Handsmen Threads

Project Overview:

HandsMen Threads, a premium men's fashion brand, is embracing Salesforce to take its operations to the next level. The goal is clear: streamline day-to-day processes, deliver an exceptional customer experience, and manage inventory more efficiently—all through the power of Salesforce.

At the heart of the project is the roll-out of **Salesforce CRM**, which will serve as the central hub for managing customer orders, tracking inventory in real-time, running targeted marketing campaigns, and delivering smooth, responsive customer support.

A big focus of the project is making sure the data stays clean and accurate, right from the moment someone enters it in the system. This helps the team make better decisions and ensures everything runs like clockwork.

As part of the project, several new features has been rolled out to boost customer satisfaction and streamline everyday operations:

- **Order Confirmation Emails**: Every time a customer places an order, they'll get a confirmation email right away. It's a simple touch that builds trust and keeps communication clear.
- **Smart Loyalty Program**: Customers will automatically see their loyalty status update based on their purchase history. This means more tailored rewards and a better overall experience that keeps them coming back.
- **Stock Level Alerts**: If any item in the inventory drops below five units, the warehouse team will get an alert. This proactive system helps avoid stock-outs and keeps things running smoothly.
- **Midnight Bulk Order Updates**: Every night at midnight, the system will handle bulk order updates—adjusting stock levels and updating financial records—so that everything is up to date for the next day's operations without interrupting end users.

This Salesforce initiative is a big step forward for HandsMen Threads, setting the stage for smarter workflows, happier customers, and stronger business growth. So that the business will grow and maintain customer relation automatically through email alerts without any hassle.

Objective:

Salesforce CRM Setup: Building a solid foundation to manage customer relationships, sales, and service—all in one place.

Smart Automation: Automating routine tasks and workflows to boost efficiency, reduce manual work, and let the team focus on what matters most.

Custom Development: Using Apex code and triggers to tailor Salesforce to the brand's unique business needs, ensuring every process works just right.

Scheduled Processing: Implementing batch jobs and scheduled processes to handle large volumes of data—like bulk order updates and inventory sync—without a hitch.

Data Security & Access Controls: Putting strong security in place to make sure sensitive information is protected and only the right people have access what they need to have access of and stripping out additional access.

Phase 1: Requirement Analysis & Planning

The first step in the Salesforce journey for HandsMen Threads focuses on building a strong foundation through thoughtful analysis and strategic planning. This phase ensures the project aligns with real business needs and delivers measurable value.

Understanding Business Requirements

i. Improve order management efficiency by centralizing customer orders in Salesforce.
ii. Ensure real-time visibility of inventory to avoid stock-outs.
iii. Automate routine tasks such as order confirmations, stock alerts, and loyalty updates.
iv. Personalize customer engagement through targeted marketing and a dynamic loyalty program.

Defining Project Scope and Objectives

In Scope:

i. Automation using Flows and Apex (where necessary).
ii. Scheduled processes for bulk updates (e.g., daily midnight order processing).
iii. Email alert automation for customer orders and low stock notifications.
iv. Loyalty program logic tied to purchase behavior of the customer.
v. Role-based access to data for sales, warehouse, and customer service teams.

Objectives:

i. Enhance operational efficiency by reducing manual workload.
ii. Deliver a more personalized, responsive customer experience.
iii. Ensure data integrity and security through structured access controls.

Designing the Data Model and Security Model

Data Model Design:

i. Custom objects for *Handsmen Customer, Handsmen Orders, Handsmen Product, Inventory, and Marketing Champaign.*
ii. Relationships between Handsmen *Customer and Marketing Campaign*, *HandsMen Product and HandsMen Order, HandsMen Order and HandsMen Customer*, and *Inventory and HandsSome Product*
iii. Support for tracking order history, fulfillment status, and loyalty progress.

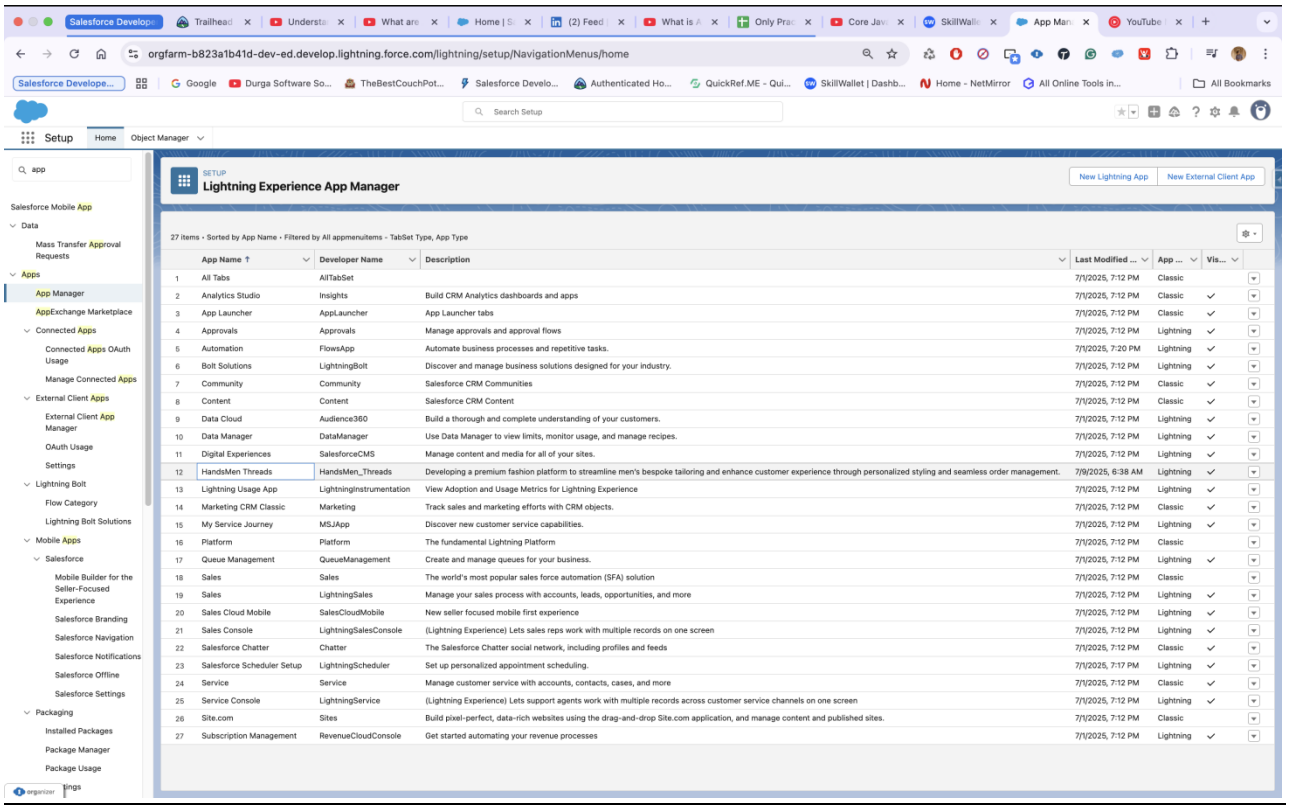
Security Model Design:

i. Role Hierarchy to reflect departments (e.g., Sales, Inventory, Marketing)
ii. Profile and Permission Sets to control access to sensitive data.

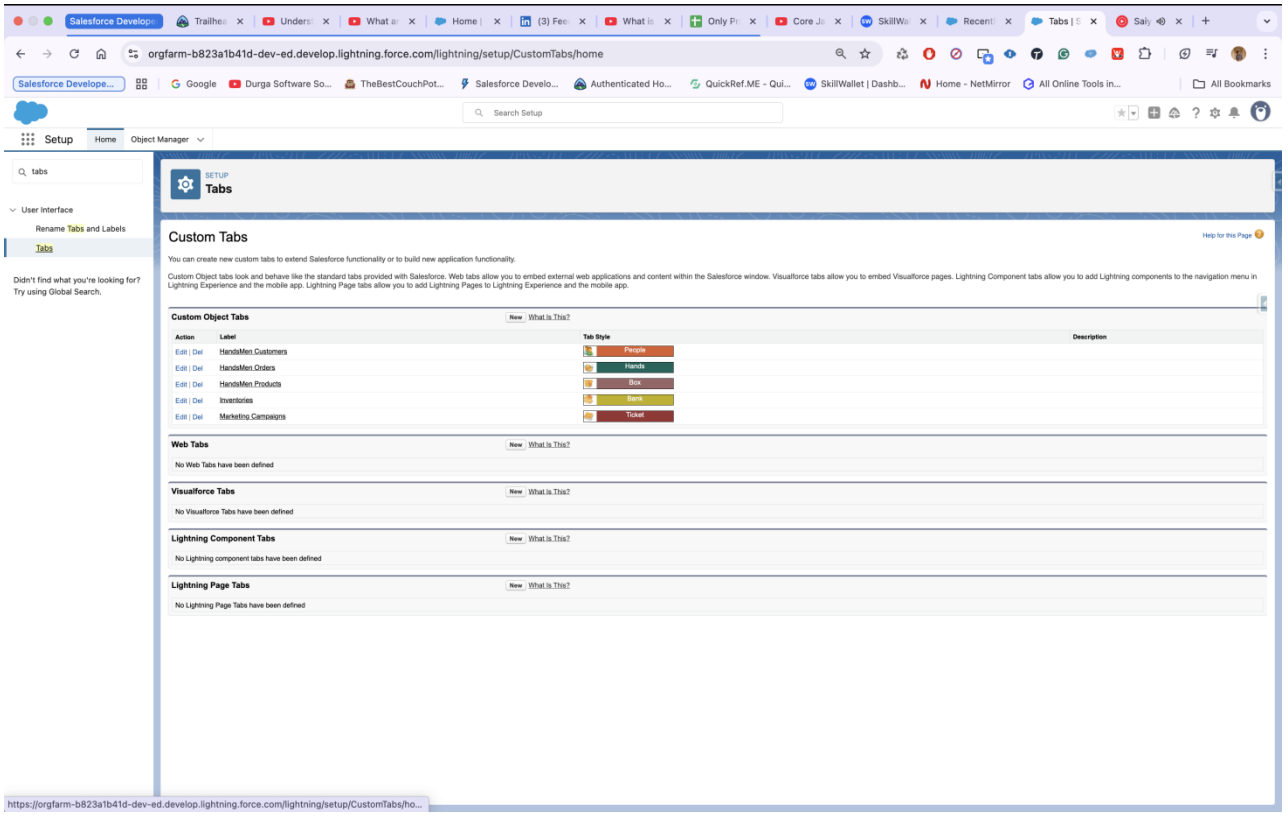
Phase 2: Salesforce Development - Backend & Configurations

Setup environment & DevOps workflow

i. Created Developer Account and build and application named - Handsmen Threads



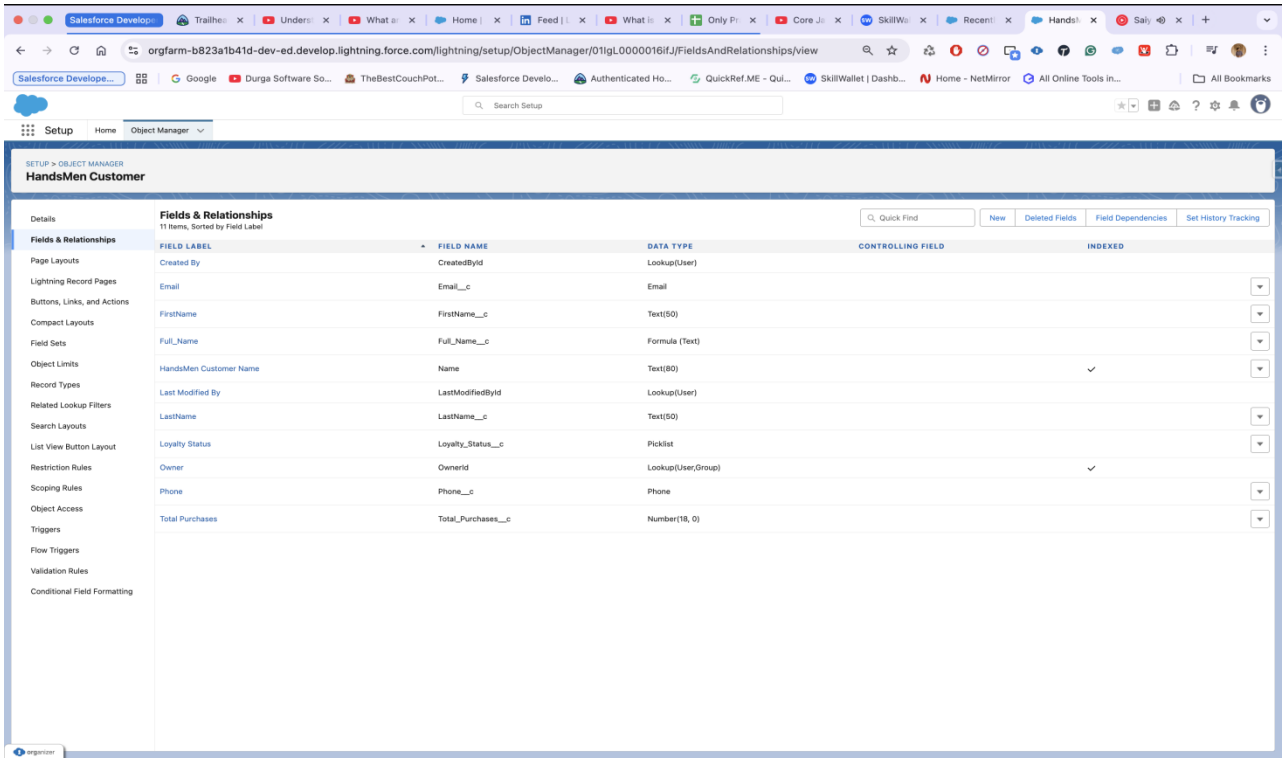
ii. Tabs for - Handsmen Threads



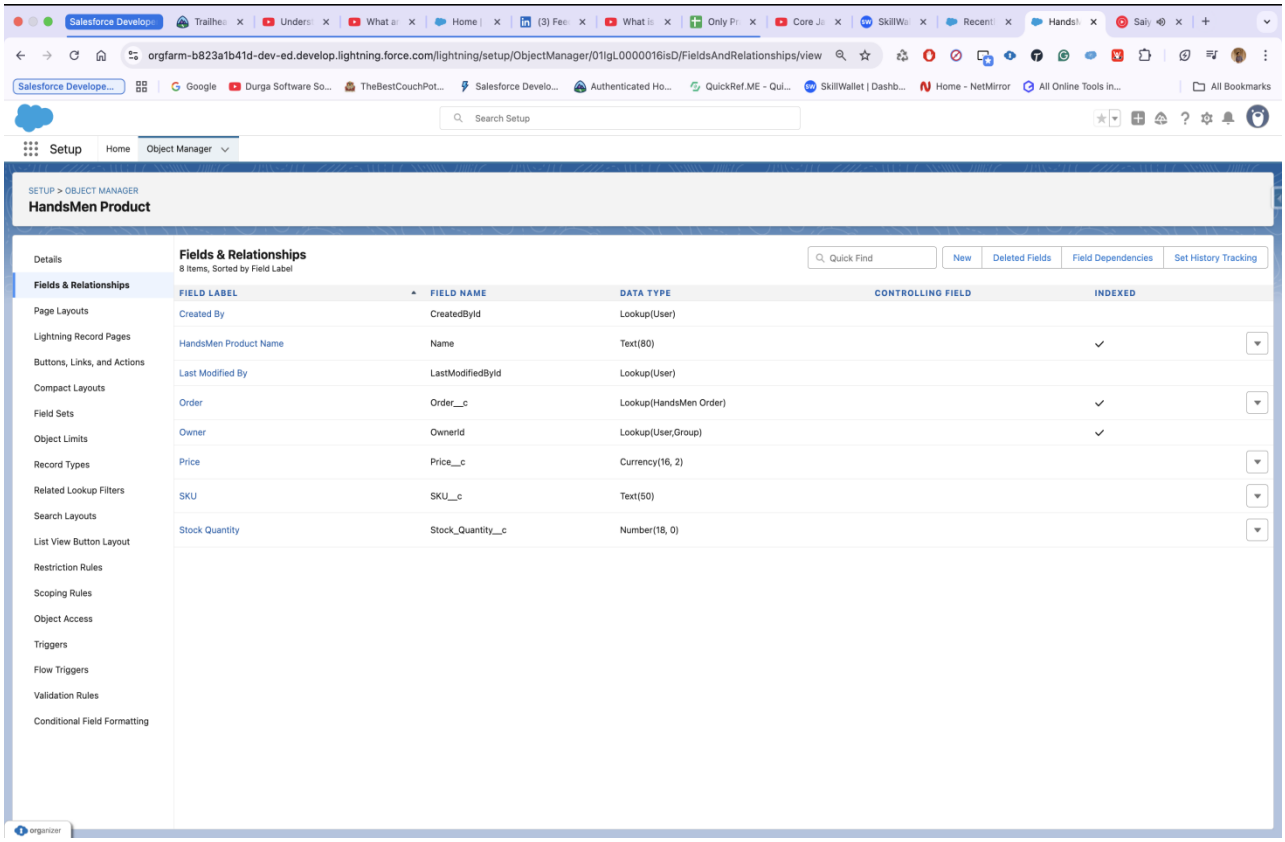
Customization of Objects, Fields, Validation Rules, Automation (Workflow Rules, Process Builder, Flows, Approval Process)

- Custom Objects and Fields creation

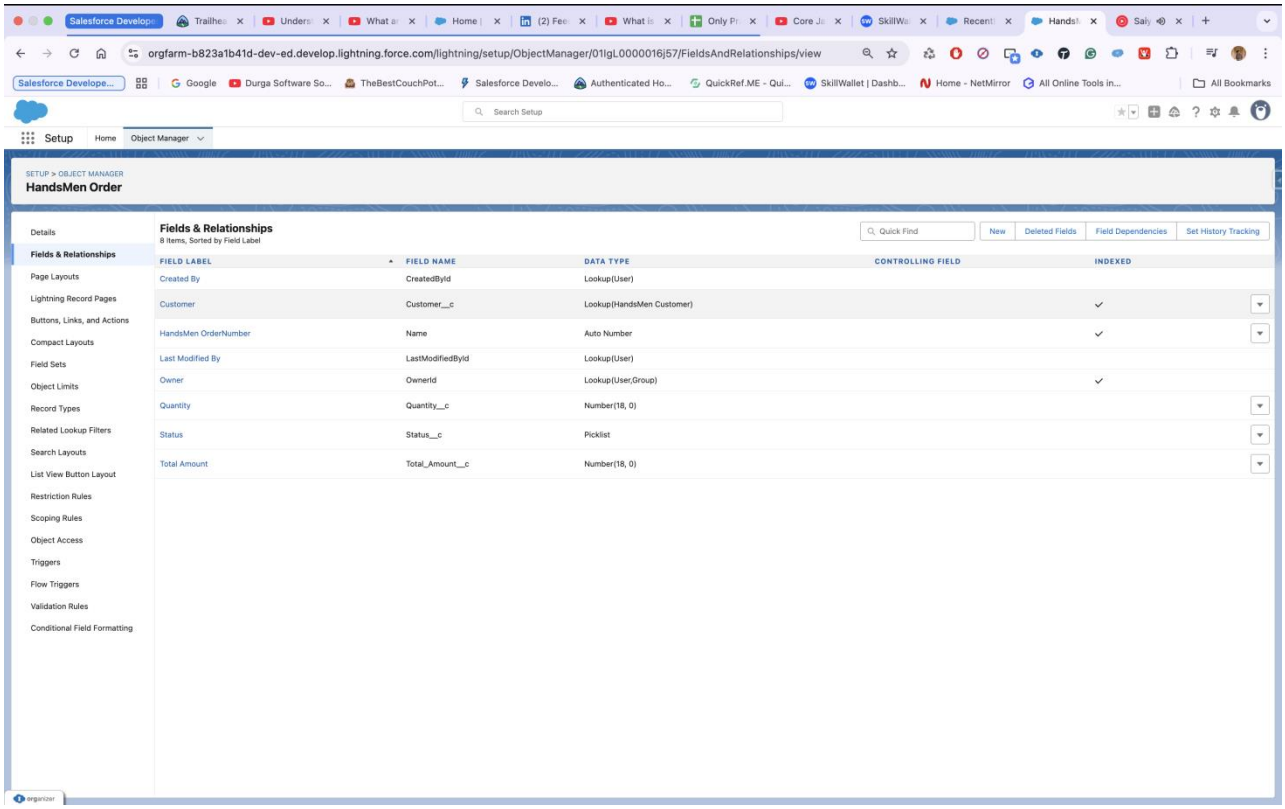
1. Custom Object - Handmen Customer



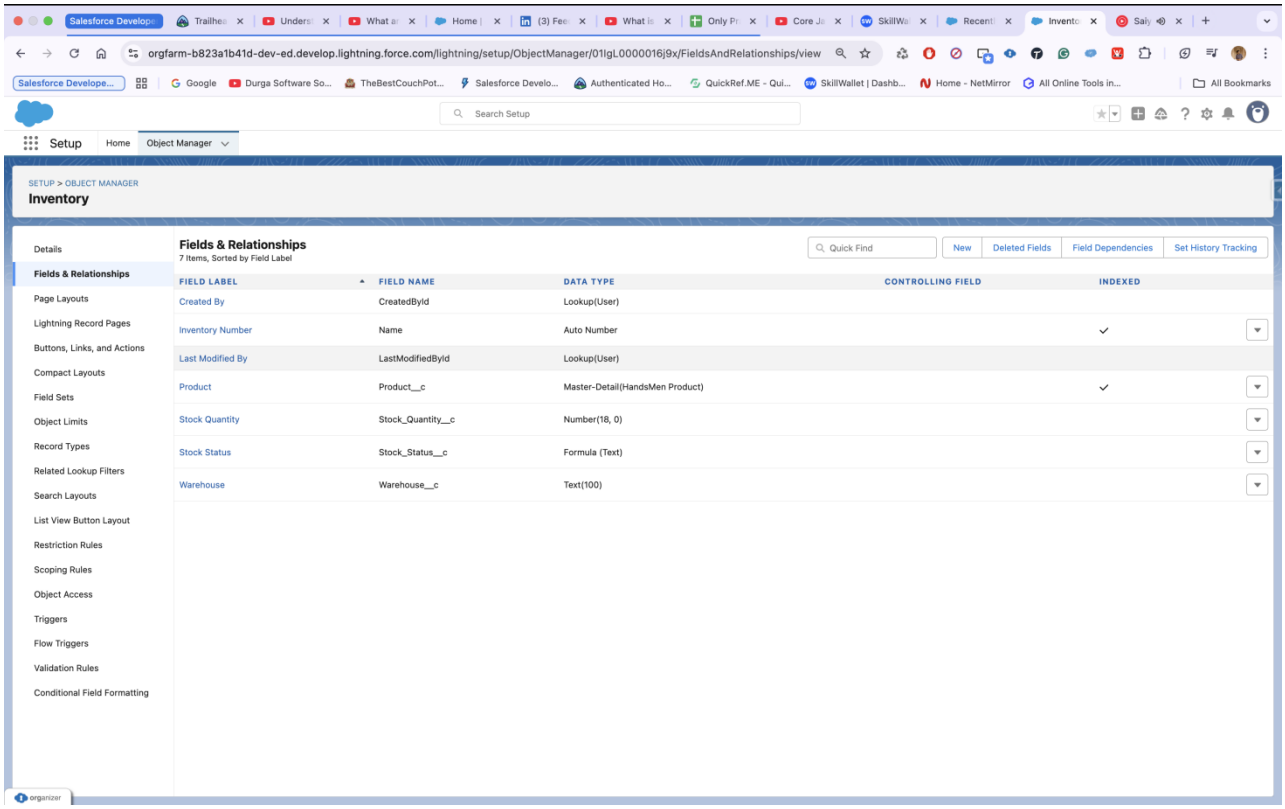
2. Custom Object - Handsmen Product



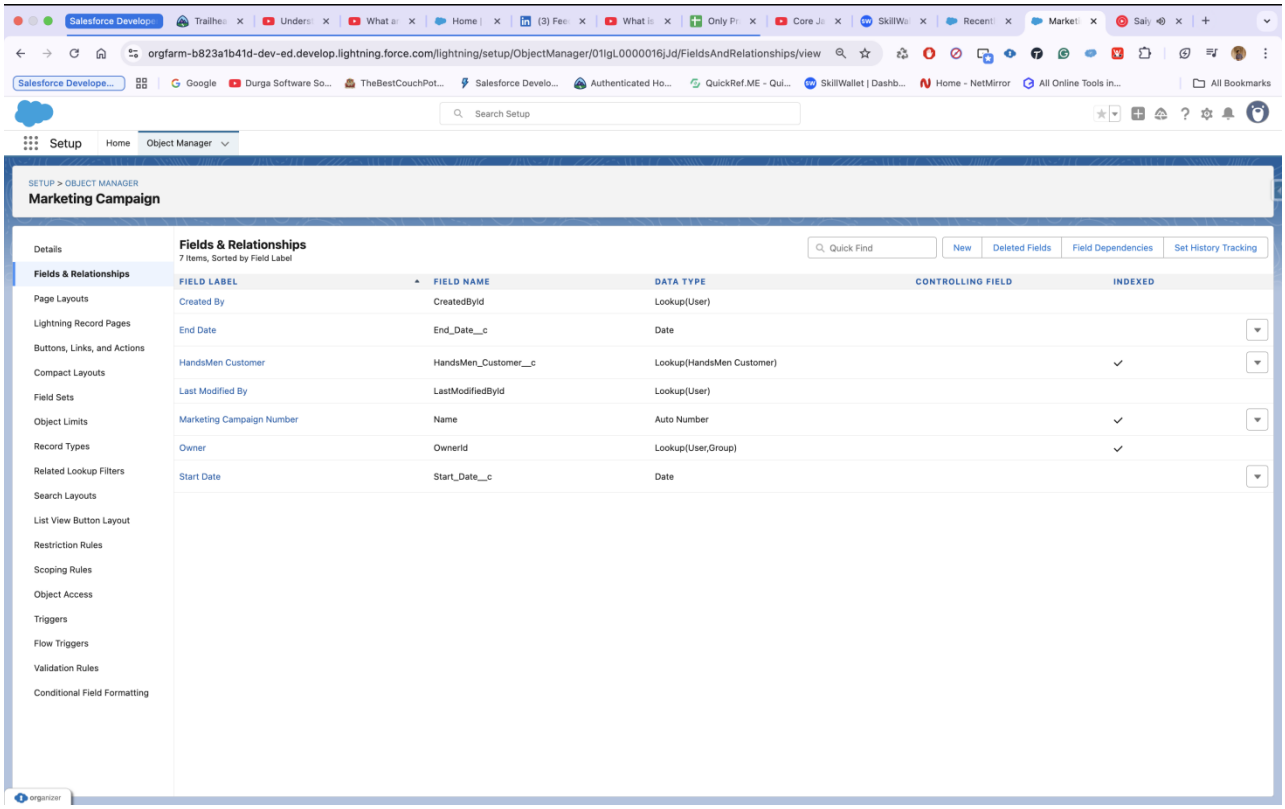
3. Custom Object - Handmen Order



4. Custom Object - Inventory

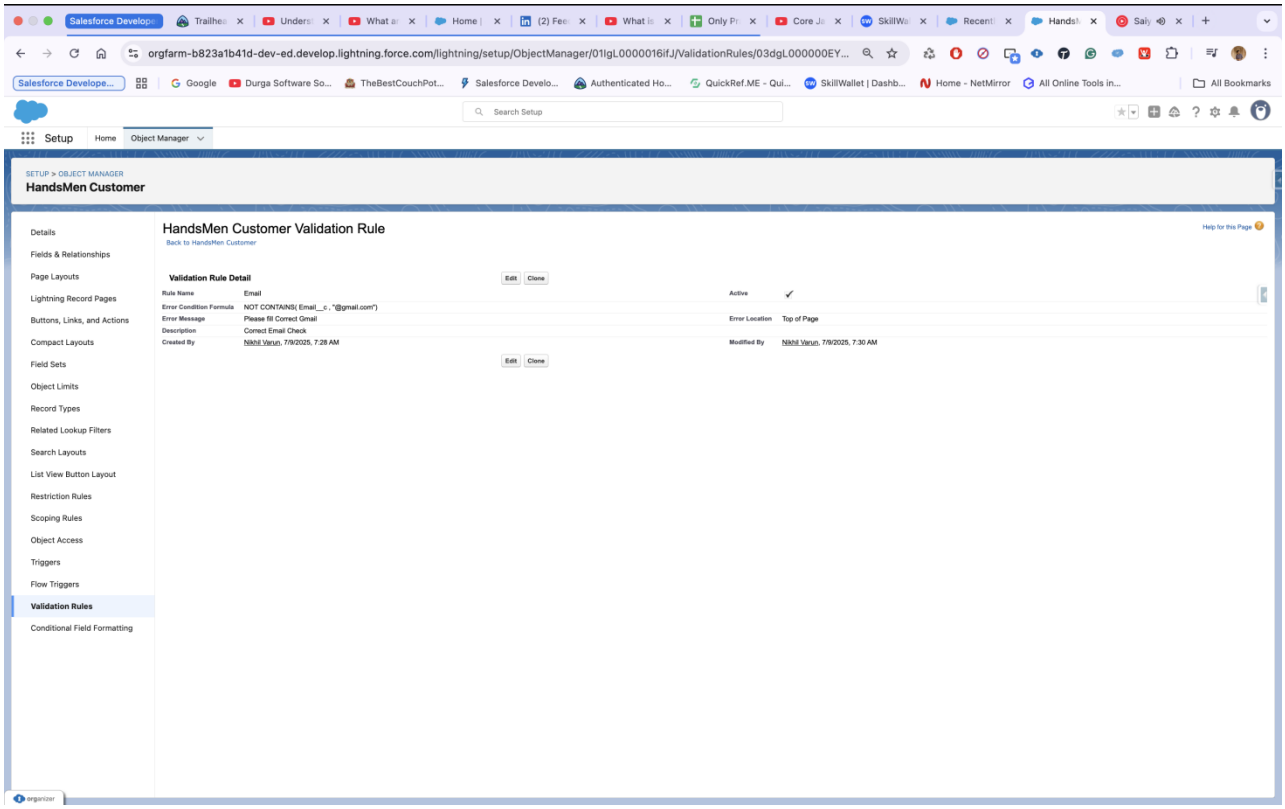


5. Custom Object - Marketing Campaign

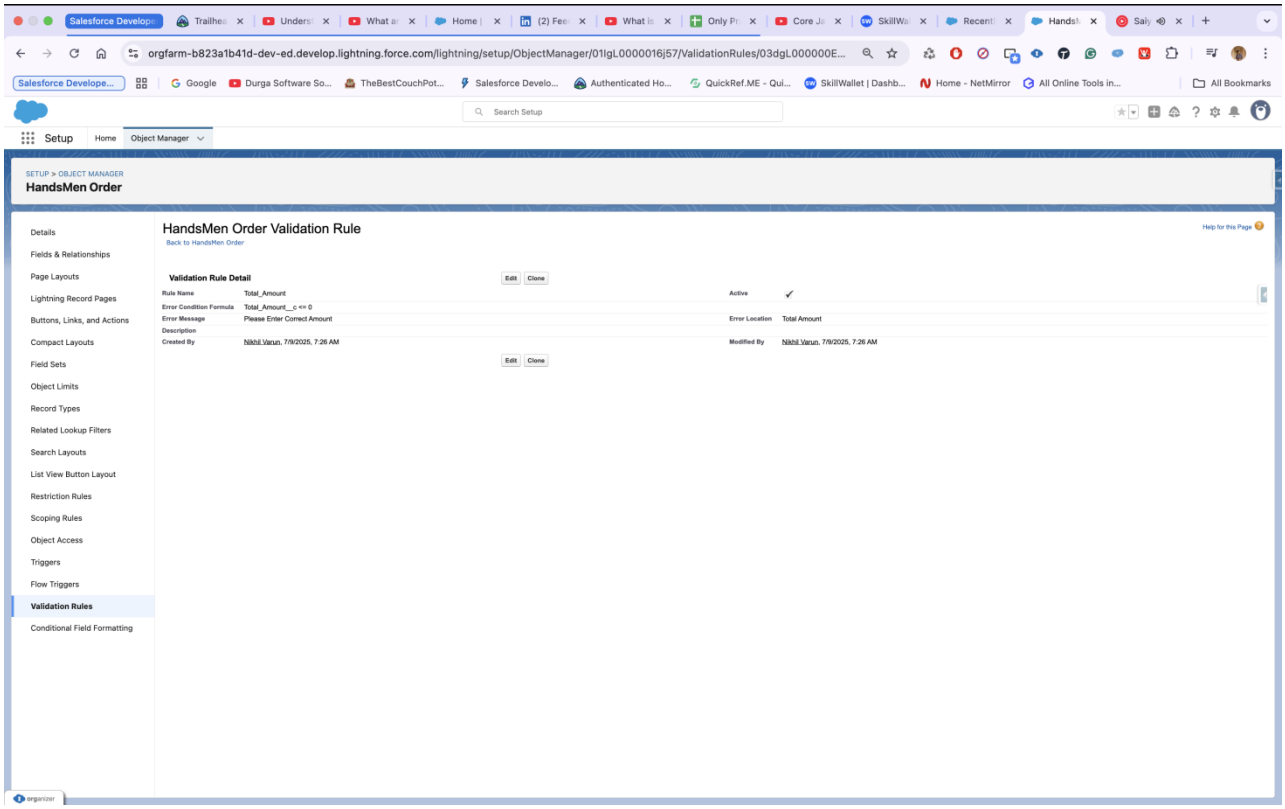


- Validation Rules Creation on 3 Objects and per business requirement

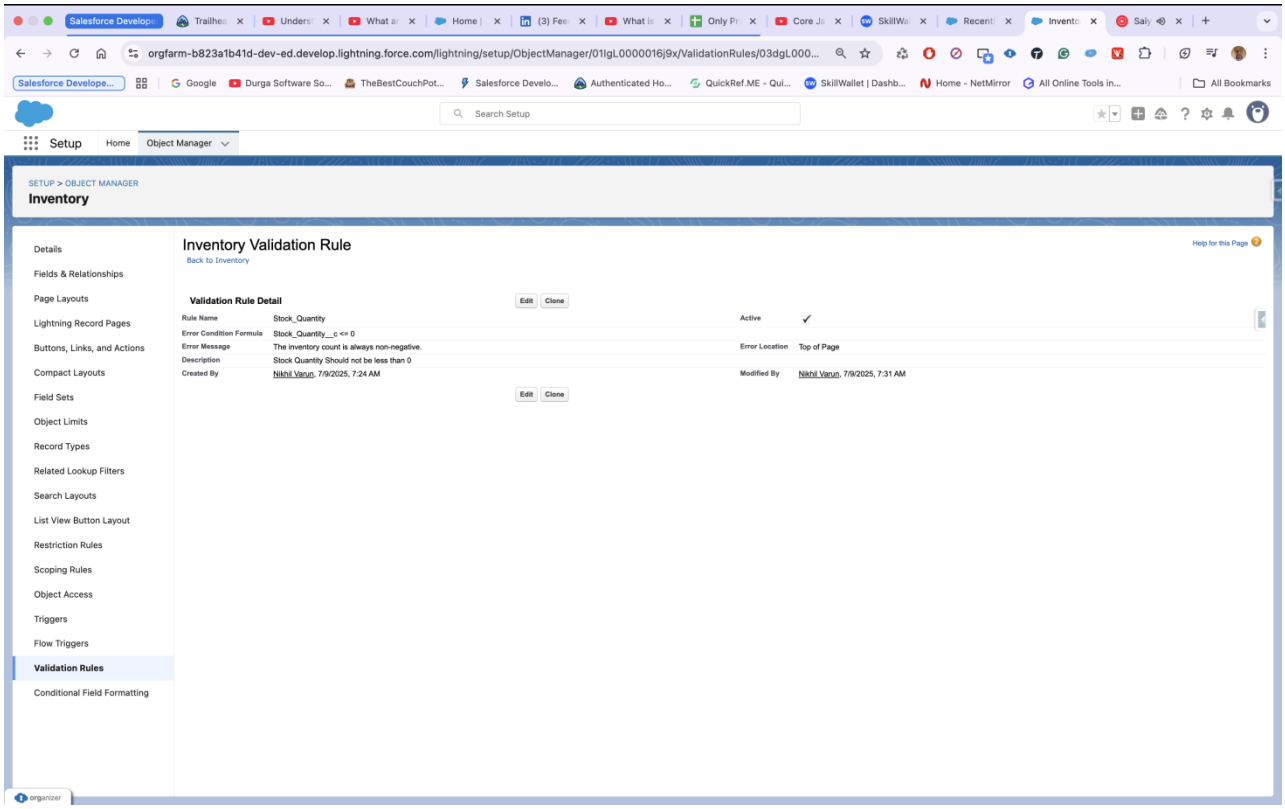
1. Validation Rule - Handsmen Customer



2. Validation Rule - Handsmen Order

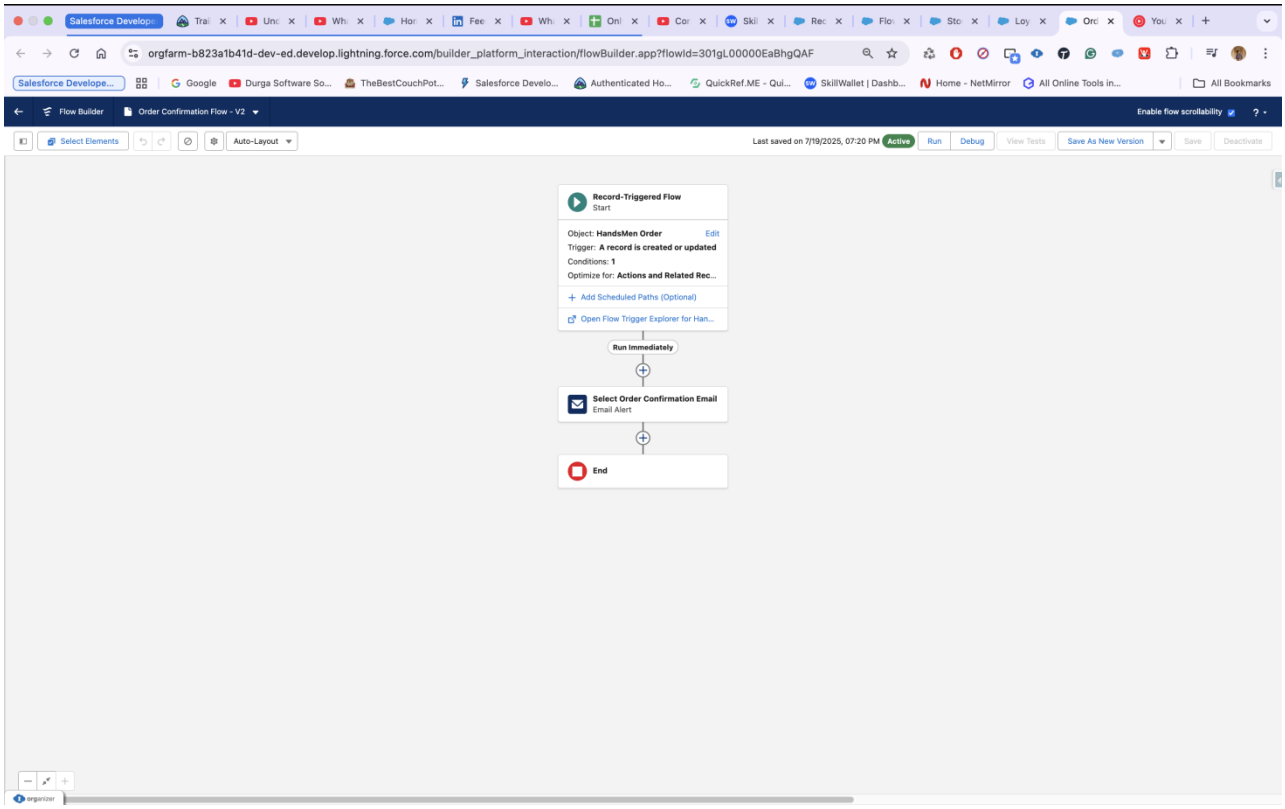


3. Validation Rule - Inventory

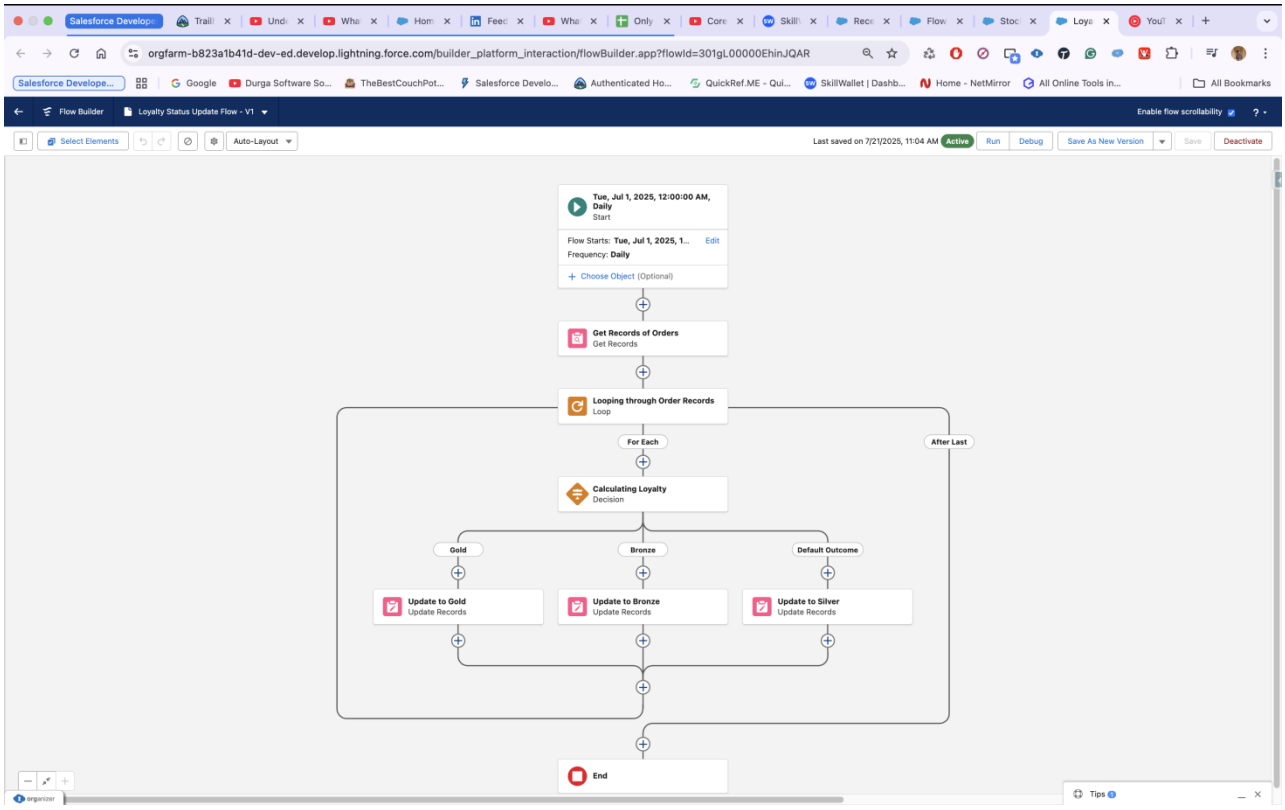


- Flows Creation

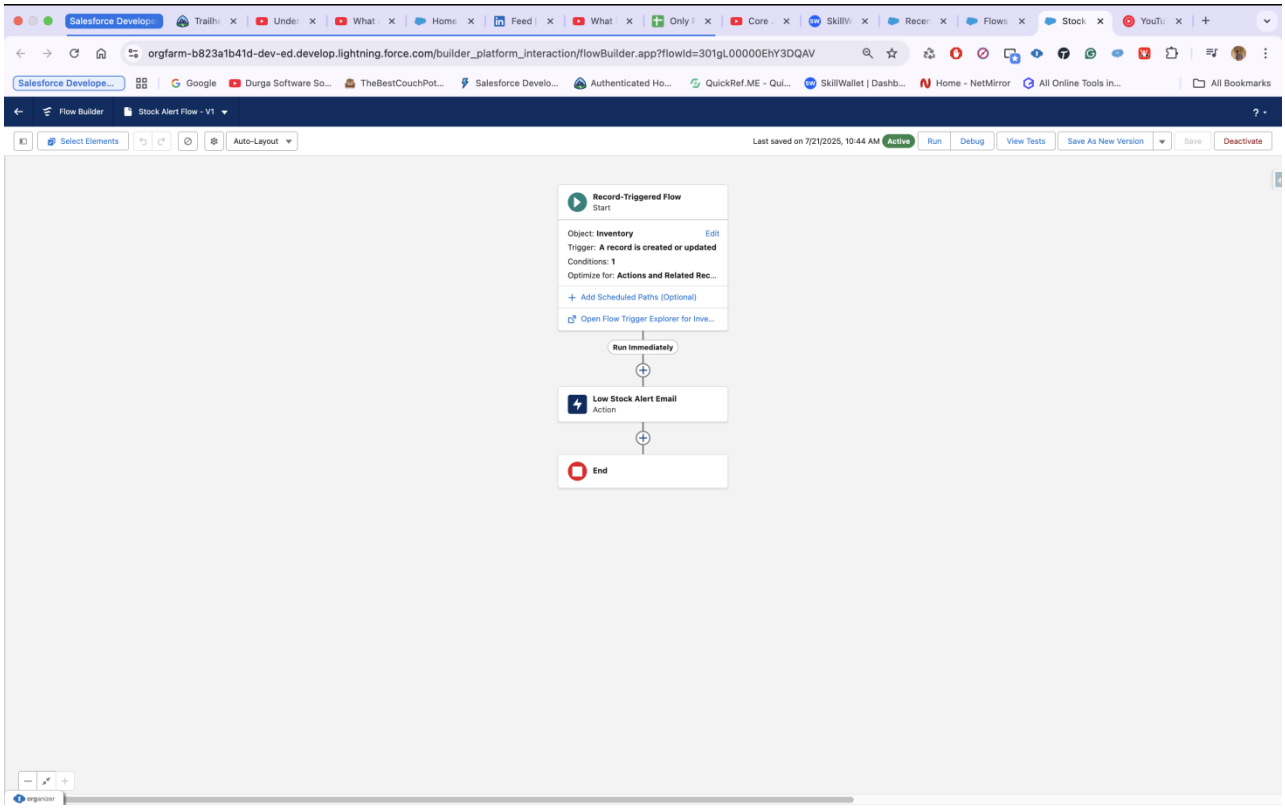
1. Flow - Record Triggered Flow - Email Alert



2. Flow - Scheduled Flow - Loyalty Status Update

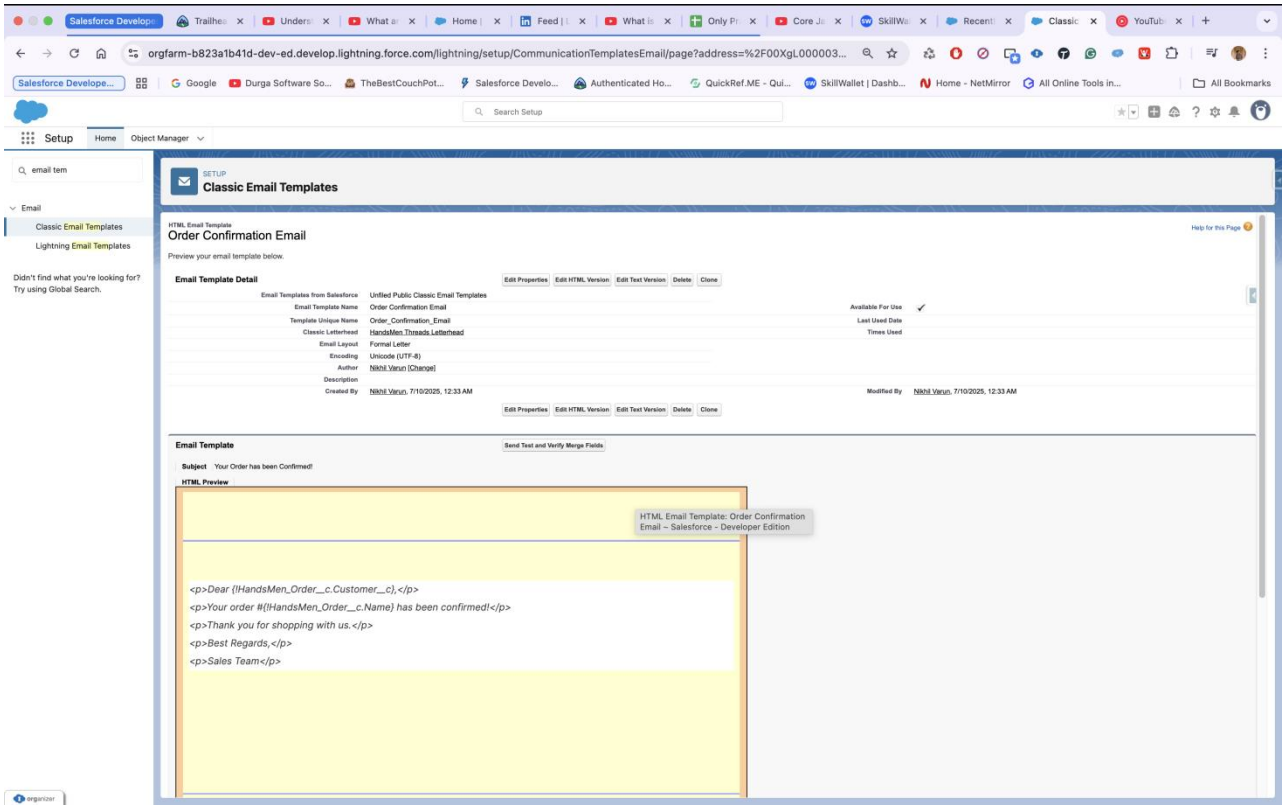


3. Flow - Record Triggered Flow - Low Stock Alert

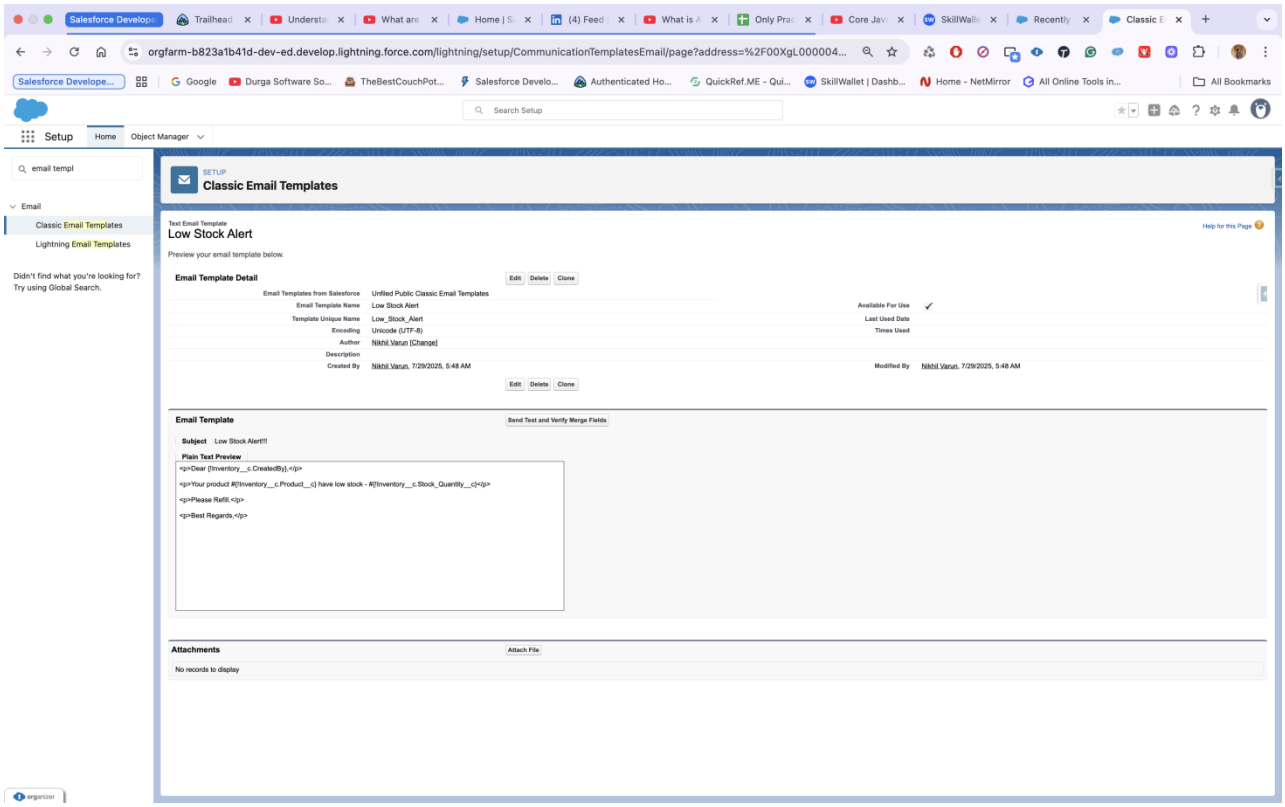


- Email Templates

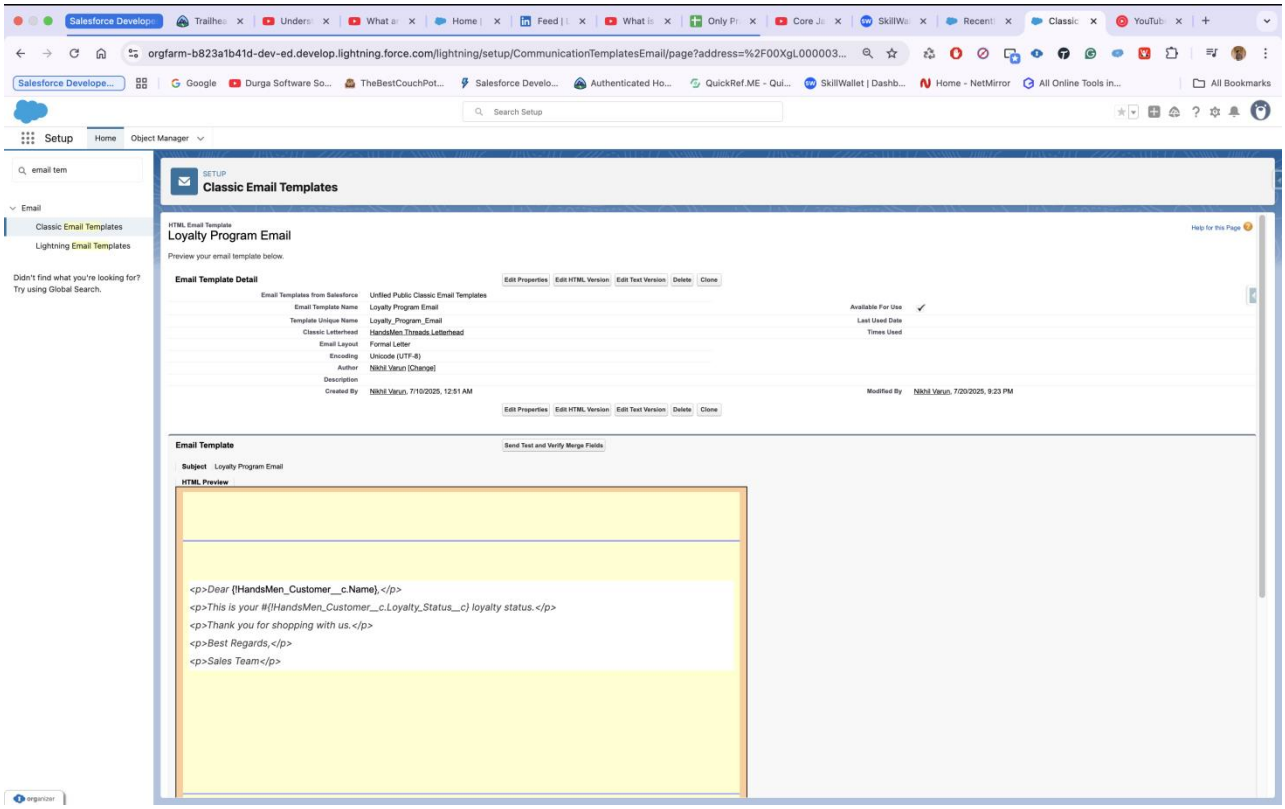
1. Order Confirmation



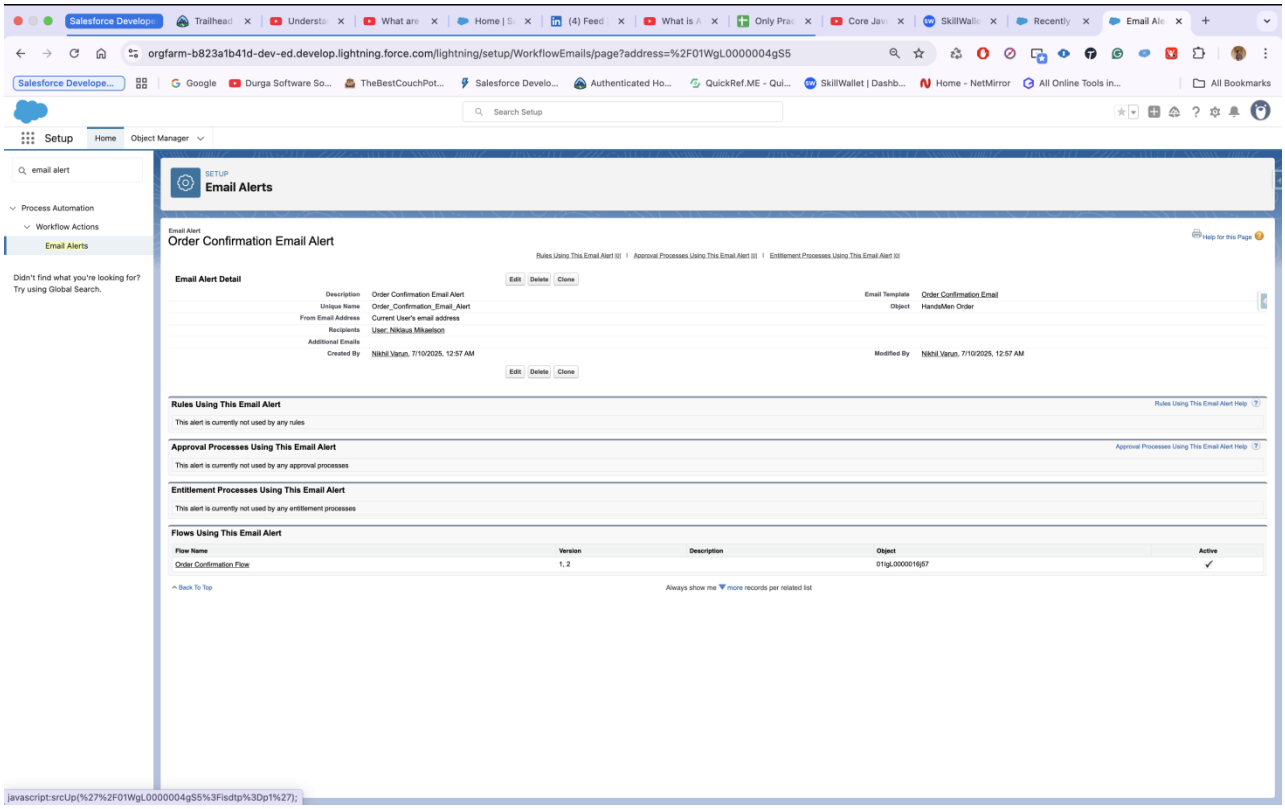
2. Low Stock Alert



3. Loyalty Program

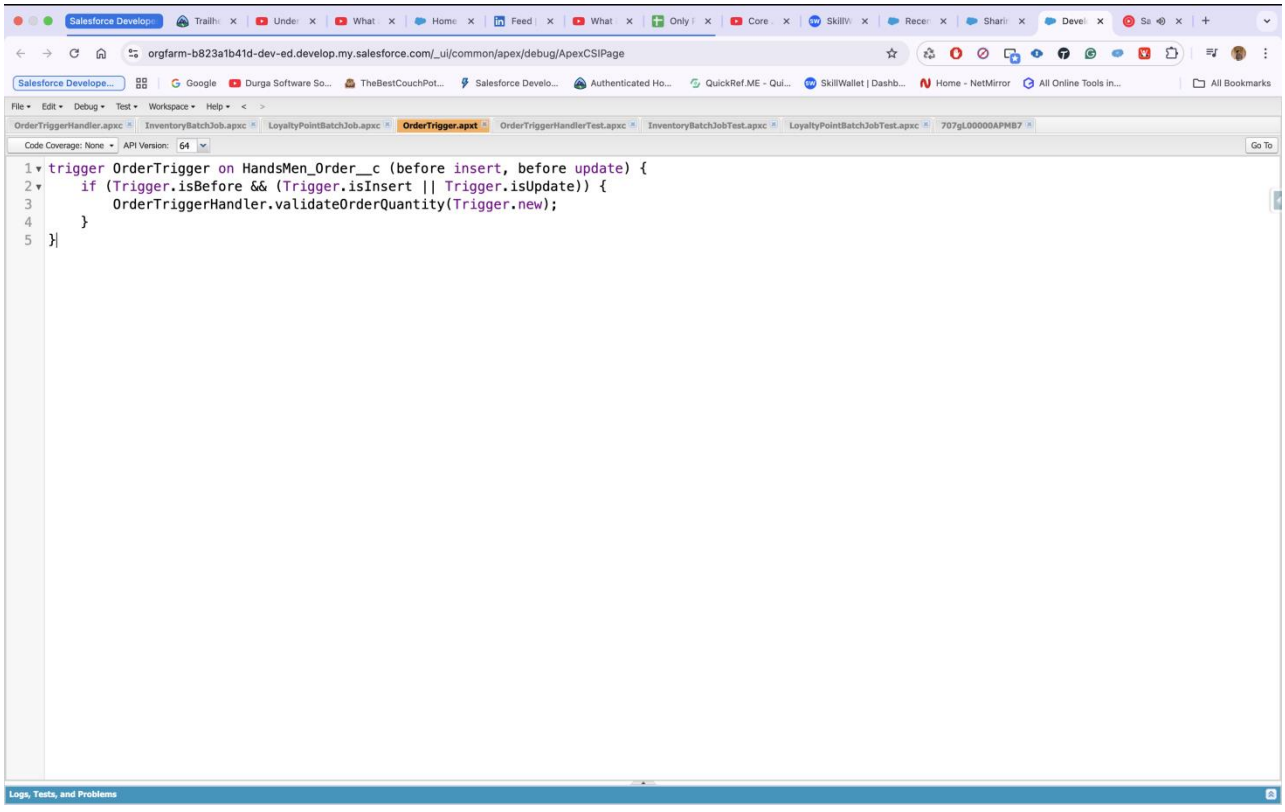


- Email Alert - Order Confirmation - Assigned to its recipient - Niklaus Mikaelson

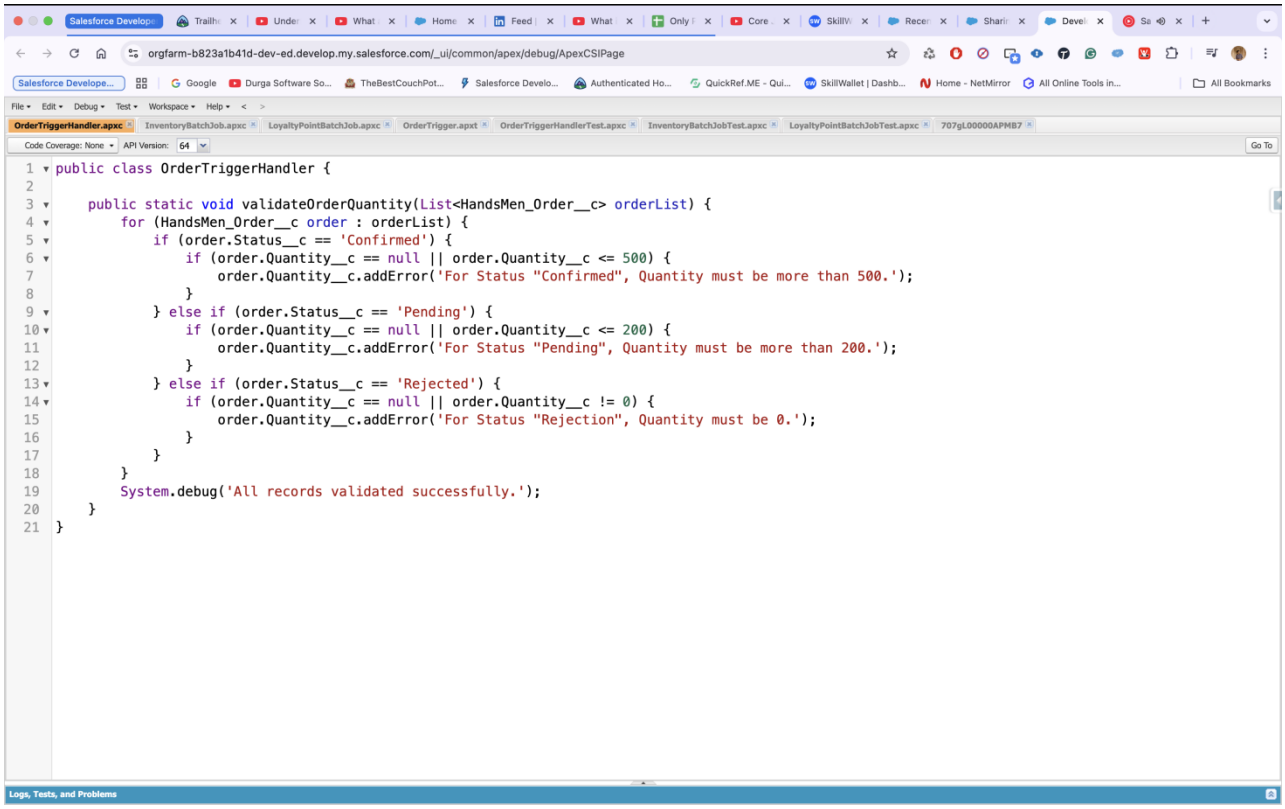


Apex Classes, Triggers, Asynchronous Apex Classes

- Trigger - OrderTrigger



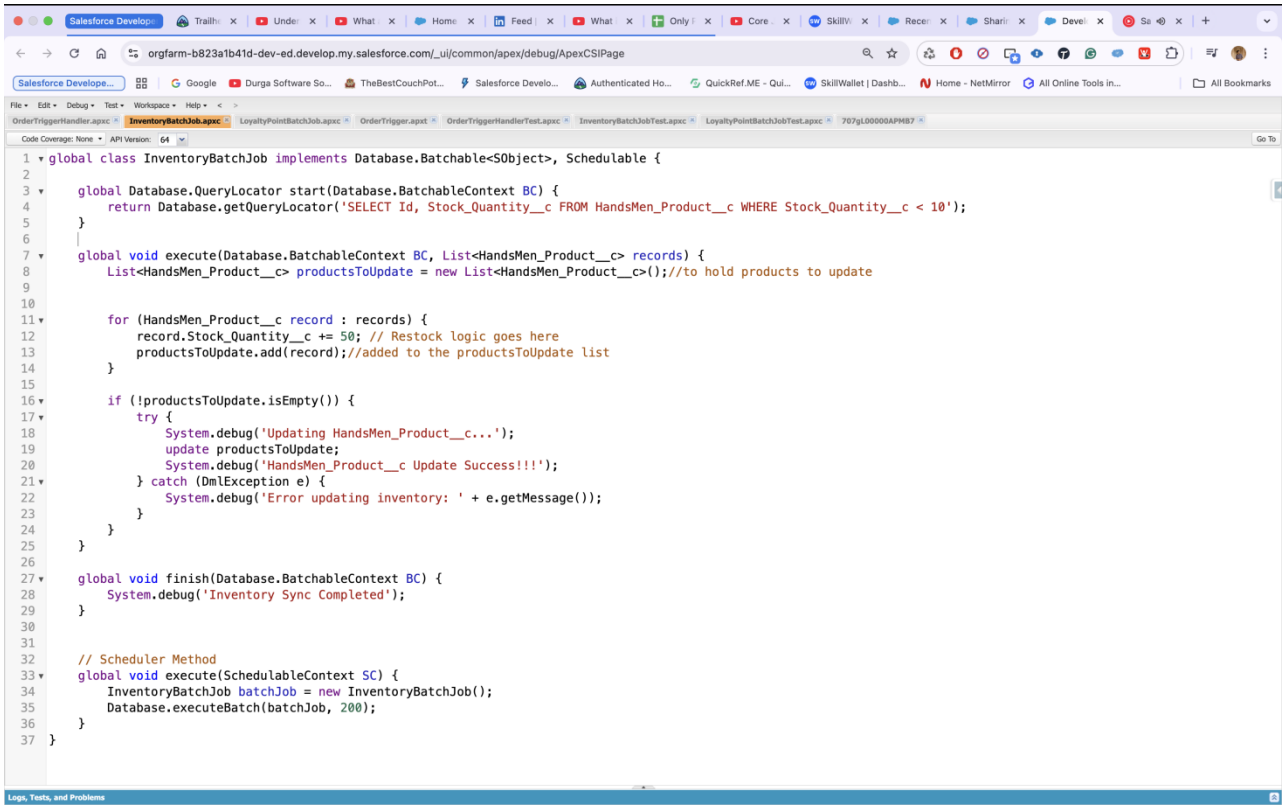
- Apex Class - OrderTriggerHandler



```apex
public class OrderTriggerHandler {

    public static void validateOrderQuantity(List<HandsMen_Order__c> orderList) {
        for (HandsMen_Order__c order : orderList) {
            if (order.Status__c == 'Confirmed') {
                if (order.Quantity__c == null || order.Quantity__c <= 500) {
                    order.Quantity__c.addError('For Status "Confirmed", Quantity must be more than 500.');
                }
            } else if (order.Status__c == 'Pending') {
                if (order.Quantity__c == null || order.Quantity__c <= 200) {
                    order.Quantity__c.addError('For Status "Pending", Quantity must be more than 200.');
                }
            } else if (order.Status__c == 'Rejected') {
                if (order.Quantity__c == null || order.Quantity__c != 0) {
                    order.Quantity__c.addError('For Status "Rejection", Quantity must be 0.');
                }
            }
        }
        System.debug('All records validated successfully.');
    }
}
```

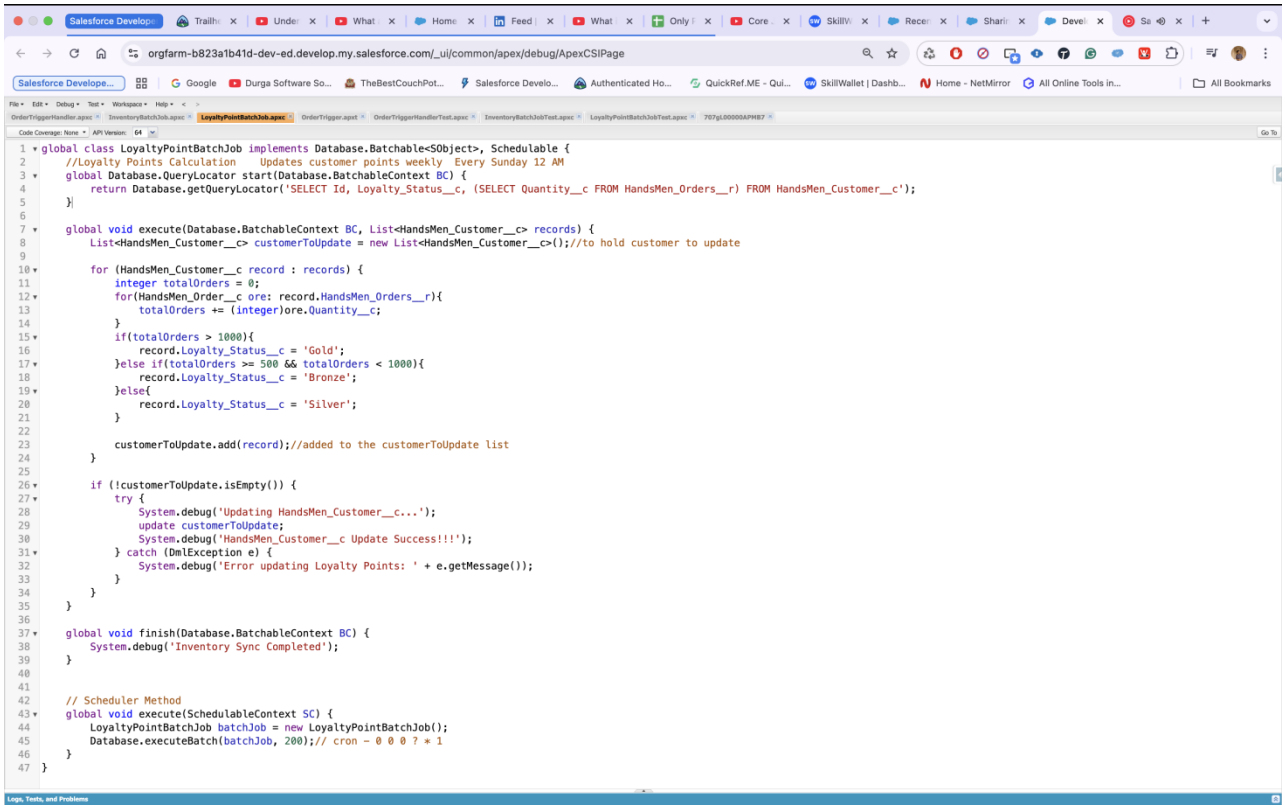
- Asynchronous Apex Classes

1. Inventory Batch Jobs



```apex
global class InventoryBatchJob implements Database.Batchable<SObject>, Schedulable {

    global Database.QueryLocator start(Database.BatchableContext BC) {
        return Database.getQueryLocator('SELECT Id, Stock_Quantity__c FROM HandsMen_Product__c WHERE Stock_Quantity__c < 10');
    }

    global void execute(Database.BatchableContext BC, List<HandsMen_Product__c> records) {
        List<HandsMen_Product__c> productsToUpdate = new List<HandsMen_Product__c>();//to hold products to update


        for (HandsMen_Product__c record : records) {
            record.Stock_Quantity__c += 50; // Restock logic goes here
            productsToUpdate.add(record);//added to the productsToUpdate list
        }

        if (!productsToUpdate.isEmpty()) {
            try {
                System.debug('Updating HandsMen_Product__c...');
                update productsToUpdate;
                System.debug('HandsMen_Product__c Update Success!!!');
            } catch (DmlException e) {
                System.debug('Error updating inventory: ' + e.getMessage());
            }
        }
    }

    global void finish(Database.BatchableContext BC) {
        System.debug('Inventory Sync Completed');
    }


    // Scheduler Method
    global void execute(SchedulableContext SC) {
        InventoryBatchJob batchJob = new InventoryBatchJob();
        Database.executeBatch(batchJob, 200);
    }
}
```

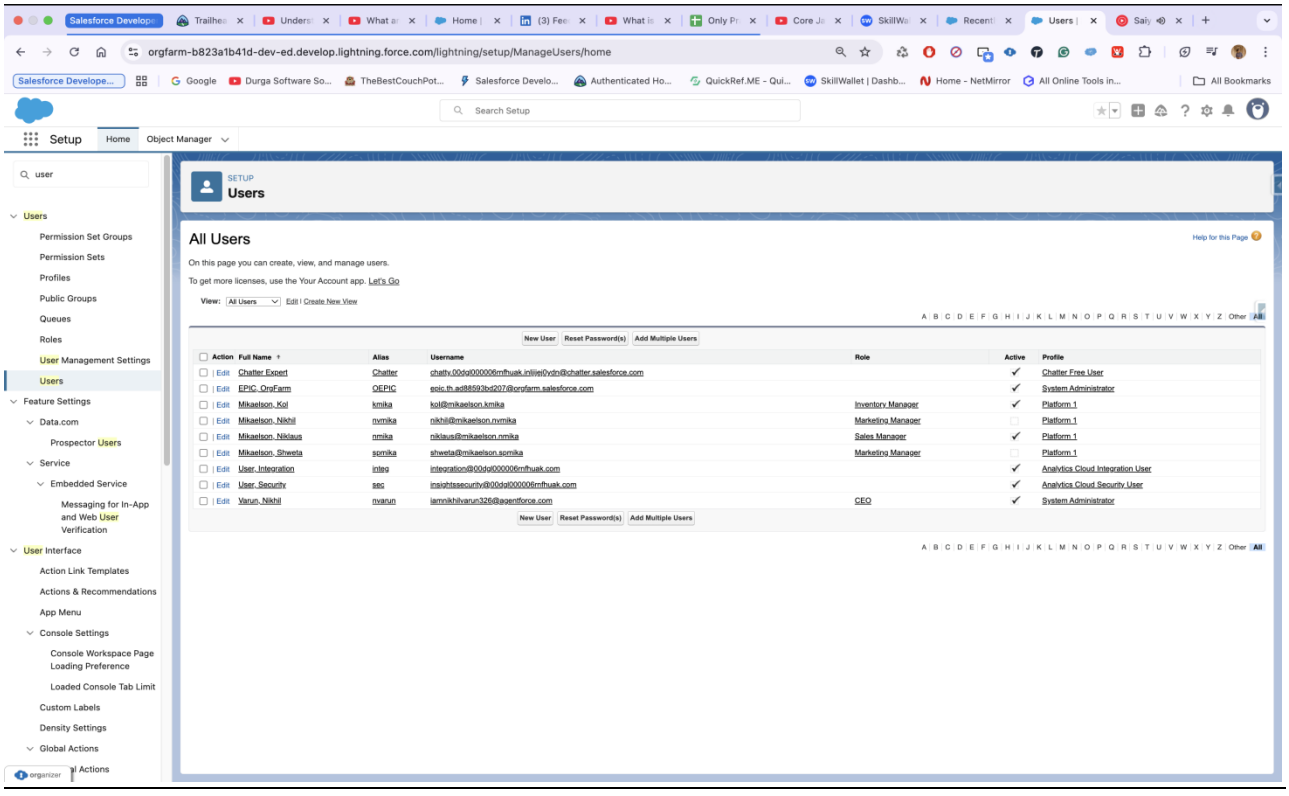
2. Loyalty Status Update Batch Job



```
global class LoyaltyPointBatchJob implements Database.Batchable<SObject>, Schedulable {
    //Loyalty Points Calculation    Updates customer points weekly  Every Sunday 12 AM
    global Database.QueryLocator start(Database.BatchableContext BC) {
        return Database.getQueryLocator('SELECT Id, Loyalty_Status__c, (SELECT Quantity__c FROM HandsMen_Orders__r) FROM HandsMen_Customer__c');
    }

    global void execute(Database.BatchableContext BC, List<HandsMen_Customer__c> records) {
        List<HandsMen_Customer__c> customerToUpdate = new List<HandsMen_Customer__c>();//to hold customer to update

        for (HandsMen_Customer__c record : records) {
            integer totalOrders = 0;
            for(HandsMen_Order__c ore: record.HandsMen_Orders__r){
                totalOrders += (integer)ore.Quantity__c;
            }
            if(totalOrders > 1000){
                record.Loyalty_Status__c = 'Gold';
            }else if(totalOrders >= 500 && totalOrders < 1000){
                record.Loyalty_Status__c = 'Bronze';
            }else{
                record.Loyalty_Status__c = 'Silver';
            }

            customerToUpdate.add(record);//added to the customerToUpdate list
        }

        if (!customerToUpdate.isEmpty()) {
            try {
                System.debug('Updating HandsMen_Customer__c...');
                update customerToUpdate;
                System.debug('HandsMen_Customer__c Update Success!!!');
            } catch (DmlException e) {
                System.debug('Error updating Loyalty Points: ' + e.getMessage());
            }
        }
    }

    global void finish(Database.BatchableContext BC) {
        System.debug('Inventory Sync Completed');
    }


    // Scheduler Method
    global void execute(SchedulableContext SC) {
        LoyaltyPointBatchJob batchJob = new LoyaltyPointBatchJob();
        Database.executeBatch(batchJob, 200);// cron - 0 0 0 ? * 1
    }
}
```

Phase 3:UI/UX Development & Customization

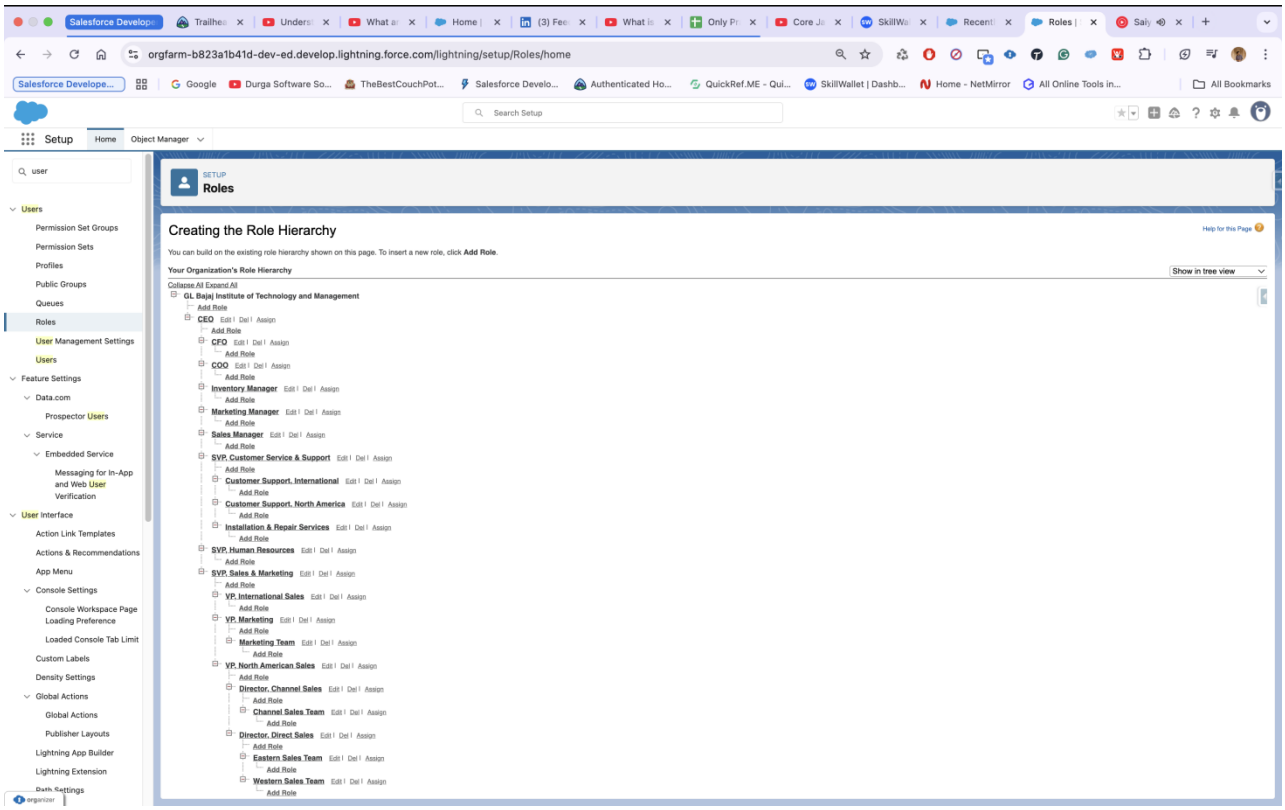
User Management



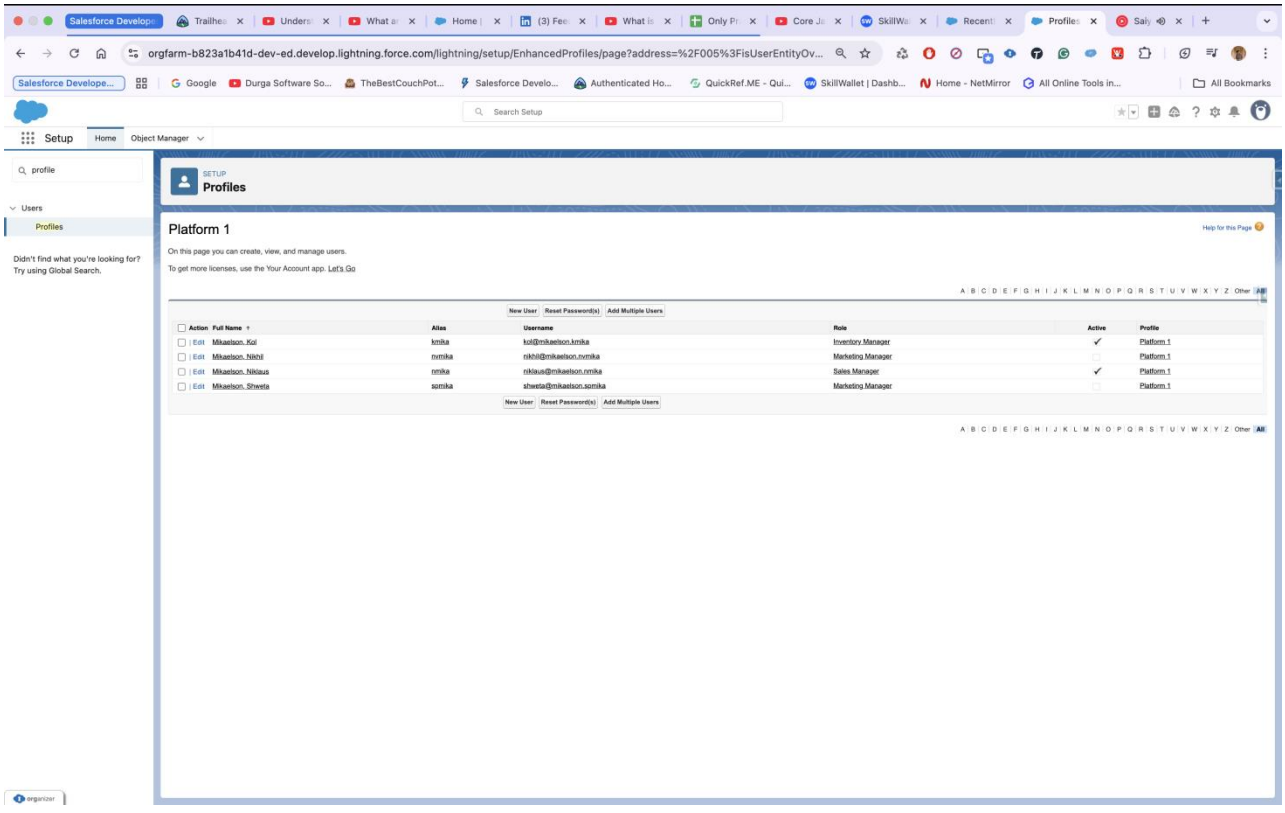
Phase 4:Data Migration, Testing & Security

Profiles, Roles and Role Hierarchy, Permission sets, Sharing Rules.

1. Roles

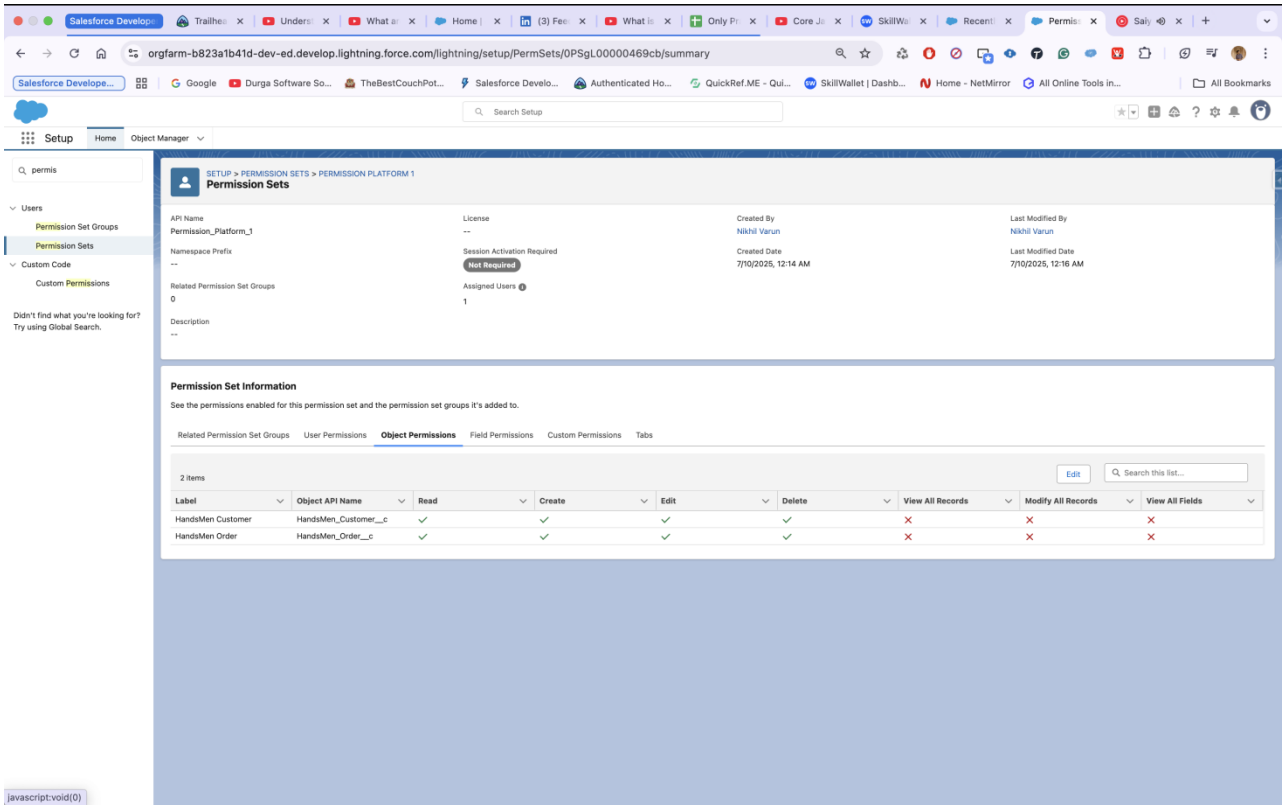


1. Profile - Platform 1 and Its Users

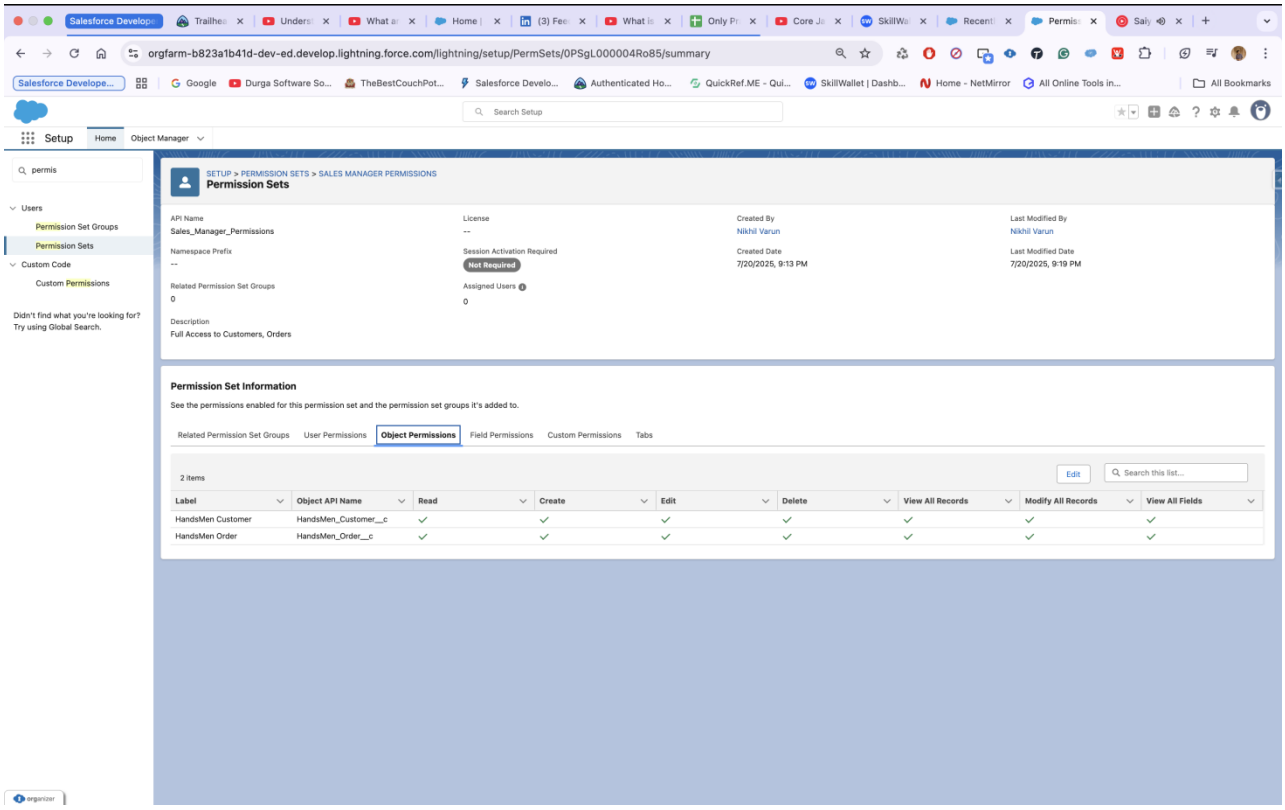


2. Permission Sets

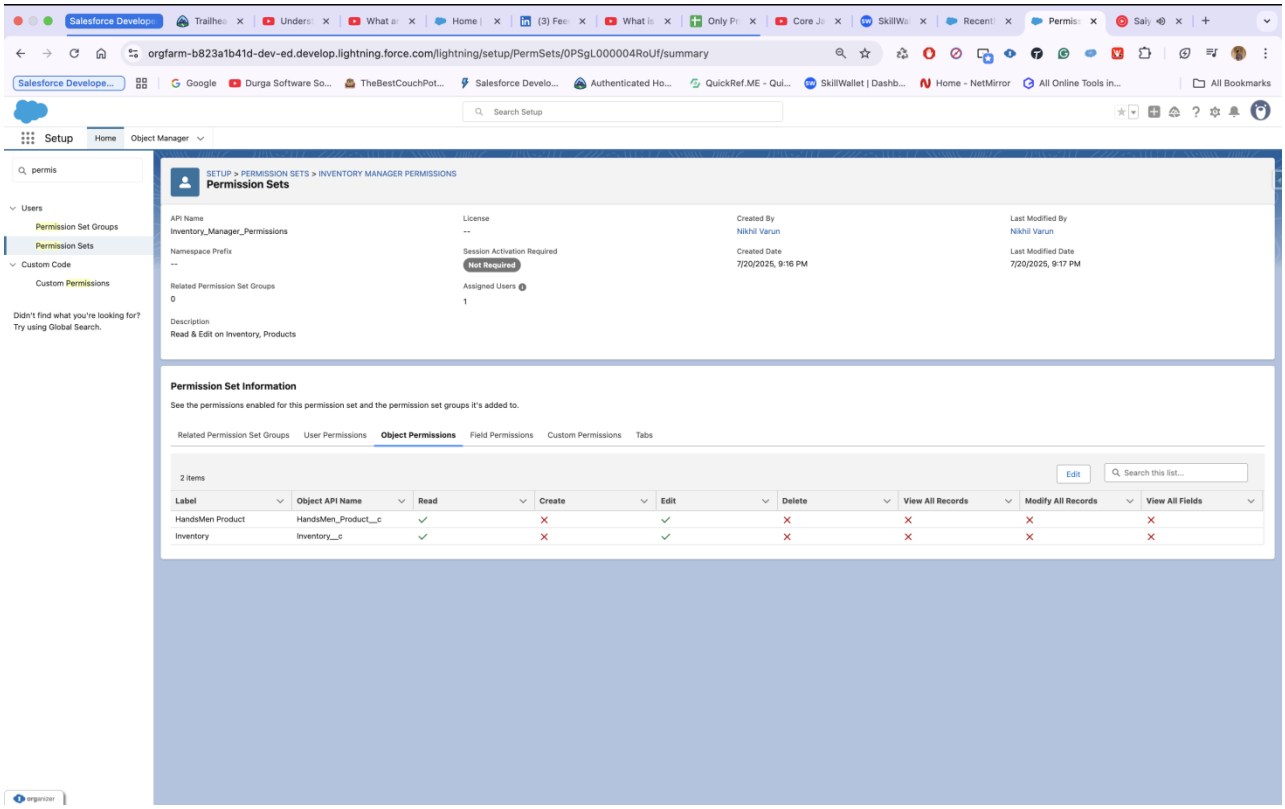
2.1 Platform 1 Permission Set



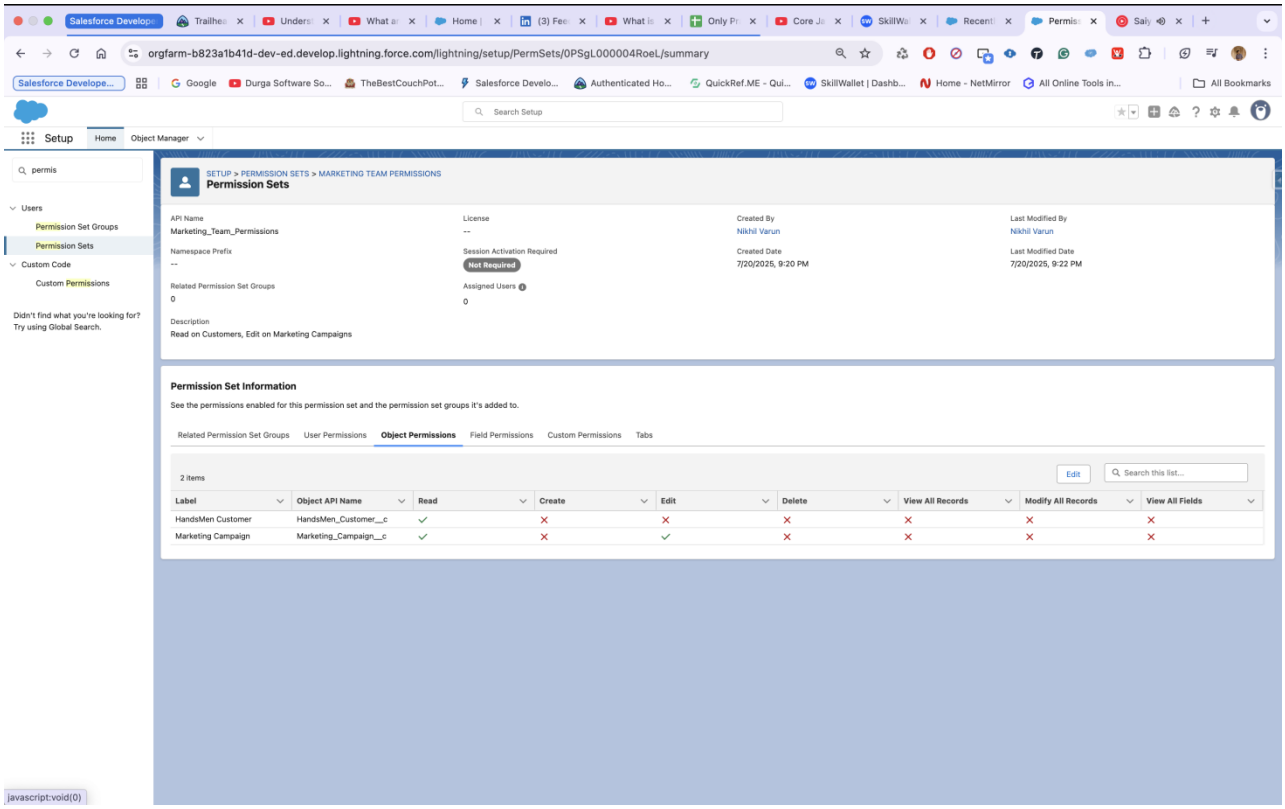
2.2 Sales Manager Permission Set



2.3 Inventory Manager Permission Set

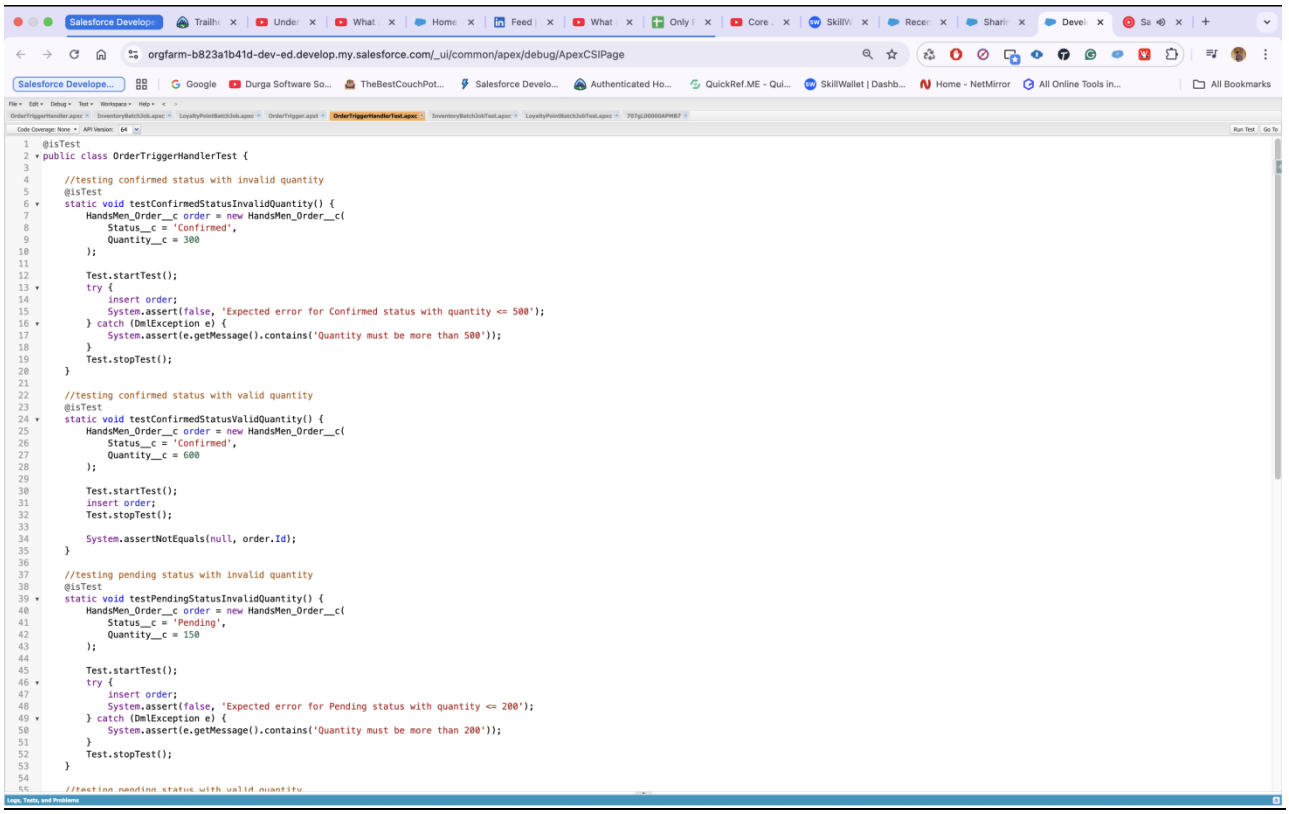


2.4 Marketing Team Permission Set

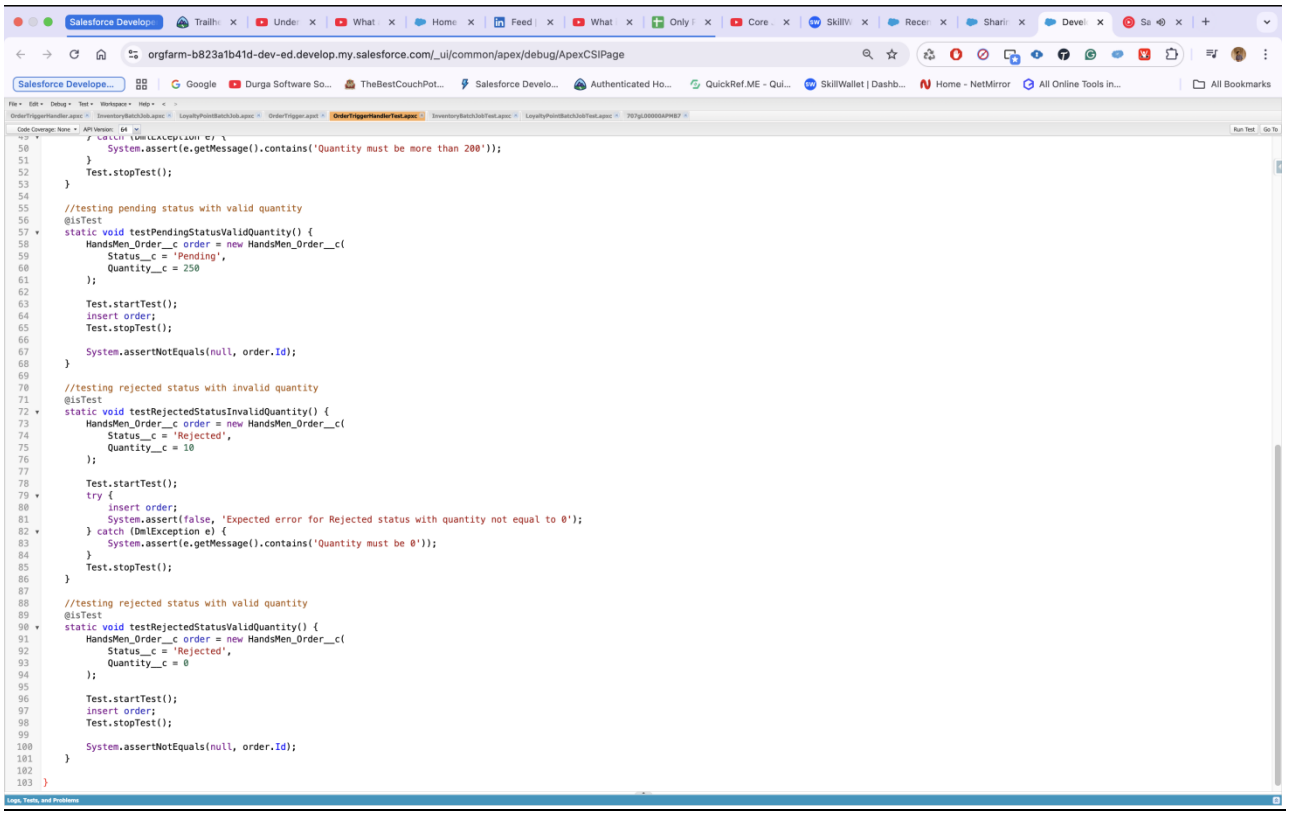


Creation of Test Classes

1. OrderTrigger Test

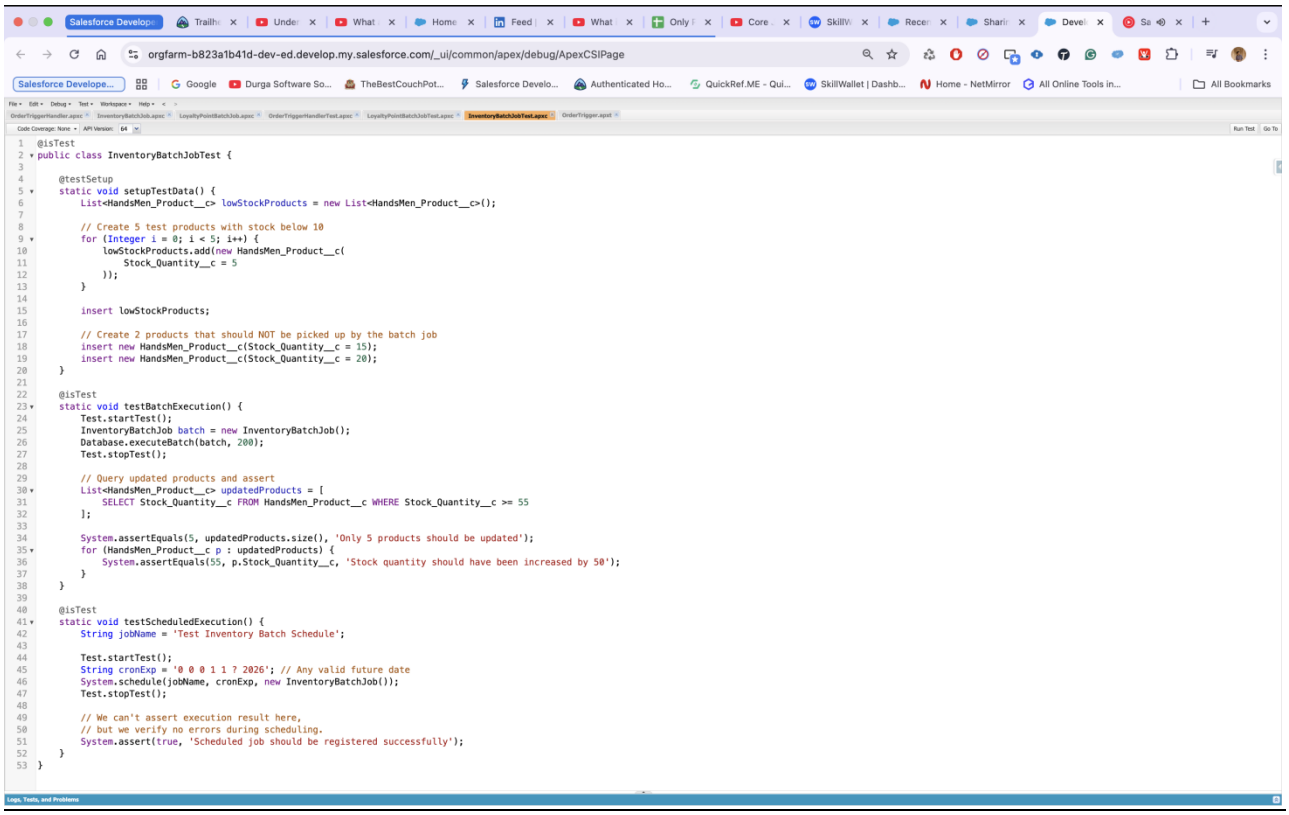


```
@isTest
public class OrderTriggerHandlerTest {

    //testing confirmed status with invalid quantity
    @isTest
    static void testConfirmedStatusInvalidQuantity() {
        HandsMen_Order__c order = new HandsMen_Order__c(
            Status__c = 'Confirmed',
            Quantity__c = 300
        );

        Test.startTest();
        try {
            insert order;
            System.assert(false, 'Expected error for Confirmed status with quantity <= 500');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('Quantity must be more than 500'));
        }
        Test.stopTest();
    }

    //testing confirmed status with valid quantity
    @isTest
    static void testConfirmedStatusValidQuantity() {
        HandsMen_Order__c order = new HandsMen_Order__c(
            Status__c = 'Confirmed',
            Quantity__c = 600
        );

        Test.startTest();
        insert order;
        Test.stopTest();

        System.assertNotEquals(null, order.Id);
    }

    //testing pending status with invalid quantity
    @isTest
    static void testPendingStatusInvalidQuantity() {
        HandsMen_Order__c order = new HandsMen_Order__c(
            Status__c = 'Pending',
            Quantity__c = 150
        );

        Test.startTest();
        try {
            insert order;
            System.assert(false, 'Expected error for Pending status with quantity <= 200');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('Quantity must be more than 200'));
        }
        Test.stopTest();
    }

    //testing pending status with valid quantity
```



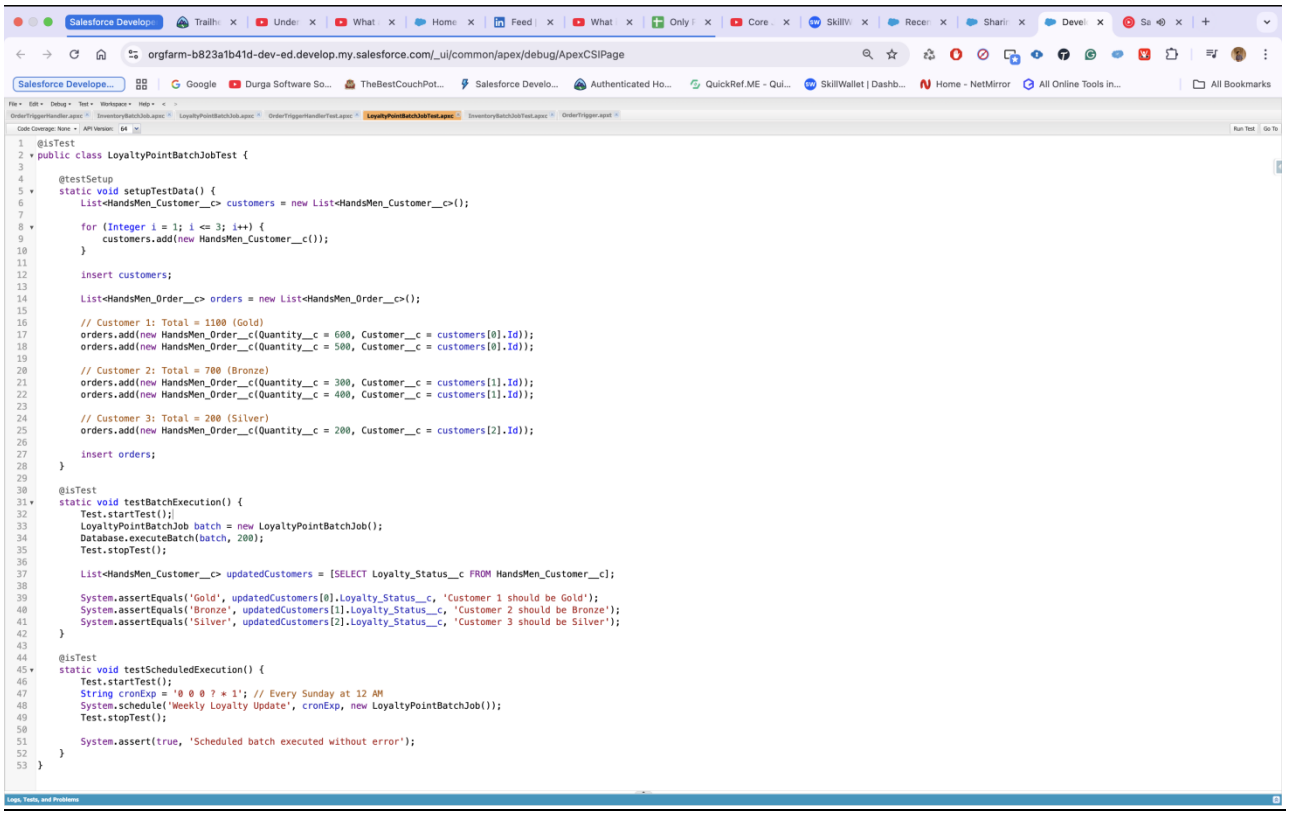
```
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('Quantity must be more than 200'));
        }
        Test.stopTest();
    }

    //testing pending status with valid quantity
    @isTest
    static void testPendingStatusValidQuantity() {
        HandsMen_Order__c order = new HandsMen_Order__c(
            Status__c = 'Pending',
            Quantity__c = 250
        );

        Test.startTest();
        insert order;
        Test.stopTest();

        System.assertNotEquals(null, order.Id);
    }

    //testing rejected status with invalid quantity
    @isTest
    static void testRejectedStatusInvalidQuantity() {
        HandsMen_Order__c order = new HandsMen_Order__c(
            Status__c = 'Rejected',
            Quantity__c = 10
        );

        Test.startTest();
        try {
            insert order;
            System.assert(false, 'Expected error for Rejected status with quantity not equal to 0');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('Quantity must be 0'));
        }
        Test.stopTest();
    }

    //testing rejected status with valid quantity
    @isTest
    static void testRejectedStatusValidQuantity() {
        HandsMen_Order__c order = new HandsMen_Order__c(
            Status__c = 'Rejected',
            Quantity__c = 0
        );

        Test.startTest();
        insert order;
        Test.stopTest();

        System.assertNotEquals(null, order.Id);
    }
}
```

2. Inventory Batch Job Test



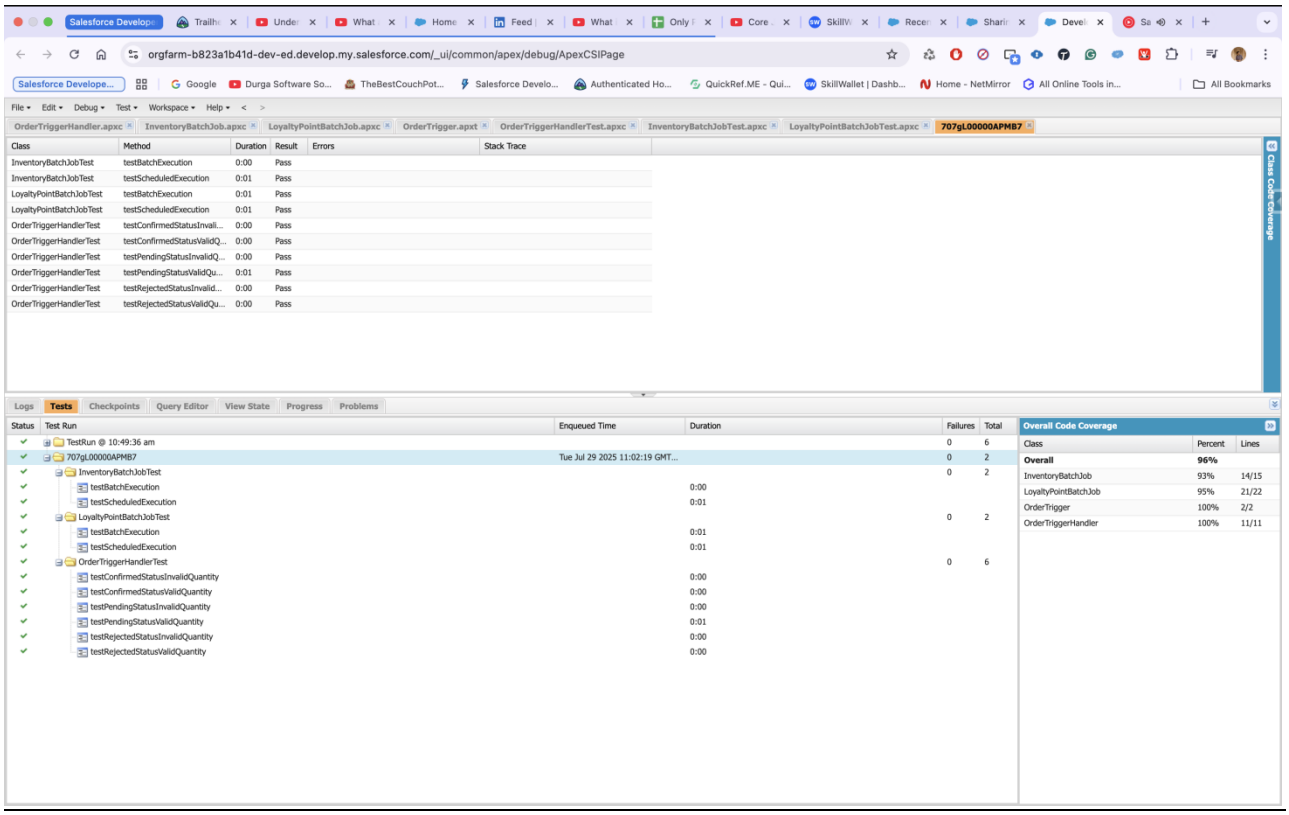
```
@isTest
public class InventoryBatchJobTest {

    @testSetup
    static void setupTestData() {
        List<HandsMen_Product__c> lowStockProducts = new List<HandsMen_Product__c>();

        // Create 5 test products with stock below 10
        for (Integer i = 0; i < 5; i++) {
            lowStockProducts.add(new HandsMen_Product__c(
                Stock_Quantity__c = 5
            ));
        }

        insert lowStockProducts;

        // Create 2 products that should NOT be picked up by the batch job
        insert new HandsMen_Product__c(Stock_Quantity__c = 15);
        insert new HandsMen_Product__c(Stock_Quantity__c = 20);
    }

    @isTest
    static void testBatchExecution() {
        Test.startTest();
        InventoryBatchJob batch = new InventoryBatchJob();
        Database.executeBatch(batch, 200);
        Test.stopTest();

        // Query updated products and assert
        List<HandsMen_Product__c> updatedProducts = [
            SELECT Stock_Quantity__c FROM HandsMen_Product__c WHERE Stock_Quantity__c >= 55
        ];

        System.assertEquals(5, updatedProducts.size(), 'Only 5 products should be updated');
        for (HandsMen_Product__c p : updatedProducts) {
            System.assertEquals(55, p.Stock_Quantity__c, 'Stock quantity should have been increased by 50');
        }
    }

    @isTest
    static void testScheduledExecution() {
        String jobName = 'Test Inventory Batch Schedule';

        Test.startTest();
        String cronExp = '0 0 0 1 1 ? 2026'; // Any valid future date
        System.schedule(jobName, cronExp, new InventoryBatchJob());
        Test.stopTest();

        // We can't assert execution result here,
        // but we verify no errors during scheduling.
        System.assert(true, 'Scheduled job should be registered successfully');
    }
}
```

3. Loyalty Point Test



```
@isTest
public class LoyaltyPointBatchJobTest {

    @testSetup
    static void setupTestData() {
        List<HandsMen_Customer__c> customers = new List<HandsMen_Customer__c>();

        for (Integer i = 1; i <= 3; i++) {
            customers.add(new HandsMen_Customer__c());
        }

        insert customers;

        List<HandsMen_Order__c> orders = new List<HandsMen_Order__c>();

        // Customer 1: Total = 1100 (Gold)
        orders.add(new HandsMen_Order__c(Quantity__c = 600, Customer__c = customers[0].Id));
        orders.add(new HandsMen_Order__c(Quantity__c = 500, Customer__c = customers[0].Id));

        // Customer 2: Total = 700 (Bronze)
        orders.add(new HandsMen_Order__c(Quantity__c = 300, Customer__c = customers[1].Id));
        orders.add(new HandsMen_Order__c(Quantity__c = 400, Customer__c = customers[1].Id));

        // Customer 3: Total = 200 (Silver)
        orders.add(new HandsMen_Order__c(Quantity__c = 200, Customer__c = customers[2].Id));

        insert orders;
    }

    @isTest
    static void testBatchExecution() {
        Test.startTest();
        LoyaltyPointBatchJob batch = new LoyaltyPointBatchJob();
        Database.executeBatch(batch, 200);
        Test.stopTest();

        List<HandsMen_Customer__c> updatedCustomers = [SELECT Loyalty_Status__c FROM HandsMen_Customer__c];

        System.assertEquals('Gold', updatedCustomers[0].Loyalty_Status__c, 'Customer 1 should be Gold');
        System.assertEquals('Bronze', updatedCustomers[1].Loyalty_Status__c, 'Customer 2 should be Bronze');
        System.assertEquals('Silver', updatedCustomers[2].Loyalty_Status__c, 'Customer 3 should be Silver');
    }

    @isTest
    static void testScheduledExecution() {
        Test.startTest();
        String cronExp = '0 0 0 ? * 1'; // Every Sunday at 12 AM
        System.schedule('Weekly Loyalty Update', cronExp, new LoyaltyPointBatchJob());
        Test.stopTest();

        System.assert(true, 'Scheduled batch executed without error');
    }
}
```

All Test Run & Coverage - Covering 96% - Required Limit is 75% - Test Acceptable



Phase 5: Deployment, Documentation & Maintenance

Explain the Deployment strategy (change sets or other methods).

- Change Sets

Change Sets are a native Salesforce deployment tool used to transfer customization—such as objects, fields, Apex classes, flows, and validation rules—from one Salesforce org to another.

They are commonly used to promote changes from a sandbox (development or testing environment) to a production org. Change Sets handle metadata only (the configuration and structure of the system), and do not include actual data like records.

1. Outbound Change Set

An outbound change set is created in the source org, when you're preparing to move components to another org.

Sending an outbound change set does not automatically apply the changes in the destination org. It must first be received and deployed in the target environment.

2. Inbound Change Set

An inbound change set is what appears in the target org after being sent from another Salesforce org.

Once received, the change set must be reviewed, validated, and deployed for the changes to take effect in the target environment. This step ensures control and accuracy before updates go live. Outbound change set will become inbound change set for the target org.

Basic description of how the system will be maintained and monitored.

- Regular Maintenance Tasks can be done:

I. User Access Reviews: Quarterly reviews of role and profile access.

II. Data Quality Checks: Scheduled reports for duplicate, incomplete, or stale records.

III. Batch & Scheduled Job Monitoring: Review Apex job logs weekly to ensure success.

- Monitoring Tools can be utilized:

I. Setup Audit Trail: Tracks configuration changes.

II. Debug Logs & Apex Exception Logs: Monitor errors and system behavior.

III. Health Check Tool: Review org security settings.

IV. Email Alerts: Trigger alerts on batch job failures, stock thresholds, or integration errors.