Salesforce Project Implementation Report: Vehicle Order Automation

Project Title: Vehicle Order Automation: Shaping the Future of Mobility with Salesforce Automation

Problem Statement

For this project, my objective is to implement the " WhatNext Vision Motors " system to automate critical vehicle order processing within Salesforce. The core problem I am addressing is the need for a more efficient and intelligent system for managing vehicle orders. Specifically, I need to automate the assignment of orders to the nearest dealers based on customer location, implement logic to prevent orders for out-of-stock vehicles, and ensure timely email reminders are sent for test drives one day before the scheduled date. This project aims to overcome manual inefficiencies and enhance customer experience through robust Salesforce automation, leveraging both Flows and Apex.

Phase 1: Problem Understanding & Industry Analysis

1.1. Executive Summary

The " Vehicle Order Automation " project involved the implementation of a custom Salesforce application designed to automate key aspects of vehicle order processing and customer engagement for a motor vehicle company. This project aimed to enhance efficiency and customer satisfaction by leveraging Salesforce's declarative and programmatic capabilities. This document details the step-by-step process I followed to build, configure, and automate the solution.

1.2. Requirement Gathering

The project requirements were provided as clear instructions within a Salesforce Virtual Internship (SFVIP2025) workspace. The moto of this project was to automate vehicle order processing using Salesforce, with specific goals outlined.

1.3. Business Process Mapping

Based on the provided requirements, I mapped the following core business processes that needed automation:

Vehicle Order Processing: Assigning orders to the nearest dealers and preventing orders for out-of-stock vehicles.

Test Drive Scheduling: Sending automated email reminders for test drives.

Vehicle Service Request Management: Though not explicitly detailed in automation, the object was created for future service needs.

1.4. Key Objectives
My main goals for this project were:

To assign vehicle orders to the nearest dealers based on the customer's location.

To prevent the creation of orders for vehicles that are out of stock.

To send email reminders for test drives one day before the scheduled date.

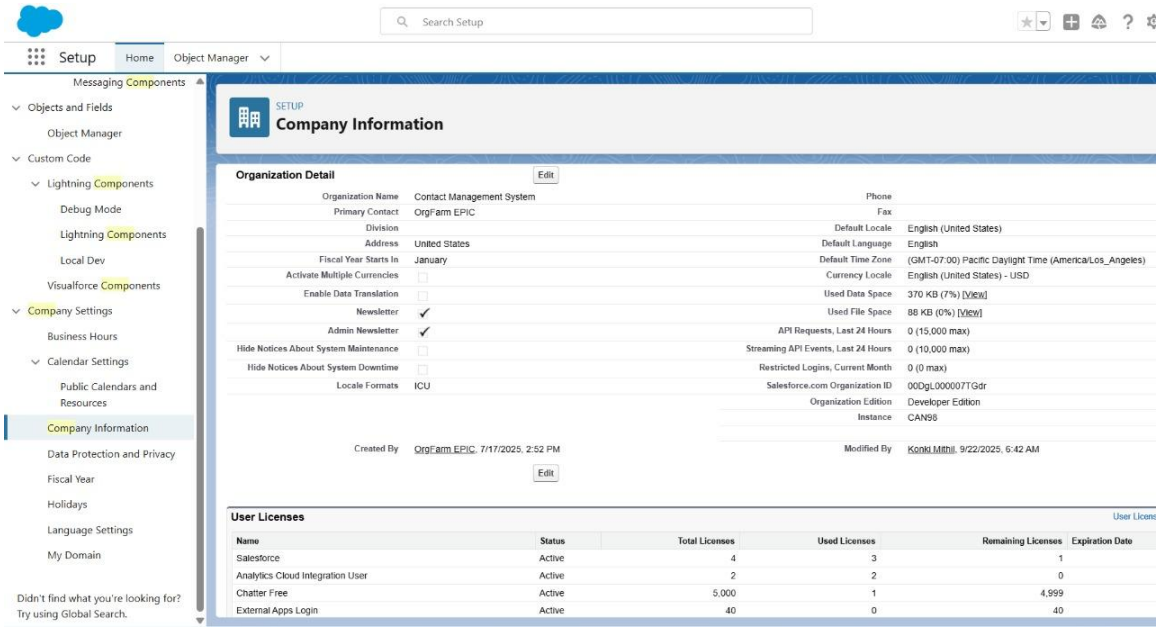
To achieve these objectives using a combination of Salesforce Flows and Apex.

Phase 2: Org Setup & Configuration

This phase involved setting up the foundational Salesforce environment. The focus was on creating a dedicated space for development and establishing the basic security and access settings required for the application.

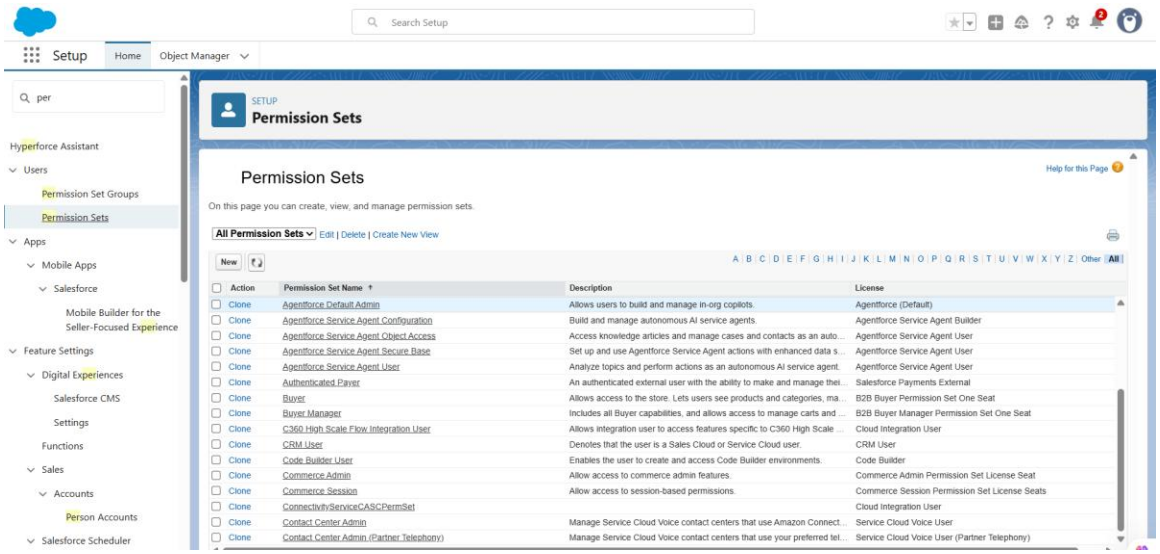
- **Salesforce Editions & Dev Org Setup** The project was built using a

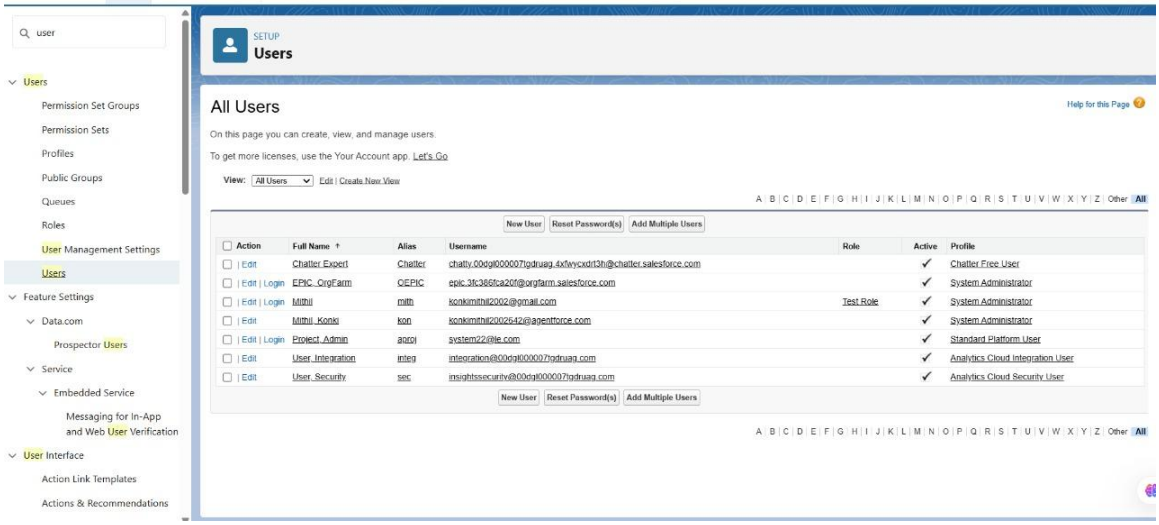
  Salesforce Developer Edition org. This edition was chosen because it is a free, fully-featured environment perfect for building and testing applications without impacting a live production org. A new developer account was signed up and a new org was provisioned specifically for this project.



- **Company Profile Setup** Upon the creation of the Developer Org, the basic company profile was inherently established. This includes default settings for locale, language, and time zone which are foundational for the org's operation.
- **User Setup & Licenses** The entire project was built and configured using the single, default **System Administrator** user that comes with a new Developer Edition org. This user has the standard Salesforce license, which grants full permissions to build all the necessary components for the project. No other users were created as the scope focused on the application's functionality.
- **Profiles** Profiles were a key part of the configuration for controlling application access. During the setup of the

  "WhatNext Vision Motors" Lightning App, it was explicitly assigned and made visible only to the **System Administrator** profile. This ensures that only users with this profile can see and use the application.

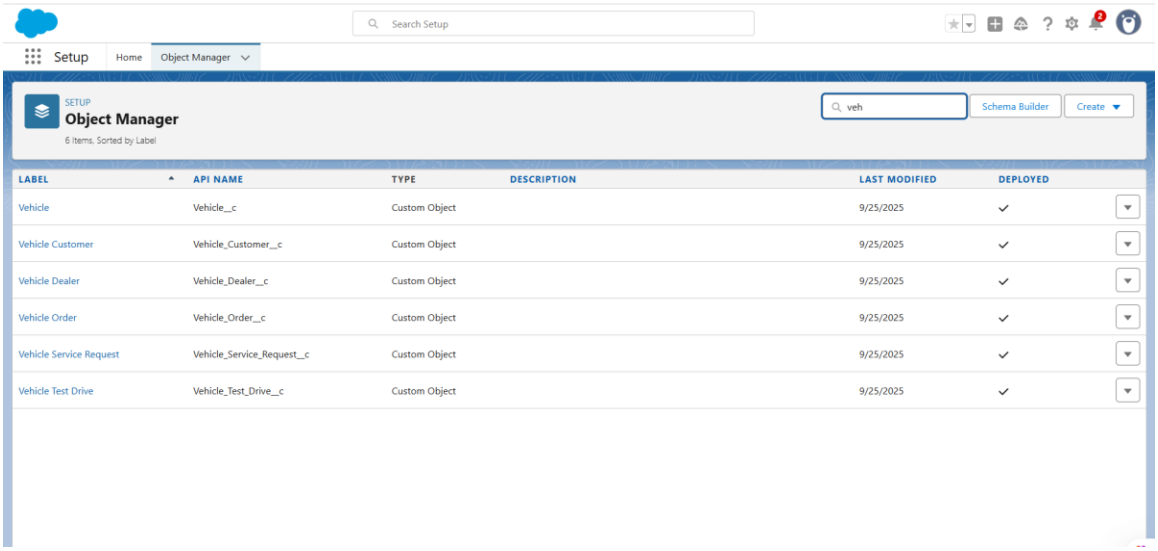




Phase 3: Data Modeling & Relationships

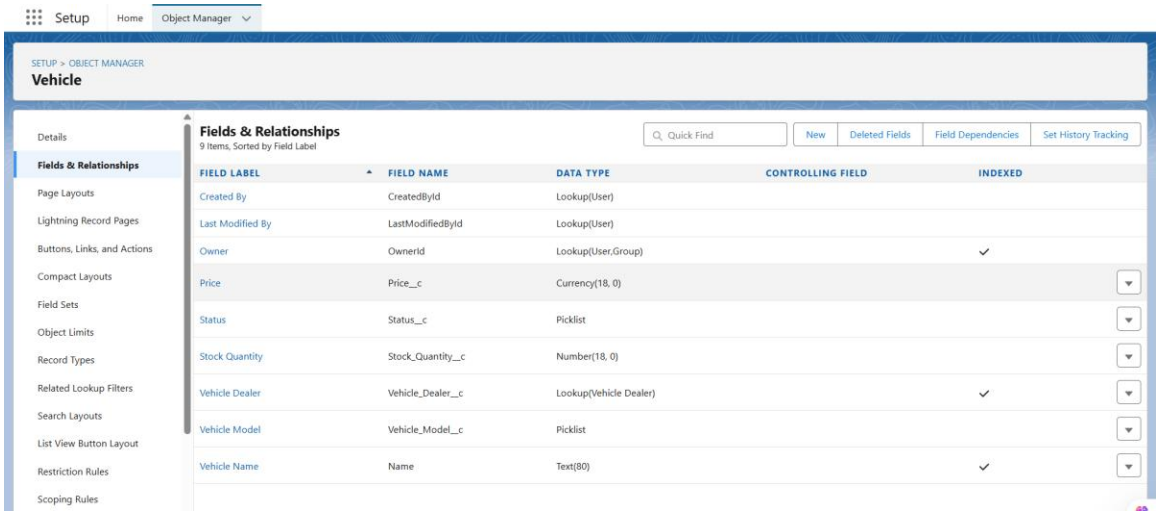
This phase was the cornerstone of the project, focused on building a logical and scalable data structure. A custom data model was created to store and connect all information about vehicles, dealers, customers, and their transactions within Salesforce.

- **Standard & Custom Objects** The application's foundation is built entirely on **custom objects**. Standard objects like Accounts or Contacts were not used to ensure the solution was tailored specifically to the automotive business process. Six custom objects were created:
    - **Vehicle__c:** To store details of each car, including model, price, and inventory.
    - **Vehicle_Dealer__c:** To manage dealership information and locations.
    - **Vehicle_Customer__c:** To maintain a database of prospective and existing customers.
    - **Vehicle_Order__c:** A transactional object to track sales orders.
    - **Vehicle_Test_Drive__c:** To schedule and manage customer test drives.
    - **Vehicle_Service_Request__c:** To handle after-sales service requests.
- **Fields** Numerous custom fields were created on each object to capture essential business data. Various data types were used to ensure data integrity, including:
    - **Picklist:** For predefined values like the **Status__c** field on the **Vehicle Order** object (e.g., Pending, Confirmed, Delivered).
    - **Number & Currency:** For quantitative data like **Stock_Quantity__c** and **Price__c** on the **Vehicle** object.
    - **Auto-Number:** To create a unique, system-generated ID for records like the **Vehicle_Order_Number__c**.
    - **Lookup Relationship:** Used to link objects together (detailed below).



- **Schema Builder** While the objects and fields were configured in the Object Manager, the **Schema Builder** is the best tool for visualizing the complete data

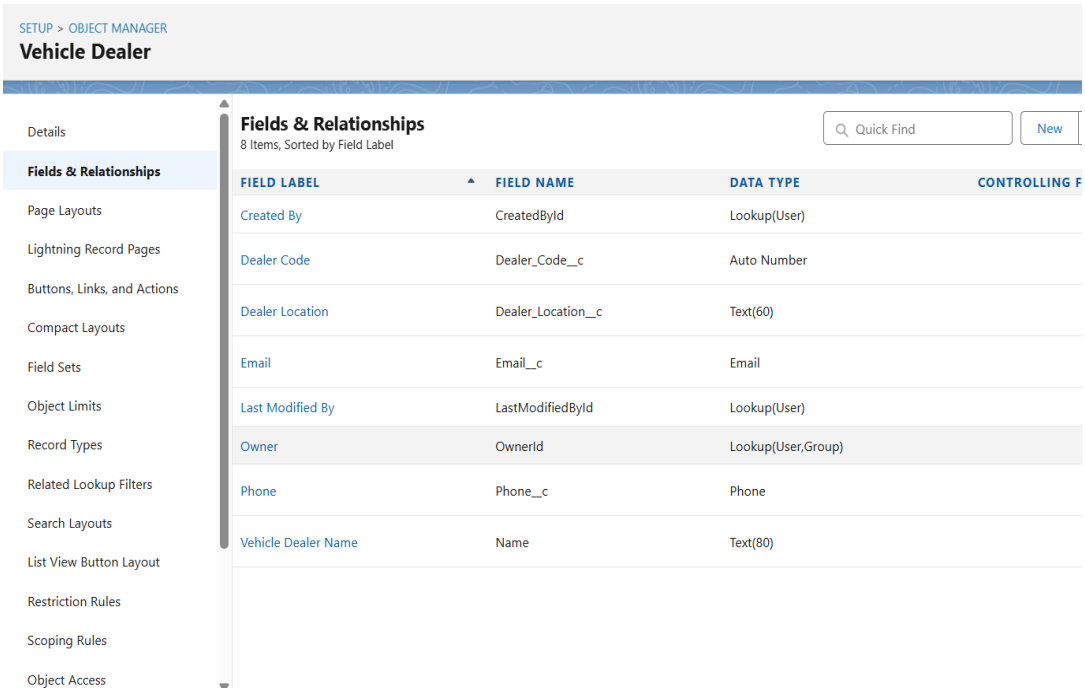
model. It provides a clear, graphical diagram showing all the custom objects and how they are connected through their relationship fields.



- **Lookup vs Master-Detail vs Hierarchical Relationships** The project exclusively used

  Lookup relationships to create connections between objects. This type of relationship creates a flexible link where the related records can exist independently.

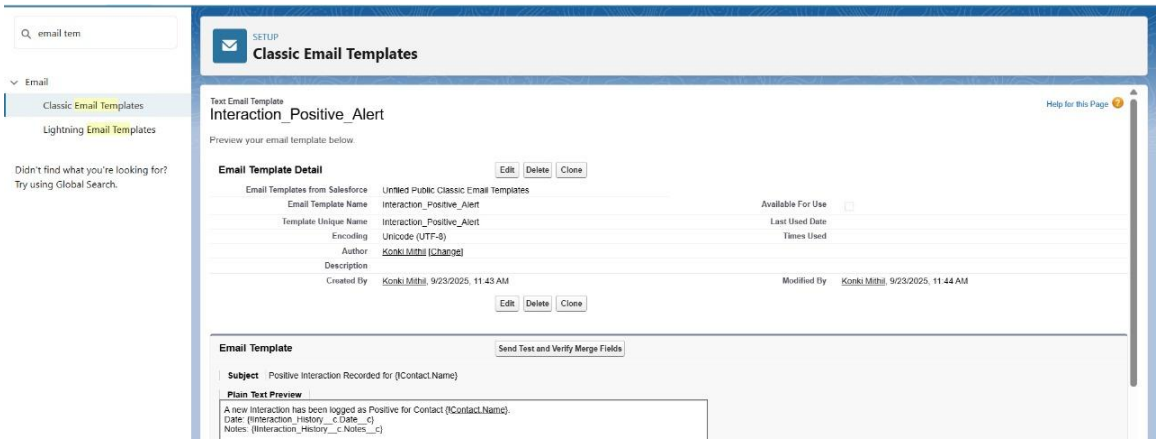
  - **Example:** A **Vehicle Order** record has a lookup field to the **Vehicle Customer** object.
  - **Reasoning:** Lookups were chosen over Master-Detail because the objects are not tightly dependent. For instance, a **Vehicle Customer** can exist in the system even if they have never placed an order. Master-Detail relationships, which create a parent-child dependency, were not suitable for this model. Hierarchical relationships were not applicable.



Phase 4: Process Automation

4.1. Email Templates & Alerts (Used in Flows)

Although explicit email templates were not shown being created in the Classic Email Templates section, the video demonstrated the creation of email bodies directly within the Flow Builder. An "Email Action" was configured within the "Test Drive Reminder" flow to send a reminder email.



4.2. Automation Engine: Flow Builder

I implemented two key flows to automate critical business processes.

4.2.1. Record-Triggered Flow: Auto-Assign Dealer
Objective: To automatically assign a vehicle order to the nearest dealer based on the customer's location.

Trigger: Fires when a Vehicle Order record is created or updated.

Entry Criteria: Status field is "Pending".

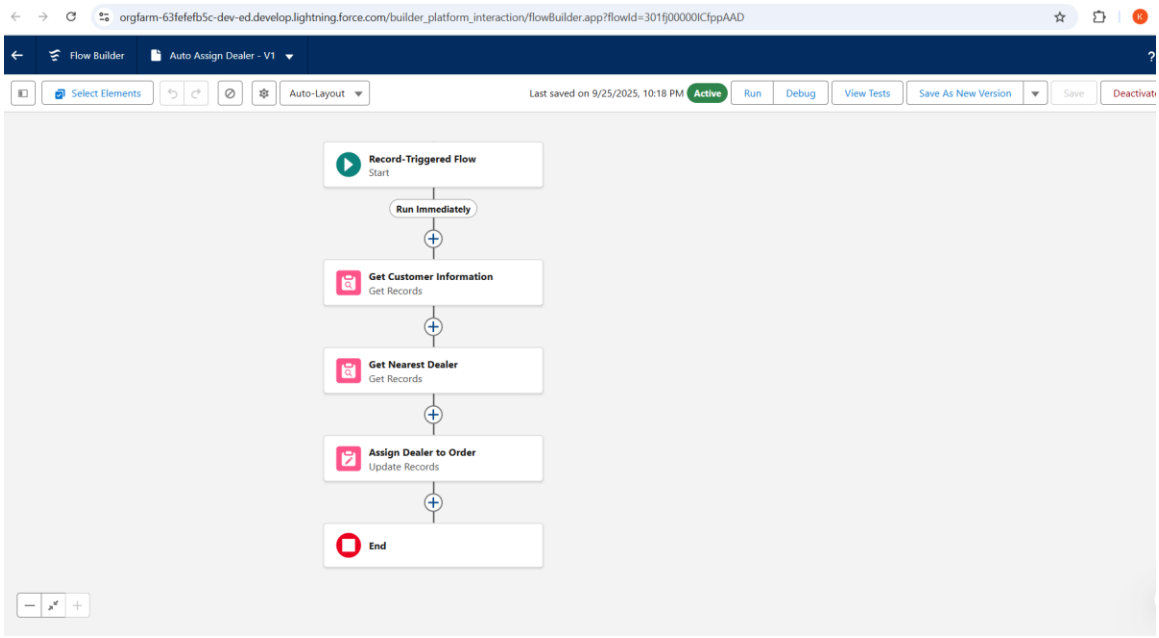
Logic:

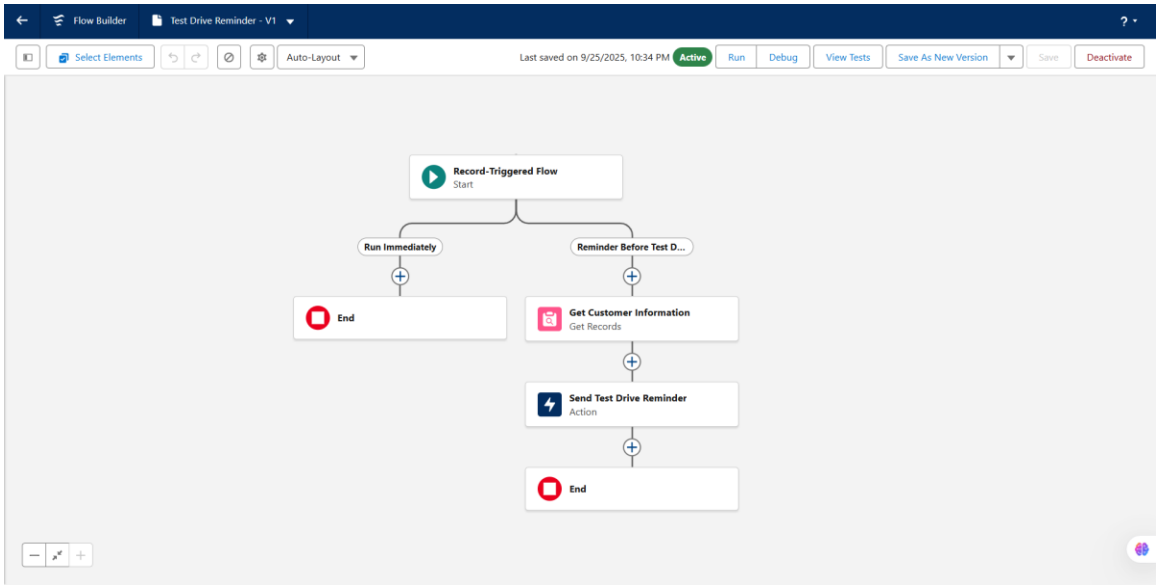
Get Customer Information: Fetches the Vehicle Customer record associated with the order to get their Address.

Get Nearest Dealer: Queries Vehicle Dealer records to find a dealer whose Location matches the customer's Address.

Update Records: Updates the Assigned Dealer lookup field on the Vehicle Order record with the ID of the found dealer.

Testing: Demonstrated by creating a Vehicle Order with a customer in "Hyderabad," and the flow correctly assigned the "EM" dealer, also located in "Hyderabad".





4.2.2. Record-Triggered Flow: Test Drive Reminder
Objective: To send an email reminder to the customer one day before their scheduled test drive.

Trigger: Fires when a Vehicle Test Drive record is created or updated.

Entry Criteria: Status field is "Scheduled".

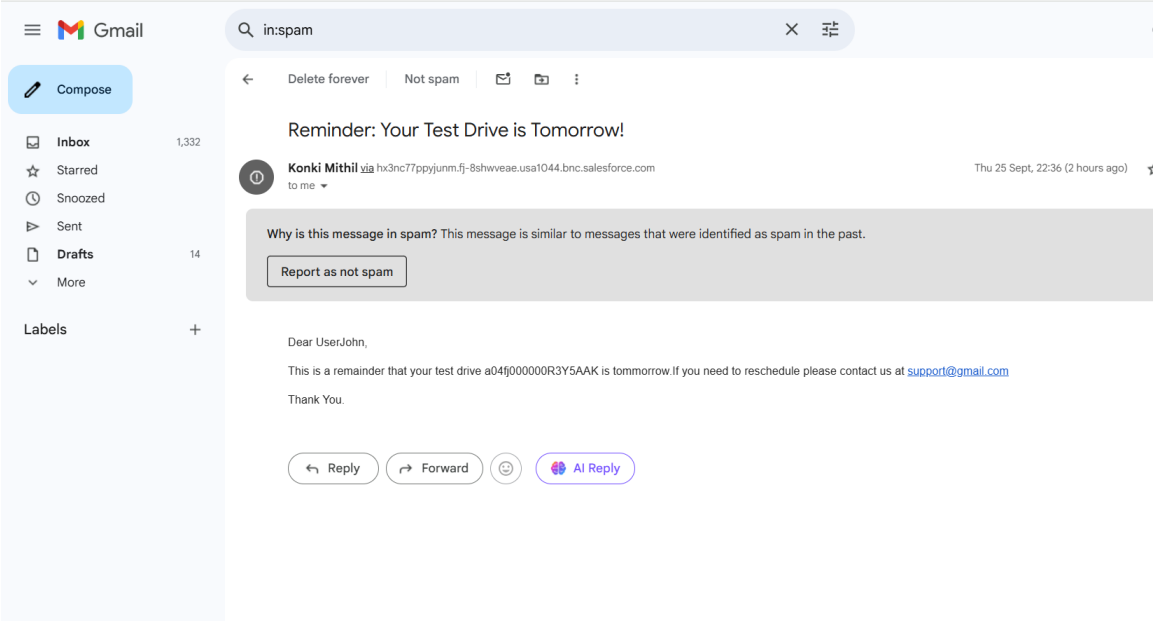
Scheduled Path: Configured to run "1 Day Before" the Test Drive Date.

Logic (within scheduled path):

Get Customer Information: Fetches the Vehicle Customer record to retrieve the customer's email and name.

Send Email: Sends an email with the subject "Reminder: Your test drive is tomorrow" and a personalized body including the customer's name and test drive ID.

Testing: Demonstrated by creating a Vehicle Test Drive for "tomorrow" with a "Scheduled" status, and the email was immediately received (due to immediate processing in debug/demo context).



Phase 5: Apex Programming (Developer)

5.1. Apex Trigger: Vehicle Order Processing

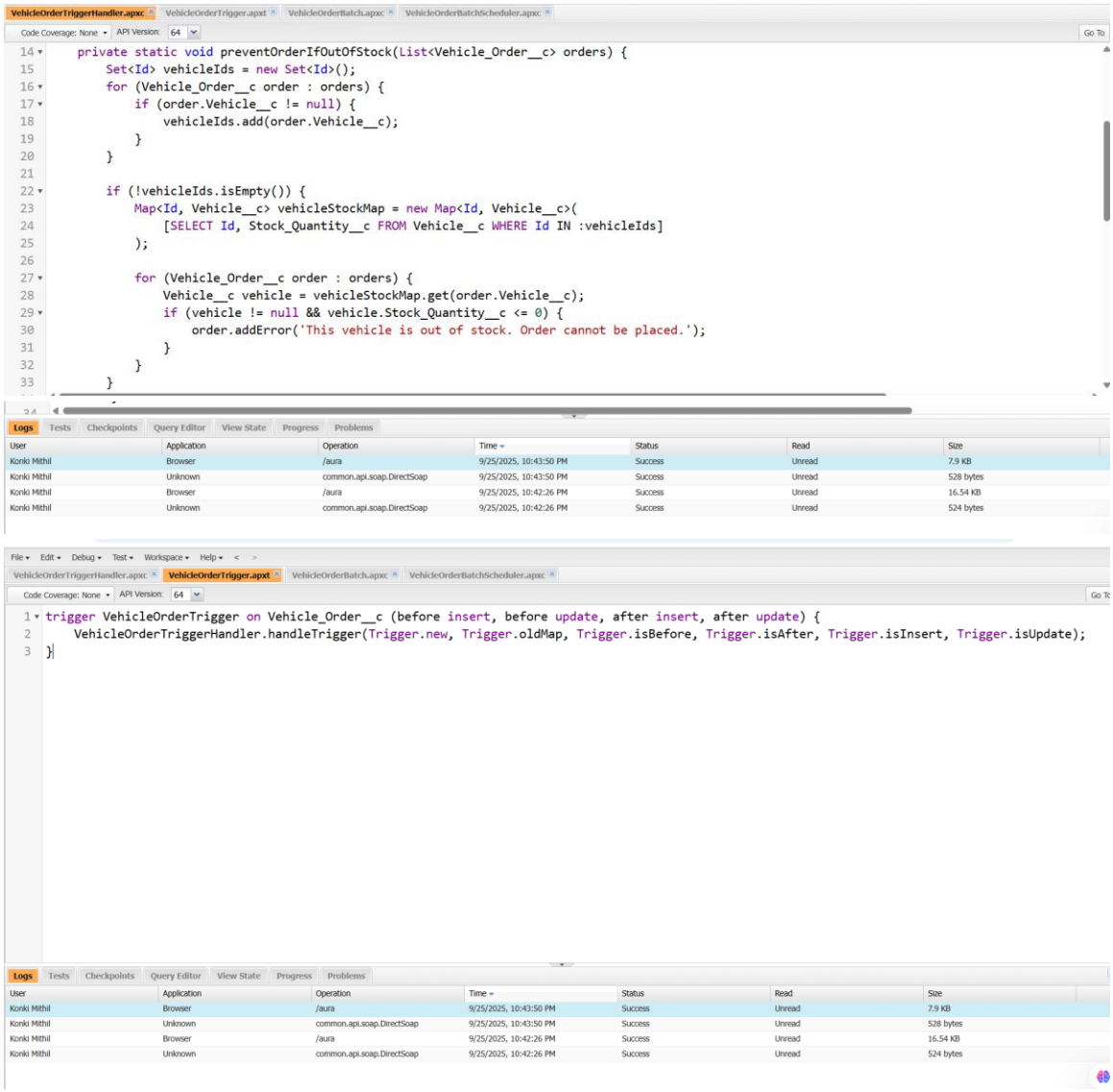
I implemented an Apex trigger and a handler class to manage complex logic on Vehicle Order records.

VehicleOrderTriggerHandler Apex Class: This class contains the actual business logic.

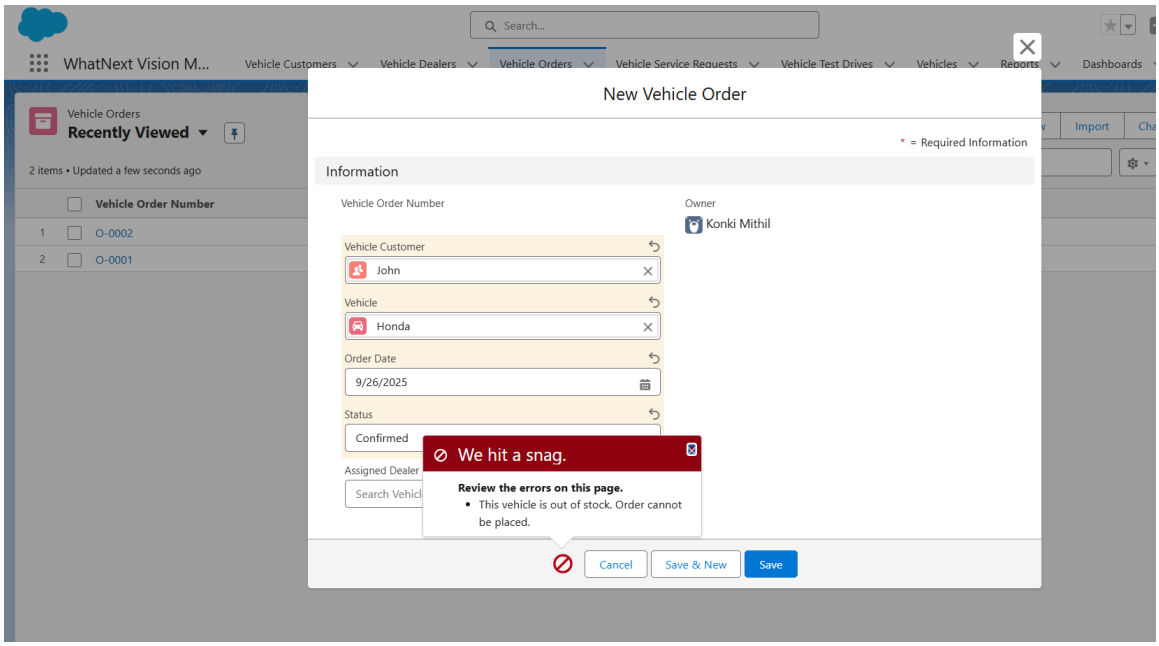
Stock Deduction: When a Vehicle Order is updated and its Status changes to "Confirmed," this logic reduces the Stock Quantity of the associated Vehicle record by 1.

Out-of-Stock Prevention: Before an order is created or updated, this logic checks if the Stock Quantity of the selected Vehicle is 0. If it is, it prevents the order from being saved and displays an "out of stock" error message.

VehicleOrderTrigger Apex Trigger: This trigger invokes the handler class on before insert, before update, after insert, and after update events on the Vehicle Order object.


VehicleOrderTriggerHandler.apxc
VehicleOrderTrigger.apxt
VehicleOrderBatch.apxc
VehicleOrderBatchScheduler.apxc
Code Coverage: None
API Version: 64
Go To
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) {
order.addError('This vehicle is out of stock. Order cannot be placed.');
}
}
}
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
User
Application
Operation
Time
Status
Read
Size
Konki Mithil
Browser
/aura
9/25/2025, 10:43:50 PM
Success
Unread
7.9 KB
Konki Mithil
Unknown
common.api.soap.DirectSoap
9/25/2025, 10:43:50 PM
Success
Unread
528 bytes
Konki Mithil
Browser
/aura
9/25/2025, 10:42:26 PM
Success
Unread
16.54 KB
Konki Mithil
Unknown
common.api.soap.DirectSoap
9/25/2025, 10:42:26 PM
Success
Unread
524 bytes

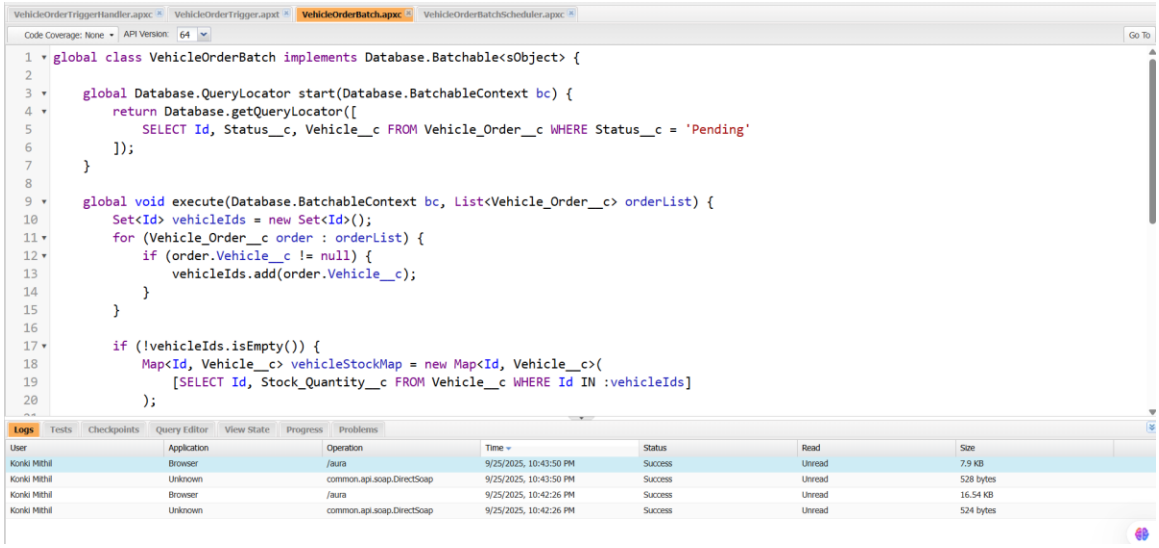
File
Edit
Debug
Test
Workspace
Help
VehicleOrderTriggerHandler.apxc
VehicleOrderTrigger.apxt
VehicleOrderBatch.apxc
VehicleOrderBatchScheduler.apxc
Code Coverage: None
API Version: 64
trigger VehicleOrderTrigger on Vehicle_Order__c (before insert, before update, after insert, after update) {
VehicleOrderTriggerHandler.handleTrigger(Trigger.new, Trigger.oldMap, Trigger.isBefore, Trigger.isAfter, Trigger.isInsert, Trigger.isUpdate);
}
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
User
Application
Operation
Time
Status
Read
Size
Konki Mithil
Browser
/aura
9/25/2025, 10:43:50 PM
Success
Unread
7.9 KB
Konki Mithil
Unknown
common.api.soap.DirectSoap
9/25/2025, 10:43:50 PM
Success
Unread
528 bytes
Konki Mithil
Browser
/aura
9/25/2025, 10:42:26 PM
Success
Unread
16.54 KB
Konki Mithil
Unknown
common.api.soap.DirectSoap
9/25/2025, 10:42:26 PM
Success
Unread
524 bytes



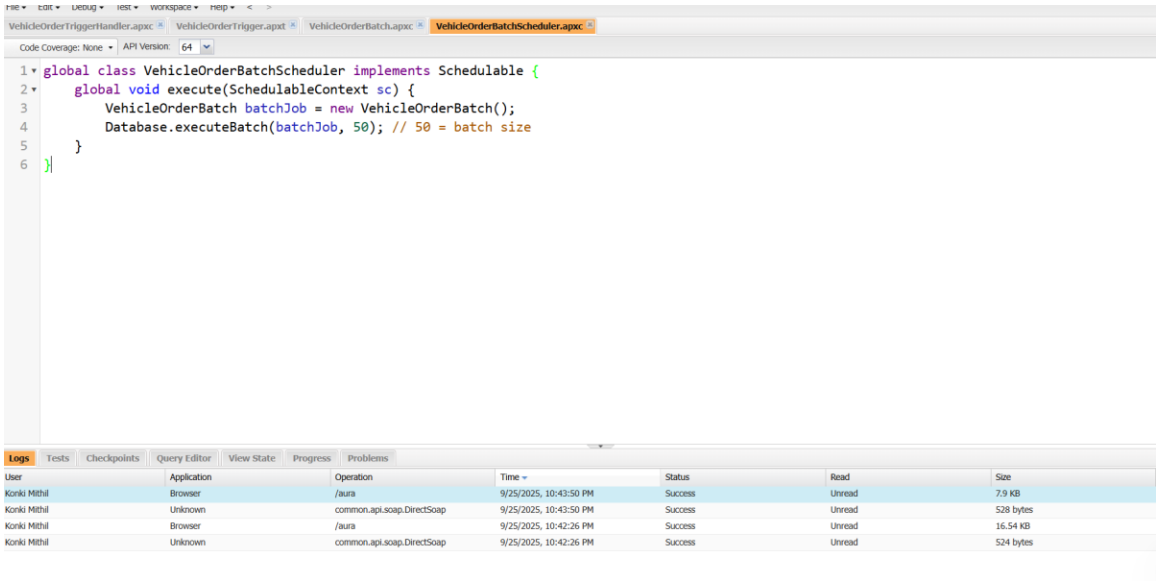
5.2. Asynchronous Apex: Vehicle Order Batch Class & Scheduler
I created a Batch Apex class and a Scheduler class to handle asynchronous processing of vehicle orders.

VehicleOrderBatch Apex Class: This class is designed to process Vehicle Order records in batches. The video describes its purpose as checking vehicle stock quantity and, if it's less than zero, setting the order status to "pending".

VehicleOrderBatchScheduler Apex Class: This class schedules the VehicleOrderBatch to run at specific intervals. The video demonstrated its creation and successful scheduling.



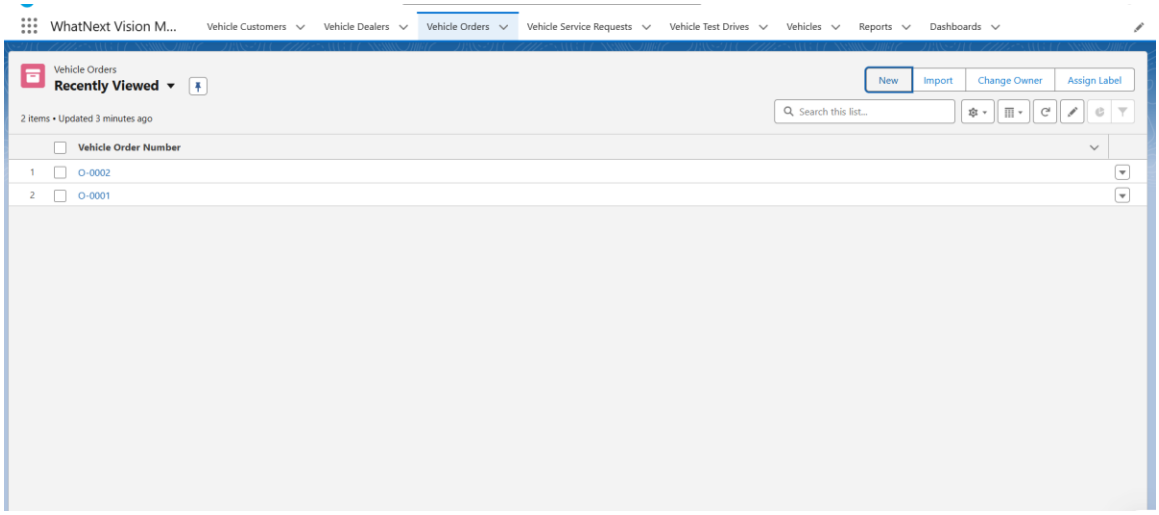
```
global class VehicleOrderBatch implements Database.Batchable<sObject> {

    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]
            );
```



Phase 6: User Interface Development

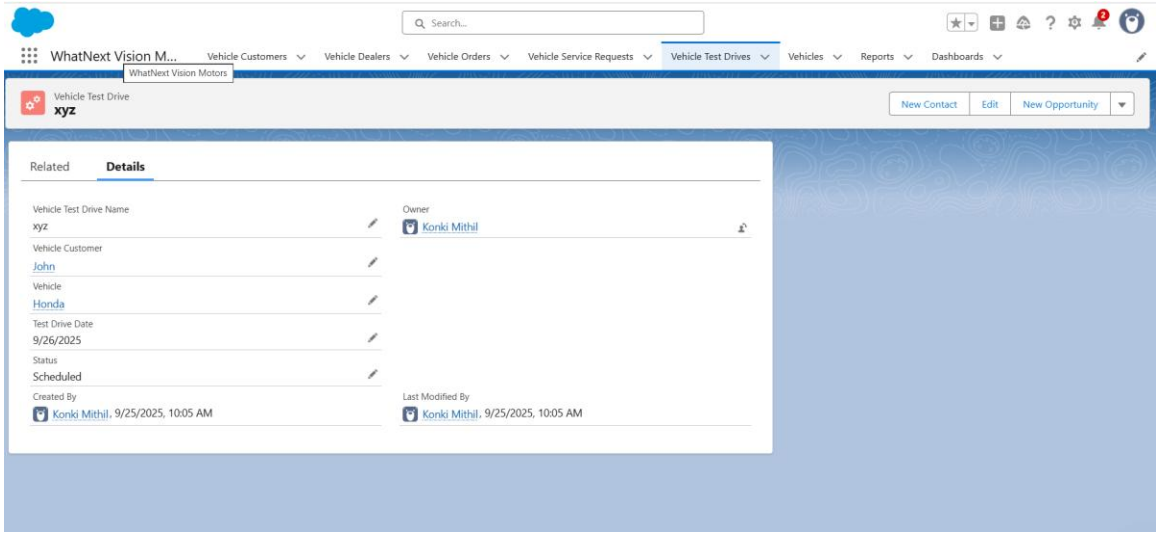
6.1. Application Creation: Lightning App Builder

I used the Lightning App Builder to create a custom application named "WhatNext Vision Motors". This app serves as the central hub for users to interact with all the custom objects and functionalities.



6.2. Navigation: Custom Object Tabs

I created custom tabs for all six custom objects (Vehicle, Vehicle Dealer, Vehicle Customer, Vehicle Order, Vehicle Test Drive, Vehicle Service Request). I then added these custom tabs, along with standard "Reports" and "Dashboards" tabs, to the "WhatNext Vision Motors" app's navigation bar. This ensures easy and intuitive navigation for users.

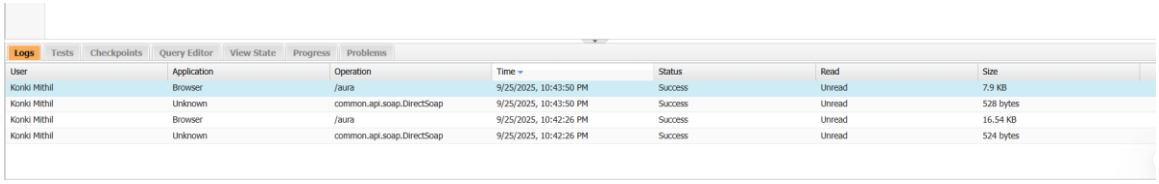


Phase 7: Integration & External Access (Out of Scope)

This project focused on building robust internal functionalities within the Salesforce platform. Integration with external systems was not part of the initial project scope.

Phase 8: Data Management & Deployment

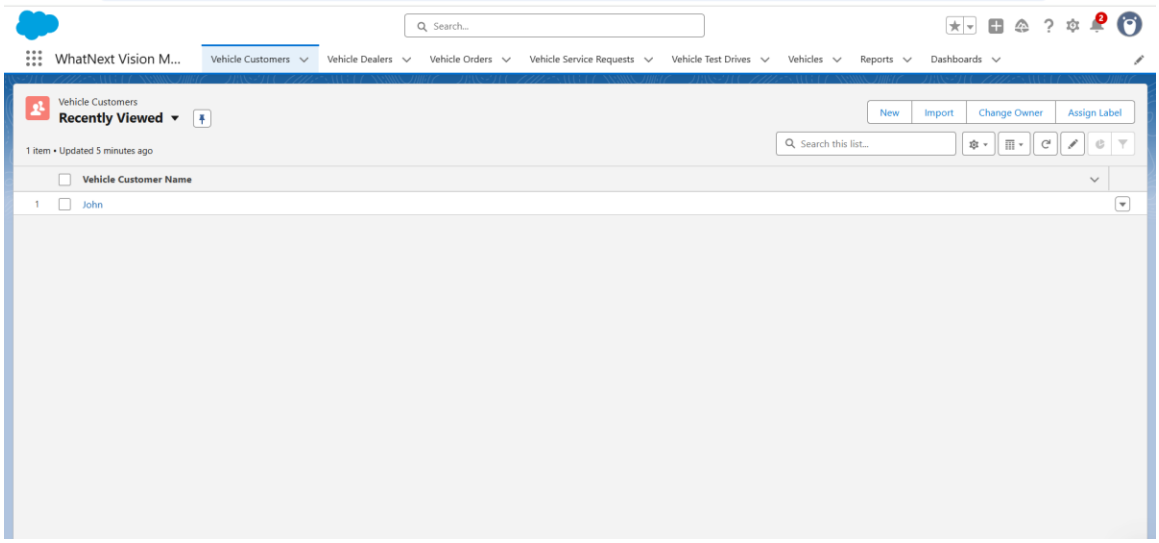
All components for the " Vehicle Order Automation " project were developed and tested directly within a single Salesforce Developer Edition org. Data entry was performed manually for demonstration purposes. Formal data migration (e.g., using Data Loader) or deployment strategies (e.g., Change Sets, SFDX) were not covered in this phase of the project.



Phase 9: Reporting, Dashboards & Security Review

9.1. Security Model Summary

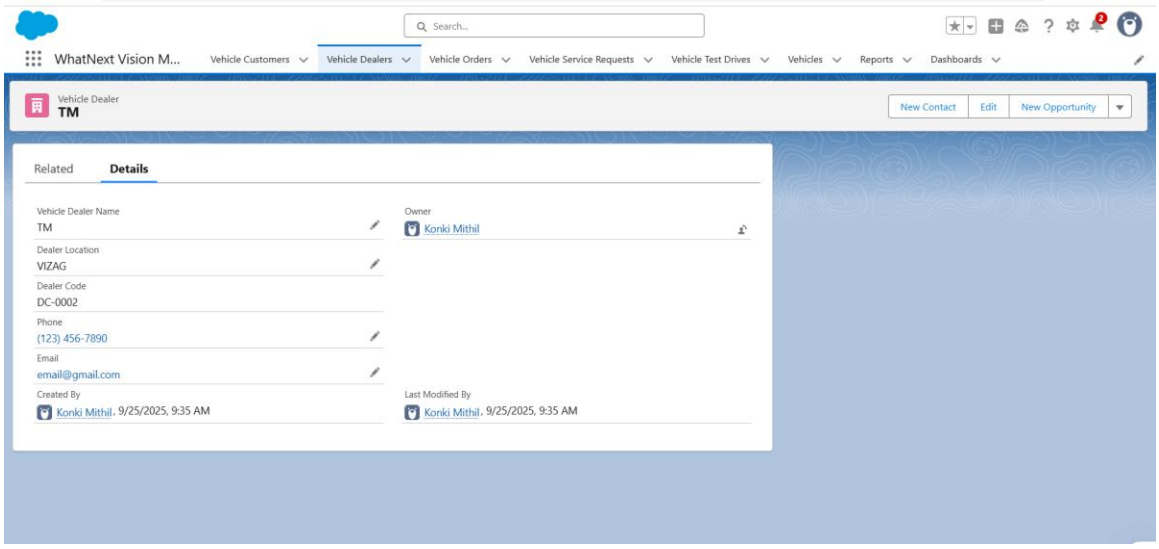
While the project was built within a Salesforce org that inherently has a security model, the



video did not delve into specific security configurations like Object-Level Security, Field-Level Security, or Organization-Wide Defaults. The demonstration was conducted under a System Administrator profile.

9.2. Reporting & Dashboard Framework

I included the standard "Reports" and "Dashboards" tabs in the "WhatNext Vision Motors" application. This provides users with direct access to Salesforce's analytical capabilities, allowing for the creation of custom reports and dashboards on all the collected vehicle, customer, order, and test drive data in the future.



Phase 10: Final Presentation & Demo Day

10.1. Project Demonstration & Key Outcomes

I successfully demonstrated the complete "WhatNext Vision Motors" project by showcasing its core functionalities:

Custom Objects & Data Entry: Demonstrated the creation of Vehicle Customer, Vehicle Dealer, and Vehicle records.

Auto-Assign Dealer Flow: Showed how a Vehicle Order automatically gets assigned to the correct dealer based on customer location.
Test Drive Reminder Flow: Demonstrated the automated email reminder being sent for a scheduled test drive.

Apex Order Processing:
Confirmed an order, showing the Stock Quantity on the Vehicle record being automatically reduced.

Attempted to create an order for an out-of-stock vehicle, demonstrating the Apex trigger's ability to prevent the order and display an error message.Batch Apex: Showcased the successful execution of the VehicleOrderBatchScheduler.

10.Conclusion

The " Vehicle Order Automation " project successfully implemented a robust and automated system for vehicle order processing and customer engagement on the Salesforce platform. By integrating custom objects, Flows, and Apex code, I addressed key business challenges related to dealer assignment, inventory management, and test drive reminders. This solution provides a strong foundation for WhatNext Vision Motors to streamline its operations, improve data accuracy, and enhance the overall customer experience in the competitive mobility sector.