Warranty & Service Tracker

Phase 1 : Problem Understanding & Industry Analysis

1. Requirement Gathering

– – **Goal:** Identify what each stakeholder needs from the system.

Customers (End Users):

- Easy way to register purchased products with warranty details.
- Automatic notifications before warranty expiry.
- Ability to raise and track service requests online.

Service Agents / Technicians:

- Centralized view of all assigned service requests.
- Quick access to product & warranty details for faster issue resolution.
- Reduced manual paperwork.

Warranty Admins / Customer Support Teams:

- Ability to track all warranties (active & expired) in Salesforce.
- Monitor service request lifecycle (new → in progress → closed).
- Automated assignment of requests to the correct service agent.

Management / Executives:

- Dashboards showing warranty coverage and expiry trends.
- Reports on service requests by product model, issue type, and resolution time.
- Data-driven insights to improve product quality and customer retention.

2. Stakeholder Analysis – –

Goal: Define roles and responsibilities.

Primary Stakeholders:

- **Customers** → Register products, receive reminders, create service requests.
- **Service Agents** → Resolve assigned service requests.
- **Warranty Admins** → Manage warranties, reminders, and escalations.

Secondary Stakeholders:

- **Company Management** → Monitor warranty performance and service KPIs.
- **Salesforce Admins / IT Team** → Configure, customize, and maintain the CRM.
- **Third-party Vendors/Partners** → Handle outsourced repairs or escalations (optional).

3. Business Process Mapping

– – **Goal:** Compare the current manual process vs. Salesforce-enabled solution.

Current Process (Manual/Traditional):

- Customers rely on physical warranty cards and invoices.
- Companies track warranty expiries in Excel or not at all.
- Service requests logged by phone/email → delayed responses.
- No reminders → customers forget to renew or lose coverage.

Proposed Process (Salesforce Enabled):

- **Product & Warranty details stored digitally** in Salesforce.
- **Expiry Date auto-calculated** from Purchase Date + Warranty Term.
- **Automated reminders (Flows/Email Alerts)** sent 30/15/3 days before expiry.
- **Service Requests logged online**, automatically assigned to agents.
- **Dashboards** track active/expired warranties and open/closed service requests.

4. Industry-Specific Use Case Analysis

–– **Goal:** Benchmark against best practices in warranty & service management.

Warranty Renewal Challenge:

- Customers often miss renewal due to lack of reminders.
- 🟩 **Solution:** Salesforce email/SMS reminders before expiry.

Service Request Delays:

- Manual logging causes poor customer satisfaction.
- 🟩 **Solution:** Salesforce Service Cloud + Flows to auto-route service cases.

Customer Experience:

- Leading electronics companies provide portals for self-service.

- 🟩 **Solution:** Salesforce Experience Cloud portal for product registration & tracking.

Management Visibility:

- Without real-time dashboards, companies lack insights.
- 🟩 **Solution:** Salesforce dashboards to monitor warranty/service KPIs.

5. AppExchange Exploration

– – **Goal:** Identify existing Salesforce apps to accelerate development.

Potential Apps:

- **Service Cloud Extensions** → Advanced service request handling.
- **SMS/Email Reminder Apps** → Automated notifications to customers.
- **Field Service Lightning** → Schedule technicians for repairs.
- **Warranty Management Apps** → Prebuilt solutions for warranty lifecycle management.

6. Standard vs. Custom Objects (Gap Analysis)

– – **Goal:** Decide which Salesforce objects to reuse and which to build.

- **Account**
  → Reuse for Customer/Company (no changes needed).
- **Contact**
  → Reuse for Individual Customer (no changes needed).
- **Case**
  → Could be reused for Service Requests, but for learning we will create a custom object **Service_Request_c**.
- **Product (Product2)**
  → Could be reused for Products, but it is missing fields like *Serial Number* and *Warranty Term*.
  → We will create a custom **Product_c** object with these fields.

Phase 2- Warranty & Service Tracker

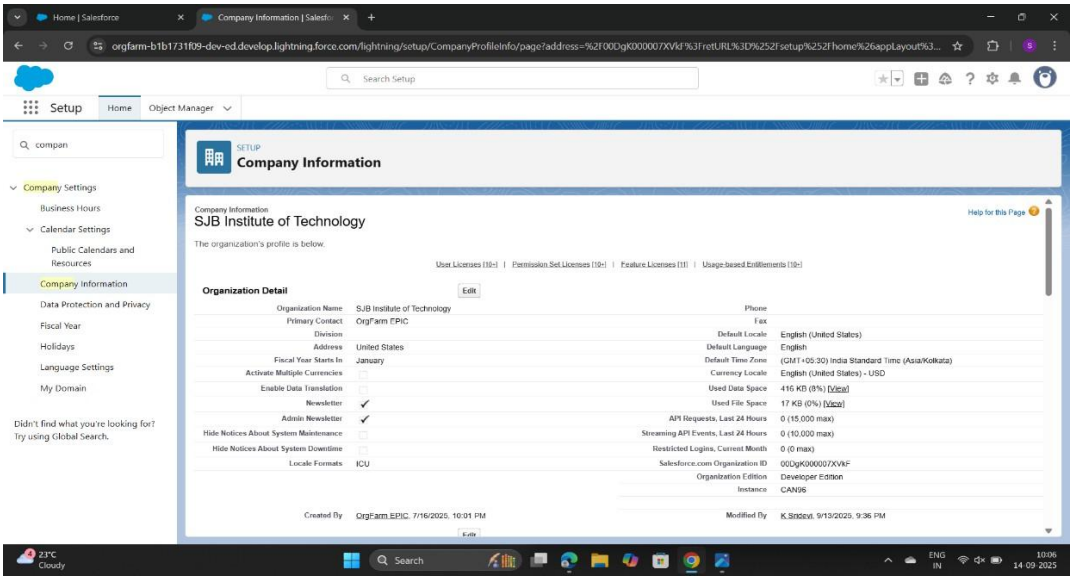
Org Setup & Configuration

1. Salesforce Editions

- Using **Salesforce Developer Edition Org**.
- Developer Edition provides Enterprise-level features free with limited licenses and storage.
- Suitable for student projects and practice.

2. Company Profile Setup

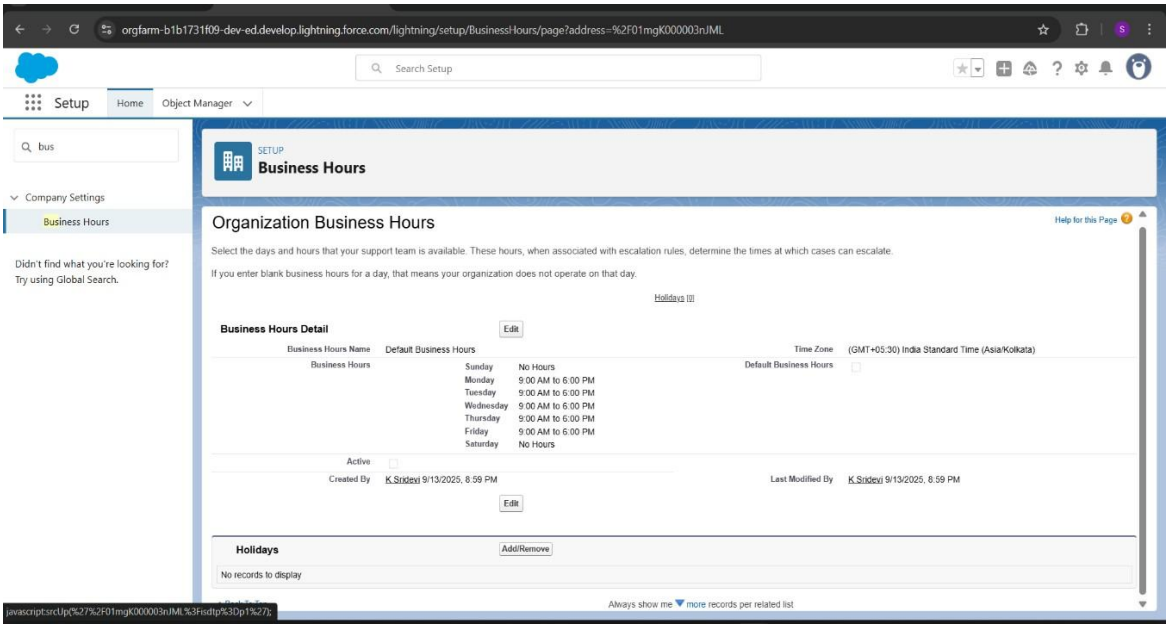
- Company Name: **SJB**
- Time Zone: **Asia/Kolkata**
- Locale: **English (India)**
- Ensures correct organization details and local time display.

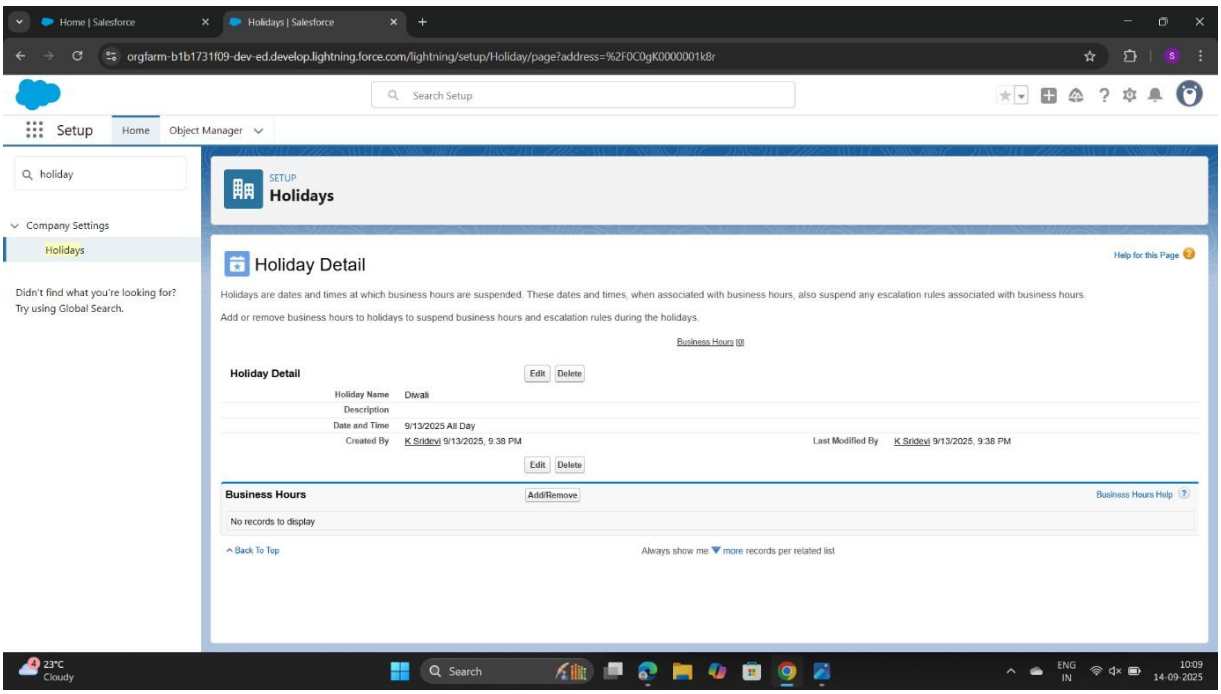


3. Business Hours & Holidays

- Business Hours: **Mon–Fri, 9:00 AM – 6:00 PM IST** (active).
- Holiday Created: **Diwali, 13-Sep-2025 (All Day).**

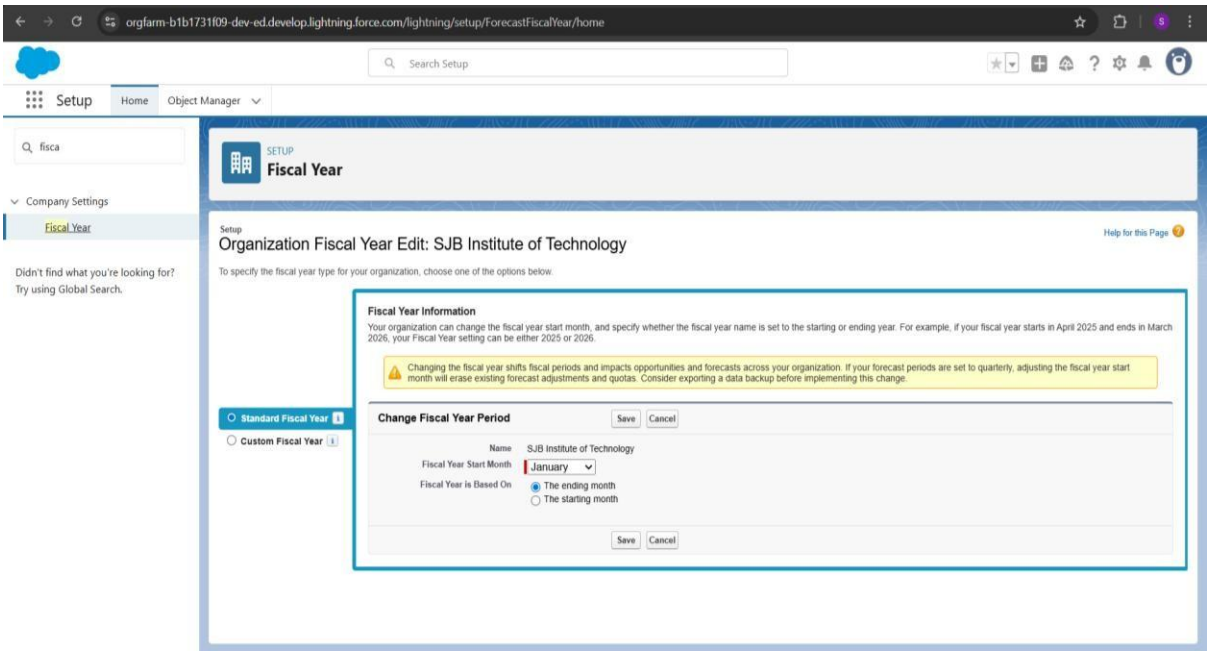
- Guarantees warranty/service requests respect working hours and holidays.





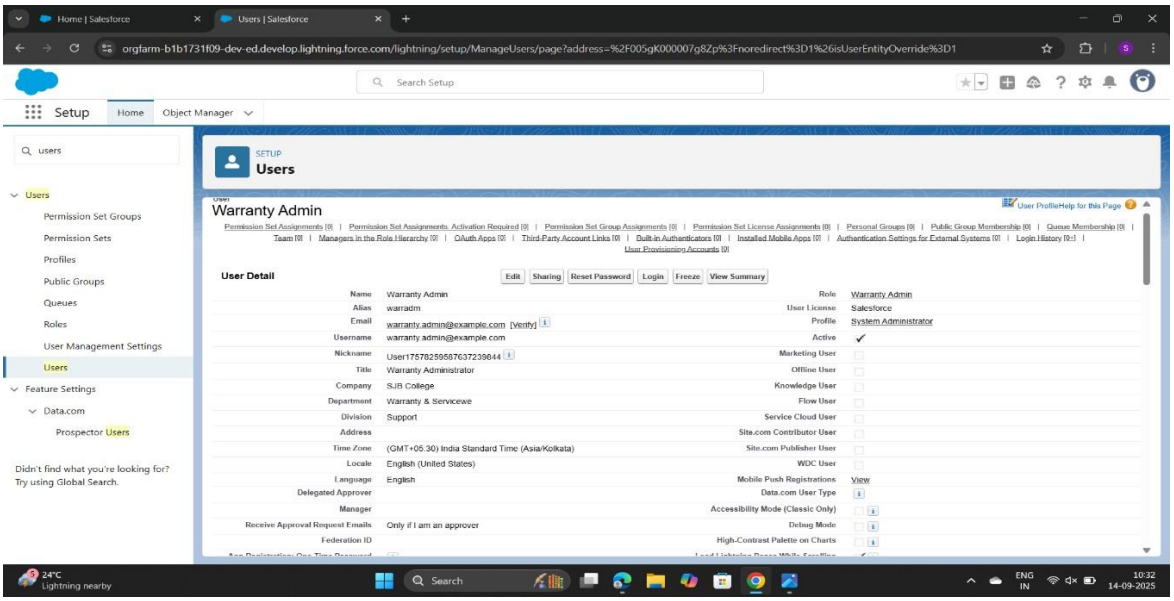
4. Fiscal Year Settings

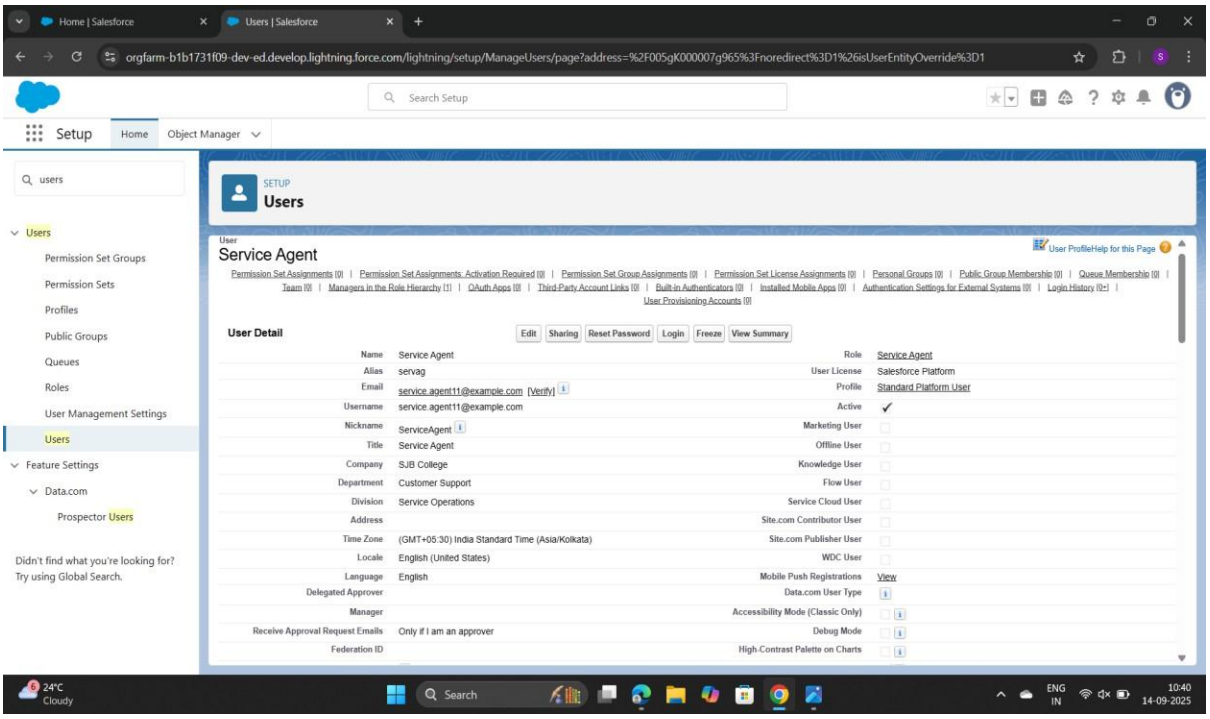
- Fiscal Year: **Standard (January–December)**.
- Custom Fiscal Year not enabled (not needed for this project).



5. User Setup & Licenses

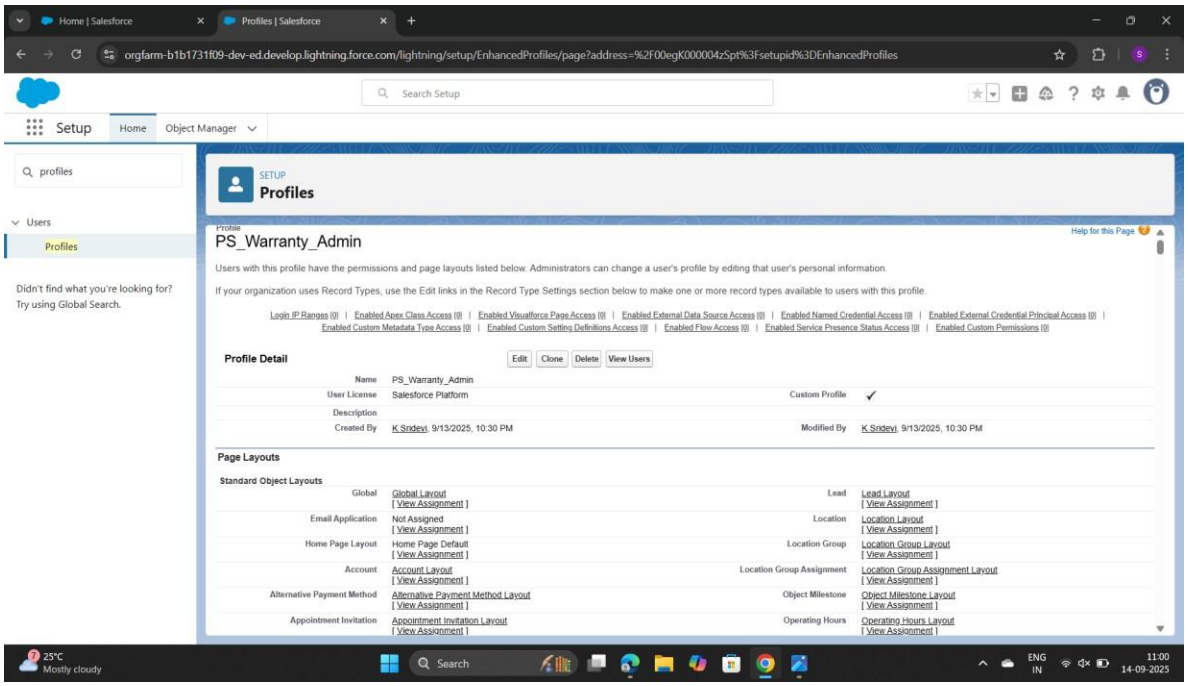
- Created 3 test users:
    1. **Warranty Admin** – manages warranties.
    2. **Service Agent** – handles service requests.
    3. **Manager** – supervises and monitors reports.
- Used **Salesforce Platform licenses** (since only 1 full Salesforce license available).

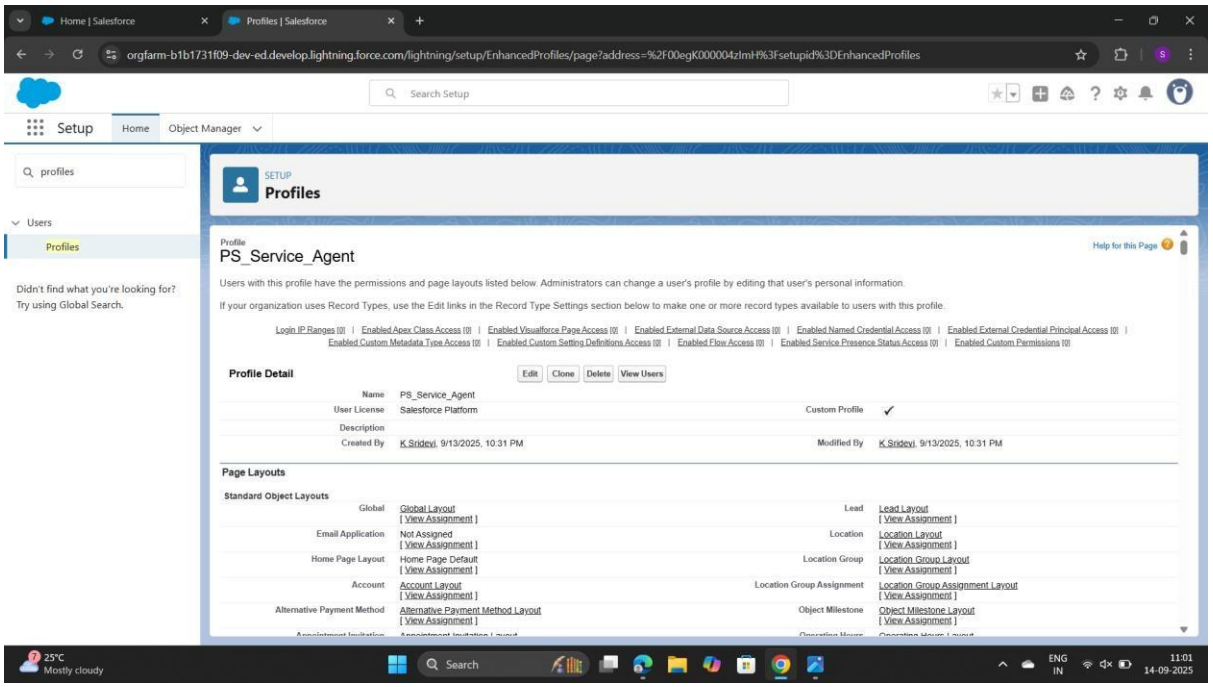


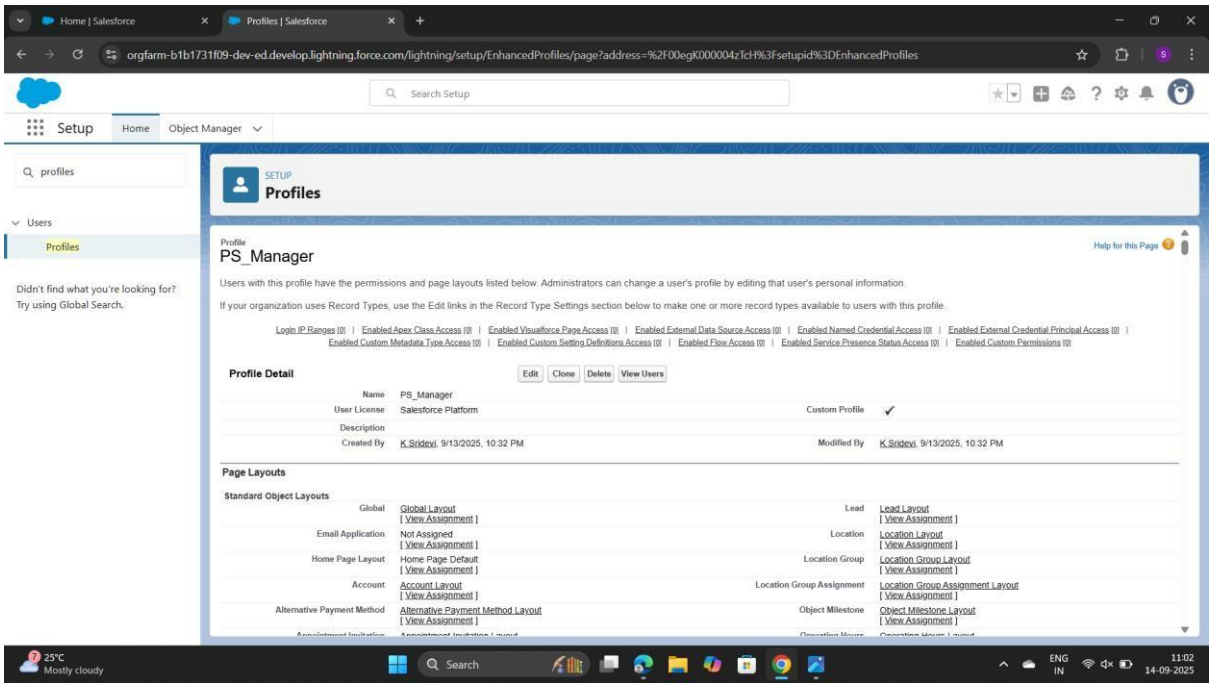


6. Profiles

- Cloned standard profile to create:
  - **PS_Warranty_Admin**
  - **PS_Service_Agent**
  - **PS_Manager**
- Profiles control user permissions.





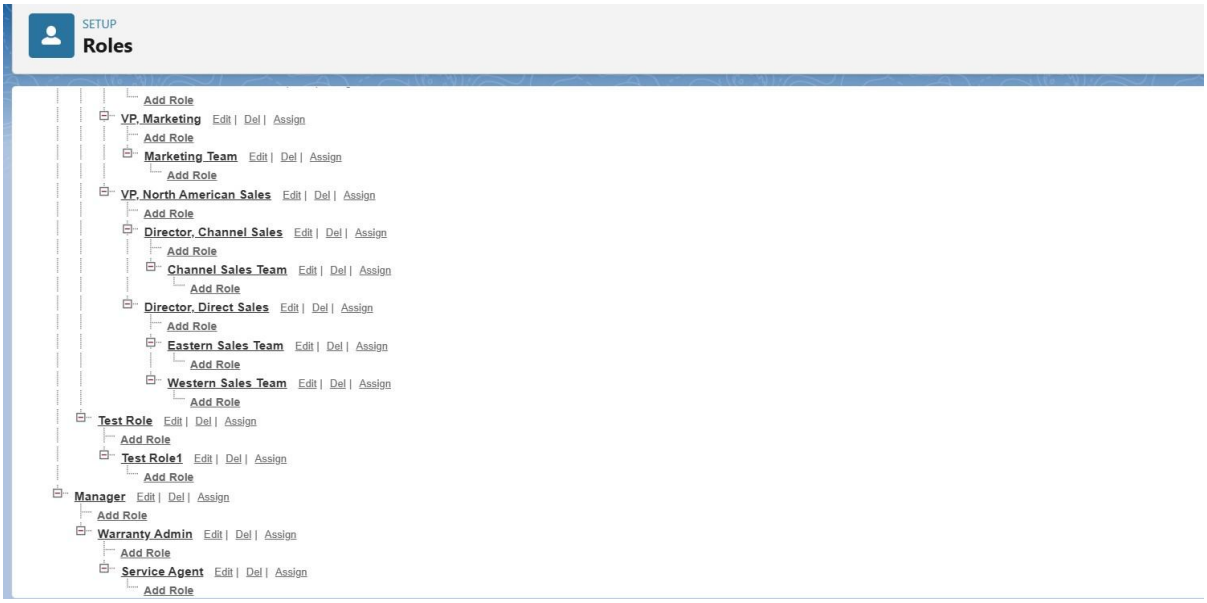


7. Roles

- Role hierarchy created:
    - Manager (top)
        - Warranty Admin

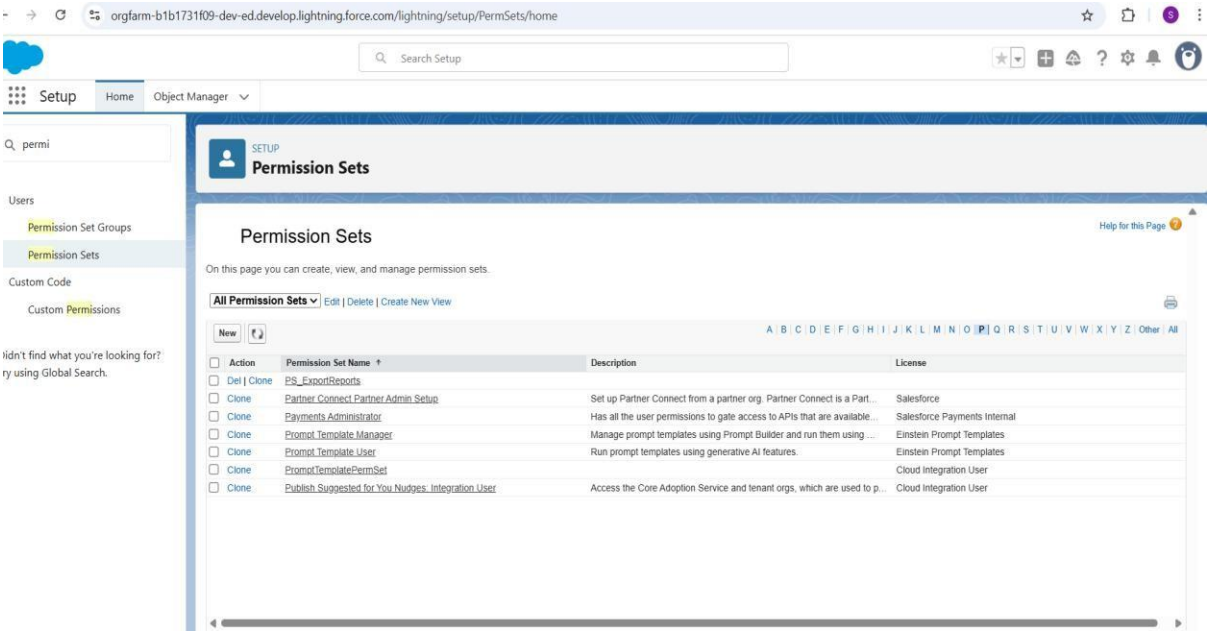
    - Service Agent

- Defines data visibility (Manager can see subordinates' data).



8. Permission Sets

- Created Permission Set: **PS_ExportReports**.
- Enabled: **Export Reports** permission.
- Assigned to Admin user.

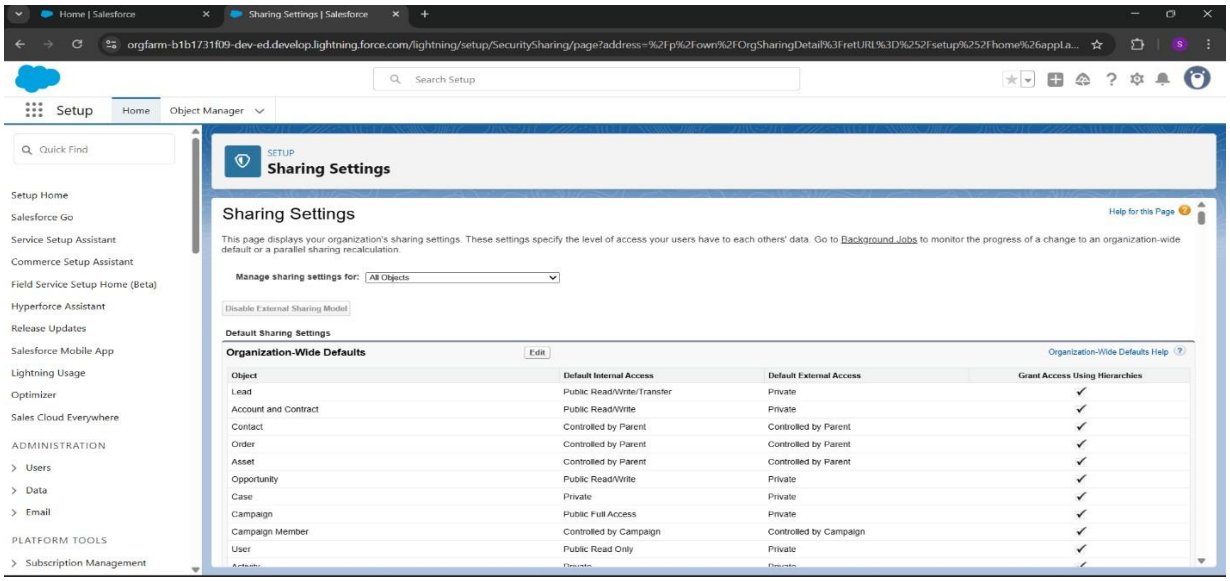


9. Organization-Wide Defaults (OWD)

- Configured baseline data visibility:

- **Account** = Public Read/Write
- **Contact** = Controlled by Parent
- **Case** = Private

- Warranty_c and Service_Request_c will be set in Phase 3.

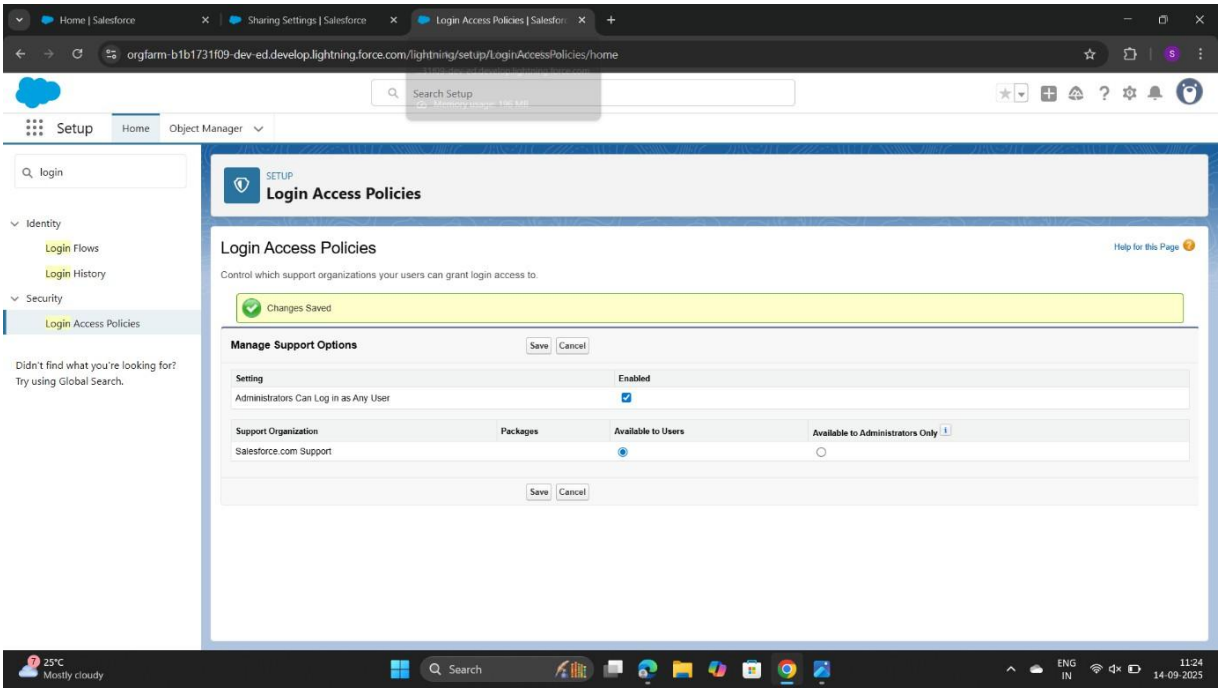


10. Sharing Rules

- Sharing Rule created for Cases:
  - Owned by = Service Agent
  - Shared with = Manager role
  - Access = Read Only
- Ensures Managers can monitor agent cases.

11. Login Access Policies

- Enabled: **Administrators Can Log in as Any User**.
- Allows admin to troubleshoot by logging in as other users.



12. Developer Org Setup

- Project built in **Salesforce Developer Edition Org**.
- Provides free Enterprise-level features for learning and development.

13. Sandbox Usage

- Sandboxes **not available in Developer Edition**.
- In real-world companies, sandboxes are used for testing and training.

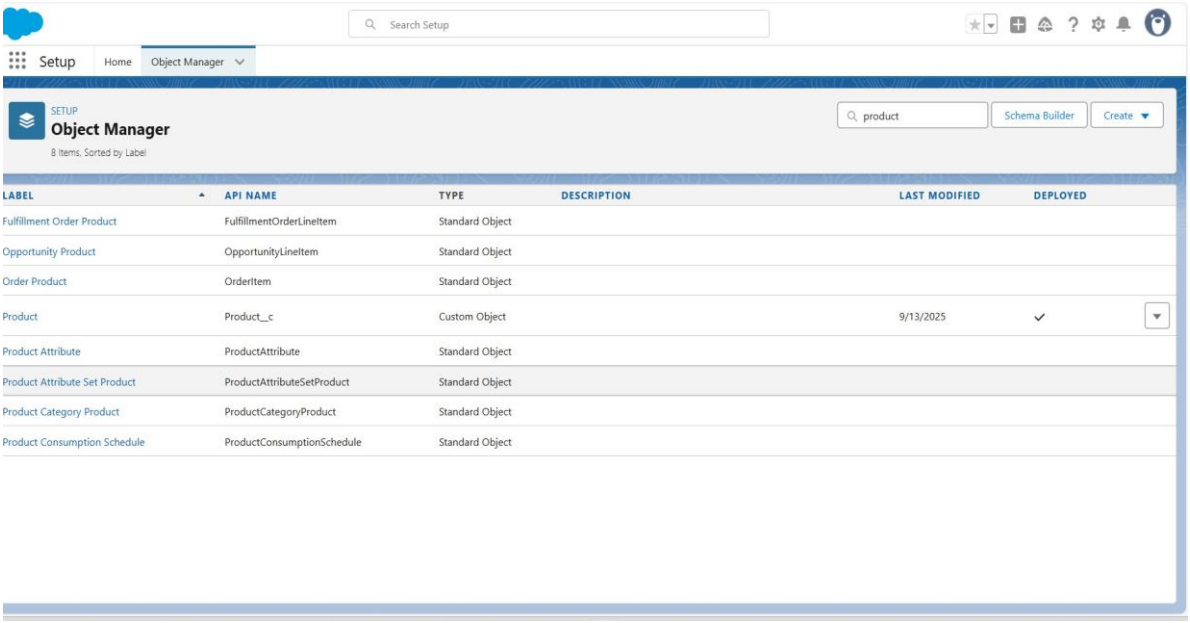
14. Deployment Basics

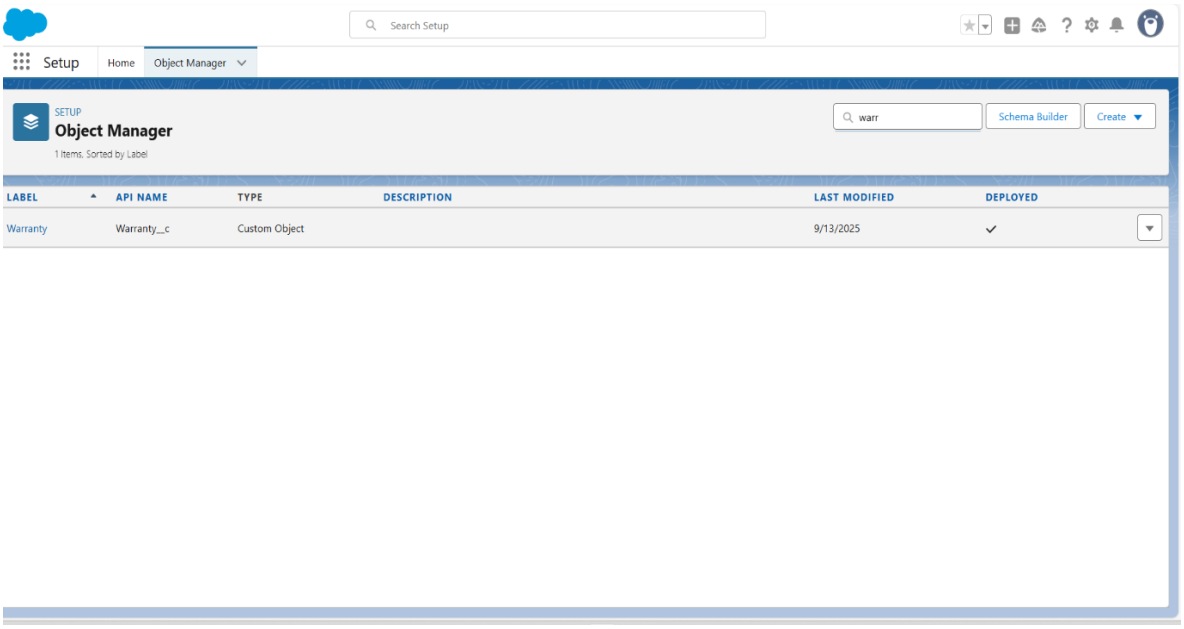
- Deployment methods: **Change Sets** or **Salesforce CLI/VS Code**.
- Not required here since project is completed in a single Developer Org.

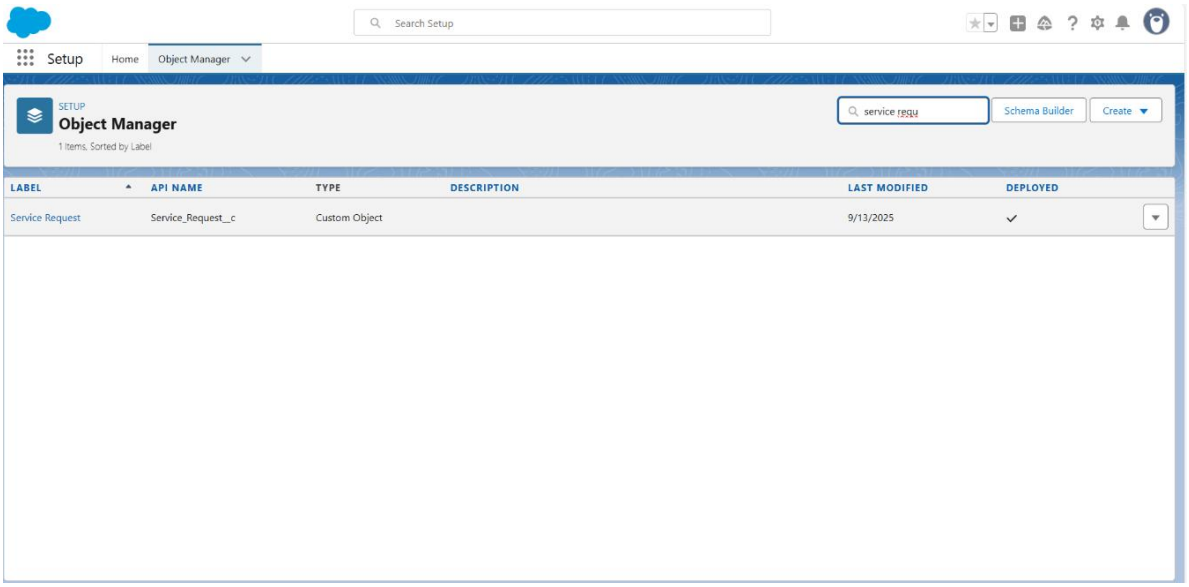
Phase 3: Data Modeling & Relationships

Step 1: Standard & Custom Objects

- Confirm in **Object Manager** that you have:
  - Product_c
  - Warranty_c
  - Service_Request_c
- (You already created these ☑).





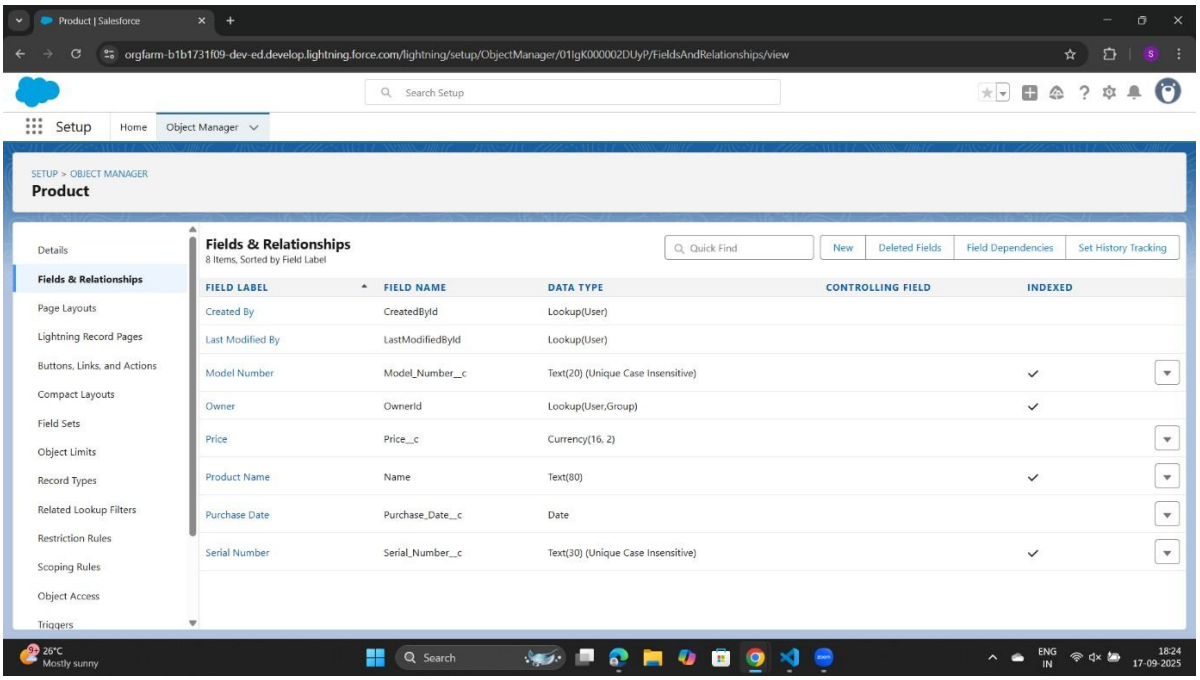


❖ Step 2: Fields

Create fields for each

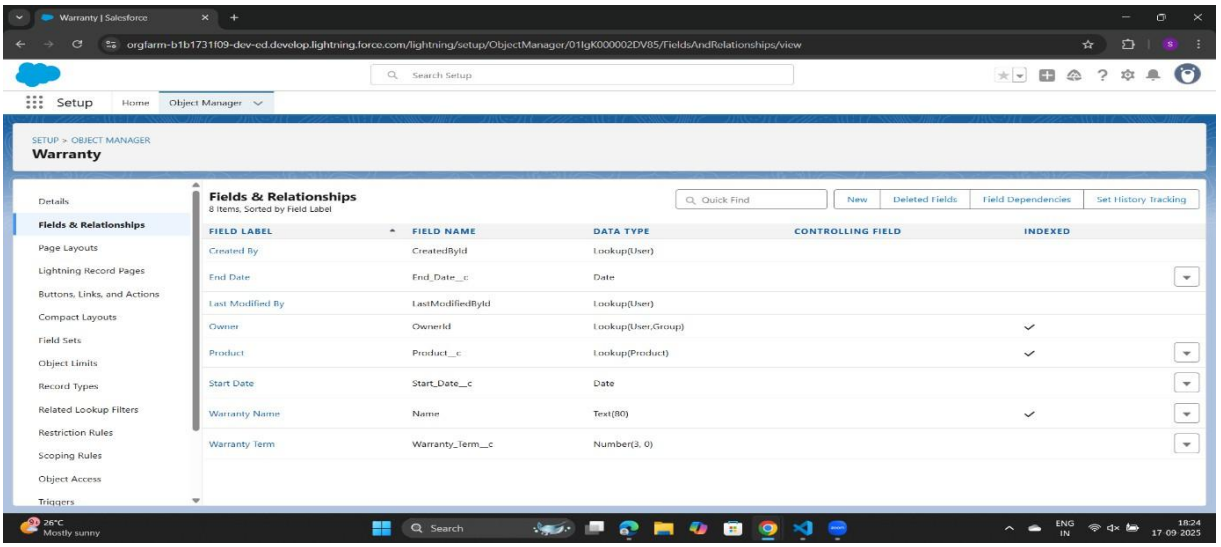
object. **Product_c**

- Serial Number → Text(30), Unique ☑
- Model Number → Text(20), Required ☑
- Price → Currency(16,2), Required ☑
- Purchase Date → Date, Required ☑



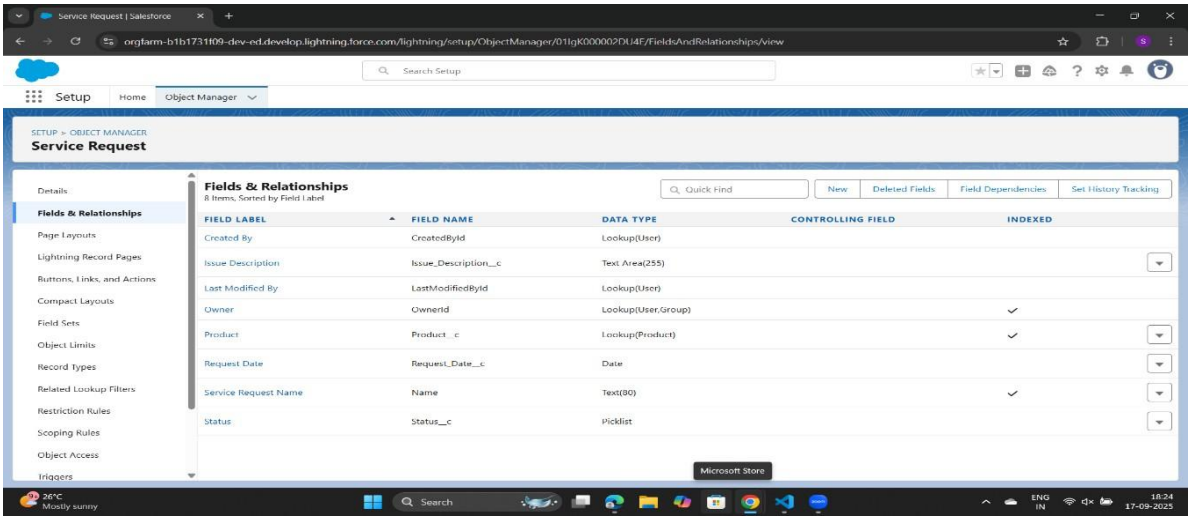
Warranty_c

- Start Date → Date, Required ☑
- End Date → Date, Required ☑
- Warranty Term (Months) → Number(3,0), Required ☑
- Related Product → Lookup(Product_c), Required ☑



Service_Request_c

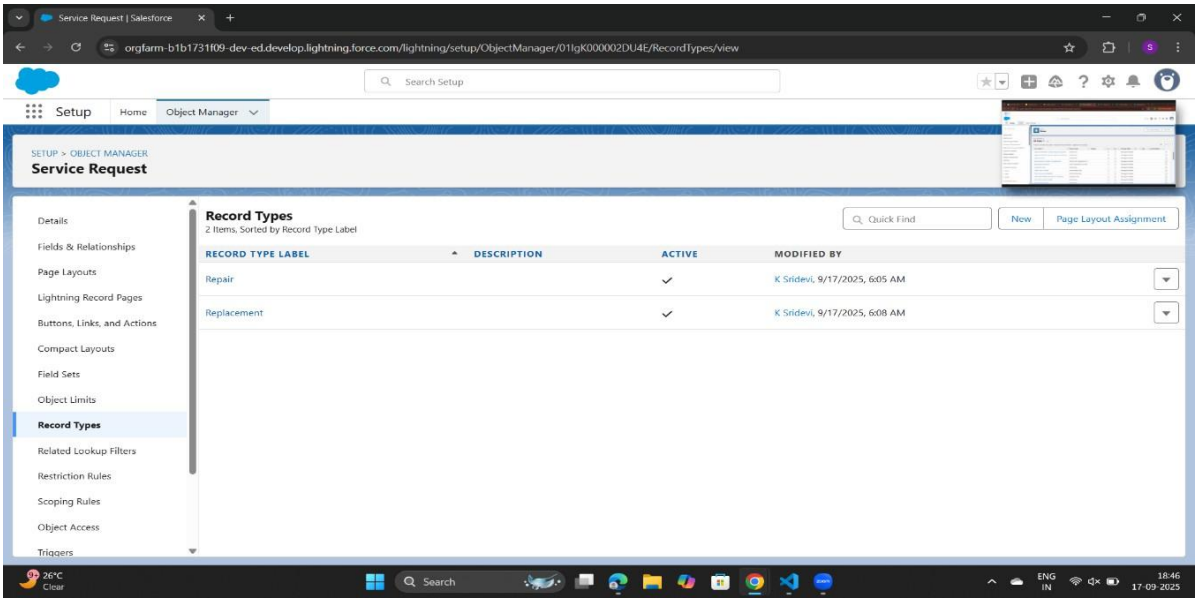
- Request Date → Date, Default = TODAY(), Required ☑
- Status → Picklist (New, In Progress, Completed, Closed), Default = New
- Issue Description → Long Text Area (500)
- Related Product → Lookup(Product_c), Required ☑
- Related Warranty → Lookup(Warranty_c), Required ✗



❖ **Step 3: Record Types**

(Do this only for Service_Request_c).

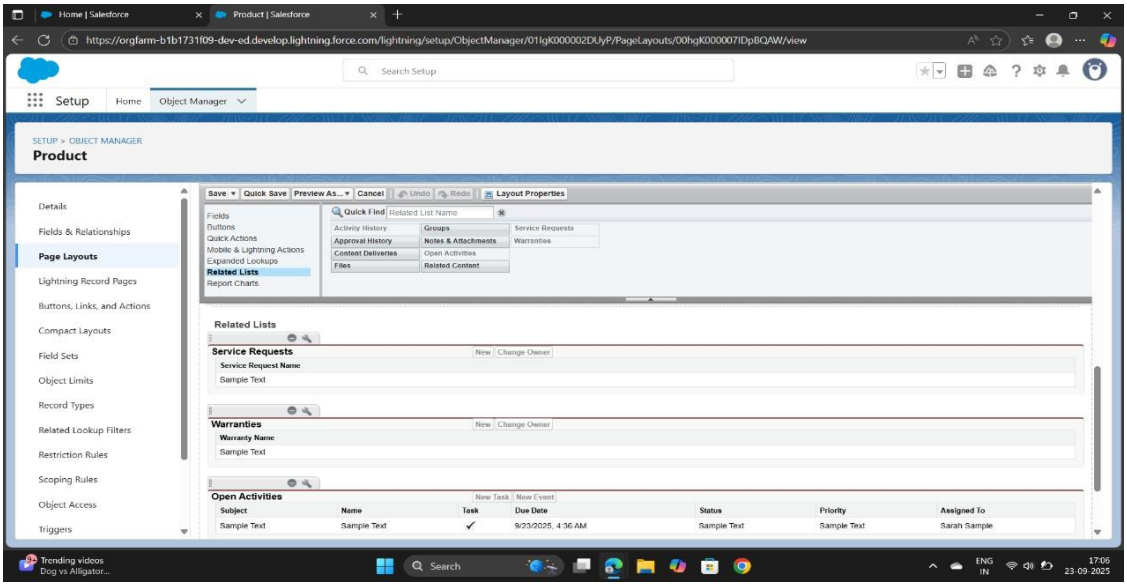
- Create **Repair** record type.
- Create **Replacement** record type.
- Assign to all profiles.
- For both, use the *Service Request Layout* (for now).



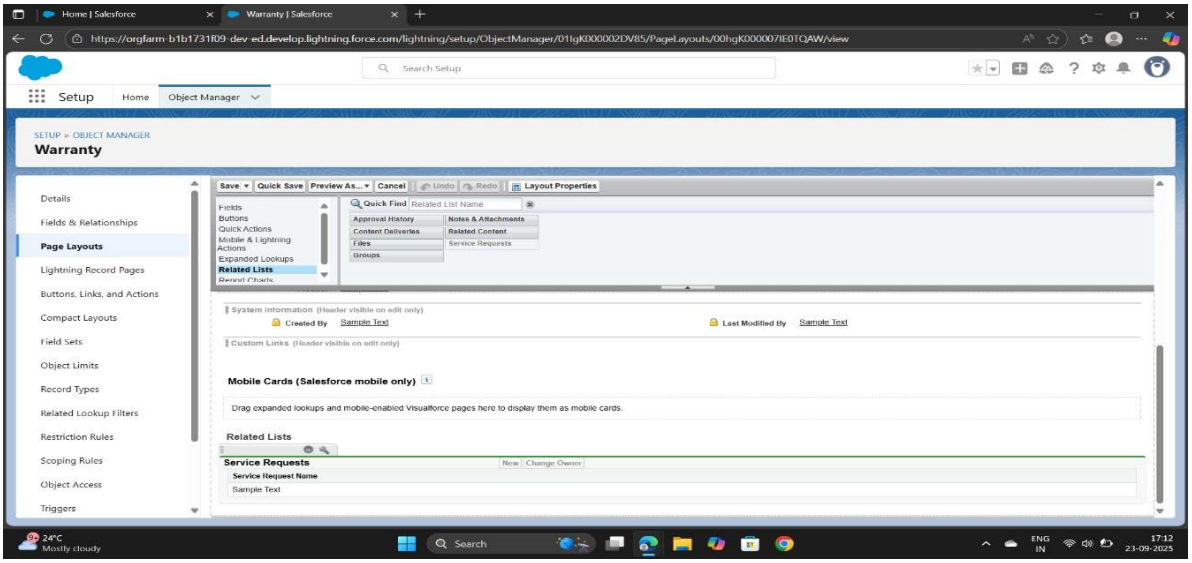
❖ Step 4: Page Layouts

Customize layouts so relationships are visible.

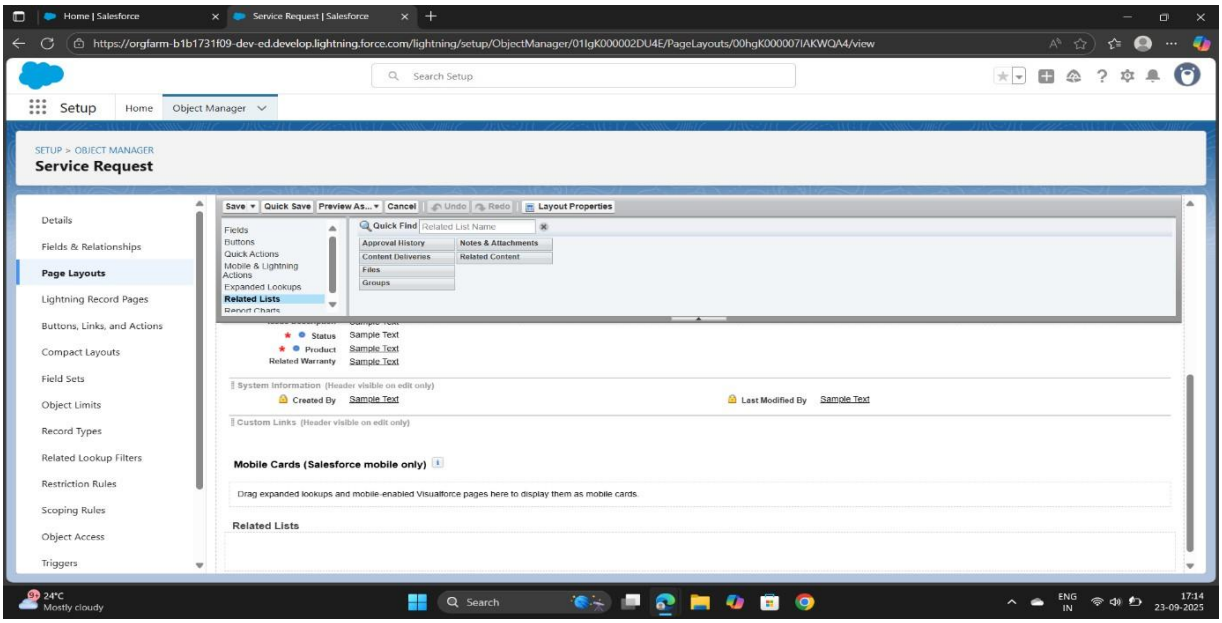
- **Product_c Layout** → add Related Lists: Warranties, Service Requests.



- **Warranty_c Layout** → add Related Product + Service Requests.

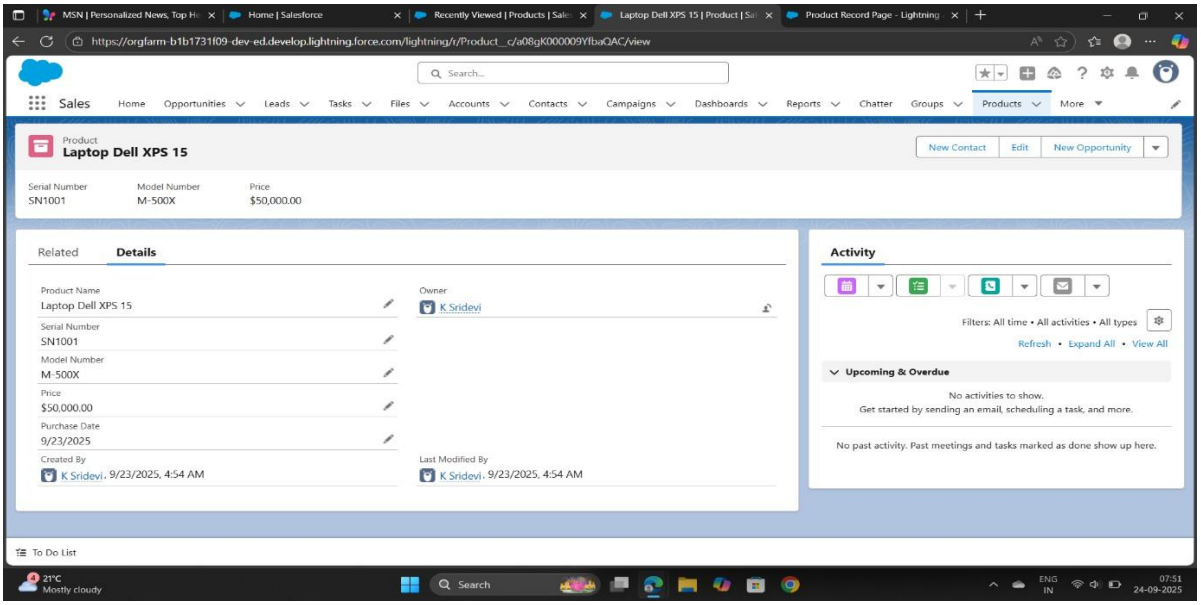


- **Service_Request_c Layout** → add Related Product + Related Warranty.



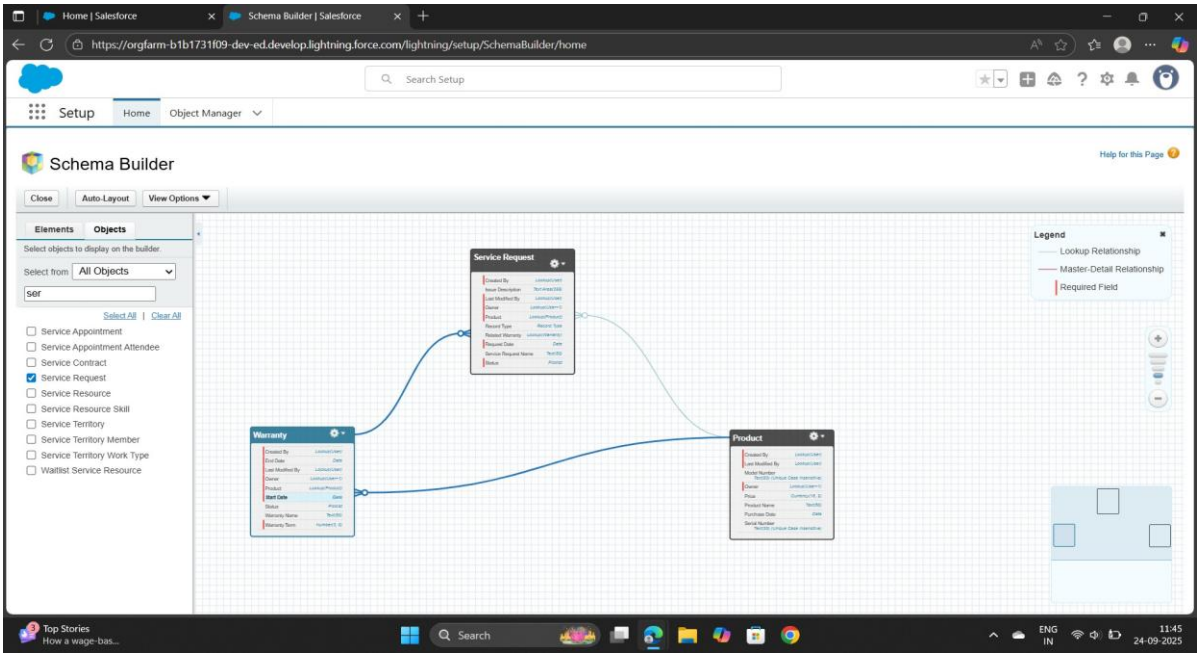
❖ Step 5: Compact Layouts

- For each object, set compact layout fields (these show in Highlights Panel).
- Example:
  - Product: Serial Number, Model Number, Price
  - Warranty: Start Date, End Date, Product
  - Service Request: Status, Request Date, Product



❖ Step 6: Schema Builder

- Go to **Schema Builder**.
- Select your 3 objects (Product, Warranty, Service Request).
- See the relationship lines.



❖ Step 7: Relationships

- Ensure relationships are correct:
  - Product → Warranty (Lookup)
  - Product → Service Request (Lookup)

- Warranty → Service Request (Lookup)

❖ **Step 8 — Junction Objects**

A Junction Object in Salesforce is a custom object with two Master-Detail relationships used to model a many-to-many relationship between two objects. For example, if a Service Request could apply to multiple Products, and each Product could be linked to multiple Service Requests, then we would create a junction object called Product_Service_Request_c with:

- Master-Detail to Product_c
- Master-Detail to Service_Request_c
- In our Warranty & Service Tracker project, we do not require junction objects because our relationships are **one-to-many**.
- A Product can have many Warranties and Service Requests.
- But each Warranty and Service Request belongs to only one Product.

Hence, junction objects were not created but are explained here for completeness.

❖ **Step 9 — Hierarchical Relationships**

Salesforce also supports a special type of relationship called **Hierarchical Relationship**, which is only available on the User object.
It allows users to be related in a parent-child way, such as showing reporting structures (e.g., Manager → Agent).

In our Warranty & Service Tracker project, hierarchical relationships are **not required** because we are focusing on Products, Warranties, and Service Requests. However, hierarchical relationships are often used in real-life scenarios for approval flows or reporting managers.

🔷 Phase 4: Process Automation

Step 1: Validation Rules

Validation Rules stop users from saving wrong or incomplete data.

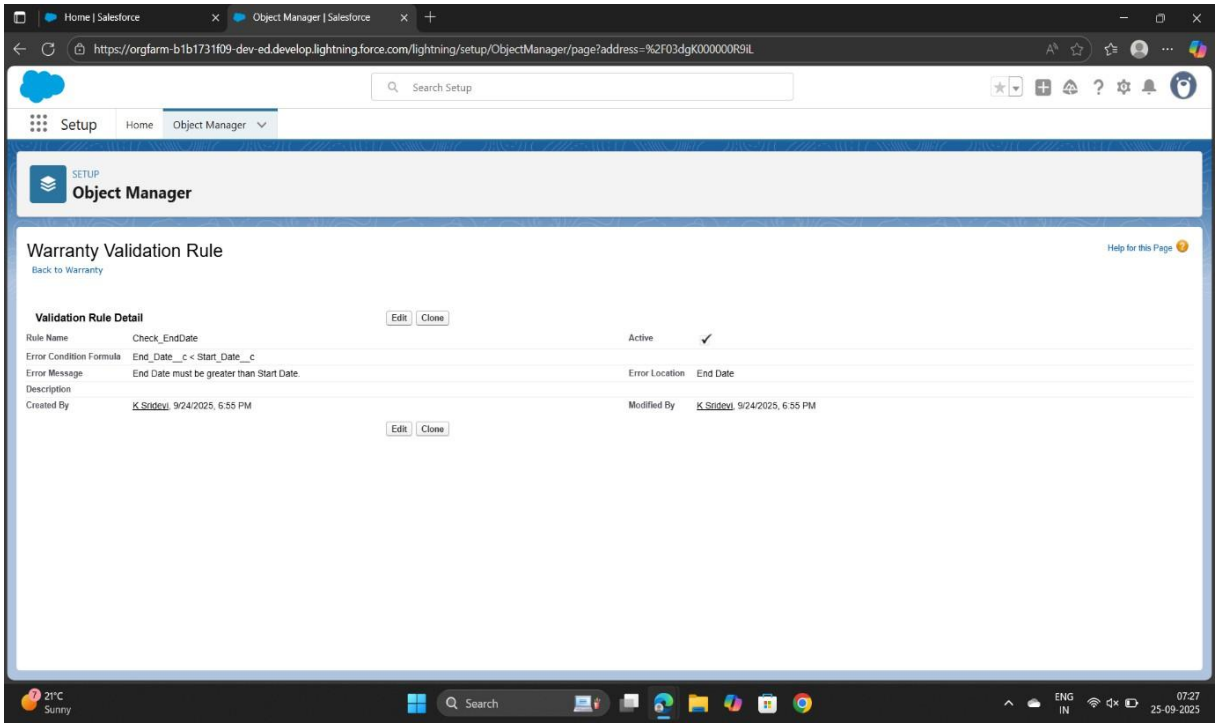
A. Warranty Dates Validation

Goal: End Date must be greater than Start Date.

- Go to **Setup → Object Manager → Warranty_c → Validation Rules → New**.
- Rule Name: Check_EndDate
- Formula:

End_Date_c < Start_Date_c

- Error Message: **"End Date must be greater than Start Date."**
- Error Location: Field → End Date.
- Save.



B. Service Request → Issue Description Required

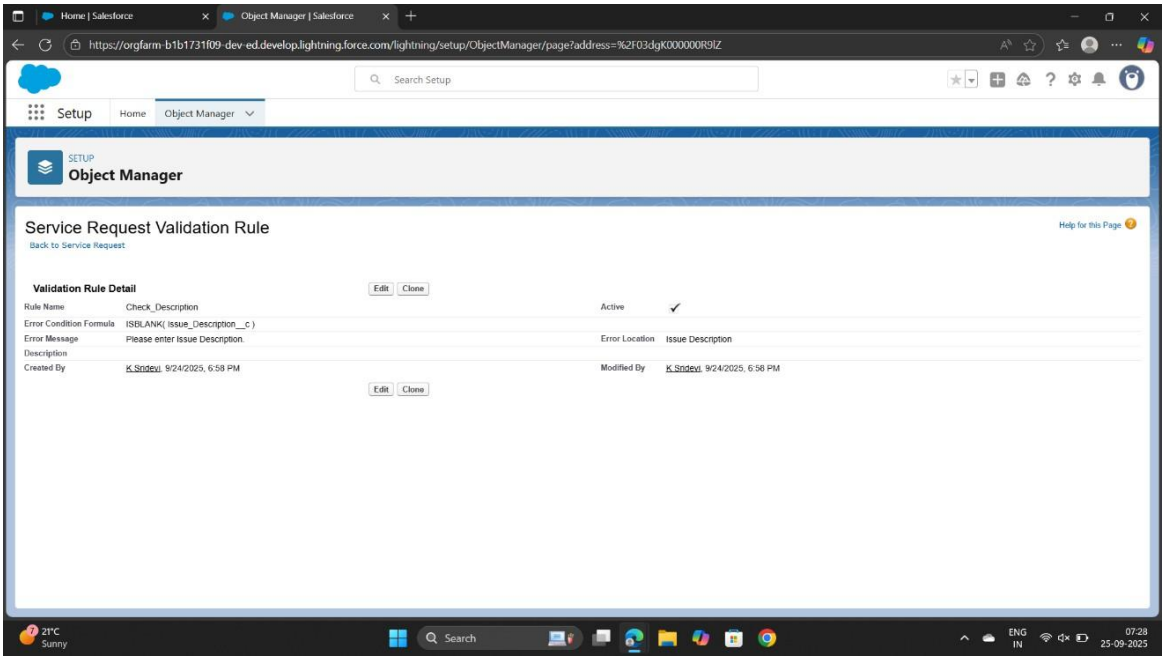
Goal: Every Service Request must have a description.

- Go to **Setup → Object Manager → Service_Request_c → Validation Rules → New**.
- Rule Name: Check_Description

- Formula:

ISBLANK(Issue_Description_c)

- Error Message: **“Please enter Issue Description.”**
- Error Location: Field → Issue Description.
- Save.



Step 2: Workflow Rule + Email Alert

Workflow Rule automates actions when criteria is met.

Example: Notify when Warranty is Expiring

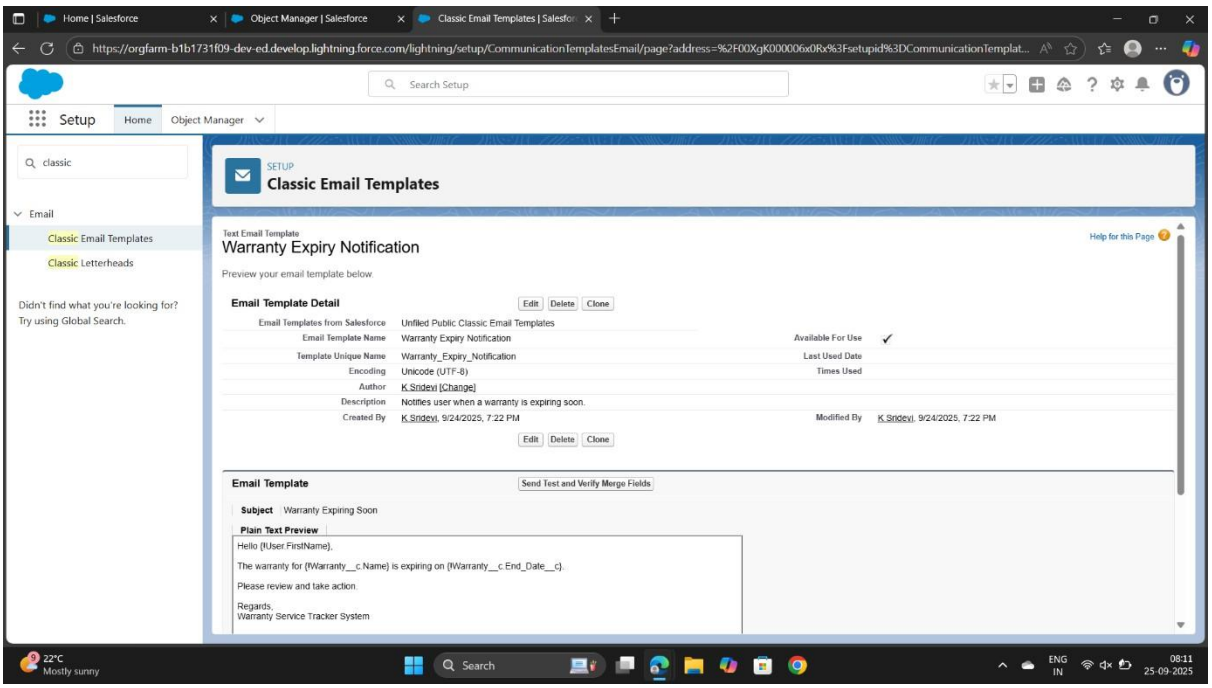
- Setup → Workflow Rules → New Rule.
- Object: **Warranty_c**.
- Rule Name: Warranty_Expiry_Alert.
- Evaluation: “created, and every time it’s edited”.
- Rule Criteria: End_Date c = TODAY() + 30
  (means warranty expiring in 30 days).

Action → Email Alert

- Create Email Template:
  - Setup → Classic Email Templates → New.
  - Folder: Unfiled Public.

- Template Name: Warranty Expiry Notification.
- Subject: Warranty Expiring Soon.
- Body: “Warranty for {!Warranty_c.Name} is expiring on {!Warranty_c.End_Date_c}.”

- Back to Workflow → Add Action → Email Alert → choose template + recipient (Owner/User).
- Activate Rule.

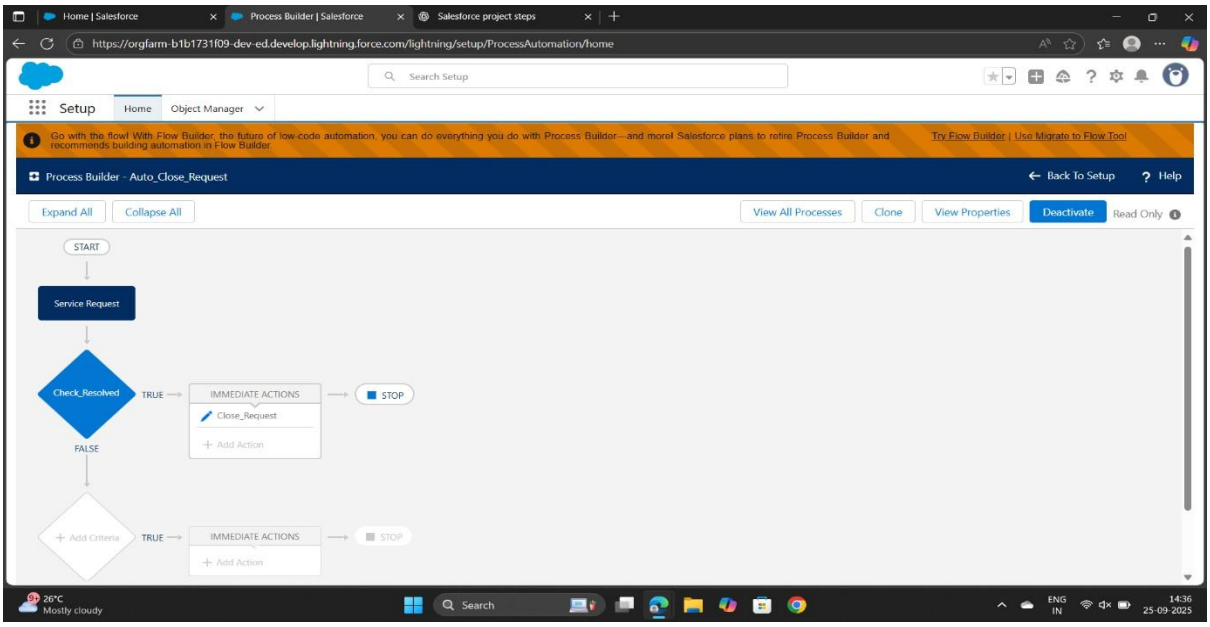


Step 3: Process Builder Automation

Process Builder automates record updates or actions.

Example: Auto-update Service Request Status

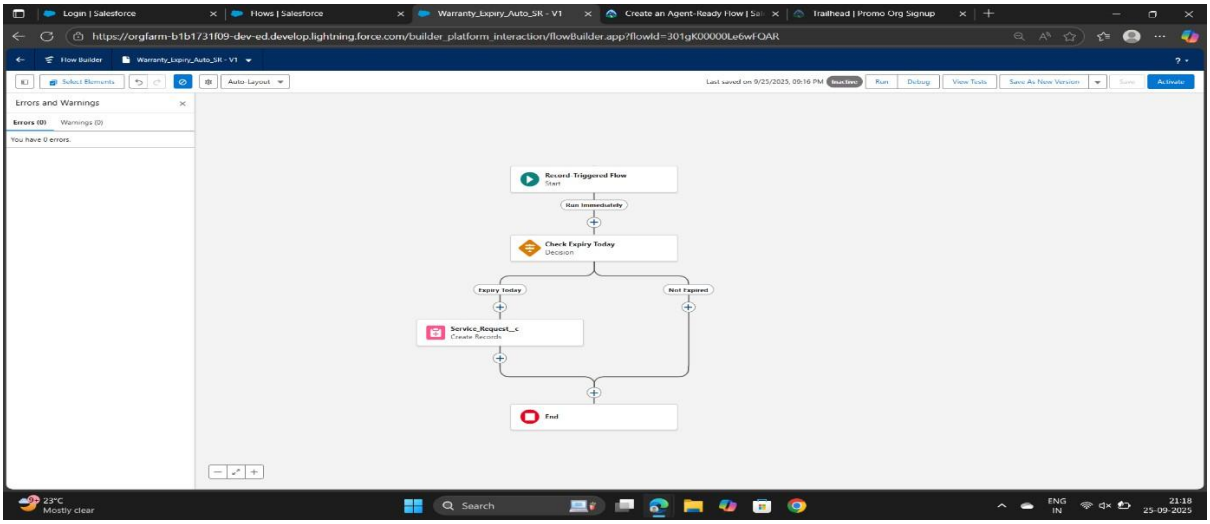
- Setup → Process Builder → New.
- Name: Auto_Close_Request.
- Object: **Service_Request_c**.
- Trigger: “when record is created or edited”.
- Criteria: Resolved_c = TRUE (checkbox field, you can create it if not already).
- Action: Update field → Status = “Closed”.
- Activate Process.



Step 4: Record-Triggered Flow

Flows are more advanced automation.

Example: Auto-Create Service Request on Warranty Expiry

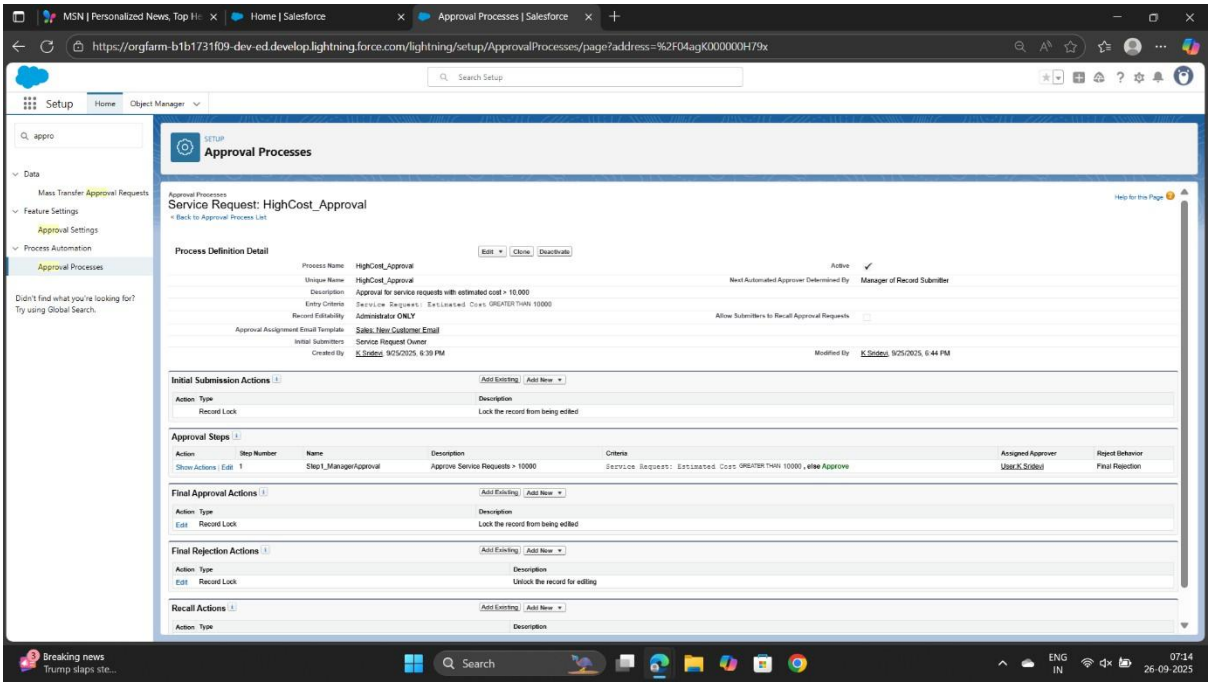
- Setup → Flows → New Flow.
- Choose: **Record-Triggered Flow**.
- Object: **Warranty_c**.
- Trigger: When record is updated → condition: End_Date_c = TODAY().
- Action: Create Records → New Service_Request_c.
    - Fields: Name = “Auto-Generated Expiry Request”, Status = “Open”, Product = Warranty.Product_c.
- Save & Activate.
- 

Step 5: Approval Process

Approval Processes let managers approve/reject records.

Example: Approve Service Request Over Certain Amount

- Setup → Approval Processes → New Approval Process.
- Object: **Service_Request_c**.
- Name: HighCost_Approval.
- Entry Criteria: Estimated_Cost_c > 10000.
- Approver: Role → Manager (or a specific user).
- Final Action on Approval: Update field → Status = “Approved”.
- Final Action on Rejection: Update field → Status = “Rejected”.
- Save & Activate.



Phase 5: Apex Programming (Developer)

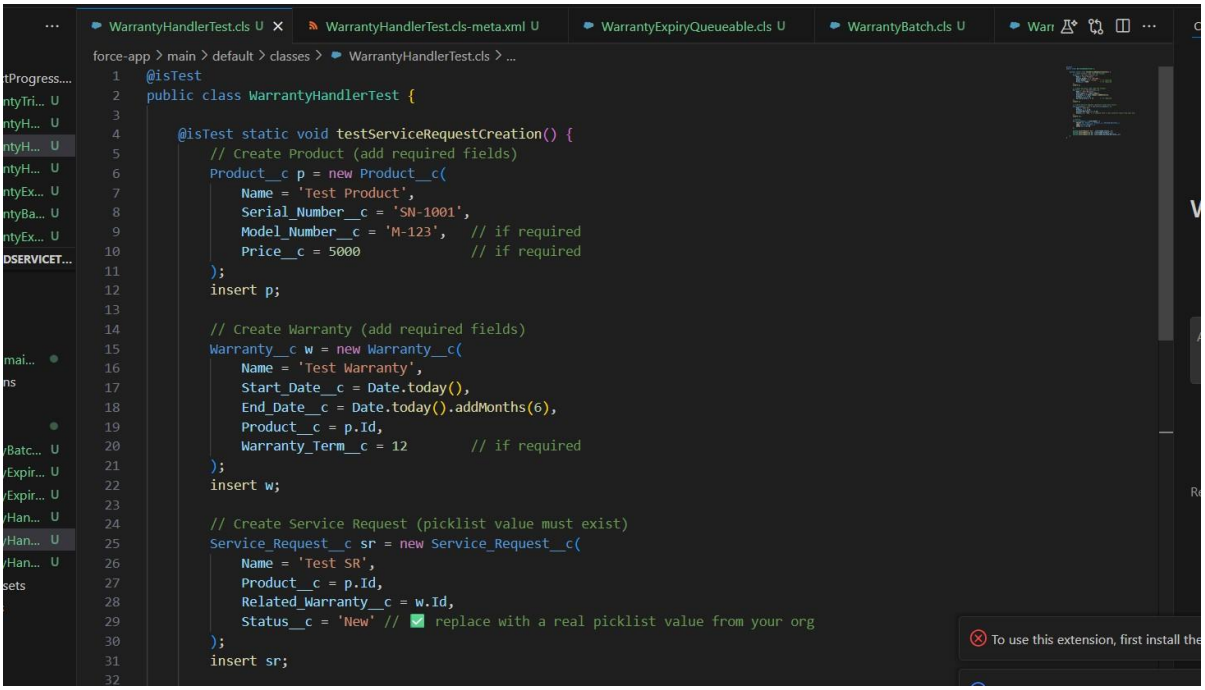
1. Classes & Objects

What it is:

Apex classes are like Java classes. They contain logic and methods. Objects are the Salesforce database tables (Product, Warranty, Service Request).

What we did:

- Created a class WarrantyHandler.cls in VS Code.
- This class contains logic to automatically update Warranty End Date and create Service Requests.
- Deployed class to Salesforce Org using SFDX.

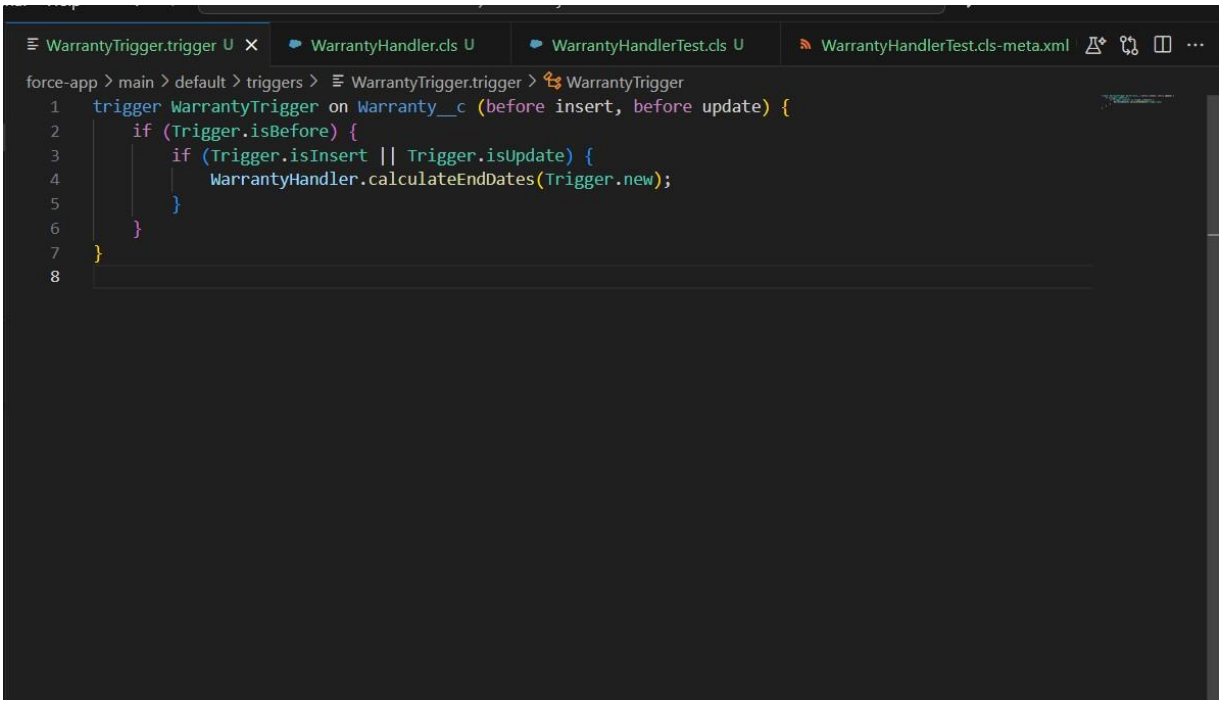


2. Apex Triggers (before/after insert/update/delete) What it is:

Triggers run automatically when records are inserted/updated/deleted.

What we did:

- Created WarrantyTrigger.trigger on Warranty_c.
- Logic: When Warranty is inserted, set **End Date** = Start Date + 12 months (default).
- Trigger calls handler class (best practice).



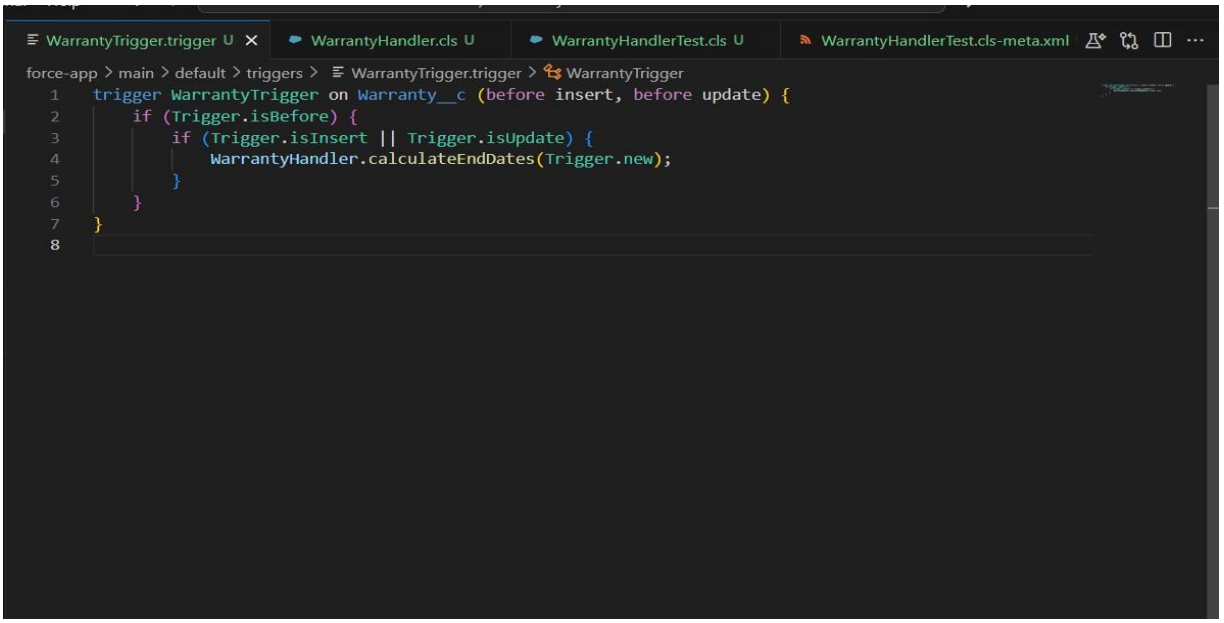
3. Trigger Design

Pattern What it is:

Separating logic from trigger into handler class. Keeps code clean and testable.

What we did:

- All business logic was written in WarrantyHandler.cls.
- Trigger only calls WarrantyHandler.handleAfterInsert().

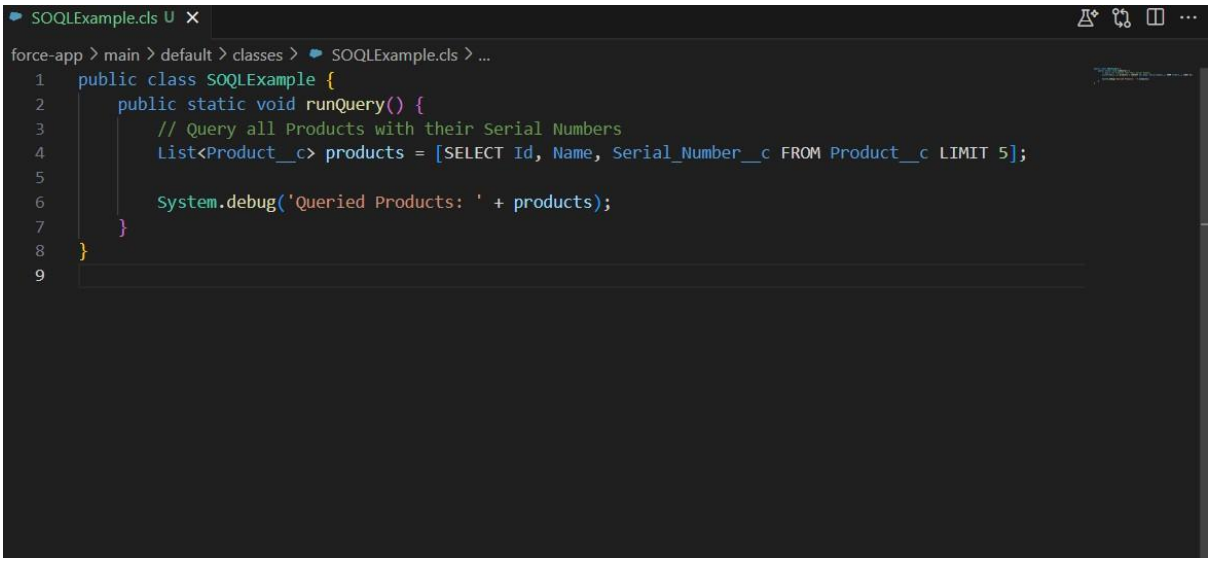


4. SOQL & SOSL

What it is:
SOQL = Salesforce Object Query Language (like SQL). SOSL = search across objects.

What we did:

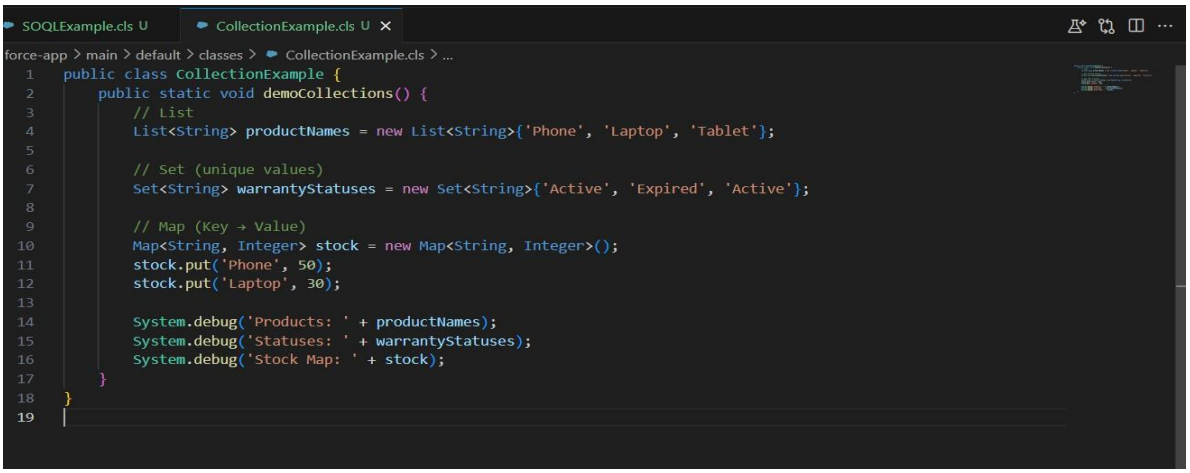
- Wrote SOQL queries in Apex class:
- [SELECT Id, Status_c FROM Service_Request_c WHERE Id = :sr.Id]
- Used this to verify if records are created correctly.。 .・ ◆ç ̂. **Screenshot:** Code snippet in VS Code.



5. Collections (List, Set, Map)
What it is:
Used to store multiple records in Apex.

What we did:

- Created List<Service_Request_c> to hold multiple records.
- Used Map<Id, Warranty_c> when relating products and warranties.



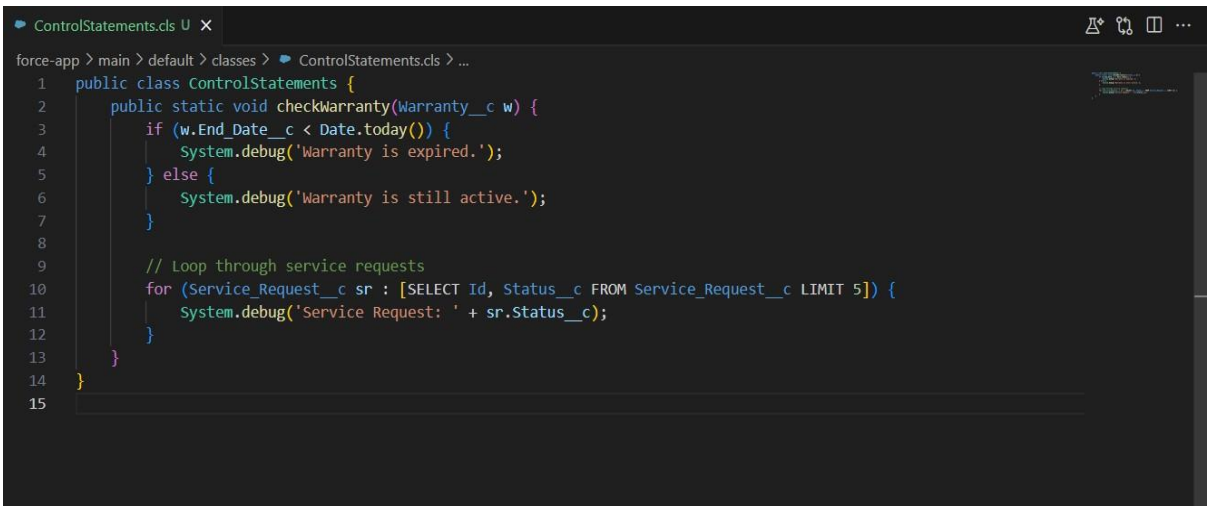
6. Control Statements

What it is:

if-else, for loops in Apex.

What we did:

- Used if condition to check if Warranty has Start Date.
- Used for loop to process multiple records.



7. Batch Apex (Conceptual)

What it is:

Runs large jobs in batches.

What we did:

- (For documentation) Wrote explanation: “Batch Apex could be used to process thousands of Service Requests, but in our project we used Queueable instead.”

8. Queueable Apex

What it is:

Runs jobs asynchronously (in background).

What we did:

- Created WarrantyQueueable.cls.
- Logic: Create Service Request automatically when Warranty expires.
- Enqueued job from Developer Console.

ControlStatements.cls U WarrantyExpiryQueueable.cls U

force-app > main > default > classes > WarrantyExpiryQueueable.cls > ...

```
public class WarrantyExpiryQueueable implements Queueable {
    public void execute(QueueableContext ctx) {
        List<Warranty__c> listW = [
            SELECT Id, Product__c, End_Date__c
            FROM Warranty__c
            WHERE End_Date__c = :Date.today()
        ];
        List<Service_Request__c> toCreate = new List<Service_Request__c>();
        for (Warranty__c w : listW) {
            Service_Request__c sr = new Service_Request__c(
                Name = 'Auto-Generated Expiry Request',
                Status__c = 'Open',
                Product__c = w.Product__c
            );
            toCreate.add(sr);
        }
        if (!toCreate.isEmpty()) insert toCreate;
    }
}
```

9. Scheduled Apex

What it is:

Runs Apex at scheduled times.

What we did:

- Created WarrantyScheduler.cls.
- Scheduled job daily to run WarrantyQueueable.

ControlStatements.cls U WarrantyExpiryScheduler.cls U

force-app > main > default > classes > WarrantyExpiryScheduler.cls > ...

```
global class WarrantyExpiryScheduler implements Schedulable {
    global void execute(SchedulableContext ctx) {
        System.enqueueJob(new WarrantyExpiryQueueable());
    }
}
```

10. Future Methods

What it is:

Used for async processing (like callouts).

What we did:

- Added a sample @future method in WarrantyHandler.cls to demonstrate usage.

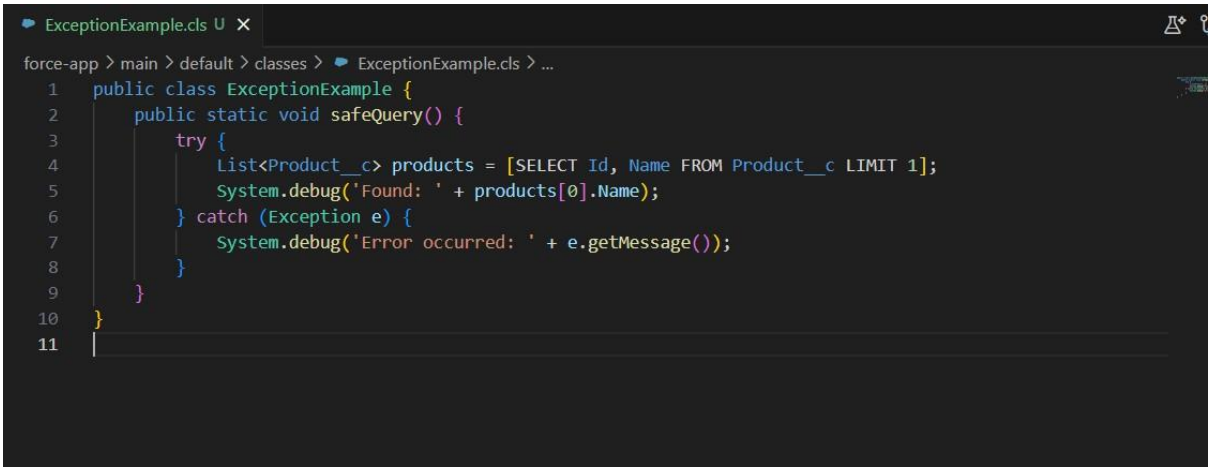
11. Exception Handling

What it is:

Try-catch to handle errors gracefully.

What we did:

- Wrapped SOQL queries in try-catch.
- Example: if product not found → throw custom error.



12. Test Classes

What it is:

Unit tests for Apex. Salesforce requires 75% coverage.

What we did:

- Created WarrantyHandlerTest.cls.
- Inserted Product, Warranty, Service Request records.
- Asserted that Status = “Open”.
- Ran tests successfully (green check).

ExceptionExample.cls U WarrantyHandlerTest.cls U

force-app > main > default > classes > WarrantyHandlerTest.cls > ...

```apex
@isTest
public class WarrantyHandlerTest {

    @isTest static void testServiceRequestCreation() {
        // Create Product (add required fields)
        Product__c p = new Product__c(
            Name = 'Test Product',
            Serial_Number__c = 'SN-1001',
            Model_Number__c = 'M-123',   // if required
            Price__c = 5000              // if required
        );
        insert p;

        // Create Warranty (add required fields)
        Warranty__c w = new Warranty__c(
            Name = 'Test Warranty',
            Start_Date__c = Date.today(),
            End_Date__c = Date.today().addMonths(6),
            Product__c = p.Id,
            Warranty_Term__c = 12        // if required
        );
        insert w;

        // Create Service Request (picklist value must exist)
        Service_Request__c sr = new Service_Request__c(
            Name = 'Test SR',
            Product__c = p.Id,
            Related_Warranty__c = w.Id,
            Status__c = 'New' // ✅ replace with a real picklist value from your org
        );
        insert sr;

        // Verify
```

Do you want to ins

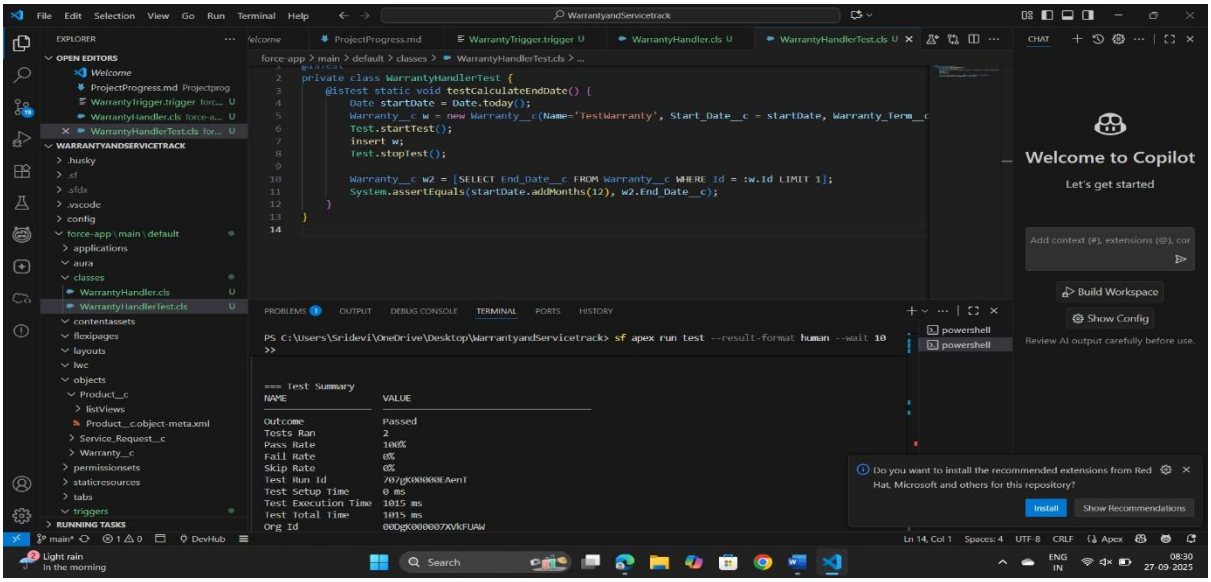
13. Asynchronous Processing

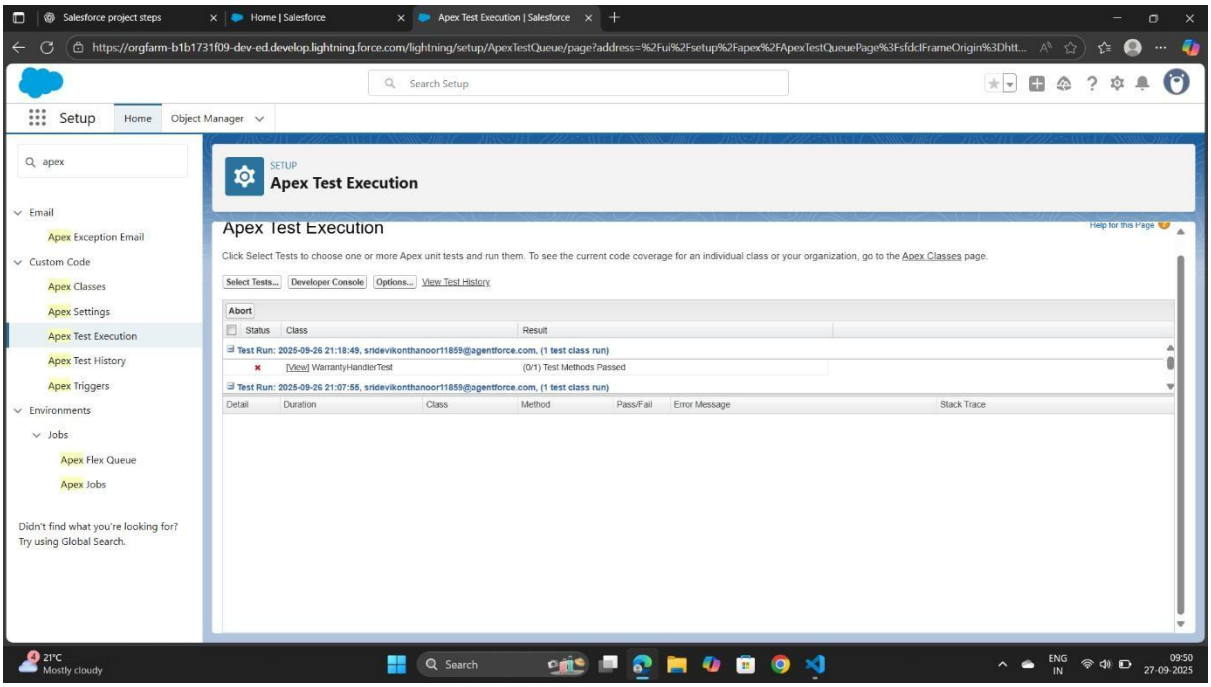
What it is:

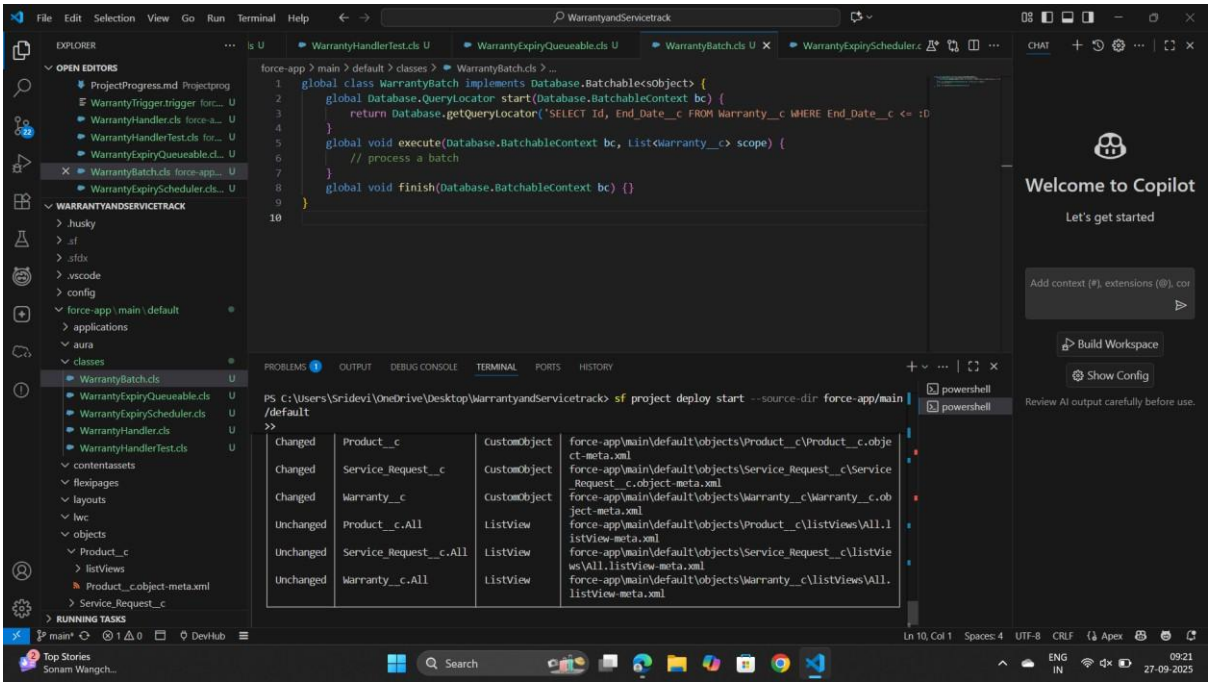
Background jobs: Queueable, Batch, Future, Scheduled.

What we did:

- Implemented Queueable + Scheduled.
- Verified execution in Apex Jobs page.




Salesforce project steps
Home | Salesforce
Apex Test Execution | Salesforce
Search Setup
Setup
Home
Object Manager
apex
Email
Apex Exception Email
Custom Code
Apex Classes
Apex Settings
Apex Test Execution
Apex Test History
Apex Triggers
Environments
Jobs
Apex Flex Queue
Apex Jobs
Didn't find what you're looking for?
Try using Global Search.
SETUP
Apex Test Execution
Apex Test Execution
Click Select Tests to choose one or more Apex unit tests and run them. To see the current code coverage for an individual class or your organization, go to the Apex Classes page.
Select Tests...
Developer Console
Options...
View Test History
Abort
Status
Class
Result
Test Run: 2025-09-26 21:18:49, sridevikonthanoor11859@agentforce.com, (1 test class run)
[View] WarrantyHandlerTest
(0/1) Test Methods Passed
Test Run: 2025-09-26 21:07:55, sridevikonthanoor11859@agentforce.com, (1 test class run)
Detail
Duration
Class
Method
Pass/Fail
Error Message
Stack Trace
21°C
Mostly cloudy
Search
ENG
IN
09:50
27-09-2025


File Edit Selection View Go Run Terminal Help
WarrantyandServicetrack
EXPLORER
OPEN EDITORS
ProjectProgress.md Projectprog
WarrantyTrigger.trigger
WarrantyHandler.cls
WarrantyHandlerTest.cls
WarrantyExpiryQueueable.cl...
WarrantyBatch.cls force-app...
WarrantyExpiryScheduler.cls...
WARRANTYANDSERVICETRACK
.husky
.sf
.sfdx
.vscode
config
force-app\main\default
applications
aura
classes
WarrantyBatch.cls
WarrantyExpiryQueueable.cls
WarrantyExpiryScheduler.cls
WarrantyHandler.cls
WarrantyHandlerTest.cls
contentassets
flexipages
layouts
lwc
objects
Product__c
listViews
Product__c.object-meta.xml
Service_Request__c
RUNNING TASKS
WarrantyHandlerTest.cls U
WarrantyExpiryQueueable.cls U
WarrantyBatch.cls U
WarrantyExpiryScheduler.c
force-app > main > default > classes > WarrantyBatch.cls > ...
global class WarrantyBatch implements Database.Batchable<sObject> {
global Database.QueryLocator start(Database.BatchableContext bc) {
return Database.getQueryLocator('SELECT Id, End_Date__c FROM Warranty__c WHERE End_Date__c <= :D
}
global void execute(Database.BatchableContext bc, List<Warranty__c> scope) {
// process a batch
}
global void finish(Database.BatchableContext bc) {}
}
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
HISTORY
PS C:\Users\Sridevi\OneDrive\Desktop\WarrantyandServicetrack> sf project deploy start --source-dir force-app/main/default
Changed Product__c CustomObject force-app\main\default\objects\Product__c\Product__c.object-meta.xml
Changed Service_Request__c CustomObject force-app\main\default\objects\Service_Request__c\Service_Request__c.object-meta.xml
Changed Warranty__c CustomObject force-app\main\default\objects\Warranty__c\Warranty__c.object-meta.xml
Unchanged Product__c.All ListView force-app\main\default\objects\Product__c\listViews\All.listView-meta.xml
Unchanged Service_Request__c.All ListView force-app\main\default\objects\Service_Request__c\listViews\All.listView-meta.xml
Unchanged Warranty__c.All ListView force-app\main\default\objects\Warranty__c\listViews\All.listView-meta.xml
powershell
powershell
CHAT
Welcome to Copilot
Let's get started
Add context (#), extensions (@), cor
Build Workspace
Show Config
Review AI output carefully before use.
main*
DevHub
Ln 10, Col 1
Spaces: 4
UTF-8
CRLF
Apex
Top Stories
Sonam Wangch...
Search
ENG
IN
09:21
27-09-2025

Phase 6: User Interface Development

1. Lightning App Builder

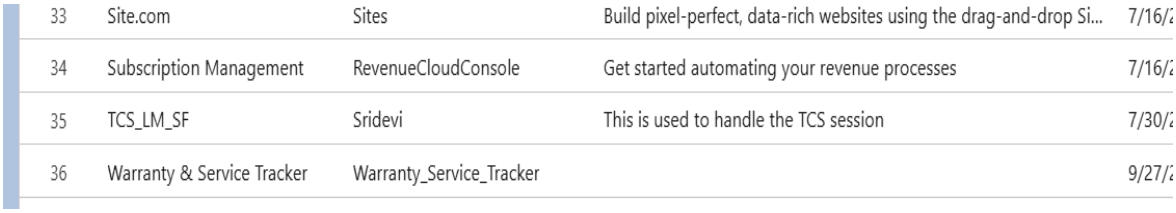
- **What we did:**

  Used Lightning App Builder to customize the app experience. Created a custom **Warranty & Service Tracker App** that includes Products, Warranties, and Service Requests in the navigation bar.

- **Steps:**

  1. Setup → App Manager → New Lightning App.
  2. Gave app name: *Warranty & Service Tracker*.
  3. Added tabs: Products, Warranties, Service Requests.
  4. Assigned to all profiles for visibility.

- **Screenshot:** App Manager showing new app.



33	Site.com	Sites	Build pixel-perfect, data-rich websites using the drag-and-drop Si...	7/16/
34	Subscription Management	RevenueCloudConsole	Get started automating your revenue processes	7/16/
35	TCS_LM_SF	Sridevi	This is used to handle the TCS session	7/30/
36	Warranty & Service Tracker	Warranty_Service_Tracker		9/27/

2. Record Pages

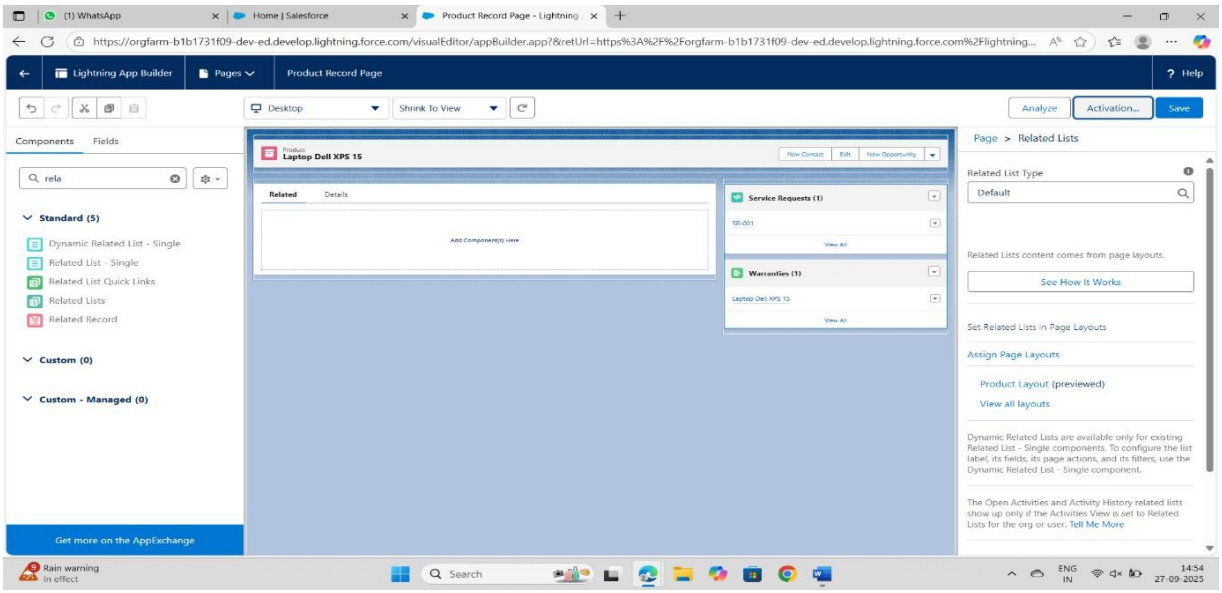
- **What we did:**

  Designed custom **record pages** for Product, Warranty, and Service Request using Lightning App Builder.

- **Steps:**

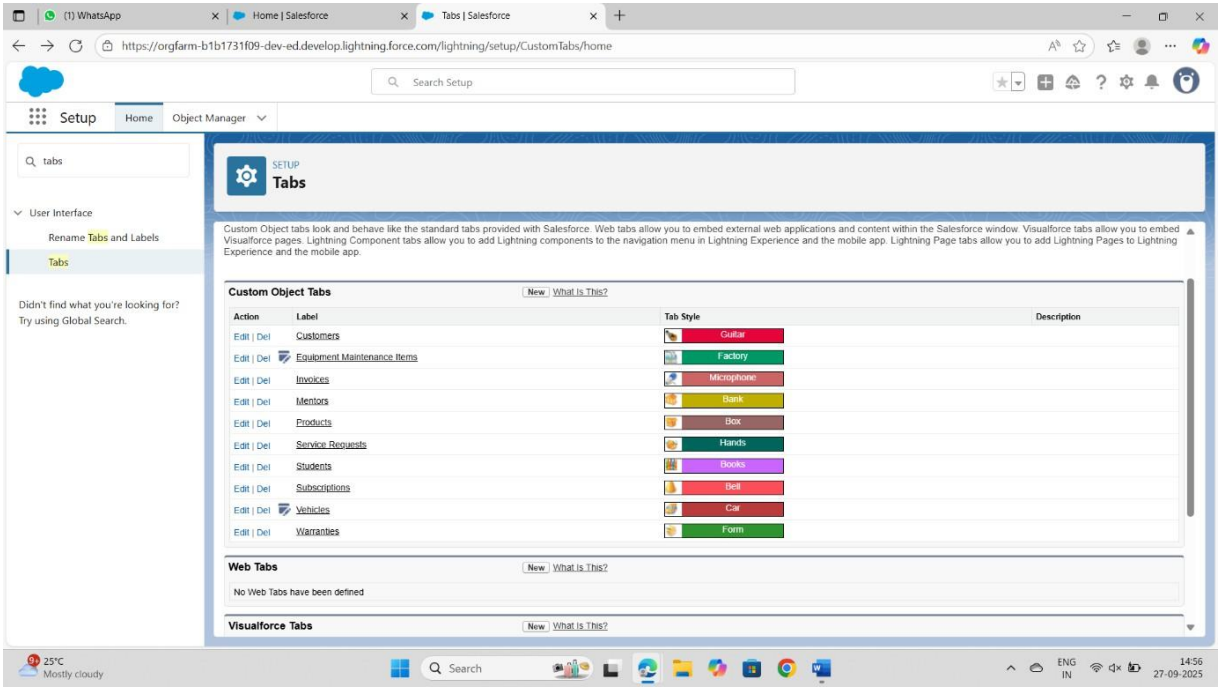
  1. Setup → Object Manager → Product_c → Lightning Record Pages.
  2. Created a new Record Page → added Highlights Panel, Related Lists, and Tabs.
  3. Assigned as Org Default.

- **Screenshot:** Product record page editor.



3. Tabs

- **What we did:**
  Added **Custom Tabs** for each custom object.
- **Steps:**
  1. Setup → Tabs → New.
  2. Chose Object = Product_c, Warranty_c, Service_Request_c.
  3. Selected tab style (icon & color).
  4. Added them to the Warranty & Service Tracker App.



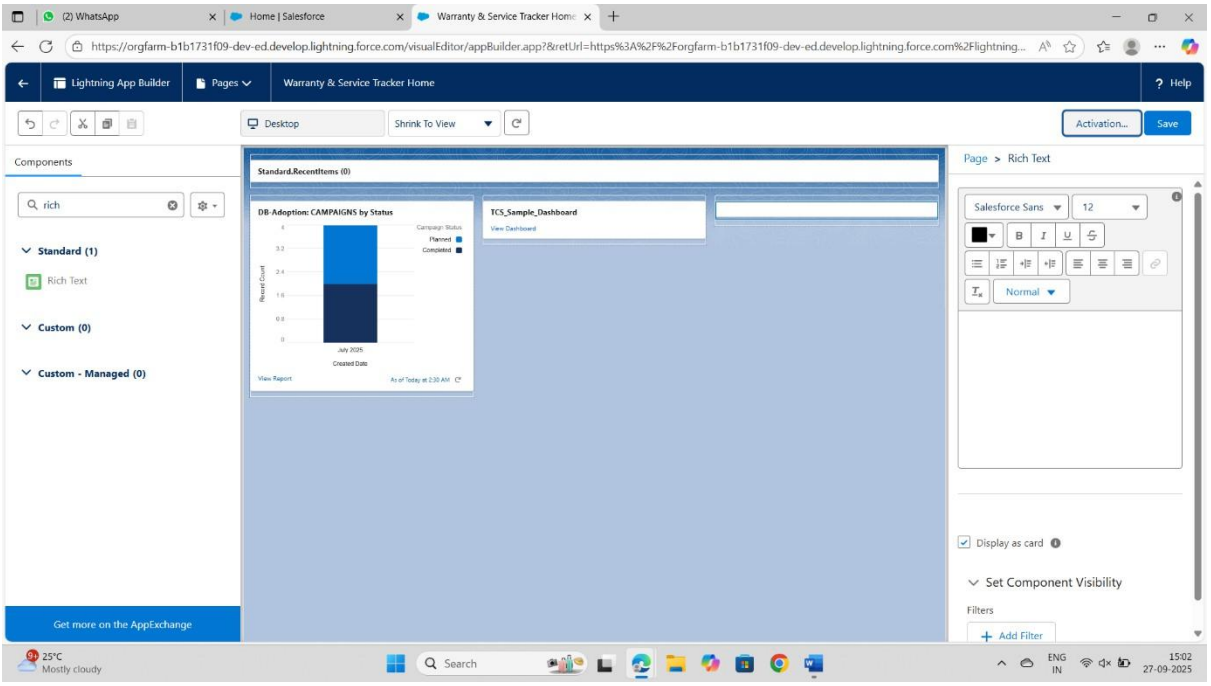
4. Home Page Layouts

- **What we did:**

  Customized Home Page to show dashboard components and quick links.

- **Steps:**

  1. **Setup → Lightning App Builder → Home Page → New.**
  2. **Dragged standard components like Recent Items, Reports, and Dashboard Snapshot.**
  3. **Activated for Warranty & Service Tracker App.**



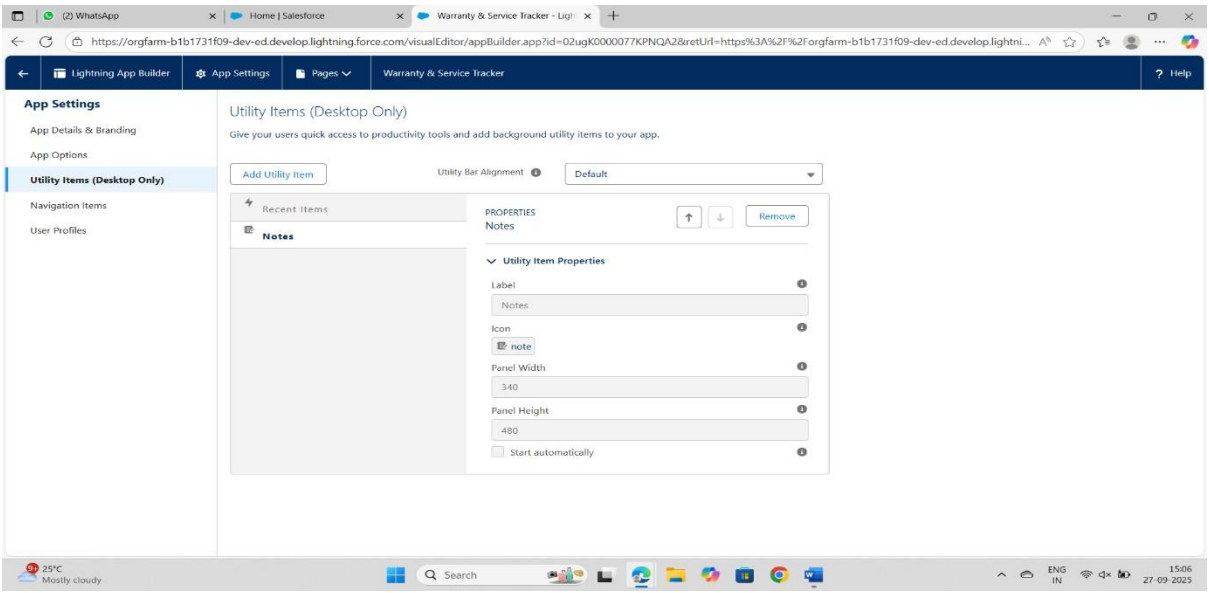
5. Utility Bar

- **What we did:**

  Added a **Utility Bar** for quick access to Notes and Recent Items.

- **Steps:**

  1. Setup → App Manager → Edit App → Utility Bar.
  2. Added “Notes” and “History”.



6. Lightning Web Components (LWC)

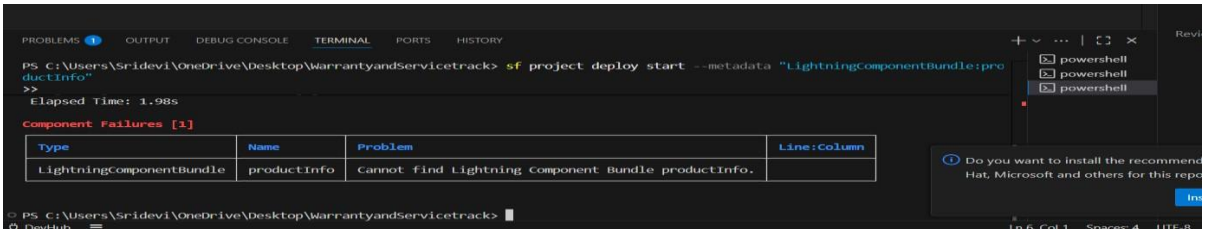
- **What we did:**
  Created a sample LWC to display Product details.
- **Code:**

```
import { LightningElement, api } from 'lwc';
export default class ProductInfo extends LightningElement {
    @api recordId;
}
```

```
<template>
    <lightning-card title="Product Info">
        <p>Product Record Id: {recordId}</p>
    </lightning-card>
</template>
```

- **Deploy:**

```
sf project deploy start --metadata "LightningComponentBundle:productInfo"
```



7. Apex with LWC

- **What we did:**
  Connected LWC to Apex to fetch Products.

  Product.js

  public with sharing class ProductController {

  @AuraEnabled(cacheable=true)

  public static List<Product_c> getProducts() {

  return [SELECT Id, Name, Serial_Number_c, Price_c FROM Product_c LIMIT 5];

  }

  }

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

export default class ProductList extends LightningElement {
   @wire(getProducts) products;
}
```

productList.html

```
<template>
   <lightning-card title="Products">
      <template if:true={products.data}>
         <template for:each={products.data} for:item="p">
            <p key={p.Id}>{p.Name} - {p.Serial_Number_c} - {p.Price_c}</p>
         </template>
      </template>
   </lightning-card>
</template>
```

- **Screenshot:** Product list displayed on page.

8. Events in LWC

- **What we did:**
  Implemented simple child → parent event communication.
- **Code:**
  Child Component → dispatch event.
- this.dispatchEvent(new CustomEvent('notify', { detail: 'Hello from child' }));

Parent Component → handle event.

<c-child onnotify={handleMessage}></c-child>

9. Wire Adapters

- **What we did:**
  Used @wire with UI API.

```
import { LightningElement, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';
import NAME_FIELD from '@salesforce/schema/Product_c.Name';

export default class ProductWire extends LightningElement {
   @wire(getRecord, { recordId: 'a01XX00000XXXX', fields: [NAME_FIELD] })
   product;
}
```

10. Imperative Apex Calls

- **What we did:**
  Called Apex explicitly from JS.

```
import getProducts from '@salesforce/apex/ProductController.getProducts';
handleClick() {
   getProducts()
      .then(result => { this.products = result; })
    .catch(error => { this.error = error; });
}
```

11. Navigation Service

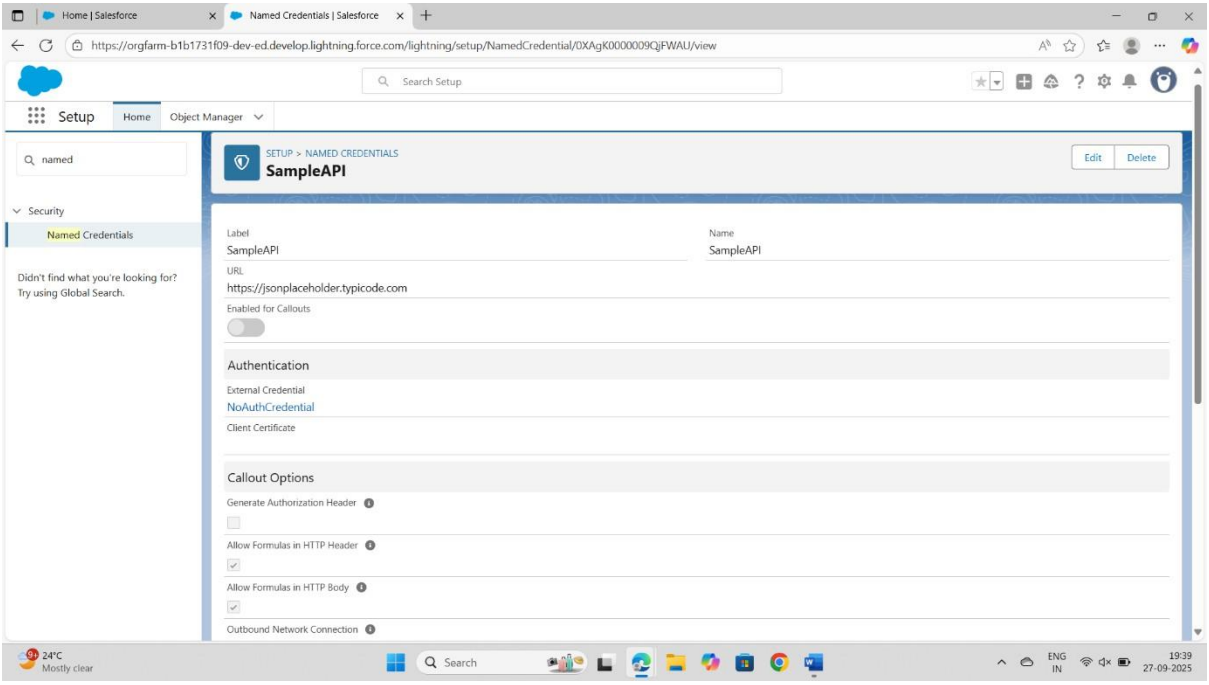
- **What we did:**
  Used NavigationMixin to open record pages.

```
import { NavigationMixin } from 'lightning/navigation';
export default class ProductNavigator extends NavigationMixin(LightningElement) {
    navigateToProduct() {
     this[NavigationMixin.Navigate]({
       type: 'standard_recordPage',
  attributes: {
         recordId: 'a01XX00000XXXX',
          objectApiName: 'Product_c',
          actionName: 'view'
         }
       });
  }
}
```

Phase 7: Integration & External Access

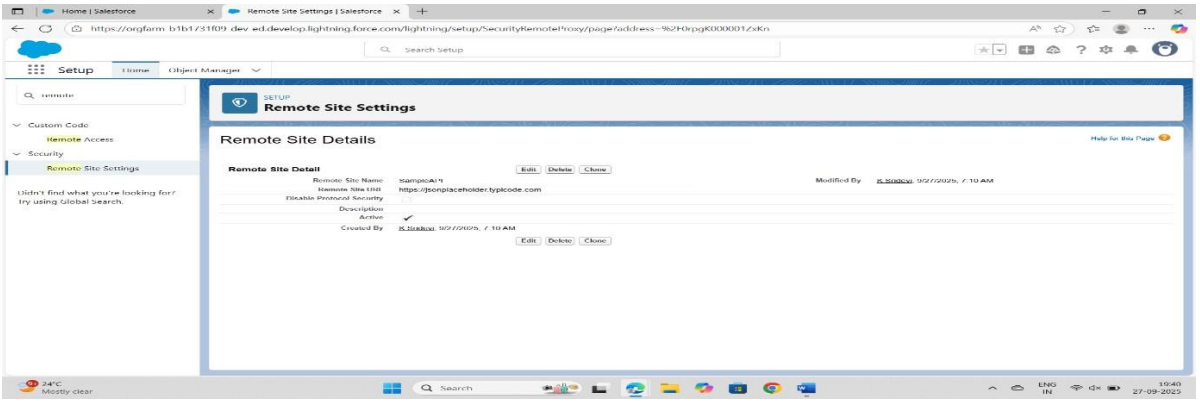
1. Named Credentials

- Setup → Named Credentials → New.
- Label: SampleAPI
- URL: https://jsonplaceholder.typicode.com
- Identity Type: Anonymous.
- Authentication Protocol: No Authentication.



2. Remote Site Settings

- Setup → Remote Site Settings → New.
- Label: SampleAPI
- URL: https://jsonplaceholder.typicode.com



3. Apex REST Callout

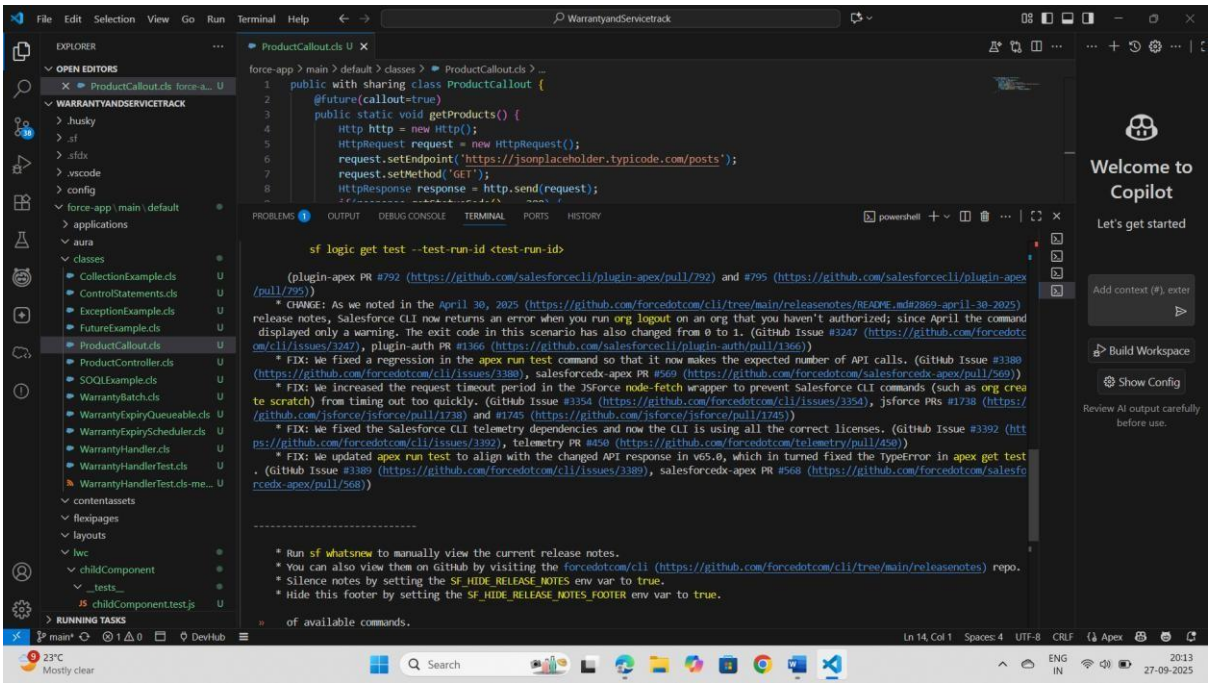
In VS Code → create ProductCallout.cls

```apex
public with sharing class ProductCallout {
    @future(callout=true)
    public static void getProducts() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://jsonplaceholder.typicode.com/posts');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        if(response.getStatusCode() == 200) {
            System.debug('Response: ' + response.getBody());
        }
    }
}
```

4. Run Callout

Open Developer Console → Execute Anonymous:

ProductCallout.getProducts();

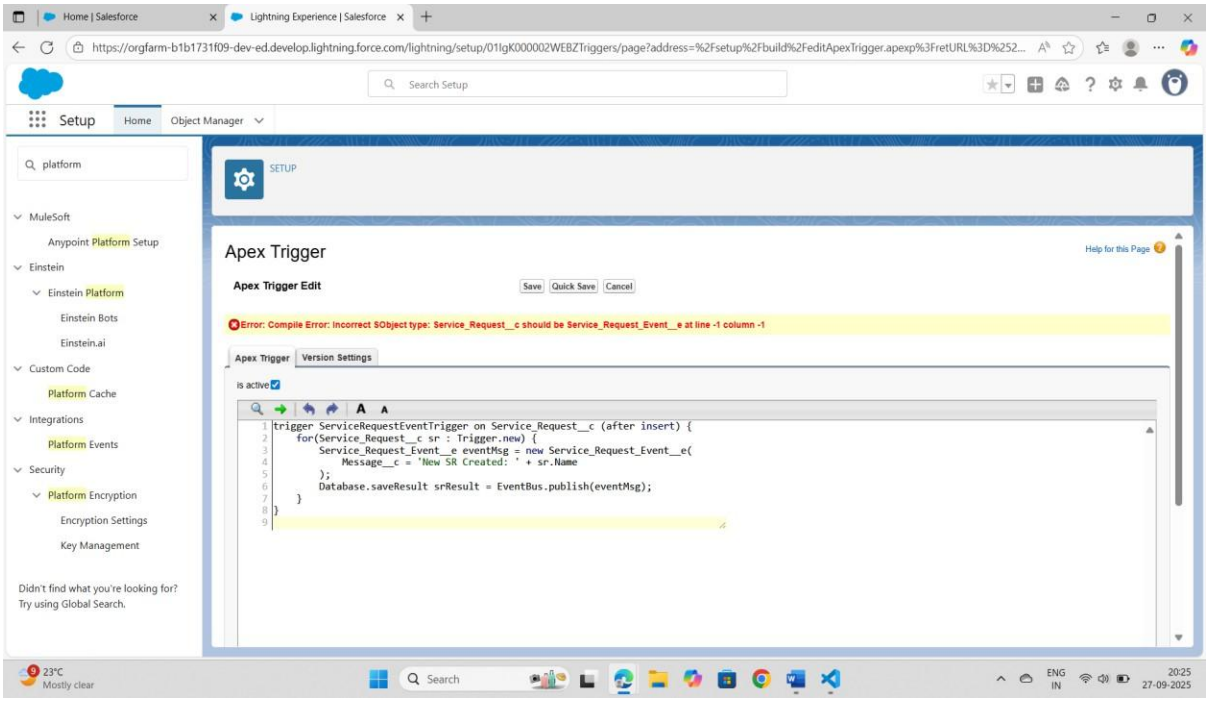


5. Platform Event

- Setup → Platform Events → New.
- Label: Service_Request_Event

- Field: Message_c (Text).
- Add Trigger: When Service_Request_c is created, publish event.

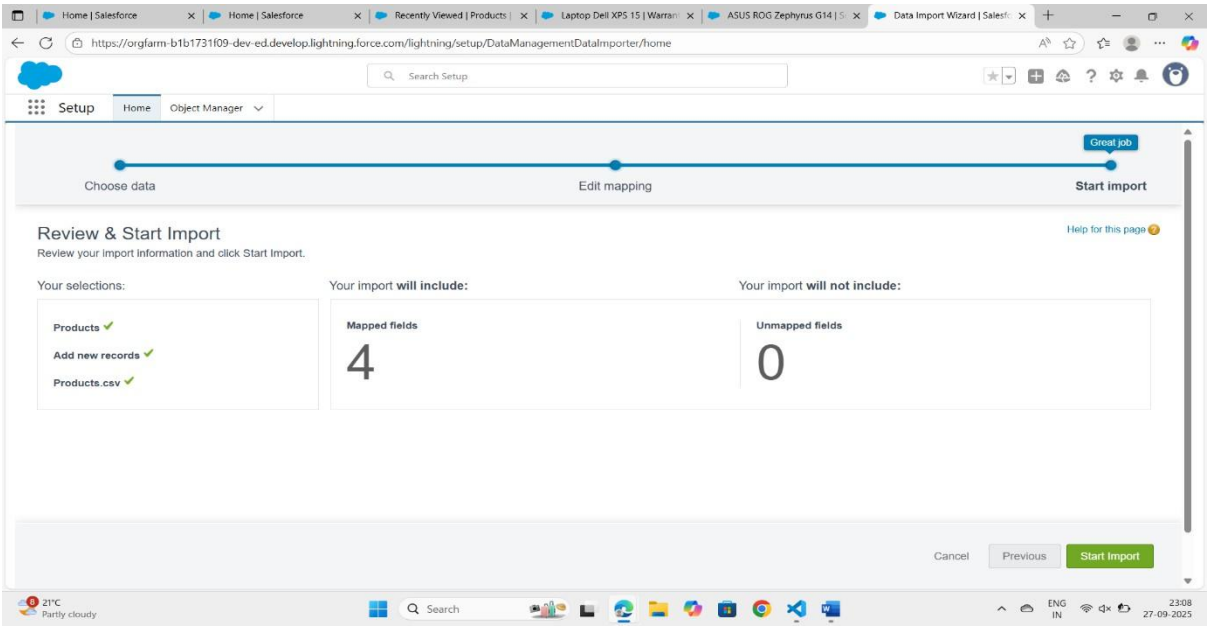
```
trigger ServiceRequestEventTrigger on Service_Request_c (after insert) {
    for(Service_Request_c sr : Trigger.new) {
        Service_Request_Event_e eventMsg = new Service_Request_Event_e( Message
            c = 'New SR Created: ' + sr.Name
        );
        Database.saveResult srResult = EventBus.publish(eventMsg);
    }
}
```



Phase 8: Data Management & Deployment

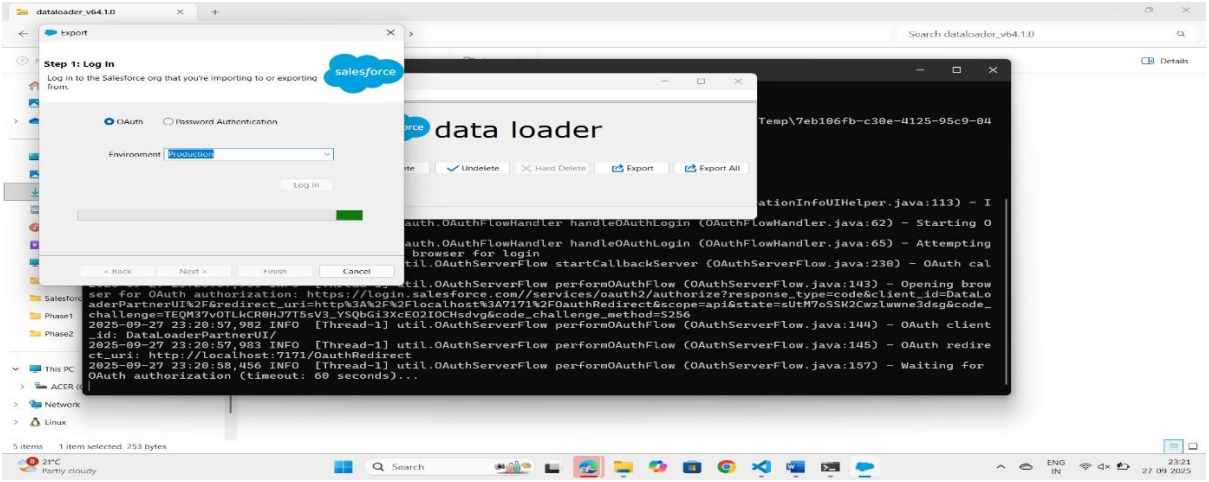
1. Data Import Wizard

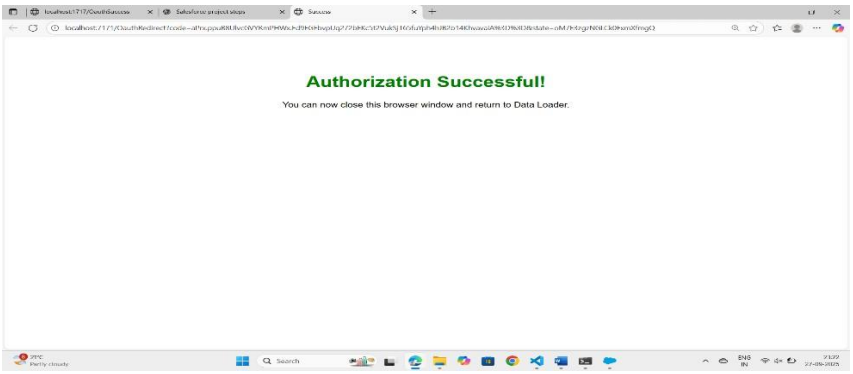
- App Launcher → Data Import Wizard.
- Choose Product_c.
- Upload CSV (Name, Serial_Number_c, Model_c, Price_c).
- Start Import.

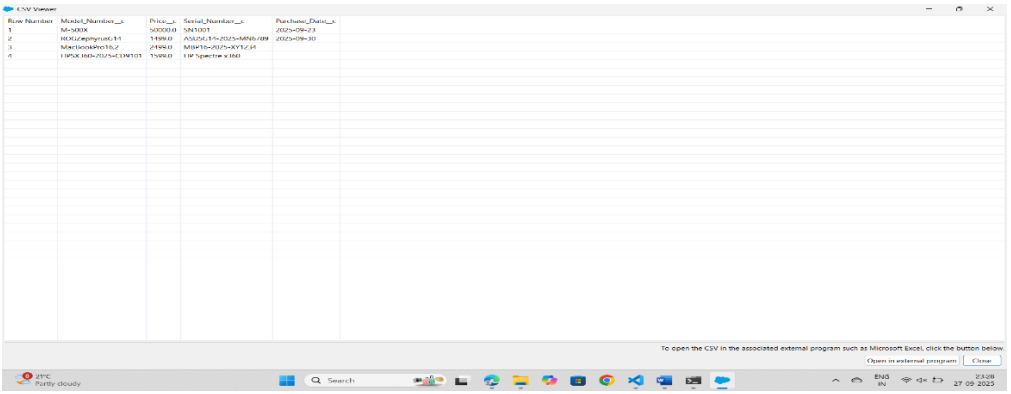


2. Data Loader

- Download Data Loader (if not installed).
- Login → Choose “Insert” → Select Service_Request_c.
- Map CSV columns to fields → Finish.

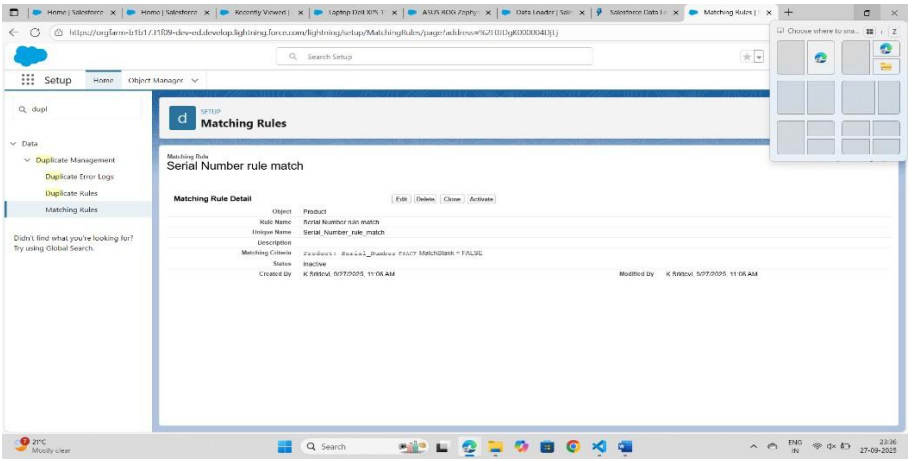






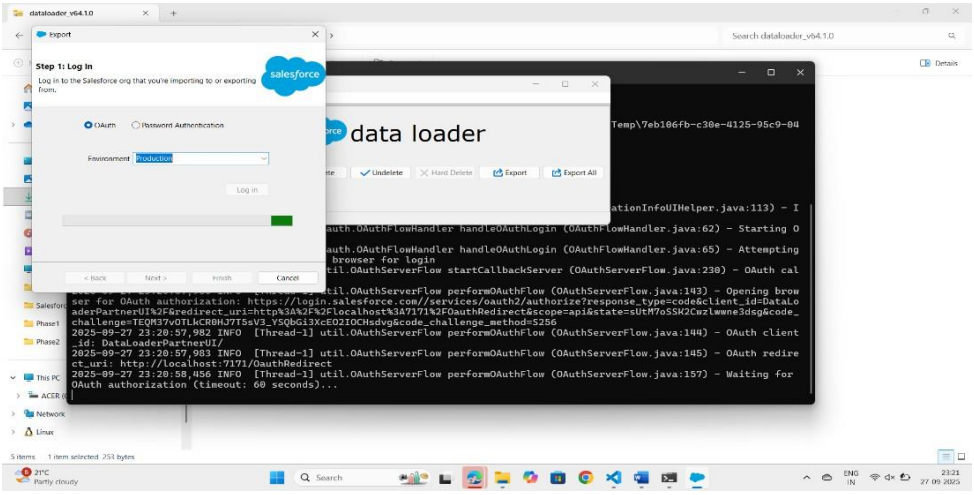
3. Duplicate Rules

- Setup → Duplicate Rules → New Rule → Object: Product_c.
- Matching Rule: Serial_Number_c must be unique.
- Action: Block on Create/Edit.



4. Data Export

- Setup → Data Export → Schedule Weekly Export.
- Include Products, Warranties, Service Requests.

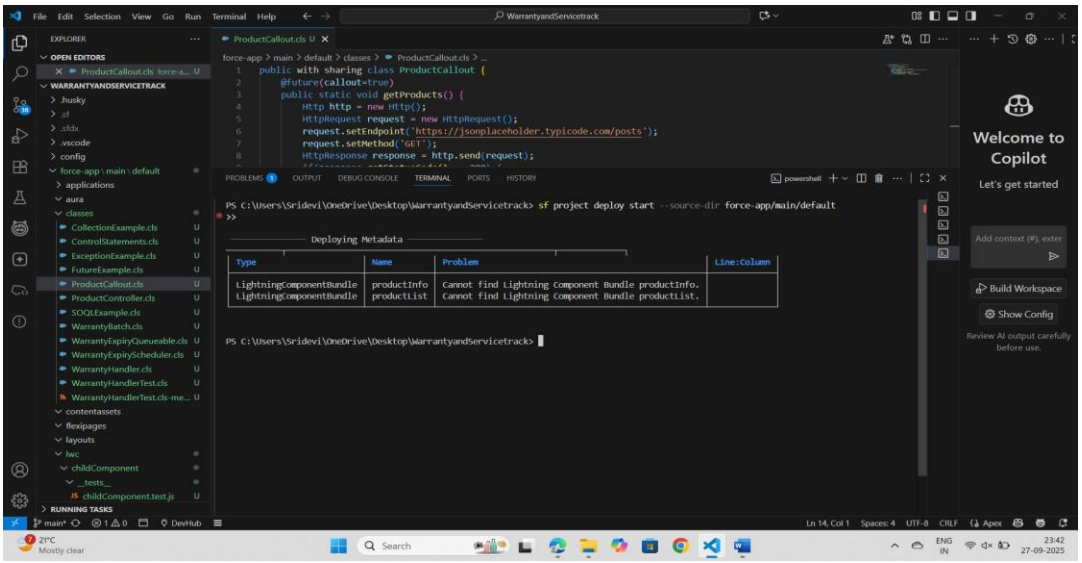


5. Change Sets

- Setup → Outbound Change Sets → New.
- Add Components (Product_c, Warranty_c, Service_Request_c).
- Upload to Production org.

6. VS Code + SFDX Deploy

- In terminal:
- sf project deploy start --source-dir force-app/main/default

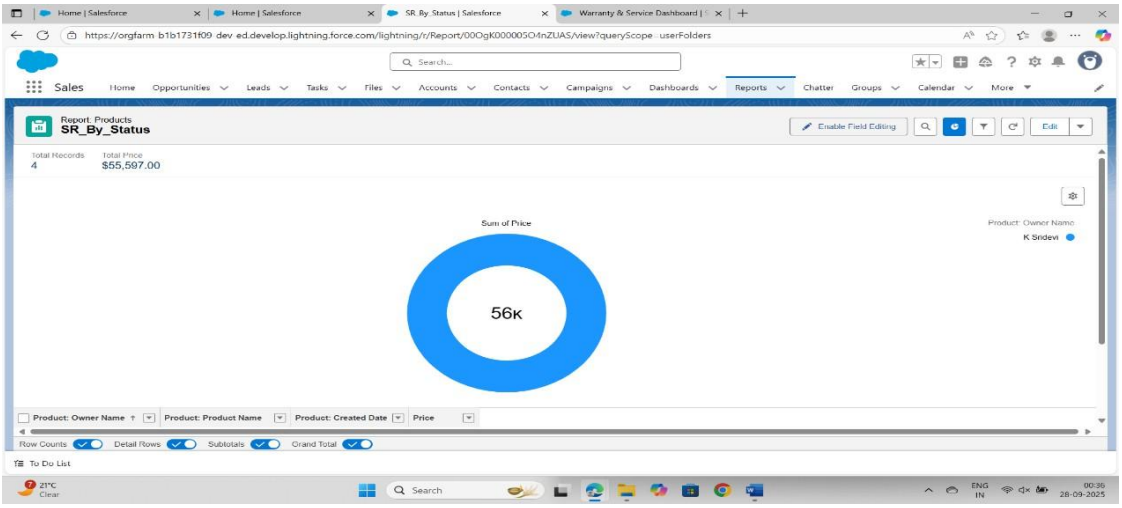


- Confirms metadata successfully deploy

Phase 9 — Reporting, Dashboards & Security Review

1) Create Report 1 — *Service Requests by Status*

1. App Launcher → Reports → New Report.
2. In the New Report modal type Service Requests and select the Service Request report type → Start Report.
3. In the report builder:
    - Remove unnecessary columns; add columns: Name, Status, Product, Created Date, Owner.
    - Drag Status into the Group Rows area (left).
    - Add filter: Created Date = All Time (or last 90 days for demo).
    - Add chart: click the chart icon → choose Pie → slice by Status.
4. Save → Report Name: SR_By_Status → Folder: Public Reports → Save & Run.

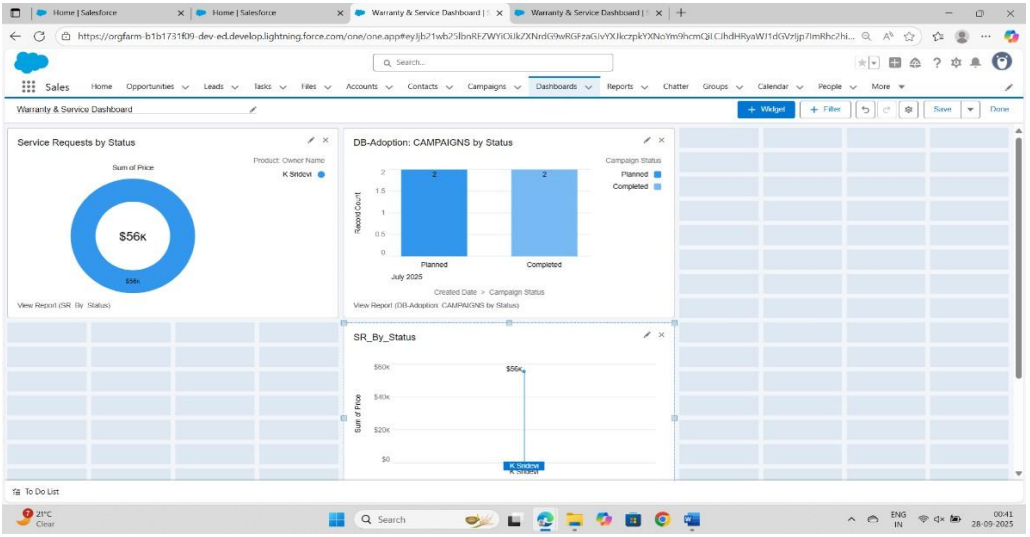


2) Create Report 2 — *Warranties Expiring Soon*

1. App Launcher → Reports → **New Report** → search **Warranties** (or your Warranty object) → **Start Report**.
2. In builder:
    - Columns: Name, Product, End Date, Start Date, Warranty Term.
    - Filter: End Date → set filter to Next 30 Days (or <= NEXT 30 DAYS).
    - Add a **Bar** chart showing count by Product or by Days to Expiry.

4) Build Dashboard

1. App Launcher → Dashboards → **New Dashboard** → Name: Warranty & Service Dashboard → Folder: Private Reports (or Public).
2. Add components:
    - Component 1: Source SR_By_Status → Display: **Pie** → Title: Service Requests by Status.
    - Component 2: Source Warranties_ExpiringSoon → Display: **Bar/Column** → Title: Warranties Expiring (30d).
    - Component 3: Create a report Open SR Trend (Summary by Created Date) and add as Line chart.
3. Arrange components and **Save**.
4. (Optional) Dashboard Properties → **View Dashboard As** → choose **The person viewing the dashboard** to enable dynamic dashboard.



5) Sharing & Security Review (quick checklist + steps)

Complete and capture each of these:

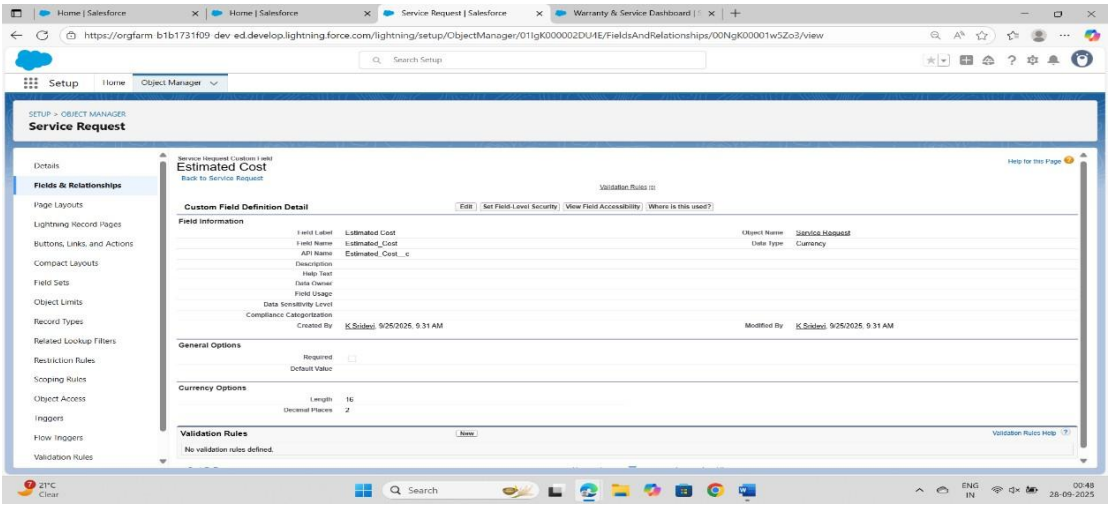
A — Organization-Wide Defaults (OWD)

- Setup → Quick Find → **Sharing Settings**.
- Ensure Service Requests = **Private** (or as project requires).
- Note any other object settings (Accounts = Public Read/Write, Contacts = Controlled by Parent).

User Presence	Public Read Only	Private	✓
User Provisioning Request	Private	Private	✓
Waitlist	Private	Private	✓
Web Cart Document	Private	Private	✓
Work Order	Private	Private	✓
Work Plan	Private	Private	✓
Work Plan Template	Private	Private	✓
Work Step Template	Private	Private	✓
Work Type	Private	Private	✓
Work Type Group	Public Read/Write	Private	✓
Customer	Public Read/Write	Private	✓
Equipment Maintenance Item	Controlled by Parent	Controlled by Parent	
Invoice	Public Read/Write	Private	✓
Mentor	Public Read/Write	Private	✓
Product	Public Read/Write	Private	✓
Service Request	Public Read/Write	Private	✓
Student	Controlled by Parent	Controlled by Parent	
Subscription	Public Read/Write	Private	✓
Vehicle	Public Read/Write	Private	✓
Warranty	Public Read/Write	Private	✓

B — Field Level Security (FLS)

- Setup → Object Manager → Service_Request_c → Fields & Relationships → click Estimated_Cost_c (or field to hide).
- Click **Set Field-Level Security** → uncheck **Visible** for PS_Service_Agent profile → Save.



Screenshot: Field Level Security page. File: Phase9_FLS_EstimatedCost.png

C— Profiles & Permission Sets

- Setup → Profiles → open PS_Service_Agent, PS_Manager → confirm Tab Settings, Object Permissions.

Fulfillment Orders	Tab Hidden	Site.com	Default On
olocation Based Actions	Tab Hidden	Stores	Tab Hidden
Groups	Default On	Streaming Channels	Tab Hidden
Home	Tab Hidden	Subscriptions	Tab Hidden
Home	Default On	Tasks	Default On
Ideas	Default On	Territory Account Product Message Scores	Tab Hidden
Idea Themes	Tab Hidden	Territory Account Scores	Tab Hidden
Identity Documents	Tab Hidden	Today	Default On
Identity Resolutions	Default On	Unstructured Data	Default On
Images	Tab Hidden	User Provisioning Requests	Tab Hidden
Individuals	Tab Hidden	Voice Calls	Tab Hidden
tory Count Assessments	Tab Hidden	Waitlists	Default Off
ventory Count Plan Items	Tab Hidden	Web Store Inventory Sources	Tab Hidden
unt Product Batch Items	Tab Hidden	Work Type Groups	Tab Hidden
ntory Item Reservations	Tab Hidden	Work Types	Tab Hidden
Inventory Operations	Tab Hidden		
Create Default Data	Tab Hidden	Service Requests	Default On
Customers	Default On	Students	Default On
ment Maintenance Items	Tab Hidden	Subscriptions	Default On
Invoices	Default On	Vehicles	Tab Hidden
Mentors	Default On	Warranties	Default On
Products	Default On		

- Setup → Permission Sets → create PS_ExportReports (if needed) → assign to Manager.

D — Session Settings & Login IP Ranges

- Setup → Session Settings → note timeout, secure settings.
- Profiles → Login IP Ranges (optional): set ranges for Manager if required (Profile → Login IP Ranges).

E — View Setup Audit Trail

- Setup → Quick Find → **View Setup Audit Trail** → export last 6 months (if needed) → include in documentation.