**Project Title: WhatNext Vision Motors – Shaping the Future of Mobility with Innovation and Excellence**

**Submitted in partial fulfillment of the SmartInternz Virtual Internship Program (July 2025)**  
**In association with Salesforce Platform**

**Project Role**

Salesforce Developer | Apex Programmer | LWC Developer | Project Lead

**Submitted by**

**Deepan Chakravarthi M**  
B.E. Computer Science Engineering  
, Tamil Nadu, India

**Submission Date**

**July 2025**

**Guided by**

SmartInternz Mentors and Faculty Coordinator

**Modules Covered**

1. Use Case
2. User Story
3. Salesforce Credentials Setup
4. Data Management – Objects
5. Data Management – Tabs
6. Data Management – App Manager
7. Data Management – Fields
8. Data Configuration
9. Data Security – Profiles
10. Data Security – Roles
11. Data Security – Users
12. Data Security – Permission Sets
13. Email Templates & Alerts
14. Flows (Automation)
15. Apex Triggers
16. Batch Jobs

**Introduction**

**WhatsNext Vision Motors** is a pioneering force in the automotive industry, renowned for its commitment to innovation and customer-centric mobility solutions. In alignment with its vision to reshape the future of transportation, the company has initiated a comprehensive Salesforce CRM project aimed at digitally transforming its customer interaction and operational workflows.

This project focuses on enhancing the **customer ordering experience** by automating key business processes such as dealer assignment, stock validation, and order status management. At the core of the implementation is a smart feature that **automatically suggests the nearest dealership** to customers based on their address, thereby reducing effort, improving order turnaround time, and significantly elevating customer convenience.

In addition, the project addresses one of the major challenges in vehicle sales — **real-time stock availability**. Customers are now prevented from placing orders for vehicles that are out of stock. This ensures **accurate inventory-based ordering**, reducing customer frustration and improving the integrity of the company's fulfillment operations.

A significant innovation within this project is the **automated bulk order status update mechanism**. Scheduled to run periodically, this process dynamically updates order statuses based on real-time inventory data — marking orders as *Confirmed* if in stock, or *Pending* if unavailable. This provides **clear communication to customers** and helps the company maintain transparency and trust.

The integration of these intelligent automation features into WhatsNext Vision Motors’ CRM infrastructure is expected to:

* **Enhance customer satisfaction** through a seamless ordering process
* **Improve operational efficiency** by reducing manual workload
* **Increase accuracy and transparency** in order fulfillment
* **Support strategic growth** by freeing up employee resources for higher-value tasks

By leveraging Salesforce's robust capabilities, WhatsNext Vision Motors is not only modernizing its systems but also reinforcing its position as a **leader in customer-focused innovation** in the mobility sector.

**Requirements & Learning Outcomes**

**Project Requirements**

The Salesforce CRM implementation for **WhatsNext Vision Motors** is designed to address the operational and customer service challenges faced in the automotive industry. The project is structured around four key requirement categories:

**Salesforce CRM Implementation**

* Centralized platform to **store and manage vehicle details**, stock levels, and dealer location data.
* Track **customer orders**, **test drive bookings**, and **service requests** efficiently.
* **Automate order assignment** based on customer location to the nearest available dealer.

**Process Automation**

* **Restrict orders** for vehicles that are out of stock to maintain fulfillment accuracy.
* Automatically assign test drives and orders to the appropriate **dealer based on geography**.
* **Trigger automated emails** to remind customers of their scheduled test drives.

**Apex and Triggers**

* Use **Apex triggers** to apply real-time business logic such as:
* Stock validation before order confirmation
* Automatic dealer lookup on order creation
* Implement **Trigger Handlers** to maintain clean, modular, and scalable code architecture.

**Batch Jobs**

* Create a **Batch Apex job** to:
  + Monitor and update vehicle availability status periodically.
  + Send scheduled email alerts for low stock and pending order processing.

**What You'll Learn**

By completing this project, the following Salesforce development concepts and skills will be reinforced:

* **Data Modeling** – Designing custom objects and relationships
* **Fields & Relationships** – Lookup, formula, picklist, and validation fields
* **Lightning App Builder** – Creating user-friendly Lightning apps and tabs
* **Record-Triggered Flows** – Automating processes without code
* **Apex & Triggers** – Writing custom logic for real-time actions
* **Batch Apex** – Handling large data volumes asynchronously
* **Scheduled Apex** – Automating time-based jobs and system tasks

**Developer Account Creation**

A free Salesforce Developer Account was created using the official Salesforce portal:  
 <https://developer.salesforce.com>

This account provides:

* Access to the Salesforce Lightning Platform
* Development tools and sandbox environment
* Ability to create custom apps, objects, and automations

**Account Verification**

After registration, a verification email was received and confirmed to activate the account.

Verification allowed:

* Full access to the Lightning Experience UI
* Configuration of password, recovery options, and login IP ranges

**Development Tool Used: Developer Console**

Instead of using Visual Studio Code or Salesforce CLI, **Developer Console** was used for all development activities.

Tasks completed using Developer Console:

* Writing Apex classes and triggers
* Executing anonymous blocks
* Debug logs and monitoring
* Testing triggers and batch classes
* Creating SOQL queries and reviewing logs

**Objects & Relationships**

In the Salesforce CRM implementation for **WhatsNext Vision Motors**, a structured data model was established to support vehicle management, customer tracking, and process automation. The following custom objects were created to handle key business functions and maintain relational integrity.

**Custom Object Overview**

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

**Creating Custom Tabs**

In the **WhatsNext Vision Motors Salesforce CRM**, custom tabs were created for each key object to ensure easy access, intuitive navigation, and efficient data management for users such as sales representatives, service agents, and administrators.

Custom tabs allow users to interact with records from custom objects directly through the Salesforce interface.

**How to Create a Custom Tab (Example: Vehicle)**

**Steps:**

1. Go to **Setup**
2. In the **Quick Find** bar, type **Tabs**
3. Click on **Tabs** under the User Interface section
4. Under **Custom Object Tabs**, click **New**
5. Select the object: **Vehicle**
6. Choose a **Tab Style** (e.g., Car icon)
7. Click **Next** on the "Add to Profiles" page (keep default access)
8. Click **Next** on the "Add to Custom Apps" page (keep default)
9. Click **Save**

**Tabs Created for Custom Objects**

| **Object Name** | **Tab Name** | **Purpose** |
| --- | --- | --- |
| Vehicle\_\_c | Vehicle | Manage vehicle models, variants, and pricing |
| Vehicle\_Dealer\_\_c | Dealer | Maintain dealer location and contact details |
| Vehicle\_Customer\_\_c | Customer | View and manage customer information |
| Vehicle\_Order\_\_c | Order | Track vehicle orders and delivery status |
| Vehicle\_Test\_Drive\_\_c | Test Drive | Schedule and manage test drive appointments |
| Vehicle\_Service\_Request\_\_c | Service Request | Handle post-sale service and maintenance requests |

**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.

**Creating a Field in Vehicle Object**

In the WhatsNext Vision Motors Salesforce CRM, custom fields are added to objects to store relevant business data. Below are the steps to create a custom **Picklist field** named *Vehicle Model* in the **Vehicle\_\_c** object.

**Steps to Create a Custom Field**

1. Go to **Setup**
2. Click on **Object Manager** from the top menu
3. In the **Quick Find** bar, type and select **Vehicle**
4. Click on the object name: **Vehicle\_\_c**

**Add a New Field**

1. Inside the Vehicle object setup page, click on **Fields & Relationships**
2. Click on the **New** button to create a new field
3. Choose **Picklist** as the data type
4. Click **Next**

**Configure Field Details**

* **Field Label**: Vehicle Model
* **Values** (Enter manually):
  + Model X
  + Model Y
  + Model Z
  + EV Prime
  + AutoPro

1. After entering the picklist values, click **Next**
2. On the profile visibility page, click **Next** to keep default access
3. On the page layout assignment, click **Next**
4. Click **Save & New** if you want to add another field, or just **Save** to finish

**Purpose of this Field**

The **Vehicle Model** picklist allows users to select from predefined vehicle types when entering a new vehicle record. This ensures:

* Standardized data entry
* Easy filtering and reporting
* Better user experience when managing vehicle inventory

This process is repeated to add other relevant fields like Variant, Fuel Type, and Transmission Type to fully define the vehicle profile.  
  
  
**Creating a Lookup Relationship in Vehicle Object**

To enable the **Vehicle\_\_c** object to reference related dealer records, a **Lookup Relationship** is created. This relationship helps in associating each vehicle with a specific authorized dealer and facilitates better reporting and automation.

**Steps to Create a Lookup Relationship Field**

1. Go to **Setup**
2. Navigate to **Object Manager**
3. In the Quick Find bar, type and select **Vehicle**

**Add New Relationship Field**

1. Click on **Fields & Relationships** under the Vehicle object
2. Click **New** to create a new custom field
3. Select **Lookup Relationship** as the data type
4. Click **Next**

**Configure Relationship**

1. On the related object selection screen, choose **Dealer\_\_c** (or Vehicle\_Dealer\_\_c if using a custom name)
2. Click **Next**

**Complete Field Setup**

1. Accept the default field label or rename it (e.g., **Associated Dealer**)
2. Click **Next** to set field-level security (keep default for all profiles)
3. Click **Next** to add the field to page layouts
4. Click **Save**

**Purpose of This Relationship**

Creating a lookup from **Vehicle\_\_c** to **Dealer\_\_c** allows the following:

* Link every vehicle to its corresponding dealer
* Use this relationship in automation (Flows, Triggers)
* Filter vehicle availability by dealer location
* Generate region-based reports for sales and inventory

This relational setup enhances data visibility and enforces consistency between vehicle records and dealership data.

**Creating a Record-Triggered Flow to Assign Nearest Dealer**

In the WhatsNext Vision Motors CRM, a record-triggered flow is used to automatically assign the nearest dealer based on the customer's address at the time of vehicle order creation. This automation improves operational efficiency and ensures accurate dealer-customer assignment.

**Steps to Create the Flow**

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

**Step 2:**  
Choose **Start from Scratch**, then click **Next**.

**Step 3:**  
Select **Record-Triggered Flow** and click **Create**.

**Step 4:**  
Configure the trigger:

* **Object**: Vehicle\_Order\_\_c
* **Trigger the Flow When**: A record is created
* **Entry Conditions**:
  + All Conditions Are Met (AND)
  + Field: Status\_\_c
  + Operator: Equals
  + Value: Pending

**Add Flow Elements**

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

* **Label**: Get Customer Information
* **Object**: Vehicle\_Customer\_\_c
* **Condition**:
  + Field: Id
  + Operator: Equals
  + Value: {!$Record.Vehicle\_Customer\_\_c}

**Step 6:**  
Click **+** → Select **Get Records**

* **Label**: Get Nearest Dealer
* **Object**: Vehicle\_Dealer\_\_c
* **Condition**:
  + Field: Dealer\_Location\_\_c
  + Operator: Equals
  + Value: {!Get\_Customer\_Information.Address\_\_c}

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

* **Label**: Assign Dealer to Order
* Use IDs and all field values from a record
* Select record(s) to update: {!Get\_Nearest\_Dealer}

**Finalizing the Flow**

**Step 8:**  
Click **Save**, name the flow: Auto Assign Dealer

**Step 9:**  
Click **Activate** to enable the flow

**Creating a Record-Triggered Flow to Send Test Drive Reminder Email**

To enhance customer engagement and reduce no-shows, WhatsNext Vision Motors uses a **Record-Triggered Flow** to automatically send a reminder email one day before the customer's scheduled test drive.

**Steps to Build the Flow**

**Step 1:**  
Create a **Record-Triggered Flow**

* Object: Vehicle\_Test\_Drive\_\_c
* Trigger the Flow When: A record is **created or updated**

**Step 2:**  
Set Entry Conditions:

* All Conditions Are Met (AND)
  + Field: Status\_\_c
  + Operator: Equals
  + Value: Scheduled

**Step 3: Add Scheduled Path**

* Click **+ Add Scheduled Paths** below the trigger
* **Label**: Reminder Before Test Drive
* **Time Source**: Test\_Drive\_Date\_\_c
* **Offset**: 1 Day Before
* Click **Done**

**Step 4: Get Customer Information**

* Click **+** → Select **Get Records**
* **Label**: Get Customer Information
* **Object**: Vehicle\_Customer\_\_c
* **Condition**: Id = {!$Record.Customer\_\_c}
* How Many Records to Store: Only the first record
* Store Record Data: Automatically store all fields

**Step 5: Send Email Reminder**

* Click **+** → Select **Action**
* **Action Type**: Send Email
* **Label**: Send Test Drive Reminder
* **Subject**: "Reminder: Your Test Drive is Tomorrow!"
* **Recipient Address**: {!Get\_Customer\_Information.Email\_\_c}
* **Rich Text Body**: Enabled
* **Body Variable**: Create Text Template variable
* API Name: EmailSent

**Step 6: Save and Activate**

* Save Flow
* **Label**: Test Drive Reminder
* **Activate** the flow

**Create Apex and Trigger Batch Jobs :**

**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

**Conclusion**

The Salesforce CRM implementation for **WhatsNext Vision Motors** marks a significant stride towards digital transformation in the automotive industry. Through this project, the company successfully integrated modern cloud-based CRM capabilities to streamline customer interactions, automate key business processes, and enhance operational transparency.

By leveraging tools such as custom objects, Lightning App Builder, record-triggered flows, Apex triggers, and batch classes, the project addressed core business requirements—ensuring real-time stock validation, efficient dealer assignment, timely communication through automated emails, and seamless management of test drives and service requests.

The implementation delivered the following tangible benefits:

* **Improved Customer Experience**: Intelligent automation such as nearest dealer assignment and test drive reminders reduced customer effort and improved satisfaction.
* **Operational Efficiency**: Automation of repetitive tasks, such as order validation and status updates, freed up staff to focus on strategic responsibilities.
* **Data Accuracy and Visibility**: Structured data models and well-defined relationships ensured better reporting, tracking, and decision-making capabilities.
* **Scalability and Maintainability**: Use of modular Apex design and declarative tools enables future growth and minimal maintenance overhead.

This project demonstrates how a well-planned Salesforce CRM solution can drive innovation, deliver value to customers, and support long-term business goals. The hands-on experience with Salesforce development, flow automation, and real-world business logic has equipped the project team with essential skills for enterprise-grade cloud CRM development.

**WhatsNext Vision Motors** now stands well-positioned to scale its digital initiatives, strengthen dealer and customer engagement, and continue shaping the future of mobility through technology-driven excellence.