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

Abstract

WhatsNext Vision Motors, an emerging automotive leader, aims to provide exceptional customer experience by digitalizing and streamlining its vehicle ordering and dealership operations. This project leverages Salesforce CRM to create an intelligent system that automates dealer assignment, order status management, test drive scheduling, and stock validation.
As a developer, my responsibility was to translate real-world business needs into scalable technical solutions within Salesforce. Through custom objects, automation tools like Flows and Apex, and scheduled processes, this implementation delivers a practical and powerful CRM application.

Project Objectives

The key objectives were:

- To eliminate manual effort in assigning dealers by automating based on location proximity.
- To ensure vehicle availability is verified before order placement.
- To send automated reminders to customers before their test drives.
- To manage stock updates and order status transitions using scheduled logic.
- To increase dealership productivity, improve order accuracy, and elevate customer satisfaction.

Technology Stack

- **Salesforce Platform (Developer Org)**
- **Lightning App Builder**
- **Custom Objects & Fields**
- **Process Automation:**
  - Record-Triggered Flows
  - Scheduled Flows
  - Apex Triggers
  - Apex Classes
  - Batch Apex
  - Scheduled Apex
- **Email Templates & Alerts**
- **Salesforce Scheduler & Cron Jobs**
- **Developer Console & Debug Logs**
- **GitHub** (for version control and submission)

Implementation Breakdown

Task 1: Developer Org Setup

Created and verified a Salesforce Developer Edition account. Ensured it had access to Lightning App Builder, Apex development, and flow automation tools.

Task 2: Creating Custom Objects

Created custom objects to represent key business entities:

a. Vehicle__c
   Purpose: Stores complete information about the vehicles available for purchase or test drives.
b. Vehicle_Dealer__c
   Purpose: Stores information about authorized dealers who sell and manage vehicles.
c. Vehicle_Customer__c
   Purpose: Maintains customer profiles who are placing orders or booking test drives.
d. Vehicle_Order__c
   Purpose: Tracks vehicle purchase orders placed by customers.
e. Vehicle_Test_Drive__c
   Purpose: Tracks bookings for test drives made by customers.
f. Vehicle_Service_Request__c
   Purpose: Tracks service and maintenance requests raised by customers.

Each object was built with standard practices including:

- Master-detail and lookup relationships.
- Required validations (like non-zero quantity).
- Default record types and page layouts.

Task 3: Adding Tabs and App

Built a custom **Lightning App** titled **Vehicle Order Management** using the App Manager. Tabs included:

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

Added icons, branding, and visibility permissions.

This made navigation easier and helped simulate a real-world CRM dashboard.

Task 4: Adding Custom Fields

Each object was enhanced with relevant fields. Examples:

Vehicle__c

Field Name	**Data Type**	**Description**
Vehicle_Name__c	Text	The name of the vehicle (e.g., Nexon EV, XUV700).
Vehicle_Model__c	Picklist	Type of vehicle model (e.g., Sedan, SUV, EV).
Stock_Quantity__c	Number	Number of units available in stock.
Price__c	Currency	Price of the vehicle in INR.
Dealer__c	Lookup	References the authorized dealer (Dealer__c).
Status__c	Picklist	Indicates if the vehicle is Available, Out of Stock, or Discontinued.

Vehicle_Dealer__c

Field Name	Data Type	Description
Dealer_Name__c	Text	Name of the dealer (e.g., TATA Gurugram).
Dealer_Location__c	Text	Geographical location of the dealer.
Dealer_Code__c	Auto Number	Unique auto-generated code for each dealer.
Phone__c	Phone	Dealer contact number.
Email__c	Email	Dealer email address.

Vehicle_Order__c

Field Name	**Data Type**	**Description**
Customer__c	Lookup	References Vehicle_Customer__c object.
Vehicle__c	Lookup	References Vehicle__c object.
Order_Date__c	Date	Date on which the order was placed.
Status__c	Picklist	Order status: Pending, Confirmed, Delivered, or Canceled.

Vehicle_Customer__c

Field Name	Data Type	Description
Customer_Name__c	Text	Full name of the customer.
Email__c	Email	Email address of the customer.
Phone__c	Phone	Contact number.
Address__c	Text	Full postal address.
Preferred_Vehicle_Type__c	Picklist	Indicates customer's preferred vehicle type (Sedan, SUV, EV).

Vehicle_Test_Drive__c

Field Name	Data Type	Description
Customer__c	Lookup	References Vehicle_Customer__c object.
Vehicle__c	Lookup	References Vehicle__c object.
Test_Drive_Date__c	Date	Scheduled date for the test drive.
Status__c	Picklist	Status: Scheduled, Completed, or Canceled.

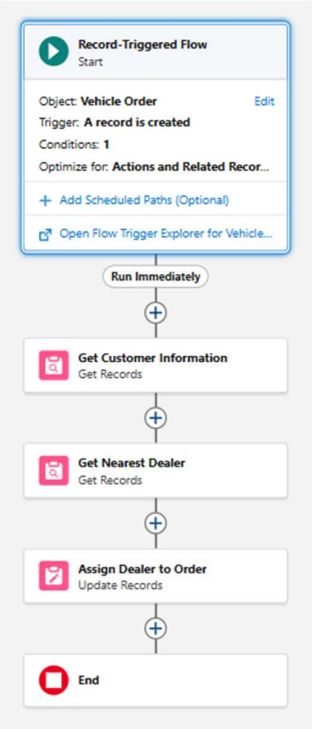
Vehicle_Service_Request__c

Field Name	Data Type	Description
Customer__c	Lookup	References Vehicle_Customer__c object.
Vehicle__c	Lookup	References Vehicle__c object.
Service_Date__c	Date	Scheduled date of vehicle servicing.
Issue_Description__c	Text	Description of the issue reported by the customer.
Status__c	Picklist	Service status: Requested, In Progress, or Completed.

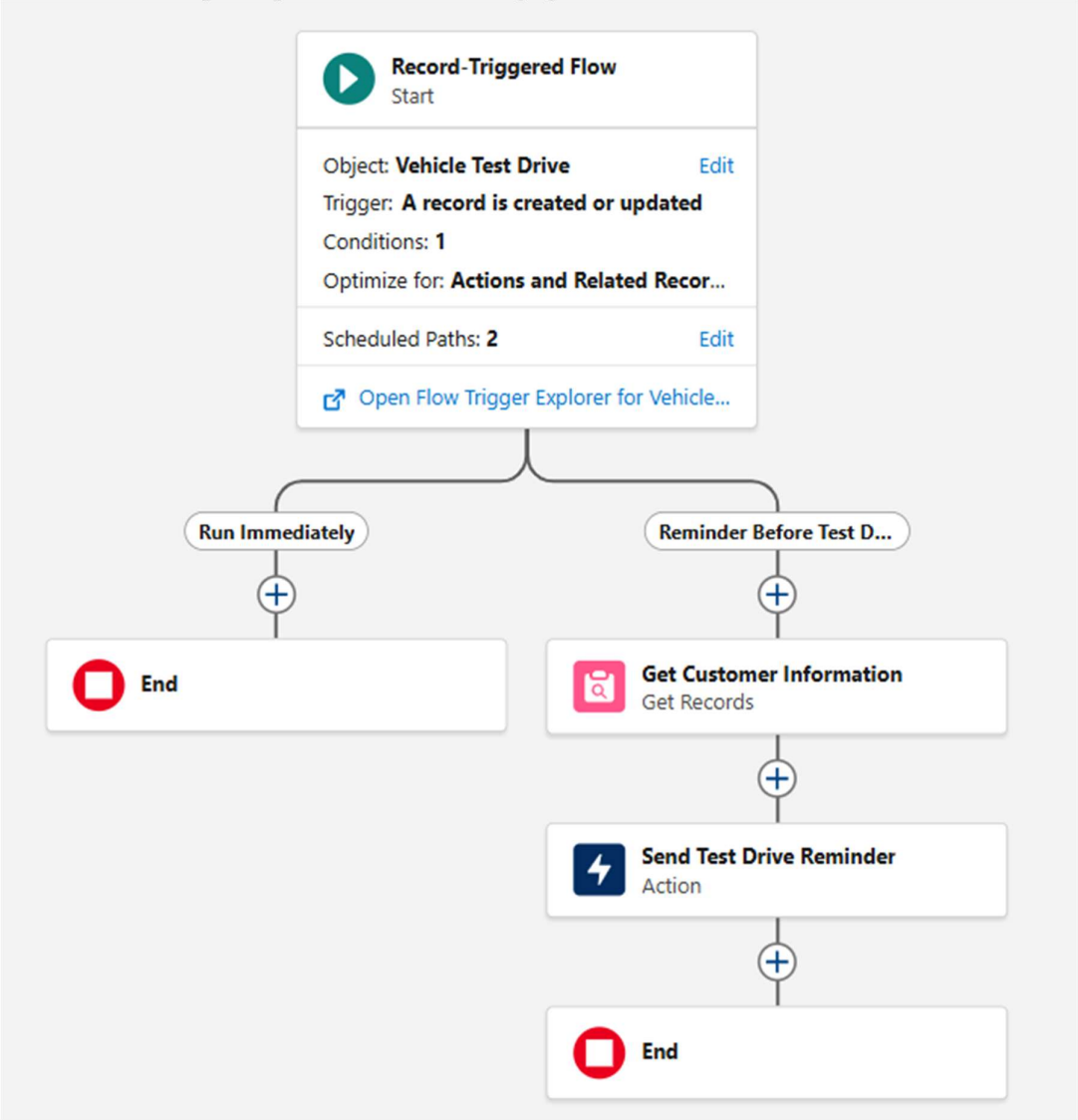
Task 5: Record-Triggered Flows

Created two **Flows** using Salesforce Flow Builder:

- **Assign Dealer Flow**
  - Triggered when a new order is created.
  - Based on the customer's location, it auto-assigns the nearest dealer from the Dealer__c object.



- **Test Drive Reminder Flow**
  - Triggered 24 hours before a scheduled test drive.
  - Sends an email alert with details to the customer.
  - Helps improve customer engagement and reduces no-shows.



Task 6: Apex Triggers and Classes

- **Trigger: VehicleOrderTrigger**
  - Fires before insert and before update on the Vehicle_Order__c object.
  - Ensures stock is available before confirming the order.
  - Deducts vehicle stock after confirmation.
  - Prevents duplicate or invalid orders.

```
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);
}
```

- **Class: VehicleOrderTriggerHandler**
  - Modular logic for validating vehicle stock and updating order total.
  - Separates logic from the trigger for better readability and testability.

```
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 && (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) {
                    order.addError('This vehicle is out of stock. Order cannot be
placed.');
                }
            }
        }
    }

    // ☑ Decrease stock when an order is confirmed
    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>(
                [SELECT Id, Stock_Quantity__c FROM Vehicle__c WHERE Id IN
:vehicleIds]
            );

            List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();
            for (Vehicle_Order__c order : orders) {
                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
                if (vehicle != null && vehicle.Stock_Quantity__c > 0) {
                    vehicle.Stock_Quantity__c -= 1;
                    vehiclesToUpdate.add(vehicle);
                }
            }

            if (!vehiclesToUpdate.isEmpty()) {
                update vehiclesToUpdate;
            }
        }
    }
}
```

Task 7: Batch Apex + Scheduler

- **Batch Class: VehicleOrderBatch**
  - Periodically scans pending orders.
  - If stock becomes available, it updates the status to "Confirmed."
  - Useful when stock is replenished but orders are still pending.

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

- **Scheduler Class: VehicleOrderBatchScheduler**
  - Uses System.schedule() with a CRON expression to run daily at 12 PM.
  - Automates the batch job without manual intervention.

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

- **Deployment of Scheduler**
  Scheduled via Developer Console

Task 8: Email Alerts and Templates

Created email templates for:

- Test drive confirmation and reminders.
- Order confirmation emails.

These are used in Flow-based automation to notify users at appropriate events.

Real-Life Simulation Example

A customer named Raj from Mumbai places an order for a sedan.
The system checks which Mumbai dealer has that sedan in stock.
Assigns the order to the nearest dealer.
If stock is unavailable, the order remains in pending.
Once stock is available the next day, the scheduler confirms the order.
An email is sent 24 hours before his test drive.
This end-to-end flow runs without human involvement.
Scheduler tested using Test.startTest() and Test.stopTest().

Folder Structure for GitHub

```
whatsnext-salesforce-crm/
├── README.md
├── Project_Documentation.pdf
├── Screenshots/
│   ├── flow_assign_dealer.png
│   └── flow_test_drive.png
│
├── Source_Code/
│   ├── VehicleOrderTrigger.apex
│   ├── VehicleOrderTriggerHandler.apex
│   ├── VehicleOrderBatch.apex
│   └── VehicleOrderBatchScheduler.apex
└── Demo_Video_Link.md
```

Conclusion

The Vehicle Management System built on Salesforce provides a comprehensive and centralized solution for managing vehicle inventory, dealers, customers, orders, test drives, and service requests. By leveraging Salesforce's powerful platform features such as custom objects, automation tools, and relational data modeling, this system enhances operational efficiency, improves customer interactions, and ensures data consistency across all business processes. The modular object design ensures scalability and easy maintenance, while also aligning well with real-world automotive business workflows.

This project also provided hands-on experience with the Salesforce Developer Org, App Builder functionalities, and demonstrated the power of low-code customization through schema design, tab creation, and relationship management.

Future Scope

- **Automation & Workflows**: Implement approval processes for test drives or service requests, and automate order confirmation using Flow or Process Builder.
- **Reports & Dashboards**: Add interactive dashboards to visualize sales performance, stock levels, and customer trends.
- **Customer Portal**: Create an Experience Cloud portal to allow customers to book test drives, place orders, and raise service requests independently.
- **Integration with External APIs**: Connect the system with payment gateways, Google Maps (for dealer locations), and SMS/email services for notifications.
- **Mobile Compatibility**: Leverage Salesforce Mobile App to allow sales executives and service agents to manage tasks on the go.
- **Einstein AI Integration**: Use Salesforce Einstein for predictive analytics—such as predicting high-demand models or identifying customers likely to return.