

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

ABSTRACT

This project was completed as part of the Salesforce Virtual Internship Program 2025, organized by SmartBridge, in partnership with AICTE and Salesforce. The goal of the project was to use Salesforce CRM tools to solve a real-world problem for a company called WhatNext Vision Motors.

The main focus of the project titled “Enhancing Customer Ordering Experience in Salesforce CRM for WhatNext Vision Motors” was to make the vehicle ordering process easier and more efficient for both customers and the company. The project made sure that customers could only order vehicles that were available in stock. It also helped automatically assign orders to the nearest dealership, based on where the customer lives.

To build this system, we used Salesforce features like custom objects, flows, and Apex code. We also created background jobs to update bulk orders regularly. These tools helped automate the process and reduce manual work.

This project improved the way the company handles orders, saved time for the sales team, and gave customers a better experience. The internship helped us learn how to use Salesforce in a real business setting, using tools like Flow Builder, Apex, and Lightning App Builder.

Objectives

The primary goals of this internship project were to:

1. Understand and apply Salesforce CRM tools to automate key business processes in a simulated automotive environment.
2. Design a custom order management system that improves efficiency, accuracy, and customer experience.
3. Build and configure custom objects and relationships to represent vehicles, customers, dealers, and orders.
4. Use declarative tools like Flow Builder to automate dealer assignment and validate stock in real time.

5. Implement Apex triggers and Batch Apex to handle order logic, inventory updates, and bulk operations.
6. Test and deploy the solution effectively, ensuring functionality aligns with business requirements.
7. Gain hands-on experience in project development, debugging, and version control using Salesforce and GitHub.

Technology Description

This internship project was built entirely on the Salesforce CRM platform, utilizing both declarative and programmatic tools to solve real-world business challenges in the automotive domain.

1. Platform: Salesforce CRM

Salesforce was used as the core cloud-based platform for building and deploying the order management system. It enabled:

- Custom Data Modeling using standard and custom objects (e.g., Vehicle, Dealer, Customer, Order)
- Process Automation with Flows, validation rules, and assignment logic
- User Interface Design using Lightning App Builder for better usability

2. Declarative Tools

The project leveraged point-and-click features extensively:

- Flow Builder: Designed Record-Triggered Flows to auto-assign dealers and validate vehicle stock
- Lightning App Builder: Created responsive page layouts and dashboards for different users
- Validation Rules: Enforced form-level conditions such as preventing submission without required fields

3. Programmatic Tools

Advanced business logic was implemented using:

- Apex Triggers: For real-time stock checks and automatic order updates
- Apex Classes: Used to organize trigger logic with proper separation of concerns (trigger handlers)
- Batch Apex: Scheduled jobs to process large records like checking stock and updating order statuses

- Scheduled Apex: Automated stock and order updates at regular intervals

4. Version Control and Submission

- GitHub was used for code versioning, maintaining the repository for Apex classes, triggers, and documentation.

5. Learning Tools

- Trailhead Modules & Superbadges: Strengthened concepts of object relationships, flows, Apex, and authentication
- SmartInternz Workspace: Used for task management, tracking progress, and project collaboration

Detailed Execution of Project Phases

Phase 1: Requirement Gathering & Planning

- Understood the business model of the fictional company, *WhatNext Vision Motors*, and identified the key pain points in order processing and stock tracking.
- Documented the functional requirements such as:
- Stock availability check
 - Dealer assignment automation
 - Order status updates
 - Test drive scheduling and notifications
- Planned the system flow and listed down all needed custom objects, automation rules, and Apex components.
- Tools Used: Google Docs, SmartInternz platform, Salesforce Workbench

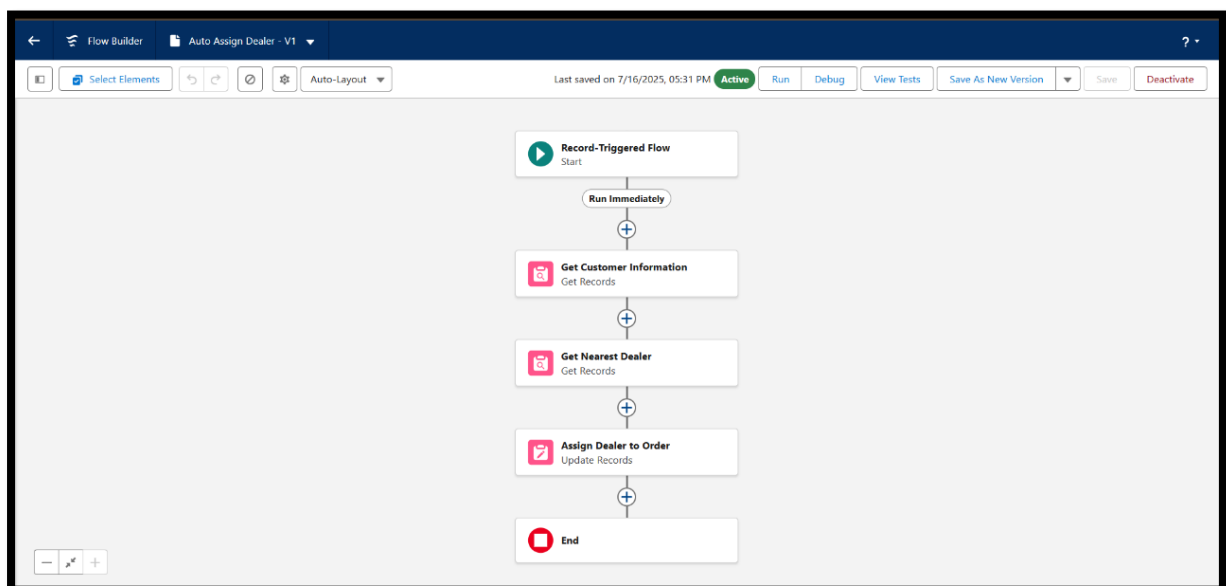
Phase 2: Data Modeling & Object Creation

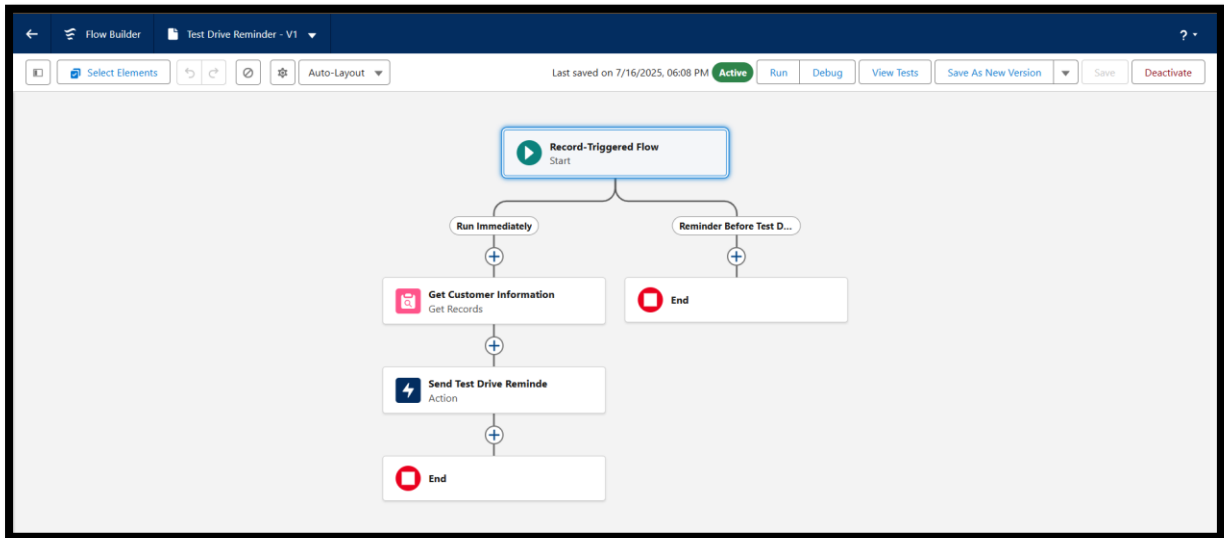
- Created **custom objects** in Salesforce to represent entities like:
 - Vehicle__c, Dealer__c, Customer__c, Order__c
- Defined relationships between objects (Lookups and Master-Detail where needed).
- Added custom fields such as stock count, city, pin code, and order status.
- Designed the schema to enable:
 - Filtering vehicles based on stock
 - Assigning dealers based on the customer city

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Dealer	Dealer__c	Lookup(Vehicle Dealer)		✓
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓
Price	Price__c	Currency(2, 2)		
Status	Status__c	Picklist		
Stock Quantity	Stock_Quantity__c	Number(18, 0)		
Vehicle Model	Vehicle_Model__c	Picklist		
Vehicle Name	Name	Text(90)		✓

Phase 3: Declarative Automation Using Flow Builder

- Built a Record-Triggered Flow for the Order__c object that:
 - Fetches the customer's city
 - Queries Dealer__c records in that city
 - Assigns the closest matching dealer
 - Checks if the selected vehicle has available stock
 - Updates the order status as "Confirmed" or "Pending" accordingly
- Used Flow elements like:
 - Get Records, Assignment, Decision, Update Records



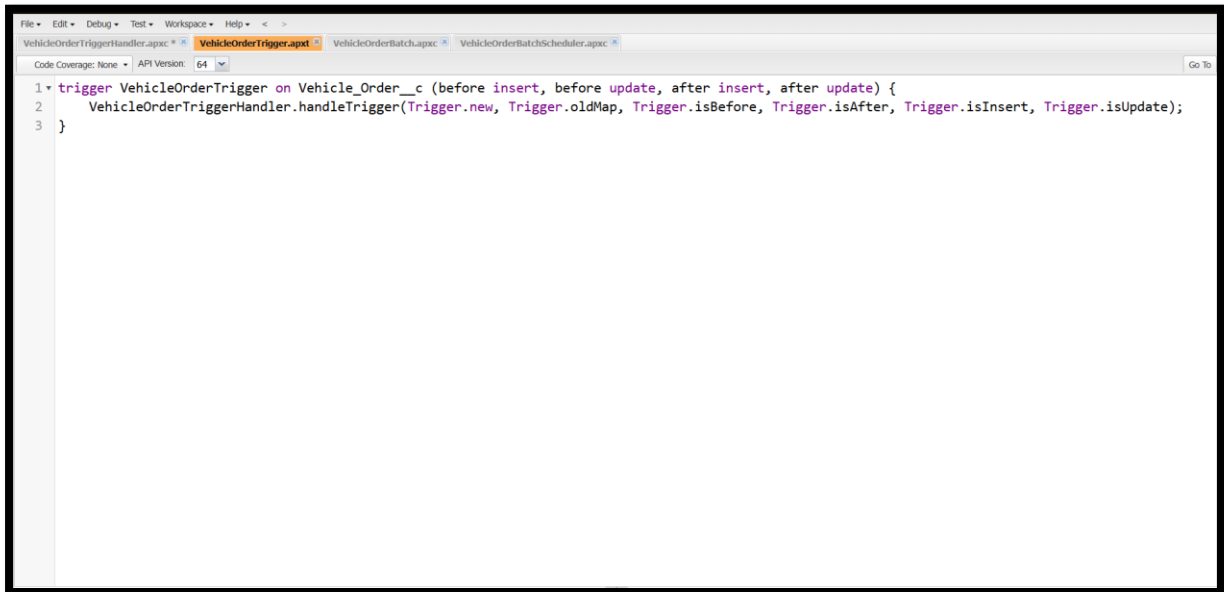


Phase 4: Programmatic Logic using Apex

- Created Apex Triggers
- To reduce stock count once an order is placed
- Prevent order creation if vehicle is unavailable (as fallback logic)
- Used trigger handlers to maintain clean, modular code
- Wrote Apex Classes with error handling (try-catch blocks) for stability

```

1 public class VehicleOrderTriggerHandler {
2     public static void handleTrigger(List<Vehicle_Order__c> newOrders, Map<Id, Vehicle_Order__c> oldOrders, Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate) {
3         if (isBefore && (isInsert || isUpdate)) {
4             preventOrderIfOutOfStock(newOrders);
5         }
6         if (isAfter && (isInsert || isUpdate)) {
7             updateStockOnOrderPlacement(newOrders);
8         }
9     }
10    private static void preventOrderIfOutOfStock(List<Vehicle_Order__c> orders) {
11        Set<Id> vehicleIds = new Set<Id>();
12        for (Vehicle_Order__c order : orders) {
13            if (order.Vehicle__c != null) {
14                vehicleIds.add(order.Vehicle__c);
15            }
16        }
17        if (vehicleIds.isEmpty()) {
18            Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>();
19            [SELECT Id, Stock_Quantity__c FROM Vehicle__c WHERE Id IN :vehicleIds];
20        }
21        for (Vehicle_Order__c order : orders) {
22            if (vehicleStockMap.containsKey(order.Vehicle__c)) {
23                Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
24                if (vehicle.Stock_Quantity__c <= 0) {
25                    order.addError('This vehicle is out of stock. Order cannot be placed.');
```



```
File Edit Debug Test Workspace Help
VehicleOrderTriggerHandler.apex VehicleOrderBatch.apex VehicleOrderBatchScheduler.apex
Code Coverage: None API Version: 64 Go To

1 trigger VehicleOrderTrigger on Vehicle_Order__c (before insert, before update, after insert, after update) {
2     VehicleOrderTriggerHandler.handleTrigger(trigger.new, trigger.oldMap, trigger.isBefore, trigger.isAfter, trigger.isInsert, trigger.isUpdate);
3 }
```

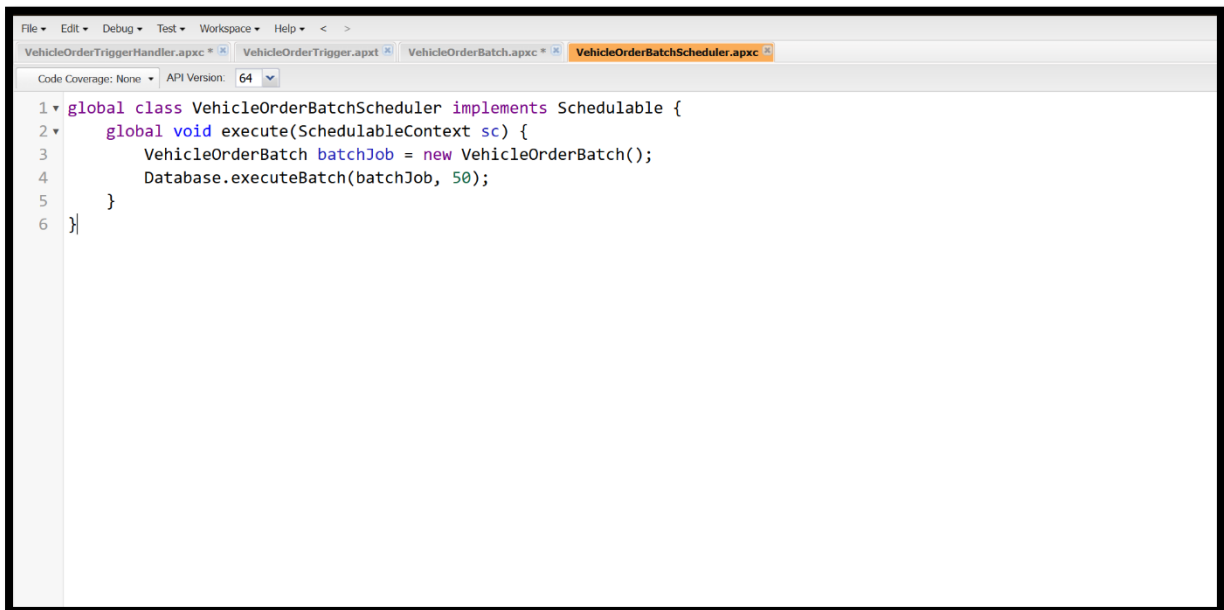


```
File Edit Debug Test Workspace Help
VehicleOrderTriggerHandler.apex VehicleOrderBatch.apex VehicleOrderBatchScheduler.apex
Code Coverage: None API Version: 64 Go To

1 global class VehicleOrderBatch implements Database.Batchable<Object> {
2     global Database.QueryLocator start(Database.BatchableContext bc) {
3         return Database.getQueryLocator([
4             SELECT Id, Status__c, Vehicle__c
5             FROM Vehicle_Order__c
6             WHERE Status__c = 'Pending'
7         ]);
8     }
9     global void execute(Database.BatchableContext bc, List<Vehicle_Order__c> orders) {
10         Set<Id> vehicleIds = new Set<Id>();
11         for (Vehicle_Order__c order : orders) {
12             if (order.Vehicle__c != null) {
13                 vehicleIds.add(order.Vehicle__c);
14             }
15         }
16         if (!vehicleIds.isEmpty()) {
17             Map<Id, Vehicle__c> vehicleStockMap = new Map<Id, Vehicle__c>([
18                 SELECT Id, Stock_Quantity__c FROM Vehicle__c WHERE Id IN :vehicleIds
19             ]);
20             List<Vehicle_Order__c> ordersToUpdate = new List<Vehicle_Order__c>();
21             List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();
22             for (Vehicle_Order__c order : orders) {
23                 Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
24                 if (vehicle.Stock_Quantity__c > 0) {
25                     order.Status__c = 'Confirmed';
26                     vehicle.Stock_Quantity__c -= 1;
27                     ordersToUpdate.add(order);
28                     vehiclesToUpdate.add(vehicle);
29                 }
30             }
31             if (!ordersToUpdate.isEmpty()) {
32                 update ordersToUpdate;
33             }
34             if (!vehiclesToUpdate.isEmpty()) {
35                 update vehiclesToUpdate;
36             }
37         }
38     }
39     global void finish(Database.BatchableContext bc) {
40         System.debug('Vehicle order batch job completed.');
```

Phase 5: Batch Processing & Scheduling

- Developed a Batch Apex Job that runs periodically to:
- Check Order__c records with “Pending” status
- Cross-reference with Vehicle__c stock levels
- Update order status to “Confirmed” if stock becomes available
- Scheduled this job using Scheduled Apex to run daily

A screenshot of an IDE window showing four tabs: VehicleOrderTriggerHandler.apxc, VehicleOrderTrigger.apxt, VehicleOrderBatch.apxc, and VehicleOrderBatchScheduler.apxc. The active tab is VehicleOrderBatchScheduler.apxc. The code is as follows:

```
1 global class VehicleOrderBatchScheduler implements Schedulable {  
2     global void execute(SchedulableContext sc) {  
3         VehicleOrderBatch batchJob = new VehicleOrderBatch();  
4         Database.executeBatch(batchJob, 50);  
5     }  
6 }
```

Phase 6: Testing & Validation

- Created test records to simulate real-world scenarios for:
 - Out-of-stock orders
 - Dealer assignment logic
 - Trigger-based stock updates
- Achieved over 75% test coverage for Apex code through:
 - Test classes and assertions
- Verified Flow logic paths via manual and debug log testing

Phase 7: Deployment & Submission

- Uploaded Apex classes and flows to GitHub with version control
- Created a demo video showcasing the full process (order creation, auto-assignment, and stock tracking)
- Submitted all deliverables via SmartInternz Workspace, including:
 - Project codebase
 - Reports
 - Documentation

Project Explanation with Real-World Example

Let's say a customer named Ramesh from Hyderabad wants to book a VisionX SUV.

Here's how the system works:

1. Order Creation

- Ramesh has an order through the company's portal (which connects to Salesforce).
- A new Order__c record is automatically created in the Salesforce CRM.

2. Dealer Assignment (Automatic)

- The system checks Ramesh's city (Hyderabad).
- It automatically looks for the nearest available dealer from the Dealer__c records.
- Let's say "Hyderabad Motors" is the closest dealer — the system assigns the order to them without human input.

3. Stock Validation

- The system checks if VisionX SUV is in stock (Vehicle__c object).
- If available → the order status becomes "Confirmed".
- If out of stock → it becomes "Pending" and will be confirmed later.

4. Batch Job for Stock Updates

- A Batch Apex job runs every day to:
 - Check all Pending orders.
 - See if the requested vehicle has been restocked.
 - If yes, update the order to "Confirmed".

So if Ramesh's SUV comes back in stock the next day, the system automatically updates her order status — no manual work needed!

5. Communication & Automation

- The system can send automated emails or SMS:

- "Your vehicle has been booked successfully."
- "We're checking availability — you'll be updated soon!"
- "Good news! Your order has been confirmed."

Tools Used:

- Salesforce Flows → For dealer assignment and status updates
- Apex Triggers → To reduce stock after an order
- Batch Apex → To handle bulk order updates
- Custom Objects → To model real-world data like Vehicles, Dealers, Orders

In Summary:

This project is a simulation of how real automotive companies can use Salesforce CRM to:

- Avoid manual tasks
- Improve customer satisfaction
- Ensure orders are processed quickly and accurately
- Handle large volumes of data automatically

It mirrors the digital transformation happening in many modern auto companies like Tata Motors, Mahindra, or even Hyundai — where CRM and automation play a huge role in delivering a smooth customer experience.

Conclusion

This internship project at *WhatNext Vision Motors* gave me the opportunity to design and implement an end-to-end Salesforce CRM-based order management system. Through this experience, I gained practical exposure in building custom objects, automating business workflows using Flows and Apex, and scaling features using Batch Apex jobs.

The solution effectively addressed core challenges faced by automotive companies, such as:

- Preventing order placement for out-of-stock vehicles
- Automating dealer assignment using customer location
- Periodically updating order statuses via batch processing

This project not only enhanced my Salesforce development skills but also improved my problem-solving, testing, and project delivery discipline.

Future Scope

1. AI-Based Order Prediction

Use Salesforce Einstein or similar tools to predict high-demand vehicle models and suggest restocking actions based on trends.

2. Real-Time Customer Updates

Enable automatic notifications through email or SMS to keep customers informed about their order status, delays, or stock changes.

3. Visual Inventory Tracker

Create an interactive dashboard to help staff monitor stock levels, order queues, and dealership performance easily.

4. Vehicle Reservation System

Allow customers to temporarily reserve vehicles for a fixed period, reducing order dropouts and improving purchase planning.

5. Multi-language Support

Enhance the platform with multi-language options to make it accessible to customers across different regions of India.