Medical Inventory Management

College Name: Bishop Appasamy College of Arts and Science
College Code: BRU3G
Team ID: NM2025TMID20473

Team Members:
- Team Leader: Pranesh S – pranesh8506s@gmail.com
- Member 1: Sakthi Dasan V – sakthidasan.b07@gmail.com
- Member 2: Indira Sidhu L – lssithu2005@gmail.com
- Member 3: Jenslin Samuel J – jenslinsamuel66@gmail.com

1. Introduction

The Medical Inventory Management System is a comprehensive Salesforce application designed to streamline and manage various operational aspects of medical inventory. It efficiently maintains supplier details, manages purchase orders, tracks product details and transactions, and monitors product expiry dates. By implementing this system, operational efficiency, data accuracy, and reporting capabilities are significantly improved.

1.1 Project Overview

This project aims to develop a Salesforce-based Medical Inventory Management application for managing suppliers, purchase orders, products, inventory transactions, and reporting. The solution is designed for colleges and medical institutions to automate the traditional inventory process, ensuring transparency and real-time monitoring.

1.2 Purpose

The purpose of this project is to:
- Provide a centralized system for managing medical inventory.
- Improve efficiency by automating purchase orders and tracking supplier details.
- Reduce errors by validating critical fields such as expected delivery dates.
- Generate reports and dashboards for decision-making and performance monitoring.

2. Development Phase

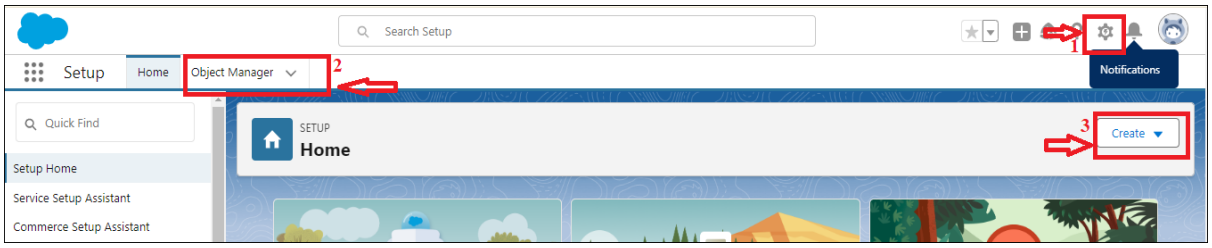
2.1 Creating Developer Account

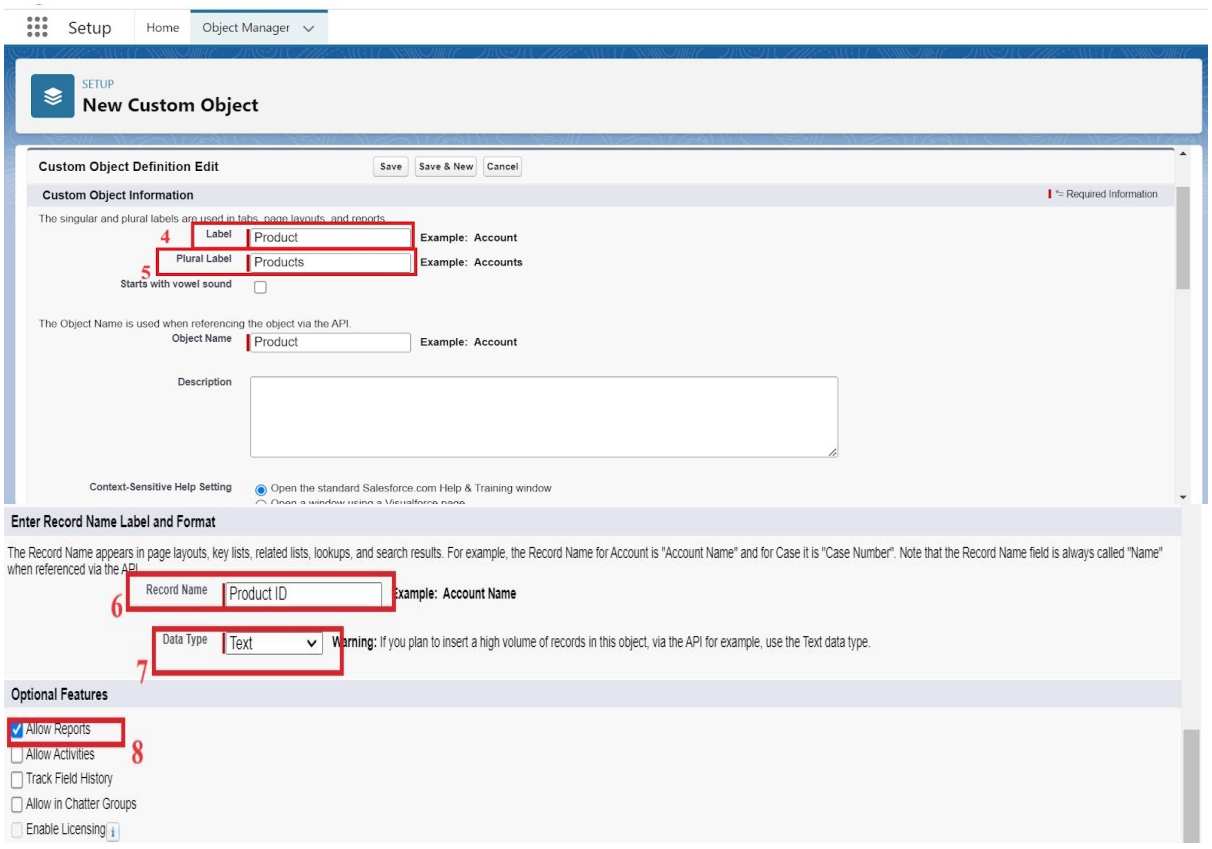
A Salesforce Developer Account was created using the official website: https://developer.salesforce.com

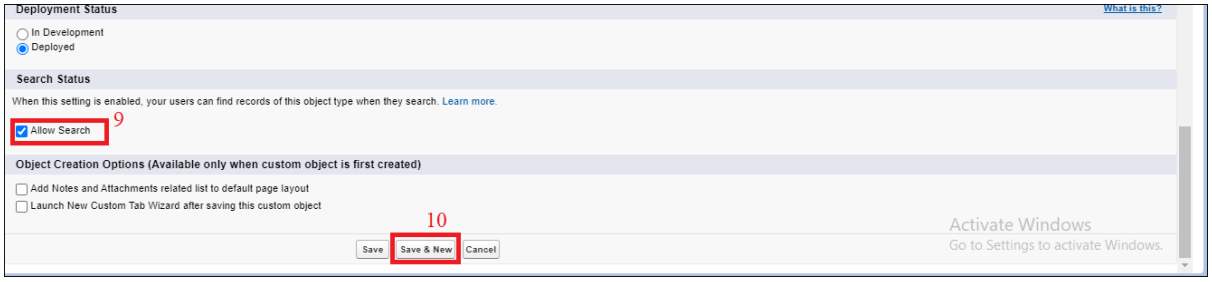
2.2 Objects Creation

Custom objects created:

- Product
- Purchase Order
- Order Item
- Inventory Transaction
- Supplier

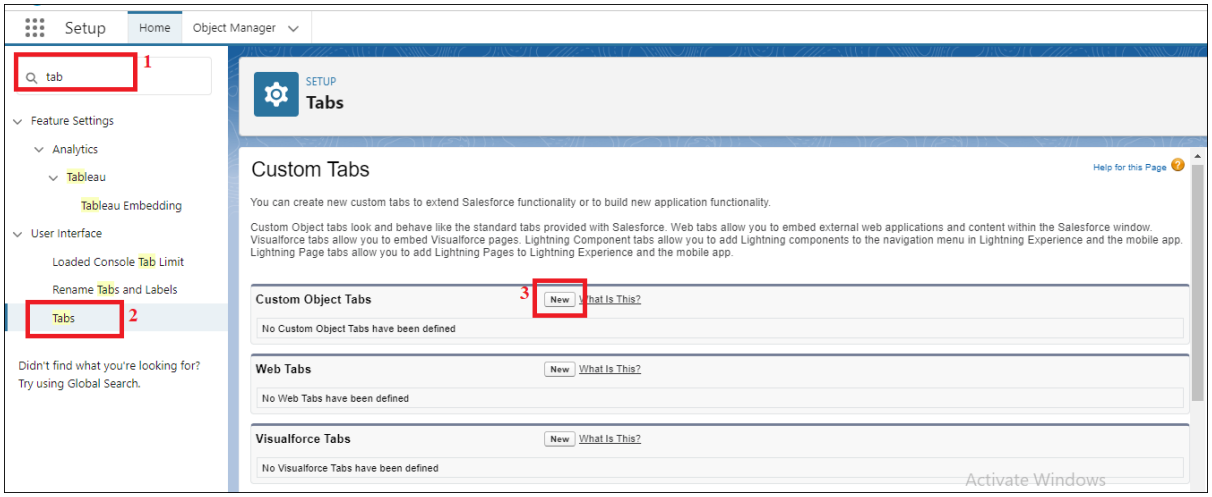


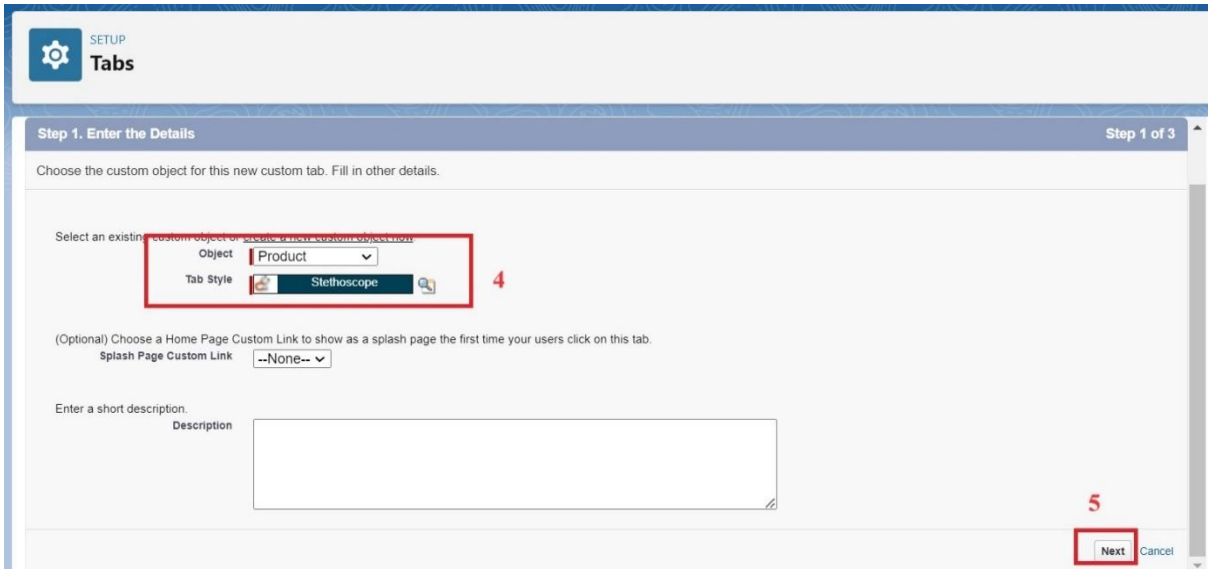


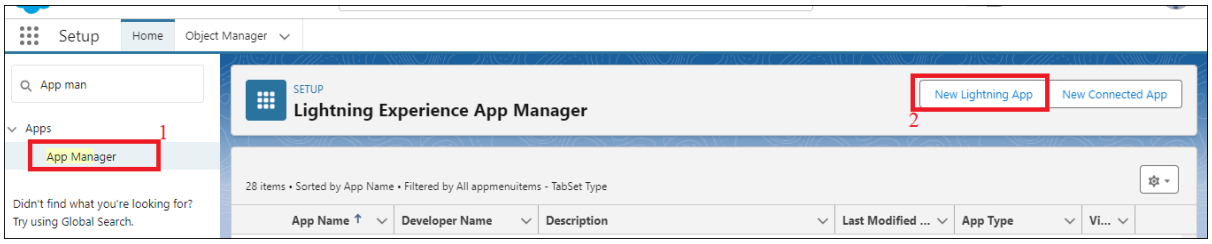


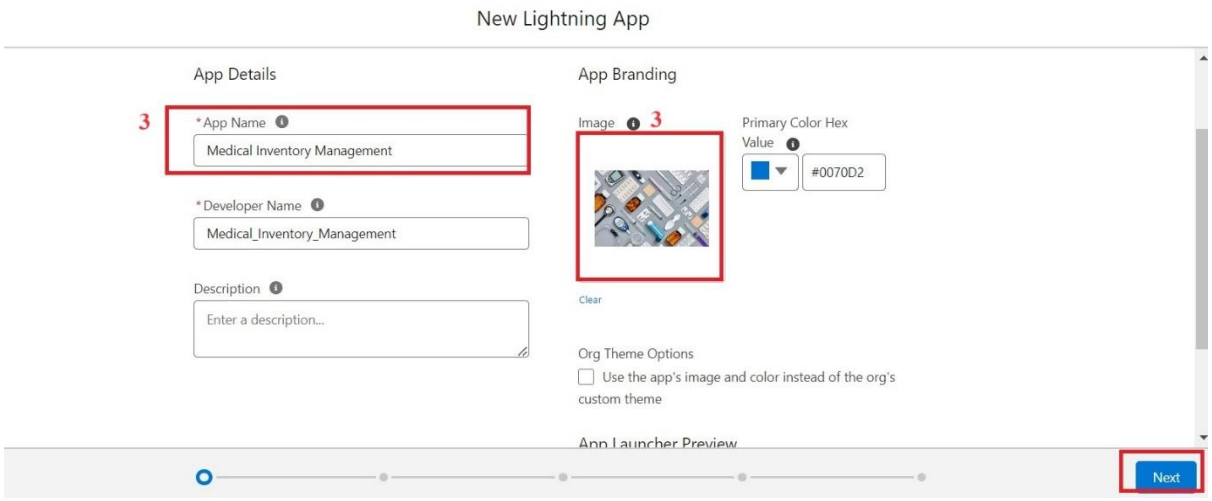
2.3 Tabs and Lightning App

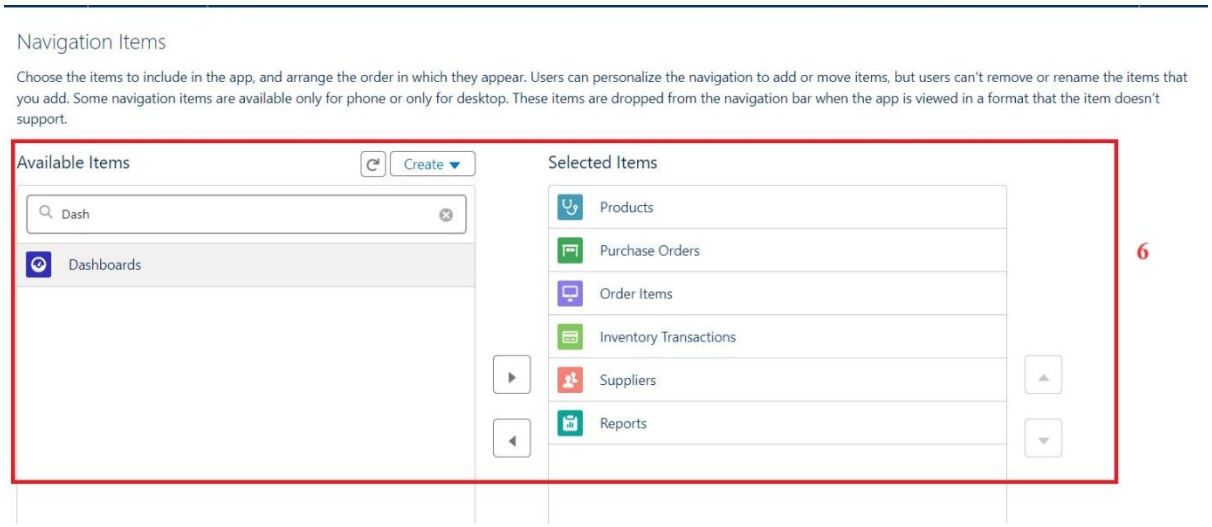
- Tabs were created for each object.
- A Lightning App named “Medical Inventory Management” was built.

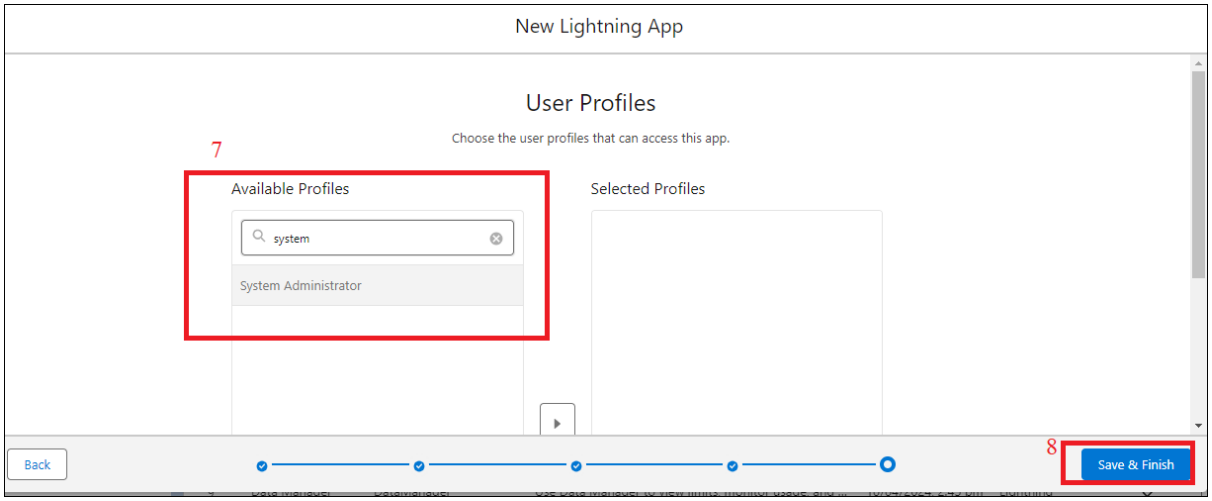



SETUP
Tabs
Step 1. Enter the Details
Step 1 of 3
Choose the custom object for this new custom tab. Fill in other details.
Object
Product
Tab Style
Stethoscope
4
(Optional) Choose a Home Page Custom Link to show as a splash page the first time your users click on this tab.
Splash Page Custom Link
--None--
Enter a short description.
Description
5
Next
Cancel


Setup
Home
Object Manager
App man
Apps
1
App Manager
Didn't find what you're looking for?
Try using Global Search.
SETUP
Lightning Experience App Manager
New Lightning App
New Connected App
2
28 items • Sorted by App Name • Filtered by All appmenuitems - TabSet Type
App Name
Developer Name
Description
Last Modified ...
App Type
Vi...

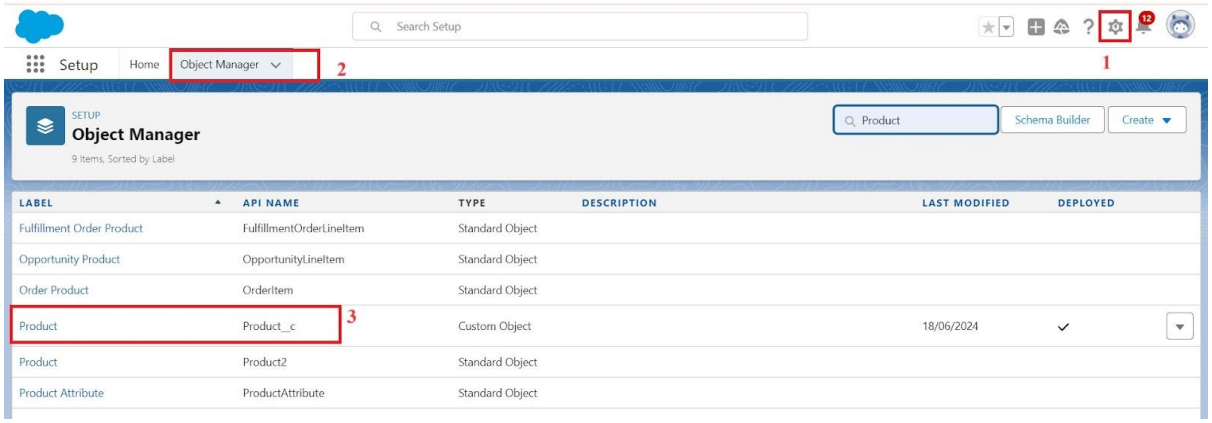

New Lightning App
App Details
3
*App Name
Medical Inventory Management
*Developer Name
Medical_Inventory_Management
Description
Enter a description...
App Branding
Image
3
Clear
Primary Color Hex Value
#0070D2
Org Theme Options
Use the app's image and color instead of the org's custom theme
App Launcher Preview
Next

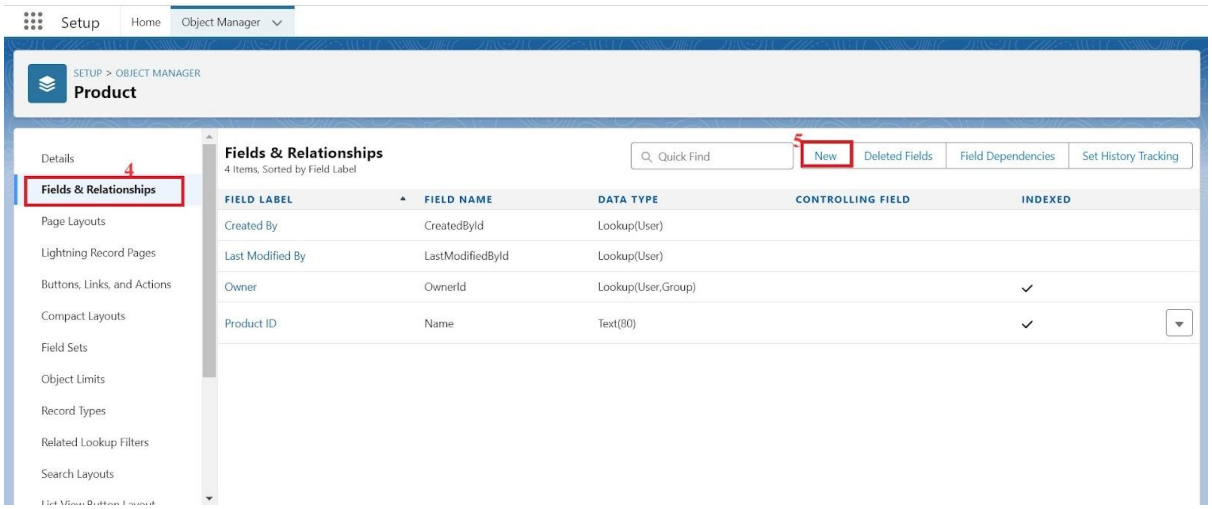


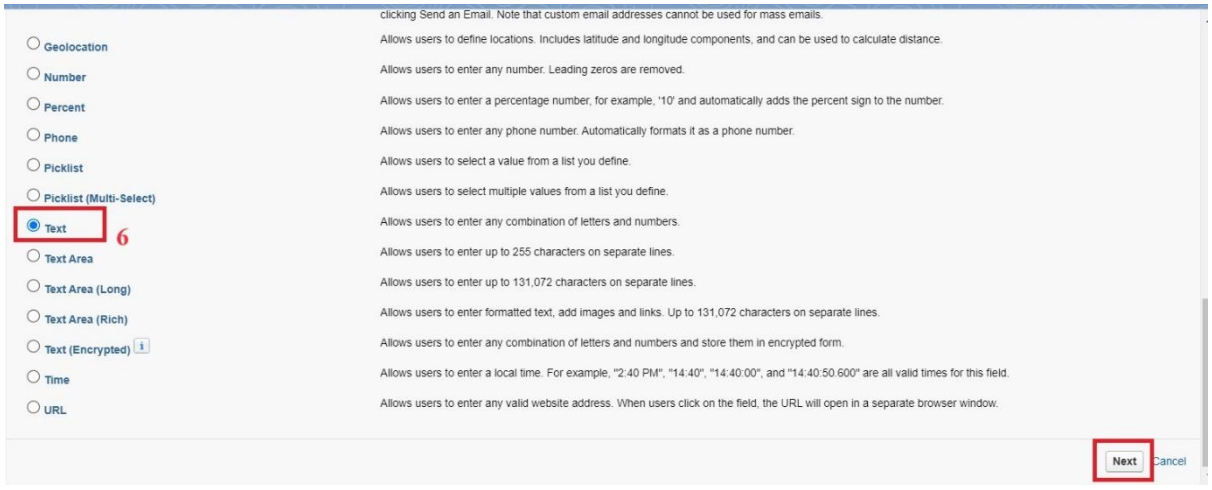


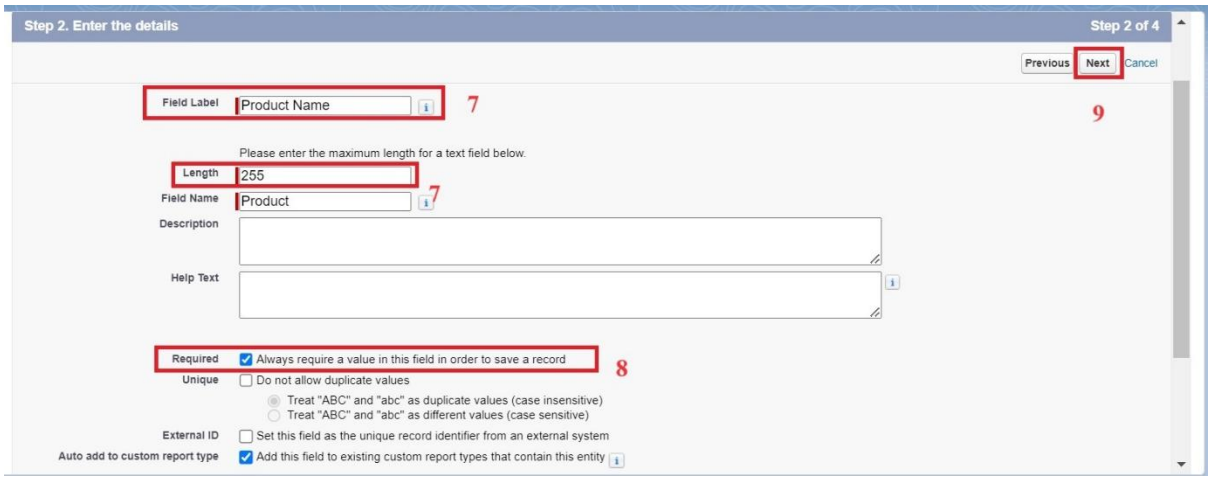
2.4 Fields and Page Layouts

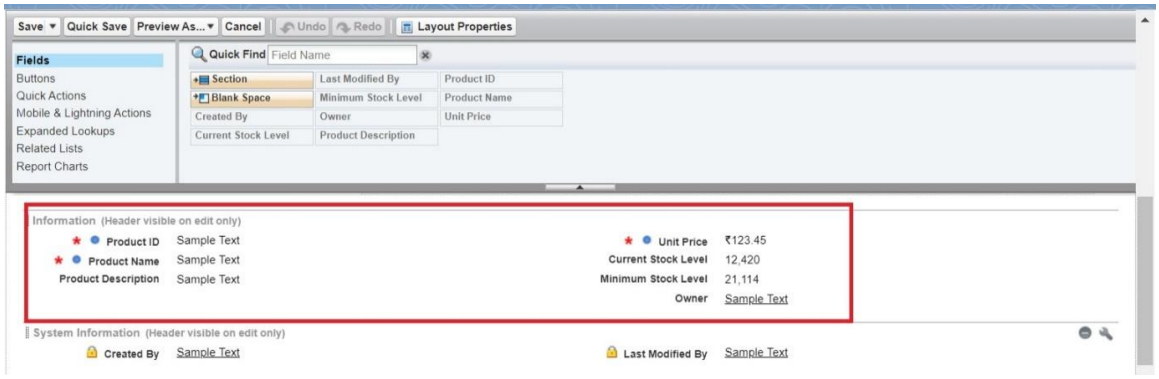
- Fields such as Product Name, Unit Price, Current Stock Level were created.
- Layouts were arranged for better usability.


Search Setup
Setup
Home
Object Manager
2
1
SETUP
Object Manager
9 items, Sorted by Label
Product
Schema Builder
Create
LABEL
API NAME
TYPE
DESCRIPTION
LAST MODIFIED
DEPLOYED
Fulfillment Order Product
FulfillmentOrderLineItem
Standard Object
Opportunity Product
OpportunityLineItem
Standard Object
Order Product
OrderItem
Standard Object
Product
Product__c
3
Custom Object
18/06/2024
Product
Product2
Standard Object
Product Attribute
ProductAttribute
Standard Object


Setup
Home
Object Manager
SETUP > OBJECT MANAGER
Product
Details
4
Fields & Relationships
Page Layouts
Lightning Record Pages
Buttons, Links, and Actions
Compact Layouts
Field Sets
Object Limits
Record Types
Related Lookup Filters
Search Layouts
Fields & Relationships
4 Items, Sorted by Field Label
Quick Find
5
New
Deleted Fields
Field Dependencies
Set History Tracking
FIELD LABEL
FIELD NAME
DATA TYPE
CONTROLLING FIELD
INDEXED
Created By
CreatedById
Lookup(User)
Last Modified By
LastModifiedById
Lookup(User)
Owner
OwnerId
Lookup(User,Group)
Product ID
Name
Text(80)

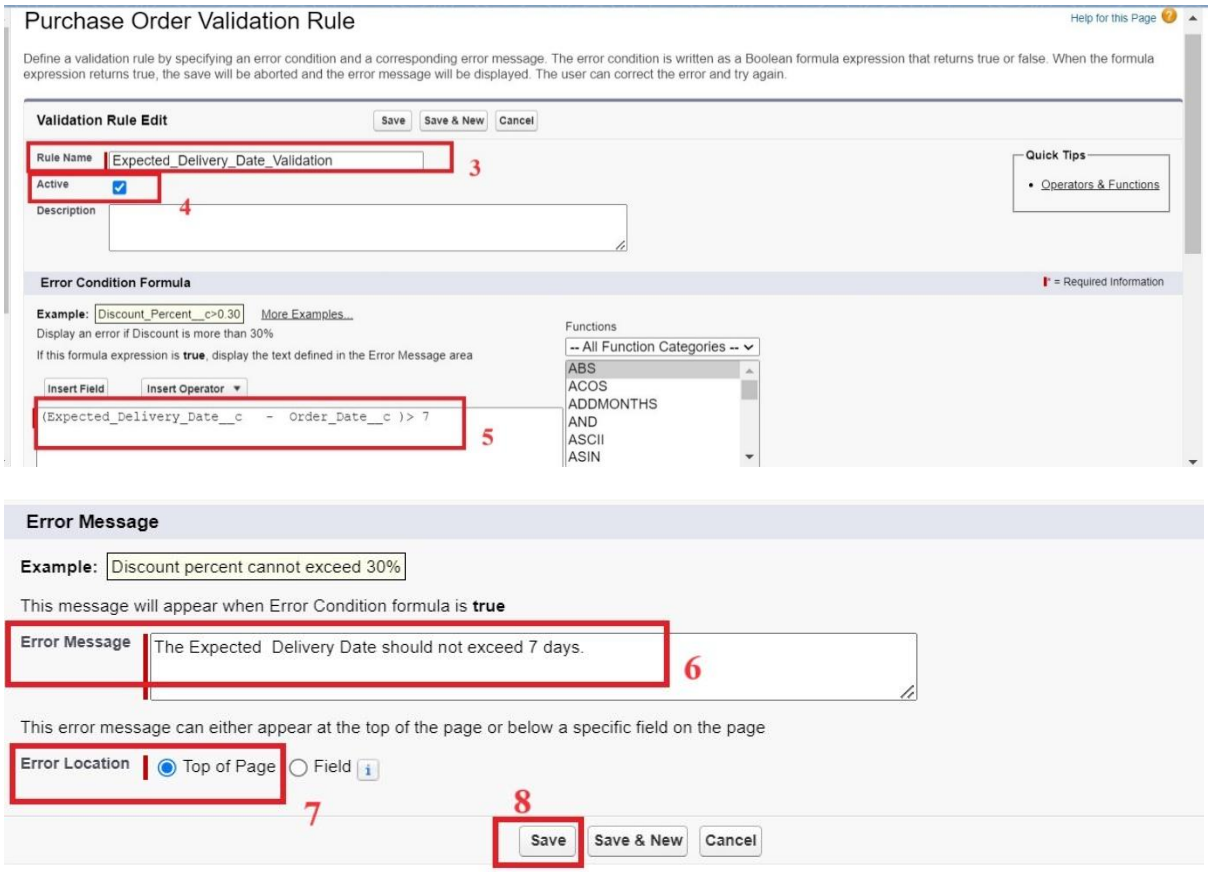

clicking Send an Email. Note that custom email addresses cannot be used for mass emails.
Geolocation
Allows users to define locations. Includes latitude and longitude components, and can be used to calculate distance.
Number
Allows users to enter any number. Leading zeros are removed.
Percent
Allows users to enter a percentage number, for example, '10' and automatically adds the percent sign to the number.
Phone
Allows users to enter any phone number. Automatically formats it as a phone number.
Picklist
Allows users to select a value from a list you define.
Picklist (Multi-Select)
Allows users to select multiple values from a list you define.
Text
6
Allows users to enter any combination of letters and numbers.
Text Area
Allows users to enter up to 255 characters on separate lines.
Text Area (Long)
Allows users to enter up to 131,072 characters on separate lines.
Text Area (Rich)
Allows users to enter formatted text, add images and links. Up to 131,072 characters on separate lines.
Text (Encrypted)
Allows users to enter any combination of letters and numbers and store them in encrypted form.
Time
Allows users to enter a local time. For example, "2:40 PM", "14:40", "14:40:00", and "14:40:50.600" are all valid times for this field.
URL
Allows users to enter any valid website address. When users click on the field, the URL will open in a separate browser window.
Next
Cancel





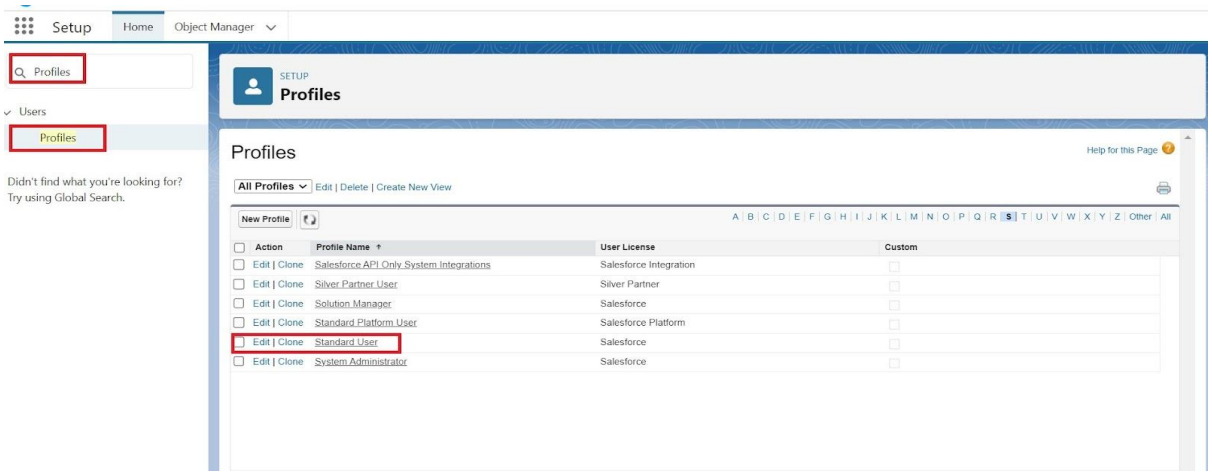
2.5 Validation Rules

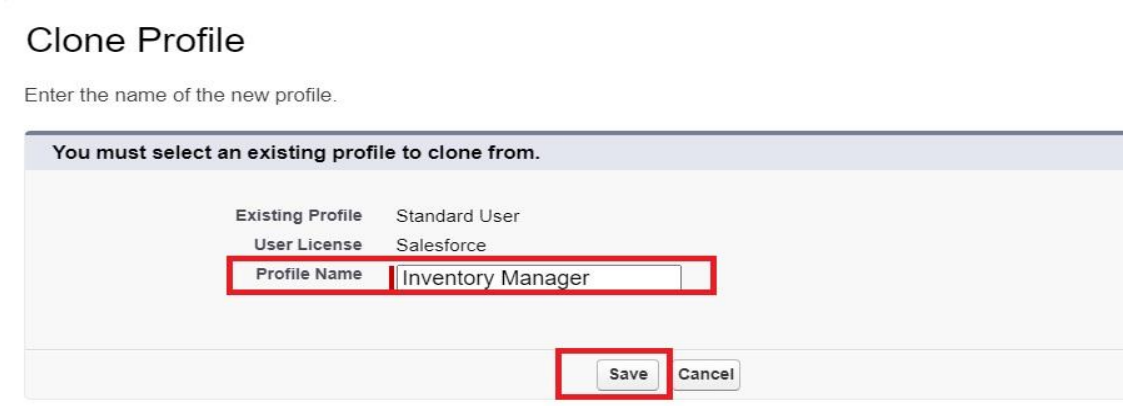
Validation was implemented for Purchase Orders:
(Expected_Delivery_Date__c - Order_Date__c) > 7
Error: “The Expected Delivery Date should not exceed 7 days.”



2.6 Profiles, Roles, and Permission Sets

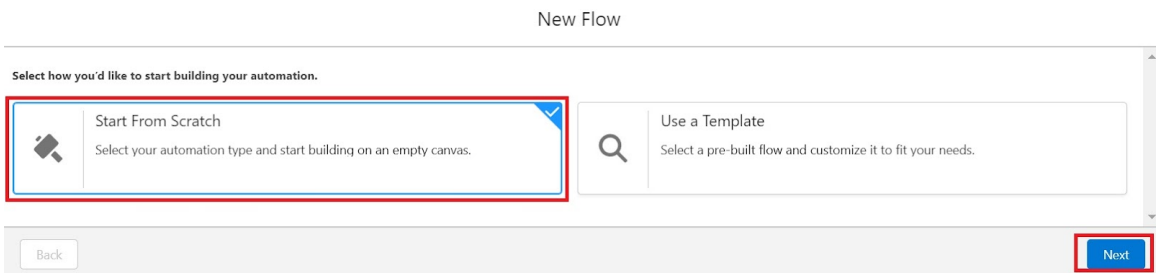
- Created a profile: Inventory Manager.
- Created a role: Purchasing Manager.
- Created permission sets for Purchase Manager Create Access.

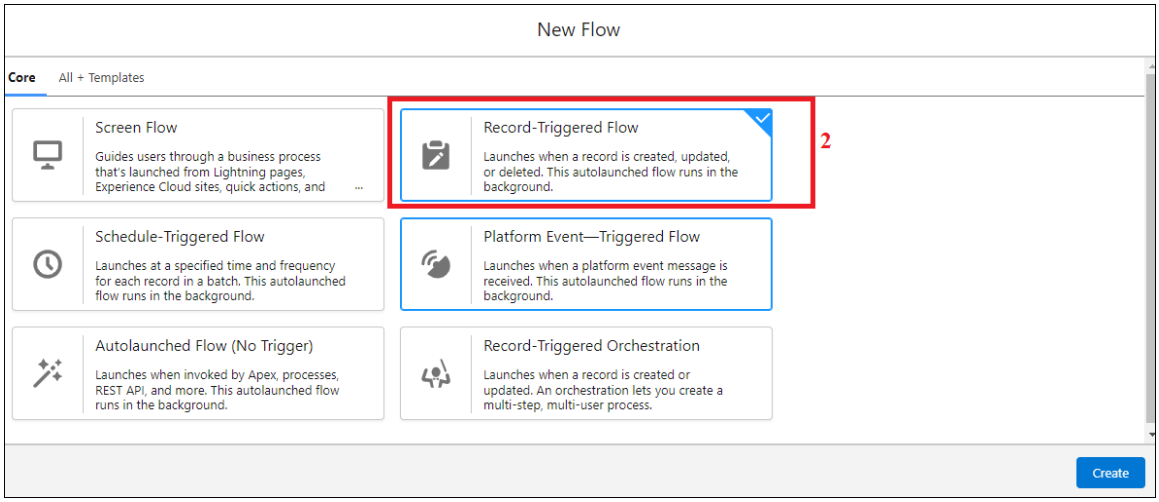


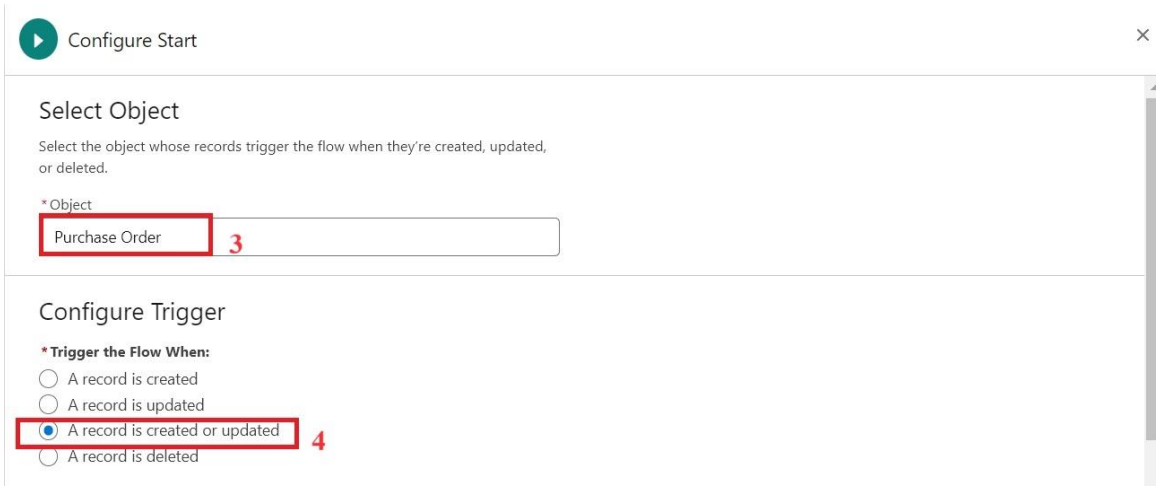


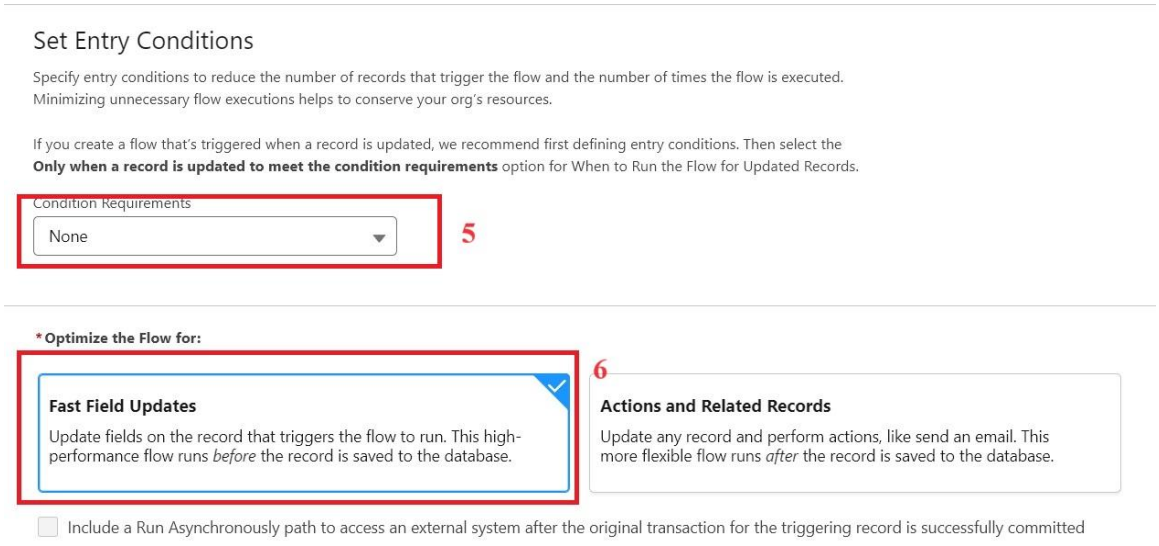
2.7 Flows

A Record-Triggered Flow was created to automatically update the Actual Delivery Date based on the order date + 3 days.


New Flow
Select how you'd like to start building your automation.
Start From Scratch
Select your automation type and start building on an empty canvas.
Use a Template
Select a pre-built flow and customize it to fit your needs.
Back
Next

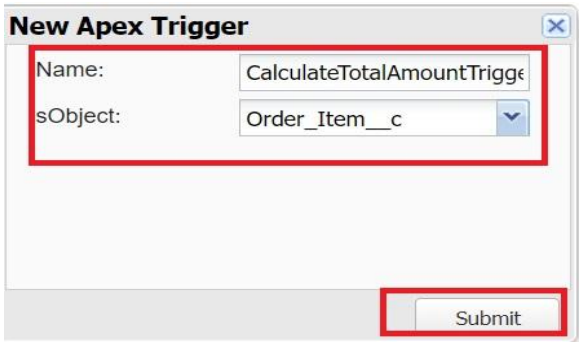

New Flow
Core
All + Templates
Screen Flow
Guides users through a business process that's launched from Lightning pages, Experience Cloud sites, quick actions, and ...
Record-Triggered Flow
Launches when a record is created, updated, or deleted. This autolaunched flow runs in the background.
2
Schedule-Triggered Flow
Launches at a specified time and frequency for each record in a batch. This autolaunched flow runs in the background.
Platform Event—Triggered Flow
Launches when a platform event message is received. This autolaunched flow runs in the background.
Autolaunched Flow (No Trigger)
Launches when invoked by Apex, processes, REST API, and more. This autolaunched flow runs in the background.
Record-Triggered Orchestration
Launches when a record is created or updated. An orchestration lets you create a multi-step, multi-user process.
Create


Configure Start
Select Object
Select the object whose records trigger the flow when they're created, updated, or deleted.
*Object
Purchase Order
3
Configure Trigger
*Trigger the Flow When:
A record is created
A record is updated
A record is created or updated
4
A record is deleted



2.8 Triggers

Apex Trigger and Handler were written to calculate the Total Order Cost dynamically from Order Items.



3. Functional and Performance Testing

- Validated creation of records for all objects.
- Tested validation rules for expected delivery dates.
- Verified reports and dashboards for accuracy.
- Ensured Apex triggers update Total Order Cost correctly.

- Checked performance in terms of record updates and flow execution.

4. Results

- Successfully created Salesforce Medical Inventory Management App.
- Flows and triggers automated key processes.
- Reports summarized purchase orders by suppliers.
- Dashboards provided real-time visualization of data.

5. Outputs and Screenshots

Outputs included:
- Custom Objects & Fields
- Validation Rules
- Flows & Triggers
- Reports & Dashboards

6. Advantages and Disadvantages

Advantages:

- Real-time tracking of medical inventory.
- Automated cost calculation and delivery date updates.
- Easy report generation for management.
- Enhanced data accuracy and security via Salesforce Cloud.

Disadvantages:

- Dependent on internet connectivity.
- Requires Salesforce knowledge for customization.
- Limited offline access.

7. Conclusion

The Medical Inventory Management Salesforce Application successfully achieved its objectives by automating inventory processes, improving accuracy, and providing visual dashboards. This solution can be extended further with advanced analytics, mobile integration, and AI-driven inventory predictions.

8. Appendix

Coding:

1.

```
trigger CalculateTotalAmountTrigger on Order_Item__c (after insert, after update, after delete, after undelete) {

// Call the handler class to handle the logic
```

```
CalculateTotalAmountHandler.calculateTotal(Trigger.new,
Trigger.old, Trigger.isInsert, Trigger.isUpdate, Trigger.isDelete,
Trigger.isUndelete);
}
```

2.

```
public class CalculateTotalAmountHandler {

    // Method to calculate the total amount for Purchase Orders
based on related Order Items
    public static void calculateTotal(List<Order_Item__c>
newItems, List<Order_Item__c> oldItems, Boolean isInsert,
Boolean isUpdate, Boolean isDelete, Boolean isUndelete) {

        // Collect Purchase Order IDs affected by changes in
Order_Item__c records
        Set<Id> parentIds = new Set<Id>();

        // For insert, update, and undelete scenarios
        if (isInsert || isUpdate || isUndelete) {
            for (Order_Item__c ordItem : newItems) {
                parentIds.add(ordItem.Purchase_Order_Id__c);
            }
```

```
    }

    // For update and delete scenarios
    if (isUpdate || isDelete) {
        for (Order_Item__c ordItem : oldItems) {
            parentIds.add(ordItem.Purchase_Order_Id__c);
        }
    }

    // Calculate the total amounts for affected Purchase
Orders
    Map<Id, Decimal> purchaseToUpdateMap = new Map<Id,
Decimal>();

    if (!parentIds.isEmpty()) {
        // Perform an aggregate query to sum the Amount__c for
each Purchase Order
        List<AggregateResult> aggrList = [
            SELECT Purchase_Order_Id__c, SUM(Amount__c)
totalAmount
            FROM Order_Item__c
            WHERE Purchase_Order_Id__c IN :parentIds
```

```
        GROUP BY Purchase_Order_Id__c
    ];

    // Map the result to Purchase Order IDs
    for (AggregateResult aggr : aggrList) {
        Id purchaseOrderId =
(Id)aggr.get('Purchase_Order_Id__c');
        Decimal totalAmount =
(Decimal)aggr.get('totalAmount');
        purchaseToUpdateMap.put(purchaseOrderId,
totalAmount);
    }

    // Prepare Purchase Order records for update
    List<Purchase_Order__c> purchaseToUpdate = new
List<Purchase_Order__c>();
    for (Id purchaseOrderId :
purchaseToUpdateMap.keySet()) {
        Purchase_Order__c purchaseOrder = new
Purchase_Order__c(Id = purchaseOrderId, Total_Order_cost__c
= purchaseToUpdateMap.get(purchaseOrderId));
        purchaseToUpdate.add(purchaseOrder);
    }
```

```
        // Update Purchase Orders if there are any changes
        if (!purchaseToUpdate.isEmpty()) {
            update purchaseToUpdate;
        }
    }
  }
}
```