✅ Phase 8: Data Management & Deployment (Detailed)

This phase focuses on **managing Salesforce data efficiently, maintaining data quality, and deploying changes safely** across environments.

☐Data Import Wizard (Simple Data Upload)

Purpose: Quickly import records into Salesforce without coding.

Where to Go:
Setup → Data Import Wizard

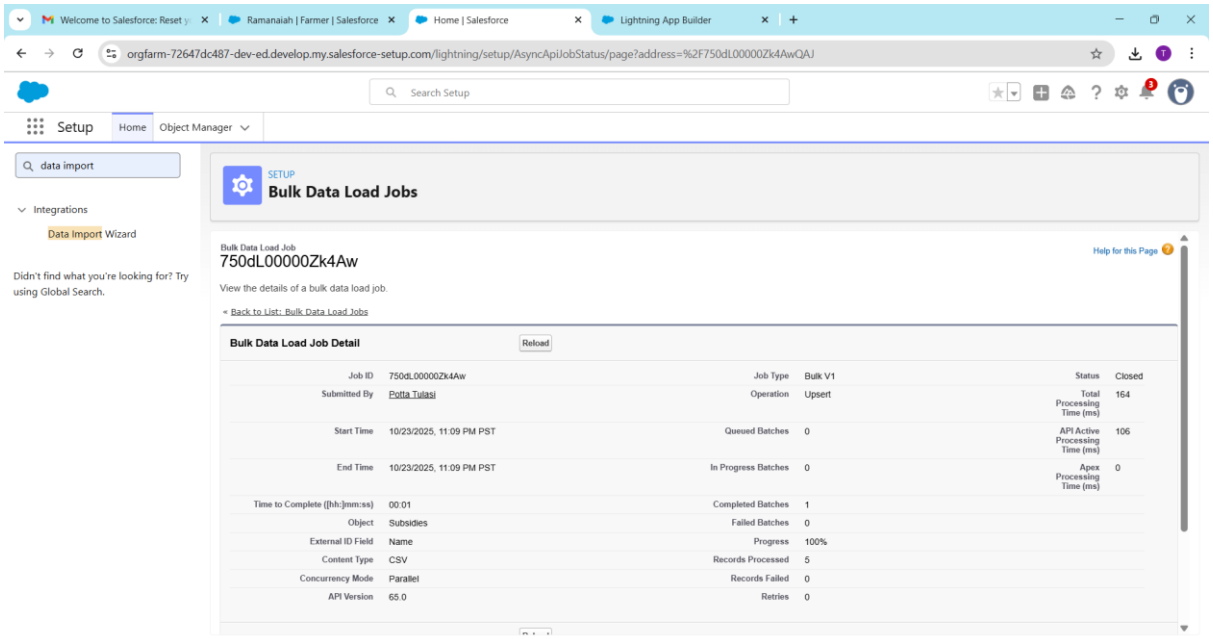
Step-by-Step Instructions:

1. Choose the **object** you want to import (e.g., Farmer__c, Buyer__c, Order__c)
2. Upload a **CSV file** with your data
3. **Map CSV columns to Salesforce fields** (e.g., Farmer Name → Name, Phone → Phone)
4. Review errors or duplicates flagged by Salesforce
5. Click **Start Import**

Best For: Small to medium data sets (up to 50,000 records).

Example in Rural Market Linkage:

- Import **new farmer registrations** collected from offline forms into the Farmer__c object.



2. Data Loader (Bulk Data Management)

Purpose: Bulk insert, update, delete, or export Salesforce records.

Where to Go:

- Download **Data Loader**: Salesforce Data Loader Download

Step-by-Step Instructions:

1. Open **Data Loader** → Login with your Salesforce credentials (Sandbox or Production)
2. Select operation:
    - **Insert:** Add new records
    - **Update:** Modify existing records
    - **Upsert:** Insert new records or update existing ones
    - **Delete:** Remove records
    - **Export:** Extract data from Salesforce
3. Upload your **CSV file** and **map fields**
4. Run the operation → check **success and error files**

Best For: Large datasets (50,000+ records), complex updates.

Example: Bulk update the **Crop__c prices** for multiple farmers after a new government subsidy adjustment.

3. Duplicate Rules (Prevent Duplicate Data)

Purpose: Maintain data integrity by **blocking duplicate records**.

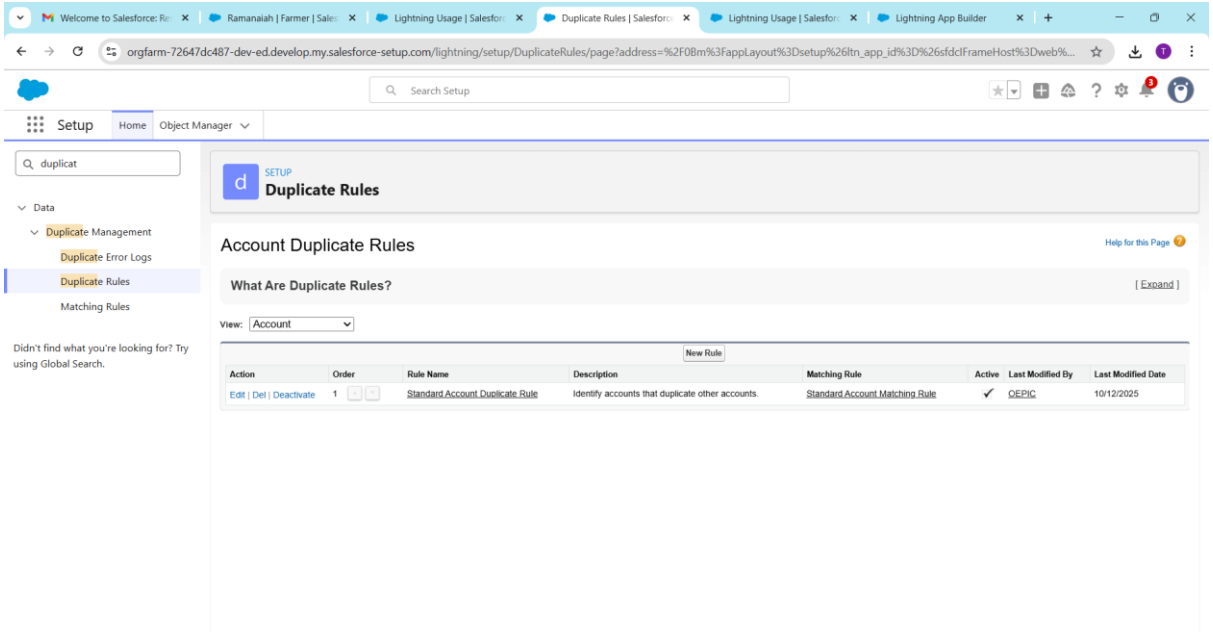
Where to Go:
Setup → Duplicate Management → Duplicate Rules

Step-by-Step Instructions:

1. Create **Matching Rules** (e.g., match farmers by Email or Phone)
2. Create **Duplicate Rule** for the object:
    - **Block:** Prevent duplicates
    - **Alert:** Notify users but allow save
3. Activate the rule

Example: Prevent multiple Farmer__c records with the same phone number.

Tip: Combine **Duplicate Rules + Matching Rules** for maximum accuracy.



4⃣ Data Export & Backup (Secure Your Data)

Purpose: Regular backups prevent data loss.

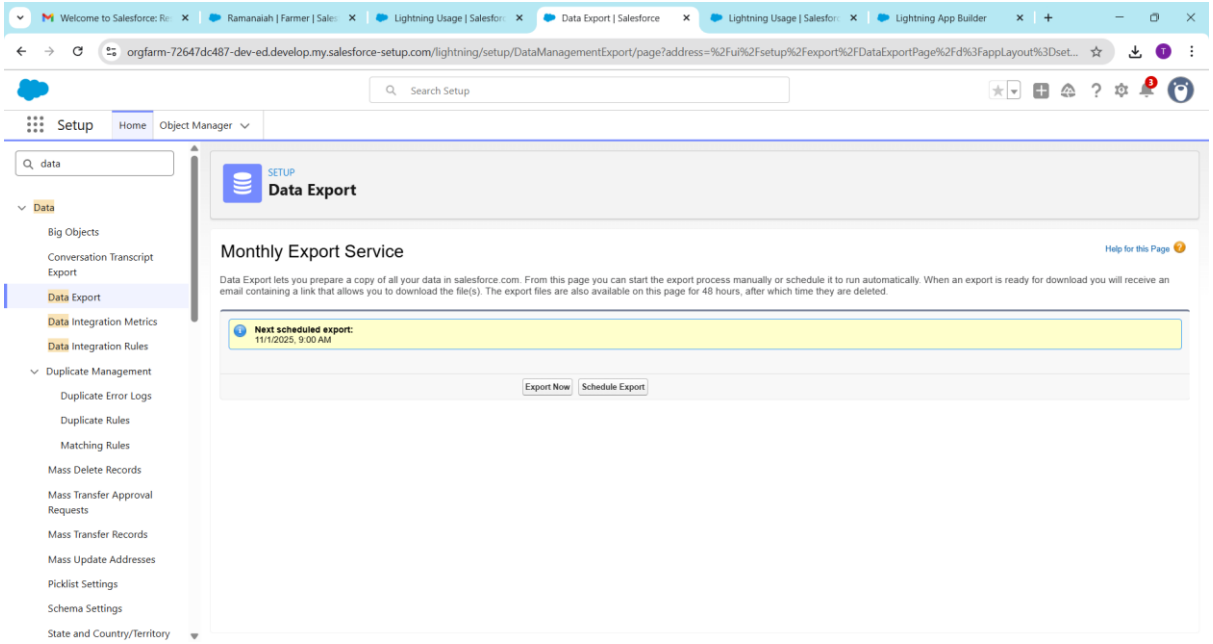
Options:

1. **Manual Export:**
   Setup → Data Export → Choose objects → Export Now
2. **Scheduled Export:** Weekly or monthly automated exports
3. Save CSV / ZIP files securely

Best Practices:

- Backup **critical objects** like Farmer__c, Order__c, Buyer__c, Subsidy_Application__c
- Keep **versioned backups** in case of rollback needs

Example: Backup all **pending subsidy records** before making a mass update.



5⃣Change Sets (Deploy Customizations Between Orgs)

Purpose: Move customizations (fields, flows, layouts) from **Sandbox → Production**.

Step-by-Step Instructions:

1. In **Sandbox → Setup → Outbound Change Sets → New Change Set**
2. Add components (objects, fields, flows, record pages)
3. Upload to **Production org**
4. In **Production → Inbound Change Sets → Validate & Deploy**

Best Practices:

- Test in Sandbox before deploying to Production
- Use descriptive **names and version notes** for each change set

Example: Deploy a new **Farmer Record Page with custom LWC** from Sandbox to Production.

6⃣Unmanaged vs Managed Packages

Type	Description	Use Case
Unmanaged Package	Editable source code	Internal projects or learning
Managed Package	Protected code, includes license management	AppExchange apps or ISV products

Example: Use a **managed package** for a **crop subsidy calculator**, while using **unmanaged packages** for your own custom flows and LWCs.

7⃣ ANT Migration Tool (Command-line Deployment)

Purpose: Automate deployment of metadata between orgs using **XML + CLI**.

Step-by-Step Instructions:

1. Install **Java + ANT + Salesforce Migration Tool**
2. Configure:
    - build.properties → Org credentials
    - build.xml → Metadata components and deployment targets
3. Run commands:
    - ant retrieve → Get metadata from org
    - ant deploy → Deploy to target org

Best For:

- Large deployments
- CI/CD pipelines
- Repeated or automated updates

8⃣ VS Code & Salesforce DX (SFDX)

Purpose: Modern development and deployment for Salesforce developers.

Step-by-Step Instructions:

1. Install **VS Code + Salesforce Extensions**
2. Authenticate your org:
3. sfdx force:auth:web:login -a SandboxOrg
4. Pull metadata from org:
5. sfdx force:source:retrieve -m ApexClass, LightningComponentBundle
6. Make changes → Push to org:
7. sfdx force:source:push

Best Practices:

- Use **scratch orgs** for testing
- Maintain **version control (Git)**

- Integrate with **CI/CD pipelines**

Example: Developer builds **LWC + Apex** in VS Code → Pushes to Sandbox → Tested → Deployed to Production.

Phase 9: Reporting, Dashboards & Security Review – Explained

1. Reports

Purpose: Summarize, analyze, and present Salesforce data.

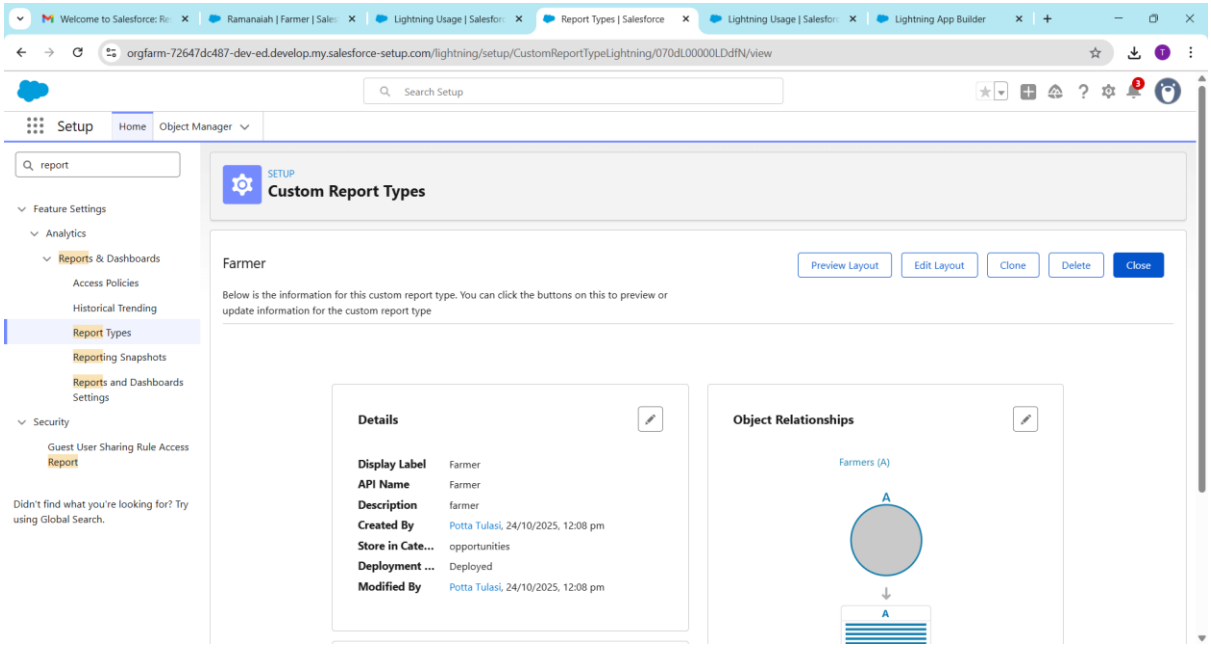
Types of Reports:

Type	Description	Example
Tabular	Simple table, like Excel sheet	List all farmers with phone numbers
Summary	Tabular + grouped data	Total orders per farmer
Matrix	Grouped by rows & columns	Crop quantity per village per buyer
Joined	Combine multiple related reports	Farmer info + Subsidy applications

Steps to Create a Report:

1. Go to **Reports → New Report**
2. Choose **Report Type** (Farmer, Order, Crop, etc.)
3. Filter the data (e.g., only active farmers)
4. Group or summarize data as needed
5. Save & run the report

✅ **Tip:** Use **summary or matrix reports** for deeper insights.

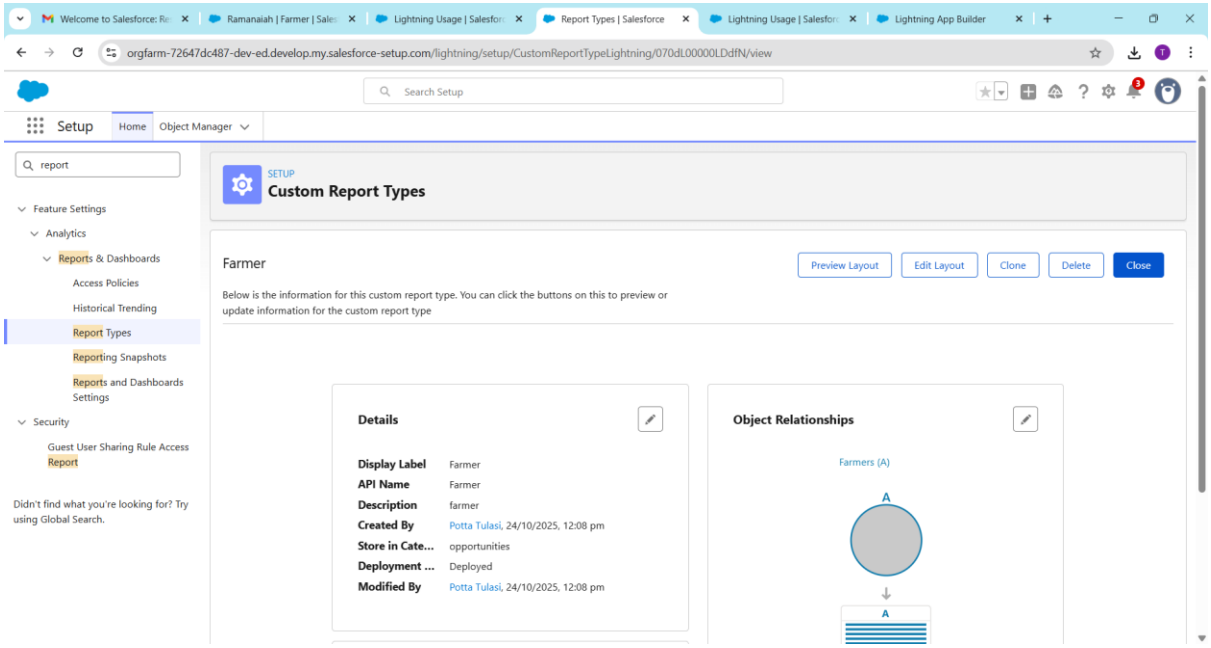


2. Report Types

- **What it is:** Defines **which objects and fields are available** in a report.
- **Standard Report Types:** Prebuilt by Salesforce (Accounts, Contacts, Orders)
- **Custom Report Types:** You can create for custom objects or relationships.

Steps to create a Custom Report Type:

1. Setup → Report Types → New Custom Report Type
2. Choose **Primary Object** (e.g., Farmer__c)
3. Add **related objects** if needed (Orders, Subsidies)
4. Select fields to include



3. Dashboards

Purpose: Visual representation of reports.

- Show charts, graphs, and metrics in **one place**.

Steps to create a Dashboard:

1. Go to **Dashboards → New Dashboard**
2. Give a name, folder, and source report(s)
3. Add **components** (charts, gauges, tables)
4. Customize layout and filters
5. Save & refresh

4. Dynamic Dashboards

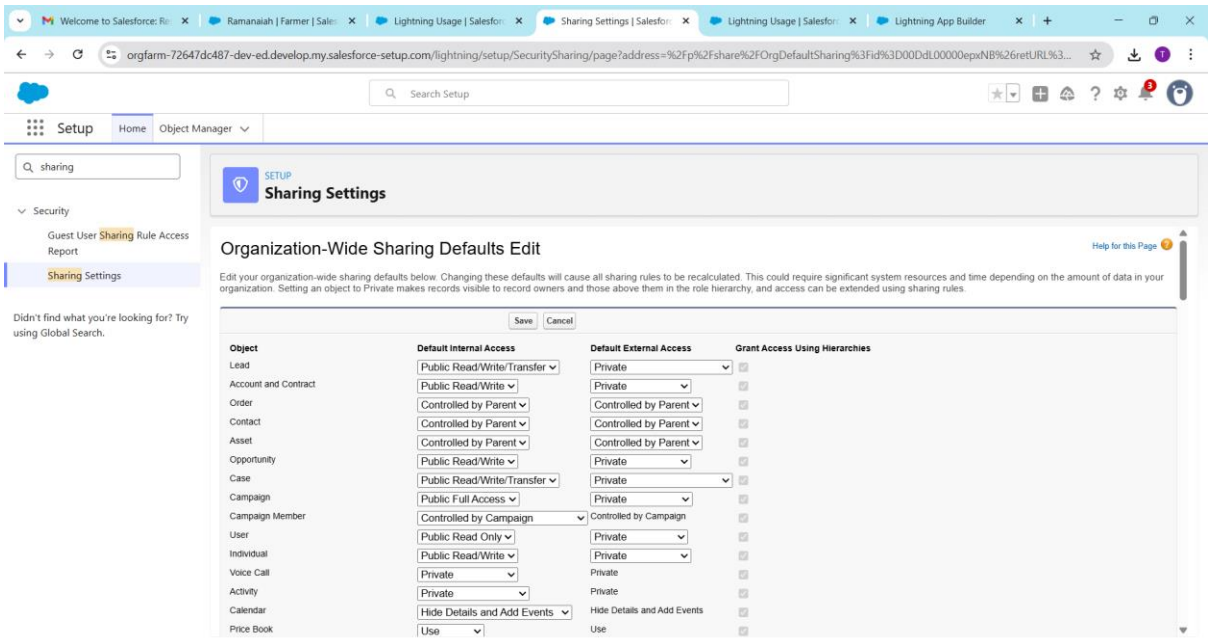
- **What it is:** Dashboards that **show data according to the logged-in user**.
- Example: Sales manager sees only their region's farmers; CEO sees all regions.
- Steps:
    1. Create a dashboard
    2. Check **"Run as Logged-in User"**
    3. Assign permissions

5. Sharing Settings

- **Purpose:** Control **who can see or edit records** in Salesforce.
- Types:
    - **Public Read/Write** → Everyone can view and edit
    - **Public Read Only** → Everyone can view, only owners edit
    - **Private** → Only record owner and above can see

Steps:

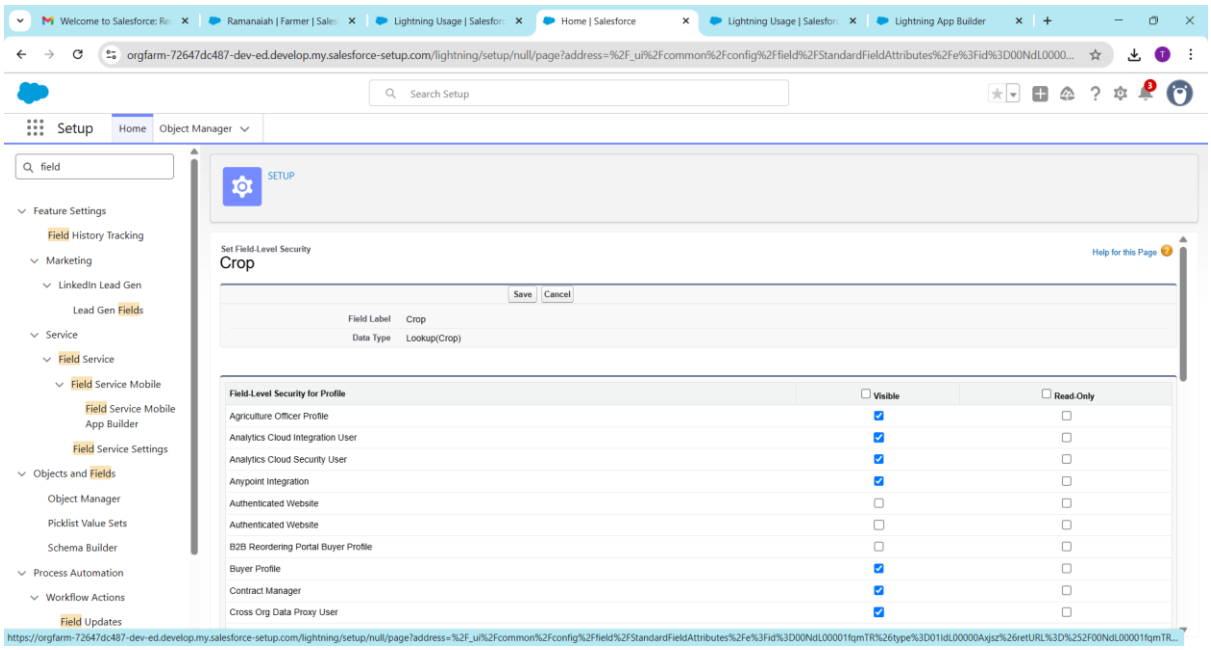
1. Setup → Sharing Settings → Edit
2. Adjust **Organization-Wide Defaults (OWD)**
3. Set sharing rules for exceptions



6. Field Level Security (FLS)

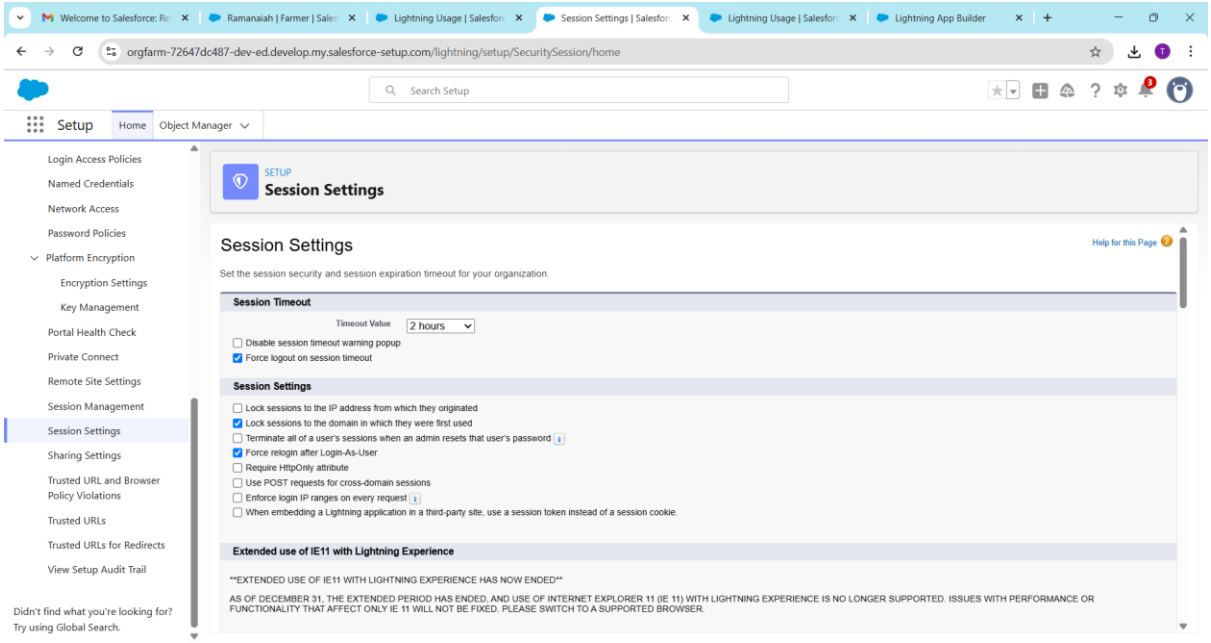
- **Purpose:** Control **who can see or edit specific fields** on a record.
- Steps:
    1. Setup → Object Manager → Object → Fields & Relationships
    2. Click field → Set Field-Level Security
    3. Choose which profiles can view or edit

Example: Only managers can see subsidy amounts.



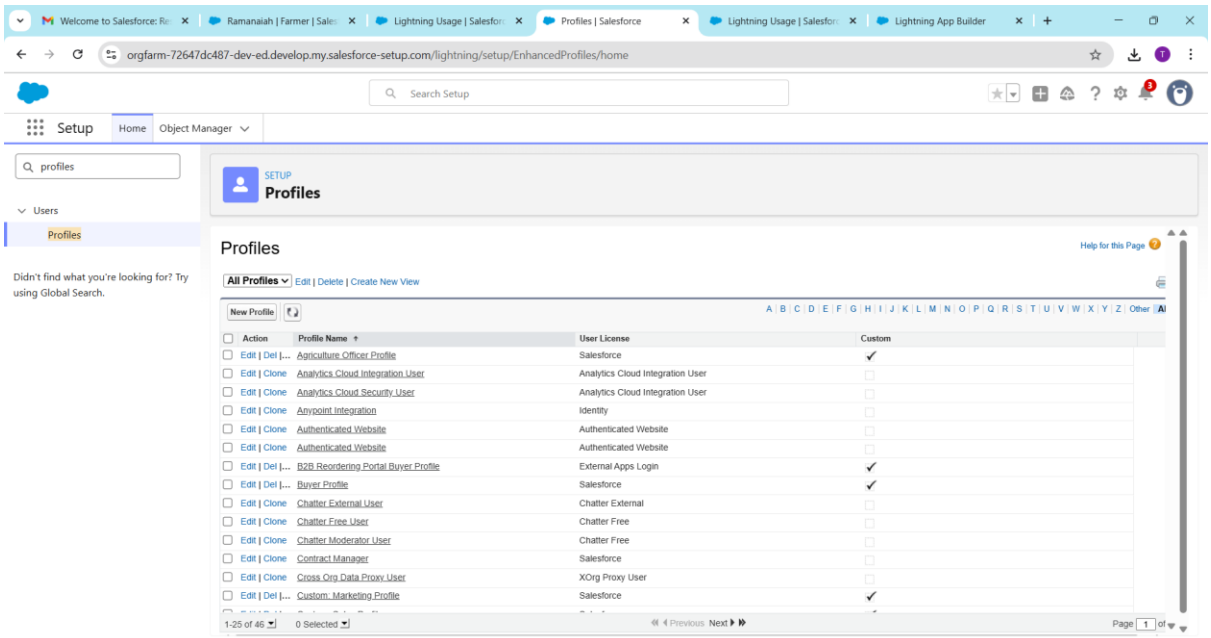
7. Session Settings

- **Purpose:** Control **user session behavior** for security.
- Can set:
    - Session timeout (e.g., 30 mins inactivity)
    - IP address restrictions
- Steps: Setup → Security → Session Settings → Adjust



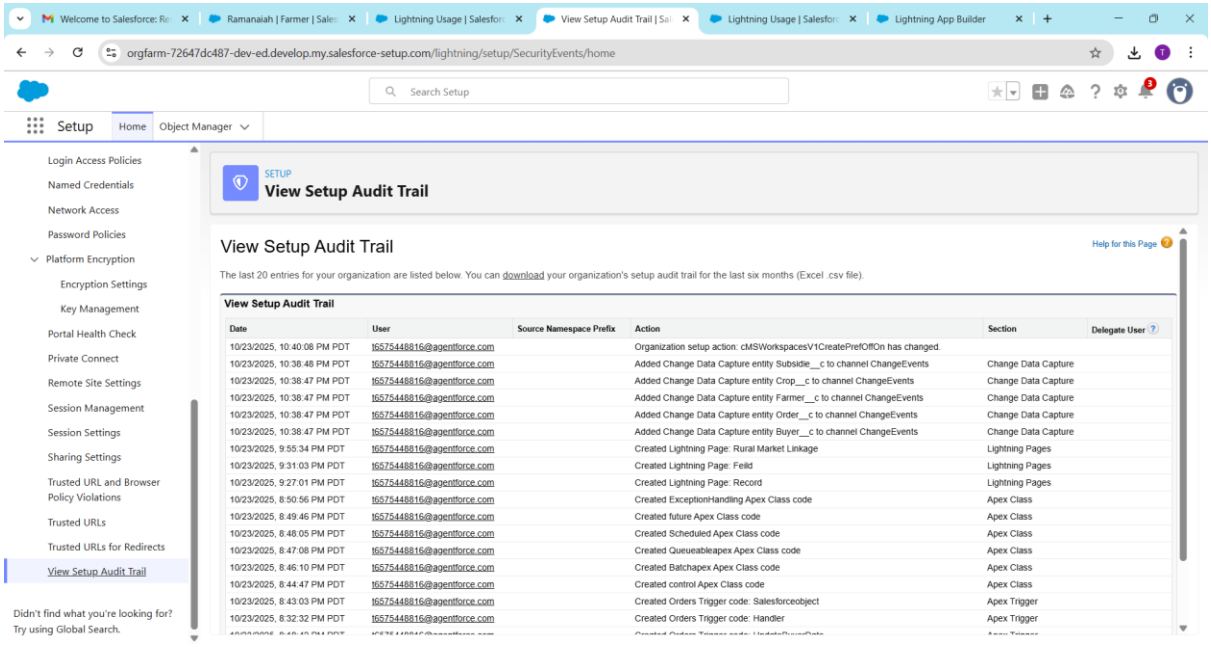
8. Login IP Ranges

- **Purpose:** Restrict Salesforce access to specific IPs.
- Steps:
  1. Setup → Profiles → Choose Profile
  2. Set **Login IP Range** (start IP → end IP)
- Example: Only allow office network to login



9. Audit Trail

- **Purpose:** Track **changes in Salesforce setup** (who changed what and when).
- Steps: Setup → Security → View Setup Audit Trail
- Example: See when a new user was added, or a field was updated

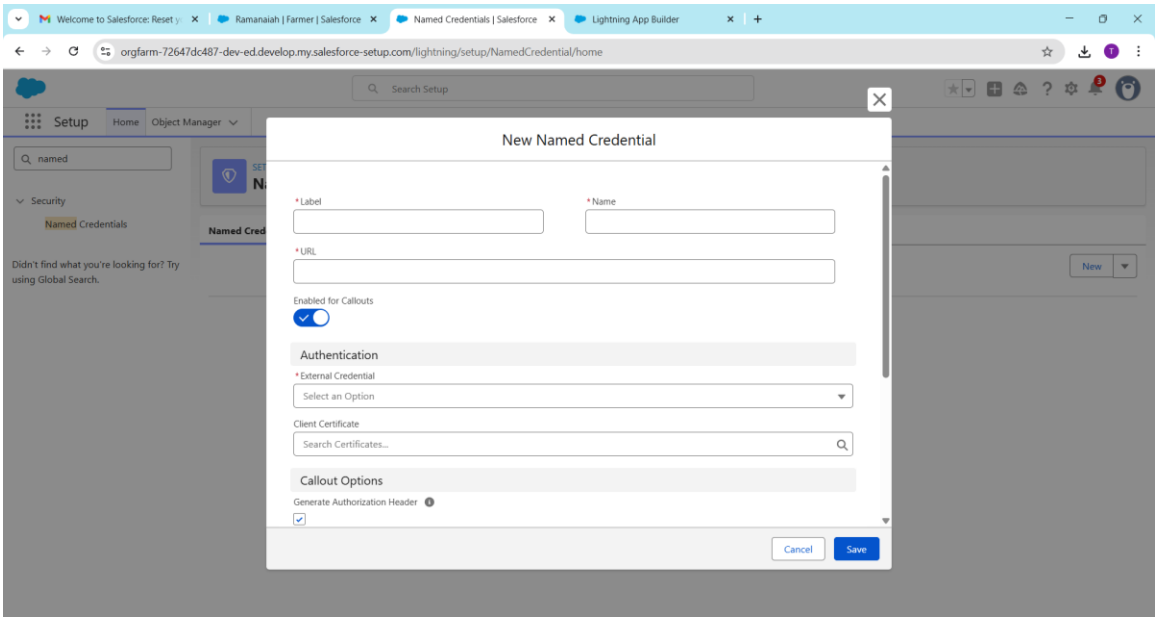


Phase 7: Integration & External Access – Salesforce

Overview:
Phase 7 focuses on connecting Salesforce to external systems, APIs, and databases, while ensuring secure access and real-time updates.

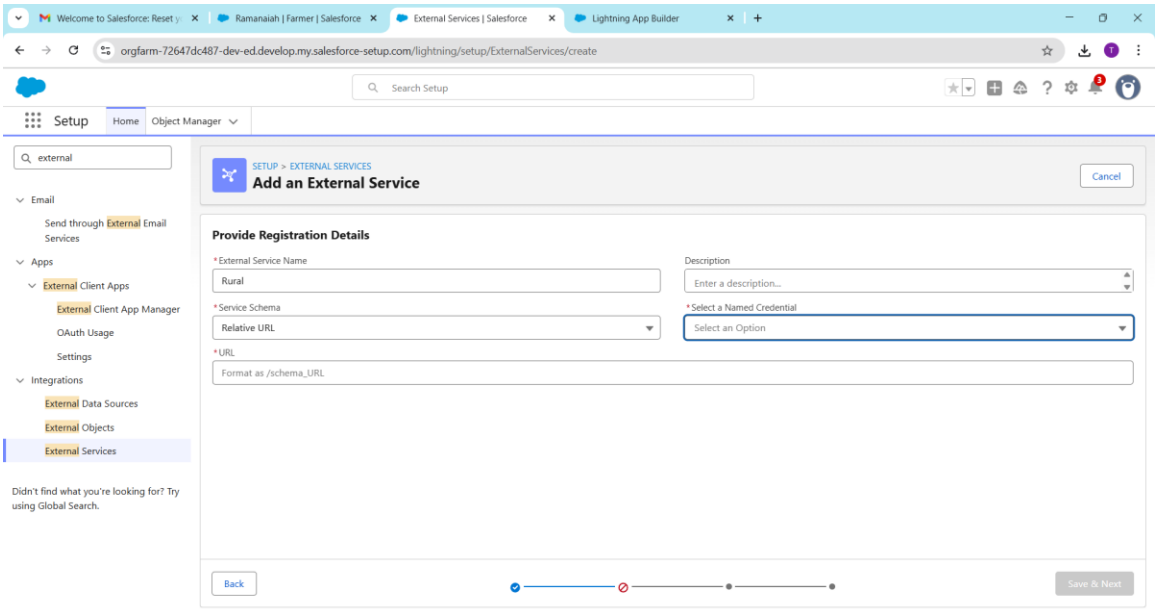
1. **Named Credentials**

- Store login credentials (username, password, OAuth token) for external systems securely.
- Steps:
    1. Go to Setup → Named Credentials → New.
    2. Enter the external system URL.
    3. Choose authentication method.
    4. Use the Named Credential in Apex or Flow.
- Example: Fetch crop prices from a government API.
- Analogy: A safe holding keys for external offices.



2. External Services

- Allows Salesforce to use external API functions without coding.
- Steps:
    1. Import the API definition (OpenAPI/Swagger).
    2. Salesforce generates actions for Flows or LWCs.
- Example: Get rainfall data for farmers' regions.
- Analogy: Ordering from a pre-made menu.



3. Web Services (REST/SOAP)

- REST: Modern, uses JSON. Good for lightweight calls.
- SOAP: Older, uses XML. Structured and formal.
- Steps:
    1. Create an Apex callout or configure external system connection.
    2. Send GET/POST requests.
- Example: Pull crop prices using REST, submit subsidy via SOAP.
- Analogy: WhatsApp message (REST) vs. formal letter (SOAP).

4. **Callouts**

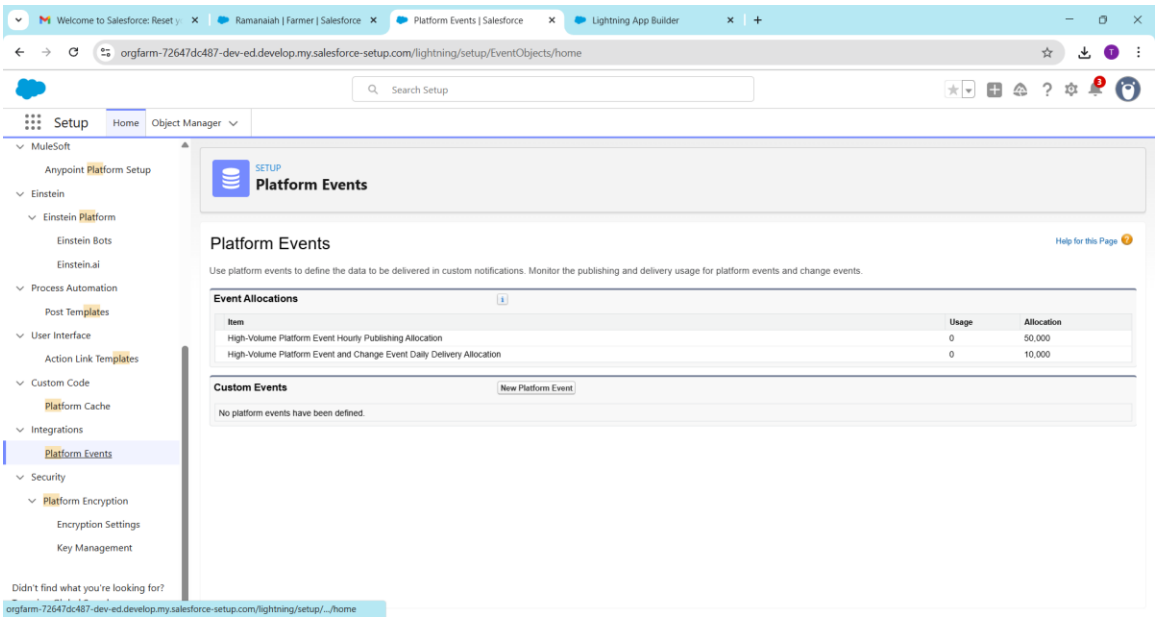
- Salesforce sends HTTP requests to external systems.
- Steps:
    1. Create an Apex class.
    2. Use HttpRequest and HttpResponse.
- Example Code:

```
HttpRequest req = new HttpRequest();
req.setEndpoint('https://api.crops.com/wheat');
req.setMethod('GET');
HttpResponse res = new Http().send(req);
System.debug(res.getBody());
```

Analogy: Making a phone call to request information.

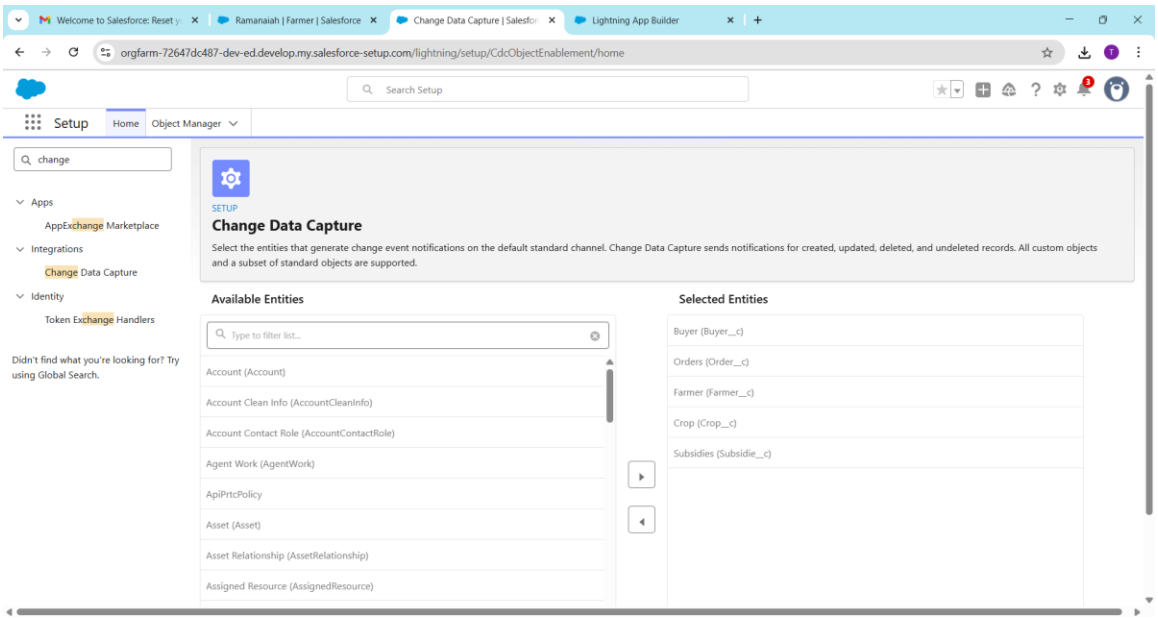
5. **Platform Events**

- Real-time messaging system inside Salesforce.
- Steps:
    1. Create Platform Event object.
    2. Publish event from Apex, Flow, or LWC.
    3. Subscribe using Process Builder, Flow, or external apps.
- Example: New order triggers buyer notification instantly.
- Analogy: Bell rings to alert everyone in the office.



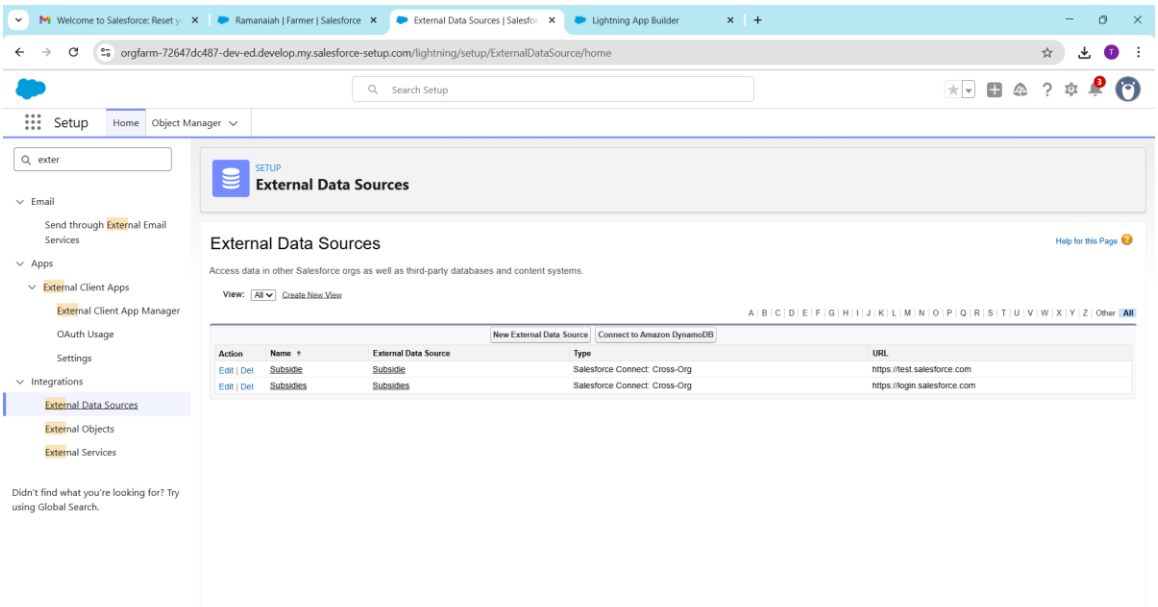
6. Change Data Capture (CDC)

- Tracks changes to Salesforce records in real-time.
- Steps:
    1. Enable CDC for objects.
    2. Subscribe to events in external systems or Salesforce.
- Example: Update in order quantity automatically notifies warehouse system.
- Analogy: Security camera sends alerts when changes occur.



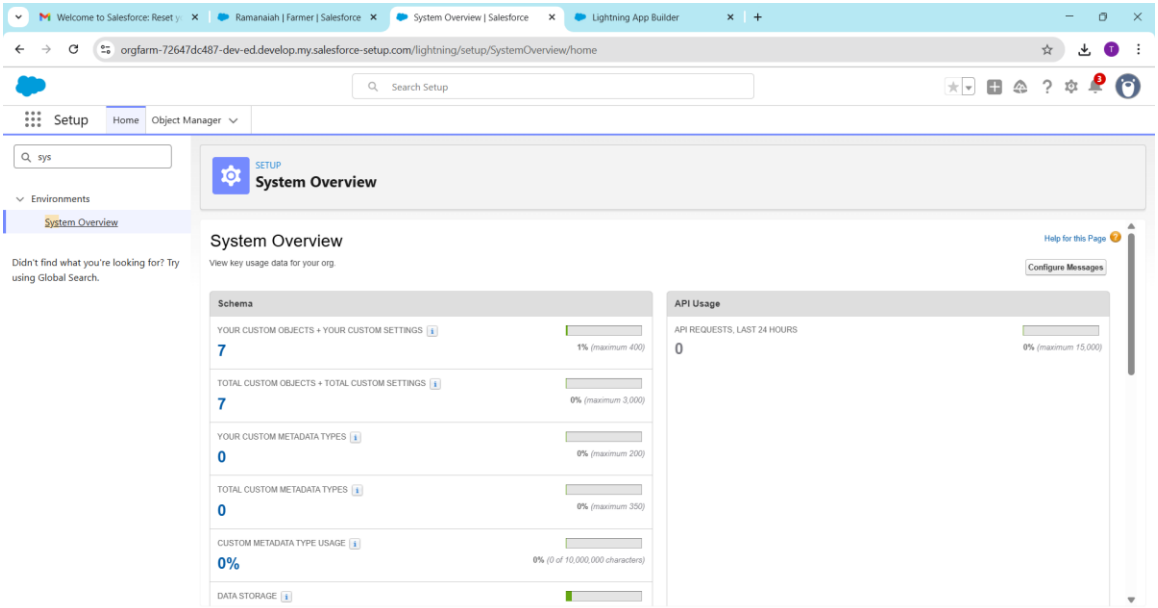
7. Salesforce Connect

- Access external database data without storing it in Salesforce.
- Steps:
    1. Configure External Data Source.
    2. Create External Objects mapped to the external database.
- Example: View warehouse inventory stored in Oracle DB.
- Analogy: Looking through a window without moving items.



API Limits

- Salesforce restricts the number of API calls per 24 hours.
- Steps:
    1. Check API usage in Setup → System Overview.
    2. Optimize callouts to avoid exceeding limits.
- Example: Free org has 15,000 API calls/day.
- Analogy: Water tank with maximum daily capacity.



9. **OAuth & Authentication**

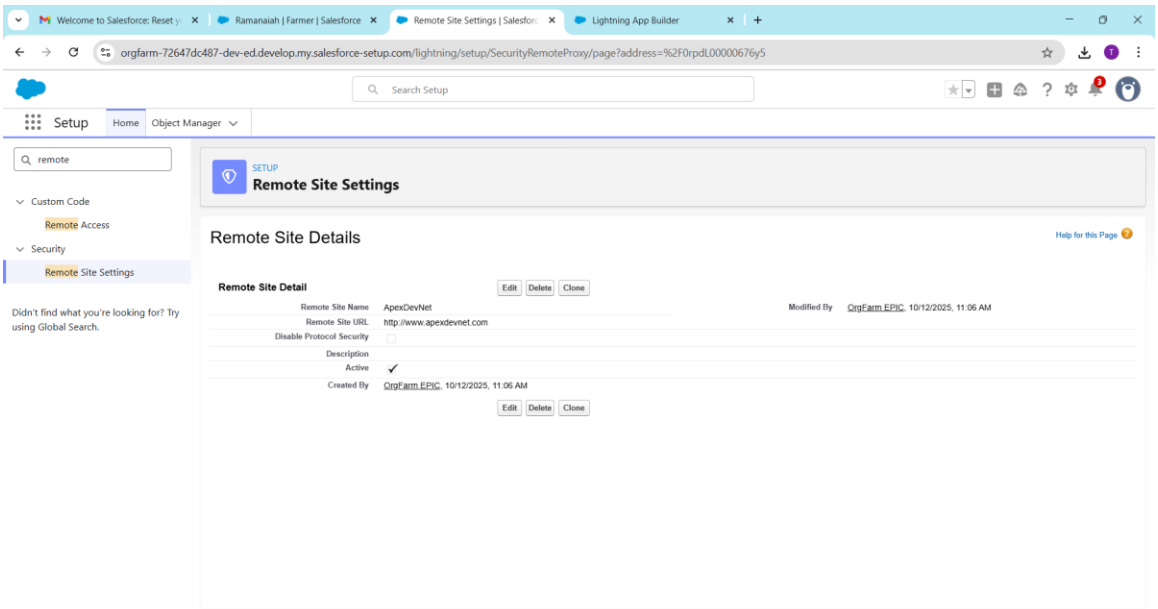
- Secure way to connect Salesforce to other apps using tokens.
- Steps:
    1. Create Connected App.
    2. Configure OAuth scopes.
    3. Use token in Apex or external system.
- Example: Salesforce connects to Google Sheets without sharing passwords.
- Analogy: Temporary access card for secure login.

10. **Remote Site Settings**

- Salesforce allows callouts only to whitelisted URLs.
- Steps:
    1. Setup → Remote Site Settings → New.
    2. Enter external system URL.
- Example: Must whitelist https://api.crops.com.
- Analogy: Approved websites your office is allowed to communicate with.

Example Workflow for Rural Market Linkage:

1. Farmer places an order → Platform Event triggers buyer notification.
2. Salesforce fetches live crop price → Callout using Named Credentials.
3. Order quantity changes → CDC notifies warehouse system.
4. Inventory data lives in external DB → Salesforce Connect displays it.



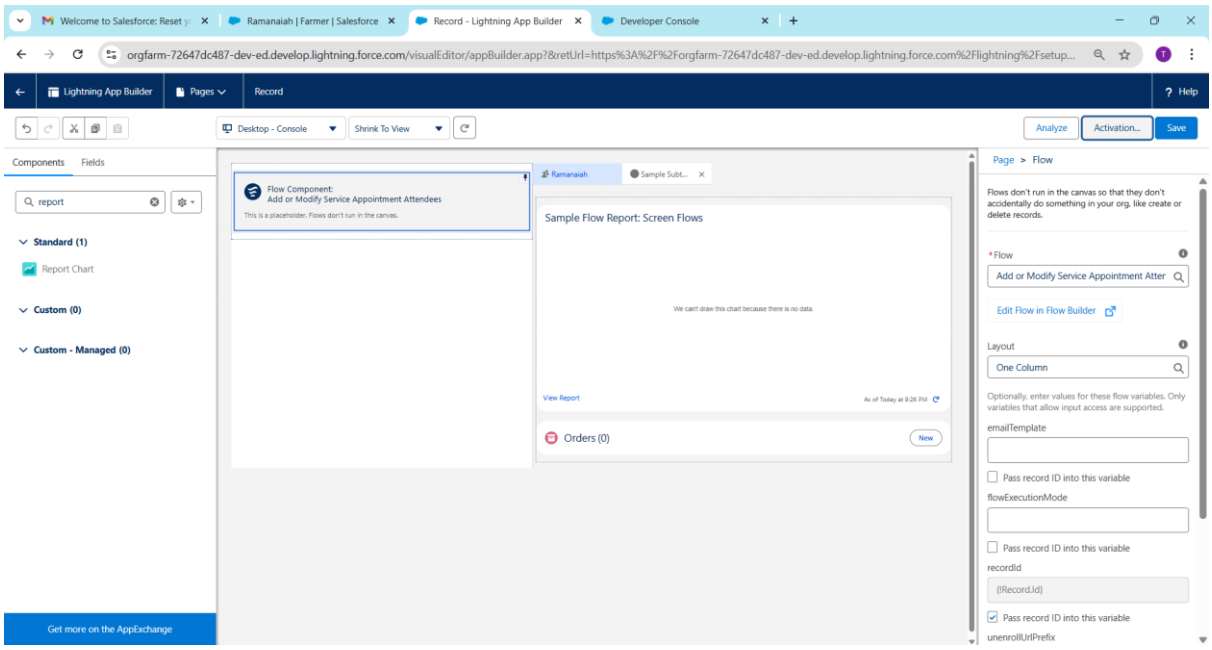
Summary:
Phase 7 enables Salesforce to securely fetch/send data, integrate with external apps and databases, react in real-time, and follow security rules while respecting API limits.

Phase 6: User Interface (UI) Development in Salesforce

1. Lightning App Builder

- A **point-and-click tool** to create custom pages in Salesforce.
- You can design pages for **Apps, Record Pages, or Home Pages** without coding.
- **Components** like lists, charts, or custom LWC can be dragged onto the page.

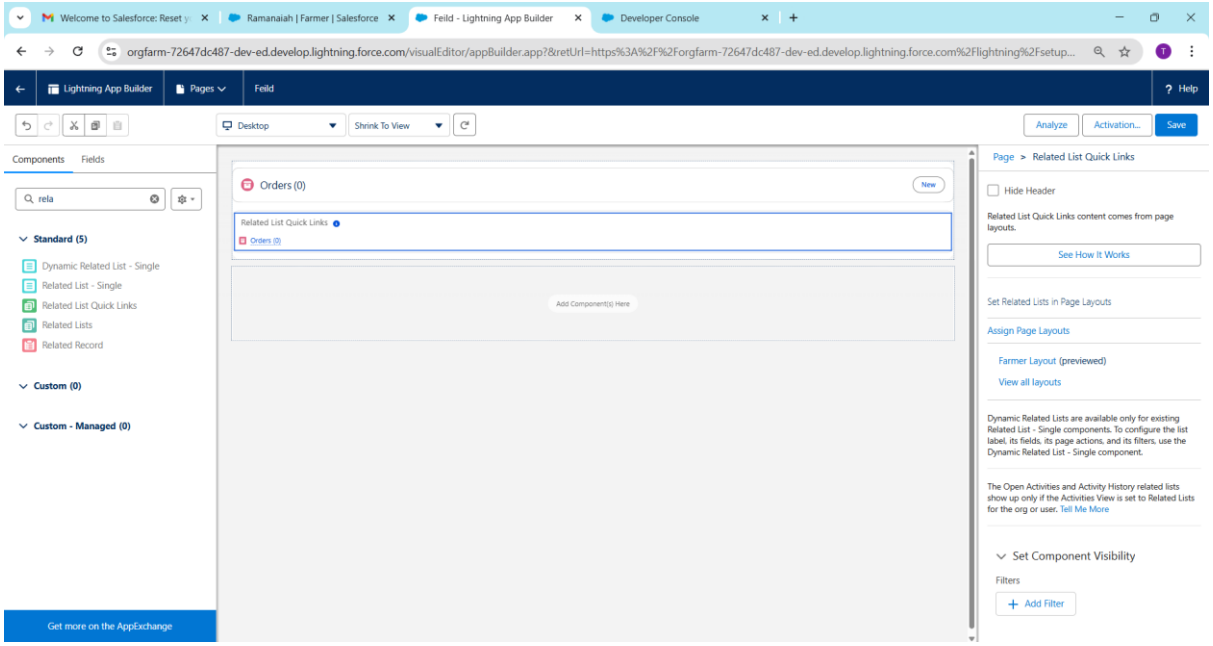
Example: For your Rural Market Linkage project, you could create a **dashboard page** showing farmers' orders and buyer stats.



2. Record Pages

- A **record page** is the layout you see when opening a specific record (e.g., Farmer__c or Order__c).
- You can **customize what fields, related lists, and components appear**.
- Can be different for **desktop, mobile, or Lightning apps**.

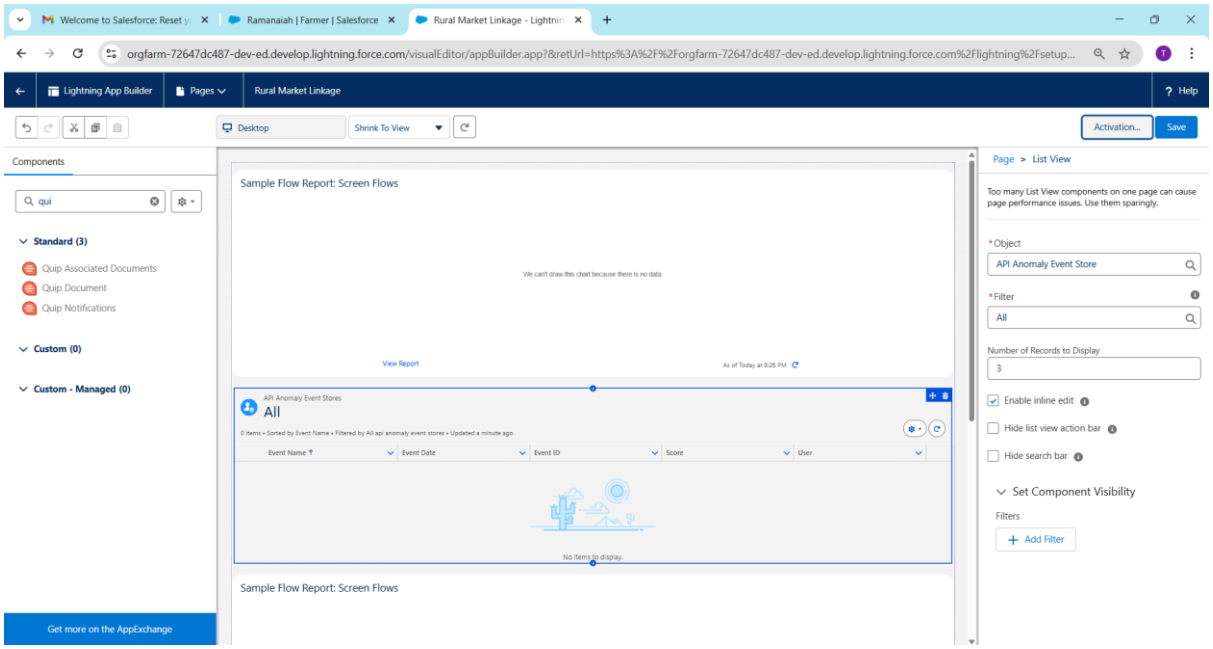
Example: A Farmer record page could show the farmer's details, crops, past orders, and subsidy applications in one view.



3. Tabs

- Tabs are like **menu items** in Salesforce.
- Can display **Objects, Visualforce pages, Lightning pages, or Web URLs**.
- Helps users **navigate between apps or features**.

Example: Create tabs for **Farmers, Buyers, Orders, and Subsidy Applications** in your app.



4. Home Page Layouts

- The **homepage is the first page users see** when logging in.
- Can include **charts, reports, lists, tasks, news**, and **custom components**.

- Helps users **quickly access relevant info**.

Example: Your home page could show **latest orders, pending subsidies, and notifications for buyers or farmers**.

5. Utility Bar

- A **persistent bar at the bottom of the screen** for quick actions.
- Can include **custom components, calculators, chatbots, or quick links**.
- Makes common tasks accessible **anywhere in the app**.

Example: A utility bar could have **"Quick Order Entry"** or **"Farmer Lookup"** for your team.

6. LWC (Lightning Web Components)

- Modern **JavaScript-based components** for building Salesforce UI.
- Faster and more **standard-compliant** than older Aura components.
- Can be used on **record pages, home pages, or apps**.

Example: Create an LWC called **"FarmerOrderSummary"** to show recent orders with dynamic updates.

7. Apex with LWC

- **Apex** is Salesforce's server-side code.
- LWCs are **frontend components** and can **call Apex** to get or update data.
- Enables **complex logic** behind simple UI.

Example: Your LWC can call an Apex method to **fetch total crops available per farmer**.

```
public with sharing class FarmerController {
   @AuraEnabled(cacheable=true)
   public static List<Farm__c> getAllCrops() {
      return [SELECT Name, Quantity__c FROM Farm__c];
   }
}
```

8. Events in LWC

- LWCs communicate using **events**.
- **Parent-child communication:** Child component can send data to parent using **Custom Events**.
- **Sibling communication:** Often goes through a **common parent** or **pub-sub model**.

Example: When a farmer selects a crop in one LWC, another LWC showing **available buyers** updates automatically.

9. Wire Adapters

- **Reactive data binding** in LWC.
- Automatically fetches **data from Salesforce objects** or Apex.
- Uses @wire decorator to bind data to the component.

```
import { LightningElement, wire } from 'lwc';
import getAllCrops from '@salesforce/apex/FarmerController.getAllCrops';

export default class CropList extends LightningElement {
  @wire(getAllCrops) crops;
}
```

- **Advantage:** No need to manually refresh data.

10. Imperative Apex Calls

- Unlike wire adapters, **imperative calls** are made **on-demand**, usually after a user action (click, form submit).
- Gives **more control over execution and error handling**.

```
handleClick() {
  getAllCrops()
    .then(result => { this.crops = result; })
    .catch(error => { this.error = error; });
}
```

Example: Fetch crops **only when a user clicks "Show Available Crops"**.

11. Navigation Service

- Lets LWCs **navigate programmatically** to pages, records, or apps.
- Uses NavigationMixin in JavaScript.

```
import { NavigationMixin } from 'lightning/navigation';
export default class NavigateToRecord extends NavigationMixin(LightningElement) {
  goToFarmerRecord() {
    this[NavigationMixin.Navigate]({
      type: 'standard__recordPage',
      attributes: {
        recordId: '001xx000003DGSzAAO',
        objectApiName: 'Farmer__c',
        actionName: 'view'
      }
    });
  }
}
```

Example: Clicking a crop in your LWC navigates to the **Farmer record page** showing orders for that crop.

🌾 Phase 5: Apex Programming (Developer)

🟪 What is Apex?

Apex is Salesforce's own programming language — similar to Java.
You use it when **Flows or Process Builder cannot do something** — for example:

- Updating hundreds of records at once
- Performing complex calculations
- Sending automatic messages after records are saved

In your project, Apex helps:

- Auto-update Buyer or Farmer information
- Schedule monthly reminders
- Handle large data updates
- Send background notifications or messages

⚙️ 1. Classes & Objects

📘 What is a Class?

A class is a *container* for logic (like a box that stores instructions).

📗 What is an Object?

An object is a *data table* — like **Farmer__c**, **Buyer__c**, **Order__c**, **Subsidy_Application__c**.

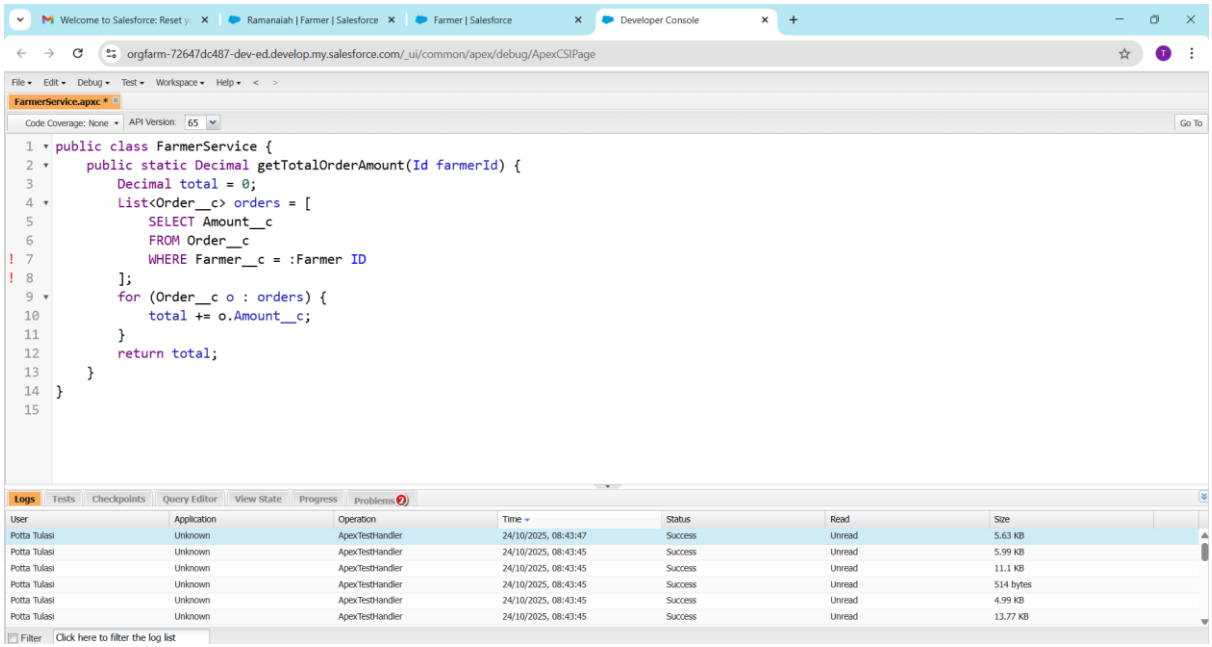
💡 Example:

Let's calculate the total value of all orders for one farmer.

```
public class FarmerService {
   public static Decimal getTotalOrderAmount(Id farmerId) {
      Decimal total = 0;
      List<Order__c> orders = [SELECT Amount__c FROM Order__c WHERE Farmer__c =
:farmerId];
      for(Order__c o : orders) {
```

```
            total += o.Amount__c;
        }
        return total;
    }
}
```

▯ This means:
When you pass a Farmer ID → Apex looks up all that farmer's orders → adds the total → and gives you the result.



⚡ 2. Apex Triggers

📘 What is a Trigger?

A Trigger is code that runs **automatically** when data changes in Salesforce.

It can run:

- **Before Insert / Update / Delete**
- **After Insert / Update / Delete**

💡 Example:

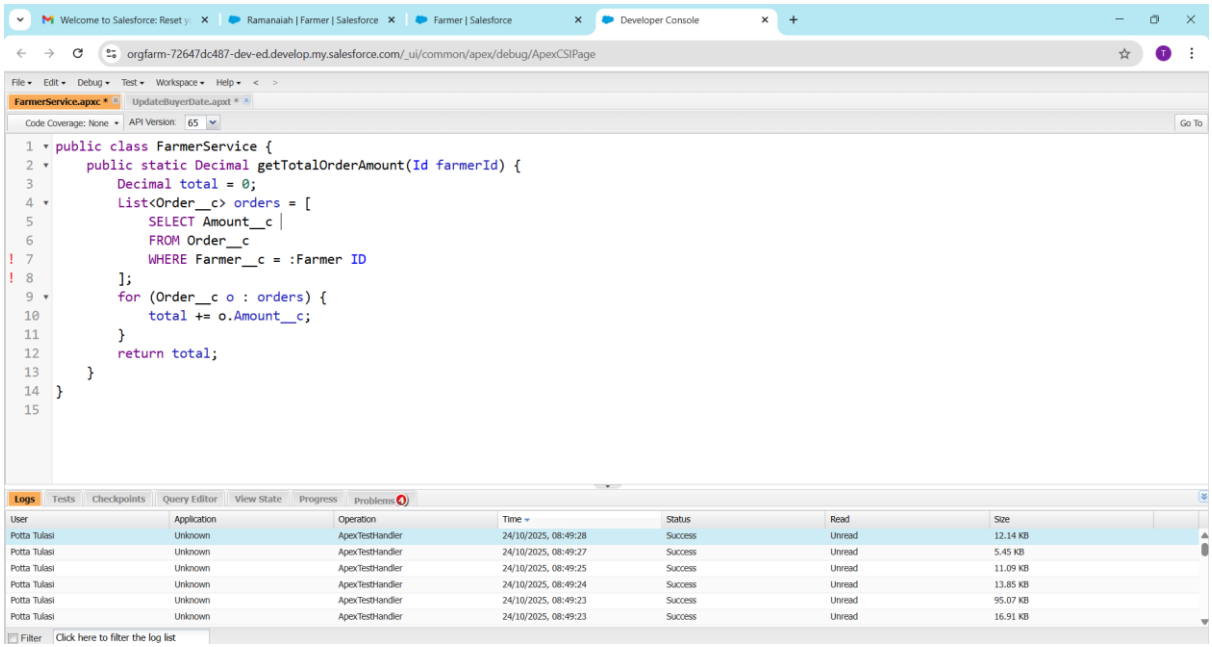
When a new Order is created → update the Buyer's "Last Order Date".

```
trigger UpdateBuyerDate on Order__c (after insert) {
    Set<Id> buyerIds = new Set<Id>();
    for(Order__c o : Trigger.new){
```

```
        if(o.Buyer__c != null){
            buyerIds.add(o.Buyer__c);
        }
    }

    List<Buyer__c> buyers = [SELECT Id, Last_Order_Date__c FROM Buyer__c WHERE Id IN :buyerIds];
    for(Buyer__c b : buyers){
        b.Last_Order_Date__c = System.today();
    }
    update buyers;
}
```

 This means:
Whenever a new Order is saved → Apex finds who the Buyer is → updates Buyer's "Last Order Date" automatically.



3. Trigger Design Pattern

Instead of writing long logic inside the trigger, we **move the code to a separate class** (for neatness).

Trigger File:

```
trigger OrderTrigger on Order__c (after insert) {
    OrderHandler.afterInsert(Trigger.new);
```

```
}
```

Handler Class:

```
public class OrderHandler {
    public static void afterInsert(List<Order__c> orders) {
        // Your logic here
    }
}
```

 This makes code easier to read and reuse.

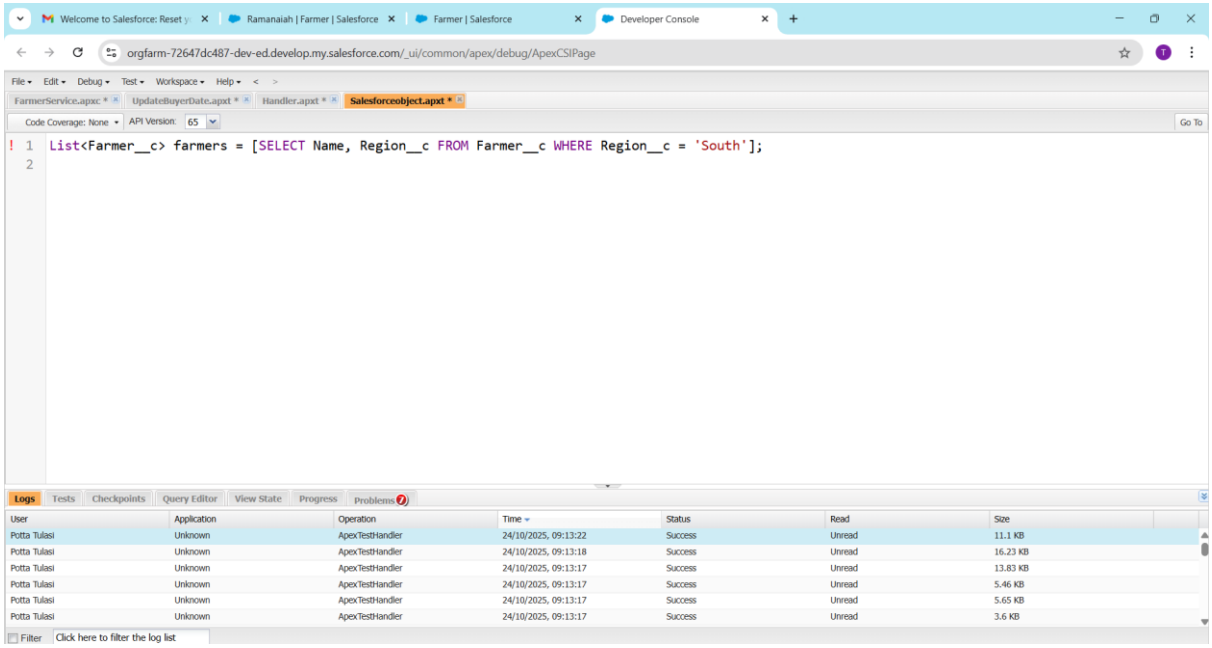
🔑 4. SOQL and SOSL

SOQL (Salesforce Object Query Language)

Used to **get data from one object**.

```
List<Farmer__c> farmers = [SELECT Name, Region__c FROM Farmer__c WHERE Region__c = 'South'];
```

 Meaning: Get all farmers from the “South” region.



SOSL (Salesforce Object Search Language)

Used to **search text in multiple objects**.

```
List<List<SObject>> result = [FIND 'Ravi' IN ALL FIELDS RETURNING Farmer__c(Name), Buyer__c(Name)];
```

 Meaning: Find “Ravi” in both Farmer and Buyer records.

5. Collections

Apex can store multiple records in memory using collections.

Type	Meaning	Example
List	Ordered list of records	List<Order__c> orders = new List<Order__c>();
Set	Unique values only	Set<Id> ids = new Set<Id>();
Map	Key → Value pairs	Map<Id, Farmer__c> farmerMap = new Map<Id, Farmer__c>(farmers);

 In your project: You use a **Set** to store unique Buyer IDs before updating them.

6. Control Statements

These are basic programming conditions — “if”, “for”, “while”.

```
for(Order__c o : orders){
   if(o.Amount__c > 5000){
      o.Status__c = 'High Value';
   } else {
      o.Status__c = 'Normal';
   }
}
update orders;
```

 Meaning:
If an order is above ₹5000 → mark it as *High Value*, else *Normal*.

7. Batch Apex

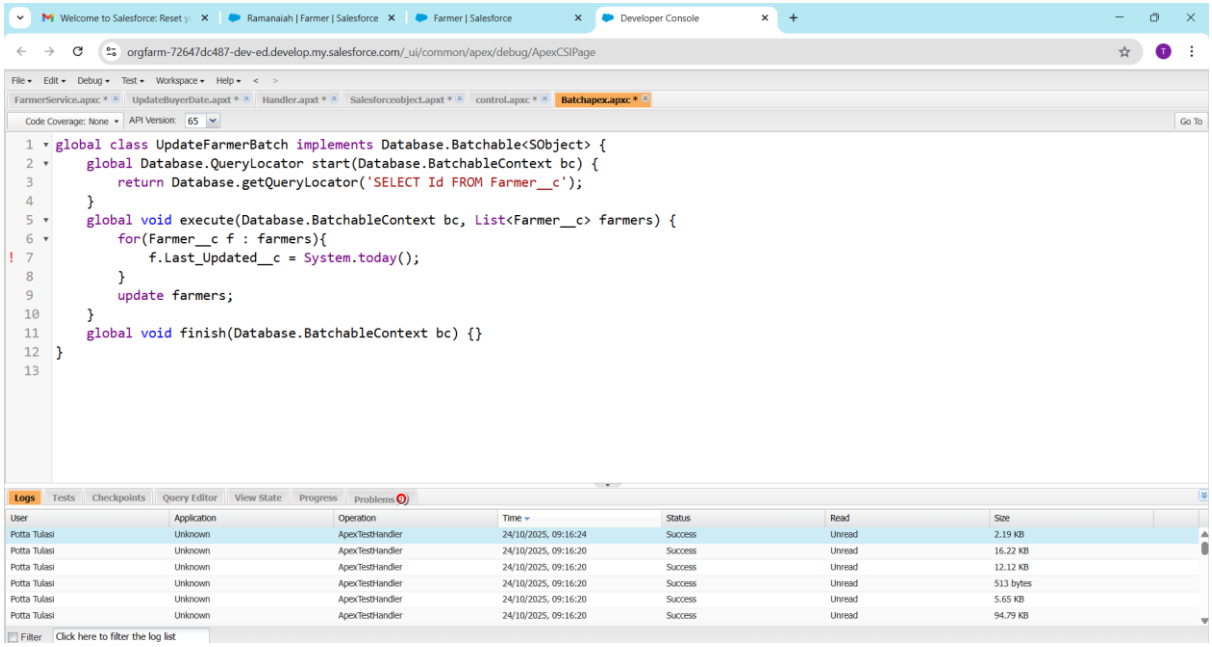
Used when you have **thousands of records** to update.
Salesforce runs them in **chunks (batches)** to prevent system overload.

```
global class UpdateFarmerBatch implements Database.Batchable<SObject> {
   global Database.QueryLocator start(Database.BatchableContext bc) {
      return Database.getQueryLocator('SELECT Id FROM Farmer__c');
   }
```

```
    global void execute(Database.BatchableContext bc, List<Farmer__c> farmers) {
        for(Farmer__c f : farmers){
            f.Last_Updated__c = System.today();
        }
        update farmers;
    }
    global void finish(Database.BatchableContext bc) {}
}
```

 You can run this batch to refresh farmer info:

Database.executeBatch(new UpdateFarmerBatch(), 100);



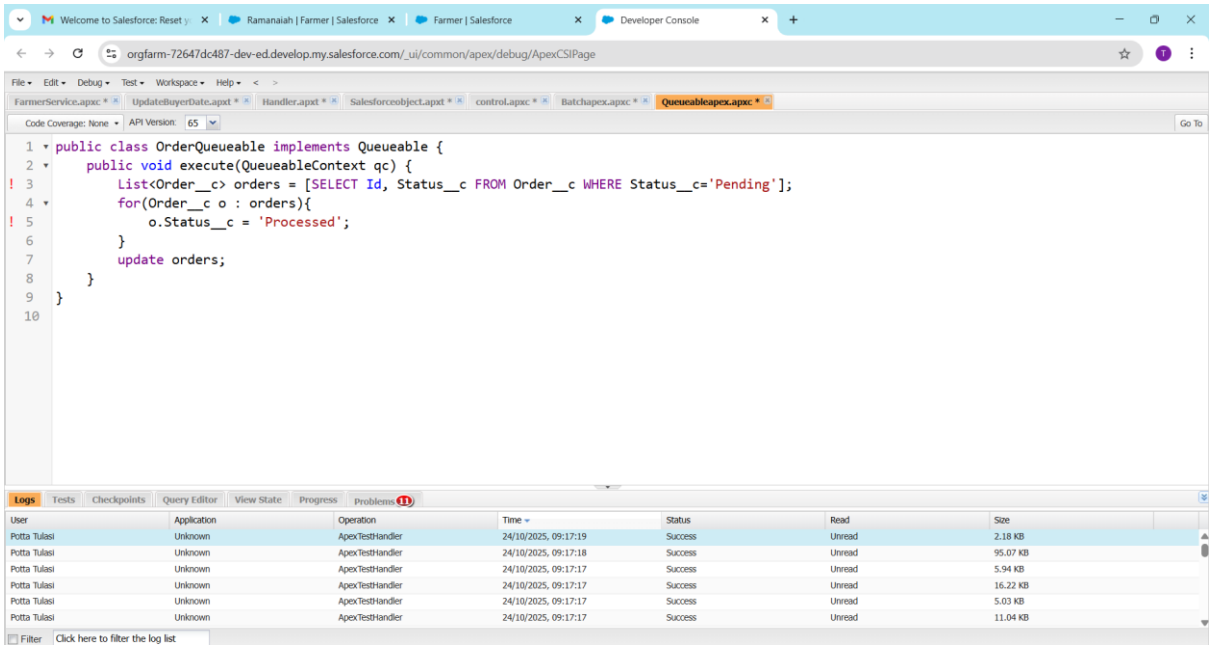
8. Queueable Apex

Used for **background jobs** that can be started by system logic.

```
public class OrderQueueable implements Queueable {
    public void execute(QueueableContext qc) {
        List<Order__c> orders = [SELECT Id, Status__c FROM Order__c WHERE Status__c='Pending'];
        for(Order__c o : orders){
            o.Status__c = 'Processed';
        }
        update orders;
```

```
    }
}
```

 This runs in the background — updating pending orders automatically.



9. Scheduled Apex

Runs Apex code **at a fixed time** (daily, weekly, monthly).

```
global class MonthlyReminder implements Schedulable {
    global void execute(SchedulableContext sc) {
        System.enqueueJob(new OrderQueueable());
    }
}
```

 Schedule it to run on the 1st of every month at 9 AM.

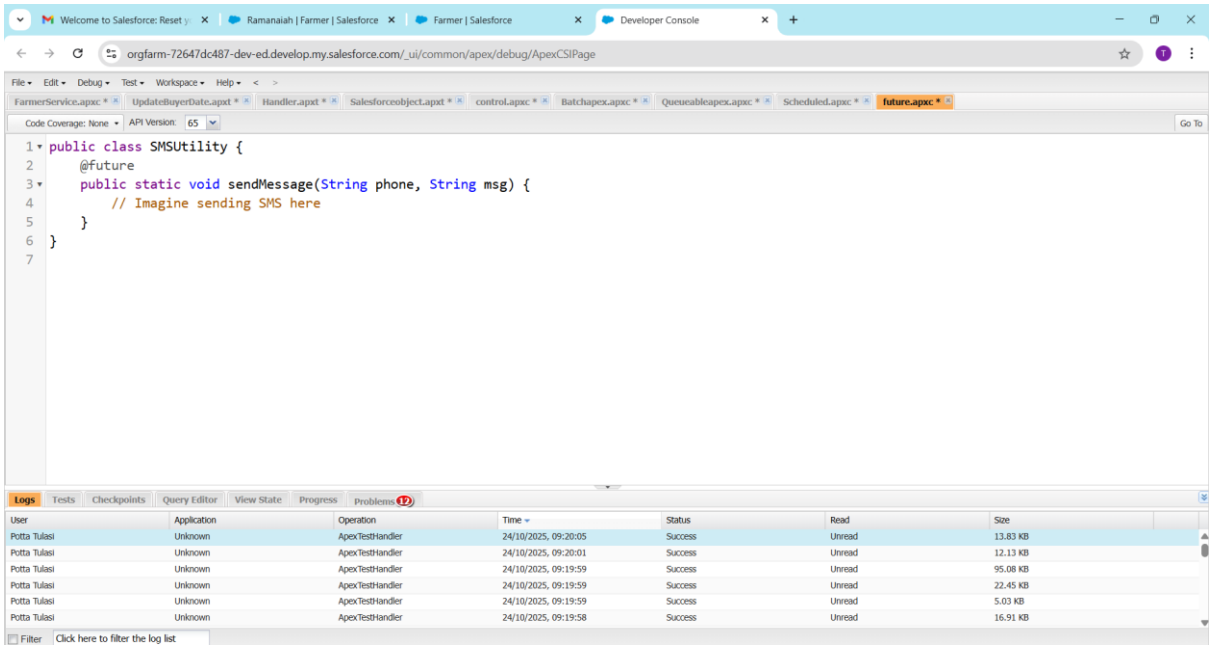
10. Future Methods

Used to run **small tasks in the background**, like sending SMS or API calls.

```
public class SMSUtility {
    @future
    public static void sendMessage(String phone, String msg) {
        // Imagine sending SMS here
    }
```

}

 Example: After an order is placed → send a thank-you SMS.



11. Exception Handling

Used to stop code from crashing when there's an error.

try {

update farmers;

} catch(DmlException e) {

System.debug('Error: ' + e.getMessage());

}

 If update fails, Salesforce shows a friendly message instead of crashing.

12. Test Classes

Before deploying Apex code, Salesforce *requires* 75% test coverage.

Example Test:

@isTest

public class TestOrderTrigger {

@isTest static void testOrderCreation() {

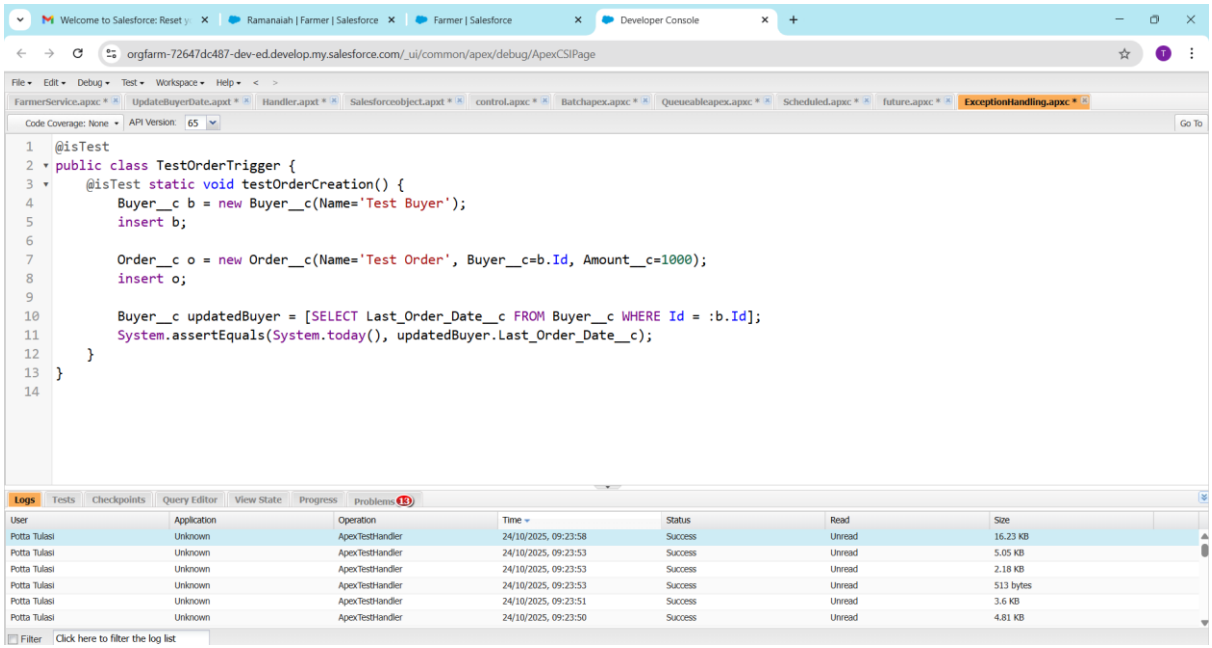
Buyer__c b = new Buyer__c(Name='Test Buyer');

insert b;

```
        Order__c o = new Order__c(Name='Test Order', Buyer__c=b.Id, Amount__c=1000);
        insert o;

        Buyer__c updatedBuyer = [SELECT Last_Order_Date__c FROM Buyer__c WHERE Id = :b.Id];
        System.assertEquals(System.today(), updatedBuyer.Last_Order_Date__c);
    }
}
```

 Tests check that your trigger updates Buyer's "Last Order Date" correctly.



⚡ 13. Asynchronous Processing Types

Type	Runs In Background	Use Case
Future	✅ Yes	Simple tasks like SMS
Batch	✅ Yes	Large data processing
Queueable	✅ Yes	Medium jobs, can chain
Scheduled	✅ Yes	Time-based execution

 Example:
Batch updates Farmer info → Queueable sends notification → Scheduled runs monthly.

🔁 14. Full Automation Flow in Project

1⃣ User creates Order →

2⃣ Trigger runs →

3⃣ Handler updates Buyer →

4⃣ Queueable Apex sends confirmation →

5⃣ Batch Apex updates all related Farmers monthly →

6⃣ Test Class validates logic before deployment

✅ 15. Outcome

After Phase 5:

- Your Salesforce backend becomes *intelligent and automated*.
- Farmers, Buyers, and Government officials don't need to do manual updates.
- All actions (orders, approvals, updates) happen automatically through Apex code.

📘 Rural Market Linkage – Phase 4: Process Automation (Admin)

◆ 1. Phase Overview

Objective:
Automate manual tasks and ensure data consistency using Salesforce automation tools such as Validation Rules, Process Builder, Approval Processes, and Flows.

Scope:
This phase focuses on automating processes for:

- Farmer registration
- Order confirmation and tracking
- Subsidy approvals
- Buyer communication

◆ 2. Tools Used

Tool	Purpose
Validation Rules	Data entry control
Workflow Rules	Simple automation
Process Builder	Multi-step automation
Approval Process	Manager approval logic
Flow Builder	Complex automation & screen flows
Email Alerts	Automated email communication
Field Updates	Automatic record changes
Tasks	Follow-up reminders
Custom Notifications	In-app alerts for users

◆ 3. Validation Rules

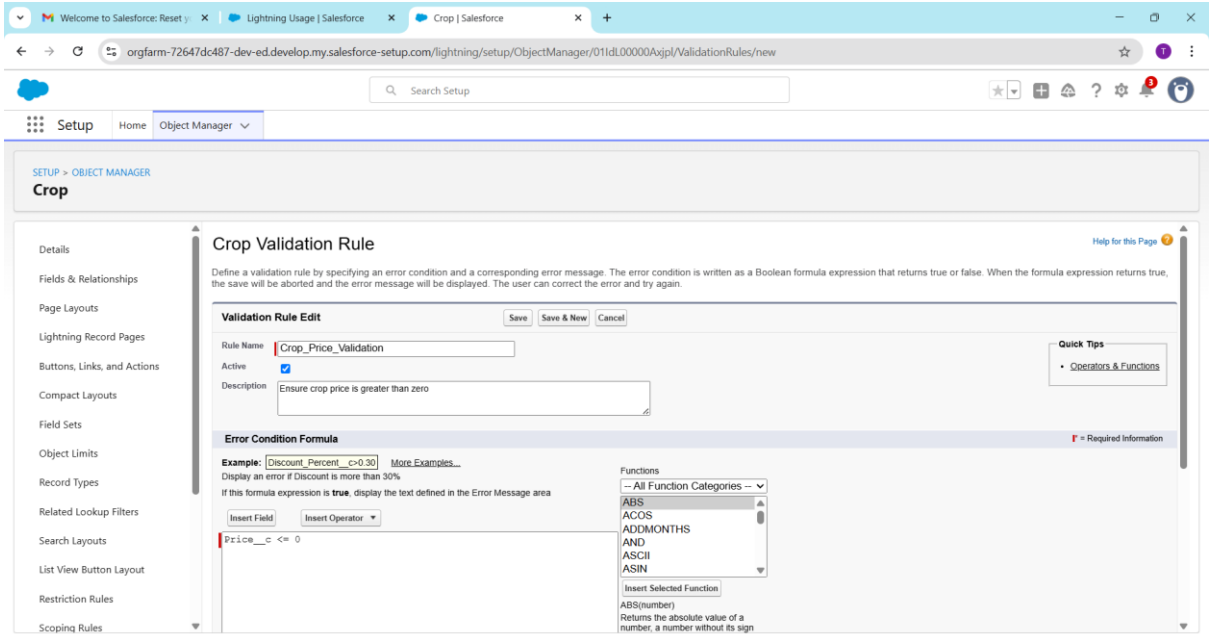
Object: Farmer__c

Purpose: Ensure mandatory data entry

Rule Name: Phone_Required

Formula:

ISBLANK(Phone__c)**Error Message:** “Please enter a valid phone number.”
Error Location: Field → Phone__c

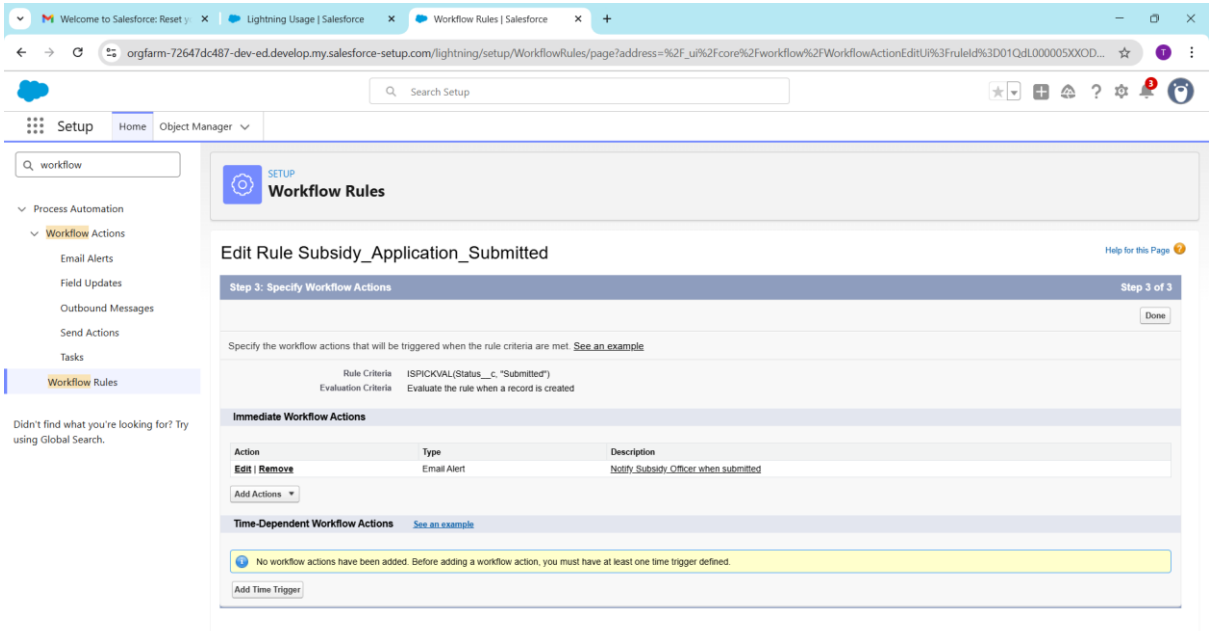


◆ 4. Workflow Rule (Optional)

Object: Order__c
Rule Criteria:
When Status__c = "Completed"

Actions:

- Send email to Farmer confirming order completion.
- Update Buyer's "Last Order Date."



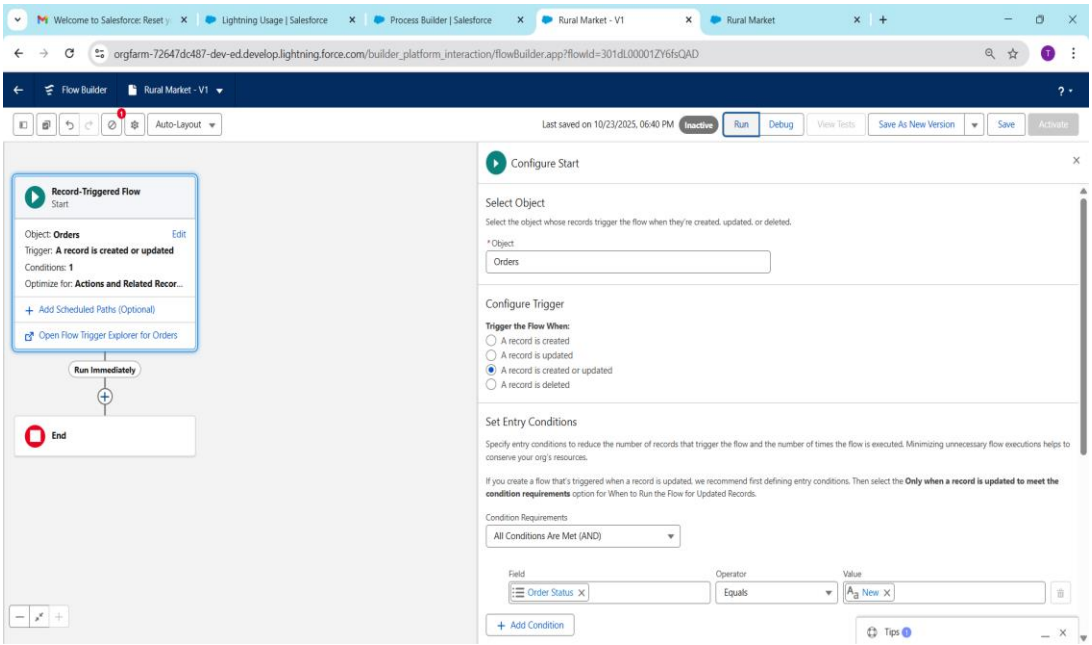
◆ 5. Process Builder

Object: Order__c

Trigger: Record Created or Updated
Criteria: Status__c = 'Confirmed'

Actions:

1. Update Buyer__c.Last_Order_Date__c = TODAY()
2. Send email to Buyer
3. Post to Chatter: “New order confirmed for {Crop_

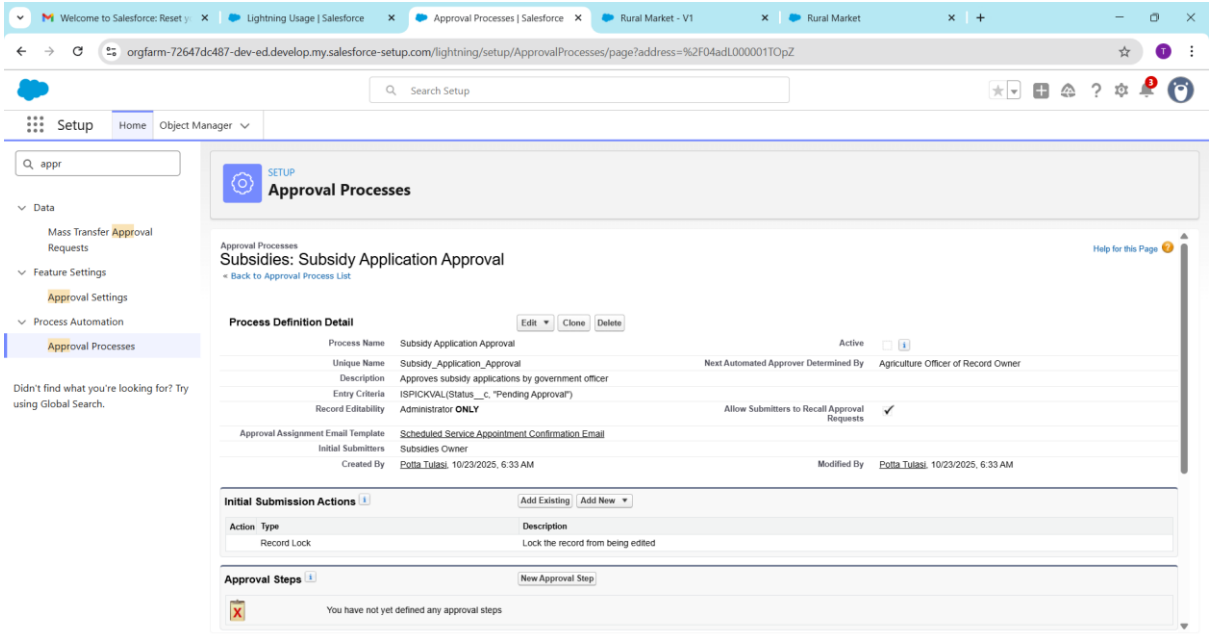


◆ 6. Approval Process

Object: Subsidy_Application__c
Entry Criteria: Amount__c > 5000

Steps:

1. Manager approval required
2. On approval → Status = "Approved"
3. On rejection → Status = "Rejected"

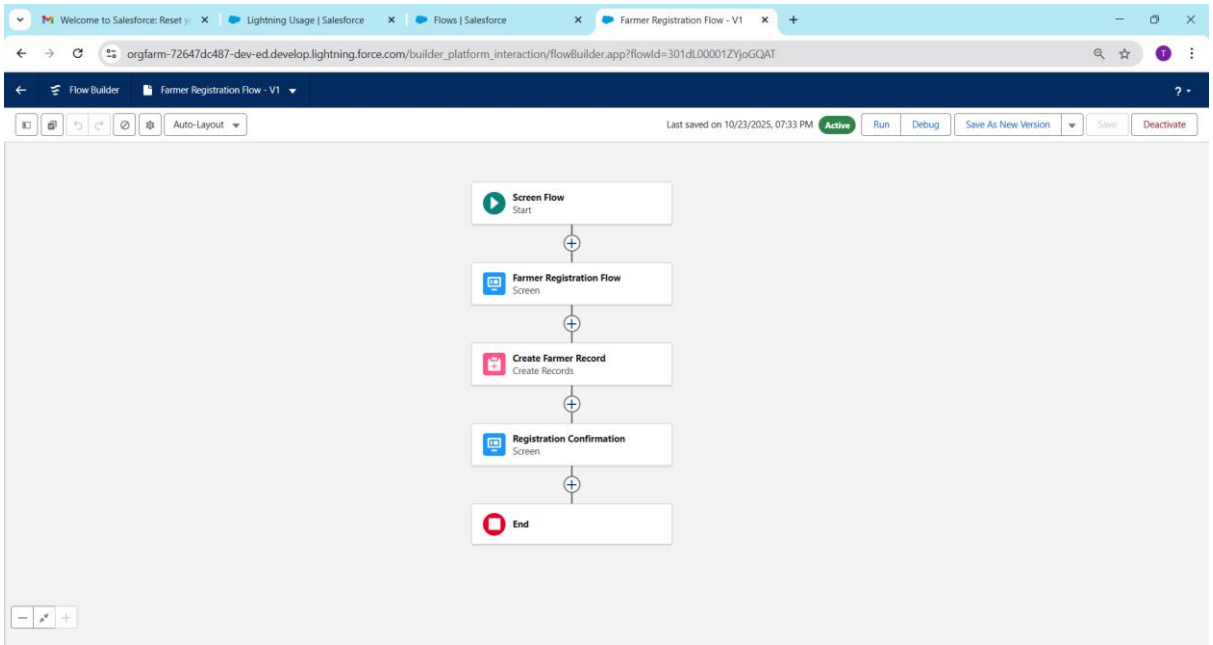


◆ 7. Flow Builder Automations

a. Screen Flow – Farmer Registration

Purpose: Allow farmers to register through the portal.
Steps:

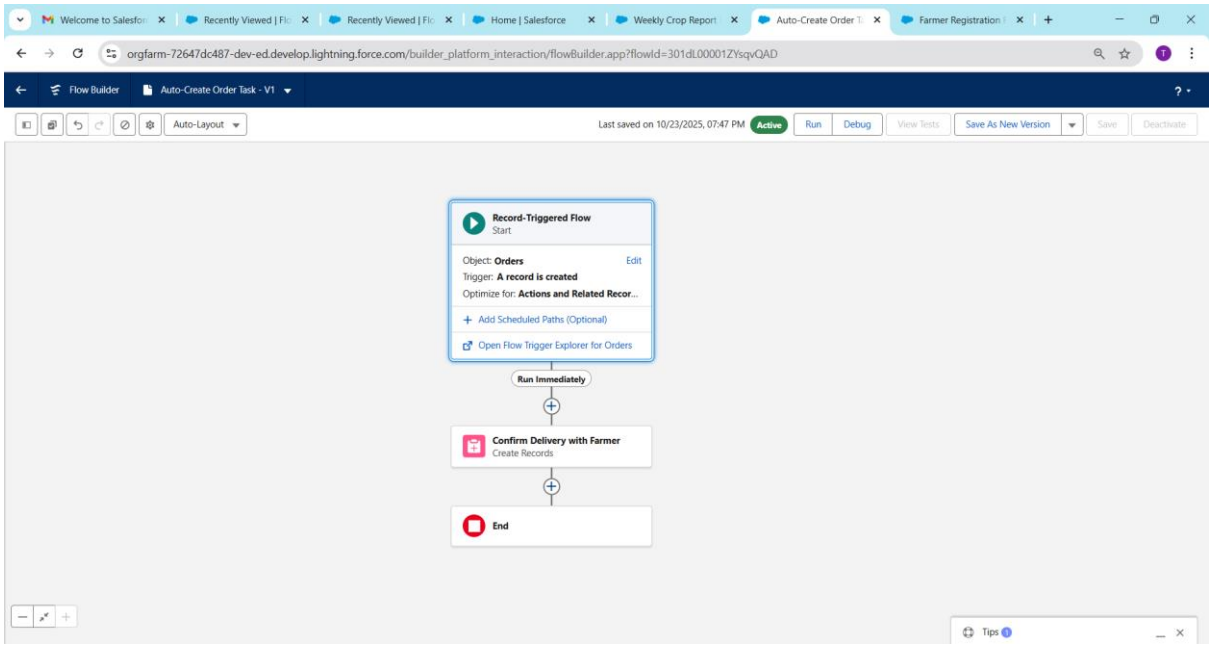
1. Screen → Enter Name, Phone, Crop
2. Create Record → Farmer__c
3. Display "Registration Successful" message



b. Record-Triggered Flow – Order Automation

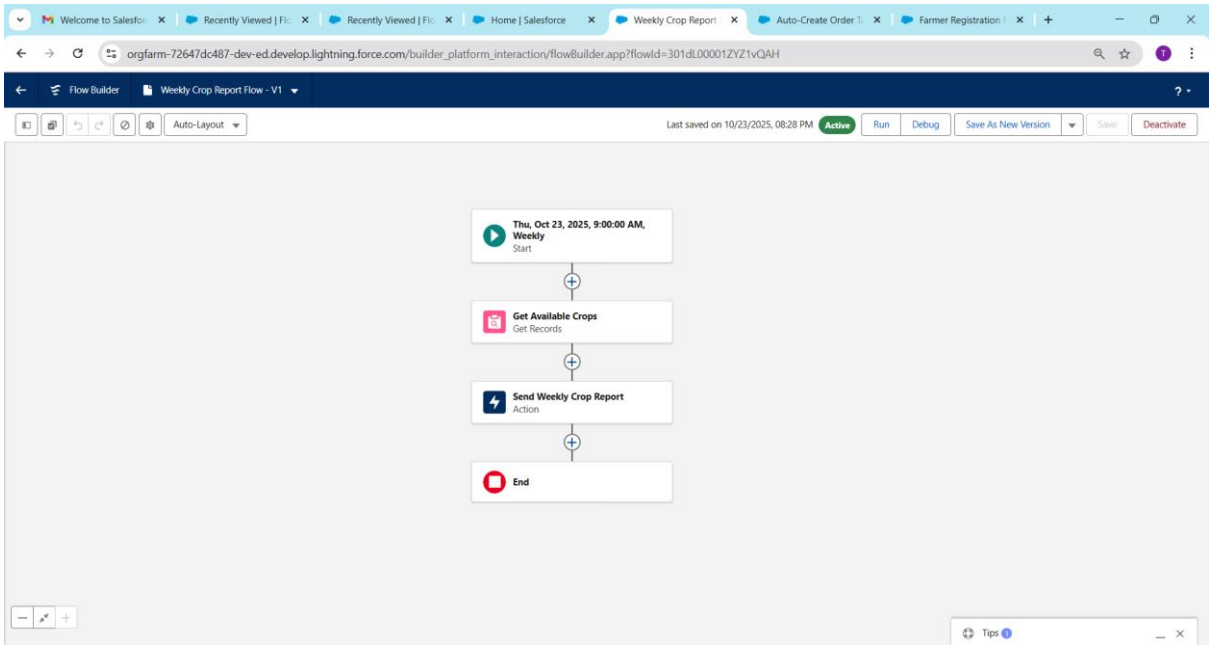
Trigger: When Order is created.
Actions:

- Update Buyer's Last Order Date
- Send Order Confirmation Email



c. Scheduled Flow – Monthly Reminder

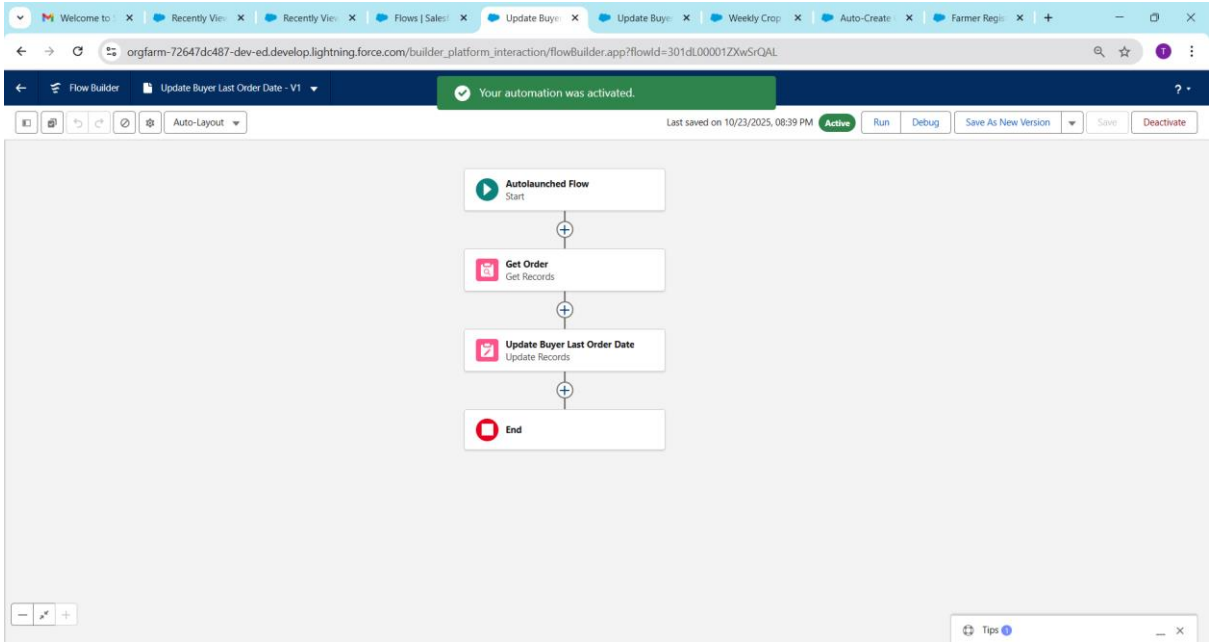
Schedule: 1st of every month
Action: Send reminder email to inactive Buyers.



d. Auto-Launched Flow – Internal Updates

Purpose: Update internal metrics (e.g., total orders by Farmer).

Trigger: Called by another Flow.

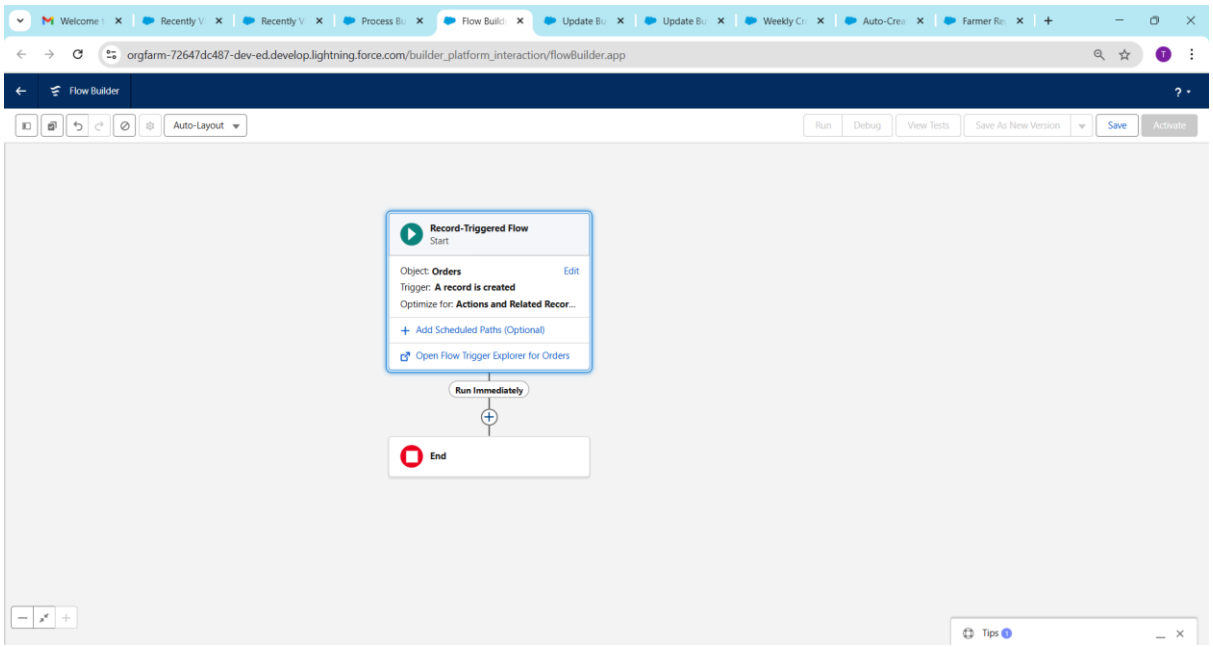


◆ 8. Email Alerts

Object: Order__c
Action: Send "Order Confirmation" Email to Buyer.

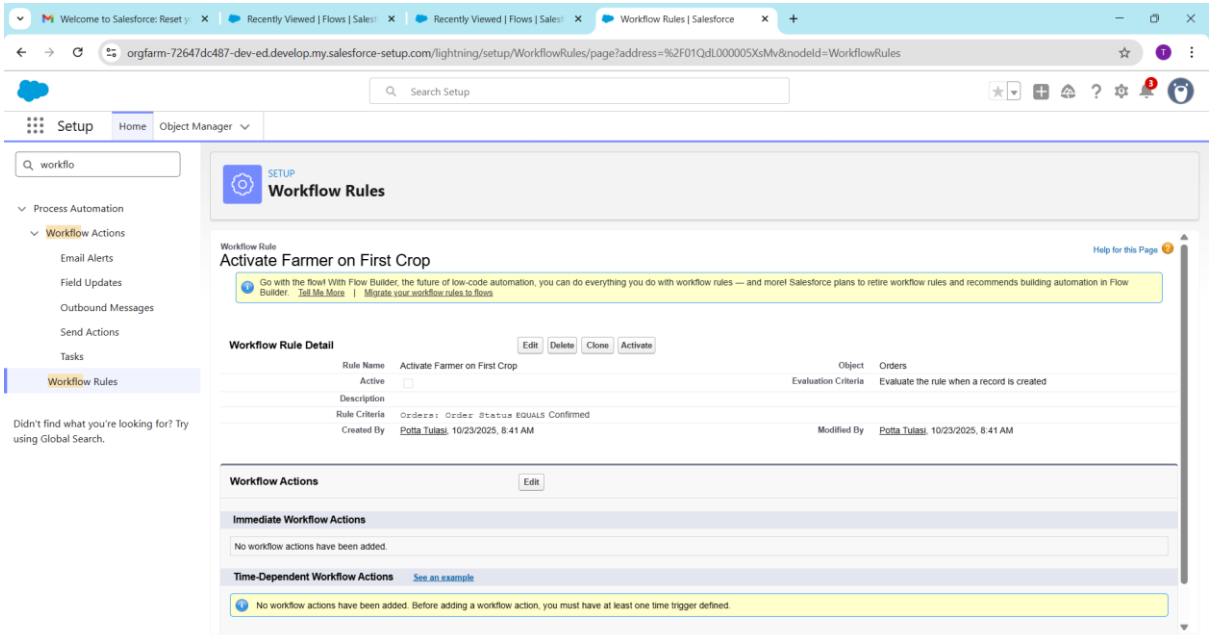
Setup Steps:

1. Create Email Template → "Order Confirmation"
2. Create Email Alert → Link Template
3. Trigger via Flow / Process Builder



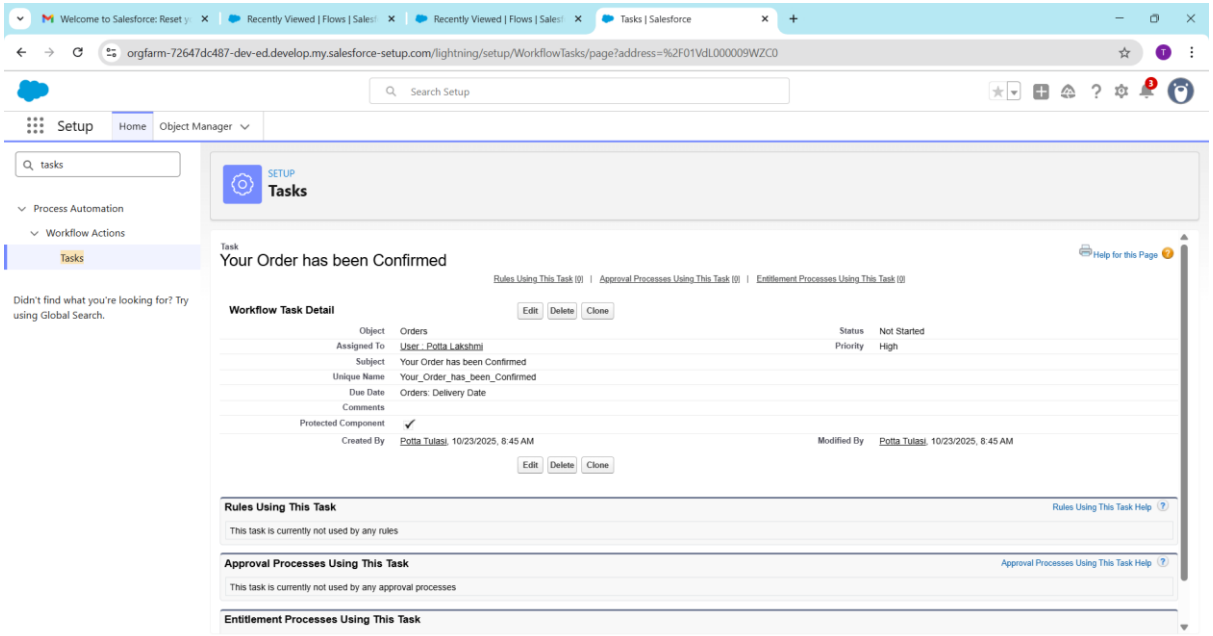
◆ 9. Field Updates

Rule:
If Order__c.Status__c = 'Delivered' →
Update Farmer__c.Last_Order_Status__c = 'Delivered'



◆ 10. Tasks

Automation:
When a Subsidy Application is approved →
Create Task for Finance Officer: *"Release Payment*



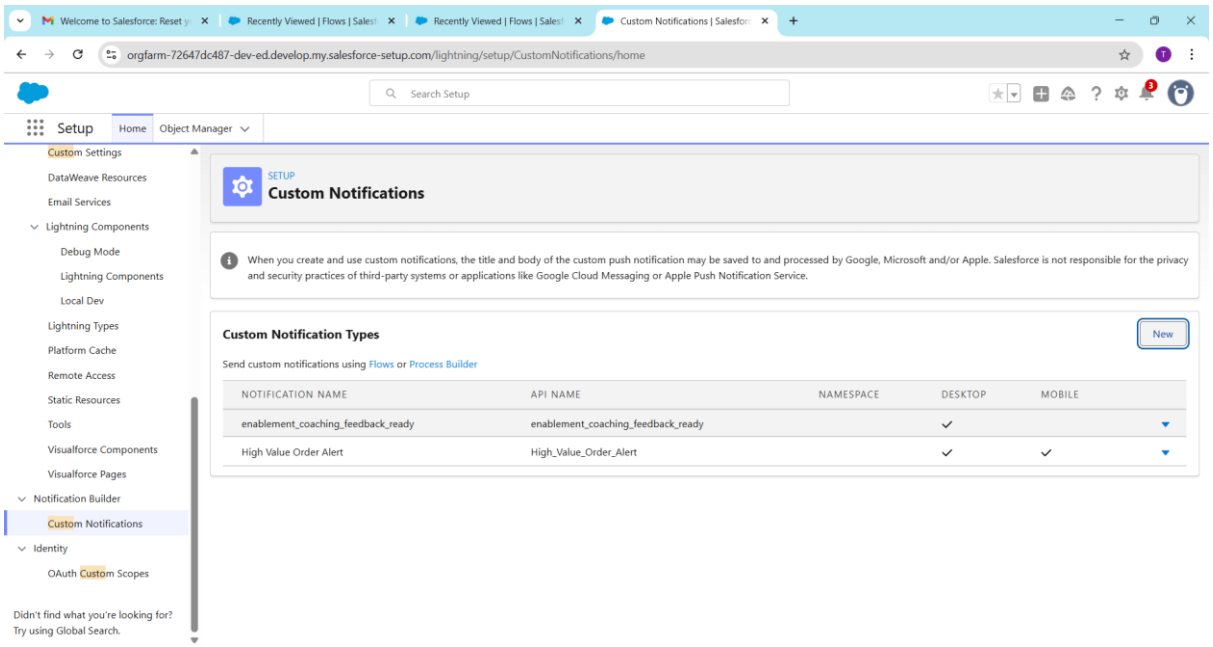
◆ 11. Custom Notifications

Scenario:
When a new order is placed → Send in-app alert to Farmer:

"New order received for your crop!"

Setup:

1. Setup → Notification Builder → Custom Notification
2. Trigger via Flow



◆ 12. Final Automation Workflow (Summary)

Process Flow Diagram:

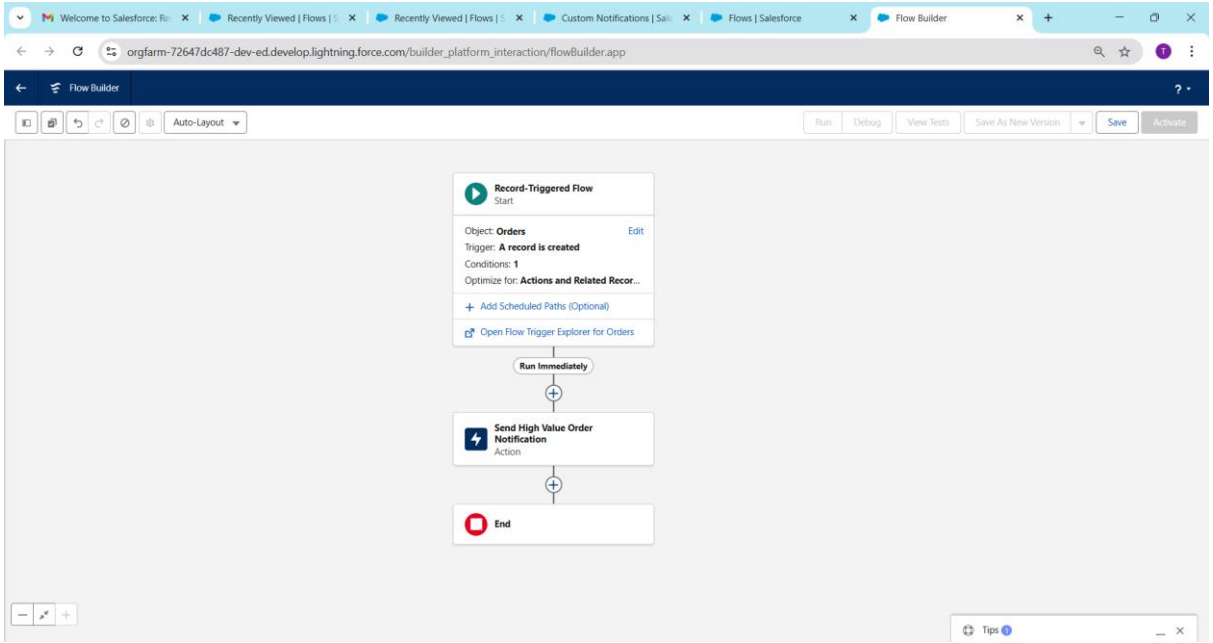
Order Created

↓

Record-Triggered Flow

↓

Update Buyer → Send Email → Notify Farmer → Update Records



◆ 13. Testing & Verification

Test Case	Trigger	Expected Outcome
New Farmer Registered	Screen Flow	Farmer record created
Order Confirmed	Process Builder	Email sent, field updated
Subsidy Request > ₹5000	Approval Process	Manager approval required
Monthly Reminder	Scheduled Flow	Email sent to inactive buyers

◆ 14. Output / Result

✅ Automated all key business processes for Farmers, Buyers, and Government Subsidy Approvals.
✅ Reduced manual work and ensured data accuracy.
✅ Enhanced transparency in the Rural Market Linkage ecosystem.

◆ 15. Conclusion

This phase integrates automation tools to streamline data validation, communication, and approvals — making the CRM more intelligent, efficient, and farmer-friendly.

PHASE 3: DATA MODELING & RELATIONSHIPS

1⃣ Standard & Custom Objects

Standard Objects

These are Salesforce's built-in objects.
Examples:

- **Account** → Can represent "Cooperatives" or "Organizations."
- **Contact** → Can represent "Farmer Contacts."
- **Opportunity** → Can represent "Orders."
- **Case** → Can represent "Complaints or Service Requests."

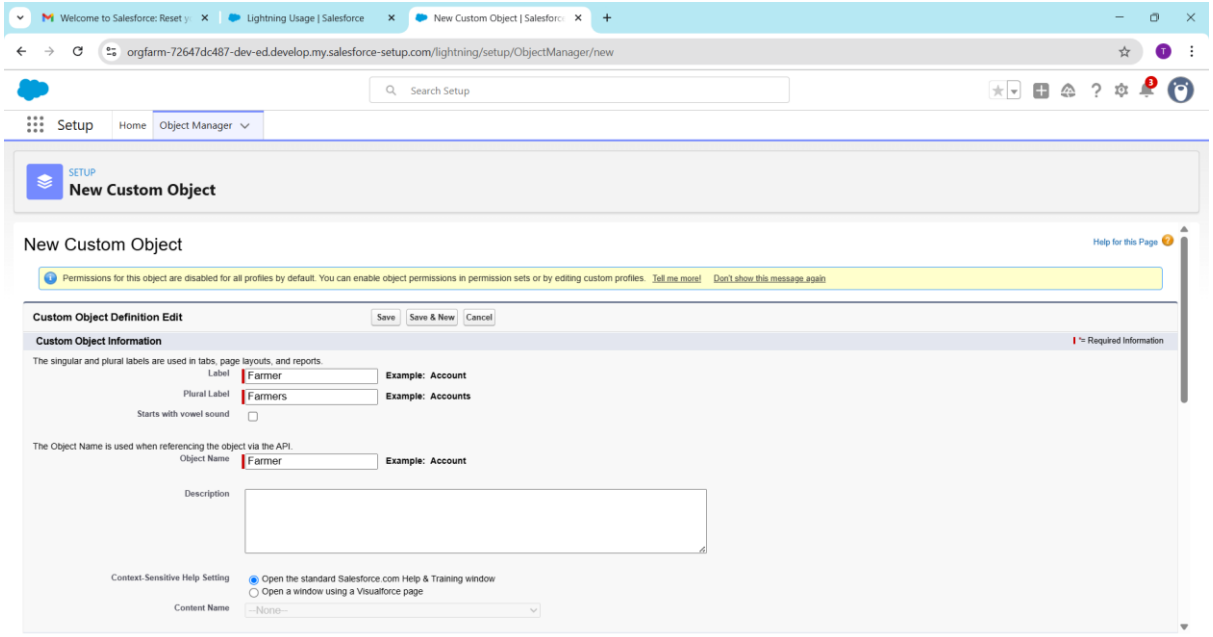
Custom Objects

You create these for your project-specific needs.
Path:
👉 Setup → Object Manager → **Create** → **Custom Object**

Object Name	API Name	Purpose
Farmer	Farmer__c	Stores farmer details
Buyer	Buyer__c	Stores buyer details
Crop	Crop__c	Details of crops
Order	Order__c	Links farmers and buyers
Subsidy Application	Subsidy_Application__c	Government subsidy tracking

✅ After creating each, check:

- Allow Reports
- Allow Activities
- Track Field History (important for auditing)

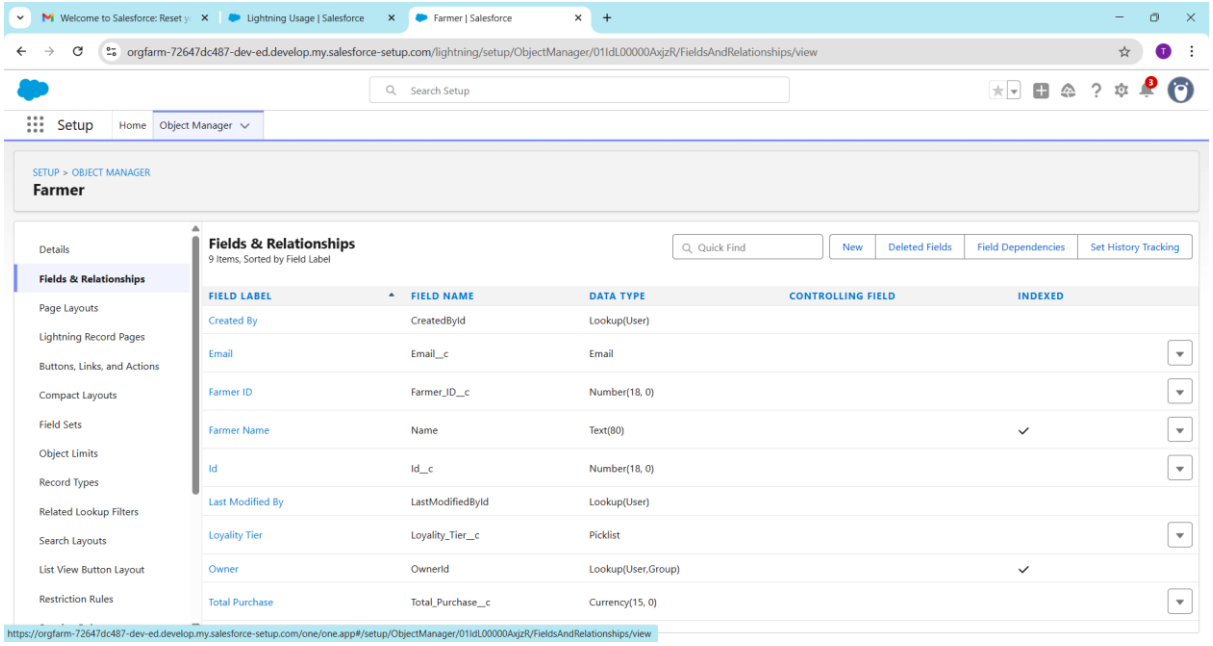


2️⃣ Fields

Each object has **fields** that store data.
Path: Object Manager → [Object Name] → Fields & Relationships → New

Field Label	Data Type	Example / Notes
Farmer Name	Text	Full name
Mobile Number	Phone	Farmer's contact
Crop Type	Picklist	Rice, Wheat, Maize
Quantity	Number	Kg or Quintals
Price	Currency	Selling price
Order Date	Date	Date of sale
Cooperative	Lookup(Cooperative)	Connects farmer with cooperative

✅ Use **Formula Fields** for calculations
Example: Total_Amount__c = Quantity__c * Price__c



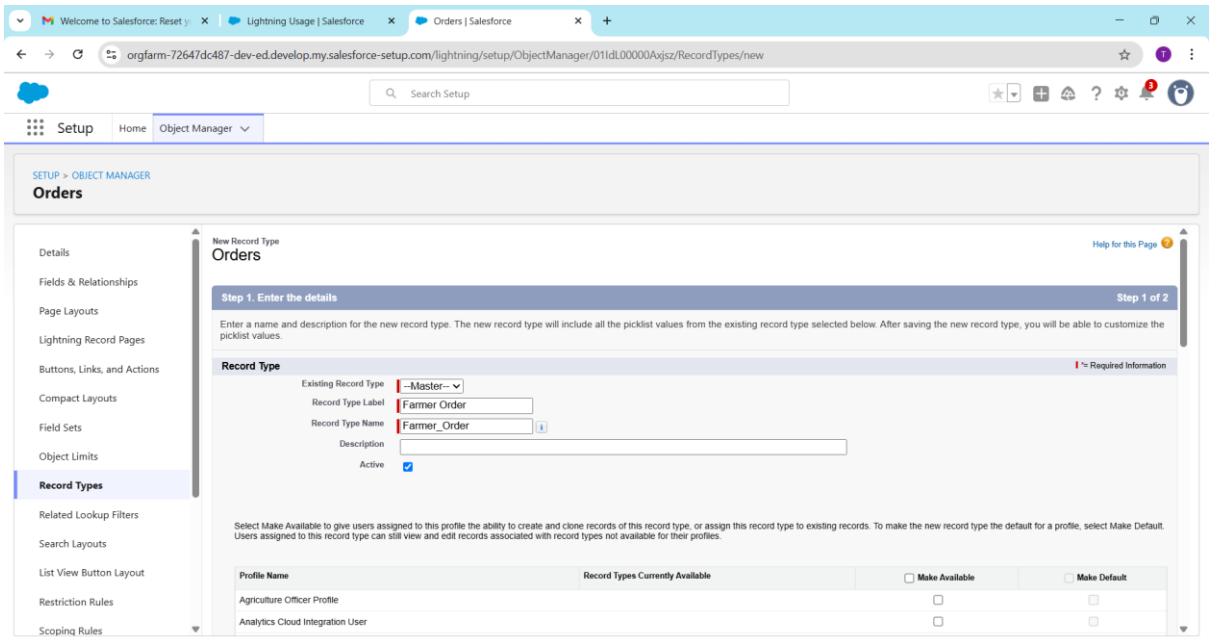
3️⃣ Record Types

Used when you need **different views or processes** for the same object.
Example:

- **Order__c**
  - Record Type 1: *Farmer Order*
  - Record Type 2: *Buyer Purchase*

Path: Object Manager → [Object] → Record Types → New

✅ Assign to profiles as needed.



4️⃣ Page Layouts

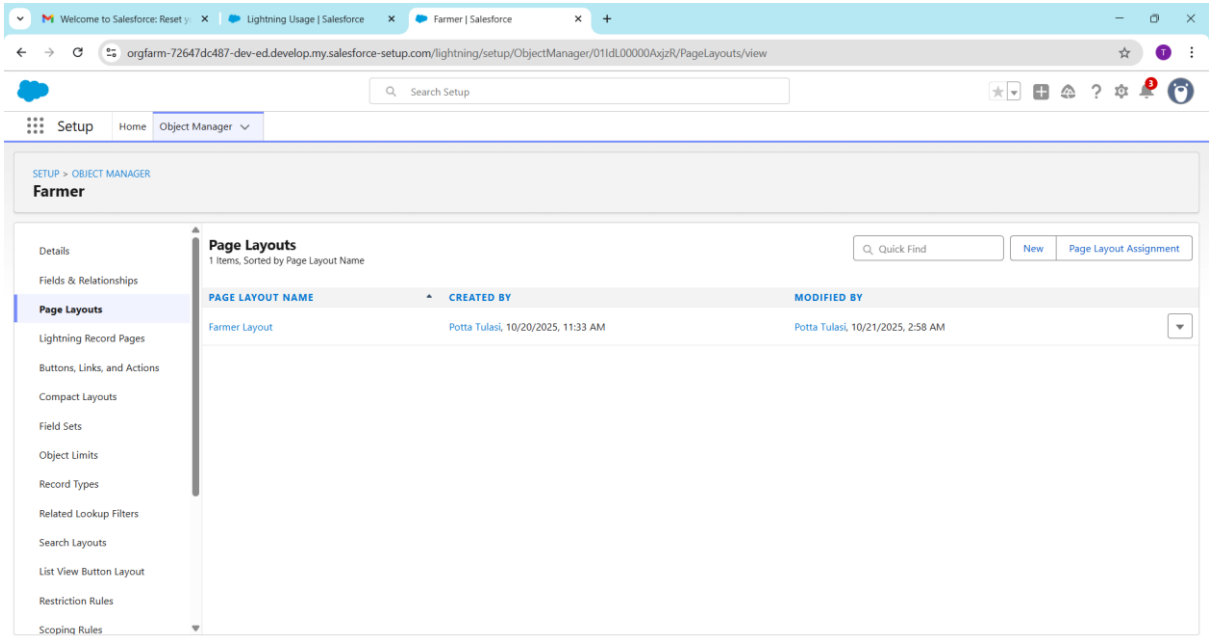
Controls what users **see and edit** on a record page.
Example:

- Farmer Page Layout → shows personal, crop, and cooperative fields
- Buyer Page Layout → shows order and payment details

Path: Object Manager → [Object] → Page Layouts → Edit

✅ You can:

- Drag & drop fields
- Add related lists (Orders, Crops)
- Assign layouts by profile

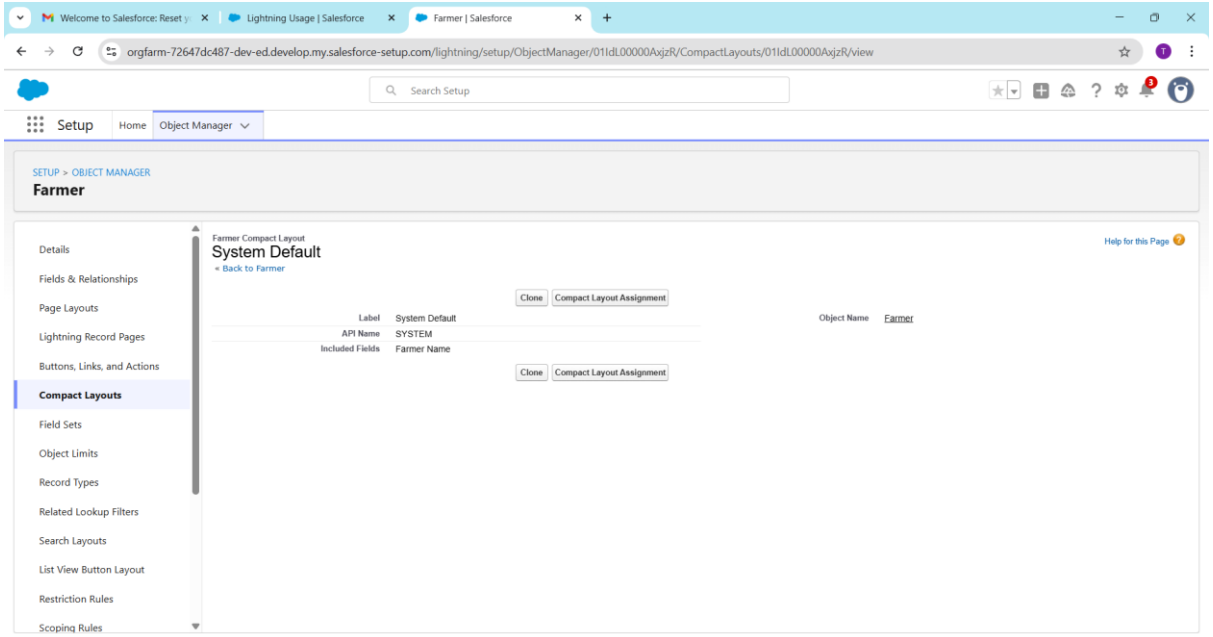


5️⃣ Compact Layouts

Controls which **fields show in highlights panel** (top area of record page).
Example:
For **Farmer__c**, include:

- Farmer Name
- Mobile Number
- Cooperative

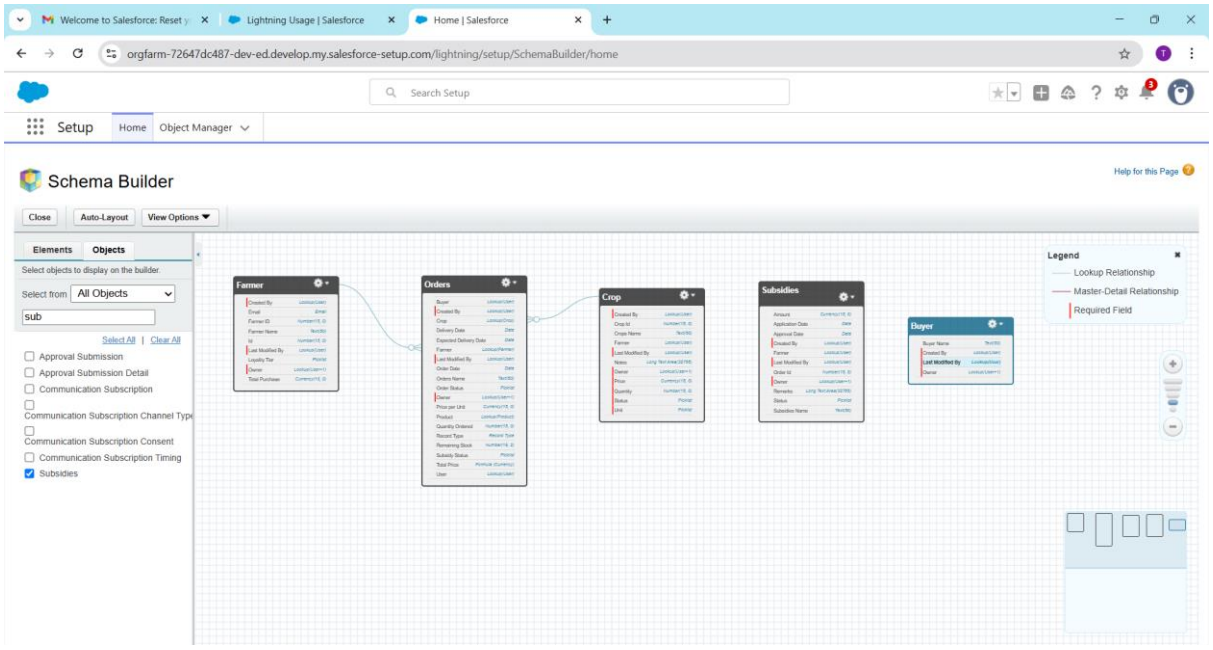
Path: Object Manager → [Object] → Compact Layouts → New → Assign as Primary



6. Schema Builder

Visual tool to see **objects + fields + relationships** in one diagram.
Path: Setup → Schema Builder

✅ Select your custom objects (Farmer, Buyer, Crop, Order)
You can drag to create relationships directly here.

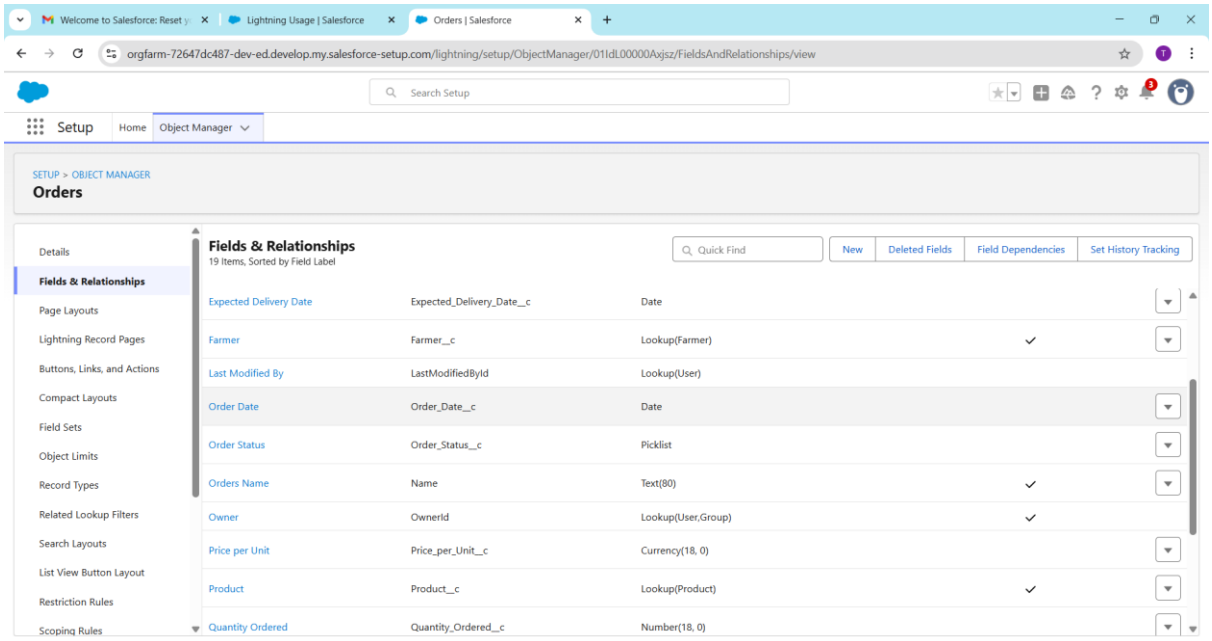


7️⃣ Relationships

Relationships connect data between objects.

Type	Description	Example
Lookup	Loose link between two objects	Order → Farmer
Master-Detail	Strong link (child depends on parent)	Crop → Cooperative
Hierarchical	Only for User object (managers)	User → Manager

How to Create:
👉 Object Manager → [Child Object] → Fields & Relationships → New → Relationship Type



8️⃣ Junction Objects (Many-to-Many)

Used when **one record relates to many records** on both sides.

Example:

- A **Farmer** can grow many **Crops**
- A **Crop** can be grown by many **Farmers**

Create **Junction Object:**
→ Name: *FarmerCrop__c*
→ Add two Master-Detail fields:

- Farmer (to Farmer__c)
- Crop (to Crop__c)

This forms **Many-to-Many Relationship** ✅

2️⃣ External Objects (for External Data)

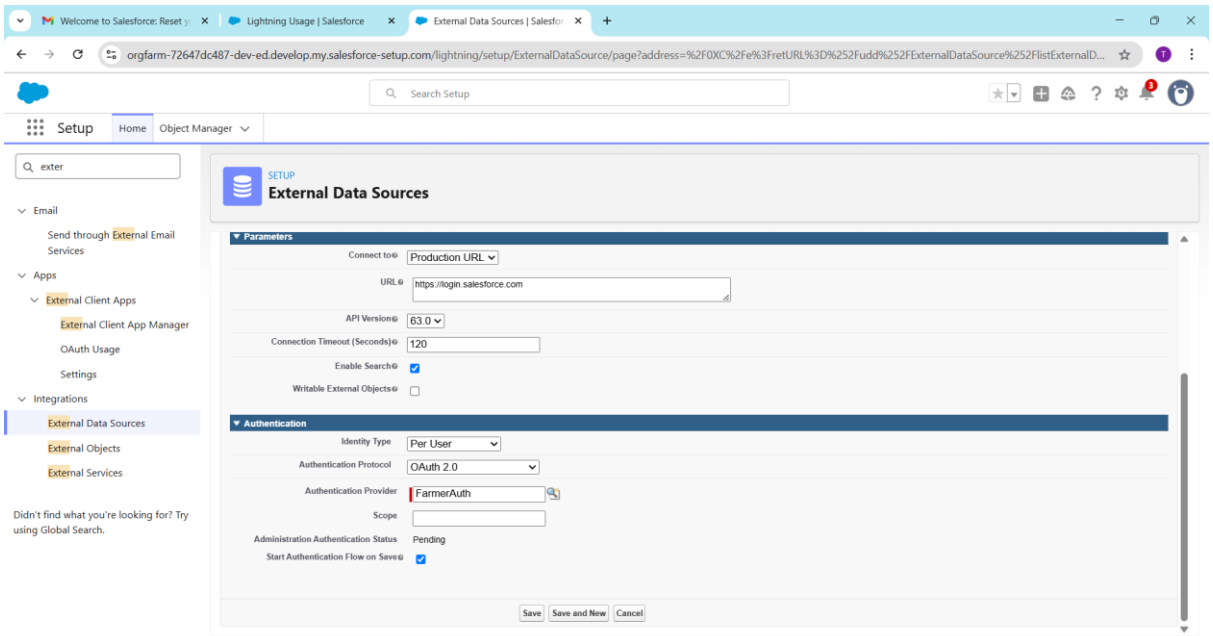
Used when data is **stored outside Salesforce** (e.g., government subsidy API or external database).

- Use **Salesforce Connect** to link external data.
- Appears like a normal object but data is fetched real-time.

Path: Setup → External Data Source → New → Define connection → Validate & Sync

Example:

- **External Object Name:** Government_Subsidy__x
  → Fields: Farmer_ID, Status, Amount

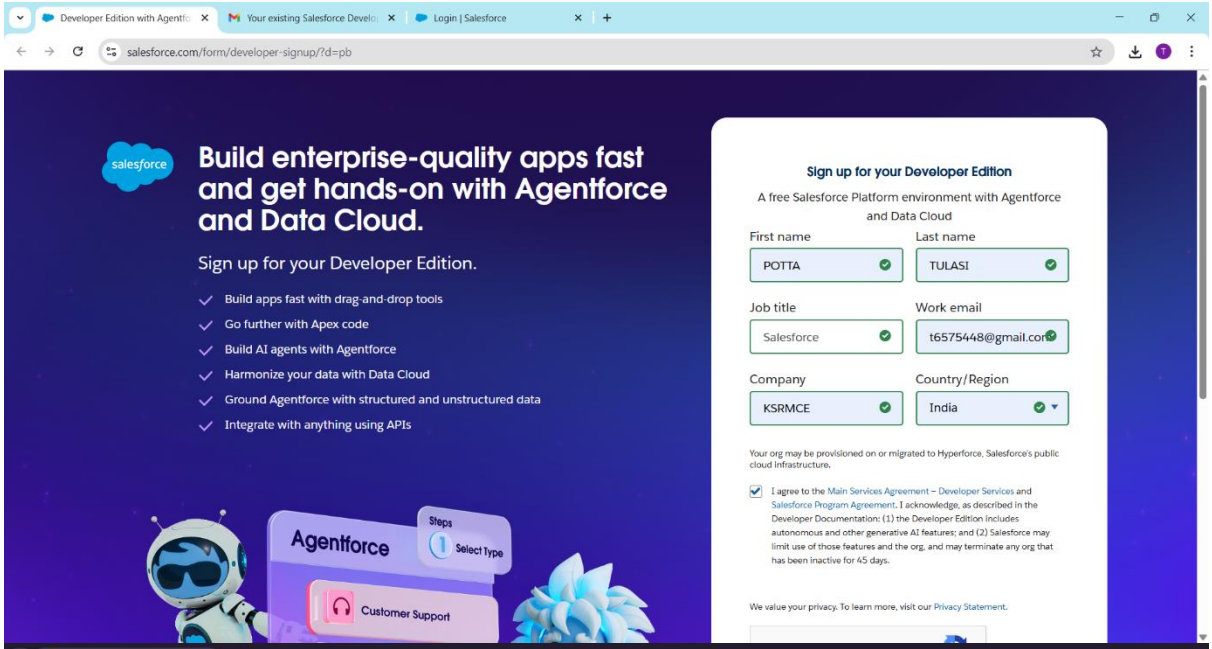


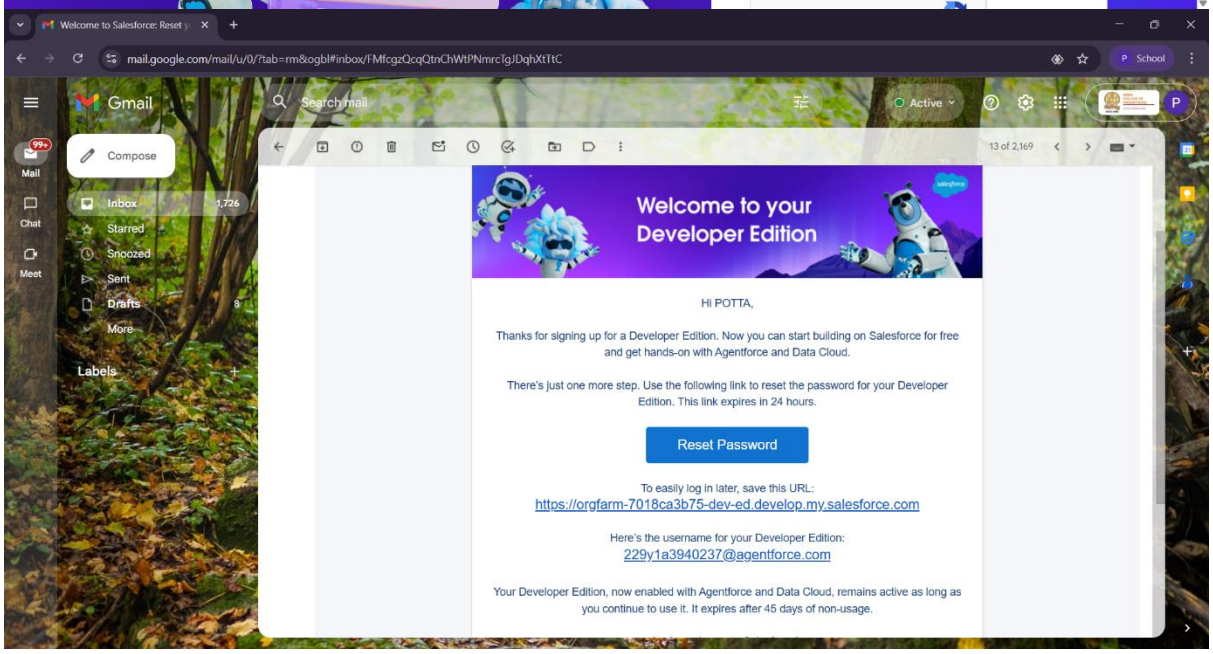
Phase 2: Org Setup & Configuration

1.Salesforce Edition

Choose the **Salesforce Developer Edition** for your project.

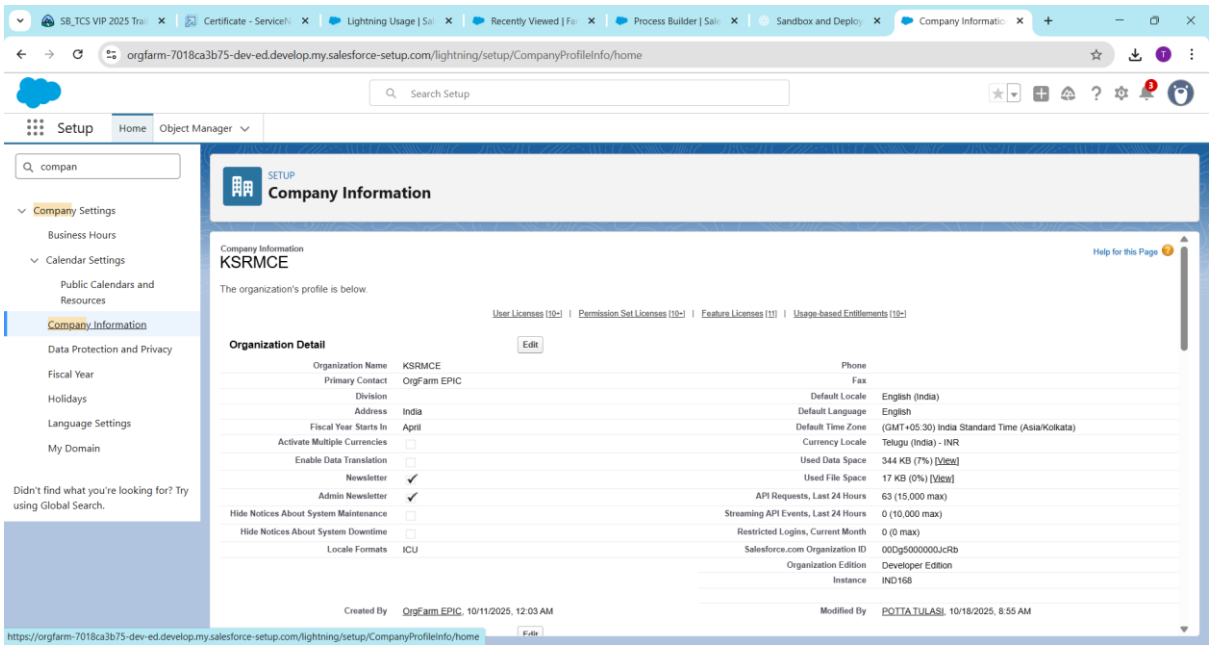
- Provides all tools needed for practice, development, and testing.
- Free to use, supports all key features, and is ideal for student projects.





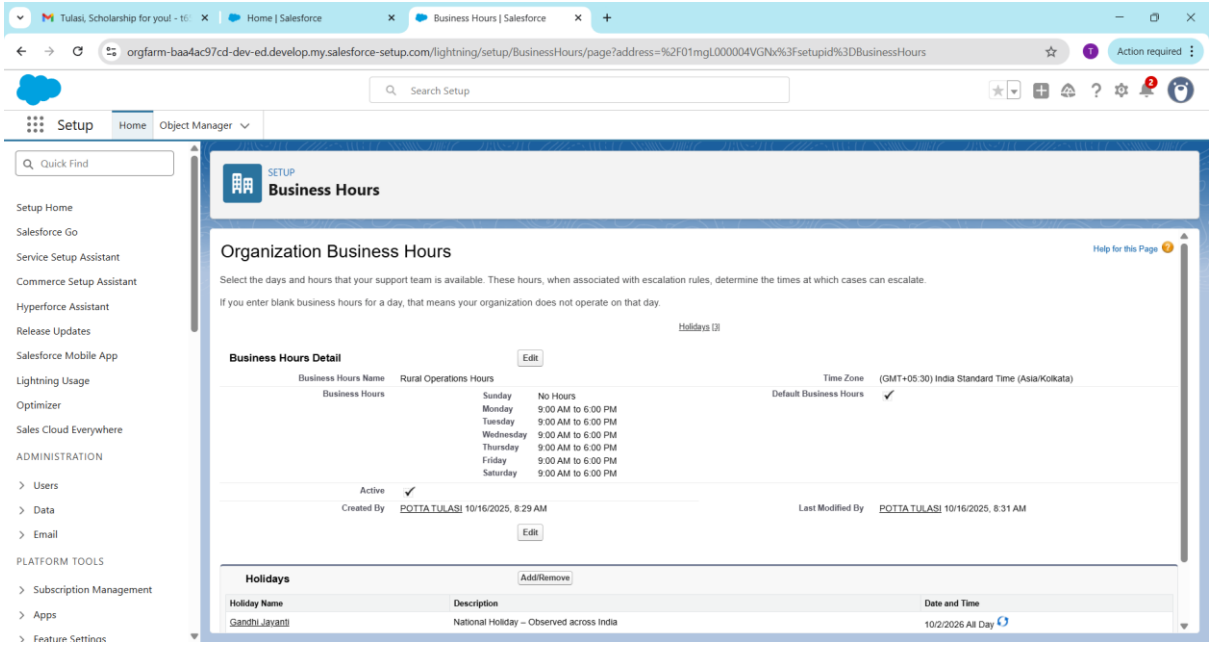
2. Company Profile Setup

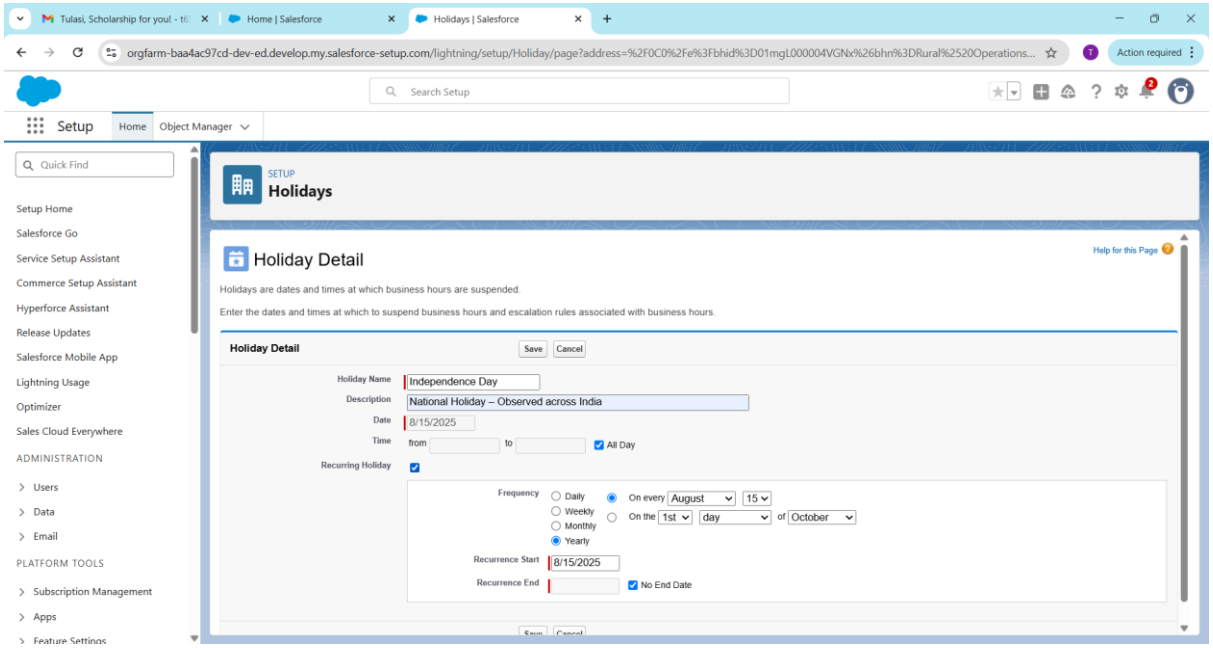
- Navigate to **Setup → Company Information**.
- Update your organization details:
  - Organization Name
  - Time Zone: **GMT +05:30**
  - Currency: **INR**
  - Default Language: **English (India)**
- Confirm the main contact details.



3. Business Hours & Holidays

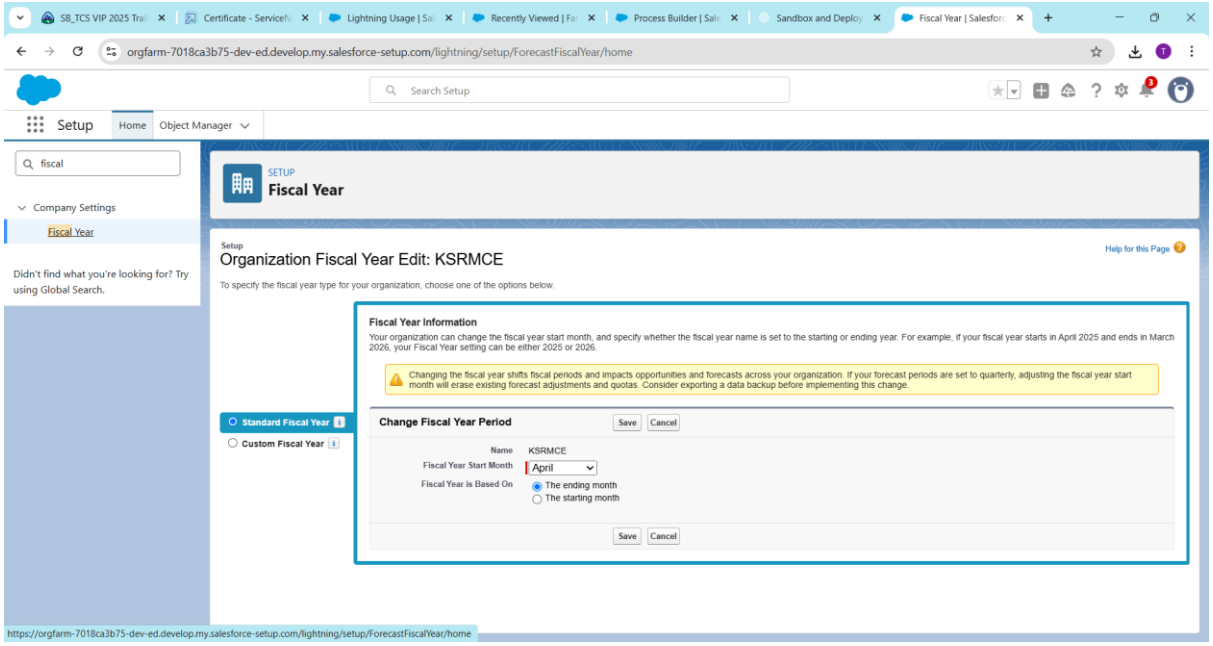
- **Business Hours:** Setup → Business Hours → New
  - Working Days: Monday to Saturday
  - Working Hours: 9:00 AM – 6:00 PM
- **Holidays:** Setup → Holidays → New
  - Add important holidays like Independence Day and Diwali.
  - Ensures workflows and escalation rules respect non-working days.





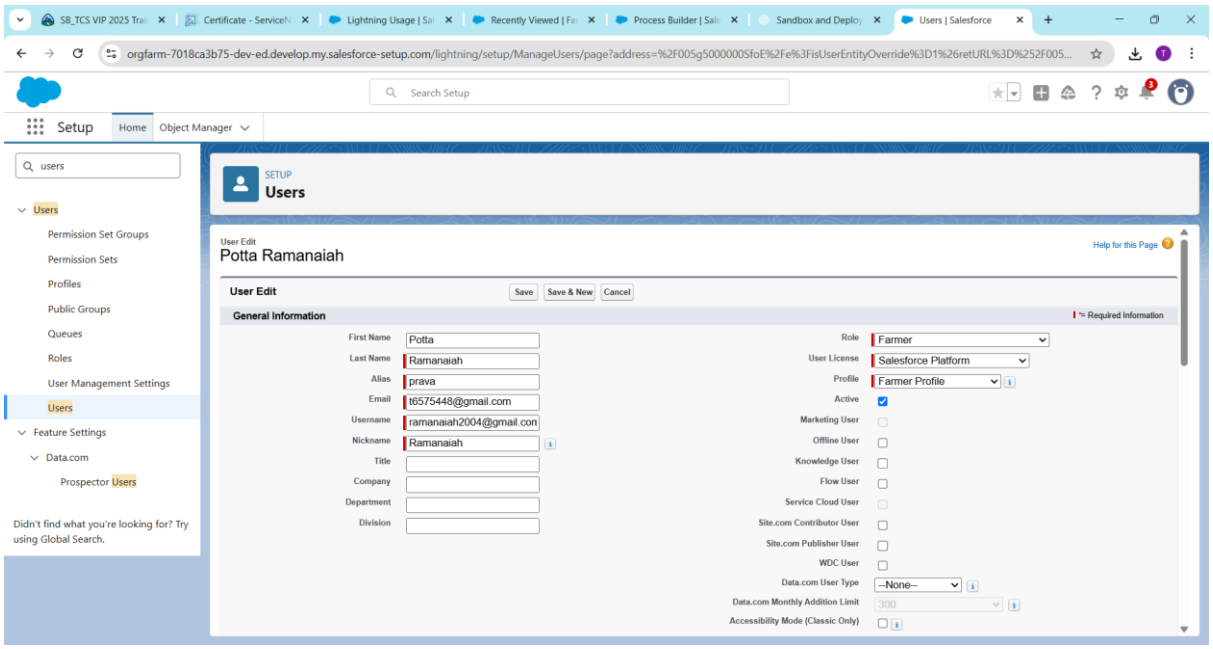
4. Fiscal Year Settings

- Navigate to **Setup → Fiscal Year**
- Select **Standard Fiscal Year**
- Set **Start Month:** April, **End Month:** March
- Matches the Indian government's financial reporting cycle.



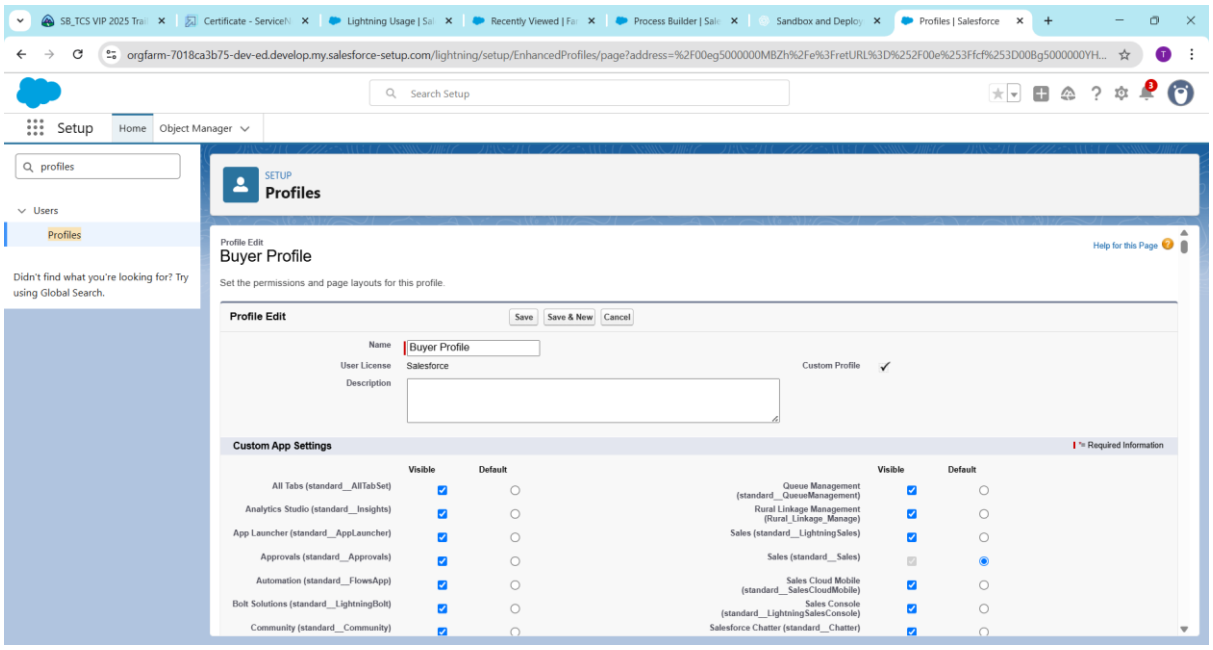
5. User Setup & Licenses

- Go to **Setup → Users → New User**
- Create accounts for:
  - **System Admin**
  - **Government Officer**
  - **Buyer**
  - **Farmer**
- Assign appropriate **Salesforce licenses** and **roles** to each user.



6. Profiles

- Navigate to **Setup → Profiles**
- Clone the **Standard User** profile and rename it.
- Create profiles such as:
  - Farmer Profile
  - Buyer Profile
  - Officer Profile
- Assign **object-level** and **field-level permissions** according to user responsibilities.



7. Roles

- Go to **Setup → Roles → Set Up Roles**
- Define a data visibility hierarchy:

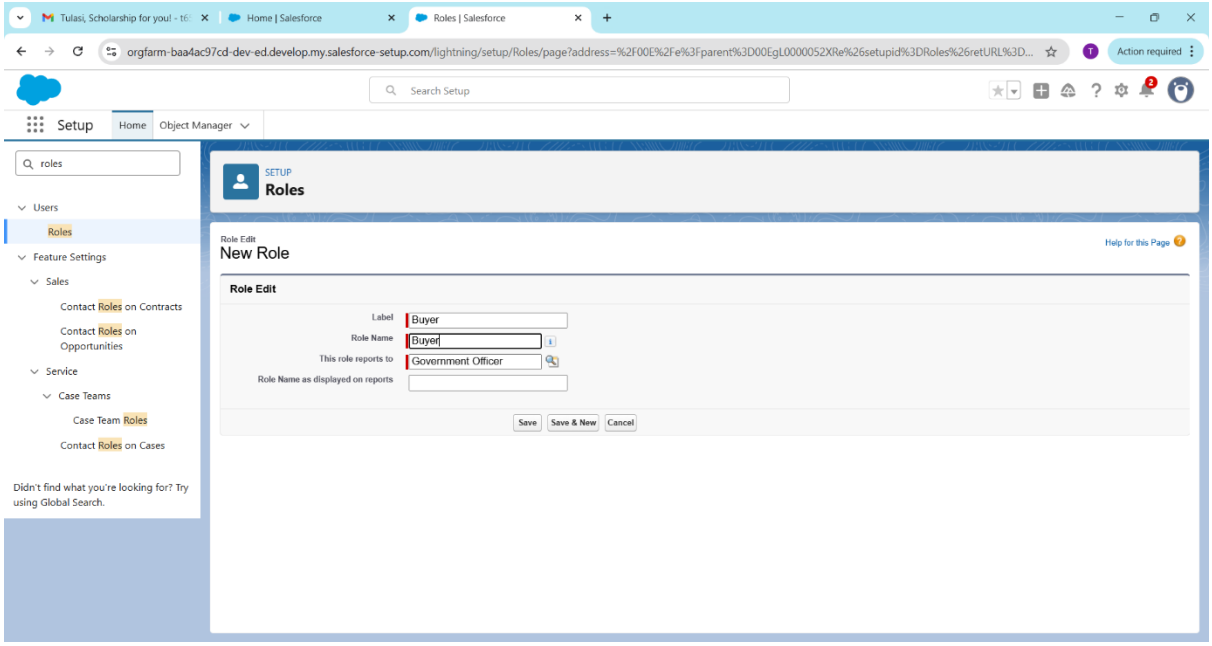
System Admin

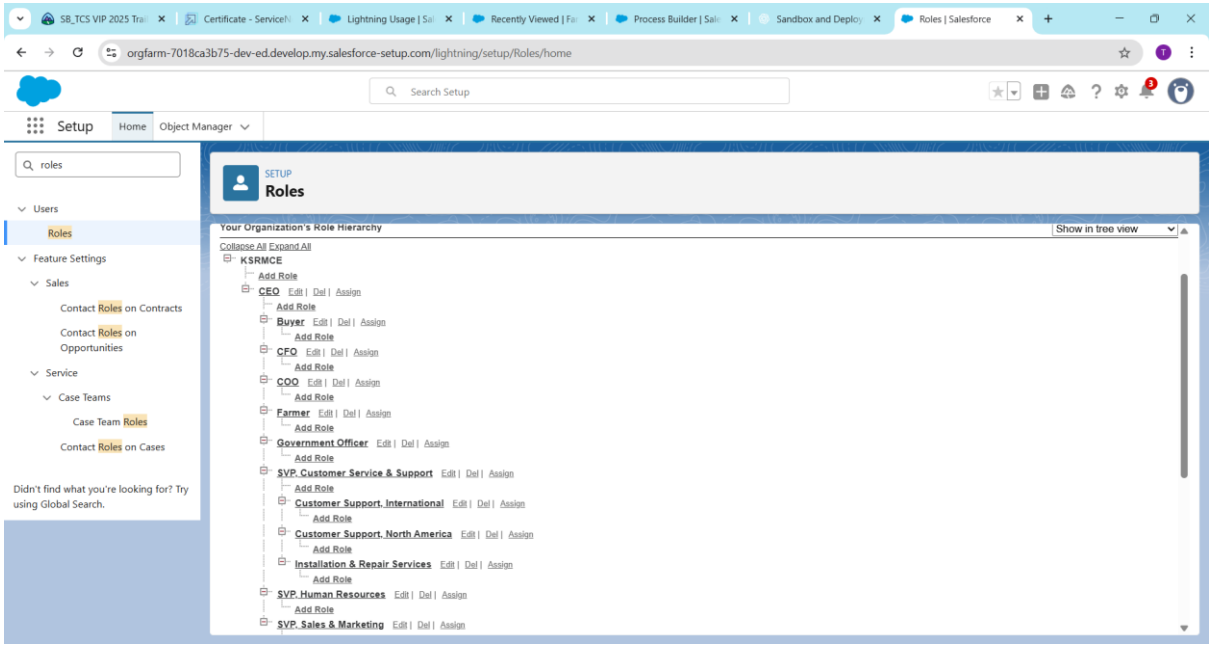
→ Government Officer

→ Buyer

→ Farmer

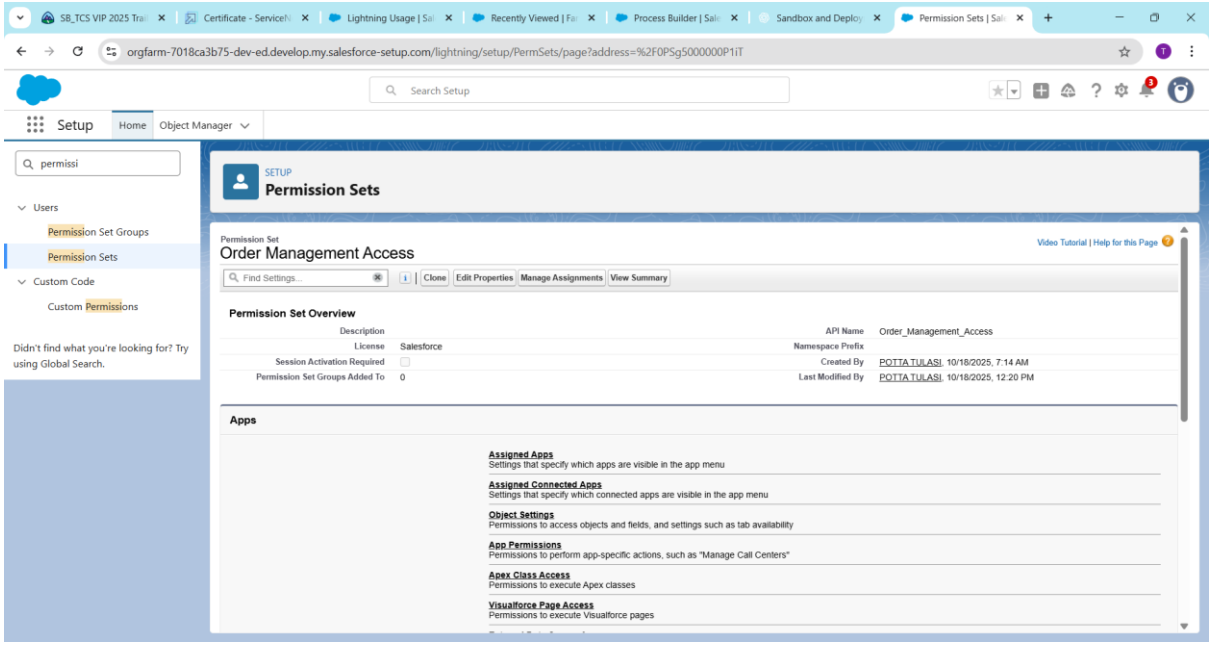
- Ensures data flows securely upward through the organization.





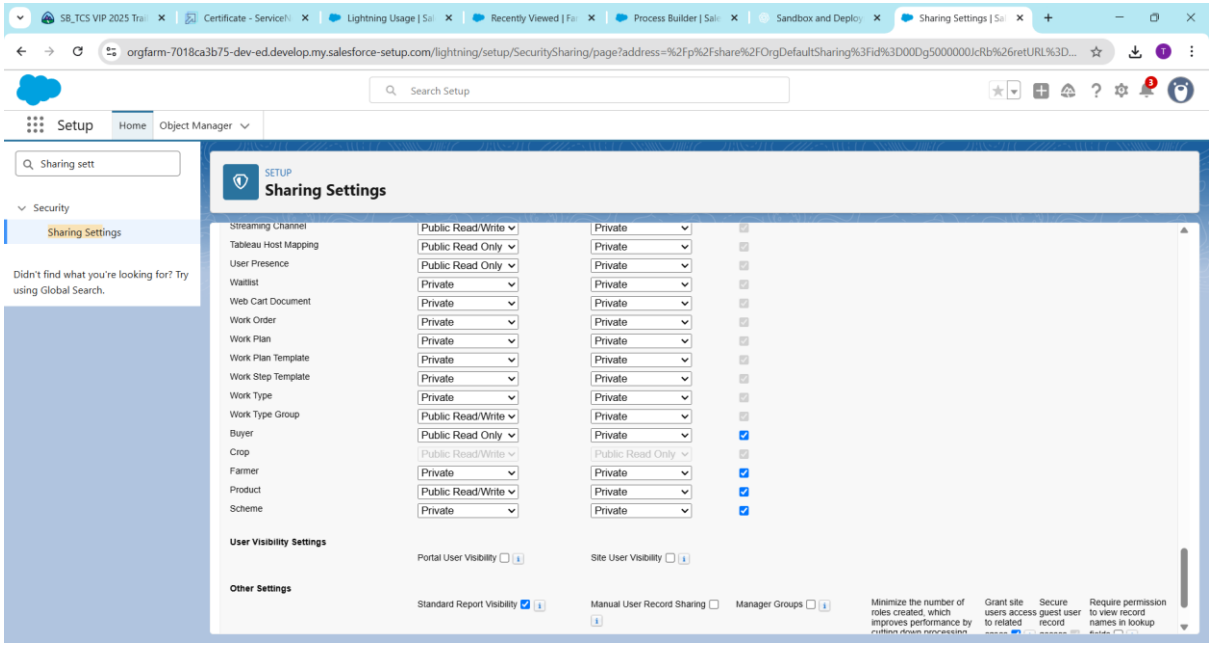
8. Permission Sets

- Navigate to **Setup → Permission Sets → New**
- Create additional privileges such as:
  - Opportunity Access
  - Data Export Access
- Assign permission sets to users who require extra access without changing their profiles.



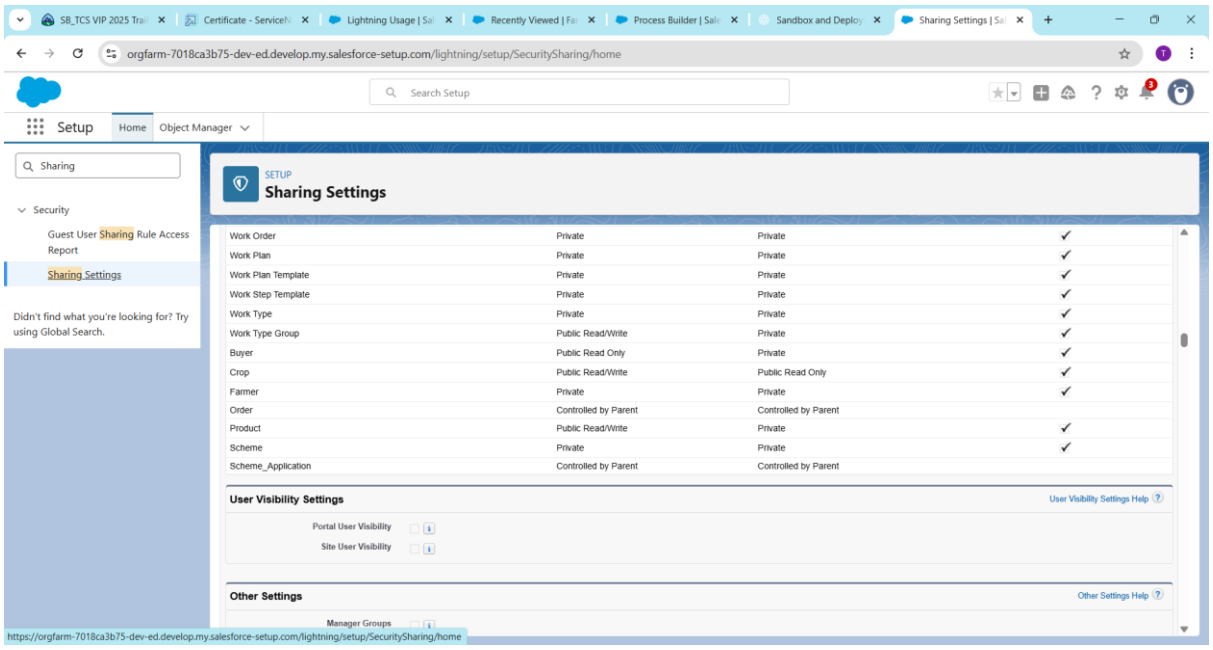
9. Organization-Wide Defaults (OWD)

- Go to **Setup → Sharing Settings → Edit**
- Configure default record access:
  - Farmer: **Private**
  - Buyer: **Private**
  - Product: **Public Read-Only**
  - Market Transaction: **Controlled by Parent**
- Ensures data security and proper sharing control.



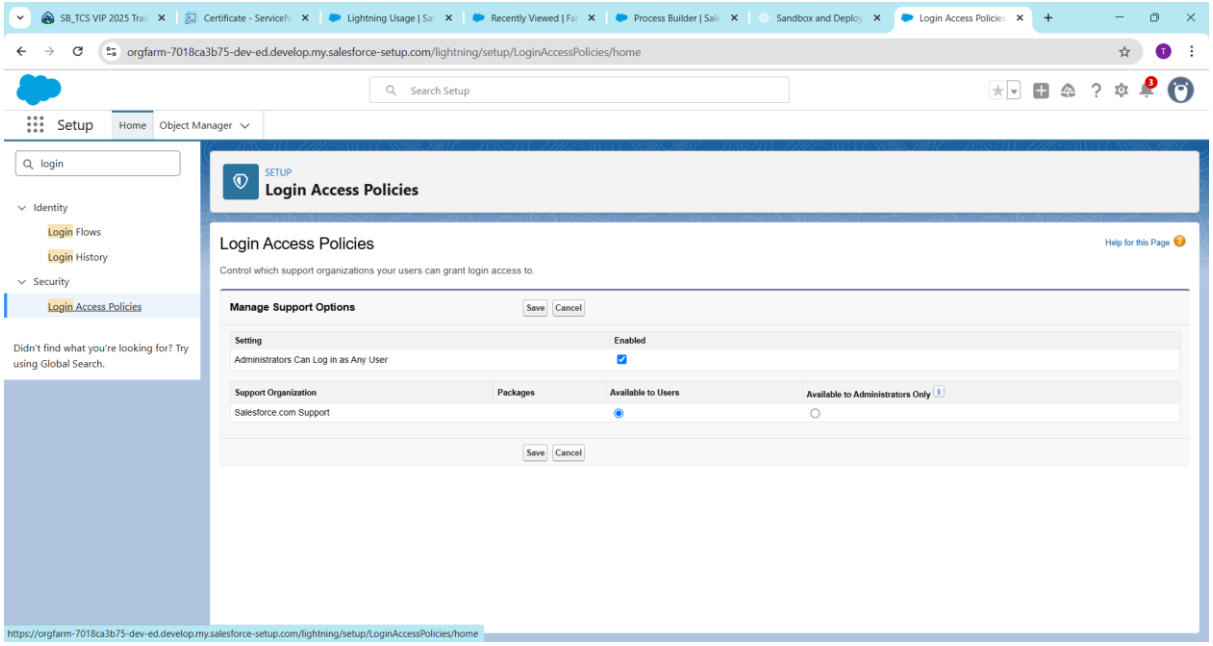
10. Sharing Rules

- Navigate to **Setup → Sharing Settings → New Rule**
- Create sharing rules for collaboration:
  - Share **Farmer** records with Buyers (Read Only)
  - Share **Opportunities** with Government Officers (Read/Write)
- Allows teamwork without compromising data security.



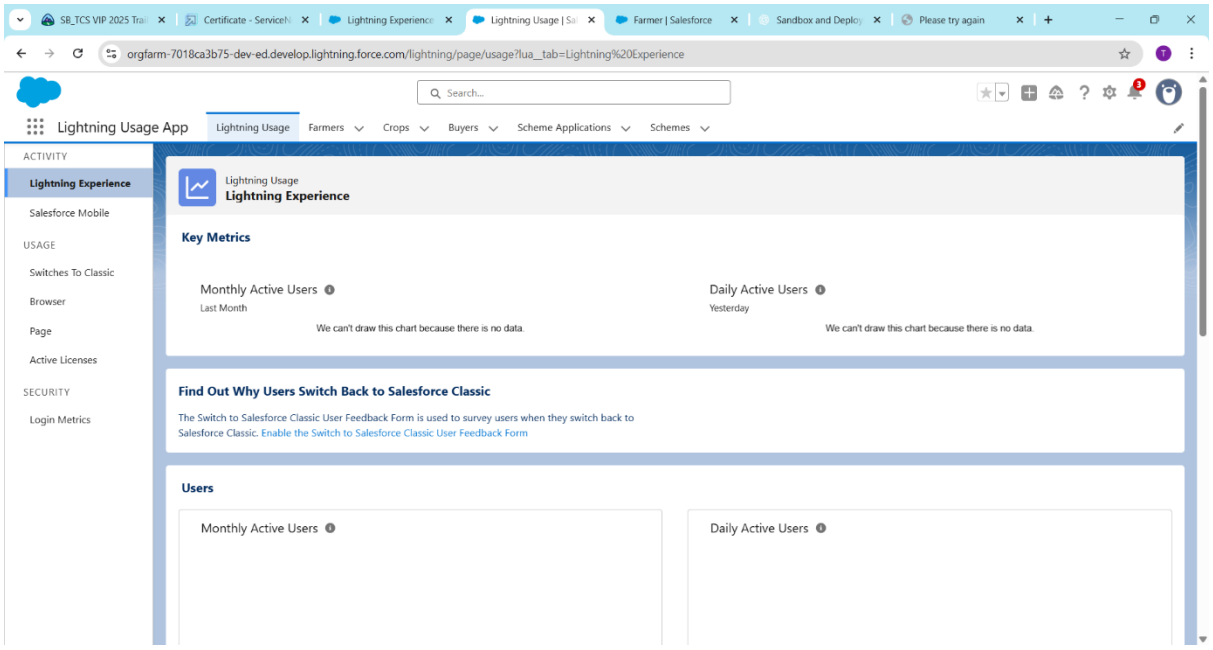
11. Login Access Policies

- Go to **Setup → Login Access Policies**
- Enable **Grant Login Access to Administrators**
- Helps administrators troubleshoot login or data issues while maintaining transparency.



12. Developer Org Setup

- Sign up for a free **Salesforce Developer Org** at developer.salesforce.com
- Navigate to **Setup → App Manager → New Lightning App**
- Create your app named **Rural Market Linkage CRM**
- This serves as the main environment for customization and testing.



.

Phase 1: Problem Understanding & Industry Analysis

Rural Market Linkage CRM

Problem Statement

Rural producers in India struggle with fragmented market access, reliance on intermediaries, and limited exposure to potential buyers. Manual systems for subsidy tracking, crop sales, and communication delay growth and transparency. The vision of the Rural Market Linkage CRM is to bridge these gaps through Salesforce—providing a unified digital hub that connects farmers, cooperatives, buyers, and agricultural officers seamlessly.

Objectives

• Create a Salesforce-powered CRM that centralizes farmer, buyer, and government data.
• Enable direct market linkages via Experience Cloud portals for farmers and buyers.
• Automate workflows for scheme applications, sales tracking, and order management.
• Integrate Service Cloud for rural support, case management, and AI chat assistance.
• Provide analytics dashboards for transparency and market insights.

Stakeholder Analysis

• Farmers: Register crops, access buyers, apply for schemes, and track sales.
• Buyers: Discover rural products, connect directly, and make transparent deals.
• Government Officers: Review scheme applications and monitor subsidy distribution.
• Admins: Manage Salesforce configuration, security, and analytics dashboards.

Business Process Mapping

• **Current Process:**
Farmers rely on middlemen and paper-based workflows, causing inefficiencies and

delayed transactions.

• **Proposed Salesforce Process:**
Farmers register on Experience Cloud, list produce, buyers place orders directly, and scheme approvals flow through automated dashboards.

Industry-Specific Use Case Analysis

• Digital transformation of rural markets for transparency and efficiency.
• Direct buyer-farmer connections to improve profitability.
• Government schemes and order tracking handled digitally.
• Analytics dashboards for data-driven decision-making.

AppExchange Exploration

• Use of existing solutions such as AgriERP and FarmForce analyzed.
• Decision to build a customized Salesforce solution for rural producers and government officers.

Conclusion

Rural Market Linkage CRM transforms traditional rural trade into a smart, data-driven ecosystem. By leveraging Salesforce capabilities—Experience Cloud, Service Cloud, and Einstein Analytics—it bridges the digital divide, empowering rural communities to access fair markets and sustainable growth opportunities.