ARUNAI ENGINEERING COLLEGE

**(An Autonomous Institution)**

**Velu nagar, Thiruvannamalai-606 603** [**www.arunai.org**](http://www.arunai.org/)

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**BACHELOR OF ENGINEERING 2024 - 2025**

**FIFTH SEMESTER**

**SALES AUTOMOBILE USING SALESFORCE CRM**

**TEAM MEMBERS:**

|  |  |  |
| --- | --- | --- |
| S.Vijay | - | 510422243060 |
| K.R.Perarasu | - | 510422243036 |
| S.Ajay devagan | - | 510422243001 |
| S.Franklin | - | 510422243011 |

**ARUNAI ENGINEERING COLLEGE**

(An Autonomous Institution)

**TIRUVANNAMALAI – 606 603**



### DEPARTMENT OF ARTIFICIAL INTELLIGNCE & DATA SCIENCE

Certified that this is a bonafide record of work done by

Name

: K.R.PERARASU

NM ID : 6BBDE6A1B5281DEBEBD47978C897CD27

University Reg.No

: 510422243036

Semester : 5

Branch Year

: AI&DS

: 2024-2025

## Staff-in-Charge Head of the Department

Submitted for the Practical Examination held on

### Internal Examiner External Examiner

# A CRM Application to Manage Mall

#### PROJECT VIEW:

The purpose of this project is to create a **Customer Relationship Management (CRM)** system that helps manage various aspects of a shopping mall, such as tenant management, customer interactions, sales tracking, and service management. This system would help mall administrators track tenants, manage customer data, improve customer service, and optimize mall operations through real-time data and analytics.

#### OBJECTIVES:

##### Business Goals:

To ensure that the CRM application can scale as the mall grows, supporting the addition of new tenants, new customer segments, and additional features.

##### Specific Outcomes:

* **Modular Design**: The system will be built in a modular way, allowing for the easy addition of new functionalities, like new sales tracking capabilities or integration with third-party marketing tools.
* **Flexible Data Architecture**: The CRM will support the addition of new data points as the mall evolves, enabling the tracking of new customer interactions or emerging trends.
* **Future-Proof System**: Integration with future technologies such as AI-based analytics or mobile apps to provide even more advanced functionality as the business grows.

##### Salesforce Key Features and Concepts Utilized:

1.**Tenant Management:**

* **Tenant Information**: Store and manage tenant details like business name, category, lease terms, contact information, and status (active, inactive, or in negotiation).
* **Lease Tracking**: Track lease start and end dates, rent payments, and renewal reminders.
* **Tenant Payments**: Record payments made by tenants and generate payment reminders for overdue rent.

2.**Customer Management:**

* **Customer Profiles**: Maintain detailed customer profiles, including their contact information, purchase history, preferences, and feedback.
* **Loyalty Program**: Track customer loyalty points and offer rewards for frequent shoppers.
* **Marketing Campaigns**: Run targeted marketing campaigns and send promotional emails or SMS to customers based on their preferences and behavior.

3.**Sales and Revenue Tracking:**

**Sales Dashboard**: Provide real-time data on sales across different stores within the mall. Track individual store performance and compare revenue growth over time.

* **Revenue Reports**: Generate detailed reports to track the overall financial performance of the mall, including rent, customer purchases, and advertising revenues.

4.**Customer Feedback and Support:**

* **Service Requests**: Allow customers to log service requests or complaints (e.g., maintenance, lost items).
* **Feedback Management**: Capture customer feedback through surveys or rating systems and track resolution times for service requests.
* **Customer Support Tickets**: Create and manage customer support tickets, assign to relevant staff, and track their status.

5.**Event Management:**

* **Event Calendar**: Schedule and manage mall events like sales promotions, product launches, or holiday specials.
* **Event Registration**: Allow customers to register for events, providing event details and ticketing.
* **Promotion Tracking**: Track the success of events and campaigns, including foot traffic and sales during the event.

6.**Reporting and Analytics:**

* **Customer Insights**: Use data to understand customer preferences, demographics, and spending patterns to drive future marketing and sales strategies.
* **Tenant Reports**: Generate reports related to tenant occupancy, revenue sharing, and lease agreements.
* **Financial Reports**: Generate monthly, quarterly, and annual financial statements for mall management.

##### Detailed Steps to Solution Design:

### 

### ****Data Model Design****

The **data model** defines how different entities are represented and how they are interrelated within the CRM system. For the mall management system, we need to model entities such as **tenants**, **customers**, **events**, **sales**, and **service requests**.

#### ****Key Data Entities****

1. **Tenant Management**:
   * **Tenant\_\_c (Custom Object)**: Represents tenants leasing space within the mall.
     + Fields:
       - Tenant Name (Text)
       - Business Type (Picklist) — E.g., Retail, Food Court, Service, etc.
       - Lease Start Date (Date)
       - Lease End Date (Date)
       - Rent Amount (Currency)
       - Payment Status (Picklist) — E.g., Paid, Pending, Overdue
       - Tenant Contact Info (Phone, Email)
   * **Tenant\_Payment\_\_c (Custom Object)**: Tracks payment details for each tenant.
     + Fields:
       - Payment Amount (Currency)
       - Payment Date (Date)
       - Payment Status (Picklist) — E.g., Paid, Pending, Late
       - Tenant (Lookup to Tenant\_\_c)
2. **Customer Management**:
   * **Customer\_\_c (Custom Object)**: Stores customer details.
     + Fields:
       - First Name (Text)
       - Last Name (Text)
       - Email (Email)
       - Phone (Phone)
       - Loyalty Points (Number)
       - Customer Preferences (Text) — E.g., preferred stores, event types.
   * **Customer\_Feedback\_\_c (Custom Object)**: Records feedback submitted by customers.
     + Fields:
       - Rating (Number)
       - Feedback Text (Long Text)
       - Date Submitted (Date)
       - Customer (Lookup to Customer\_\_c)
   * **Customer\_Purchase\_\_c (Custom Object)**: Tracks purchases made by customers.
     + Fields:
       - Product Name (Text)
       - Amount Spent (Currency)
       - Purchase Date (Date)
       - Customer (Lookup to Customer\_\_c)
3. **Event Management**:
   * **Mall\_Event\_\_c (Custom Object)**: Represents mall events such as sales, community activities, etc.
     + Fields:
       - Event Name (Text)
       - Event Date (Date)
       - Location (Text)
       - Event Type (Picklist) — E.g., Sale, Promotion, etc.
       - Event Status (Picklist) — E.g., Planned, Ongoing, Completed
   * **Event\_Registration\_\_c (Custom Object)**: Manages event registrations by customers.
     + Fields:
       - Customer (Lookup to Customer\_\_c)
       - Event (Lookup to Mall\_Event\_\_c)
       - Registration Date (Date)
       - Registration Status (Picklist) — E.g., Confirmed, Pending, Canceled
4. **Service Requests**:
   * **Service\_Request\_\_c (Custom Object)**: Records service requests made by customers.
     + Fields:
       - Request Type (Picklist) — E.g., Maintenance, Customer Support
       - Request Status (Picklist) — E.g., Open, In Progress, Resolved
       - Date Submitted (Date)
       - Customer (Lookup to Customer\_\_c)

#### ****Relationships and Data Integrity****

* **Tenant to Payments**: One-to-many relationship between Tenant\_\_c and Tenant\_Payment\_\_c.
* **Customer to Purchases**: One-to-many relationship between Customer\_\_c and Customer\_Purchase\_\_c.
* **Customer to Feedback**: One-to-many relationship between Customer\_\_c and Customer\_Feedback\_\_c.
* **Event to Registrations**: One-to-many relationship between Mall\_Event\_\_c and Event\_Registration\_\_c.
* **Customer to Service Requests**: One-to-many relationship between Customer\_\_c and Service\_Request\_\_c.

### ****User Interface Design****

The **User Interface (UI)** should be intuitive, responsive, and provide distinct experiences for mall administrators, tenants, and customers.

#### ****Admin Interface****

* **Tenant Management Dashboard**:
  + List view of all tenants with their key metrics (lease dates, payment status, business type).
  + A dynamic list showing tenants who have overdue payments or are nearing lease expiration.
  + A "Tenant Detail Page" with detailed information on lease status, payment history, and tenant contact.
* **Customer Management Dashboard**:
  + A list of customers with filtering options (e.g., loyalty points, registration status).
  + View customer feedback, event participation, and their purchase history.
* **Event Management**:
  + Calendar view showing all upcoming and past events, with links to view or edit events.
  + A dashboard showing the status of event registrations and participation analytics.
* **Sales and Revenue Dashboard**:
  + Visualizations for mall-wide sales, tenant sales, and revenue metrics.
  + Option to generate detailed sales reports by tenant, time period, or event.

#### ****Tenant Interface****

* **Tenant Dashboard**:
  + Personalized dashboard showing sales performance, overdue payments, and lease milestones.
  + Key indicators for performance — e.g., current month's sales, pending tasks, upcoming renewals.
* **Payment and Lease Management**:
  + A page to track payment status, make payments, and view invoices.
  + A reminder system that alerts tenants about upcoming payments or lease expirations.
* **Event Participation Page**:
  + A list of upcoming mall events that tenants can register for.
  + Event registration tracking and participation insights.

#### ****Customer Interface****

* **Customer Portal**:
  + A home page showing personalized promotions, loyalty points, and event recommendations.
  + Customers can view past purchases, submit feedback, and manage preferences.
* **Event Registration**:
  + A dedicated page for browsing and registering for upcoming events.
  + Customers can see event details, dates, and register online.
* **Feedback and Support**:
  + A feedback form for customers to submit their experience or issues, tied to a service request.

### ****Business Logic****

The **Business Logic** layer defines the rules, calculations, and actions that automate processes within the CRM application. It can be implemented using **Apex code**, **Process Builder**, **Flows**, and **Scheduled Jobs**.

#### ****Apex Code for Custom Logic****

* **Tenant Lease Expiry Notification**:
  + **Apex Trigger**: Automatically sends a notification to mall administrators when a tenant's lease is nearing expiry.
  + **Apex Class**: Calculates upcoming lease renewals and generates reminders.
* **Customer Feedback Processing**:
  + **Apex Trigger**: Automatically creates a service request when a customer submits negative feedback.
  + **Apex Class**: Analyzes feedback data to determine if follow-up actions are needed (e.g., send a thank-you note or escalate a complaint).
* **Event Registration Management**:
  + **Apex Trigger**: Automatically registers a customer for an event when they sign up, and updates Event\_Registration\_\_c with confirmation details.
  + **Apex Class**: Validates the customer’s eligibility for specific events (e.g., loyalty program points, event capacity).

#### ****Automation with Process Builder and Flows****

* **Tenant Payment Reminders**:
  + **Process Builder**: Sends automatic email reminders to tenants who have pending payments or overdue balances.
  + **Scheduled Flows**: Set up a scheduled flow to run every night to check for overdue payments and trigger reminders.
* **Customer Loyalty Program**:
  + **Flow**: Automatically assigns loyalty points to customers based on specific actions like purchases or event attendance.
  + **Process Builder**: Triggers email notifications to customers when they achieve a loyalty milestone.
* **Service Request Automation**:
  + **Flow**: Automatically assigns service requests to mall support teams based on customer feedback or service type.
  + **Process Builder**: Sends updates to customers when their service request status changes (e.g., resolved, in progress).

#### ****Validation and Workflow Rules****

* **Payment Validation**:
  + Use **workflow rules** to validate if the Tenant\_Payment\_\_c object is properly created when a payment is made and to ensure correct amounts are processed.
* **Customer Data Validation**:
  + Ensure customer email, phone numbers, and other critical data are validated to prevent duplicates and incorrect entries.
* **Event Registration Capacity**:
  + A **validation rule** ensures that an event does not exceed the maximum number of registrants.

#### ****Reporting and Analytics****

* **Sales Performance Reports**:
  + Use **Salesforce Reports** to create dashboards and reports for mall administrators, such as:
    - Total sales per tenant
    - Total sales per month or quarter
    - Top-performing tenants
* **Customer Feedback Reports**:
  + Create **Reports** to analyze customer feedback trends, sentiment, and identify areas for improvement

##### 

Create custom object

### Create Tenant Object

From the setup page ==> Click on Object Manager ==> Click on Create ==> Click on Custom Object.

1. Enter the label name:  Tenant

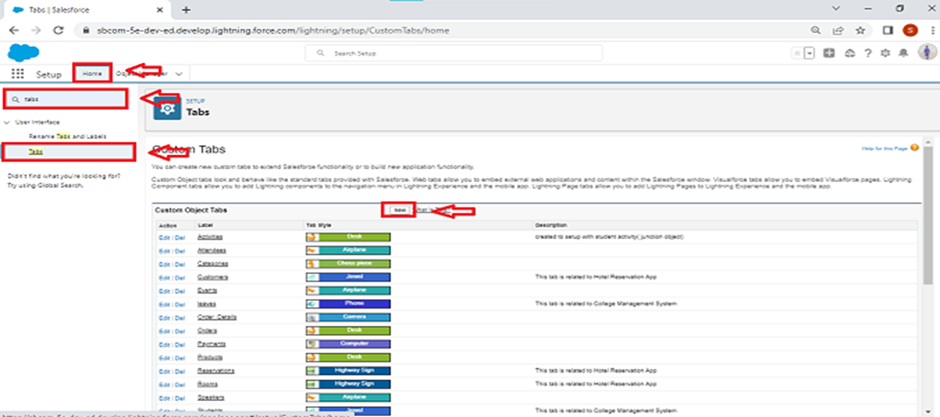
2. Plural label name: Tenants

3. Enter Record Name Label and Format

* Record Name : Tenant Name
* Data Type: Text

4. Click on Allow reports.

5. Allow search and Save



### Create Lease Tracking Object

1. Enter the label name ==>Lease Tracking

1. Plural label name ==> Leases Tracking

1. Enter Record Name Label and Format

* Record Name : Lease Tracking No
* Data Type : Auto Number
* Display Format - TT - {000000}

4. Click on Allow reports.

1. Allow search and Save

### 

### Create Tenant Issues Object

1. Enter the label name ==> Tenant Issue

1. Plural label name ==> Tenant Issues

1. Enter Record Name Label and Format

* Record Name :  Issues
* Data Type : Auto number

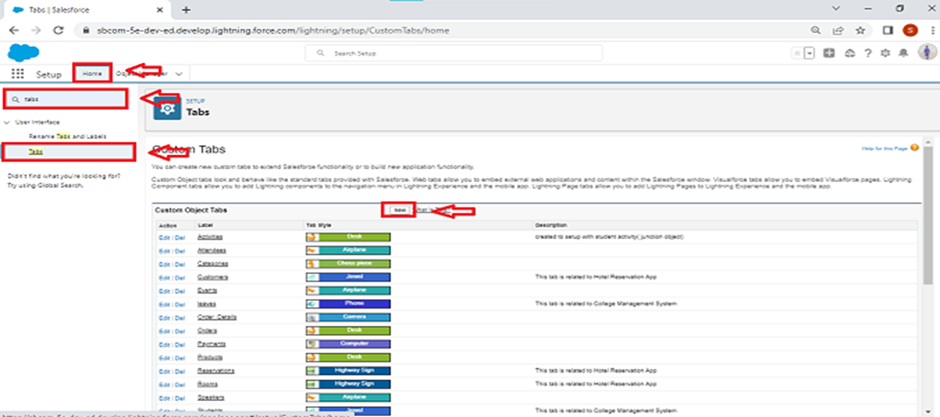
1. Click on Allow reports.

1. Allow search and Save

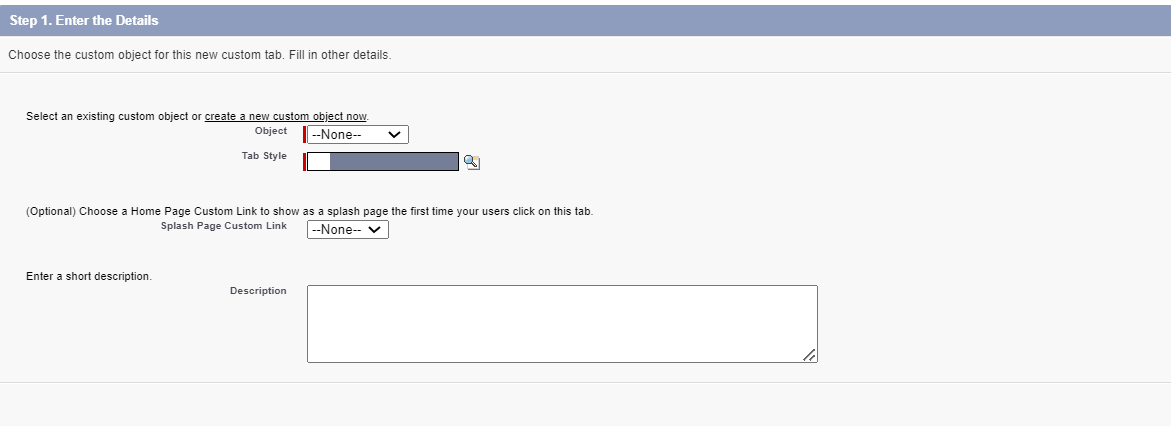
  

### Create a custom tab for tenant object

1. Go to setup page ==> type Tabs in Quick Find bar ?==>click on tabs ==> New (under custom object tab)



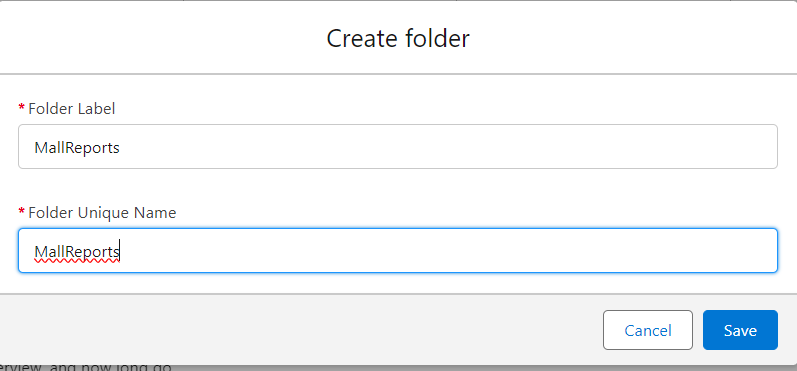
1. Select Object(Tenant) ==> Select the tab style ==> Next (Add to profiles page) keep it as default ==> Next (Add to Custom App)  uncheck the include tab ==>  Save.
2. Make sure to append tab to users' existing personal customizations is checked.



### Create Reports And Dashboards

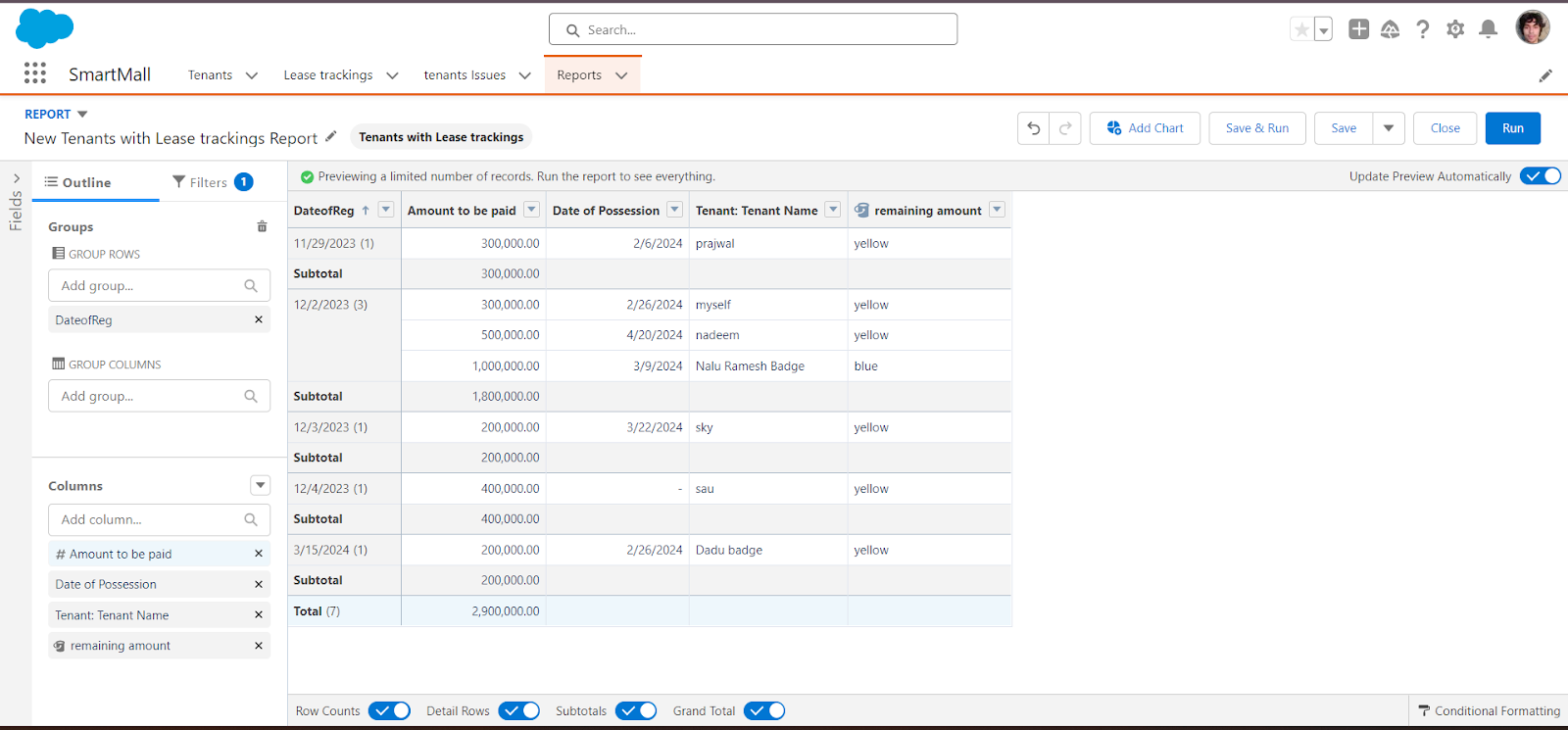
### Create a Report of lease Management Records

The Manager needs a report which shows the tenant and their joining date and their Remaining payment details and also group this by date of Registration, and make a bucket list of remaining amount as greater than 1000000 as red, less than 1000 and greater than 500000 as blue and less than equal to 500000 as yellow.

1) Create a new Folder and name it as MallReports

2) Click On new report ==> Select object Activities with LeaseTracking ==> Click on start report

3) Click on the Amount to be paid column and click on bucket this list and name it as Remaining amount



4) Save the report named as lease report and Save it in MallReports.

### Create fields on Lease Tracking Object

NOTE : - Fields in lease Tracking objects are as  follow below data types:

|  |  |  |
| --- | --- | --- |
| S No | Field Label | Data Type |
| 1 | Related Tenant | Master Detail Relationship  (Related to - Tenant) |
| 2 | Date of Possession | Date |
| 3 | End Date of Possession | Date |
| 4 | Total Year of Contract | Number |
| 5 | Total rent(Yearly) | Number |
| 6 | Amount Paid | Number |
| 7 | Amount to be paid | Formula field  (Total Rent - Amount Paid)  Return Data Type- Number |

6] Enter the Error Message as “Enter a date after 60 days ”, select the Error location as Field and select the field as “DateofPossession”, and click Save.

### Create Validation Rule For Lease Tracking Object :

A) Create Validation rule on Date of Possession-

Note:- check if the Date of Possession is after 60 days from today or not if not then show error.

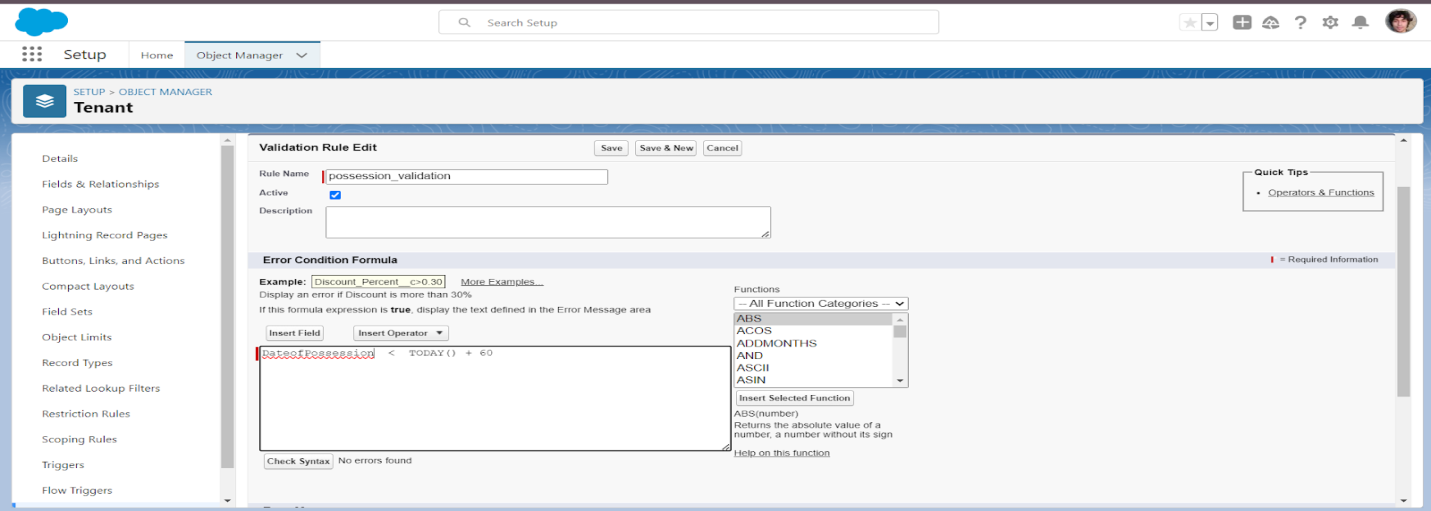
1] Go to setup ==> click on Object Manager ==> type object name(Lease tracking) in quick find bar==>click on the object.

2] Click on the validation rule ==> click New.

3] Enter the Rule name as “Possession Validation“.

4] Insert the Error Condition Formula as : -

5] Formula :   Date of Possession  <   TODAY() + 60



6] Enter the Error Message as “Enter a date after 60 days ”, select the Error location as Field and select the field as “DateofPossession”, and click Save.

### Create fields on Tenant Issues

|  |  |  |
| --- | --- | --- |
| S No | Field Label | Data Type |
| 1 | Related tenant | Master Detail Relationship  (Related to - Tenant) |
| 2 | Issue Related to | Multi Select Picklist       1)  ELECTRICITY     2)  INFRASTRUCTURE     3)  PLUMBING     4)  RENT     5)  OTHER |
| 3 | Subject | Text Are (long) |
| 4 | Phone Number | Number |
| 5 | Status | Pick List     1. Not contacted 2. Open 3. In progress 4. Working 5. closed |
| 6 | Priority | Picklist     1. Low 2. Medium 3. High |
| 7 | Origin | Picklist     1. Phone 2. Mail 3. Web |
| 8 | Email id | Email |
| 9 | Date | Date     (Default Value - Today()) |

### 

### Create the Lightning App

An app is a collection of items that work together to serve a particular function. In Lightning Experience, Lightning apps gives users access to sets of objects, tabs, and other items all in one convenient bundle in the navigation bar.

Lightning apps let you brand your apps with a custom color and logo. You can even include a utility bar and Lightning page tabs in your Lightning app. Members of your org can work more efficiently by easily switching between apps.

There are two types of Salesforce Applications:

* Standard Apps
* Custom Apps

Standard Apps:

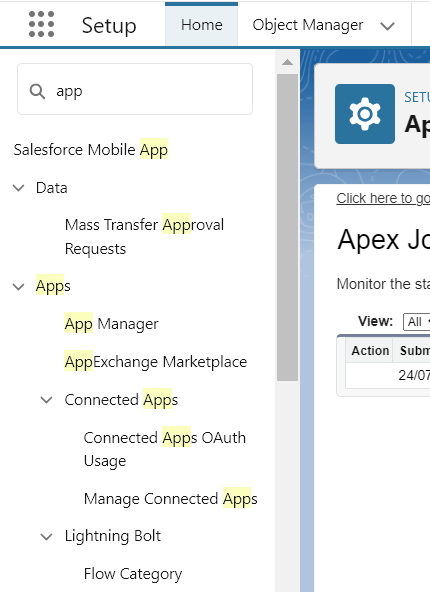
Standard apps come with every occurrence of Salesforce as default. Community, Call Center, Content, Sales, Marketing, Salesforce Chatter, Site.com, and App Launcher are included in these apps. The description, logo, and label of a standard app cannot be altered.

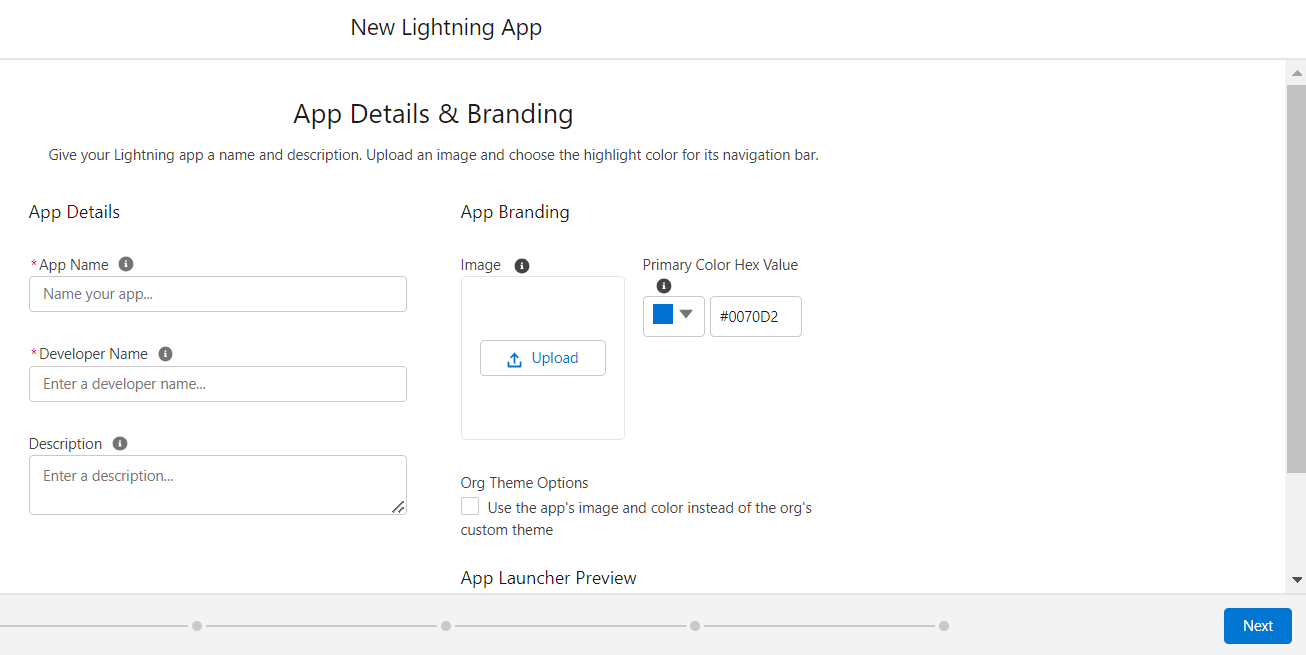
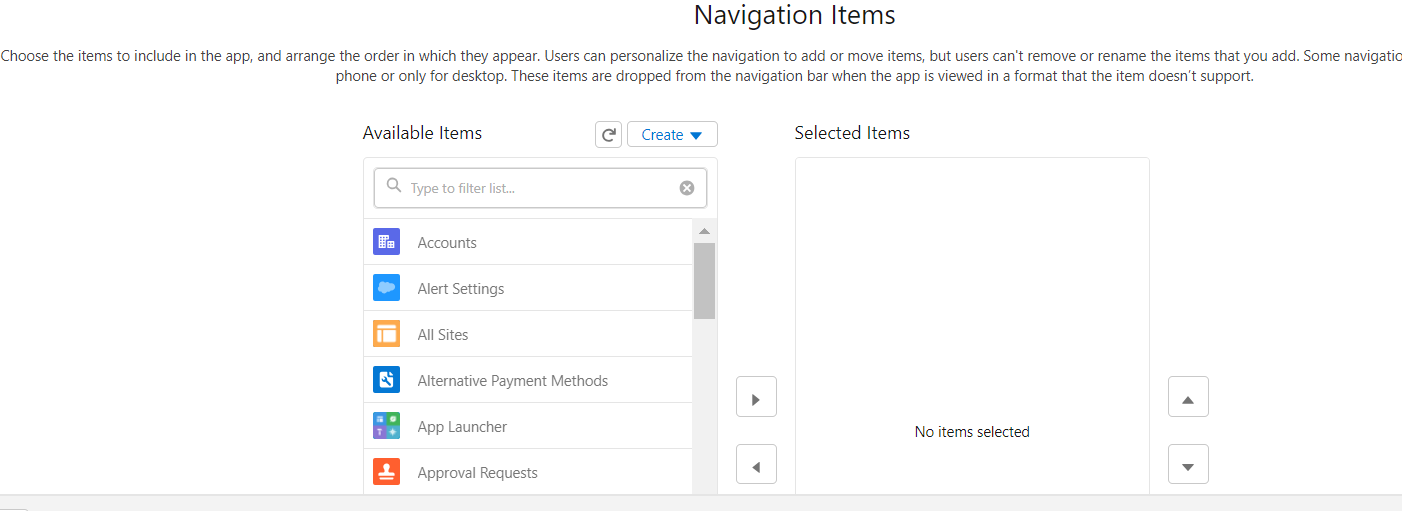
Custom Apps:

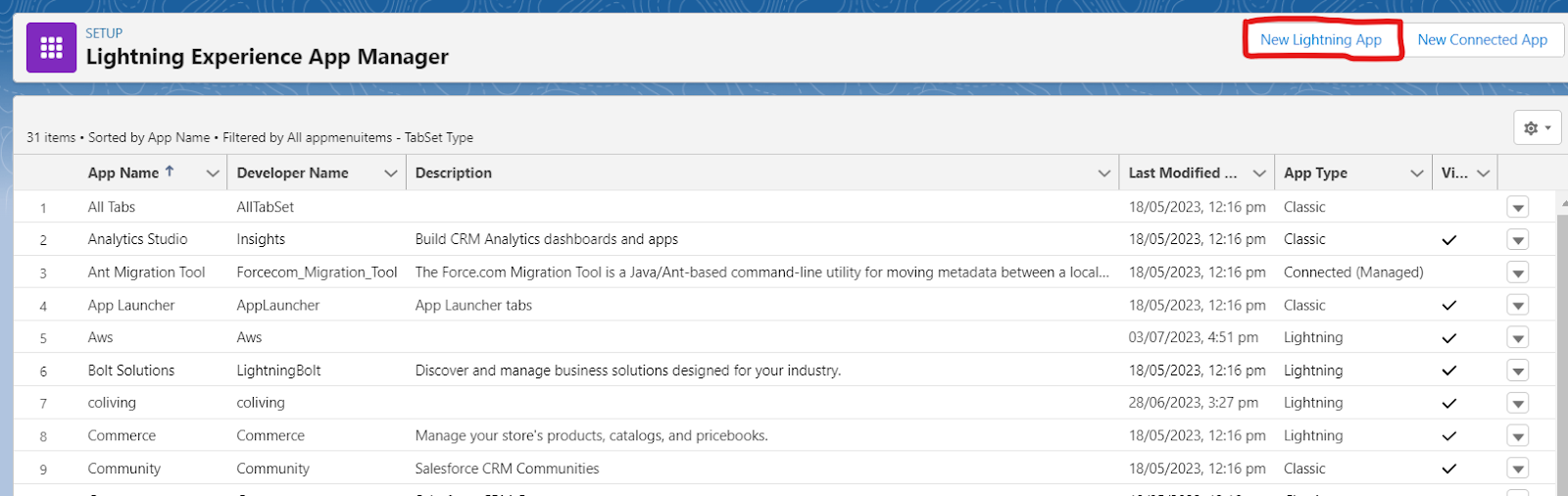
Custom apps are created according to the needs of a company. They can be made by putting custom and standard tabs together. Logos for custom apps can be changed.

### Steps to create custom app in salesforce

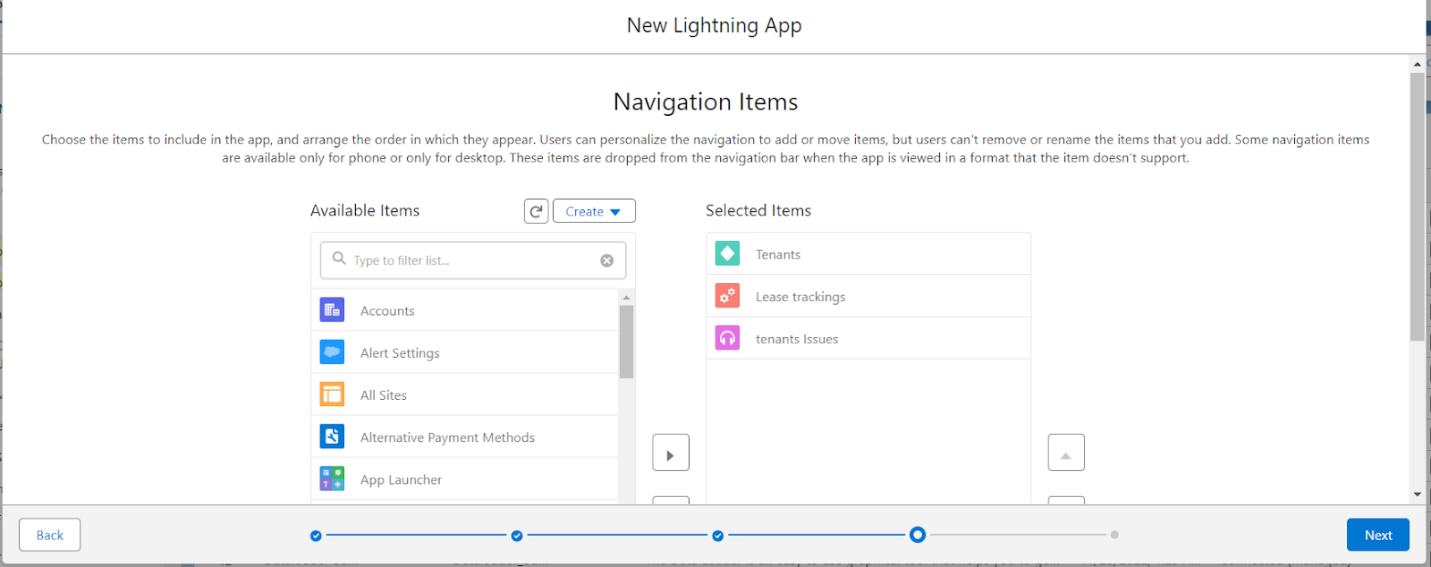
1. Go to setup, by clicking the gear icon present in the top right corner.
2. Navigate to the Home bar and in the quick find box, search for App.



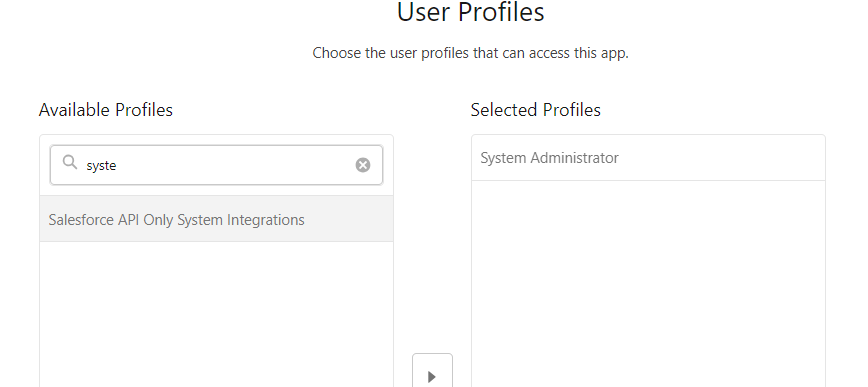
1. Click on APP MANAGER.
2. You can notice the screen like this. Now click on New Lightning App.You will find like this below.
3. Enter the App name( Here we entered ‘SmartMall’),the developer name gets automatically populated. If an image is required, you can browse the image and upload it.
4. Click Next, Next and you can see a Navigation Items window like this:



1. In the filter list, enter Tenant , Lease Tracking, Tenant issues and move them in the Selected items from Available items.



1. Click on Next , and you can see User Profiles. This option is used when we want only certain profiles to access them.
2. Enter System Administrator in the filter box and add the system Administrator to the selected profile list.

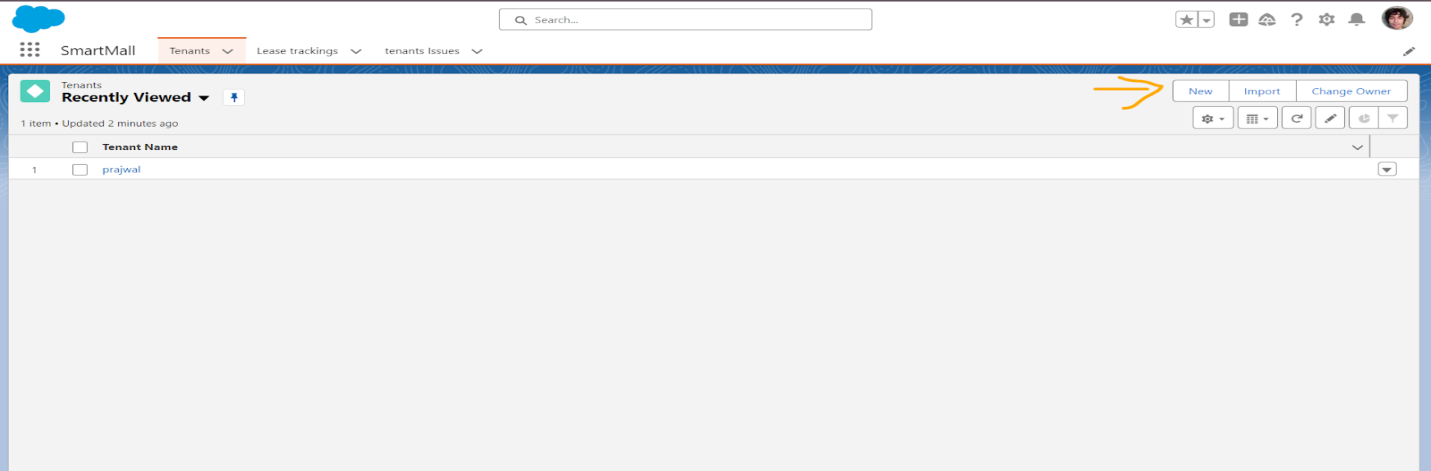


1. Click on Save and Finish.
2. Now navigate to the App launcher and search for SmartMall and you can find the SmartMall app.

### Record Insertion

### Inserting Records in Tenant Object

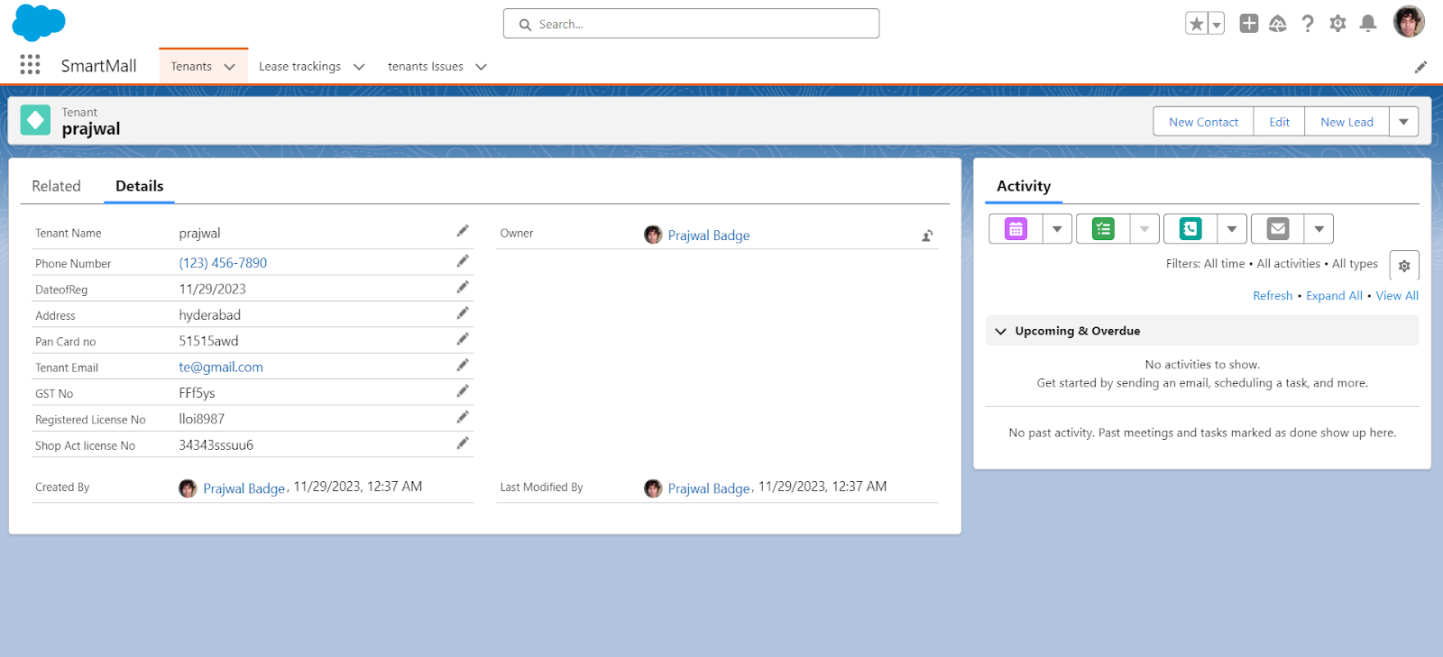
1)  Click on the App Launcher and search Tenant Object then click New in the right corner to create a record.



2) Fill every field with valid data, especially the fields on which you have created a validation rule.

3) If you Enter Phone Number more or less than 10 digits it will show an error.

4) Similarly, if you enter DateofReg a Past date it will show an error. 5)  After creating a record the page will look like this



###### Fields & Relationships

When we talk about Salesforce, Fields represent the data stored in the columns of a relational database.

It can hold any valuable information that you require for a specific object. Hence, the overall searching, deletion, and editing of the records become simpler and quicker.

###### Types of Fields

* + 1. Standard Fields
    2. Custom Fields

As the name suggests, the Standard Fields are the predefined fields in Salesforce that perform a standard task. The main point is that you can’t simply delete a Standard Field until it is a non-required standard field. Otherwise, users have the option to delete them at any point from the application freely. Moreover, we have some fields that you will find common in every Salesforce application. They are,

>>Created By

>> Owner

>> Last Modified

>> Field Made During object Creation

###### Custom Fields:

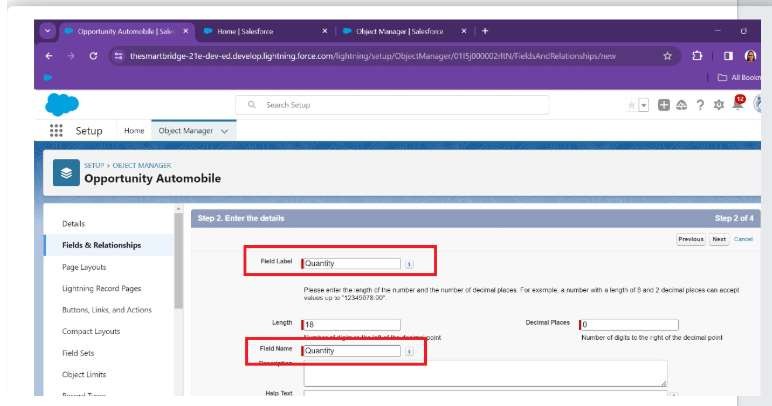
On the other side of the coin, Custom Fields are highly flexible, and users can change them according to requirements. Moreover, each organizer or company can use them if necessary. It means you need not always include them in the records, unlike Standard fields. Hence, the final decision depends on the user, and he can add/remove Custom Fields of any given form.

### Creating Opportunity Master Detail Relationship Field in Opportunity AutoMobile Object:

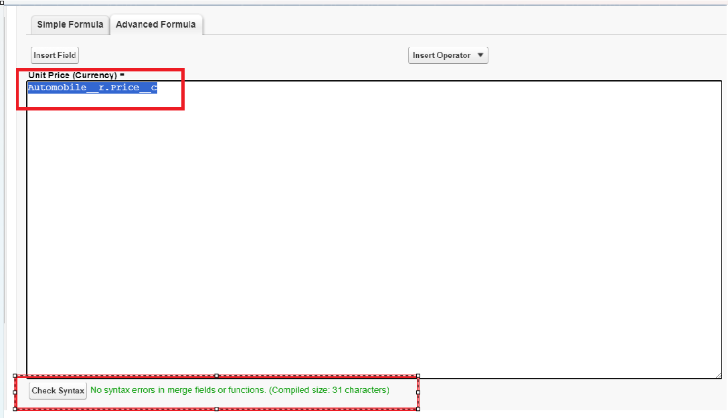
##### Creating the AutoMobile Information Lookup Field in Opportunity Automobile Object:

**Creating Quantity Number Field in Opportunity Automobile Object**

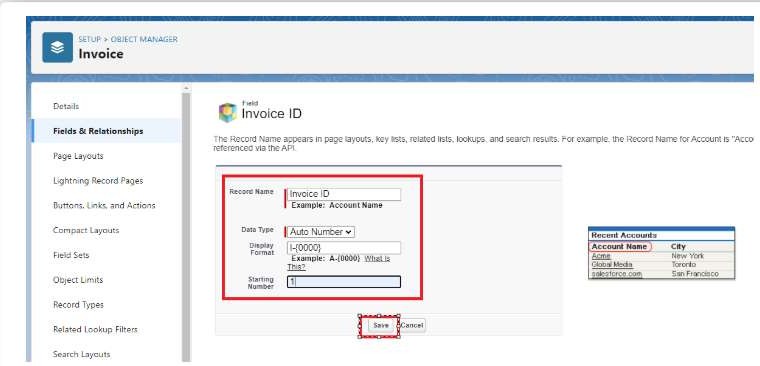


##### Creating Formula Field in Opportunity Automobile Object

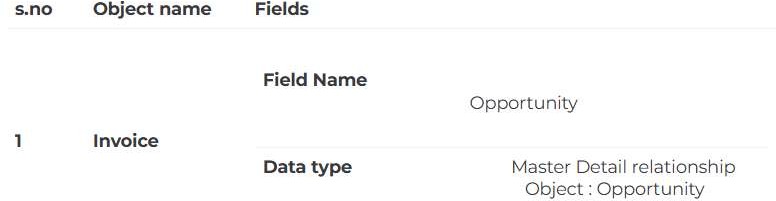


**Updating field in Invoice Object**



**Creating Remaining Fields in Objects**



### Page Layouts:

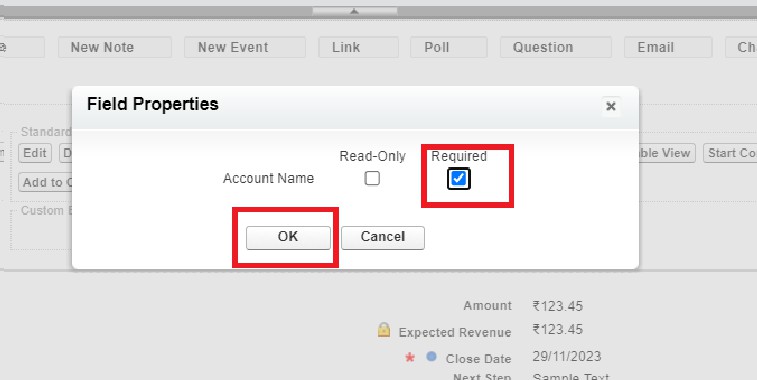
Page Layout in Salesforce allows us to customize the design and organize detail and edit pages of records in Salesforce. Page layouts can be used to control the appearance of fields, related lists, and custom links on standard and custom objects' detail and edit pages.

##### Edit the Page layout for Opportunity Object

Step 1: Go to Setup >> Click on Object Manager >> On the search bar, select Opportunity Layout. You can notice Page Layouts on the left panel

Step 2: Click on Page Layouts, Click on ‘Opportunity Layouts’.

Step 3: In the Opportunity Detail Section, you can see various fields. Go on Account And Click on that Properties icon of Account name Field.



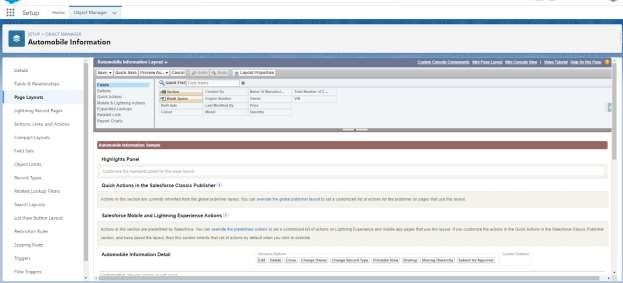
Step 4: check the Required box for Account name and click on Ok. Step 5: Click on Save.

##### Edit the Page layout for Automobiles Information

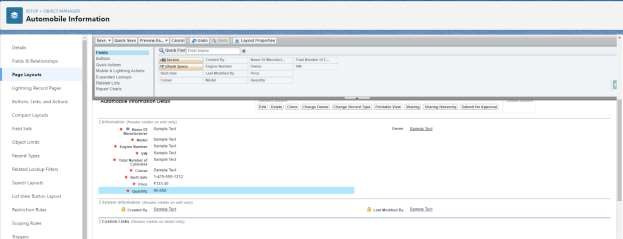
Step 1: Go to Setup >> Click on Object Manager >> On the search bar, select Automobile Information. You can notice Page Layouts on the left panel

Step 2: Click on Page Layouts. Click on ‘Automobile Information Layout’.

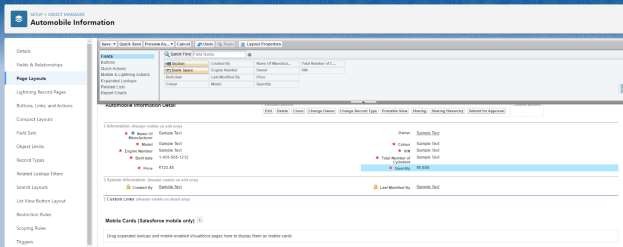


Step 3: Just Go for each one field of Automobile Information Object, Click on Gear Icon and mark as Required just as Done for Above Account Object. After required is done it will show the red color as given in below image.





Step 4 : Adjust the Fields as given below for A good looking view.



Step 5 : Click on Save.

### Apex Trigger

Apex can be invoked by using triggers. Apex triggers enable you to perform custom actions before or after changes to Salesforce records, such as insertions, updates, or deletions.

A trigger is Apex code that executes before or after the following types of operations: insert

update delete merge upsert undelete

For example, you can have a trigger run before an object's records are inserted into the database, after records have been deleted, or even after a record is restored from the Recycle Bin.

You can define triggers for top-level standard objects that support triggers, such as a Contact or an Account, some standard child objects, such as a CaseComment, and custom objects. To define a trigger, from the object management settings for the object whose triggers you want to access, go to Triggers.

There are primarily two types of Apex Triggers:

**Before Trigger:** This type of trigger in Salesforce is used either to update or validate the values of a record before they can be saved into the database. So, basically, the before trigger validates the record first and then saves it. Some criteria or code can be set to check data before it gets ready to be inserted into the database.

**After Trigger:** This type of trigger in Salesforce is used to access the field values set by the system and affect any change in the record. In other words, the after trigger makes changes to the value from the data inserted in some other record.

##### Opportunity Automobile quantity

###### Code:

public class OpportunityHandlerClass {

public static void opportunityAutomobileQuantity(List<Opportunity> LstOpportunity, Map<Id,Opportunity> OldMapOpportunity){

set<Id> opportunityIds = new set<Id>(); for(Opportunity opp : LstOpportunity){

if(opp.StageName =='Closed Won' ){ opportunityIds.add(opp.Id);

}

}

Map<Id,Opportunity\_Automobile c> lstOpportunityAutomobile =new Map<Id,Opportunity\_Automobile c>([SELECT Id, Opportunity c, Automobile c, Quantity c, Unit\_Price c, Total\_Price c FROM Opportunity\_Automobile c Where Opportunity c IN: opportunityIds]);

set<Id> AutoInformationIds = new set<Id>();

for(Opportunity\_Automobile c OppAuto: lstOpportunityAutomobile.values()){ if(OppAuto.Automobile c != null){

AutoInformationIds.add(OppAuto.Automobile c);

}

}

List<Automobile\_Information c> lstAutomobileInfomation = new List<Automobile\_Information c>(); Map<Id,Automobile\_Information c> MapAutomobileInformation = New

Map<Id,Automobile\_Information c>([SELECT Quantity c, Price c, Name, Id FROM Automobile\_Information c WHERE Id IN: AutoInformationIds]);

For(Opportunity\_Automobile c AutoOpp : lstOpportunityAutomobile.Values()){ decimal num = 0;

if(AutoOpp.Automobile c == MapAutomobileInformation.get(AutoOpp.Automobile c).Id && OldMapOpportunity.get(AutoOpp.Opportunity c).stagename != 'Closed Won'){

num = MapAutomobileInformation.get(AutoOpp.Automobile c).Quantity c- AutoOpp.Quantity c;

MapAutomobileInformation.get(AutoOpp.Automobile c).quantity c = num; lstAutomobileInfomation.add(MapAutomobileInformation.get(AutoOpp.Automobile c));

}

}

If(!lstAutomobileInfomation.IsEmpty()){ update lstAutomobileInfomation;

}

}

}

###### Trigger Handler :

trigger OpportunityTrigger on Opportunity (before update, After Update) { if(trigger.isbefore && trigger.isUpdate){

OpportunityHandlerClass.opportunityAutomobileQuantity(trigger.new, trigger.oldMap);

}

}

##### Opportunity-Automobile Error

public class OpportunityAutomobileHandler {

public static void quantityErrorOnAutomobileInformation(List<Opportunity\_Automobile c> lstOpportunityAutomobile){

set<Id> AutomobileIds = new Set<Id>();

For(Opportunity\_Automobile c OppAutomobile : lstOpportunityAutomobile){ if(oppAutomobile.Automobile c != null){

AutomobileIds.add(oppAutomobile.Automobile c);

}

}

Map<Id,Automobile\_Information c> lstAutomobileInformation = new map<Id,Automobile\_Information c>([SELECT Id, CreatedById, Quantity c, Price c FROM Automobile\_Information c WHERE Id IN: AutomobileIds]);

For(Opportunity\_Automobile c OppAutomobile : lstOpportunityAutomobile){

If(OppAutomobile.Automobile c == lstAutomobileInformation.get(OppAutomobile.Automobile c).Id && lstAutomobileInformation.get(OppAutomobile.Automobile c).Quantity c <

OppAutomobile.Quantity c){

OppAutomobile.addError('the Number of Automobile u want are not Available !! the Automobile are Available Count is ' + .get(OppAutomobile.Automobile c).Quantity c );

}

}

}

##### } Trigger Handler :

trigger OpportunityAutoMobileTrigger on Opportunity\_Automobile c (before insert, before Update) { if(trigger.isbefore && trigger.isinsert || trigger.isupdate){

OpportunityAutomobileHandler.quantityErrorOnAutomobileInformation(trigger.new);

}

}

##### Invoice Creation Trigger

public class InvoiceCreation {

public static void OpportunityClosedwonInvoiceGeneration(List<Opportunity> lstOpportunity, Map<Id,Opportunity>OldMapOpportunity){

set<Id> oppIds = new Set<Id>(); For(Opportunity opp : lstOpportunity){

if(Opp.StageName == 'Closed Won' && OldMapOpportunity.get(opp.Id).StageName != opp.StageName){

oppIds.add(opp.Id);

}

}

List<Opportunity\_Automobile c> lstOpportunityAutomobile = [SELECT Unit\_Price c, Total\_Price c, Automobile c, Quantity c,Opportunity c, Id FROM Opportunity\_Automobile c WHERE Opportunity c IN: oppIds];

List<Invoice c> lstInvoice = new List<Invoice c>(); For(Opportunity\_Automobile c oppAuto : lstOpportunityAutomobile){

Invoice c i = new Invoice c(); i.Quantity c = oppAuto.Quantity c; i.Unit\_Price c = oppAuto.Unit\_Price c;

i.Total\_Price c = oppAuto.Total\_Price c; i.Purchase\_Date c = date.today(); i.Opportunity c = oppAuto.Opportunity c; lstInvoice.add(i);

}

if(!lstInvoice.isempty()){ insert lstInvoice;

}

}

##### Trigger Handler :

trigger OpportunityTrigger on Opportunity (before update, After Update) { if(trigger.isbefore && trigger.isUpdate){

OpportunityHandlerClass.opportunityAutomobileQuantity(trigger.new, trigger.oldMap);

}

IF(trigger.isafter && trigger.isupdate){ InvoiceCreation.OpportunityClosedwonInvoiceGeneration(trigger.new, trigger.oldMap);

}

}

##### Check contact role:

###### Trigger:

public class ContactRoleCheck {

public static void CheckcontactRoleonOpportunity(List<Opportunity> lstOpportunity, Map<Id,Opportunity>OldMapOpportunity){

List<OpportunityContactRole> lstContactRole = [SELECT Id From OpportunityContactRole WHERE OpportunityId IN: OldMapOpportunity.keyset()];

For(Opportunity opp : lstOpportunity){

if(Opp.StageName == 'Closed Won' && OldMapOpportunity.get(opp.Id).StageName != opp.StageName){ If(lstContactRole.isempty()){

opp.adderror('Please add contact Role on opportunity whenever Opportunity is Going to Closed

Won.');

}

}

}

}

}

###### Trigger Handler :

trigger OpportunityTrigger on Opportunity (before update, After Update) { if(trigger.isbefore && trigger.isUpdate){

OpportunityHandlerClass.opportunityAutomobileQuantity(trigger.new, trigger.oldMap); ContactRoleCheck.CheckcontactRoleonOpportunity(trigger.new, trigger.oldMap);

}

IF(trigger.isafter && trigger.isupdate){ InvoiceCreation.OpportunityClosedwonInvoiceGeneration(trigger.new, trigger.oldMap);

}

}

##### LWC Component:

###### Create Apex Class to Get Invoices:

public class OpportunityInvoiceswithLWC { @AuraEnabled(cacheable=true)

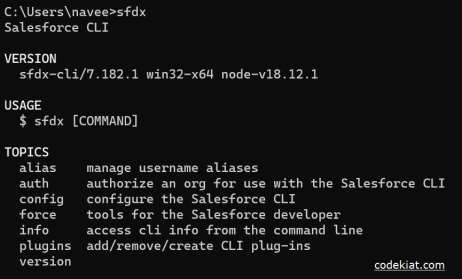
public static List<Invoice c> getInvoices(string OpportunityId){

return [SELECT Id, Quantity c, Purchase\_Date c, Opportunity c, Unit\_Price c, Total\_Price c, Name FROM Invoice c WHERE Opportunity c =: OpportunityId];

}

###### }Install Salesforce CLI:



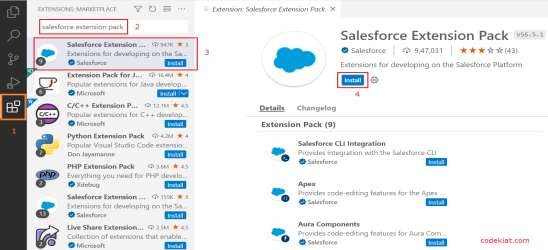
**Install Microsoft VS Code:**

VS Code, or Visual Studio Code, is a free, open-source code editor developed by Microsoft. It is a lightweight, cross-platform code editor that provides features such as debugging, Git integration, and support for a wide range of programming languages.

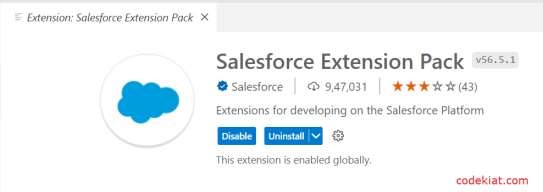
[Download the version of the software](https://code.visualstudio.com/download) that is compatible with your operating system and install it.

The following instructions are for Windows OS. Other operating systems may have slightly different steps.

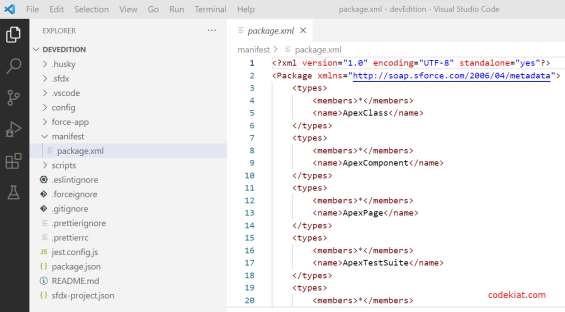
###### Install the Salesforce Extension Pack:



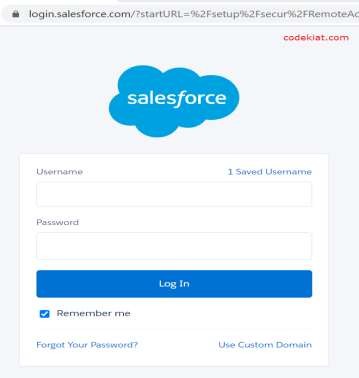


###### Create a project in VS Code:



**Authorize an org:**



###### Create Lightning Web Component:

**XML File Code:**

<?xml version="1.0" encoding="UTF-8"?>

<LightningComponentBundle xmlns="<http://soap.sforce.com/2006/04/metadata>">

<apiVersion>58.0</apiVersion>

<isExposed>true</isExposed>

<targets>

<target>lightning RecordAction</target>

<target>lightning RecordPage</target>

</targets>

</LightningComponentBundle>

###### JS File Code :

import { LightningElement, api, track, wire } from 'lwc';

import getInvoices from '@salesforce/apex/OpportunityInvoiceswithLWC.getInvoices'; export default class InvoiceOpportunity extends LightningElement {

@api recordId;

@track invoiceCollection cols = [

{label:"ID" , fieldName:'Name'},

{label:"Opportunity Id" , fieldName:'Opportunity c'},

{label:"Quantity" , fieldName:'Quantity c'},

{label:"Total Price" , fieldName:'Total\_Price c'},

{label:"Purchase Date" , fieldName:'Purchase\_Date c'}

]

@wire(getInvoices,{OpportunityId:'$recordId'}) invoicefunction({data,error}){ console.log(this.recordId +'this is record Id');

if(data){ console.log(data);

this.invoiceCollection = data

}if(error){ console.log('this is error') console.log('error');

}

}

}

###### HTML File :

<template>

<lightning-card >

<div style="text-align: center; font-size:larger;"><strong>Opportunity Ivoices</strong></div>

<lightning-datatable key-field="Id"

data={invoiceCollection} columns={cols}

></lightning-datatable>

</lightning-card>

</template>

##### Create Lightning Web Component:

###### JS File Code :

import { LightningElement, api, track, wire } from 'lwc';

import getInvoices from '@salesforce/apex/OpportunityInvoiceswithLWC.getInvoices'; export default class InvoiceOpportunity extends LightningElement {

@api recordId;

@track invoiceCollection cols = [

{label:"ID" , fieldName:'Name'},

{label:"Opportunity Id" , fieldName:'Opportunity c'},

{label:"Quantity" , fieldName:'Quantity c'},

{label:"Unit Price" , fieldName:'Unit\_Price c'},

{label:"Total Price" , fieldName:'Total\_Price c'},

{label:"Purchase Date" , fieldName:'Purchase\_Date c'}

]

@wire(getInvoices,{OpportunityId:'$recordId'}) invoicefunction({data,error}){ console.log(this.recordId +'this is record Id');

if(data){ console.log(data);

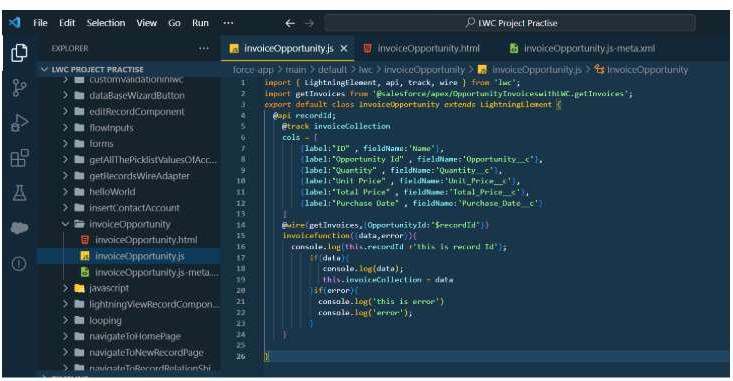
this.invoiceCollection = data

}if(error){ console.log('this is error') console.log('error');

}

}

}

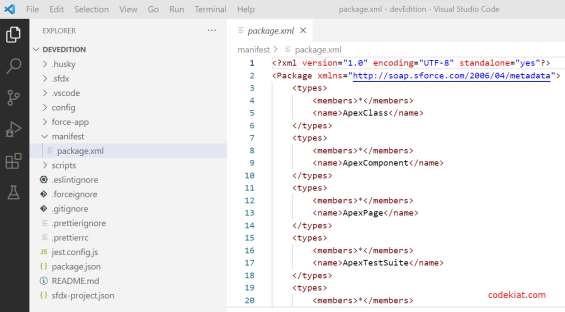


##### Create Button to Add on Opportunity

Select the InvoiceOpportunity component Label :- Invoices

Name :- Invoices



### Delete opportunity Schedule Class

##### Objective :

Through this schedulable class, we can see all the Closed Lost Opportunities.

We can delete all the Closed lost Opportunities by this Scheduled method on every monday as weekly.

1. Login to the respective account and navigate to the gear icon in the top right corner.
2. Click on the Developer console. Now you will see a new console window.
3. In the toolbar, you can see FILE. Click on it and navigate to new and create New apex class.
4. Name the class as “DeleteClosedLostOpportunities ”

#### CODE SNIPPET :

public class DeleteClosedLostOpportunities implements Schedulable{ public static void execute(SchedulableContext sc){

List<Opportunity> getLostOpportunities = [SELECT Id, Name From Opportunity Where StageName =: '

\Closed Lost' LIMIT 50000]; if(!getLostOpportunities.IsEmpty()){

Delete getLostOpportunities;

}

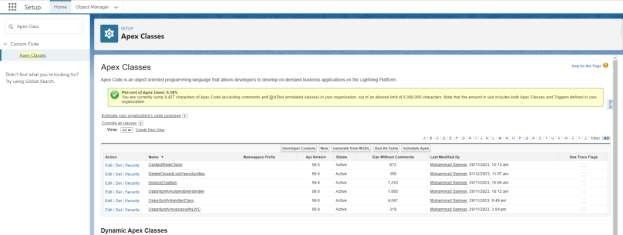
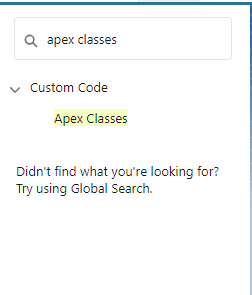
}

}

##### Schedule the Apex class:

Go to the Home page in your salesforce account.

In the search bar, enter Apex and click on Apex Classes.

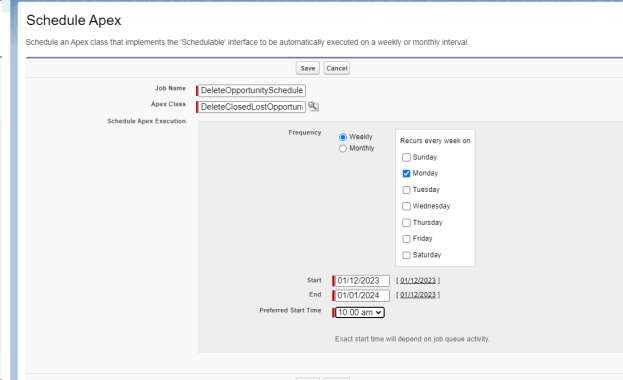




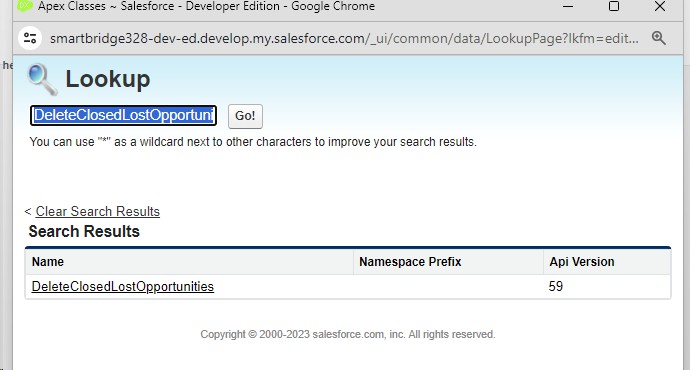
 

Click on Schedule Apex and enter the Job name.

* + Job Name : DeleteOpportunitySchedule

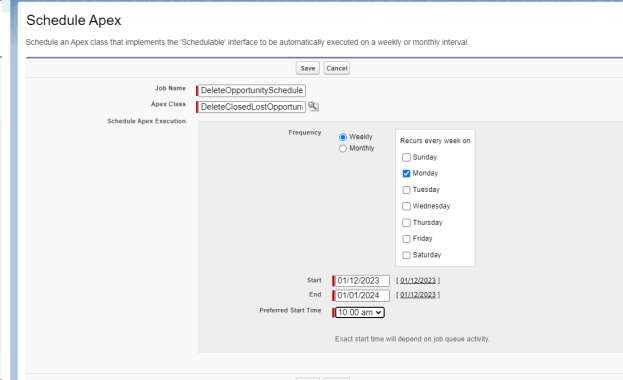


Now click on the search icon present near the Apex class : Goto the Lookup icon beside ? click on it ? select DeleteClosedLostOpportunities.

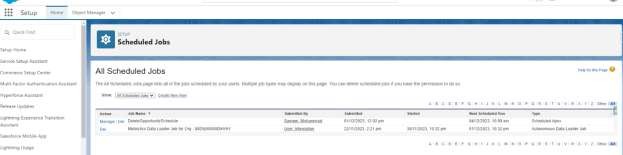


In the Schedule Apex section , select weekly and select Monday mentioned and preferred time as 10:00 AM.





Click on Save. Now enter Apex in the search box and select Apex jobs.



You can see that the batch job is in queue and will run whenever the day mentioned comes.

### Reports:

Reports give you access to your Salesforce data. You can examine your Salesforce data in almost infinite combinations, display it in easy-to-understand formats, and share the resulting insights with others.

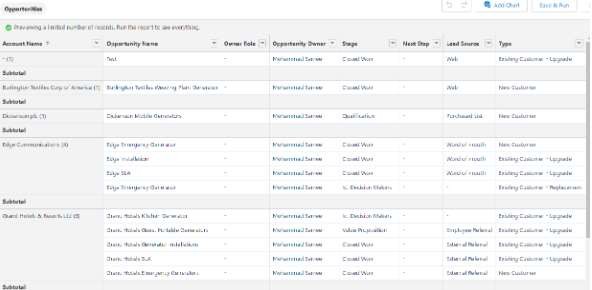
Before building, reading, and sharing reports, review these reporting basics.

##### Types of Reports in Salesforce

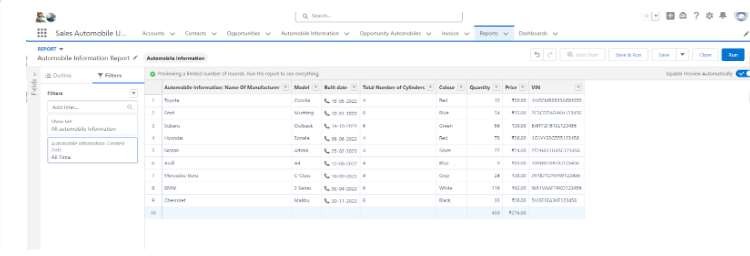
1. Tabular
2. Summary
3. Matrix
4. Joined Reports

## Create Report on Opportunity:



**Create Report on Automobile Information:**

##### Filters:-



## Dashboard:

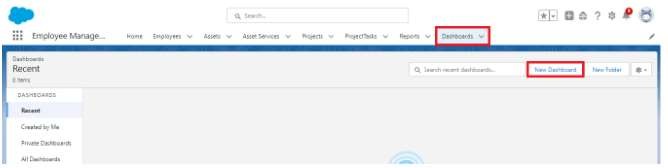
Dashboards help you visually understand changing business conditions so you can make decisions based on the real-time data you’ve gathered with reports. Use dashboards to help users identify trends, sort out quantities

, and measure the impact of their activities. Before building, reading, and sharing dashboards, review these dashboard basics.

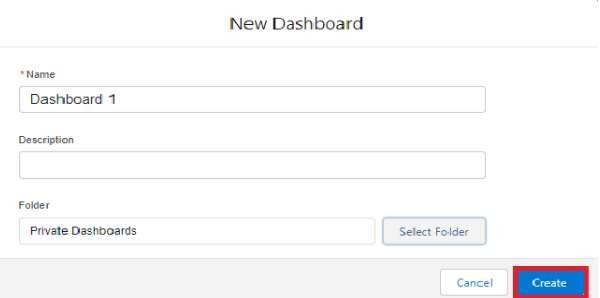
## Sales Dashboard:

##### Create Dashboard

1. Go to the app ? click on the Dashboards tabs.

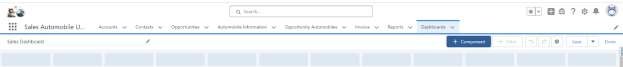


1. Give a Name and click on Create.



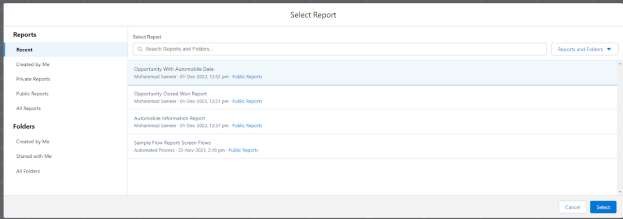
Name : Automobile Sales

1. Select add component



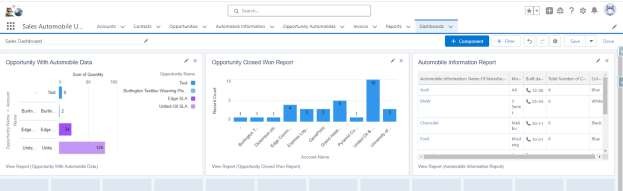
1. Select a Report and click on select.





1. Click Add then click on Save and then click on Done.

The Created Dashboard will look like this.



## Conclusion:

##### Summary of Achievements:

The Salesforce Automobile Information CRM project has successfully integrated sales and service functionalities into a single platform, improving both internal operations and customer satisfaction. The CRM system now allows seamless tracking of inventory, sales opportunities, and customer interactions while automating key processes to boost efficiency. With enhanced reporting and analytics, the organization can make data-driven decisions to grow their business and improve the customer experience. The solution is scalable, customizable, and alignswith the long-term goals of the automobile dealership, setting a foundation for future growth and digital transformation.