GARAGE MANAGEMENT SYSTEM

B.Bhagya sree(22341A0512)

Project Overview:

The Garage Management System (GMS) is a customized Salesforce CRM solution designed to automate and streamline essential operations in an automobile repair shop. This CRM platform addresses several garage-related challenges such as managing customer information, tracking vehicle service history, handling appointment scheduling, managing tasks for mechanics, and overseeing spare parts inventory and billing activities.

The system provides an intuitive interface for both administrators and garage staff to manage operations efficiently while ensuring customers experience a smooth and transparent service process. With features like dynamic dashboards, validation rules, and automation, GMS ensures quick response times, improved service quality, and better decision-making capabilities. Ultimately, this project supports effective customer relationship management and empowers garages to stay competitive in today's fast-paced service environment.

Objective:

The primary objective of creating the Garage Management System (GMS) CRM is to digitize and automate routine tasks in an automotive workshop. This CRM improves customer relationship management by maintaining complete service histories, enabling smooth appointment scheduling, and sending early service reminders. It supports efficient garage operations through effective job allocation and real-time stock management.

By utilizing Salesforce's automation and reporting tools, the system lowers human errors, accelerates service turnaround times, and enhances transparency for both customers and employees. Ultimately, GMS boosts operational efficiency and improves customer satisfaction.

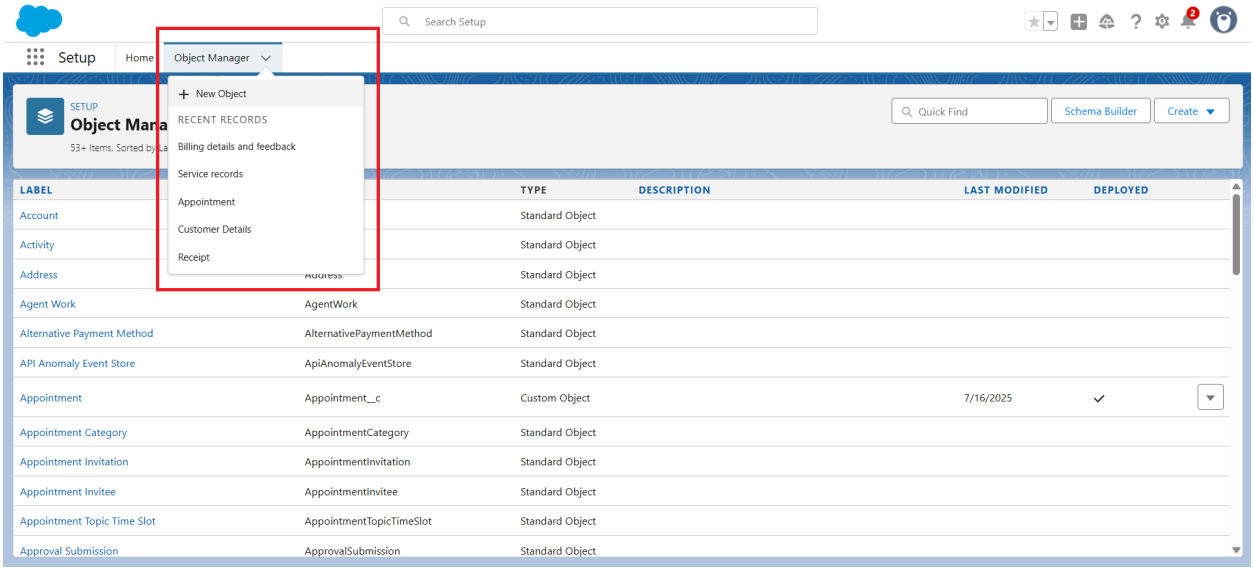
Phase 1: Requirement Analysis & Planning

Automotive garages frequently encounter difficulties in handling customer records, tracking service tasks, overseeing inventory, and ensuring billing accuracy, particularly when relying on manual or partially digital processes. The demand for a unified system that can manage all garage activities in one platform led to the creation of this Salesforce-based Garage Management System. Primary user requirements include easier service bookings, live job status tracking, automated billing, inventory notifications, and quick access to vehicle service histories. Additionally, the system helps with staff assignments and effective task monitoring.

Phase 2: Salesforce Development – Backend & Configurations

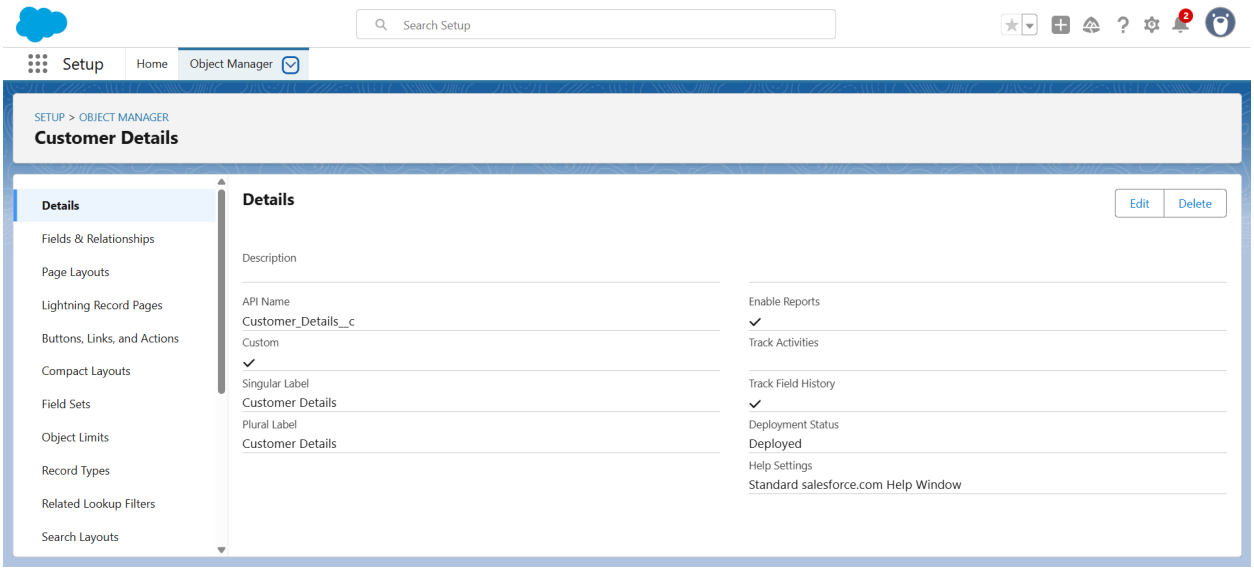
The Garage Management System was built and configured using the Salesforce Developer Edition. Custom objects, fields, and permissions were designed to handle essential functions like managing customer information and appointments. Field history tracking was implemented for key records to maintain data accuracy. Backend logic, including appointment updates and billing, was managed using automation tools such as Process Builder and Flows. Security measures were enforced through role hierarchies, sharing settings, and access control.

1.Objects:



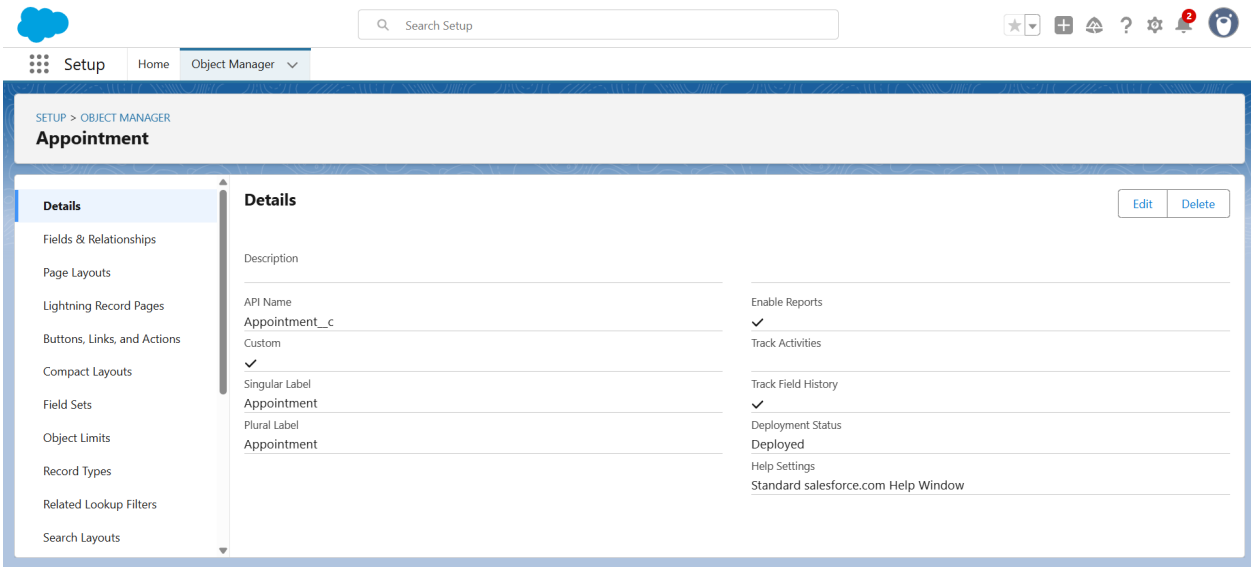
i. Customer Details:

The *Customer Details* object holds all personal and contact information of garage customers. It contains fields such as Customer Name, Phone Number, Email, and Vehicle Information. This object is designed to efficiently manage customer records, allowing quick access during appointments and billing processes. It also supports reporting and field history tracking to enhance data analysis and traceability.



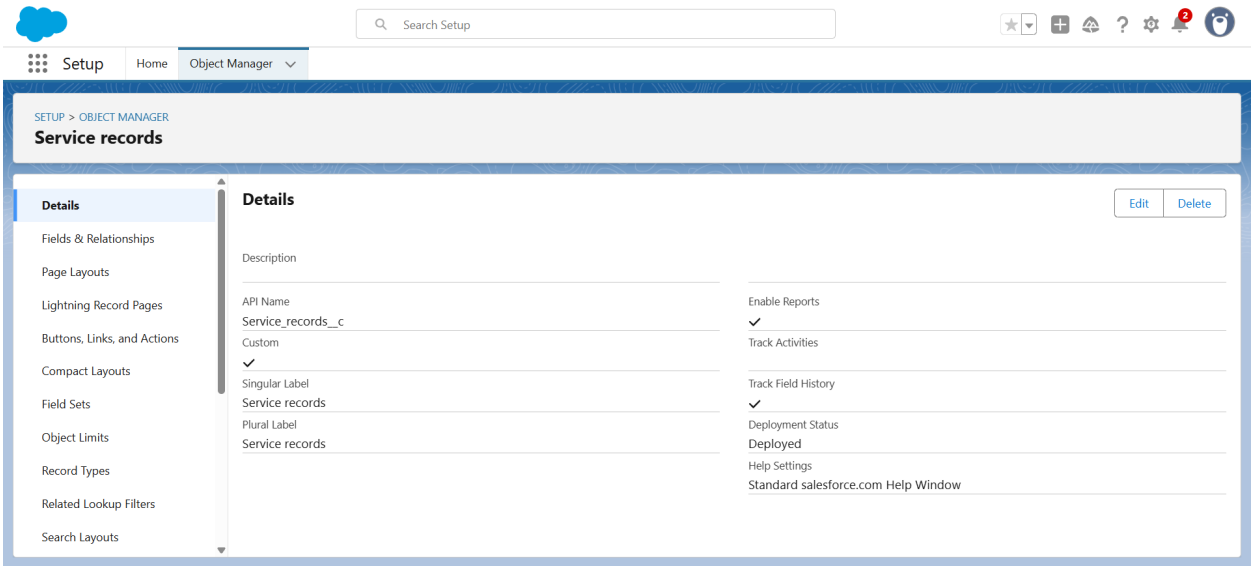
ii. Appointment

The *Appointment* object records all customer bookings related to vehicle servicing. It generates a unique Appointment Name automatically (app-{000}) and includes fields such as Date, Time, Service Type, and Assigned Mechanic. Automations are connected to this object to handle sending confirmations and reminders, making the service process more efficient.



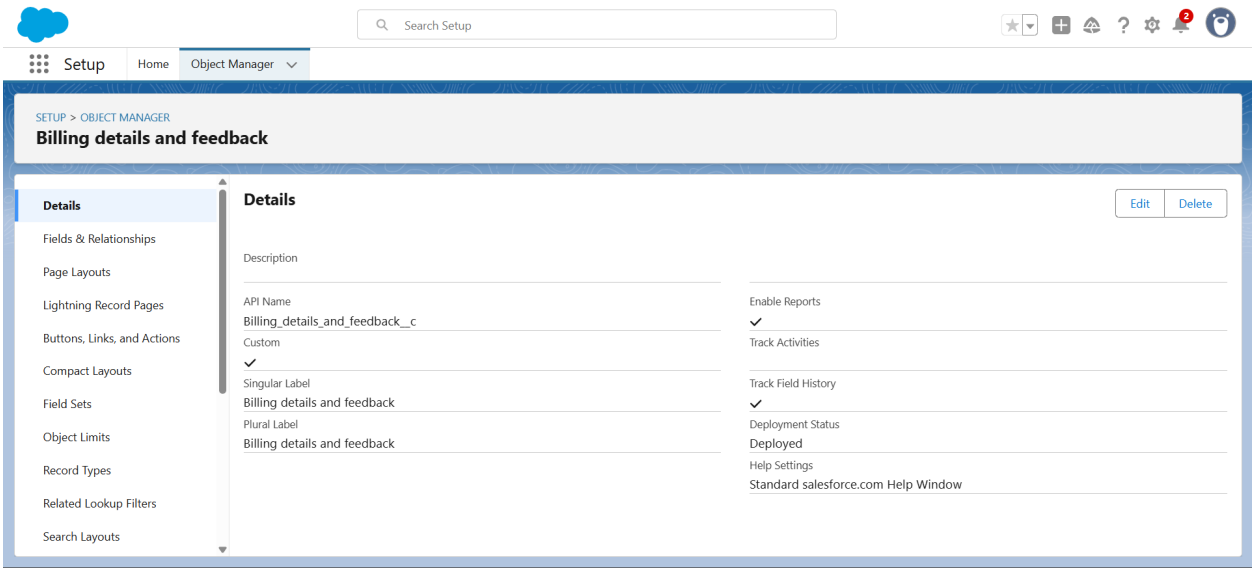
iii. Service Records

The *Service Records* object keeps a log of all services performed on each vehicle. It includes information like Service Date, Description, Parts Replaced, and Service Cost. Using an auto-generated numbering format (ser-{000}), it ensures that every record is uniquely identified. This object is vital for future references, warranty claims, and building customer trust.



iv. Billing Details and Feedback

This object manages billing information and gathers customer feedback after service completion. It includes fields such as Total Cost, Payment Mode, Service Satisfaction Rating, and Comments. Each record is auto-numbered in the format (bil-{000}). This object ensures organized billing and feedback tracking.

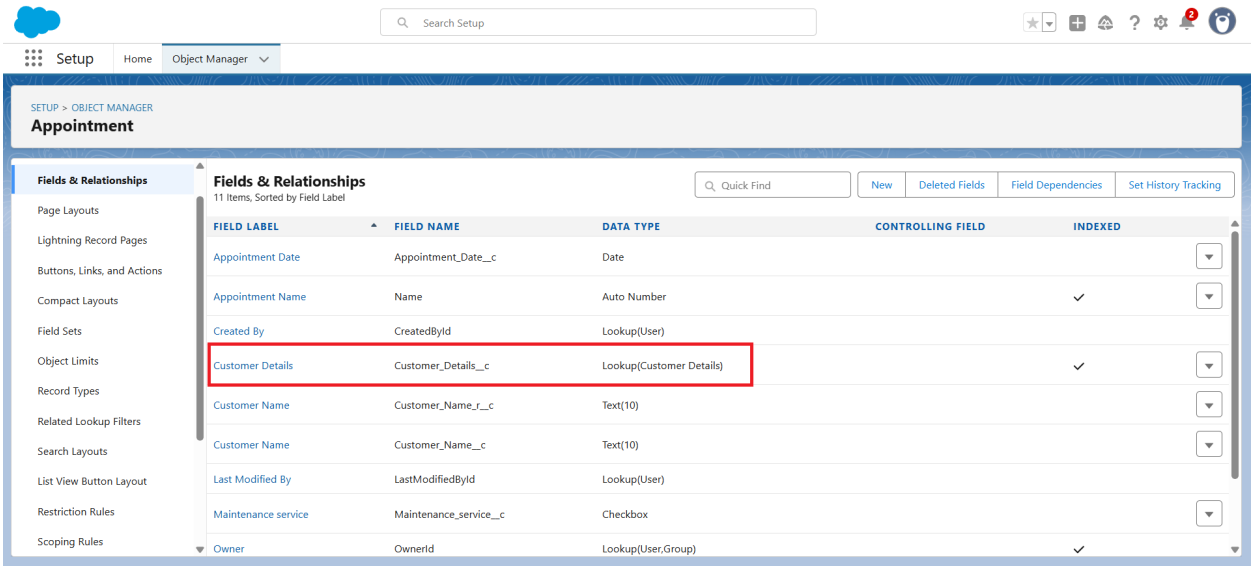


2. Fields

Fields hold essential information for each object, functioning like columns in a database. They simplify viewing, editing, searching, and managing records. There are two categories of fields: **Standard Fields** and **Custom Fields**. Standard Fields are predefined by Salesforce, such as *Created By*, *Owner*, and *Last Modified*, and cannot be removed easily. Custom Fields, on the other hand, are user-created and adaptable, allowing us to capture garage-specific details like *Vehicle Type*, *Appointment Date*, or *Service Cost*. In the Garage Management System, custom fields ensured we gathered only the data that was necessary.

i. Lookup Field on Appointment Object:

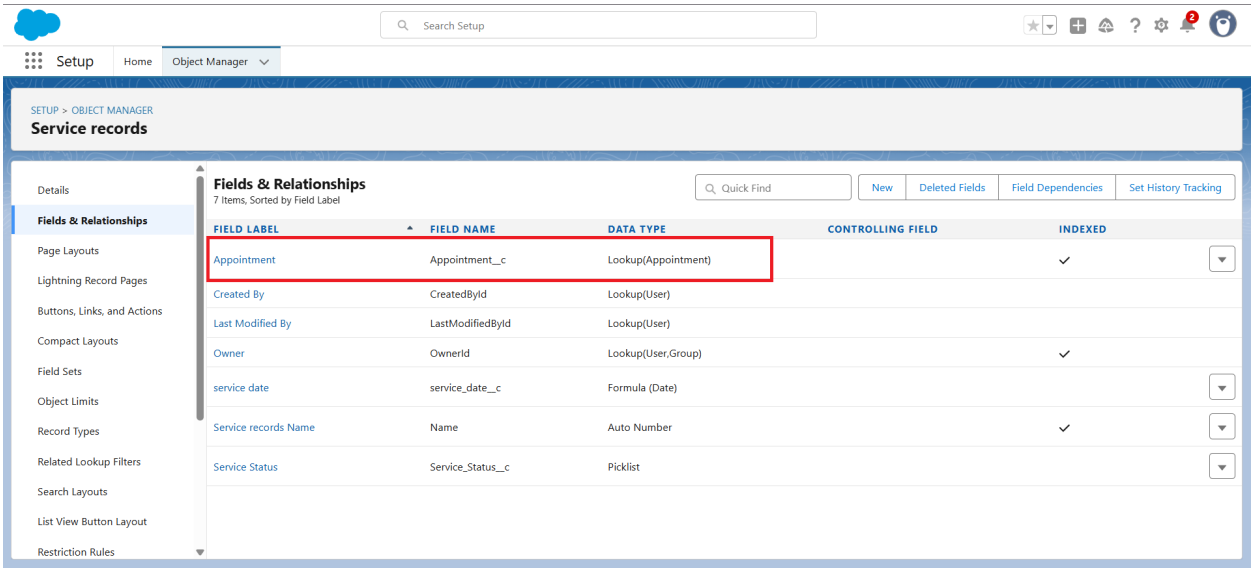
In the Appointment object, I created a Lookup relationship field to link it with the Customer Details object. This setup ensures that every appointment is connected to a specific customer. To achieve this, I navigated to the Appointment object in the Object Manager under Setup, selected "Fields & Relationships," and added a new Lookup Relationship. I selected Customer Details as the related object and completed the configuration by saving the field.



ii. Lookup Field on Service Records Object:

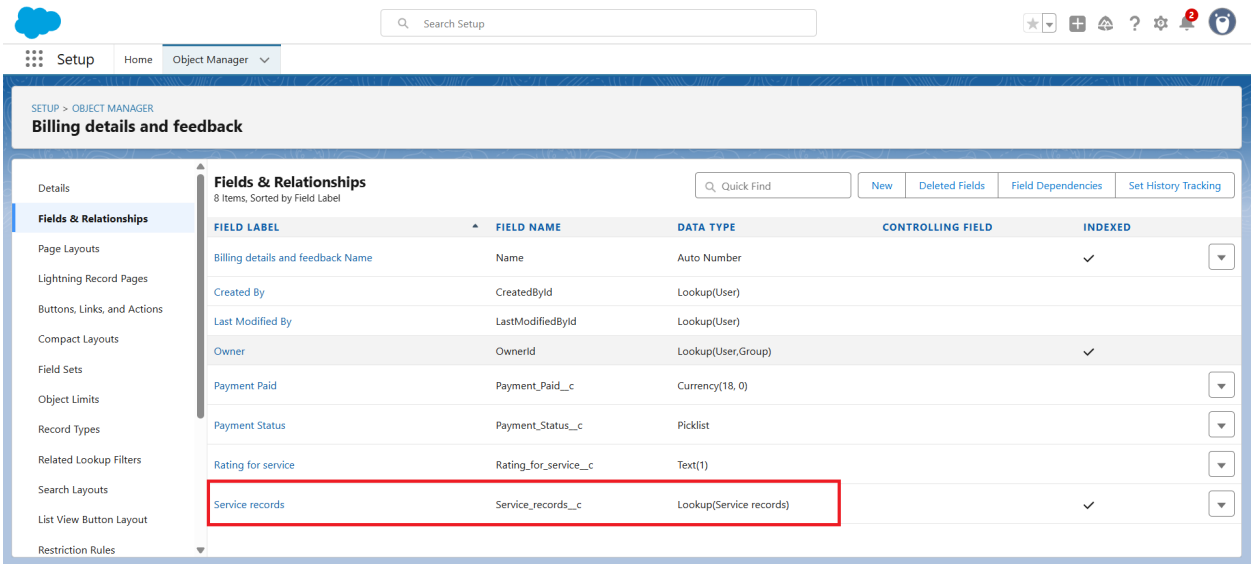
In the Service Records object, I created a Lookup field to connect each entry with its related Appointment. I set this field as mandatory and added a filter to allow selection of

only those appointments that were created before the service record. This filter checks the Appointment Date against the Created Date and displays an error message if the condition isn't satisfied. This approach maintains both data accuracy and logical flow.



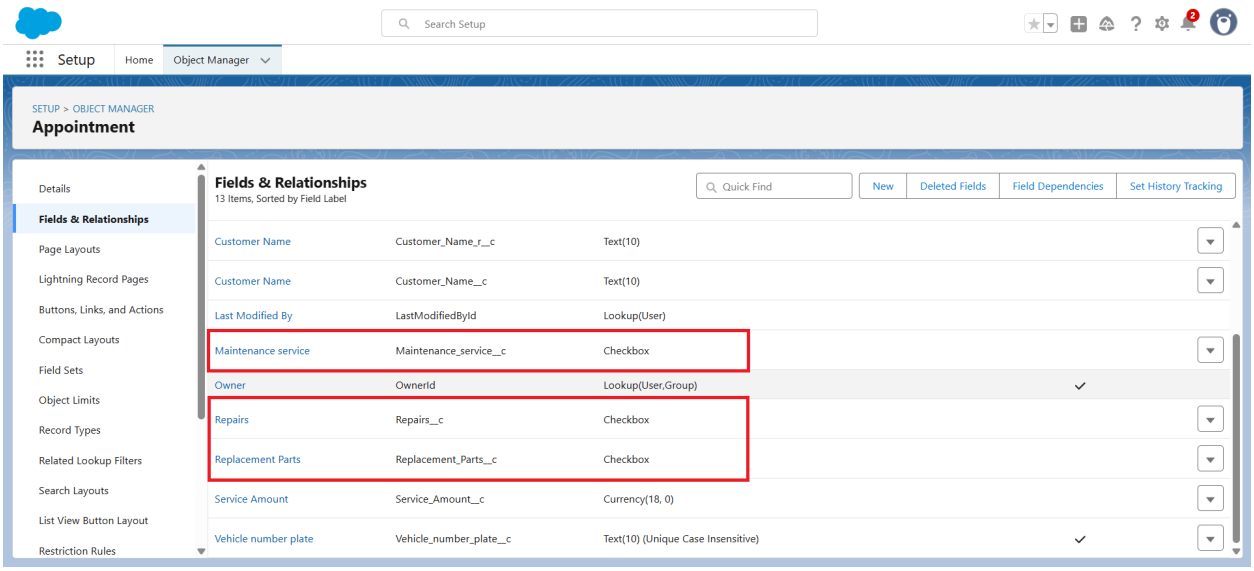
iii.Lookup Field on Billing Details and Feedback Object:

In the Billing Details and Feedback objects, I added a Lookup field to establish a connection with the Service Records. This setup allows each billing entry and feedback to be linked to a specific service. Using the standard process in Object Manager, I created the field and chose Service Records as the related object before saving.



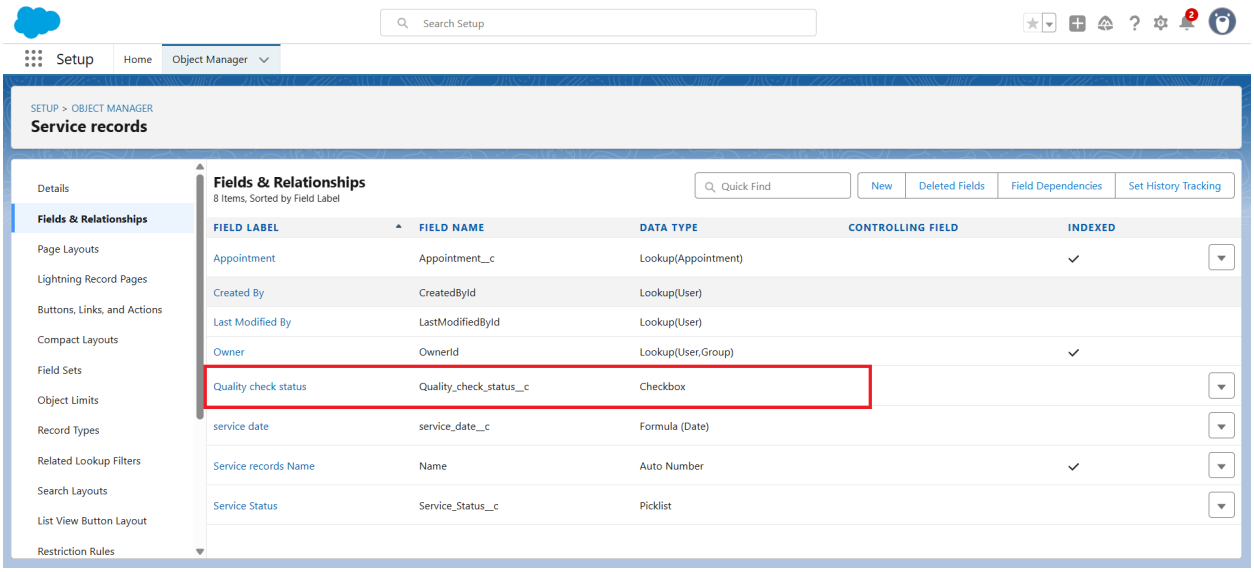
iv. Checkbox Fields on Appointment Object:

I implemented three checkbox fields within the Appointment object to represent the type of service being provided: "Maintenance Service," "Repairs," and "Replacement Parts." Each field was individually created using the Checkbox data type, with the default value set to unchecked. This configuration simplifies the tracking of services chosen during an appointment.



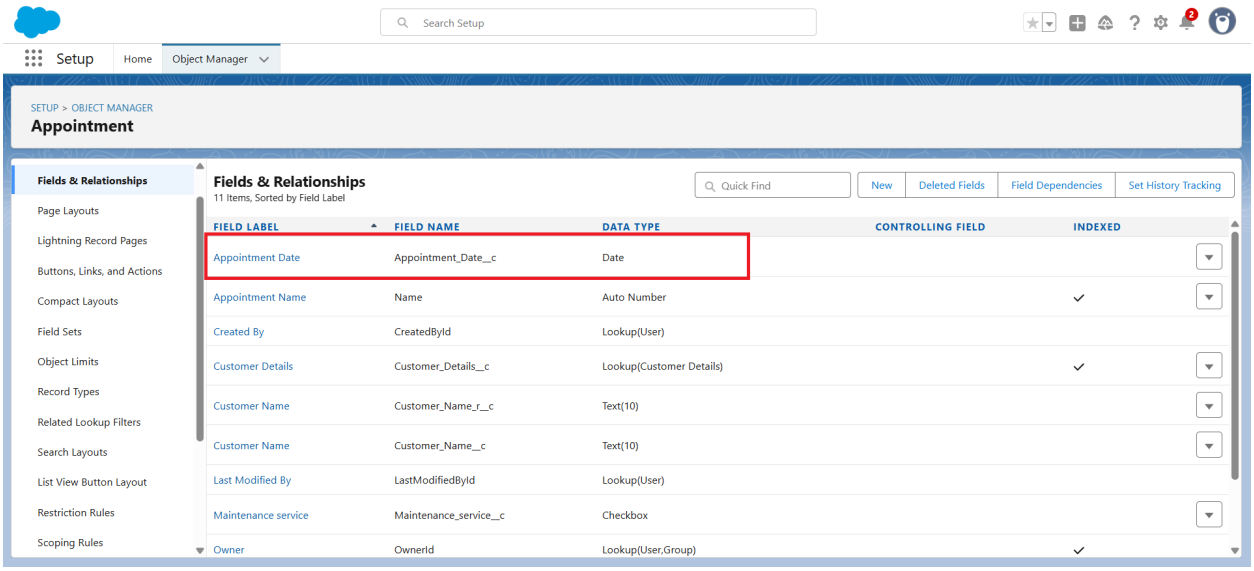
v.Checkbox Field on Service Records Object:

Within the Service Records object, I added a checkbox field named “Quality Check Status” to indicate whether the quality inspection has been completed. The field was configured using the Checkbox data type and set to unchecked by default, allowing service representatives to mark it once the verification process is finished.



vi.Date Field on Appointment Object:

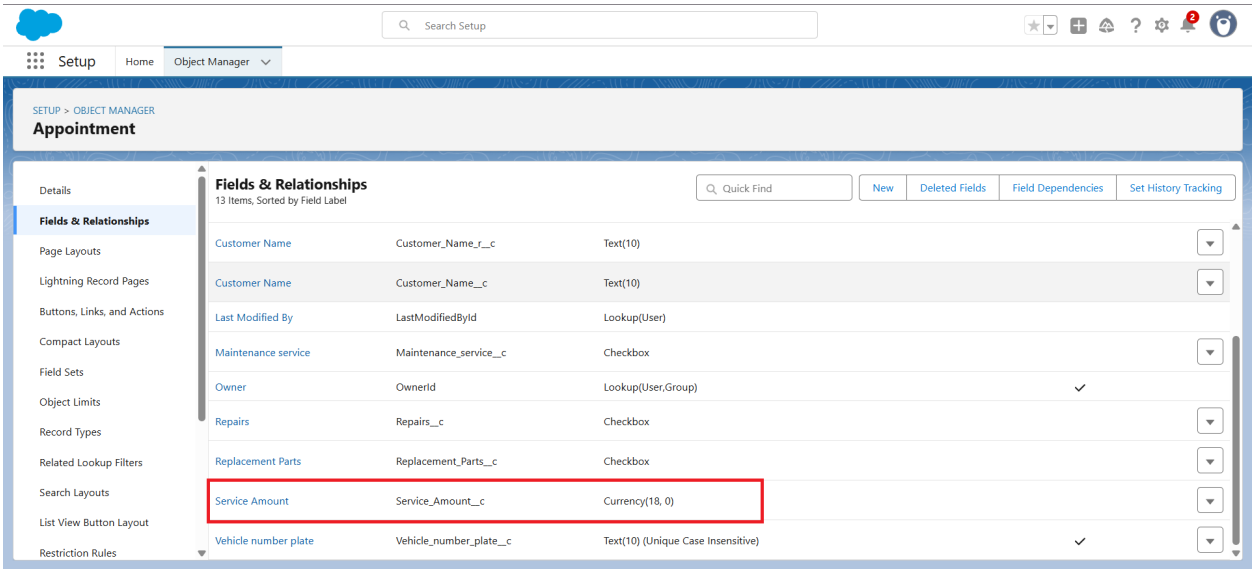
To record the scheduled date of an appointment, I added a Date field named “Appointment Date” within the Appointment object. This field was marked as required to ensure that no appointment can be saved without specifying a date.



vii.Currency Field on Appointment Object:

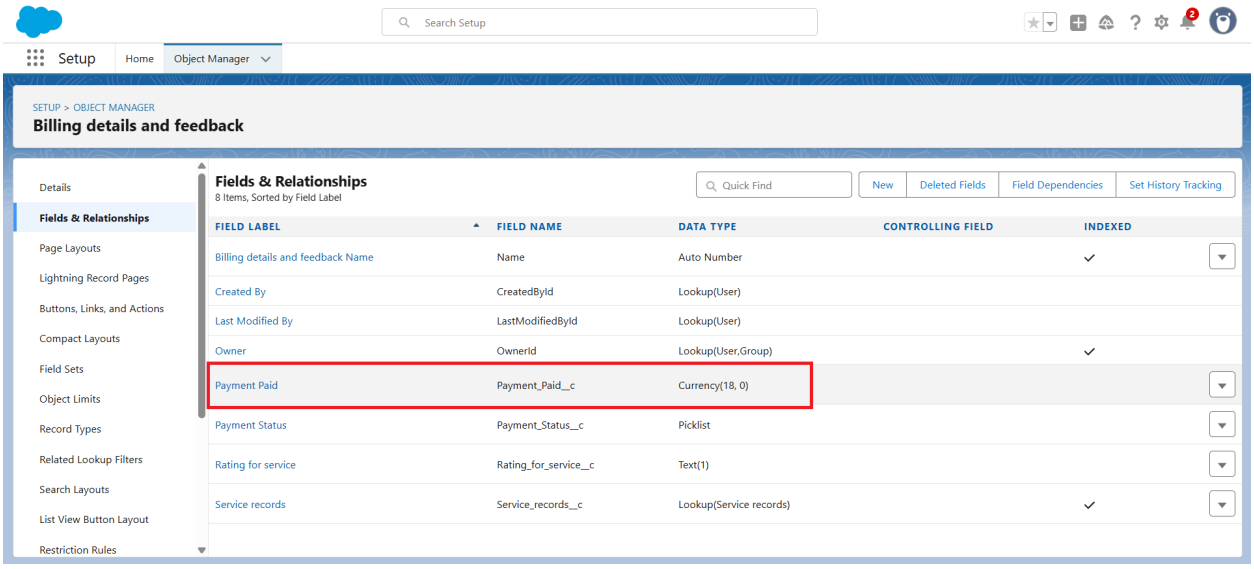
I introduced a Currency field named “Service Amount” in the Appointment object to record the cost associated with the appointment services. To maintain data integrity, I

set the field to read-only for all user profiles during the field-level security configuration.



viii.Currency Field on Billing Details and Feedback Object:

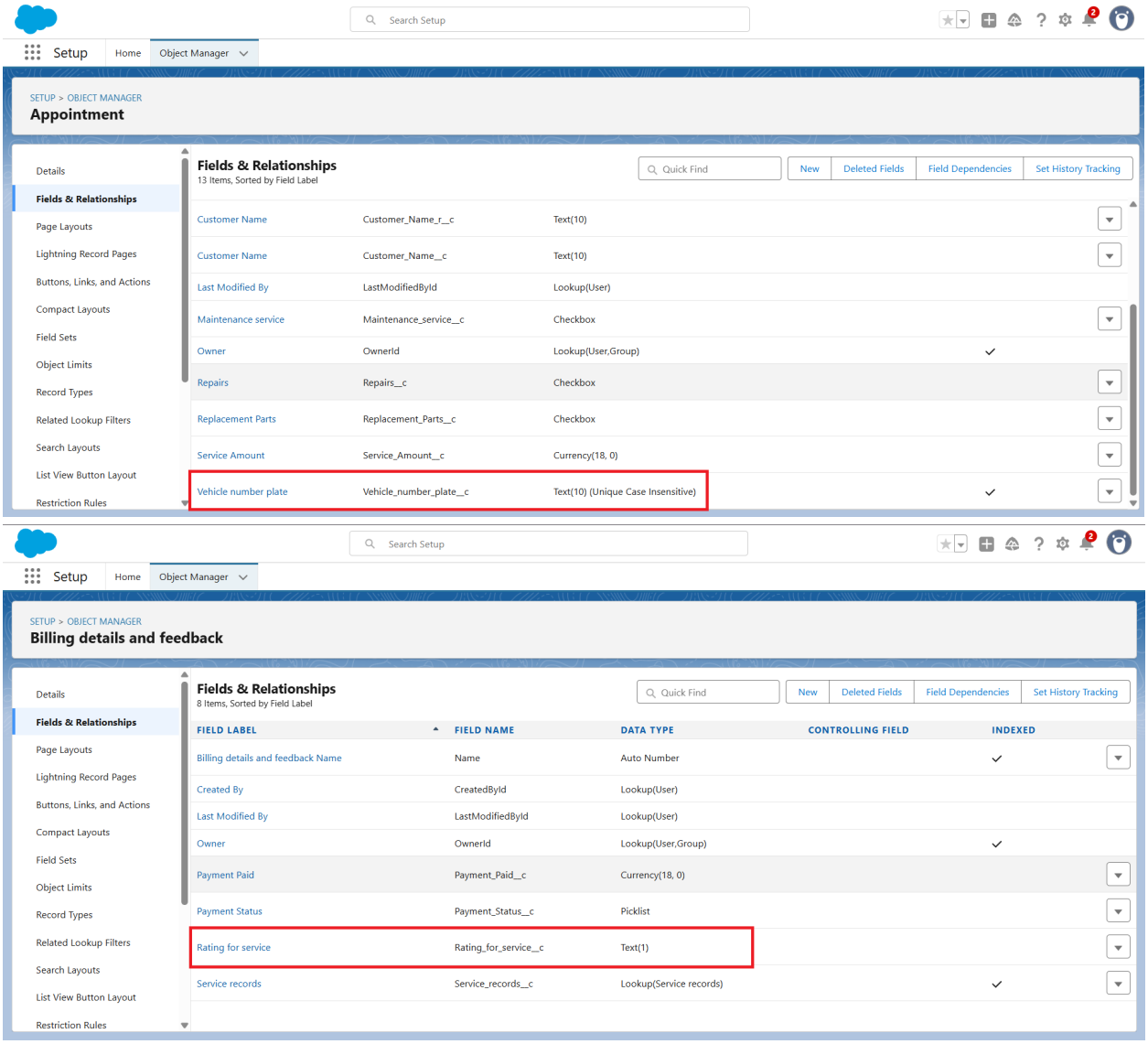
I also added a Currency field named “Payment Paid” in the Billing Details and Feedback object to capture the amount paid by the customer for the service. This field supports accurate tracking of payments, contributing to effective financial management and summary reporting.



ix:Text Field on Appointment Object ,Billing Details and Feedback Object:

To capture key appointment and feedback details, I added a required and unique text

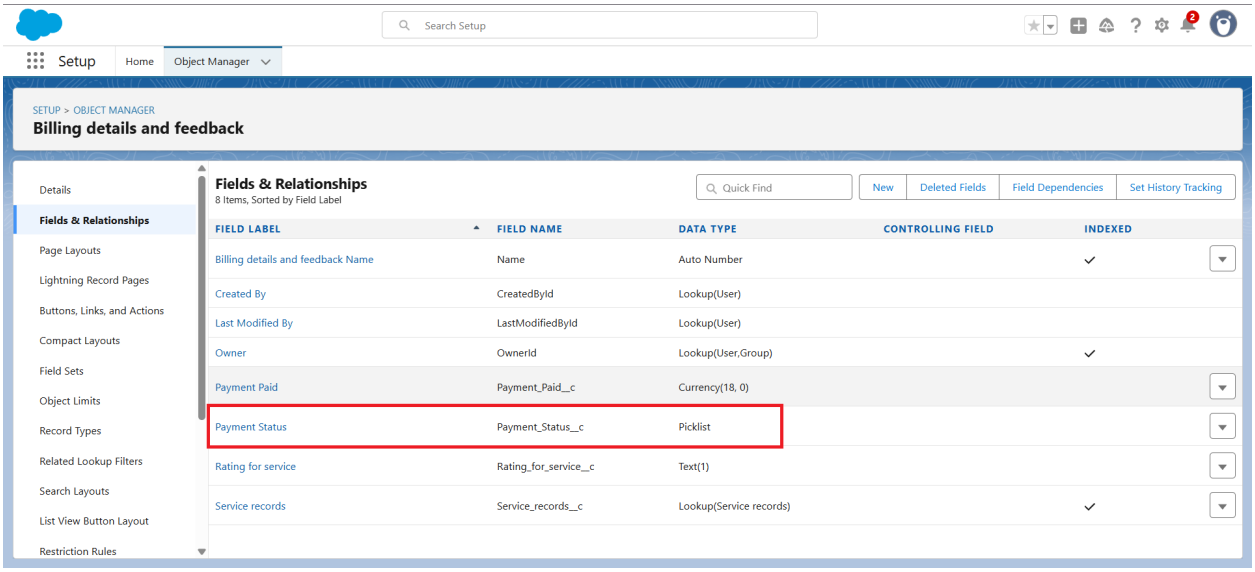
field called “Vehicle Number Plate” (with a maximum length of 10 characters) to the Appointment object. Additionally, a “Rating for Service” text field was included in the Billing Details and Feedback object to let customers give a quick, single-digit rating. These fields help improve data accuracy and enhance the customer experience.

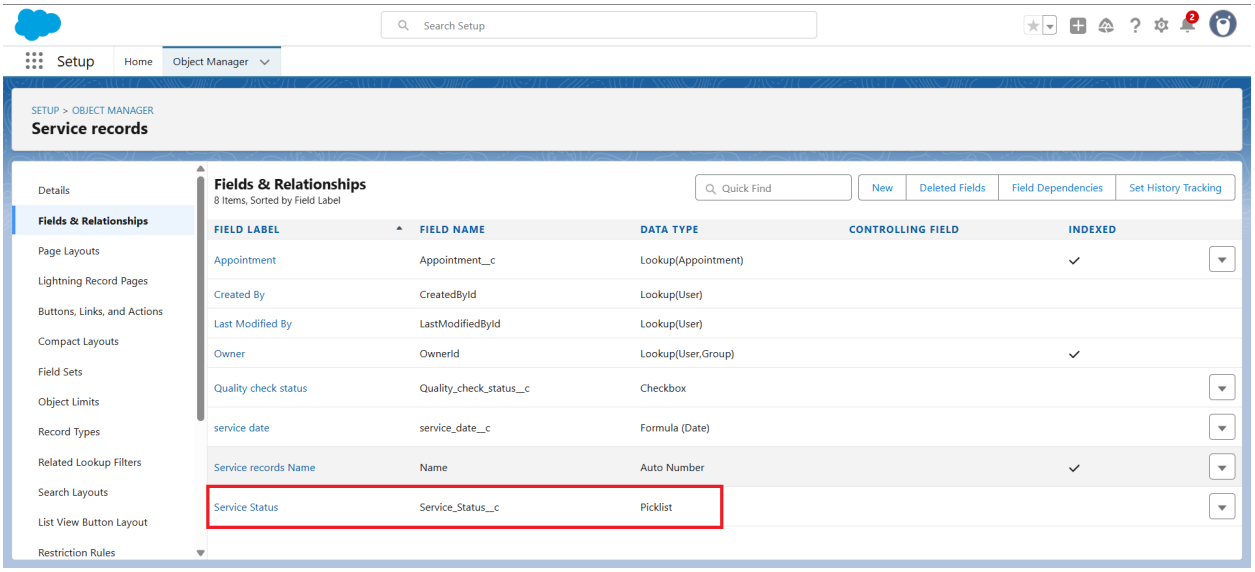


x.Picklist Fields on Service record,Billing Details and Feedback Objects:

To enhance operational efficiency, a "Service Status" picklist was introduced in the Service Records object, including the options "Started" and "Completed" to monitor service progress. Similarly, a "Payment Status" picklist was added to the Billing Details

and Feedback object, offering "Pending" and "Completed" selections to support payment tracking. These additions improve transparency in service and payment processes.



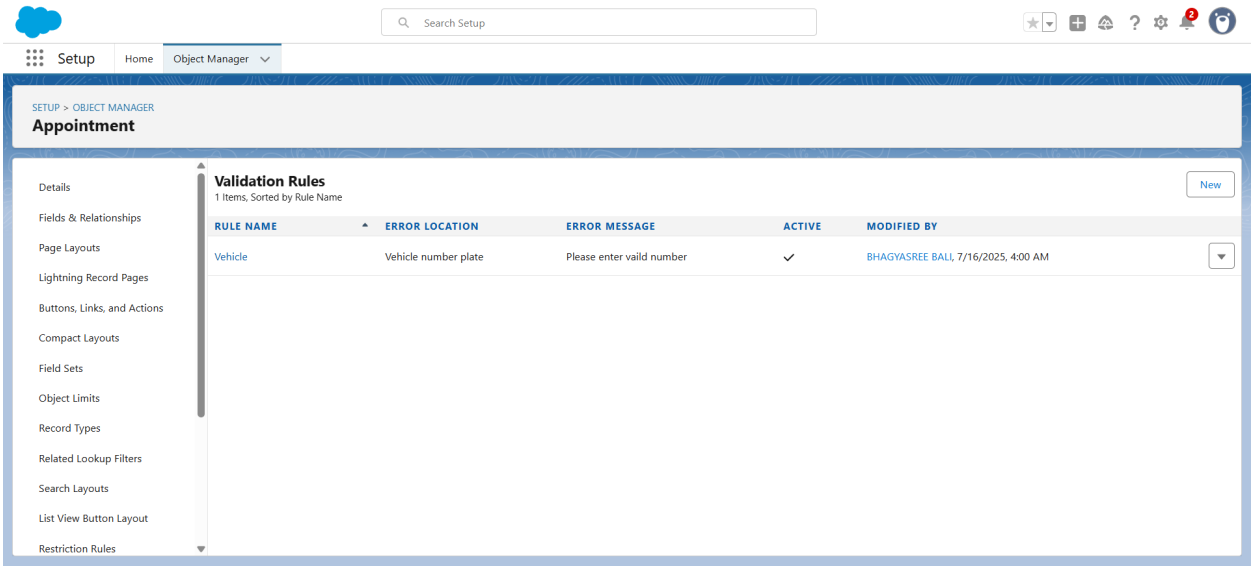


3. Validation Rule

A validation rule in Salesforce is a logic-based mechanism designed to maintain data accuracy and integrity when a user attempts to create or update a record. These rules are activated during the save process and verify that the entered data complies with defined business requirements.

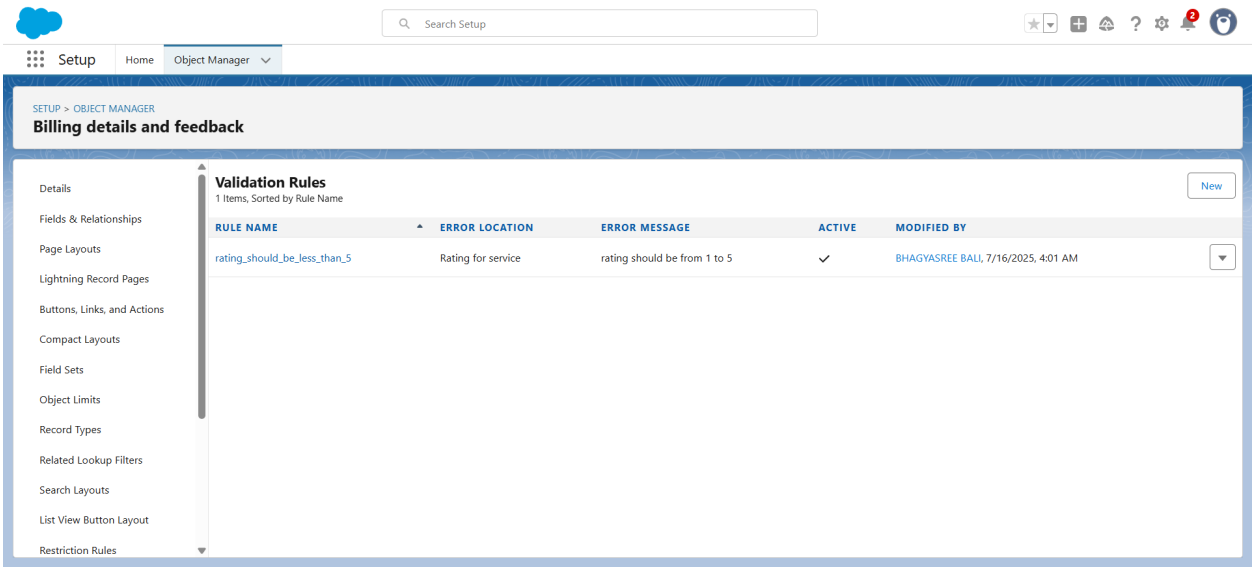
i. Validation Rule on Appointment Object – Vehicle Number Plate Format

To ensure that vehicle numbers are entered correctly, a validation rule named **"Vehicle"** was implemented on the **Appointment** object. This rule uses a **REGEX formula** to verify that the input follows the standard Indian vehicle number plate format. The expected format includes two capital letters, followed by two numbers, then two more capital letters, and finally four digits (e.g., **AP30BT8906**). If the format is incorrect, the system displays the message **"Please enter valid number"** on the **Vehicle Number Plate** field. This rule ensures uniformity and prevents incorrect data entries.



ii.Validation Rule on Billing Details and Feedback Object – Rating Validation:

To ensure accurate ratings, a validation rule named **"rating_should_be_less_than_5"** was added to the **Billing Details and Feedback** object. This rule applies a **REGEX pattern** to confirm that the rating entered is a single digit between **1 and 5**. If the input falls outside this range, the user receives an error message stating **"Rating should be from 1 to 5"** on the **Rating for Service** field. This rule promotes standardized feedback and prevents invalid or extreme rating entries.

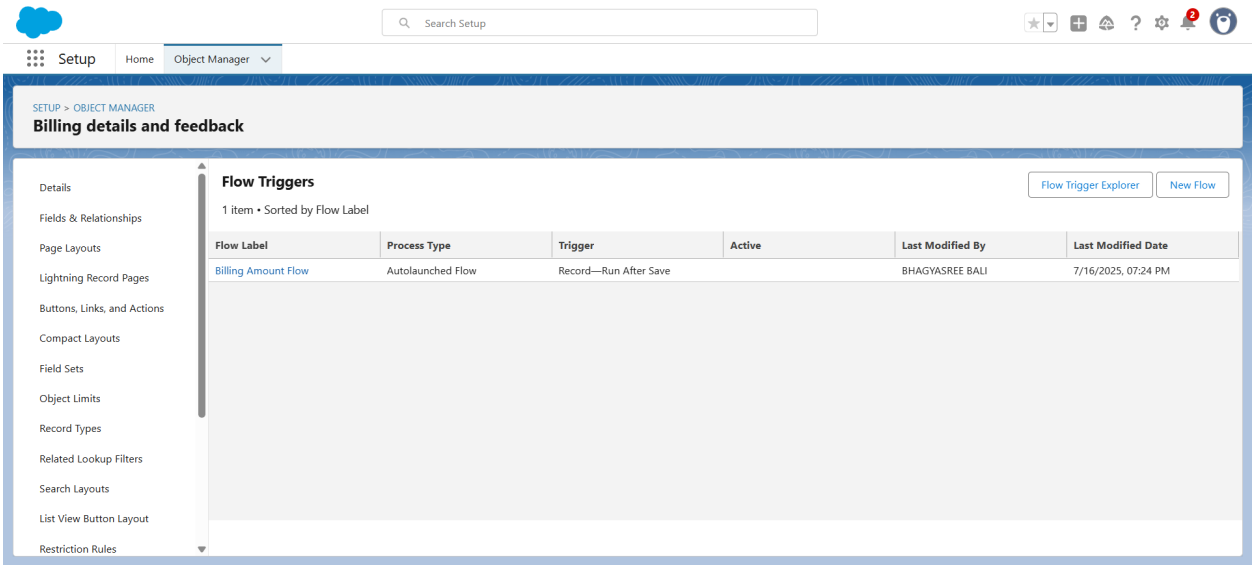


5. Flows

In Salesforce, a **Flow** is a powerful automation tool that enables users to **streamline and automate complex business processes** without the need for coding. By using a **drag-and-drop visual interface**, administrators can build flows that gather data from users, create or update records, send notifications, make decisions, and guide users through step-by-step screens.

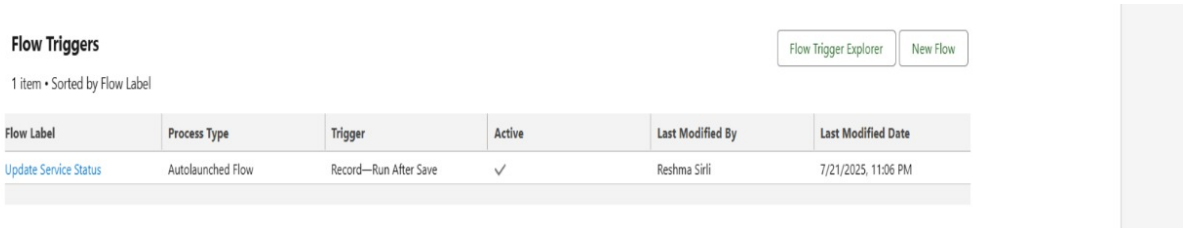
i. Flow on Billing Details and Feedback Object – Amount Update and Email Alert

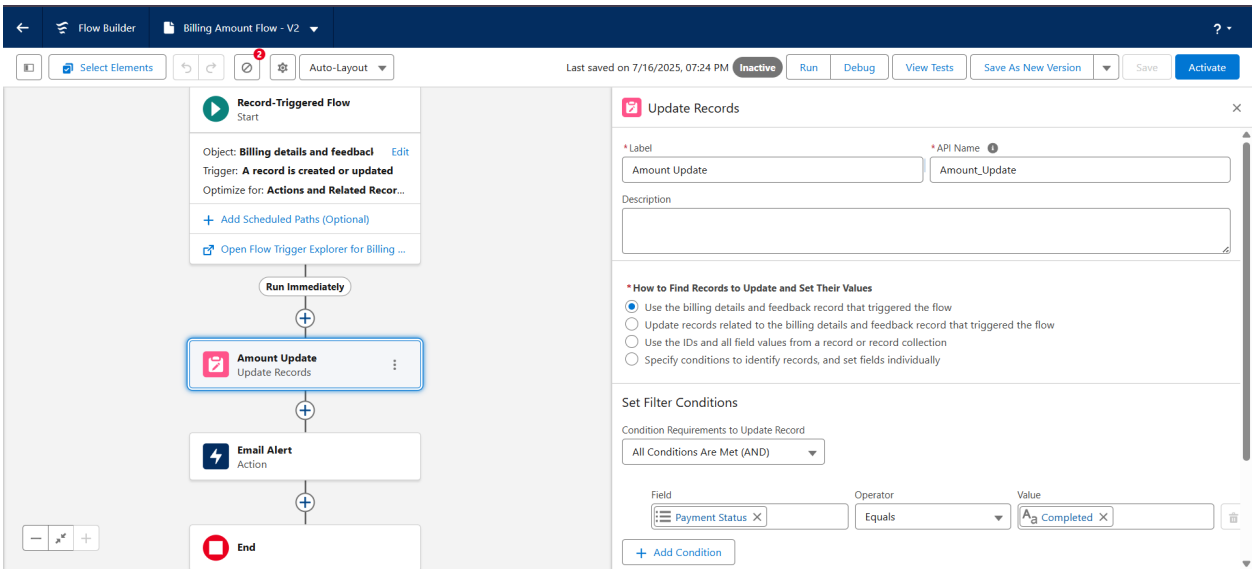
A **Record-Triggered Flow** was set up on the **Billing Details and Feedback** object to automate payment tracking and customer confirmation. When the **Payment Status** changes to **"Completed"**, the flow retrieves the related **Service Amount** and updates the **Payment Paid** field accordingly. Additionally, it sends a **personalized thank-you email** to the customer using a **Text Template** and the **Send Email** action. This automation ensures real-time updates and helps build customer relationships through timely appreciation.



ii. Flow on Service Records Object – Auto Update Service Status

A **Record-Triggered Flow** was implemented on the **Service Records** object to automate service status updates. When the **Quality Check Status** is marked as true, the flow automatically updates the related **Service Status** in the **Billing Details and Feedback** object to "**Completed**". This automation ensures accurate service tracking, eliminates manual updates, and enhances operational efficiency across both service and billing functions.





6. APEX TRIGGERS

In Salesforce, Apex Triggers are blocks of code that execute automatically when certain record events take place—such as insertions, updates, deletions, or restorations. They are often used to carry out custom operations like validating data, sending alerts, or updating records behind the scenes. Triggers help streamline business processes and can be created for both standard and custom objects, usually through the object's "Triggers" settings.

There are two main types of triggers: **Before Triggers** and **After Triggers**. Before Triggers are executed prior to data being stored in the database and are typically used to validate or modify incoming data—for instance, ensuring required fields are filled. After Triggers run once the data has been committed to the database and are useful when performing operations like updating related records or sending notifications based on saved values. These triggers can handle various events such as insert, update, delete, and undelete.

Apex Handler:

To automatically determine the total cost of selected services, an **Apex Handler** was developed with the name `calculateServiceAmountHandler`. This class contains a method that accepts a list of `Appointment__c` records and computes the service cost based on the selected service types, such as **Maintenance**, **Repairs**, or **Replacement Parts**. For instance, if all three services are chosen, the total cost will reflect the

combination. If only one or two are selected, the calculation is adjusted accordingly. This approach ensures pricing accuracy based on the specific mix of services a customer selects.

To implement this logic, an **Apex Trigger** called `calculateServiceAmountTrigger` was created.

This trigger is associated with the `Appointment__c` object and is set to execute **before insert** and **before update** events. This ensures that the service amount is calculated anytime a new appointment is created or an existing one is modified. The trigger simply invokes the appropriate method from the handler class. This structure keeps all business logic centralized in the handler, promoting clean, maintainable, and reusable code, aligned with Salesforce development best practices.

Phase 3: UI/UX Development & Customization

During this stage, a **custom Lightning App** was developed using the **App Manager**. This app acts as a central hub for all project-related objects and features. It includes custom tabs such as **Appointments**, **Billing Details**, and **Feedback**, allowing users to quickly access important records. To improve the user experience, elements like icons, branding, and personalized app settings were also added for better navigation and usability.

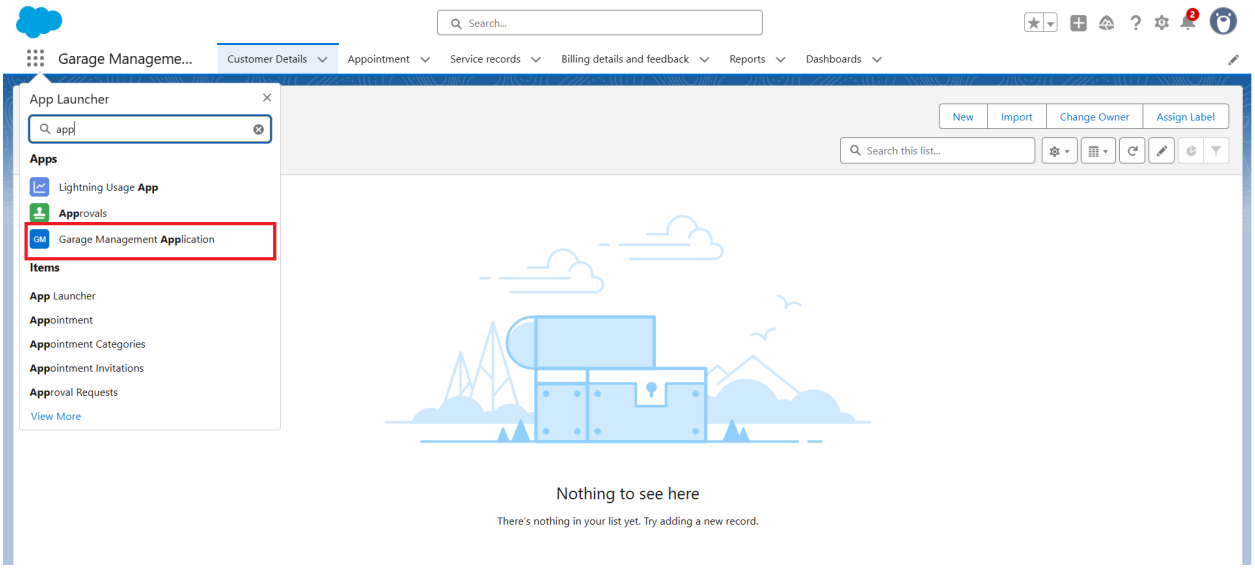
1. Lightning App Setup (App Manager)

As part of this phase, a **custom Lightning App** was set up using the **App Manager** within Salesforce Setup. This app functions as the **main workspace** for users handling vehicle-related services. It delivers a tailored and unified user interface, enabling team members to easily access everything they need from one centralized location—similar to a command center.

We assigned a **custom name**, **logo**, and **navigation style** to the app to align with the project's theme. It was then added to the App Launcher for convenient access. The app included several **custom tabs** tailored for key users, such as:

- **Appointments__c** – used for managing vehicle service schedules
- **Vehicles__c** – for maintaining vehicle and customer data
- **Billing_Details_and_Feedback__c** – to oversee payments and gather feedback
- Other relevant tabs like **Service_Record__c** and **Customer__c**

To enhance usability, we set the **Navigation Style** to **Standard**, enabling users to move smoothly between pages using the top navigation bar. The app also supports **Lightning Record Pages**, dashboards, reports, and custom components—creating a robust and efficient user workspace.



2. Page Layouts & Dynamic Forms

During this phase, we created and refined **Page Layouts** for each object to enhance data entry and maintain a clean interface. Fields were organized in a logical order, and unnecessary elements were removed to keep only the most relevant information visible to users.

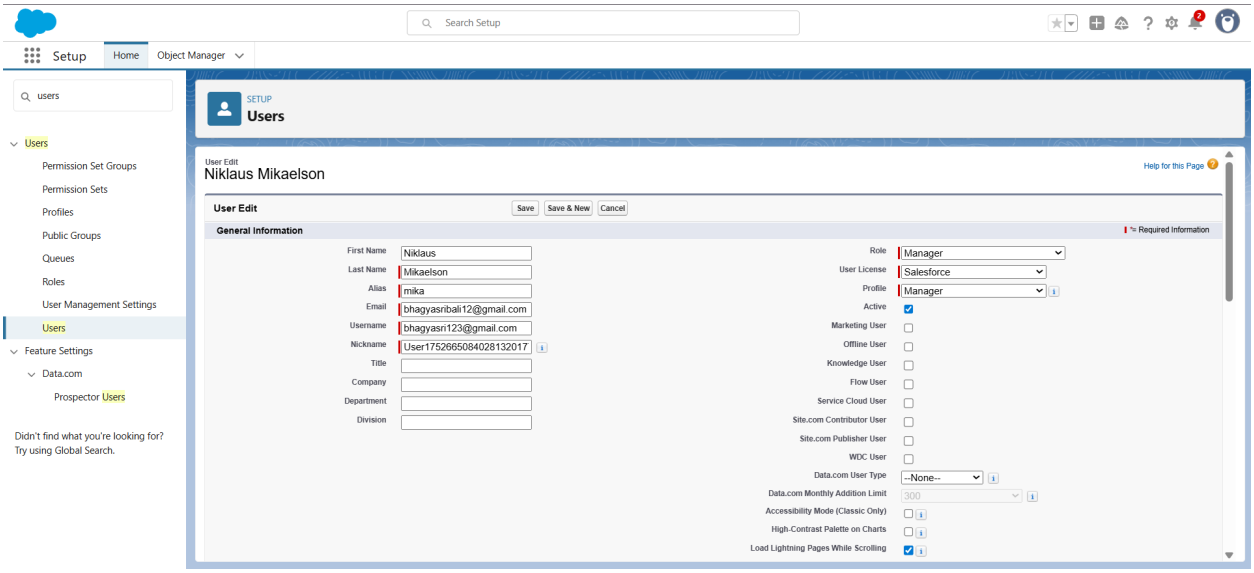
With the help of **Dynamic Forms**, the pages became more interactive and responsive. Fields now appear based on user input. For instance, on the **Appointment__c** page, the **Service Amount** field is displayed only when a specific service (such as Maintenance or Repairs) is chosen. This not only improves user interaction but also reduces potential errors and brings a more modern look to the system.

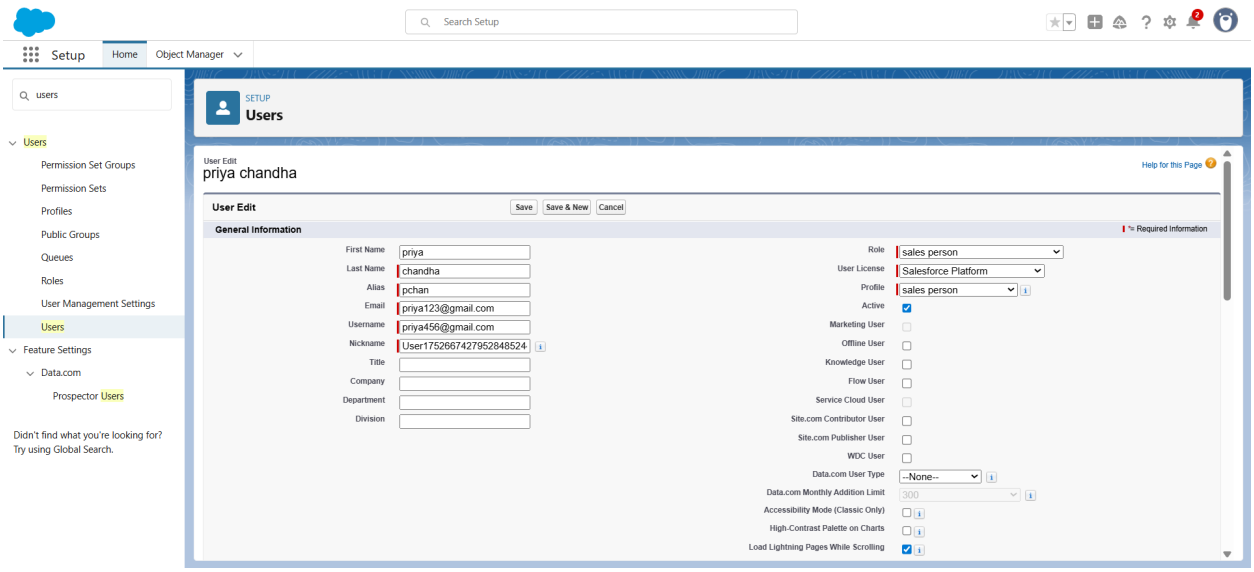
3. User Management

During this phase of development, we implemented an effective user management system to control access within the Salesforce application. Various user profiles were created, such as **Admin**, **Service Staff**, and **Billing Executive**. Each of these profiles was assigned specific roles and responsibilities within the system.

To extend access without changing the base profile, we used **Permission Sets**. For example, if a service team member needed temporary access to reports, a permission set could be assigned to allow that.

We also implemented **Role Hierarchies** so that higher-level users, like Admins, could access data managed by users beneath them. This structure supports data security and ensures that users only have access to information relevant to their role.



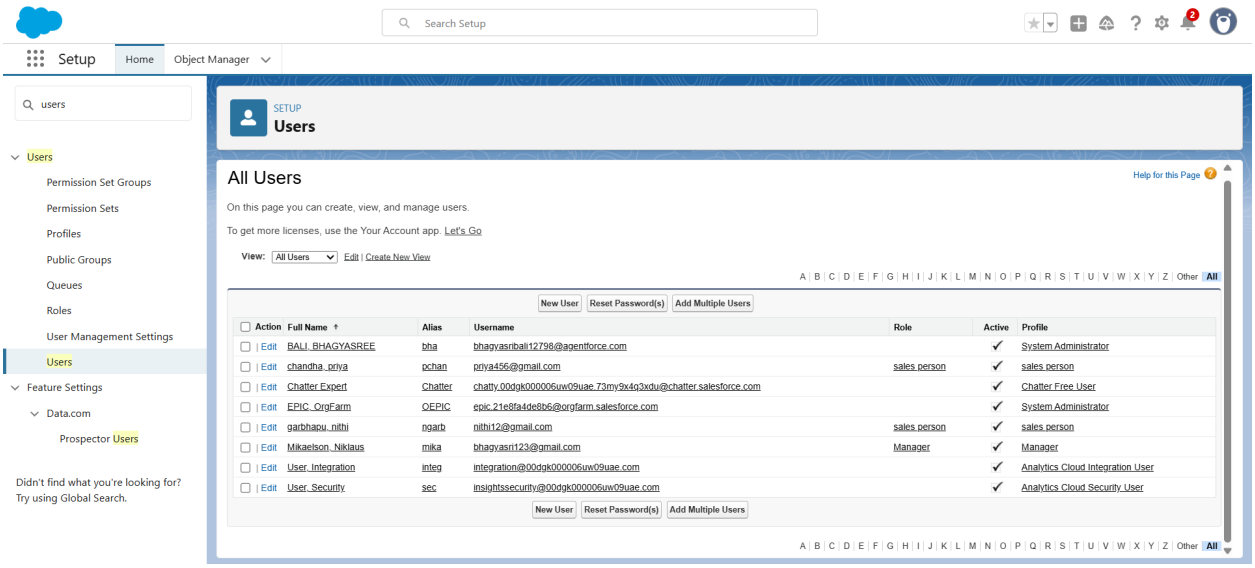


As part of setting up users, we added and configured several users with suitable access roles and profiles to maintain a secure and role-specific system. This was done by navigating to **Setup > Users > New User** and entering the necessary details such as

name, email, username, and alias, and then selecting the appropriate **User License** and **Profile**.

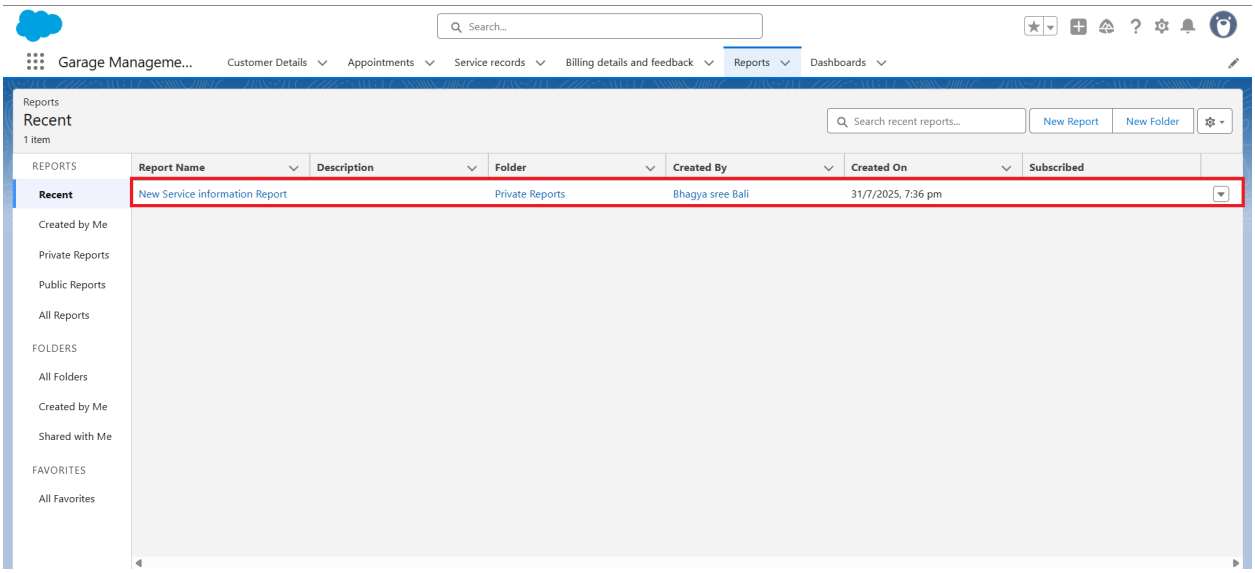
Specifically, we created **three users** with the following attributes:

- **Role**: Sales Person
- **User License**: Salesforce Platform
- **Profile**: Sales Person



Each user was assigned a specific role to maintain proper data visibility according to the role hierarchy. We opted for the **Salesforce Platform** license to grant access to custom apps and standard objects while excluding full CRM features. This was ideal for enabling essential functionalities like access control, approval processes, and report visibility.

Furthermore, we conducted user-specific testing to confirm that permissions, sharing settings, and page layouts performed as expected across different roles. This process also helped us validate key features such as **record-level security**, **field accessibility**, and **workflow automation** from various user viewpoints — ensuring the system remained secure, efficient, and user-friendly for all teams.



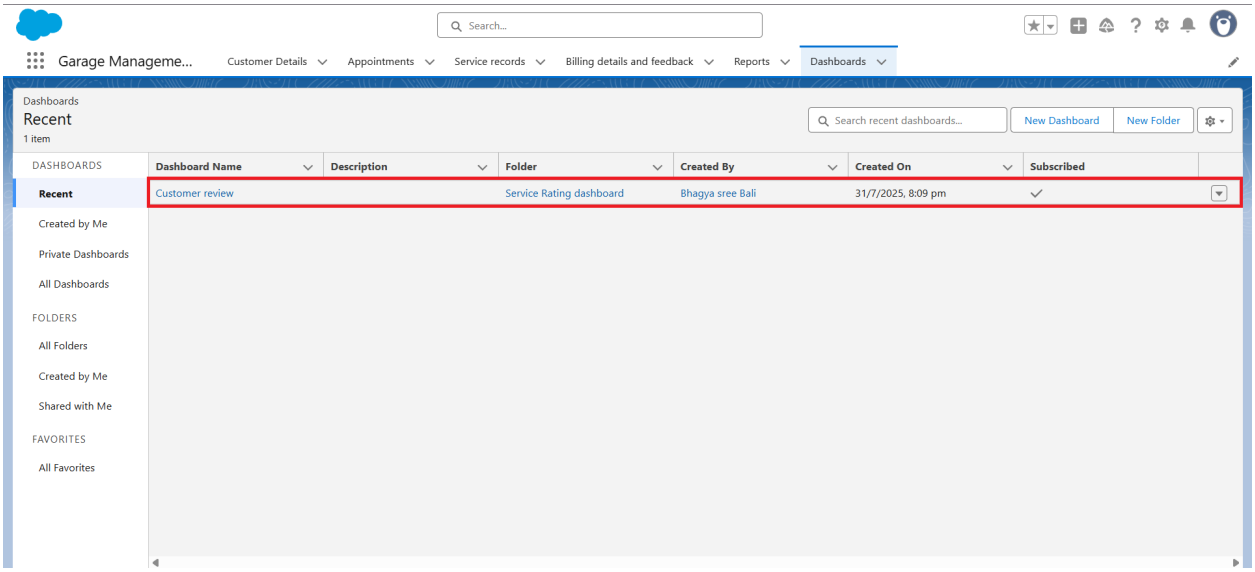
Dashboards in Salesforce offer a visual summary of report data through charts, graphs, and tables. This helps in quickly monitoring key performance indicators (KPIs) and evaluating service performance at a glance.

We used dashboards to track essential metrics such as **service quality**, **customer satisfaction**, and **payment status** trends over time.

We began by creating a **Dashboard Folder** named **Service Rating Dashboard** to neatly organize all dashboards related to our garage management system. This approach helps maintain structure and simplifies folder-level access control when needed.

Within this folder, we built a new **Dashboard** using data from our custom report (**New Service Information Report**). A **Line Chart** was added to visualize service ratings and trends over time. To enhance readability, we applied a clear visual theme and saved the dashboard for ongoing performance tracking.

Additionally, we set up a **weekly subscription** to this dashboard, ensuring automated email updates are sent every Monday. This keeps stakeholders informed without requiring manual effort.



5. Lightning Pages

In this phase, we customized **Lightning Record Pages** using the **Lightning App Builder** to enhance the user experience and improve layout clarity. For key custom objects such as **Appointments** and **Vehicle Details**, we designed tailored record pages specific to each use case.

We included **standard components** like the **Highlights Panel** to display key record information at the top, and **Tabs** to organize related details — such as customer info, vehicle data, and billing — into collapsible sections.

Additionally, we embedded **Report Charts** directly into the record pages to display contextual insights — such as a service rating trend chart or billing summary. This approach not only made the interface more interactive but also allowed users to access meaningful analytics without needing to switch tabs or screens. The design is fully responsive and user-centric, improving the efficiency and productivity of the service staff.

6. Lightning Web Component (LWC) Development *(Optional Bonus)*

To further enhance the **Appointment Page**, we optionally developed a **custom Lightning Web Component (LWC)**. This LWC allowed users to dynamically select a service and instantly view the corresponding service amount — creating a seamless, **real-time price update** experience without the need to refresh the page.

The component mimicked a modern app-like interface, improving interaction speed and usability. It pulled data from related records and rendered it using JavaScript logic within the LWC, following best practices in modular development. Through this approach, the feature demonstrated the power of customization through coding and how Lightning Components can bring life to a standard CRM setup.

Phase 4: Data Migration, Testing & Security

1. Data Loading Process

To import bulk records into Salesforce efficiently, we used both **Data Import Wizard** and **Data Loader** based on the complexity and volume of data.

For simple imports involving fewer than 50,000 records, the **Data Import Wizard** was used directly from the setup menu, offering a user-friendly interface.

For larger datasets, such as service appointments and feedback, we used **Data Loader**, which allowed CSV-based imports with better control over success and error logs. These tools ensured a smooth and structured population of data into custom objects like `Appointments__c` and `Customer_Details__c`.

2. Validation Rules

Validation Rules are used to ensure that the data entered into Salesforce records is complete and accurate before being saved. In our project, we implemented validation rules on important fields to maintain data quality and prevent incomplete submissions.

For example, while creating an `Appointment__c` record, we enforced rules so that fields like **Customer Name** and **Service Date** must be entered. If the user leaves these fields empty or enters them in invalid formats, the system displays a clear error message and prevents the record from being saved.

These rules play a vital role in making sure the data remains consistent, avoids logical errors, and supports proper reporting. By using validation rules, we reduced the chances of missing or incorrect information and improved the reliability of the data.

This is especially important in a vehicle service management setup, where even small details—like timing, service type, and vehicle info—matter a lot.

3. Field History Tracking

Field History Tracking was enabled on key custom objects like `Appointment__c`, `Customer__c`, and `Vehicle__c` to track changes made to important fields. This feature helps us keep a log of what values were changed, when, and by whom.

For example, when the **Service Status** or **Appointment Date** is updated, the history tracking captures the old and new values. This ensures transparency and helps with audits, user accountability, and issue resolution in case of any confusion or mistakes.

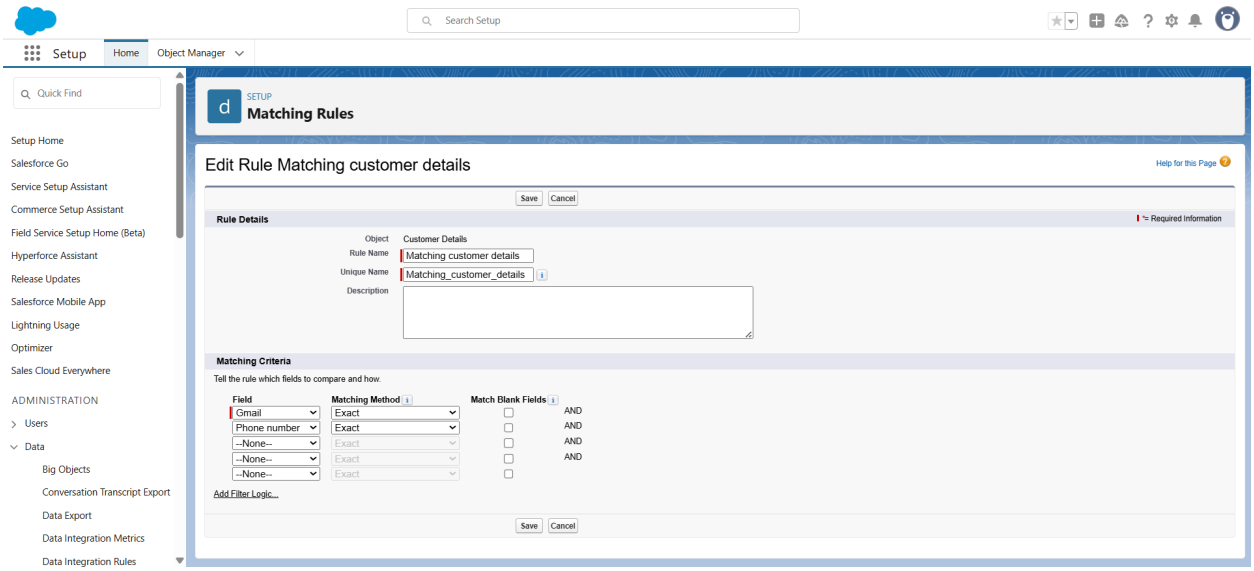
4. Creating a Matching Rule for Customer Details

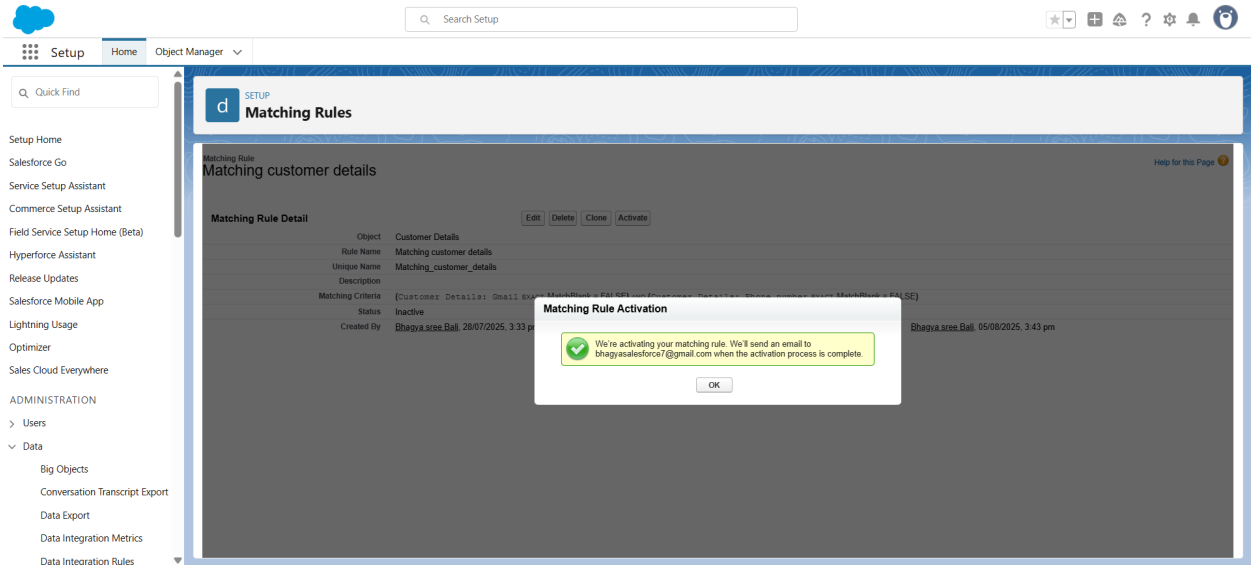
To maintain clean and accurate data in the **Customer Details** object, we first created a custom **Matching Rule**. From the **Setup** menu, we searched for Matching Rules using the **Quick Find** box and clicked on **New Rule**.

We selected the object as **Customer Details** and proceeded to define the rule. The rule was named *Matching Customer Details*, and its unique name was auto-filled by Salesforce.

For the matching criteria, we chose two key fields: **Email** and **Phone Number**, both set to the **Exact Match** condition. This ensures that records are flagged as matches only when both fields are exactly the same.

After saving the rule, we **activated** it so that it could be used in duplicate detection.





5. Creating a Duplicate Rule for Customer Details

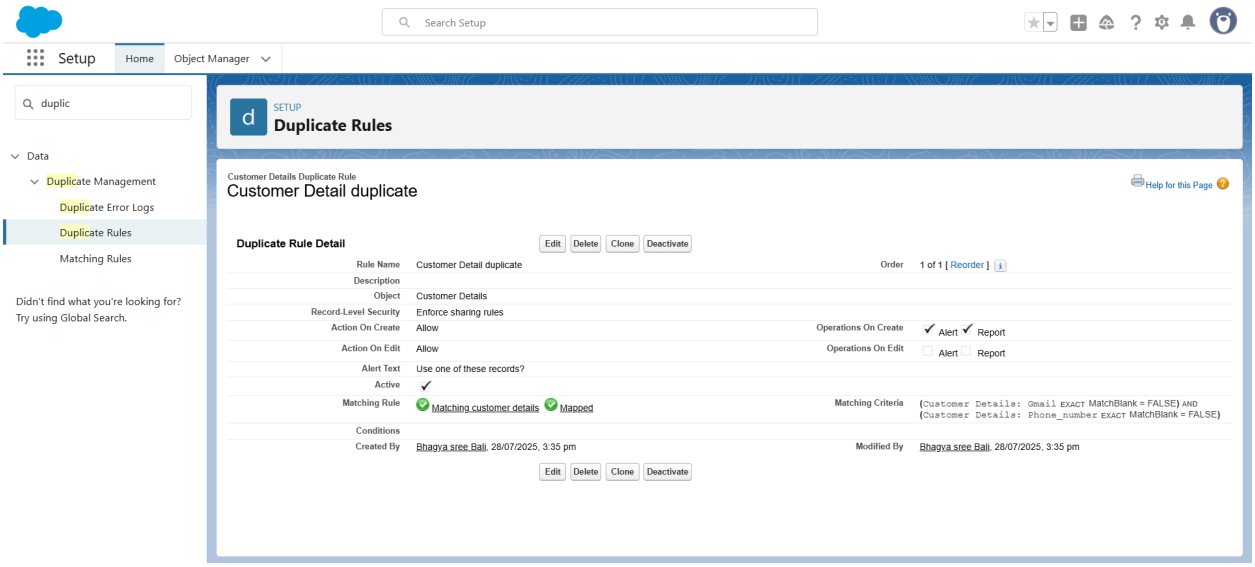
After setting up the **Matching Rule**, we moved on to creating a **Duplicate Rule**. Again, from **Setup**, we searched for **Duplicate Rules** and clicked on **New Rule**. We selected **Customer Details** as the object. The rule was named *Customer Detail Duplicate*, and under the **Matching Rule** section, we selected the previously created rule *Matching Customer Details*. This ensures that the system will use our exact **Email** and **Phone Number** criteria to detect duplicates. Once configured, the rule was saved and activated.

Now, whenever a duplicate customer record is entered, Salesforce will detect it and either block or warn the user based on the rule's settings — helping us avoid duplicate

entries and maintain clean data integrity. In our project, we created a Duplicate Rule for the **Customer Details** object. This rule uses a Matching Rule that checks for duplicates based on exact matches of **Email** and **Phone Number** fields.

After configuring the Duplicate Rule, we activated it to ensure it is enforced during record creation or updates — helping prevent redundancy and confusion in customer data.

This rule is especially valuable when the team is importing large amounts of customer information or when multiple users are handling customer entries. It ensures that the data stays unified and consistent, contributing to better reporting, efficiency, and improved user experience. Custom alert messages can also be set, which notify users in case a duplicate is detected.



6. Profiles

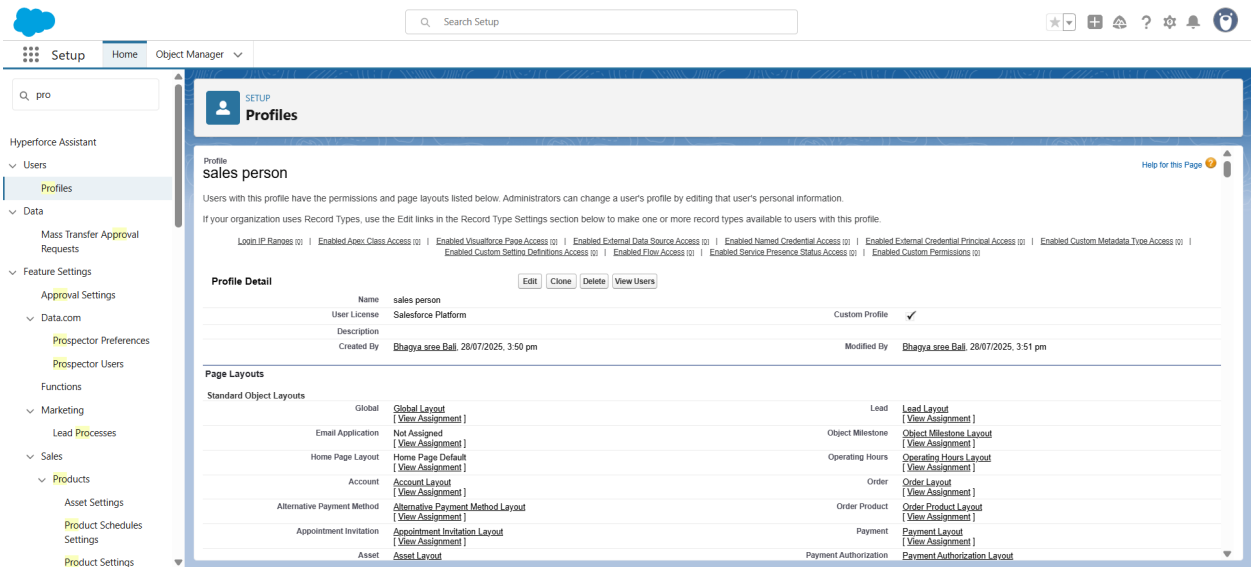
A **Profile** in Salesforce is a collection of settings and permissions that determine what users can see and do within the system. It controls access at various levels such as objects, fields, tabs, apps, and system features. Specifically, profiles manage object permissions (like create, read, edit, delete), field-level permissions, tab visibility, user permissions, Apex class and Visualforce page access, record types, login hours, and IP address ranges. Every user in Salesforce is assigned **one profile**, which defines their functional access according to their job responsibilities.

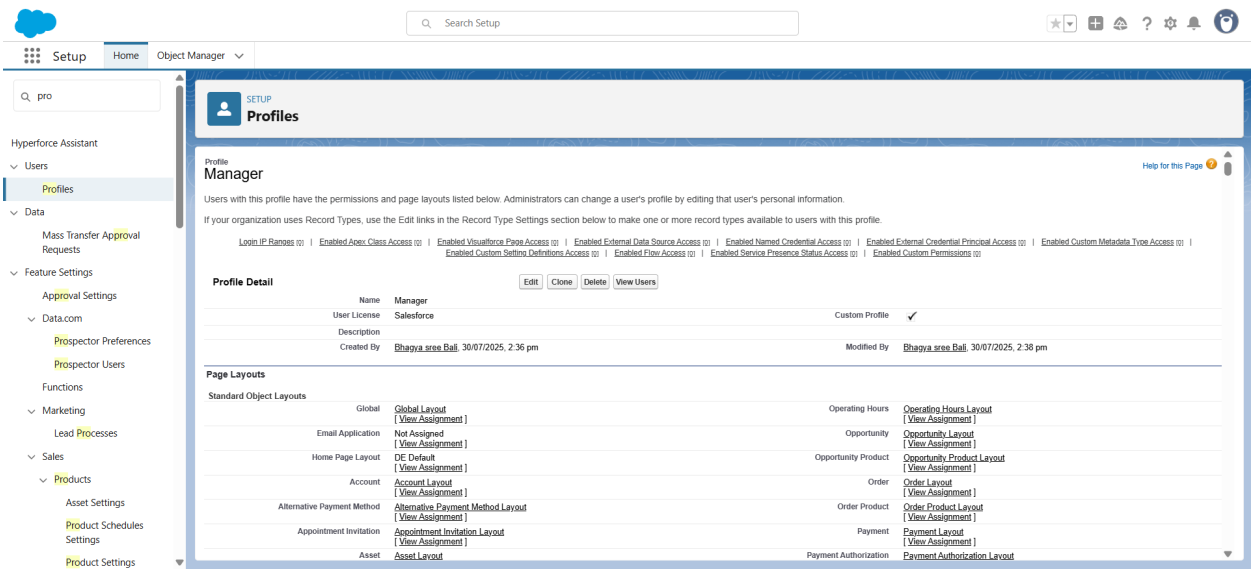
There are two main types of profiles in Salesforce: **Standard Profiles** and **Custom Profiles**.

- **Standard Profiles** are provided by Salesforce by default and include roles such as *System Administrator*, *Standard User*, *Read Only*, *Marketing User*, *Contract Manager*, and *Solution Manager*. These profiles come with predefined permissions and cannot be deleted or renamed.
- On the other hand, **Custom Profiles** are created by administrators to suit specific business needs. They offer greater flexibility in assigning customized permissions and can be deleted if no users are currently assigned to them.

In this project, a custom profile named **"Sales Person"** was created and assigned to users handling customer and booking data. This profile granted access only to the necessary objects and features, ensuring security and role-specific usability.

To create a **Manager Profile**, begin by going to **Setup**, then type **"Profiles"** into the Quick Find box and select the **Profiles** link. On the Profiles page, identify and **clone the "Standard User"** profile, as it provides a good baseline of permissions. Give the new profile the name **"Manager"** and click **Save**.This cloned profile can now be customized to suit the managerial responsibilities.

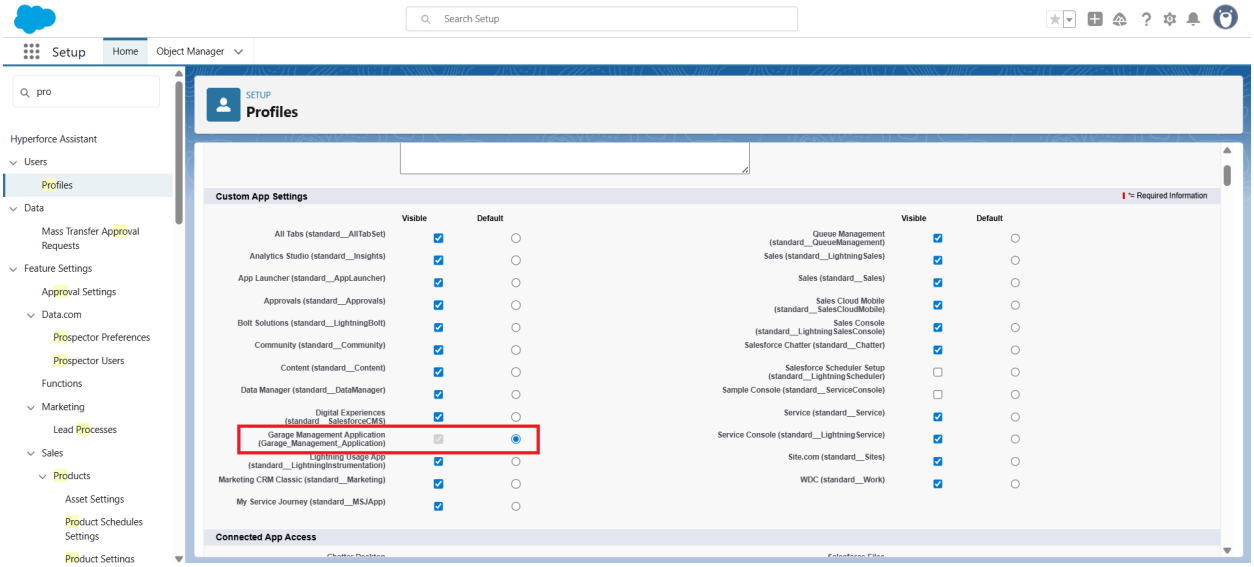


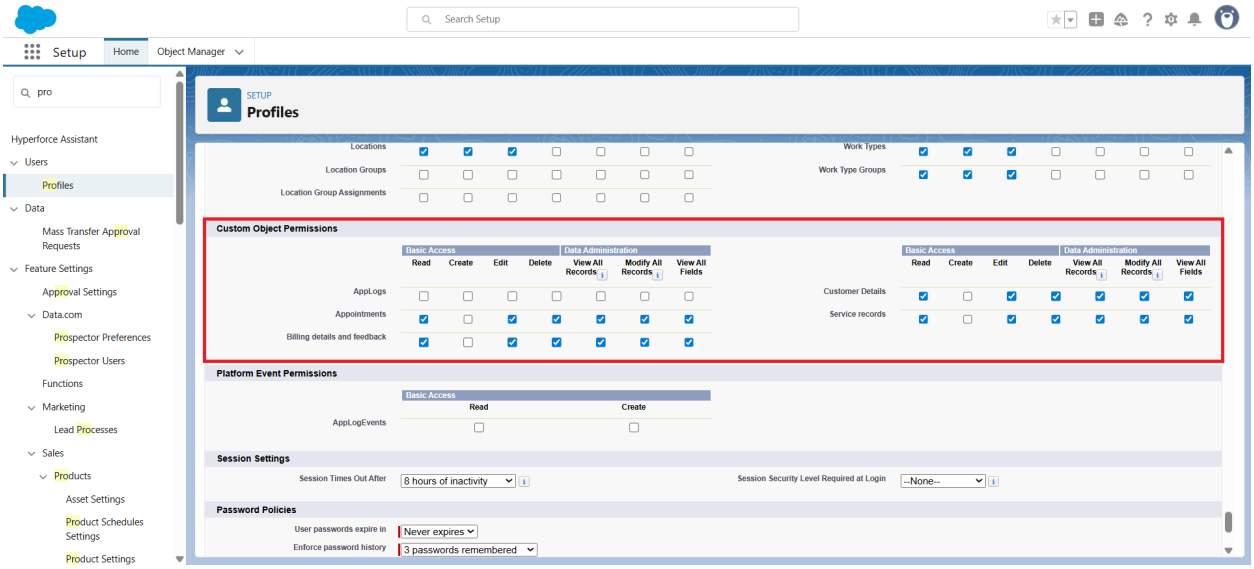


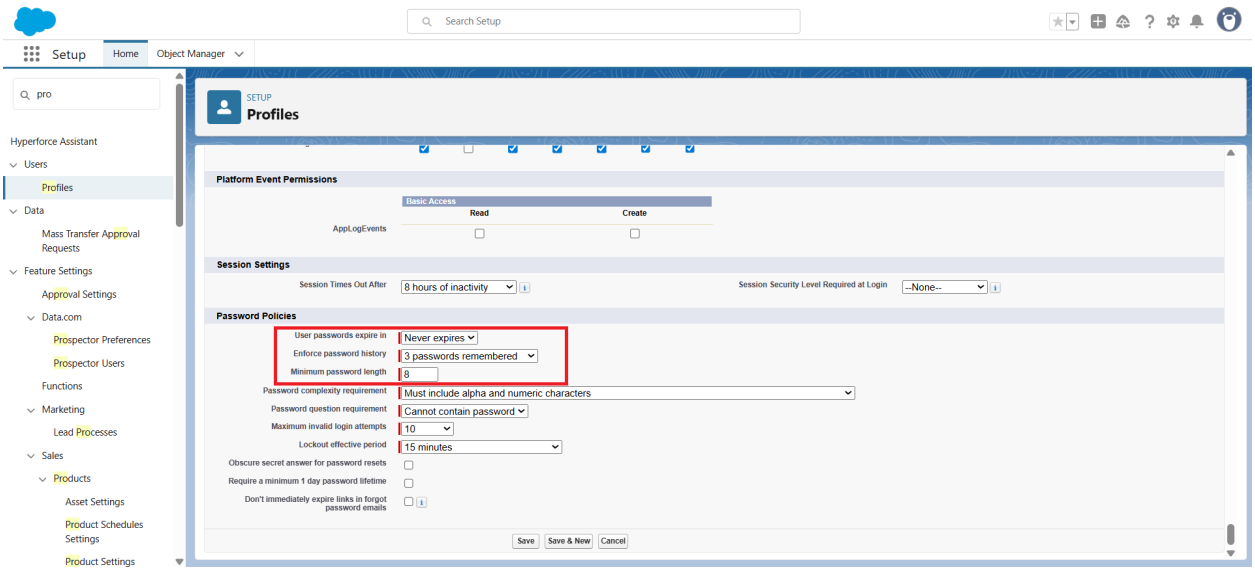
Next, on the profile detail page, click **Edit** to modify specific settings. In the Custom App Settings section, set **Garage Management** as the default app to ensure the Manager lands on the relevant interface upon login. Then, scroll down to **Custom Object Permissions**.

For the **Appointments, Billing Details & Feedback, Service Records, and Customer Details** objects, assign the necessary access rights. Usually, for a Manager, full CRUD access (Create, Read, Edit, Delete) is needed to oversee and manage operations effectively.

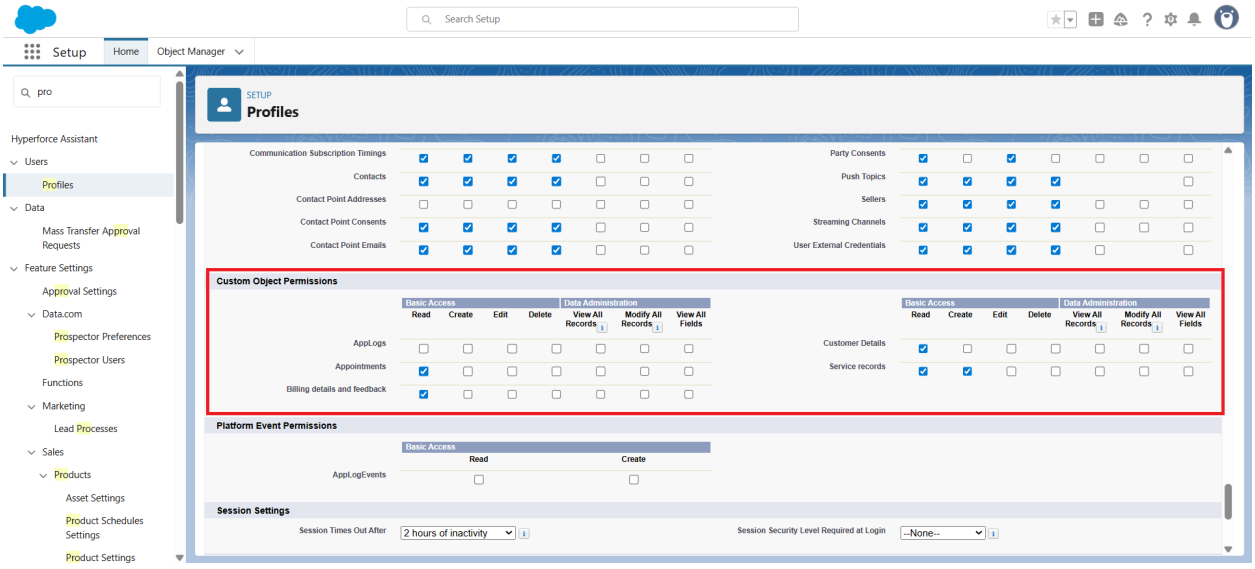
In the **Session Settings**, change the **'Timeout after'** value to 8 hours of inactivity to reduce frequent re-logins during long working hours. Update the **Password Policies** to enhance usability: set the password to **'Never Expire'**, which is ideal for trusted managerial roles, and enforce a minimum password length of 8 characters for basic security. After reviewing all changes, click **Save** to finalize the Manager profile.


Search Setup
Setup
Home
Object Manager
pro
Hyperforce Assistant
Users
Profiles
Data
Mass Transfer Approval Requests
Feature Settings
Approval Settings
Data.com
Prospector Preferences
Prospector Users
Functions
Marketing
Lead Processes
Sales
Products
Asset Settings
Product Schedules Settings
Product Settings
SETUP
Profiles
Custom App Settings
Required Information
Visible
Default
All Tabs (standard__AllTabSet)
Analytics Studio (standard__Insights)
App Launcher (standard__AppLauncher)
Approvals (standard__Approvals)
Bolt Solutions (standard__LightningBolt)
Community (standard__Community)
Content (standard__Content)
Data Manager (standard__DataManager)
Digital Experiences (standard__SalesforceCMS)
Garage Management Application (Garage_Management_Application)
Lightning Usage App (standard__LightningInstrumentation)
Marketing CRM Classic (standard__Marketing)
My Service Journey (standard__MSJApp)
Queue Management (standard__QueueManagement)
Sales (standard__LightningSales)
Sales (standard__Sales)
Sales Cloud Mobile (standard__SalesCloudMobile)
Sales Console (standard__LightningSalesConsole)
Salesforce Chatter (standard__Chatter)
Salesforce Scheduler Setup (standard__LightningScheduler)
Sample Console (standard__ServiceConsole)
Service (standard__Service)
Service Console (standard__LightningService)
Site.com (standard__Sites)
WDC (standard__Work)
Connected App Access


Search Setup
Setup
Home
Object Manager
pro
Hyperforce Assistant
Users
Profiles
Data
Mass Transfer Approval Requests
Feature Settings
Approval Settings
Data.com
Prospector Preferences
Prospector Users
Functions
Marketing
Lead Processes
Sales
Products
Asset Settings
Product Schedules Settings
Product Settings
SETUP
Profiles
Locations
Location Groups
Location Group Assignments
Work Types
Work Type Groups
Custom Object Permissions
Basic Access
Data Administration
Read
Create
Edit
Delete
View All Records
Modify All Records
View All Fields
AppLogs
Appointments
Billing details and feedback
Customer Details
Service records
Platform Event Permissions
AppLogEvents
Session Settings
Session Times Out After
8 hours of inactivity
Session Security Level Required at Login
--None--
Password Policies
User passwords expire in
Never expires
Enforce password history
3 passwords remembered



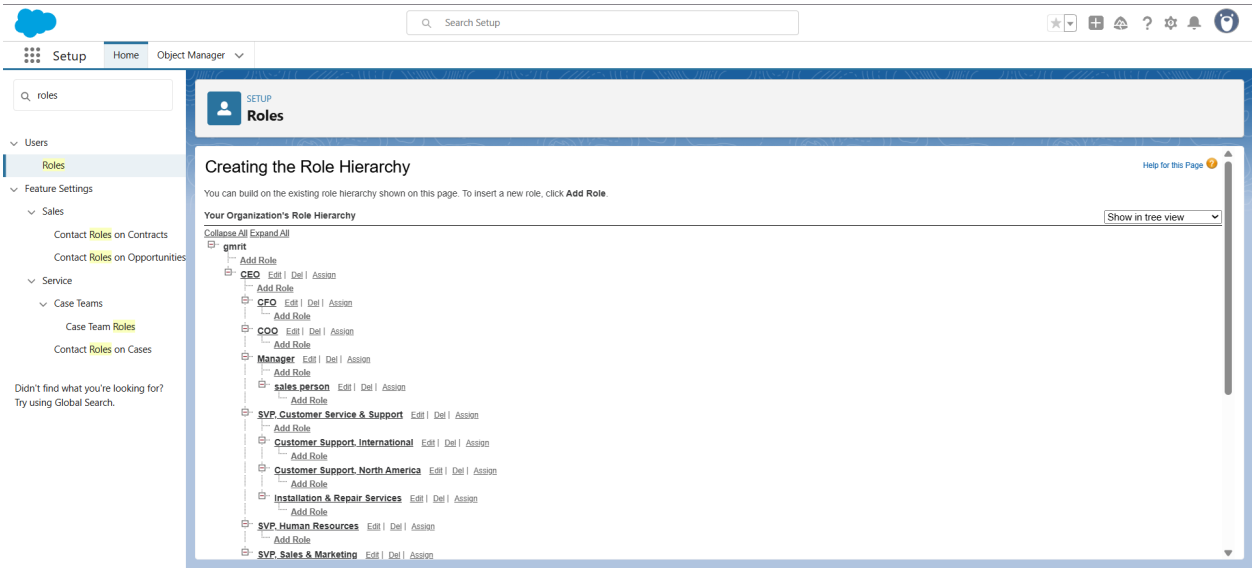
For the **Sales Person Profile**, repeat similar steps. Navigate to **Profiles**, then clone the **Salesforce Platform User** profile. Name the new profile **Sales Person** and click **Save**. On the profile page, click **Edit**. Again, set the **Garage Management** app as the default under **Custom App Settings**. Scroll to **Custom Object Permissions** and assign the required permissions for **Appointments, Billing Details & Feedback, Service Records**, and **Customer Details**. However, unlike the Manager profile, the Sales Person typically only gets **Read** and **Create** permissions, possibly **Edit** access depending on the business rules, but usually no **Delete** rights to prevent accidental data loss.



This approach ensures that users are aligned with organizational responsibilities and security protocols. These structured profiles help streamline access and functionality according to the user's role, enhancing overall system usability and data safety.

7. Roles and Role Hierarchy

A role defines a user's position in the organizational hierarchy and determines record-level access to data. Unlike profiles, which control what a user can do, roles control what a user can see. The role hierarchy allows users at higher levels (like managers) to access, edit, or report on all the data owned by users beneath them in the hierarchy. This is especially useful for sales and service organizations where visibility of team performance and customer records is necessary. For example, a Regional Manager can view all records created by Sales Reps under their team.



To create a **Manager role**, go to **Setup** → **search for Roles** in the Quick Find box → click on **Set Up Roles** → click **Expand All**, then click **Add Role** under the desired superior role. Enter label as **"Manager"** and the Role Name auto-fills. Click **Save**. This new role fits into your role hierarchy.

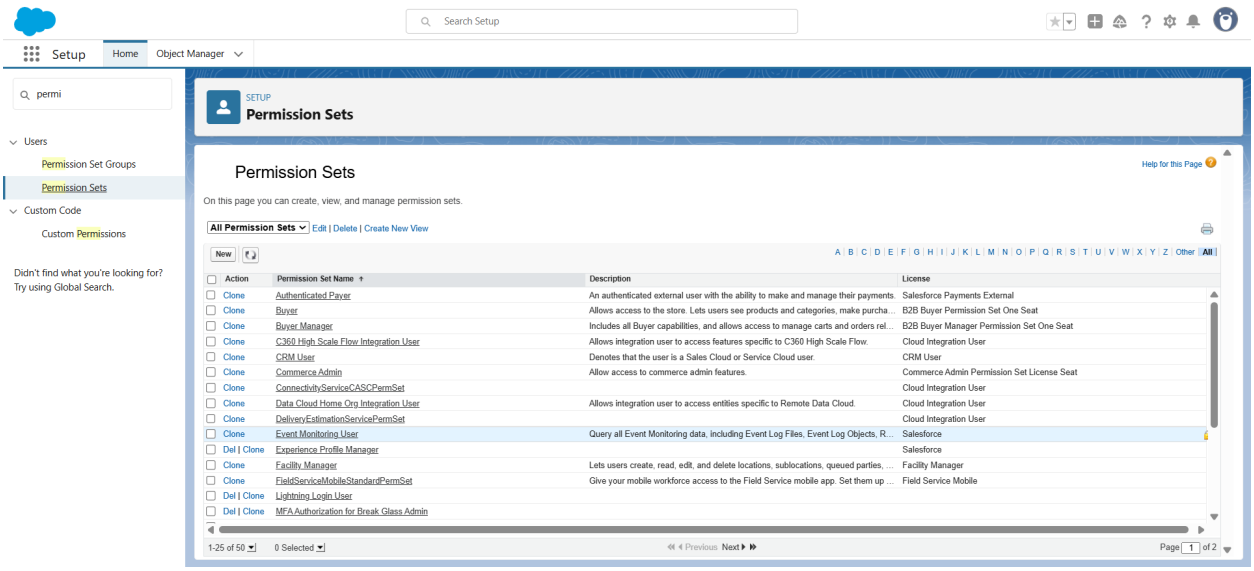
To create additional roles, like **Sales Executive** or **Technician**, again go to **Roles**, click the **+** under the Manager, then click **Add Role**. Fill in the label (e.g., "Sales Executive") and click **Save**. These roles will now be children under the Manager, which means the Manager can access the records owned by these roles. This structure enforces data visibility in a secure, hierarchical manner, while also supporting collaborative work through shared access.

8. Permission Sets and Sharing Rules

Permission Sets are a flexible way to grant additional access to users without changing their existing **profiles**. While profiles control the baseline access like what objects and apps a user can use, permission sets allow administrators to extend permissions to users on top of their profile access. This is especially useful when you want to give temporary or additional privileges to certain users (like allowing Salespersons to access Reports or Dashboards).

To create a Permission Set:

- Go to **Setup → search for Permission Sets**
- Click **New**, give it a label (e.g., "Report Access"), assign an app license
- Then add object, field, or system permissions as needed
- Finally, **assign it to the target users.**



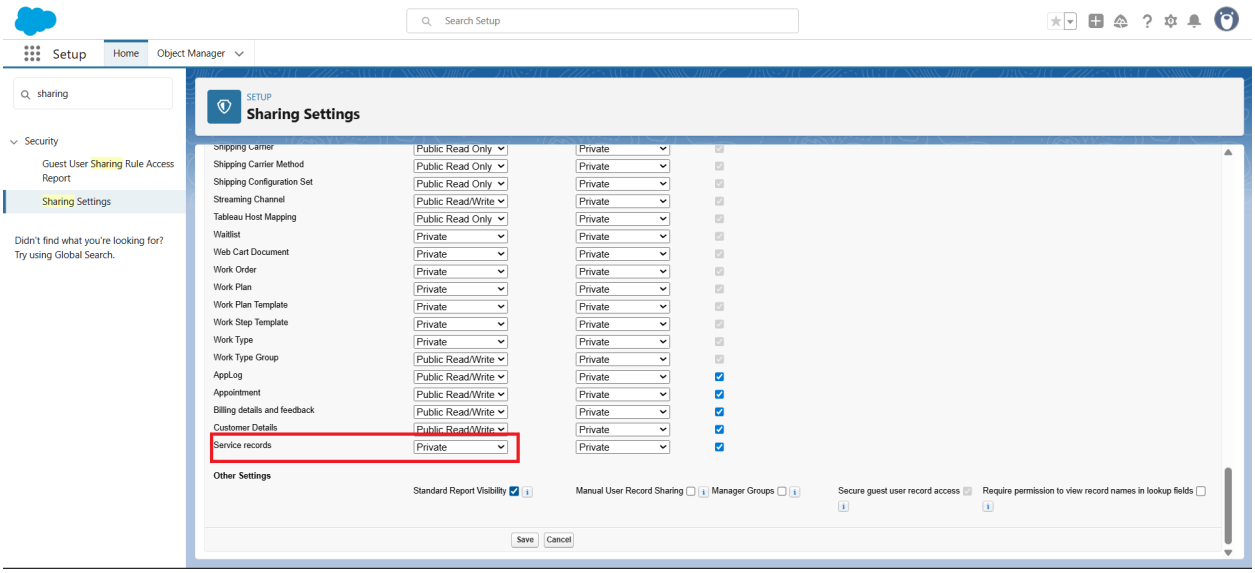
Sharing Rules, on the other hand, define automatic record-level access among users based on roles, public groups, or territories. They are used when **Organization-Wide Defaults (OWD)** are set to private or read-only, and you want to open up access for records collaboration.

For instance, if the **Service Executive** should see all **Customer Feedback records** created by the **Technician** role, a sharing rule can be set up.

To create one:

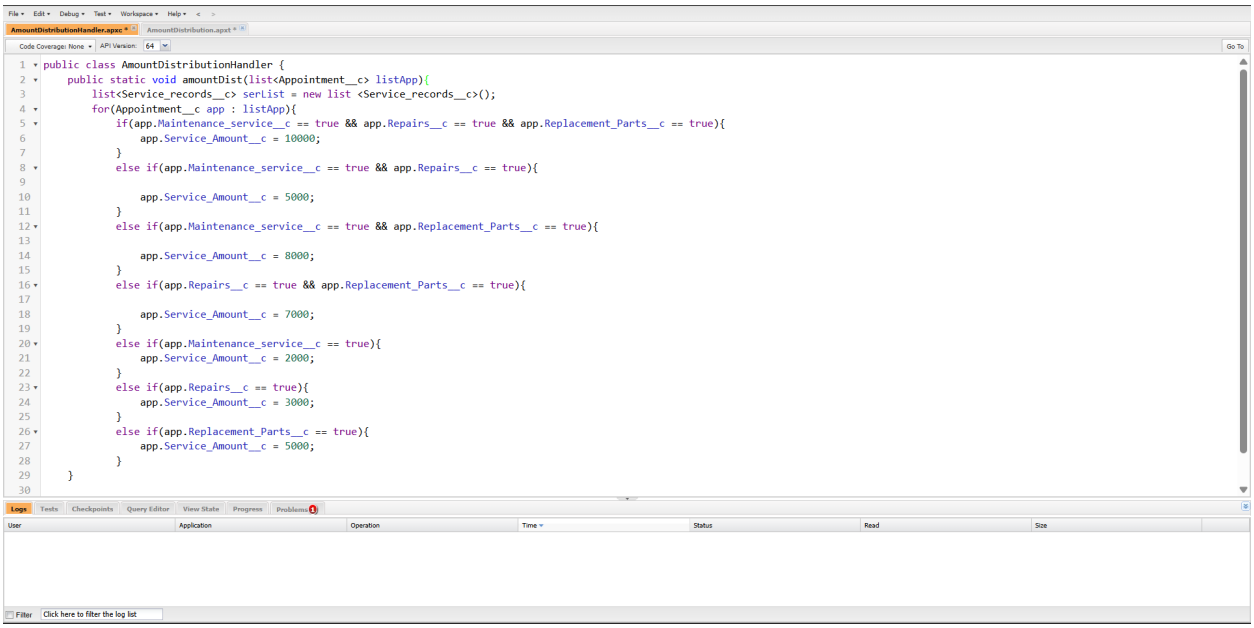
- Go to **Setup**
- Search **"Sharing Settings"**
- Click **New Sharing Rule**
- Define the rule type (e.g., Role → Role)
- Choose the users, and assign the access level (**Read / Read-Write**)

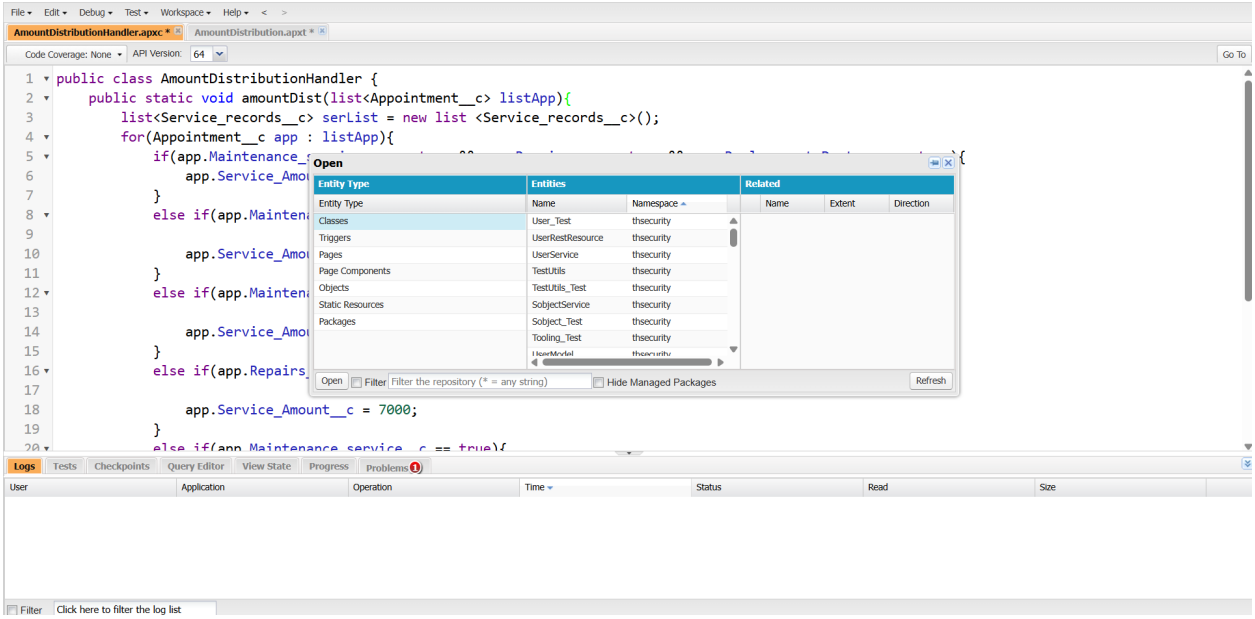
Together, **Permission Sets and Sharing Rules** give Salesforce administrators precise control over both functional access and data visibility, ensuring secure collaborative workflows across the organization.



9. Creation of Test Classes & Test Case Preparation:

As part of the testing phase, it is essential to ensure that all Salesforce functionalities developed during the project are working as expected. To achieve this, we create **Apex test classes** to validate the logic of custom code such as triggers, classes, and flows. Each test class is written to simulate real-life user actions and verify whether the system behaves correctly under those conditions.



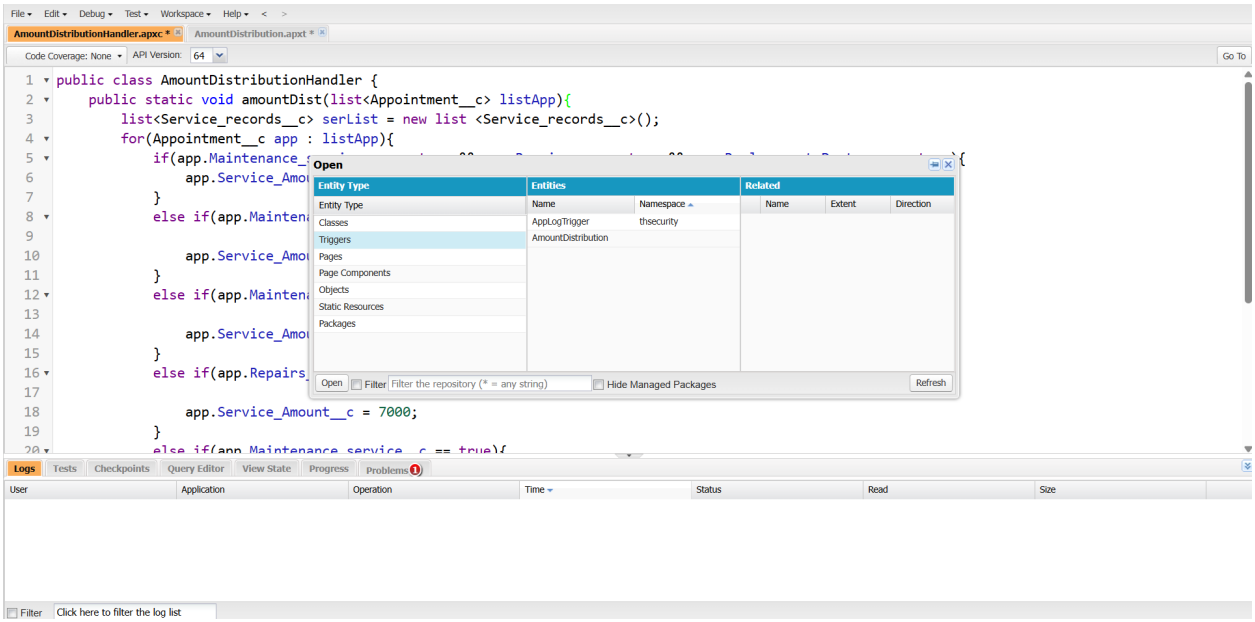


As part of the development process, **test classes were created in Apex** to validate the functional correctness of various Salesforce components including **triggers, approval processes, workflows, and flows**. Each test class was designed to simulate real-world scenarios and ensure that all automation and logic handled the expected input and generated the correct output.

Test cases were written for the following features:

- Booking Creation
- Approval Process Execution
- Automatic Task Creation
- Flow Execution
- Custom Triggers

Each test case includes both **input data simulation** and **assertions** to validate expected outcomes. These test classes help ensure that all business logic performs as expected under different conditions and maintains overall data integrity.



Phase 5: Deployment, Documentation & Maintenance

1. Deployment Strategy

In **Salesforce**, the deployment phase is a critical step where the developed components are moved from the **sandbox (testing)** environment to the **production** (live) environment.

For this project, the deployment was executed using **Change Sets**, which is a standard Salesforce-native tool that ensures safe and structured transfer of metadata components.

Outbound Change Sets were created in the sandbox by including all the necessary

components such as:

- Custom objects
- Fields
- Validation rules
- Workflows
- Approval processes
- Process Builder flows
- Triggers
- Profiles
- Permission sets
- Page layouts

These change sets were uploaded to the production environment, where they were reviewed and deployed after successful validation.

In cases where complex or unsupported metadata (like certain **Apex classes** or large volumes of data) needs to be migrated, tools like **Ant Migration Tool** or **Salesforce CLI with SFDX** can also be considered as alternative deployment strategies in future expansion. **Manual validation** was done post-deployment to ensure every component functions correctly.

2. System Maintenance and Monitoring

After deployment, maintaining system health is essential to ensure smooth business operations. The system will be continuously monitored for performance, error issues, and data integrity. Regular audits will be scheduled to verify field-level changes, security settings (such as profiles and permission sets), and active automation (flows, triggers, workflows).

Monitoring includes tracking login history, API usage, field history tracking, and system logs using tools like **Setup Audit Trail**, **Debug Logs**, and **Health Check**. Salesforce Admins are responsible for ensuring that user permissions align with appropriate team roles evolve and that the system remains compliant with organizational policies.

Periodic backups will be scheduled using **Salesforce Data Export** features or third-party tools. Feedback loops are enabled through reports and dashboards to track usage and identify potential bottlenecks in processes like record creation, approvals, or

assignments.

For every Salesforce feature implemented — such as booking creation, approval processes, automation flows, validation rules, triggers — detailed documentation is maintained. Each feature is accompanied by test cases, expected inputs and outputs, and screenshots showing system behavior before and after execution. This makes it easier for any admin or developer to understand and debug the system in the future.

Conclusion:

This Garage Management System project has been created with Salesforce to simplify and automate major operations involved with a vehicle service center – important areas like service records, billing, feedback, and customer management. It adds better system quality, improved usability, and real-time interaction between employees, technicians, and clients.

Across the development life cycle, we utilized several Salesforce features, including custom objects, page layouts, validation rules, automation tools (such as Flows and Workflow Rules), approval processes, and reports to make a complete working and user-friendly application. Profiles and roles were planned to provide data-level security and role-based access. Permission Sets and Sharing Rules also added flexibility in visibility of records.

Admin and developer work was well-documented and screenshots of each configuration step – from creating custom fields to automation flow setups – were taken and included for clarity. Every feature in the Booking System, Billing integration, Feedback module or Tasks Automation was supported by test classes and real-time validation through sample inputs and expected outputs.

Deployment was done using Change Sets to migrate metadata from sandbox to production in a safe and controlled manner. Troubleshooting methodology, error management, and post-deployment verification have also been documented for maintenance purposes.

Testing Approach

Manual as well as automated testing was performed for each component. Test cases were coded and run for flows, approval processes, and triggers to provide functionality, data validity, and user experience. Visualforce Pages and Apex Test Classes confirmed back-end logic and ensured code coverage.

Future Enhancements

Future enhancements to further improve the system can be:

- Chatbot integration for customer support
- AI-driven service recommendations
- Real-time SMS/WhatsApp notifications
- Mobile App Integration
- Technician performance analytics and dashboards