# Phase 1: Requirement Gathering & Analysis

## Problem Statement

Finding **reliable and verified local service providers** such as plumbers, electricians, and cleaners is often a challenge.

Customers frequently face:

- Delays in service delivery
- Lack of trust and communication
- No proper tracking of service requests

On the other hand, **service providers** struggle with:

- Managing bookings and schedules
- Handling payments efficiently
- Maintaining smooth communication with customers

This disconnect results in **inefficiency, dissatisfaction**, and **missed opportunities** for both customers and service providers.

## Project Overview

The proposed solution is a Local Business Service CRM, designed to bridge the gap between customers and service providers.

This platform will streamline the process of booking, managing, and completing service requests, ensuring trust, transparency, and efficiency throughout.

### Key Features

- Online Service Requests: Customers can request services such as plumbing, electrical work, or cleaning.
- Job Notifications: Service providers receive instant notifications for new jobs and can accept or reject them.
- Automated Reminders: For scheduled services like AC maintenance every 6 months.
- Ratings & Feedback: Customers can share reviews to improve service quality.
- Admin Dashboard: The agency/company can track service requests, completed jobs, and revenue in real-time.

This system ensures timely and reliable services for customers and smooth operational management for service providers.

## Objectives

The main objectives of the project are to:

1. Improve Efficiency – Enable quick and hassle-free service booking.
2. Automate Tasks – Reduce manual effort with automated reminders, job assignments, and payment follow-ups.
3. Ensure Data Accuracy – Maintain verified service provider details and customer history.
4. Enhance User Experience – Provide trusted providers, easy booking, and faster response times.
5. Better Reporting – Deliver clear insights into requests, jobs, and revenues for data-driven decisions.

## Activities in Phase 1

### 1. Requirement Gathering

- Identify pain points of both customers and service providers.
- Collect functional and non-functional requirements.
- Define the scope and limitations of the project.

### 2. Stakeholder Analysis

| Stakeholder | Description |
|---|---|
| **Customers** | Service seekers who require reliable and timely assistance. |
| **Service Providers** | Professionals such as plumbers, electricians, and cleaners. |
| **Admin/Agency** | Entity managing operations, ensuring communication, and tracking performance. |

### 3. Business Process Mapping

Process Flow:

1. Customer submits a service request.
2. System assigns a service provider (who may accept or reject).
3. Job completion and customer confirmation.
4. Feedback collection and payment processing.

4. Industry-Specific Use Case Analysis

- AC Service Reminders – Automatic notifications every 6 months for maintenance.
- Urgent Plumber Booking – Book a plumber within 2 hours for emergencies.
- Cleaning Services – Schedule weekly or monthly cleaning with reminders.

- Electrician Services – Quick booking for urgent electrical repairs.

These use cases ensure the system supports both routine and emergency services.

5. AppExchange Exploration

- Explore existing Salesforce AppExchange solutions that enhance CRM functionality.
- Identify reusable components to save time and development cost.
- Evaluate integration options with:
  - Payment gateways
  - Notification systems
  - Reporting tools

### Conclusion

Phase 1 lays the foundation for the Local Business Service CRM by:

- Understanding the challenges of customers and providers
- Defining clear objectives
- Mapping out efficient business processes

Through use case analysis and AppExchange exploration, this phase sets the groundwork for building a trusted, efficient, and growth-oriented CRM platform that benefits all stakeholders.

---

# Phase 2: Org Setup & Configuration

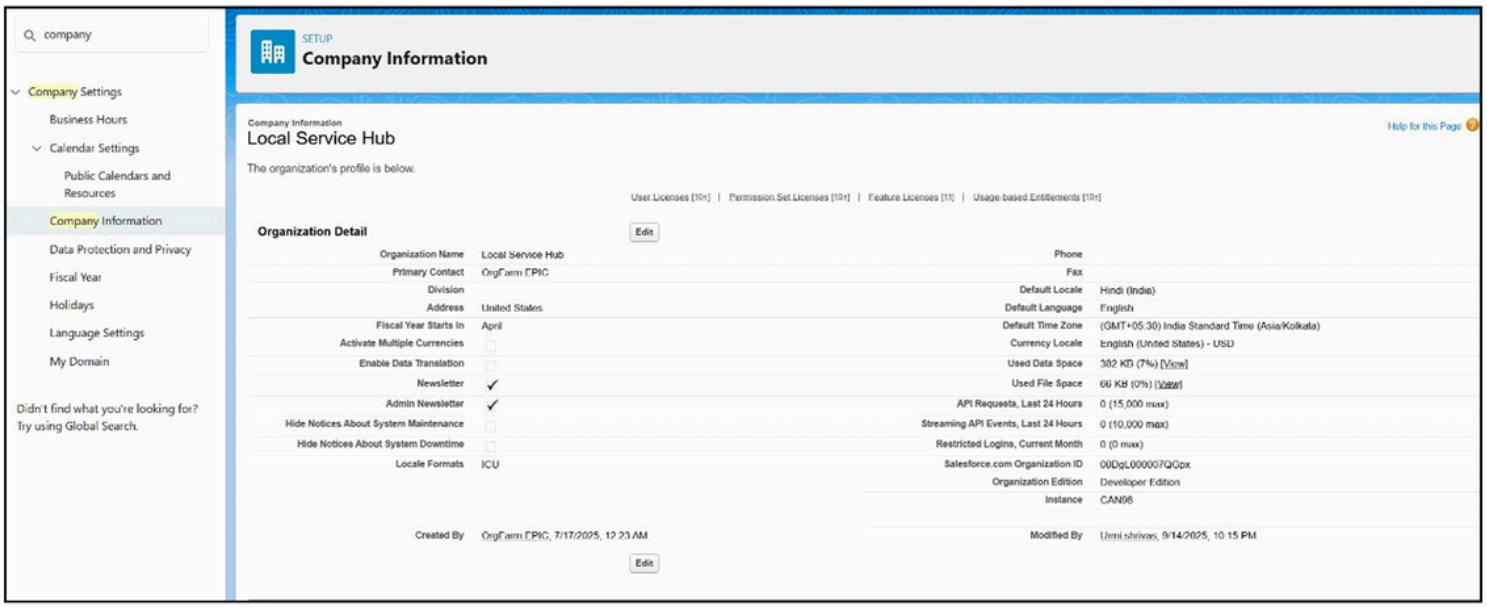
### 1. Salesforce Edition

- Edition Selected: *Developer Edition Org (Free)* – used to build the Local Business Service CRM.
- Sandbox Enabled: For testing and safe deployment of configuration changes.
- This setup ensures that development and testing can be performed without impacting live production data.

### 2. Company Profile Setup

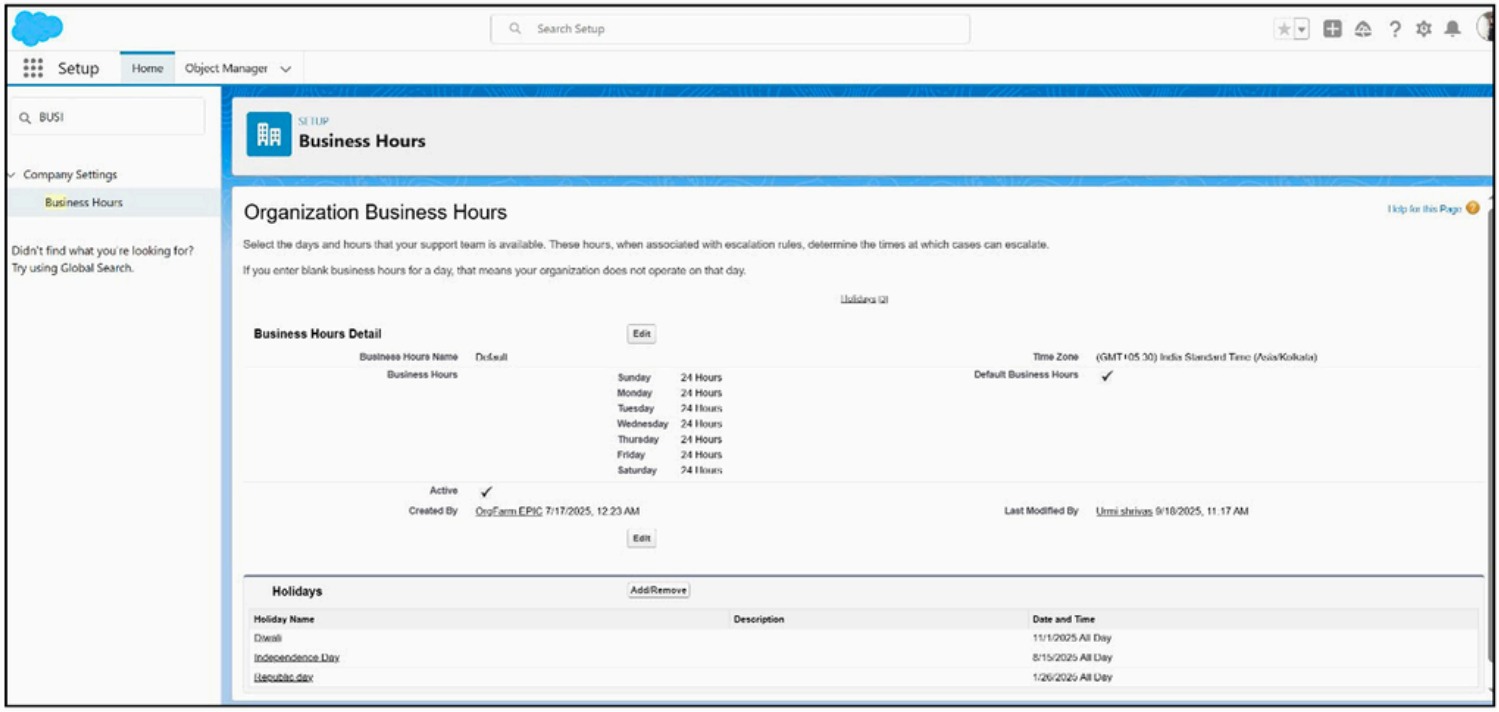
| Attribute | Configuration |
| --- | --- |
| **Organization Name** | Local Business Service CRM |
| **Time Zone** | Asia/Kolkata (IST) |
| **Locale** | English (India) |
| **Currency** | INR (₹) |

These settings define the organizational identity and ensure consistency across reports, records, and transactions.




## 3. Business Hours & Holidays

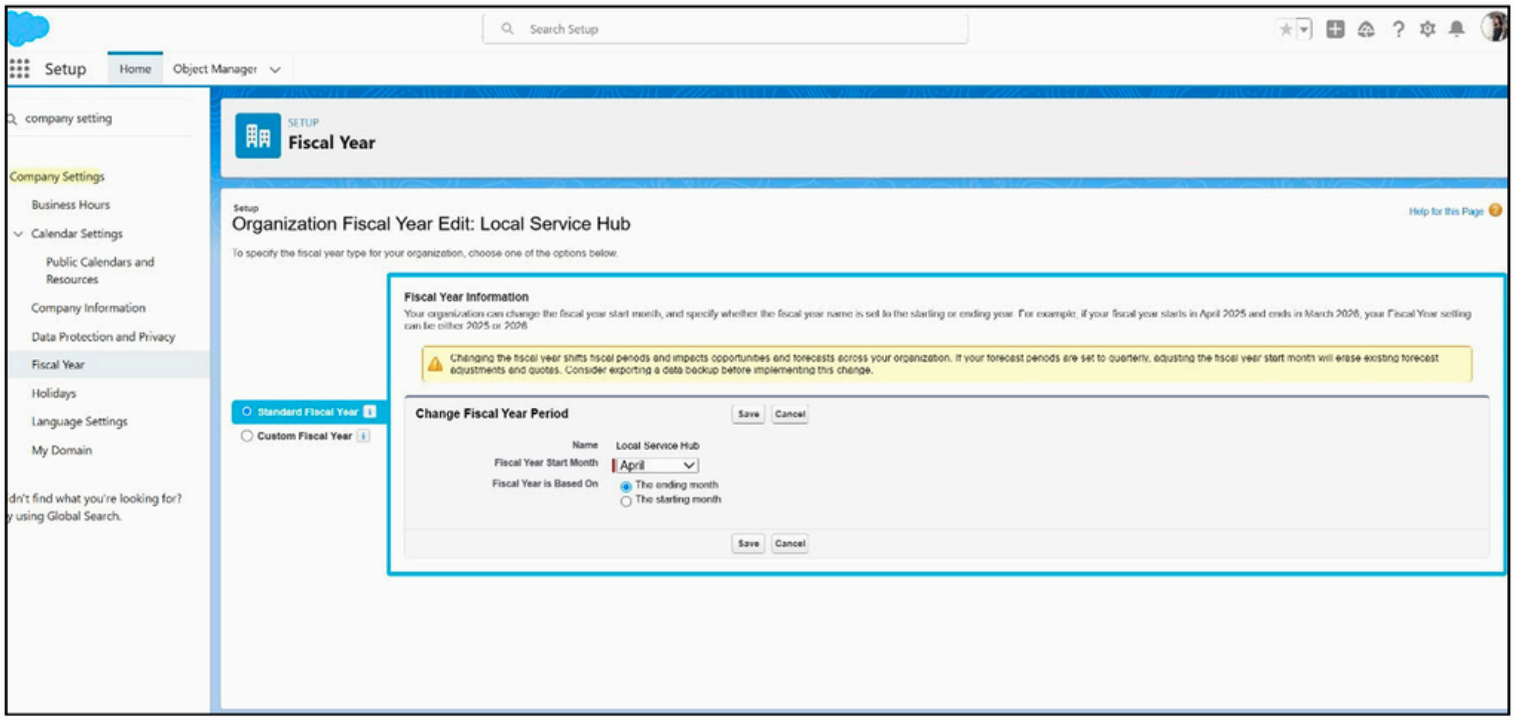
- **Working Days:** Monday to Friday
- **Working Hours:** 9:00 AM – 6:00 PM




This configuration ensures that support and service operations are executed within defined business hours, improving scheduling, response time, and customer satisfaction.

## 4. Fiscal Year Setting

- Fiscal Year: *Standard (April – March)*
- This aligns with the institutional accounting cycle in India, ensuring that financial and operational reports are consistent with the organization's fiscal policies.



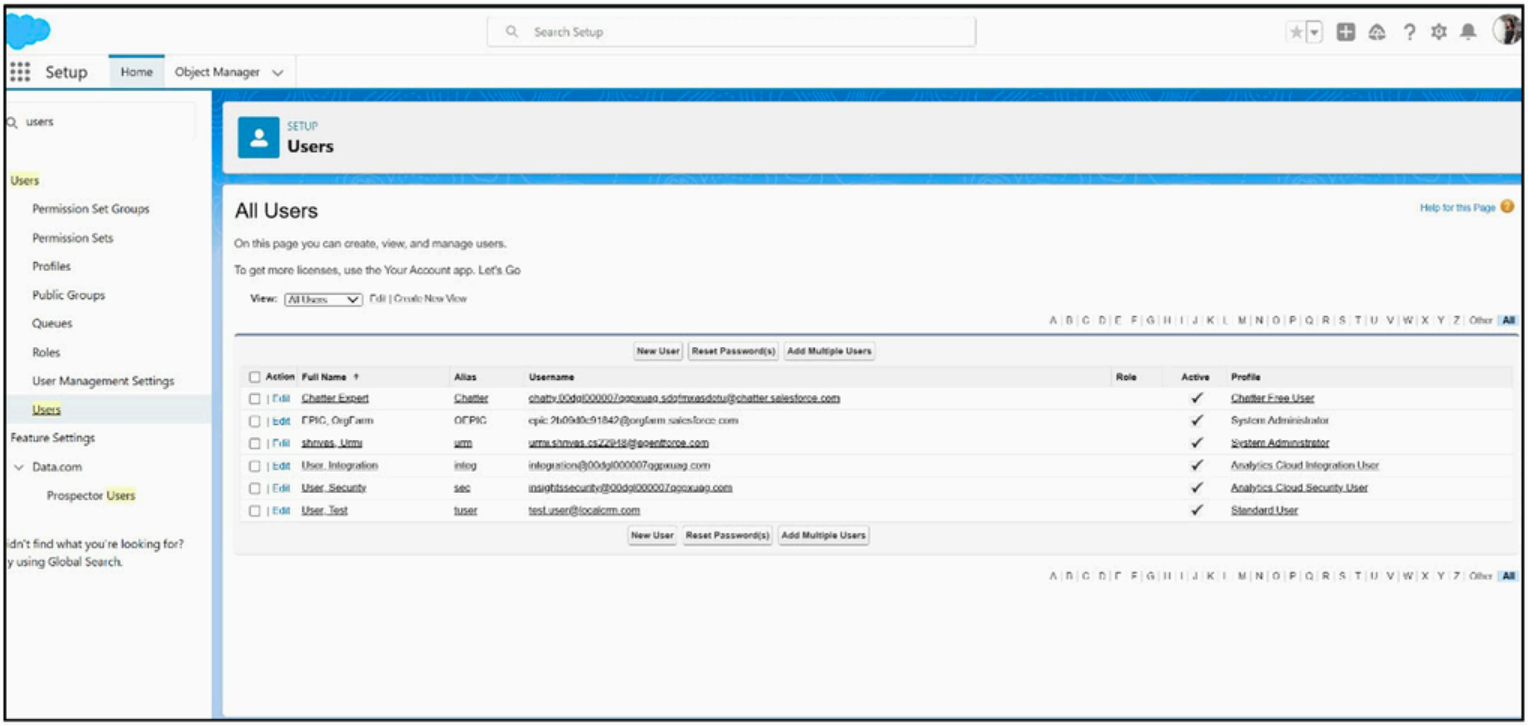
By following this, the CRM supports accurate revenue tracking and timely financial analysis.

## 5. User Setup & Licenses

- Users Created:
  - Admin
  - Customer
  - Service Provider

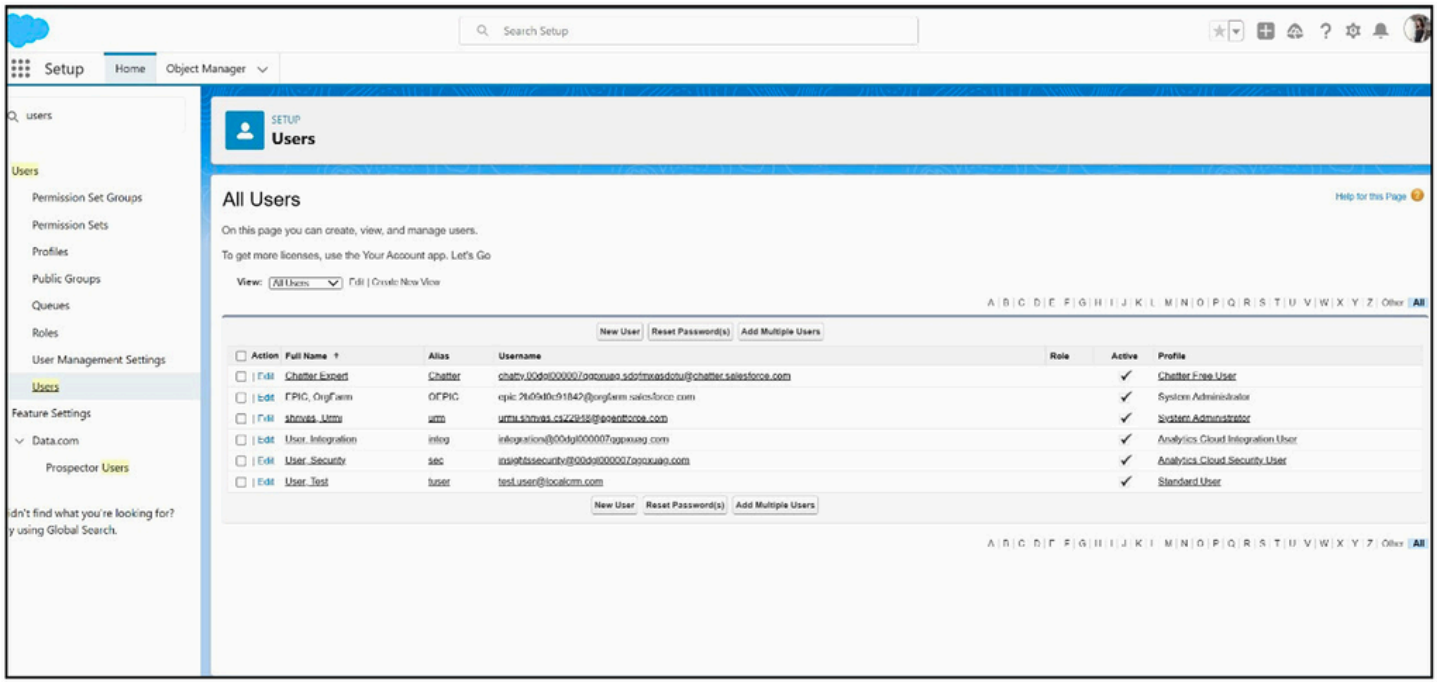
Each user was assigned a license and role-based access according to their functional requirements.
This ensures:

- Proper access rights
- Data security
- Smooth collaboration between all stakeholders




## 🔐 6. Profiles & Role Permissions

| Profile Name | Access Level & Responsibilities |
| --- | --- |
| **System Administrator** | Full access to all system settings, data, and customization features. |
| **Customer Manager** | Can manage customer records, handle service requests, and monitor service status. |
| **Provider Manager** | Can manage all service providers, assign jobs, and oversee their activities. |
| **Customer Agent** | Can handle and update only their **assigned customer requests**. |
| **Service Provider** | Can **view and update** the status of their **assigned jobs** only. |

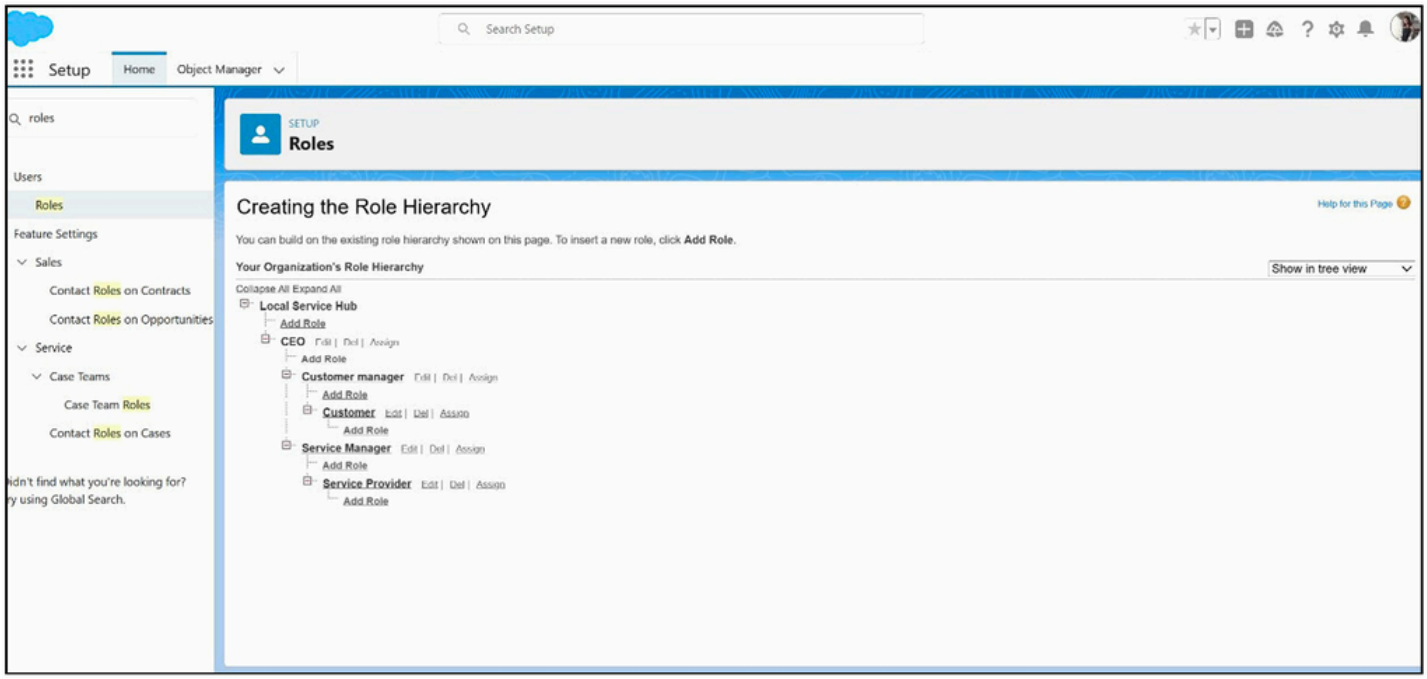



This role-based setup ensures data integrity, efficient workflow, and role-appropriate visibility across the system.

## 7. Roles Hierarchy

A role hierarchy is implemented to ensure proper data visibility and access control based on the organizational structure.

- Higher roles (e.g., CEO/Admin) can view and manage records of all subordinates.
- Mid-level roles (e.g., Managers) can manage their team's data.
- Lower roles (e.g., Agents/Service Providers) have access only to their relevant records.

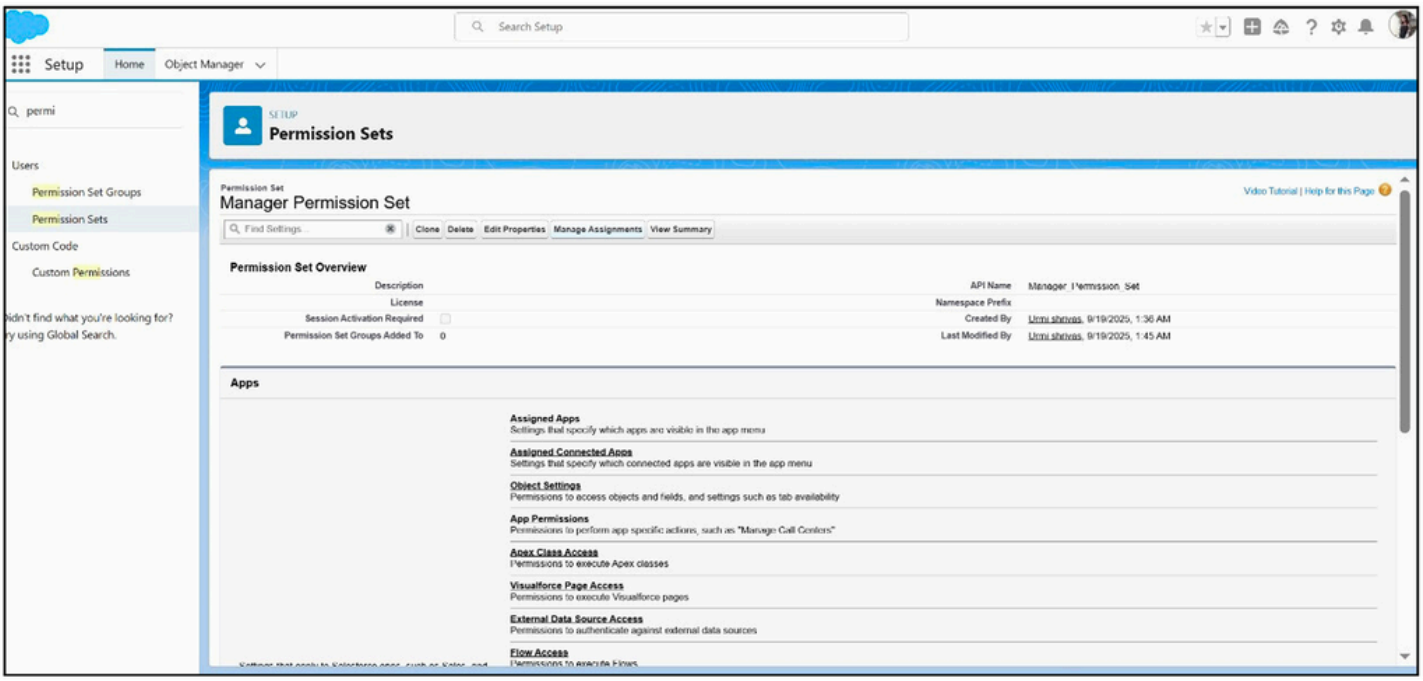



This ensures data security, accountability, and smooth flow of information across the organization.

## 🔑 8. Permission Sets

Permission Sets are created to provide additional access beyond what profiles offer. These are assigned based on user requirements.

| Permission Set Name | Purpose / Access Granted |
|---|---|
| **Extra Access** | Provides additional object or field-level permissions. |
| **Report Access** | Allows users to create, edit, and view reports. |
| **Dashboard Access** | Enables users to view dashboards for performance insights. |




🔹 *These permission sets are assigned dynamically as per user role and project needs.*

## 🧱 9. Organization-Wide Defaults (OWD)

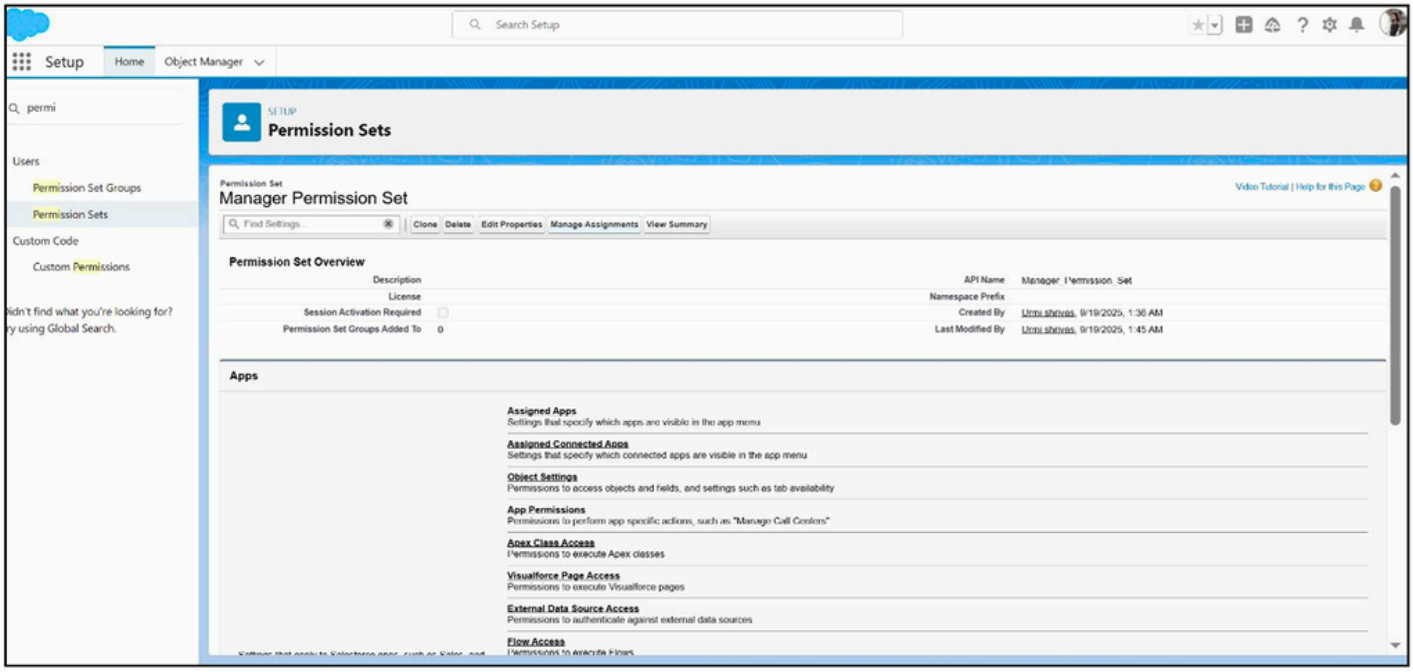
To ensure record-level security and controlled data sharing, appropriate OWD settings are defined.

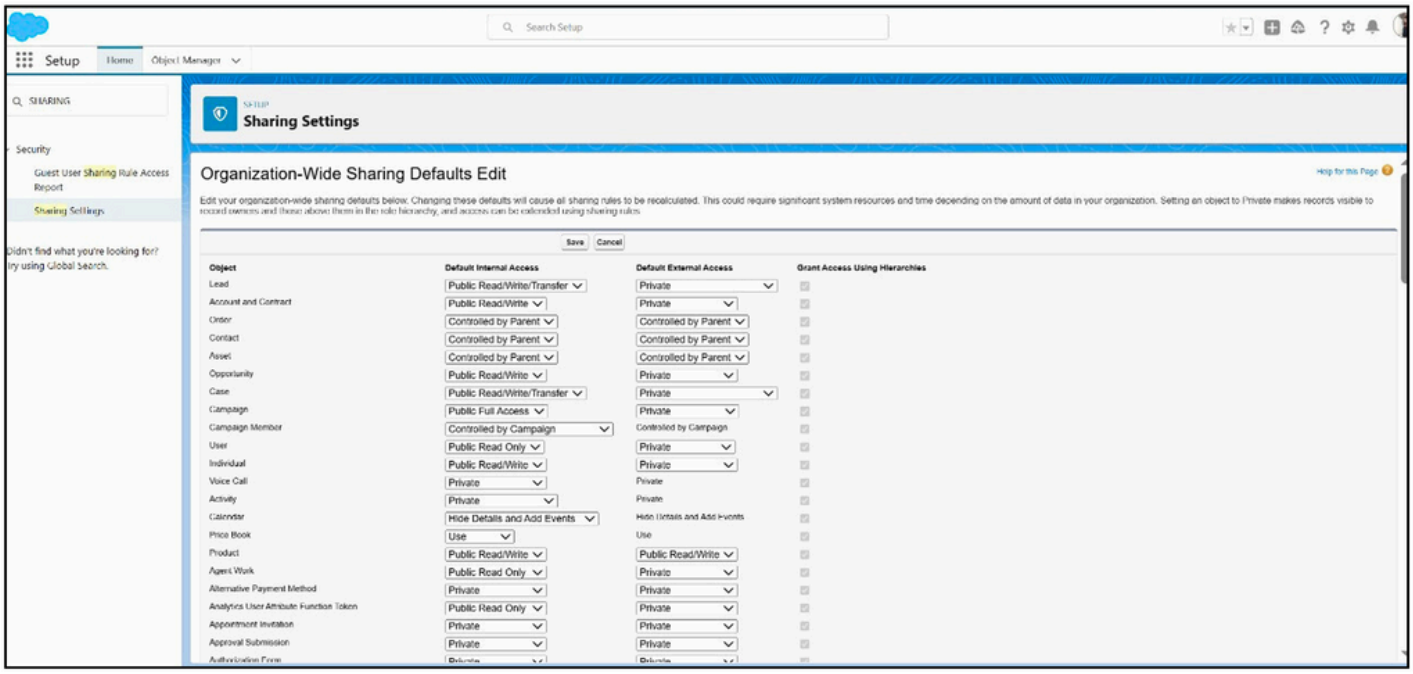
| Object | Default Access |
| --- | --- |
| **Customer** | Private |
| **Service Request** | Private |

This configuration ensures that records are visible only to the owner and authorized roles.

**Sharing Rules**

| Data Type | Shared With | Access Level |
|---|---|---|
| **Customer Records** | Customer Manager | Read/Write |
| **Service Requests** | Provider Manager | Read/Write |
| **Reports** | Admin Team | Read-Only |



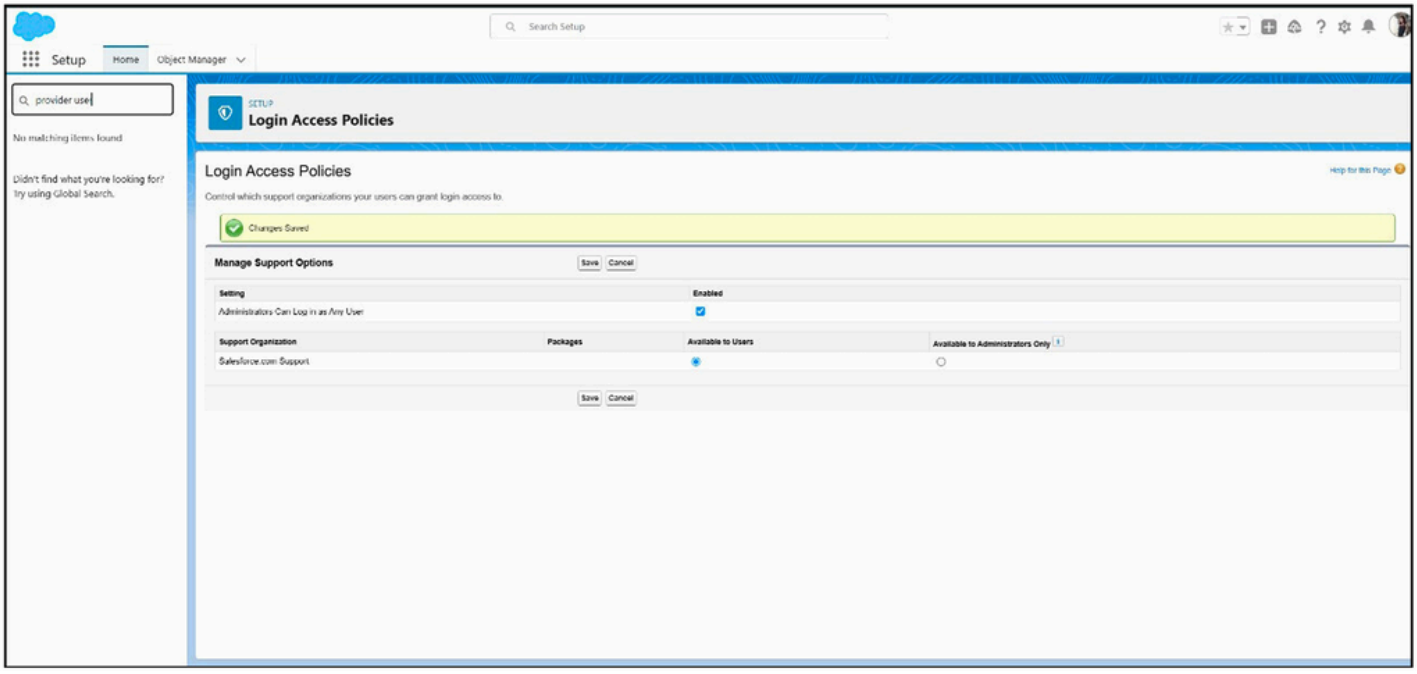



This structured sharing model balances security and collaboration efficiently.

## 10. Login Access Policies

To maintain secure and controlled access, the following login policies are applied:

- IP Restrictions: Only trusted IP ranges are allowed to access the org.
- Login Hours: Access limited to Monday–Friday, 9:00 AM – 6:00 PM.
- Admin Access: “Grant Login Access to Admin” enabled for troubleshooting and support.




These settings enhance **system security** and prevent unauthorized access.

## 11. Developer Org Setup

A Salesforce Developer Edition (Free) was created to build and customize the Local Business Service CRM.

Configuration Summary:

- Company Profile, Fiscal Year, Business Hours, and Holidays – Configured as per project needs.
- Users, Roles, and Profiles – Defined for each stakeholder (Admin, Customer, Provider).
- Security Settings – Implemented using OWD, Sharing Rules, and Permission Sets.
- Integration with VS Code:
  - Connected using Salesforce CLI for efficient development, version control, and deployment.

This ensures a robust, scalable, and developer-friendly environment.

### 12. Sandbox Usage

**A Sandbox environment was created for safe testing of customizations and configurations.**

- **Used for validating changes before deploying to the production org.**
- **Ensures bug-free and stable releases.**
- **Allows team collaboration during testing without affecting live data.**

### 13. Deployment Basics

Efficient deployment processes are set up to ensure smooth and secure transition of changes from Sandbox → Production.

| Deployment Method | Purpose |
|---|---|
| **Change Sets** | For migrating metadata between orgs. |
| **VS Code + Salesforce CLI** | For advanced deployments, version control, and automation. |

This process ensures safe, traceable, and consistent deployments across environments.

### Conclusion

This phase completes the core configuration and environment setup for the Local Business Service CRM.
By defining:

- Roles & Permissions
- Security Policies
- Sandbox & Deployment Strategy

...the system is now ready for customization, automation, and app development in the upcoming phases, ensuring security, scalability, and operational efficiency.

---

# Phase 3: Data Modeling and Relationship

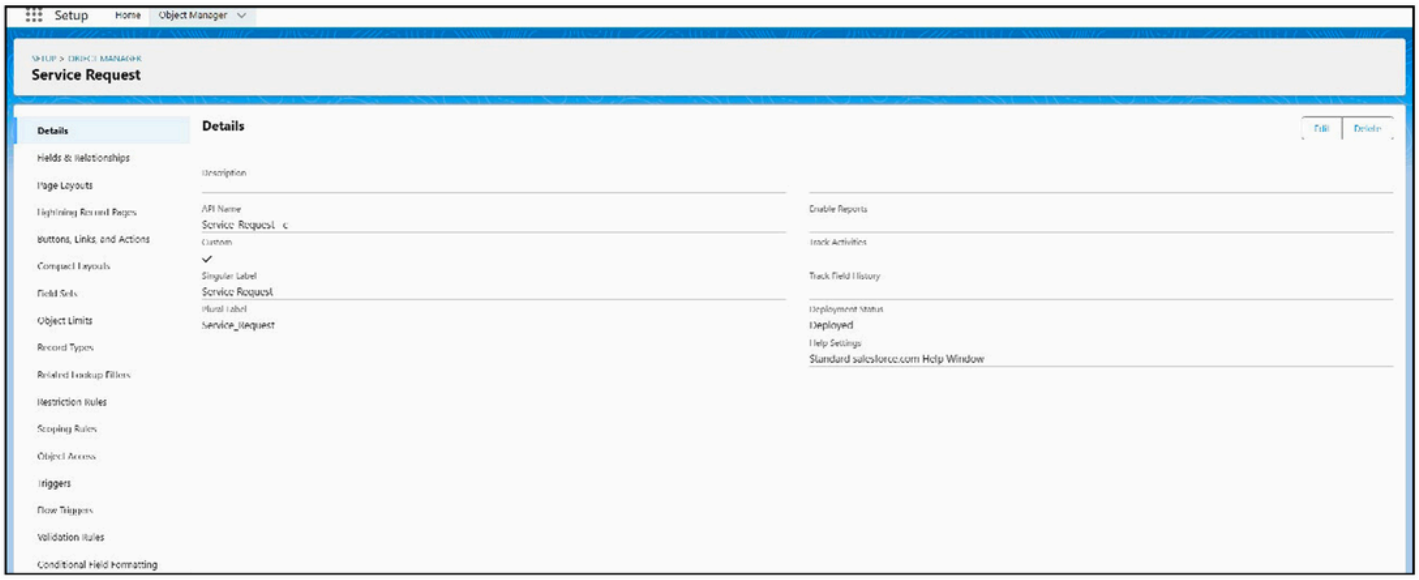
### 1. Standard and Custom Fields

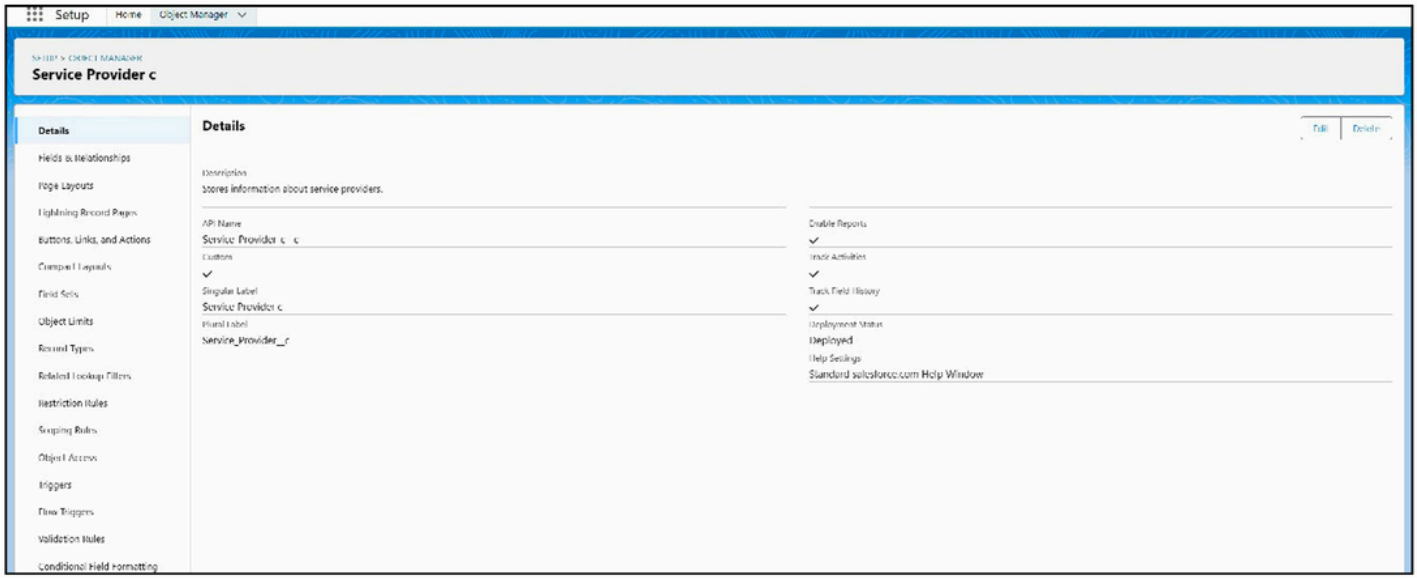
**Standard Objects Used:**

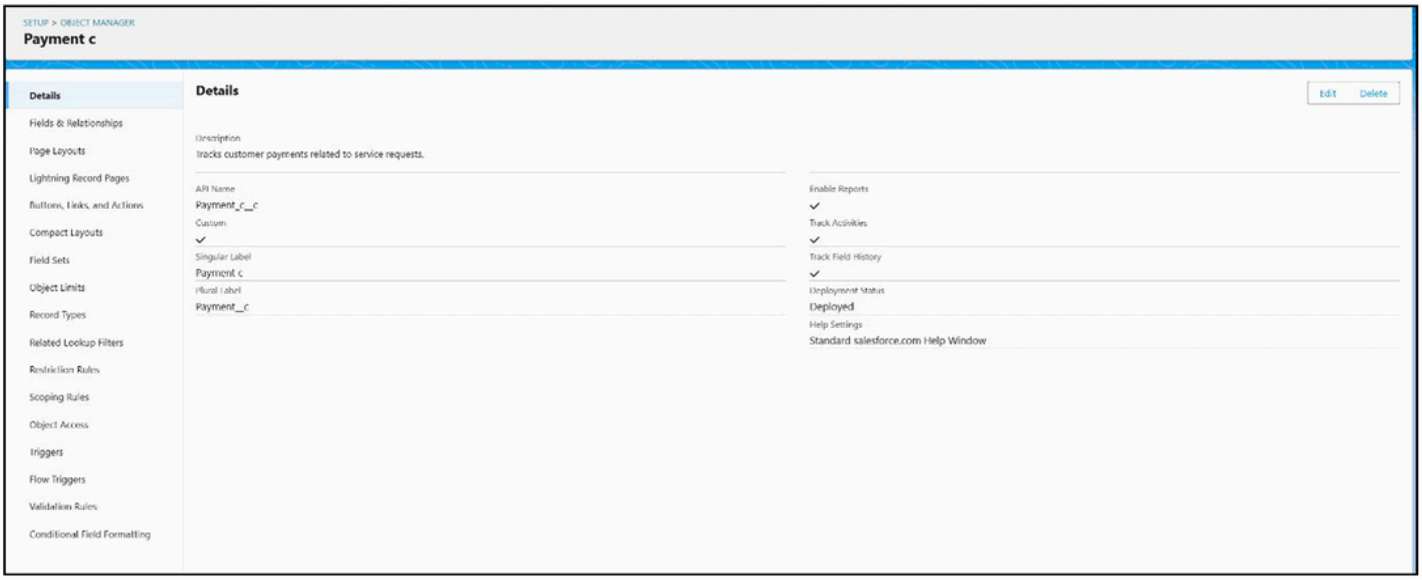
- **Account:** Stores information about companies or clients.
- **Contact:** Stores details of individuals related to accounts.
- **Opportunity:** Tracks potential sales or deals.
- **Case:** Manages customer service requests.

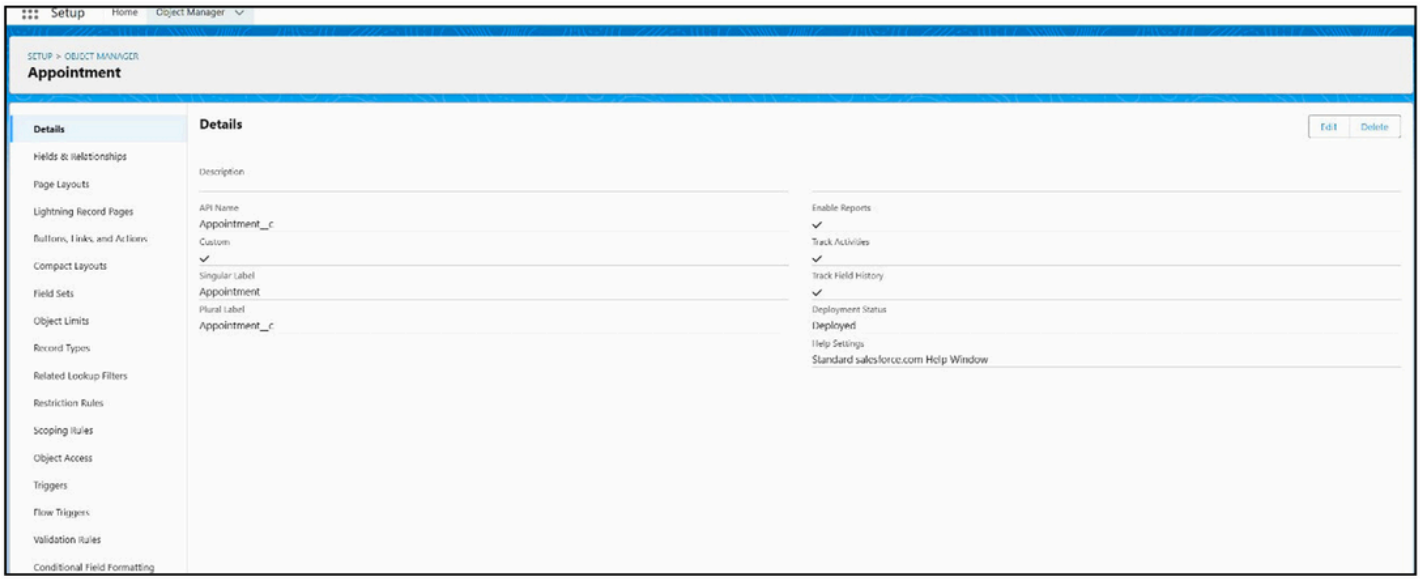
**Custom Objects Created:**

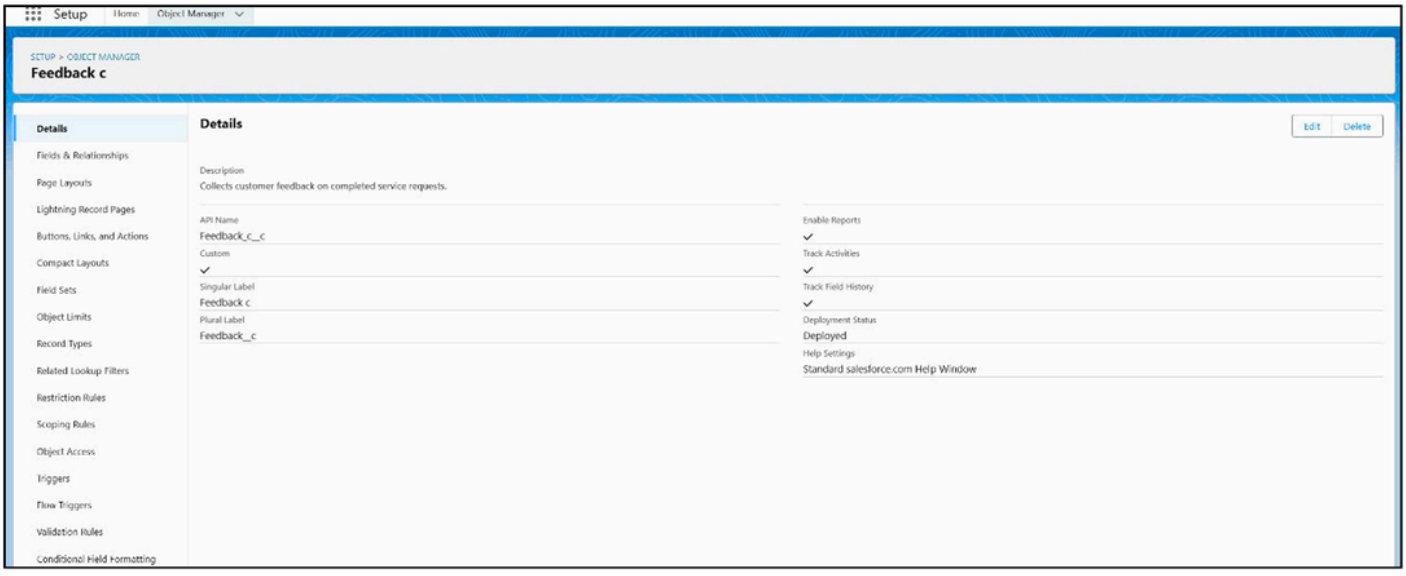
- **Service Request:** To manage service requests from customers.
- **Payment:** To track payment details.
- **Service Provider:** To store details of local service providers.
- **Appointment:** To schedule appointments between customers and service providers.
- **Feedback:** To collect customer feedback on services provided.
- **Customer Address:** To store addresses of customers for service delivery.

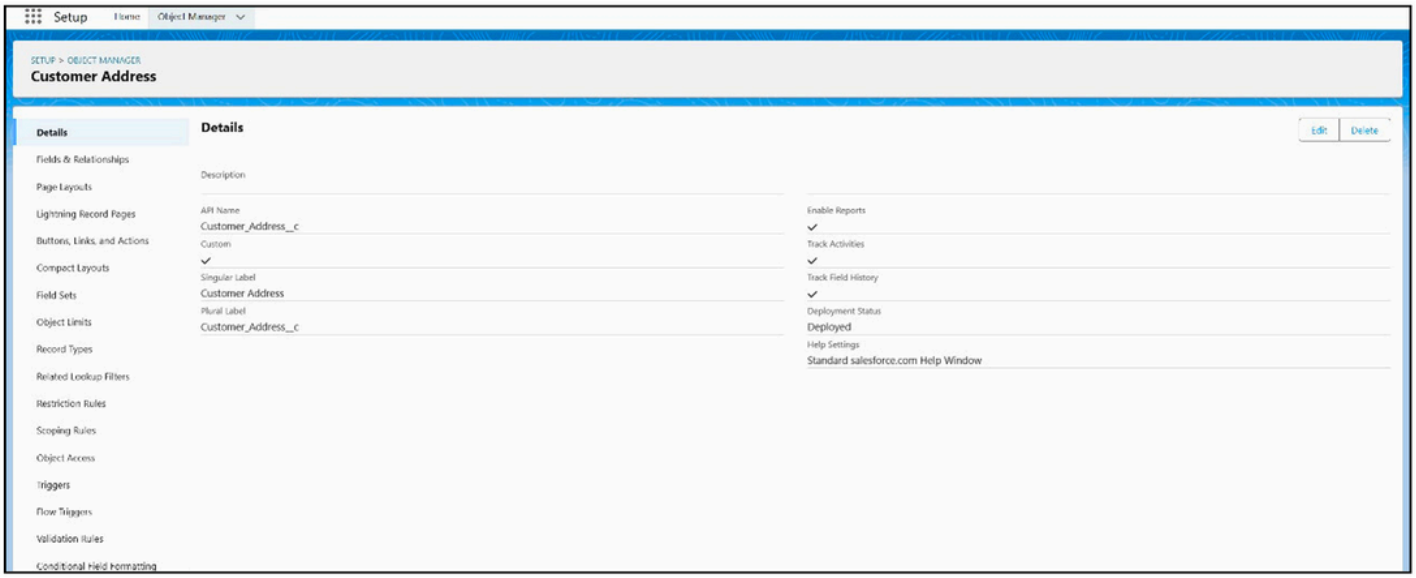



SETUP > OBJECT MANAGER
Payment c
Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Restriction Rules
Scoping Rules
Object Access
Triggers
Flow Triggers
Validation Rules
Conditional Field Formatting
Details
Edit
Delete
Description
Tracks customer payments related to service requests.
API Name
Payment_c__c
Custom
Singular Label
Payment c
Plural Label
Payment__c
Enable Reports
Track Activities
Track Field History
Deployment Status
Deployed
Help Settings
Standard salesforce.com Help Window


Setup
Home
Object Manager
SETUP > OBJECT MANAGER
Appointment
Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Restriction Rules
Scoping Rules
Object Access
Triggers
Flow Triggers
Validation Rules
Conditional Field Formatting
Details
Edit
Delete
Description
API Name
Appointment__c
Custom
Singular Label
Appointment
Plural Label
Appointment__c
Enable Reports
Track Activities
Track Field History
Deployment Status
Deployed
Help Settings
Standard salesforce.com Help Window


Setup
Home
Object Manager
SETUP > OBJECT MANAGER
Feedback c
Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Restriction Rules
Scoping Rules
Object Access
Triggers
Flow Triggers
Validation Rules
Conditional Field Formatting
Details
Edit
Delete
Description
Collects customer feedback on completed service requests.
API Name
Feedback_c__c
Custom
Singular Label
Feedback c
Plural Label
Feedback__c
Enable Reports
Track Activities
Track Field History
Deployment Status
Deployed
Help Settings
Standard salesforce.com Help Window




## 2. Fields

**Service Request:** Request ID, Customer Name, Service Type, Description, Status, Request Date

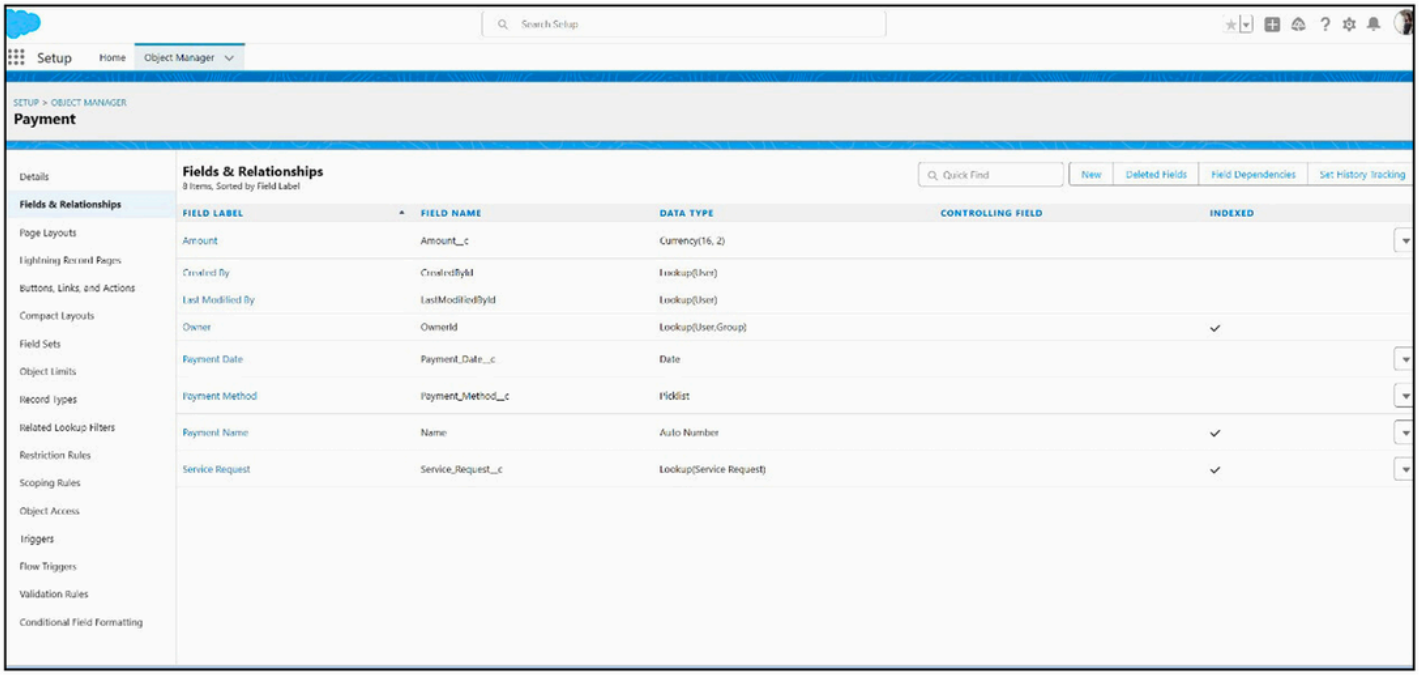
**Payment:** Payment ID, Service Request ID, Amount, Payment Method, Payment Date, Status

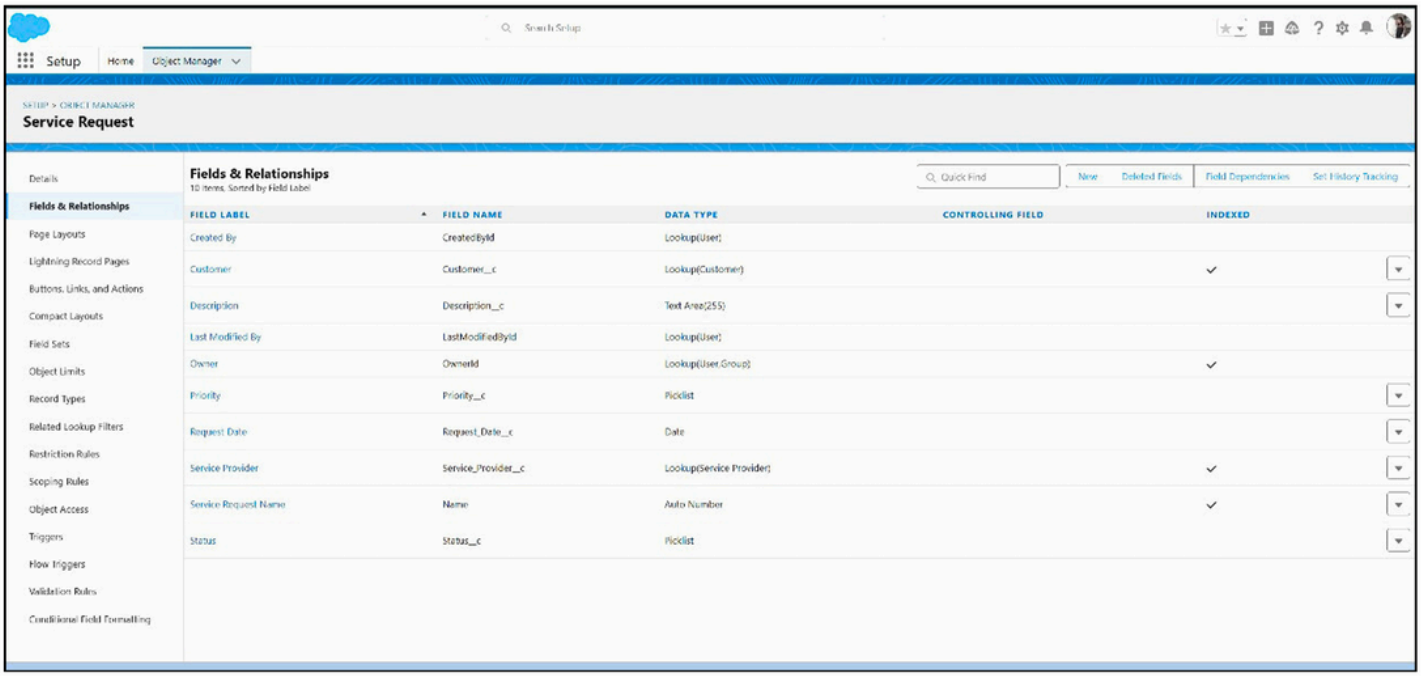
**Service Provider:** Provider ID, Name, Service Type, Contact Number, Email, Rating

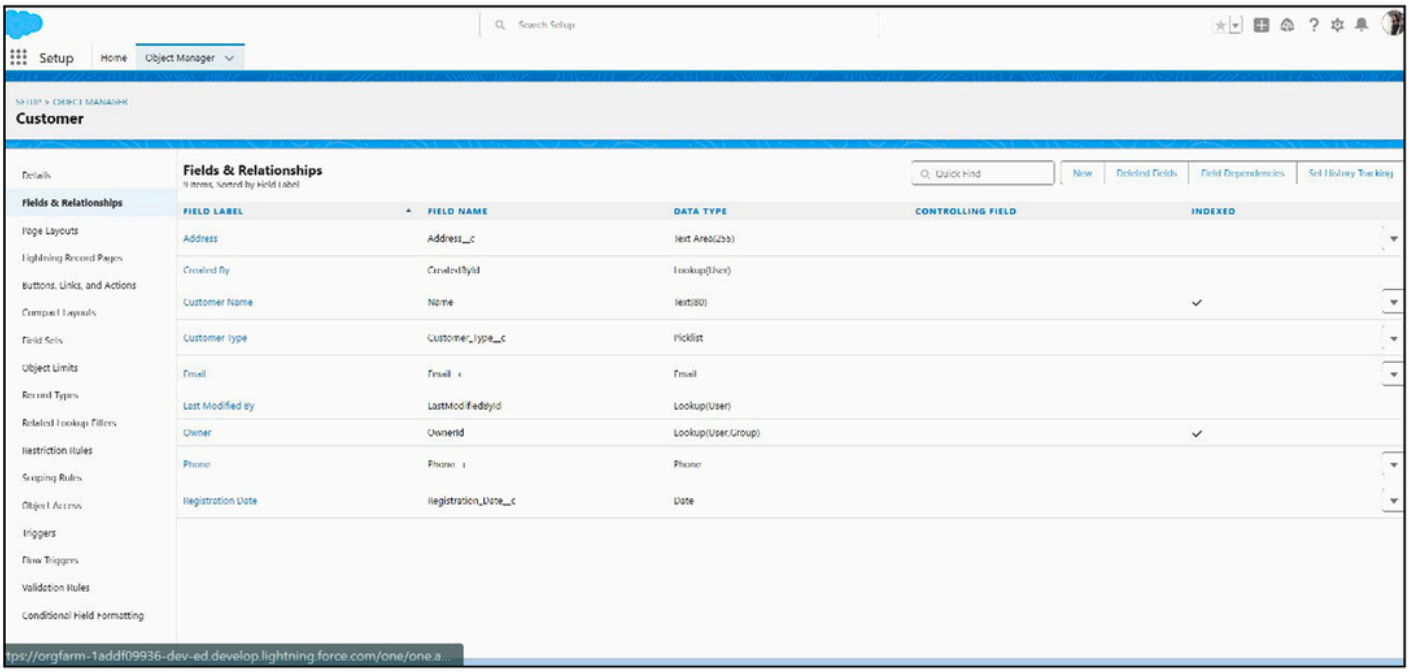
**Appointment:** Appointment ID, Customer Name, Service Provider Name, Date & Time, Status

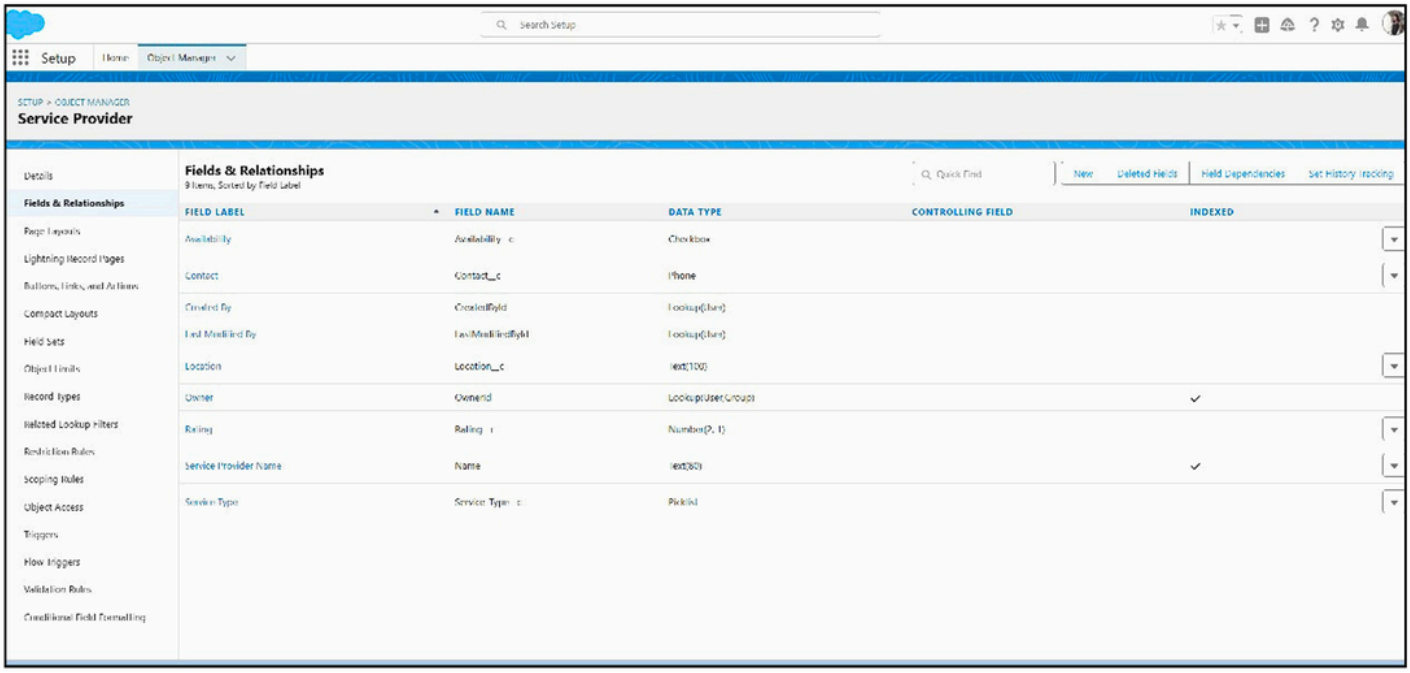
**Feedback:** Feedback ID, Customer Name, Service Provider Name, Rating, Comments, Feedback Date

**Customer Address:** Address ID, Customer Name, Street, City, State, Pincode




Search Setup
Setup
Home
Object Manager
SETUP > OBJECT MANAGER
Service Request
Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Restriction Rules
Scoping Rules
Object Access
Triggers
Flow Triggers
Validation Rules
Conditional Field Formatting
Fields & Relationships
10 Items, Sorted by Field Label
Quick Find
New
Deleted Fields
Field Dependencies
Set History Tracking
FIELD LABEL
FIELD NAME
DATA TYPE
CONTROLLING FIELD
INDEXED
Created By
CreatedById
Lookup(User)
Customer
Customer__c
Lookup(Customer)
Description
Description__c
Text Area(255)
Last Modified By
LastModifiedById
Lookup(User)
Owner
OwnerId
Lookup(User,Group)
Priority
Priority__c
Picklist
Request Date
Request_Date__c
Date
Service Provider
Service_Provider__c
Lookup(Service Provider)
Service Request Name
Name
Auto Number
Status
Status__c
Picklist


Search Setup
Setup
Home
Object Manager
SETUP > OBJECT MANAGER
Customer
Details
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Restriction Rules
Scoping Rules
Object Access
Triggers
Flow Triggers
Validation Rules
Conditional Field Formatting
Fields & Relationships
9 Items, Sorted by Field Label
Quick Find
New
Deleted Fields
Field Dependencies
Set History Tracking
FIELD LABEL
FIELD NAME
DATA TYPE
CONTROLLING FIELD
INDEXED
Address
Address__c
Text Area(255)
Created By
CreatedById
Lookup(User)
Customer Name
Name
Text(80)
Customer Type
Customer_Type__c
Picklist
Email
Email__c
Email
Last Modified By
LastModifiedById
Lookup(User)
Owner
OwnerId
Lookup(User,Group)
Phone
Phone__c
Phone
Registration Date
Registration_Date__c
Date
tps://orgfarm-1addf09936-dev-ed.develop.lightning.force.com/one/one.a...




## 3. Record Types

**Service Request Object Layouts (Multiple Record Types):**

**a) Normal Request Layout**

Fields: Request ID, Customer, Service Provider, Request Date, Status, Description, Priority

**b) Emergency Request Layout**

Fields: Request ID, Customer, Service Provider, Request Date, Status (customized)

Extra Fields: Response Time, Extra Charges, Priority (High)

**c) Cleaning Service Layout**

Fields: Request ID, Customer, Service Provider, Request Date, Status (customized)

Extra Fields: Cleaning Type, Estimated Hours, Description

**d) Plumbing / Electrical Layouts**

Fields: Request ID, Customer, Service Provider, Request Date, Status, Description, Priority

## 4. Payment Object Layouts

**a) Cash Payment Layout**

Fields: Payment ID, Amount, Payment Date, Collected By

**b) UPI Payment Layout**

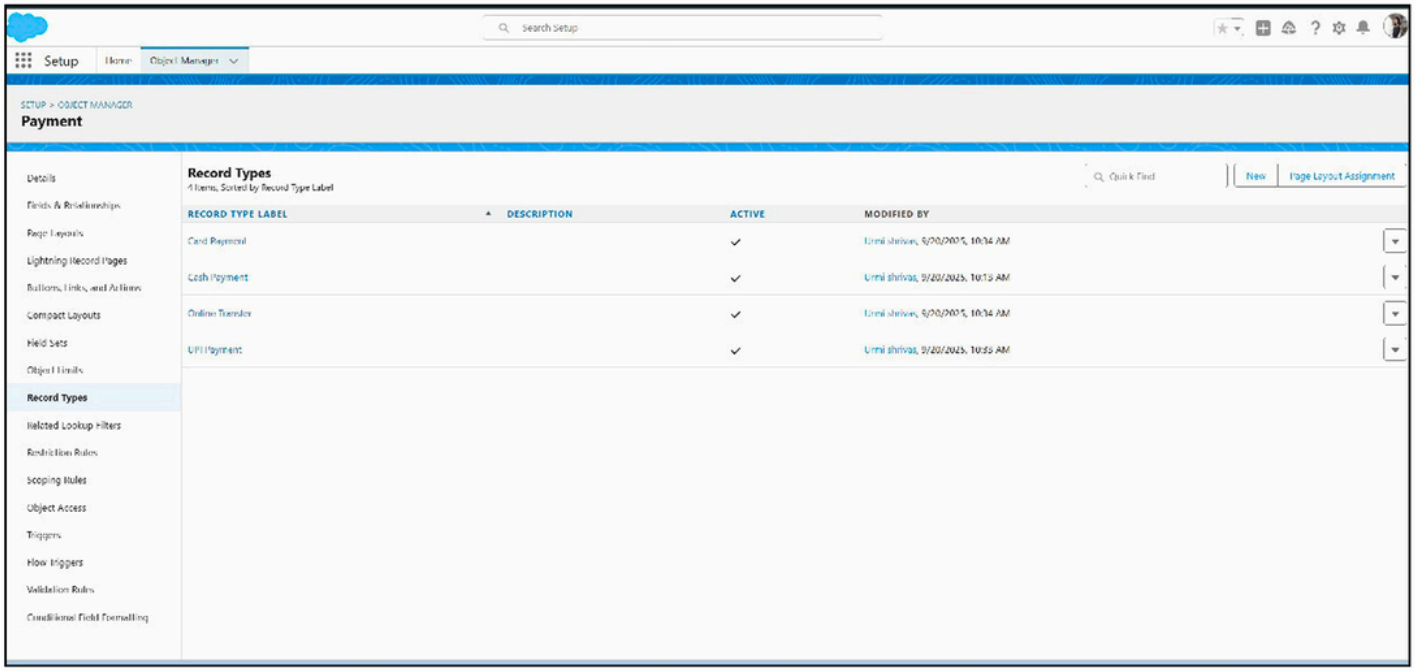
Fields: Payment ID, Amount, Payment Date, Transaction ID, UPI Reference

**c) Card Payment Layout**

Fields: Payment ID, Amount, Payment Date, Card Type, Last 4 Digits

**d) Wallet/Online Layout**

Fields: Payment ID, Amount, Payment Date, Wallet Name / Bank Reference

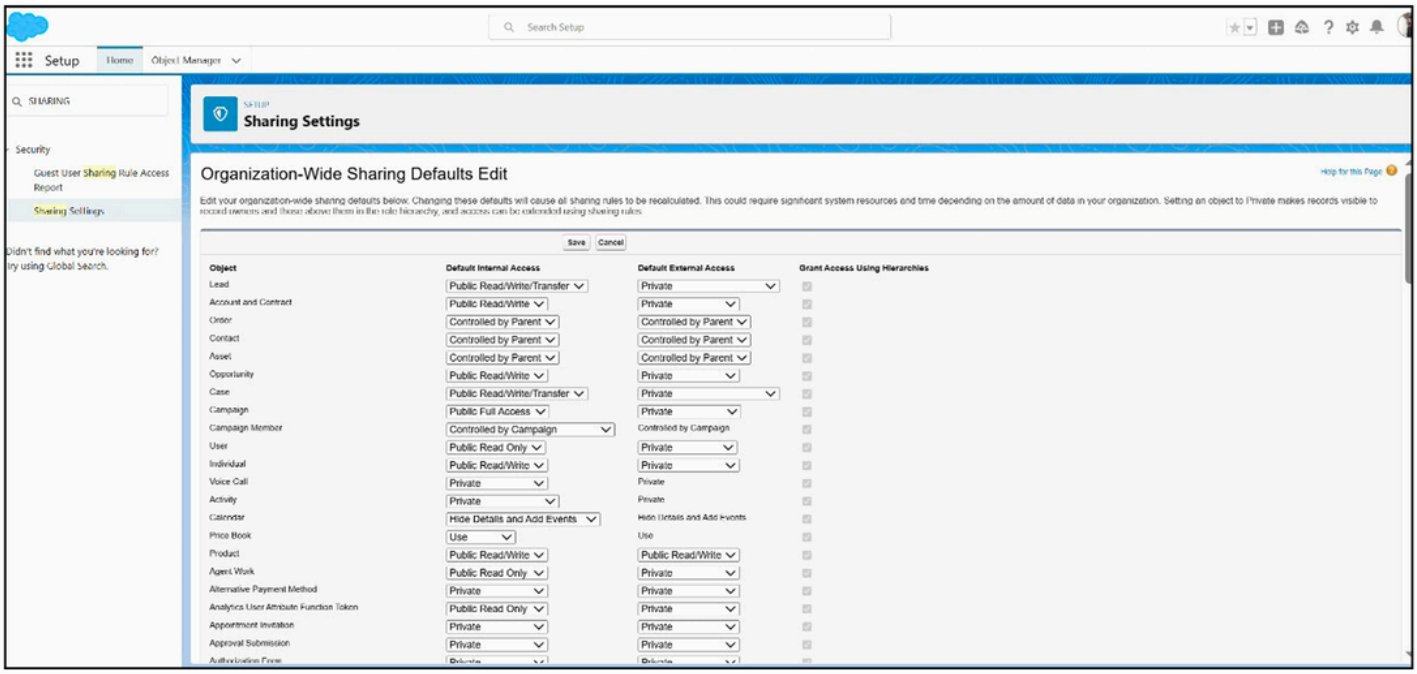



## 5. Sharing Rules

**Customer Records:** Shared with Customer Manager (Read/Write)

**Service Requests:** Shared with Provider Manager (Read/Write)

**Reports:** Shared with Admin Team (Read-Only)



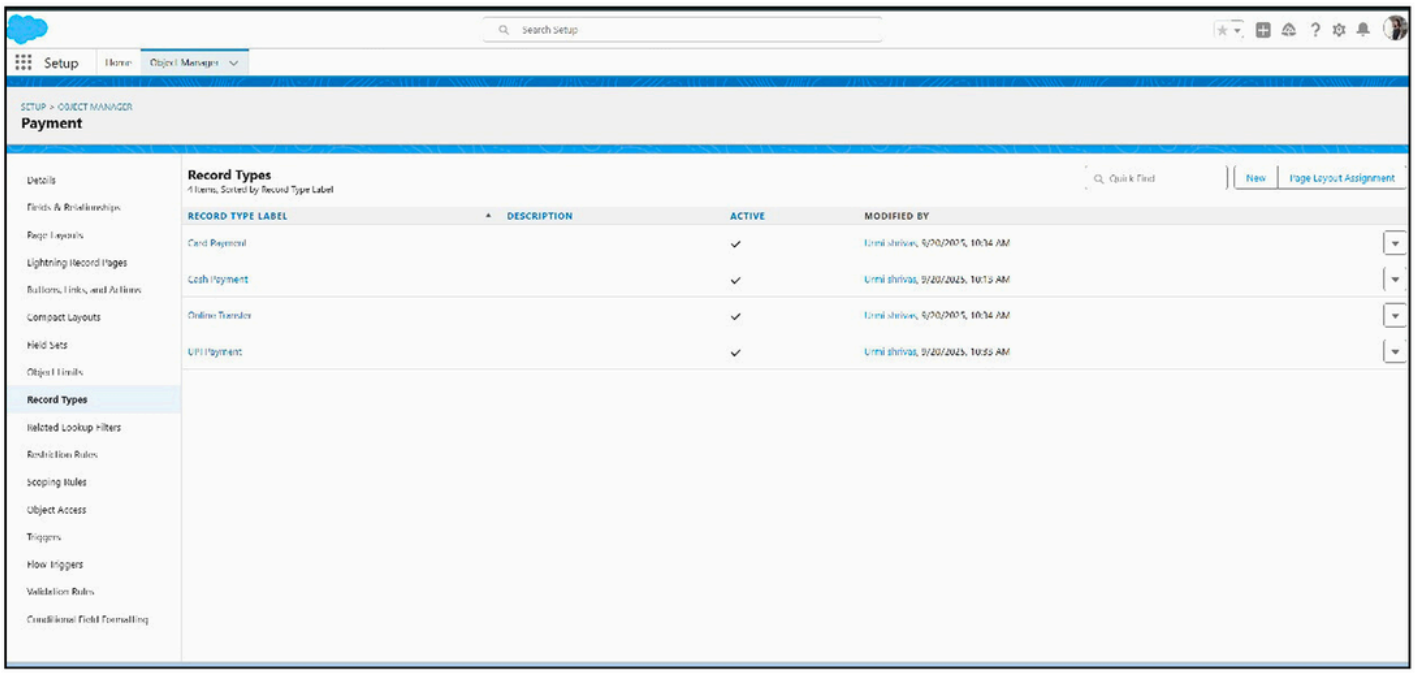
## 6. Page Layouts

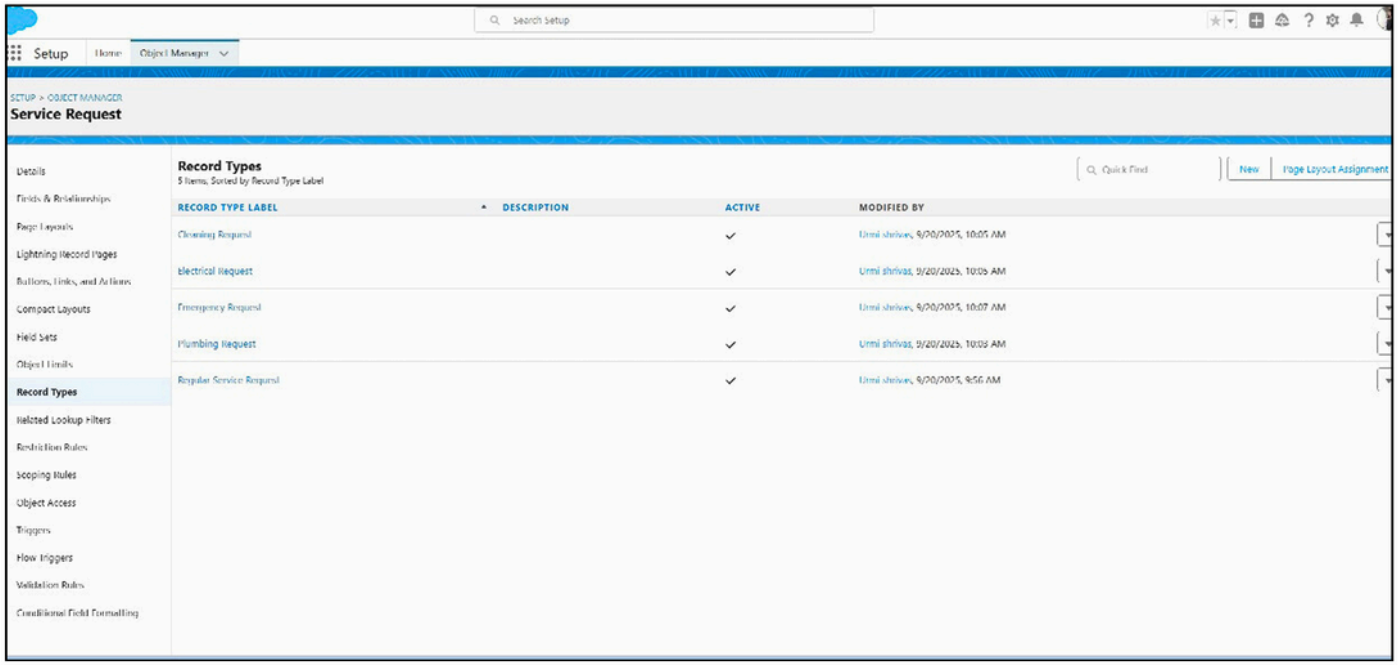
**Customer:**

- **Customer Info:** Name, Email, Phone, Address, Customer Type, Registration Date
- **Related Records (1-column):** Service Requests

**Service Provider:**

- **Provider Info:** Name, Service Type, Rating, Availability, Contact, Location
- **Related Records (1-column):** Service Requests



## 7. Compact Layouts

**Customer:**

Key Fields: Name, Email, Phone, Customer Type

Layout Name: Customer_Compact_Layout (Set as Primary)

**Service Provider:**

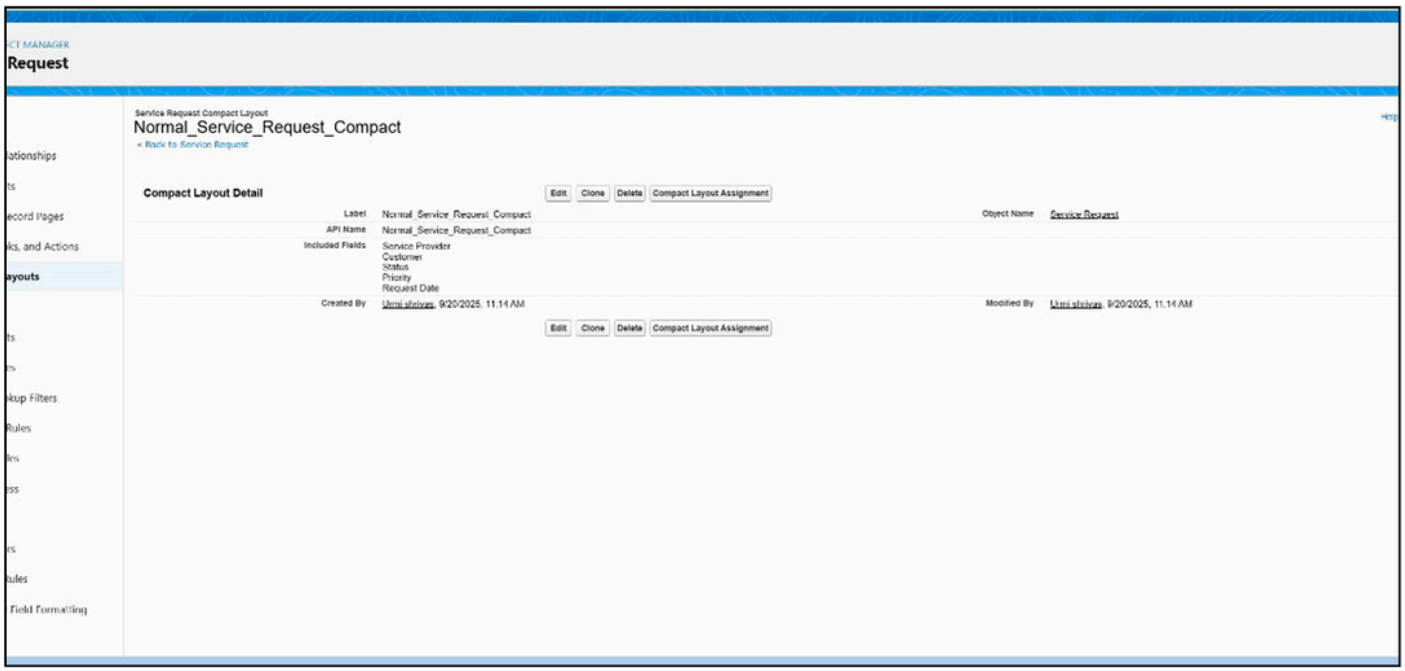
Key Fields: Name, Service Type, Rating, Availability

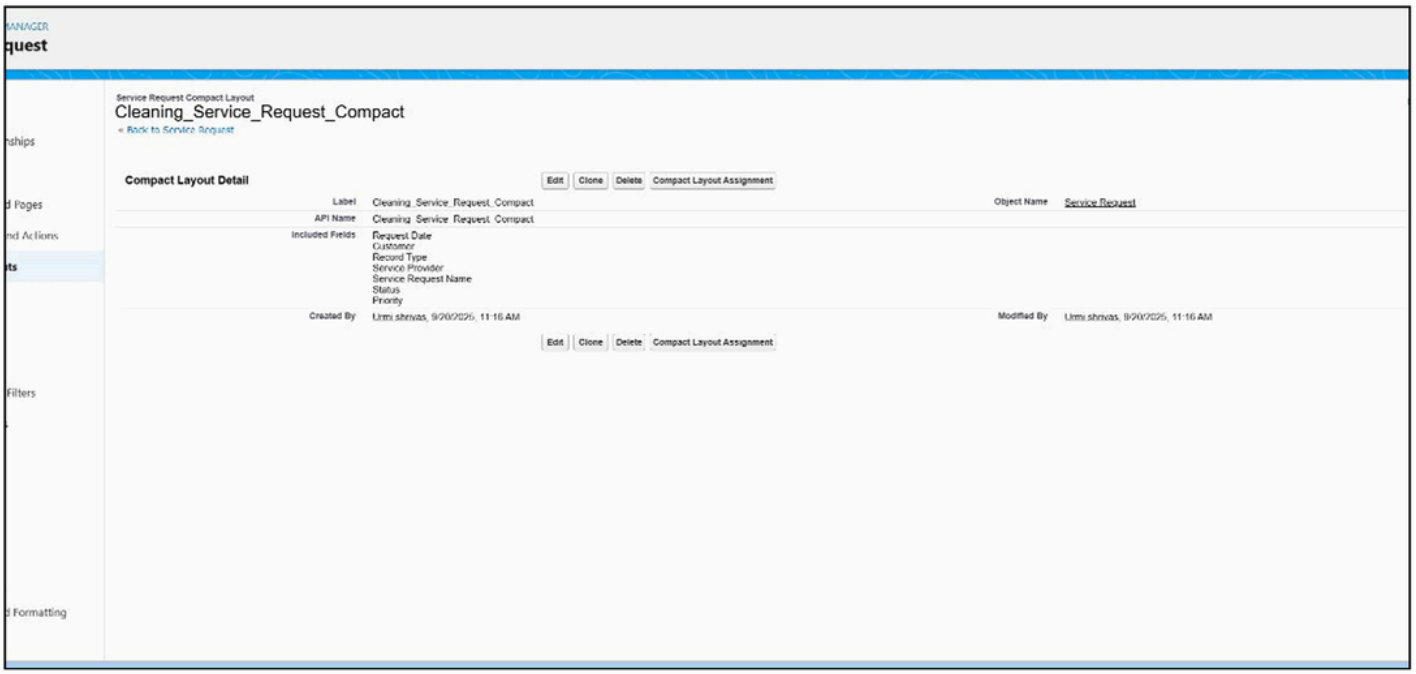
Layout Name: Service_Provider_Compact_Layout (Set as Primary)

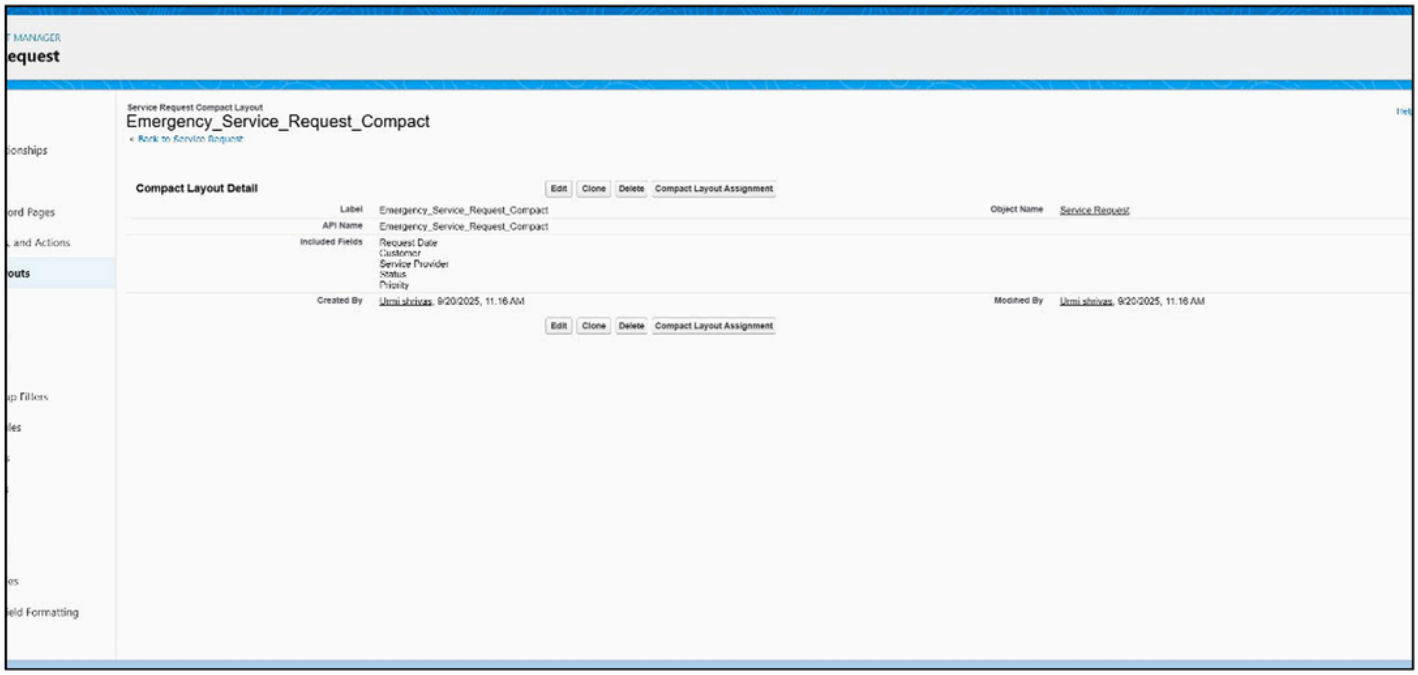
**Service Request:**

Key Fields: Request ID, Customer, Service Provider, Status, Priority

Layout Name: Service_Request_Compact_Layout (Set as Primary)






## 8. Schema Builder

**Schema Builder** provides a visual representation of all objects and their relationships.

**Objects Displayed:**
Customer, Service Provider, Service Request, Payment, Appointment, Feedback, Customer Address

**Relationships:**

- **Customer → Service Request:** Lookup / Master-Detail
- **Service Provider → Service Request:** Lookup / Master-Detail
- **Service Request → Payment:** Lookup / Master-Detail

- **Service Request → Appointment:** Lookup
- **Service Request → Feedback:** Lookup
- **Customer → Customer Address:** Lookup

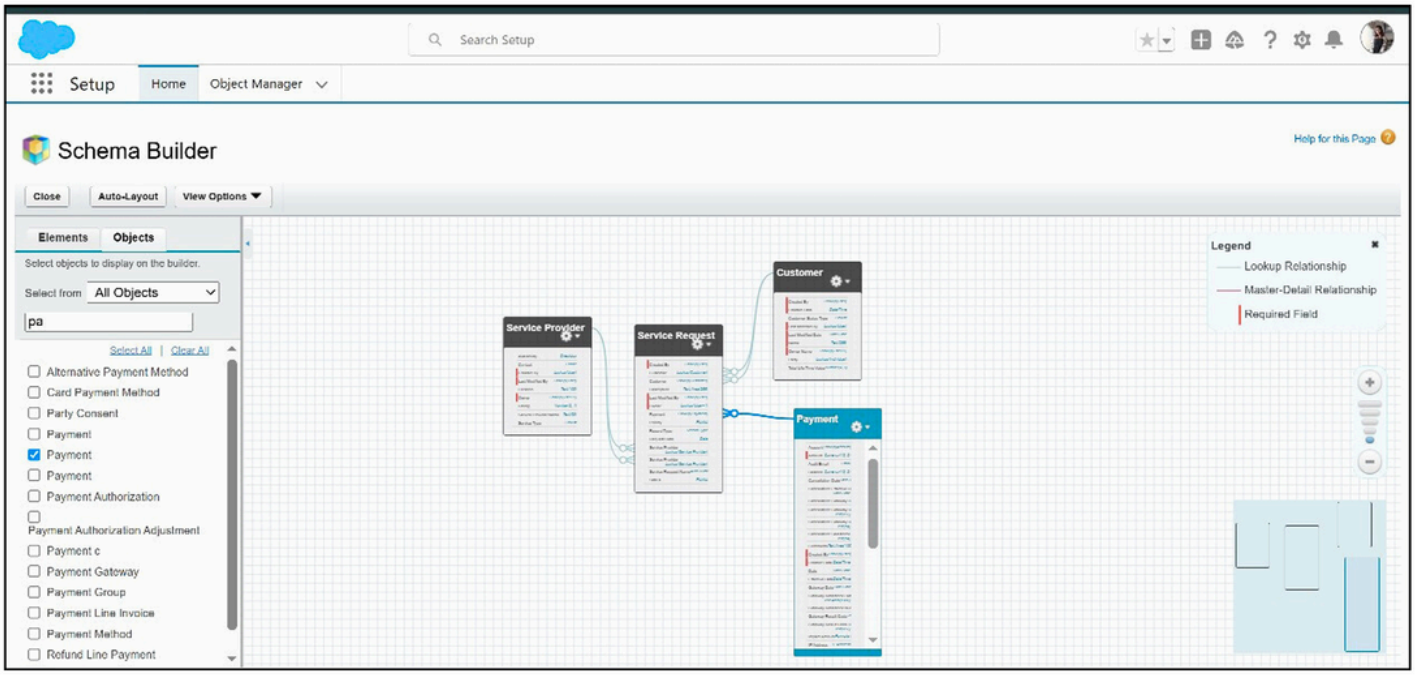
**Lookup Relationship:**
Used to link **Customer → Customer Address** and **Service Request → Appointment**, allowing child records to exist independently.

**Master-Detail Relationship:**
Used to connect **Service Request → Payment** and **Service Provider → Service Request**, ensuring ownership inheritance, sharing, and cascade deletion.

**Hierarchical Relationship:**
Not used (optional for user hierarchy).

**Junction Object:**
Created (if required) to connect two objects in a many-to-many relationship, e.g., **Service Provider ⇔ Services Offered**.

**External Object:**
Not implemented, as no external data connection was required.




# Phase 4: Process Automation (Admin)

## 1. Validation Rules

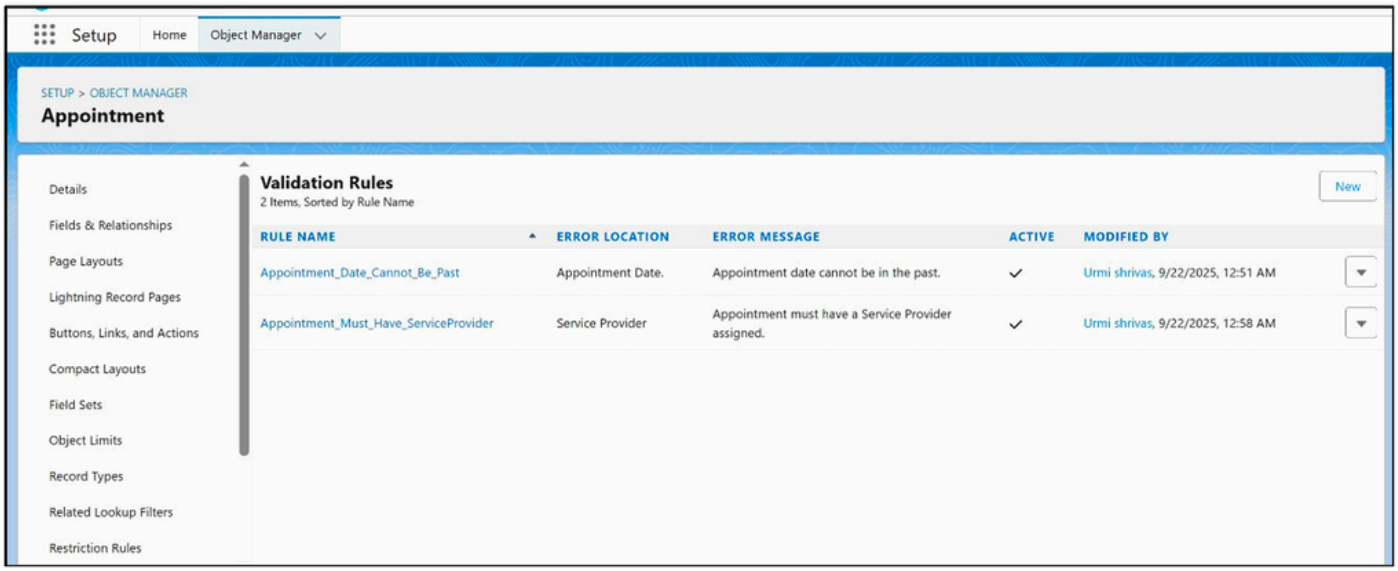
### Service Request

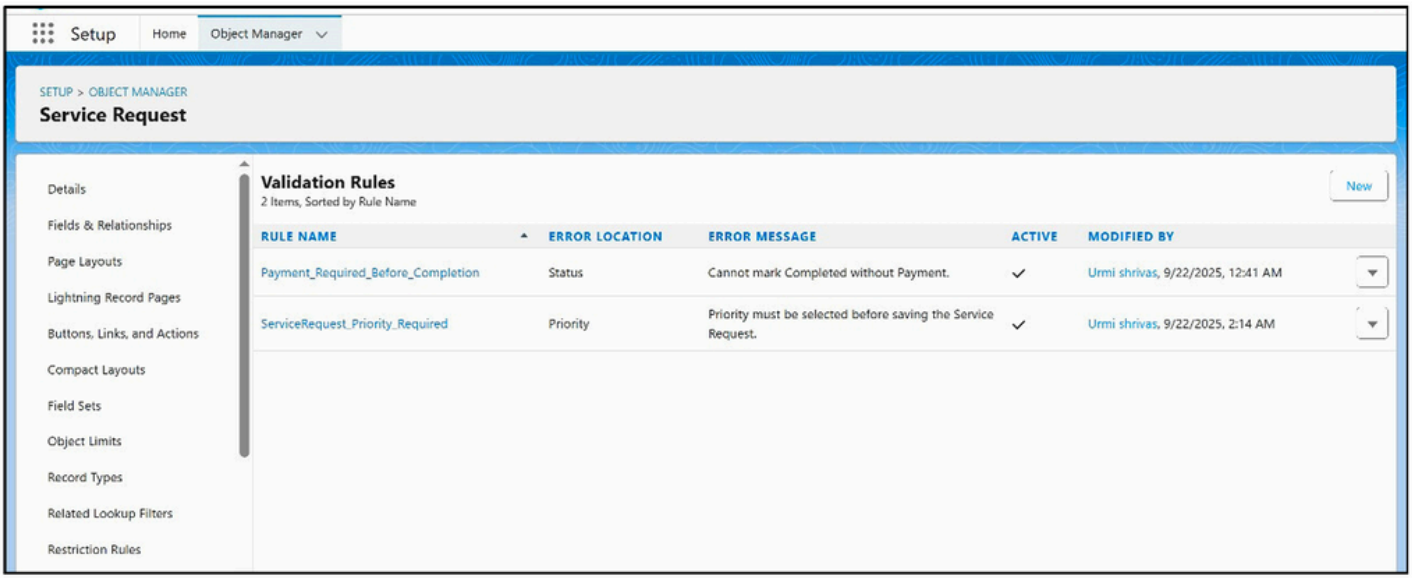
- Mandatory field validations such as *Service Type* and *Customer Name* are required before saving the record.
- Restricted invalid data entries — for example, *Service Request Date* cannot be in the past.

**Appointment**

- *Appointment Date* must be greater than or equal to the current date.
- Prevented blank entries in important fields such as *Service Provider*.

These rules ensure **data integrity** and **prevent incorrect inputs** in the system.




## 2. Workflow Rules

- No active workflow rules were implemented in this project.
- Instead, **Flows** were used to automate processes, as per **Salesforce best practices** and the recommendation to move away from traditional workflow rules.

## 3. Process Builder

- No active Process Builders were implemented.
- All automation logic was handled through **Flows**, since Salesforce recommends **migrating from Process Builder to Flow Builder** for better flexibility and long-term maintenance.

## 4. Approval Process

**Object:** Payment
**Purpose:** To validate and approve payment requests before processing.

**Process Steps:**

1. **Submission:** Service Provider submits a payment record for approval.
2. **Approval Stage:** Request is sent to the Admin for review and decision.

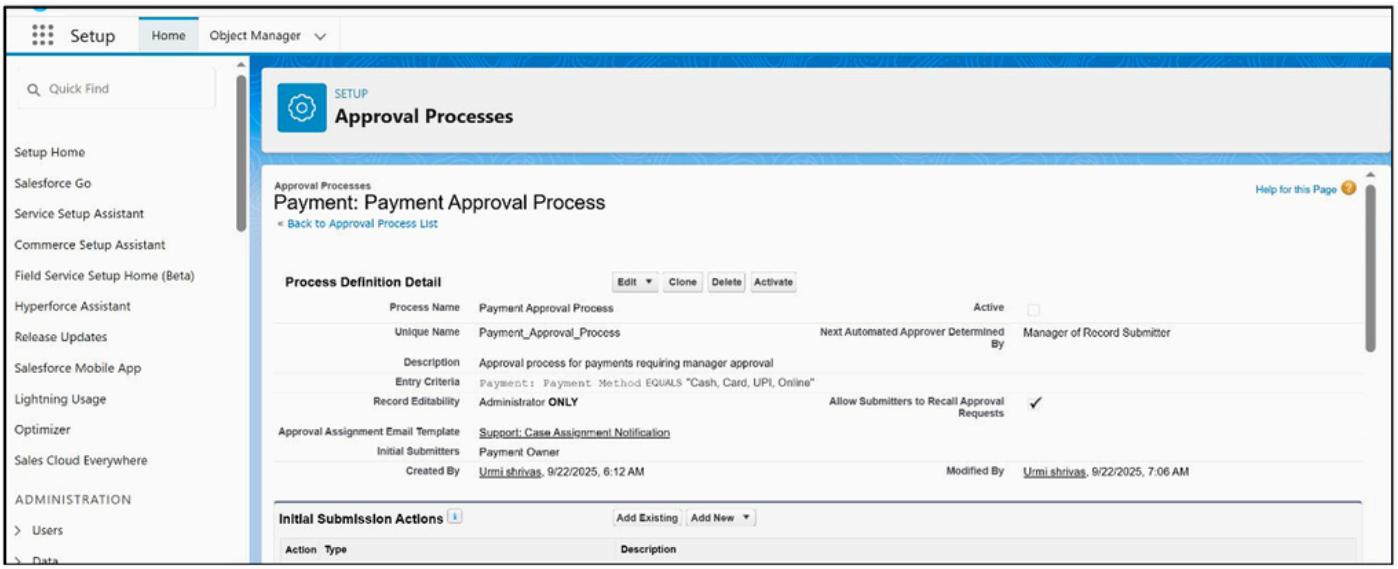
**On Approval:**

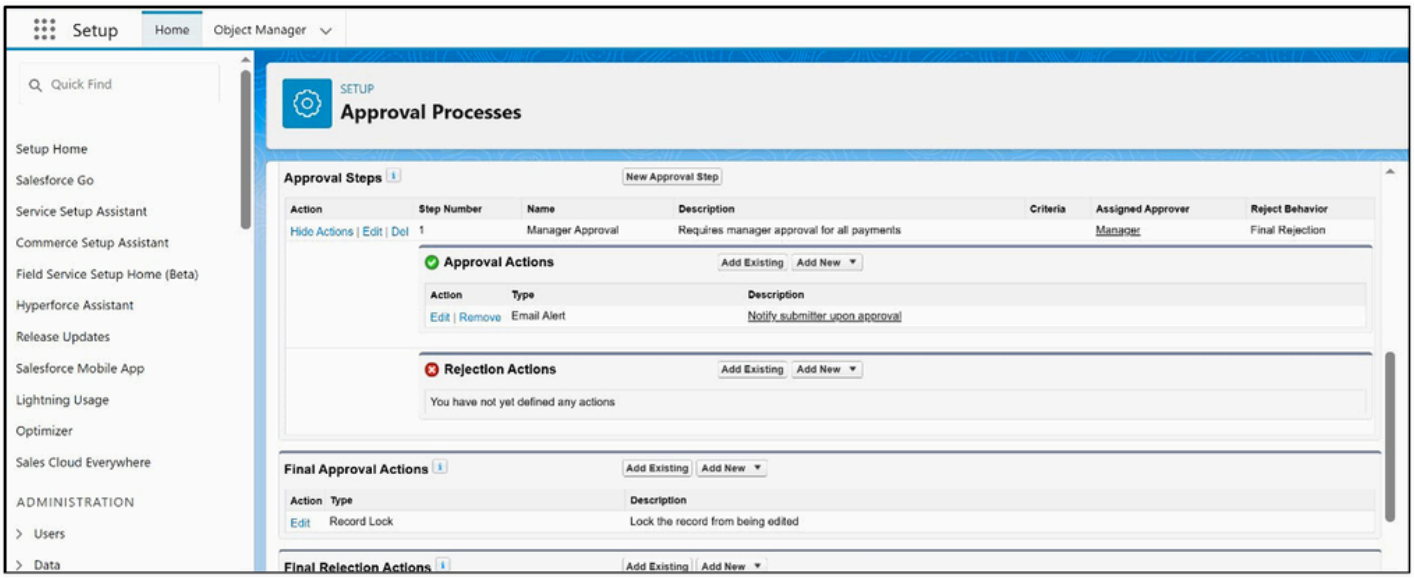
- *Payment Status* is automatically updated to **"Approved"**.
- The record becomes eligible for further processing.

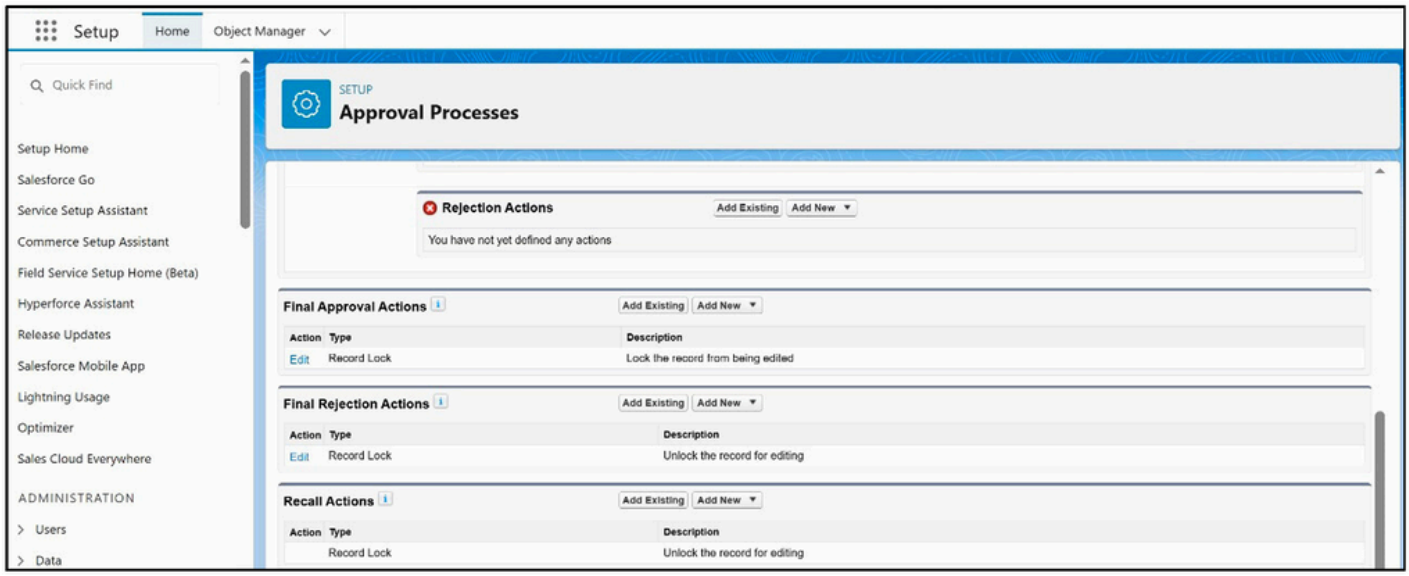
**On Rejection:**

- *Payment Status* changes to **"Rejected"**.
- The Service Provider receives a notification to recheck or update payment details.

This process ensures **financial accuracy** and **admin oversight** in payment transactions.



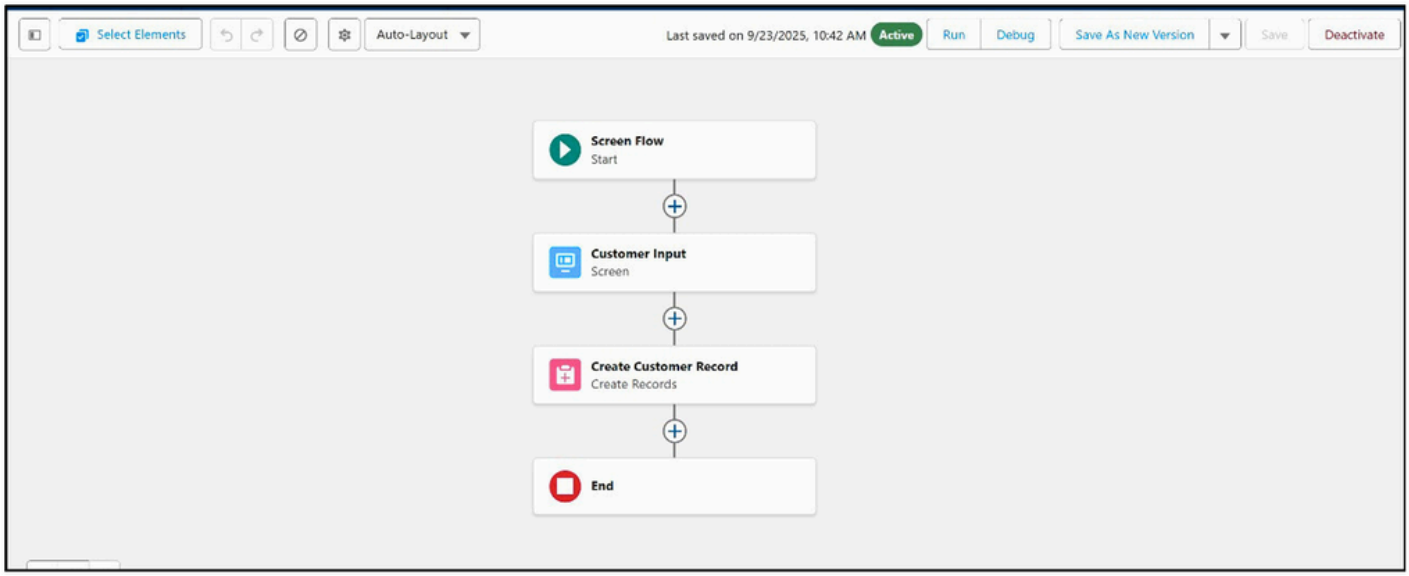



## 5. Flow Builder (Screen, Record-Triggered, Scheduled, Auto-Launched)

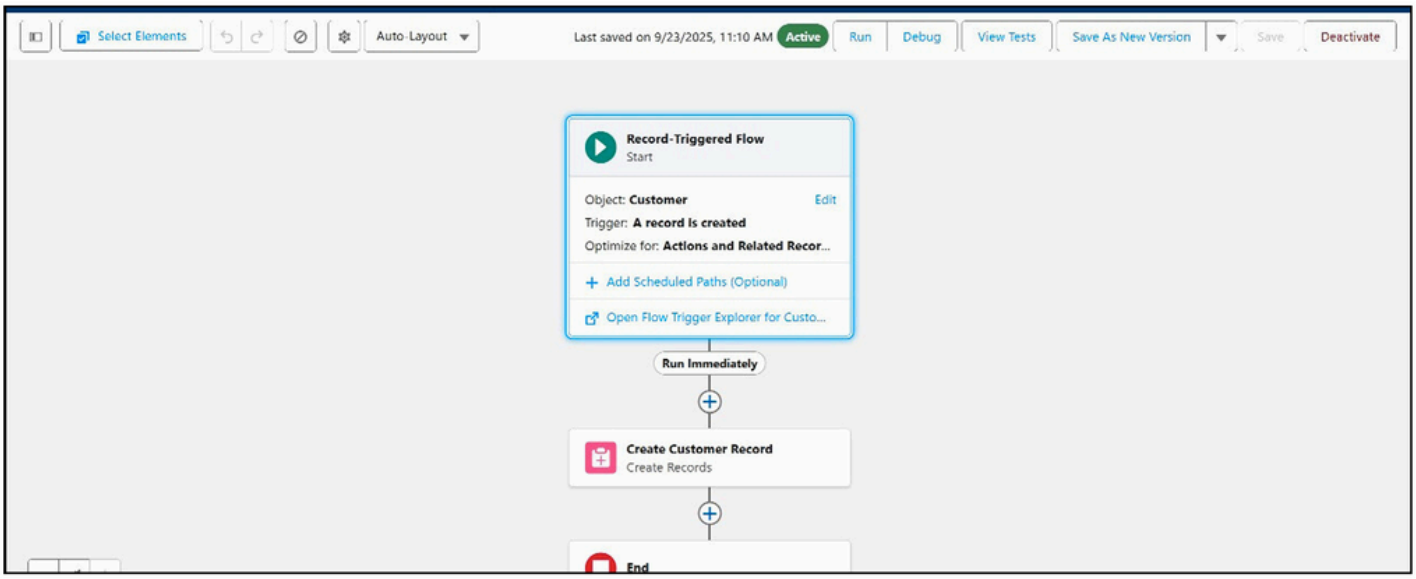
### Screen Flow

- Created for **Appointment Booking**.
- Users can select *Service Type*, *Date*, and *Service Provider*.
- Upon submission, the **Appointment record** is automatically created in the system.



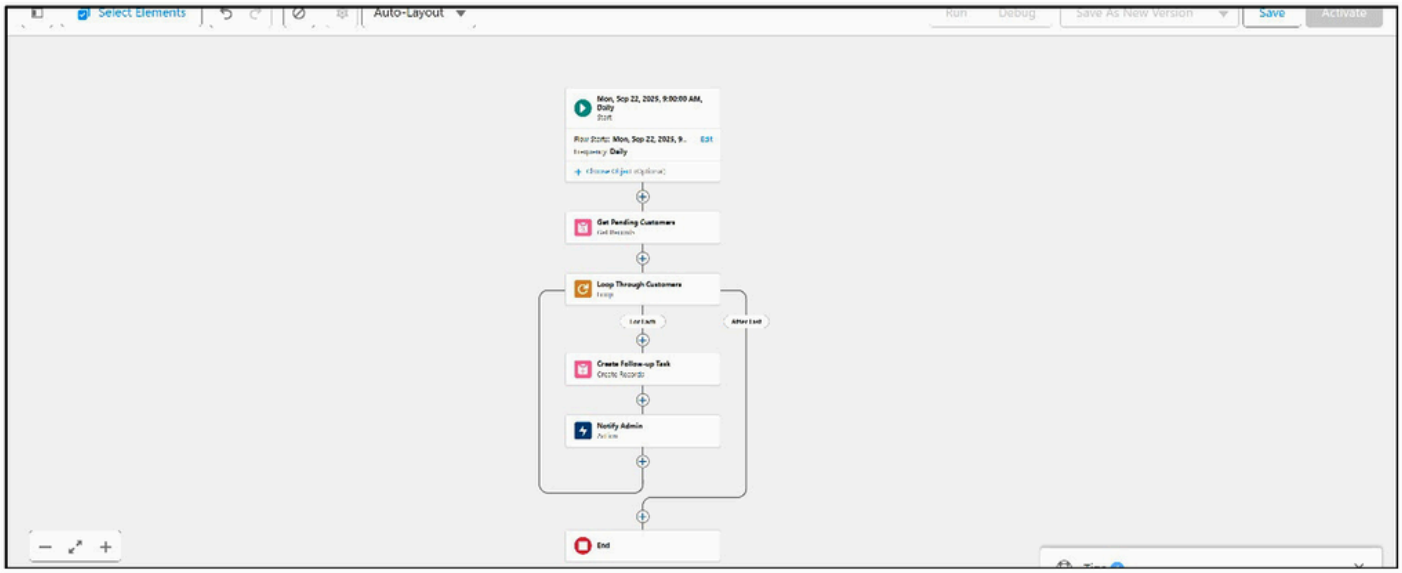

**Record-Triggered Flow**

- Created on the **Service Request** object.
- When a new Service Request is created → *Status* automatically set to “Pending”.
- When updated → Admin receives a notification.



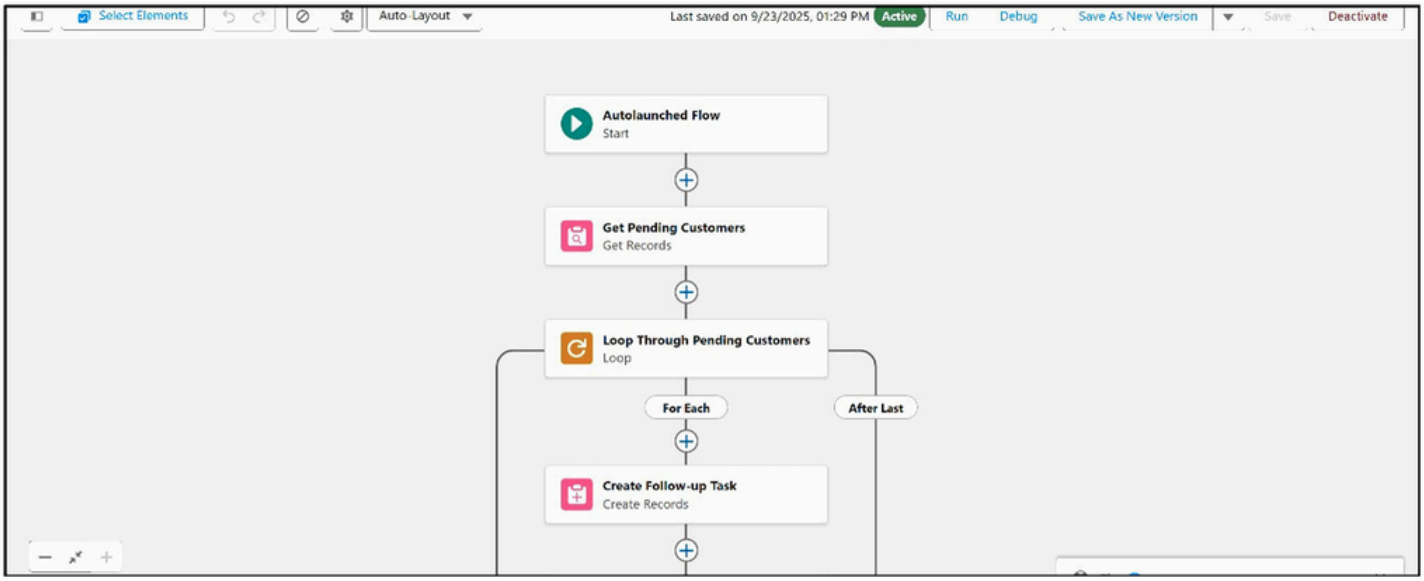

**Scheduled Flow**

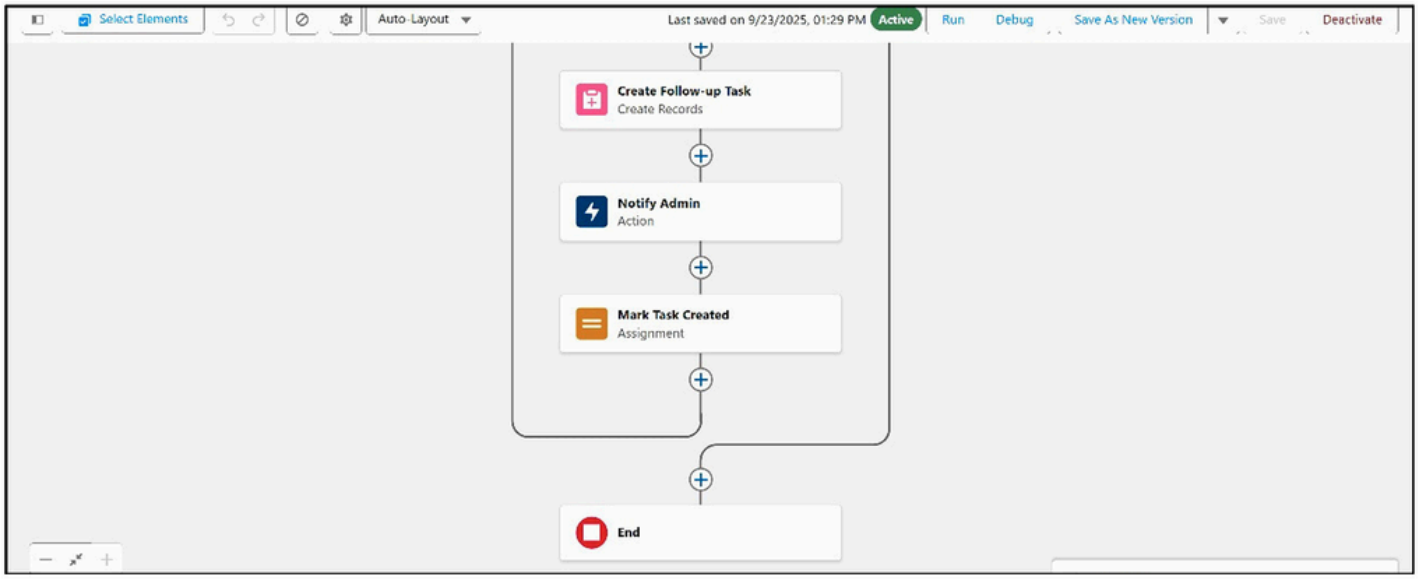
- Created for **Appointment Reminders**.
- Runs **daily at a fixed time** to send reminders before the appointment date.
- Ensures customers and providers are notified in advance.




**Auto-Launched Flow**

- Developed for **Payment Approval Automation**.
- Automatically triggered during the payment approval process.
- Updates *Payment Status* based on the Admin's decision:
  - Approved → Status set to "Approved"
  - Rejected → Status set to "Rejected"
- Ensures the system handles **payment updates automatically** without manual changes.






## 6. Email Alerts

### Customer Follow-Up Notification

- Triggered when a customer follow-up is required.
- Sends automated email to the assigned user for necessary action.

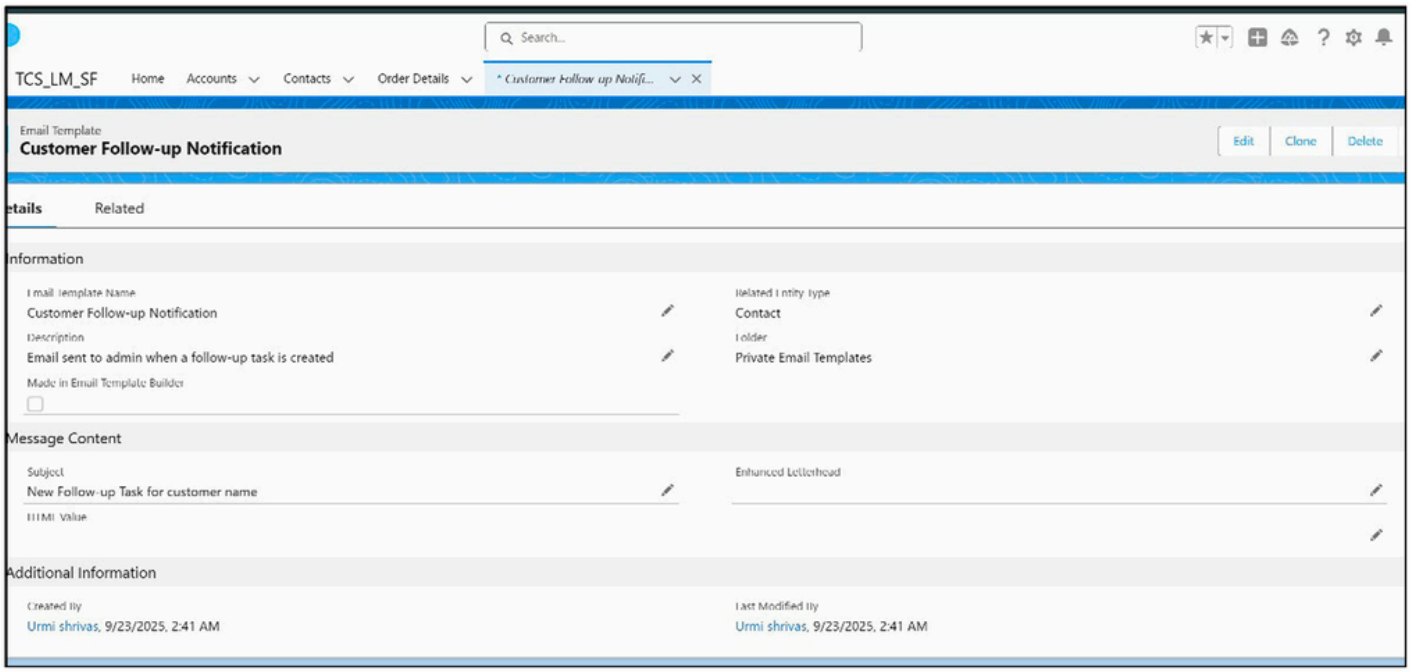
### Customer Task Assigned Notification

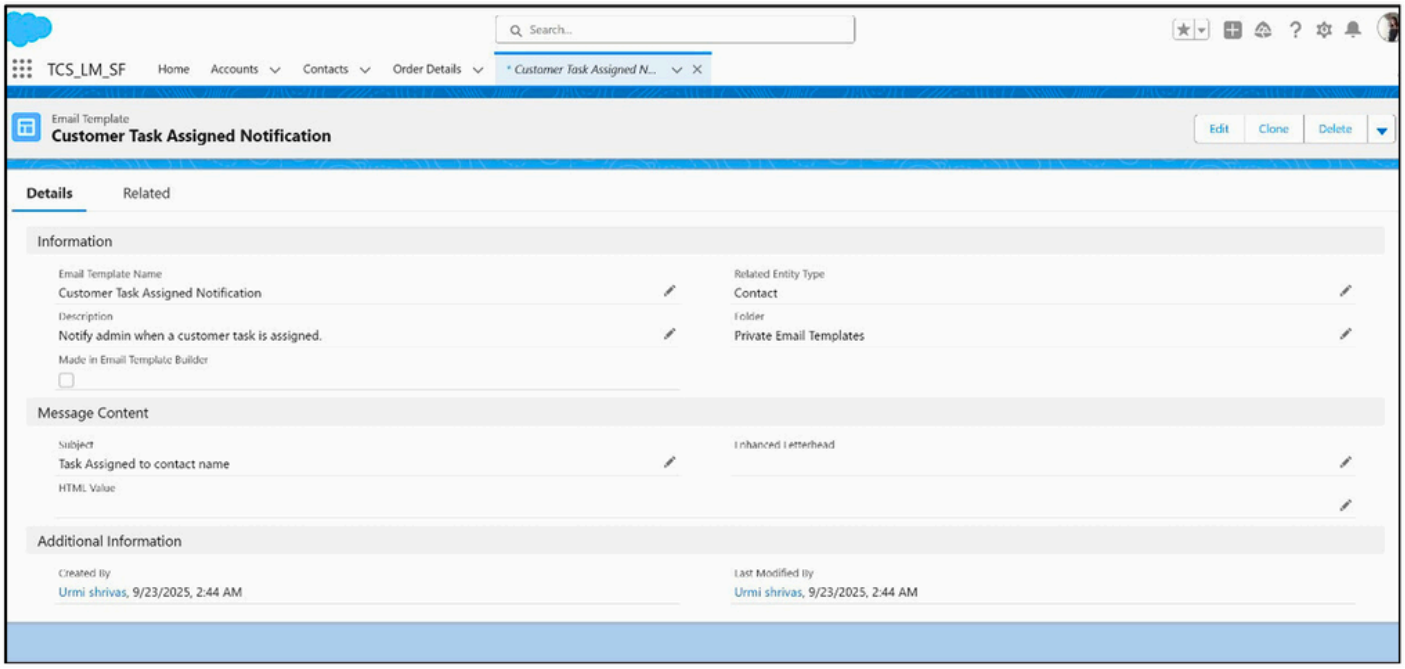
- Triggered when a new task is assigned.
- Sends email notification to the respective user with task details.

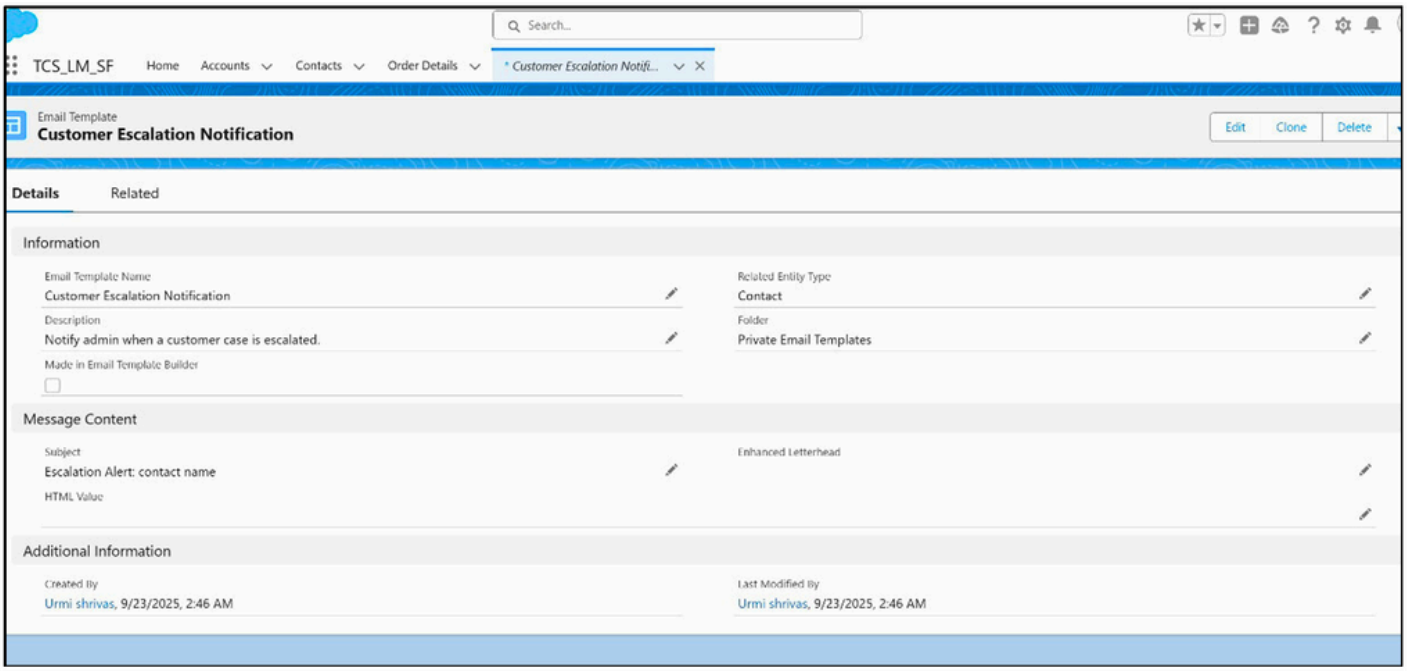
### Customer Escalation Notification

- Triggered when a customer issue is escalated.
- Sends an email alert to the higher authority (Admin/Manager) for immediate attention.

These alerts **automate communication** and ensure **timely follow-ups and actions**.


Search...
TCS_LM_SF
Home
Accounts
Contacts
Order Details
* Customer Follow-up Notifi...
Email Template
Customer Follow-up Notification
Edit
Clone
Delete
etails
Related
nformation
Email Template Name
Customer Follow-up Notification
Related Entity Type
Contact
Description
Email sent to admin when a follow-up task is created
Folder
Private Email Templates
Made in Email Template Builder
Message Content
Subject
New Follow-up Task for customer name
Enhanced Letterhead
HTML Value
Additional Information
Created By
Urmi shrivas, 9/23/2025, 2:41 AM
Last Modified By
Urmi shrivas, 9/23/2025, 2:41 AM


Search...
TCS_LM_SF
Home
Accounts
Contacts
Order Details
* Customer Task Assigned N...
Email Template
Customer Task Assigned Notification
Edit
Clone
Delete
Details
Related
Information
Email Template Name
Customer Task Assigned Notification
Related Entity Type
Contact
Description
Notify admin when a customer task is assigned.
Folder
Private Email Templates
Made in Email Template Builder
Message Content
Subject
Task Assigned to contact name
Enhanced Letterhead
HTML Value
Additional Information
Created By
Urmi shrivas, 9/23/2025, 2:44 AM
Last Modified By
Urmi shrivas, 9/23/2025, 2:44 AM




## 7. Field Updates

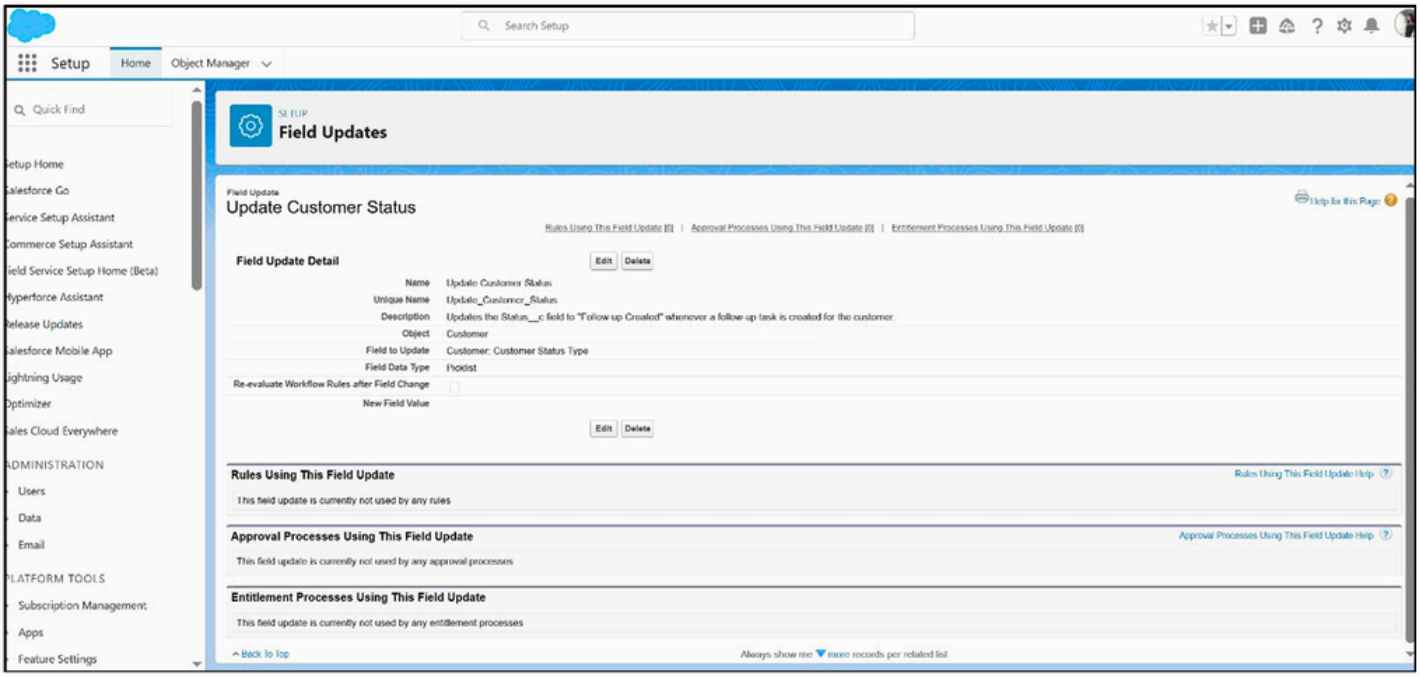
### Customer Status Update

- Automatically updates *Customer Status* based on actions performed in the system.

**Examples:**

- When a *Service Request* is completed → Status changes to **“Service Completed”**.
- During *Payment Approval* → Status updates to **“Payment Approved”** or **“Payment Rejected”**.

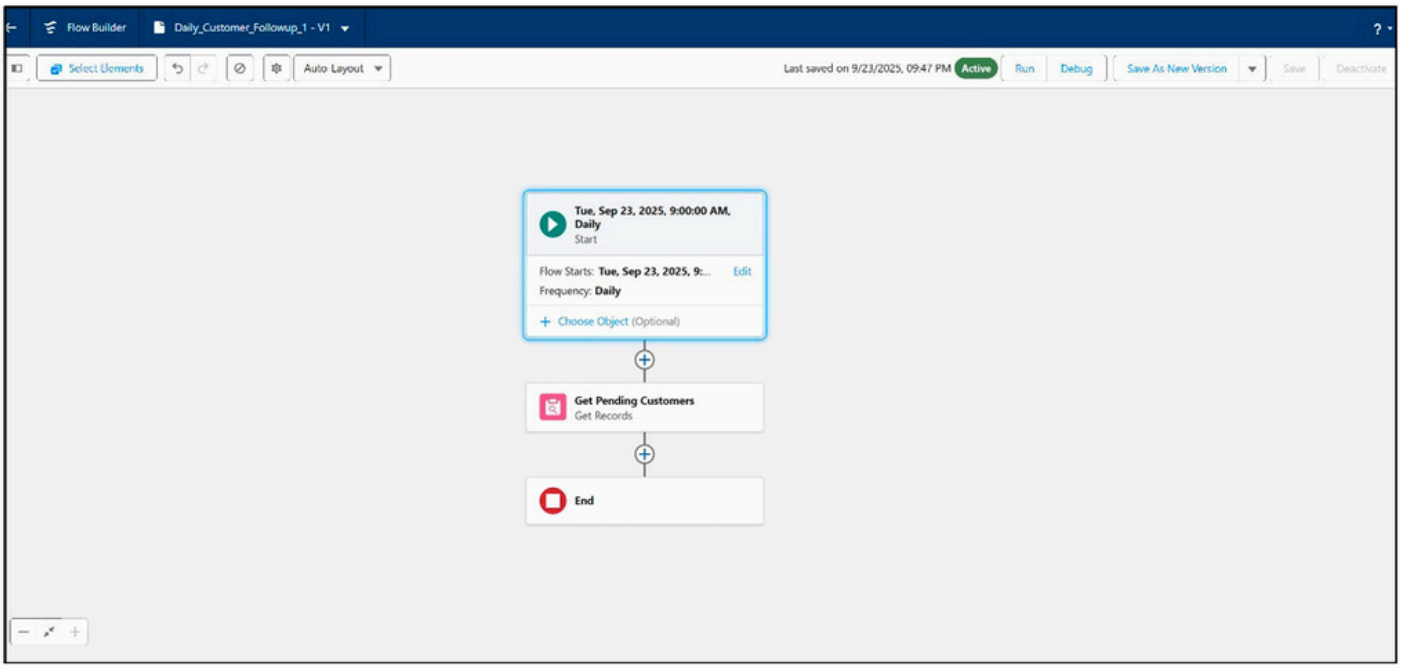
This automation maintains **data consistency** and **reduces manual intervention**.



## 8. Task

**Scheduled Tasks for Appointments**

- Tasks are automatically generated using a **Scheduled Flow**.
- These tasks remind users about upcoming appointments **one day before the scheduled date**.
- Tasks are assigned to respective users to ensure **timely follow-ups**.
- This reduces missed appointments and improves **customer service efficiency**.



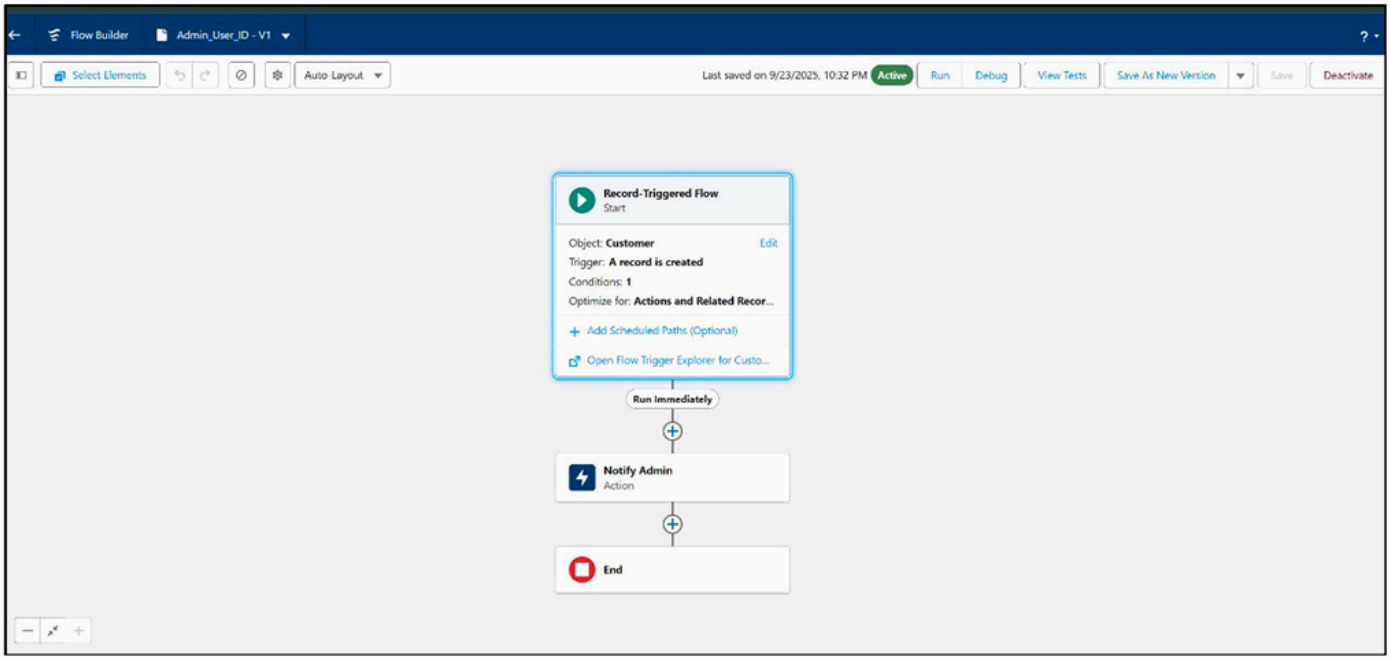

## 9. Custom Notifications

- Implemented using **Record-Triggered Flow** to deliver **real-time alerts** to users.

**Customer Service Notifications:**

- Triggered when a new *Service Request* is created or updated.
- Instantly notifies the assigned user about the change, ensuring they stay informed without manually checking records.

This approach enhances **real-time communication** and **response speed** across the CRM.




---

# Phase 5: Apex Programming (Developers)

**1. Classes & Objects**

Created an Apex Class Customer with variables: name, age, email, phoneNumber, city, isActive.

Added a constructor to initialize variables.

Implemented methods:

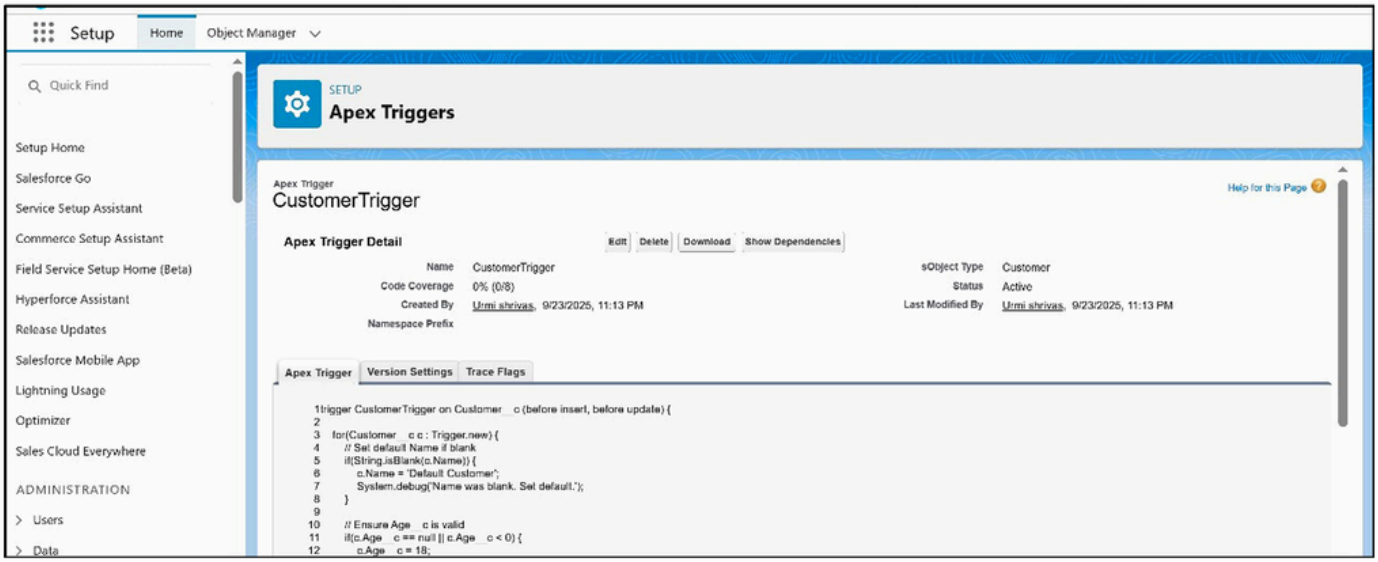
- displayInfo() → prints customer details.
- isAdult() → checks if age ≥ 18.

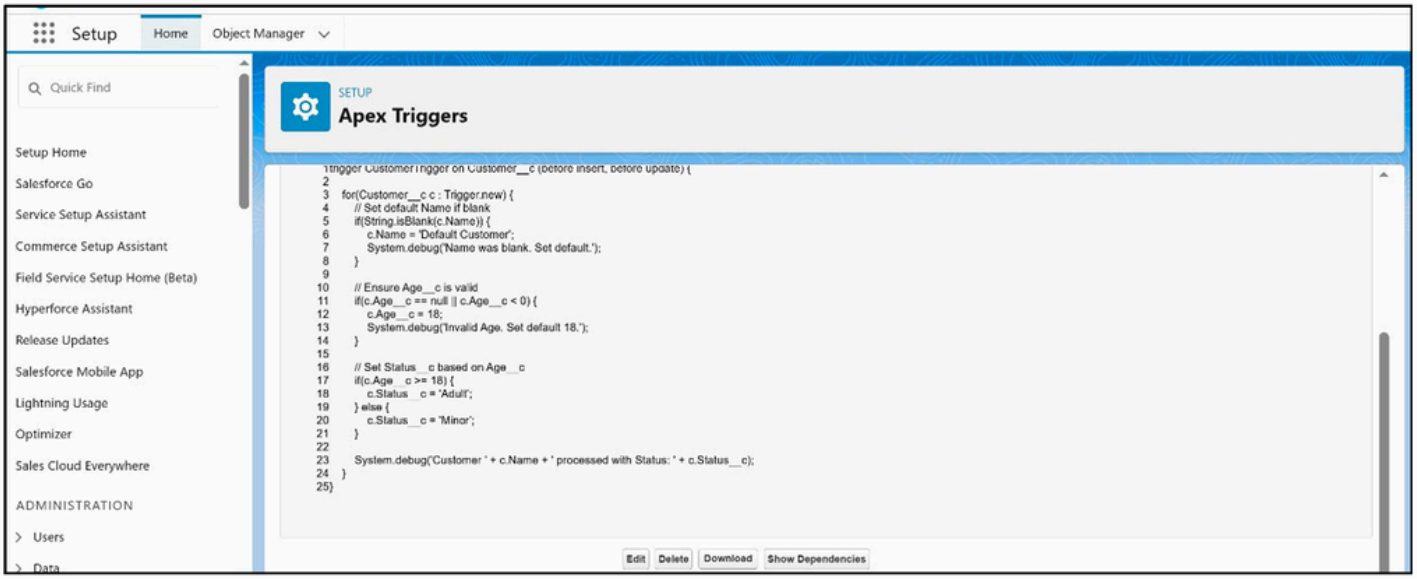
Created multiple Customer objects and called their methods.

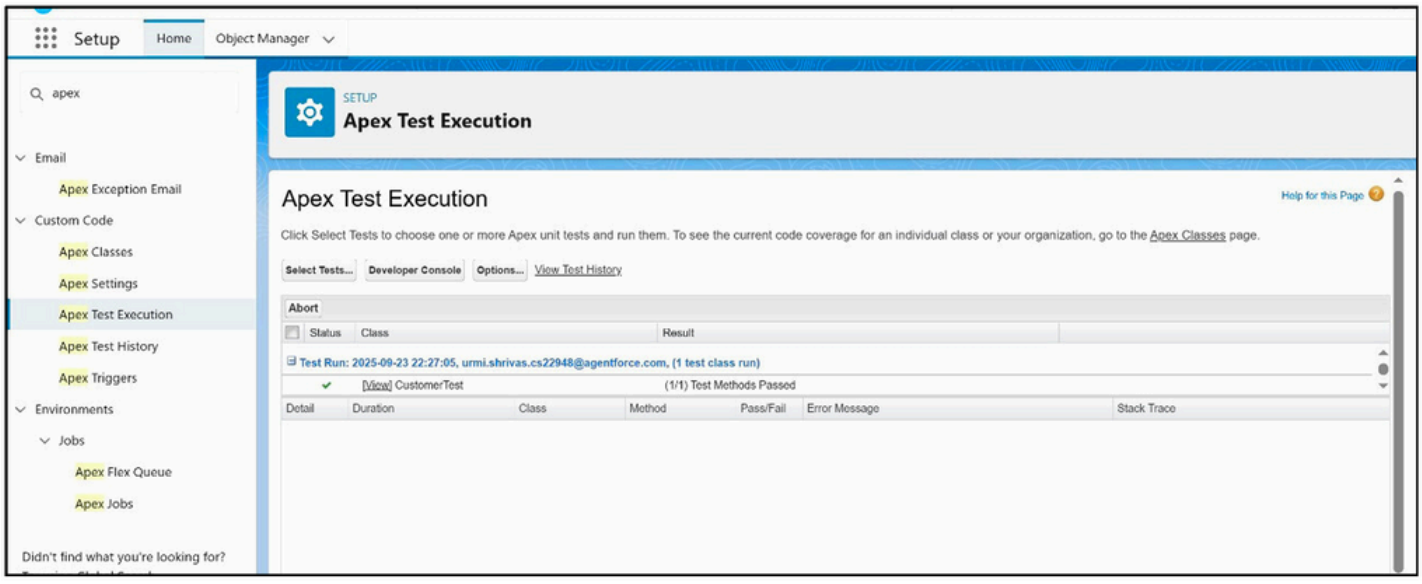
Added a static test method (runTest()) to test class functionality.

Used System.debug() to verify outputs.

Executed the class via **Apex Test Execution** in the org.



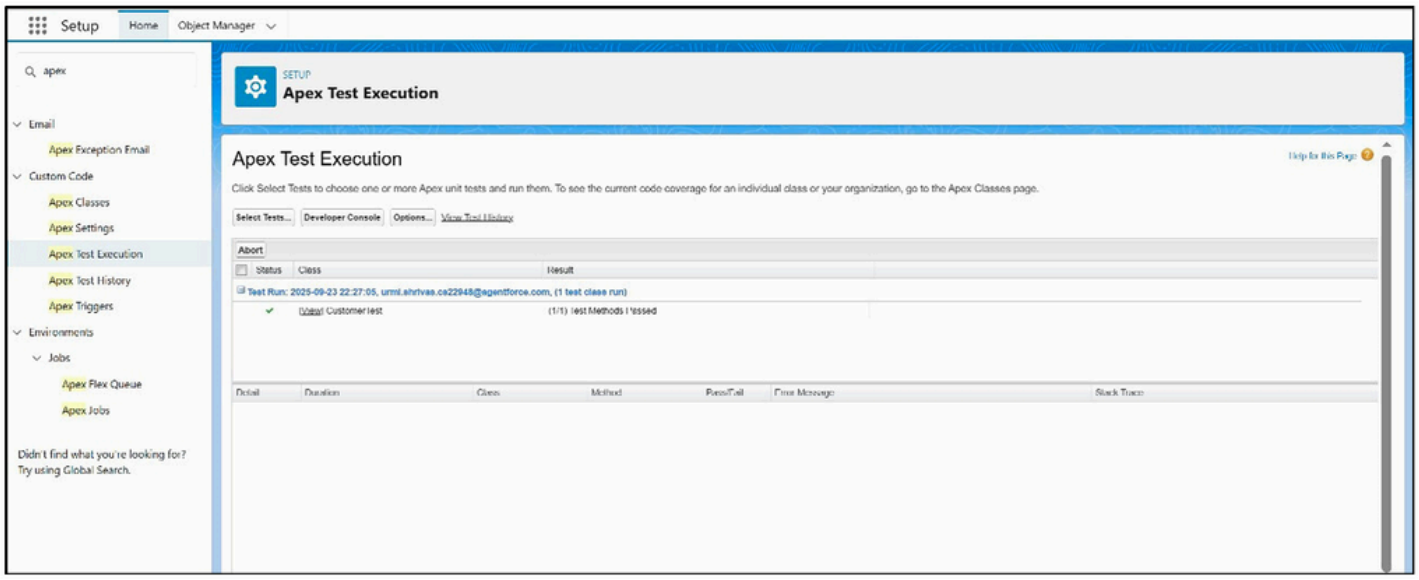



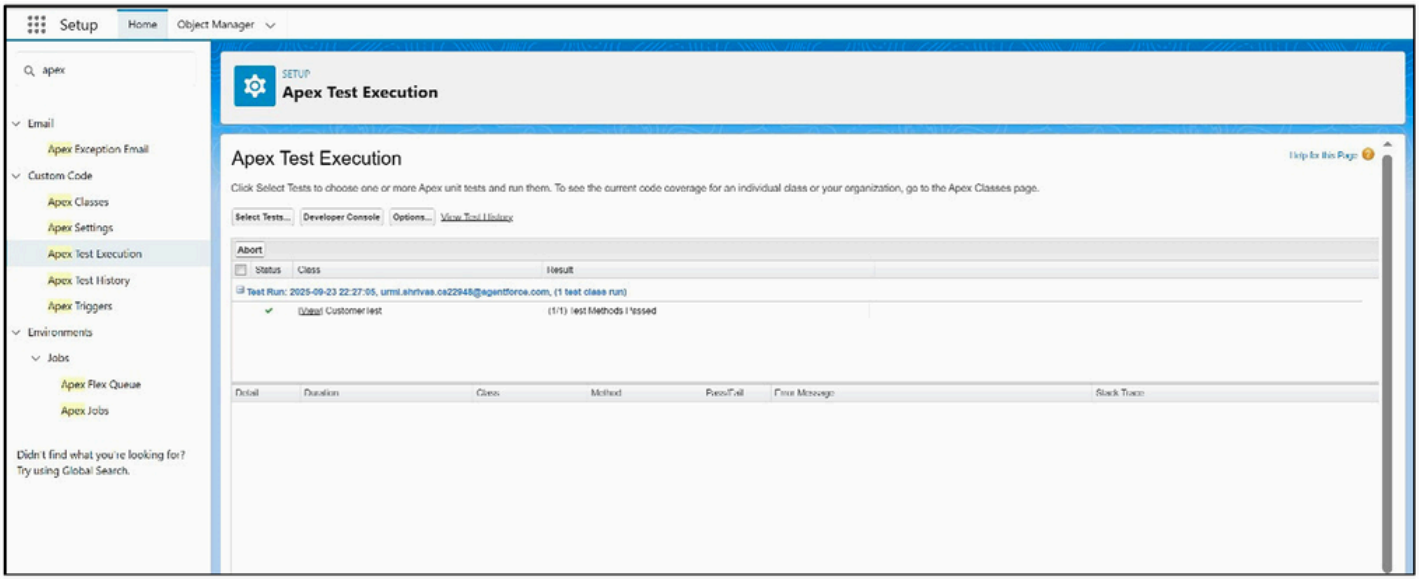



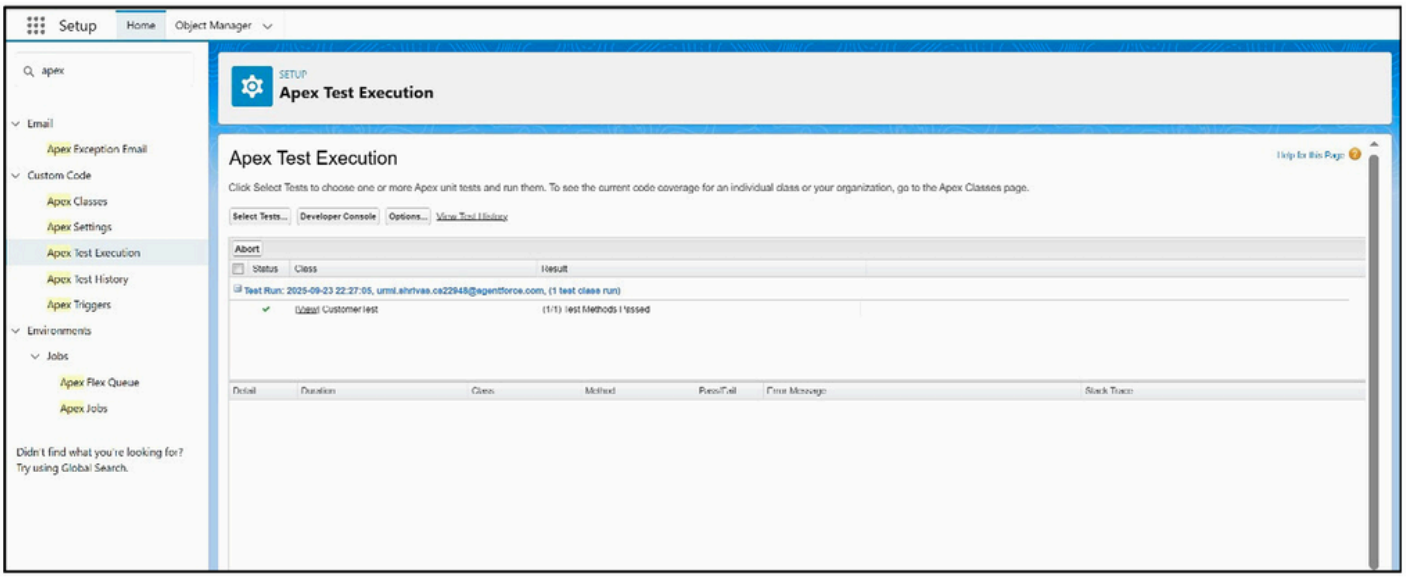

**2. Apex Triggers (before/after insert/update/delete)**

Created CustomerTrigger on Customer__c (before insert/update).

Validated Name (default if blank) and Age__c (default if null/negative).
Set Status__c based on age: Adult or Minor.
Created CustomerTriggerTest class with test records to verify logic.
Ran the test via **Apex Test Execution** and checked debug logs.
**Outcome:** Trigger works correctly; validations and status assignment are verified.






**3. Trigger Design Pattern**

Moved trigger logic to a handler class CustomerTriggerHandler.

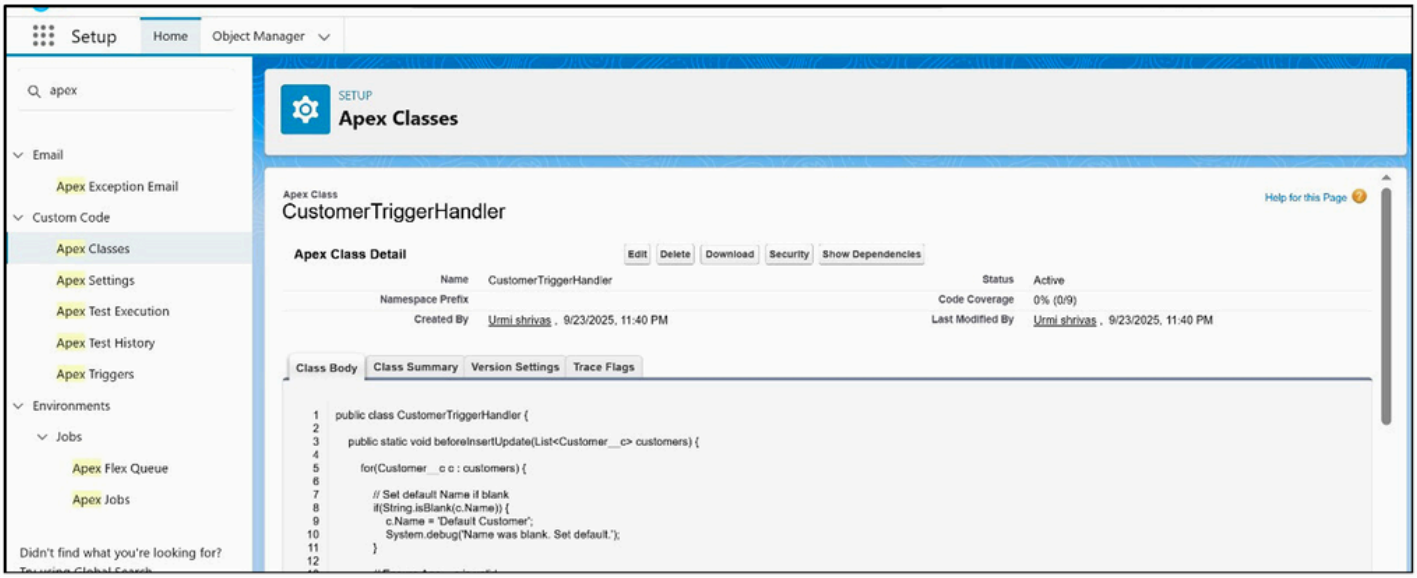
beforeInsertUpdate(List<Customer__c> customers) handles validations and status assignment.

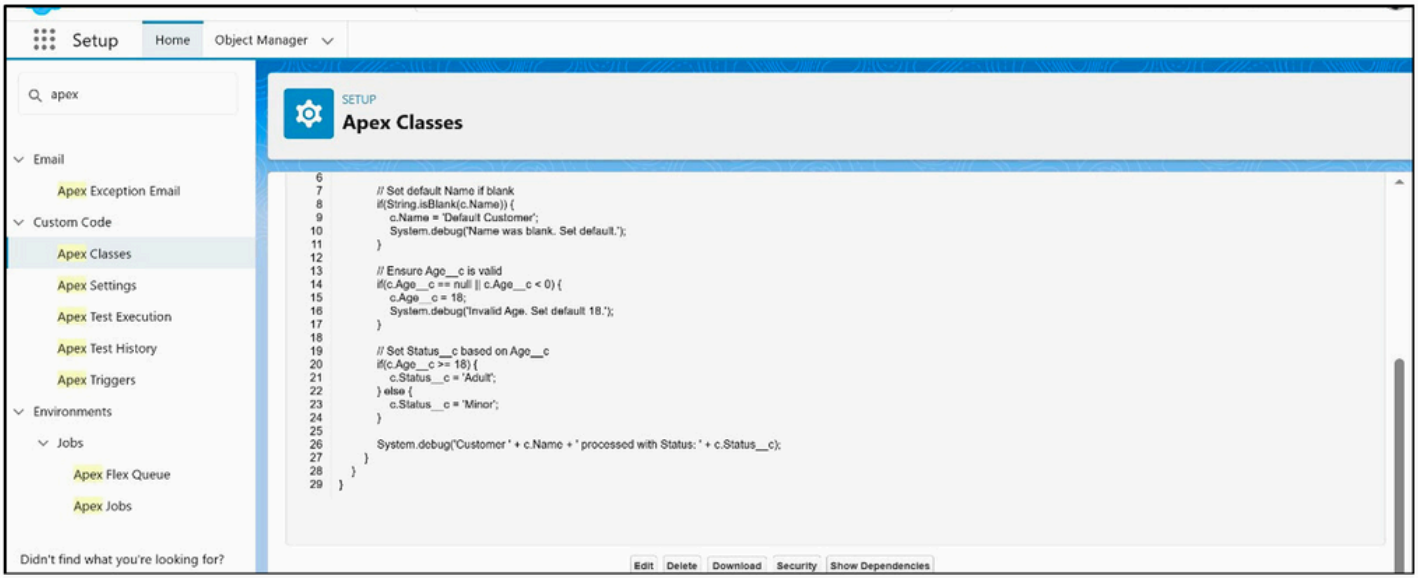
Updated trigger (CustomerTriggerHandlerTrigger) to call the handler class, making it clean and maintainable.

Created a test class CustomerTriggerHandlerTest to verify handler logic.

Ran test via **Apex Test Execution** → debug logs confirmed correct execution.

**Outcome:** Trigger logic is now organized, reusable, and follows Salesforce best practices.






**4. Apex Programming – SOQL & SOSL**

Created Apex Class CustomerQueryExample to implement:

- **SOQL query** → retrieve adult customers (Age__c ≥ 18).
- **SOSL search** → search customer records by name (Customer_Name__c).

  Created test class CustomerQueryExampleTest to:
  Insert sample Customer__c records.
  Call the main class methods for testing.
  Verified functionality using **Apex Test Execution**.
  **Outcome:** Practiced retrieving and searching records in Apex; learned proper test class creation and query execution.


Setup
Home
Object Manager
apex
Email
Apex Exception Email
Custom Code
Apex Classes
Apex Settings
Apex Test Execution
Apex Test History
Apex Triggers
Environments
Jobs
Apex Flex Queue
Apex Jobs
Didn't find what you're looking for?
SETUP
Apex Classes
Class Body
Class Summary
Version Settings
Trace Flags
public class CustomerQueryExample {
// Method using SOQL
public static void queryCustomers() {
List<Customer__c> adults = [SELECT Customer_Name__c, Age__c, Status__c
FROM Customer__c
WHERE Age__c >= 18
ORDER BY Customer_Name__c];
System.debug('Adult Customers: ' + adults);
}
// Method using SOSL
public static void searchCustomers(String searchText) {
List<List<SObject>> searchResults = [FIND :searchText
IN ALL FIELDS
RETURNING Customer__c(Customer_Name__c, Age__c, Status__c)];
System.debug('Search Results: ' + searchResults);
}
}


Setup
Home
Object Manager
apex
Email
Apex Exception Email
Custom Code
Apex Classes
Apex Settings
Apex Test Execution
Apex Test History
Apex Triggers
Environments
Jobs
Apex Flex Queue
Apex Jobs
Didn't find what you're looking for?
SETUP
Apex Classes
Class Body
Class Summary
Version Settings
Trace Flags
public class CustomerQueryExample {
// Method using SOQL
public static void queryCustomers() {
List<Customer__c> adults = [SELECT Customer_Name__c, Age__c, Status__c
FROM Customer__c
WHERE Age__c >= 18
ORDER BY Customer_Name__c];
System.debug('Adult Customers: ' + adults);
}
// Method using SOSL
public static void searchCustomers(String searchText) {
List<List<SObject>> searchResults = [FIND :searchText
IN ALL FIELDS
RETURNING Customer__c(Customer_Name__c, Age__c, Status__c)];
System.debug('Search Results: ' + searchResults);
}
}


Setup
Home
Object Manager
apex
Email
Apex Exception Email
Custom Code
Apex Classes
Apex Settings
Apex Test Execution
Apex Test History
Apex Triggers
Environments
Jobs
Apex Flex Queue
Apex Jobs
Didn't find what you're looking for?
SETUP
Apex Classes
Class Body
Class Summary
Version Settings
Trace Flags
public class CustomerQueryExample {
// Method using SOQL
public static void queryCustomers() {
List<Customer__c> adults = [SELECT Customer_Name__c, Age__c, Status__c
FROM Customer__c
WHERE Age__c >= 18
ORDER BY Customer_Name__c];
System.debug('Adult Customers: ' + adults);
}
// Method using SOSL
public static void searchCustomers(String searchText) {
List<List<SObject>> searchResults = [FIND :searchText
IN ALL FIELDS
RETURNING Customer__c(Customer_Name__c, Age__c, Status__c)];
System.debug('Search Results: ' + searchResults);
}
}



### 5. Collections (List, Set, Map)

Created Apex Class CustomerCollectionsExample to demonstrate:

- **List** → stored all Customer__c records.
- **Set** → stored unique customer names from the list.
- **Map** → stored Customer__c records with Id as key for quick access.

  Created test class CustomerCollectionsExampleTest to test and verify via debug logs.

  **Outcome:** Learned handling duplicates, ordering, and key-value pairs in Apex collections.

### 6. Control Statements

Created Apex Class CustomerControlExample to demonstrate:

- **If-Else** → check if customer is Adult or Minor.
- **For Loop** → iterate over all Customer__c records.
- **While Loop** → process customers until all are handled.

- **Switch Statement** → handle different Status__c values.
  Created test class CustomerControlExampleTest to verify logic.
  **Outcome:** Practiced using conditional logic and loops in Apex and controlling code flow based on record data.

**7. Batch Apex**

Created Batch Apex Class CustomerBatchExample to:
Process all Customer__c records in batches.
Update Status__c based on Age__c (Adult or Minor).
Implemented start(), execute(), and finish() methods.
Created CustomerBatchExampleTest to test execution using Database.executeBatch().
**Outcome:** Practiced processing large record sets asynchronously using Batch Apex.

**8. Queueable Apex**

Created CustomerQueueableExample to process Customer__c records asynchronously and update Status__c.
Created test class CustomerQueueableExampleTest to execute using System.enqueueJob.
**Outcome:** Learned background job execution and asynchronous processing with Queueable Apex.

**9. Scheduled Apex**

Created CustomerScheduledExample to automatically update Status__c of customers based on Age__c.
Scheduled job using a CRON expression.
Created CustomerScheduledExampleTest to verify scheduled execution in test context.
**Outcome:** Practiced automating tasks with Scheduled Apex and scheduling jobs programmatically.

**10. Future Methods**

Created CustomerFutureExample to update Status__c asynchronously.
Created CustomerFutureExampleTest to test in asynchronous context.
**Outcome:** Practiced background updates without blocking users using Future Methods.

**11. Exception Handling**

Created CustomerExceptionExample to handle invalid or null Age__c values and update Status__c accordingly.
Created CustomerExceptionExampleTest to insert valid and invalid records and test exception handling.
**Outcome:** Learned to handle runtime errors effectively without breaking execution flow.

**12. Test Classes**

Created separate test classes for each Apex class and trigger.
Inserted valid, invalid, and edge-case data.
Used Test.startTest() / Test.stopTest() for asynchronous testing.
Used System.assert, **Apex Test Execution**, and **Debug Logs** for verification.

Ensured 75%+ code coverage for deployment.
**Outcome:** Learned to write effective unit tests for both synchronous and asynchronous logic, verifying safe execution of Apex code.

**13. Asynchronous Processing Summary**

- **Batch Apex:** Processes large records in chunks using Database.executeBatch.
- **Queueable Apex:** Runs background jobs using System.enqueueJob.
- **Scheduled Apex:** Schedules jobs with CRON expressions.
- **Future Methods:** Runs operations asynchronously without blocking users.
  **Outcome:** Gained expertise in using asynchronous methods for background processing and performance improvement.

---

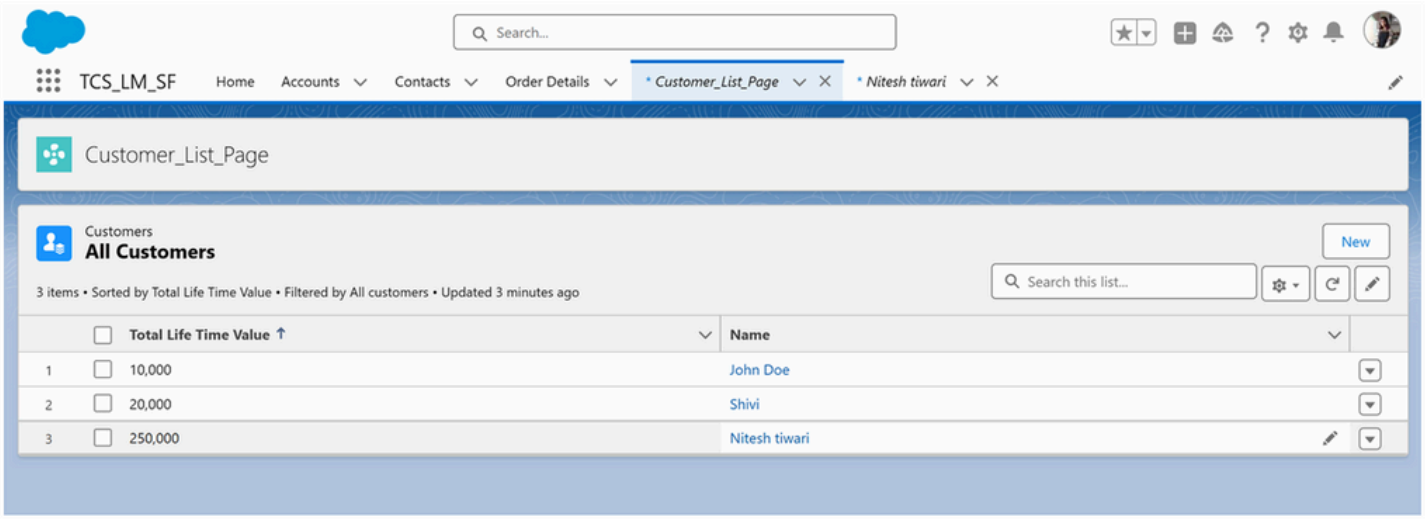
# Phase 6: User Interface Development

## 1. Lightning App Builder

Objective:
To create a Lightning App Page that displays all Customer__c records in a user-friendly layout.

Steps Implemented:

1. Navigated to Setup → Lightning App Builder → New → App Page
2. Page Name: Customer_List_Page
3. Layout: One Region (simple view)
4. Components Added:
   - List View: Object → Customer__c, List View → “All”
   - Report Chart (optional): Linked to “Sample Flow Report: Screen Flows”
5. Save & Activate: Assigned to Org Default for all users
6. Records Added:
   - Created 5–6 sample Customer__c records
   - Filled fields: Name, Customer Status Type, Total Lifetime Value, Age__c



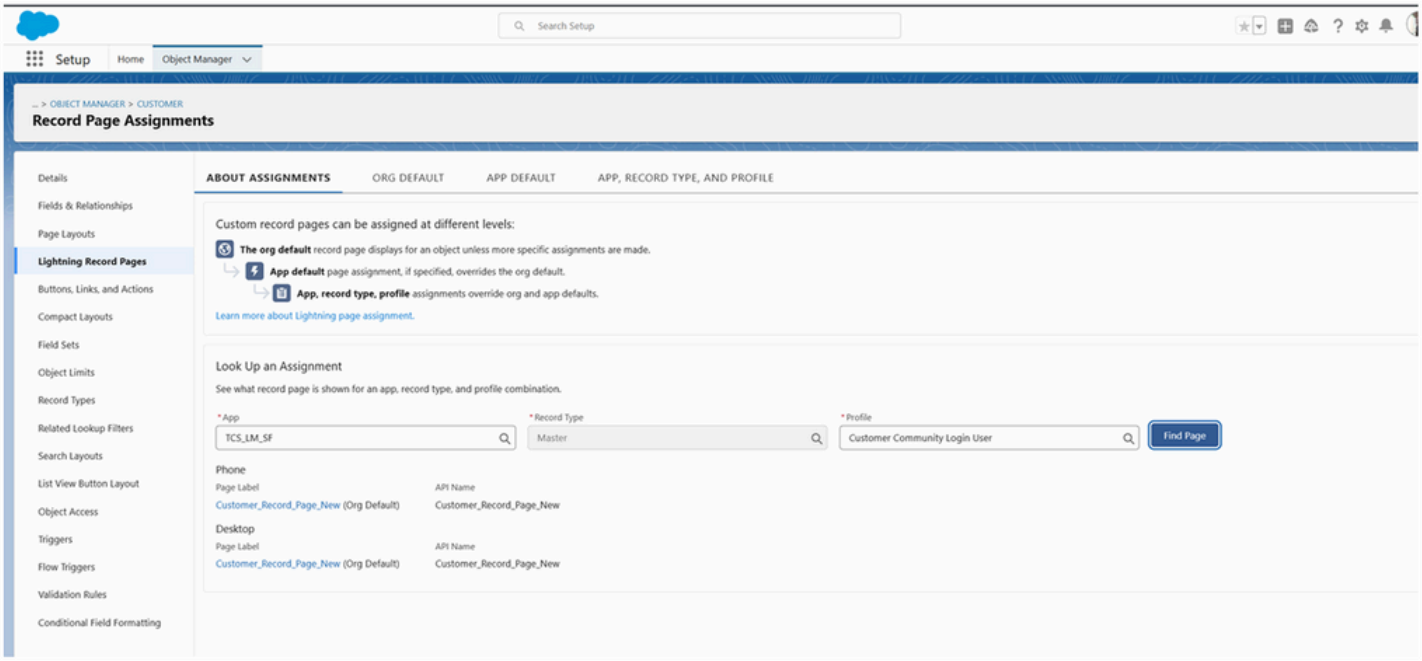

Outcome:
App Page now displays the Customer List and Report Chart.
Users can view records and monitor metrics directly from the app page.

## 2. Record Type

Objective:
To define record types for Customer__c based on different customer categories (e.g., Individual, Business).

Steps Implemented:

1. Setup → Object Manager → Customer__c → Record Types → New
2. Created record types such as Individual Customer and Business Customer
3. Assigned different Page Layouts and Picklist Values per record type.
4. Activated record types for relevant profiles.




Outcome:
Record types now allow users to manage different kinds of customers with custom fields and layouts, improving data accuracy and relevance.

## 3. Tabs

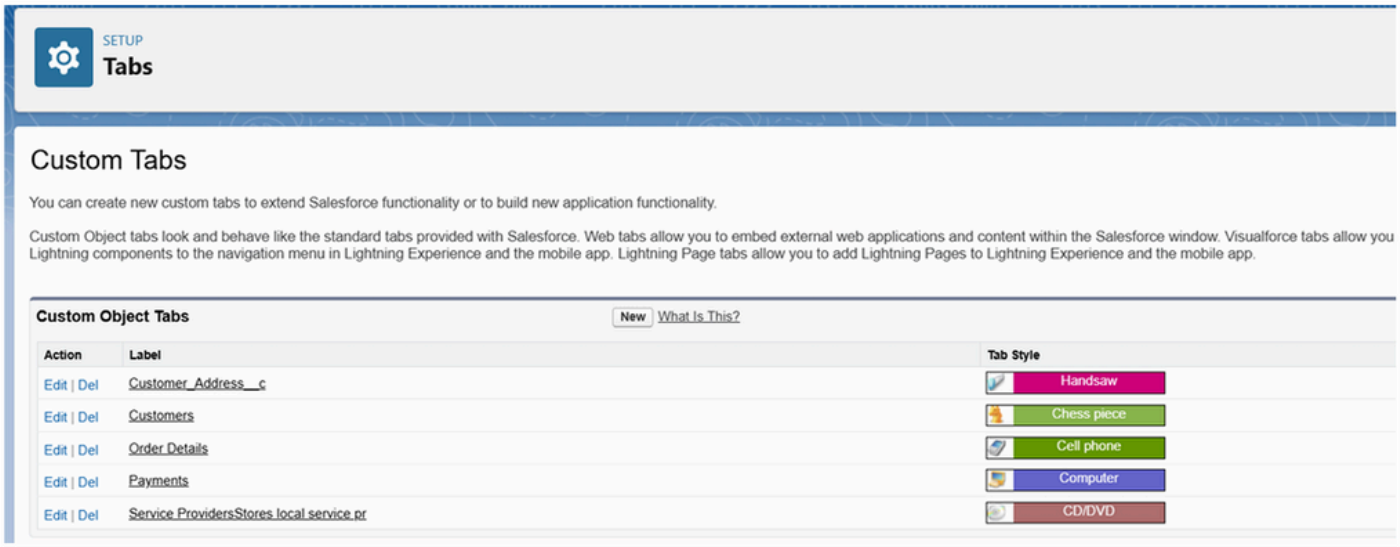
Objective:
To organize multiple sections on a record or app page for better navigation and user experience.

Steps Implemented:

1. Lightning App Builder → Open Record Page → Add Tabs component
2. Added tabs for:
    - Details: Record detail view
    - Related Lists: Linked records such as Service Requests or Payments
    - Reports/Charts: Analytical insights

Save & Activate:

- Page saved as Customer_Record_Page_New
- Assigned as Org Default




Outcome:

Users can easily switch between sections without scrolling.

This improves page organization and usability.

## 4. Home Page Layouts

Objective:

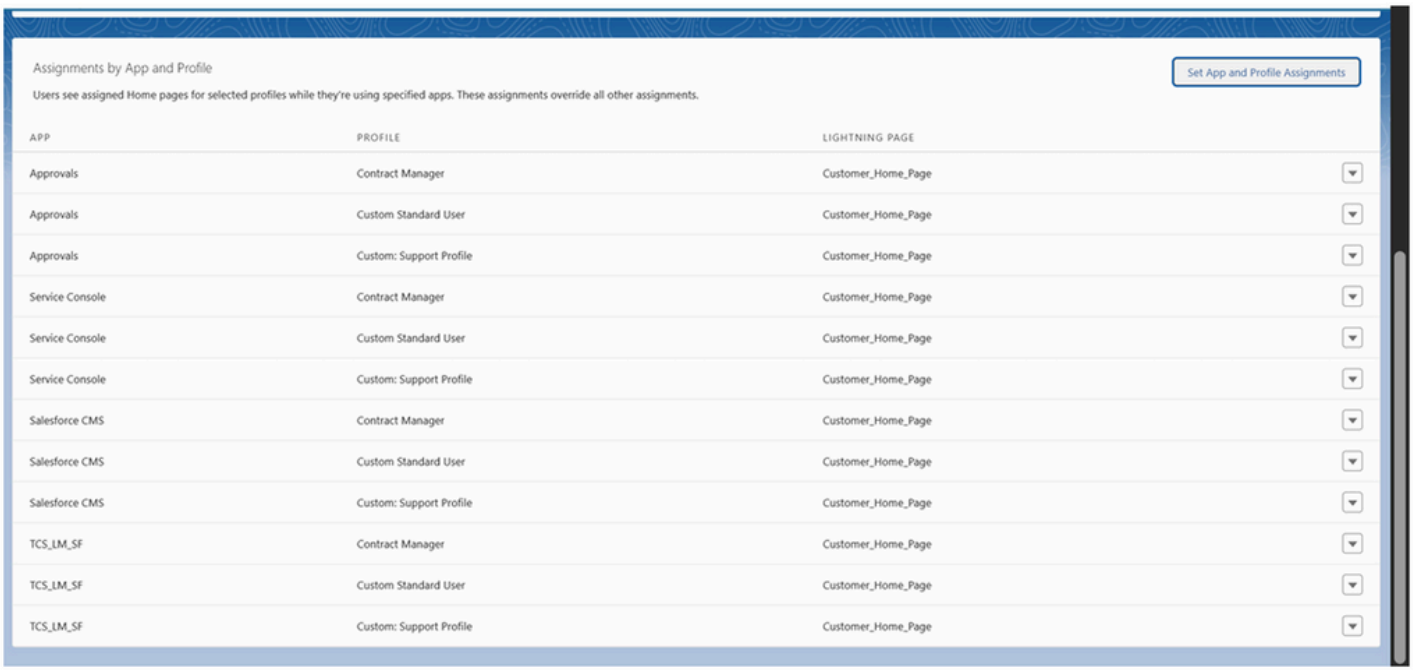
To create a customized Home Page that displays key information for CRM users.

Steps Implemented:

1. Setup → Lightning App Builder → New → Home Page
2. Page Name: Customer_Home_Page
3. Layout: Standard / One Region
4. Components Added:
   - Report Charts → Display key metrics
   - List Views → Quick access to records
   - Rich Text / Image Components → Display announcements or instructions
   - Tabs (optional) → For better organization

Save & Activate:

- Assigned as Org Default or specific to App Profiles



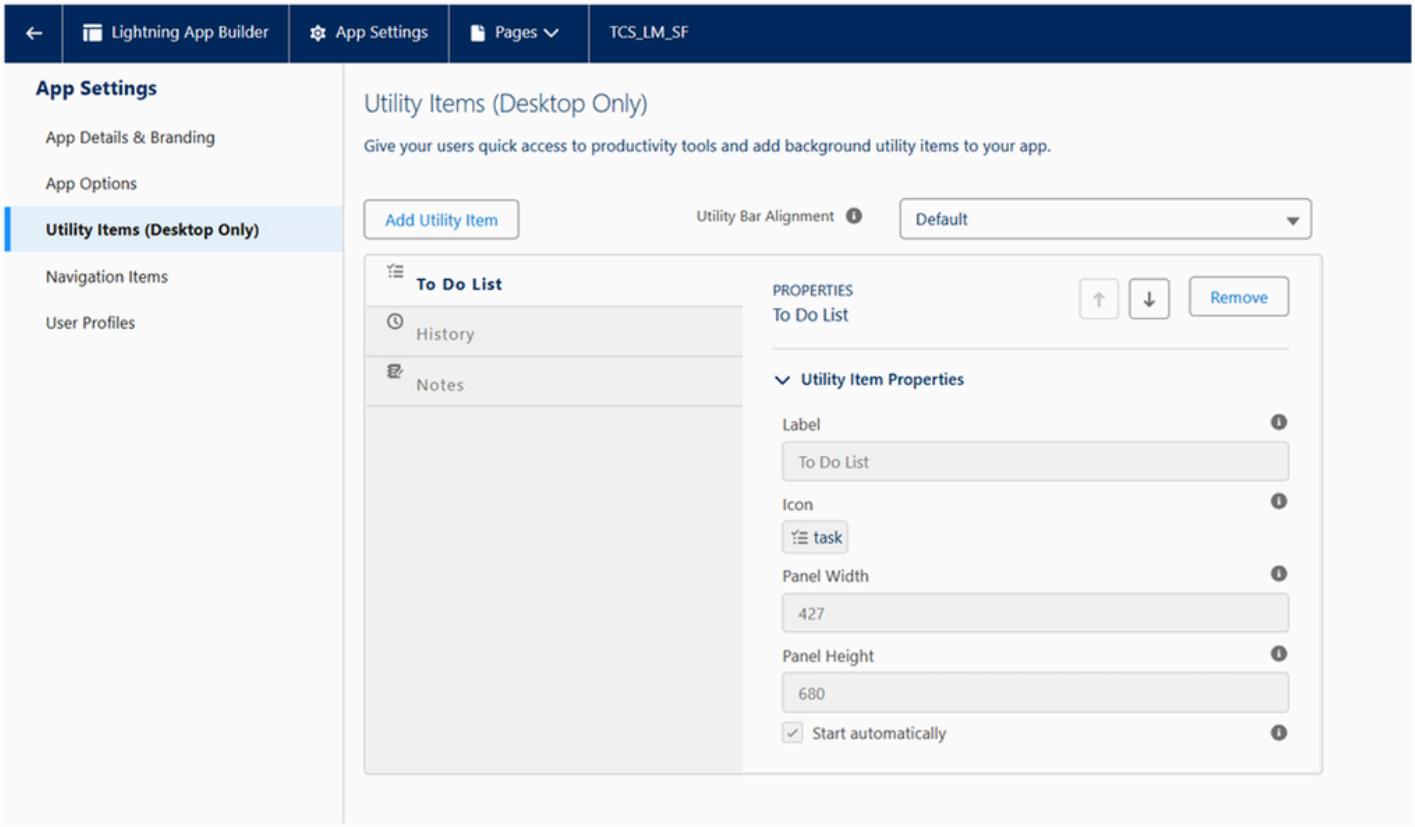

Outcome:
The Home Page displays dashboards, key metrics, and record lists, allowing users to quickly access important data from a single interface.

## 5. Utility Bar

Objective:
To enhance user productivity by adding quick-access tools in the TCS_LM_SF App for the Local Business Service CRM.

Steps Implemented:

1. Edit App: Setup → App Manager → TCS_LM_SF → Edit
2. Details Updated (optional):
   - Name: *Customer CRM App*
   - Description: *Manage customers, tasks, and service requests*
3. Navigation & Tabs:
   - Console navigation enabled
   - Tabs added: Customers, Service Requests, Reports/Dashboard
4. Utility Bar:
   Added utilities such as:
   - Notes
   - Task List / Jobs
   - Customer History
   - Dashboard
   - Calculator
5. Set Utility Properties: Label, Icon, Panel Size, and Behavior
6. Save & Activate: Assigned as App Default
7. Verification: Opened app to confirm all utilities were functional



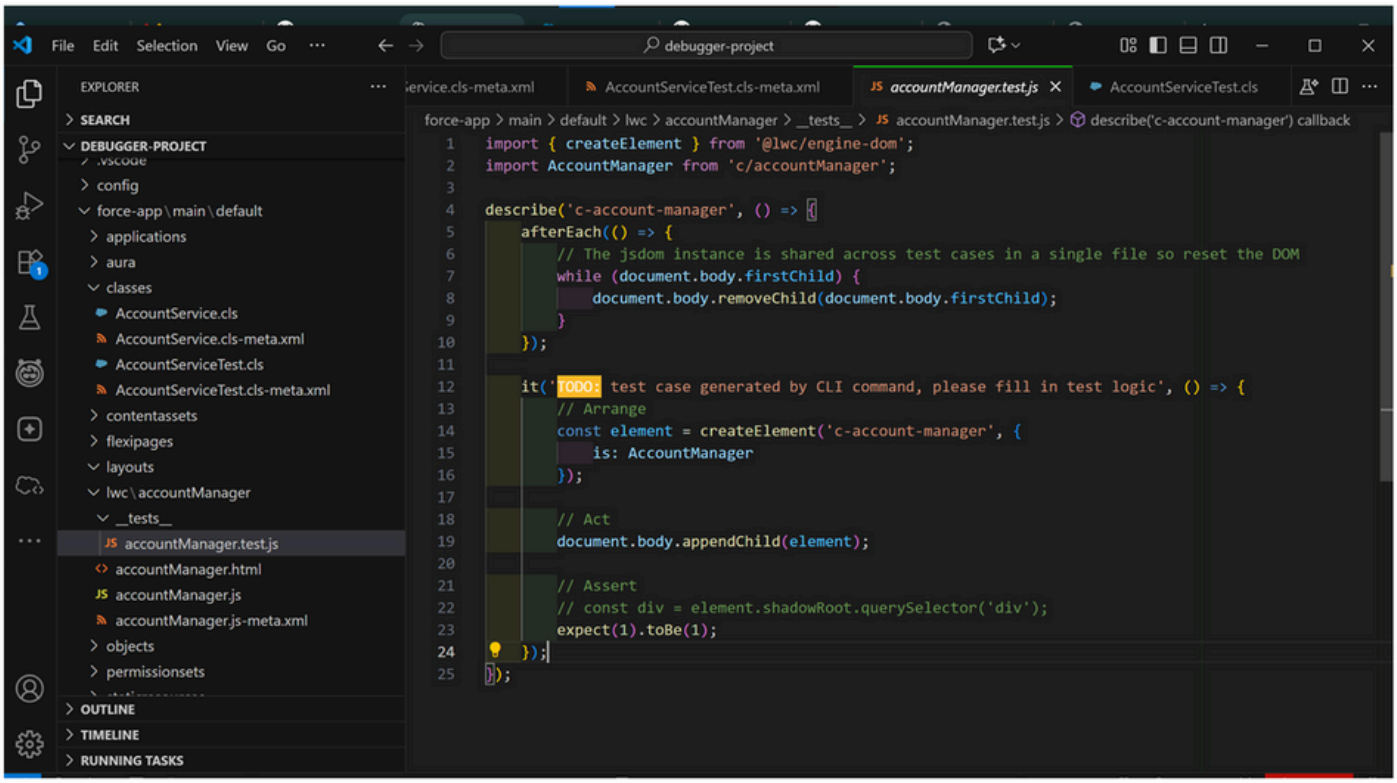

Outcome:
Users can now access key tools like tasks, history, and dashboards directly from the Utility Bar, improving workflow efficiency.

## 6. Lightning Web Component (LWC)

Purpose:
To provide a modern, responsive interface for managing customer accounts — including creating, updating, and fetching account details.

Implementation Details:

- JavaScript (JS): Handles user input and calls Apex methods via @AuraEnabled.
- HTML: Uses Lightning input fields for Name, Type, Email, and Phone with buttons for Create, Update, and Fetch actions.
- Meta.xml: Configured to be used on Record Page, App Page, and Home Page via Lightning App Builder.



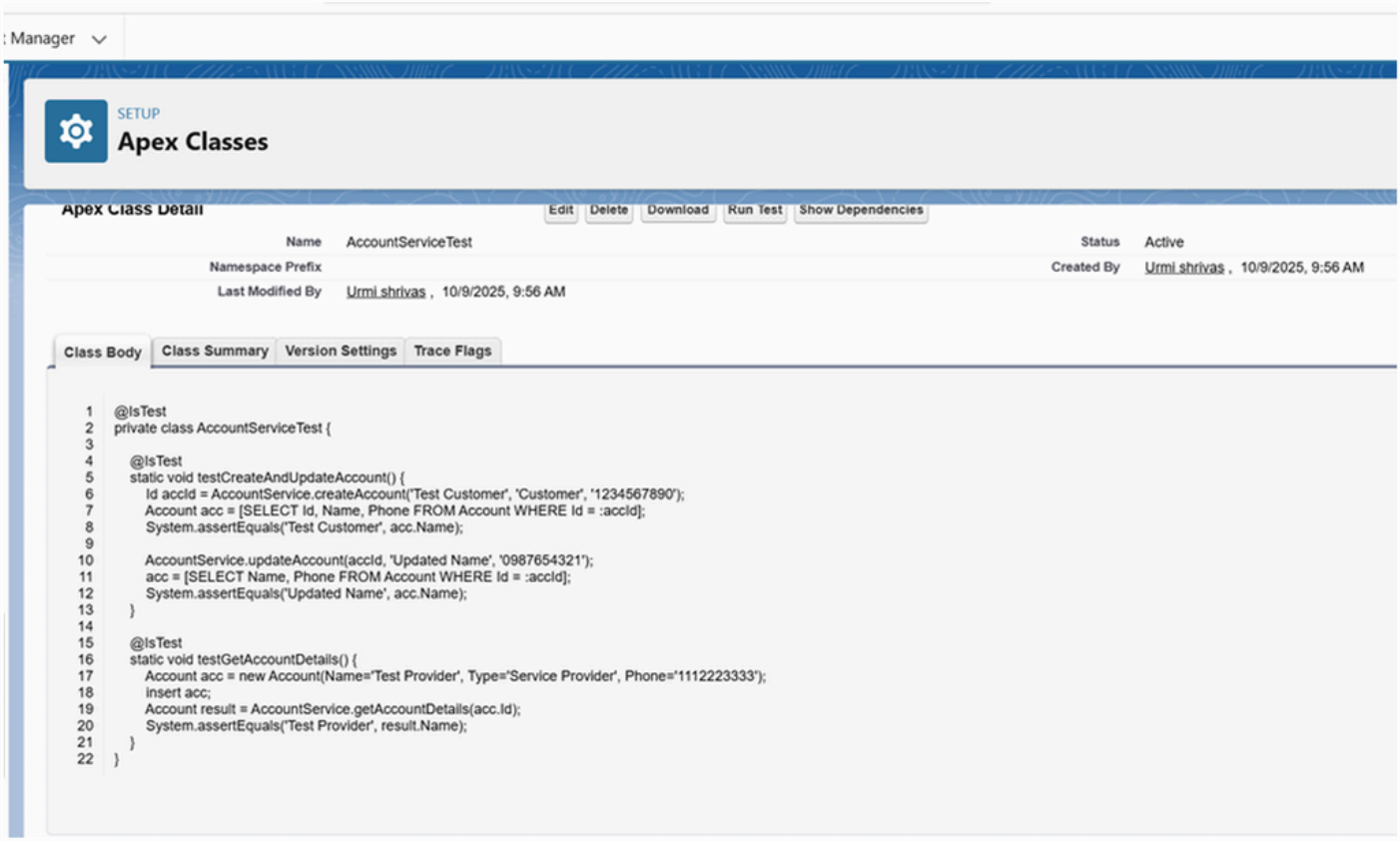

Outcome:
LWC enables a seamless and interactive experience for users to manage Accounts directly from the Salesforce UI.

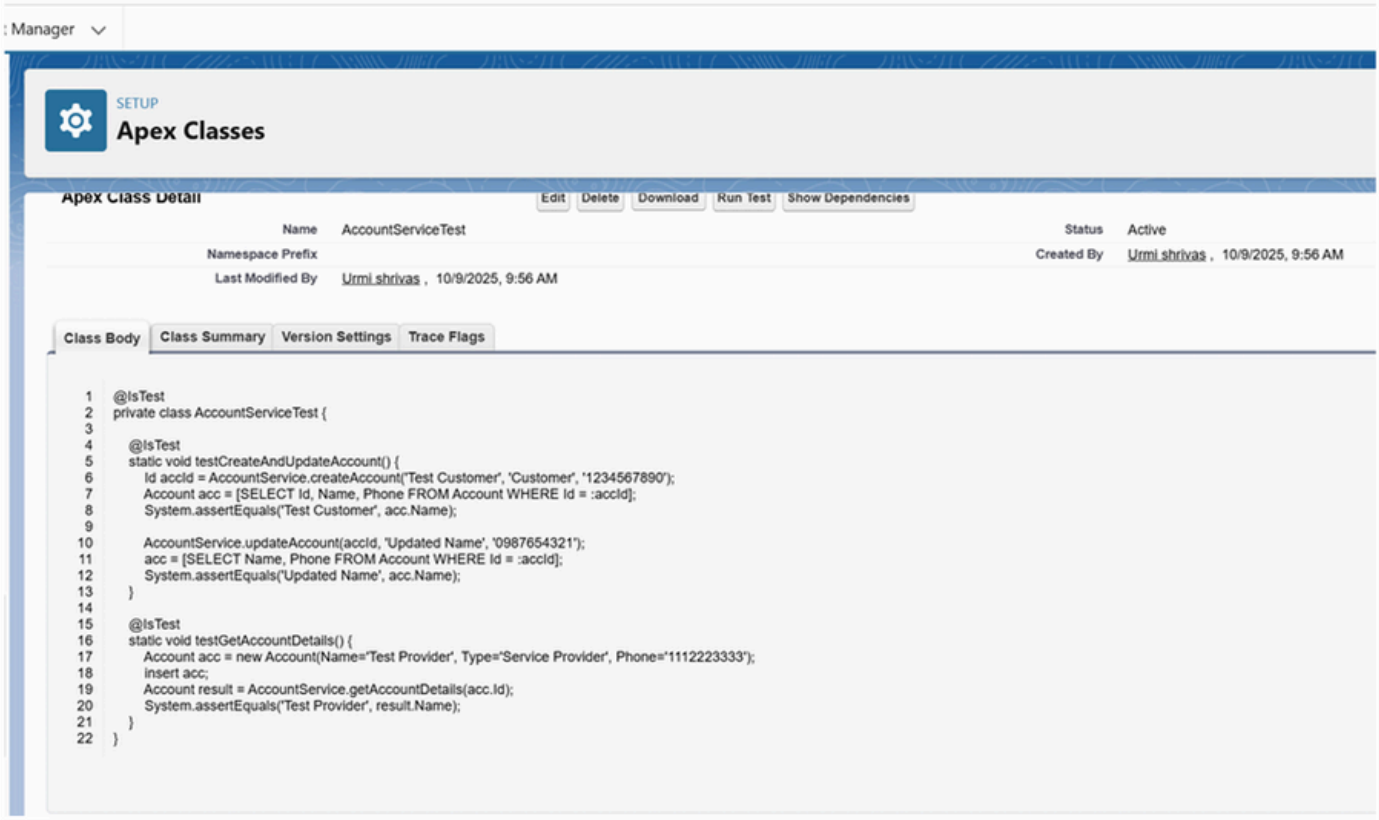
## 7. Apex with LWC

Purpose:
To manage backend logic for Account operations — create, update, and fetch — used by the Lightning Web Component.

Implementation Details:

- Apex Class: AccountService
- Methods:
  - createAccount(name, type, phone) → Creates new Account record
  - updateAccount(accountId, name, phone) → Updates existing Account
  - getAccountDetails(accountId) → Fetches Account details
- All methods are annotated with @AuraEnabled for LWC access.




Outcome:

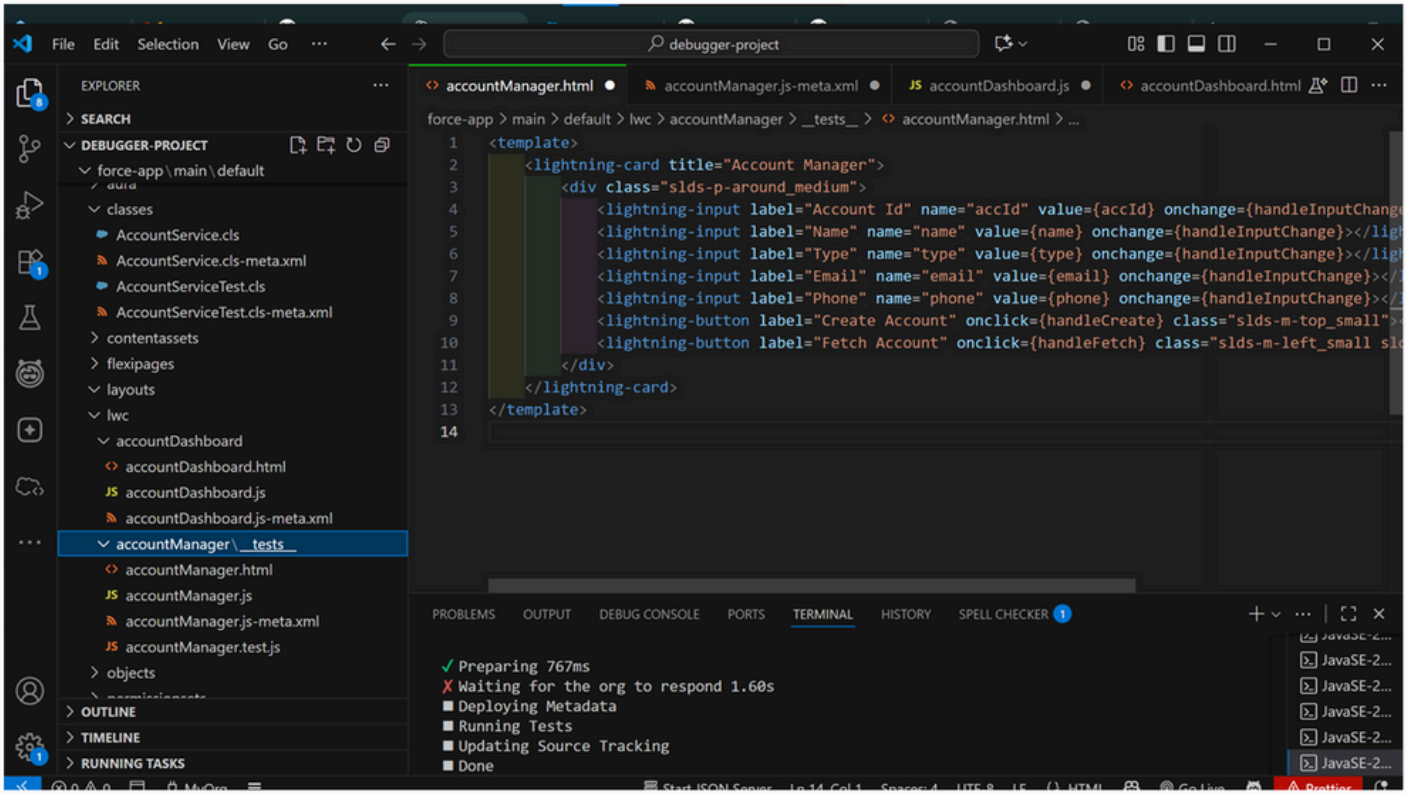
Established a clear separation between frontend (LWC) and backend (Apex), ensuring secure, maintainable, and efficient communication.

## 8. Events in LWC

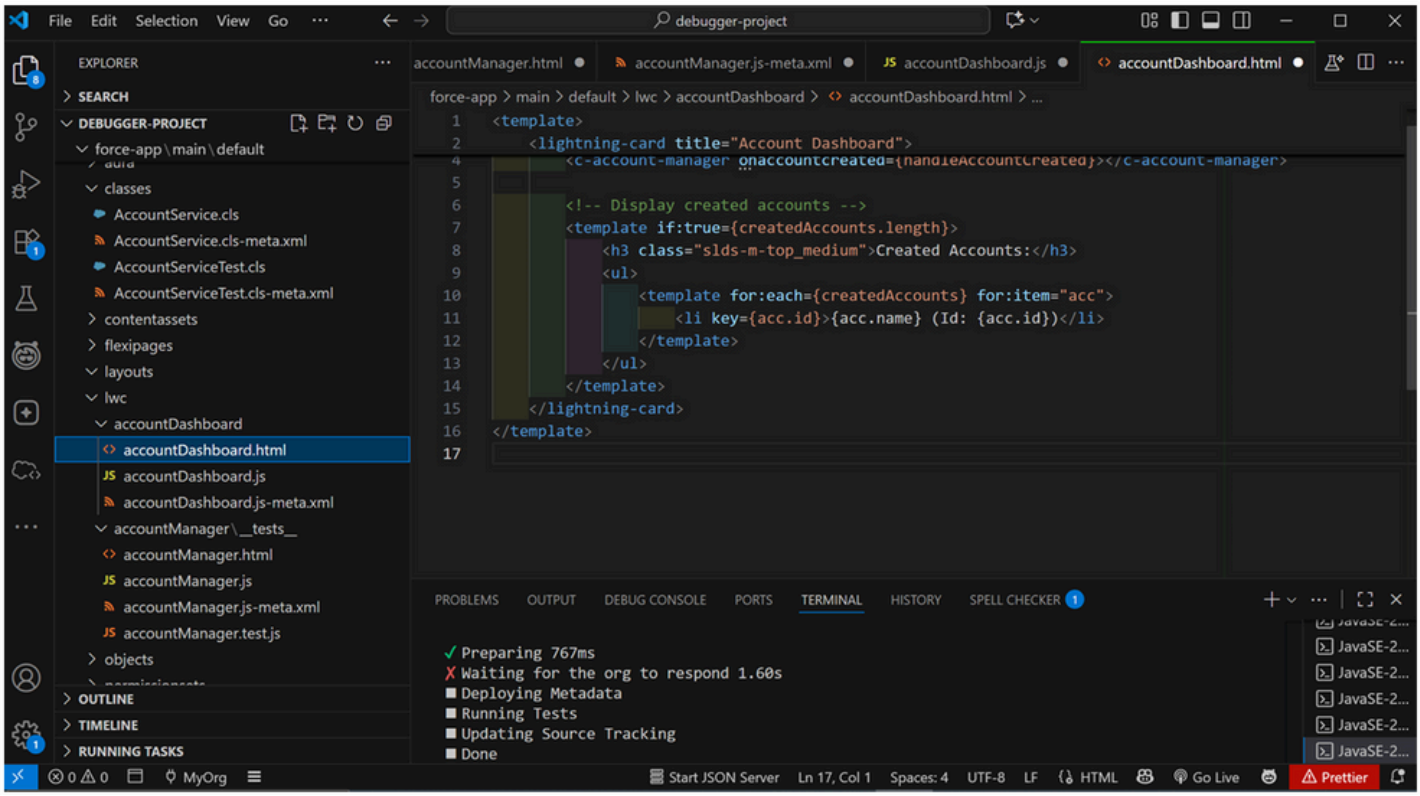
Purpose:
To demonstrate component communication (child-to-parent interaction).
When a new account is created in the child component, it notifies the parent component automatically.

Implementation Details:

- Child Component (accountManager):
  - Handles inputs (Name, Type, Phone).
  - Calls createAccount() from Apex.
  - Dispatches a CustomEvent (accountcreated) with account data.




- Parent Component (accountDashboard):
  - Listens for the accountcreated event.
  - Updates and displays the newly created account dynamically.



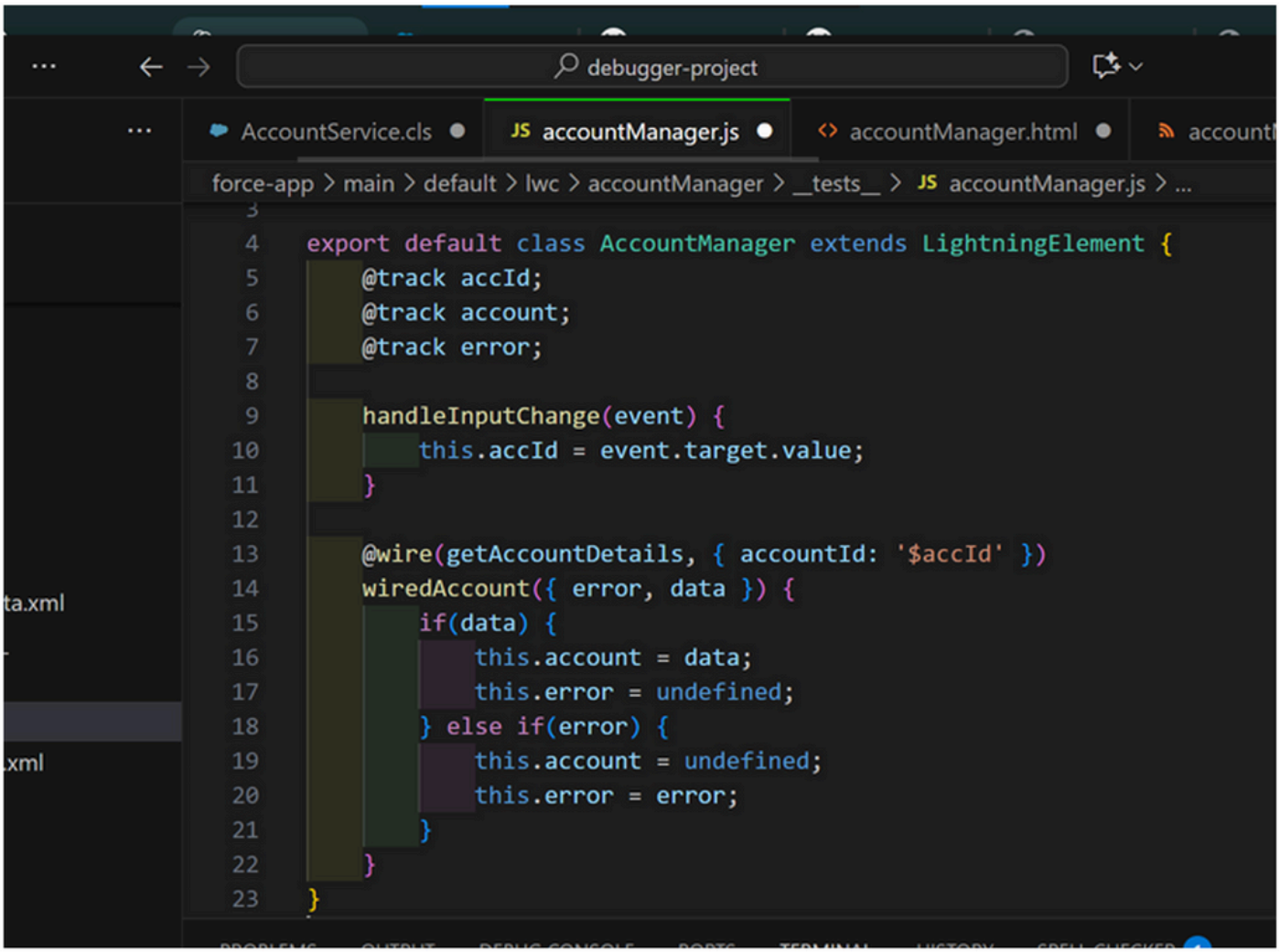

Outcome:
Enabled real-time communication between components, improving dynamic updates and interactivity in the LWC interface.

## 9. Wire Adapters in LWC

Purpose:
To fetch Salesforce data reactively without explicit Apex method calls.

Implementation:

- Used @wire adapter to automatically fetch Account details when a user enters an Account Id.
- Demonstrated how data updates automatically when the record changes.




Outcome:
Improved performance and user experience through reactive data binding in LWC.

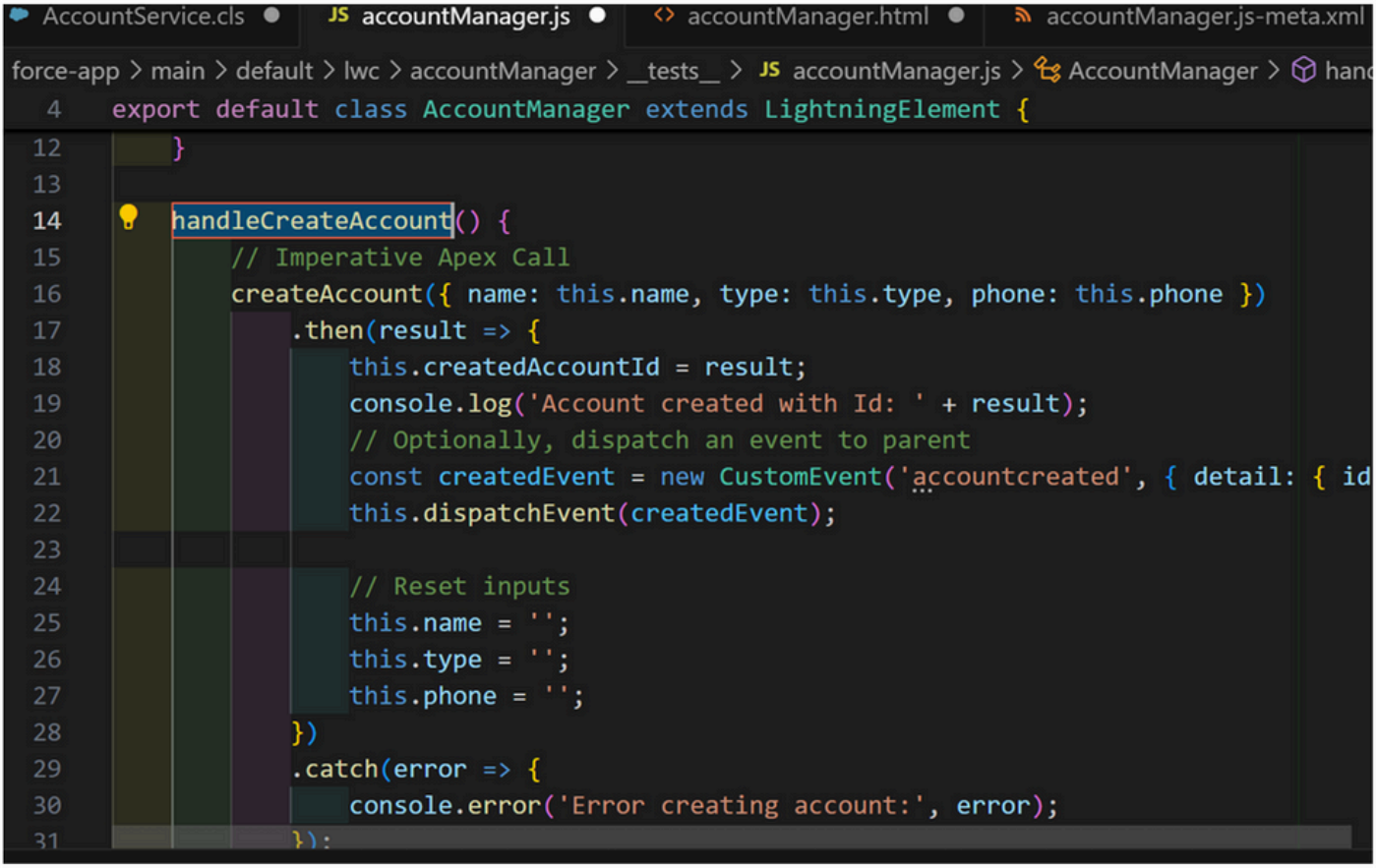
## 10. Imperative Apex Calls

Purpose:
To manually call Apex methods from JavaScript when user actions occur.

Implementation:

- Used imperative calls to trigger createAccount() method only when the user clicks a button.

Outcome:
Provided developers full control over when and how Apex logic executes, allowing custom, event-based automation within LWC.




---

# Phase 7: Integration & External Access

*(This project focused primarily on internal Salesforce CRM operations; external integrations were studied conceptually but not implemented.)*

**1. Named Credentials**

- **Definition:** Securely stores external system URLs and authentication details.
- **Purpose:** Enables Apex or Flow to call external services without hardcoding credentials.
- **Usage in Project:** *Not implemented in this project.*

**2. External Services**

- **Definition:** Allows Salesforce to connect to external REST APIs declaratively using Flow or Apex.
- **Purpose:** Automatically generates Apex actions from an OpenAPI specification for easy integration.
- **Usage in Project:** *Not used in this project.*

**3. Web Services (REST / SOAP)**

- **Definition:** Enables Salesforce to expose its own REST/SOAP APIs or consume external APIs.
- **Purpose:** Facilitates data exchange between Salesforce and other systems.
- **Usage in Project:** *Not implemented in this project*.

**4. Callouts**

- **Definition:** Apex code or Flows can make HTTP requests (GET, POST, PUT, DELETE) to external APIs.
- **Purpose:** Used to fetch or send data to external systems.
- **Usage in Project:** *Not used in this project*.

**5. Platform Events**

- **Definition:** Event-driven messaging system in Salesforce.
- **Purpose:** Allows asynchronous notifications between Salesforce and external systems when data changes.
- **Usage in Project:** *Not implemented in this project*.

**6. Change Data Capture (CDC)**

- **Definition:** Captures record changes in real-time and publishes them as events.
- **Purpose:** Enables external systems or Salesforce components to react instantly to data updates.
- **Usage in Project:** *Not used in this project*.

**7. Salesforce Connect**

- **Definition:** Provides real-time access to external data without importing it into Salesforce.
- **Purpose:** Connects external databases or services using OData protocol.
- **Usage in Project:** *Not implemented in this project*.

**8. API Limits**

- **Definition:** Salesforce enforces limits on API requests to ensure system performance and security.
- **Purpose:** Prevents excessive or abusive API usage.
- **Usage in Project:** *Not applicable*.

**9. OAuth & Authentication**

- **Definition:** Industry-standard protocols for authenticating users and authorizing system access.
- **Purpose:** Ensures secure access to Salesforce APIs and external systems.
- **Usage in Project:** *Not required*.

**10. Remote Site Settings**

- **Definition:** Configuration that authorizes Salesforce to make HTTP callouts to specific external URLs.
- **Purpose:** Adds a security layer by controlling which URLs Salesforce can access.
- **Usage in Project:** *Not required.*

---

# Phase 8: Data Management & Deployment

This phase focuses on managing data efficiently and ensuring smooth deployment processes in Salesforce. Tools like **Data Import Wizard**, **Data Export**, and **VS Code with Salesforce CLI** were used to maintain data integrity and streamline project operations.

**1. Data Import Wizard**

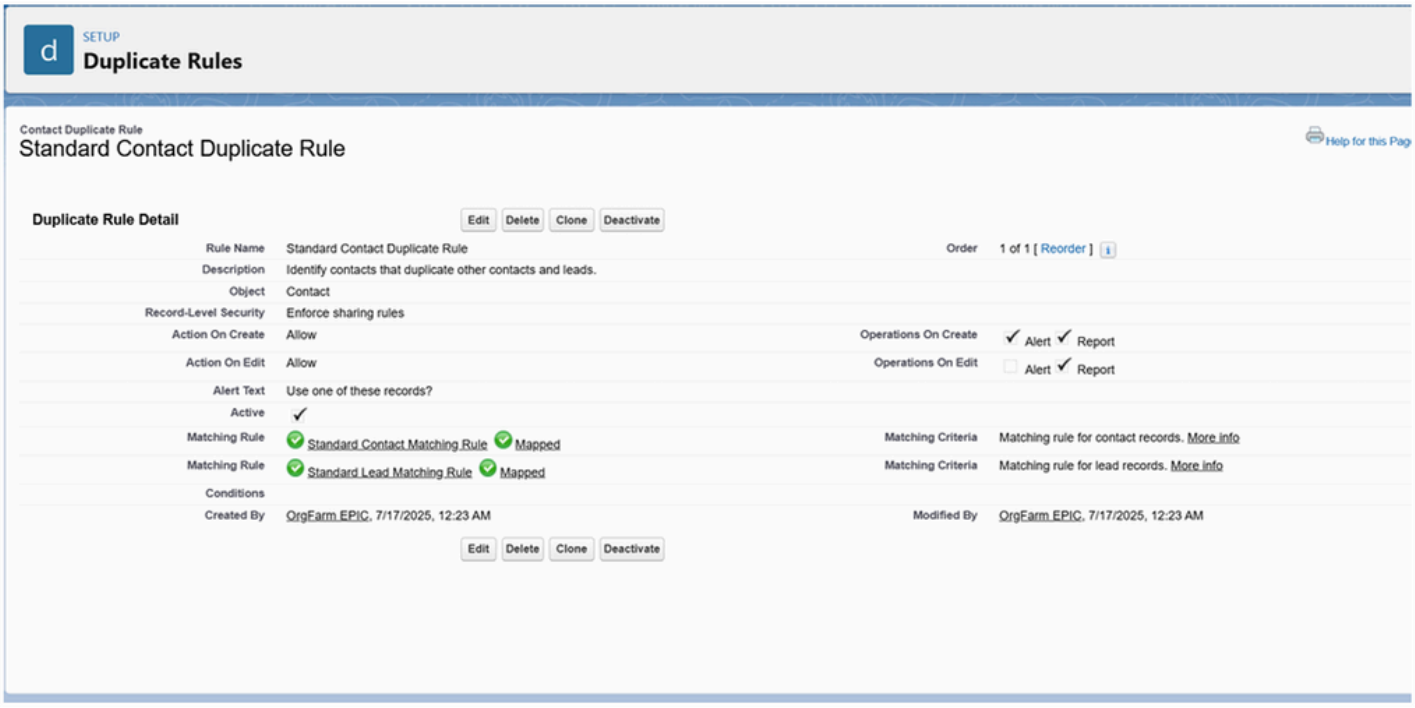
- **Definition:**
  The Data Import Wizard is a built-in Salesforce tool that allows quick and accurate data import.
- **Purpose:**
  To replace manual data entry with fast and reliable bulk imports. It is primarily used for testing, migration, and adding large volumes of records.
- **Usage in Project:**
  - Imported client data from CSV files into Salesforce.
  - Bulk uploaded marketing event leads for testing and record creation.


## 2. Data Loader

- **Definition:**
  Data Loader is a client application used for bulk operations like **insert, update, upsert, delete, and export**.
- **Purpose:**
  Handles large datasets efficiently and supports advanced field mapping and automation.
- **Usage in Project:**
  *Not implemented*, since the dataset was small and all records were handled using the Data Import Wizard.

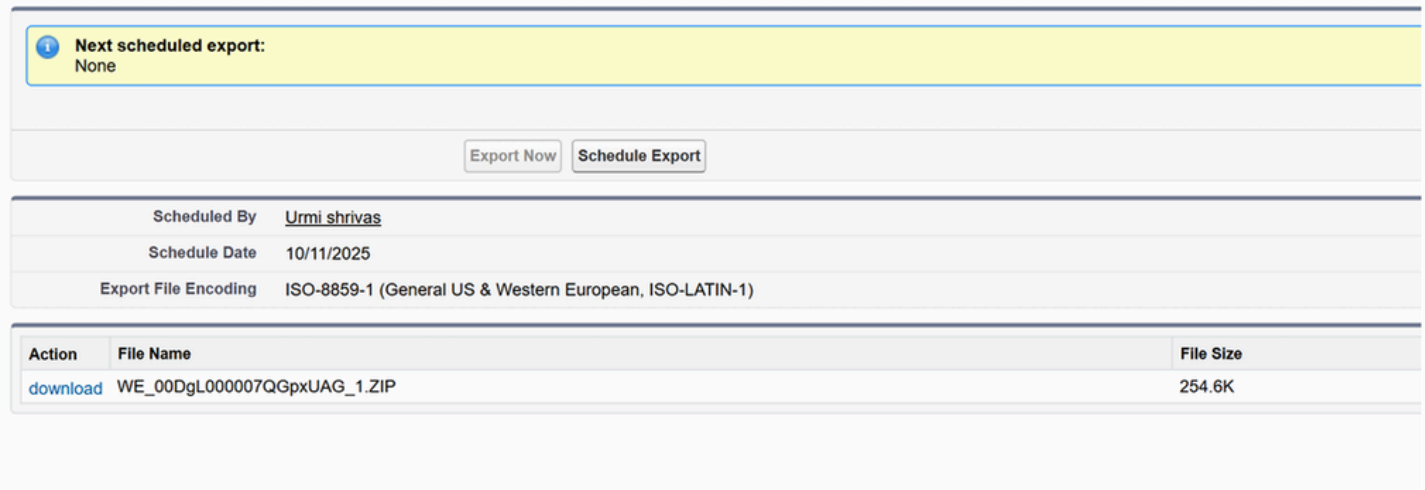
## 3. Duplicate Rules

- **Definition:**
  Salesforce provides standard duplicate management rules to prevent the creation of redundant records.
- **Purpose:**
  The **Standard Contact Duplicate Rule** checks matching criteria such as *Email, First Name,* and *Last Name* to identify duplicates.
- **Usage in Project:**
  The rule was tested to ensure duplicate contacts trigger alerts while still allowing essential data entry when required.




## 4. Data Export & Backup

- **Definition:**
  Salesforce Data Export enables exporting data in CSV format for backup or migration.
- **Purpose:**
  To maintain data integrity and prevent data loss through regular backups.
- **Usage in Project:**
  - Exported data for **Contacts**, **Leads**, and **Accounts** into secure CSV files.
  - Ensured offline backup for data safety and future reference.




## 5. Change Sets

- **Definition:**
  A Change Set is used to deploy metadata (objects, fields, rules, etc.) from one Salesforce

org to another.

- **Purpose:**
  Enables smooth migration between sandbox and production environments without manual recreation.
- **Usage in Project:**
  *Not required*, as all configurations and customizations were developed within a single org.

**6. Unmanaged vs Managed Packages**

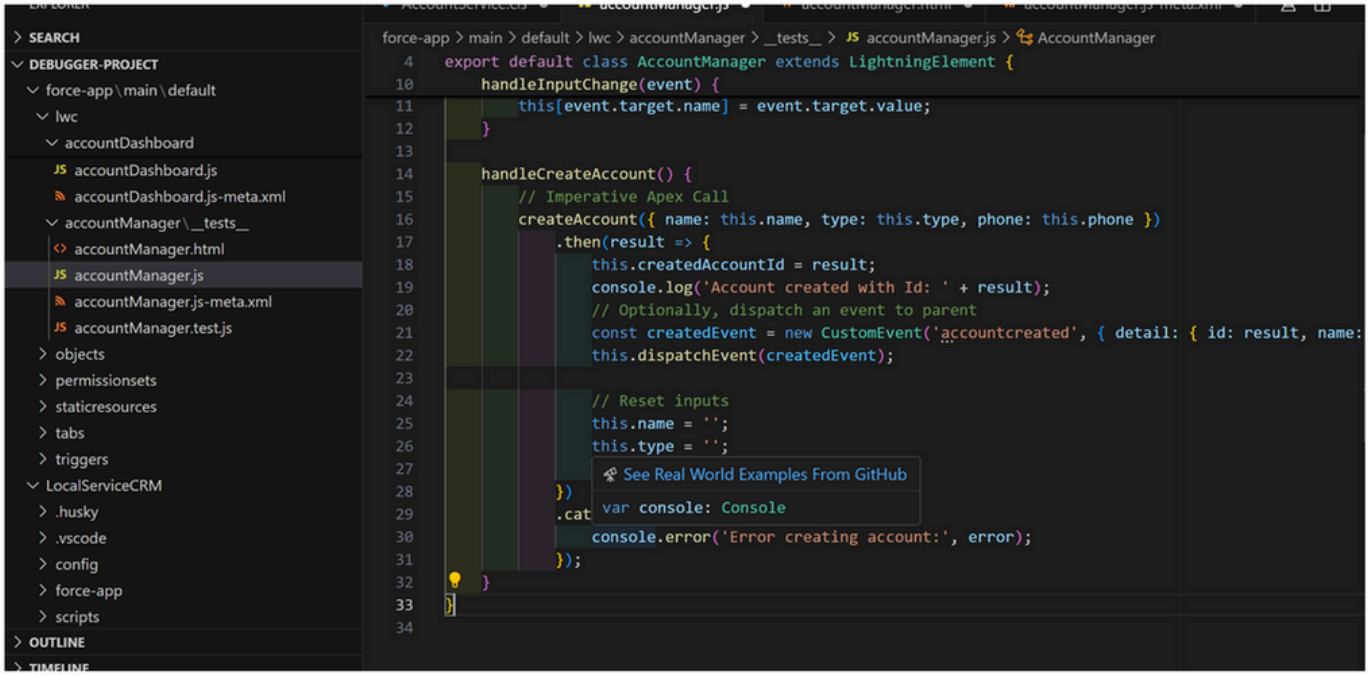
- **Unmanaged Package:**
  Components can be freely modified after installation in the target org. Commonly used for internal sharing or learning projects.
- **Managed Package:**
  Components are locked; updates and versioning are managed by the publisher. Often used for **AppExchange** products.
- **Usage in Project:**
  Packages were *not required* since all development occurred in a single Salesforce org.

**7. ANT Migration Tool**

- **Definition:**
  The ANT Migration Tool is a command-line utility that enables scripted deployment and retrieval of metadata between orgs.
- **Purpose:**
  Ideal for automated and bulk deployments in large-scale or enterprise-level Salesforce projects.
- **Usage in Project:**
  *Not implemented* due to the small project scope.

**8. VS Code & Salesforce CLI (SFDX)**

- **Definition:**
  Visual Studio Code (VS Code) integrated with Salesforce CLI (SFDX) is used for metadata management and version-controlled development.
- **Purpose:**
  Allows developers to connect local projects to Salesforce orgs, perform deployments, and manage source code efficiently.
- **Usage in Project:**
  - Setup completed successfully and commands like sf org list, sf project status, and sf org open were tested.
  - Encountered minor connection issues, so most customization and testing were completed directly in the **Salesforce org (browser)**.



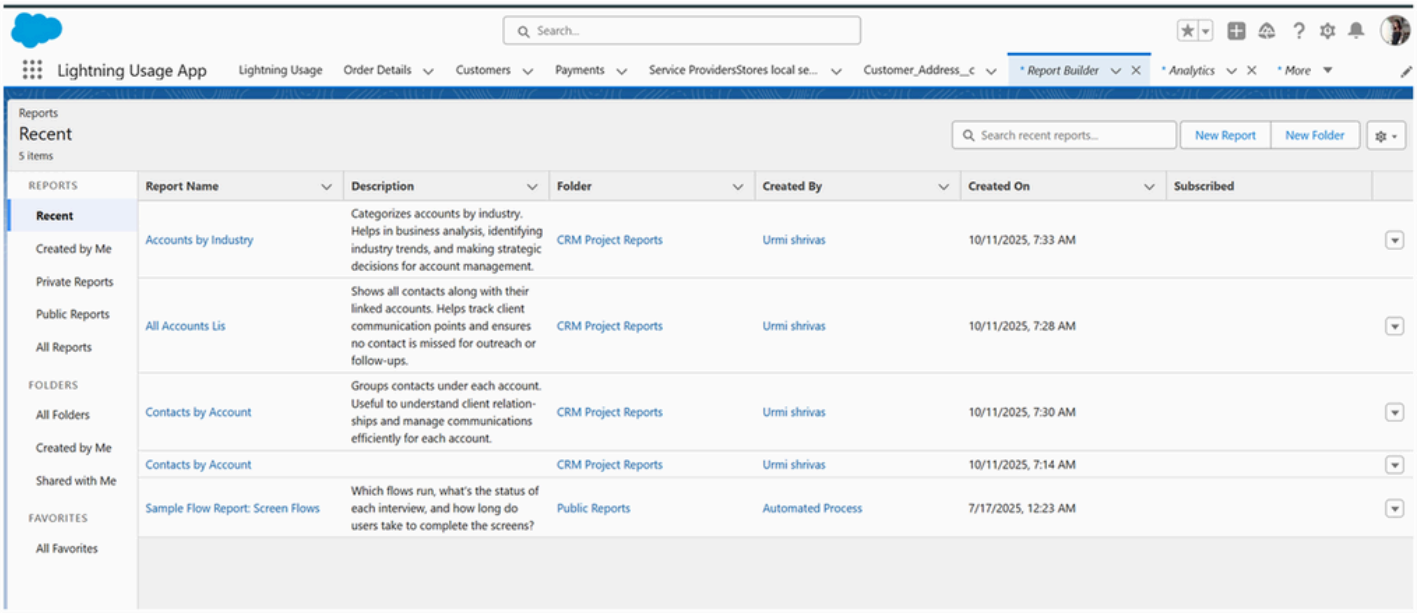

---

# Phase 9: Reporting, Dashboards & Security Review

This phase focuses on analyzing project data using **Reports and Dashboards**, and ensuring **data security** through Salesforce's security features such as sharing settings, field-level security, and audit tracking.

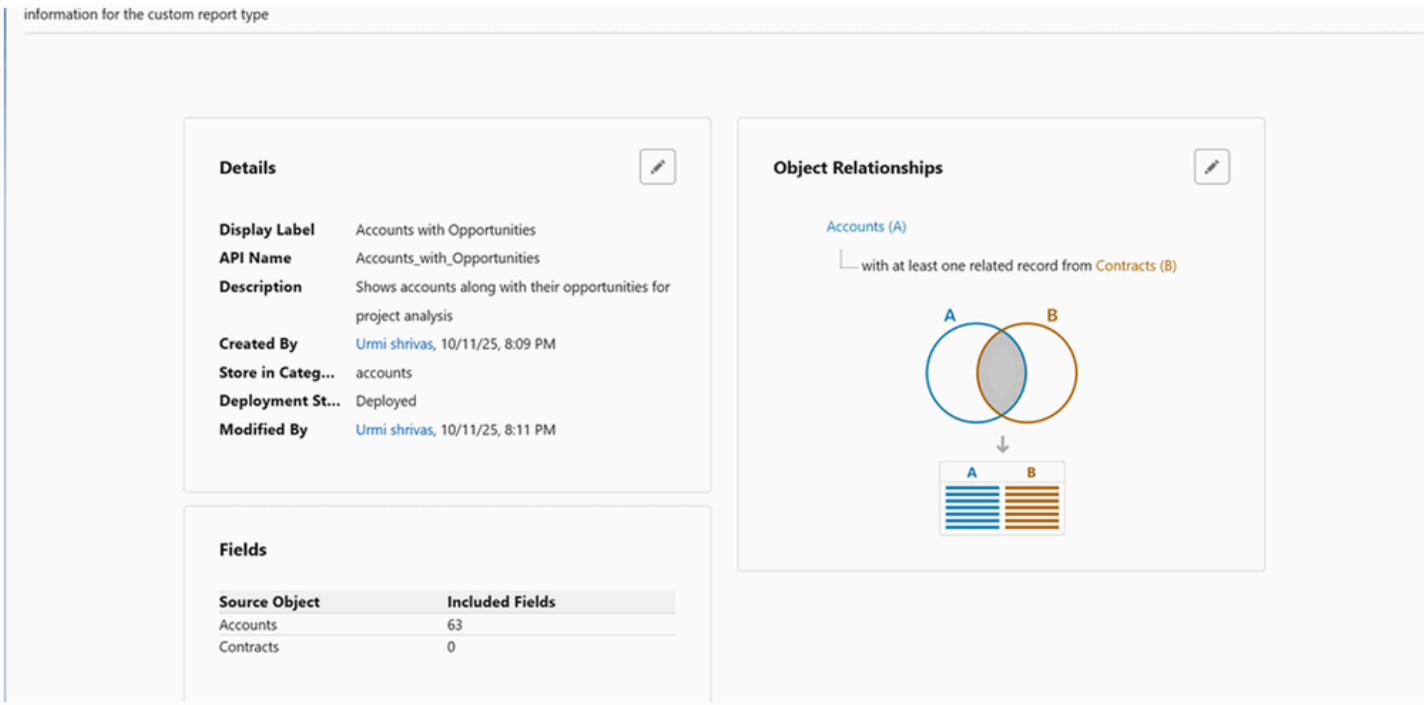
**1. Reports (Tabular, Summary, Matrix, Joined)**

- **Folder:**
  All reports are stored in the **CRM Project Reports** folder for centralized access.
- **Types Implemented:**
  - **Tabular Report:** Simple list of records (e.g., All Customers).
  - **Summary Report:** Grouped data showing totals per group (e.g., Accounts by Industry).
  - **Matrix Report:** Data grouped by both rows and columns for cross-analysis (e.g., Contacts by Region and Account Type).
  - **Joined Report:** Combines data from multiple report types (e.g., Service Requests and Payments).
- **Purpose:**
  Reports are aligned with project requirements such as tracking **clients, contacts, industry distribution,** and **owner responsibilities.**
- **Filters & Grouping:**
  Applied dynamically as per project needs to ensure meaningful insights and clean data visualization.



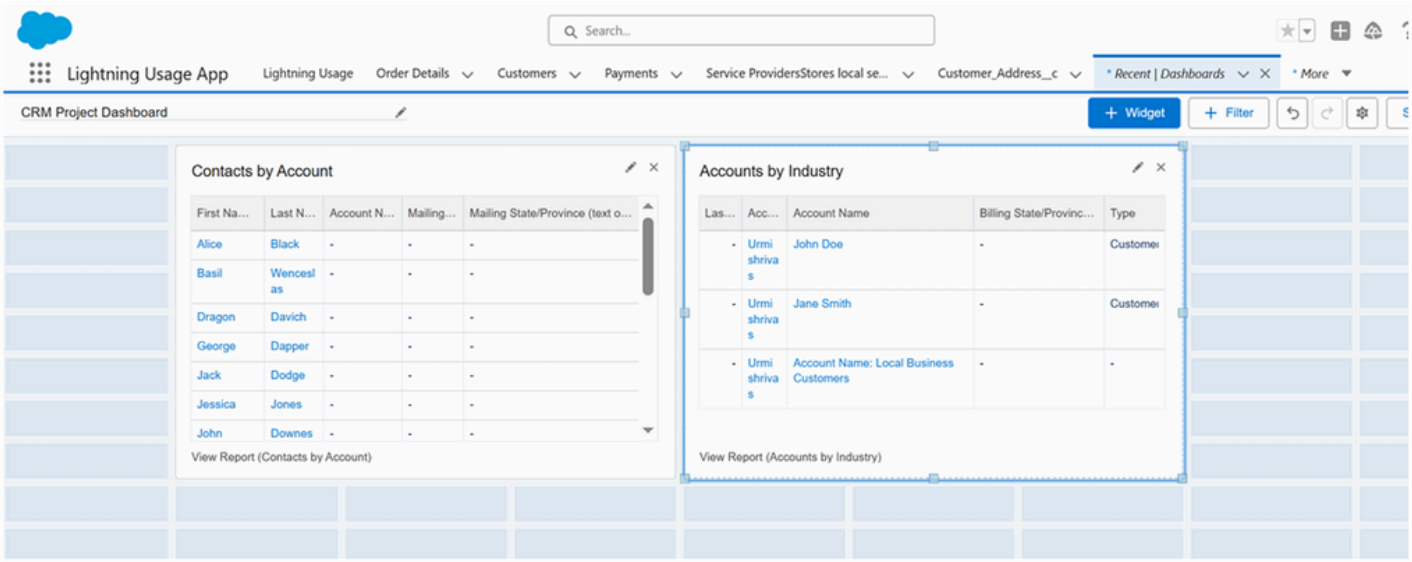

## 2. Report Types

- **Definition:**
  A **Report Type** defines which records and fields are available for reporting in Salesforce. It determines the relationship between objects included in the report.
- **Purpose in Project:**
  - Custom report types were configured to pull accurate data from **Accounts** and **Contacts.**
  - Enabled grouping, filtering, and detailed analysis specific to the CRM project's needs.
  - Ensured that only relevant fields were visible to maintain data consistency.




## 3. Dashboards

- **Purpose:**
  Dashboards provide a **visual summary of CRM data**, combining key reports for quick and interactive analysis.
- **Widgets Implemented:**
  - **Contacts by Account:** Displays the number of contacts under each account to monitor client relationships.
  - **Accounts by Industry:** Visualizes the distribution of accounts by industry for strategic planning.
- **Outcome:**
  The dashboard allows for **quick tracking of performance metrics**, **team workload monitoring**, and **data-driven decision-making**.



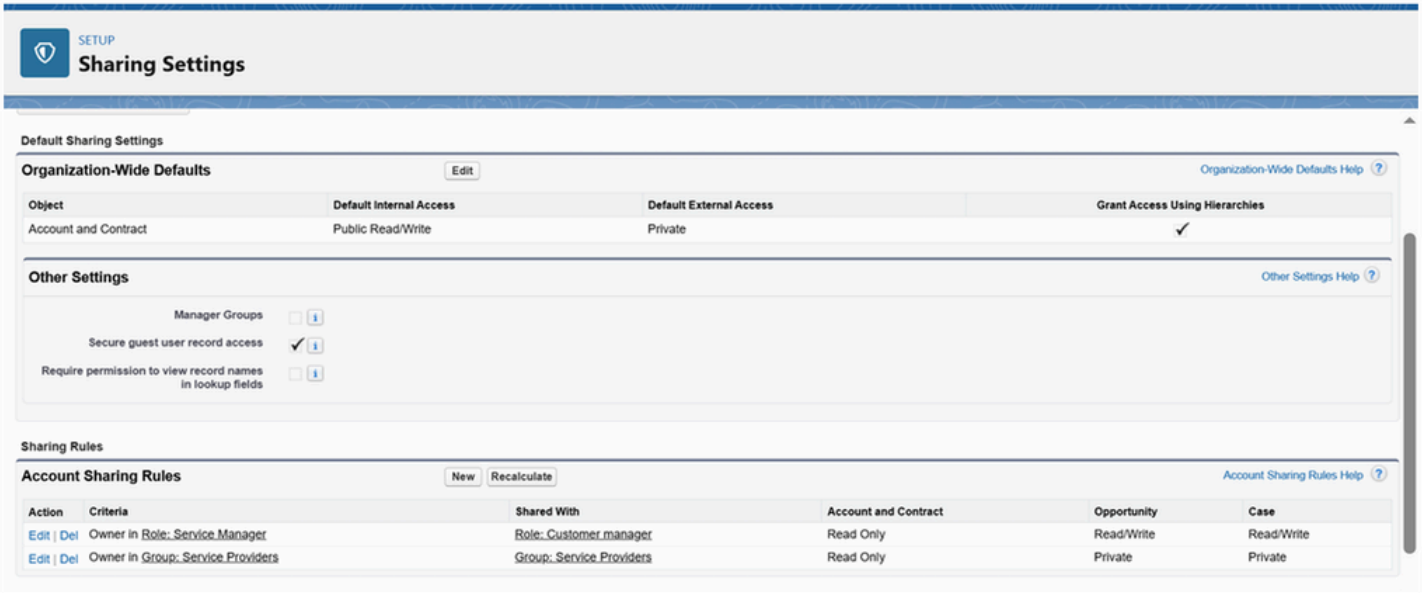

### 4. Dynamic Dashboards

- **Definition:**
  Dynamic Dashboards show data according to the **logged-in user's access and records.**
- **Purpose in Project:**
  Each team member can view **their own assigned accounts and contacts**, ensuring data personalization.
- **Outcome:**
  Promotes transparency, efficiency, and secure visibility of user-specific data.

### 5. Sharing Settings

- **Objective:**
  To define **role-based access control** within the CRM system.
- **Implementation in Project:**
  - **Service Provider:** Handles daily service updates and record maintenance.
  - **Service Manager:** Oversees operations, monitors task progress, and ensures data quality.

- **Outcome:**
  This ensures **controlled access, accountability,** and smooth workflow within the project hierarchy.




### 6. Field-Level Security

- **Definition:**
  Controls **visibility and edit access** for individual fields based on user profiles.
- **Purpose in Project:**
  Sensitive fields (e.g., Payment Details, Customer Value) were hidden from junior roles.
- **Outcome:**
  Ensures **data privacy** while granting appropriate access to relevant users.

### 7. Session Settings

- **Purpose:**
  Defines **session timeout and security policies** to protect the CRM system.
- **Implementation:**
  - Configured to automatically log out inactive users after a set period.
  - Prevents unauthorized system access.
- **Outcome:**
  Enhanced CRM security and ensured compliance with standard security protocols.

### 8. Login IP Ranges

- **Definition:**
  Restricts user login access to Salesforce from specific IP addresses.
- **Implementation:**
  Configured IP ranges for **Service Managers** and **Service Providers** to allow access **only from the company network.**

- **Outcome:**
  Ensures **controlled, secure, and organization-based data access.**

**9. Audit Trail**

- **Definition:**
  Tracks all **setup changes and configurations** made in Salesforce.
- **Usage in Project:**
  - Monitored modifications to **fields, sharing rules,** and **security settings.**
  - Helped identify configuration changes for troubleshooting and accountability.
- **Outcome:**
  Improved transparency and governance throughout the project lifecycle.