

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

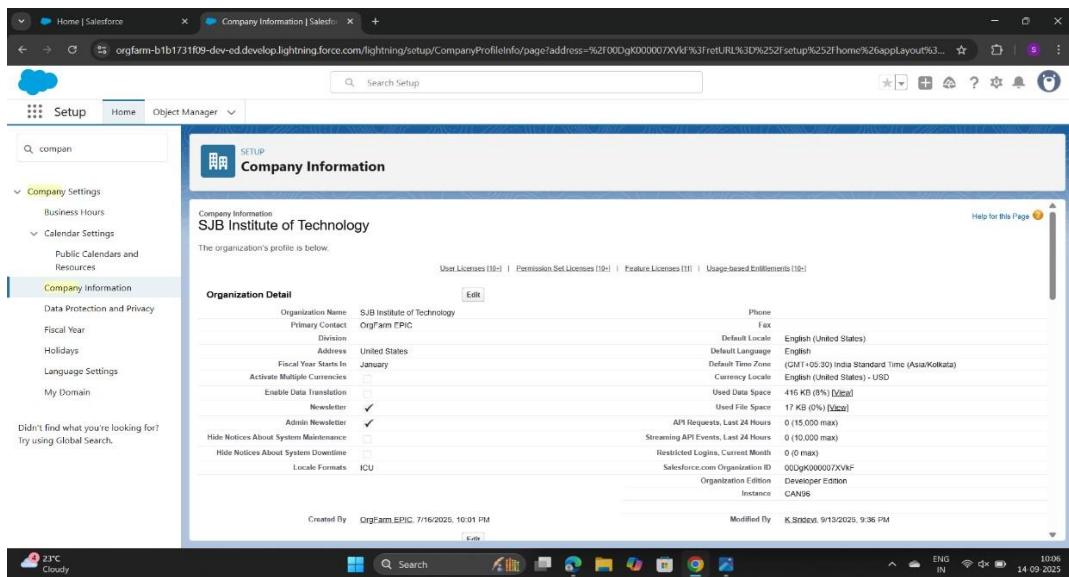
Org Setup & Configuration

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.

Business Hours Detail

Business Hours Name	Default Business Hours	Time Zone
Business Hours	Sunday: No Hours Monday: 9:00 AM to 6:00 PM Tuesday: 9:00 AM to 6:00 PM Wednesday: 9:00 AM to 6:00 PM Thursday: 9:00 AM to 6:00 PM Friday: 9:00 AM to 6:00 PM Saturday: No Hours	(GMT+05:30) India Standard Time (Asia/Kolkata)

Holidays

Add/Remove	
No records to display	

Holiday Detail

Holiday Name	Description	Business Hours
Diwali	Date and Time: 9/13/2025 All Day	Business Hours Help

Business Hours

Add/Remove	
No records to display	

4. Fiscal Year Settings

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

The screenshot shows the Salesforce Setup interface under the 'Fiscal Year' section. A modal window titled 'Change Fiscal Year Period' is displayed, prompting the user to change the fiscal year period. The 'Fiscal Year Start Month' dropdown is set to 'January'. The 'Fiscal Year is Based On' section has 'The ending month' selected. A warning message at the top of the modal states: 'Changing the fiscal year shifts fiscal periods and impacts opportunities and forecasts across your organization. If your forecast periods are set to quarterly, adjusting the fiscal year start month will erase existing forecast adjustments and quotas. Consider exporting a data backup before implementing this change.' Buttons for 'Save' and 'Cancel' are at the bottom of the modal.

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).

The screenshot shows the Salesforce Setup interface under the 'Users' section. A user named 'Warranty Admin' is selected, showing their details. The 'User Detail' section includes fields like Name (Warranty Admin), Alias (warrantyadmin), Email (warranty.admin@example.com), Username (warranty.admin@example.com), Nickname (User17570239507637239044), Title (Warranty Administrator), Company (SJT College), Department (Warranty & Services), Division (Support), Address, Time Zone (GMT+05:30) India Standard Time (Asia/Kolkata), Locale (English (United States)), Language (English), Delegated Approver, Manager, Receive Approval Request Emails (Only if I am an approver), Federation ID, and Federation URL. To the right, a 'Role' section lists 'Warranty Admin' and 'System Administrator' with checkboxes for Marketing User, Offline User, Knowledge User, Flow User, Service Cloud User, Site.com Contributor User, Site.com Publisher User, WDC User, Mobile Push Registrations, Data.com User Type (View), Accessibility Mode (Classic Only), Debug Mode, and High-Contrast Palette on Charts. A note at the bottom states: 'For Distribution One-Time Download'.

6. Profiles

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

PS_Service_Agent

Name: PS_Service_Agent
User License: Salesforce Platform
Description: Created By: K.Sridayi, 9/13/2025, 10:31 PM
Modified By: K.Sridayi, 9/13/2025, 10:31 PM

Page Layouts

Standard Object Layouts	Global	Lead
Global	Global Layout [View Assignment]	Lead Layout [View Assignment]
Email Application	Not Assigned [View Assignment]	Location Layout [View Assignment]
Home Page Layout	Home Page Default [View Assignment]	Location Group Layout [View Assignment]
Account	Account Layout [View Assignment]	Location Group Assignment Layout [View Assignment]
Alternative Payment Method	Alternative Payment Method Layout [View Assignment]	Object Milestone Layout [View Assignment]

PS_Manager

Name: PS_Manager
User License: Salesforce Platform
Description: Created By: K.Sridayi, 9/13/2025, 10:32 PM
Modified By: K.Sridayi, 9/13/2025, 10:32 PM

Page Layouts

Standard Object Layouts	Global	Lead
Global	Global Layout [View Assignment]	Lead Layout [View Assignment]
Email Application	Not Assigned [View Assignment]	Location Layout [View Assignment]
Home Page Layout	Home Page Default [View Assignment]	Location Group Layout [View Assignment]
Account	Account Layout [View Assignment]	Location Group Assignment Layout [View Assignment]
Alternative Payment Method	Alternative Payment Method Layout [View Assignment]	Object Milestone Layout [View Assignment]

7. Roles

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

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

The screenshot shows the 'Roles' page in the Salesforce Setup. The tree structure includes:

- Add Role
- VP_Marketing (Edit | Del | Assign)
 - Add Role
 - Marketing_Team (Edit | Del | Assign)
 - Add Role
- VP_North_American_Sales (Edit | Del | Assign)
 - Add Role
 - Director_Channel_Sales (Edit | Del | Assign)
 - Add Role
 - Channel_Sales_Team (Edit | Del | Assign)
 - Add Role
- Director_Direct_Sales (Edit | Del | Assign)
 - Add Role
 - Eastern_Sales_Team (Edit | Del | Assign)
 - Add Role
- Western_Sales_Team (Edit | Del | Assign)
 - Add Role

- Test_Role (Edit | Del | Assign)
- Add Role
 - Test_Role1 (Edit | Del | Assign)
 - Add Role
- Manager (Edit | Del | Assign)
- Add Role
- Warranty_Admin (Edit | Del | Assign)
- Add Role
- Service_Agent (Edit | Del | Assign)
- Add Role

8. Permission Sets

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

The screenshot shows the 'Permission Sets' page in the Salesforce Setup. The table lists the following permission sets:

Action	Permission Set Name	Description	License
<input type="checkbox"/>	PS_ExportReports	Set up Partner Connect from a partner org. Partner Connect is a Part...	Salesforce
<input type="checkbox"/>	Partner_Connect_Partner_Admin_Setup	Has all the user permissions to gate access to APIs that are available...	Salesforce Payments Internal
<input type="checkbox"/>	Payments_Administrator	Manage prompt templates using Prompt Builder and run them using ...	Einstein Prompt Templates
<input type="checkbox"/>	Prompt_Template_Manager	Run prompt templates using generative AI features.	Einstein Prompt Templates
<input type="checkbox"/>	Prompt_Template_User		
<input type="checkbox"/>	PromptTemplatePermSet		Cloud Integration User
<input type="checkbox"/>	Publish_Suggested_for_You_Nudges_Integration_User	Access the Core Adoption Service and tenant orgs, which are used to p...	Cloud Integration 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.

Object	Default Internal Access	Default External Access	Grant Access Using Hierarchies
Lead	Public Read/Write/Transfer	Private	✓
Account and Contract	Public Read/Write	Private	✓
Contact	Controlled by Parent	Controlled by Parent	✓
Order	Controlled by Parent	Controlled by Parent	✓
Asset	Controlled by Parent	Controlled by Parent	✓
Opportunity	Public Read/Write	Private	✓
Case	Private	Private	✓
Campaign	Public Full Access	Private	✓
Campaign Member	Controlled by Campaign	Controlled by Campaign	✓
User	Public Read Only	Private	✓
Activity	Private	Private	✓

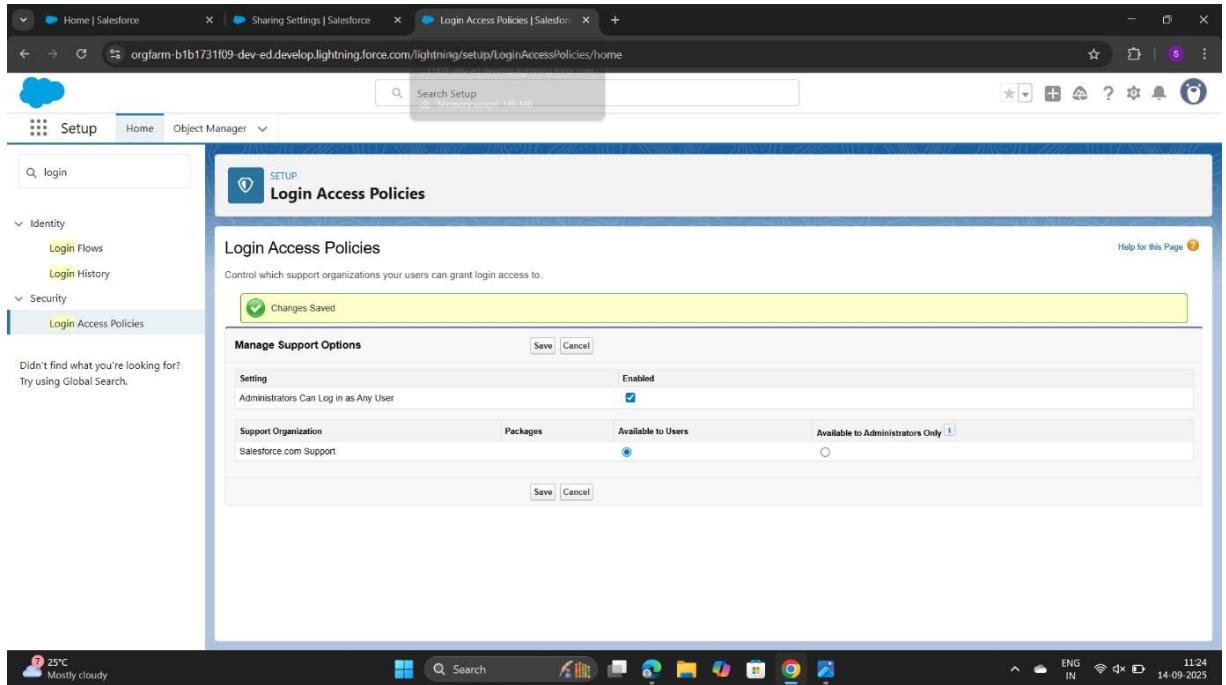
10. Sharing Rules

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

Action	Criteria	Shared With	Case
Edit Del	Owner in Role: Service Agent	Role: Manager	Read Only

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:
 - o Product_c
 - o Warranty_c
 - o Service_Request_c
- (You already created these).

The screenshot shows the Salesforce Object Manager interface. The search bar at the top right contains "product". The main table lists various objects with their API names, types, descriptions, last modified date, and deployment status. The row for "Product" (API name: Product_c) is highlighted, showing it was last modified on 9/13/2025 and is deployed.

LABEL	API NAME	TYPE	DESCRIPTION	LAST MODIFIED	DEPLOYED
Fulfillment Order Product	FulfillmentOrderLineitem	Standard Object			
Opportunity Product	OpportunityLineItem	Standard Object			
Order Product	OrderItem	Standard Object			
Product	Product_c	Custom Object		9/13/2025	✓
Product Attribute	ProductAttribute	Standard Object			
Product Attribute Set Product	ProductAttributeSetProduct	Standard Object			
Product Category Product	ProductCategoryProduct	Standard Object			
Product Consumption Schedule	ProductConsumptionSchedule	Standard Object			

The screenshot shows the Salesforce Object Manager interface with a search bar containing "warr". The main table lists objects, and the row for "Warranty" (API name: Warranty_c) is highlighted, indicating it was last modified on 9/13/2025 and is deployed.

LABEL	API NAME	TYPE	DESCRIPTION	LAST MODIFIED	DEPLOYED
Warranty	Warranty_c	Custom Object		9/13/2025	✓

The screenshot shows the Salesforce Object Manager interface. At the top, there's a search bar with the placeholder "Search Setup". Below it, a navigation bar includes "SETUP", "Home", and "Object Manager". The main area is titled "Object Manager" with a sub-header "1 items. Sorted by Label". A table lists one item:

LABEL	API NAME	TYPE	DESCRIPTION	LAST MODIFIED	DEPLOYED
Service Request	Service_Request__c	Custom Object		9/13/2025	<input checked="" type="checkbox"/>

❖ 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

The screenshot shows the Salesforce Object Manager interface for the "Product" object. The left sidebar has a "Fields & Relationships" section with various options like Page Layouts, Lightning Record Pages, etc. The main area is titled "Fields & Relationships" with a sub-header "8 items. Sorted by Field Label". A table lists the fields:

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedBy	Lookup(User)		<input checked="" type="checkbox"/>
Last Modified By	LastModifiedBy	Lookup(User)		<input checked="" type="checkbox"/>
Model Number	Model_Number__c	Text(20) (Unique Case Insensitive)		<input checked="" type="checkbox"/>
Owner	OwnerId	Lookup(User,Group)		<input checked="" type="checkbox"/>
Price	Price__c	Currency(16, 2)		<input checked="" type="checkbox"/>
Product Name	Name	Text(80)		<input checked="" type="checkbox"/>
Purchase Date	Purchase_Date__c	Date		<input checked="" type="checkbox"/>
Serial Number	Serial_Number__c	Text(30) (Unique Case Insensitive)		<input checked="" type="checkbox"/>

Warranty_c

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

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
End Date	End_Date__c	Date		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		
Product	Product_c	Lookup(Product)		
Start Date	Start_Date__c	Date		
Warranty Name	Name	Text(80)		
Warranty Term	Warranty_Term__c	Number(3, 0)		

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

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Issue Description	Issue_Description__c	Text Area(255)		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		
Product	Product_c	Lookup(Product)		
Request Date	Request_Date__c	Date		
Service Request Name	Name	Text(80)		
Status	Status__c	Picklist		

❖ 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).

The screenshot shows the Salesforce Setup interface with the URL <https://orgfarm-b1b1731f09-dev-ed.lightning.force.com/lightning/setup/ObjectManager/01lgK000002DU4E/RecordTypes/view>. The page title is "Service Request". The left sidebar is expanded, showing "Record Types" selected under "Object Manager". The main content area displays a table titled "Record Types" with two items: "Repair" and "Replacement". The "Repair" record type is active and was modified by K Sridevi on 9/17/2025, 6:05 AM. The "Replacement" record type is also active and was modified by K Sridevi on 9/17/2025, 6:08 AM. The table includes columns for "Record Type Label", "Description", "Active", and "Modified By". A "Quick Find" search bar and "New" and "Page Layout Assignment" buttons are at the top right of the table.

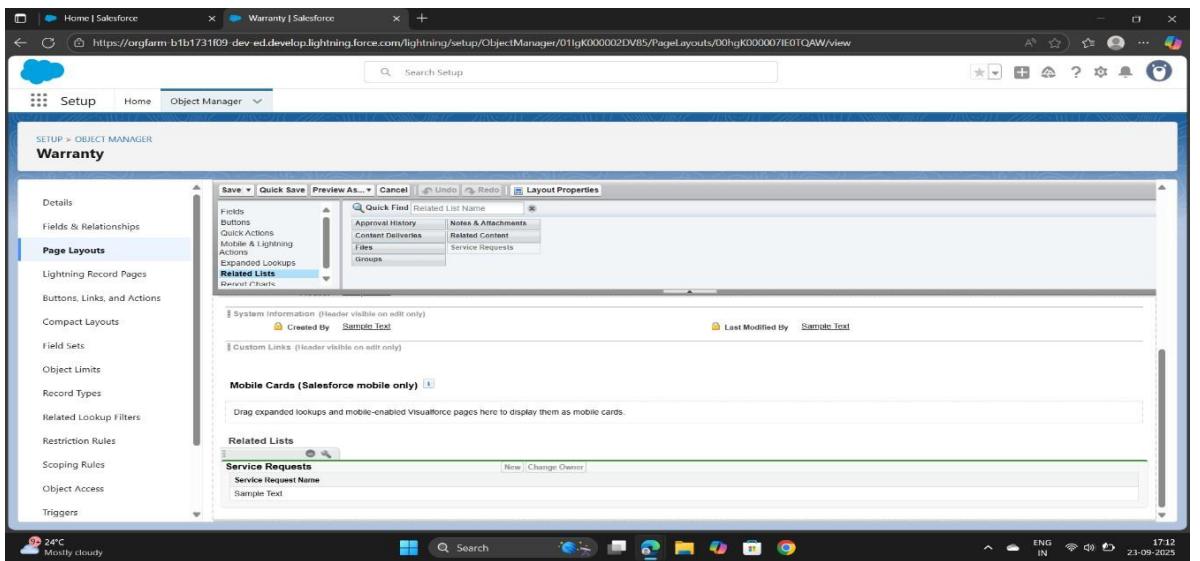
❖ Step 4: Page Layouts

Customize layouts so relationships are visible.

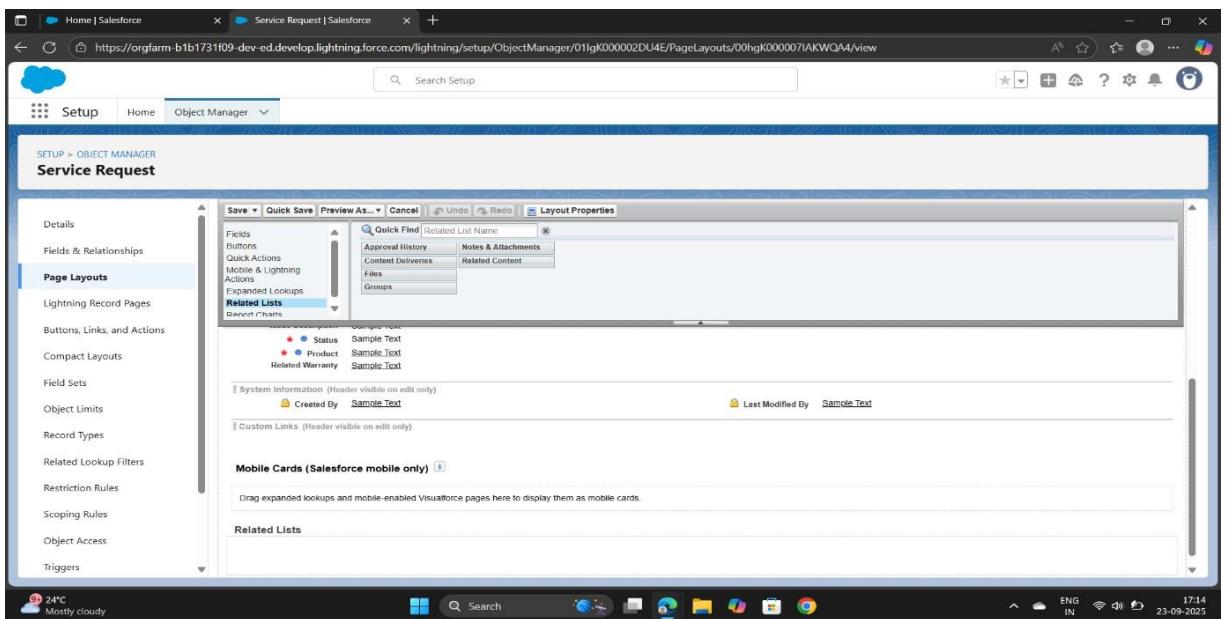
- **Product_c Layout** → add Related Lists: Warranties, Service Requests.

The screenshot shows the Salesforce Setup interface with the URL <https://orgfarm-b1b1731f09-dev-ed.lightning.force.com/lightning/setup/ObjectManager/01lgK000002DUyP/PageLayouts/00hgK000007IDpBQAW/view>. The page title is "Product". The left sidebar is expanded, showing "Page Layouts" selected under "Object Manager". The main content area shows the "Page Layout Properties" for the "Product" object. Under "Related Lists", there are three sections: "Service Requests", "Warranties", and "Open Activities". Each section has a "New" and "Change Owner" button. The "Service Requests" section shows "Service Request Name" and "Sample Text". The "Warranties" section shows "Warranty Name" and "Sample Text". The "Open Activities" section shows a table with columns: Subject, Name, Task, Due Date, Status, Priority, and Assigned To. One row is listed with "Subject: Sample Text", "Name: Sample Text", "Task: ✓", "Due Date: 9/23/2025, 4:36 AM", "Status: Sample Text", "Priority: Sample Text", and "Assigned To: Sarah Sample".

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

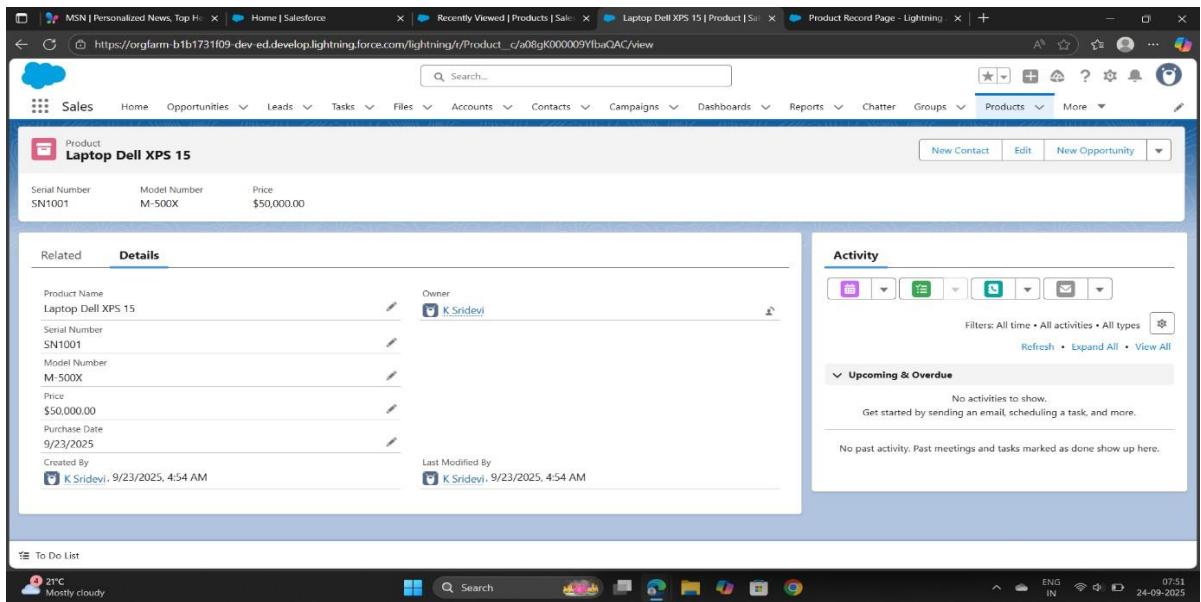


- **Service_Request_c Layout** → add Related Product + Related Warranty.



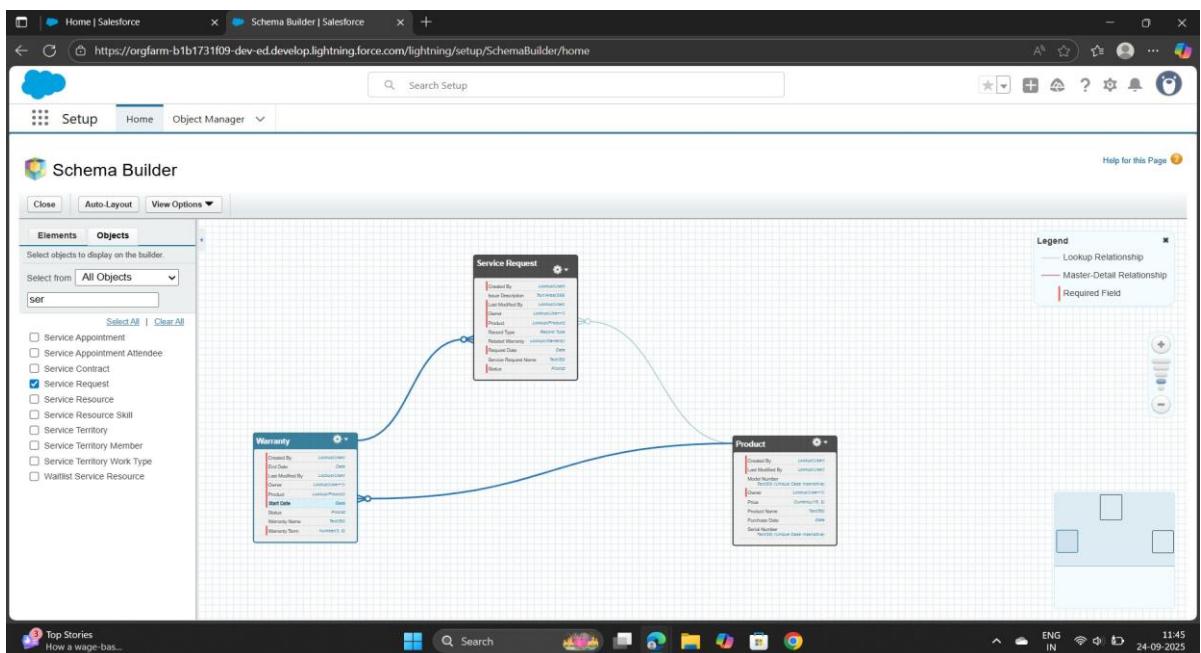
❖ Step 5: Compact Layouts

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

- o 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.

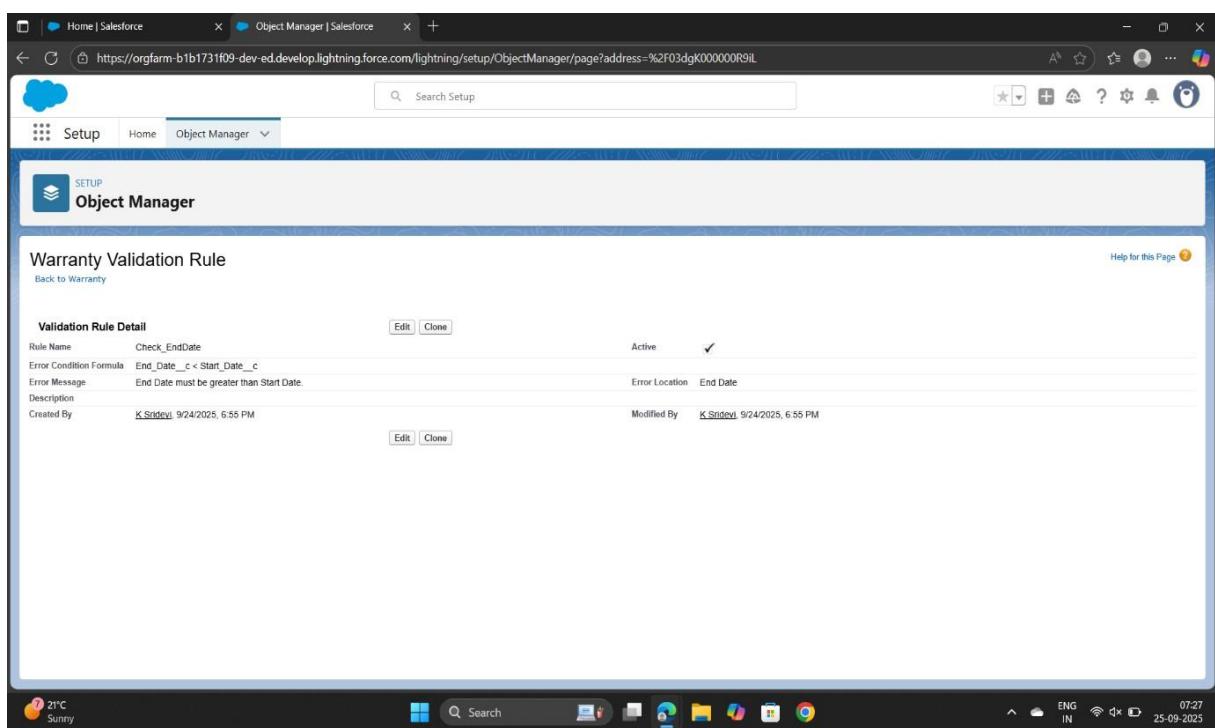
A. Warranty Dates Validation

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

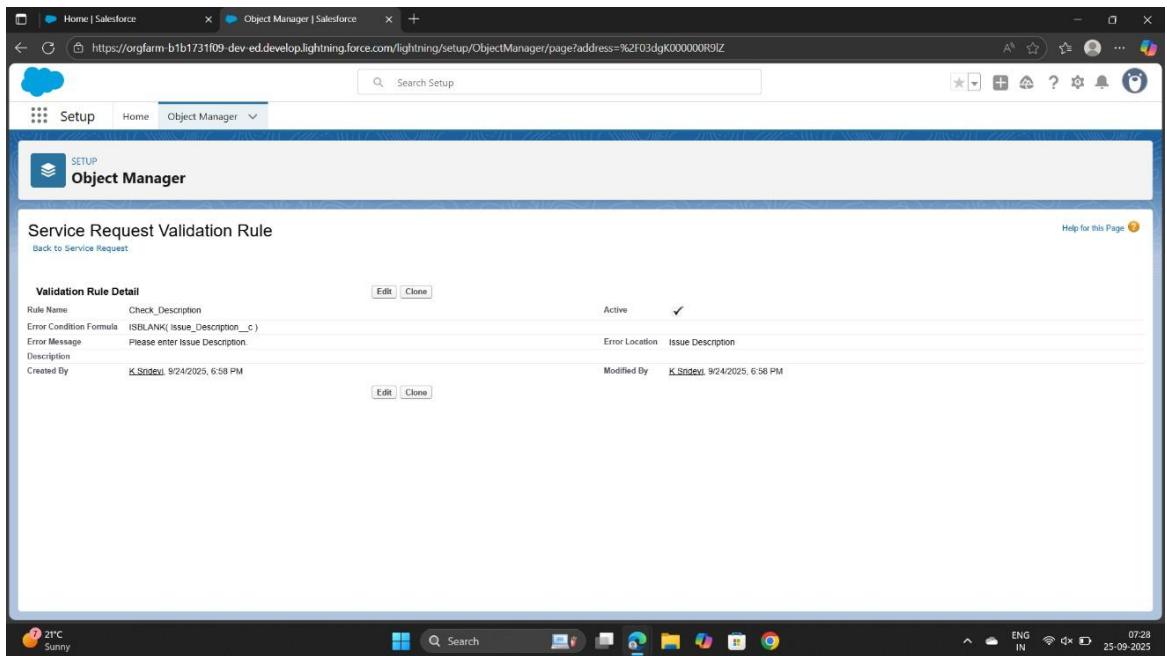
Goal: Every Service Request must have a description.

- 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.

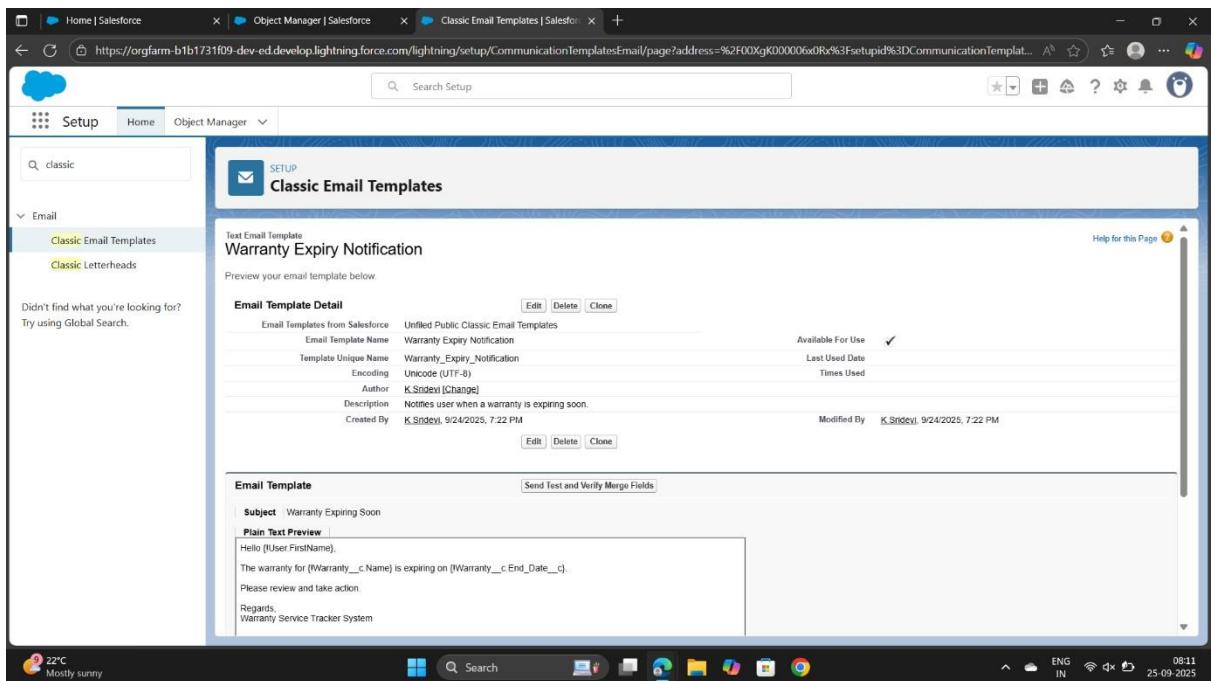
Example: Notify when Warranty is Expiring

- 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.

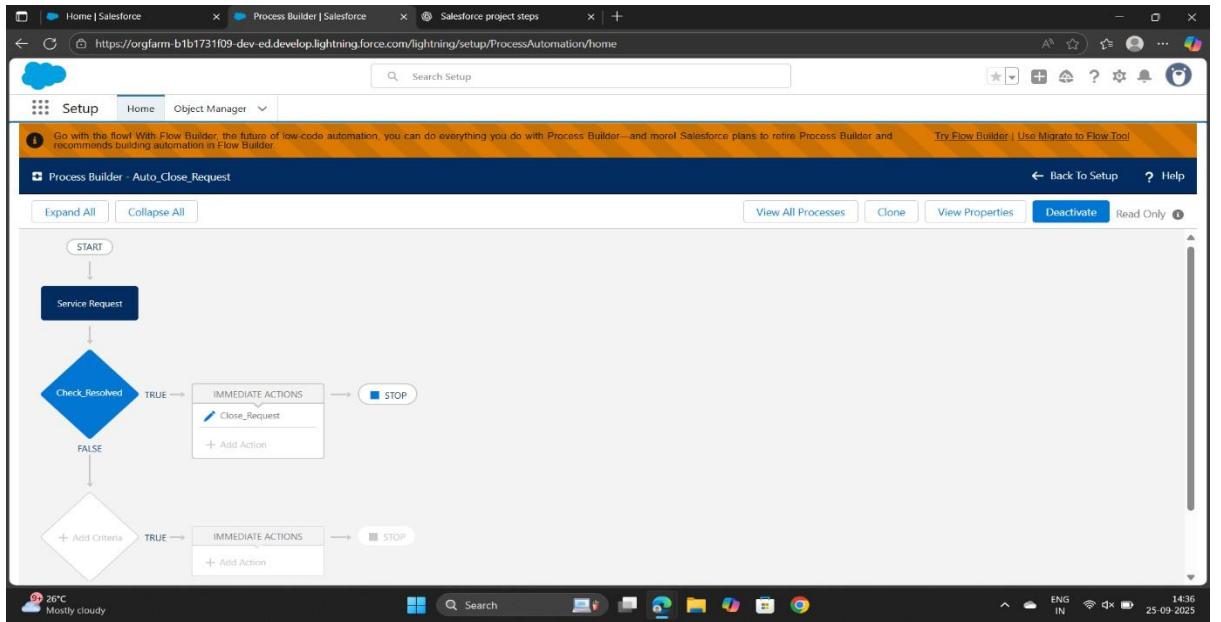


Step 3: Process Builder Automation

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.

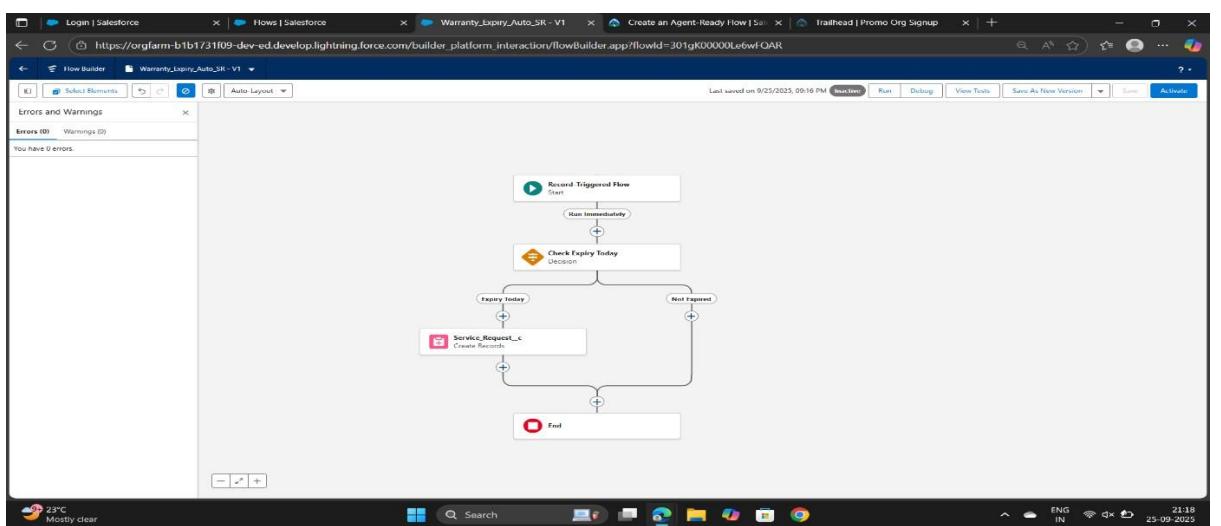


Step 4: Record-Triggered Flow

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.

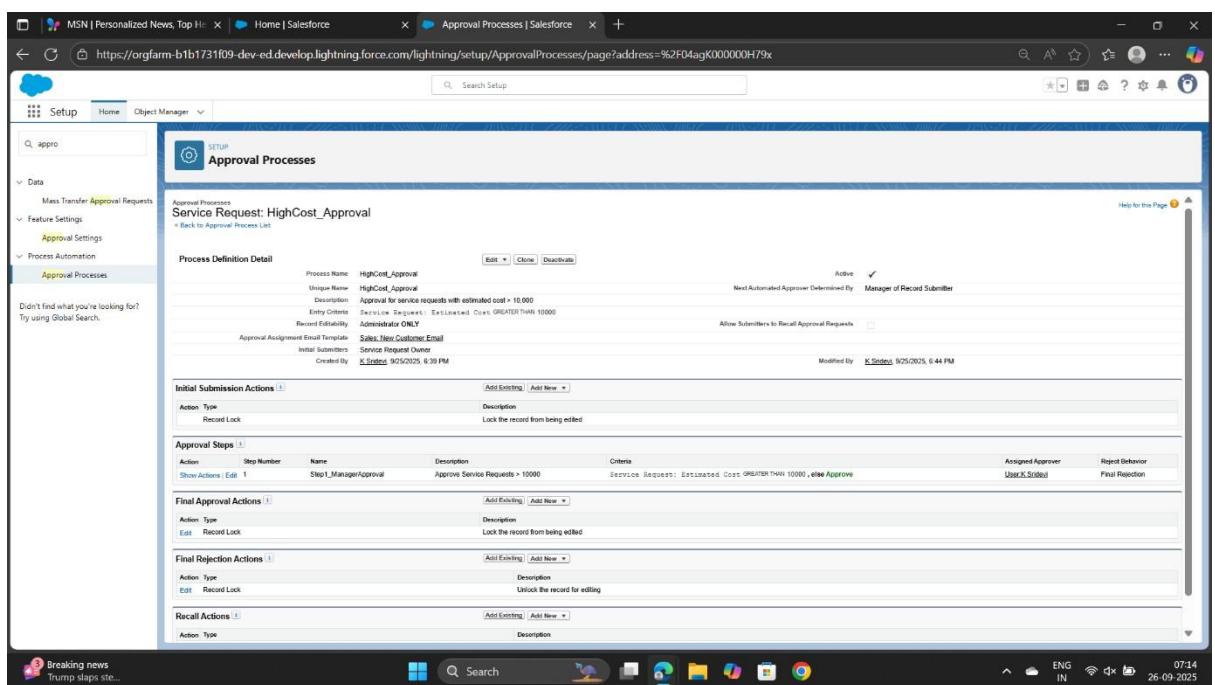


Step 5: Approval Process

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)

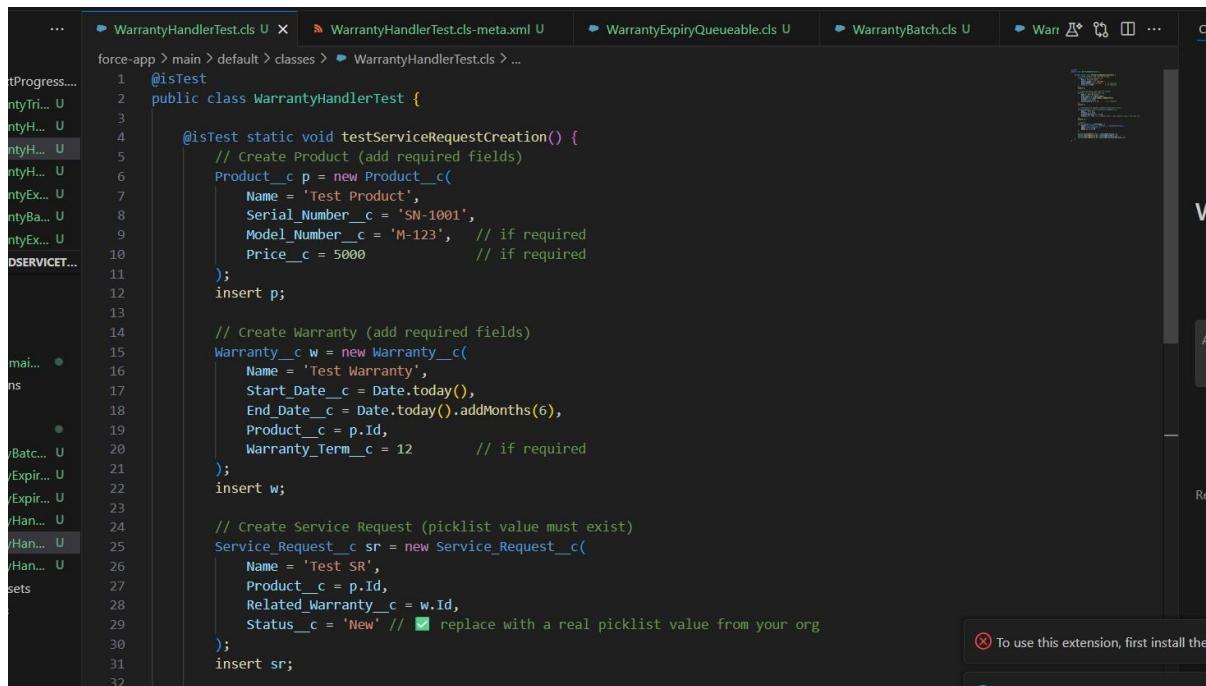
1. Classes & Objects

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.



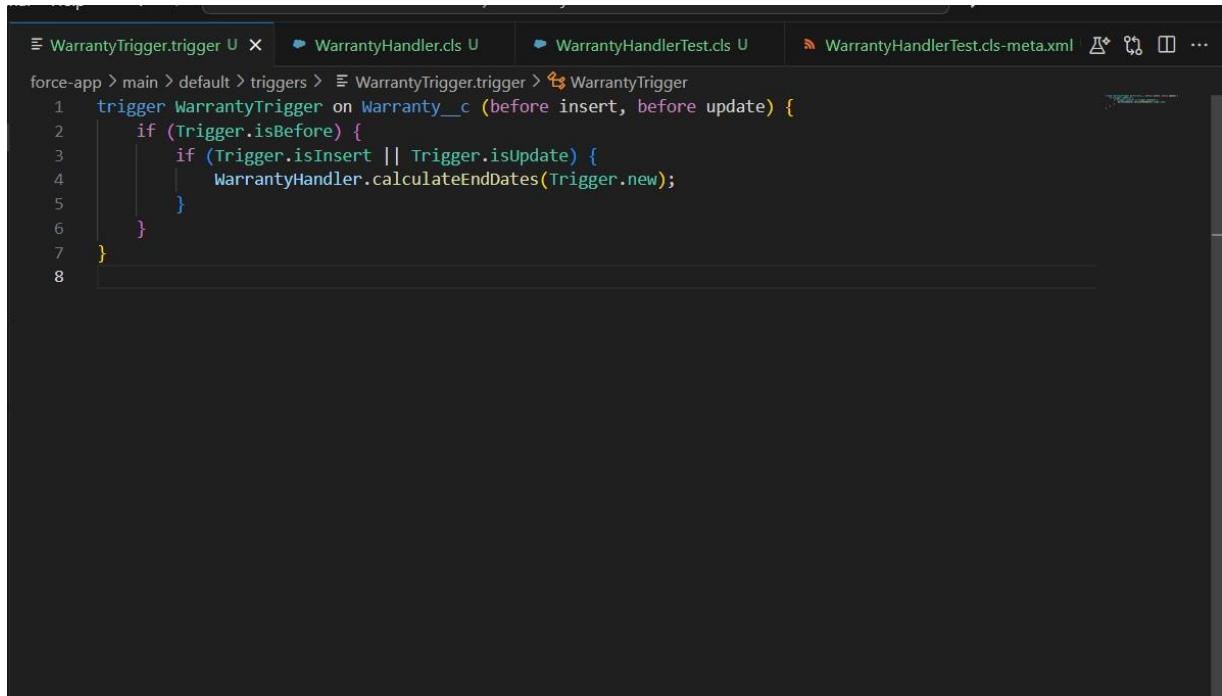
```
... ➜ WarrantyHandlerTest.cls U ➜ WarrantyHandlerTest.cls-meta.xml U ➜ WarrantyExpiryQueueable.cls U ➜ WarrantyBatch.cls U ➜ War...
1 @isTest
2 public class WarrantyHandlerTest {
3     @isTest static void testServiceRequestCreation() {
4         // Create Product (add required fields)
5         Product__c p = new Product__c(
6             Name = 'Test Product',
7             Serial_Number__c = 'SN-1001',
8             Model_Number__c = 'M-123',    // if required
9             Price__c = 5000            // if required
10        );
11        insert p;
12
13        // Create Warranty (add required fields)
14        Warranty__c w = new Warranty__c(
15            Name = 'Test Warranty',
16            Start_Date__c = Date.today(),
17            End_Date__c = Date.today().addMonths(6),
18            Product__c = p.Id,
19            Warranty_Term__c = 12        // if required
20        );
21        insert w;
22
23        // Create Service Request (picklist value must exist)
24        Service_Request__c sr = new Service_Request__c(
25            Name = 'Test SR',
26            Product__c = p.Id,
27            Related_Warranty__c = w.Id,
28            Status__c = 'New' // ✅ replace with a real picklist value from your org
29        );
30        insert sr;
31
32 }
```

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).



```
force-app > main > default > triggers > WarrantyTrigger.trigger > WarrantyTrigger
1 trigger WarrantyTrigger on Warranty__c (before insert, before update) {
2     if (Trigger.isBefore) {
3         if (Trigger.isInsert || Trigger.isUpdate) {
4             WarrantyHandler.calculateEndDates(Trigger.new);
5         }
6     }
7 }
8
```

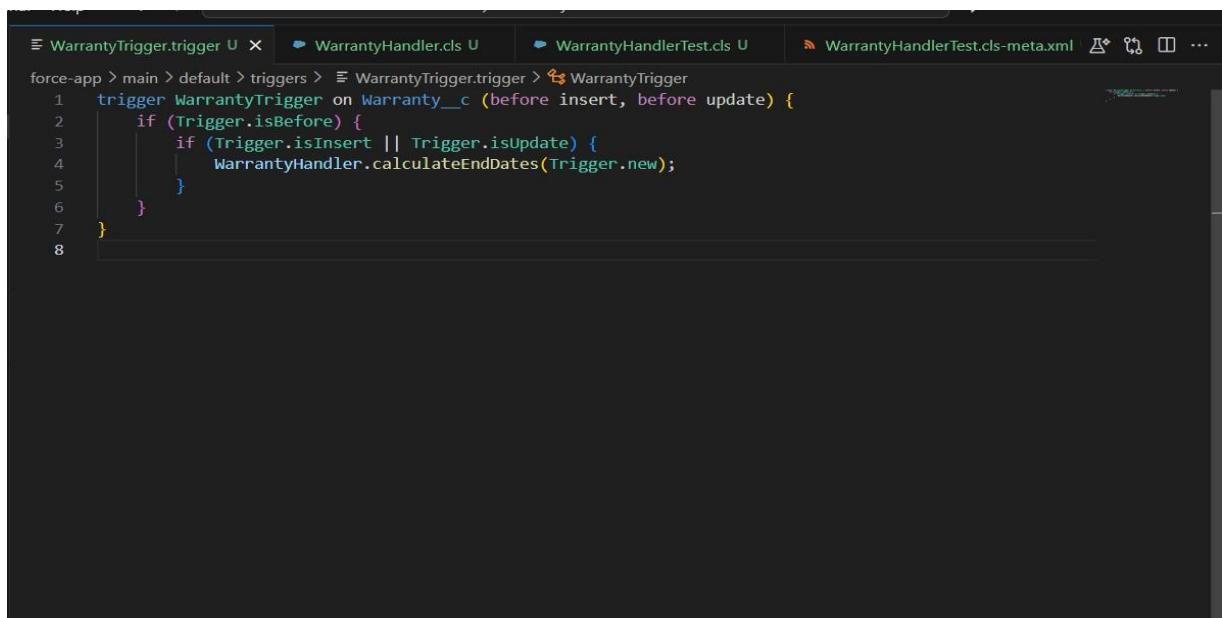
3. Trigger Design

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().



```
force-app > main > default > triggers > WarrantyTrigger.trigger > WarrantyTrigger
1 trigger WarrantyTrigger on Warranty__c (before insert, before update) {
2     if (Trigger.isBefore) {
3         if (Trigger.isInsert || Trigger.isUpdate) {
4             WarrantyHandler.calculateEndDates(Trigger.new);
5         }
6     }
7 }
8
```

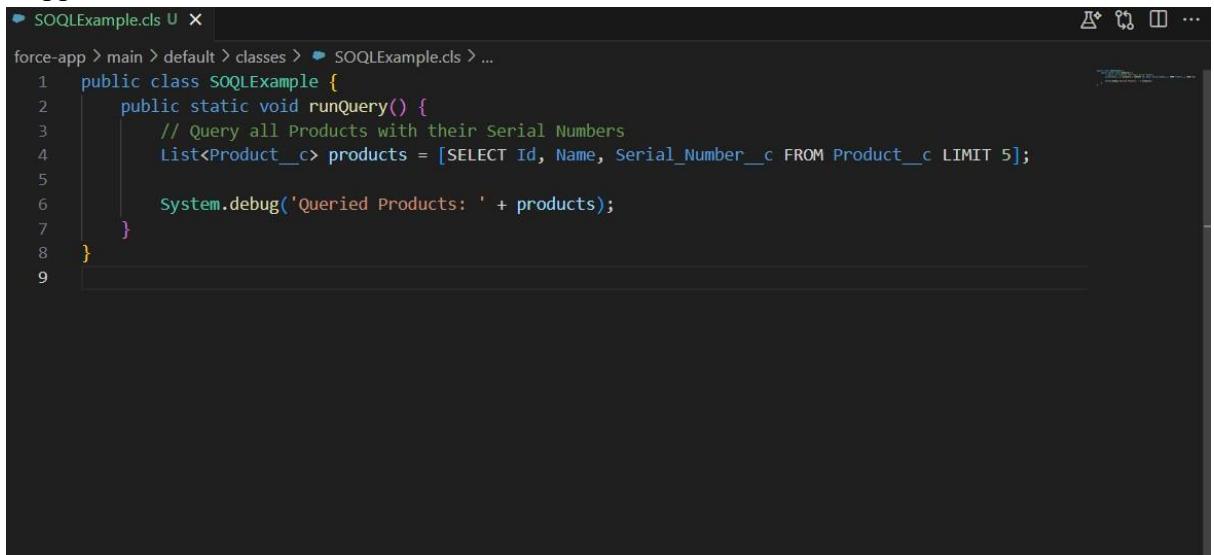
4. SOQL & SOSL

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. . .   **Screenshot:** Code snippet in VS Code.



```
▶ SOQLExample.cls U X
force-app > main > default > classes > ▶ SOQLExample.cls > ...
1  public class SOQLExample {
2      public static void runQuery() {
3          // Query all Products with their Serial Numbers
4          List<Product__c> products = [SELECT Id, Name, Serial_Number__c FROM Product__c LIMIT 5];
5
6          System.debug('Queried Products: ' + products);
7      }
8  }
```

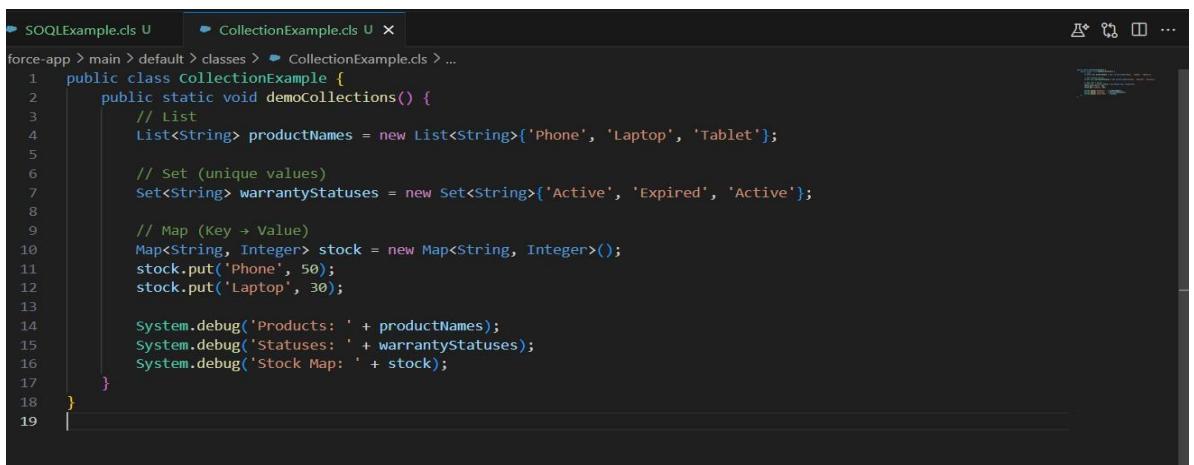
5. Collections (List, Set, Map)

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.



```
▶ SOQLExample.cls U   ▶ CollectionExample.cls U X
force-app > main > default > classes > ▶ CollectionExample.cls > ...
1  public class CollectionExample {
2      public static void demoCollections() {
3          // list
4          List<String> productNames = new List<String>{'Phone', 'Laptop', 'Tablet'};
5
6          // Set (unique values)
7          Set<String> warrantyStatuses = new Set<String>{'Active', 'Expired', 'Active'};
8
9          // Map (Key → Value)
10         Map<String, Integer> stock = new Map<String, Integer>();
11         stock.put('Phone', 50);
12         stock.put('Laptop', 30);
13
14         System.debug('Products: ' + productNames);
15         System.debug('Statuses: ' + warrantyStatuses);
16         System.debug('Stock Map: ' + stock);
17     }
18 }
19 }
```

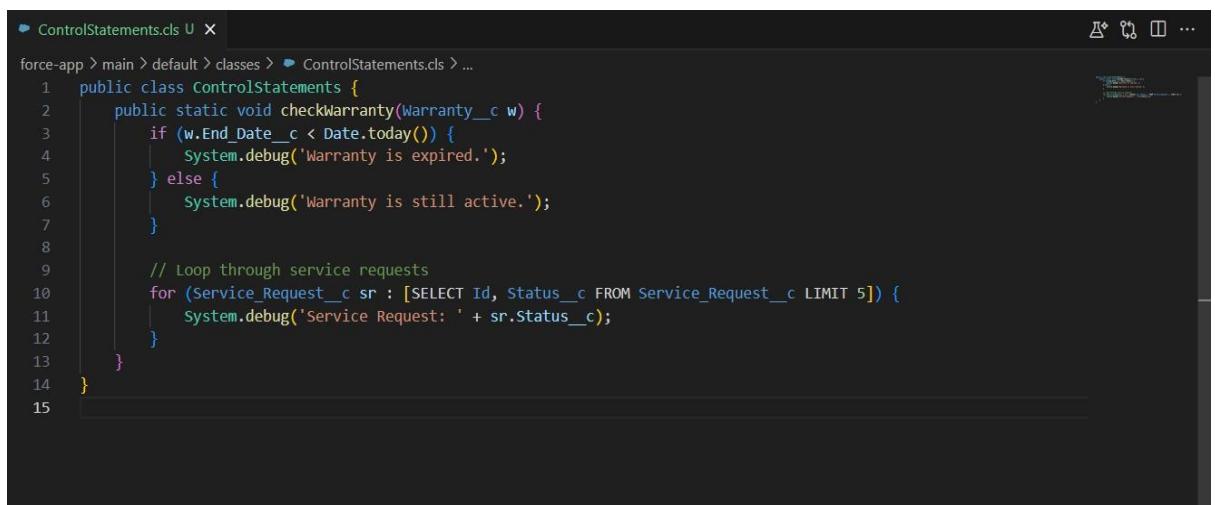
6. Control Statements

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.



The screenshot shows a code editor window with the file 'ControlStatements.cls' open. The code is written in Apex and contains logic to check if a warranty has expired and to loop through service requests. The code is as follows:

```
force-app > main > default > classes > ControlStatements.cls > ...
1  public class ControlStatements {
2      public static void checkWarranty(Warranty__c w) {
3          if (w.End_Date__c < Date.today()) {
4              System.debug('Warranty is expired.');
5          } else {
6              System.debug('Warranty is still active.');
7          }
8
9          // Loop through service requests
10         for (Service_Request__c sr : [SELECT Id, Status__c FROM Service_Request__c LIMIT 5]) {
11             System.debug('Service Request: ' + sr.Status__c);
12         }
13     }
14 }
15
```

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.

The screenshot shows the Salesforce code editor with the file `WarrantyExpiryQueueable.cls` open. The code implements the `Queueable` interface and contains logic to find warranty records expiring today and create new service requests for them.

```
1 public class WarrantyExpiryQueueable implements Queueable {
2     public void execute(QueueableContext ctx) {
3         List<Warranty__c> listW = [
4             SELECT Id, Product__c, End_Date__c
5             FROM Warranty__c
6             WHERE End_Date__c = :Date.today()
7         ];
8         List<Service_Request__c> toCreate = new List<Service_Request__c>();
9         for (Warranty__c w : listW) {
10             Service_Request__c sr = new Service_Request__c(
11                 Name = 'Auto-Generated Expiry Request',
12                 Status__c = 'Open',
13                 Product__c = w.Product__c
14             );
15             toCreate.add(sr);
16         }
17         if (!toCreate.isEmpty()) insert toCreate;
18     }
19 }
20 }
```

9. Scheduled Apex

What it is:

Runs Apex at scheduled times.

What we did:

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

The screenshot shows the Salesforce code editor with the file `WarrantyExpiryScheduler.cls` open. It implements the `Schedulable` interface and uses the `System.enqueueJob` method to call the `WarrantyExpiryQueueable` class.

```
1 global class WarrantyExpiryScheduler implements Schedulable {
2     global void execute(SchedulableContext ctx) {
3         System.enqueueJob(new WarrantyExpiryQueueable());
4     }
5 }
6 }
```

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.

```
force-app > main > default > classes > FutureExample.cls > ...
1  public class FutureExample {
2      @future
3      public static void logWarrantyCount() {
4          Integer count = [SELECT COUNT() FROM Warranty__c];
5          System.debug('Total warranties: ' + count);
6      }
7  }
```

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.

```
force-app > main > default > classes > ExceptionExample.cls > ...
1  public class ExceptionExample {
2      public static void safeQuery() {
3          try {
4              List<Product__c> products = [SELECT Id, Name FROM Product__c LIMIT 1];
5              System.debug('Found: ' + products[0].Name);
6          } catch (Exception e) {
7              System.debug('Error occurred: ' + e.getMessage());
8          }
9      }
10 }
```

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).

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

```
1  @isTest
2  public class WarrantyHandlerTest {
3
4      @isTest static void testServiceRequestCreation() {
5          // Create Product (add required fields)
6          Product__c p = new Product__c(
7              Name = 'Test Product',
8              Serial_Number__c = 'SN-1001',
9              Model_Number__c = 'M-123',    // if required
10             Price__c = 5000            // if required
11         );
12         insert p;
13
14         // Create Warranty (add required fields)
15         Warranty__c w = new Warranty__c(
16             Name = 'Test Warranty',
17             Start_Date__c = Date.today(),
18             End_Date__c = Date.today().addMonths(6),
19             Product__c = p.Id,
20             Warranty_Term__c = 12        // if required
21         );
22         insert w;
23
24         // Create Service Request (picklist value must exist)
25         Service_Request__c sr = new Service_Request__c(
26             Name = 'Test SR',
27             Product__c = p.Id,
28             Related_Warranty__c = w.Id,
29             Status__c = 'New' // ✅ replace with a real picklist value from your org
30         );
31         insert sr;
32
33         // Verify
34     }
35 }
```

13. Asynchronous Processing

What it is:

Background jobs: Queueable, Batch, Future, Scheduled.

What we did:

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

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

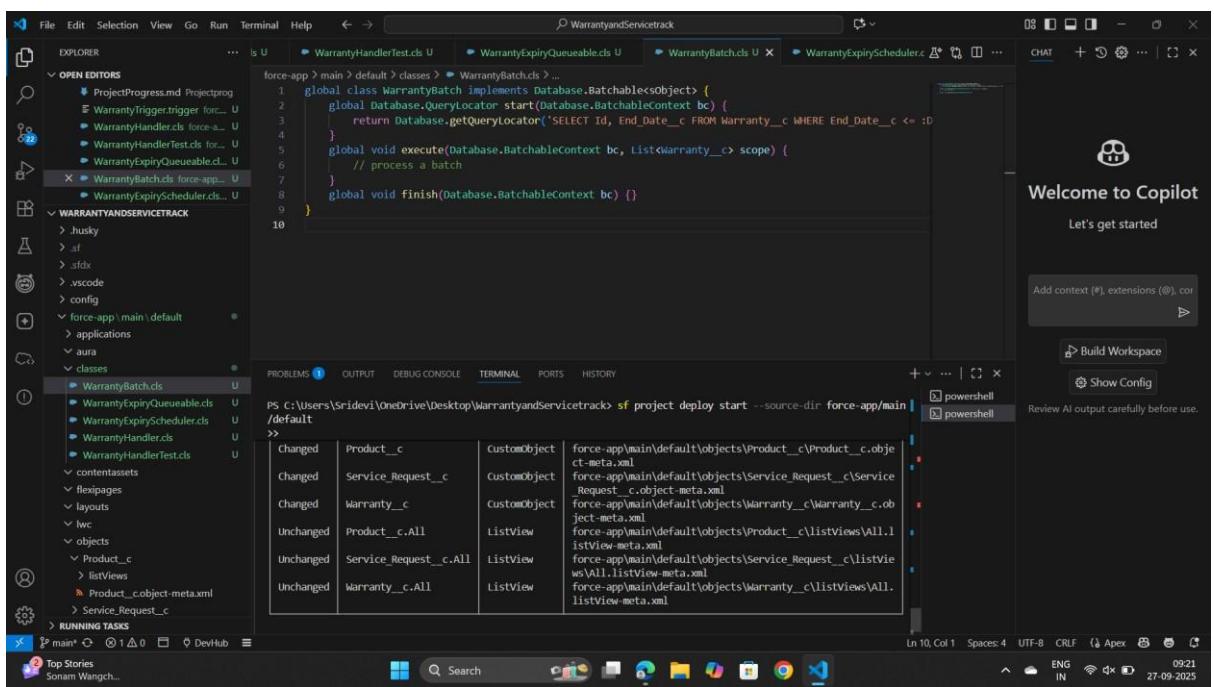
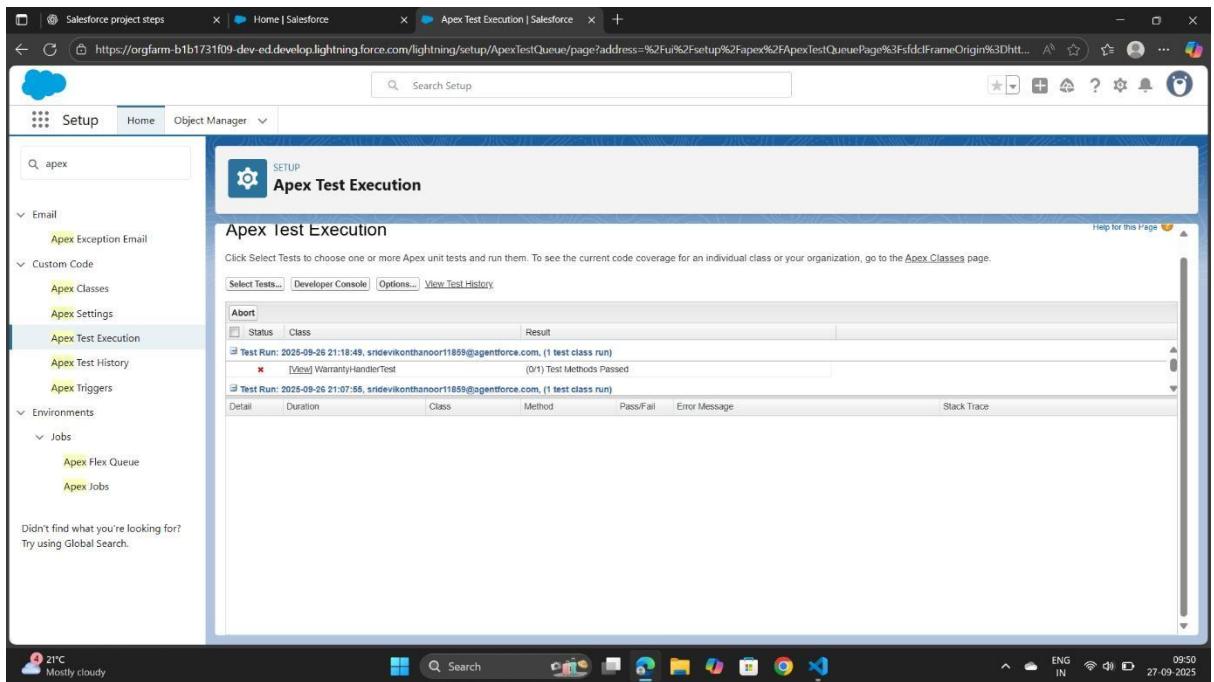
- File Explorer:** Shows the project structure under "WARRANTYANDSERVICETRACK".
- Code Editor:** Displays an Apex class named "WarrantyHandlerTest.cls" with the following code:

```
force:app > main > default > classes > WarrantyHandlerTest.cls > ...
private class WarrantyHandlerTest {
    @isTest static void testCalculateEndDate() {
        Date startDate = Date.today();
        Warranty__c w = new Warranty__c(Name='TestWarranty', Start_Date__c = startDate, Warranty_Term__c = 1);
        Test.startTest();
        insert w;
        Test.stopTest();
        Warranty__c w2 = [SELECT End_Date__c FROM Warranty__c WHERE Id = :w.Id LIMIT 1];
        System.assertEquals(startDate.addMonths(12), w2.End_Date__c);
    }
}
```

- Terminal:** Shows the command "PS C:\Users\Sridevi\OneDrive\Desktop\WarrantyandServicetrack> sfdx apex run test --result-format human --wait 10" being run.
- Output:** Displays the "PROBLEMS" tab and a "Test Summary" table:

NAME	VALUE
Outcome	Passed
Tests Ran	2
Pass Rate	100%
Fail Rate	0%
Skip Rate	0%
Test Suite Id	700g000000Aen1
Test Setup Time	0 ms
Test Execution Time	1015 ms
Test Total Time	1015 ms
Org Id	00Dg0000007XkFUuW

- Right Panel:** Shows a "Welcome to Copilot" message and a "Review AI output carefully before use." note.



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

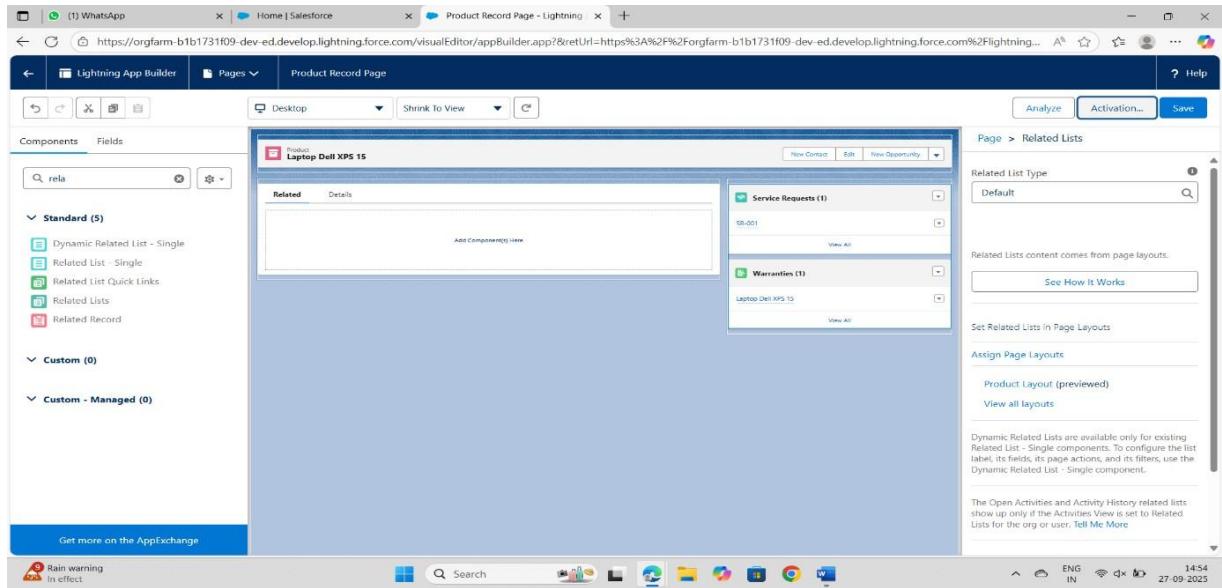
- **What we did:**

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.



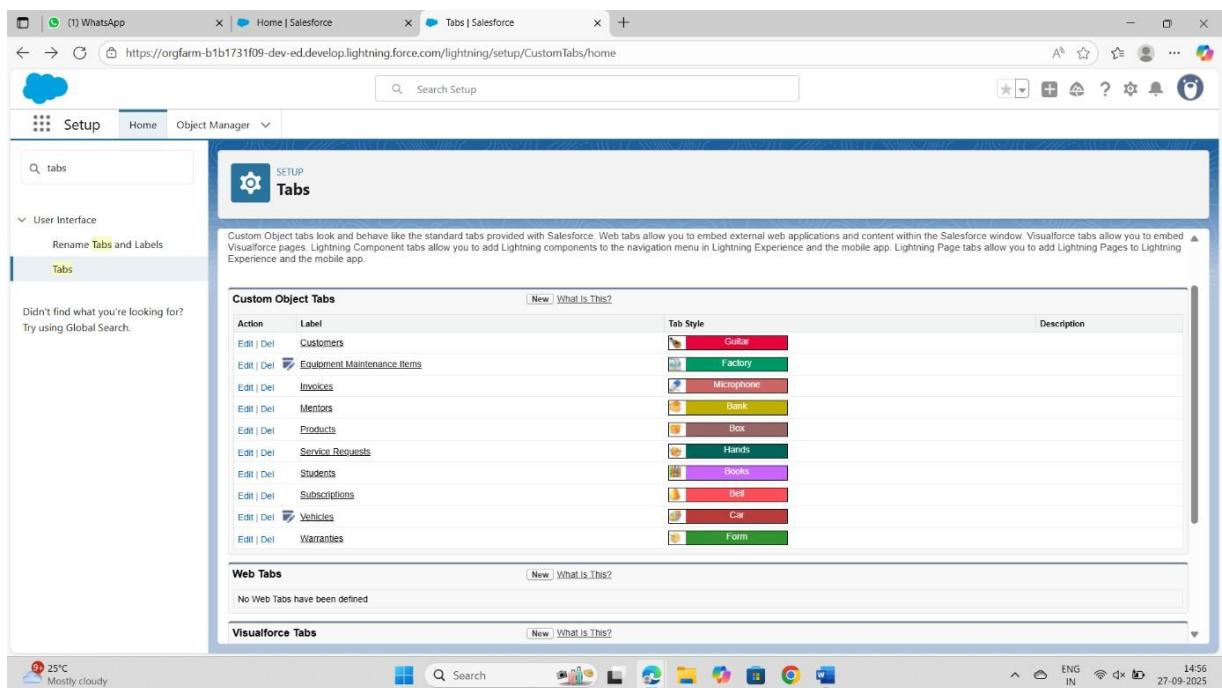
3. Tabs

- **What we did:**

Added **Custom Tabs** for each custom object.

- **Steps:**

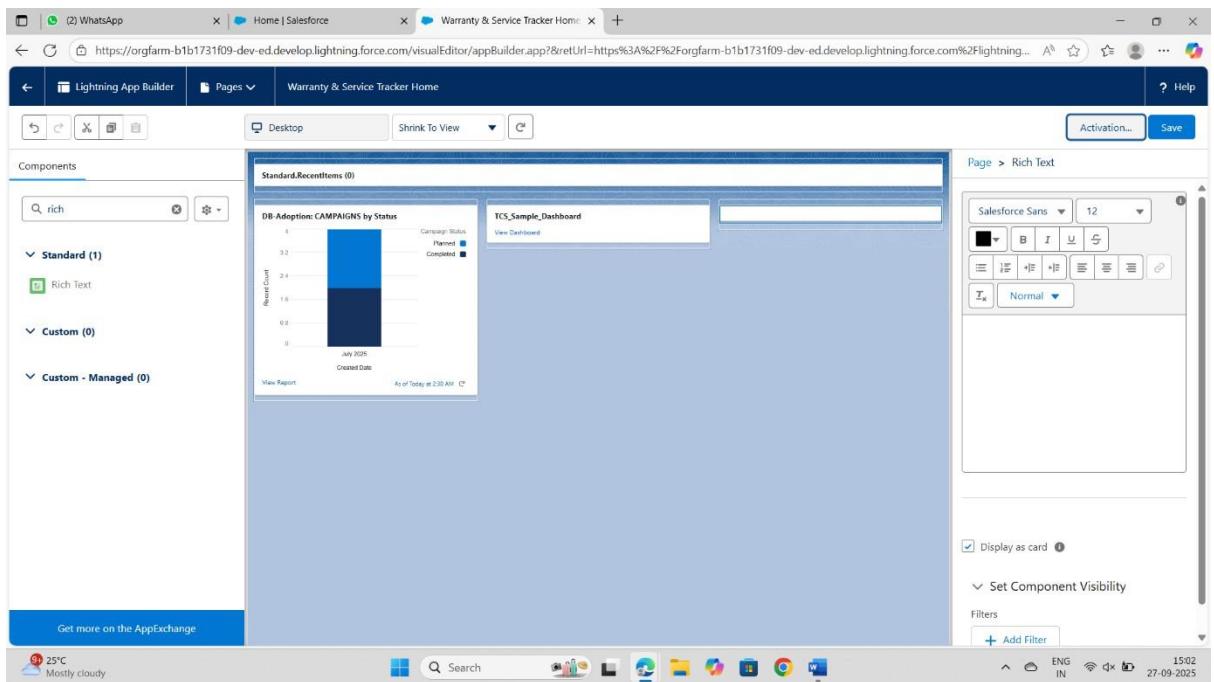
1. Setup → Tabs → New.
2. Chose Object = Product_c, Warranty_c, Service_Request_c.
3. Selected tab style (icon & color).
4. Added them to the Warranty & Service Tracker App.



4. Home Page Layouts

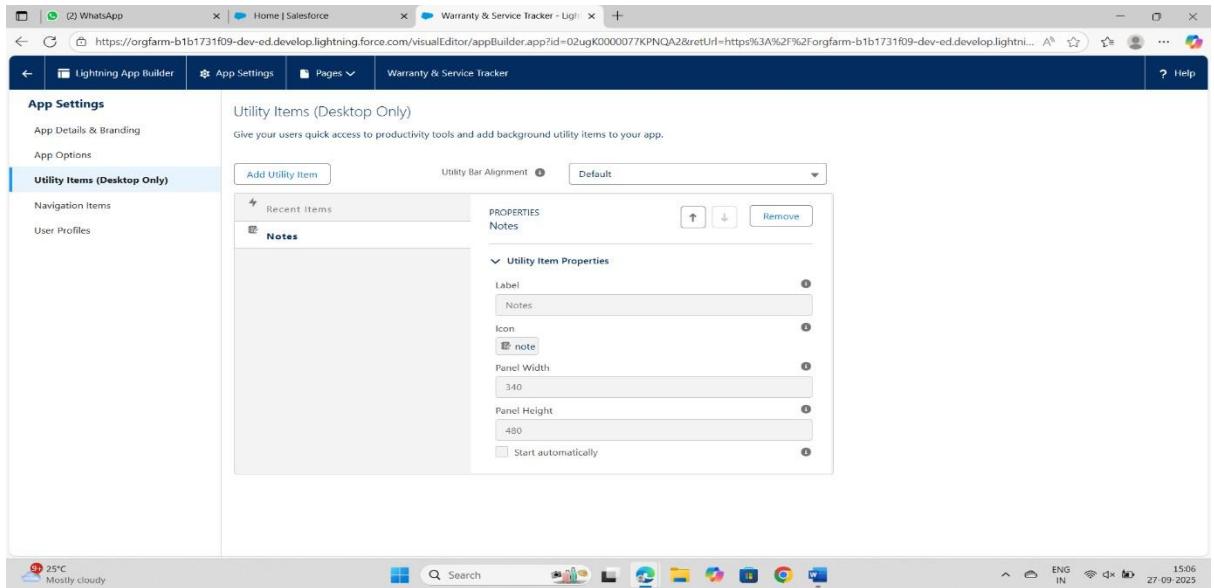
- **What we did:**
- Customized Home Page to show dashboard components and quick links.**
- **Steps:**

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:**

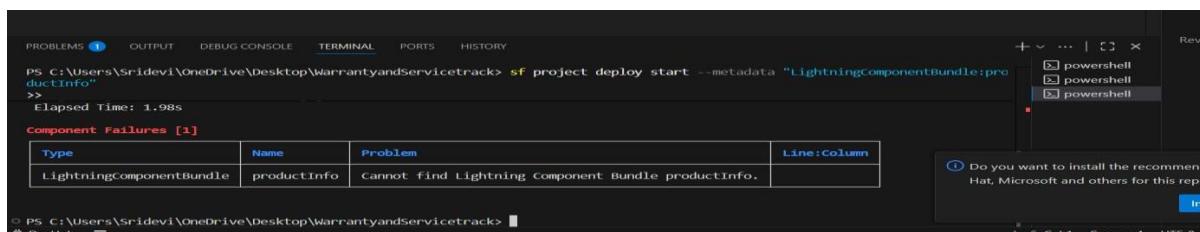
```
import { LightningElement, api } from 'lwc';

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';

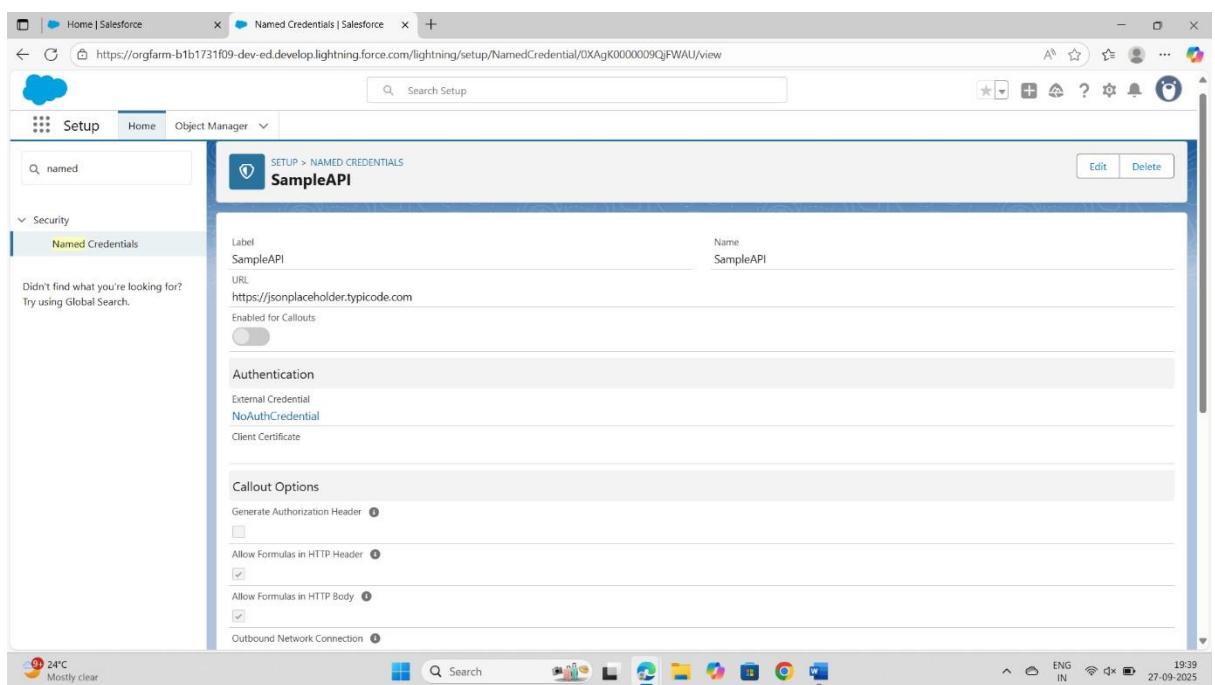
export default class ProductNavigator extends NavigationMixin(LightningElement) {

    navigateToProduct() {
        this[NavigationMixin.Navigate]({
            type: 'standard_recordPage',
            attributes: {
                recordId: 'a01XX00000XXXX',
                objectApiName: 'Product_c',
                actionName: 'view'
            }
        });
    }
}
```

Phase 7: Integration & External Access

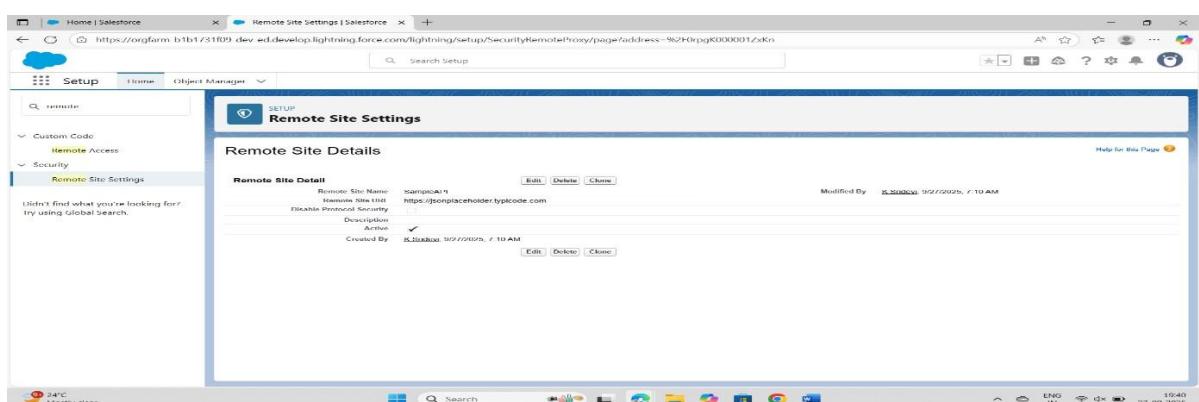
1. Named Credentials

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



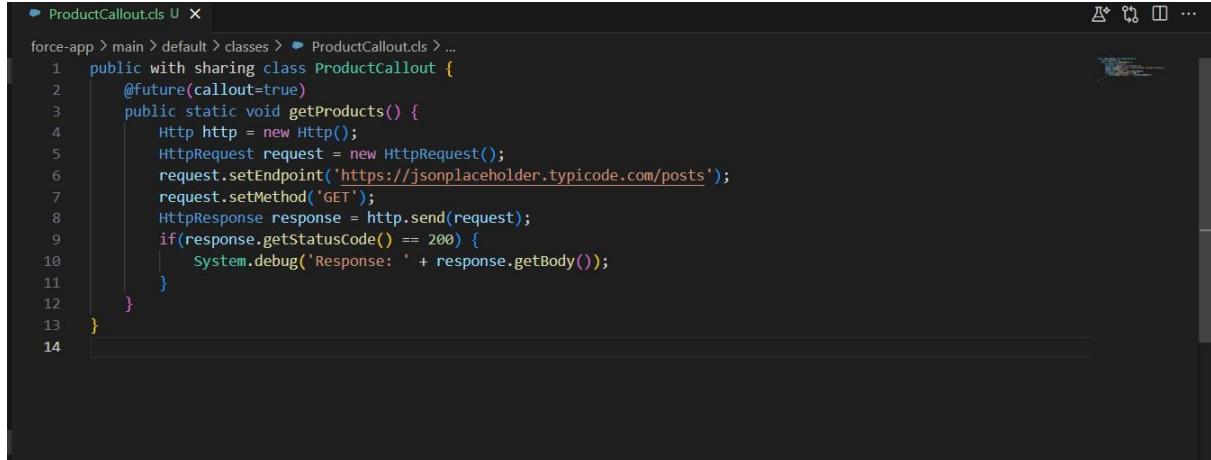
2. Remote Site Settings

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



3. Apex REST Callout

In VS Code → create ProductCallout.cls

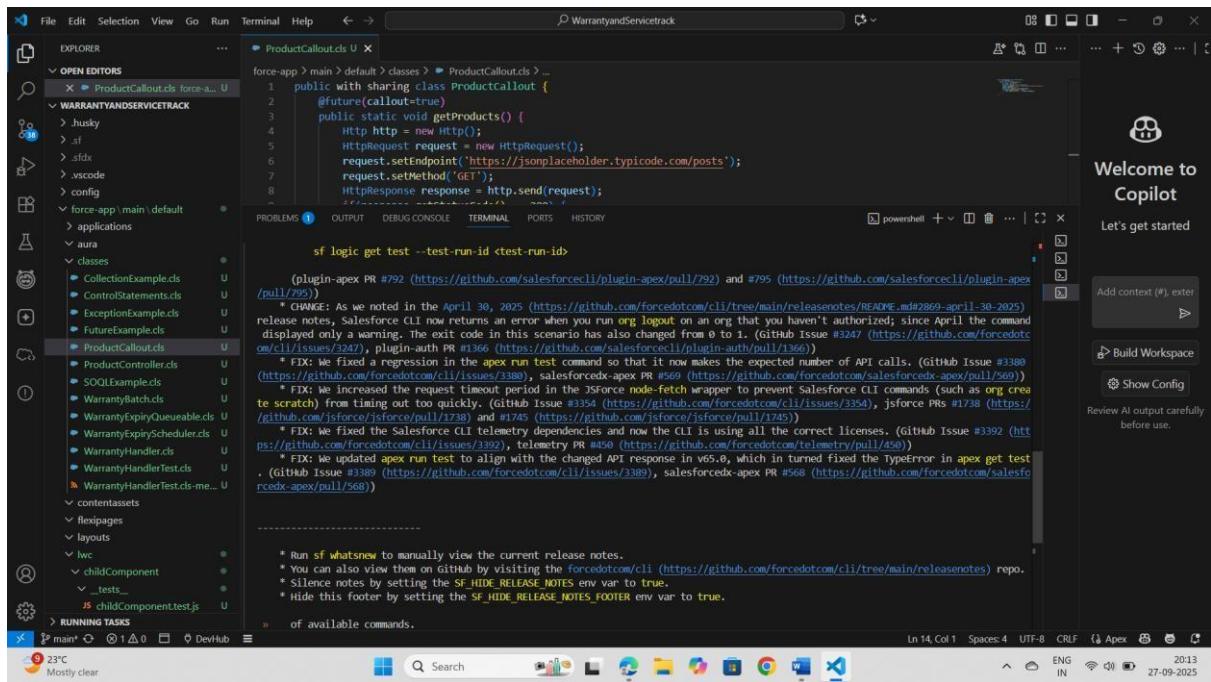


```
ProductCallout.cls
force-app > main > default > classes > ProductCallout.cls > ...
1  public with sharing class ProductCallout {
2      @future(callout=true)
3      public static void getProducts() {
4          Http http = new Http();
5          HttpRequest request = new HttpRequest();
6          request.setEndpoint('https://jsonplaceholder.typicode.com/posts');
7          request.setMethod('GET');
8          HttpResponse response = http.send(request);
9          if(response.getStatusCode() == 200) {
10              System.debug('Response: ' + response.getBody());
11          }
12      }
13  }
14 }
```

4. Run Callout

Open Developer Console → Execute Anonymous:

ProductCallout.getProducts();



ProductCallout.cls

```
force-app > main > default > classes > ProductCallout.cls > ...
1  public with sharing class ProductCallout {
2      @future(callout=true)
3      public static void getProducts() {
4          Http http = new Http();
5          HttpRequest request = new HttpRequest();
6          request.setEndpoint('https://jsonplaceholder.typicode.com/posts');
7          request.setMethod('GET');
8          HttpResponse response = http.send(request);
9          if(response.getStatusCode() == 200) {
10              System.debug('Response: ' + response.getBody());
11          }
12      }
13  }
14 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS HISTORY

sf logic get test --test-run-id ctest-run-id
(plugin-apex PR #792 (<https://github.com/salesforcecli/plugin-apex/pull/792>) and #795 (<https://github.com/salesforcecli/plugin-apex/pull/795>))
* CHANGE: As we noted in the April 30, 2025 (<https://github.com/forcedotcom/cli/tree/main/releasenotes/README.md#apr2869-april-30-2025>) release notes, Salesforce CLI now returns an error when you run org logout on an org that you haven't authorized; since April the command displayed only a warning. The exit code in this scenario has also changed from 0 to 1. ([GitHub Issue #3247](#) (<https://github.com/forcedotcom/cli/issues/3247>)), plugin-auth PR #1366 (<https://github.com/salesforcecli/plugin-auth/pull/1366>)
* FIX: We fixed a regression in the apex run test command so that it now makes the expected number of API calls. ([GitHub Issue #3380](#) (<https://github.com/forcedotcom/cli/issues/3380>), salesforcedx-apex PR #569 (<https://github.com/forcedotcom/salesforcedx-apex/pull/569>))
* FIX: We increased the request timeout period in the JSForce node-fetch wrapper to prevent Salesforce CLI commands (such as org create/scratch) from timing out too quickly. ([GitHub Issue #3354](#) (<https://github.com/forcedotcom/cli/issues/3354>), jsforce PR #1738 (<https://github.com/jsforce/jsforce/pull/1738>) and #1745 (<https://github.com/jsforce/jsforce/pull/1745>))
* FIX: We fixed the Salesforce CLI telemetry dependencies and now the CLI is using all the correct licenses. ([GitHub Issue #3392](#) (<https://github.com/forcedotcom/cli/issues/3392>), telemetry PR #450 (<https://github.com/forcedotcom/telemetry/pull/450>))
* FIX: We updated apex run test to align with the changed APT response in v65.0, which in turn fixed the TypeError in apex get test. ([GitHub Issue #3389](#) (<https://github.com/forcedotcom/cli/issues/3389>), salesforcedx-apex PR #568 (<https://github.com/forcedotcom/salesforcedx-apex/pull/568>))

* Run sf what'snew to manually view the current release notes.
* You can also view them on GitHub by visiting the [forcedotcom/cli](https://github.com/forcedotcom/cli) (<https://github.com/forcedotcom/cli/tree/main/releasenotes>) repo.
* Silence notes by setting the SF_HIDE_RELEASE_NOTES env var to true.
* Hide this footer by setting the SF_HIDE_RELEASE_NOTES_FOOTER env var to true.

of available commands.

Ln 14 Col 1 Spaces: 4 UTF-8 CRLF ENG IN 2013 27-09-2025

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
```

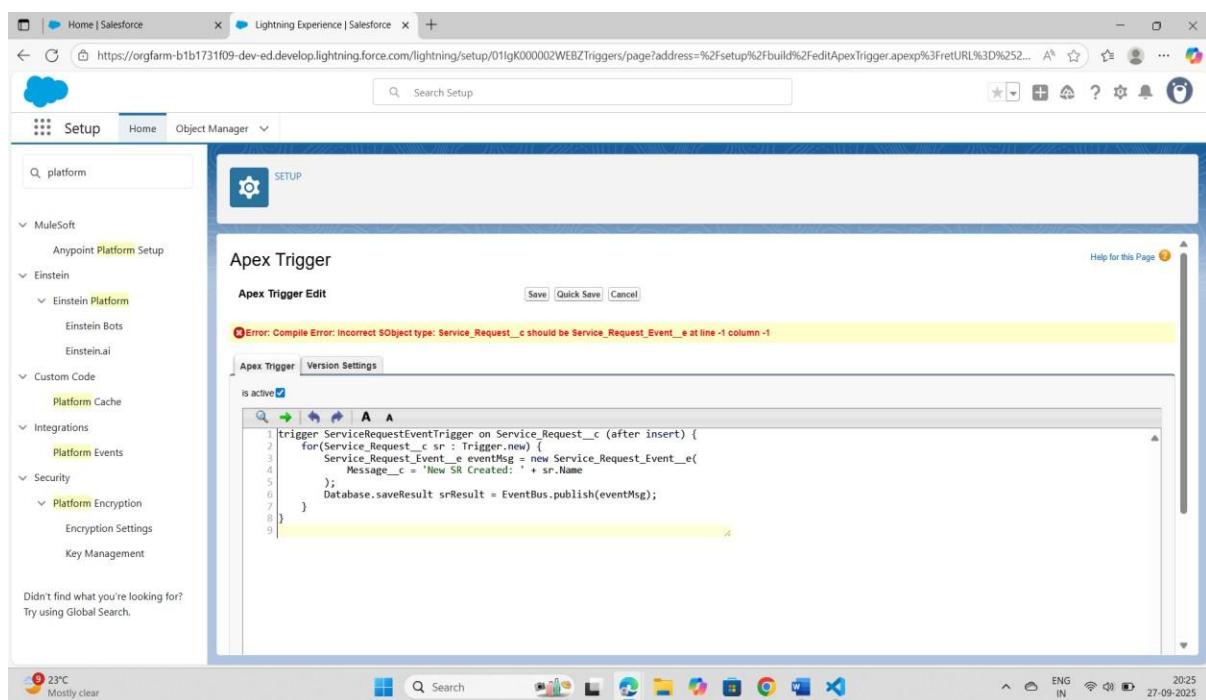
```
            c = 'New SR Created: ' + sr.Name
```

```
    );
```

```
    Database.saveResult srResult = EventBus.publish(eventMsg);
```

```
}
```

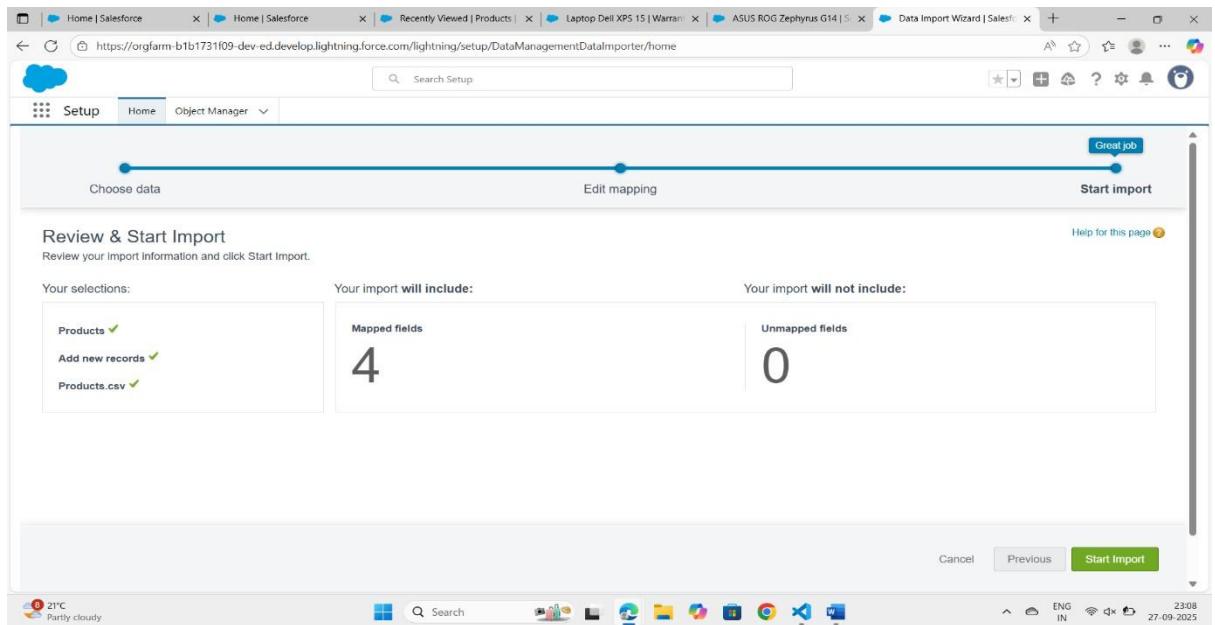
```
}
```



Phase 8: Data Management & Deployment

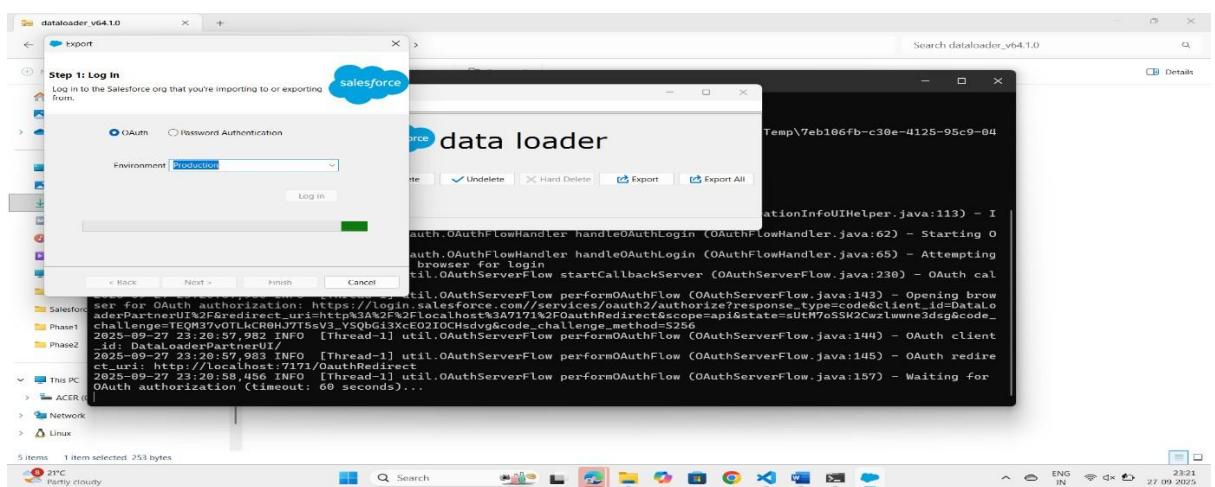
1. Data Import Wizard

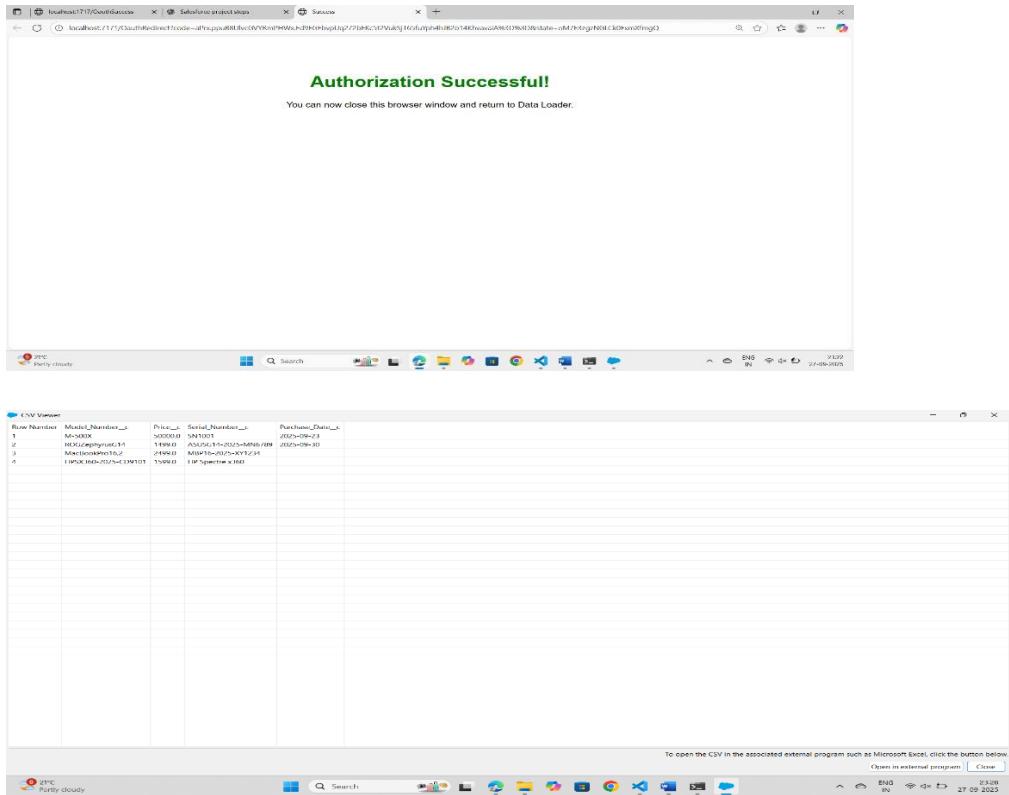
- App Launcher → Data Import Wizard.
- Choose Product_c.
- Upload CSV (Name, Serial_Number_c, Model_c, Price_c).
- Start Import.



2. Data Loader

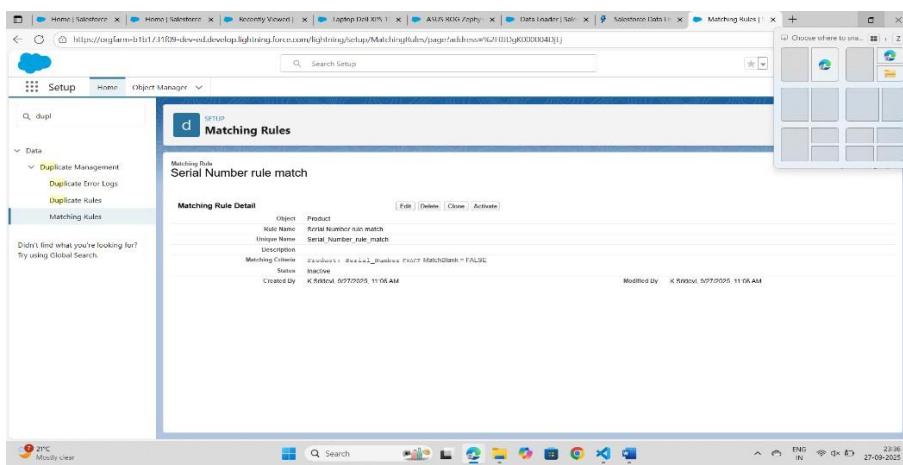
- Download Data Loader (if not installed).
- Login → Choose “Insert” → Select Service_Request_c.
- Map CSV columns to fields → Finish.





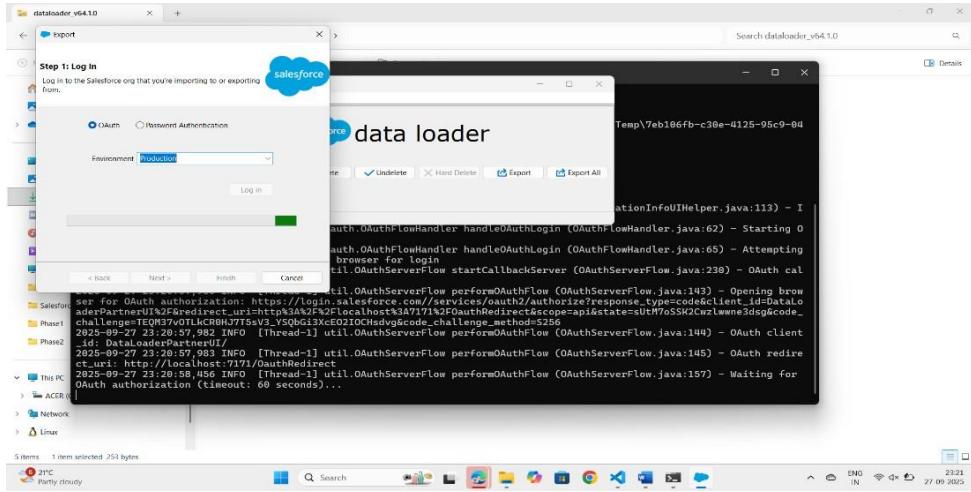
3. Duplicate Rules

- 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.

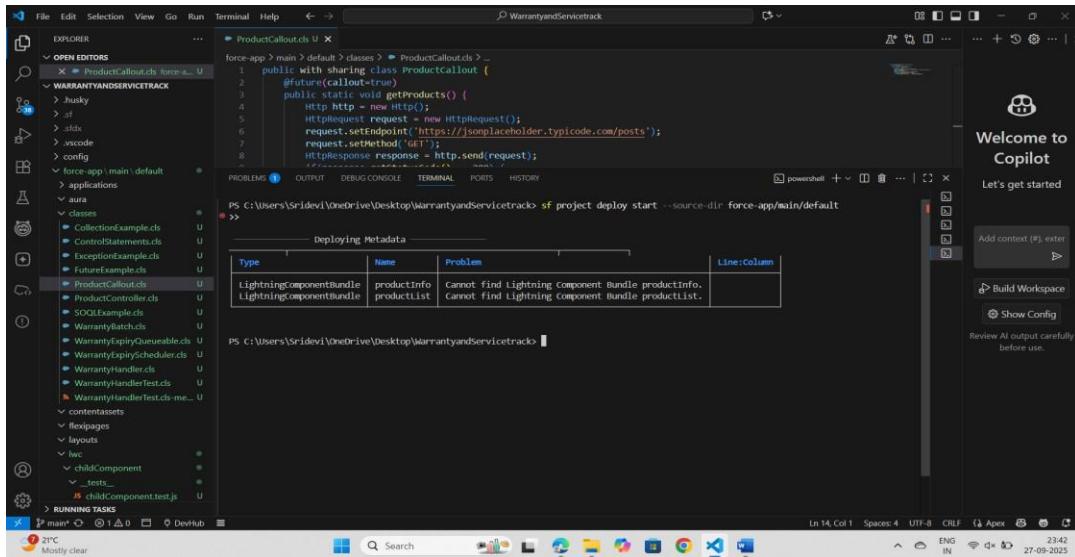


5. Change Sets

- 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

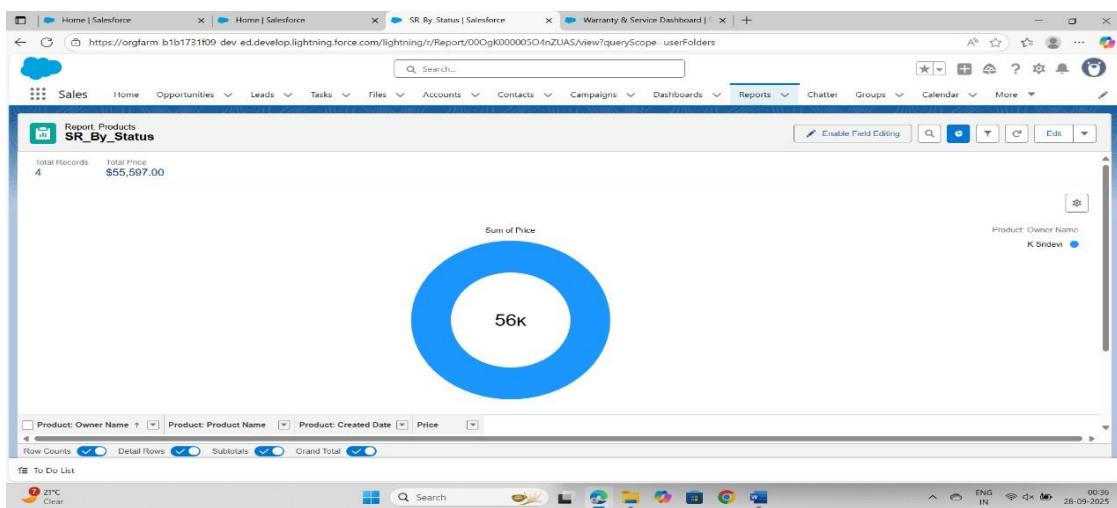


- Confirms metadata successfully deploy

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.

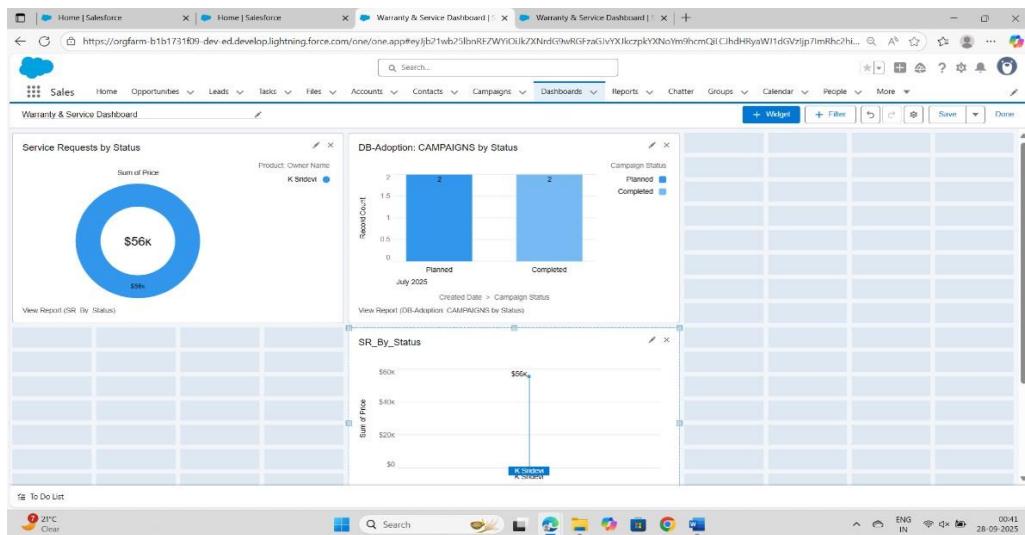


2) Create Report 2 — *Warranties Expiring Soon*

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:

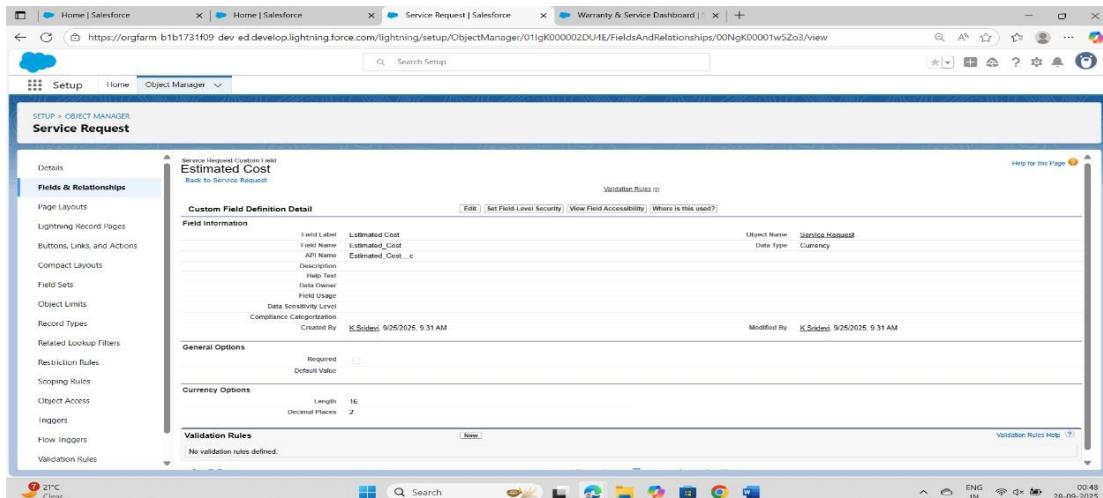
A — Organization-Wide Defaults (OWD)

- 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
Location 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
Inventory Count Assessments	Tab Hidden	Waitlists	Default Off
Inventory Count Plan Items	Tab Hidden	Web Store Inventory Sources	Tab Hidden
Joint Product Batch Items	Tab Hidden	Work Type Groups	Tab Hidden
Inventory 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
Equipment 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.

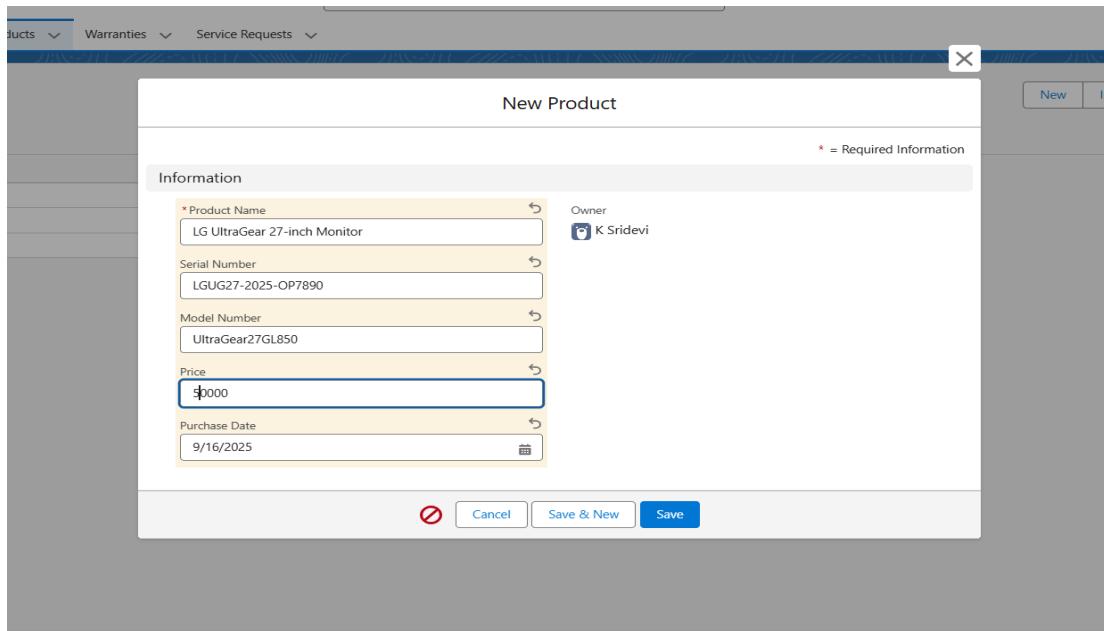
The screenshot shows the 'View Setup Audit Trail' page in the Salesforce Setup interface. The page lists 20 entries from the last six months. Each entry includes a timestamp, user, source namespace prefix, action details, section, and a 'Delegate User' link. The actions listed include various setup configurations such as Duplicate Rule creation, Matching Rule creation, Custom Objects creation, Security Controls creation, and External Credentials creation. The interface includes a left sidebar with navigation links like Feature Settings, Sales, Contact Intelligence, Setup, Lead Intelligence, View Setup, and Security. The top navigation bar shows tabs for Home, Object Manager, and the current page, 'View Setup Audit Trail'. The bottom status bar shows system information like battery level, signal strength, and the date (28-09-2025).

Date	User	Source Namespace Prefix	Action	Section	Delegate User
9/27/2025, 11:08:12 AM PDT	sdev@orphanage11859@apexforce.com		For duplicate rule Serial Number rule match: changed matching rules	Duplicate Rule	
9/27/2025, 11:08:12 AM PDT	sdev@orphanage11859@apexforce.com		Created new 011g00000ZDU0P Duplicate rule "Serial Number rule match". Set "Record Level Security" to "Enforce sharing rules"	Duplicate Rule	
9/27/2025, 11:09:28 AM PDT	sdev@orphanage11859@apexforce.com		For matching rule Serial Number rule match: added matching criteria where matching method is Exact, the field is Serial_Number and match blank fields is "Does Not Match If Null"	Matching Rule	
9/27/2025, 11:09:28 AM PDT	sdev@orphanage11859@apexforce.com		For matching rule Serial Number rule match: matching engine set to Exact Match Engine	Matching Rule	
9/27/2025, 11:09:28 AM PDT	sdev@orphanage11859@apexforce.com		Created new Product matching rule Serial Number rule match	Matching Rule	
9/27/2025, 7:52:02 AM PDT	sdev@orphanage11859@apexforce.com		Created custom field Message (Text) on Service_Request_Event	Custom Objects	
9/27/2025, 7:48:16 AM PDT	sdev@orphanage11859@apexforce.com		Created custom object Service_Request_Event	Custom Objects	
9/27/2025, 7:10:26 AM PDT	sdev@orphanage11859@apexforce.com		Remote Proxy Insert SampleAPI: https://jsonplaceholder.typicode.com	Security Controls	
9/27/2025, 7:09:11 AM PDT	sdev@orphanage11859@apexforce.com		Created a new parameter: ExternalCredential (Parameter Type: Authentication, External Credential: NoAuthCredential) for SampleAPI	Named Credentials	
9/27/2025, 7:09:11 AM PDT	sdev@orphanage11859@apexforce.com		Created a new parameter: Uri (Parameter Type: Uri, Parameter Value: https://jsonplaceholder.typicode.com) for SampleAPI	Named Credentials	
9/27/2025, 7:09:11 AM PDT	sdev@orphanage11859@apexforce.com		Created a new named credential: SampleAPI	Named Credentials	
9/27/2025, 7:08:12 AM PDT	sdev@orphanage11859@apexforce.com		Created a new parameter: Custom (Parameter Type: Authentication Protocol Variant, Parameter Value: NoAuthentication) for NoAuthCredential	External Credentials	
9/27/2025, 7:08:12 AM PDT	sdev@orphanage11859@apexforce.com		Created a new external credential: NoAuthCredential	External Credentials	
9/27/2025, 2:36:48 AM PDT	sdev@orphanage11859@apexforce.com		Changed Lightning Page: Warranty & Service Tracker UtilityBar	Lightning Pages	

Phase 10 — Final Presentation & Demo Day

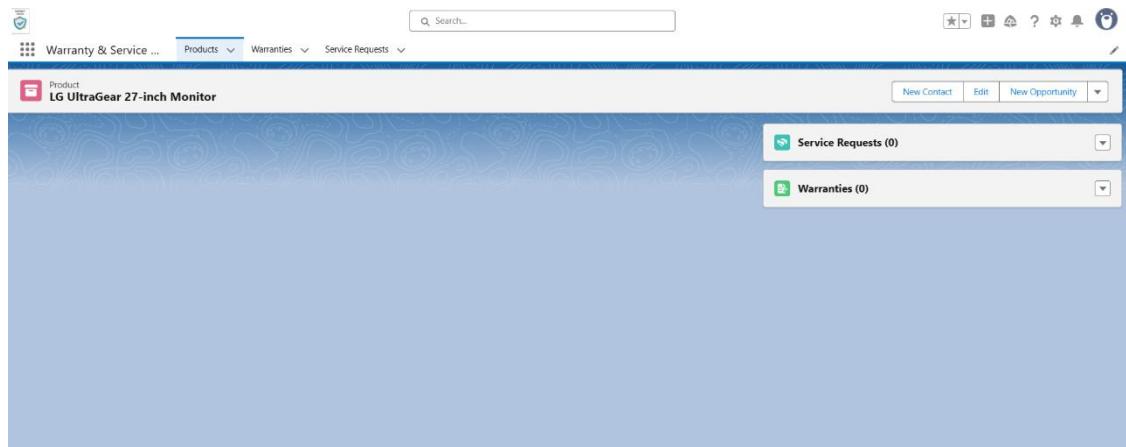
Test Case 1 — Create Product Record

- **Input (before save)**
 - Where: App Launcher → Products → **New Product** modal/form.
 - Capture the filled form showing Name, Serial Number, Model, Price.
 - **Filename:** Phase10_TC1_Product_Input.png
 - **Caption:** “Product creation form — input values (Name, Serial Number, Model, Price).”



The screenshot shows a 'New Product' modal window. At the top, there's a header bar with tabs for 'Products', 'Warranties', and 'Service Requests'. Below the header, the main title is 'New Product'. A note says '* = Required Information'. The form has a section titled 'Information' containing fields for Product Name (LG UltraGear 27-inch Monitor), Serial Number (LGUG27-2025-OP7890), Model Number (UltraGear27GL850), Price (\$000), and Purchase Date (9/16/2025). To the right of the form, there's an 'Owner' field with a user icon and the name 'K Sridevi'. At the bottom of the modal, there are three buttons: a red cancel button, a 'Save & New' button, and a blue 'Save' button.

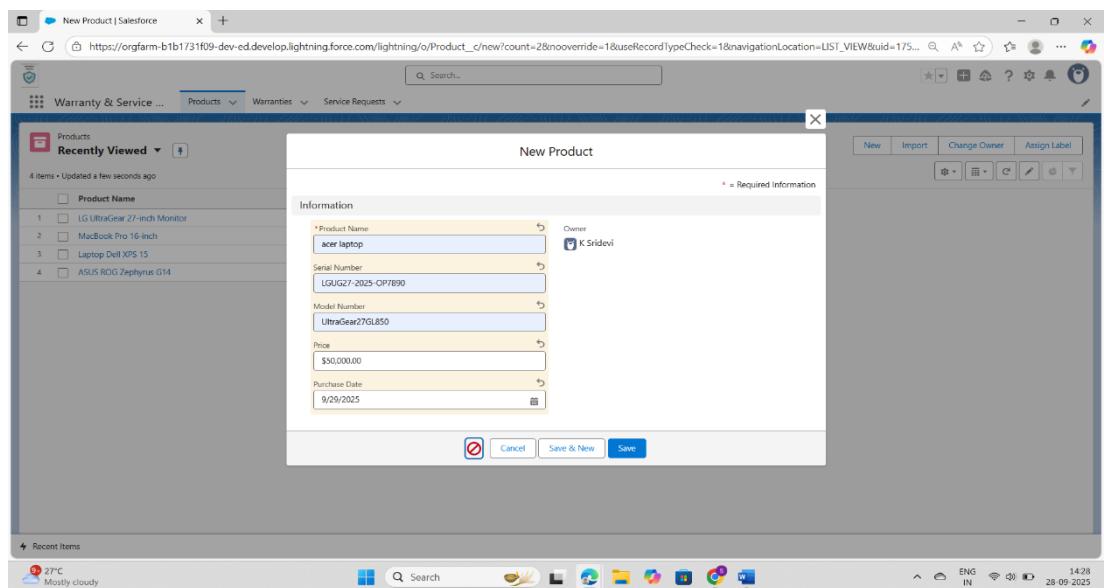
- **Output (after save)**
 - Where: Product record page (top shows record name).
 - Capture the record detail (Serial Number visible) and Related Lists.
 - **Filename:** Phase10_TC1_Product_Output.png
 - **Caption:** “Saved Product record showing Serial Number and related lists.”



Test Case 2 — Validation Rule (Duplicate Serial Number)

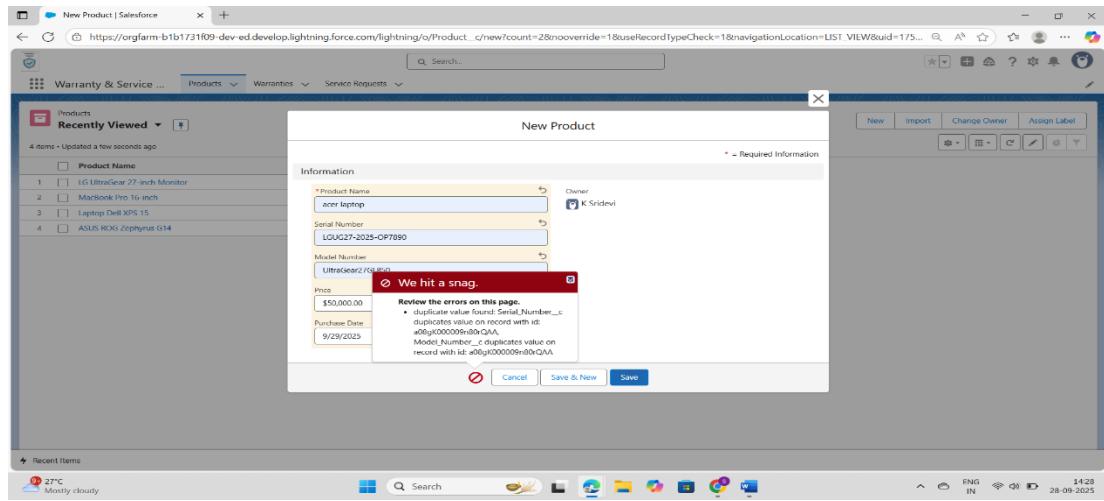
- **Input**

- Where: Products → New → enter **same Serial Number** as existing product.
- Capture the filled form just before Save.
- **Filename:** Phase10_TC2_DuplicateSerial_Input.png
- **Caption:** “Attempt to create a product with duplicate Serial Number.”



- **Output (error message)**

- Immediately after Save attempt: capture the validation error popup / inline error message.
- Ensure the error text is visible.
- **Filename:** Phase10_TC2_DuplicateSerial_Error.png
- **Caption:** “Validation error shown: Serial Number must be unique.”



- **Optional evidence**

- Setup → Object Manager → Product_c → Validation Rules → screenshot of the validation rule definition (rule name & formula).

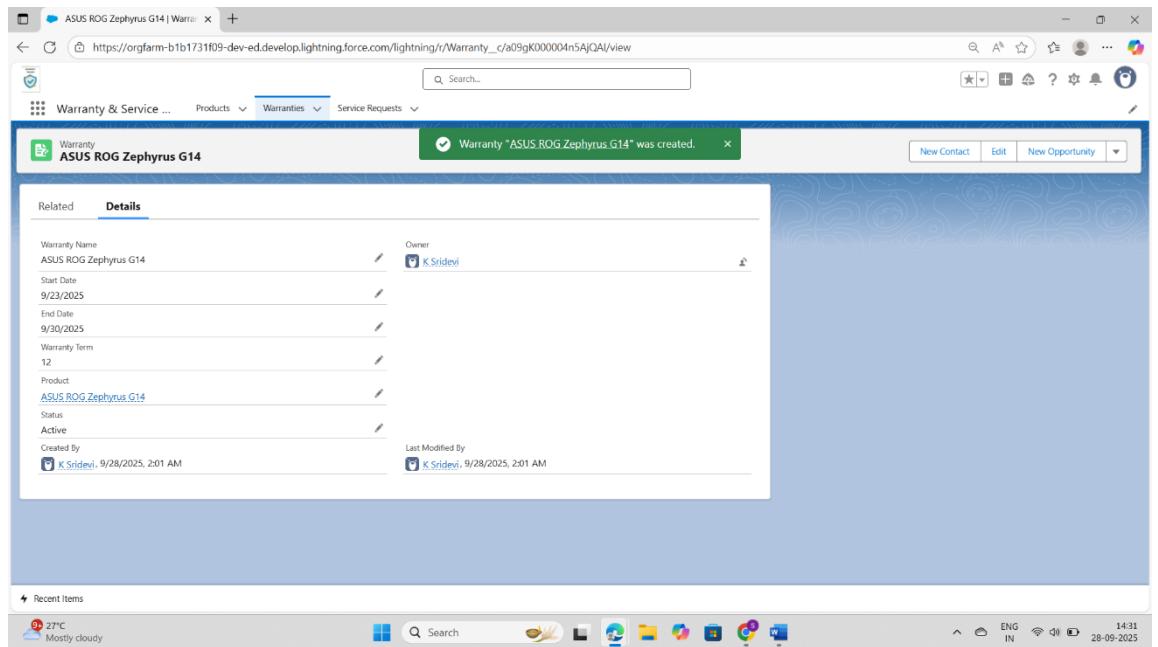
Test Case 3 — Create Warranty (link to Product)

- **Input**

- Where: Warranties → New → select Product, Start Date, Warranty Term.
- Capture the filled form.
- **Filename:** Phase10_TC3_Warranty_Input.png
- **Caption:** “Warranty creation form (linked Product, Start Date, Term).”

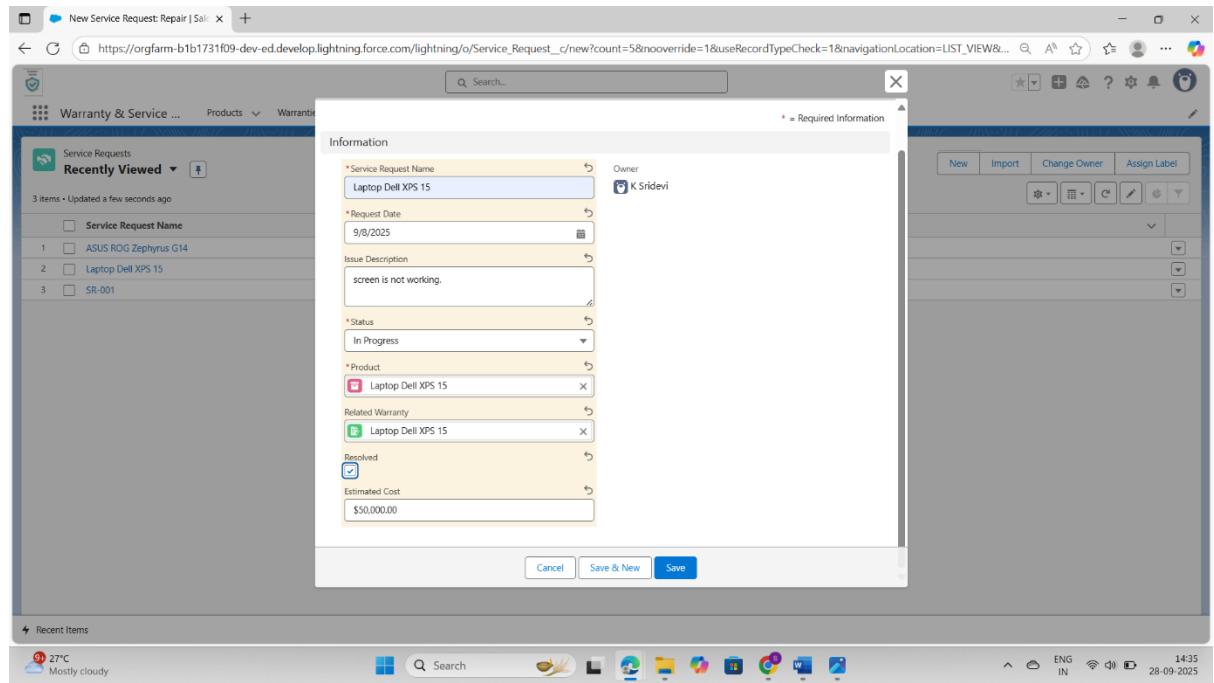
- **Output (after save)**

- Warranty record page showing End Date populated and Product lookup visible.
- Also show Product → Related List displaying this Warranty.
- **Caption:** “Warranty record with calculated End Date and linked Product (also visible in Product Related List).”



Test Case 4 — Service Request Approval Process

- **Input (create SR and submit)**
 - Where: Service Requests → New → fill Name, Product, Estimated Cost > 10000 → Save → show **Submit for Approval** dialog before submit.
 - Capture the filled SR form and the submit dialog.
 - **Caption:** “Service Request creation (Estimated Cost > 10000) and Submit for Approval dialog.”



- **Output 1 (Manager Approve list)**

- Login as Manager (or use “Login As”) → App Launcher → Approvals (or Home → Items to Approve) → show pending approval item.
- Capture the Approver screen showing the SR pending.

- **Output 2 (after approval)**

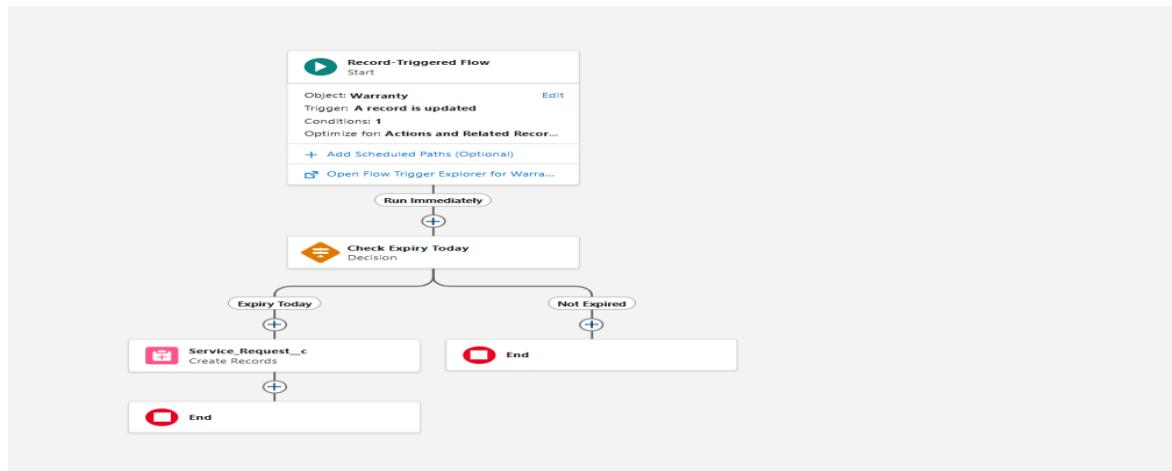
- After Approve action: open SR record → show Approval History related list and Status field updated to Approved.
- **Caption:** “Service Request approved — Approval History and Status updated.”

9/8/2025	<input type="button" value="Edit"/>	
Issue Description	screen is not working.	<input type="button" value="Edit"/>
Status	Closed	<input type="button" value="Edit"/>
Product	Laptop Dell XPS 15	<input type="button" value="Edit"/>
Related Warranty	Laptop Dell XPS 15	<input type="button" value="Edit"/>
Resolved	<input checked="" type="checkbox"/>	<input type="button" value="Edit"/>
Estimated Cost	\$50,000.00	<input type="button" value="Edit"/>
Created By	K Sridevi, 9/28/2025, 2:05 AM	Last Modified By K Sri

Test Case 6 — Flow Automation (Auto-create SR on Expiry)

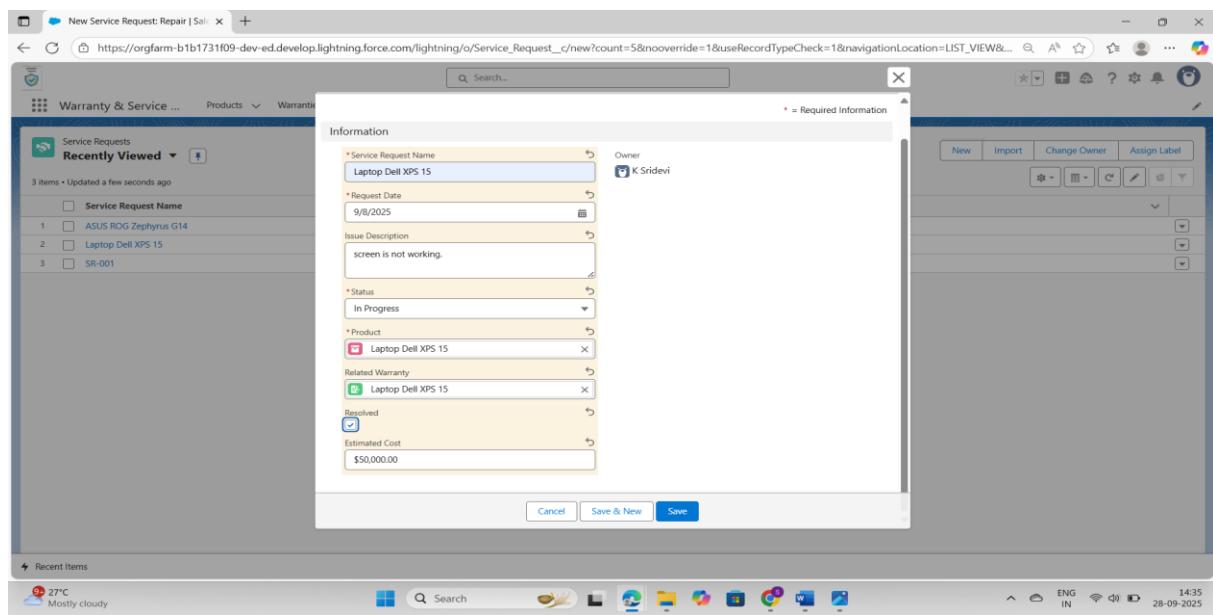
- **Flow config evidence (optional)**

- Setup → Flows → open the Flow → capture the Flow canvas showing the decision and Create Records element.



- **Input (trigger)**

- Edit Warranty → set End Date to today → Save (or run scheduled flow).
Capture the edit form.



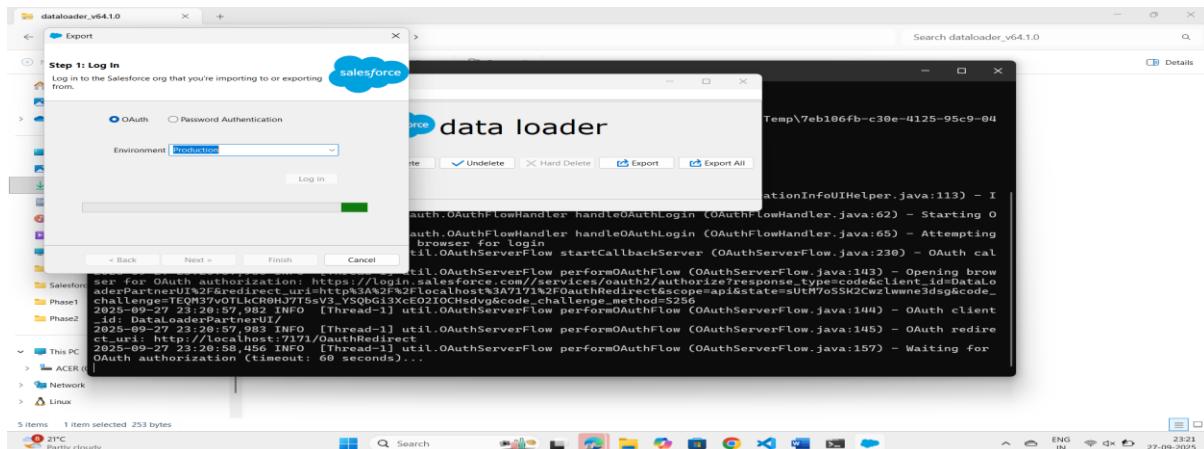
- **Output (SR auto-created)**

- Service Requests → filter by Name Auto-Generated Expiry Request or check Created Date → capture the SR record created (show Product lookup).
- **Caption:** “Service Request auto-created by Flow when Warranty expired.”

Test Case 7 — Trigger Execution (Apex trigger effects)

- **Input**

- Insert/Update Warranty record via UI or Data Loader (capture the create/edit form or the Data Loader mapping screen).



- **Output (trigger action)**

- Show resulting state: e.g., End_Date set, related fields updated, or new records created.

Test Case 8 — Report Generation & Dashboard updates

- **Input (report filters)**

- Reports → open SR_By_Status builder → show filters applied (date range etc.).

A screenshot of a report titled 'Report: Products SR_By_Status'. The report interface includes a header with the report name and a large blue circular graphic on the right. Below the header is a table with four columns: 'Product: Owner Name', 'Product: Product Name', 'Product: Created Date', and 'Price'. The table lists five products, each with its owner, name, creation date, and price. At the bottom of the table, there are subtotal and total rows. The table has a light gray background with white text and blue links for product names.

- **Output (report results & chart)**

- Run report → show grouped results + chart. Capture full report view.
- Dashboard → open Warranty & Service Dashboard showing charts updated after creating records in earlier tests.
- **Filename (report):** Phase10_TC8_Report_Output.png
- **Filename (dashboard):** Phase10_TC8_Dashboard_Output.png
- **Caption:** “Report results and Dashboard charts reflecting test data.”

