

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

-Tanvi Shetty

Project Overview:

The Salesforce CRM project at WhatsNext Vision Motors is a strategic CRM project whose purpose is to transform the customer ordering experience and optimize automobile industry operations. The purpose of this CRM solution is to simplify the vehicle ordering procedure and implement such intelligent features as auto-suggestions of the location of dealers in case it is customer-provided address, or the direct real-time updating of current stock availability. It avoids taking orders of cars that are not in stock, and this enhances service satisfaction and customer order accuracy. Moreover, it has an automated scheduler that ensures an update of bulk order statuses (status to be changed to either of the following: Confirmed or Pending) on the basis of available stocks. These aspects respond to some crucial business requirements, including a decrease in manual work, enhancing transparency, and optimizing experience improvement serving the customer, as the company vision is to support customers with innovational and customer-oriented mobility solutions.

Key Features:

1. Automated Dealer Search:

The CRM also makes a clever course of action to search the nearest dealer place through the address of the customer to increase convenience and minimize the manual work in finding a dealer.

2. Real Time Stock Availability Check-up:

Customers will be able only to order those cars which are already in the inventory, so it will help to exclude the chance that the order might not be kept and it will increase the consistency of ordering.

3. Order Placement Control:

It does not allow the events at system level to create the orders on cars that are not in the stocks and only valid and bankable orders are released.

4. Bulk Order Status Scheduler:

An automated scheduling system is in place and always keeps updated about the status of bulk orders, putting them on Pending or Confirmed condition depending on real-time stocks available, thereby providing transparency and quick notifications.

5. Operational Automation:

Such processes as status updates and order validation are automated, which decreases their number of workers and contributes to efficiency.

Business Needs:

1. Improved Customer Experience:

The CRM also builds customer confidence and satisfaction through an easy ordering process and simply giving the customer real-time information about the status of available inventory.

2. Better Accuracy in Orders:

Restricting the listing of unavailable cars avoids the situation that would require an accurate and efficient order fulfilment process.

3. Operational Efficiency:

Through automation of repetitive work, administrative overhead is minimized and the personnel has more time to concentrate in high-value strategic work.

4. Honesty and Communication:

Correct order statuses enhance contact between customers and the company making clear expectations about the completion of orders.

5. Scalability and Agility:

The CRM also allows the firm to expand its operations without any disruptions and within time adjusts to the shifting market and customer realities.

Objectives:

The main purpose of developing the CRM of WhatsNext Vision Motors could be the improvement of customer satisfaction and streamline the car ordering and delivery. The CRM is designed to deliver a smooth and transparent customer experience by automating dealer recommendations, real-time checking of stock availability and orders status. This directly helps in the management of the customer relations through decreasing errors, thwarting orders delays and making communication smooth in the purchasing process. Moreover, automatization of the major administrative processes allows the company to be more efficient, reduces the weight of manual work and devotes resources to strategic growth programs. All in all, the CRM aligns with the vision of bestowing a contemporary, client-focused, customizable automotive service environment of the company.

Phase 1: Requirement Analysis & Planning

Learning About Business Requirements

The main business requirement was to automate, centralize customer, vehicle, store, and booking management software of WhatsNext Vision Motors. The problems that faced the manual or the fragmented systems included:

- Late response in the confirmation of availability of vehicles.
- Failure to follow-up prospective customers.

- Poor process of allocating the test drives to dealers in the local areas.
- There was no centralized way to monitor sales orders, stocks and customer feedback.

Identification of User's Needs:

- A central customer, vehicle and order management CRM.
- Automatic distribution of test drives on the closest available dealer.
- Timely bequest of vehicle stock positions (on hand inventory).
- Good customer communication through emails/SMS Notification.
- Sales and dealership-accessible dashboards that can be accessed via mobile or web.

Project Scope and Project Objectives

Scope:

- The use of custom CRM on Salesforce using custom objects.
- Integration of backend automation with real time updates and alerts.
- Incorporation with third party services in the future (SMS, WhatsApp).
- Responsive interface designs of the customers and staff.

Core Objectives:

1. Enhance Customer Experience
 - Quick and customized answer to questions.
 - Simple test drive booking and follow up and tracking of orders.
2. Streamline Operations:
 - Automatic validation of the stocks placed when orders are submitted.
 - To minimize human error and manual data entry.
3. Facilitate Evidence-Based Results:
 - Dealer performance dashboards, conversion rates of test drives and so on.
4. Ensure Scalability:
 - The design of CRM should be able to cover additional cities, dealers, and types of vehicles in future.
5. Role-based and Secure Access:
 - Only the dealers have the capability to see/maintain their own leads.
 - Admins can see and command everything.
6. Salesforce-native Integration:
 - Maintain low-code through standard use of Salesforce capabilities (Flows, Apex, Validation Rules).

Design of Data Model and Security Model

Data Model (Objects & Relationships)

Object Name	Type	Description	Relationships
Vehicle_Customer__c	Custom Object	Stores customer profile, contact, and address details	Related to Orders & Test Drives
Vehicle__c	Custom Object	Contains vehicle model, price, stock, and specifications	Related to Dealer & Orders
Vehicle_Order__c	Custom Object	Captures order info like selected vehicle, date, status	Related to Customer & Vehicle
Vehicle_Test_Drive__c	Custom Object	Schedules test drive slot, confirms availability	Lookup to Dealer, Customer
Vehicle_Service_Request__c	Custom Object	Tracks vehicle servicing requests	Related to Customer & Vehicle
Vehicle_Dealer__c	Custom Object	Stores dealer name, location, contact, vehicle coverage info	Related to Customer & Vehicle

Security Model

Level	Implementation
Organization-Wide Defaults (OWD)	Private for custom objects to protect data visibility
Role Hierarchy	Admin > Regional Manager > Dealer Executive
Sharing Rules	Manual sharing for records if needed
Profiles and Permission Sets	Fine-grained control over object access (Read, Edit, Create, Delete)

Phase 2: Salesforce Development – Backend & Configurations

Setup Environment & DevOps Workflow

To ensure a smooth and collaborative development process, we established the following setup and deployment pipeline:

- Salesforce Developer Edition org was used for initial customizations and prototyping.
- Scratch orgs and Sandbox environments were utilized for isolated development and testing.
- SFDX CLI (Salesforce DX) was configured to manage source code and metadata.
- Version Control: Git was used for source tracking and collaboration among team members.
- Deployment Workflow:
 - Source code → Scratch Org (Dev) → Full Sandbox (QA) → Production
 - Metadata deployed using Change Sets and SFDX Commands.
 - Ensured rollback plan during deployment for fail-safes.

Customization of Objects, Fields, Validation Rules & Automation

Custom Objects Created:

- Vehicle_Customer__c
- Vehicle__c
- Vehicle_Dealer__c
- Vehicle_Order__c
- Vehicle_Test_Drive__c
- Vehicle_Service_Request__c

Each object was designed with necessary fields (lookup, picklist, checkbox, number, etc.) and Record Types to distinguish between test drives and bookings when needed.

<div> <div> <div></div> <div>Setup</div> </div> <div> <div>Home</div> <div>Object Manager</div> </div> </div>						
<div> <div> <div>SETUP</div> <div>Object Manager</div> <div>6 Items, Sorted by Label</div> </div> <div> <div>Q vehicle</div> <div>Schema Builder</div> <div>Create</div> </div> </div>						
LABEL	API NAME	TYPE	DESCRIPTION	LAST MODIFIED	DEPLOYED	
Vehicle	Vehicle__c	Custom Object	Stores vehicle details	7/23/2025	✓	▼
Vehicle Customer	Vehicle_Customer__c	Custom Object	Stores customer details	7/23/2025	✓	▼
Vehicle Dealer	Vehicle_Dealer__c	Custom Object	Stores authorized dealer info	7/23/2025	✓	▼
Vehicle Order	Vehicle_Order__c	Custom Object	Tracks vehicle purchases	7/23/2025	✓	▼
Vehicle Service Request	Vehicle_Service_Request__c	Custom Object	Tracks vehicle servicing requests	7/23/2025	✓	▼
Vehicle Test Drive	Vehicle_Test_Drive__c	Custom Object	Tracks test drive bookings	7/23/2025	✓	▼

Figure 1: Data Management Objects

Custom Fields Under Each Object:

1. Vehicle__c (Custom Object):

Vehicle_Name__c , Vehicle_Model__c ,Stock_Quantity__c , Price__c (Currency) , Dealer__c , Status__c

2.Vehicle_Dealer__c (Custom Object):

Dealer_Name__c ,Dealer_Location__c ,Dealer_Code__c ,Phone__c ,Email__c

3. Vehicle_Order__c (Custom Object):

Customer__c ,Vehicle__c ,Order_Date__c ,Status__c

4. Vehicle_Customer__c (Custom Object):

Customer_Name__c ,Email__c ,Phone__c ,Address__c ,Preferred_Vehicle_Type__c

5. Vehicle_Test_Drive__c (Custom Object):

Customer__c ,Vehicle__c ,Test_Drive_Date__c ,Status__c

6. Vehicle_Service_Request__c (Custom Object):

Customer__c ,Vehicle__c ,Service_Date__c ,Issue_Description__c ,Status__c

Validation Rules:

1. Prevent Order if Vehicle is Out of Stock

- Purpose: Prevents users from placing an order for a vehicle that has no stock left.
- Validation Rule:
if(Stock_Quantity__c <= 0, throw error)
- Function Name: preventOrderIfOutOfStock(List<Vehicle_Order__c> orders)
- Implemented In:
Trigger — before insert and before update on Vehicle_Order__c
- Logic:
Uses addError() method to display a custom error and stop the DML operation if the selected vehicle's Stock_Quantity__c is 0 or less.

2. Auto-Decrement Stock on Confirmed Order

- Purpose: Ensures that whenever an order is confirmed, the stock count of the selected vehicle is reduced by 1.
- Business Logic:
if(Status__c == 'Confirmed', Stock_Quantity__c -= 1)
- Function Name: updateStockOnOrderPlacement(List<Vehicle_Order__c> orders)
- Implemented In:
Trigger — after insert and after update on Vehicle_Order__c
- Logic:
Fetches the related vehicle, checks status, and updates the stock accordingly using DML.

3. Auto-Confirm Pending Orders If Stock Available (Batch Automation)

- Purpose: Automatically confirms orders that are still pending if the vehicle has stock.
- Business Logic:
if(Status__c == 'Pending' AND Stock_Quantity__c > 0, Status__c = 'Confirmed'; Stock - = 1)
- Function Name: execute(SchedulableContext sc) inside VehicleOrderBatch class
- Implemented In:
Batch Apex — Scheduled Job
- Logic:
Collects all pending orders in bulk and updates them in batches if the selected vehicles have available stock. Ensures system performance and data integrity.

Automation Implemented:

Type	Use Case
Workflow Rule	Send confirmation email upon Test Drive booking
Flows (Record-Triggered)	Auto-assign nearest dealer based on customer's city

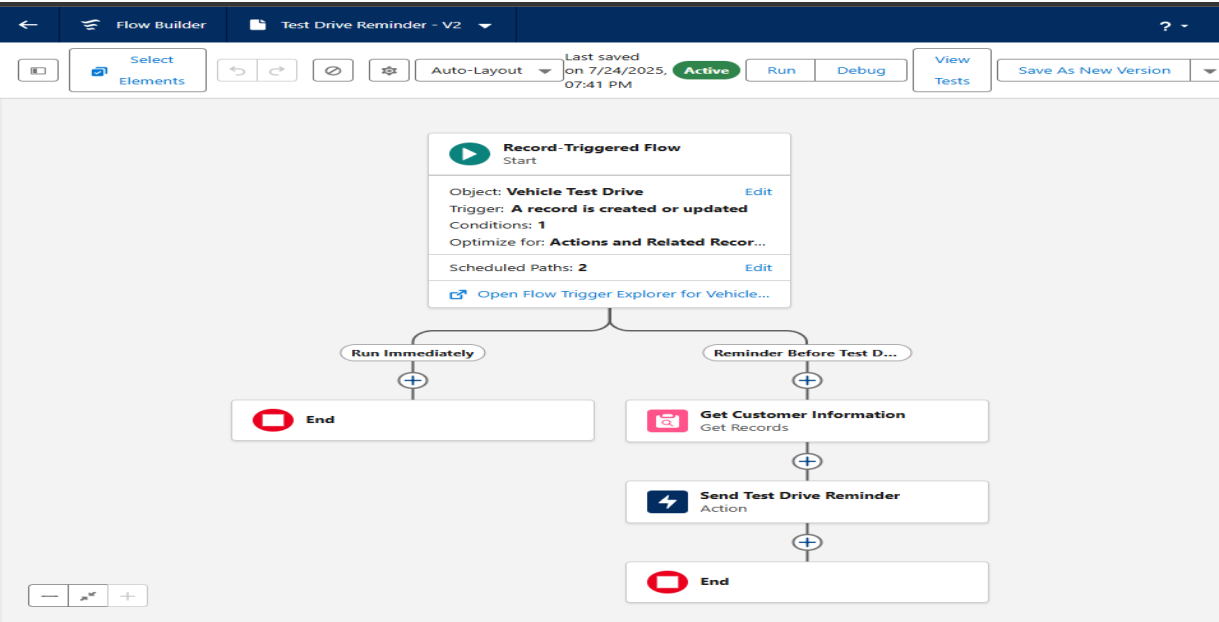


Figure 2: Test Drive Reminder

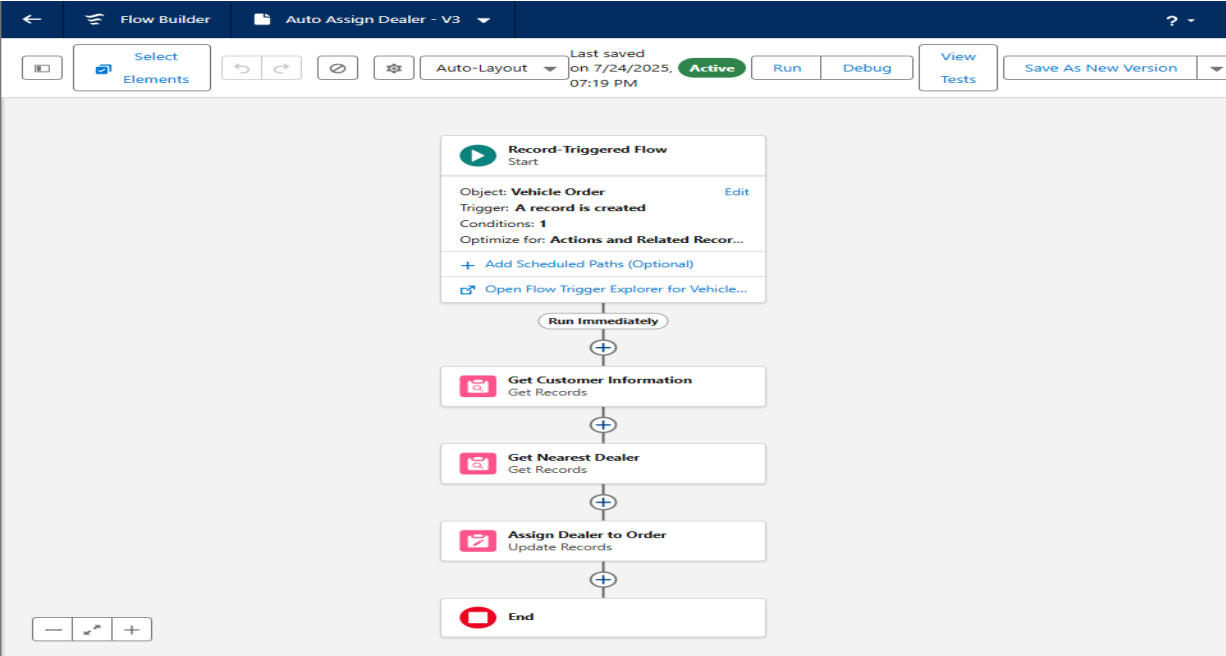


Figure 3: Auto Assign Dealer

Apex Classes, Triggers, Asynchronous Apex

VehicleOrderTriggerHandler Class :

This Apex class manages vehicle order validation and inventory updates in a Salesforce application. It is triggered during insert and update operations on Vehicle_Order__c records.

Rule 1: Prevent Order if Vehicle is Out of Stock

- Function: preventOrderIfOutOfStock(...)
- Trigger Context: before insert/update
- Purpose: Stops order placement if the vehicle's Stock_Quantity__c is zero or less.
- Logic: Checks stock and throws an error using order.addError() if out of stock.

Rule 2: Update Stock on Order Confirmation

- Function: updateStockOnOrderPlacement(...)
- Trigger Context: after insert/update
- Purpose: Reduces the stock count of the vehicle by 1 when an order is Confirmed.
- Logic: Fetches stock, decrements it, and updates the Vehicle__c record.

This handler ensures that:

- No orders are placed for unavailable vehicles.
- Stock is automatically managed upon confirmed orders.

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 && (isInsert || isUpdate)) {

            preventOrderIfOutOfStock(newOrders);

        }

        if (isAfter && (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>(
            [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.');
            }
        }
    }
}

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;
}
}
}
}
}

```

Batch Class: VehicleOrderBatch

This batch class automates the processing of pending vehicle orders.

- **start():** Fetches all Vehicle_Order__c records where Status__c is 'Pending'.
- **execute():** For each order:
 - Checks if the related vehicle is in stock.
 - If available, updates the order status to 'Confirmed' and decreases vehicle stock by 1.
- **finish():** Logs a message after batch completion.

Purpose: Ensures vehicle orders are confirmed only if stock is available, and updates inventory accordingly.

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>(
            [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) {
            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.');
```

Scheduler Class: VehicleOrderBatchScheduler

This Apex scheduler automates the periodic execution of the VehicleOrderBatch job.

- **execute()**: Instantiates the VehicleOrderBatch class and starts the batch job with a batch size of 50.

Purpose: Enables scheduled processing of pending vehicle orders (e.g., daily, weekly), ensuring timely order confirmation and stock updates without manual intervention.

Code:

```

global class VehicleOrderBatchScheduler implements Schedulable {

    global void execute(SchedulableContext sc) {

        VehicleOrderBatch batchJob = new VehicleOrderBatch();

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

VehicleOrderTrigger:

This Apex trigger runs on Vehicle_Order__c for:

- before insert, before update, after insert, after update

It calls `VehicleOrderTriggerHandler.handleTrigger()` to:

- Prevent orders if vehicle is out of stock
- Reduce stock on confirmed orders

Code:

```
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);
```

```
}
```

Resulting Benefits

- Reduced manual errors through validation and automation.
- Faster customer response due to auto-dealer assignment and emails.
- Real-time stock tracking and alerts for out-of-stock situations.
- Improved user experience by enforcing logical and error-free workflows.
- Scalability ensured by designing loosely coupled Apex classes and reusable flows.

Phase 3: UI/UX Development & Customization

Enhancing the user interface and experience in Salesforce was one of the phases that aimed at making sure that it is easy to use, to navigate and understand it easily by various types of users such as sales executives and dealer personnel.

Setting up Lightning App through App Manager

WhatsNext Vision CRM is a custom Lightning App developed on the App Manager. The major tabs that are present in this app are:

- Vehicles
- Vehicle Orders
- Vehicle Customers
- Vehicle Dealers
- Vehicle Service Request
- Vehicle Test Drive

- Dashboards
- Reports

This centralization facilitates day to day activities of the sales and dealer users.

Page Layouts

A unique page design was being used to each custom object:

Vehicle:

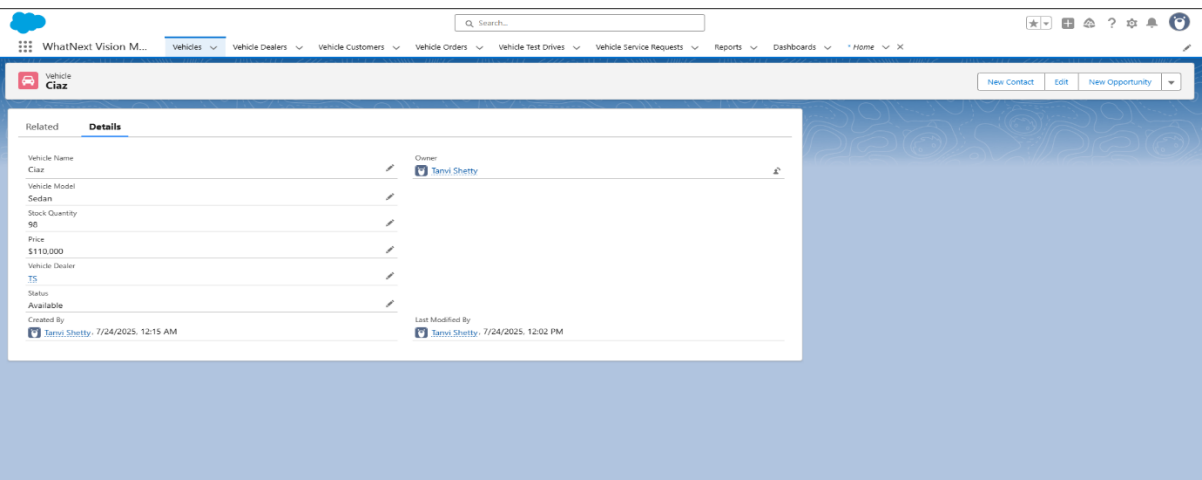


Figure :Vehicle layout with fields such as Vehicle Name, Model, Price, Availability etc.

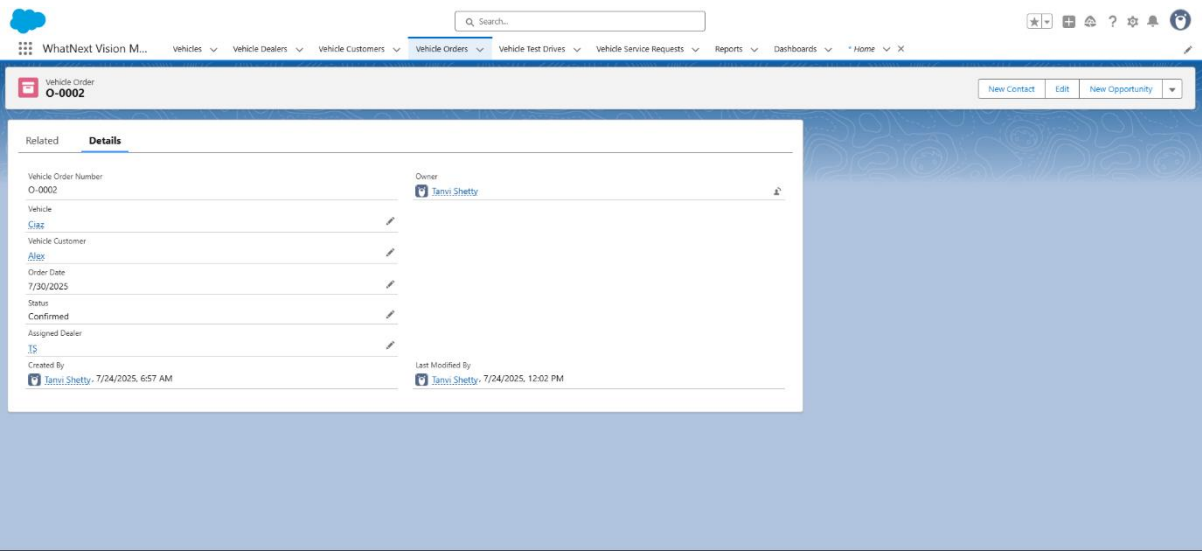
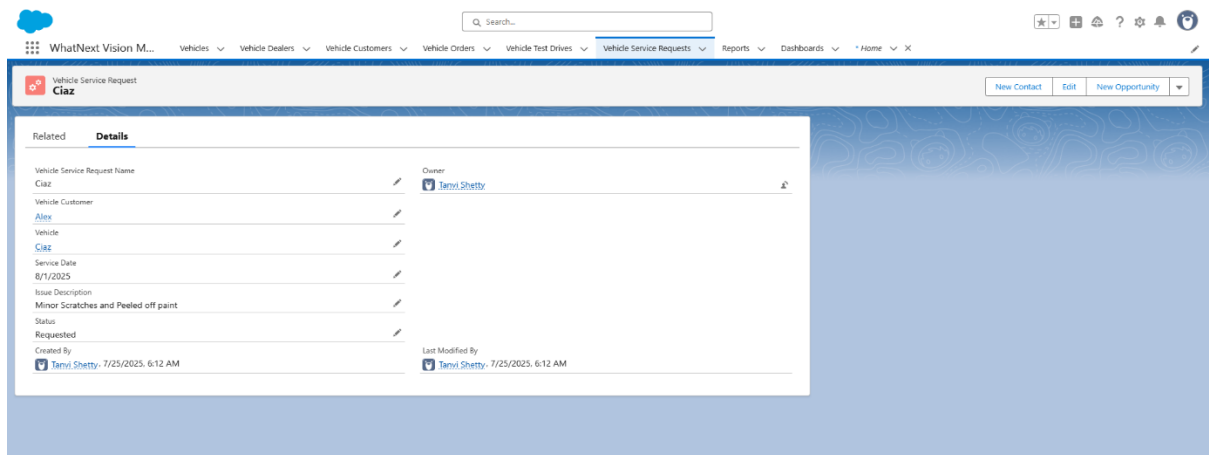


Figure :Vehicle Orders layout with fields such as Vehicle Order Number , Customer, Model, Order Date, Status etc.



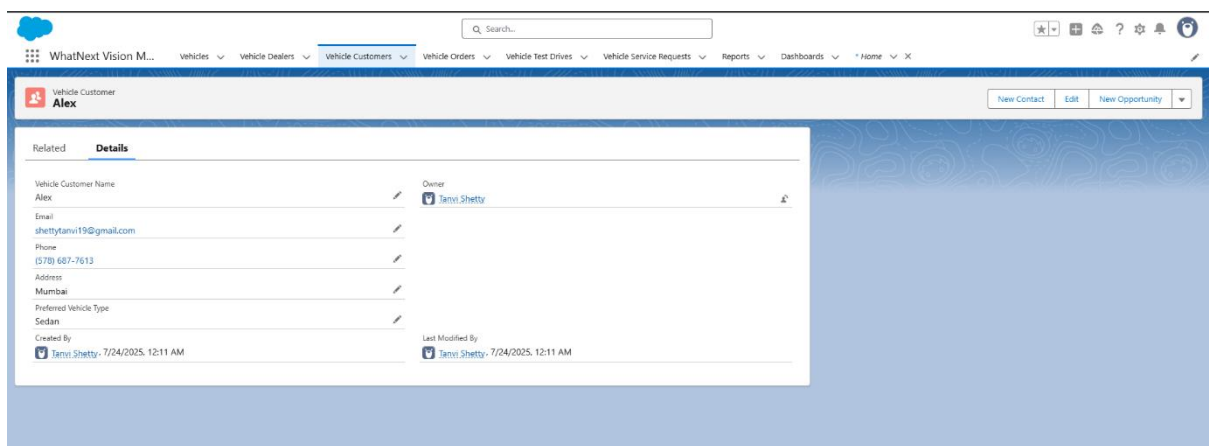
WhatNext Vision M... Vehicles Vehicle Dealers Vehicle Customers Vehicle Orders Vehicle Test Drives Vehicle Service Requests Reports Dashboards Home X

Vehicle Service Request **Ciaz** New Contact Edit New Opportunity

Related Details

Vehicle Service Request Name	Ciaz	Owner	Tami Shetty
Vehicle Customer	Alex		
Vehicle	Ciaz		
Service Date	8/1/2025		
Issue Description	Minor Scratches and Peeled off paint		
Status			
Requested			
Created By	Tami Shetty - 7/25/2025, 6:12 AM	Last Modified By	Tami Shetty - 7/25/2025, 6:12 AM

Figure :Vehicle Service Request Layout with fields such as Vehicle Model, Customer, Service Date , Issue Description etc.



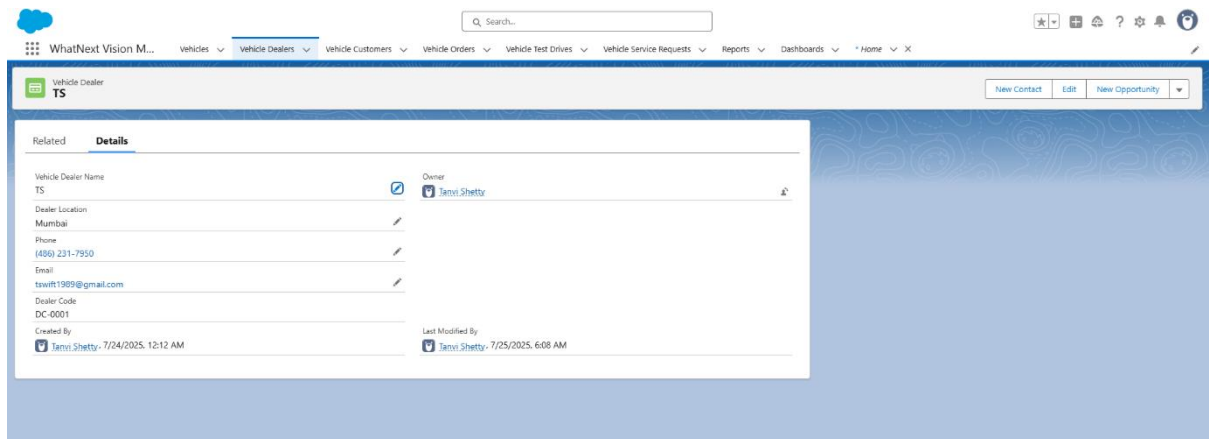
WhatNext Vision M... Vehicles Vehicle Dealers Vehicle Customers Vehicle Orders Vehicle Test Drives Vehicle Service Requests Reports Dashboards Home X

Vehicle Customer **Alex** New Contact Edit New Opportunity

Related Details

Vehicle Customer Name	Alex	Owner	Tami Shetty
Email	shettytami19@gmail.com		
Phone	(978) 687-7613		
Address	Mumbai		
Preferred Vehicle Type	Sedan		
Created By	Tami Shetty - 7/24/2025, 12:11 AM	Last Modified By	Tami Shetty - 7/24/2025, 12:11 AM

Figure :Vehicle Customer Layout with fields such as Vehicle Customer, Email, Phone, Address, Preferred Vehicle Type etc.



WhatNext Vision M... Vehicles Vehicle Dealers Vehicle Customers Vehicle Orders Vehicle Test Drives Vehicle Service Requests Reports Dashboards Home X

Vehicle Dealer **TS** New Contact Edit New Opportunity

Related Details

Vehicle Dealer Name	TS	Owner	Tami Shetty
Dealer Location	Mumbai		
Phone	(4080) 231-7950		
Email	tsav1989@gmail.com		
Dealer Code	DC-0001		
Created By	Tami Shetty - 7/24/2025, 12:12 AM	Last Modified By	Tami Shetty - 7/25/2025, 6:08 AM

Figure :Vehicle Dealer Layout with fields such as Vehicle Dealer Name , Location, Email, Phone, Dealer Code etc.

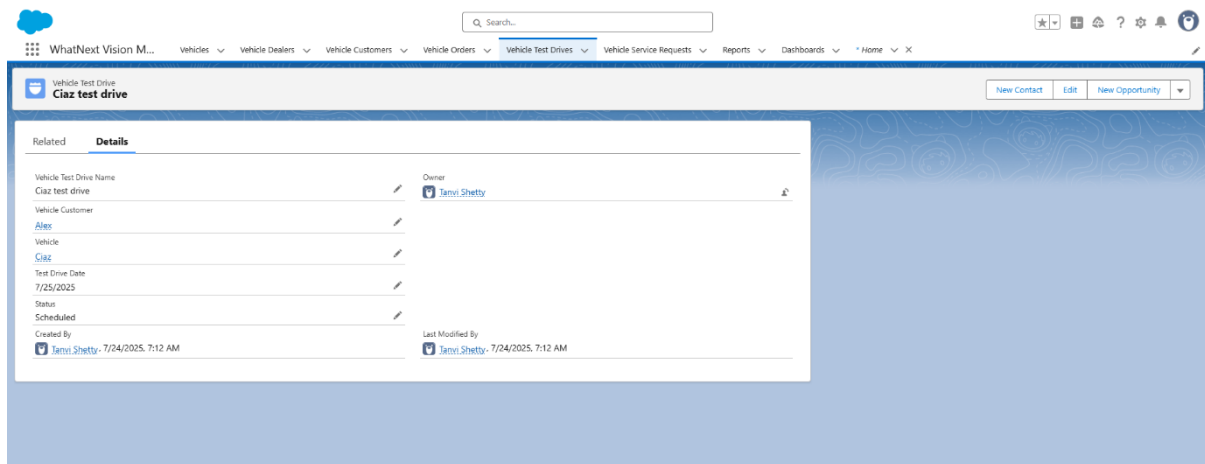


Figure :Vehicle Customer Layout with fields such as Vehicle Customer, Vehicle , Test Drive Date Status etc.

These designs were actually clear and the relevant fields were exposed to the user.

Dynamic Forms

On-forms were used with dynamic visibility rules that would improve user experience. For example: The Order Status field displays visibility, depending on the logic of stocks.

Test Drive responses can be viewed only after completion of test drive.

This made data entry to be simplified and reduced the possibility of the mistake generated by the user.

User Administration

Several user-types had different permission set:

- Sales Executive: Test drives /order create/ edit.
- Dealer Staff: Check current orders and their stock of vehicles.
- CRM admin: Ability to have every piece of data and configuration.

Object level and field level access was managed using Profiles and permission sets.

Reports and Dashboards

Important reports as well as dashboards were designed to deliver real-time answers:

- Test Drive Status Report: It manages scheduled, completed and cancelled test drives.
- Vehicle Stock Dashboard: A dashboard with visualization on available vs booked vehicles.
- Pending Orders Report: Displays pending orders because of out of stock.

These tools aided the management to make informed judgment and monitor the CRM performance.

Lightning record pages

Page-specific record screens were configured to have:

- More organization using tabs (e.g. Details, Related Orders).
- Pull up key fields (Order Status, Stock Count) into Highlights Panel.
- Associated List of associated records (e.g. Orders with regard to a vehicle).

These pages provided the user with a unified and a sensible experience in any of the devices.

Phase 4: Data Migration, Testing & Security

This phase ensured that the system was properly seeded with accurate data, thoroughly tested for logic and performance, and secure for different user roles. Below are the key activities conducted:

Data Loading Process

To load initial data into custom objects such as Vehicle, Order, and Test Drive, we can use:

- Data Import Wizard: For smaller datasets like vehicle listings and test drive bookings.
- Data Loader: Used for bulk uploading customer order records and large sets of vehicle inventory. It supported .CSV uploads and allowed us to map fields easily.

These tools helped ensure that all legacy data and test records were properly added to the system before testing.

Field History Tracking

Field history tracking was enabled on critical objects:

- Order: Tracked changes in Order Status, Assigned Dealer.
- Vehicle: Tracked Stock Availability and Price changes.

This allowed audit trails and accountability for updates made by sales or dealer users.

Duplicate Rules Matching Rules

In order to avert repeated data entry:

Matching Rules were set on the Customer and Order objects depending on the fields of email and phones.

The Duplicate Rules were forced in place to warn or prevent the users in saving duplicate data by keeping the CRM data clean.

Profile, Roles, and permission sets

- Permission Sets:

The permission sets allowed giving additional edit rights to a few chosen users without changing the base profiles.

Sharingrules

To enable, object-level sharing rules were defined to permit:

- The salespeople should watch the other sales unit cars test drive to coordinate.
- Dealer personnel being able to only view Orders placed under its dealership (through criteria-based sharing).

This guaranteed privacy and efficiency in operations.

Test Classes

The unit tests that were conducted were done in Apex Test Classes written to test:

- Auto update Order Status logic.
- Apex class to delegating the closest available dealer.

All the test classes had expected code coverage >75% to go into Salesforce. Data was put into the sample test with the help of @testSetup methods.

Phase 5: Deployment, Documentation & Maintenance

This is the process of transferring the developed and complete CRM system to production and makes it functional with good documentation and support system and an agreed maintenance strategy.

Deployment Strategy

We deployed Salesforce environments using Change Sets. The procedure entailed:

- To go to the Developer Sandbox then develop an Outbound Change Set.
- All custom elements: custom objects (Vehicle, Order, Test Drive), custom fields, validation rules, flows, Apex classes and triggers.
- Checking inbound Change Set In Production to ascertain no dependency problem.
- Implementation following validation and post implementation verification.
- This is a way of making the deployment organized and well managed, but with no need to re-create components using manual processes.

System Maintenance Monitoring

To keep and follow up the system (after implementation):

- Salesforce export options are used to schedule regular Data Backups.

- The Field History Tracking is active as to observe the alterations in the crucial objects (Order, Vehicle).
- Flows and Apex error alerting to point out failures to admin.
- Admin User Login Weekly to audit access to users, verify dashboards, and sort any user problems.

Updates or developments made to the system (e.g. new fields or flows) will be tested in the sandbox first before being released in production through Change Sets.

Troubleshooting Method

A systematic problem-solving approach had been used, and recorded:

1. Issue reported by the user: Entered in shared spreadsheet with time and position.
2. First Check: Admin checks logs, field history or debugging flow logs.
3. Produce: Admin tries to recreate problem in sandbox.
4. Fix: in case of a bug detected, a patch or updated component is produced in sandbox and deployed again.
5. Fix details and steps: To have an idea in future, fix details and steps are added to the internal project documentation.

This makes sure there is minimal interference and effective resolution.

Conclusion

WhatsNext Vision Motors CRM project was completed successfully enabling the achievement of its objectives with respect to streamlining vehicle orders, dealer allocations and simplification of test-drive schedules. The solution now offers the sales and dealer teams a central facility to operate as a result of its strong data controls, security roles, and automation. The system has been proved, implemented and delivered with all the documents. This secure base can now be relied upon as future improvements are made to improve the business process even more.