**PROJECT TITLE: Medical Inventory Management System**

**College Name:** KG College of Arts and Science  
**College Code:** bru4y

**TEAM ID: NM2025TMID23491**

**TEAM MEMBERS:**

● **Team Leader Name: SRI SANJAYY S** -2322ja53@kgcas.com   
● **Team Member 1: SUDHARSHAN MYILSAMY**– 2322ja54@kgcas.com  
 ● **Team Member 2: SUMITHRA S** – 2322ja55@kgcas.com   
● **Team Member 3: SURYANARAYANA C P** – 2322ja56@gcas.com

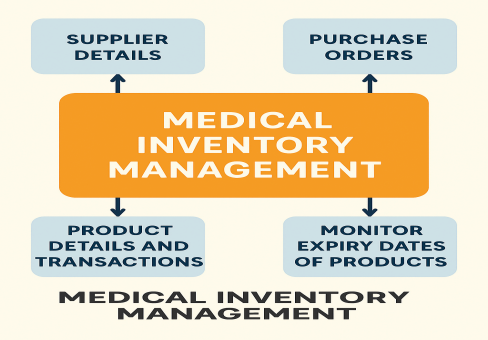
**DEMO VIDEO LINK:**

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**

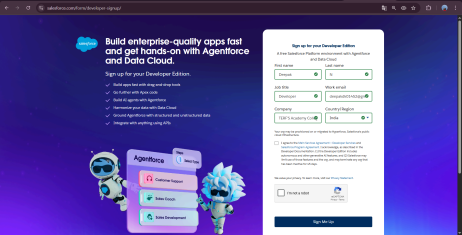
The system aims to efficiently maintain supplier details, manage purchase orders, track product details and transactions, and monitor the expiry dates of products. Maintain detailed records of suppliers, including contact information. Catalog product information, including descriptions, stock levels.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

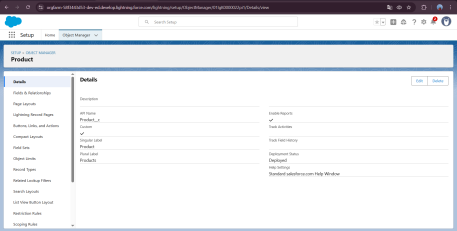
**DEVELOPMENT PHASE**

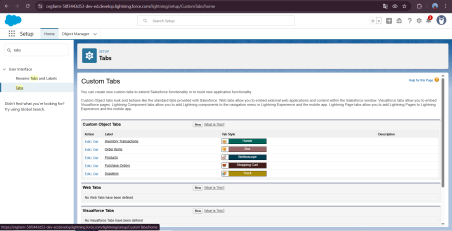
**Creation of Developer Account**

● A Salesforce Developer account was created using the signup link:

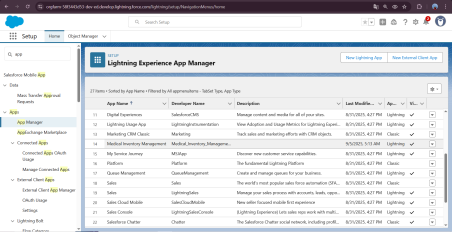
https://www.salesforce.com/form/developer-signup

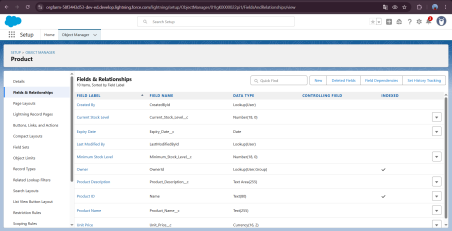
**Creating a Product Object**

**Creating a tab for Product Object and Remaining Tabs**

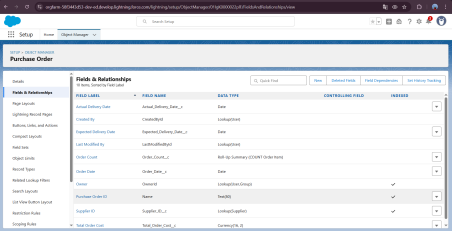
****

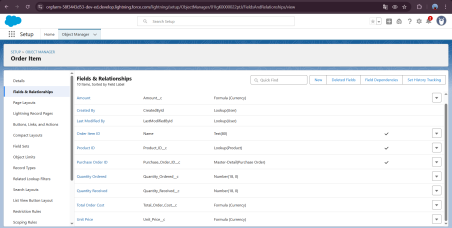
**Create a Lightning App for Medical Inventory Management**

**Creating a Text Field in Product Object**

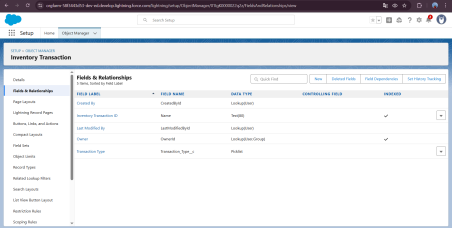
****

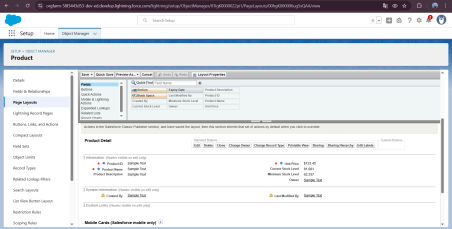
**Creating Lookup Relationship in Purchase Order Object**

**Creating a Unit Price Formula Field in Order Item object**

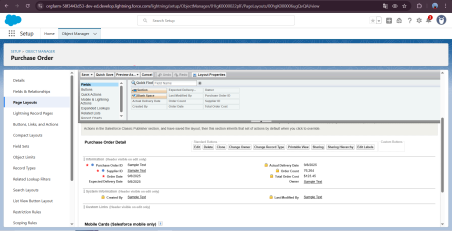
****

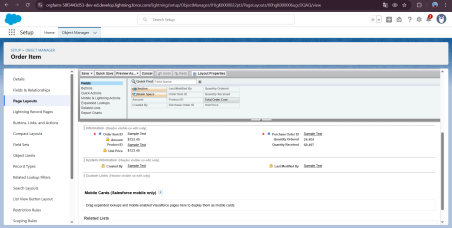
**Creating a Picklist Field in Inventory Transaction Object**

**To edit a Page Layout in Product Object**

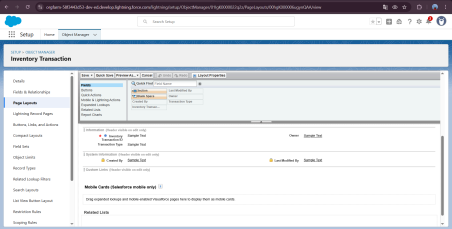
****

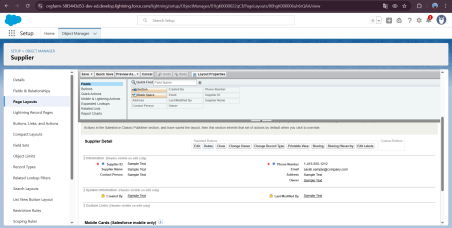
**To edit a Page Layout in Purchase Order Object**

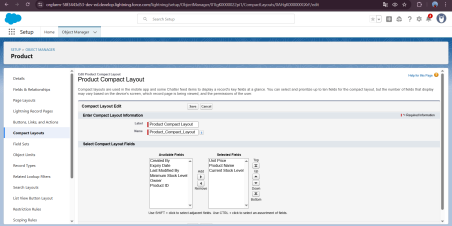
**To edit a Page Layout in Order Item Object**

****

**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:**

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;

}

}

}

}

**Future Enhancements**

● Add **barcode scanning** for products to make stock entry faster. ● Implement **email or SMS alerts** 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.