

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

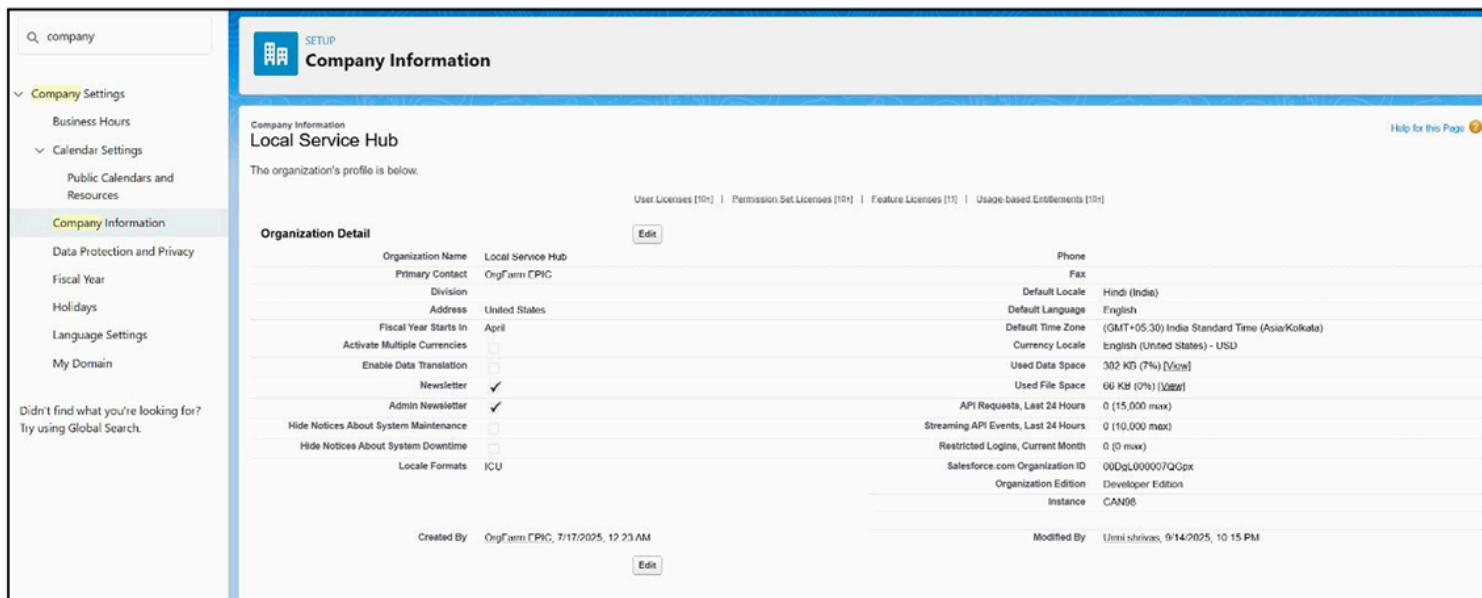
1. Salesforce Edition

- 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.



The screenshot shows the 'Company Information' page in the Salesforce Setup. The left sidebar has a 'Company Settings' section with 'Company Information' selected. The main content area is titled 'Company Information' and shows the 'Local Service Hub' organization profile. It includes sections for 'Organization Detail' (with fields like Organization Name, Primary Contact, Division, Address, Fiscal Year Starts In, etc.) and 'Phone' (with fields like Default Locale, Default Language, Default Time Zone, etc.). Other sections include 'User Licenses', 'Permission Set Licenses', 'Feature Licenses', 'Usage-based Entitlements', and various performance metrics like API Requests and Streaming API Events. The bottom of the page shows the 'Created By' and 'Modified By' information.

3. Business Hours & Holidays

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

Business Hours Detail

Business Hours Name	Day	Time Zone	Default Business Hours
Business Hours	Sunday	24 Hours	(GMT+05:30) India Standard Time (Asia/Kolkata)
	Monday	24 Hours	
	Tuesday	24 Hours	
	Wednesday	24 Hours	
	Thursday	24 Hours	
	Friday	24 Hours	
	Saturday	24 Hours	

Holidays

Holiday Name	Description	Date and Time
Dwali		11/1/2025 All Day
Independence Day		8/15/2025 All Day
Decouac day		1/26/2026 All Day

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.

Organization Fiscal Year Edit: Local Service Hub

To specify the fiscal year type for your organization, choose one of the options below.

Fiscal Year Information

Your organization can change the fiscal year start month, and specify whether the fiscal year name is set to the starting or ending year. For example, if your fiscal year starts in April 2025 and ends in March 2026, your Fiscal Year setting can be either 2025 or 2026.

Change Fiscal Year Period

Name	Local Service Hub
Fiscal Year Start Month	April
Fiscal Year is Based On	<input checked="" type="radio"/> The ending month <input type="radio"/> The starting month

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

Action	Full Name	Alias	Username	Role	Active	Profile
<input type="checkbox"/>	Chatter Expert	Chatter	chatty:00d0000007oxoxuan.sddfrmxsdotu@chatter.salesforce.com		<input checked="" type="checkbox"/>	Chatter Free User
<input type="checkbox"/>	EPIC_OrgTeam	OPIC	epic:2h09k0c91842@orgteam.salesforce.com		<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	shives_Urm	urm	urma.shives.cs72918@googltzrc.com		<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	User_Integration	integ	integration@00dg0000007gpueug.com		<input checked="" type="checkbox"/>	Analytics Cloud Integration User
<input type="checkbox"/>	User_Security	sec	insightssecurity@00da000007ogoxuag.com		<input checked="" type="checkbox"/>	Analytics Cloud Security User
<input type="checkbox"/>	User_Test	tuser	test.user@localcm.com		<input checked="" type="checkbox"/>	Standard User

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.

The screenshot shows the Salesforce Setup interface with the 'User Management Settings' page open. The left sidebar has 'Users' selected. The main area is titled 'All Users' and shows a list of users with various roles and profiles assigned. The columns include Action, Full Name, Alias, Username, Role, Active, and Profile. The list includes users like Chatter_Enabled, OFPIC, shreyas_1990, infog, User_Security, and test_user.

Action	Full Name	Alias	Username	Role	Active	Profile
<input type="checkbox"/>	Chatter_Enabled	Chatter	chatty.00d00000000000000000000000000000@chatter.salesforce.com	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Chatter Free User
<input type="checkbox"/>	OPIC_OrgForm	OPIC	opic.2609000000000000000000000000000@orgfarm.salesforce.com	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	shreyas_1990	shreyas_1990	shreyas.1990.cs2938@geantforce.com	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	User_Integration	infog	integration@00dg00000000007@geantforce.com	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Analytics Cloud Integration User
<input type="checkbox"/>	User_Security	sec	insightssecurity@00do00000000000@geantforce.com	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Analytics Cloud Security User
<input type="checkbox"/>	User_Test	test_user	test_user@localrm.com	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standard User

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.

The screenshot shows the Salesforce Setup interface with the 'Roles' tab selected. On the left, there's a sidebar with 'Sales' and 'Service' sections, and a search bar at the top. The main area displays a hierarchical tree titled 'Your Organization's Role Hierarchy'. The hierarchy starts with 'Local Service Hub', which branches into 'CEO', 'Customer manager', 'Customer', 'Service Manager', and 'Service Provider'. Each node has 'Edit | Del | Assign' options. An 'Add Role' button is located next to each parent node. A 'Help for this Page' link is in the top right corner.

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.

The screenshot shows the Salesforce Setup interface with the 'Permission Sets' tab selected. A specific permission set, 'Manager Permission Set', is highlighted. The page provides a detailed overview of the permission set, including its API name ('Manager Permission Set'), namespace prefix ('Umls_shivas'), and creation and modification dates. The 'Apps' section lists various permissions such as 'Assigned Apps', 'Assigned Connected Apps', 'Object Settings', 'App Permissions', 'Apex Class Access', 'Visualforce Page Access', 'External Data Source Access', and 'Flow Access'.

- ◆ 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

Q_ permis

Users

Permission Set Groups

Permission Sets

Custom Code

Custom Permissions

Didn't find what you're looking for? Try using Global Search.

Search Setup

Permission Sets

Manager Permission Set

Q_ Find Settings | Clone | Delete | Edit Properties | Manage Assignments | View Summary

Permission Set Overview

Description	API Name
License	Namespace Prefix
Session Activation Required	Created By
Permission Set Groups Added To	Last Modified By

Assigned Apps
Settings that specify which apps are visible in the app menu

Assigned Connected Apps
Settings that specify which connected apps are visible in the app menu

Object Settings
Permissions to access objects and fields, and settings such as tab availability

App Permissions
Permissions to perform app specific actions, such as "Manage Call Centers"

Apex Class Access
Permissions to execute Apex classes

Visualforce Page Access
Permissions to execute Visualforce pages

External Data Source Access
Permissions to authenticate against external data sources

Flow Access
Permissions to execute Flows

Help for this Page

Q_ SHARING

Security

Guest User Sharing Rule Access Report

Sharing Settings

Didn't find what you're looking for? Try using Global Search.

Search Setup

Sharing Settings

Organization-Wide Sharing Defaults Edit

Edit your organization-wide sharing defaults below. Changing these defaults will cause all sharing rules to be recalculated. This could require significant system resources and time depending on the amount of data in your organization. Setting an object to Private makes records visible to record owners and those above them in the role hierarchy, and access can be extended using sharing rules.

Save | Cancel

Object	Default Internal Access	Default External Access	Grant Access Using Hierarchies
Lead	Public Read/Write/Transfer	Private	<input type="checkbox"/>
Account and Contract	Public Read/Write	Private	<input type="checkbox"/>
Order	Controlled by Parent	Controlled by Parent	<input type="checkbox"/>
Contact	Controlled by Parent	Controlled by Parent	<input type="checkbox"/>
Asset	Controlled by Parent	Controlled by Parent	<input type="checkbox"/>
Opportunity	Public Read/Write	Private	<input type="checkbox"/>
Case	Public Read/Write/Transfer	Private	<input type="checkbox"/>
Campaign	Public Full Access	Private	<input type="checkbox"/>
Campaign Member	Controlled by Campaign	Controlled by Campaign	<input type="checkbox"/>
User	Public Read Only	Private	<input type="checkbox"/>
Individual	Public Read/Write	Private	<input type="checkbox"/>
Voice Call	Private	Private	<input type="checkbox"/>
Activity	Private	Private	<input type="checkbox"/>
Calendar	Hide Details and Add Events	Hide Details and Add Events	<input type="checkbox"/>
Price Book	Use	Use	<input type="checkbox"/>
Product	Public Read/Write	Public Read/Write	<input type="checkbox"/>
Agent Work	Public Read Only	Private	<input type="checkbox"/>
Alternative Payment Method	Private	Private	<input type="checkbox"/>
Analytics User Attribute Function Token	Public Read Only	Private	<input type="checkbox"/>
Appointment Invitation	Private	Private	<input type="checkbox"/>
Approval Submission	Private	Private	<input type="checkbox"/>
Authorization Form	Private	Private	<input type="checkbox"/>

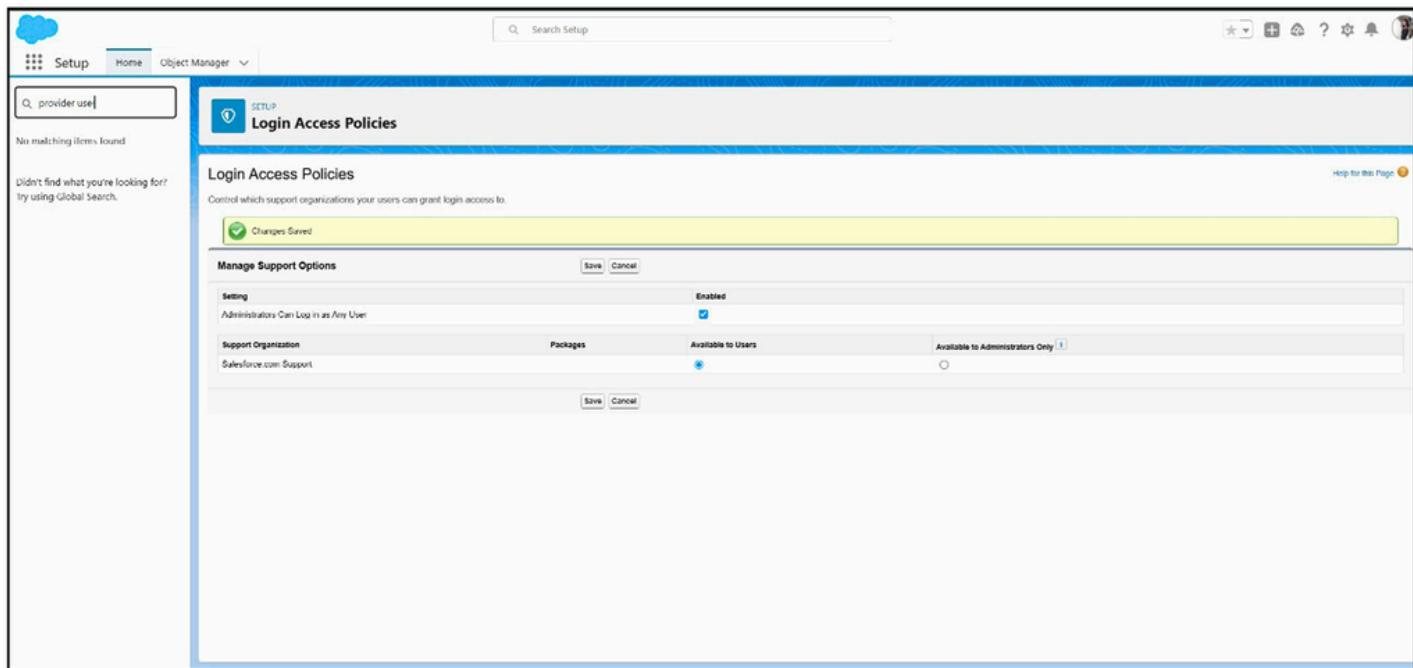
Help for this Page

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.

This screenshot shows the 'Service Request' object setup page in the Salesforce Object Manager. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. The left sidebar lists various configuration tabs: 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, and Conditional Field Formatted. The main content area is titled 'Details' and contains sections for Description (stores information about service requests), API Name (Service_Request_c), Singular Label (Service Request), Plural Label (Service_Requests), and several checkboxes for features like Enable Reports, Track Activities, Track Field History, Deployment Status (Deployed), Help Settings, and Standard Salesforce.com Help Window. At the bottom right are 'Edit' and 'Delete' buttons.

This screenshot shows the 'Service Provider' object setup page in the Salesforce Object Manager. The top navigation bar includes 'Setup', 'Home', and 'Object Manager'. The left sidebar lists various configuration tabs: 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, and Conditional Field Formatted. The main content area is titled 'Details' and contains sections for Description (stores information about service providers), API Name (Service_Provider_c), Singular Label (Service Provider_c), Plural Label (Service_Providers_c), and several checkboxes for features like Enable Reports, Track Activities, Track Field History, Deployment Status (Deployed), Help Settings, and Standard Salesforce.com Help Window. At the bottom right are 'Edit' and 'Delete' buttons.

Payment c	
Details	Edit Delete
Fields & Relationships	Details
Page Layouts	Description Tracks customer payments related to service requests.
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	

Setup Home Object Manager

SETUP > OBJECT MANAGER
Appointment

Details	Details	
Helds & Relationships	Description	Edit Delete
Page Layouts		
Lightning Record Pages	API Name	
Buttons, Links, and Actions	Appointment__c	
Compact Layouts	Custom	
Field Sets	✓	
Object Limits	Singular Label	
Record Types	Appointment	
Related Lookup Filters	Plural Label	
Restriction Rules	Appointment__c	
Scoping Rules		
Object Access		
Triggers		
Flow Triggers		
Validation Rules		
Conditional Field Formulation		

Details	
Fields & Relationships	Edit Delete
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	
Description	
Collects customer feedback on completed service requests.	
API Name	Edit Delete
Feedback_c	
Custom	
✓	
Singular Label	Edit Delete
Feedback c	
Plural Label	Edit Delete
Feedback_c	
Enable Reports	Edit Delete
✓	
Track Activities	Edit Delete
✓	
Track Field History	Edit Delete
✓	
Deployment Status	Edit Delete
Deployed	
Help Settings	Edit Delete
Standard.salesforce.com Help Window	

The screenshot shows the Salesforce Setup interface with the title "Customer Address". The left sidebar lists various setup categories like Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, etc. The main content area is titled "Details" and contains sections for "Fields & Relationships" and "Details". Under "Fields & Relationships", there are fields for API Name (Customer_Address__c), Singular Label (Customer Address), and Plural Label (Customer_Address__c). Under "Details", there are sections for Description, API Name (Customer_Address__c), Singular Label (Customer Address), Plural Label (Customer_Address__c), and a "Related Objects" section with checkboxes for "Enable Reports", "Track Activities", "Track Field History", "Deployment Status" (set to Deployed), and "Help Settings". A link to "Standard Salesforce.com Help Window" is also present.

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

The screenshot shows the Salesforce Setup interface with the title "Payment". The left sidebar lists various setup categories. The main content area is titled "Fields & Relationships" and contains a table with 8 items, sorted by Field Label. The table has columns for FIELD LABEL, FIELD NAME, DATA TYPE, CONTROLLING FIELD, and INDEXED. The data is as follows:

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Amount	Amount__c	Currency(16, 2)		
Created By	CreatedBy	Lookup(User)		
Last Modified By	LastModifiedBy	Lookup(User)		
Owner	OwnerId	Lookup(User, Group)		
Payment Date	Payment_Date__c	Date		
Payment Method	Payment_Method__c	IdList		
Payment Name	Name	Auto Number		
Service Request	Service_Request__c	Lookup(Service Request)		

Fields & Relationships					
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts	Created By	CreatedById	Lookup(User)		
Lightning Record Pages	Customer	Customer__c	Lookup(Customer)		▼
Buttons, Links, and Actions	Description	Description__c	Text Area(255)		▼
Compact Layouts	Last Modified By	LastModifiedById	Lookup(User)		▼
Field Sets	Owner	OwnerId	Lookup(User,Group)		▼
Object Limits	Priority	Priority__c	Picklist		▼
Record Types	Request Date	RequestDate__c	Date		▼
Related Lookup Filters	Service Provider	ServiceProvider__c	Lookup(Service Provider)		▼
Restriction Rules	Service Request Name	Name	Auto Number		▼
Scoping Rules	Status	Status__c	Picklist		▼
Object Access					
Triggers					
Flow Triggers					
Validation Rules					
Conditional Field Formating					

Fields & Relationships					
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts	Address	Address__c	Text Area(255)		▼
Lightning Record Pages	Created By	CreatedBy	Lookup(User)		▼
Buttons, Links, and Actions	Customer Name	Name	Text(60)		▼
Compact Layouts	Customer Type	CustomerType__c	Picklist		▼
Field Sets	Email	Email__c	Email		▼
Object Limits	Last Modified by	LastModifiedById	Lookup(User)		▼
Record Types	Owner	OwnerId	Lookup(User,Group)		▼
Related Lookup Filters	Phone	Phone__c	Phone		▼
Restriction Rules	Registration Date	RegistrationDate__c	Date		▼
Scoping Rules					
Object Access					
Triggers					
Flow Triggers					
Validation Rules					
Conditional Field Formating					

Fields & Relationships					
	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts	Availability	Availability_c	Checkbox		
Lightning Record Types	Contact	Contact_c	Phone		
Balloons, Links, and Actions	Created By	CreatedBy	Lookup(Users)		
Compact Layouts	Last Modified By	LastModifiedBy	Lookup(Users)		
Field Sets	Location	Location_c	Text(100)		
Object Limits	Owner	Owner	Lookup(User Groups)		
Record Types	Rating	Rating_c	Number(2, 1)		
Related Lookup Filters	Service Provider Name	Name	Text(60)		
Restriction Rules	Service Type	Service_Type_c	Picklist		
Scoping Rules					
Object Access					
Triggers					
Flow Triggers					
Validation Rules					
Conditional Field Formatted					

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

The screenshot shows the Salesforce Setup interface with the 'Object Manager' selected. Under the 'Payment' object, the 'Record Types' section is active. A table lists four record types: Card Payment, Cash Payment, Online Transfer, and UPPI Payment. Each row includes columns for 'Record Type Label', 'Description', 'Active' status, and 'Modified By'. The table has a header row with 'Record Type Label', 'Description', 'Active', and 'Modified By'.

Record Type Label	Description	Active	Modified By
Card Payment		✓	Umeshivra, 5/20/2025, 10:34 AM
Cash Payment		✓	Umeshivra, 5/20/2025, 10:19 AM
Online Transfer		✓	Umeshivra, 5/20/2025, 10:34 AM
UPI Payment		✓	Umeshivra, 5/20/2025, 10:35 AM

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)

The screenshot shows the Salesforce Setup interface with the 'Sharing Settings' page open. The left sidebar shows 'SHARING' selected under 'Security'. The main area is titled 'Organization-Wide Sharing Defaults Edit' and contains a table for setting sharing defaults for various objects. The table has columns for 'Object', 'Default Internal Access', 'Default External Access', and 'Grant Access Using Hierarchies'. Each object row has dropdown menus for selecting sharing levels like 'Public Read/Write/Transfer', 'Private', or 'Controlled by Parent'.

Object	Default Internal Access	Default External Access	Grant Access Using Hierarchies
Lead	Public Read/Write/Transfer	Private	
Account and Contract	Public Read/Write	Private	
Order	Controlled by Parent	Controlled by Parent	
Contact	Controlled by Parent	Controlled by Parent	
Asset	Controlled by Parent	Controlled by Parent	
Opportunity	Public Read/Write	Private	
Case	Public Read/Write/Transfer	Private	
Campaign	Public Full Access	Private	
Campaign Member	Controlled by Campaign	Controlled by Campaign	
User	Public Read Only	Private	
Individual	Public Read/Write	Private	
Voice Call	Private	Private	
Activity	Private	Private	
Calendar	Hide Details and Add Events	Use	
Price Book	Use	Public Read/Write	
Product	Public Read/Write	Private	
Agent Work	Public Read Only	Private	
Alternative Payment Method	Private	Private	
Analytics User Attribute Function Token	Public Read Only	Private	
Appointment Invitation	Private	Private	
Approval Submission	Private	Private	
Authorization Form	Private	Private	

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

The screenshot shows the Salesforce Setup interface with the following details:

- Object Manager:** Payment
- Record Types:** 4 items, Sorted by Record Type Label
- Columns:** RECORD TYPE LABEL, DESCRIPTION, ACTIVE, MODIFIED BY
- Data:**

Record Type Label	Description	Active	Modified By
Cash Payment		✓	Urmil shrivastava, 5/20/2025, 10:34 AM
Cash Payment		✓	Urmil shrivastava, 5/20/2025, 10:35 AM
Online Transfer		✓	Urmil shrivastava, 5/20/2025, 10:34 AM

The screenshot shows the Salesforce Setup interface with the following details:

- Object Manager:** Service Request
- Record Types:** 5 items, Sorted by Record Type Label
- Columns:** RECORD TYPE LABEL, DESCRIPTION, ACTIVE, MODIFIED BY
- Data:**

Record Type Label	Description	Active	Modified By
Cleaning Request		✓	Urmil shrivastava, 5/20/2025, 10:05 AM
Electrical Request		✓	Urmil shrivastava, 5/20/2025, 10:05 AM
Emergency Request		✓	Urmil shrivastava, 5/20/2025, 10:07 AM
Plumbing Request		✓	Urmil shrivastava, 5/20/2025, 10:09 AM
Regular Service Request		✓	Urmil shrivastava, 5/20/2025, 9:56 AM

7. Compact Layouts

Customer:

Key Fields: Name, Email, Phone, Customer Type

Layout Name: Customer_Compact_Layout (Set as Primary)

Service Provider:

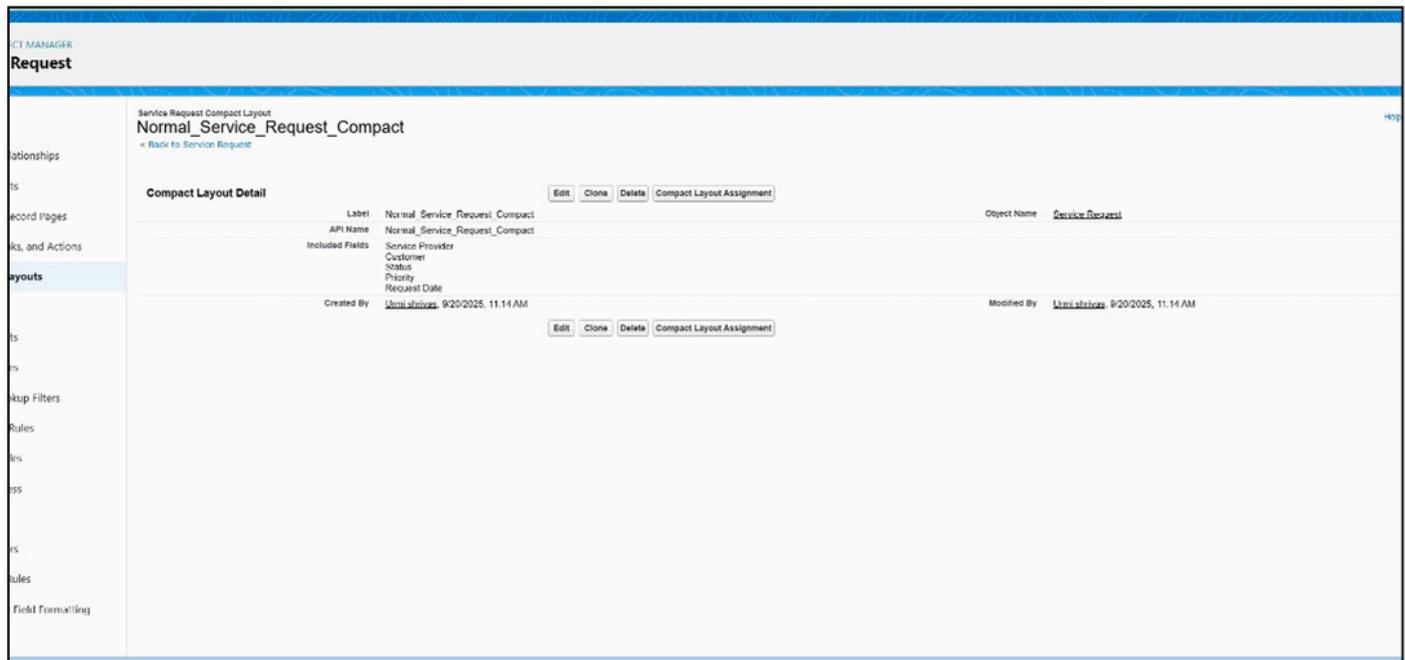
Key Fields: Name, Service Type, Rating, Availability

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)



MANAGER
request

Relationships

Pages and Actions

Items

Filters

Formatting

Service Request Compact Layout
Cleaning_Service_Request_Compact
« Back to Service Request

Compact Layout Detail

	Label	API Name	Object Name
Included Fields	Cleaning_Service_Request_Compact	Cleaning_Service_Request_Compact	Service Request
	Request Date		
	Customer		
	Record Type		
	Service Provider		
	Service Request Name		
	Status		
	Priority		

Created By: Umi shivdas, 9/20/2025, 11:16 AM Modified By: Umi shivdas, 9/20/2025, 11:16 AM

[Edit](#) [Clone](#) [Delete](#) [Compact Layout Assignment](#)

MANAGER
request

Relationships

Pages and Actions

Items

Filters

Formatting

Service Request Compact Layout
Emergency_Service_Request_Compact
« Back to Service Request

Compact Layout Detail

	Label	API Name	Object Name
Included Fields	Emergency_Service_Request_Compact	Emergency_Service_Request_Compact	Service Request
	Request Date		
	Customer		
	Service Provider		
	Status		
	Priority		

Created By: Umi shivdas, 9/20/2025, 11:16 AM Modified By: Umi shivdas, 9/20/2025, 11:16 AM

[Edit](#) [Clone](#) [Delete](#) [Compact Layout Assignment](#)

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:

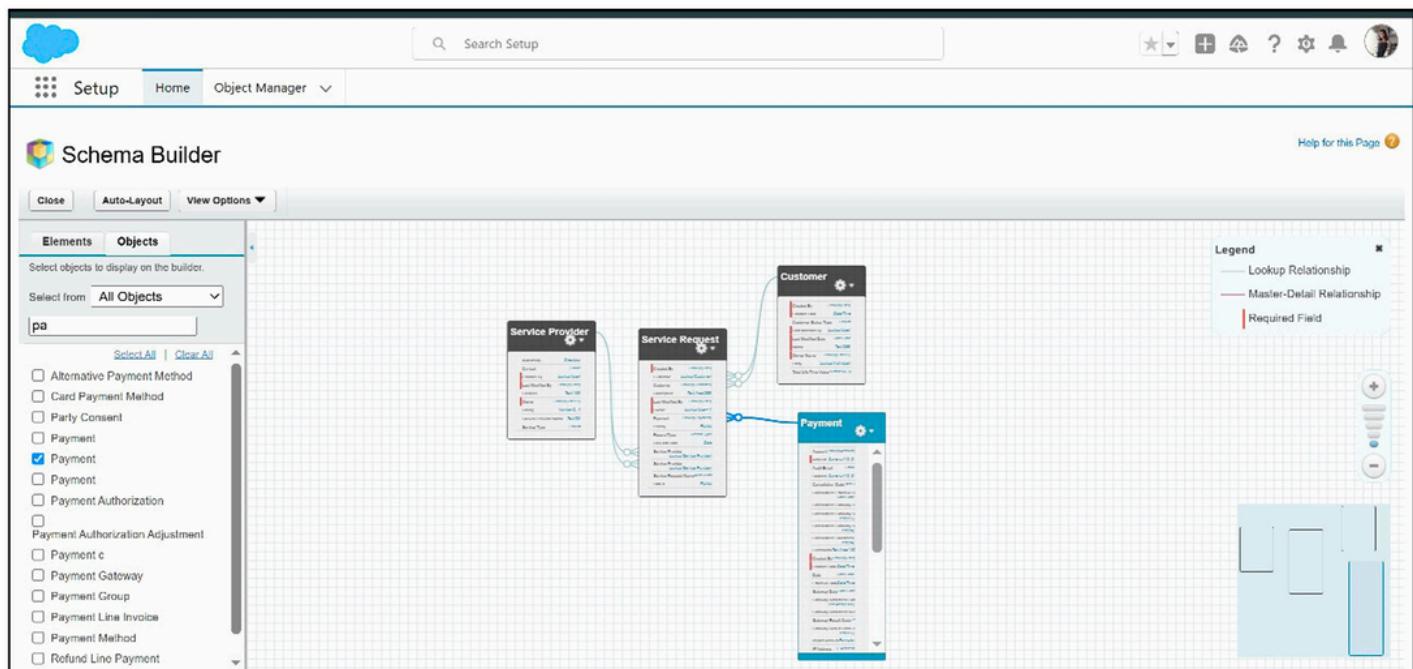
Not used (optional for user hierarchy).

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.

The screenshot shows the Salesforce Object Manager interface for the 'Appointment' object. The left sidebar lists various setup options like Details, Fields & Relationships, Page Layouts, etc. The main area displays a table titled 'Validation Rules' with two items:

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
Appointment_Date_Cannot_Be_Past	Appointment Date.	Appointment date cannot be in the past.	✓	Urmishrivs, 9/22/2025, 12:51 AM
Appointment_Must_Have_ServiceProvider	Service Provider	Appointment must have a Service Provider assigned.	✓	Urmishrivs, 9/22/2025, 12:58 AM

The screenshot shows the Salesforce Object Manager interface for the 'Service Request' object. The left sidebar lists various setup options like Details, Fields & Relationships, Page Layouts, etc. The main area displays a table titled 'Validation Rules' with two items:

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
Payment_Required_Before_Completion	Status	Cannot mark Completed without Payment.	✓	Urmishrivs, 9/22/2025, 12:41 AM
ServiceRequest_Priority_Required	Priority	Priority must be selected before saving the Service Request.	✓	Urmishrivs, 9/22/2025, 2:14 AM

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.

The screenshot shows the Salesforce Setup interface with the 'Approval Processes' page selected. The page title is 'Payment: Payment Approval Process'. The 'Process Definition Detail' section contains the following information:

Field	Value	Action
Process Name	Payment Approval Process	Edit Clone Delete Activate
Unique Name	Payment_Approval_Process	Active <input type="checkbox"/>
Description	Approval process for payments requiring manager approval	Next Automated Approver Determined By <input type="checkbox"/>
Entry Criteria	Payment: Payment Method EQUALS "Cash, Card, UPI, Online"	Manager of Record Submitter <input type="checkbox"/>
Record Editability	Administrator ONLY	Allow Submitters to Recall Approval Requests <input checked="" type="checkbox"/>
Approval Assignment Email Template	Support: Case Assignment Notification	
Initial Submitters	Payment Owner	
Created By	Urmil shrivastava, 9/22/2025, 6:12 AM	Modified By <input type="checkbox"/> Urmil shrivastava, 9/22/2025, 7:06 AM

Below this, there is a section titled 'Initial Submission Actions' with buttons for 'Add Existing' and 'Add New'.

Setup Home

Salesforce Go

Service Setup Assistant

Commerce Setup Assistant

Field Service Setup Home (Beta)

Hyperforce Assistant

Release Updates

Salesforce Mobile App

Lightning Usage

Optimizer

Sales Cloud Everywhere

ADMINISTRATION

> Users

> Data

SETUP Approval Processes

Approval Steps New Approval Step

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Hide Actions	1	Manager Approval	Requires manager approval for all payments	Manager		Final Rejection

✓ Approval Actions Add Existing Add New ▾

Action	Type	Description
Edit	Email Alert	Notify submitter upon approval

✗ Rejection Actions Add Existing Add New ▾

You have not yet defined any actions		
--------------------------------------	--	--

Final Approval Actions Add Existing Add New ▾

Action	Type	Description
Edit	Record Lock	Lock the record from being edited

Final Rejection Actions Add Existing Add New ▾

Action	Type	Description
Edit	Record Lock	Lock the record from being edited

Setup Home

Salesforce Go

Service Setup Assistant

Commerce Setup Assistant

Field Service Setup Home (Beta)

Hyperforce Assistant

Release Updates

Salesforce Mobile App

Lightning Usage

Optimizer

Sales Cloud Everywhere

ADMINISTRATION

> Users

> Data

SETUP Approval Processes

✗ Rejection Actions Add Existing Add New ▾

You have not yet defined any actions		
--------------------------------------	--	--

Final Approval Actions Add Existing Add New ▾

Action	Type	Description
Edit	Record Lock	Lock the record from being edited

Final Rejection Actions Add Existing Add New ▾

Action	Type	Description
Edit	Record Lock	Unlock the record for editing

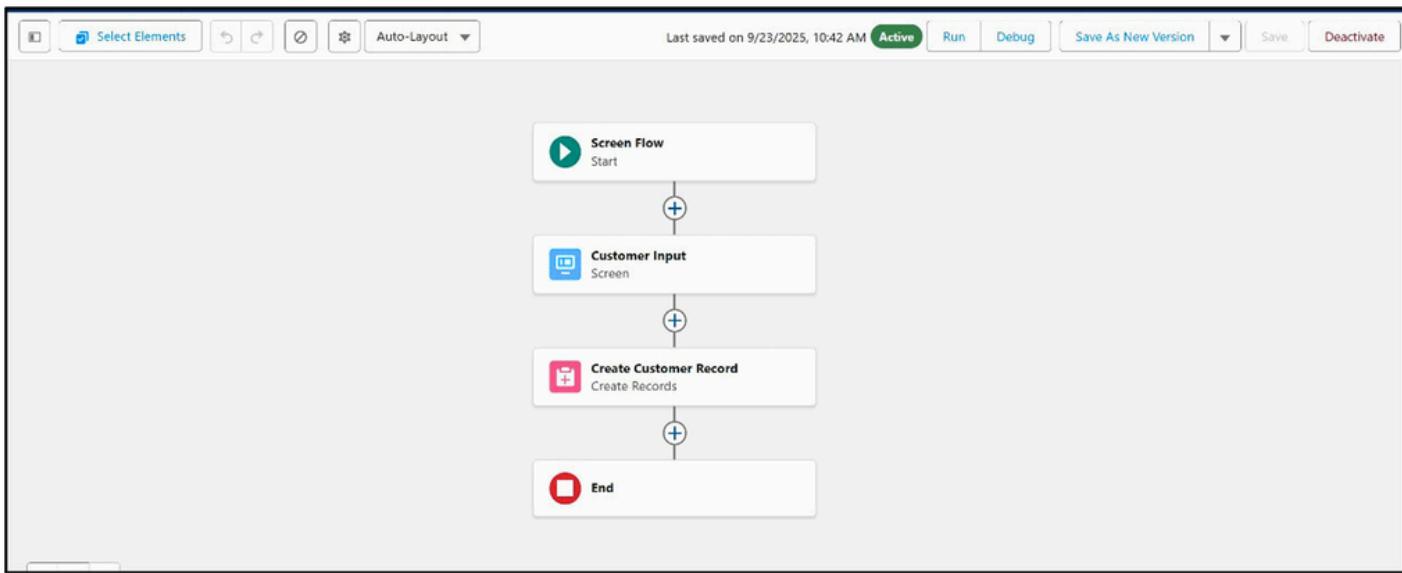
Recall Actions Add Existing Add New ▾

Action	Type	Description
Edit	Record Lock	Unlock the record for editing

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

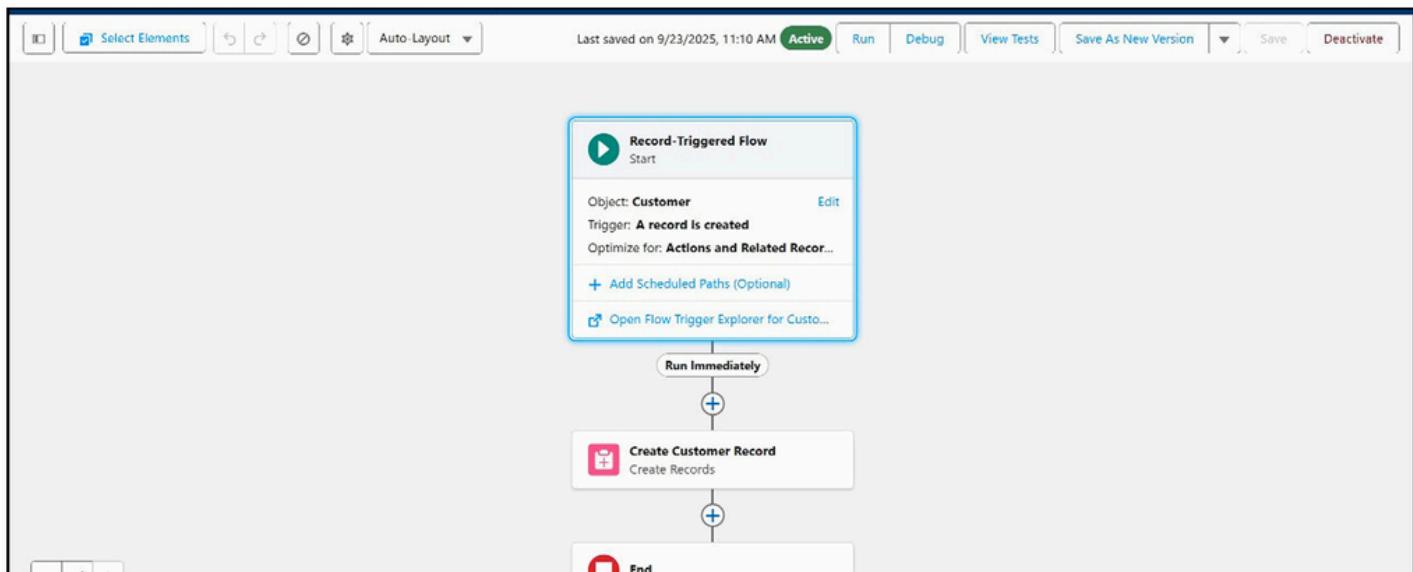
Screen Flow

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



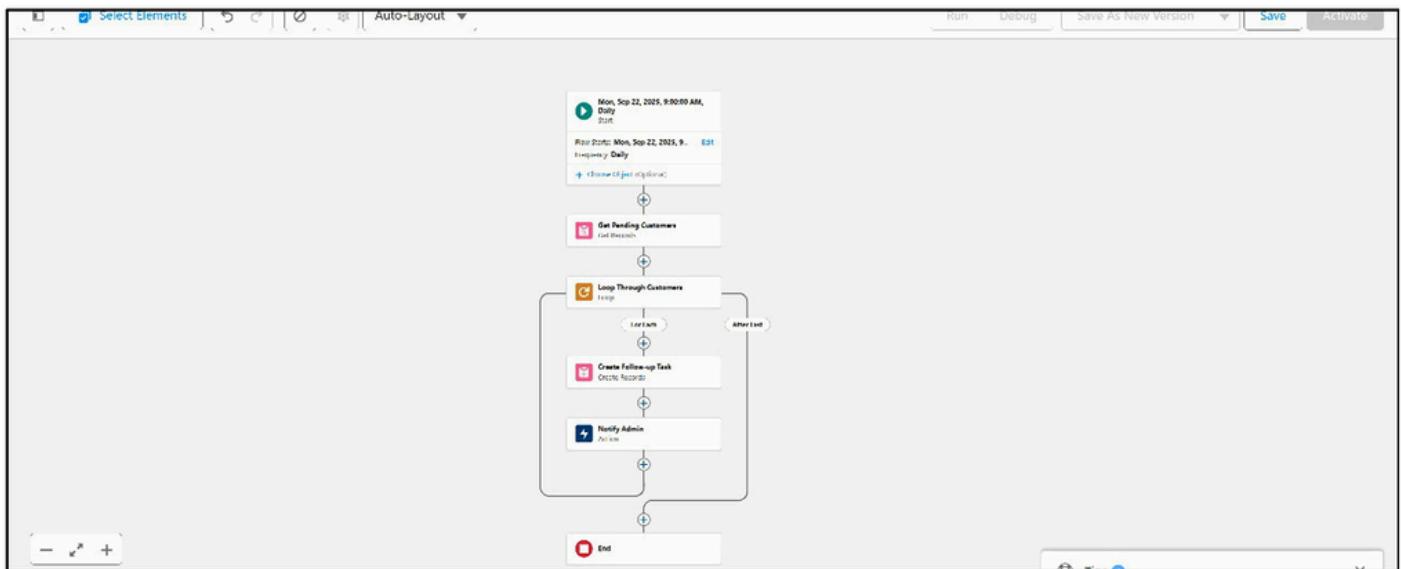
Record-Triggered Flow

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



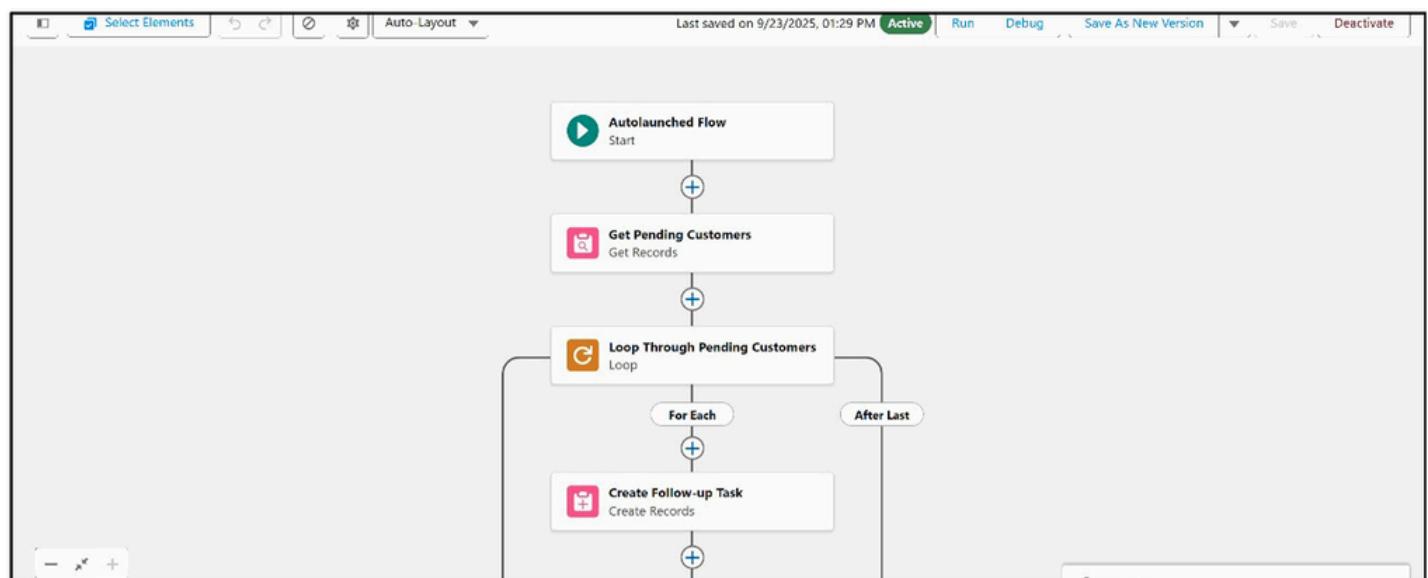
Scheduled Flow

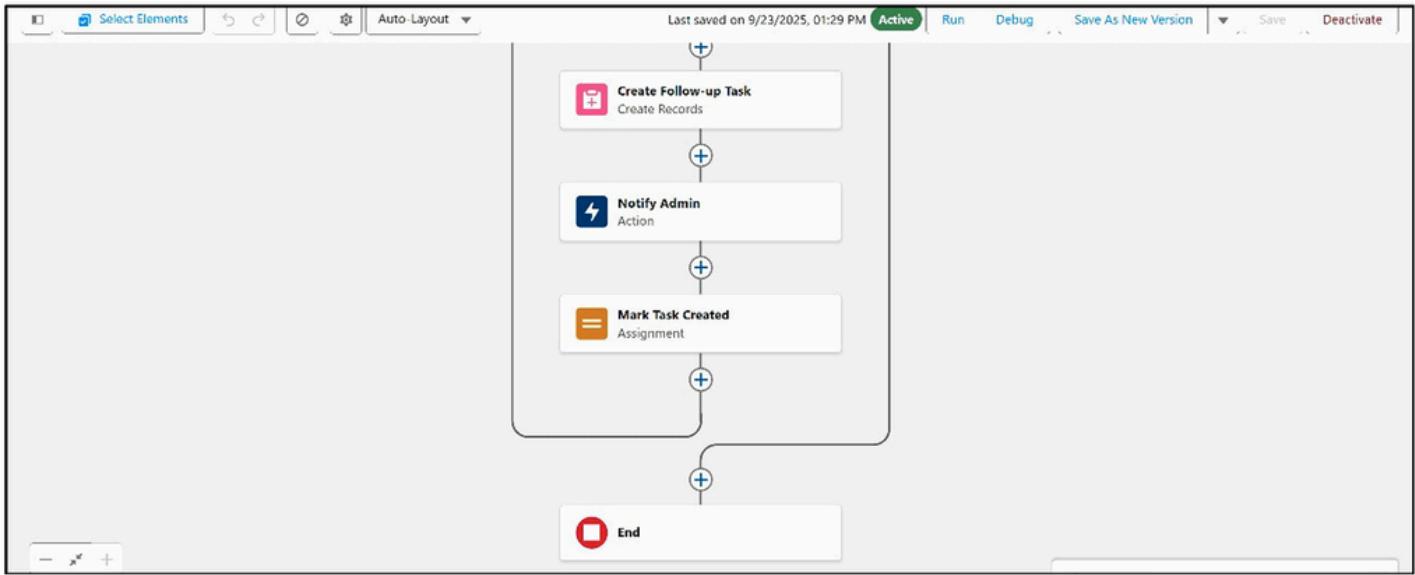
- Created for **Appointment Reminders**.
- Runs **daily at a fixed time** to send reminders before the appointment date.
- Ensures customers and providers are notified in advance.



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**.

TCS_LM_SF Home Accounts ▾ Contacts ▾ Order Details ▾ * Customer Follow up Notif... X

Search...

Email Template
Customer Follow-up Notification

Details Related

Information

Email Template Name: Customer Follow-up Notification

Description: Email sent to admin when a follow-up task is created

Made in Email Template Builder:

Related Entity Type: Contact

Folder: Private Email Templates

Message Content

Subject: New Follow-up Task for customer name

Enhanced Letterhead

HTML Value:

Additional Information

Created By: Urmil shrivastava, 9/23/2025, 2:41 AM

Last Modified By: Urmil shrivastava, 9/23/2025, 2:41 AM

TCS_LM_SF Home Accounts ▾ Contacts ▾ Order Details ▾ * Customer Task Assigned N... X

Search...

Email Template
Customer Task Assigned Notification

Details Related

Information

Email Template Name: Customer Task Assigned Notification

Description: Notify admin when a customer task is assigned.

Made in Email Template Builder:

Related Entity Type: Contact

Folder: Private Email Templates

Message Content

Subject: Task Assigned to contact name

Enhanced Letterhead

HTML Value:

Additional Information

Created By: Urmil shrivastava, 9/23/2025, 2:44 AM

Last Modified By: Urmil shrivastava, 9/23/2025, 2:44 AM

The screenshot shows the Salesforce Email Template page for 'Customer Escalation Notification'. The page includes sections for Details, Related, Information, Message Content, and Additional Information. It displays fields like Email Template Name, Description, Subject, HTML Value, and various status indicators.

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**”.

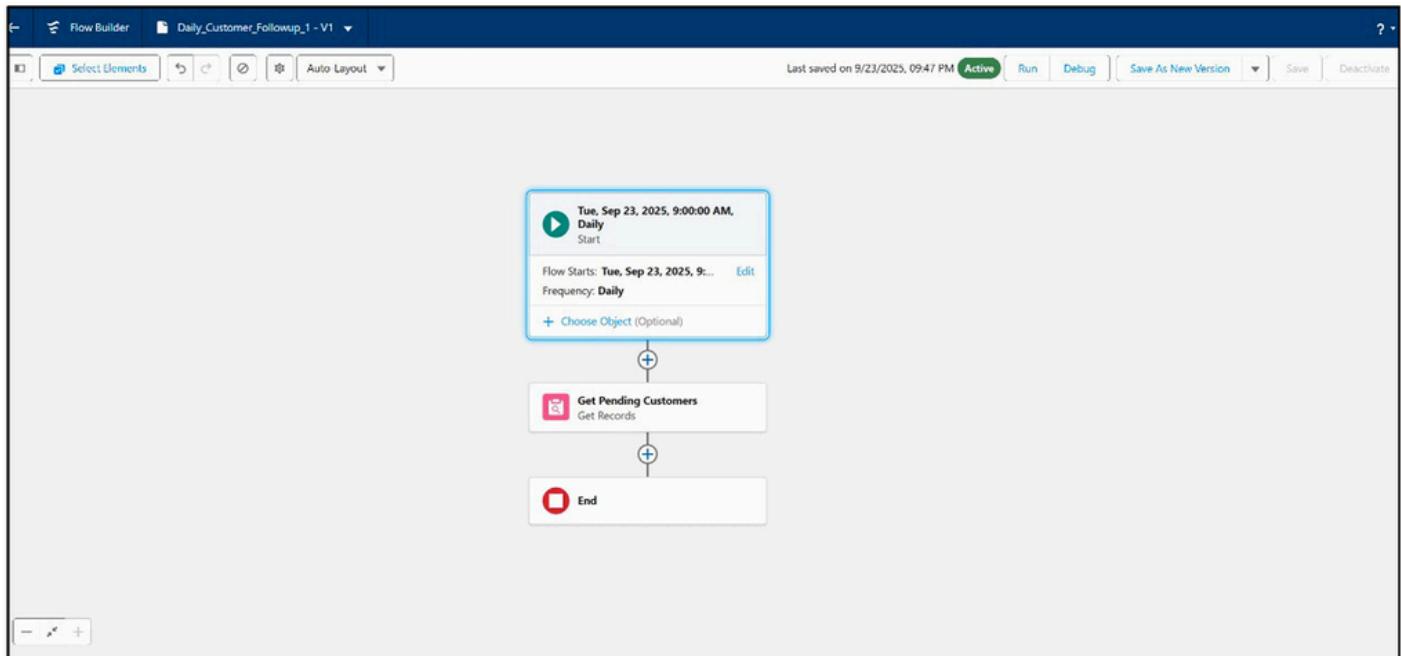
This automation maintains **data consistency** and **reduces manual intervention**.

The screenshot shows the Salesforce Setup page under the 'Field Updates' section. It displays a 'Field Update Detail' for 'Update Customer Status' with details like Name, Unique Name, Description, Object, Field to Update, and Field Data Type. Below this are sections for 'Rules Using This Field Update', 'Approval Processes Using This Field Update', and 'Entitlement Processes Using This Field Update'.

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**.



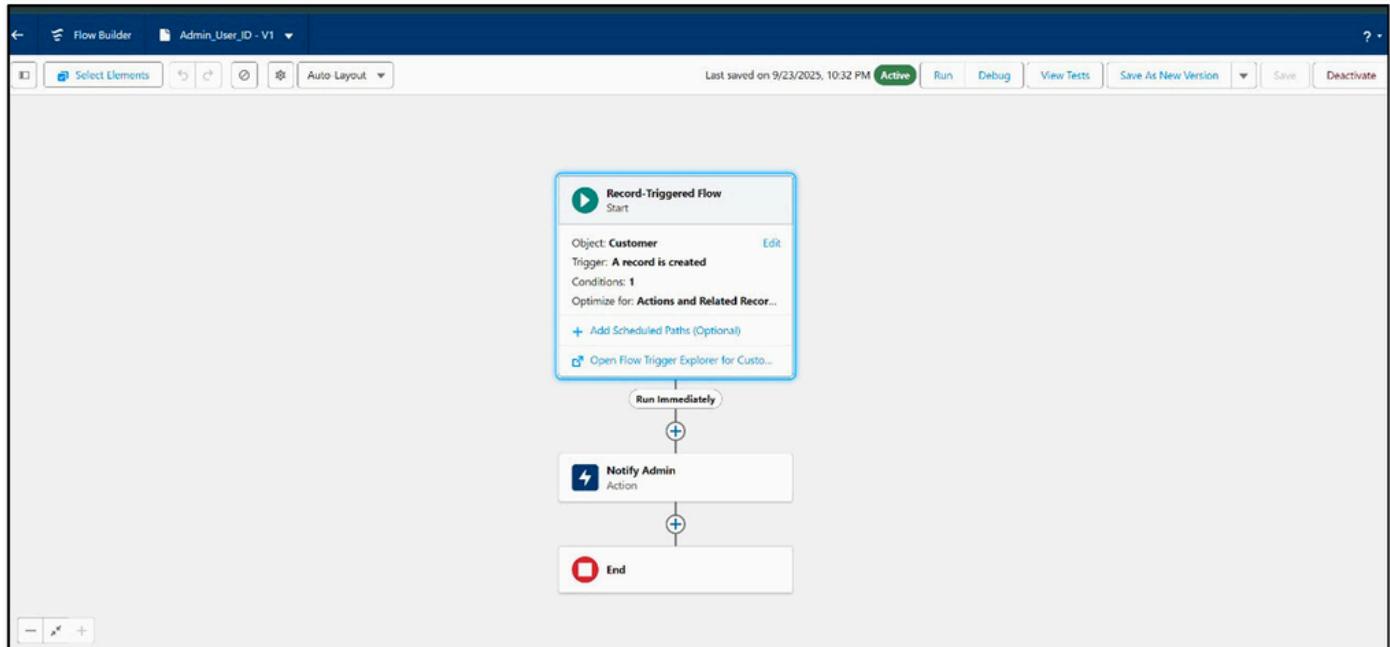
9. Custom Notifications

- 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.

Setup Home

Salesforce Go

Service Setup Assistant

Commerce Setup Assistant

Field Service Setup Home (Beta)

Hyperforce Assistant

Release Updates

Salesforce Mobile App

Lightning Usage

Optimizer

Sales Cloud Everywhere

ADMINISTRATION

> Users

> Data

Apex Triggers

CustomerTrigger

Apex Trigger Detail

Name: CustomerTrigger
Code Coverage: 0% (0/8)
Created By: Urmil shrivastava, 9/23/2025, 11:13 PM
Namespace Prefix:

sObject Type: Customer
Status: Active
Last Modified By: Urmil shrivastava, 9/23/2025, 11:13 PM

Help for this Page

```
trigger CustomerTrigger on Customer__c (before insert, before update) {
    for(Customer__c c : Trigger.new) {
        // Set default Name if blank
        if(String.isBlank(c.Name)) {
            c.Name = 'Default Customer';
            System.debug('Name was blank. Set default.');
        }
        // Ensure Age__c is valid
        if(c.Age__c == null || c.Age__c < 0) {
            c.Age__c = 18;
            System.debug('Invalid Age. Set default 18.');
        }
        // Set Status__c based on Age__c
        if(c.Age__c >= 18) {
            c.Status__c = 'Adult';
        } else {
            c.Status__c = 'Minor';
        }
        System.debug('Customer ' + c.Name + ' processed with Status: ' + c.Status__c);
    }
}
```

Setup Home

Salesforce Go

Service Setup Assistant

Commerce Setup Assistant

Field Service Setup Home (Beta)

Hyperforce Assistant

Release Updates

Salesforce Mobile App

Lightning Usage

Optimizer

Sales Cloud Everywhere

ADMINISTRATION

> Users

> Data

Apex Triggers

```
trigger CustomerTrigger on Customer__c (before insert, before update) {
    for(Customer__c c : Trigger.new) {
        // Set default Name if blank
        if(String.isBlank(c.Name)) {
            c.Name = 'Default Customer';
            System.debug('Name was blank. Set default.');
        }
        // Ensure Age__c is valid
        if(c.Age__c == null || c.Age__c < 0) {
            c.Age__c = 18;
            System.debug('Invalid Age. Set default 18.');
        }
        // Set Status__c based on Age__c
        if(c.Age__c >= 18) {
            c.Status__c = 'Adult';
        } else {
            c.Status__c = 'Minor';
        }
        System.debug('Customer ' + c.Name + ' processed with Status: ' + c.Status__c);
    }
}
```

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?

Apex Test Execution

Help for this Page

Select Tests... Developer Console Options... View Test History

Abort

Status	Class	Result
Test Run: 2025-09-23 22:27:05, urmil.shrivastava.cs22948@agentforce.com, (1 test class run)	CustomerTest	(1/1) Test Methods Passed

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.

The screenshot shows the Salesforce Setup interface with the 'Apex' search term applied. The 'Apex Test Execution' section is highlighted in the sidebar under 'Custom Code'. The main area displays a table of test results for a run on 2025-09-23 at 22:27:05. One test, 'CustomerTest', was run and passed, with 1/1 test methods executed. A detailed table below shows the test's duration, class, method, result, error message, and stack trace.

Detail	Duration	Class	Method	Result	Error Message	Stack Trace
		CustomerTest		Passed	(1/1) test Methods Executed	

This screenshot is identical to the one above, showing the same successful test run details for 'CustomerTest' on 2025-09-23 at 22:27:05. The test passed with 1/1 method executed. The detailed table below the summary table is also present.

Detail	Duration	Class	Method	Result	Error Message	Stack Trace
		CustomerTest		Passed	(1/1) test Methods Executed	

Apex Test Execution

Click Select Tests to choose one or more Apex unit tests and run them. To see the current code coverage for an individual class or your organization, go to the Apex Classes page.

Select Tests...	Developer Console	Options...	View Test History
<input type="checkbox"/> Abort			
	Status	Class	Result
Test Run: 2025-09-23 22:27:05, urmi.shrivastava@agentforce.com, (1 test class run)			
<input checked="" type="checkbox"/> CustomerTest (1/1) Test Methods Passed			

Detail Duration Class Method Pass/Fail Error Message Stack Trace

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.

Apex Class

CustomerTriggerHandler

Apex Class Detail

Name	CustomerTriggerHandler	Status	Active
Namespace Prefix		Code Coverage	0% (0/9)
Created By	Urmi shrivas , 9/23/2025, 11:40 PM	Last Modified By	Urmi shrivas , 9/23/2025, 11:40 PM

Class Body

```

1 public class CustomerTriggerHandler {
2
3     public static void beforeInsertUpdate(List<Customer__c> customers) {
4
5         for(Customer__c c : customers) {
6
7             // Set default Name if blank
8             if(String.isBlank(c.Name)) {
9                 c.Name = 'Default Customer';
10                System.debug('Name was blank. Set default.');
11            }
12        }
13    }
14}

```

The screenshot shows the Salesforce Setup interface with the 'Apex Classes' page selected. The sidebar on the left has a search bar and categories: Email, Custom Code (with Apex Classes selected), Environments, Jobs, and Apex Flex Queue. The main area displays the following Apex code:

```

6   // Set default Name if blank
7   if(String.isBlank(c.Name)) {
8       c.Name = 'Default Customer';
9       System.debug('Name was blank. Set default.');
10  }
11
12
13  // Ensure Age__c is valid
14  if(c.Age__c == null || c.Age__c < 0) {
15      c.Age__c = 18;
16      System.debug('Invalid Age. Set default 18.');
17  }
18
19  // Set Status__c based on Age__c
20  if(c.Age__c >= 18) {
21      c.Status__c = 'Adult';
22  } else {
23      c.Status__c = 'Minor';
24  }
25
26  System.debug('Customer ' + c.Name + ' processed with Status: ' + c.Status__c);
27
28 }
29

```

Below the code are buttons: Edit, Delete, Download, Security, and Show Dependencies.

4. Apex Programming – SOQL & SOSL

Created Apex Class CustomerQueryExample to implement:

- **SOQL query** → retrieve adult customers ($\text{Age_c} \geq 18$).
- **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

Q 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

```
1 public class CustomerQueryExample {  
2     // Method using SOQL  
3     public static void queryCustomers() {  
4         List<Customer__c> adults = [SELECT Customer_Name__c, Age__c, Status__c  
5             FROM Customer__c  
6             WHERE Age__c >= 18  
7             ORDER BY Customer_Name__c];  
8         System.debug('Adult Customers: ' + adults);  
9     }  
10    // Method using SOSL  
11    public static void searchCustomers(String searchText) {  
12        List<List<SObject>> searchResults = [FIND :searchText  
13            IN ALL FIELDS  
14            RETURNING Customer__c(Customer_Name__c, Age__c, Status__c)];  
15        System.debug('Search Results: ' + searchResults);  
16    }  
17 }  
18 }  
19 }
```

Setup Home Object Manager

Q 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

```
1 public class CustomerQueryExample {  
2     // Method using SOQL  
3     public static void queryCustomers() {  
4         List<Customer__c> adults = [SELECT Customer_Name__c, Age__c, Status__c  
5             FROM Customer__c  
6             WHERE Age__c >= 18  
7             ORDER BY Customer_Name__c];  
8         System.debug('Adult Customers: ' + adults);  
9     }  
10    // Method using SOSL  
11    public static void searchCustomers(String searchText) {  
12        List<List<SObject>> searchResults = [FIND :searchText  
13            IN ALL FIELDS  
14            RETURNING Customer__c(Customer_Name__c, Age__c, Status__c)];  
15        System.debug('Search Results: ' + searchResults);  
16    }  
17 }  
18 }  
19 }
```

Setup Home Object Manager

Q 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

```
1 public class CustomerQueryExample {  
2     // Method using SOQL  
3     public static void queryCustomers() {  
4         List<Customer__c> adults = [SELECT Customer_Name__c, Age__c, Status__c  
5             FROM Customer__c  
6             WHERE Age__c >= 18  
7             ORDER BY Customer_Name__c];  
8         System.debug('Adult Customers: ' + adults);  
9     }  
10    // Method using SOSL  
11    public static void searchCustomers(String searchText) {  
12        List<List<SObject>> searchResults = [FIND :searchText  
13            IN ALL FIELDS  
14            RETURNING Customer__c(Customer_Name__c, Age__c, Status__c)];  
15        System.debug('Search Results: ' + searchResults);  
16    }  
17 }  
18 }  
19 }
```

The screenshot shows the Salesforce Setup interface with the search bar containing 'apex'. The 'Apex Classes' section is selected under 'Custom Code'. The 'Class Body' tab is active, displaying the following Apex code:

```

1 public class CustomerQueryExample {
2
3     // Method using SOQL
4     public static void queryCustomers() {
5         List<Customer__c> adults = [SELECT Customer_Name__c, Age__c, Status__c
6             FROM Customer__c
7             WHERE Age__c >= 18
8             ORDER BY Customer_Name__c];
9
10        System.debug('Adult Customers: ' + adults);
11    }
12
13    // Method using SOSL
14    public static void searchCustomers(String searchText) {
15        List<List<SObject>> searchResults = [FIND :searchText
16            IN ALL FIELDS
17            RETURNING Customer__c(Customer_Name__c, Age__c, Status__c)];
18
19        System.debug('Search Results: ' + searchResults);
20    }
21

```

The sidebar on the left lists various Apex categories like Email, Custom Code, Environments, and Jobs.

This screenshot is identical to the one above, showing the same Apex code for the CustomerQueryExample class in the Salesforce Setup interface.

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

The screenshot shows a Salesforce App Page titled "Customer_List_Page". The page header includes the organization name "TCS_LM_SF" and the user "Nitesh tiwari". The main content area displays a list of "All Customers" with the following data:

	Total Life Time Value	Name
1	10,000	John Doe
2	20,000	Shivi
3	250,000	Nitesh tiwari

There are search and filter tools at the top right of the list view.

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.

... > OBJECT MANAGER > CUSTOM
Record Page Assignments

Details

Fields & Relationships

Page Layouts

Lightning Record Pages

Buttons, Links, and Actions

Compact Layouts

Field Sets

Object Limits

Record Types

Related Lookup Filters

Search Layouts

List View Button Layout

Object Access

Triggers

Flow Triggers

Validation Rules

Conditional Field Formatting

ABOUT ASSIGNMENTS

Custom record pages can be assigned at different levels:

The org default record page displays for an object unless more specific assignments are made.

App default page assignment, if specified, overrides the org default.

App, record type, profile assignments override org and app defaults.

Learn more about Lightning page assignment.

Look Up an Assignment

See what record page is shown for an app, record type, and profile combination.

*App: TCS_LM_SF

*Record Type: Master

*Profile: Customer Community Login User

Find Page

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

Custom Tabs

You can create new custom tabs to extend Salesforce functionality or to build new application functionality.

Custom Object tabs look and behave like the standard tabs provided with Salesforce. Web tabs allow you to embed external web applications and content within the Salesforce window. Visualforce tabs allow you Lightning components to the navigation menu in Lightning Experience and the mobile app. Lightning Page tabs allow you to add Lightning Pages to Lightning Experience and the mobile app.

Custom Object Tabs		New	What Is This?
Action	Label	Tab Style	
Edit Del	Customer_Address_c		Handsaw
Edit Del	Customers		Chess piece
Edit Del	Order Details		Cell phone
Edit Del	Payments		Computer
Edit Del	Service ProvidersStores local service pr		CD/DVD

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:
 - o Report Charts → Display key metrics
 - o List Views → Quick access to records
 - o Rich Text / Image Components → Display announcements or instructions
 - o Tabs (optional) → For better organization

Save & Activate:

- Assigned as Org Default or specific to App Profiles

Assignments by App and Profile			Set App and Profile Assignments
APP	PROFILE	LIGHTNING PAGE	
Approvals	Contract Manager	Customer_Home_Page	<input type="button" value="▼"/>
Approvals	Custom Standard User	Customer_Home_Page	<input type="button" value="▼"/>
Approvals	Custom: Support Profile	Customer_Home_Page	<input type="button" value="▼"/>
Service Console	Contract Manager	Customer_Home_Page	<input type="button" value="▼"/>
Service Console	Custom Standard User	Customer_Home_Page	<input type="button" value="▼"/>
Service Console	Custom: Support Profile	Customer_Home_Page	<input type="button" value="▼"/>
Salesforce CMS	Contract Manager	Customer_Home_Page	<input type="button" value="▼"/>
Salesforce CMS	Custom Standard User	Customer_Home_Page	<input type="button" value="▼"/>
Salesforce CMS	Custom: Support Profile	Customer_Home_Page	<input type="button" value="▼"/>
TCS_LM_SF	Contract Manager	Customer_Home_Page	<input type="button" value="▼"/>
TCS_LM_SF	Custom Standard User	Customer_Home_Page	<input type="button" value="▼"/>
TCS_LM_SF	Custom: Support Profile	Customer_Home_Page	<input type="button" value="▼"/>

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

The screenshot shows the 'Utility Items (Desktop Only)' section of the Lightning App Builder. On the left, a sidebar lists 'App Settings' options: App Details & Branding, App Options, Utility Items (Desktop Only), Navigation Items, and User Profiles. The 'Utility Items (Desktop Only)' option is selected and highlighted with a blue background. The main content area is titled 'Utility Items (Desktop Only)' and contains the sub-instruction: 'Give your users quick access to productivity tools and add background utility items to your app.' Below this, there's a 'Utility Bar Alignment' dropdown set to 'Default'. A large central panel displays a list titled 'To Do List' containing 'History' and 'Notes'. To the right of the list are 'PROPERTIES' for 'To Do List' (with up/down arrows and a 'Remove' button) and 'Utility Item Properties' settings. These properties include: Label ('To Do List'), Icon ('task'), Panel Width (427), Panel Height (680), and a checked checkbox for 'Start automatically'.

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.

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "DEBUGGER-PROJECT". It includes ".vscode", "config", "force-app\main\default", "aura", "classes" (with "AccountService.cls" and "AccountService.cls-meta.xml"), "contentassets", "flexpages", "layouts", "lwc\accountManager" (with "_tests_"), and "objects", "permissionsets".
- Search Bar:** Displays "debugger-project".
- Code Editor (Right):** The active file is "accountManager.test.js". The code is a Jest test for a Lightning Web Component (LWC). It imports "createElement" from "@lwc/engine-dom" and "AccountManager" from "c/accountManager". The test uses "describe", "afterEach", and "it" blocks. A TODO comment indicates the need to fill in test logic.

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.

: Manager

The screenshot shows the Apex Classes page in the Salesforce setup. The class is named "AccountServiceTest". It is annotated with @IsTest and contains two test methods: testCreateAndUpdateAccount and testGetAccountDetails. Both methods use AccountService.createAccount and AccountService.getAccountDetails respectively, and perform assertions on the returned account objects.

```
1 @IsTest
2 private class AccountServiceTest {
3
4     @IsTest
5     static void testCreateAndUpdateAccount() {
6         Id accId = AccountService.createAccount('Test Customer', 'Customer', '1234567890');
7         Account acc = [SELECT Id, Name, Phone FROM Account WHERE Id = :accId];
8         System.assertEquals('Test Customer', acc.Name);
9
10        AccountService.updateAccount(accId, 'Updated Name', '0987654321');
11        acc = [SELECT Name, Phone FROM Account WHERE Id = :accId];
12        System.assertEquals('Updated Name', acc.Name);
13    }
14
15    @IsTest
16    static void testGetAccountDetails() {
17        Account acc = new Account(Name='Test Provider', Type='Service Provider', Phone='1112223333');
18        insert acc;
19        Account result = AccountService.getAccountDetails(acc.Id);
20        System.assertEquals('Test Provider', result.Name);
21    }
22 }
```

: Manager

This screenshot is identical to the one above, showing the same Apex class "AccountServiceTest" with its two test methods. The code remains the same, demonstrating the separation of concerns between the frontend LWC and the backend Apex logic.

```
1 @IsTest
2 private class AccountServiceTest {
3
4     @IsTest
5     static void testCreateAndUpdateAccount() {
6         Id accId = AccountService.createAccount('Test Customer', 'Customer', '1234567890');
7         Account acc = [SELECT Id, Name, Phone FROM Account WHERE Id = :accId];
8         System.assertEquals('Test Customer', acc.Name);
9
10        AccountService.updateAccount(accId, 'Updated Name', '0987654321');
11        acc = [SELECT Name, Phone FROM Account WHERE Id = :accId];
12        System.assertEquals('Updated Name', acc.Name);
13    }
14
15    @IsTest
16    static void testGetAccountDetails() {
17        Account acc = new Account(Name='Test Provider', Type='Service Provider', Phone='1112223333');
18        insert acc;
19        Account result = AccountService.getAccountDetails(acc.Id);
20        System.assertEquals('Test Provider', result.Name);
21    }
22 }
```

Outcome:

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

8. Events in LWC

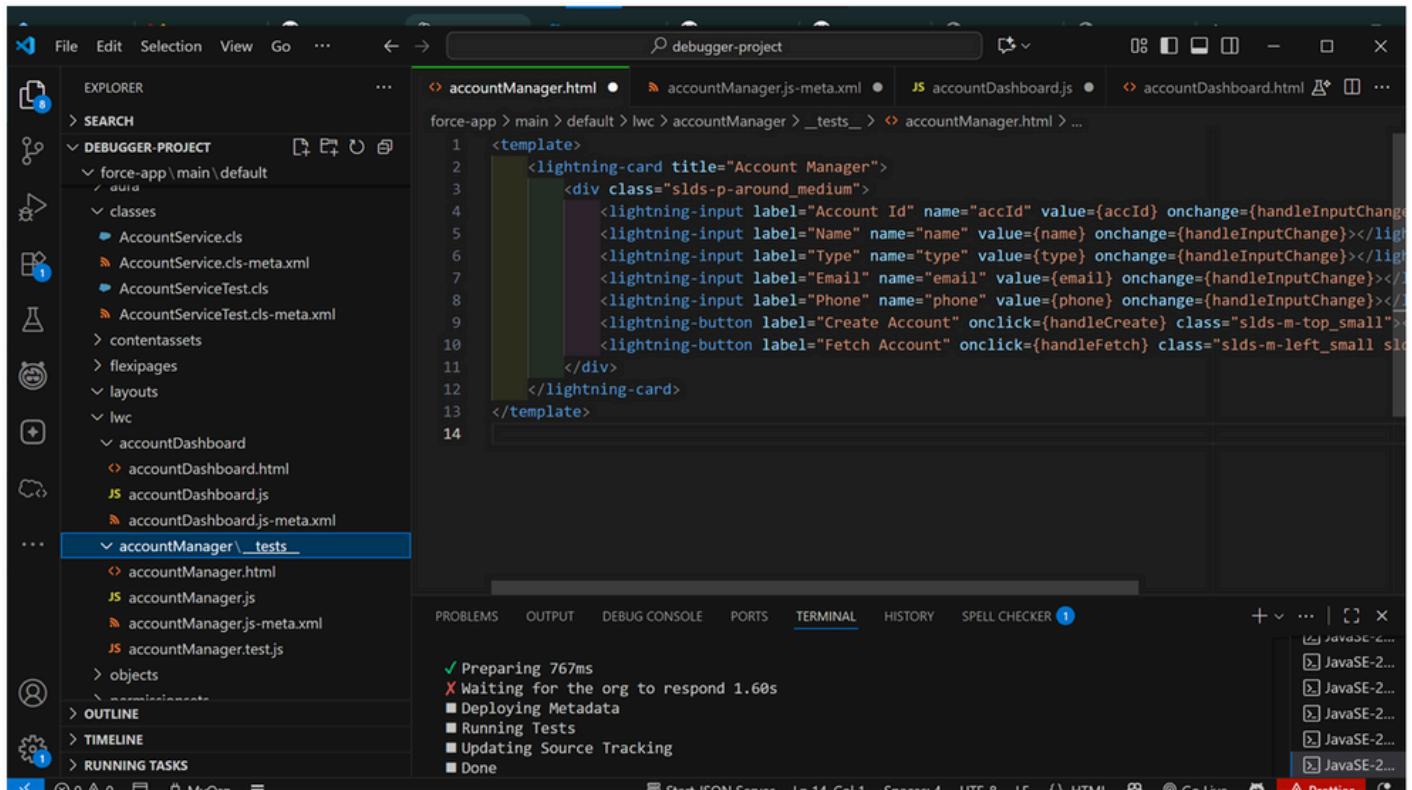
Purpose:

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.



The screenshot shows the Salesforce IDE interface with the following details:

- EXPLORER** panel on the left: Shows the project structure under "DEBUGGER-PROJECT". It includes "classes" (AccountService.cls, AccountService.cls-meta.xml, AccountServiceTest.cls, AccountServiceTest.cls-meta.xml), "contentassets", "flexipages", "layouts", "lwc" (accountDashboard, accountDashboard.html, accountDashboard.js, accountDashboard.js-meta.xml), and "accountManager" (accountManager.html, accountManager.js, accountManager.js-meta.xml, accountManager.test.js).
- CODE EDITOR**: The main editor pane displays the code for "accountManager.html". The code defines a lightning-card component with a title "Account Manager". Inside, there is a form with six lightning-input fields for Account Id, Name, Type, Email, and Phone, each with an onchange event handler. A lightning-button labeled "Create Account" has an onclick event handler. The component uses SLDs classes like "slds-p-around_medium" and "slds-m-top_small".
- TERMINAL**: The bottom right shows the terminal output with the following logs:
 - ✓ Preparing 767ms
 - ✗ Waiting for the org to respond 1.60s
 - Deploying Metadata
 - Running Tests
 - Updating Source Tracking
 - Done

- Parent Component (accountDashboard):

- Listens for the accountcreated event.
- Updates and displays the newly created account dynamically.

```
<template>
  <lightning-card title="Account Dashboard">
    <c-account-manager onaccountcreated={handleAccountCreated}></c-account-manager>
    <!-- Display created accounts -->
    <template if:true={createdAccounts.length}>
      <h3 class="slds-m-top_medium">Created Accounts:</h3>
      <ul>
        <template for:each={createdAccounts} for:item="acc">
          <li key={acc.id}>{acc.name} (Id: {acc.id})</li>
        </template>
      </ul>
    </template>
  </lightning-card>
</template>
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL HISTORY SPELL CHECKER 1

✓ Preparing 767ms
✗ Waiting for the org to respond 1.60s
■ Deploying Metadata
■ Running Tests
■ Updating Source Tracking
■ Done

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.

```
4  export default class AccountManager extends LightningElement {  
5      @track accId;  
6      @track account;  
7      @track error;  
8  
9      handleInputChange(event) {  
10         this.accId = event.target.value;  
11     }  
12  
13     @wire(getAccountDetails, { accountId: '$accId' })  
14     wiredAccount({ error, data }) {  
15         if(data) {  
16             this.account = data;  
17             this.error = undefined;  
18         } else if(error) {  
19             this.account = undefined;  
20             this.error = error;  
21         }  
22     }  
23 }
```

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.

```
4  export default class AccountManager extends LightningElement {  
12 }  
13  
14 handleCreateAccount() {  
15     // Imperative Apex Call  
16     createAccount({ name: this.name, type: this.type, phone: this.phone })  
17         .then(result => {  
18             this.createdAccountId = result;  
19             console.log('Account created with Id: ' + result);  
20             // Optionally, dispatch an event to parent  
21             const createdEvent = new CustomEvent('accountcreated', { detail: { id: result } });  
22             this.dispatchEvent(createdEvent);  
23  
24             // Reset inputs  
25             this.name = '';  
26             this.type = '';  
27             this.phone = '';  
28         })  
29         .catch(error => {  
30             console.error('Error creating account:', error);  
31     })  
}:
```

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.

Bulk Data Load Jobs												
Job ID	750gL00000FJadR											
Submitted By	Urmi.shrivastav											
Start Time	10/11/2025, 1:11 AM PST											
End Time	10/11/2025, 1:11 AM PST											
Time to Complete ([hh:]mm:ss)	00:01											
Object	Contact											
External ID Field	Id											
Content Type	CSV											
Concurrency Mode	Parallel											
API Version	65.0											
Job Type	Bulk V1											
Operation	Upsert											
Status	Closed											
Total Processing Time (ms)	695											
API Active Processing Time (ms)	488											
Apex Processing Time (ms)	183											
Completed Batches	1											
Failed Batches	0											
Progress	100%											
Records Processed	20											
Records Failed	0											
Retries	0											
Reload												
Batches												
View Request	View Result	Batch ID	Start Time	End Time	Total Processing Time (ms)	API Active Processing Time (ms)	Apex Processing Time (ms)	Records Processed	Records Failed	Retry Count	State Message	Status
View Request	View Result	751gL00000Cc2ec	10/11/2025, 1:11 AM	10/11/2025, 1:11 AM	695	488	183	20	0	0	Completed	

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.

D SETUP Duplicate Rules

Contact Duplicate Rule
Standard Contact Duplicate Rule

Help for this Page

Duplicate Rule Detail

[Edit](#) [Delete](#) [Clone](#) [Deactivate](#)

Order 1 of 1 [Reorder]

Rule Name	Standard Contact Duplicate Rule	Operations On Create	<input checked="" type="checkbox"/> Alert <input checked="" type="checkbox"/> Report
Description	Identify contacts that duplicate other contacts and leads.	Operations On Edit	<input type="checkbox"/> Alert <input checked="" type="checkbox"/> Report
Object	Contact		
Record-Level Security	Enforce sharing rules		
Action On Create	Allow		
Action On Edit	Allow		
Alert Text	Use one of these records?	Matching Criteria	Matching rule for contact records. More info
Active	<input checked="" type="checkbox"/>	Matching Criteria	Matching rule for lead records. More info
Matching Rule	Standard.Contact Matching Rule Mapped		
Matching Rule	Standard.Lead Matching Rule Mapped		
Conditions			
Created By	OrgFarm_EPIC, 7/17/2025, 12:23 AM	Modified By	OrgFarm_EPIC, 7/17/2025, 12:23 AM

[Edit](#) [Delete](#) [Clone](#) [Deactivate](#)

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.

Next scheduled export:
None

[Export Now](#) [Schedule Export](#)

Scheduled By Urmishrivastava

Schedule Date 10/11/2025

Export File Encoding ISO-8859-1 (General US & Western European, ISO-LATIN-1)

Action	File Name	File Size
download	WE_00DgL000007QGpxUAG_1.ZIP	254.6K

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)**.

```
force-app > main > default > lwc > accountManager > __tests__ > JS accountManager.js > AccountManager
  4  export default class AccountManager extends LightningElement {
  5    handleInputChange(event) {
  6      this[event.target.name] = event.target.value;
  7    }
  8
  9    handleCreateAccount() {
 10      // Imperative Apex Call
 11      createAccount({ name: this.name, type: this.type, phone: this.phone })
 12      .then(result => {
 13        this.createdAccountId = result;
 14        console.log('Account created with Id: ' + result);
 15        // Optionally, dispatch an event to parent
 16        const createdEvent = new CustomEvent('accountcreated', { detail: { id: result, name: this.name } });
 17        this.dispatchEvent(createdEvent);
 18
 19        // Reset inputs
 20        this.name = '';
 21        this.type = '';
 22      })
 23      .catch(error => {
 24        console.error('Error creating account:', error);
 25      });
 26    }
 27  }
 28
 29
 30
 31
 32
 33
 34
```

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.

Recent							
REPORTS	Report Name	Description	Folder	Created By	Created On	Subscribed	
Recent	Accounts by Industry	Categorizes accounts by industry. Helps in business analysis, identifying industry trends, and making strategic decisions for account management.	CRM Project Reports	Urmi shrivas	10/11/2025, 7:33 AM		<input type="button" value="View"/>
Created by Me	All Accounts List	Shows all contacts along with their linked accounts. Helps track client communication points and ensures no contact is missed for outreach or follow-ups.	CRM Project Reports	Urmi shrivas	10/11/2025, 7:28 AM		<input type="button" value="View"/>
Private Reports	Contacts by Account	Groups contacts under each account. Useful to understand client relationships and manage communications efficiently for each account.	CRM Project Reports	Urmi shrivas	10/11/2025, 7:30 AM		<input type="button" value="View"/>
Public Reports	Contacts by Account	CRM Project Reports	Urmi shrivas	10/11/2025, 7:14 AM			<input type="button" value="View"/>
All Reports	Sample Flow Report: Screen Flows	Which flows run, what's the status of each interview, and how long do users take to complete the screens?	Public Reports	Automated Process	7/17/2025, 12:23 AM		<input type="button" value="View"/>
FOLDERS							
All Folders							<input type="button" value="View"/>
Created by Me							<input type="button" value="View"/>
Shared with Me							<input type="button" value="View"/>
FAVORITES							<input type="button" value="View"/>
All Favorites							

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:**

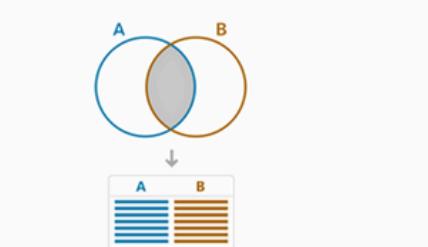
- Custom report types were configured to pull accurate data from **Accounts** and **Contacts**.
- Enabled grouping, filtering, and detailed analysis specific to the CRM project's needs.
- Ensured that only relevant fields were visible to maintain data consistency.

information for the custom report type

Details

Display Label	Accounts with Opportunities
API Name	Accounts_with_Opportunities
Description	Shows accounts along with their opportunities for project analysis
Created By	Urmi shrivas, 10/11/25, 8:09 PM
Store in Categ...	accounts
Deployment St...	Deployed
Modified By	Urmi shrivas, 10/11/25, 8:11 PM

Object Relationships

Accounts (A)  with at least one related record from Contracts (B)



Fields

Source Object	Included Fields
Accounts	63
Contracts	0

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**.

The screenshot shows a Salesforce Lightning Project Dashboard titled "CRM Project Dashboard". At the top, there's a navigation bar with links like "Lightning Usage App", "Lightning Usage", "Order Details", "Customers", "Payments", "Service Providers", "Stores", "Customer_Address_c", and "Recent | Dashboards". Below the navigation is a search bar and a toolbar with icons for "Widget", "Filter", and others.

The main area contains two report cards:

- Contacts by Account:** A table with columns: First Name, Last Name, Account Name, Mailing Address, and Mailing State/Province. The data includes rows for Alice Black, Basil Wenceslas, Dragon Davich, George Dapper, Jack Dodge, Jessica Jones, and John Downes.
- Accounts by Industry:** A table with columns: Last Name, Account Name, Billing State/Province, and Type. The data includes rows for Urmishrivas (John Doe and Jane Smith) and Urmishrivas (Account Name: Local Business Customers).

At the bottom of each report card is a link to "View Report".

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.

The screenshot shows the 'Sharing Settings' page in Salesforce. At the top, there's a 'Default Sharing Settings' section with 'Organization-Wide Defaults'. It includes a table for 'Object' (Account and Contract) with 'Default Internal Access' (Public Read/Write) and 'Default External Access' (Private). A checkbox for 'Grant Access Using Hierarchies' is checked. Below this is an 'Other Settings' section with checkboxes for 'Manager Groups', 'Secure guest user record access' (checked), and 'Require permission to view record names in lookup fields'. Under 'Sharing Rules', there's a 'Account Sharing Rules' section with a table for 'Action' (Edit | Del) and 'Criteria' (Owner in Role: Service Manager, Owner in Group: Service Providers). The 'Shared With' column lists 'Role: Customer manager' and 'Group: Service Providers'. The 'Account and Contract' row shows 'Read Only' access, while 'Opportunity' and 'Case' show 'Read/Write' and 'Private' respectively. There are 'New' and 'Recalculate' buttons at the top of the rules table.

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.