Phase 1 Problem Understanding & Industry Analysis

Goal of Phase 1

To understand the pain points in the healthcare donation process, analyze stakeholders, and design an improved workflow using Salesforce CRM. This phase focuses on identifying requirements, mapping current vs. proposed processes, and researching industry best practices.

---

1 Requirement Gathering

What was done:

- Studied pain points of **blood banks, hospitals, NGOs, patients, and donors**.
- Identified key challenges in the current system.

Key Requirements Identified:

- Difficulty in quickly finding a suitable donor during emergencies.
- No centralized system to track donor availability (blood group, organ compatibility, location).
- Manual communication between hospitals, patients, and donors causes delays.
- Lack of reminders for repeat donations.
- No analytics for monitoring donor trends and shortages.

□ **Output:** Requirements document listing problems to solve.

---

2 Stakeholder Analysis

Stakeholders Identified:

- **Donors** → Individuals willing to donate blood/organs.
- **Patients & Families** → Need urgent access to donations.
- **Hospitals/Clinics** → Require quick donor matching and patient history tracking.
- **Blood Banks & NGOs** → Manage donor lists and blood stock.
- **Government Health Agencies** → Require donation data for policy and public health planning.

Goals of Stakeholders:

- Donors: Easy scheduling & reminders.
- Patients: Quick access to compatible donors.
- Hospitals: Real-time updates on donor availability.
- NGOs/Blood Banks: Centralized donor management.
- Government: Data analytics for shortages and trends.

□ **Output:** Stakeholder matrix (stakeholder + their goals).

---

3 Business Process Mapping

Current Process (Manual) – "As-Is":

1. Patient urgently needs blood.
2. Family calls multiple hospitals/blood banks.
3. If a donor is found → manual coordination begins.
4. Organ donations delayed due to poor inter-hospital communication.

Proposed Process (With Salesforce CRM) – "To-Be":

1. Patient request logged into CRM (via web/app/call center).
2. CRM searches donor database (blood group, organ compatibility, location).
3. Nearest donor notified automatically (SMS/Email/App).
4. Hospital updates donation status in real-time.
5. Dashboards show donations, stock, and shortages for administrators.

□ **Output:** "As-Is" and "To-Be" process flow diagrams.

---

4 Industry-Specific Use Case Analysis

Healthcare Industry Pain Points:

- Time-sensitive emergency donations.
- Lack of real-time donor-patient matching.
- No follow-up system for repeat donors.

Salesforce CRM Use Cases:

- **Lead Management:** Donor = Lead, Patient Request = Case.
- **Service Console:** Hospital staff manage requests on a single screen.

- **Automation:** Automated reminders for donor's next donation date.
- **Dashboards:** Admins monitor donation trends, stock, and shortages.

 Output: Documented healthcare-specific Salesforce use cases.

---

5 AppExchange Exploration

Apps/Tools Explored:

- **Salesforce Health Cloud** – Comprehensive healthcare solution.
- **Donor Management Apps** – Used by nonprofits for donor records.
- **Appointment Management Add-ons** – Enhance scheduling and follow-ups.

Goal: Identify what's already available and plan unique features like **smart donor matching** and **AI-powered reminders**.

 Output: AppExchange research notes.

End of Phase 1 Deliverables

- Problem statement & requirements document.
- Stakeholder analysis chart.
- Business process flow ("As-Is" vs "To-Be").
- Healthcare-specific Salesforce use cases.
- AppExchange research notes.

Phase 2 – Org Setup & Configuration

Goal of Phase 2

To configure the Salesforce Org with **company details, business hours, fiscal year, users, profiles, roles, OWD, and sharing rules** so that the system is secure and ready for custom object creation.

Steps Followed

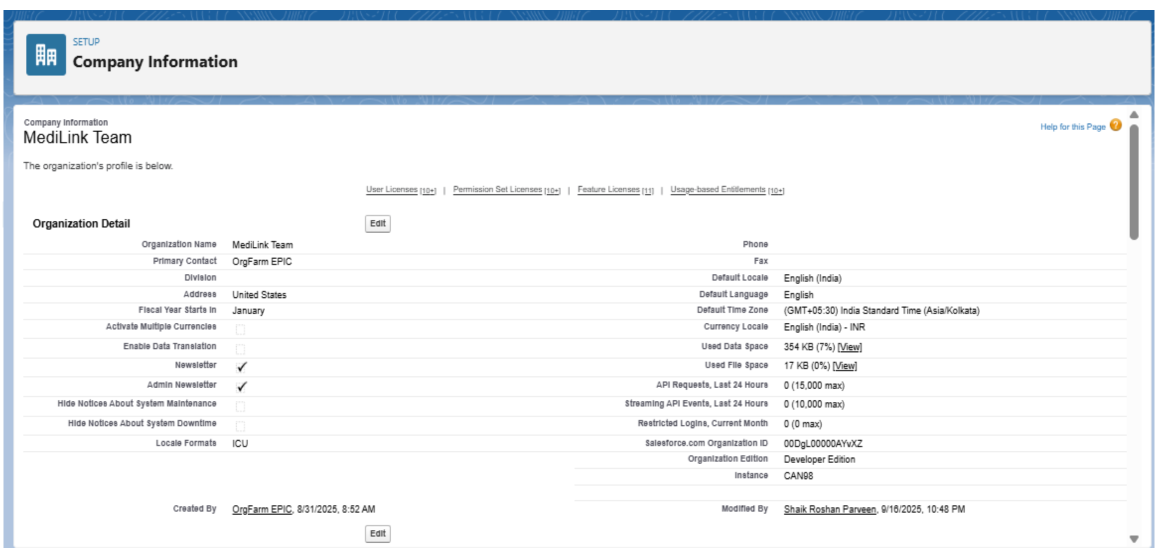
1. Salesforce Edition & Org Setup

- Logged into **Salesforce Developer Edition**.
- Went to **Setup → Quick Find → Company Information**.
- Verified edition = Developer Edition.

---

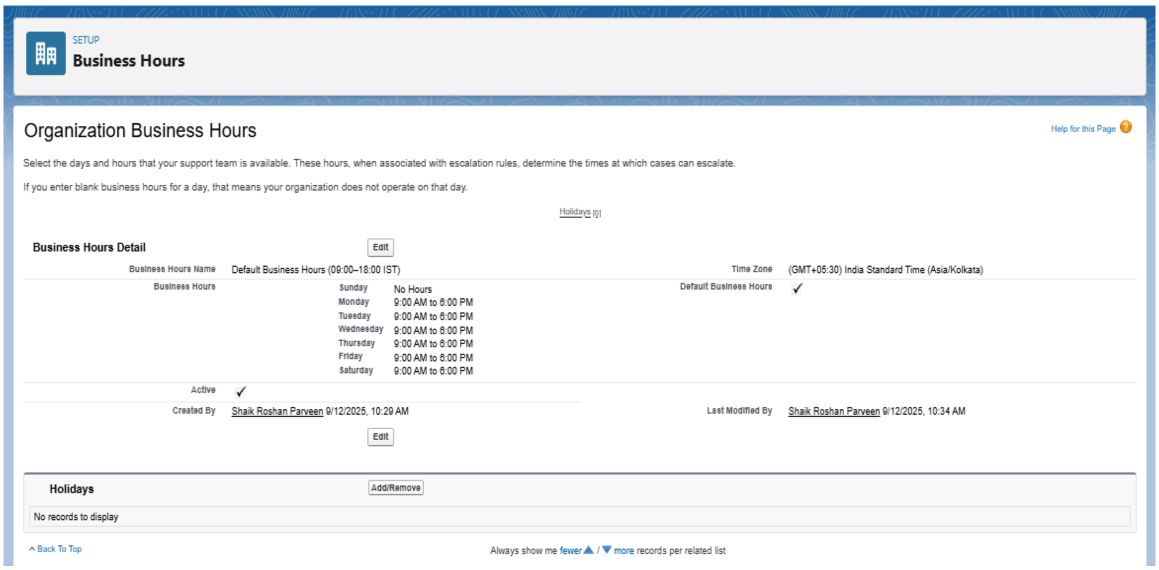
2. Company Profile Setup

- **Setup → Quick Find → Company Information**.
- Edited:
    - Time Zone → *Asia/Kolkata*
    - Default Currency → *INR*
    - Default Languages → *English, Telugu, Gujarati*
- Saved changes.



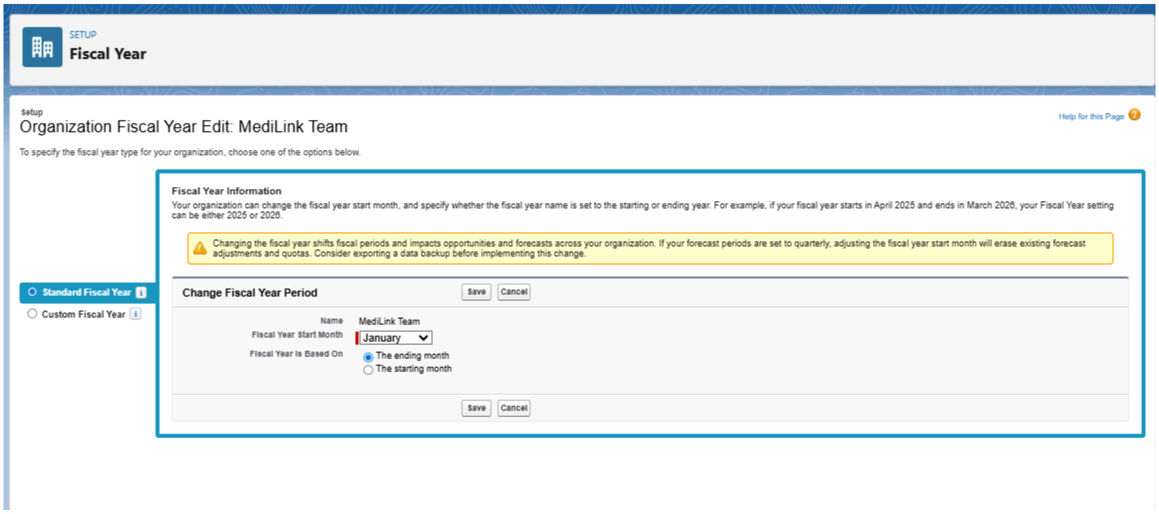
3. Business Hours & Holidays

- **Setup → Quick Find → Business Hours**.
- Clicked **New Business Hours**.
- Entered:
  - Label: *Hospital Business Hours*
  - Time Zone: Asia/Kolkata
- Checked **Active** and **Use these as default**.
- Set **Sunday → Closed**.
- Saved.



4. Fiscal Year Settings

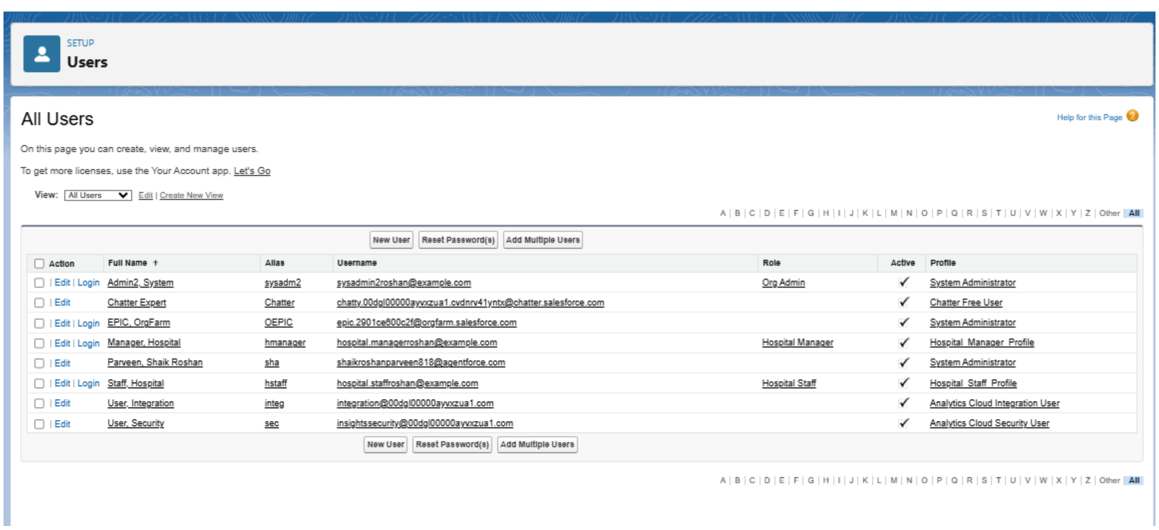
- **Setup → Quick Find → Fiscal Year**.
- Verified that *Standard Fiscal Year* is enabled.
- Did not enable custom fiscal year.



---

5. User Setup & Licenses

- **Setup → Quick Find → Users → Users → New User**.
- Filled required fields:
  - First Name, Last Name, Email, Username.
  - User License: *Salesforce Platform*.
  - Profile: (selected custom profile after creation).
  - Role: (selected role from hierarchy).
- Checked **Generate password**.
- Saved.
- Repeated for each test user (Hospital Manager, Hospital Staff).

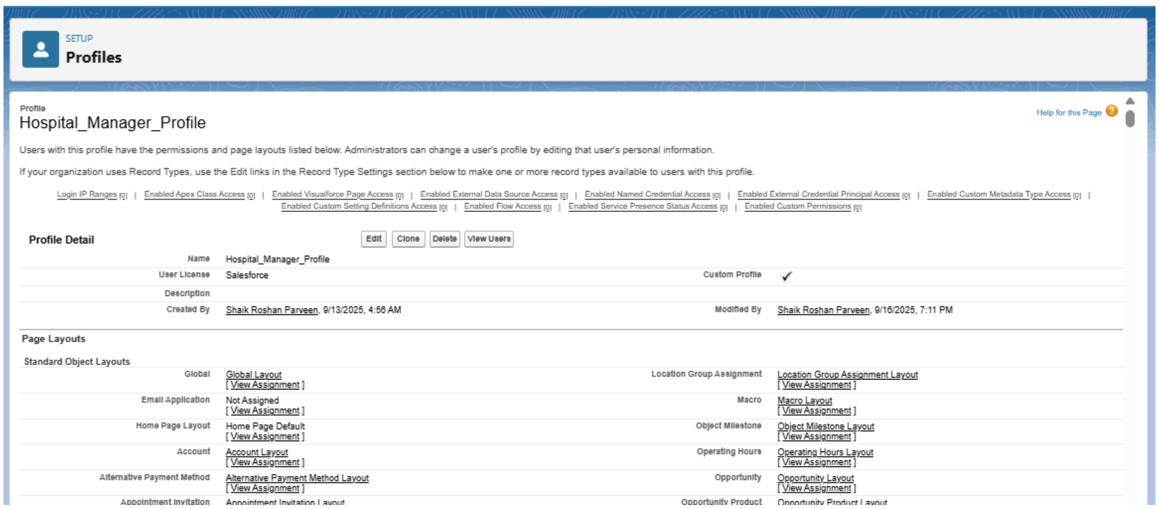


Action	Full Name ↑	Alias	Username	Role	Active	Profile
Edit | Login	Admin2, System	sysadm2	sysadmin2roshan@example.com	Org Admin	✓	System Administrator
Edit	Chatter Expert	Chatter	chatty.00dgl00000ayvxzua1.cvdnrv41yntx@chatter.salesforce.com		✓	Chatter Free User
Edit | Login	EPIC, OrgFarm	OEPIC	epic.2901ce600c2f@orgfarm.salesforce.com		✓	System Administrator
Edit | Login	Manager, Hospital	hmanager	hospital.managerroshan@example.com	Hospital Manager	✓	Hospital_Manager_Profile
Edit	Parveen, Shaik Roshan	sha	shaikroshanparveen818@agentforce.com		✓	System Administrator
Edit | Login	Staff, Hospital	hstaff	hospital.staffroshan@example.com	Hospital Staff	✓	Hospital_Staff_Profile
Edit	User, Integration	integ	integration@00dgl00000ayvxzua1.com		✓	Analytics Cloud Integration User
Edit	User, Security	sec	insightssecurity@00dgl00000ayvxzua1.com		✓	Analytics Cloud Security User

---

6. Profiles

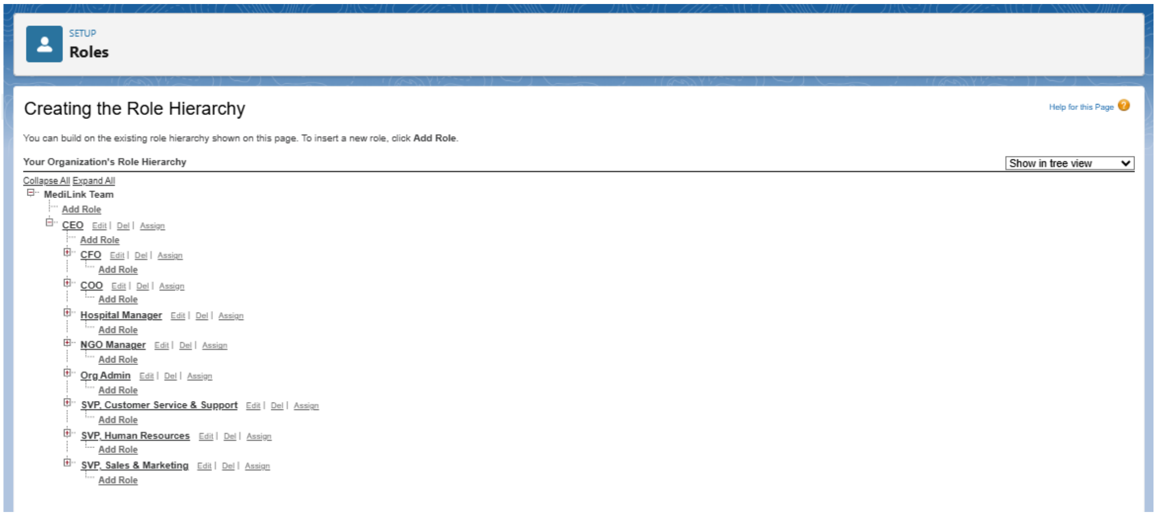
- **Setup → Quick Find → Profiles**.
- Selected **Standard User → Clone**.
- Named it **Hospital_Manager_Profile**.
- Edited **Object Settings**:
  - Donor__c → Read, Create, Edit, Delete.
  - Request__c → Read, Create, Edit, Delete.
  - Appointment__c → Read, Create, Edit.
- Set **Tabs → Default On** for Donor, Request, Appointment.
- Saved.
- Repeated to create **Hospital_Staff_Profile** (no Delete permission on Request).



---

7. Roles & Role Hierarchy

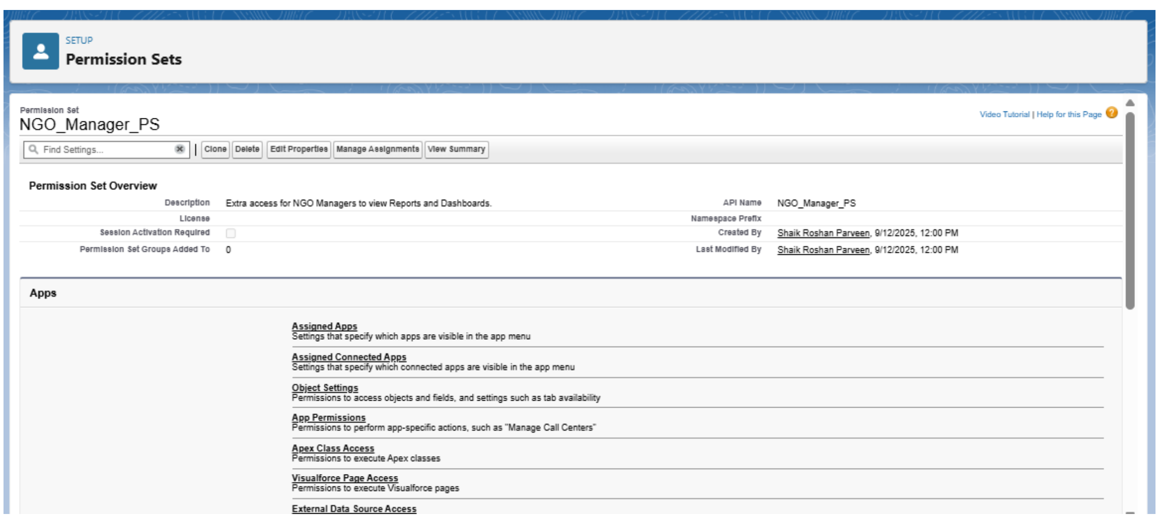
- **Setup → Quick Find → Roles → Set Up Roles**.
- Clicked **Add Role**.
- Created roles in hierarchy:
  - System Admin (top)
  - Hospital Manager
  - Hospital Staff
- Saved each role.



---

8. Permission Sets (Optional)

- **Setup → Quick Find → Permission Sets → New**.
- Created:
  - NGO_Manager_PS
  - Donor_Portal_Access_PS
- Saved (assignments not done yet).



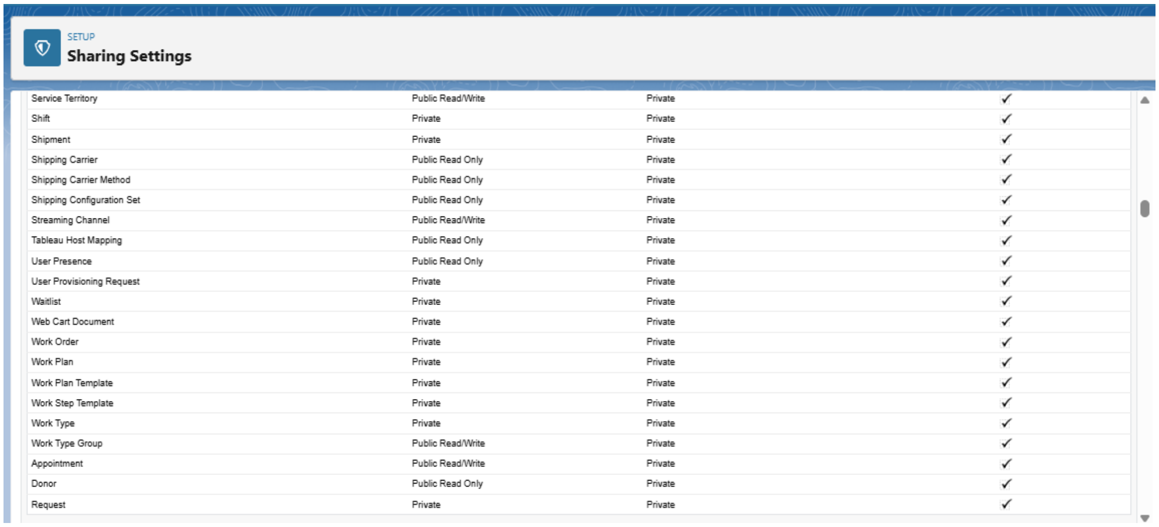
---

9. Org-Wide Defaults (OWD)

- **Setup → Quick Find → Sharing Settings**.
- Edited OWD defaults:
  - Donor__c → Public Read

- Request__c → Private
- Appointment__c → Controlled by Parent

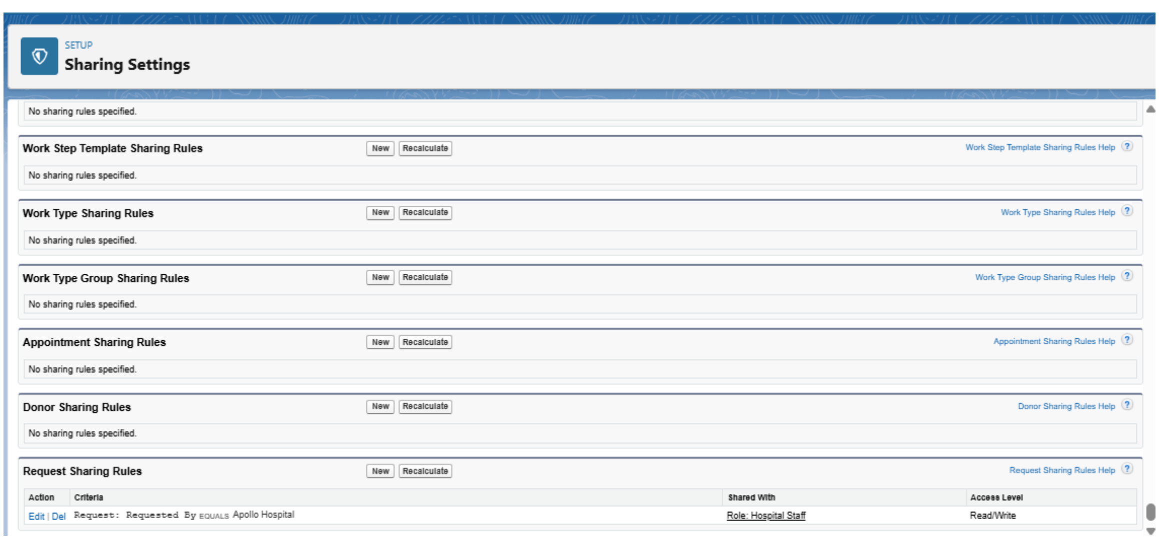
- Save



---

10. Sharing Rules

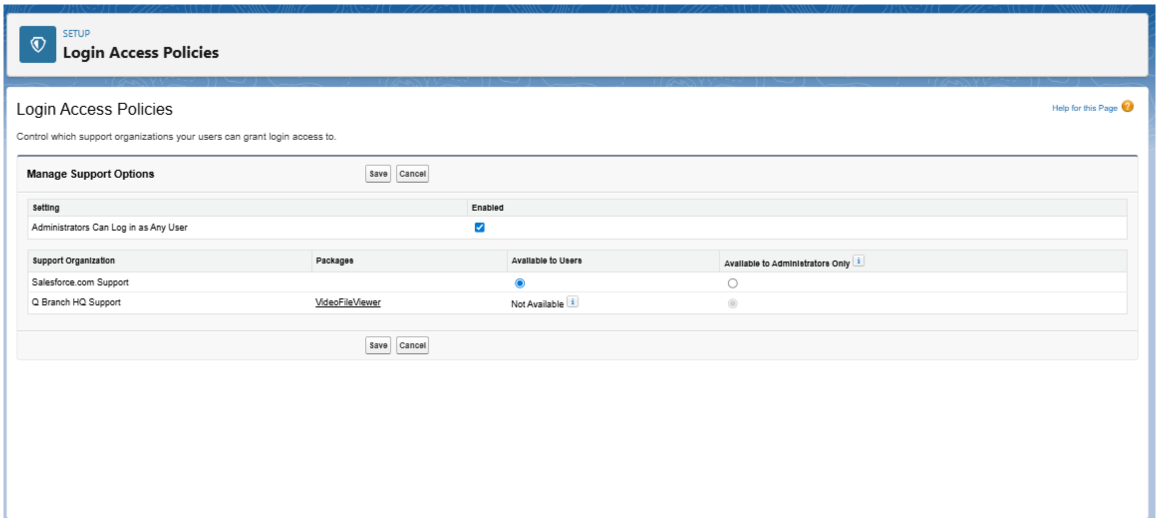
- **Setup** → **Quick Find** → **Sharing Settings** → Scroll to Request Sharing Rules.
- Clicked **New Sharing Rule**.
- Chose criteria: Requested by Apollo Hosipital
- Saved.



---

11. Login Access Policies

- **Setup → Quick Find → Login Access Policies**.
- Checked **Administrators Can Log in as Any User**.
- Saved.



☐ Outcome

At the end of Phase 2, I successfully configured:

- Company details (timezone, currency, languages).
- Business hours & fiscal year.
- Users, profiles, and roles.
- Security model with OWD & sharing rules.
- Admin login access for user testing.

Phase 3 Documentation – Data Modeling & Relationships

☐ Goal of Phase 3

To design and implement the data model for the Hospital Donor Management System using Salesforce objects, fields, record types, layouts, and relationships. This ensures that all donor, request, and appointment data is properly structured, accessible, and secure.

---

1 Standard & Custom Objects

What was done:

- Reviewed **Standard Objects** (like User, Contact).
- Created **Custom Objects** specific to the project:
  - **Donor__c** → Stores donor details.
  - **Request__c** → Stores patient donation requests (blood/organ).
  - **Appointment__c** → Stores scheduled donation appointments.

□ **Output:** Custom objects for Donor, Request, Appointment created.

---

2 Fields

What was done:

- Added necessary **Custom Fields** to capture data:

Donor__c Fields:

- Name, Gender, DOB, Phone, Email, Address, Blood Group, Organ Donor (Checkbox), Availability.

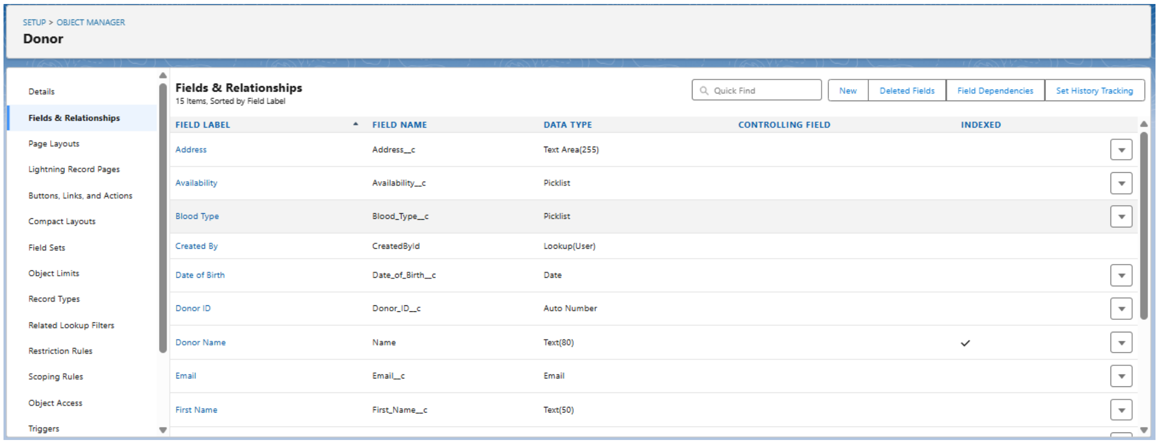
Request__c Fields:

- Request Type (Blood/Organ), Urgency, Status, Related Donor Lookup.

Appointment__c Fields:

- Appointment Date, Time, Notes, Donor Lookup, Request Lookup.

□ **Output:** Custom fields created per object.



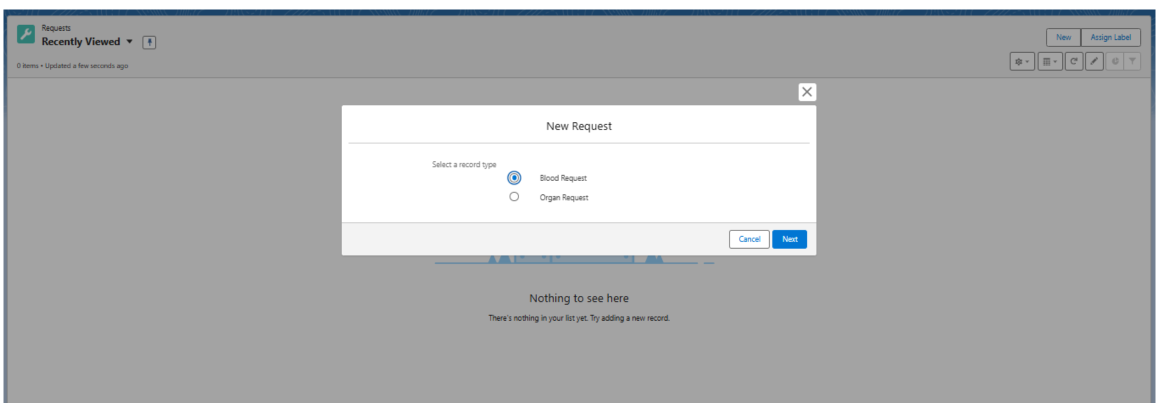
---

3 Record Types

What was done:

- Applied **Record Types** only where needed:
  - **Donor__c:** Default record type only (all donors are similar).
  - **Request__c:** Two record types created → **Blood Request** & **Organ Request**.
  - **Appointment__c:** Default record type only.

☐ **Output:** Record Types designed to separate blood vs. organ requests.



---

4 Page Layouts

What was done:

- Created **different layouts for Manager & Staff** profiles.

Donor__c Layouts:

- **Manager Layout:** Full access to all donor fields (Name, Gender, DOB, Phone, Email, Availability, Organ Donor).
- **Staff Layout:** Limited view (hides sensitive fields like DOB, Email, Phone).

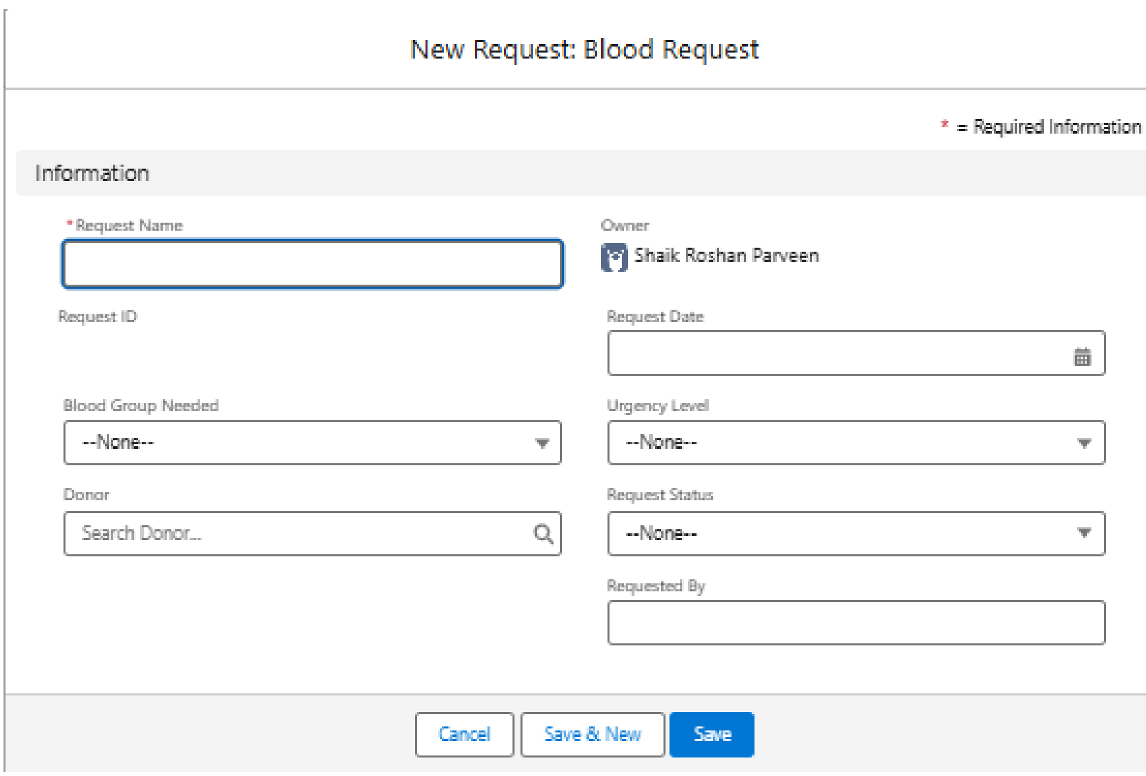
Request__c Layouts:

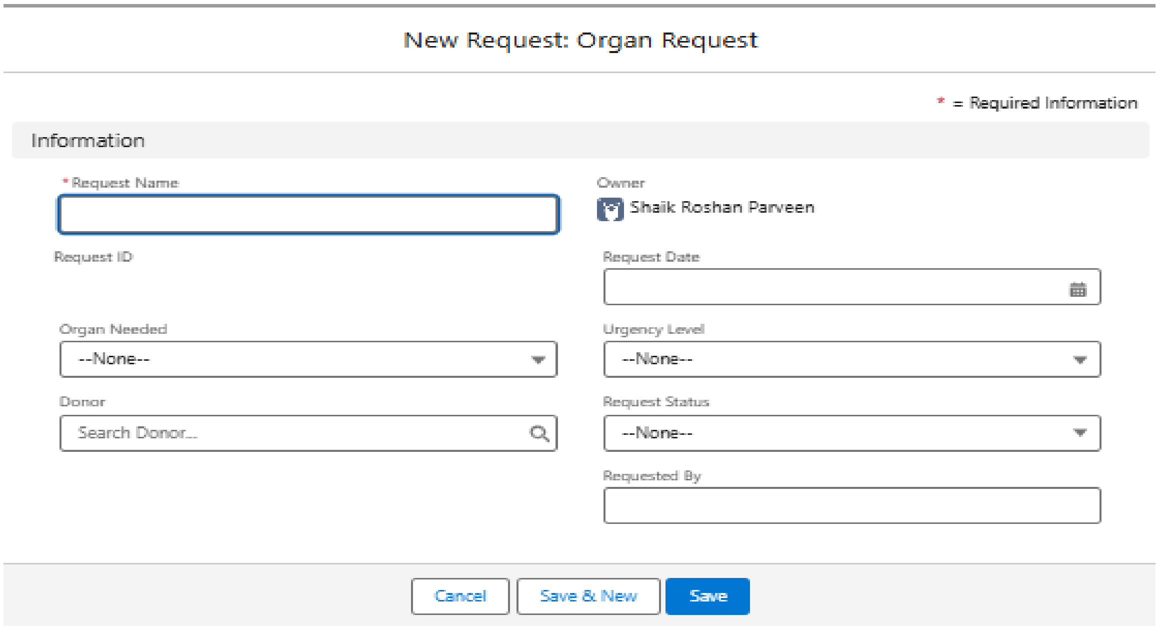
- **Manager Layout:** Full details including urgency & requested by.
- **Staff Layout:** Can create/edit but restricted from viewing/deleting manager-only fields.

Appointment__c Layouts:

- **Manager Layout:** Can see appointment notes.
- **Staff Layout:** Notes field hidden.

☐ **Output:** Separate layouts for each profile with controlled visibility.



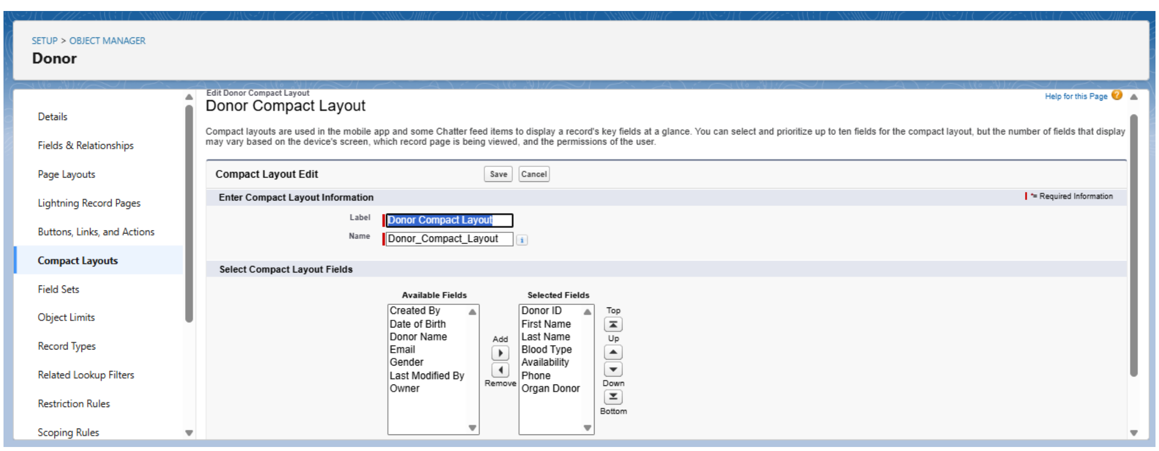


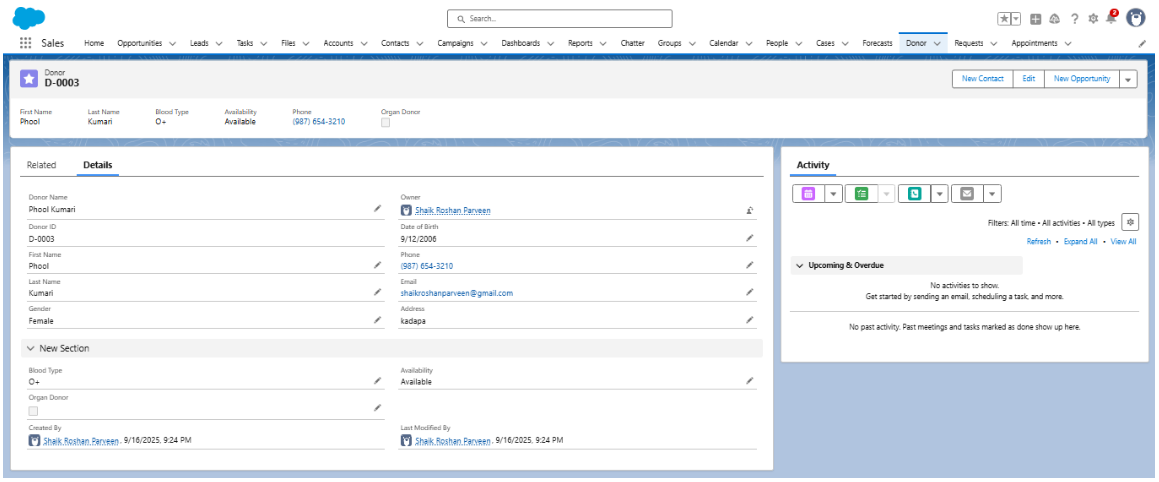
5 Compact Layouts

What was done:

- Created **Compact Layouts** for quick view in highlights panel.
- Example: Donor__c → Name, Blood Group, Availability, Phone.

☐ **Output:** Compact layouts applied to display key fields in record previews.



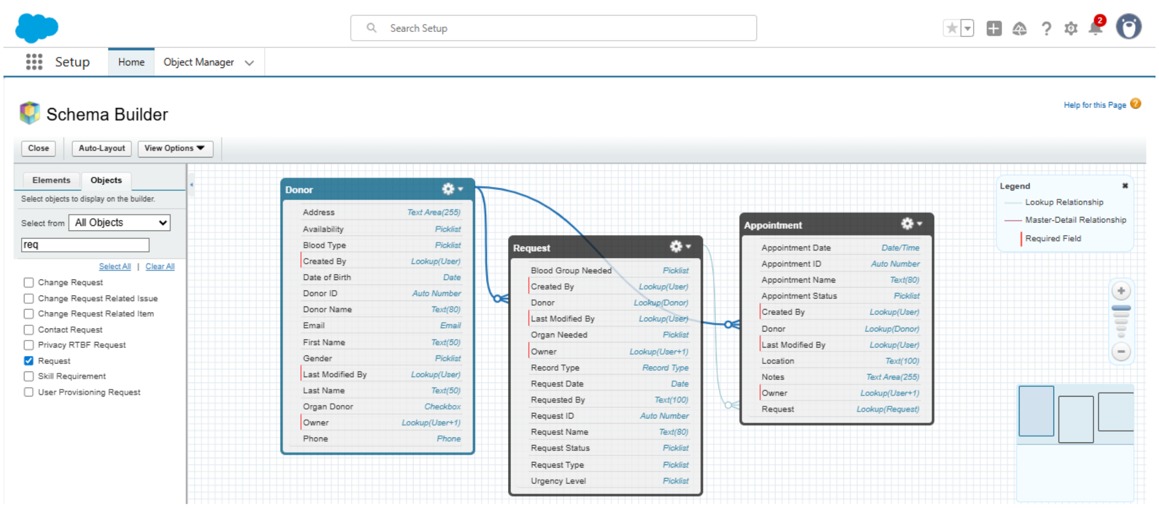


6 Schema Builder

What was done:

- Used **Schema Builder** to visualize objects and their relationships.
- Verified **Donor ↔ Request ↔ Appointment** connections.
- Ensured all relationships (lookup/master-detail) are properly mapped.

☐ **Output:** Graphical data model available for reference.



7 Relationships

What was done:

- Applied correct relationship types:
- **Lookup Relationship:** Appointment__c → Donor__c (one donor can have many appointments).
- **Lookup Relationship:** Appointment__c → Request__c (one request can have many appointments).

 Output: Clear relationship structure defined between objects.

Phase 4: Process Automation

Overview

In Phase 4, we implemented **Process Automation** in Salesforce to streamline operations for our *Blood & Organ Donation CRM*. This ensures data quality, reduces manual effort, and enables faster response during emergencies. We explored **Validation Rules, Flows, Email Alerts, Field Updates and Approval Processes**.

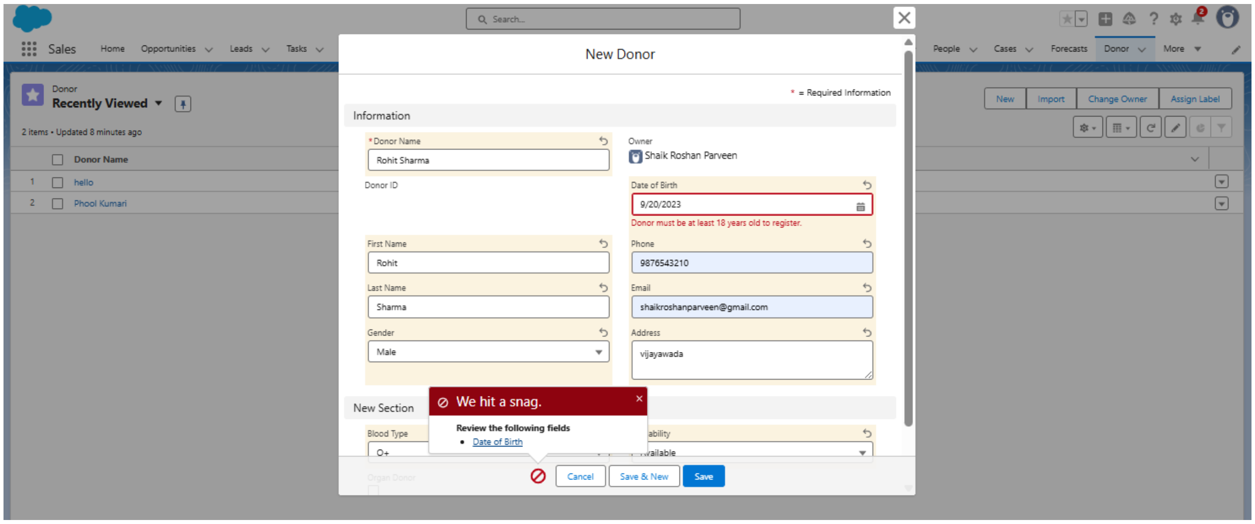
---

1. Validation Rules

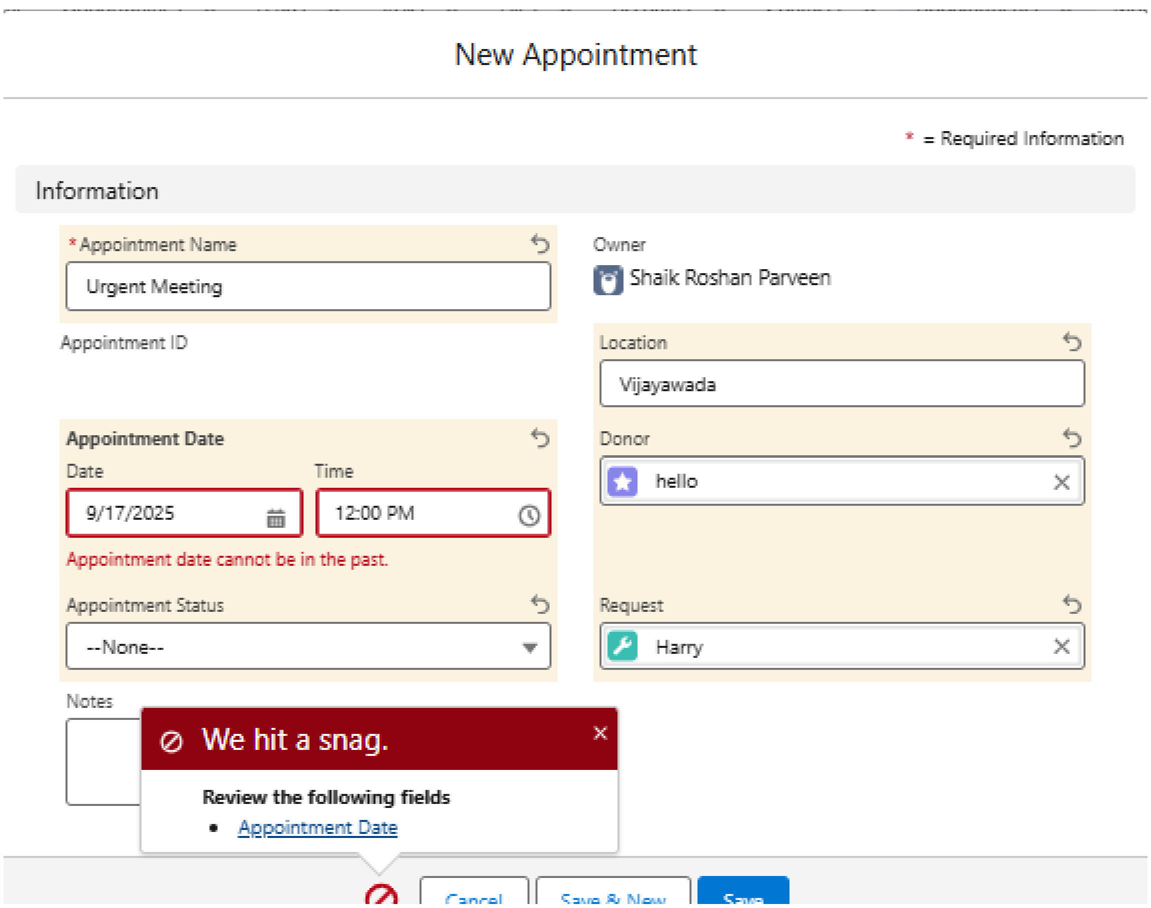
Purpose: Ensure data accuracy and prevent invalid records.

Rules Implemented:

1. **Donor Minimum Age** — Donors must be at least 18 years old.
   - Formula: `TODAY() - Date_of_Birth__c < 6570`

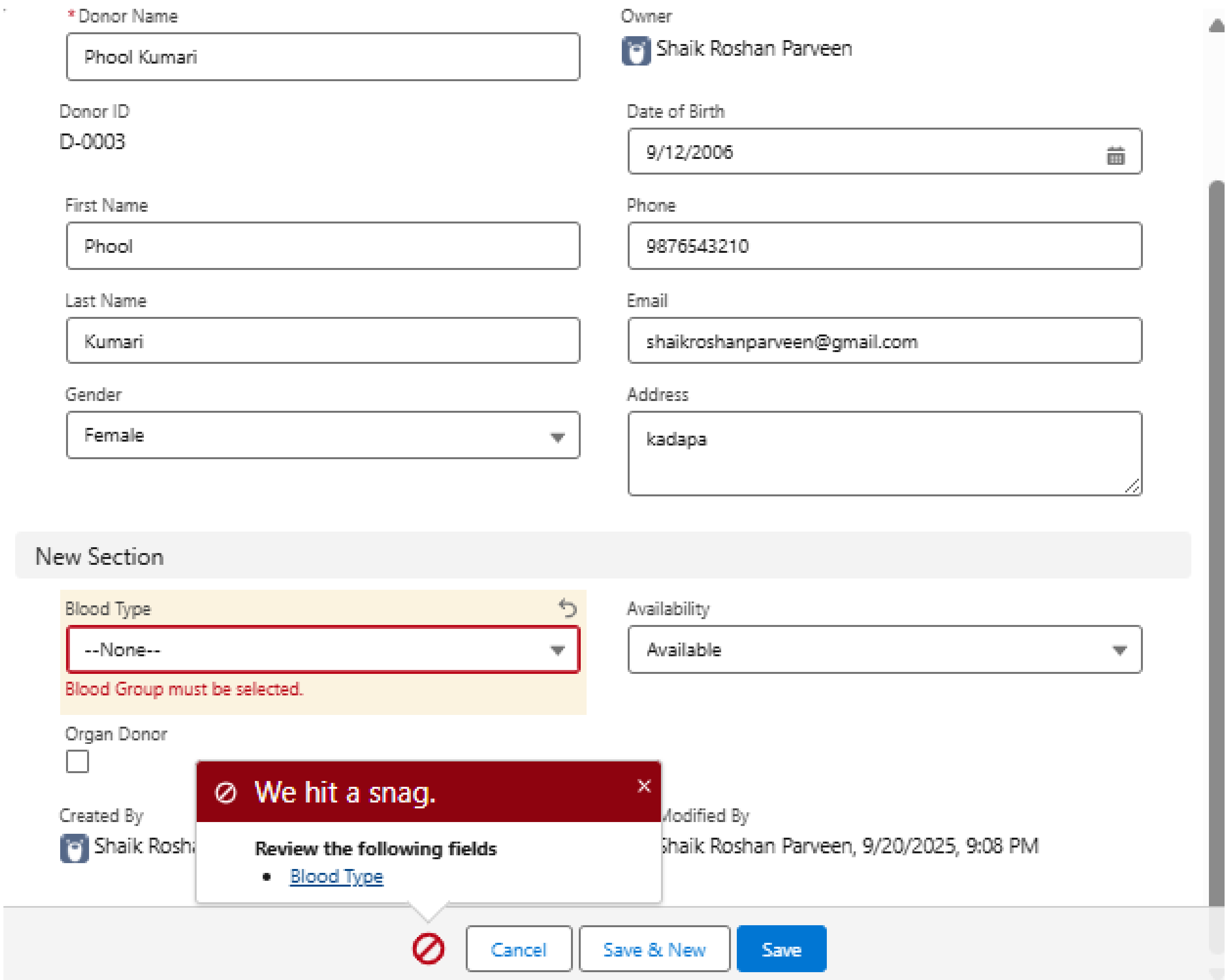


2. **Appointment Date Check** — Appointment date/time cannot be in the past.
    - Formula: `Appointment_Date__c < NOW()`



3. **Blood Group Required** — Donor must have a blood group selected.
   - Formula: `ISBLANK(TEXT(Blood_Type__c))`

Outcome: Only valid donor, appointment, and request data can be saved in the system.



---

2. Workflow Rules

Not implemented as standalone in Phase 4; replaced by more advanced Flow automations for better scalability and control.

---

3. Process Builder

Existing use cases replaced with Flow Builder for automating field updates and sending notifications.

---

4. Flow Builder

Purpose: Most powerful automation tool (Screen, Record-Triggered, Scheduled, Auto-launched).

Flows Implemented:

1. **Record-Triggered Flow (Total Units Available)**
   - Trigger: On Blood_Request__c create/edit.
   - Logic:
     - Get all donors with matching blood type.
     - Sum their `Units_Available__c`.
     - Update Request field `Total_Units_Available__c`.
   - Condition: If Total Units ≥ Required Units → request is marked “Fulfillable.”

Outcome: Automatically calculates donor availability for each request.

2. Record-Triggered Flow (Appointment Fulfillment)

- **Trigger:** On `Appointment__c` create/edit.
- **Logic:**
  - Get the related `Request__c` record.
  - Check if `Request_Status__c = In Progress` and `Donor__r.Availability__c = "Available"`.
  - If true →
    - Send **Email Alert** (appointment confirmation).
    - Update the related `Request__c.Request_Status__c = "Fulfilled"`.
- **Condition:** Only runs when Appointment is created or updated for a donor linked to an In Progress request.
- appointment is scheduled with an available donor.

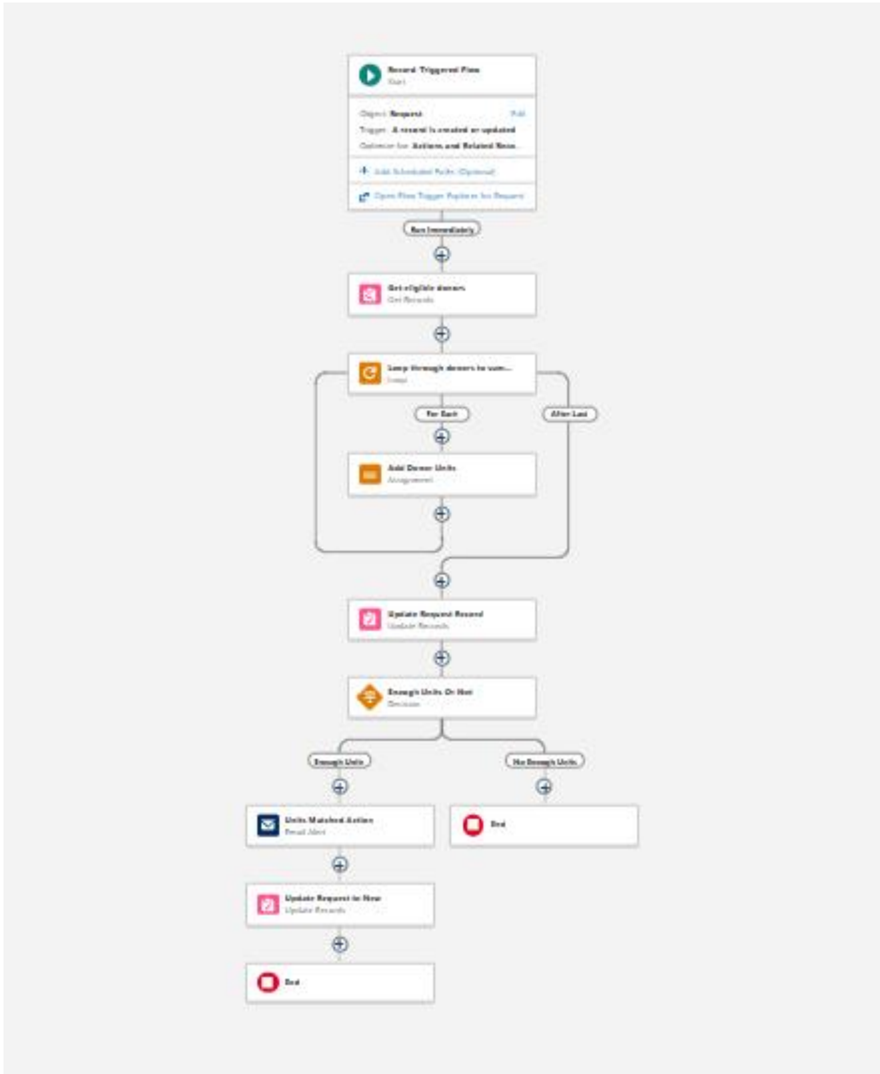


Fig: Total Units Available

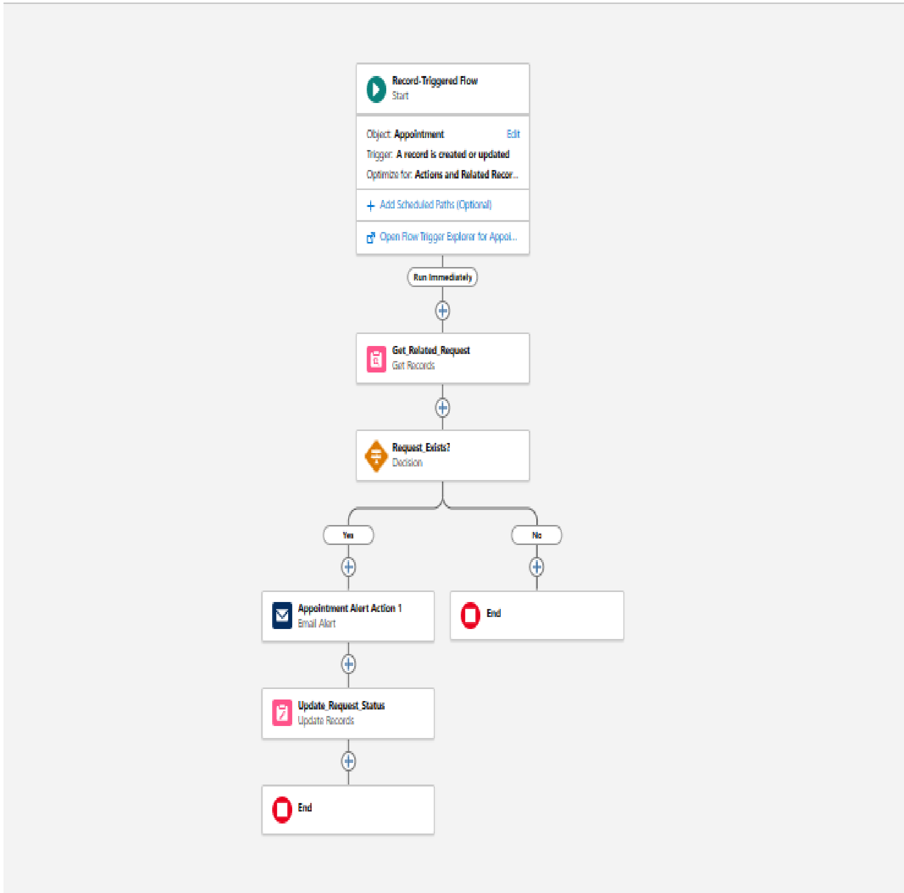


Fig: Appointment Fullfillment

---

5. Email Alerts

- Configured **Classic Email Templates** (New Request Notification).
- Appointment Confirmation Email, Blood Request Fulfillable Notification, Blood Request Approved.

Outcome: Consistent, automated communication.

Shaik Roshan Parveen via fqruulm4qvtikn.gl-ayvxzua1.can98.bnc.salesforce.com 8:48 PM (2 hours ago)
to hospital.staff@yourdomain.com, sysadmin2@yourdomain.com, hospital.manager@yourdomain.com, me

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

Report not spam

Hello ,

The blood request re8 can now be fulfilled.

Blood Type: O+
Units Required: 5
Total Units Available: 20
Please take necessary action.

Thanks,
MediLink Org

Appointment Scheduled for Request:JMJ hospital Spam ×

Shaik Roshan Parveen via yda7eyh9kicdx9.gl-ayvxzua1.can98.bnc.salesforce.com 9:08 PM (2 hours ago)
to hospital.staff@yourdomain.com, sysadmin2@yourdomain.com, hospital.manager@yourdomain.com, me

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

Report not spam

Hello Coordinator,

An appointment has been scheduled for donor .
The request dear is now Fulfilled.

Hello {!Request__c.Requested_By__c},

Good news! Your blood request has been **approved**.

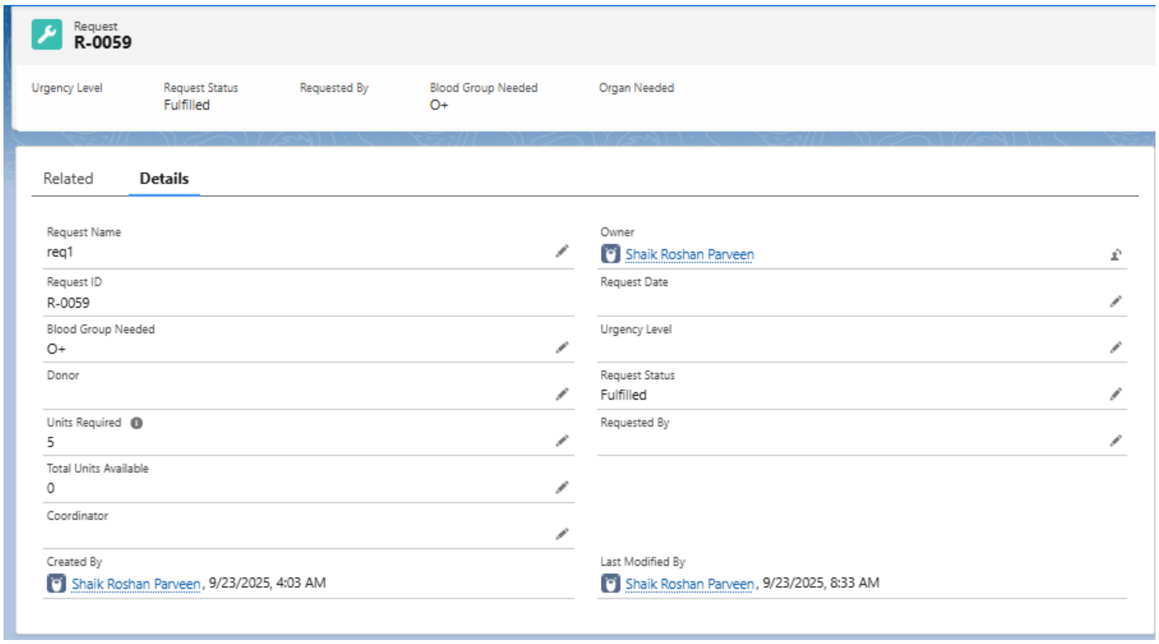
Request Details:
- Request ID: {!Request__c.Request_Name__c}
- Units Approved: {!Request__c.Units_Required__c}
- Blood Group: {!Request__c.Blood_Group_Needed__c}

You will be contacted shortly regarding the next steps.

Thank you,
Medilink Org

6. Field Updates

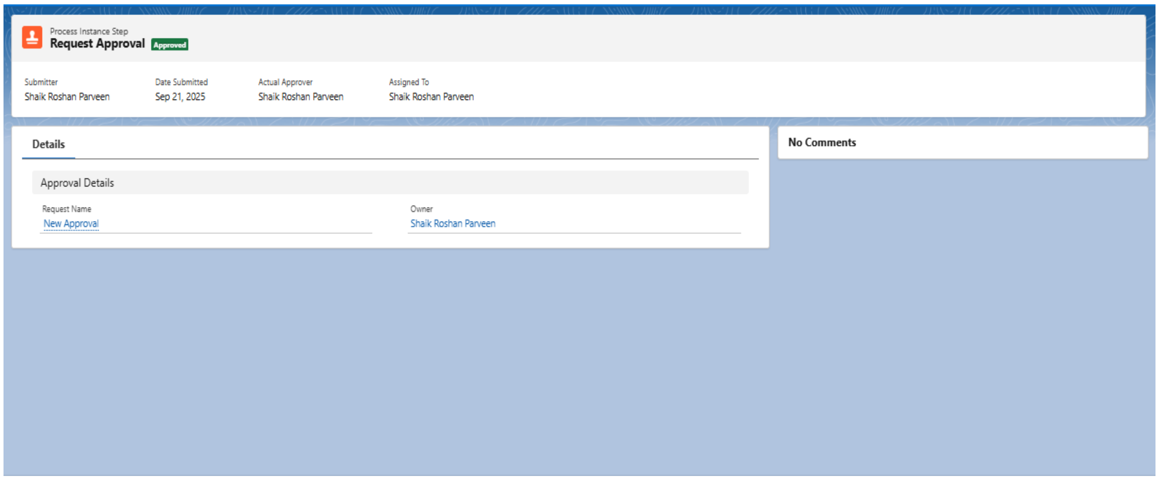
- In Process Builder, `Status__c` automatically updates to **Closed** once fulfilled.
- Helps maintain request lifecycle without manual intervention.

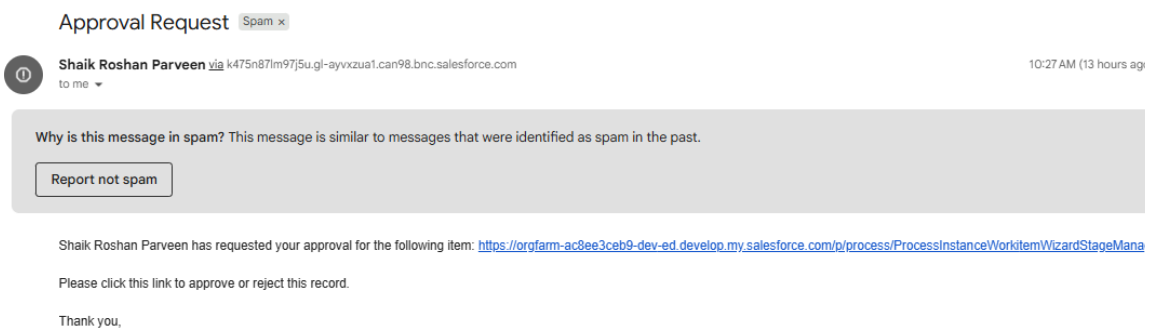


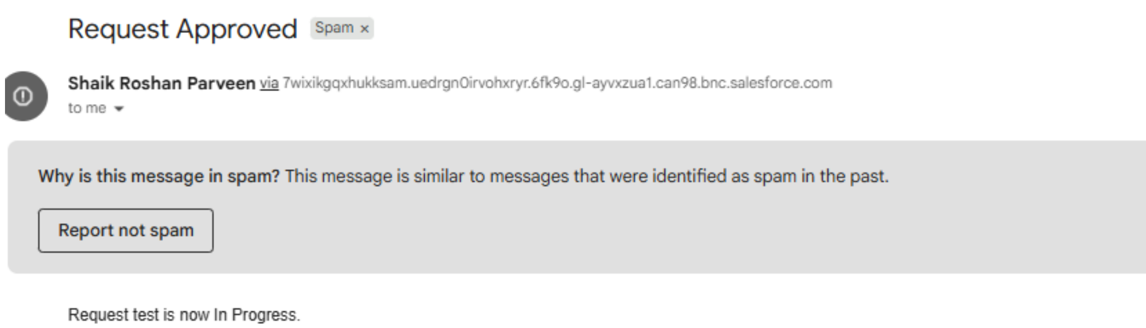
9. Approval Process

- Designed an **Approval Process** for **Blood Donation Requests**.
- Request goes through: Coordinator(or any user assigned).
- Once approved, request is marked as “Approved for Blood.”

Outcome: Sensitive requests follow compliance and multi-level approval.


Process Instance Step
Request Approval
Approved
Submitter
Shaik Roshan Parveen
Date Submitted
Sep 21, 2025
Actual Approver
Shaik Roshan Parveen
Assigned To
Shaik Roshan Parveen
Details
No Comments
Approval Details
Request Name
New Approval
Owner
Shaik Roshan Parveen


Approval Request
Spam
Shaik Roshan Parveen via k475n87lm97j5u.gl-ayvxzua1.can98.bnc.salesforce.com
10:27 AM (13 hours ag
to me
Why is this message in spam? This message is similar to messages that were identified as spam in the past.
Report not spam
Shaik Roshan Parveen has requested your approval for the following item: https://orgfarm-ac8ee3ceb9-dev-ed.develop.my.salesforce.com/p/process/ProcessInstanceWorkitemWizardStageMana
Please click this link to approve or reject this record.
Thank you,


Request Approved
Spam
Shaik Roshan Parveen via 7wixikgqxhukksam.uedrgn0irvohxryr.6fk9o.gl-ayvxzua1.can98.bnc.salesforce.com
to me
Why is this message in spam? This message is similar to messages that were identified as spam in the past.
Report not spam
Request test is now In Progress.

Conclusion

Phase 4 introduced **intelligent automation** into the CRM system:

- Data is validated before saving.
- Coordinators and volunteers are notified instantly.
- Status updates and donor availability calculations happen automatically.
- Sensitive requests follow a structured approval chain.

This automation reduces errors, saves time, and ensures **faster life-saving response** in critical donation scenarios.

Phase 5: Apex Programming (Developer)

Overview

In Phase 5, we implemented **Apex Programming** to extend automation beyond declarative tools. Apex was used for custom logic to process blood requests, manage donor availability, and schedule automated jobs. This ensured **real-time updates, data consistency, and scalability**.

---

1. Apex Classes & Objects

Implemented Classes:

- **`BloodRequestHandler.cls`**
  - Contains core logic for processing blood requests.
  - Prevents recursion using a static flag.
  - Calculates total units available by blood type.
  - Deducts donor units when a request is fulfillable.
  - Updates request status (`New → In Progress`).
- **`DailyBloodRequestScheduler.cls`**
  - Implements `Schedulable` interface.
  - Runs daily to re-check pending requests.
  - Updates donor availability and request status in bulk.

```
public class BloodRequestHandler {

    // Static variable to prevent recursion
    public static Boolean isProcessing = false;

    public static void processRequests(List<Request__c> reqList) {

        // Prevent recursion
        if(isProcessing) return;
        isProcessing = true;

        if(reqList == null || reqList.isEmpty()) return;

        // Fetch fresh Request records from DB
        List<Request__c> freshRequests = [
            SELECT Id, Name, Blood_Group_Needed__c, Units_Required__c, Request_Status__c, Total_Units_Available__c
            FROM Request__c
            WHERE Id IN :reqList
        ];

        // Collect all blood types needed
        Set<String> bloodTypes = new Set<String>();
```

2. Apex Triggers (after insert, after update)

- **`BloodRequestTrigger.trigger`**
  - Fires on `Request__c` after insert and update.
  - Calls `BloodRequestHandler.processRequests()` to execute the business logic.
  - Ensures requests are processed immediately upon creation/update.

```
trigger BloodRequestTrigger on Request__c (after insert, after update) {

    List<Request__c> reqsToProcess = new List<Request__c>();
    for(Request__c req : Trigger.new){
        if(req.Units_Required__c != null){
            reqsToProcess.add(req);
        }
    }

    if(!reqsToProcess.isEmpty()){
        BloodRequestHandler.processRequests(reqsToProcess);
    }
}
```

3. Trigger Design Pattern

- Followed **one trigger per object** principle.
- Trigger kept lean by delegating logic to `BloodRequestHandler.cls`.
- Added a static Boolean variable to prevent recursive updates.

4. SOQL

- Used SOQL queries to fetch required records:
  - Fetch **Requests** with needed fields.
  - Fetch **Donors** filtered by blood type and availability.
- Example:
- `List<Donor__c> allDonors = [`
- `    SELECT Id, Name, Units_Available__c, Blood_Type__c`
- `    FROM Donor__c`
- `    WHERE Blood_Type__c IN :bloodTypes`
- `    AND Units_Available__c > 0`
- `    ORDER BY Units_Available__c DESC`
- `];`

---

5. Collections: List, Set, Map

- **List:** Stored donors to be updated in bulk.
- **Set:** Collected distinct blood types required by pending requests.
- **Map:** Grouped donors by blood type (`Map<String, List<Donor__c>>`).

---

6. Control Statements

- **If-Else:** Checked whether requests had enough donors.
- **For Loops:** Iterated through donors to sum units and deduct availability.
- **Break Statements:** Stopped looping once required units were deducted.

---

7. Scheduled Apex

- **`DailyBloodRequestScheduler.cls`** implemented `Schedulable`.
- Runs daily to:
  - Find pending requests.
  - Recalculate donor availability.
  - Update request statuses accordingly.

Outcome: Automation runs in background without manual intervention.

```
global class DailyBloodRequestScheduler implements Schedulable {

    global void execute(SchedulableContext sc) {
        processPendingRequests();
    }

    // Public method you can call manually for testing
    public void processPendingRequests() {

        // 1 Get all pending requests (status not 'In Progress')
        List<Request__c> pendingRequests = [
            SELECT Id, Blood_Group_Needed__c, Units_Required__c, Request_Status__c, Total_Units_Available__c
            FROM Request__c
            WHERE Request_Status__c != 'In Progress'
              AND Units_Required__c != null
        ];

        if(pendingRequests.isEmpty()) return;

        // 2 Collect all blood types needed
        Set<String> bloodTypesNeeded = new Set<String>();
        for(Request__c req : pendingRequests){
```

☐ Conclusion

Phase 5 introduced **Apex-driven automation** into the project. With handler classes, triggers, SOQL queries, collections, and a scheduled job, the system now:

- Automatically matches donors with requests.
- Deducts donor units in real time.
- Updates request statuses (`New` → `In Progress`).
- Re-checks pending requests daily.

This ensures the CRM can **scale**, remain **data-consistent**, and handle **complex donor-request matching** beyond point-and-click automation.