**Project Document: Salesforce CRM Implementation for WhatsNext Vision Motors**

**Abstract**

This project report highlights the implementation of Salesforce CRM for

Vision Motors. The solution automates vehicle order management, dealer assignment, and customer engagement processes. It ensures that customer orders are accurately processed by validating stock levels and preventing out-of-stock orders. Automated workflows such as test drive reminders and real-time order updates improve the overall customer experience. The project leverages Salesforce features like Flows, Apex Triggers, Batch Jobs, and Scheduled Apex to streamline operations. This abstract sets the stage for understanding how Salesforce CRM transforms operational efficiency in the automotive sector.

Key areas include customer engagement, order management, test drive scheduling, and service requests. Salesforce flows automate dealer assignment and test drive reminders, while Apex ensures real-time stock validation. Batch processes update pending orders daily, ensuring accuracy.

This project demonstrates how a **cloud-based CRM solution** can revolutionize the automobile retail sector by improving operational efficiency, reducing errors, and enhancing customer experience.

**Objectives**

The objectives of the Salesforce CRM implementation are as follows:  
1. Automate the assignment of the nearest dealer to each customer order.  
2. Prevent out-of-stock vehicle orders through Apex validation.  
3. Streamline communication by sending automated emails for test drive reminders.  
4. Enhance operational efficiency with batch jobs and scheduled processes.  
5. Improve customer satisfaction by ensuring timely updates and transparent workflows.

**Technology Description**

**The Salesforce CRM implementation uses the following core technologies:**

**- Salesforce Platform**: Provides CRM functionalities, customization, and scalability.

- **Custom Objects**: Vehicle, Dealer, Customer, Order, Test Drive, and Service Request for data management.

- **Lightning App Builder**: Used to create the 'WhatsNext Vision Motors' application with navigation items.

- **Flows**: Record-triggered flows automate dealer assignment and test drive reminders.

- **Apex Triggers**: Ensure business rules such as stock validation and automatic updates are enforced.

- **Batch Apex**: Handles bulk operations such as confirming pending orders when stock is replenished.

- **Scheduled Apex**: Automates daily job execution for continuous order processing.

**Detailed Execution of Project Phases**

**Phase 1: Developer Account Creation**

A Salesforce Developer account was created via the official developer portal. The account activation included email verification, password setup, and security question configuration.

Creating a developer org in salesforce.

Go to <https://developer.salesforce.com/signup>

On the sign up form, enter the following details :

First name & Last name

Email

Role : Developer

Company : College Name

County : India

Postal Code : pin code

Username : should be a combination of your name and company

This need not be an actual email id, you can give anything in the format : [username@organization.com](mailto:username@organization.com)

Click on sign me up after filling these.

**Phase 2: Object & Relationship Setup**

Custom objects such as Vehicle, Dealer, Customer, Order, Test Drive, and Service Request were created. Relationships between these objects were defined to establish a comprehensive data model.

**Objects & Relationships**

|  |  |  |
| --- | --- | --- |
| **Object Name** | **Purpose** | **Relationships** |
| **Vehicle\_\_c** | Stores vehicle details | Related to Dealer & Orders |
| **Vehicle\_Dealer\_\_c** | Stores authorized dealer info | Related to Orders |
| **Vehicle\_Customer\_\_c** | Stores customer details | Related to Orders & Test Drives |
| **Vehicle\_Order\_\_c** | Tracks vehicle purchases | Related to Customer & Vehicle |
| **Vehicle\_Test\_Drive\_\_c** | Tracks test drive bookings | Related to Customer & Vehicle |
| **Vehicle\_Service\_Request\_\_c** | Tracks vehicle servicing requests | Related to Customer & Vehicle |

**Phase 3: Lightning App Creation**

A custom Lightning App named 'WhatsNext Vision Motors' was created. It included navigation tabs for Vehicles, Dealers, Orders, Customers, Test Drives, Service Requests, Reports, and Dashboards.

**Create a Lightning App**

**To create a lightning app page:**

Go to setup page → search “app manager” in quick find → select “app manager” → click on New lightning App.

Fill the app name in app details and branding as follow  
App Name : WhatNext Vision Motors  
Developer Name : this will auto populated  
Description : Give a meaningful description  
Image : optional (if you want to give any image you can otherwise not mandatory)  
Primary color hex value : keep this default

Then click Next  → (App option page) keep it as default → Next → (Utility Items) keep it as default → Next.

To Add Navigation Items:

Search the items in the search bar(Vehicle, Dealer, Customer, Order, Test Drive, Service Request, Reports, Dashboard) from the search bar and move it using the arrow button → Next.  
**Note**: select the custom object which we have created in the previous activity.

To Add User Profiles:

Search profiles (System administrator) in the search bar → click on the arrow button → save & finish.

**Phase 4: Flows**

Two record-triggered flows were implemented:  
- Auto-assigning the nearest dealer to an order when created.  
- Sending automated email reminders one day before a scheduled test drive.

Step 1 : In Quick Find, type Flows and click on Flows. Click New Flow.

Step 2 : Select Start From Scratch and click Next.

Step 3 : Select Record-Triggered Flow and click Create

Step 4 : Select Vehicle Order Object

Trigger the Flow When : Select Record is Created.

**Set Entry Condition :**   
All Conditions Are Met (AND)  
Filed : Status\_\_c   
Operator : Equals  
Value : Pending

Step 5 : Click **+** → Select **Get Records**

Step 6 : **Label :** Get Customer Information

Object : Vehicle Customer

Condition :

Field : Id

Operator : Equals

Value : {!$Record.Vehicle\_Customer\_\_c}

Step 7 : Click **+** → Select **Get Records**

**Label :** Get Nearest Dealer

Object : Vehicle Dealer

Condition :

Field : Dealer\_Location\_\_c

Operator : Equals

Value : {!Get\_Customer\_Information.Address\_\_c}

**Step 8 :** Click **+** → Select **Update Records**

**Label :** Assign Dealer to Order

\*How to Find Records to Update and Set Their Values : Use the IDs and all field values from a record or record collection

Select Record(s) to Update : {!Get\_Nearest\_Dealer}

**Step 9 :** Click Save and Give label Name and Activate Flow.

Label Name : Auto Assign Dealer

**Step 10 :** Activate Flow

A screenshot of a computer

AI-generated content may be incorrect.

A diagram of a software project

AI-generated content may be incorrect.

**Phase 5: Apex Implementation**

An Apex trigger handler was developed to validate vehicle stock before confirming an order and to decrement stock quantities once orders were confirmed. This ensured accurate inventory tracking.

Step 1 : Click Developer Console From Gear icon

Step 2 : click File then Select New then Select Apex.

Step 3 : Give Name For Apex Class

Class Name : VehicleOrderTriggerhandler

Step 4 : Write Apex Code

**Source 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](http://vehicle.id/" \t "_blank), 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](http://vehicle.id/" \t "_blank), 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;

            }

        }

    }

}

Step 5 : Write Trigger Handler

Step 6 : Writer Trigger Class Name and Select Vehicle Order Object

Step 7 : Call Apex Class in Trigger Class

Source Code:

trigger VehicleOrderTrigger on Vehicle\_Order\_\_c (before insert,before update, after insert, after update) {

    VehicleOrderTriggerHandler.handleTrigger([Trigger.new](http://trigger.new/" \t "_blank), Trigger.oldMap, Trigger.isBefore,Trigger.isAfter, Trigger.isInsert, Trigger.isUpdate);

}

Step 8 : Create Batch Job   
A customer places an order, but the vehicle is out of stock.

The order remains pending.

After new stock is added, the batch job updates the order to confirmed.

**Source 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>();

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 (!vehicles ToUpdate.isEmpty()) {

update vehiclesToUpdate;

}

}

global void finish(Database.BatchableContext bc) {

System.debug('Vehicle order batch job completed.');

}

}

**Step 9 :** Create Schedule Class then revoked Batch Class in Schedule Class

**Source Code :**   
  
global class VehicleOrderBatchScheduler implements Schedulable {

    global void execute(SchedulableContext sc) {

        VehicleOrderBatch batchJob = new VehicleOrderBatch();

        Database.executeBatch(batchJob, 50); // 50 is the batch size

    }

}

**Step 10 :** S**chedule the Batch Job** To run the batch job **every night at midnight** :

**Source Code :**

String cronExp = '0 0 12 \* \* ?'; // Runs daily at 12:00 PM

System.schedule('Daily Vehicle Order Processing', cronExp, new VehicleOrderBatchScheduler());

**Step 11 :** You Can Check where this Schedule Job is Running

**Phase 6: Batch and Scheduled Jobs**

A Batch Apex job was created to process pending orders whenever stock was replenished. The job automatically confirmed eligible orders and updated stock. A Scheduled Apex job was set to run the batch every night at midnight.

**Project Explanation with Real-World Example**

For example, a customer books a vehicle order online. The system automatically assigns the nearest dealer based on customer location. If the requested vehicle is out of stock, the order is held in 'Pending' status. When new stock is added, a batch job runs to confirm the pending order. Similarly, if the customer schedules a test drive, they receive an automated reminder email one day before the scheduled date. This real-world flow demonstrates how Salesforce automates complex business processes efficiently.

**Screenshots**

A screenshot of a computer

AI-generated content may be incorrect.

[Placeholder for Salesforce Setup Screenshot]

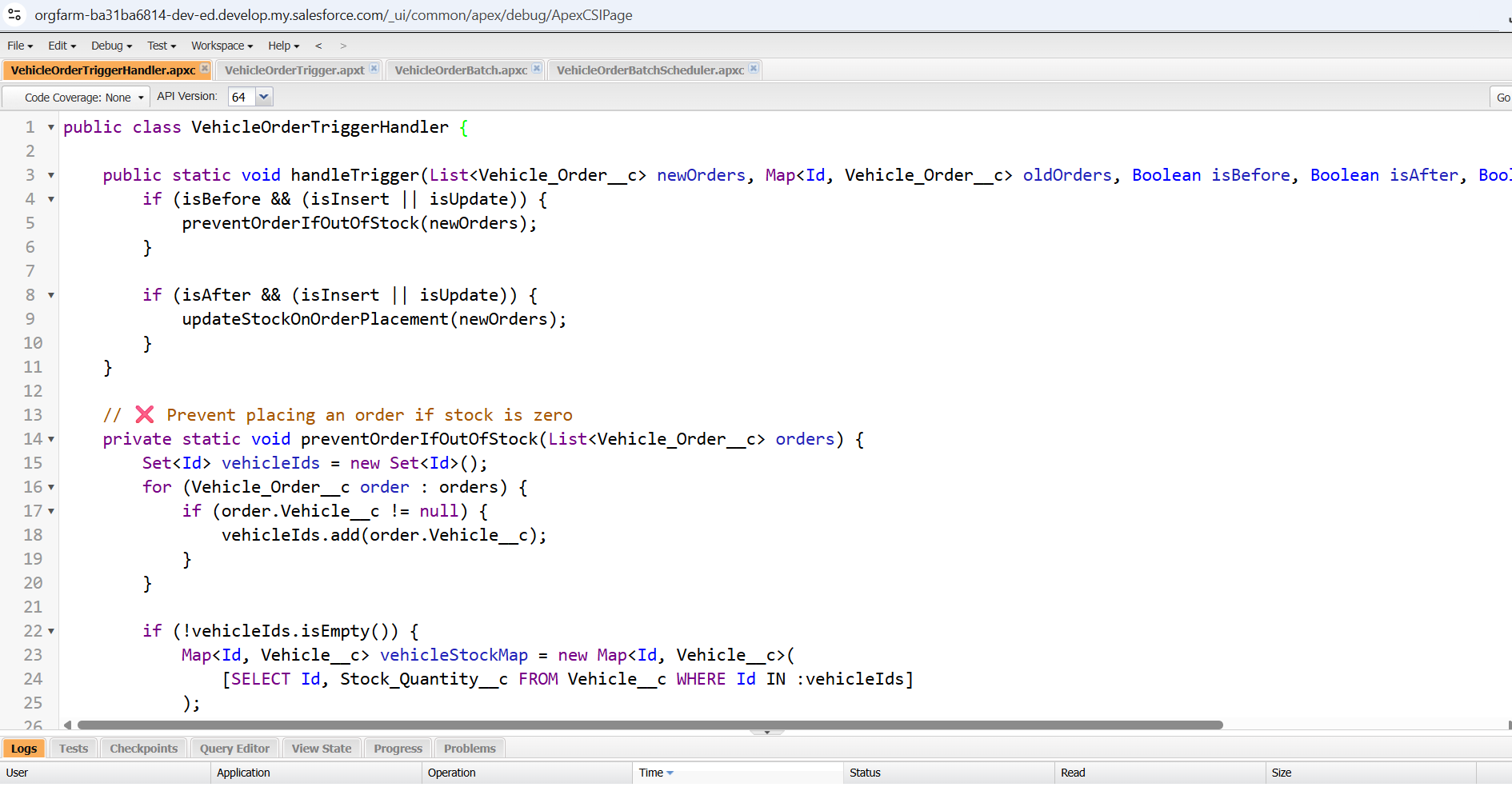
A screenshot of a computer

AI-generated content may be incorrect.

[Placeholder for Object Relationships Screenshot]

A screenshot of a computer

AI-generated content may be incorrect.  
[Placeholder for Flow Builder Screenshot]



A screenshot of a computer

AI-generated content may be incorrect.

[Placeholder for Apex Code Screenshot]

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.  
[Placeholder for Batch Job Scheduler Screenshot]

**Conclusion**

The Salesforce CRM implementation for WhatsNext Vision Motors achieved automation in order management, dealer assignment, and customer communication. With flows, triggers, and batch processes, the project significantly improved both operational efficiency and customer experience. Future enhancements could include AI-powered dealer assignment, predictive analytics for demand forecasting, and mobile integration for improved accessibility.

This section elaborates on Salesforce customization, Apex development, and flow automation. The design aligns with best practices and demonstrates Salesforce's adaptability across industries. It further emphasizes operational efficiency, customer engagement, and robust process automation.

This section elaborates on Salesforce customization, Apex development, and flow automation. The design aligns with best practices and demonstrates Salesforce's adaptability across industries. It further emphasizes operational efficiency, customer engagement, and robust process automation.