Introduction

Hi, I'm Kommana Anil Kumar, currently pursuing my graduation at SRM University - AP, and this is my Salesforce CRM project, created to address a common but often overlooked problem in the fashion and small retail industry. In many growing local or mid-level brands, the real challenge isn't the quality of clothes or materials, it's the broken system behind selling them. Despite having loyal customers, poor backend operations such as missed orders, inaccurate stock data, and lack of communication can gradually harm the brand. Everything might look perfect from the outside—good product photos, happy customers—but inside, things fall apart. Orders get missed, stock appears available when it's not, and customer satisfaction quietly drops.

Many of these businesses still rely on outdated methods like writing things down or using Excel. One person takes the order, other packs it, and someone else updates the stock based on word-ofmouth, leading to confusion, fraud, or duplicate handling. There's no real tracking of loyal customers or automated messages like "your order is confirmed" or "thanks for shopping." It's not due to neglect, but simply a lack of a system that can manage these tasks automatically.

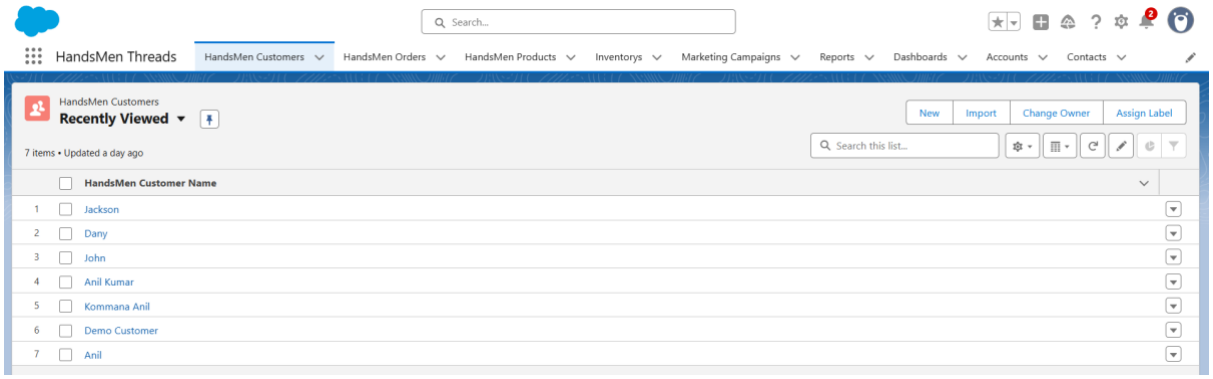
This unmanaged workflow causes a silent but dangerous leak. The business wants to grow, but the foundation is unstable. What these businesses truly need is not a new clothing line or fancy website, but a smart system—a brain behind the business—that connects orders, stock, customers, and processes. That's where Salesforce comes in.

This project uses Salesforce tools like Flows, Apex, and custom objects to build a system that thinks, reacts, and automates essential tasks. Instead of manually updating sheets or messaging customers, the system does it for them—seamlessly, reliably, and intelligently. This project is about solving that internal chaos—not by changing the business itself, but by changing how it operates behind the scenes. The goal is to create a system that listens and acts on its own, like a well-designed trigger, keeping everything in sync without requiring constant human input.

App Overview

To provide easy navigation and access for the end users, we created a custom Salesforce App name "HandsMen Threads". This app bundles all essential objects and functionalities into a unified workspace. Within the app, we added custom tabs for each key objects such as custom objects we created above and some standard objects like reports, dashboards. These tabs allow users to view create and manage record directly from the interface without needing to navigate through setup. The App ensures a centralized and role specific environment for efficient operations. Proper tab visibility was set based on user profiles to ensure a clean and secure UI experience.

The below image shows the tab output of the app after creating tabs and adding them to the created salesforce app (HandsMen Threads).



The above image shows the easy access of the tabs which are essentially used by the user to continue the process without any disturbance or any problem.

This system uses five custom objects to represent different parts of the business. Each object has a key field that helps store manage the data accurately. The below is breakdown of each field and stating the use of their creation.

- **HandsMen_Customer__c**
  Stores detailed information about each customer. And tracks the loyalty of the customer by grading them on total purchases they made.
  Key Fields:
  1. **Name -** Customer's full name. Store the full name of the customer after taking the first and last name.
  2. **Email** – For Sending order confirmation and updates, after placing an order.
  3. **Phone** – For contact purposes.
  4. **Loyalty Status** – A picklist showing the loyalty of the customer (Gold, Silver, bronze) which is given to the customer.
  5. **Total Purchases** – A field for tracking total purchases made.

- **HandsMen_Product__c**
  Acts as product catalog storing all the fashion items and holding the prices as well.
  1. **Name** – Holds the product name.
  2. **SKU** – A unique identifier for each product, like a QR.
  3. **Price** – Cost of the item.
  4. **Stock Quantity** – Available stock of the product.

- **HandsMen_Order__c**
  Stores customer order records and helps to store the history of order. Handling the order confirmation or rejection.
  Key Fields:
  1. **Order Number** – Unique Id for each order to track them unnoticed.
  2. **Status** – A picklist with values like pending, confirmed and rejected (not in the customer hand)
  3. **Quantity** – Number of products in the order.

4. **Total Amount** – Total Value of the order.

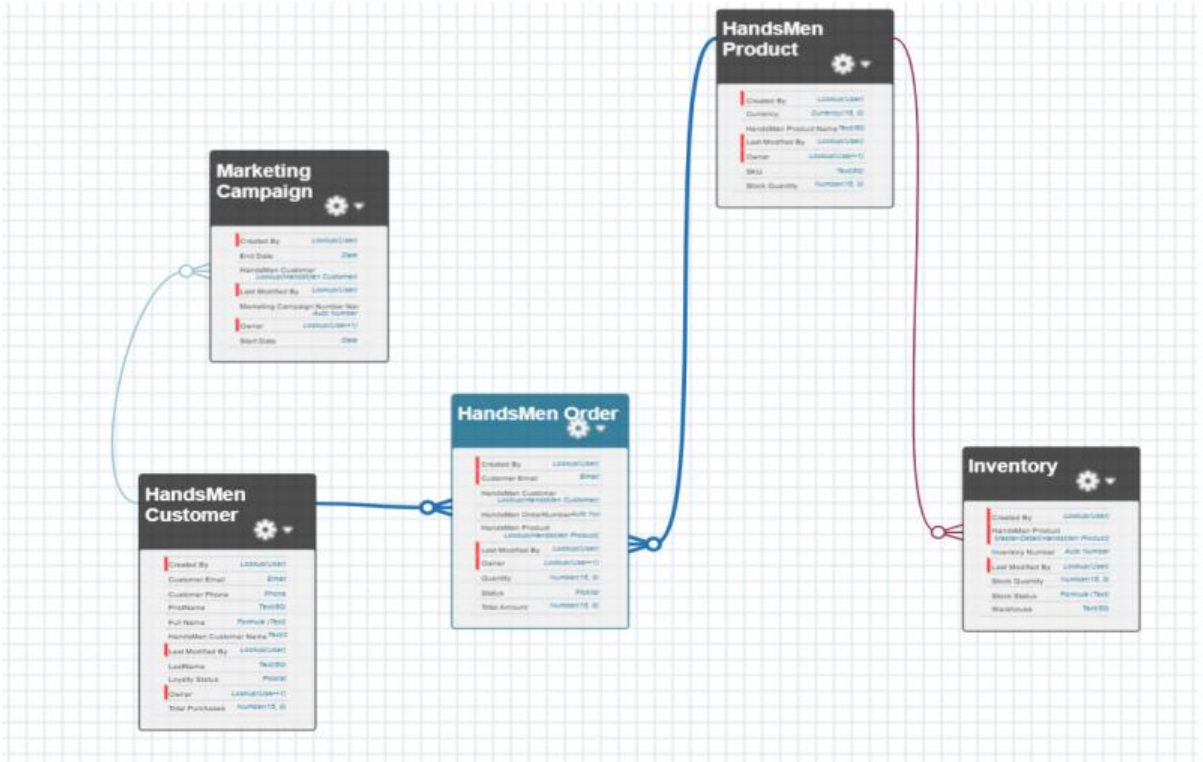
- **Inventory__c**
  Tracks the inventory levels across warehouses or location. And sending emails to the managers when they didn't meet levels.
  Key Fields:
  1. **Inventory ID** – Automatically generated record name.
  2. **Warehouse** – A text field showing which warehouse the stock is in and available.
  3. **Stock Status** – A formula which updates it if the stock is less than 5.
  4. **Stock Quantity** – Amount of stock available in that location.

- **Marketing_Campaign__c**
  Stores data related to promotional and marketing campaigns are done for a particular product.
  Key Fields:
  1. **Campaign Name** – The title of the campaign.
  2. **Start Date** – When the campaign begins.
  3. **End Date** - When the campaign ends.

The above listed fields are designed to function as a simple ecommerce business system which works on an order delivery format.

User Interface Demonstration

Designing the Schema



To Support the business logic of HandsMen Threads, a proper data schema had to be designed first. This means creating custom objects that match the real-word elements of businesslike orders, products, customers, inventory, and campaigns. Each object is carefully structured with the right fields, like order status, quantity, price, stock level, and loyalty status. Relationships are classified into two types as lookup relationships and master detail relationships. Each relationship is given considering the schema not to lose it stability. This scheme acts like the backbone of the system, making sure all the data is organized, easy to access and ready for automation.

Validation rules

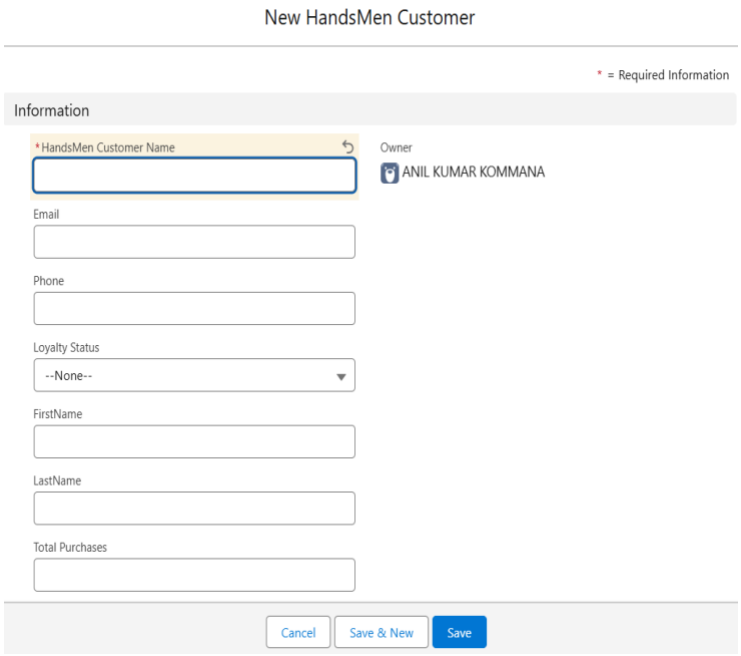
Validation rules in Salesforce are like gate keepers. When a user tries to enter wrong info in form, the rule says “no this won’t be allowed” or throw some error. We make rule using formula – if formula is true, system block saving. For example, if you want no user to put character in a phone number, you can write a rule say phone must be of only numbers with a length of 10 or 12. The app administrator uses it so garbage data will not enter into the system and break or corrupt it. It’s a simple tool but it makes big difference in keeping data clean.

In this project we created three validation rules, such as Total amount cannot be zero, inventory stock cannot be zero as well as a formula for email validation these ensure the data is given in the format that we/administrator need to.

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	NOT CONTAINS(Email, "@gmail.com")

Record Creation

To demonstrate record creation, let's take the example of adding a new customer to the system. Using the “Customer” tab, I click on “New” and fill in the required fields such as the customer’s full name, contact number, email, preferred communication channel, and loyalty status. Once saved, this customer record is automatically assigned a unique ID and becomes available for future bookings or campaigns. Next, I move to the “Order” object, where I create a new order by selecting the previously created customer, choosing the product, specifying the quantity, and setting the order status to "Confirmed." Save the booking is saved.



In Order to create an Order record, we first need a product so as for the first step after creating the user, create the product which includes product name, quantity and price per quantity. Which are basic that anybody can fill these forms for records. After that when the order is confirmed, the user will get a confirmation email, and his/her order proceed for further steps.

Business Process Automations

Email Templates and Alerts

Emails are considered one of the modes of messaging used from the beginning of the messaging era. And Salesforce adapted email alert based on a trigger and help users to notify about things they probably need. Email templates dynamically pull data from the Salesforce records, ensuring personalized communication without manual effort. This saves time and maintains a uniform brand voice across interactions. Not only do they increase customer satisfaction as well as productivity. Templates eliminate the need to draft email every time from scratch, allowing sales, marketing, and support teams to focus on high value tasks.

To provide linear communication within the salesforce app (HandsMen Threads), were configured as a part of the Salesforce automation process. These components ensure timely and professional interaction with both customers and internal users. So, for this HandsMen threads decided to create three different types of email templates for each process like order confirmation, low stock alert, and loyalty status updates.

Each one of the templates we created has an Email alert which is later used in flows for automation triggering purposes. This automation reduces the manual drafting of any email

and immediately triggers the respective Email alert when such things happen. The below images show us the template and how the email is going to be received by the user.

Dear Anil,

Your order #O-0003 has been confirmed!

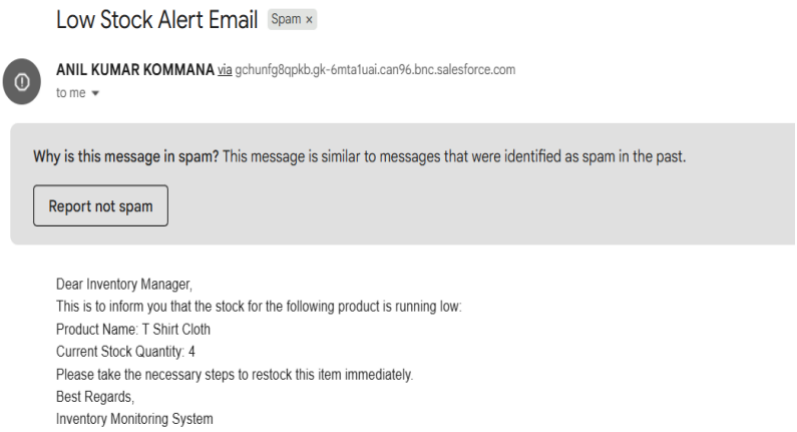
Thank you for shopping with us.

Best Regards,

Sales Team

This above template is an example of an order confirmation email with an order ID of #O-0003, which has been placed successfully and will be delivered to the person who orders, Anil after a while.

Congratulations! You are now a Bronze member and you are eligible for our Loyalty Rewards Program.
Enjoy exclusive discounts, early access to offers, and special member benefits.
Thank you for your continued Support.



The above two emails are the cause of the email templates which we created and are triggered when the conditions are met. Which is why we use flows to automate this process and ensure that no manual intervention is needed. The one on the left is for the loyalty updates, which shows the user is a bronze member of the company and the one on the right is for the Inventory manager saying that a specific item is out of stock and we need to refill it. As for the context, the employees are fictional, and the company is not real. These are only for our understanding of how this email automation works inside a complex spider web connected through many dots and known as Salesforce.

Automation with Flows.

In Salesforce, Flows are a part of the Lightning flow automation bundle. They allow users to automate complex business processes using a point and click interface without the requirement of apex code, it's like a no code platform. For the HandsMen threads project, flows were designed to handle real-time automation such as email notification, updating records, managing inventory etc. The main objective behind using or implementing flows in the project was to reduce the manual effort of staff, maintain data across objects without any problem, and reduce time for customer communication.

In the HandsMen threads we created three different flows for mixed results such as order confirmation flow for updating the order status to be confirmed and sending an email. The loyalty update flow is for updating the loyalty status and as well as for sending the user about their loyalty status. The flows which were used are given as below

Flow Name	Type	Trigger Condition	Purpose / Action Performed
Order Confirmation Flow	Record Triggered	When an order (HandsMen_Order__c) status is updated to Confirmed	Send an order confirmation email to the customer using an email template.
Loyalty Status Update Flow	Schedules flow	When the total purchases made by user meets the conditions	Runs every once in a day and updates the

			loyalty status of the customer
Inventory Stock Alert Flow	Record Triggered	When Inventory. Quantity__c falls below a value of 5	Send an internal email alert to the inventory team notifying low stock for a product.

Each flow has its own way of triggering and its own conditions to follow. For example, the loyalty status update flows. Which first gets all the record including if any duplicates and loop around each one of them, checking for the conditions like total purchases greater than 1000 less than 500 and middle to assign the status as Gold, Bronze and Silver to the customer. It's not a single process it is a group of processes that are trying to do one thing. Using flows is the best practice in Salesforce because they increase the efficiency of process and reduce time consumption.

Apex and Batch Apex

Apex is a strongly typed, object-oriented programming language developed by Salesforce. It allows developers to execute things that we can only do in the Salesforce environment. Like accessing data from the cloud, updating reports, creating reports and managing records. Apex is often considered as a similar language to java; it is like an extension to java programming language. Apex uses complex logic that cannot be achieved by using flows or visual force pages.

In the HandsMen threads project, Apex was used to handle scenarios where the operation needed to run multiple records at the same time, logic required for updating records such as Stock deduction after an order etc. Apex Enables developers to write custom triggers, classes, scheduled jobs and many more, making the life of developers as easy as possible and making it essential for large-scale enterprise applications which are built on the Salesforce.

Order Total Trigger

This is a trigger which was written to calculate the total amount of the order and enter its respective field. This apex code is designed to automatically calculate the total amount of an order before it is inserted or updated, based on the quantity purchased. The code below demonstrates how the flow works.

```
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.HandsMen_Product__c != null) {
            productIds.add(order.HandsMen_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.HandsMen_Product__c != null && productMap.containsKey(order.HandsMen_Product__c)) {
            HandsMen_Product__c product = productMap.get(order.HandsMen_Product__c);
            if (order.Quantity__c != null) {
                order.Total_Amount__c = order.Quantity__c * product.Price__c;
            }
        }
    }
}
```

The above code demonstrates how the code works. The trigger deceleration which is trigger OrderTotalTrigger on HandsMen_Order__c (before insert, before update) tells us that the code is declared to be triggered when the record is inserted into the database or updated into the database. The next step involves collecting all the product ids from the orders and loops through each order in Trigger.new (the list of new or updated orders)

```
    Set<Id> productIds = new Set<Id>();
    for (HandsMen_Order__c order : Trigger.new) {
        if (order.HandsMen_Product__c != null) {
            productIds.add(order.HandsMen_Product__c);
        }
    }
```

If the order has a product selected, the product id is added to the set and this set ensures the uniqueness and helps with efficient querying later. Now for the next step we need to get all the product prices and map to their product ids.

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

The SoQL statements of the Salesforce helps us to retrieve all of them and store them in the Map name productMap. And the final step is to calculate the total amount and add them to its field. Simple error handlings like Quantity checking and product presence ensures the code to get a coverage of 100%.

Stock Deduction Trigger

This trigger is written to update the stock of the product whenever the order is placed. The deduction of stock shows us the real-time advantage of not manually entering this data into the database. The trigger automatically deducts the stock when an order is confirmed, it runs on the order object after insert and after update, meaning the logic executes after the order record has been saved to the database. The code is shown below.

```
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.HandsMen_Product__c != null) {
            productIds.add(order.HandsMen_Product__c);
        }
    }
    if (productIds.isEmpty()) return;
    Map<Id, Inventory__c> inventoryMap = new Map<Id, Inventory__c>(
        [SELECT Id, Stock_Quantity__c, HandsMen_Product__c
         FROM Inventory__c
         WHERE HandsMen_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.HandsMen_Product__c != null) {
            for (Inventory__c inv : inventoryMap.values()) {
                if (inv.HandsMen_Product__c == order.HandsMen_Product__c) {
                    inv.Stock_Quantity__c -= order.Quantity__c;
                    inventoriesToUpdate.add(inv);
                    break;
                }
            }
        }
    }
    if (!inventoriesToUpdate.isEmpty()) {
        update inventoriesToUpdate;
    }
}
```

This code first collects all the product IDs whose order is confirmed and stores them in a set as it ensures the uniqueness of the id. In the next step the code checks if there are no products with confirmed status and exits. If there exist any products with confirmed status those will be retrieved from the database and added to the InventoryMap for easy access.

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

Store all the results of SOQL, query inside the map and loop through them to deduct the stock. For every confirmed order, finds the matching inventory record by comparing the product via lookup field. Decrease the stock quantity based on the order's quantity. Add the updated inventory to the record to a list for later update. The final step is to check whether any modifications are made and update them to the database.

Apex was used in the HandsMen Threads project to implement essential backend logic that couldn't be achieved through flows or any other. It enabled automated processes like stock deduction and total amount calculation. The use of apex helped us to reduce manual tasks for some departments and reduce time.

User Management and Security

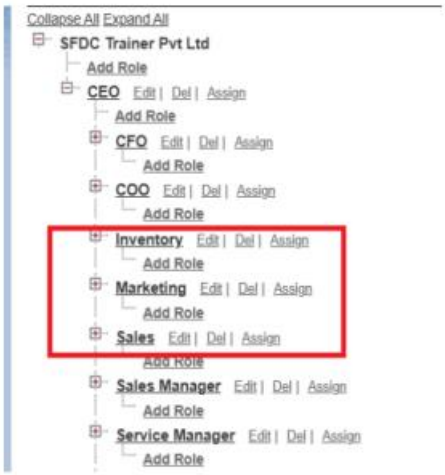
Profiles

Profiles in Salesforce define a user's base level access to objects, fields, and system functions. For this project, custom profile was created to align with real organizational roles such as "**Platform 1**". Profile determines whether a user can create, edit, read or delete any data in standard or custom object created. Adding some more profiles also allows to control tab visibility, record type access, page layout assignments, and app permissions. For example, a user with Platform 1 as his profile could view or update something in the app, considering the permission sets assigned to them.

Roles

Roles in Salesforce establish a hierarchy that determines record-level visibility and data sharing across the organization. HandsMen Threads adopted a role-based hierarchy that resembles its internal structure—from Admin to Associates. Roles were designed to support data sharing upwards, meaning managers can view their team's records, but not the other way around. For example, a sales Manager in a higher role can view all customer orders placed by their team members. This structure guarantees the reporting accuracy and facilitates collaboration across departments while maintaining security. These roles are crucial in determining record ownership visibility and ensure that sensitive records are only accessed by relevant rolesz.

For each object or sector let's say for Sales, Inventory and for marketing we have created roles for them accordingly. Each role is lower than the CEO (chief executive officer) and reports to him after any inconsistencies. And can't do anything without the permission of the CEO or from the system administrator.



The above is a picture demonstrating how the hierarchy structure works in the salesforce states that it ensures the level of access at each level of the hierarchy.

Creating Users

In HandsMen threads salesforce environment, users represent the individuals interacting with the system – each assigned specific credentials and access rights. Users include many things like Sales Executive, Warehouse staff, Inventory manager, and Marketing Team members. Each user is linked to a profile (Platform 1) and potentially one or more Roles will be given to the users depending on what job they are doing, and what responsibilities that job holds. Though this is a fictional company we have created three fictional employees having the responsibility to view and edit the data.

The three users we created are assigned three different job roles such as Sales manager, Inventory manager and marketing manager having specific job tasks that they need to complete and were also assigned some permission sets. Which we will discuss in the next stages.

Username	**Role**	**License**	**Profile**	**Title**
Niklaus Mikaelson	Sales	Salesforce Platform	Platform 1	Sales Manager
Kol Mikaelson	Inventory	Salesforce Platform	Platform 1	Inventory Manager

Jonny Roy	Marketing	Salesforce Platform	Platform 1	Marketing Manager

Permission Sets

In Salesforce, Permission Sets are a powerful way to grant additional access to users without changing their profiles. While profiles define baseline access, permission sets offer flexibility by allowing you to assign specific permissions on top of what a profile allows. Think the permission sets as "add-ons" – they are ideal when we want some permissions temporarily, selectively or for a specific task, without having to create a new profile.

In this we have created three different permission sets for three different users with three different roles, as Sales, Inventory and Marketing manager. Allowing them to have some access to the data that they can access. The table below shows the permission sets and their permission access to specific users.

Name	**Read**	**Create**	**Edit**	**View**	**Objects**	**User Assigned**
Sales Permission set	Yes	Yes	Yes	Yes	Customers and Orders	Sales manager
Inventory Permission set	Yes	No	Yes	No	Inventory and Products	Inventory Manager
Marketing Permission set	Yes	No	Yes	No	Read on Customer, Edit on campaigns	Marketing Manager

The table above gives us a detailed understanding of what permission sets to what and how they help in giving access to data for some users.

Role	**Access Level**
Sales Manager	Full Access to Customers, Orders
Inventory Manager	Read & Edit on Inventory, Products
Marketing Team	Read on Customers, Edit on Marketing Campaigns

Outcomes

The implementation of Salesforce Enhancement Threats project leads to a digital transformation of their internal business processes from order management and inventory management to customer engagement and marketing automation every function was streamlined using combination of things which we can do in Salesforce ecosystem things that require manual workload were significantly reduced data accuracy security are increased and the real time role based access system helped to give uses a certain level of access to the data within the company ensuring no data loss this project demonstrated how Salesforce can be used to suit the specific needs of fashion retail business ultimately enabling HandsMen threats to operate more efficiently and deliverer better overall customer experience Not only did this help handsome threats to embark into the salesforce technology also helped me as the creator of this app to understand how the real world problems are to be solved by using many steps like knowing the problem understanding the problem and trying to find a solution which is both efficient and user friendly So sales for V is a better opportunity for small businesses to increase their online marketing or take advantage of their online business to another level.

Conclusion

This HandsMen Threads Salesforce project was good and there's so much learning for making a custom business solution on Salesforce platform. We used declarative tools like flows, email template, validation rules and programmatic things like Apex trigger, class, and batch job. This helps us automate big process in fashion retail – like order manage, inventory control, and customer engage.

When we develop, we focus on making clean, scalable data models for keeping data good and user experience nice. We also ensured user security with profiles, roles, and permission sets. Business logics like order total calculation, stock deduction, and loyalty Status updating were automated efficiently, so no need for manual work and increase in real-time accuracy.

Project also taught us about platform limit and best practice, like governor limit, bulk process, and SOQL optimize, which very important for building an enterprise app. More importantly, it showed the real-world value of Salesforce for making operations smooth, improving customer relations, and improving decision-making through automated data processes.

Overall, this project has not only improved our technical knowledge of the Salesforce ecosystem but also strengthened our problem-solving skills workflow planning, designing an approach towards a problem, which ensures a stable solution. All of which are essential for a person to solve a real-world problem and to get into companies that focus on solving real world problems like Salesforce development or administrator.