

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 have 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—with 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 Campaign*.
- ii. Relationships between *Handsmen Customer* and *Marketing Campaign*, *Handsmen Product* and *Handsmen Order*, *Handsmen Order* and *Handsmen Customer*, and *Inventory* and *Handsome 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

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

The screenshot shows the Salesforce Setup interface under the 'Lightning Experience App Manager'. The left sidebar navigation includes 'Data', 'Apps' (with 'App Manager' selected), 'Connected Apps', 'External Client Apps', 'Lightning Bolt', 'Mobile Apps', 'Salesforce', 'Packaging', and 'Settings'. The main content area displays a table titled '27 items · Sorted by App Name · Filtered by All appmruitems - TabSet Type, App Type'. The table lists various apps, including 'All Tabs', 'Analytics Studio', 'App Launcher', 'Approvals', 'Automation', 'Bolt Solutions', 'Community', 'Content', 'Data Cloud', 'Digital Experiences', 'Handsmen Threads' (which is highlighted in blue), 'Lightning Usage App', 'Marketing CRM Classic', 'My Service Journey', 'Platform', 'Queue Management', 'Sales', 'Sales Cloud Mobile', 'Salesforce Chatter', 'Salesforce Scheduler Setup', 'Service', 'Service Console', 'Site.com', and 'Subscription Management'. Each row contains columns for 'App Name', 'Developer Name', 'Description', 'Last Modified', 'App Type', and 'Vis...'. The 'Handsmen Threads' entry has a 'Developer Name' of 'Handsmen_Threads' and a 'Description' of 'Developing a premium fashion platform to streamline men's bespoke tailoring and enhance customer experience through personalized styling and seamless order management.'

- Tabs for - Handsmen Threads

The screenshot shows the Salesforce Setup interface under the 'Tabs' section. The left sidebar navigation includes 'User Interface' (with 'Rename Tabs and Labels' and 'Tabs' selected) and 'Global Search'. The main content area is titled 'Custom Tabs' with a sub-section 'Custom Object Tabs'. It displays a table with columns 'Action', 'Level', 'Tab Style' (with options like 'People', 'Hands', 'Bolt', 'Bank', and 'Ticket'), and 'Description'. The table lists custom tabs for 'Handsmen.Customers', 'Handsmen.Orders', 'Handsmen.Products', 'Inventories', and 'Marketing.Campaigns'. Below this, there are sections for 'Web Tabs' (no tabs defined), 'Visualforce Tabs' (no tabs defined), 'Lightning Component Tabs' (no tabs defined), and 'Lightning Page Tabs' (no tabs defined). The URL at the bottom of the page is <https://orgfarm-b823a1b41d-dev-ed.develop.lightning.force.com/lightning/setup/CustomTabs/home>.

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

● Custom Objects and Fields creation

1. Custom Object - Handmen Customer

The screenshot shows the Salesforce Setup interface with the 'Object Manager' selected. Under the 'Handmen Customer' object, the 'Fields & Relationships' section is displayed. The table lists various fields with their labels, field names, data types, and controlling fields. The 'INDEXED' column indicates whether each field is indexed.

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Email	Email_c	Email		
FirstName	FirstName__c	Text(50)		
Full_Name	Full_Name__c	Formula (Text)		
HandsMen Customer Name	Name	Text(80)		✓
Last Modified By	LastModifiedById	Lookup(User)		
LastName	LastName__c	Text(50)		
Loyalty Status	Loyalty_Status__c	Picklist		
Owner	OwnerId	Lookup(User,Group)		✓
Phone	Phone__c	Phone		
Total Purchases	Total_Purchases__c	Number(18, 0)		

2. Custom Object - Handsmen Product

The screenshot shows the Salesforce Setup interface with the 'Object Manager' selected. Under the 'HandsMen Product' object, the 'Fields & Relationships' section is displayed. The table lists various fields with their labels, field names, data types, and controlling fields. The 'INDEXED' column indicates whether each field is indexed.

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
HandsMen Product Name	Name	Text(80)		✓
Last Modified By	LastModifiedById	Lookup(User)		
Order	Order__c	Lookup(HandsMen Order)		✓
Owner	OwnerId	Lookup(User,Group)		✓
Price	Price__c	Currency(16, 2)		
SKU	SKU__c	Text(50)		
Stock Quantity	Stock_Quantity__c	Number(18, 0)		

3. Custom Object - Handmen Order

The screenshot shows the Salesforce Setup interface with the URL <https://orgfarm-b823a1b41d-dev-ed.lightning.force.com/lightning/setup/ObjectManager/01gL0000016j57/FieldsAndRelationships/view>. The page title is "Handmen Order". The left sidebar lists various setup categories like Details, Fields & Relationships, Page Layouts, etc. The main content area displays the "Fields & Relationships" table with the following data:

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		✓
Customer	Customer__c	Lookup(HandsMen Customer)		✓
HandsMen OrderNumber	Name	Auto Number		✓
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓
Quantity	Quantity__c	Number(18, 0)		
Status	Status__c	Picklist		
Total Amount	Total_Amount__c	Number(18, 0)		

4. Custom Object - Inventory

The screenshot shows the Salesforce Setup interface with the URL <https://orgfarm-b823a1b41d-dev-ed.lightning.force.com/lightning/setup/ObjectManager/01gL0000016j9x/FieldsAndRelationships/view>. The page title is "Inventory". The left sidebar lists various setup categories like Details, Fields & Relationships, Page Layouts, etc. The main content area displays the "Fields & Relationships" table with the following data:

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		✓
Inventory Number	Name	Auto Number		✓
Last Modified By	LastModifiedById	Lookup(User)		
Product	Product__c	Master-Detail(HandsMen Product)		✓
Stock Quantity	Stock_Quantity__c	Number(18, 0)		
Stock Status	Stock_Status__c	Formula (Text)		
Warehouse	Warehouse__c	Text(100)		

5. Custom Object - Marketing Campaign

The screenshot shows the Salesforce Setup interface for the Marketing Campaign object. The left sidebar lists various setup categories like Page Layouts, Lightning Record Pages, Buttons, etc. The main content area is titled "Fields & Relationships" and displays a table of fields:

	FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Page Layouts	Created By	CreatedBy	Lookup(User)		
Lightning Record Pages	End Date	End_Date__c	Date		
Buttons, Links, and Actions	HandsMen Customer	HandsMen_Customer__c	Lookup(HandsMen Customer)		
Compact Layouts	Last Modified By	LastModifiedBy	Lookup(User)		
Field Sets	Marketing Campaign Number	Name	Auto Number		
Object Limits	Owner	OwnerId	Lookup(User,Group)		
Record Types	Start Date	Start_Date__c	Date		

- Validation Rules Creation on 3 Objects and per business requirement

1. Validation Rule - Handsmen Customer

The screenshot shows the Salesforce Setup interface for the HandsMen Customer object. The left sidebar lists various setup categories. The main content area is titled "HandsMen Customer Validation Rule" and displays the details of a validation rule:

Rule Name	Email	Status
Validation Formula	NOT ISNULL(Email__c, '') AND Email__c != ''	Active
Error Message	Please fit Correct Email	
Description	Correct Email Check	Error Location
Created By	Nikhil Venk	Top of Page
		Modified By
		Nikhil Venk, 7/9/2025, 7:30 AM

2. Validation Rule - Handsmen Order

The screenshot shows the Salesforce Setup interface for the 'Handsmen Order' object. The left sidebar lists various setup categories like Details, Fields & Relationships, Page Layouts, etc. The 'Validation Rules' section is currently selected. The main content area displays the 'Validation Rule Detail' for the 'Handsmen Order Validation Rule'. The rule details are as follows:

Rule Name	Total_Amount	Active
Error Condition Formula	Total_Amount__c <= 0	<input checked="" type="checkbox"/>
Error Message	Please Enter Correct Amount	
Description		
Created By	Nikhil Varun, 7/9/2025, 7:26 AM	
Modified By	Nikhil Varun, 7/9/2025, 7:26 AM	

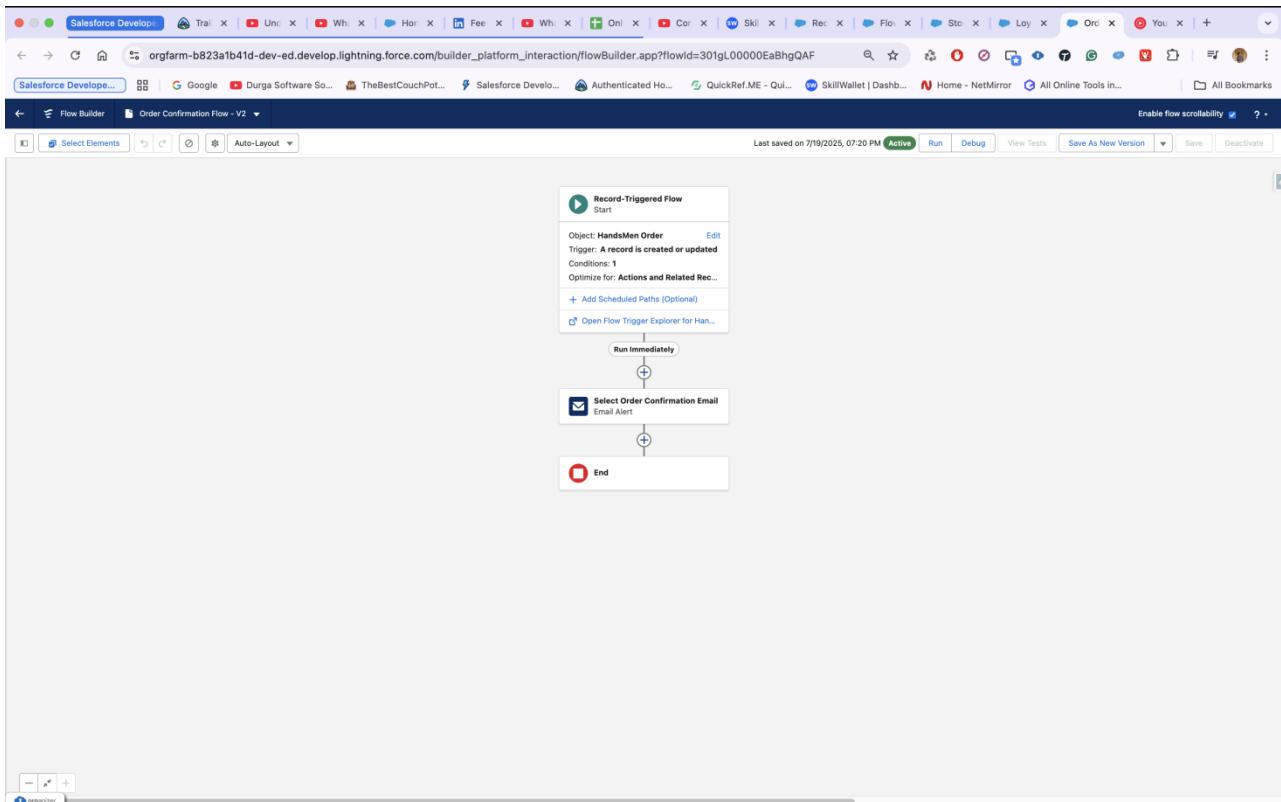
3. Validation Rule - Inventory

The screenshot shows the Salesforce Setup interface for the 'Inventory' object. The left sidebar lists various setup categories like Details, Fields & Relationships, Page Layouts, etc. The 'Validation Rules' section is currently selected. The main content area displays the 'Validation Rule Detail' for the 'Inventory Validation Rule'. The rule details are as follows:

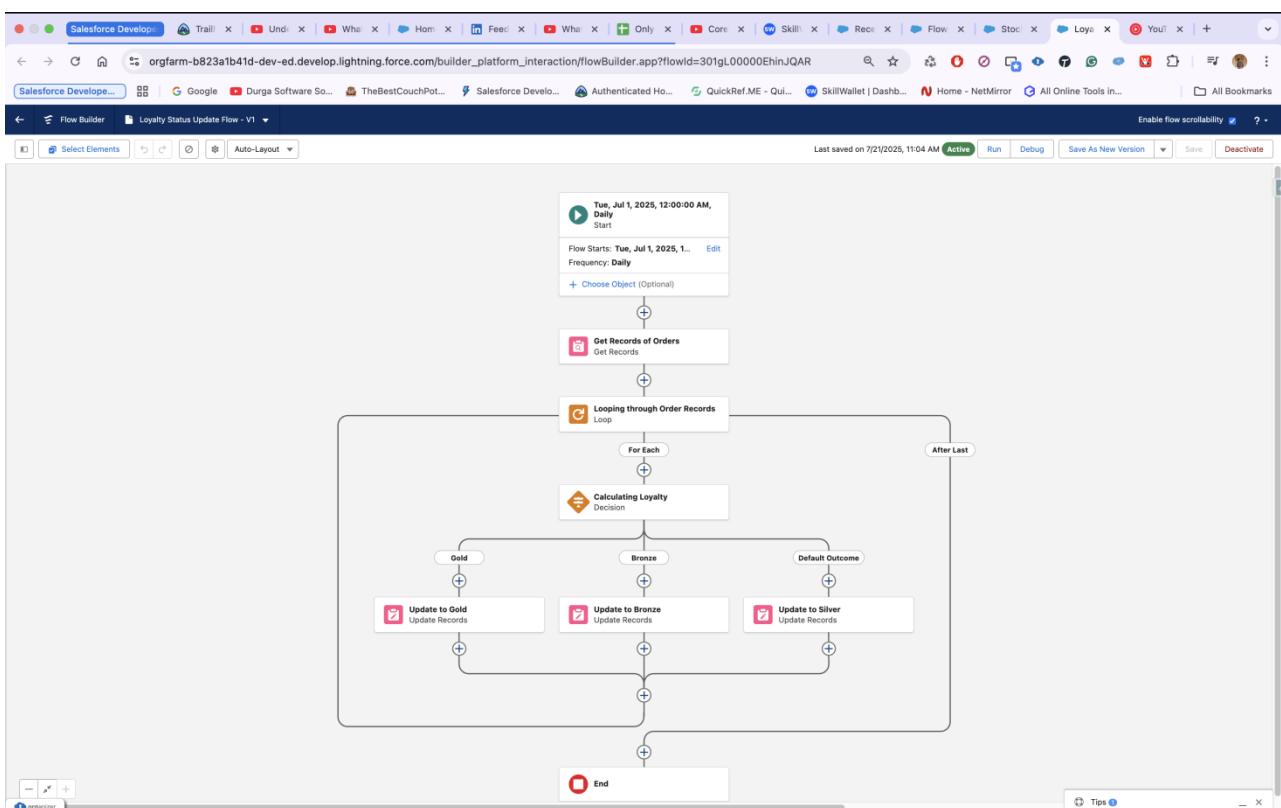
Rule Name	Stock_Quantity	Active
Error Condition Formula	Stock_Quantity__c <= 0	<input checked="" type="checkbox"/>
Error Message	The inventory count is always non-negative.	
Description	Stock Quantity Should not be less than 0	
Created By	Nikhil Varun, 7/9/2025, 7:24 AM	
Modified By	Nikhil Varun, 7/9/2025, 7:31 AM	

● Flows Creation

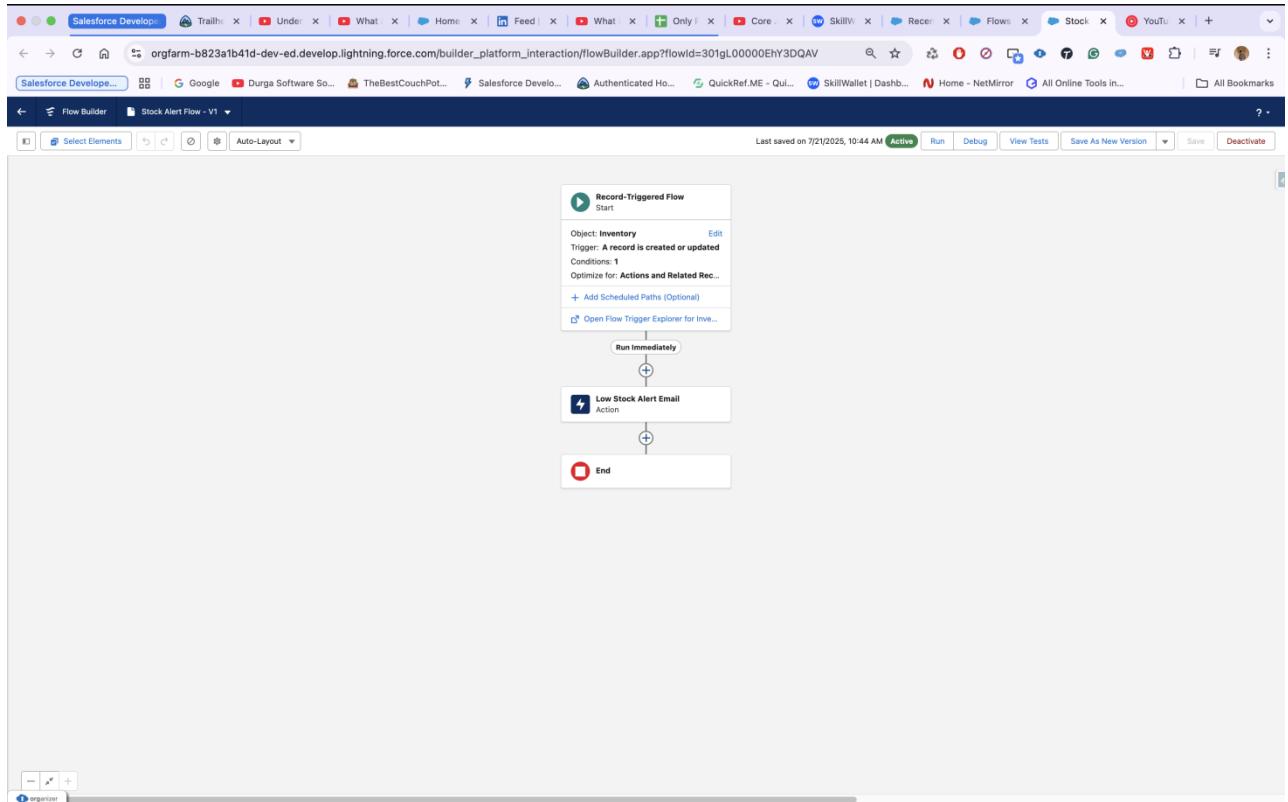
1. Flow - Record Triggered Flow - Email Alert



2. Flow - Scheduled Flow - Loyalty Status Update



3. Flow - Record Triggered Flow - Low Stock Alert



● Email Templates

1. Order Confirmation

The screenshot shows the Salesforce Setup interface under the "Email" section. A new "Classic Email Template" is being created for "Order Confirmation Email". The template details include:

- Email Template Name:** Order Confirmation Email
- Template Unique Name:** Order_Confirmation_Email
- Classic Letterhead:** HandsMen_Threads_Letterhead
- Email Layout:** Format Letter
- Encoding:** Unicode (UTF-8)
- Author:** Nikhil Varun [Change]
- Description:** Created By Nikhil Varun, 7/10/2025, 12:33 AM

The "Available For Use" checkbox is checked. The "HTML Preview" section shows the email content:

```
<p>Dear {!HandsMen_Order__c.Customer__c},</p>
<p>Your order # {!HandsMen_Order__c.Name} has been confirmed!</p>
<p>Thank you for shopping with us.</p>
<p>Best Regards,</p>
<p>Sales Team</p>
```

2. Low Stock Alert

The screenshot shows the Salesforce Setup interface with the 'Classic Email Templates' page open. A new email template named 'Low Stock Alert' has been created. The template details are as follows:

- Email Template Detail:**
 - Email Template Name: Low Stock Alert
 - Template Unique Name: Low_Stock_Alert
 - Encoding: Unicode (UTF-8)
 - Author: Nikhil Varun [Change]
 - Description: Created By Nikhil Varun, 7/29/2025, 5:48 AM
- Email Template:**
 - Subject: Low Stock Alert!!
 - Plain Text Preview:

```
<p>Dear {!Inventory__c.CreatedBy},</p>
<p>Your product # {!Inventory__c.Product__c} have low stock - # {!Inventory__c.Stock_Quantity__c}</p>
<p>Please Refill.</p>
<p>Best Regards,</p>
```
 - Send Test and Verify Merge Fields
- Attachments:** No records to display.

3. Loyalty Program

The screenshot shows the Salesforce Setup interface with the 'Classic Email Templates' page open. A new email template named 'Loyalty Program Email' has been created. The template details are as follows:

- Email Template Detail:**
 - Email Template Name: Loyalty Program Email
 - Template Unique Name: Loyalty_Program_Email
 - Classic Letterhead: HandsMen Threads Letterhead
 - Email Layout: Formal Letter
 - Encoding: Unicode (UTF-8)
 - Author: Nikhil Varun [Change]
 - Description: Created By Nikhil Varun, 7/10/2025, 12:51 AM
- Email Template:**
 - Subject: Loyalty Program Email
 - HTML Preview:

```
<p>Dear {!HandsMen_Customer__c.Name},</p>
<p>This is your # {!HandsMen_Customer__c.Loyalty_Status__c} loyalty status.</p>
<p>Thank you for shopping with us.</p>
<p>Best Regards,</p>
<p>Sales Team</p>
```
 - Send Test and Verify Merge Fields

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

Screenshot of the Salesforce Setup interface showing the creation of an Email Alert named "Order Confirmation Email Alert". The alert is assigned to "User: Niklaus Mikaelson" and uses the "Order Confirmation Email" template for "HandsMen Order" objects. The page also shows sections for "Rules Using This Email Alert", "Approval Processes Using This Email Alert", "Entitlement Processes Using This Email Alert", and "Flows Using This Email Alert".

Apex Classes, Triggers, Asynchronous Apex Classes

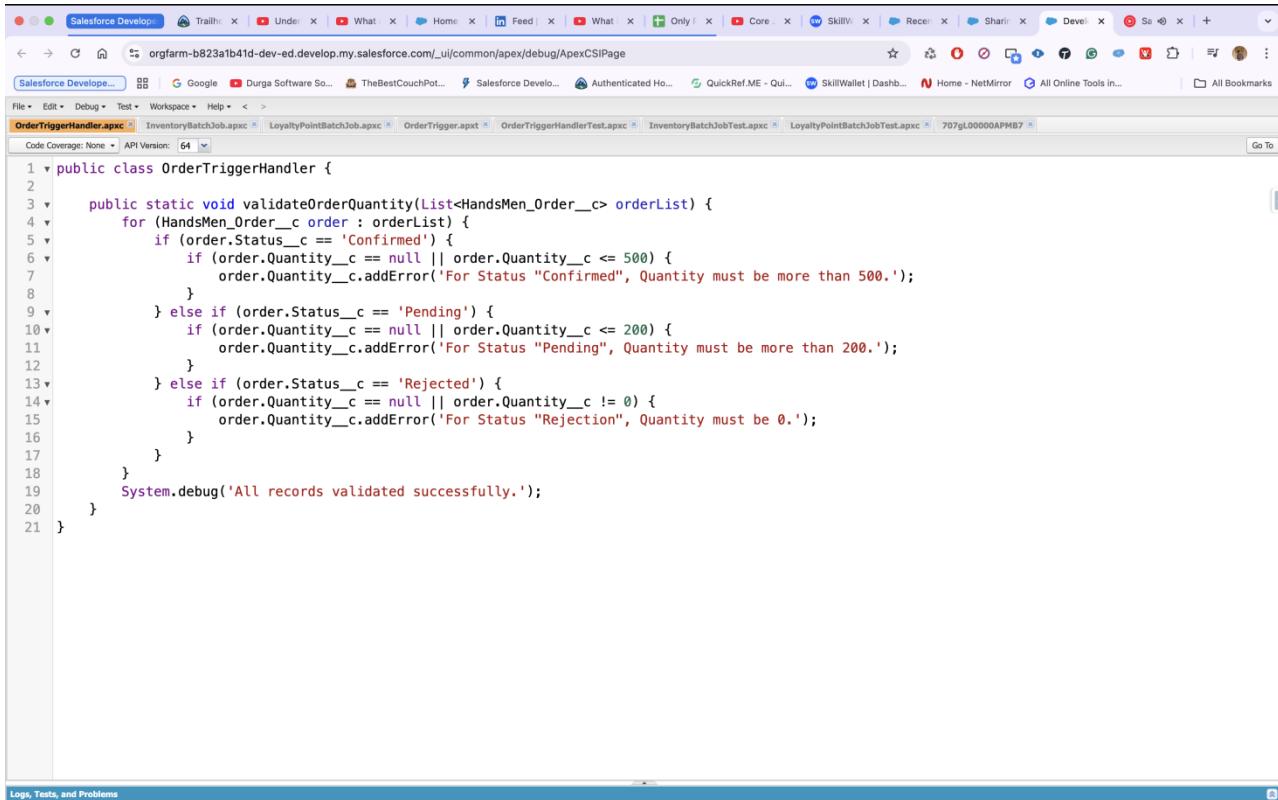
- Trigger - OrderTrigger

```

1 trigger OrderTrigger on HandsMen_Order__c (before insert, before update) {
2     if ((Trigger.isBefore && (Trigger.isInsert || Trigger.isUpdate)) {
3         OrderTriggerHandler.validateOrderQuantity(Trigger.new);
4     }
5 }

```

● Apex Class - OrderTriggerHandler

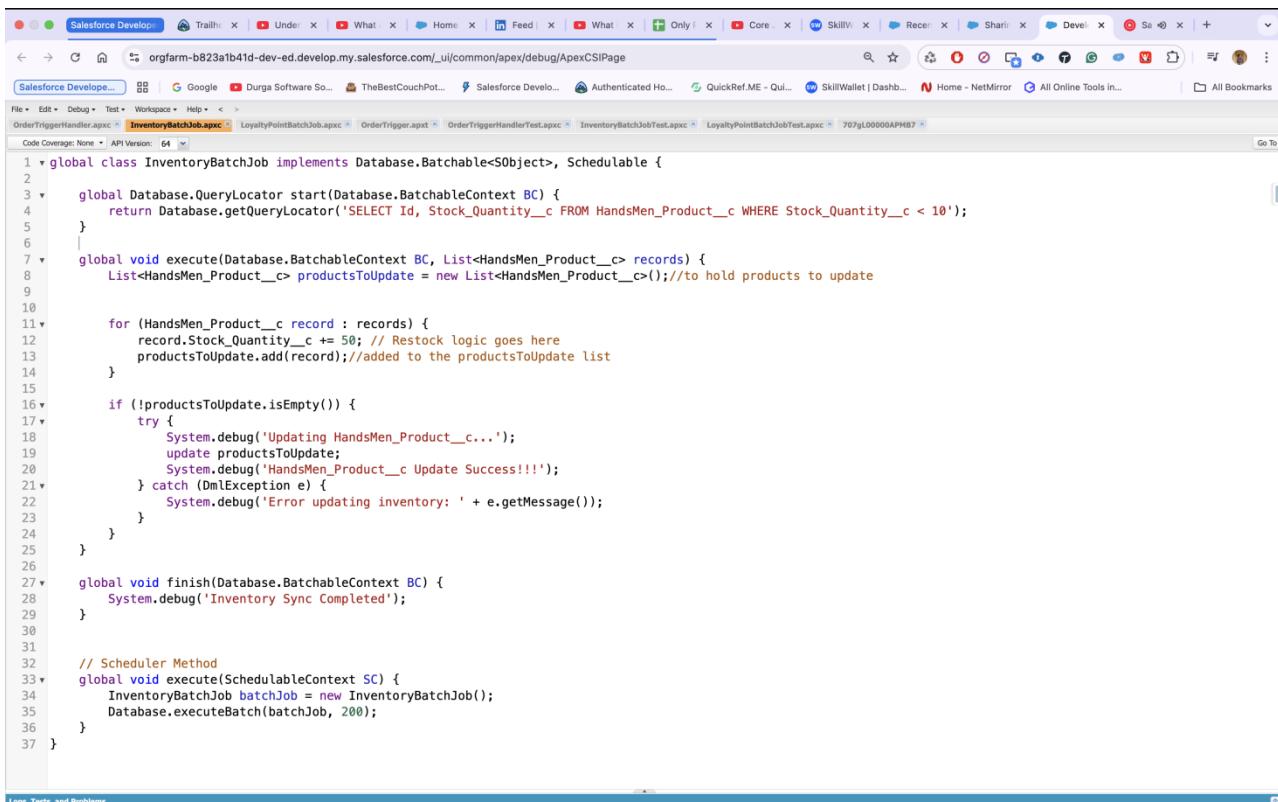


The screenshot shows the Salesforce Developer Console with the OrderTriggerHandler.apxc file open. The code implements a static method validateOrderQuantity that iterates through a list of Orders. It checks the status of each order. If the status is 'Confirmed' and the quantity is null or less than or equal to 500, it adds an error message. If the status is 'Pending' and the quantity is null or less than or equal to 200, it adds an error message. If the status is 'Rejected' and the quantity is null or not equal to 0, it adds an error message. Finally, it prints a success message to the debug log.

```
1 public class OrderTriggerHandler {
2
3     public static void validateOrderQuantity(List<HandsMen_Order__c> orderList) {
4         for (HandsMen_Order__c order : orderList) {
5             if (order.Status__c == 'Confirmed') {
6                 if (order.Quantity__c == null || order.Quantity__c <= 500) {
7                     order.Quantity__c.addError('For Status "Confirmed", Quantity must be more than 500.');
8                 }
9             } else if (order.Status__c == 'Pending') {
10                if (order.Quantity__c == null || order.Quantity__c <= 200) {
11                    order.Quantity__c.addError('For Status "Pending", Quantity must be more than 200.');
12                }
13            } else if (order.Status__c == 'Rejected') {
14                if (order.Quantity__c == null || order.Quantity__c != 0) {
15                    order.Quantity__c.addError('For Status "Rejection", Quantity must be 0.');
16                }
17            }
18        }
19        System.debug('All records validated successfully.');
20    }
21 }
```

● Asynchronous Apex Classes

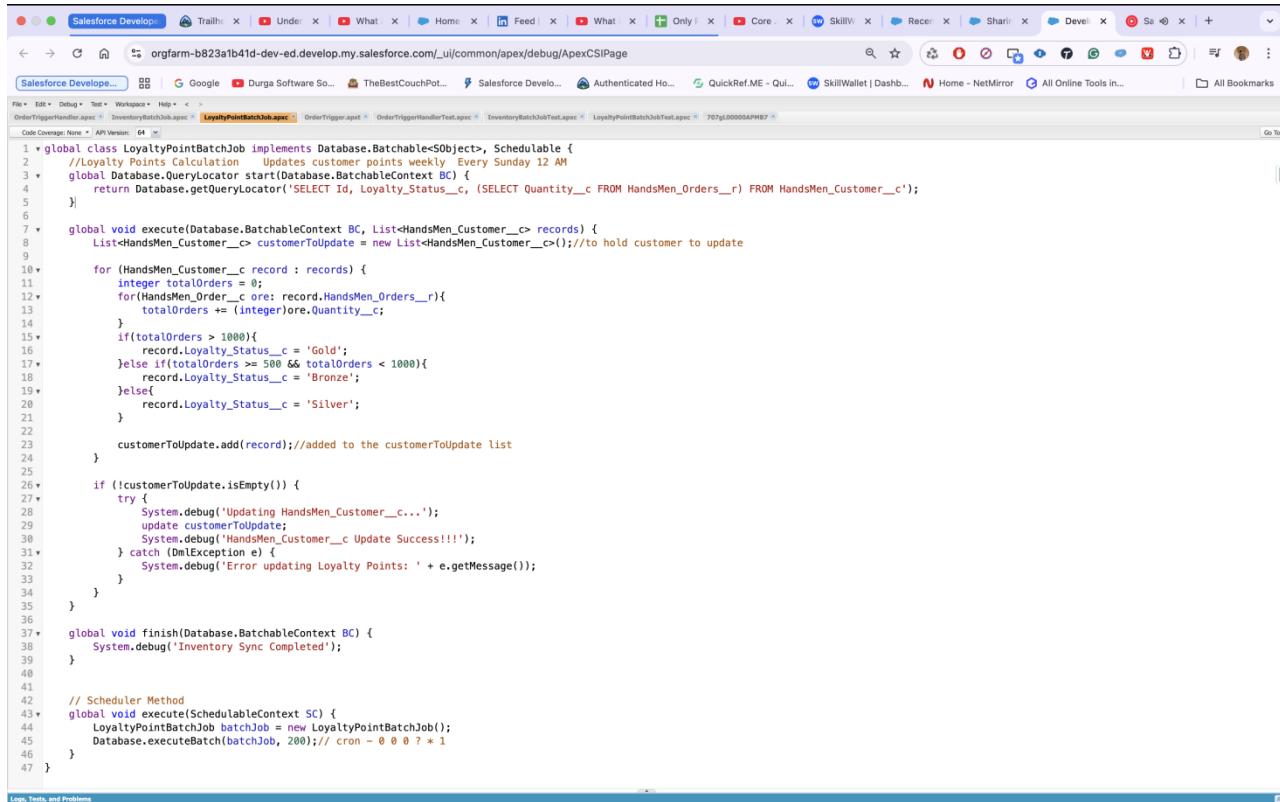
1. Inventory Batch Jobs



The screenshot shows the Salesforce Developer Console with the InventoryBatchJob.apxc file open. This is a scheduled Apex class that implements the Database.Batchable interface. It starts by querying for products with a stock quantity less than 10. It then loops through these records, updating their stock quantity by adding 50. A list of products to update is maintained. If there are products to update, it tries to update them. If an exception occurs during the update, it is caught and the error message is logged. Finally, it logs a success message indicating the inventory sync is completed.

```
1 *global class InventoryBatchJob implements Database.Batchable<SObject>, Schedulable {
2
3     *global Database.QueryLocator start(Database.BatchableContext BC) {
4         return Database.getQueryLocator('SELECT Id, Stock_Quantity__c FROM HandsMen_Product__c WHERE Stock_Quantity__c < 10');
5     }
6
7     *global void execute(Database.BatchableContext BC, List<HandsMen_Product__c> records) {
8         List<HandsMen_Product__c> productsToUpdate = new List<HandsMen_Product__c>(); //to hold products to update
9
10    for (HandsMen_Product__c record : records) {
11        record.Stock_Quantity__c += 50; // Restock logic goes here
12        productsToUpdate.add(record); //added to the productsToUpdate list
13    }
14
15    if (!productsToUpdate.isEmpty()) {
16        try {
17            System.debug('Updating HandsMen_Product__c...');
18            update productsToUpdate;
19            System.debug('HandsMen_Product__c Update Success!!!!');
20        } catch (DmlException e) {
21            System.debug('Error updating inventory: ' + e.getMessage());
22        }
23    }
24 }
25
26
27     *global void finish(Database.BatchableContext BC) {
28         System.debug('Inventory Sync Completed');
29     }
30
31
32     // Scheduler Method
33     *global void execute(SchedulableContext SC) {
34         InventoryBatchJob batchJob = new InventoryBatchJob();
35         Database.executeBatch(batchJob, 200);
36     }
37 }
```

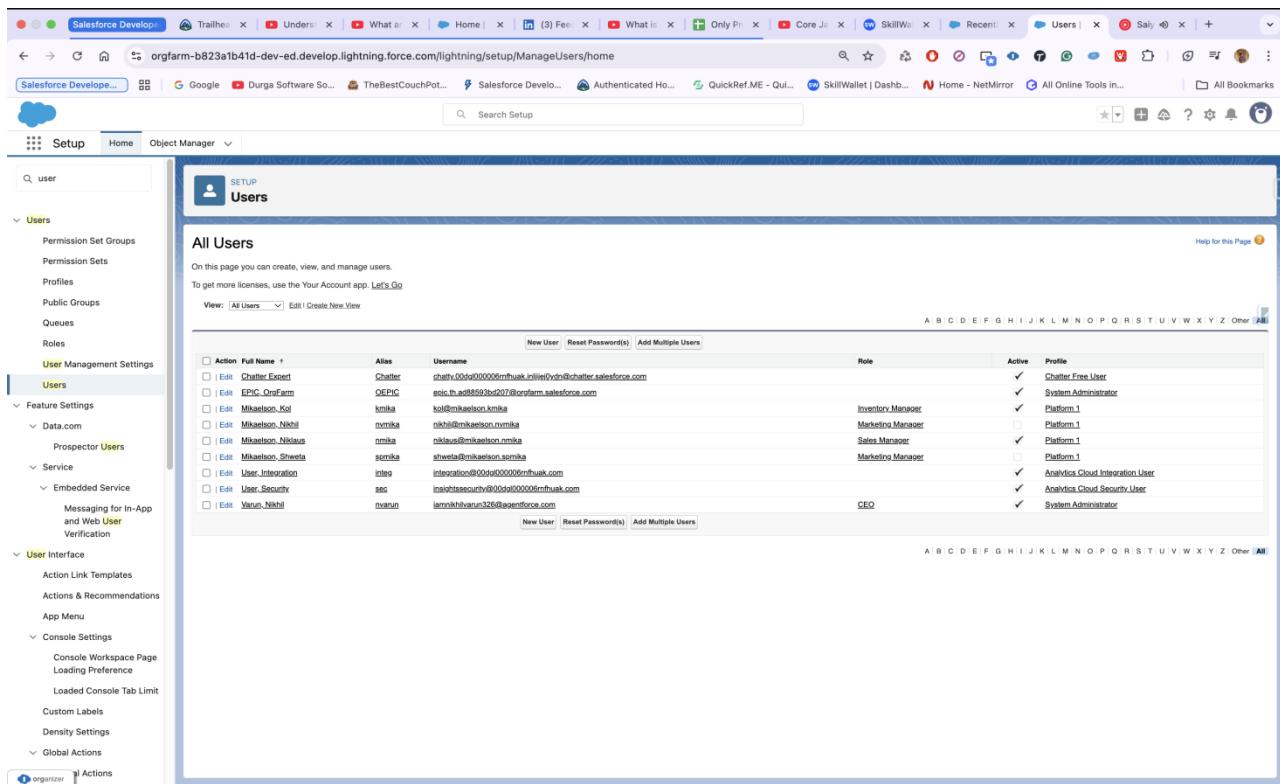
2. Loyalty Status Update Batch Job



```
1 *global class LoyaltyPointBatchJob implements Database.Batchable<SObject>, Schedulable {
2     //Loyalty Points Calculation Updates customer points weekly Every Sunday 12 AM
3     global Database.QueryLocator start(Database.BatchableContext BC) {
4         return Database.getQueryLocator('SELECT Id, Loyalty_Status__c, (SELECT Quantity__c FROM HandsMen_Orders__r) FROM HandsMen_Customer__c');
5     }
6     global void execute(Database.BatchableContext BC, List<HandsMen_Customer__c> records) {
7         List<HandsMen_Customer__c> customerToUpdate = new List<HandsMen_Customer__c>();//to hold customer to update
8
9         for (HandsMen_Customer__c record : records) {
10             integer totalOrders = 0;
11             for(HandsMen_Order__c ore: record.HandsMen_Orders__r){
12                 totalOrders += (integer)ore.Quantity__c;
13             }
14             if(totalOrders > 1000){
15                 record.Loyalty_Status__c = 'Gold';
16             }else if(totalOrders >= 500 && totalOrders < 1000){
17                 record.Loyalty_Status__c = 'Bronze';
18             }else{
19                 record.Loyalty_Status__c = 'Silver';
20             }
21
22             customerToUpdate.add(record);//added to the customerToUpdate list
23         }
24     }
25
26     if (!customerToUpdate.isEmpty()) {
27         try {
28             System.debug('Updating HandsMen_Customer__c...');  
update customerToUpdate;  
System.debug('HandsMen_Customer__c Update Success!!!!');
29         } catch (DmlException e) {
30             System.debug('Error updating Loyalty Points: ' + e.getMessage());
31         }
32     }
33 }
34 }
35 }
36
37 *global void finish(Database.BatchableContext BC) {
38     System.debug('Inventory Sync Completed');
39 }
40
41
42 // Scheduler Method
43 *global void execute(SchedulableContext SC) {
44     LoyaltyPointBatchJob batchJob = new LoyaltyPointBatchJob();
45     Database.executeBatch(batchJob, 200); // cron - 0 0 0 ? * 1
46 }
47 }
```

Phase 3: UI/UX Development & Customization

User Management



The screenshot shows the Salesforce Lightning User Management page. On the left, there's a sidebar with navigation links like Setup, Home, Object Manager, and various system settings. The main content area is titled "All Users" and displays a list of users. Each user entry includes columns for Action, Full Name, Alias, Username, Role, Active status, and Profile. The users listed are Chatter_Expert, QERIC, Mikaelson_Kol, rmika, Mikaelson_Nikhil, rmika, Mikaelson_Niklaus, rmika, Mikaelson_Shetra, somika, User_Integration, intergration@00000000000000000000000000000000, User_Security, sec, and Varun.Nikhil, mvarun. The interface is clean and modern, typical of the Lightning Experience.

Phase 4: Data Migration, Testing & Security

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

1. Roles

The screenshot shows the 'Roles' section of the Salesforce Setup. On the left, a sidebar lists various setup categories like Users, Profiles, and Roles. The 'Roles' category is selected. The main content area is titled 'Creating the Role Hierarchy' and displays a hierarchical tree of roles under 'Your Organization's Role Hierarchy'. At the top level is 'GL Bajaj Institute of Technology and Management'. Below it are several roles: CEO, CFO, COO, Inventory Manager, Marketing Manager, Sales Manager, SVP Customer Service & Support, Customer Support, International, Customer Support, North America, Installation & Repair Services, SVP Human Resources, SVP Sales & Marketing, VP International Sales, VP Marketing, Marketing Team, VP North American Sales, Director Channel Sales, Director Direct Sales, Eastern Sales Team, and Western Sales Team. Each role has 'Edit | Del | Assign' buttons next to it. A 'Show in tree view' link is located in the top right corner of the hierarchy area.

1. Profile - Platform 1 and Its Users

The screenshot shows the 'Profiles' section of the Salesforce Setup. The sidebar shows 'Profiles' is selected. The main content area is titled 'Platform 1' and contains a message: 'On this page you can create, view, and manage users. To get more licenses, use the Your Account app. Let's Go'. Below this is a table for managing users:

Action	Full Name	Alias	Username	Role	Active	Profile
<input type="checkbox"/> Edit	Mitaelson_Xol	kmla	kmla@mitaelson.kmla	Inventory Manager	<input checked="" type="checkbox"/>	Platform 1
<input type="checkbox"/> Edit	Mitaelson_Nibl	rmlka	rmlka@mitaelson.rmlka	Marketing Manager	<input type="checkbox"/>	Platform 1
<input type="checkbox"/> Edit	Mitaelson_Niklaus	rmka	rkaua@mitaelson.rmka	Sales Manager	<input checked="" type="checkbox"/>	Platform 1
<input type="checkbox"/> Edit	Mitaelson_Shweta	sumka	shweta@mitaelson.sumka	Marketing Manager	<input type="checkbox"/>	Platform 1

At the bottom of the table are buttons for 'New User', 'Reset Password(s)', and 'Add Multiple Users'. There are also navigation links for letters A through Z and an 'Other' link.

2. Permission Sets

2.1 Platform 1 Permission Set

The screenshot shows the Salesforce Setup interface for a permission set named "Permission_Platform_1". The main details pane shows the API name, license, session activation requirement, and related permission set groups. The "Object Permissions" tab is selected, displaying a table of permissions for two objects: "HandsMen Customer" and "HandsMen Order". Both objects have "Read", "Create", "Edit", and "Delete" permissions enabled. "View All Records" and "Modify All Records" are disabled for both. "View All Fields" is enabled for both.

Label	Object API Name	Read	Create	Edit	Delete	View All Records	Modify All Records	View All Fields
HandsMen Customer	HandsMen_Customer__c	✓	✓	✓	✓	✗	✗	✗
HandsMen Order	HandsMen_Order__c	✓	✓	✓	✓	✗	✗	✗

2.2 Sales Manager Permission Set

The screenshot shows the Salesforce Setup interface for a permission set named "Sales_Manager_Permissions". The main details pane shows the API name, license, session activation requirement, and related permission set groups. The "Object Permissions" tab is selected, displaying a table of permissions for two objects: "HandsMen Customer" and "HandsMen Order". Both objects have "Read", "Create", "Edit", and "Delete" permissions enabled. "View All Records" and "Modify All Records" are enabled for both. "View All Fields" is enabled for both.

Label	Object API Name	Read	Create	Edit	Delete	View All Records	Modify All Records	View All Fields
HandsMen Customer	HandsMen_Customer__c	✓	✓	✓	✓	✓	✓	✓
HandsMen Order	HandsMen_Order__c	✓	✓	✓	✓	✓	✓	✓

2.3 Inventory Manager Permission Set

The screenshot shows the Salesforce Setup interface with the URL <https://orgfarm-b823a1b41d-dev-ed.lightning.force.com/lightning/setup/PermSets/0PSgL000004RoUf/summary>. The page title is "SETUP > PERMISSION SETS > INVENTORY MANAGER PERMISSIONS". The permission set details are as follows:

API Name	License	Created By	Last Modified By
Inventory_Manager_Permissions	--	Nikhil Varun	Nikhil Varun
Namespace Prefix	Session Activation Required	Created Date	Last Modified Date
--	Not Required	7/20/2025, 9:16 PM	7/20/2025, 9:17 PM
Related Permission Set Groups	Assigned Users		
0	1		
Description	Read & Edit on Inventory, Products		

Permission Set Information
See the permissions enabled for this permission set and the permission set groups it's added to.

Related Permission Set Groups User Permissions Object Permissions Field Permissions Custom Permissions Tabs

Label	Object API Name	Read	Create	Edit	Delete	View All Records	Modify All Records	View All Fields
HandsMen Product	HandsMen_Product__c	✓	✗	✓	✗	✗	✗	✗
Inventory	Inventory__c	✓	✗	✓	✗	✗	✗	✗

2.4 Marketing Team Permission Set

The screenshot shows the Salesforce Setup interface with the URL <https://orgfarm-b823a1b41d-dev-ed.lightning.force.com/lightning/setup/PermSets/0PSgL000004RoEl/summary>. The page title is "SETUP > PERMISSION SETS > MARKETING TEAM PERMISSIONS". The permission set details are as follows:

API Name	License	Created By	Last Modified By
Marketing_Team_Permissions	--	Nikhil Varun	Nikhil Varun
Namespace Prefix	Session Activation Required	Created Date	Last Modified Date
--	Not Required	7/20/2025, 9:20 PM	7/20/2025, 9:22 PM
Related Permission Set Groups	Assigned Users		
0	0		
Description	Read on Customers, Edit on Marketing Campaigns		

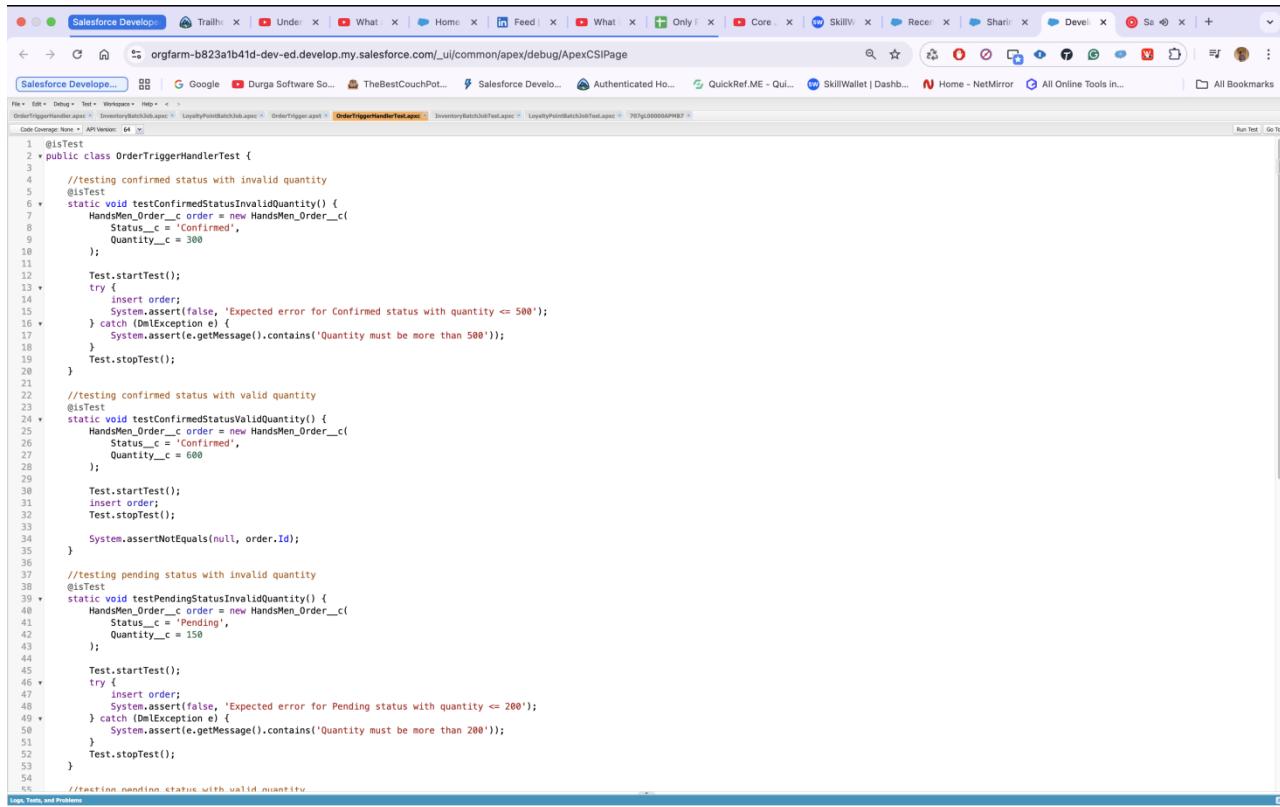
Permission Set Information
See the permissions enabled for this permission set and the permission set groups it's added to.

Related Permission Set Groups User Permissions Object Permissions Field Permissions Custom Permissions Tabs

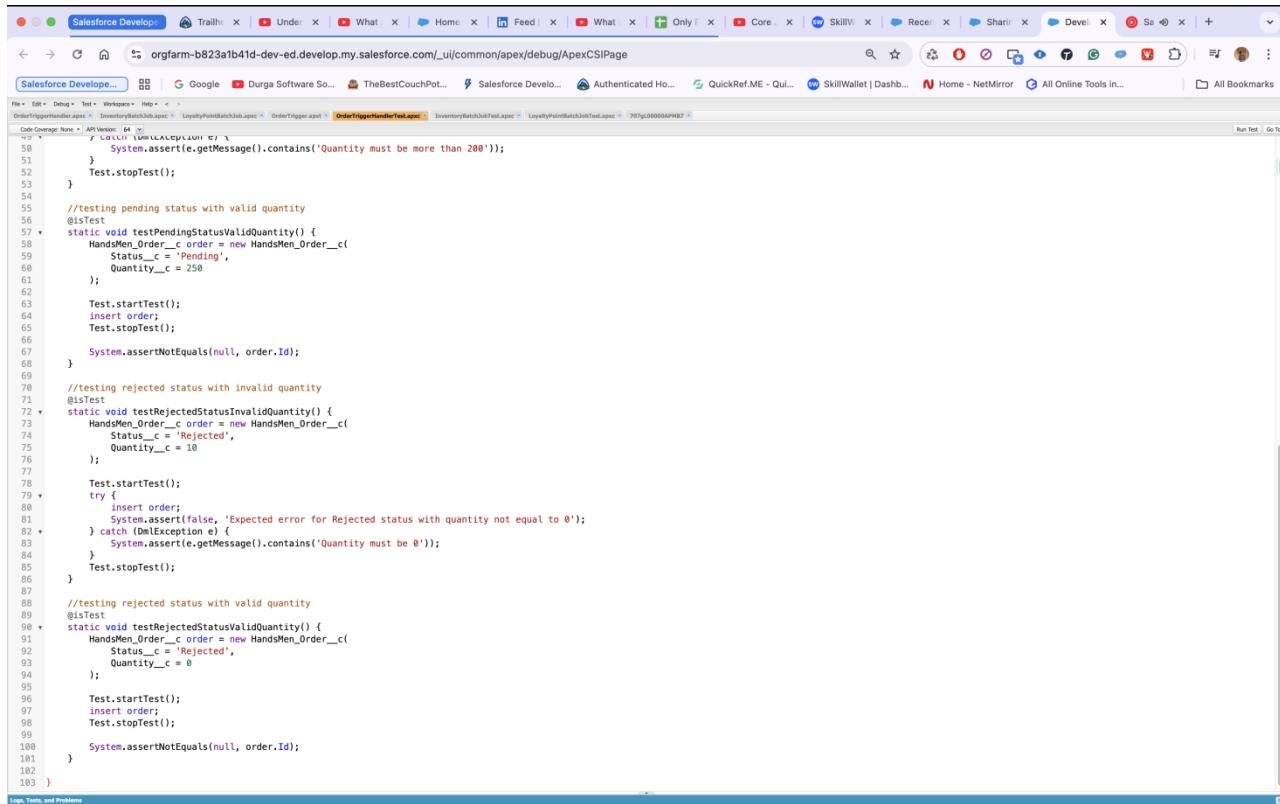
Label	Object API Name	Read	Create	Edit	Delete	View All Records	Modify All Records	View All Fields
HandsMen Customer	HandsMen_Customer__c	✓	✗	✗	✗	✗	✗	✗
Marketing Campaign	Marketing_Campaign__c	✓	✗	✓	✗	✗	✗	✗

Creation of Test Classes

1. OrderTrigger Test



```
1 @isTest
2 +public class OrderTriggerHandlerTest {
3
4     //testing confirmed status with invalid quantity
5     @isTest
6     static void testConfirmedStatusInvalidQuantity() {
7         HandsMen_Order__c order = new HandsMen_Order__c{
8             Status__c = 'Confirmed',
9             Quantity__c = 300
10        };
11
12        Test.startTest();
13        try {
14            insert order;
15            System.assert(false, 'Expected error for Confirmed status with quantity <= 500');
16        } catch (DmException e) {
17            System.assertEquals(e.getMessage().contains('Quantity must be more than 500'));
18        }
19        Test.stopTest();
20    }
21
22    //testing confirmed status with valid quantity
23    @isTest
24    static void testConfirmedStatusValidQuantity() {
25        HandsMen_Order__c order = new HandsMen_Order__c{
26            Status__c = 'Confirmed',
27            Quantity__c = 600
28        };
29
30        Test.startTest();
31        insert order;
32        Test.stopTest();
33
34        System.assertNotEquals(null, order.Id);
35    }
36
37    //testing pending status with invalid quantity
38    @isTest
39    static void testPendingStatusInvalidQuantity() {
40        HandsMen_Order__c order = new HandsMen_Order__c{
41            Status__c = 'Pending',
42            Quantity__c = 150
43        };
44
45        Test.startTest();
46        try {
47            insert order;
48            System.assert(false, 'Expected error for Pending status with quantity <= 200');
49        } catch (DmException e) {
50            System.assertEquals(e.getMessage().contains('Quantity must be more than 200'));
51        }
52        Test.stopTest();
53    }
54
55    //testing pending status with valid quantity
56    @isTest
57    static void testPendingStatusValidQuantity() {
58        HandsMen_Order__c order = new HandsMen_Order__c{
59            Status__c = 'Pending',
60            Quantity__c = 250
61        };
62
63        Test.startTest();
64        insert order;
65        Test.stopTest();
66
67        System.assertNotEquals(null, order.Id);
68    }
69
70    //testing rejected status with invalid quantity
71    @isTest
72    static void testRejectedStatusInvalidQuantity() {
73        HandsMen_Order__c order = new HandsMen_Order__c{
74            Status__c = 'Rejected',
75            Quantity__c = 10
76        };
77
78        Test.startTest();
79        try {
80            insert order;
81            System.assert(false, 'Expected error for Rejected status with quantity not equal to 0');
82        } catch (DmException e) {
83            System.assertEquals(e.getMessage().contains('Quantity must be 0'));
84        }
85        Test.stopTest();
86    }
87
88    //testing rejected status with valid quantity
89    @isTest
90    static void testRejectedStatusValidQuantity() {
91        HandsMen_Order__c order = new HandsMen_Order__c{
92            Status__c = 'Rejected',
93            Quantity__c = 0
94        };
95
96        Test.startTest();
97        insert order;
98        Test.stopTest();
99
100       System.assertNotEquals(null, order.Id);
101   }
102 }
```



```
1 @isTest
2 +public class OrderTriggerHandlerTest {
3
4     //testing confirmed status with invalid quantity
5     @isTest
6     static void testConfirmedStatusInvalidQuantity() {
7         HandsMen_Order__c order = new HandsMen_Order__c{
8             Status__c = 'Confirmed',
9             Quantity__c = 300
10        };
11
12        Test.startTest();
13        try {
14            insert order;
15            System.assert(false, 'Expected error for Confirmed status with quantity <= 500');
16        } catch (DmException e) {
17            System.assertEquals(e.getMessage().contains('Quantity must be more than 500'));
18        }
19        Test.stopTest();
20    }
21
22    //testing confirmed status with valid quantity
23    @isTest
24    static void testConfirmedStatusValidQuantity() {
25        HandsMen_Order__c order = new HandsMen_Order__c{
26            Status__c = 'Confirmed',
27            Quantity__c = 600
28        };
29
30        Test.startTest();
31        insert order;
32        Test.stopTest();
33
34        System.assertNotEquals(null, order.Id);
35    }
36
37    //testing pending status with invalid quantity
38    @isTest
39    static void testPendingStatusInvalidQuantity() {
40        HandsMen_Order__c order = new HandsMen_Order__c{
41            Status__c = 'Pending',
42            Quantity__c = 150
43        };
44
45        Test.startTest();
46        try {
47            insert order;
48            System.assert(false, 'Expected error for Pending status with quantity <= 200');
49        } catch (DmException e) {
50            System.assertEquals(e.getMessage().contains('Quantity must be more than 200'));
51        }
52        Test.stopTest();
53    }
54
55    //testing pending status with valid quantity
56    @isTest
57    static void testPendingStatusValidQuantity() {
58        HandsMen_Order__c order = new HandsMen_Order__c{
59            Status__c = 'Pending',
60            Quantity__c = 250
61        };
62
63        Test.startTest();
64        insert order;
65        Test.stopTest();
66
67        System.assertNotEquals(null, order.Id);
68    }
69
70    //testing rejected status with invalid quantity
71    @isTest
72    static void testRejectedStatusInvalidQuantity() {
73        HandsMen_Order__c order = new HandsMen_Order__c{
74            Status__c = 'Rejected',
75            Quantity__c = 10
76        };
77
78        Test.startTest();
79        try {
80            insert order;
81            System.assert(false, 'Expected error for Rejected status with quantity not equal to 0');
82        } catch (DmException e) {
83            System.assertEquals(e.getMessage().contains('Quantity must be 0'));
84        }
85        Test.stopTest();
86    }
87
88    //testing rejected status with valid quantity
89    @isTest
90    static void testRejectedStatusValidQuantity() {
91        HandsMen_Order__c order = new HandsMen_Order__c{
92            Status__c = 'Rejected',
93            Quantity__c = 0
94        };
95
96        Test.startTest();
97        insert order;
98        Test.stopTest();
99
100       System.assertNotEquals(null, order.Id);
101   }
102 }
```

2. Inventory Batch Job Test

The screenshot shows the Salesforce Developer Console with the code for `InventoryBatchJobTest.apc`. The code is a test class for an inventory batch job. It includes methods for setting up test data, executing the batch job, and querying updated products. The code uses Apex assertions to verify the correctness of the batch execution.

```
1 @isTest
2 public class InventoryBatchJobTest {
3
4     @TestSetup
5     static void setupTestData() {
6         List<HandsMen_Product__c> lowStockProducts = new List<HandsMen_Product__c>();
7
8         // Create 5 test products with stock below 10
9         for (Integer i = 0; i < 5; i++) {
10             lowStockProducts.add(new HandsMen_Product__c{
11                 Stock_Quantity__c = 5
12             });
13         }
14
15         insert lowStockProducts;
16
17         // Create 2 products that should NOT be picked up by the batch job
18         insert new HandsMen_Product__c(Stock_Quantity__c = 15);
19         insert new HandsMen_Product__c(Stock_Quantity__c = 20);
20     }
21
22     @isTest
23     static void testBatchExecution() {
24         Test.startTest();
25         InventoryBatchJob batch = new InventoryBatchJob();
26         Database.executeBatch(batch, 200);
27         Test.stopTest();
28
29         // Query updated products and assert
30         List<HandsMen_Product__c> updatedProducts = [
31             SELECT Stock_Quantity__c FROM HandsMen_Product__c WHERE Stock_Quantity__c >= 55
32         ];
33
34         System.assertEquals(5, updatedProducts.size(), 'Only 5 products should be updated');
35         for (HandsMen_Product__c p : updatedProducts) {
36             System.assertEquals(55, p.Stock_Quantity__c, 'Stock quantity should have been increased by 50');
37         }
38     }
39
40     @isTest
41     static void testScheduledExecution() {
42         String jobName = 'Test Inventory Batch Schedule';
43
44         Test.startTest();
45         String cronExp = '0 0 0 1 ? 2026'; // Any valid future date
46         System.schedule(jobName, cronExp, new InventoryBatchJob());
47         Test.stopTest();
48
49         // We can't assert execution result here,
50         // but we verify no errors during scheduling.
51         System.assert(true, 'Scheduled job should be registered successfully');
52     }
53 }
```

3. Loyalty Point Test

The screenshot shows the Salesforce Developer Console with the code for `LoyaltyPointBatchJobTest.apc`. The code is a test class for a loyalty point batch job. It sets up test data for customers and orders, executes the batch job, and asserts that the loyalty status was updated correctly for each customer. The code also verifies scheduled execution.

```
1 @isTest
2 public class LoyaltyPointBatchJobTest {
3
4     @TestSetup
5     static void setupTestData() {
6         List<HandsMen_Customer__c> customers = new List<HandsMen_Customer__c>();
7
8         for (Integer i = 1; i <= 3; i++) {
9             customers.add(new HandsMen_Customer__c());
10        }
11
12        insert customers;
13
14        List<HandsMen_Order__c> orders = new List<HandsMen_Order__c>();
15
16        // Customer 1: Total = 1100 (Gold)
17        orders.add(new HandsMen_Order__c(Quantity__c = 600, Customer__c = customers[0].Id));
18        orders.add(new HandsMen_Order__c(Quantity__c = 500, Customer__c = customers[0].Id));
19
20        // Customer 2: Total = 700 (Bronze)
21        orders.add(new HandsMen_Order__c(Quantity__c = 300, Customer__c = customers[1].Id));
22        orders.add(new HandsMen_Order__c(Quantity__c = 400, Customer__c = customers[1].Id));
23
24        // Customer 3: Total = 200 (Silver)
25        orders.add(new HandsMen_Order__c(Quantity__c = 200, Customer__c = customers[2].Id));
26
27        insert orders;
28    }
29
30    @isTest
31    static void testBatchExecution() {
32        Test.startTest();
33        LoyaltyPointBatchJob batch = new LoyaltyPointBatchJob();
34        Database.executeBatch(batch, 200);
35        Test.stopTest();
36
37        List<HandsMen_Customer__c> updatedCustomers = [SELECT Loyalty_Status__c FROM HandsMen_Customer__c];
38
39        System.assertEquals('Gold', updatedCustomers[0].Loyalty_Status__c, 'Customer 1 should be Gold');
40        System.assertEquals('Bronze', updatedCustomers[1].Loyalty_Status__c, 'Customer 2 should be Bronze');
41        System.assertEquals('Silver', updatedCustomers[2].Loyalty_Status__c, 'Customer 3 should be Silver');
42    }
43
44    @isTest
45    static void testScheduledExecution() {
46        Test.startTest();
47        String cronExp = '0 0 0 ? * 1'; // Every Sunday at 12 AM
48        System.schedule('Weekly Loyalty Update', cronExp, new LoyaltyPointBatchJob());
49        Test.stopTest();
50
51        System.assert(true, 'Scheduled batch executed without error');
52    }
53 }
```

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

The screenshot shows the Salesforce Developer Console interface. At the top, there are tabs for various Apex classes: OrderTriggerHandler.apxc, InventoryBatchJob.apxc, LoyaltyPointBatchJob.apxc, OrderTrigger.apxt, OrderTriggerHandlerTest.apxc, InventoryBatchJobTest.apxc, LoyaltyPointBatchJobTest.apxc, and 707gL00000APMB7. Below the tabs, there is a table with columns: Class, Method, Duration, Result, Errors, and Stack Trace. The table lists several test methods for each class, all of which have passed (Result: Pass). On the right side of the interface, there is a vertical sidebar titled "Class Code Coverage" which displays the overall code coverage statistics.

Class	Method	Duration	Result	Errors	Stack Trace
InventoryBatchJobTest	testBatchExecution	0:00	Pass		
InventoryBatchJobTest	testScheduledExecution	0:01	Pass		
LoyaltyPointBatchJobTest	testBatchExecution	0:01	Pass		
LoyaltyPointBatchJobTest	testScheduledExecution	0:01	Pass		
OrderTriggerHandlerTest	testConfirmedStatusInval...	0:00	Pass		
OrderTriggerHandlerTest	testConfirmedStatusValidQ...	0:00	Pass		
OrderTriggerHandlerTest	testPendingStatusInvalidQ...	0:00	Pass		
OrderTriggerHandlerTest	testPendingStatusValidQu...	0:01	Pass		
OrderTriggerHandlerTest	testRejectedStatusInvalid...	0:00	Pass		
OrderTriggerHandlerTest	testRejectedStatusValidQu...	0:00	Pass		

Logs Tests Checkpoints Query Editor View State Progress Problems

Status	Test Run	Enqueued Time	Duration	Failures	Total	Overall Code Coverage																		
Test Run	707gL00000APMB7	Tue Jul 29 2025 11:02:19 GMT...		0	6	<table border="1"><thead><tr><th>Class</th><th>Percent</th><th>Lines</th></tr></thead><tbody><tr><td>Overall</td><td>96%</td><td></td></tr><tr><td>InventoryBatchJob</td><td>93%</td><td>14/15</td></tr><tr><td>LoyaltyPointBatchJob</td><td>95%</td><td>21/22</td></tr><tr><td>OrderTrigger</td><td>100%</td><td>2/2</td></tr><tr><td>OrderTriggerHandler</td><td>100%</td><td>11/11</td></tr></tbody></table>	Class	Percent	Lines	Overall	96%		InventoryBatchJob	93%	14/15	LoyaltyPointBatchJob	95%	21/22	OrderTrigger	100%	2/2	OrderTriggerHandler	100%	11/11
Class	Percent	Lines																						
Overall	96%																							
InventoryBatchJob	93%	14/15																						
LoyaltyPointBatchJob	95%	21/22																						
OrderTrigger	100%	2/2																						
OrderTriggerHandler	100%	11/11																						
	TestRun @ 10:49:36 am			0	2																			
	707gL00000APMB7			0	2																			
	InventoryBatchJobTest			0	2																			
	testBatchExecution		0:00																					
	testScheduledExecution		0:01																					
	LoyaltyPointBatchJobTest			0	2																			
	testBatchExecution		0:01																					
	testScheduledExecution		0:01																					
	OrderTriggerHandlerTest			0	6																			
	testConfirmedStatusInvalidQuantity		0:00																					
	testConfirmedStatusValidQuantity		0:00																					
	testPendingStatusInvalidQuantity		0:00																					
	testPendingStatusValidQuantity		0:01																					
	testRejectedStatusInvalidQuantity		0:00																					
	testRejectedStatusValidQuantity		0:00																					

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.