

# SOFTWARE ENGINEERING

## BLOOD BANK MANAGEMENT SYSTEM



**SRM**  
UNIVERSITY *AP*  
— Andhra Pradesh

**PUDI JAHAVI - AP17110010060**

**MAKKENA ALEKHYA - AP17110010112**

**POPURI SATHVIKA - AP17110010108**

**A JOSHITHA - AP17110010144**

---

## SRS on BloodBank Management System

### Objective

The main objective of this specification is to support the automated tracking of blood products from the initial ordering of a blood transfusion for a patient, through to the taking of a blood sample for cross matching, to administration of a blood transfusion and subsequent updates to care records.

To allow the probable recipients to make search and match the volunteer donors, and make a request for the blood.

### Introduction

Blood sales and blood purchase are entered and maintained in this project. Blood stock reports, sales reports and blood purchase reports are managed in this project. It will help us to find the blood group with the most efficient time to take care of the blood and it is more easy to hand over the blood to the hospitals to help people to get blood on time.

This all thing has been stored and been seen in this Blood Bank Management System. To help more people trying best to do so.

### Purpose

Blood Bank Management Software is designed & suitable for several Blood Banks either operating as individual organizations or part of Hospitals. It covers all Blood banking process from Donor recruitment, donor management, mobile sessions, component preparation, screening covering all tests, blood stock inventory maintenance, patient registration, cross matching, patient issues

### Intended User

Anybody can use this BBMS to Donor as well as who need blood e.g., PublicHospitals, Blood Banks, etc.

### Project Scope

The scope of the specification includes the following scenarios:

- Routine blood transfusion;  
Transfusion for special requirements (for example, cytomegalovirus (CMV) seronegative blood, irradiated blood or antigen negative blood);
- Emergency issue of blood;
- Management of returned and unused blood units.

## Limitations

- End User's will not be able to get the information about the availability of the blood in the bank of which he/she donated.
- Only the Admin has all right to edit the things in the End User's Profile.

## Overall Description

### System Features

#### Functional Requirements

- Login  
The system provides security features through username-password matching where only authorized users can access the system with different authorization levels.  
Admin  
Input:-Username, Password  
Output: - Invalid or Update Blood Details, logout
- Donor Profile Registration  
This allows the healthy public to register as volunteer donors.  
Input:- Donor/ Recipient Id, Name, Date of Birth, Sex, Blood Group, Address, Contact Number, Email Address, Diseases (if any),Aadhar Card No.  
Output: - Successfully Registered.
- Blood Stock Management  
The blood bank staffs can manage the blood stock starting from the blood collection, to blood screening, processing, storage, transference and transfusion through this system. Each process or work-flow can be traced from the database. The system will also raise alert to the staff whenever the blood quantity is below

its par level or when the blood in stock has expired.

#### Donor/Recipient Management

The records of all donors/recipient and their history are kept in one centralized database and thus reducing duplicate data in the database. The record donation is maintained by the system.

Input:-Blood Type

Output:-No. of Blood Bottle Available=

- Reporting

The system is able to generate pre-defined reports such as the list of donors, recipients, staff, the blood quantity in the bank and charts.

Input:-Admin Username, Admin Password

Output:-Today's Report, Month Report, Year Report

## Non Functional Requirements

### Availability

- The system should be available at all times, meaning the user can access it using an application.
- In case of a of a hardware failure or database corruption, a replacement page will be shown. Also in case of a hardware failure or database corruption, backups of the database should be retrieved from the application data folder and saved by the administrator.
- It means 24 x 7 availability.

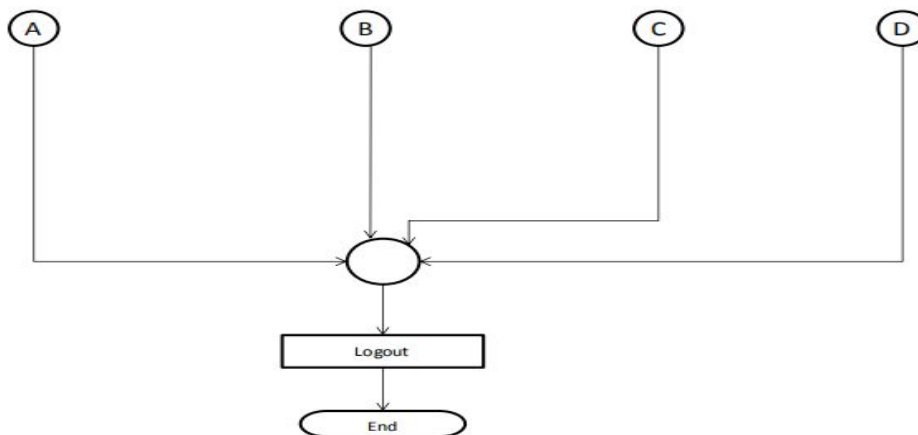
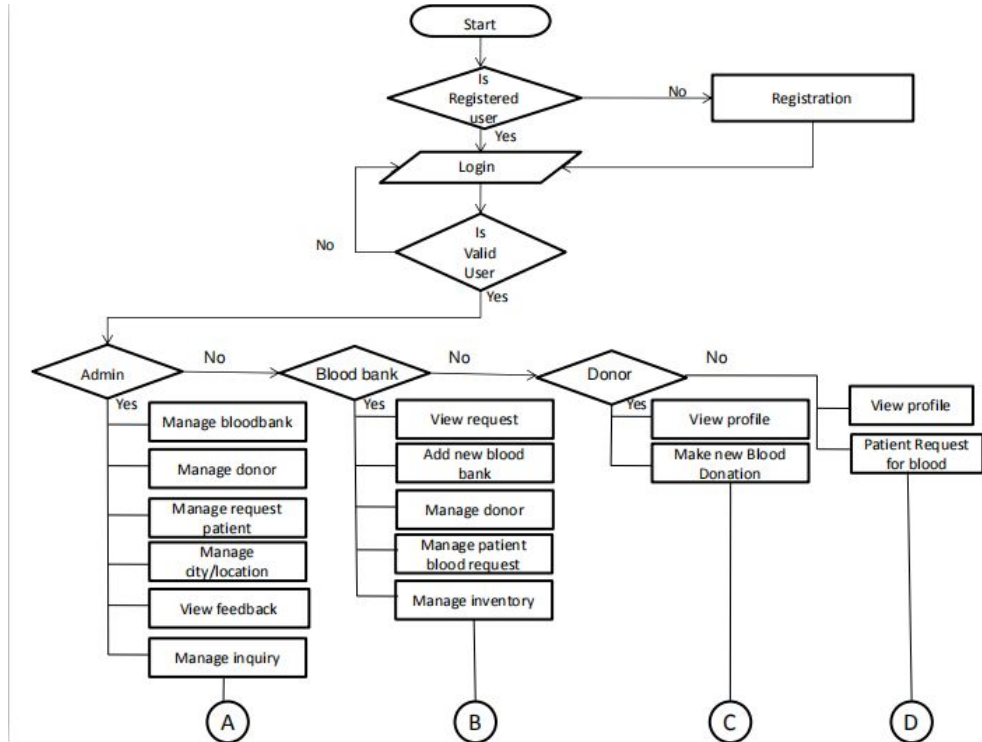
### Security

- The system is interactive and the delays involved are less.
- When connecting to the server the delay is based editing on the distance of the 2 systems and the configuration between them so there is a high probability that there will be or not a successful connection in less than 20 seconds for sake of good communication.

### Reliability

- As the system provide the right tools for problem solving it is made in such a way that the system is reliable in its operations and for securing the sensitive details.

## FLOW CHART



## DATA FLOW DIAGRAM

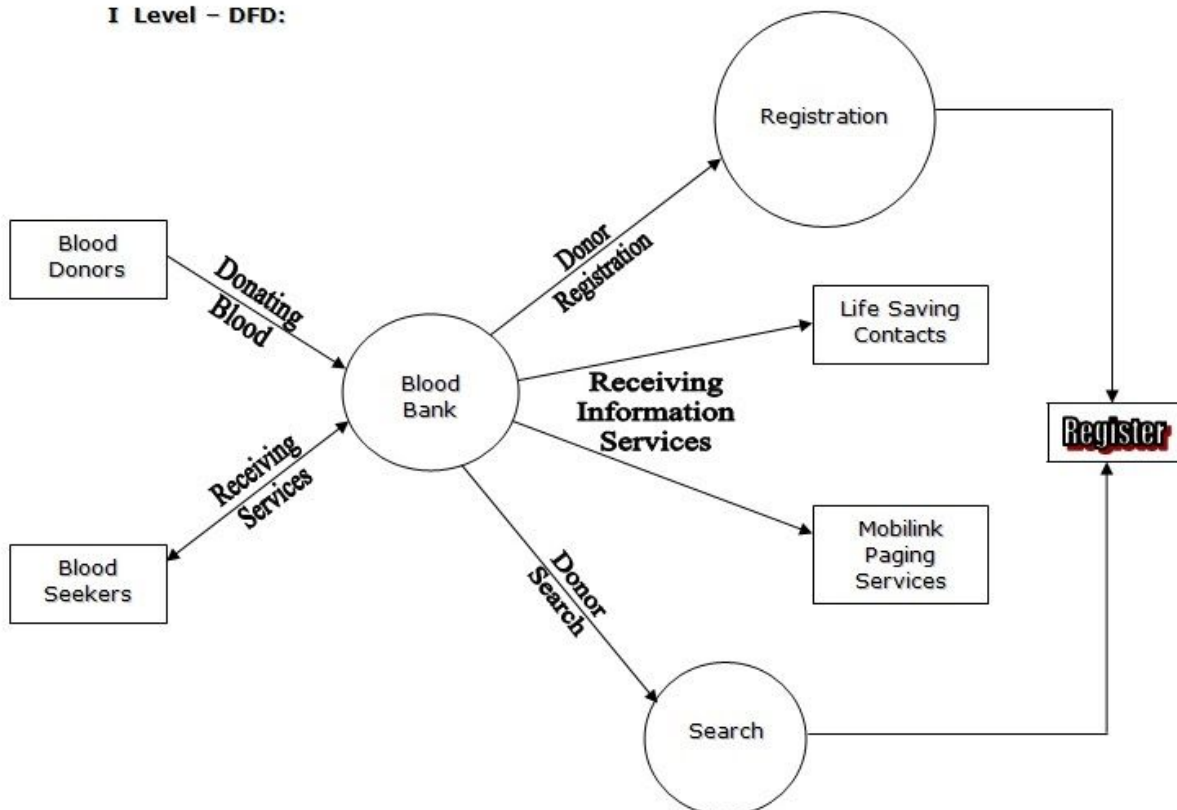
### LEVEL0

Context Level - DFD:

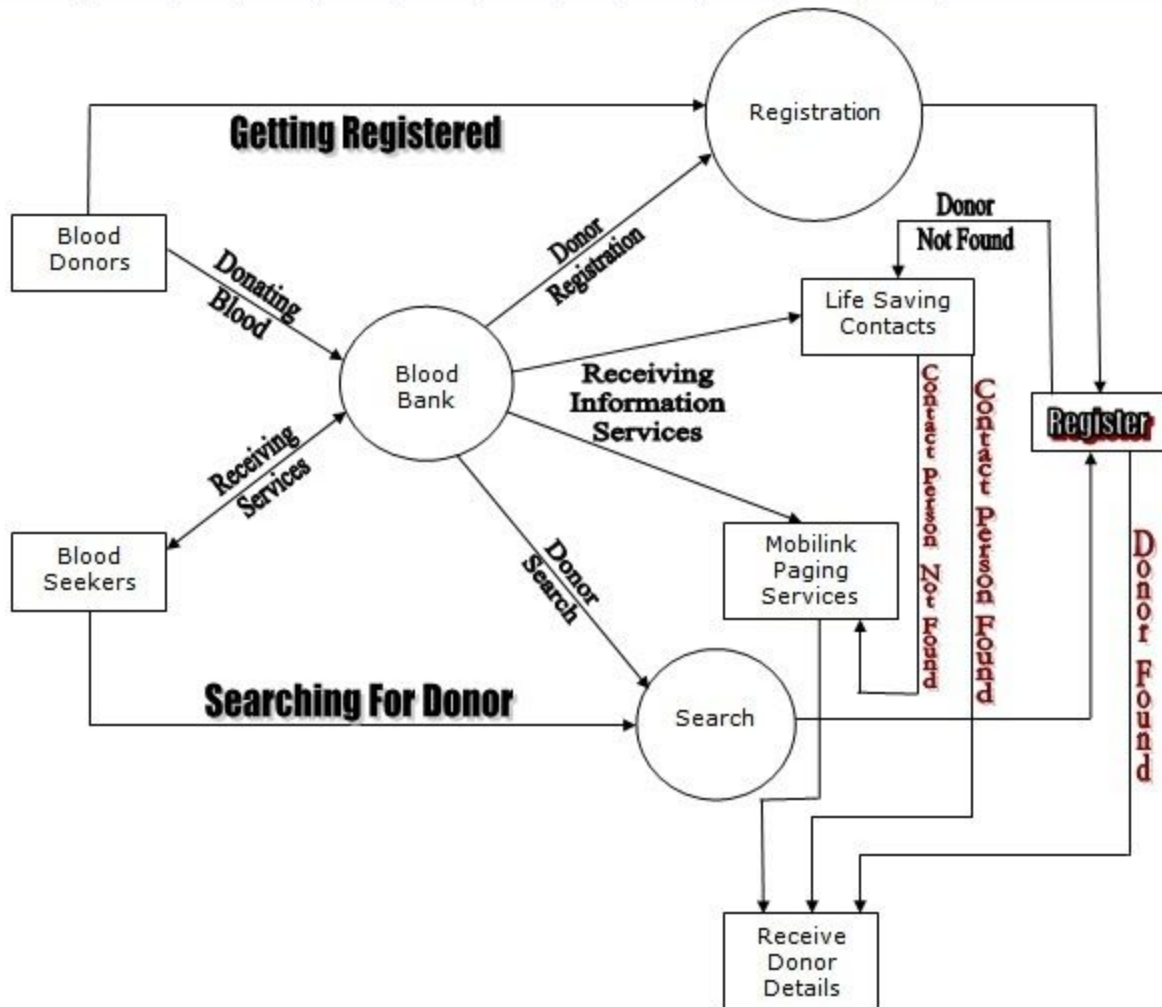


### LEVEL 1

I Level - DFD:



## LEVEL 2



## DATA DICTIONARY

### Table Name: Usertype\_mst

**Primary key:-**u\_id

**Description:-** This Table is store the user information

Field Name	Data Type	Size	Constraints	Description
U_id	bigint	4	Primary key	To store the user id
U_type	nvarchar	50	NOT NULL	To store the user type

### Table Name: State\_mst

**Primary key:-**state id

**Description:-** This Table is store the states information

Field Name	Data Type	Size	Constraints	Description
State_id	bigint	4	Primary key	To store the state id
State name	nvarchar	50	NOT NULL	To store the state name



**Table Name: City\_mst**

**Primary key:-**city id

**Foreign key:-**state\_id

**Description:-** This Table is store cities information with states wise

Field Name	Data Type	Size	Constraints	Description
city_id	bigint	4	Primary key	To store the city_id
state_id	bigint	4	Foreign key	References of the state id from state_mst
city_name	nvarchar	15	NOT NULL	Reference of the city name form city_mst

**Table Name: Location\_mst**

**Primary key:-**location id

**Foreign key:-**city\_id

**Description:-** This Table is store location information cities wise

Field Name	Data Type	Size	Constraints	Description
location_id	bigint	4	Primary key	To store the location id
City_id	bigint	4	Foreign key	Reference fo the city id from city_mst
location_name	nvarchar	50	NOT NULL	To store the location name

**Table Name: Registration**

**Primary key:-**R\_id

**Foreign key:-**u\_id

**Description:-** This Table is store user registration information

Field Name	Data Type	Size	Constraint	Description
Reg_id	Numeric	4	Primary key	To store the reg_id
U_id	Numeric	10	Foreign key	References of the User_id from usertype_mst
Name	Nvarchar	50	NOT NULL	To store the name
Cont_no	nvarchar	10	NOT NULL	To store the cont no
Address	nvarchar	50	NOT NULL	To store the address
State	nvarchar	15	NOT NULL	To store the state
City	nvarchar	15	NOT NULL	To store the city
Location	nvarchar	15	NOT NULL	To store the location
Pin-code	nvarchar	10	NOT NULL	To store the pin code
Email	nvarchar	30	NOT NULL	To store the email

Birth date	datetime		NOT NULL	To store the birthdate
Gender	nvarchar	6	NOT NULL	To store the gender
User_name	nvarchar	20	Unique key	To store the user name
Password	nvarchar	20	NOT NULL	To store the password
Security_que	nvarchar	30	NOT NULL	To store the security que
Answer	nvarchar	15	NOT NULL	To store the answer
Flag	Bit	1	NOT NULL	To store the flag

**Table Name: Bloodbank\_mst**

**Primary key:-**b\_id

**Foreign key:-**Reg\_id,loc\_id,city\_id

**Description:-** This Table is store blood bank information location and cities wise

Field Name	Data Type	Size	Constraint	Description
b_id	bigint	4	Primary Key	To store the blood bank _id
reg_id	bigint	4	Foreign key	References of the registration_id from registration
Bb_name	nvarchar	10	NOT NULL	To store the blood bank name
Loc_id	bigint	4	Foreign key	References the location id from location_mst
City id	bigint	4	Foreign key	References of the city id from city_mst
Contact no	Numeric	12	NOT NULL	To store the contact no
Status	bit	1	NOT NULL	To store the status
ddate	datetime		NOT NULL	To store the donated date

**Table Name: Donation\_mst**

**Primary key:-**d\_id

**Foreign key:-**Reg\_id,bb\_id

**Description:-** This Table is store donor information for blood bank wise

Field Name	Data Type	Size	Constraint	Description
D_id	bigint	4	Primary Key	To store the donation id
Reg_id	bigint	4	Foreign key	References of the registration id from registration
Name	nvarchar	10	NOT NULL	To store the name
b_id	bigint	4	Foreign key	References of the blood bank id from bloodbank_mst
bgroup	nvarchar	10	NOT NULL	To store the blood group
Qty	nvarchar	20	NOT NULL	To store the qty
ddate	datetime		NOT NULL	To store donated date

**Table Name: inquiry\_Form**

**Primary key:-** inq\_id

**Description:-** This Table to store inquiry information submitted for user and visitor

Field Name	Data Type	Size	Constraint	Description
Inq_Id	bigint	4	Primary key	To store the id
Name	nvarchar	20	NOT NULL	To store the name
Inquiry	nvarchar	Max	NOT NULL	To store the inquiry
Address	nvarchar	50	NOT NULL	To store the address
Phone no	nvarchar	20	NOT NULL	To store the phone no
Email	nvarchar	30	NOT NULL	To store the email
Date	Datetime		NOT NULL	To store the date
Flage	bit	1	NOT NULL	To store the flage

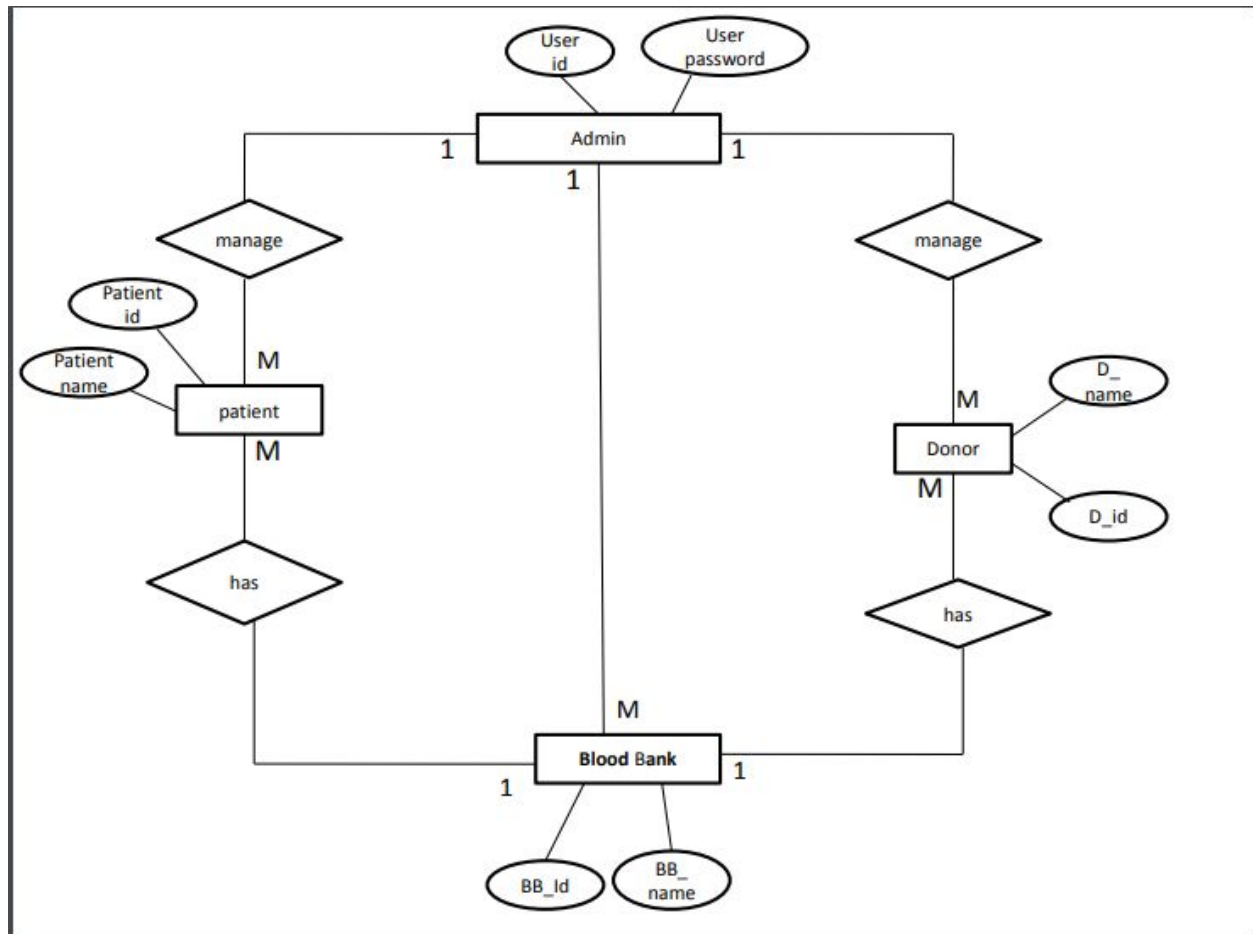
**Table Name: Feedback**

**Primary key:-** fid

**Description:-** This Table to store feedback information

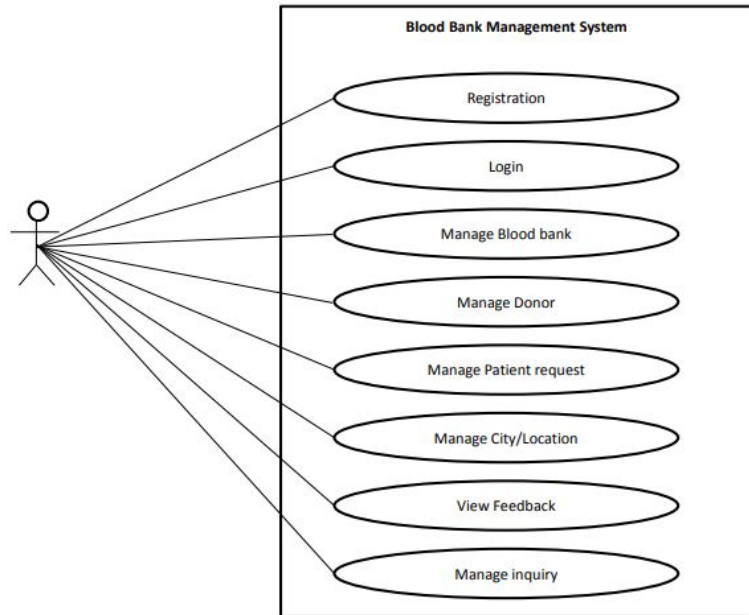
Field Name	Data Type	Size	Constraints	Description
fid	bigint	4	Primary key	To store the id
Name	nvarchar	50	NOT NULL	To store the name
Email	nvarchar	50	NOT NULL	To store the email
Feed back	nvarchar	max	NOT NULL	To store the feed back

## E-R DIAGRAM

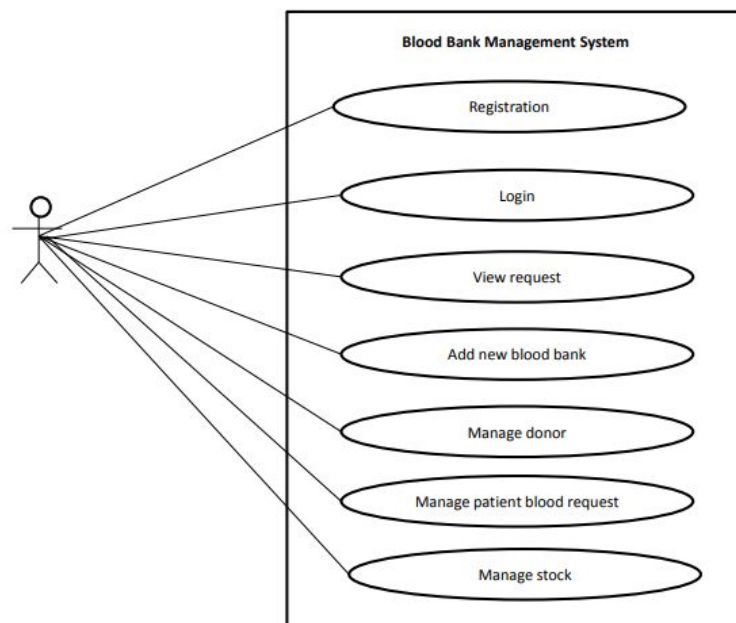


## USE CASE

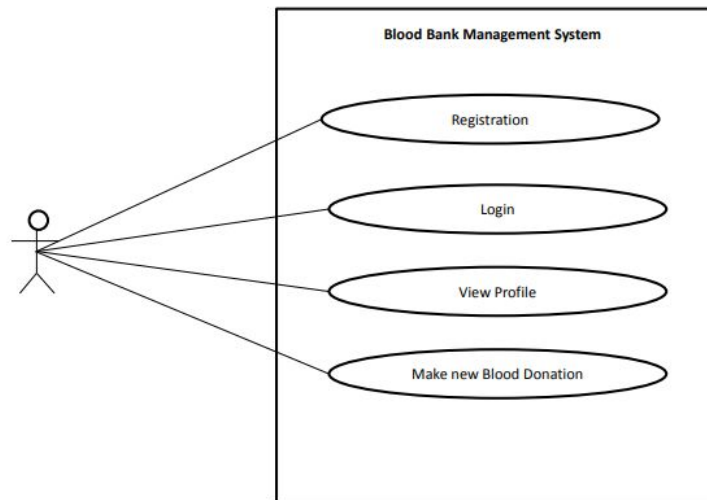
**Admin:**



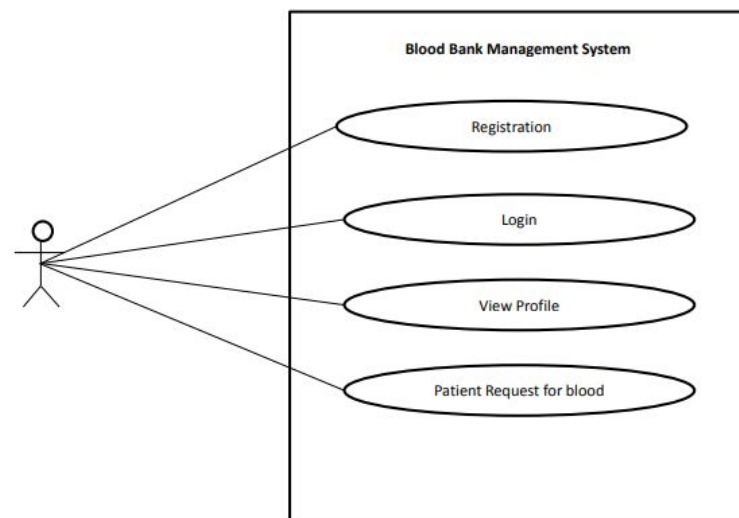
**Blood bank:**



Donor:



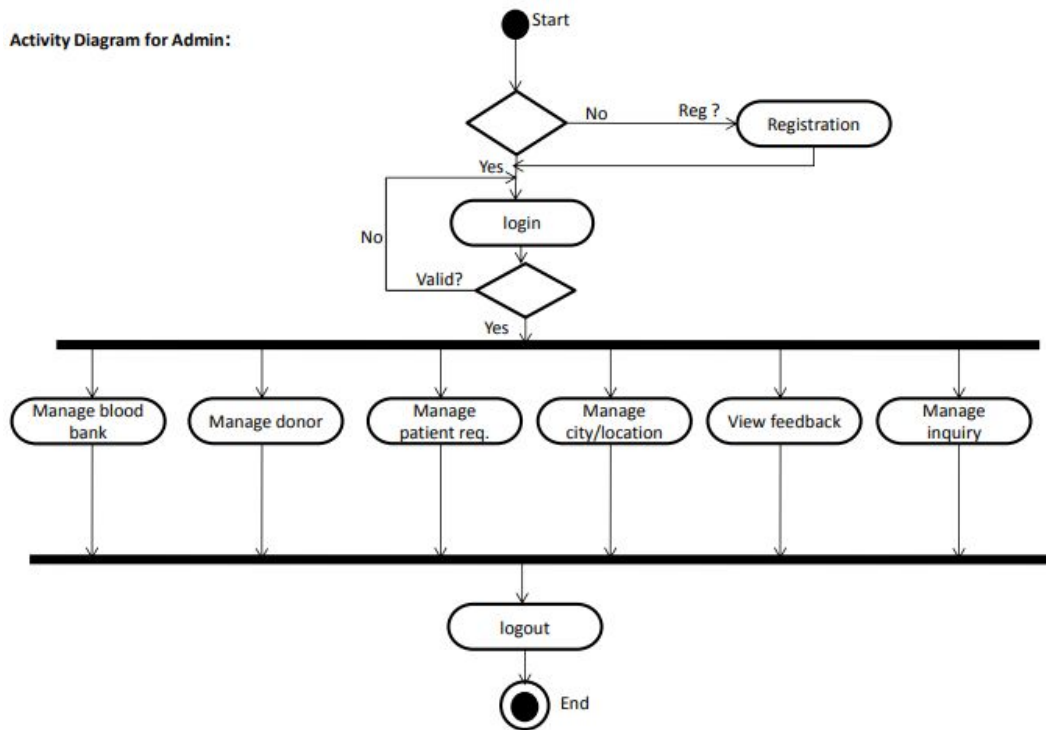
Patient :



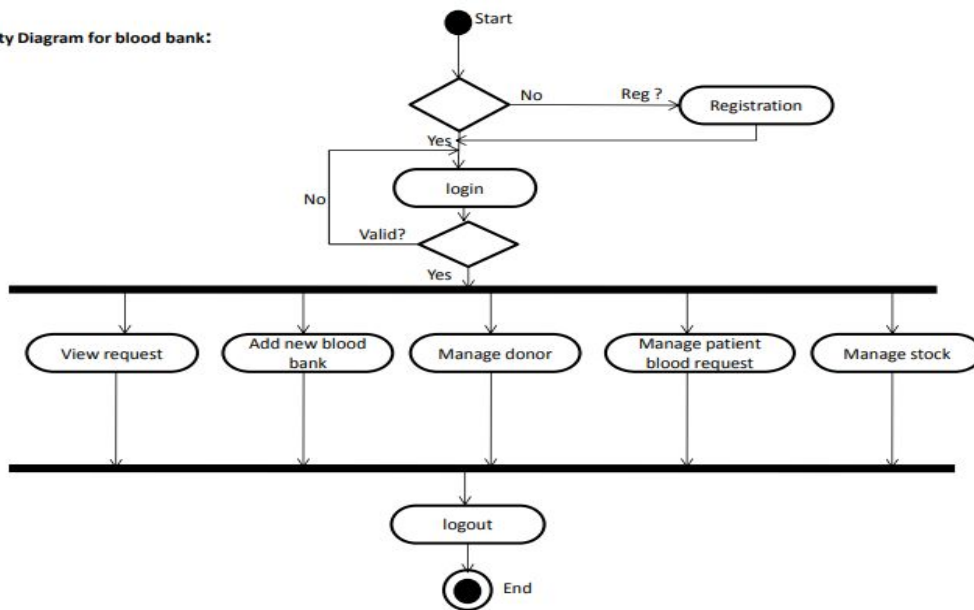


## ACTIVITY DIAGRAMS

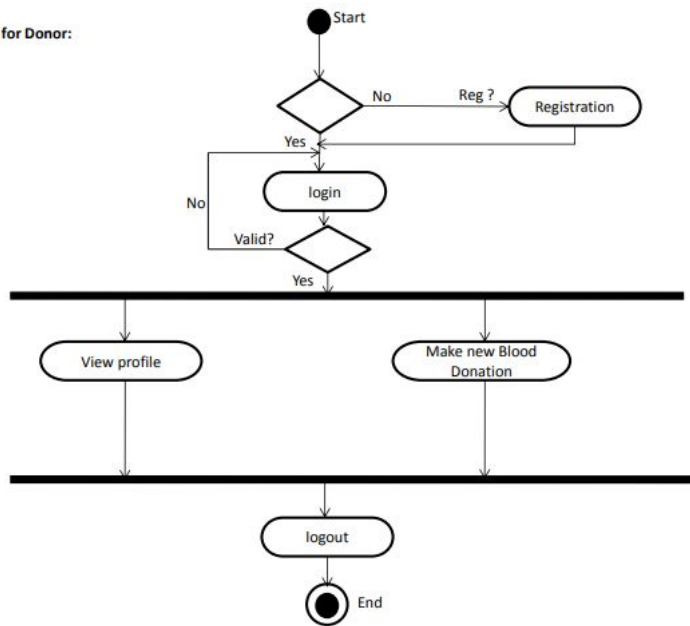
Activity Diagram for Admin:



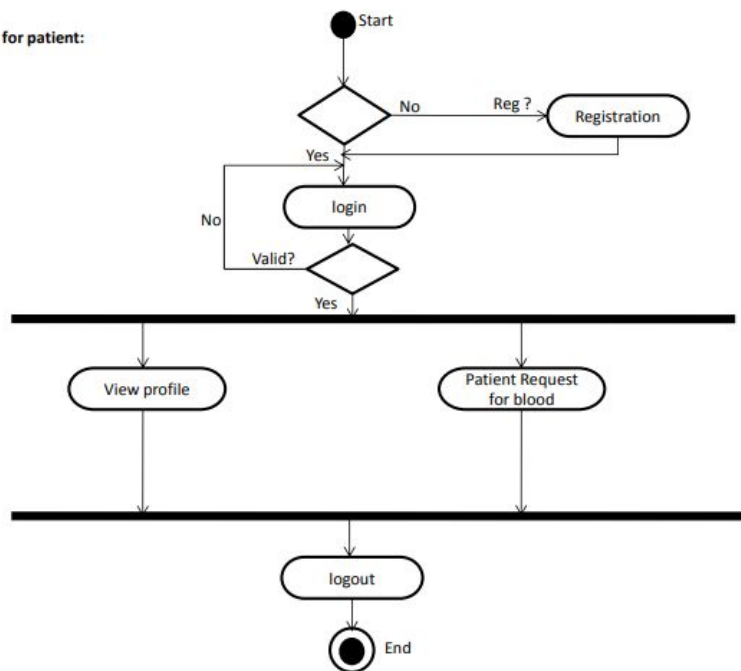
Activity Diagram for blood bank:



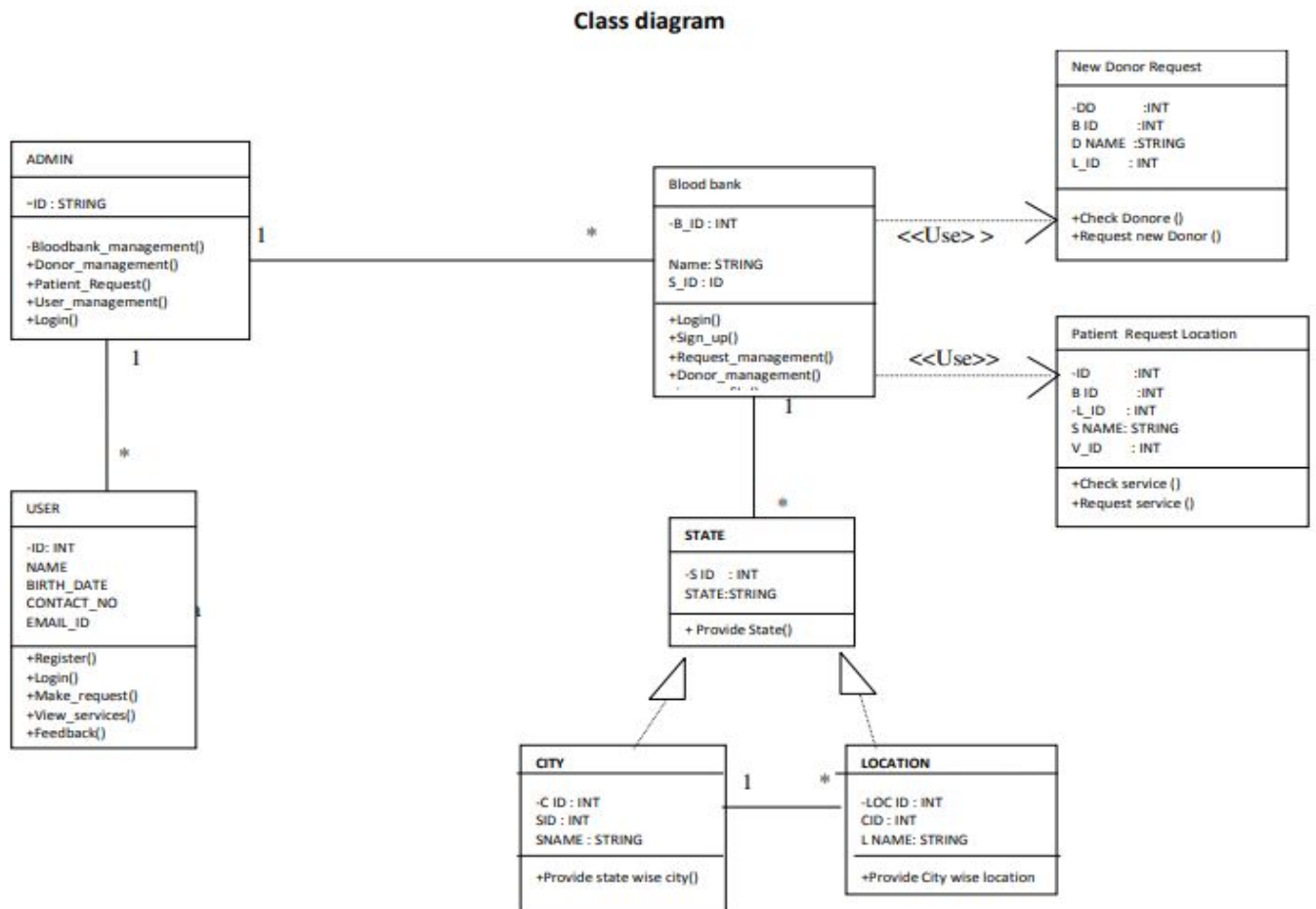
Activity Diagram for Donor:



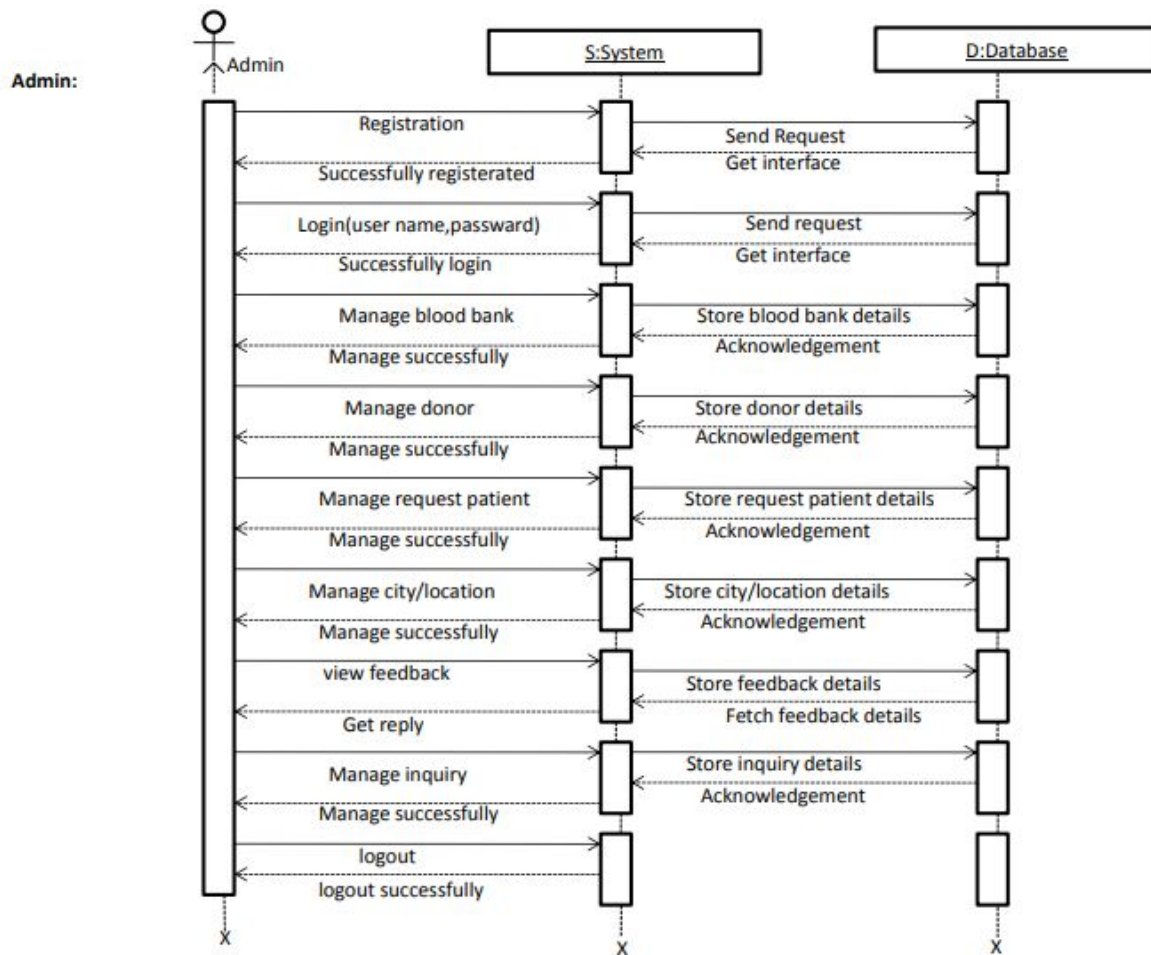
Activity Diagram for patient:



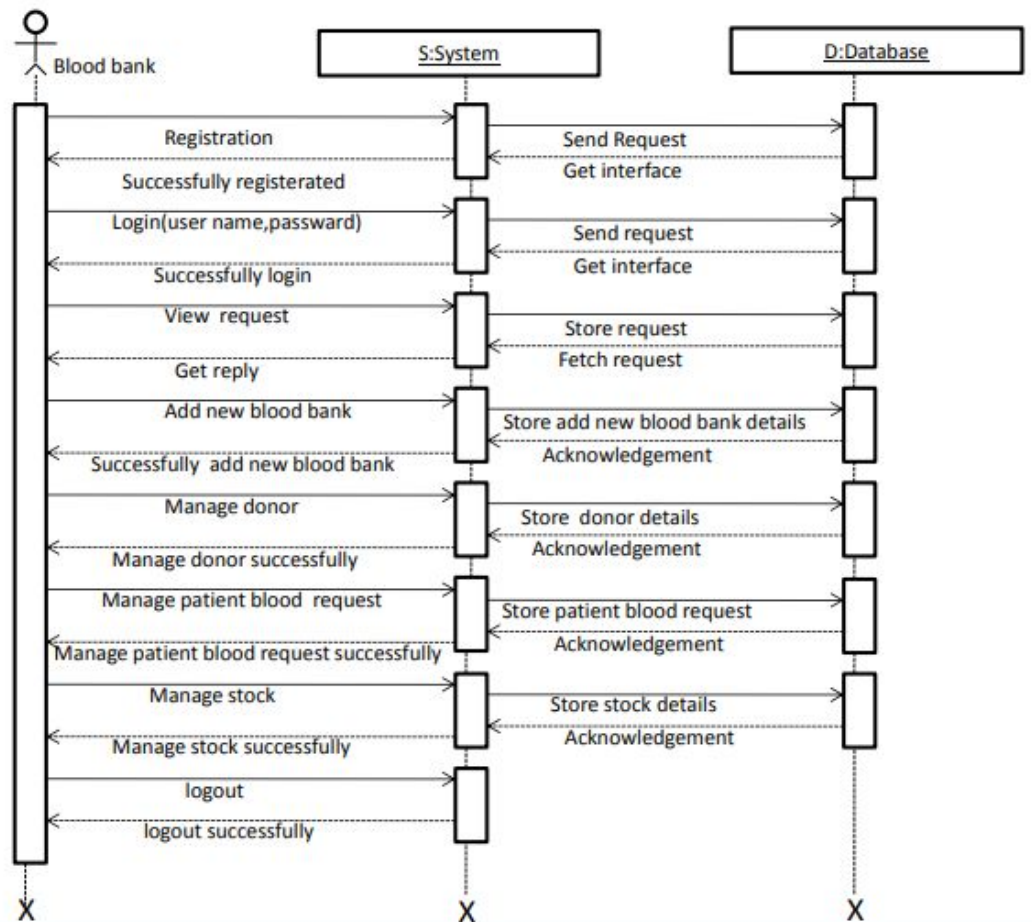
## CLASS DIAGRAM



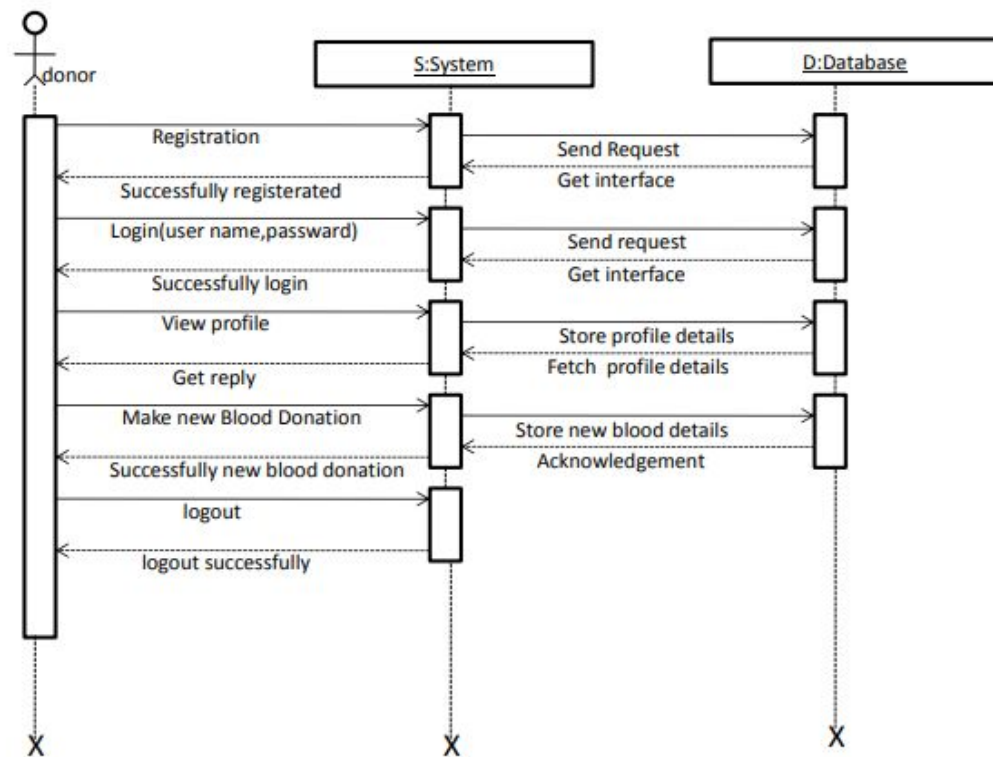
## SEQUENCE DIAGRAMS



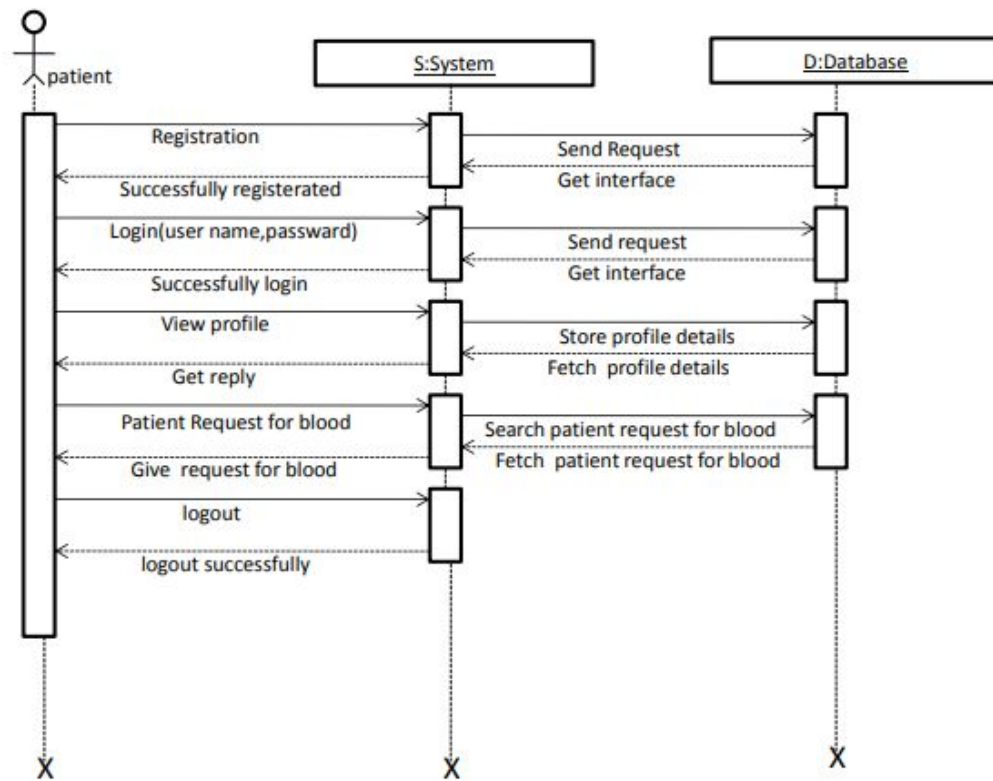
**Blood bank :**



Donor :

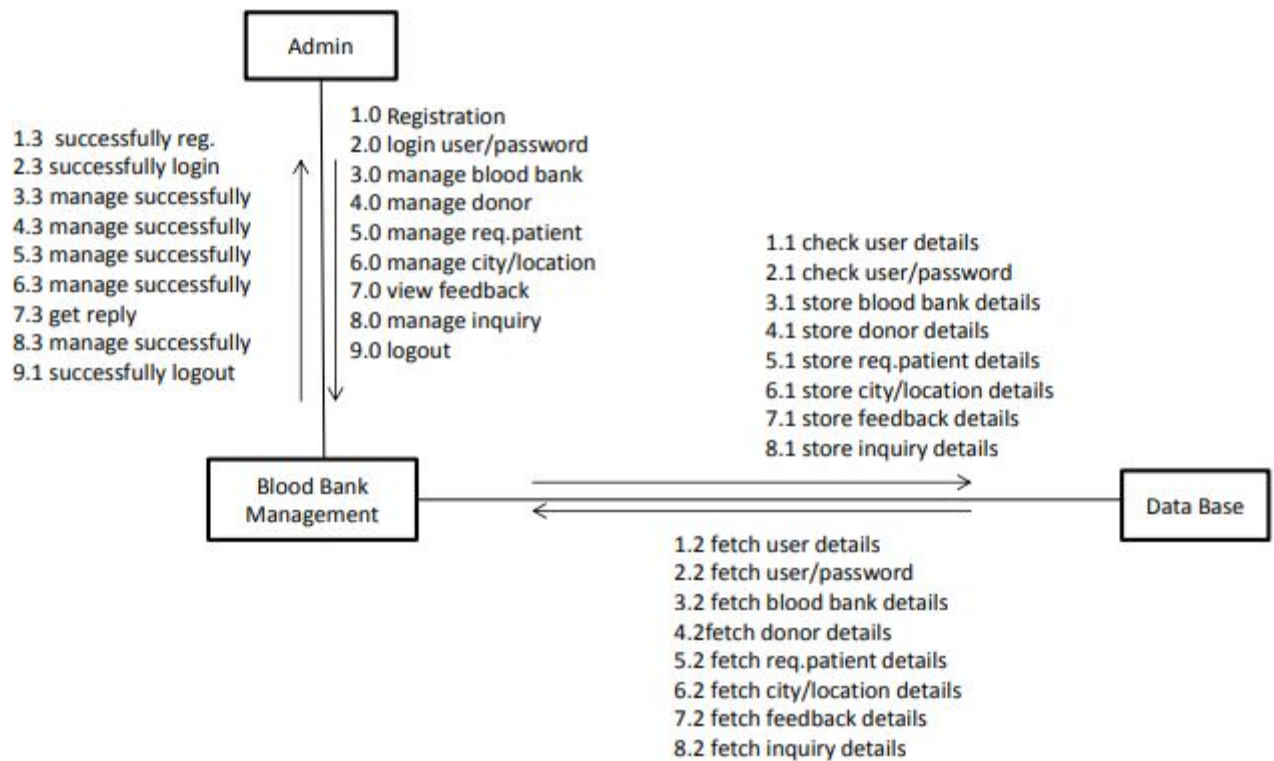


Patient :



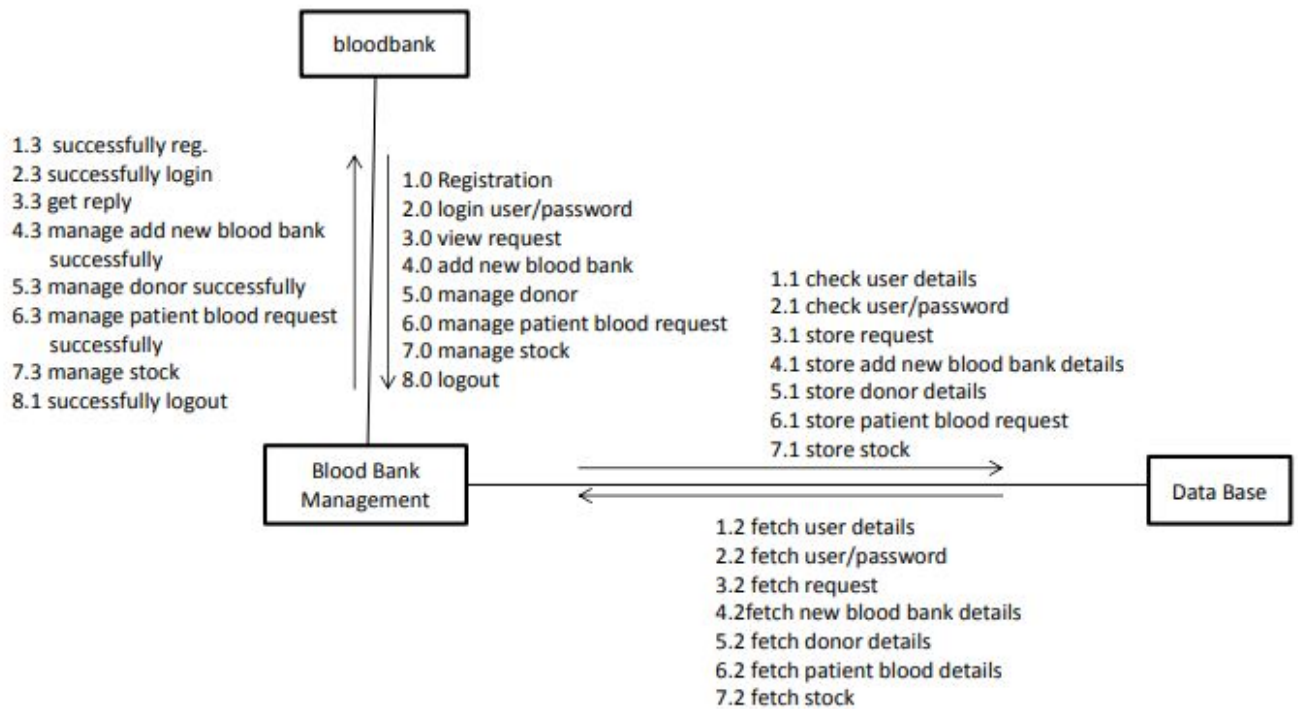
## Collaboration Diagram

Admin:

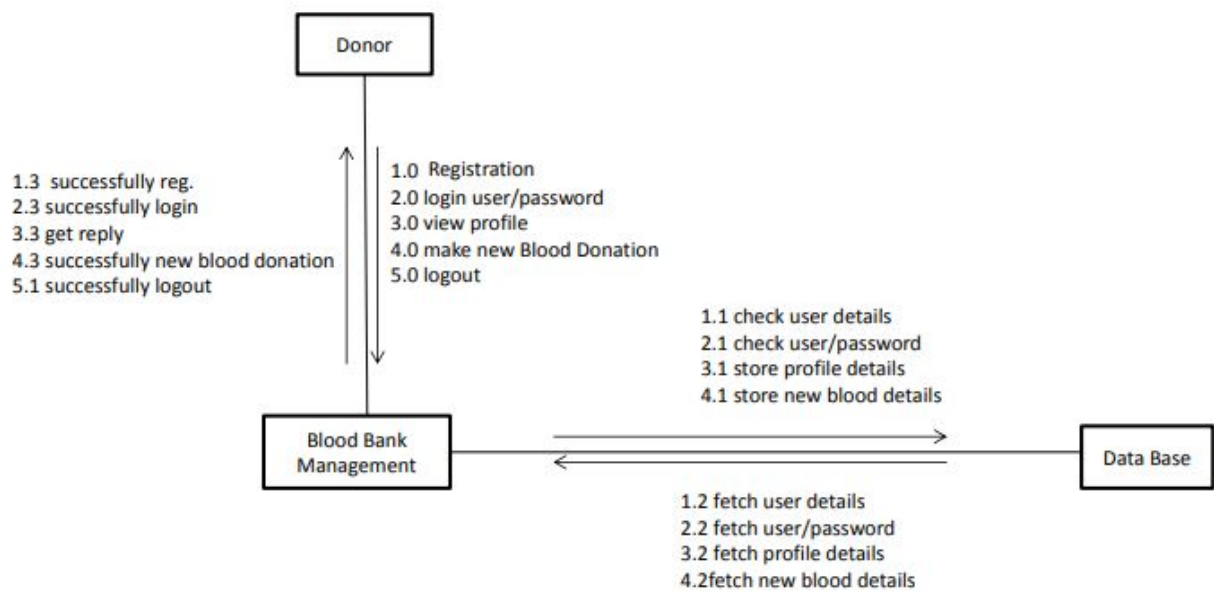




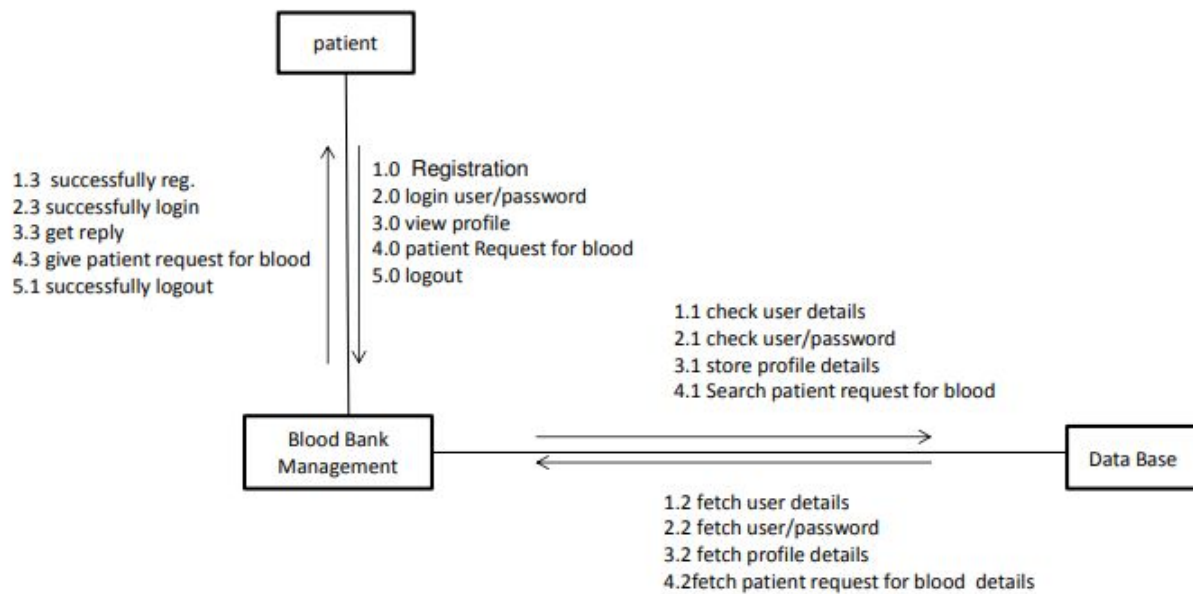
### Blood bank:



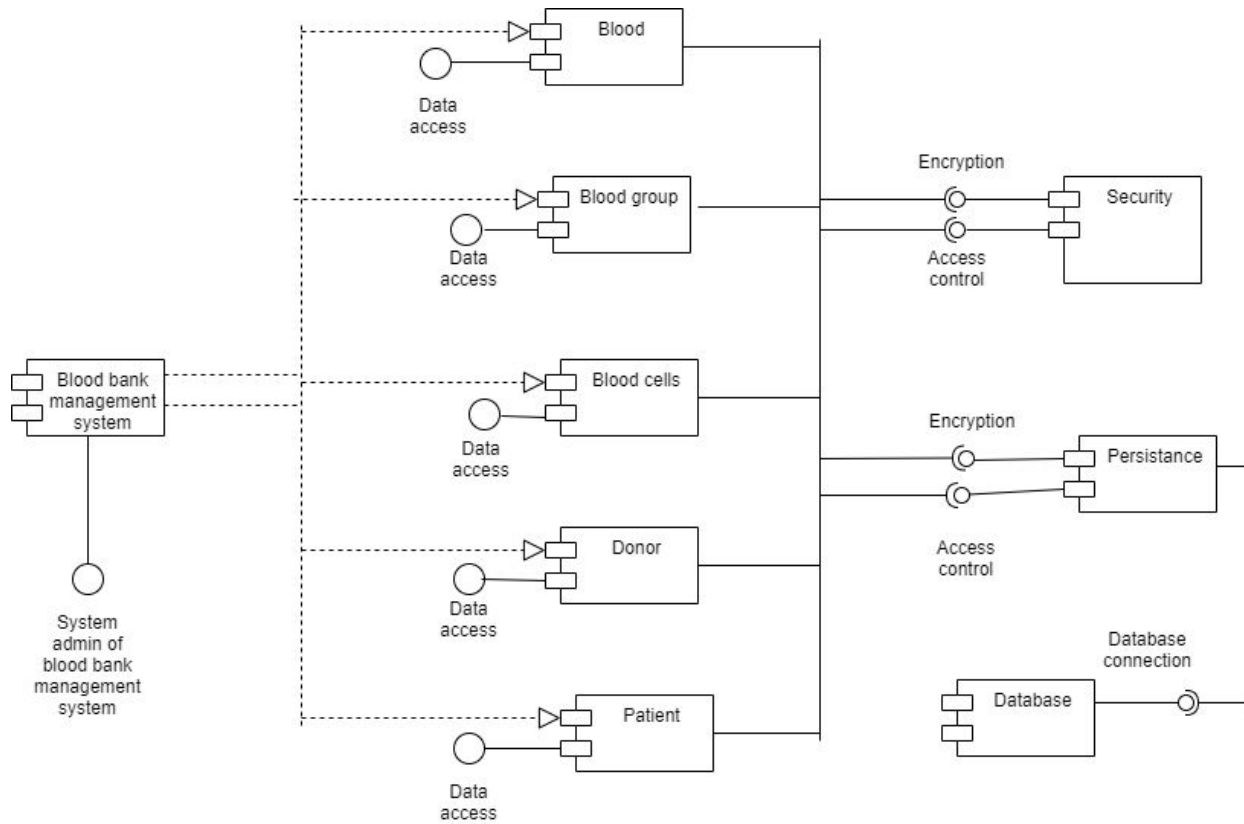
Donor :



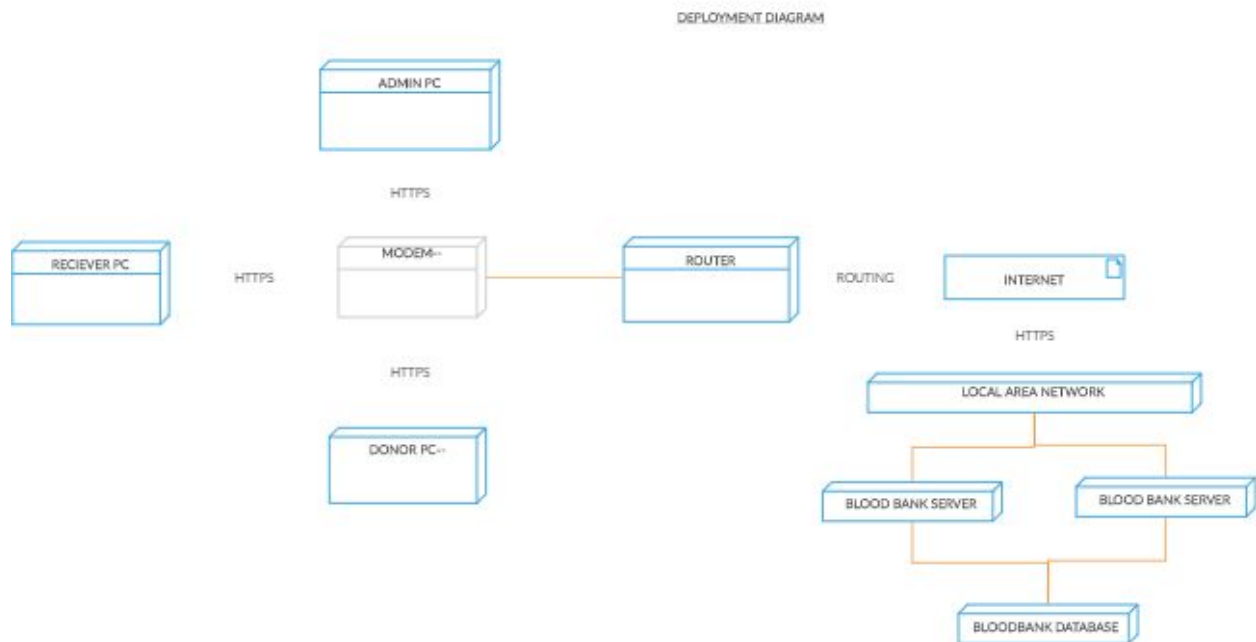
**Patient :**



## Component Diagram :



## Deployment Diagram :



## Unit Testing :

```
public class Complex {
    int real_part; int imaginary_part;
    public Complex(int r, int i) {
        real_part=r;
        imaginary_part=i;
    }
}
```

```

}

public Complex() {
    real_part=0;
    imaginary_part=0;
}

public boolean Equal(Complex c) {
    boolean result = false;
    if ((real_part==c.get_r()) && (imaginary_part==c.get_i())) result=true;
    return result;
}


public Complex Add(Complex c) {
    Complex result = new
    Complex(c.get_r()+real_part,c.get_i()+imaginary_part);
    return result;
}

public int get_r() { return real_part;}
public int get_i() { return imaginary_part; }
}

```

JUnit test cases for the above complex class

```
package javaapplication4;
```



```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
```

```
public class ComplexTest {
```

```
    Complex c1;
```

```
    Complex c2;
```

```
    public ComplexTest() {
    }
}
```

```
@BeforeClass
```

```
public static void setUpClass() {
}
}
```

```
@AfterClass
```

```
public static void tearDownClass() {
}
}
```



@Before

```
public void setUp() {  
    c1 = new Complex(7,3);  
    c2 = new Complex(12,6);  
}
```

@After

```
public void tearDown() {  
}
```

/\*\*

\* Test of Equal method, of class Complex.

\*/

@Test

```
public void testEqual() {  
    System.out.println("Equal");  
    assertTrue(!c2.Equal(c1));  
  
    assertTrue(c1.Equal(new Complex(7,3)));  
}
```

/\*\*

\* Test of Add method, of class Complex.

\*/



@Test

public void testAdd() {

    System.out.println("Add");

    Complex result = c1.Add(new Complex(5,3));

    assertEquals(result.get\_r(),c2.get\_r());

    assertEquals(result.get\_i(),c2.get\_i());

}

/\*\*

 \* Test of get\_r method, of class Complex.

\*/

@Test

public void testGet\_r() {

    System.out.println("get\_r");

    Complex instance = new Complex(5,10);

    int expResult = 5;

    int result = instance.get\_r();

    assertEquals(expResult, result);

}

/\*\*

 \* Test of get\_i method, of class Complex.

\*/

@Test

```
public void testGet_i() {  
    System.out.println("get_i");  
    Complex instance = new Complex(5,10);  
    int expResult = 10;  
    int result = instance.get_i();  
    assertEquals(expResult, result);  
  
}  
  
}
```

Testsuite: javaapplication4.ComplexTest

Add

Equal

get\_i

get\_r

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.09 sec

----- Standard Output -----


Add

Equal

get\_i

get\_r

-----




Integration Testing :

CalculatorService.java

```
public interface CalculatorService {  
    public double add(double input1, double input2);  
    public double subtract(double input1, double input2);  
    public double multiply(double input1, double input2);  
    public double divide(double input1, double input2);  
}
```

MathApplication.java

```
public class MathApplication {  
    private CalculatorService calcService;  
  
    public void setCalculatorService(CalculatorService calcService){  
        this.calcService = calcService;  
    }  
  
    public double add(double input1, double input2){  
        return calcService.add(input1, input2);  
    }  
  
    public double subtract(double input1, double input2){
```



```
        return calcService.subtract(input1, input2);
    }

    public double multiply(double input1, double input2){
        return calcService.multiply(input1, input2);
    }

    public double divide(double input1, double input2){
        return calcService.divide(input1, input2);
    }
}
```

Integration Test cases

MathApplicationTest.java

```
import static org.mockito.Mockito.when;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;

// @RunWith attaches a runner with the test class to initialize the test data
```



```
@RunWith(MockitoJUnitRunner.class)
```

```
public class MathApplicationTest {
```

```
    // @InjectMocks annotation is used to create and inject the mock object
```

```
    @InjectMocks
```

```
    MathApplication mathApplication = new MathApplication();
```

```
    // @Mock annotation is used to create the mock object to be injected
```

```
    @Mock
```

```
    CalculatorService calcService;
```

```
    @Test
```

```
    public void testAdd(){
```

```
        // add the behavior of calc service to add two numbers
```

```
        when(calcService.add(10.0, 20.0)).thenReturn(30.00);
```

```
        // test the add functionality
```

```
        Assert.assertEquals(mathApplication.add(10.0, 20.0), 30.0, 0);
```

```
    }
```

```
}
```

output

Testsuite: javaapplication4.MathApplicationTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.416 sec



test:

BUILD SUCCESSFUL (total time: 2 seconds)