

# Sales Automobile Using Salesforce CRM

**Team Members:**

**Naan Mudhalvan ID:**

RAMANAN M

1613151D7C8A2CE9B714B821724D69FD

RAGULPRASAD K

7CE55BCADADV9459B5449FA36B9D490B

RAJASEKAR G

CA34C3C22F56AB7AEE648F08DC8D3392

VADIVELAN K

A1F3F32AEB8D6CE8BF740082EEBCFD66

## **1. Project Overview:**

This project involves the development and customization of Salesforce CRM for managing the entire automobile sales process. The solution will enable streamlined sales operations, from lead management to the creation of invoices. The main features will include the creation of custom objects for automobile information, invoice management, and the automation of key processes such as opportunity and invoice handling.

The objective is to enhance the sales process by leveraging Salesforce's robust tools, including creating objects, fields, custom tabs, Apex triggers, Lightning Web Components (LWC), and more. This will help to manage customer accounts, automobile data, invoices, and related sales activities efficiently.

## **2. Objective:**

The objectives for the Sales Automobile Using Salesforce CRM project are clearly defined to ensure measurable success and alignment with both business and technical goals. These objectives guide the development process and ensure the solution will meet the needs of the business, sales teams, and customers.

The business goals of the Sales Automobile Using Salesforce CRM project are to streamline and optimize the entire sales process, from lead generation to invoice creation, by leveraging Salesforce's powerful CRM capabilities. The project aims to increase sales efficiency by automating key processes, enhance customer engagement through a 360-degree view of customer data, and improve data accuracy for better decision-making. Additionally, the solution seeks to optimize inventory management by providing real-time visibility into stock levels and sales trends, accelerate

invoice processing for faster revenue realization, and drive business growth by enabling better tracking, reporting, and forecasting of sales performance.

The specific outcomes of the Sales Automobile Using Salesforce CRM project include the creation of custom objects and fields for managing automobile inventory, sales opportunities, and invoices, which will be fully integrated into the Salesforce platform. The project will deliver automation through Apex triggers and workflows to streamline sales processes, from generating invoices to tracking opportunity quantities. Custom tabs and page layouts will be developed to improve user experience, while the Salesforce Lightning App and Lightning Web Components (LWC) will enhance accessibility and interactivity across devices. Additionally, comprehensive reports and dashboards will be implemented to provide real-time insights into sales performance, inventory levels, and customer interactions. This integration and automation will result in faster, more accurate data entry, improved operational efficiency, and enhanced decision-making capabilities for sales teams and management.

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

#### ➤ **Custom Objects & Fields:**

Custom Objects like Automobile Information and Invoice store specific data related to the sales process, allowing you to track cars, pricing, and invoices in Salesforce.

#### ➤ **Page Layouts & Lightning Apps:**

Custom Page Layouts for Opportunities and Invoices ensure relevant information is easily accessible, while a Lightning App consolidates the interface for users to manage their workflow efficiently.

➤ **Apex Triggers & Classes:**

Apex Triggers automate processes such as updating the automobile quantity in an Opportunity and creating an invoice when an Opportunity is closed, reducing manual tasks.

➤ **Lightning Web Components (LWC):**

A custom LWC component displays related Invoice data on Opportunity pages, enhancing the user interface and providing real-time access to invoice details.

➤ **Reports & Dashboards:**

Reports and Dashboards provide visual insights into sales performance, inventory levels, and invoice statuses, helping track business health and sales trends.

➤ **Salesforce Lightning Experience:**

The **Lightning Experience** improves user productivity with an intuitive and responsive interface, allowing users to efficiently navigate and manage automobile sales data.

➤ **Apex Schedulers:**

Apex Schedulers automate recurring tasks, such as sending invoice reminders or syncing data, ensuring timely actions without manual intervention.

## **4. Detailed Steps to Solution Design for Automobile Sales Management in Salesforce:**

### **➤ Automobile Information Object:**

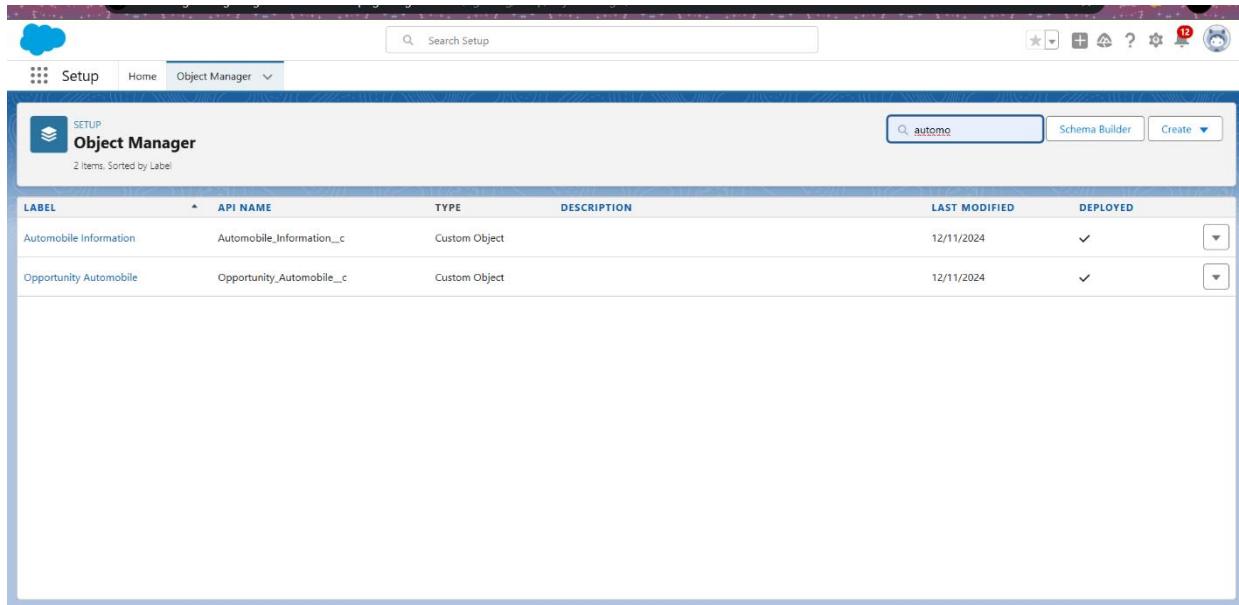
This object stores data related to the automobiles sold, such as make, model, year, price, and VIN.

### **➤ Invoice Object:**

This object holds details about invoices related to Opportunities.

### **➤ Automobile Object:**

This object may track additional attributes related to automobiles or individual transactions (e.g., specific inventory details).



The screenshot shows the Salesforce Object Manager interface. At the top, there's a navigation bar with 'Setup', 'Home', and 'Object Manager'. A search bar contains the text 'autoomo'. Below the header, a table lists two objects:

Label	API Name	Type	Description	Last Modified	Deployed
Automobile Information	Automobile_Information__c	Custom Object		12/11/2024	✓
Opportunity Automobile	Opportunity_Automobile__c	Custom Object		12/11/2024	✓

The screenshot shows the Salesforce Object Manager page. At the top, there's a search bar with 'Search Setup' and a search icon. Below it, a navigation bar has 'Setup' selected, along with 'Home' and 'Object Manager'. The main area is titled 'Object Manager' with a sub-header '5 Items, Sorted by Label'. A search bar at the top right contains 'invoice'. Below is a table with columns: LABEL, API NAME, TYPE, DESCRIPTION, LAST MODIFIED, and DEPLOYED. The data includes:

LABEL	API NAME	TYPE	DESCRIPTION	LAST MODIFIED	DEPLOYED
Credit Memo Invoice Application	CreditMemoInvApplication	Standard Object			
Invoice	Invoice_c	Custom Object		12/11/2024	✓
Invoice	Invoice	Standard Object			
Invoice Line	InvoiceLine	Standard Object			
Payment Line Invoice	PaymentLineInvoice	Standard Object			

## ➤ Creating a Custom Tab:

Custom object tabs are the user interface for custom applications that you build in salesforce.com. They look and behave like standard salesforce.com tabs such as accounts, contacts, and opportunities.

The screenshot shows the Salesforce Tabs setup page. The URL in the browser is 'mailamengineeringcollege-5be-dev-ed.lightning.force.com/lightning/setup/CustomTabs/page?address=%2F01rdM00000M7wT8%3Fsetupid%3DCustomTabs'. The page title is 'Tabs' under 'SETUP'. On the left, a sidebar shows 'User Interface' with 'Rename Tabs and Labels' and 'Tabs' selected. The main content area shows a 'Custom Object Tab' for 'Opportunity Automobiles'. It includes a 'Custom Tab Definition Detail' table with fields: Tab Label (Opportunity Automobiles), Object (OpportunityAutomobile), Description, Created By (RAMANAN.M. 12/11/2024, 7:44 pm), Tab Style (Car), Splash Page Custom Link, Modified By (RAMANAN.M. 12/11/2024, 7:44 pm), and Help for this Page.

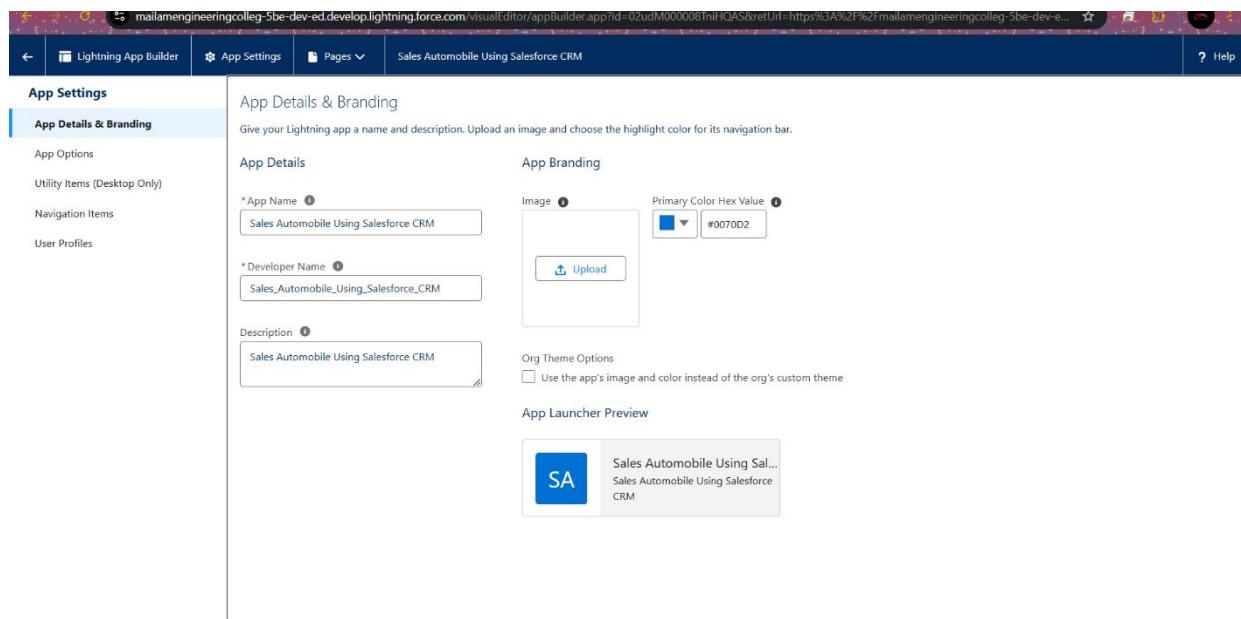
Custom Tab Definition Detail	
Tab Label	Opportunity Automobiles
Object	OpportunityAutomobile
Description	
Created By	RAMANAN.M. 12/11/2024, 7:44 pm
Tab Style	Car
Splash Page Custom Link	
Modified By	RAMANAN.M. 12/11/2024, 7:44 pm

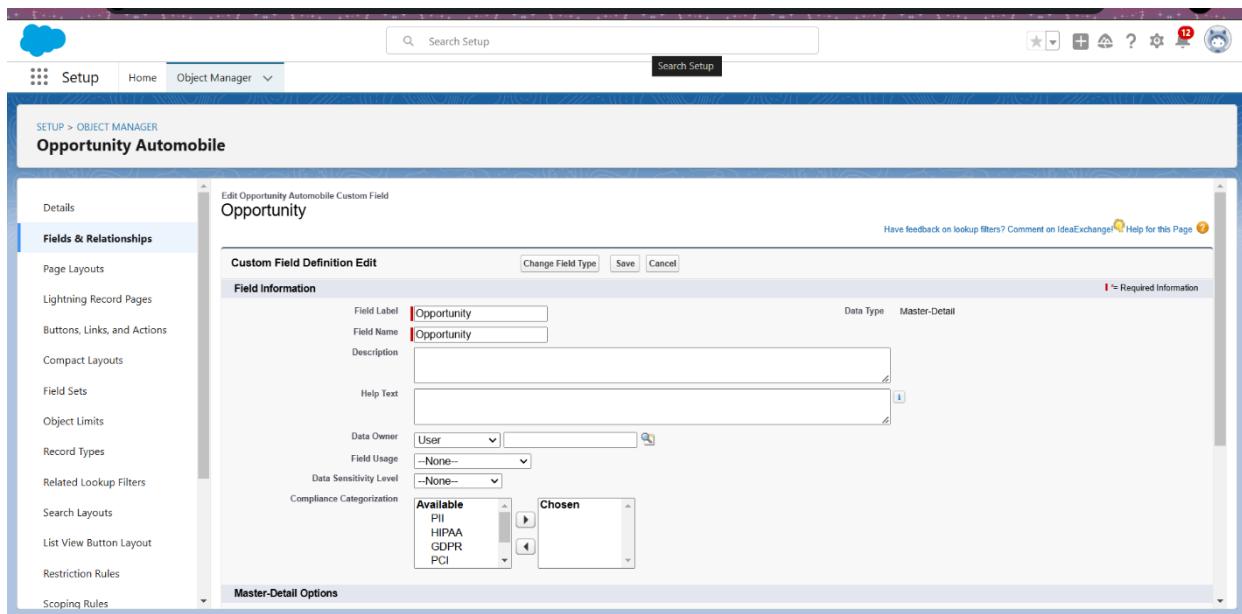
## ➤ Lightning App

- Create a Custom Lightning App that integrates the following components:
  - Opportunity Records
  - Automobile Information records
  - Invoices related to Opportunities

The app should include:

- Navigation to all relevant objects (Opportunities, Automobiles, Invoices).
- A dashboard to visualize Total Sales, Invoices due, Opportunity stage.

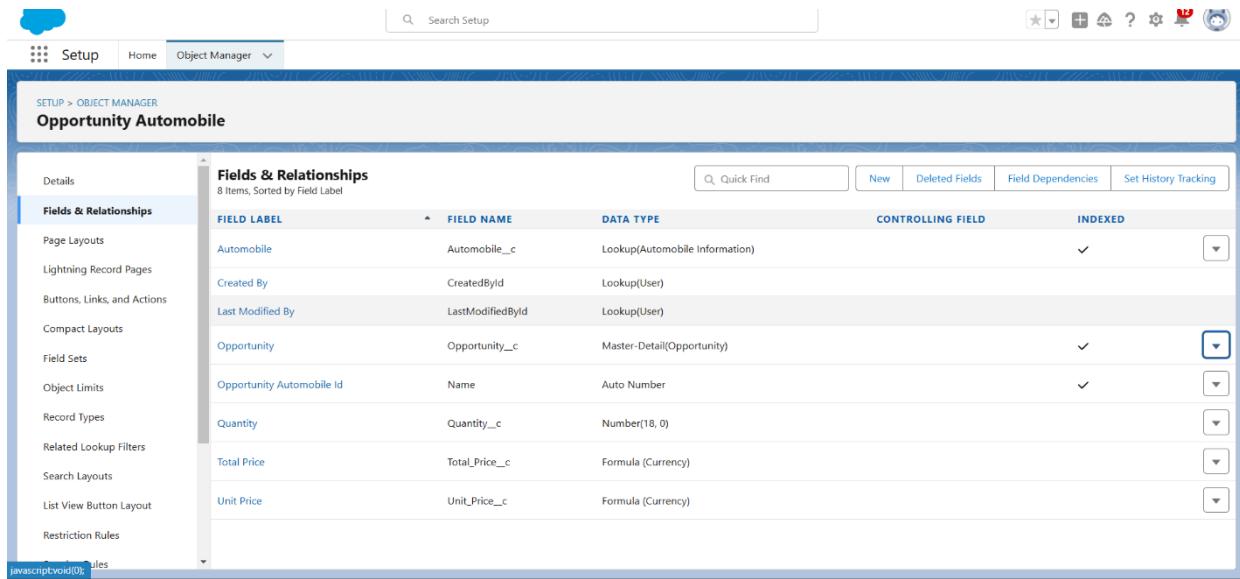




The screenshot shows the Salesforce Setup interface with the following details:

- Page Header:** Search Setup, Home, Object Manager.
- Section:** SETUP > OBJECT MANAGER
- Object:** Opportunity Automobile
- Panel:** Fields & Relationships
- Form:** Custom Field Definition Edit for the 'Opportunity' field.
- Field Information:**
  - Field Label: Opportunity
  - Field Name: Opportunity
  - Description: (empty)
  - Help Text: (empty)
  - Data Owner: User
  - Field Usage: -None-
  - Data Sensitivity Level: -None-
  - Compliance Categorization: Available (PII, HIPAA, GDPR, PCI) and Chosen (empty)
- Buttons:** Change Field Type, Save, Cancel.
- Help:** Have feedback on lookup filters? Comment on IdeaExchange. Help for this Page.

## ➤ Creating Fields and Relationships:



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

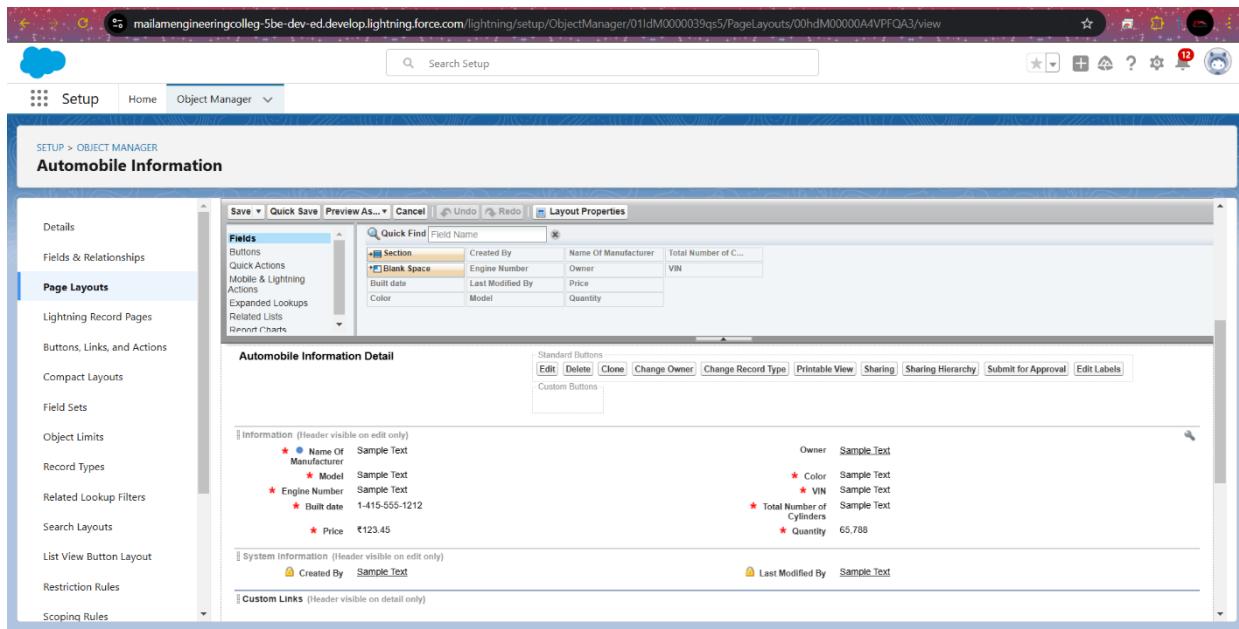
- Page Header:** Search Setup, Home, Object Manager.
- Section:** SETUP > OBJECT MANAGER
- Object:** Opportunity Automobile
- Panel:** Fields & Relationships
- List:** Fields & Relationships (8 items, Sorted by Field Label)
 

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Automobile	Automobile__c	Lookup(Automobile Information)		✓
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Opportunity	Opportunity__c	Master-Detail(Opportunity)		✓
Opportunity Automobile Id	Name	Auto Number		✓
Quantity	Quantity__c	Number(18, 0)		
Total Price	Total_Price__c	Formula (Currency)		
Unit Price	Unit_Price__c	Formula (Currency)		

## ➤ 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 Automobiles Information

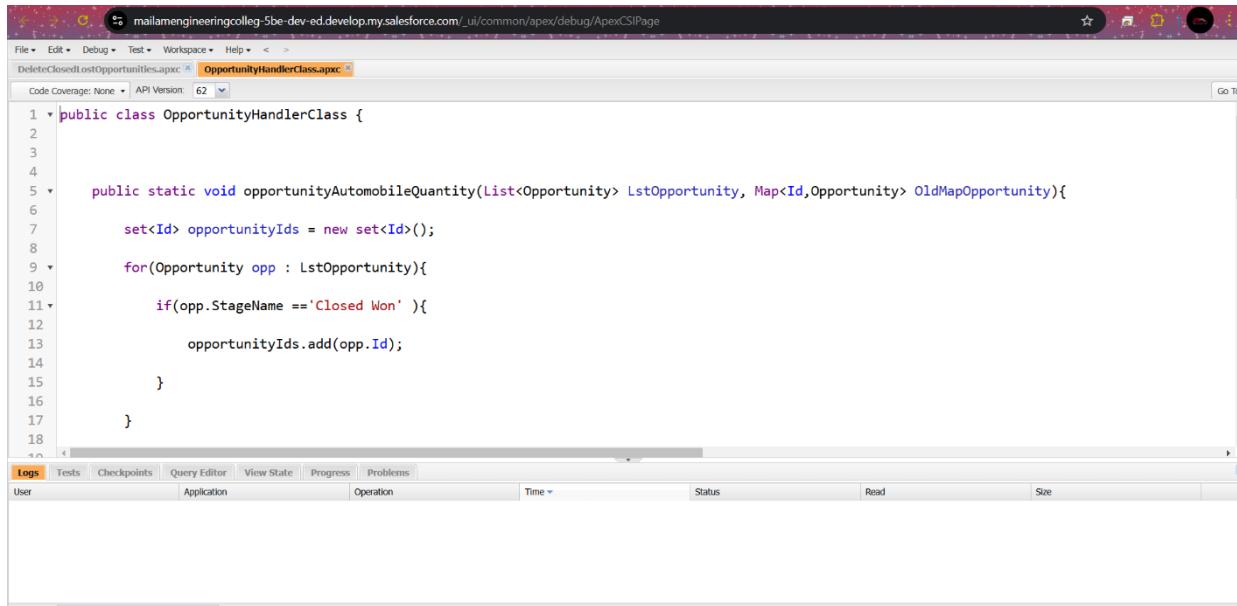


The screenshot shows the Salesforce Setup interface with the URL <https://mailamengineeringcollege-5be-dev-ed.lightning.force.com/lightning/setup/ObjectManager/01lM0000039qs5/PageLayouts/00hdM00000A4VPFQa3/view>. The left sidebar under 'Object Manager' for 'Automobile Information' includes options like Details, Fields & Relationships, Page Layouts (which is selected), Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Search Layouts, List View Button Layout, Restriction Rules, and Scoping Rules. The main content area displays the 'Automobile Information Detail' page layout with sections for Information, System Information, and Custom Links. Fields shown include Name Of Manufacturer, Engine Number, Model, Built date, Price, Owner, VIN, Total Number of Cylinders, and Quantity. Standard buttons for Edit, Delete, Clone, Change Owner, Change Record Type, Printable View, Sharing, Sharing Hierarchy, Submit for Approval, and Edit Labels are located at the top right of the layout.

## ➤ Apex Triggers:

An Apex trigger is a set of instructions that execute when certain events occur on a Salesforce object (like when a record is created, updated, deleted, or restored).

## Creating OpportunityHandlerClass



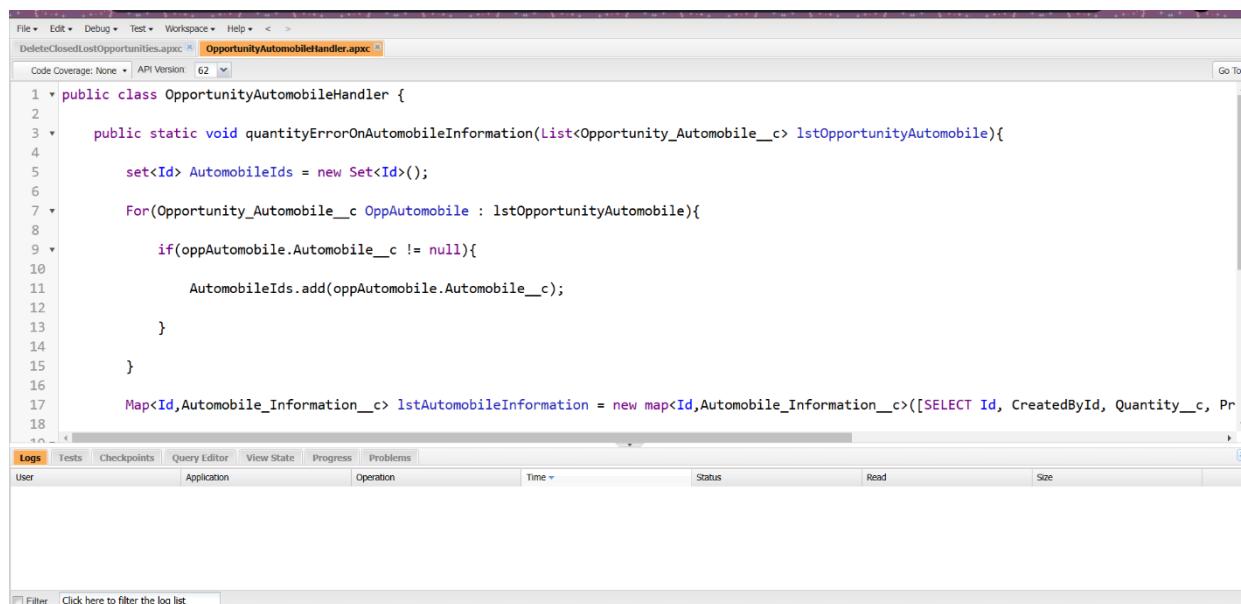
```

1 public class OpportunityHandlerClass {
2
3
4
5     public static void opportunityAutomobileQuantity(List<Opportunity> LstOpportunity, Map<Id,Opportunity> OldMapOpportunity){
6
7         set<Id> opportunityIds = new set<Id>();
8
9         for(Opportunity opp : LstOpportunity){
10
11             if(opp.StageName =='Closed Won'){
12
13                 opportunityIds.add(opp.Id);
14
15             }
16
17         }
18
}

```

The screenshot shows the Salesforce IDE interface with the code editor open. The file is named OpportunityHandlerClass.apxc. The code implements a static method opportunityAutomobileQuantity that takes a list of Opportunities and a map of old opportunities. It iterates through the list, adding the Id of any opportunity with a 'Closed Won' stage to a set of opportunityIds.

## Creating OpportunityAutomobileHandlerClass



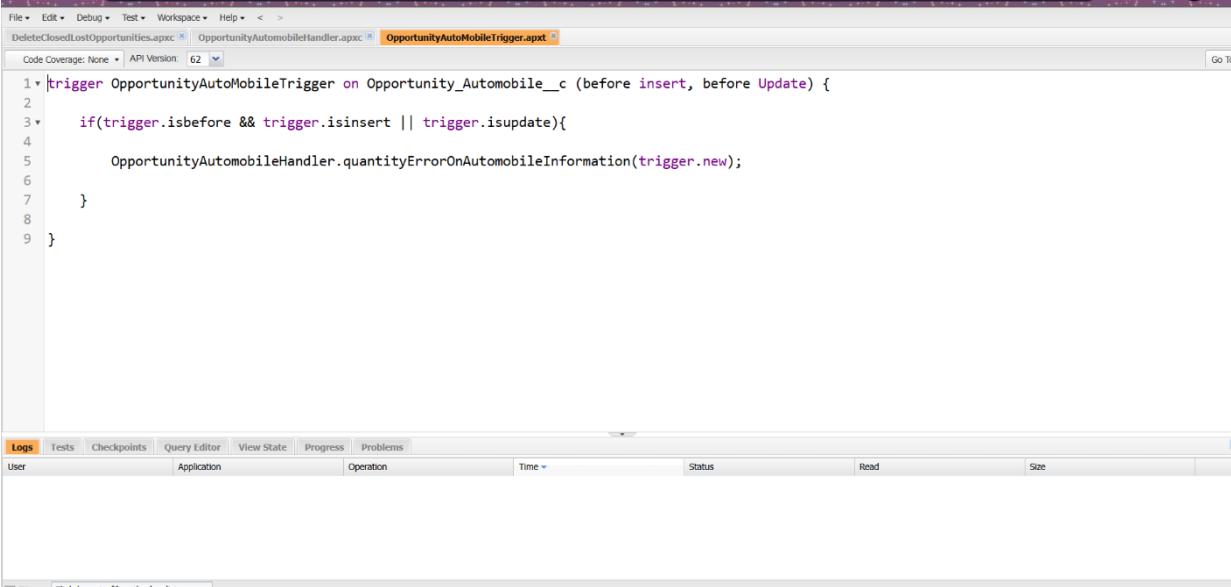
```

1 public class OpportunityAutomobileHandler {
2
3     public static void quantityErrorOnAutomobileInformation(List<Opportunity_Automobile__c> lstOpportunityAutomobile){
4
5         set<Id> AutomobileIds = new Set<Id>();
6
7         For(Opportunity_Automobile__c OppAutomobile : lstOpportunityAutomobile){
8
9             if(OppAutomobile.Automobile__c != null){
10
11                 AutomobileIds.add(OppAutomobile.Automobile__c);
12
13             }
14
15         }
16
17         Map<Id, Automobile_Information__c> lstAutomobileInformation = new map<Id, Automobile_Information__c>([SELECT Id, CreatedById, Quantity__c, Pr
}

```

The screenshot shows the Salesforce IDE interface with the code editor open. The file is named OpportunityAutomobileHandler.apxc. The code implements a static method quantityErrorOnAutomobileInformation that takes a list of Opportunity\_Automobile\_\_c objects. It adds the Id of each automobile to a set of automobileIds.

## Creating OpportunityAutoMobileTrigger



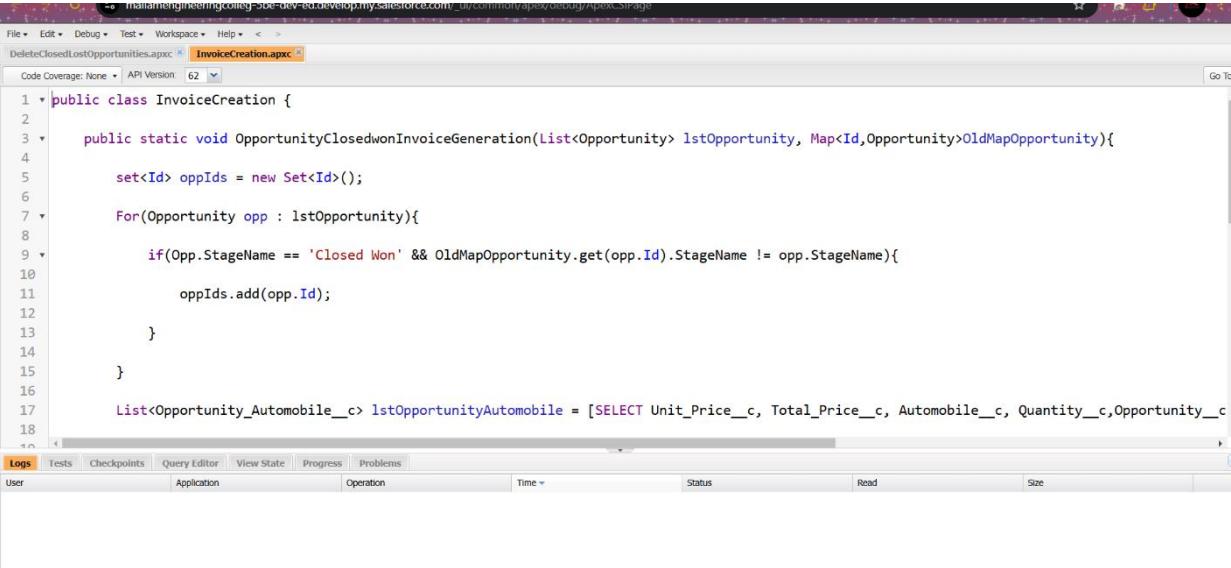
```

trigger OpportunityAutoMobileTrigger on Opportunity_Automobile__c (before insert, before update) {
  if(trigger.isbefore && trigger.isinsert || trigger.isupdate){
    OpportunityAutomobileHandler.quantityErrorOnAutomobileInformation(trigger.new);
  }
}

```

The screenshot shows the Salesforce IDE interface with the code editor open. The code defines a trigger named OpportunityAutoMobileTrigger on the Opportunity\_Automobile\_\_c object. It handles both insert and update events. Inside the trigger, there is a condition that checks if it's a before event and either an insert or an update. If true, it calls the quantityErrorOnAutomobileInformation method from the OpportunityAutomobileHandler class.

## Creating InvoiceCreation class



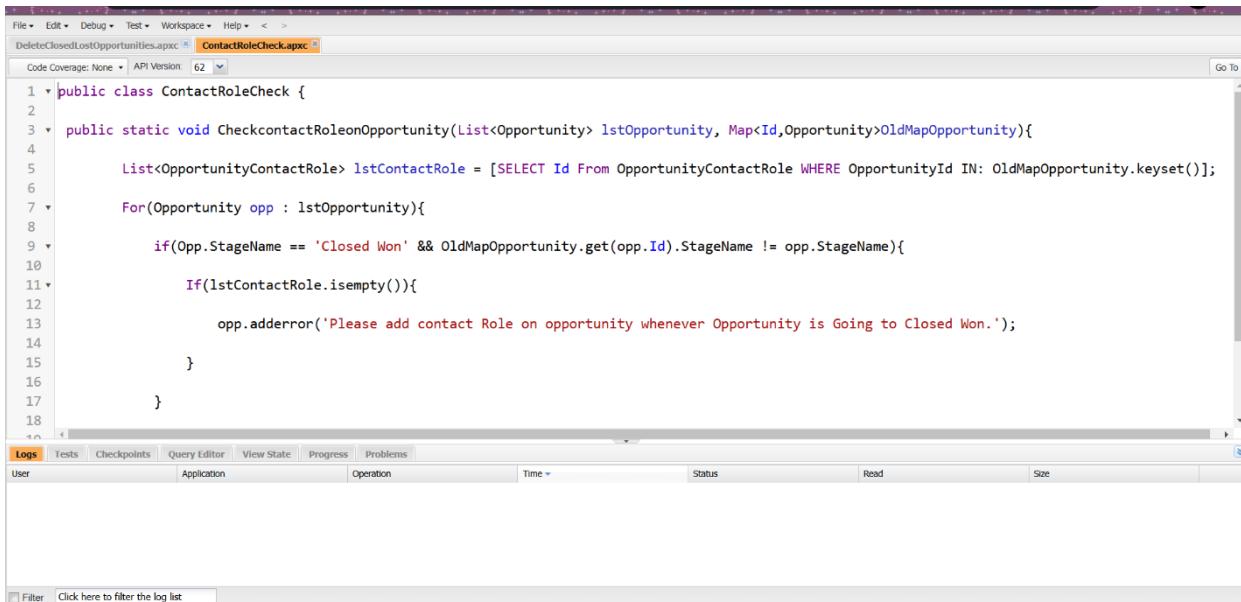
```

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
      FROM Opportunity_Automobile__c WHERE Opportunity__c IN :oppIds];
  }
}

```

The screenshot shows the Salesforce IDE interface with the code editor open. The code defines a public class named InvoiceCreation. It contains a static void method called OpportunityClosedwonInvoiceGeneration. This method takes two parameters: a list of opportunities and a map of old opportunities by ID. It initializes a set of opportunity IDs. Then, it iterates through the list of opportunities. For each opportunity, it checks if its stage name is 'Closed Won' and if the stage name of the corresponding old opportunity is different. If so, it adds the opportunity ID to the set. Finally, it queries a list of Opportunity\_Automobile\_\_c records where the Opportunity\_\_c ID is in the set of IDs.

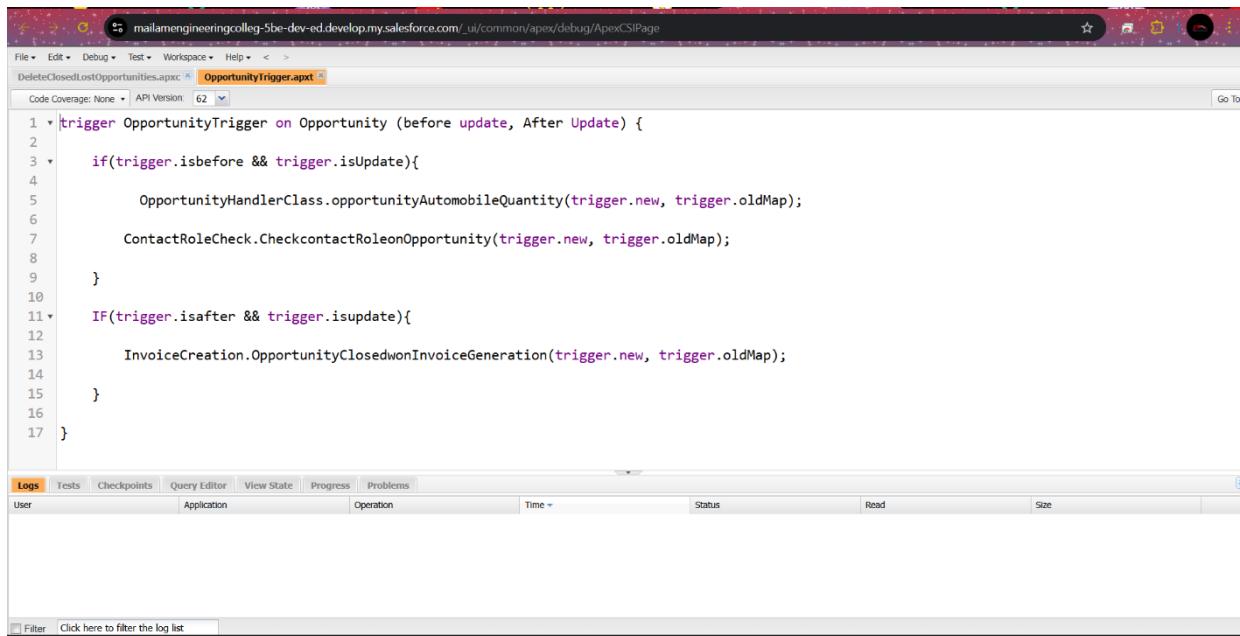
## Creating ContactRoleCheck class



The screenshot shows the Salesforce IDE interface with the ContactRoleCheck.apxc file open. The code implements a static method to check if a contact role is assigned to an opportunity when its stage is 'Closed Won'. If no contact role is found, it adds an error message to the opportunity.

```
1 public class ContactRoleCheck {
2
3     public static void CheckcontactRoleonOpportunity(List<Opportunity> lstOpportunity, Map<Id,Opportunity>OldMapOpportunity){
4
5         List<OpportunityContactRole> lstContactRole = [SELECT Id From OpportunityContactRole WHERE OpportunityId IN: OldMapOpportunity.keySet()];
6
7         For(Opportunity opp : lstOpportunity){
8
9             if(Opp.StageName == 'Closed Won' && OldMapOpportunity.get(opp.Id).StageName != opp.StageName){
10
11                 If(lstContactRole.isEmpty()){
12
13                     opp.adderror('Please add contact Role on opportunity whenever Opportunity is Going to Closed Won.');
14
15                 }
16
17             }
18
19         }
20
21     }
22 }
```

## Creating OpportunityTrigger



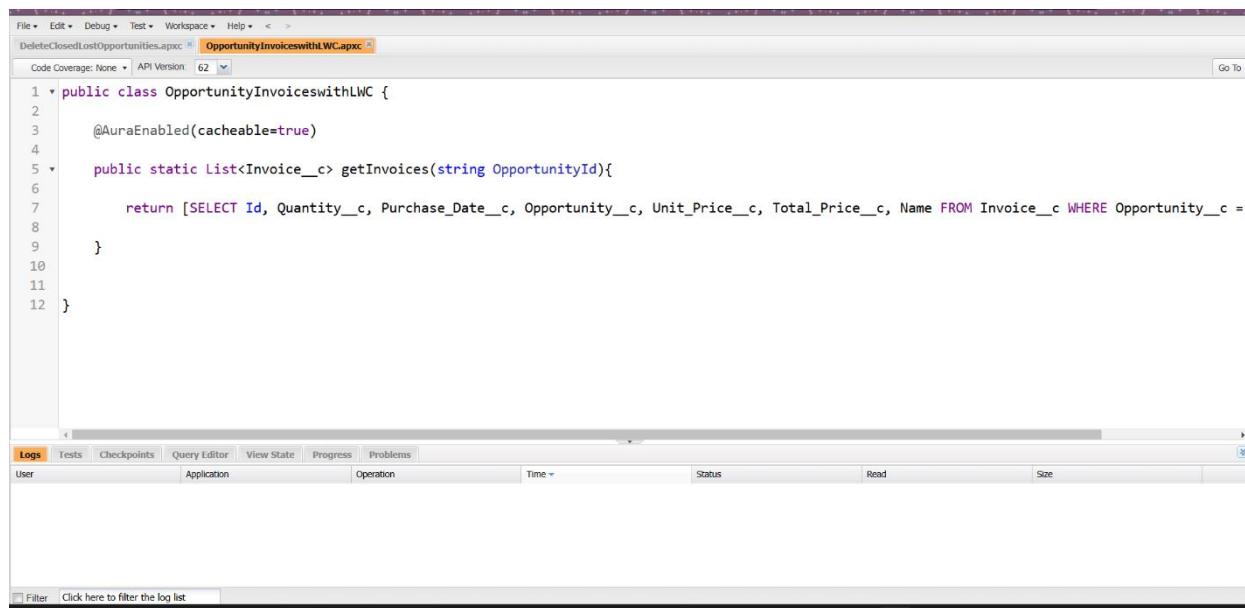
The screenshot shows the Salesforce IDE interface with the OpportunityTrigger.apxt file open. The trigger handles both before update and after update events. It calls the OpportunityHandlerClass.oppotunityAutomobileQuantity method and the ContactRoleCheck.CheckcontactRoleonOpportunity method. Additionally, it calls the InvoiceCreation.OpportunityClosedwonInvoiceGeneration method for after update events.

```
1 trigger OpportunityTrigger on Opportunity (before update, After Update) {
2
3     if(trigger.isbefore && trigger.isUpdate){
4
5         OpportunityHandlerClass.oppotunityAutomobileQuantity(trigger.new, trigger.oldMap);
6
7         ContactRoleCheck.CheckcontactRoleonOpportunity(trigger.new, trigger.oldMap);
8
9     }
10
11     IF(trigger.isafter && trigger.isupdate){
12
13         InvoiceCreation.OpportunityClosedwonInvoiceGeneration(trigger.new, trigger.oldMap);
14
15     }
16
17 }
```

## ➤ LWC Component:

Lightning Web Components (LWC) are a modern framework for building user interfaces in Salesforce. Unlike Aura Components (Salesforce's previous framework), LWC is based on modern web standards, including web components, HTML, CSS, and JavaScript. It offers faster performance, better developer productivity, and is more aligned with web development best practices.

### Create apexclass to get invoice



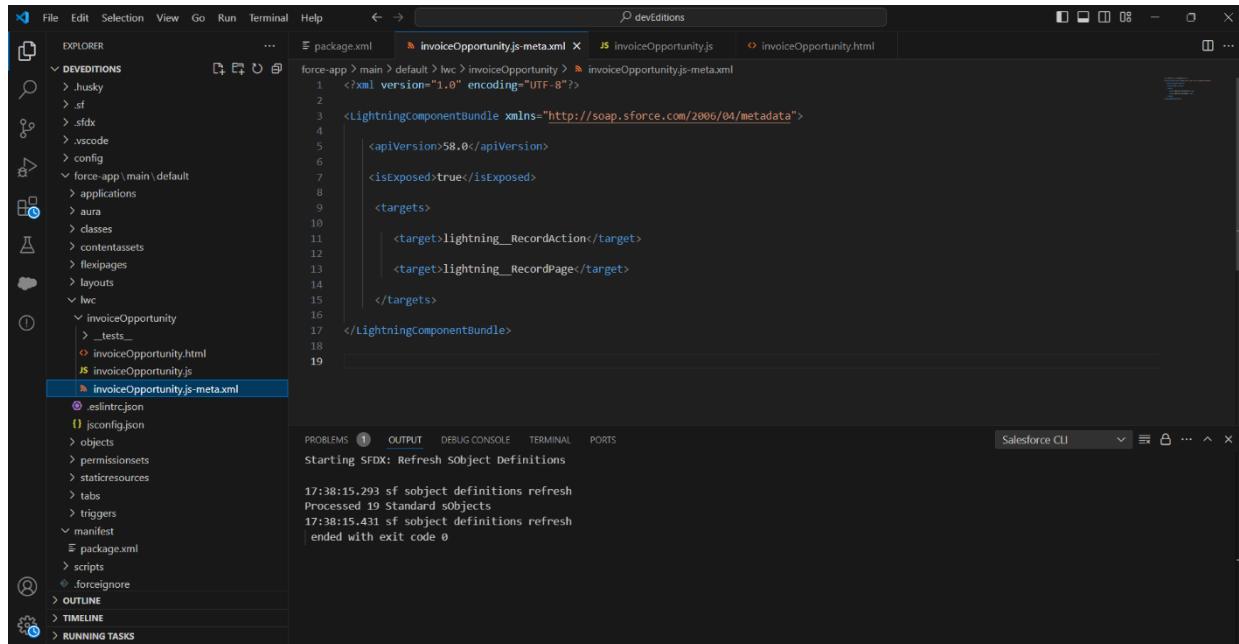
```
File • Edit • Debug • Test • Workspace • Help • < >
DeleteClosedLostOpportunities.apxc | OpportunityInvoiceswithLWC.apxc
Code Coverage: None | API Version: 62 | Go To
1 public class OpportunityInvoiceswithLWC {
2     @AuraEnabled(cacheable=true)
3     public static List<Invoice__c> getInvoices(string OpportunityId){
4         return [SELECT Id, Quantity__c, Purchase_Date__c, Opportunity__c, Unit_Price__c, Total_Price__c, Name FROM Invoice__c WHERE Opportunity__c = :OpportunityId];
5     }
6 }
7
8
9
10
11
12 }
```

The screenshot shows the Salesforce Developer Console interface. The top navigation bar includes File, Edit, Debug, Test, Workspace, Help, and a search bar. Below the navigation is a toolbar with Code Coverage (None), API Version (62), and a Go To button. The main area displays the code for the OpportunityInvoiceswithLWC class. The code defines a static method getInvoices that queries the Invoice\_\_c object based on the Opportunity\_\_c field. The bottom of the screen features a tabs bar with Log (selected), Tests, Checkpoints, Query Editor, View State, Progress, and Problems. Below the tabs is a log table with columns for User, Application, Operation, Time, Status, Read, and Size. A filter input "Filter Click here to filter the log list" is at the bottom left.

## Create Lightning Web Component

The lwc component name is InvoiceOpportunity.

### XML File

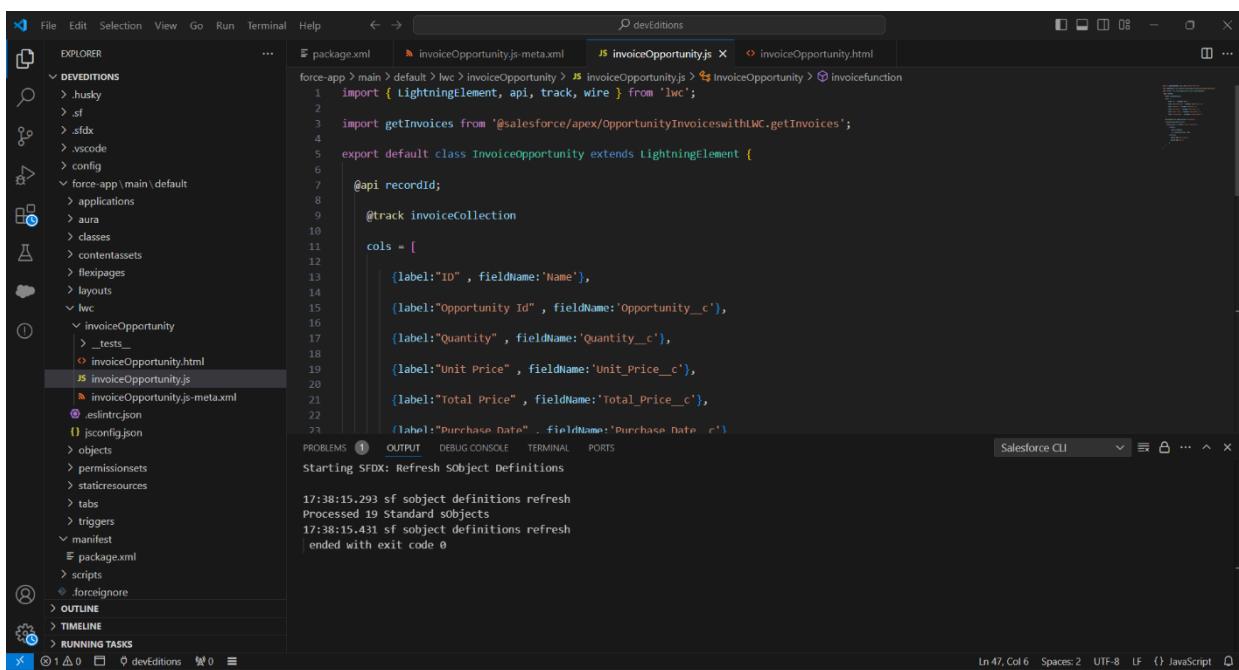


```

<?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



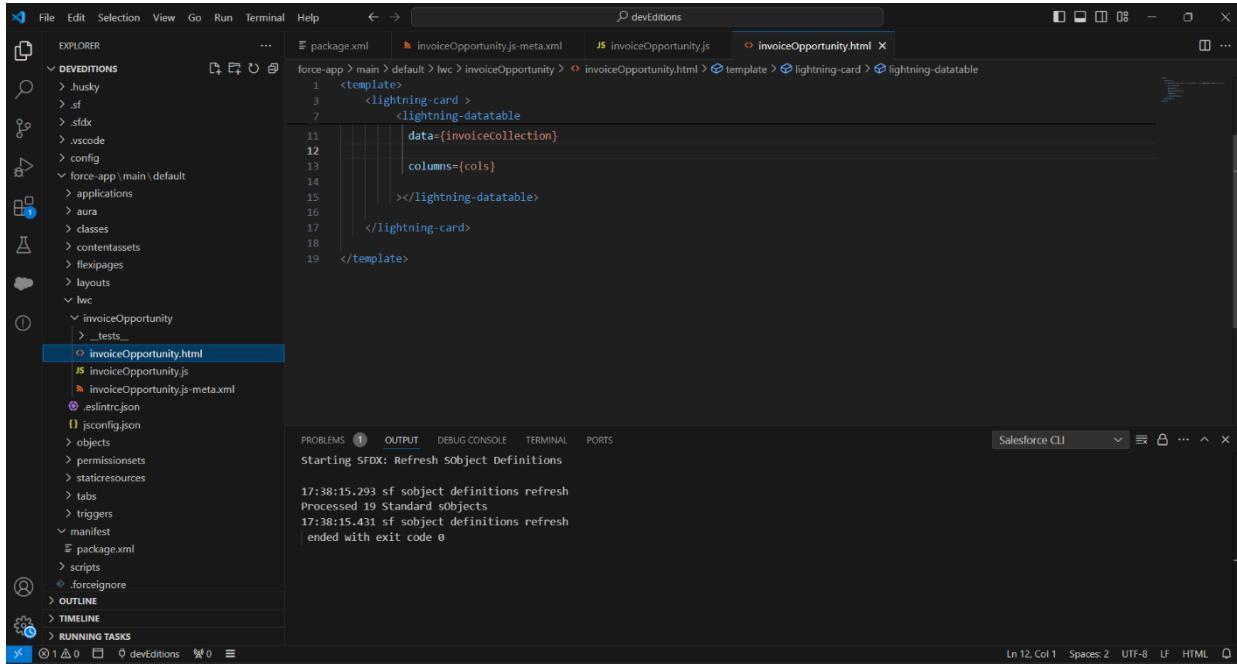
```

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'}
    ];
}

```

## HTML File



```

File Edit Selection View Go Run Terminal Help <- > 🔍 devEditions
EXPLORER ... package.xml invoiceOpportunity.js-meta.xml invoiceOpportunity.js invoiceOpportunity.html
DEVENTIONS <template>
> .husky
> .sf
> .sfdx
> .vscode
> config
> force-app/main/default
> applications
> aura
> classes
> contentassets
> flexipages
> layouts
> lwc
> invoiceOpportunity
> _tests_
> invoiceOpportunity.html
> invoiceOpportunity.js
> invoiceOpportunity.js-meta.xml
> .eslintrc.json
> .jsconfig.json
> objects
> permissionssets
> staticresources
> tabs
> triggers
> manifest
> package.xml
> scripts
> .forceignore
> OUTLINE
> TIMELINE
> RUNNING TASKS
Ln 1 Col 1 Spaces: 2 UTF-8 LF HTML

```

```

<template>
  <lightning-card>
    <lightning-data-table>
      data={invoiceCollection}
      columns={cols}
    </lightning-data-table>
  </lightning-card>
</template>

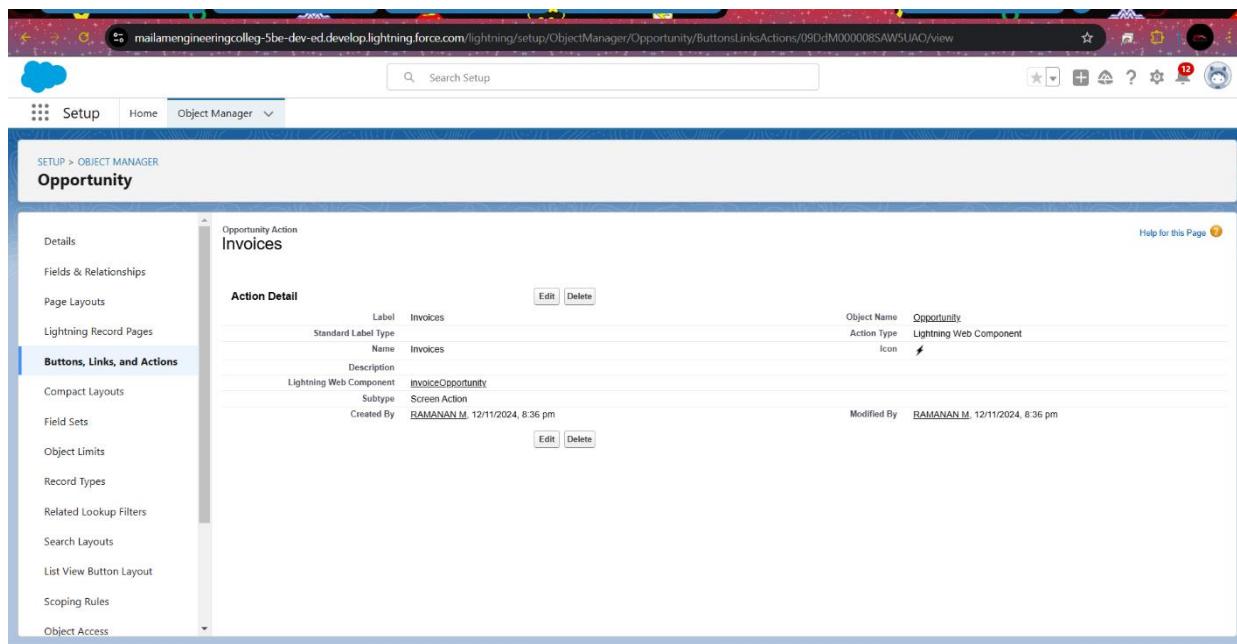
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Starting SFDX: Refresh Object Definitions

17:38:15.293 sf object definitions refresh  
 Processed 19 Standard Objects  
 17:38:15.431 sf object definitions refresh  
 ended with exit code 0

## Create Button to add an opportunity



mailamengineeringcollg-5be-dev-ed.lightning.force.com/lightning/setup/ObjectManager/Opportunity/ButtonsLinksActions/09DdM000008SAW5UAO/view

Setup Home Object Manager

SETUP > OBJECT MANAGER Opportunity

**Action Detail**

Label	Value	Object Name	Value
Standard Label Type	Invoices	Action Type	Lightning Web Component
Name	Invoices	Icon	⚡
Description	invoiceOpportunity	Modified By	RAMANAN M 12/11/2024, 8:36 pm
Lightning Web Component	Screen Action	Created By	RAMANAN M 12/11/2024, 8:36 pm
Subtype			
Created By			

Help for this Page

Opportunity Action Invoices

Details

Fields & Relationships

Page Layouts

Lightning Record Pages

**Buttons, Links, and Actions**

Compact Layouts

Field Sets

Object Limits

Record Types

Related Lookup Filters

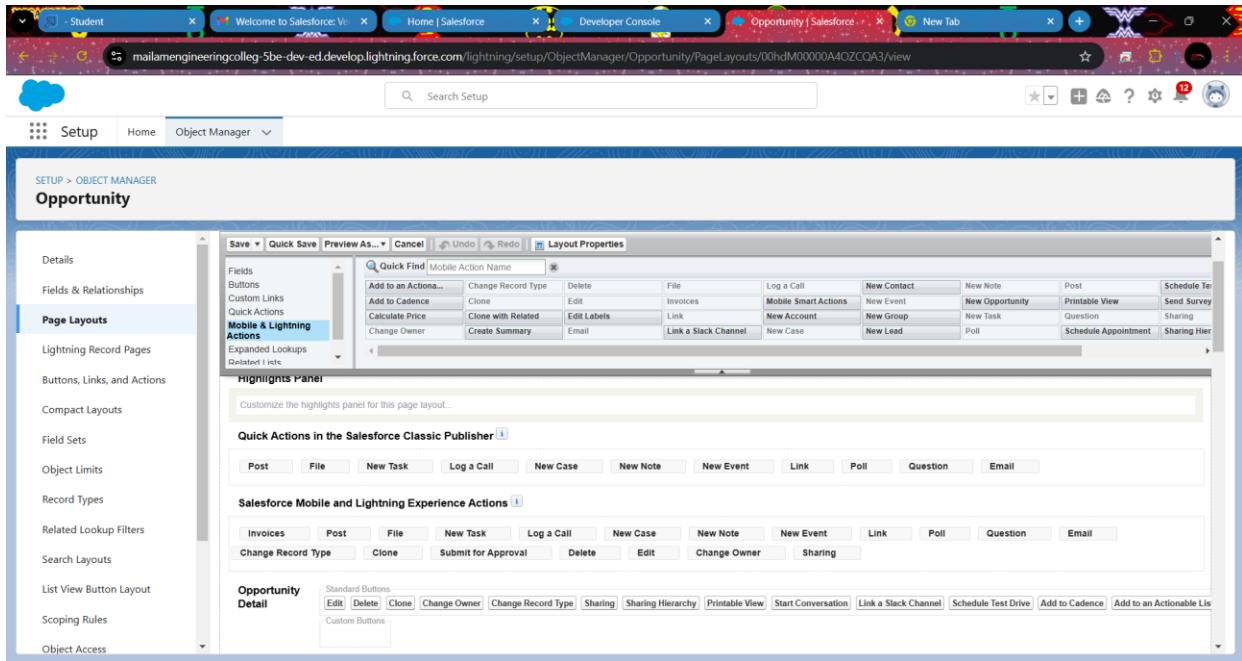
Search Layouts

List View Button Layout

Scoping Rules

Object Access

## Add Invoice opportunity into opportunity record page



SETUP > OBJECT MANAGER  
**Opportunity**

Details  
Fields & Relationships  
**Page Layouts**  
Lightning Record Pages  
Buttons, Links, and Actions  
Compact Layouts  
Field Sets  
Object Limits  
Record Types  
Related Lookup Filters  
Search Layouts  
List View Button Layout  
Scoping Rules  
Object Access

Save | Quick Save | Preview As... | Cancel | Undo | Redo | Layout Properties

Quick Find: Mobile Action Name

Add to Actions...	Change Record Type	Delete	File	Log a Call	New Contact	New Note	Post	Schedule Test Drive
Add to Cadence	Clone	Edit	Invoices	Mobile Smart Actions	New Event	New Opportunity	Printable View	Send Survey
Calculate Price	Clone with Related	Edit Labels	Link	New Account	New Group	New Task	Question	Sharing
Change Owner	Create Summary	Email	Link a Slack Channel	New Case	New Lead	Poll	Schedule Appointment	Sharing Hier

Highlights Panel  
Customize the highlights panel for this page layout...

Quick Actions in the Salesforce Classic Publisher

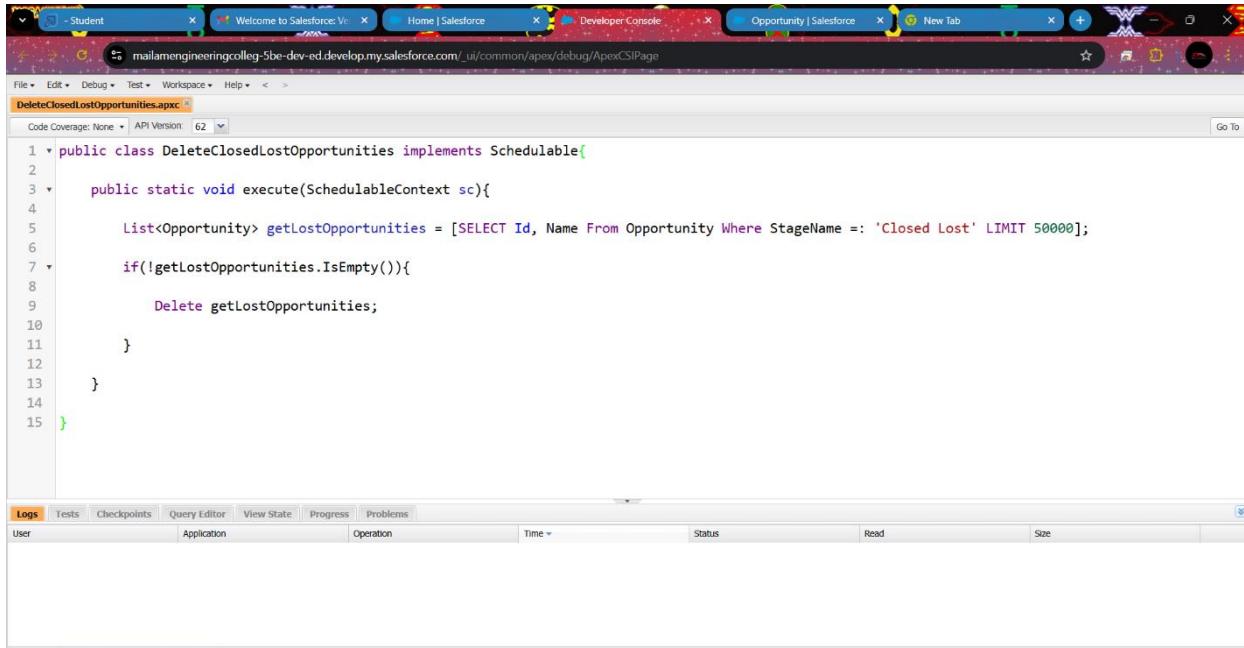
Post	File	New Task	Log a Call	New Case	New Note	New Event	Link	Poll	Question	Email
------	------	----------	------------	----------	----------	-----------	------	------	----------	-------

Salesforce Mobile and Lightning Experience Actions

Invoices	Post	File	New Task	Log a Call	New Case	New Note	New Event	Link	Poll	Question	Email
Change Record Type	Clone	Submit for Approval	Delete	Edit	Change Owner	Sharing					

Opportunity Detail  
Standard Buttons: Edit, Delete, Clone, Change Owner, Change Record Type, Sharing, Sharing Hierarchy, Printable View, Start Conversation, Link a Slack Channel, Schedule Test Drive, Add to Cadence, Add to an Actionable List  
Custom Buttons

## ➤ Apex Schedulers: Delete opportunity schedule class



```
File | Edit | Debug | Test | Workspace | Help | < >
DeleteClosedLostOpportunities.apxc | Go To
Code Coverage: None | API Version: 62 | Logs Tests Checkpoints Query Editor View State Progress Problems
1 * public class DeleteClosedLostOpportunities implements Scheduledable{
2
3     public static void execute(SchedulableContext sc){
4
5         List<Opportunity> getLostOpportunities = [SELECT Id, Name From Opportunity Where StageName =: 'Closed Lost' LIMIT 50000];
6
7         if(!getLostOpportunities.IsEmpty()){
8
9             Delete getLostOpportunities;
10
11         }
12
13     }
14
15 }
```

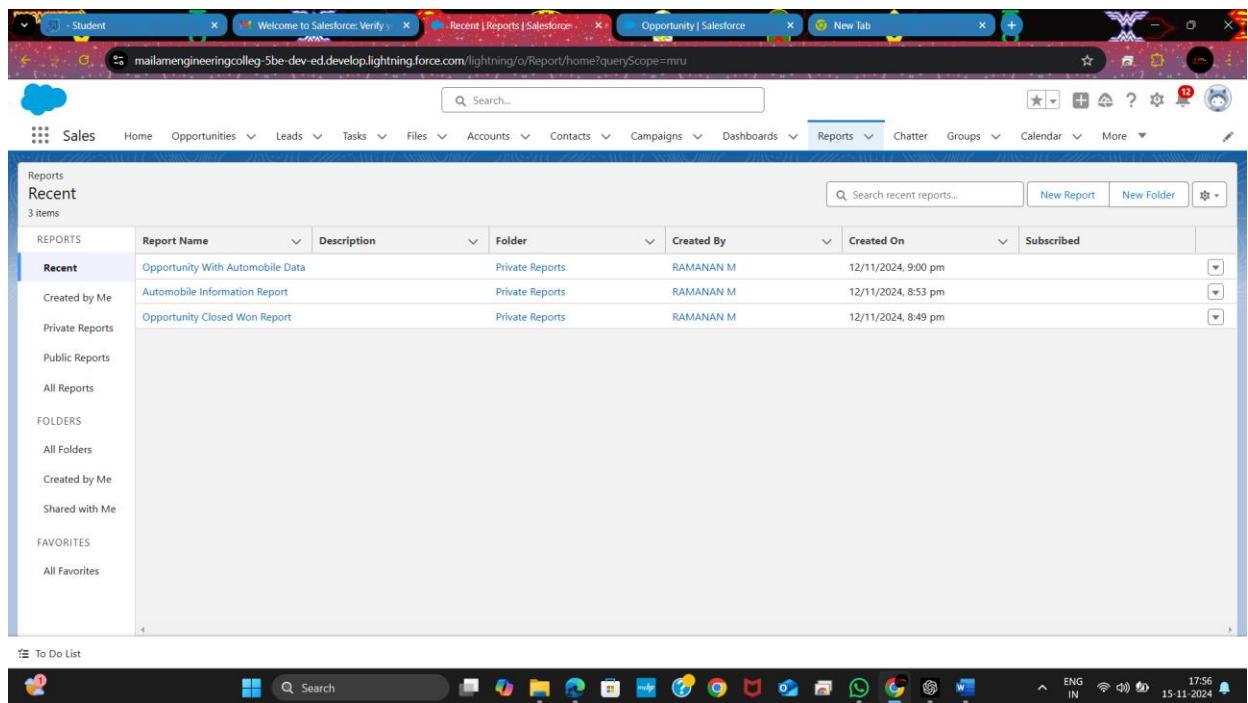
Logs Tests Checkpoints Query Editor View State Progress Problems

User	Application	Operation	Time	Status	Read	Size

## ➤ 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.

Here we create three reports namely Opportunity with automobile data, opportunity closed won report, and Automobile Information report.

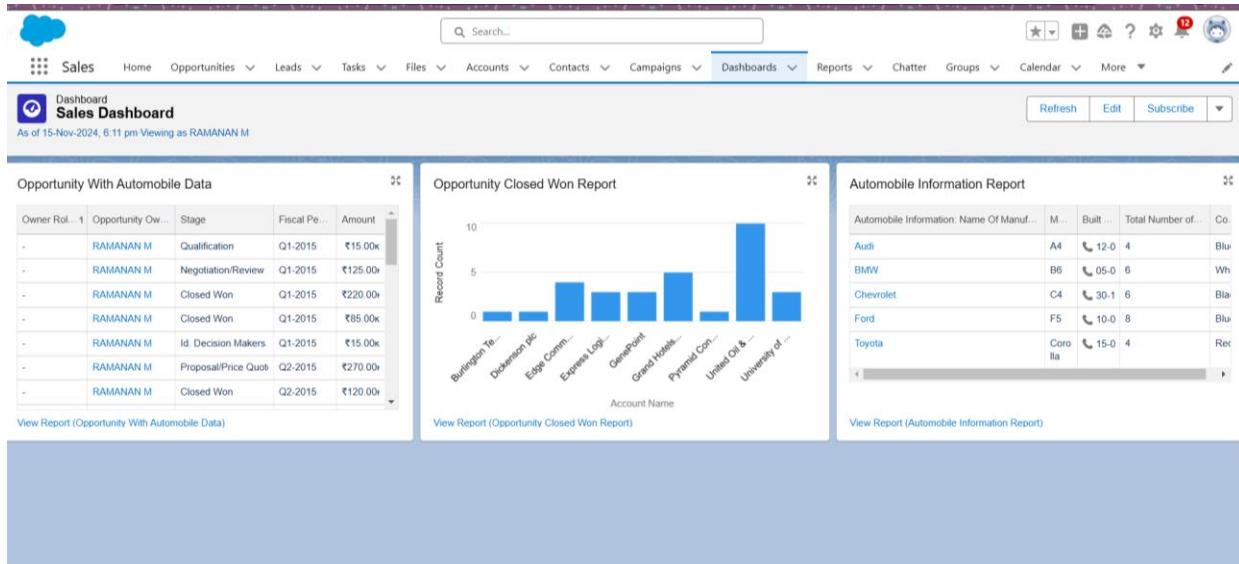


Report Name	Description	Folder	Created By	Created On	Subscribed
Opportunity With Automobile Data	Private Reports	RAMANAN M	12/11/2024, 9:00 pm		
Automobile Information Report	Private Reports	RAMANAN M	12/11/2024, 8:53 pm		
Opportunity Closed Won Report	Private Reports	RAMANAN M	12/11/2024, 8:49 pm		

## ➤ 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.

## The Created Dashboard:



## 5. Testing and Validation for the Automobile Sales CRM Project:

In the Automobile Sales CRM project, testing and validation play a crucial role in ensuring the reliability, accuracy, and overall functionality of the application. The project involves multiple components such as Apex classes, Apex triggers, Lightning Web Components (LWC), and custom objects. For Apex classes and triggers, unit tests are designed to verify business logic, such as calculating automobile prices, updating opportunity quantities, and handling complex workflows. These unit tests simulate real-world scenarios by inserting mock data and validating outcomes with assertions. Additionally, tests cover edge cases, such as handling invalid or missing inputs, ensuring that the system behaves as expected under all conditions, and confirming that bulk processing does not exceed Salesforce governor limits. Apex test classes ensure that the code is fully covered (with at least 75% test coverage) and compliant with Salesforce's deployment requirements.

For the user interface (UI), testing focuses on Lightning Web Components (LWC) to ensure that the system is intuitive, responsive, and user-friendly. The UI testing validates the correct functionality of components such as automobile search, invoice generation, and opportunity management. Tests are implemented to simulate user interactions, such as filtering automobiles by make, adding items to opportunities, and updating quantities. Using Jest for testing LWC, developers verify that components respond correctly to user input, trigger appropriate events, and dynamically update the interface based on real-time data. These tests ensure that the final product offers a seamless user experience, with minimal bugs or UI inconsistencies, and that all components interact smoothly with Salesforce's backend systems.

## **6. Key Scenarios Addressed by Salesforce in the Implementation Project:**

### **1. Managing Automobile Inventory**

**Scenario:** The business needs to manage a dynamic inventory of automobiles with details such as make, model, price, stock availability, and other relevant attributes. The sales team needs a centralized system to view and update automobile information in real time.

#### **Salesforce Solution:**

- Custom Object for Automobile Inventory: A custom object is created to store automobile details (e.g., model, price, stock quantity).
- Lightning Web Component (LWC) is built to enable the sales team to view and search for automobiles in real time.
- The system can auto-update stock quantities based on sales or returns through Apex triggers.

## 2. Customer Relationship Management (CRM)

**Scenario:** The business needs to track detailed customer information, including contact details, previous interactions, and automobile purchases. The system should also allow for effective follow-ups and communication with leads, prospects, and customers.

### **Salesforce Solution:**

- Contact and Account Management: Salesforce's Contact and Account objects are customized to track customer information and interactions related to automobile purchases.
- Lead Management: Leads are captured through forms or imports and converted into Opportunities when ready for further engagement.
- Task and Event Tracking: Salesforce's Task and Event functionalities are used to create reminders, track follow-up calls, and schedule meetings with customers.

### **7. Conclusion:**

The Automobile Sales CRM project has successfully implemented a comprehensive solution that streamlines key business processes, enhances sales operations, and improves customer relationship management. By leveraging Salesforce's powerful features, including custom objects, Apex triggers, Lightning Web Components (LWC), and automation tools like Process Builder and Flows, the project has effectively addressed critical use cases such as managing automobile inventory, tracking sales opportunities, generating invoices, and automating workflows. The system enables real-time data updates, detailed reporting and analytics, and seamless integration of sales and customer data, ultimately driving increased sales efficiency and better customer experiences. Through this

implementation, the business now has a scalable, flexible CRM solution that supports both operational needs and strategic growth.

## For Demo Video



[Click Here](#)