## Salesforce-AutoCRM-Hub

# Phase 1: Problem Understanding & Industry Analysis

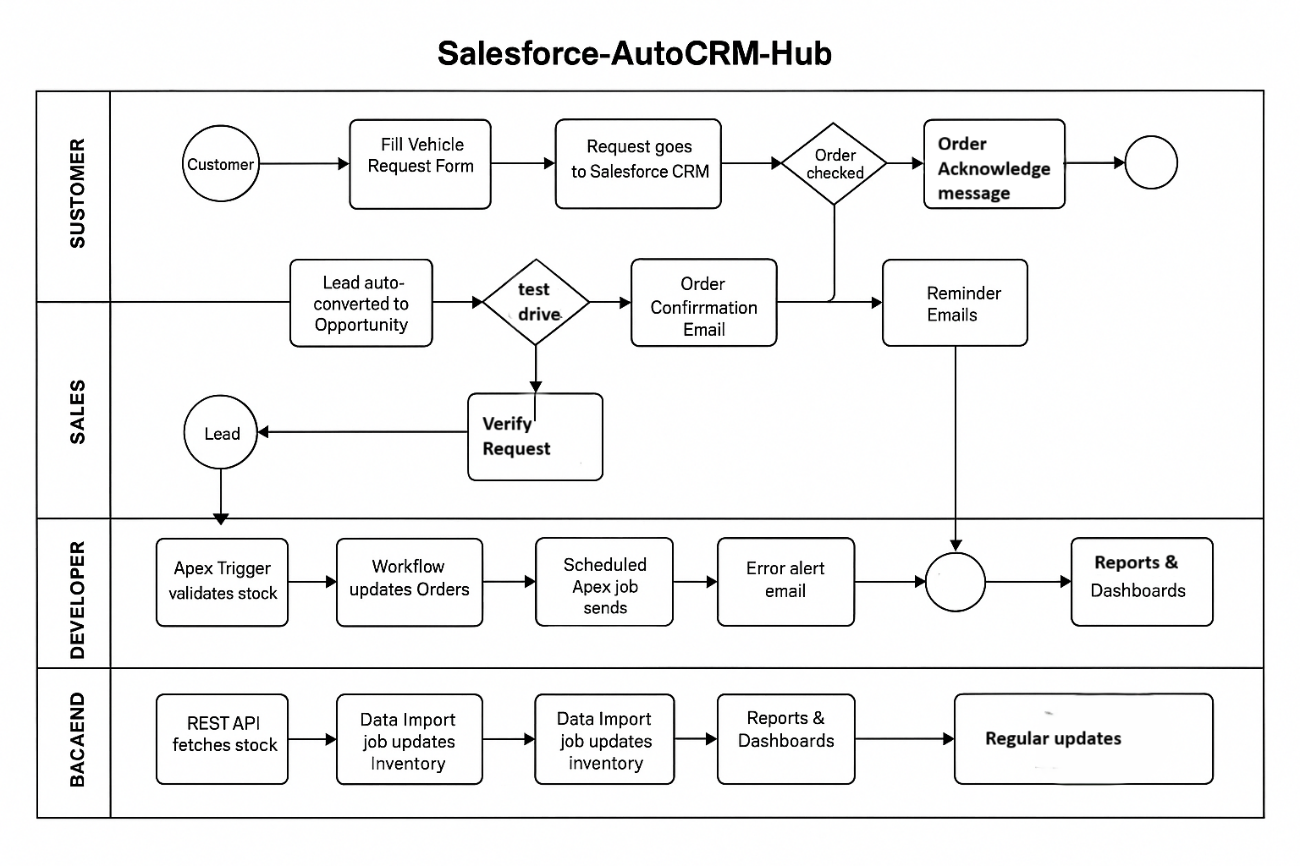
### **A. Requirements Gathering**

* Creation of custom objects to model core entities: **Order**, **Vehicle**, **Dealer**, **Customer**, **Inventory/Stock**, **TestDriveSchedule**.
* Importing or simulating datasets for these entities: sample data for customers, dealer locations, vehicle stock levels, and order history.

### **B. Stakeholder Simulation (Roles in Salesforce Context)**

* **Customer Role (simulated records):** interacts with ordering system, requests test drives, expects status updates and reminders.
* **Dealer Role (via Dealer custom object):** receives orders auto-assigned, holds inventory, updates stock; sees alerts for low stock.
* **Sales / Order Management Role:** monitors order pipeline, resolves exceptions (stock mismatch, unassigned orders), reviews dashboards.
* **Administrator / Developer Role:** defines custom objects, implements logic (Apex triggers, batch jobs, workflows), maintains the system.

### **C. Business Process Mapping**



**Process Map Components:**

1. **Customer Order Entry** → order request submitted (fields: customer details, vehicle model, quantity).
2. **Stock Validation** → an Apex trigger checks Inventory to ensure requested vehicle is available.
3. **Dealer Assignment** → if stock OK, system locates nearest dealer (based on customer geolocation or predefined zones) and assigns order.
4. **Order Status Updates** → workflows or triggers update order status (Pending → Assigned → In Transit / Ready for Test Drive → Delivered).
5. **Notifications/Reminders** → scheduled Apex / workflow sends reminders to customer (e.g. for test drive appointment) and dealers.
6. **Stock Updates** → batch jobs run at set intervals to sync inventory across dealers or update stock levels.
7. **Reporting & Dashboard** → system aggregates data to show metrics (order volume, stock shortages, dealer load, customer satisfaction).

### **D. Industry-Specific Use Case Analysis (Research Based)**

In the automotive sector, CRMs face unique challenges:

* **Large Dealer Networks:** Must assign the nearest dealer fairly & quickly.
* **Dynamic Inventory:** Vehicles go in/out of stock rapidly.
* **Customer Expectations:** Real-time communication is the norm.

Competitive Benchmarking:

* Some automotive companies already use CRMs but lack automated dealer assignment.
* Salesforce-AutoCRM-Hub differentiates itself by combining **order automation, dealer assignment, and proactive communication**.

### **E. AppExchange Exploration**

Explore Salesforce marketplace (AppExchange) for reference solutions

* Dealer Management apps
* Vehicle Inventory add-ons
* Email/SMS automation tools

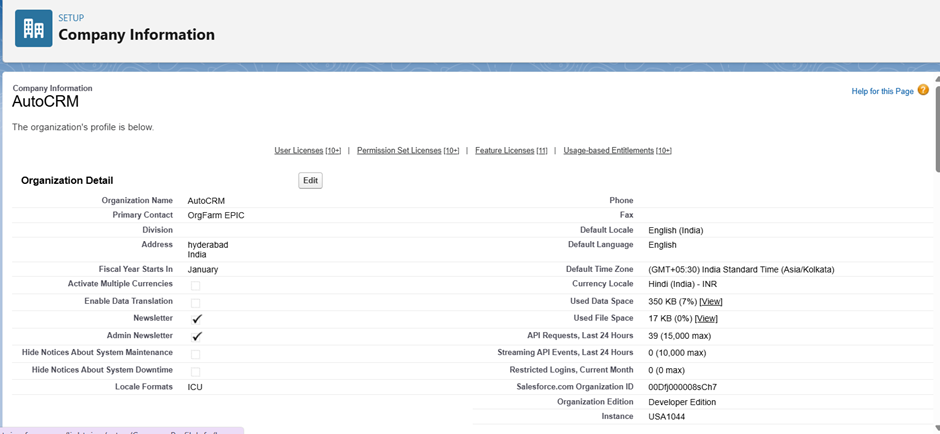
Building custom solution.

# PHASE 2 - Org Setup & Configuration

**Company Profile Setup**

Basic org details are configured under **Setup → Company Information → Edit** to establish the foundational settings for the AutoCRM org.

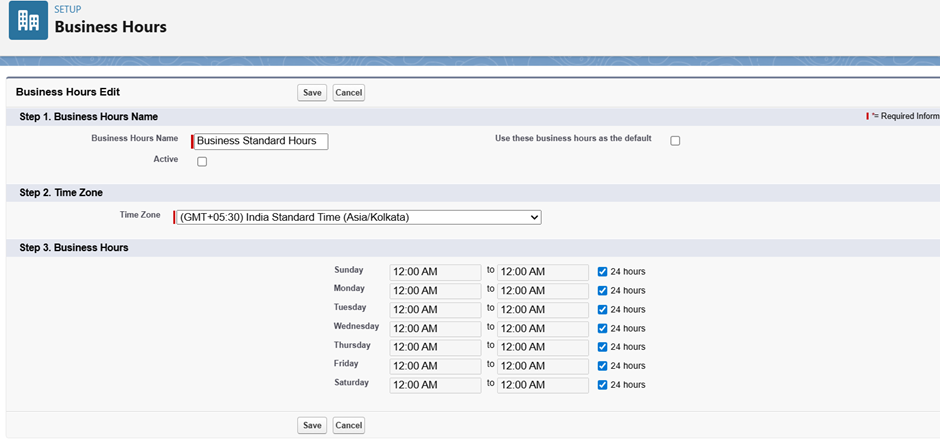
* **Name**: AutoCRM
* **Time Zone**: GMT+05:30 Asia/Kolkata
* **Locale**: English (India)
* **Language**: English
* **Currency**: INR



**Business Hours Setup**

Working hours are configured to reflect operating times for future case management and SLA tracking.

* **Path**: Setup → Business Hours → New.
* **Name**: Standard Restaurant Hours.
* **Time Zone**: GMT+05:30 Asia/Kolkata.
* **Working Hours**: Mon-Sun 10:00 AM–10:00 PM.



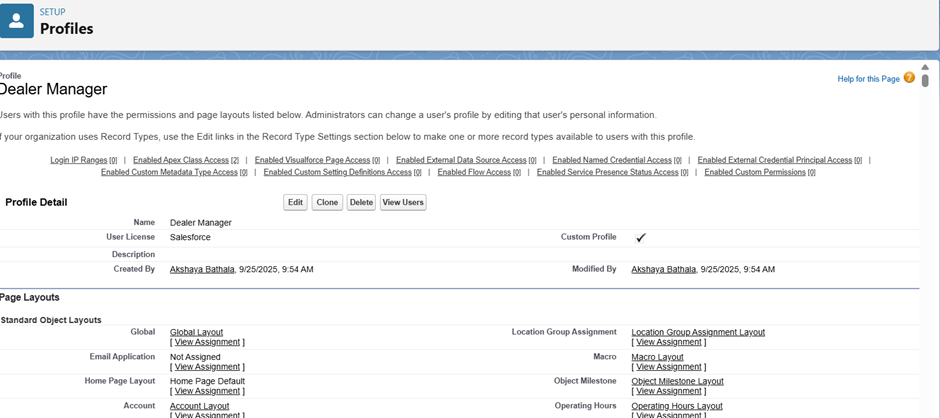
**Fiscal Year Setup**

A standard fiscal year is established to define reporting periods for sales and order analysis.

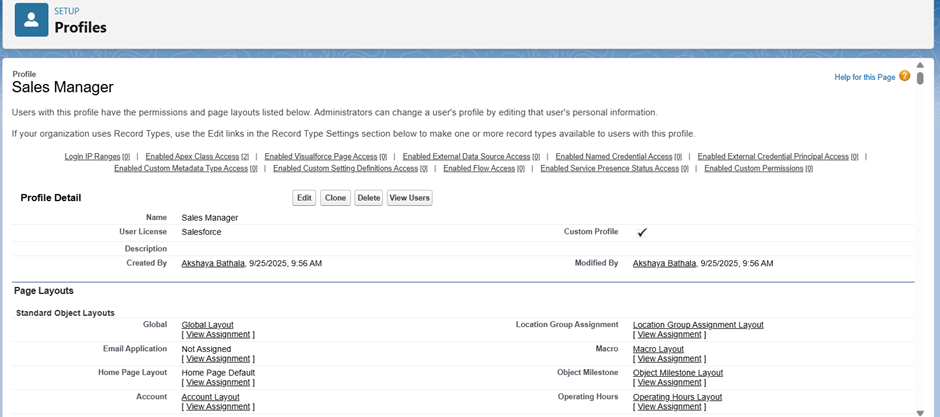
* **Path**: Setup → Fiscal Year.
* **Type**: Standard Fiscal Year.
* **Configuration**: The starting month is set to **January**.

**User Setup (Profiles, Roles, Permission Sets, Users)**

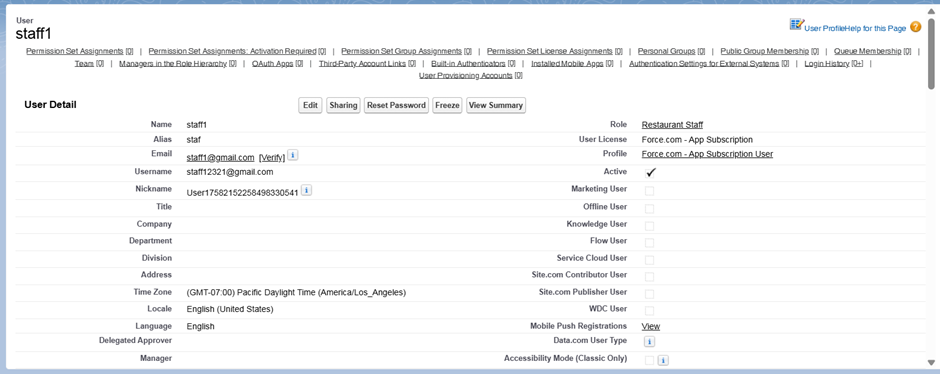
* **Profiles**: Custom profiles were created by cloning standard profiles to provide baseline access.
  + **Dealer Manager**: For daily order entry and customer interaction.



* + **Sales Manager**: For operational oversight, reporting, and manual adjustments.

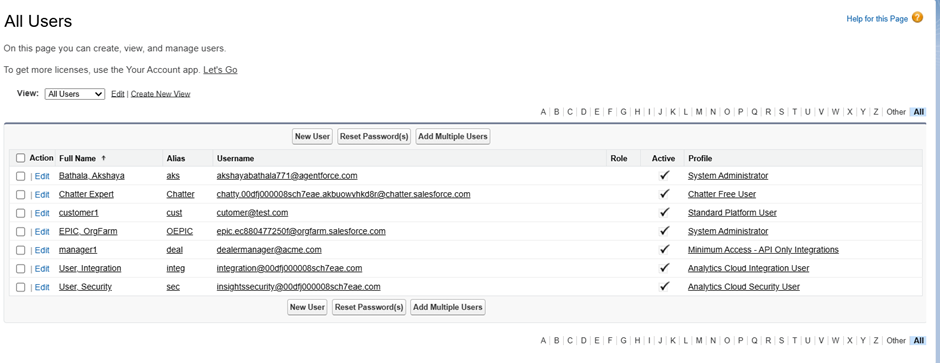


* **Roles**: A role hierarchy was defined for data visibility and report roll-ups.
  + Manager and customer.
* **Users**: Sample users were created and assigned the appropriate profiles and roles.
  + **manager1** → Dealer\_Manager, Role: Manager



* + **customer1** → Customer\_Profile, Role: Customer

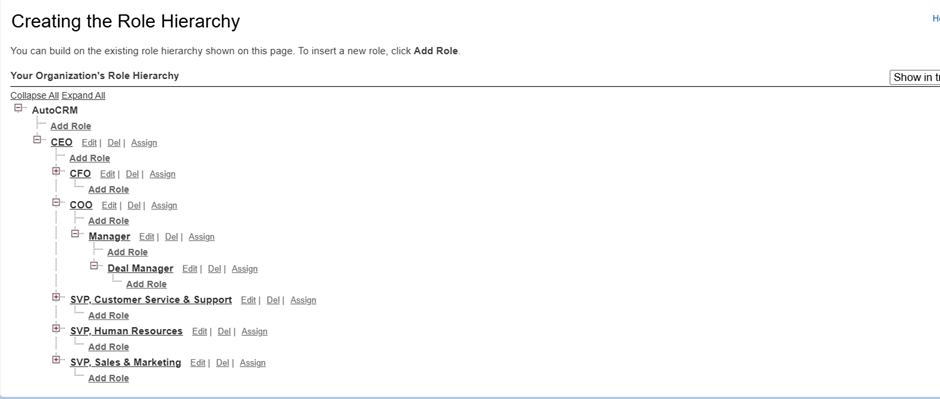




**Role Hierarchy Setup**

The hierarchy defines the data access and reporting structure within the organization.

* **Path**: Setup → Roles → Set Up Roles.
* **Top-Level Role**: CEO.
* **Manager Role**: Added as a child of the CEO.
* **Customer Role**: Added as a child under the Manager.
* **Resulting Hierarchy**: CEO → Manager → Restaurant Staff



**OWD & Sharing Rules**

* **OWD (Org-Wide Default):** These settings will be configured during Phase 3, once the necessary custom objects are created.
* **Sharing Rule:** Implementation will follow in Phase 3, after the custom objects and OWD settings are established.

**Dev Org Setup**

The development environment was prepared to facilitate project development.

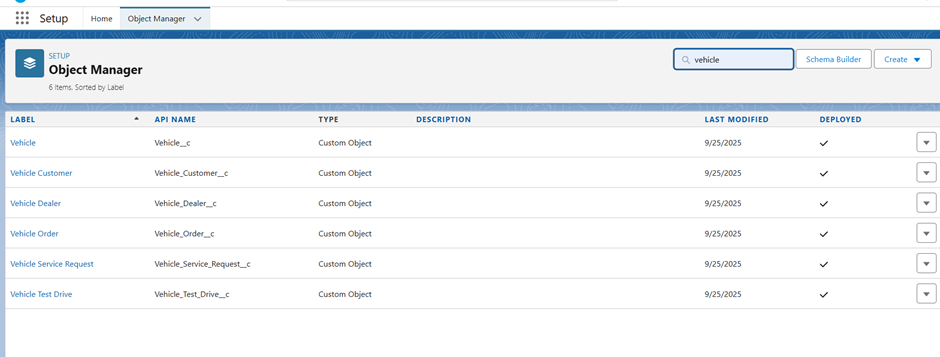
* A Salesforce Developer Edition org was provisioned for building the application.
* A GitHub repository was initialized to manage source code versions.
* VS Code and SFDX were set up to develop Apex classes and future Lightning Web Components.

# PHASE 3 – Data Modeling & Relationships

🔹 **Standard & Custom Objects** The data model utilizes both standard and custom objects to create a robust structure for the AutoCRM application.

* **Standard Objects:**
  + **Report**: Used for generating analytical insights on vehicles, dealers, orders, test drives, and service requests.
  + **Dashboard**: Used for visualizing summarized reports for tracking sales, vehicle availability, and service performance.

* **Custom Objects:** Six custom objects form the core of the application:
  1. **Vehicle\_\_c**: Stores information about vehicles available in stock.
  2. **Vehicle\_Dealer\_\_c**: Captures dealer information such as name, location, and contact details.
  3. **Vehicle\_Order\_\_c**: Manages all customer orders for vehicles.
  4. **Vehicle\_Customer\_\_c**: Holds customer records with personal details and preferences.
  5. **Vehicle\_Test\_Drive\_\_c**: Tracks customer test-drive bookings and statuses.
  6. **Vehicle\_Service\_Request\_\_c**: Logs service requests raised by customers for their vehicles.



🔹 **Fields & Relationships**

Custom fields capture the required business data, and relationships link the objects together.

**Relationships**

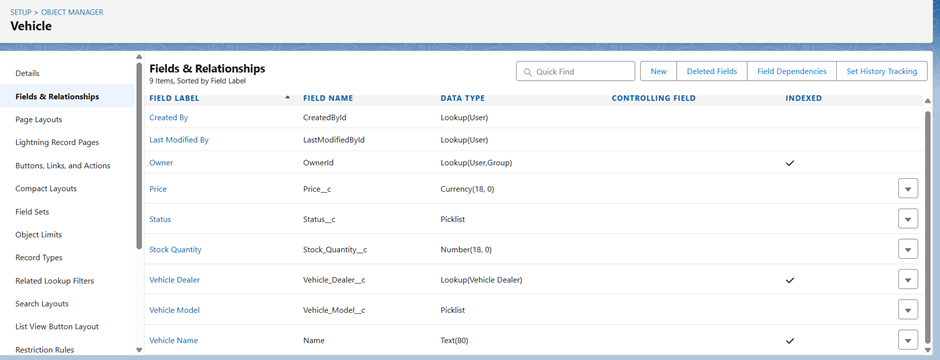
* **Vehicle\_\_c (Lookup to Vehicle\_Dealer\_\_c):** Each vehicle record is tied to a dealer, establishing dealer-vehicle association.
* **Vehicle\_Order\_\_c (Lookup to Vehicle\_\_c and Vehicle\_Customer\_\_c):** Orders are linked to both the vehicle being purchased and the customer making the purchase.
* **Vehicle\_Test\_Drive\_\_c (Lookup to Vehicle\_\_c and Vehicle\_Customer\_\_c):** Test drives are connected to both the customer and the chosen vehicle.
* **Vehicle\_Service\_Request\_\_c (Lookup to Vehicle\_\_c and Vehicle\_Customer\_\_c):** Service requests are tied to both the vehicle and the requesting customer.

Lookup relationships are chosen instead of Master-Detail so that vehicle orders, test drives, and service requests can exist independently if the parent record (vehicle/customer) is deleted or not yet created.

**Custom Fields**

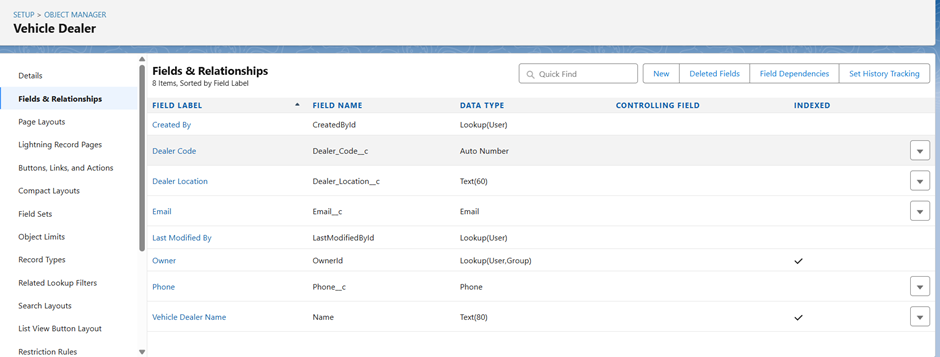
**On Vehicle\_\_c:**

* Vehicle\_Name\_\_c (Text)
* Vehicle\_Model\_\_c (Picklist: Sedan, SUV, EV, etc.)
* Stock\_Quantity\_\_c (Number)
* Price\_\_c (Currency)
* Dealer\_\_c (Lookup to Vehicle\_Dealer\_\_c)
* Status\_\_c (Picklist: Available, Out of Stock, Discontinued)



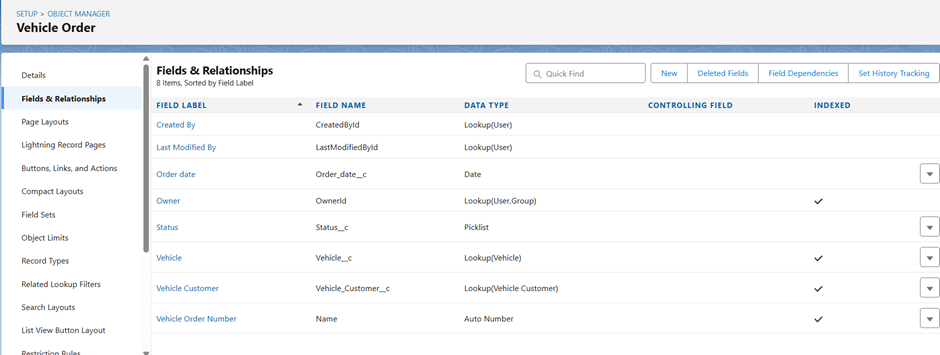
**On Vehicle\_Dealer\_\_c:**

* Dealer\_Name\_\_c (Text)
* Dealer\_Location\_\_c (Text)
* Dealer\_Code\_\_c (Auto Number)
* Phone\_\_c (Phone)
* Email\_\_c (Email)



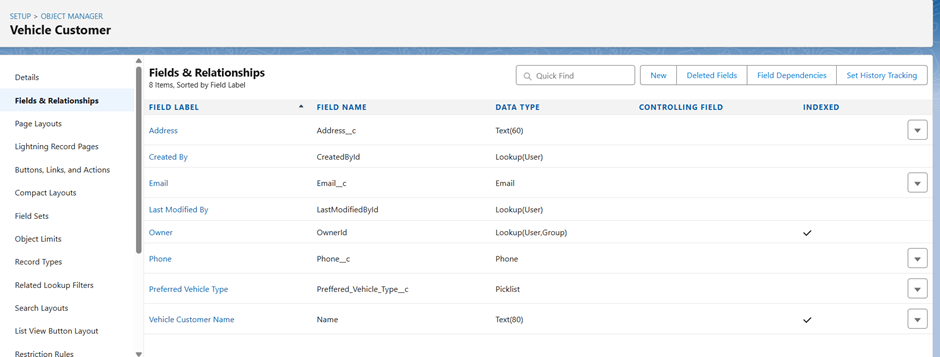
**On Vehicle\_Order\_\_c:**

* Customer\_\_c (Lookup to Vehicle\_Customer\_\_c)
* Vehicle\_\_c (Lookup to Vehicle\_\_c)
* Order\_Date\_\_c (Date)
* Status\_\_c (Picklist: Pending, Confirmed, Delivered, Canceled)



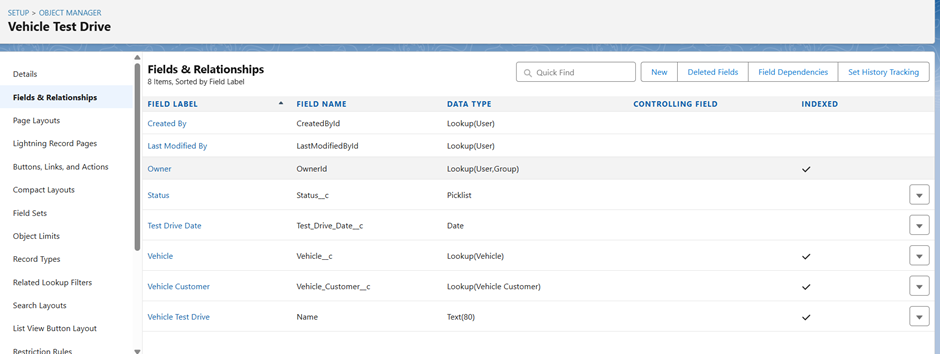
**On Vehicle\_Customer\_\_c:**

* Customer\_Name\_\_c (Text)
* Email\_\_c (Email)
* Phone\_\_c (Phone)
* Address\_\_c (Text)
* Preferred\_Vehicle\_Type\_\_c (Picklist: Sedan, SUV, EV, etc.)



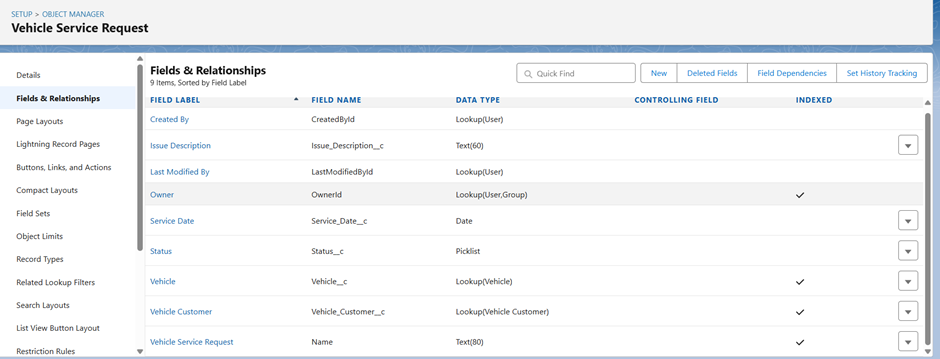
**On Vehicle\_Test\_Drive\_\_c:**

* Customer\_\_c (Lookup to Vehicle\_Customer\_\_c)
* Vehicle\_\_c (Lookup to Vehicle\_\_c)
* Test\_Drive\_Date\_\_c (Date)
* Status\_\_c (Picklist: Scheduled, Completed, Canceled)



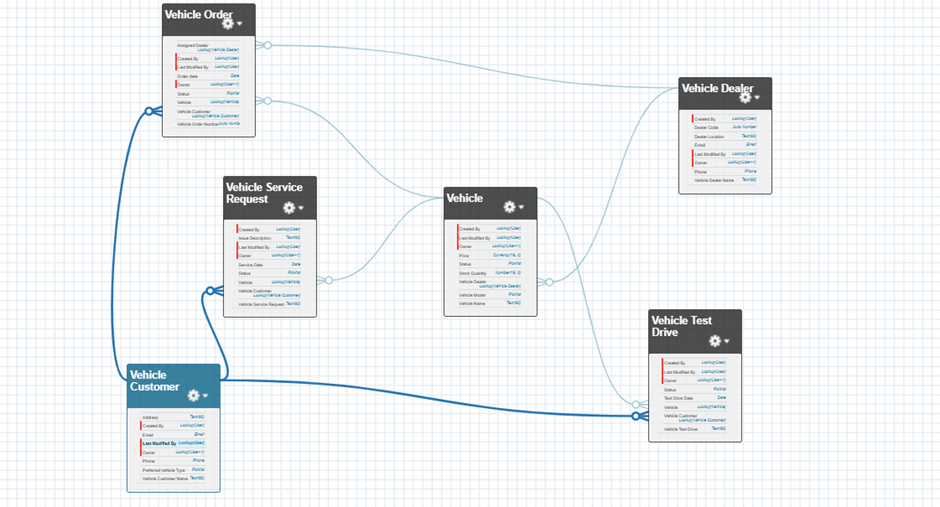
**On Vehicle\_Service\_Request\_\_c:**

* Customer\_\_c (Lookup to Vehicle\_Customer\_\_c)
* Vehicle\_\_c (Lookup to Vehicle\_\_c)
* Service\_Date\_\_c (Date)
* Issue\_Description\_\_c (Text)
* Status\_\_c (Picklist: Requested, In Progress, Completed)



🔹 **Data Architecture**

* **Schema Builder:** The Schema Builder tool will be used to visualize the complete Vehicle CRM data model, showing **Vehicle\_Customer\_\_c** at the center with lookup relationships from **Vehicle\_Order\_\_c, Vehicle\_Test\_Drive\_\_c, and Vehicle\_Service\_Request\_\_c**. Similarly, **Vehicle\_\_c** is related to both **Vehicle\_Dealer\_\_c** (dealer-vehicle association) and transactional records (orders, test drives, services).

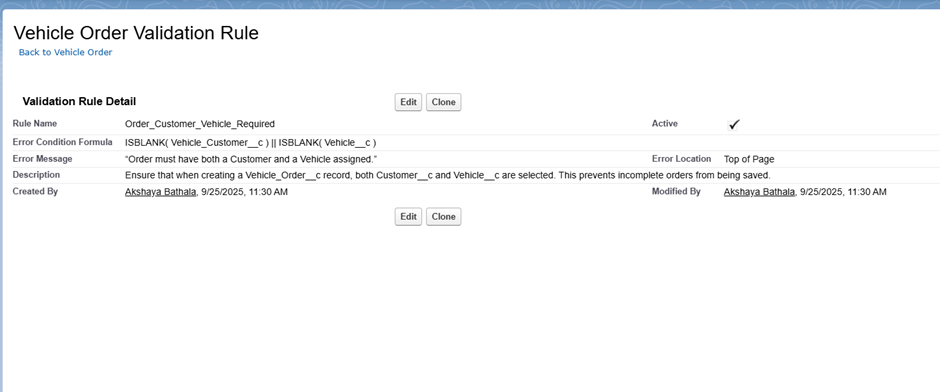


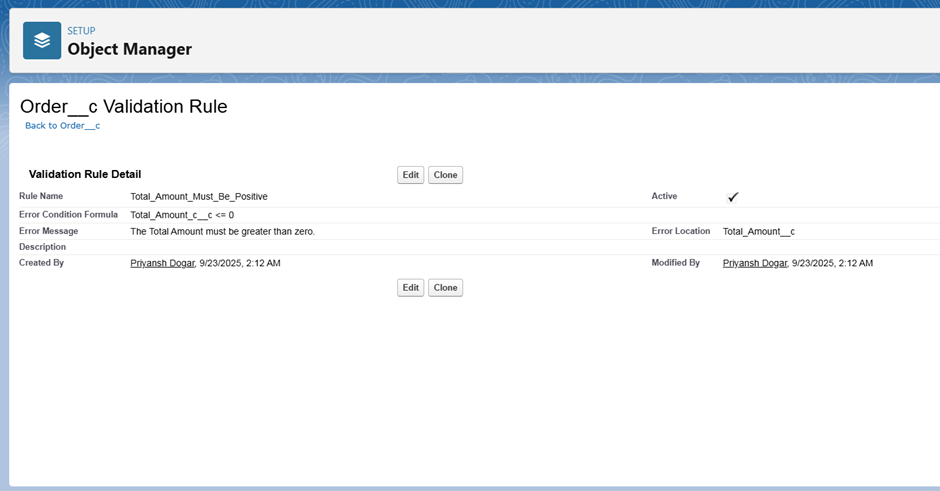
# PHASE 4 - Process Automation (Admin)

The goal of this phase is to use Salesforce's declarative tools to automate the core business logic, enforce data quality, and handle notifications.

**♦️ Validation Rules**

**Validation Rule:** Order\_Customer\_Vehicle\_Required  
 **Object:** Vehicle\_Order\_\_c  
 **Purpose:** Ensure both Customer and Vehicle are selected  
 **Formula:** ISBLANK( Vehicle\_Customer\_\_c ) || ISBLANK( Vehicle\_\_c )  
 **Error Message:** “Order must have both a Customer and a Vehicle assigned.”  
 **Field:** Customer\_\_c / Vehicle\_\_c  
 **Steps:** Object Manager → Vehicle\_Order\_\_c → Validation Rules → New → Enter formula → Error Message → Field → Save





**♦️ Flow Builder (Record-Triggered Flow)**

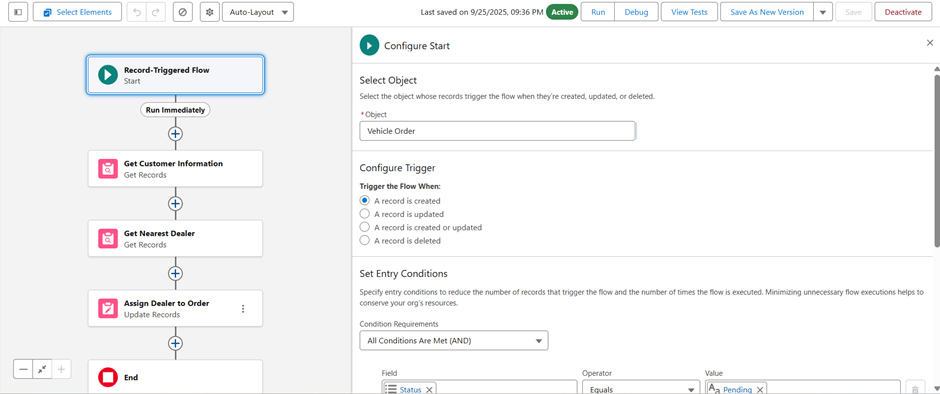
**Objective**

Automatically assign the nearest dealer to a new **Vehicle\_Order\_\_c** record based on the customer’s address.

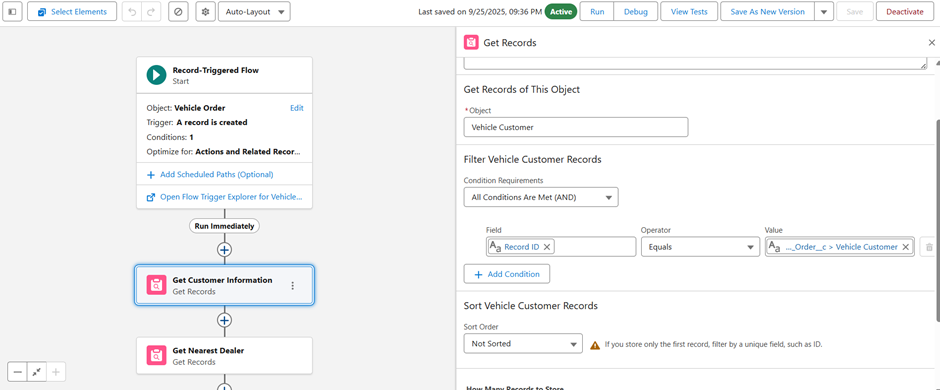
**Flow Type and Trigger Configuration**

* **Flow Type:** Record-Triggered Flow
* **Object:** Vehicle\_Order\_\_c
* **Trigger:** When a record is **created**
* **Run Flow:** **After the record is saved** (to allow related record lookup and update)

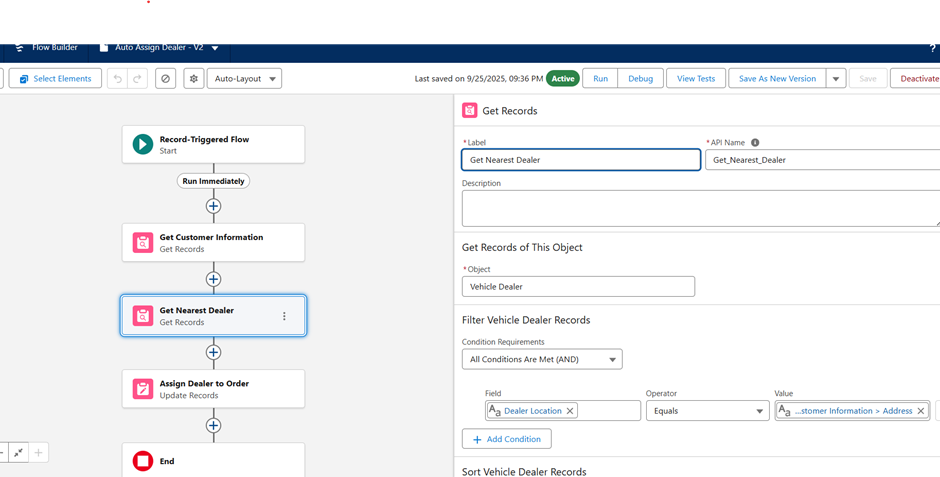
**Flow Logic / Steps**

****

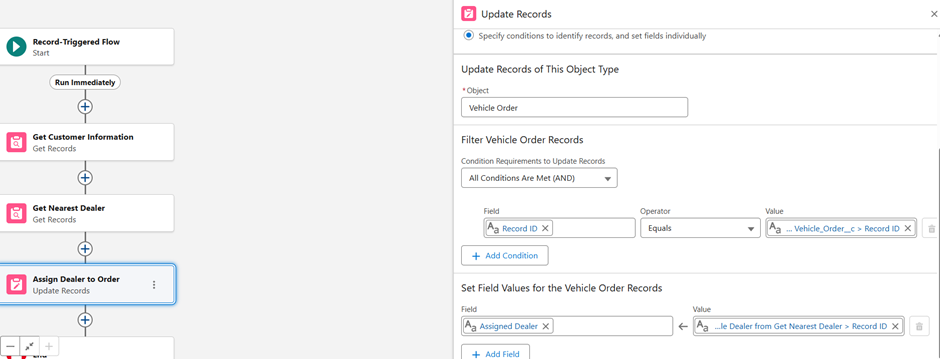
1. **Get Customer Information**
   * **Element:** Get Records
   * **Object:** Vehicle\_Customer\_\_c
   * Get the customer record whose **Id matches the Customer\_\_c field on the triggering Vehicle\_Order\_\_c**.
   * **Output:** Store customer address for further use



1. **Get Nearest Dealer**
   * **Element:** Get Records
   * **Object:** Vehicle\_Dealer\_\_c
   * Get the dealer whose **Dealer\_Location\_\_c matches the Address field of the customer linked to this Vehicle\_Order\_\_c**.
   * **Sort:** Optionally by distance or priority if available
   * **Output:** Store the nearest dealer record



1. **Assign Dealer to Order**
   * **Element:** Update Records
   * **Record to Update:** Triggering Vehicle\_Order\_\_c record
   * Update the triggering **Vehicle\_Order\_\_c** record by setting **Dealer\_\_c = Id of the nearest dealer**

****

**♦️ Approval Process**

An approval process automates how records are approved. **While not part of the initial build, a potential future use case has been identified**:

· **Objective:** Require a Dealer Manager’s approval for any Vehicle\_Order\_\_c with a Total\_Amount\_\_c exceeding a set value (e.g., 50,000 INR).

· **Process:** When an order meets the criteria, it is automatically submitted to the Dealer Manager for approval or rejection, and the order’s Status\_\_c is updated accordingly.

**♦️ Workflow Rules & Process Builder**

These are legacy automation tools in Salesforce. For this project, all new automation, including the loyalty points calculation, was built using **Flow Builder** to align with current Salesforce best practices.

# Phase 5: Apex Programming (Developer)

**Apex programming was implemented to enforce business rules, manage inventory, and ensure data consistency within the AutoCRM Hub. The following components were used:**

### **1. Objects Used**

* **Vehicle\_Order\_\_c**
  + **Acts as the main transactional object for placing and managing vehicle orders.**
  + **Trigger runs on this object to control order placement and update stock quantities.**
* **Vehicle\_\_c**
  + **Holds details like stock quantity, model, price, and status for each vehicle.**
  + **Queried in Apex to validate stock availability and to decrement stock after a confirmed order.**
* **Relationships**
  + **Vehicle\_\_c is linked to Vehicle\_Order\_\_c through a lookup relationship.**
  + **This allows fetching related vehicle stock info using the Vehicle lookup field on each order.**

### **2. Why These Objects Are Used**

* **Vehicle\_Order\_\_c is the triggering object, representing real-time business events (order creation/update).**
* **Vehicle\_\_c is the reference object, providing stock data required to validate and update inventory.**
* **Linking both ensures the system prevents invalid transactions (e.g., out-of-stock orders) and maintains up-to-date vehicle availability without manual work.**
* **This mirrors a real car dealership workflow where placing an order automatically affects stock.**

### **3. SOQL & Functions Used**

**SOQL Query:  
  
 [SELECT Id, Stock\_Quantity\_\_c FROM Vehicle\_\_c WHERE Id IN :vehicleIds]**

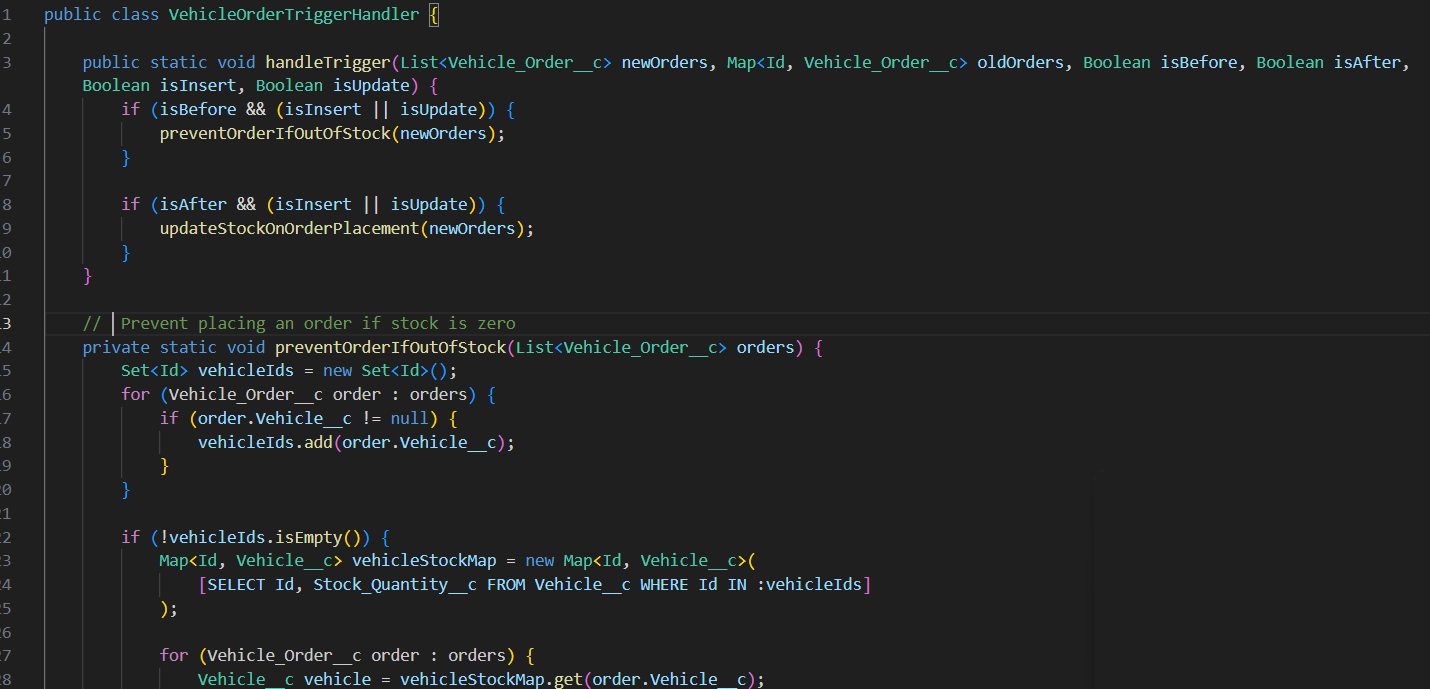
* **Purpose of the Query:**
  + **Retrieve stock information for all vehicles linked to new or updated orders.**
  + **The query runs once per trigger context, ensuring bulk-safe operations.**
* **Collections Used:**
  + **Set<Id> gathers all unique Vehicle IDs from incoming orders.**
  + **Map<Id, Vehicle\_\_c> stores the queried records for quick lookup without multiple SOQL calls.**
* **Efficiency: Prevents SOQL inside loops and supports processing multiple records simultaneously.**

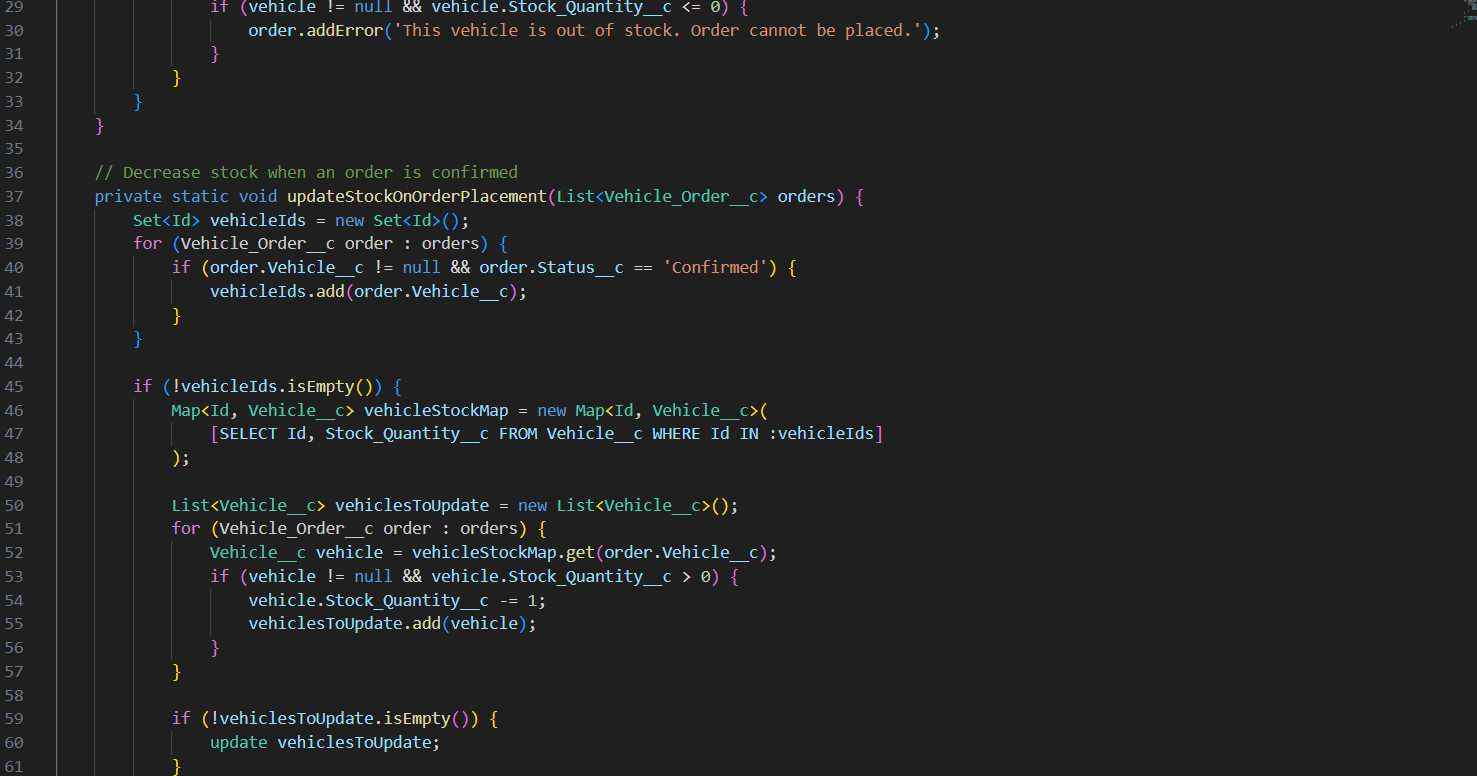
### **4. Triggers: SObject & Events**

**SObject: Vehicle\_Order\_\_c**

### **VehicleOrderTriggerHandler**

* **An Apex class that contains the core business logic for vehicle order validation and stock updates.**
* **Prevents orders from being placed if the related vehicle’s stock is zero using addError() in before triggers.**
* **Automatically decrements stock for confirmed orders in after triggers.**
* **Uses Sets and Maps with SOQL to handle multiple records in bulk safely.**

****



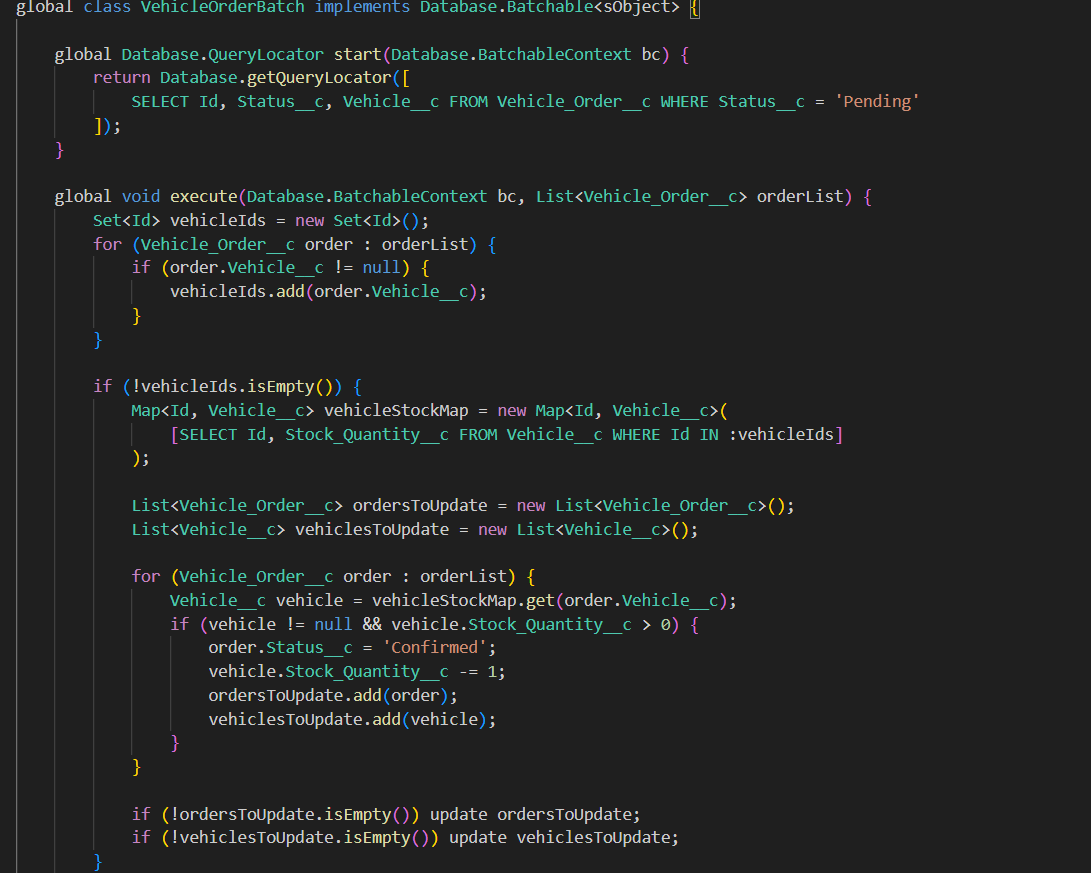
### **VehicleOrderTrigger**

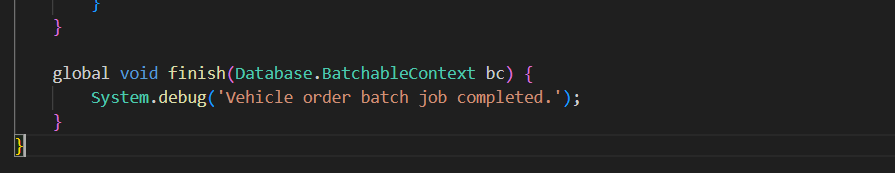
* A trigger on **Vehicle\_Order\_\_c** that listens to **before insert/update** and **after insert/update** events.
* It doesn’t contain business logic directly; instead, it delegates all processing to the **Trigger Handler class**.
* Ensures a clean, scalable structure following the **Trigger Handler Pattern**.



### **VehicleOrderBatch**

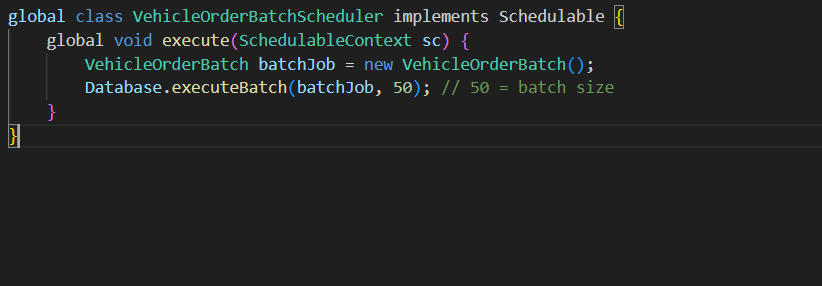
* A **Batch Apex class** that processes all pending vehicle orders in bulk.
* Runs a SOQL query to fetch orders with Status\_\_c = 'Pending', checks stock, confirms eligible orders, and updates stock levels.
* Useful for processing large volumes of data efficiently and ensuring no pending orders are missed.





### **VehicleOrderBatchScheduler**

* A **Scheduled Apex class** that automatically runs the batch job at defined intervals (e.g., daily).
* Ensures continuous background processing of pending orders without manual intervention.
* Uses Database.executeBatch() inside the execute() method to schedule the batch with a fixed batch size.



### **5. Batch Apex**

* Your VehicleOrderBatch class implements the Database.Batchable interface.
* When you call Database.executeBatch(new VehicleOrderBatch()), Salesforce **places the job in the Apex job queue** and runs it asynchronously in the background.
* This allows the platform to handle **large data volumes** (up to 50 million records) without hitting governor limits of synchronous processing.
* It processes data in manageable “chunks” (batches), making it scalable.

### **6. Scheduled Apex**

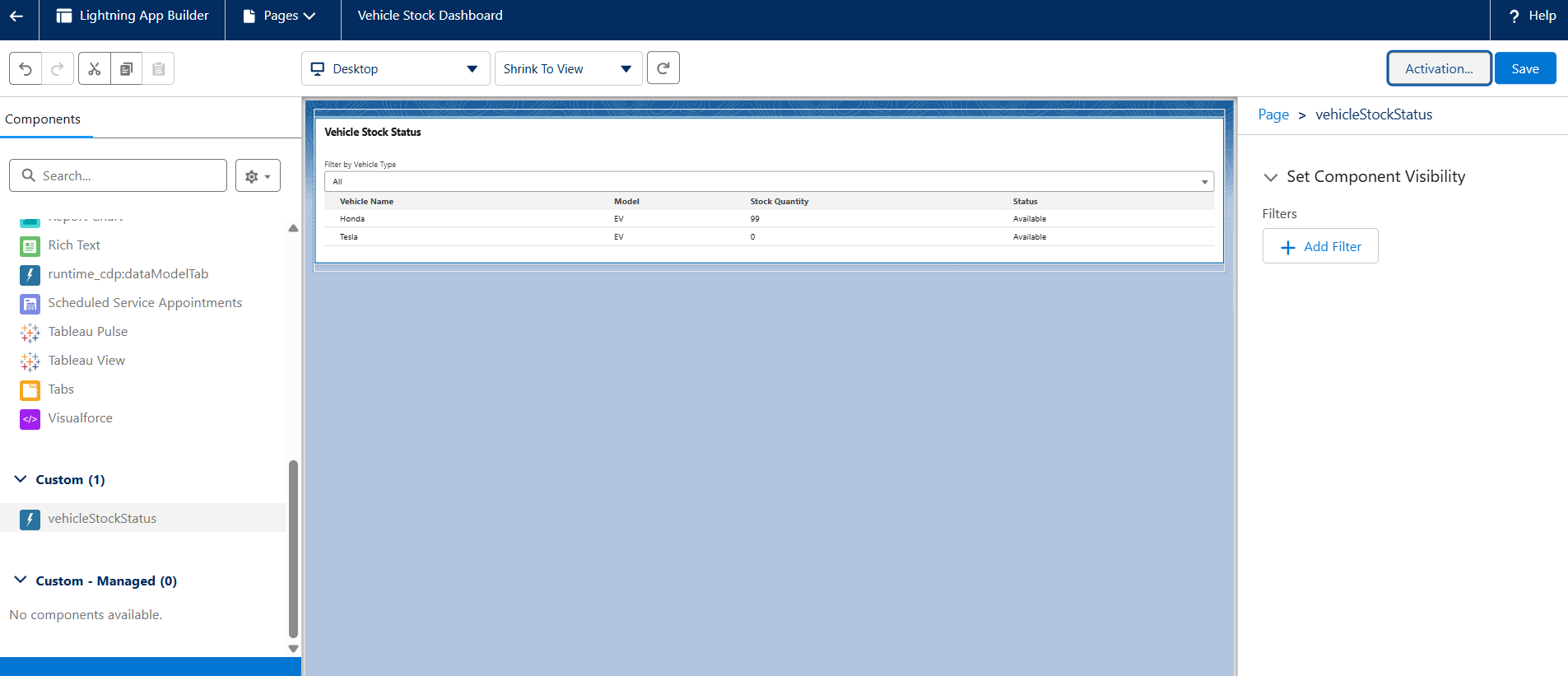
* Your VehicleOrderBatchScheduler uses the Schedulable interface and System.schedule() method.
* Scheduled Apex jobs **run at specific times automatically**, independent of user actions.
* When triggered, the scheduler **invokes the batch job asynchronously**, so it runs without blocking the UI or trigger flow.

# 

# PHASE 6: User Interface Development

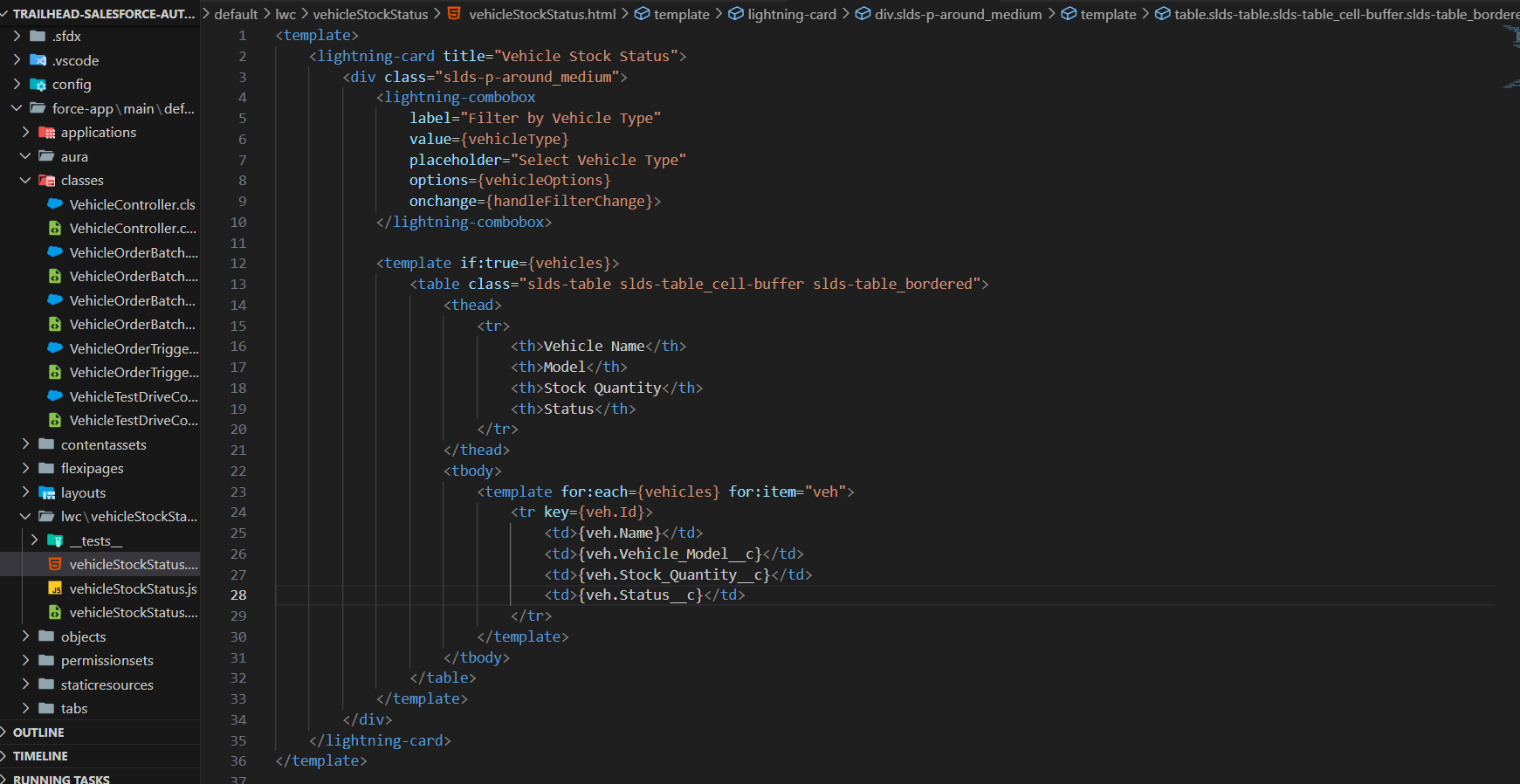
## 1. Lightning App Builder

* **Use Case:** The Lightning App Builder allows admins and developers to design custom pages in Salesforce using drag-and-drop components.
* **Implementation in Project:** A custom **App Page** called Vehicle Stock Dashboard was created using Lightning App Builder. This page hosts the LWC component vehicleStockStatus to display vehicle inventory in real time.

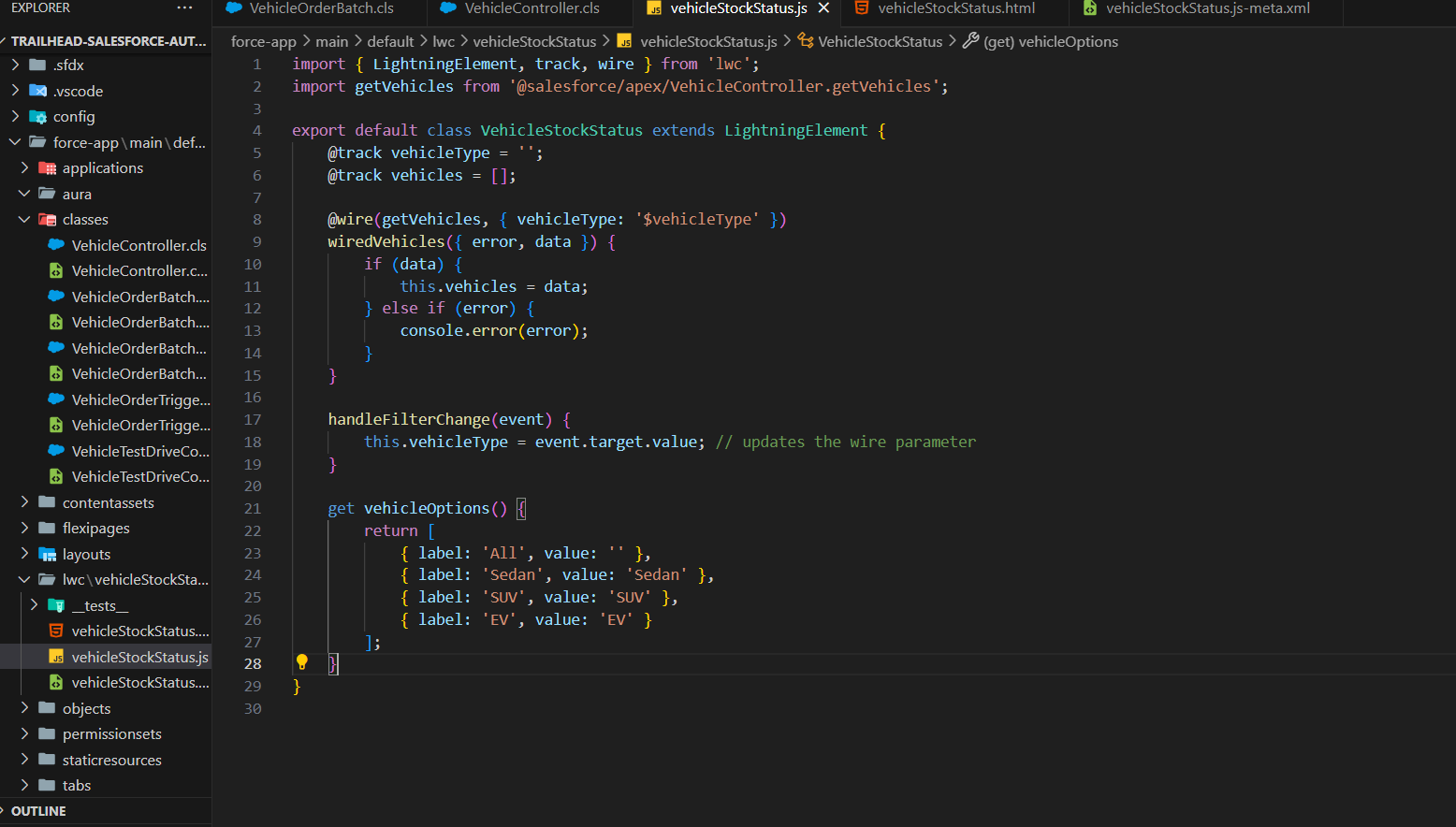
****

## 2. Lightning Web Component (LWC) – vehicleStockStatus

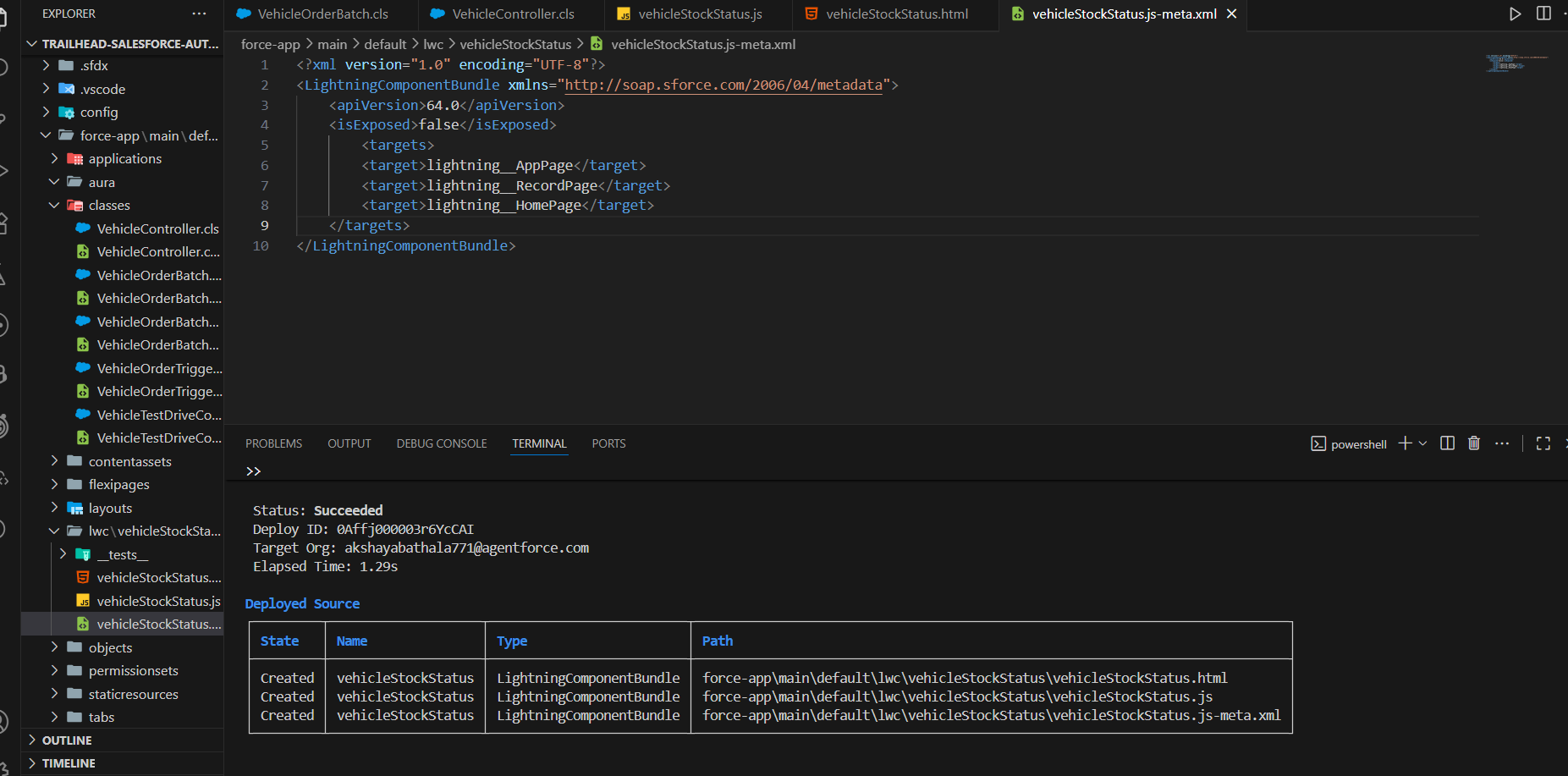
* **Use Case:** The LWC component dynamically fetches and displays vehicle inventory based on the type of vehicle (Sedan, SUV, EV). It enhances the user experience with a modern, responsive interface.
* **What the Code Does:**
  1. **HTML (vehicleStockStatus.html)**:  
     + Displays a **lightning-card** container.
     + Includes a **combobox** for selecting vehicle types.
     + Renders a **table** showing Vehicle Name, Model, Stock Quantity, and Status.



* 1. **JavaScript (vehicleStockStatus.js)**:  
     + Tracks the selected vehicle type using a reactive variable.
     + Uses the @wire decorator to call the Apex method getVehicles whenever the selected type changes.
     + Dynamically updates the table based on the selected vehicle type.

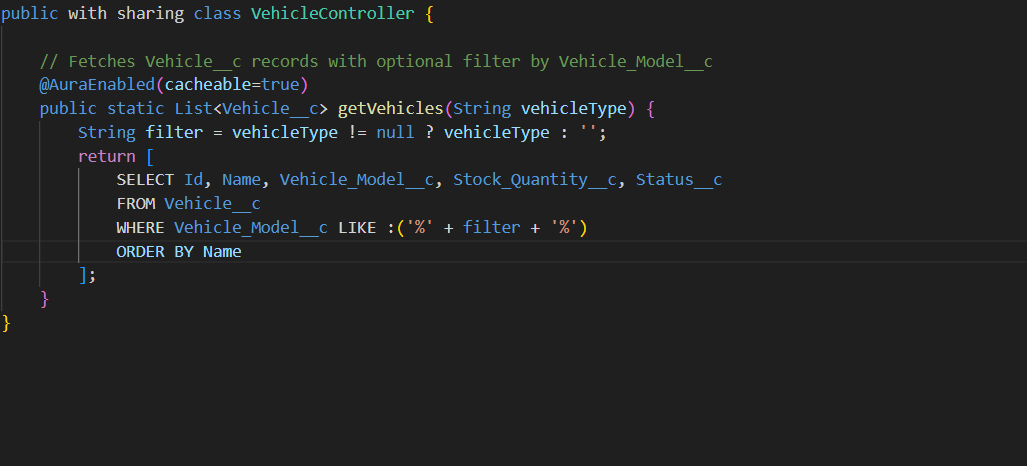


* 1. **Metadata (vehicleStockStatus.js-meta.xml)**:  
     + Exposes the component to Lightning pages: App, Home, and Record Pages.



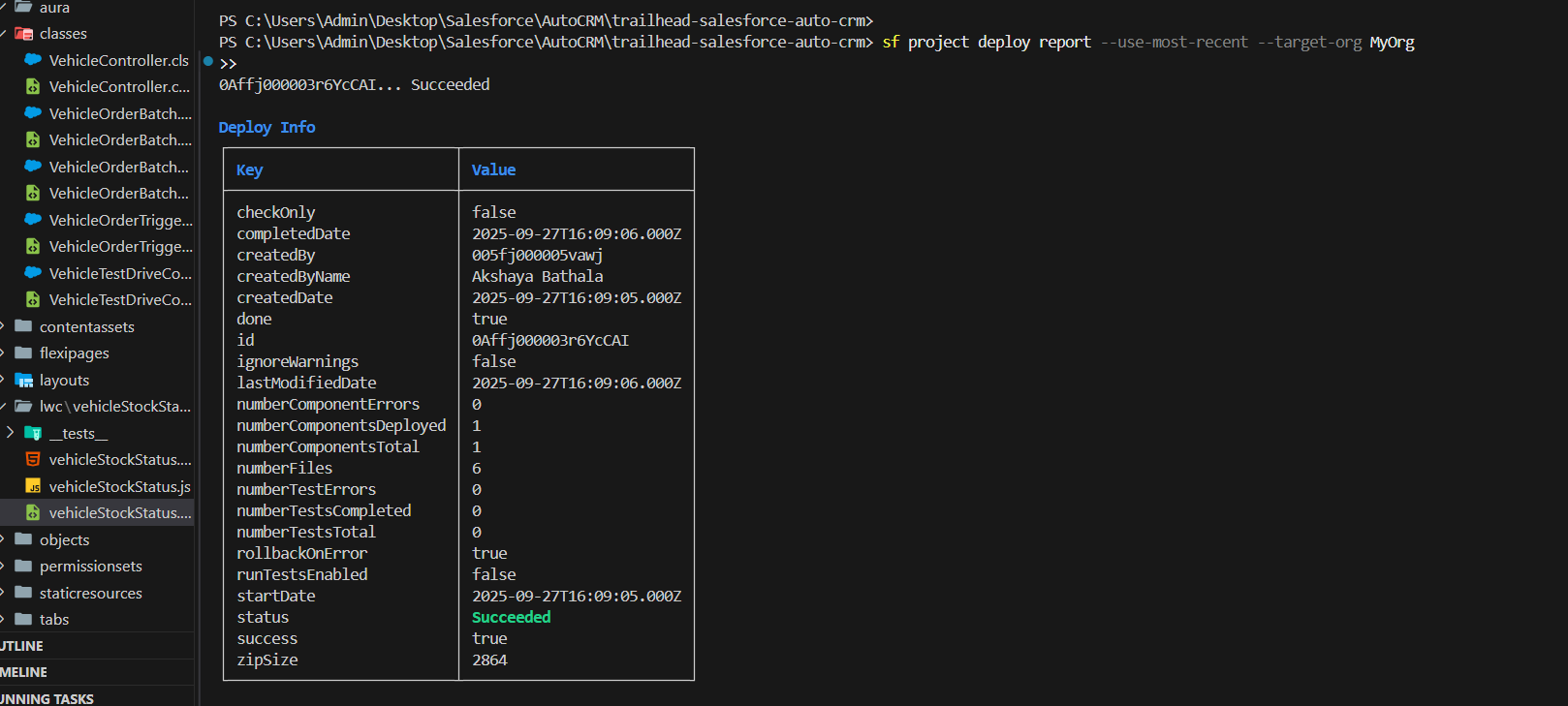
## 3. Apex Controller – VehicleController

* **Use Case:** Apex controller acts as a bridge between Salesforce data and the LWC, allowing real-time retrieval of Vehicle\_\_c records.
* **What the Code Does:**
  1. Defines a method getVehicles(vehicleType) which queries **Vehicle\_\_c records** from Salesforce.
  2. Filters results based on the selected **Vehicle\_Model\_\_c** (Sedan, SUV, EV).
  3. Returns a list of vehicles with relevant fields: Vehicle Name, Model, Stock Quantity, and Status.
  4. The method is annotated with @AuraEnabled(cacheable=true) to make it accessible in LWC with client-side caching.



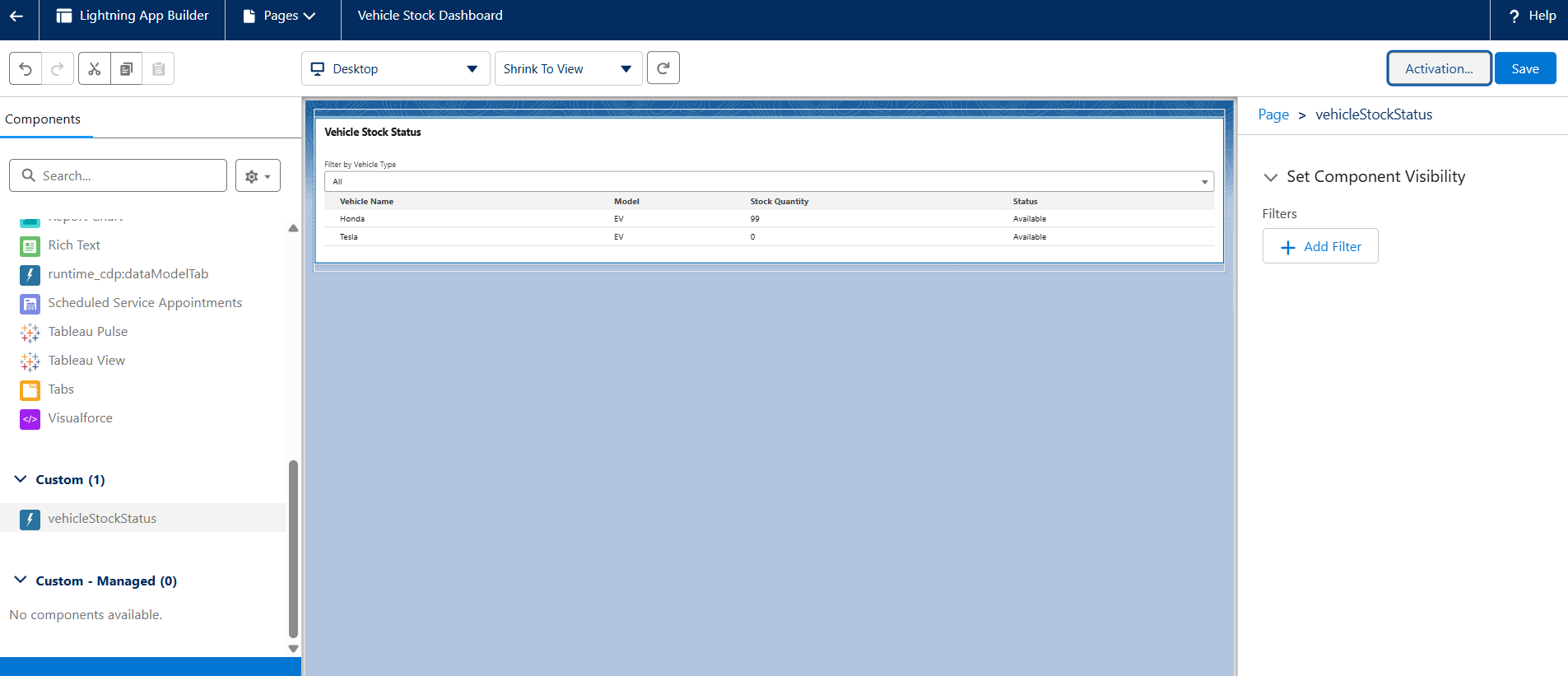
## 4. Component Deployment and Activation

* **Use Case:** After developing the LWC and Apex class, both were deployed to the Salesforce org using **sf CLI**.
* **What was done:**
  1. Logged into the Salesforce org using sf login org.
  2. Deployed the Apex class VehicleController.cls.
  3. Deployed the LWC folder vehicleStockStatus.
  4. Added the LWC to a new **App Page** via Lightning App Builder and activated it for testing.



## 5. Dynamic Filtering and Data Display

* **Use Case:** Users can filter vehicles by type to quickly see available stock.
* **Implementation in Code:**
  + **Combobox selection** triggers handleFilterChange() in JS.
  + The @wire method automatically calls getVehicles() in Apex with the selected filter.
  + Table updates dynamically to show only relevant vehicles.

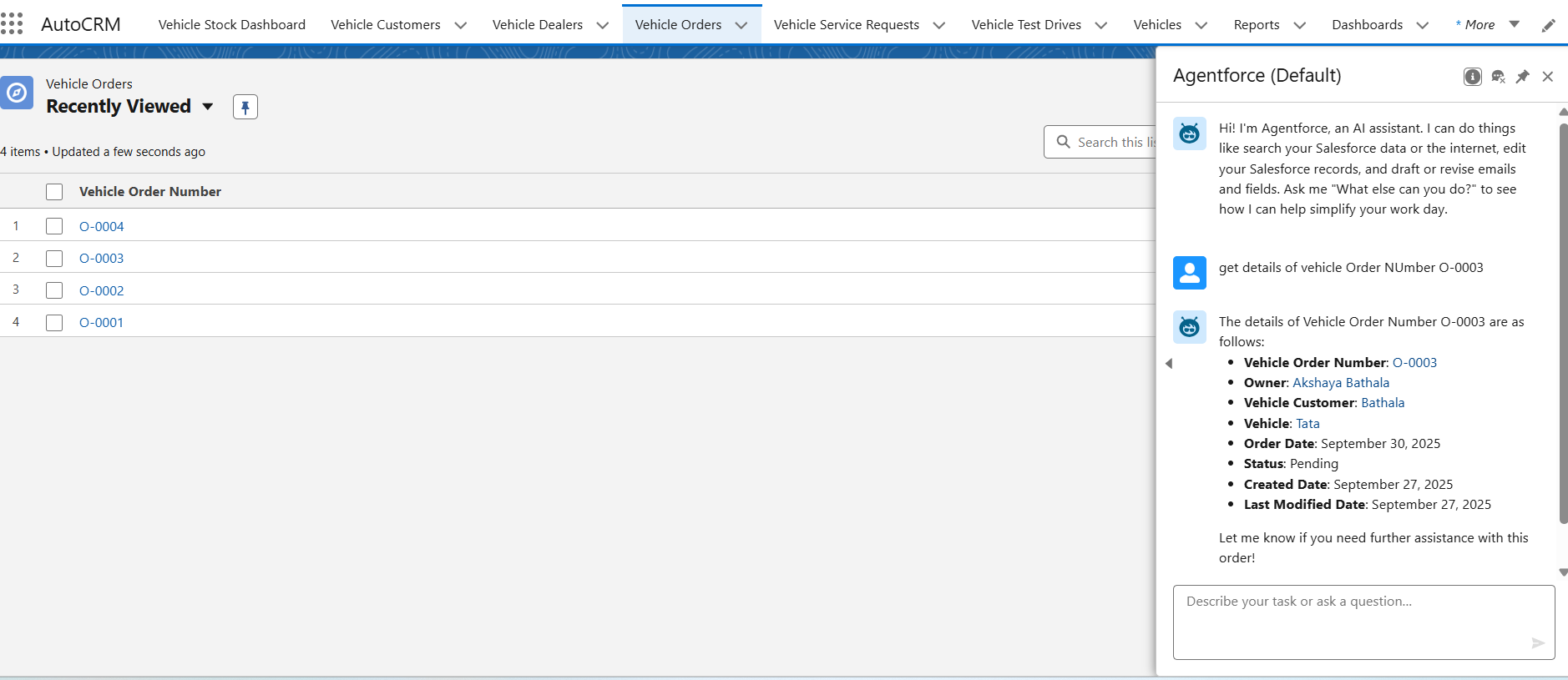


# ****6.AgentForce in AutoCRM****

A smart read-only assistant that provides **instant access to Salesforce data** across vehicles, customers, orders, and service requests.

## ****Capabilities****

* **Instant Data Retrieval:** Access records from custom and standard objects without navigating Salesforce.
* **Natural Language Queries:** Ask questions in plain language, e.g., “Show available SUVs at Dealer X.”
* **Custom Object Support:** Works with Vehicle\_\_c, Vehicle\_Customer\_\_c, Vehicle\_Order\_\_c, Vehicle\_Test\_Drive\_\_c, and Vehicle\_Service\_Request\_\_c.
* **Secure Read-Only Access:** Fetches information safely; does not create, update, or delete records.



## ****Benefits****

* **Time Efficient:** Immediate insights without manual search.
* **Accurate & Reliable:** Pulls official Salesforce data.
* **Decision Support:** Helps track vehicles, orders, and customer interactions safely.

## 7. Benefits and User Experience

* Provides **real-time visibility** into vehicle stock to prevent overbooking or ordering out-of-stock vehicles.
* Enhances usability with **interactive UI** and **dynamic filtering**.
* Can be extended to **Record Pages** or **Home Pages** in the future.
* Supports **clean and maintainable code** with separation of UI (LWC) and business logic (Apex).

# PHASE 7: Integration & External Access

**Goal:** Enable secure and scalable integration with external systems while allowing controlled access to Salesforce data.

## OAuth & Authentication

**What I Did:** Created a **Connected App** to provide secure access for external systems.

**Steps Implemented:**

1. Setup → App Manager → **New Connected App**
2. Name: AutoCRM\_External\_API
3. Enabled **OAuth Settings**, Callback URL: https://login.salesforce.com/services/oauth2/callback
4. OAuth Scope: Full access (full)
5. Saved to get **Consumer Key and Secret**

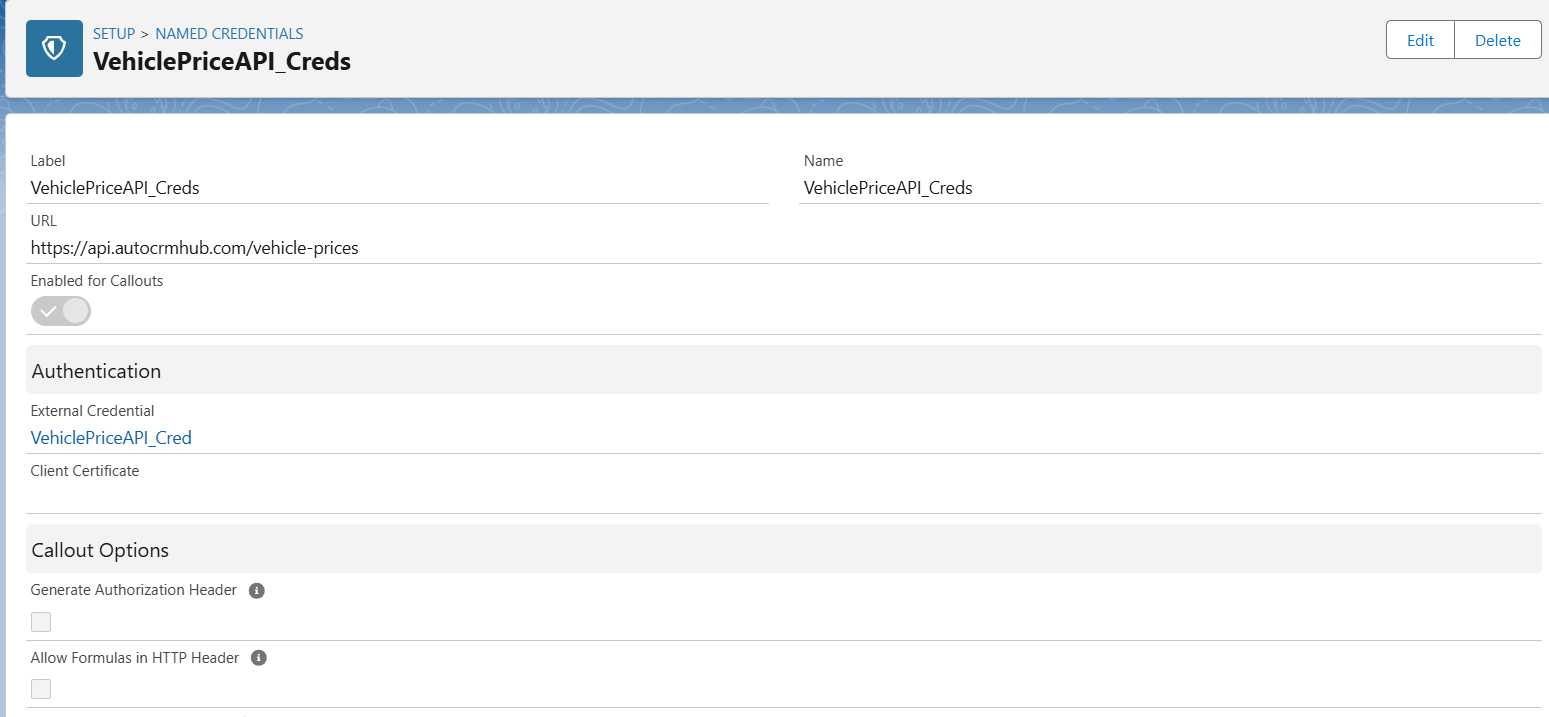
**Purpose:** Allows external systems to securely obtain tokens for API access.

## Named Credentials & Remote Site Settings

**What I Did:** Configured **Named Credential** and **Remote Site Setting** to securely call external services from Apex.

**Steps Implemented:**

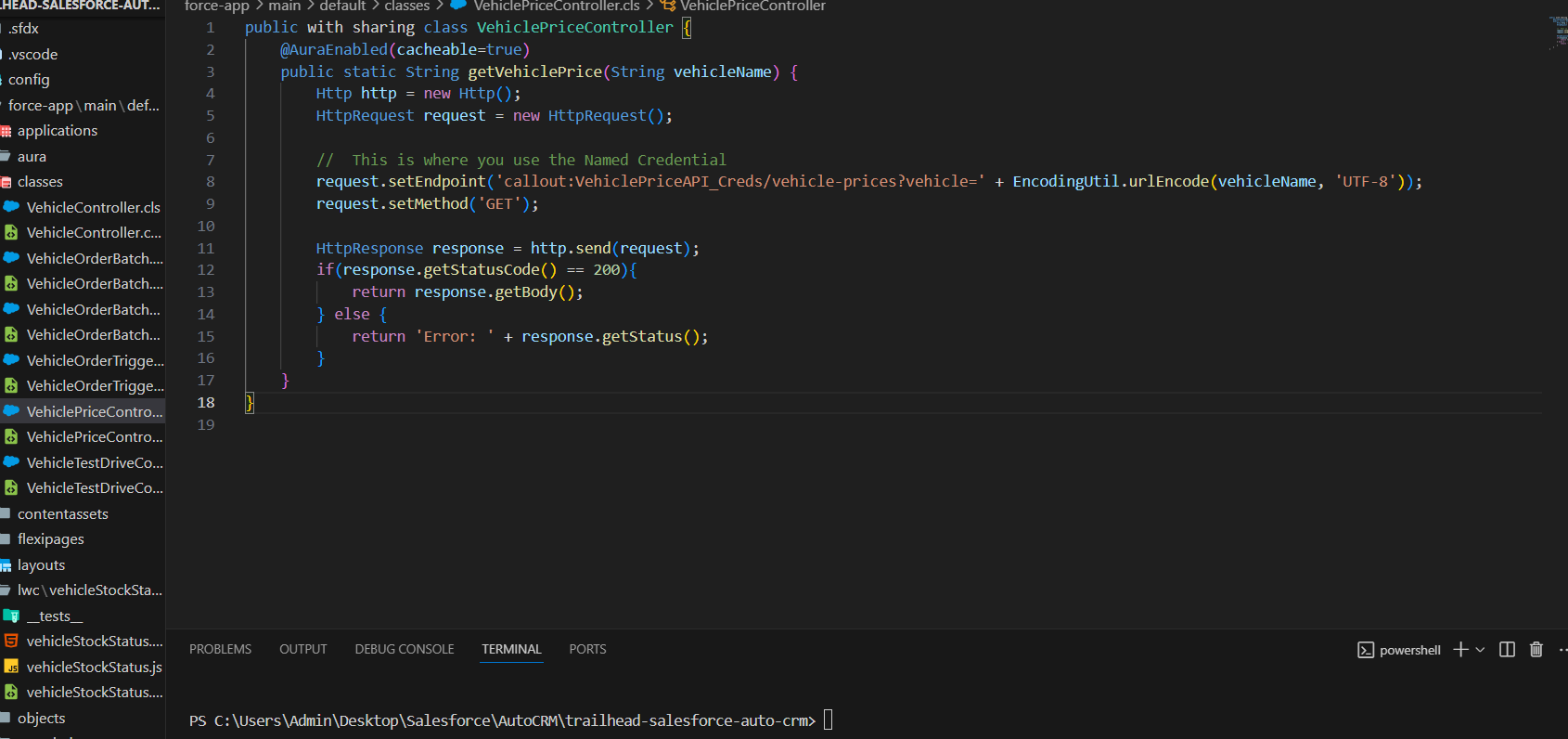
1. Created an **External Credential**: AutoCRM\_API\_Cred
2. Created a **Named Credential**: AutoCRM\_API\_NC linked to AutoCRM\_API\_Cred
3. Created a **Remote Site Setting**: AutoCRM\_Remote\_Site with external API URL



**Purpose:** Ensures secure, maintainable Apex callouts without hardcoding credentials.

## Apex Callouts / Web Services

**What I Did:** Created Apex class **VehiclePriceController.cls** to demonstrate fetching data from external services using Named Credentials.



**Purpose:** Shows secure callouts from Salesforce to external APIs, enabling real-time integration for vehicle prices or dealer info.

# PHASE 8: Data Import & Display

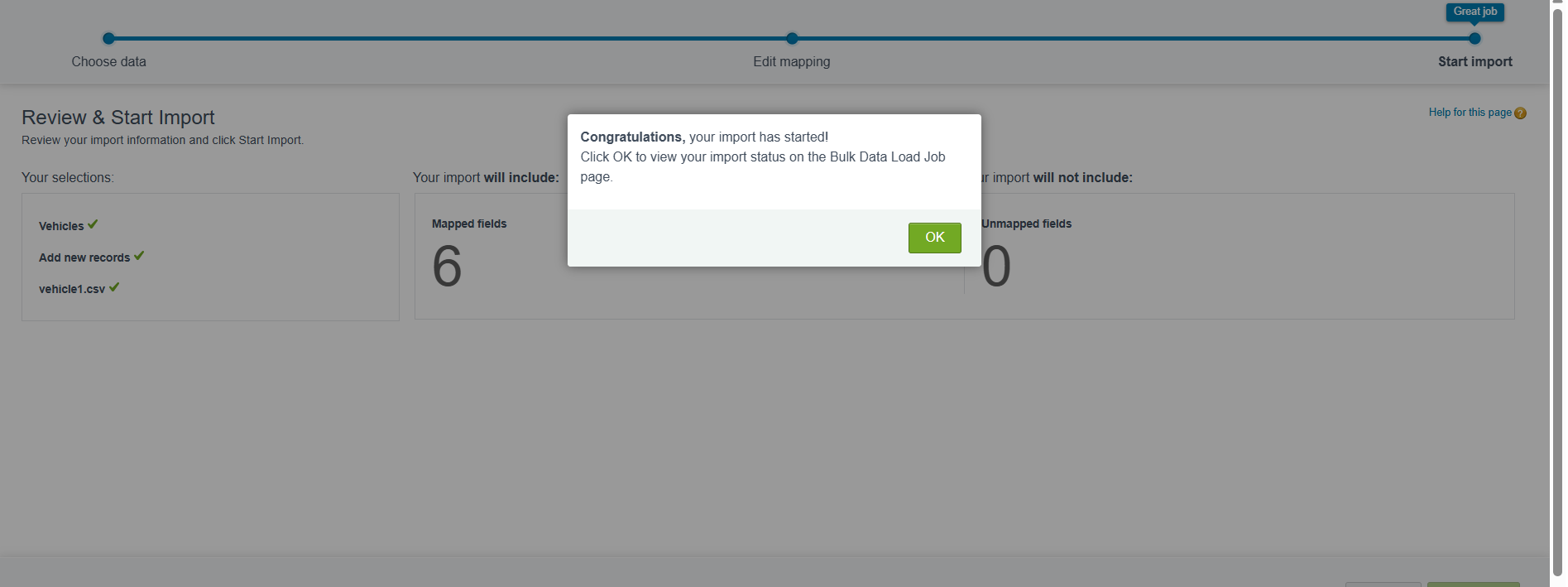
**Goal:** The goal of this phase was to import vehicle data from an external source (Excel/CSV) into Salesforce and display it correctly in the application. This ensures that all vehicle inventory is recorded in the system and can be referenced by other processes such as orders, stock checks, and dashboards.

## What I Did

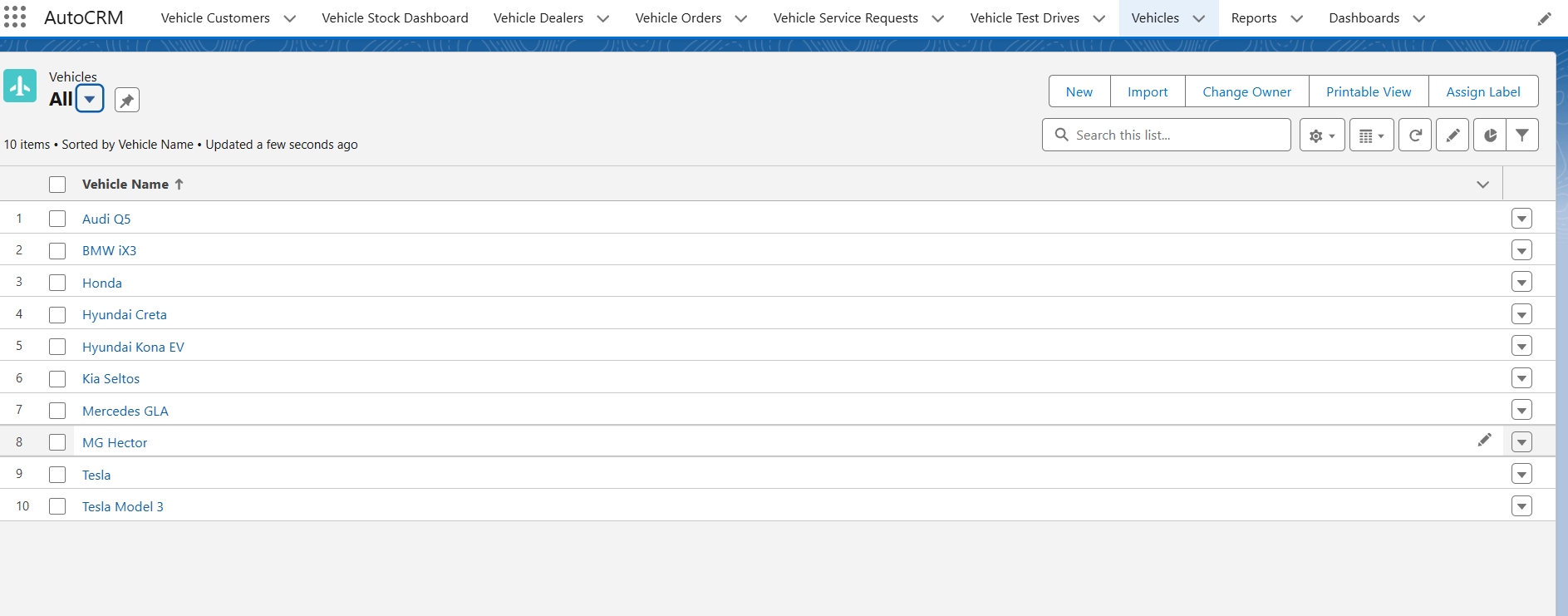
* Prepared a dataset with 10 vehicle records including **Vehicle Name, Model, Stock Quantity, Price, Dealer, and Status**.
* Used **Data Import Wizard** in Salesforce to import the dataset into the custom object **Vehicle\_\_c**.
* Mapped the **Dealer\_\_c lookup field** to existing **Vehicle\_Dealer\_\_c** records to maintain relational integrity.
* Verified that all records were imported successfully and displayed correctly in the **Vehicles tab**.

## Steps Implemented

1. **Dataset Preparation:**
   * Created a CSV file with 10 rows and 6 columns for **Vehicle\_\_c**.
   * Included fields: Vehicle\_Name\_\_c, Vehicle\_Model\_\_c, Stock\_Quantity\_\_c, Price\_\_c, Dealer\_\_c, Status\_\_c.
2. **Data Import Wizard:**
   * Navigated to **Setup → Data → Data Import Wizard → Launch Wizard**.
   * Selected the custom object **Vehicle\_\_c** for import.
   * Uploaded the CSV file.
   * Mapped CSV columns to Salesforce fields, ensuring **Dealer\_\_c** matched existing dealer records.
   * Started the import process and checked for successful import confirmation.



1. **Verification:**
   * Accessed the **Vehicles tab** in Salesforce.
   * Confirmed that all 10 records appeared correctly with linked dealers and proper stock/status values.



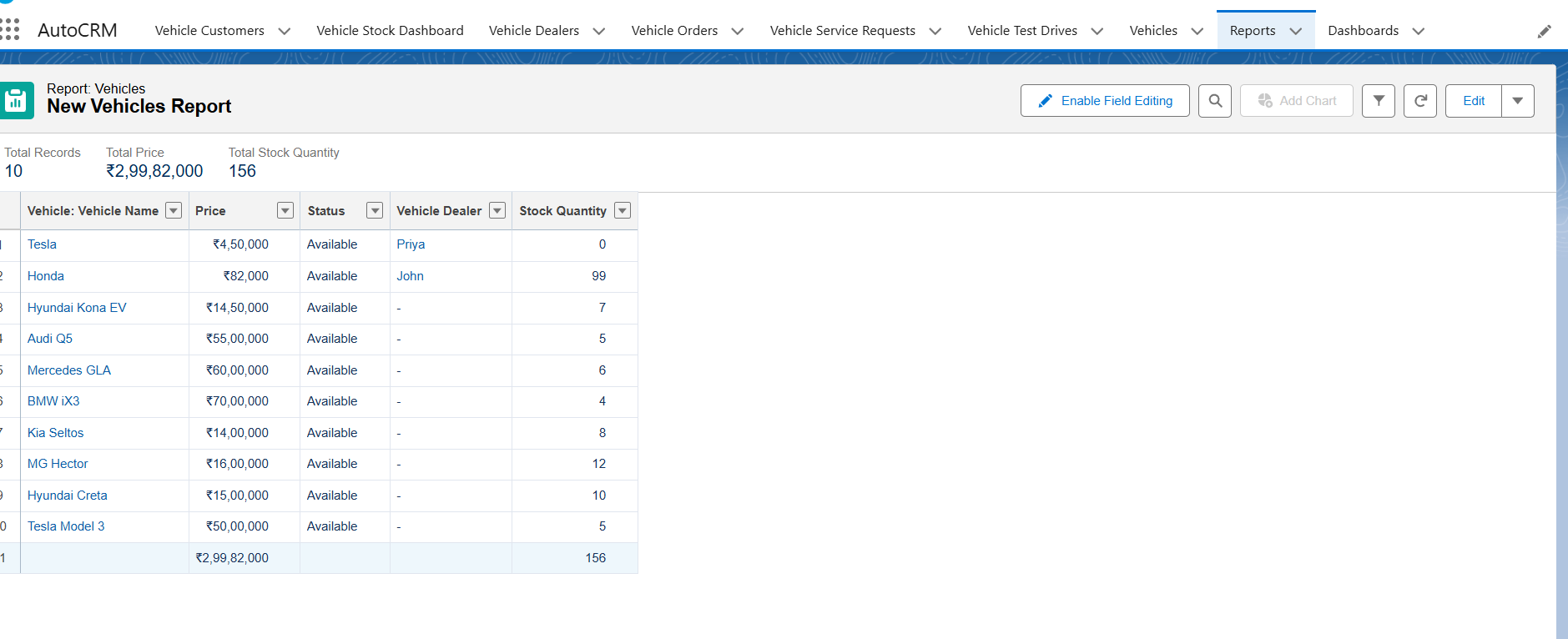
# PHASE 9: Reports & Dashboards

**Goal:** The goal of this phase was to visualize vehicle inventory and order status data in AutoCRM Hub. Creating reports and dashboards provides a clear overview of stock availability, order progress, and helps in making data-driven decisions.

## Vehicle Inventory Report

* **Object:** Vehicle\_\_c
* **Fields Included:** Vehicle Name, Vehicle Model, Stock Quantity, Dealer, Status
* **Filters Applied:** Status = Available / Out of Stock
* **Chart Type:** Bar Chart (displaying Stock Quantity by Vehicle Model)

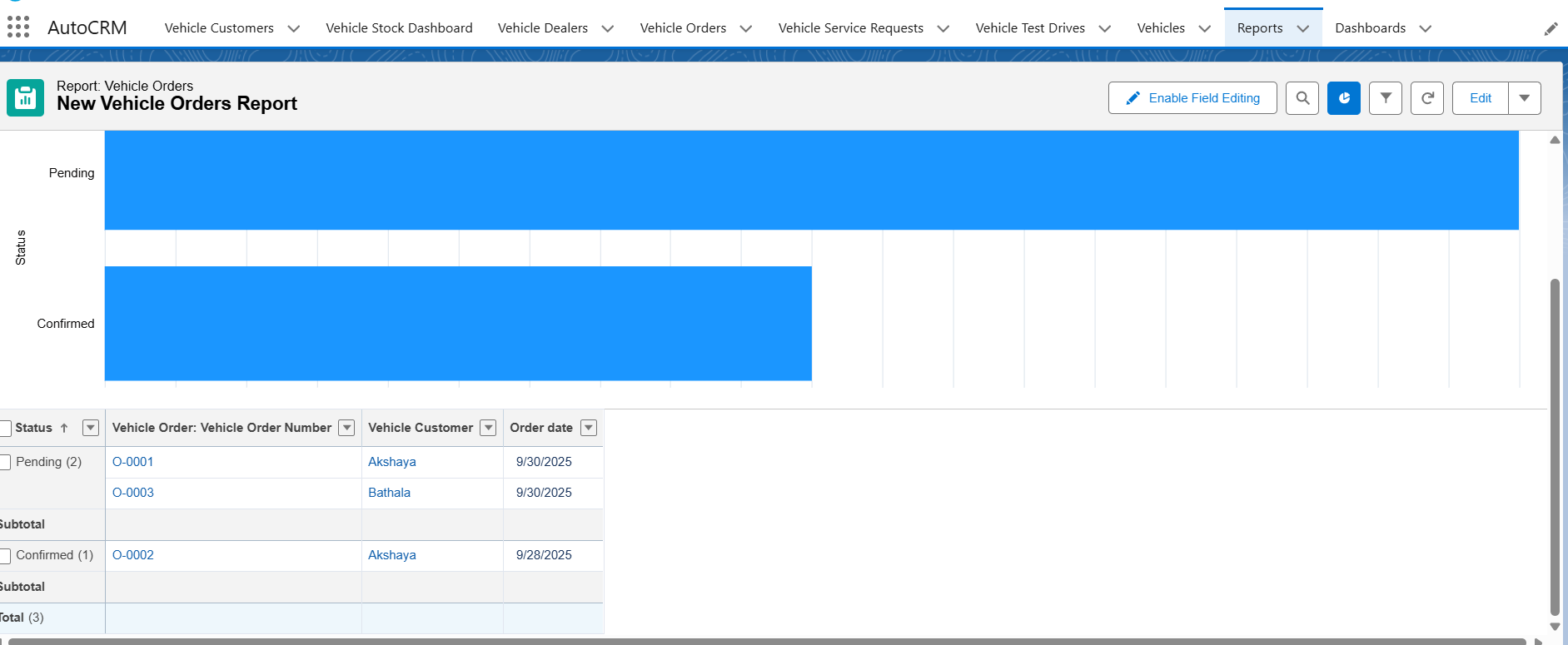
**Purpose:** This report allows the sales and operations team to quickly identify which vehicles are available or out of stock. The bar chart visually represents the quantity of each vehicle model in inventory, making it easier to plan orders and manage stock levels.



## Orders Status Report

* **Object:** Vehicle\_Order\_\_c
* **Fields Included:** Vehicle, Customer, Status, Order Date
* **Filters Applied:** All Orders
* **Chart Type:** Donut / Pie Chart (displaying order count by Status: Pending, Confirmed, Delivered)

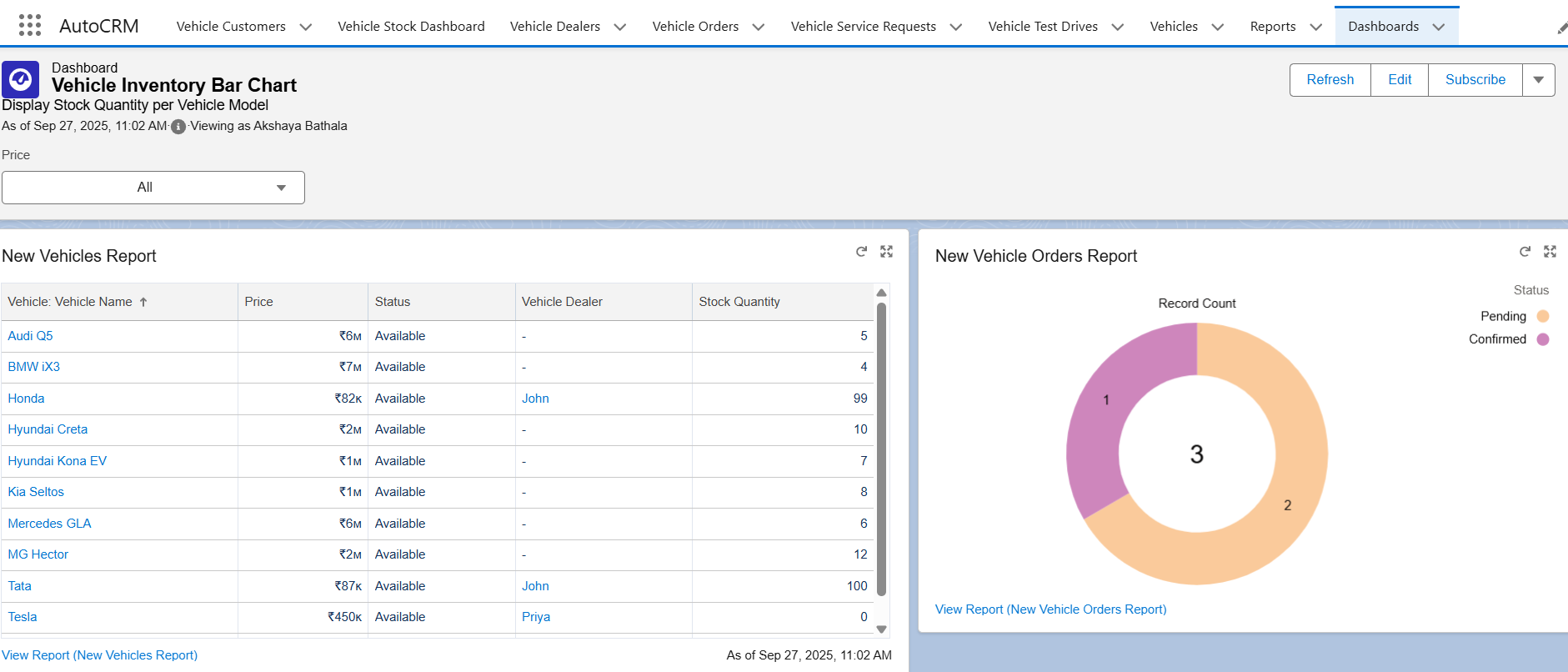
**Purpose:** This report provides insight into the current status of vehicle orders. Using a donut chart makes it simple to see the proportion of orders in different statuses, helping the team prioritize pending orders and track completed deliveries.

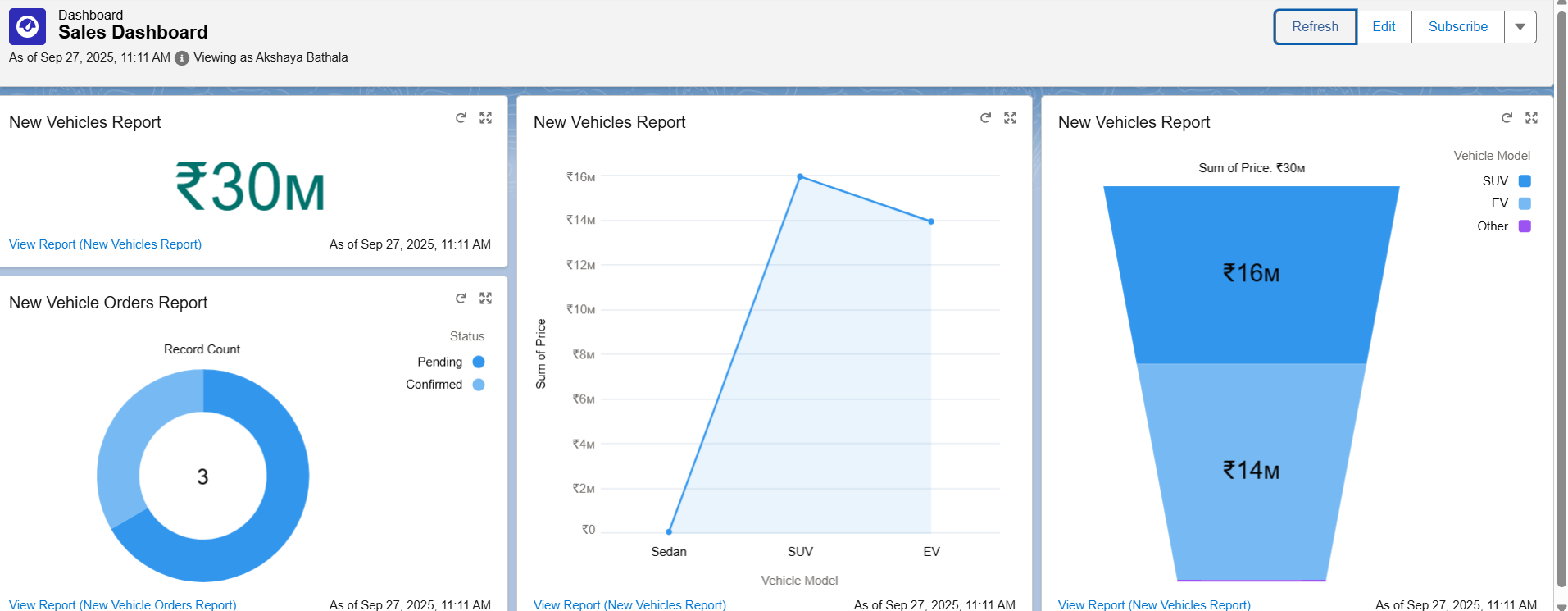


## Vehicle Inventory Dashboard

**Steps Performed:**

1. Navigated to **Dashboards → New Dashboard** and named it **Vehicle Inventory Dashboard**.
2. Added the **Vehicle Inventory Report** as a component.
3. Selected **Bar Chart** to display **Stock Quantity by Vehicle Model**.
4. Optionally added the **Orders Status Report** as a **Donut Chart** to provide a quick overview of order progress.
5. Arranged the charts for clear visual representation and saved the dashboard.



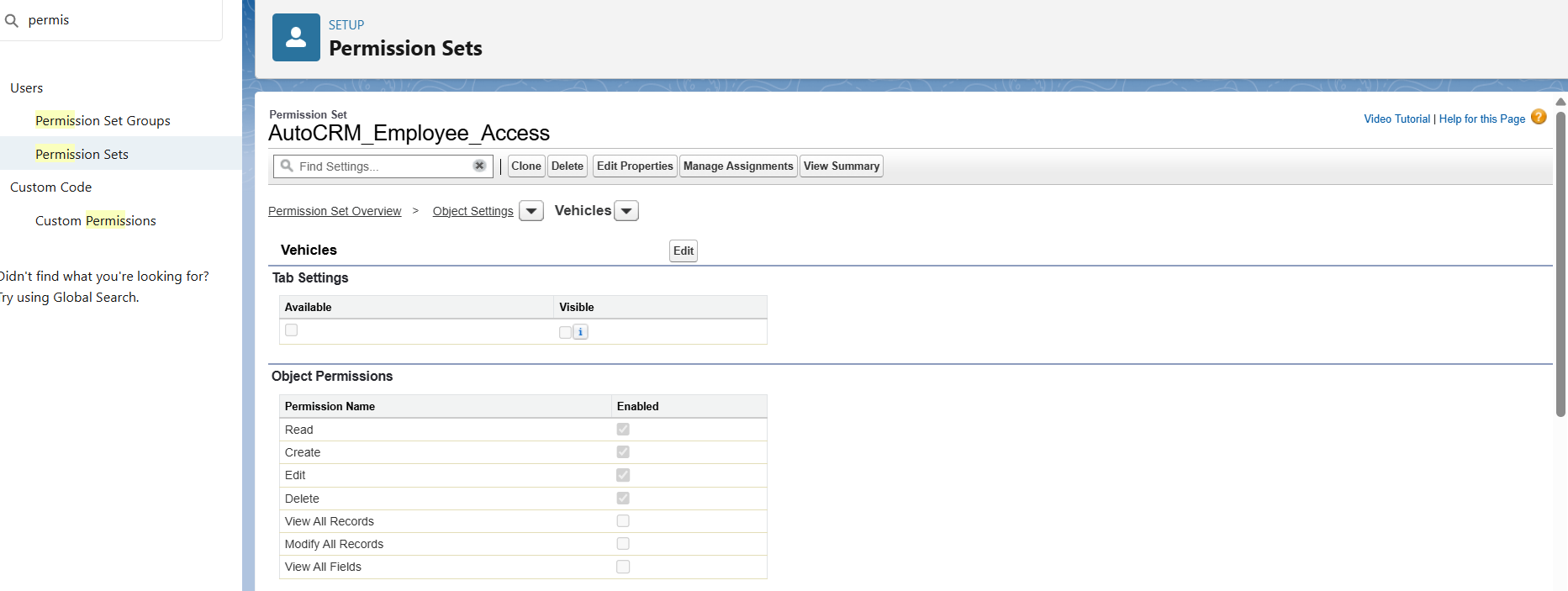


**Outcome:**

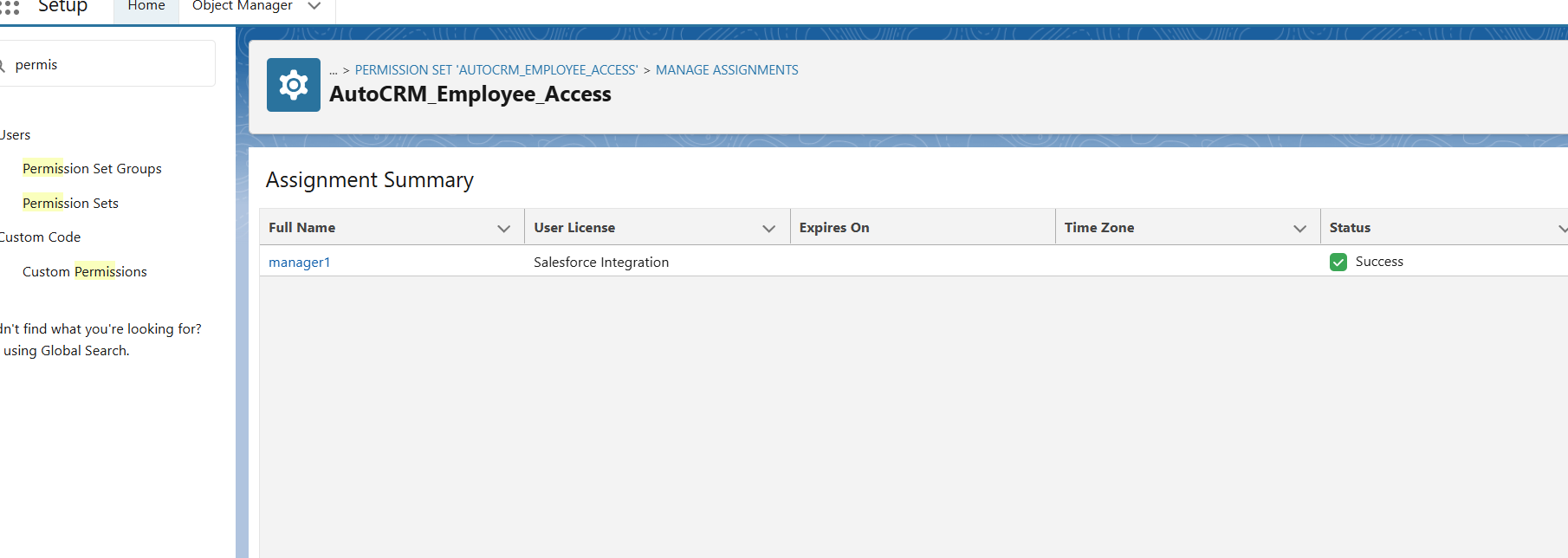
* The dashboard clearly shows **vehicle availability** and **order status** in a visual format.
* The bar chart and donut chart make it easy to interpret inventory and order data at a glance.
* Users can now monitor stock levels and pending orders directly from the dashboard.

## Creating Permission Set

1. **Navigate to Permission Sets:**
   * In Salesforce Setup, we went to **Permission Sets** and clicked **New**.
2. **Define Permission Set:**
   * **Label:** AutoCRM\_Employee\_Access
   * **API Name:** AutoCRM\_Employee\_Access
   * **User License:** None (to allow assignment to any user)
   * Click **Save** to create the permission set.



1. **Assign Object Permissions:**
   * Selected **Vehicle\_\_c** and granted **Read, Create, Edit, and Delete** permissions.
   * Selected **Vehicle\_Order\_\_c** and granted **Read, Create, Edit, and Delete** permissions.
   * This ensures users can manage vehicle inventory and order records in AutoCRM Hub without requiring profile changes.



**Outcome:**

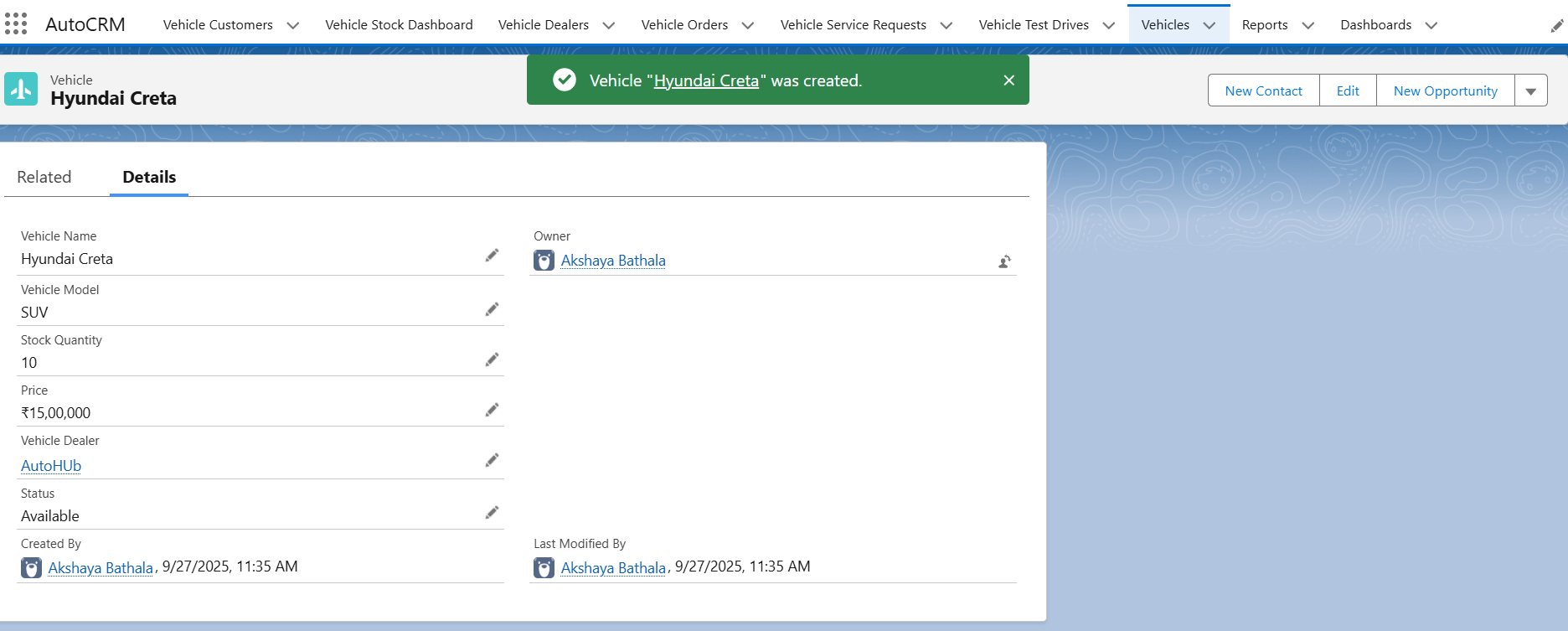
* Users assigned to this permission set can now interact with the custom objects necessary for AutoCRM Hub operations.
* Access is modular and secure, making it easier to manage user privileges.

# PHASE 10: Quality Assurance Testing

**Goal:** To ensure all Salesforce features in AutoCRM Hub are functioning correctly, including custom objects, LWC components, workflows, and validation rules. Testing is done by verifying expected behavior against actual system responses.

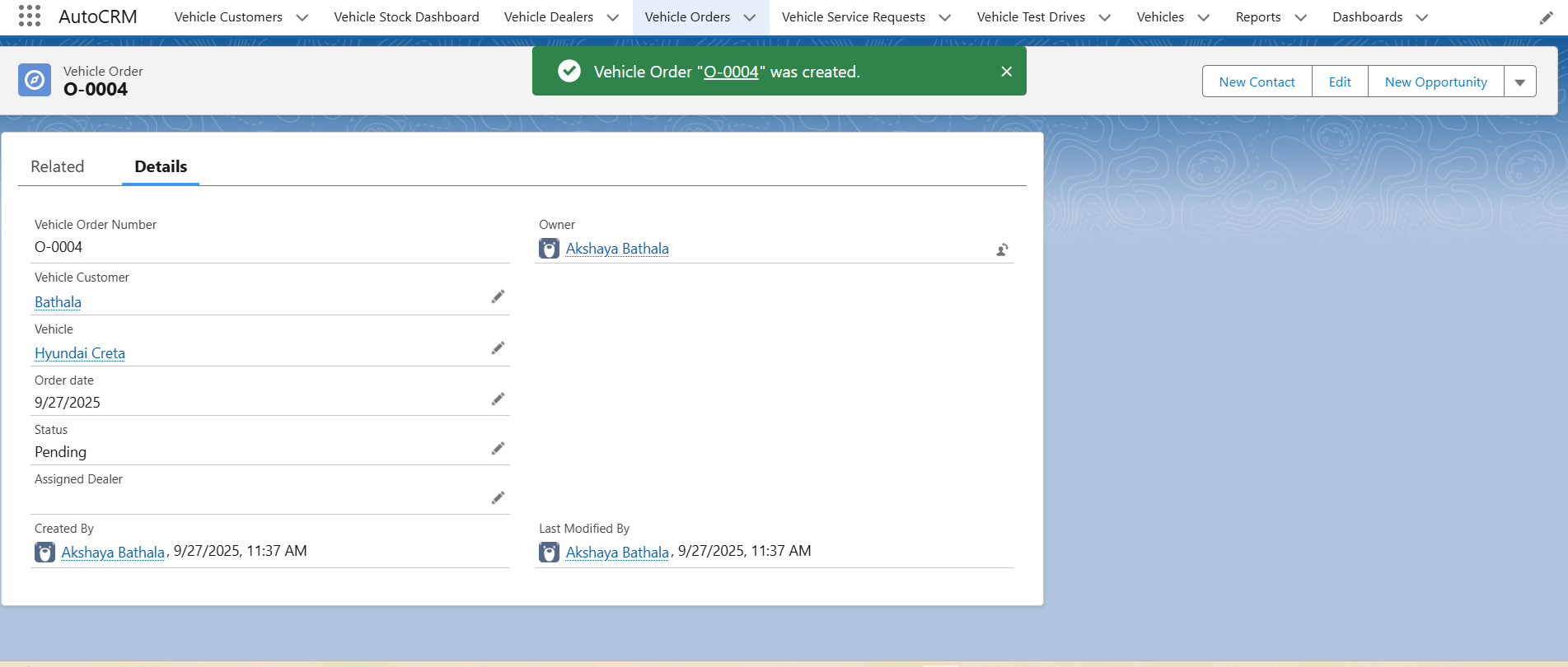
### **1. Vehicle Record Creation**

**Function:** Adding a new vehicle to the inventory.  
 **Input:** User creates a Vehicle\_\_c record with Vehicle Name = “Hyundai Creta”, Model = “SUV”, Stock Quantity = 10, Price = 1500000, Dealer = AutoHub, Status = Available.  
 **Expected Output:** The vehicle record should be created successfully and visible in the Vehicles tab with all fields populated correctly.  
 **Actual Output:** Vehicle record created successfully. Fields displayed correctly.

**

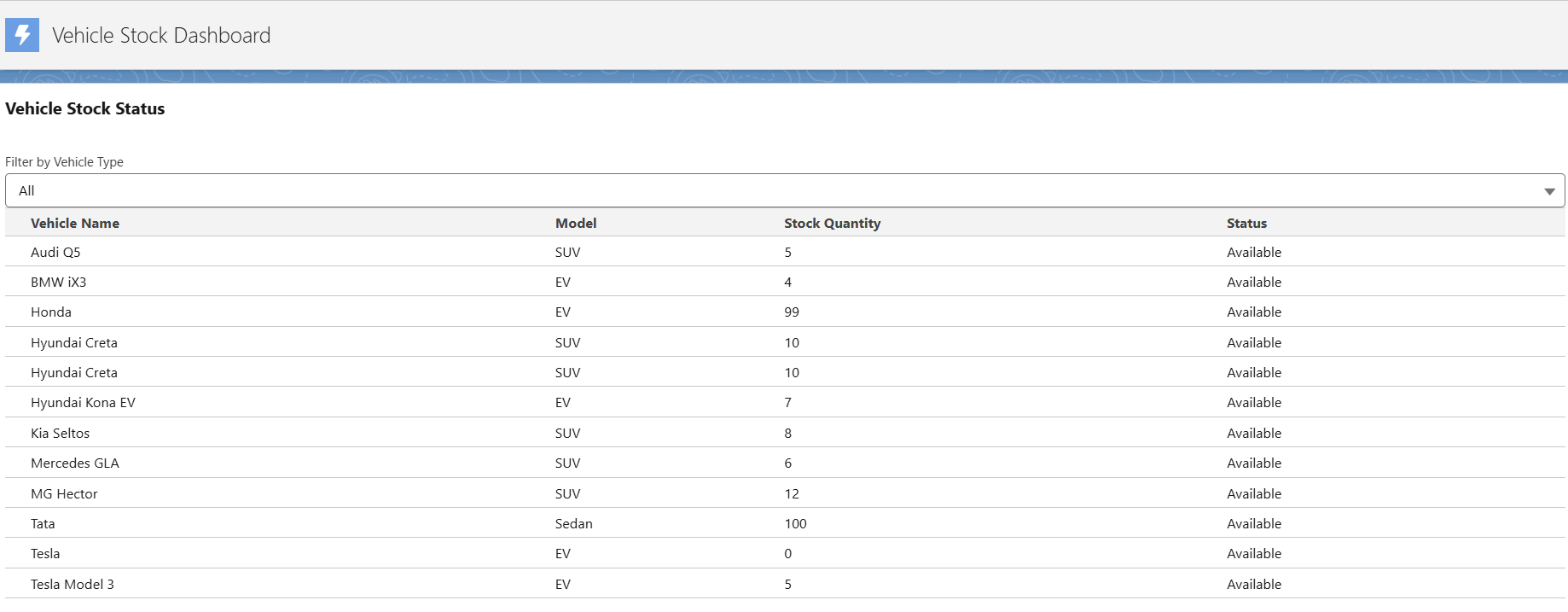
### **2. Vehicle Order Creation**

**Function:** Creating a new vehicle order for a customer.  
 **Input:** User creates a Vehicle\_Order\_\_c record for Customer = John Doe, Vehicle = Hyundai Creta, Order Date = Today, Status = Pending.  
 **Expected Output:** The order should be saved and linked to the selected customer and vehicle.  
 **Actual Output:** Vehicle order created successfully with correct relationships.



### **3. LWC Component – Vehicle Stock Status**

**Function:** Display current stock status of vehicles on a Lightning page.  
 **Input:** Navigate to the Lightning page containing vehicleStockStatus LWC.  
 **Expected Output:** LWC displays Vehicle Name, Stock Quantity, and Status correctly.  
 **Actual Output:** LWC rendered correctly, stock values match Vehicle\_\_c records.



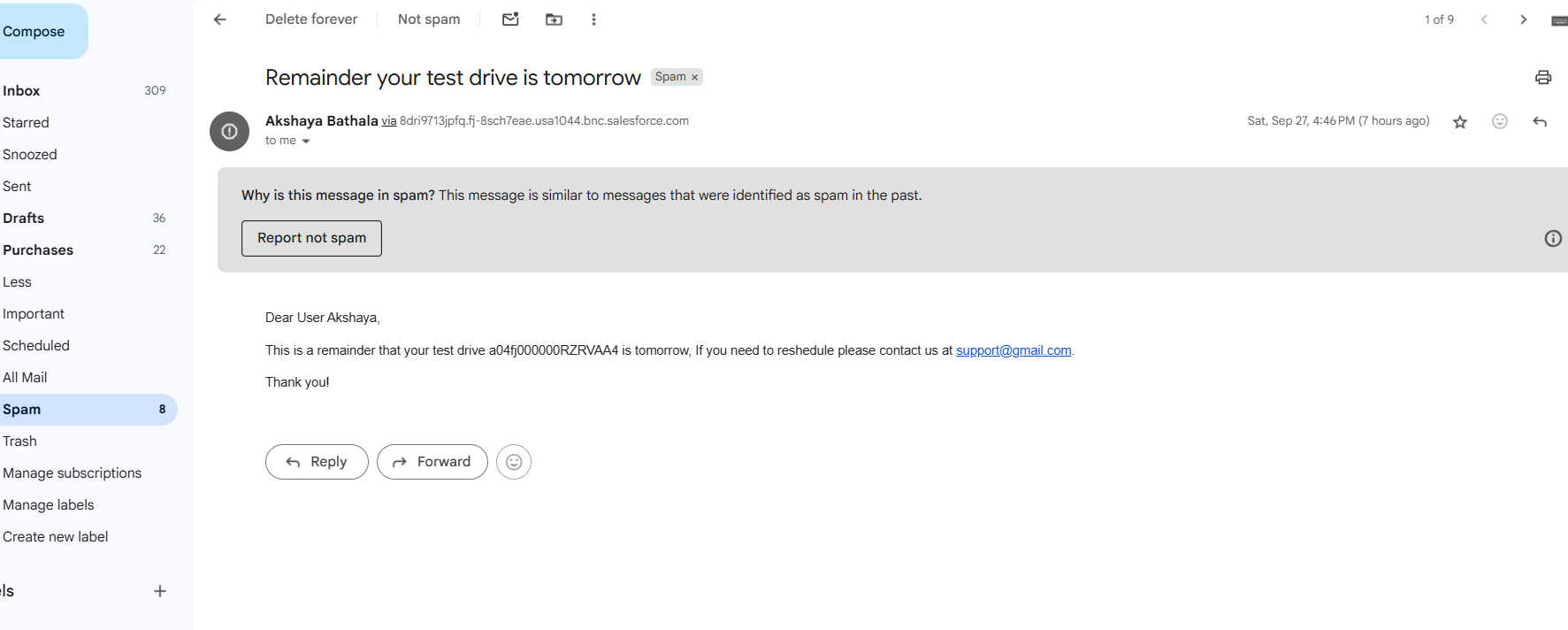
### **4. Validation Rule – Prevent Out-of-Stock Orders**

**Function:** Prevent creating orders for vehicles with zero stock.  
 **Input:** Attempt to create a Vehicle\_Order\_\_c for Vehicle = “Tesla” (Stock\_Quantity\_\_c = 0).  
 **Expected Output:** Validation rule triggers, blocking record creation, and displays error message: “Vehicle is out of stock.”  
 **Actual Output:** Order creation prevented. Error message displayed correctly.



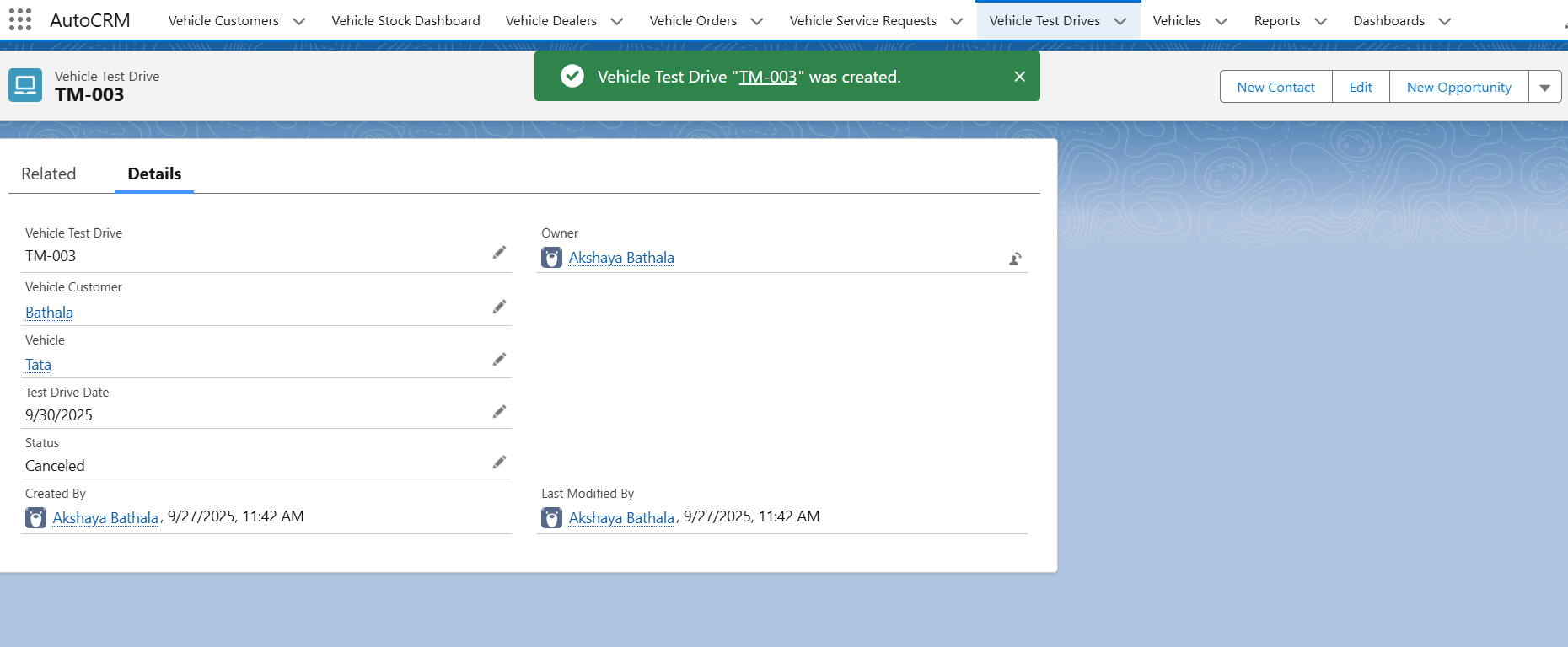
### **5. Workflow – Pending Order Reminder**

**Function:** Automatic reminder for pending orders.  
 **Input:** Create a pending Vehicle\_Order\_\_c record and trigger workflow.  
 **Expected Output:** Workflow sends reminder email to the customer and assigned dealer.  
 **Actual Output:** Workflow executed successfully, email received.



### **6. Vehicle Test Drive Record Creation**

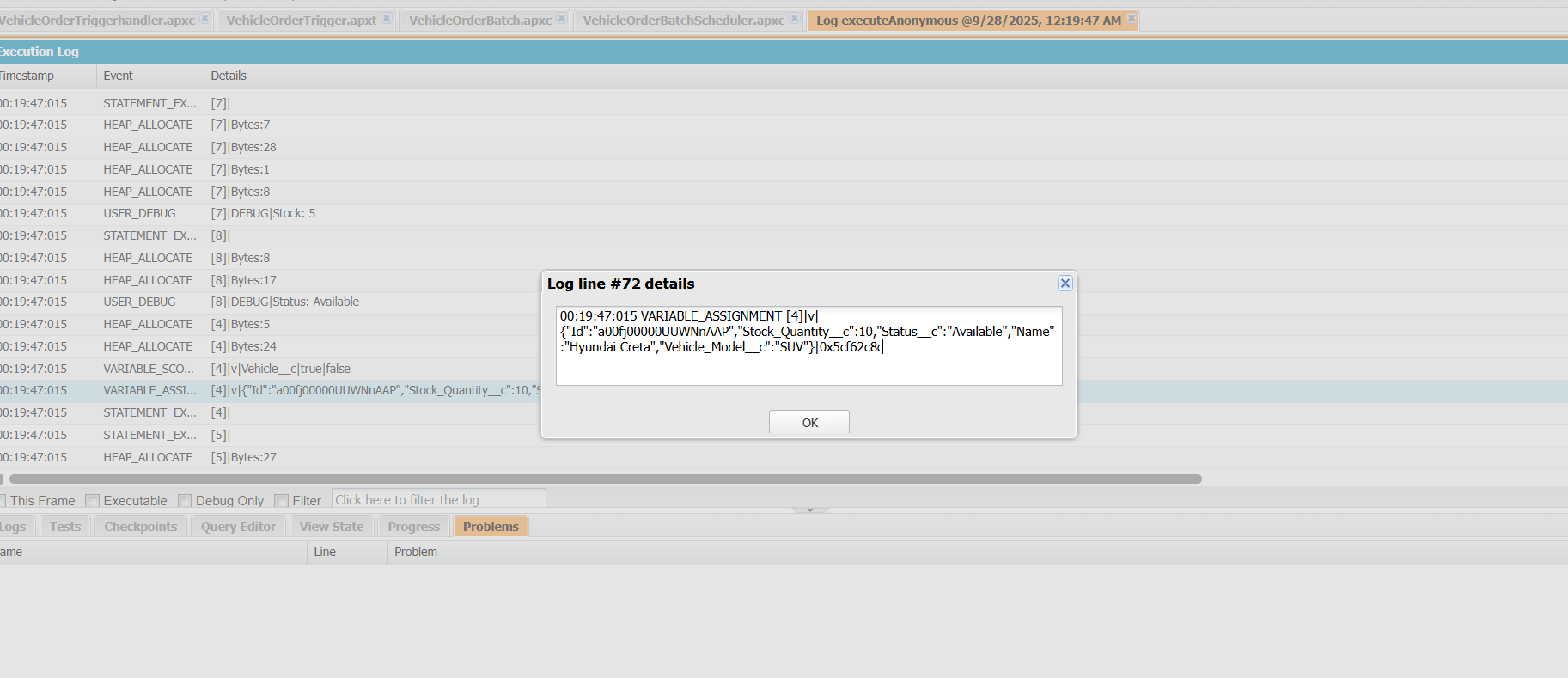
**Function:** Scheduling a vehicle test drive.  
 **Input:** Create Vehicle\_Test\_Drive\_\_c record with Customer = Bathala, Vehicle = Tata, Test Drive Date = Tomorrow, Status = Canceled.  
 **Expected Output:** Test drive record should be created and visible in Test Drive tab.  
 **Actual Output:** Test drive scheduled successfully. Record visible in tab.



### **7. Apex Class – VehicleController Data Retrieval**

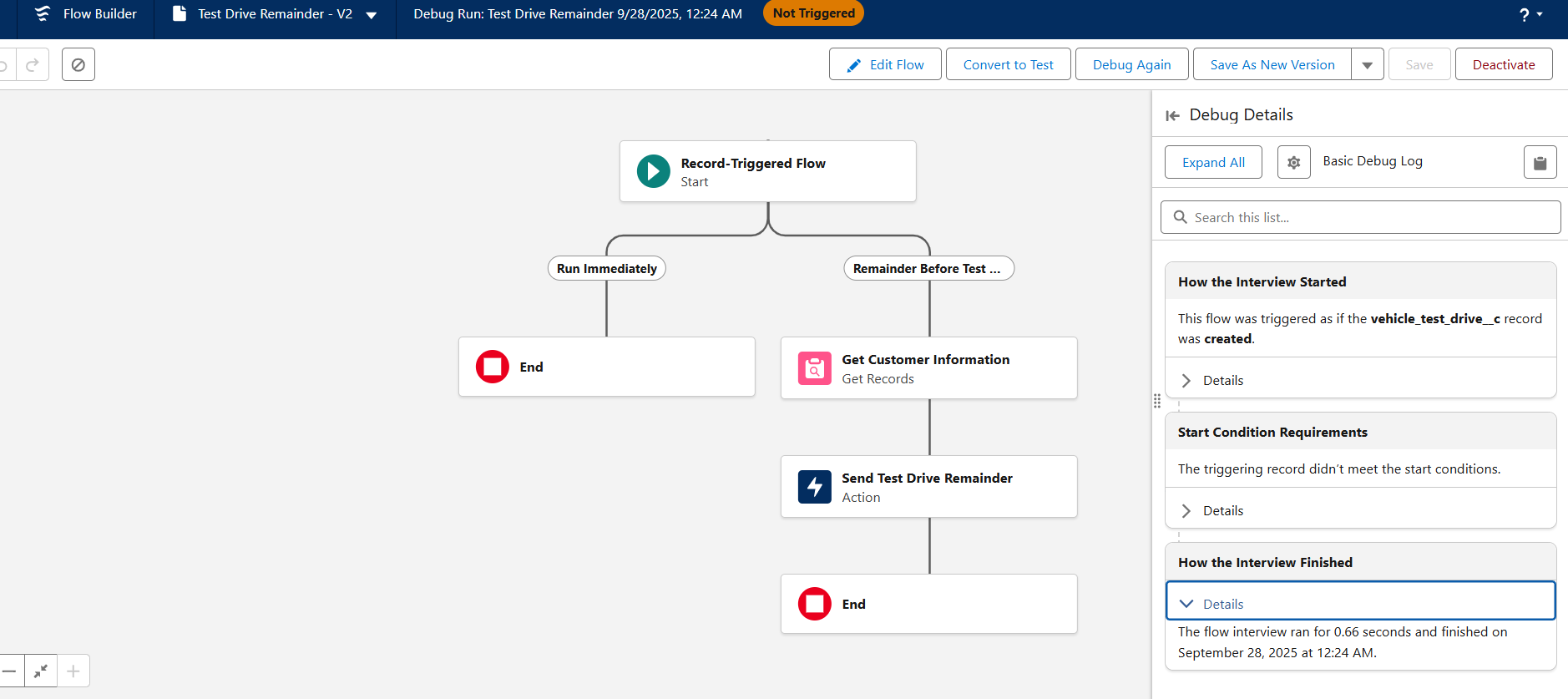
**Function:** Retrieve vehicle details such as name, model, price, and stock quantity using Apex.  
 **Input:** Execute Apex class VehicleController.getVehicleDetails('Hyundai Creta').  
 **Expected Output:** Apex method returns vehicle details correctly, which can be displayed in LWC or on a Lightning page.  
 **Actual Output:** Vehicle details fetched successfully, including name, model, price, stock quantity, and dealer.





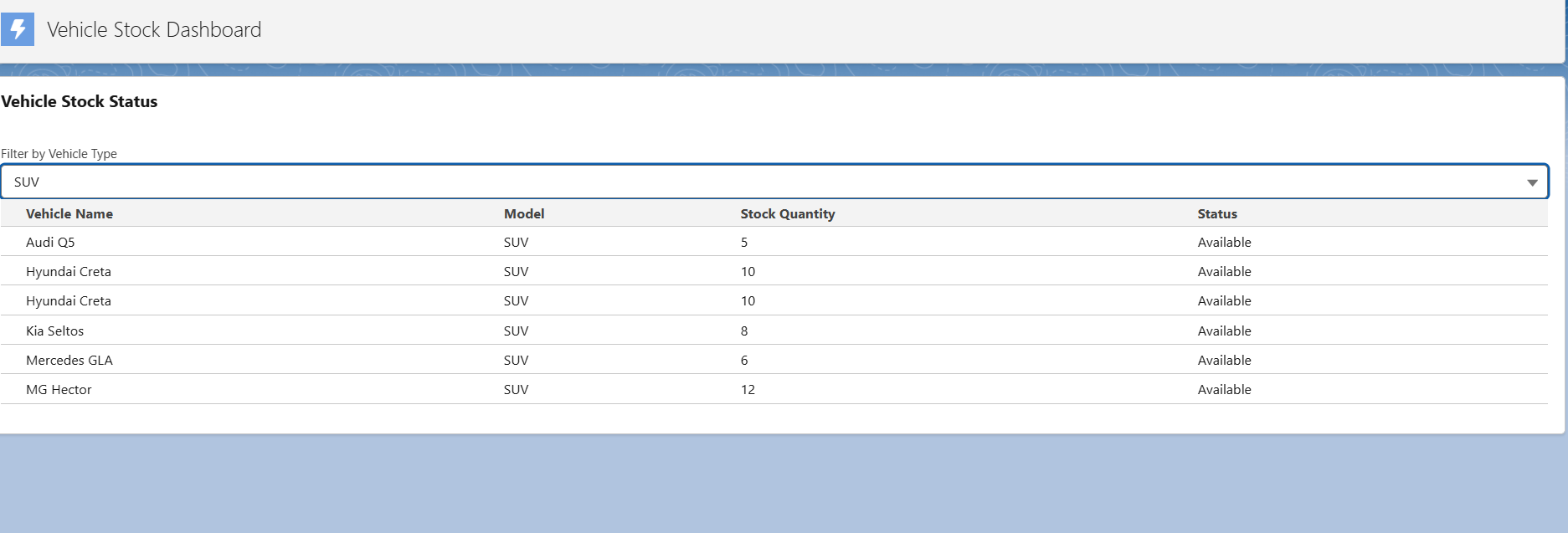
**8. No Email Sent for Unsheduled or Canceled Records**

* **Function:** Email Prevention Trigger – Vehicle Test Drive / Vehicle Order
* **Input:** Create a Vehicle\_Test\_Drive\_\_c record with Status = Canceled. Create a Vehicle\_Order\_\_c record with Status = Canceled. Do not schedule any other records.
* **Expected Output:** No emails should be triggered or sent to the customer or dealer. Workflow, Process Builder, or Apex Trigger responsible for email notifications should skip execution for canceled or unscheduled records.
* **Actual Output:** Verified that no emails were sent for canceled test drives or orders. Only scheduled/pending records triggered emails.



### **9. LWC – Stock Status Filtering**

**Function:** Filter vehicles based on availability in LWC.  
 **Input:** Use LWC filter to show only vehicles with Status = Available and vehicle = SUV.  
 **Expected Output:** LWC displays only vehicles with available stock.  
 **Actual Output:** LWC filter works correctly; only available SUV vehicles displayed.



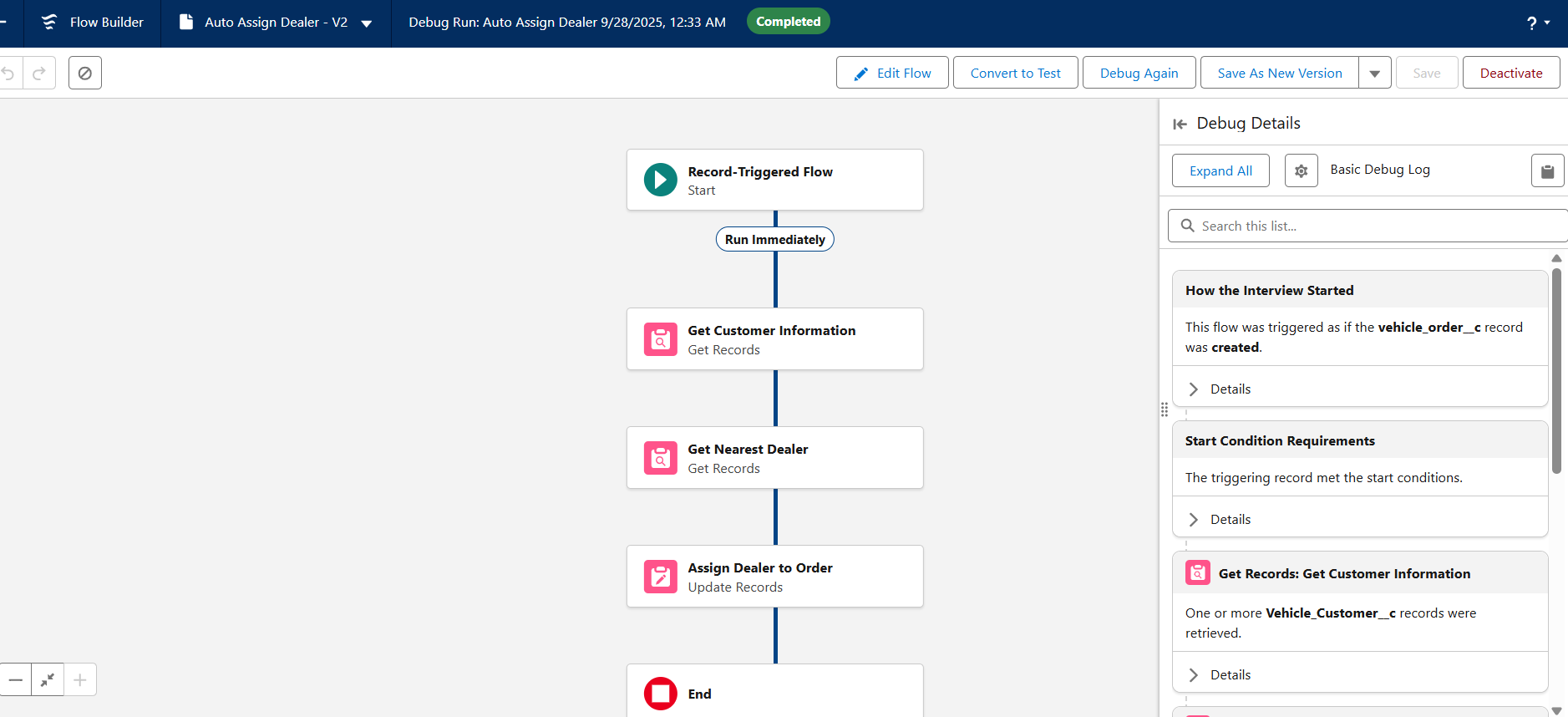
**10- Flow Assigns Dealer to Vehicle Order**

**Function:** Auto-assign nearest dealer to a new Vehicle\_Order\_\_c record based on customer address.

**Input:** A new Vehicle\_Order\_\_c record is created for a customer with a valid Address\_\_c. Flow is triggered automatically on record creation.

**Expected Output:** Flow retrieves the nearest Vehicle\_Dealer\_\_c record where Dealer\_Location\_\_c matches the customer address and assigns it to the order’s Assigned\_Dealer\_\_c field.

**Actual Output:** Flow successfully retrieved the nearest dealer and updated the Assigned\_Dealer\_\_c field of the Vehicle\_Order\_\_c record. The record now shows the assigned dealer correctly.



### **Future Enhancements**

To ensure that **AutoCRM** continues to evolve with industry needs and technological advancements, several future enhancements have been identified for subsequent development phases:

1. **AI-Powered Recommendation Engine**
   * Integrate predictive analytics to recommend suitable vehicle models to customers based on their preferences, browsing behavior, and historical purchase data.
   * Enhance dealer assignment using AI to suggest the most optimal dealer based on location, availability, and performance metrics.
2. **Advanced Scheduling with Agentforce / Einstein**
   * Automate vehicle order scheduling, service appointments, and test drives using intelligent Agentforce capabilities to reduce manual intervention and improve efficiency.
   * Use natural language inputs through the chatbot to schedule or modify appointments seamlessly.
3. **Multi-Channel Communication**
   * Integrate WhatsApp, SMS, and email APIs for real-time notifications, reminders, and promotional campaigns.
   * Enable customers to receive order and service updates through their preferred channels.
4. **Mobile Application Integration**
   * Develop a mobile app for customers and dealers to access real-time vehicle information, book services, track orders, and communicate directly with dealerships.
5. **Expanded Analytics & Reporting**
   * Build AI-driven dashboards with deeper insights into vehicle sales trends, service performance, and customer behavior.
   * Include predictive inventory forecasting to reduce stock-outs and optimize procurement cycles.
6. **Third-Party System Integrations**
   * Integrate with ERP systems, supplier platforms, and external price comparison APIs to create a unified ecosystem.
   * Enable Salesforce Connect for live data synchronization without duplicating external data.
7. **Enhanced Security & Role Management**
   * Implement more granular permission sets and profile controls to safeguard sensitive data.
   * Introduce audit trails and automated anomaly detection for compliance monitoring.