

**Garage Management system**

College: 7155 - PSG Institute of Technology and Applied Research

Team ID: NM2024TMID00698

Team Lead: ABIRAMI S (D56DE6BC50088979B7F49E21B88C33A1)

Team Members:

| Amirthaa B | 2CD510412827F3B8A164BC37D2A586AF |
| --- | --- |
| Koushika Purna Velamati | EBF0F1A978570B282184BFF889BD2223 |
| Shruthi R A | 3E6C591A76F4F4B88844A3E322F44439 |
| Abirami S | D56DE6BC50088979B7F49E21B88C33A1 |

**1.Project Overview**

The Garage Management System is a valuable tool for automotive repair facilities, helping them deliver top-notch service, increase operational efficiency, and build lasting customer relationships. With its user-friendly interface and powerful features, GMS empowers garages to thrive in a competitive market while ensuring a seamless and satisfying experience for both customers and staff.

**2.Objectives**

**Business Goals:**The primary business goals of implementing a Garage Management System on Salesforce are to enhance operational efficiency, improve customer satisfaction, and drive revenue growth for automotive service centers. By automating key workflows such as appointment scheduling, service tracking, and customer communication, the system aims to reduce manual errors and streamline day-to-day operations. With a centralized platform for managing customer data and vehicle histories, service centers can deliver personalized experiences and build lasting customer relationships.

**Specific Outcomes:**

* Streamlined scheduling and communication increase customer satisfaction by reducing wait times and keeping clients informed.
* Automated workflows cut administrative time by 20-30%, letting staff focus on core tasks and reduce errors.
* Real-time tracking reduces overstock and shortages by 25%, ensuring timely access to parts.
* Easy access to data boosts productivity and job satisfaction, reducing turnover.

**3.Salesforce Key Features and Concepts Utilized**

This project leverages key Salesforce functionalities and concepts to create a seamless and effective garage management system:

1. Objects
   * Created custom objects like Customer Details,Appointment,Service Records,Billing details and Feedback in order to organize and track all essential data.
   * Defined lookup relationships between objects to establish clear connections, ensuring data consistency and easy access.
2. Tabs and Lightning App
   * Set up tabs for each object for easy navigation within the Garage Management Application.
   * Customized the Lightning app for streamlined access, branding, and organization.
3. Fields
   * Create fields lookup,checkbox,date,currency,text,picklist,formula for customer details object.
4. Validation Rule and Duplicate Rule
   * Create validation rule for objects like appointment, service records,billing details and feedback
   * Create a matching rule to customer details object and a duplicate rule to customer details object.
5. Profiles and Role Hierarchy
   * Creation of manager profile and Sales person profile separately to define what a user can do in salesforce
   * Create a Manager role and a salesperson role as well.
6. Users and Public Groups
   * Create various users as customers with their role,User licence,Profile.
   * Create a public group named sales team and add the available members.
7. Flows and Apex Triggers
   * Create a Record-triggered flow with the object Billing details and feedback and the other appropriate details.
   * Created an apex handler for Amount Distribution for each Service the customer selected for their Vehicle.
8. Reports
   * Designed custom report types and reports to provide insights into Rating for Service,Payment Status.
9. Dashboards
   * Created a dashboard integrated with the homepage to offer real-time visibility into project metrics and progress.

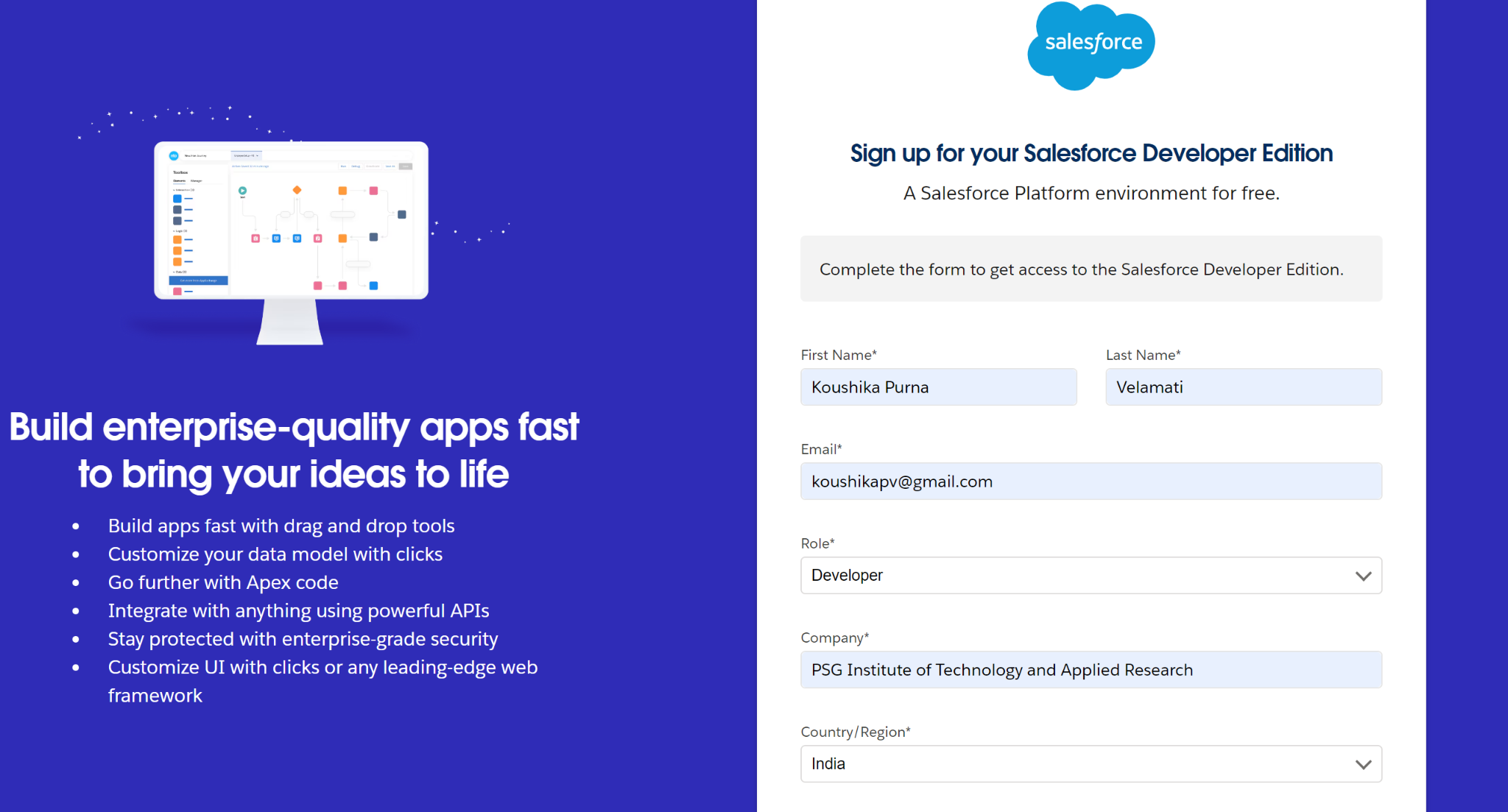
These Salesforce features collectively ensure that the project operates with high efficiency, transparency, and data-driven decision-making for the garage management system.

**4.Detailed Steps to Solution Design**

### Salesforce

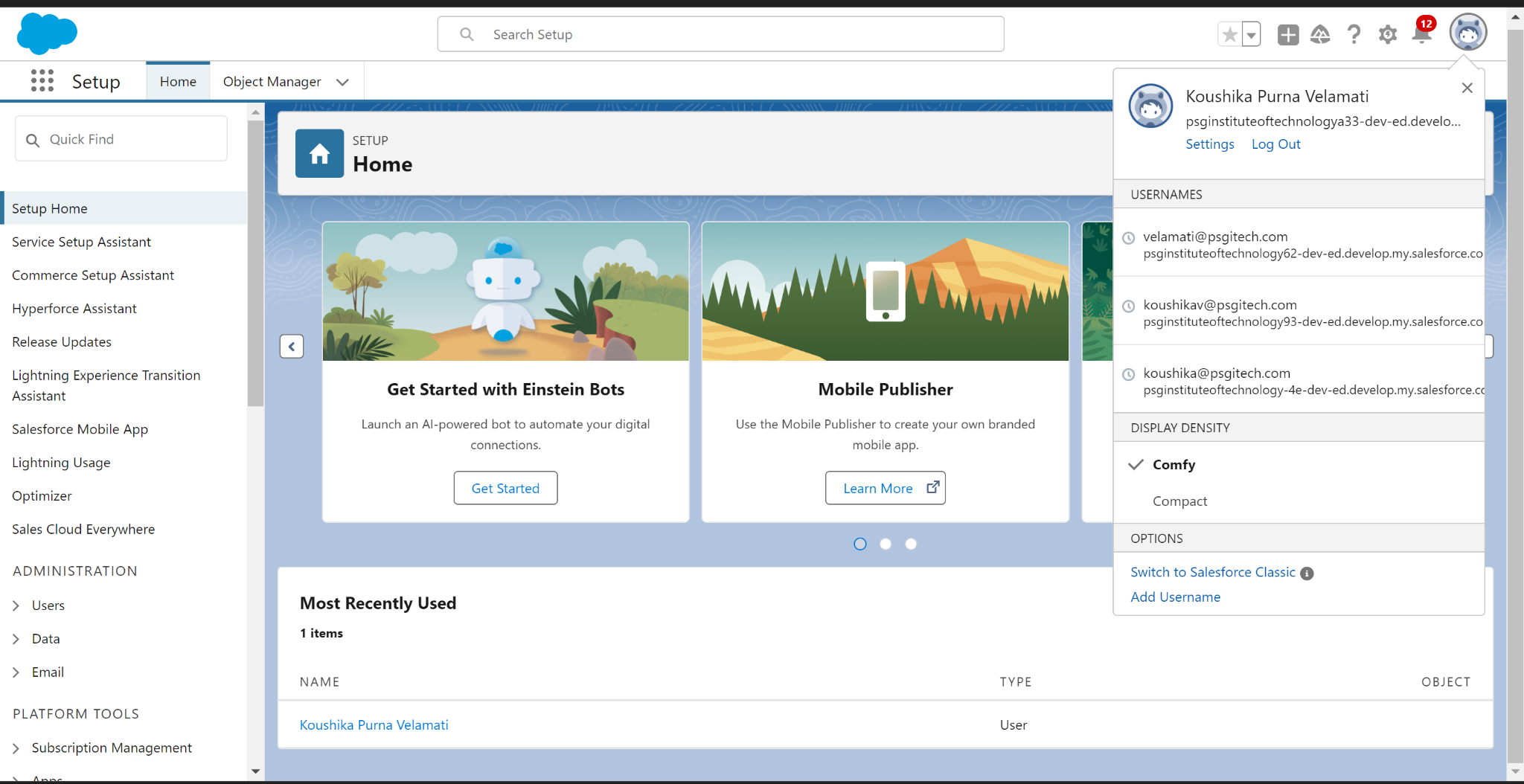
### 1. Created Salesforce Developer Account

We began by creating a Salesforce Developer Account to access a development environment where we could build and customize the application. This setup provided the necessary tools to design, test, and deploy the application.

****

**2. Activated Salesforce Account**

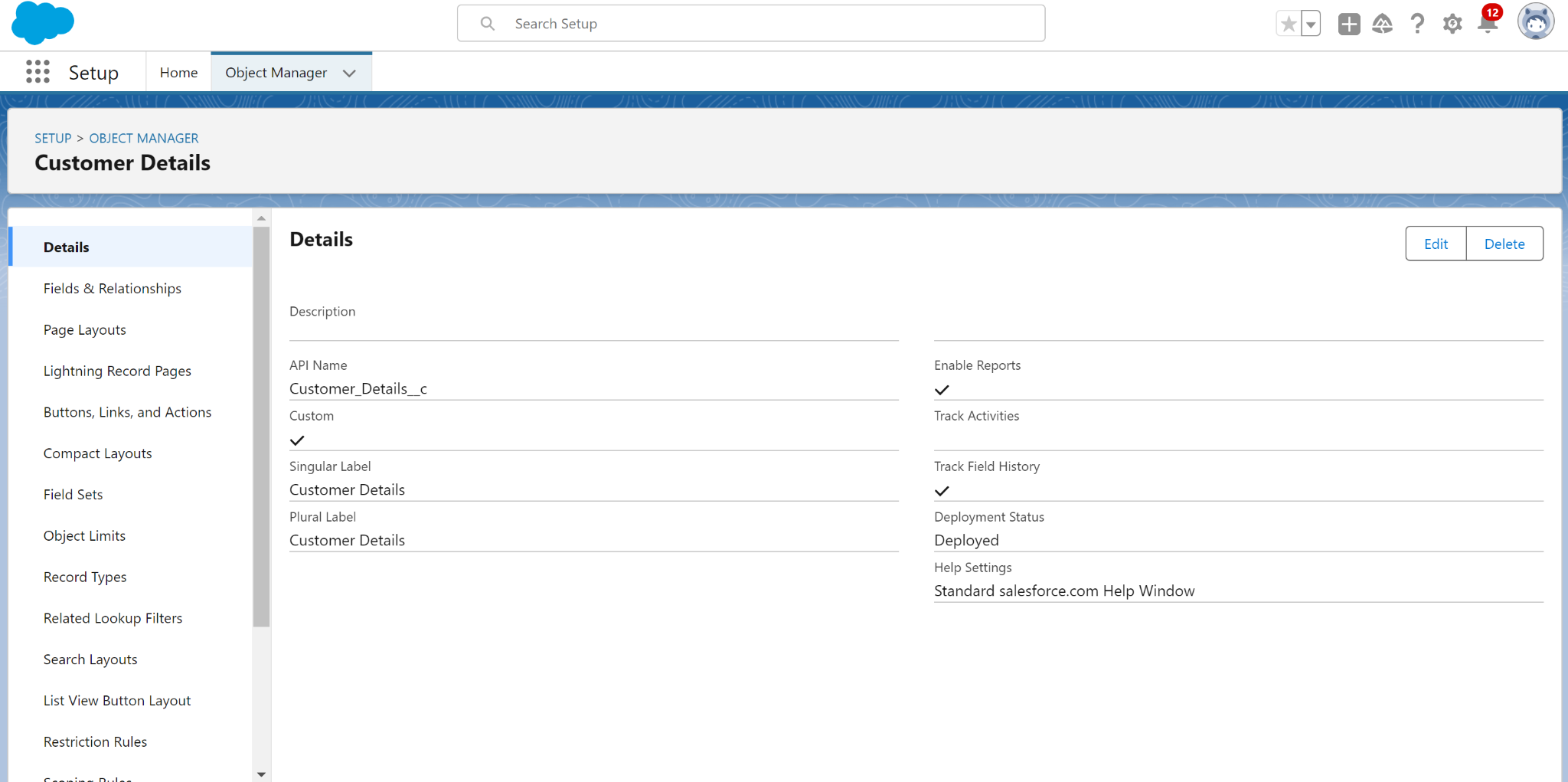
We accessed the email inbox used during signup to locate the account verification email, which typically arrives within 5-10 minutes. After clicking on "Verify Account," we set a password, answered a security question, and clicked "Change Password." This completed the activation process, redirecting us to the Salesforce setup page.

****

### Object

**3. Created Customer Details Object**We created a custom object for customer information by navigating to the Object Manager from the setup page and selecting "Create" > "Custom Object." We entered the following details:

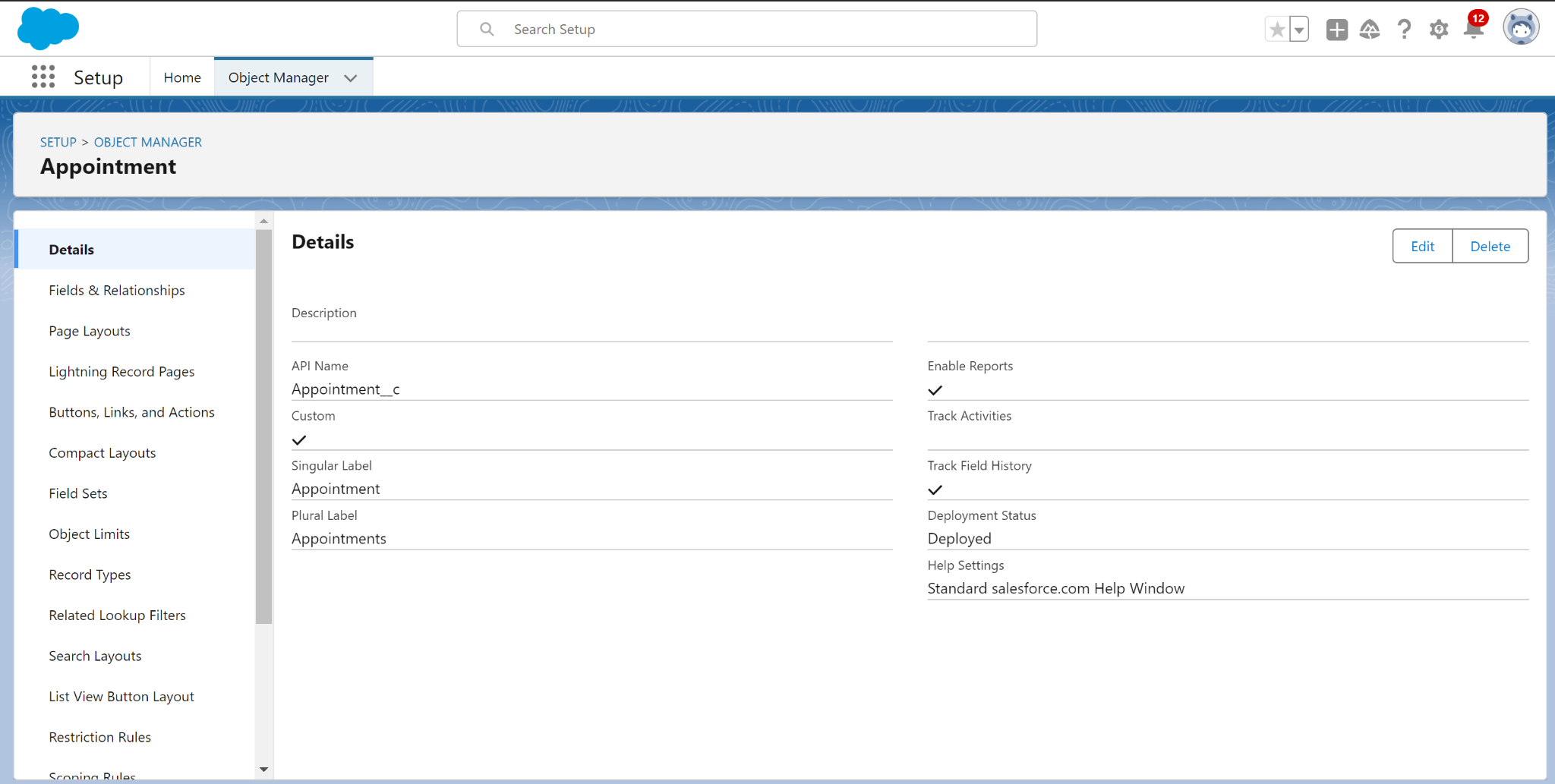
* **Label Name**: Customer Details
* **Plural Label Name**: Customer Details
* **Record Name Label**: Customer Name
* **Data Type**: Text

Additionally, we enabled "Allow reports," "Track Field History," and "Allow search," then saved the configuration.****

**4. Created Appointment Object**We created an object to manage appointment records by navigating to the Object Manager, selecting "Create" > "Custom Object," and entering the following details:

* **Label Name**: Appointment
* **Plural Label Name**: Appointments
* **Record Name Label**: Appointment Name
* **Data Type**: Auto Number
* **Display Format**: app-{000}
* **Starting Number**: 1

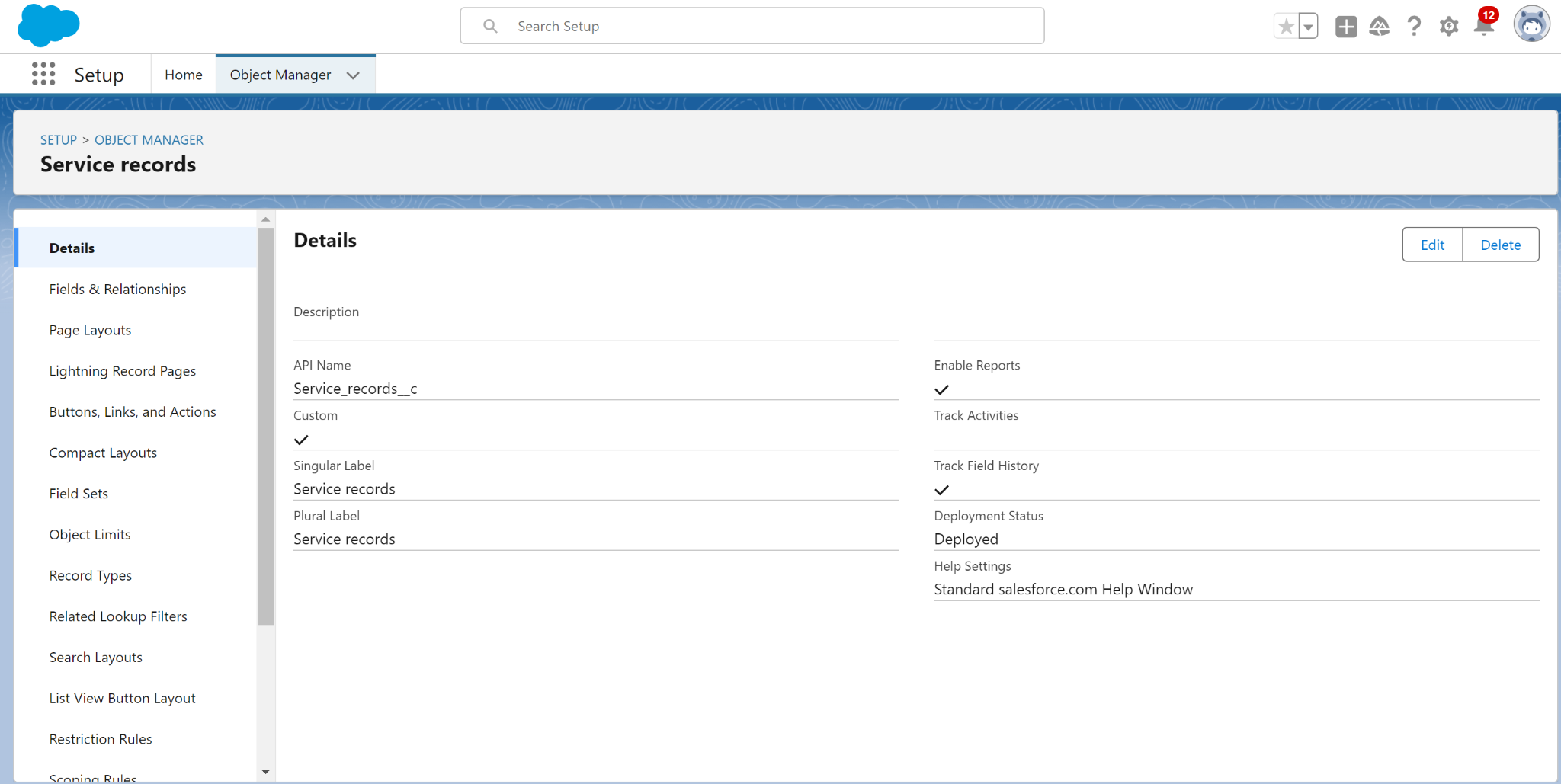
We enabled "Allow reports," "Track Field History," and "Allow search," then saved the object.

****

**5. Created Service Records Object**We created an object to store service records by navigating to the Object Manager, selecting "Create" > "Custom Object," and entering the following details:

* **Label Name**: Service records
* **Plural Label Name**: Service records
* **Record Name Label**: Service records Name
* **Data Type**: Auto Number
* **Display Format**: ser-{000}
* **Starting Number**: 1

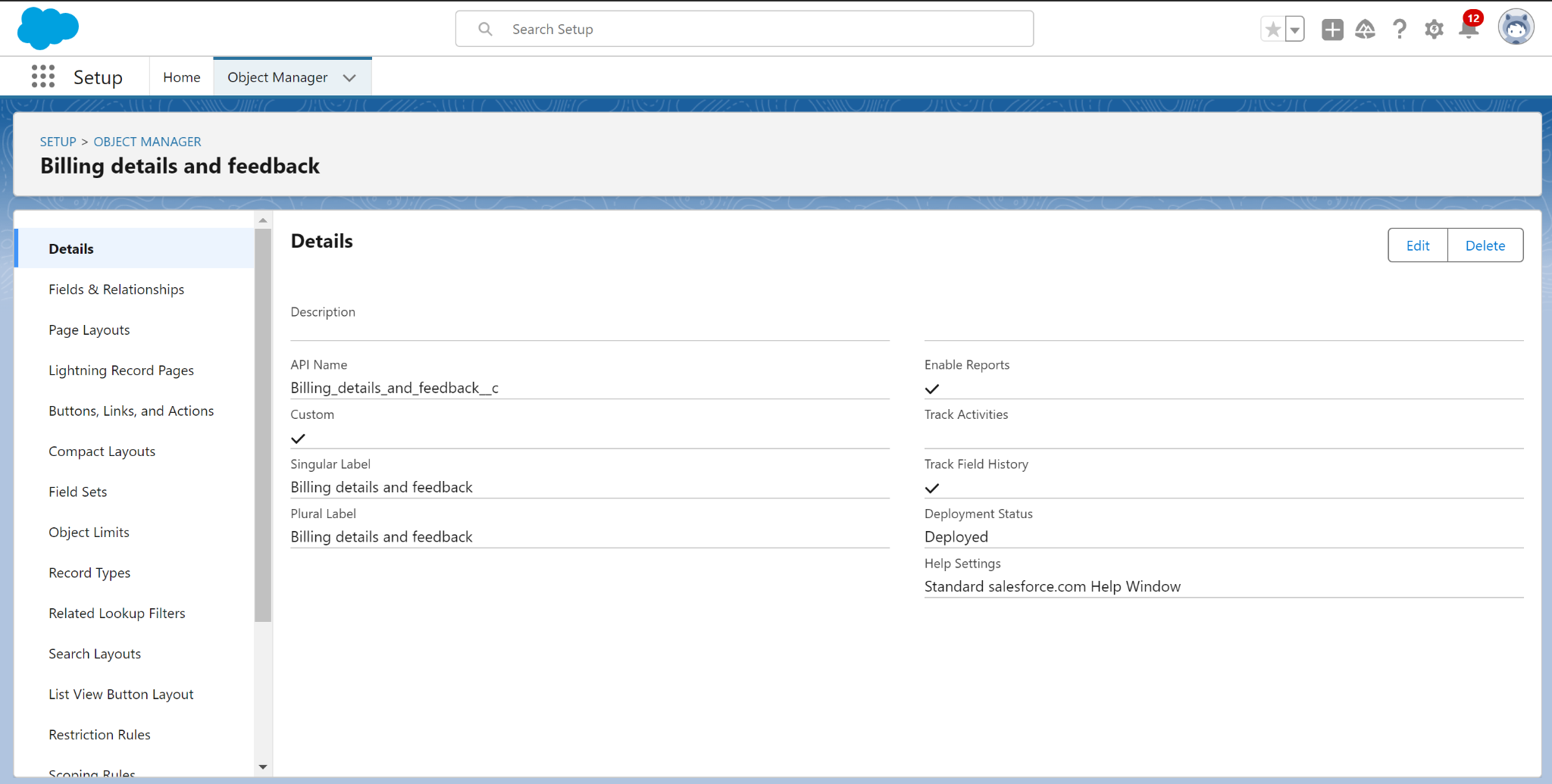
We enabled "Allow reports," "Track Field History," and "Allow search," then saved the object.

****

**6. Created Billing Details and Feedback Object**We set up an object to manage billing details and feedback by navigating to the Object Manager, selecting "Create" > "Custom Object," and entering the following details:

* **Label Name**: Billing details and feedback
* **Plural Label Name**: Billing details and feedback
* **Record Name Label**: Billing details and feedback Name
* **Data Type**: Auto Number
* **Display Format**: bill-{000}
* **Starting Number**: 1

We enabled "Allow reports," "Track Field History," and "Allow search," then saved the object.

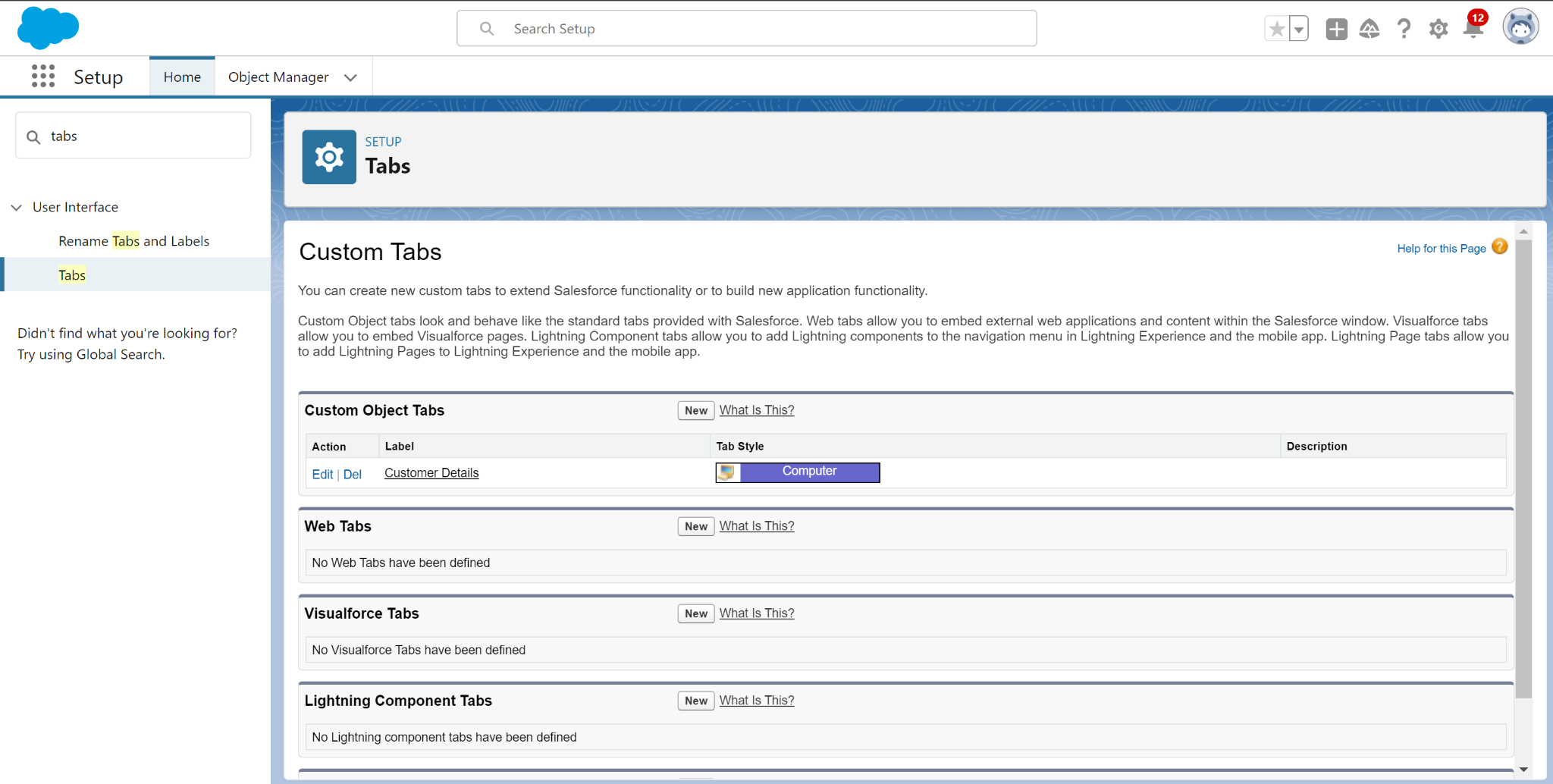
****

### Tabs

**7. Created a Custom Tab for Customer Details**We created a custom tab for the Customer Details object by navigating to the setup page, typing "Tabs" in the Quick Find bar, and clicking on "New" under Custom Object Tabs. We then followed these steps:

* **Select Object**: Customer Details
* **Select Tab Style**
* **Add to Profiles Page**: Kept as default
* **Add to Custom App**: Unchecked "Include Tab"

We ensured "Append tab to users' existing personal customizations" was checked, then saved the tab.

****

**8. Created Custom Tabs for Remaining Objects**We created custom tabs for the remaining objects—Appointments, Service records, and Billing details and feedback—by following the same steps as in the previous tab creation:

* Navigated to the setup page, typed "Tabs" in the Quick Find bar, and selected "New" under Custom Object Tabs.
* Selected each object (Appointments, Service records, Billing details and feedback) and chose an appropriate tab style.
* Kept "Add to Profiles Page" as default and unchecked "Include Tab" under "Add to Custom App."
* Ensured "Append tab to users' existing personal customizations" was checked.

Each tab was saved accordingly.

****

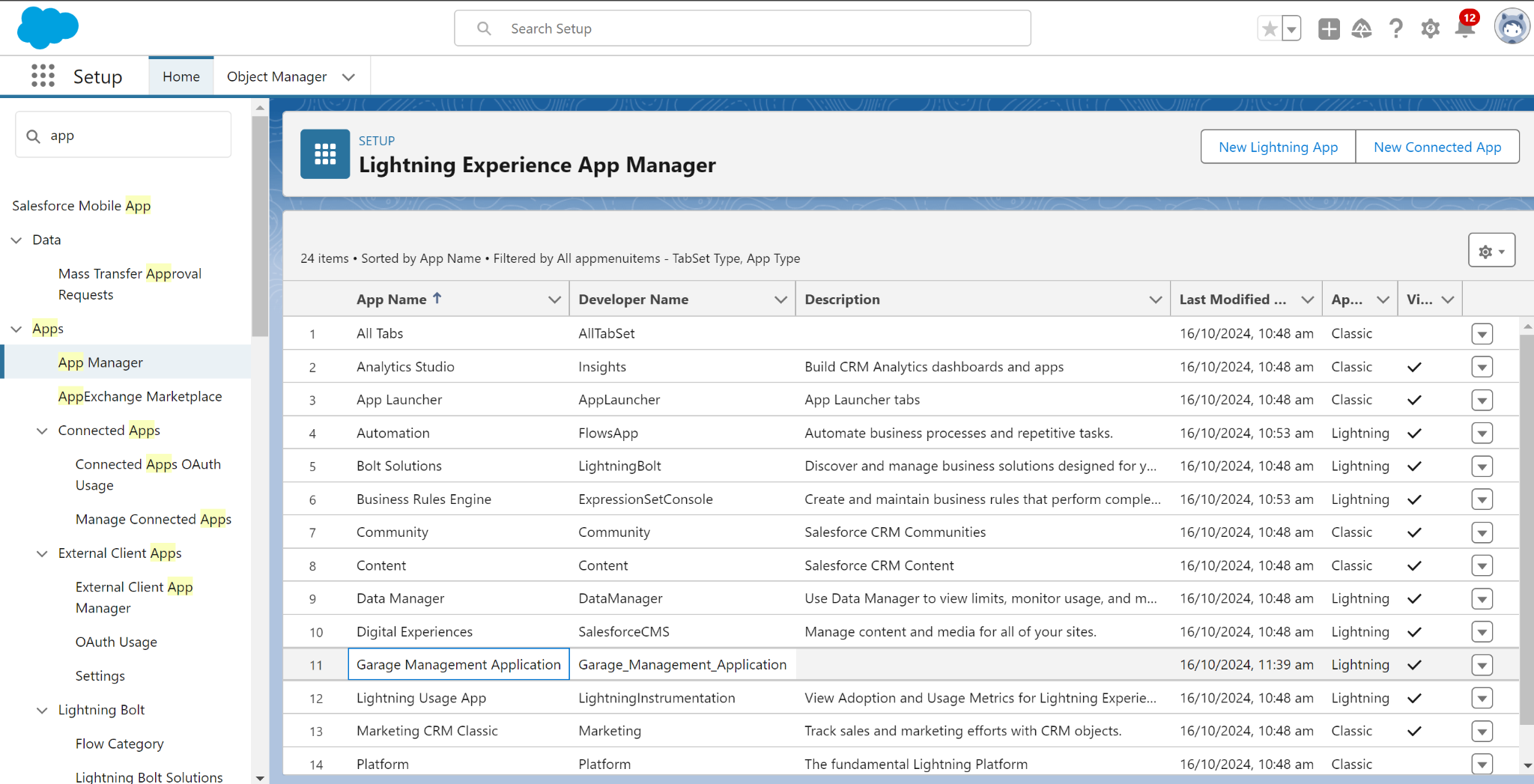
### The Lightning App

**9. Created Garage Management Lightning App**We created a Lightning App named "Garage Management Application" by navigating to the "App Manager" in the setup page and clicking on "New Lightning App." The following steps were followed:

* **App Name**: Garage Management Application
* **App Options**: Kept as default
* **Utility Items**: Kept as default

For navigation items, we added **Customer Details, Appointments, Service records, Billing details and feedback, Reports,** and **Dashboards** by selecting them from the search bar and moving them using the arrow button.

Finally, we added the **System Administrator** profile by searching for it, moving it with the arrow button, and clicking "Save & Finish."

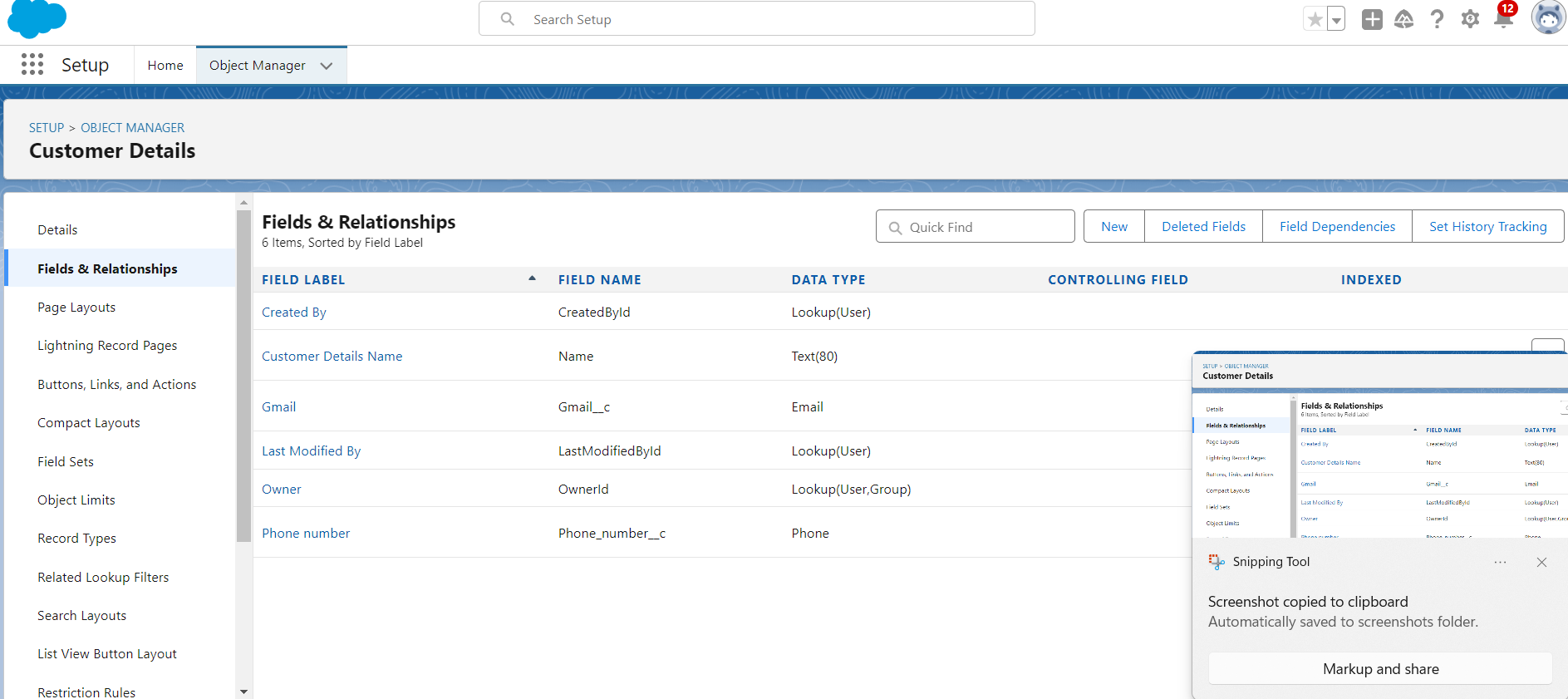
****

**10. Created Fields for Customer Details Object**We added fields to the Customer Details object to store phone and email information. The steps were as follows:

* Navigated to the setup page, selected Object Manager, searched for "Customer Details," and selected it.
* Chose **Fields & Relationships** > **New**.

For each field:

1. **Phone Number Field**:
   * **Data Type**: Phone
   * **Field Label**: Phone number
   * **Field Name**: Auto-generated
   * Clicked **Next**, **Next**, then **Save and New** to continue.
2. **Email Field**:
   * **Data Type**: Email
   * **Field Label**: Gmail
   * **Field Name**: Auto-generated
   * Clicked **Next**, **Next**, then saved the field.

****

**11. Created Lookup Fields for Related Objects**

**1. Lookup Field on Appointment Object**

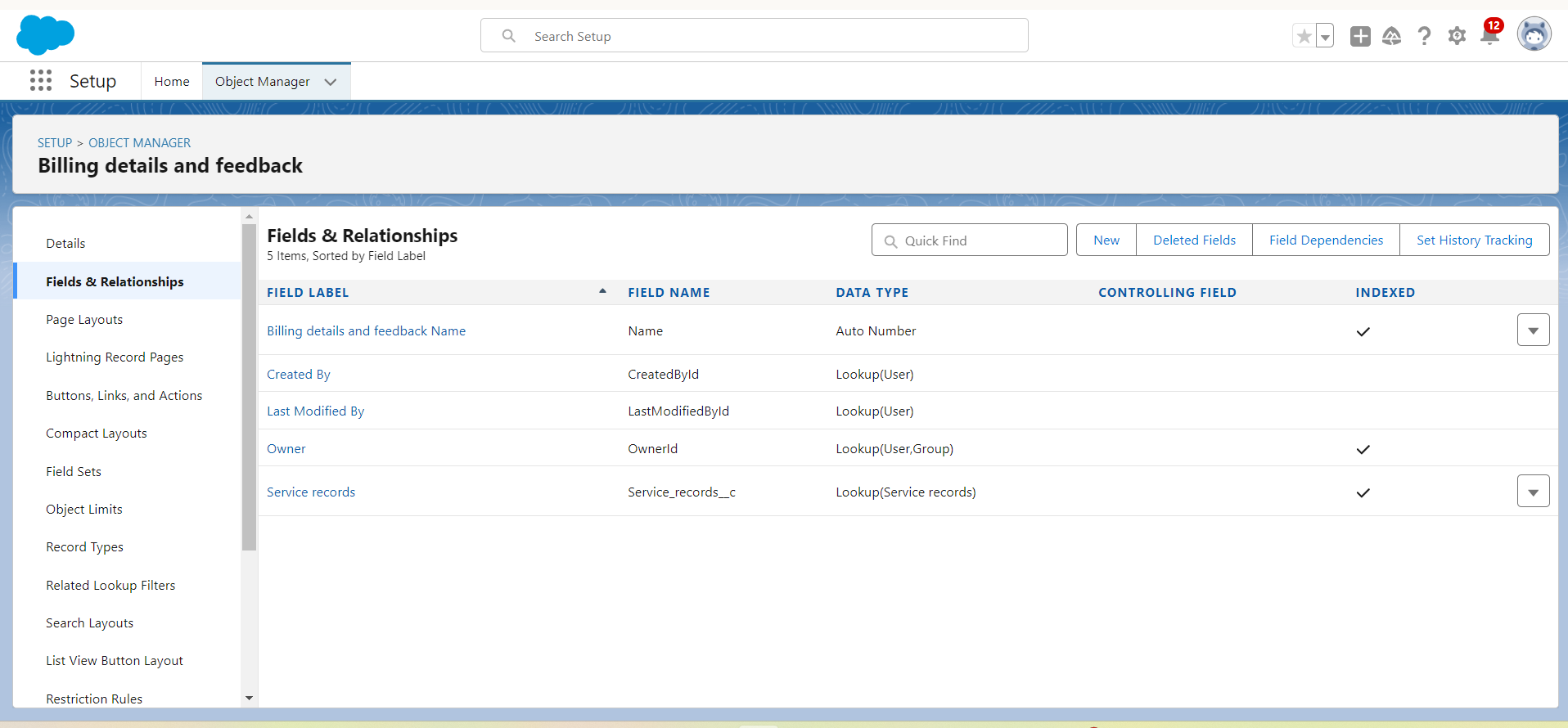
* From the setup page, searched for "Appointment" in Object Manager, then went to **Fields & Relationships** > **New**.
* Chose **Lookup Relationship**, selected **Customer Details**, and clicked **Next** > **Save**.

**2. Lookup Field on Service Records Object**

* Searched for "Service records" in Object Manager, then went to **Fields & Relationships** > **New**.
* Chose **Lookup Relationship**, selected **Appointment**, marked as **Required**, and added a **Lookup Filter** (Appointment Date < Created Date).
* Clicked **Next** > **Save**.

**3. Lookup Field on Billing Details and Feedback Object**

* Searched for "Billing details and feedback" in Object Manager, then went to **Fields & Relationships** > **New**.
* Chose **Lookup Relationship**, selected **Service records**, and clicked **Next** > **Save & New**.

****

**12. Created Checkbox Fields**

**1. Checkbox Field on Appointment Object**

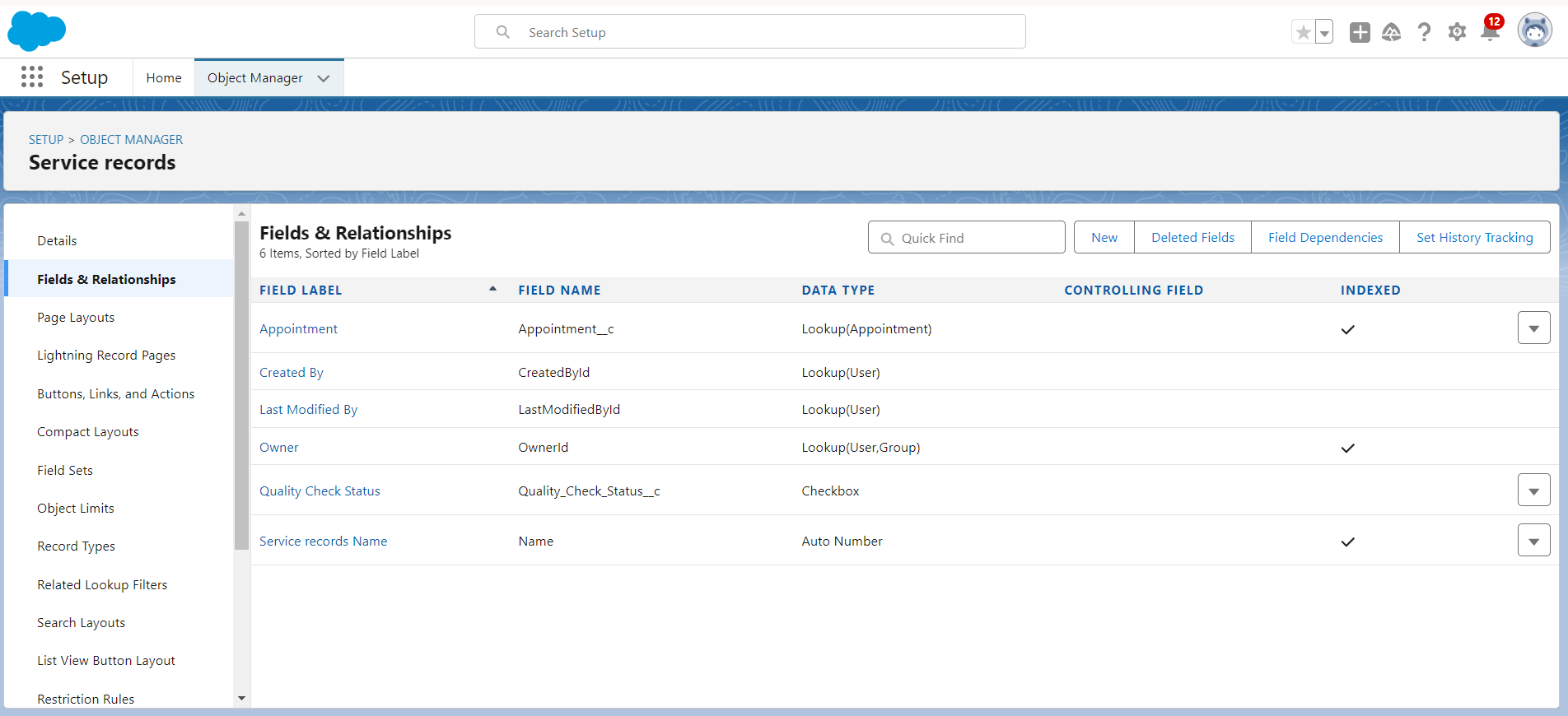
* From the setup page, searched for "Appointment" in Object Manager, then went to **Fields & Relationships** > **New**.
* Chose **Checkbox** as the data type, then:
  + **Field Label**: Maintenance service
  + **Field Name**: Auto-populated
  + **Default Value**: Unchecked
* Clicked **Next** > **Save**.

**2. Additional Checkbox Fields on Appointment Object**

* Repeated the steps above to create the following fields:
  + **Field Label**: Repairs, **Default Value**: Unchecked
  + **Field Label**: Replacement Parts, **Default Value**: Unchecked
* Clicked **Next** > **Save** for each.

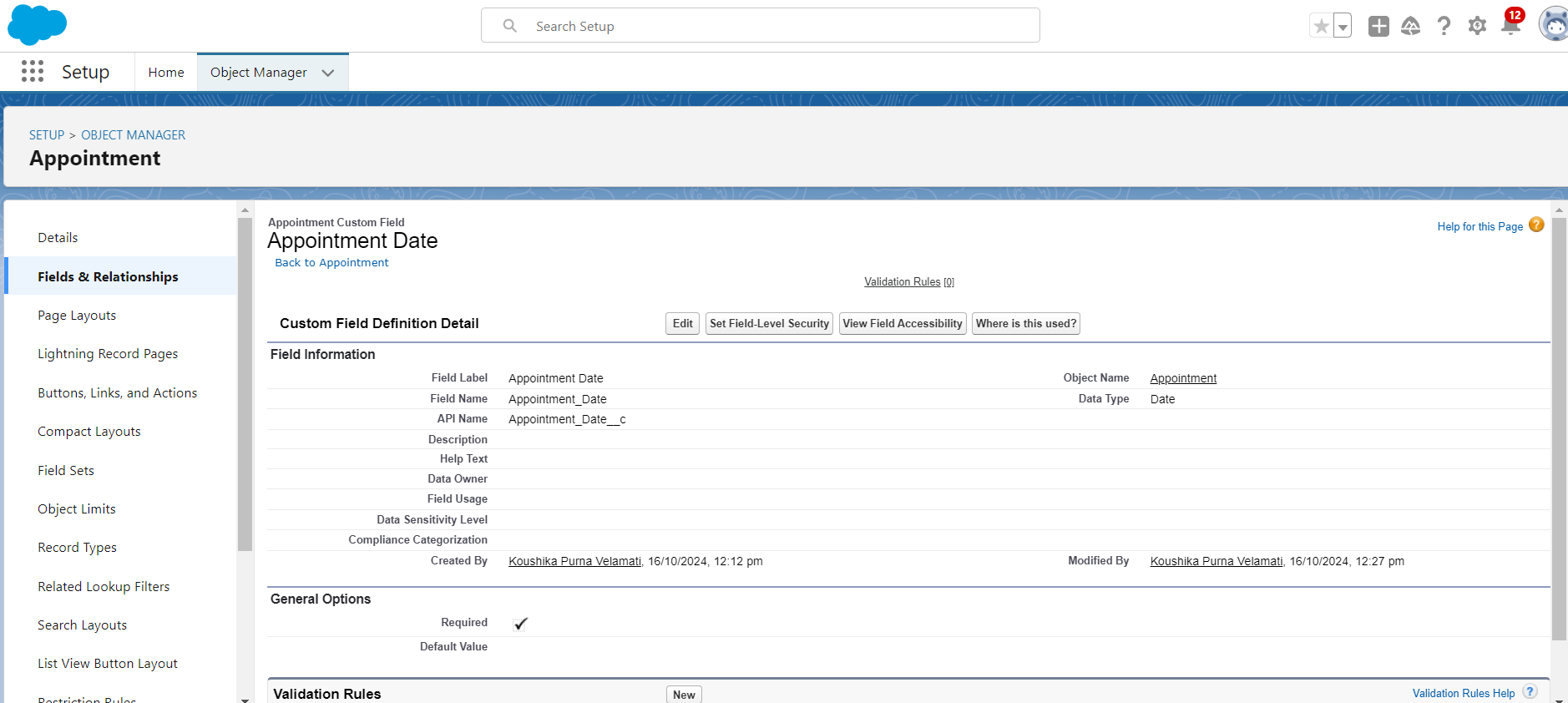
**3. Checkbox Field on Service Records Object**

* Searched for "Service records" in Object Manager, then went to **Fields & Relationships** > **New**.
* Chose **Checkbox** as the data type, then:
  + **Field Label**: Quality Check Status
  + **Field Name**: Auto-populated
  + **Default Value**: Unchecked
* Clicked **Next** > **Save**.

****

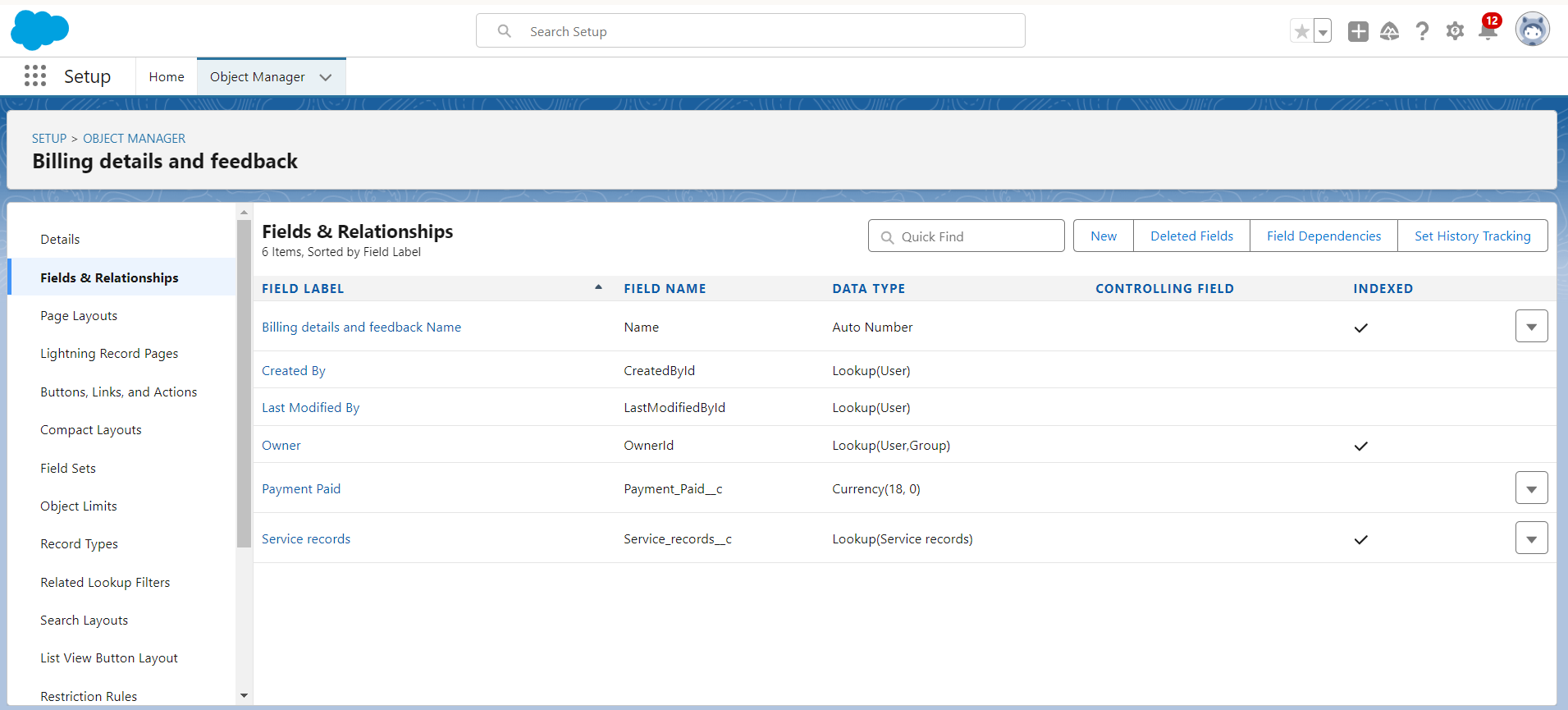
**13. Created Date Field on Appointment Object**

* From the setup page, searched for "Appointment" in Object Manager, then went to **Fields & Relationships** > **New**.
* Selected **Date** as the data type, then:
  + **Field Label**: Appointment Date
  + **Field Name**: Auto-populated
  + Marked it as **Required**.
* Clicked **Next** > **Save**.

****

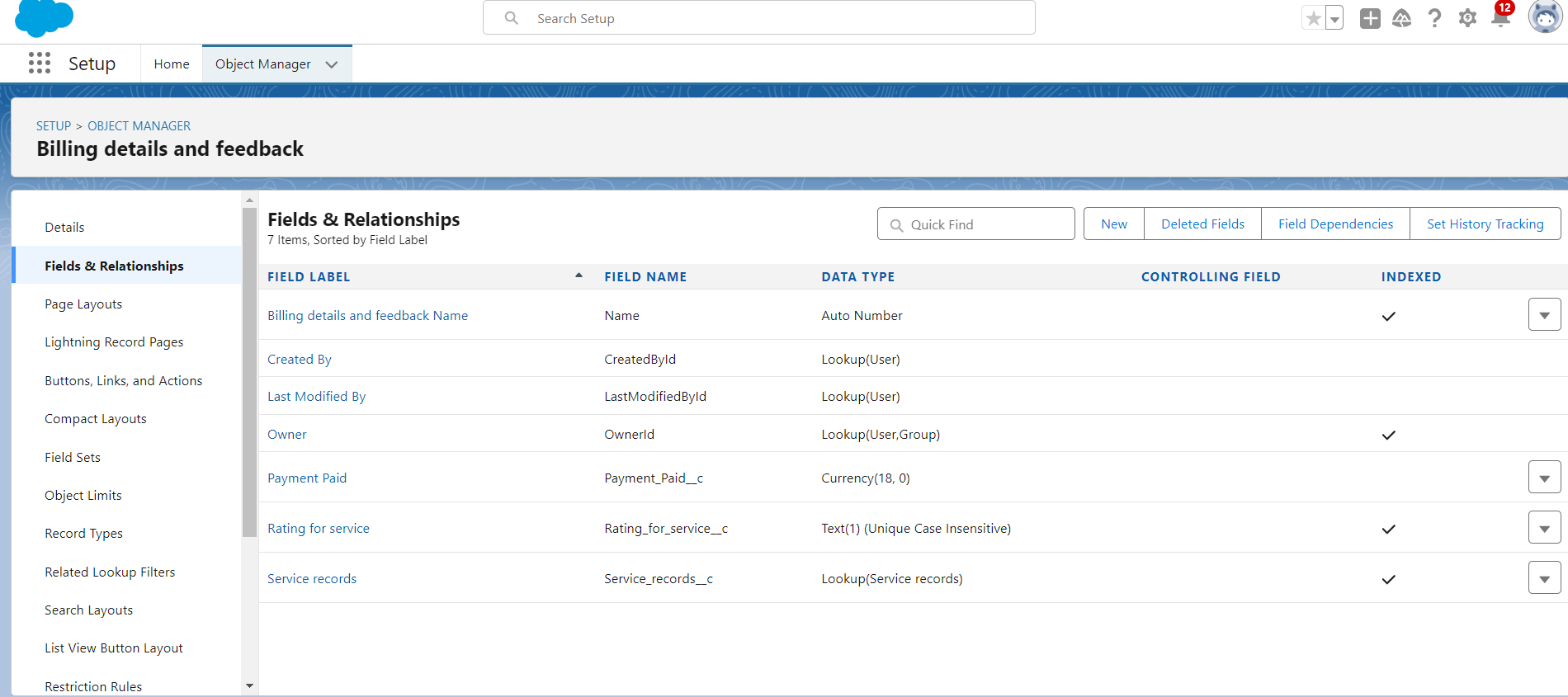
**14. Created Currency Fields**

We created currency fields on the **Appointment** and **Billing details and feedback** objects. For the **Appointment** object, we selected **Currency** as the data type, labeled the field **Service Amount**, and set it as **Read-Only** for all profiles in field-level security. After clicking **Next** and **Save**, we repeated the same process for the **Billing details and feedback** object, labeling the field **Payment Paid**.

****

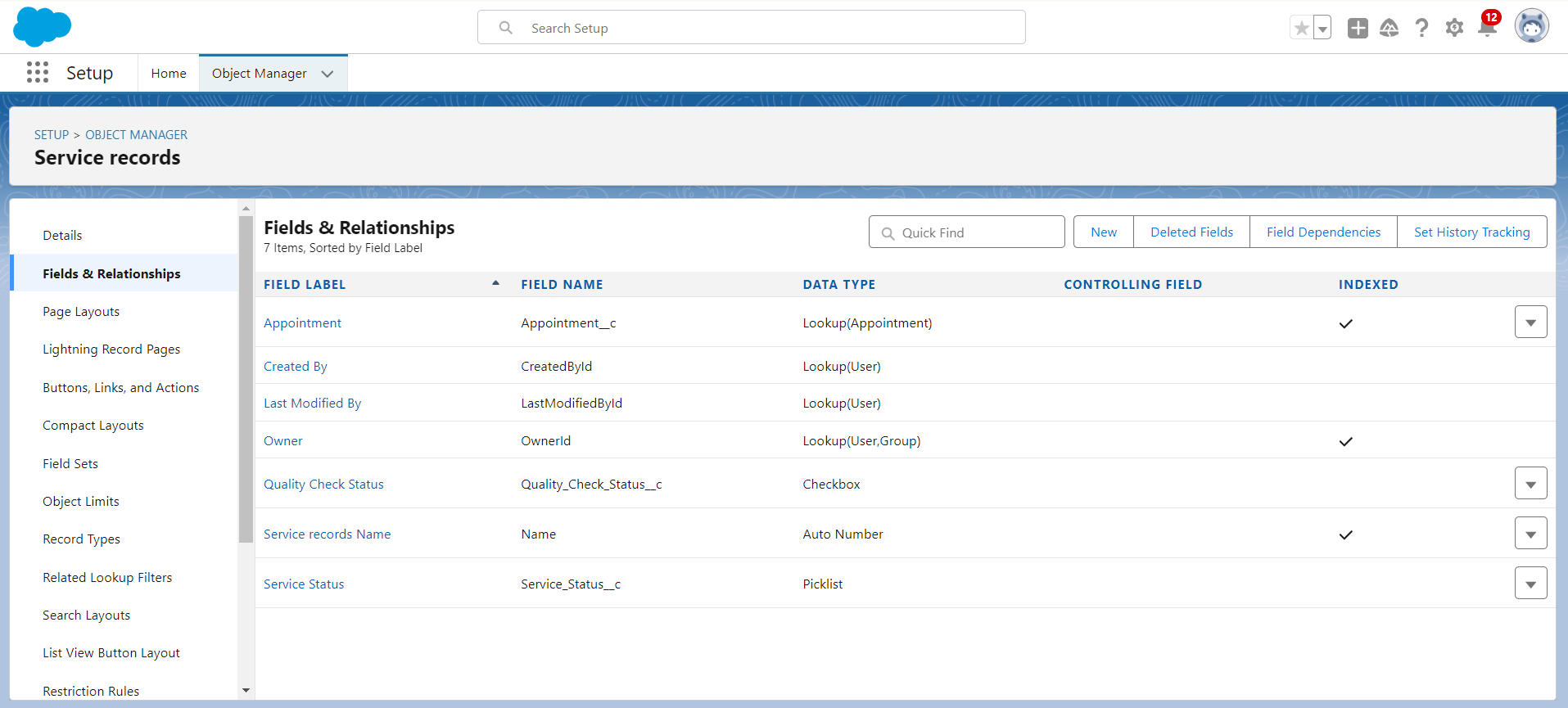
**15. Created Text Fields**

We created **Text fields** on the **Appointment** and **Billing details and feedback** objects. For the **Appointment** object, we selected **Text** as the data type, labeled the field **Vehicle number plate**, set the length to 10, and marked it as **Required** and **Unique**. After clicking **Next** > **Save**, we repeated the process for the **Billing details and feedback** object, labeling the field **Rating for service**, setting the length to 1, and marking it **Required** and **Unique**.

****

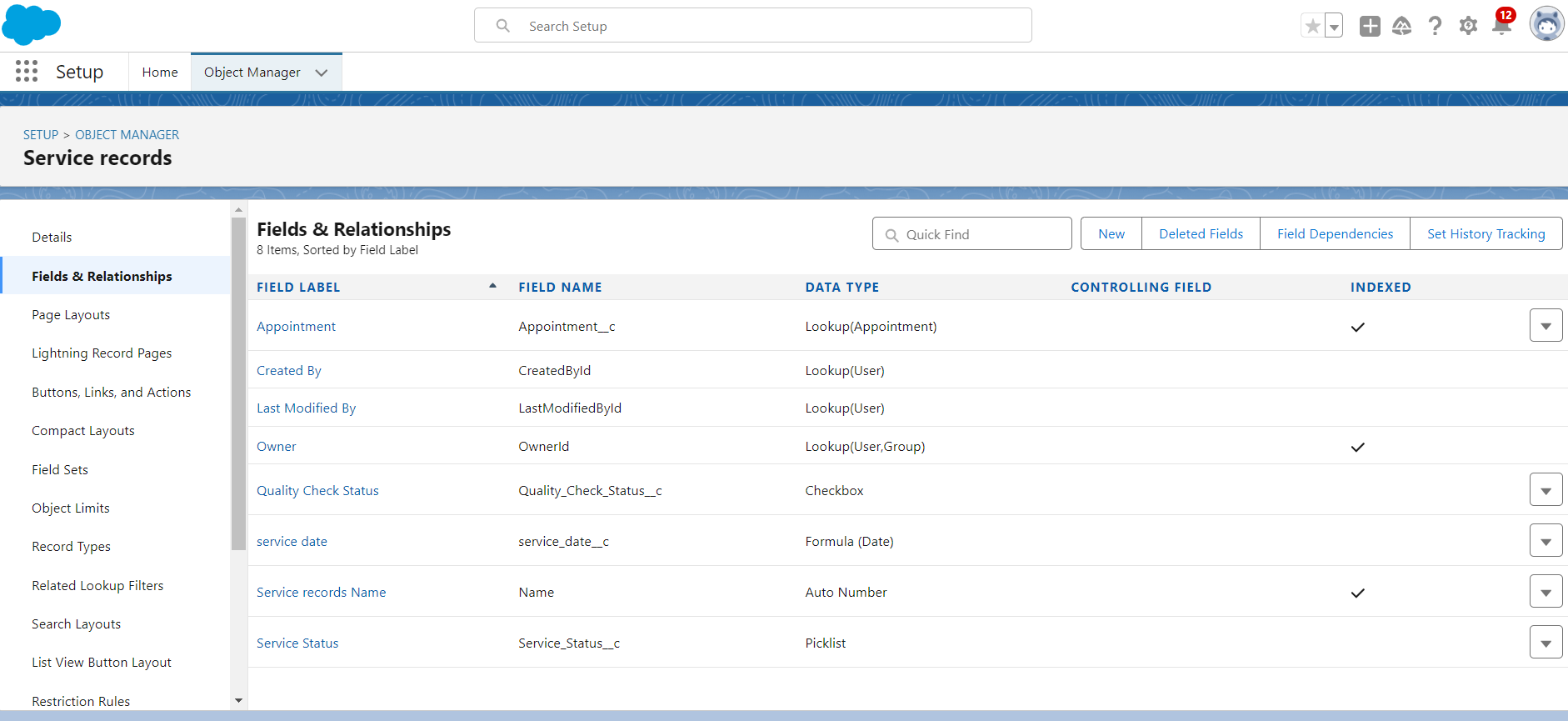
**16. Created Picklist Fields**

We created **Picklist fields** on the **Service records** and **Billing details and feedback** objects. For the **Service records** object, we selected **Picklist** as the data type, labeled the field **Service Status**, and added the values "Started" and "Completed." After clicking **Next** > **Save**, we repeated the process for the **Billing details and feedback** object, labeling the field **Payment Status** and adding the values "Pending" and "Completed."

****

**17. Created Formula Field in Service Records Object**

We created a **Formula field** in the **Service records** object by selecting **Formula** as the data type. The **Field Label** and **Field Name** were set to **Service Date**, and the **Formula Return Type** was chosen as **Date**. The formula used was **CreatedDate**. After clicking **Check Syntax**, we proceeded with **Next** > **Save**.

****

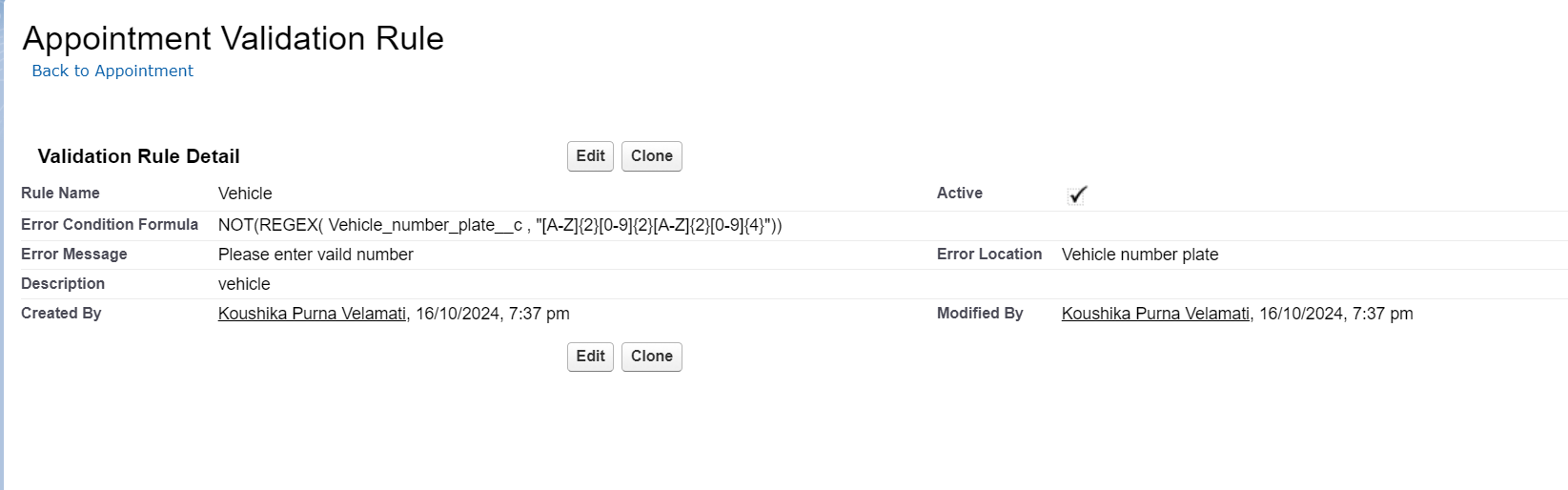
### Validation Rule

**18. Created Validation Rule in Appointment Object**

We created a **Validation Rule** in the **Appointment** object by going to **Object Manager** and editing the **Appointment** object. Under **Validation Rules**, we clicked **New** and named the rule **Vehicle**. The error condition formula was set to:

NOT(REGEX(Vehicle\_number\_plate\_\_c, "[A-Z]{2}[0-9]{2}[A-Z]{2}[0-9]{4}")).

The **Error Message** was set to "Please enter valid number," with the error location set to **Field** and the field selected as **Vehicle number plate**. After clicking **Save**, the rule was created.

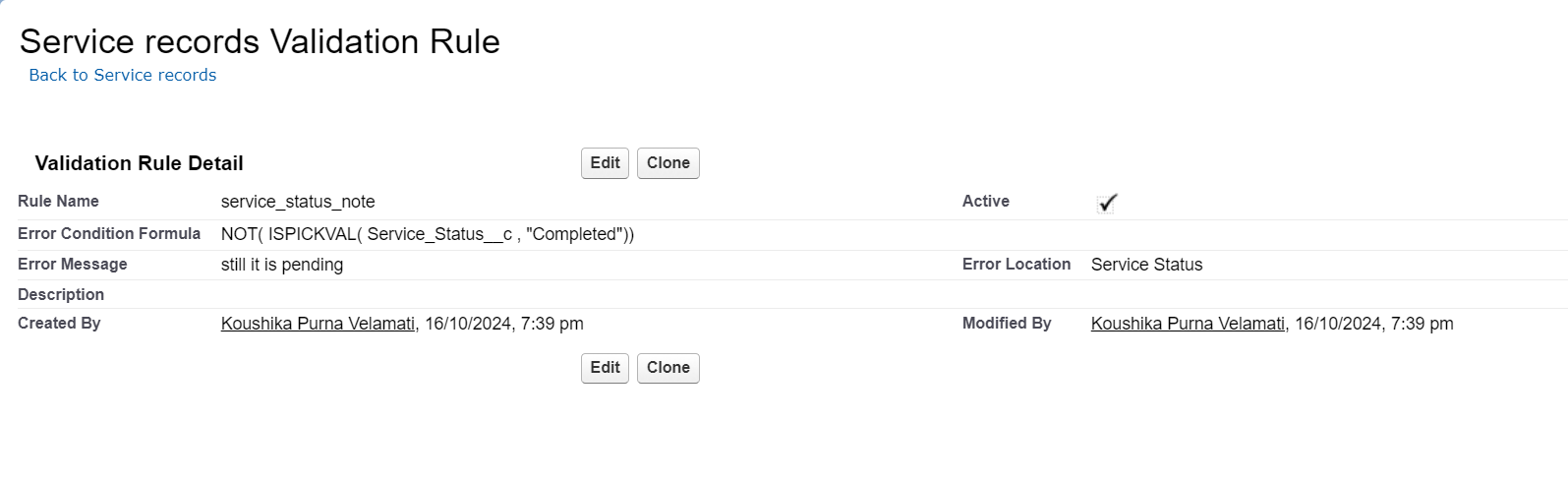
****

**19. Created Validation Rule in Service Records Object**

We created a **Validation Rule** in the **Service records** object by going to **Object Manager** and editing the **Service records** object. Under **Validation Rules**, we clicked **New** and named the rule **service\_status\_note**. The error condition formula was set to:

NOT(ISPICKVAL(Service\_Status\_\_c, "Completed")).

The **Error Message** was set to "Still it is pending," with the error location set to **Field** and the field selected as **Service status**. After clicking **Save**, the rule was created.

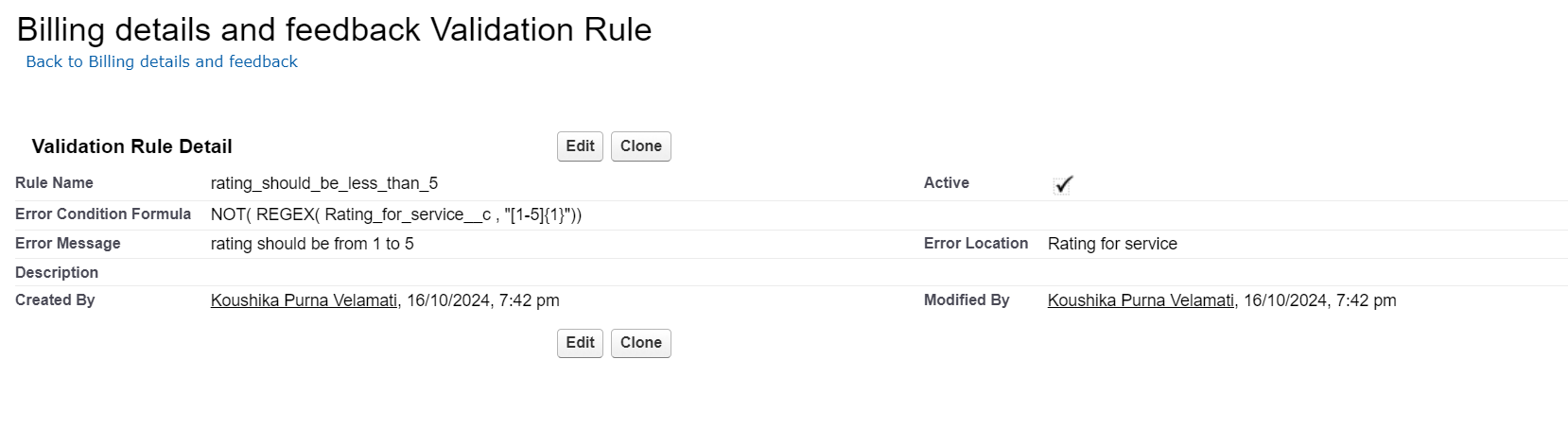
****

**20. Created Validation Rule in Billing Details and Feedback Object**

We created a **Validation Rule** in the **Billing details and feedback** object by going to **Object Manager** and editing the **Billing details and feedback** object. Under **Validation Rules**, we clicked **New** and named the rule **rating\_should\_be\_less\_than\_5**. The error condition formula was set to:

NOT(REGEX(Rating\_for\_service\_\_c, "[1-5]{1}")).

The **Error Message** was set to "Rating should be from 1 to 5," with the error location set to **Field** and the field selected as **Rating for Service**. After clicking **Save**, the rule was created.

****

### Duplicate rule

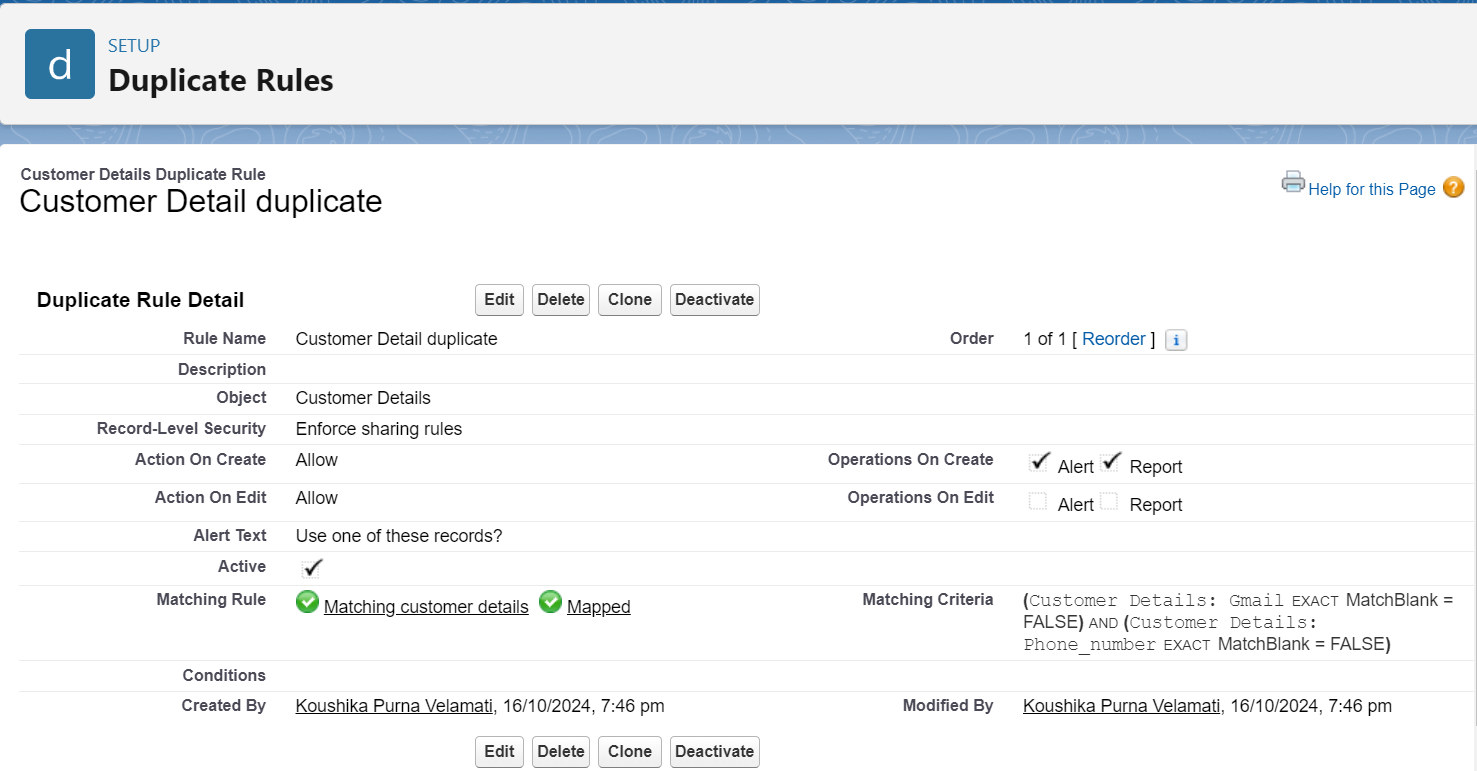
**21. Created Matching Rule in Customer Details Object**We created a **Matching Rule** for the **Customer details** object by searching for **Matching Rule** in the Quick Find box in setup and clicking **New Rule**. We selected **Customer details** as the object and clicked **Next**. The rule was named **Matching customer details**, with the unique name auto-populated. The matching criteria were defined as:

* **Gmail**: Exact
* **Phone Number**: Exact  
  After clicking **Save**, we activated the rule by selecting **Activate**.

### 

**22. Created Duplicate Rule in Customer Details Object**

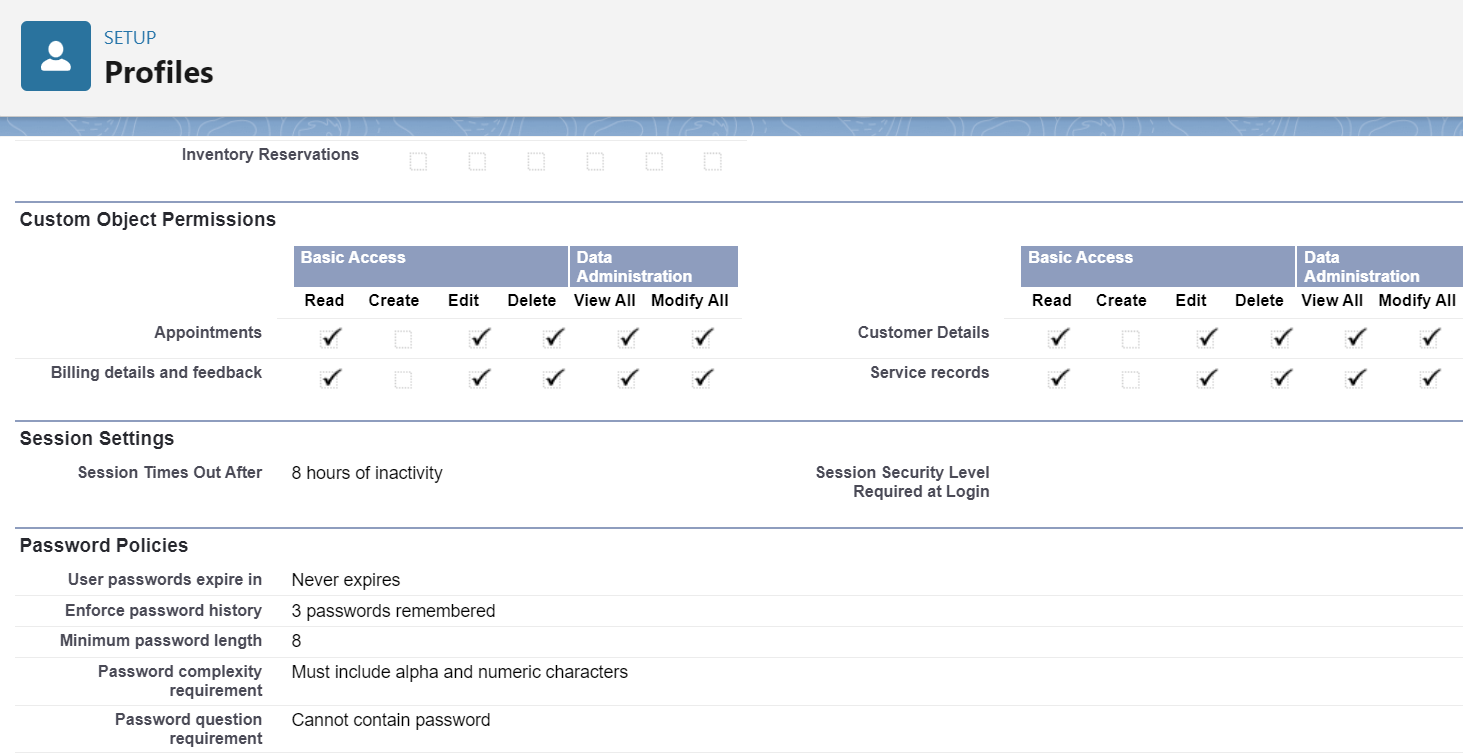
We created a **Duplicate Rule** for the **Customer details** object by searching for **Duplicate Rules** in the Quick Find box and clicking **New Rule**. We selected **Customer details** as the object and named the rule **Customer Detail duplicate**. In the **Matching Rule** section, we selected the previously created rule, **Matching customer details**, and clicked **Save**. After saving the duplicate rule, we clicked **Activate** to enable it.



### Profiles

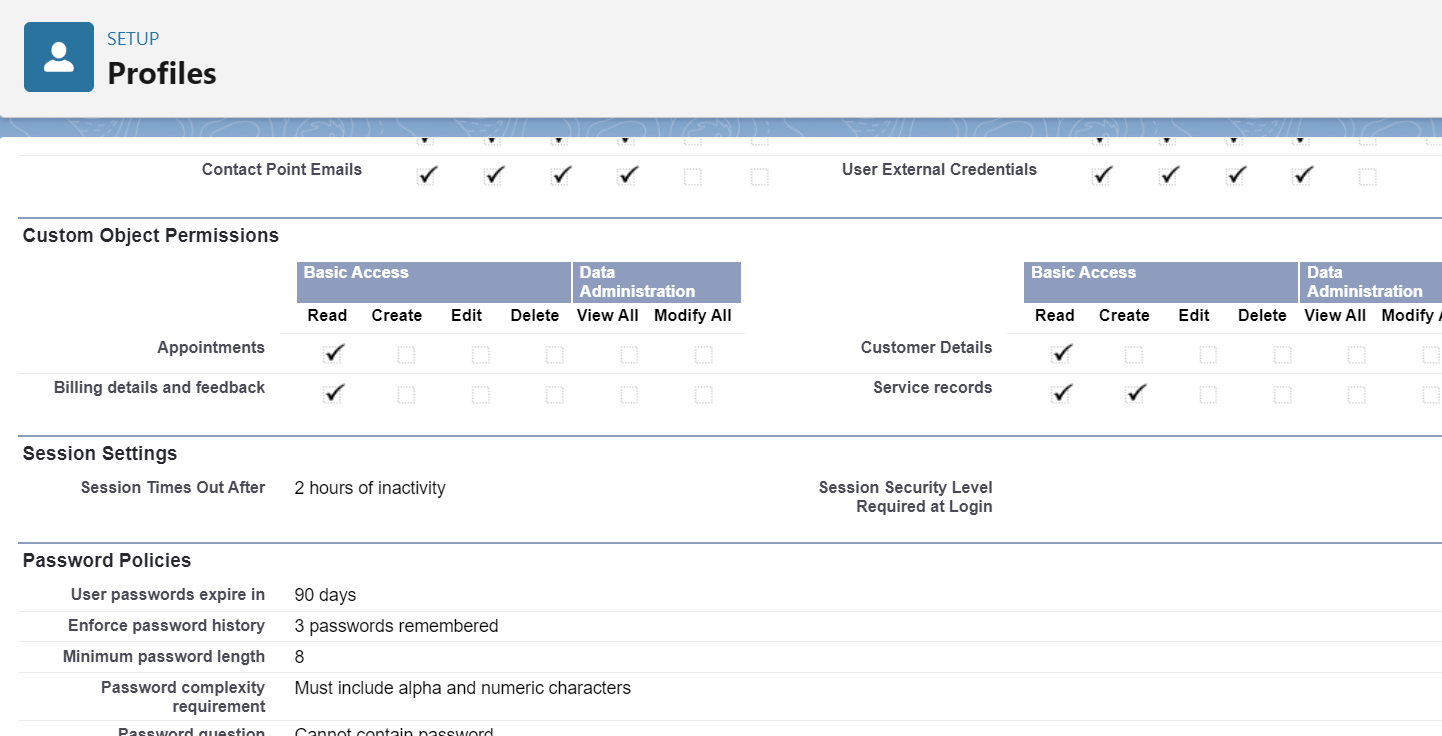
**23. Created Manager Profile**We created a **Manager Profile** by going to **Setup**, searching for **Profiles**, and cloning the **Standard User** profile. The new profile was named **Manager** and saved. After clicking **Edit**, we set the **Custom App Settings** to default for the **Garage Management** app. We then updated the **Custom Object Permissions** to grant access for **Appointments**, **Billing details and feedback**, **Service records**, and **Customer details** objects, as shown in the diagram. Additionally, we adjusted the **Session Timeout** to 8 hours of inactivity and changed the **Password Policies**:

* **User passwords expire**: Never expires
* **Minimum password length**: 8  
  After making these changes, we clicked **Save**.

****

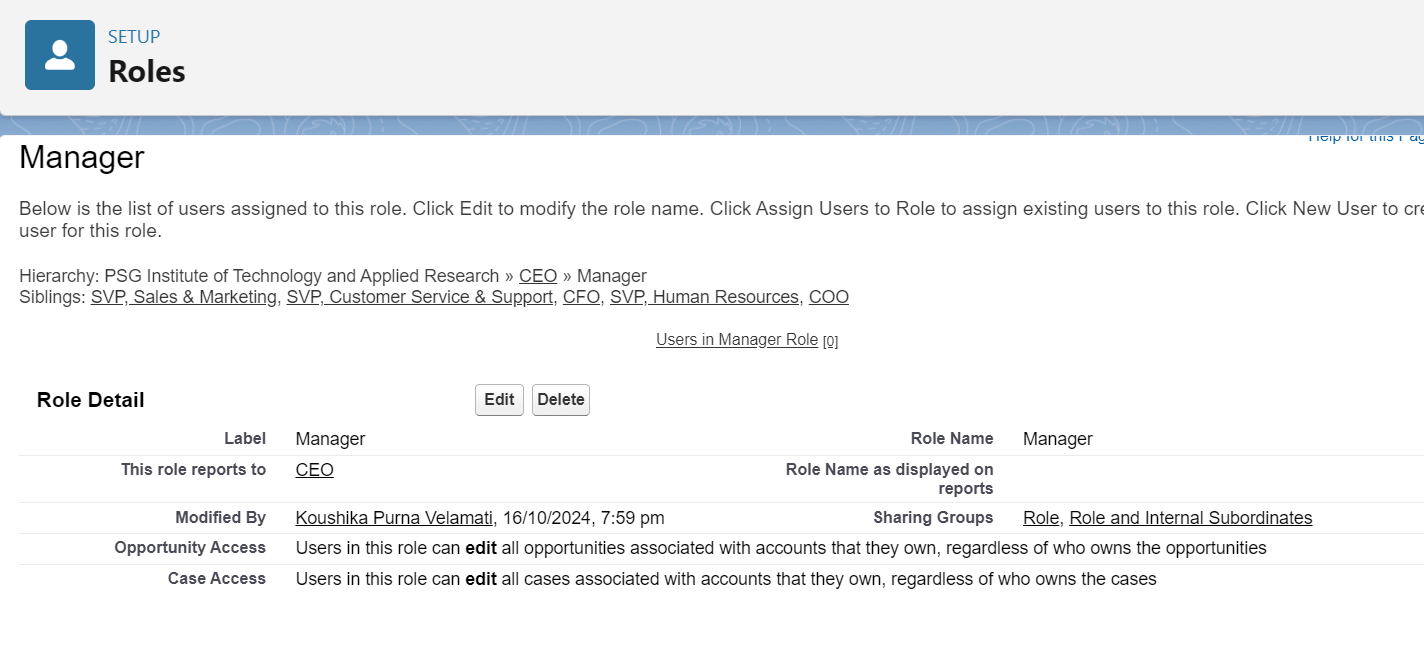
**24. Created Sales Person Profile**

We created a **Sales Person Profile** by going to **Setup**, searching for **Profiles**, and cloning the **Salesforce Platform User** profile. The new profile was named **Sales Person** and saved. After clicking **Edit**, we set the **Custom App Settings** to default for the **Garage Management** app. We then updated the **Custom Object Permissions** to grant access for **Appointments**, **Billing details and feedback**, **Service records**, and **Customer details** objects, as shown in the diagram. Finally, we clicked **Save** to finalize the profile setup.

****

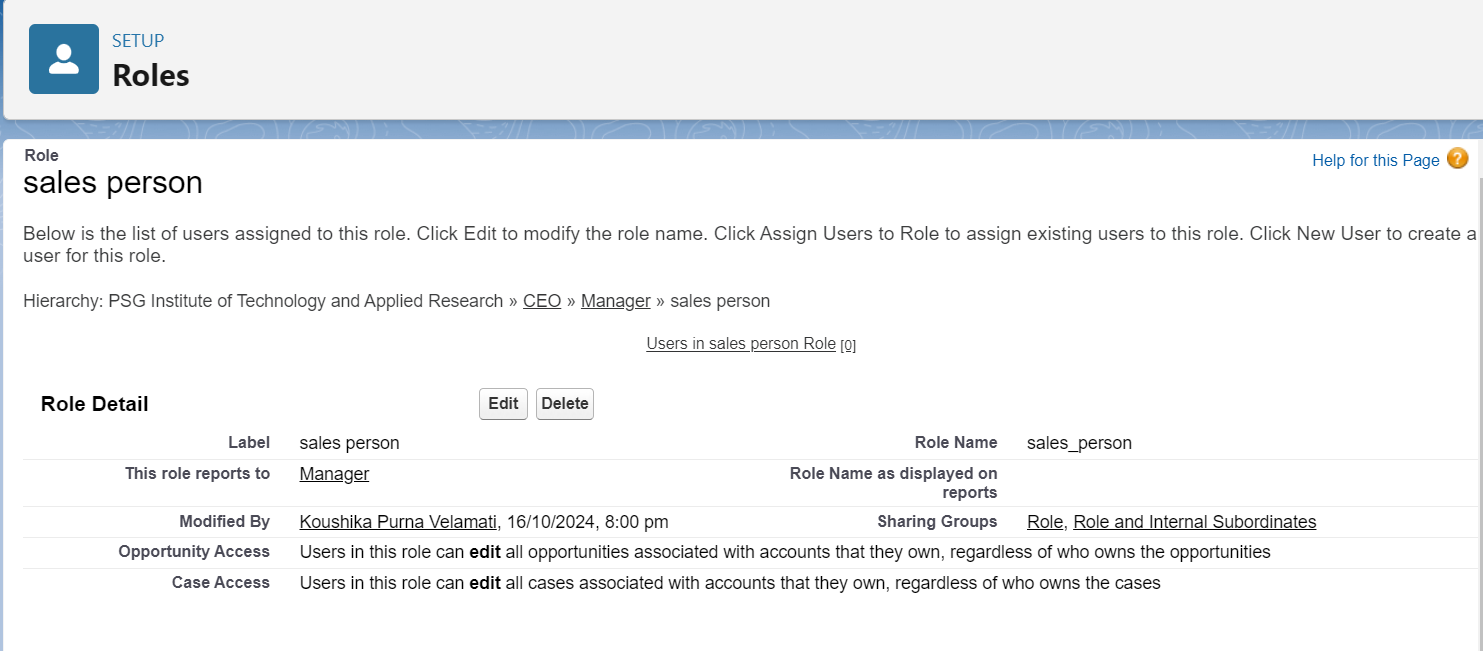
### Role & Role Hierarchy

**25. Created Manager Role**

We created a **Manager Role** by searching for **Roles** in the Quick Find box and clicking on **Setup Roles**. After expanding all roles, we clicked **Add Role** under the appropriate hierarchy. We named the role **Manager**, and the **Role Name** was auto-populated. Finally, we clicked **Save** to complete the role creation.****

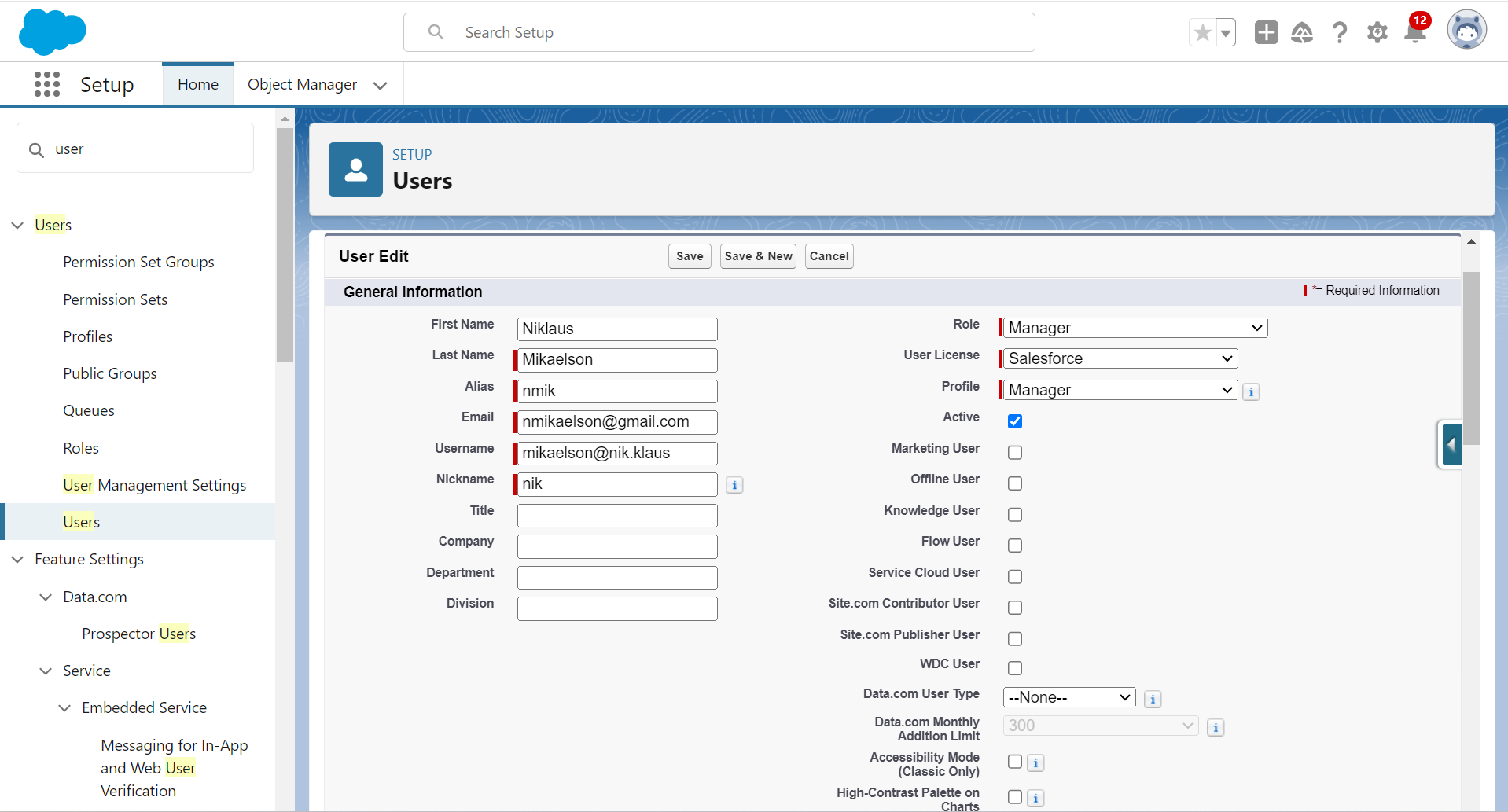
**26. Created Sales Person Role under Manager**

We created a **Sales Person Role** under the **Manager** by searching for **Roles** in the Quick Find box and clicking on **Setup Roles**. After clicking the **plus** icon next to the **CEO** role, we clicked **Add Role** under **Manager**. We named the role **Sales Person**, and the **Role Name** was auto-populated. Finally, we clicked **Save** to create the role.

****

### Users

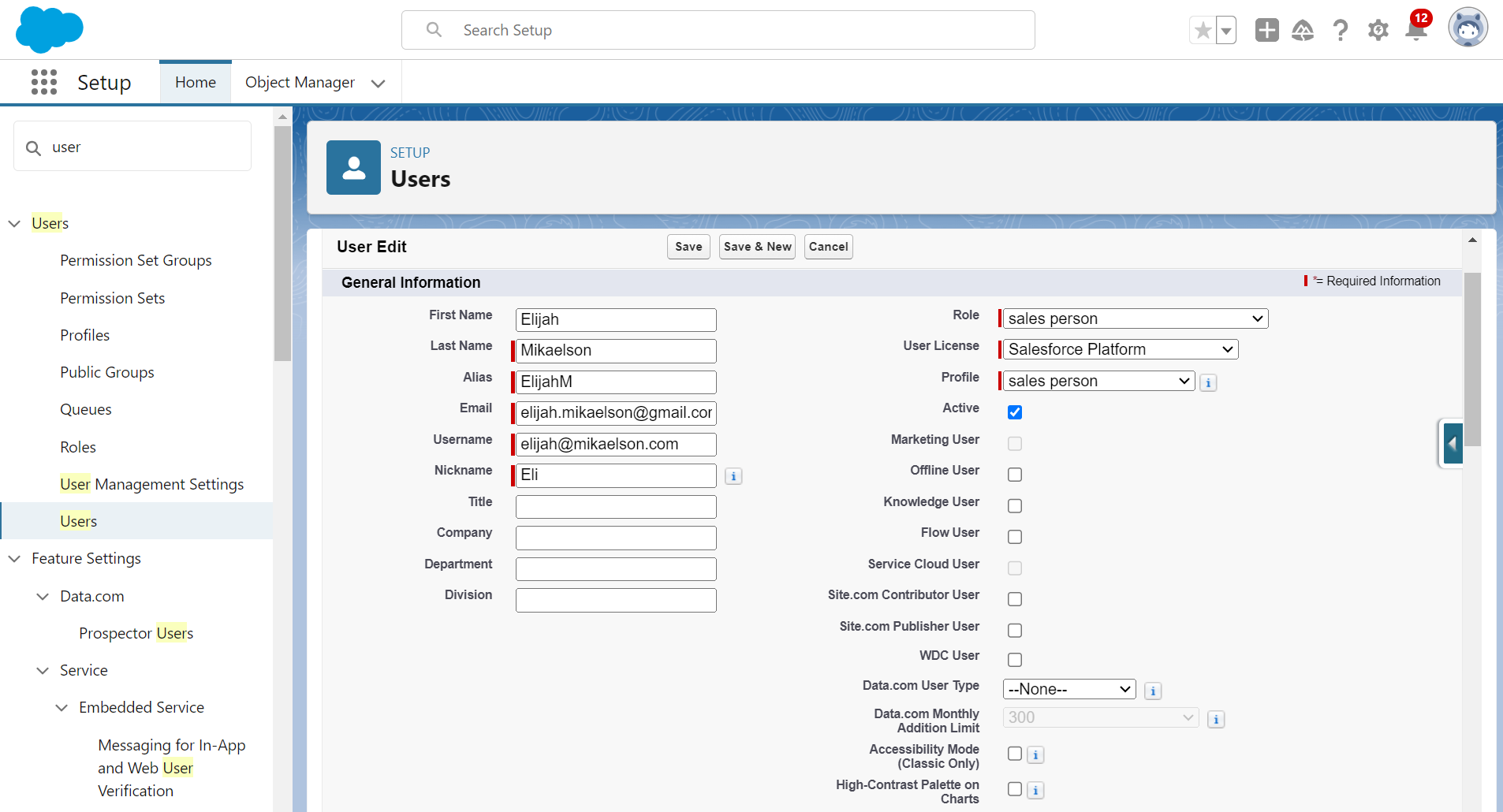
**27. Created User**

We created a new **User** by navigating to **Setup**, searching for **Users** in the Quick Find box, and selecting **Users**. After clicking **New User**, we filled in the required fields: First Name as Niklaus, Last Name as Mikaelson, Alias with a custom name, Email with a personal ID, and Username in the format text@text.text. We also provided a nickname, set the **Role** as Manager, selected **Salesforce** for the **User License**, and assigned the **Manager** profile. After filling in the details, we clicked **Save** to create the user.****

**28. Created Additional Users**We repeated the previous steps to create additional users, this time assigning the following details for each user:

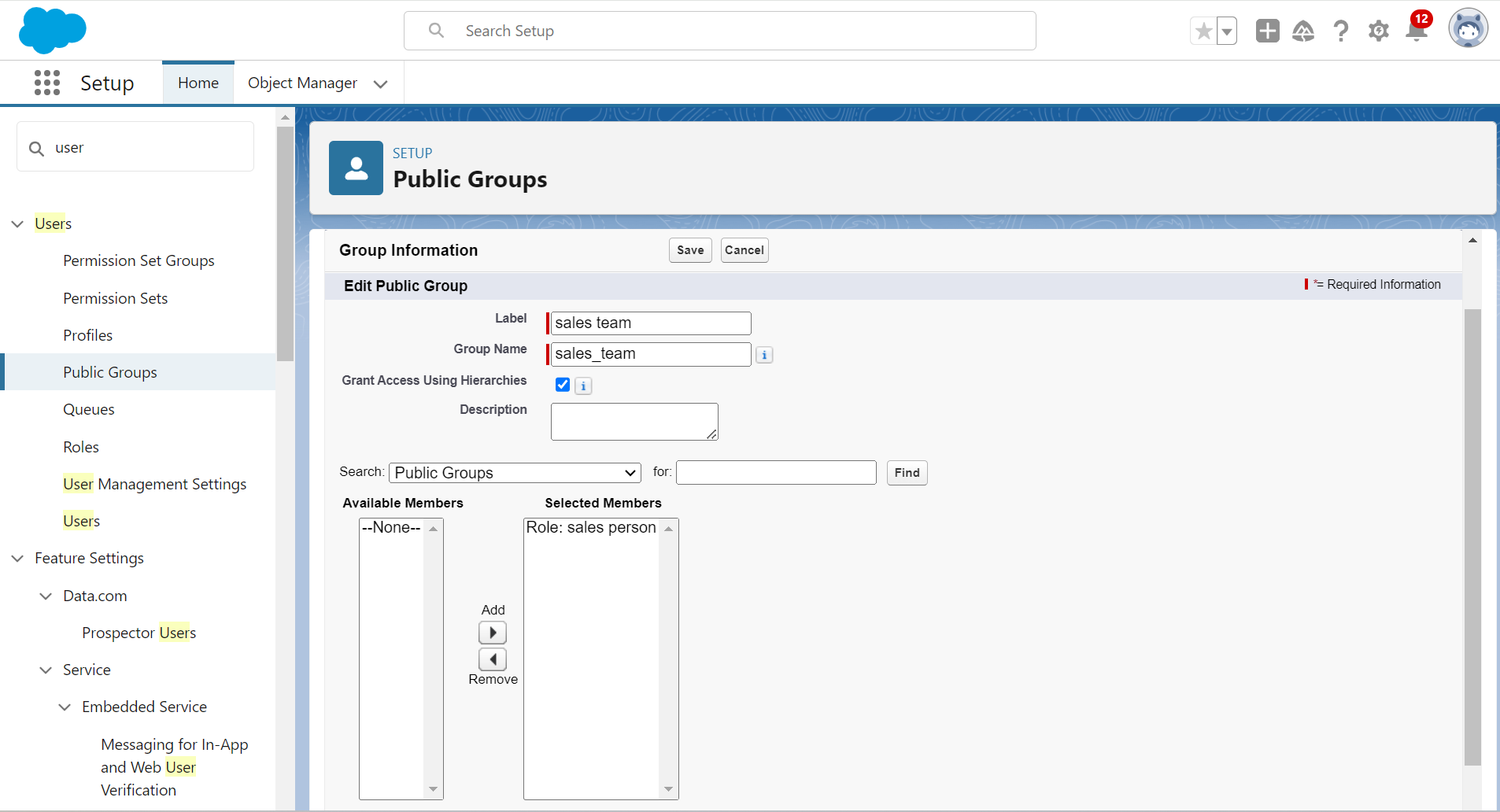
* **Role**: Sales Person
* **User License**: Salesforce Platform
* **Profile**: Sales Person

We ensured to create at least three users with these permissions by following the same process and clicking **Save** after each user setup.

****

### Public groups

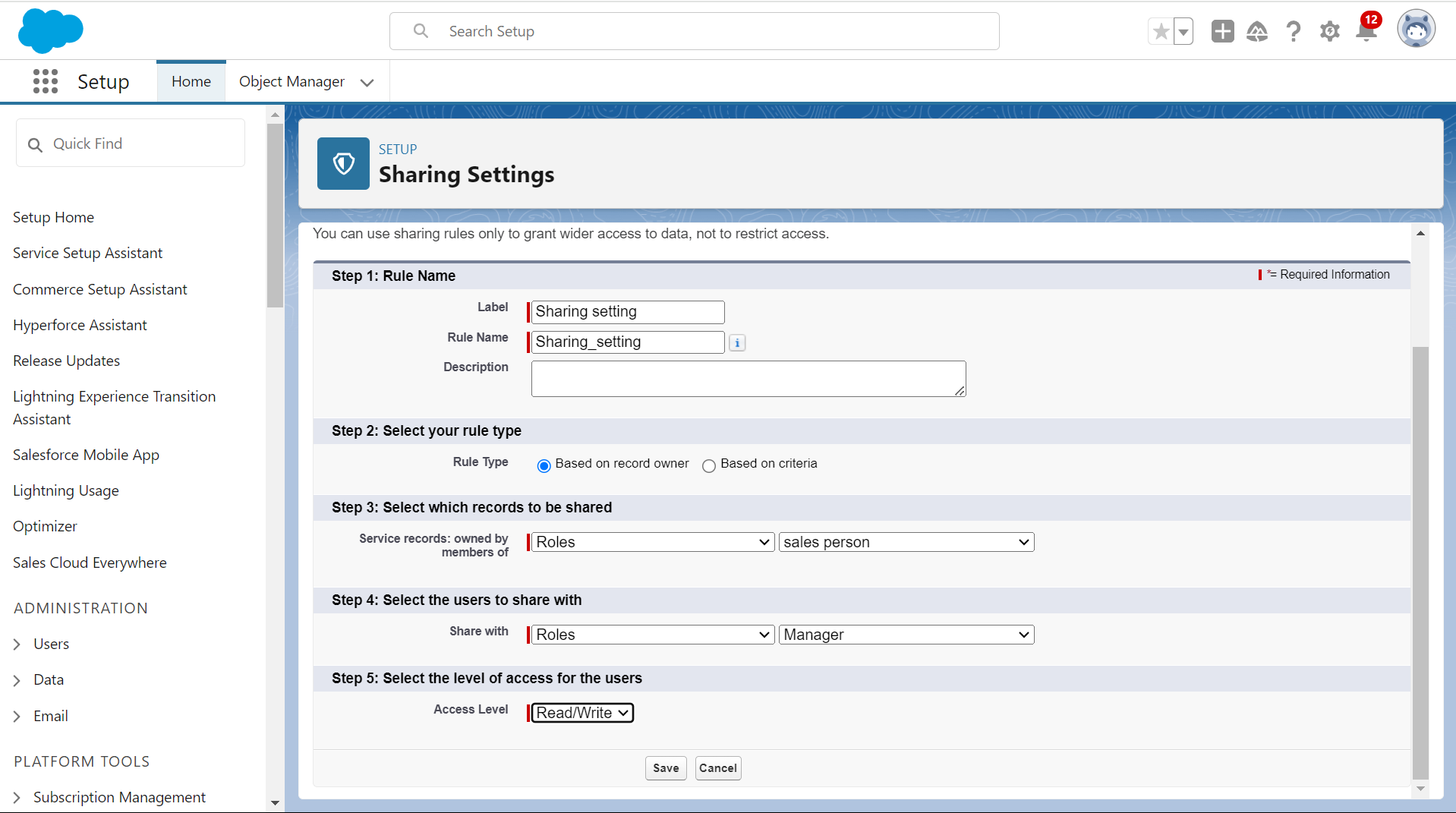
**29. Created New Public Group**We created a new **Public Group** by navigating to **Setup**, typing **Users** in the Quick Find box, and selecting **Public Groups**. Then, we clicked **New** and gave the label as **Sales Team** (with the group name auto-populated). In the **Available Members** section, we searched for **Roles**, selected **Sales Person**, and added it to the **Selected Members**. Finally, we clicked **Save** to create the group.

****

### Sharing Setting

**30. Created Sharing Settings**

To configure **Sharing Settings**, we went to **Setup**, typed **Users** in the Quick Find box, selected **Sharing Settings**, and clicked **Edit**. We changed the **OWD (Organization-Wide Default)** setting for the **Service Records** object to **Private**. After saving and refreshing, we scrolled down and clicked **New** under **Service Records Sharing Rules**. For the new rule, we gave the **Label** as **Sharing Setting** (with the **Rule Name** auto-populated). In Step 3, we selected **Roles** and **Sales Person** as the records to be shared. In Step 4, we selected **Roles** and **Manager** as the sharing recipients, and in Step 5, we set the **Access Level** to **Read/Write**. Finally, we clicked **Save**.

****

### Flows

**31. Creating a Flow**To create a flow, go to **Setup**, type **Flow** in the Quick Find box, and click on **Flows**. Select **New Flow** and choose **Record-Triggered Flow**, then click **Create**. In the flow setup, select **Billing details and feedback** as the object and set the trigger to **A record is Created or Updated**. For optimization, choose **Actions and Related Records** and click **Done**.

Under **Record-Triggered Flow**, click the **+** symbol and select **Update Records**. For the **Label Name**, enter "Amount Update," and let the API name auto-populate. Set a filter condition where **Payment\_Status\_\_c** equals **Completed**, and under **Field Values**, update the **Payment\_Paid\_\_c** field to the value of {!$Record.Service\_records\_\_r.Appointment\_\_r.Service\_Amount\_\_c}. Click **Done**.

Next, create a new resource. In the toolbox, select **New Resource**, choose **Variable**, and set the resource type to **Text Template**. Name the resource "alert" and change the view to **Plain Text**. In the body, enter the message:

"Dear {!$Record.Service\_records\_\_r.Appointment\_\_r.Customer\_Name\_\_r.Name},

I hope this message finds you well. I wanted to take a moment to express my sincere gratitude for your recent payment for the services provided by our garage management team. Your prompt payment is greatly appreciated, and it helps us continue to provide top-notch services to you and all our valued customers.

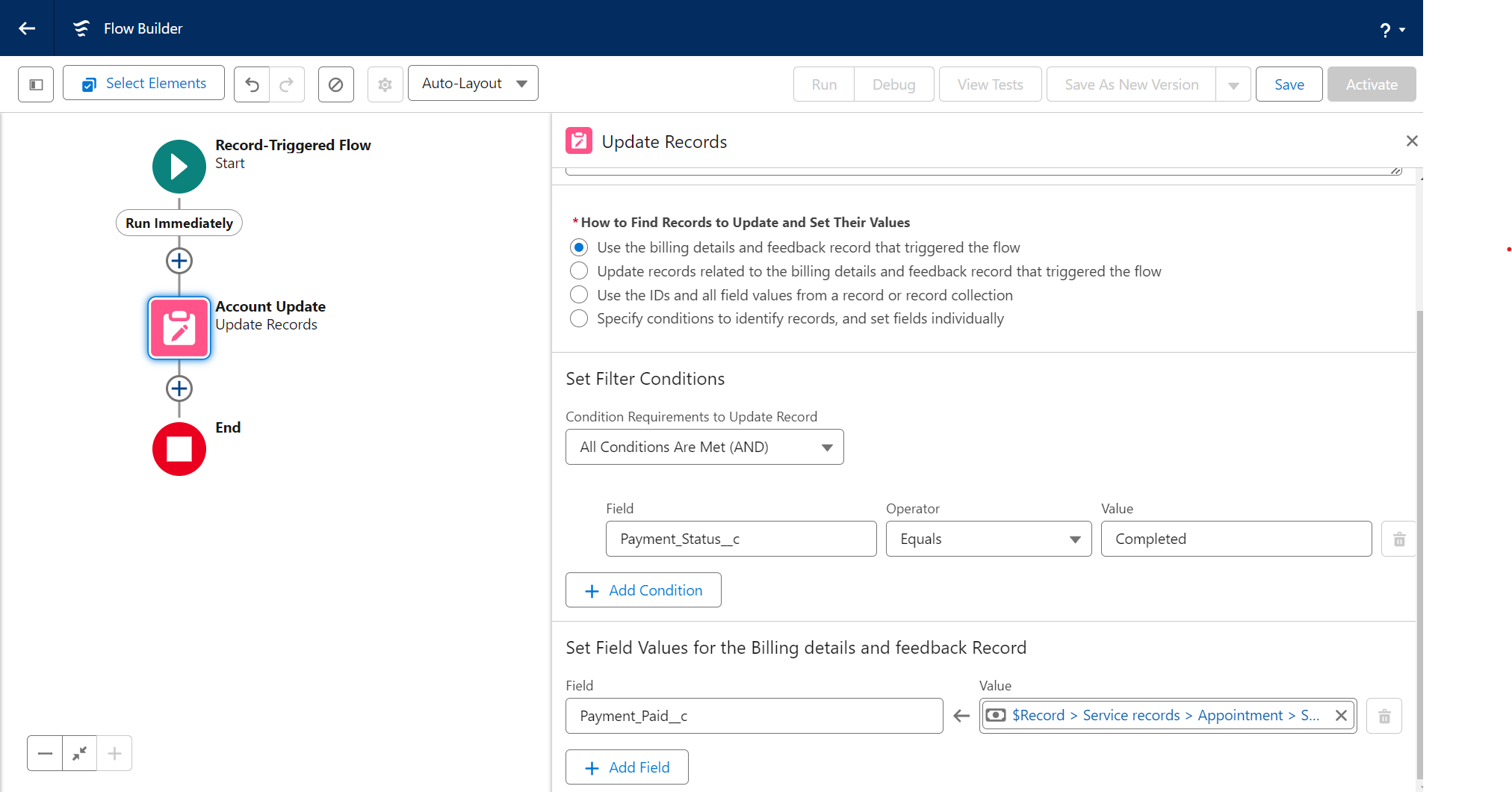
Amount paid: {!$Record.Payment\_Paid\_\_c}

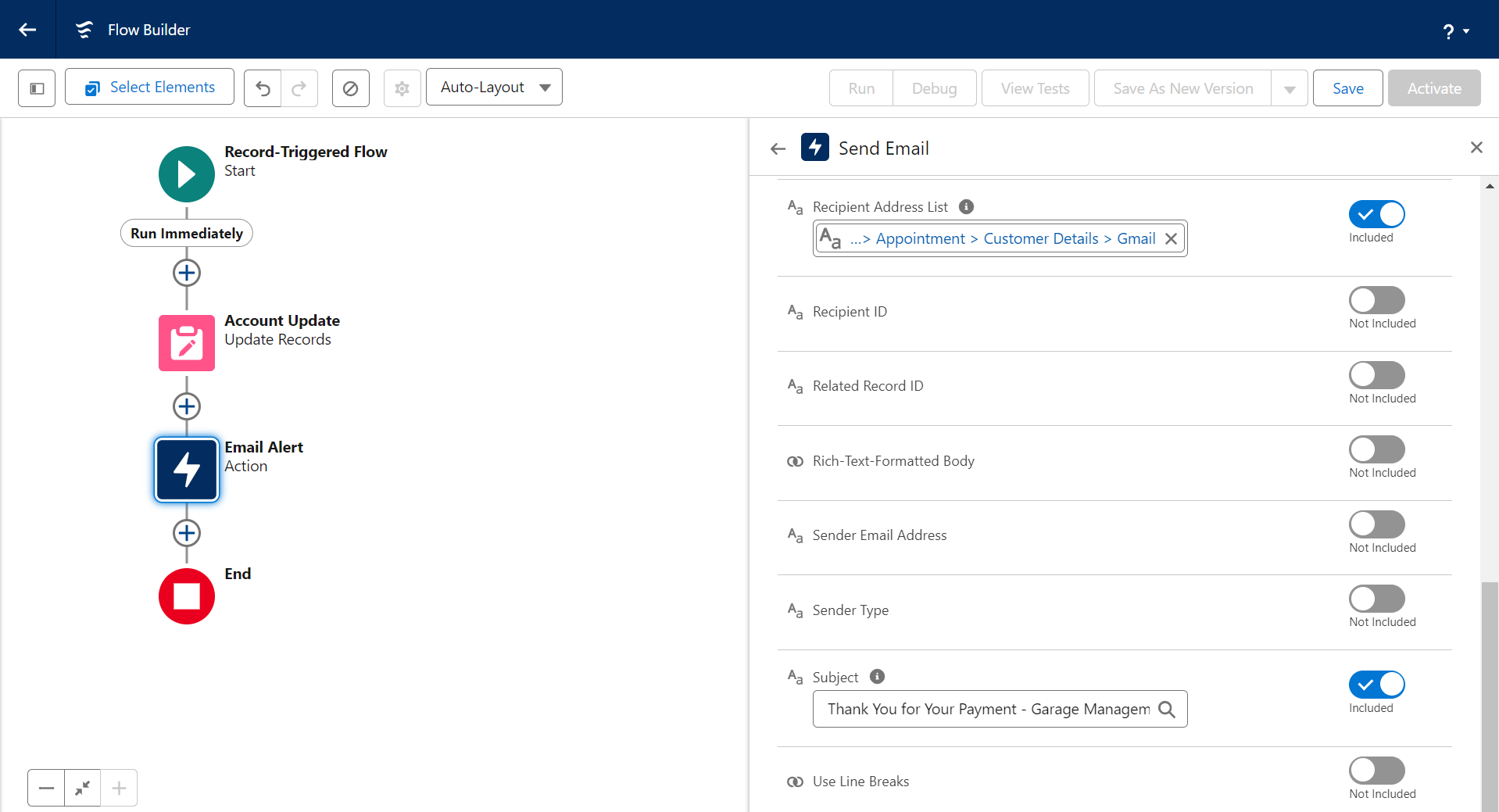
Thank you for coming."

Click **Done**.

Then, click **Add Element**, select **Action**, search for **Send Email**, and click on it. Label the action as **Email Alert** and let the API name auto-populate. Enable the body for input values and select the text template **Body: {!alert}**. Set the recipient's email address from the record, selecting {!$Record.Service\_records\_\_r.Appointment\_\_r.Customer\_Name\_\_r.Gmail\_\_c}. Set the subject as **Thank You for Your Payment - Garage Management** and click **Done**.

Finally, click **Save**, give the flow a label (the API name will auto-populate), click **Save** again, and click **Activate**.

****

****

### Apex Trigger

**32.Apex Handler: Amount Distribution Use Case**This use case handles the amount distribution for each service selected by the customer for their vehicle. The AmountDistributionHandler class is used to calculate and assign the service amounts based on the services selected in the Appointment\_\_c object.

1. **Creating the Apex Handler Class:**To create the handler, log in to the respective Trailhead account and navigate to the gear icon in the top right corner. Click on **Developer Console** to open a new console window. In the toolbar, click **File**, select **New**, and create a new Apex class named **AmountDistributionHandler**.

**Code for AmountDistributionHandler:**

public class AmountDistributionHandler {

public static void amountDist(List<Appointment\_\_c> listApp){

List<Service\_records\_\_c> serList = new List<Service\_records\_\_c>();

for(Appointment\_\_c app : listApp){

if(app.Maintenance\_service\_\_c == true && app.Repairs\_\_c == true && app.Replacement\_Parts\_\_c == true){

app.Service\_Amount\_\_c = 10000;

}

else if(app.Maintenance\_service\_\_c == true && app.Repairs\_\_c == true){

app.Service\_Amount\_\_c = 5000;

}

else if(app.Maintenance\_service\_\_c == true && app.Replacement\_Parts\_\_c == true){

app.Service\_Amount\_\_c = 8000;

}

else if(app.Repairs\_\_c == true && app.Replacement\_Parts\_\_c == true){

app.Service\_Amount\_\_c = 7000;

}

else if(app.Maintenance\_service\_\_c == true){

app.Service\_Amount\_\_c = 2000;

}

else if(app.Repairs\_\_c == true){

app.Service\_Amount\_\_c = 3000;

}

else if(app.Replacement\_Parts\_\_c == true){

app.Service\_Amount\_\_c = 5000;

}

}

}

}

**Creating the Trigger:**

In the Developer Console, click on **File**, select **New**, and then click on **Trigger**. Enter the **Trigger Name** as **AmountDistribution** and select the **Object** as **Appointment\_\_c**.

**Code for AmountDistribution Trigger:**

trigger AmountDistribution on Appointment\_\_c (before insert, before update) {

if(trigger.isbefore && (trigger.isinsert || trigger.isupdate)){

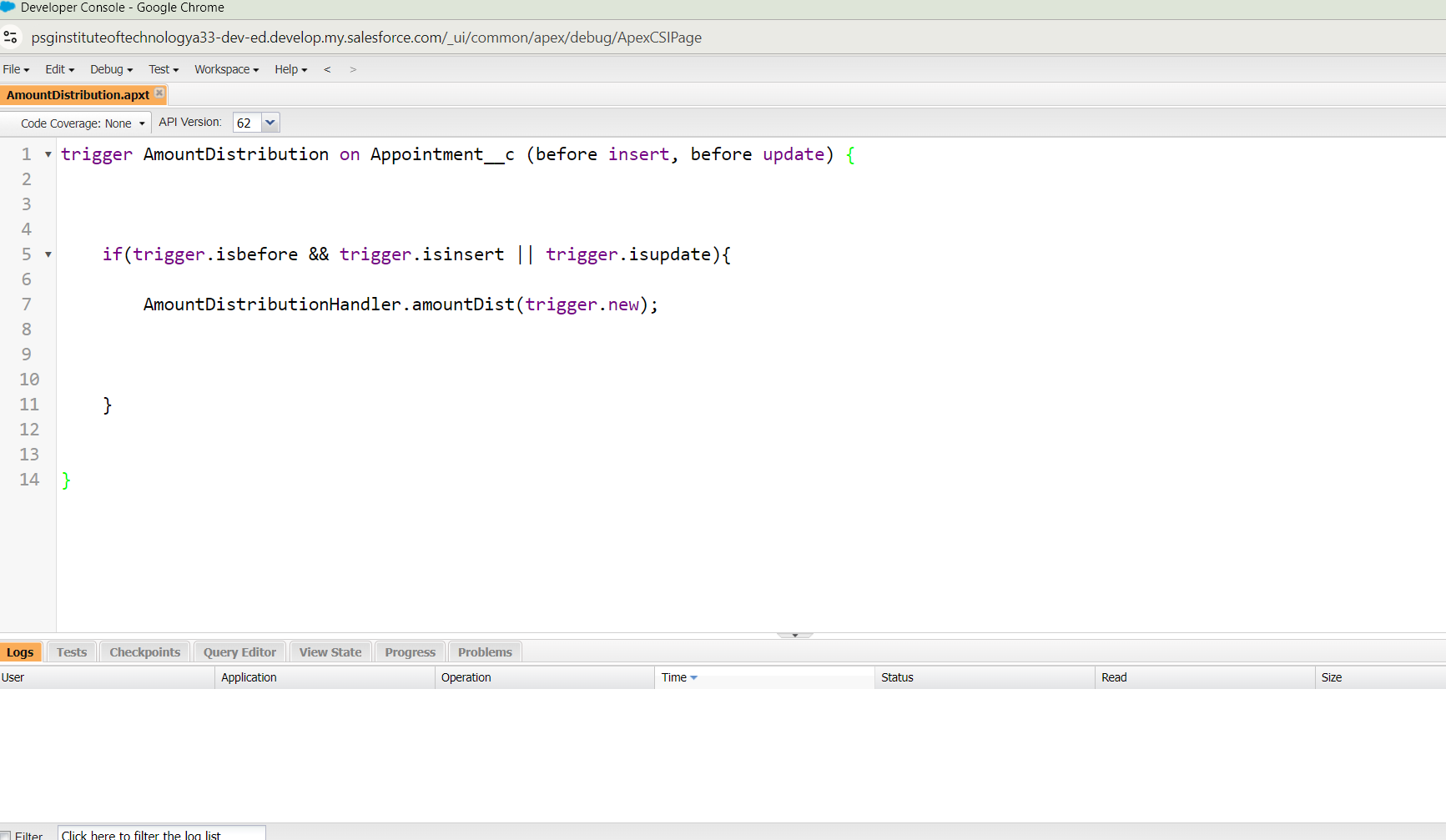
AmountDistributionHandler.amountDist(trigger.new);

}

}

The trigger is designed to fire before insert or update events on the Appointment\_\_c object. When the trigger is fired, it calls the amountDist method from the AmountDistributionHandler class to calculate the service amount based on the selected services.

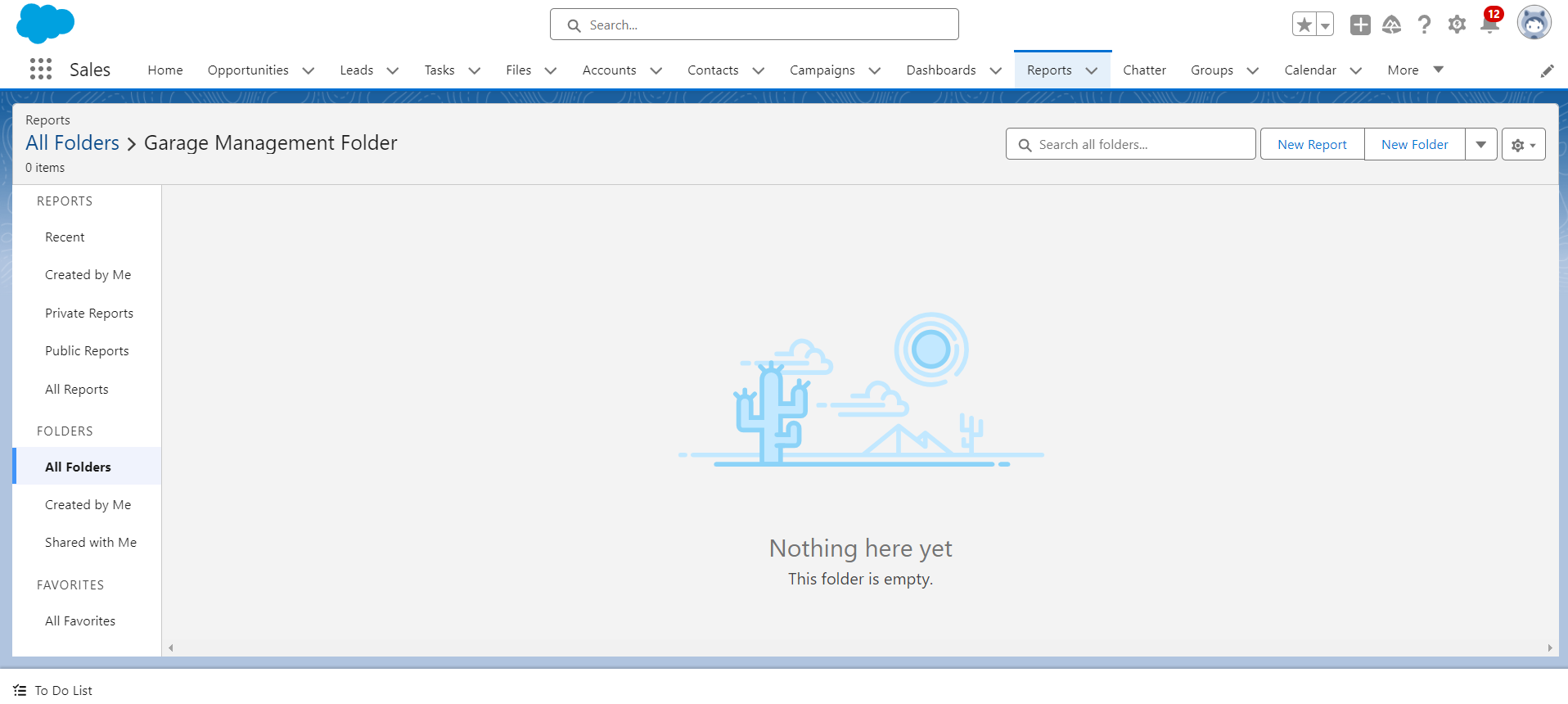
By using this handler and trigger, the system automatically calculates and assigns the appropriate service amount to the Service\_Amount\_\_c field on the Appointment\_\_c object, based on the selected services.



### Reports

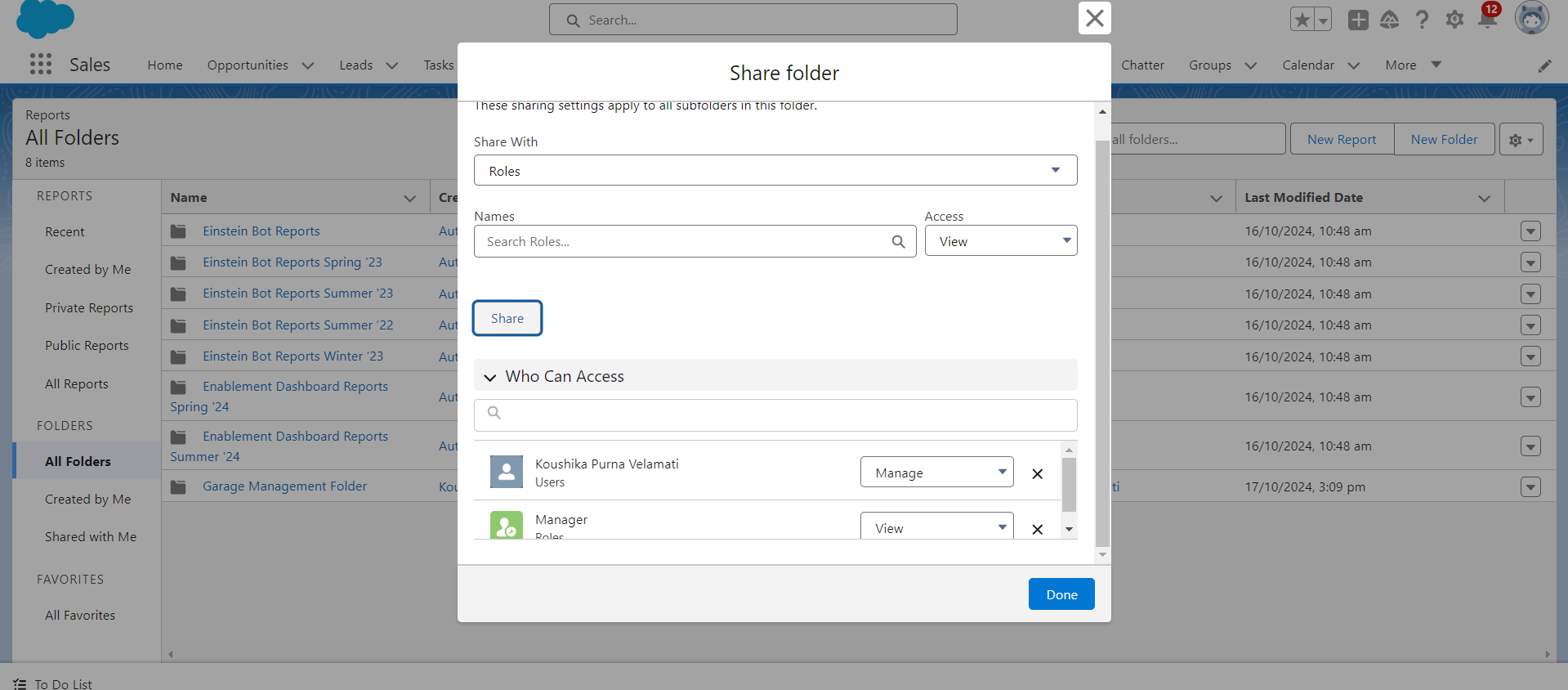
**33. Creating a Report Folder**

To create a report folder in Salesforce, go to the **App Launcher** and search for **Reports**. Select the **Reports** tab, then click on **New Folder**. In the folder creation window, provide the **Folder Label** as **Garage Management Folder**. The **Folder Unique Name** will be auto-populated. Finally, click **Save** to create the folder. This folder will help you organize reports related to the garage management system.

****

**34. Sharing a Report Folder**

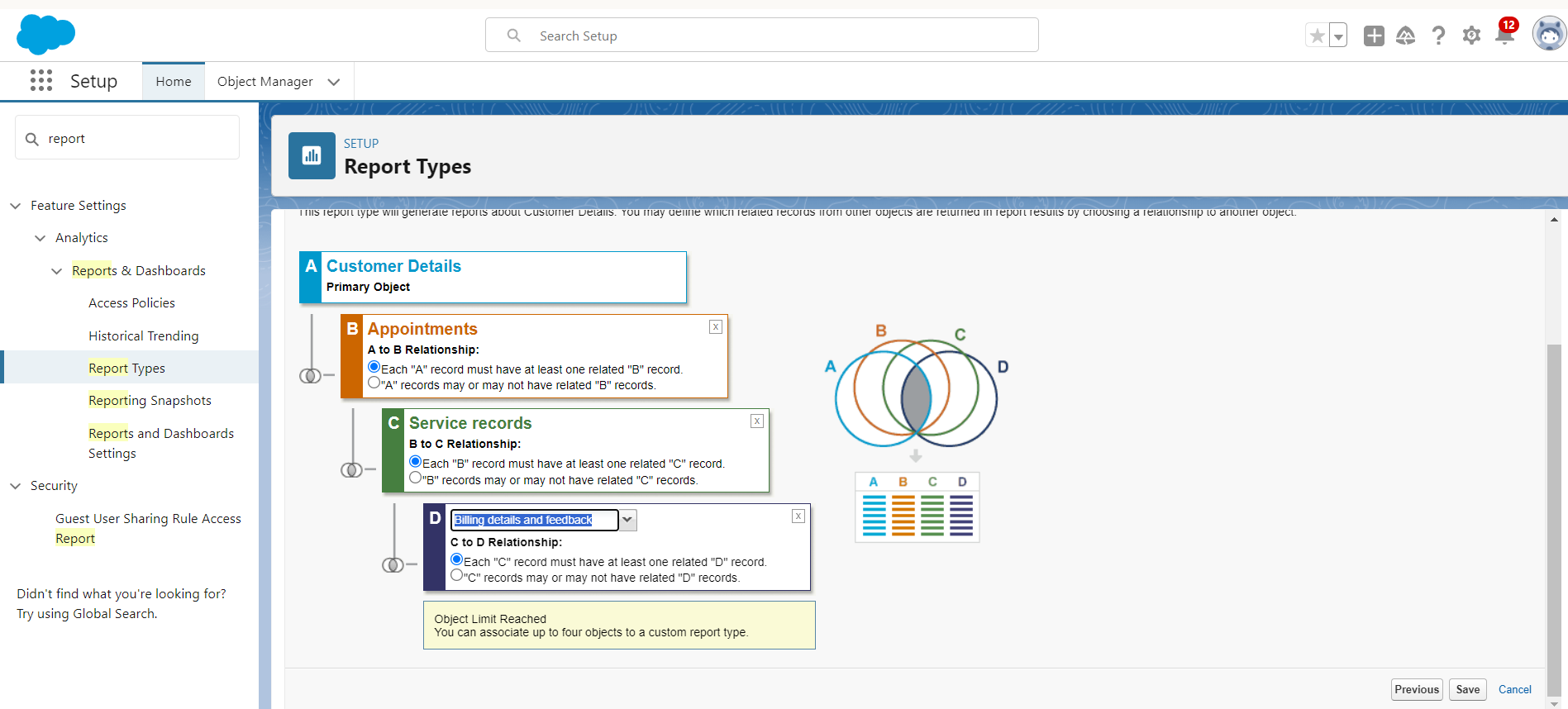
To share a report folder, go to the **App** and click on the **Reports** tab. Then, click on the **All Folders** section, locate the **Garage Management Folder**, and click on the drop-down arrow next to it. Select **Share**. In the sharing settings, choose **Roles** in the **Share With** field and search for **Manager** in the **Name** field. Set the access level to **View** for the **Manager** role. After that, click **Share** and then click **Done** to complete the sharing process.

****

**35. Create a Report Type**

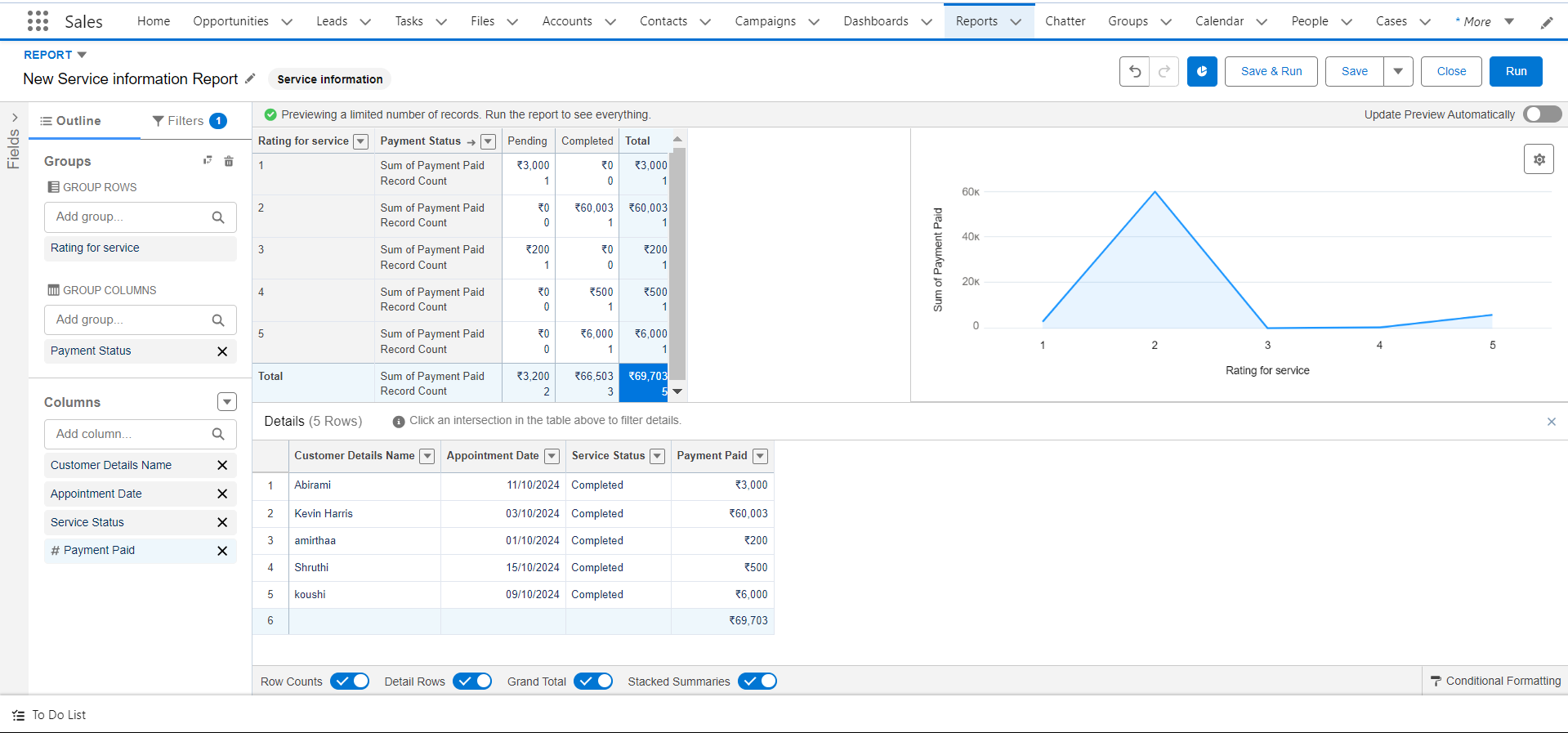
To create a custom report type, go to **Setup**, type **Report Type** in the Quick Find box, and click on **Continue**. Then, click on **New Custom Report Type**. Select **Customer Details** as the Primary Object. Enter the **Report Type Label** as **Service Information** (the **Report Type Name** will auto-populate) and keep the **Description** the same. Choose **Other Reports** for **Store in Category**, and set the **Deployment Status** to **Deployed**, then click **Next**.

Next, click on the **Related Object** box. Select **Appointment** as the first related object, and then select **Service Records** as the second related object. Repeat the process to select **Billing Details and Feedback** as another related object. Finally, click **Save** to complete the creation of the report type.

****

**36.Create a Report**

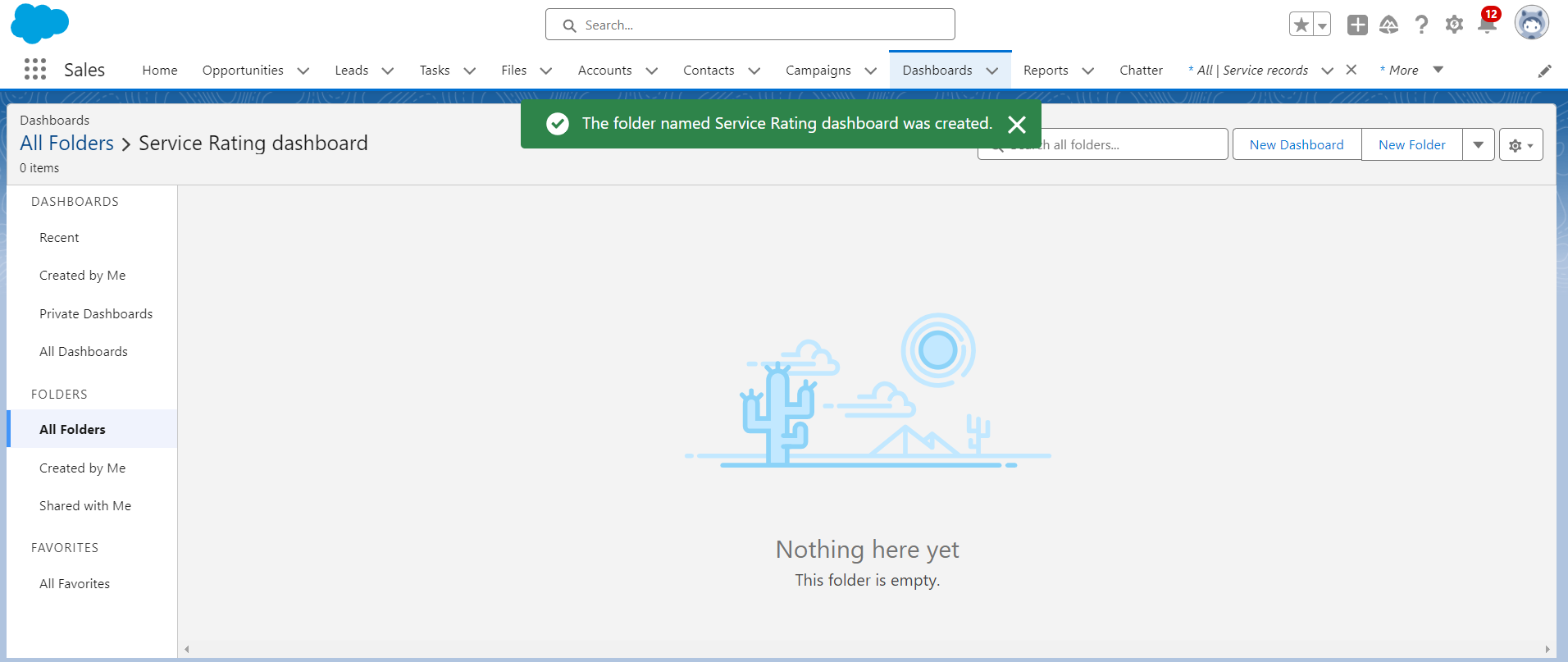
To create a report, first ensure that you have created the latest 10 records in each object (Customer Details, Appointment, Service Records, and Billing Details and Feedback) and filled in all fields for a better experience. Then, go to the app and click on the **Reports** tab. Click **New Report**, select **Other Reports** as the category, and search for **Service Information**. Select it and click on **Start Report**. In the outline pane, choose the fields such as **Customer Name**, **Appointment Date**, **Service Status**, and **Payment Paid** for the **Columns Section**, and remove any unnecessary fields. For the **Group Rows Section**, select **Rating for Service** and **Payment Status**. Add a **Line Chart**, then click **Save**. Name the report **New Service Information Report** (the **Report Unique Name** will auto-populate), select the folder you created earlier, and click **Save** to finish the report creation.

****

### Dashboards

**37.Create a Dashboard Folder**

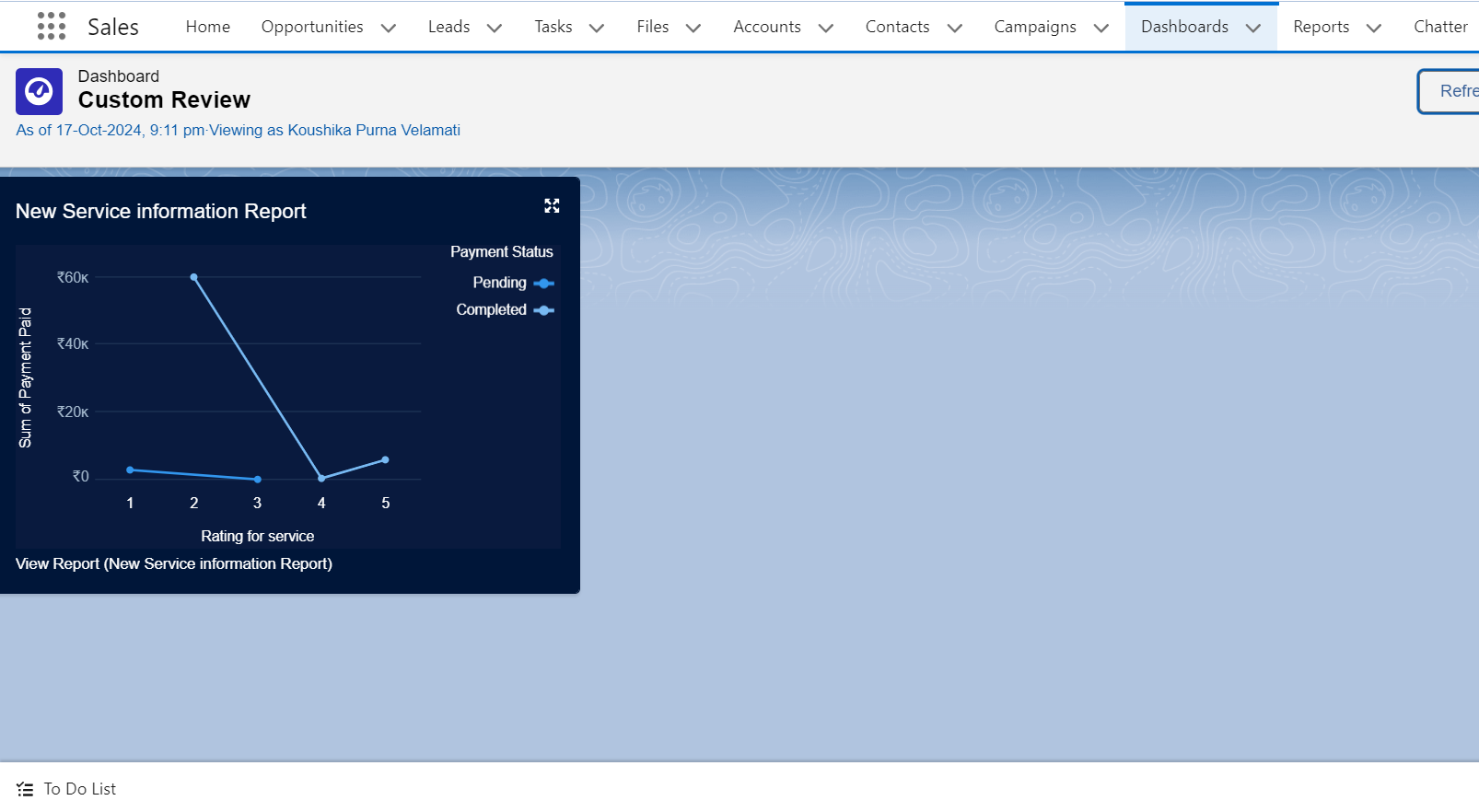
To create a dashboard folder, click on the **App Launcher** and search for **Dashboard**. Then, click on the **Dashboard** tab and click **New Folder**. In the folder creation window, give the folder label as **Service Rating Dashboard** (the **Folder Unique Name** will be auto-populated). Finally, click **Save** to create the folder.

****

**38.Creation of Dashboard**

To create a dashboard, go to the app and click on the **Dashboards** tab. Click on **Create** and provide a name for the dashboard. Select the **Service Rating Dashboard** folder you created earlier and click **Create**. Next, click on **Add Component** and choose a report. Select the **Line Chart** as the chart type, and adjust the theme if needed. After configuring the chart, click **Add**, then **Save**, and finally **Done**.

To set up subscriptions, click **Subscribe** at the top right, set the frequency to **Weekly**, choose **Monday** as the day, and click **Save**.

****

**5.Conclusion**

Summary of Achievements:  
The Garage Management System project on Salesforce led to a series of impactful achievements that significantly enhanced the performance and growth potential of the service center. Key accomplishments include:

* Increased Operational Efficiency: The automation of workflows, from booking appointments to managing service progress and customer communication, reduced the administrative burden on staff by up to 30%. This allowed employees to focus more on core activities, such as vehicle diagnostics and repair, improving service turnaround times and the center’s overall productivity.
* Revenue Growth through Targeted Upselling: Leveraging Salesforce’s data insights, the system enabled the identification of targeted upsell and cross-sell opportunities based on customer and vehicle history. By recommending relevant additional services, the center experienced an increase in potential revenue streams, aiming for a 10-15% growth through enhanced service offerings.
* Optimized Inventory Management: Real-time tracking and alerts for low stock helped streamline inventory processes. The system reduced instances of overstocking and shortages by 25%, ensuring that critical parts were readily available for service needs, minimizing delays, and reducing inventory costs.
* Reduced Errors and Service Delays: By automating manual processes and data entry, the system minimized errors and service delays by 15-20%, leading to a smoother, more reliable service experience. This efficiency enhancement further supported high-quality service and contributed to positive customer feedback.
* Improved Employee Productivity and Satisfaction: With centralized access to customer and service data, staff could complete their tasks more effectively, leading to increased productivity. Reducing repetitive administrative tasks also improved job satisfaction, helping to reduce turnover and build a more motivated team.

These accomplishments combined to strengthen the service center’s operational foundation, boost profitability, and establish a customer-centric approach to service. By delivering high-quality, timely services and building lasting relationships, the center is well-positioned for sustainable growth and competitive advantage.