

# Project Title: Blood Donor Management System

## Problem Statement

Blood donation and availability are often managed manually through registers and phone calls.

- Lack of centralized system leads to delays in finding compatible donors during emergencies.
- Miscommunication between hospitals, blood banks, and donors results in shortages.
- Difficulty in maintaining donor history, eligibility, and tracking regular donors.
- Emergency blood requests are not communicated effectively to potential donors.
- No proper reporting system to analyze donation trends and shortages.

## Project Overview

- A centralized, automated platform for managing donors, blood inventory, and requests.
- Real-time donor matching based on blood group and location.
- Hospitals and blood banks can raise urgent blood requests and notify eligible donors instantly.
- Dashboards and reports for monitoring donor activity, blood availability, and hospital demand.
- Secure storage of donor and patient data with privacy compliance.

## Objective

- Digitize donor registration, hospital requests, and blood availability tracking.
- Provide dashboards for hospitals, blood banks, and administrators to monitor real-time data.
- Allow donors to view their donation history, eligibility, and receive alerts.
- Generate real-time alerts for urgent requests and donor reminders.
- Ensure data accuracy, privacy, and security at all levels.

### Use Cases

- Donor Registration & Management: Donors register, update health status, and check eligibility for donation.
- Blood Inventory Management: Hospitals and blood banks update available blood units with expiry tracking.
- Blood Request & Allocation: Hospitals raise requests; system matches with nearest eligible donors.
- Notifications & Alerts: Donors get alerts for urgent requests, upcoming eligibility, and reminders.

## Phase 1: Problem Understanding & Industry Analysis

### Requirement Gathering

Collect requirements from hospitals, donors, patients, and blood banks.

Identify key pain points: manual processes, delays in finding donors, miscommunication, lack of real-time updates.

Document system requirements: donor registration, blood request, alerts, inventory tracking, and reporting.

### Stakeholder Analysis

Donors: Register, update health status, track donation history, and receive alerts.

Hospitals: Request blood urgently and manage inventory efficiently.

Blood Banks: Update stock, manage expiry dates, and ensure timely donor-hospital matches.

Healthcare Administrators: Monitor availability, donor activity, and compliance with standards.

Patients: Benefit indirectly from faster donor matching and emergency response.

## Business Process Mapping

Current Process: Manual donor identification via phone calls or registers.

Challenges: Slow emergency response, no centralized donor history, limited reporting.

Future Process: Centralized donor database, automated matching, dashboards for hospitals and admins, real-time donor notifications.

## Industry-Specific Use Case Analysis

Donor Management: Digital registration, eligibility monitoring, health checks, and reminders.

Emergency Response: Instant donor matching for hospital requests.

Inventory Tracking: Blood unit expiry monitoring and shortage alerts.

Compliance: Ensure privacy, security, and adherence to government guidelines for blood donation.

## AppExchange Exploration

Explore existing healthcare and blood bank management apps for Salesforce.

Identify reusable components such as donor/volunteer management, healthcare CRMs, and alert systems.

Evaluate whether customization or integration of existing apps provides better efficiency.

# Phase 2: Org Setup & Configuration

## Step 1. Salesforce Editions

- Using Salesforce Developer Edition Org.
- Developer Edition provides Enterprise-level features free with limited licenses and storage.
- Suitable for student projects and practice

## Step 2. Company Profile Setup

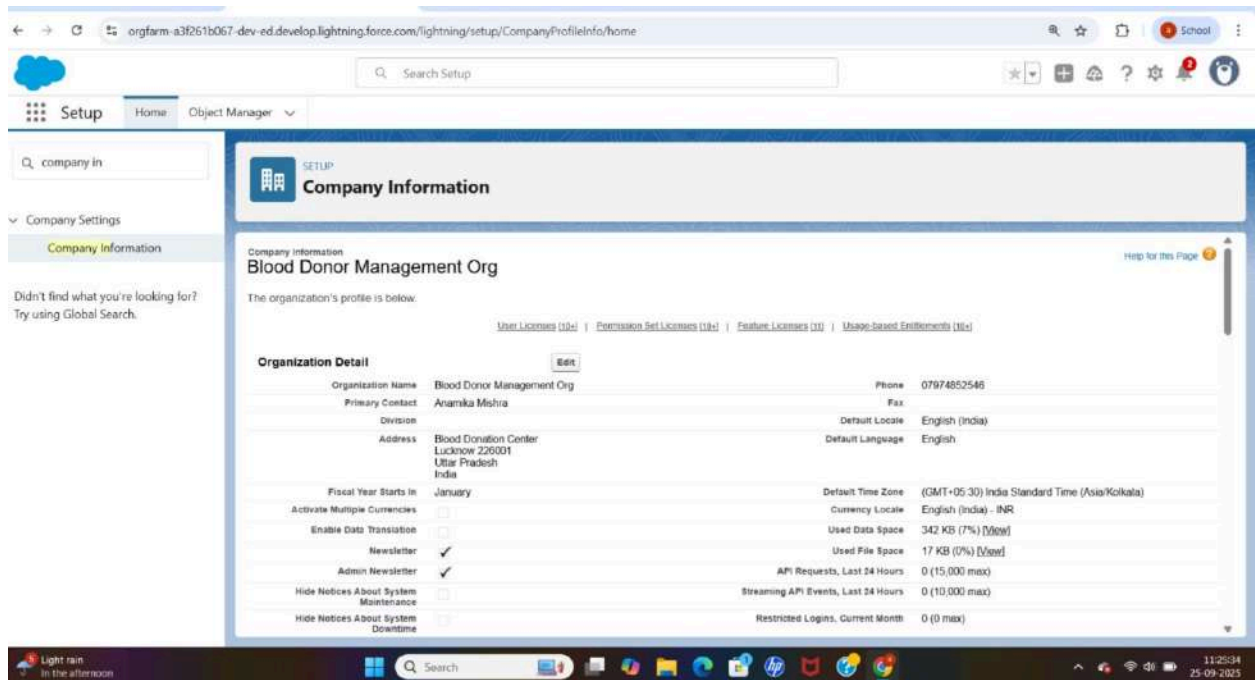
Organization Name: Blood Donor Management Org

Default Locale: English (India)

Default Language: English

Default Time Zone: (GMT +05:30) India Standard Time (Asia/Kolkata)

Default Currency: INR (Indian Rupee)



## Step 3. Business Hours

Name: Blood Donation Hours

Time Zone: GMT +05:30 India Standard Time

Start: 9:00 AM

End: 6:00 PM

Finish setting your Salesforce x Business Hours | Salesforce x

creative-shark-3xakd0-dev-ed.trailblaze.lightning.force.com/lightning/setup/BusinessHours/page?address=%2F01mNS00000CJGBN

Search Setup

Setup Home Object Manager

bush

Company Settings

Business Hours

Didn't find what you're looking for? Try using Global Search.

### Organization Business Hours

Select the days and hours that your support team is available. These hours, when associated with escalation rules, determine the times at which cases can escalate.

If you enter blank business hours for a day, that means your organization does not operate on that day.

Holidays 0

#### Business Hours Detail

Business Hours Name: Blood Donation Hours

Time Zone: (GMT+05:30) India Standard Time (Asia/Kolkata)

Day	Hours
Sunday	24 Hours
Monday	24 Hours
Tuesday	24 Hours
Wednesday	24 Hours
Thursday	24 Hours
Friday	24 Hours
Saturday	24 Hours

Active: ☒

Created By: Anamika Mishra 20/09/2025, 4:00 pm

Last Modified By: Anamika Mishra 20/09/2025, 4:00 pm

#### Holidays

Add/Remove

No records to display

Back To Top

Always show me more records per related list

32°C Partly sunny

Search

16:04:40 20-09-2025

Holiday Name: World Blood Donor Day

Date: 6/14/2026 (All Day)

Warranty-and-Service-Tracker | Lightning Experience | Salesforce | Holidays | Salesforce | Welcome to Salesforce: Reset |

orgtam-a3f261b067-dev-ed.develop.lightning.force.com/lightning/setup/Holiday/page?address=%2F0CDGL00000021BJ

Search Setup

Setup Home Object Manager

hol

Company Settings

Holidays

Didn't find what you're looking for? Try using Global Search.

### Holiday Detail

Holidays are dates and times at which business hours are suspended. These dates and times, when associated with business hours, also suspend any escalation rules associated with business hours.

Add or remove business hours to holidays to suspend business hours and escalation rules during the holidays.

Business Hours 0

#### Holiday Detail

Holiday Name: World Blood Donor Day

Description: World Blood Donor Day - Celebrate and promote blood donation globally

Date and Time: 6/14/2026 All Day

Recurring Holiday: Occurs every June 14 effective 6/14/2026

Created By: Anamika Mishra 9/20/2025, 12:46 PM

Last Modified By: Anamika Mishra 9/20/2025, 12:46 PM

#### Business Hours

Add/Remove

No records to display

Back To Top

Always show me more records per related list

27°C Mostly clear

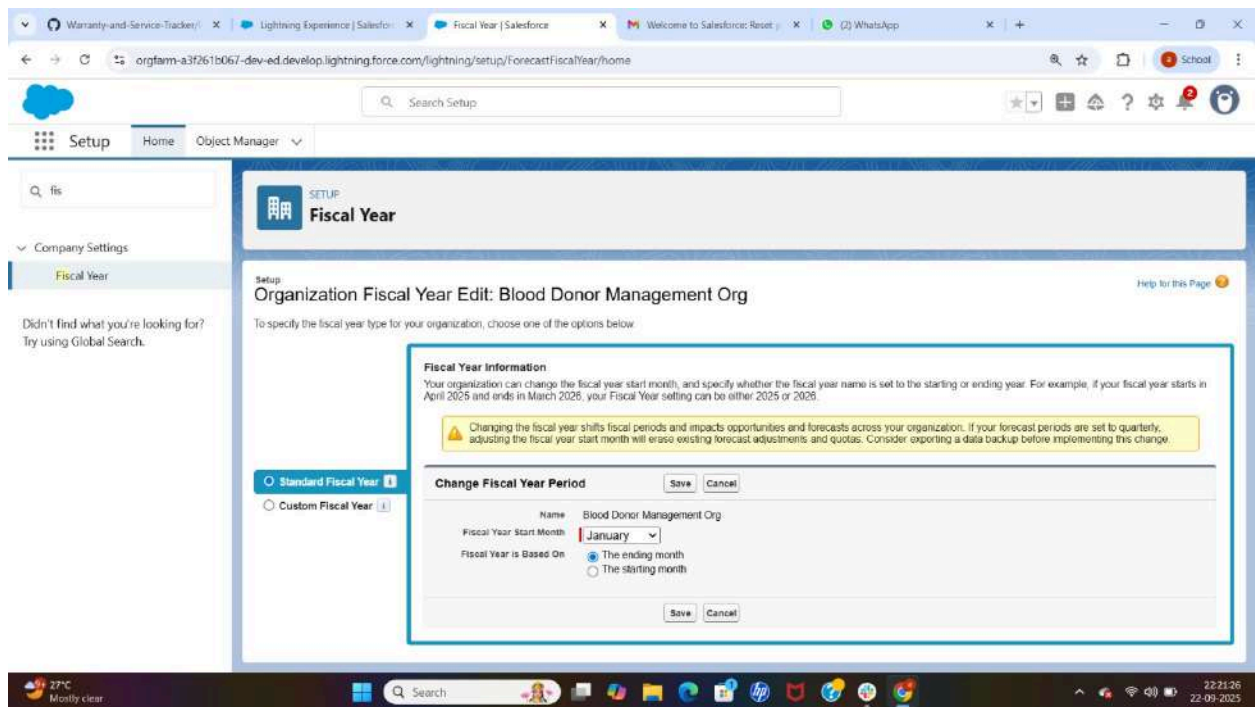
Search

22:17:09 22-09-2025

## Step 4. Fiscal Year Settings

Fiscal Year: Standard (January–December).

Custom Fiscal Year not enabled (not needed for this project).



## Step 5. User Setup & Licenses

Created 2 Test Users

User 1:

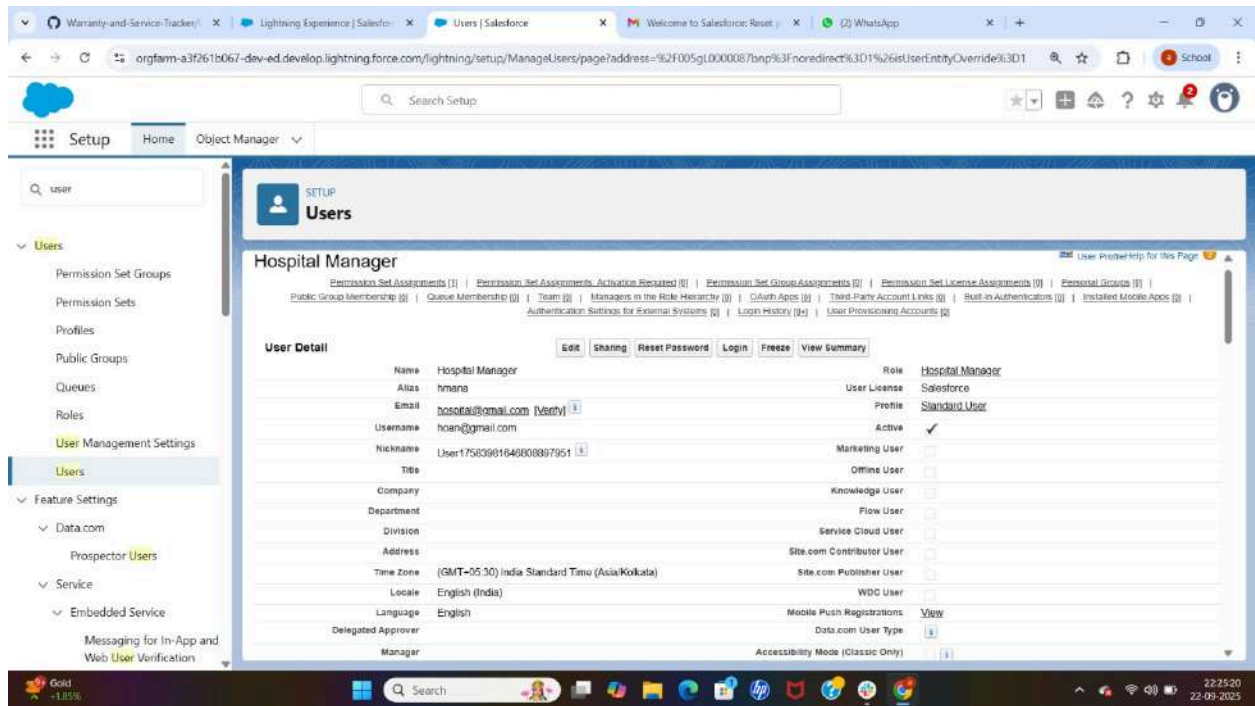
First Name: Hospital

Last Name: Manager

Role: Hospital Manager

User License: Salesforce

Profile: Standard User



User 2:

First Name: Blood

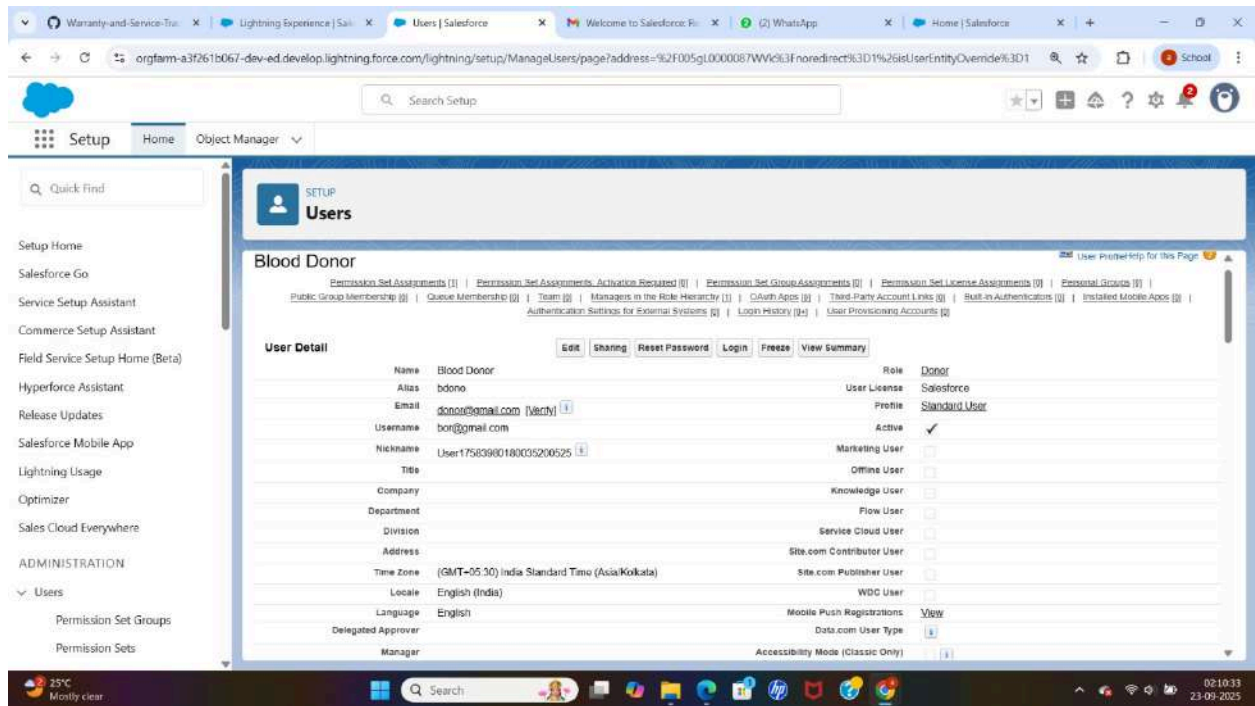
Last Name: Donor

Role: Donor

User License: Salesforce

Profile: Standard User





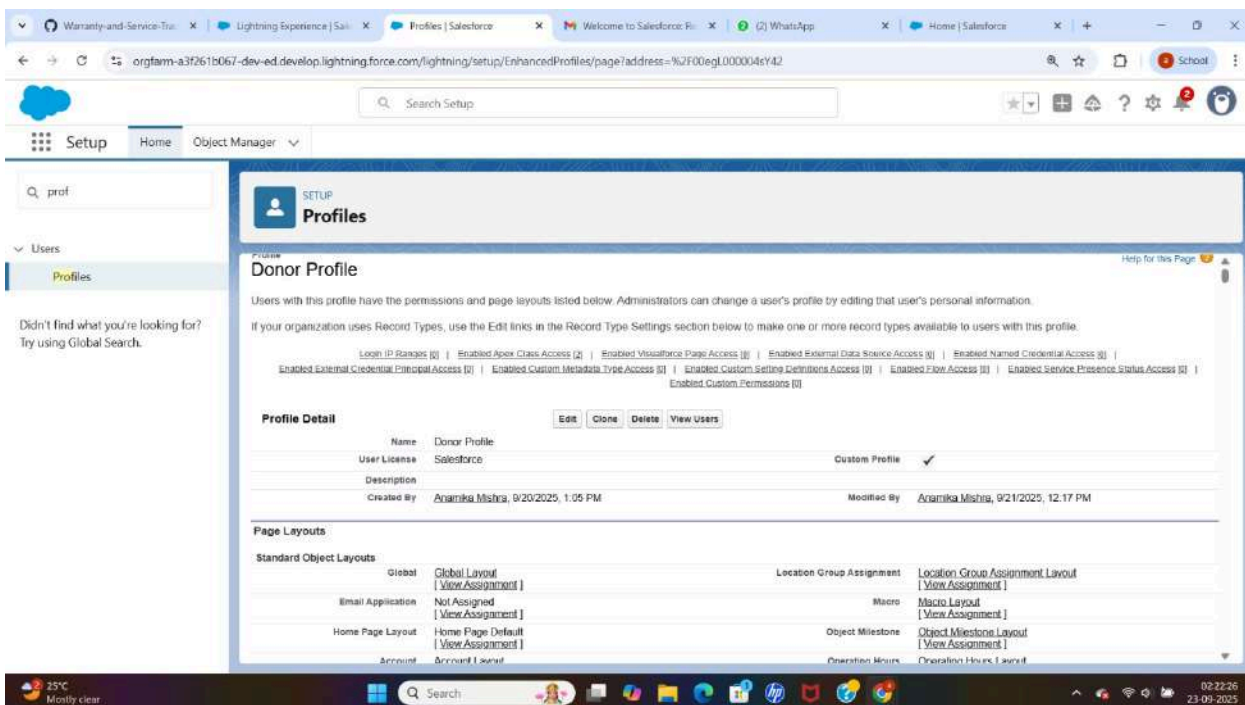
## Step 6. Profiles

1. Go to Setup → Profiles.

2. Clone a profile:

Name: Donor Profile

Permission: Donor\_\_c → Read/Edit

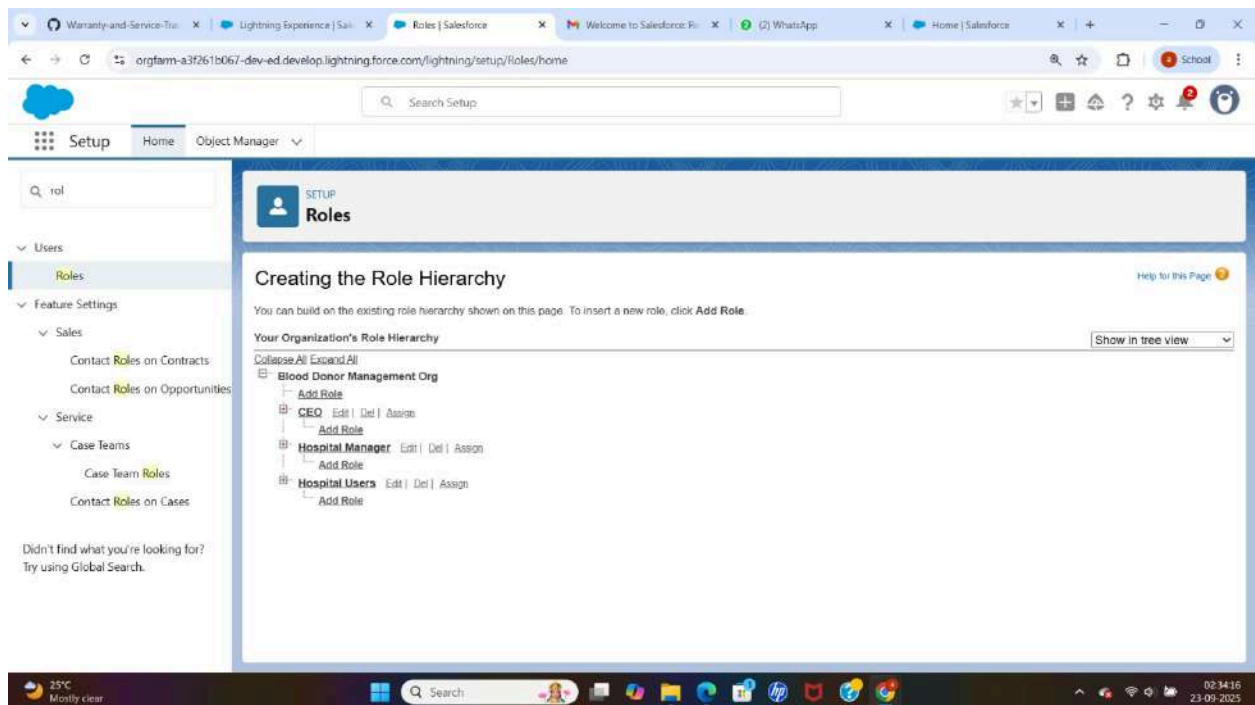


## Step 7. Roles

Role hierarchy created

Blood Donor Management Org (top)

Hospital Manager

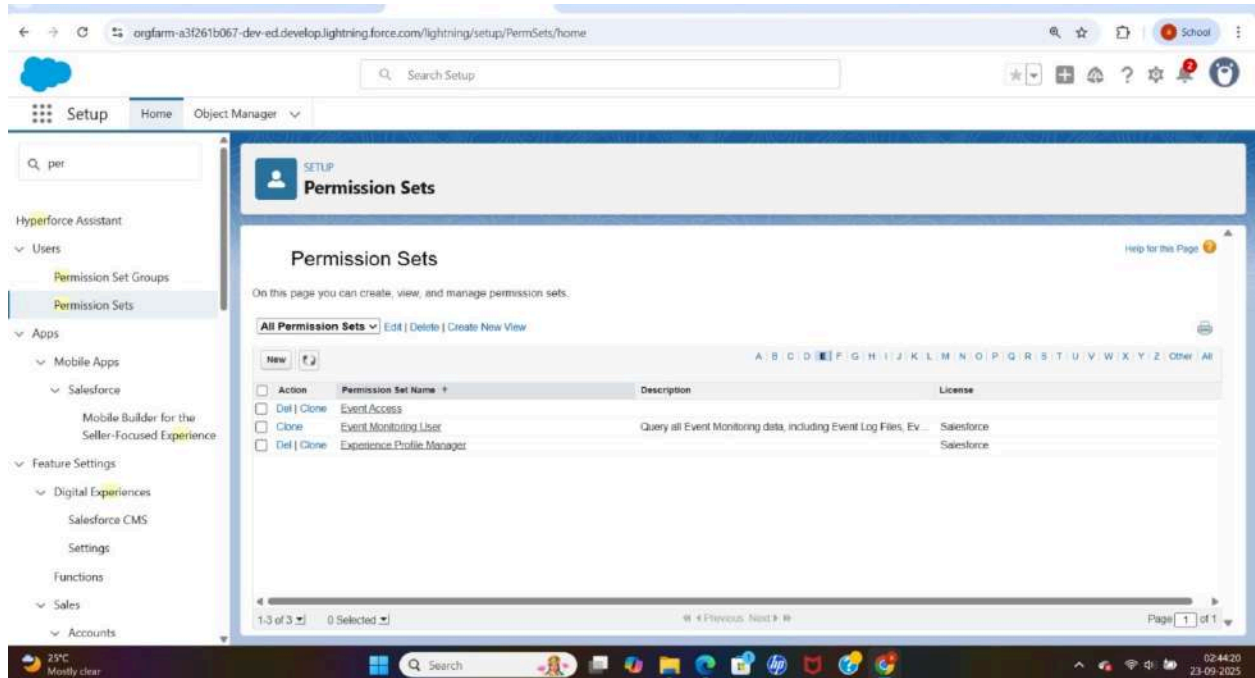


## Step 8. Permission sets

Name: Event Access

Object Settings: Donation\_Event\_\_c → Read/Write (enabled)

Assign this permission set to Hospital User



## Step 9. OWD (Organization-Wide Defaults)

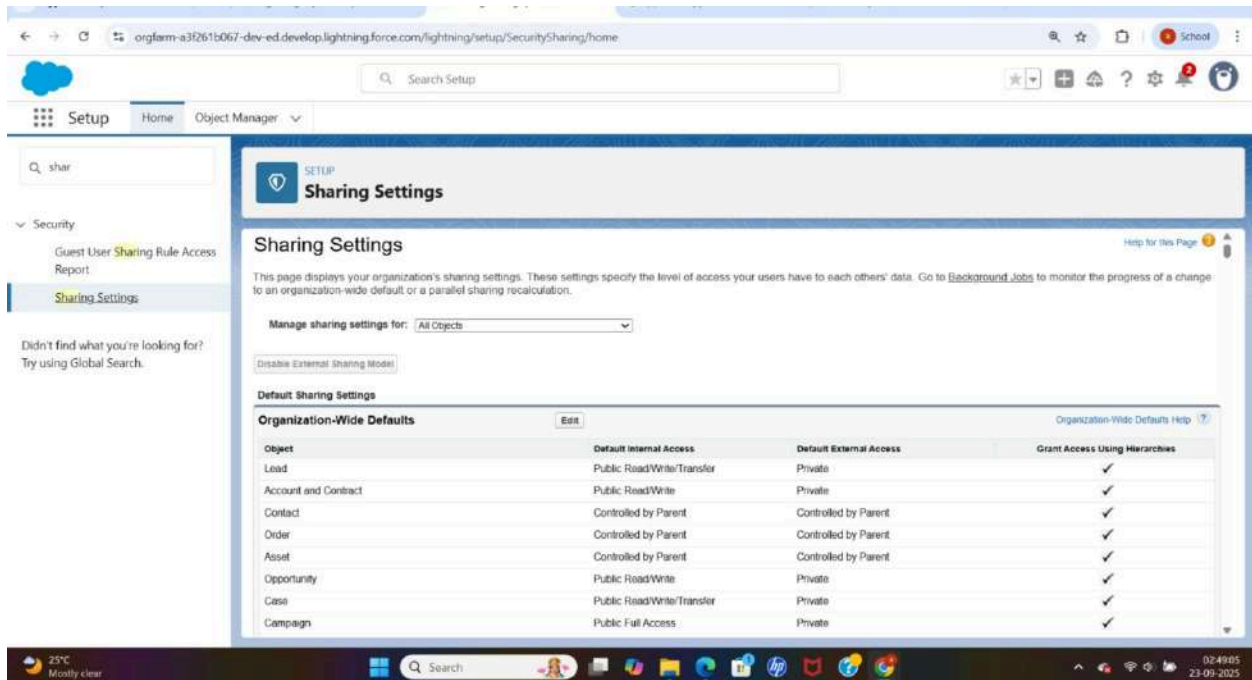
1. Go to Setup → Sharing Settings.

2. Set object access:

Donor\_\_c → Private

Donation\_Request\_\_c → Public Read-Only

Donation\_Event\_\_c → Public Read/Write



## Step 10. Sharing Rules

1. Go to Setup → Sharing Settings.

2. For Donor\_\_c, create a new sharing rule:

Name: Share Donors with Hospital Manager

Rule Type: Based on Role

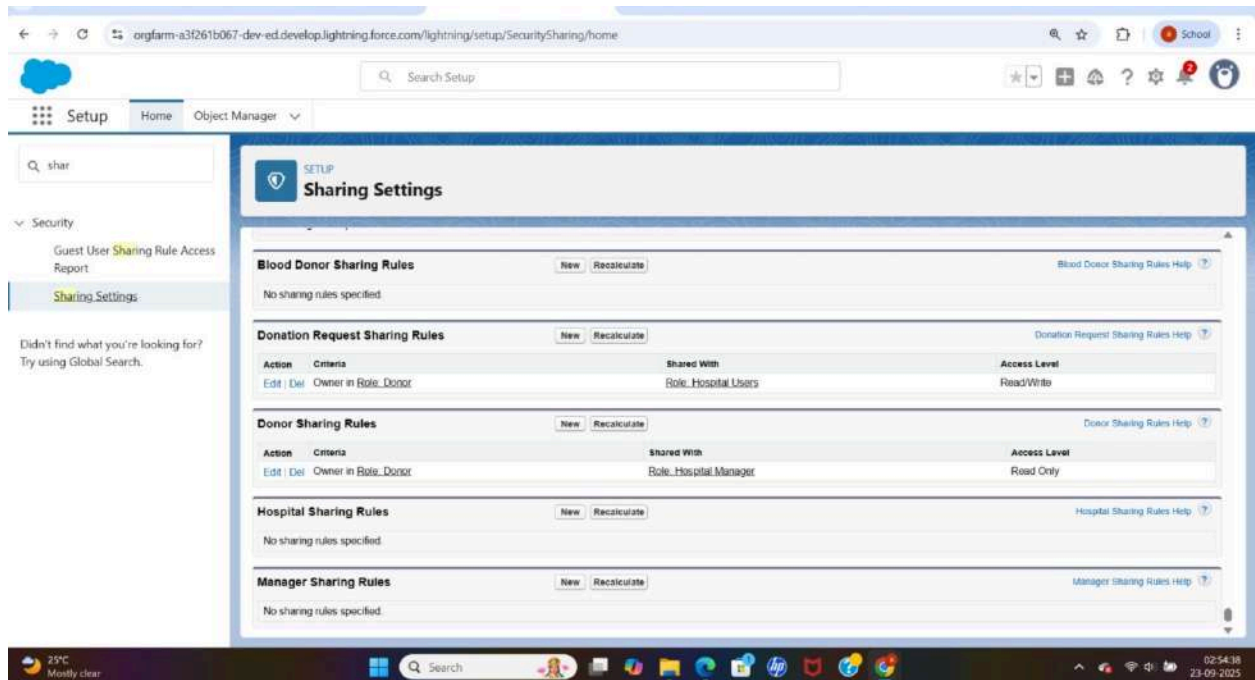
Share with: Hospital Manager

Access: Read Only

3. For Donation\_Request\_\_c, create a new sharing rule:

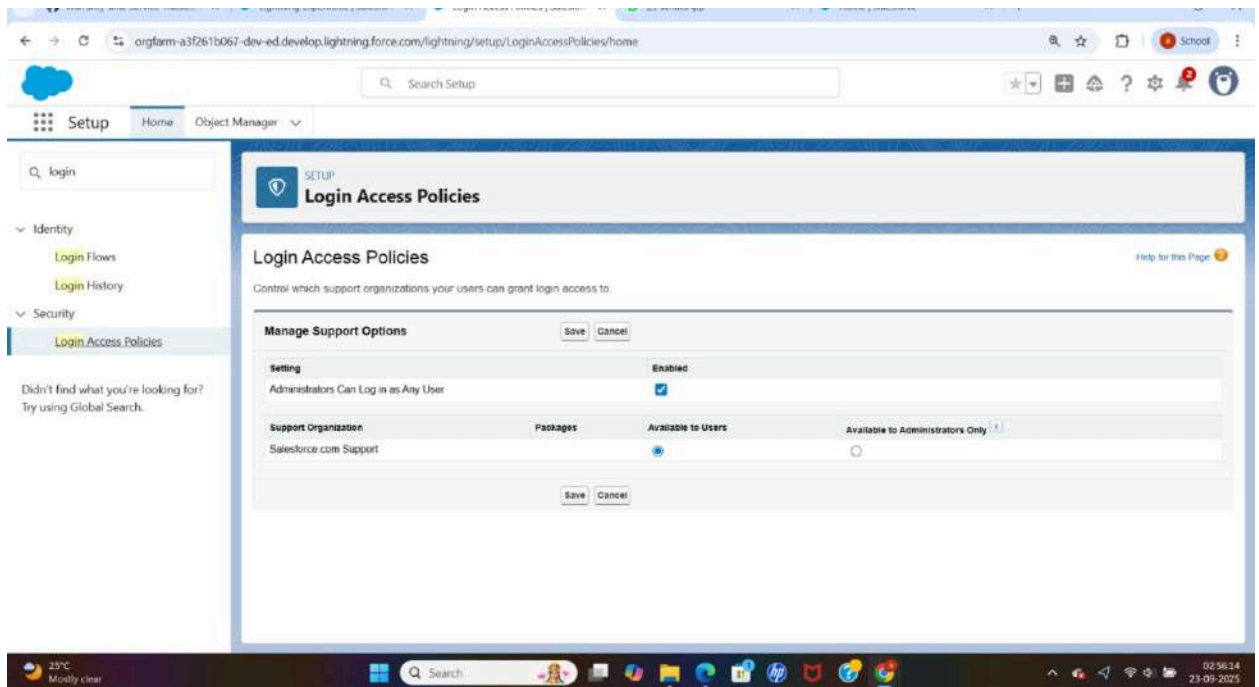
Share with: All Hospital Users

Access: Read/Write



## Step 11: Login Access Policies

1. Setup Login Access Policies
2. Enable: Administrators Can Log in as Any User



## Step 12. Developer Org Setup

Project developed in Salesforce Developer Edition Org.

Provides free Enterprise-level features suitable for learning and development purposes.

## Step 13. Sandbox Usage

Sandboxes are not available in Developer Edition.

In real-world companies, sandboxes are used for testing, development, and training without affecting Production.

## Step 14. Deployment Basics

Deployment methods include Change Sets or Salesforce CLI / VS Code.

Not required here since the project is fully developed and tested within a single Developer Org.

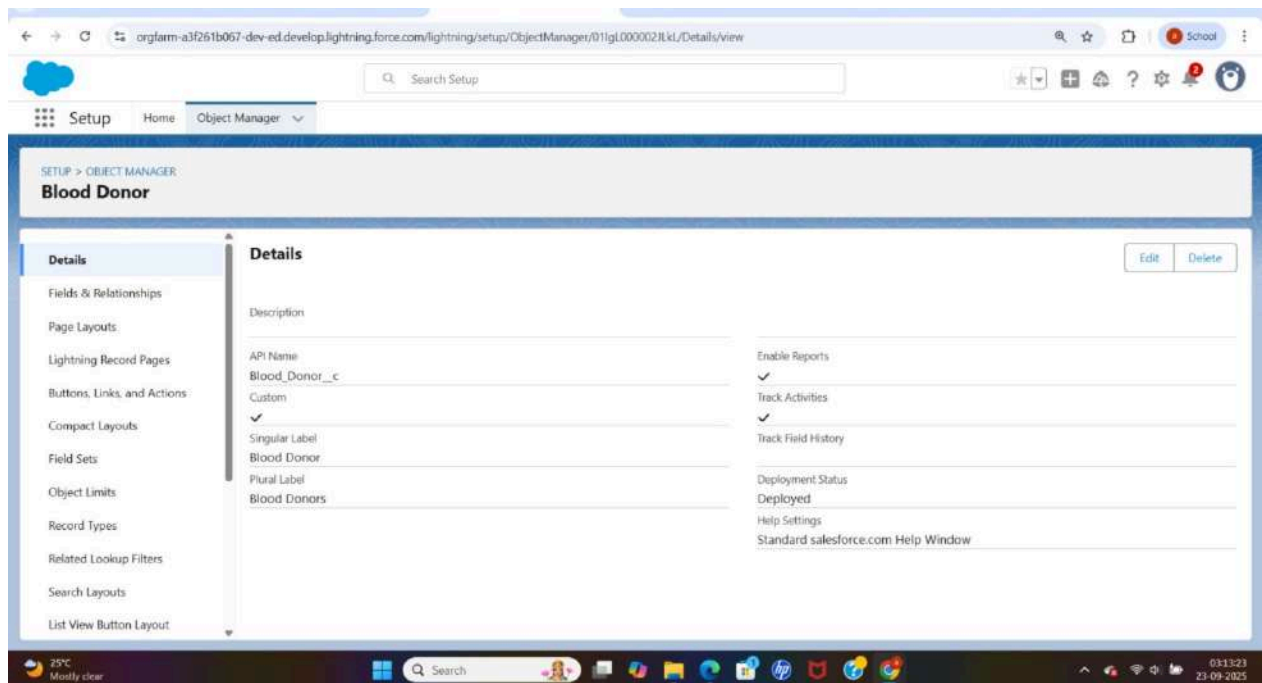
## Phase 3: Data Modeling & Relationships

### Step 1. Standard & Custom Objects

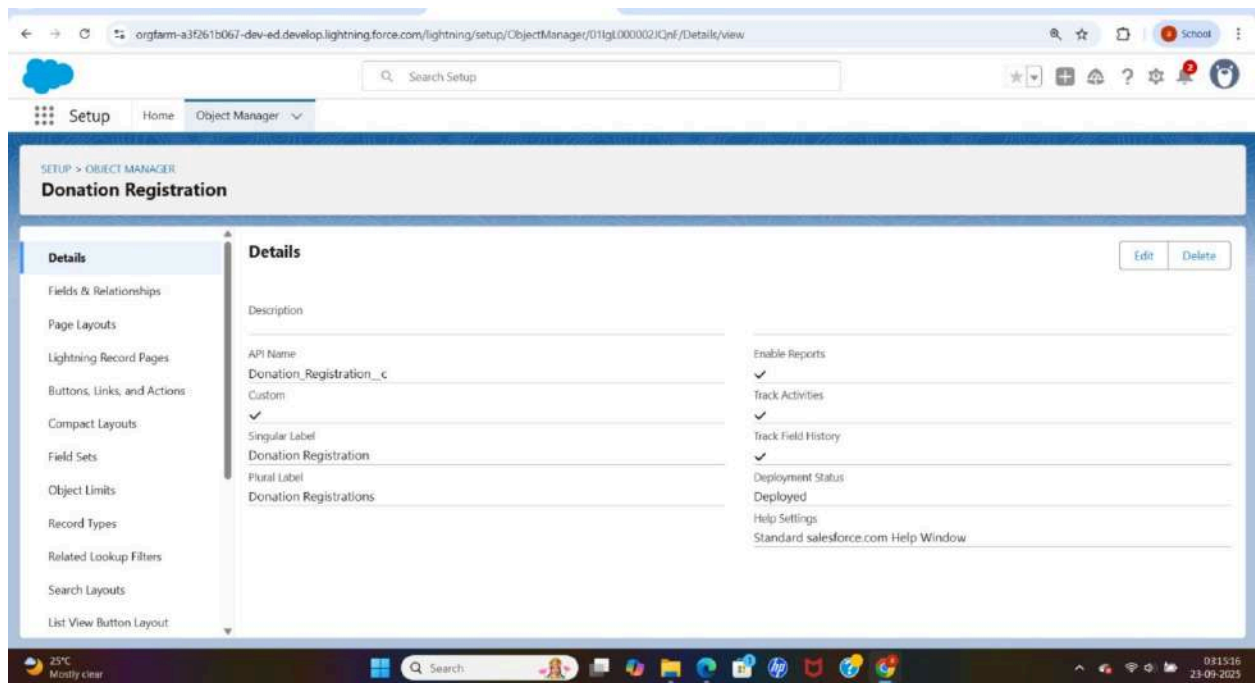
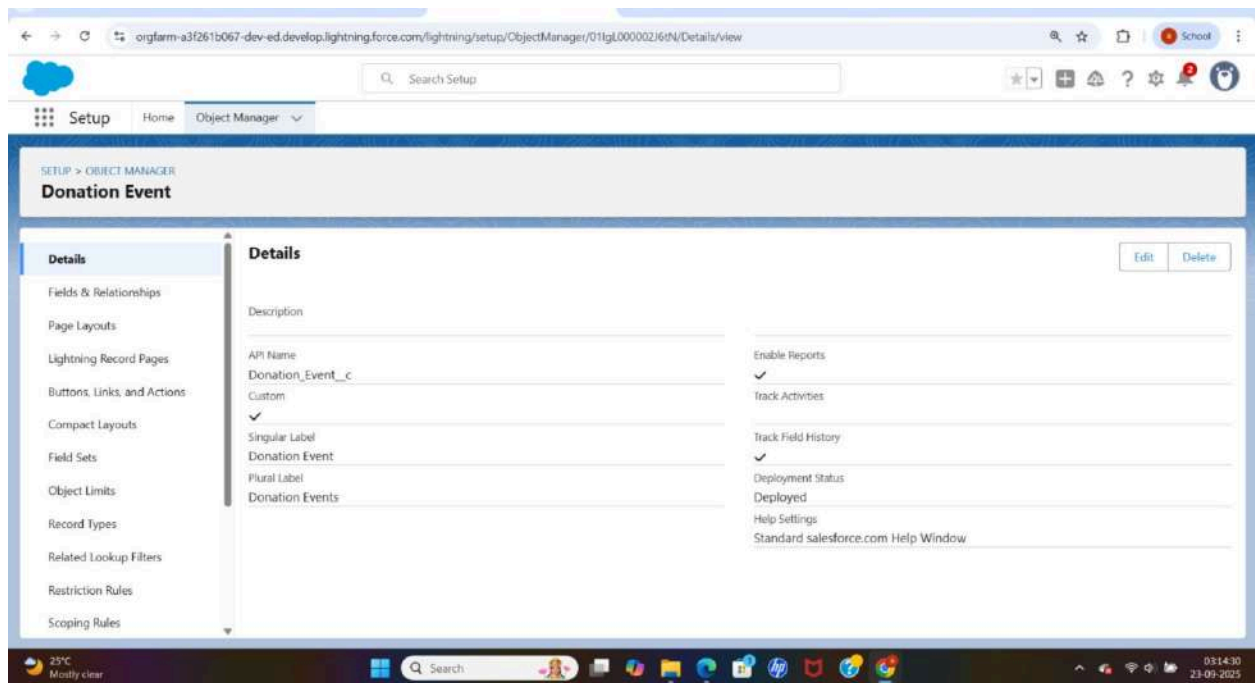
Go to Setup → Object Manager → Create Custom Object.

Create these objects:

- Blood Donor
- Donation Event
- Donation Registration







## Step 2. Fields

1. Go to Object Manager → [Object] → Fields & Relationships → New.



## 2. Blood Donor fields:

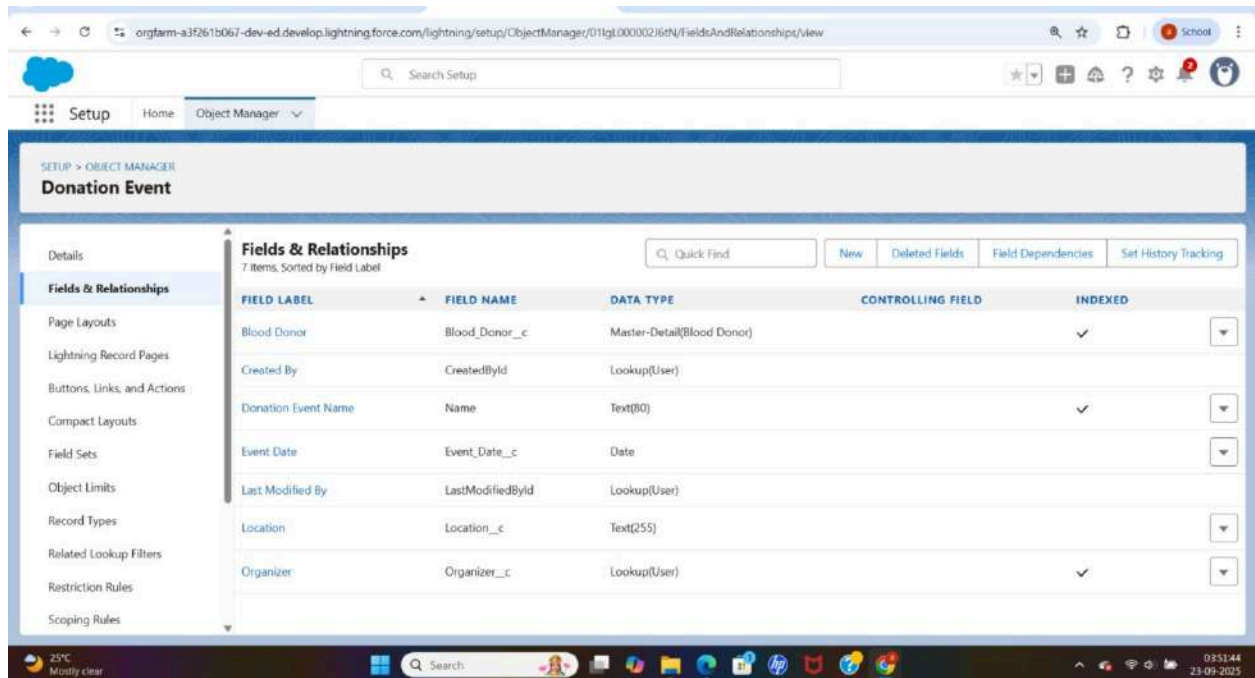
- Blood Type → Picklist
- Age → Number
- Gender → Picklist
- Last Donation Date → Date
- Contact Number → Phone
- Hospital → Lookup

The screenshot shows the Salesforce Setup interface for the 'Blood Donor' object. The 'Fields & Relationships' section is active, displaying a table of 11 fields. The table has columns for Field Label, Field Name, Data Type, Controlling Field, and Indexed. The fields listed are: Age (Number), Blood Donor (Lookup), Blood Donor Name (Text), Blood Type (Picklist), Contact Number (Phone), Created By (Lookup), Gender (Picklist), and Hospital (Lookup). The 'Blood Donor' field is highlighted in blue.

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Age	Age_c	Number(3, 0)		
Blood Donor	Blood_Donor_c	Lookup(Donor)		✓
Blood Donor Name	Name	Text(80)		✓
Blood Type	Blood_Type_c	Picklist		
Contact Number	Contact_Number_c	Phone		
Created By	CreatedById	Lookup(User)		
Gender	Gender_c	Picklist		
Hospital	Hospital_c	Lookup(Hospital)		✓

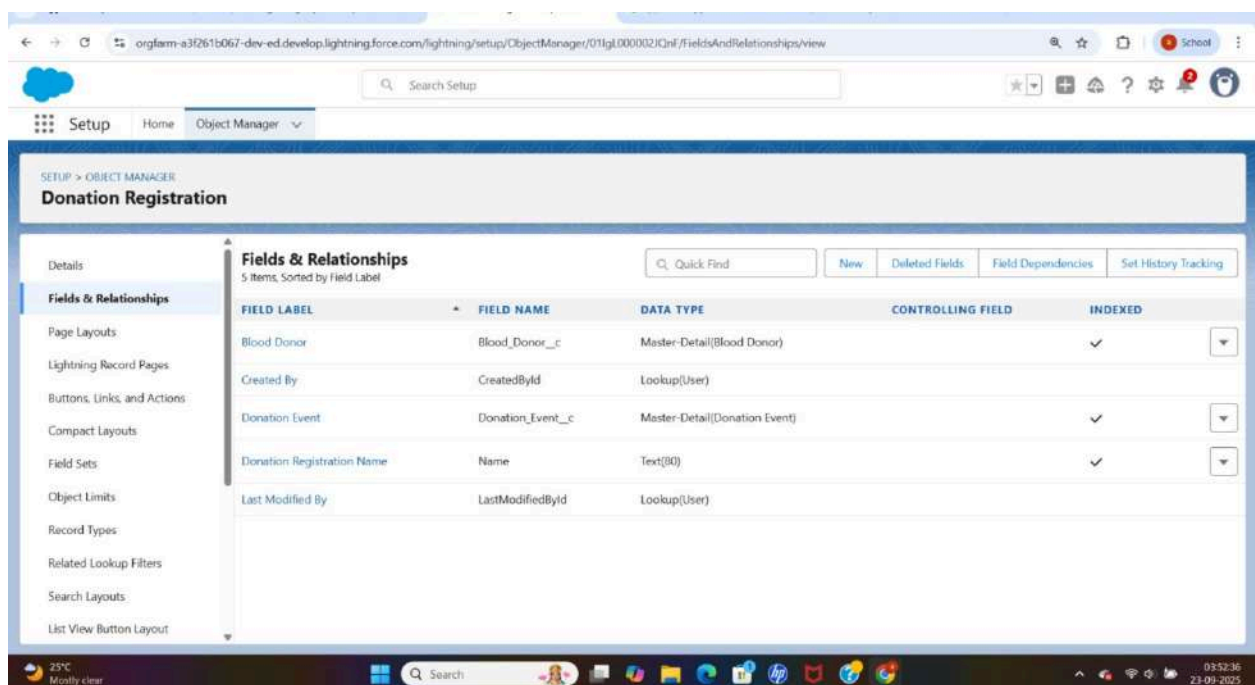
## 3. Donation Event fields:

- Event Date → Date
- Location → Text.
- Organizer → Lookup



#### 4. Donation Registration fields:

- Donor → Master-Detail (Blood Donor)
- Event → Master-Detail (Donation Event)



## Step 3. Record Types

1. Record Types = different layouts/picklists.

2. Blood Donor → Individual & Corporate Donor.
3. Setup → Object Manager → Record Types → New.

The screenshot shows the Salesforce Setup interface. The left sidebar contains a navigation menu with options: Details, Fields & Relationships, Page Layouts, Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types (highlighted), Related Lookup Filters, Search Layouts, and List View Button Layout. The main content area is titled 'Blood Donor' and 'Record Type Individual Donor'. It includes an 'Edit' button and a description: 'For personal blood donors'. Below this is a table titled 'Picklists Available for Editing' with columns 'Action', 'Field', and 'Modified Date'. The table lists two picklists: 'Blood Type' and 'Gender', both modified on 9/23/2025 at 4:23 AM. The bottom of the screen shows a Windows taskbar with a search bar and various application icons.

Action	Field	Modified Date
Edit	Blood Type	9/23/2025, 4:23 AM
Edit	Gender	9/23/2025, 4:23 AM

The screenshot shows the Salesforce Setup interface for the 'Corporate Donor' record type. The left sidebar is identical to the previous screenshot. The main content area is titled 'Blood Donor' and 'Record Type Corporate Donor'. It includes an 'Edit' button and a description: 'For organizations donating blood as a group'. Below this is a table titled 'Picklists Available for Editing' with columns 'Action', 'Field', and 'Modified Date'. The table lists two picklists: 'Blood Type' and 'Gender', both modified on 9/23/2025 at 4:26 AM. The bottom of the screen shows a Windows taskbar with a search bar and various application icons.

Action	Field	Modified Date
Edit	Blood Type	9/23/2025, 4:26 AM
Edit	Gender	9/23/2025, 4:26 AM

## Step 4. Page Layouts

Setup: Object Manager → Object → Page Layouts → New Sections

Blood Donor - Personal Information, Blood Type , Hospital, Last Donation Date

The screenshot shows the Salesforce Lightning Setup interface for the 'Blood Donor' object. The left sidebar contains a navigation menu with options: Details, Fields & Relationships, Page Layouts (selected), Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Search Layouts, and List View Button Layout. The main content area is titled 'Blood Donor' and displays the 'Page Layouts' configuration. A 'New Section' is being created with the following fields: Blood Donor Name, Blood Type, Gender, Age, Contact Number, Hospital, and Last Donation Date. The 'Owner' field is set to 'Sample Text'. The 'Quick Find' search bar is visible at the top of the main content area.

Donation Event - Event Information, Organizer, Location

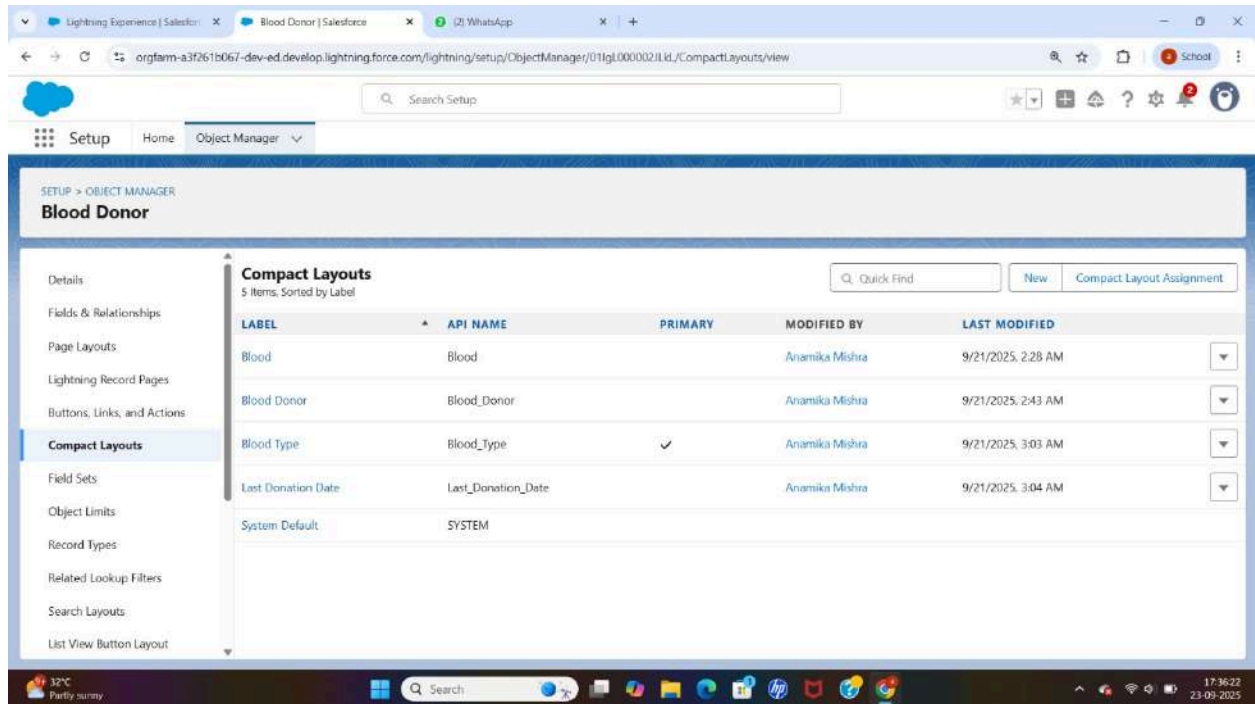
The screenshot shows the Salesforce Lightning Setup interface for the 'Donation Event' object. The left sidebar contains a navigation menu with options: Details, Fields & Relationships, Page Layouts (selected), Lightning Record Pages, Buttons, Links, and Actions, Compact Layouts, Field Sets, Object Limits, Record Types, Related Lookup Filters, Restriction Rules, and Scoping Rules. The main content area is titled 'Donation Event' and displays the 'Page Layouts' configuration. A 'New Section' is being created with the following fields: Donation Event Name, Event Date, Location, Organizer, and Blood Donor. The 'Owner' field is set to 'Sample Text'. The 'Quick Find' search bar is visible at the top of the main content area.

## Step 5. Compact Layouts

Setup: Object Manager → [Object] → Compact Layouts → New

Example: Blood Donor Last Donation Date

Fields: Name, Blood Type



## Step 6. Schema Builder

Go to Setup → Schema Builder.

Drag objects onto canvas.

Create Master-Detail and Lookup relationships visually.



orgfarm-a3f261b067-dev-ed.develop.lightning.force.com/lightning/setup/SchemaBuilder/home

Search Setup

Setup Home Object Manager

### Schema Builder

Close Auto-Layout View Options

**Elements** Objects

Drag and drop new custom objects and fields into the schema.

**OBJECT**

- Object

**FIELDS**

- Auto Number
- Formula
- Roll-Up Summary
- Lookup
- Hierarchy
- Master-Detail
- Checkbox
- Currency
- Date
- Date/Time
- Time
- Email
- Number
- Percent

**Legend**

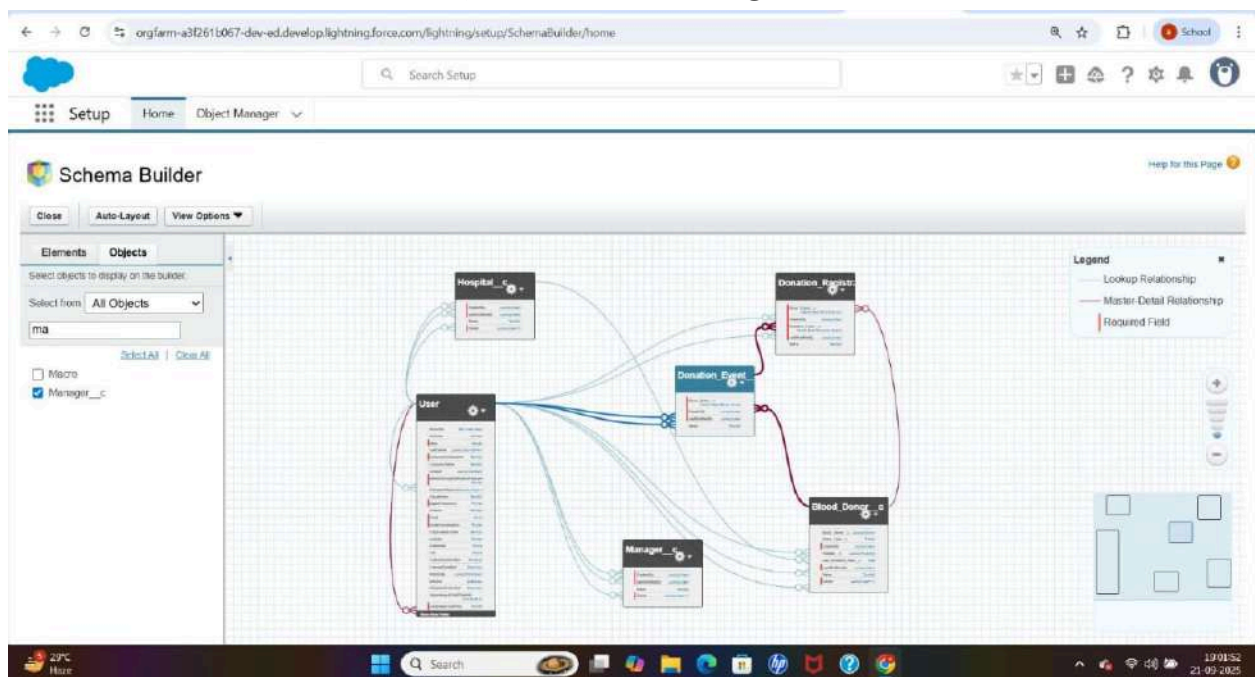
- Lookup Relationship
- Master-Detail Relationship
- Required Field

```
graph TD; Hospital_c --> Blood_Donor__c; Hospital_c --> Donor__c; Blood_Donor__c --> Donation_Event__c;
```

## Step 7. Lookup vs Master-Detail vs Hierarchical Relationships

- Relationship Type - Lookup

- Example - Blood Donor Hospital
  - Reason for Choice - Donor can exist independently
- 
- Relationship Type - Master-Detail
  - Example - Donation Registration      Donor
  - Reason for Choice - Strong dependency
- 
- Relationship Type - Master-Detail
  - Example - Donation Registration Event
  - Reason for Choice - Strong dependency
- 
- Relationship Type - Hierarchical
  - Example - User Manager
  - Reason for Choice - Only for User reporting



## Step 8. Junction Objects

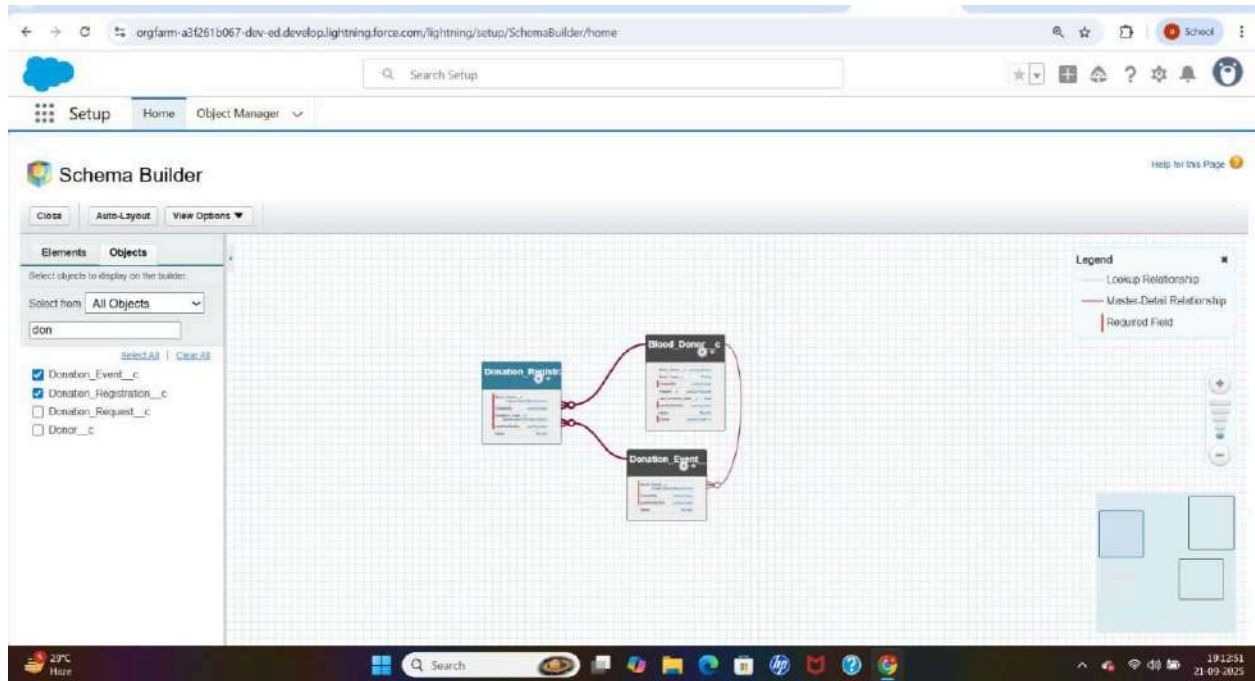
Used for many-to-many relationships

Donor - Donation Event

Junction Object - Donation Registration

Master-Detail to Blood Donor

Master-Detail to Donation Event



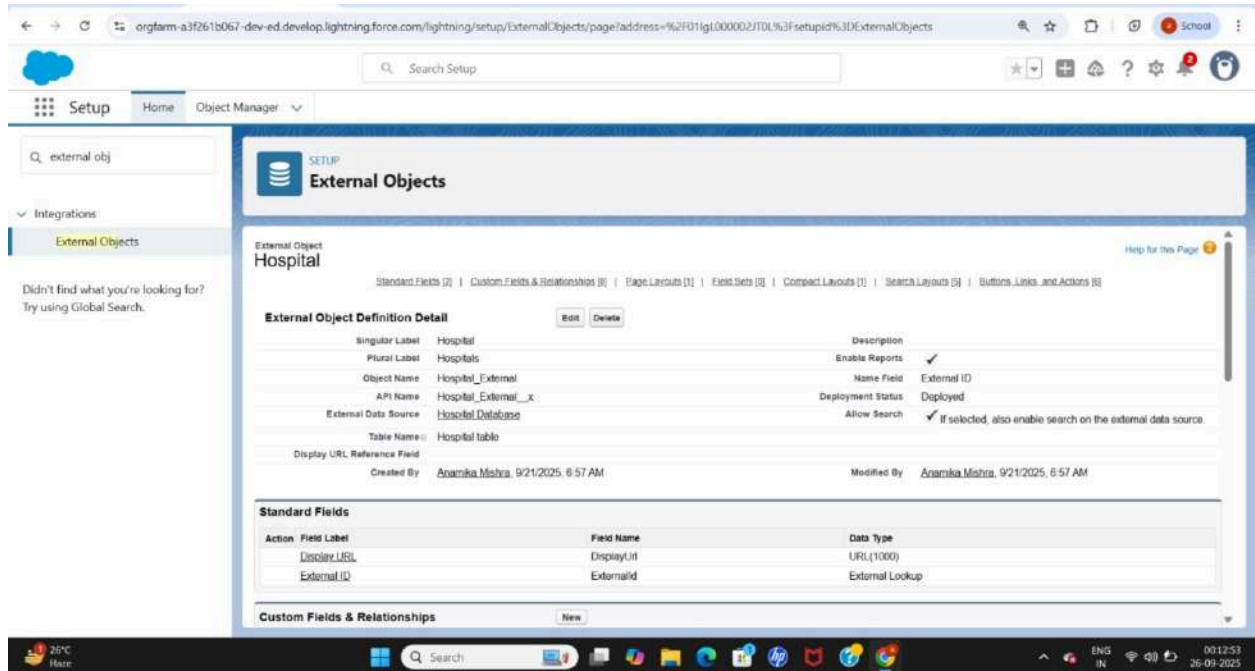
## Step 9. External Objects

Use to access data from external systems without importing it into Salesforce.

Example: Connect to a hospital database outside Salesforce.

Setup: Setup → External Objects → Create → Connect using Salesforce Connect.

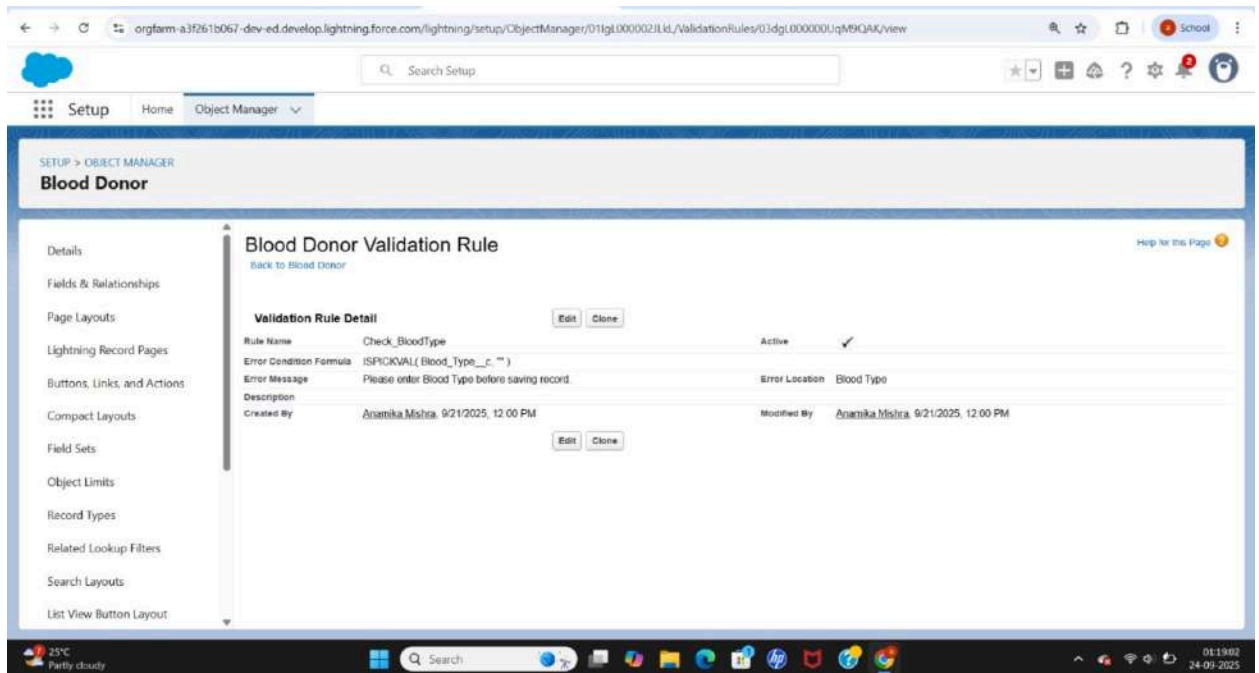




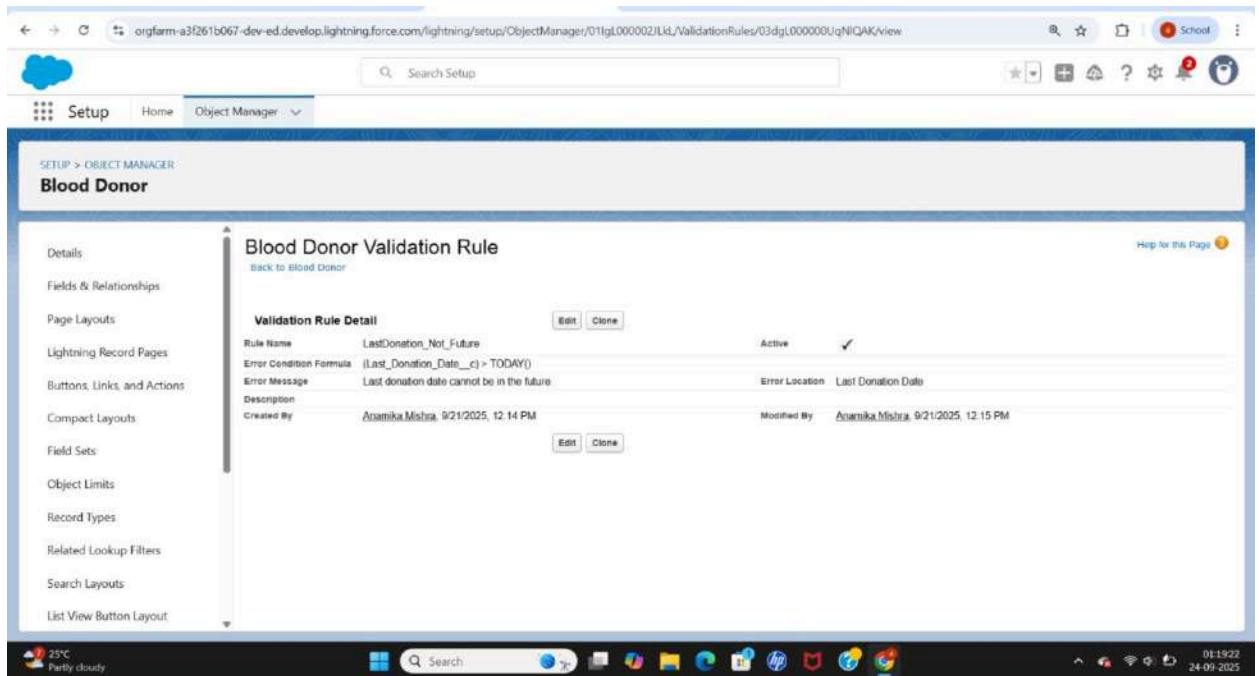
## Phase 4: Process Automation (Admin)

### Step 1. Validation Rules

- Rule Name: Check\_BloodType
- Formula: `ISPICKVAL(Blood_Type__c, "")`
- Error Message: "Please enter Blood Type before saving."
- Error Location: Field → Blood Type



- Rule Name: LastDonation\_Not\_Future
- Formula: Last\_Donation\_Date\_\_c > TODAY()
- Error Message: Last donation date cannot be in the future.
- Error Location: Field → Last Donation Date



## Step 2. Workflow Rules

Rule Name: Donor\_Reminder\_90days

Evaluation: Created, and edited to meet criteria

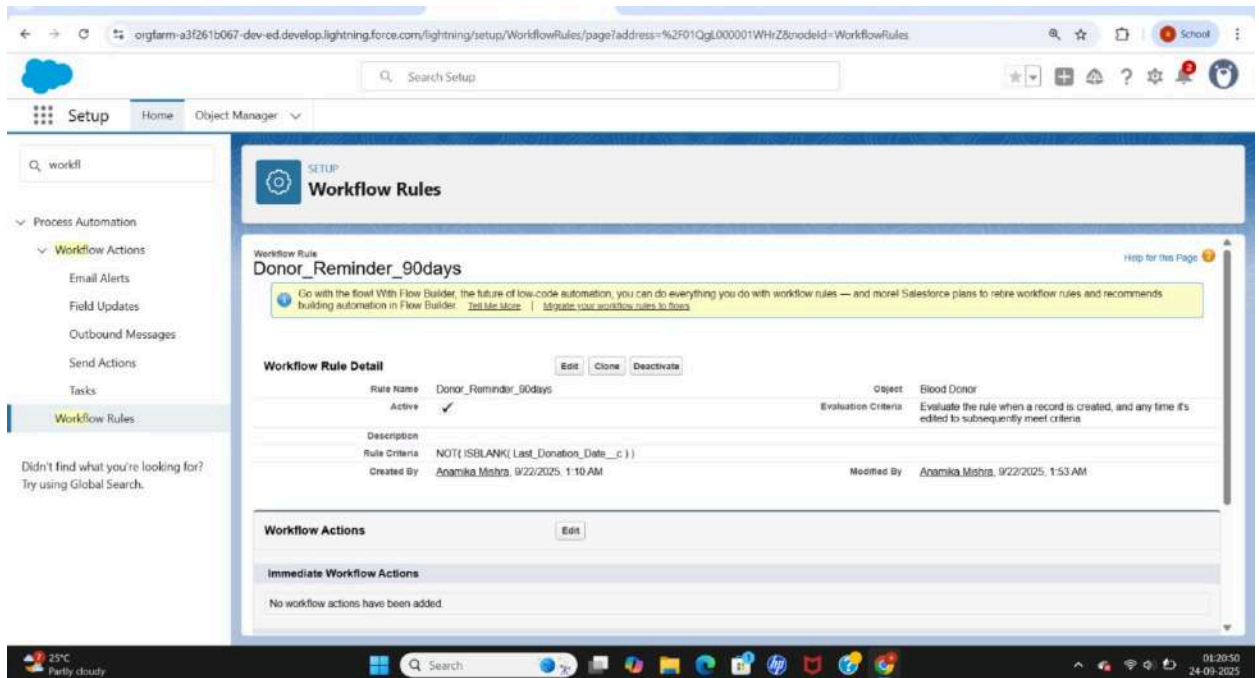
Criteria (Formula): NOT(ISBLANK(Last\_Donation\_Date\_\_c))

Time Trigger: 90 days after Last\_Donation\_Date\_\_c

Action: Email Alert → Template: Donor\_Reminder\_Template

Subject: Time for Your Next Blood Donation

Recipient: Donor Email (Email\_\_c)



## Step 3. Process Builder

### 1. Create Process

Name: Update Donor Last Donation Date

Starts: When a record changes

### 2. Select Object

Object: Donation Registration

Start when: Record is created or edited

### 3. Define Criteria

Name: Donation Completed

Condition: Donation Status = Completed

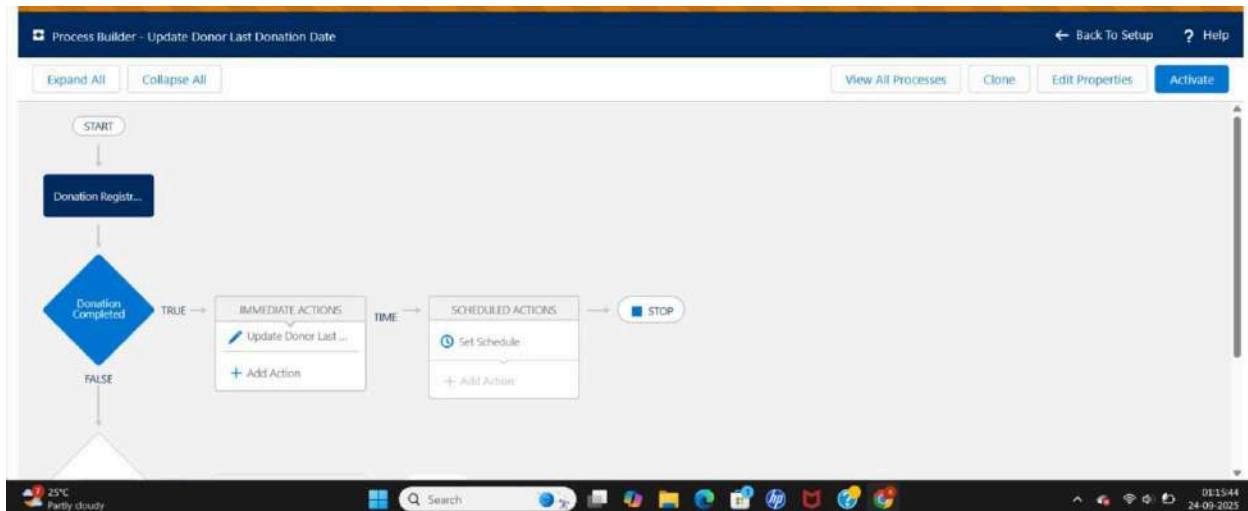
Execute only when specified changes are made: Yes

### 4. Define Action

Action Type: Update Records

Update: Related Donor record

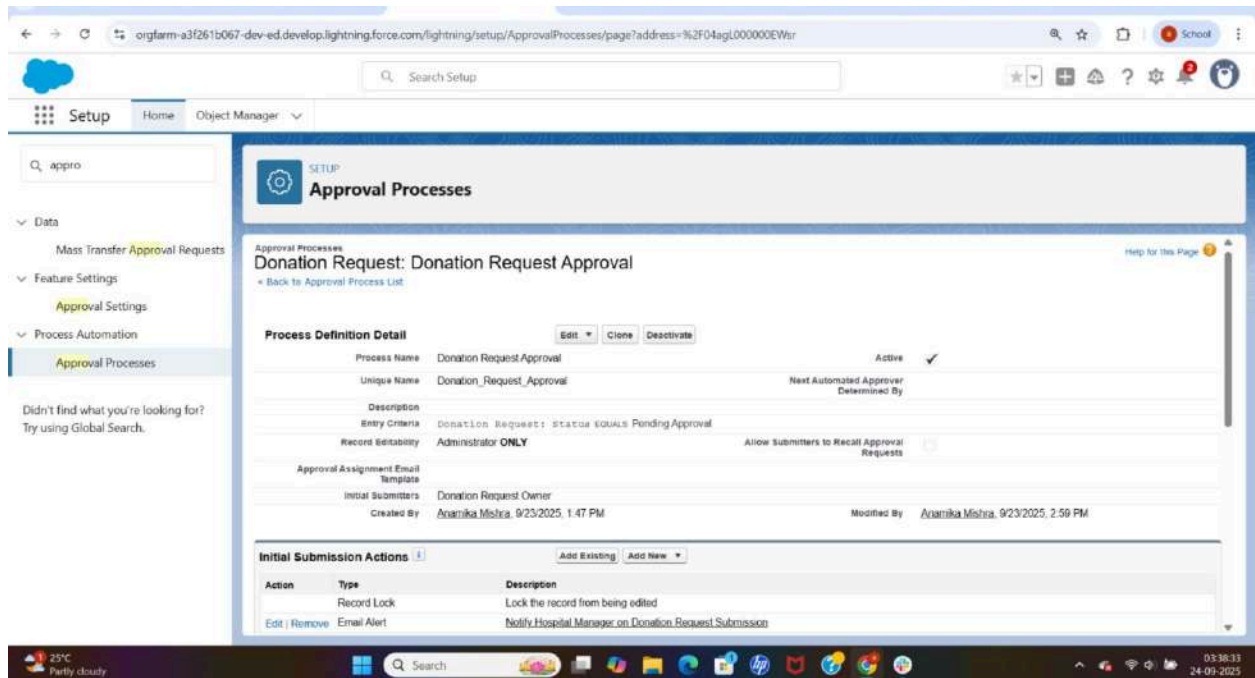
Field: Last Donation Date = Donation Registration.Donation Date



## Step 4. Approval Process

### Donation Request Approval Process

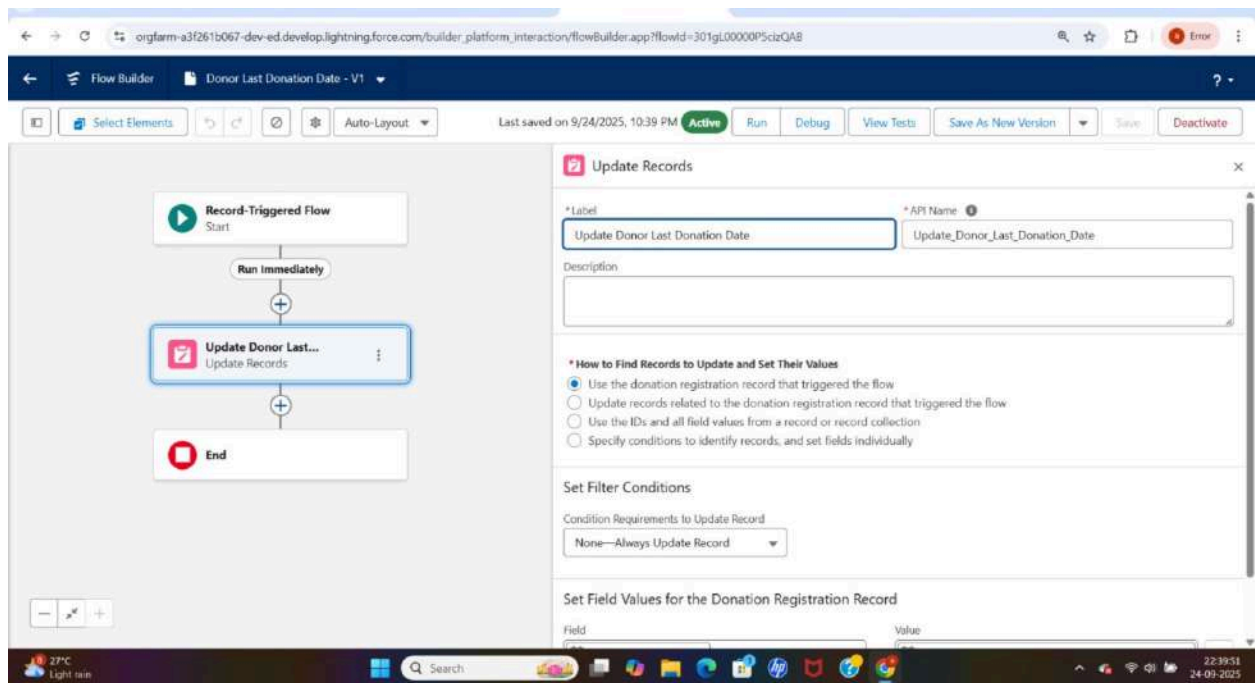
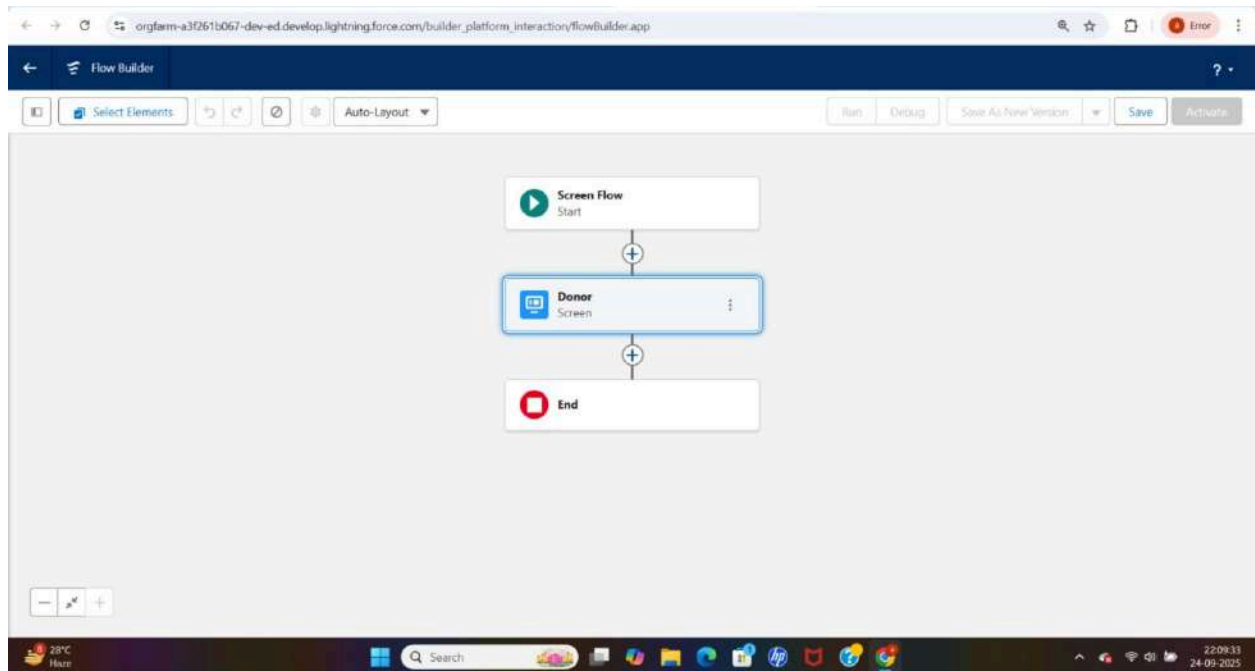
1. Setup: Approval Processes → Donation\_Request\_\_c → Jump Start Wizard.
2. Name & Entry: Donation Request Approval; records with Status\_\_c = "Pending Approval".
3. Approver: Hospital Manager.
4. Actions:
  - On submit: Lock record
  - If approved: Status → Approved, notify owner
  - If rejected: Status → Rejected, notify requester
5. Activate: Process ready for use.

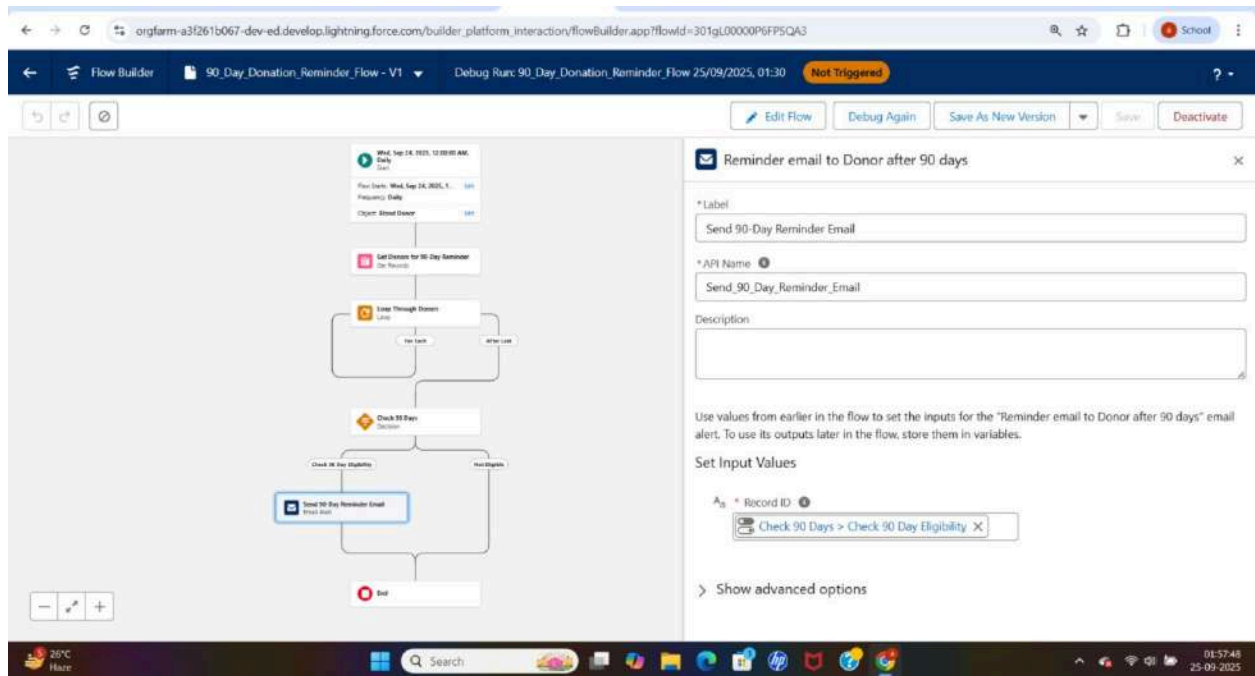


## Step 5. Flow Builder (Screen, Record-Triggered, Scheduled, Auto-Launched)

### Use Cases in Blood Donor Management

1. Screen Flow: Register a new donor (collect name, age, blood group, contact info).
2. Record-Triggered Flow: Update donor's Last Donation Date whenever a donation record is added.
3. Scheduled Flow: Send reminder emails after 90 days of last donation.
4. Auto-Launched Flow: Assign donors to nearest hospital automatically.





## Step 6. Email Alerts

1. Setup → Email Alerts → New Email Alert

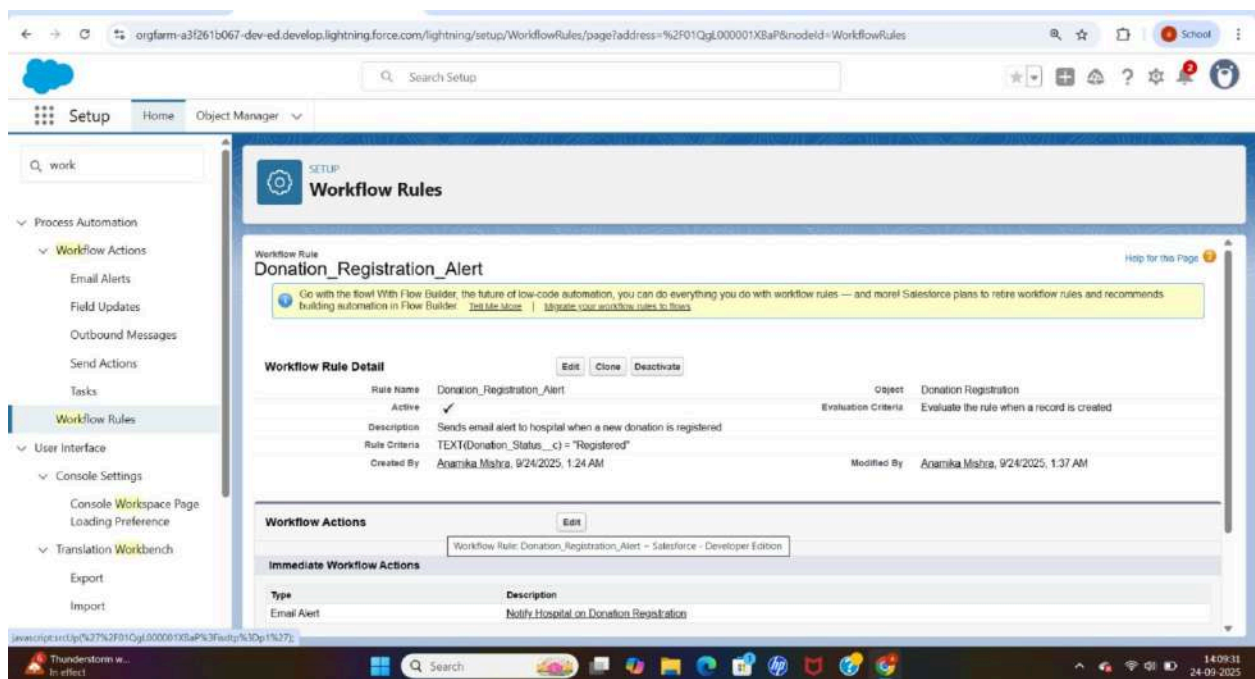
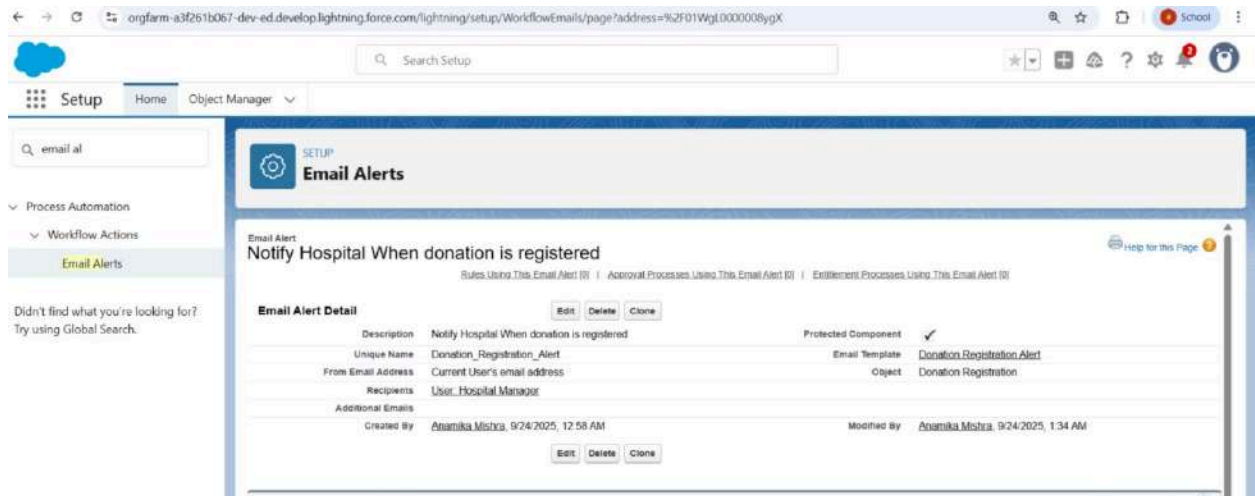
Description: Notify hospital when donation is registered

Unique Name: Donation\_Registration\_Alert

Object: Donation Registration

Choose recipients: User

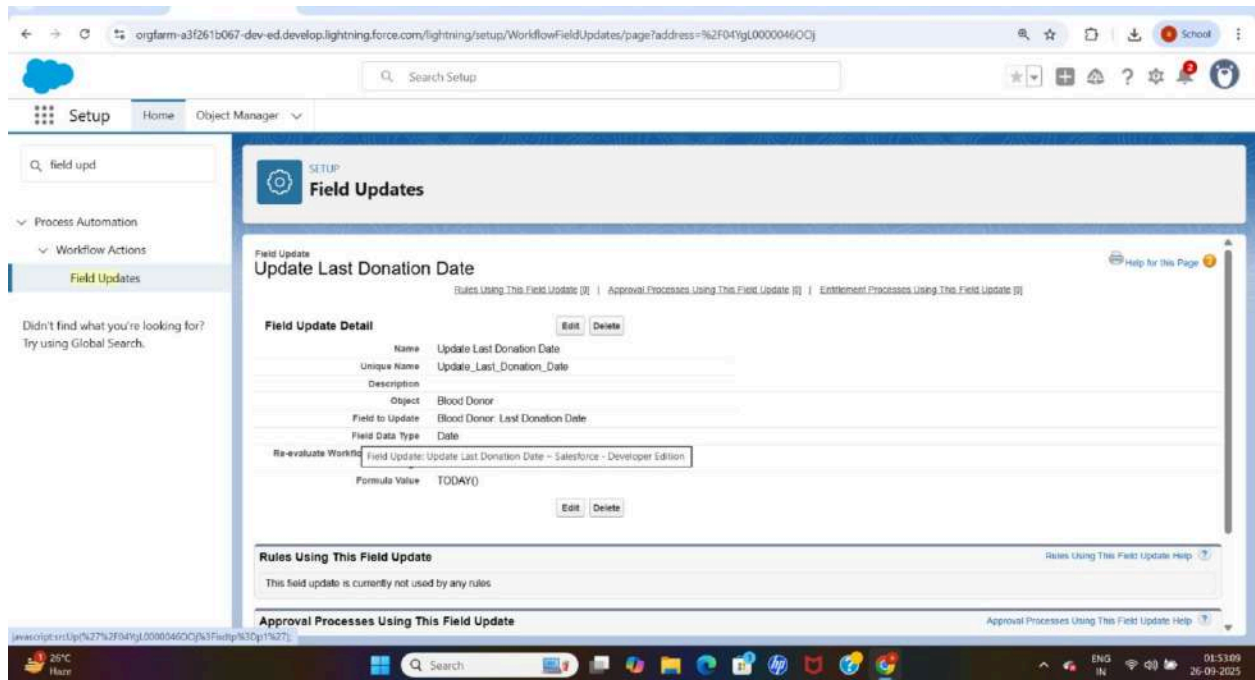




## Step 7. Field Updates

Name: Update\_Last\_Donation\_Date

Select the field to update: Choose Last Donation Date on Blood Donor object.

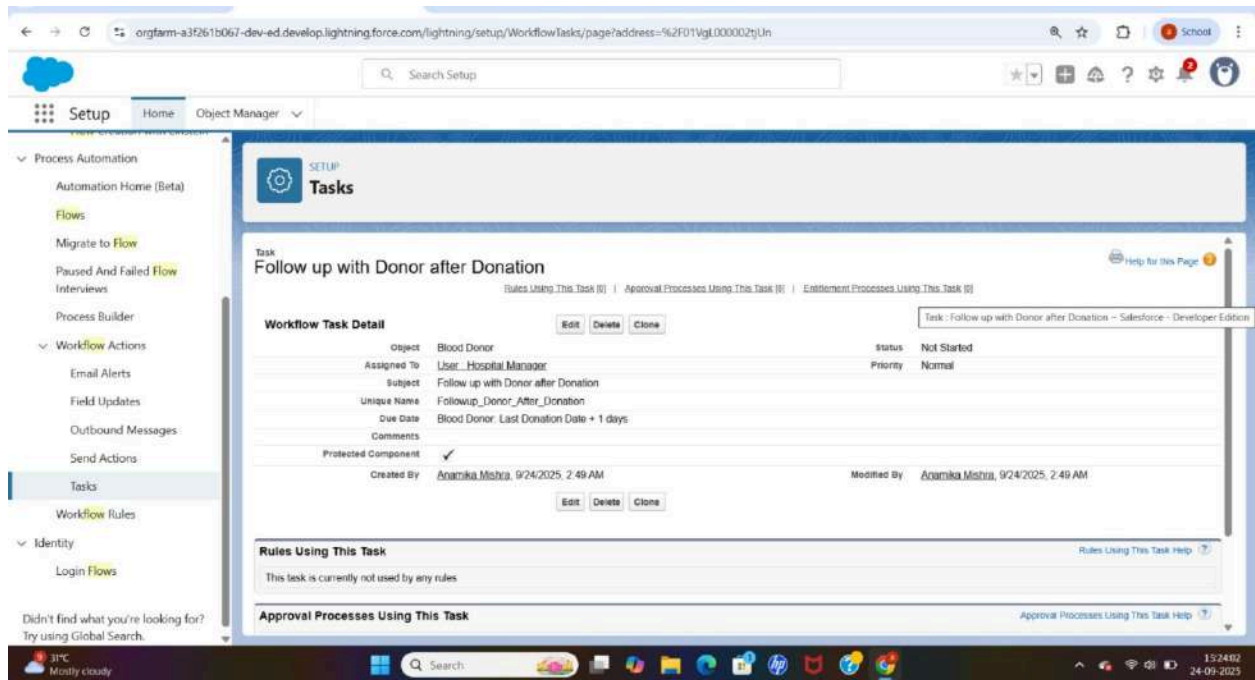


## Step 8. Tasks

Object: Blood Donor

Assigned To: Hospital Manager

Subject: Follow up with Donor after Donation



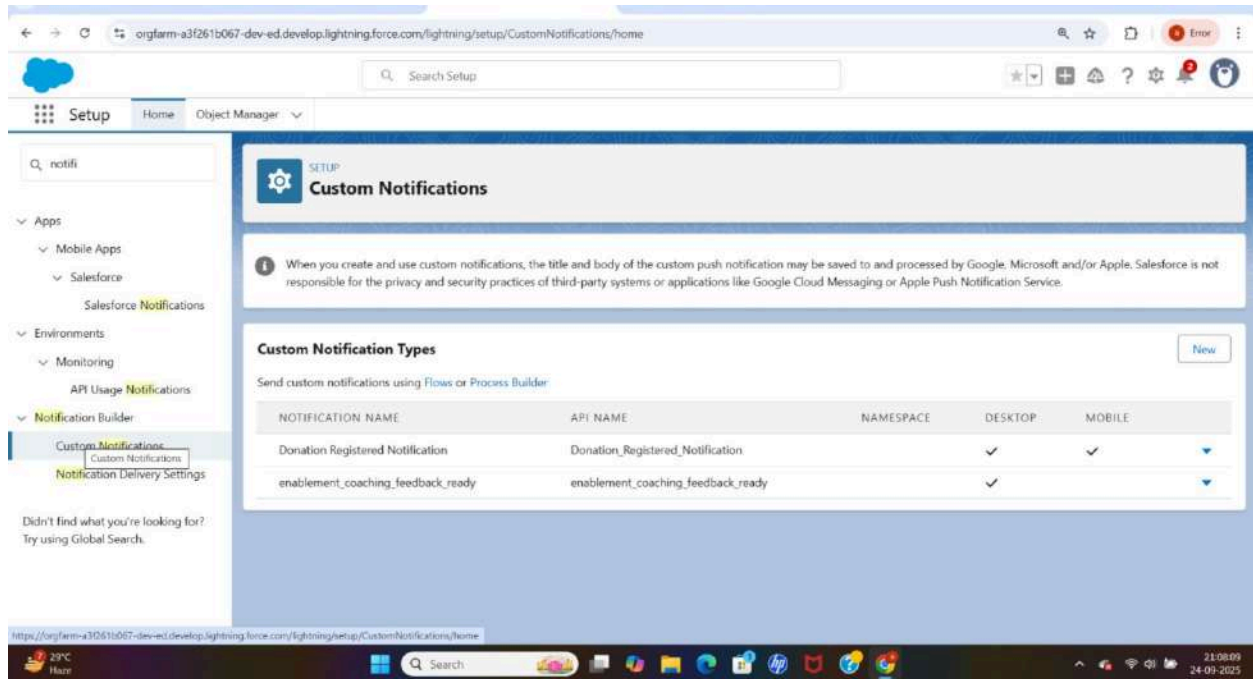
## Step 9. Custom Notifications

Name: Donation Registered Notification

Object: Donation Registration

Supported Channels -

- Desktop (for Salesforce users)
- Mobile (for Salesforce Mobile App)



## Phase 5: Apex Programming (Developer)

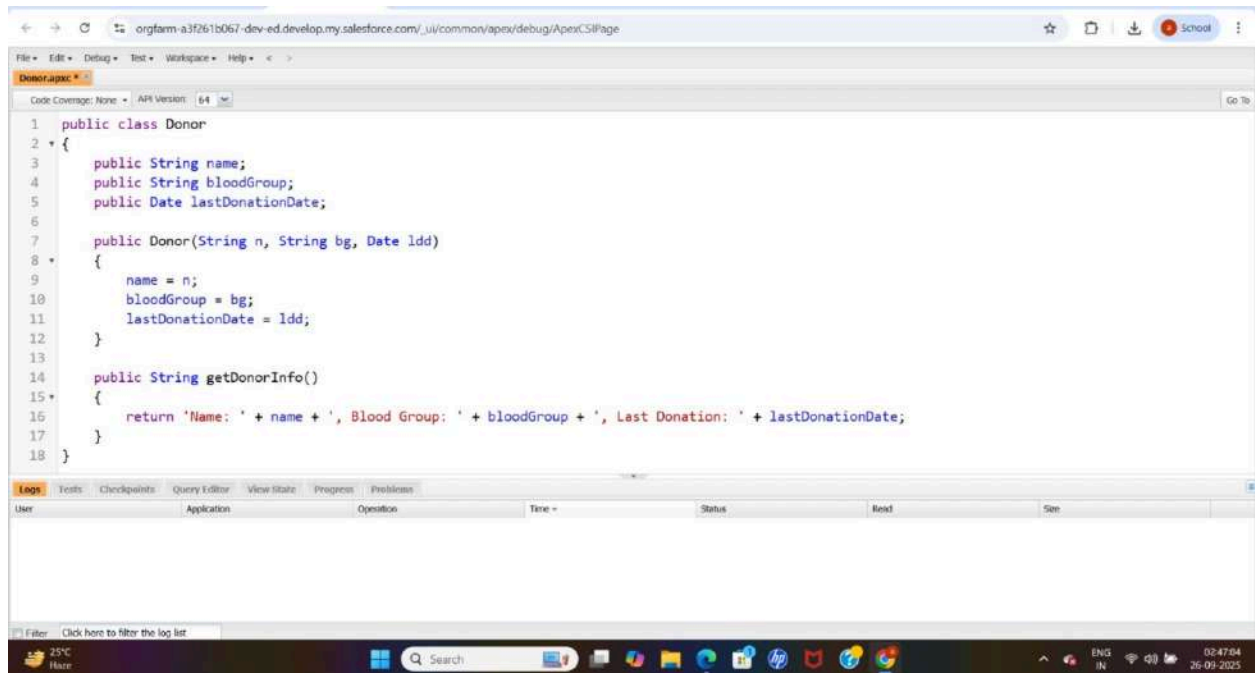
### Step 1. Classes & Objects

In Apex, a class is a blueprint that defines variables (data) and methods (functions).

An object is an instance of a class, which represents a real-world entity.

In your Blood Donor Management Project, you can use classes to represent Donors, Hospitals, Donations, and Events.

This makes the system more organized, reusable, and scalable.

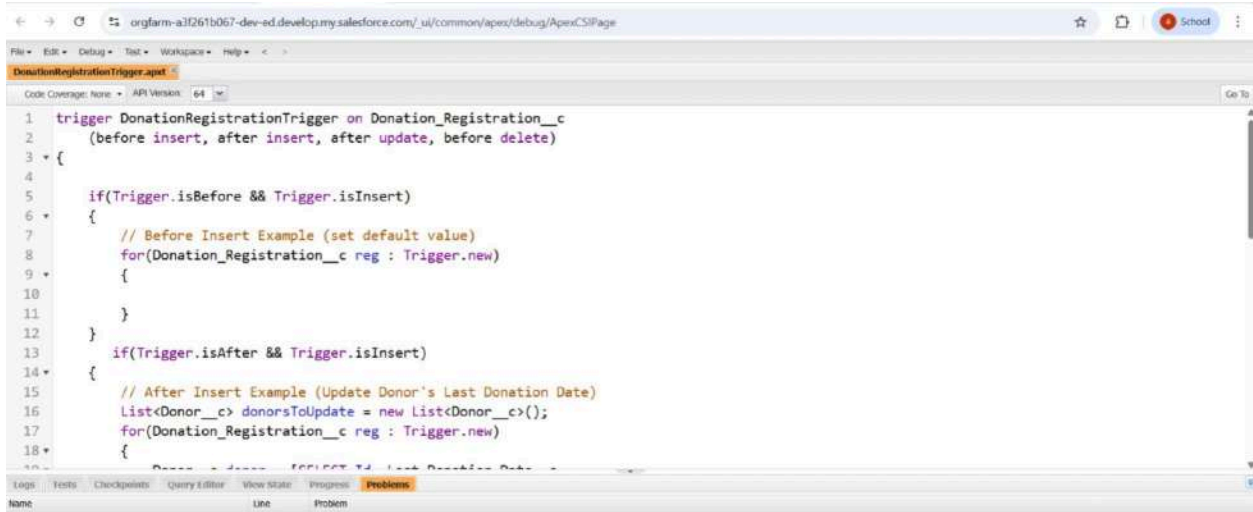


```
public class Donor
{
    public String name;
    public String bloodGroup;
    public Date lastDonationDate;

    public Donor(String n, String bg, Date ldd)
    {
        name = n;
        bloodGroup = bg;
        lastDonationDate = ldd;
    }

    public String getDonorInfo()
    {
        return 'Name: ' + name + ', Blood Group: ' + bloodGroup + ', Last Donation: ' +
lastDonationDate;
    }
}
```

Step 2. Apex Triggers (before/after insert/update/delete)



trigger DonationRegistrationTrigger on Donation\_Registration\_\_c  
(before insert, after insert, after update, before delete)

```
{
    if(trigger.isBefore && Trigger.isInsert)
    {
        // Before Insert Example (set default value)
        for(Donation_Registration__c reg : Trigger.new)
        {
        }
    }
    if(trigger.isAfter && Trigger.isInsert)
    {
        // After Insert Example (Update Donor's Last Donation Date)
        List<Donor__c> donorsToUpdate = new List<Donor__c>();
        for(Donation_Registration__c reg : Trigger.new)
        {
            Donor__c donor = [SELECT Id, Last_Donation_Date__c
                              FROM Donor__c
                              WHERE Id = :reg.Donor__c LIMIT 1];
            donor.Last_Donation_Date__c = reg.Donation_Date__c;
            donorsToUpdate.add(donor);
        }
    }
}
```

```

        if(!donorsToUpdate.isEmpty())
        {
            update donorsToUpdate;
        }
    }

    if(Trigger.isAfter && Trigger.isUpdate)
    {
        // After Update Example
        for(Donation_Registration__c reg : Trigger.new)
        {
            System.debug('Donation record updated: ' + reg.Id);
        }
    }

    if(Trigger.isBefore && Trigger.isDelete)
    {
        // Before Delete Example
        List<Donor__c> donorsToUpdate = new List<Donor__c>();
        for(Donation_Registration__c reg : Trigger.old)
        {

        }
        if(!donorsToUpdate.isEmpty())
        {
            update donorsToUpdate;
        }
    }
}

```

### Step 3. Trigger Design Pattern

```

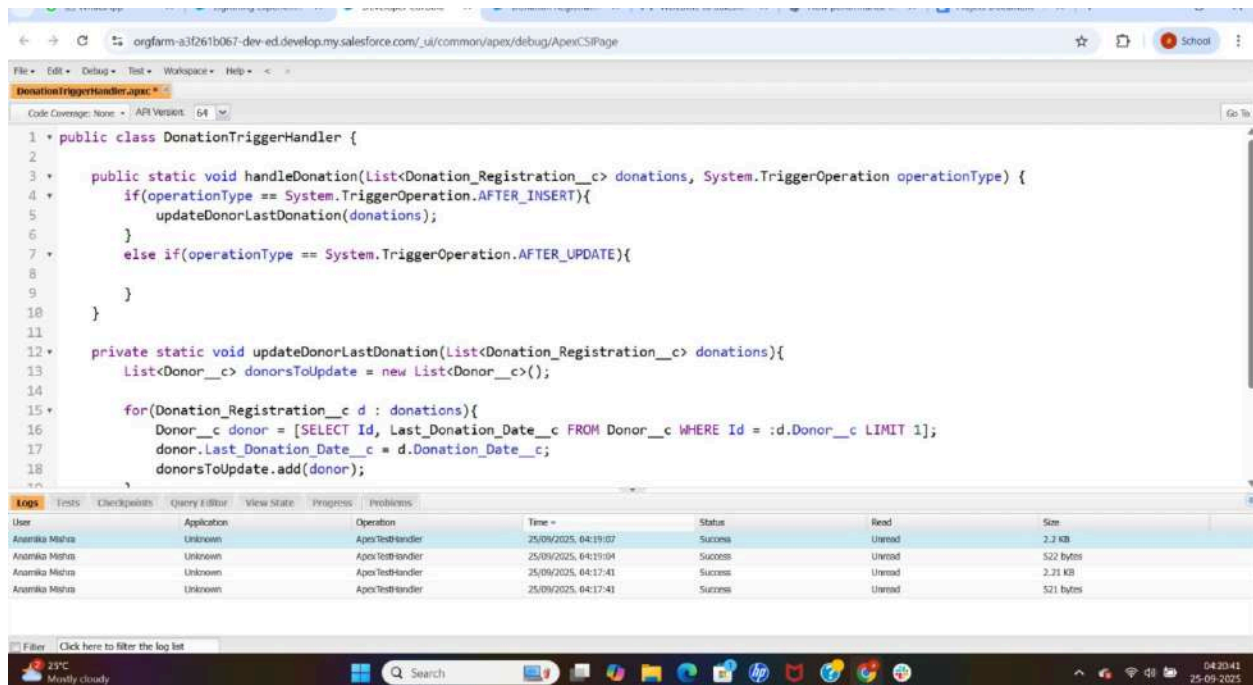
public class DonationTriggerHandler {
    public static void handleDonation(List<Donation_Registration__c> donations,
    System.TriggerOperation operationType) {
        if(operationType == System.TriggerOperation.AFTER_INSERT){
            updateDonorLastDonation(donations)}
    }
}

```

```

else if(operationType == System.TriggerOperation.AFTER_UPDATE){
    } private static void
updateDonorLastDonation(List<Donation_Registration__c> donations){
    List<Donor__c> donorsToUpdate = new List<Donor__c>();
    for(Donation_Registration__c d : donations){
        Donor__c donor = [SELECT Id, Last_Donation_Date__c FROM Donor__c
WHERE Id = :d.Donor__c LIMIT 1];
        donor.Last_Donation_Date__c = d.Donation_Date__c;
        donorsToUpdate.add(donor); }
    if(!donorsToUpdate.isEmpty()){
        update donorsToUpdate;
    } }

```



## Step 4. SOQL & SOSL

SOQL: Salesforce Object Query Language → fetch records from Salesforce objects.

```

List<Donor__c> oPositiveDonors = [SELECT Name, Phone__c FROM Donor__c
WHERE Blood_Group__c = 'O+'];

```



SOSL: Salesforce Object Search Language → search text across multiple objects.

List<List<SObject>> searchResults = [FIND 'Anamika\*' IN ALL FIELDS

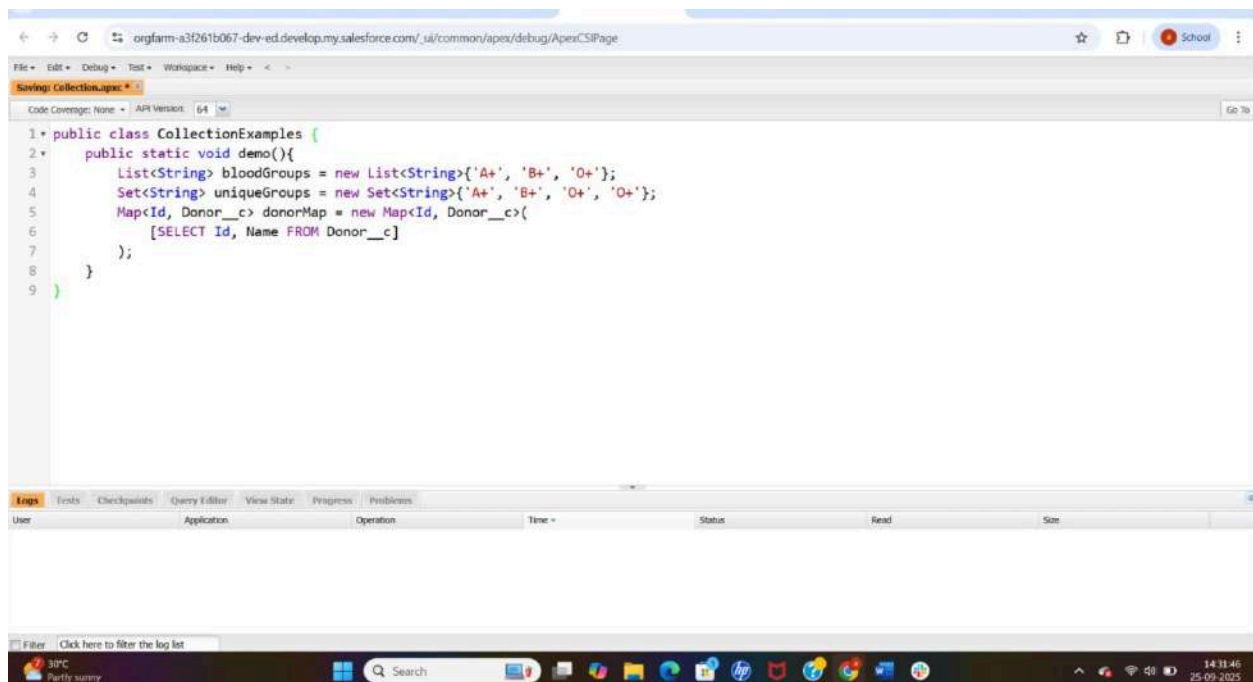
RETURNING Donor\_\_c(Name, Phone\_\_c)];

## Step 5. Collections: List, Set, Map

List: Ordered collection.

Set: Unique elements.

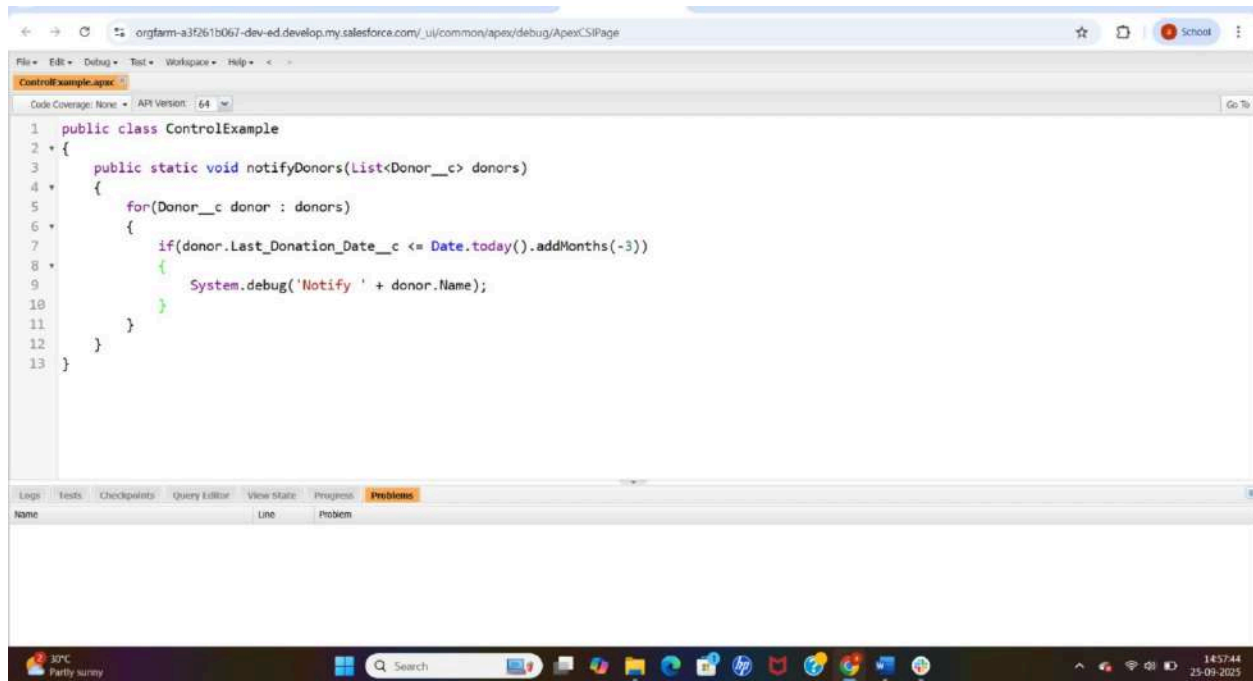
Map: Key-value pairs.



```
public class CollectionExamples {  
    public static void demo(){  
        List<String> bloodGroups = new List<String>{'A+', 'B+', 'O+'};  
        Set<String> uniqueGroups = new Set<String>{'A+', 'B+', 'O+', 'O+'};  
        Map<Id, Donor__c> donorMap = new Map<Id, Donor__c>(  
            [SELECT Id, Name FROM Donor__c]  
        );  
    }  
}
```

## Step 6. Control Statements

Send SMS only if donor hasn't donated in 3 months.



```
public class ControlExample {  
    public static void notifyDonors(List<Donor__c> donors){  
        for(Donor__c donor : donors){  
            if(donor.Last_Donation_Date__c <= Date.today().addMonths(-3)){  
                System.debug('Notify ' + donor.Name);  
            }  
        }  
    }  
}
```

## Step 7. Batch Apex

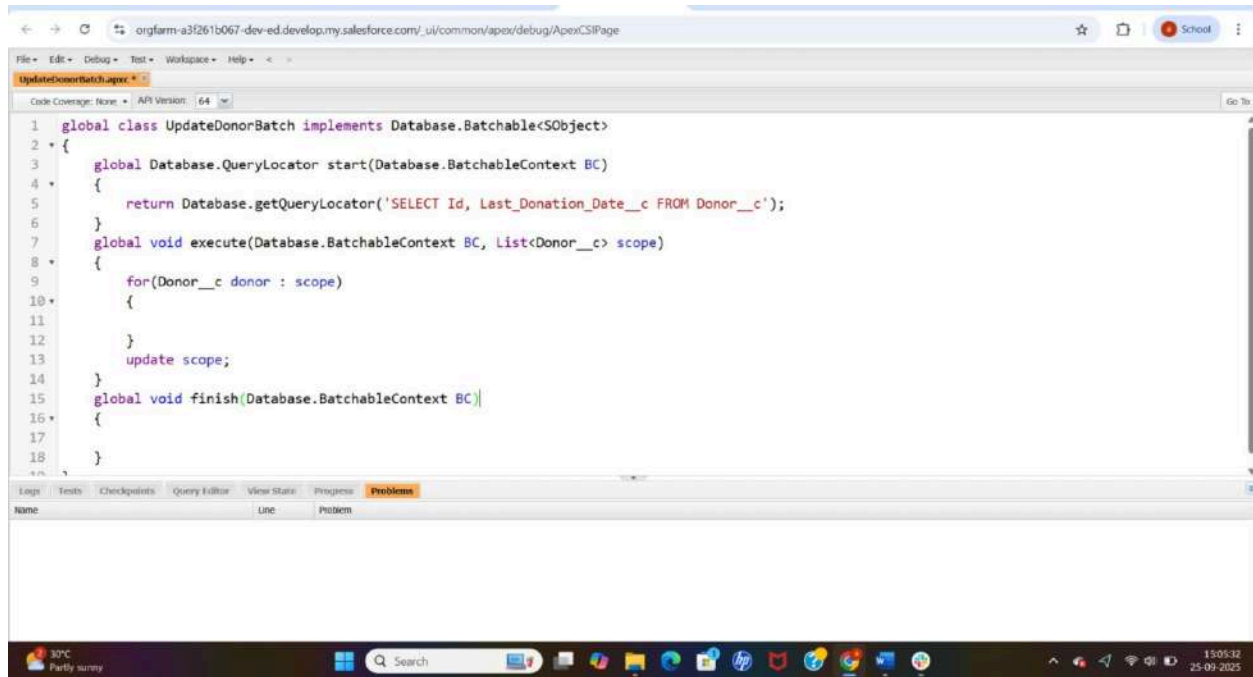
Update all donors' last donation date based on donation records

```
global class UpdateDonorBatch implements Database.Batchable<SObject>{  
    global Database.QueryLocator start(Database.BatchableContext BC){
```

```

return Database.getQueryLocator('SELECT Id, Last_Donation_Date__c FROM
Donor__c'); }
global void execute(Database.BatchableContext BC, List<Donor__c> scope){
    for(Donor__c donor : scope){ }
    update scope;}
global void finish(Database.BatchableContext BC){}}

```



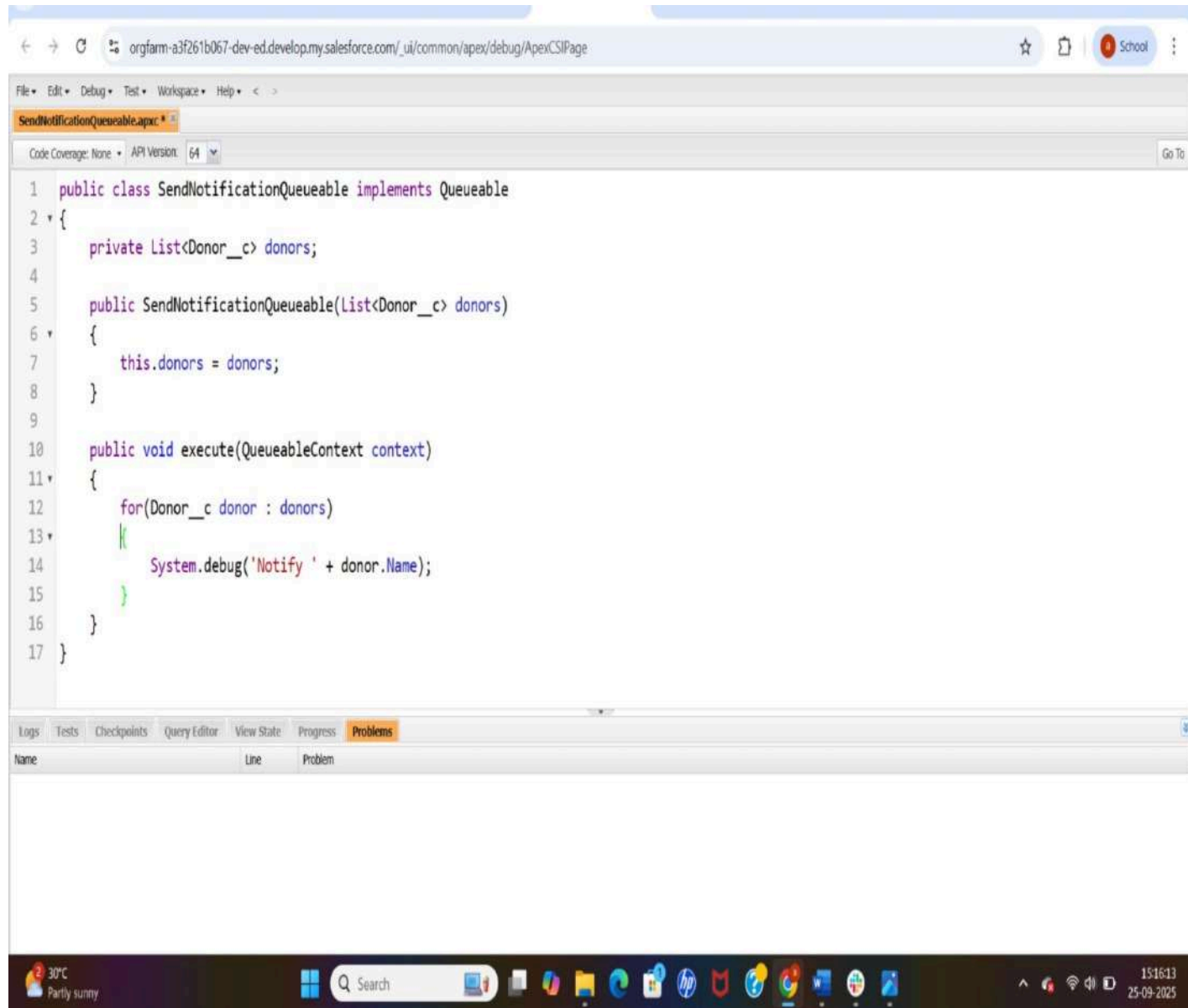
## Step 8. Queueable Apex

Send notifications to donors after donation recorded.

```

public class SendNotificationQueueable implements Queueable {
    private List<Donor__c> donors;
    public SendNotificationQueueable(List<Donor__c> donors){
        this.donors = donors; }
    public void execute(QueueableContext context){
        for(Donor__c donor : donors){
            System.debug('Notify ' + donor.Name); }}}

```



## Step 9. Scheduled Apex

Notify Donors weekly

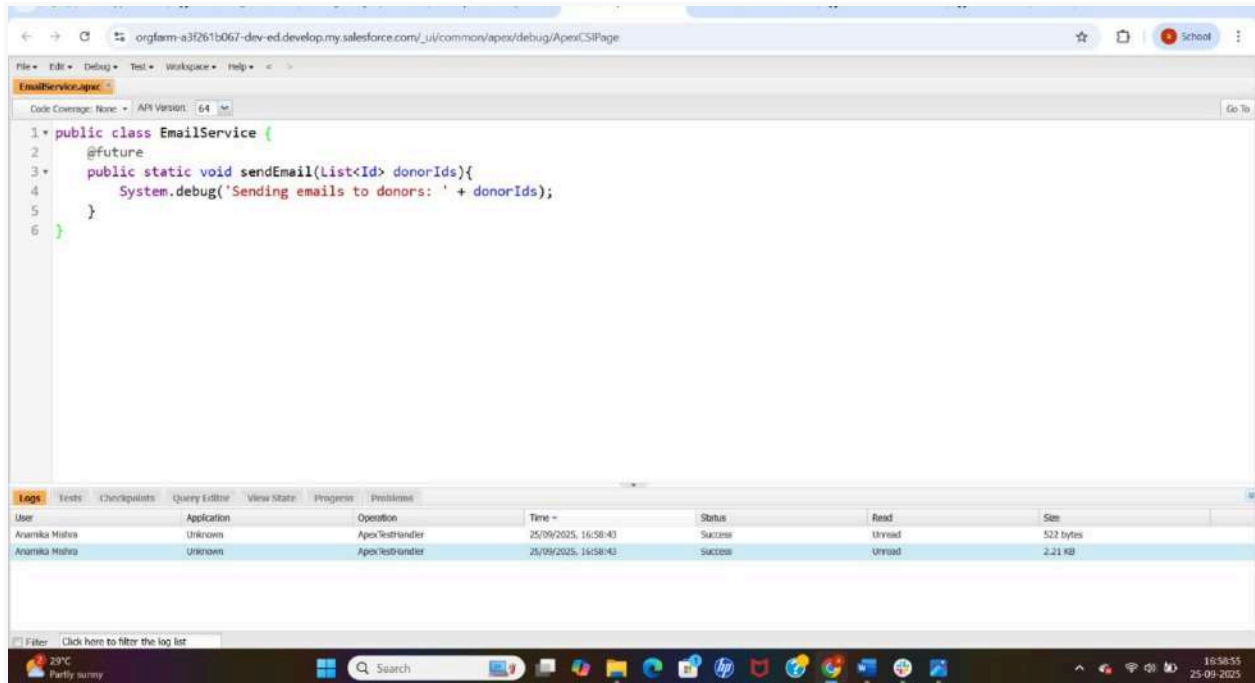
Class Name: weeklyNotification

```
global class WeeklyNotification implements Schedulable{
    global void execute(SchedulableContext sc) {
        List<Donor__c> donors = [SELECT Id, Name FROM Donor__c
```

```
WHERE Last_Donation_Date__c <= :Date.today().addMonths(-3)]; }}
```

## Step 10. Future Methods

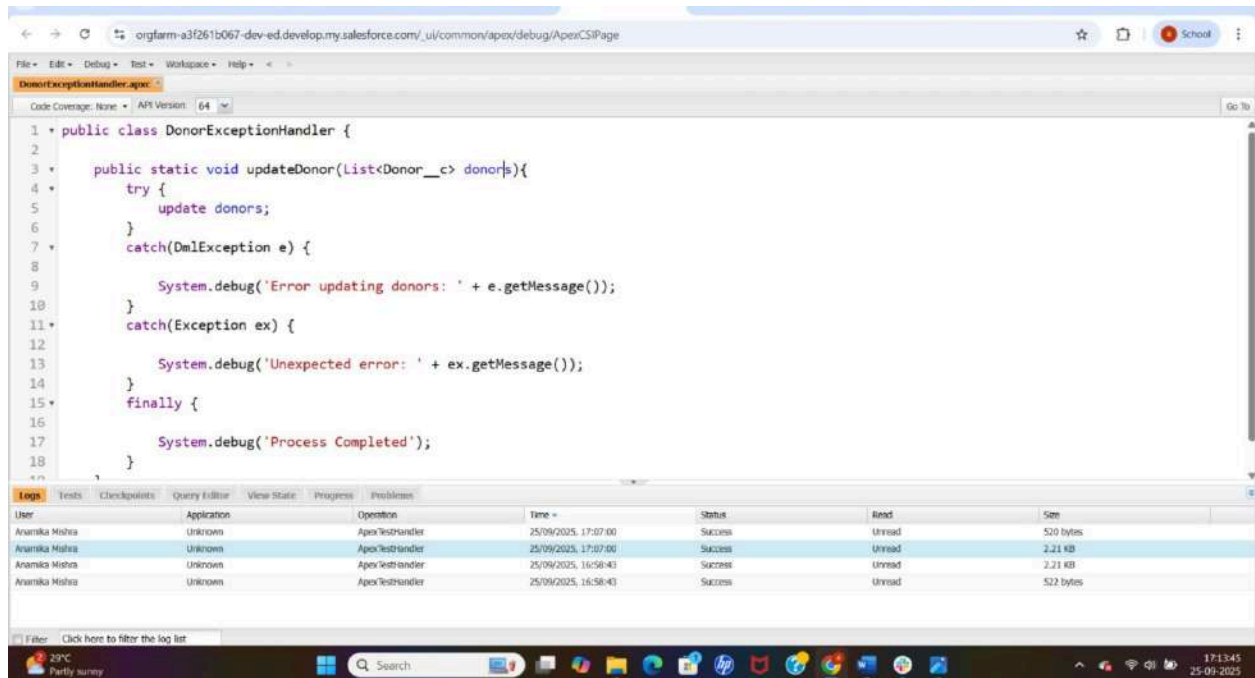
Class Name: EmailService



```
public class EmailService {  
    @future  
    public static void sendEmail(List<Id> donorIds){  
        System.debug('Sending emails to donors: ' + donorIds);  
    }  
}
```

## Step 11. Exception Handling

Class Name : DonorExceptionHandler



```

public class DonorExceptionHandler {

    public static void updateDonor(List<Donor__c> donors){
        try {
            update donors;
        }
        catch(DmlException e) {

            System.debug('Error updating donors: ' + e.getMessage());
        }
        catch(Exception ex) {

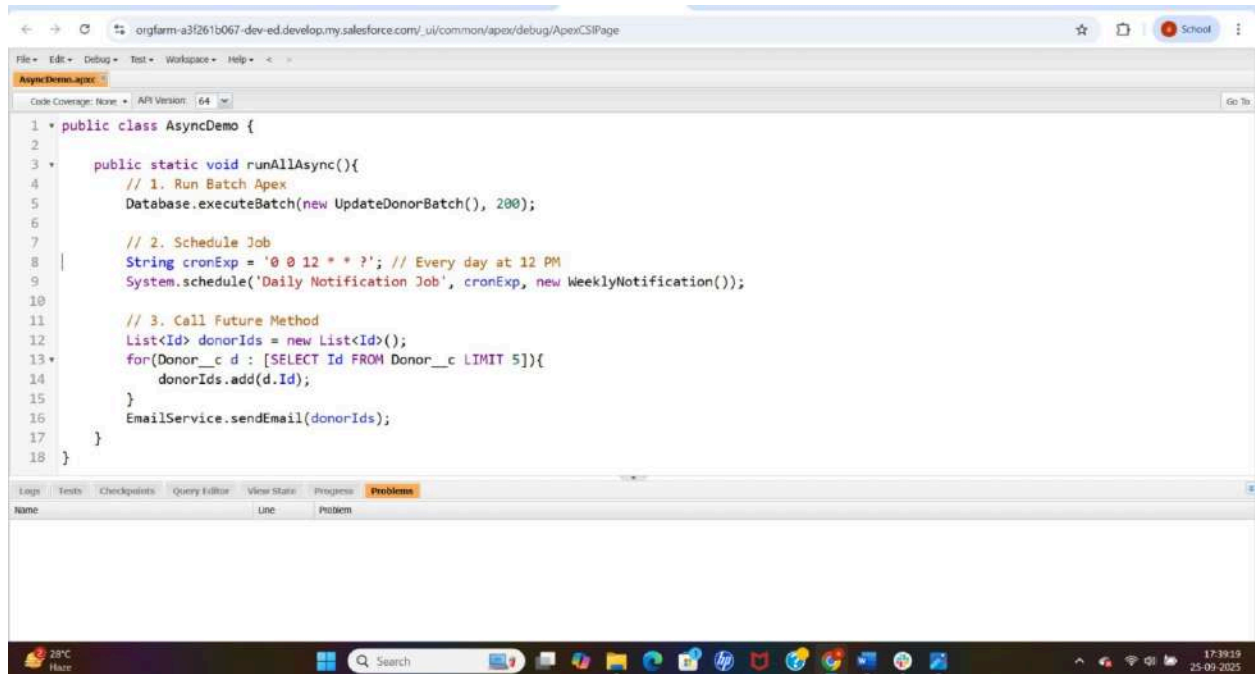
            System.debug('Unexpected error: ' + ex.getMessage());
        }
        finally {

            System.debug('Process Completed');
        }
    }
}

```

## Step 12. Asynchronous Processing

## Class Name - AsyncDemo



```
public class AsyncDemo {
```

```
    public static void runAllAsync(){
```

```
        // 1. Run Batch Apex
```

```
        Database.executeBatch(new UpdateDonorBatch(), 200);
```

```
        // 2. Schedule Job
```

```
        String cronExp = '0 0 12 * * ?'; // Every day at 12 PM
```

```
        System.schedule('Daily Notification Job', cronExp, new WeeklyNotification());
```

```
        // 3. Call Future Method
```

```
        List<Id> donorIds = new List<Id>();
```

```
        for(Donor__c d : [SELECT Id FROM Donor__c LIMIT 5]){
```

```
            donorIds.add(d.Id);
```

```
        }
```

```
        EmailService.sendEmail(donorIds);
```

```
    }
```

```
}
```

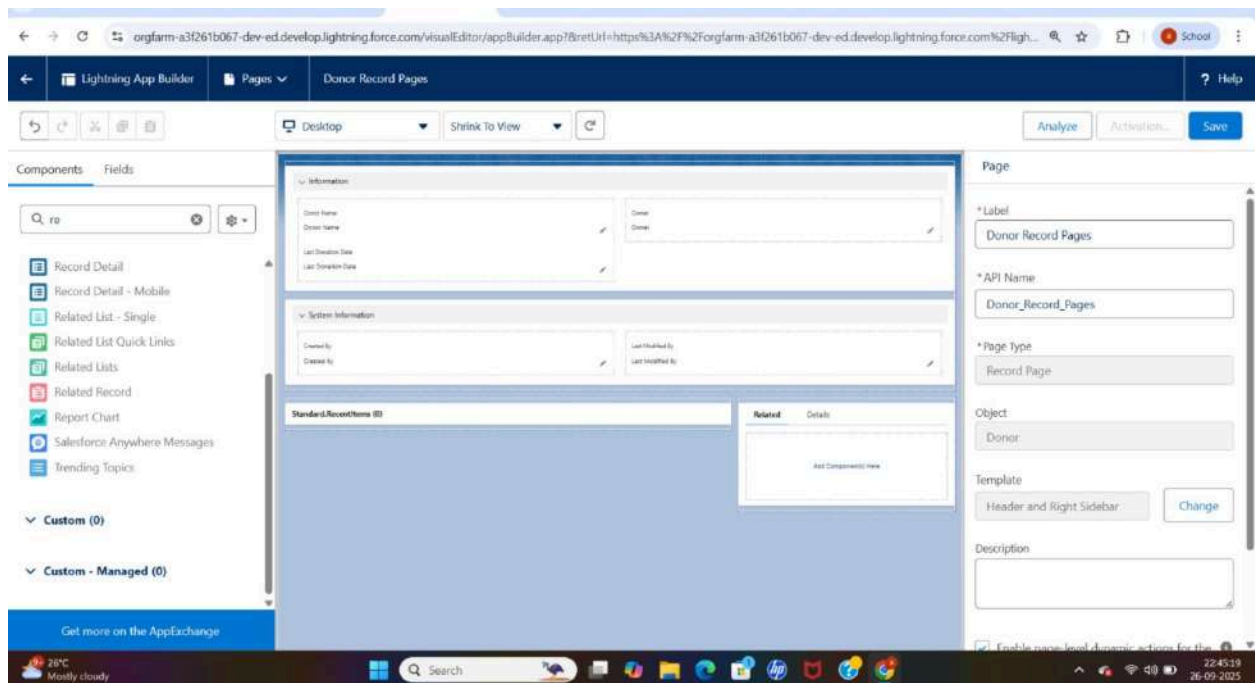
# Phase 6: User Interface Development

## Step 1. Lightning App Builder

Lightning App Builder is a point-and-click tool in Salesforce that allows administrators to design and customize user interfaces without code. It provides drag-and-drop components to create pages for apps, records, and home screens.

For the Blood Donor Management project, it helps to design screens for \*\*Donor, Donation, and Blood Camp

Purpose - The purpose of Lightning App Builder in this project is to design donor and donation pages, dashboards, and workflows that make blood donor management faster, simpler, and more effective for all users.



## Step 2. Record Pages

A Record Page in Salesforce Lightning App Builder is a customized layout that controls how a single record (like a Donor or a Donation) is displayed.



It allows administrators to add components such as record details, related lists, charts, and flows on one page.

#### Purpose of Record Pages

To give users a consolidated view of all important information related to a record.

To make data entry and updates faster and easier.

To integrate flows, charts, and quick actions directly into the record screen.

### Step 3. Tabs

Purpose: Organize multiple sections on a page into separate clickable tabs.

Use Case (Blood Donor App):

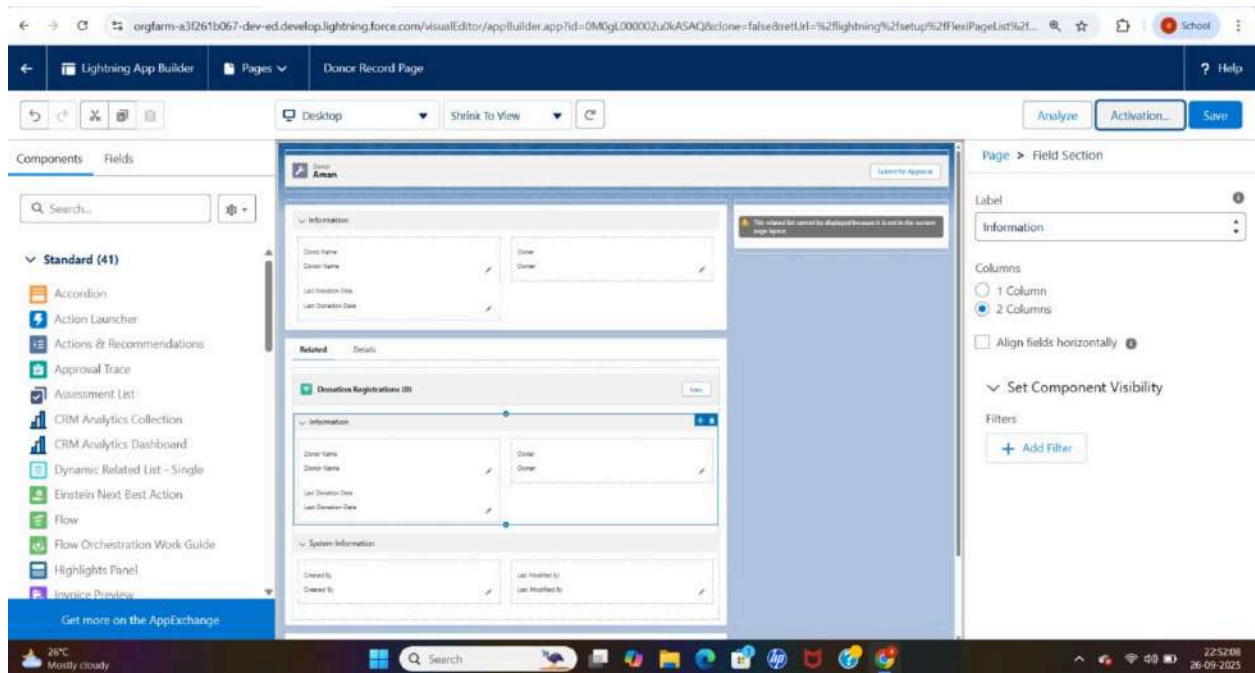
Tab 1: Donor Details

Tab 2: Donation History

Tab 3: Health Records / Notes

Steps to Add:

1. Open Lightning App Builder → Edit donor record page.
2. Drag Tabs component to the page.
3. Add Tab Labels and drag components inside each tab.



## Step 4. Apex With LWC

Apex is the server-side language of Salesforce, used for database operations and business logic. Lightning Web Components (LWC) are used for building modern, fast, and reusable user interfaces. By combining both, LWC handles the user interface while Apex fetches or updates data from Salesforce objects, enabling dynamic and interactive applications (e.g., searching donors by blood group in a Blood Donor Management project).

```
public with sharing class DonorController {
    @AuraEnabled(cacheable=true)
    public static List<Donor__c> getDonorsByBloodGroup(String bloodGroup) {
        return [
            SELECT Id, Name, Blood_Group__c, Contact_Number__c
            FROM Donor__c
            WHERE Blood_Group__c = :bloodGroup
        ];
    }
}
```

## Step 5. Events in LWC

Events in Lightning Web Components (LWC) are custom or standard signals used to communicate between components or to trigger actions.

Purpose: To allow components to communicate in a structured way without tight coupling.

Types of Events in LWC

Standard DOM Events

Built-in events like click, change, input.

Example: `<button onclick={handleClick}>Click Me</button>`.

Custom Events

Used to send data between components.

Created using CustomEvent in JavaScript.

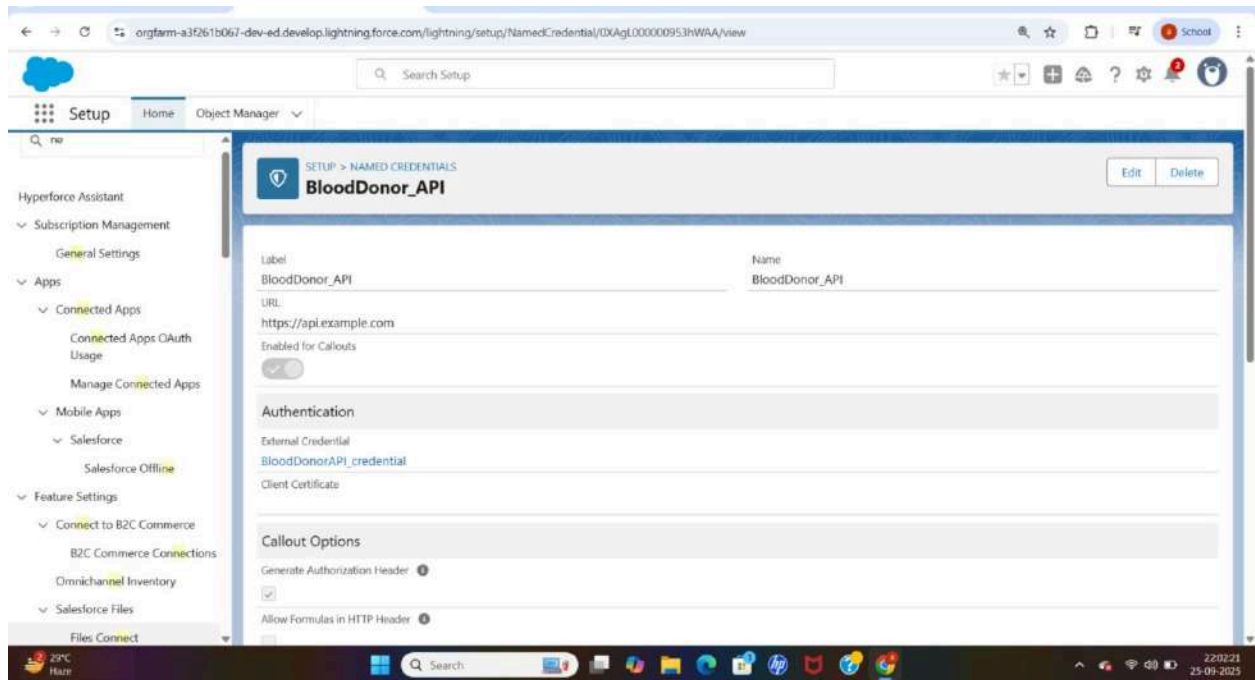
Can carry data in the detail property.

## Phase 7. Integration & External Access

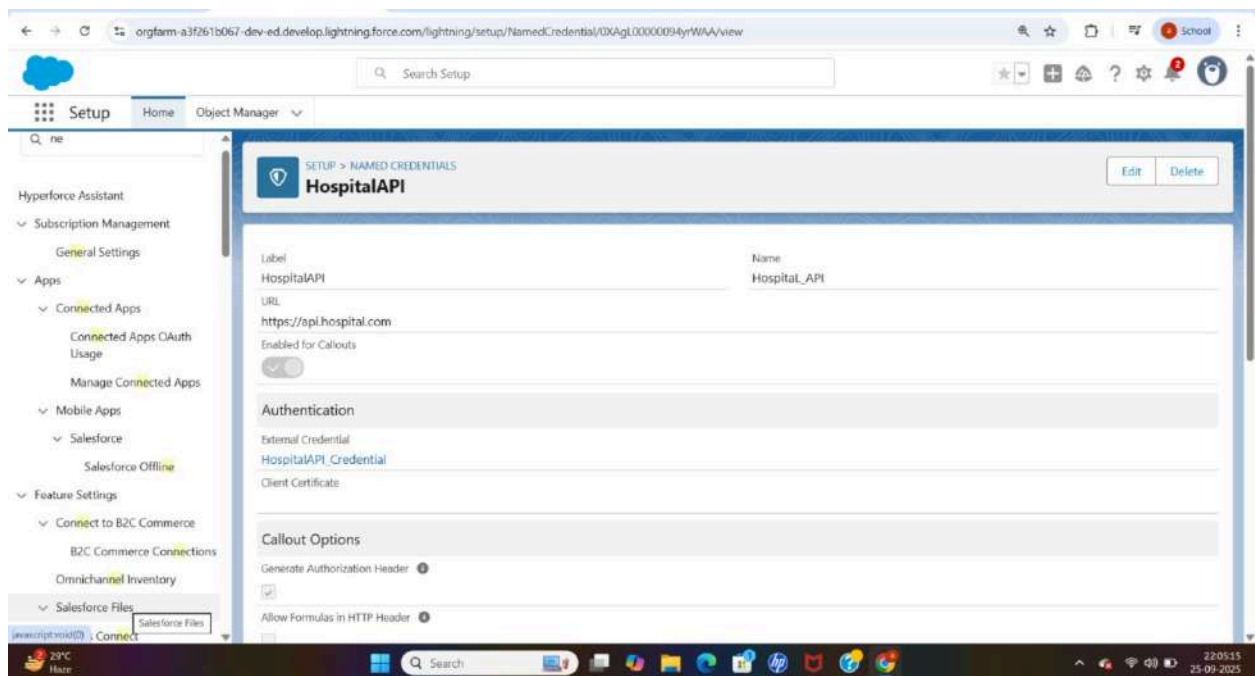
### Step 1. Named Credentials

Label - BloodDonor\_API

Name - BloodDonor\_API



Label - HospitalAPI  
Name - Hospital\_API



Step 2. External Service Name

BloodDonorService

. Description

Service to manage Blood Donor records through API integration.

. Service Schema

Uploaded JSON schema (blooddonor\_service\_schema.json) defining:

Get Donors

Add Donor

Get Donor by ID

Update Donor by ID

. Named Credential

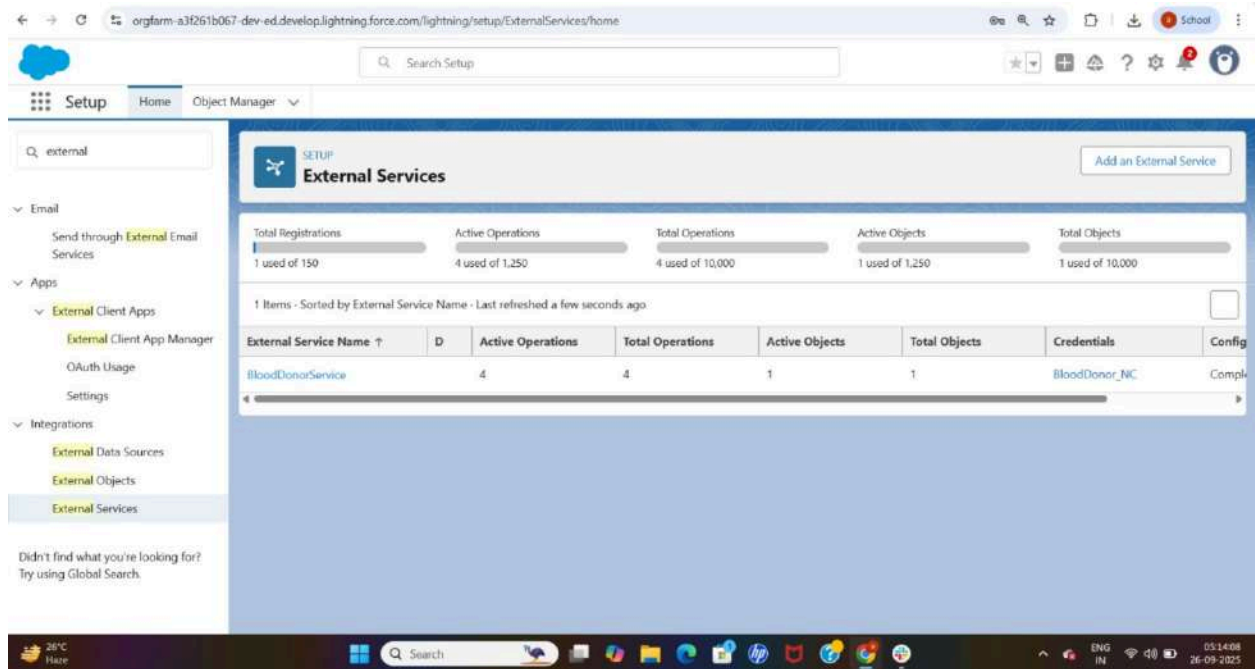
BloodDonor\_NC (used for authentication and endpoint configuration)

. Result

Registration Completed

Active Operations: 4

Total Objects Created: 1



### Step 3. Callouts

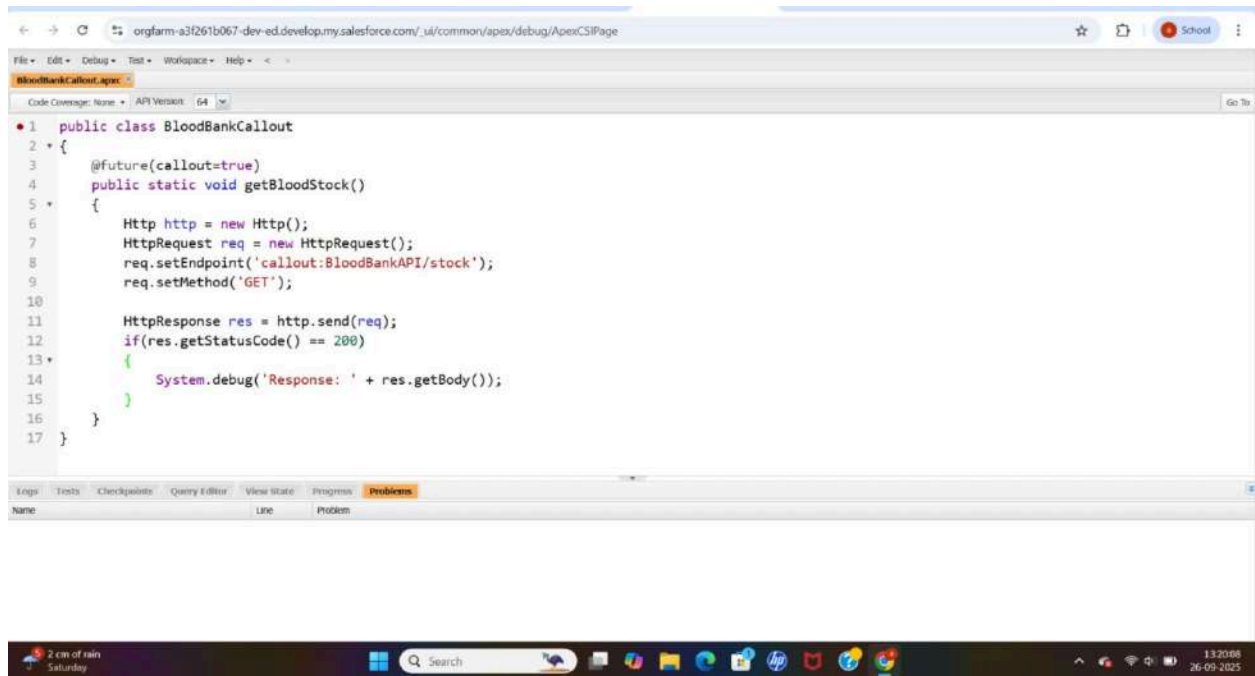
Go to Setup → Remote Site Settings → New Remote Site

Remote Site Name: Blood\_Bank\_API

Setup → Named Credentials → New

Label: BloodBankAPI

Class Name - BloodBankCallout

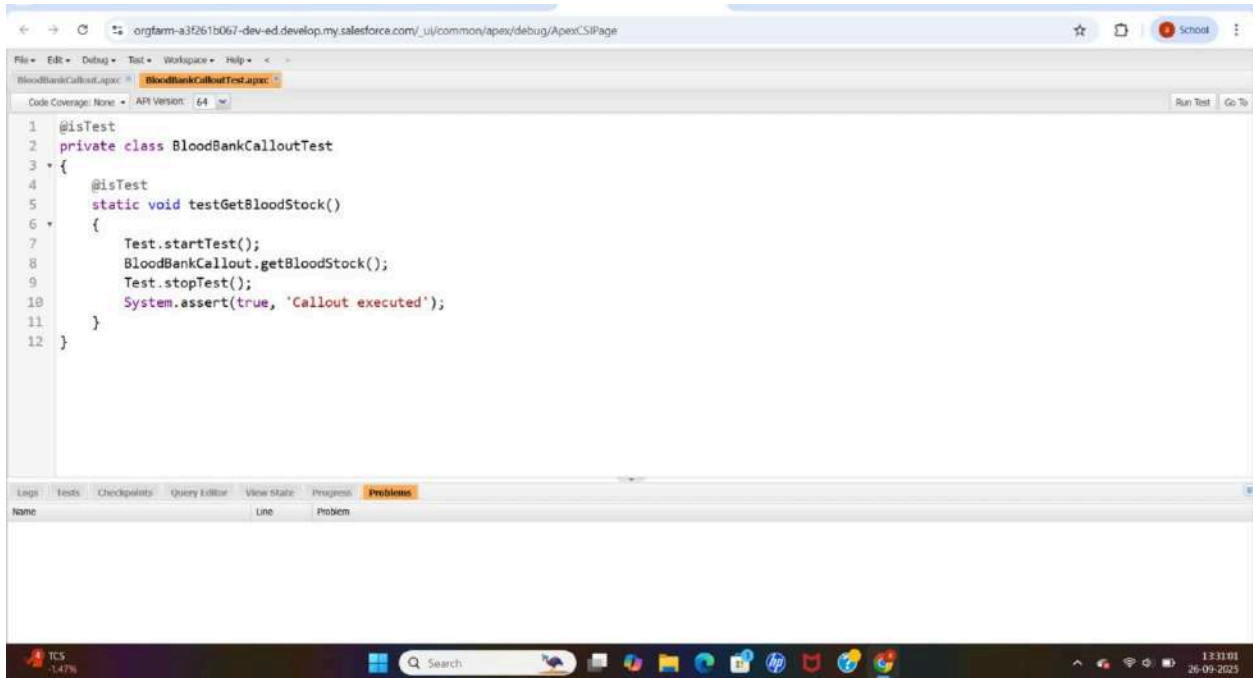


```
public class BloodBankCallout
{
    @future(callout=true)
    public static void getBloodStock()
    {
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:BloodBankAPI/stock');
        req.setMethod('GET');

        HttpResponse res = http.send(req);
        if(res.getStatusCode() == 200)
        {
            System.debug('Response: ' + res.getBody());
        }
    }
}
```

Class Name - BloodBankCalloutTest





```
@isTest
private class BloodBankCalloutTest
{
    @isTest
    static void testGetBloodStock()
    {
        Test.startTest();
        BloodBankCallout.getBloodStock();
        Test.stopTest();
        System.assert(true, 'Callout executed');
    }
}
```

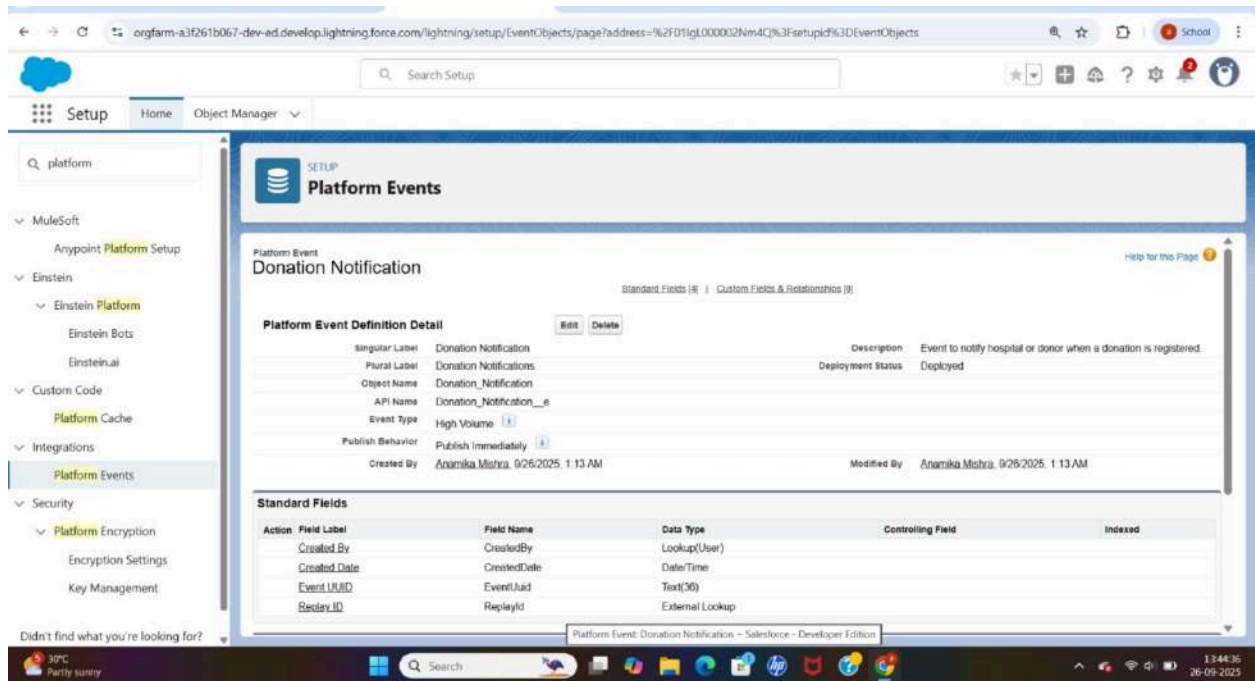
## Step 4. Platform Events

Go to Setup → Platform Events → New Platform Event.

Label: Donation Notification

Plural Label: Donation Notifications

API Name: Donation\_Notification\_\_e



@isTest

private class BloodBankCalloutTest

{

    @isTest

    static void testGetBloodStock()

    {

        Test.startTest();

        BloodBankCallout.getBloodStock();

        Test.stopTest();

        System.assert(true, 'Callout executed');

    }

}

## Step 5. Change Data Capture

Go to Setup → Quick Find → Change Data Capture.

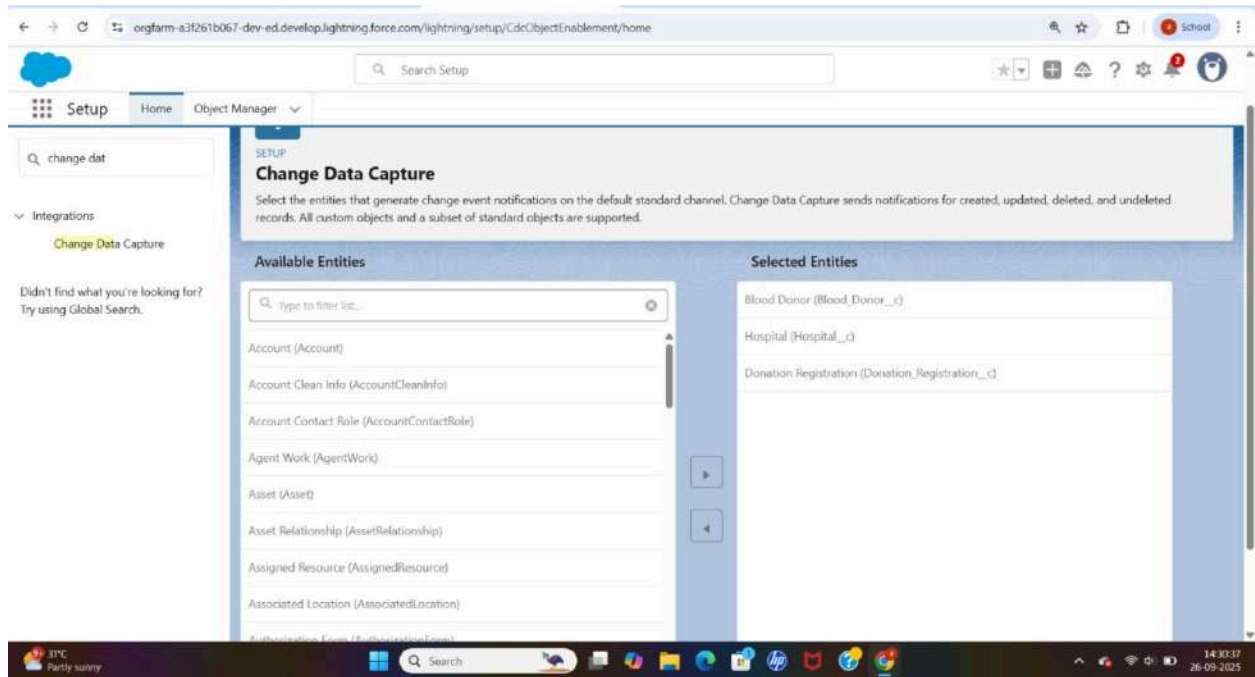
Click Enable Entities.

Select the objects you want to track (for example):

Blood\_Donor\_\_c (Donor object)

Donation\_Registration\_\_c (Donation records)

Hospital\_\_c (Hospital details)

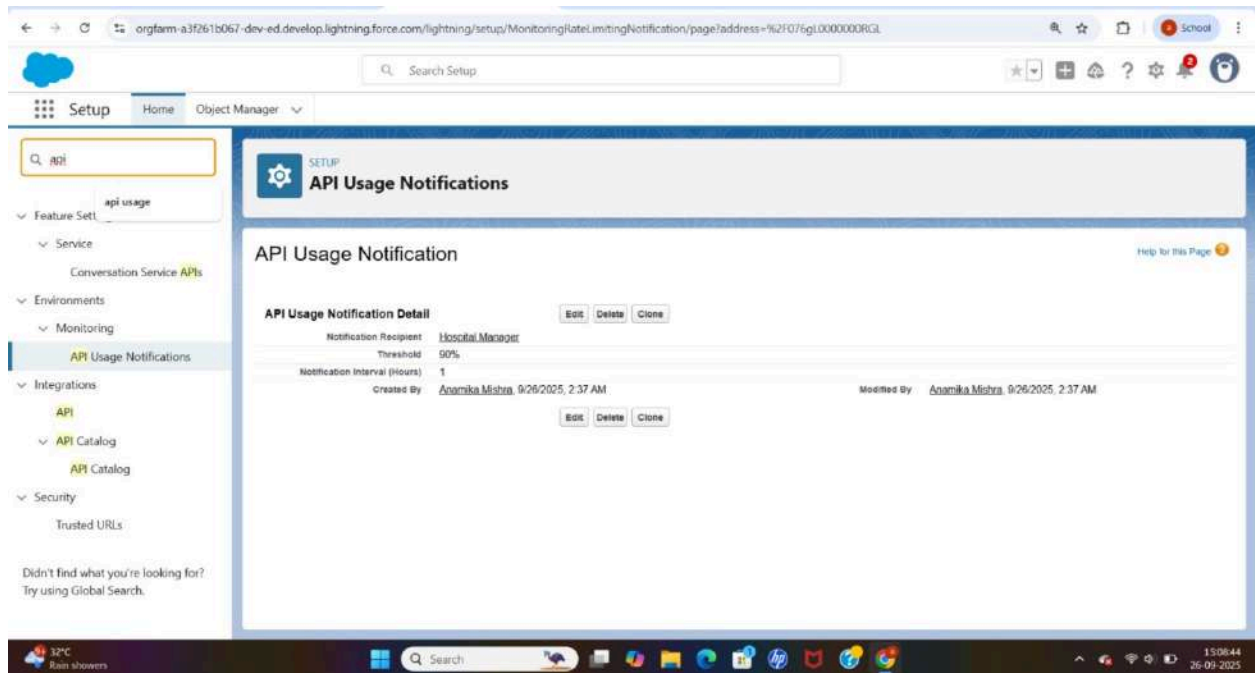


## Step 6. API Limits

Recipient: Hospital Manager

Threshold: 90%

Interval: 1 hour



## Step 7. OAuth & Authentication

Setup → App Manager → New Connected App

Fill details:

Connected App Name: BloodDonorApp

API Name: BloodDonorApp

Contact Email: aapka email

Enable OAuth Settings:

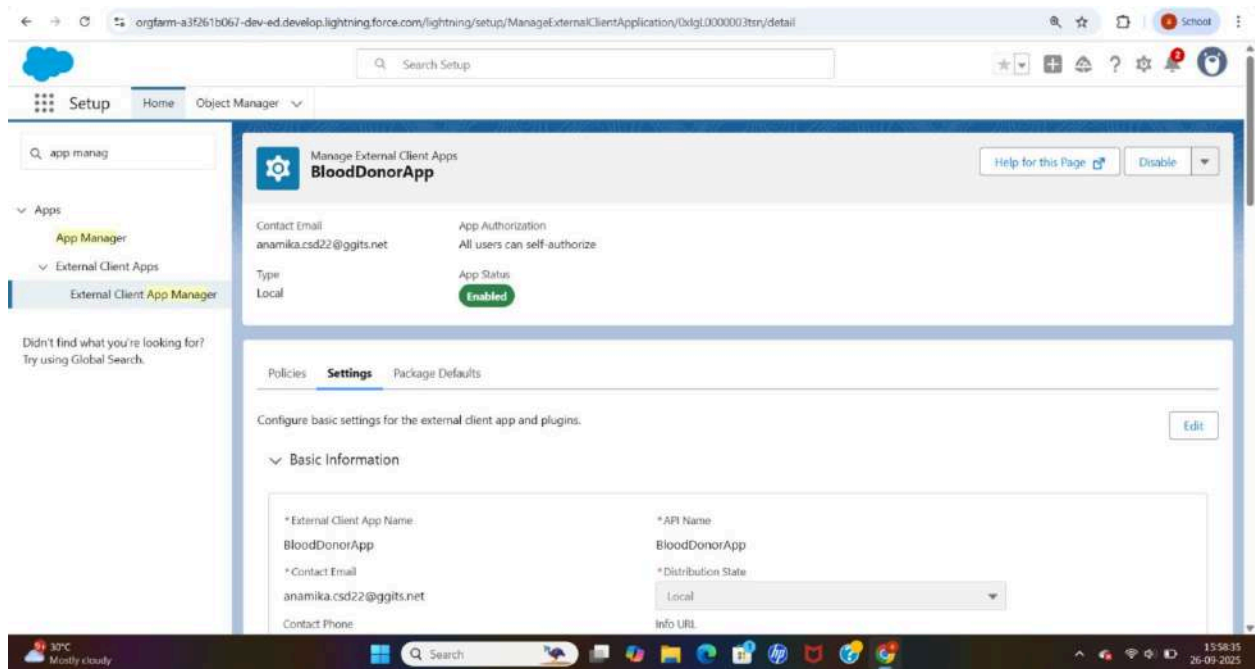
Enable OAuth Settings → check

Callback URL: <https://login.salesforce.com/services/oauth2/callback> (testing)

Selected OAuth Scopes:

Full access (full)

Perform requests on your behalf at any time (refresh\_token, offline\_access)



## Step 8. Remote Site Settings

### Purpose:

Remote Site Settings in Salesforce allow your org to make callouts to external APIs securely. For the Blood Donor Management project, this is required to access external blood donor data.

### 1. Navigate to Remote Site Settings

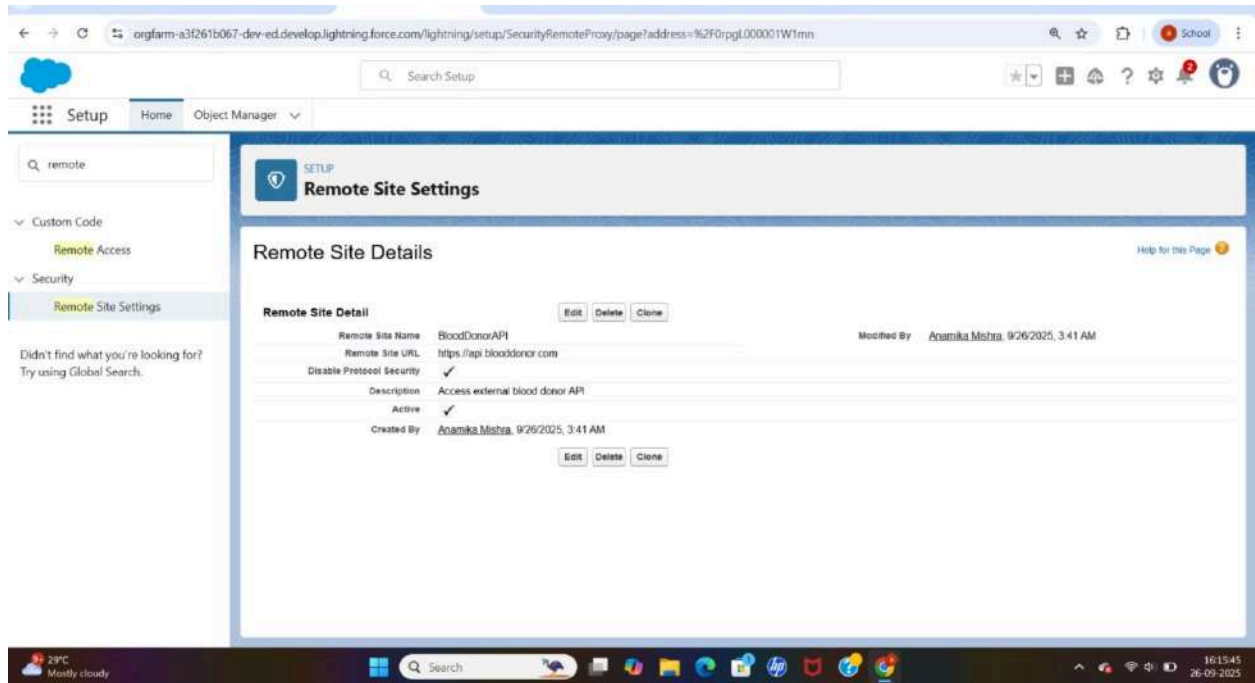
Go to Setup → Quick Find → Remote Site Settings → New Remote Site.

### 2. Enter Remote Site Details

Remote Site Name: BloodDonorAPI

Remote Site URL: <https://api.blooddonor.com>

Description: Access external blood donor API



## Phase 8: Data Management & Deployment

### Step 1. Data Loader

#### Purpose

Bulk import, export, update, delete Salesforce records.

Useful for large data operations (thousands or millions of records).

Supports CSV file format and automation via command line.

#### Steps (Process)

Download & Install Data Loader → From Salesforce Setup → Data Loader.

Login → Use Salesforce credentials (or OAuth).

Select Operation → Insert, Update, Upsert, Delete, Export, Export All.

Choose Object → Select the Salesforce object to work on (e.g., Contact, Account).

Select CSV File → Map CSV columns to Salesforce fields.

Run Operation → Process the data.

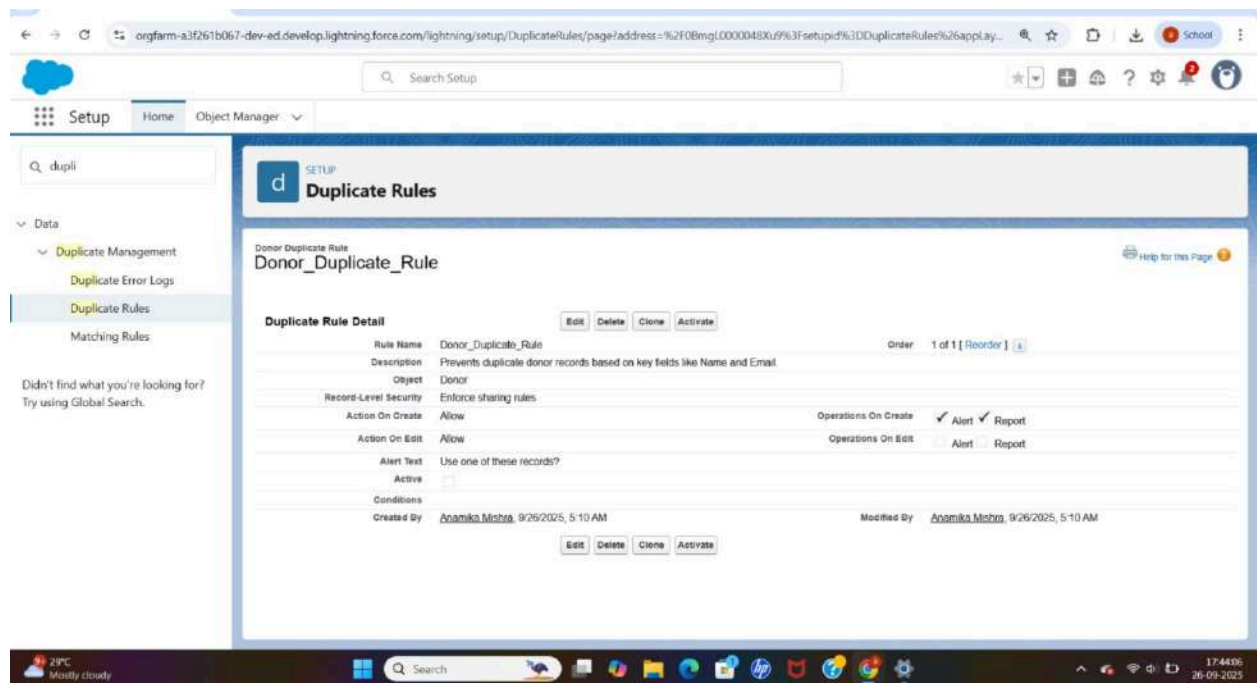
Check Results → Review success and error files generated by Data Loader.

## Step 2. Duplicate Rules

Rule Name - Donor\_Duplicate\_Rule

Description - Prevents duplicate donor records based on key fields like Name and Email.

Object - Donor



## Step 4: Data Export & Backup

Purpose: Backup donor data regularly to avoid loss.

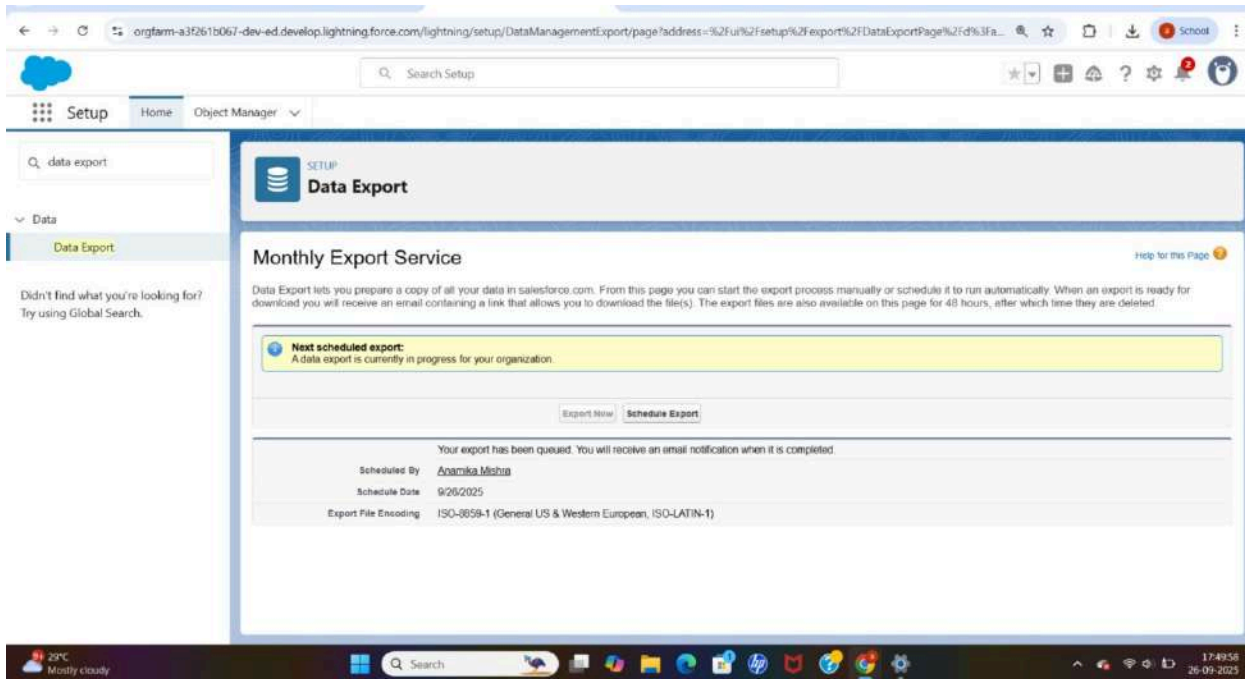
1. Select objects:

BloodDonor\_\_c

DonationRegistration\_\_c

4. Export & download ZIP file with CSVs.



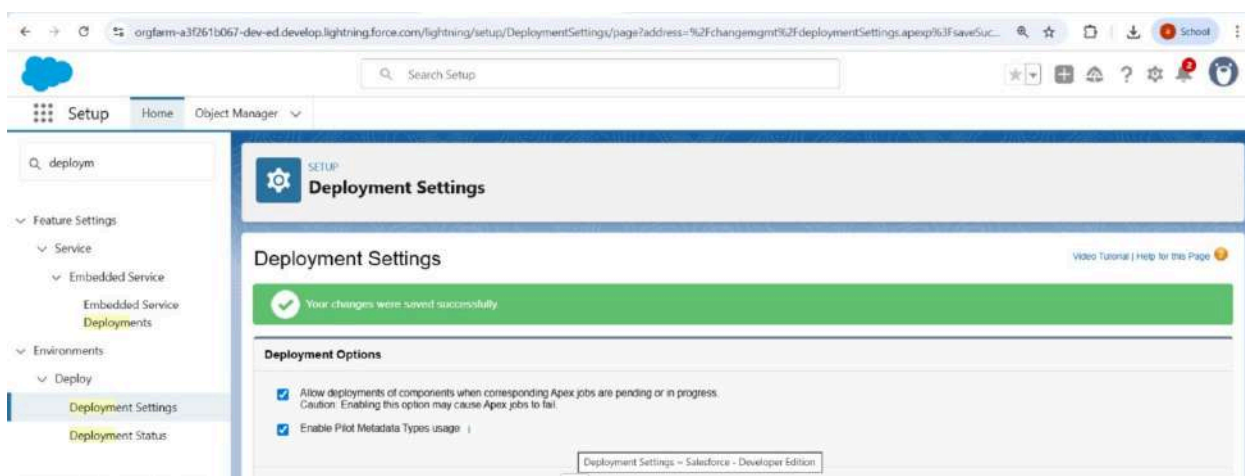


## Step 5: Change Sets

A Change Set is a Salesforce tool used to deploy customizations and configurations from one Salesforce environment (usually Sandbox) to another (usually Production) without manually recreating them. It is mainly used for moving metadata like objects, fields, flows, Apex classes, triggers, and other components.

Types:

1. Outbound Change Set – Created in the source org (Sandbox) to send components to another org.
2. Inbound Change Set – Received in the target org (Production) to deploy the components.



## Step 6. ANT Migration Tool

### Purpose

Automate deployment of metadata between Salesforce orgs.

Useful for continuous integration (CI) and version control.

Works well for large projects with many components.

### Steps (Process)

Install Java & ANT → Ensure Java and Apache ANT are installed on your system.

Download Salesforce ANT Migration Tool → From Salesforce Setup → Tools → Force.com Migration Tool.

Set Up project folder → Create a folder for project and build files.

Configure build.xml & package.xml → Define metadata to retrieve/deploy.

Retrieve Metadata → Use ANT command:

```
ant retrieve
```

Deploy Metadata → Use ANT command:

```
ant deploy
```

## Phase 9. Reporting, Dashboards & Security Review

### Step 1. Reports(Tabular, Summary, Matrix,Joined)

### Purpose

Define which objects and fields are available for reports in Salesforce.

Control how records are related in reports.

Allow creation of custom reports for specific business needs.

### Small Steps (Process)

Go to Setup → Search Report Types.

Click “New Custom Report Type”.

Select Primary Object → Choose the main object (e.g., Accounts, Orders).

Define Related Objects → Add related objects if needed.

Set Deployment Status → In Development (editable) or Deployed (available to all users).

Save → Use this report type when creating reports.

## Step 2. Report Types

### Purpose

Visually display Salesforce data in charts, graphs, tables, and gauges.

Provide quick insights and analytics for decision-making.

Summarize reports for managers and teams.

### Steps (Process)

Go to Dashboards Tab → Click New Dashboard

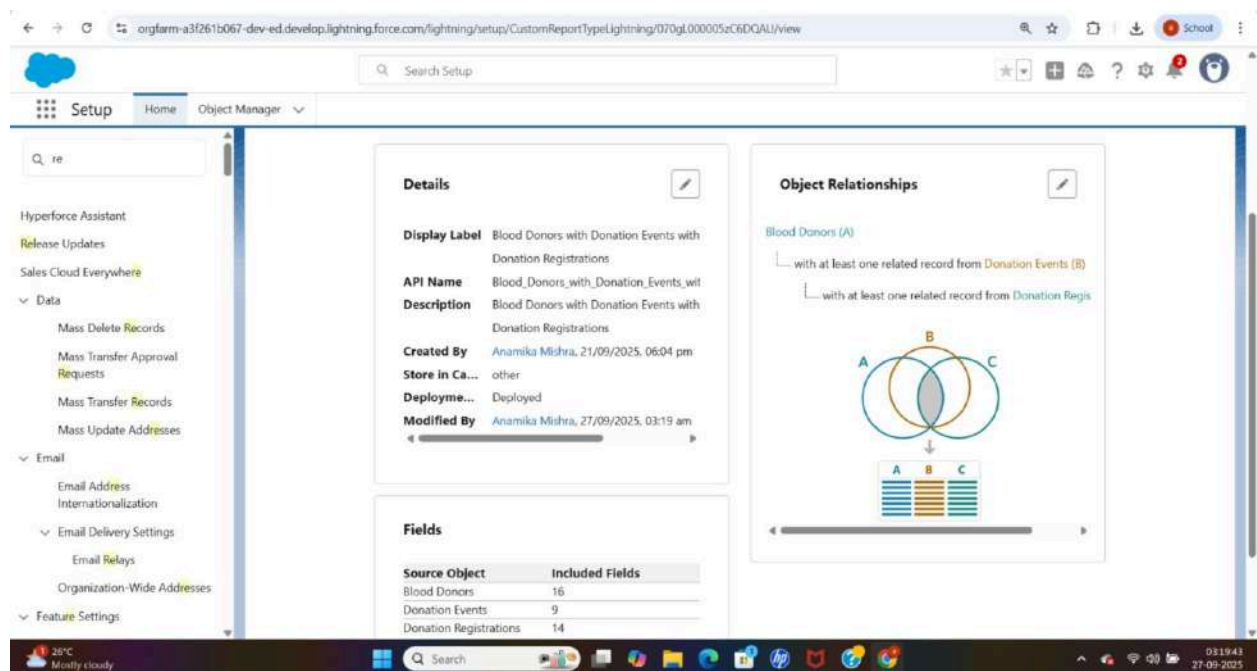
Name Dashboard → Give a meaningful name and select a folder.

Add Components → Charts, graphs, tables, gauges.

Select Source Reports → Choose reports that will feed data to dashboard components.

Customize Components → Adjust chart types, filters, labels.

Save & Refresh → Ensure data is up-to-date.



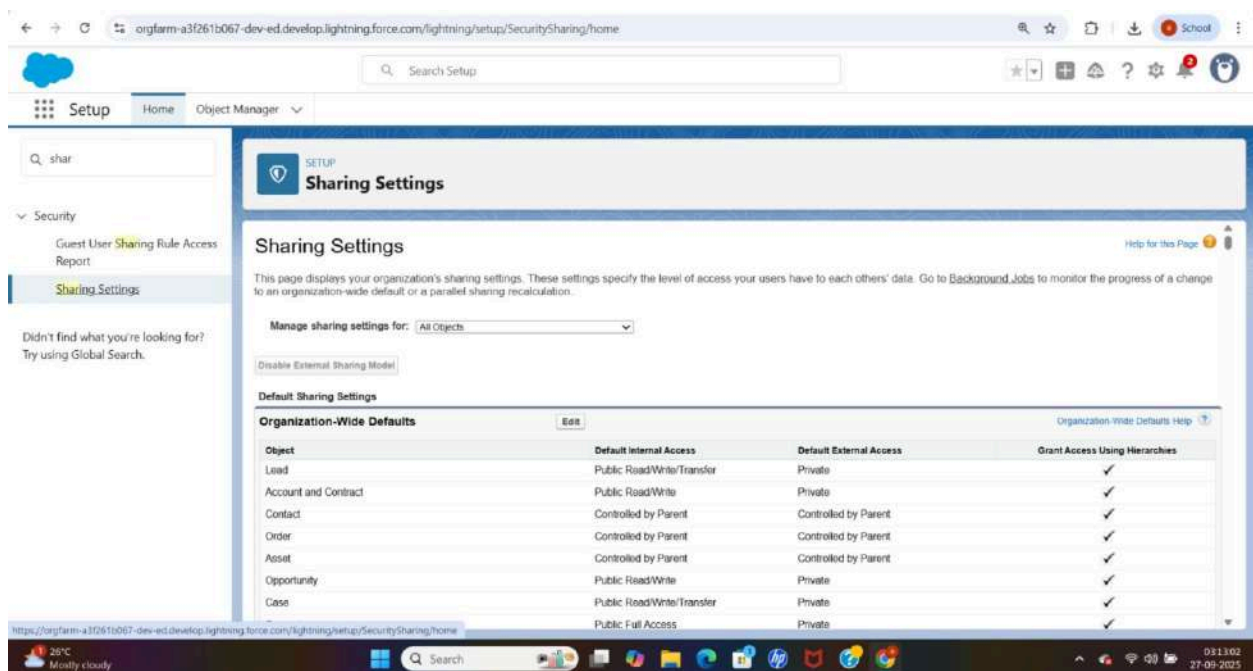
## Step 3. Sharing Settings

### Purpose

Control record-level access in Salesforce.  
Define who can view, edit, or delete records.  
Maintain data security while enabling collaboration.

### Steps (Process)

Go to Setup → Search Sharing Settings.  
Set Organization-Wide Defaults (OWD) → Define baseline access for each object (Public, Private, Controlled).  
Create Sharing Rules → Grant access to specific groups/roles.  
Use Role Hierarchies → Inherit access based on role levels.  
Set Manual Sharing → Allow record owners to share records individually.  
Test Access → Log in as a user to verify permissions.



## Step 4. Field Level Security

### Purpose

Control visibility and editability of individual fields in Salesforce objects.  
Protect sensitive data while allowing access to necessary information.  
Apply security at the profile or permission set level.

## Steps (Process)

Go to Setup → Search Field Accessibility or Profiles.

Select Object → Choose the object containing the field.

Select Field → Click on the field to edit security.

Set Access Levels →

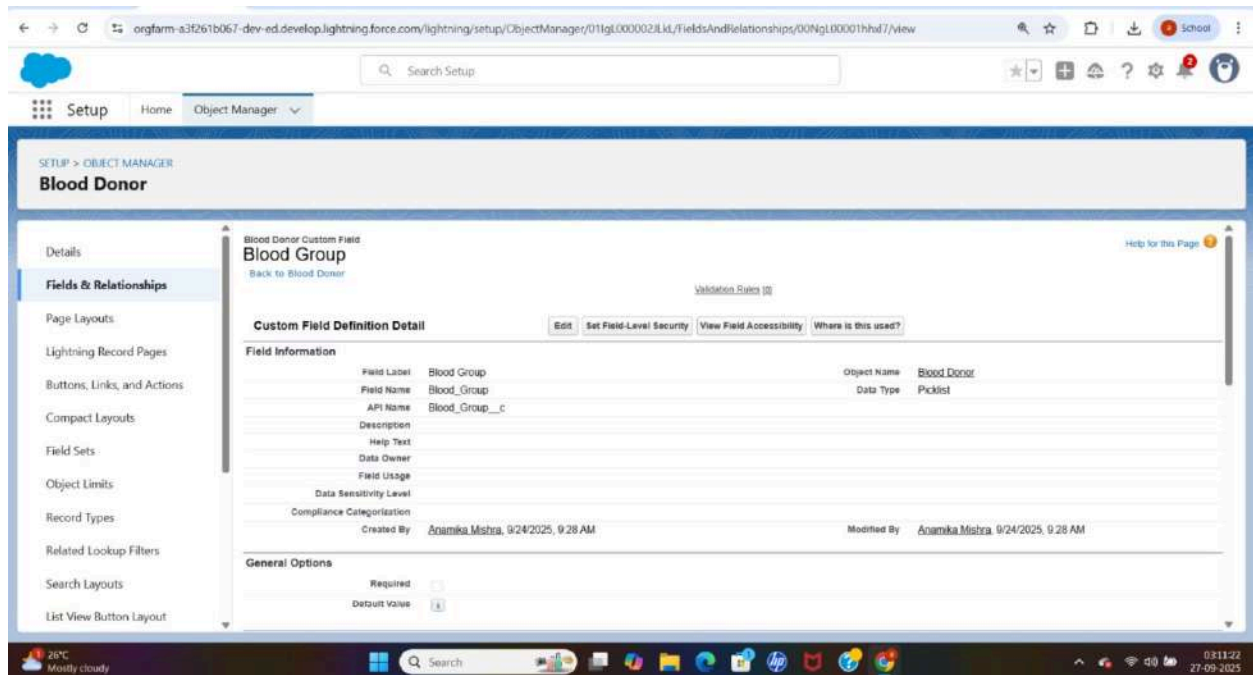
Visible → User can see the field.

Read-Only → User can view but not edit.

Hidden → Field is not visible.

Save Changes → Apply settings for profiles or permission sets.

Test Access → Log in as user to confirm field visibility.



## Step 5. Session Settings

### Purpose

Control user session behavior in Salesforce for security and performance.

Define session timeout, security policies, and login access.

Protect org data by managing how and when sessions expire.

### Steps (Process)

Go to Setup → Search Session Settings.

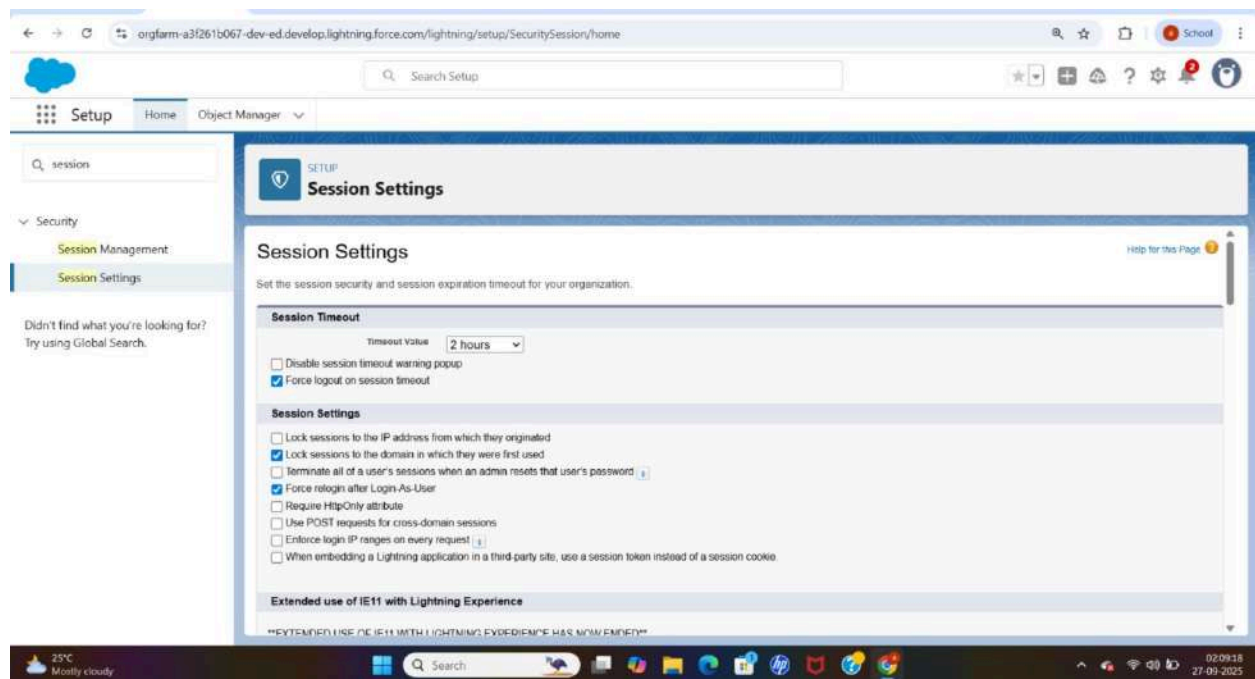
Set Session Timeout → Choose duration (e.g., 30 min, 2 hours).

Configure Security Policies → Enable/disable secure session settings (e.g., IP ranges, login hours).

Enable/Disable Persistent Sessions → Decide whether users stay logged in across browser restarts.

Save Changes → Apply settings.

Test → Log in as a user to verify session behavior.



## Step 6 . Login IP Ranges

### Purpose

Restrict Salesforce login access to specific IP address ranges.

Enhance org security by controlling where users can log in from.

Useful for enforcing network-based access control.

### Steps (Process)

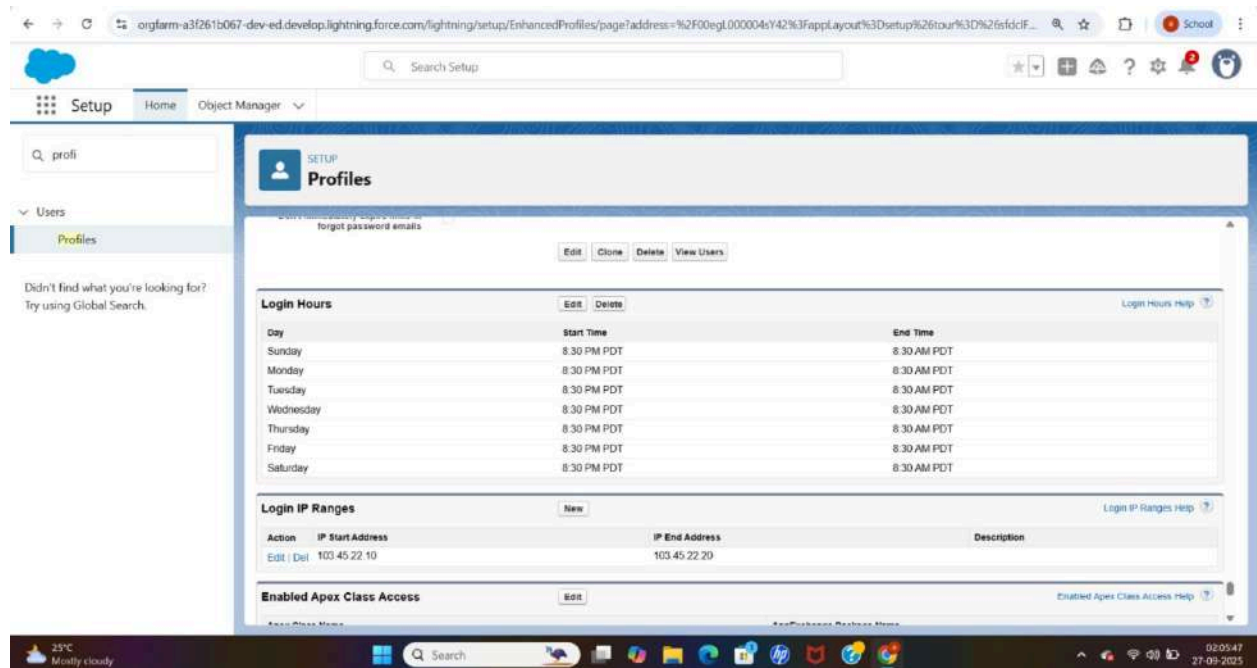
Go to Setup → Search Profiles (IP ranges are set per profile).

Select Profile → Click the profile you want to set IP ranges for.

Find Login IP Ranges Section → Click New.

Enter IP Range → Start IP and End IP addresses (e.g., 103.45.22.10 to 103.45.22.20).

Save Changes → Restriction applies immediately.



## Step 7. Audit Trail

### Purpose

Track administrative changes for accountability.

Support compliance and security audits.

Help troubleshoot unexpected behavior by reviewing configuration changes.

### Steps (Process)

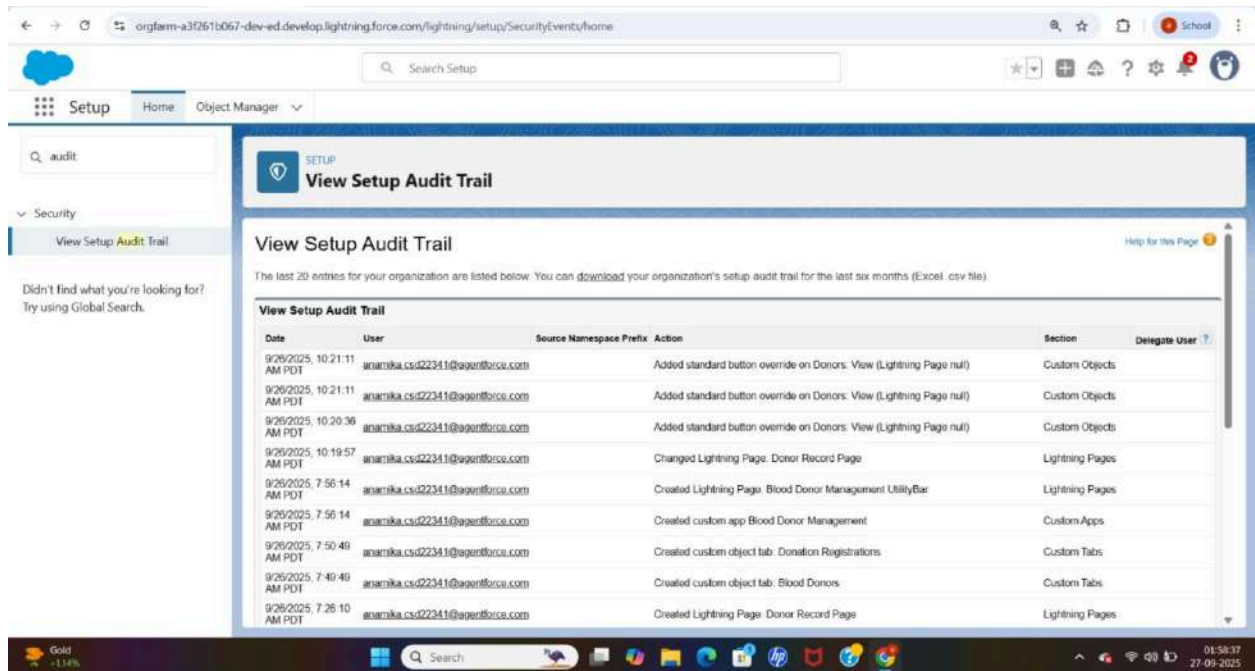
Go to Setup → Search View Setup Audit Trail.

Review Recent Changes → See changes for the last 180 days.

Download Audit Log → Export for analysis or compliance documentation.

Filter by Date/User → Focus on specific changes.





## Phase 10. Final Presentation & Demo Day

### 1. Pitch Presentation

**Purpose:** Introduce the project, its value, and impact.

**Problem Statement:** Challenges in blood donation and management.

**Solution:** How the Blood Donor Management System addresses these challenges.

**Impact:** Efficiency, timely blood availability, better donor engagement.

### 2. Demo Walkthrough

**Purpose:** Showcase the project in action.

**Steps:**

1. Login & Home Page overview.
2. Donor registration process.
3. Blood inventory & donation tracking.
4. Reports & dashboards highlights.
5. Any automation (notifications, alerts).

### 3. Feedback Collection

**Purpose:** Gather suggestions for improvement.

**Methods:**

Online forms (Google Forms, SurveyMonkey).

Verbal feedback during demo.

**Focus Areas:** Usability, completeness, design, functionality, real-world applicability.

### 4. Handoff Documentation

**Purpose:** Enable smooth continuation or deployment by other users or developers.

Content:

System overview and architecture.

Salesforce objects, fields, and relationships.

Reports, dashboards, and automation details.

User manual for donor registration, reporting, and notifications.

Deployment or setup instructions.

## Conclusion

The Blood Donor Management System built on Salesforce provides a comprehensive solution to streamline blood donation processes. By centralizing donor information, tracking donations, managing blood inventory, and enabling automated notifications, the system improves efficiency, reduces errors, and ensures timely availability of blood for patients in need.

The project demonstrates how Salesforce's powerful features—like Lightning Web Components, Apex, Reports, and Dashboards—can be leveraged to create a user-friendly, scalable, and impactful solution. Overall, this system not only enhances hospital operations but also fosters better engagement with donors, ultimately contributing to saving lives.







