Title:– WhatNext Vision Motors: Shaping the Future of Mobility with Innovation and Excellence

Project Overview:

WhatsNext Vision Motors has transformed its customer interaction and operational workflows by deploying an advanced Salesforce-based CRM system designed for complete vehicle ordering and dealership operations. The platform supports customer activities such as placing vehicle orders, scheduling test drives, and submitting service requests, while ensuring real-time stock accuracy and automated dealer allocation based on the customer's location.

To maintain seamless processing, the system incorporates intelligent automation tools including Record-Triggered Flows, Apex Triggers, Batch Apex, and Scheduled Apex. These components work together to manage end-to-end order lifecycle events such as status updates, stock validation, and timely customer notifications.

Overall, this CRM solution enhances business efficiency by minimizing manual tasks, reducing mistakes, accelerating order handling, and creating a unified, customer-friendly vehicle purchasing experience.

Objectives:

- The system streamlines the order management process by removing manual data entry and automating routine tasks.
  This leads to quicker processing and a smooth purchasing experience for customers.
- Customer location is automatically analyzed, and the order is routed to the most suitable nearby dealership.
  This minimizes delivery time and enhances overall convenience.
- Vehicle stock is validated in real time, ensuring that customers can place orders **only** for available models.
  This avoids false commitments and improves customer trust.
- Scheduled Apex functionality delivers automated reminder emails prior to a customer's test drive.
  This helps increase attendance rates and keeps customers engaged.
- Batch Apex processes are used to regularly refresh inventory quantities and auto-confirm pending orders.
  This maintains accurate stock visibility and enables an uninterrupted workflow.

- The CRM platform ensures a smooth, error-free order lifecycle supported by instant communication and clear status updates.
  This improves operational reliability and boosts customer satisfaction.

Student Outcomes:

- Gained practical experience in developing Salesforce CRM applications using Apex classes, flows, and Lightning components.
- Learned how to design and implement automated workflows for order handling and inventory management.
- Understood how to create effective data models that reflect real-world business operations and relationships.
- Developed the ability to build approval-less automation systems and intelligent dealer allocation mechanisms.
- Acquired hands-on knowledge of Batch Apex and Scheduled Apex for managing large-volume operations and scheduled tasks.
- Gained exposure to integrating CRM features with customer activities such as test drive scheduling and service management.

System Requirements:

Hardware Requirements:

- Computer with minimum 4 GB RAM, Dual-core processor
- Stable internet connection

Software Requirements:

- Salesforce Developer Edition Org
- Modern Web Browser (Chrome / Firefox)

Skills Required:

- Salesforce Data Modeling & Configuration
- Lightning App Builder
- Security & Access Management
- Apex Classes & Trigger Handlers
- Batch Apex & Scheduled Apex
- Record Triggered Flows

Phases Overview:

Phase No.	Phase Name	Description	Page Numbers
1	Requirement Analysis & Planning	Identify business requirements for CRM vehicle ordering and workflow automation	5-9
2	Salesforce Development – Backend & Configuration	Create objects, fields, automations, triggers and batch logic	8-38
3	UI/UX Development & Customization	Build an intuitive CRM app with Lightning pages and layouts	39-42
4	Security	Load data, test system reliability, and configure system-level security	42-43
5	Documentation & Maintenance	Deploy CRM app, monitor performance and maintain system health	43-46

Project Main Overview:

WhatsNext Vision Motors has developed an end-to-end Vehicle Management System on the Salesforce CRM platform to create a smooth and unified customer journey across ordering, test drives, and after-sales services. The system intelligently assigns each new order to the closest dealership by analyzing the customer's address and restricts order creation for vehicles that are currently unavailable in stock.

The solution is supported by a well-structured data model consisting of custom objects such as **Vehicle__c, Vehicle_Order__c, Vehicle_Dealer__c, Vehicle_Customer__c, Vehicle_Test_Drive__c**, and **Vehicle_Service_Request__c**, which collectively store comprehensive details about vehicles, customers, orders, dealership operations, test drives, and service activities.

Apex Triggers and Batch Apex routines ensure that stock validation happens instantly before confirming any order, and pending orders are updated automatically when inventory is replenished. Record-Triggered Flows handle dealer assignment and send automated reminders for test drive bookings, while Scheduled Apex takes care of daily order processing tasks.

This CRM framework significantly enhances operational efficiency, reduces manual workload, and provides a smart, automated, and scalable solution tailored for the automotive domain.

Main Objectives

- Ensure automatic assignment of customer orders to the nearest dealership based on location.
- Prevent customers from ordering vehicles that are currently unavailable in inventory.
- Enable scheduled status updates for orders using batch automation.
- Send automated reminder emails for upcoming test drive appointments.
- Maintain complete and accurate history for test drives, service requests, and the entire order lifecycle.
- Provide real-time visual dashboards for tracking orders, stock levels, and dealership performance.
- Implement role-based permissions for administrators, dealers, and customers.
- Deliver a fully digital, integrated experience for vehicle ordering and after-sales services.

Phase 1: Requirement Analysis & Planning:

1. Understanding Business Requirements:

Objective:

Understand how automotive companies manage the complete vehicle purchase cycle including vehicle stock availability, dealer assignment, customer coordination, and scheduled services and identify operational challenges that reduce efficiency and customer satisfaction.

The goal is to build a Salesforce-based solution that automates order processing, stock validation, and dealer mapping while providing centralized visibility and analytics for the management of vehicle orders.

Approach:

- Gather and analyze requirements from Sales Managers, Dealer Partners, Service Coordinators, and Customers to understand the current vehicle ordering workflow and pain points.
- Study how customers select vehicles, how dealers confirm orders, and how stock availability is tracked across multiple branches.
- Identify challenges such as wrong dealer mapping, delayed order confirmation, manual stock updates, scheduling follow-ups, and lack of automated communication.
- Conduct requirement study using ChatGPT, Google, Salesforce Documentation, and Trailhead to design a scalable and secure CRM-based Vehicle Management System.

Key Business Requirements Identified:

- Provide a Salesforce CRM application for placing and managing vehicle orders.
- Auto-assign orders to the nearest dealer based on customer location.
- Restrict order booking when vehicle stock is unavailable.
- Enable customers to schedule test drives through CRM and receive reminders.
- Allow managers to track stock availability, pending orders, and test-drive schedules through dashboards.
- Ensure secure, role-based visibility for Admin, Dealer, and Customer logins.
- Generate analytical dashboards and reports for the sales team to monitor order trends and customer engagement.

2. Defining Project Scope & Objectives:

Project Scope:

- Build a Salesforce-based Vehicle Ordering System to automate customer orders, dealer assignment, stock validation, and scheduled test drive reminders.
- Integrate automation technologies such as Flows, Apex, and Batch Apex for smooth processing and real-time updates.
- Provide customers with a self-service interface to place orders and track test-drive bookings.
- Implement security and access controls for Admins, Dealers, and Customers.
- Allow sales managers to monitor vehicle stock, order confirmation rates, and conversion insights through dashboards.

Objectives Summary:

- Simplify the vehicle ordering workflow and remove manual follow-up processes.
- Improve customer experience through automated dealer assignment.
- Prevent stock-related errors by validating inventory before order creation.
- Increase productivity through automation of reminders and order-status updates.
- Ensure secure access and protected customer data using profiles and role-based visibility.
- Support management decision-making using CRM dashboards and analytics.

3. Gathering & Analyzing User Needs

Users Involved:

- Customers: Place vehicle orders, schedule test drives, check order status.
- Dealers: View assigned orders, update order progress, manage delivery.
- Sales Managers: Monitor stock availability, dealer performance, and order fulfillment.
- System Administrator: Configure user access, monitor workflows, and ensure data security.

Key Functional Needs:

- Intuitive customer ordering screen and test drive booking interface.
- Automatic dealer assignment based on customer address / location.
- Stock validation before confirming an order.
- Dealer dashboard to view assigned orders and update order status.
- Automated email alerts for test drive reminders and order status updates.

- Reports and dashboards to track orders, stock, and dealer performance.

Tools Used:

- Google Forms – To collect business and user requirements from sales and dealer teams.
- Miro Boards – To visualize customer ordering and dealer assignment workflows.
- User Personas – To design personalized experiences for customers, dealers, and sales managers.
  (Tools listed only for real-time project standard documentation.)

4. Identifying Key Salesforce Features & Tools Required

Salesforce Features Planned:

- **Custom Objects:**
  - Vehicle__c → Stores vehicle information and stock quantity.
  - Vehicle_Dealer__c → Stores dealer details and location.
  - Vehicle_Customer__c → Stores customer contact and address details.
  - Vehicle_Order__c → Tracks orders placed by customers and dealer assignment.
  - Vehicle_Test_Drive__c → Tracks test drive scheduling and completion.
  - Vehicle_Service_Request__c → Tracks servicing and maintenance requests.
- **Standard Object:**
  - Account, Contact, Report, Dashboard.
- **Automations:**
  - Record-Triggered Flows for dealer auto-assignment and email reminders.
  - Scheduled Flows for date-based notifications.
- **Apex:**
  - Trigger Handler for stock validation and stock reduction.
  - Batch Apex for auto-updating pending orders after stock refill.
  - Scheduled Apex for batch execution.
- **UI:**
  - Lightning App Pages for vehicle, order, and test drive screens.
- **Email Services:**
  - Email templates and alerts for reminders and order status updates.

- **Security:**
  - Profiles, Permission Sets, Role Hierarchy, Field-Level Security, and component visibility.

5. Designing Data Model and Security Model:

Data Model Includes:

1. **Vehicle__c (Custom Object)**
   - Stores vehicle model and stock quantity.
   - Linked to Dealer for stock distribution.
   - Includes fields like Price, Model, Status, and Stock Quantity.
2. **Vehicle_Order__c (Custom Object)**
   - Tracks customer order details.
   - Linked to Customer and Vehicle.
   - Includes fields like Order Date, Dealer, Status.
3. **Vehicle_Customer__c (Custom Object)**
   - Stores customer identity and address.
   - Used for order and test drive records.
4. **Vehicle_Dealer__c (Custom Object)**
   - Stores dealer information and location.
   - Used to auto-assign orders based on proximity.
5. **Vehicle_Test_Drive__c (Custom Object)**
   - Tracks test drive bookings.
   - Linked to customer and vehicle.
6. **Security Model Design:**
   - Role Hierarchy: Admin → Dealer → Customer
   - Profiles: Admin Profile, Dealer Profile, Customer Profile
   - Record-Level Security: Sharing rules provide dealers access only to assigned orders
   - Field-Level Security: Protects sensitive customer and vehicle data
   - Component Visibility: Shows relevant pages dynamically based on login role

Summary:

This phase provided an in-depth understanding of the complete automotive ordering workflow and highlighted the key processes where automation can significantly reduce manual work. Through detailed analysis of the needs of customers, dealers, sales personnel, and management, the project scope and system architecture were clearly established.

The finalized data model and security structure support scalable automation, secure information flow, and seamless collaboration across customers, dealership teams, and administrative users, ensuring a reliable and efficient CRM environment.

Phase 2: Salesforce Development – Backend & Configurations:

This phase establishes the core system logic for CRM vehicle ordering.

Major Configurations:

- Designed custom objects and fields to manage vehicle details, customer orders, test-drive scheduling, and service requests.
- Implemented validation rules to restrict incorrect data entry and avoid duplicates within the system.
- Built automated Flows to handle dealer assignment and to send reminder notifications for scheduled test drives.
- Developed Apex Trigger Handlers to validate stock availability and update inventory levels during order creation.
- Utilized Batch Apex to automatically process and update pending orders whenever vehicle stock is replenished.
- Configured Scheduled Apex to run batch processes daily, ensuring regular updates and smooth system operations.

Milestone 1: Salesforce Account:

Introduction:

Are you new to Salesforce? Not sure exactly what it is, or how to use it? Don't know where you should start on your learning journey? If you've answered yes to any of these questions, then you're in the right place. This module is for you. Welcome to Salesforce! Salesforce is game-changing technology, with a host of productivity-boosting features, that will help you sell smarter and faster.

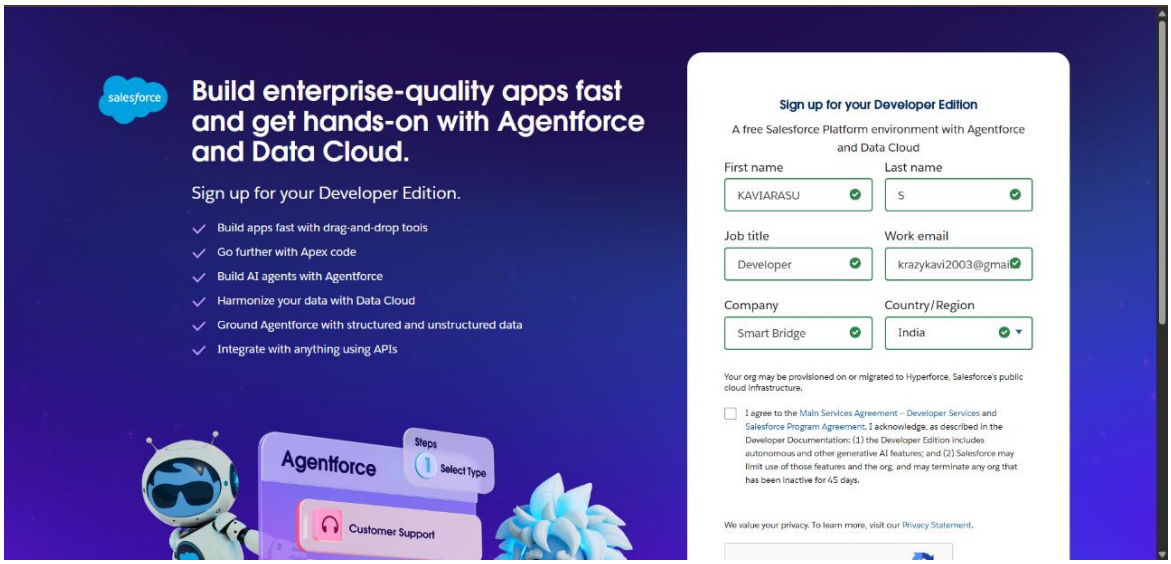
What Is Salesforce?

Salesforce is your customer success platform, designed to help you sell, service, market, analyze, and connect with your customers.

Activity 1: Creating Developer Account:

Creating a developer org in salesforce.

1. Go to https://developer.salesforce.com/signup
2. On the sign-up form, enter the following details:

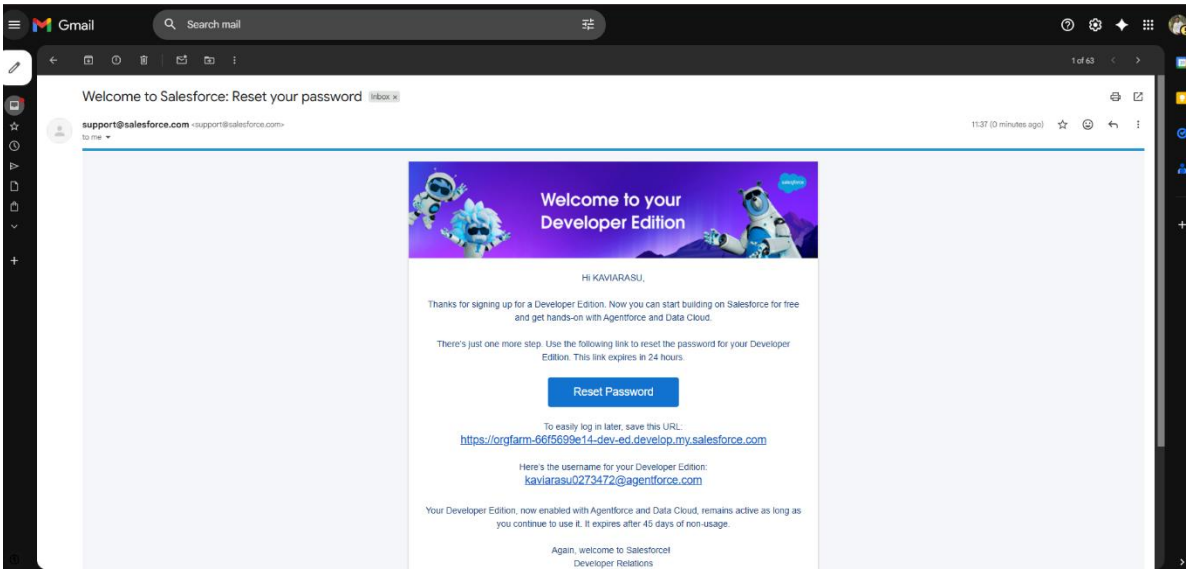


1) First name & Last name
2) Email
3) Job Title: Developer
4) Company: College Name
5) Country: India

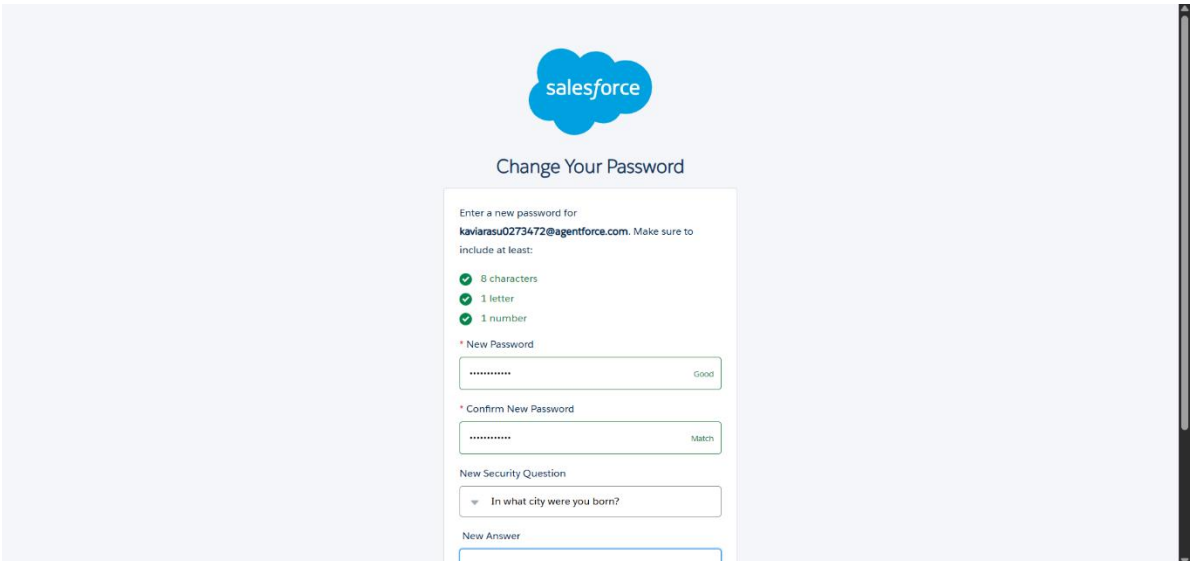
Click on sign me up after filling these.

Activity 2: Account Activation

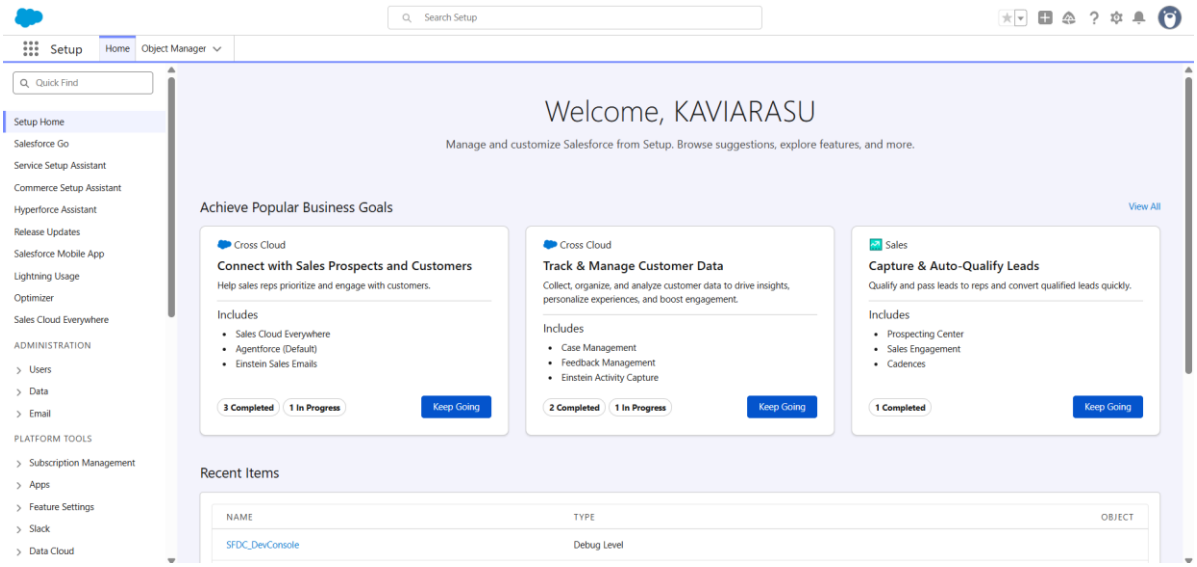
1. Go to the inbox of the email that you used while signing up. Click on the verify account to activate your account. The email may take 10-30mins and sometimes 2 hours.



2. Click on Verify Account
3. Give a password and answer a security question and click on change password.



4. Then you will redirect to your salesforce setup page.

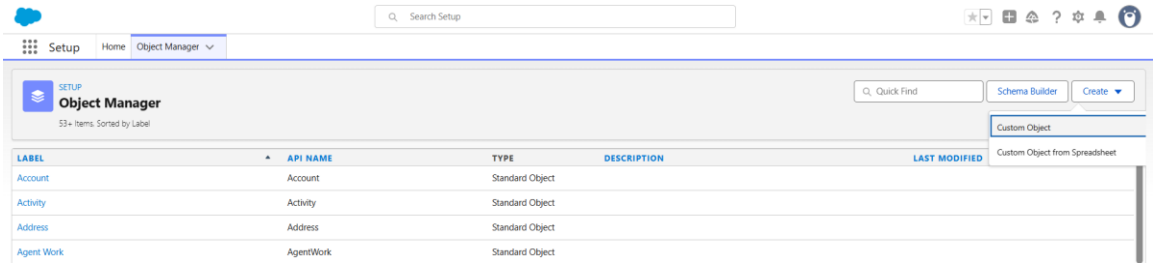


Milestone 2: Objects Creation:

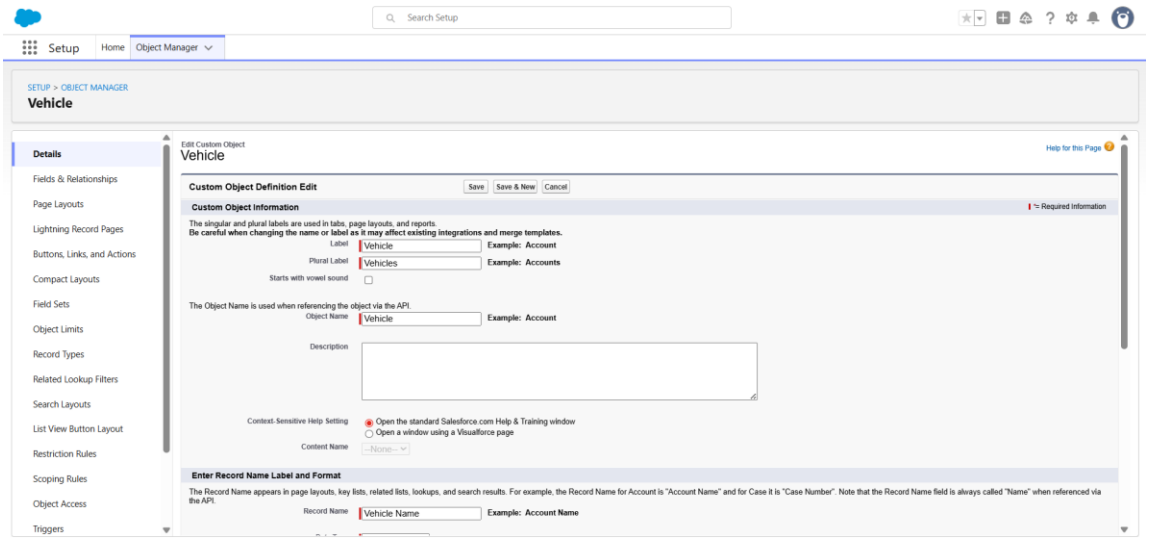
Activity 1: Creating the Vehicle Custom Object

Follow the steps below to create the **Vehicle** object in Salesforce:

1. **From Setup Page**
    - Click on **Setup**.
    - In the Setup page, click on **Object Manager** from the top navigation bar.
2. **Create a New Custom Object**
    - Click on **Create**.
    - Select **Custom Object**.

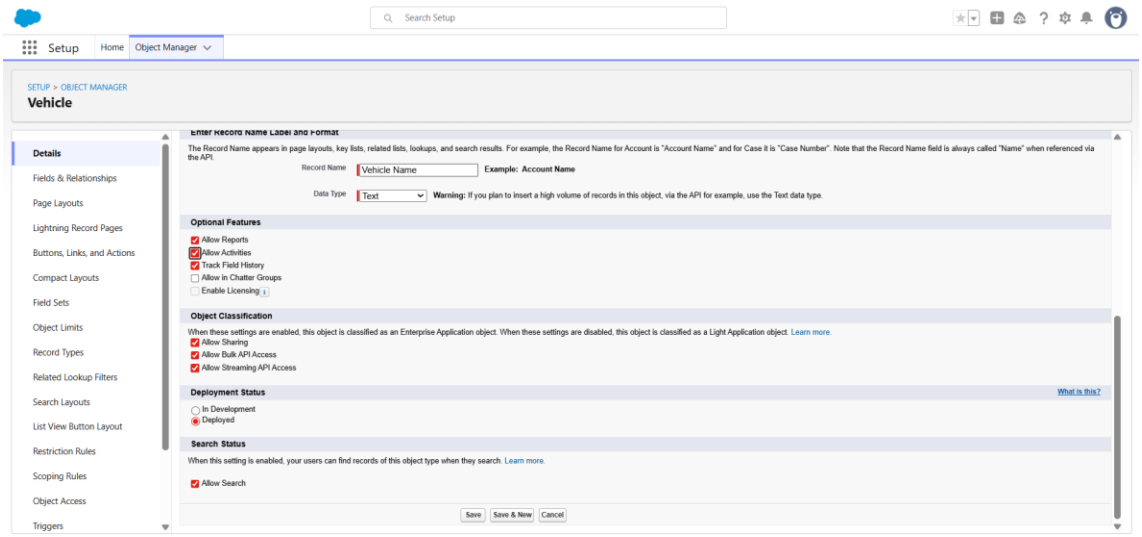


3. **Enter Basic Object Details**
    - **Label**: Vehicle
    - **Plural Label**: Vehicles
    - **Object Name / API Name** will be auto-filled based on the label.



4. **Configure Record Name**
    - In the **Record Name** field, enter: Vehicle Name
    - Set **Data Type** as: Text.

5. **Enable Optional Features**
    - Check **Allow Reports** to include this object in Salesforce reports.
    - Check **Allow Search** to make Vehicle records searchable.
    - Check **Track Field History** to enable field history tracking for this object.
6. **Save the Object**
    - Click on **Save & New** if you want to create another custom object, or click **Save** to finish creating the **Vehicle** object.



Activity 2: Creating Vehicle Dealer Object

Step 1: From Setup Page

- Click on **Setup**.
- In the Setup page, click on **Object Manager** from the top navigation bar.

Step 2: Create a New Custom Object

- Click on **Create**.
- Select **Custom Object**.

Step 3: Enter Basic Object Details

- **Label:** Vehicle Dealer
- **Plural Label:** Vehicle Dealers
- The API Name will be auto-filled.

Step 4: Configure Record Name

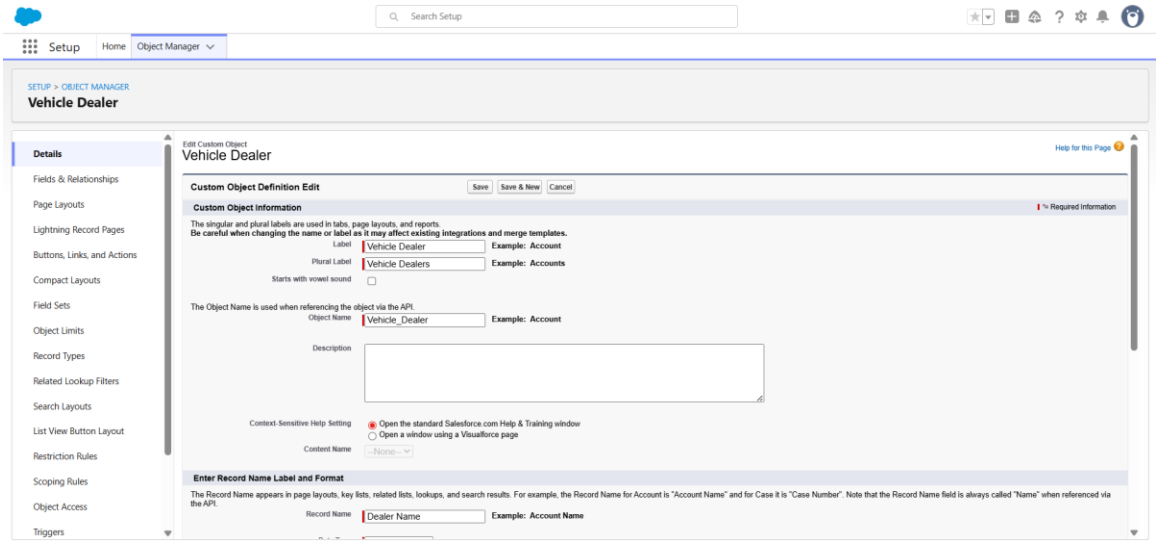
- **Record Name:** Dealer Name
- **Data Type:** Text

Step 5: Enable Optional Features

- Check **Allow Reports**
- Check **Allow Search**
- Check **Track Field History**

Step 6: Save the Object

- Click **Save & New** to create the next object.



Activity 3: Creating Vehicle Customer Object

Step 1: From Setup Page

- Click on **Setup**.
- Open **Object Manager**.

Step 2: Create a New Custom Object

- Click **Create → Custom Object**.

Step 3: Enter Basic Object Details

- **Label:** Vehicle Customer
- **Plural Label:** Vehicle Customers

Step 4: Configure Record Name

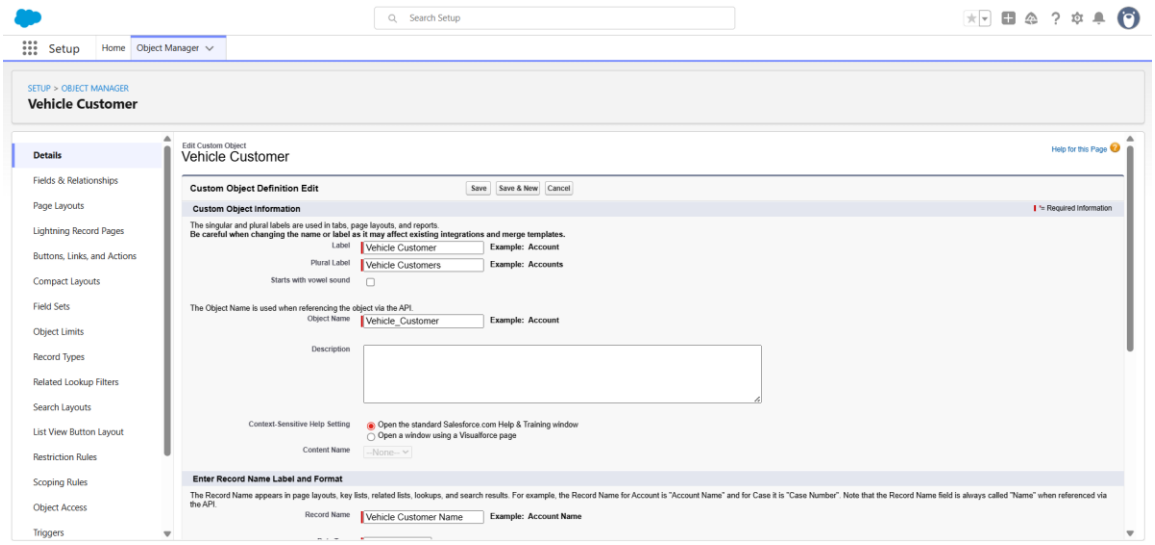
- **Record Name:** Customer Name
- **Data Type:** Text

Step 5: Enable Optional Features

- Check **Allow Reports**
- Check **Allow Search**
- Check **Track Field History**

Step 6: Save the Object

- Click **Save & New**.



Activity 4: Creating Vehicle Order Object

Step 1: From Setup Page

- Click on **Setup → Object Manager**

Step 2: Create a New Custom Object

- Click **Create → Custom Object**

Step 3: Enter Basic Object Details

- **Label:** Vehicle Order
- **Plural Label:** Vehicle Orders

Step 4: Configure Record Name

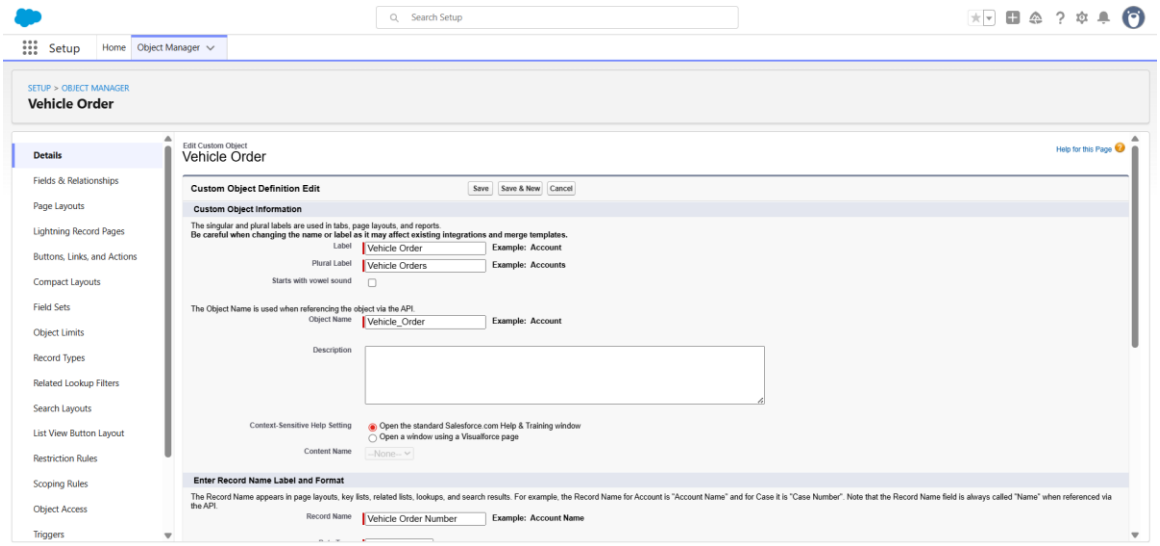
- **Record Name:** Order ID
- **Data Type:** Auto Number
    - Display Format: {0000}
    - Starting Number: **1**

Step 5: Enable Optional Features

- Check **Allow Reports**
- Check **Allow Search**
- Check **Track Field History**

Step 6: Save the Object

- Click **Save & New**.



Activity 5: Creating Vehicle Test Drive Object

Step 1: From Setup Page

- Go to **Setup → Object Manager**

Step 2: Create a New Custom Object

- Click on **Create → Custom Object**

Step 3: Enter Basic Object Details

- **Label:** Vehicle Test Drive
- **Plural Label:** Vehicle Test Drives

Step 4: Configure Record Name

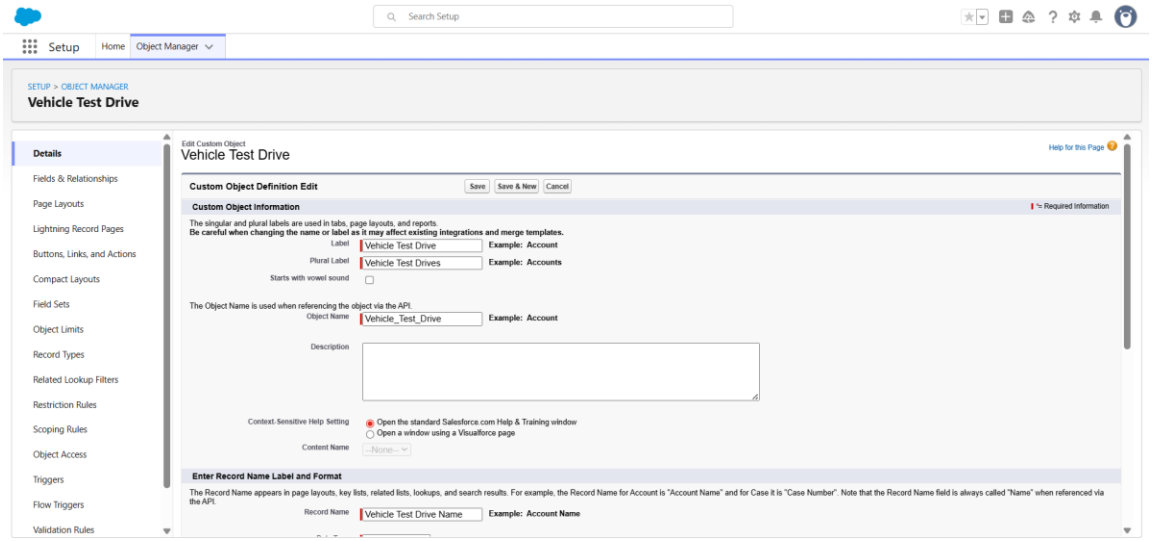
- **Record Name:** Test Drive ID
- **Data Type:** Auto Number
    - Display Format: {0000}
    - Starting Number: **1**

Step 5: Enable Optional Features

- Check **Allow Reports**
- Check **Allow Search**
- Check **Track Field History**

Step 6: Save the Object

- Click **Save & New**.



Activity 6: Creating Vehicle Service Request Object

Step 1: From Setup Page

- Navigate to **Setup → Object Manager**

Step 2: Create a New Custom Object

- Click **Create → Custom Object**

Step 3: Enter Basic Object Details

- **Label:** Vehicle Service Request
- **Plural Label:** Vehicle Service Requests

Step 4: Configure Record Name

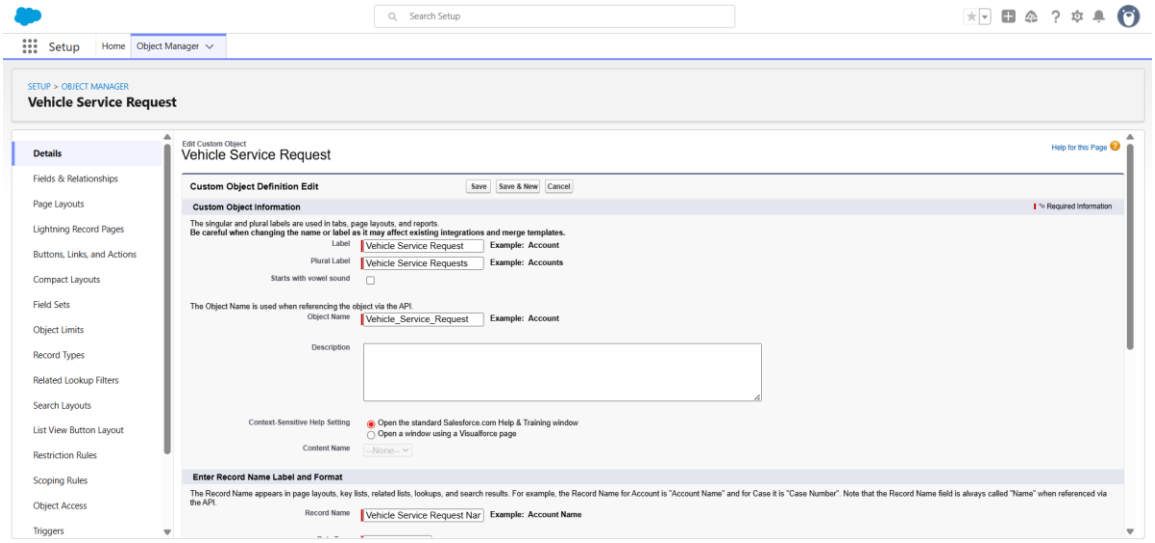
- **Record Name:** Service Request ID
- **Data Type:** Auto Number
    - Display Format: {0000}
    - Starting Number: **1**

Step 5: Enable Optional Features

- Check **Allow Reports**
- Check **Allow Search**
- Check **Track Field History**

Step 6: Save the Object

- Click **Save** to finish.



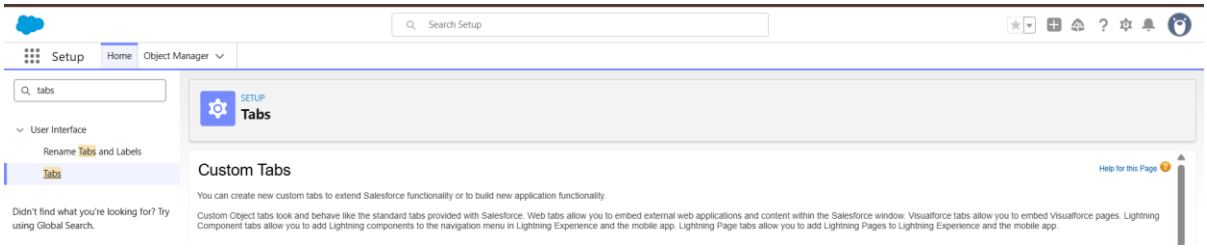
Milestone 3- Tabs

Activity: Creating Tabs for All Objects

Follow the steps below to create tabs for each custom object used in the project.

Step 1: Navigate to Tabs in Setup

- Go to **Setup**.
- In the **Quick Find** box, type **Tabs**.
- Click on **Tabs** from the search results.



Step 2: Create a New Custom Object Tab

- Under the **Custom Object Tabs** section, click on **New**.

Step 3: Select Object and Tab Style

- In the **Object** dropdown, select the required custom object (e.g., Vehicle, Vehicle Dealer, Vehicle Customer, etc.).
- In the **Tab Style** section, choose an appropriate icon and style for the tab.

Step 4: Assign Tab to Profiles

- Click **Next**.

- On the **Add to Profiles** page, keep the default visibility settings as suggested by Salesforce.
  (No changes required if the default configuration is sufficient.)

Step 5: Configure Tab for Apps

- Click **Next**.
- On the **Add to Custom Apps** page, **uncheck** the **Include Tab** option if you do not want the tab to be added automatically to apps.

Step 6: Apply User Customization Setting

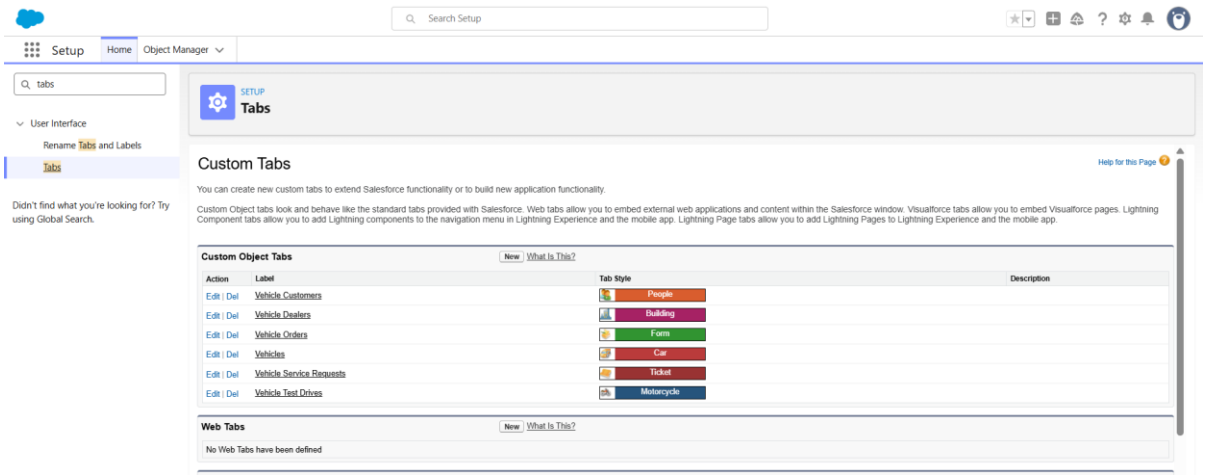
- Ensure that the option **Append tab to the user's existing personal customizations** is **checked**.

Step 7: Save the Tab

- Click **Save** to create the tab for the selected custom object.

Step 8: Repeat for Other Objects

- Repeat the above steps for each custom object created in **Milestone-2** (such as Vehicle, Vehicle Dealer, Vehicle Customer, Vehicle Order, Vehicle Test Drive, and Vehicle Service Request).

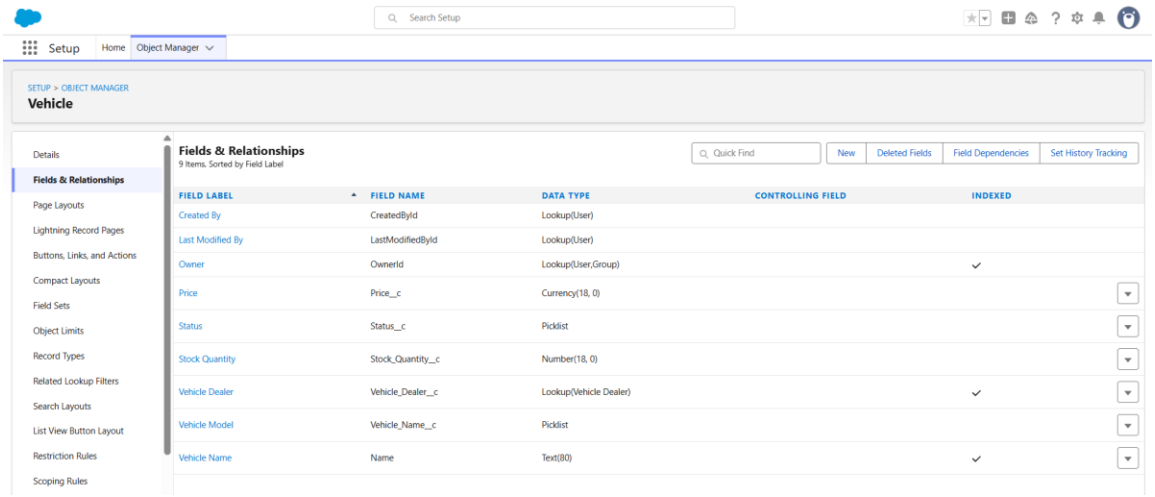


Milestone 4: Fields & Relationships

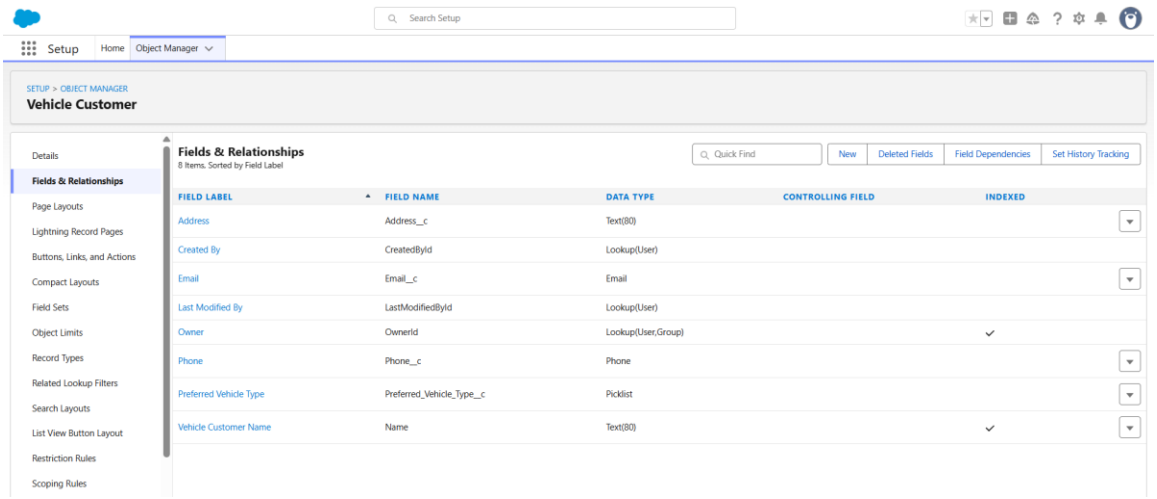
Object Name	Field Name	Data Type	Notes / Relationship
Vehicle__c	Vehicle_Name__c	Text	Record Name
	Vehicle_Model__c	Picklist (Sedan, SUV, EV, etc.)	Model Type
	Stock_Quantity__c	Number	Stock Available
	Price__c	Currency	Vehicle Price
	Dealer__c	Lookup (Vehicle_Dealer__c)	Assigned Dealer
	Status__c	Picklist (Available, Out of Stock, Discontinued)	Vehicle Availability
Vehicle_Dealer__c	Dealer_Name__c	Text	Record Name
	Dealer_Location__c	Text	Dealer Address
	Dealer_Code__c	Auto Number	Unique Dealer ID
	Phone__c	Phone	Dealer Contact
	Email__c	Email	Dealer Email
Vehicle_Order__c	Order ID	Auto Number	Record Name
	Customer__c	Lookup (Vehicle_Customer__c)	Ordered By Customer
	Vehicle__c	Lookup (Vehicle__c)	Ordered Vehicle
	Order_Date__c	Date	Order Date
	Status__c	Picklist (Pending, Confirmed, Delivered, Canceled)	Order Status

Vehicle_Customer__c	Customer_Name__c	Text	Record Name
	Email__c	Email	Customer Email
	Phone__c	Phone	Customer Phone
	Address__c	Text	Customer Address
	Preferred_Vehicle_Type__c	Picklist (Sedan, SUV, EV, etc.)	Vehicle Preference
Vehicle_Test_Drive__c	Test Drive ID	Auto Number	Record Name
	Customer__c	Lookup (Vehicle_Customer__c)	Test Drive Customer
	Vehicle__c	Lookup (Vehicle__c)	Test Drive Vehicle
	Test_Drive_Date__c	Date	Scheduled Date
	Status__c	Picklist (Scheduled, Completed, Canceled)	Test Drive Status
Vehicle_Service_Request__c	Service Request ID	Auto Number	Record Name
	Customer__c	Lookup (Vehicle_Customer__c)	Service Requested By
	Vehicle__c	Lookup (Vehicle__c)	Vehicle Under Service
	Service_Date__c	Date	Service Requested Date
	Issue_Description__c	Text	Problem Details
	Status__c	Picklist (Requested, In Progress, Completed)	Service Status

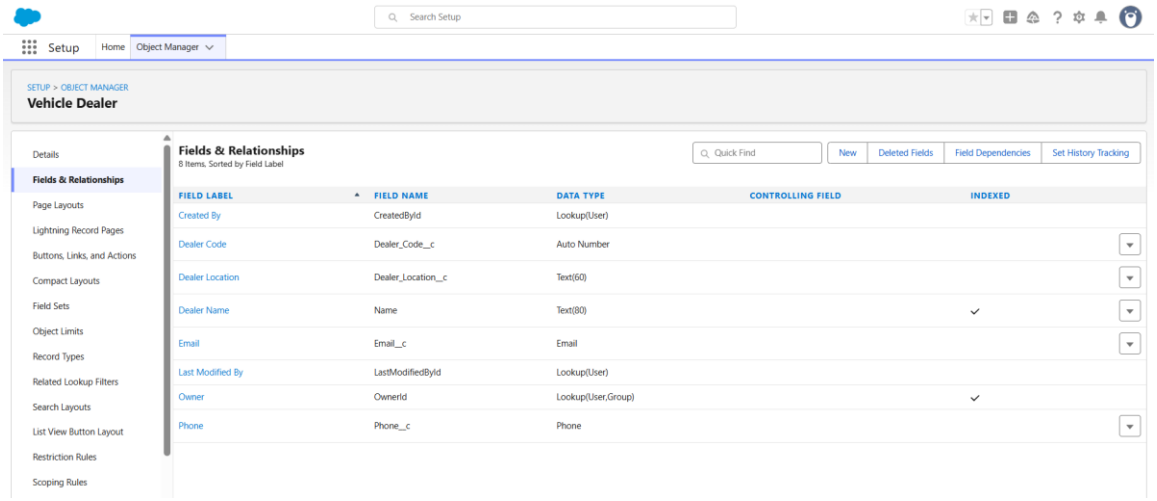
Fields Created in Vehicle Object:



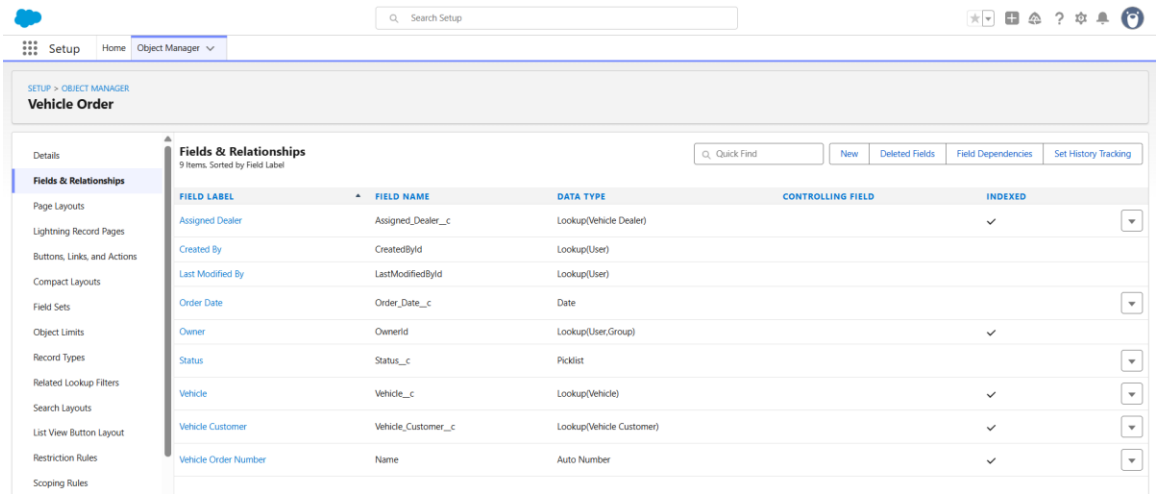
Fields Created in Vehicle Object:



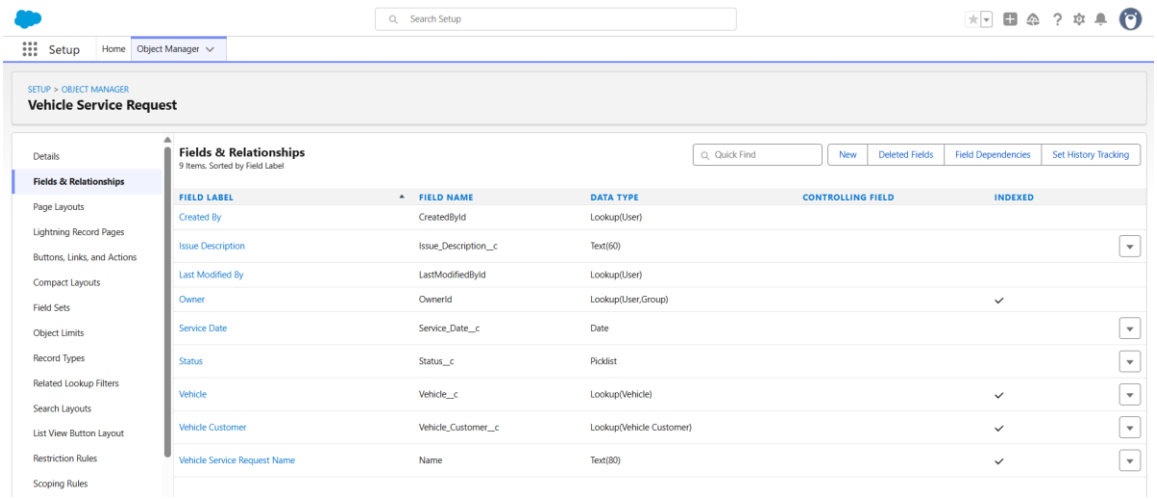
Fields Created in Vehicle Dealer Object:



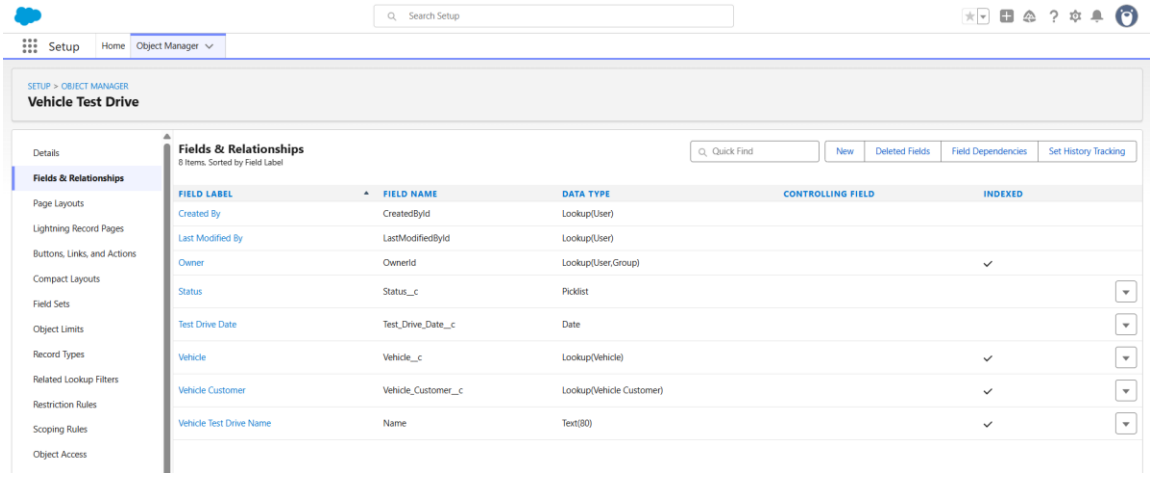
Fields Created in Vehicle Order Object:



Fields Created in Vehicle Service Request Object:



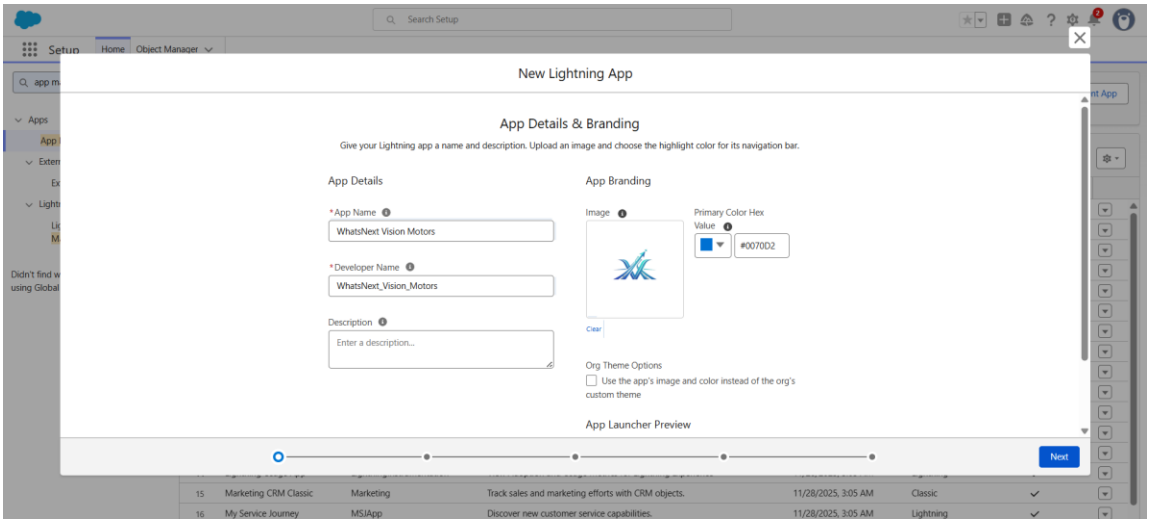
Fields Created in Vehicle Test Drive Object:



Milestone 5 – Create a Lightning App :- WhatNext Vision Motors

Step 1: Create a New Lightning App

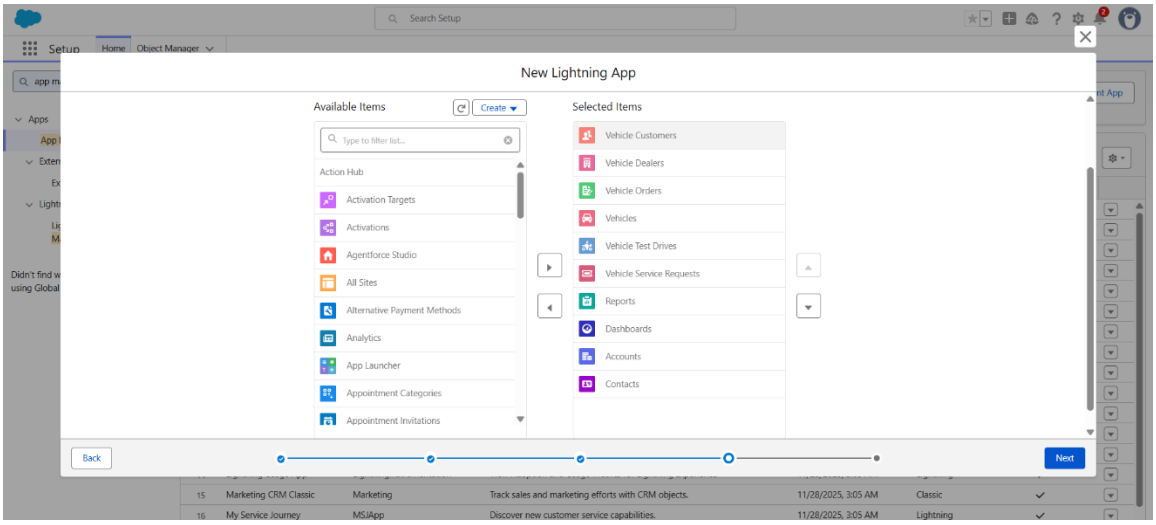
- Go to **Setup**.
- In the Setup menu, select **App Manager**.
- Click on **New Lightning App**.
- Enter the **App Name**: **WhatsNext Vision Motors.**
- Add a description (optional image can be uploaded if needed).



- Click **Next**, then **Next**, then **Next** again to continue with the default settings.

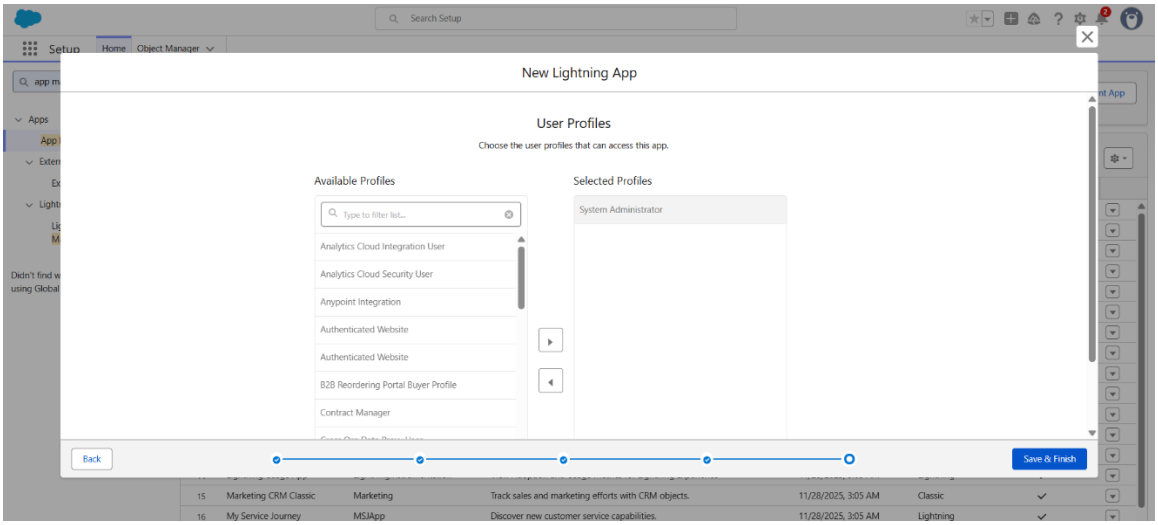
Step 2: Add Navigation Items

- Search for and add the following items to the app navigation:
  - **Vehicle**
  - **Vehicle Dealer**
  - **Vehicle Customer**
  - **Vehicle Order**
  - **Vehicle Test Drive**
  - **Vehicle Service Request**
  - **Reports**
  - **Dashboards**
- Click **Next** to proceed.



Step 3: Assign User Profiles

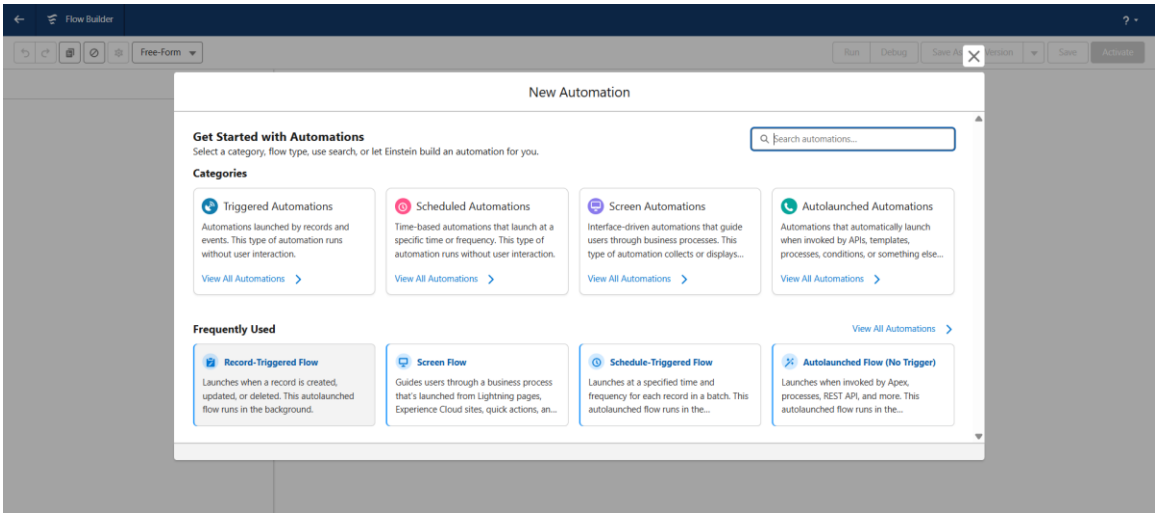
- In the profile selection screen, search for **System Administrator**.
- Move it to the **Selected Profiles** list.
- Click **Save & Finish** to complete the app setup.
-



Milestone 6 – Record-Triggered Flow to Auto-Assign Nearest Dealer

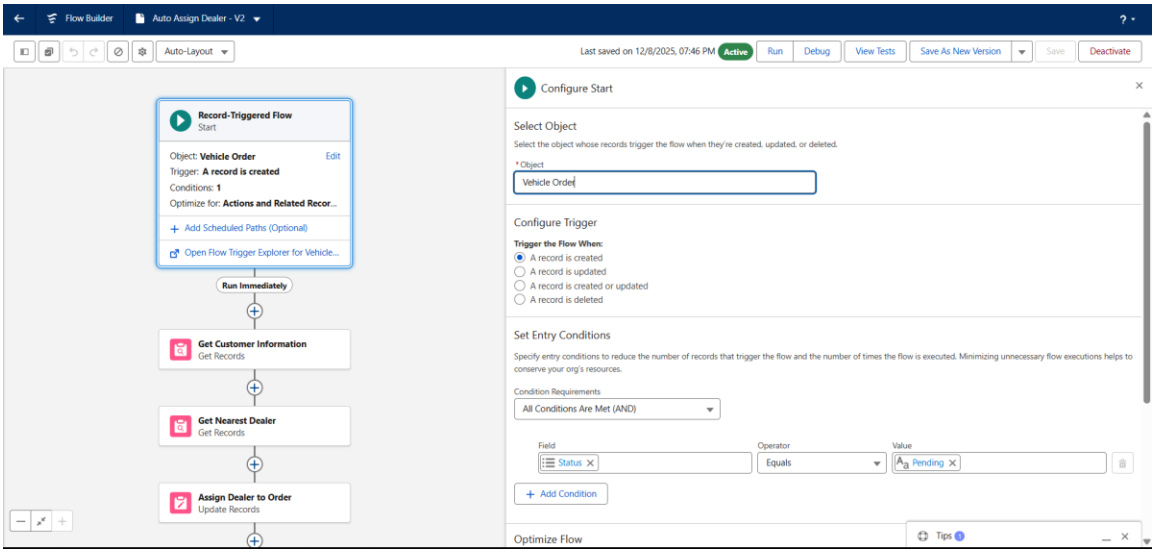
Step 1: Create the Flow

- Go to **Setup**.
- In the **Quick Find** search box, type **Flows**.
- Click **Flows → New Flow**.
- Select **Start from Scratch**, then click **Next**.
- Choose **Record-Triggered Flow**, then click **Create**.



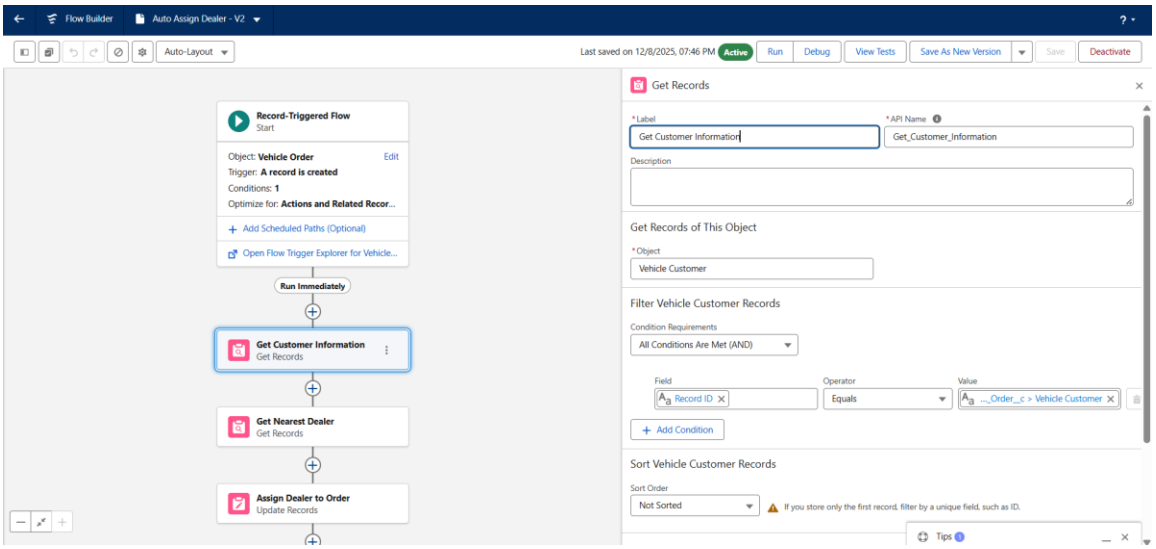
Step 2: Configure the Trigger

- **Object:** Vehicle Order
- **Trigger Condition:** When a record is created
- **Entry Criteria:**
  - **Field:** Status__c
  - **Operator:** Equals
  - **Value:** Pending
- **Condition Logic:** All Conditions Are Met (AND)



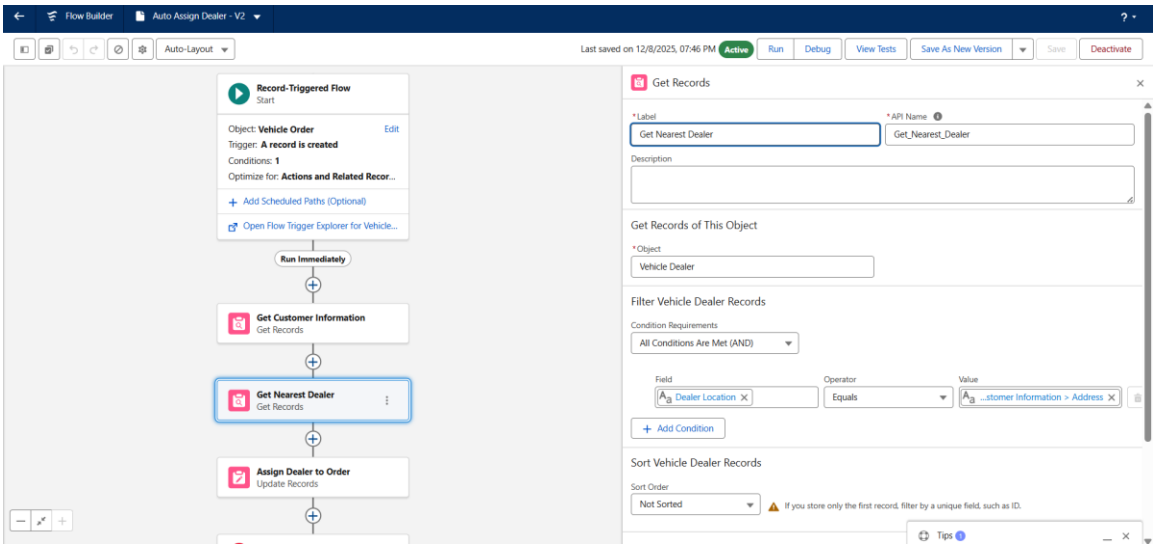
Step 3: Get Customer Details

- Click + → Select **Get Records**.
- **Label:** Get Customer Information
- **Object:** Vehicle Customer
- **Filter Condition:**
  - **Id = {!$Record.Vehicle_Customer__c}**
- **Record Retrieval:**
  - Select **Only the first record**
  - Choose **Automatically store all fields**



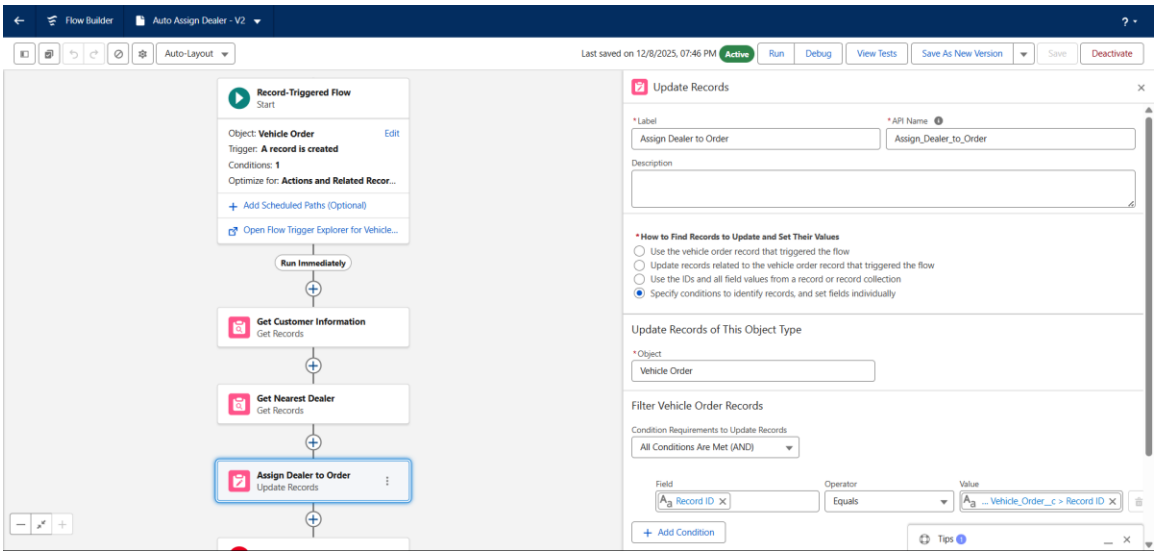
Step 4: Get Nearest Dealer

- Click + → Select **Get Records**.
- **Label:** Get Nearest Dealer
- **Object:** Vehicle Dealer
- **Filter Condition:**
  - **Dealer_Location__c = {!Get_Customer_Information.Address__c}**
- **Record Retrieval:**
  - Select **Only the first record**
  - Choose **Automatically store all fields**



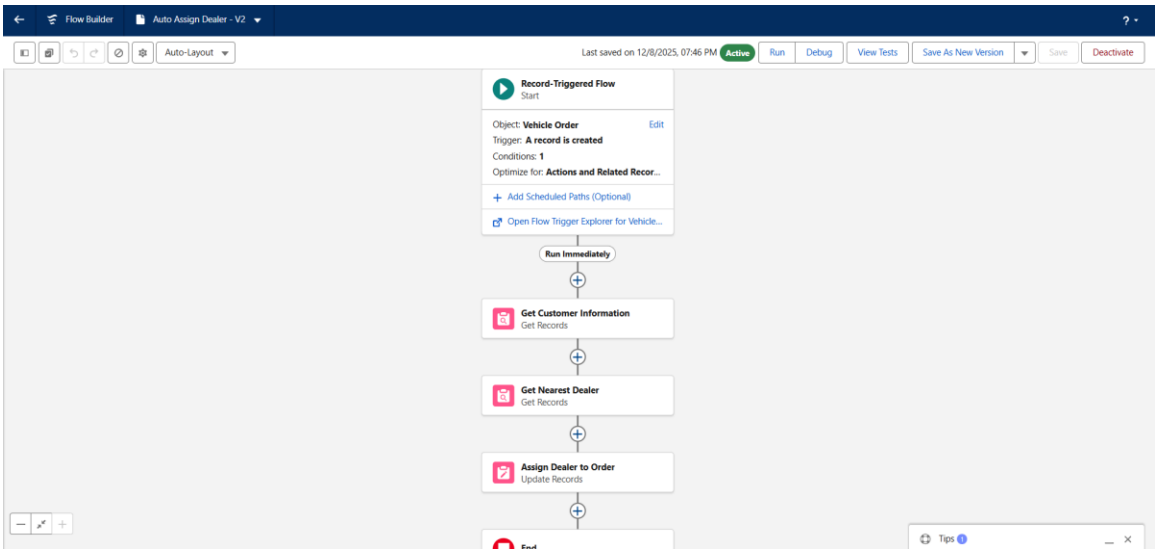
Step 5: Assign Dealer to the Order

- Click + → Select **Update Records**.
- **Label:** Assign Dealer to Order
- **Choose Update Method:**
  - Select **Use the IDs and all field values from a record**
- **Record to Update:**
  - Set value to **{!Get_Nearest_Dealer}**



Step 6: Save and Activate the Flow

- Enter Flow Name: **Auto Assign Dealer**
- Click **Save**.
- Click **Activate** to enable the flow.



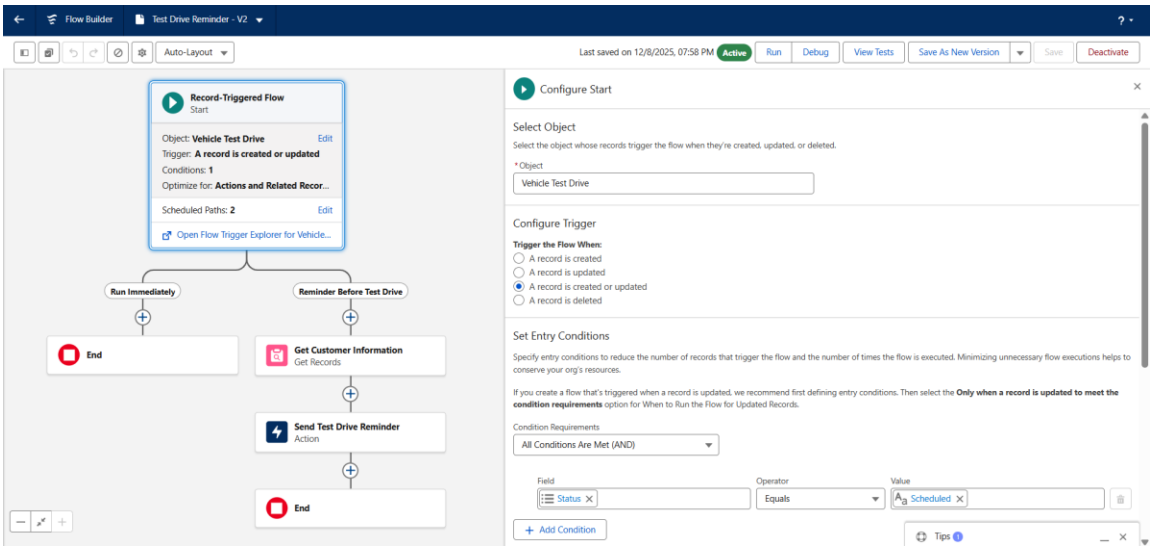
Milestone 6 (Flow – 2) – Record-Triggered Flow to Send Test Drive Reminder Email

Step 1: Create the Flow

- Go to **Setup**.
- In the **Quick Find** box, type **Flows** and click **Flows**.
- Click **New Flow**.
- Select **Record-Triggered Flow** and click **Create**.

Step 2: Configure the Trigger

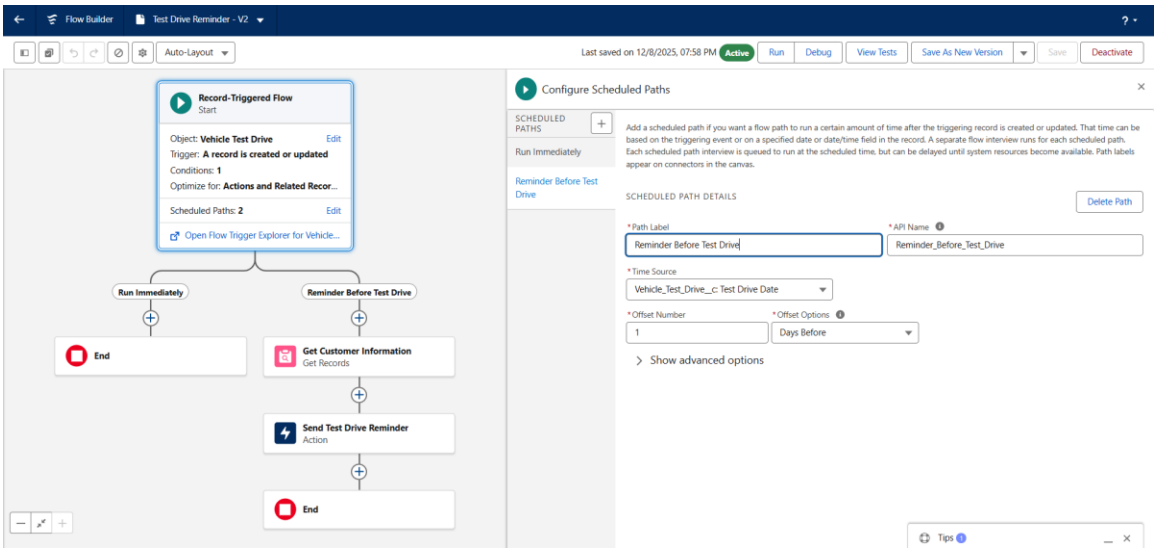
- **Object:** Vehicle Test Drive
- **Trigger the Flow When:** A record is created or updated
- **Entry Conditions:**
  - **Field:** Status__c
  - **Operator:** Equals
  - **Value:** Scheduled
- **Condition Logic:** All Conditions Are Met (AND)



Step 3: Create Scheduled Path

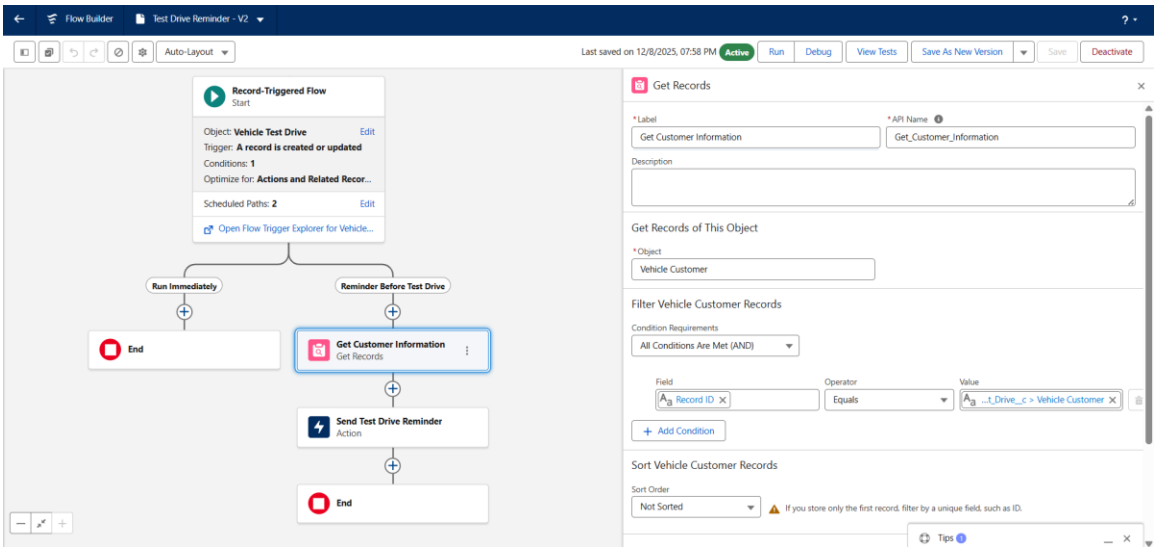
- In the Start element, click + **Add Scheduled Paths** (below the trigger).
- **Label:** Reminder Before Test Drive
- **Time Source:** Test_Drive_Date__c
- **Offset Number:** 1
- **Offset Option: Days Before**

- Click **Done** to save the scheduled path.



Step 4: Get Customer Details

- Click on the + icon along the **Scheduled Path**.
- Select **Get Records**.
- **Label:** Get Customer Information
- **Object:** Vehicle Customer
- **Filter Condition:**
  - **Id = {!$Record.Customer__c}**
- **Record Retrieval:**
  - Select **Only the first record**
  - Choose **Automatically store all fields**

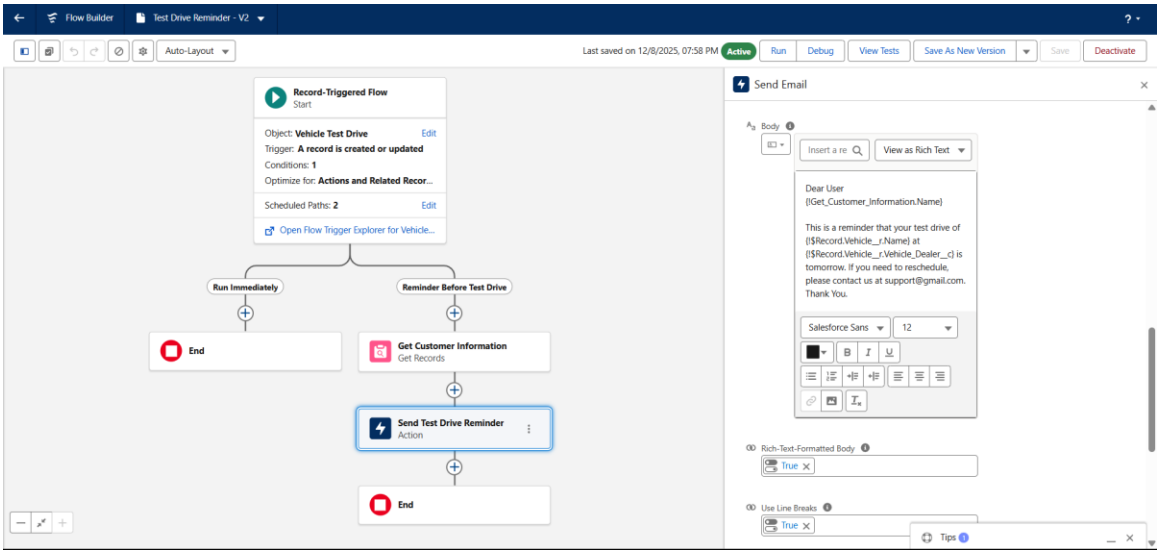


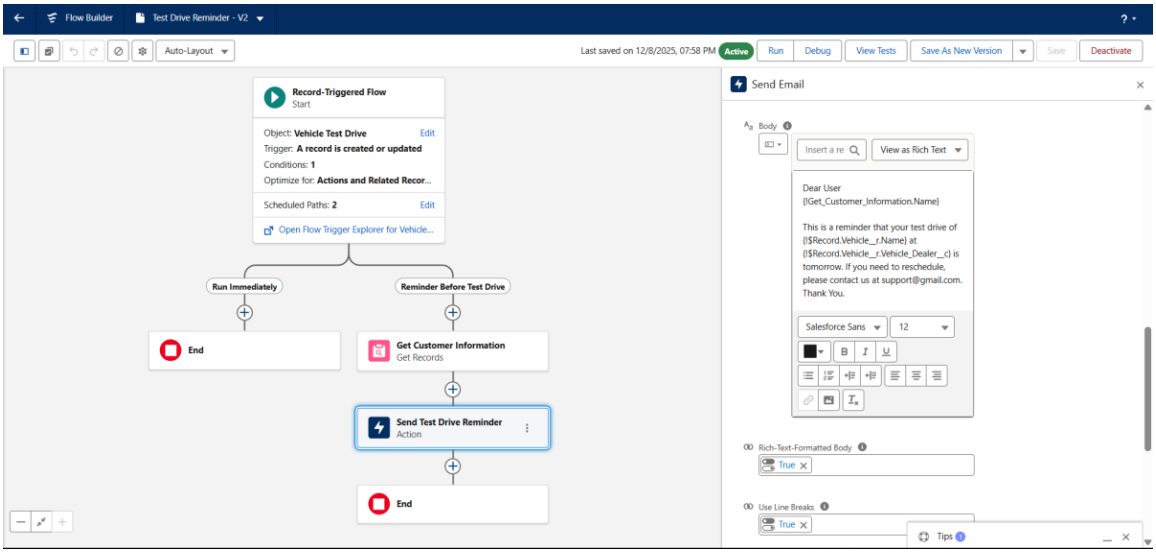
Step 5: Send Reminder Email

- Click + after the **Get Customer Information** element.
- Select **Action**.
- **Action Type:** Send Email
- **Label:** Send Test Drive Reminder
- **Subject:** "Reminder: Your Test Drive is Tomorrow!"
- **Recipient Address:** {!Get_Customer_Information.Email__c}
- Enable **Rich Text Body**.
- Set **Body Variable API Name:** EmailSent
- **Email Body:**

  Dear User {!Get_Customer_Information.Name}

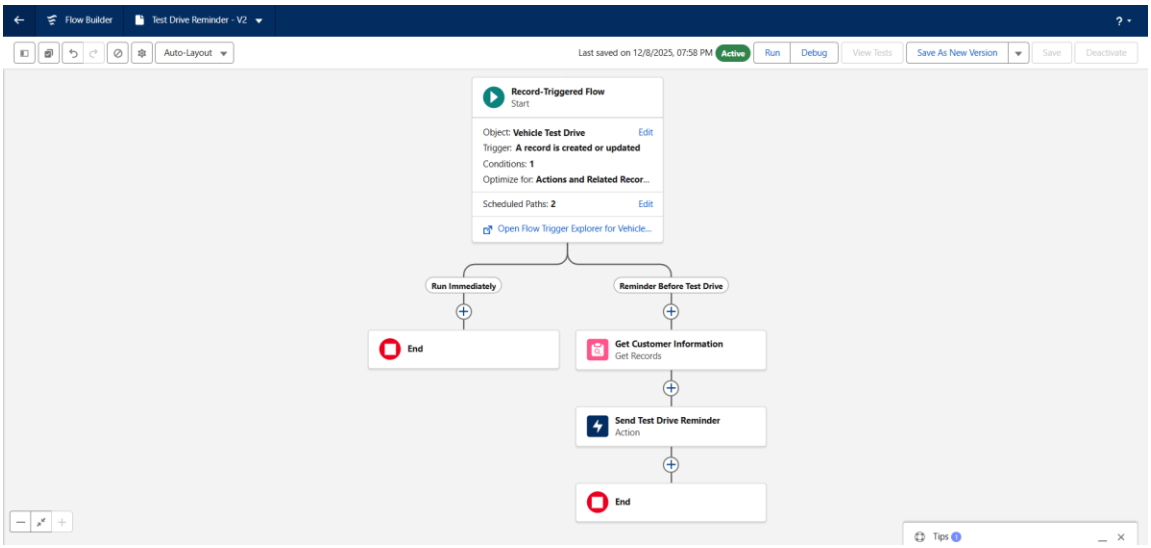
  This is a reminder that your test drive of {!$Record.Vehicle__r.Name} at {!$Record.Vehicle__r.Vehicle_Dealer__c} is tomorrow. If you need to reschedule, please contact us at support@gmail.com.
  Thank You.





Step 6: Save and Activate the Flow

- Click **Save**.
- Enter **Flow Label:** Test Drive Reminder
- Click **Activate** to enable the reminder flow.



Milestone 7 – Apex Trigger, Batch Job & Scheduler with Full Code

Step 1: Open Developer Console

- Click the **Gear** icon in Salesforce (top-right corner).
- Select **Developer Console** from the dropdown.

Step 2: Create a New Apex Class

- In the Developer Console, click **File → New → Apex Class**.
- In the popup, enter the class name: **VehicleOrderTriggerHandler**.
- Click **OK** to create the class.

Step 3: Add Code for the Trigger Handler Class

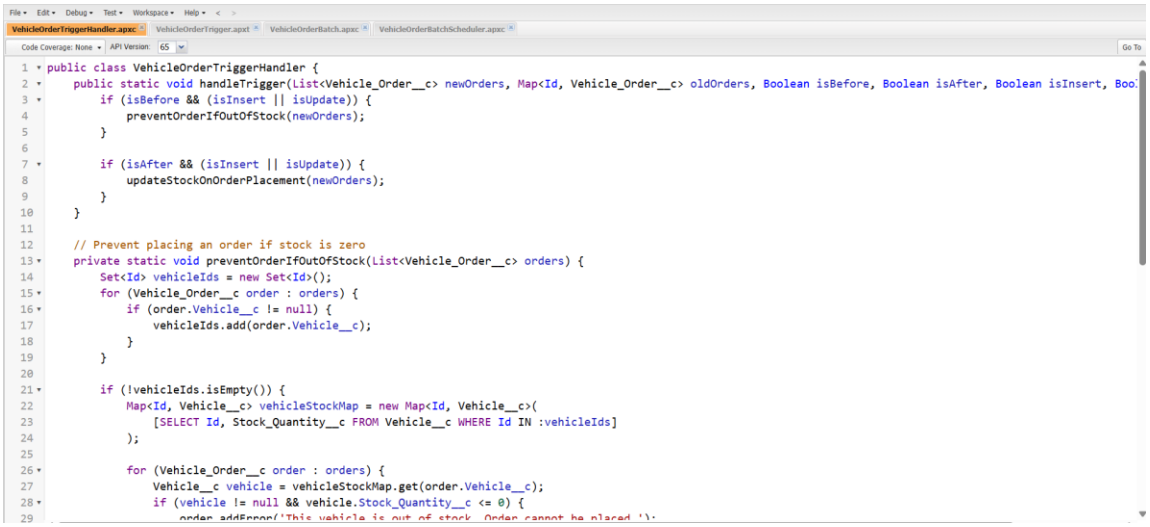
- Replace the default template with the following code:

```
public class VehicleOrderTriggerHandler {
    public static void handleTrigger(
        List<Vehicle_Order__c> newOrders,
            Map<Id, Vehicle_Order__c> oldOrders, Boolean isBefore, Boolean
      isAfter, Boolean isInsert, Boolean isUpdate) {
        if (isBefore) {
            if (isInsert || isUpdate) {
                preventOrderIfOutOfStock(newOrders);
            }
        }
        if (isAfter) {
            if (isInsert || isUpdate) {
                updateStockOnOrderPlacement(newOrders);
            }
        }
    }
    // Method to prevent orders when the vehicle is out of stock
    private static void preventOrderIfOutOfStock(List<Vehicle_Order__c> orders) {
        Set<Id> vehicleIds = new Set<Id>();
        for (Vehicle_Order__c order : orders) {
            if (order.Vehicle__c != null) {
                vehicleIds.add(order.Vehicle__c);
            }
        }
        if (!vehicleIds.isEmpty()) {
            Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>();
```

```
        for (Vehicle__c vehicle : [
            SELECT Id, Stock_Quantity__c
            FROM Vehicle__c
            WHERE Id IN :vehicleIds
        ]) {
            vehicleStockMap.put(vehicle.Id, vehicle);
        }
        for (Vehicle_Order__c order : orders) {
            if (vehicleStockMap.containsKey(order.Vehicle__c)) {
                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
                if (vehicle.Stock_Quantity__c <= 0) {
                    order.addError('This vehicle is out of stock. Order cannot be placed.');
                }
            }
        }
    }
    // Method to update vehicle stock when an order is placed
    private static void updateStockOnOrderPlacement(List<Vehicle_Order__c> orders) {
        Set<Id> vehicleIds = new Set<Id>();
        for (Vehicle_Order__c order : orders) {
            if (order.Vehicle__c != null && order.Status__c == 'Confirmed') {
                vehicleIds.add(order.Vehicle__c);
            }
        }
        if (!vehicleIds.isEmpty()) {
            Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>();
            for (Vehicle__c vehicle : [
                SELECT Id, Stock_Quantity__c
                FROM Vehicle__c
                WHERE Id IN :vehicleIds
            ]) {
                vehicleStockMap.put(vehicle.Id, vehicle);
            }
            List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();
```

```
            for (Vehicle_Order__c order : orders) {
                if (vehicleStockMap.containsKey(order.Vehicle__c)) {
                    Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
                    if (vehicle.Stock_Quantity__c > 0) {
                        vehicle.Stock_Quantity__c -= 1;
                        vehiclesToUpdate.add(vehicle);
                    }
                }
            }
            if (!vehiclesToUpdate.isEmpty()) {
                update vehiclesToUpdate;
            }
        }
    }
}
```

- Click **File → Save**.



```
public class VehicleOrderTriggerHandler {
    public static void handleTrigger(List<Vehicle_Order__c> newOrders, Map<Id, Vehicle_Order__c> oldOrders, Boolean isBefore, Boolean isAfter, Boolean isInsert, Boo
        if (isBefore && (isInsert || isUpdate)) {
            preventOrderIfOutOfStock(newOrders);
        }

        if (isAfter && (isInsert || isUpdate)) {
            updateStockOnOrderPlacement(newOrders);
        }
    }

    // Prevent placing an order if stock is zero
    private static void preventOrderIfOutOfStock(List<Vehicle_Order__c> orders) {
        Set<Id> vehicleIds = new Set<Id>();
        for (Vehicle_Order__c order : orders) {
            if (order.Vehicle__c != null) {
                vehicleIds.add(order.Vehicle__c);
            }
        }

        if (!vehicleIds.isEmpty()) {
            Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>(
                [SELECT Id, Stock_Quantity__c FROM Vehicle__c WHERE Id IN :vehicleIds]
            );

            for (Vehicle_Order__c order : orders) {
                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
                if (vehicle != null && vehicle.Stock_Quantity__c <= 0) {
```

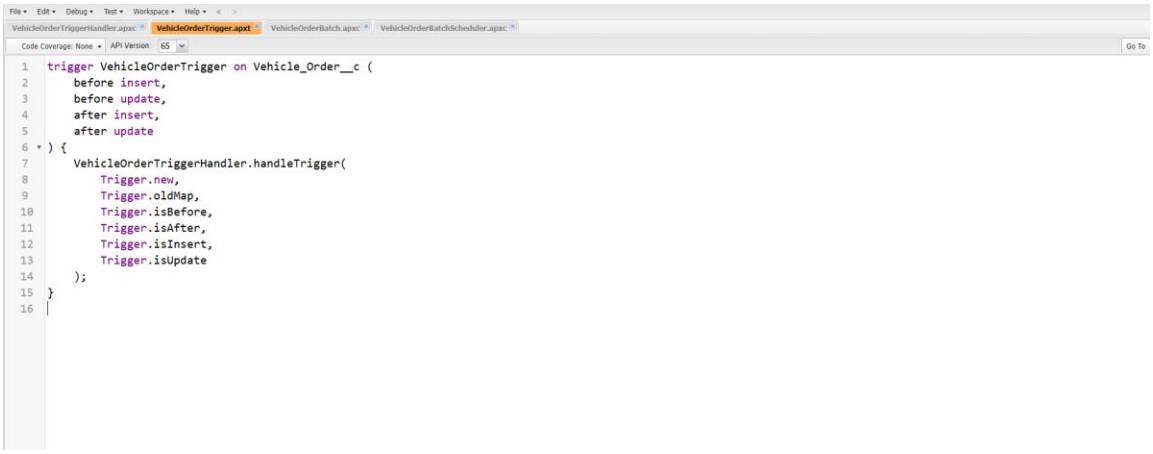
Step 4: Create the Trigger on Vehicle Order

- In Developer Console, click **File → New → Apex Trigger**.
- Enter **Trigger Name:** VehicleOrderTrigger.
- Select **sObject:** Vehicle_Order__c.
- Click **Submit**.

Step 5: Add Trigger Code to Call the Handler

trigger VehicleOrderTrigger on Vehicle_Order__c (before insert, before update, after insert, after update) {
VehicleOrderTriggerHandler.handleTrigger(Trigger.new,Trigger.oldMap, Trigger.isBefore, Trigger.isAfter, Trigger.isInsert, Trigger.isUpdate);
}

- Click **File → Save** to save the trigger.



Step 6: Create the Batch Apex Class

- In Developer Console, click **File → New → Apex Class**.
- Enter class name: **VehicleOrderBatch**.
- Click **OK**.
- Paste the following code:

global class VehicleOrderBatch implements Database.Batchable<sObject> {
global Database.QueryLocator start(Database.BatchableContext bc) {
return Database.getQueryLocator([
SELECT Id, Status__c, Vehicle__c
FROM Vehicle_Order__c
WHERE Status__c = 'Pending'
]);
}
global void execute(Database.BatchableContext bc, List<Vehicle_Order__c> orderList) {
Set<Id> vehicleIds = new Set<Id>();
for (Vehicle_Order__c order : orderList) {

```
            if (order.Vehicle__c != null) {
                vehicleIds.add(order.Vehicle__c);
            }
        }
        if (!vehicleIds.isEmpty()) {
            Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>([
                SELECT Id, Stock_Quantity__c
                FROM Vehicle__c
                WHERE Id IN :vehicleIds
            ]);
            List<Vehicle_Order__c> ordersToUpdate = new
List<Vehicle_Order__c>();
            List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();
            for (Vehicle_Order__c order : orderList) {
                if (vehicleStockMap.containsKey(order.Vehicle__c)) {
                    Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
                    if (vehicle.Stock_Quantity__c > 0) {
                        order.Status__c = 'Confirmed';
                        vehicle.Stock_Quantity__c -= 1;
                        ordersToUpdate.add(order);
                        vehiclesToUpdate.add(vehicle);
                    }
                }
            }
            if (!ordersToUpdate.isEmpty()) {
                update ordersToUpdate;
            }
            if (!vehiclesToUpdate.isEmpty()) {
                update vehiclesToUpdate;
            }
        }
    }

    global void finish(Database.BatchableContext bc) {
        System.debug('Vehicle order batch job completed.');
    }
}
```

- Click **File → Save**.

```
global class VehicleOrderBatch implements Database.Batchable<sObject> {

    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([
            SELECT Id, Status__c, Vehicle__c FROM Vehicle_Order__c WHERE Status__c = 'Pending'
        ]);
    }

    global void execute(Database.BatchableContext bc, List<Vehicle_Order__c> orderList) {
        Set<Id> vehicleIds = new Set<Id>();
        for (Vehicle_Order__c order : orderList) {
            if (order.Vehicle__c != null) {
                vehicleIds.add(order.Vehicle__c);
            }
        }

        if (!vehicleIds.isEmpty()) {
            Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>(
                [SELECT Id, Stock_Quantity__c FROM Vehicle__c WHERE Id IN :vehicleIds]
            );

            List<Vehicle_Order__c> ordersToUpdate = new List<Vehicle_Order__c>();
            List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();

            for (Vehicle_Order__c order : orderList) {
                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
                if (vehicle != null && vehicle.Stock_Quantity__c > 0) {
                    order.Status__c = 'Confirmed';
                    vehicle.Stock_Quantity__c -= 1;
```

Step 7: Create the Schedule Class

- In Developer Console, click **File → New → Apex Class**.
- Enter class name: **VehicleOrderBatchScheduler**.
- Click **OK**.
- Paste the following code:

```
global class VehicleOrderBatchScheduler implements Schedulable {

    global void execute(SchedulableContext sc) {
        VehicleOrderBatch batchJob = new VehicleOrderBatch();
        Database.executeBatch(batchJob, 50); // 50 is the batch size
    }
}
```

- Click **File → Save**.

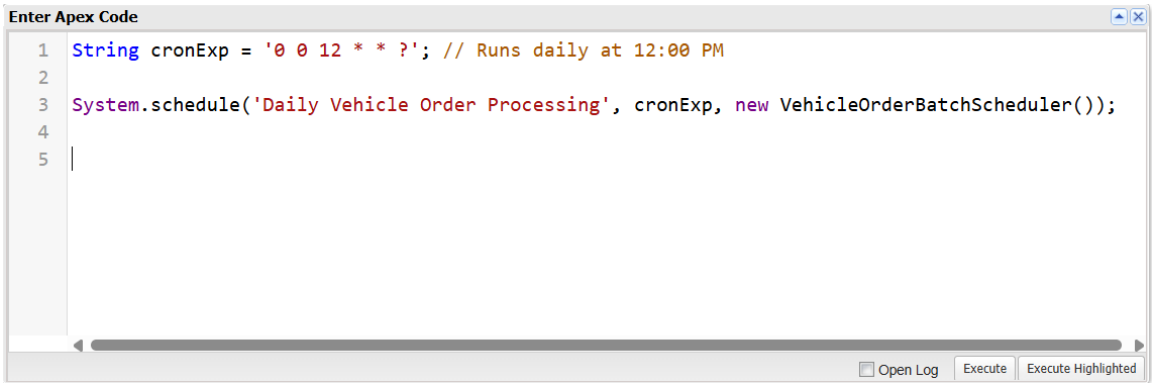
```
global class VehicleOrderBatchScheduler implements Schedulable {
    global void execute(SchedulableContext sc) {
        VehicleOrderBatch batchJob = new VehicleOrderBatch();
        Database.executeBatch(batchJob, 50); // 50 = batch size
    }
}
```

Step 8: Schedule the Batch Job Using Cron Expression

- Open **Execute Anonymous Window** in Developer Console (**Debug → Open Execute Anonymous Window**).
- Enter the following code to schedule the job to run daily at 12:00 PM:

```
String cronExp = '0 0 12 * * ?'; // Runs daily at 12:00 PM
System.schedule('Daily Vehicle Order Processing', cronExp, new VehicleOrderBatchScheduler());
```

- Check **Open Log** if you want to see the execution log.
- Click **Execute**.



Step 9: Functional Behavior of the Batch Job (Documentation Note)

- When a customer places an order and the vehicle is out of stock, the order remains in **Pending** status.
- After new stock is added for that vehicle, the **VehicleOrderBatch** job:
  - Checks all **Pending** orders.
  - Confirms orders where stock is available.
  - Reduces the corresponding vehicle stock quantity.

Phase 3: UI/UX Development & Customization

UI/UX (User Interface and User Experience) is essential in making a CRM system intuitive, efficient, and easy to use for daily business activities. In the WhatsNext Vision Motors CRM solution, the interface was crafted to prioritize clarity, quick access, and effortless navigation. This ensures that users like sales teams, dealership staff, and customer service personnel can operate the system without complications.

This phase focuses on building a clean and well-organized interface using Salesforce Lightning tools such as Lightning App Builder, custom components, object tabs, and optimized page layouts. The objective was to deliver a smooth workflow for viewing vehicle records, searching customer details, assigning dealers, managing orders, scheduling test drives, and tracking service requests—resulting in a consistent and user-friendly experience across the entire platform.

Creation of Lightning App – "WhatNext Vision Motors CRM"

A custom Lightning App named "WhatNext Vision Motors CRM" was created to provide a centralized entry point for all modules. The app includes branding, navigation items, utilities, and profile access criteria.

The Lightning App contains navigation items representing all major object modules:

- Vehicles
- Vehicle Dealers
- Vehicle Customers
- Vehicle Orders
- Vehicle Test Drives
- Vehicle Service Requests
- Reports
- Dashboards

This ensures that users can access every CRM feature from a single application interface without searching through the App Launcher.

Navigation & User Flow

When a user opens the Lightning App, the CRM navigation bar clearly organizes the complete business process:

1. Viewing vehicles in stock
2. Viewing list of dealers and their locations
3. Accessing customer details
4. Managing orders and checking delivery status
5. Scheduling or tracking test drives

6. Recording and monitoring service requests

Each navigation item was placed in the most logical sequence to match an actual automotive customer-sales lifecycle. This enhances user workflow memory and reduces learning time for new users.

Designing with Salesforce Lightning Experience

To improve usability and aesthetics, the CRM interface includes:

- Card-style layout sections
- Icons and colors aligned with automotive brand theme
- Minimum scroll design
- Field grouping based on workflow logic

The Lightning framework ensures users complete tasks with fewer clicks — for example:

- From a vehicle record, users can directly open "Related Orders"
- From the customer page, users can create a "New Vehicle Order" with a single button click
- From order record, users can review stock details and dealer information instantly

Tab Creation for CRM Objects

Tabs were created for all custom objects:

- Vehicle
- Vehicle Dealer
- Vehicle Customer
- Vehicle Order
- Vehicle Test Drive
- Vehicle Service Request

Benefits of Tabs:

- Faster access to records for all user roles
- No need to search through multiple apps
- Easier navigation during live customer interactions or support calls

UI/UX Benefits After Implementation

Improvement	Impact
Fast navigation & visibility	Faster customer service and sales
Organized page layouts	Reduced data entry errors
Dynamic forms	Only relevant fields are shown to users
Clear tab-based access	Higher productivity and ease of use
Card-based screen design	Clean and modern UI experience
Lightning App centralized entry	No confusion for first-time users

Overall Summary of Phase 3

The UI/UX customization transforms the CRM from a simple data storage tool into a powerful business-friendly sales portal. The system is designed so that a sales executive, customer support team, or dealer coordinator can perform daily tasks in the least number of clicks while maintaining complete data accuracy.

With the Lightning App, optimized layouts, dynamic forms and Salesforce standard components, the CRM delivers:

- Smooth navigation
- Faster order management
- Clear visibility of vehicle stock
- High adoption rate among users
- Professional and premium interface experience

Phase 4: Testing & Security:

Security settings were configured to ensure that data access is appropriately controlled and aligned with the functional requirements of the project.

1. **Organization-Wide Defaults (OWD)**

OWD settings were applied to the key custom objects involved in order and service processing:

- **Vehicle Order** → *Private*
- **Vehicle Test Drive** → *Private*
- **Vehicle Service Request** → *Private*

This ensures that only record owners and users with higher-level permissions (such as administrators or assigned dealers) can view or modify these records.

2. Sharing Rules

Sharing rules were created to grant controlled access where needed, ensuring that:

- Dealers can only access orders assigned to them.
- Internal users can view records required for processing without exposing unauthorized customer details.

These rules help maintain confidentiality while still allowing workflow collaboration across teams.

Phase 4 Summary:

Phase 4 focused on validating the functionality, reliability, and security of the WhatsNext Vision Motors CRM system. All automation components—including Apex triggers, Batch Apex processes, and Record-Triggered Flows—were thoroughly tested to ensure accurate stock handling, correct dealer assignment, and timely reminder notifications. User Acceptance Testing further confirmed that customers, dealers, and administrators can seamlessly perform real-world operations such as placing orders, scheduling test drives, and managing assignments. Security settings were also finalized with strict OWD configurations and controlled sharing rules to protect sensitive customer and order data. Overall, this phase ensured that the system is stable, secure, and ready for deployment.

Phase 5: Documentation & Maintenance:

Maintenance, Monitoring & Troubleshooting

Maintenance ensures that vehicle inventory, order processing, dealer assignments, and customer records remain accurate and up to date.

Ongoing monitoring includes:

- Reviewing Lightning App performance and page-load time.
- Ensuring profiles, permission sets and sharing rules remain aligned with organizational roles (sales team, service team, dealer admin).
- Monitoring flows for:
  - Order assignment issues
  - Test drive reminder email failures
  - Dealer auto-assignment updates
- Reviewing trigger execution logs and batch processing for:
  - Stock decrement errors
  - Order confirmation logic
- Updating picklists, price lists, and vehicle inventory regularly.
- Tracking user and customer feedback to improve UI design and automation reliability.

Although developed inside a Salesforce Developer Org for learning and demonstration, these maintenance activities are crucial for real-time enterprise deployment to ensure application stability, scalability, and reliability.

Project Documentation

Documentation for the WhatNext Vision Motors Salesforce CRM serves as a comprehensive record of the system's purpose, design, and development.

It ensures that business requirements — such as vehicle inventory, dealer assignments, ordering, test drives, and service management — are clearly mapped to system functionalities.

Documentation Benefits

- Acts as a blueprint for developers, admins, and testers.
- Supports new user onboarding and training.
- Helps with troubleshooting, audits and continuous improvement.
- Enables reusability and scalability for future enhancements.

Guidelines for Documentation Submission

- Submit PDF or Word in professional format.
- Use clear headings and bullet points.

- Maintain Times New Roman (12 or 13 size) for readability.
- Errors, plagiarism and misalignment must be avoided.

Mandatory Sections to Include

Project Overview

The WhatNext Vision Motors CRM is a Salesforce-based application that manages the vehicle business lifecycle including inventory management, dealer allocation, customer management, order processing, test drives and service booking.

Customers and internal team members can:

- View available vehicles
- Place orders based on stock availability
- Schedule test drives
- Raise vehicle service requests

The CRM improves customer satisfaction through automation, reduced manual effort, and accurate dealer assignment based on customer location.

Objectives

- Automate customer order handling and dealer assignment
- Prevent orders for out-of-stock vehicles
- Automate test drive reminders through scheduled email notifications
- Enhance customer satisfaction and transparency across the purchase workflow
- Improve business decision-making through real-time vehicle availability and reporting

Phase 1: Requirement Analysis & Planning

- Identify the need to manage vehicle stock, customer details, dealer distribution and order processing inside Salesforce
- Define features:
    - Vehicle inventory
    - Dealer master
    - Vehicle orders
    - Test drives
    - Service requests
- Design data model and security model including custom objects, relationships, page layouts and access levels

Phase 2: Salesforce Development – Backend & Configurations

- Developer Org configuration and Lightning App setup

- Create custom objects and all required fields
- Configure validation rules to prevent invalid/duplicate data
- Build automation using Flows, Triggers, Batch and Scheduled Apex for:
    - Stock validation
    - Stock decrement on order confirmation
    - Daily batch job to convert Pending orders to Confirmed when stock is replenished

Phase 3: UI/UX Development & Customization

- Lightning App WhatNext Vision Motors
- Page layouts for:
    - Vehicles
    - Dealer
    - Customer
    - Orders
    - Test Drives
    - Service Requests
- Navigation Items based on business modules
- Profiles and permission setups based on internal roles

Phase 4: Testing & Security

- Unit Testing, UAT testing for automation and backend logic
- Security testing for object-level and field-level access

Phase 5: Documentation & Maintenance

- Regular monitoring of flows, triggers and scheduled jobs ensures application reliability

Conclusion:

The WhatsNext Vision Motors CRM project showcases how Salesforce can efficiently automate the complete automotive ordering lifecycle, including vehicle stock validation, dealer allocation, order processing, test-drive notifications, and service management. Through the combined use of Flows, Apex Triggers, Batch Apex, and Scheduled Apex, the system eliminates manual tasks, enhances data accuracy, reduces processing delays, and significantly improves the customer experience.

Although developed within a Salesforce Developer Org for demonstration purposes, the overall design, logic, and automation framework are scalable and can be seamlessly adapted for real-world automotive businesses. This project proves the capability of Salesforce to deliver a reliable, intelligent, and production-ready solution for modern vehicle management operations.