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

---

Project Overview

WhatNext Vision Motors, a rising leader in the automotive sector, set out to modernize customer experiences and streamline vehicle order management through a customized Salesforce CRM implementation. The project's key objectives included automating dealer assignments, validating vehicle stock in real time, and enhancing customer engagement using Lightning Apps and backend automations. By addressing operational bottlenecks and reducing manual errors, the CRM supports Vision Motors' commitment to innovation and mobility excellence.

Project Objectives

- Centralize and digitize vehicle, dealer, and customer data
- Automate dealer assignment based on customer geography
- Validate vehicle stock availability during order creation
- Enable seamless test-drive scheduling and service requests
- Improve reporting and analytics for better business decisions
- Establish scalable automations for future enhancements (AI recommendations, chatbot integration)

Implementation Phases

Phase 1: Requirement Analysis and Planning

- Business Needs: Overcome manual delays and inventory mismanagement, enable smoother customer journeys
- Scope Definition: Cover vehicle ordering, dealer management, test-drive tracking, automated communications
- Stakeholder Mapping: Sales, service, dealer network, customer support

Phase 2: Salesforce Development – Backend Configuration

- Objects & Fields: Custom objects (Vehicle, Order, Dealer, Customer, Test Drive)
- Validation Rules: Stock availability, required fields, business logic enforcement
- Automations: Flows, Workflow Rules, Process Builders for dealer assignment, order status updates, email reminders
- Apex Development: Triggers and batch classes for backend efficiency

Phase 3: User Interface & User Experience (UI/UX)

- Lightning App: "WhatNext Vision Motors" app in App Launcher
- Dynamic Forms & Layouts: Modern, responsive record pages for all key objects
- User Management: Role-based access for sales teams, dealers, and support staff

Phase 4: Data Migration, Testing, and Security

- Data Import: Ensured seamless migration of legacy data using Data Loader & Import Wizard
- Field History Tracking: Enabled for auditability and transparency
- Duplicate Management: Matching and duplicate rules to maintain clean records
- Security Model: Role hierarchies, profiles, permission sets, sharing rules

Phase 5: Deployment and Maintenance

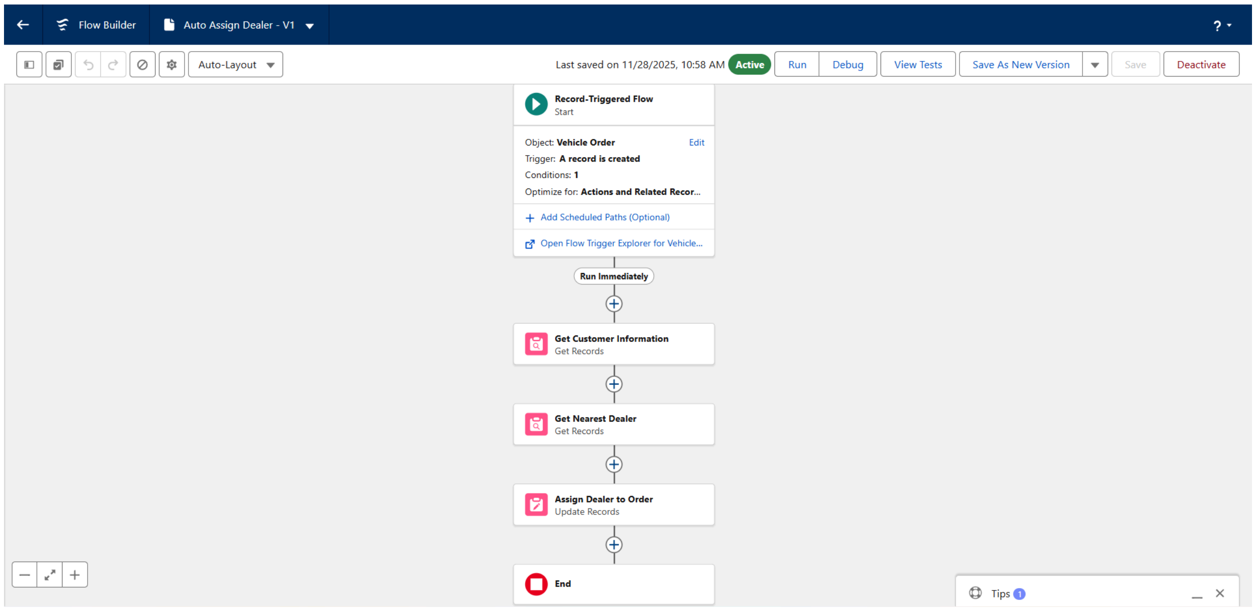
- Deployment: Managed via Salesforce change sets and versioning
- Maintenance: Defined troubleshooting workflows and scheduled health checks
- Documentation: Maintained logic records and support references for future teams

Conclusion

The Salesforce CRM project for WhatNext Vision Motors successfully delivered on its aims to streamline customer order management and enhance operational workflows. Automated dealer assignment, real-time stock validation, and robust reporting resulted in improved customer satisfaction and internal efficiency. The CRM platform lays a scalable foundation for future technological expansion and continued innovation in the mobility sector.

Additional Points

Screen captures



Last saved on 11/28/2025, 10:58 AM

Object: Vehicle Order
Trigger: A record is created
Conditions: 1
Optimize for: Actions and Related Recor...

Figure 1. Auto Assign Dealer Flow

This flow allows the app to get the location information of the user and the dealer, so that it can assign the nearest dealer the user can go to.

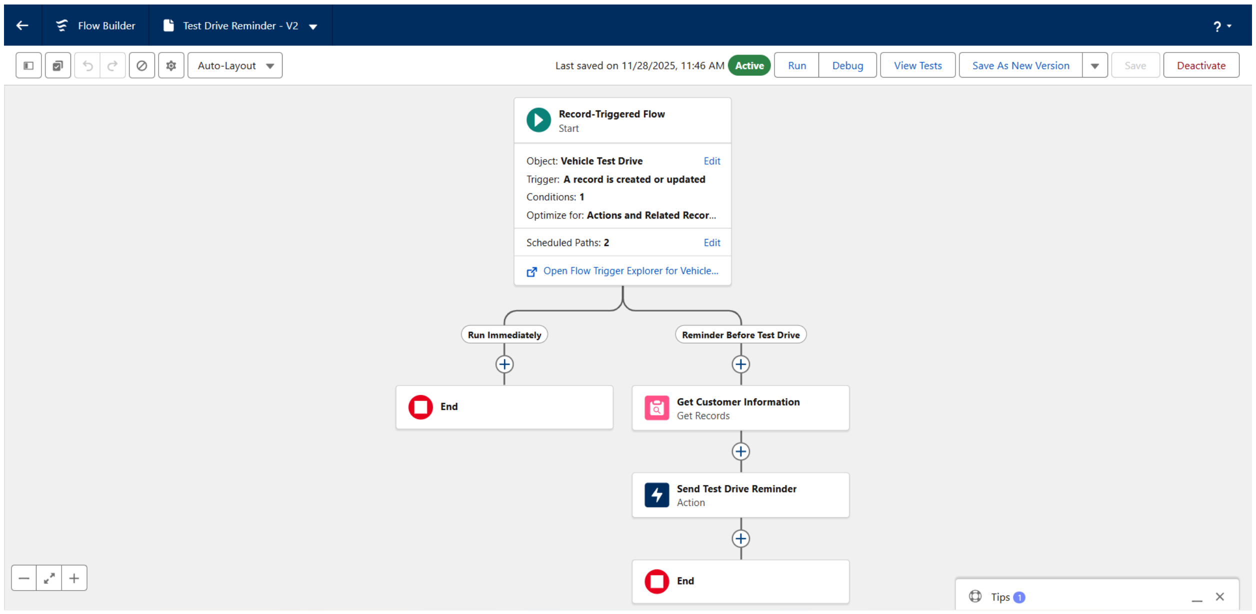
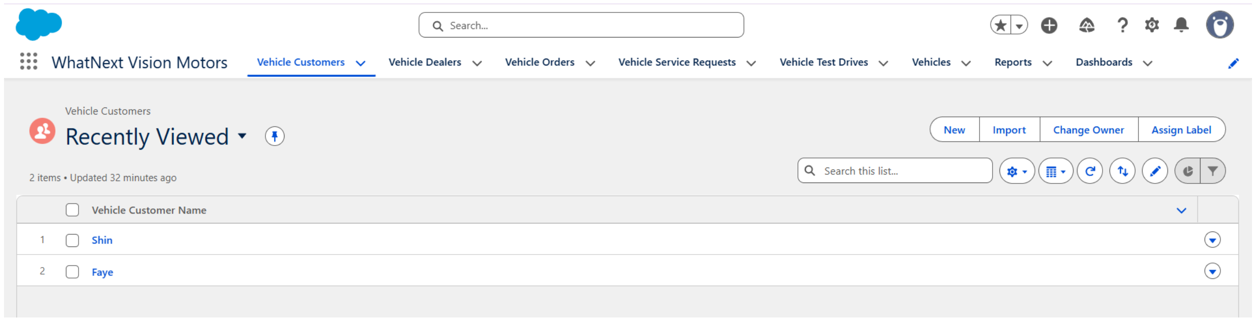


Figure 2. Test Drive Reminder Flow

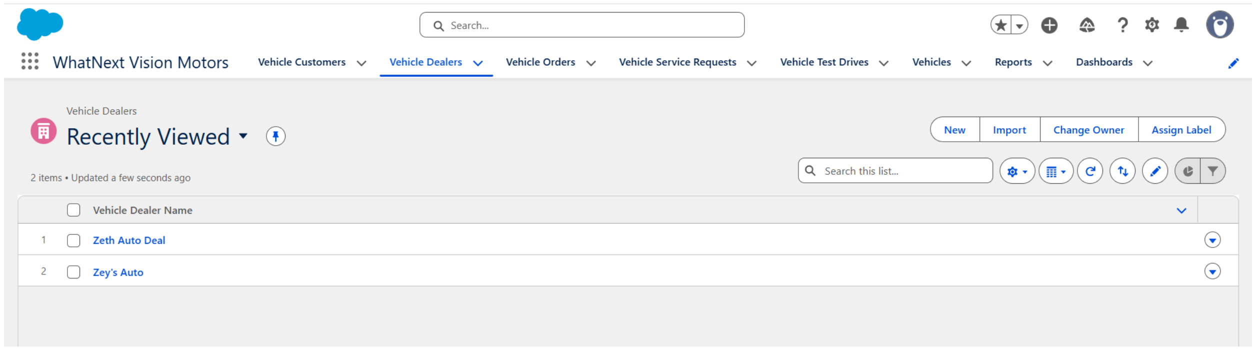
This lets the app remind the user of their scheduled driver test by getting the information from the user



	Vehicle Customer Name
1	Shin
2	Faye

Figure 3. Vehicle Customer Object

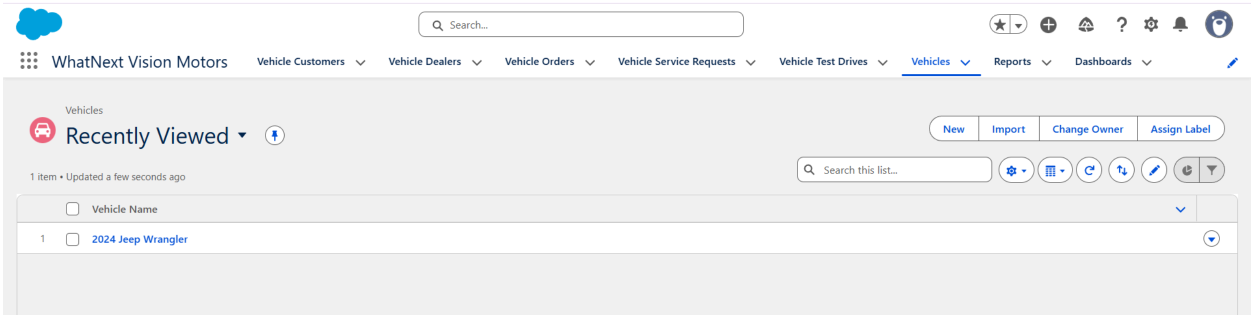
Allows us to create customers who are willing to buy a vehicle



	Vehicle Dealer Name
1	Zeth Auto Deal
2	Zey's Auto

Figure 4. Vehicle Dealer Object

Allows us to create vehicle dealers who will sell cars to customers

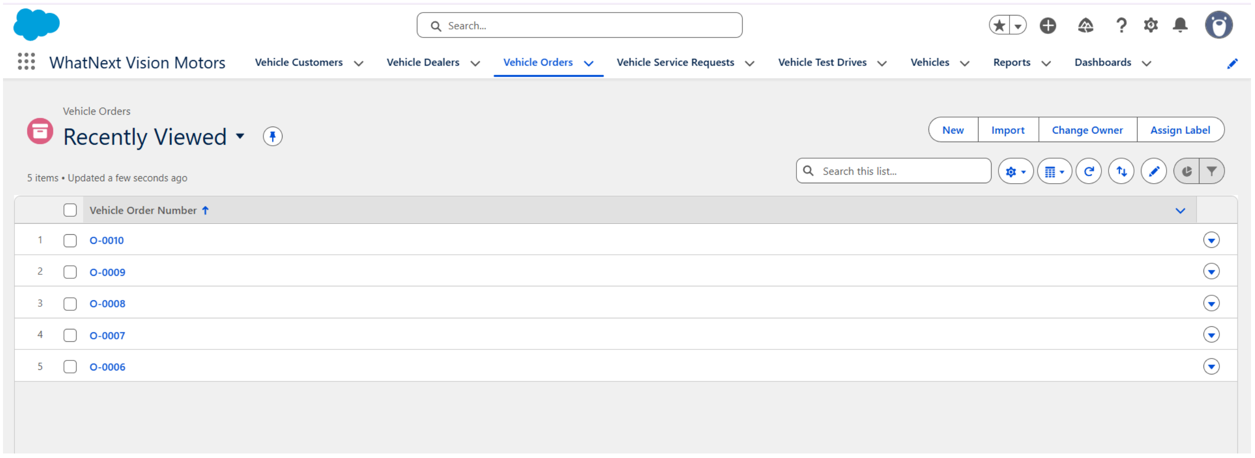


1 item • Updated a few seconds ago

	Vehicle Name
1	2024 Jeep Wrangler

Figure 5. Vehicle Object

Allows us to add cars to the list, this is where the customers can choose which vehicle they like

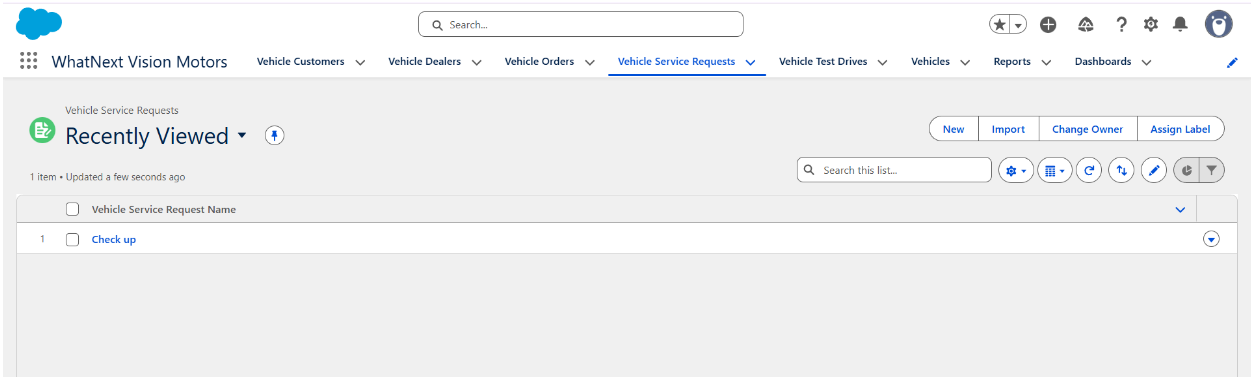


5 items • Updated a few seconds ago

	Vehicle Order Number ↑
1	O-0010
2	O-0009
3	O-0008
4	O-0007
5	O-0006

Figure 6. Vehicle Order Object

Allows us to see how many orders are made by different users, and which dealer they are communicating with



1 item • Updated a few seconds ago

	Vehicle Service Request Name
1	Check up

Figure 7. Vehicle Service Request Object

Allows the customers to send a request for a service that they want for their vehicle

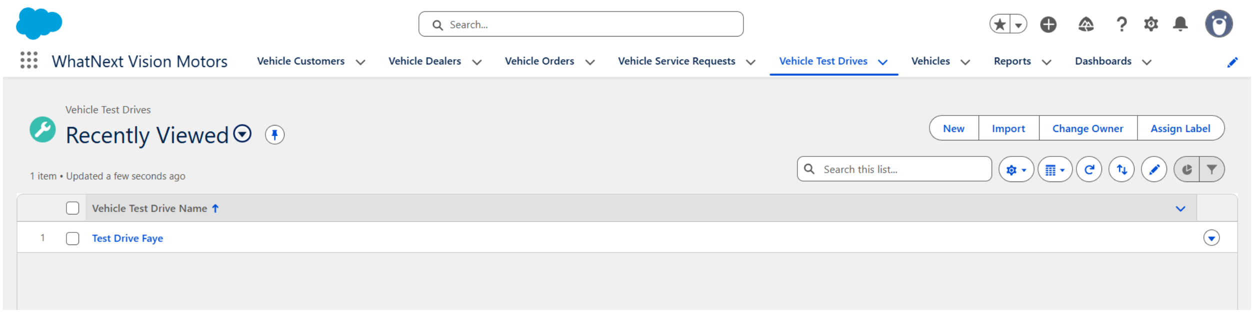
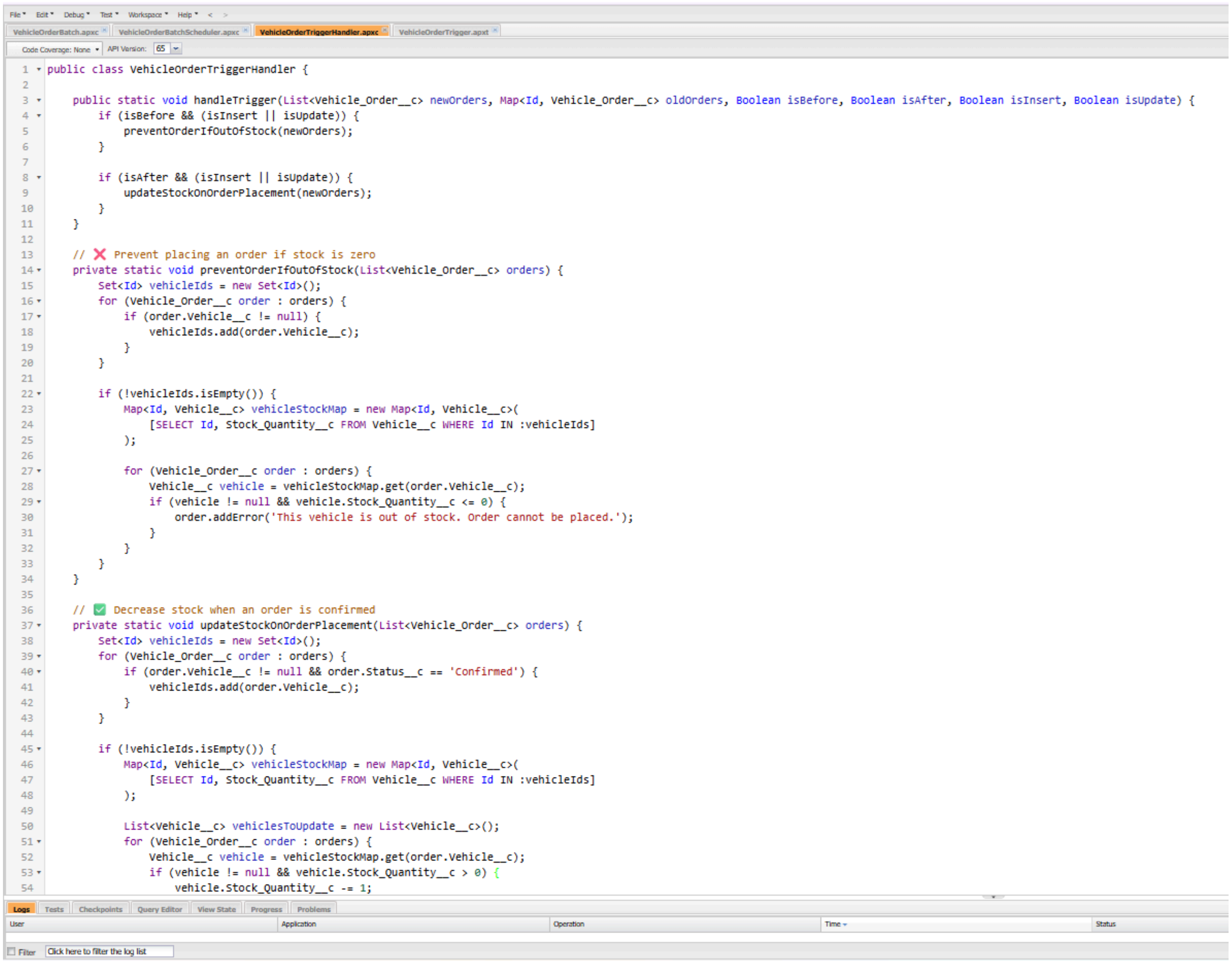


Figure 8. Vehicle Test Drives

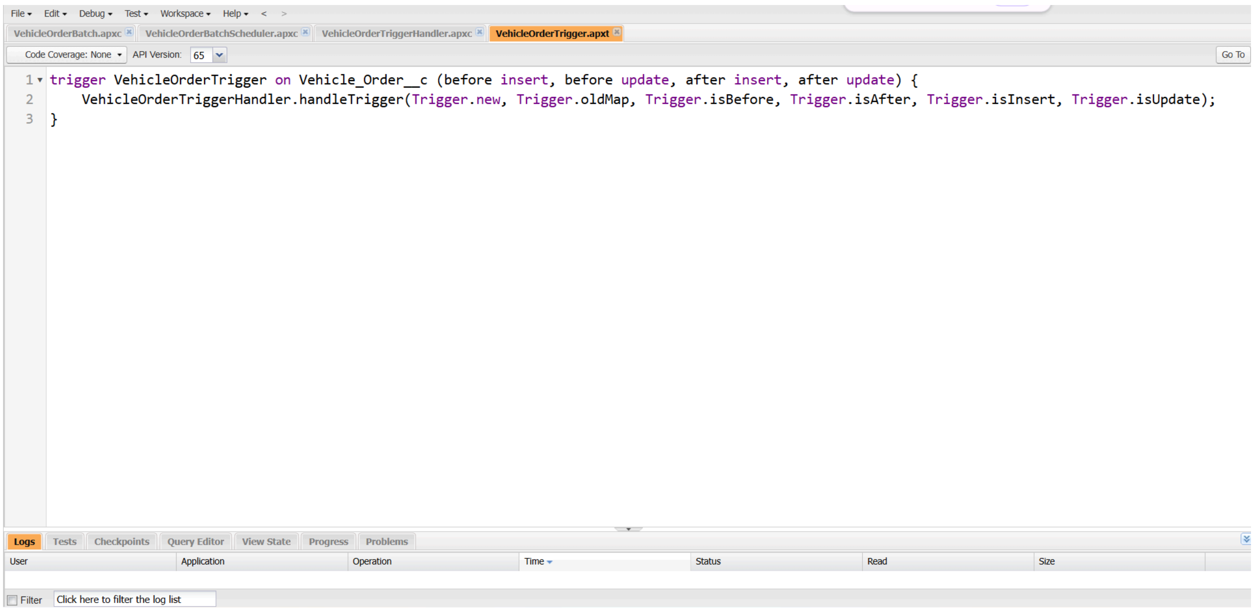
Allows us to schedule test drives for the customer to allow them to experience the vehicle



```
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;
```

Figure 9. VehicleOrderTriggerHandler

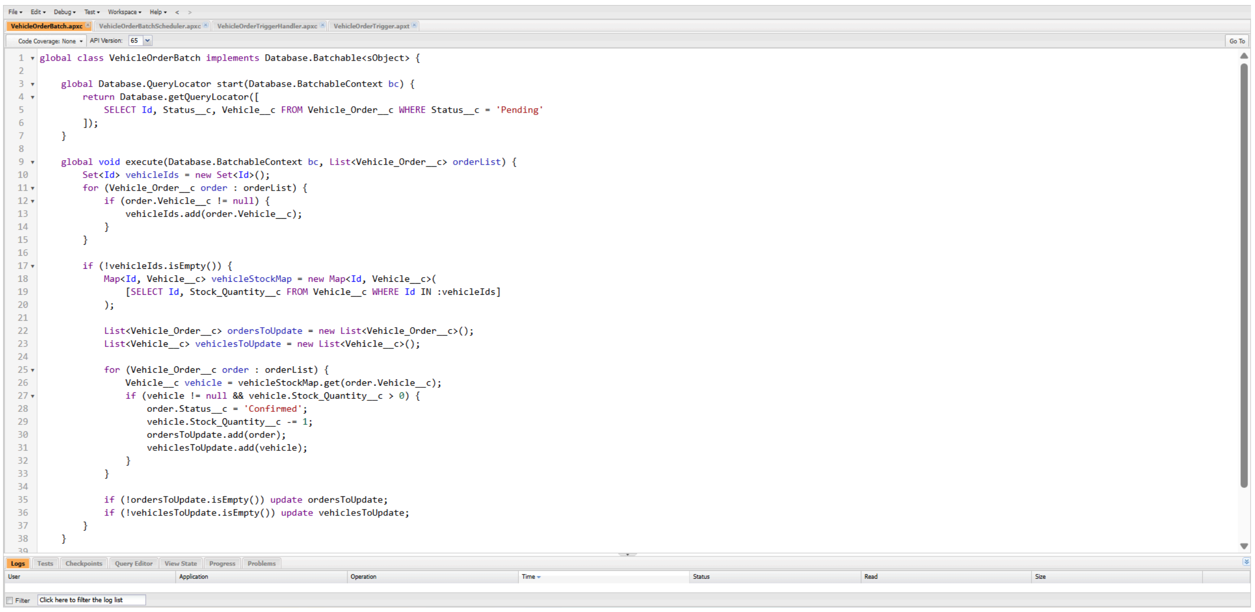
Allows us to Prevent placing an order if stock is zero and Decrease stock when an order is confirmed



```
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);
}
```

Figure 10. VehicleOrderTrigger

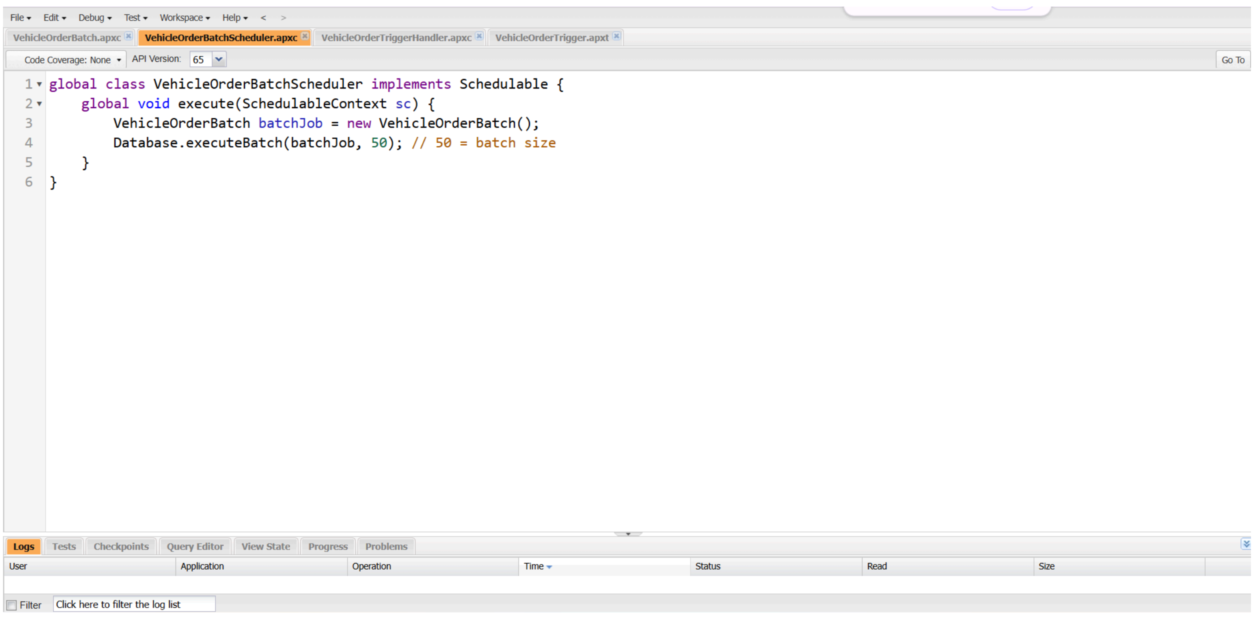
automatic action that runs whenever someone creates or updates a Vehicle Order record in Salesforce



```
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;
        }
    }
```

Figure 11. VehicleOrderBatch

used to process a large number of vehicle orders


File Edit Debug Test Workspace Help
VehicleOrderBatch.apxc
VehicleOrderBatchScheduler.apxc
VehicleOrderTriggerHandler.apxc
VehicleOrderTrigger.apxt
Code Coverage: None
API Version: 65
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems
User Application Operation Time Status Read Size
Filter
Click here to filter the log list

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

Figure 12. VehicleOrderBatchScheduler

implements the Schedulable interface and is responsible for scheduling the VehicleOrderBatch