Salesforce CRM Project Documentation

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

Project Overview

WhatNext Vision Motors, an emerging leader in the automotive sector, aimed to modernize its customer interaction and operational processes through the implementation of a customized Salesforce CRM solution. The primary focus of the project was to streamline vehicle order management, ensure accurate dealer assignment, and enhance customer engagement via automation.

The previous manual processes often led to delays, stock mismanagement, and customer dissatisfaction. To overcome these challenges, the new CRM system was designed with features such as real-time stock validation, automatic dealer assignment based on customer location, test drive reminders via email, and backend automation through Apex triggers and batch classes.

The CRM platform also provides a user-friendly interface using Lightning Apps and Dynamic Forms, ensuring an efficient experience for internal users. Overall, the solution improves efficiency, reduces errors, and lays a scalable foundation for future enhancements like AI-based vehicle recommendations or chatbot support.

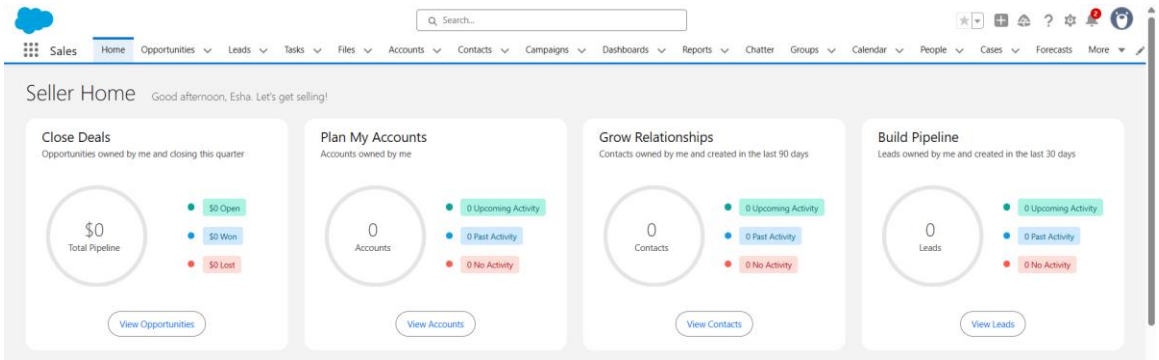
Objectives

The key objectives of this Salesforce CRM implementation are:

1. **Automate Order and Dealer Assignment**
    - Automatically assign the nearest dealer based on the customer's city at the time of order placement.
2. **Prevent Out-of-Stock Orders**
    - Ensure customers can only place orders for vehicles currently in stock using validation rules and Apex triggers.
3. **Send Test Drive Reminders**
    - Use scheduled email flows to remind customers of their upcoming test drives, reducing missed appointments.
4. **Improve User Experience**

- Implement Lightning Apps and Dynamic Forms to provide a clean, responsive interface for managing records.

5. **Maintain a Scalable Backend**
    - Use modular Apex classes and scheduled batch jobs to automate stock updates and order confirmations in bulk.



Phase 1: Requirement Analysis & Planning

The initial phase of the project focused on understanding the business needs of WhatNext Vision Motors and translating them into system requirements within the Salesforce ecosystem. The objective was to build a CRM that supports the complete vehicle management lifecycle—starting from inventory tracking to customer orders and post-sales interactions.

Business Requirements

The following core requirements were identified:

- Centralized storage and management of vehicle, dealer, and customer data.
- Real-time validation of vehicle stock at the time of order placement.
- Automatic assignment of the nearest dealer based on customer address.
- Tracking of test drives and vehicle service requests.
- Automation of key workflows to reduce manual intervention.

Defining Project Scope

To meet the business objectives, the system was designed to include:

- Custom objects for managing vehicles, orders, dealers, customers, test drives, and service requests.

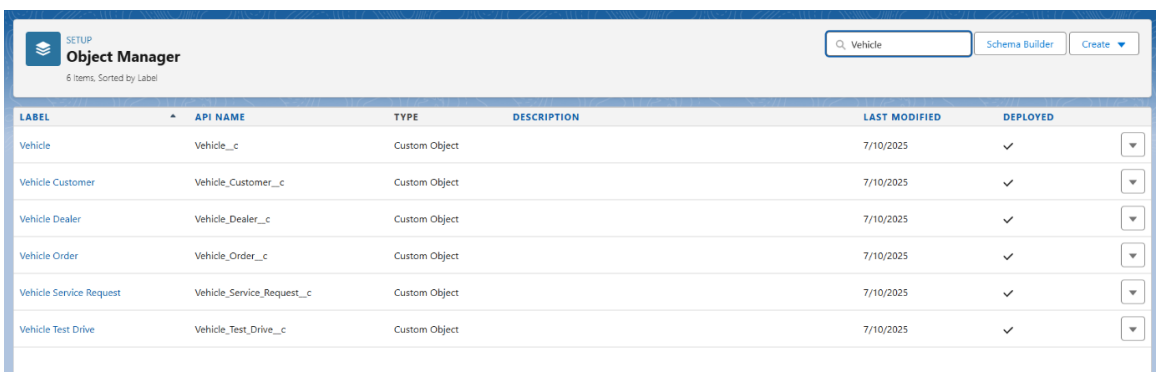
- Record-triggered flows to assign dealers and send email notifications.
- Apex triggers to validate stock availability and update inventory levels.
- Batch Apex to process pending orders based on stock replenishment.

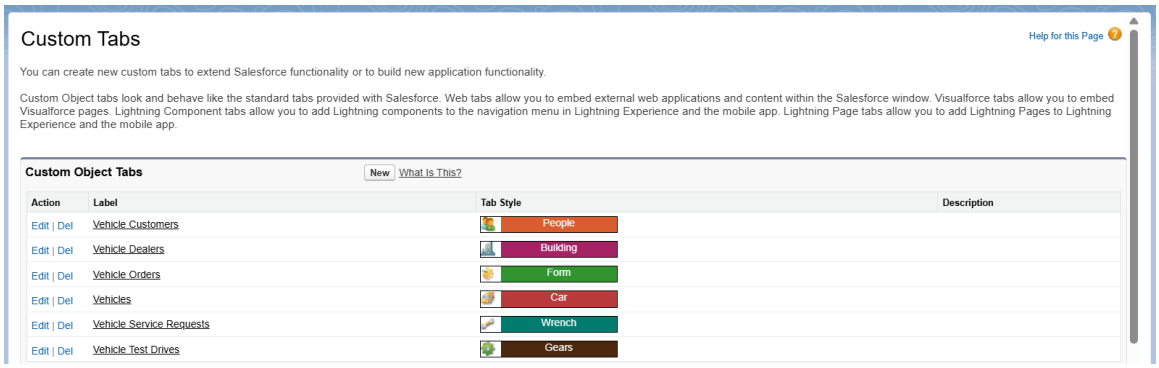
Data Model

Six custom objects were created to reflect the business structure:

Object Name	Purpose
Vehicle__c	Stores vehicle details and stock info
Vehicle_Dealer__c	Contains dealer information
Vehicle_Customer__c	Stores customer details
Vehicle_Order__c	Tracks vehicle orders
Vehicle_Test_Drive__c	Schedules and tracks test drives
Vehicle_Service_Request__c	Manages service history and issues

These objects are interlinked using lookup relationships to ensure data integrity.





Security Model

- Standard Salesforce profiles were used with additional **Permission Sets** to grant access to custom objects.
- **Field-Level Security** and **Role Hierarchy** ensured that users could only view or edit data relevant to their responsibilities.
- **Field History Tracking** was enabled on critical fields such as Stock_Quantity__c (Vehicle) and Status__c (Order) for audit purposes.

Phase 2: Salesforce Development – Backend & Configurations

- **Setup Environment & DevOps Workflow**

  To begin the development process, a **Salesforce Developer Org** was set up for building and testing all customizations and automation features.

  - **Environment:** Salesforce Lightning Experience (Developer Edition)
  - **User Profiles/Roles:** Standard profiles were used for testing. No custom profiles were created.
  - **Deployment Method:** Metadata was deployed using **Change Sets** from the sandbox to production.

- **Customization of Objects, Fields, Validation Rules, and Automation**

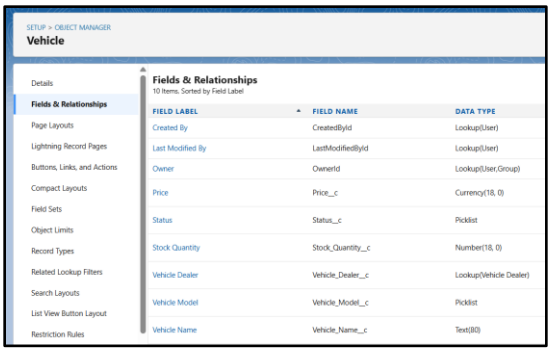
Custom Objects and Fields

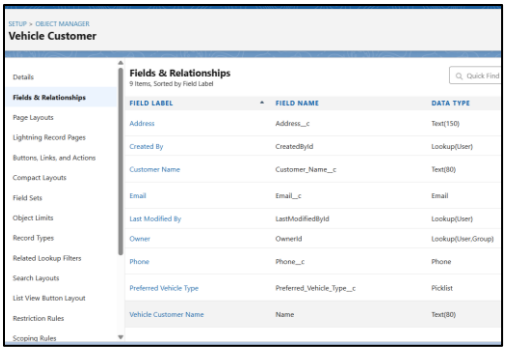
The following custom objects were created and configured to support the business flow:

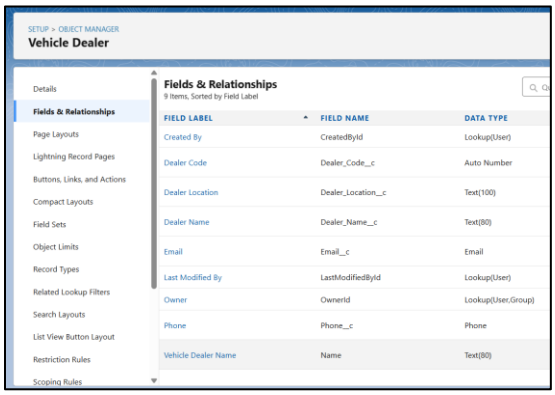
- **Vehicle** – Stores vehicle name, stock count, model, etc.
- **Dealer** – Stores dealer location and vehicle availability
- **Customer** – Stores customer details and address
- **Order** – Captures vehicle orders and order status

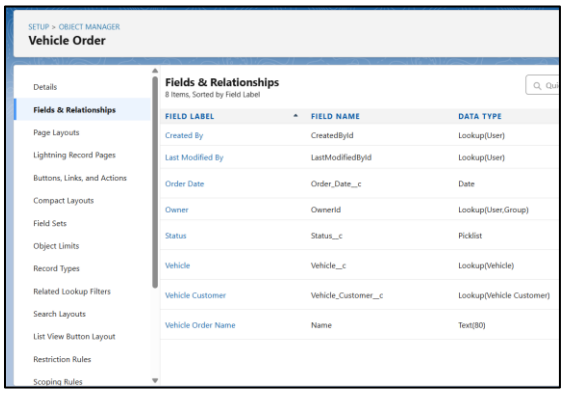
Relationships:

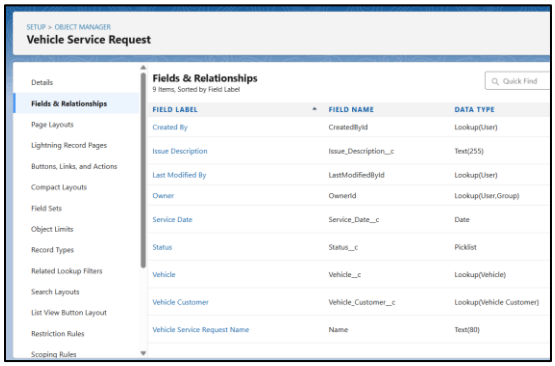
- Order → Vehicle: Lookup
- Order → Dealer: Lookup
- Order → Customer: Master-Detail or Lookup (based on implementation)

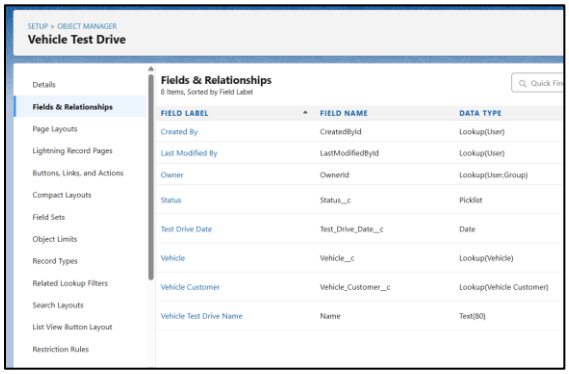












Validation Rules

- **Out-of-Stock Order Blocker:**
  Prevents the creation of an order if the selected vehicle has zero stock.

Automation: Workflow Tools

- **Flows (Record-Triggered):**

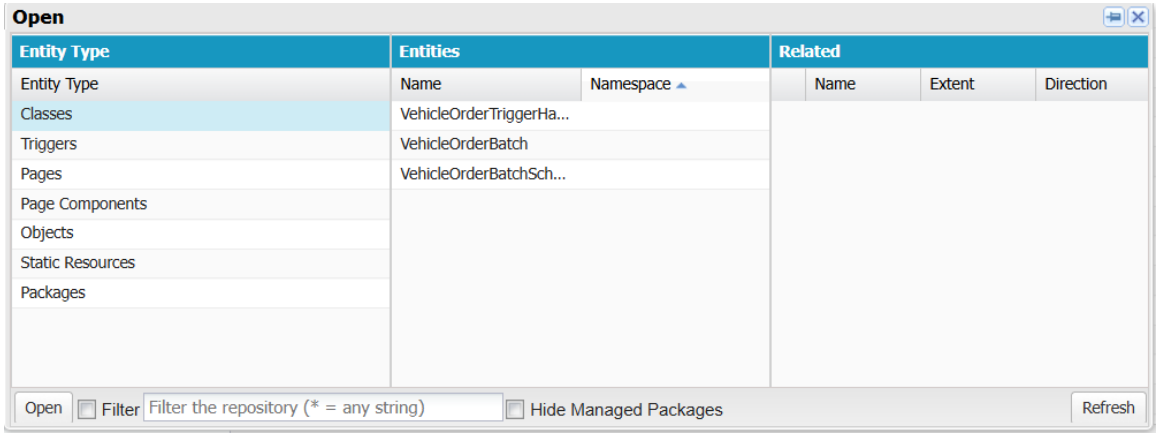
- Auto-assign the **nearest dealer** based on the customer's address using a **Record-Triggered Flow** on Order object.
- Send **test drive reminders** via **Scheduled Flows**.

Apex Classes and Triggers

Apex Classes:

Apex Classes were written to modularize the trigger logic and support backend automation:

- VehicleOrderTriggerHandler handles stock checks and updates in the trigger.
- VehicleOrderBatch checks for pending orders and confirms them if stock is available.
- VehicleOrderBatchScheduler schedules the batch job to run daily at 12 PM. All classes follow best practices using bulk-safe operations and reusable methods.

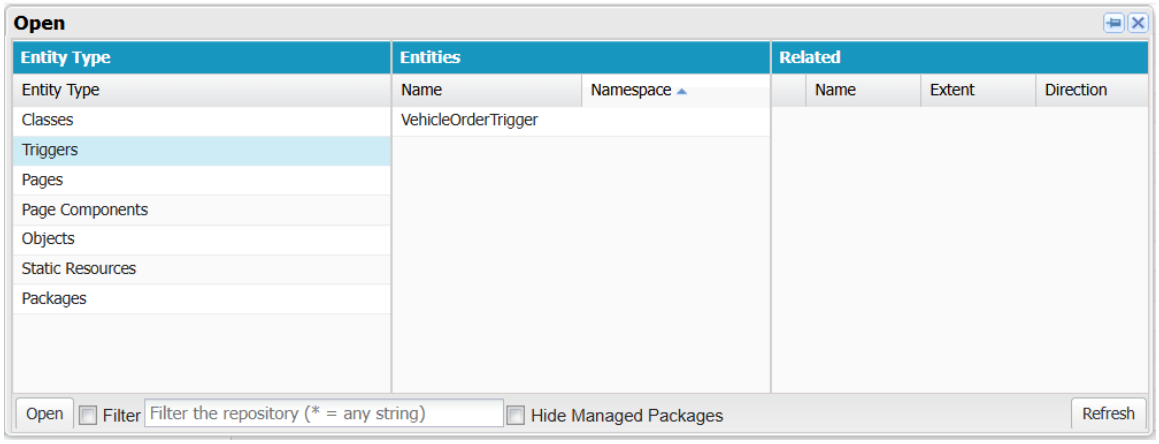


Apex Trigger:

Apex Trigger was written on the **Order** object to perform:

- Stock availability validation
- Auto-dealer assignment (if not handled by Flow)
- Order status update logic (Pending or Confirmed)

Trigger follows best practices using a **Trigger Handler pattern**.

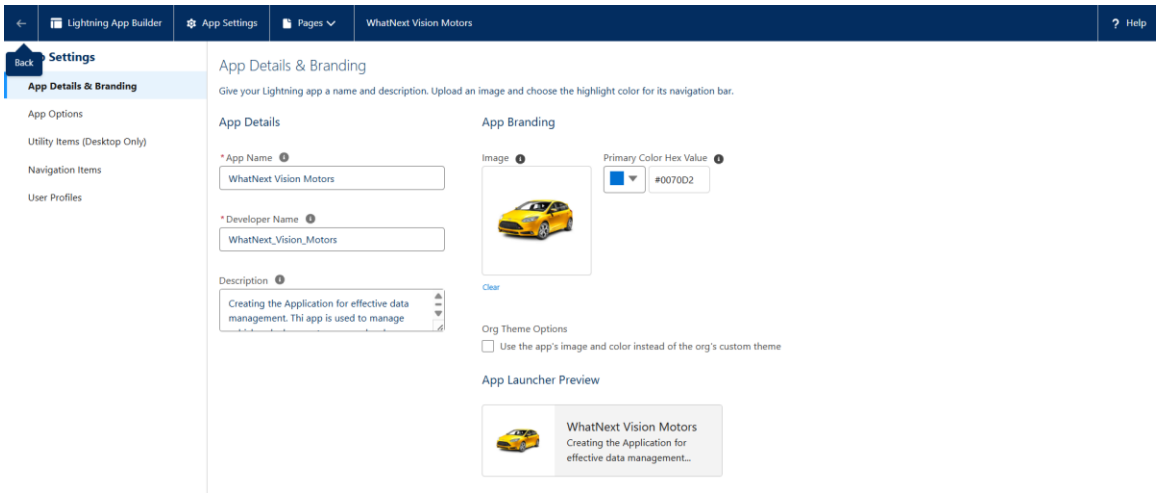


Phase 3: UI/UX Development & Customization

Lightning App Setup via App Manager

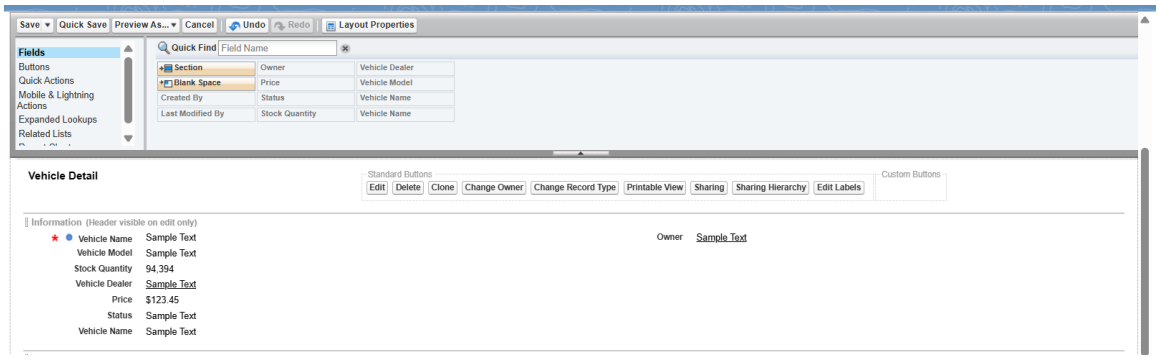
A custom Lightning App named "WhatNext Vision Motors" was created using App Manager. This app includes relevant custom tabs like Vehicles, Dealers, Orders, Customers, Test Drives, and Service Requests for easy navigation.

- Lightning App created: *WhatNext Vision Motors*
- Tabs: Vehicles, Dealers, Customers, Orders, Test Drives, Services
- Used **Dynamic Forms** for fields based on status & availability
- Highlight panels, related lists added to Lightning Pages



Page Layouts and Dynamic Forms:

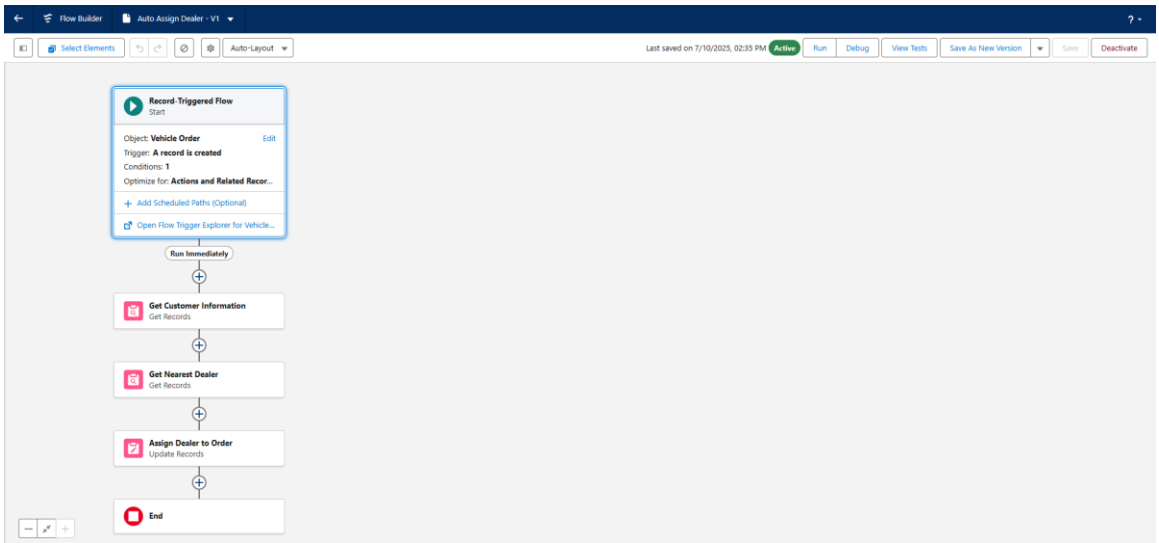
Page layouts were customized for key objects such as Vehicle__c, Vehicle_Order__c, and Vehicle_Test_Drive__c to ensure clean UI and contextual field visibility. Dynamic Forms were used to place fields directly on the Lightning Record Page and conditionally show fields based on values like order status or vehicle availability.



Flow 1: Auto Dealer Assignment

This flow runs on Vehicle_Order__c creation and:

- Fetches customer's address
- Finds a dealer in the same city
- Assigns that dealer to the order

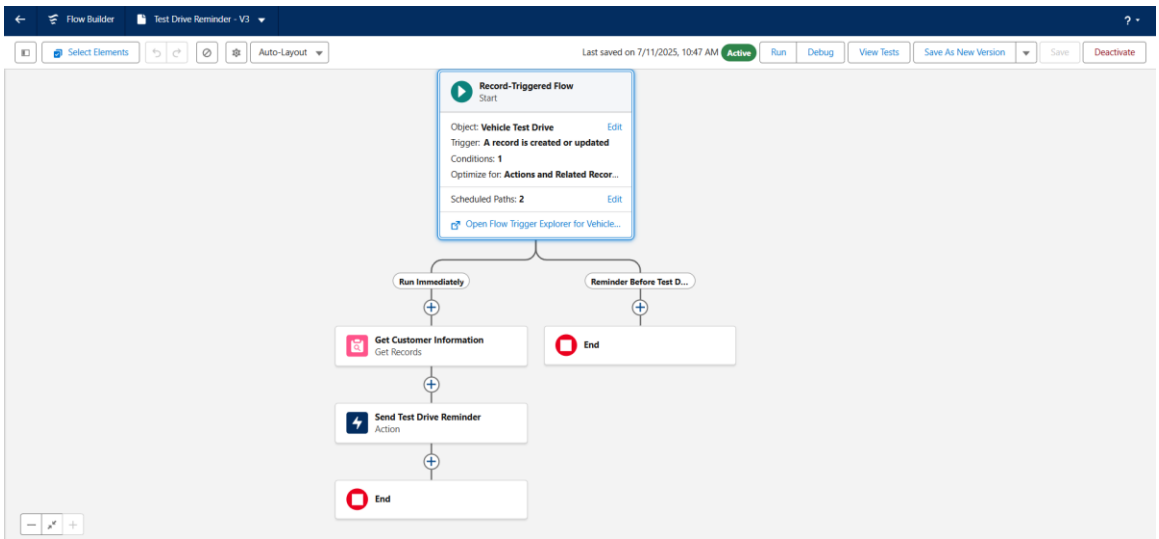


Flow 2: Test Drive Reminder

This Record-Triggered Flow:

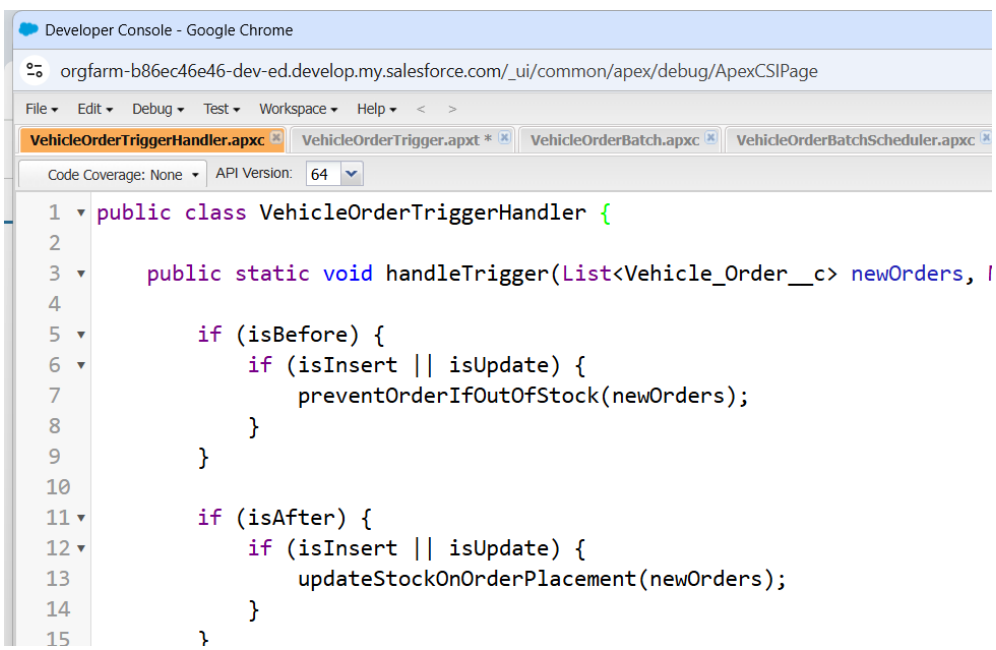
- Runs on Vehicle_Test_Drive__c creation/update

- Sends email 1 day before scheduled test drive

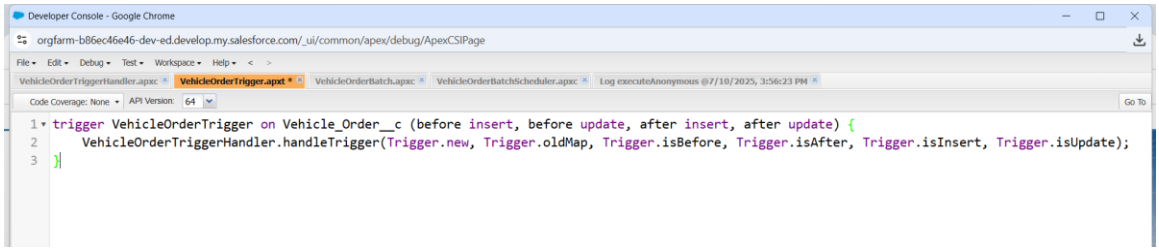


Apex Trigger & Handler

- Trigger: VehicleOrderTrigger
- Handler: VehicleOrderTriggerHandler
    - Prevents out-of-stock orders
    - Updates stock when order is confirmed

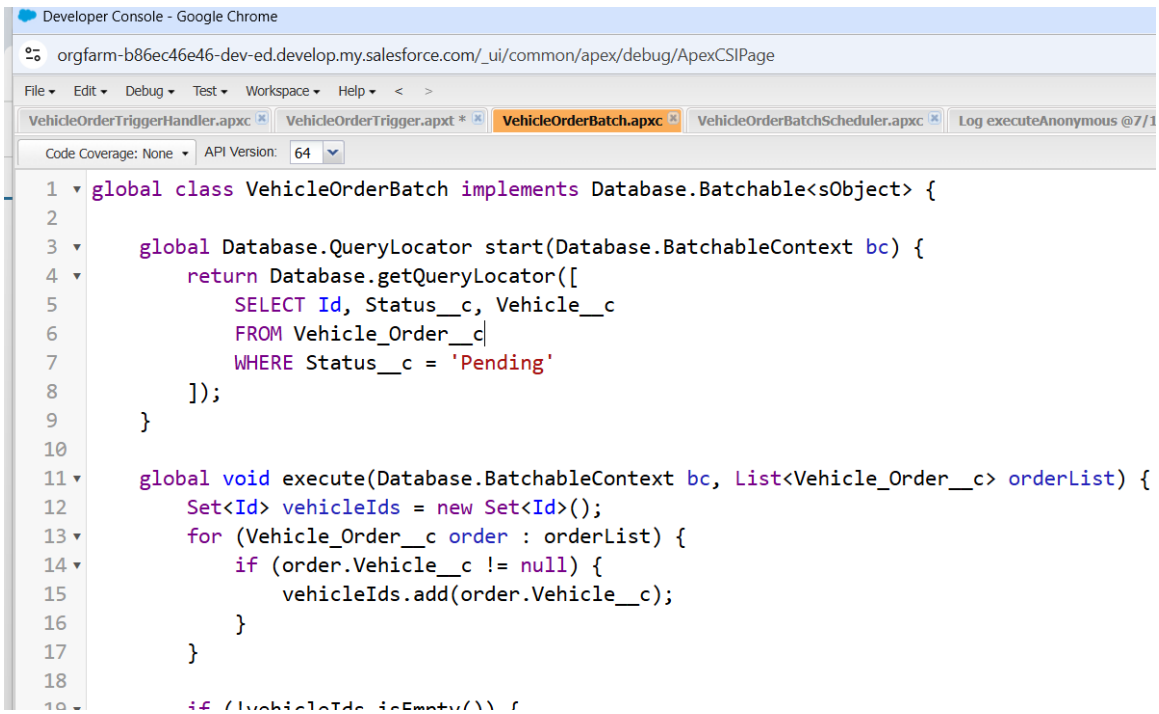



```
1  public class VehicleOrderTriggerHandler {
2
3      public static void handleTrigger(List<Vehicle_Order__c> newOrders, 
4
5          if (isBefore) {
6              if (isInsert || isUpdate) {
7                  preventOrderIfOutOfStock(newOrders);
8              }
9          }
10
11         if (isAfter) {
12             if (isInsert || isUpdate) {
13                 updateStockOnOrderPlacement(newOrders);
14             }
15         }
```



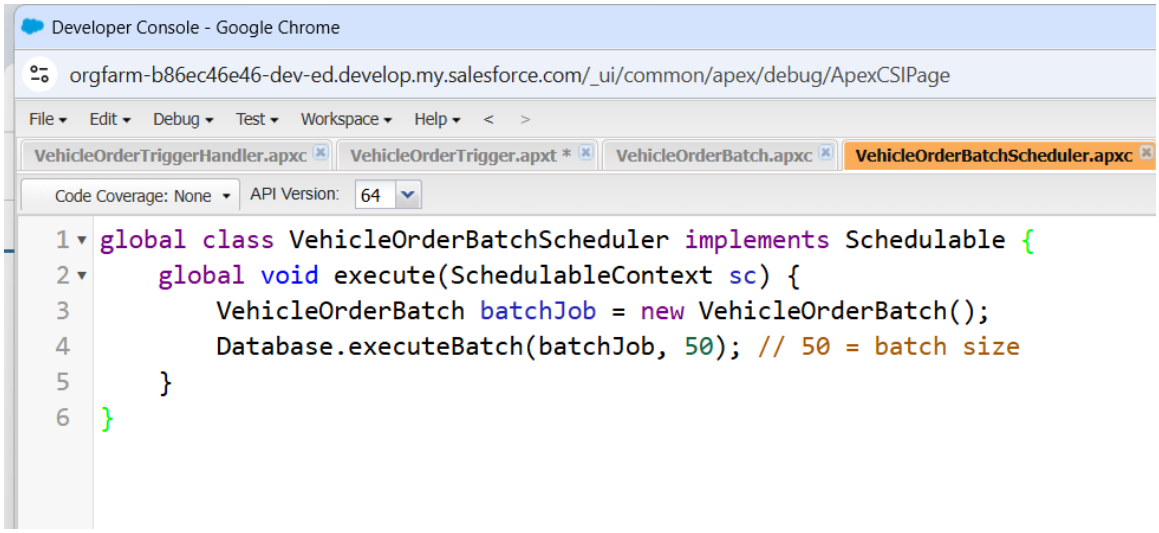
Apex Batch Class

- Class: VehicleOrderBatch
- Runs daily
- Checks for pending orders and available stock
- Updates status to *Confirmed* and adjusts stock



Scheduled Apex

- Class: VehicleOrderBatchScheduler
- Cron job runs daily at 12 PM
- Executes batch class automatically



Phase 4: Data Migration, Testing & Security

Data Loading Process

To load initial data into Salesforce (such as vehicles, dealers, and customers), the following tools were used:

Tools Used:

- **Data Import Wizard**:
  Used for importing standard object data (like Accounts, Contacts).
- **Data Loader**:
  Used for large volumes and for custom objects like Vehicle__c, Dealer__c, Order__c.

Steps:

1. Exported CSV files with sample records.
2. Mapped columns to corresponding Salesforce fields.
3. Used Data Loader to insert records for:
   - Vehicle__c
   - Dealer__c
   - Customer__c
   - Order__c (with valid relationships)

Field History Tracking, Duplicate Rules, and Matching Rules

Field History Tracking:

Enabled for the following objects to track changes:

- **Vehicle__c:** Stock__c field
- **Order__c:** Status__c and Dealer__c fields

Duplicate & Matching Rules:

- **Matching Rule:** Custom rule defined on Customer__c based on Email__c and Phone__c
- **Duplicate Rule:** Prevents duplicate customers from being inserted

Profiles, Roles, Permission Sets, and Sharing Rules

Profiles and Roles:

- Standard profiles like **Standard User** and **System Administrator** were used.
- **Role Hierarchy** established:
  - CEO
    └── Sales Manager
    └── Sales Rep

Permission Sets:

- Created **Order Management Access** permission set
- Assigned to users who need create/read access to Orders and Vehicles

Sharing Rules:

- **Public Read/Write** for most custom objects
- **Manual Sharing** allowed for sensitive customer records

Preparation of test cases for each and every salesforce features like booking creation, Approval Process, Automatic Task creation, flows, triggers etc.

1. Create a Vehicle :

INPUT:

Vehicle Name: Test Car

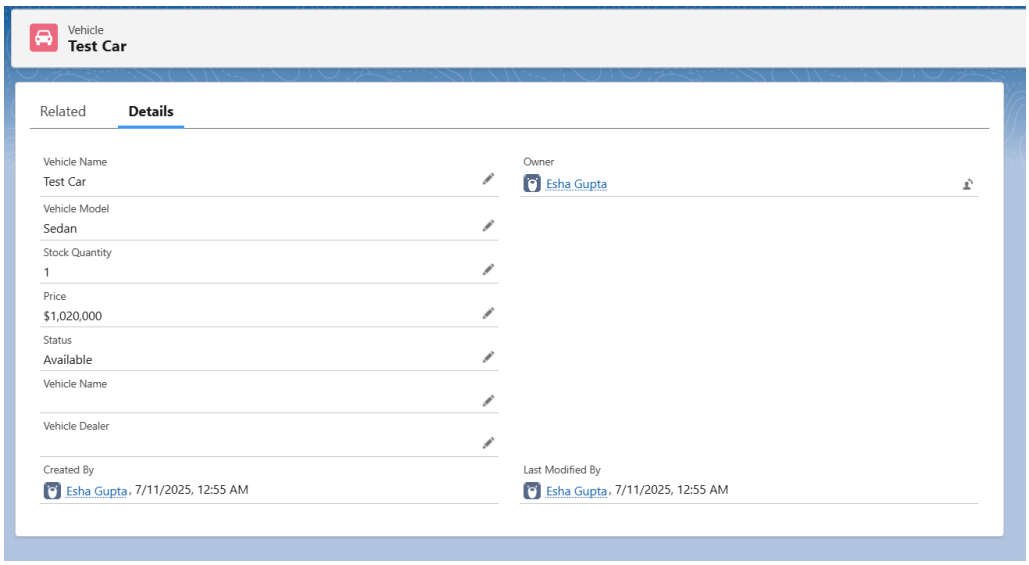
Vehicle Model: Sedan

Stock Quantity: 1

Price: 1020000

Status: Available

Dealer: Select existing Vehicle Dealer



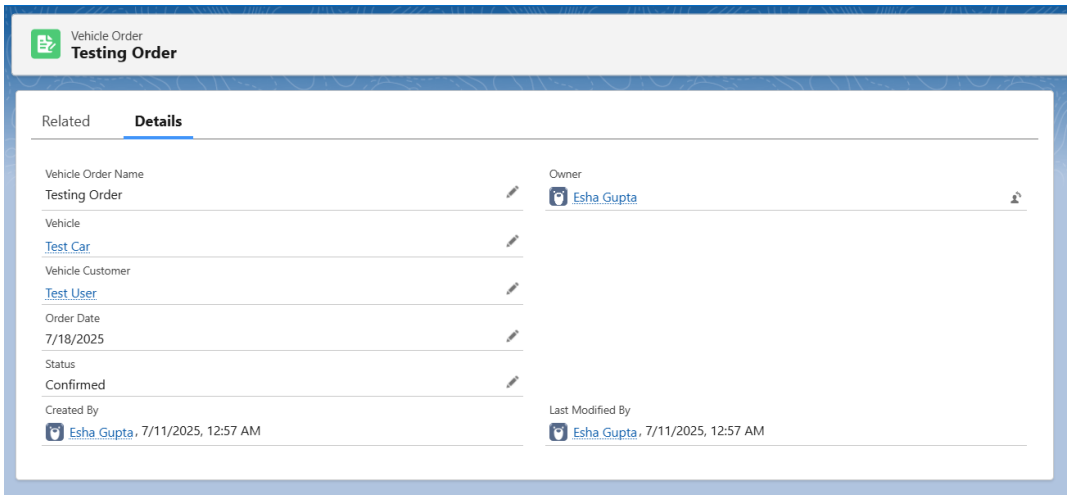
2. Test Stock = 0 (Error Case):

INPUT:

Edit the Stock Quantity of the above vehicle → Set it to 0.

Go to Vehicle Orders tab → Click New.

- **Vehicle:** Test Car
- **Status:** Confirmed
- **Customer:** Select any existing customer OUTPUT



3. Test Stock > 0 (Confirmed Order)

INPUT:

Steps:

1. Set vehicle Stock Quantity back to 1.

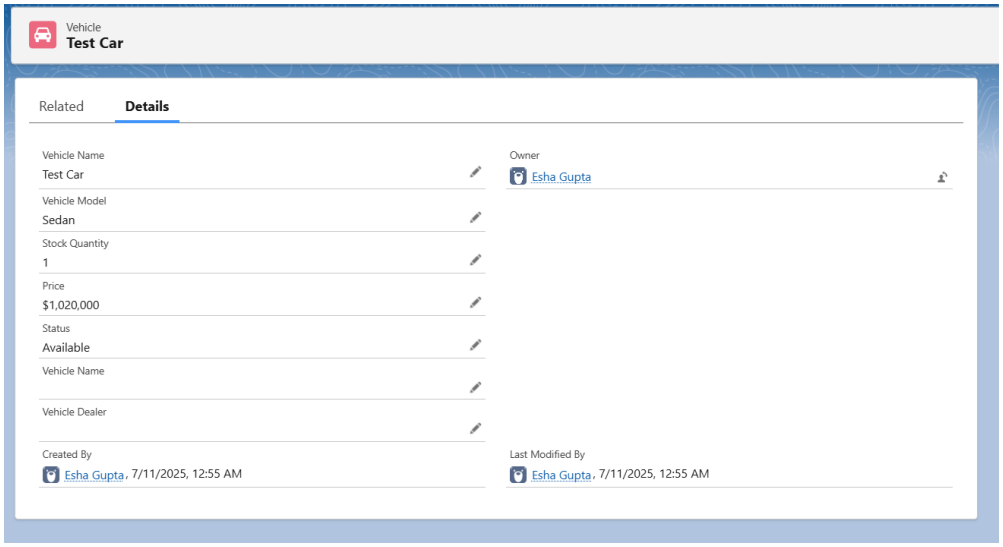
2. Create a Vehicle Order:

○ Status: Confirmed

○ Vehicle: Test Car

○ Vehicle stock should reduce from 2 → 1 automatically.

OUTPUT:

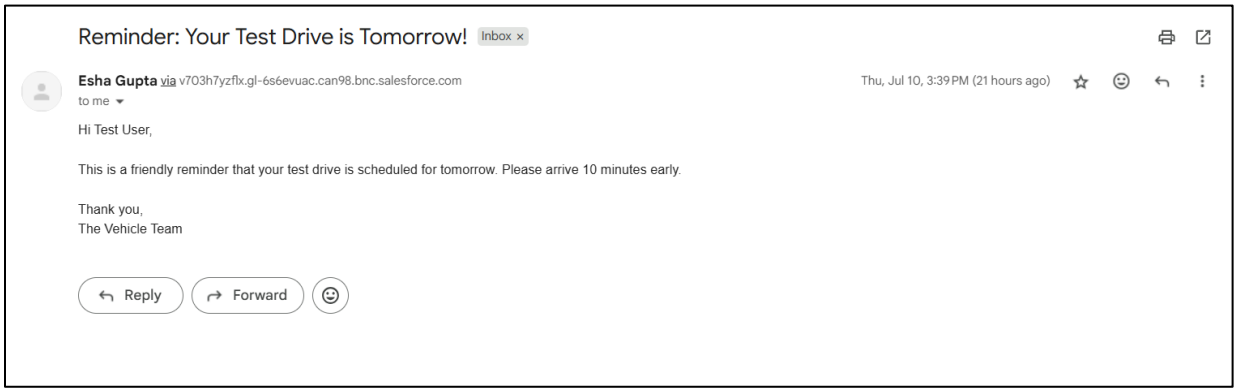


4. Test Drive Reminder Email:

Customer: Select any customer with email

Status: Scheduled Test Drive Date: Tomorrow (pick tomorrow's date)

OUTPUT:



Test Batch Job for Pending Orders :

INPUT:

Create a Pending Order when stock is 0:

1. Set Test Car stock to 0.

2. Create a Vehicle Order:

○ Status: Pending

Update stock:

● Set Stock Quantity = 1

Run batch manually:

VehicleOrderBatch job = new VehicleOrderBatch();

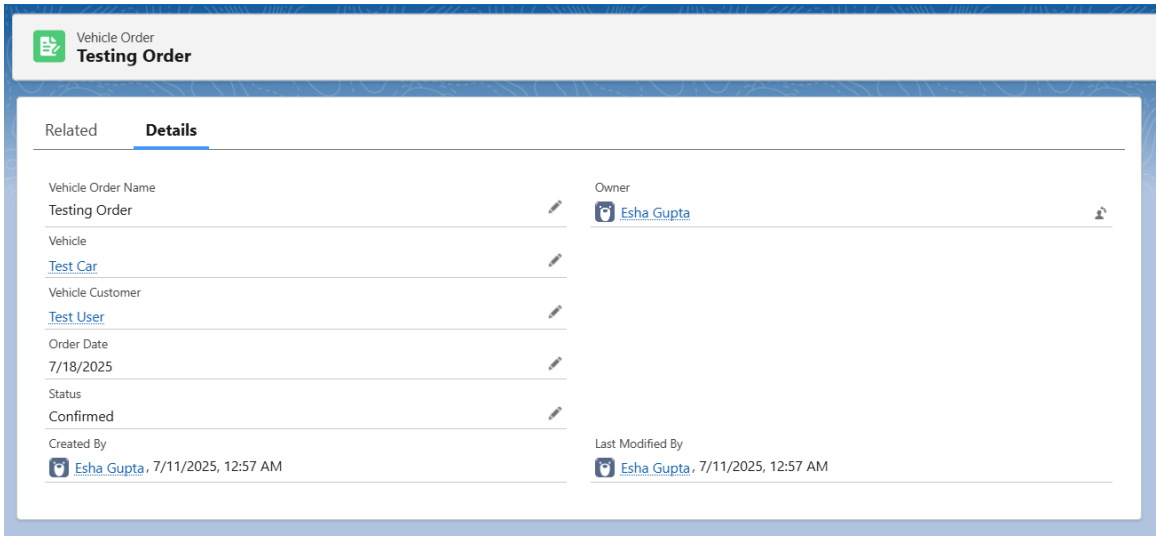
Database.executeBatch(job, 50);

OUTPUT:

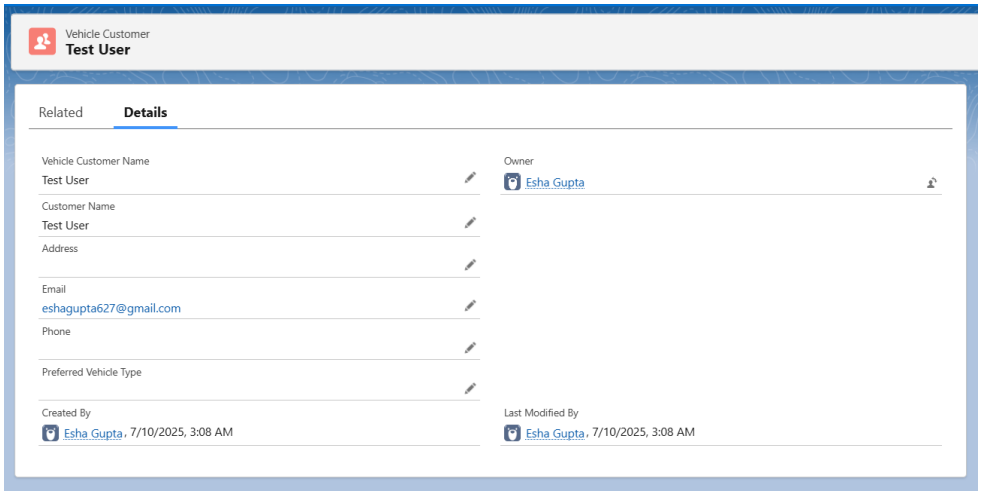
Expected Result:

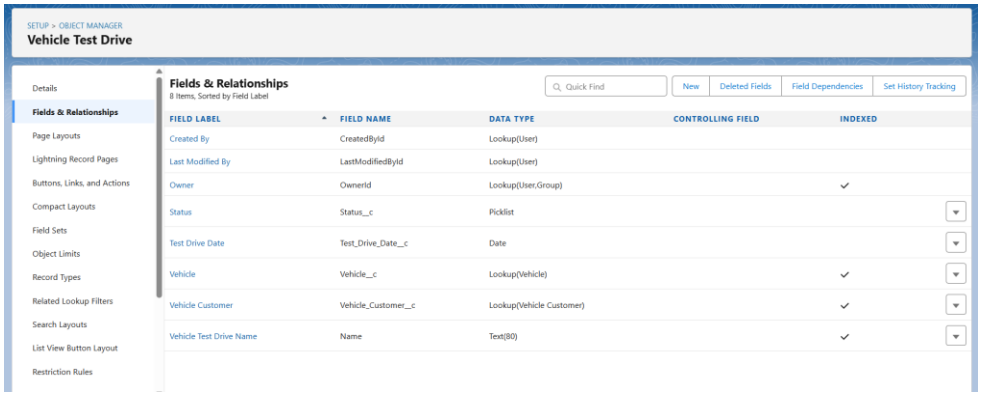
● Your Pending Order should become Confirmed.

● Vehicle stock should reduce by 1.



Creation of Test Cases





To ensure Apex code is deployable and functional, **Test Classes** were created for:

- OrderTriggerHandler
- DealerAssignmentService
- StockValidationTrigger

Test Class Features:

- Minimum 75% coverage
- Positive and negative test cases
- Used @isTest annotation with test data setup

Phase 5: Deployment, Documentation & Maintenance

Deployment Strategy

To deploy the developed features from the Developer Org to the live/production environment, the **Change Set** deployment method was used.

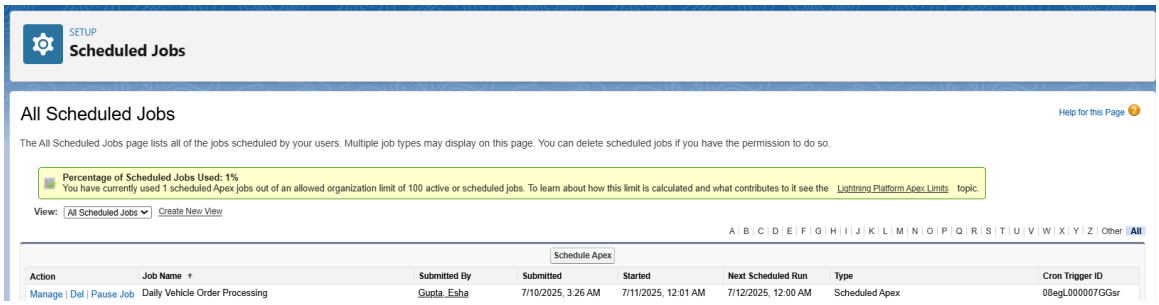
Deployment Steps:

1. Created an **Outbound Change Set** in the source org.
2. Added all custom components:
    - Custom objects, fields, flows, validation rules, triggers, and Apex classes.
3. Uploaded the Change Set to the **Target Org** (production/sandbox).
4. Validated and deployed it from **Inbound Change Sets** in the target org.
5. Post-deployment manual verification was done to ensure everything works as expected.

Testing & Sample Scenarios

Test Cases:

- Create vehicle and order with 0 stock → error
- Set stock = 2 → place order → stock becomes 1
- Create pending order → update stock → batch job confirms order



System Maintenance and Monitoring

To ensure smooth system performance after deployment, the following basic maintenance strategy was defined:

1. Monitoring

- Use **Apex Jobs** to monitor scheduled jobs or batch classes.
- Use **Debug Logs** to trace errors or unexpected behavior.
- Enable **Email Alerts** for test drive reminders or failed processes.

2. User Feedback Loop

- Sales and operations team were asked to use the system for a few days post-deployment.
- Collected feedback via manual walkthroughs to identify any missing features or issues.

3. Updates and Fixes

- Minor updates (like adding help text or updating field labels) were handled in sandbox and redeployed via Change Sets.
- Scheduled quarterly reviews for enhancements or UI improvements.

Troubleshooting Approach

If any issues arise in the production environment, the following steps will be followed:

Step 1: Reproduce the Issue

- Try to replicate the problem in a sandbox or developer org.

Step 2: Enable Debug Logs

- Set debug logs for the impacted user and analyze the flow or Apex execution.

Step 3: Check Apex Jobs or Flows

- If it's related to background processing, check Apex Job failures or Flow error emails.

Step 4: Fix and Retest

- Modify the logic (Flow or Apex).
- Retest in sandbox and re-deploy using Change Set.

Conclusion

The Salesforce implementation at **WhatsNext Vision Motors** successfully achieved its objective of streamlining the customer ordering process and improving operational workflows. Key achievements include:

- Automated **nearest dealer assignment** using Flows or Triggers
- **Stock validation** to prevent out-of-stock orders
- **Scheduled logic** to update order statuses (if Batch Apex was implemented)
- Enhanced **customer experience** through automation
- Reduced **manual intervention** for internal teams

This project not only enhances the company's customer-facing processes but also establishes a strong foundation for future Salesforce expansion and automation. Through this initiative, WhatsNext Vision Motors has moved a step closer toward its vision of **innovation and excellence in mobility.**