Smartbridge

Prepared by : **Suhail Mohammad**
University : **SRM AP**

WhatNext Vision Motors – Vehicle Management System

July 15, 2025

Project Overview

The WhatNext Vision Motors Vehicle Management System is a custom Salesforce CRM solution designed to streamline the process of vehicle orders, test drives, and service requests. It centralizes vehicle stock tracking, automates dealer assignment, and ensures seamless interaction between customers and vehicle dealers. The solution enhances operational efficiency through automated flows, triggers, and dashboards.

Key Features:

- Custom Objects for Vehicles, Dealers, Customers, Orders, Test Drives, and Service Requests
- Automated Dealer Assignment
- Reminder Emails for Test Drives
- Order Stock Validation and Update via Apex
- Dynamic Reports & Dashboards

Objectives

The primary goal of this CRM is to automate and optimize the end-to-end vehicle management process for WhatNext Vision Motors.

Objectives include:

- Improve customer experience via timely service and test drive notifications.
- Automate order assignment and prevent overbooking.
- Maintain accurate stock levels and streamline dealer-customer interaction.
- Enable real-time reporting for better decision-making

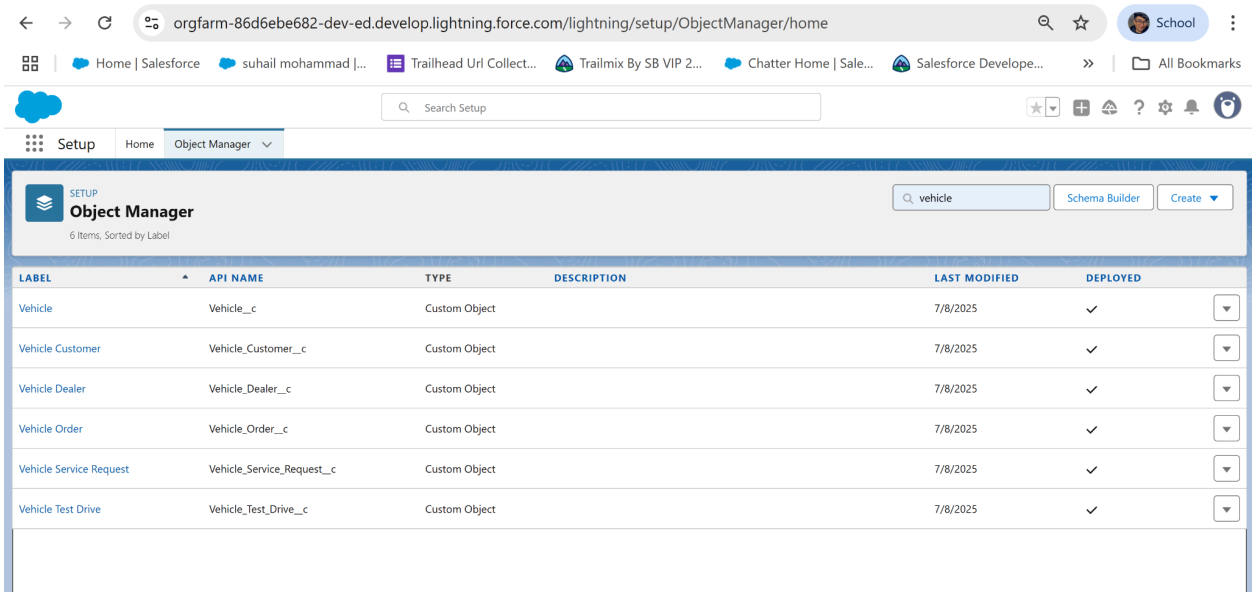
Phase 1: Requirement Analysis & Planning

Understanding Business Requirements:

- Customers need to request test drives, services, and place orders via CRM.
- Dealers need visibility into stock and customer requests.
- Admins need reports and automation to streamline operations.

Defining Project Scope & Objectives:

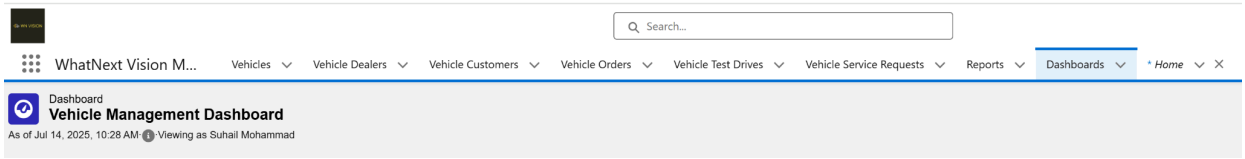
- Design a system with six custom objects (Vehicle, Dealer, Customer, etc.)



- Enable automation via flows and Apex for dealer assignment and stock validation.
- Provide dashboards for performance tracking.

Design Data Model & Security Model:

- Lookup relationships between objects to link dealers, customers, and vehicles.
- Field-level security enforced through profiles.
- Tabs created for each custom object for navigation and control.



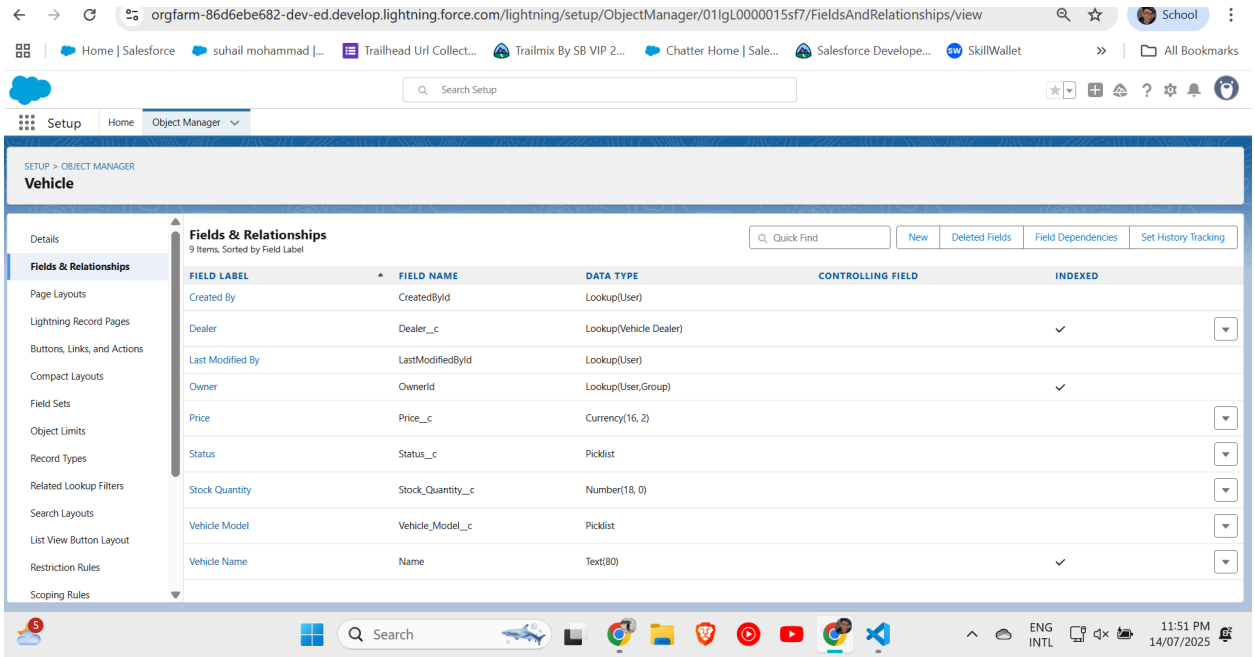
Phase 2: Salesforce Development – Backend & Configurations

Setup & DevOps:

- Developer Console used for Apex class & trigger creation.
- Flow Builder for automation.

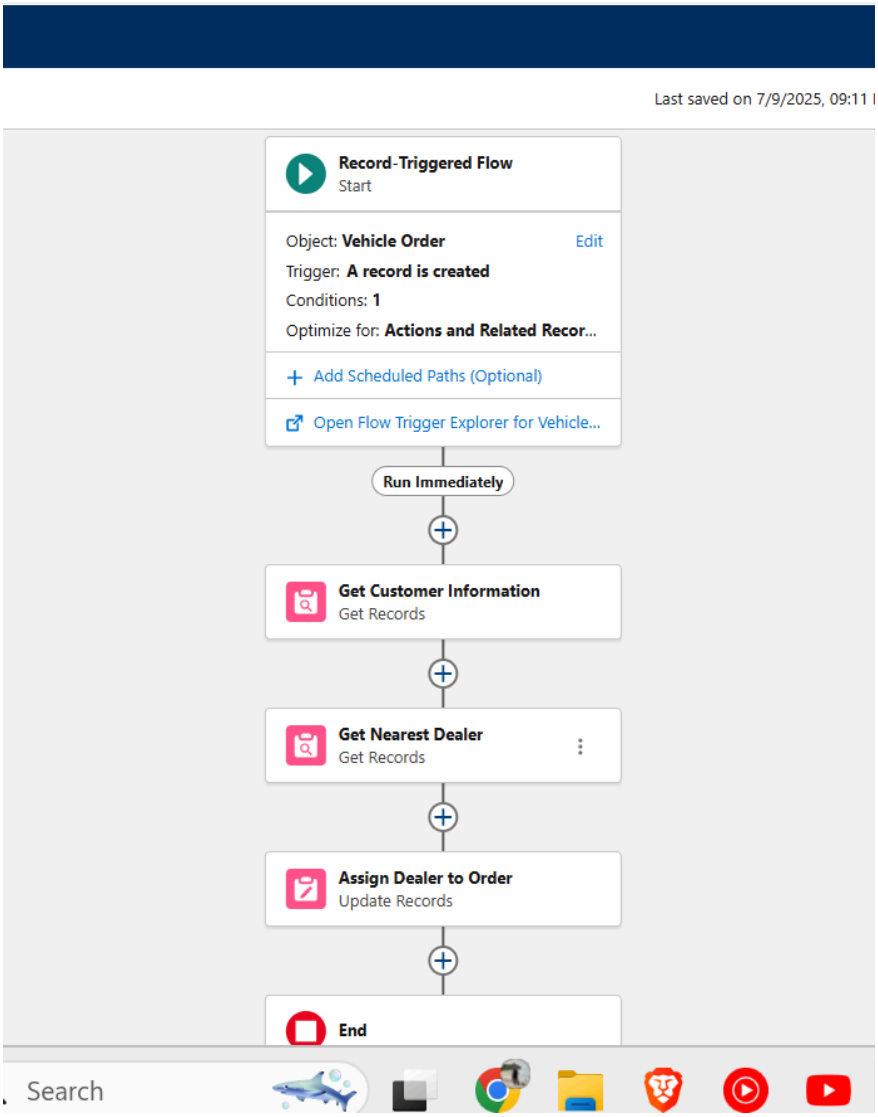
Customization:

- Created objects, fields (e.g., `Vehicle Model`, `Stock Quantity`, `Order Date`)

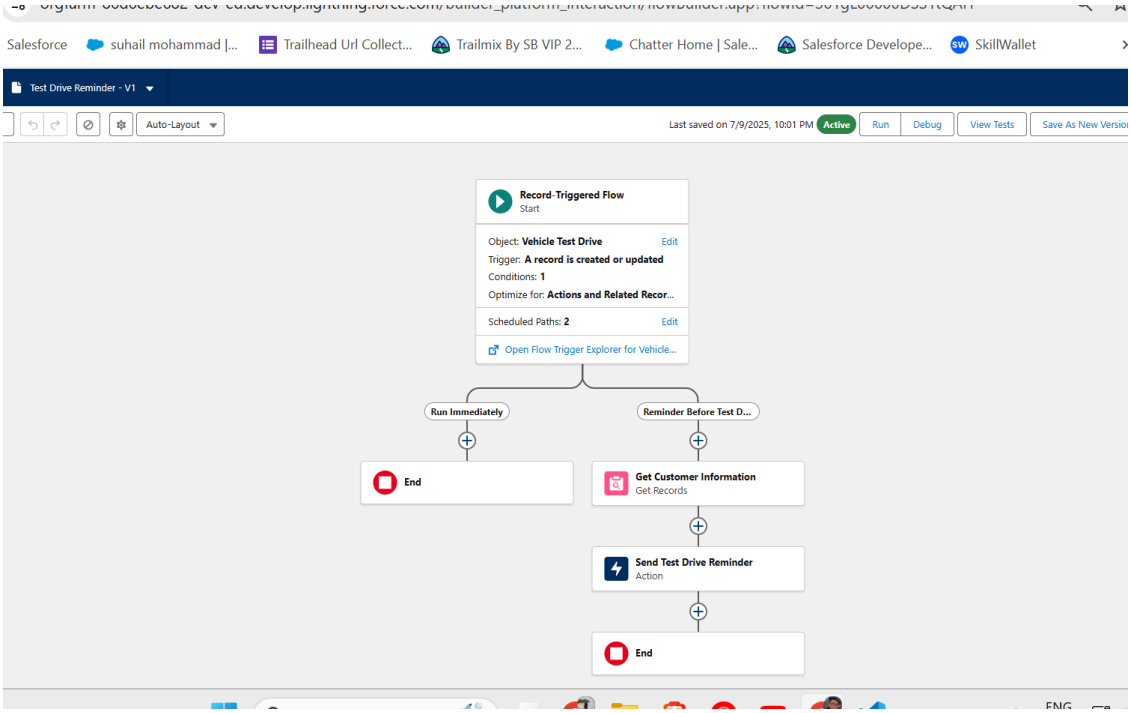


- Validation rule to prevent ordering if stock is unavailable.

- Flows:
    - Auto-assign dealer to orders



- Test drive reminder email



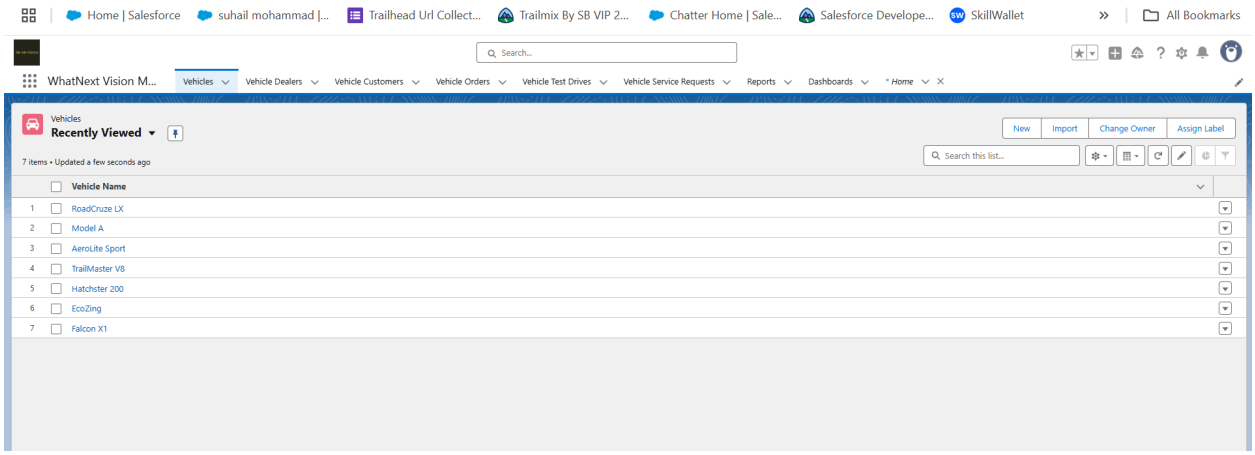
Apex:

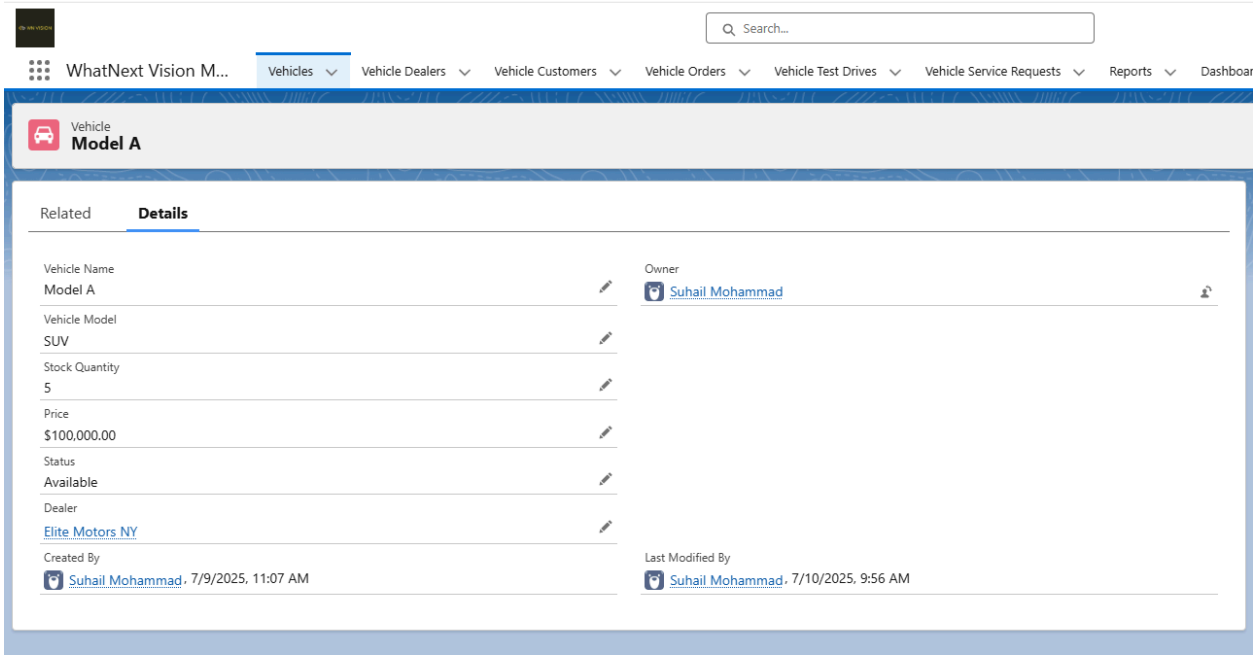
- `VehicleOrderTriggerHandler`: Prevents orders when stock is 0 and updates stock.
- Trigger `VehicleOrderTrigger`: Calls handler methods on insert/update.
- Batch Class `VehicleOrderBatch`: Confirms orders and updates stock.
- Scheduled via `VehicleOrderBatchScheduler`.

Phase 3: UI/UX Development & Customization

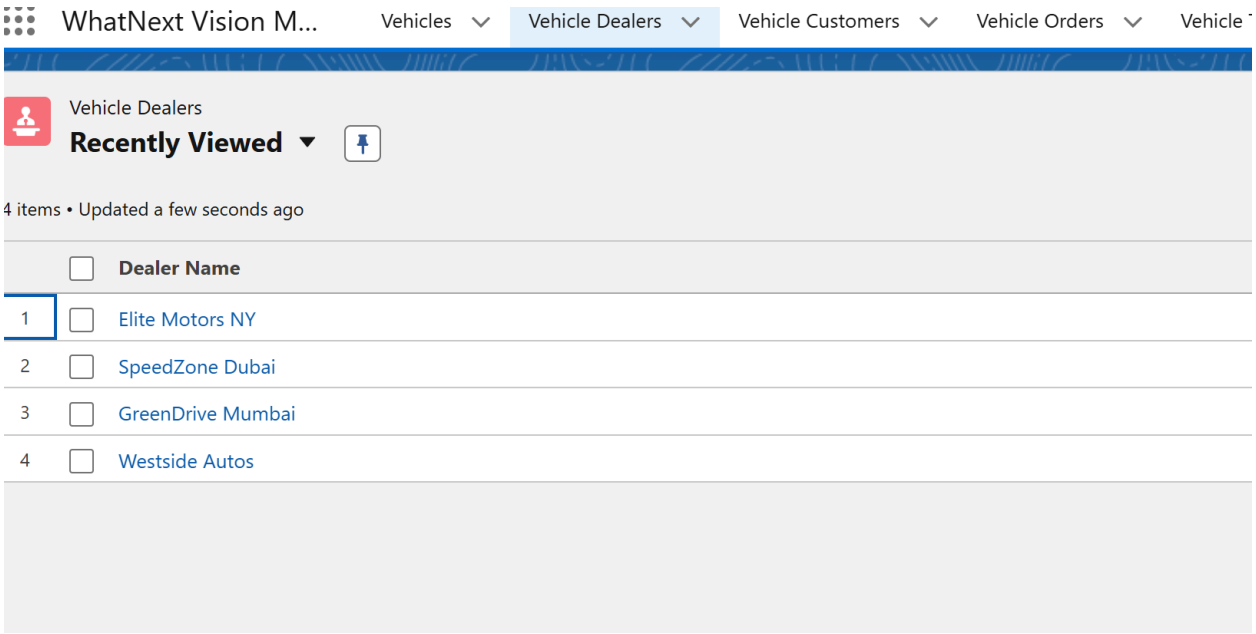
Lightning App: WhatNext Vision Motors with all key tabs.

- Vehicle Tab

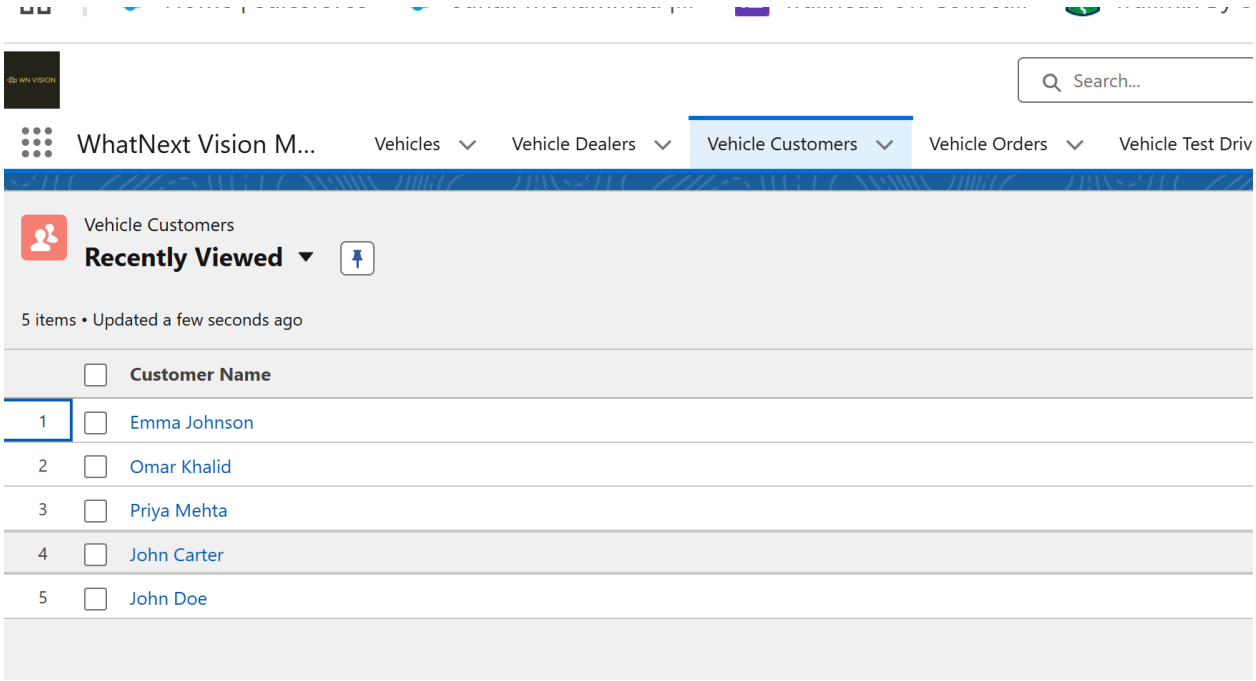




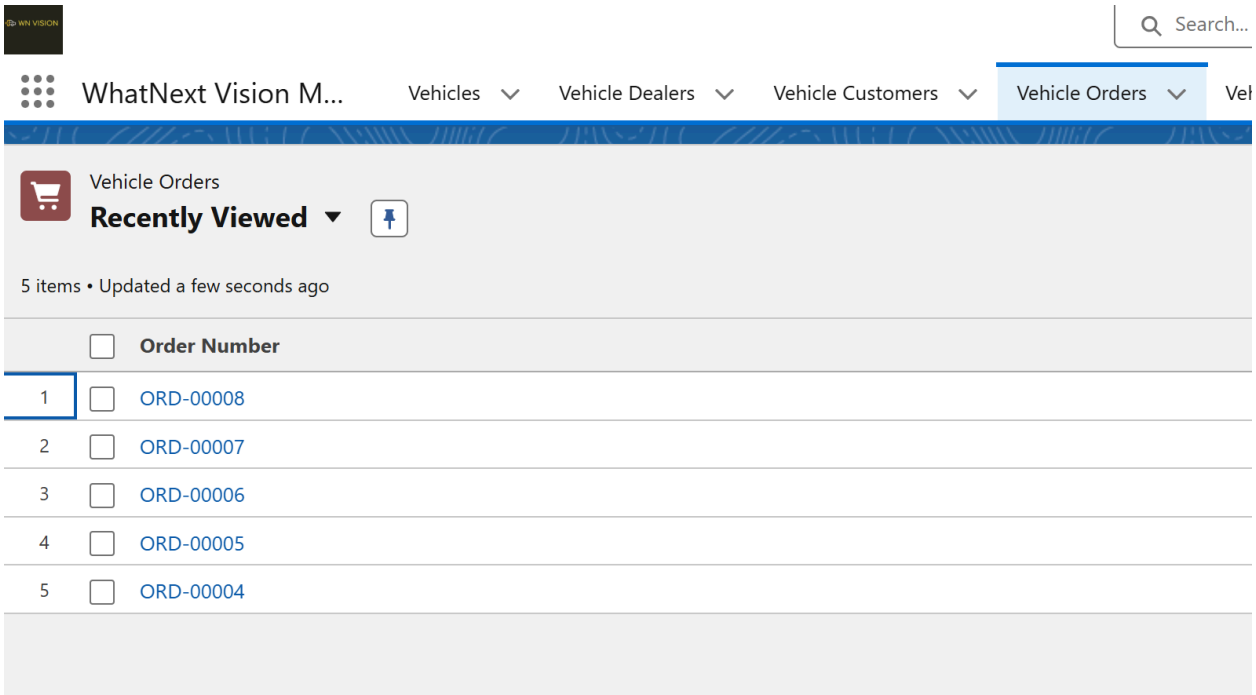
- Dealer Tab



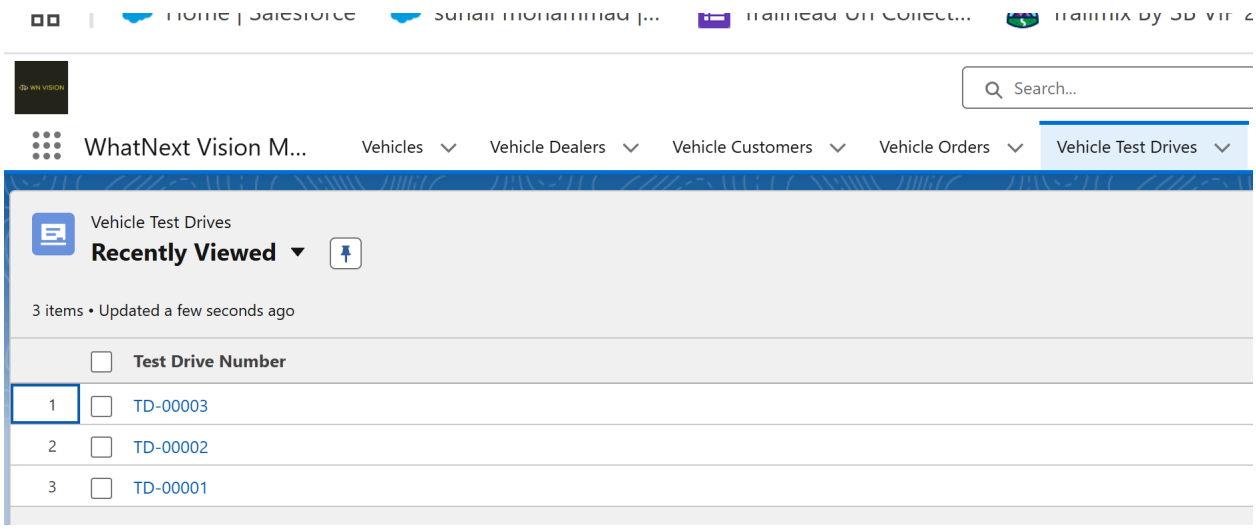
- Customer Tab



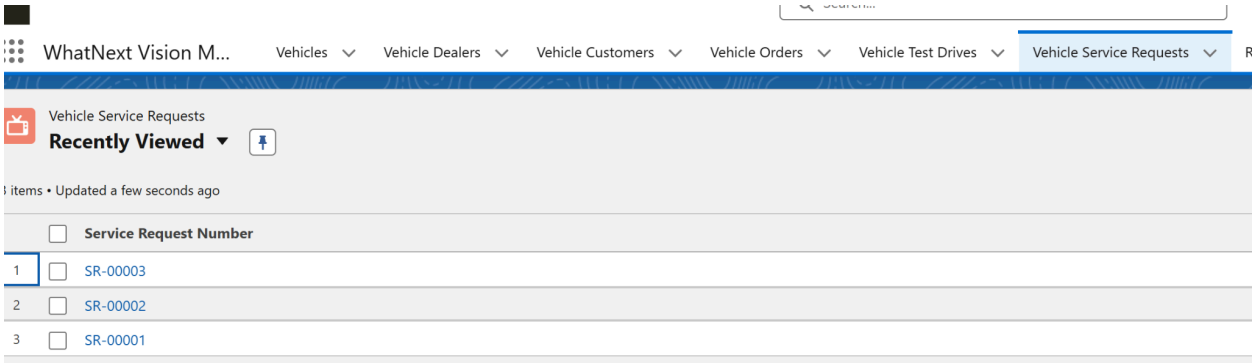
- Order Tab



- Test Drive Tab

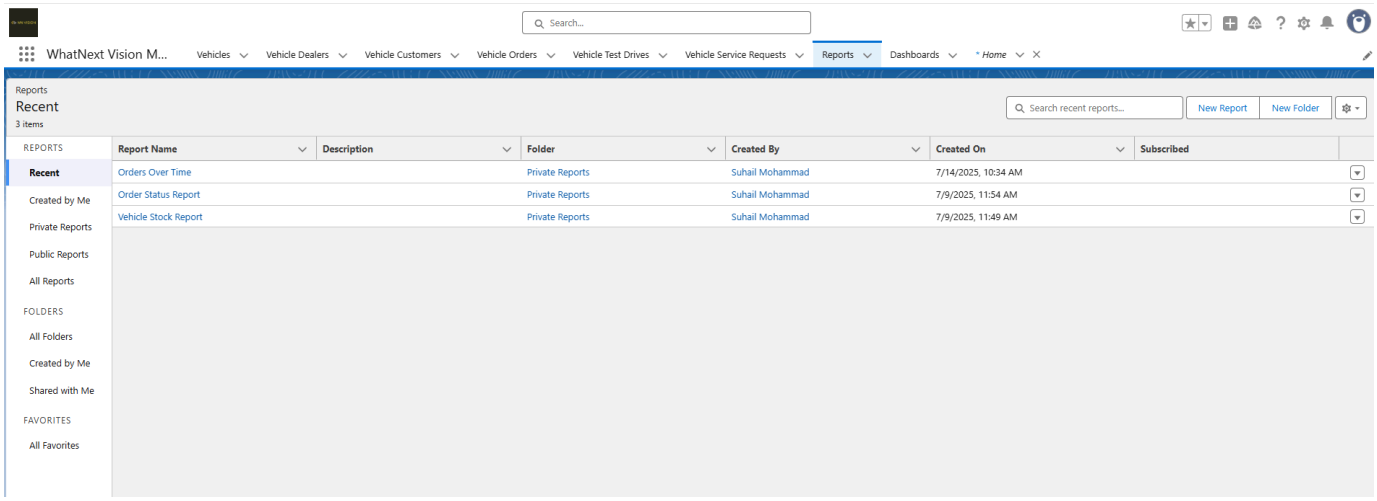


- Service Request Tab

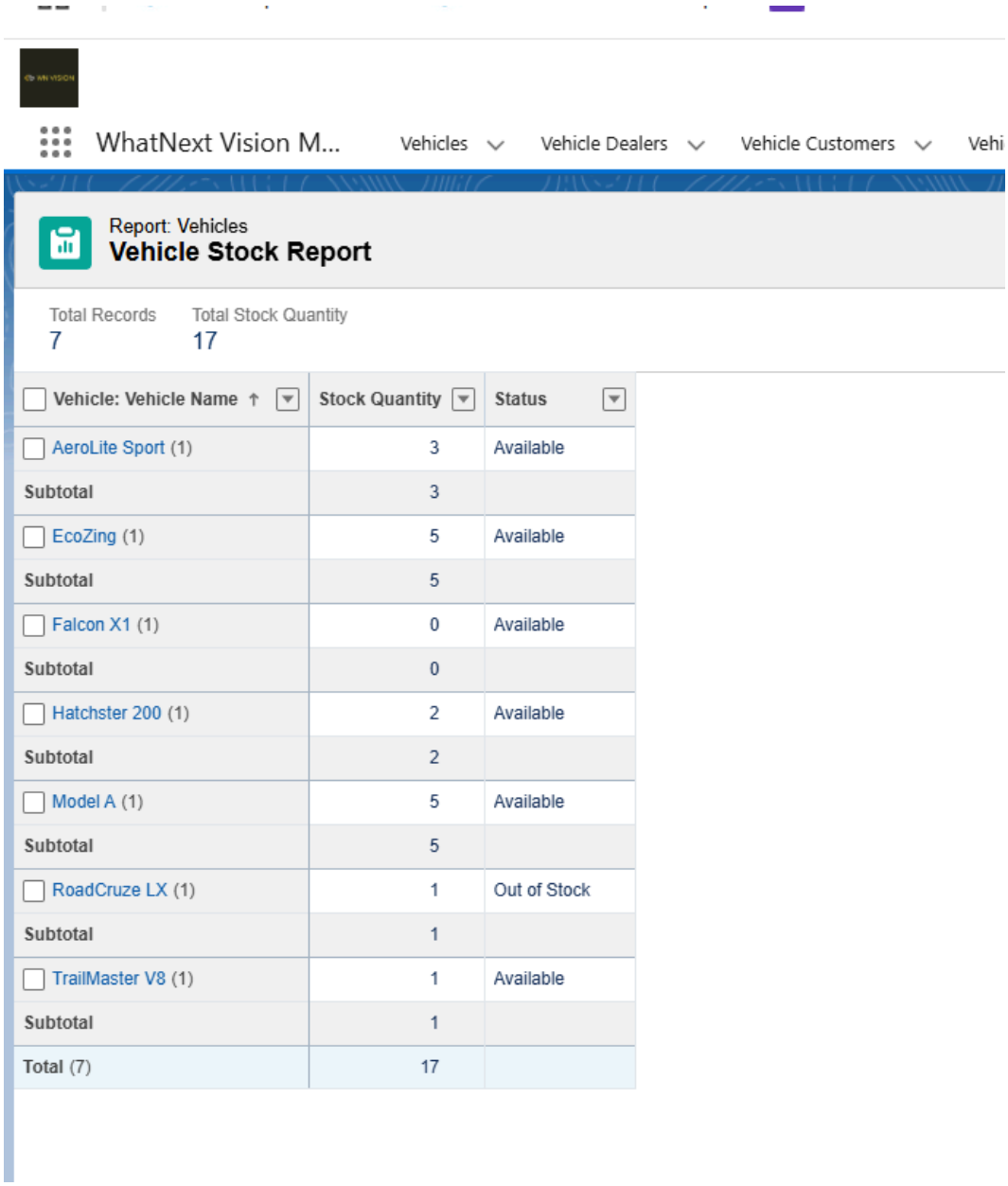


Page Layouts customized per object with relevant fields.

Reports & Dashboards:

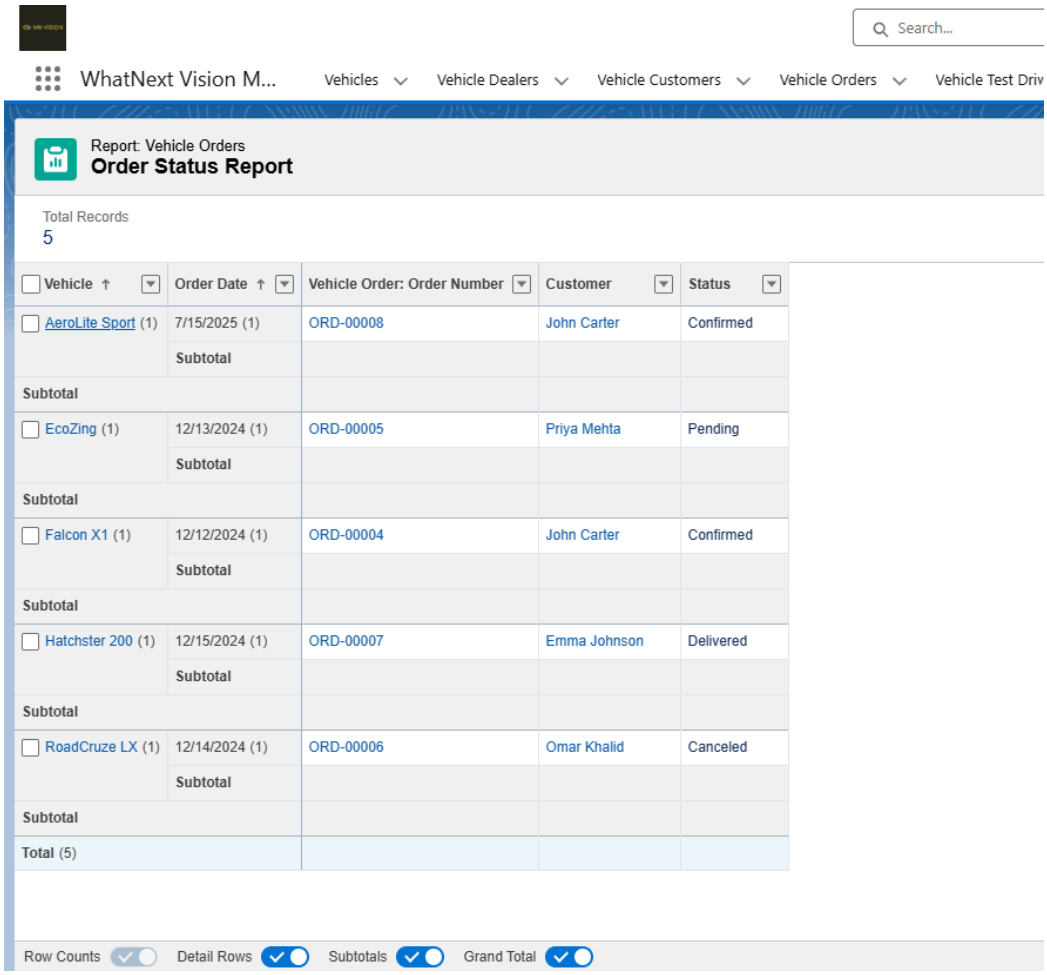


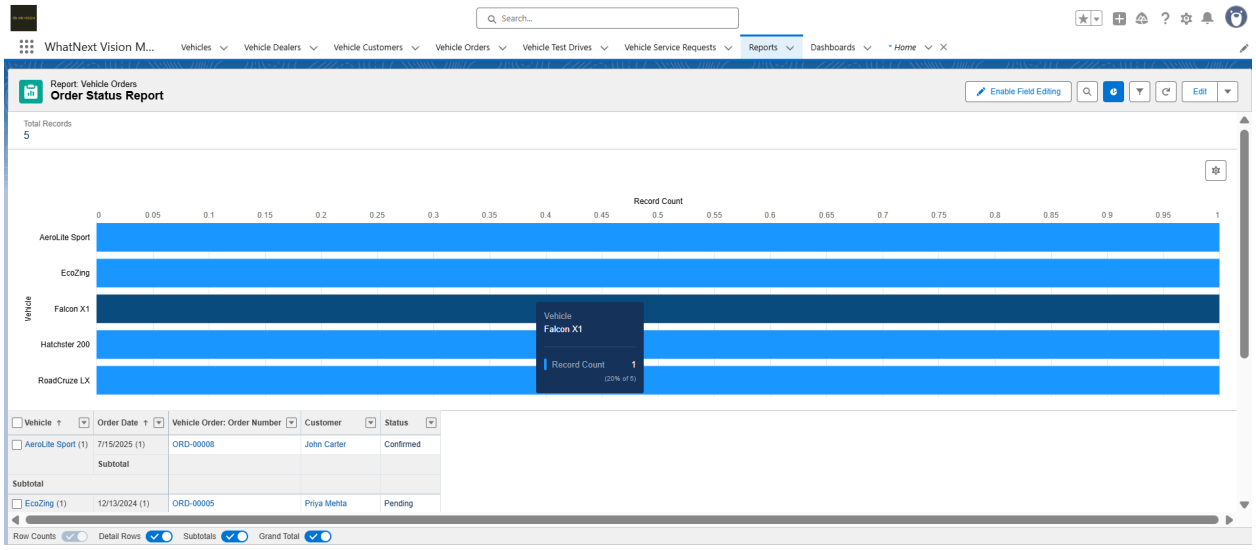
- Vehicle Stock Report



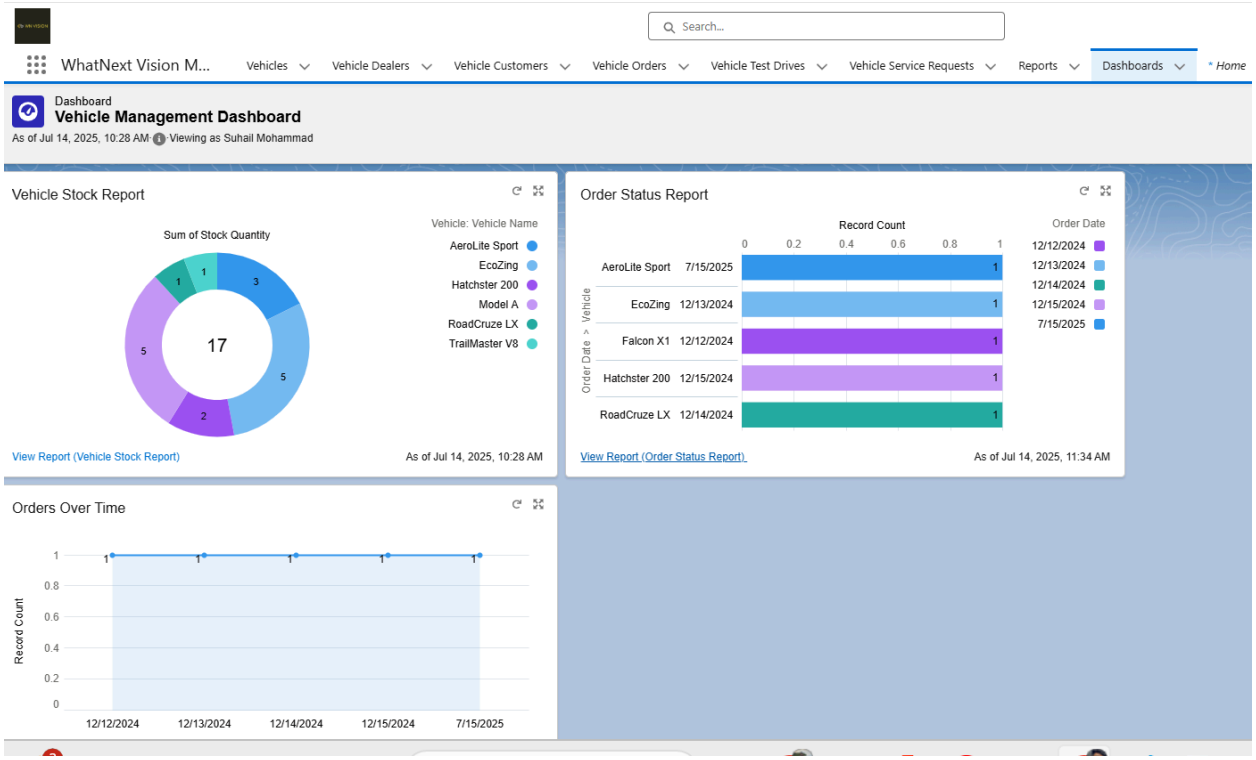
Vehicle: Vehicle Name	Stock Quantity	Status
AeroLite Sport (1)	3	Available
Subtotal	3	
EcoZing (1)	5	Available
Subtotal	5	
Falcon X1 (1)	0	Available
Subtotal	0	
Hatchster 200 (1)	2	Available
Subtotal	2	
Model A (1)	5	Available
Subtotal	5	
RoadCruze LX (1)	1	Out of Stock
Subtotal	1	
TrailMaster V8 (1)	1	Available
Subtotal	1	
Total (7)	17	

- Order Status Report





- Vehicle Management Dashboard



Phase 4: Data Migration, Testing & Security

Data Loading:

- Manual entry of test data via record pages.
- Could support Data Loader for bulk insert.

Field History, Matching & Duplicate Rules:

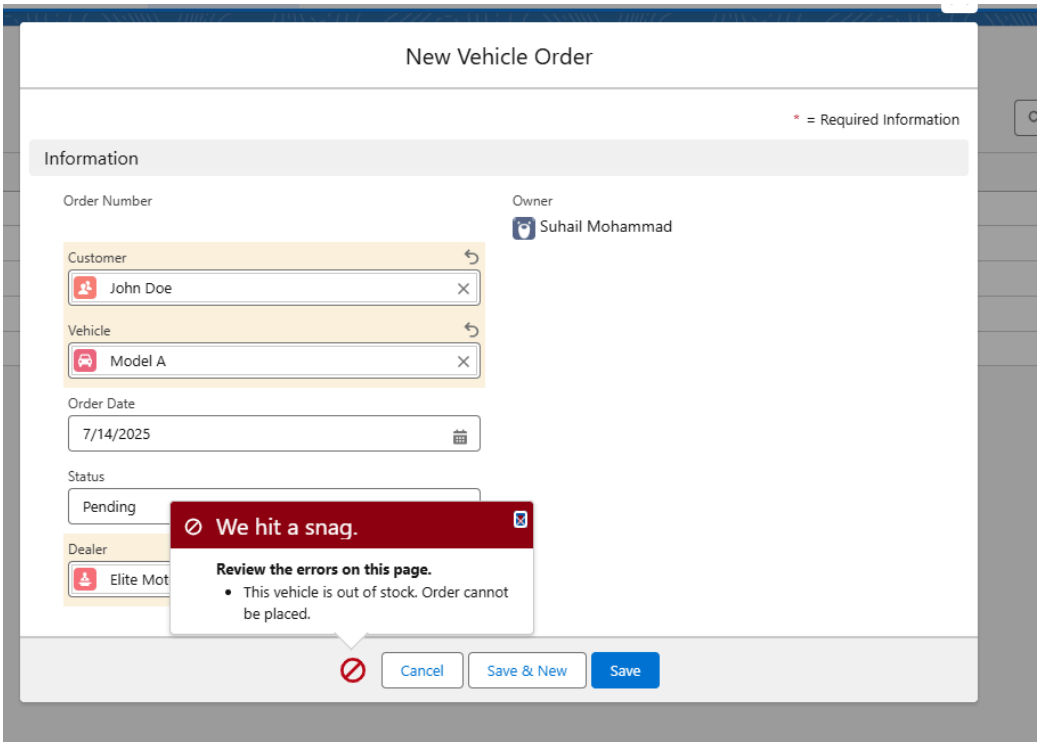
- Status fields used for tracking orders, services, and test drives.
- Matching Rules for dealers based on location.

Security Model:

- Roles: Admin, Dealer, Customer Service
- Profiles & Permission Sets created as needed.
- Sharing Rules applied for visibility control.

Testing:

- Created test cases:
    - **Order Creation:** Verified dealer auto-assignment.
    - **Test Drive Reminder:** Verified scheduled emails.
    - **Stock Validation:** Prevented order for out-of-stock vehicles.



Phase 5: Deployment, Documentation & Maintenance

Deployment Strategy:

- Components moved using Change Sets.
- Batch jobs scheduled using Anonymous Apex.

Maintenance:

- Regular monitoring of Scheduled Jobs & Error Logs.
- Logs tracked via Debug Logs and Scheduled Job Monitoring.

Troubleshooting:

- Trigger issues handled via logs & debug points.
- Batch failures debugged using test runs

Future Enhancements

- Chatbot for vehicle selection assistance.
- AI recommendation engine for suggesting vehicles.
- Customer portal with self-service options.

Conclusion

The Vehicle Management System for WhatNext Vision Motors successfully integrates multiple Salesforce features to build a functional and scalable CRM. It automates business processes, improves user experience, and provides actionable insights through reports. The project demonstrates effective use of Flow, Apex, Data Modeling, and UI customization.

Annexure

Codes:

VehicleOrderTriggerHandler

```
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);
      }
    }
  }
```

```
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.');
                }
            }
```

```
        }
    }
}

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;
        }
    }
  }
}
```

`VehicleOrderTrigger`:

```
 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);
 }
```

21

`VehicleOrderBatch`:

```
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>();


      for (Vehicle__c vehicle : [
        SELECT Id, Stock_Quantity__c
```

```
        FROM Vehicle__c
        WHERE Id IN :vehicleIds
    ]) {
        vehicleStockMap.put(vehicle.Id, vehicle);
    }

    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.');
  }
}
```

`VehicleOrderBatchScheduler`:

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