SYSTEM DESIGN DOCUMENT – BLOODCONNECT

1. TABLE SCHEMAS

Table 1: USERS

Stores all users who can either donate or request blood. Admins are identified using the is_admin flag.

```
CREATE TABLE users (

        user_id INT AUTO_INCREMENT PRIMARY KEY,

        name VARCHAR(100) NOT NULL,

        email VARCHAR(100) UNIQUE NOT NULL,

        password_hash VARCHAR(255) NOT NULL,

        is_admin BOOLEAN DEFAULT FALSE,

        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);
```

Table 2: DONORS

Stores donor-specific details submitted through the donor form.

```
CREATE TABLE donors (

        donor_id INT AUTO_INCREMENT PRIMARY KEY,

        user_id INT NOT NULL,

        full_name VARCHAR(100) NOT NULL,

        age INT NOT NULL,

        date_of_birth DATE NOT NULL,

        last_donation_date DATE,

        disease_history TEXT,

        availability BOOLEAN DEFAULT TRUE,

        FOREIGN KEY (user_id) REFERENCES users(user_id)

);
```

Table 3: RECEIVERS

Acts as both the receiver and blood request table.

It stores receiver details and blood request information.

```
CREATE TABLE receivers (
        receiver_id INT AUTO_INCREMENT PRIMARY KEY,
        user_id INT NOT NULL,
        full_name VARCHAR(100) NOT NULL,
        age INT NOT NULL,
        date_of_birth DATE NOT NULL,
        blood_group_needed VARCHAR(5) NOT NULL,
        quantity_units INT NOT NULL,
        reason TEXT,
        doctor_prescription TEXT,
        latitude DECIMAL(9,6),
        longitude DECIMAL(9,6),
        status ENUM('PENDING','APPROVED','REJECTED') DEFAULT 'PENDING',
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

Table 4: ADMINS

Stores admin login credentials.

```
CREATE TABLE admins (
        admin_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        email VARCHAR(100) UNIQUE NOT NULL,
        password_hash VARCHAR(255) NOT NULL
);
```

2. FUNCTIONAL REQUIREMENTS

1.1 USER REGISTRATION (REQ-USER-101)

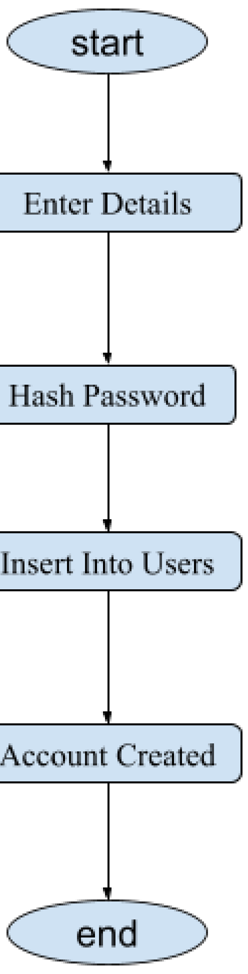
SQL QUERY:

```
INSERT INTO users (name, email, password_hash, is_admin)
VALUES (?, ?, ?, 0);
```

ALGORITHM:

```
1.Start
2.User enters name, email, and password
3.System validates email uniqueness
4.Hash password for security
5.Insert user details into users table
6.Display success message
7.End
```

FLOWCHART:



1.2 USER LOGIN (REQ-USER-102)

SQL QUERY:

SELECT * FROM users WHERE email = ? AND password_hash = ?;

ALGORITHM:

1.Start

2.User inputs email and password

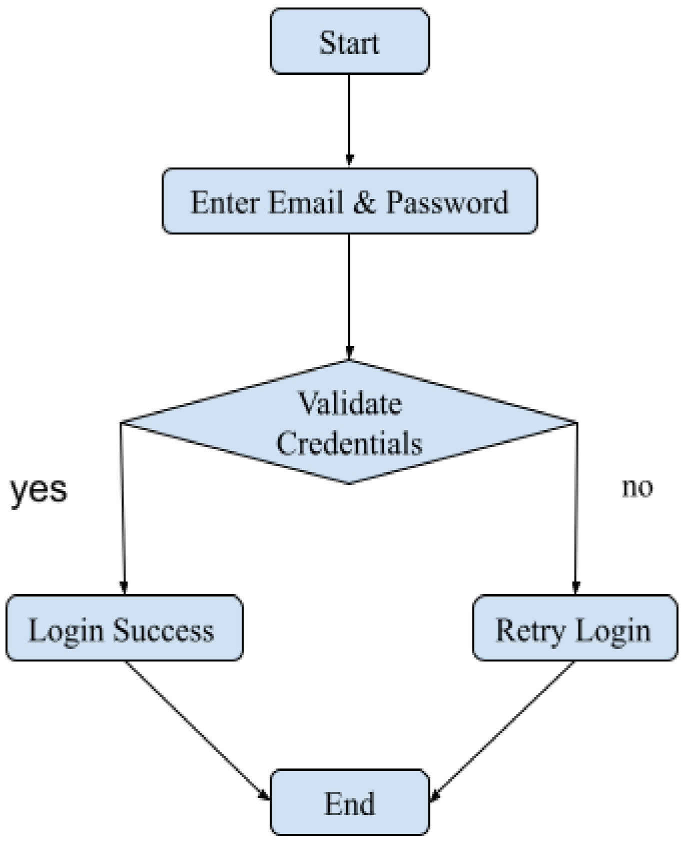
3.System validates credentials

4.If valid → redirect to homepage

5.Else → show error and retry

6.End

FLOWCHART:



1.3 ADMIN LOGIN (REQ-ADMIN-201)

SQL QUERY:

SELECT * FROM admins WHERE email = ? AND password_hash = ?;

ALGORITHM:

1.Start

2.Admin enters credentials

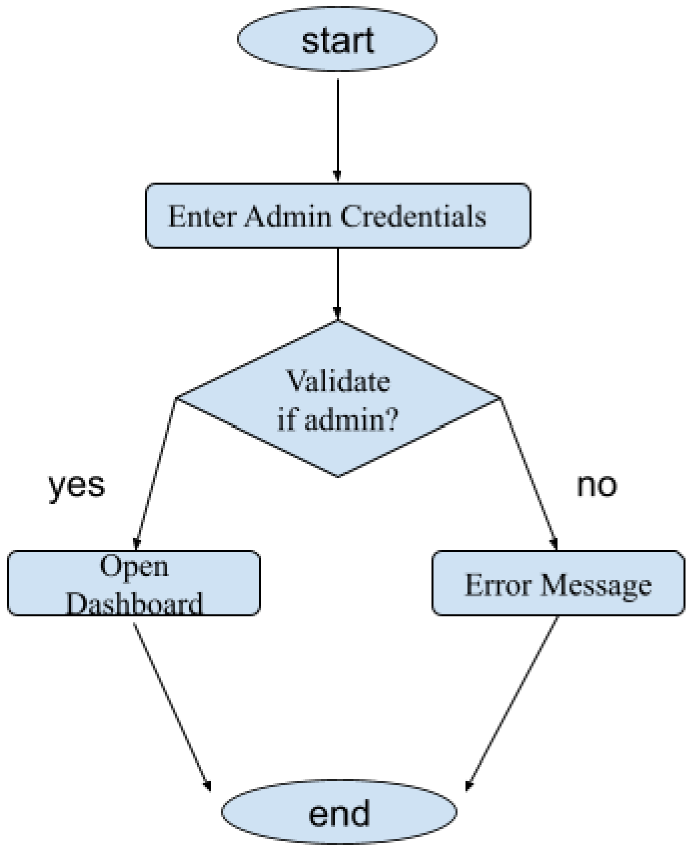
3.Validate admin details

4.If correct → open admin dashboard

5.Else → show error

6.End

FLOWCHART:



2.1 DONOR FORM SUBMISSION (REQ-REQUEST-301)

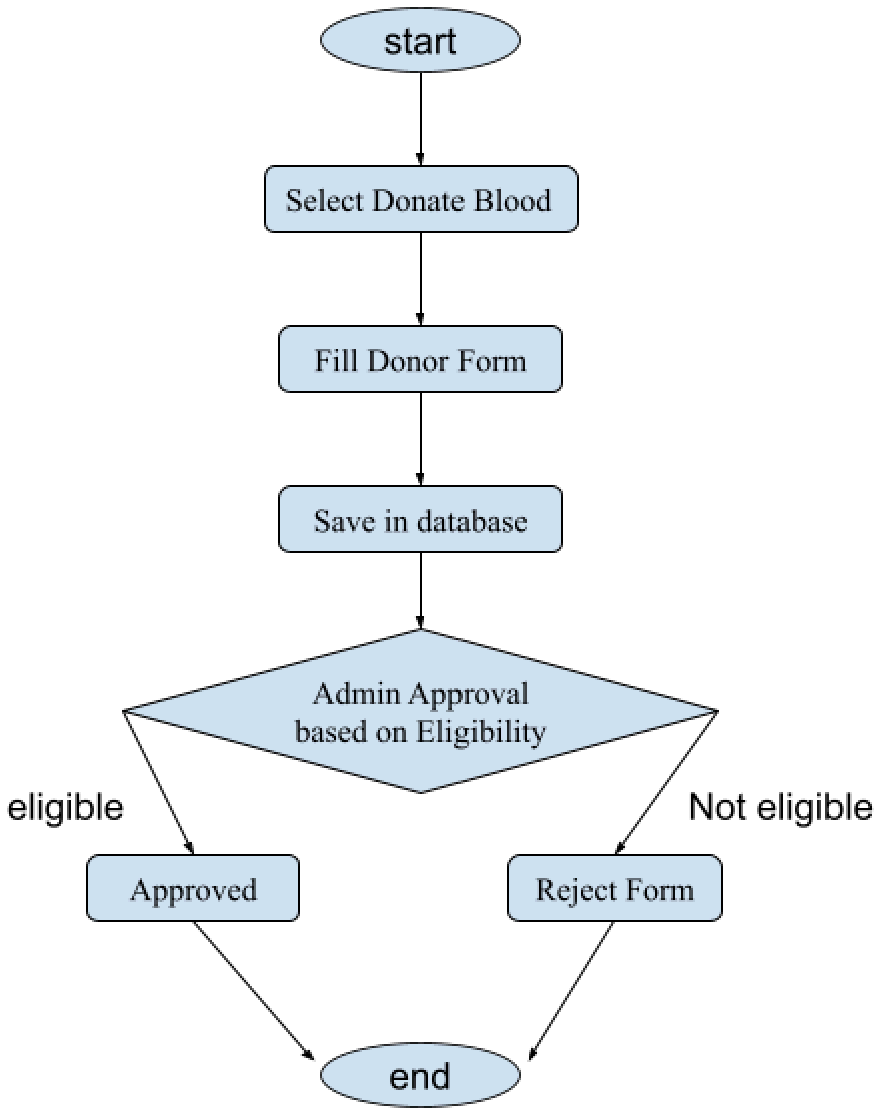
SQL QUERY:

```
INSERT INTO donors (user_id, full_name, age, date_of_birth,
                    last_donation_date, disease_history, availability)
VALUES (?, ?, ?, ?, ?, ?, ?);
```

ALGORITHM:

```
1.Start
2.User chooses Donate Blood option
3.Fills donor form (name, age, DOB, medical details)
4.System checks eligibility:
5.Must be ≥ 3 months since last donation
6.Must have no disqualifying disease
7.If eligible → send for admin approval
8.Admin reviews and approves/rejects
9.End
```

FLOWCHART:



3.1 RECEIVER FORM SUBMISSION (REQ-REQUEST-301)

SQL QUERY:

```
INSERT INTO receivers (user_id, full_name, age, date_of_birth,
            blood_group_needed, quantity_units, reason, doctor_prescription,
            latitude, longitude)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
```

ALGORITHM:

1.Start

2.User selects Request Blood option

3.Fills receiver form (personal details, blood group, location)

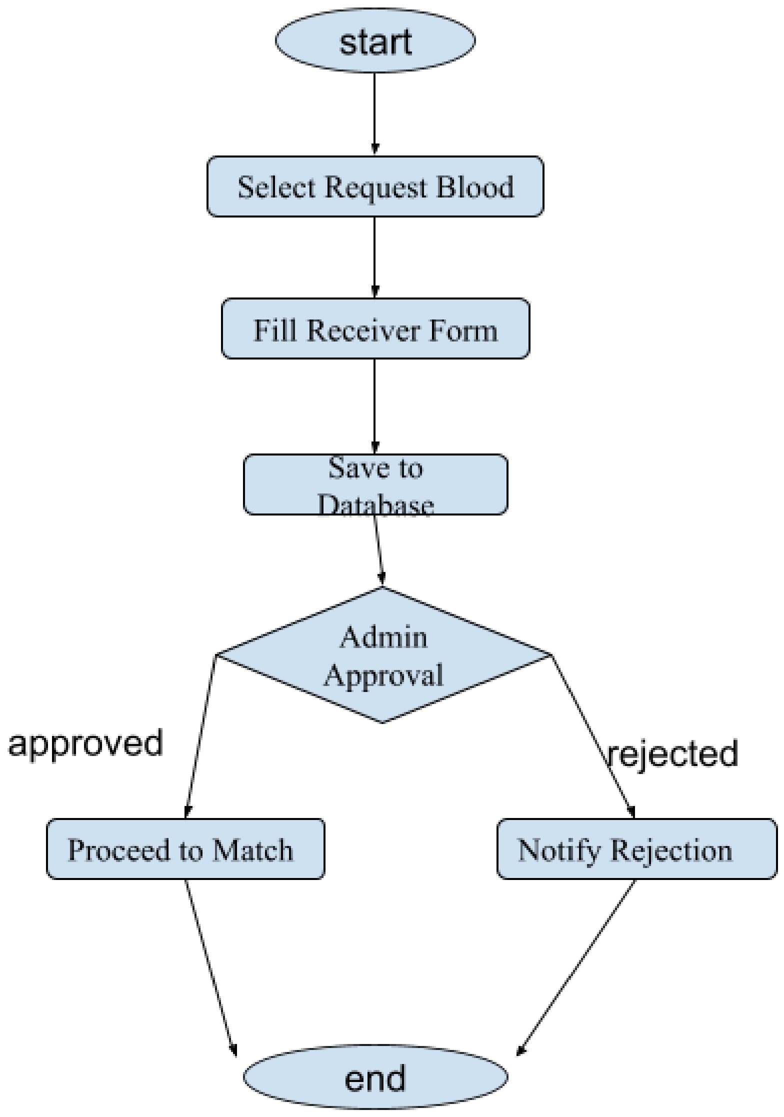
4.System stores data in receivers table (status = PENDING)

5.Admin reviews and approves/rejects

6.Approved → moves to donor matching

7.End

FLOWCHART:



4.1 DONOR–RECEIVER MATCHING (REQ-MATCH-401)

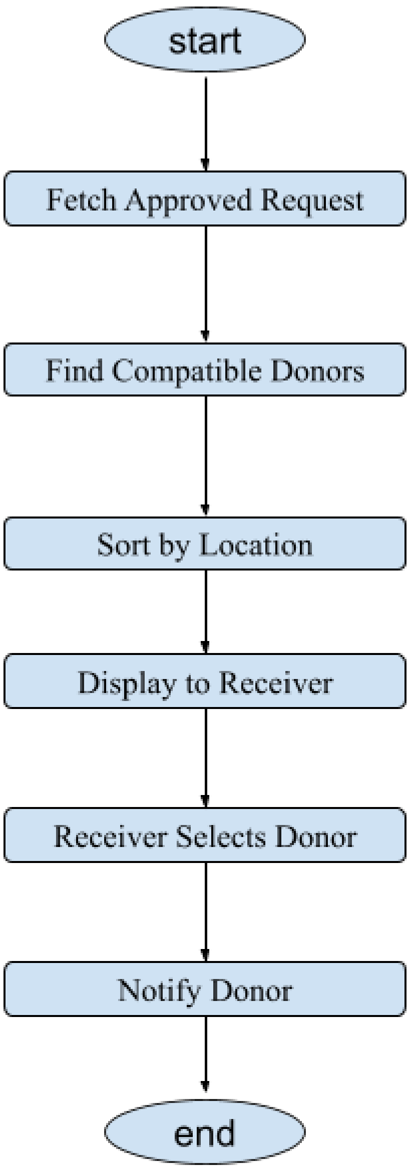
SQL QUERY:

```
SELECT d.donor_id, d.full_name, u.name, u.email
FROM donors d
JOIN users u ON d.user_id = u.user_id
WHERE d.availability = TRUE
AND d.disease_history IS NULL
AND d.age BETWEEN 18 AND 65
ORDER BY ABS(d.latitude - ?) + ABS(d.longitude - ?) ASC;
```

ALGORITHM:

```
1.Start
2.Retrieve approved receiver request
3.Find donors with same blood group and available status
4.Sort by nearest location
5.Display donor list to receiver
6.Receiver selects donor
7.Notify donor
8.End
```

FLOWCHART:


start
Fetch Approved Request
Find Compatible Donors
Sort by Location
Display to Receiver
Receiver Selects Donor
Notify Donor
end

4.2 COMMUNICATION PHASE (REQ-COMM-501)

SQL QUERY:

```
UPDATE donors SET availability = FALSE WHERE donor_id = ?;
```

ALGORITHM:

1.Start

2.Donor accepts receiver request

3.System locks donor availability

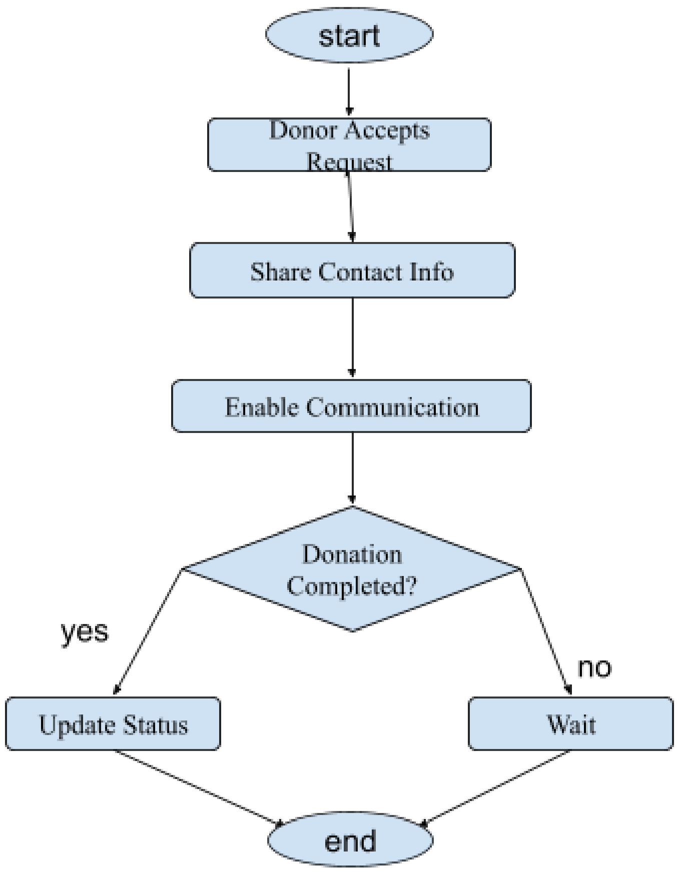
4.Exchange contact details securely

5.Enable communication (chat/call)

6.After donation, update request status → FULFILLED

7.End

FLOWCHART:



5. OVERALL SYSTEM FLOW (Covers REQ-USER-101 to REQ-COMM-501)

ALGORITHM (Summary):

1.Start

2.User Registers and Logs In

3.Chooses to Donate or Request

4.If Donate → Fill Donor Form → Eligibility Check → Admin Approval

5.If Request → Fill Receiver Form → Admin Approval

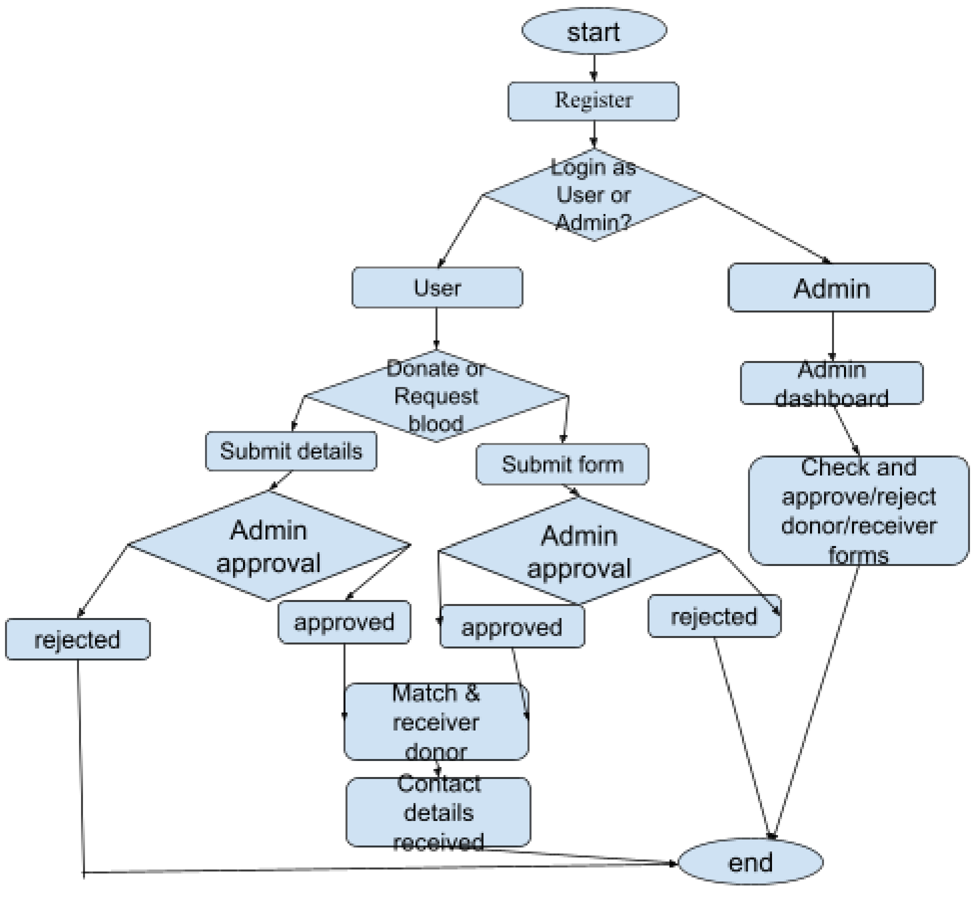
6.System Matches Donor and Receiver

7.Receiver selects Donor

8.Communication Established

9.End

FLOWCHART:


start
Register
Login as User or Admin?
User
Admin
Donate or Request blood
Admin dashboard
Submit details
Submit form
Check and approve/reject donor/receiver forms
Admin approval
Admin approval
rejected
approved
approved
rejected
Match & receiver donor
Contact details received
end