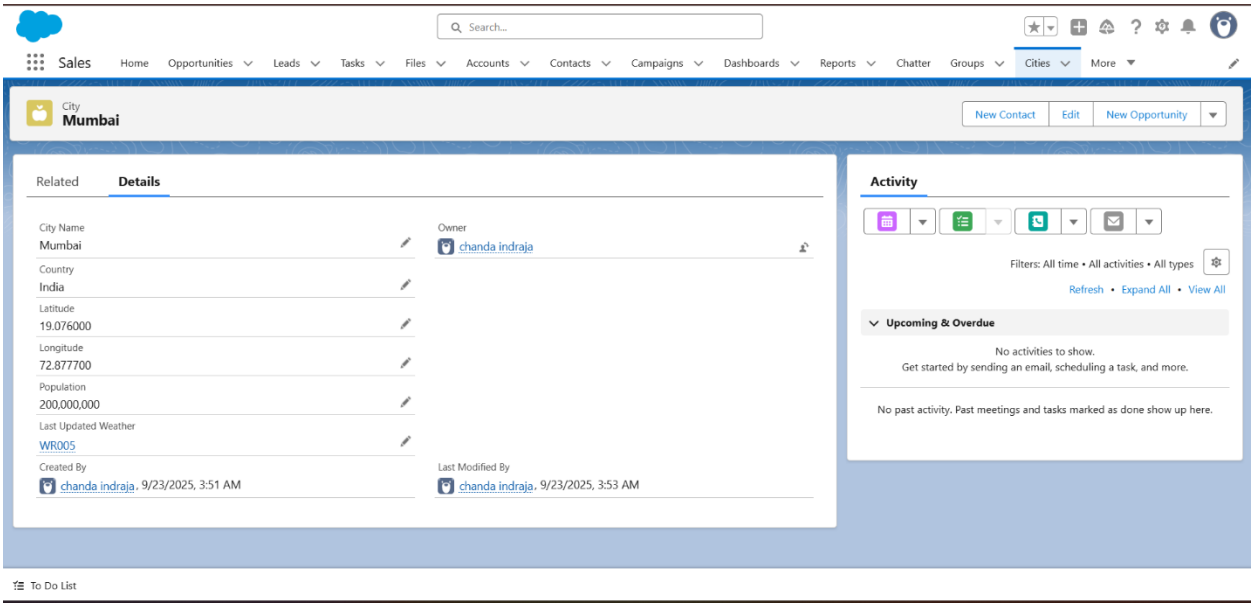
1. Reports(Tabular, Summary, Matrix, Joined)

Reports help users analyze Salesforce data in different formats.

Types of Reports:

- **Tabular Reports**
  - Simple, spreadsheet-like format.
  - Best for **lists** (e.g., "All Accounts in New York").
  - Cannot create groupings or dashboards.
- **Summary Reports**
  - Similar to Tabular, but supports **grouping rows**.
  - Allows subtotals and charts.
  - Example: "Opportunities grouped by Stage."
- **Matrix Reports**
  - Group records by both **rows and columns**.
  - Best for comparing related totals.
  - Example: "Revenue by Sales Rep and by Quarter."
- **Joined Reports**
  - Combine multiple report blocks (each with different report type).
  - Best for complex reporting needs.
  - Example: "Cases with related Opportunities and Activities."





2. Report Types

Standard Report Types

- Auto-generated by Salesforce when you create **standard or custom objects**.
- Provide access to most fields and relationships of the object.
- Examples:
    - Accounts with Contacts
    - Opportunities with Products
- **Limitations**:
    - Cannot always include **custom relationships**.
    - Fields may be restricted compared to Custom Report Types.

Custom Report Types (CRTs)

- Created manually by Admin when **standard report types don't meet requirements**.
- Benefits:
    - Control which objects and fields are available in the report.
    - Define **object relationships** (e.g., "With" or "Without" records).
    - Add custom labels, descriptions, and categories for easier use.
- Example Use Case:
    - "Weather Records with related Cities and without Forecasts."

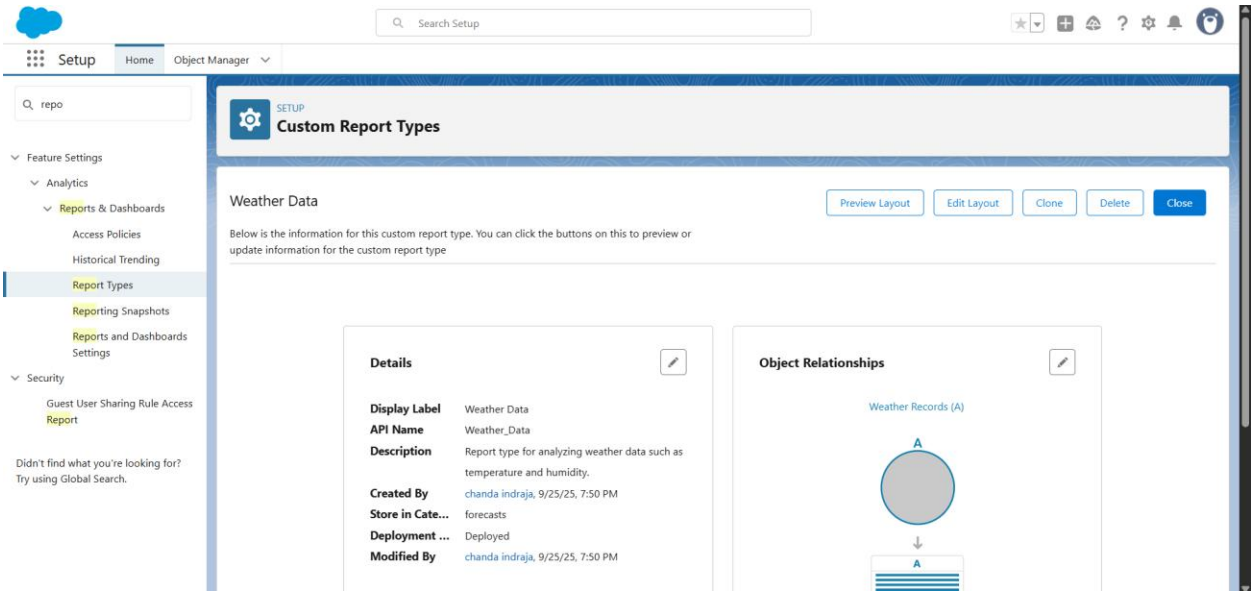
Primary Object

- The **main object** that the report focuses on.
- All records displayed will start from this object.
- Example:

- If **Weather__c** is the primary object → the report shows Weather data.

Related Objects

- Objects that are connected through **lookup or master-detail relationships**.
- Relationship types in CRTs:
  - **With** → Report includes related records.
  - **With or Without** → Includes both matching and non-matching records.
- Example:
  - Weather__c (Primary) → City__c (Related).

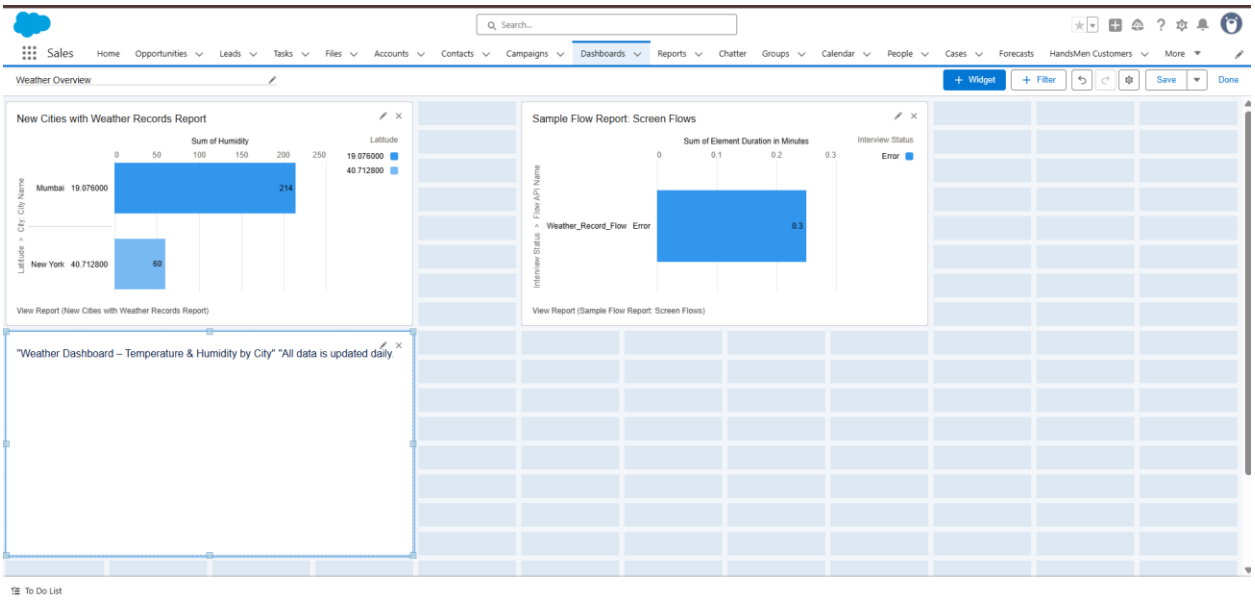


3. Dashboards

- **Purpose**: Visual representation of report data.
- **Components**: Charts, tables, metrics, gauges.
- **Source**: Each component is based on one underlying report.
- **Use Case**: "Executive Dashboard showing Sales Revenue, Pipeline, and Closed Deals."

Features:

- Interactive charts for quick insights.
- Supports multiple components on a single page.
- Can schedule dashboards to refresh automatically.

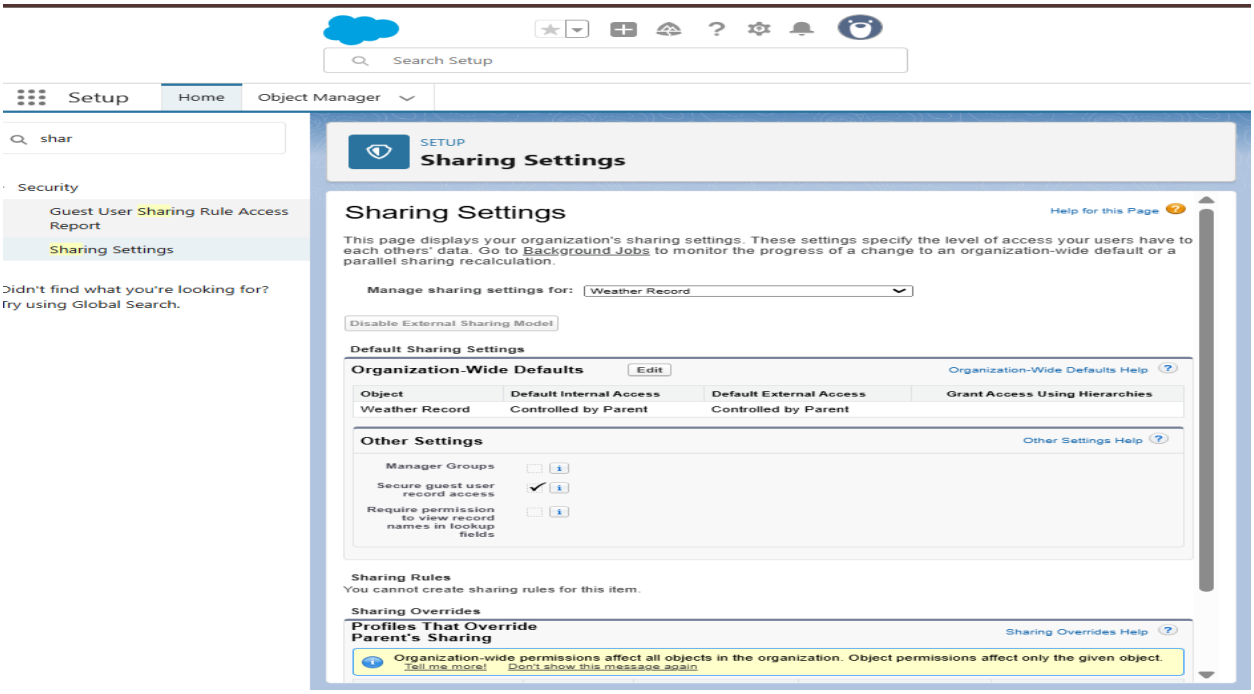


4. Dynamic Dashboards

- **Purpose**: Show data according to the **logged-in user's security and permissions**.
- **Benefits**:
  - No need to create multiple dashboards for different users.
  - Ensures users only see data they are authorized to view.
- **Best Use Case**: Sales Managers vs. Sales Reps viewing the same dashboard but with different data visibility.

5. Sharing Settings

- **Purpose**: Define how records are shared between users.
- **Options**:
  - **Private** → Only record owner & admins can access.
  - **Public Read Only** → Everyone can see but not edit.
  - **Public Read/Write** → Everyone can see & edit.
- **Tools**:
  - **Organization-Wide Defaults (OWD)** – baseline access.
  - **Role Hierarchy** – managers can access subordinates' data.
  - **Sharing Rules** – exceptions to open access.
  - **Manual Sharing** – record owner shares specific records.



6. Field-Level Security (FLS)

Purpose

- Control **who can see or edit** individual fields in Salesforce objects.
- Ensures **sensitive information** is protected while giving appropriate access to users.
- Works at the **Profile or Permission Set level**.

Options / Settings

- **Visible / Hidden**
  - Determines if a field is **shown or completely hidden** from users.
  - Hidden fields do not appear in **Page Layouts, Reports, or List Views**.
- **Read-Only**
  - Allows users to **view** the field but **cannot edit** it.
  - Useful for fields that should be **visible for reference** but protected from changes.
- **Editable**
  - Users can **view and update** the field.
  - Usually granted to trusted roles/profiles only.

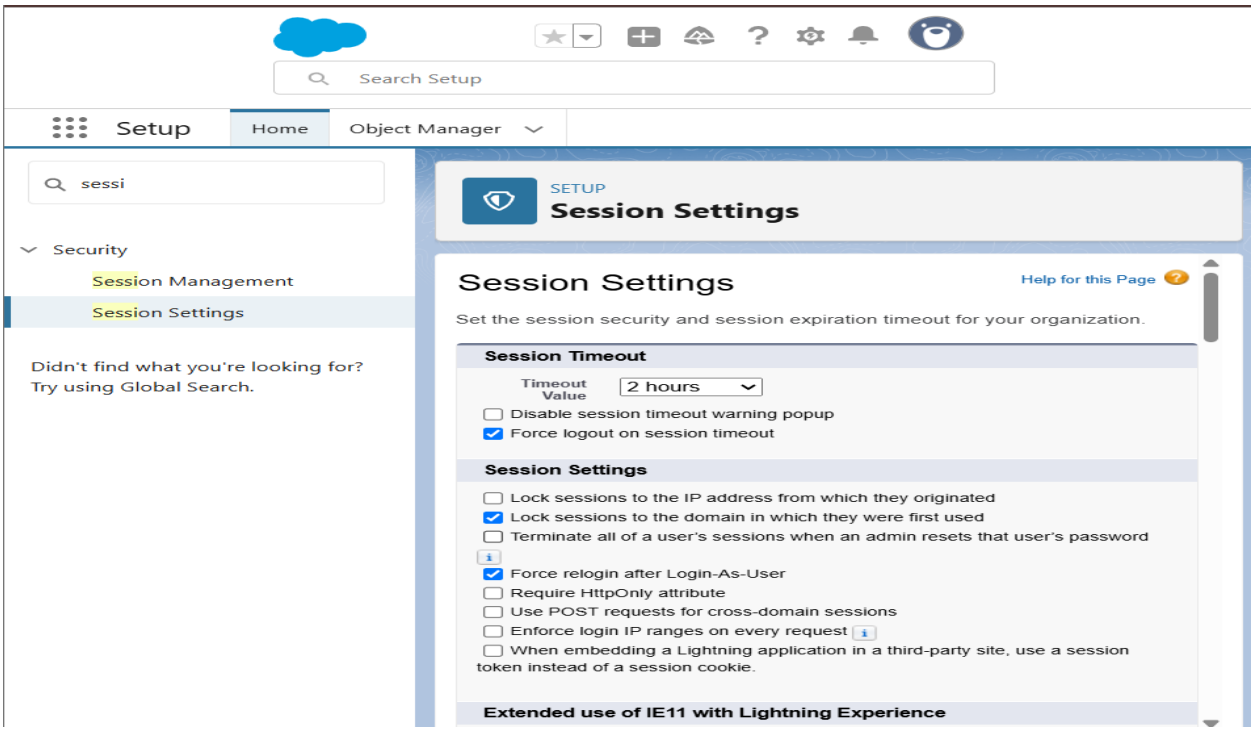
7. Session Settings

Purpose

- Define **security policies for user sessions** in Salesforce.
- Protect sensitive data and prevent unauthorized access.

Key Features / Options

- **Session Timeout**
  - Auto-logout users after a period of inactivity.
  - Example: 15, 30, 60 minutes depending on org security requirements.
- **Restrict Concurrent Logins**
  - Prevent multiple logins using the same user credentials simultaneously.
- **IP Restrictions on Sessions**
  - Limit login or session access to trusted IP addresses.
- **Enable Multi-Factor Authentication (MFA)**
  - Adds an extra security layer using a verification code or app.



8. Login IP Ranges

Purpose

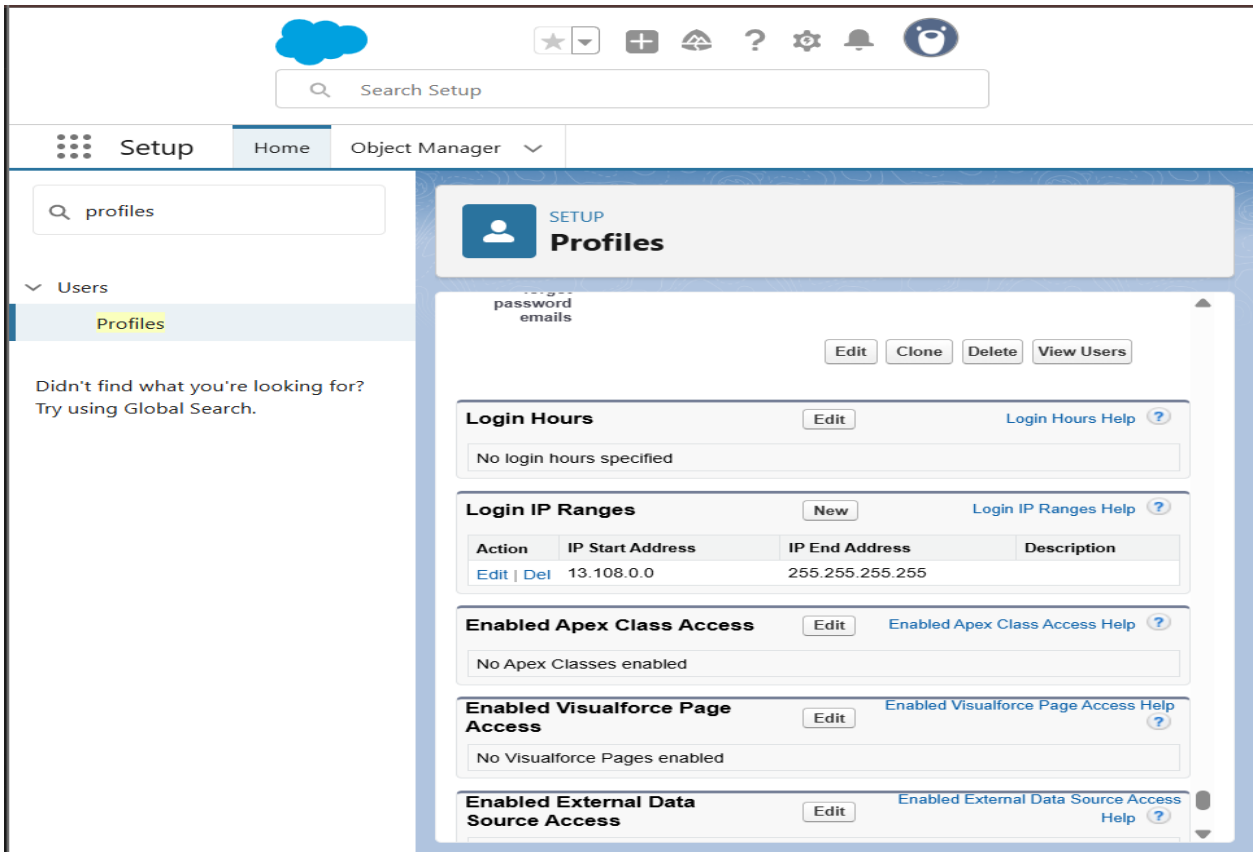
- Restrict access to Salesforce **based on trusted IP addresses**.
- Ensures that users can only log in from **approved networks**.

Setup

- Defined at the **Profile level** for each user type.
- Users outside the allowed IP range **cannot log in**.
- IP ranges can be set per profile, e.g., Sales team vs. Support team.

Best Use Cases

- Prevent unauthorized logins from **public networks**.
- Enforce **office network or VPN-only access**.
- Protect sensitive org data in multi-location environments.



9. Audit Trail

Purpose

- Track all **configuration changes** in Salesforce setup.
- Useful for **troubleshooting, compliance, and security audits**.

Key Features

- **Tracks Setup Changes**
  - Examples: fields, objects, workflows, validation rules, permissions.
- **History Availability**
  - Last 20 entries visible directly in Setup Audit Trail.
  - Full 180-day history downloadable as CSV for long-term auditing.
- **User Information Tracked**
  - Who made the change, what was changed, and when it occurred.

Best Practices

- Regularly **download audit trail data** for compliance reviews.
- Use in conjunction with **Field-Level Security and Sharing Rules** for complete visibility.
- Helps in **troubleshooting configuration issues** when errors occur.

Conclusion

The **SkyCast Weather Information System** provides a robust and scalable solution for managing, analyzing, and reporting weather-related data. By leveraging Salesforce features such as **custom objects, data import tools, reporting, dashboards, and security controls**, the system ensures accurate data capture, efficient processing, and secure access for authorized users. With integrated reporting and dashboards, stakeholders can monitor weather patterns in real-time, make informed decisions, and maintain compliance with organizational security standards. Overall, SkyCast delivers a comprehensive platform to transform raw weather data into actionable insights, enhancing operational efficiency and decision-making capabilities.