

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

## **Overview**

WhatNext Vision Motors is dedicated to modernizing mobility through a strategic Salesforce CRM upgrade designed to enhance both operational efficiency and the customer journey. The core of this initiative involves ensuring that orders are reliable; the system prevents customers from ordering vehicles that are not in stock and automatically routes orders to the dealership closest to the buyer. To support this, background automation manages tasks such as sending polite test drive reminders and updating order statuses without manual interference. These processes are powered technically by Apex triggers for inventory monitoring and scheduled batch jobs that maintain accurate stock levels, ultimately simplifying the purchase process and minimizing business errors.

## **Core Objectives**

The primary goals are to boost customer satisfaction and streamline internal workflows by reducing manual mistakes. Technically, the project aims to deliver:

- Apex triggers to handle stock validation.
- Batch jobs to manage stock updates.
- Scheduled Apex to automate the processing of orders.

## **Phase 1: Requirement Analysis & Planning**

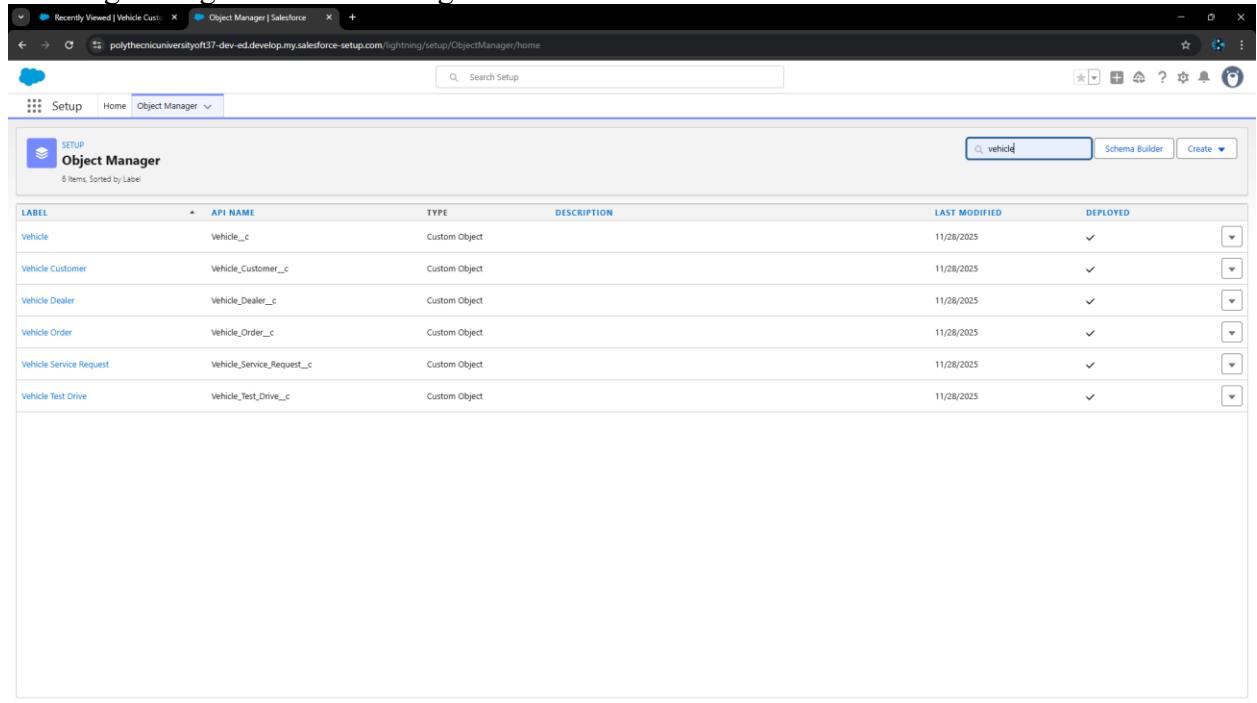
The project strategy was built around four essential requirements:

1. **CRM Implementation:** The system must centralize the management of vehicle details, stock availability, and dealer data, while efficiently tracking service requests and customer test drives. Workflows were required to assign orders based on location.
2. **Process Automation:** Crucial business rules included preventing orders for out-of-stock items, auto-assigning local dealers, and triggering automated email reminders for appointments.
3. **Apex and Triggers:** The team needed to implement triggers to enforce logic like stock validation and ensure code maintainability through modular trigger handlers.
4. **Batch Jobs:** Development of batch Apex was required to periodically audit vehicle stock levels and send notifications regarding replenishment and order processing.

## **Phase 2: Backend Development & Configuration**

This phase focused on constructing the data architecture within Salesforce. The team created specific Custom Objects to manage the data relationships effectively.

- **Custom Objects Created:** The system architecture includes objects for Vehicle, Vehicle Customer, Vehicle Dealer, Vehicle Order, Vehicle Service Request, and Vehicle Test Drive.
- **Tab Configuration:** To make these objects accessible to users, custom tabs were configured for each object (e.g., specific tabs for Vehicles, Customers, and Orders), allowing for organized data management within the Salesforce window.



The screenshot shows the Salesforce Object Manager interface. At the top, there's a navigation bar with tabs for 'Setup' and 'Object Manager'. Below the navigation is a search bar with the placeholder 'Search Setup' and a search icon. A secondary search bar at the top right contains the text 'vehicle'. On the far right of the header are icons for 'Schema Builder' and 'Create'. The main area is titled 'Object Manager' and displays a table of six custom objects. The columns are labeled: 'LABEL', 'API NAME', 'TYPE', 'DESCRIPTION', 'LAST MODIFIED', and 'DEPLOYED'. The objects listed are: Vehicle (Vehicle\_\_c, Custom Object, 11/28/2025, ✓), Vehicle Customer (Vehicle\_Customer\_\_c, Custom Object, 11/28/2025, ✓), Vehicle Dealer (Vehicle\_Dealer\_\_c, Custom Object, 11/28/2025, ✓), Vehicle Order (Vehicle\_Order\_\_c, Custom Object, 11/28/2025, ✓), Vehicle Service Request (Vehicle\_Service\_Request\_\_c, Custom Object, 11/28/2025, ✓), and Vehicle Test Drive (Vehicle\_Test\_Drive\_\_c, Custom Object, 11/28/2025, ✓). Each row has a small dropdown arrow icon at the end of the 'DEPLOYED' column.

Figure 1. Object Creation

## Phase 3: UI/UX Development & Automation

To improve the user experience, a custom "New Lightning App" was built using the App Manager. This allows users to personalize their navigation bar, ensuring that essential items—like Vehicle Customers and Orders—are easily accessible on both desktop and phone interfaces.

- **Field Creation:** Specific data points were defined for the objects. For example, the Vehicle Customer object was configured with fields for Address, Email, Phone, and Customer Name to ensure comprehensive data capture. The Vehicle Order object tracks

the Order Date, Dealer, and Customer.

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Address	Address_c	Text(60)		
Created By	CreatedById	Lookup(User)		
Email	Email_c	Email		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User/Group)		✓
Phone	Phone_c	Phone		
Preferred Vehicle Type	Preferred_Vehicle_Type_c	Picklist		
Vehicle Customer Name	Name	Text(80)		✓
Vehicle Order	Vehicle_Order_c	Lookup(Vehicle Order)		✓
Vehicle Test Drive	Vehicle_Test_Drive_c	Lookup(Vehicle Test Drive)		✓

Figure 2. Field Creation1

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Order Date	Order_Date_c	Date		
Owner	OwnerId	Lookup(User/Group)		✓
Status	Status_c	Picklist		
Vehicle	Vehicle_c	Lookup(Vehicle)		✓
Vehicle Customer	Vehicle_Customer_c	Lookup(Vehicle Customer)		✓
Vehicle Order Name	Name	Text(80)		✓

Figure 3. Field Creation2

- Flow Automation:** Two primary automation flows were built:

1. **Auto Assign Dealer:** A record-triggered flow that gathers customer information, identifies the nearest dealer, and updates the order record to assign that dealer.

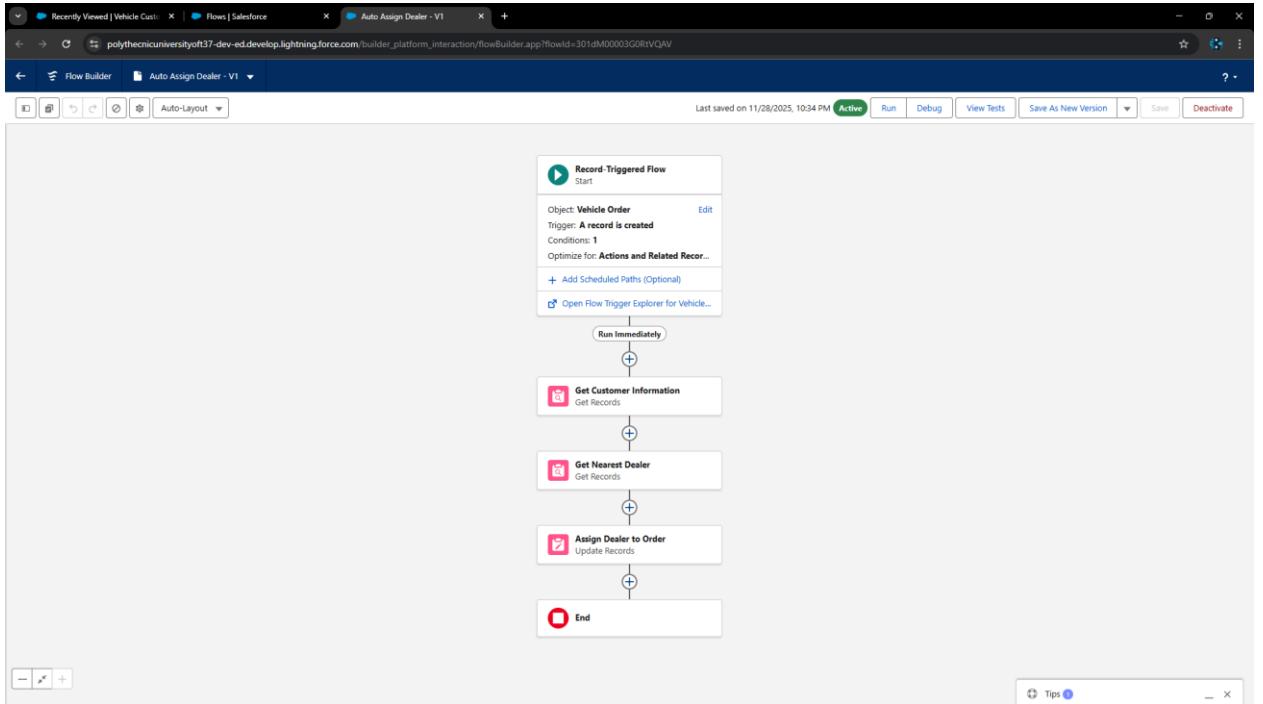


Figure 4. Auto Assign Dealer Flow

2. **Test Drive Reminder:** A flow designed to automatically send email reminders to customers regarding their upcoming test drives.

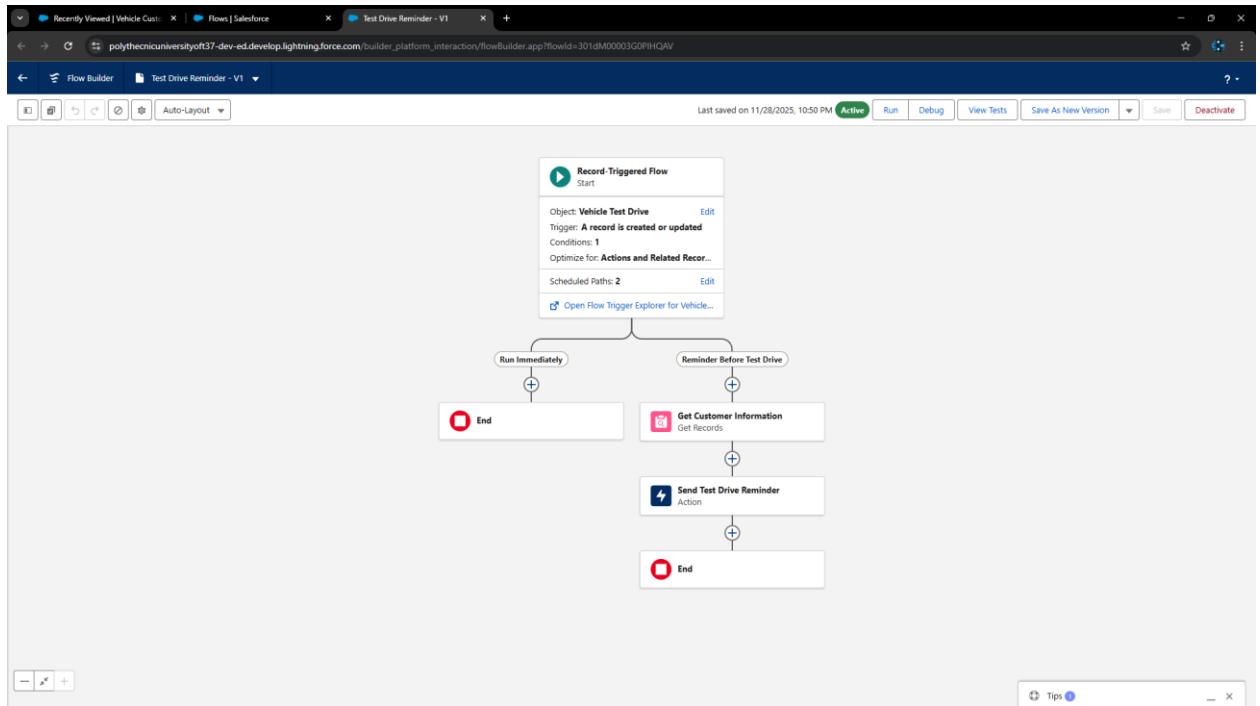
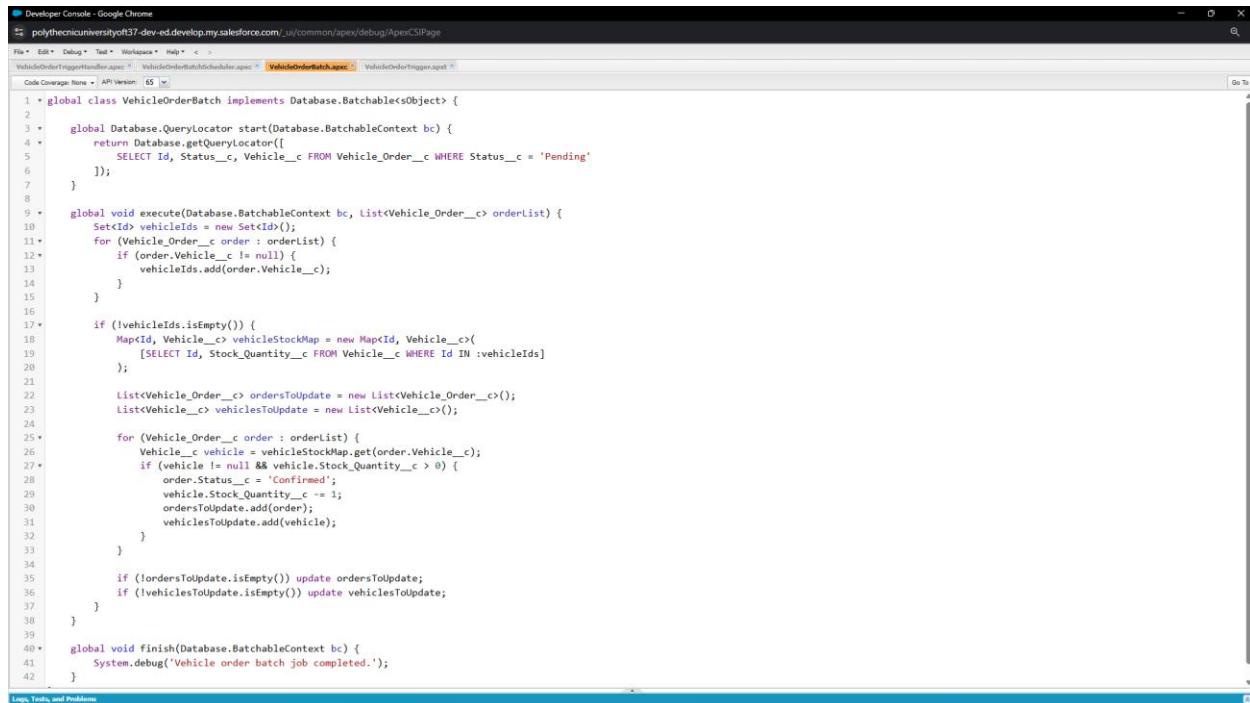


Figure 5. Test Drive Reminder Flow

## Phase 4: Data Migration, Testing & Security

This phase involved the heavy lifting regarding code and logic implementation.

- **Apex Development:** Developers utilized the Developer Console to create the VehicleOrderTriggerHandler class. The trigger logic is set to run on Vehicle Order events (before/after insert and update) to enforce business rules.
- **Batch Processing:** A global class named VehicleOrderBatch was implemented to handle high-volume updates.
  - **Logic:** The batch job queries for orders with a "Pending" status. It collects Vehicle IDs and checks the stock map.
  - **Execution:** If the Vehicle\_\_c is not null and the Stock\_Quantity\_\_c is greater than zero, the system updates the order status to "Confirmed".
- **Testing:** Test cases were run to verify inputs and outputs, such as checking a Honda Civic order against quantity levels and ensuring email reminders (e.g., "Reminder: Your Test Drive is Tomorrow!") were delivered correctly to the spam/inbox.



The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is polytechnicuniversityof37-dev-ed.develop.my.salesforce.com/\_ui/common/apex/debug/ApexCSIPage. The tab bar shows three tabs: VehicleOrderTriggerHandler.apex, VehicleOrderBatch.apex (which is the active tab), and VehicleOrderTrigger.apex. The code editor displays the Apex code for the VehicleOrderBatch class. The code implements a Database.Batchable<sObject> interface. It starts by querying pending vehicle orders. Then, it iterates through the results to collect vehicle IDs. Finally, it checks the stock map for each vehicle and updates the order status if the vehicle ID is not null and the stock quantity is greater than zero. The code also includes logic to update the order and vehicle status after processing.

```
1 * global class VehicleOrderBatch implements Database.Batchable<sObject> {
2 *
3 *     global Database.QueryLocator start(Database.BatchableContext bc) {
4 *         return Database.getQueryLocator([
5 *             SELECT Id, Status__c, Vehicle__c FROM Vehicle_Order__c WHERE Status__c = 'Pending'
6 *         ]);
7 *     }
8 *
9 *     global void execute(Database.BatchableContext bc, List<Vehicle_Order__c> orderList) {
10 *         Set<Id> vehicleIds = new Set<Id>();
11 *         for (Vehicle_Order__c order : orderList) {
12 *             if (order.Vehicle__c != null) {
13 *                 vehicleIds.add(order.Vehicle__c);
14 *             }
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 *
22 *             List<Vehicle_Order__c> ordersToUpdate = new List<Vehicle_Order__c>();
23 *             List<Vehicle__c> vehiclesToUpdate = new List<Vehicle__c>();
24 *
25 *             for (Vehicle_Order__c order : orderList) {
26 *                 Vehicle__c vehicle = vehicleStockMap.get(order.Vehicle__c);
27 *                 if (vehicle != null && vehicle.Stock_Quantity__c > 0) {
28 *                     order.Status__c = 'Confirmed';
29 *                     vehicle.Stock_Quantity__c -= 1;
30 *                     ordersToUpdate.add(order);
31 *                     vehiclesToUpdate.add(vehicle);
32 *                 }
33 *             }
34 *
35 *             if (!ordersToUpdate.isEmpty()) update ordersToUpdate;
36 *             if (!vehiclesToUpdate.isEmpty()) update vehiclesToUpdate;
37 *         }
38 *     }
39 *
40 *     global void finish(Database.BatchableContext bc) {
41 *         System.debug('Vehicle order batch job completed.');
42 *     }
43 *
```

Figure 6. Vehicle Order Batch

The screenshot shows the Salesforce Developer Console in Google Chrome. The title bar says "Developer Console - Google Chrome". The URL is "polytechnicuniversityoft37-dev-ed.develop.my.salesforce.com/\_ui/common/apex/debug/ApexCSIPage". The tabs at the top are "VehicleOrderTriggerHandler.apfc", "VehicleOrderBatchScheduler.apfc", "VehicleOrderBatch.apfc", and "VehicleOrderTrigger.apot". The "VehicleOrderTrigger.apot" tab is selected. The code editor contains the following Apex trigger:

```
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 7. Vehicle Order Trigger

The screenshot shows the Salesforce Developer Console in Google Chrome. The title bar says "Developer Console - Google Chrome". The URL is "polytechnicuniversityoft37-dev-ed.develop.my.salesforce.com/\_ui/common/apex/debug/ApexCSIPage". The tabs at the top are "VehicleOrderTriggerHandler.apfc", "VehicleOrderBatchScheduler.apfc", "VehicleOrderBatch.apfc", and "VehicleOrderTrigger.apot". The "VehicleOrderBatchScheduler.apfc" tab is selected. The code editor contains the following Apex class:

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

Figure 8. Vehicle Order Batch Scheduler

```

1 * public class VehicleOrderTriggerHandler {
2
3     public static void handleTrigger(List<Vehicle_Order__c> newOrders, Map<Id, Vehicle_Order__c> oldOrders, Boolean isBefore, Boolean isAfter, Boolean isInsert, Boolean isUpdate) {
4
5         if (isBefore) {
6
7             if (isInsert || isUpdate) {
8
9                 preventOrderIfOutOfStock(newOrders);
10            }
11        }
12    }
13
14
15
16    if (isAfter) {
17
18        if (isInsert || isUpdate) {
19
20            updateStockOnOrderPlacement(newOrders);
21        }
22    }
23
24
25
26
27 }
28
29 // Method to prevent orders when the vehicle is out of stock
30
31 private static void preventOrderIfOutOfStock(List<Vehicle_Order__c> orders) {
32
33     Set<Id> vehicleIds = new Set<Id>();
34
35     for (Vehicle_Order__c order : orders) {
36
37         if (order.Vehicle__c != null) {
38
39             vehicleIds.add(order.Vehicle__c);
40
41     }
42 }

```

Figure 9. Vehicle Order Trigger Handler

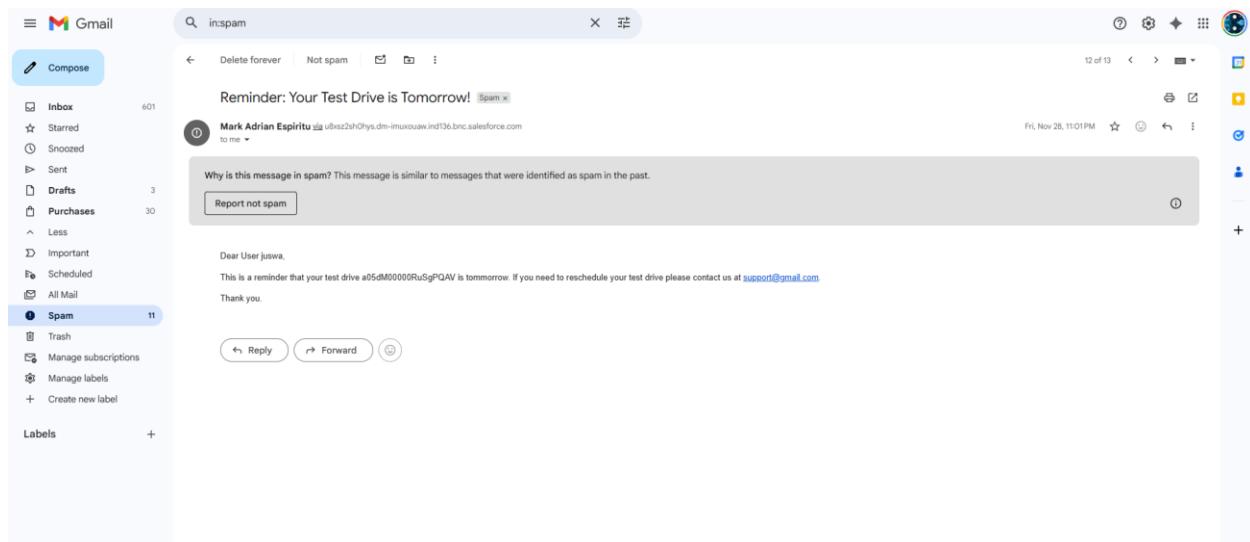


Figure 10. Testing of Test Drive Reminder

## Phase 5: Deployment, Documentation & Maintenance

The final phase marked the full deployment of the system, including all custom objects, flows, Apex classes, and dashboards.

- Validation:** Functional testing was conducted to ensure that automations—specifically dealer assignment and stock validation—executed without errors.
- Maintenance Strategy:** To ensure continued accuracy and efficiency, the maintenance plan involves regular monitoring of Apex logs and reviewing flow executions.

**Conclusion:** By integrating real-time stock validation, automated dealer assignment, and proactive communication, WhatNext Vision Motors has established a reliable, error-reduced system that simplifies operations for admins and smooths the purchasing process for customers.