## Project Title: HandsMen Threads: A Custom Salesforce CRM for Fashion Retail

### Problem Statement:

For the "HandsMen Threads" project, I am addressing the operational challenges of a growing men's fashion brand. The core problem is the lack of a centralized and automated system to manage their key business functions. They are currently struggling with disconnected processes for tracking customer information, managing the product catalog, processing sales orders, and monitoring inventory levels. This results in operational inefficiencies, a high risk of data errors, potential stock management issues, and a limited ability to track customer loyalty. To solve this, I will implement a custom Salesforce application that unifies all these functions. My goal is to create a single platform to streamline order processing, provide real-time inventory and sales data, and build a robust system for managing customer relationships and marketing campaigns, enabling the business to scale effectively.

# 1. Phase 1: Problem Understanding & Industry Analysis

1.1. Executive Summary

The "HandsMen Threads" project involved the end-to-end implementation of a custom Salesforce application designed for a men's fashion retail company. The primary goal was to create a centralized system to manage customers, products, inventory, sales orders, and marketing initiatives, thereby moving away from siloed data and manual processes. This document outlines the ten phases of the project, detailing the steps I took to build, configure, and automate the solution.

1.2. Requirement Gathering

The project requirements were sourced from a set of instructions for a Salesforce internship project. These instructions provided a clear blueprint for the necessary objects, fields, automation, and custom code needed to build a functional proof-of-concept application.

1.3. Business Process Mapping

Based on the requirements, I mapped the following core business processes:

Lead-to-Customer: Managing customer information and loyalty.
Order-to-Cash: Handling sales orders from creation to confirmation.
Inventory Management: Tracking product stock levels and alerting on shortages.
Campaign Management: Creating and tracking marketing campaigns.

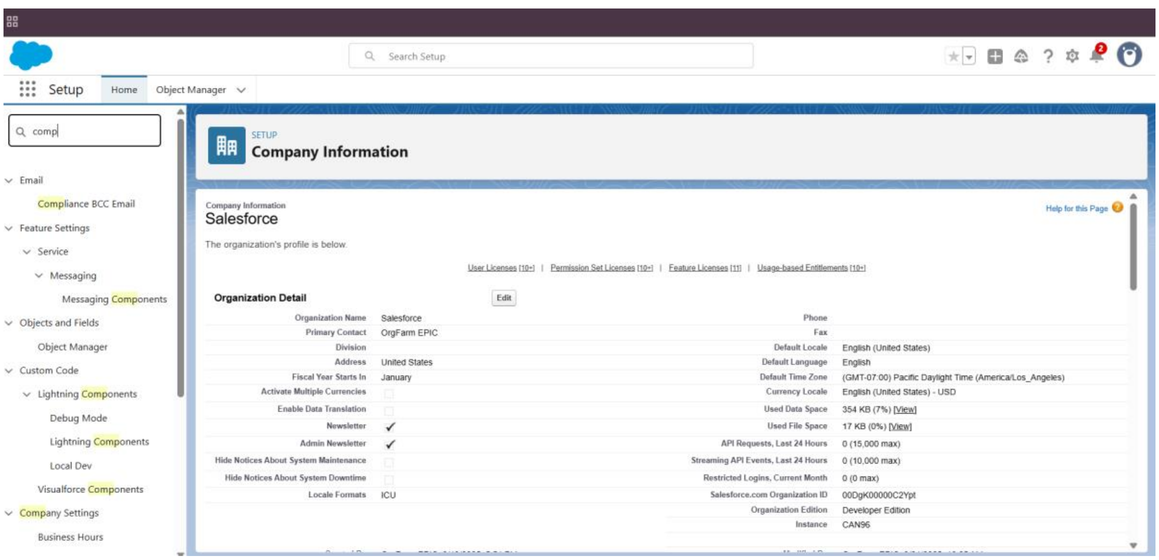
### 1.4. Key Objectives

To build a robust and scalable data model using custom objects.
To automate repetitive tasks using Salesforce's declarative tools (Flows, Validation Rules).
To implement complex business logic using Apex triggers and batch classes.
To provide a simple and intuitive user interface through a custom Lightning App.

# 2. Phase 2: Org Setup & Configuration

### 2.1. Developer Org Setup

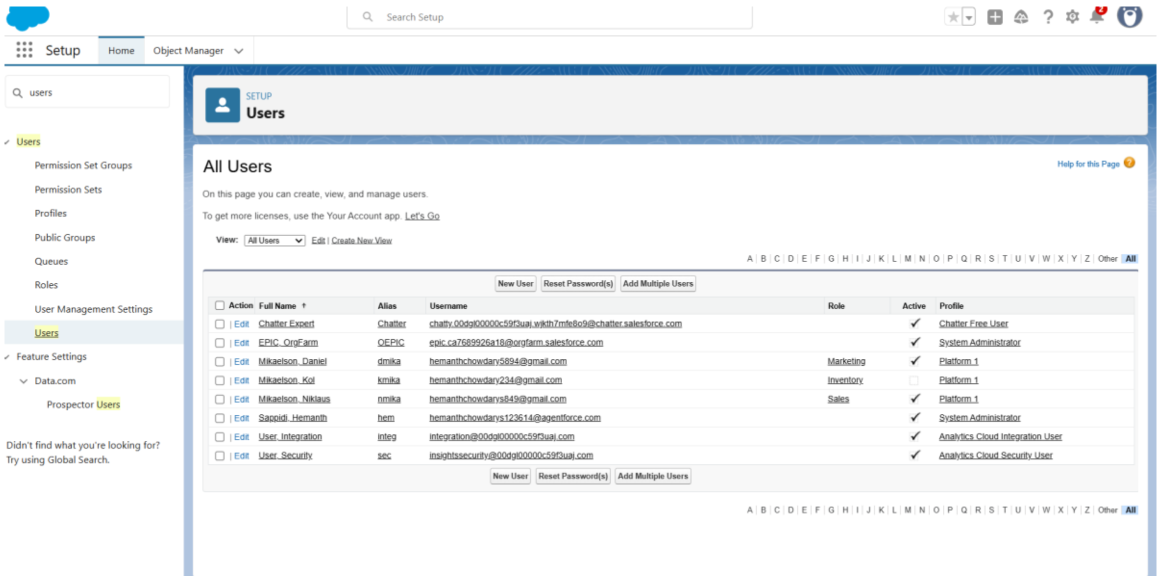
The project's foundation was a brand new Salesforce Developer Edition org. I initiated this by navigating to the Salesforce developer site, filling in the required information, and verifying the new account via email. This provided a clean and isolated environment for the entire build.




### 2.2. User & License Management

To simulate a real-world multi-user environment, I created three distinct users. Each user was assigned a Salesforce license and a custom profile to define their base-level access.

Nicholas Miklson (Sales Role)
Co Miklson (Inventory Role)
Daniel Miklson (Marketing Role)




2.3. Security Model: Profiles

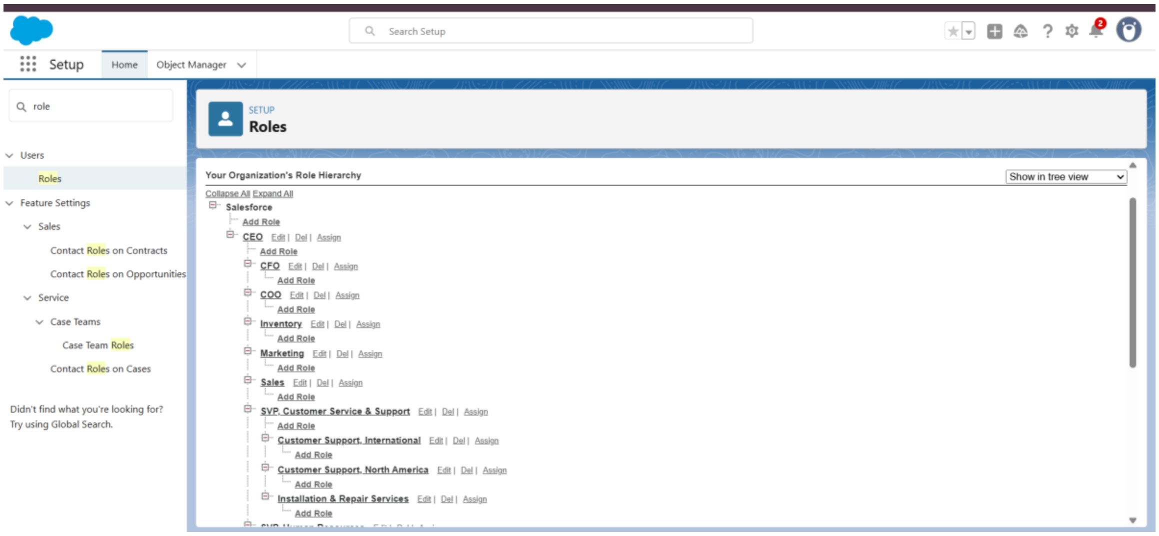
To manage user permissions at a foundational level, I created a custom profile. Instead of building from scratch, I cloned the standard "Standard User" profile to create a new profile named "Platform One". This new profile served as the baseline for all non-admin users in the system. Initially, it had minimal access, which I later augmented with Permission Sets.

[Add Screenshot Here: The profile list in Setup, with an arrow pointing to the newly created "Platform One" profile.]

2.4. Security Model: Roles & Hierarchy

I established a basic role hierarchy to model the organizational structure and to enable record access roll-up for future reporting needs. All new roles were created to report to the CEO.

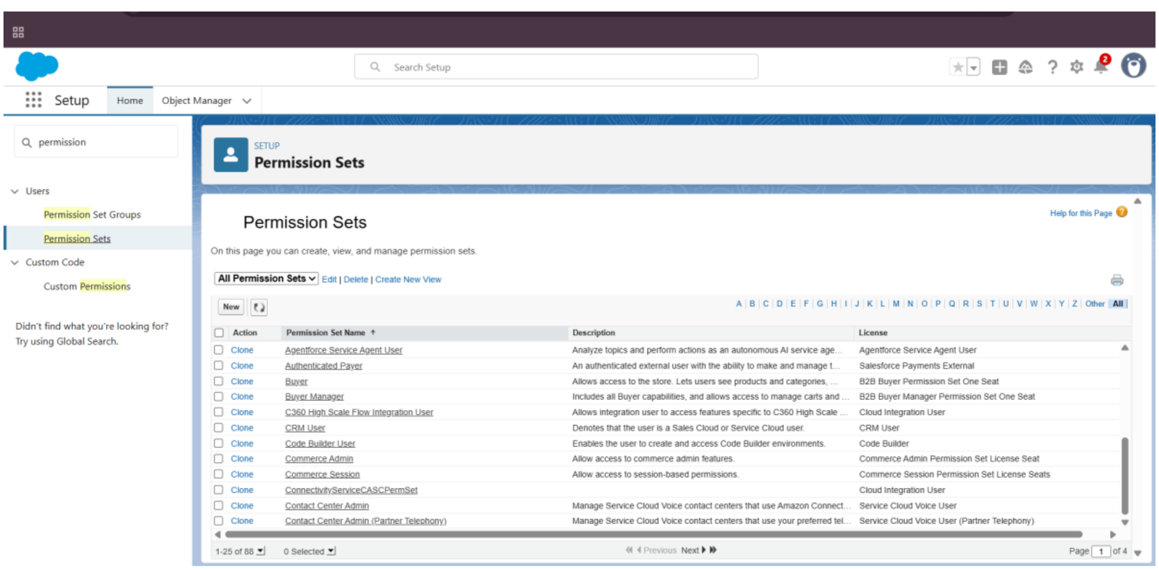
CEO
Sales Manager (Assigned to Nicholas Miklson)
Inventory Manager (Assigned to Co Miklson)
Marketing Manager (Assigned to Daniel Miklson)




2.5. Security Model: Permission Sets

To grant specific, granular permissions to users based on their job function, I created three Permission Sets. This approach is more flexible than managing permissions solely on profiles.

Sales Permission Set: Granted full Create, Read, Edit, and Delete (CRUD) access to the Handsmen Customer and Handsmen Order objects. This was assigned to the Sales user.
Inventory Permission Set: Granted Read and Edit access to the Inventory and Handsmen Product objects. This was assigned to the Inventory user.
Marketing Permission Set: Granted Read access to Handsmen Customer and Edit access to Marketing Campaign. This was assigned to the Marketing user.



# 3. Phase 3: Data Modeling & Relationships

## 3.1. Custom Object Architecture

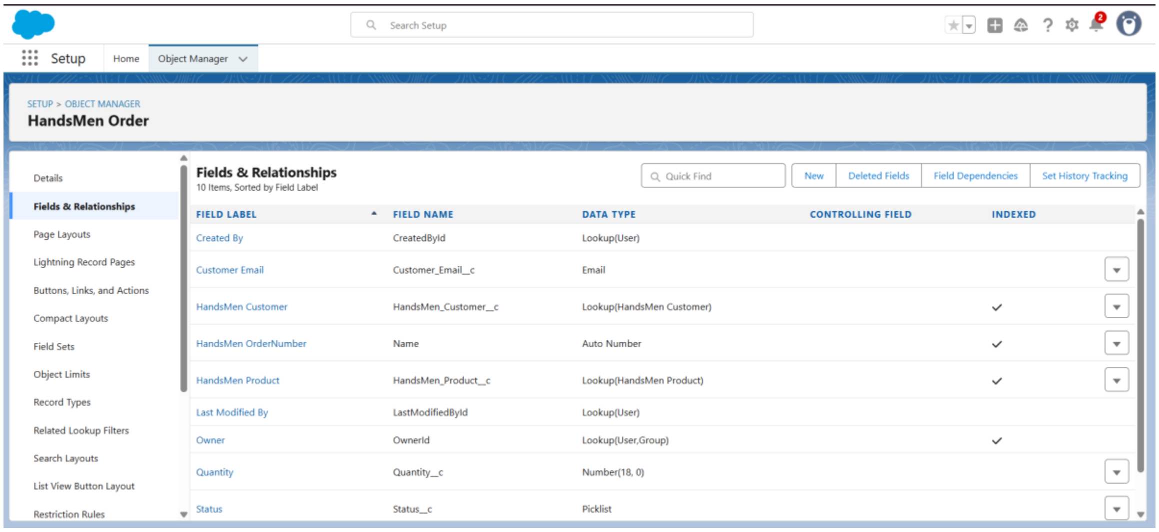
I created five core custom objects to structure the application's data:

Handsmen Customer: To store all information related to customers, including contact details and purchase history.
Handsmen Product: To act as the master catalog for all products sold by the company.
Handsmen Order: To capture individual sales transactions, linking customers to the products they purchase.
Inventory: To track stock levels for each product at a specific location or warehouse.
Marketing Campaign: To manage marketing initiatives and track their association with customers.

## 3.2. Field Implementation Details

I added numerous custom fields to these objects to capture all required business data. For example, on the Handsmen Customer object, I created:

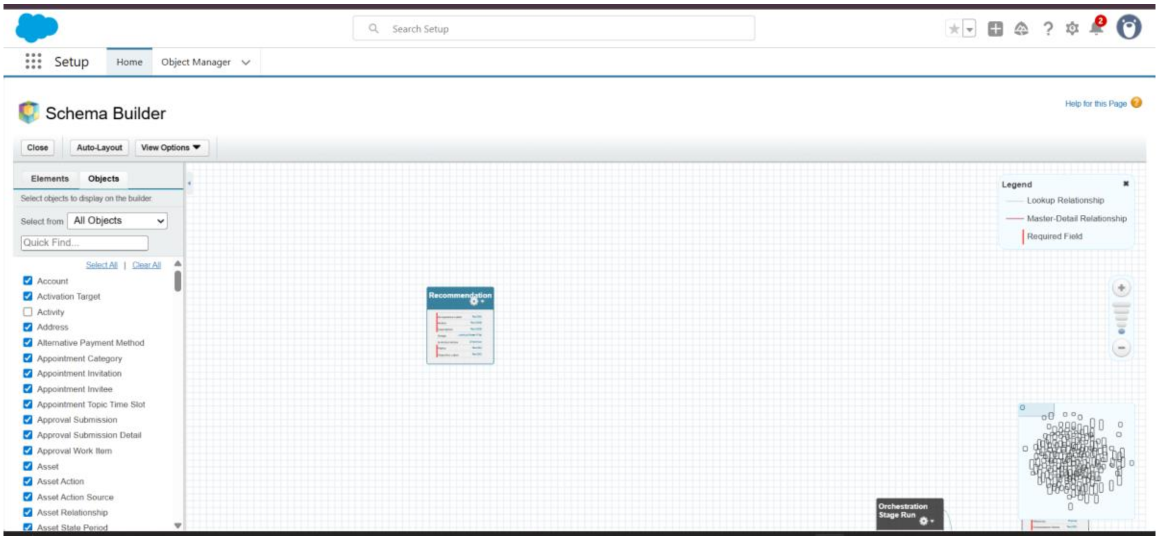
Email (Email)
Phone (Phone)
First Name (Text)
Last Name (Text)
Full Name (Formula - concatenates First and Last Name)
Loyalty Status (Picklist: Gold, Silver, Bronze)
Total Purchases (Number)




3.3. Relationship Modeling (Lookup vs. Master-Detail)
I carefully chose the relationship types to model the connections between objects accurately.

Lookup Relationships: I used lookups for relationships where the objects could exist independently.
Handsmen Order -> Handsmen Customer: An order must have a customer, but a customer can exist without placing an order.
Handsmen Order -> Handsmen Product: An order is for a product, but a product can exist in the catalog before it is ever ordered.

Master-Detail Relationship: I used a Master-Detail relationship where one object's existence is entirely dependent on the other.
Handsmen Product (Master) -> Inventory (Detail): An inventory record for stock quantity is meaningless without a product to which it belongs. Deleting a product record will cascade-delete all its related inventory records.




# 4. Phase 4: Process Automation (Admin)

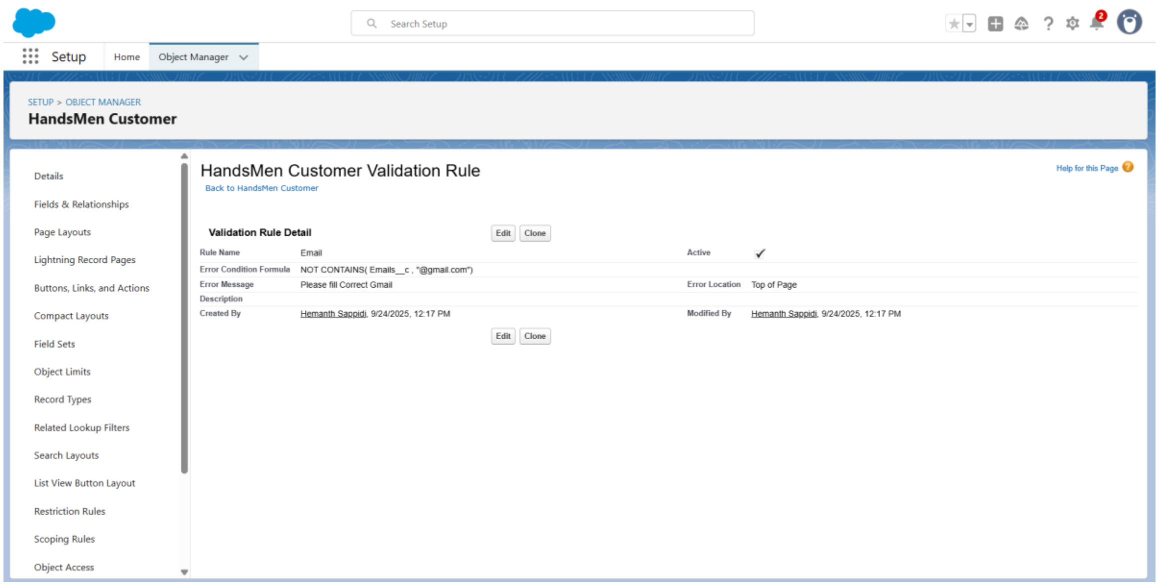
## 4.1. Data Integrity: Validation Rules

To enforce business rules and ensure high-quality data, I implemented three validation rules.

Object: Handsmen Order
Rule: Total Amount must be positive.
Formula: Total_Amount__c <= 0
Error Message: "Please enter correct amount"

Object: Inventory
Rule: Stock Quantity cannot be negative.
Formula: Stock_Quantity__c <= 0
Error Message: "The inventory count is never less than zero"

Object: Handsmen Customer
Rule: Customer email must be a Gmail address.
Formula: NOT(CONTAINS(Email__c, "@gmail.com"))
Error Message: "Please fill correct Gmail"




4.2. Communication: Email Templates & Alerts
I created a Classic Letterhead and three email templates to standardize communications.

Templates: Order Confirmation (HTML), Low Stock Alert (Text), Loyalty Program (HTML).
Email Alerts: I then configured three corresponding Email Alerts to package these templates for use in my flows.

[Add Screenshot Here: The setup page for the "Order Confirmation Email Alert," showing the object, email template, and recipient type selected.]

4.3. Automation Engine: Flow Builder
4.3.1. Record-Triggered Flow: Order Confirmation
Objective: To automatically send a confirmation email when an order is finalized.
Trigger: Fires when an Handsmen Order record is updated.
Entry Criteria: Status field is changed to "Confirmed".
Action: Calls the "Order Confirmation" Email Alert, sending the HTML email to the customer.

[Add Screenshot Here: The canvas of the Order Confirmation Flow, showing the start element with trigger conditions and the "Send Email Alert" action element.]

4.3.2. Record-Triggered Flow: Low Stock Alert
Objective: To notify the inventory manager when product stock runs low.

Trigger: Fires when an Inventory record is created or updated.
Entry Criteria: Stock Quantity is less than 5.
Action: Calls the "Low Stock Alert" Email Alert, sending a text email to the inventory manager.

[Add Screenshot Here: The canvas of the Low Stock Alert Flow, showing the trigger and the email alert action.]

4.3.3. Scheduled Flow: Loyalty Status Update
Objective: To run a daily batch process that updates the loyalty status of all customers based on their spending.
Trigger: Runs on a daily schedule (e.g., 12:00 AM).
Logic:
Get Records: Fetches all Handsmen Customer records.
Loop: Iterates through each customer record.
Decision: Checks the Total Purchases value.
If > 1000 -> Path to "Gold"
If <= 500 -> Path to "Bronze"
Default -> Path to "Silver"
Update Records: Updates the Loyalty Status field on the current customer record based on the decision outcome.
Send Email: Calls the "Loyalty Program" Email Alert to notify the customer of their status.

[Add Screenshot Here: A detailed view of the Loyalty Status scheduled flow on the Flow canvas, highlighting the Loop and Decision elements.]

# 5. Phase 5: Apex Programming (Developer)

5.1. Apex Trigger: Order Total Calculation
Business Rationale: The total amount of an order should be calculated automatically to prevent manual errors. It should be the product's price multiplied by the quantity ordered.

Implementation: I created a 'before insert' and 'before update' trigger on the Handsmen Order object. It queries the related Product's price and performs the calculation before saving the record.

[Add Screenshot Here: The Developer Console showing the complete code for the OrderTotalTrigger.]

5.2. Apex Trigger: Stock Deduction Logic
Business Rationale: When an order is confirmed, the quantity of the ordered product should be automatically deducted from the main inventory to ensure stock levels are always accurate.

Implementation: I created an 'after update' trigger on the Handsmen Order object. It checks if the status was changed to "Confirmed" and then updates the Stock_Quantity__c on the related Inventory record.

[Add Screenshot Here: The Developer Console showing the StockDeductionTrigger code.]

5.3. Asynchronous Apex: Inventory Batch Job
Business Rationale: To handle large-scale data processing on inventory records without hitting governor limits, I created the structure for a Batch Apex class. This could be used for nightly stock reconciliation or other bulk updates.

Implementation: I created an Apex Class InventoryBatchJob that implements the Database.Batchable<sObject> interface. The video demonstrated its creation and successful execution from the Developer Console.

[Add Screenshot Here: The InventoryBatchJob Apex class code in the Developer Console.]
[Add Screenshot Here: The Execute Anonymous window with the code to run the batch job and the successful log.]

# 6. Phase 6: User Interface Development

6.1. Application Creation: Lightning App Builder
To provide users with a dedicated and intuitive interface, I used the Lightning App Builder to create a custom app.
App Name: HandsMen Threads
Configuration: I added a logo (placeholder), a description, and selected the necessary navigation items.

[Add Screenshot Here: The "App Details & Branding" step in the Lightning App Builder for the HandsMen Threads app.]

6.2. Navigation: Custom Object Tabs
I created a custom tab for each of the five custom objects and for the standard Reports and Dashboards objects. I then added these tabs to the "HandsMen

Threads" app's navigation bar to ensure users could easily access all parts of the system.

[Add Screenshot Here: The final, running "HandsMen Threads" application, showing the full navigation bar with tabs for Customers, Products, Orders, Inventories, etc.]

## 7. Phase 7: Integration & External Access (Out of Scope)

This project focused on building a self-contained application within the Salesforce platform. Integration with external systems (e.g., e-commerce websites, accounting software) using REST/SOAP APIs was not part of the initial project scope but could be considered a future enhancement.

## 8. Phase 8: Data Management & Deployment (Out of Scope)

All metadata components (Objects, Fields, Code, etc.) were created manually in the Developer Org. Data was entered manually for testing purposes. A formal deployment process using Change Sets, SFDX, or other migration tools was not included in this project phase.

## 9. Phase 9: Reporting, Dashboards & Security Review

### 9.1. Security Model Summary

A comprehensive security model was a key part of this project. Access was controlled using a combination of:
Profiles: To define the baseline permissions.
Roles: To control the data visibility hierarchy.
Permission Sets: To grant specific, additive permissions based on job function.

### 9.2. Reporting & Dashboard Framework

While the creation of specific reports and dashboards was not part of this project's scope, I prepared the application for future reporting needs. By adding the "Reports" and "Dashboards" tabs to the application, I ensured that users have a direct path to create their own analytics once data populates the system.

## 10. Phase 10: Final Presentation & Demo Day

### 10.1. Project Demonstration & Key Outcomes

I successfully demonstrated the functionality of the "HandsMen Threads" application by walking through several key use cases:
- Creating a Customer and Product: Showcased the custom objects and fields.
- Placing an Order: Demonstrated the OrderTotalTrigger automatically calculating the total amount.

- Confirming an Order: Showcased the StockDeductionTrigger reducing the inventory quantity and the Order Confirmation Flow sending an email.
- Triggering a Low Stock Alert: Manually updated an inventory record below the threshold to trigger the alert flow.
- Testing Loyalty Status: Manually ran the scheduled flow in debug mode to show the Loyalty Status field being updated on a customer record.

[Add Screenshot Here: An example of a successful order confirmation email received in an inbox during the demo.]
[Add Screenshot Here: The Inventory record before and after an order was confirmed, showing the stock quantity being reduced.]

## 10.2. Conclusion

The "HandsMen Threads" project successfully delivered a functional and automated CRM application on the Salesforce platform. By leveraging a combination of declarative tools and custom Apex code, I built a solution that meets all the core requirements for managing a retail fashion business's key operations. The project serves as a strong foundation that can be expanded with further features like reporting, integrations, and more complex automation.