PROJECT TITLE

Medical Inventory Management System

College Name: TERF's Academy College of Arts and Science
College Code: bru5e

TEAM ID: NM2025TMID27527

TEAM MEMBERS:

- **Team Leader Name:** DEEPAK N – deepakdeepak01452@gmail.com
- **Team Member 1:** ABDUL KALAM AZATH A – akazath277@gmail.com
- **Team Member 2:** VISHNU C R – smilycrd@gmail.com
- **Team Member 3:** DEVADHARSHINI M – ddharshini820@gmail.com

---

1. INTRODUCTION

1.1 Project Overview

The Medical Inventory Management System is a Salesforce-based application designed to streamline the process of managing medical products, suppliers, and purchase orders. It maintains supplier details, tracks product stock levels, monitors expiry dates, and manages purchase transactions. The application also provides automation features such as validation rules, flows, and triggers to improve efficiency and reduce manual work.

1.2 Purpose

The purpose of this project is to build a reliable system that ensures:

- Accurate tracking of products and stock levels.
- Timely identification of expired products.
- Efficient management of supplier and purchase order details.
- Better decision-making with reports and dashboards.

This reduces errors, saves time, and improves operational accuracy in healthcare inventory management.

---

2. DEVELOPMENT PHASE

Creation of Developer Account

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

Objects Created

- **Product** (Product Name, Description, Stock Level, Unit Price, Expiry Date)
- **Supplier** (Supplier Name, Contact Information)
- **Purchase Order** (Order Date, Supplier, Expected & Actual Delivery Date)
- **Order Item** (Products under each Purchase Order)
- **Inventory Transaction** (Stock movement and total cost calculation)

Configurations

- Created fields and relationships between objects (e.g., Purchase Order ↔ Order Item ↔ Product).
- Developed a **Lightning App** with all relevant tabs (Product, Supplier, Purchase Order, Inventory).
- Implemented **Validation Rules** (e.g., Expiry Date cannot be less than today, Stock Level cannot be negative).
- Added **Apex Trigger** to calculate total order amount automatically.
- Built **Flows** for auto-updating delivery dates.
- Created **Reports** for purchase details, supplier performance, and product expiry.
- Built **Dashboards** for graphical representation of stock levels and supplier activity.

---

3. FUNCTIONAL AND PERFORMANCE TESTING

Functional Testing

- Checked validation rules by entering incorrect data (e.g., negative stock levels).
- Verified triggers for total amount calculation on order items.
- Tested flows for auto-updating delivery dates.
- Checked reports for supplier performance and expired products.

Performance Testing

- Verified that the system handles multiple purchase orders and product records without errors.
- Ensured dashboards load data quickly for visualization.

---

4. 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).

---

5. 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.

---

6. 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.

7. APPENDIX

Step 1: Create Apex Trigger

Go to **Developer Console** → **File** → **New** → **Apex Trigger**

Trigger Name: CalculateTotalAmountTrigger
sObject: Order_Item__c

```
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
    );
}
```

Step 2: Create Apex Class (Handler)

Go to **Developer Console → File → New → Apex Class**

Class Name: CalculateTotalAmountHandler

```
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) {
                if (ordItem.Purchase_Order_Id__c != null) {
                    parentIds.add(ordItem.Purchase_Order_Id__c);
                }
            }
        }

        // For update and delete scenarios
        if (isUpdate || isDelete) {
            for (Order_Item__c ordItem : oldItems) {
                if (ordItem.Purchase_Order_Id__c != null) {
```

```
                    parentIds.add(ordItem.Purchase_Order_Id__c);
                }
            }
        }

        // Calculate the total amounts for affected Purchase Orders
        if (!parentIds.isEmpty()) {

            // Aggregate query to sum Amount__c for each Purchase
Order
            Map<Id, Decimal> purchaseToUpdateMap = new Map<Id,
Decimal>();

            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
            ];

            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())
{
                purchaseToUpdate.add(new Purchase_Order__c(
                    Id = purchaseOrderId,
                    Total_Order_cost__c =
purchaseToUpdateMap.get(purchaseOrderId)
                ));
            }
```

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

8. 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.