EX NO:1	Write the complete problem statement
DATE	

AIM:

To prepare PROBLEM STATEMENT for any project.

ALGORITHM:

1. The problem statement is the initial starting point for a project.
2. A problem statement describes what needs to be done without describing how.
3. It is basically a one-to-three-page statement that everyone on theproject agrees with that describes what will be done at a high level.
4. The problem statement is intended for a broad audience and shouldbe written in non- technical terms.
5. It helps the non-technical and technical personnel communicate byproviding a description of a problem.
6. It doesn't describe the solution to the problem.

INPUT:

1. The input to requirement engineering is the problem statement prepared by customer.
2. It may give an overview of the existing system along with broad expectations from the new system.
3. The first phase of requirements engineering begins with requirements elicitation i.e. gathering of information about requirements.
4. Here, requirements are identified with the help of customer and existing system processes.

Problem:

Blood banks play a critical role in healthcare by ensuring the availability of blood for emergencies, surgeries, and regular medical needs. However, many blood banks face challenges in managing patient registrations, inventory, hospital requests, and appointments, often due to reliance on outdated or manual systems. This can lead to inefficient management, delayed blood provision, and, in some cases, unfulfilled hospital needs. To address these issues, a centralized Blood Bank Management System is needed to streamline operations and ensure timely access to blood supplies for hospitals and patients.

Background:

The demand for blood has been steadily increasing, but blood banks are often hindered by inefficient management processes. Many blood banks still use manual methods for tracking donations, requests, and inventory, resulting in delays, errors, and limited visibility into real-time inventory levels. A centralized system could address these limitations by integrating all essential functionalities such as patient registration, appointment booking, hospital request management, inventory control, and reporting, enabling efficient operations and reducing the risk of critical shortages.

Relevance:

An efficient Blood Bank Management System is crucial for saving lives. By improving the management of blood donations and supplies, this system can ensure that blood is readily available when and where it is needed. Such a system benefits patients, hospitals, and blood banks by reducing wait times, preventing stockouts, and providing accurate data for decision-making. Additionally, automated and streamlined processes enhance the operational efficiency of blood banks, allowing them to serve more effectively.

Objectives:

The primary objective of this project is to develop a centralized Blood Bank Management System that enhances operational efficiency, reduces errors, and ensures a steady supply of blood to meet patient and hospital needs. Specific objectives include:

1. **Patient Registration**: Enable efficient registration of new patients, capturing their details and medical history.
2. **Appointment Booking**: Allow patients to book appointments for blood donation or transfusion with ease.
3. **Hospital Blood Requests**: Facilitate hospitals in requesting specific blood types and quantities based on real-time inventory levels.
4. **Approval and Management of Requests**: Provide blood bank administrators the tools to review and approve requests from hospitals, ensuring alignment with available inventory.
5. **Inventory Management**: Implement real-time tracking of blood stock levels, including donor blood types, and manage the storage and expiration of blood units.
6. **Report Generation**: Generate comprehensive reports on blood inventory, donation appointments, and request fulfillments, aiding in compliance and resource planning.

Result:

EX NO:2	Write the software requirement specification document
DATE	

AIM:

To do requirement analysis and develop Software Requirement Specification Sheet(SRS) for any Project.

ALGORITHM:

SRS shall address are the following:

a) **Functionality.** What is the software supposed to do?

b) **External interfaces.** How does the software interact with people, the system‘s hardware, other hardware, and other software?

c) **Performance.** What is the speed, availability, response time, recovery time of various software functions, etc.?

d) **Attributes.** What is the portability, correctness, maintainability, security, etc. considerations?

e) **Design constraints imposed on an implementation.** Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

1. Introduction

1.1 Purpose

This document describes the requirements for developing a Blood Bank Management System (BBMS). The goal of this system is to streamline blood bank operations like patient registration, appointment booking, blood requests, inventory management, and reporting. By implementing BBMS, blood banks will be able to efficiently manage blood supplies, reduce manual errors, and provide faster services to patients and hospitals.

1.2 Scope

The BBMS will be a web-based application accessible by blood bank administrators, hospital staff, and blood donors. This system will handle all core functions, such as tracking blood inventory, managing requests, scheduling appointments, and generating reports, to keep the blood bank running smoothly and to ensure blood is available when needed.

1.3 Definitions, Acronyms, and Abbreviations

- **BBMS**: Blood Bank Management System
- **Admin**: Blood Bank Administrator
- **Donor**: Individual donating blood
- **Patient**: Individual who needs a blood transfusion
- **Hospital Staff**: Authorized personnel from hospitals who request blood

1.4 Overview
This document details the required functionalities, interfaces, and performance standards for BBMS. It's a roadmap for the development team and a reference for users to understand what the system will offer.

2. Overall Description

2.1 Product Perspective
BBMS is a centralized solution that brings together the most essential blood bank operations in one place. It will integrate with existing databases and be accessible online, making it easier for users to perform their roles without delays or mix-ups.

2.2 Product Functions

- **Patient Registration**: Register and manage patient profiles.
- **Appointment Booking**: Schedule donation and transfusion appointments.
- **Blood Requests**: Let hospitals request specific blood types.
- **Approval and Inventory Management**: Admins can approve blood requests and manage blood stock.
- **Report Generation**: Generate reports on blood inventory, donations, and usage.

2.3 User Classes and Characteristics

- **Admin**: Manages the overall blood bank operations, approves blood requests, and tracks inventory.
- **Hospital Staff**: Submit and track blood requests for their patients.
- **Donor**: Register, view, and book appointments for blood donation.
- **Patient**: Individuals who need blood transfusions, registered and managed by the system.

2.4 Operating Environment
The system will be accessible via a web browser and compatible with major desktop and mobile devices.

2.5 Design and Implementation Constraints

- The system must ensure privacy and security for patient data.
- It should be able to handle multiple users at once.

2.6 Assumptions and Dependencies

- Users will have internet access to log in.
- The system will need a database for storing all records and data.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Patient Registration Module

- The system allows patients to register with personal and medical information.
- The system validates information to ensure accuracy.
- Admins can view and update patient information if needed.

3.1.2 Appointment Booking Module

- Patients and donors can book, reschedule, or cancel appointments.
- Available slots for donations and transfusions are displayed.
- Users receive reminders for upcoming appointments via email or SMS.

3.1.3 Hospital Blood Request Module

- Hospital staff can request specific blood types and quantities.
- The system provides updates on the status of blood requests.
- Admins are notified of new requests as they come in.

3.1.4 Admin Approval and Inventory Management Module

- Admins can review and approve or deny requests based on stock availability.
- Admins manage blood inventory by adding, updating, or removing units as needed.
- The system alerts admins if blood units are close to expiry.

3.1.5 Report Generation Module

- The system generates reports on inventory, donation history, and request status.
- Users can download reports in PDF and Excel formats.
- Reports can be filtered by date, blood type, and request status.

3.2 Non-Functional Requirements

3.2.1 Performance Requirements

- The system should handle up to 1,000 users at once without slowing down.
- Response time should be under 2 seconds for all actions.

3.2.2 Security Requirements

- Users must log in to access the system.
- Sensitive data, like patient information, should be encrypted.
- Only authorized users should have access to certain functions based on their roles.

3.2.3 Usability Requirements

- The interface should be simple and easy to navigate.
- The system should be usable on both desktop and mobile devices.

3.2.4 Reliability Requirements

- The system should be operational 99.9% of the time.
- Any issues should be resolved within 5 minutes.

4. External Interface Requirements

4.1 User Interfaces

- The system will be responsive, adapting to different screen sizes for a smooth user experience.
- Each user type (admin, hospital staff, donor) will have access to functions relevant to their role.

4.2 Hardware Interfaces

- Compatible with standard desktop and mobile devices.

4.3 Software Interfaces

- The system will use an SQL database for secure data storage.
- The system will support SMS and email services for reminders and notifications.

5. Additional Requirements

5.1 Data Privacy and Compliance
All data, especially patient information, will be handled securely and in compliance with relevant data protection regulations.

5.2 Documentation
Clear user manuals and system documentation will be available for users and technical staff.

Result:

EX NO:3	Draw the entity relationship diagram
DATE	

AIM:

To Draw the Entity Relationship Diagram for any project.

ALGORITHM:

Step 1: Mapping of Regular Entity Types

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relation Types

Step 4: Mapping of Binary 1:N Relationship Types.

Step 5: Mapping of Binary M:N Relationship Types.

Step 6: Mapping of Multivalued attributes.

INPUT:

Entities

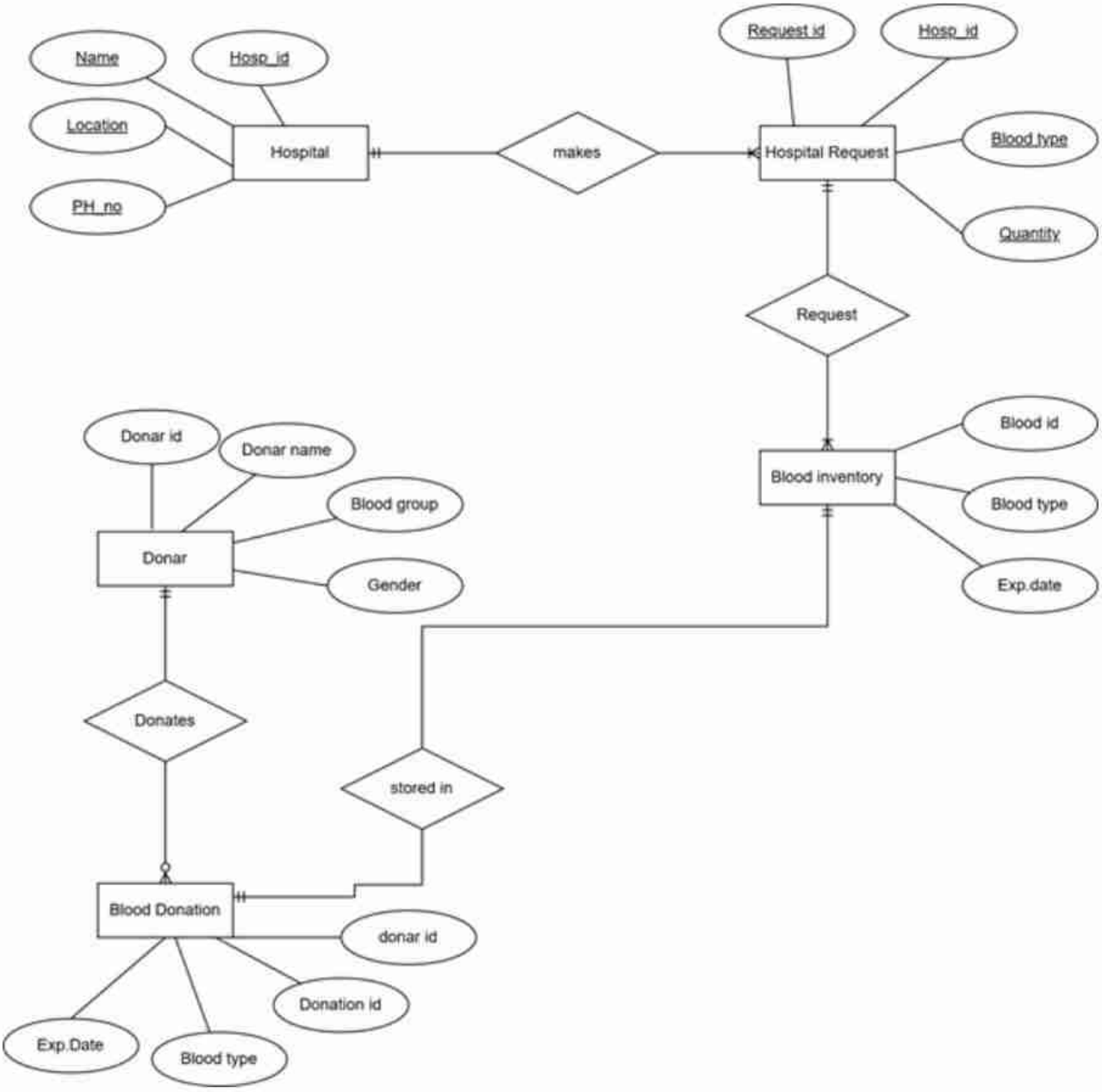
Entity Relationship Matrix

Primary Keys

Attributes

Mapping of Attributes with Entities

Result:


Name
Hosp_id
Location
PH_no
Hospital
makes
Request id
Hosp_id
Hospital Request
Blood type
Quantity
Request
Donar id
Donar name
Blood group
Donar
Gender
Blood id
Blood inventory
Blood type
Exp.date
Donates
stored in
Blood Donation
donar id
Donation id
Exp.Date
Blood type

EX NO:4	Draw the data flow diagrams at level 0 and level 1
DATE	

AIM:

To Draw the Data Flow Diagram for any project and List the Modules in the Application.

ALGORITHM:

1. Open the Visual Paradigm to draw DFD (Ex.Lucidchart)
2. Select a data flow diagram template
3. Name the data flow diagram
4. Add an external entity that starts the process
5. Add a Process to the DFD
6. Add a data store to the diagram
7. Continue to add items to the DFD
8. Add data flow to the DFD
9. Name the data flow
10. Customize the DFD with colours and fonts
11. Add a title and share your data flow diagram

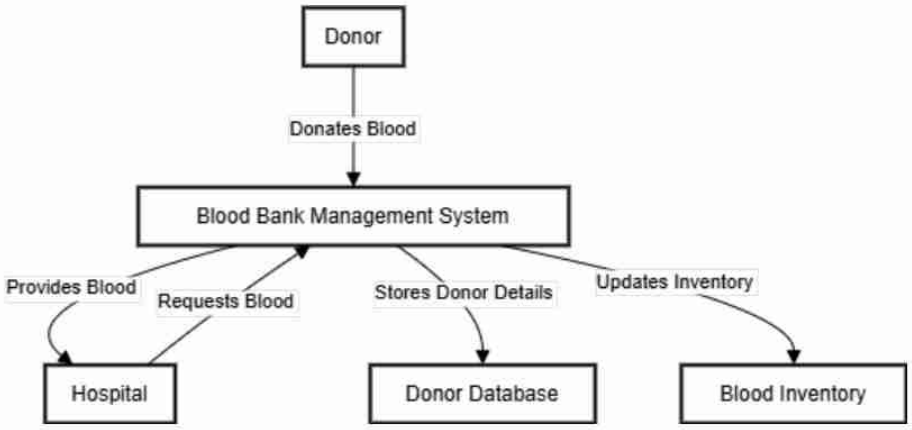
INPUT:

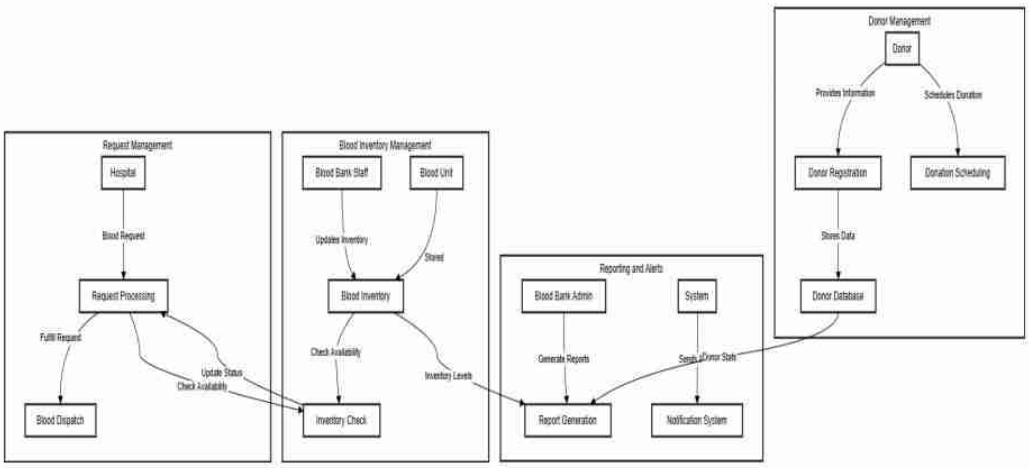
Processes

Datastores

External Entities

Result:


Donor
Donates Blood
Blood Bank Management System
Provides Blood
Requests Blood
Stores Donor Details
Updates Inventory
Hospital
Donor Database
Blood Inventory


Request Management
Hospital
Blood Request
Request Processing
Fulfill Request
Update Status
Check Availability
Blood Dispatch
Blood Inventory Management
Blood Bank Staff
Blood Unit
Updates Inventory
Stored
Blood Inventory
Check Availability
Inventory Levels
Inventory Check
Reporting and Alerts
Blood Bank Admin
System
Generate Reports
Sends Donor Stats
Report Generation
Notification System
Donor Management
Donor
Provides Information
Schedules Donation
Donor Registration
Donation Scheduling
Stores Data
Donor Database

EX NO:5	Draw use case diagram
DATE	

AIM:

To Draw the Use Case Diagram for any project

ALGORITHM:

Step 1: Identify Actors

Step 2: Identify Use Cases

Step 3: Connect Actors and Use Cases

Step 4: Add System Boundary

Step 5: Define Relationships

Step 6: Review and Refine

Step 7: Validate

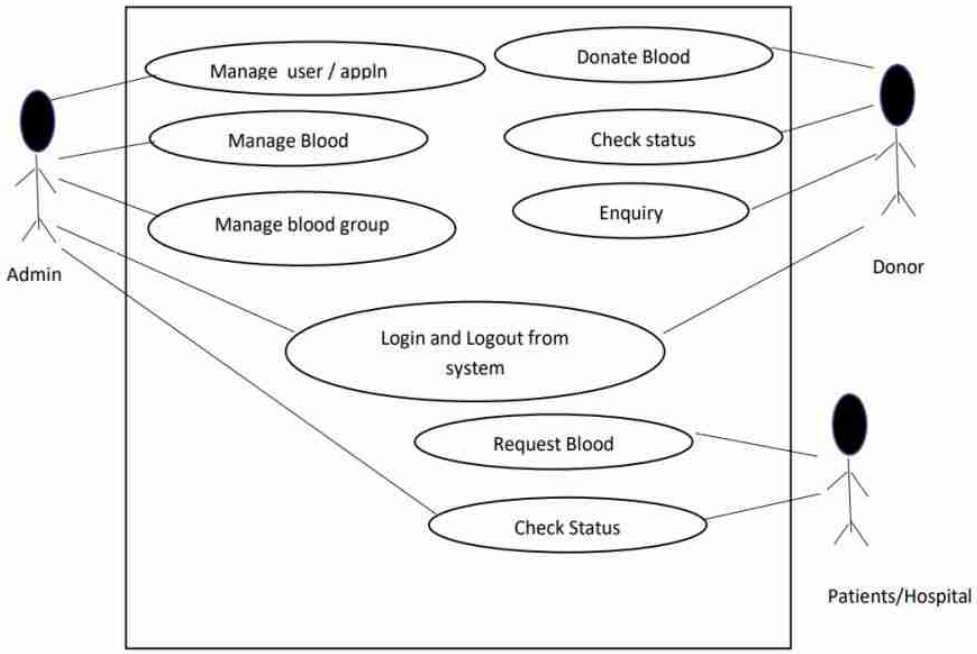
INPUTS:

Actors

Use Cases

Relations

Result:


Manage user / appln
Donate Blood
Manage Blood
Check status
Manage blood group
Enquiry
Admin
Donor
Login and Logout from system
Request Blood
Check Status
Patients/Hospital

EX NO:6	Draw activity diagram of all use cases.
DATE	

AIM:

To Draw the activity Diagram for any project

ALGORITHM:

Step 1: Identify the Initial State and Final States

Step 2: Identify the Intermediate Activities Needed

Step 3: Identify the Conditions or Constraints

Step 4: Draw the Diagram with Appropriate Notations

INPUTS:

Activities

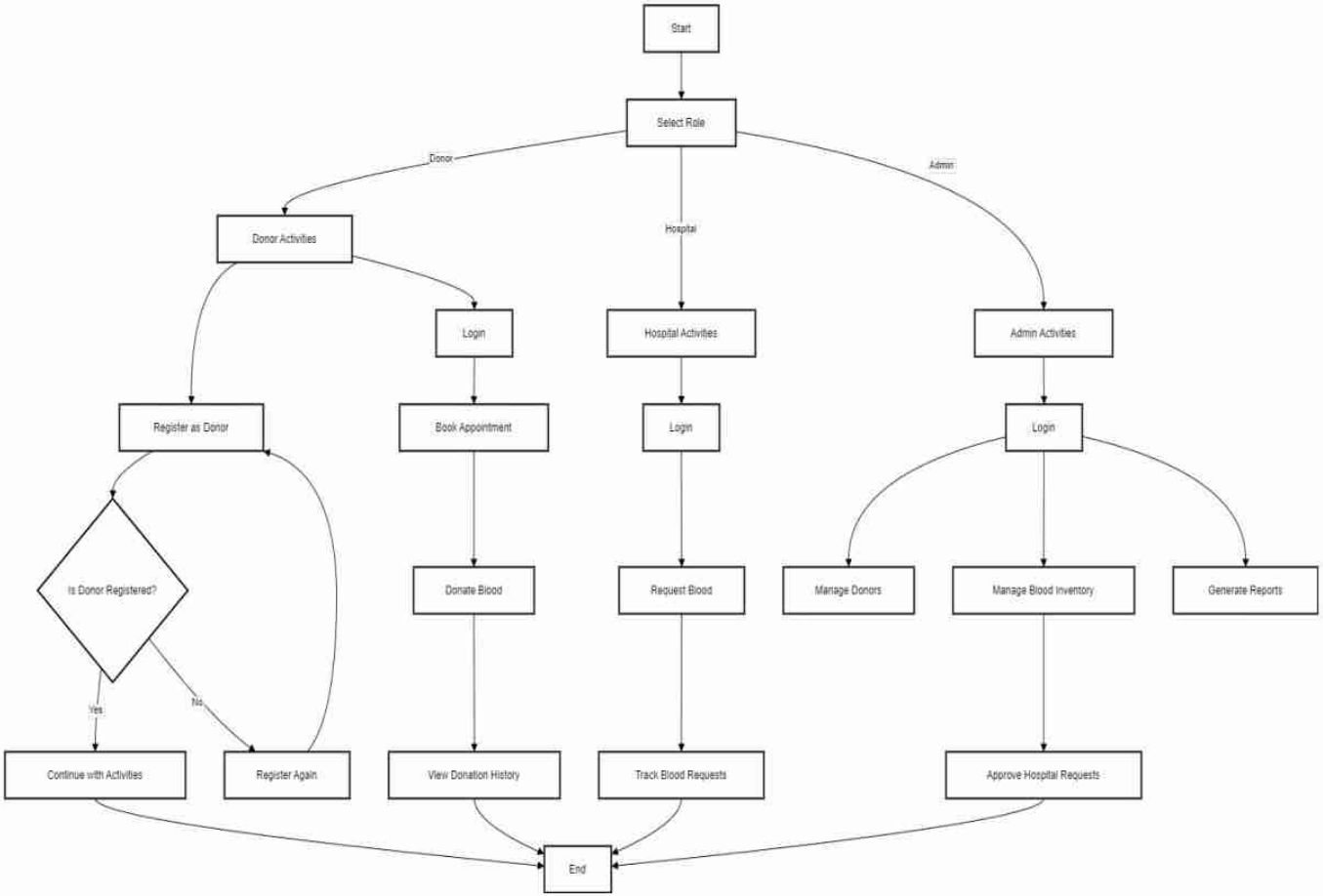
Decision Points

Guards

Parallel Activities

Conditions

Result:


Start
Select Role
Donor
Hospital
Admin
Donor Activities
Login
Hospital Activities
Admin Activities
Register as Donor
Book Appointment
Login
Login
Is Donor Registered?
Donate Blood
Request Blood
Manage Donors
Manage Blood Inventory
Generate Reports
Yes
No
Continue with Activities
Register Again
View Donation History
Track Blood Requests
Approve Hospital Requests
End

EX NO:7	Draw state chart diagram of all use cases.
DATE	

AIM:

To Draw the State Chart Diagram for any project

ALGORITHM:

STEP-1: Identify the important objects to be analysed.

STEP-2: Identify the states.

STEP-3: Identify the events.

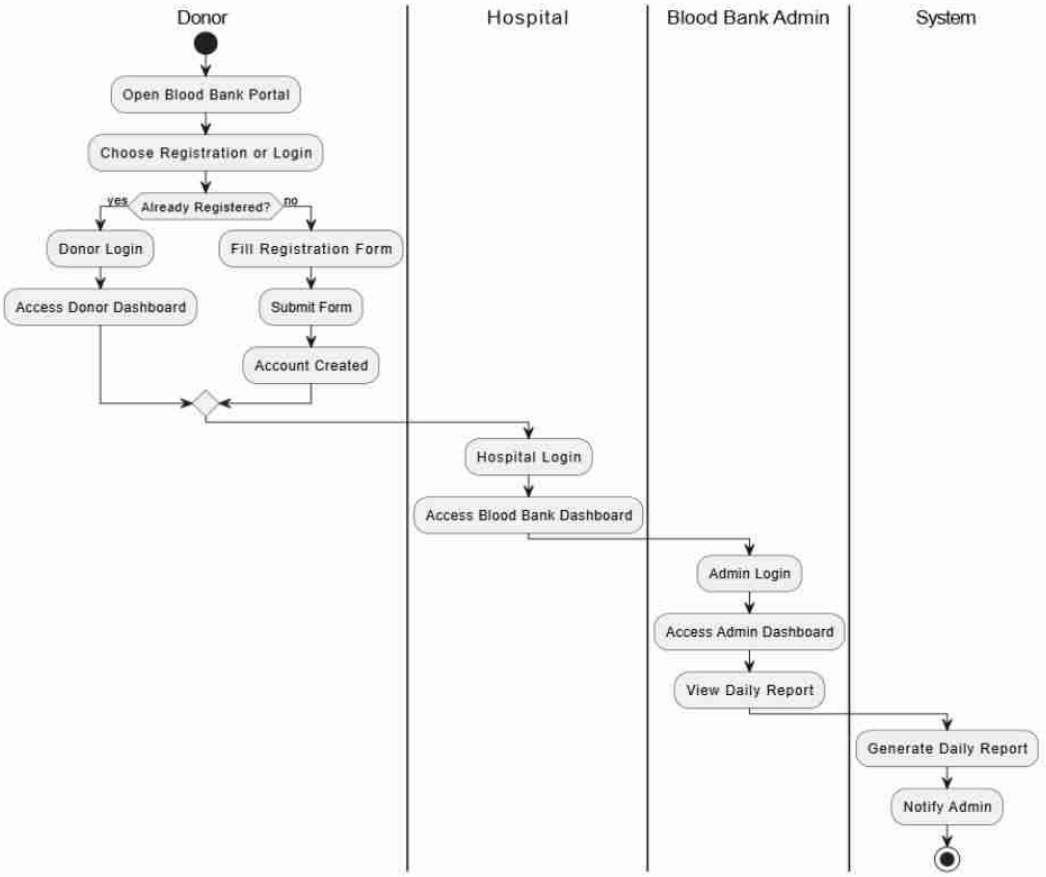
INPUTS:

Objects

States

Events

Result:


Donor
Hospital
Blood Bank Admin
System
Open Blood Bank Portal
Choose Registration or Login
Already Registered?
yes
no
Donor Login
Fill Registration Form
Access Donor Dashboard
Submit Form
Account Created
Hospital Login
Access Blood Bank Dashboard
Admin Login
Access Admin Dashboard
View Daily Report
Generate Daily Report
Notify Admin

EX NO:8	Draw sequence diagram of all use cases.
DATE	

AIM:

To Draw the Sequence Diagram for any project

ALGORITHM:

1. Identify the Scenario
2. List the Participants
3. Define Lifelines
4. Arrange Lifelines
5. Add Activation Bars
6. Draw Messages
7. Include Return Messages
8. Indicate Timing and Order
9. Include Conditions and Loops
10. Consider Parallel Execution
11. Review and Refine
12. Add Annotations and Comments
13. Document Assumptions and Constraints
14. Use a Tool to create a neat sequence diagram

INPUTS:

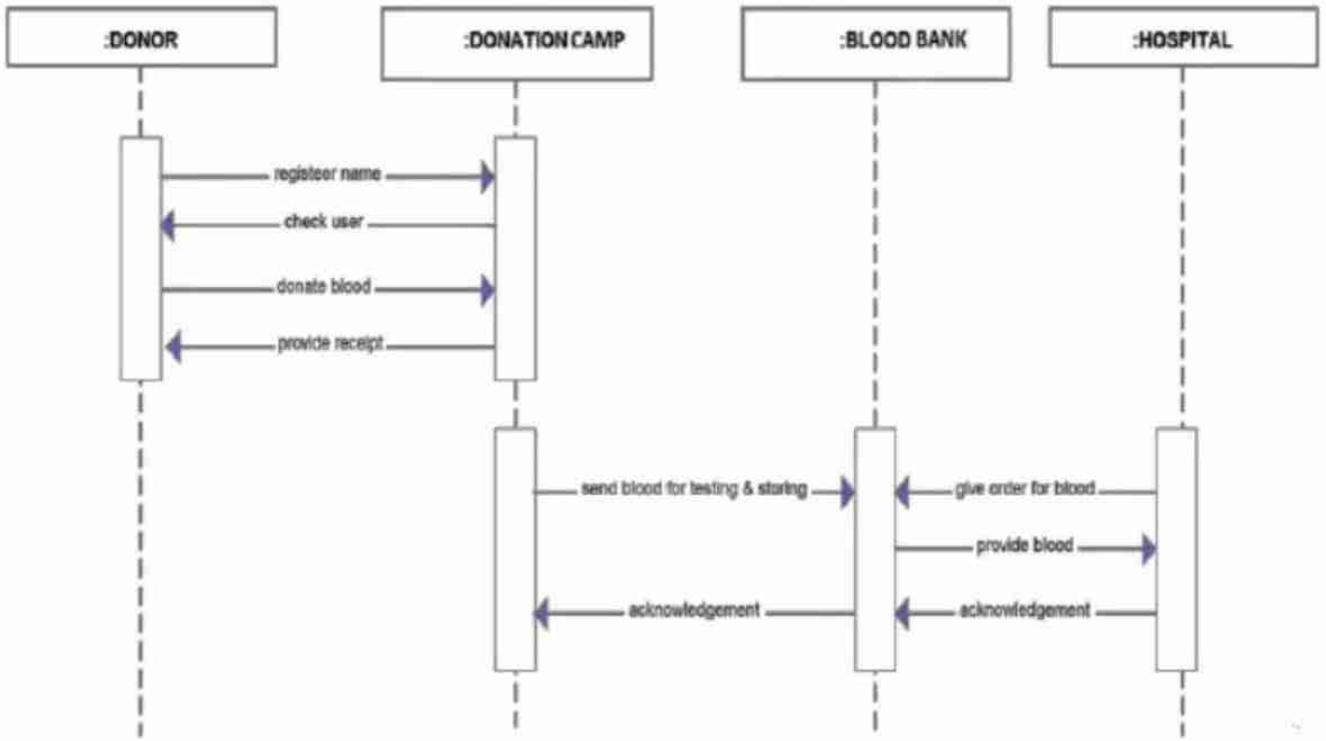
Objects taking part in the interaction.

Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:


:DONOR
:DONATION CAMP
:BLOOD BANK
:HOSPITAL
registeer name
check user
donate blood
provide receipt
send blood for testing & storing
give order for blood
provide blood
acknowledgement
acknowledgement

EX NO:9	Draw collaboration diagram of all use cases
DATE	

Draw collaboration diagram of all use cases

AIM:

To Draw the Collaboration Diagram for any project

ALGORITHM:

Step 1: Identify Objects/Participants

Step 2: Define Interactions

Step 3: Add Messages

Step 4: Consider Relationships

Step 5: Document the collaboration diagram along with any relevant explanations or annotations.

INPUTS:

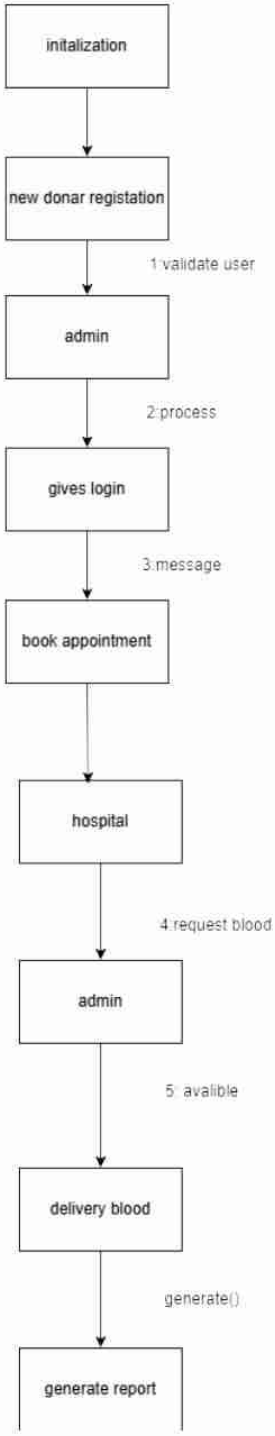
Objects taking part in the interaction.

Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:


initalization
new donar registation
1:validate user
admin
2:process
gives login
3:message
book appointment
hospital
4:request blood
admin
5: avalible
delivery blood
generate()
generate report

EX NO:10	Assign objects in sequence diagram to classes and make class diagram.
DATE	

AIM:

To Draw the Class Diagram for any project

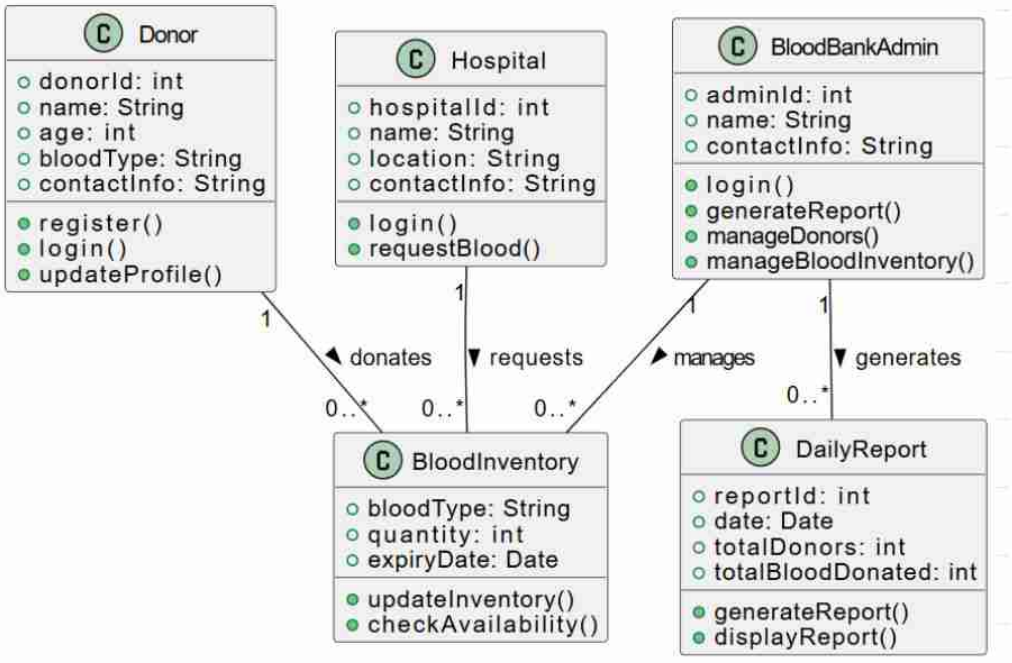
ALGORITHM:

1. Identify Classes
2. List Attributes and Methods
3. Identify Relationships
4. Create Class Boxes
5. Add Attributes and Methods
6. Draw Relationships
7. Label Relationships
8. Review and Refine
9. Use Tools for Digital Drawing

INPUTS:

1. Class Name
2. Attributes
3. Methods
4. Visibility Notation

Result:


Donor
donorId: int
name: String
age: int
bloodType: String
contactInfo: String
register()
login()
updateProfile()
Hospital
hospitalId: int
name: String
location: String
contactInfo: String
login()
requestBlood()
BloodBankAdmin
adminId: int
name: String
contactInfo: String
login()
generateReport()
manageDonors()
manageBloodInventory()
1
donates
0..*
1
requests
0..*
1
manages
0..*
1
generates
0..*
BloodInventory
bloodType: String
quantity: int
expiryDate: Date
updateInventory()
checkAvailability()
DailyReport
reportId: int
date: Date
totalDonors: int
totalBloodDonated: int
generateReport()
displayReport()

EX NO:11	Mini Project-Blood Bank Management System
DATE	

Aim:

The Blood Bank Management System aims to manage donor records, blood inventory, and hospital requests efficiently, ensuring a timely supply of blood for patients in need. It helps optimize blood donation and availability, facilitating better organization and accessibility within blood banks.

Algorithm:

1. **Donor Registration**: Collect and verify donor details, checking eligibility based on criteria like age and health.
2. **Blood Donation Entry**: Record the donated blood, updating the inventory with blood type and quantity.
3. **Inventory Management**: Monitor and update stock levels; alert admin if stock is low for any blood type.
4. **Hospital Blood Request**: Accept and verify blood requests from hospitals, checking stock availability.
5. **Appointment Scheduling**: Allow donors to book appointments, checking slot availability.
6. **Request Processing**: Approve or deny requests based on inventory, notifying hospitals of the status.
7. **Generate Reports**: Summarize data on blood stock, donors, and pending requests for administrative review.

Program:

```
import streamlit as st

import sqlite3

import pandas as pd

from datetime import datetime


# Database Setup

def create_tables():

    with sqlite3.connect("database.db") as conn:

        cursor = conn.cursor()

        cursor.execute('''CREATE TABLE IF NOT EXISTS Donor

                    (id INTEGER PRIMARY KEY, name TEXT, age INTEGER, blood_type TEXT,
contact_info TEXT, last_donation_date TEXT)''')
```

```
        cursor.execute('''CREATE TABLE IF NOT EXISTS Appointment
                          (id INTEGER PRIMARY KEY, donor_id INTEGER, doctor_id INTEGER, date TEXT,
time TEXT, status TEXT)''')
        cursor.execute('''CREATE TABLE IF NOT EXISTS Doctor
                          (id INTEGER PRIMARY KEY, name TEXT, specialization TEXT)''')
        cursor.execute('''CREATE TABLE IF NOT EXISTS BloodInventory
                          (id INTEGER PRIMARY KEY AUTOINCREMENT, blood_type TEXT UNIQUE,
quantity INTEGER)''')
        cursor.execute('''CREATE TABLE IF NOT EXISTS BloodRequest
                          (id INTEGER PRIMARY KEY, hospital_name TEXT, blood_type TEXT, quantity
INTEGER, status TEXT)''')
        cursor.execute('''CREATE TABLE IF NOT EXISTS TransactionLog
                          (id INTEGER PRIMARY KEY, transaction_type TEXT, date TEXT, details TEXT)''')
        conn.commit()

create_tables()

# Helper functions
def add_donor(name, age, blood_type, contact_info, last_donation_date):
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO Donor (name, age, blood_type, contact_info, last_donation_date)
VALUES (?, ?, ?, ?, ?)",
                       (name, age, blood_type, contact_info, last_donation_date))
        conn.commit()

def get_donors():
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Donor")
```

```
        return cursor.fetchall()


def add_doctor(name, specialization):
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO Doctor (name, specialization) VALUES (?, ?)",
                       (name, specialization))
        conn.commit()


def get_doctors():
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Doctor")
        return cursor.fetchall()


def book_appointment(donor_id, doctor_id, date, time):
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO Appointment (donor_id, doctor_id, date, time, status) VALUES (?,
?, ?, ?, 'Pending')",
                       (donor_id, doctor_id, date, time))
        conn.commit()


def get_appointments(donor_id):
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Appointment WHERE donor_id = ?", (donor_id,))
        return cursor.fetchall()
```

```
def manage_inventory(blood_type, quantity):
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT quantity FROM BloodInventory WHERE blood_type = ?",
(blood_type,))
        result = cursor.fetchone()
        if result:
            cursor.execute("UPDATE BloodInventory SET quantity = quantity + ? WHERE blood_type =
?", (quantity, blood_type))
        else:
            cursor.execute("INSERT INTO BloodInventory (blood_type, quantity) VALUES (?, ?)",
(blood_type, quantity))
        conn.commit()

def get_inventory():
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM BloodInventory")
        return cursor.fetchall()

def add_blood_request(hospital_name, blood_type, quantity):
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO BloodRequest (hospital_name, blood_type, quantity, status)
VALUES (?, ?, ?, 'Pending')",
                       (hospital_name, blood_type, quantity))
        conn.commit()

def get_blood_requests():
```

```
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM BloodRequest WHERE status = 'Pending'")
        return cursor.fetchall()

def generate_report():
    with sqlite3.connect("database.db") as conn:
        cursor = conn.cursor()

        # Number of donors
        cursor.execute("SELECT COUNT(*) FROM Donor")
        total_donors = cursor.fetchone()[0]

        # Number of donations
        cursor.execute("SELECT COUNT(*) FROM Appointment WHERE status = 'Completed'")
        total_donations = cursor.fetchone()[0]

        # Requests sent by hospitals
        cursor.execute("SELECT COUNT(*) FROM BloodRequest")
        total_requests = cursor.fetchone()[0]

        # Approved requests
        cursor.execute("SELECT COUNT(*) FROM BloodRequest WHERE status = 'Approved'")
        approved_requests = cursor.fetchone()[0]

        # Data summary
        report_data = {
            "Total Donors Registered": total_donors,
```

```
            "Total Donations Completed": total_donations,
            "Total Requests by Hospitals": total_requests,
            "Approved Requests": approved_requests,
        }
        return report_data

# Main App
def main():
    st.title("Blood Bank Management System")

    # User Role Selection
    role = st.sidebar.selectbox("Choose Role", ["Donor", "Hospital Admin", "Blood Bank Admin"], key="role")

    if role == "Donor":
        st.header("Donor Interface")
        donor_menu = st.selectbox("Choose an option", ["Register", "View Details", "Book Appointment", "View Appointments"], key="donor_menu")

        if donor_menu == "Register":
            st.subheader("Register as a Donor")
            name = st.text_input("Name")
            age = st.number_input("Age", min_value=18, max_value=65)
            blood_type = st.selectbox("Blood Type", ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"], key="blood_type")
            contact_info = st.text_input("Contact Information")
            last_donation_date = st.date_input("Last Donation Date")

            if st.button("Register"):
```

```
            add_donor(name, age, blood_type, contact_info, last_donation_date.strftime('%Y-%m-%d'))

            st.success("Donor registered successfully!")


            # Display donor ID and details

            donors = get_donors()

            donor_id = donors[-1][0]

            st.write(f"Your Donor ID: {donor_id}")

            st.write(f"Name: {name}, Age: {age}, Blood Type: {blood_type}, Contact: {contact_info},
Last Donation: {last_donation_date}")


        elif donor_menu == "View Details":

            st.subheader("View Donor Details")

            donors = get_donors()

            for donor in donors:

                st.write(f"Name: {donor[1]}, Age: {donor[2]}, Blood Type: {donor[3]}, Contact: {donor[4]},
Last Donation: {donor[5]}")


        elif donor_menu == "Book Appointment":

            st.subheader("Book an Appointment")

            donor_id = st.number_input("Enter your Donor ID", min_value=1, step=1)

            date = st.date_input("Preferred Date")

            time = st.time_input("Preferred Time")


            # Fetch available doctors by name and specialization

            doctors = get_doctors()

            doctor_choices = {f"{doc[1]} ({doc[2]})": doc[0] for doc in doctors}  # {name (specialization):
id}


            if doctors:
```

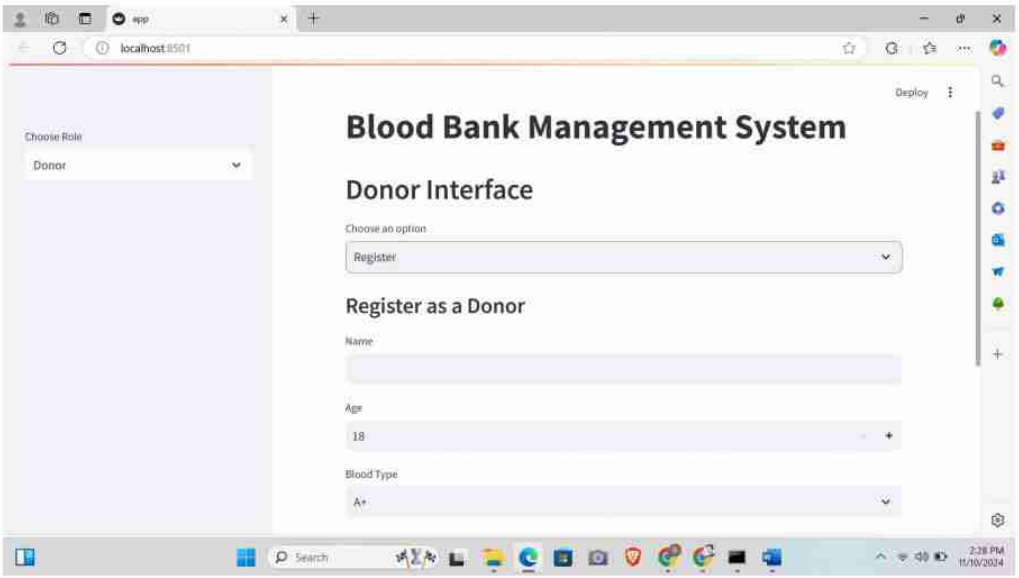
```
            doctor_id = st.selectbox("Select Doctor", list(doctor_choices.keys()))

            if st.button("Book Appointment"):

                book_appointment(donor_id, doctor_choices[doctor_id], date.strftime('%Y-%m-%d'),
time.strftime('%H:%M'))

                st.success("Appointment booked successfully! Awaiting confirmation.")

        else:

            st.write("No doctors available to select.")


    elif donor_menu == "View Appointments":

        st.subheader("View Appointments")

        donor_id = st.number_input("Enter your Donor ID", min_value=1, step=1)

        appointments = get_appointments(donor_id)

        if appointments:

            for appointment in appointments:

                st.write(f"Appointment ID: {appointment[0]}")

                st.write(f"Donor Name: {appointment[1]}")

                st.write(f"Doctor: {appointment[2]} ({appointment[3]})")

                st.write(f"Date: {appointment[4]}, Time: {appointment[5]}")

                st.write(f"Status: {appointment[6]}")

                st.write("---")

        else:

            st.write("No appointments found for this donor.")


elif role == "Hospital Admin":

    st.header("Hospital Admin Interface")

    hospital_menu = st.selectbox("Choose an option", ["Request Blood", "View Request Status"],
key="hospital_menu")


    if hospital_menu == "Request Blood":
```

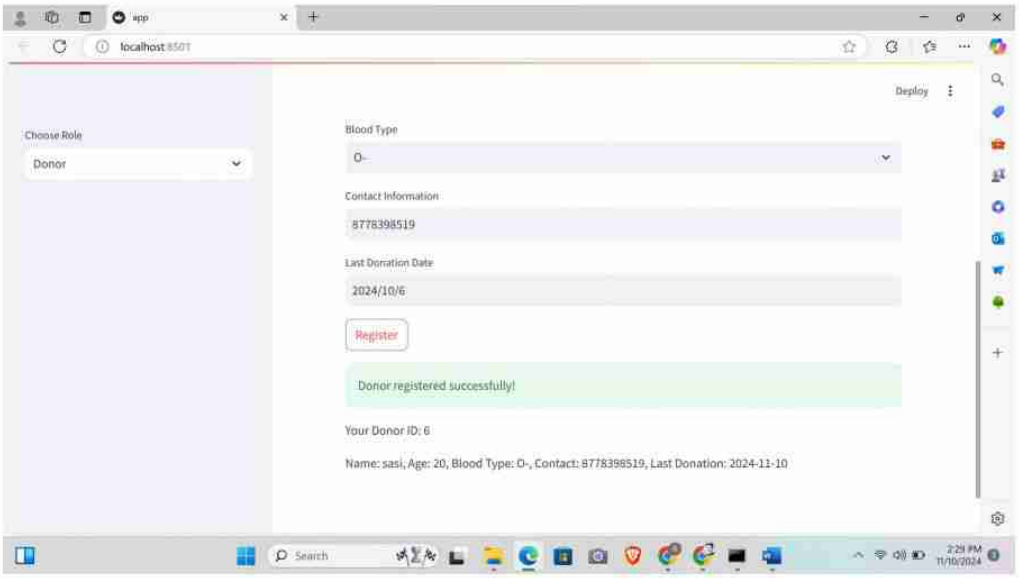
```
            st.subheader("Request Blood")

            hospital_name = st.text_input("Hospital Name")

            blood_type = st.selectbox("Blood Type", ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"],
key="hospital_blood_type")

            quantity = st.number_input("Quantity", min_value=1)



            if st.button("Request Blood"):

                add_blood_request(hospital_name, blood_type, quantity)

                st.success(f"Blood request for {blood_type} ({quantity} units) successfully made.")



        elif hospital_menu == "View Request Status":

            st.subheader("View Request Status")

            blood_requests = get_blood_requests()

            for request in blood_requests:

                st.write(f"Request ID: {request[0]}")

                st.write(f"Hospital: {request[1]}")

                st.write(f"Blood Type: {request[2]}")

                st.write(f"Quantity: {request[3]}")

                st.write(f"Status: {request[4]}")

                st.write("---")



    elif role == "Blood Bank Admin":

        st.header("Blood Bank Admin Interface")

        admin_menu = st.selectbox("Choose an option", ["Manage Inventory", "Generate Report"],
key="admin_menu")



        if admin_menu == "Manage Inventory":

            st.subheader("Manage Blood Inventory")
```

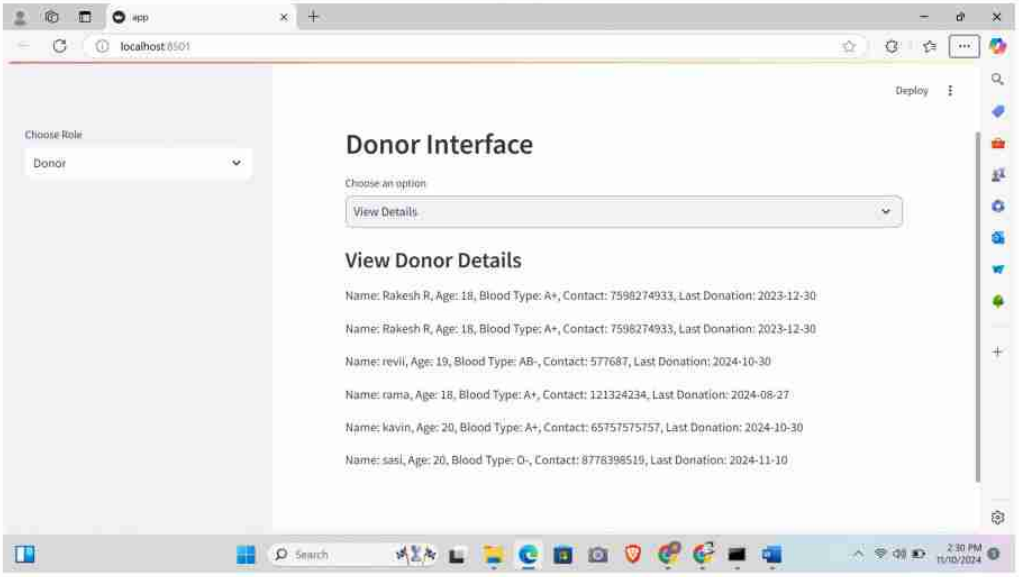
```
            blood_type = st.selectbox("Blood Type", ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"],
    key="admin_blood_type")

            quantity = st.number_input("Quantity", min_value=1)



            if st.button("Update Inventory"):

                manage_inventory(blood_type, quantity)

                st.success(f"Blood inventory for {blood_type} updated by {quantity} units.")



        elif admin_menu == "Generate Report":

            st.subheader("Generate Report")

            report_data = generate_report()

            for key, value in report_data.items():

                st.write(f"{key}: {value}")



    if __name__ == "__main__":

        main()
```

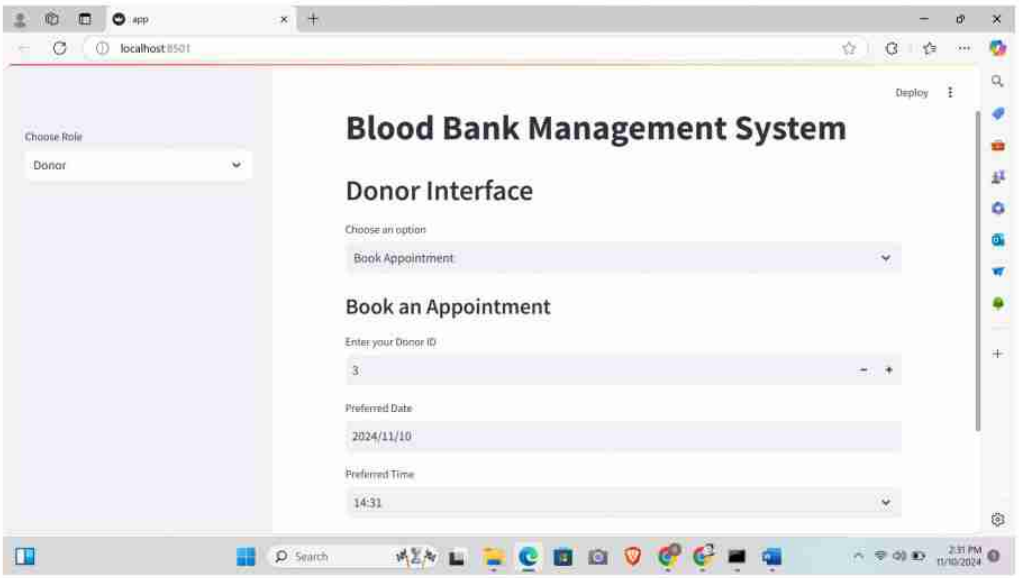
Conclusion

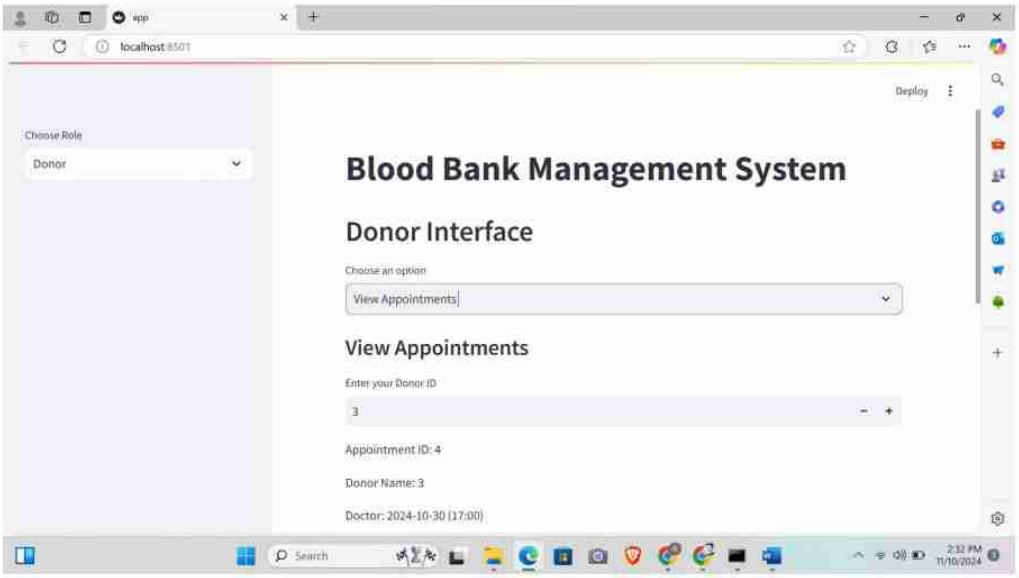
The Blood Bank Management System developed using Streamlit and SQLite offers a user-friendly interface for donors, hospital admins, and blood bank administrators to effectively manage blood donation processes, track inventory, and handle blood requests. This system enhances the efficiency of blood donation management by centralizing key functionalities and providing real-time updates on donor registrations, blood availability, and request statuses.

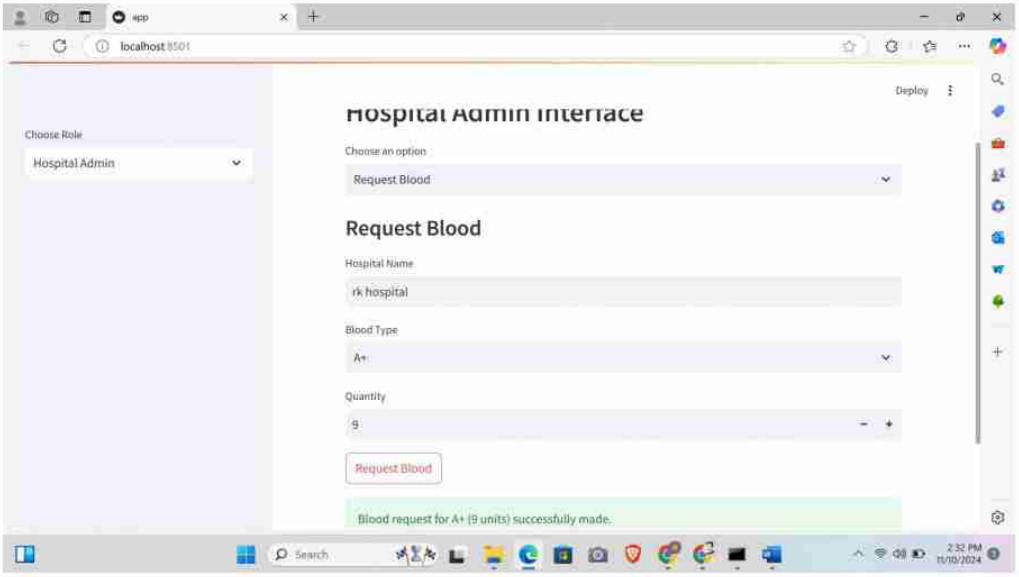

app
localhost:8501
Deploy
Choose Role
Donor
Blood Bank Management System
Donor Interface
Choose an option
Register
Register as a Donor
Name
Age
18
Blood Type
A+
Search
2:28 PM
11/10/2024

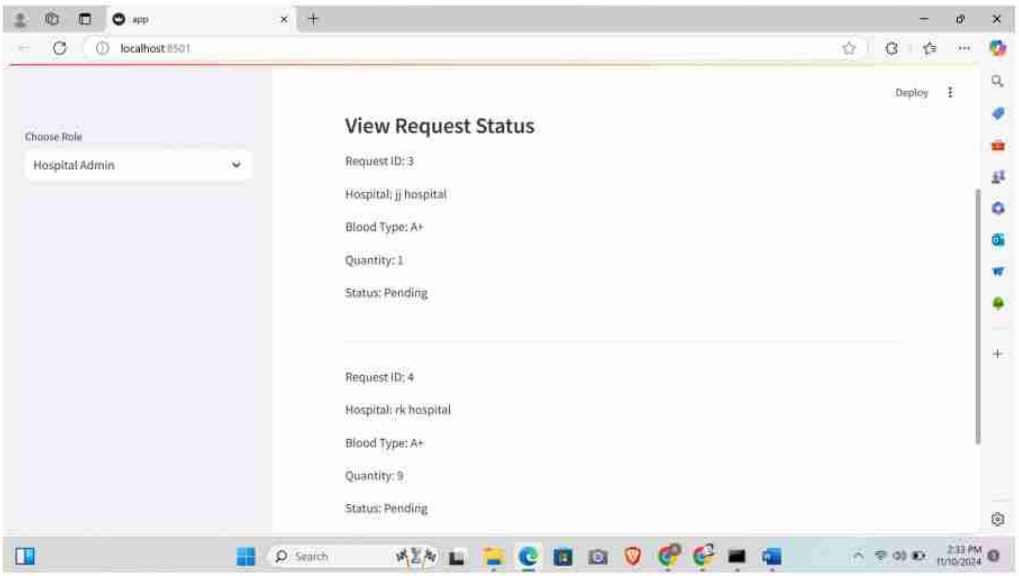

app
localhost:8501
Deploy
Choose Role
Donor
Blood Type
O-
Contact Information
8778398519
Last Donation Date
2024/10/6
Register
Donor registered successfully!
Your Donor ID: 6
Name: sasi, Age: 20, Blood Type: O-, Contact: 8778398519, Last Donation: 2024-11-10
Search
2:29 PM
11/10/2024

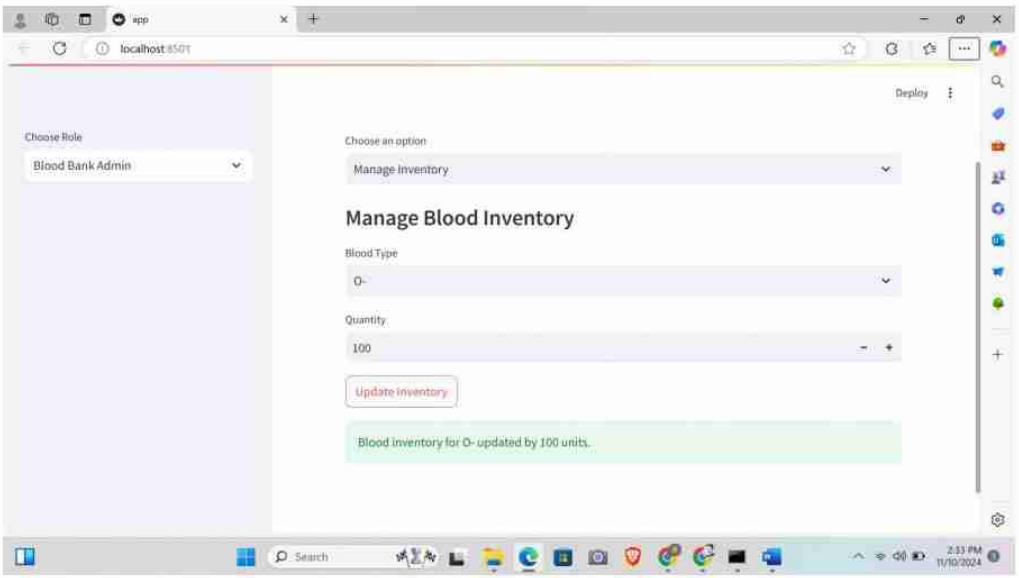

app
localhost:8501
Deploy
Choose Role
Donor
Donor Interface
Choose an option
View Details
View Donor Details
Name: Rakesh R, Age: 18, Blood Type: A+, Contact: 7598274933, Last Donation: 2023-12-30
Name: Rakesh R, Age: 18, Blood Type: A+, Contact: 7598274933, Last Donation: 2023-12-30
Name: revii, Age: 19, Blood Type: AB-, Contact: 577687, Last Donation: 2024-10-30
Name: rama, Age: 18, Blood Type: A+, Contact: 121324234, Last Donation: 2024-08-27
Name: kavin, Age: 20, Blood Type: A+, Contact: 65757575757, Last Donation: 2024-10-30
Name: sasi, Age: 20, Blood Type: O-, Contact: 8778398519, Last Donation: 2024-11-10
Search
2:30 PM
11/10/2024

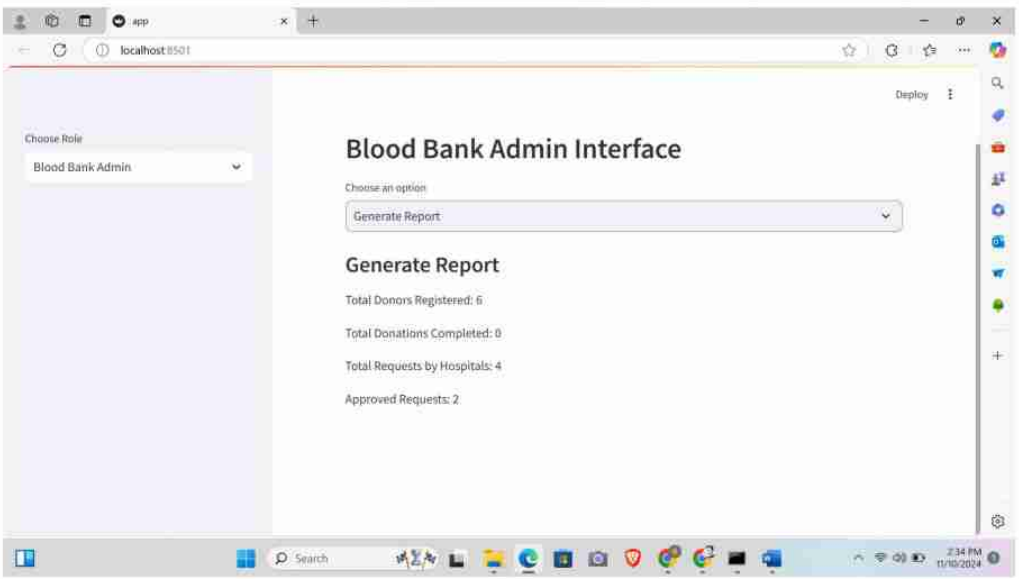

app
localhost:8501
Deploy
Choose Role
Donor
Blood Bank Management System
Donor Interface
Choose an option
Book Appointment
Book an Appointment
Enter your Donor ID
3
Preferred Date
2024/11/10
Preferred Time
14:31
Search
2:31 PM
11/10/2024


app
localhost:8501
Deploy
Choose Role
Donor
Blood Bank Management System
Donor Interface
Choose an option
View Appointments
View Appointments
Enter your Donor ID
3
Appointment ID: 4
Donor Name: 3
Doctor: 2024-10-30 (17:00)
Search
2:32 PM
11/10/2024


app
localhost:8501
Deploy
Choose Role
Hospital Admin
Hospital Admin Interface
Choose an option
Request Blood
Request Blood
Hospital Name
rk hospital
Blood Type
A+
Quantity
9
Request Blood
Blood request for A+ (9 units) successfully made.
Search
2:32 PM
11/10/2024


app
localhost:8501
Deploy
Choose Role
Hospital Admin
View Request Status
Request ID: 3
Hospital: jj hospital
Blood Type: A+
Quantity: 1
Status: Pending
Request ID: 4
Hospital: rk hospital
Blood Type: A+
Quantity: 9
Status: Pending
Search
2:33 PM
11/10/2024


app
localhost:8501
Deploy
Choose Role
Blood Bank Admin
Choose an option
Manage Inventory
Manage Blood Inventory
Blood Type
O-
Quantity
100
Update Inventory
Blood inventory for O- updated by 100 units.
Search
2:33 PM
11/10/2024


app
localhost:8501
Deploy
Choose Role
Blood Bank Admin
Blood Bank Admin Interface
Choose an option
Generate Report
Generate Report
Total Donors Registered: 6
Total Donations Completed: 0
Total Requests by Hospitals: 4
Approved Requests: 2
Search
2:34 PM
11/10/2024