HandsMen Threads: Elevating the Art of Sophistication in Men's Fashion

PROJECT OVERVIEW

This project implements a customized Salesforce CRM solution for HandsMen Threads to enhance customer engagement, streamline operations, and improve data management. The system features a tailored data model designed to organize business information efficiently while maintaining high data integrity through UI-driven validation and automated processes. Key workflows include automated order confirmations, real-time loyalty status updates based on purchase history, stock level alerts to support warehouse replenishment, and scheduled batch processing for bulk order and inventory updates. By utilizing Salesforce tools such as Lightning App Builder, Record-Triggered Flows, Apex Triggers, and Asynchronous Apex, the project delivers an integrated and scalable solution that strengthens customer relations, supports decision-making, and optimizes overall business performance.

OBJECTIVES

The primary objective of this project was to develop and implement a customized Salesforce CRM solution for HandsMen Threads to streamline business operations, maintain accuracy and consistency of the data, and enhance customer experience and satisfaction.

By establishing a centralized data structure through well-designed data model that stores customer, product, and order information in an organized and accessible format, the project aims to:

1. Ensure data accuracy and reliability by implementing validation rules and proper data controls to maintain clean, accurate, and consistent records across the business organization.
2. Automate key business processes by using flows, Apex triggers, and scheduled jobs to handle order confirmations, stock alerts, loyalty updates, and bulk order processing, reduce manual work and minimizing human error.
3. Enhance customer engagement and experience by delivering timely notifications and personalized updates, fostering trust and strengthening customer relationships.
4. Optimize warehouse coordination, inventory management, and financial updates to support smoother daily operations and prepare the business for future growth.

PHASE 1: REQUIREMENT ANALYSIS & PLANNING

Understanding Business Requirements

An analysis of HandsMen Threads' operational challenges was conducted, focusing on customer management, order processing, inventory tracking, loyalty monitoring, and internal communication gaps. The CRM system aimed to centralize customer data, automate order workflows, optimize stock management, and improve customer engagement through timely notifications and personalized updates.

Project Scope and Objectives

- Create a centralized Salesforce CRM for customers, order, product, inventory, and marketing records.
- Automate key process such as order confirmation, loyalty status updates, and stock alerts.
- Allow staff to track real-time product availability and customer purchase history.
- Improve communication through automated email templates and notifications.
- Provide dashboards for sales trends and inventory performance.

Data Model

The figure below showed the Entity–Relationship Diagram (ERD) illustrated how key business records interact within the HandsMen Threads Salesforce CRM. HandsMen Customers were linked to multiple orders, while each order referenced a specific product. Products also connected to inventory records for stock tracking across locations. Additionally, customers associated with multiple marketing campaigns. This structure ensured organized, scalable, and efficient data management across the system.

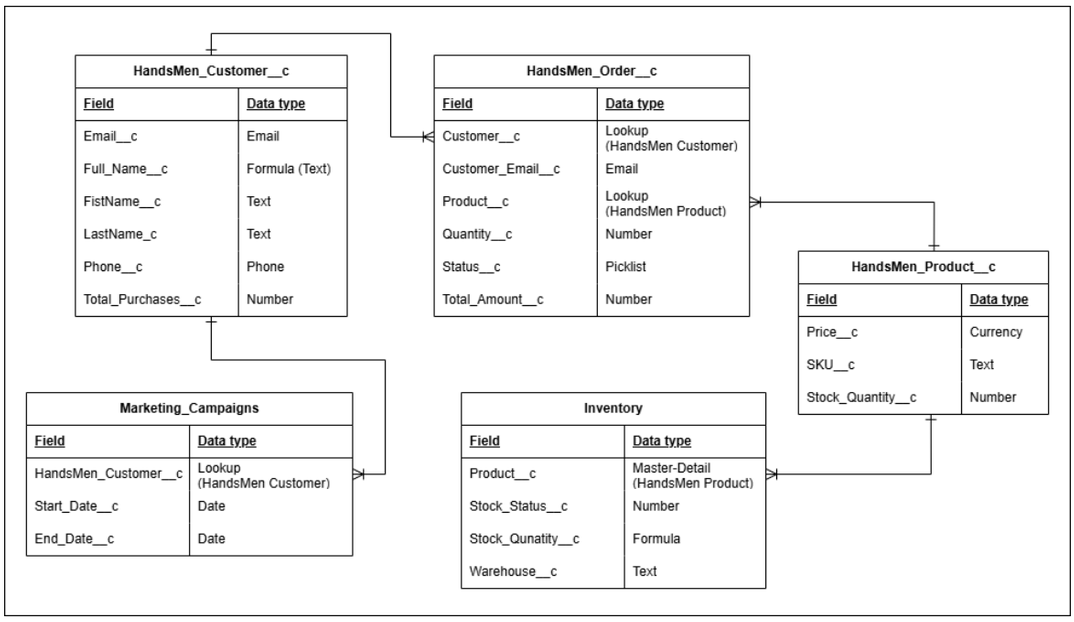


Fig. 1. Entity Relationship Diagram of HandsMen Threads

Security Model

- **Platform 1:** The profile was named Platform 1, and its user license was Salesforce, with full access to HandsMen Product and Inventory Objects. It controlled what users could see and do within these objects.

Table 1. Data Security Model

Role	**Access Level**
Sales Manager	Full access to customers and orders
Marketing Team	Read on customers, edit on marketing campaigns
Inventory Manager	Read and edit access on inventory and products

The table showed the data security model of user roles and access levels within HandsMen Threads Salesforce CRM. Sales Managers had full access to customers and orders, Marketing Team could view customers and manage marketing campaigns, and the Inventory Managers could read and edit inventory and products.

Stakeholders Mapping

- The CEO has top-level access to all data.
- Sales manager role reported to CEO and managed customers and orders.
- Marketing team role reported to CEO and handled campaigns and customer information.

- Inventory manager role also reported to CEO and was responsible for tracking and updating stocks and products.

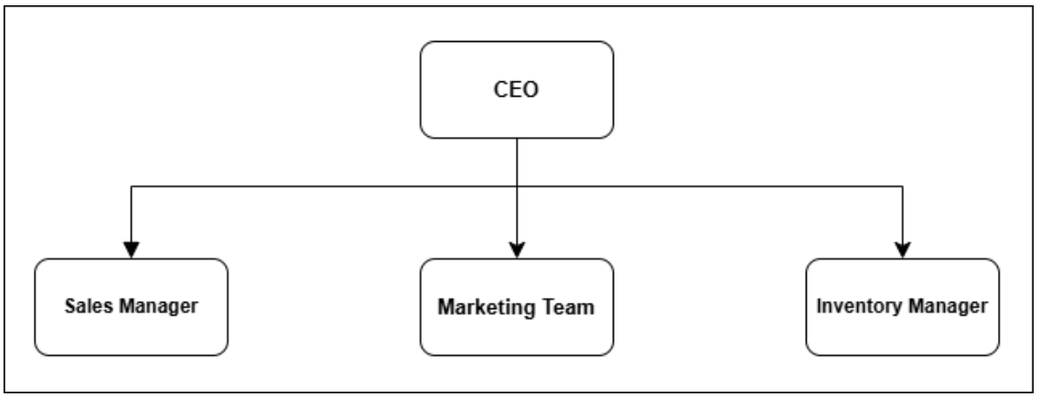
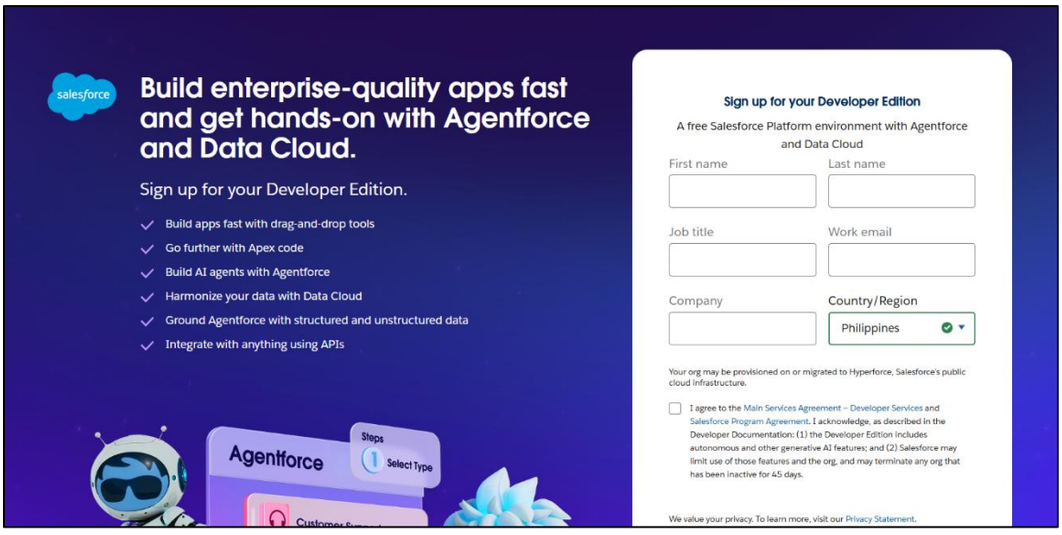


Fig. 2. Role Hierarchy of HandsMen Threads

PHASE 2: SALESFORCE DEVELOPMENT – BACKEND & CONFIGURATIONS

Environment Setup & DevOps Workflow

- Developer Org account was created using https://developer.salesforce.com/signup. The account was created and verified. Password set and access was granted to the Salesforce Setup page.



Customization of Objects, Fields, Validation Rules, Automation

The following **custom objects** were created:

1. **HandsMen Customer** – Stored customer details such as email, phone, loyalty status, name
2. **HandsMen Order** – Stored customer orders such as product, status, total amount.
3. **HandsMen Product** – Stored product catalogue details such as SKU, price, quantity.
4. **Inventory** – Tracked stock level and warehouse location.
5. **Marketing Campaign** – Managed promotions & campaigns.

Table 2. Custom Objects and their Key Fields

Object Name	API Name	Key Fields
HandsMen Customer	HandsMen_Customer__c	Email__c, FirstName__c, Full_Name__c, LastName__c, Loyalty_Status__c, Phone__c, Total_Purchases__c
HandsMen Order	HandsMen_Order__c	Customer__c, Customer_Email__c, Product__c, Quantity__c, Status__c, Total_Amount__c
HandsMen Product	HandsMen_Product__c	Order__c, Price__c. SKU__c, Stock_Quantity__c
Inventory	Inventory__c	Product__c, Stock_Quantity__c, Stock_Status__c, Warehouse__c
Marketing Campaign	Marketing_Campaign__c	Start_Date__c, End_Date__c, HandsMen_Customer__c

Table 3. Validation Rules

Object	Field	Validation Rule
HandsMen_Order__c	Total_Amount__c	Total_Amount__c ≤ 0
Inventory__c	Stock_Quantity__c	Stock_Quantity__c ≤ 0
HandsMen_Customer__c	Email__c	NOT CONTAINS (Email__c, “@gmail.com”)

Table 3 showed validation rules applied to objects to prevent incorrect data entry. For HandsMen_Order__c, orders could not have a total amount less than or equal to zero, ensuring valid financial records. Inventory__c prevents stock quantities from being zero or negative, maintaining accurate inventory levels, while HandsMen_Customer__c enforces that email addresses contain a proper domain (e.g., “@gmail.com”), ensuring reliable customer contact information.

Automation (Flows)

Order Confirmation Flow

- Triggered when Order__c.Status__c = Confirmed.
- Sent a confirmation email to the customer

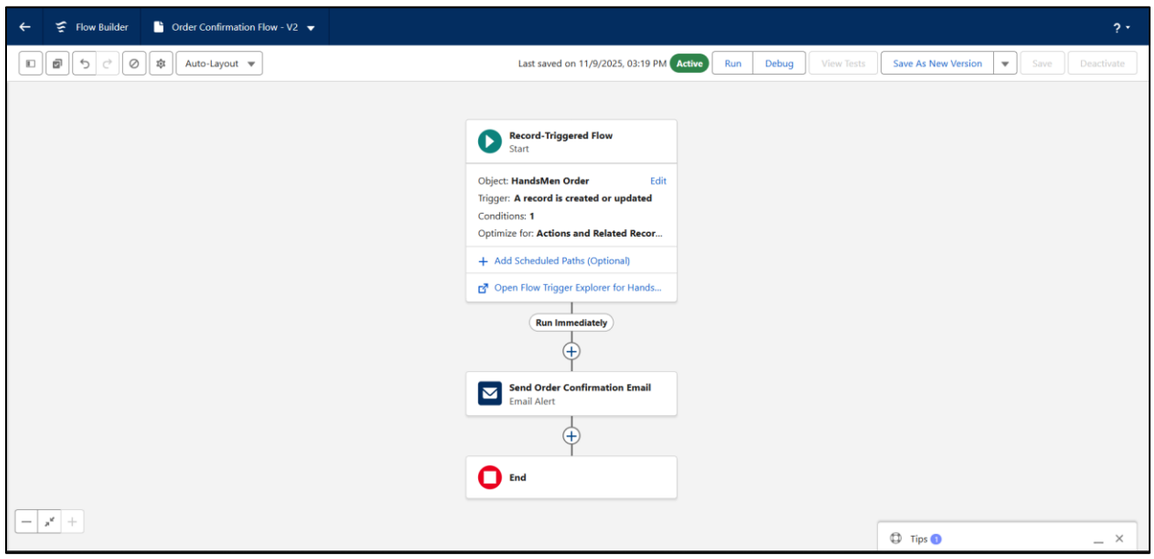


Fig. 3. Order Confirmation Flow

Stock Alert Flow

- Triggered when Inventory__c.Stock_Quantity__c < 5
- Sent an alert email to the sales team when the stock goes beyond 5 units.

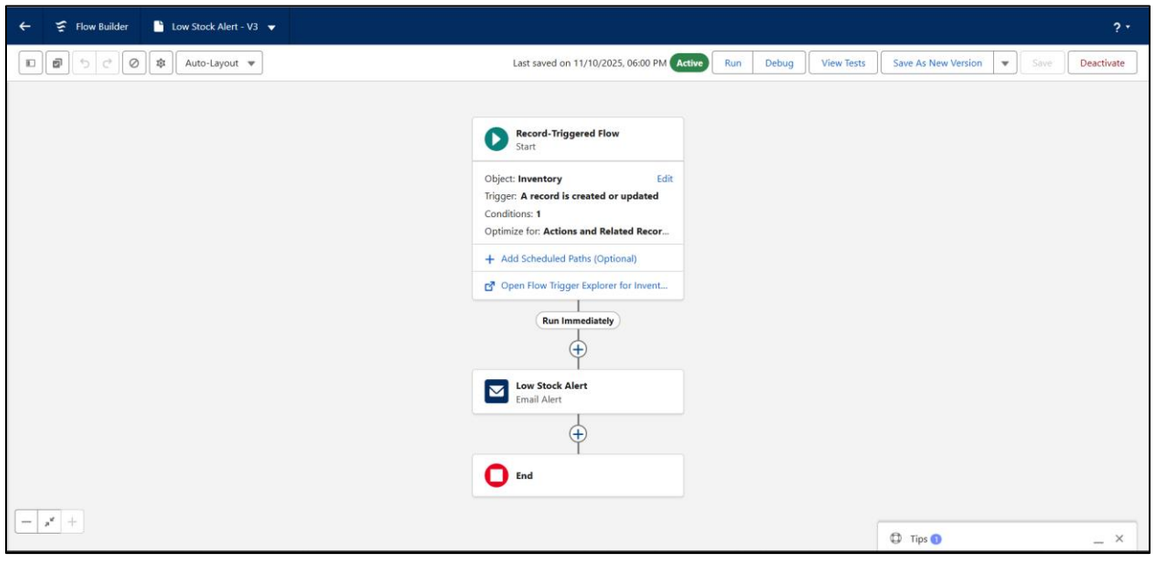


Fig. 4. Stock Alert Flow

Scheduled Flow: Loyalty Status Update

- Ran daily to update Loyalty_Status__c
- Sent an email to the customer when they qualify to the loyalty rewards.

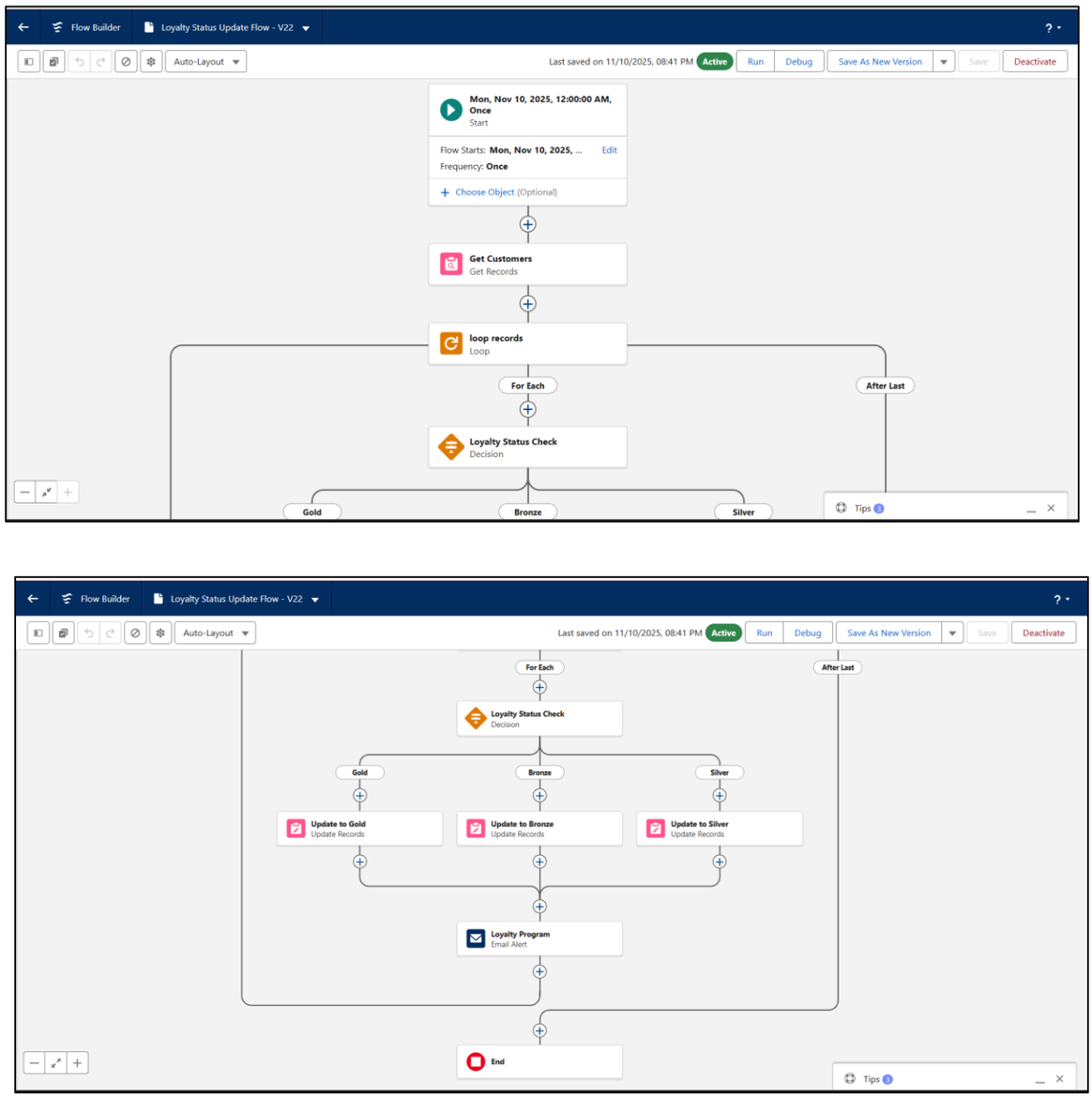


Fig. 5. Loyalty Status Update Scheduled Flow

Email Templates & Email Alerts

The following email templates were created:

Table 4. Email Templates

Template Name	Type	Trigger
Order Confirmation	HTML	Sent when order is placed
Low Stock Alert	Text	Sent when Inventory__c.Stock_Quntity__c < 5
Loyalty Program Email	HTML	Sent when customer qualifies for loyalty rewards

Corresponding Email Alerts were created using these templates and linked to automation flows.

Fig. 6. HandsMena Threads Letterhead

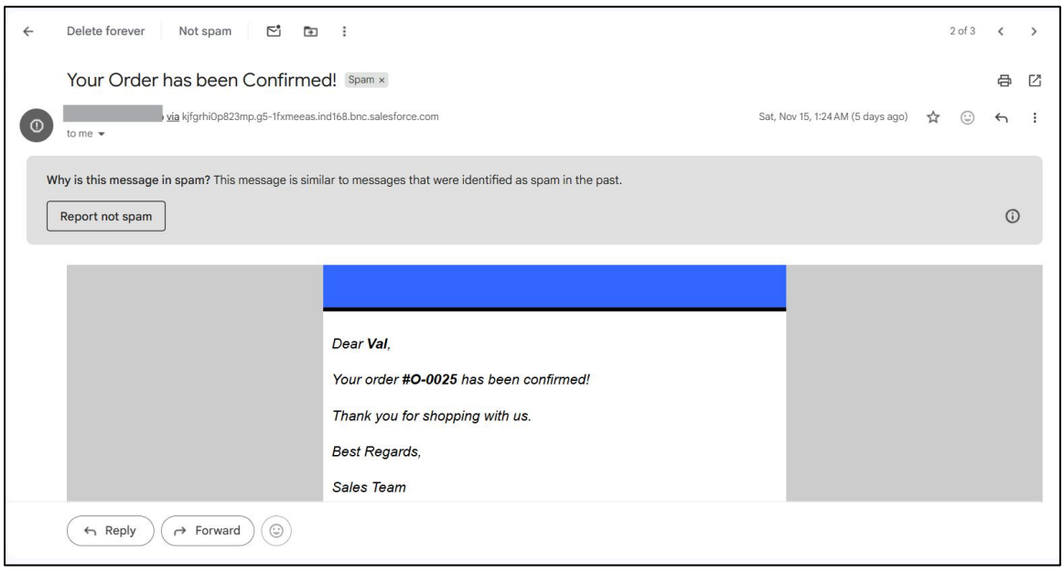


Fig. 7. Order Confirmation Email

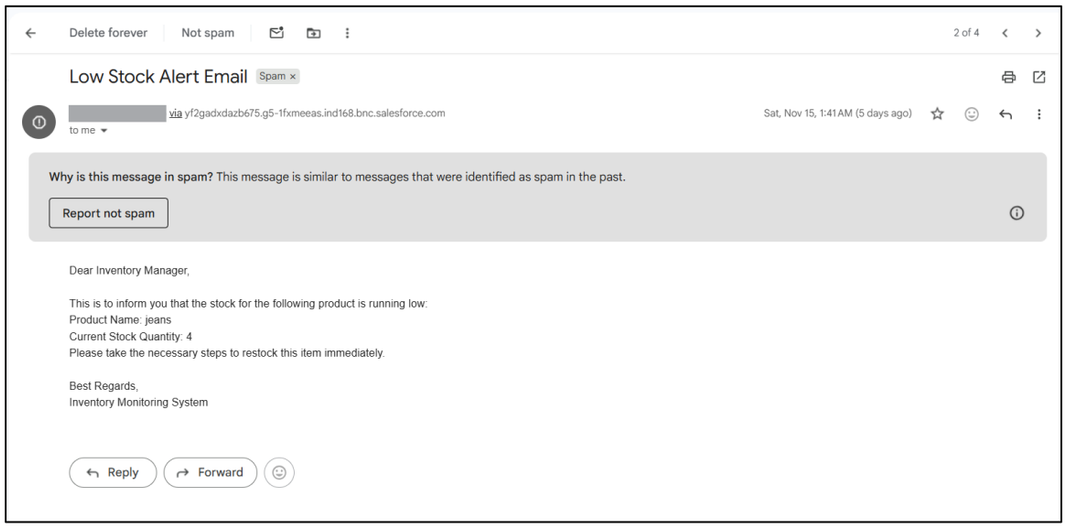


Fig. 8. Low Stock Email Alert

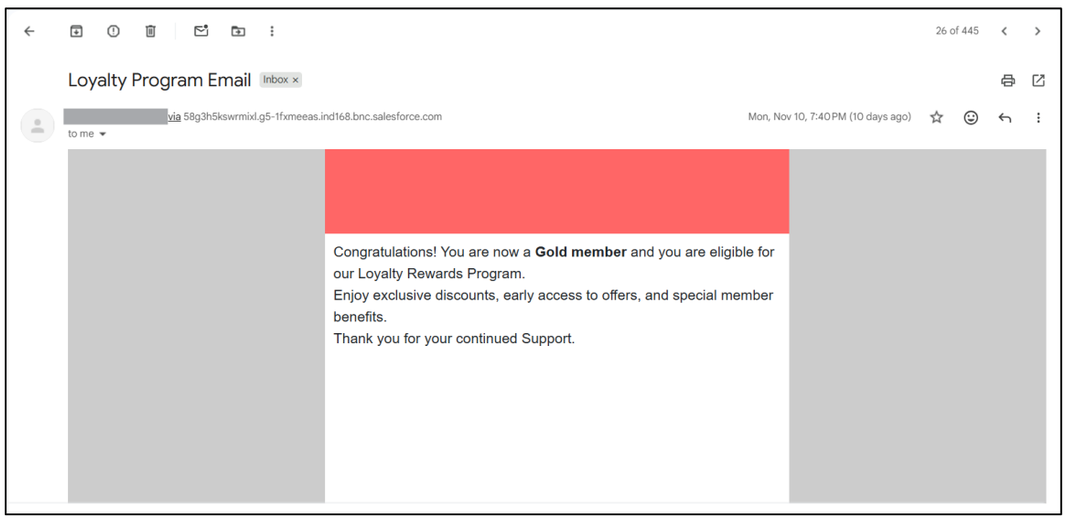


Fig. 9. Loyalty Program Email

Apex Classes & Triggers

- **Update Order Total** – trigger automatically updated the Total_Amount__c whenever an order record was saved. It ensured that the total amount reflected the correct calculation.

```
trigger OrderTotalTrigger on HandsMen_Order__c (before insert, before update) {
    Set<Id> productIds = new Set<Id>();

    for (HandsMen_Order__c order : Trigger.new) {
        if (order.Product__c != null) {
            productIds.add(order.Product__c);
        }
    }

    Map<Id, HandsMen_Product__c> productMap = new Map<Id, HandsMen_Product__c>(
        [SELECT Id, Price__c FROM HandsMen_Product__c WHERE Id IN :productIds]
    );

    for (HandsMen_Order__c order : Trigger.new) {
        if (order.Product__c != null && productMap.containsKey(order.Product__c)) {
            HandsMen_Product__c product = productMap.get(order.Product__c);
            if (order.Quantity__c != null) {
                order.Total_Amount__c = order.Quantity__c * product.Price__c;
            }
        }
    }
}
```

- **Stock Deduction** – reduced the stock quantity in the Inventory__c object when an order is confirmed.

```
trigger StockDeductionTrigger on HandsMen_Order__c (after insert, after update) {
    Set<Id> productIds = new Set<Id>();

    for (HandsMen_Order__c order : Trigger.new) {
        if (order.Status__c == 'Confirmed' && order.Product__c != null) {
            productIds.add(order.Product__c);
        }
    }

    if (productIds.isEmpty()) return;

    Map<Id, Inventory__c> inventoryMap = new Map<Id, Inventory__c>(
        [SELECT Id, Stock_Quantity__c, Product__c
         FROM Inventory__c
         WHERE Product__c IN :productIds]
    );

    List<Inventory__c> inventoriesToUpdate = new List<Inventory__c>();

    for (HandsMen_Order__c order : Trigger.new) {
        if (order.Status__c == 'Confirmed' && order.Product__c != null) {
            for (Inventory__c inv : inventoryMap.values()) {
                if (inv.Product__c == order.Product__c) {
                    inv.Stock_Quantity__c -= order.Quantity__c;
                    inventoriesToUpdate.add(inv);
                    break;
                }
            }
        }
    }

    if (!inventoriesToUpdate.isEmpty()) {
        update inventoriesToUpdate;
    }
}
```

- **Loyalty Status Update** – updated the customer's loyalty status based on their total purchases. Once the customer met the required threshold, the system automatically upgraded their loyalty status.

```
trigger LoyaltyStatusTrigger
on HandsMen_Customer__c (before insert, before update) {

    for (HandsMen_Customer__c customer : Trigger.new) {

        Boolean shouldUpdate = true;

        if (Trigger.isUpdate) {
            HandsMen_Customer__c oldCustomer = Trigger.oldMap.get(customer.Id);

            shouldUpdate = oldCustomer.Total_Purchases__c != customer.Total_Purchases__c;
        }

        if (shouldUpdate && customer.Total_Purchases__c != null) {
            if (customer.Total_Purchases__c > 1000) {
                customer.Loyalty_Status__c = 'Gold';
            } else if (customer.Total_Purchases__c <= 500) {
                customer.Loyalty_Status__c = 'Bronze';
            } else {
                customer.Loyalty_Status__c = 'Silver';
            }
        }
    }
}
```

Batch Jobs Processes

- **Loyalty Points Calculation** – calculated and updated customer loyalty points on weekly schedule, ran every Sunday at 12 AM.

```
global class LoyaltyPointsBatch implements Database.Batchable<SObject>, Schedulable {
global Database.QueryLocator start(Database.BatchableContext BC) {
return Database.getQueryLocator(
'SELECT Id, Total_Purchases__c, Loyalty_Status__c FROM HandsMen_Customer__c'
);
}
    global void execute(Database.BatchableContext BC, List<SObject> records) {
        List<HandsMen_Customer__c> customersToUpdate = new List<HandsMen_Customer__c>();

        for (SObject s : records) {
            HandsMen_Customer__c c = (HandsMen_Customer__c)s;

            if (c.Total_Purchases__c > 1000) {
                c.Loyalty_Status__c = 'Gold';
            } else if (c.Total_Purchases__c <= 500) {
                c.Loyalty_Status__c = 'Bronze';
            } else {
                c.Loyalty_Status__c = 'Silver'; // default
            }

            customersToUpdate.add(c);
        }

        if (!customersToUpdate.isEmpty()) {
            try {
                update customersToUpdate;
            } catch (DmlException e) {
                System.debug('Error updating loyalty status: ' + e.getMessage());
            }
        }
    }

    global void finish(Database.BatchableContext BC) {
        System.debug('Loyalty Status Update Completed');
    }

    global void execute(SchedulableContext SC) {
        LoyaltyPointsBatch LoyaltyBatchJob = new LoyaltyPointsBatch();
        Database.executeBatch(LoyaltyBatchJob, 200);
    }
}
```

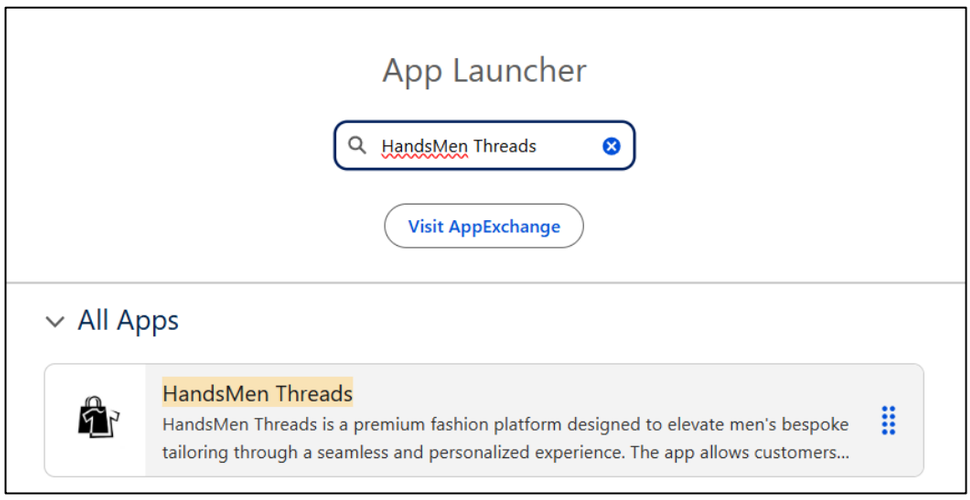
- **Inventory Sync** – synchronized stock levels between Salesforce and the external warehouse system and ran daily at 2 AM and updated inventory records to reflect the most recent stock data.

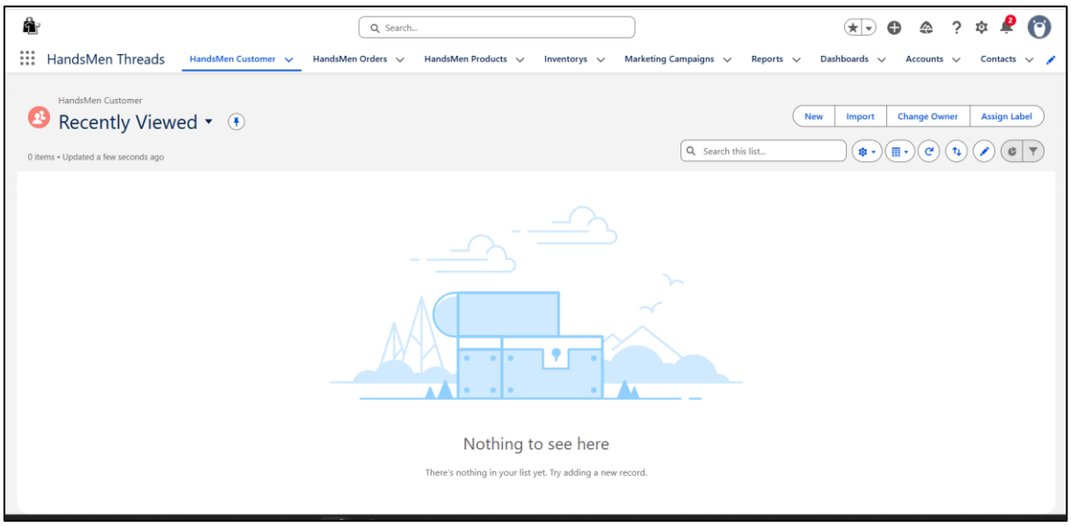
```
global class InventoryBatchJob implements Database.Batchable<SObject>, Schedulable {
global Database.QueryLocator start(Database.BatchableContext BC) {

    return Database.getQueryLocator(
'SELECT Id, Stock_Quantity__c FROM Product__c WHERE Stock_Quantity__c < 10'
);
}
    global void execute(Database.BatchableContext BC, List<SObject> records) {
        List<HandsMen_Product__c> productsToUpdate = new List<HandsMen_Product__c>();

        // Cast SObject list to Product__c list
        for (SObject record : records) {

            HandsMen_Product__c product = (HandsMen_Product__c) record;
            product.Stock_Quantity__c += 50; // Restock logic
            productsToUpdate.add(product);
            }

        if (!productsToUpdate.isEmpty()) {
        try {
            update productsToUpdate;
            } 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);
    }
}
```

PHASE 3: UI/UX DEVELOPMENT & CUSTOMIZATION

Lightning App Setup.

The Lightning App was configured through the App Manager to provide users with a unified interface for accessing essential objects and tools.





Page Layouts & Dynamic Forms.

Page layouts and dynamic forms were designed to improve data entry, display, and overall user interaction in every objects.

- **HandsMen Customer**

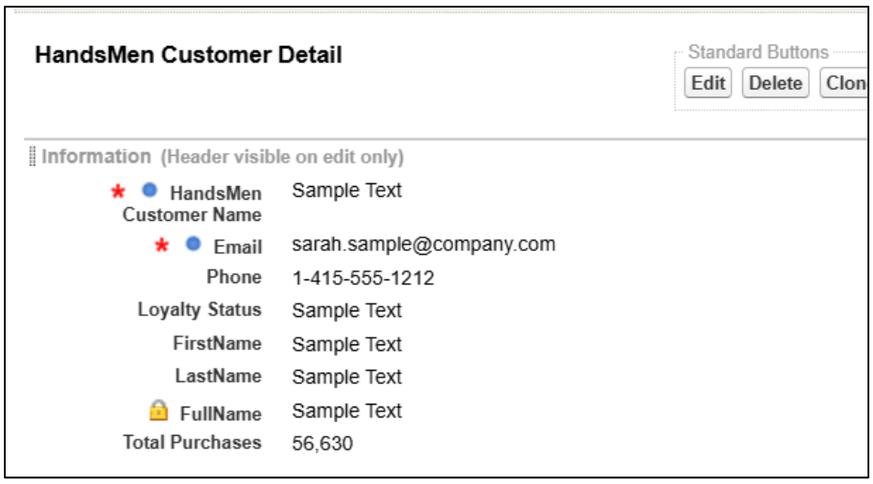


Fig. 10. HeadsMen Customer Detail Page Layout

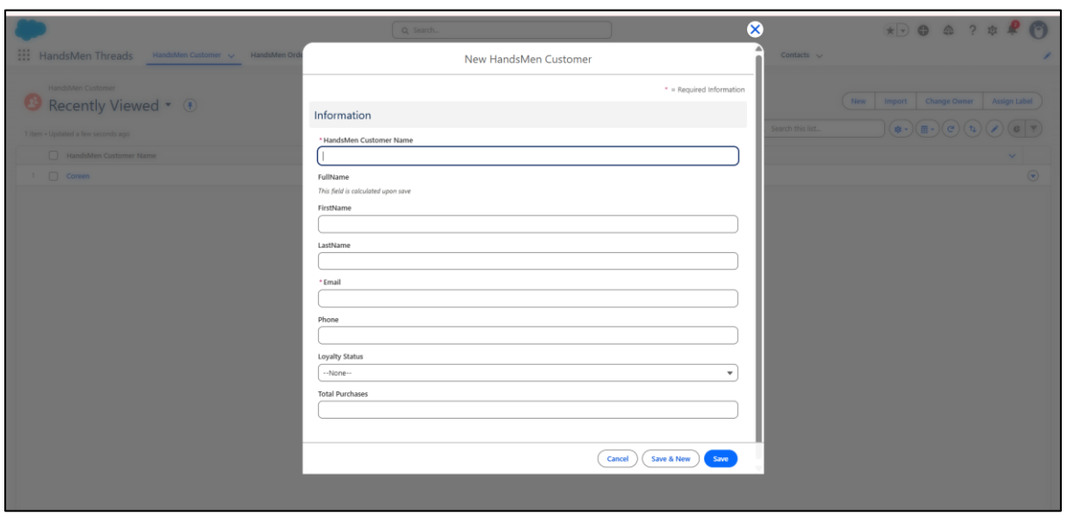


Fig. 11. Form when you add new customer

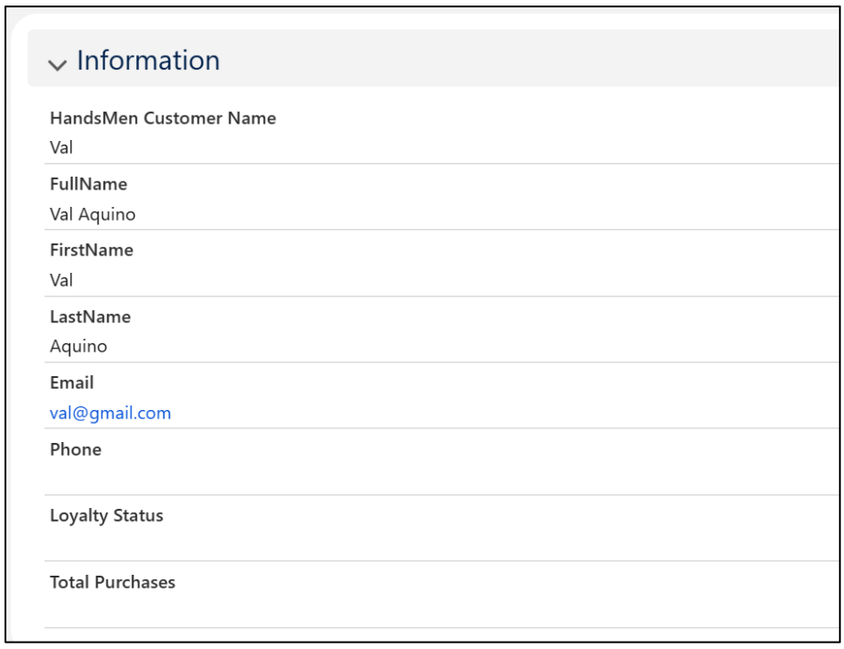


Fig. 12. Example forms of customer

- **HandsMen Order**

Fig. 13. HandsMen Order Detail Page Layout

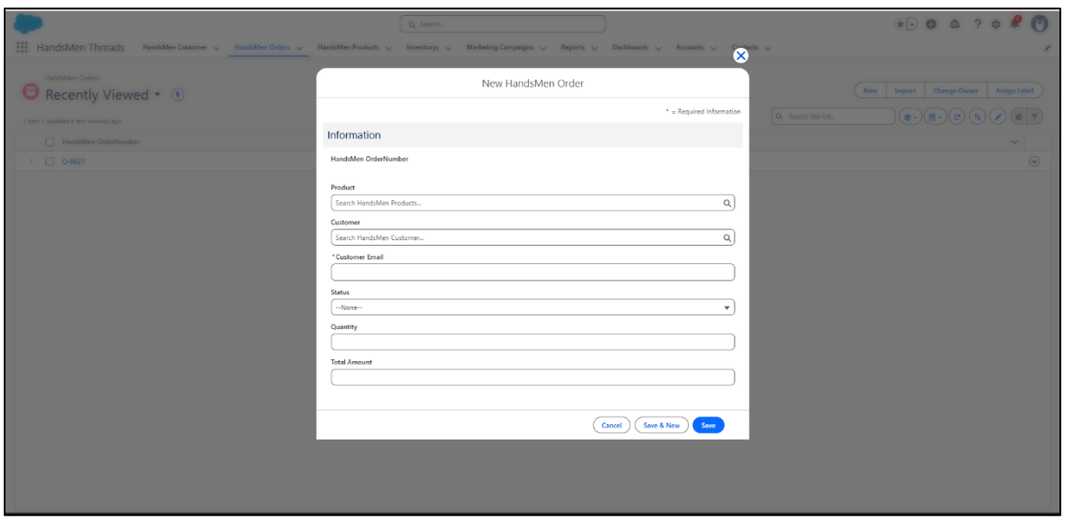


Fig. 14. Form when you add order

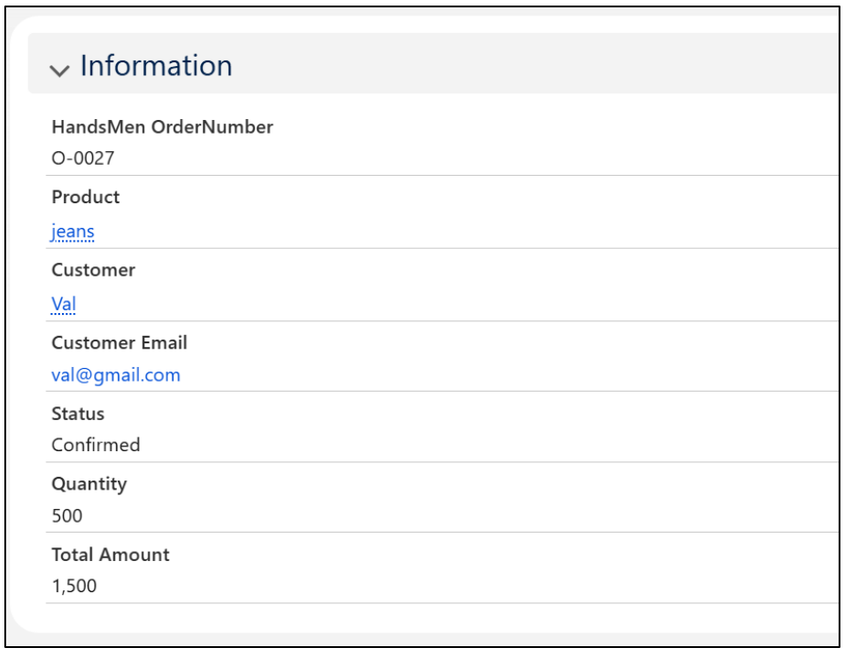


Fig. 15. Example of added order

- **HandsMen Product**

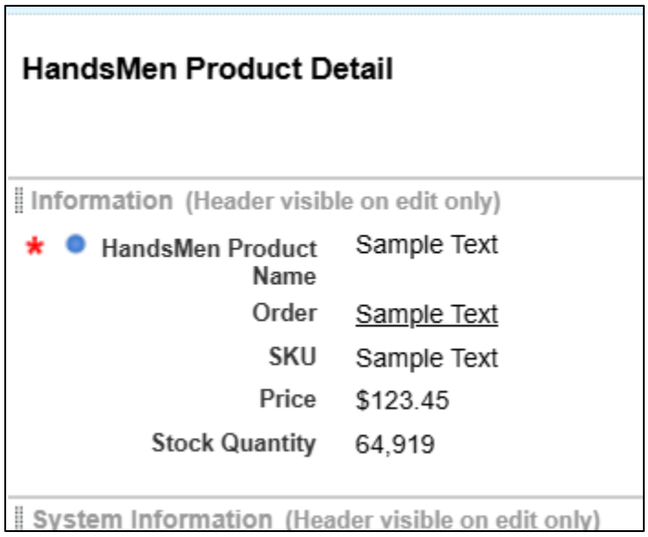


Fig. 16. HandsMen Product Detail Page Layout

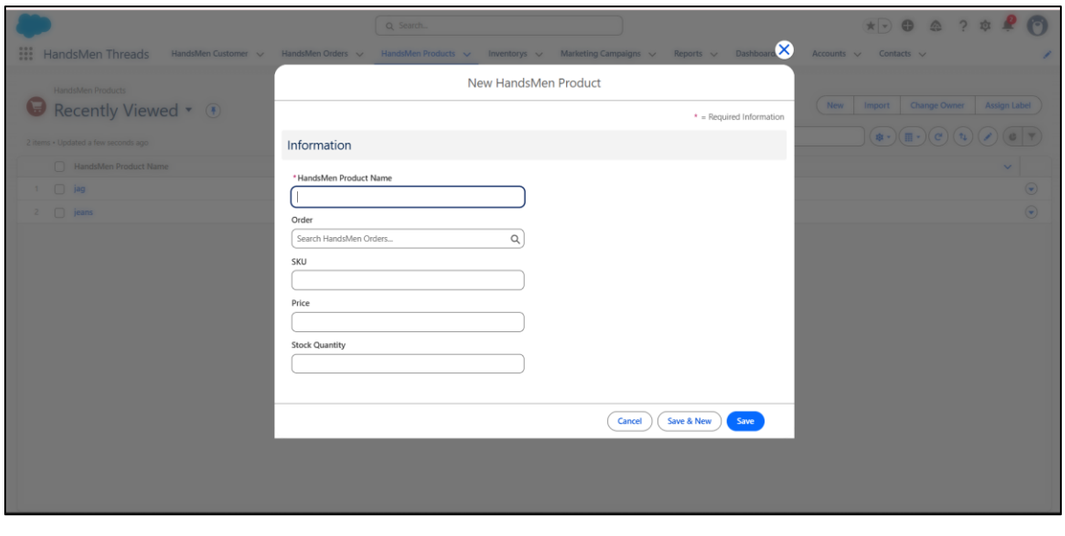


Fig. 17. Form when you add new product

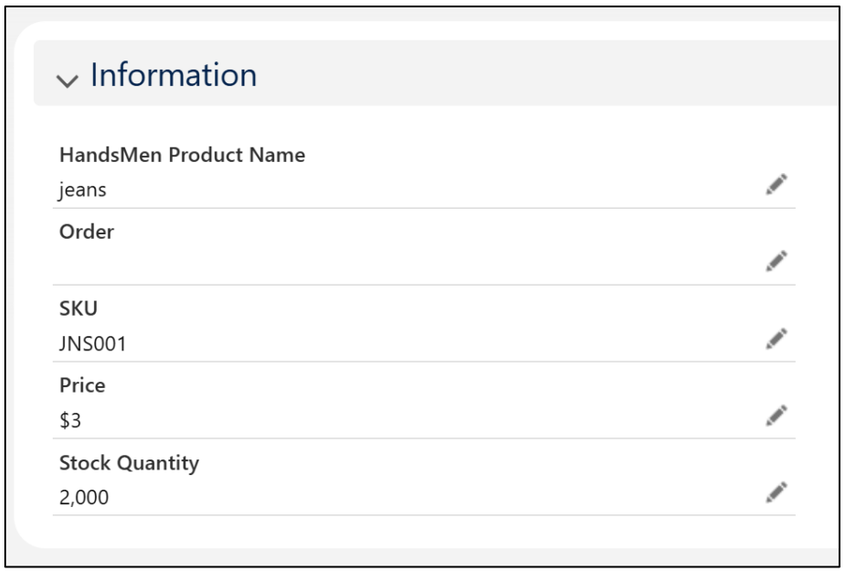


Fig. 18. Example added product

- **Inventory**

Fig. 19. Inventory Detail Page Layout

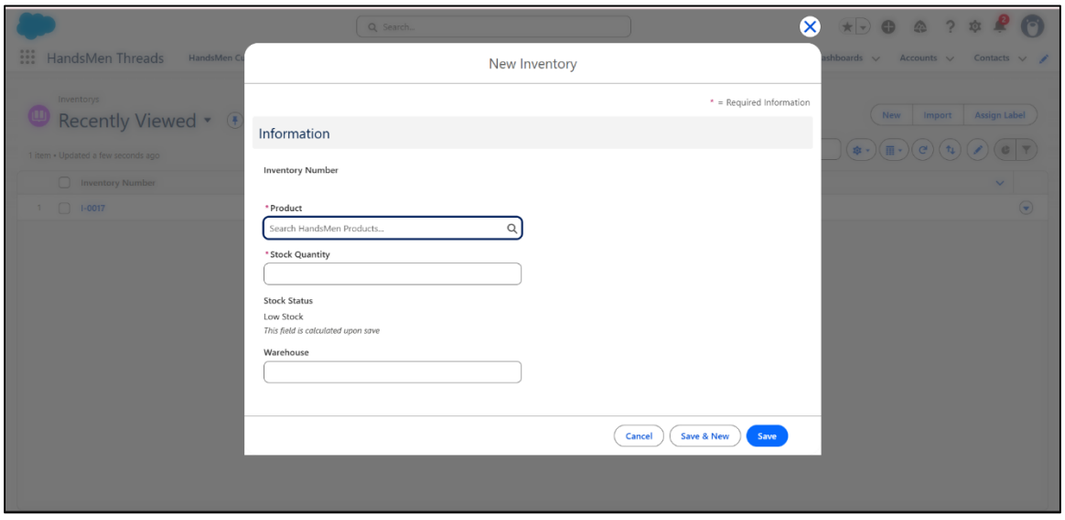


Fig. 20. Form when you add new inventory

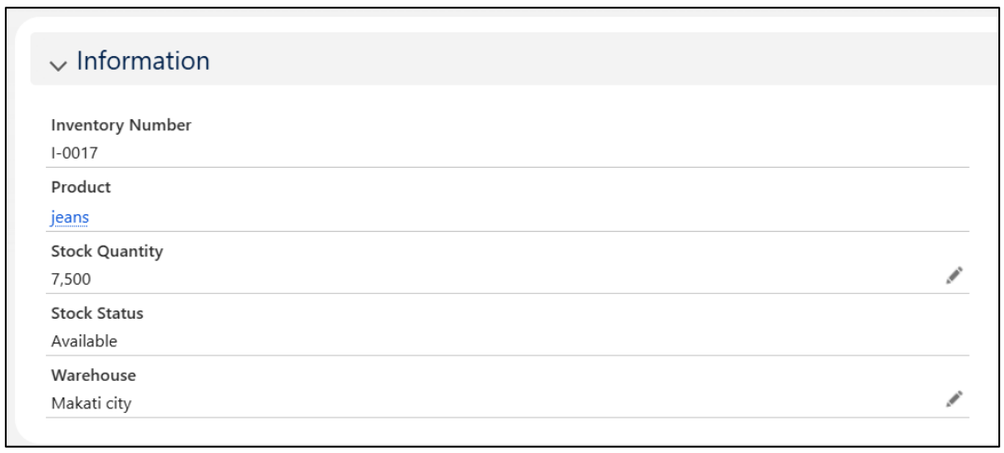


Fig. 21. Example of an inventory

- **Marketing Management**

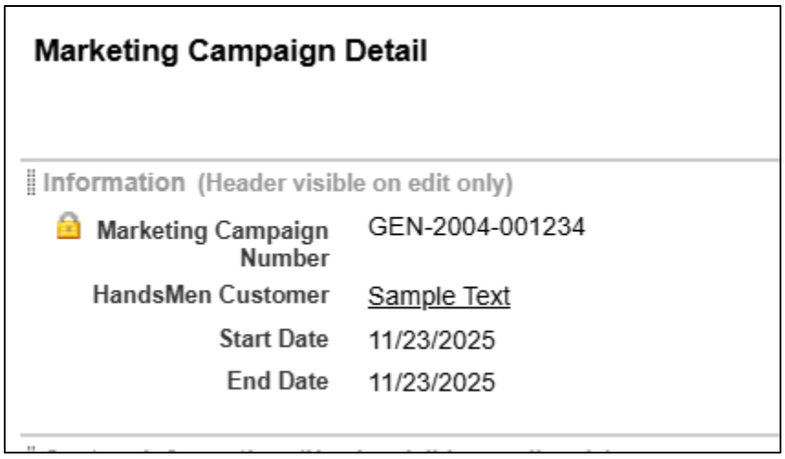


Fig. 22. Marketing Campaign Page Layout

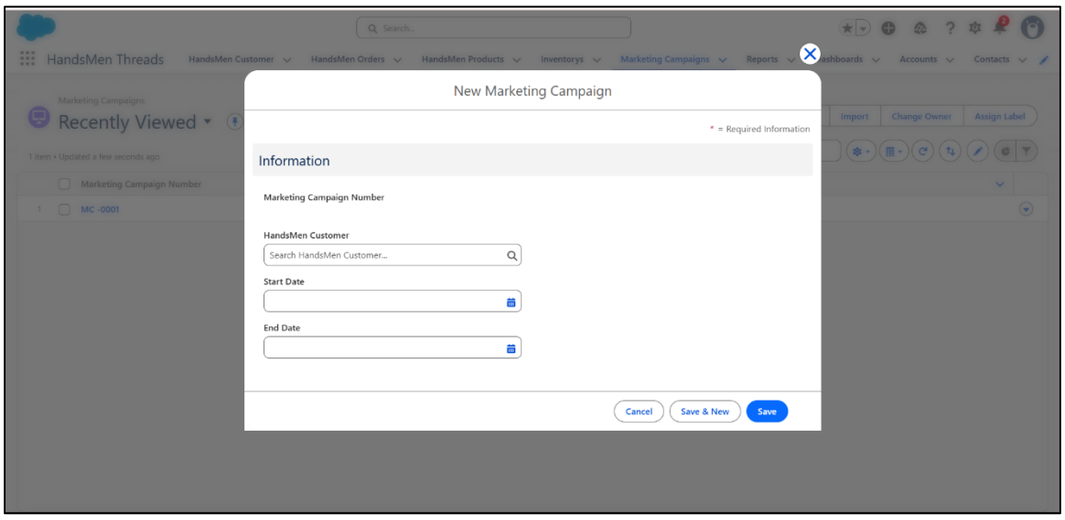


Fig. 23. Form when you add new marketing campaign

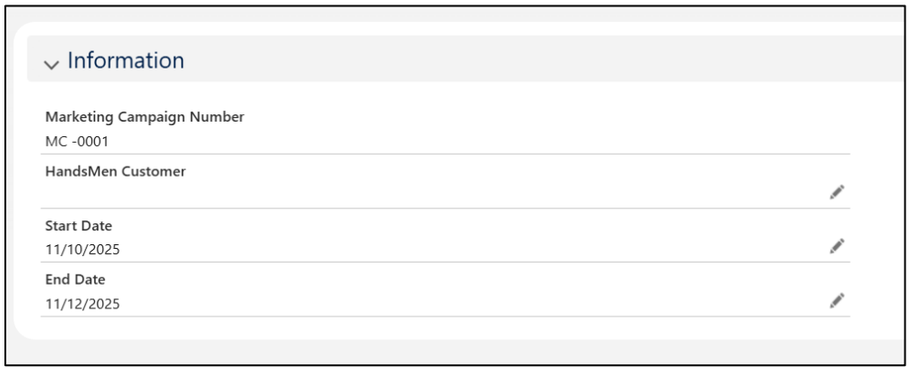


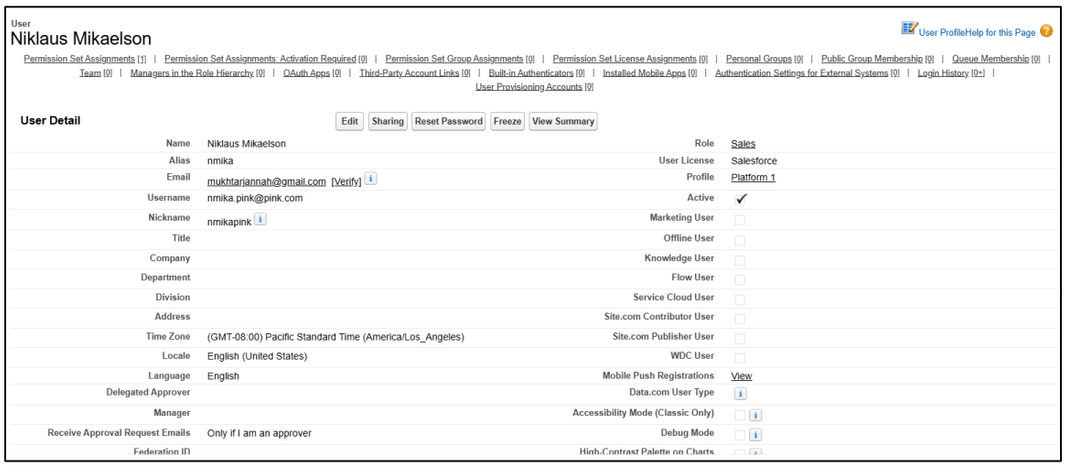
Fig. 24. Example of a Marketing Campaign

User Management

Created three users for them to handle Sales, Marketing, and Inventory.

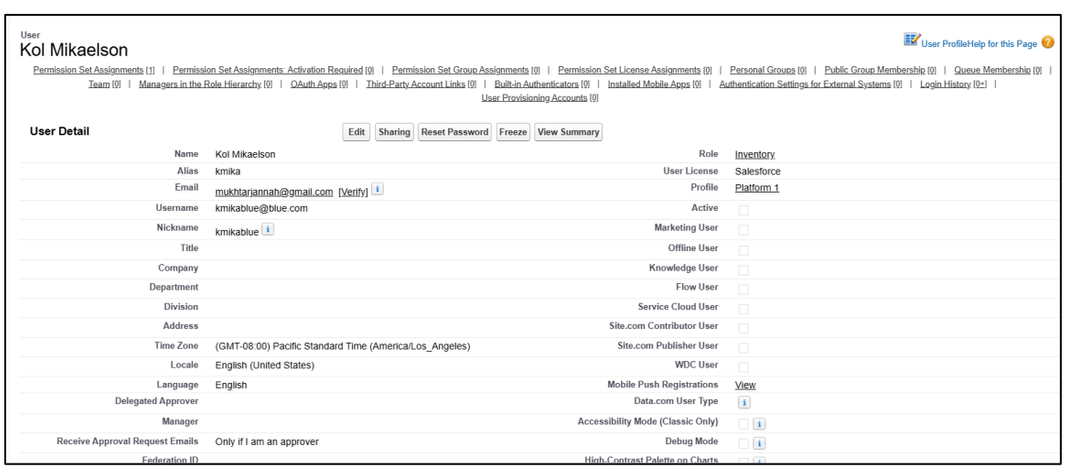
1. **Niklaus Mikaelson – Sales Management**

   The Sales manager handled customer records, processed orders, monitored purchase-related activities.



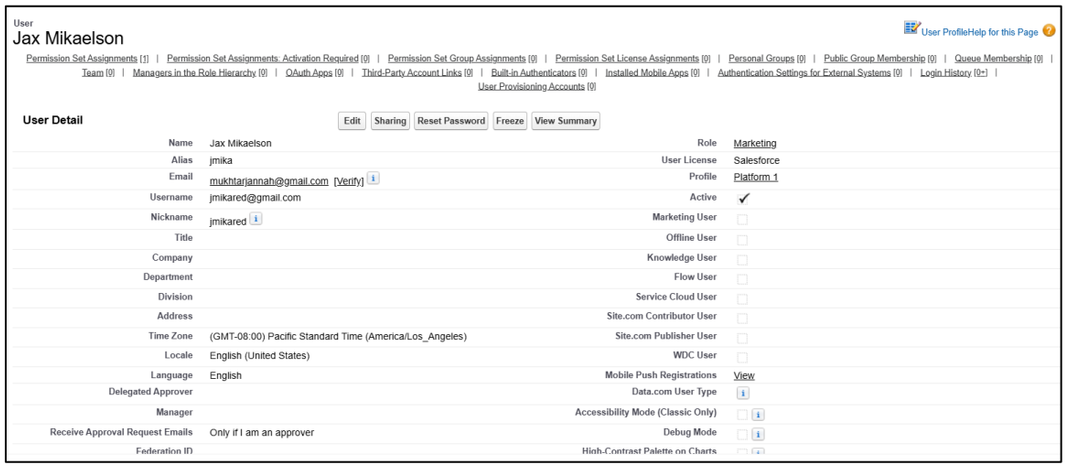
2. **Kol Mikaelson – Inventory Management**

   The Inventory manager tracked stock level, updated product quantities, and ensured accurate inventory information across the organization.



3. Jax Mikaelson

The marketing manager managed campaign records, monitored customer engagement, and promotional activities.



Reports & Dashboard

Reports and dashboards were created to provide clear visual insights into the system's operational performance. These included summaries of sales activity, customer behaviour, inventory levels, and marketing engagement. The visualizations helped users monitor trends, identify issues quickly, and make data-driven decisions to improve overall business efficiency.

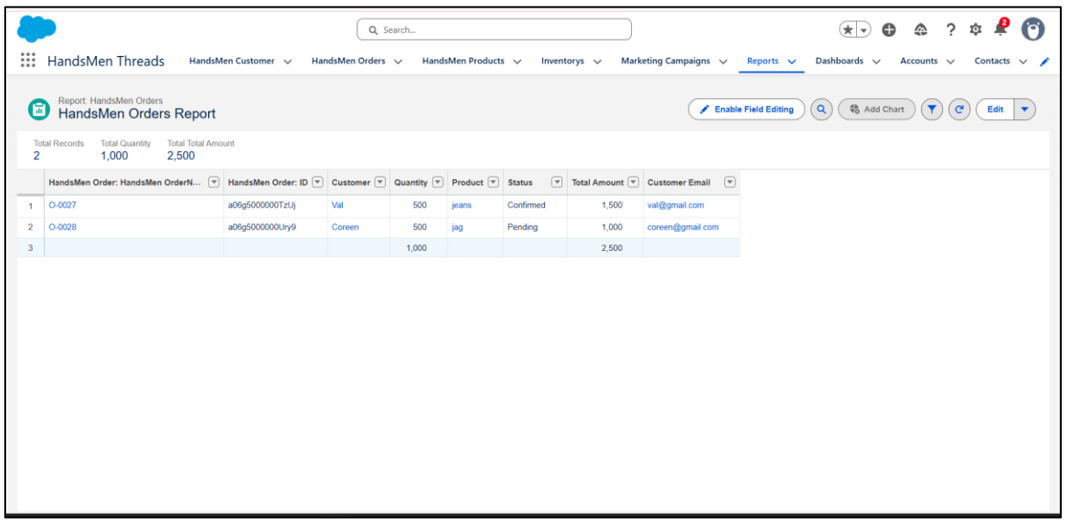


Fig. 25. Example of HandsMen Order Reports

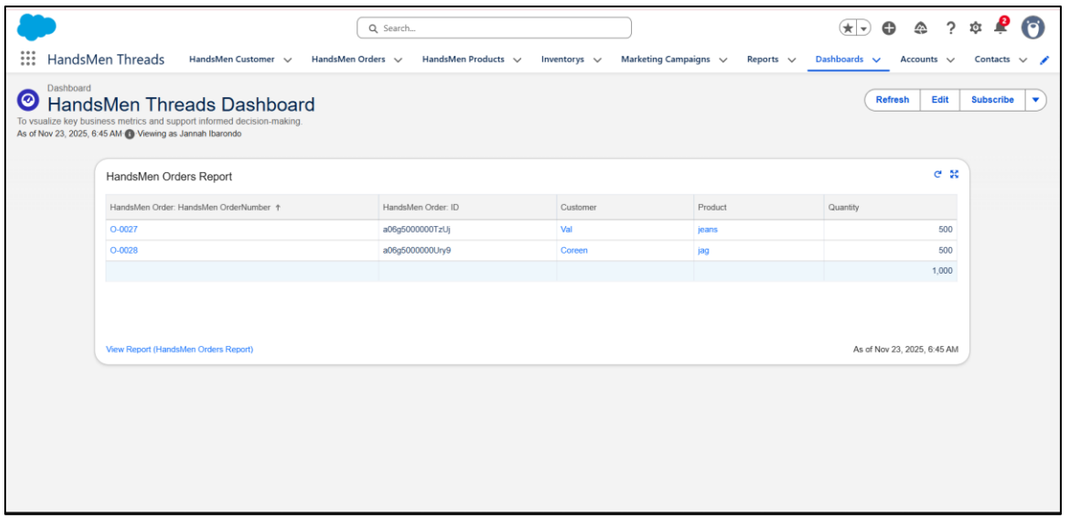
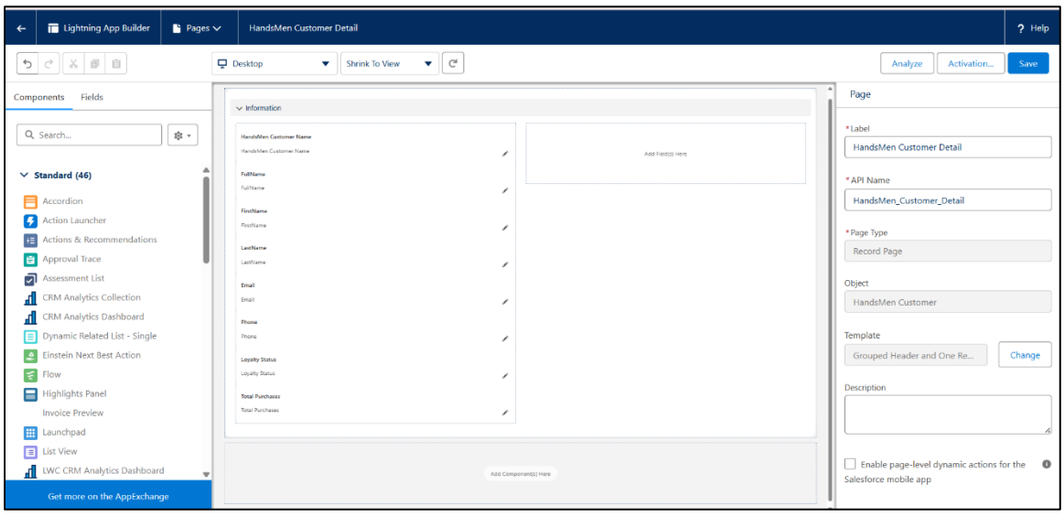


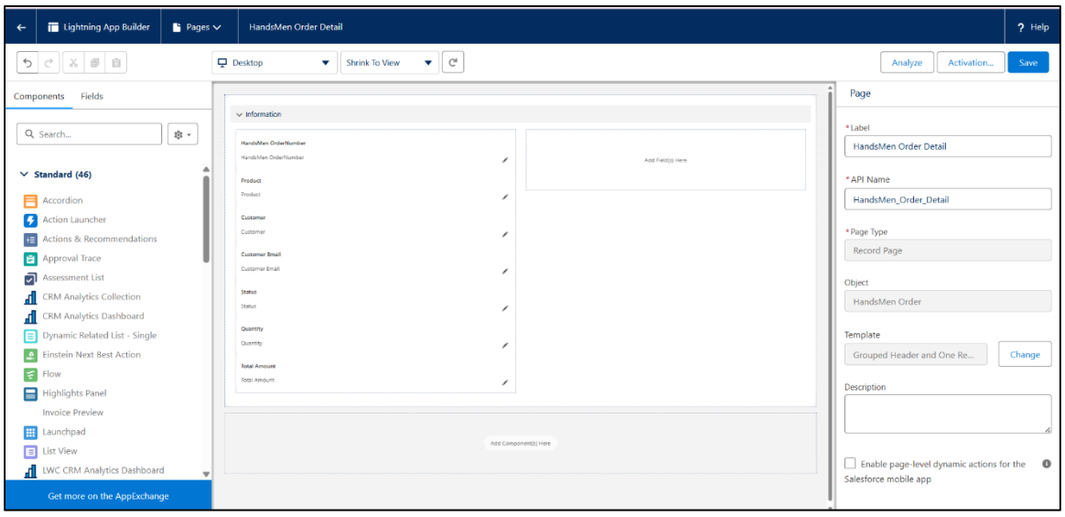
Fig. 26. Example of HandsMen Threads Dashboard

Lightning Pages

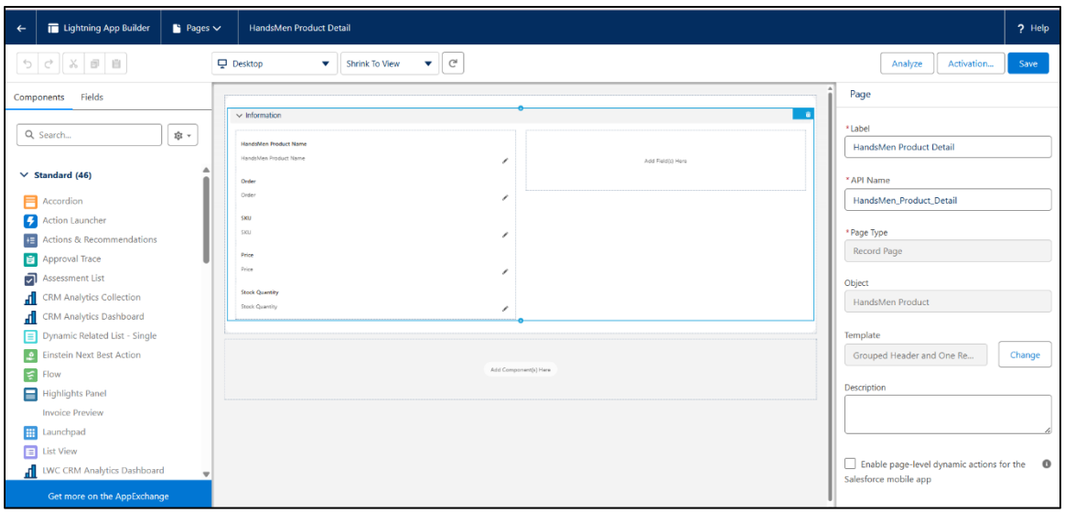
- **HandsMen Customer**



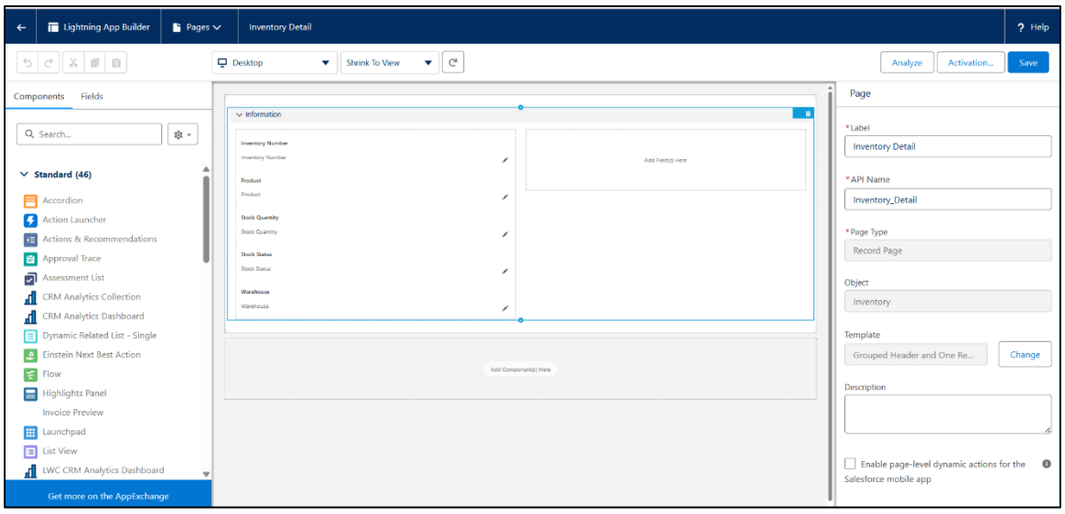
- **HandsMen Order**



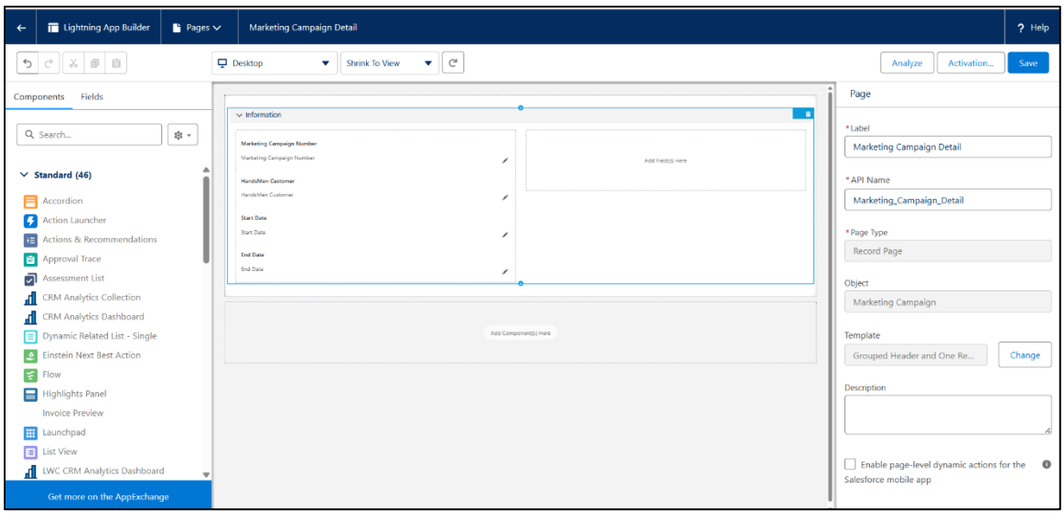
- **HandsMen Product**



- **Inventory**



- **Marketing Campaigns**



PHASE 4: DATA MIGRATION, TESTING & SECURITY

- **Field History Tracking**

Field History Tracking was configured across multiple custom objects to ensure transparency, accountability, and accurate monitoring of critical data changes within the HandsMen Threads CRM. This setup allowed the system to record past values, new values, and the user who performed the modification, supporting auditing and troubleshooting activities.

For the HandsMen Customer object, history tracking was enabled for the Total_Purchases__c field. This ensured that any changes to a customer's total purchase count were routinely recorded, allowing the system to monitor spending patterns and verify loyalty updates. Within the HandsMen Order object, the fields Total_Amount__c and Quantity__c were tracked. Recording changes to these fields helped maintain accuracy in order processing, especially when updates occurred during order reviews or corrections. For the HandsMen Product object, tracking was enabled on the Price__c and Stock_Quantity__c fields. This allowed the system to document every adjustment in product pricing and available stock, ensuring full visibility for sales and inventory teams. Finally, the Inventory object tracked changes to the Stock_Quantity__c field. This supported effective stock monitoring by capturing all inventory adjustments performed by warehouse personnel.

Overall, field history tracking strengthened the system's data integrity by maintaining a reliable audit trail for the most business-critical fields.

- **Duplicate Rules & Matching Rules**

Matching rules were configured by creating a custom matching rule for the HandsMen Customer object using the Email field as the unique identifier. The rule was activated to allow Salesforce to detect duplicate customer records. A duplicate rule was then created using this matching rule, set to alert or block users when a record with a duplicate email was entered. This ensured data accuracy and prevented multiple customer entries with the same contact information.

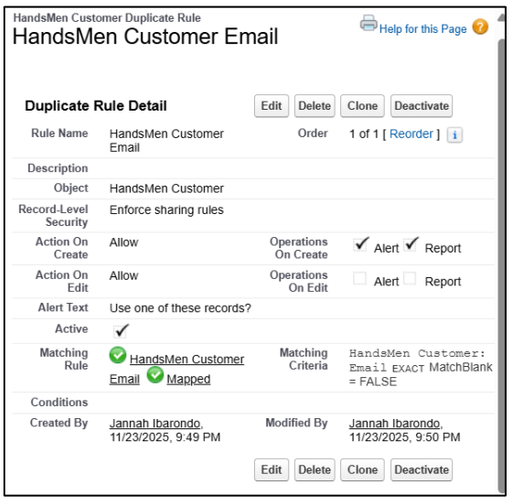


Fig. 28. Duplicate Rule

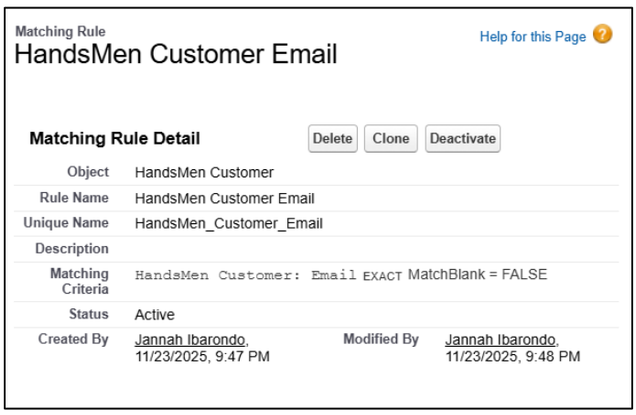


Fig. 27. Matching Rule

Profiles

The profile was named Platform 1 and was created with a Salesforce user license. Full access permissions were granted for the HandsMen Product and Inventory objects, allowing users assigned to this profile to create, view, edit, and delete records within these objects. This profile controlled what users could see and do within the system, ensuring that only authorized personnel could manage product information and inventory levels, thereby supporting accurate data management and operational efficiency.

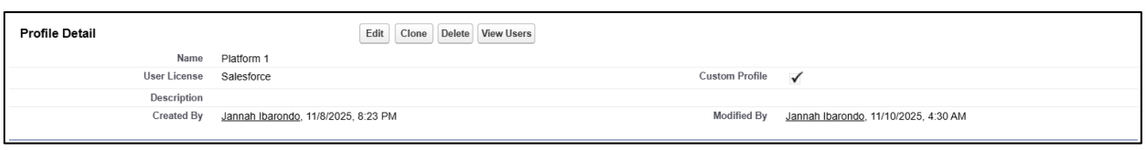


Fig. 29. Platform 1 Profile

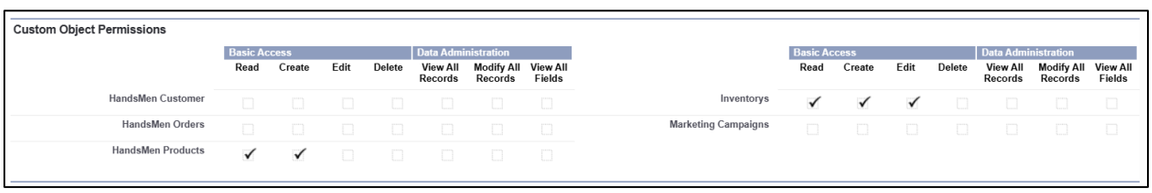


Fig. 30. Permission to HandsMen Products & Inventorys

Roles and Role Hierarchy

Fig. 31. Role Hierarchy of HandsMen Threads Organization

The CEO had top-level access to all data. The Sales role reported to the CEO and managed customer orders. The marketing role also reported to the CEO and handled campaigns and customer data. The inventory role reported to the CEO and was responsible for tracking and updating stock and products. This hierarchy ensured appropriate data visibility and access control, allowing each role to perform its functions while maintaining data security across the business organization.

Permission Sets

- **Sales Permission Set**

  The Sales Manager profile was created to provide full access to Customers and Orders. Niklaus Mikaelson was assigned to this profile to manage customer information and oversee order processing. This setup ensures that the Sales Manager can create, view, edit, and delete records within these objects to support daily sales operations.

- **Inventory Permission Set**

  The Inventory Manager profile was configured to allow read and edit access to Inventory and Product records. Kol Mikaelson was assigned to this profile to manage stock levels and product details. This ensures that the Inventory Manager can update and maintain accurate inventory and product information for the organization.

- **Marketing Permission Set**

  The Marketing Team profile was set up to allow read access to Customers and edit access to Marketing Campaigns. Jax Mikaelson was assigned to this profile to manage promotional campaigns and view customer information. This configuration ensures that the Marketing Team can update campaigns while maintaining appropriate visibility of customer data.

Test Classes Creation

Test classes were created to verify the functionality of Apex triggers, classes and batch classes developed for the HandsMen Threads Salesforce CRM system. The purpose of these test classes is to ensure that automation works correctly and maintains data integrity while simulating realistic scenarios in a safe test environment.

Test Classes Developed:

1. **OrderTotalTrigger_Test**

The test class verified that inventory was correctly deducted whenever an order with the status “Confirmed” was created. During testing, a sample product, inventory, and order were inserted, and it was confirmed that the Stock_Quantity__c decreased by the order quantity. The results showed that inventory was updated correctly, and the trigger logic executed successfully.

```
@isTest
public class OrderTotalTrigger_Test {
    @isTest
    static void testOrderTotalCalculation() {
        // 1. Create sample Product
        HandsMen_Product__c product = new HandsMen_Product__c(
            Name = 'Test Product',
            Price__c = 100
        );
        insert product;

        // 2. Create sample Customer
        HandsMen_Customer__c customer = new HandsMen_Customer__c(
            FirstName__c = 'Test Customer',
            Email__c = 'testcustomer@gmail.com'
        );
        insert customer;

        // 3. Create Order linked to product and customer
        HandsMen_Order__c order = new HandsMen_Order__c(
            Product__c = product.Id,
            Customer__c = customer.Id,
            Customer_Email__c = 'testcustomer@gmail.com',
            Quantity__c = 5,
            Status__c = 'Pending'
        );

        insert order; // Trigger runs here

        // 4. Query the inserted order
        HandsMen_Order__c insertedOrder = [
            SELECT Total_Amount__c
            FROM HandsMen_Order__c
            WHERE Id = :order.Id
        ];

        // 5. Assert Total_Amount__c is correct
        System.assertEquals(500, insertedOrder.Total_Amount__c,
            'Total_Amount__c should be Quantity * Price');
    }
}
```

Fig. 32. Test code of OrderTotalTrigger_Test

Status	Class	Result
Test Run: 2025-11-24 02:50:13, jannah.ibarondo13884@agentforce.com, (1 test class run)		
✔	[View] OrderTotalTrigger_Test	(1/1) Test Methods Passed

Fig. 33. Test result for OrderTotalTrigger_Test – passed

2. StockDeductionTrigger_Test

The purpose of the test was to verify that inventory is correctly deducted when an order with the status “Confirmed” is created. In this test scenario, a sample product, inventory record, and order were inserted, and it was checked that the Stock_Quantity__c field decreased by the order quantity. The results showed that the inventory was correctly updated, indicating that the trigger logic executed successfully.

```
@isTest
public class StockDeductionTrigger_Test {

    @isTest
    static void testStockDeduction() {
        // 1. Create sample Product
        HandsMen_Product__c product = new HandsMen_Product__c(
            Name = 'Test Product',
            Price__c = 3
        );
        insert product;

        // 2. Create Inventory record linked to Product
        Inventory__c inv = new Inventory__c(
            Product__c = product.Id,
            Stock_Quantity__c = 1000
        );
        insert inv;

        // 3. Create Order with Status = Confirmed (trigger condition)
        HandsMen_Order__c order = new HandsMen_Order__c(
            Product__c = product.Id,
            Quantity__c = 502,
            Status__c = 'Confirmed',
            Customer_Email__c = 'test@gmail.com'
        );

        insert order; // Trigger runs here

        // 4. Query inventory after trigger execution
        Inventory__c updatedInv = [
            SELECT Stock_Quantity__c
            FROM Inventory__c
            WHERE Id = :inv.Id
        ];

        // 5. Validate that stock was deducted correctly
        System.assertEquals(498, updatedInv.Stock_Quantity__c,
            'Stock should be reduced by order quantity');
    }
}
```

Fig. 34. Test code of StockDeductionTrigger_Test

Status	Class	Result
Test Run: 2025-11-24 03:27:56, jannah.ibarondo13884@agentforce.com, (1 test class run)		
✔	[View] StockDeductionTrigger_Test	(1/1) Test Methods Passed

Fig. 35. Test result for StockDeductionTrigger_Test – passed

3. LoyaltyStatusTrigger_Test

The purpose of the LoyaltyStatusTrigger_Test was to ensure that customer loyalty statuses are correctly updated based on their total purchases. In the test scenario, sample customers with varying Total_Purchases__c values were created. The results confirmed that customers with purchases over 1000 were assigned “Gold,” those with 500 or less were assigned “Bronze,” and those in between were assigned “Silver,” indicating that the trigger logic worked as intended.

```
@isTest
public class LoyaltyStatusTrigger_Test {
    @isTest
    static void testLoyaltyStatusUpdate() {
        // 1. Create Customers with different purchase totals
        HandsMen_Customer__c goldCustomer = new HandsMen_Customer__c(
            FirstName__c = 'Gold Customer',
            Email__c = 'gold@gmail.com',
            Total_Purchases__c = 1500
        );

        HandsMen_Customer__c silverCustomer = new HandsMen_Customer__c(
            FirstName__c = 'Silver Customer',
            Email__c = 'silver@gmail.com',
            Total_Purchases__c = 700
        );

        HandsMen_Customer__c bronzeCustomer = new HandsMen_Customer__c(
            FirstName__c = 'Bronze Customer',
            Email__c = 'bronze@gmail.com',
            Total_Purchases__c = 400
        );

        insert new List<HandsMen_Customer__c>{goldCustomer, silverCustomer, bronzeCustomer};

        // 2. Trigger runs automatically on insert or update
        // No additional action needed

        // 3. Query customers
        List<HandsMen_Customer__c> updatedCustomers = [
            SELECT Loyalty_Status__c, Total_Purchases__c
            FROM HandsMen_Customer__c
            WHERE Id IN :new List<Id>{goldCustomer.Id, silverCustomer.Id, bronzeCustomer.Id}
        ];

        // 4. Assert loyalty statuses
        for(HandsMen_Customer__c c : updatedCustomers){
            if(c.Total_Purchases__c > 1000){
                System.assertEquals('Gold', c.Loyalty_Status__c);
            } else if(c.Total_Purchases__c <= 500){
                System.assertEquals('Bronze', c.Loyalty_Status__c);
            } else {
                System.assertEquals('Silver', c.Loyalty_Status__c);
            }
        }
    }
}
```

Fig. 36. Test code for LoyaltyStatusTrigger_Test

Fig. 37. Test result for LoyaltyStatusTrigger_Test – passed

4. InventoryBatchJob_Test

The purpose of the InventoryBatchJob_Test was to verify that the Inventory Batch Job correctly updates stock levels for products below the defined threshold. In the test scenario, sample products with low stock were created, and the batch job was executed to apply the restocking logic. The results showed that stock quantities were correctly updated, confirming that the batch job functioned as intended.

```
@isTest
public class InventoryBatchJob_Test {
    @isTest
    static void testInventoryBatchJob() {
        // 1. Create sample products with low stock
        List<HandsMen_Product__c> products = new List<HandsMen_Product__c>();
        products.add(new HandsMen_Product__c(Name='Product 1', Stock_Quantity__c=5));
        products.add(new HandsMen_Product__c(Name='Product 2', Stock_Quantity__c=2));
        insert products;

        // 2. Run the batch
        Test.startTest();
        InventoryBatchJob batch = new InventoryBatchJob();
        Database.executeBatch(batch, 200);
        Test.stopTest();

        // 3. Verify that the stock was updated correctly
        List<HandsMen_Product__c> updatedProducts = [
            SELECT Stock_Quantity__c FROM HandsMen_Product__c
        ];

        for (HandsMen_Product__c p : updatedProducts) {
            System.assert(p.Stock_Quantity__c >= 50,
                'Stock should be restocked by 50 units');
        }
    }
}
```

Fig. 38. Test code for InventoryBatchJob_Test

Status	Class	Result
Test Run: 2025-11-24 03:44:20, jannah.ibarondo13884@agentforce.com, (1 test class run)		
✔	[View] InventoryBatchJob_Test	(1/1) Test Methods Passed

Fig. 39. Test result for InventoryBatchJob_Test – passed

5. LoyaltyPointsBatch_Test

The purpose of the LoyaltyPointsBatch_Test was to verify that customer loyalty statuses are correctly updated based on their Total_Purchases__c. In the test scenario, sample customers were created with purchase amounts corresponding to bronze, silver, and gold tiers. The batch job was executed, and the results showed that the loyalty statuses were correctly updated for all test customers, confirming that the batch logic worked as intended.

```
@isTest
public class LoyaltyPointsBatch_Test {

    @isTest
    static void testLoyaltyPointsBatch() {
        // 1. Create sample customers
        HandsMen_Customer__c cust1 = new HandsMen_Customer__c(
            Name = 'Customer Gold',
            Total_Purchases__c = 1500,
            Email__c = 'test@gmail.com'
        );
        HandsMen_Customer__c cust2 = new HandsMen_Customer__c(
            Name = 'Customer Silver',
            Total_Purchases__c = 700,
            Email__c = 'test@gmail.com'
        );
        HandsMen_Customer__c cust3 = new HandsMen_Customer__c(
            Name = 'Customer Bronze',
            Total_Purchases__c = 300,
            Email__c = 'test@gmail.com'
        );

        insert new List<HandsMen_Customer__c>{cust1, cust2, cust3};

        // 2. Execute batch class
        Test.startTest();
        LoyaltyPointsBatch batch = new LoyaltyPointsBatch();
        Database.executeBatch(batch, 200);
        Test.stopTest();

        // 3. Verify results
        cust1 = [SELECT Loyalty_Status__c FROM HandsMen_Customer__c WHERE Id = :cust1.Id];
        cust2 = [SELECT Loyalty_Status__c FROM HandsMen_Customer__c WHERE Id = :cust2.Id];
        cust3 = [SELECT Loyalty_Status__c FROM HandsMen_Customer__c WHERE Id = :cust3.Id];

        System.assertEquals('Gold', cust1.Loyalty_Status__c, 'Gold customer should be Gold');
        System.assertEquals('Silver', cust2.Loyalty_Status__c, 'Silver customer should be Silver');
        System.assertEquals('Bronze', cust3.Loyalty_Status__c, 'Bronze customer should be Bronze');
    }
}
```

Fig. 40. Test code for LoyaltyPointsBatch_Test

Status	Class	Result
Test Run: 2025-11-24 03:53:09, jannah.ibarondo13884@agentforce.com, (1 test class run)		
✔	[View] LoyaltyPointsBatch_Test	(1/1) Test Methods Passed

Fig. 41. Test result for LoyaltyPointsBatch_Test – passed

PHASE 5: DEPLOYMENT, DOCUMENTATION & MAINTENANCE

1. **Deployment Strategy**

   All configurations and customizations, including objects, fields, automation, triggers, flows, email alerts, and batch jobs, were developed directly within the main Salesforce org. Deployment was straightforward because no migration between environments was required. Each feature was manually tested after implementation to ensure correct behavior and system reliability. Once validated, the components were finalized and made ready for use in the system.

2. **Maintenance & Monitoring**

   System maintenance will be carried out through periodic reviews of automation rules, batch jobs, validation rules, and user permissions. Logs from Apex Jobs, Debug Logs, and Flow Error logs will be monitored regularly to detect any workflow or automation failures. Additionally, administrators will update page layouts, user roles, and permission sets as business needs evolve.

3. **Troubleshooting Approach**

   A structured troubleshooting process was followed:

   1. Review errors shown in the UI, debug logs, or Apex Test failures.
   2. Determine whether the problem is related to missing fields, incorrect permissions, automation errors, or Apex logic.
   3. Modify affected components (flows, triggers, validation rules, fields, etc.).
   4. Execute Apex tests, manually test UI behavior, and confirm correct functionality.
   5. Record the issue, resolution steps, and updated components for future reference.

CONCLUSION

The Salesforce CRM solution for HandsMen Threads successfully streamlined business operations, maintains data integrity, and enhanced customer satisfaction. Automation through Flows, Apex triggers, and scheduled jobs reduced manual work, while email alerts and loyalty updates improved engagement. The system was scalable, user-friendly, and aligned with business goals.

RECOMMENDATIONS

1. Add more validation rules—such as mandatory fields, format checks, and logical constraints—to maintain clean and reliable data across all modules.
2. Implement additional marketing automation such as personalized email campaigns, loyalty rewards reminders, and abandoned cart follow-ups to enhance customer retention.
3. Use Einstein to predict sales trends, high-value customers, and product demand, enabling better inventory and marketing decisions.