HandsMen Threads: A Comprehensive Implementation Guide for Salesforce

Executive Summary

This report provides a comprehensive, expert-level guide to the design, development, and implementation of the "HandsMen Threads" application on the Salesforce platform. The project serves as a practical, real-world simulation of a bespoke retail management system for a men's fashion brand. The core objective of the application is to streamline and automate key business processes, creating a unified system for managing customer relationships, processing sales orders, tracking product inventory, and executing targeted marketing campaigns.

The application architecture leverages a combination of Salesforce's declarative and programmatic capabilities to build a robust and scalable solution. Key business processes automated within this project include:

- **Customer Relationship Management (CRM):** Centralizing customer data, tracking purchase history, and managing a tiered loyalty program.
- **Order Processing:** A streamlined workflow for creating, confirming, and fulfilling customer orders, with automated calculations and inventory synchronization.
- **Inventory Management:** Real-time tracking of stock levels, with automated alerts for low inventory to prevent stockouts.
- **Marketing Outreach:** Management of marketing campaigns and automated email communications for customer engagement.

The purpose of this document is to serve as an in-depth, step-by-step implementation manual. It is designed for Salesforce professionals, including developers and administrators, who wish to understand the practical application of core platform features. Beyond simple procedural instructions, this report dissects the underlying architectural decisions and highlights Salesforce best practices, explaining not just *how* each component is built, but *why* specific tools and patterns were chosen. By following this guide, readers will gain a nuanced understanding of how to

translate business requirements into a fully functional and efficient Salesforce application.

Section 1: Foundational Setup - Architecting the Digital Workspace

The successful deployment of any Salesforce application begins with the establishment of a proper development environment and a well-defined user workspace. These foundational steps are critical for ensuring the stability, scalability, and usability of the final product. This section details the initial setup of the Salesforce Developer Edition organization and the creation of the custom "HandsMen Threads" Lightning application, which serves as the primary interface for all users.

1.1 Establishing the Developer Environment

All development work should be conducted in an isolated environment to prevent any impact on live production data and users. The Salesforce Developer Edition organization is a free, full-featured environment designed specifically for this purpose.

The process begins by navigating to the Salesforce Developer signup page (https://developer.salesforce.com/signup). The signup form requires basic information, including name, email, role (select "Developer"), and a unique username. The username must be in an email format (e.g., username@company.com) but does not need to be a real, deliverable email address; it serves as the unique identifier for logging into this specific Salesforce org.

Upon submission, a verification email is sent to the provided email address. This email contains a link to set the password for the new Developer org. Once the password is set, the environment is ready. This isolated org acts as the sandbox for the entire "HandsMen Threads" project, allowing for safe experimentation, development, and testing of all objects, code, and automation before any potential deployment to a production environment.

1.2 Creating the "HandsMen Threads" Lightning Application

To provide users with a focused and efficient workspace, a custom Lightning application is created. A custom app tailors the user experience (UX) by presenting only the necessary tabs, tools, and branding relevant to the "HandsMen Threads" business operations. This avoids the clutter of standard applications like "Sales" or "Service" and improves user adoption and productivity.

The creation process is handled within the Salesforce **Setup** menu via the **App Manager**.

1. **Initiation:** From the App Manager, click the "New Lightning App" button to launch the creation wizard.
2. **App Details & Branding:** The first step is to define the app's identity. The **App Name** is set to "HandsMen Threads". A description can be added for administrative context, and a custom logo can be uploaded to reinforce the brand identity within the application.
3. **App Options:** The wizard then presents options for navigation style (Standard or Console) and device support. For this project, standard navigation is sufficient.
4. **Utility Items:** Utility items are not required for this initial build but could be added later to provide quick access to tools like a calculator or recent items from the footer bar.
5. **Navigation Items:** This is the most critical step for defining the user's workspace. The following items are selected from the "Available Items" list and moved to the "Selected Items" list to form the app's primary navigation bar:
    - **Custom Objects:** HandsMen Customers, HandsMen Orders, HandsMen Products, Inventories, Marketing Campaigns. These are the core entities of the application.
    - **Standard Objects:** Reports and Dashboards are included for analytics and business intelligence. Accounts and Contacts are included as they are standard CRM objects that may be used in future expansions of the application.
6. **User Profiles:** Access to the application is granted by assigning it to specific user profiles. For the initial setup, the app is assigned to the **System Administrator** profile, ensuring that developers and administrators can immediately access and configure it.
7. **Save and Finish:** Completing the wizard creates the "HandsMen Threads" app. It

can now be accessed from the App Launcher in the top-left corner of the Salesforce interface. The result is a clean, branded workspace containing a curated set of tabs, providing a purpose-built environment for managing the fashion retail business.

Section 2: Architecting the Data Model - The Application's Blueprint

The data model is the bedrock of any Salesforce application. It defines the structure of the database, the types of information stored, and the relationships between different data entities. A well-designed data model is intuitive, scalable, and ensures data integrity. This section details the creation of the complete data model for "HandsMen Threads," including all custom objects, fields, relationships, and dynamic formula fields that represent the core concepts of the business.

To provide a high-level overview, the following table summarizes the custom objects and their key attributes that form the application's schema.

Object API Name	Purpose	Key Fields & Data Types
HandsMen_Customer__c	Stores individual client information, purchase history, and loyalty status.	Full_Name__c (Formula), Email (Email), Loyalty_Status__c (Picklist)
HandsMen_Product__c	Represents a unique product available for sale, such as a specific t-shirt or jacket.	Price__c (Currency), SKU__c (Text)
HandsMen_Order__c	Represents a single sales transaction, linking a customer to a product.	HandsMen_Order_Number__c (Auto-Number), Status__c (Picklist), Total_Amount__c (Number)
Inventory__c	Tracks the stock level of a specific product in a warehouse.	Stock_Quantity__c (Number), Stock_Status__c (Formula)

Marketing_Campaign__c	Manages marketing initiatives, including their duration and associated customers.	Start_Date__c (Date), End_Date__c (Date)

2.1 Defining Core Business Entities (Custom Objects)

Custom objects are the database tables that store the application's unique data. For "HandsMen Threads," five custom objects are created to represent the primary business entities. This is done via the **Object Manager** in Setup.

- **HandsMen Customer (HandsMen_Customer__c):** This object stores information about the brand's clients.
  - **Label:** HandsMen Customer
  - **Plural Label:** HandsMen Customers
  - **Record Name:** HandsMen Customer Name (Standard Text)
- **HandsMen Product (HandsMen_Product__c):** This object represents the items available for sale.
  - **Label:** HandsMen Product
  - **Plural Label:** HandsMen Products
  - **Record Name:** HandsMen Product Name (Standard Text)
- **HandsMen Order (HandsMen_Order__c):** This object tracks each individual sale. A key architectural decision here is to use an auto-number for the record name to ensure every order has a unique, sequential, and user-friendly identifier.
  - **Label:** HandsMen Order
  - **Plural Label:** HandsMen Orders
  - **Record Name:** HandsMen Order Number
  - **Data Type:** Auto-Number
  - **Display Format:** O-{0000}
  - **Starting Number:** 1
- **Inventory (Inventory__c):** This object tracks stock levels for each product. It also uses an auto-number for unique identification.
  - **Label:** Inventory
  - **Plural Label:** Inventories
  - **Record Name:** Inventory Number
  - **Data Type:** Auto-Number
  - **Display Format:** I-{00000}

    - **Starting Number:** 1
- **Marketing Campaign (Marketing_Campaign__c):** This object is for managing marketing initiatives.
    - **Label:** Marketing Campaign
    - **Plural Label:** Marketing Campaigns
    - **Record Name:** Marketing Campaign Number
    - **Data Type:** Auto-Number
    - **Display Format:** MC-{00000}
    - **Starting Number:** 1

For all objects, the "Allow Reports" and "Allow Search" features are enabled to ensure that the data within these objects is reportable and searchable throughout the application.

2.2 Enhancing User Interface Accessibility (Custom Tabs)

While objects define the database structure, they are not visible to users in the UI by default. Custom tabs must be created to expose these objects within the "HandsMen Threads" application. This is done from the **Tabs** page in Setup.

For each of the five custom objects, a new tab is created. A crucial part of this process is selecting an appropriate **Tab Style**. The style assigns a unique color and icon to the tab, providing a visual cue that helps users quickly identify and navigate between different objects. The selected styles are:

- **HandsMen Customer:** People (icon)
- **HandsMen Product:** Box (icon)
- **HandsMen Order:** Shopping Cart (icon)
- **Inventory:** Building (icon)
- **Marketing Campaign:** Mail (icon)

These tabs are made visible to the relevant profiles and are automatically included in the "HandsMen Threads" app because they were selected during the app creation process.

2.3 Defining Object Attributes (Fields)

With the objects created, the next step is to define their attributes by adding custom fields. Each field has a specific data type that enforces the kind of data that can be stored (e.g., text, number, date, email). The fields are created from the **Fields & Relationships** section of each object in the Object Manager.

- **On HandsMen_Customer__c:**
  - First_Name__c: Text(60)
  - Last_Name__c: Text(60)
  - Email: Email
  - Phone: Phone
  - Loyalty_Status__c: Picklist, with values: Gold, Silver, Bronze.
  - Total_Purchases__c: Number
- **On HandsMen_Product__c:**
  - SKU__c: Text(60) (Stock Keeping Unit)
  - Price__c: Currency
- **On HandsMen_Order__c:**
  - Status__c: Picklist, with values: Pending, Confirmed, Rejection.
  - Quantity__c: Number
  - Total_Amount__c: Number
  - Customer_Email__c: Email (This field is critical for sending order confirmations).
- **On Inventory__c:**
  - Warehouse__c: Text(60)
  - Stock_Quantity__c: Number
- **On Marketing_Campaign__c:**
  - Start_Date__c: Date
  - End_Date__c: Date

Once these fields are created, the record detail page for each object becomes a functional data entry form. For example, the HandsMen Order record page will now display all its relevant fields, as shown in the target state below.

!(image_2.png)

2.4 Establishing Connections (Object Relationships)

Object relationships are the "glue" that connects the data model, creating a 360-degree view of the business. They allow records in one object to be associated with records in another. The "HandsMen Threads" application utilizes two types of relationships: Lookup and Master-Detail.

- **Lookup Relationships:** These create a loose coupling between objects.
    - **HandsMen Order to HandsMen Customer:** An order must be associated with a customer. A lookup relationship is used here because the lifecycle of an order and a customer are independent. Deleting a customer should not automatically delete their past orders, which are critical financial records.
    - **HandsMen Order to HandsMen Product:** An order must contain a product. This is also a lookup, allowing for flexibility.
- **Master-Detail Relationship:** This creates a tight, parent-child coupling between objects.
    - **Inventory (Detail) to HandsMen Product (Master):** This is a critical architectural choice. An inventory record represents the stock of a *specific* product. Its existence is entirely dependent on the product. By using a Master-Detail relationship:
        1. **Cascade-Delete:** If a HandsMen Product record is deleted (e.g., the product is discontinued), all of its associated Inventory records are automatically deleted. This ensures data integrity and prevents "orphaned" inventory records.
        2. **Security Inheritance:** The security settings (ownership and sharing) of the Inventory record are inherited from its parent HandsMen Product record, simplifying the security model.
        3. **Required Relationship:** The product field on the inventory record is inherently required.

The choice between Lookup and Master-Detail is fundamental. The project's use of both demonstrates a nuanced understanding of data architecture, applying tight coupling where parent-child dependency is logical (Inventory-Product) and loose coupling where record independence is required (Order-Customer).

2.5 Creating Dynamic Data (Formula Fields)

Formula fields dynamically compute values based on other fields in the same record or related records. They are powerful tools for displaying calculated information without requiring manual data entry or complex automation.

- **Full_Name__c on HandsMen_Customer__c:** This formula provides a user-friendly full name by combining the First_Name__c and Last_Name__c fields.
  - **Return Type:** Text
  - **Formula:** $First_Name__c & " " & Last_Name__c$
- **Stock_Status__c on Inventory__c:** This formula provides an at-a-glance status of the inventory level, making it easy for users to identify items that need restocking.
  - **Return Type:** Text
  - **Formula:** $IF(Stock_Quantity__c > 10, "Available", "Low Stock")$

These formula fields automatically update whenever the fields they reference are changed, ensuring the displayed information is always current.

Section 3: Ensuring Data Integrity with Validation Rules

Data integrity is paramount for any business application. Validation rules are a powerful mechanism for enforcing business logic and ensuring that only clean, valid data is saved into the system. They act as gatekeepers at the point of data entry, preventing users from saving records that violate predefined criteria. The "HandsMen Threads" application implements three key validation rules to maintain the quality of its data.

3.1 Validating Order Amounts

A sales order should never have a negative or zero value. To enforce this fundamental business rule, a validation rule is created on the HandsMen_Order__c object.

- **Rule Name:** Total_Amount_Validation
- **Error Condition Formula:** $Total_Amount__c <= 0$
- **Error Message:** "Please enter a correct amount."
- **Error Location:** Field: Total_Amount__c

This rule fires whenever a user attempts to save an order record where the Total

Amount is less than or equal to zero. The save operation is blocked, and the specified error message is displayed next to the field, guiding the user to correct the input.

3.2 Preventing Negative Stock Quantities

Inventory levels cannot logically be negative. A validation rule on the Inventory__c object prevents users from accidentally or intentionally setting the stock quantity to a value below zero.

- **Rule Name:** Stock_Quantity_Validation
- **Error Condition Formula:** $Stock_Quantity__c < 0$
- **Error Message:** "The inventory count can never be less than zero."
- **Error Location:** Top of Page

This rule ensures that the Stock Quantity field always contains a plausible value, protecting downstream processes like order fulfillment and reporting from being corrupted by nonsensical data.

3.3 Enforcing Email Domain Standards

For specific marketing or communication purposes, a business might need to enforce a particular email format. The project includes a validation rule on the HandsMen_Customer__c object to demonstrate this capability. The rule is designed to ensure that the customer's email address is from a specific domain (in this case, "gmail.com").

- **Rule Name:** Email_Domain_Check
- **Error Condition Formula:** $NOT(CONTAINS(Email, "gmail.com"))$
- **Error Message:** "Please fill correct Gmail."
- **Error Location:** Field: Email

This rule checks if the text in the Email field does not contain the substring "gmail.com". If it doesn't, the rule triggers, and the record cannot be saved.

It is important to recognize that while this rule serves as an excellent academic example of string function validation, a real-world implementation would require a more robust solution. This formula is brittle; it would incorrectly reject valid addresses like "user@googlemail.com" and would not validate the actual format of the email address itself. For a production environment, a more advanced formula using regular expressions (RegEx) would be the best practice. For example, a formula like $NOT(REGEX(Email, "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}"))$ would validate the structural integrity of the email address (e.g., name@domain.com) rather than just checking for a specific provider. This project's implementation serves as a foundational lesson, from which more complex and practical validation logic can be built.

Section 4: Implementing a Robust Security Framework

A multi-user application requires a sophisticated security model to ensure that users can only see and do what is appropriate for their job function. Salesforce provides a layered security framework that controls access at the object, record, and field levels. The "HandsMen Threads" project implements this framework using a combination of Profiles, Roles, and Permission Sets to create a secure and scalable system.

4.1 Defining Access Levels with Custom Profiles

Profiles are the foundation of user access in Salesforce. They determine what a user can *do* in the application, defining their object permissions (Create, Read, Edit, Delete), field permissions, app assignments, and system permissions.

A key best practice is to never modify standard profiles. Instead, they should be cloned to create custom profiles. For this project, the standard "Standard User" profile is cloned to create a new custom profile named **"Platform One"**. This profile will serve as the baseline level of access for all non-administrative users of the "HandsMen Threads" application.

After cloning, the "Platform One" profile is edited to grant the minimum necessary object permissions for the custom objects:

- **HandsMen_Product__c:** Read, Create

- **Inventory__c:** Read, Create, Edit

This establishes a "least privilege" baseline. By default, users with this profile have limited access. More specific permissions will be granted additively using Permission Sets.

4.2 Structuring the Organization with a Role Hierarchy

While profiles control what users can *do*, roles control what users can *see*. The Role Hierarchy is designed to mirror a company's organizational chart. Users at a higher level in the hierarchy can view, edit, and report on all data owned by users below them.

For "HandsMen Threads," a simple hierarchy is established under the CEO:

- CEO
  - Sales
  - Inventory
  - Marketing

This structure means that a user in the "CEO" role can see all the data created by users in the "Sales," "Inventory," and "Marketing" roles. This vertical data sharing is automatic and is a cornerstone of Salesforce's record access model.

4.3 Onboarding Personnel (Users)

With profiles and roles defined, individual user accounts can be created. Each user is assigned one profile and one role, which together define their access rights. Three users are created to represent employees in different departments:

- **User 1 (Sales):**
  - **Name:** Nicholas Miklson
  - **Profile:** Platform One
  - **Role:** Sales
- **User 2 (Inventory):**
  - **Name:** Co Miklson

    - **Profile:** Platform One
    - **Role:** Inventory
- **User 3 (Marketing):**
    - **Name:** Daniel Miklson
    - **Profile:** Platform One
    - **Role:** Marketing

Each user is also assigned a "Salesforce" User License, which is a prerequisite for assigning the "Platform One" profile.

4.4 Granting Granular Permissions (Permission Sets)

The modern best practice for Salesforce security is to use a combination of a minimal-access profile and additive permission sets. A permission set is a collection of settings and permissions that can be used to grant additional access to users without changing their profile. This model is far more scalable and manageable than creating and maintaining dozens of unique profiles.

"HandsMen Threads" utilizes this pattern by creating three permission sets, each tailored to a specific job function:

- **Sales Permission Set:** Grants the permissions needed by the sales team.
    - **Object Permissions:** Full access (Read, Create, Edit, Delete) on HandsMen_Customer__c and HandsMen_Order__c.
    - **Assigned User:** Nicholas Miklson
- **Inventory Permission Set:** Grants permissions for managing inventory and products.
    - **Object Permissions:** Read and Edit access on Inventory__c and HandsMen_Product__c.
    - **Assigned User:** Co Miklson
- **Marketing Permission Set:** Grants permissions required for marketing activities.
    - **Object Permissions:** Read access on HandsMen_Customer__c and Edit access on Marketing_Campaign__c.
    - **Assigned User:** Daniel Miklson

By using this Profile + Permission Set model, the application maintains a clean and flexible security architecture. If a new permission is needed for a specific user, a new permission set can be created and assigned without disrupting the baseline profile shared by many users. This approach is a hallmark of a well-architected Salesforce implementation.

Section 5: Automating Business Communications

Effective and timely communication is crucial for customer satisfaction and internal operations. Salesforce allows for the creation of branded, reusable email assets that can be integrated into automated processes. This section details the creation of the email letterhead, templates, and alerts that power the communication strategy for "HandsMen Threads."

5.1 Branding Communications (Classic Letterhead)

A letterhead provides a consistent branding wrapper for HTML emails, defining the logo, colors, and contact information in the header and footer. For this project, a Classic Letterhead is created to standardize the look and feel of customer-facing emails.

1. **Creation:** From Setup, navigate to **Classic Letterhead** and create a new one.
2. **Properties:** The letterhead is named "HandsMen Threads" and marked as "Available for Use."
3. **Branding:** A logo can be uploaded to the header, and background colors can be set for both the header and footer sections. For this project, a simple red color is used to visually frame the email content.

This letterhead will be applied to all HTML email templates, ensuring a professional and consistent brand presentation.

5.2 Crafting Communication Blueprints (Classic Email Templates)

Email templates are pre-written email bodies that can include merge fields to personalize the content with data from Salesforce records. "HandsMen Threads" requires three distinct templates for different business scenarios.

- **Order Confirmation (HTML):** This template is sent to customers when their order is confirmed. Being an HTML template, it can use the "HandsMen Threads" letterhead for branding. The body includes merge fields to pull in the customer's name and order number, creating a personalized message.
- **Low Stock Alert (Text):** This is an internal-facing alert sent to inventory managers. As it is a simple, functional notification, a plain Text template is sufficient. It does not use the letterhead but includes merge fields for the product name and current stock quantity.
- **Loyalty Program Email (HTML):** This template is used to notify customers of their new status in the loyalty program. It is an HTML template that leverages the letterhead and includes personalized content about their membership level.

Modernization Note: Classic vs. Lightning Email Templates

The project utilizes Classic Email Templates. While functional, it is important to note that for new development, **Lightning Email Templates** are the modern standard. Lightning templates offer a more intuitive drag-and-drop builder, support for responsive design, and are better integrated into the Lightning Experience. While the principles of using merge fields and templates remain the same, developers and administrators should favor Lightning Email Templates for future projects to take advantage of the enhanced feature set and superior user experience.

5.3 Defining Triggers for Communication (Email Alerts)

An Email Alert is the mechanism that binds an email template to a specific object and defines the recipients. It is the final piece of the communication puzzle, making the templates actionable. Three email alerts are configured to correspond with the three templates.

- **Order Confirmation Email Alert:**
  - **Object:** HandsMen_Order__c
  - **Email Template:** Order Confirmation
  - **Recipient Type:** Email Field

    - **Recipient:** The Customer_Email__c field on the HandsMen_Order__c record. This is a crucial configuration that ensures the email is sent to the actual customer who placed the order. To make this possible, the Customer_Email__c field was added to the order object specifically for this purpose.
- **Low Stock Alert:**
    - **Object:** Inventory__c
    - **Email Template:** Low Stock Alert
    - **Recipient Type:** User
    - **Recipient:** A specific internal user (e.g., the inventory manager, represented by the user "Co Miklson").
- **Loyalty Program Email Alert:**
    - **Object:** HandsMen_Customer__c
    - **Email Template:** Loyalty Program Email
    - **Recipient Type:** Email Field
    - **Recipient:** The Email field on the HandsMen_Customer__c record.

With these email alerts configured, the application now has a complete set of communication tools ready to be triggered by automation, such as the Salesforce Flows detailed in the next section.

Section 6: Advanced Automation with Salesforce Flow

Salesforce Flow is the platform's premier declarative automation tool, enabling the creation of sophisticated business processes without writing code. Flow can be used to guide users through screens, launch logic based on record changes, or run processes on a schedule. "HandsMen Threads" leverages Flow to automate several critical, time-sensitive business operations, demonstrating the tool's versatility.

6.1 Real-Time Order Processing (Order Confirmation Flow)

When a customer's order is confirmed, they should receive an email notification immediately. This is a real-time requirement perfectly suited for a Record-Triggered Flow.

!(image_5.png)

The flow is configured as follows:

- **Trigger:** The flow is configured to run on the **HandsMen Order** object. It triggers whenever **a record is updated**.
- **Entry Criteria:** The flow should not run on every update, only on the specific update that matters. The condition is set to **Status__c Equals "Confirmed"**. Furthermore, the flow is optimized to run **only when a record is updated to meet the condition requirements**. This is a critical setting that ensures the email is sent once when the status changes to "Confirmed," and not every subsequent time a confirmed order is edited for other reasons.
- **Action:** The flow's only action is to call the **"Order Confirmation" Email Alert** that was previously created. The flow automatically passes the context of the triggering HandsMen Order record to the email alert, which then uses the Customer_Email__c field to send the branded confirmation email to the correct recipient.

This flow provides instant, automated communication, enhancing the customer experience and reducing manual work for the sales team.

6.2 Proactive Inventory Management (Low Stock Alert Flow)

Preventing stockouts is crucial for any retail business. This flow proactively monitors inventory levels and sends an internal alert when stock for a product runs low.

!(image_6.png)

The flow's configuration is:

- **Trigger:** The flow runs on the **Inventory** object and triggers whenever **a record is created or updated**.
- **Entry Criteria:** The condition for this flow to execute is **Stock_Quantity__c Less Than 5**. This means any time an inventory record is created with a low quantity or updated (e.g., due to a sale) to a quantity below 5, the flow will fire.
- **Action:** The flow calls the **"Low Stock Alert" Email Alert**. This sends the plain-text email to the designated internal user, notifying them of the specific product and its low stock level, prompting them to take restocking action.

This automation acts as an early warning system, helping the business maintain healthy inventory levels and avoid lost sales.

6.3 Automated Customer Engagement (Loyalty Status Scheduled Flow)

Not all business processes are real-time. Some, like recalculating customer loyalty tiers based on total purchases, are better suited for batch processing. A Scheduled-Triggered Flow is the ideal tool for this requirement. It runs automatically at a specified time without any user interaction.

The loyalty status flow is designed to run daily and update all customer records.

- **Trigger:** The flow is configured as a **Scheduled-Triggered Flow**, set to run once every day (e.g., at 12:00 AM).
- **Logic:**
    1. **Get Records Element:** The flow begins by fetching all HandsMen_Customer__c records.
    2. **Loop Element:** It then iterates through the collection of customer records, processing them one by one.
    3. **Decision Element:** Inside the loop, a decision element checks the value of the Total_Purchases__c field for the current customer record. It has multiple outcomes:
        - **Gold:** $Total_Purchases__c > 1000$
        - **Bronze:** $Total_Purchases__c <= 500$
        - **Silver:** This is the default outcome if neither of the other conditions is met.
    4. **Update Records Element:** Based on the outcome of the decision, an Update Records element updates the Loyalty_Status__c picklist field on the current customer record to the corresponding value ("Gold," "Bronze," or "Silver").
    5. **Action Element:** After updating the status, the flow calls the **"Loyalty Program" Email Alert**, sending the customer a branded email congratulating them on their new loyalty status.

By using a scheduled flow, the system efficiently processes all customers in a single batch operation during off-peak hours, ensuring the loyalty program is always up-to-date without impacting system performance during the business day. The contrast between the real-time record-triggered flows and this scheduled flow highlights the importance of choosing the right automation trigger for the specific business context.

Section 7: Programmatic Logic with Apex

While declarative tools like Flow are powerful, some business requirements involving complex calculations, sophisticated data manipulation, or high-performance transaction control are better addressed with Apex, Salesforce's proprietary, strongly-typed, object-oriented programming language. "HandsMen Threads" leverages Apex for critical back-end logic that requires the precision and performance of code.

To help understand when to use which tool, the following table compares the automation strategies used in the project.

Use Case	Implemented Tool	Rationale / Best Practice
Update Order Total	Apex Trigger (before context)	Apex before trigger is the most efficient method for performing calculations and updating fields on the *same record* before it is saved. It avoids an extra DML operation, improving performance.
Deduct Stock	Apex Trigger (after context)	An after trigger is required because the logic needs to update a *separate, related record* (Inventory). This can only be done after the Order record has been successfully saved and has an ID.
Update Loyalty Status	Scheduled Flow	This is a non-urgent, batch process that needs to run on a schedule across all customer records. A Scheduled Flow is the ideal

		declarative tool for this, avoiding the complexity of writing a schedulable Apex class.

7.1 Calculating Order Totals on the Fly (OrderTotalTrigger)

When an order is created or updated, the total amount should be calculated automatically based on the product's price and the quantity ordered. An Apex trigger operating in the before insert and before update events is the most efficient way to achieve this.

!(image_4.png)

Code Analysis:

Apex

```
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>(

    );

    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 = product.Price__c * order.Quantity__c;
            }
        }
    }
}
```

1. **Bulkification:** The code first collects all unique HandsMen_Product__c IDs from the orders being processed (Trigger.new) into a Set. This is a critical bulkification pattern.
2. **Single SOQL Query:** It then performs a single SOQL query to retrieve all necessary product records at once, storing them in a Map for efficient lookup. This prevents placing SOQL queries inside a loop, which would quickly exceed governor limits.
3. **before Context Efficiency:** The code iterates through the orders again, calculates the Total_Amount__c, and assigns the value directly to the order record in the Trigger.new collection. Because this happens in a before trigger, the change is part of the original transaction. No extra update statement is needed, making it highly performant.

7.2 Synchronizing Orders and Inventory (StockDeductionTrigger)

When an order is confirmed, the corresponding quantity must be deducted from the product's inventory. This logic involves updating a separate, related record (Inventory__c), which mandates the use of an after insert or after update trigger.

!(image_3.png)

Code Analysis:

Apex

```
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>(

    );

    // Additional logic to map product ID to inventory record
    //...

    for (HandsMen_Order__c order : Trigger.new) {
        //... logic to find the correct inventory record
        //... and deduct order.Quantity__c from inventory.Stock_Quantity__c
    }

    update inventoryMap.values();
}
```

1. **after Context:** The trigger runs after the order has been saved. This is necessary because the Status__c must be finalized as "Confirmed" in the database before stock is deducted.
2. **Condition Check:** The code first checks if the order's status is "Confirmed." The stock deduction logic only runs for confirmed orders.
3. **Querying Related Data:** It queries the Inventory__c records related to the products in the confirmed orders.
4. **Data Manipulation:** The core logic (partially shown in the image) involves iterating through the confirmed orders, finding the matching inventory record from the map, subtracting the order.Quantity__c from the inventory.Stock_Quantity__c, and preparing the inventory records for an update.
5. **Single DML Statement:** Finally, a single update DML statement is called on the collection of modified inventory records. This adheres to the bulkification best

practice of having only one DML statement outside the loop.

7.3 Scalable Data Processing (InventoryBatchJob)

For processing very large data sets (thousands or millions of records), standard triggers can hit governor limits. Batch Apex is the solution for asynchronous, long-running jobs that can process records in manageable chunks. The project includes a Batch Apex class, InventoryBatchJob, to demonstrate this capability.

A Batch Apex class implements the Database.Batchable interface, which has three methods:

1. **start():** This method is called once at the beginning of the job. It collects the records to be processed by returning a Database.QueryLocator or an Iterable. For example, it could query for all Inventory__c records with low stock.
2. **execute():** This is the core method where the processing logic occurs. It is called for each chunk of records (the default size is 200). This is where you would update fields, make callouts, or perform other business logic.
3. **finish():** This method is called once after all chunks have been processed. It is typically used for sending a summary email or kicking off another process.

The batch job is invoked from the Developer Console's Execute Anonymous window using a command like:
Database.executeBatch(new InventoryBatchJob());
This schedules the job to run asynchronously, processing all targeted records reliably and efficiently without impacting real-time user operations.

Conclusion and Final Project Showcase

The "HandsMen Threads" application successfully demonstrates the power and flexibility of the Salesforce platform by translating a set of real-world business requirements into a cohesive, automated, and scalable solution. By strategically combining declarative tools and programmatic code, the application provides a comprehensive system for managing a modern retail operation. Each component, from the foundational data model to the complex Apex triggers, plays a specific and crucial role in the overall architecture.

The true value of the application is realized when observing the end-to-end business process in action:

1. **Customer Onboarding:** A new customer, "Harsh," is created in the system. The record is immediately visible in the HandsMen Customers list view, serving as the central repository for all client data.
2. **Order Placement:** An order is created for Harsh. The user selects the customer and a product (e.g., "t-shirt"). Upon entering a quantity, the OrderTotalTrigger fires instantly, calculating the Total Amount before the record is even saved.
3. **Order Confirmation and Fulfillment:** The order status is updated to "Confirmed." This single change sets off a cascade of automated events:
    - The **Order Confirmation Flow** is triggered, immediately sending a professionally branded confirmation email to the customer's address on file.
    - Simultaneously, the StockDeductionTrigger fires, locating the inventory record for the "t-shirt" and reducing the Stock_Quantity__c by the amount ordered. This ensures inventory levels are always synchronized with sales in real-time.
4. **Proactive Management:** If the stock deduction causes the inventory level to drop below the predefined threshold, the **Low Stock Alert Flow** is triggered, sending an immediate notification to the inventory management team to initiate a restocking process.
5. **Ongoing Engagement:** At the end of each day, the **Loyalty Status Scheduled Flow** runs, analyzing Harsh's total purchase history. It automatically updates his Loyalty_Status__c and sends a targeted email, fostering customer engagement and rewarding brand loyalty without any manual intervention.

This narrative showcases how individual components—objects, fields, flows, and triggers—work in concert to create a seamless and intelligent system. The "HandsMen Threads" project serves as a powerful testament to how Salesforce can be tailored to meet unique business needs, automate complex processes, and ultimately drive efficiency and growth. The architectural patterns and best practices employed, such as the Profile/Permission Set security model, the deliberate choice between Lookup and Master-Detail relationships, and the strategic use of before vs. after triggers, provide a robust blueprint for building high-quality, enterprise-grade applications on the platform.