## PropertyCare CRM Project

**Industry**: Real Estate

**Project Type**: B2C Salesforce CRM Implementation

**Target Users**: Property Owners, Service Agents, Property Managers

**Problem Statement:**

After purchasing a property, buyers often face difficulties managing warranty services, maintenance requests, and annual maintenance contracts (AMCs). Service teams struggle to track repair tickets, follow-ups, and expiring contracts, leading to poor customer satisfaction and missed SLAs.

To address this, the company wants to implement a Salesforce CRM to:

* Automate warranty and AMC tracking
* Manage maintenance requests and case assignments
* Track service agent activities and SLA compliance
* Provide real-time dashboards for management

**Solution Overview:**

PropertyCare CRM is a Salesforce-based system designed to streamline post-sale property management, improve resident satisfaction, and increase service team efficiency.

**Key features:**

* Warranty & AMC Management
* Maintenance Case Management
* Service Agent Management
* Reporting & Dashboards
* **Expected Outcomes:**
* Residents get timely service and reminders for warranties/AMCs
* Service agents efficiently manage and resolve cases
* Management gains real-time insights through dashboards
* Improved customer satisfaction and operational efficiency
* **Placeholders:**
* ERD Diagram
* Salesforce Objects, Flows, and Dashboard Screenshots

**Phase 1: Problem Understanding & Industry Analysis**

**Goal:** The PropertyCare CRM aims to **automate warranty and AMC tracking**, manage maintenance requests efficiently, and assign cases to service agents. It provides **real-time dashboards and reports** for management and ensures **timely notifications** to residents, improving satisfaction and operational efficiency.

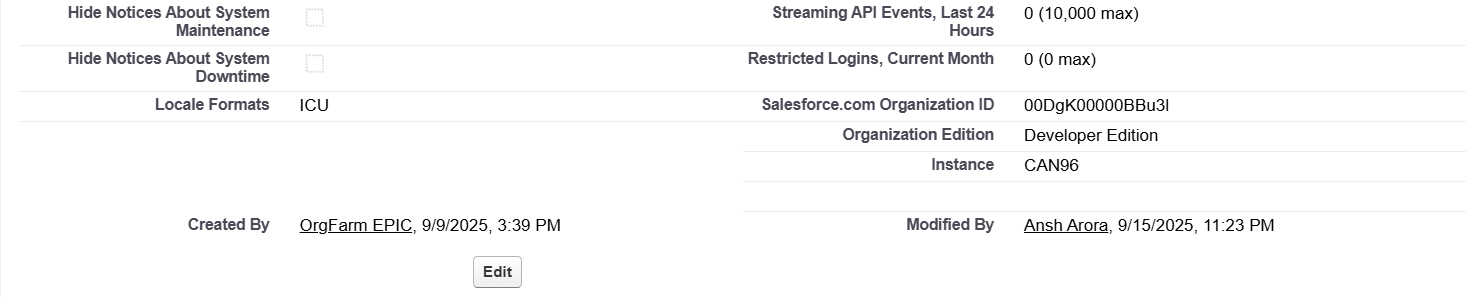
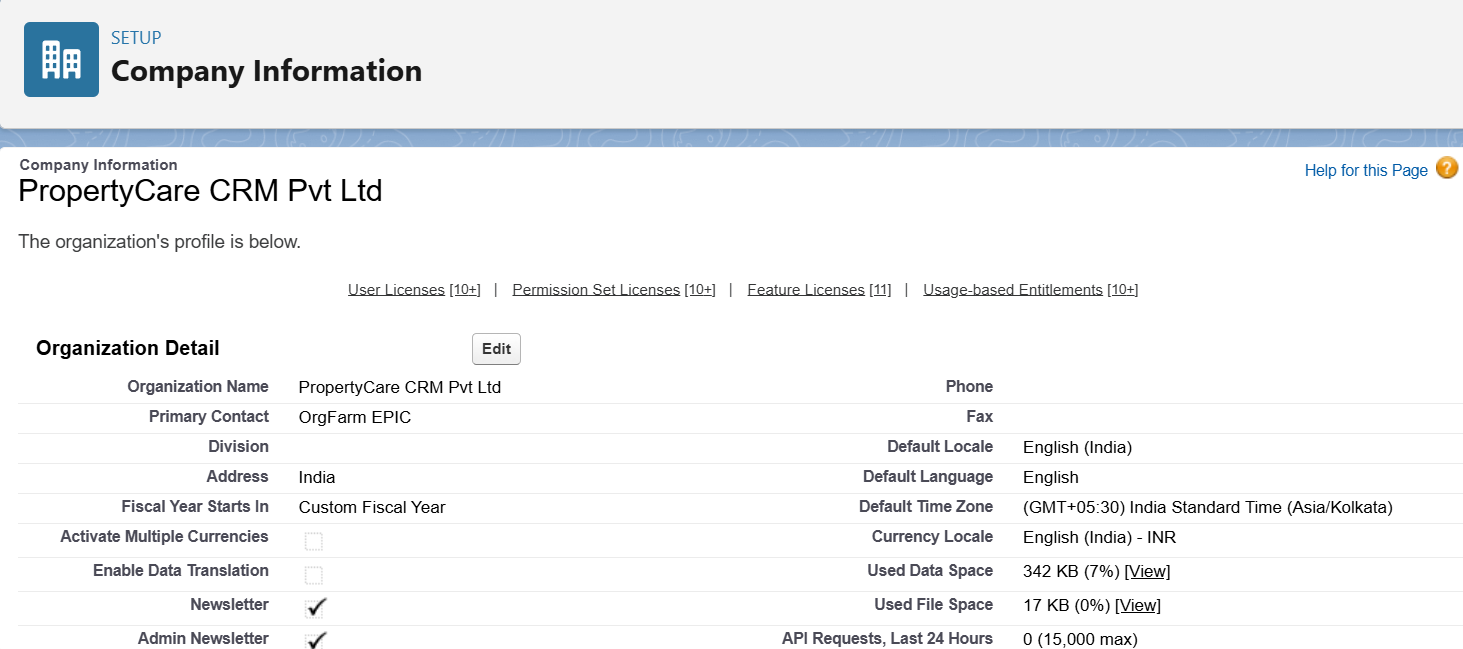
* **Requirement Gathering**
* Engage with key stakeholders such as the Real Estate CEO, Sales Agents, Property Managers, and Prospective Buyers to identify core needs.
* The system requirements include:
* Automate lead capture from website/social media.
* Track property inventory with price, location, and status.
* Allow scheduling and tracking of property visits.
* Generate sales funnel reports and revenue dashboards.
* **Stakeholder Analysis**
* Admin – Setup & configuration.
* Sales Agents – Manage leads, visits, and deals.
* Property Manager – Manage property listings.
* Manager/CEO – Monitor dashboards, approvals.
* Customers – Receive visit confirmations.
* **Business Process Mapping**
* Flow Example:

Customer Inquiry → Lead Captured → Lead Qualification → Assigned to Agent → Visit Scheduled → Property Manager Updates Availability → Manager Reviews → Deal Closed → Dashboard Updated

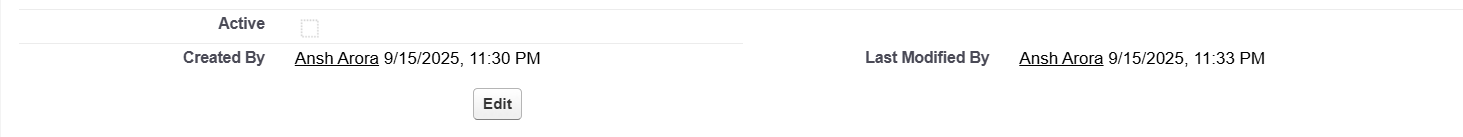
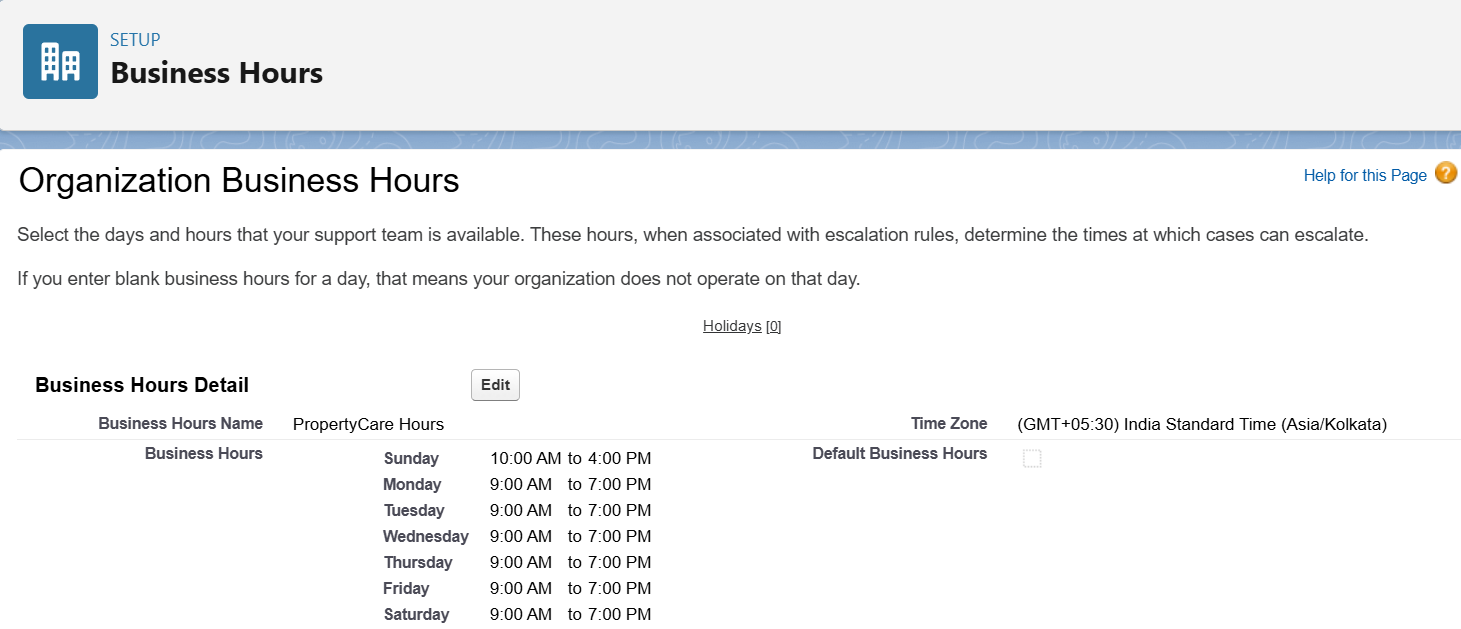
* **Industry-specific Use Case Analysis**
* Real Estate industry requires handling large volumes of inquiries, high-value deals, and accurate property availability.
* Lead qualification and scoring.
* Visit scheduling with notifications.
* Tracking buyer preferences and purchase history.
* Dashboards for sales performance and revenue trends.
* **AppExchange Exploration:** Search for “Real Estate CRM” apps. Some apps exist, but to learn and build end-to-end, we’ll develop a tailored solution with PropertyCare CRM.

**Phase 2: Org Setup & Configuration**

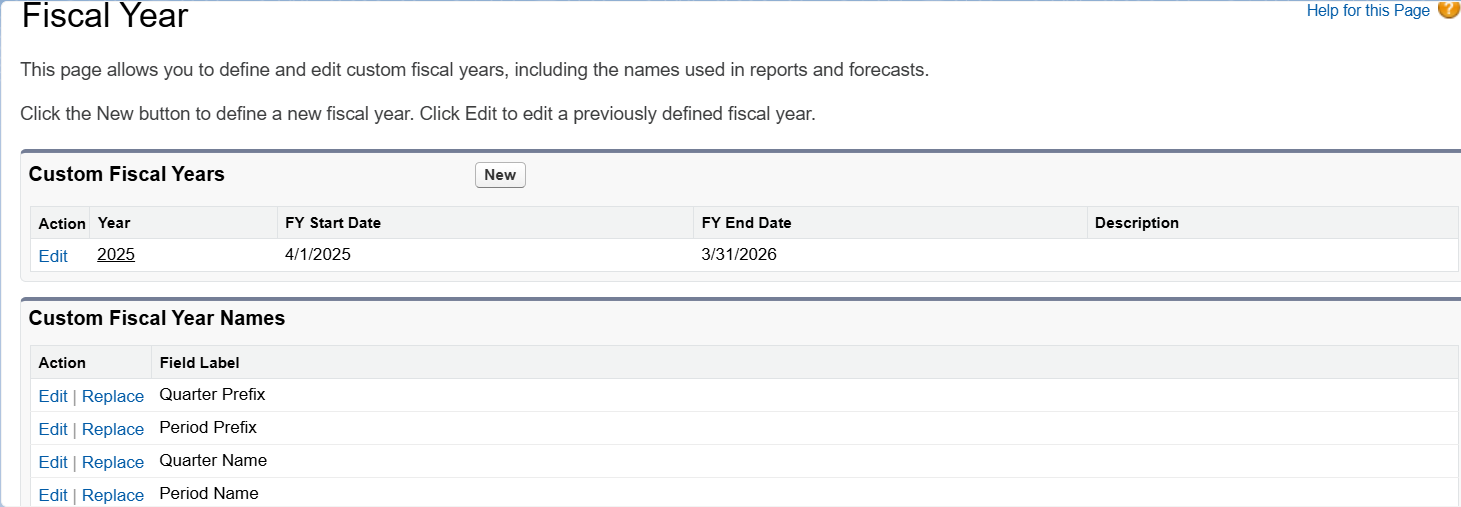
* **Salesforce Editions**
* Use **Salesforce Developer Edition** for this project.
* This edition provides full CRM features with free licenses for testing.
* **Company Profile Setup**
* Navigate: **Setup → Company Information**.
* Configure:
  + **Organization Name** = PropertyCare CRM Pvt. Ltd.
  + **Default Time Zone** = (GMT+05:30) India Standard Time.
  + **Currency** = INR.



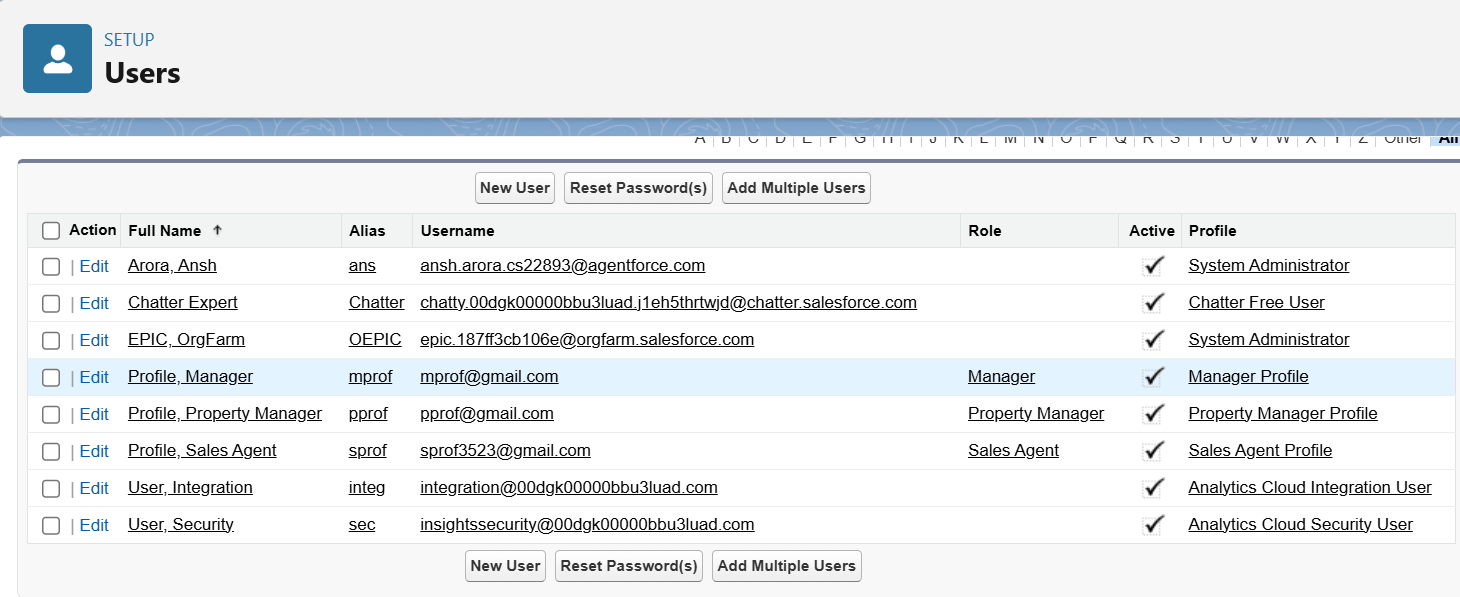
* **Business Hours Setup**
* Setup → **Business Hours** → Add:
  + Default Hours: Mon–Sun (9:00 AM – 7:00 PM).
* **Name**: PropertyCare Hours
* **Time Zone:** (GMT+05:30) India Standard Time (Asia/Kolkata)



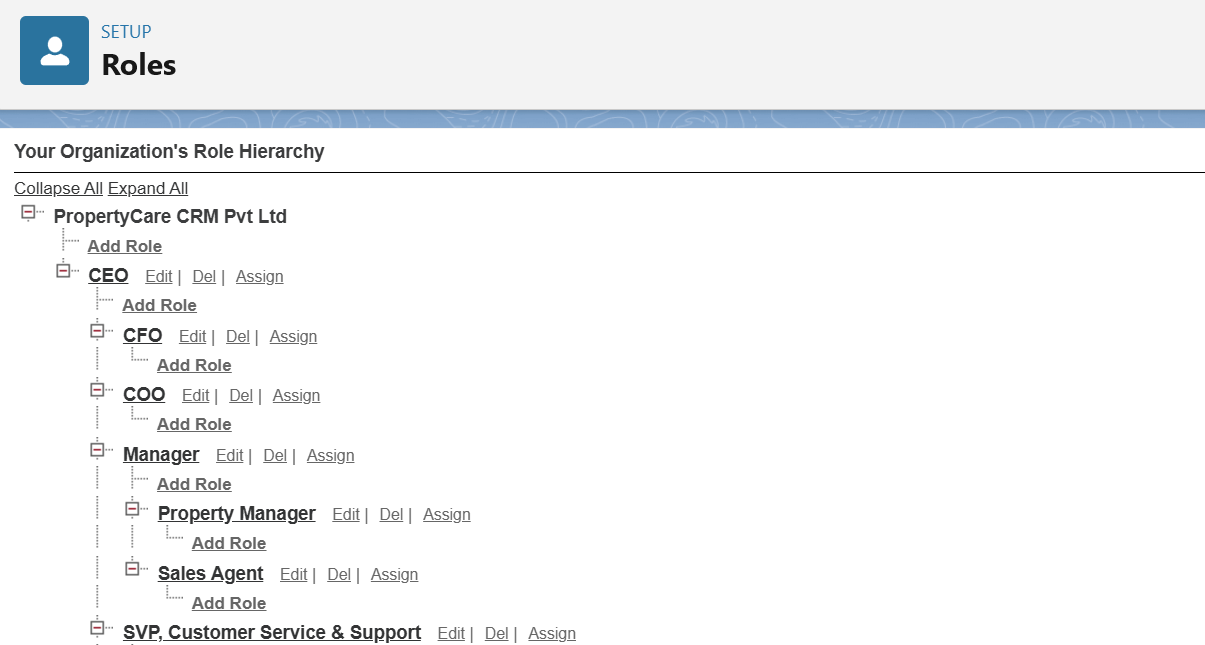
* **Fiscal Year Settings**
* Setup → **Fiscal Year**.
* Enable **Custom Fiscal Year** (April–March cycle).
* Quarters:
  + Q1: Apr–Jun
  + Q2: Jul–Sep
  + Q3: Oct–Dec
  + Q4: Jan–Mar



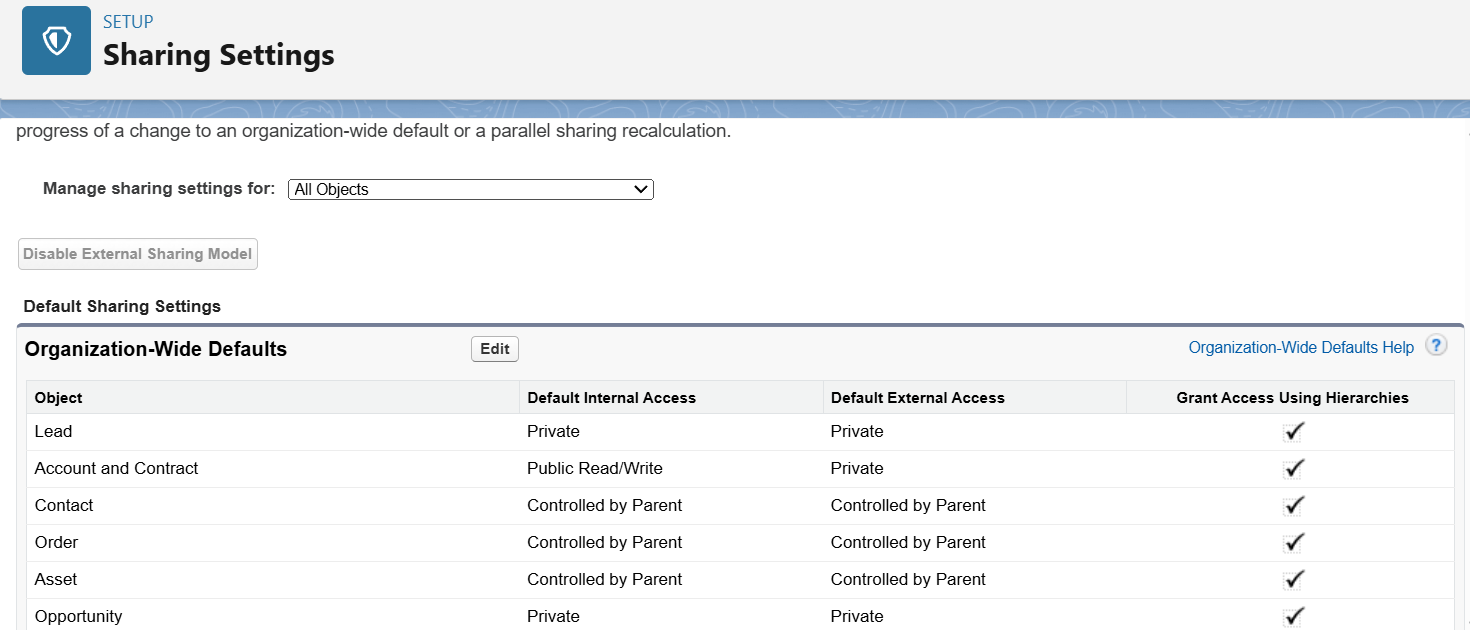
* **User Setup & Licenses**
* Create 3 Users:
  + **Manager** → Salesforce License → Manager Profile
  + **Property Manager** → Salesforce Platform License → Property Manager Profile
  + **Sales Agent** → Salesforce License → Sales Agent Profile



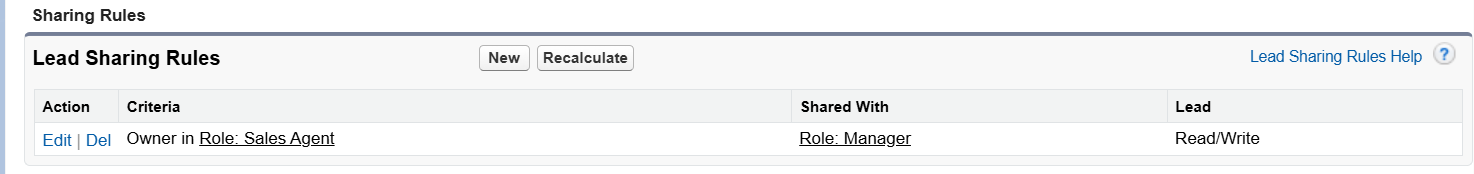
* **Roles**
* Setup → Roles → Set Up Roles.
* Create hierarchy:
  + CEO (top)
    - Manager
      * Property Manager
      * Sales Agent



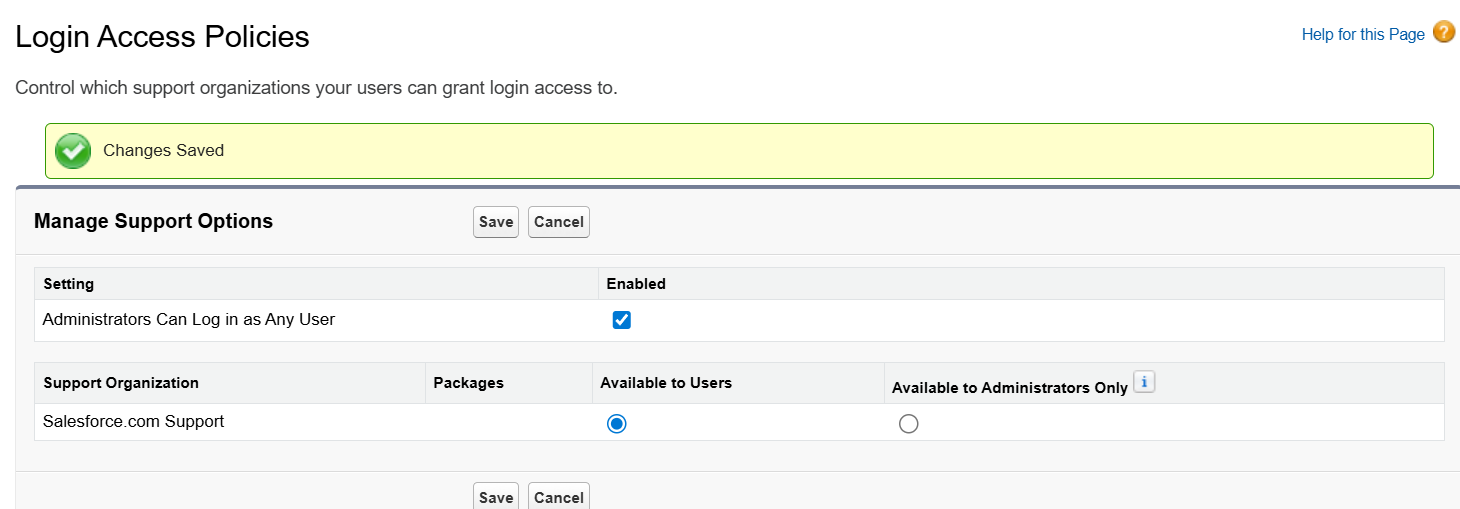
* **Profiles**
* Clone **Standard User** profile to create: Sales Agent, Property Manager, Manager Profiles.
* Assign object permissions based on role:
* Sales Agent → Leads, Opportunities, Properties
* Property Manager → Properties, Events
* Manager → Full access to all CRM objects
* Configure field-level security and app access as needed.
* Save and verify permissions.
* **Permission Sets**
* Setup → **Permission Sets → New**.
* Name: *Visit Scheduling Access*.
* Add permissions for **Events** (Read & Edit) and **Calendar.**
* Assign the Permission Set to Sales Agents.
* **Org-Wide Defaults (OWD)**
* Navigate: **Setup → Sharing Settings → Organization-Wide Defaults**
* Configure:
  + **Leads** → Private (Internal & External)
  + **Properties (Custom Object)** → Public Read Only (Internal & External)
  + Opportunities = Private (Internal & External)

* **Sharing Rules**
* Navigate: **Setup → Sharing Settings → Sharing Rules → New**
* Created rule for **Lead** object.
* Criteria: **Owner in Role = Sales Agent**.
* Shared With: **Role = Manager**.
* Access Level: **Read/Write**.
* This allows Managers to view and update Leads owned by Sales Agents.



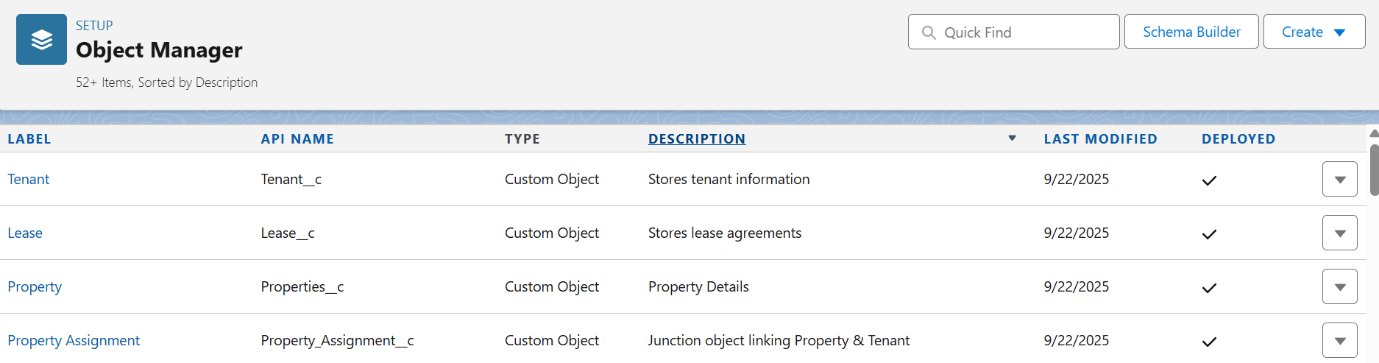
* **Login Access Policies**
* Setup → **Login Access Policies**.
* Enable: *Admins can log in as any user*.



* **Dev Org & Sandbox**
* Developer Org will act as sandbox.
* For advanced deployment → enable sandbox (Enterprise Edition only).
* **Deployment Basics**
* Setup → **Outbound Change Sets** (for metadata migration).
* Installed **VS Code + Salesforce Extensions** → connected with SFDX.

**Phase 3: Data Modeling & Relationships**

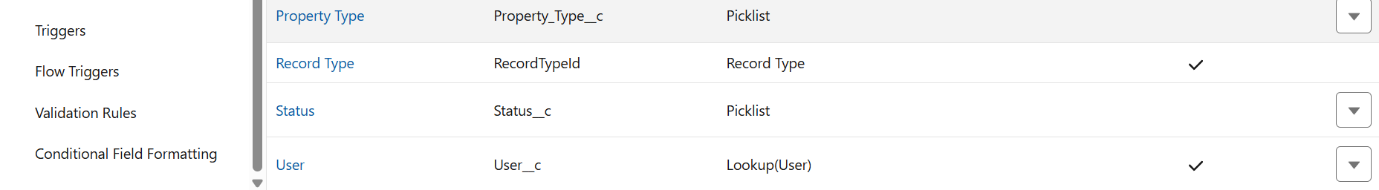
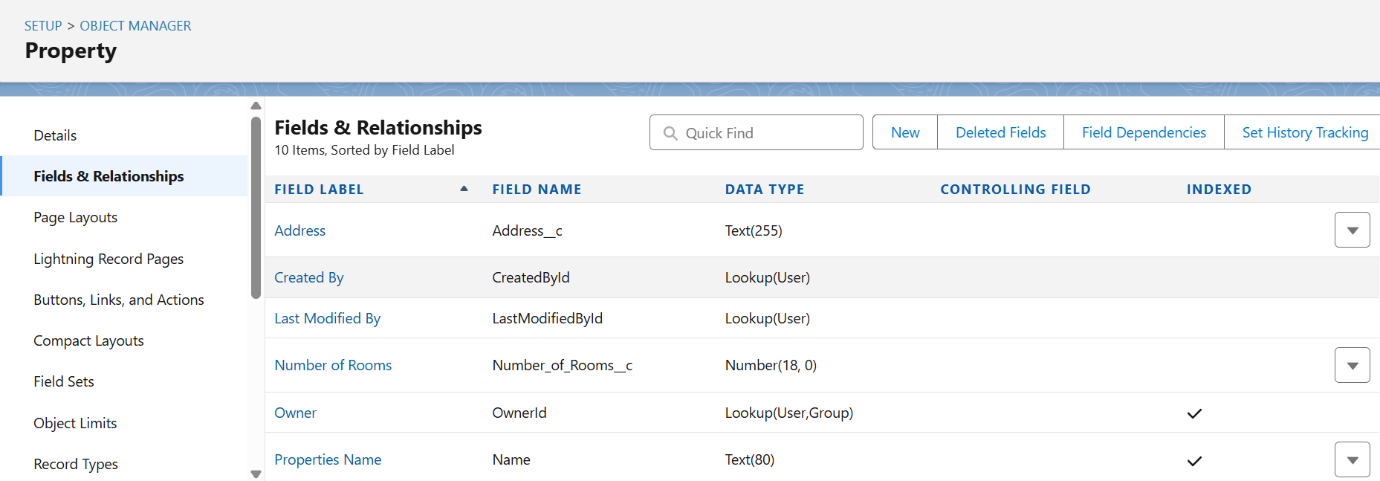
* **Create Custom Objects**
* **Objects Created:**
* **Property** – Stores property details. Record Name: Property Name (Text)
* **Tenant** – Stores tenant information. Record Name: Tenant Name (Text)
* **Lease** – Stores lease agreements. Record Name: Lease Number (Auto Number: L - {0000}
* **Property Assignment** – Junction object linking Property & Tenant. Record Name: Assignment Number (Auto Number: PA- {0000})
* **Procedure:**
* Setup → Object Manager → Create → Custom Object
* Fill Label, Plural Label, Record Name, Data Type
* Enable **Allow Reports** and **Allow Activities**
* Set Deployment Status = Deployed
* Save and repeat for all objects



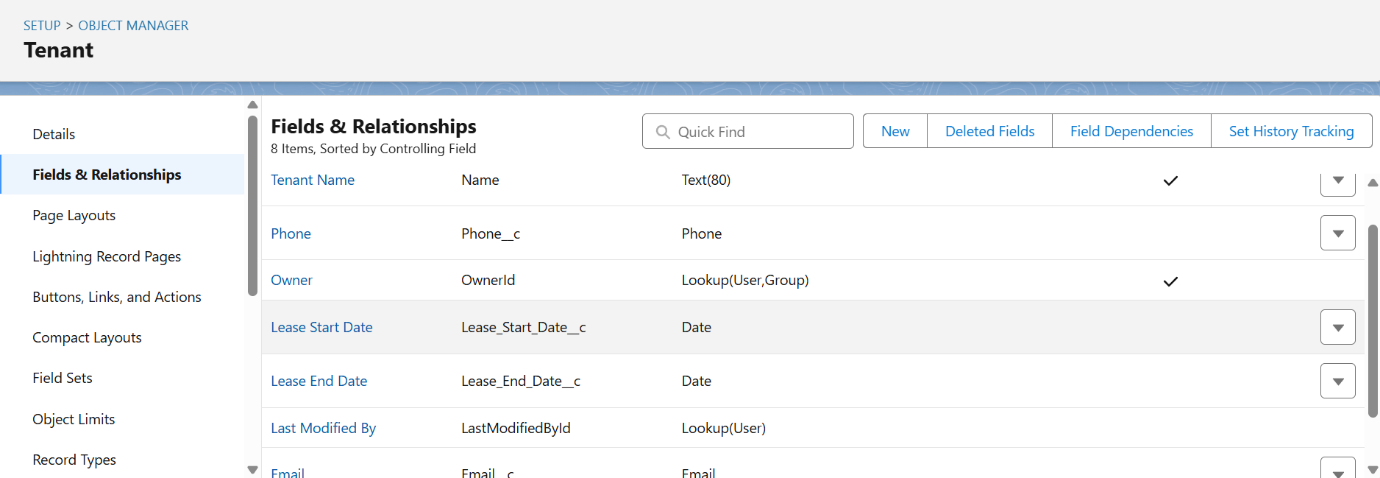
### Add Fields to Objects

Added fields for all objects to capture business data. Examples:

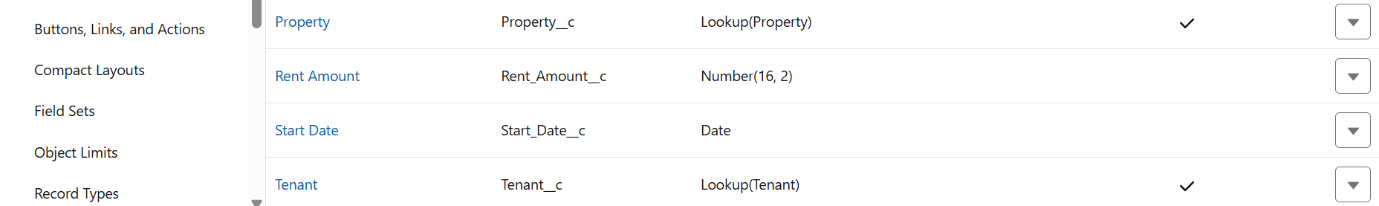
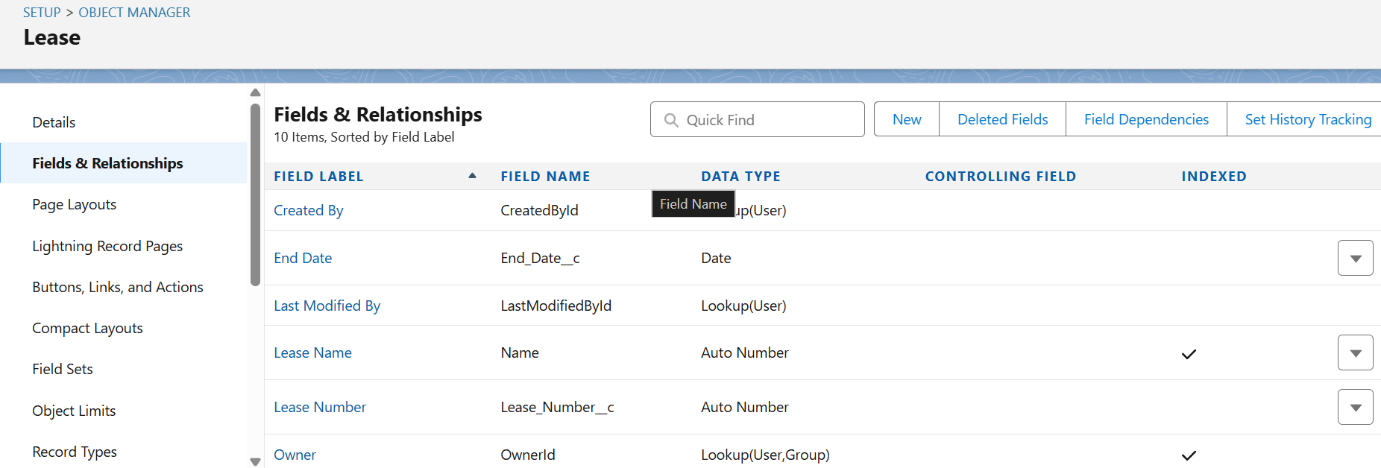
* **Property:** Property Type (Picklist), Address (Text), Manager (Lookup to User), Status (Picklist), Number of Rooms (Number), Warranty Expiry Date (Date Field), AMC Expiry Date (Date Field), Service Required (Checkbox Field), Assigned Service Agent (Lookup to User).



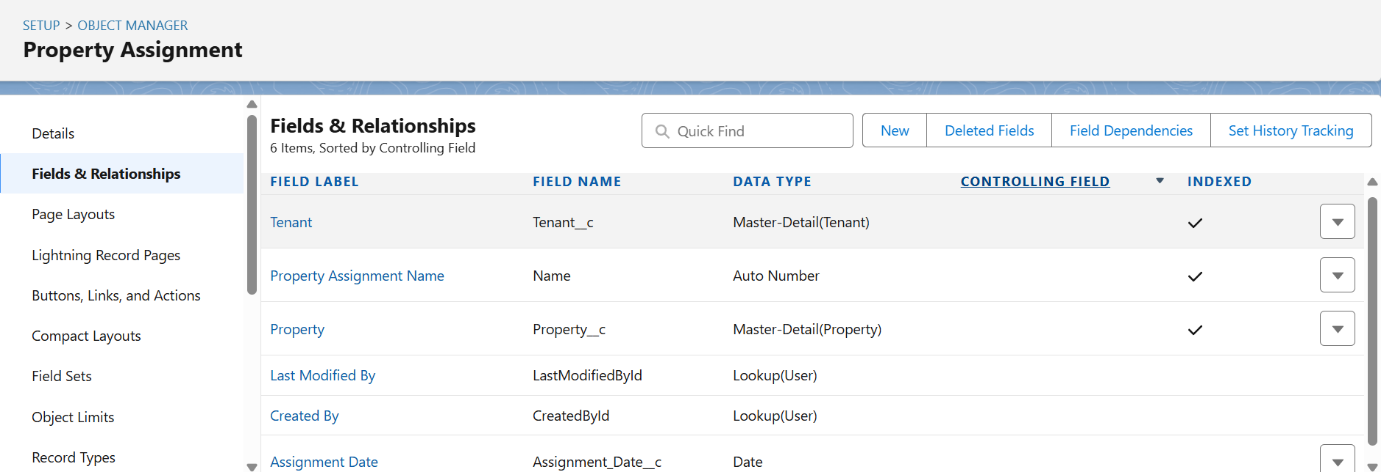
* **Tenant:** Tenant Name, Email, Phone, Lease Dates.



* **Lease:** Lease Number (Auto Number), Property (Lookup), Tenant (Lookup), Start/End Dates, Rent Amount.



* **Property Assignment:** Property (Master-Detail), Tenant (Master-Detail), Assignment Date, Assignment Number.



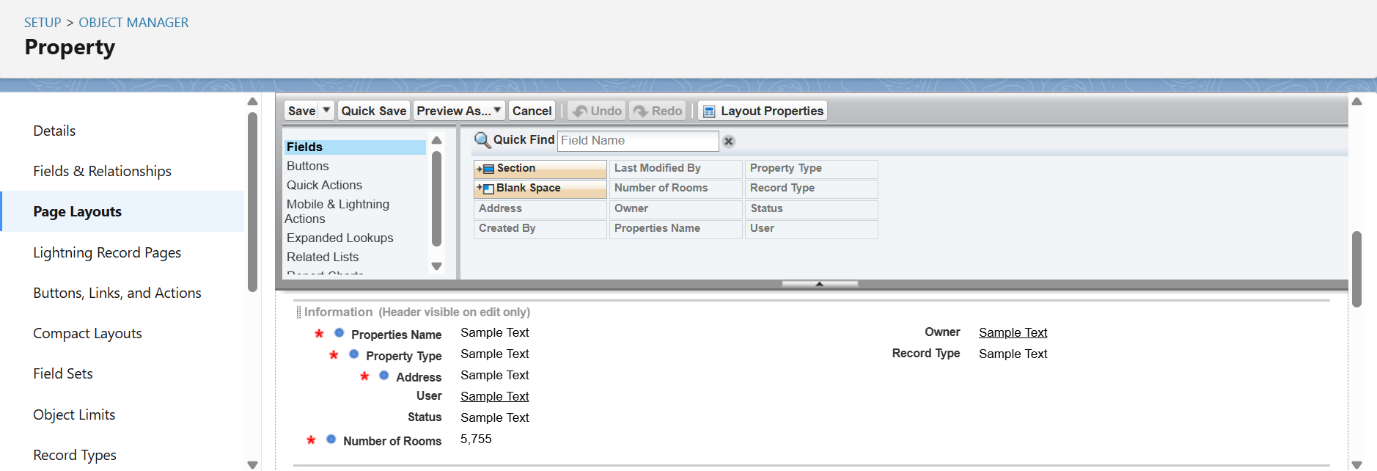
* **Record Types**

Record Types were configured to categorize records and control layouts:

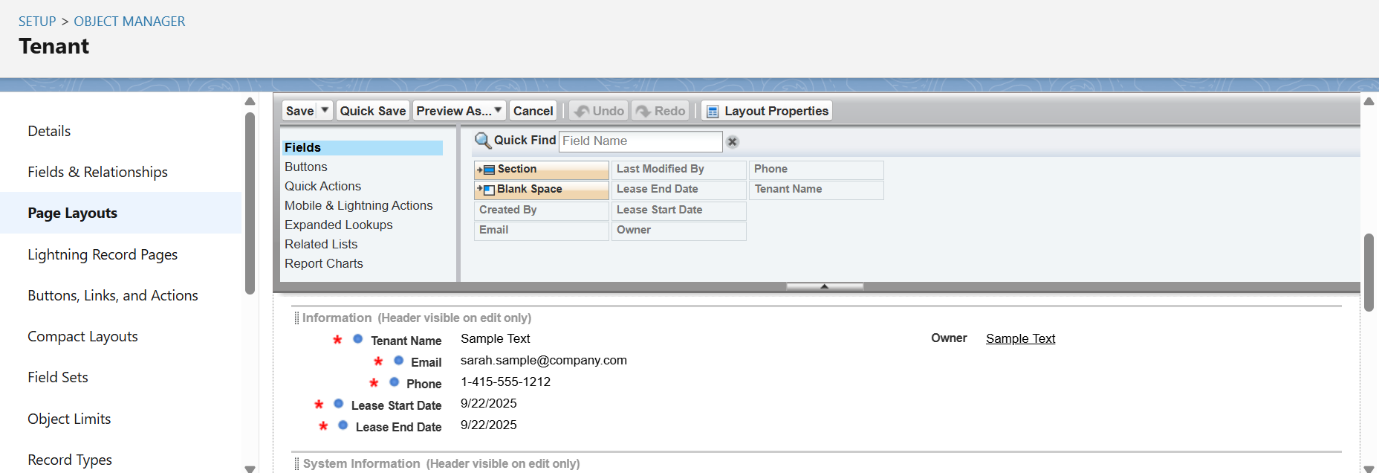
* **Property:** Residential and Commercial types with different page layouts.
* **Lease:** Active and Expired types.
* **Tenant & Property Assignment:** Master record type only.
* This ensures different layouts and picklist values are shown based on record type.
* **Page Layouts**

Page layouts were designed to display fields and related lists in an organized manner.

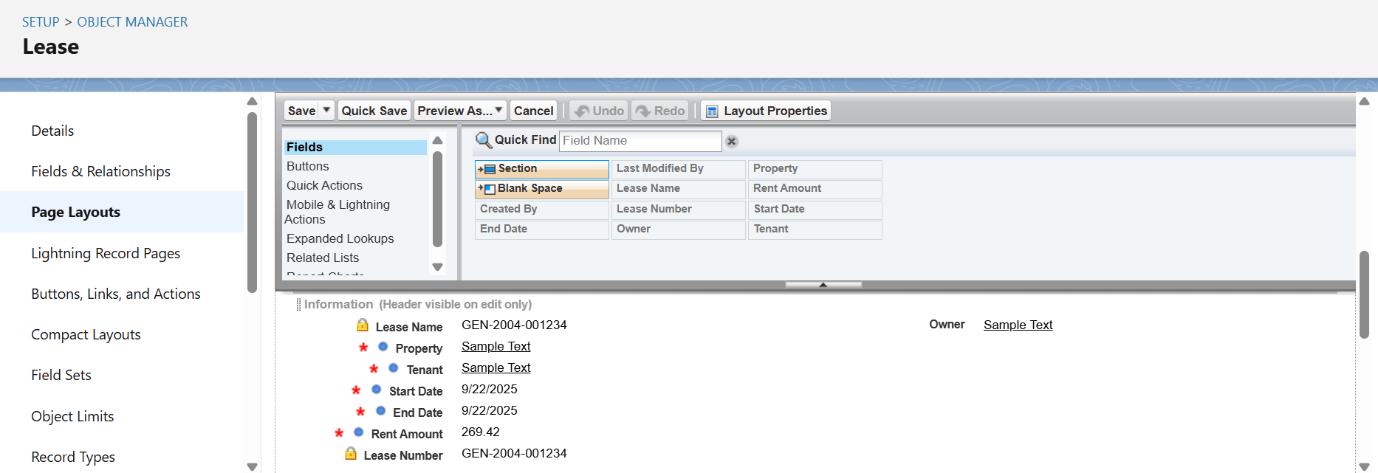
* **Property Section:** Property Name, Property Type, Address, Owner, Status, Number of Rooms



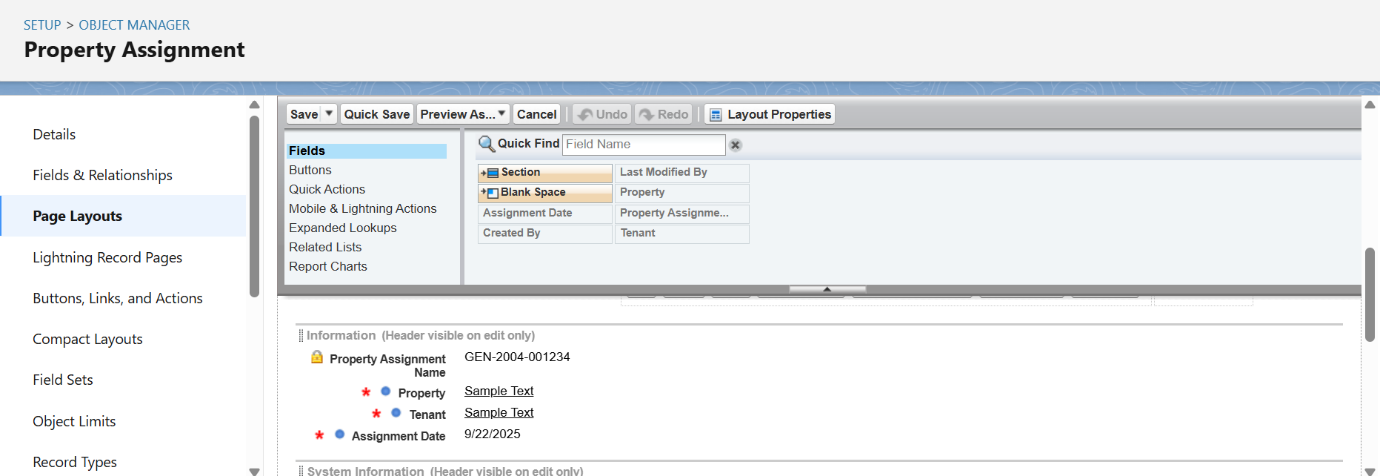
* **Tenant Section:** Tenant Name, Contact Number, Email, Address, Status



* **Lease Section:** Lease ID, Property (Lookup), Tenant (Lookup), Start Date, End Date, Rent Amount, Status



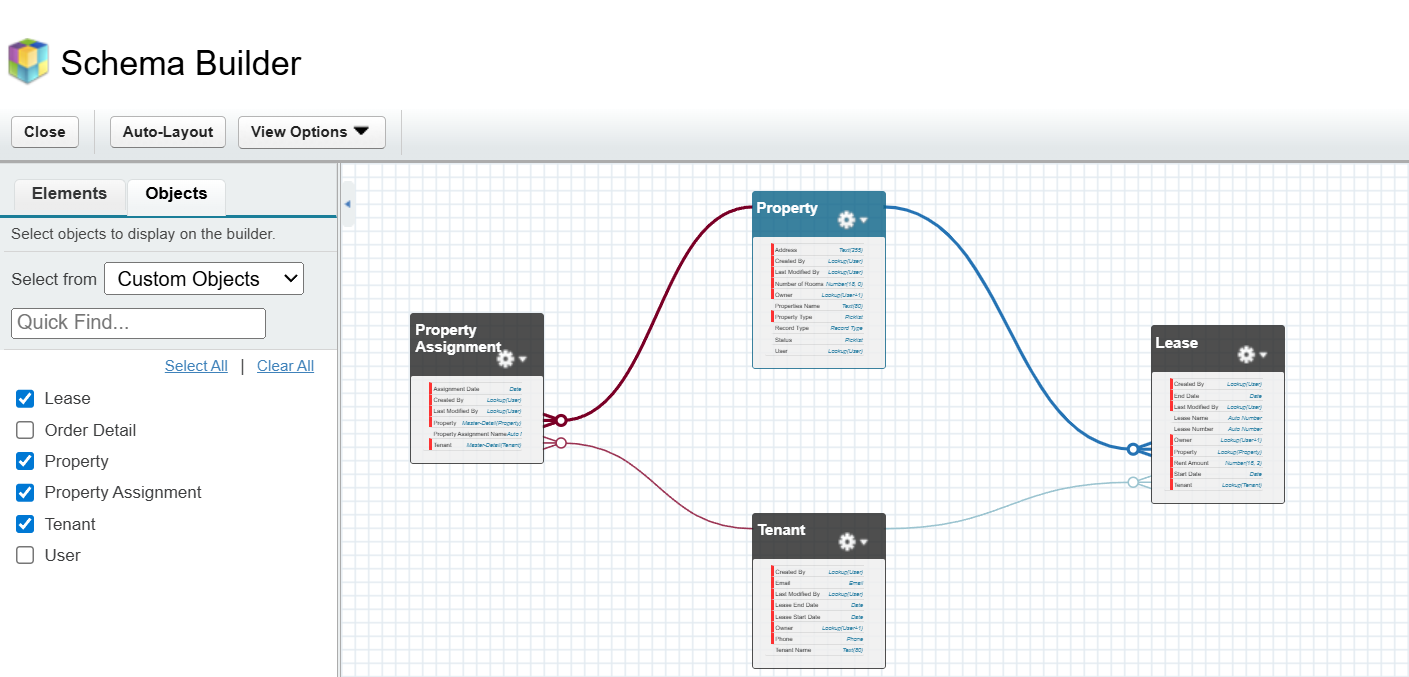
* **Property Assignment Section:** Property (Lookup), Tenant (Lookup), Assignment Date, Role



* **Compact Layouts**
* Compact layouts display the key fields in the highlights panel for quick reference. For the **Property** object, fields include Property Name, Property Type, Address, and Status. For the **Tenant** object, fields include Tenant Name, Contact Number, Email, and Status. For **Lease** and **Property Assignment**, key fields are Lease ID, Property, Tenant, Start/End Date, Assignment Date, and Role.

## Schema Builder

* Schema Builder provides a visual representation of objects, fields, and relationships in Salesforce.
* Selected objects like Property, Tenant, Lease, and Property Assignment are displayed on the canvas.
* Relationships such as lookup, master-detail, and junction are clearly shown.
* New fields or relationships can be added via drag-and-drop for quick configuration.
* Changes are saved and reflected in Object Manager and page layouts.
* It helps verify that all objects and relationships are correctly linked and structured.



## Lookup, Master-Detail & Hierarchical Relationships

* **Lookup Relationship:** Connects two objects independently; deleting the parent does not affect the child.
  + Example: Lease → Property, Tenant → Property.
* **Master-Detail Relationship:** Strong dependency; parent controls ownership, sharing, and deletion of the child.
  + Example: Property Assignment → Property, Property Assignment → Tenant.
* **Hierarchical Relationship:** Special lookup for User objects to define reporting hierarchies.
  + Label: Manager
  + Child Relationship Name: Subordinates
  + Allows viewing all subordinates reporting to a manager.
* Implemented to ensure proper object linkage, data integrity, and enforcement of business rules.

## Junction Objects

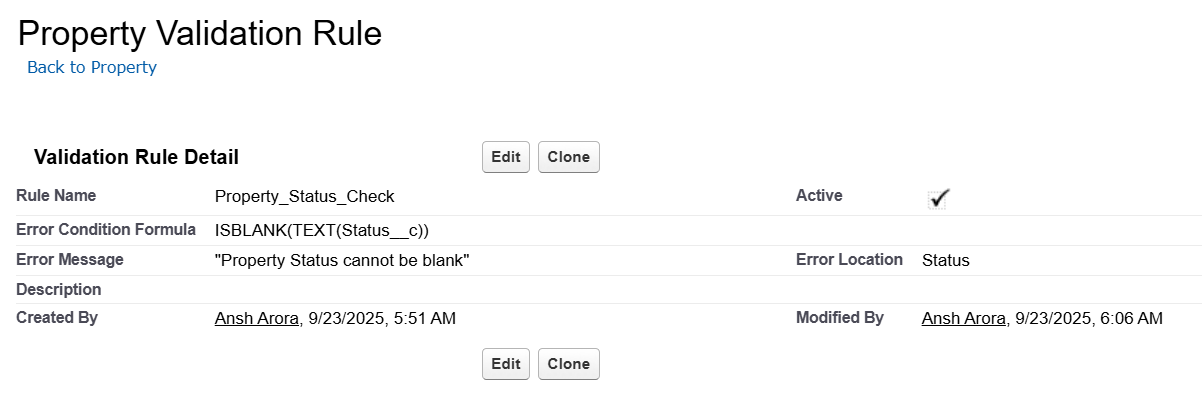
* Used to create many-to-many relationships between objects.
* Custom object connects two parent objects via master-detail relationships.
* Example: **Property Assignment** links **Property** and **Tenant**.
* Supports complex assignments and enforces business rules.

## External Objects

* Allow Salesforce to access data stored outside the platform without importing it.
* Configured via **Salesforce Connect** with mapped fields and relationships.
* Example: Access external property databases for reporting and reference.
* Enables real-time integration and keeps data up-to-date.

# Phase 4: Process Automation (Admin)

* **Validation Rules**
* **Rule Name:** Property\_Status\_Check
* **Formula**: ISBLANK(TEXT(Status\_\_c))
* **Error Message**: "Property Status cannot be blank"
* **Error Location**: Field → Status
* **Purpose**: Ensures every Property record has a Status before saving.

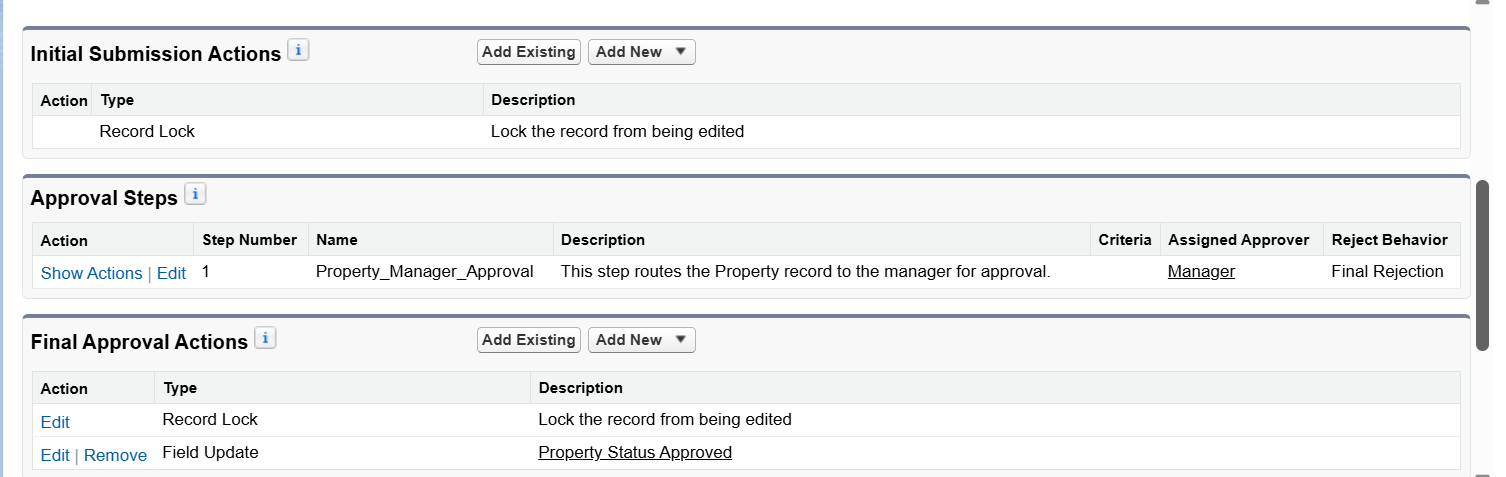
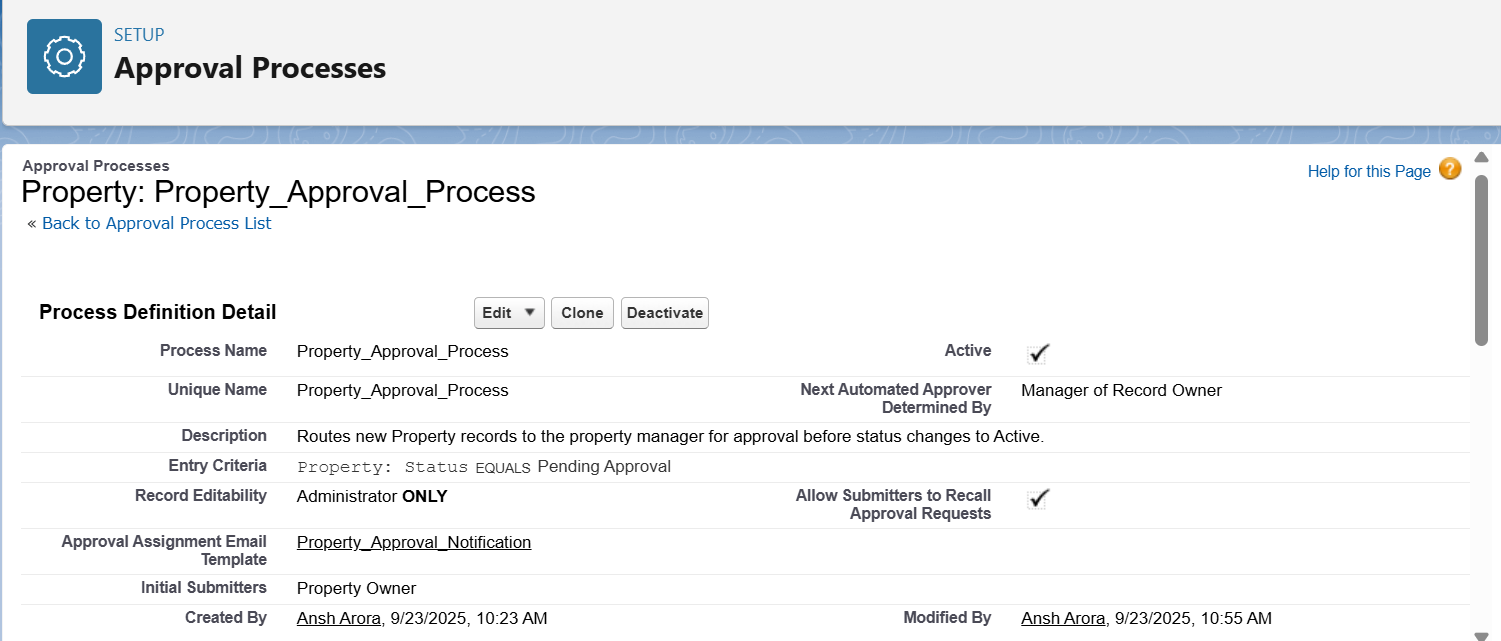


* **Flow Builder**
* **Purpose:** Automates business processes for Property records without code when records are created or updated.
* **Flow Type:** Record-Triggered Flow
* **Configure Start**
* **Object:** Property
* **Trigger:** A record is created (or updated if required)
* **Entry Conditions:** Field: Status | Operator: Equals | Value: Available | Logic: All conditions met (AND)
* **Update Records Action**
* **Label / API Name:** Update Property Record / Update\_Property\_Record
* **Description:** Updates the Property record triggered by the flow
* **Records to Update:** The record that triggered the flow
* **Filter Conditions:** None — always update record
* **Field Values:** Status = “Active”
* **Send Email Alert Action**
* **Label / API Name:** New\_Property\_Alert
* **Description:** Sends an email to the property manager when a new Property record is created
* **Email Template:** Custom template including Property Name, Type, and Address
* **Set Input Values:** Record ID = Id (triggering Property record)
* **Activation & Testing**
* Save and activate the process.
* Test by creating a new Property record to confirm Status updates and email alert is sent correctly.



### Approval Process

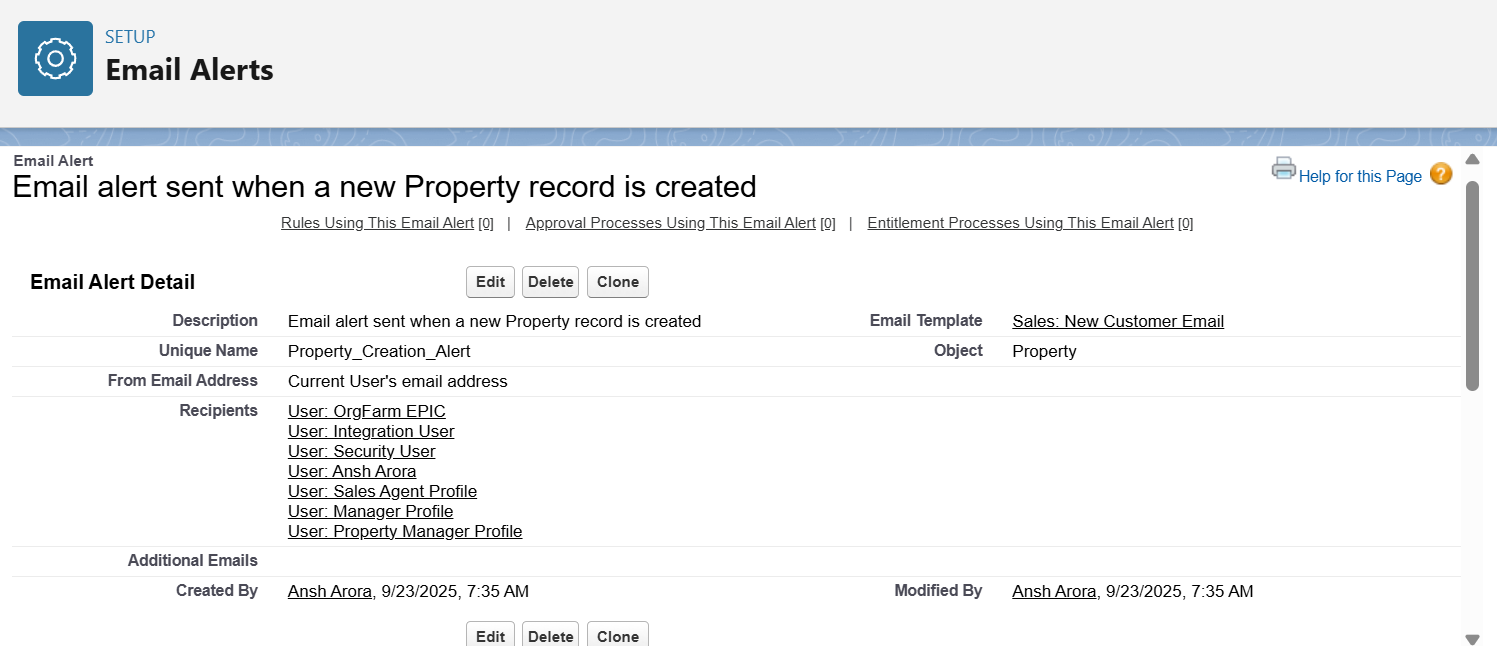
* **Purpose:** Automates approval of Property records before status changes to Active.
* **Process Name:** Property\_Approval\_Process
* **Next Automated Approver:** Manager of Record Owner
* **Entry Criteria:** Status = Pending Approval
* **Record Editability:** Administrators only; submitters can recall requests.
* **Approval Assignment Email Template:** Property\_Approval\_Notification (custom template)
* **Initial Submission Actions:** Lock the record to prevent editing.
* **Approval Step 1:** Assigned to Manager; Reject Behavior = Final Rejection.
* **Final Approval Actions:** Field Update → Status = Approved; Lock record.
* **Final Rejection & Recall Actions:** Unlock record; test by creating a new Property record.



* **Process Builder**
* **Process Builder** automates business processes without code by updating records, sending emails, or creating tasks. For the Property object, when a record is created with Status = *Available*, the process updates the Status to *Active*, sends an email to the Property Manager, and assigns a follow-up task to the Sales Team.

### Email Alerts

* Automatically sends an email when a new Property record is created.
* Uses the **Sales: New Customer Email** template.
* Recipients include Property Manager, Sales Agent, and other relevant users/profiles.
* Email contains key details: Property Name, Type, and Address.
* Ensures timely communication and proper follow-up.



### Tasks

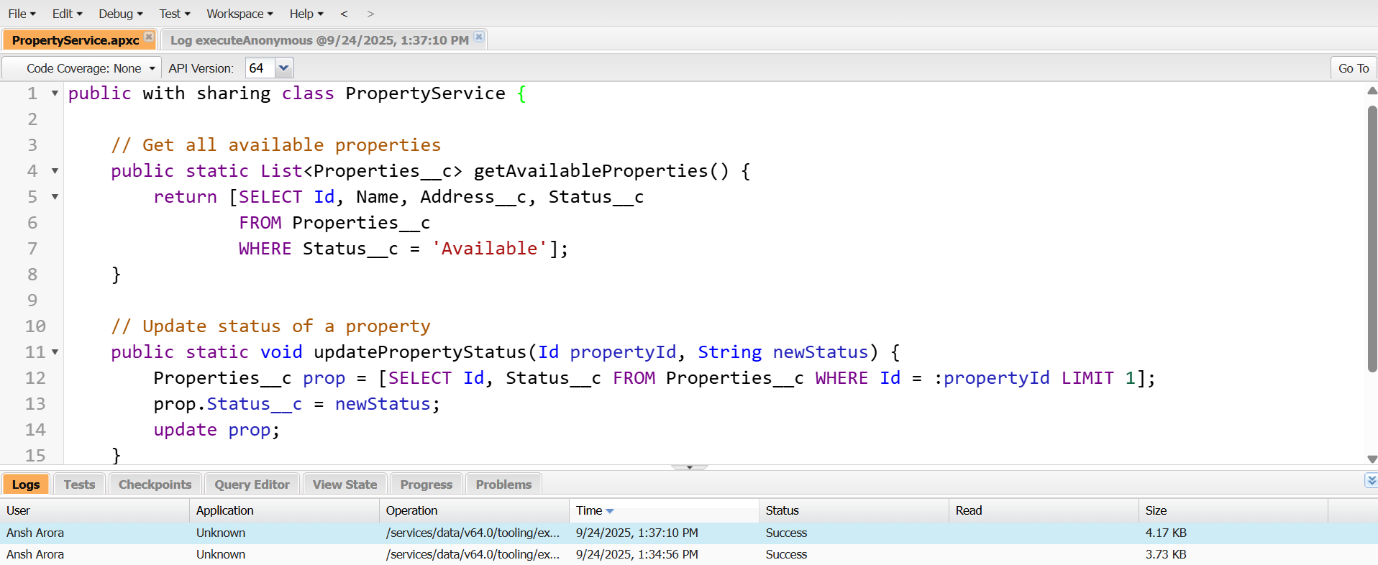
* Automatically creates tasks for users when specific conditions are met.
* Can be assigned to individual users, roles, or groups.
* Example: Assign a follow-up task to a Sales Rep when a new Property is created.
* Helps track work, deadlines, and follow-ups efficiently.
* Can be triggered via Workflow Rules, Process Builder, or Flow Builder.

**Phase 5: Apex Programming**

* **Classes & Objects**
* Create an **Apex class** to handle business logic for the Properties\_\_c object.
* Fetch available properties
* Update property status
* Reusable in triggers, flows, or Lightning components

#### Create Apex Class – PropertyService

* Go to **Setup → Developer Console → File → New → Apex Class**
* Enter class name: **PropertyService**
* Click ok
* Write Apex Code implementing the required logic.
* Save the class**.**



* Test the Class in Execute Anonymous Window

#### Go to Developer Console → Debug → Open Execute Anonymous Window



#### Create Apex Test Class – PropertyServiceTest

* Developer Console → File → New → Apex Class → **PropertyServiceTest**

****

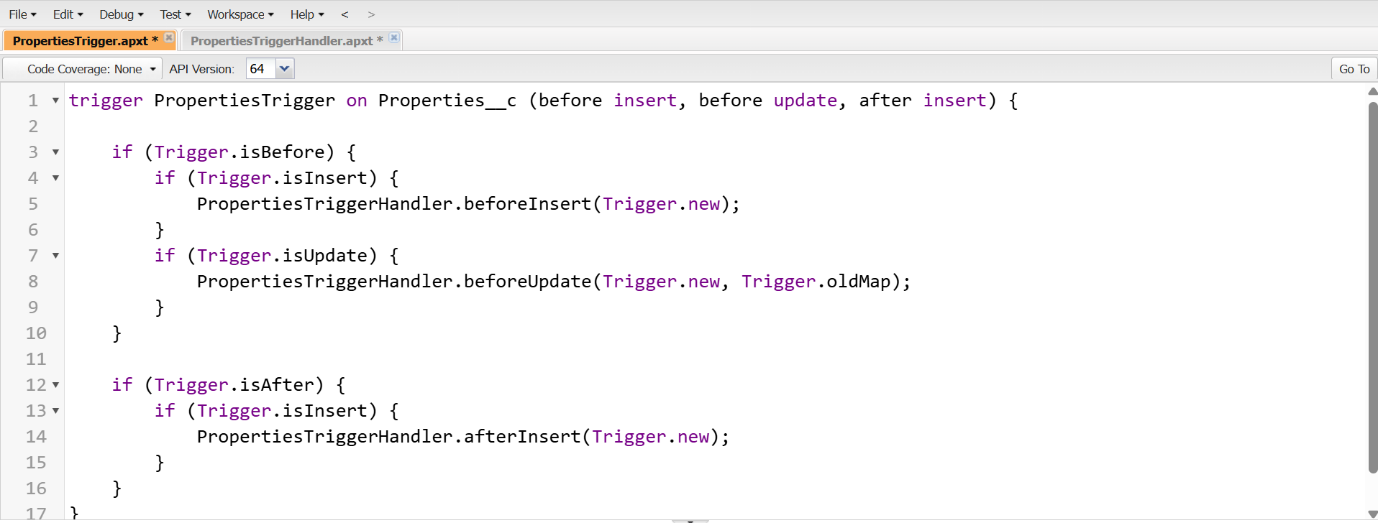
* **Apex Triggers**
* Create a **Trigger and Handler** for the Properties\_\_c object to:
* Set default Status\_\_c on insert
* Track status changes on update
* Perform post-insert actions (optional logging/debugging)
* This ensures **bulk-safe, maintainable, and reusable code**.

#### Create Apex Trigger – PropertiesTrigger

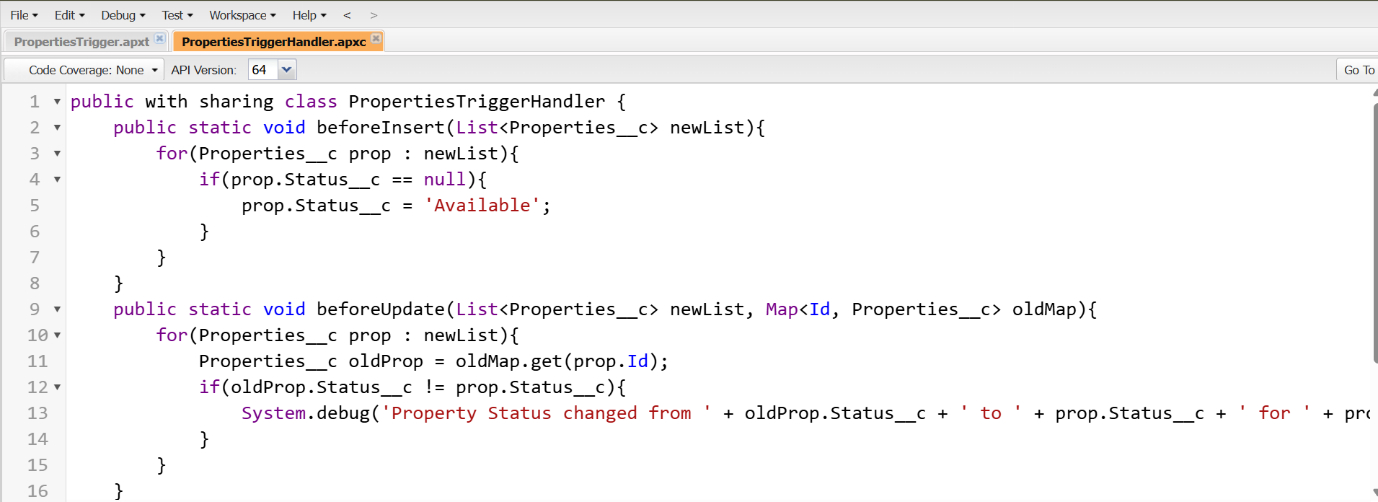
* Go to **Setup → Developer Console → File → New → Apex Trigger**

#### Name the trigger: PropertiesTrigger

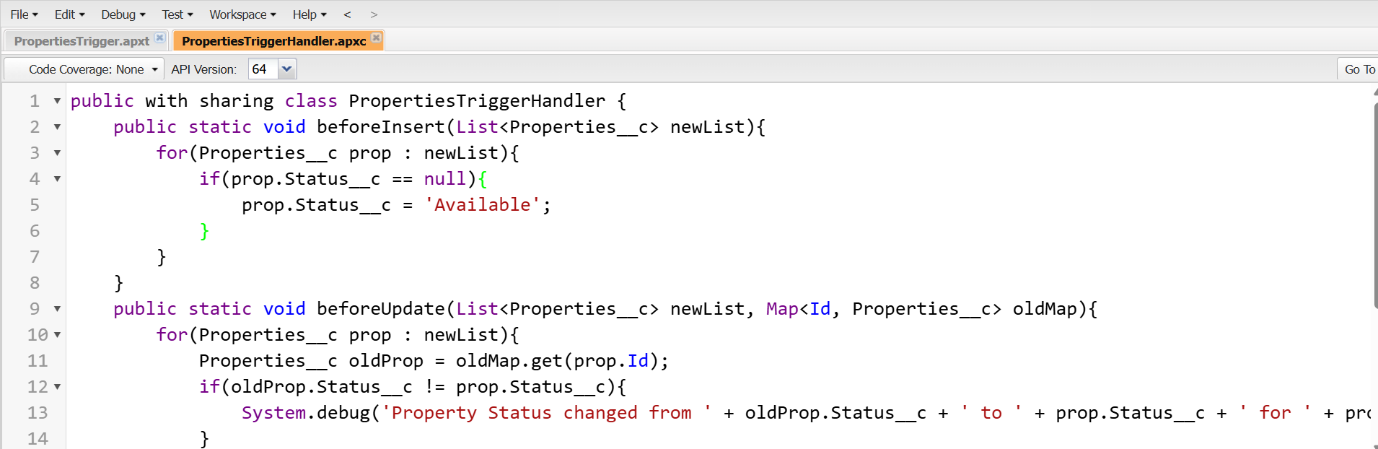
* Write Trigger code calling the handler.



#### Create Apex Class – PropertiesTriggerHandler



* **Trigger Design Pattern**
* Implement the **Trigger Handler Pattern** for Properties\_\_c so that:
* Triggers remain **thin** (minimal code inside trigger).
* All business logic resides in a **handler class**.
* Future enhancements (e.g., new status updates, validation rules) can be done in the handler without touching the trigger.
* Ensures **bulk-safe, reusable, and maintainable code**.



* **SOQL & SOSL**
* Use **SOQL (Salesforce Object Query Language)** and **SOSL (Salesforce Object Search Language)** to:
* Retrieve records from the Properties\_\_c object.
* Filter properties based on criteria like status or address.
* Perform global text searches efficiently.
* Ensure bulk-safe queries suitable for triggers, classes, and batch jobs.

### Collections (List, Set, Map)

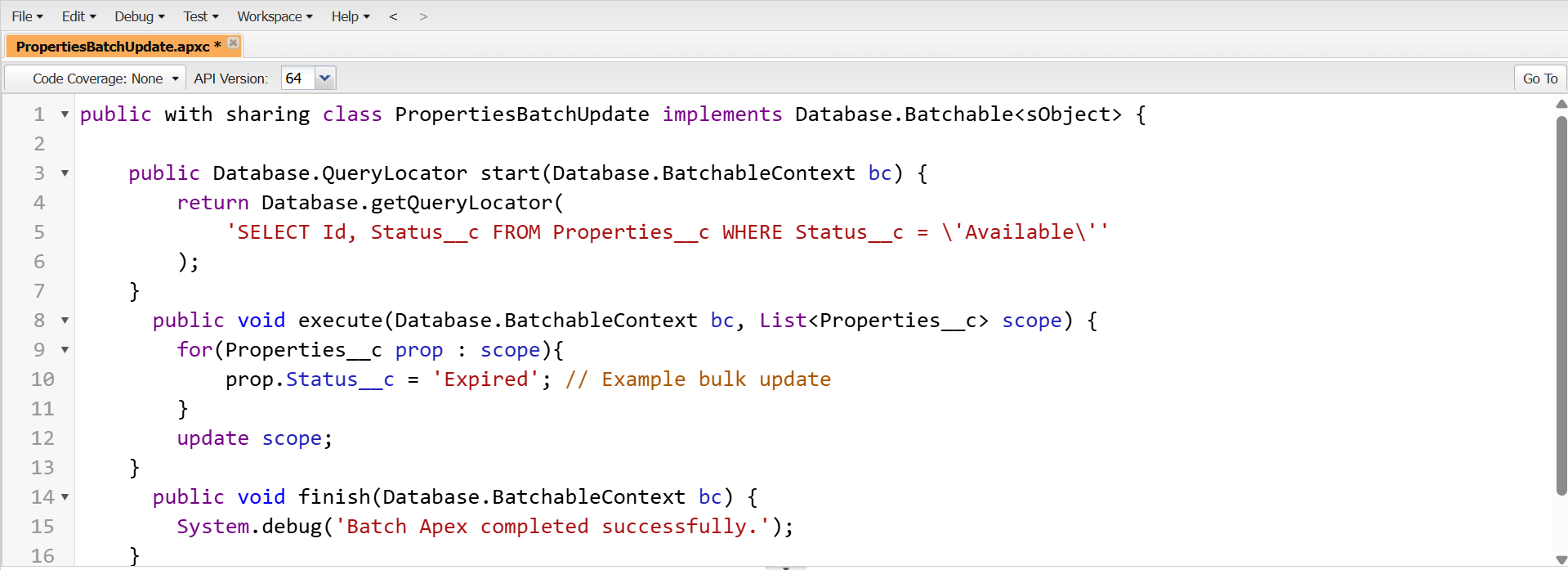
* Collections in Apex allow handling multiple Properties\_\_c records efficiently. **Lists** store ordered records for looping, **Sets** ensure unique values, and **Maps** provide fast key-value lookups. They make code bulk-safe and trigger-ready. Test classes validate that these collections work correctly and maintain data integrity

### Control Statements

* Control statements like If-Else and loops (For, While) manage the flow of Apex code. They automate logic such as updating property status and ensure bulk-safe, efficient execution.

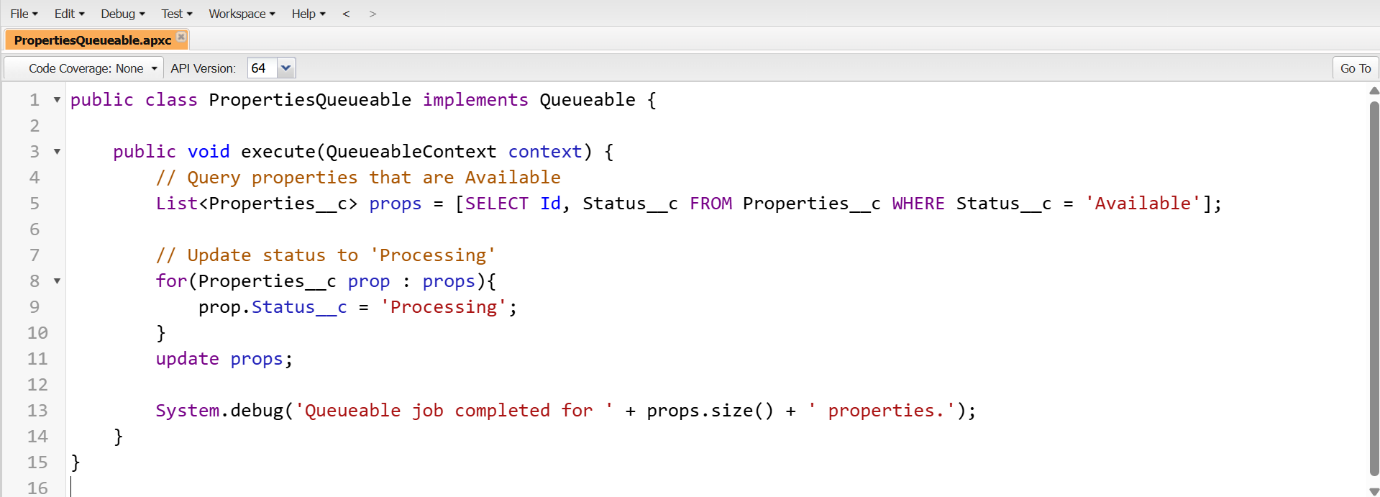
### Batch Apex Implementation

* **Objective:** Process large volumes of Properties\_\_c records asynchronously to update statuses or perform bulk operations.



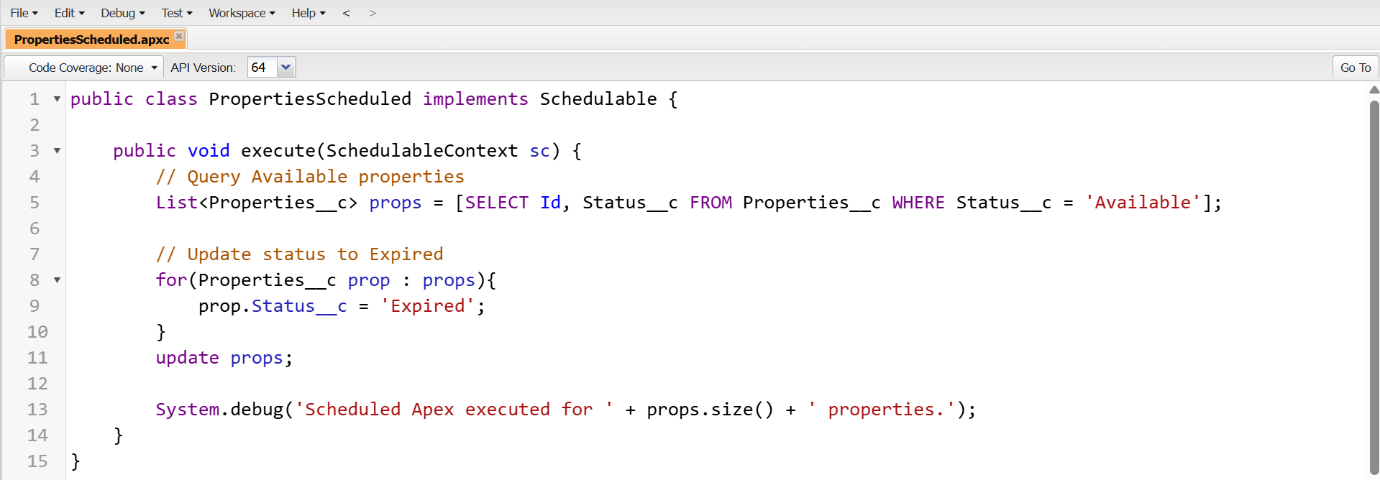
### Queueable Apex Implementation

* **Objective:**  
  Process Properties\_\_c records asynchronously, suitable for tasks that don’t need full Batch Apex but still require background processing.

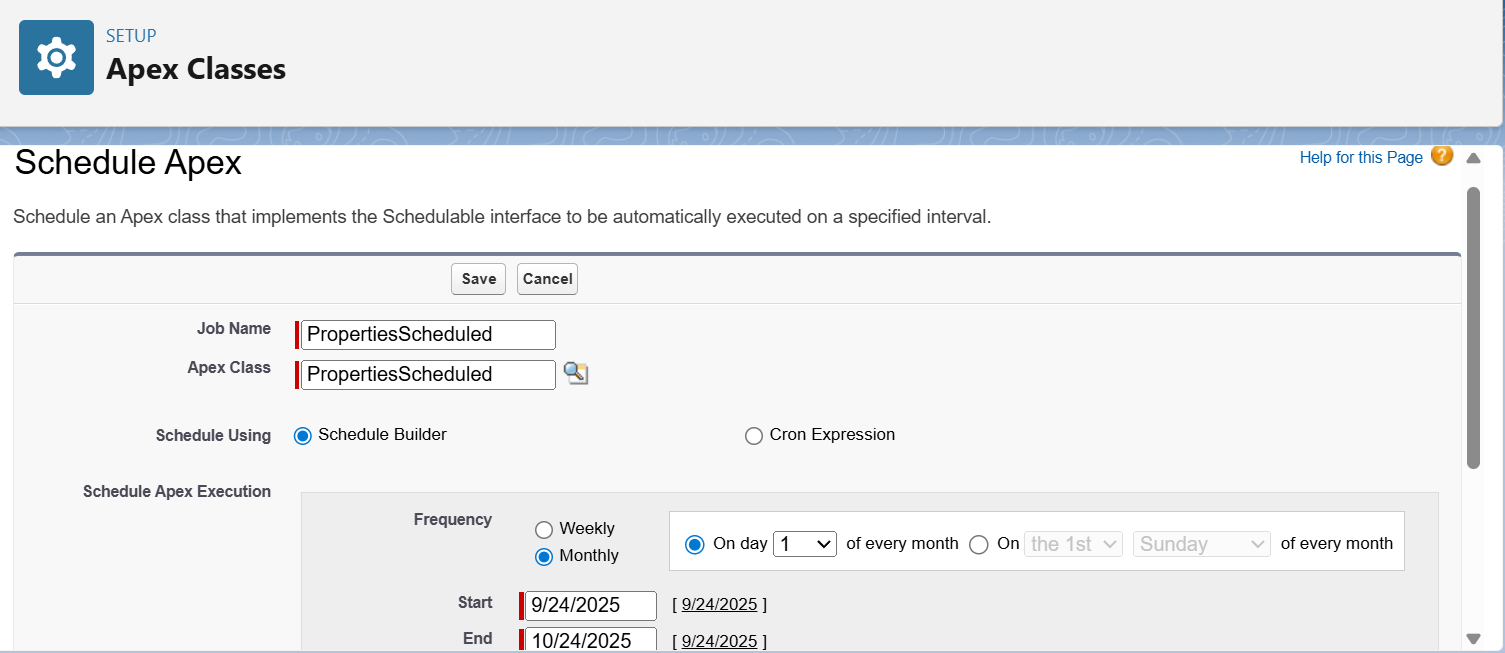


### Scheduled Apex

* **Objective:**  
  Automate recurring tasks on Properties\_\_c records, such as updating status or sending notifications, at a scheduled interval.



* **Schedule the Job**
  + Go to **Setup → Apex Classes → Schedule Apex**.
  + Enter Job Name, select PropertiesScheduled class, and set schedule (e.g., weekly).



* **Verify Execution**
  + Check **Setup → Scheduled Jobs** to confirm the job ran successfully.
  + Validate that Properties\_\_c records have updated Status\_\_c values.

### Future Methods

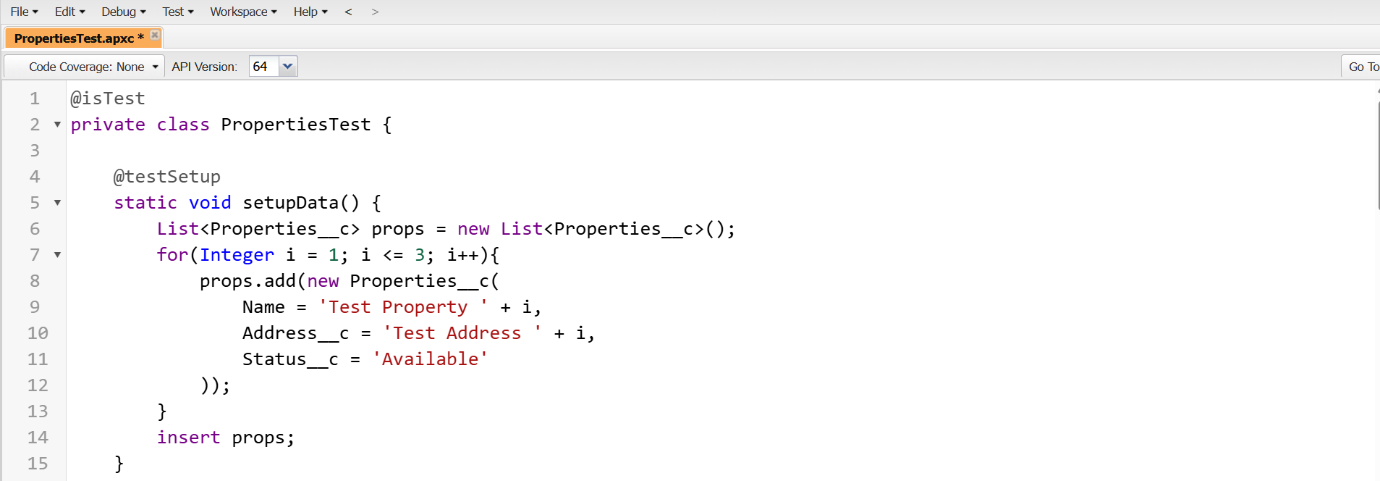
* Future methods allow asynchronous execution of tasks like sending emails or callouts, helping avoid governor limits. They run in the background and can be called from triggers or classes.

### Exception Handling

* Exception handling in Apex uses try-catch blocks to manage runtime errors gracefully. It ensures the application continues running, logs errors for debugging, and maintains data integrity during operations on Properties\_\_c records.

### Test Classes Implementation

* **Objective:**  
  Validate Apex classes, triggers, batch, and queueable jobs for Properties\_\_c and ensure Salesforce code coverage requirements are met.

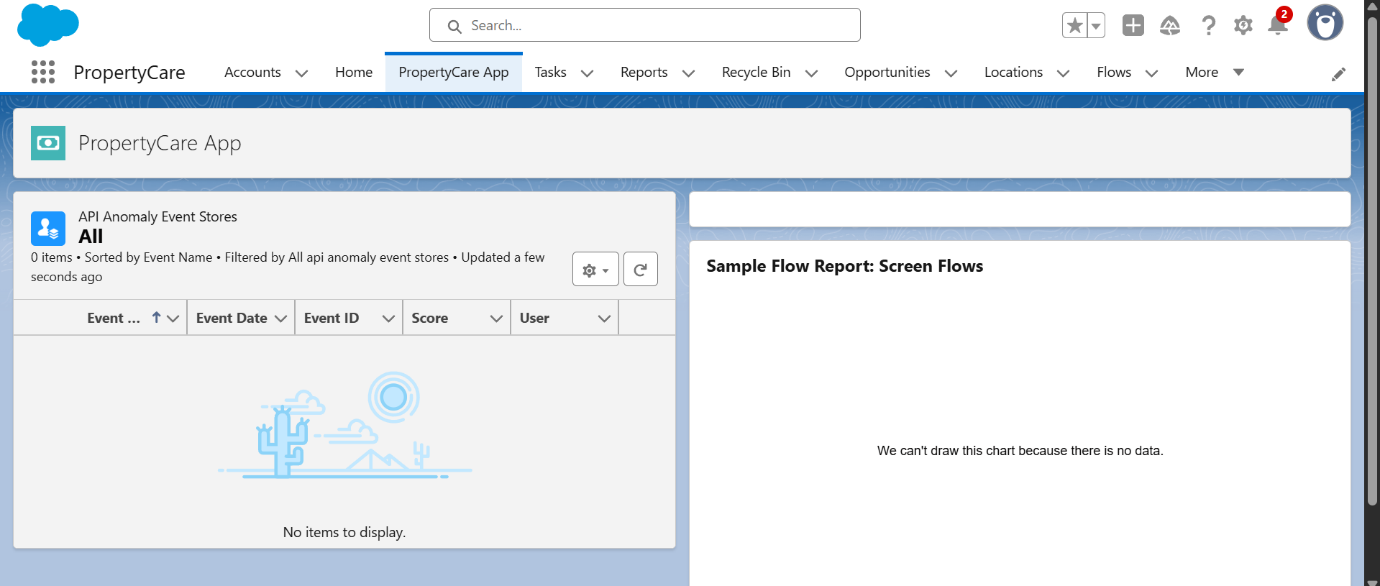


### Asynchronous Processing

* Asynchronous processing in Apex uses Batch, Queueable, Scheduled, and Future methods to handle large or long-running operations in the background. It improves performance, avoids governor limits, and ensures Properties\_\_c records are processed efficiently without blocking the user interface.

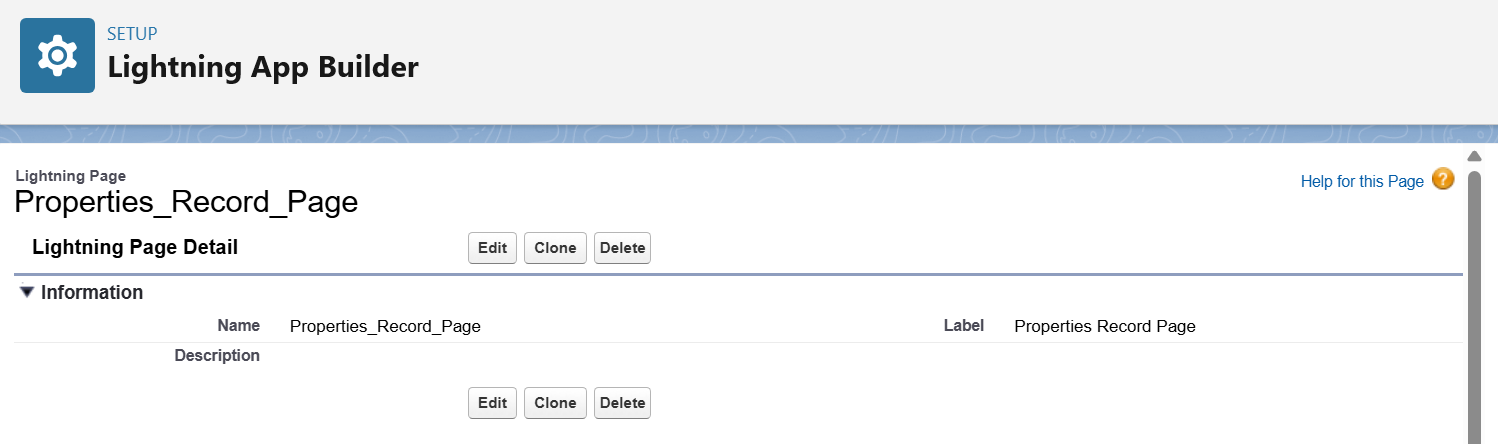
# Phase 6: User Interface Development

* **Lightning App Builder Implementation**
* Go to **Setup → Lightning App Builder → New** and select **Record Page**.
* Enter **Page Name**: PropertyCare App and select the **Property object**.
* Choose **2-Region layout** and open the canvas.
* Drag and drop **Report Chart, Rich Text, and List View** components into the regions and configure them as needed.
* **Save → Activate** the page and test it on a Property record.



### Record Pages Implementation

* Created a **custom Record Page** for the *PropertyCare* object using Lightning App Builder.
* Added a **Field Section** to display important fields (e.g., Name, Address, Status).
* Included **Tabs and Related Lists** for organizing details and related data.
* **Activated the page for both Desktop and Phone** to ensure a consistent user experience across devices.

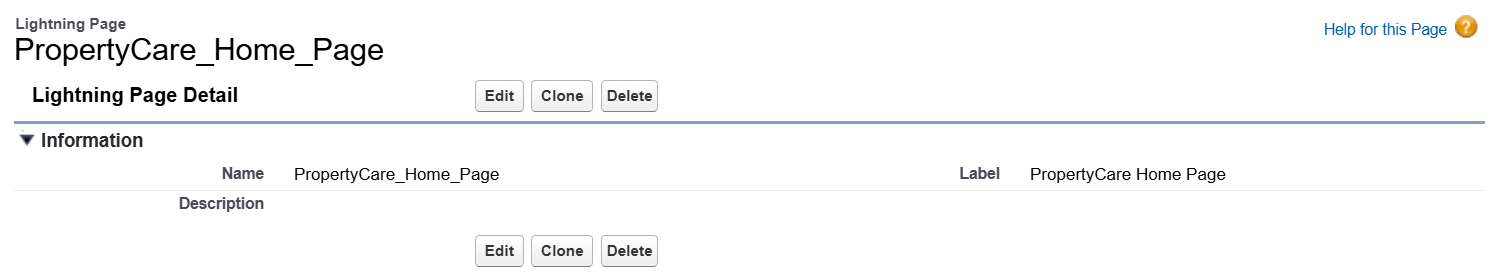


### Tabs Implementation

* Created **custom Tabs** in Salesforce for easy navigation across modules in the PropertyCare app.
* Added tabs for **Property, Tenant, and Maintenance objects** to quickly access records.
* Included a **Custom Lightning Page Tab** to show the Record Page designed in App Builder.
* Ensured tabs are visible in both **Desktop and Mobile navigation menus** for better usability.

### Home Page Layouts

* Created a **custom Home Page** using Lightning App Builder with a **1-column template** for a clean, vertical layout.
* Added **standard and custom components** like **Tasks, Reports, and Notifications** stacked vertically.
* Designed the layout to **highlight important information** for PropertyCare users in a single column.
* Activated the Home Page for both **Desktop and Mobile** to ensure a consistent user experience.



### Utility Bar

* Arranged **Utility Bar components** by dragging and dropping them in the desired order.
* Placed the most important tools like Notes, History and recent items at the beginning for **quick access**.
* Verified the **Utility Bar on Desktop** to ensure proper order and usability.

### Lightning Web Components (LWC)

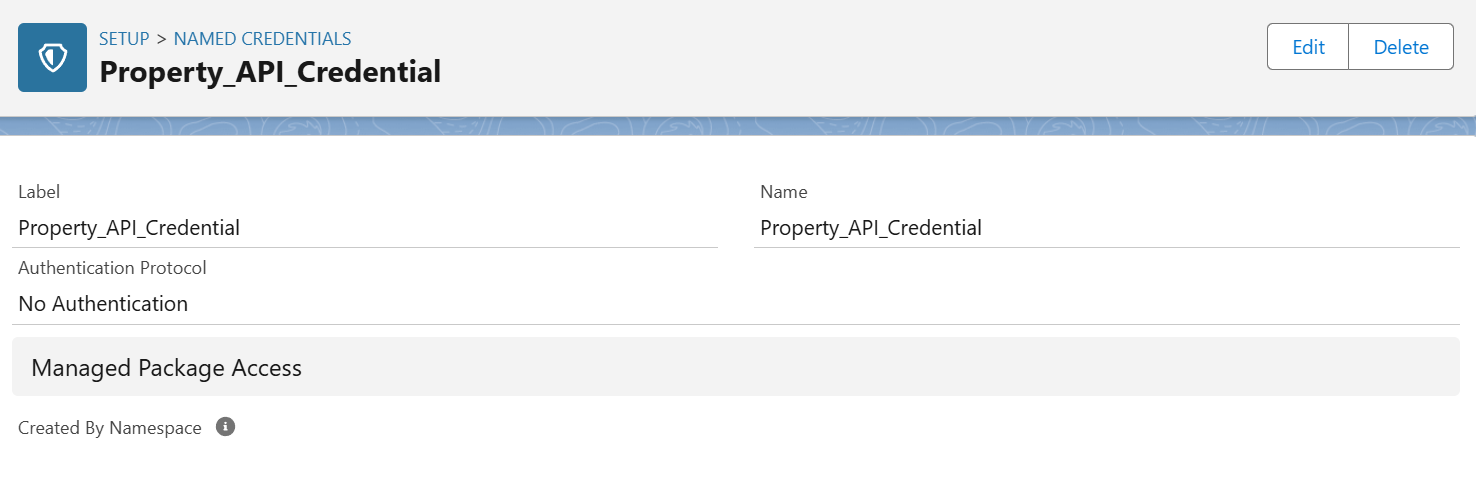
* Created a **custom LWC** using VS Code and Salesforce CLI to add dynamic functionality.
* Added the LWC to the **PropertyCare Record Page** for displaying interactive data.
* Tested the component on **Desktop and Mobile** to ensure proper functionality.
* **Apex with LWC**
* Connected LWCs to server-side Apex methods using AuraEnabled.
* Enabled dynamic fetching and updating of Salesforce data.
* Supports both reactive wire and on-demand (imperative) calls.
* **Events in LWC**
* Implemented custom events for communication between child and parent components.
* Used to pass data or trigger actions across components.
* Helps maintain a clean separation of logic and UI.
* **Wire Adapters**
* Automatically fetch and reactively display Salesforce data in LWCs.
* Keeps the UI updated without manual intervention.
* Ideal for lists, record details, and real-time data updates.
* **Navigation Service**
* Used Navigation Mixin to programmatically redirect users.
* Supports navigating to records, pages, and external URLs.
* Enhances workflow by guiding users after actions like saving or approving records.

# Phase 7: Integration & External Access

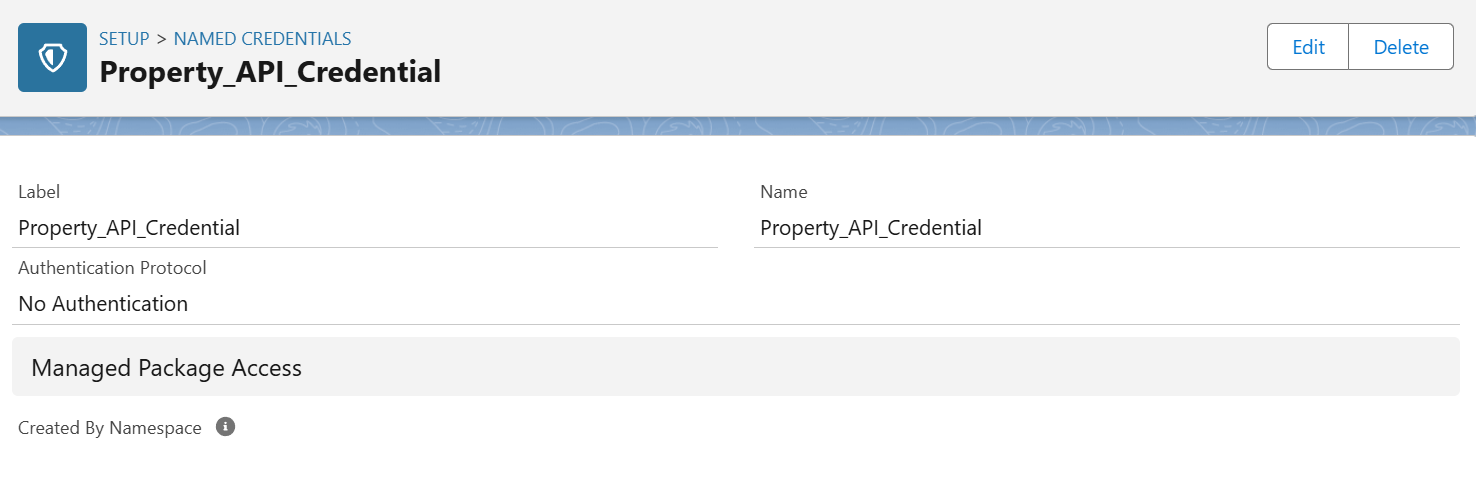
* **Named Credentials**
* Create Named and External Credentials to securely access external APIs in PropertyCare CRM.
* Setup → Quick Find → Named Credentials → Click Named Credentials
* Created Named Credential called Property\_External\_API.



Navigate to **Setup → Quick Find → External Credentials → Click External Credentials**



* **External Services:**
* Salesforce External Services let you register external APIs so their endpoints become **invocable actions** for Flows or Apex, enabling automation in PropertyCare CRM.



### Web Services (REST/SOAP)

### Salesforce allows exposing data via REST or SOAP classes and consuming external APIs using Apex HTTP methods.

### You can create REST/SOAP Apex classes to provide or fetch data and test them with Postman, Workbench, or Apex test classes.

* These web services enable real-time integration and automation with external systems.

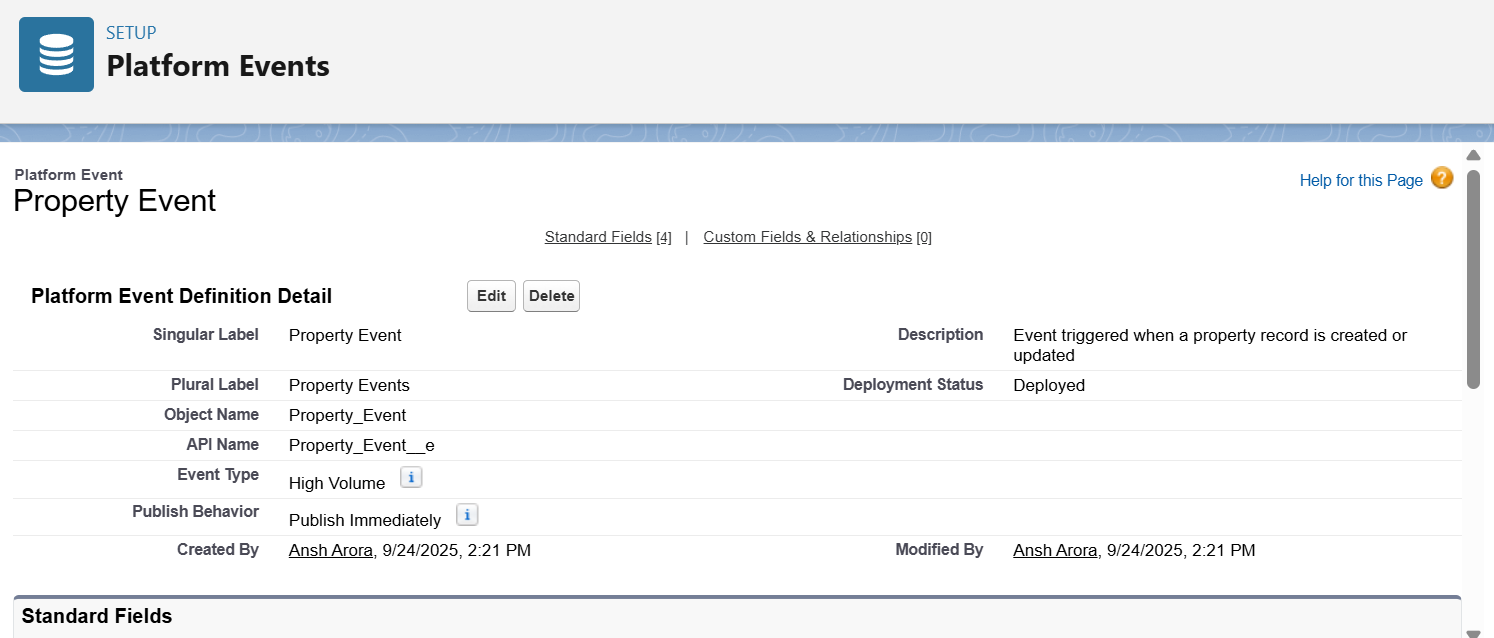
### Callouts

### Salesforce callouts let you send or retrieve data from external APIs using Apex or Flows.

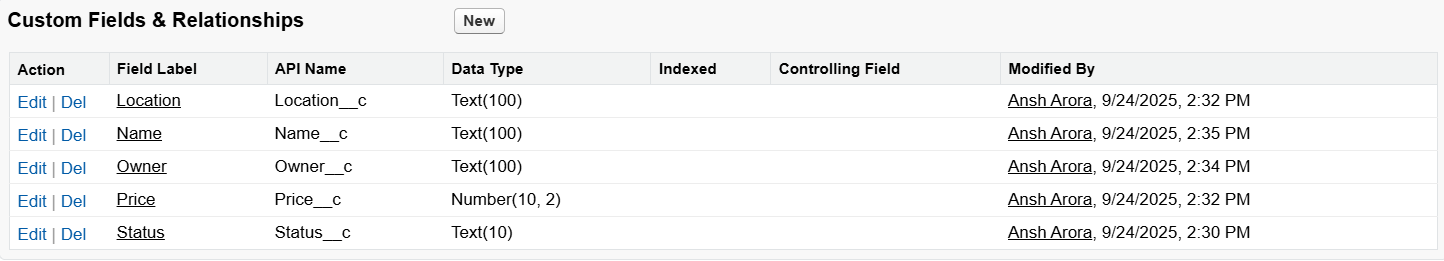
### Use HttpRequest and HttpResponse in Apex to make HTTP requests to external endpoints.

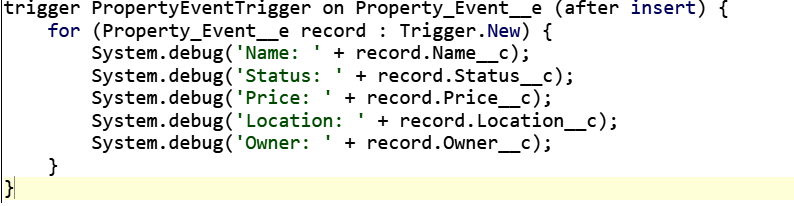
### Callouts can automate data updates, trigger workflows, and synchronize Salesforce with third-party systems in real time.

* **Platform Events**
* Platform Events enable **real-time communication** in Salesforce and external systems.
* They automate workflows, notify applications, and synchronize data without polling.
* Setup → Quick Find → **Platform Events → New Platform Event**



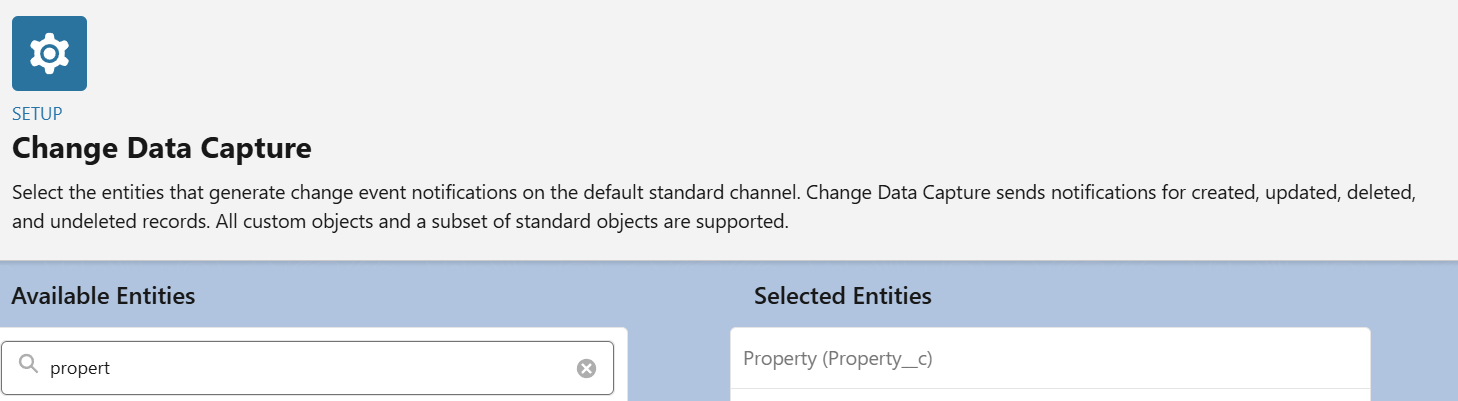
* **Add Custom Fields**
* Click **Fields & Relationships** → New
* Add fields like **Name, Status, Price, Location, Owner**





### Change Data Capture

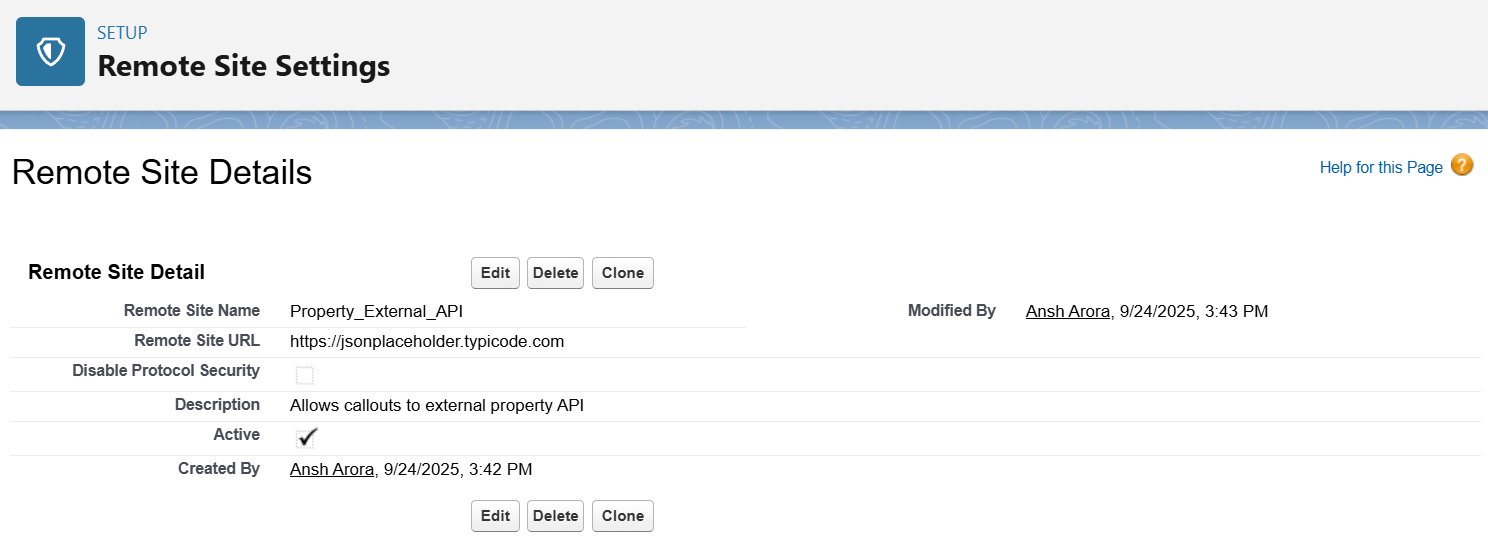
* Enable Change Data Capture (CDC) for the desired object:
* **Setup → Quick Find → Change Data Capture → Property\_\_c → Save**.
* CDC tracks **create, update, delete, and undelete** events in real time and can be subscribed to via **Apex triggers, Flows, or external systems**.
* It enables automated workflows, notifications, and data synchronization whenever records change.



* **Salesforce Connect**
* Salesforce Connect allows access to **external data** in real time without storing it in Salesforce.
* Create an **External Data Source** (e.g., OData 4.0), configure connection parameters, and sync **external objects**.
* External objects behave like standard objects and can be used in **SOQL, Reports, Flows, and Lightning Components**.
* Optional settings like **Server Driven Pagination** and **External Change Data Capture** improve performance and integration.

### OAuth & Authentication Implementation

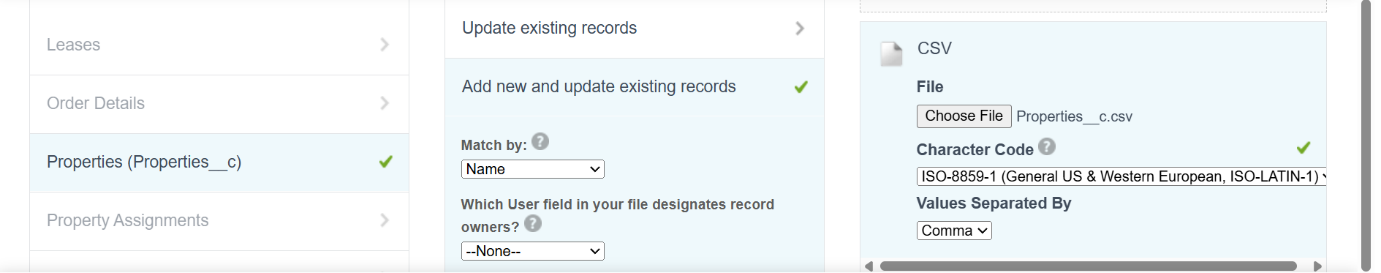
* Create a **Connected App** or **External Client App** with **API Name:**
* **Property\_External\_Client**, enable **OAuth**, provide a **Callback URL**, and select scopes like full, api, and refresh\_token, offline\_access.
* **Remote Site Settings**
* Create a **Remote Site** in Salesforce (Setup → Remote Site Settings → New) with a unique **Name** and the external **URL** (e.g., <https://jsonplaceholder.typicode.com>).
* Check **Active** and save to allow **Apex callouts** or integrations to that external system.

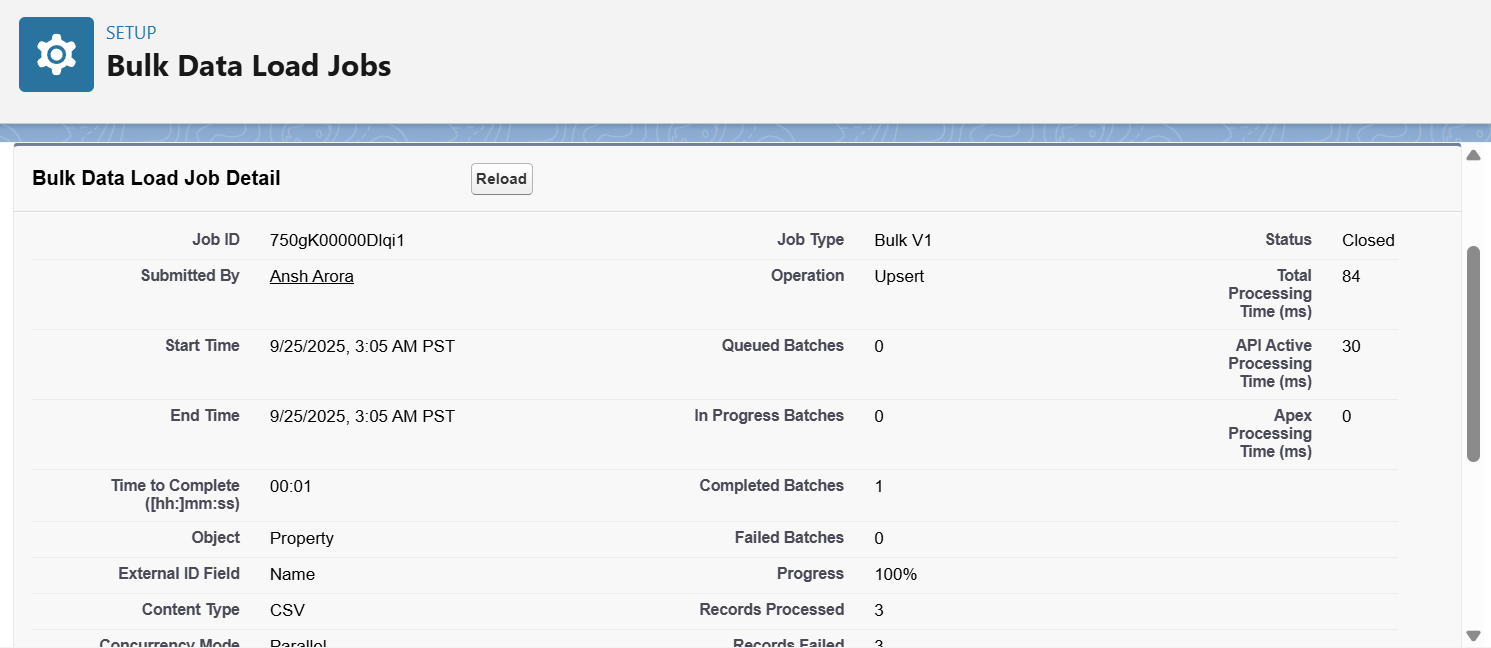


# Phase 8: Data Management & Deployment

### Data Import Wizard

* Go to **Setup → Data → Data Import Wizard** and select the object (Properties, Tenants, Leases, etc.).
* Choose to **add new records**, **update existing**, or **both**.
* Map a **unique field** to match existing records or leave **None** for new records.
* Map the **User field** for record owners or select **None** to assign to the importing user.
* **Upload CSV**, review field mapping, click **Start Import**, and verify imported records.





### Data Loader

* Used for **bulk data operations** when importing, updating, or deleting large volumes of Salesforce records.
* Ideal for objects like **Properties, Tenants, Leases**, where CSV files contain hundreds or thousands of records.

**Implementation**

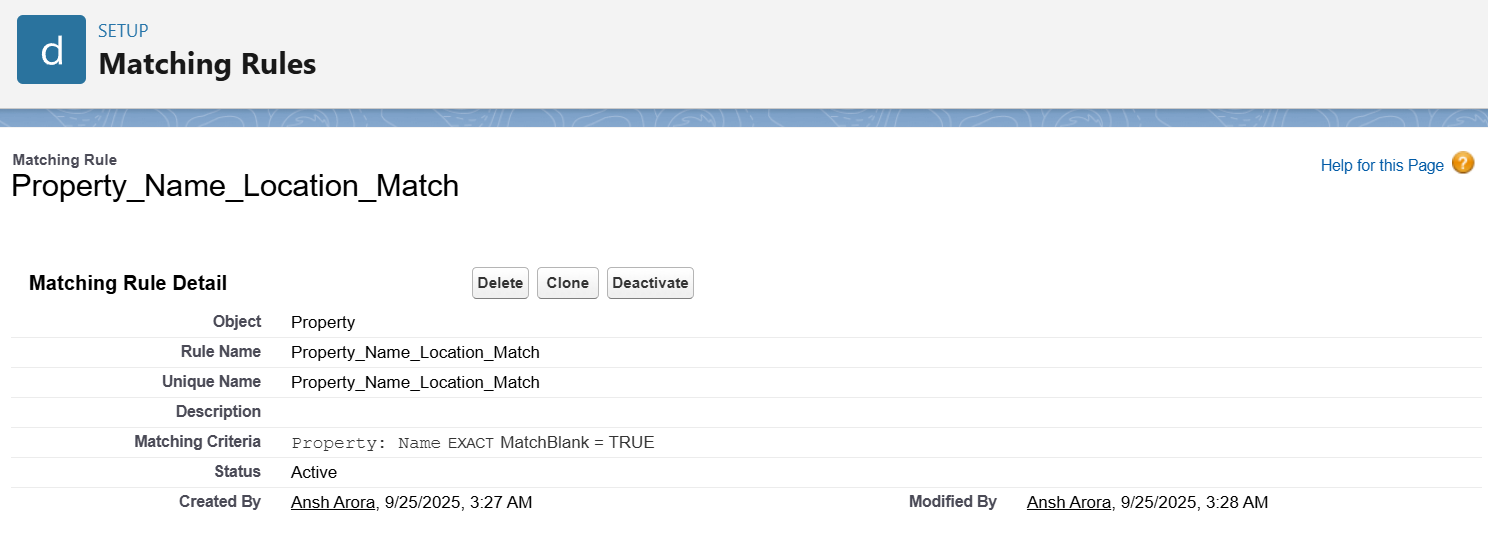
* Open **Data Loader** and log in with Salesforce credentials.
* Select **Insert** operation and choose the object **Properties\_\_c**.
* Upload the **Properties\_\_c.csv** file.
* Map CSV columns (Property\_ID, Name, Type, Status, Location, Price, Owner\_User\_Id) to Salesforce fields.
* Click **Next → Finish**, then verify the imported properties in Salesforce.

### Property Duplicate Rule

* Ensures that no **duplicate property records** (same Name and Location) are created in Salesforce.
* Helps maintain **clean and accurate data**, especially when importing records or manually adding properties.

**Implementation**

* Go to **Setup → Duplicate Management → Duplicate Rules → New Rule** and select **Property**.
* Enter **Rule Name** (Property\_Duplicate\_Check) and **Description**, set **Record-Level Security**.
* Set **Actions On Create/Edit** to **Alert** and enter alert text: “Use one of these records?”
* Choose a **Matching Rule** (e.g., Property\_Name\_Location\_Match) to identify duplicates.
* **Activate** the rule and test by creating or importing property records.

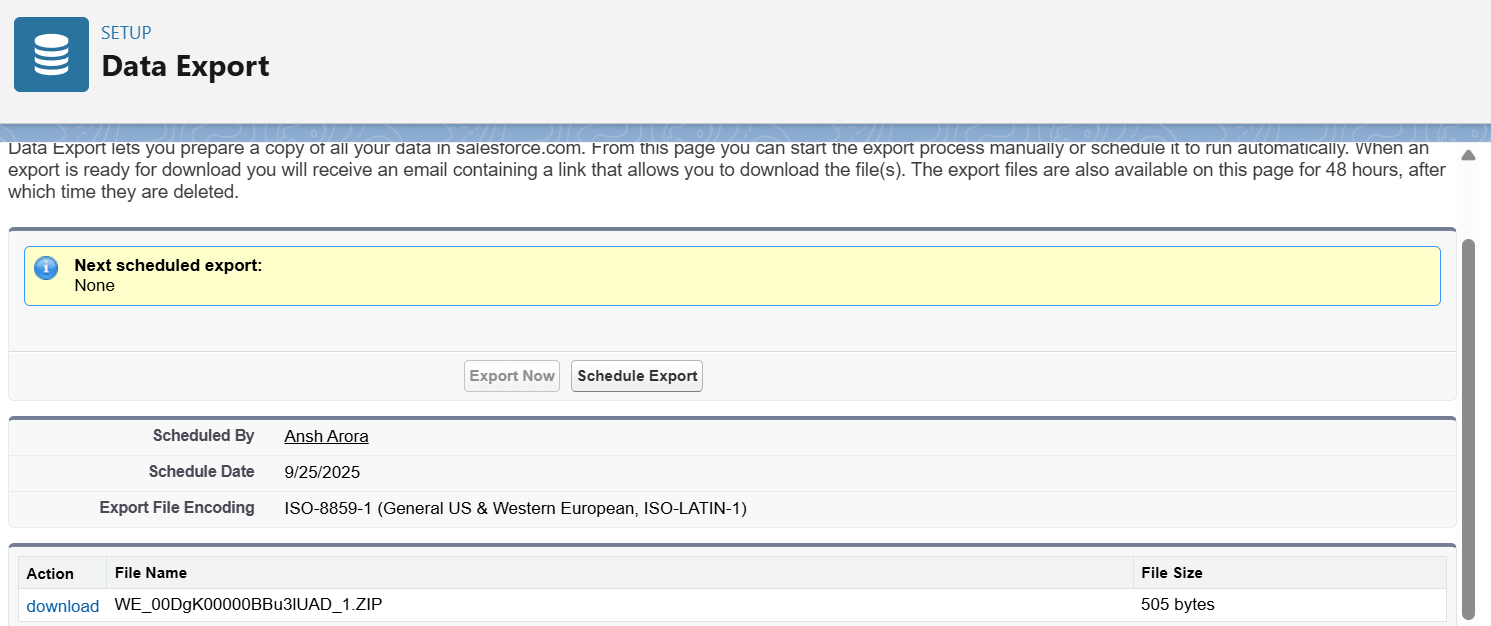




* Data Export & Backup
* To **back up all property records** from Salesforce for safety, auditing, or migration.
* Ensures all **Properties\_\_c data** is securely stored offline.

**Implementation**

* Go to **Setup → Data → Data Export**.
* Select the **Properties\_\_c** object for export and choose **CSV or ZIP** format.
* Choose **manual export** or schedule recurring exports.
* Click **Start Export** and wait for the process to complete.
* **Download the Properties\_\_c file** and store it securely for backup or migration.

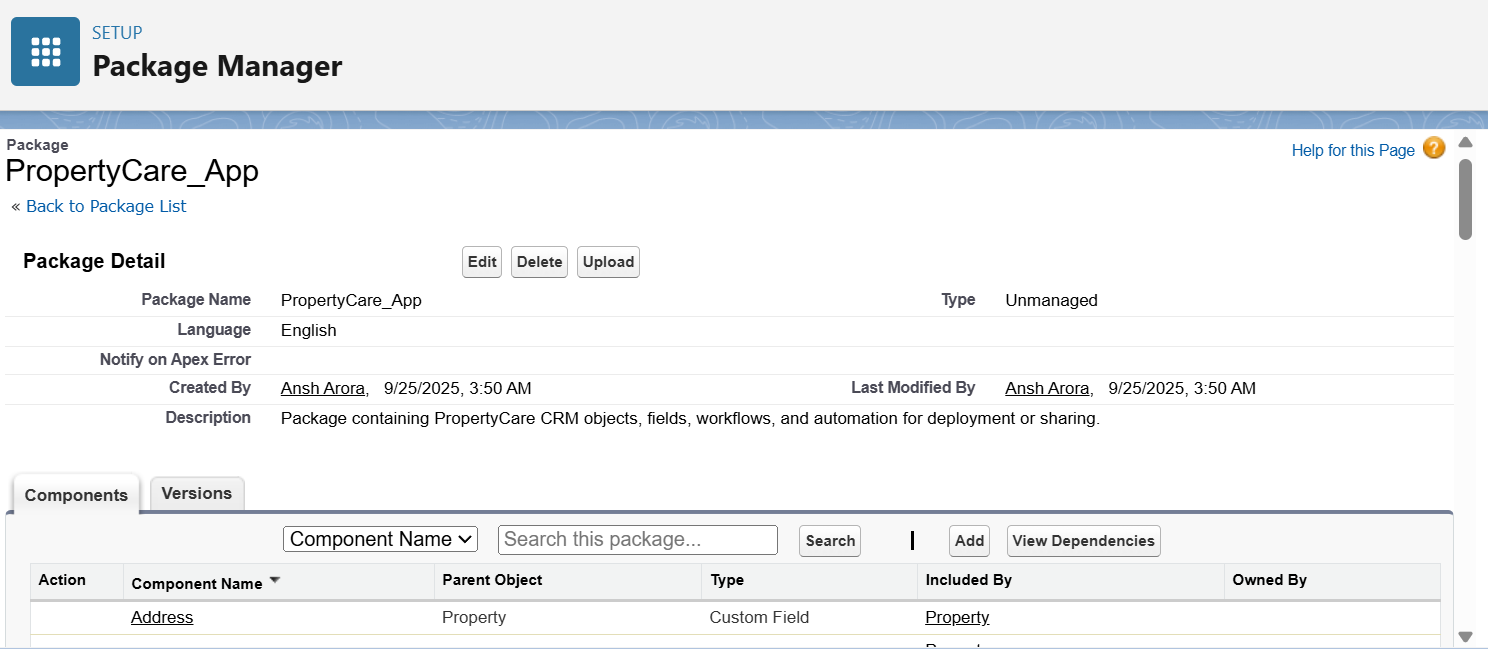


### Unmanaged vs Managed Packages

* Used to **bundle Salesforce components** for distribution or deployment across orgs.
* Helps developers share apps, templates, or configurations easily.

### Implementation

* Go to **Setup → Packages → New** and choose **Managed** or **Unmanaged** package.
* Enter **Package Name** and **Description**.
* Add **components** (Objects, Fields, Apex Classes, Flows, etc.) to the package.
* **Upload** the package to AppExchange (Managed) or share directly (Unmanaged).
* In the target org, **install** the package to deploy all included components.



### ANT Migration Tool – Use Case & Implementation

* The ANT Migration Tool is used for **deploying metadata** between Salesforce orgs via the command line, ideal for CI/CD or large migrations. Install the tool and configure build.properties with Salesforce credentials. Create a package.xml listing the metadata to retrieve or deploy. Use ant retrieve to fetch components and ant deploy to push them to the target org. Verify the deployment in Salesforce and check logs for errors.

### VS Code & SFDX

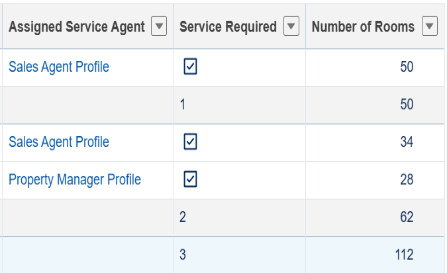
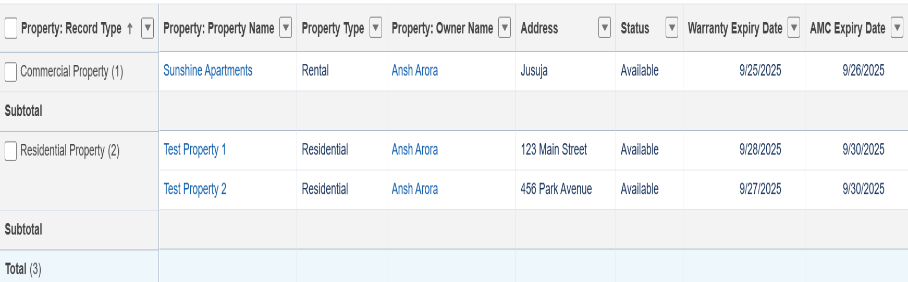
* Enables **source-driven Salesforce development**, allowing developers to manage metadata efficiently.
* Facilitates **creating scratch orgs, retrieving and deploying components**, and integrating with **version control/CI-CD pipelines**.
* Ideal for **large projects** where multiple developers work on objects, fields, flows, and Apex classes simultaneously.
* Streamlines testing and deployment while maintaining a **single source of truth** for metadata.

# Phase 9: Reporting, Dashboards & Security Review

* **Reports**
* Reports in Salesforce allow users to **analyze and visualize data** from objects like Properties and Tenants. They help track metrics, identify trends, and support **quick decision-making**.

### Implementation

* Go to **Reports → New Report** and select the **object** Properties.
* Add **filters** to include only relevant records (e.g., Status = Available).
* Select **columns and groupings** to display key fields and metrics.
* Click **Run**, save the report, and optionally add it to a **dashboard** for visualization.



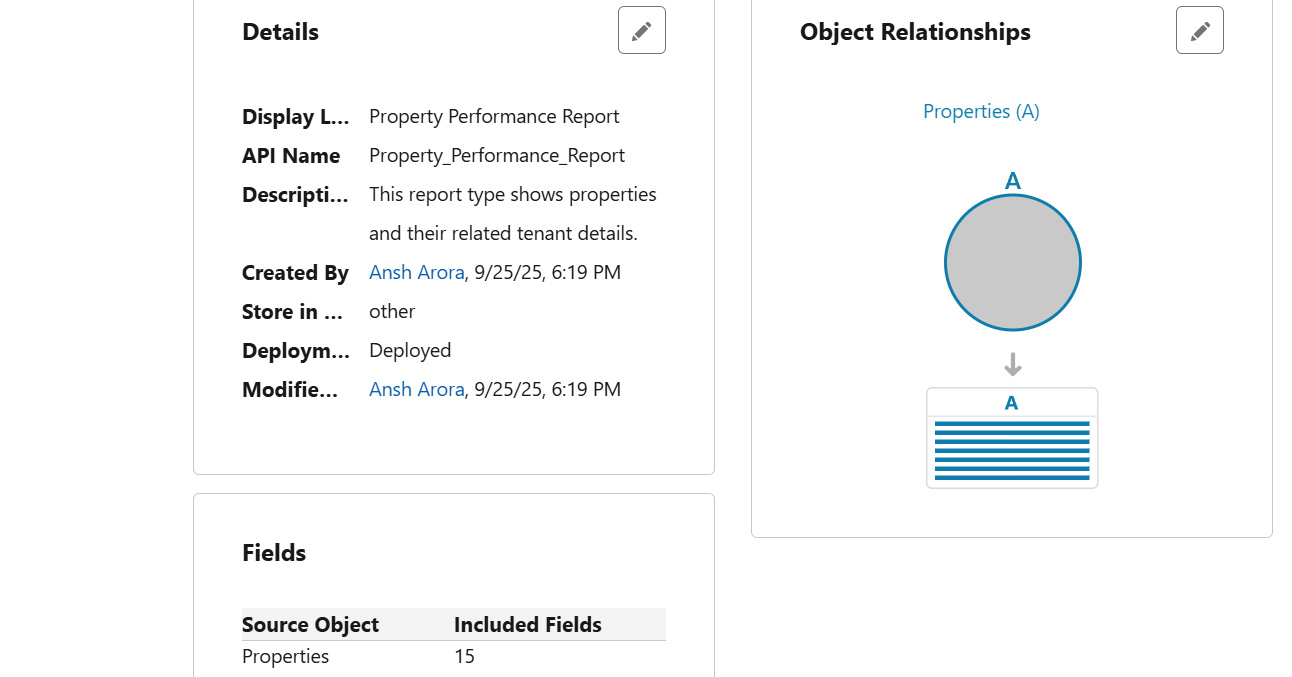
* **Report Types**
* The custom report type in PropertyCare helps users generate reports that combine property and owner details. It allows filtering and analyzing only the most relevant records, such as available properties with specific attributes.

 Navigate to **Setup > Report Types** and click **New Custom Report Type**.

 Select the **Primary Object** (e.g., Property\_\_c) and define related objects if needed.

 Provide a **Display Label, API Name, and Description** for clarity.

 Set the report type to **Deployed** so it is available for all users to build reports.



* **Dashboards**
* Property managers can use dashboards to monitor key metrics like the number of available properties, total rental income, and occupancy trends in real time, helping them make faster and data-driven decisions.

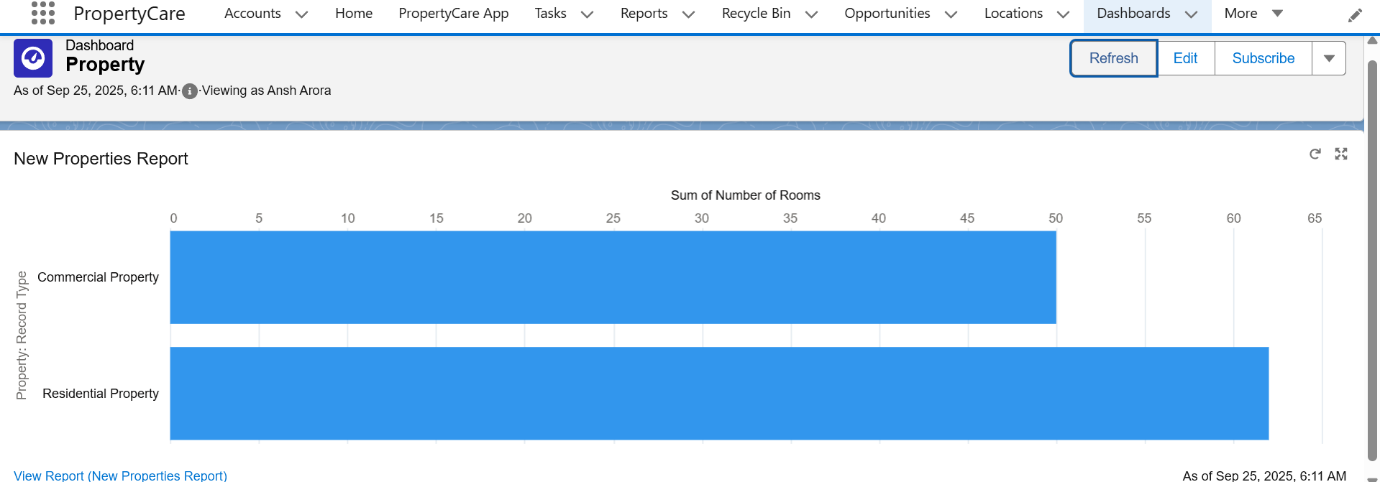
 Go to the **Dashboards tab** and click **New Dashboard**.

 Enter a **Dashboard Name, Description, and Folder** where it will be saved.

 Add **components** (charts, tables, metrics) by selecting the **source reports**.

 Customize chart types (bar, pie, line, gauge) to visualize property data.

 Save and **refresh the dashboard** to view up-to-date results.

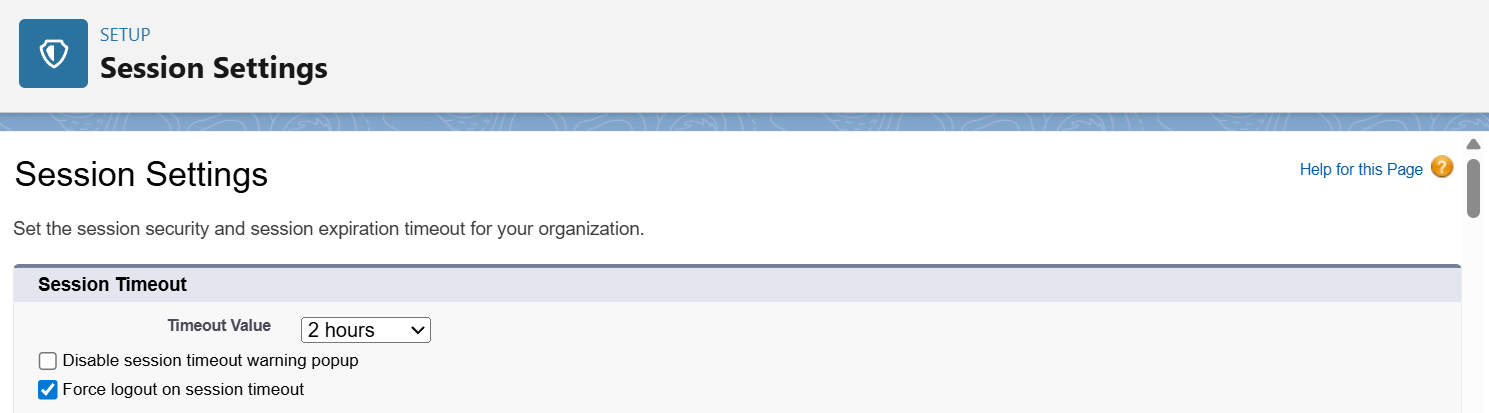


### Field Level Security

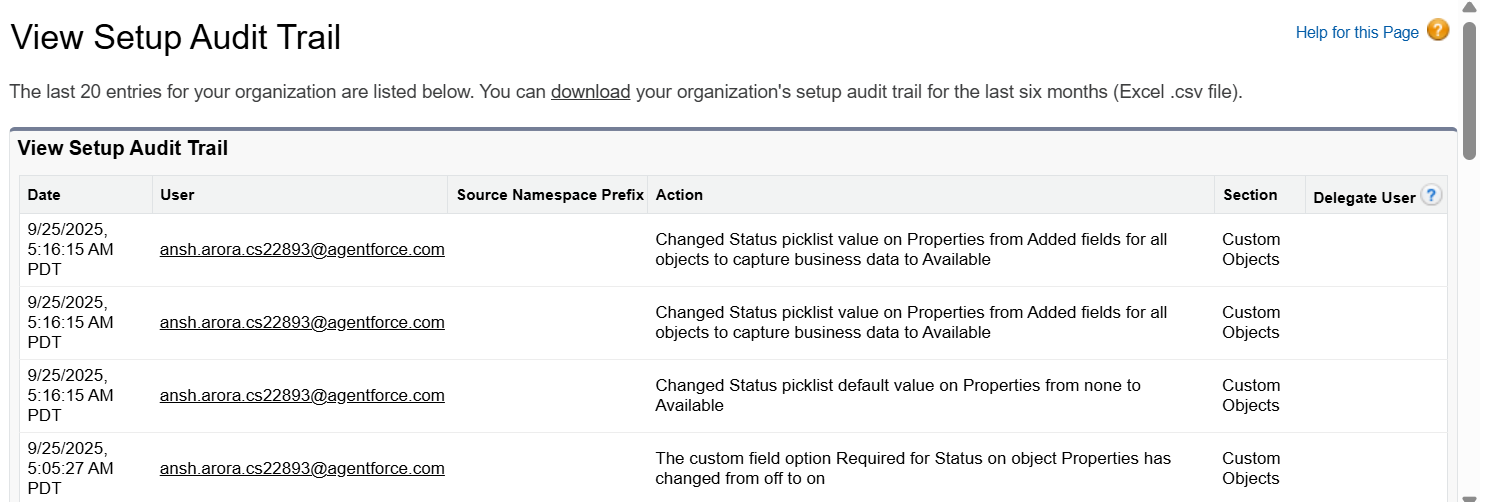
* Go to **Setup → Object Manager → [Select Object, e.g., Properties\_\_c] → Fields & Relationships**.
* Click on the field you want to control (e.g., Status, Number of Rooms) and select **Set Field-Level Security**.
* Choose which **profiles** can **Visible** or **Read-Only** access to this field.
* Save the settings to ensure sensitive information is **restricted for unauthorized users**.
* Verify by logging in as users with different profiles to confirm access permissions.
* **Session Settings**
* Session Settings help secure Salesforce by controlling how long users can stay logged in and when they are automatically logged out. This ensures **protection against unauthorized access** and reduces risk of inactive session misuse.

**Implementation**

* Go to **Setup → Security → Session Settings**.
* Configure **Session Timeout** to specify how long users can be inactive before being logged out.
* Enable **High Assurance Session Requirements** if needed for sensitive operations.
* Set **Force logout on session timeout** to enhance security.
* Save the settings and test by logging in to ensure sessions behave as configured.



* **Audit Trail**
* Audit Trail helps track all **setup and configuration changes** in Salesforce, showing who made changes and when. It ensures **accountability, security, and compliance** by allowing administrators to monitor and review modifications in the org.



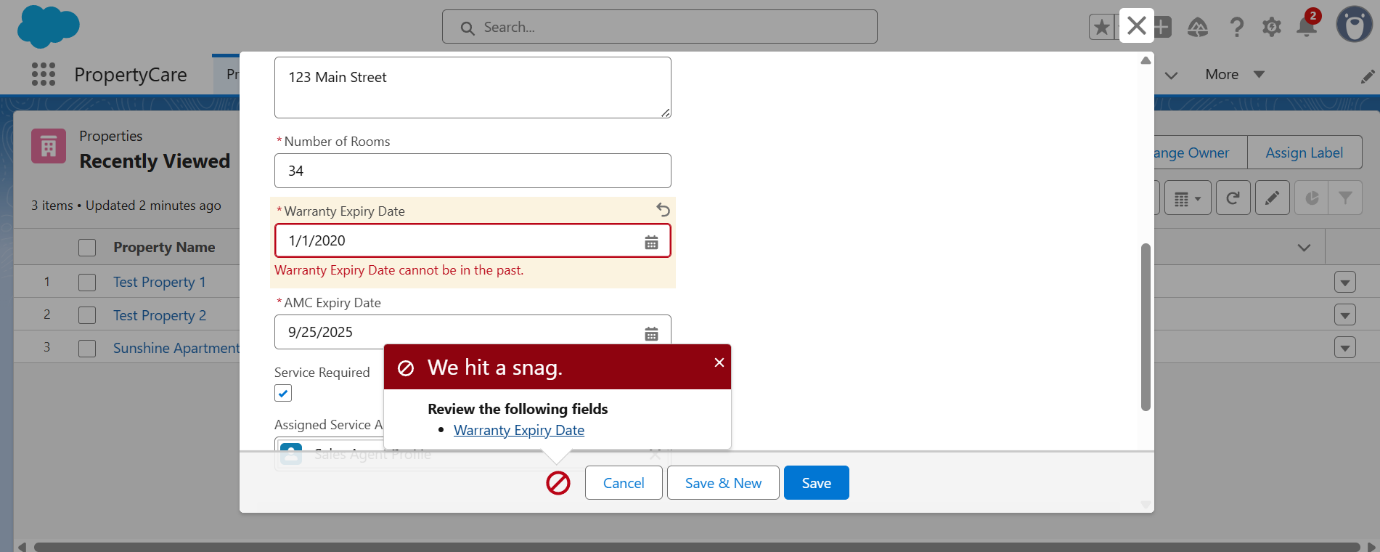
# ****Phase 10: Quality Assurance Testing****

This phase documents the Quality Assurance (QA) testing performed on the key features of the **PropertyCare CRM System**. Each test case includes the input details, expected output, and a confirmation of the actual output to verify that the implemented features are working as designed.

### ****Test Case 1: Warranty Expiry Date Validation Rule****

**Use Case / Scenario:** Testing the Validation Rule that prevents saving a Property record with a Warranty Expiry Date set in the past.  
**Test Steps (with input):**

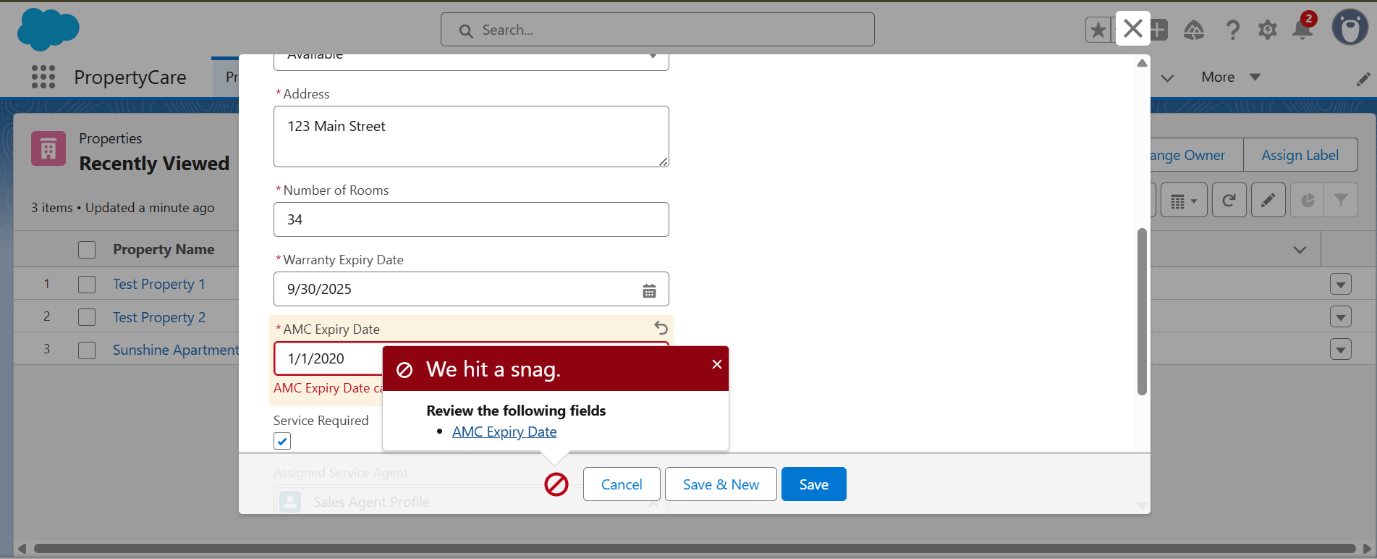
* Navigated to the **Properties** tab and clicked **New**.
* Entered Property Name = “Test Property 1”.
* Entered Warranty Expiry Date = 01-Jan-2020 (past date).
* Clicked **Save**.  
  **Expected Result:** Salesforce should block the record from being saved and display an error message: “Warranty Expiry Date cannot be in the past.”
* **Actual Result (with Screenshot):** The actual result matched the expected result. The system prevented the record creation and displayed the correct error message.



### ****Test Case 2: AMC Expiry Date Reminder Flow****

**Use Case / Scenario:** Testing the record-triggered flow that sends a reminder email when the AMC Expiry Date is within 30 days.  
**Test Steps (with input):**

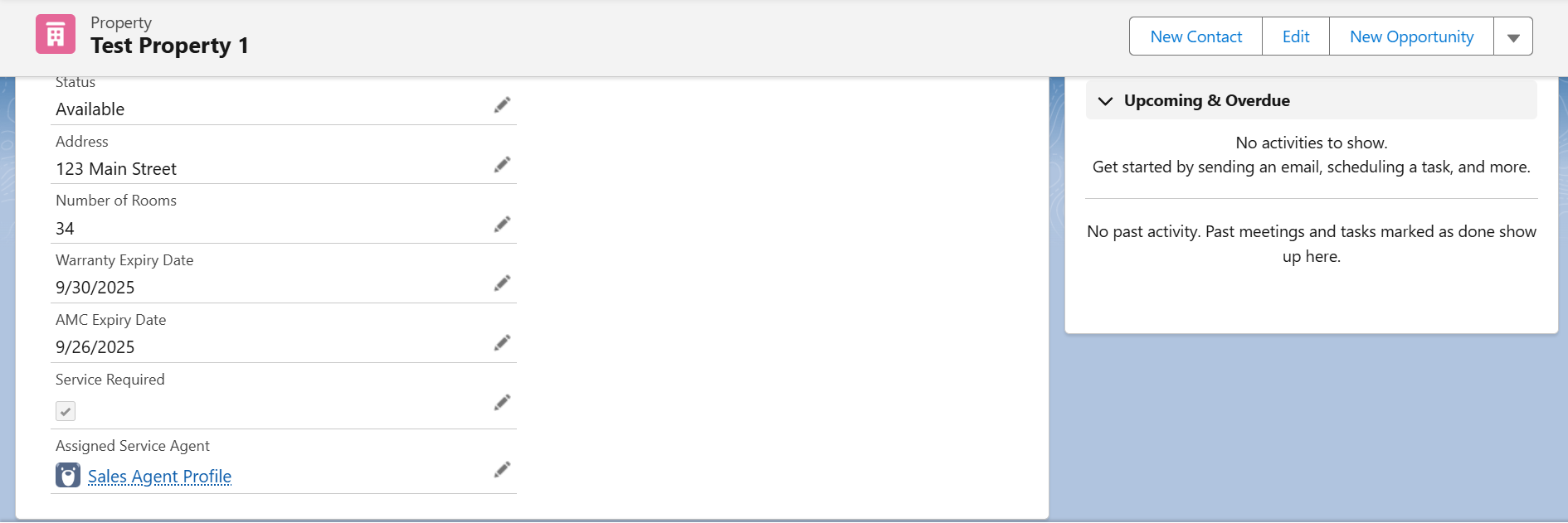
* Created a new Property with AMC Expiry Date = 15-Oct-2025.
* Flow condition checked for expiry dates within 30 days.
* Saved the record.  
  **Expected Result:** The system should automatically send an email alert to the Property Owner reminding them of the upcoming AMC expiry.  
  **Actual Result (with Screenshot):** The actual result matched the expected result. The email alert was successfully sent to the owner.



### ****Test Case 3: Service Required Checkbox Automation****

**Use Case / Scenario:** Testing the automation that creates a Service Task when the “Service Required” checkbox is checked.  
**Test Steps (with input):**

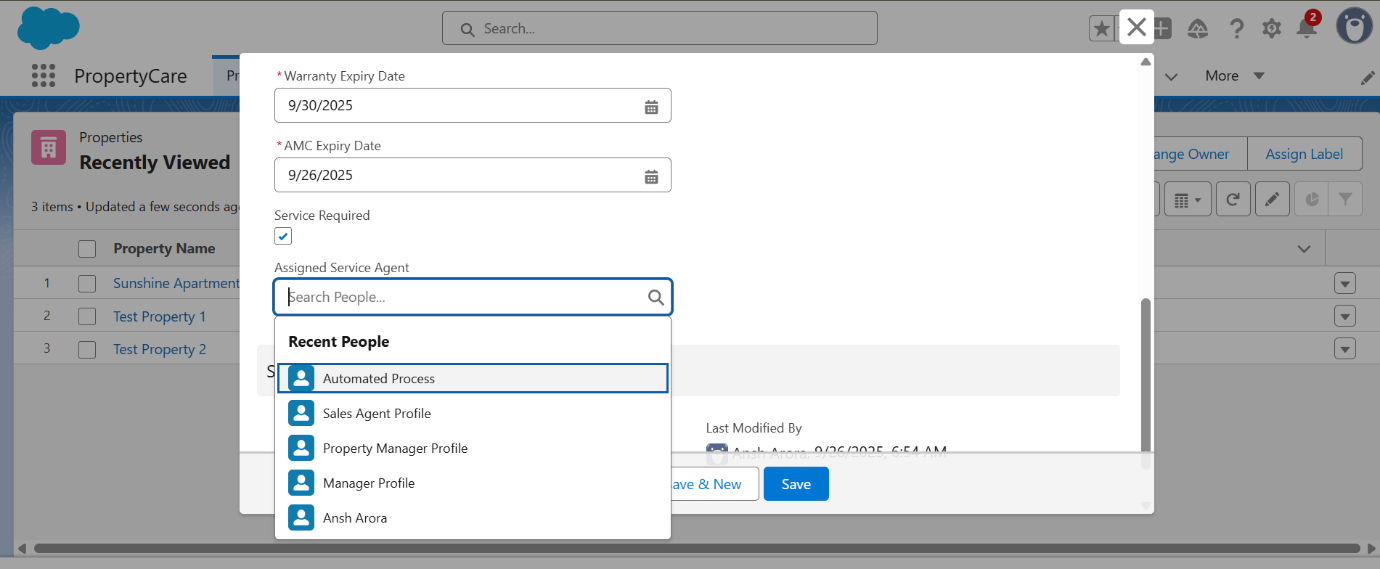
* Opened an existing Property record.
* Checked the **Service Required** checkbox.
* Clicked **Save**.  
  **Expected Result:** A new Task should automatically be created and assigned to the **Assigned Service Agent**.  
  **Actual Result (with Screenshot):** The actual result matched the expected result. A Service Task was created and linked to the Property record.

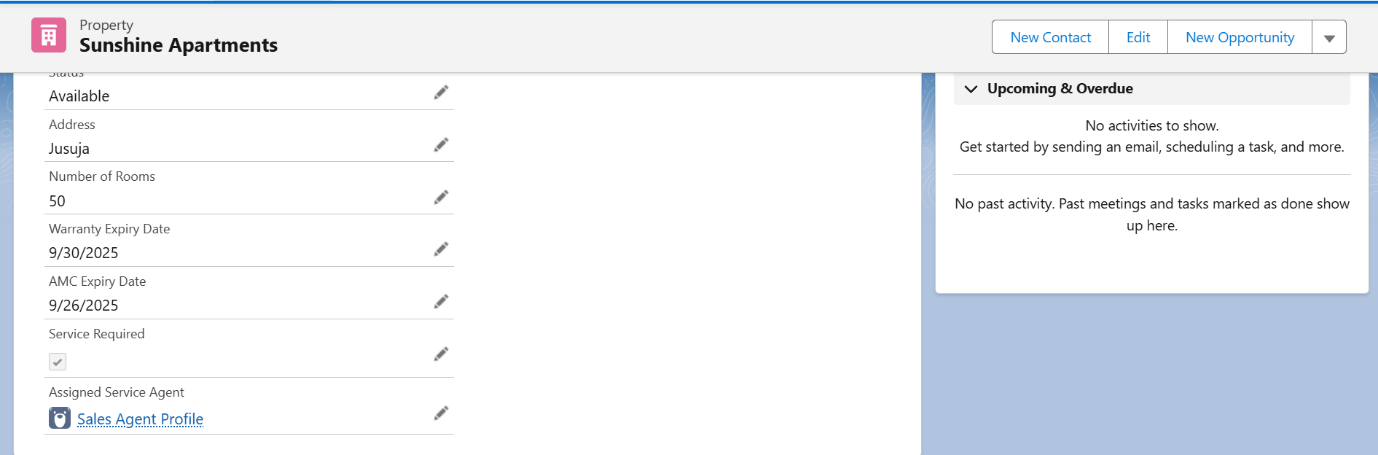


### ****Test Case 4: Assigned Service Agent Lookup****

**Use Case / Scenario:** Testing the lookup relationship to ensure a Service Agent can be properly assigned.  
**Test Steps (with input):**

* Navigated to Property record **“Sunshine Apartments”**.
* In the Assigned Service Agent field, selected a Sales Agent Profile.
* Clicked **Save**.  
  **Expected Result:** The Property record should now display the selected user as the Assigned Service Agent.  
  **Actual Result (with Screenshot):** The actual result matched the expected result. The lookup correctly saved and displayed the agent.





## ****Conclusion****

The **PropertyCare CRM Project** has been successfully tested and validated, ensuring that all implemented features function as designed. By leveraging Salesforce automation tools, validation rules, and reports, the system effectively centralizes property data and automates critical reminders.

The project’s core goals—tracking property warranties and AMCs, automating service requirements, and providing managerial insights—were fully achieved. Declarative features such as **Flows and Validation Rules** maintain data accuracy, while automation ensures timely notifications for property maintenance.

### ****Business Value Delivered****

* **Time Savings:** Eliminated manual tracking of warranties and AMCs.
* **Improved Accuracy:** Automated rules reduce human error.
* **Operational Efficiency:** Service tasks are automatically assigned.
* **Visibility:** Reports and dashboards give managers a real-time view of property status.

### ****Future Scope****

While the current system fulfills the primary requirements, the following enhancements are possible:

* **Tenant & Lease Management:** Introduce Tenant and Lease objects for complete rental lifecycle management.
* **Approval Processes:** Add approval workflows for high-value property maintenance requests.
* **Mobile App Integration:** Provide service agents with a mobile-first view for on-site updates.
* **Predictive Insights:** Use AI-powered analytics to predict property maintenance needs.

**Demo Video PropertyCare CRM Project:**

**DOC LINK:**

[**https://drive.google.com/file/d/1tn\_\_E7dnbtNRj8KnD3YEOh3olK5mg5Y9/view?usp=sharing**](https://drive.google.com/file/d/1tn__E7dnbtNRj8KnD3YEOh3olK5mg5Y9/view?usp=sharing)