PROJECT TITLE: Medical Inventory Management System

College Name: United College of Arts and Science

College Code: brubl

TEAM ID: NM2025TMID27831

TEAM MEMBERS:

- **Team Leader Name:** Mithuna A mithunaanand4@gmail.com
- **Team Member 1:** Priyanka R priyankaramesh2804@gmail.com
- **Team Member 2:** Kaleeshwaran N kaleeshjack05@gamil.com
- **Team Member 3:** Deepan kumar S deebankumar332006@gmail.com

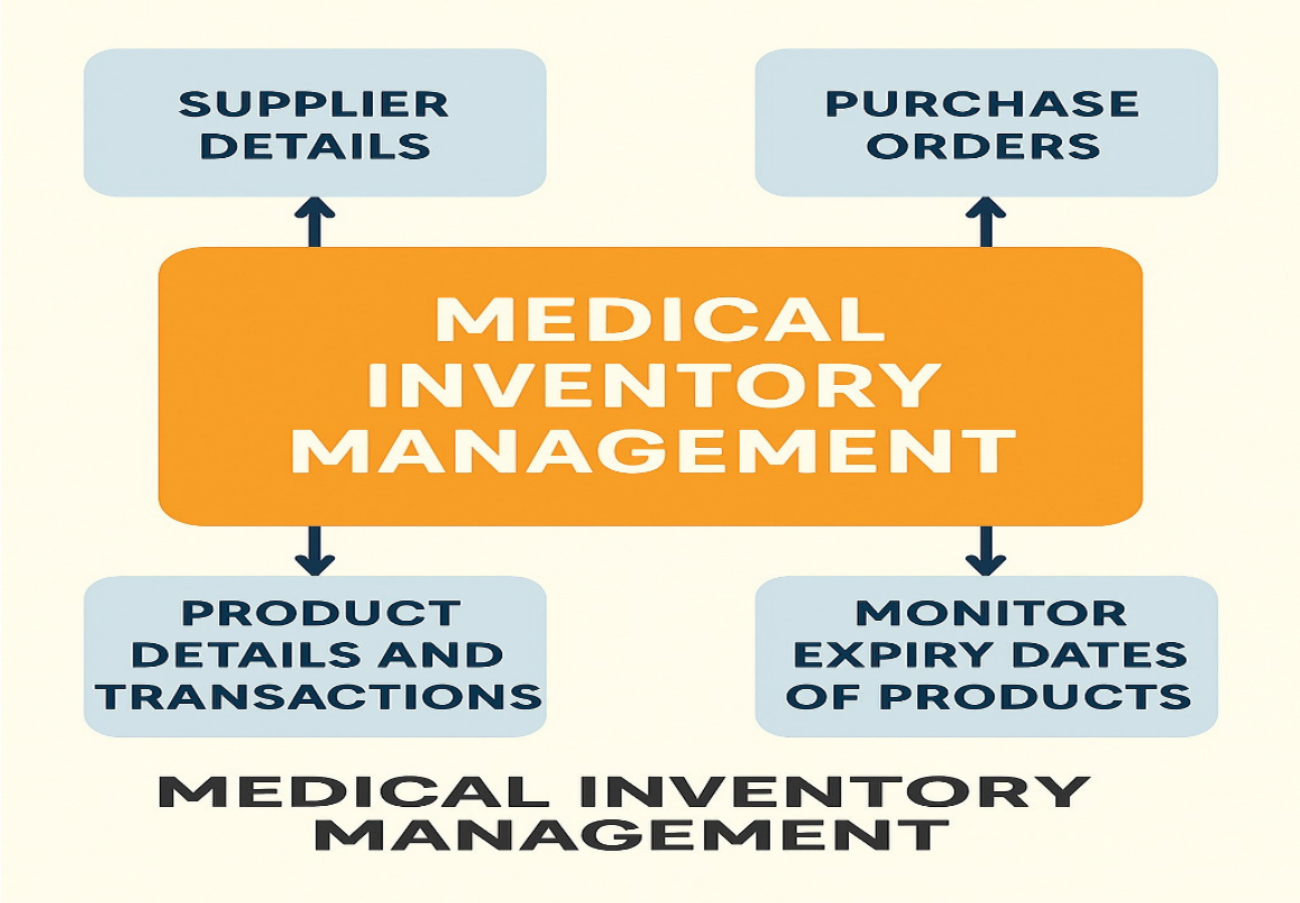
DEMO VIDEO LINK:

NM_Demo video(Medical inventory management).mp4

INTRODUCTION

Project Overview

The Medical Inventory Management System is a comprehensive Salesforce application designed to streamline and manage various operational aspects of the medical inventory. It can efficiently maintain supplier details, manage purchase orders, track product details and transactions, and monitor expiry dates of products, thereby improving operational efficiency, data accuracy, and reporting capabilities.



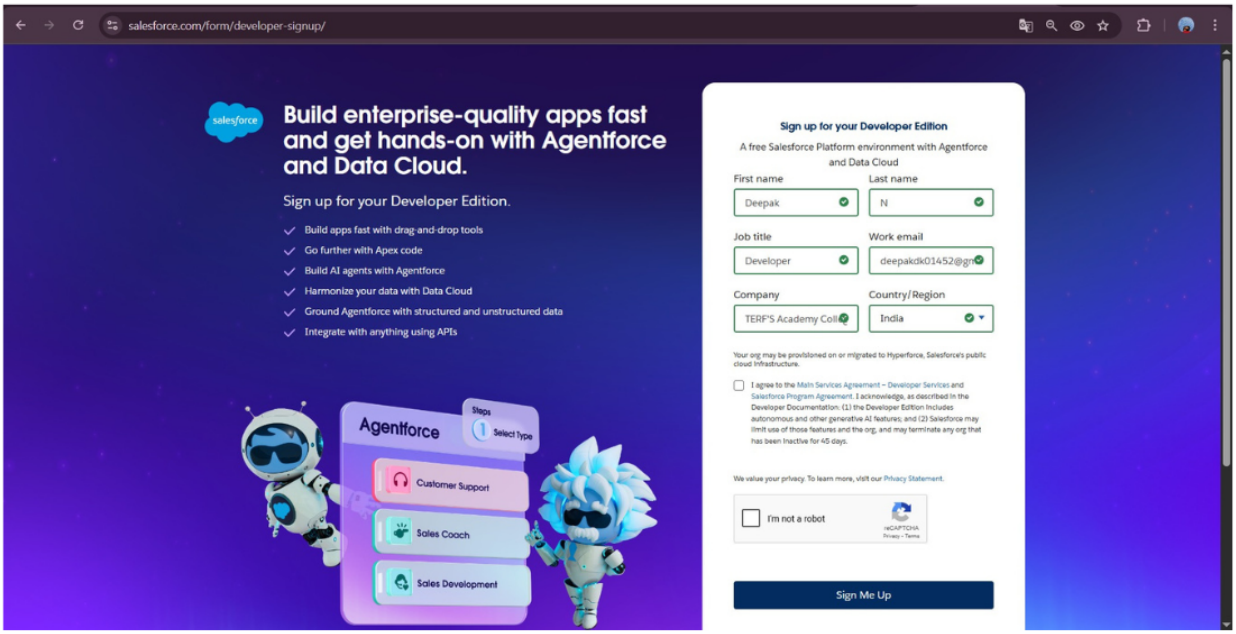
Purpose

The system aims to efficiently maintain supplier details, manage purchase orders, track product details and transactions, and monitor the expiry dates of products. Maintain detailed records of suppliers, including contact information. Catalog product information, including descriptions, stock levels. Monitor and track product expiry dates to avoid using expired items. Comprehensive reports to track supplier performance, and purchase orders.

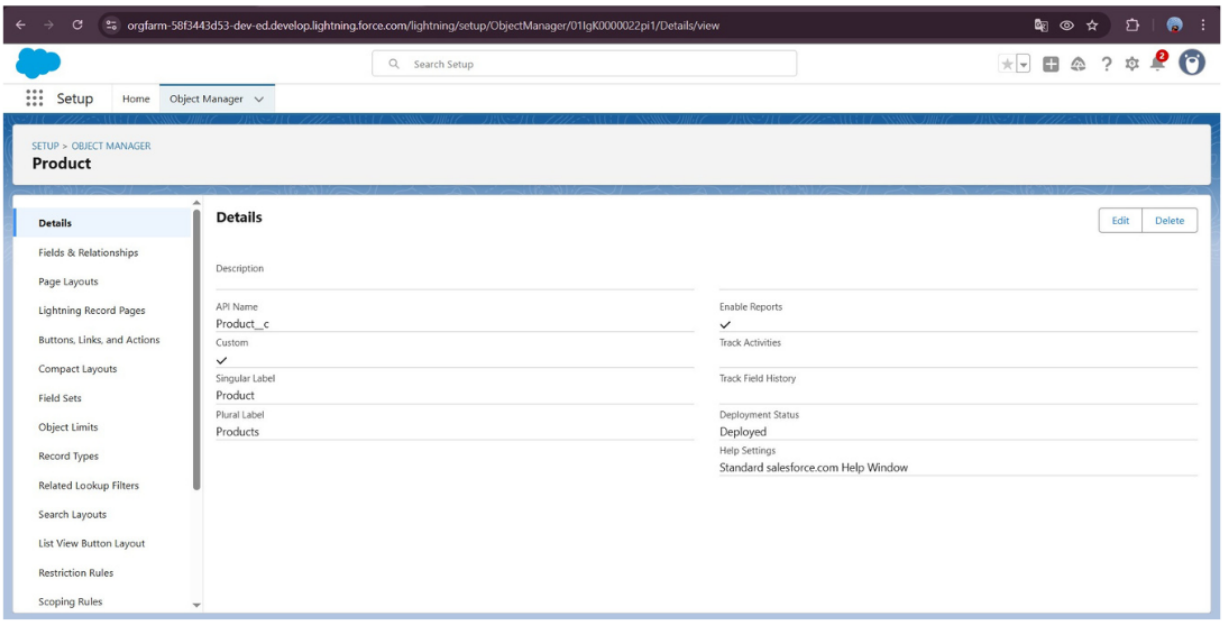
DEVELOPMENT PHASE

Creation of Developer Account

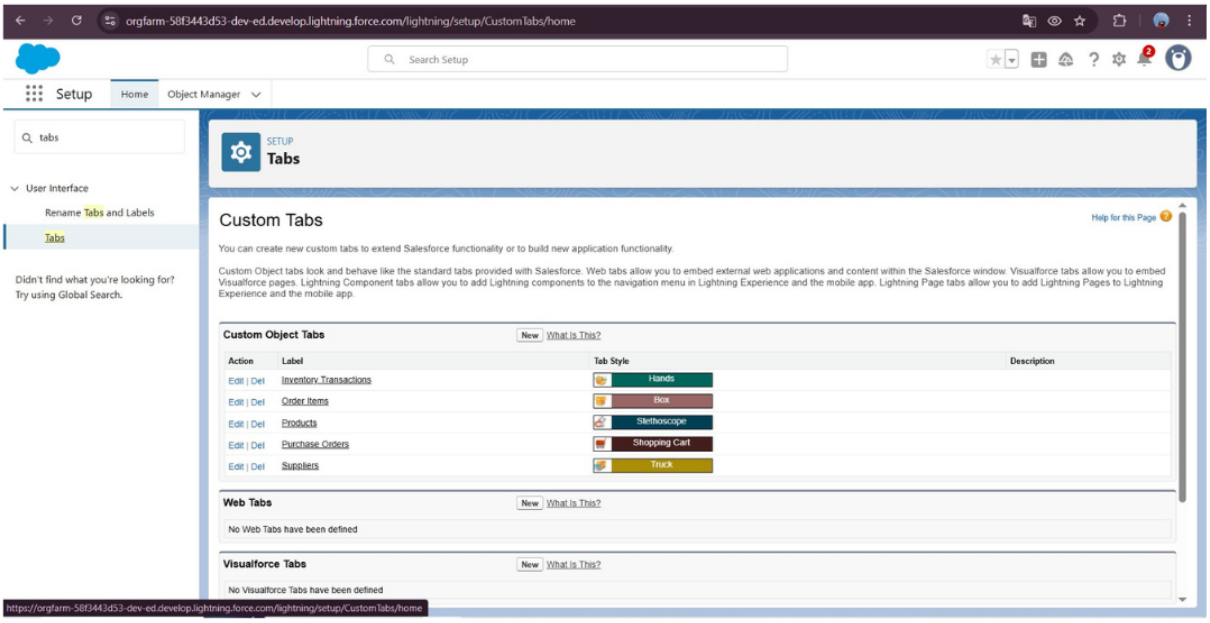
- A Salesforce Developer account was created using the signup link: https://www.salesforce.com/form/developer-signup



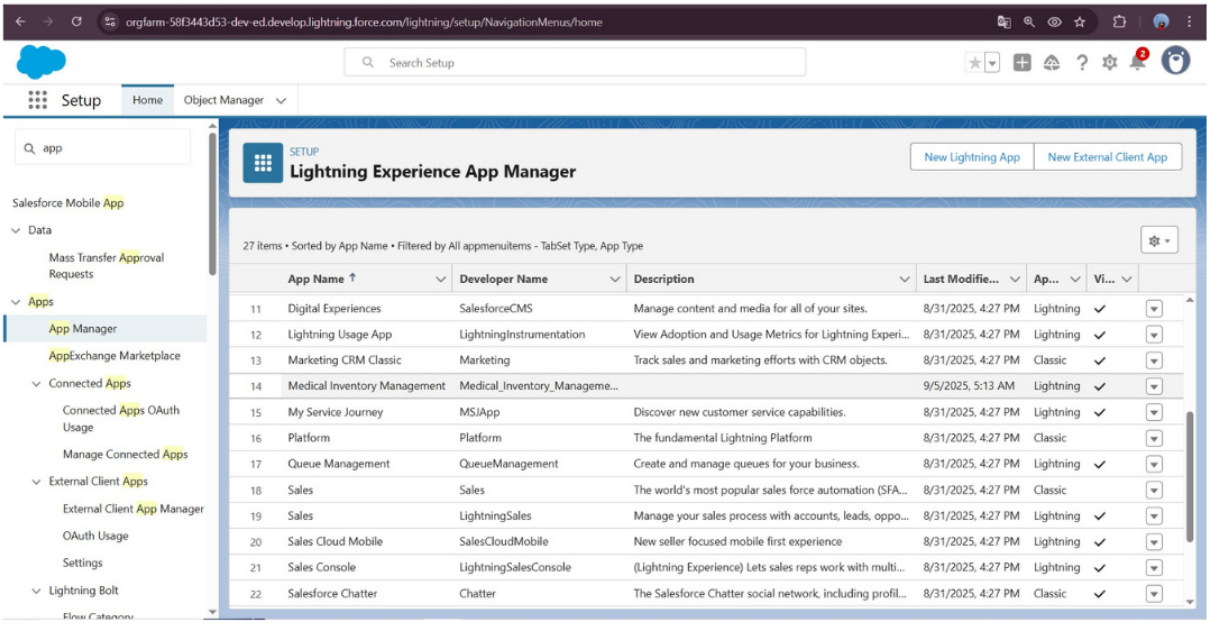
Creating a Product Object



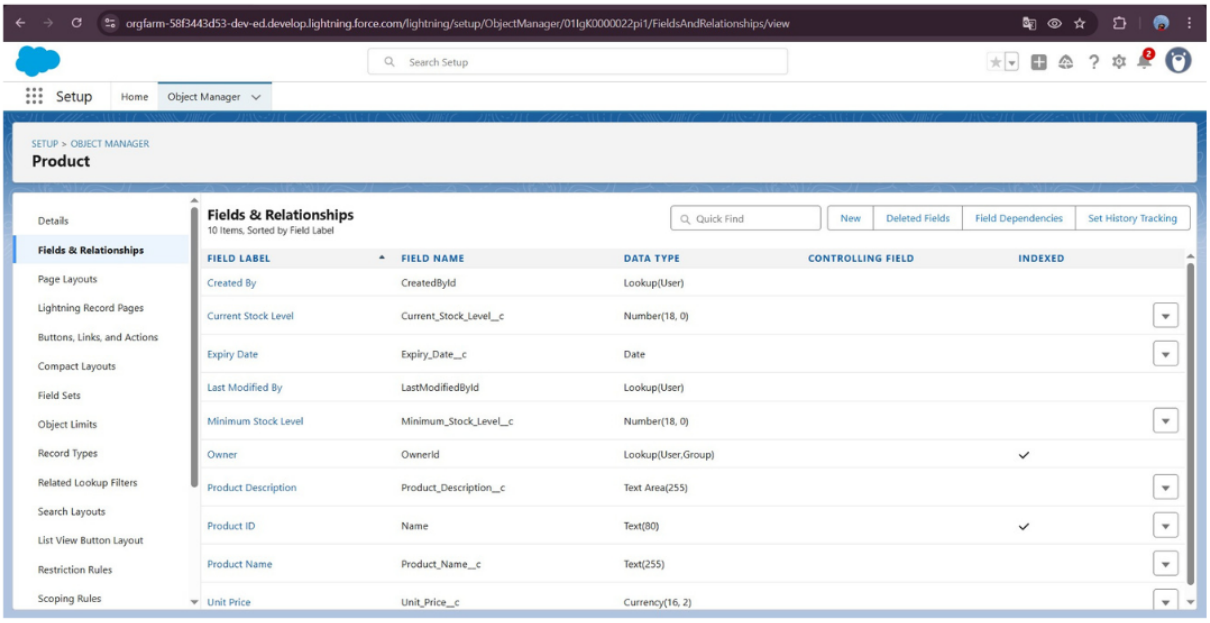
Creating a tab for Product Object and Remaining Tabs



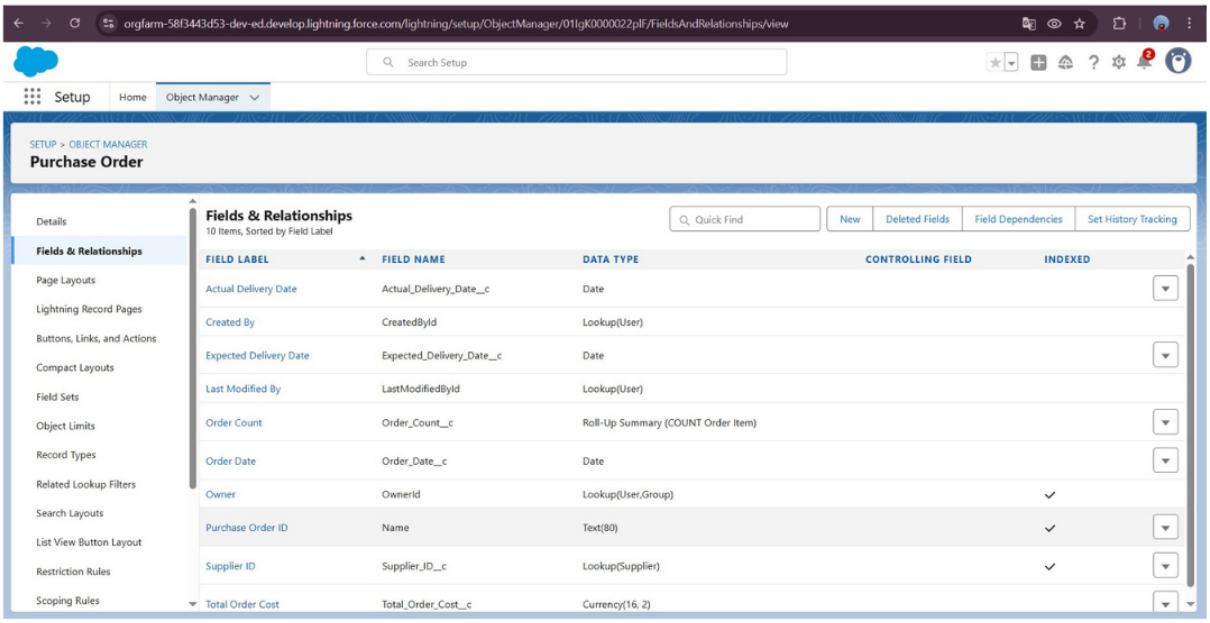
Create a Lightning App for Medical Inventory Management



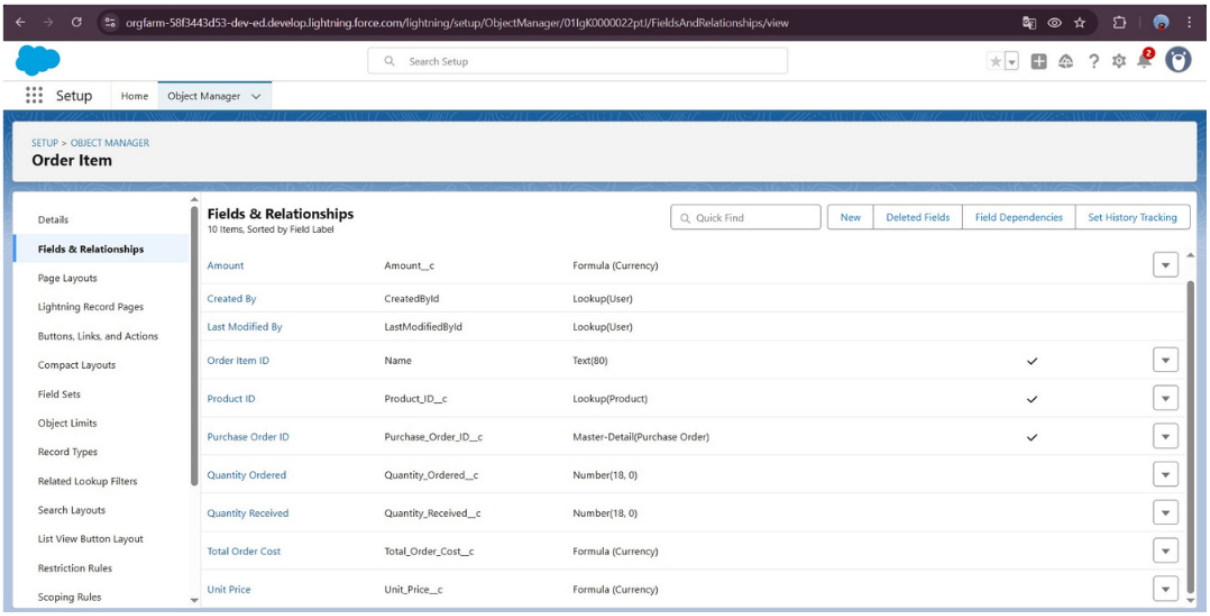
Creating a Text Field in Product Object



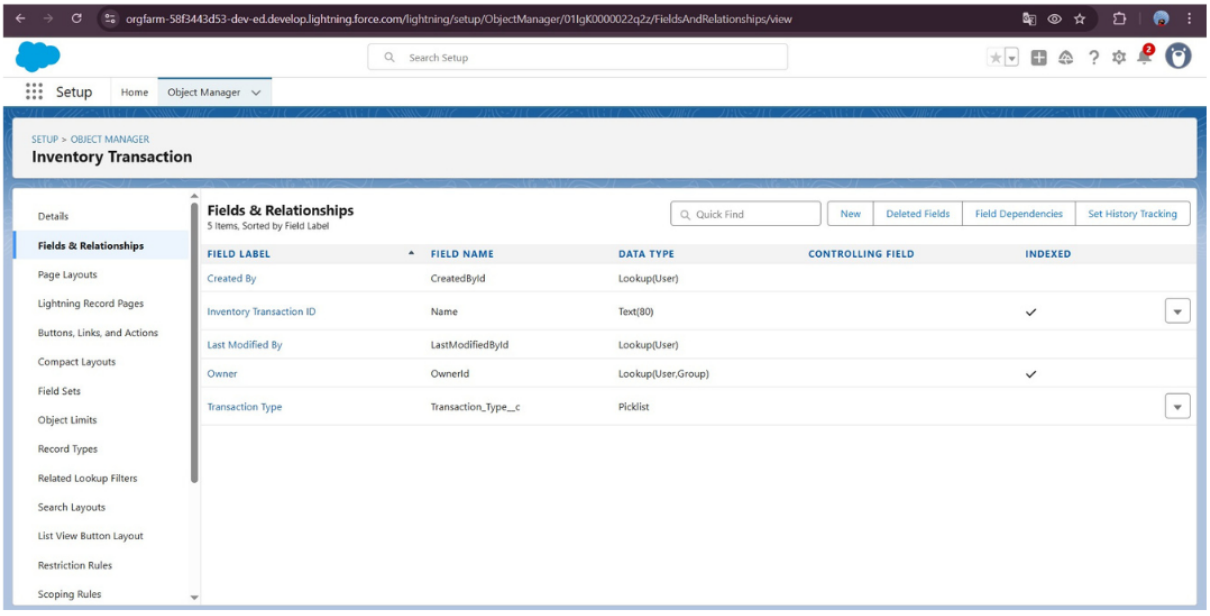
Creating Lookup Relationship in Purchase Order Object



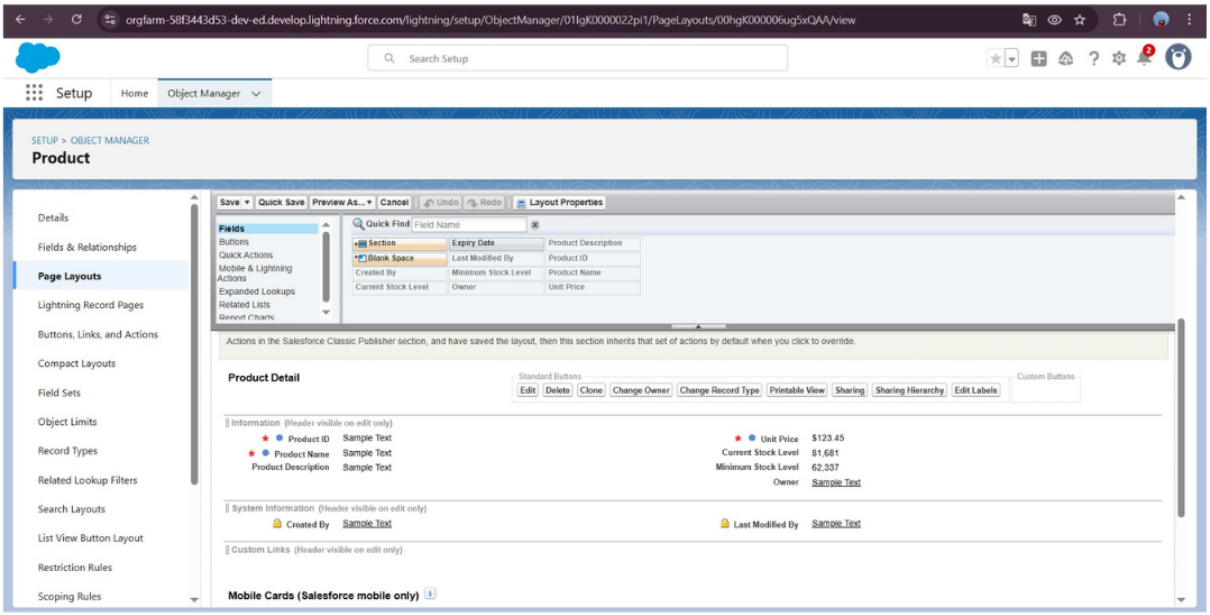
Creating a Unit Price Formula Field in Order Item object



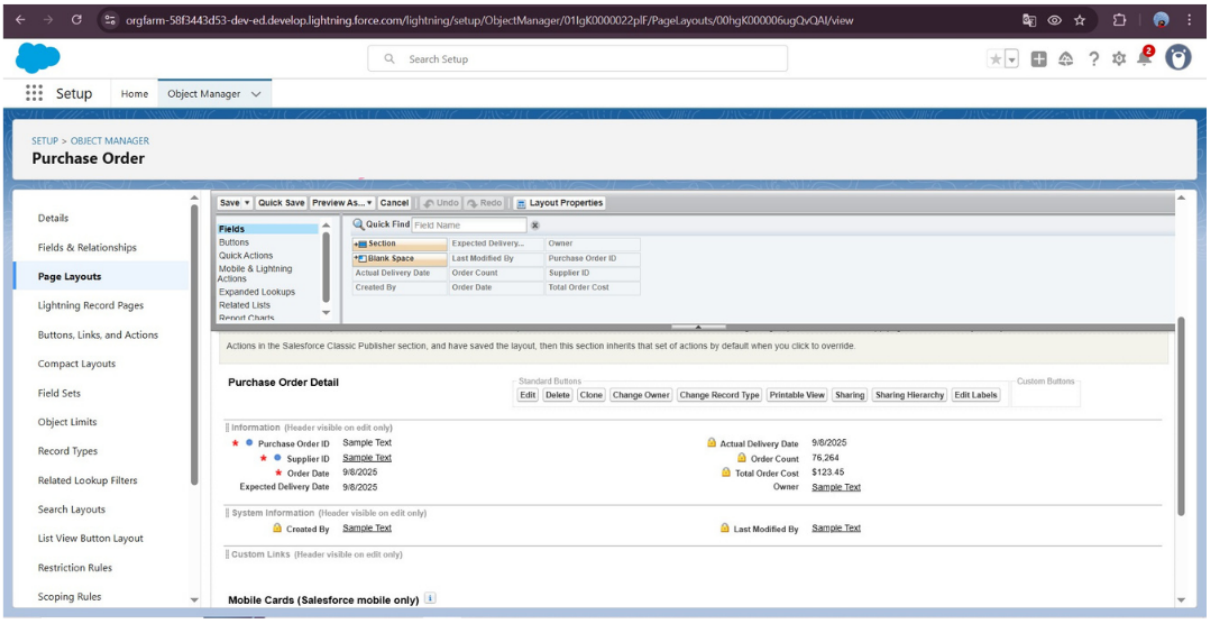
Creating a Picklist Field in Inventory Transaction Object



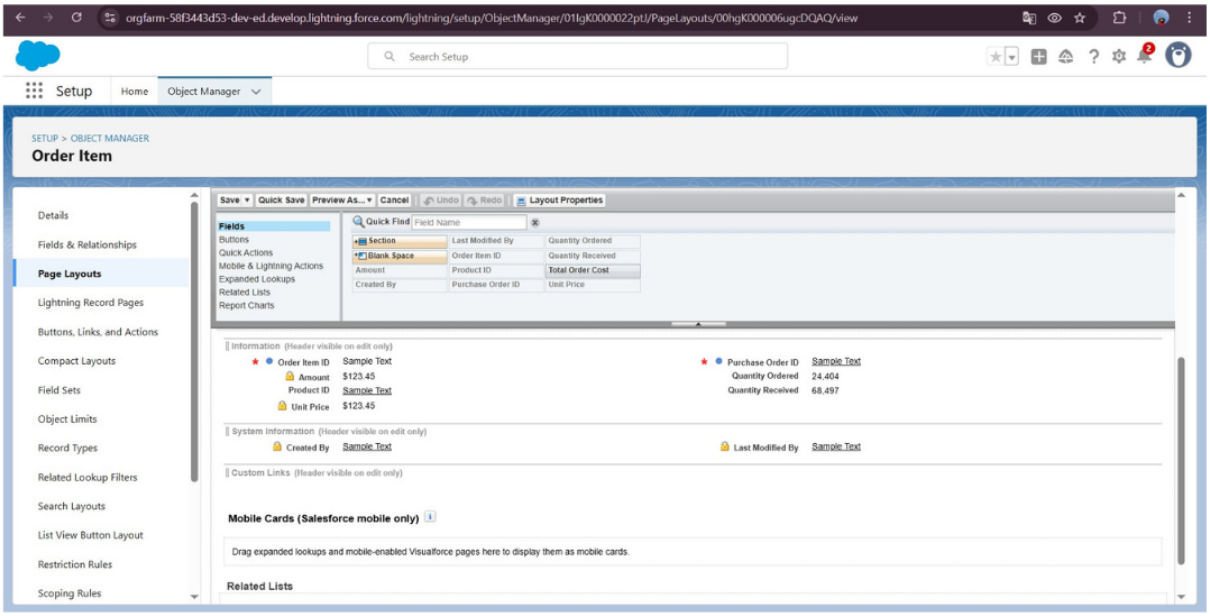
To edit a Page Layout in Product Object



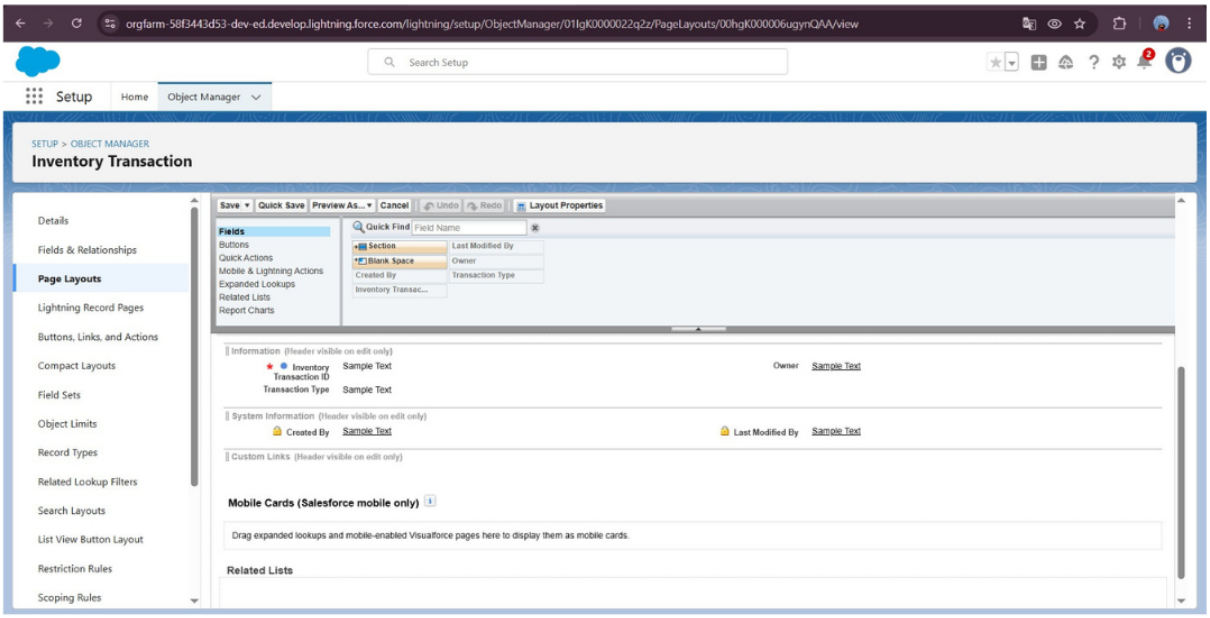
To edit a Page Layout in Purchase Order Object



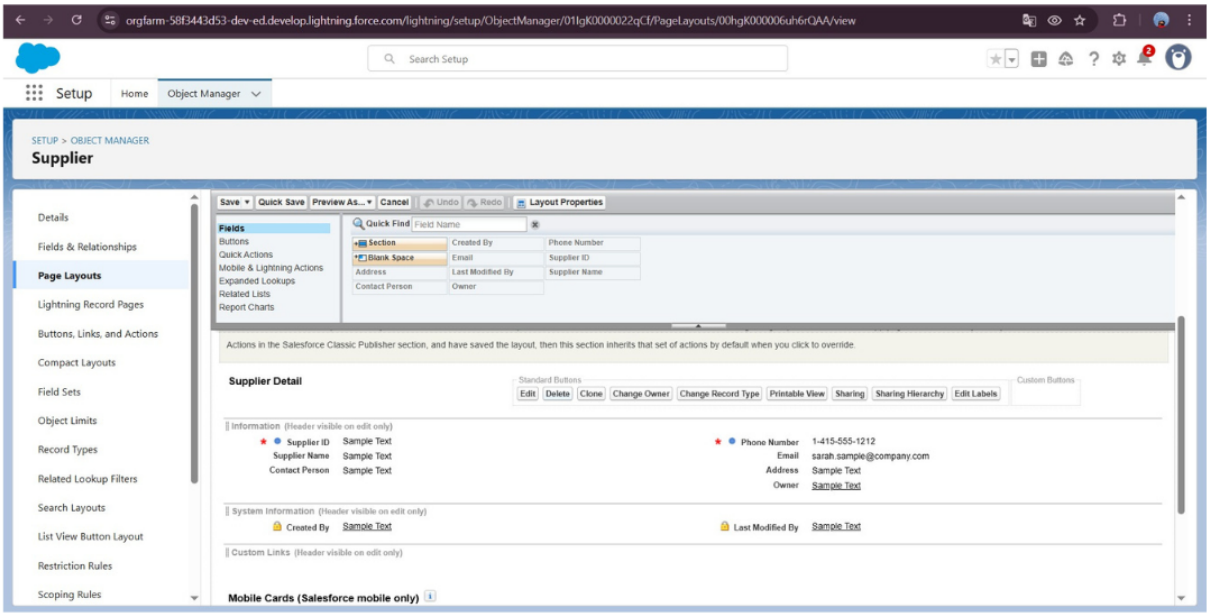
To edit a Page Layout in Order Item Object



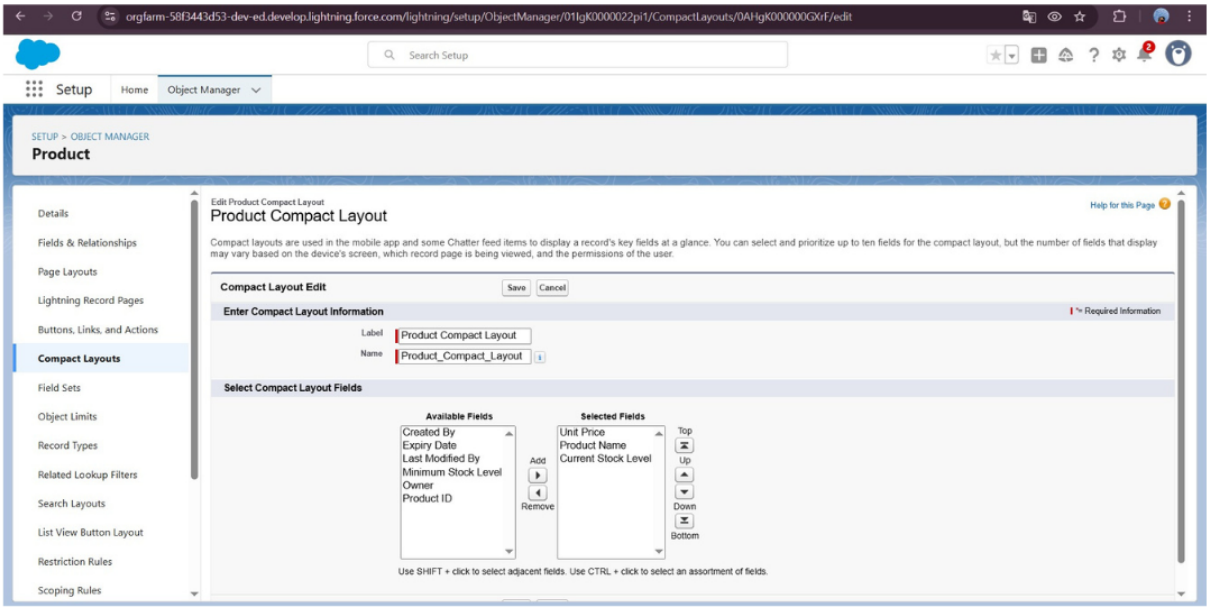
To edit a Page Layout in Inventory Transaction Object



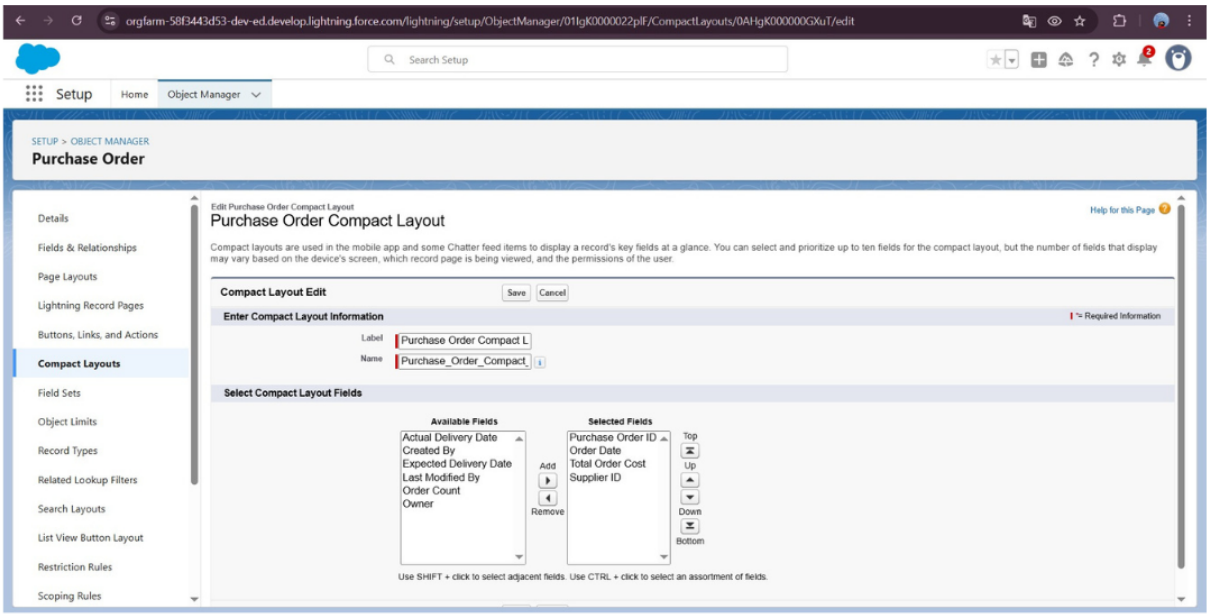
To edit a Page Layout in Supplier Object



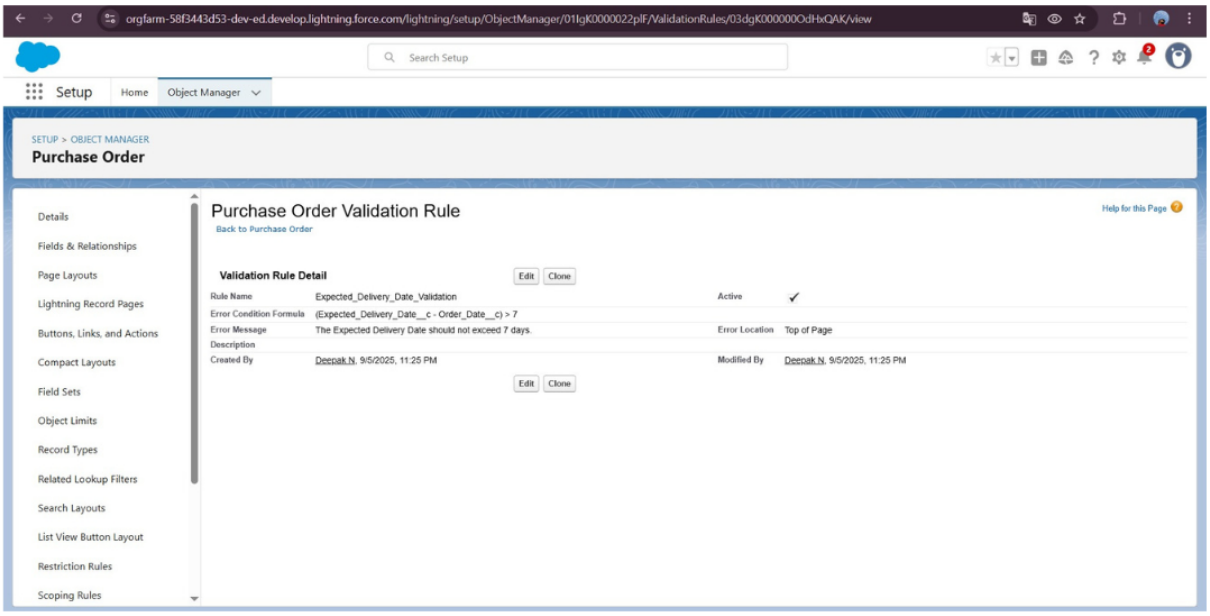
To create a Compact Layout to a Product Object



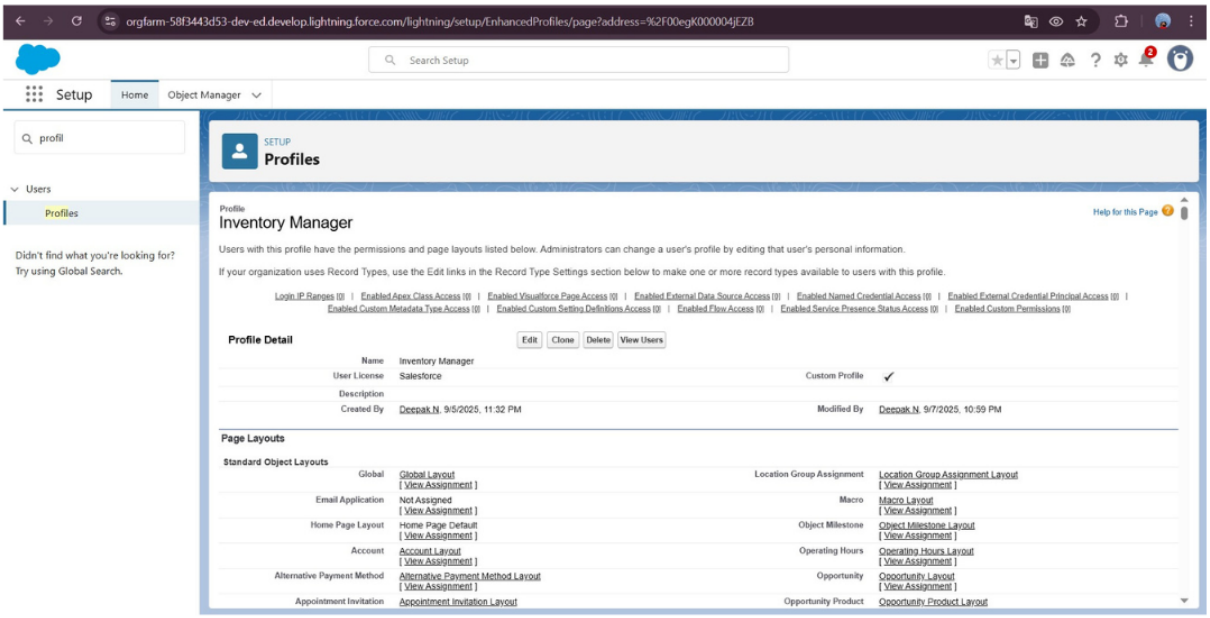
To create a Compact Layout to a Purchase Order Object



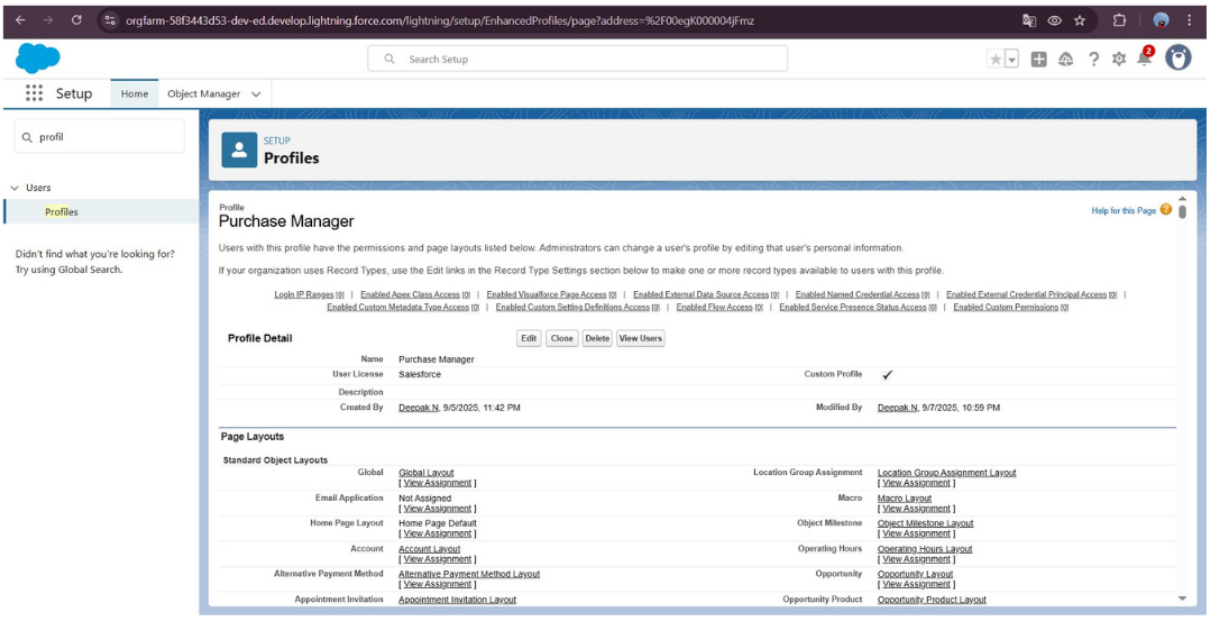
To create an Expected Delivery Date Validation rule to a Employee Object



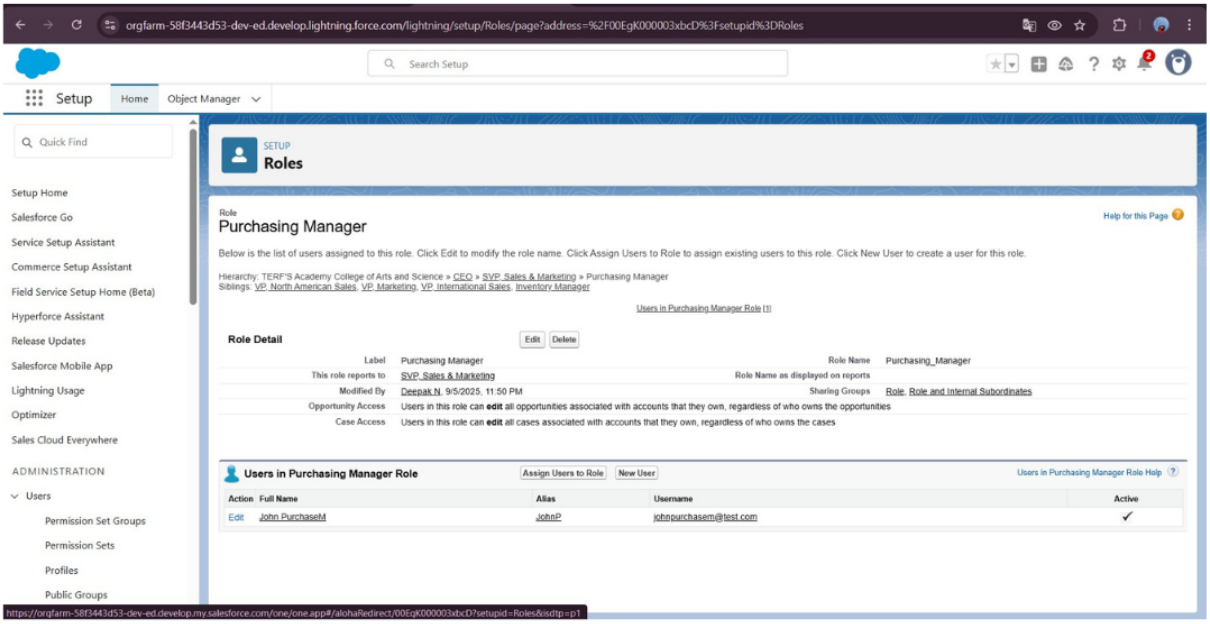
To create an Inventory Manager Profile



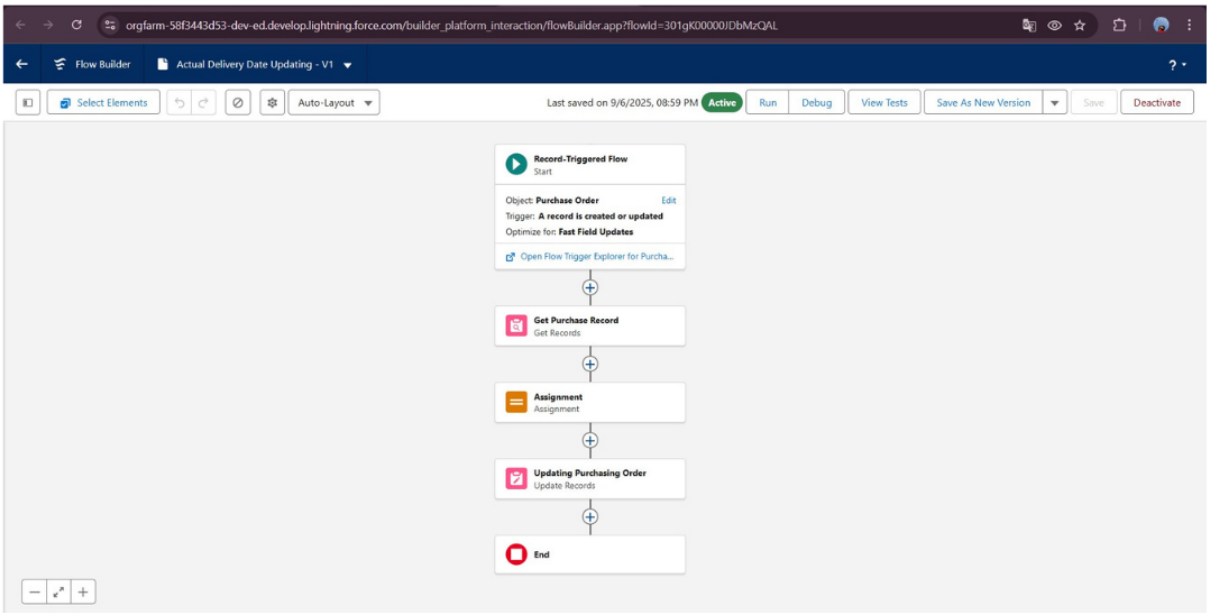
To create an Purchase Manager Profile



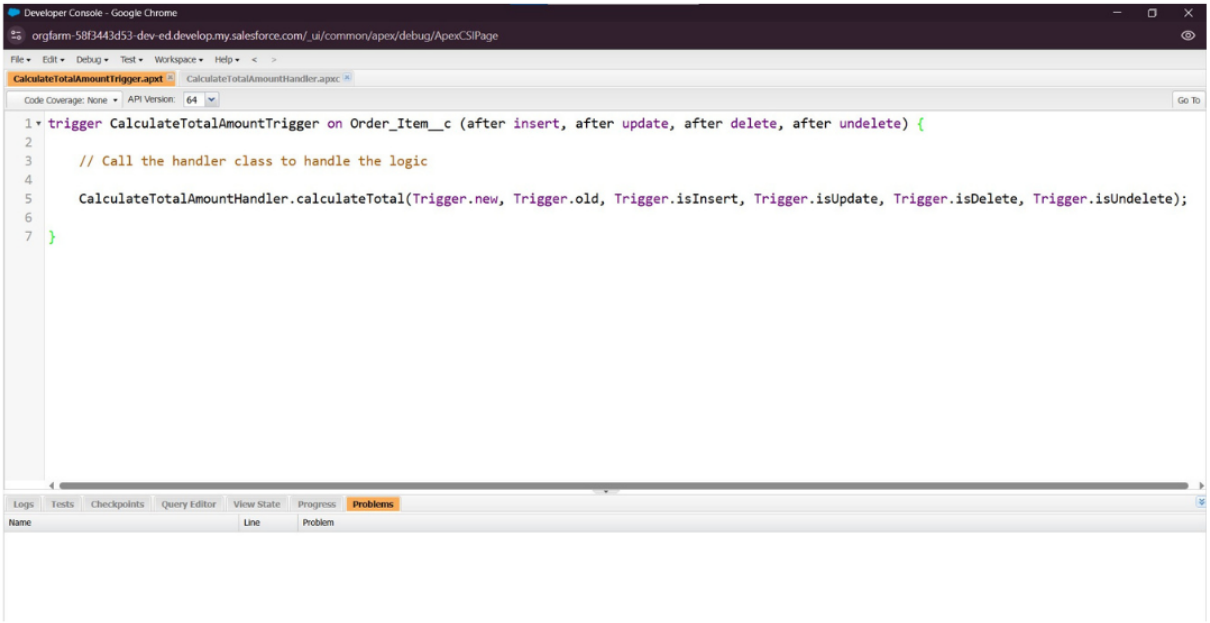
Create a Purchasing Manager Role.



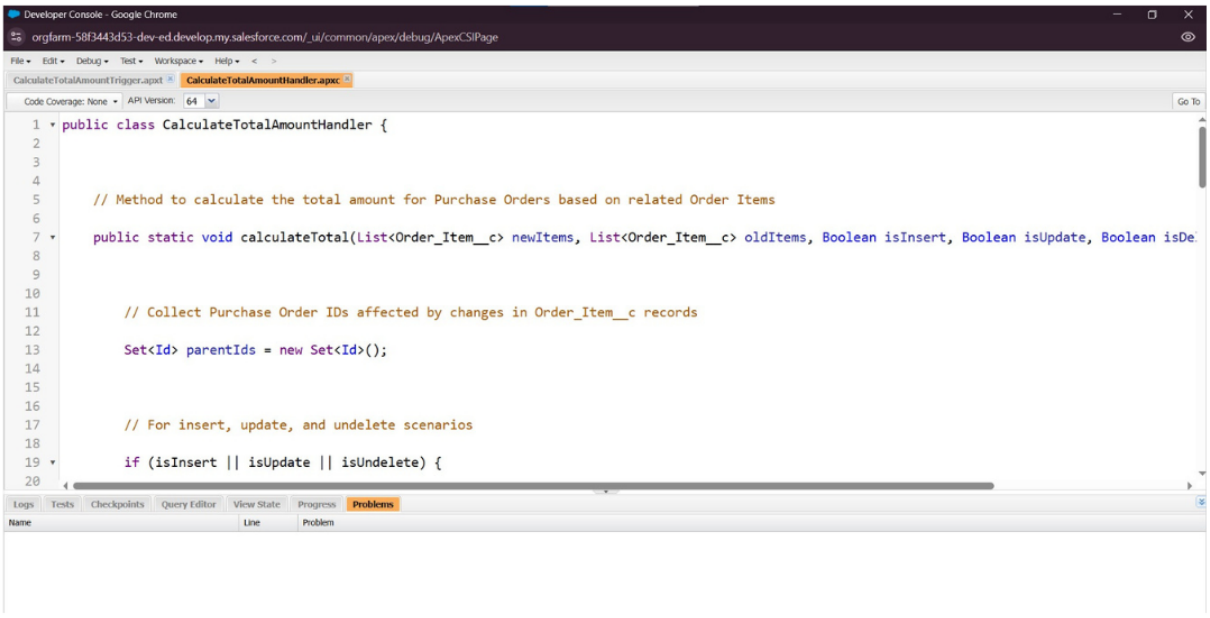
Create Flow to update the Actual Delivery Date.



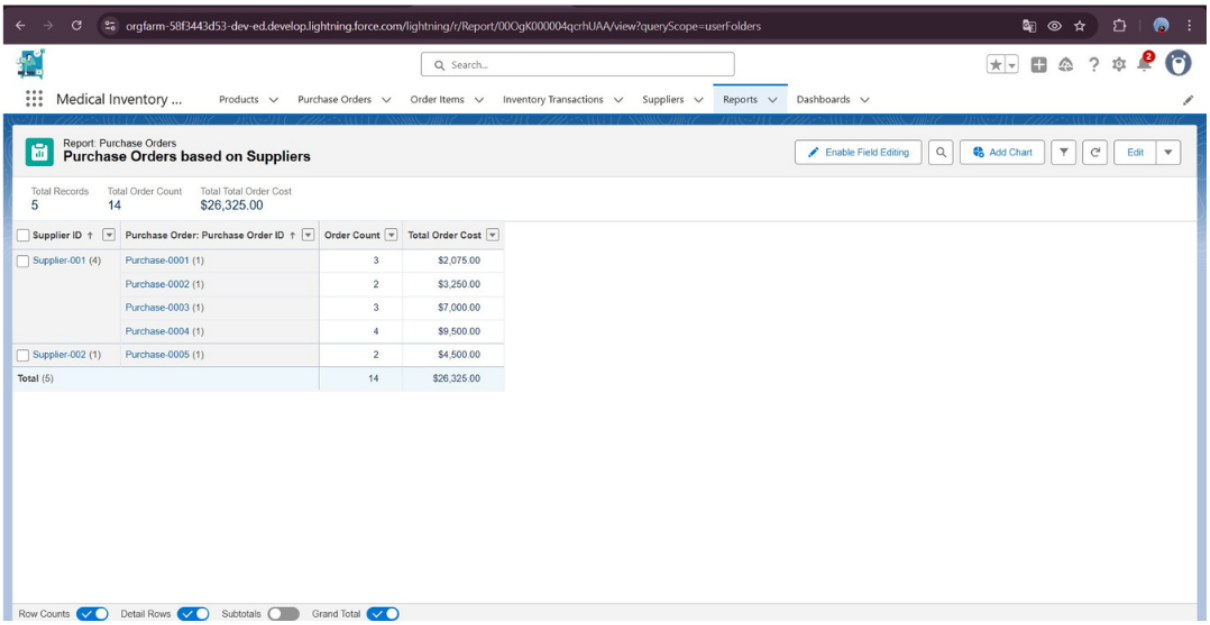
Create a Trigger to Calculate total amount on Order Item.



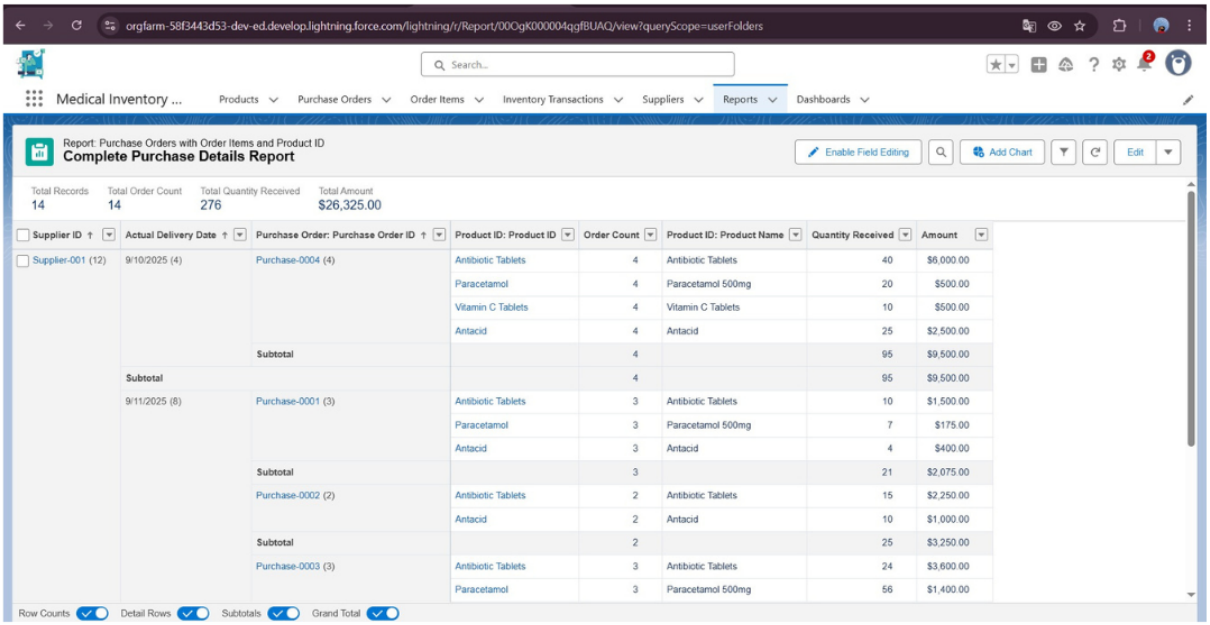
Choose Apex Class: Name it as CalculateTotalAmountHandler



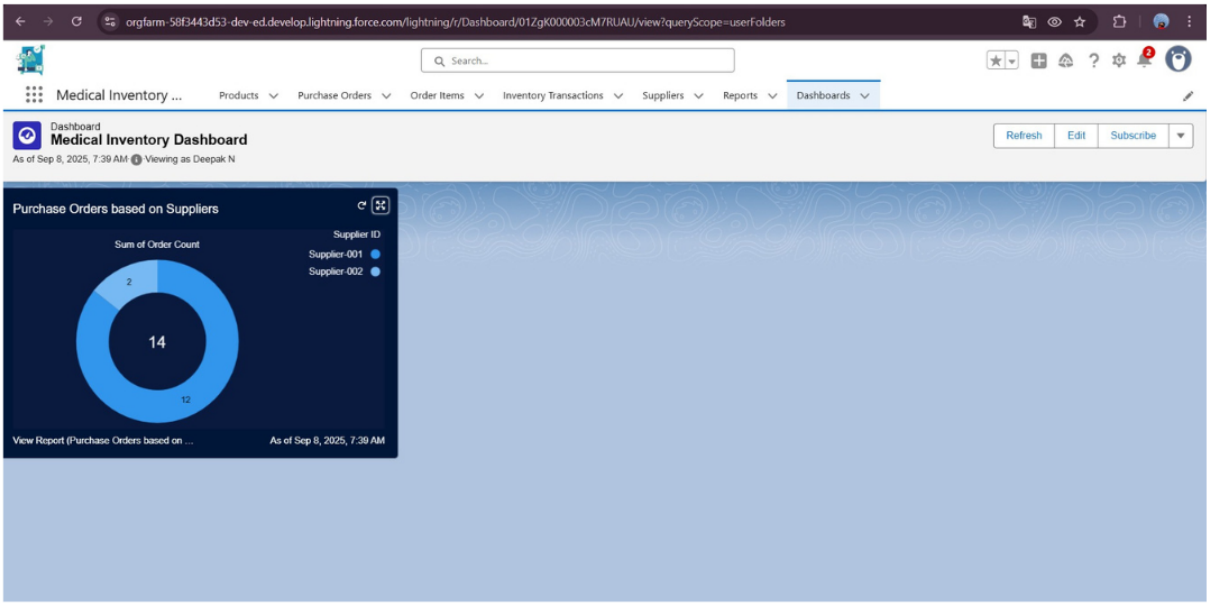
Create a Purchase Orders based on Suppliers(Summary) Report



Create a Complete Purchase Details Report



View Dashboard



RESULTS

- Tabs for Product, Supplier, Purchase Order, Inventory.
- Reports for Expired Products and Supplier Performance.
- Dashboard showing Stock Levels and Purchase Order Summary.
- Trigger execution results (auto-calculated total order amount).
- Validation Rule error messages (when wrong data is entered).

ADVANTAGES & DISADVANTAGES

Advantages

- Accurate tracking of products and expiry dates.
- Easy management of supplier and purchase orders.
- Reduced manual work with automation (flows and triggers).
- Visual dashboards for quick decision-making.

Disadvantages

- Requires Salesforce knowledge for customization.
- Limited offline functionality.
- Integration with external systems (e.g., hospital management software) not implemented yet.

APPENDIX

Create an Apex Trigger:

```
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);
}
```

Create Apex Class:

```
publicclassCalculateTotalAmountHandler {

     //Methodtocalculate the total amount for Purchase Orders based on related Order
Items
    publicstaticvoidcalculateTotal(List<Order_Item__c> newItems, List<Order_Item__c>
oldItems,BooleanisInsert, Boolean isUpdate, Boolean isDelete, Boolean isUndelete) {

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

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

        //Forupdateand 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;
            }
        }
    }
}
```

Future Enhancements

- Add **barcode scanning** for products to make stock entry faster.
- Implement **email orSMSalerts** for products nearing expiry.
- Create **mobile-friendly pages** for quick access by staff.
- Add **AI predictions** for stock demand and reordering.
- Integrate with **external hospital systems** for real-time updates.

CONCLUSION

The Medical Inventory Management System successfully streamlines the operations of managing medical supplies using Salesforce. It ensures better accuracy, reduces errors, and improves efficiency in handling suppliers, purchase orders, and products. With features like validation rules, flows, triggers, reports, and dashboards, the project demonstrates the practical use of Salesforce in real-time business scenarios.