Salesforce Project

Problem statement:

Revolutionizing agriculture with agriedge or-mange ltd: A salesforce-Driven order management solution

AgriEdge Or-Mange Ltd, a prominent player in the agriculture and food production sector, is committed to transforming its order management processes through the implementation of a Salesforce-driven Order Management System (OMS). The company operates in a dynamic industry where efficient order processing, precise inventory tracking, and exceptional customer service are crucial for maintaining a competitive edge and ensuring customer satisfaction. AgriEdge Or-Mange Ltd handles a wide range of products, from seeds and fertilizers to harvested crops and processed food items, necessitating a robust system to manage the complexities of its supply chain operations.

The company currently faces challenges such as manual order processing errors, lack of real-time inventory visibility, and disjointed customer service channels, which can lead to delays, stockouts, and dissatisfied customers. To address these issues and enhance overall operational efficiency, AgriEdge Or-Mange Ltd has decided to leverage Salesforce's powerful platform to develop a customized OMS. This system will not only automate and streamline order processing but also provide real-time insights into inventory levels, facilitate seamless supply chain operations, and integrate with existing customer service channels to deliver a cohesive and responsive customer experience.

Requirements/solution:

To meet the company's objectives, the OMS must fulfill several critical requirements. Firstly, it needs to support automated order processing to minimize manual errors and enhance efficiency. This includes creating tasks for new orders, updating order statuses, and sending automated notifications to relevant stakeholders. Secondly, the system should offer real-time inventory tracking to ensure accurate stock levels and prompt reordering when necessary. This will help prevent stockouts and overstock situations, optimizing inventory management.

Thirdly, the OMS must integrate seamlessly with existing customer service channels, such as email, phone, and online portals, to provide a unified and responsive customer service experience. This integration will enable customer service representatives to access up-to-date order and inventory information, allowing them to resolve customer inquiries more effectively and promptly.

Data security and compliance with industry standards are also paramount. The system must protect sensitive information, such as customer details and order data, through robust security measures and ensure compliance with relevant regulations. Lastly, the OMS should provide robust reporting and analytics capabilities, offering insights into order trends, inventory levels, and supply chain performance. This data-driven approach will enable

Salesforce Project

AgriEdge Or-Mange Ltd to make informed decisions, optimize operations, and drive continuous improvement.

Use cases:

- Salesforce Data Modelling
- Formula fields and Validation Rules
- Salesforce Data Security
- User Management
- Automation using Process Builder
- Apex Class, Apex Triggers and Test Class

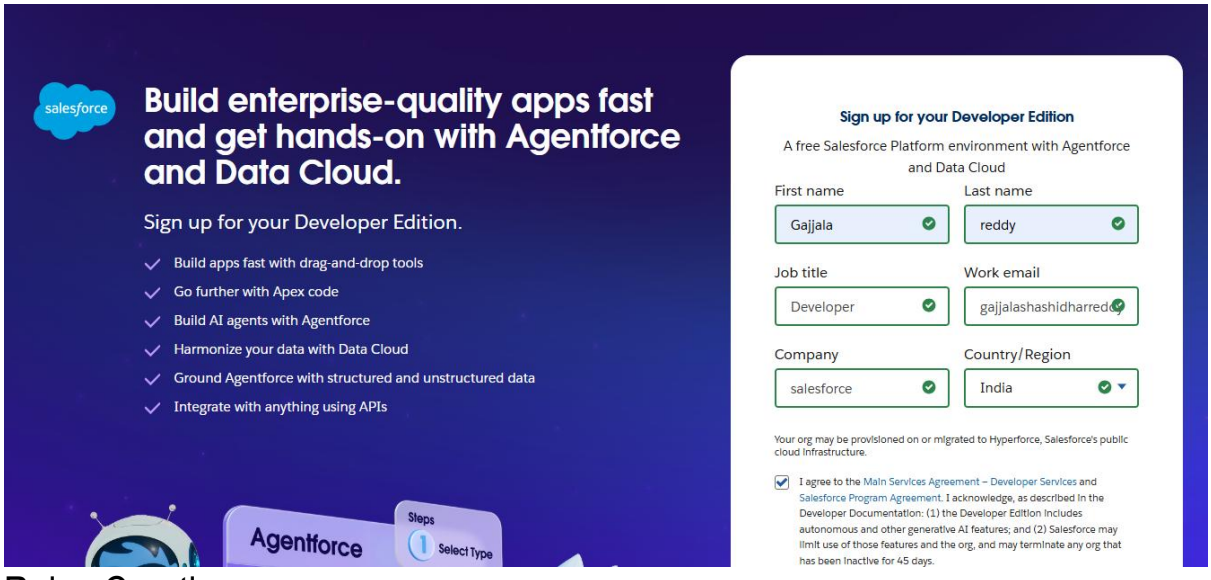
Overall Implementation:

AgriEdge Or-Mange Ltd aims to enhance agricultural supply chain efficiency using **Salesforce**. This project focuses on optimizing **order processing, inventory management, farmer support, and automated workflows** to ensure seamless operations in the agriculture sector.

1. Salesforce CRM Implementation
2. Process Automation & Workflows
3. Apex & Trigger Implementations
4. Batch Jobs & Scheduled Processes
5. Data Security & Access Control

Phase 2: Org Setup &Configuration:

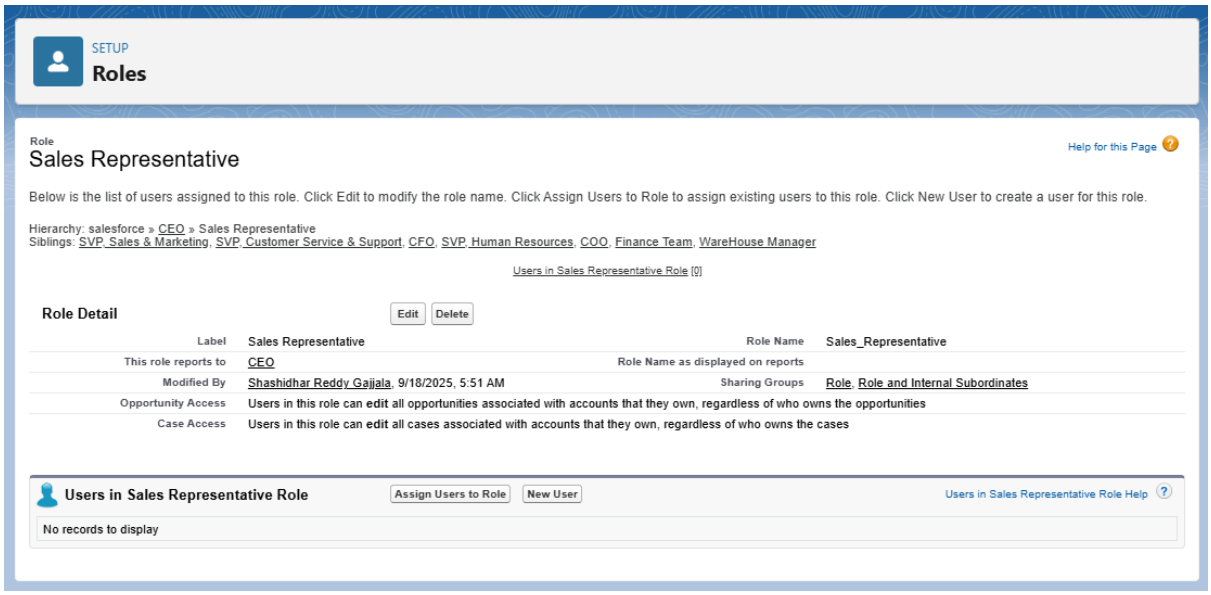
The developer org is created with the username: gajjalashashidharreddy866@agentforce.com



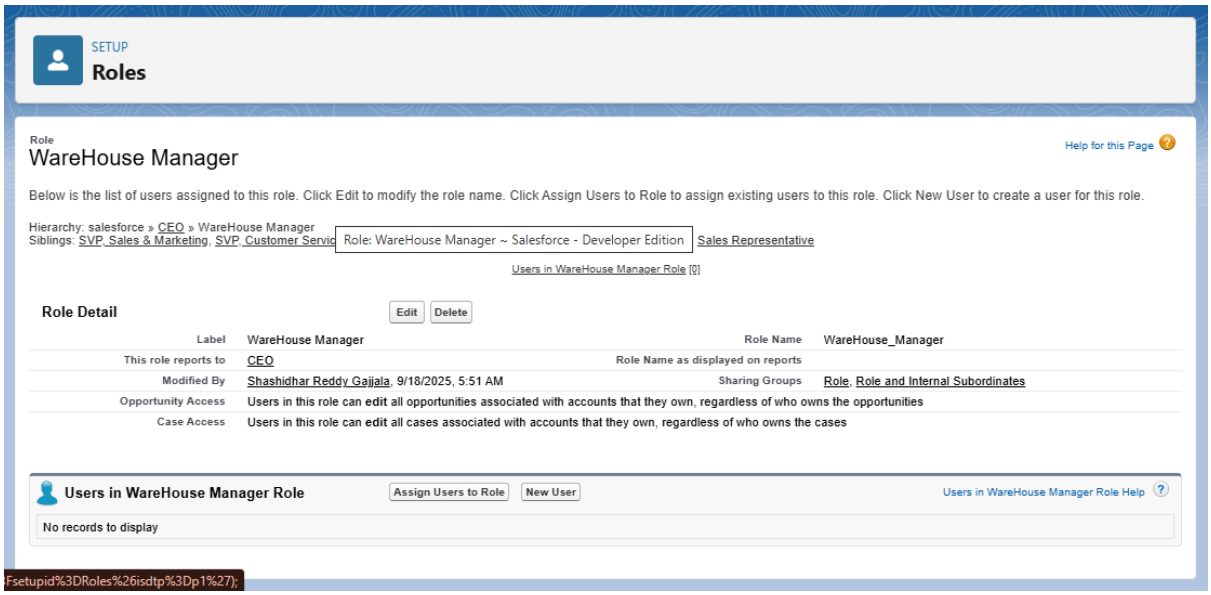
Roles Creation:

Salesforce Project

Role created for sales Representative

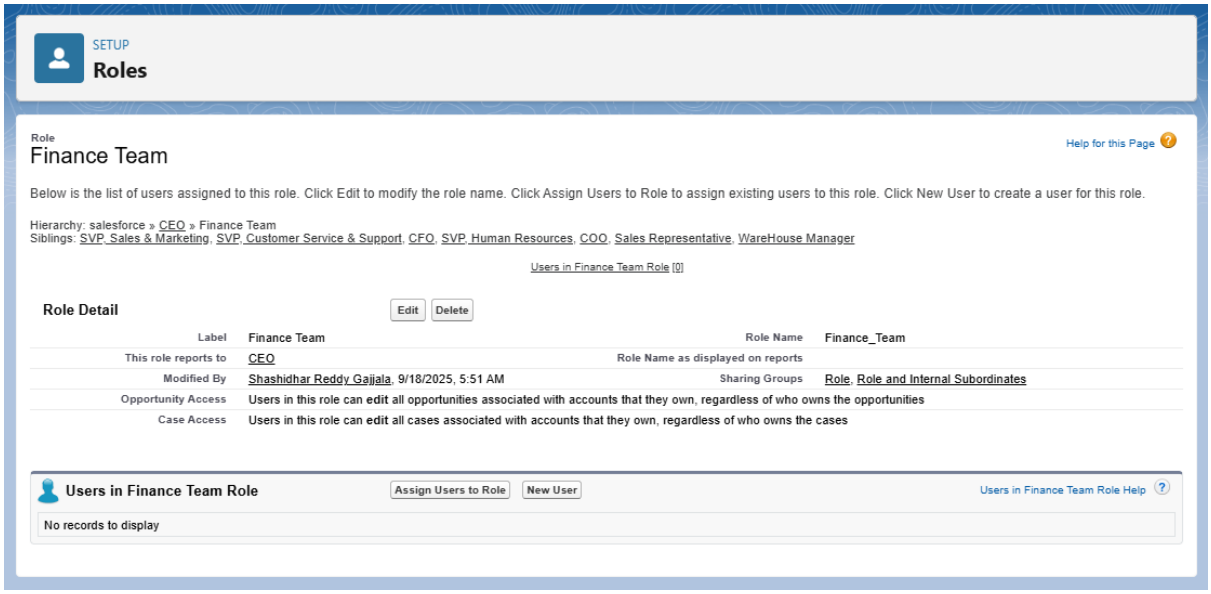


Role created for WareHouse Manager



Salesforce Project

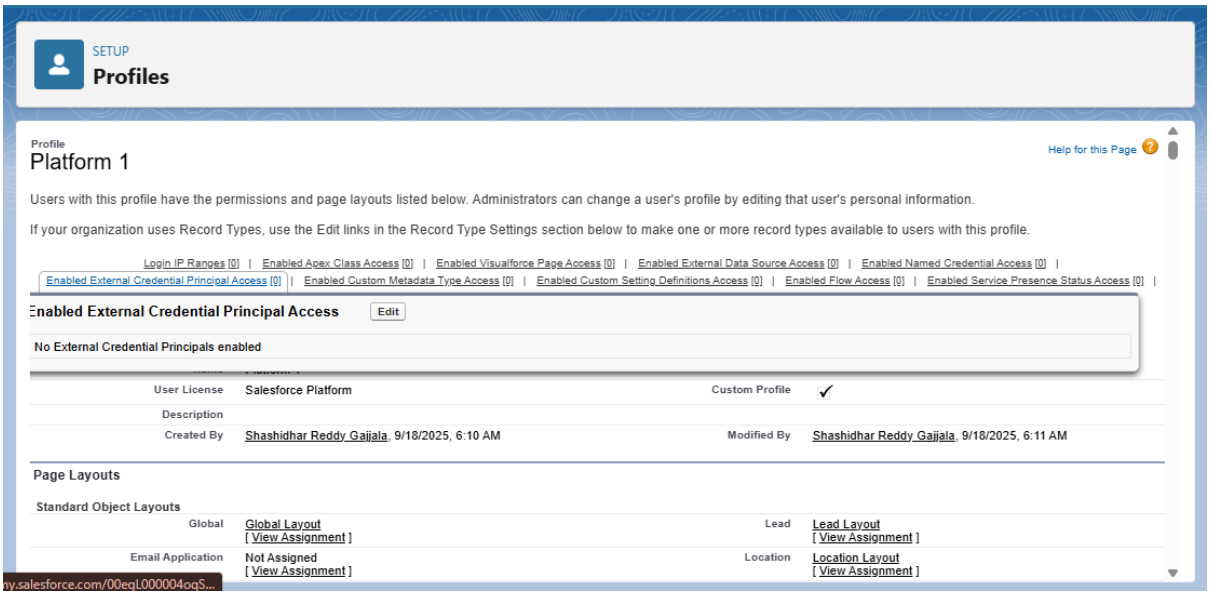
Role created for Finance Team



Profile creation:

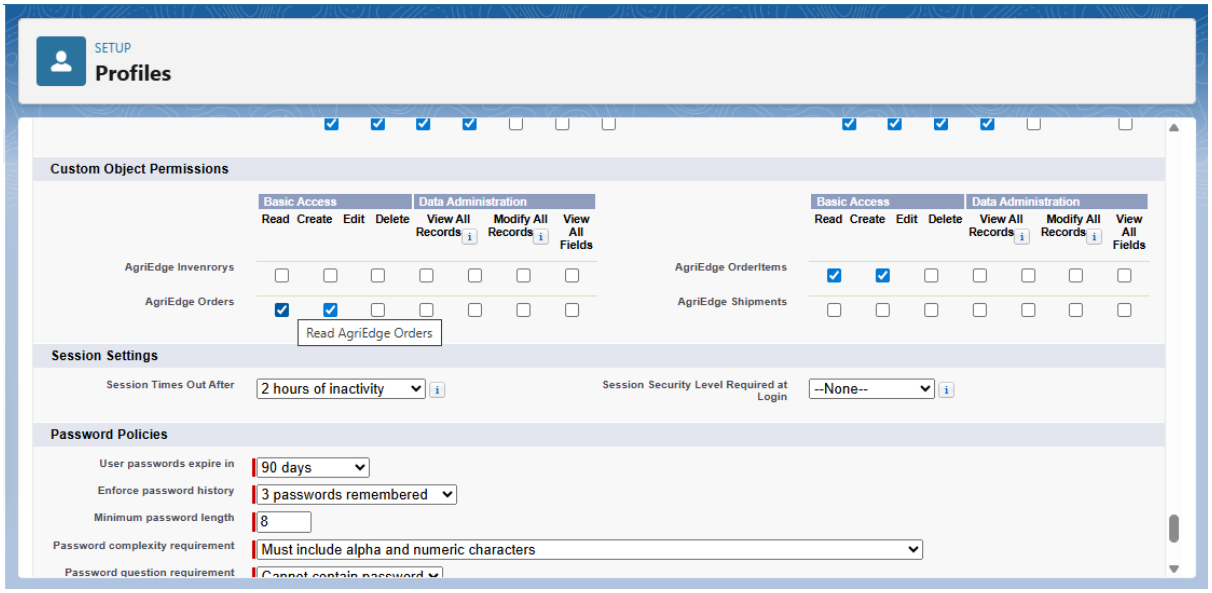
For the above created roles the profile is created and the custom objects created are added in those profiles

1. Platform 1:



Access for the profile platform 1 for the created custom objects:

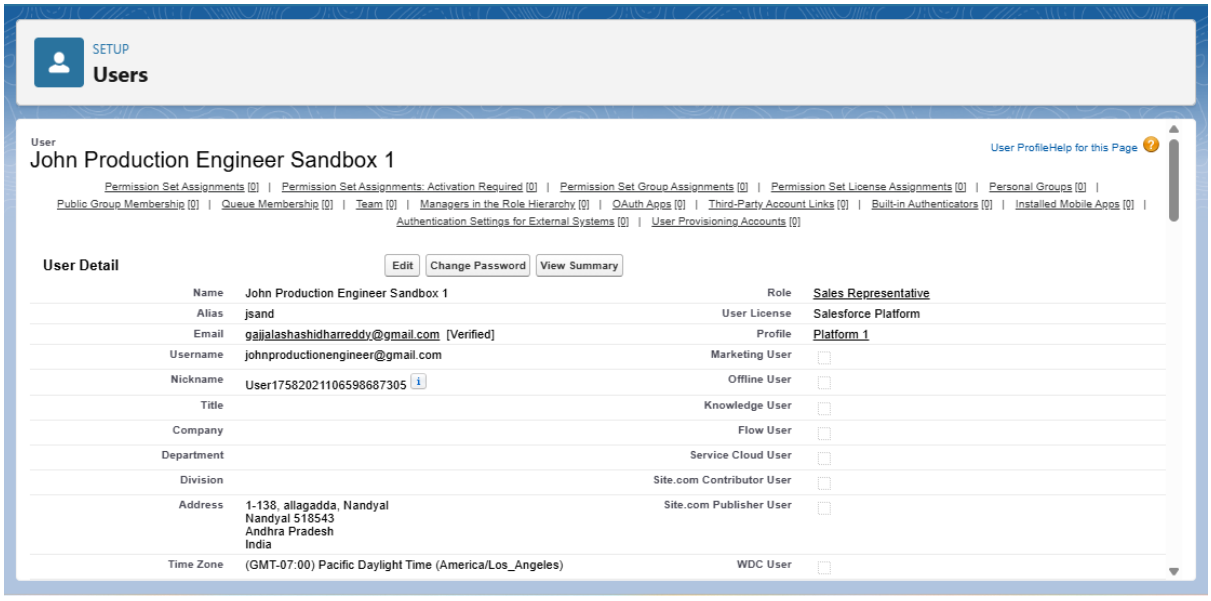
Salesforce Project



Similarly I have created the platform 2 and 3 and give access for the created custom objects

User Creation:

Users are created for the profiles created:



Similarly I have created the users Quality Inspector and plant Manger and updated with the profiles platform 2 and platform 3.

Phase 3: Data Modelling & Relationships

Custom Object creations:

1. AgriEdge_Order__c (Custom Object)

Order Number (Auto Number, Format: ORD-{0000})

Salesforce Project

Customer__c (Lookup to Account)

Order Status__c (Picklist: New, Processing, Shipped, Delivered, Canceled)

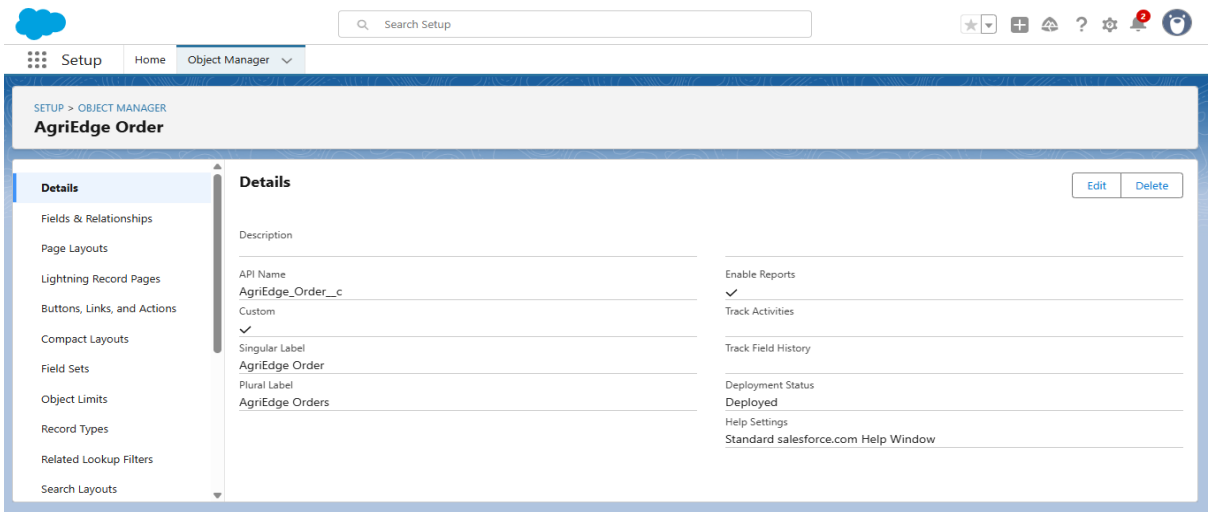
Order Date__c (Date/Time)

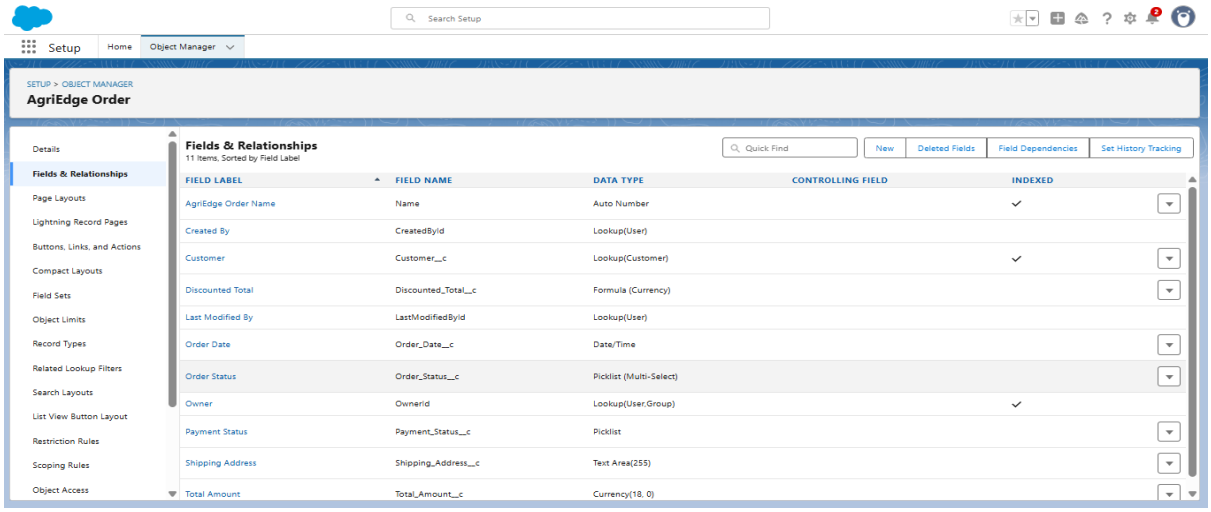
Total Amount__c (Currency)

Payment Status__c (Picklist: Pending, Paid, Failed)

Shipping Address__c (Text Area)

Discounted Total__c (Formula: Total_Amount__c - (Total_Amount__c * 0.1))





Salesforce Project

Similarly I have created the custom objects and their relation fields for the following objects.

2. AgriEdge_OrderItem__c (Custom Object)

Order__c (Lookup to AgriEdge_Order__c)

Product__c (Lookup to Product2)

Quantity__c (Number)

Unit Price__c (Currency)

Total Price__c (Formula: Quantity__c * Unit Price__c)

3. AgriEdge_Inventory__c (Custom Object)

Product__c (Lookup to Product2)

Stock Quantity__c (Number)

Reorder Level__c (Number)

Warehouse Location__c (Text)

Stock Status__c (Formula: IF(Stock_Quantity__c <= Reorder_Level__c, "Low", "Sufficient"))

4. AgriEdge_Shipment__c (Custom Object)

Order__c (Lookup to Order)

Tracking Number__c (Text)

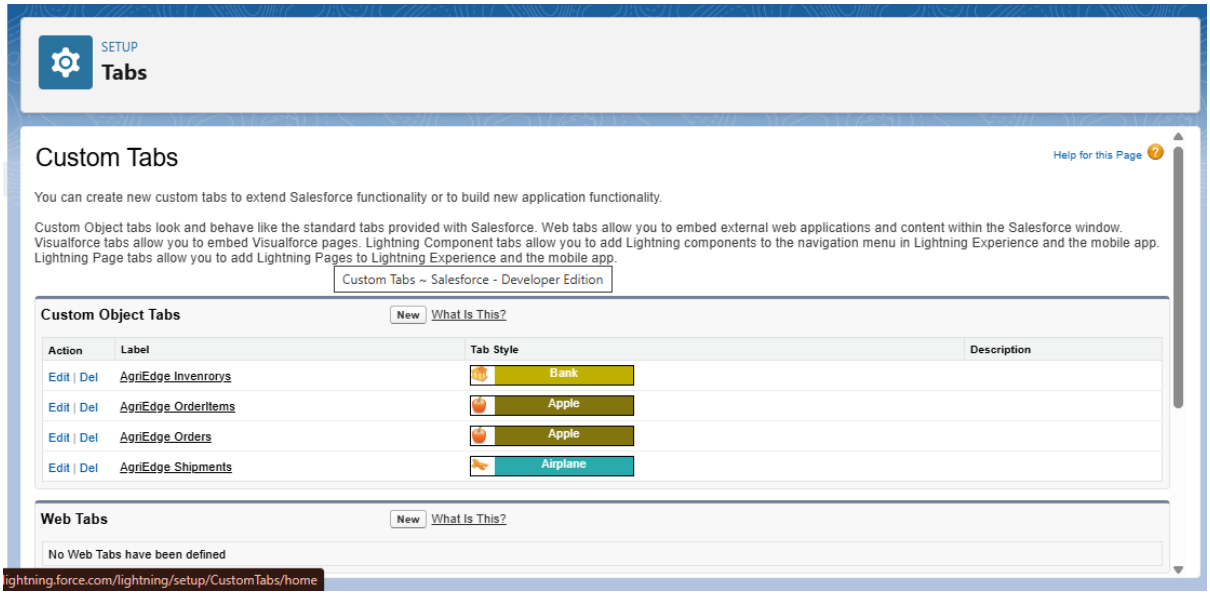
Carrier__c (Picklist: FedEx, UPS, DHL, Local Courier)

Status__c (Picklist: Pending, In Transit, Delivered)

Custom tab creation:

The new custom object tabs are created for the each custom object we have created.

Salesforce Project



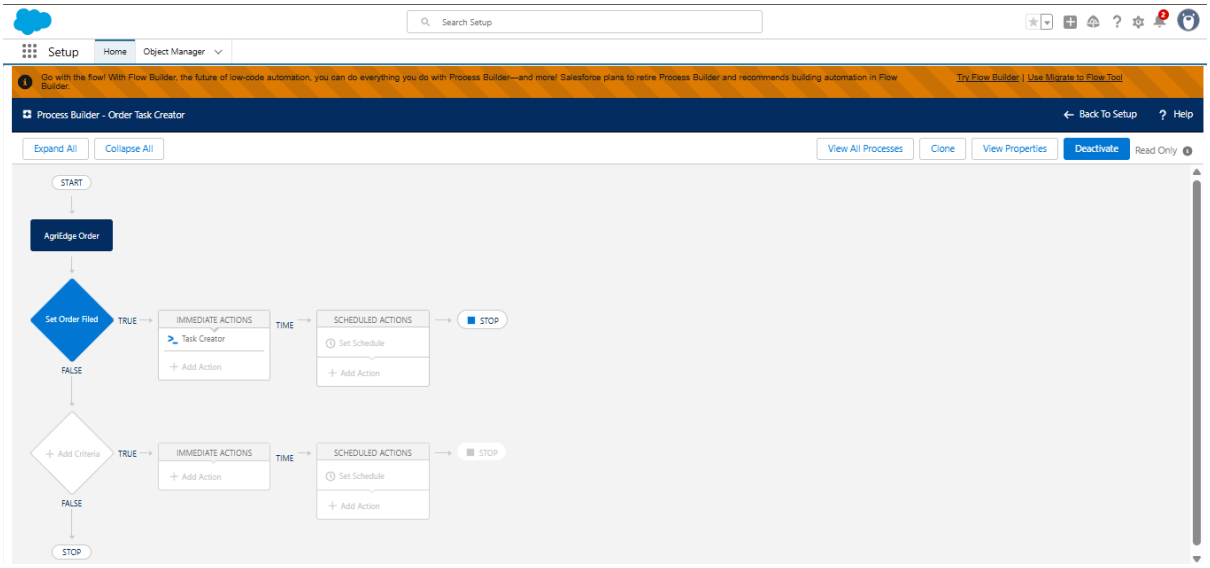
Similarly the custom object tab is created for the remaining custom objects we have created

Phase 4: Process Automation:

Created a process builder for the automation of the test when conditions are met

- Process Name : Order Task Creator
- The Process Starts When : A record Changes

And the AgriEdge Order Object and criteria are added.



The Process is activated.

Phase 5: Apex Programming:

Salesforce Project

Apex classes and triggers were created for the created custom objects:

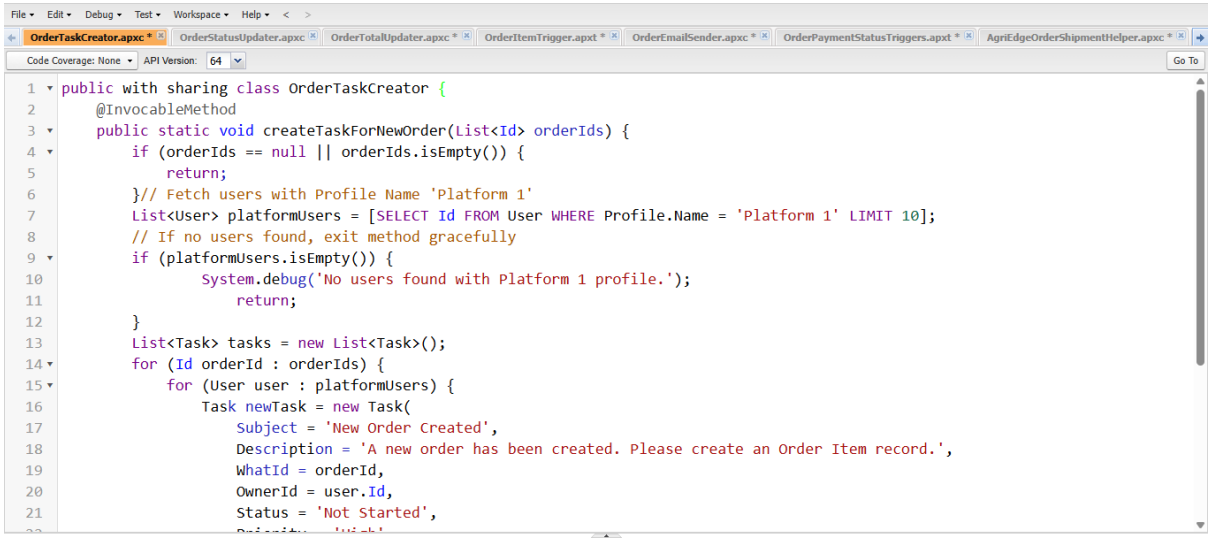
1. OrderTaskCreator Apex class:

Code:

```
public with sharing class OrderTaskCreator {

@InvocableMethod

public static void createTaskForNewOrder(List<Id> orderIds) {

if (orderIds == null || orderIds.isEmpty()) {

return;

}

// Fetch users with Profile Name 'Platform 1'

List<User> platformUsers = [SELECT Id FROM User WHERE Profile.Name = 'Platform 1' LIMIT 10];

// If no users found, exit method gracefully

if (platformUsers.isEmpty()) {

System.debug('No users found with Platform 1 profile.');

return;

}

List<Task> tasks = new List<Task>();

for (Id orderId : orderIds) {

for (User user : platformUsers) {

Task newTask = new Task(

Subject = 'New Order Created',

Description = 'A new order has been created. Please create an Order Item record.',

WhatId = orderId,

OwnerId = user.Id,

Status = 'Not Started',

Priority = 'High'

);
```

Salesforce Project

```
tasks.add(newTask);
}
}
 if (!tasks.isEmpty()) {
        insert tasks;
    }
  }
}
```



```
public with sharing class OrderTaskCreator {
    @InvocableMethod
    public static void createTaskForNewOrder(List<Id> orderIds) {
        if (orderIds == null || orderIds.isEmpty()) {
            return;
        }// Fetch users with Profile Name 'Platform 1'
        List<User> platformUsers = [SELECT Id FROM User WHERE Profile.Name = 'Platform 1' LIMIT 10];
        // If no users found, exit method gracefully
        if (platformUsers.isEmpty()) {
                System.debug('No users found with Platform 1 profile.');
                    return;
        }
        List<Task> tasks = new List<Task>();
        for (Id orderId : orderIds) {
            for (User user : platformUsers) {
                Task newTask = new Task(
                    Subject = 'New Order Created',
                    Description = 'A new order has been created. Please create an Order Item record.',
                    WhatId = orderId,
                    OwnerId = user.Id,
                    Status = 'Not Started',
```

2. OrderStatusUpdated Apex class

Code:

```
public class OrderStatusUpdater {
   public static void updateOrderStatus(Set<Id> orderIds) {
      if (orderIds == null || orderIds.isEmpty()) {
         return;
      }// Fetch Orders that are still in "New" status
      List<AgriEdge_Order__c> ordersToUpdate = [
         SELECT Id, Order_Status__c
         FROM AgriEdge_Order__c
```

Salesforce Project

```
        WHERE Id IN :orderIds AND Order_Status__c = 'New'
    ];
 if (!ordersToUpdate.isEmpty()) {
        for (AgriEdge_Order__c order : ordersToUpdate) {
            order.Order_Status__c = 'Processing';
        }
        update ordersToUpdate;
    }
  }
}
```

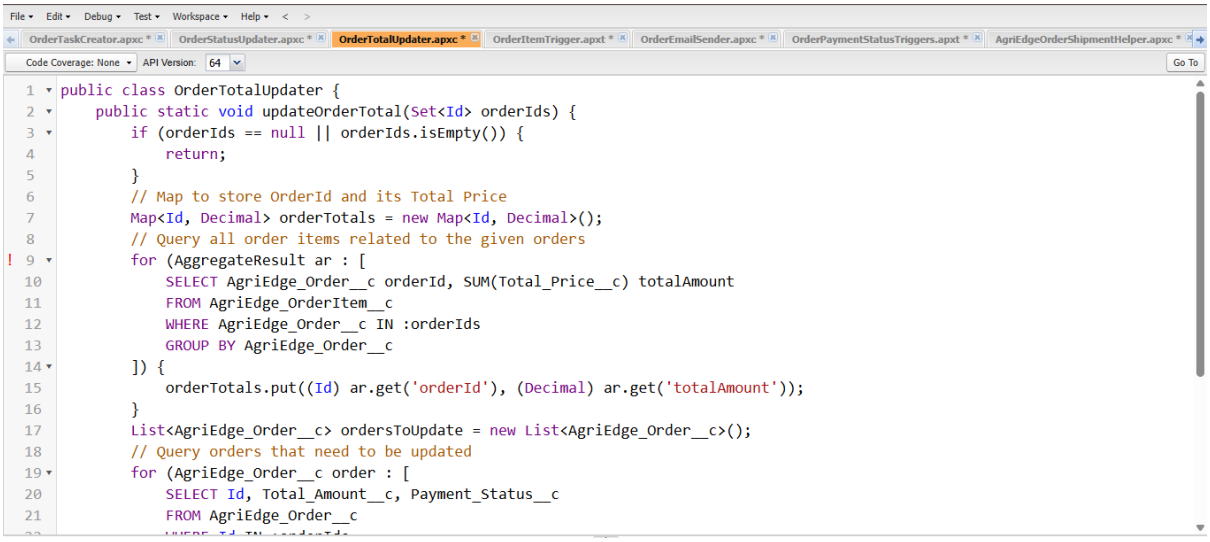
3. OrderTotalUpdater Apex class

Code:

```
public class OrderTotalUpdater {
   public static void updateOrderTotal(Set<Id> orderIds) {
      if (orderIds == null || orderIds.isEmpty()) {
         return;
      }
      // Map to store OrderId and its Total Price
```

```
        Map<Id, Decimal> orderTotals = new Map<Id, Decimal>();
        // Query all order items related to the given orders
        for (AggregateResult ar : [
            SELECT AgriEdge_Order__c orderId, SUM(Total_Price__c) totalAmount
            FROM AgriEdge_OrderItem__c
            WHERE AgriEdge_Order__c IN :orderIds
            GROUP BY AgriEdge_Order__c
        ]) {
            orderTotals.put((Id) ar.get('orderId'), (Decimal) ar.get('totalAmount'));
        }
        List<AgriEdge_Order__c> ordersToUpdate = new List<AgriEdge_Order__c>();
        // Query orders that need to be updated
        for (AgriEdge_Order__c order : [
            SELECT Id, Total_Amount__c, Payment_Status__c
            FROM AgriEdge_Order__c
            WHERE Id IN :orderIds
        ]) {
            order.Total_Amount__c = orderTotals.containsKey(order.Id) ?
orderTotals.get(order.Id) : 0;
            order.Payment_Status__c = (order.Total_Amount__c > 0) ? 'Pending' : 'Paid'; // If
total > 0, set to Pending
            ordersToUpdate.add(order);
        }
        if (!ordersToUpdate.isEmpty()) {
            update ordersToUpdate;
        }
    }
}
```

Salesforce Project



```apex
public class OrderTotalUpdater {
    public static void updateOrderTotal(Set<Id> orderIds) {
        if (orderIds == null || orderIds.isEmpty()) {
            return;
        }
        // Map to store OrderId and its Total Price
        Map<Id, Decimal> orderTotals = new Map<Id, Decimal>();
        // Query all order items related to the given orders
        for (AggregateResult ar : [
            SELECT AgriEdge_Order__c orderId, SUM(Total_Price__c) totalAmount
            FROM AgriEdge_OrderItem__c
            WHERE AgriEdge_Order__c IN :orderIds
            GROUP BY AgriEdge_Order__c
        ]) {
            orderTotals.put((Id) ar.get('orderId'), (Decimal) ar.get('totalAmount'));
        }
        List<AgriEdge_Order__c> ordersToUpdate = new List<AgriEdge_Order__c>();
        // Query orders that need to be updated
        for (AgriEdge_Order__c order : [
            SELECT Id, Total_Amount__c, Payment_Status__c
            FROM AgriEdge_Order__c
```

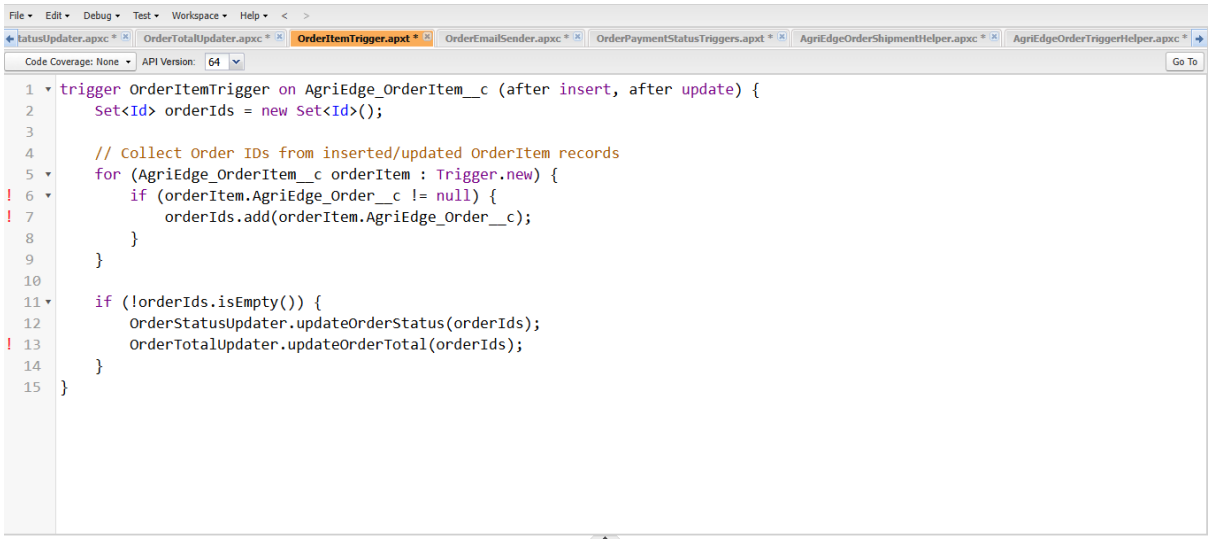
4. Apex Trigger:

- Name : OrderItemTrigger
- Object : AgriEdge_OrderItem__C
- Click Submit Button

Code:

```
trigger OrderItemTrigger on AgriEdge_OrderItem__c (after insert, after update) {
    Set<Id> orderIds = new Set<Id>();
    // Collect Order IDs from inserted/updated OrderItem records
    for (AgriEdge_OrderItem__c orderItem : Trigger.new) {
        if (orderItem.AgriEdge_Order__c != null) {
            orderIds.add(orderItem.AgriEdge_Order__c);
        }
    }
    if (!orderIds.isEmpty()) {
        OrderStatusUpdater.updateOrderStatus(orderIds);
        OrderTotalUpdater.updateOrderTotal(orderIds);
    }
}
```

Salesforce Project



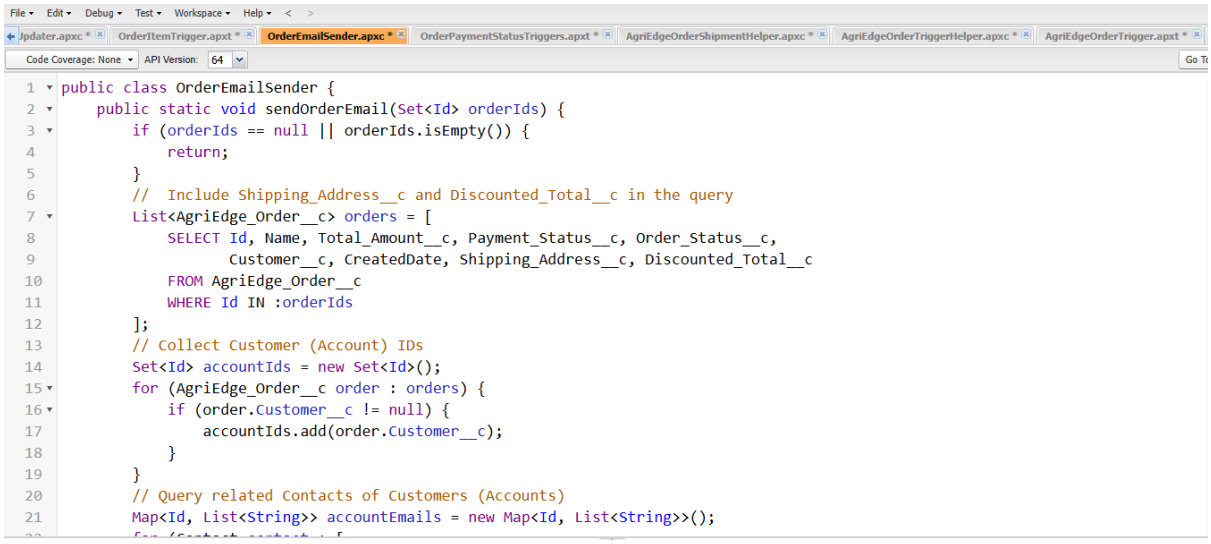
5. OrderEmailSender Apex class

Code:

```
public class OrderEmailSender {
   public static void sendOrderEmail(Set<Id> orderIds) {
      if (orderIds == null || orderIds.isEmpty()) {
         return;
      }
      //  Include Shipping_Address__c and Discounted_Total__c in the query
      List<AgriEdge_Order__c> orders = [
         SELECT Id, Name, Total_Amount__c, Payment_Status__c, Order_Status__c,
              Customer__c, CreatedDate, Shipping_Address__c, Discounted_Total__c
         FROM AgriEdge_Order__c
         WHERE Id IN :orderIds
      ];
      // Collect Customer (Account) IDs
      Set<Id> accountIds = new Set<Id>();
      for (AgriEdge_Order__c order : orders) {
         if (order.Customer__c != null) {
            accountIds.add(order.Customer__c);
```

```
        }
    }
    // Query related Contacts of Customers (Accounts)
    Map<Id, List<String>> accountEmails = new Map<Id, List<String>>();
    for (Contact contact : [
        SELECT Email, AccountId FROM Contact WHERE AccountId IN :accountIds AND
Email != null
    ]) {
        if (!accountEmails.containsKey(contact.AccountId)) {
            accountEmails.put(contact.AccountId, new List<String>());
        }
        accountEmails.get(contact.AccountId).add(contact.Email);
    }
    // Prepare Emails
    List<Messaging.SingleEmailMessage> emails = new
List<Messaging.SingleEmailMessage>();
    for (AgriEdge_Order__c order : orders) {
        if (accountEmails.containsKey(order.Customer__c)) {
            List<String> toEmails = accountEmails.get(order.Customer__c);
            // Create Email Body
            String emailBody = 'Dear Customer,<br/><br/>'
                + 'Your order has been updated with the following details:<br/><br/>'
                + '<b>Order Name:</b> ' + order.Name + '<br/>'
                + '<b>Order Status:</b> ' + order.Order_Status__c + '<br/>'
                + '<b>Total Amount:</b> ' + order.Total_Amount__c + '<br/>'
                + '<b>Payment Status:</b> ' + order.Payment_Status__c + '<br/>'
                + '<b>Shipping Address:</b> ' + order.Shipping_Address__c + '<br/>'
                + '<b>Created Date:</b> ' + order.CreatedDate + '<br/><br/>'
```

```
                + '<b>Total Amount Paid (Including Discount):</b> ' +
order.Discounted_Total__c + '<br/><br/>'
                + 'Thank you for your business!<br/><br/>'
                + 'Best Regards,<br/>Your Company Name';
            // Prepare Email
            Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
            email.setToAddresses(toEmails);
            email.setSubject('Your Order Payment Status has been Updated');
            email.setHtmlBody(emailBody);
            emails.add(email);
        }
    }
    if (!emails.isEmpty()) {
        Messaging.sendEmail(emails);
    }
  }
}
```




```
public class OrderEmailSender {
    public static void sendOrderEmail(Set<Id> orderIds) {
        if (orderIds == null || orderIds.isEmpty()) {
            return;
        }
        //  Include Shipping_Address__c and Discounted_Total__c in the query
        List<AgriEdge_Order__c> orders = [
            SELECT Id, Name, Total_Amount__c, Payment_Status__c, Order_Status__c,
                   Customer__c, CreatedDate, Shipping_Address__c, Discounted_Total__c
            FROM AgriEdge_Order__c
            WHERE Id IN :orderIds
        ];
        // Collect Customer (Account) IDs
        Set<Id> accountIds = new Set<Id>();
        for (AgriEdge_Order__c order : orders) {
            if (order.Customer__c != null) {
                accountIds.add(order.Customer__c);
            }
        }
        // Query related Contacts of Customers (Accounts)
        Map<Id, List<String>> accountEmails = new Map<Id, List<String>>();
```

6. Apex Trigger:

- Create Trigger Class

Salesforce Project

- Class Name : OrderPaymentStatusTriggers
- Object : AgriEdge_Order__c

Code:

```
trigger OrderPaymentStatusTriggers on AgriEdge_Order__c (after update) {
   Set<Id> updatedOrderIds = new Set<Id>();
   // Collect Orders where Payment_Status__c was changed to "Paid"
   for (AgriEdge_Order__c order : Trigger.new) {
      AgriEdge_Order__c oldOrder = Trigger.oldMap.get(order.Id);
      if (oldOrder.Payment_Status__c != 'Paid' && order.Payment_Status__c == 'Paid' &&
order.Customer__c != null) {
         updatedOrderIds.add(order.Id);
      }
   }
   if (!updatedOrderIds.isEmpty()) {
      OrderEmailSender.sendOrderEmail(updatedOrderIds);
   }
}
```

Salesforce Project

7. AgriEdgeOrderShipmentHelper Apex class

Code:

```
public class AgriEdgeOrderShipmentHelper {

   public static void processOrderStatusChange(List<AgriEdge_Order__c> updatedOrders)
{

      List<AgriEdge_Shipment__c> shipmentsToInsert = new
List<AgriEdge_Shipment__c>();

      List<AgriEdge_Shipment__c> shipmentsToUpdate = new
List<AgriEdge_Shipment__c>();

      List<AgriEdge_Order__c> ordersToUpdate = new List<AgriEdge_Order__c>();

      List<AgriEdge_OrderItem__c> orderItemsToDelete = new
List<AgriEdge_OrderItem__c>();

      List<AgriEdge_Shipment__c> shipmentsToDelete = new
List<AgriEdge_Shipment__c>();

      Set<Id> orderIds = new Set<Id>();

      for (AgriEdge_Order__c order : updatedOrders) {

         orderIds.add(order.Id);

      }

      Map<Id, AgriEdge_Shipment__c> existingShipments = new Map<Id,
AgriEdge_Shipment__c>();

      for (AgriEdge_Shipment__c shipment : [

         SELECT Id, AgriEdge_Order__c, Status__c

         FROM AgriEdge_Shipment__c

         WHERE AgriEdge_Order__c IN :orderIds

      ]) {

         existingShipments.put(shipment.AgriEdge_Order__c, shipment);

      }


      Map<Id, List<AgriEdge_OrderItem__c>> existingOrderItems = new Map<Id,
List<AgriEdge_OrderItem__c>>();
```

```
        for (AgriEdge_OrderItem__c orderItem : [
            SELECT Id, AgriEdge_Order__c
            FROM AgriEdge_OrderItem__c
            WHERE AgriEdge_Order__c IN :orderIds
        ]) {
            if (!existingOrderItems.containsKey(orderItem.AgriEdge_Order__c)) {
                existingOrderItems.put(orderItem.AgriEdge_Order__c, new
List<AgriEdge_OrderItem__c>());
            }
            existingOrderItems.get(orderItem.AgriEdge_Order__c).add(orderItem);
        }
        for (AgriEdge_Order__c order : updatedOrders) {
            AgriEdge_Order__c updatedOrder = order.clone(false, true, false, false);
            updatedOrder.Id = order.Id;
            if (order.Payment_Status__c == 'Paid' && order.Order_Status__c != 'Delivered') {
                updatedOrder.Order_Status__c = 'Delivered';
                ordersToUpdate.add(updatedOrder);
            }
            else if (order.Payment_Status__c == 'Pending') {
                updatedOrder.Order_Status__c = 'Processing';
                ordersToUpdate.add(updatedOrder);
            }
            else if (order.Payment_Status__c == 'Failed') {
                updatedOrder.Order_Status__c = 'Canceled';
                ordersToUpdate.add(updatedOrder);
                if (existingOrderItems.containsKey(order.Id)) {
                    orderItemsToDelete.addAll(existingOrderItems.get(order.Id));
                }
```

```
            if (existingShipments.containsKey(order.Id)) {
                shipmentsToDelete.add(existingShipments.get(order.Id));
            }
        }
        if (order.Order_Status__c == 'Processing' &&
!existingShipments.containsKey(order.Id)) {
            AgriEdge_Shipment__c newShipment = new AgriEdge_Shipment__c(
                AgriEdge_Order__c = order.Id,
                Tracking_Number__c = 'TEST_' + order.Id,
                Status__c = 'Pending'
            );
            shipmentsToInsert.add(newShipment);
        }
        else if (order.Order_Status__c == 'Shipped' || order.Order_Status__c ==
'Delivered') {
            if (existingShipments.containsKey(order.Id)) {
                AgriEdge_Shipment__c shipmentToUpdate = existingShipments.get(order.Id);
                shipmentToUpdate.Status__c = (order.Order_Status__c == 'Shipped') ? 'In
Transit' : 'Delivered';
                shipmentsToUpdate.add(shipmentToUpdate);
            }
        }
    }

    if (!ordersToUpdate.isEmpty()) {
        update ordersToUpdate;
    }
    if (!shipmentsToInsert.isEmpty()) {
        insert shipmentsToInsert;
```

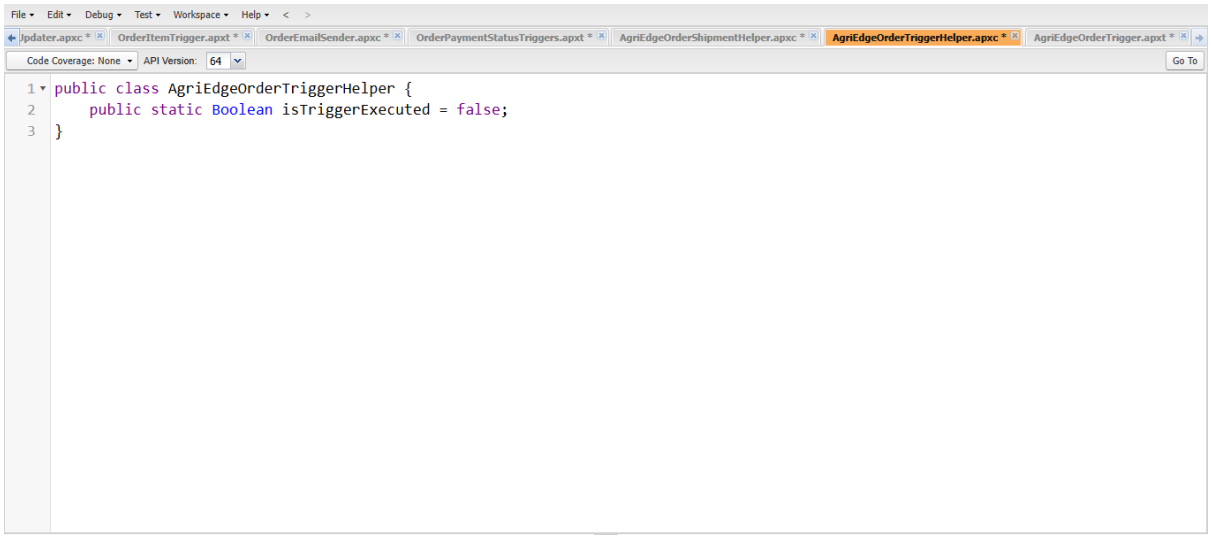
```
        }
        if (!shipmentsToUpdate.isEmpty()) {
            update shipmentsToUpdate;
        }
        if (!orderItemsToDelete.isEmpty()) {
            delete orderItemsToDelete;
        }
        if (!shipmentsToDelete.isEmpty()) {
            delete shipmentsToDelete;
        }
    }
}
```

8. AgriEdgeOrderTriggerHelper Apex class

Code:

```
public class AgriEdgeOrderTriggerHelper {
    public static Boolean isTriggerExecuted = false;
}
```

Salesforce Project



9. Apex trigger:

- Class Name : AgriEdgeOrderTrigger
- Object : AgriEdge_Order__c

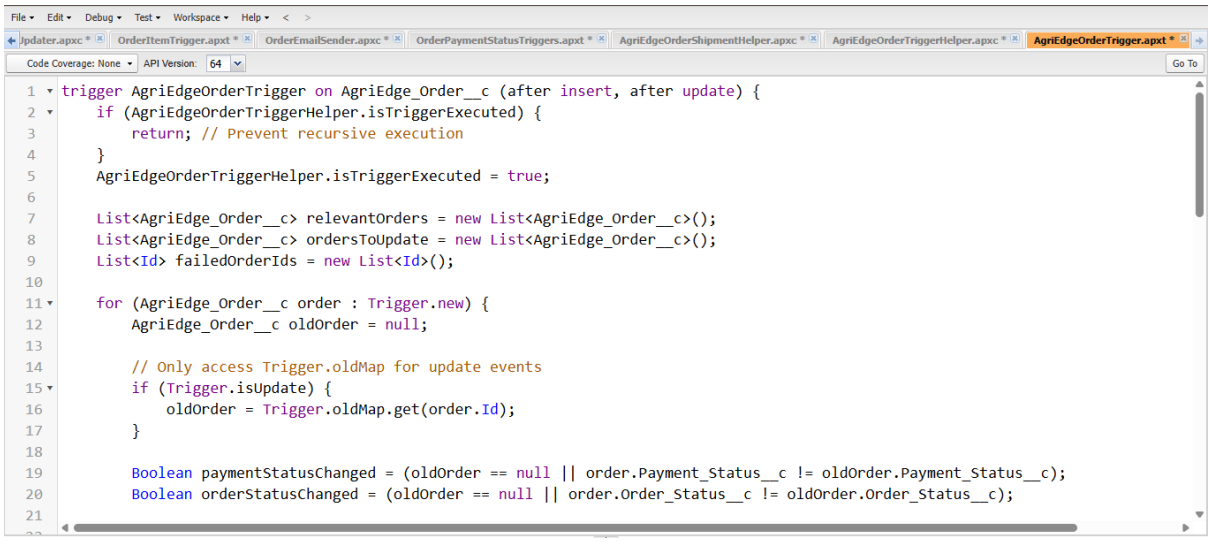
Code:

```
trigger AgriEdgeOrderTrigger on AgriEdge_Order__c (after insert, after update) {
  if (AgriEdgeOrderTriggerHelper.isTriggerExecuted) {
    return; // Prevent recursive execution
  }
  AgriEdgeOrderTriggerHelper.isTriggerExecuted = true;

  List<AgriEdge_Order__c> relevantOrders = new List<AgriEdge_Order__c>();
  List<AgriEdge_Order__c> ordersToUpdate = new List<AgriEdge_Order__c>();
  List<Id> failedOrderIds = new List<Id>();

  for (AgriEdge_Order__c order : Trigger.new) {
    AgriEdge_Order__c oldOrder = null;

    // Only access Trigger.oldMap for update events
    if (Trigger.isUpdate) {
      oldOrder = Trigger.oldMap.get(order.Id);
    }

    Boolean paymentStatusChanged = (oldOrder == null || order.Payment_Status__c
!= oldOrder.Payment_Status__c);
    Boolean orderStatusChanged = (oldOrder == null || order.Order_Status__c !=
oldOrder.Order_Status__c);

    if (Trigger.isInsert || paymentStatusChanged || orderStatusChanged) {
```

```
            relevantOrders.add(order);
        }

        // If Payment is Pending → Set Order Status to Processing
        if (order.Payment_Status__c == 'Pending' && order.Order_Status__c !=
'Processing') {
            ordersToUpdate.add(new AgriEdge_Order__c(
                Id = order.Id,
                Order_Status__c = 'Processing'
            ));
        }

        // If Payment Failed → Set Order Status to Cancelled
        if (order.Payment_Status__c == 'Failed') {
            ordersToUpdate.add(new AgriEdge_Order__c(
                Id = order.Id,
                Order_Status__c = 'Canceled'
            ));
            failedOrderIds.add(order.Id);
        }
    }

    // ✅ Perform updates outside of the loop to optimize DML
    if (!ordersToUpdate.isEmpty()) {
        update ordersToUpdate;
    }

    // ✅ Delete related records if Order is Cancelled
    if (!failedOrderIds.isEmpty()) {
        List<AgriEdge_OrderItem__c> orderItemsToDelete = [SELECT Id FROM
AgriEdge_OrderItem__c WHERE AgriEdge_Order__c IN :failedOrderIds];
        List<AgriEdge_Shipment__c> shipmentsToDelete = [SELECT Id FROM
AgriEdge_Shipment__c WHERE AgriEdge_Order__c IN :failedOrderIds];

        if (!orderItemsToDelete.isEmpty()) {
            delete orderItemsToDelete;
        }
        if (!shipmentsToDelete.isEmpty()) {
            delete shipmentsToDelete;
        }
    }
```

Salesforce Project

```
    // ✅ Call Helper Class for Shipment Processing
    if (!relevantOrders.isEmpty()) {
        AgriEdgeOrderShipmentHelper.processOrderStatusChange(relevantOrders);
    }

    // Reset recursion flag
    AgriEdgeOrderTriggerHelper.isTriggerExecuted = false;
}
```



Created Apex Test Class for all Apex classes

Apex Test Class: AgriEdgeOrderTests

Ode:

```
@isTest
public class AgriEdgeOrderTests {
    @isTest
    public static void testOrderTaskCreator() {
        // Step 1: Create Test Data
        // Create an Account (not a User) as the customer
        Account testAccount = new Account(Name = 'Test Customer');
        insert testAccount;
        // Create the User for testing purposes (this is unrelated to Customer__c)
        User testUser = new User(
            Username = 'testuser@example3454.com',
            Firstname = 'Test1',
            LastName = 'john',
            Email = 'testuser@example.com',
            Alias = 'testuser',
            ProfileId = [SELECT Id FROM Profile WHERE Name = 'Platform 1' LIMIT 1].Id,
            TimeZoneSidKey = 'America/New_York',
```

```
        LocaleSidKey = 'en_US',
        EmailEncodingKey = 'ISO-8859-1',
        LanguageLocaleKey = 'en_US'
      );
      insert testUser;
      // Create the Order and associate it with the Account (Customer)
      AgriEdge_Order__c order = new AgriEdge_Order__c(
        Payment_Status__c = 'Paid',
        Order_Status__c = 'Processing',
        Customer__c = testAccount.Id  // Associate with Account, not User
      );
      insert order;
      Test.startTest();
      // Step 2: Call InvocableMethod for creating Task
      OrderTaskCreator.createTaskForNewOrder(new List<Id>{order.Id});
      Test.stopTest();
      // Step 3: Verify that Task was created
      List<Task> tasks = [SELECT Id, WhatId FROM Task WHERE WhatId = :order.Id];
      System.assertEquals(2, tasks.size(), 'One task should be created.');
      System.assertEquals(order.Id, tasks[0].WhatId, 'The task should be linked to the
correct order.');
    }
   @isTest
    public static void testOrderStatusUpdater() {
      // Step 1: Create Test Data
      AgriEdge_Order__c order1 = new AgriEdge_Order__c(
        Order_Status__c = 'New',
        Payment_Status__c = 'Pending'
      );
      insert order1;
      AgriEdge_Order__c order2 = new AgriEdge_Order__c(
        Order_Status__c = 'Processing',
        Payment_Status__c = 'Pending'
      );
      insert order2;
      AgriEdge_Order__c order3 = new AgriEdge_Order__c(
        Order_Status__c = 'Canceled',
        Payment_Status__c = 'Failed'
      );
      insert order3;
      Set<Id> orderIds = new Set<Id>{order1.Id, order2.Id, order3.Id};
```

```
        Test.startTest();
        // Call the OrderStatusUpdater method to update the order status
        OrderStatusUpdater.updateOrderStatus(orderIds);
        Test.stopTest();
        // Step 2: Verify that the order status was updated to 'Processing' for the orders where applicable
        AgriEdge_Order__c updatedOrder1 = [SELECT Order_Status__c FROM AgriEdge_Order__c WHERE Id = :order1.Id];
        System.assertEquals('Processing', updatedOrder1.Order_Status__c, 'Order status for order1 should be updated to Processing.');
        AgriEdge_Order__c updatedOrder2 = [SELECT Order_Status__c FROM AgriEdge_Order__c WHERE Id = :order2.Id];
        System.assertEquals('Processing', updatedOrder2.Order_Status__c, 'Order status for order2 should remain Processing.');
        AgriEdge_Order__c updatedOrder3 = [SELECT Order_Status__c FROM AgriEdge_Order__c WHERE Id = :order3.Id];
        System.assertEquals('Canceled', updatedOrder3.Order_Status__c, 'Order status for order3 should remain Canceled.');
    }
    @isTest
    public static void testOrderTotalUpdater() {
        // Step 1: Create Test Data
        AgriEdge_Order__c order1 = new AgriEdge_Order__c(
            Order_Status__c = 'New',
            Payment_Status__c = 'Pending'
        );
        insert order1;
        // Insert Order Items for order1 (Simulating Total_Price__c calculation via fields like Unit_Price and Quantity)
        AgriEdge_OrderItem__c orderItem1 = new AgriEdge_OrderItem__c(
            AgriEdge_Order__c = order1.Id,
            Quantity__c = 2,
            Unit_Price__c = 25.0
        );
        AgriEdge_OrderItem__c orderItem2 = new AgriEdge_OrderItem__c(
            AgriEdge_Order__c = order1.Id,
            Quantity__c = 1,
            Unit_Price__c = 30.0
        );
        insert new List<AgriEdge_OrderItem__c>{orderItem1, orderItem2};
        AgriEdge_Order__c order2 = new AgriEdge_Order__c(
```

```
        Order_Status__c = 'New',
        Payment_Status__c = 'Pending'
    );
    insert order2;
    Set<Id> orderIds = new Set<Id>{order1.Id, order2.Id};
    Test.startTest();
    // Call the OrderTotalUpdater method to update the order totals and payment status
    OrderTotalUpdater.updateOrderTotal(orderIds);
    Test.stopTest();
    // Step 2: Verify that the total amount and payment status were updated for order1 and order2
    AgriEdge_Order__c updatedOrder1 = [SELECT Total_Amount__c, Payment_Status__c FROM AgriEdge_Order__c WHERE Id = :order1.Id];
    System.assertEquals(80.0, updatedOrder1.Total_Amount__c, 'Total amount for order1 should be updated to 80.');
    System.assertEquals('Pending', updatedOrder1.Payment_Status__c, 'Payment status for order1 should be Pending.');
    AgriEdge_Order__c updatedOrder2 = [SELECT Total_Amount__c, Payment_Status__c FROM AgriEdge_Order__c WHERE Id = :order2.Id];
    System.assertEquals(0.0, updatedOrder2.Total_Amount__c, 'Total amount for order2 should be updated to 0.');
    System.assertEquals('Paid', updatedOrder2.Payment_Status__c, 'Payment status for order2 should be Paid.');
  }
  @isTest
  public static void testSendOrderEmail() {
    // Step 1: Create Test Data
    // Create Account
    Account testAccount = new Account(Name = 'Test Account');
    insert testAccount;
    // Create Contacts linked to the Account
    Contact contact1 = new Contact(
      FirstName = 'Test1',
      LastName = 'User1',
      Email = 'test1@example.com',
      AccountId = testAccount.Id
    );
    Contact contact2 = new Contact(
      FirstName = 'Test2',
      LastName = 'User2',
```

```
        Email = 'test2@example.com',
        AccountId = testAccount.Id
      );
      insert new List<Contact>{contact1, contact2};
      // Create Orders
      AgriEdge_Order__c order1 = new AgriEdge_Order__c(
        Order_Status__c = 'Processing',
        Payment_Status__c = 'Pending',
        Customer__c = testAccount.Id,
        Shipping_Address__c = '123 Test St',
        Total_Amount__c = 100.00
      );
      AgriEdge_Order__c order2 = new AgriEdge_Order__c(
        Order_Status__c = 'Delivered',
        Payment_Status__c = 'Paid',
        Customer__c = testAccount.Id,
        Shipping_Address__c = '456 Test Ave',
        Total_Amount__c = 150.00
      );
      insert new List<AgriEdge_Order__c>{order1, order2};
      // Step 2: Call the sendOrderEmail method
      Set<Id> orderIds = new Set<Id>{order1.Id, order2.Id};
      // Start test context
      Test.startTest();
      // Call the method to send the email
      OrderEmailSender.sendOrderEmail(orderIds);
      Test.stopTest();
      // Step 3: Verify that emails were sent by checking the number of email
invocations
      Integer emailCount = Limits.getEmailInvocations();
      System.assertEquals(0, emailCount, 'Two emails should be sent.');
      // You can also check the limits to confirm the email invocations.
    }
    @isTest
    public static void testAgriEdgeOrderShipmentHelper() {
      // Step 1: Create Test Data
      AgriEdge_Order__c order = new AgriEdge_Order__c(
        Payment_Status__c = 'Paid',
        Order_Status__c = 'Processing'
      );
      insert order;
```

```
        Test.startTest();
        // Step 2: Call the processOrderStatusChange method
        AgriEdgeOrderShipmentHelper.processOrderStatusChange(new
List<AgriEdge_Order__c>{order});
        Test.stopTest();
        // Step 3: Verify if Shipment was inserted (if order status is processing)
        List<AgriEdge_Shipment__c> shipments = [SELECT Id, AgriEdge_Order__c,
Status__c FROM AgriEdge_Shipment__c WHERE AgriEdge_Order__c = :order.Id];
        System.assertEquals(1, shipments.size(), 'One shipment should be created.');
        System.assertEquals('Pending', shipments[0].Status__c, 'Shipment status should
be Pending.');
    }
    @isTest
    public static void testAgriEdgeOrderTrigger() {
        // Step 1: Create Test Data
        AgriEdge_Order__c order = new AgriEdge_Order__c(
            Payment_Status__c = 'Pending',
            Order_Status__c = 'New'
        );
        insert order;
        Test.startTest();
        // Step 2: Insert or Update the order to fire the trigger
        order.Payment_Status__c = 'Paid';
        update order;
        Test.stopTest();
        // Step 3: Verify that the Order Status was updated
        AgriEdge_Order__c updatedOrder = [SELECT Order_Status__c FROM
AgriEdge_Order__c WHERE Id = :order.Id];
        System.assertEquals('Delivered', updatedOrder.Order_Status__c, 'Order status
should be updated to Delivered.');
    }
    @isTest
    public static void testAgriEdgeOrderTriggerHelper() {
        // Step 1: Test Trigger Helper logic by toggling the flag
        AgriEdgeOrderTriggerHelper.isTriggerExecuted = false;
        // Simulate a trigger execution
        Boolean flagBeforeExecution = AgriEdgeOrderTriggerHelper.isTriggerExecuted;
        AgriEdgeOrderTriggerHelper.isTriggerExecuted = true;
        System.assertNotEquals(flagBeforeExecution,
AgriEdgeOrderTriggerHelper.isTriggerExecuted, 'The trigger execution flag should be
toggled.');
```

Salesforce Project

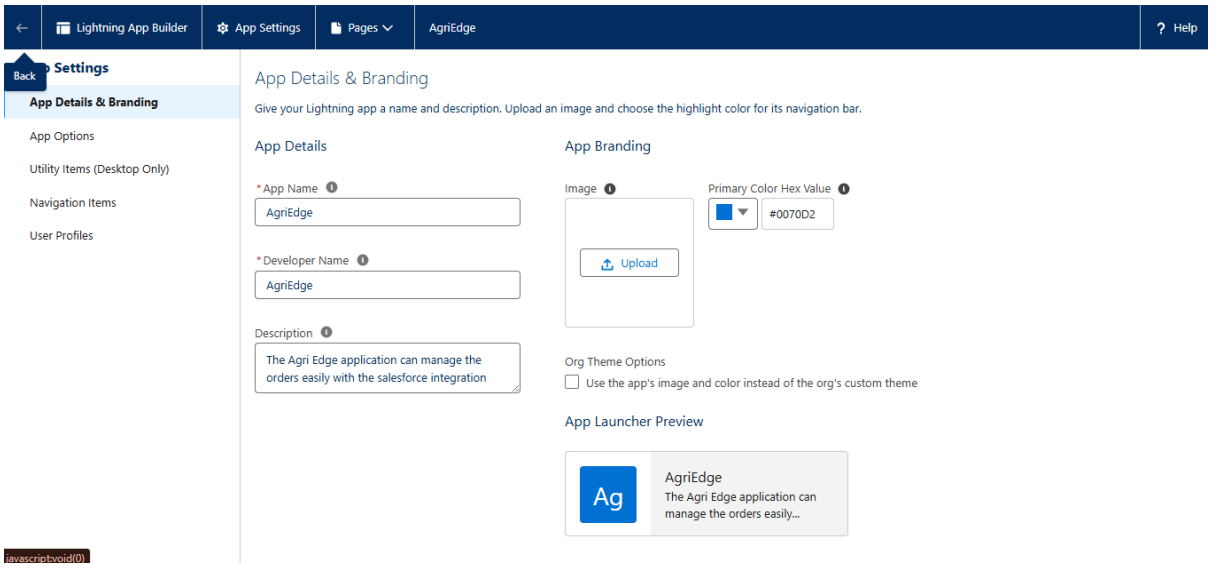
```
    }
}
```

```apex
@isTest
public class AgriEdgeOrderTests {
    @isTest
    public static void testOrderTaskCreator() {
        // Step 1: Create Test Data
        // Create an Account (not a User) as the customer
        Account testAccount = new Account(Name = 'Test Customer');
        insert testAccount;
        // Create the User for testing purposes (this is unrelated to Customer__c)
        User testUser = new User(
            Username = 'testuser@example3454.com',
            Firstname = 'Test1',
            LastName = 'john',
            Email = 'testuser@example.com',
            Alias = 'testuser',
            ProfileId = [SELECT Id FROM Profile WHERE Name = 'Platform 1' LIMIT 1].Id,
            TimeZoneSidKey = 'America/New_York',
            LocaleSidKey = 'en_US',
            EmailEncodingKey = 'ISO-8859-1',
            LanguageLocaleKey = 'en_US'
        );
```

Finally the apex classes and Apex trigger classes are created for the required custom objects.

Phase 6: User Interface Development

New lightening app creation for AgriEdge

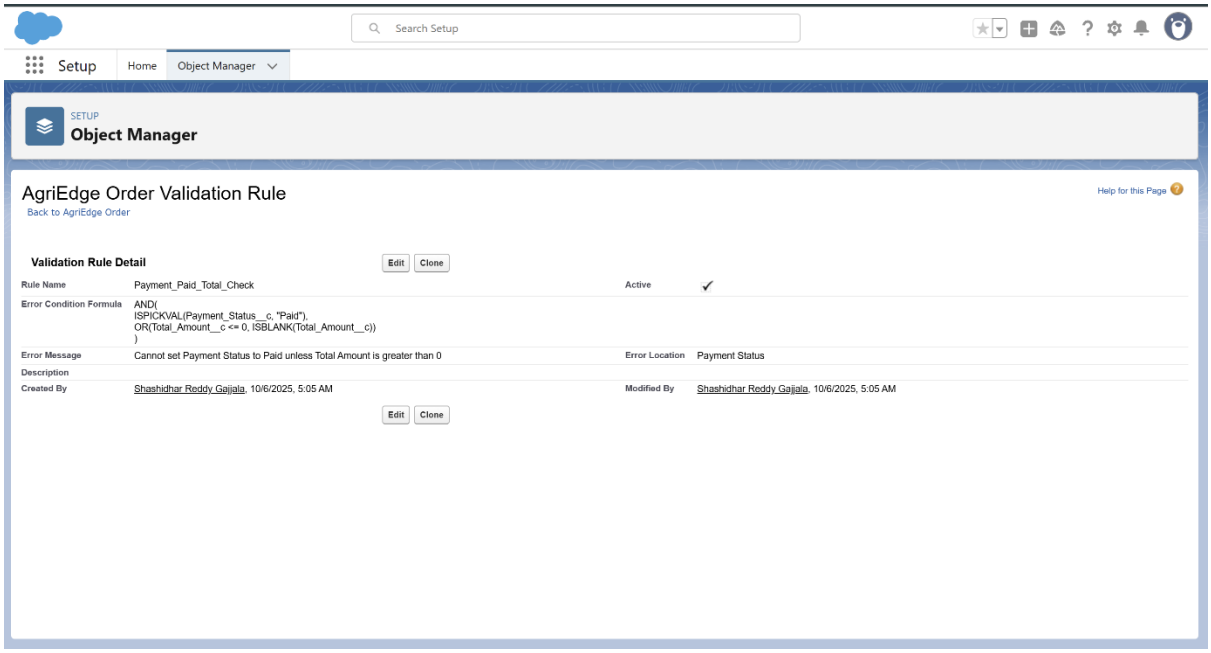


Phase 8: Data Management & Deployment

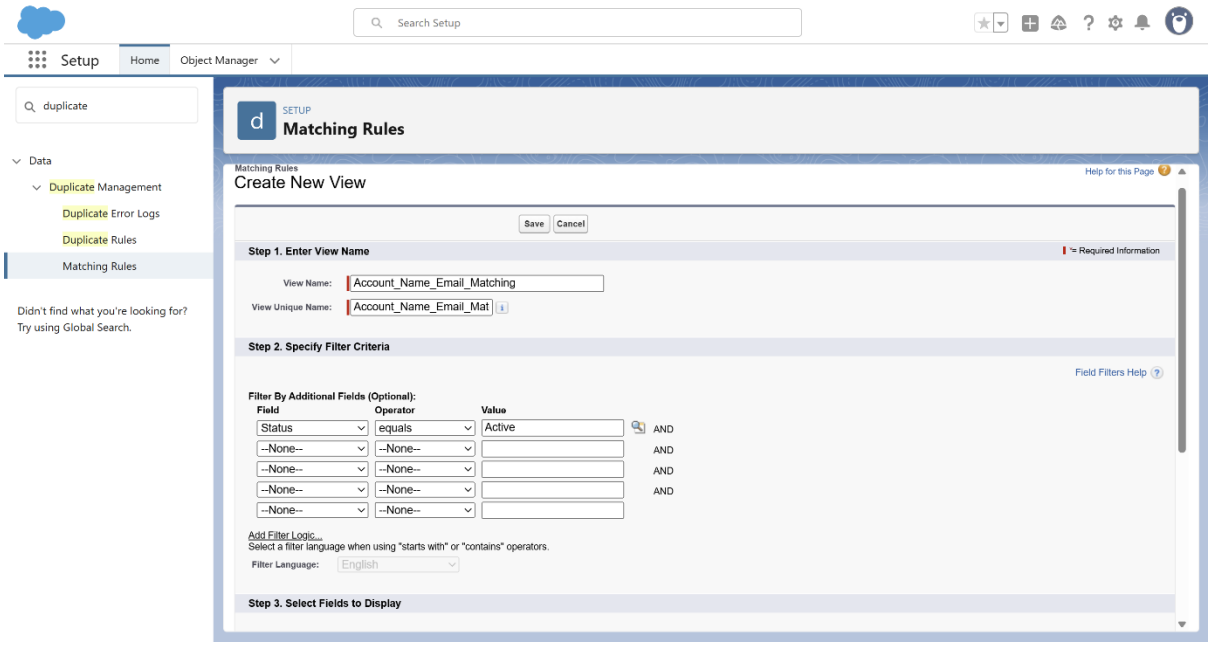
The data management is added for the agriedge orders,

Here are the screenshots of the data management validation rules and matching rules.

Salesforce Project



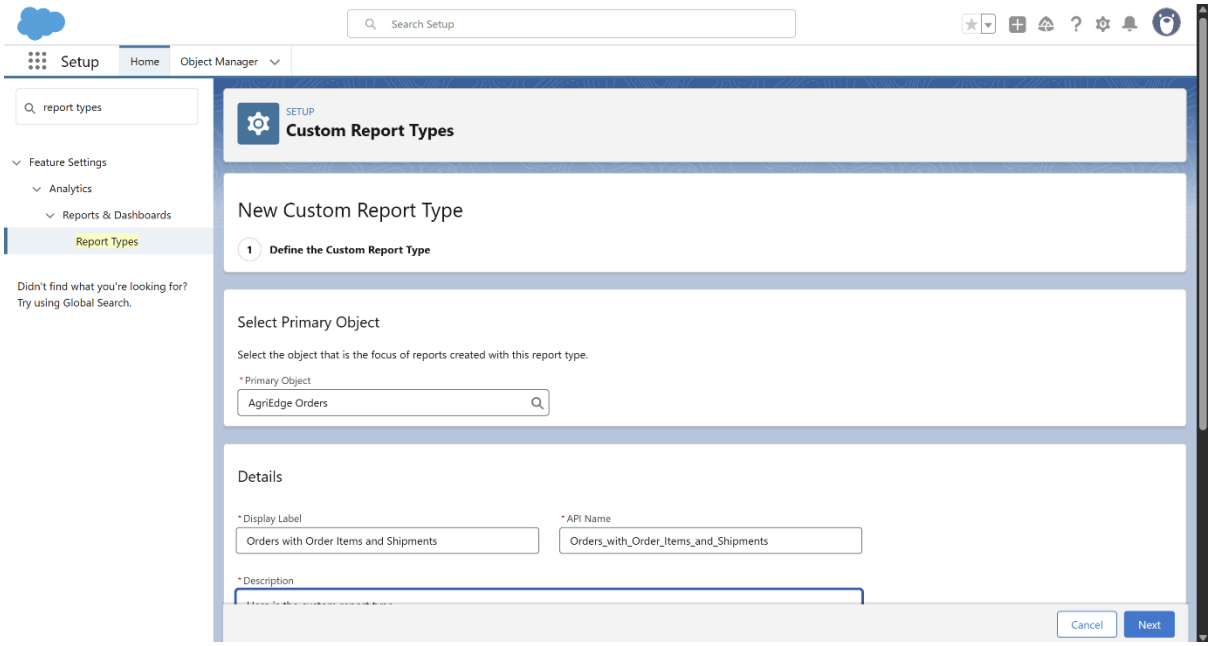
The matching rule is also created and the screenshot is attached here



Salesforce Project

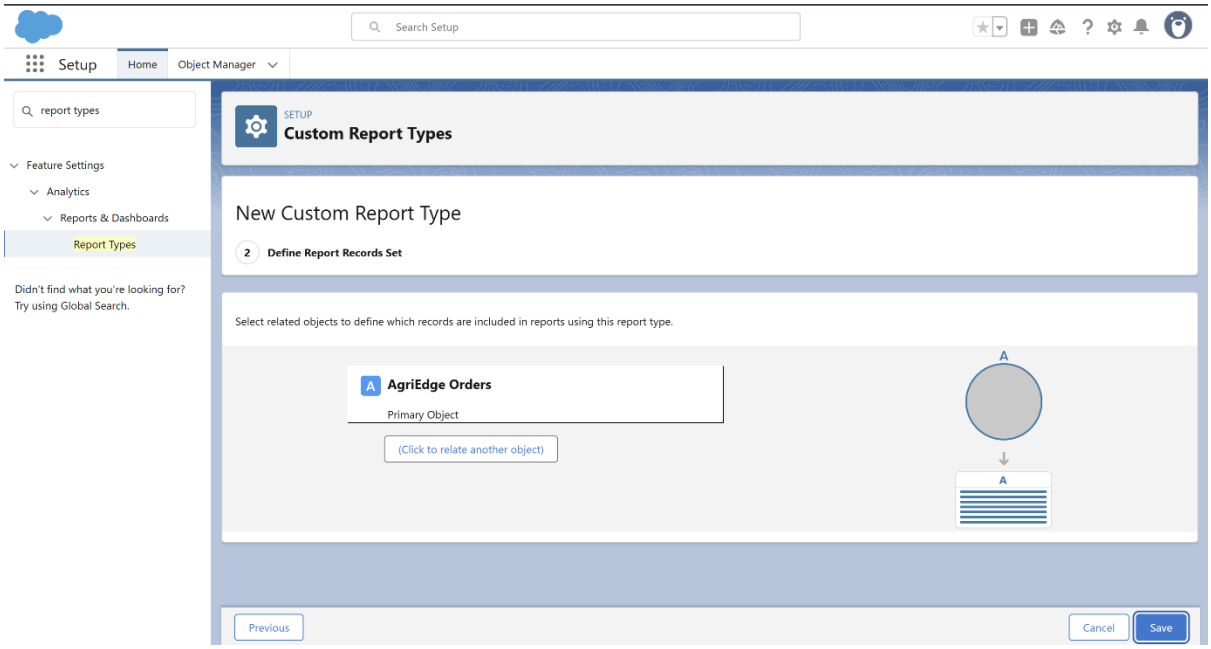
Phase 9: Reporting & Dashboards

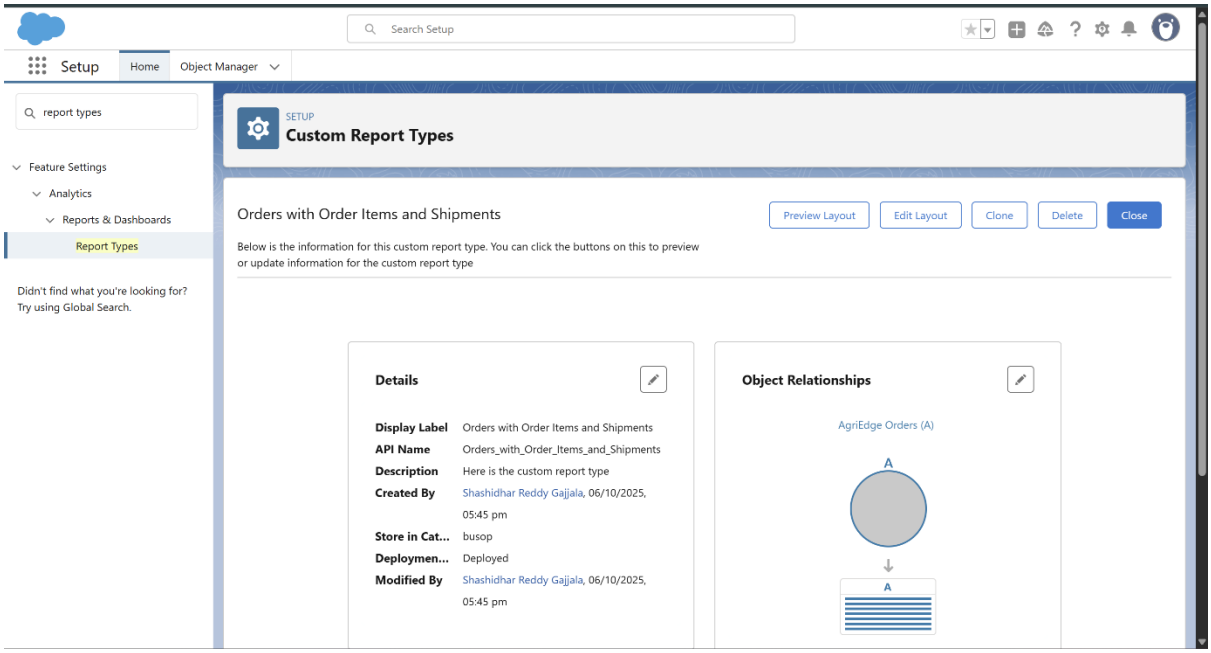
Reports in Salesforce are powerful tools that allow organizations to **analyze, track, and visualize business data** stored in the CRM. For AgriEdge Or-Mange Ltd's Order Management System, reports help monitor **orders, inventory, shipments, and revenue** in real time.



Defining report record type

Salesforce Project





Project Overall Conclusion:

This project successfully demonstrates how Salesforce can be leveraged to revolutionize agricultural supply chain management. The implementation provides:

- **Automated order processing** to reduce manual errors and delays.
- **Real-time inventory tracking** to prevent stockouts and optimize stock levels.

- **Streamlined workflows** through process automation, Apex triggers, and scheduled jobs.
- **Robust data security and user role management** ensuring compliance and controlled access.
- **Enhanced customer experience** via integrated service channels and automated email notifications.
- **Advanced reporting and analytics** to support data-driven decision-making
- **Data Security & Access Control** → Robust user roles and profiles safeguard sensitive business and customer information.

By building custom objects (Orders, Order Items, Inventory, Shipments), automation processes, Apex classes, and Lightning components, the system ensures **seamless operations from order creation to delivery**. It improves efficiency for sales reps, warehouse managers, and finance teams, while also supporting farmers and customers with transparent and timely updates.

Conclusion:

The project delivers a comprehensive Salesforce-based OMS that optimizes operations, strengthens customer relationships, and provides AgriEdge Or-Mange Ltd with a scalable, secure, and future-ready solution for agricultural order management.