

GARAGE MANAGEMENT APPLICATION

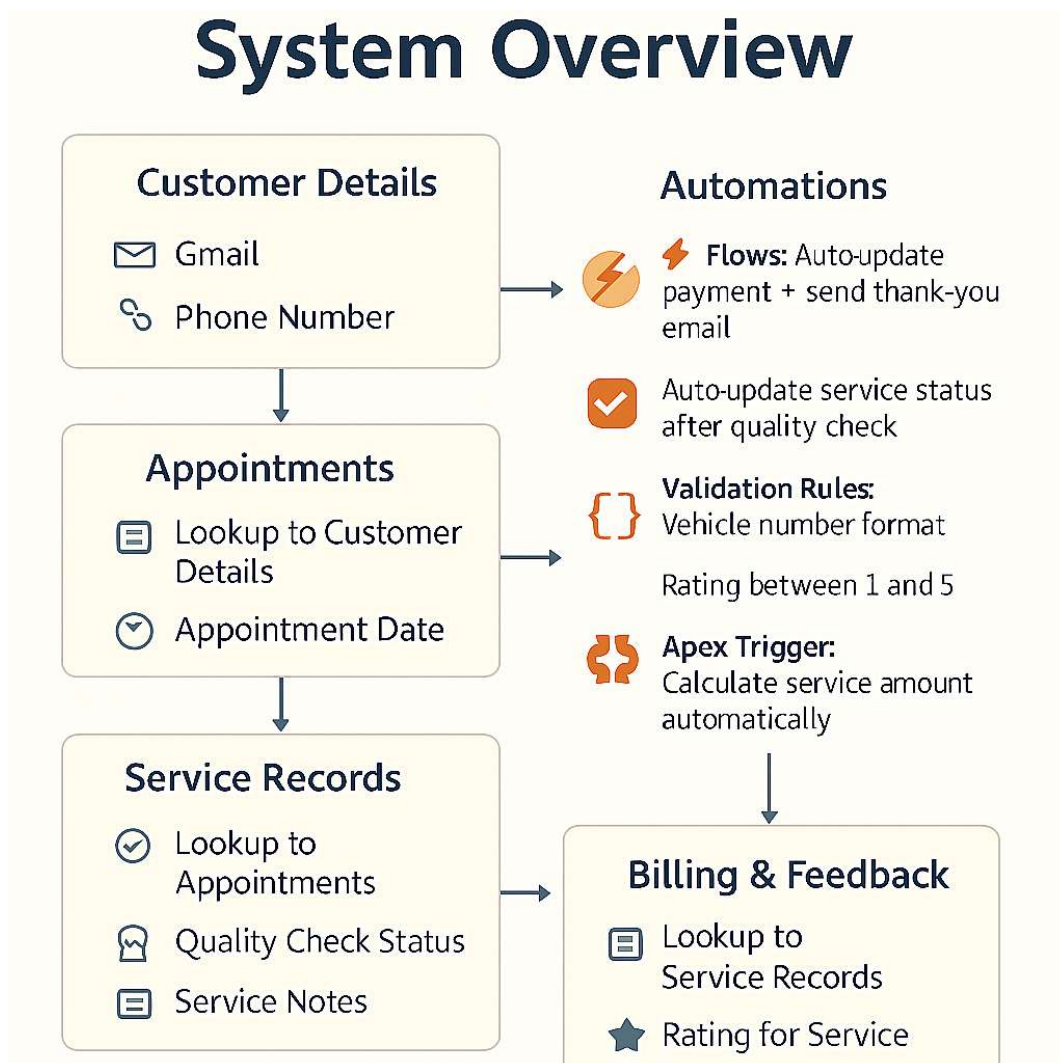
By

Reddyvari Sindhu

from

Annamacharya Institute of Technology & Sciences,
Utukur, C.K.Dinne, Kadapa.

System Overview



Project Overview

The **Garage Management CRM** is a custom-built solution developed on the Salesforce platform to modernize and automate vehicle servicing operations. The system centralizes critical garage functions such as “customer management”, “appointment scheduling”, “vehicle tracking”, and “service history maintenance”. It enables garage staff to efficiently handle mechanic assignments, service requests, billing, and customer feedback in one place. By using “custom objects”, “flows”, and “validation rules”, the CRM reduces manual workload and increases operational accuracy.

Automation is a key strength of the system, with features like “automatic service record creation”, “billing updates”, and “email notifications” triggered through flows. Role-based access control ensures that data is securely handled by the right users, such as Managers and Salespersons. The CRM also supports decision-making through “real-time reports and dashboards”, offering insights into customer satisfaction, billing status, and service trends. The user interface is intuitive and tailored using Lightning App Builder with dynamic layouts. Ultimately, the system transforms traditional paper-based workflows into an efficient, digital-first process that improves both service delivery and business growth.

Objectives

- To digitize garage operations and reduce dependency on manual record-keeping:

The CRM replaces paper-based processes with a centralized digital platform. This improves efficiency, data accuracy, and long-term record management.

- To improve customer management by maintaining detailed vehicle and service histories:

All customer interactions, vehicle details, and service records are stored in one place.

This supports better follow-ups and personalized service.

- To streamline appointment bookings, mechanic assignments, and job tracking:

Appointments are booked with proper time management, and mechanics are auto-assigned. This reduces overlap, delays, and confusion.

- To implement automation for tasks like service creation, approvals, and notifications:

Flows and triggers automate routine processes like service record generation and email alerts. This saves time and minimizes human error.

- To enhance service quality and operational efficiency, leading to better customer satisfaction and business growth:

Faster workflows and improved communication increase customer trust. This results in higher retention and supports business expansion.

Phase 1: Requirement Analysis & Planning

Understanding Business Requirements

The garage operates with multiple departments handling bookings, repairs, billing, and customer follow-ups. Previously, these processes were managed using physical registers or spreadsheets, leading to miscommunication, scheduling conflicts, and data loss. The business required a centralized CRM to efficiently manage customer data, service appointments, mechanic assignments, billing, and feedback. By digitizing these functions, the goal was to improve service delivery speed, ensure data consistency, and enhance the overall customer experience through automation.

Defining Project Scope and Objectives

The project scope includes core modules such as “Customer Management”, “Appointment Scheduling”, “Service Tracking”, “Billing & Payments”, and “Customer Feedback”. It also covers automation features like service record creation, email alerts after service, and approval processes for high-value bills. The CRM is designed to streamline job allocation, track performance, and maintain detailed records for transparency and audit.

Excluded from scope are external payment gateway integration and mobile app development, which are planned as future enhancements.

Designing the Data Model and Security Model

Five custom objects were designed to capture business needs:

- Customer_Details__c: Holds contact and vehicle information.
- Appointment__c: Manages bookings and scheduling.
- Service_Records__c: Tracks work done and service outcomes.
- Billing_Feedback__c: Combines payment status and customer satisfaction data.

Relationships were created between these objects using “lookup fields”, ensuring data integrity and logical flow. The “security model” includes a clearly defined role hierarchy: “Admin > Manager > Salesperson”, with custom profiles granting appropriate object-level and field-level access. “Permission sets” were used to extend access for specific tasks like billing approvals. The system uses “private OWD settings” with “sharing rules” to allow role-based data sharing between salespersons and managers.

Phase 2: Salesforce Development - Backend & Configurations

Environment Setup & DevOps Summary

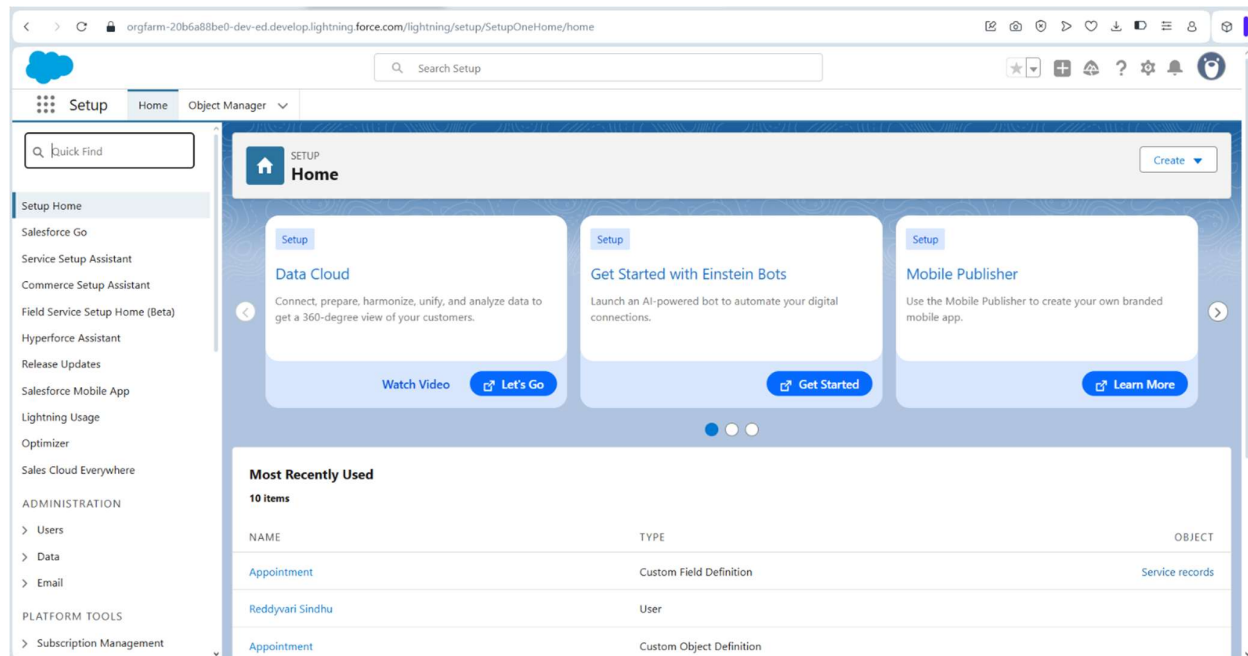
A Salesforce Developer Org was created by signing up at [developer.salesforce.com/signup] (<https://developer.salesforce.com/signup>).

The signup form required details like Name, Email, Role (Developer), College Name, Country, PIN Code, and a custom Username in email format.

After submission, a verification email was received and used to activate the account.

Upon verification, a new password was set and a security question answered to complete account setup.

The user was then redirected to the Salesforce Setup page (Lightning Experience).



This org served as the development and testing environment for the entire Garage Management CRM project.

All custom objects, fields, flows, triggers, and validation rules were configured within this org.

No separate Sandbox or production environment was used due to the project's academic/demo scope.

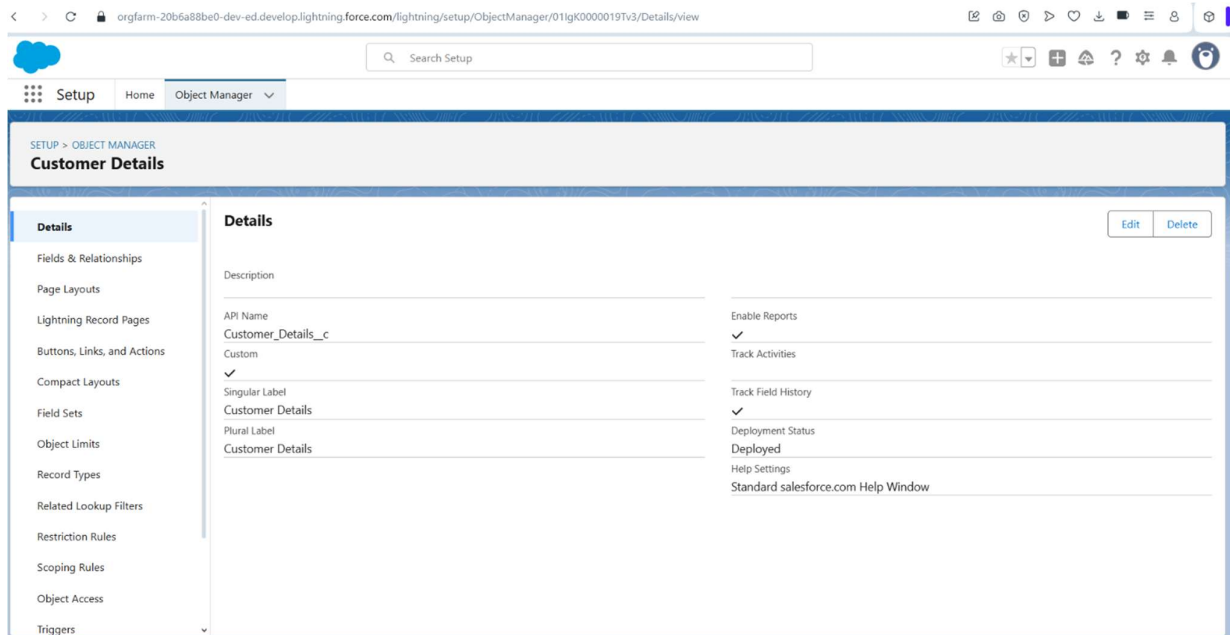
For versioning and deployment simulation, best practices like organizing metadata and modular testing were followed.

Change Sets or Unmanaged Packages can be used in real scenarios for DevOps and deployment to other environments.

Custom Objects Information

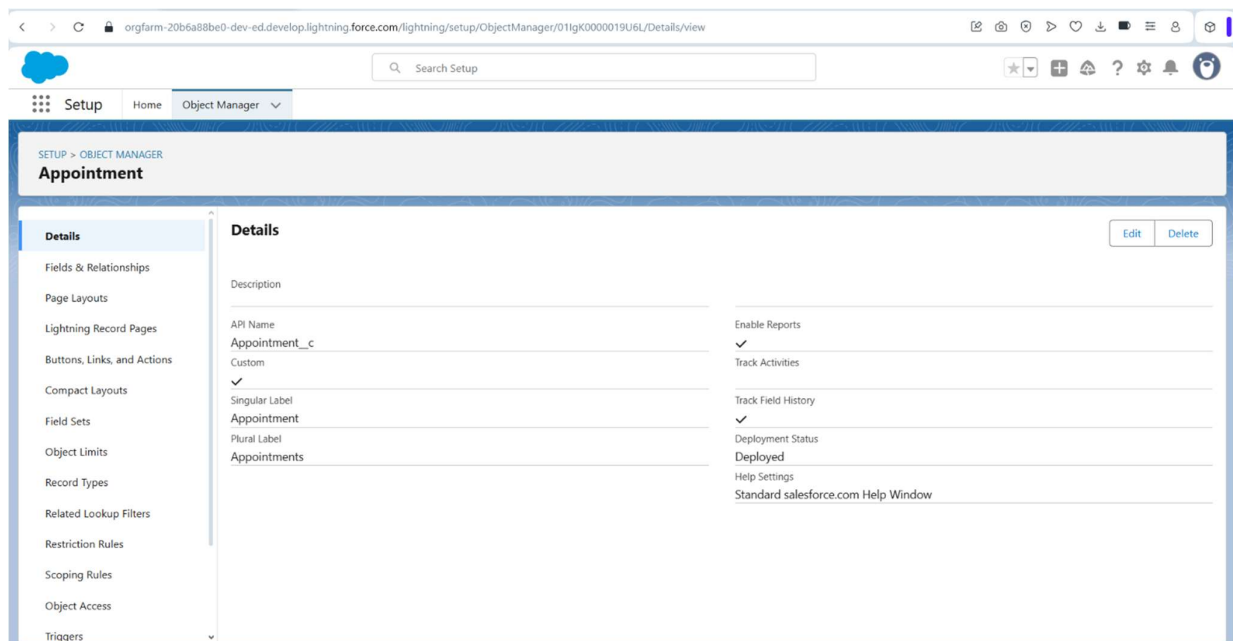
1. Customer_Details__c

This object captures essential customer information including Name, Phone Number, Email, and Vehicle Details such as Vehicle Number, Brand, and Model. It serves as the parent record for customer-related operations. Other objects like `Appointment__c` reference this object using a Lookup relationship, ensuring that all appointments and service records are linked to the appropriate customer. This object supports the identification of repeat customers and helps maintain service history by customer and vehicle.



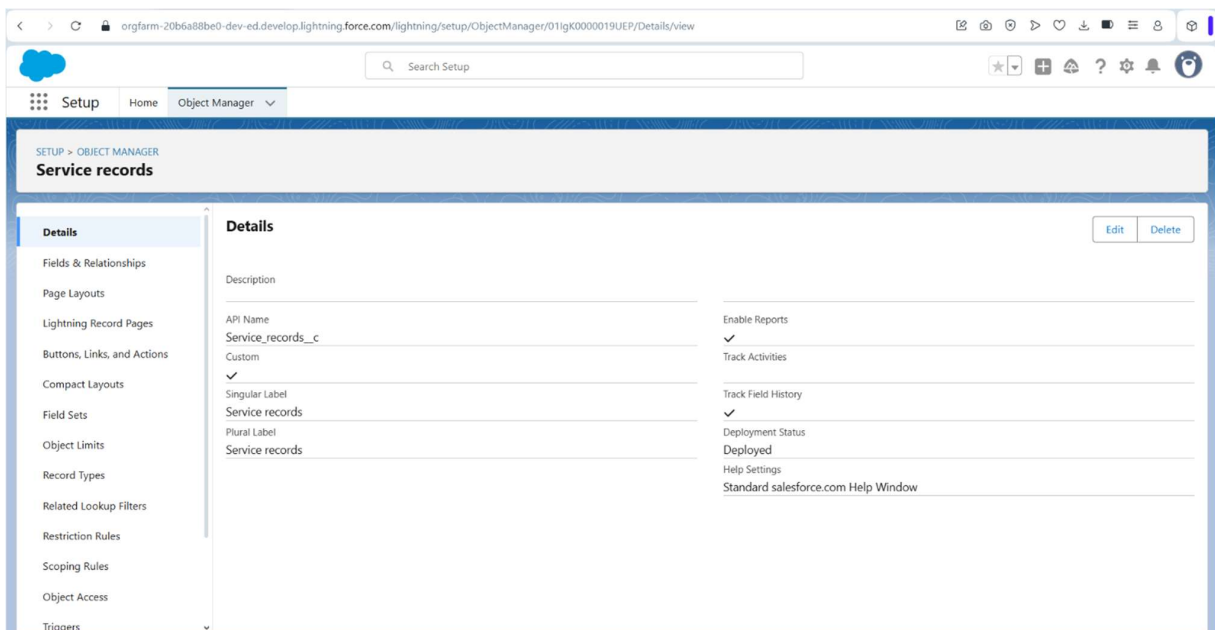
2. Appointment__c

This object is used to manage service booking details. Key fields include Appointment Date, Appointment Status, and checkboxes like Maintenance Service, Repairs, and Replacement Parts. It has a Lookup to 'Customer_Details__c', connecting each appointment to the respective customer. It also has a Lookup to 'Service_Records__c', allowing users to track service delivery based on the appointment. Automation such as mechanic assignment and service amount calculation is triggered when this record is created or updated.



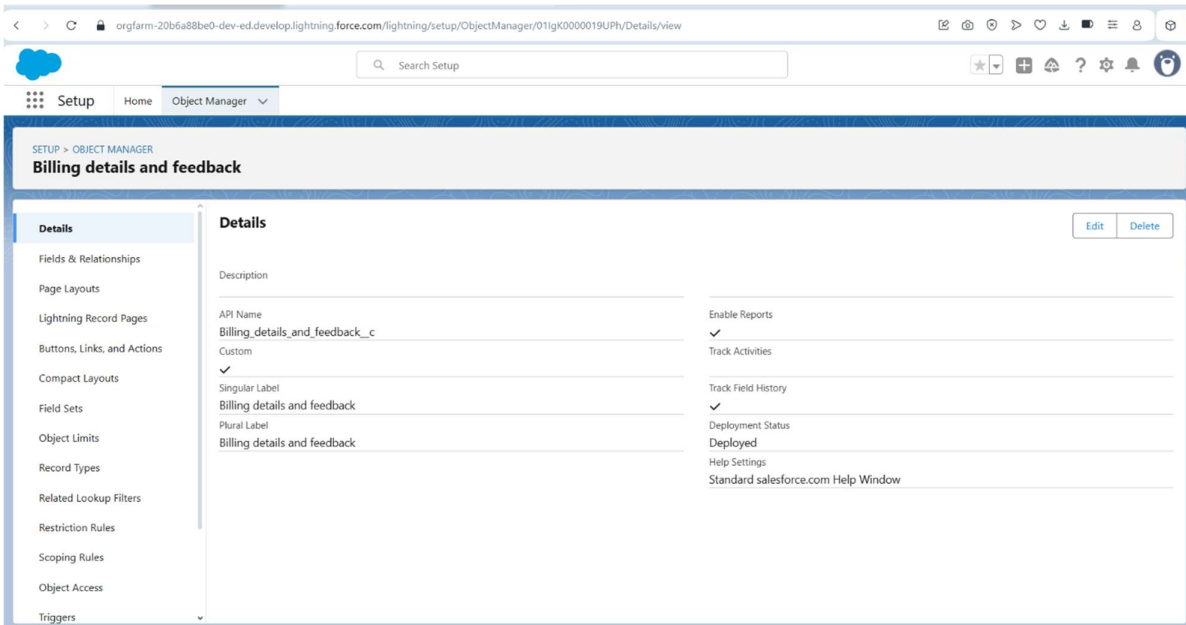
3. Service_Records__c

This object logs the actual service provided to the customer's vehicle. It contains fields like Service Type, Service Date, Mechanic Assigned, and Service Completed (checkbox). It is linked via a Lookup to Appointment__c, allowing traceability from service back to the appointment and customer. It also plays a key role in automation — when a service is marked completed, this can trigger flows to update billing or customer feedback components.



4. Billing_Feedback__c

A combined object that manages both payment and customer feedback. Fields include Billing Amount, Payment Status, Payment Paid, Feedback Rating, and Customer Comments. It connects to Service_Records__c through a Lookup field, allowing billing and feedback to be traced back to the specific service performed. Flows use this object to send automated thank-you emails and update payment information when service is complete.



Field Types

1. Lookup Fields

Used to establish relationships between different objects.

Example Use Cases:

- * Appointment__c has a lookup to Customer_Details__c to link each appointment to a specific customer.
- * Service_Records__c has a lookup to Appointment__c for tracking service against bookings.
- * Billing_Feedback__c has a lookup to Service_Records__c for linking feedback and billing to completed services.

These fields help maintain data integrity and enable cross-object reporting and automation.

Checkbox Fields

Used for binary selections like Yes/No or True/False.

Example Use Cases:

- * Service_Completed__c on Service_Records__c indicates whether the job is done.
- * Bill_Paid__c on Billing_Feedback__c flags if the customer has cleared their dues.

These are commonly used in flows, visibility rules, and status tracking.

Date Fields

Capture important calendar-based inputs.

Example Use Cases:

- * Appointment_Date__c on Appointment__c for scheduling services.
- * Service_Date__c on Service_Records__c to record when service was performed.

Date fields are used in reports, automations, and reminders.

Currency Fields

Store financial amounts like charges and payments.

Example Use Cases:

- `Service_Amount__c` on `Appointment__c` to store total billing based on selected services.
- `Payment_Paid__c` on `Billing_Feedback__c` to track actual payment made.

Currency fields support invoice generation, revenue analytics, and approval flows.

Text Fields

Used to enter and store free-form text.

Example Use Cases:

- `Vehicle_Number__c` on `Customer_Details__c` stores vehicle registration details.
- `Mechanic_Notes__c` on `Service_Records__c` records any technician comments.

These fields offer flexibility for detailed data capture.

Picklist Fields

Allow users to choose from a set list of values.

Example Use Cases:

- `Service_Type__c` (e.g., Oil Change, Engine Repair) on `Service_Records__c`.
- `Appointment_Status__c` (e.g., Scheduled, Completed) on `Appointment__c`.
- `Feedback_Rating__c` (e.g., Excellent, Good, Poor) on `Billing_Feedback__c`.

Picklists support consistent data, filters, and conditional visibility.

Formula Fields

Auto-calculate values based on logic and other field inputs.

Example Use Cases:

`Final_Amount__c` formula that calculates total cost including tax.

`Status_Display__c` to show a user-friendly status message (e.g., “Pending Payment”) based on logic.

Formula fields reduce manual effort and provide real-time computed values across records.

Absolutely! Here's an **extended and detailed version** of your validation rule summary, explaining both the business logic and technical purpose behind them:

Validation Rules

1. Vehicle Number Plate Validation (`Appointment__c`)

This rule ensures that the Vehicle Number Plate entered by the user follows a standardized Indian format (e.g., `AP31AB1234`). It uses a REGEX formula to match the format `[A-Z]{2}[0-9]{2}[A-Z]{2}[0-9]{4}`.

If the pattern doesn't match, an error message is shown:

“Please enter valid number.”

The screenshot displays the Salesforce Setup interface for the 'Appointment' object. The left sidebar shows the navigation menu with 'Setup' selected. The main content area is titled 'Appointment Validation Rule' and includes a 'Back to Appointment' link. The 'Validation Rule Detail' section shows the following information:

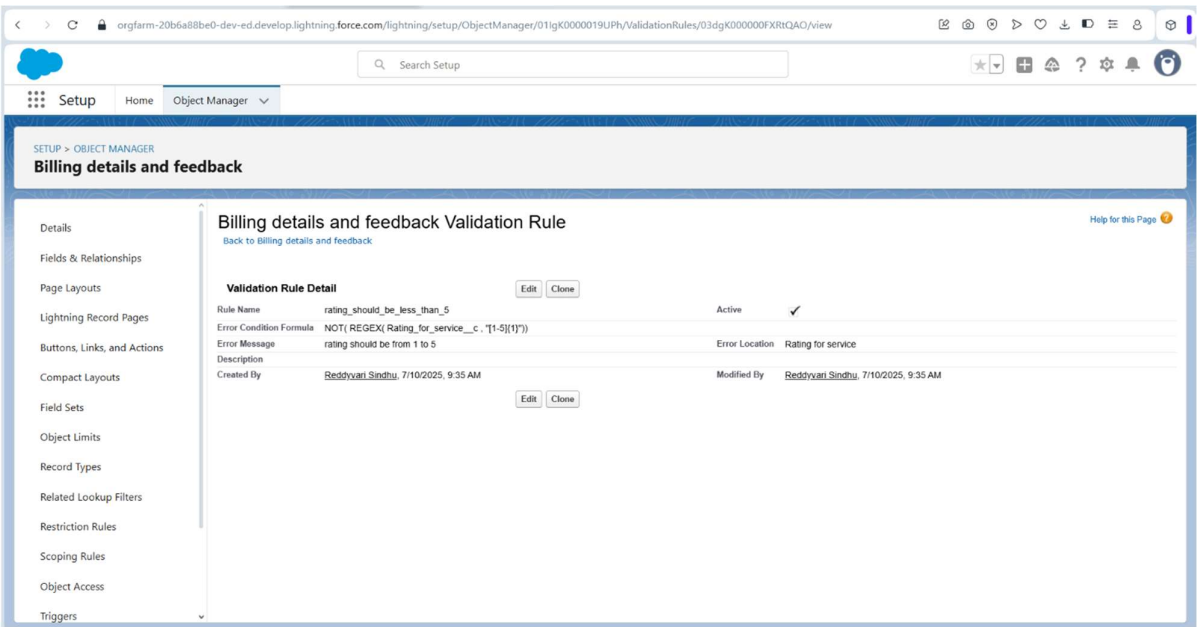
Field	Value
Rule Name	Vehicle
Error Condition Formula	NOT(REGEX(Vehicle_number_plate__c ,"[A-Z]{2}[0-9]{2}[A-Z]{2}[0-9]{4}"))
Error Message	Please enter valid number
Error Location	Vehicle number plate
Description	
Created By	Reddyxari.Sindhu, 7/10/2025, 9:32 AM
Modified By	Reddyxari.Sindhu, 7/10/2025, 9:32 AM

The 'Active' checkbox is checked, indicating the rule is active. There are 'Edit' and 'Clone' buttons for the rule details.

The rule prevents invalid or inconsistent vehicle numbers from being saved, which helps with accurate tracking and reporting.

2. Service Rating Validation (`Billing_Feedback__c`)

This rule ensures that the Feedback Rating entered by the customer is a number between 1 and 5, using the REGEX pattern `[1-5]{1}`.



If an invalid value is entered, the system throws an error:

“Rating should be from 1 to 5.”

This enforces consistency in feedback data, allowing the business to measure service satisfaction accurately in reports and dashboards.

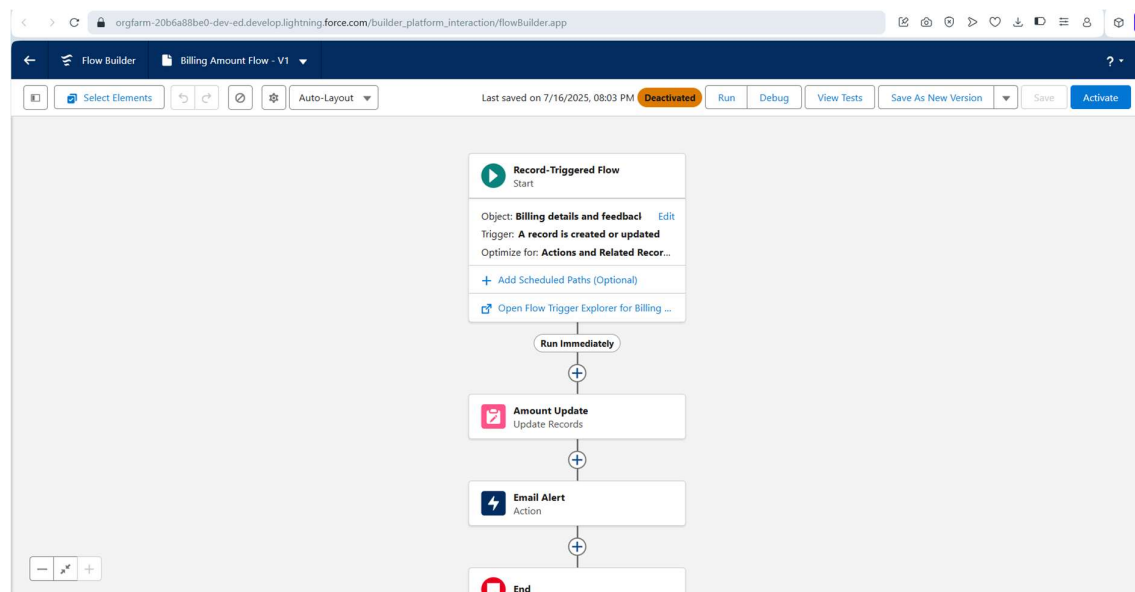
Both rules were implemented at the field level and prevent records from being saved if invalid data is entered. They contribute to data standardization, improve report accuracy, and ensure reliable input for automation and decision-making processes.

Automation Flows

Two record-triggered flows were developed to automate key business operations and reduce manual effort in the Garage Management CRM:

1. Billing Amount Flow (`Billing_Feedback__c`)

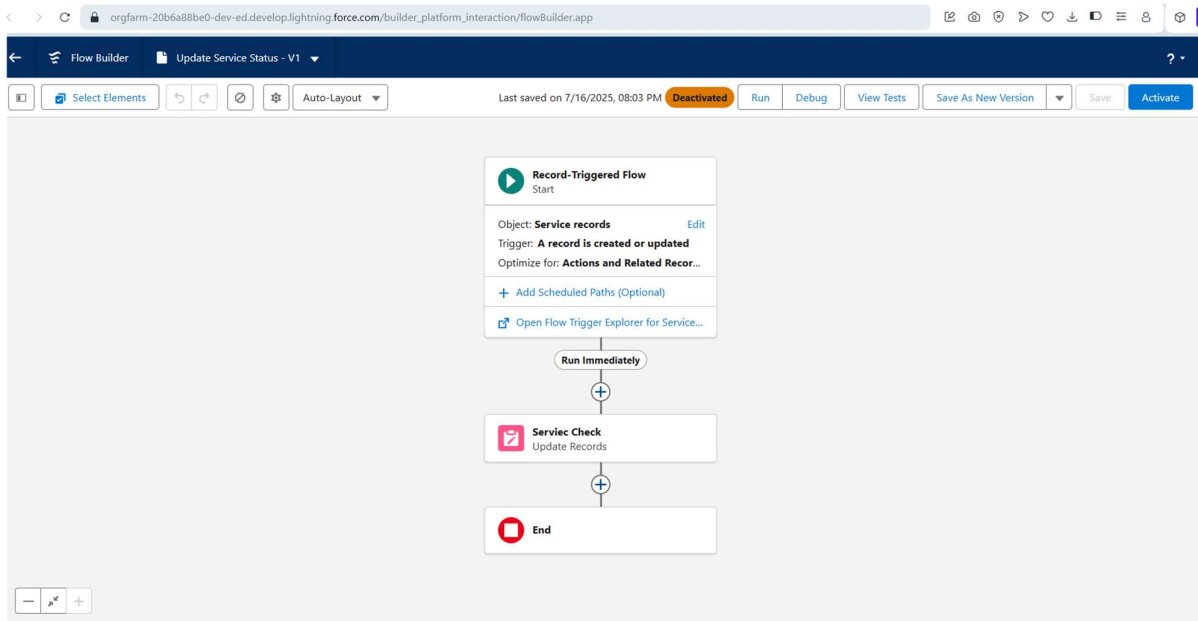
This flow is triggered when a billing record is created or updated. If the `Payment_Status__c` field equals “Completed”, it updates the `Payment_Paid__c` field with the actual service amount (fetched from related Appointment via Service Record). It also sends a thank-you email to the customer using a Text Template and the customer's email stored in the record. This enhances customer experience and provides automated post-service communication.



2. Update Service Status Flow (`Service_Records__c`)

This flow is triggered when a service record is created or updated. If the `Quality_Check_Status__c` field is marked True, the flow updates the `Service_Status__c` field to Completed. This ensures that only services which have passed quality review are marked as finished, supporting better operational accuracy.

Both flows are optimized for Actions and Related Records, making them efficient and responsive. These automations improve data consistency, save time, and support customer satisfaction and service integrity.



Apex Classes

Apex is Salesforce's programming language used to write custom backend logic.

In this project, an Apex class named 'AmountDistributionHandler' was created.

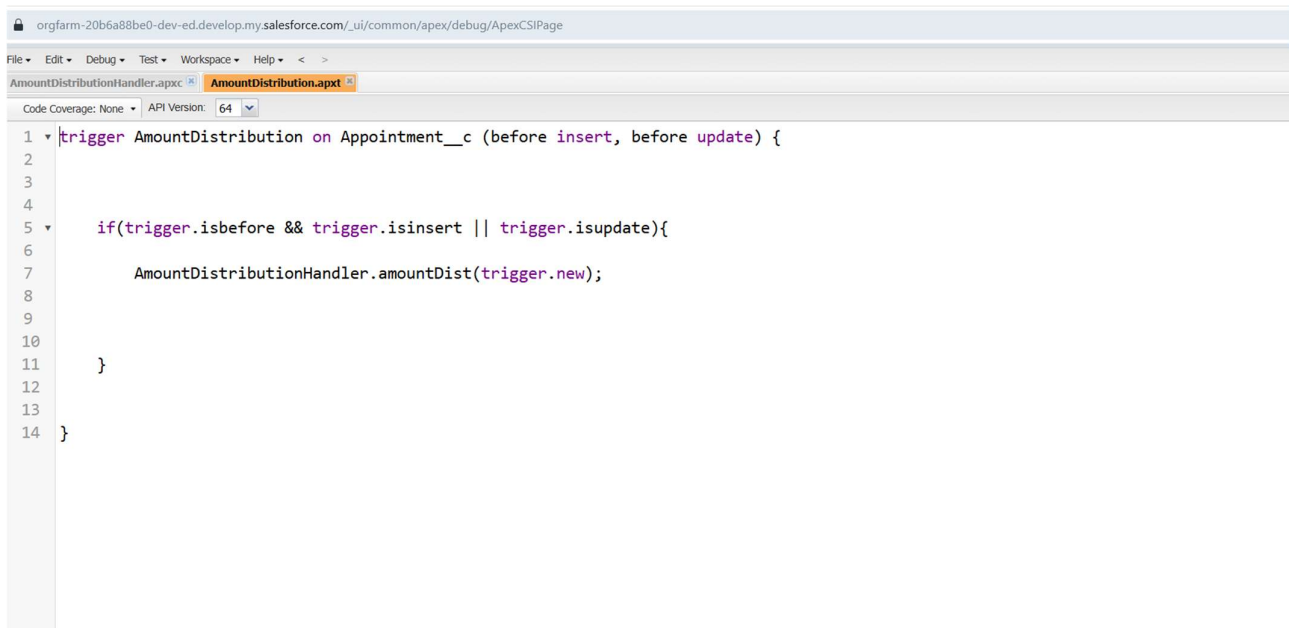
```
1 public class AmountDistributionHandler {
2
3     public static void amountDist(list<Appointment__c> listApp){
4
5         list<Service_records__c> serList = new list<Service_records__c>();
6
7         for(Appointment__c app : listApp){
8
9             if(app.Maintenance_service__c == true && app.Repairs__c == true && app.Replacement_Parts__c == true){
10
11                 app.Service_Amount__c = 10000;
12             }
13
14             else if(app.Maintenance_service__c == true && app.Repairs__c == true){
15
16                 app.Service_Amount__c = 5000;
17             }
18
19             else if(app.Maintenance_service__c == true && app.Replacement_Parts__c == true){
20
21                 app.Service_Amount__c = 8000;
22             }
23
24         }
25     }
26 }
```

The screenshot shows the Apex IDE with the code for the 'AmountDistributionHandler' class. The code is a static method 'amountDist' that takes a list of 'Appointment__c' objects as input. It iterates through each appointment and updates the 'Service_Amount__c' field based on the presence of 'Maintenance_service__c', 'Repairs__c', and 'Replacement_Parts__c' fields. The logic is as follows: if all three fields are true, the service amount is set to 10000; if 'Maintenance_service__c' and 'Repairs__c' are true, the service amount is set to 5000; if 'Maintenance_service__c' and 'Replacement_Parts__c' are true, the service amount is set to 8000. The IDE interface includes a toolbar with 'File', 'Edit', 'Debug', 'Test', 'Workspace', and 'Help' menus, and a status bar at the bottom showing 'Logs', 'Tests', 'Checkpoints', 'Query Editor', 'View State', 'Progress', 'Problems', and 'User'.

It calculates the Service Amount for each appointment based on selected services. The services include Maintenance, Repairs, and Replacement Parts. Different combinations of these services result in different billing amounts. The class has a method `amountDist()` that receives a list of `Appointment__c` records. This logic is called using a before insert/update trigger on the Appointment object. The automation ensures accurate, consistent billing without manual data entry. It helps enforce business rules and simplifies future maintenance. Apex enables advanced automation that goes beyond what Flows or Process Builder can do.

Apex Handler and Trigger

An Apex class named `AmountDistributionHandler` was developed to automatically calculate and assign service charges based on the combination of services selected in an appointment — including Maintenance Service, Repairs, and Replacement Parts. The class contains a method `amountDist()` which uses conditional logic to set a fixed amount in the `Service_Amount__c` field depending on which service checkboxes are marked as `true`.

The image is a screenshot of the Salesforce IDE's Apex editor. The top bar shows the URL 'orgfarm-20b6a88be0-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage'. Below the menu bar (File, Edit, Debug, Test, Workspace, Help), there are tabs for 'AmountDistributionHandler.apxc' and 'AmountDistribution.apxt'. The 'AmountDistribution.apxt' tab is active. The code editor shows the following Apex trigger code:

```
1 trigger AmountDistribution on Appointment__c (before insert, before update) {  
2  
3  
4  
5     if(trigger.isbefore && trigger.isinsert || trigger.isupdate){  
6  
7         AmountDistributionHandler.amountDist(trigger.new);  
8  
9  
10  
11     }  
12  
13  
14 }
```

A corresponding Apex Trigger `AmountDistribution` was created on the `Appointment__c` object and set to fire before insert and before update. This trigger ensures

that whenever a new appointment is created or an existing one is modified, the billing logic is executed automatically via the handler.

This approach eliminates manual data entry for service pricing, ensures consistency in billing, and enforces standard business rules. It also enhances automation by integrating logic directly at the object level, making the system smarter and more reliable during real-time operations.

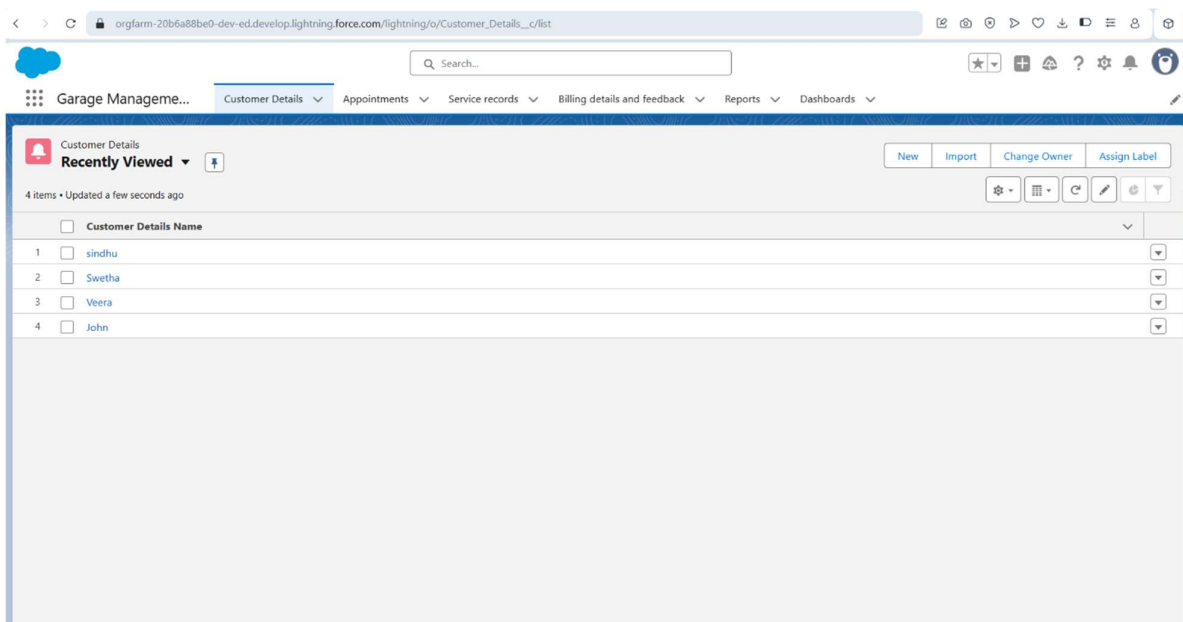
Phase 3: UI/UX Development & Customization

This phase focused on building a user-friendly, visually consistent, and functional user interface (UI) for the Garage Management CRM using Salesforce Lightning tools. The goal was to ensure that both garage staff and managers could easily navigate, interact with, and manage records in the system.

1. Lightning App Setup – Garage Management App

A custom Lightning App named Garage Management was created using the App Manager in Salesforce. This app served as the central entry point for users working on customer details, appointments, services, and billing. Key configurations included:

App visibility restricted to specific profiles (Manager and Sales Person).



Four custom object tabs were created:

- * Customer Details
- * Appointments
- * Service Records
- * Billing Details and Feedback
- * Each tab was added via the Custom Tabs section and linked to the Garage Management app.

The Garage Management app simplifies navigation by grouping all related modules into a single interface.

2. User Management – Profiles and Access

Manager Profile

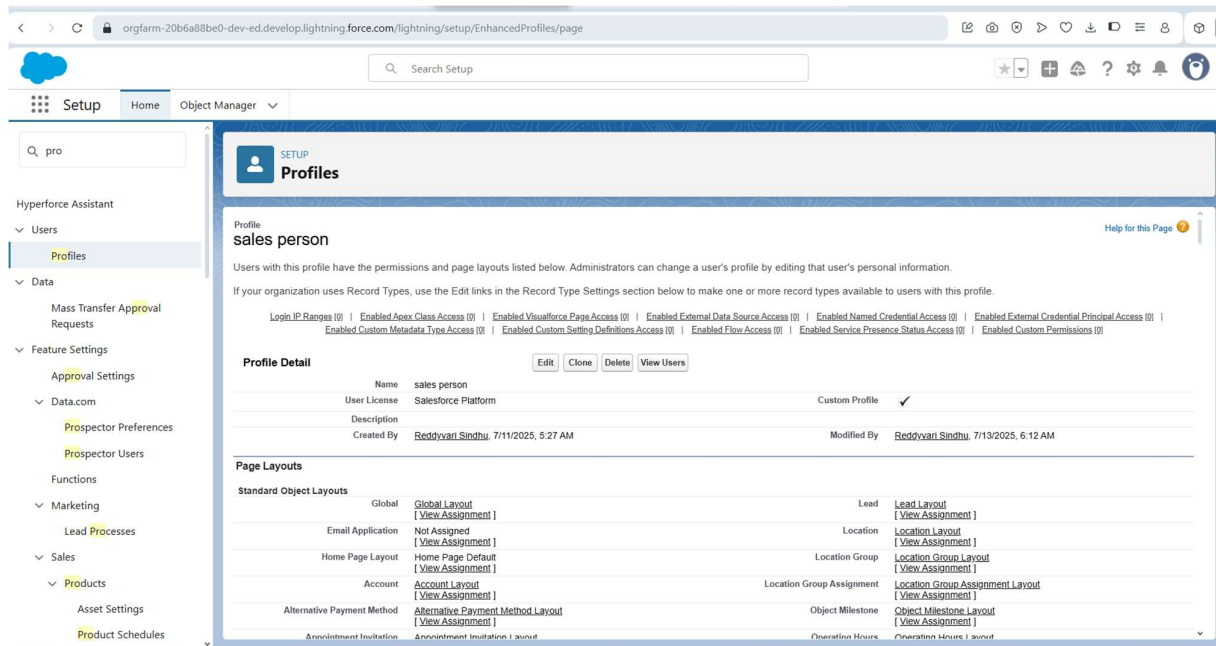
- * Cloned from Standard User.
- * Assigned as the default profile for senior users.
- * Granted full Create, Read, Edit, Delete (CRED) access to all custom objects.
- * Session timeout set to 8 hours, and password policy set to "Never Expires".
- * Garage Management was set as the default app upon login.

The screenshot shows the Salesforce Setup interface for the 'Manager' profile. The left sidebar contains navigation links for Setup, Home, and Object Manager. The main content area displays the 'Manager' profile details, including a list of permissions (Login IP Ranges, Apex Class Access, Visualforce Page Access, External Data Source Access, Named Credential Access, External Credential Principal Access, Custom Metadata Type Access, Custom Setting Definitions Access, Flow Access, Service Presence Status Access, and Custom Permissions) and a table of page layouts. The page layouts table lists various standard object layouts and their corresponding views, such as Global Layout, Location Group Assignment Layout, Email Application, Macro Layout, Home Page Default, Object Milestone Layout, Account Layout, Operating Hours Layout, Alternative Payment Method Layout, Opportunity Layout, and Appointment Invitation Layout.

Standard Object Layouts	Global	Location Group Assignment
Global Layout	Global Layout [View Assignment]	
Email Application	Not Assigned [View Assignment]	
Home Page Default	Home Page Default [View Assignment]	
Account Layout	Account Layout [View Assignment]	
Alternative Payment Method	Alternative Payment Method Layout [View Assignment]	
Appointment Invitation	Appointment Invitation Layout	
Location Group Assignment	Location Group Assignment Layout [View Assignment]	
Macro	Macro Layout [View Assignment]	
Object Milestone	Object Milestone Layout [View Assignment]	
Operating Hours	Operating Hours Layout [View Assignment]	
Opportunity	Opportunity Layout [View Assignment]	
Opportunity Product	Opportunity Product Layout	

Sales Person Profile

- * Cloned from Salesforce Platform User.
- * Given limited object permissions (typically Read, Create, Edit).
- * Excluded from sensitive fields like internal comments or full billing configuration.
- * Garage Management app set as default for easy access.



These user profiles ensured role-based access control, enhancing security and usability across the team.

3. Page Layouts & Dynamic Forms

Each custom object was configured with a tailored page layout to ensure users see only relevant fields:

Fields like Billing Amount and Service Feedback were placed logically and grouped using sections.

- * For the Appointments and Service Records objects:
- * Dynamic Forms were used to show/hide fields based on field values.
- * Example: The “Feedback” section only appears after a service is marked as completed.

* Related records like Service Records within an Appointment were displayed using Related Lists.

4.Lightning Record Pages

Using the Lightning App Builder, custom record pages were built for each object:

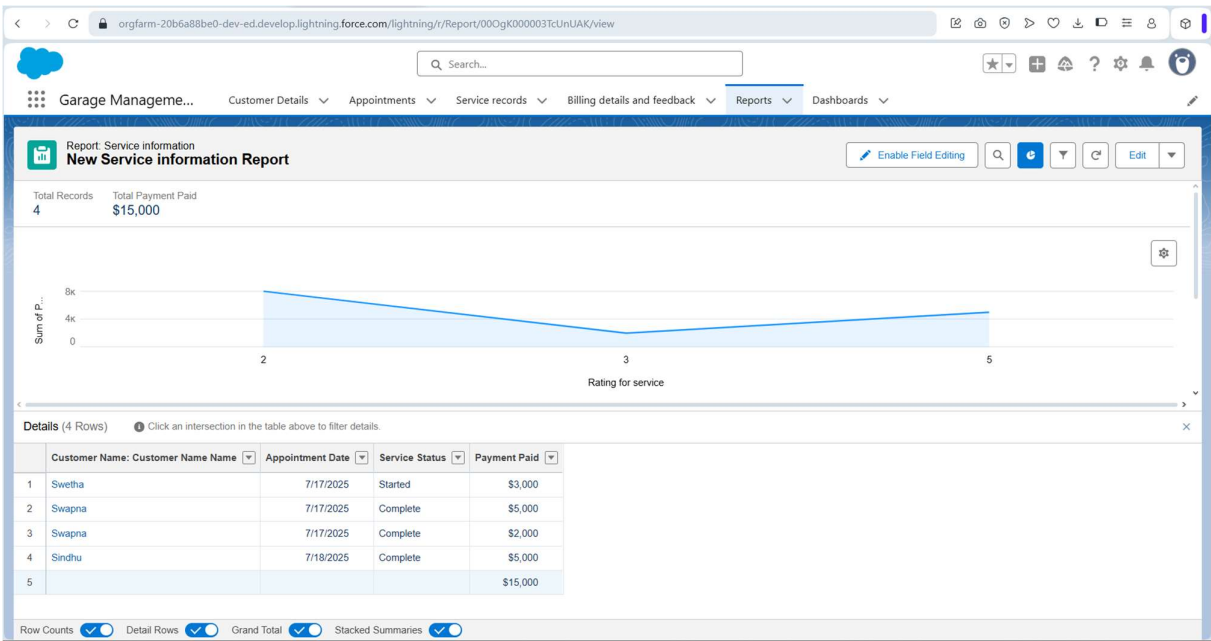
- * Highlights Panels: Displayed key fields like Customer Name, Appointment Date, Service Status at the top.
- * Tabbed Layouts: Helped divide form content into tabs like "Service Info", "Billing Details", "Feedback".
- * Related Lists & Related Record Components: Displayed associated records (e.g., all service records under an appointment).
- * Mobile responsiveness was also ensured for use on Salesforce mobile app.

These custom pages offered a cleaner and more intuitive user experience, reducing training time.

5.Reports

A custom report was created using Report Builder with the following configuration:

- * Report Type: “Service Information” under Other Reports



Selected Fields:

- * Customer Name
- * Appointment Date
- * Service Status
- * Payment Paid

Group Rows:

- * Rating for Service
- * Payment Status

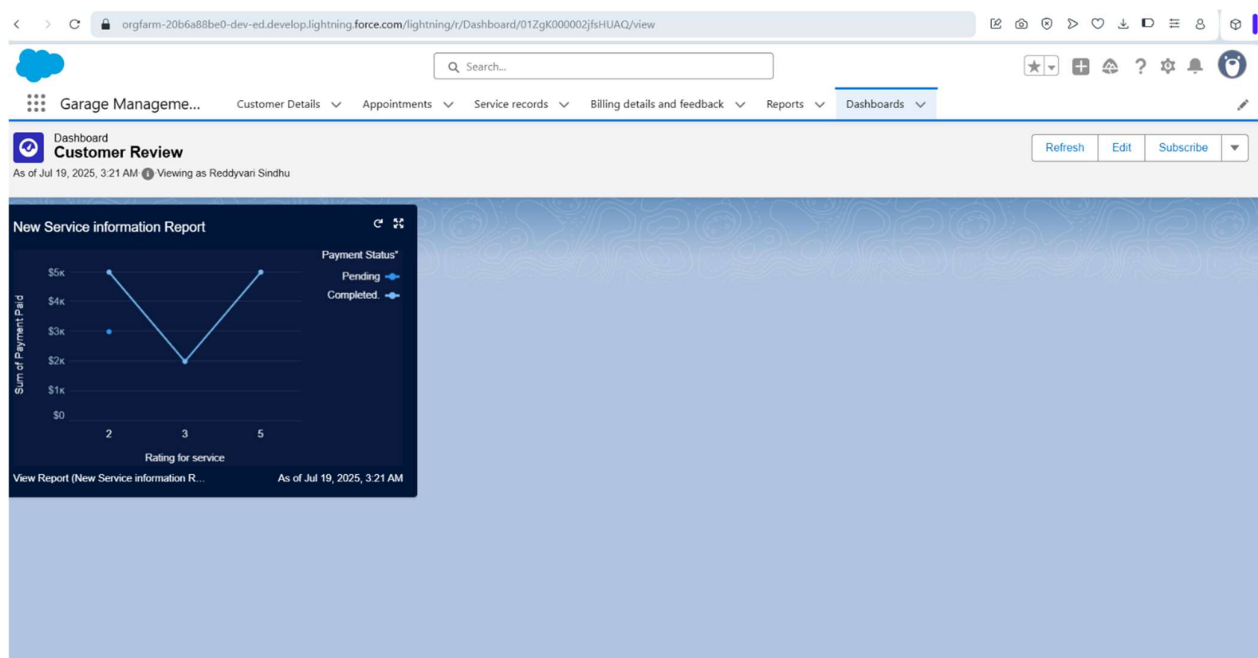
Chart: A Line Chart was added to visualize how payment and service ratings correlate over time.

Purpose: Help managers monitor customer satisfaction and identify bottlenecks or issues in billing or service quality.

6.Dashboards

A dashboard was created using the above report with the following setup:

- * Component Type: Line Chart



- * Theme: Customized for visibility
- * Title: “Garage Service Report”
- * Subscription:
- * Scheduled Weekly
- * Sent every Monday
- * Helps management track weekly performance and customer trends

This dashboard serves as a decision-making tool for business stakeholders to evaluate service delivery and identify improvement areas.

Lightning Pages

Lightning Pages are custom record page layouts that define what users see when they open a record (e.g., an appointment or a service record). You customized these pages using the Lightning App Builder to make them user-friendly and role-specific.

1. Display Key Fields Prominently

For example, the `Appointment__c` Lightning Page displays fields like Appointment Date, Vehicle Number, and Status in the ****highlights panel**** for quick reference.

2. Organize Details with Tabs

Pages like `Service_Records__c` are organized into tabs (e.g., Details, Related, Notes) so that users can easily access relevant sections without scrolling too much.

3. Include Related Lists and Lookups

You used Lightning Pages to show related records like services under appointments or billing linked to a service record using related list components.

4. Support Role-Specific UX

The layout helps Sales Persons or Managers view only what’s necessary for their job (e.g., Managers see more reporting-related fields, Sales Persons see only service and appointment data).

5. Integrate with Automation

Flows or quick actions (e.g., “Send Payment Confirmation”) can be added to Lightning Pages, enabling users to trigger automated tasks directly from the page.

6.Improve Overall Navigation

By using Lightning Pages for each object, you created a clean and modern UI for navigating garage operations.

Phase 4: Data Migration, Testing & Security

This phase focuses on ensuring secure data handling, accurate role-based access, and functional testing of all major CRM features. While the project was developed on a Salesforce Developer Org (sandbox-like environment), key production-like configurations were implemented to mirror real-world behavior.

1.Data Migration

No bulk data import was performed using Data Import Wizard or Data Loader in this project. Instead, “manual data entry” was used to create realistic sample records across all custom objects:

* `Customer_Details__c`

* `Appointment__c`

* `Service_Records__c`

* `Billing_Feedback__c`

Creating records manually allowed for better testing of validations, flows, triggers, and relationship mappings in a controlled way.

2.Matching Rules and Duplicate Rule

To maintain clean and non-redundant customer data, both a Matching Rule and a Duplicate Rule were created:

Matching Rule

- * Created on the `Customer_Details__c` object
- * Checks for duplicate entries based on “Gmail” and “Phone Number” fields
- * Matching method used: “Exact Match”
- * Ensures two customers with the same contact information are not accidentally added.

Duplicate Rule

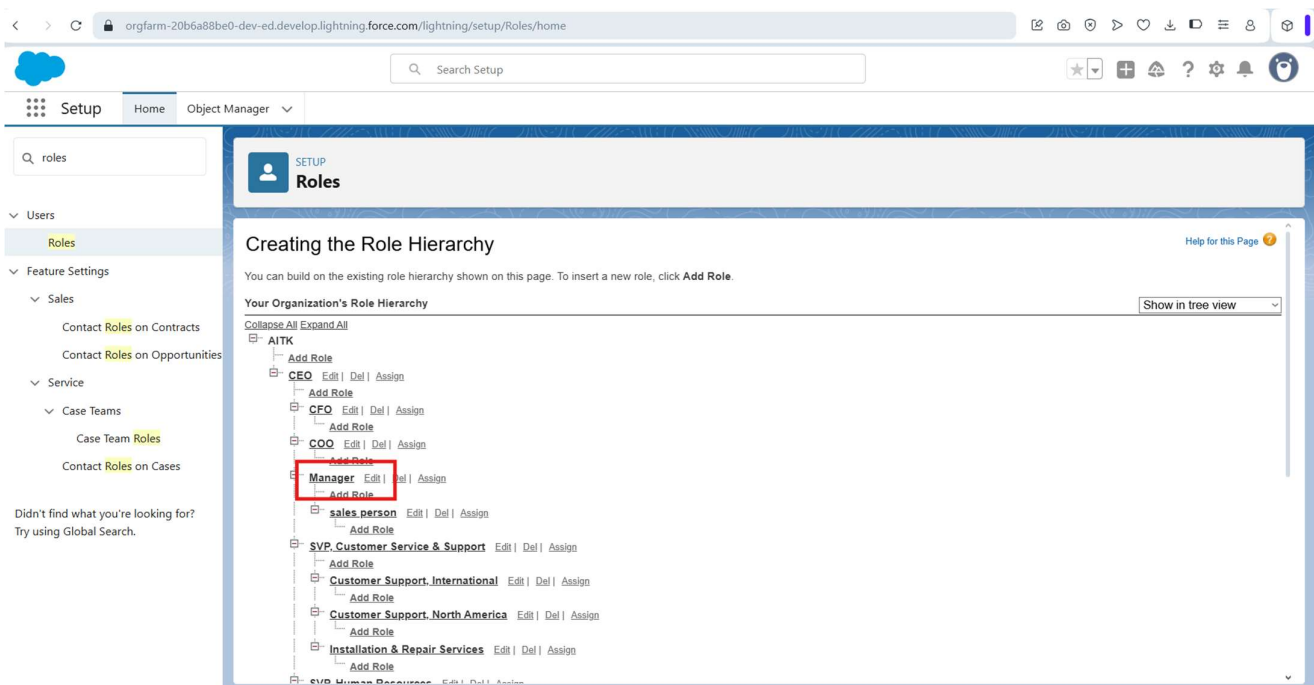
- * Uses the above matching rule
- * Prevents saving duplicate customer records
- * Error message prompts users to review the existing data

Both rules were activated to enforce real-time data quality during record creation or updates.

3.Roles and Role Hierarchy

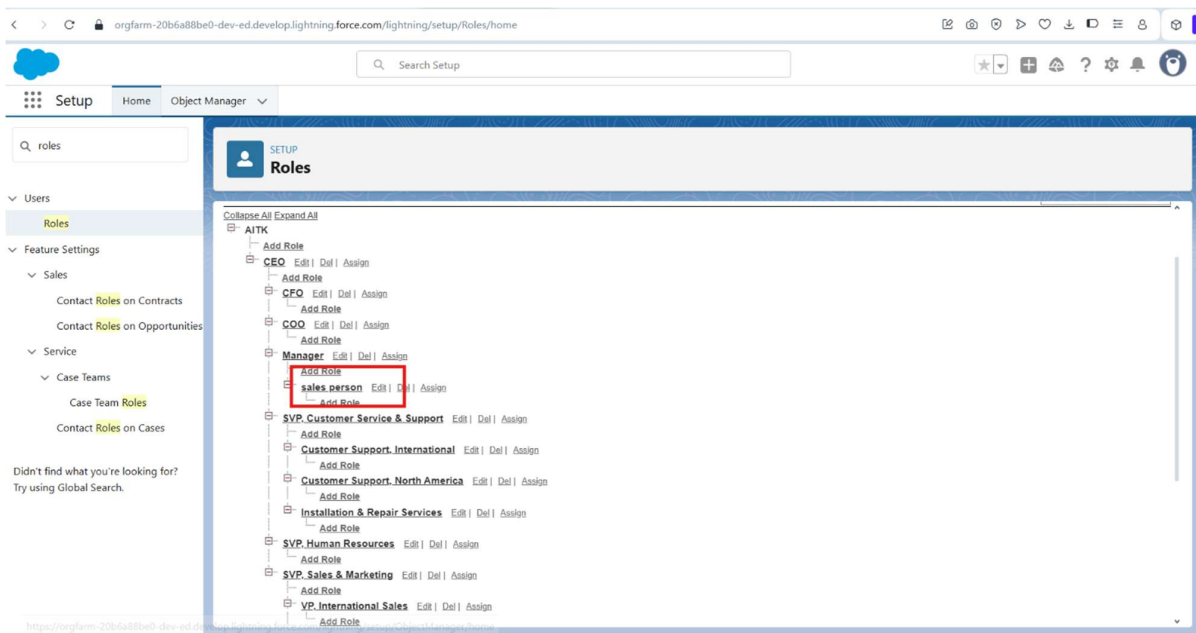
A structured role hierarchy was configured to simulate team-based access control. The roles created were:

1. Manager (Top-level)



The screenshot shows the Salesforce Setup interface for the 'Roles' section. The left sidebar contains a navigation menu with 'Setup' selected, and a search bar for 'roles'. The main content area is titled 'Creating the Role Hierarchy' and displays 'Your Organization's Role Hierarchy' as a tree view. The hierarchy starts with 'AITK' at the top, followed by 'CEO', 'CFO', 'COO', and 'Manager'. The 'Manager' role is highlighted with a red box. Below 'Manager' are roles like 'sales person', 'SVP, Customer Service & Support', 'Customer Support, International', 'Customer Support, North America', 'Installation & Repair Services', and 'SVP Human Resources'. Each role in the hierarchy has an 'Add Role' link next to it. The right sidebar includes a 'Show in tree view' button and a 'Help for this Page' link.

2. Sales Person 1 (Reports to Manager)



This hierarchy enables record access inheritance, where a manager can view and control the records created by the salespeople under them. It also supports future sharing logic or approvals that depend on role levels.

4. Sharing Settings

To secure service-related data and allow selective visibility:

The OWD (Organization-Wide Default) for the `Service_Records__c` object was set to Private, meaning records are only visible to the record owner and users above them in the role hierarchy.

A Sharing Rule (Sharing setting) was created:

- * Source: Records owned by users in the Sales Person role
- * Target: Shared with the Manager role
- * Access Level: Read/Write

- * Purpose: Managers can supervise and manage service records without changing OWD to public

This configuration enhances record-level security while maintaining operational transparency for supervisors.

5.Profiles & Permission Management

Two custom profiles were created by cloning standard Salesforce profiles:

Manager Profile

- * Cloned from: Standard User
- * Full access to all custom objects
- * Assigned to Manager users
- * Default app set to: Garage Management
- * Custom password policy: never expires, 8-character minimum

Sales Person Profile

- * Cloned from: Salesforce Platform User
- * Assigned to Sales team users
- * Object permissions limited to create, read, and edit (no delete)
- * Sharing and flows used to manage extended access

These profiles ensure role-specific visibility, enforce field-level security, and restrict access to only required components.

6.Apex Trigger and Handler Execution

To automate billing logic:

- * An Apex Class named `AmountDistributionHandler` was created.
- * A Trigger named `AmountDistribution` was set on the `Appointment__c` object.
- * The trigger runs before insert and before update.

* It invokes logic from the handler to assign billing amounts automatically based on service checkboxes like:

- * Maintenance
- * Repairs
- * Replacement Parts

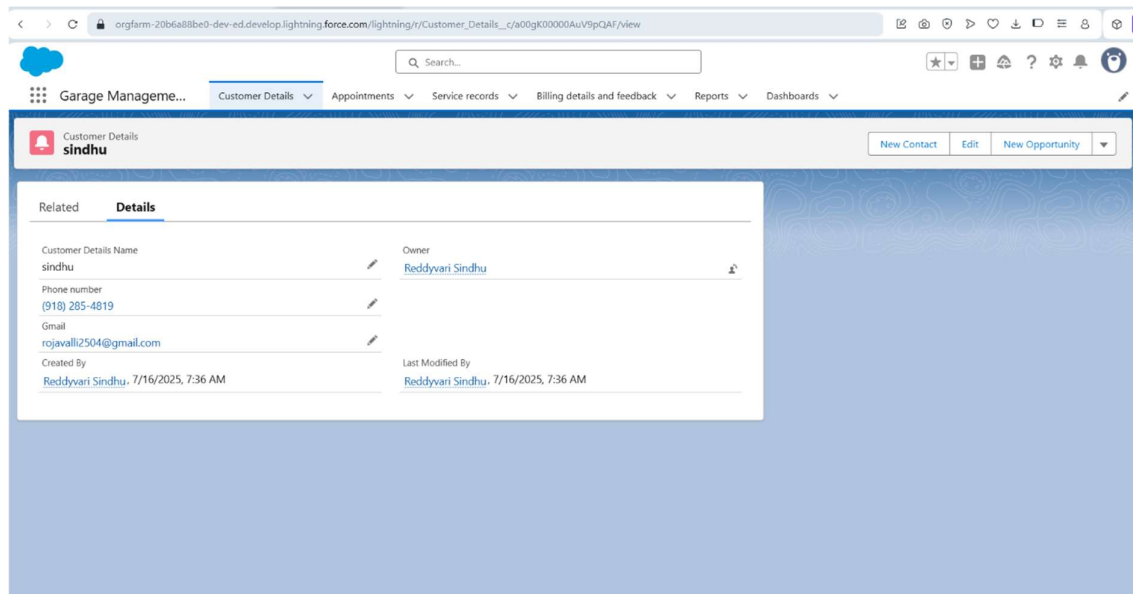
This ensures service charges are assigned consistently and without manual intervention.

7. Testing and Feature Validation

To verify the functionality of all features and ensure business rules were correctly enforced, manual testing was performed on every component:

Tested Features:

- * Appointment creation and modification



- * Approval Process simulation
- * Flow-triggered updates (e.g., Payment confirmation, Service status updates)
- * Trigger-based billing automation

Garage Manage... Customer Details Appointments Service records Billing details and feedback Reports Dashboards

Appointment **app-001** New Contact Edit New Opportunity

Related Details

Appointment Name	app-001	Owner	Reddyvari Sindhu
Customer Details	Swetha		
Appointment Date	7/17/2025		
Maintenance Service	<input type="checkbox"/>		
Repairs	<input type="checkbox"/>		
Replacement Parts	<input checked="" type="checkbox"/>		
Service Amount	\$5,000		
Vehicle number plate	TS09AB4567		
Customer Name	Swetha		
Created By	Reddyvari Sindhu, 7/14/2025, 7:49 AM	Last Modified By	Reddyvari Sindhu, 7/14/2025, 7:54 AM

orgfarm-20b6a88be0-dev-ed.develop.lightning.force.com/lightning/r/Service_records__c/a02gk000002si7IQAA/view

Garage Manage... Customer Details Appointments Service records Billing details and feedback Reports Dashboards

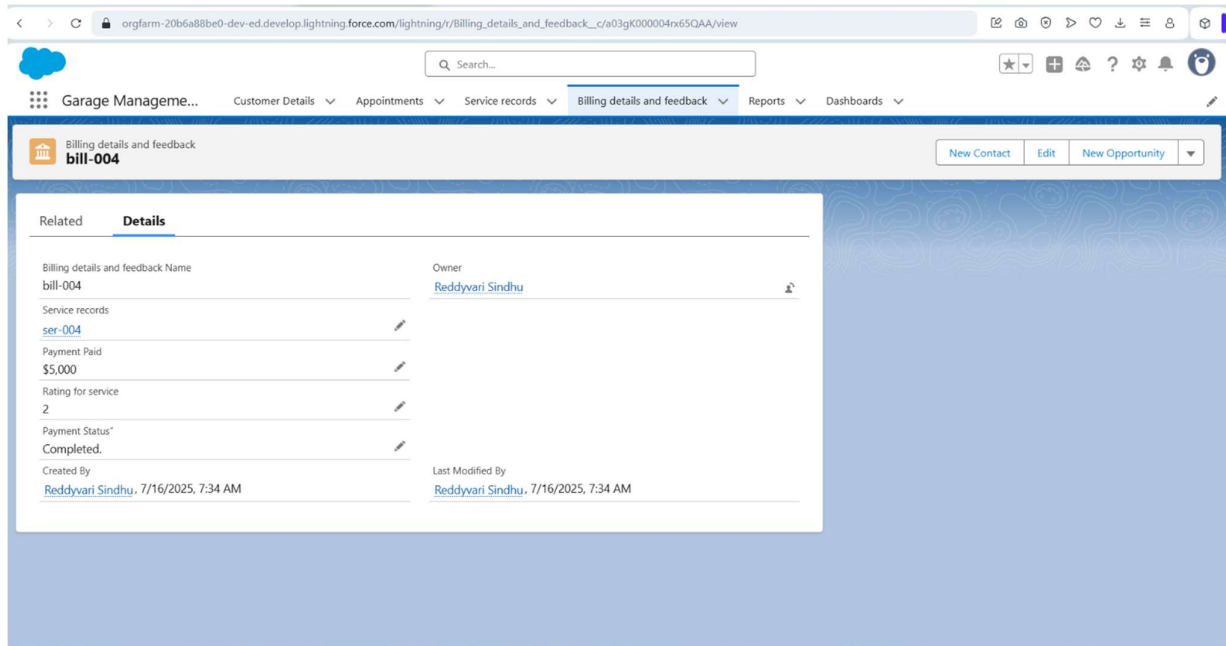
Service records **ser-005** New Contact Edit New Opportunity

Related Details

Service records Name	ser-005	Owner	Reddyvari Sindhu
Appointment	app-001		
Quality Check Status	<input checked="" type="checkbox"/>		
service date	7/14/2025		
Service Status	Started		
Created By	Reddyvari Sindhu, 7/14/2025, 10:09 AM	Last Modified By	Reddyvari Sindhu, 7/14/2025, 10:09 AM

* Validation rules (e.g., Vehicle number plate format, Rating between 1–5)

* Reports & Dashboard data visibility



Phase 5: Deployment, Documentation & Maintenance

1. Deployment Strategy

Since the entire project was developed and tested within a Salesforce Developer Org, no formal deployment to production was required. However, in real-world scenarios, Change Sets or Unmanaged Packages would be used to deploy components from a Sandbox to Production or another org. This ensures that metadata such as objects, fields, flows, Apex classes, and validation rules can be securely and consistently transferred.

2. System Maintenance and Monitoring

The Garage Management CRM system is designed to be scalable and easy to maintain. Regular maintenance includes reviewing automation flows, checking error logs, validating user access, and updating picklist values or form layouts based on operational changes. Admins can monitor flow executions and Apex errors using Debug Logs, Email Alerts, and the Flow Error Email Notifications provided by Salesforce.

3. Troubleshooting Approach

A structured troubleshooting method is in place to resolve issues quickly. Debug logs are used to trace errors in Flows or Apex triggers. Validation failures and automation misfires are diagnosed by reviewing the error messages shown during record creation or updates. In addition, a Log__c custom object can be optionally created in future to store error details and improve system transparency. Documentation of all configuration steps and logic helps with quick issue identification and resolution during handover or enhancement.

Additional Points

1. Custom Lightning App:

Created a Lightning app named Garage Management that groups all custom objects and tabs for easy access.

2. Custom Tabs for Navigation:

Tabs were created for all major custom objects — Customer Details, Appointments, Service Records, Billing & Feedback — improving user navigation.

3. Custom Profiles:

Two profiles were implemented — Manager and Sales Person — each with specific object-level and field-level permissions.

4. Role Hierarchy Setup:

Created a role structure where Manager is at the top, and two Sales Person roles report under it to control data visibility.

5. Validation Rules:

Implemented validation on:

Vehicle Number Format

Feedback Rating

6. Flows for Automation:

Two record-triggered flows:

Automatically update payment and send thank-you emails.

Automatically update service status after quality check.

7. Apex Trigger with Handler Class:

Apex trigger on Appointment__c calculates service charges dynamically using a handler class based on selected service types.

8. Matching & Duplicate Rules:

Configured Matching and Duplicate Rules on Customer_Details__c to prevent duplicate records based on Gmail and Phone Number.

9. Sharing Rules & Private OWD:

Service_Records__c object set to Private, with sharing rules allowing Managers to see records owned by Sales Persons.

10. Custom Report & Dashboard:

Built a report grouped by Rating and Payment Status, and created a line chart dashboard subscribed weekly by managers.

Conclusion

The Garage Management CRM project successfully streamlined vehicle service operations using the Salesforce platform. Key modules like customer handling, appointment booking, service tracking, billing, and feedback were implemented with automation, validation, and role-based access. Custom objects, flows, triggers, and reports were configured to improve efficiency, accuracy, and user experience. The solution reduces manual work, enhances customer communication, and provides real-time visibility to managers. With scope for future enhancements like chatbot integration or AI-based service suggestions, the system is scalable and ready for real-world adoption.