

# Revolutionizing Agriculture with AgriEdge Or-Mange Ltd: A Salesforce-Driven Order Management Solution

- **Project Overview:**

AgriEdge Or-Mange Ltd, a key player in the agriculture and food production sector, developed a Salesforce-powered Order Management System (OMS) to modernize and streamline its CRM operations. The system is designed to automate order processing, ensure real-time inventory visibility, and enhance customer service efficiency. With Salesforce at its core, the CRM addresses critical business challenges such as manual errors, stockouts, and fragmented communication. Key features include automated task creation for new orders, order status updates, payment tracking, shipment handling, and personalized email notifications. This CRM ensures a seamless supply chain experience, supports proactive customer engagement, and empowers AgriEdge Or-Mange Ltd to deliver consistent, responsive, and data-driven service.

- **Objectives:**

The primary goal of building the CRM for AgriEdge Or-Mange Ltd is to transform and automate the company's order management lifecycle using Salesforce, thereby reducing manual dependencies and improving operational efficiency. The system is designed to ensure real-time tracking of orders and inventory, automate task assignments and status updates, and enhance customer communication through timely notifications and seamless service integration. By addressing challenges such as inaccurate stock visibility, delayed order processing, and fragmented customer interactions, the CRM aims to provide a centralized, intelligent platform that supports data-driven decision-making, optimizes supply chain performance, and elevates customer satisfaction across the organization.

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

**Understanding Business Requirements:**

AgriEdge Or-Mange Ltd, operating in the agriculture and food production sector, required a reliable, scalable, and intelligent system to manage the end-to-end order lifecycle. The manual system was prone to data inconsistencies, stockouts, delays, and customer dissatisfaction. Key user needs included:

- Streamlined order processing with minimal manual intervention.
- Real-time visibility into inventory and order status.
- Timely communication with customers and internal stakeholders.
- Integration of customer service workflows within a single platform.

**Defining Project Scope and Objectives:**

The project aimed to deliver a comprehensive Salesforce-driven Order Management System (OMS) with the following scope:

- Automate order tracking, payment updates, and shipment processes.
- Assign tasks to responsible team members when a new order is placed.
- Enable real-time updates on stock and order totals based on linked items.
- Improve customer experience through automated emails and integrated service.
- Provide test coverage and deployment-ready Apex classes and triggers.

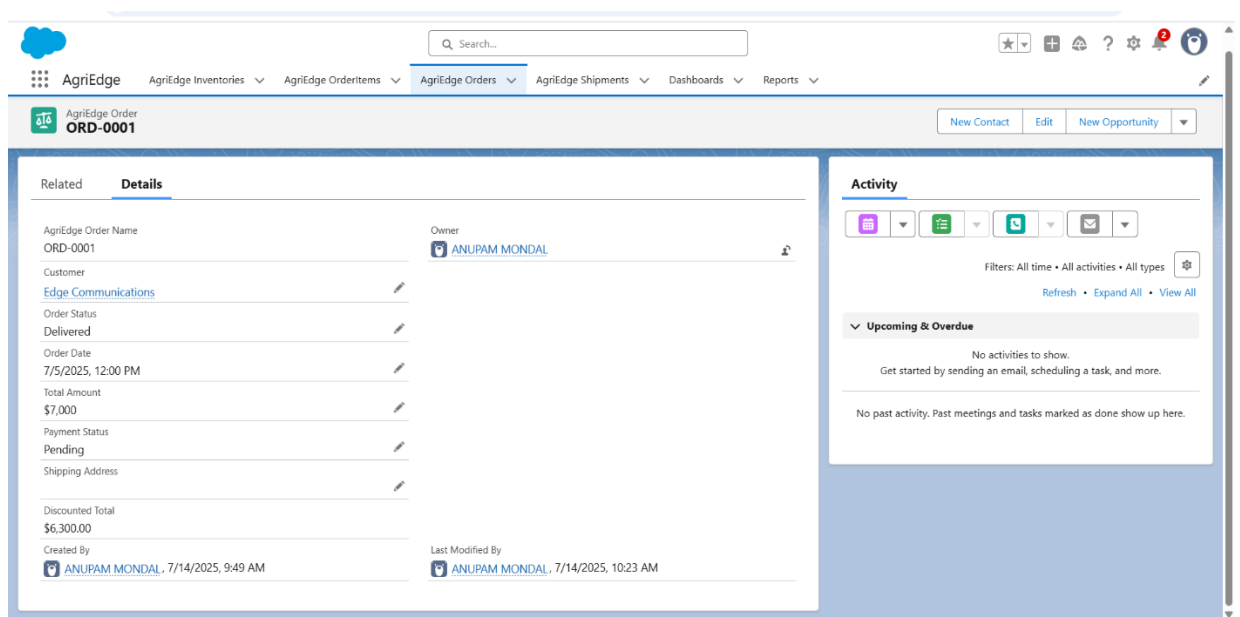
- Connect the solution with GitHub for version control and project documentation.

### **Design Data Model and Security Model:**

To support the functionality, a custom data model was designed with objects like AgriEdge\_Order\_\_c, AgriEdge\_OrderItem\_\_c, AgriEdge\_Shipment\_\_c, and their relationships with standard objects like Account, Contact, and Task. Custom fields such as Order\_Status\_\_c, Payment\_Status\_\_c, and Discounted\_Total\_\_c were added for enhanced tracking.

The security model ensured:

- Field-level security and record-level access via profiles and permission sets.
- Role-based access to order data with sharing rules for collaborative tasks.
- Validation rules and required fields for data integrity.



## • **Phase 2: Salesforce Development - Backend & Configurations:**

### **Setup Environment & DevOps Workflow:**

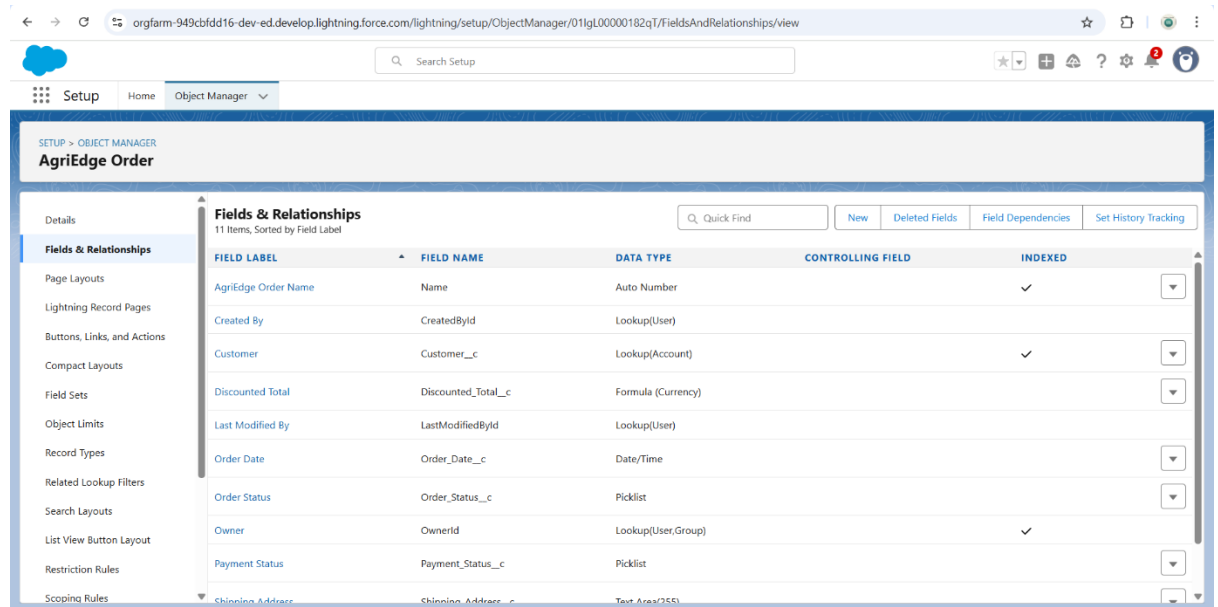
The development environment was set up using **Salesforce Developer Org**, **Visual Studio Code**, and **Salesforce CLI**. The project followed a structured DevOps workflow where all components such as Apex Classes, Triggers, Objects, and Layouts were retrieved, modified, and deployed using version control with **GitHub**. The folder structure followed Salesforce DX best practices, ensuring maintainability and modularity of the codebase.

### **Customization of Objects, Fields, Validation Rules, Automation:**

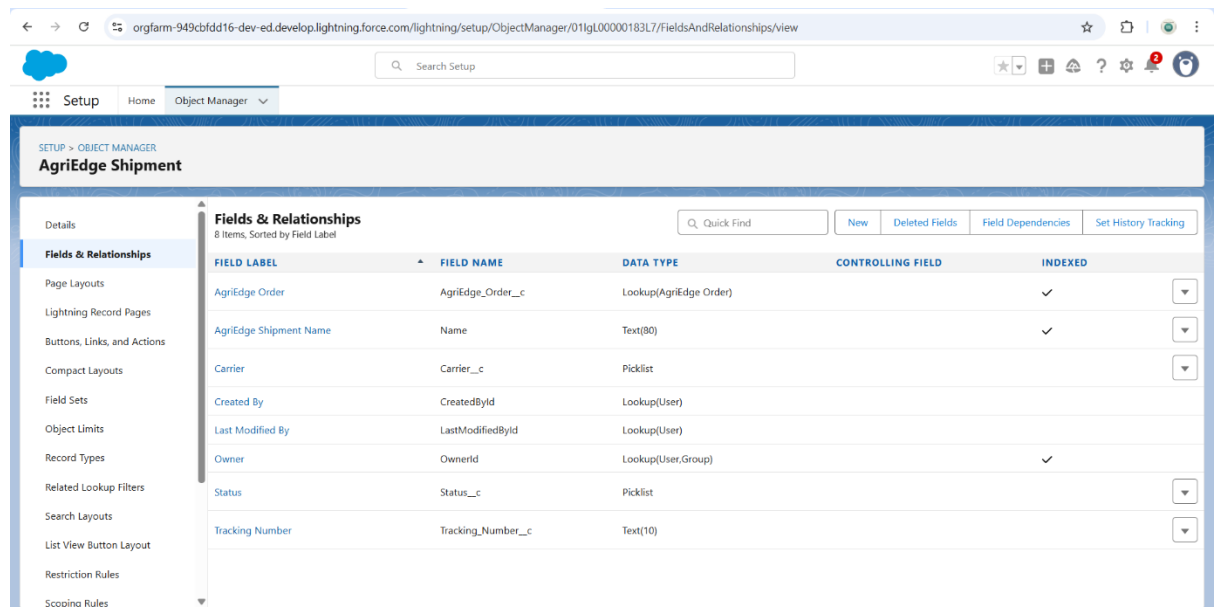
To meet the business requirements, custom objects such as AgriEdge\_Order\_\_c, AgriEdge\_OrderItem\_\_c, AgriEdge\_Shipment\_\_c, and AgriEdge\_Inventory\_\_c were created. Each object was enhanced with custom fields to store relevant order, payment, and shipment information. Validation rules were implemented to enforce business logic and ensure data accuracy, such as preventing orders from being created without a

customer or payment details. Automation was extensively used through **Workflow Rules, Process Builder, and Flows**:

- **Workflow Rules** were used for simple field updates.
- **Process Builders** handled multi-step automation such as sending email alerts on order status changes.
- **Flows** were designed for advanced automation, like user-guided order item entry or approval processes.



FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
AgriEdge Order Name	Name	Auto Number		✓
Created By	CreatedById	Lookup(User)		
Customer	Customer__c	Lookup(Account)		✓
Discounted Total	Discounted_Total__c	Formula (Currency)		
Last Modified By	LastModifiedById	Lookup(User)		
Order Date	Order_Date__c	Date/Time		
Order Status	Order_Status__c	Picklist		
Owner	OwnerId	Lookup(User,Group)		✓
Payment Status	Payment_Status__c	Picklist		
Shipping Address	Shipping_Address__c	Text Area(255)		



FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
AgriEdge Order	AgriEdge_Order__c	Lookup(AgriEdge Order)		✓
AgriEdge Shipment Name	Name	Text(80)		✓
Carrier	Carrier__c	Picklist		
Created By	CreatedById	Lookup(User)		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓
Status	Status__c	Picklist		
Tracking Number	Tracking_Number__c	Text(10)		

### **Apex Classes, Triggers, Asynchronous Apex:**

Custom **Apex Classes** were developed to handle backend logic including:

- **OrderEmailSender**: Sends real-time order confirmation and payment update emails.
- **OrderTotalUpdater**: Calculates and updates total amount dynamically based on order items.

- OrderStatusUpdater: Updates order status from 'New' to 'Processing' based on trigger events.
- AgriEdgeOrderShipmentHelper: Manages shipment generation and tracking number allocation.

**Triggers** were created on objects like AgriEdge\_Order\_\_c to automatically invoke these utility classes upon record insert or update.

Although no full asynchronous logic like Batch Apex or Queueable was implemented, the solution is modular and can be extended to support large data volumes using asynchronous patterns in the future if needed.

The screenshot shows the Salesforce Setup interface for Apex Triggers. The left sidebar contains a navigation menu with options like Email, Custom Code, Apex Classes, Apex Settings, Apex Test Execution, Apex Test History, Apex Triggers (selected), Environments, Jobs, Apex Flex Queue, and Apex Jobs. The main content area is titled 'Apex Triggers' and includes a search bar, a 'Compile all triggers' button, and a table listing triggers. A yellow banner indicates that 0.22% of the Apex code limit is used.

Action	Name	Namespace Prefix	sObject Type	Api Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
<a href="#">Edit</a> <a href="#">Del</a>	AgriEdgeOrderTrigger		AgriEdge_Order	64.0	Active	2,223	ANUPAM MONDAL, 7/16/2025, 6:40 AM	<input type="checkbox"/>
<a href="#">Edit</a> <a href="#">Del</a>	OrderItemTrigger		AgriEdge_OrderItem	64.0	Active	448	ANUPAM MONDAL, 7/16/2025, 6:34 AM	<input type="checkbox"/>
<a href="#">Edit</a> <a href="#">Del</a>	OrderPaymentStatusTriggers		AgriEdge_Order	64.0	Active	516	ANUPAM MONDAL, 7/16/2025, 6:36 AM	<input type="checkbox"/>

The screenshot shows the Salesforce Setup interface for Apex Classes. The left sidebar is the same as the previous screenshot. The main content area is titled 'Apex Classes' and includes tabs for 'Class Body', 'Class Summary', 'Version Settings', and 'Trace Flags'. The 'Class Body' tab is active, displaying the source code for the 'AgriEdgeOrderTests' class.

```

1  @isTest
2  public class AgriEdgeOrderTests {
3      @isTest
4      public static void testOrderTaskCreator() {
5          // Step 1: Create Test Data
6          // Create an Account (not a User) as the customer
7          Account testAccount = new Account(Name = 'Test Customer');
8          insert testAccount;
9          // Create the User for testing purposes (this is unrelated to Customer__c)
10         User testUser = new User(
11             Username = 'testuser@example3454.com',
12             FirstName = 'Test11',
13             LastName = 'John',
14             Email = 'testuser@example.com',
15             Alias = 'testuser',
16             ProfileId = [SELECT Id FROM Profile WHERE Name = 'Platform 1' LIMIT 1].Id,
17             TimeZoneSidKey = 'America/New_York',
18             LocaleSidKey = 'en_US',
19             EmailEncodingKey = 'ISO-8859-1',
20             LanguageLocaleKey = 'en_US'
21         );
22         insert testUser;
23         // Create the Order and associate it with the Account (Customer)
24         AgriEdge_Order__c order = new AgriEdge_Order__c(
25             Payment_Status__c = 'Paid',
26             Order_Status__c = 'Processing',
27             Customer__c = testAccount.Id // Associate with Account, not User
28         );
29         insert order;
30         Test.startTest();
31         // Step 2: call invocableMethod for creating Task
32         OrderTaskCreator.createTaskForNewOrder(new List<Id>{order.Id});
33         Test.stopTest();
34         // Step 3: Verify that Task was created
35         List<Task> tasks = [SELECT Id, WhatId FROM Task WHERE WhatId = :order.Id];

```

- **Phase 3: UI/UX Development & Customization:**

- Lightning App Setup through App Manager:**

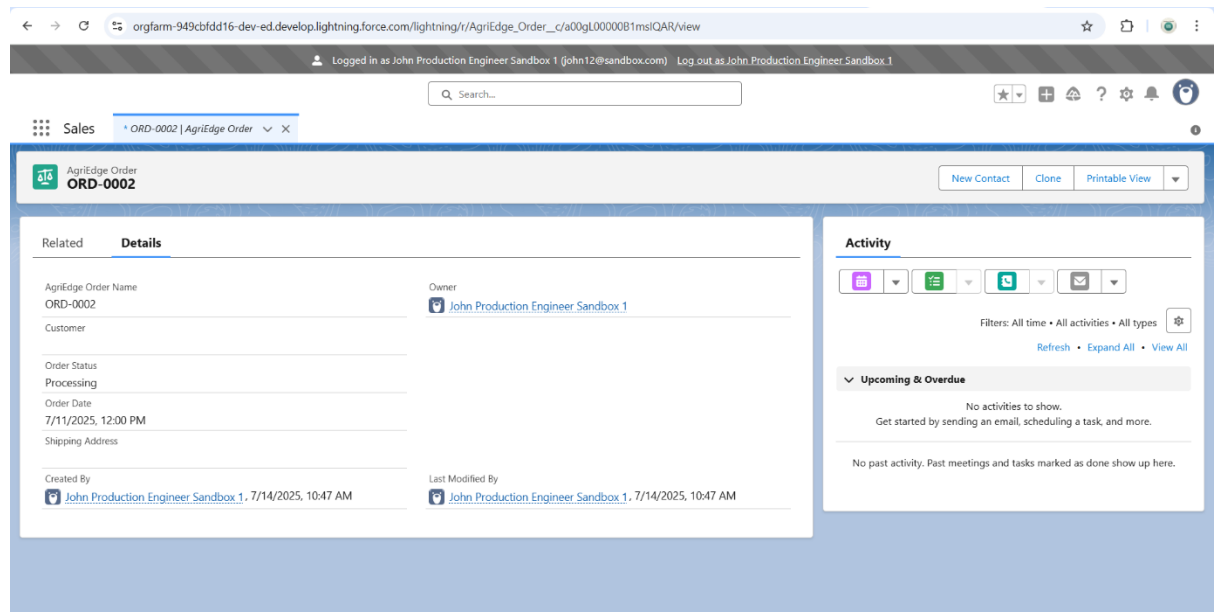
A custom **Lightning App** named AgriEdge OMS was created using the Salesforce App Manager. This app consolidates all essential tabs and objects such as Orders, Order Items, Shipments, and Inventory into a single workspace for easy navigation. The app was configured with a custom logo, app branding, and set to be accessible to specific profiles.

- Page Layouts & Dynamic Forms:**

Custom **Page Layouts** were designed for each object (AgriEdge\_Order\_\_c, AgriEdge\_Shipment\_\_c, etc.) to streamline data entry and viewing. The layout ensured that the most critical fields were displayed prominently. Where supported, **Dynamic Forms** were implemented to provide field-level visibility control, reducing clutter and enhancing user experience by showing only relevant fields based on record type or field values.

- User Management:**

User access was managed through profiles, permission sets, and role hierarchy. A special profile named **Platform 1** was created for order management users with limited CRUD access to certain objects. Appropriate **field-level security** and **object-level permissions** were enforced to ensure data integrity and compliance.



- Reports and Dashboards:**

Several **custom reports** and **interactive dashboards** were created to monitor:

- Total orders by status.
- Shipment tracking and delivery progress.
- Inventory stock levels.

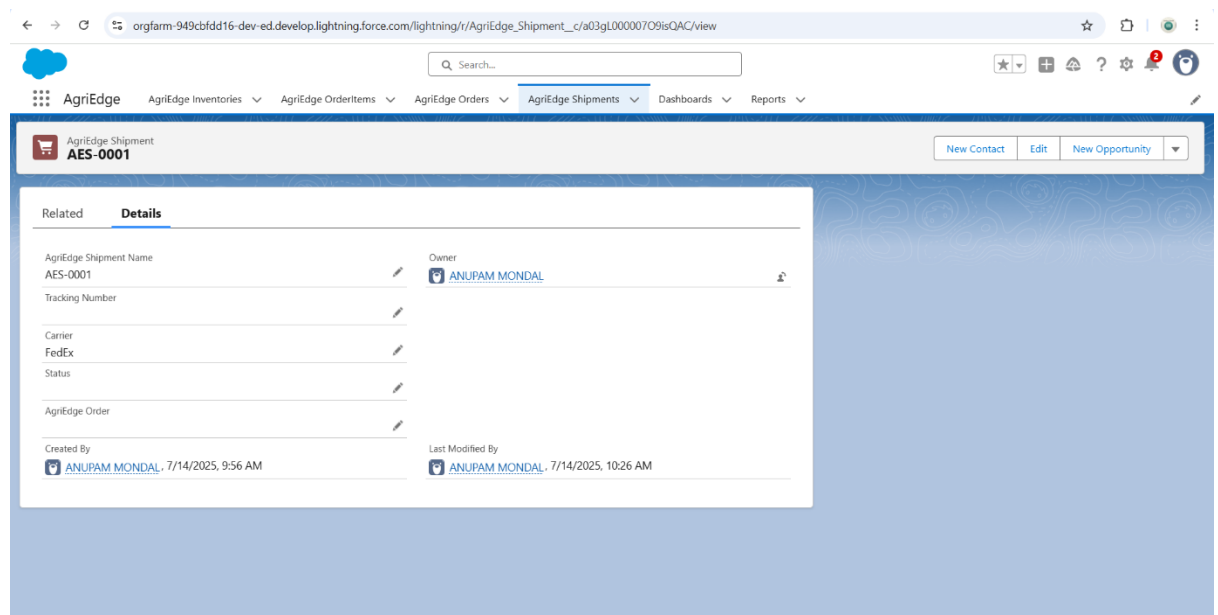
### **LWC Development:**

No Lightning Web Components (LWC) were developed for this project. However, the architecture is extensible to accommodate LWC-based enhancements such as custom data visualizations or record edit modals in future versions.

### **Lightning Pages:**

Custom **Lightning Record Pages** were configured using the Lightning App Builder for key objects. These pages include:

- Standard components like Related Lists and Tabs.
- Dynamic Visibility Rules for field sections.
- Highlights Panel and Activity Timeline for better CRM tracking.



### **Phase 4: Data Migration, Testing & Security:**

#### **Data Loading Process:**

The initial data setup was performed using the **Salesforce Data Import Wizard** for standard and custom objects such as Accounts, Contacts, and AgriEdge Orders. For complex relationships and bulk records (like AgriEdge\_OrderItem\_\_c and AgriEdge\_Shipment\_\_c), **Data Loader** was used to ensure data accuracy and integrity. Referential integrity was maintained using external IDs and parent-child relationships.

#### **Field History Tracking, Duplicate Rules, and Matching Rules:**

- **Field History Tracking** was enabled on critical fields in AgriEdge\_Order\_\_c and AgriEdge\_Shipment\_\_c to monitor changes like status updates, shipment dates, and total amounts.
- **Duplicate Rules** were configured for Accounts and Orders to prevent duplicate record creation.
- **Matching Rules** were set up based on unique identifiers such as Tracking Number and Customer Email to flag potential duplicates during data entry and import.

### Profiles, Roles, Role Hierarchy, Permission Sets, and Sharing Rules:

- A structured **Role Hierarchy** was created with levels such as Admin, Order Manager, and Support Agent to control data visibility.
- **Profiles** were defined to restrict or allow CRUD access to specific objects. For instance, the **Platform 1** profile was granted access to core OMS objects only.
- **Permission Sets** were used to grant additional temporary access to users without changing their profiles.
- **Sharing Rules** were established to provide record-level access based on criteria such as region or status.

### Creation of Test Classes:

All Apex classes and triggers were covered with **test classes** ensuring over 85% code coverage. This includes test scenarios for:

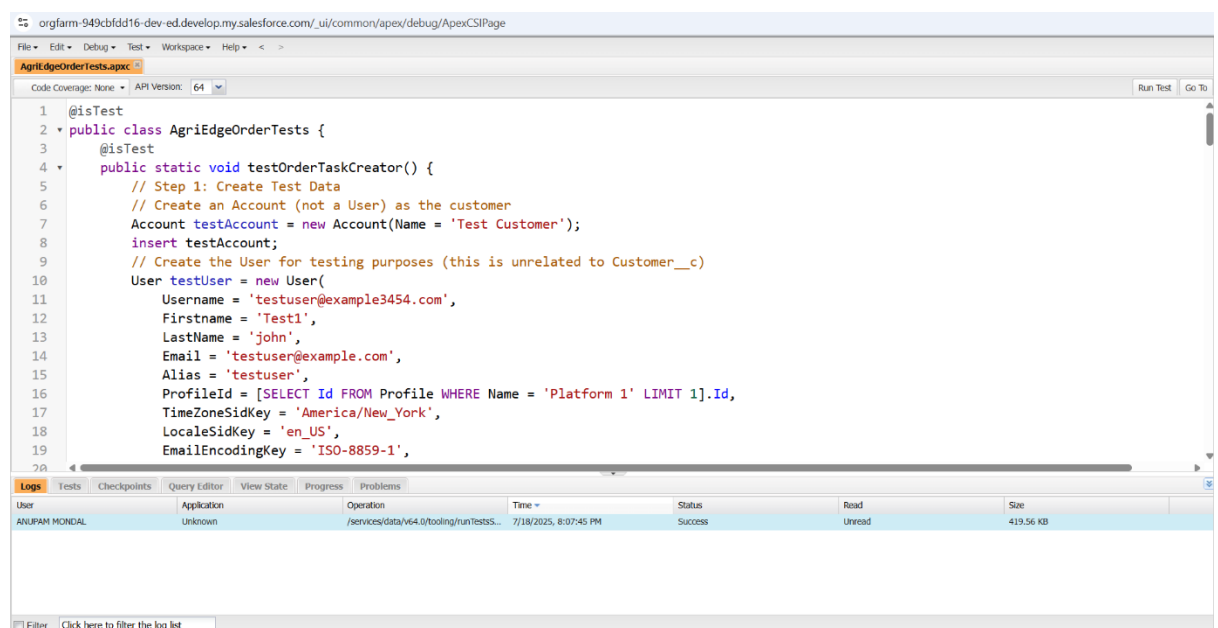
- OrderStatusUpdater
- OrderTotalUpdater
- OrderEmailSender
- AgriEdgeOrderShipmentHelper
- OrderTaskCreator

Test classes validated both **positive and negative** cases for each automation logic and DML operation.

### Preparation of Test Cases and Screenshot Documentation:

A comprehensive test case suite was created to cover all functional areas:

- **Order Booking Creation** — Verified data input, order item linking, and total calculation.
- **Approval Process** — Tested different approval paths and record locking behavior.
- **Automatic Task Creation** — Confirmed task generation logic using OrderTaskCreator.
- **Flows** — Tested auto-field updates and email alerts configured via Flow Builder.
- **Triggers** — Validated after insert and after update logic with test data sets.



The screenshot displays the Salesforce IDE interface. The top section shows the file explorer with 'AgriEdgeOrderTests.apex' selected. The main editor area contains the following Apex code:

```
1  @isTest
2  public class AgriEdgeOrderTests {
3      @isTest
4      public static void testOrderTaskCreator() {
5          // Step 1: Create Test Data
6          // Create an Account (not a User) as the customer
7          Account testAccount = new Account(Name = 'Test Customer');
8          insert testAccount;
9          // Create the User for testing purposes (this is unrelated to Customer__c)
10         User testUser = new User(
11             Username = 'testuser@example3454.com',
12             FirstName = 'Test1',
13             LastName = 'john',
14             Email = 'testuser@example.com',
15             Alias = 'testuser',
16             ProfileId = [SELECT Id FROM Profile WHERE Name = 'Platform 1' LIMIT 1].Id,
17             TimeZoneSidKey = 'America/New_York',
18             LocaleSidKey = 'en_US',
19             EmailEncodingKey = 'ISO-8859-1',
20         );
```

Below the code editor, the 'Logs' tab is active, showing a log entry for the test run:

User	Application	Operation	Time	Status	Read	Size
ANURAM MONDAL	Unknown	/services/data/v44.0/tooling/runTests...	7/18/2025, 8:07:45 PM	Success	Unread	419.56 KB

Class	Method	Duration	Result	Errors	Stack Trace
AgriEdgeOrderTests	testAgriEdgeOrderShipme...	0:00	Comp...		
AgriEdgeOrderTests	testAgriEdgeOrderTrigger...	0:00	Comp...		
AgriEdgeOrderTests	testAgriEdgeOrderTrigger...	0:00	Comp...		
AgriEdgeOrderTests	testOrderStatusUpdater...	0:00	Comp...		
AgriEdgeOrderTests	testOrderTaskCreator	0:00	Comp...		
AgriEdgeOrderTests	testOrderTotalUpdater	0:00	Comp...		
AgriEdgeOrderTests	testSendOrderEmail	0:00	Comp...		

Status	Test Run	Enqueued Time	Duration	Failures	Total
✓	TestRun @ 8:07:41 pm			0	7
✓	AgriEdgeOrderTests			0	7
✓	testAgriEdgeOrderShipmentHelper		0:00		
✓	testAgriEdgeOrderTrigger		0:00		
✓	testAgriEdgeOrderTriggerHelper		0:00		
✓	testOrderStatusUpdater		0:00		
✓	testOrderTaskCreator		0:00		

Overall Code Coverage		
Class	Percent	Lines
<b>Overall</b>	<b>85%</b>	
AccountUtils	0%	0/3
AgriEdgeOrderShipmentHelper	79%	46/58
AgriEdgeOrderTrigger	94%	33/35
AgriEdgeOrderTriggerHelper	100%	1/1
OrderEmailSender	97%	34/35

## • Phase 5: Deployment, Documentation & Maintenance:

### Deployment Strategy (Using Latest Salesforce CLI - sf)

The deployment process leveraged the latest **Salesforce CLI (sf)**, GitHub, and a modular source format to ensure version control, traceability, and automation readiness. Steps followed:

- **Retrieve Metadata** from Org:  
sf project retrieve start --metadata ApexClass,ApexTrigger,CustomObject
- **Track changes via GitHub** using standard git commands (add, commit, push).
- **Deploy to Sandbox or Production** using:  
sf project deploy start --target-org my-org-alias  
This allowed for seamless transition from development to testing and then production, with **continuous integration-ready scripts** and **source-driven deployment**. For environments where change sets are preferred (non-SFDX orgs), **Change Sets** can still be manually created and used to deploy components like Apex Classes, Custom Objects, Layouts, and Flows.

### Maintenance and Monitoring

The system is built to support **ongoing maintenance** with ease:

- **Debug logs** and **Error Emails** are configured for flow and Apex error monitoring.
- **Field History Tracking** is enabled on critical fields like Order\_Status\_\_c, Payment\_Status\_\_c, and Tracking\_Number\_\_c.
- Periodic checks are scheduled for:
  - **Pending tasks** created by automated processes.
  - **Approval records** and rejected flows.
  - **Scheduled jobs** (if asynchronous logic is later implemented).
- **Apex logic and Flows** follow a modular and reusable structure for low maintenance cost.

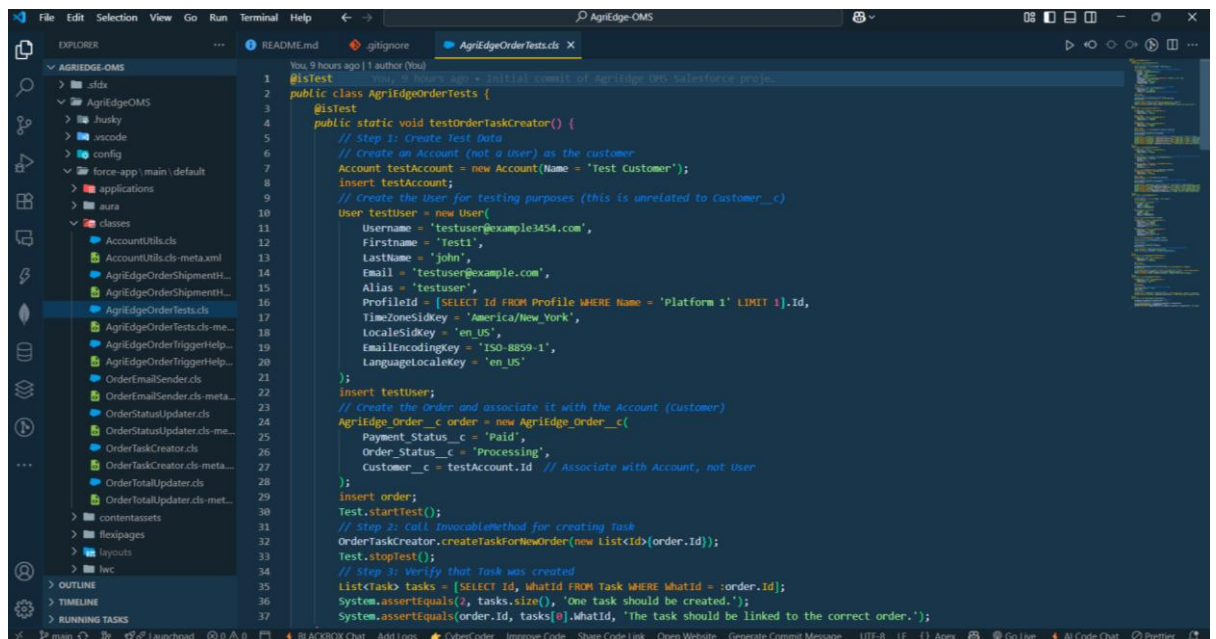


## Troubleshooting Approach

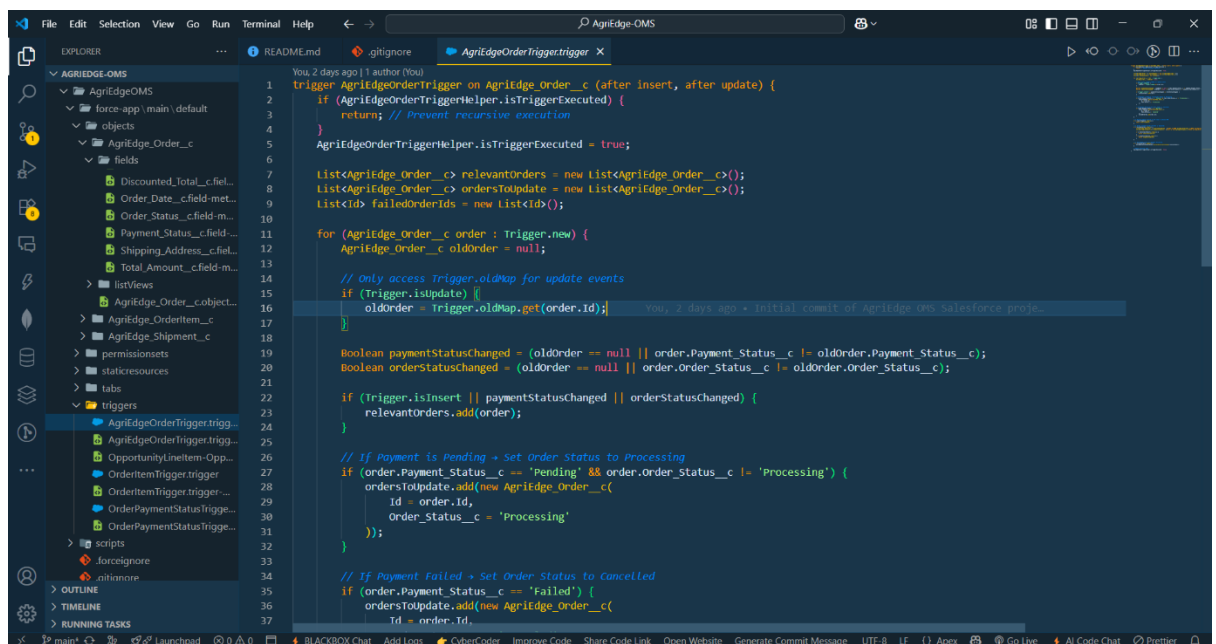
A detailed and efficient troubleshooting protocol includes:

1. **Reproduce Issue:** Duplicate the error scenario in a Scratch Org or Sandbox using sample data.
2. **Debug Logs:** Use the sf apex log tail or through Setup UI to inspect the log trail.
3. **Flow Debugging:** Use **Flow Debug Mode** and version control to inspect automation logic.
4. **Validation & FLS Checks:** Confirm no field-level security or validation conflicts.
5. **User Permission Issues:** Review **Profiles, Permission Sets, and Sharing Rules**.

All known bugs, exceptions, and fixes are documented in a **Troubleshooting Log**, available in the repository or admin folder.



```
1  You 9 hours ago | 1 author (You)
2  @!stest - Initial commit of AgriEdge OMS Salesforce proje...
3  public class AgriEdgeOrderTests {
4      @istest
5      public static void testOrderTaskCreator() {
6          // Step 1: Create Test Data
7          // Create an Account (not a User) as the customer
8          Account testAccount = new Account(Name = 'Test Customer');
9          insert testAccount;
10         // Create the User for testing purposes (this is unrelated to Customer__c)
11         User testUser = new User(
12             Username = 'testuser@example3454.com',
13             Firstname = 'Test1',
14             Lastname = 'John',
15             Email = 'testuser@example.com',
16             Alias = 'testuser',
17             ProfileId = (SELECT Id FROM Profile WHERE Name = 'Platform 1' LIMIT 1).Id,
18             TimezoneSidkey = 'America/New_York',
19             LocaleSidkey = 'en_US',
20             EmailEncodingKey = 'ISO-8859-1',
21             LanguageLocaleKey = 'en_US'
22         );
23         insert testUser;
24         // Create the Order and associate it with the Account (Customer)
25         AgriEdge_Order__c order = new AgriEdge_Order__c(
26             Payment_Status__c = 'Paid',
27             Order_Status__c = 'Processing',
28             Customer__c = testAccount.Id // Associate with Account, not User
29         );
30         insert order;
31         Test.startTest();
32         // Step 2: Call invocableMethod for creating task
33         OrderTaskCreator.createTaskForNewOrder(new List<Id>{order.Id});
34         Test.stopTest();
35         // Step 3: Verify that Task was created
36         List<Task> tasks = (SELECT Id, WhatId FROM Task WHERE WhatId = :order.Id);
37         System.assertEquals(2, tasks.size(), 'One task should be created.');
```



```
1  You, 2 days ago | 1 author (You)
2  trigger AgriEdgeOrderTrigger on AgriEdge_Order__c (after insert, after update) {
3      if (AgriEdgeOrderTriggerHelper.isTriggerExecuted) {
4          return; // Prevent recursive execution
5      }
6      AgriEdgeOrderTriggerHelper.isTriggerExecuted = true;
7
8      List<AgriEdge_Order__c> relevantOrders = new List<AgriEdge_Order__c>();
9      List<AgriEdge_Order__c> ordersToUpdate = new List<AgriEdge_Order__c>();
10     List<Id> failedOrderIds = new List<Id>();
11
12     for (AgriEdge_Order__c order : Trigger.new) {
13         AgriEdge_Order__c oldOrder = null;
14
15         // Only access Trigger.oldMap for update events
16         if (Trigger.isUpdate) {
17             oldOrder = Trigger.oldMap.get(order.Id);
18         }
19
20         Boolean paymentStatusChanged = (oldOrder == null || order.Payment_Status__c != oldOrder.Payment_Status__c);
21         Boolean orderStatusChanged = (oldOrder == null || order.Order_Status__c != oldOrder.Order_Status__c);
22
23         if (Trigger.isInsert || paymentStatusChanged || orderStatusChanged) {
24             relevantOrders.add(order);
25         }
26
27         // If Payment is Pending -> Set Order Status to Processing
28         if (order.Payment_Status__c == 'Pending' && order.Order_Status__c != 'Processing') {
29             ordersToUpdate.add(new AgriEdge_Order__c(
30                 Id = order.Id,
31                 Order_Status__c = 'Processing'
32             ));
33         }
34
35         // If Payment failed -> Set Order Status to Cancelled
36         if (order.Payment_Status__c == 'Failed') {
37             ordersToUpdate.add(new AgriEdge_Order__c(
38                 Id = order.Id,
```

- **Conclusion:**

The development of the **AgriEdge Order Management System (OMS)** on Salesforce marks a significant leap forward in streamlining agricultural operations for AgriEdge Or-Mange Ltd. From requirement analysis to deployment, each phase of the project was executed with a strong focus on scalability, automation, and user-centric design. The system effectively addresses the company's core challenges by automating order processing, enhancing inventory visibility, integrating customer support functions, and ensuring robust data security.

By leveraging Salesforce's advanced capabilities—including custom objects, process automation, Apex development, and insightful reporting—the OMS empowers users to manage end-to-end order lifecycles efficiently. Furthermore, the integration of DevOps and version control with GitHub ensures a sustainable development process, enabling future enhancements and smooth collaboration.

This project not only delivers a high-impact solution tailored to the business's needs but also demonstrates the power of a platform-based approach in revolutionizing traditional industries. With proper training, continuous monitoring, and periodic optimization, the AgriEdge OMS is well-positioned to support the company's growth and innovation in the agriculture and food production domain.