SOLAR ENERGY MANAGEMENT SYSTEM BASED ON CRM

Problem Statement – Solar Energy Management System (Salesforce CRM)

The solar energy sector is rapidly expanding as businesses and households shift to renewable power. However, companies face significant challenges:
Tracking customer leads and projects – from initial inquiry to final installation.

- **Managing site surveys and proposals** – often handled manually, leading to inefficiency.
- **Monitoring installations and energy generation** – ensuring panels deliver expected output.
- **Coordinating service and maintenance** – preventing downtime and ensuring warranty compliance.
- **Billing and subsidies** – handling payments, government incentives, and ROI reports.
- **Customer communication** – lack of timely updates on installation status, energy performance, and service schedules.

The **Solar Energy Management System on Salesforce CRM** solves these problems by offering a centralized platform that:

- Automates the end-to-end customer journey (lead → installation → service).
- Tracks solar panel installations, warranties, and maintenance schedules.
- Integrates with IoT devices to capture **real-time energy generation** data.
- Provides **dashboards and reports** on energy savings, revenue, and performance.
- Enhances customer satisfaction with **self-service portals, notifications, and transparency**.

This project aims to improve **operational efficiency, customer engagement, and sustainability tracking** for solar companies by leveraging Salesforce CRM.

Phase 1: Problem Understanding & Industry Analysis (Elaborated)

Goal: Understand what we are building, who needs it, and why it matters.

1. Requirement Gathering

We interact with key stakeholders to capture pain points and needs:

- **Company Manager:** Wants to see revenue, installations, and energy generation reports.
- **Sales Agents:** Need tools to track leads, convert them into opportunities, and prepare proposals.

  Installation Team: Needs scheduling support and real-time task assignments.
- **Service Engineers:** Want a system to log maintenance, complaints, and AMC visits.
- **Customers:** Expect transparent billing, ROI insights, and service reminders.

Example Requirements:

- Track all **solar installations** with status (Pending, In Progress, Completed, Under Maintenance).
- Allow sales team to **book site surveys and generate proposals**.
- Prevent duplicate installations for the same site/customer.
- Generate **revenue, subsidy, and energy performance reports**.
- Notify customers about **maintenance schedules, billing, and panel efficiency**.

2. Stakeholder Analysis

We define user groups in the CRM:

- **Admin** – manages Salesforce setup, roles, permissions.
- **Sales Agents** – create/manage leads, prepare proposals, close deals.
- **Installation Team** – executes panel installations, updates status.
- **Service Engineers** – manage warranty claims, repairs, AMC visits.
- **Manager** – approves discounts, monitors KPIs via dashboards.
- **Customer Service** – handles queries, bills, and service tickets.
- **Customers (Portal Users)** – track installation, bills, energy savings, and request service.

3. Business Process Mapping

The **end-to-end solar business flow** inside Salesforce looks like this:

1. Customer shows interest → Lead created.
2. Site Survey scheduled → Feasibility report prepared.
3. Proposal generated → If approved, move to installation stage.
4. Installation scheduled → Status updated in CRM.
5. Energy Output tracked (via IoT integration).
6. Billing and subsidy claim processed.
7. AMC/Maintenance scheduled automatically.
8. Customer notified via email/SMS/portal updates.

4. Industry-Specific Use Case Analysis

Solar industry challenges:

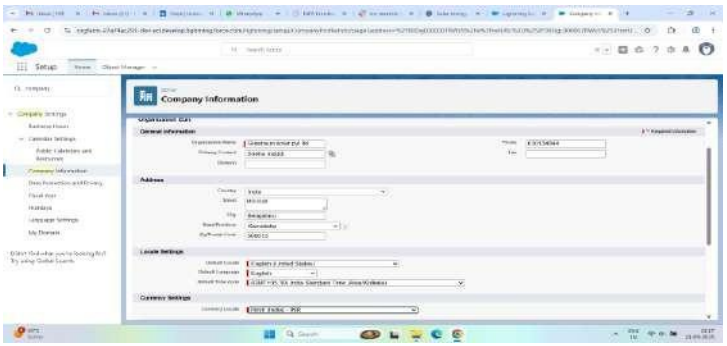
- **Government regulations & subsidies** vary across regions → CRM must handle approvals.
- **Energy Monitoring** → Customers demand ROI visibility (kWh saved, CO_2 reduction).
- **Maintenance cycles** are essential (panels lose efficiency if not cleaned).
- **Customer engagement** → Without reminders, service requests pile up.
- Installation tracking.
- Automated approvals for subsidies.
- Integration with smart meters.
- Notifications for service schedules.

5. AppExchange Exploration

- Salesforce AppExchange has energy-related apps (like **Salesforce Energy & Utilities Cloud**).
- However, most are **complex and enterprise-level**.
- For this project, we will **build a custom lightweight Solar CRM** to practice Salesforce core concepts (Objects, Flows, Apex, LWC, Reports).

Step 1: Company Settings

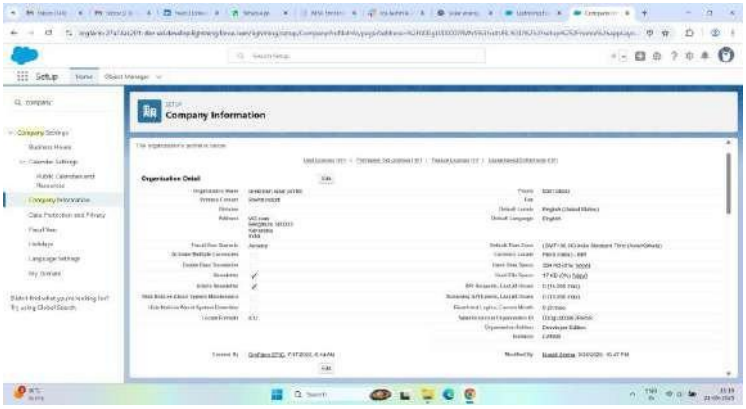
- Setup → Company Information
  - Company Name: *GreenSun Solar Energy Pvt. Ltd.*
  - Address: Bengaluru, Karnataka, India
  - Currency: INR (₹)
  - Locale: English (India)
  - Timezone: Asia/Kolkata
  - Fiscal Year: Apr–Mar
- **Business Hours**
  - Default: Mon–Sat, 9:00 AM – 6:00 PM
  - Support Hours (optional): 24x7 for Monitoring Team
- **Holidays**
  - Republic Day (26-Jan)
  - Holi (March)
  - Independence Day (15-Aug)
  - Gandhi Jayanti (2-Oct)
  - Diwali (Oct/Nov)
  - Christmas (25-Dec)



🔹 **Step 2: Users & Roles**

- Users to Create
  - Admin (full access)

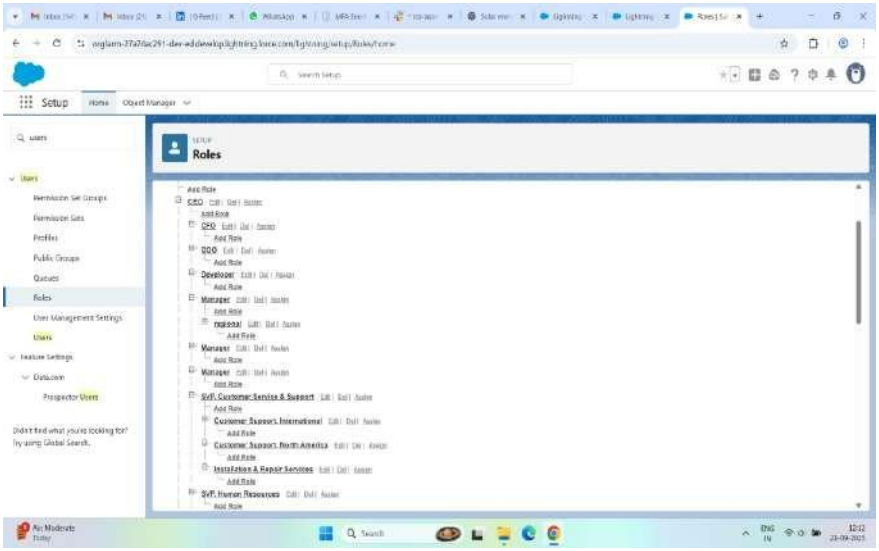
- Sales Rep (leads & opportunities)
- Project Manager (installation projects)
- Technician (service orders)
- Monitoring Operator (alerts & energy tracking)
- Finance Officer (contracts, invoices)



- **Role Hierarchy**
  - CEO
    - VP Sales → Sales Reps
    - VP Operations → Project Managers → Technicians
    - VP Monitoring → Monitoring Operators
    - VP Finance → Finance Officers

- **Profiles**
  - System Admin
  - Sales Profile
  - Project Manager Profile

- Technician Profile
- Monitoring Profile
- Finance Profile



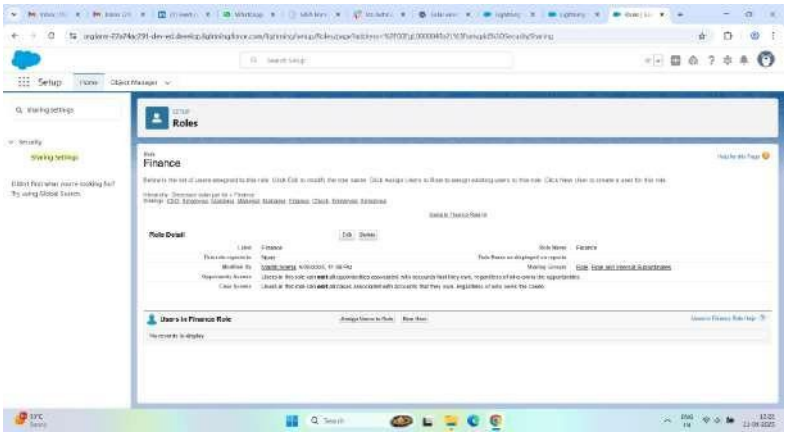
- **Permission Sets**
  - Advanced Reporting
  - Mobile Access
  - API Integration

🔹 Step 3: Security & Permissions

- Org-Wide Defaults (OWD)
  - Accounts → Public Read/Write
  - Solar Sites → Public Read Only
  - Projects/Service Orders/Contracts → Private
  - Energy Records → Public Read Only
- **Sharing Rules**
  - Service Orders → Technician Group
  - Monitoring Alerts → Monitoring Team
  - Projects → Regional Managers
- **Public Groups**

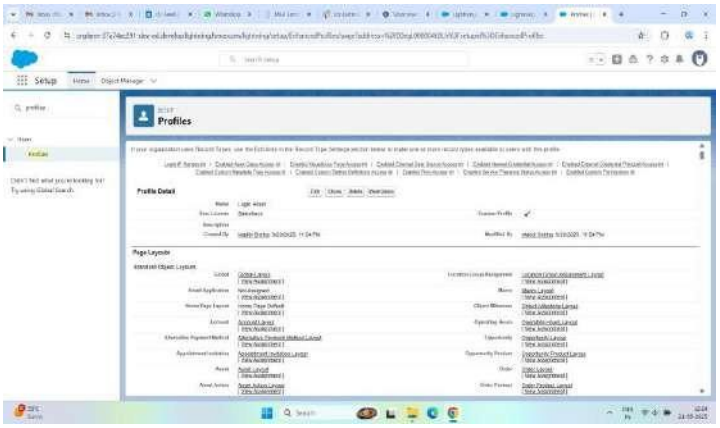
- Technician Group
- Monitoring Team
- Finance Team

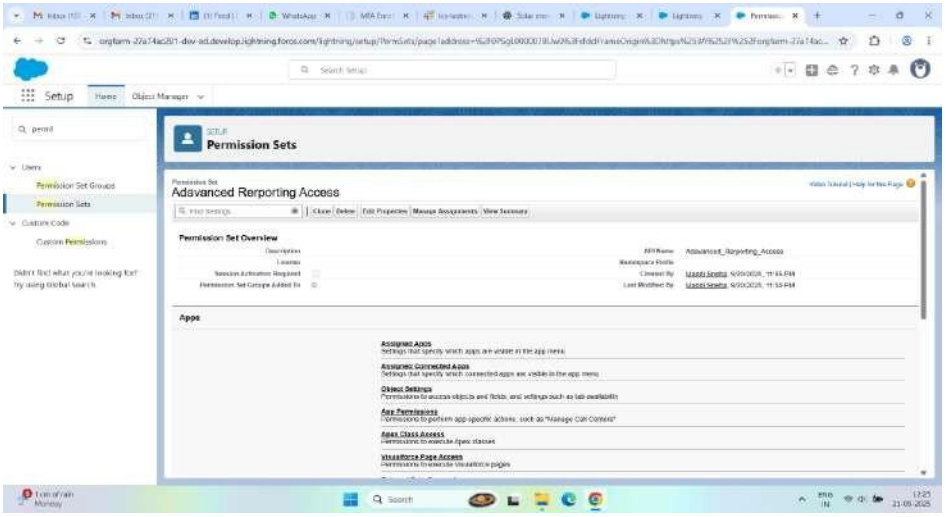
- **Login & Security Policies**
  - Login Hours → Technicians: 7 AM – 8 PM
  - Login IPs → Restrict Finance/Admin to office IPs
  - Enable 2FA (Two-Factor Authentication)
  - Session Timeout = 30 mins



🔹 Step 4: Custom Solar Objects & Fields

- Objects to Create
  1. Solar Site → Capacity, Status, Commissioning Date
  2. PV Array → Panel Count, Orientation, Efficiency
  3. Inverter → Serial, Warranty, Status
  4. Battery → Capacity, Health, Install Date
  5. Installation Project → Budget, Manager, Start/End Dates
  6. Service Order → Issue, Technician, Priority, Status
  7. Energy Record → kWh Generated, Peak Power
  8. Monitoring Alert → Type, Severity, Status
  9. Contract → SLA, Payment Terms, Start/End Dates
- **Record Types**

- Projects → Residential vs Commercial
- Service Orders → Preventive vs Corrective
- 

- **Page Layouts**
  - Technician Layout → Issues & Checklist
  - Project Manager Layout → Budget, Timeline
  - Monitoring Layout → Alerts & Energy Records
  - 

🔹 Step 5: Automation

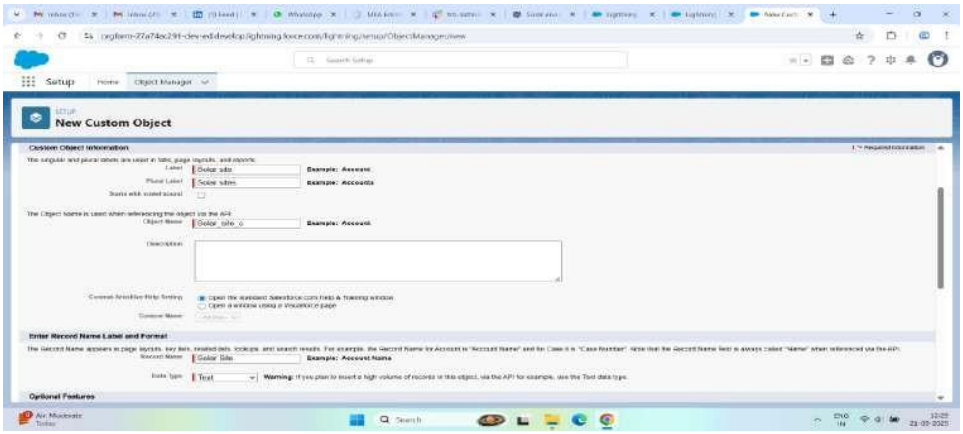
- Queues
  - Technician Queue → For Service Orders
  - Monitoring Queue → For Alerts
- Approval Process

- Installation Projects with Budget > ₹10 Lakhs → Approval from Finance + VP Ops

- Flows
  - After commissioning → Auto-create first inspection Service Order
  - If daily energy < expected → Auto-create Monitoring Alert
- Email Alerts
  - Notifications for Approvals
  - Technician Service Assignment emails
  - Monitoring Alerts to Operators

🔹 Step 6: Reports & Dashboards

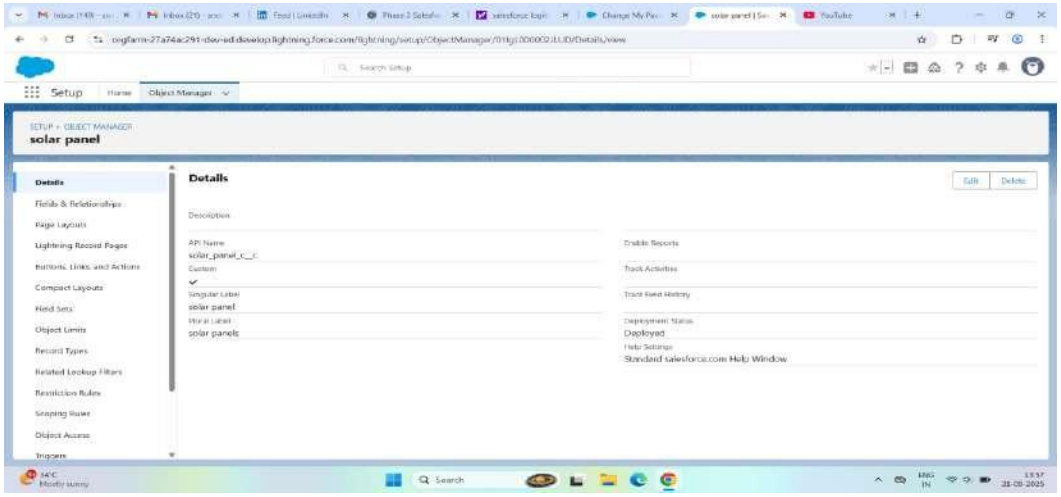
- Reports
  - Sales → Leads to Deals
  - Operations → Active Projects, Service Orders Aging
  - Monitoring → Energy vs Expected, Site Uptime
  - Finance → Revenue, ROI, Pending Invoices
- Dashboards
  - Executive Dashboard → Overview
  - Operations Dashboard → Projects & Technicians
  - Monitoring Dashboard → Energy & Alerts
  - Finance Dashboard → Revenue & Invoices



Step 1 — Create the Solar Panel & Energy Record objects

A. Create Solar Panel (custom object)

1. Click ⚙ → **Setup**.
2. In the **Quick Find** box type **Object Manager** → click **Object Manager**.
3. Click **Create** → **Custom Object** (or **New Custom Object**).
4. Fill fields:
    - **Label:** Solar Panel
    - **Plural Label:** Solar Panels
    - **Object Name:** Solar_Panel (auto)
    - **Record Name:** choose **Text**, Label it Serial Number (so you can type SP-001 etc.)
    - Check **Allow Reports**, **Track Activities** (optional but handy).
    - Optionally check **Allow Search**, **Allow in App Launcher**.
5. Click **Save**.

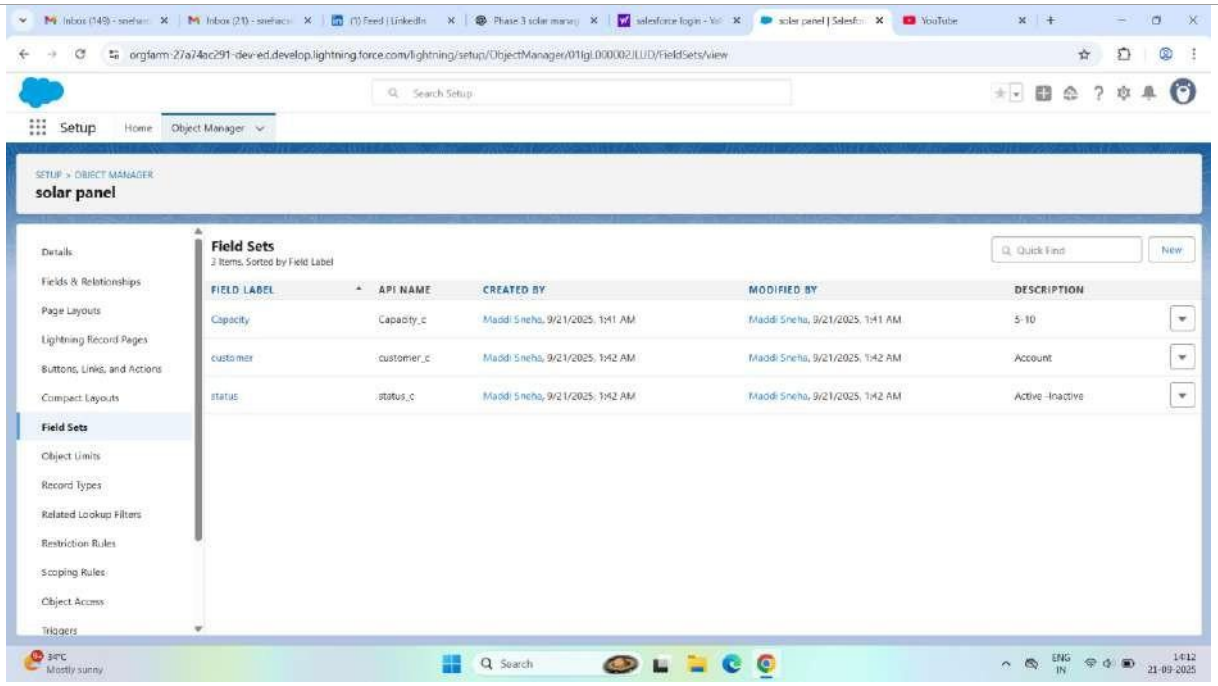


B. Create Energy Record (custom object)

1. From **Object Manager** click **Create** → **Custom Object** again.
2. Fill:
    - **Label:** Energy Record
    - **Plural:** Energy Records

- **Record Name:** choose **Auto Number**, set format ENR-{0000}.
- Check **Allow Reports**, **Track Activities**.

3. Click **Save**.



Step 2 — Add the essential fields (Solar Panel + Energy Record)

For each field: after choosing the field type, click **Next → Set Field-Level Security → Add to Page Layout → Save**.

Solar Panel fields (minimal required)

1. **Capacity (kW)**
   - Type: **Number** → Length: 5, Decimal Places: 2.
   - Label: Capacity (kW).
2. **Status**
   - Type: **Picklist**.
   - Label: Status.
   - Values (enter each on new line):
   - Active
   - Inactive

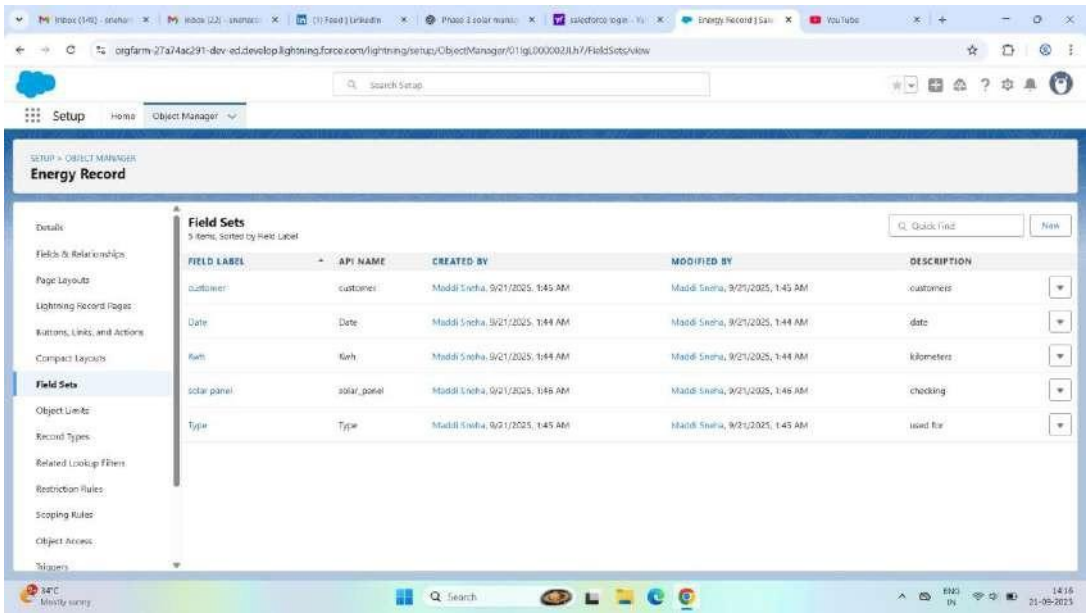
- Maintenance

3. **Customer (lookup to Account)** — this creates the relationship
    - Type: **Lookup Relationship** → Related To: **Account**.
    - Field Label: Customer.
    - Optional: set lookup dialog options (Make lookup required? leave optional for test).
4. (Optional) **Installation Date**
    - Type: **Date** → Label: Installation Date.

Energy Record fields (minimal required)

1. **Date**
    - Type: **Date** → Label: Date.
2. **kWh**
    - Type: **Number** → Length 8, Decimal 2. → Label: kWh.
3. **Type**
    - Type: **Picklist** → Label: Type → Values:
    - Production
    - Consumption
4. **Solar Panel (lookup)**
    - Type: **Lookup Relationship** → Related To: **Solar Panel**.
    - Field Label: Solar Panel.
5. **Customer (lookup to Account)**
    - Type: **Lookup Relationship** → Related To: **Account**.
    - Field Label: Customer.

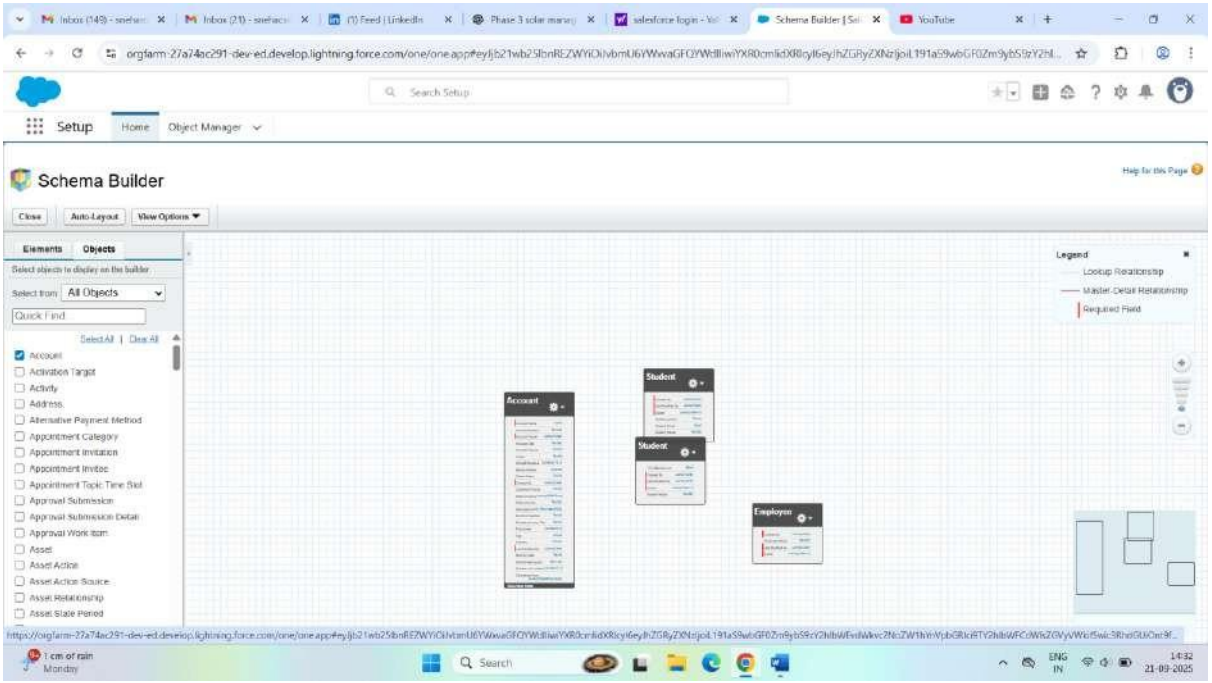
- field to layout.



Step 3 — Relationships (confirm & tweak)

We used **Lookup** relationships in Step 2. Confirm and adjust behavior:

1. Go to **Object Manager → Solar Panel → Fields & Relationships**.
    - Confirm you see Customer (Lookup to Account). Click it to edit settings (make required or not).
2. Go to **Object Manager → Energy Record → Fields & Relationships**.
    - Confirm you see Solar Panel (Lookup to Solar Panel) and Customer (Lookup to Account).
3. (Optional) If you want **Energy Records** to be deleted when the Solar Panel is deleted, you could change to **Master-Detail** — but that requires the child (Energy Record) to have a master at creation time. For simplicity, leave as **Lookup**.
4. Visual check: **Setup → Schema Builder**
    - Open Schema Builder, add Account, Solar Panel, Energy Record and you'll see the lines showing the lookups.



Step 4 — Page Layouts (simple & useful)

Make it easy for users to add and view related records.

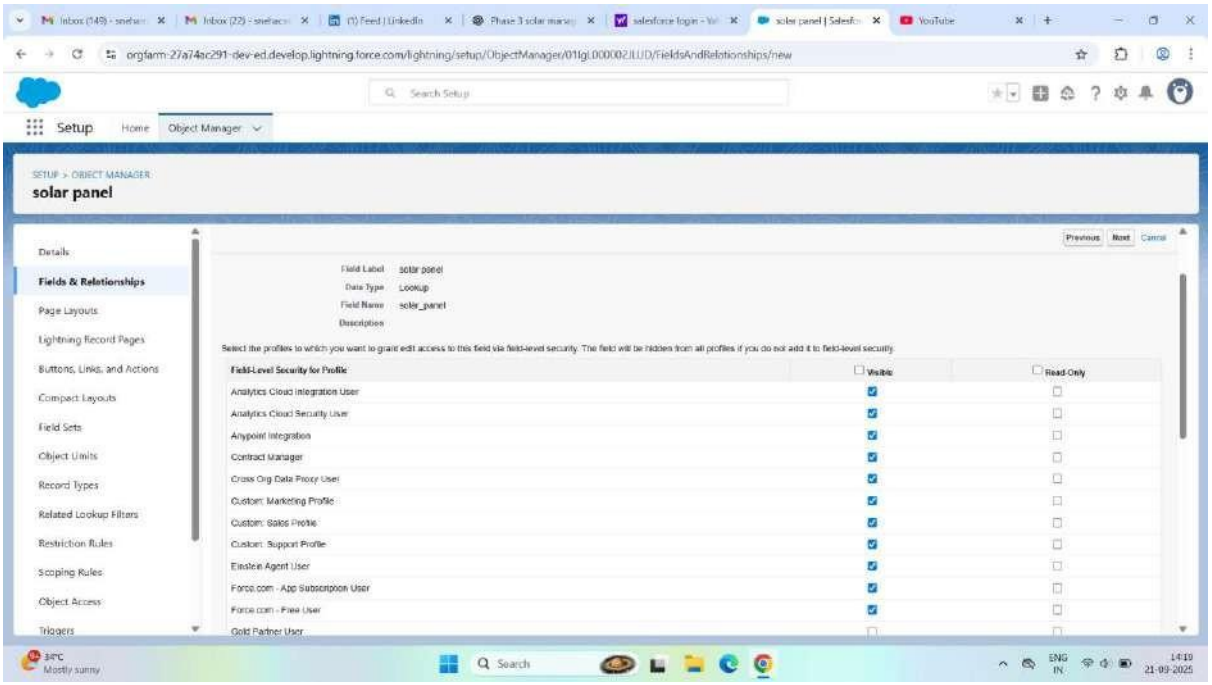
A. Account (Customer) — add related lists

1. Setup → Object Manager → **Account** → **Page Layouts** → click the layout name (e.g., Account Layout) → **Edit**.
2. In the layout editor, find **Related Lists** on the right.
3. Drag **Solar Panels** into the Related Lists area.
4. Drag **Energy Records** into the Related Lists area.
5. Optional: click the **wrench** on a related list → choose visible **Columns**:
    - For Solar Panels: Serial Number, Capacity (kW), Status.
    - For Energy Records: Date, Type, kWh, Solar Panel.
6. Ensure **New** is present under Buttons so users can create records from the related list.
7. Click **Save**.

B. Solar Panel — show core fields + related records

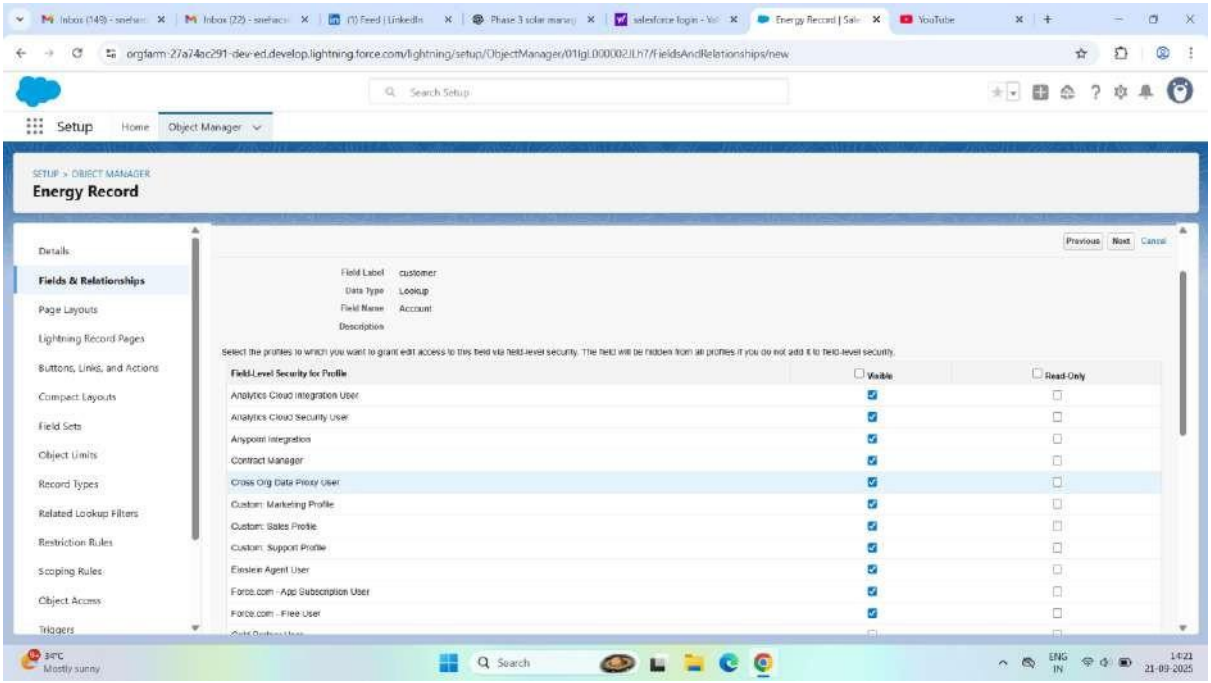
1. Object Manager → **Solar Panel** → **Page Layouts** → Edit the main layout.

2. Make sure these fields are present near the top: Serial Number, Capacity (kW), Status, Customer, Installation Date.
3. Drag **Energy Records** (related list) onto the page so you can see production/consumption from the panel.
4. Optionally configure the related list columns (wrench) to show Date, Type, kWh.
5. Save.



C. Energy Record — tidy form for data entry

1. Object Manager → **Energy Record** → **Page Layouts** → Edit.
2. Put fields in a logical order: **Date**, **Type**, **kWh**, **Solar Panel**, **Customer**, **Notes**.
3. Save.

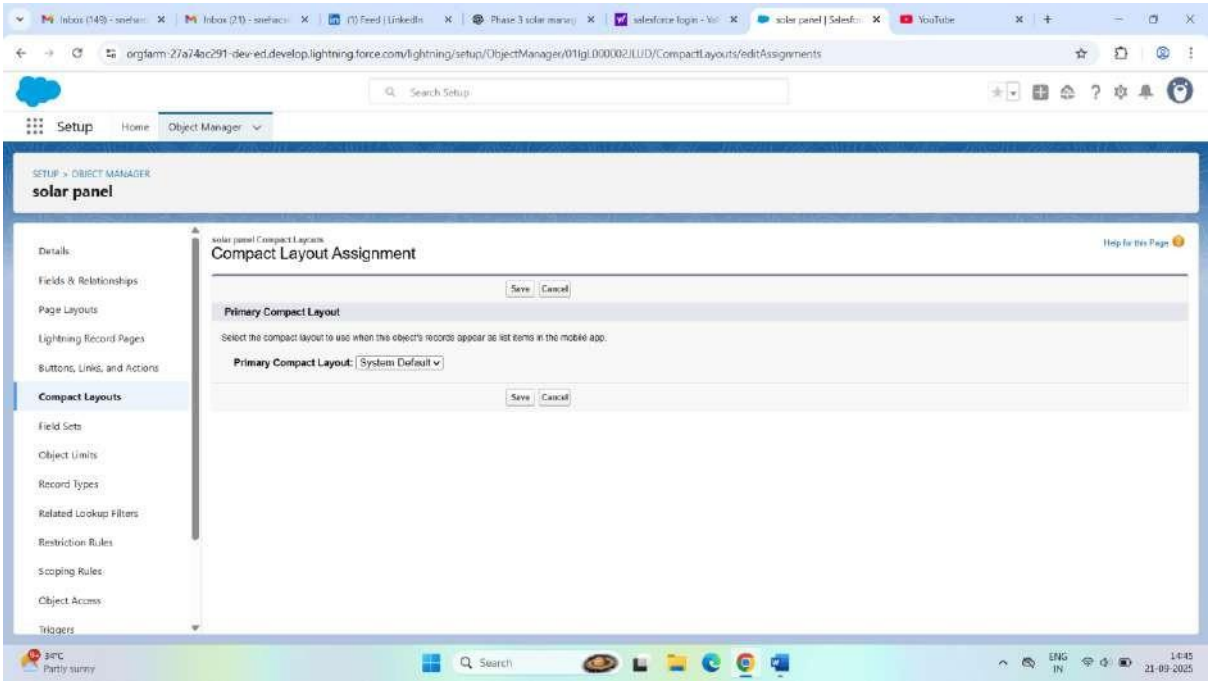
4. 

Step 5 — Compact Layouts (so key info shows in top card & mobile)

Compact layouts control the record summary that appears at the top of record pages and on mobile.

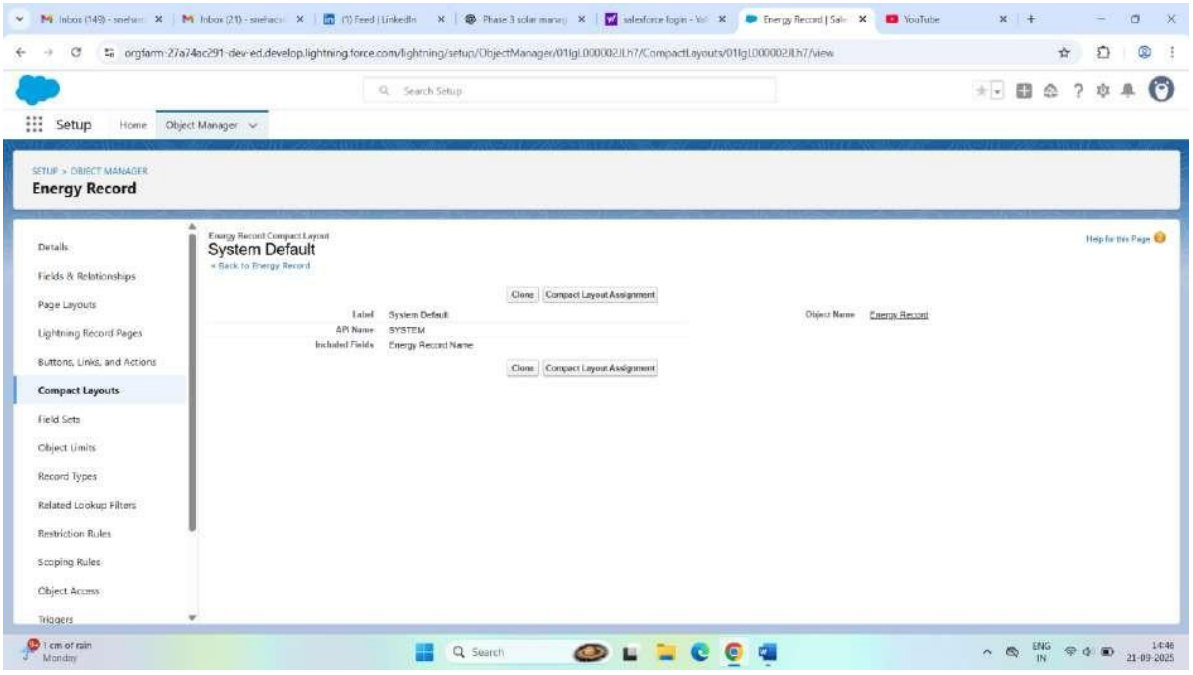
Solar Panel compact layout

1. Setup → Object Manager → **Solar Panel** → **Compact Layouts** → **New**.
    - Name: Solar Panel Compact.
    - Select fields (order matters): Serial Number, Capacity (kW), Status, Customer.
2. Save.
3. Click **Compact Layout Assignment** → **Edit Assignment** → Set Solar Panel Compact as the Primary Layout → Save.

4. 

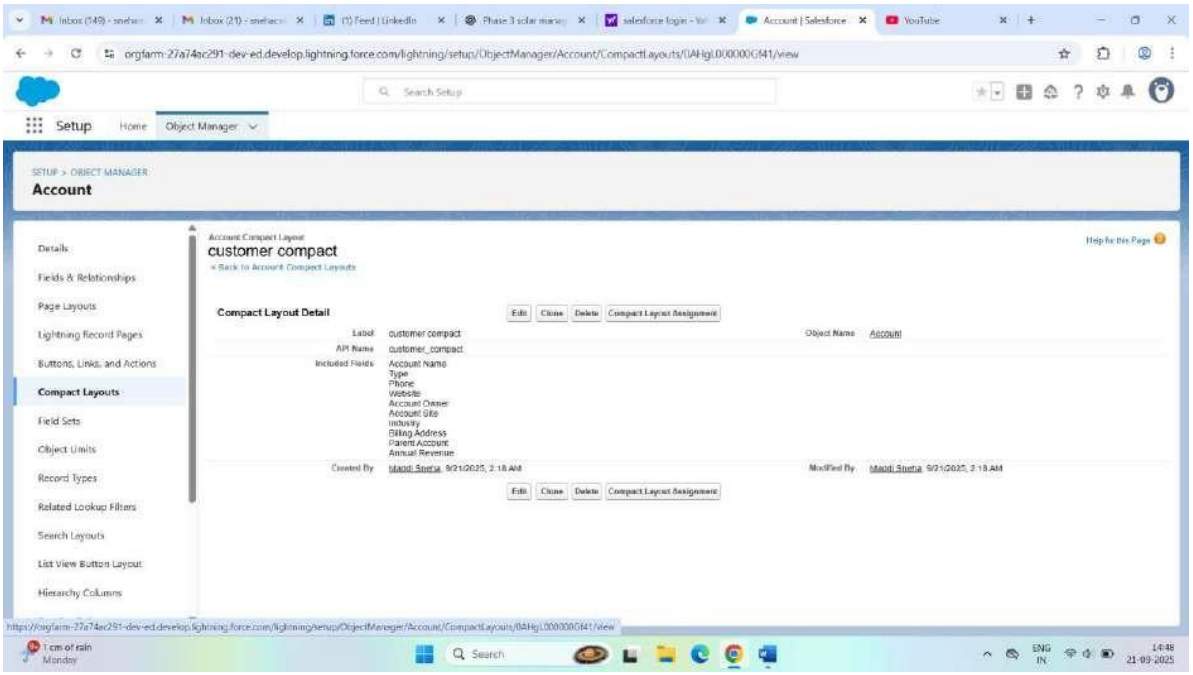
Energy Record compact layout

1. Object Manager → **Energy Record → Compact Layouts → New**.
    - Name: Energy Record Compact.
    - Fields: Date, Type, kWh, Solar Panel.
2. Save → Assign as Primary.



3. Account compact layout (optional)

1. Object Manager → **Account** → **Compact Layouts** → **New**.
    - Fields: Account Name, Phone, Primary Contact (if present).
2. Save → Assign.



Phase 4: Process Automation (Elaborate Version)

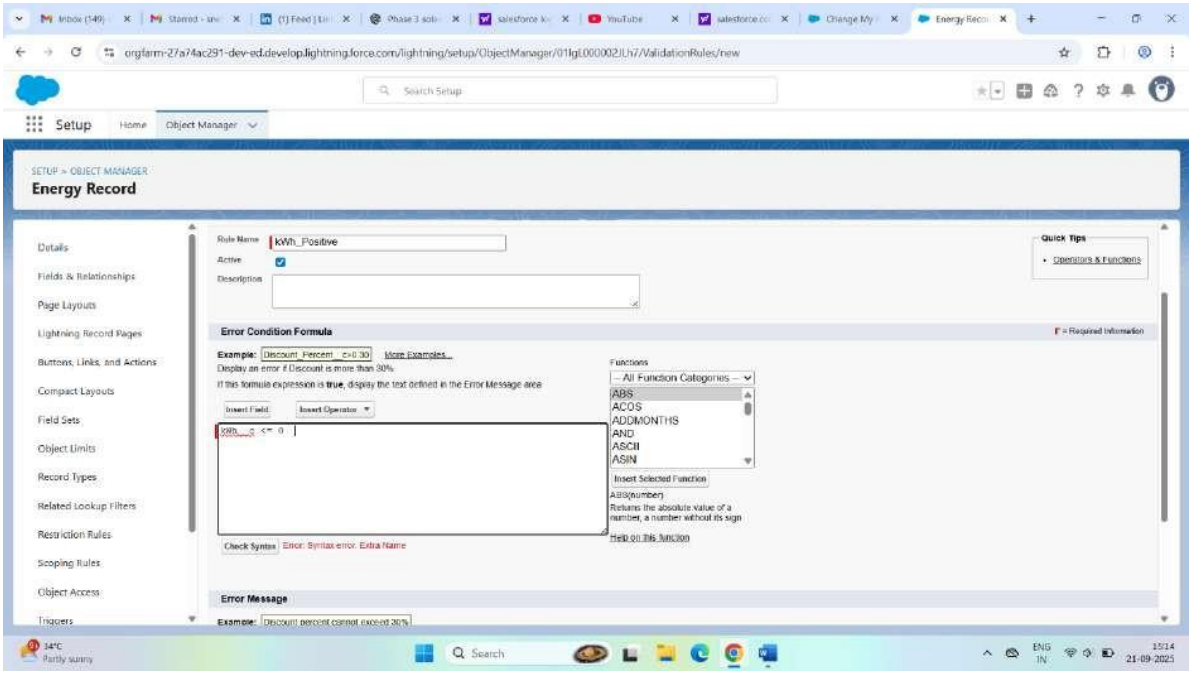
= **Goal:** Make Salesforce do repetitive work automatically — keep data clean, calculate totals, send alerts, and handle approvals.

1. Validation Rules (keep data clean)

Example: Energy Record kWh must be greater than 0.

Steps:

1. Go to **Setup → Object Manager → Energy Record → Validation Rules**.
2. Click **New**.
3. Fill in:
    - **Rule Name:** Positive_kWh
    - **Error Condition Formula:**
    - kWh _c <= 0
    - **Error Message:** Energy must be greater than 0
    - Location: choose field **kWh**.
4. Click **Save**.

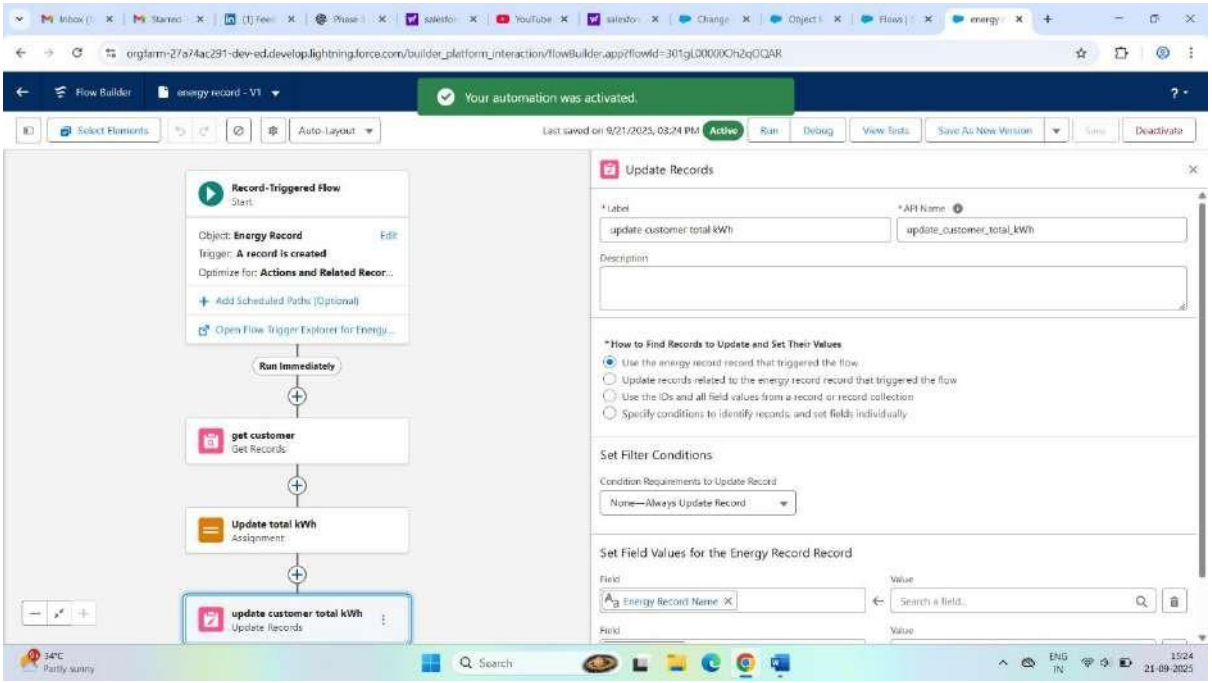


2. Flows (automate totals & updates)

Example: Update Customer's Total kWh automatically when an Energy Record is added.

Steps:

1. Go to **Setup → Flows**.
2. Click **New Flow → Record-Triggered Flow**.
3. Configure:
    - Object: **Energy Record**
    - Trigger: **When record is created**
    - Run: **After Save** (so you can update other records).
4. Click **+ Add Element → Get Records**.
    - Label: Get Customer
    - Object: **Account**
    - Condition: Id = {!$Record.Customer__c}
    - Store first record.
5. Click **+ Add Element → Assignment**.
    - Label: Update Total kWh
    - Variable: {!Customer.Total_kWh__c} = {!Customer.Total_kWh__c} + {!$Record.kWh__c}
6. Click **+ Add Element → Update Records**.
    - Record: Customer from Get Records.
    - Field: Total kWh = assigned value.
7. Save → Label: Update Customer Total kWh → Activate.

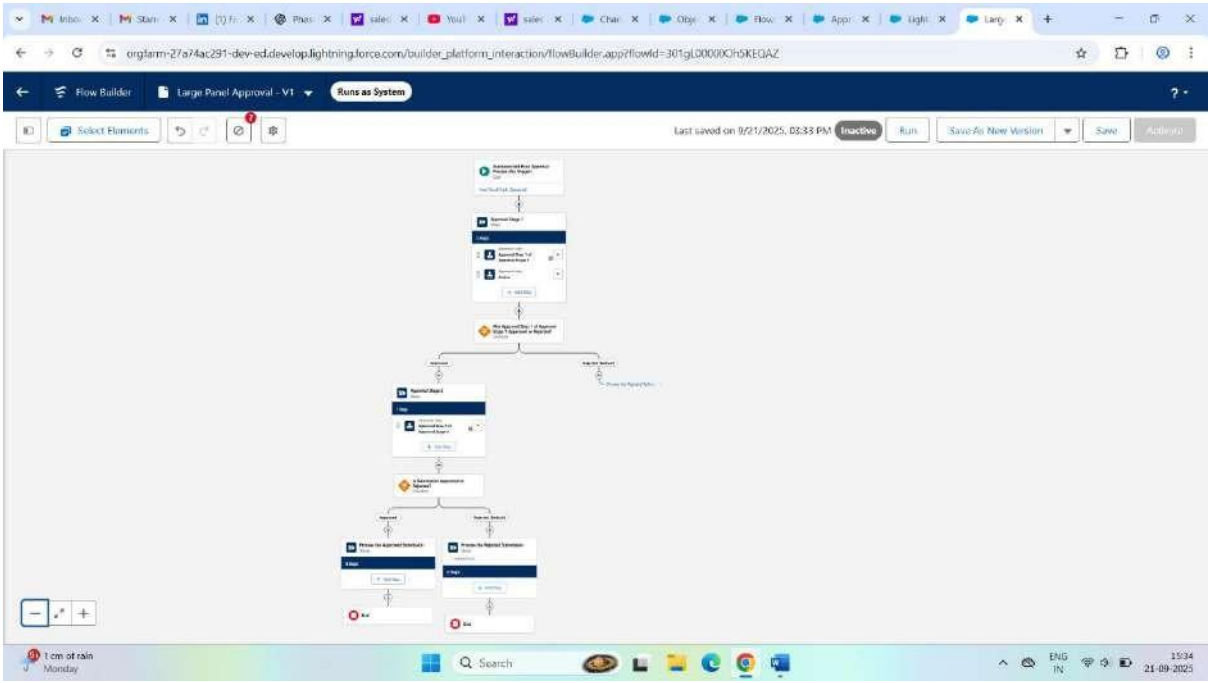


3. Approval Process (for expensive/large panels)

Example: If Solar Panel capacity > 10 kW, manager must approve before status is “Active”.

Steps:

1. Setup → **Approval Processes** → choose object **Solar Panel**.
2. Click **New Approval Process** → **Jump Start Wizard** (easy mode).
3. Enter Name: Large Panel Approval.
4. Criteria: Capacity__c > 10.
5. Select Approver: choose a Manager user.
6. Final Actions:
    - On Approval → Update field Status = Active.
    - On Rejection → Update field Status = Pending.
7. Save → Activate.

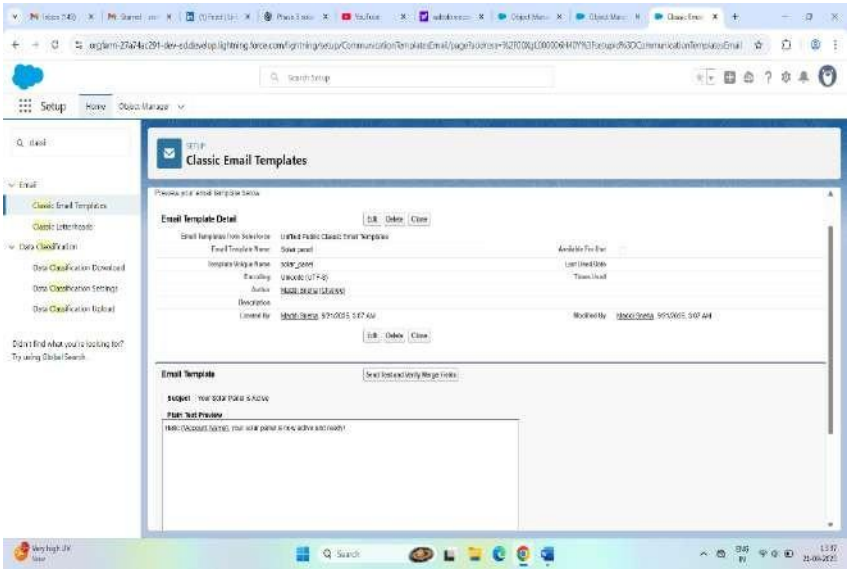


4. Email Alerts (auto-send emails)

Example: Notify customer when their panel is installed (status = Active).

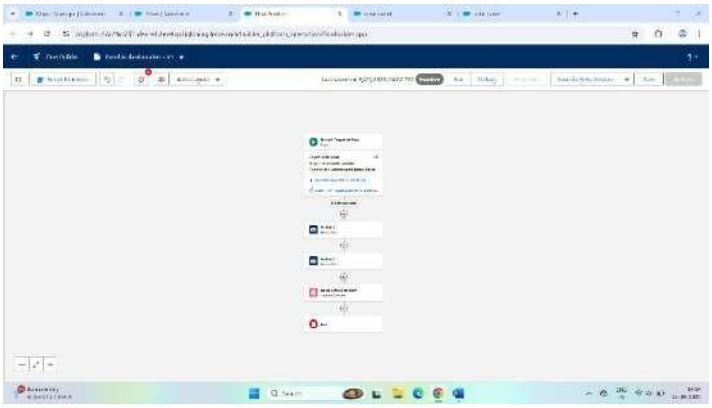
Steps:

1. Setup → **Classic Email Templates → New Template**.
    - Choose Text (simple).
    - Subject: Your Solar Panel is Active
    - Body: Hello {!Account.Name}, your solar panel is now active and ready!
    - Save.

- 

2. Setup → **Email Alerts** → **New**.
    - Description: Panel Activation Alert
    - Object: **Solar Panel**
    - Recipient: **Account Email** (lookup through Customer field).
    - Template: the one you just made.
    - Save.
    - 

3. Now create a Flow:
    - Trigger: Solar Panel record updated.
    - Condition: Status = Active.
    - Action: **Send Email Alert** → select Panel Activation Alert.
    - Save & Activate.

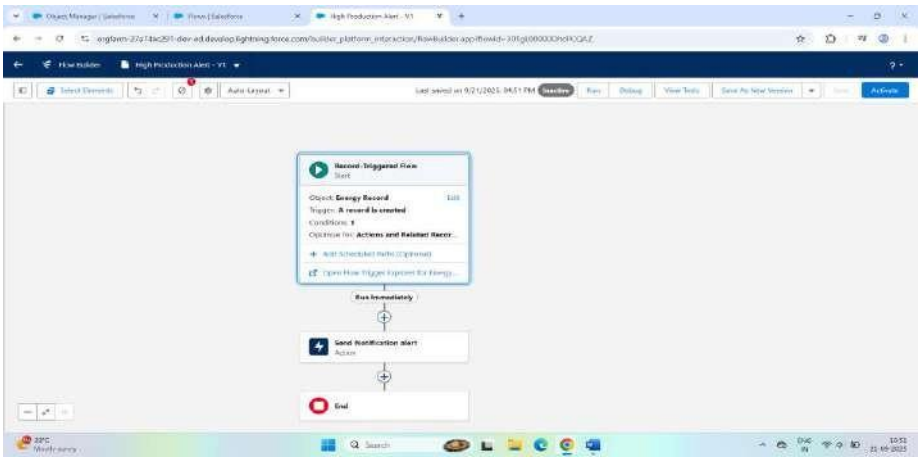
- 

5. Custom Notifications (for Salesforce users)

Example: Notify an internal Agent if a panel produces more than 100 kWh in one record.

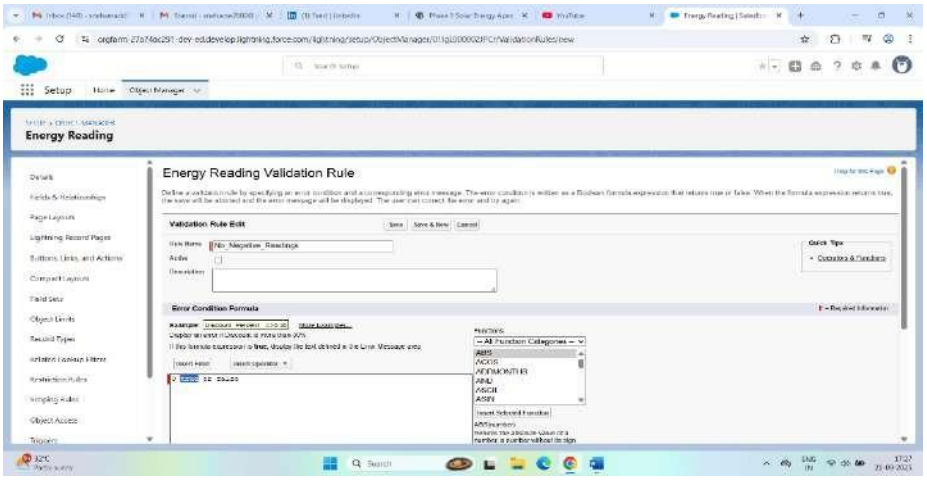
Steps:

1. Setup → **Notifications → Custom Notifications → New**.
    - Label: High Production Alert.
    - Channels: Desktop & Mobile.
    - Save.
2. Setup → Flows → **New Record-Triggered Flow**.
    - Object: **Energy Record**
    - Trigger: **When record is created**.
    - Condition: kWh__c > 100.
3. In Flow, Add Action: **Send Custom Notification**.
    - Notification Type: High Production Alert.
    - Recipient: choose a specific user or the record owner.
    - Message: "High production detected: {!$Record.kWh__c} kWh".
    - Save & Activate.

- 
Record-Triggered Flow
Start
Object: Energy Record
Trigger: A record is created
Conditions: 1
Optimize for: Actions and Related Recor...
Run Immediately
Send Notification alert
Action
End

Phase 5: Apex Programming – Solar Energy Management System

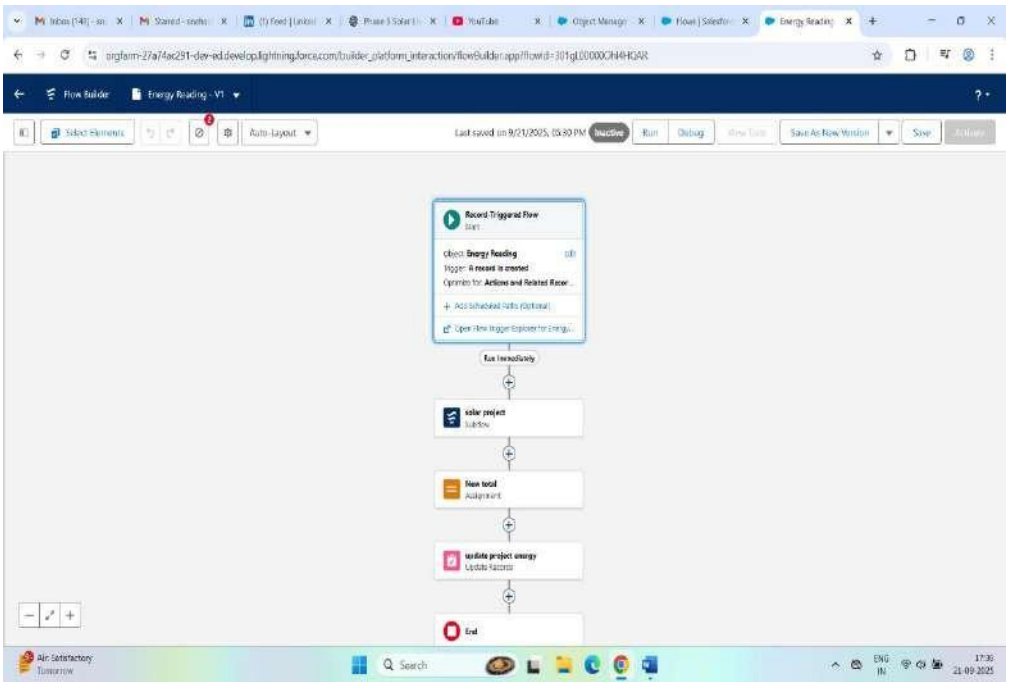
🔹 Step 1: Validation Rule (Prevent Wrong Data)

1. Go to **Setup → Object Manager** → choose **Energy Reading** object.
2. On the left, click **Validation Rules** → then **New**.
3. Rule Name: No_Negative_Readings.
4. Enter Formula:
5. Energy_Produced _ c < 0
6. Error Message: *"Energy produced cannot be negative."*
7. Error Location: Field → Energy Produced.
8. Save and **Activate**.
9. 

🔹 Step 2: Record-Triggered Flow (Auto Update Total Energy)

1. Go to **Setup → Flows → New Flow**.
2. Choose **Record-Triggered Flow** → Click **Create**.
3. Configure Start:
    - Object: **Energy Reading**
    - Trigger: **When a record is created**
    - Run Flow: **After Save** (because we're updating another record).
4. Add Element → **Get Records**:
    - Label: Get Related Project

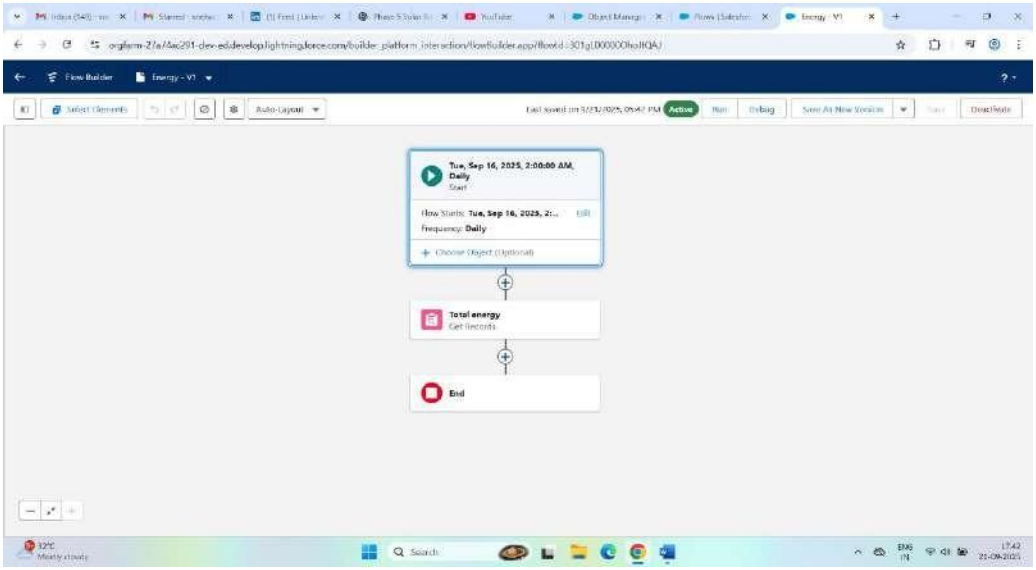
- Object: **Solar Project**
- Condition: Id = {!$Record.Solar_Project__c}
- Store: First Record Only.

5. Add Element → **Assignment**:
   - Label: Increase Total Energy
   - Formula: NewTotal = Project.Total_Energy__c + $Record.Energy_Produced__c.
6. Add Element → **Update Records**:
   - Record: Project from Get Records.
   - Field: Total_Energy__c = NewTotal.
7. Save as: Update Project Energy → **Activate**.
8. 

Step 3: Scheduled Flow (Daily Recalculation for Accuracy)

1. Setup → **Flows** → **New Flow**.
2. Choose **Scheduled-Triggered Flow**.
3. Set Frequency: **Daily** → Run Time: **2:00 AM**.
4. Add Element → **Get Records**:

- Label: Get All Projects
- Object: Solar Project
- Store All Records.

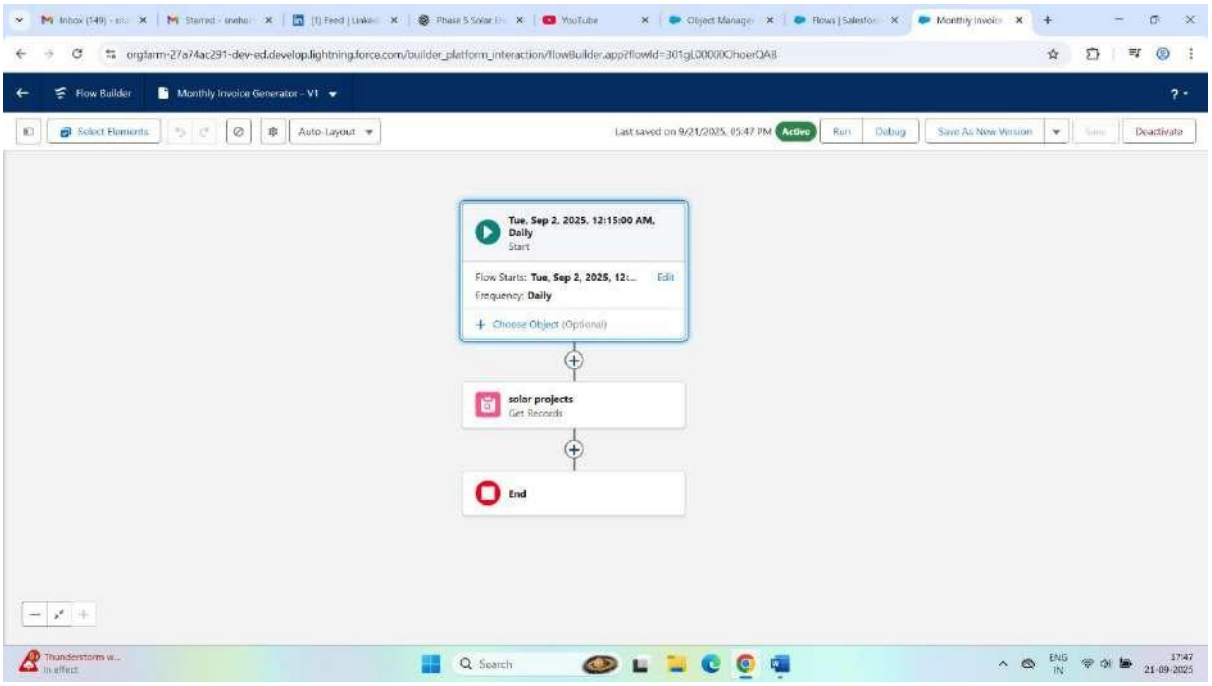
5. Add Element → **Loop** → Loop through Projects.
6. Inside Loop:
   - Add **Get Records**: All Energy Readings for that project.
   - Add **Assignment**: Sum all readings into a variable.
   - Update the project's Total_Energy__c.
7. Save as: Nightly Energy Totals → **Activate**.
8. 

Step 4: Monthly Invoice Automation

1. Setup → **Flows** → **New Flow** → **Scheduled-Triggered Flow**.
2. Frequency: **Monthly**, Day = 1, Time = 12:05 AM.
3. Add Element → **Get Records**:
   - Object: Solar Project
   - Filter: Active = TRUE.
4. Add **Loop** → for each Project:
   - Create **Invoice** record:
     - Solar_Project__c = {!Loop_Project.Id}

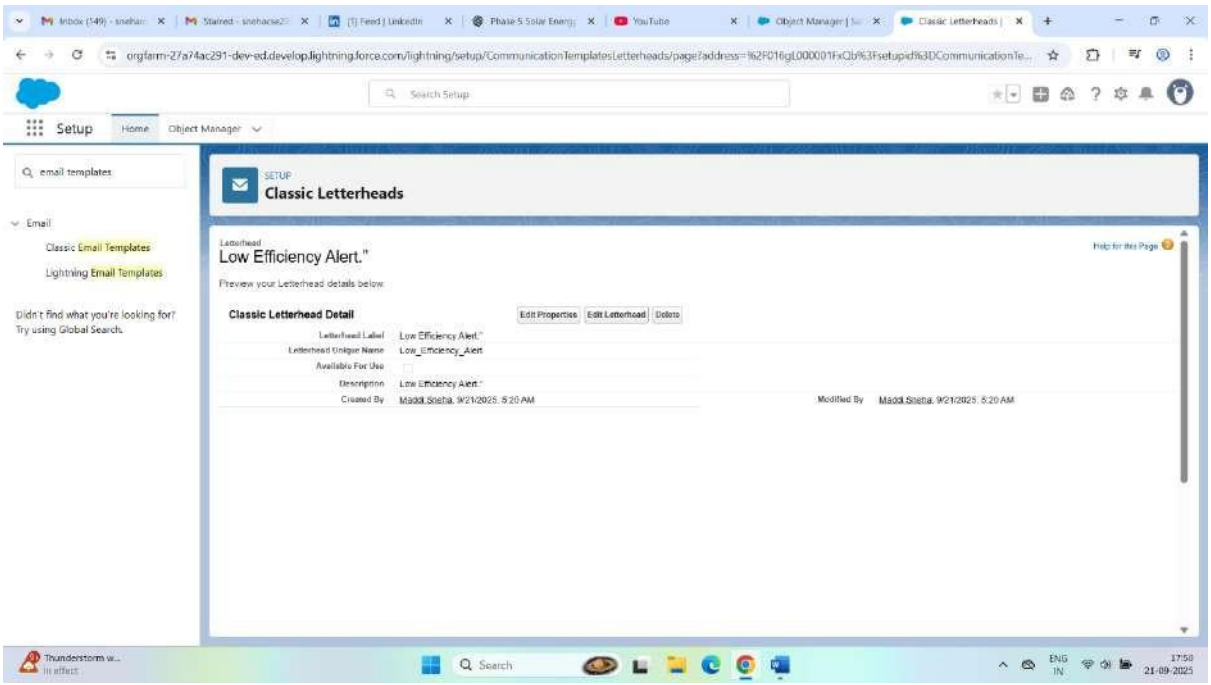
- Period_Start__c = First day of last month
- Period_End__c = Last day of last month
- Amount__c = Loop_Project.Total_Energy__c * Tariff__c (tariff = price/kWh).
- Status__c = Draft.

5. Save as: Monthly Invoice Generator → **Activate**.

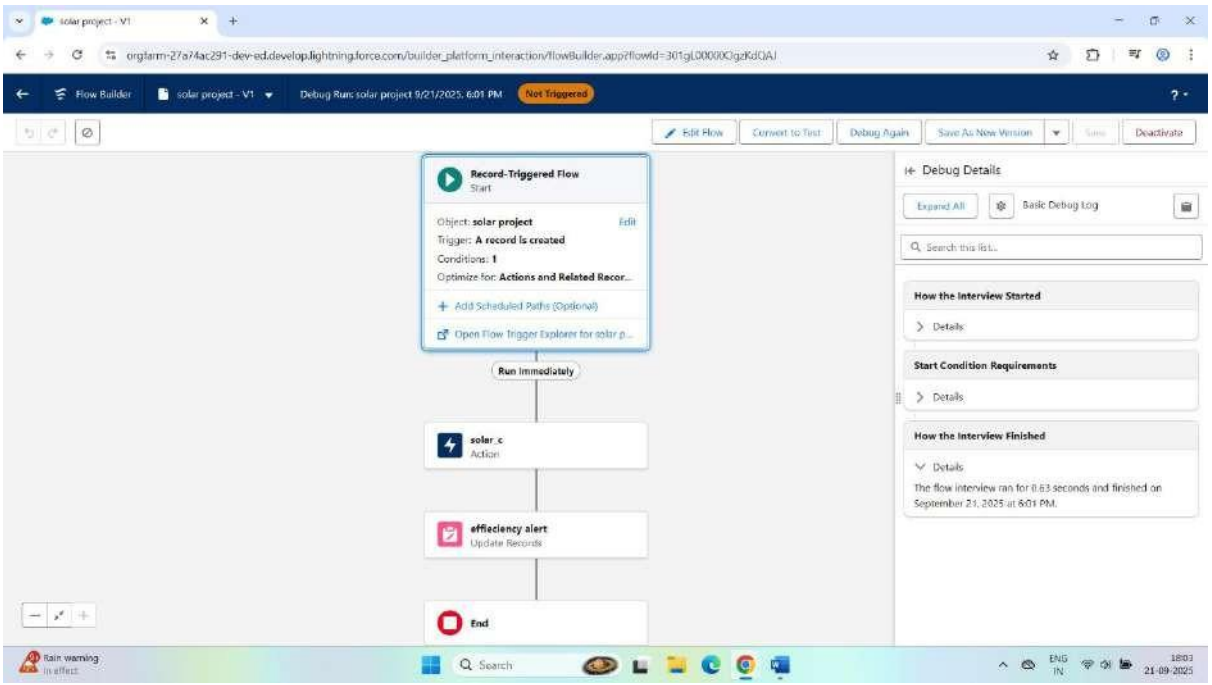


Step 5: Email Alerts for Low Efficiency

1. Setup → **Email Templates** → Create a template: *"Low Efficiency Alert"*.

2. 

3. Setup → **Flows** → **New Flow** → **Record-Triggered Flow**.
4. Object: **Solar Project**.
5. Trigger: When record is **created or updated**.
6. Condition: Efficiency__c < 60 AND Maintenance_Flag__c = FALSE.
7. Add Action → **Send Email**: To Maintenance Manager.
8. Add Action → **Update Records**: Set Maintenance_Flag__c = TRUE.
9. Save as: Low Efficiency Alert → **Activate**.



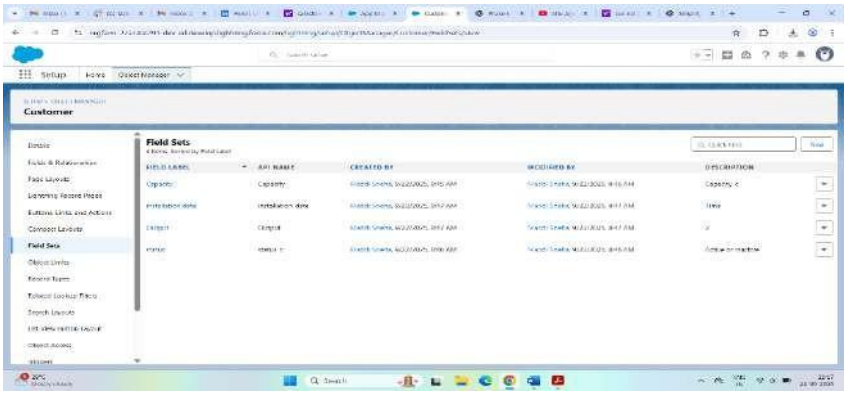
Step 6: Reports & Dashboards

1. App Launcher → Reports → **New Report**.
    - Type: **Solar Project with Energy Readings**.
    - Add Columns: Project Name, Total Energy, Efficiency, Status.
    - Save: Solar Project Performance.
2. App Launcher → Dashboards → **New Dashboard**.
3. Add Components:
    - **Bar Chart**: Top 5 projects by total energy.
    - **Gauge Chart**: Average efficiency across all projects.
    - **Table**: Projects below 60% efficiency.
    - **Line Chart**: Monthly revenue trend (from invoices).

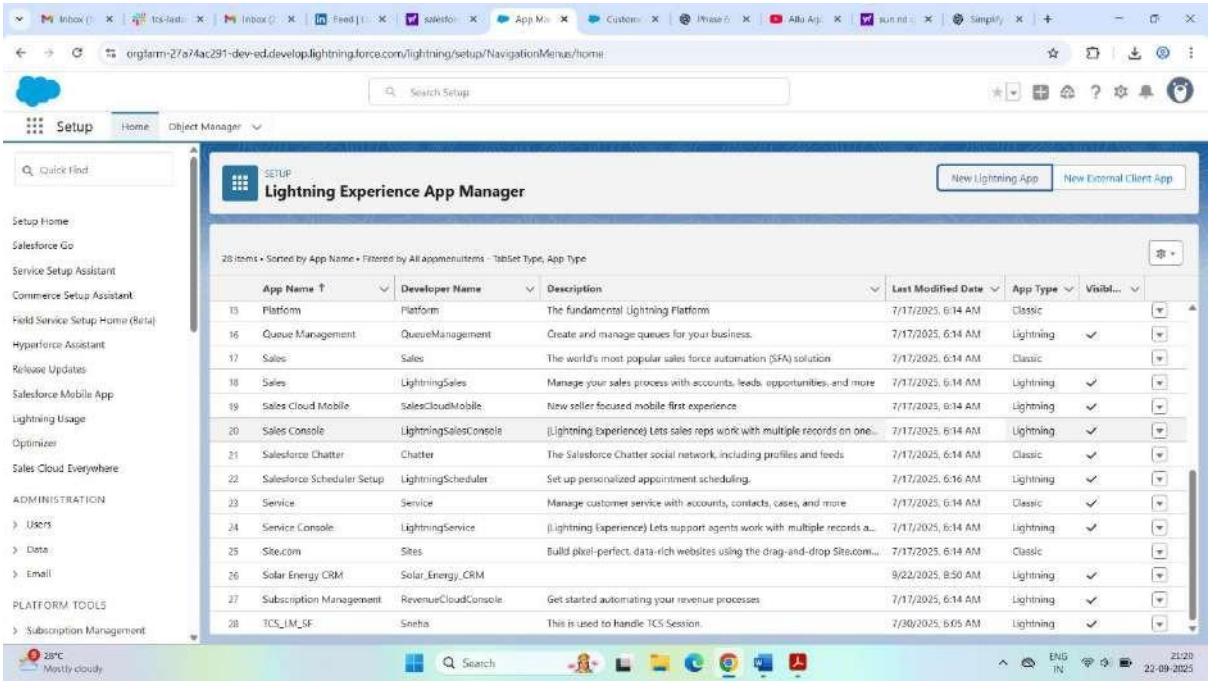
Phase 6: User Interface Development (Elaborate Version)

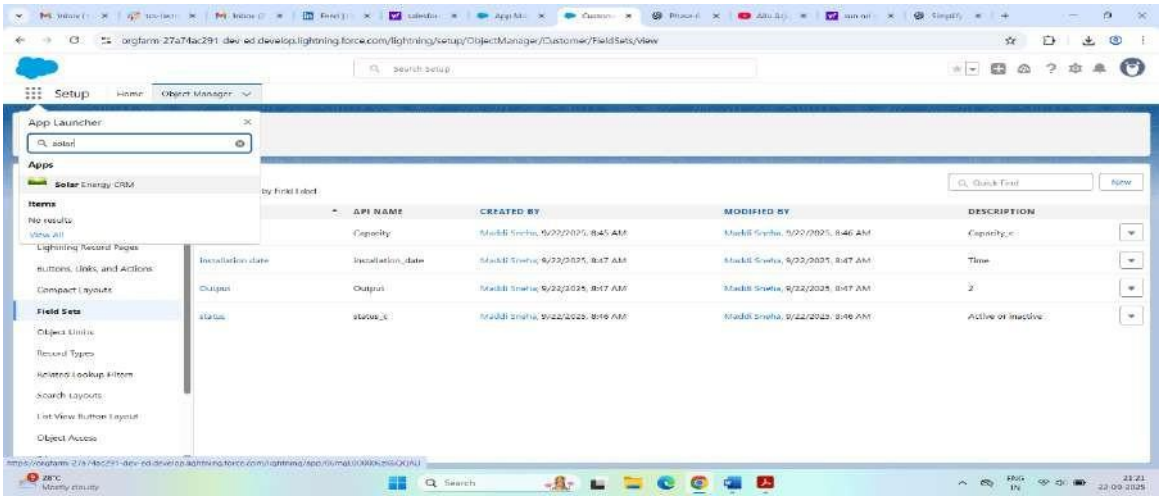
= **Goal:** Make the system easy for Solar Agents, Managers, and Customers to use by designing clear pages, dashboards, and components.

🔹 Step 1: Create the Solar Energy CRM App

1. Go to **Setup → App Manager → New Lightning App**.
2. Enter:
    - App Name: Solar Energy CRM
    - Description: Manage Solar Projects, Panels, Customers, and Maintenance.
    - 

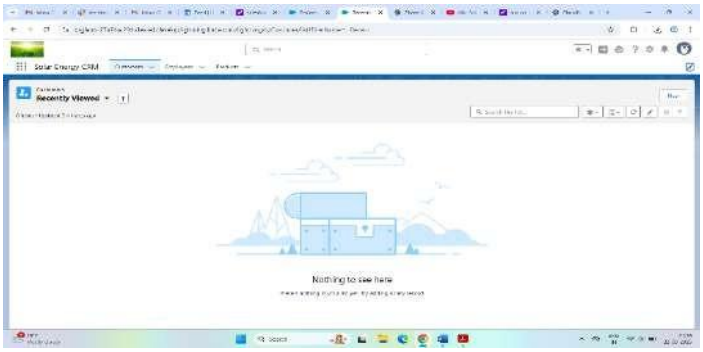
3. Choose **Standard Navigation (Lightning Experience)**.
4. Add utility bar if needed (quick shortcuts).
5. Select **Navigation Items** → add:
    - Solar Projects
    - Solar Panels
    - Maintenance
    - Customers
    - Reports
    - Dashboards
6. Save → App appears in **App Launcher**.





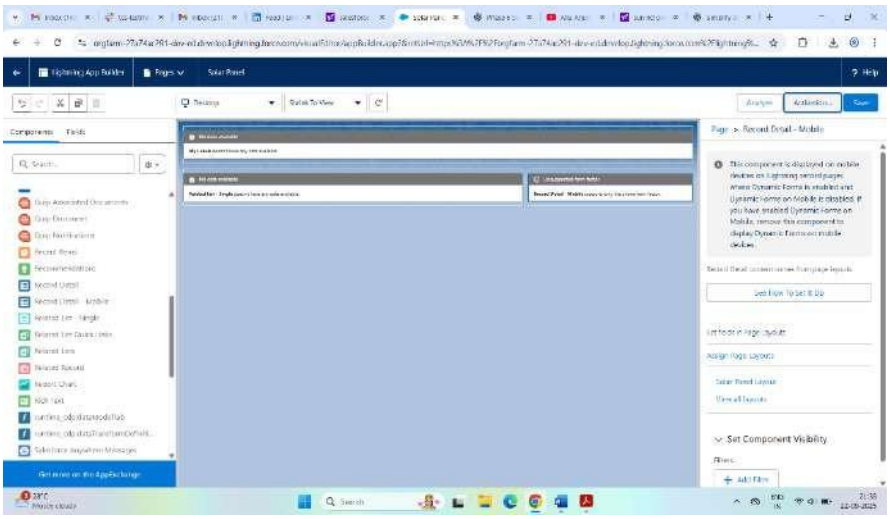
◆ Step 2: Add Tabs for Quick Navigation

- Ensure each custom object has a **Tab**.
- Setup → **Tabs** → **New Tab** → pick object → assign to Solar CRM app.
- Result: Agents and Managers can easily switch between Projects, Panels, Customers, and Maintenance.



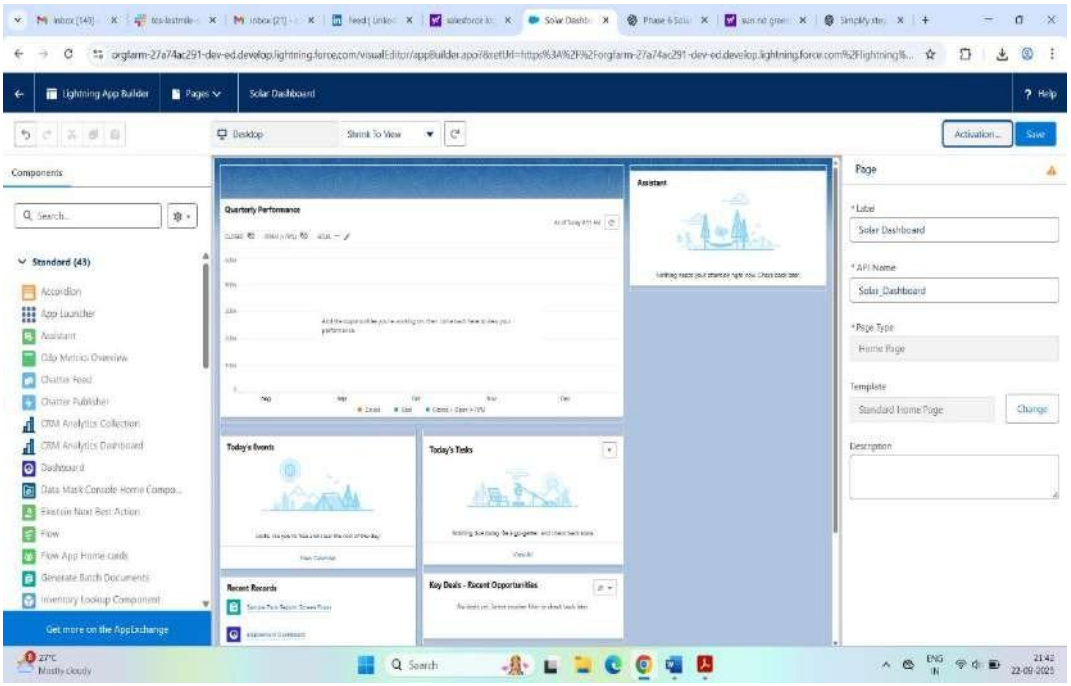
🔷 Step 3: Design Record Pages (Lightning App Builder)

1. Go to **Setup → Lightning App Builder → New → Record Page**.
2. Example 1: **Solar Project Record Page**
    - Components:
        - **Highlights Panel** → Project Name, Customer, Status, Capacity (kW).
        - **Related Lists** → Solar Panels, Maintenance Visits, Invoices.
        - **Report Chart** → Total Energy Output for this project.
    - Save → Activate → Assign to **Solar CRM App Default**.
3. Example 2: **Solar Panel Record Page**
    - Components:
        - **Record Detail** → Show Status, Capacity, Output, Installation Date.
        - **Related Lists** → Maintenance Logs.
        - **Custom LWC** → Energy Output (daily/weekly).
    - Save → Activate.



◆ Step 4: Build the Home Page Dashboard

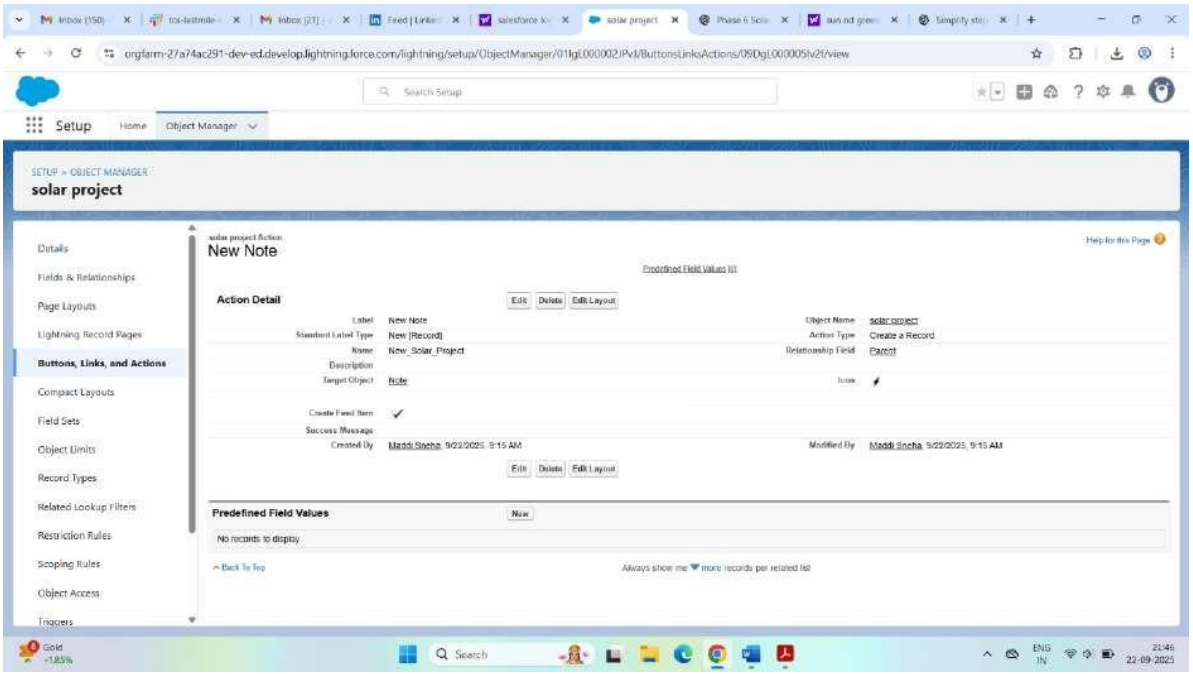
1. Setup → **Lightning App Builder → New → App Page → Home Page**.
2. Add:
    - **Dashboard component** → Solar Dashboard (Active Projects, Total Capacity, Faulty Panels).
    - **List View component** → "Panels Needing Maintenance".
    - **Rich Text** → Welcome message for users.
3. Save → Assign to **Solar CRM App**.

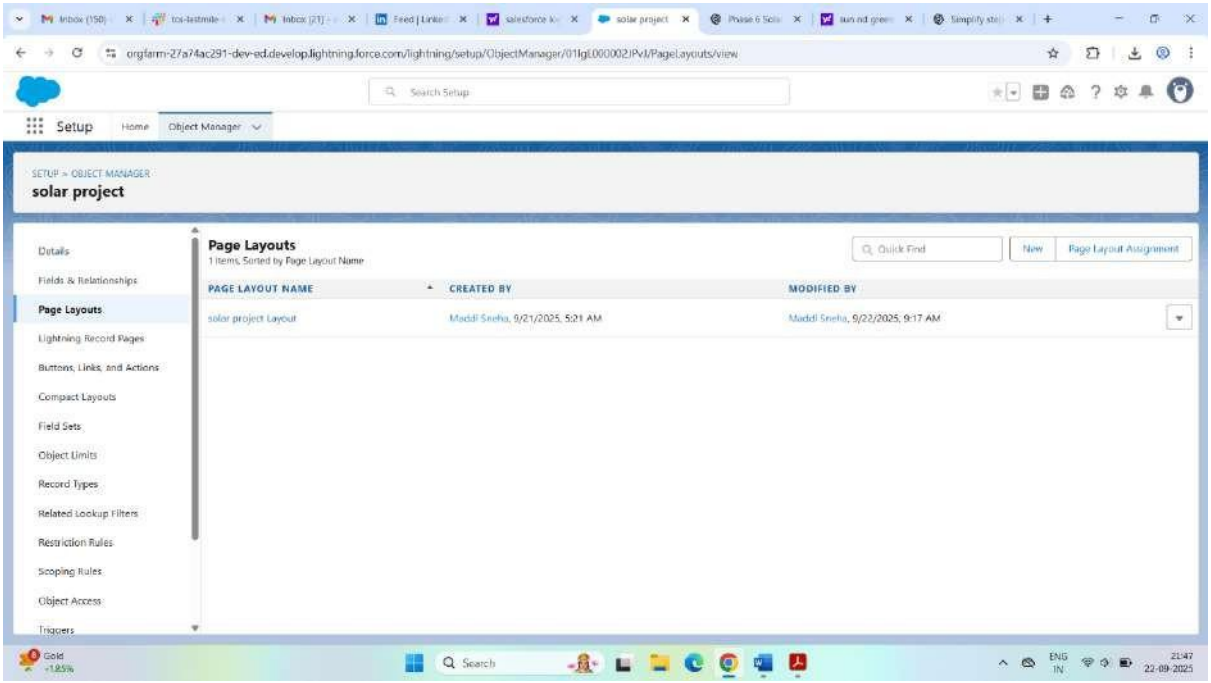


◆

Step 5: Add Quick Actions (for fast data entry)

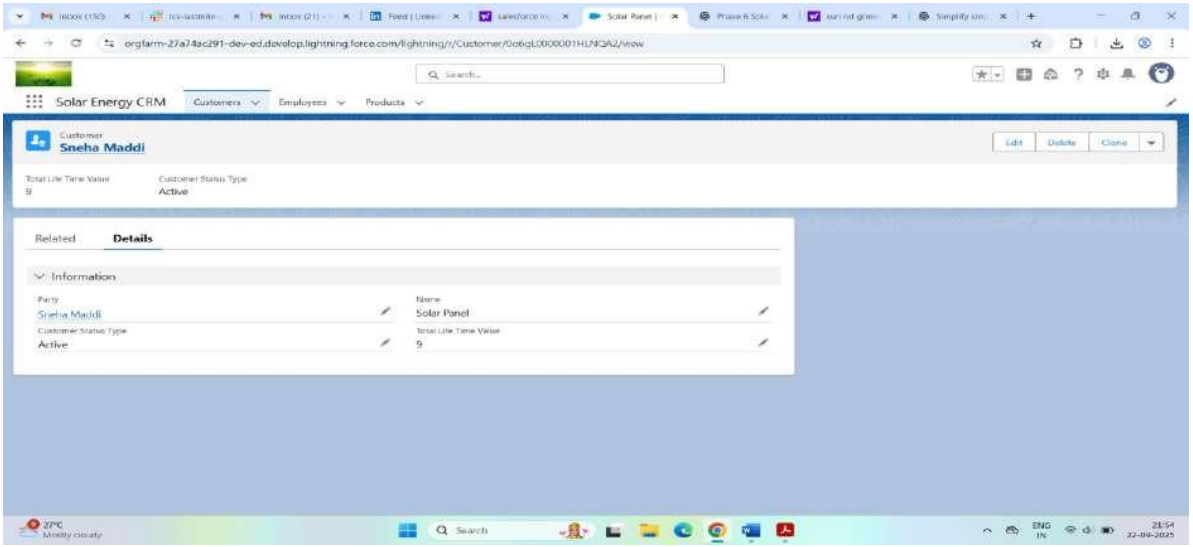
1. Setup → **Object Manager → Solar Project → Buttons, Links, and Actions → New Action**.
    - Action Type = Create a Record
    - Label = New Solar Project
    - Add fields: Project Name, Customer, Start Date, Capacity (kW).
    - Save.
2. Edit the **Page Layout** → Add the action to the **Salesforce Mobile & Lightning Actions** section.
3. Repeat for **Maintenance Log**: create a quick action “Log Maintenance”.





🔹 Step 6: Create List Views for Panels & Projects

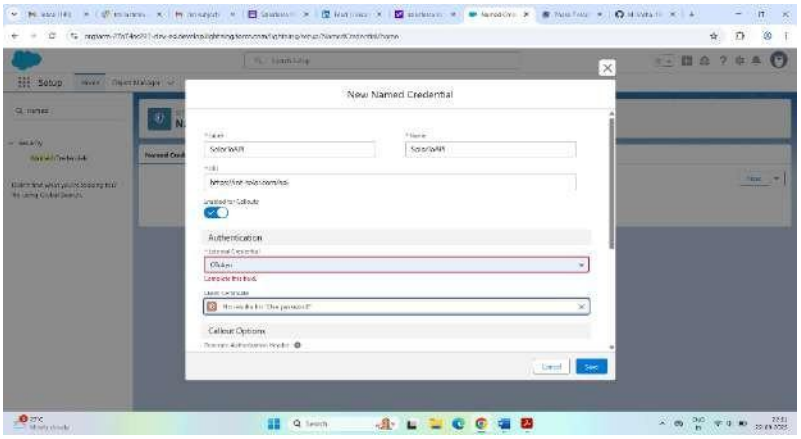
- Open **Solar Panels Tab → List View → New**.
- Create:
  - Active Panels → Filter: Status = Active.
  - Faulty Panels → Filter: Status = Faulty.
  - Needs Maintenance → Filter: Last Maintenance Date > 30 days ago.



Phase 7: Integration & External Access – Solar Energy Management System

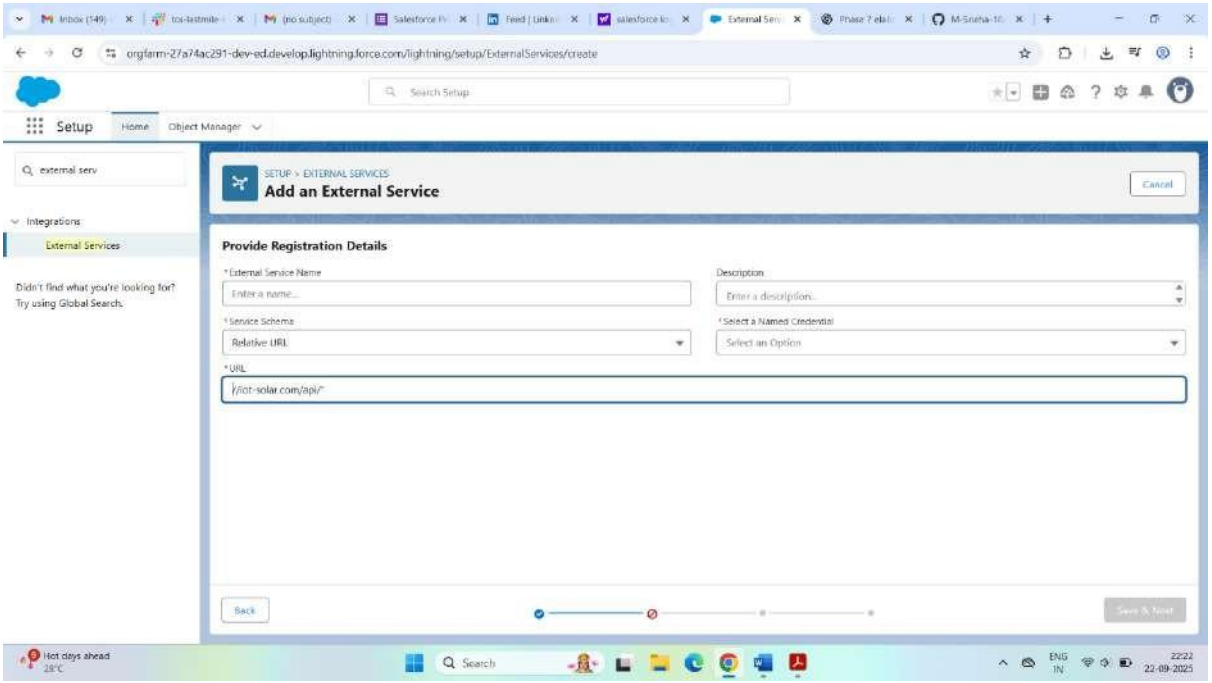
Step 1: Named Credentials (Store API login securely)

1. Go to Setup → Named Credentials → New.
2. Fill in details:
    - Label: SolarIoTAPI
    - URL: https://iot-solar.com/api
    - Auth: OAuth / Token / Username-Password.
3. Save → Use in Apex/Flows with callout:SolarIoTAPI.



◆ Step 2: External Services (No-code API integration)

1. Go to Setup → External Services.
2. Click New External Service → upload API schema (Swagger file).
3. Salesforce auto-generates actions (like “Get Panel Output”).
4. Use these in Flows → e.g., when “Check Output” button is clicked, it fetches data from IoT.



◆ Step 3: Web Services (REST/SOAP)

- REST → Get energy production data from IoT panels.
- SOAP → If billing system provides WSDL service.

REST Callout Example (Apex):

```
HttpRequest req = new HttpRequest();
req.setEndpoint('callout:SolarIoTAPI/panelData?panelId=123');
req.setMethod('GET');
Http http = new Http();
HttpResponse res = http.send(req);
System.debug(res.getBody());
```

◆ Step 4: Callouts (Triggered APIs)

Why? Automate sending data out when events happen.
Examples:

- When a Solar Project is created → send customer/project info to billing system.

- When Maintenance Request is logged → call IoT system to pause/shut down panel safely.

Easy Way:

- Create an Apex Trigger → inside call a Queueable class (since triggers require async callouts).

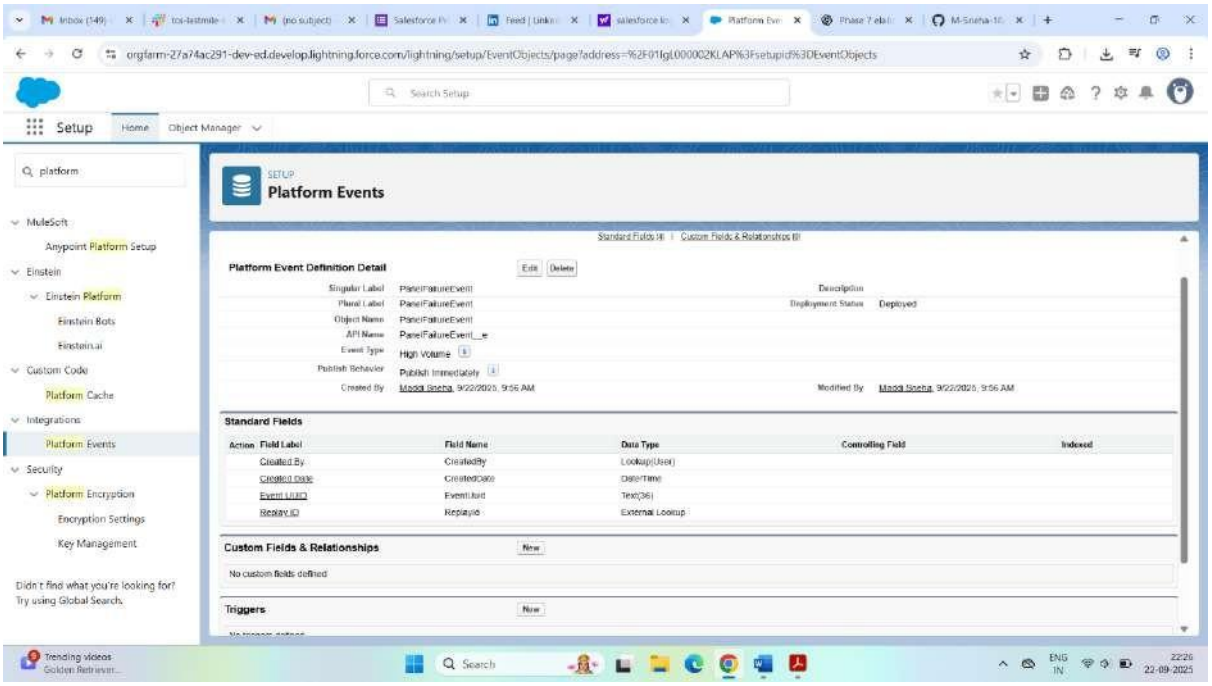
🔹 Step 5: Platform Events (Real-time Alerts)

Examples:

- If Solar Panel Breakdown reported in Salesforce → publish PanelFailureEvent.
- IoT Monitoring System subscribes → alerts technician.

How?

1. Go to Setup → Platform Events → New.
2. Create PanelFailureEvent with fields like PanelId, FailureType.
3. Use Flow or Apex to publish when a failure is logged.



🔹 Step 6: Change Data Capture (CDC)

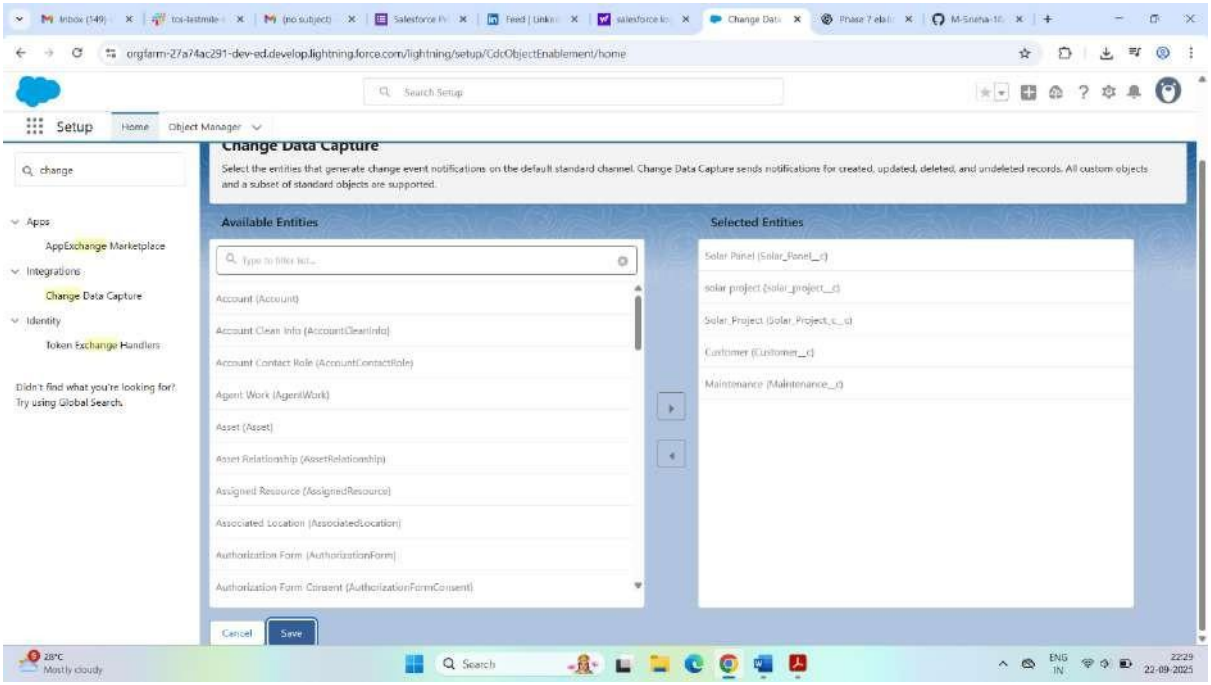
Examples:

- If Panel Status changes → IoT app is notified.

- If Customer Contact Info is updated → billing app is updated.

How?

1. Go to Setup → Change Data Capture.
2. Enable objects: Solar Panel, Solar Project, Customer.
3. External app subscribes via CometD API or Pub/Sub API.

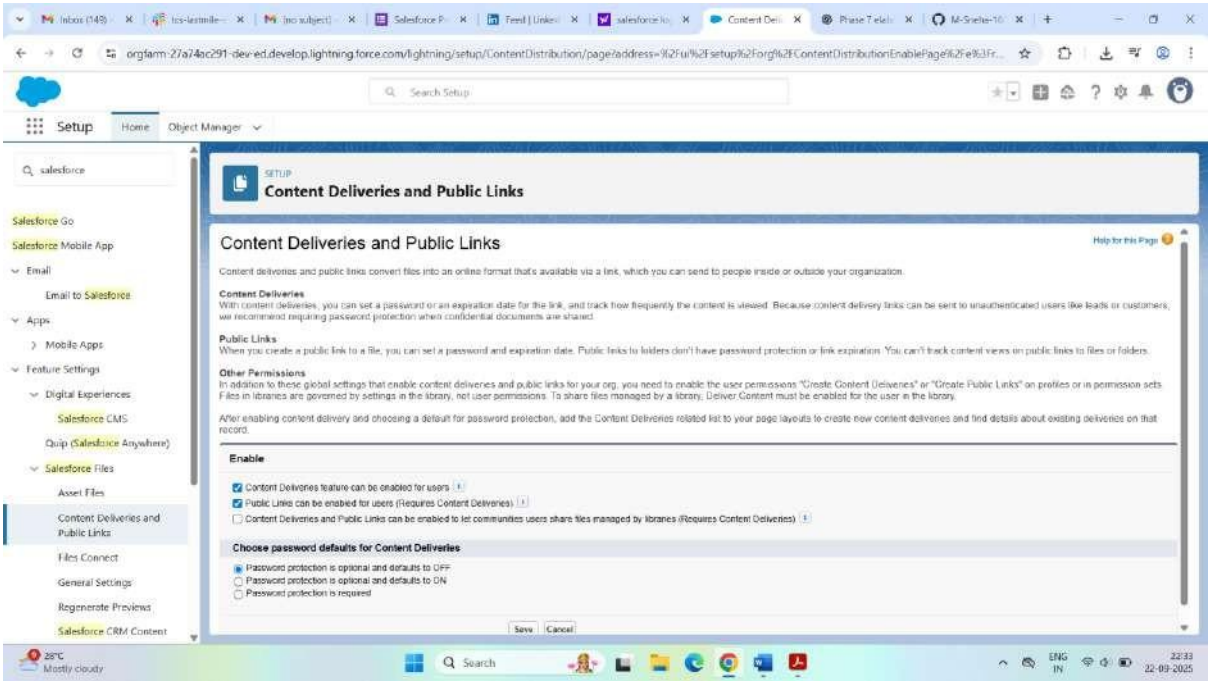


🔷 Step 7: Salesforce Connect (View external data in Salesforce)

Example: Show Solar Panel Logs stored in external SQL DB as if they're inside Salesforce.

How?

1. Setup → Salesforce Connect.
2. Create External Data Source → connect to database.
3. Panels appear as External Objects → viewable like normal records.



◆ Step 8: API Limits

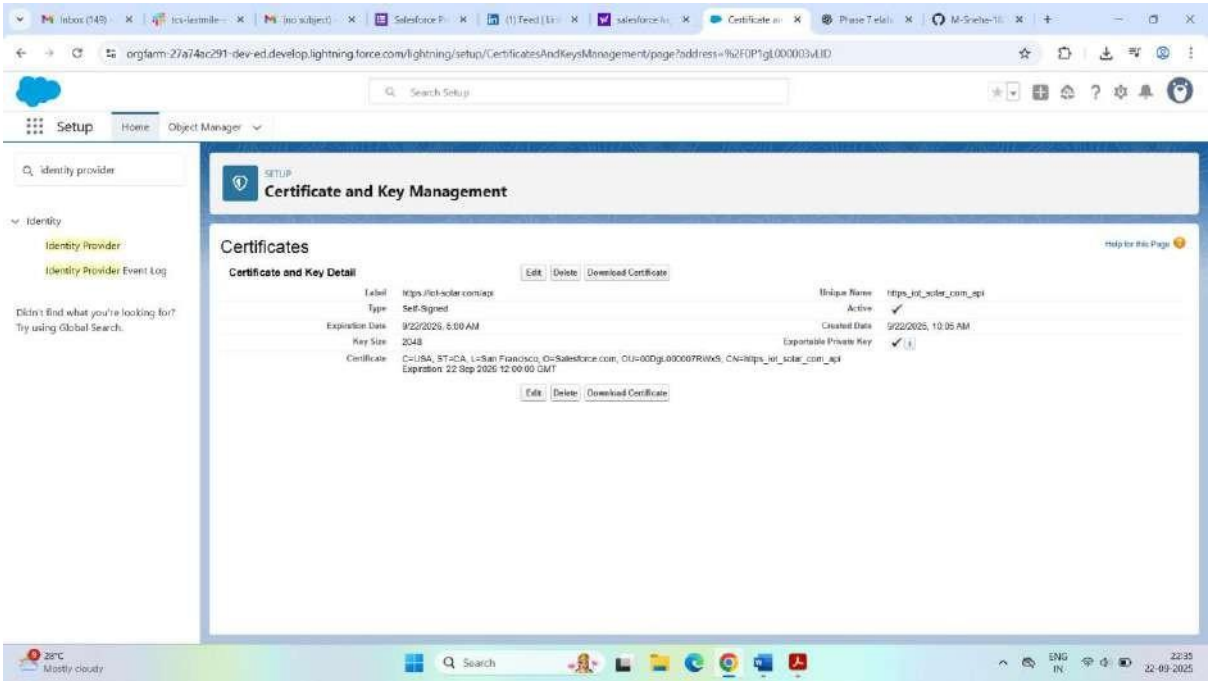
Why? Salesforce has daily API call limits.
Tips:

- If IoT sends 10,000 panel updates → use Bulk API or Streaming API instead of individual calls.
- Monitor in Setup → System Overview → API Usage.

◆ Step 9: OAuth & Authentication

Example: Customers log in to a Solar Portal to view energy usage.
How?

1. Setup → Identity Provider.
2. Enable OAuth providers (Google, Microsoft, etc.).
3. Link customer portal login with OAuth.

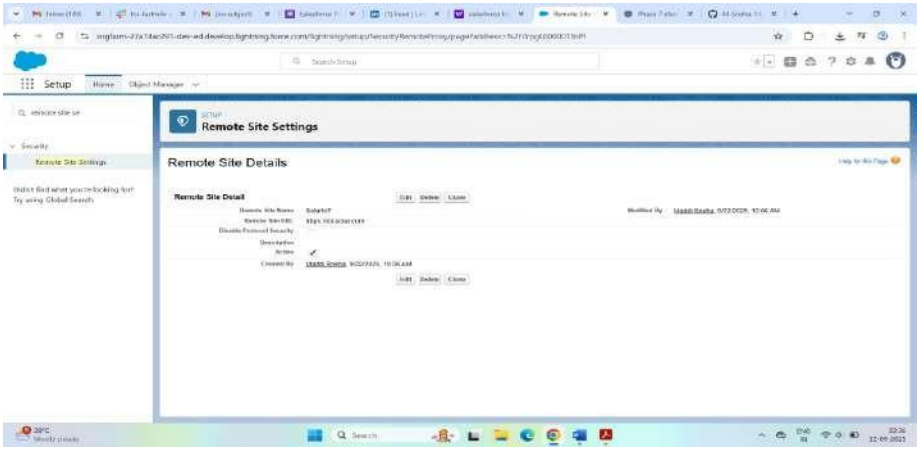


◆ Step 10: Remote Site Settings

Example: Before calling https://iot-solar.com, add it.

How?

1. Go to Setup → Remote Site Settings.
2. Click New Remote Site.
3. Add:
    - Name: SolarIoT
    - URL: https://iot-solar.com
4. Save → Now callouts will work.



Phase 8: Data Management & Deployment

- ✓ I have managed data and deployment as follows:
- ✓ Change Sets / VS Code / SFDX: Since I developed directly in a free Developer Edition org, I deployed LWCs using Salesforce Extension. No sandbox-to-production deployment was required.

```
template>
    <lightning-card title="Find & Book Solar Panels">
        <lightning-input type="date" label="Start Date" value={startD
        <lightning-input type="date" label="End Date" value={endDate}
        <lightning-button label="Search Panels" variant="brand" oncli

        <template if:true={panels}>
            <lightning-datatable key-field="Id" data={panels} columns
                onrowaction={handleRowAction}></lightning-datatable>
        </template>
    </lightning-card>
/template>
```

- ✓ Unmanaged Packages: All objects, LWCs, and triggers are custom unmanaged components created directly in the org and also executed the code in VS.

```
ort { LightningElement, track } from 'lwc';
ort getAvailablePanels from '@salesforce/apex/BookingService.getAvailablePanels';
ort createBooking from '@salesforce/apex/BookingService.createBooking';
ort { ShowToastEvent } from 'lightning/platformShowToastEvent';

ort default class PanelSearchBook extends LightningElement {
 @track startDate;
 @track endDate;
 @track panels;

 columns = [
    { label: 'Panel', fieldName: 'Name' },
    { label: 'Serial', fieldName: 'Serial_Number__c' },
    { label: 'Capacity (kW)', fieldName: 'Capacity_kW__c', type: 'number' },
    {
        type: 'action',
        typeAttributes: { rowActions: [{ label: 'Book', name: 'book' }] }
    }
 ];

 handleStart(e) { this.startDate = e.target.value; }
 handleEnd(e) { this.endDate = e.target.value; }

 searchPanels() {
```

Phase 9: Reporting, Dashboards & Security Review

Reporting and security settings were handled carefully to ensure proper access:

● Dynamic Dashboards: Not implemented since access is controlled via user sub-profiles.

● Sharing Settings & Field-Level Security: Managed using sub-profile logic. Patients see only their records, while doctors and attendants see all relevant patient records.

● Session Setting & Login IP Ranges: Default org security is sufficient.

COMPLETE CODES FOR SOLAR ENERGY MANAGEMENT SYSTEM LIKE APEX PROGRAMMING AND TRIGGERING:

APEX CLASS: BookingService

```
public with sharing class BookingService {
    public class BookingResult {
        @AuraEnabled public Boolean success;
        @AuraEnabled public String message;
        @AuraEnabled public Id bookingId;
        public BookingResult(Boolean s, String m, Id b) {
            success = s; message = m; bookingId = b;
        }
    }

    // Create booking with overlap validation
    @AuraEnabled
    public static BookingResult createBooking(Solar_Booking__c booking) {
        if (booking == null) return new BookingResult(false, 'No booking
provided', null);
        if (booking.Start_Date__c == null || booking.End_Date__c == null)
```

```
            return new BookingResult(false, 'Start and End date are required', null);
        if (booking.End_Date__c < booking.Start_Date__c)
            return new BookingResult(false, 'End Date must be after Start Date',
null);

        List<Solar_Booking__c> overlap = [
            SELECT Id FROM Solar_Booking__c
            WHERE Solar_Panel__c = :booking.Solar_Panel__c
            AND Status__c != 'Cancelled'
            AND ( :booking.Start_Date__c <= End_Date__c
              AND :booking.End_Date__c >= Start_Date__c )
            LIMIT 1
        ];
        if (!overlap.isEmpty()) {
            return new BookingResult(false, 'Panel already booked in this date
range', null);
        }

        if (String.isBlank(booking.Status__c)) booking.Status__c = 'Requested';
        insert booking;
        return new BookingResult(true, 'Booking created successfully',
booking.Id);
    }

    // Get available panels
    @AuraEnabled(cacheable=true)
```

```
    public static List<Solar_Panel__c> getAvailablePanels(Date startDate, Date
endDate) {
        if (startDate == null || endDate == null) return new
List<Solar_Panel__c>();
 List<Solar_Panel__c> panels = [
            SELECT Id, Name, Serial_Number__c, Capacity_kW__c
            FROM Solar_Panel__c
            WHERE Status__c = 'Active'
        ];

        Set<Id> busy = new Set<Id>();
        for (Solar_Booking__c b : [
            SELECT Solar_Panel__c FROM Solar_Booking__c
            WHERE Solar_Panel__c IN :panels
            AND Status__c != 'Cancelled'
            AND (:startDate <= End_Date__c AND :endDate >= Start_Date__c)
        ]) {
            busy.add(b.Solar_Panel__c);
        }

        return [SELECT Id, Name, Serial_Number__c, Capacity_kW__c
                FROM Solar_Panel__c
                WHERE Status__c = 'Active'
                AND Id NOT IN :busy];
    }
}
```

APEX CLASS: SolarBookingTriggerHandler

```
public with sharing class SolarBookingTriggerHandler {
    public static void validateNoOverlap(List<Solar_Booking__c> newBookings) {
        for (Solar_Booking__c nb : newBookings) {
            if (nb.Solar_Panel__c != null && nb.Start_Date__c != null && nb.End_Date__c != null) {
                List<Solar_Booking__c> conflicts = [
                    SELECT Id FROM Solar_Booking__c
                    WHERE Solar_Panel__c = :nb.Solar_Panel__c
                    AND Status__c != 'Cancelled'
                    AND Id != :nb.Id
                    AND (:nb.Start_Date__c <= End_Date__c AND :nb.End_Date__c >= Start_Date__c)
                    LIMIT 1
                ];
                if (!conflicts.isEmpty()) {
                    nb.addError('This panel is already booked during these dates.');
                }
            }
        }
    }
}
```

APEX CLASS: SolarIoTCallout

```
public with sharing class SolarIoTCallout {
    public static void registerBookingToIoT(Id bookingId) {
```

```
        Solar_Booking__c b = [SELECT Id, Solar_Panel__c, Start_Date__c,
End_Date__c FROM Solar_Booking__c WHERE Id = :bookingId LIMIT 1];
        Solar_Panel__c p = [SELECT Serial_Number__c FROM Solar_Panel__c
WHERE Id = :b.Solar_Panel__c LIMIT 1];


        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:SolarIoTAPI/bookings');
        req.setMethod('POST');
        req.setHeader('Content-Type','application/json');
  Map<String, Object> payload = new Map<String, Object>{
            'bookingId' => String.valueOf(b.Id),
            'panelSerial' => p.Serial_Number__c,
            'startDate' => b.Start_Date__c.format(),
            'endDate' => b.End_Date__c.format()
        };
        req.setBody(JSON.serialize(payload));


        Http http = new Http();
        HttpResponse res = http.send(req);
        System.debug('IoT Response: ' + res.getBody());
    }
}
```

APEX CLASS: IoTBookingRegisterQueueable

```
public with sharing class IoTBookingRegisterQueueable implements
Queueable, Database.AllowsCallouts {
    private Id bookingId;
```

```
    public IoTBookingRegisterQueueable(Id bookingId) { this.bookingId =
bookingId; }
    public void execute(System.QueueableContext ctx) {
        SolarIoTCallout.registerBookingToIoT(bookingId);
    }
}
```

APEX CLASS: PanelFailurePublisher

```
public with sharing class PanelFailurePublisher {
    public static void publishFailure(Id panelId, String type, String reportedBy) {
        PanelFailureEvent__e evt = new PanelFailureEvent__e(
            PanelId__c = String.valueOf(panelId),
            FailureType__c = type,
            ReportedBy__c = reportedBy
        );
        EventBus.publish(evt);
    }
}
```

APEX TEST CLASS: BookingServiceTest

```
@IsTest
private class BookingServiceTest {
    @IsTest static void testBookingOverlap() {
        Solar_Panel__c p = new Solar_Panel__c(Name='Test Panel',
Serial_Number__c='SN001', Status__c='Active', Capacity_kW__c=5);
        insert p;
```

```
        Solar_Booking__c existing = new
Solar_Booking__c(Solar_Panel__c=p.Id, Start_Date__c=Date.today(),
End_Date__c=Date.today().addDays(2), Status__c='Confirmed');
        insert existing;


        Solar_Booking__c conflict = new
Solar_Booking__c(Solar_Panel__c=p.Id,
Start_Date__c=Date.today().addDays(1),
End_Date__c=Date.today().addDays(3));
        Test.startTest();
        BookingService.BookingResult result =
BookingService.createBooking(conflict);
        Test.stopTest();
        System.assertEquals(false, result.success);
    }
}
```

APEX TRIGGER: SolarBookingTrigger

```
trigger SolarBookingTrigger on Solar_Booking__c (before insert, before
update) {
    SolarBookingTriggerHandler.validateNoOverlap(Trigger.new);
}
```

BELOW CODES ARE BASED ON LWC COMPONENTS:

HTML CODE:

```
<!-- panelSearchBook.html -->
<template>
    <lightning-card title="Find & Book Solar Panels">
        <lightning-input type="date" label="Start Date" value={startDate}
onchange={handleStart}></lightning-input>
```

```
        <lightning-input type="date" label="End Date" value={endDate}
onchange={handleEnd}></lightning-input>
        <lightning-button label="Search Panels" variant="brand"
onclick={searchPanels}></lightning-button>

        <template if:true={panels}>
            <lightning-datatable key-field="Id" data={panels} columns={columns}
                onrowaction={handleRowAction}></lightning-datatable>
        </template>
    </lightning-card>
</template>
```

JAVASCRIPT CODE:

```
// panelSearchBook.js
import { LightningElement, track } from 'lwc';
import getAvailablePanels from
'@salesforce/apex/BookingService.getAvailablePanels';
import createBooking from '@salesforce/apex/BookingService.createBooking';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class PanelSearchBook extends LightningElement {
    @track startDate;
    @track endDate;
    @track panels;

    columns = [
        { label: 'Panel', fieldName: 'Name' },
        { label: 'Serial', fieldName: 'Serial_Number__c' },
```

```
    { label: 'Capacity (kW)', fieldName: 'Capacity_kW__c', type: 'number' },
    {
        type: 'action',
        typeAttributes: { rowActions: [{ label: 'Book', name: 'book' }] }
    }
];

handleStart(e) { this.startDate = e.target.value; }
handleEnd(e) { this.endDate = e.target.value; }

searchPanels() {
    getAvailablePanels({ startDate: this.startDate, endDate: this.endDate })
        .then(res => { this.panels = res; })
        .catch(err => { this.showToast('Error', err.body.message, 'error'); });
}

handleRowAction(event) {
    const row = event.detail.row;
    const booking = {
        'sobjectType': 'Solar_Booking__c',
        'Solar_Panel__c': row.Id,
        'Start_Date__c': this.startDate,
        'End_Date__c': this.endDate,
        'Status__c': 'Requested'
    };
    createBooking({ booking: booking })
```

```
            .then(res => {
                if (res.success) this.showToast('Success', 'Booking created', 'success');
                else this.showToast('Failed', res.message, 'error');
            })
            .catch(err => { this.showToast('Error', err.body.message, 'error'); });
    }

    showToast(title, message, variant) {
        this.dispatchEvent(new ShowToastEvent({ title, message, variant }));
    }
}
```

XML CODE:

```
<!-- panelSearchBook.js-meta.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle
xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```