**Project Requirements Document (PRD): Lifeline**

**Version:** 1.1  
**Date:** June 7, 2025

**1. User Journeys & Core Features**

The system is built around three primary user personas and one system persona. All user accounts are stored in a single unified system.

* **Persona: Donor**
  + **Primary Goals:** Easily register and update their profile, understand their eligibility status through a quiz, see current blood needs, and respond to urgent donation calls.
  + **Interface:** Mobile Application.
* **Persona: Blood Bank Staff**
  + **Primary Goals:** Manage the donor database, conduct on-site health screenings for walk-in donors, manage the entire lifecycle of individual blood bags (inventory), review and fulfill hospital requests, and manage cross-match reporting.
  + **Interface:** Web Application (Dashboard).
* **Persona: Hospital Staff**
  + **Primary Goals:** Quickly and accurately submit blood requests for patients, track the real-time status of those requests, and view the details of allocated blood units.
  + **Interface:** Web Application (Specialized Hospital View).
* **Persona: System (Automation & Rules Engine)**
  + **Primary Goals:** Automatically identify eligible donors based on needs, trigger notifications, and apply inventory management rules (e.g., FEFO - First-Expired, First-Out).
  + **Key Processes:** Rules-Based Match Engine → Notification Service.

**2. Domain & Process Flow**

The end-to-end process is event-driven and reactive to clinical needs.

1. \*\*Request Trigger

**Refined PRD Section 6: Database Schema & Settings (MySQL)**

**Design Principle:**  
To ensure a single source of truth for identity and authentication, we will use a unified Users table. This table will contain universal information for every person who can log in (email, password, role). Role-specific data will be stored in separate "profile" tables, linked one-to-one with the Users table. This approach is normalized, secure, and scalable.

**Revised SQL Schema (MySQL - Corrected & Normalized):**

Generated sql

-- Set foreign key checks to 0 to avoid errors during table drops

SET FOREIGN\_KEY\_CHECKS=0;

-- Drop tables in reverse order of creation

DROP TABLE IF EXISTS `AppSettings`;

DROP TABLE IF EXISTS `DeferralCriteria`;

DROP TABLE IF EXISTS `Notifications`;

DROP TABLE IF EXISTS `BloodRequestItems`;

DROP TABLE IF EXISTS `BloodRequests`;

DROP TABLE IF EXISTS `BloodBags`;

DROP TABLE IF EXISTS `HealthScreenings`;

DROP TABLE IF EXISTS `DonorProfiles`; -- New Profile Table

DROP TABLE IF EXISTS `StaffDetails`; -- New Profile Table

DROP TABLE IF EXISTS `Users`; -- The ONE Users table

DROP TABLE IF EXISTS `Hospitals`;

DROP TABLE IF EXISTS `BloodBanks`;

-- Set foreign key checks back to 1 after dropping tables

SET FOREIGN\_KEY\_CHECKS=1;

-- Table: BloodBanks

CREATE TABLE `BloodBanks` (

`BankID` INT AUTO\_INCREMENT PRIMARY KEY,

`Name` VARCHAR(255) NOT NULL,

`Address` TEXT NOT NULL,

`City` VARCHAR(100) NOT NULL,

`ContactPhone` VARCHAR(20),

`OperatingHours` VARCHAR(255)

) ENGINE=InnoDB;

-- Table: Hospitals

CREATE TABLE `Hospitals` (

`HospitalID` INT AUTO\_INCREMENT PRIMARY KEY,

`Name` VARCHAR(255) NOT NULL,

`Address` TEXT NOT NULL,

`City` VARCHAR(100) NOT NULL,

`ContactPhone` VARCHAR(20)

) ENGINE=InnoDB;

-- =================================================================

-- CORE IDENTITY & AUTHENTICATION TABLE

-- =================================================================

-- Table: Users (Unified table for ALL actors)

CREATE TABLE `Users` (

`UserID` INT AUTO\_INCREMENT PRIMARY KEY,

`FirstName` VARCHAR(100) NOT NULL,

`LastName` VARCHAR(100) NOT NULL,

`Email` VARCHAR(255) UNIQUE NOT NULL,

`PasswordHash` VARCHAR(255) NOT NULL,

`ContactPhone` VARCHAR(20),

`Role` ENUM('donor', 'blood\_bank\_staff', 'hospital\_staff') NOT NULL,

`IsActive` BOOLEAN DEFAULT TRUE,

`CreatedAt` TIMESTAMP DEFAULT CURRENT:\*\*

\* \*\*Path A (Hospital Need):\*\* A Hospital Staff submits a new Blood Request for a patient.

\* \*\*Path B (Inventory Need):\*\* The system detects that stock for a particular blood type has fallen below a critical threshold defined in `AppSettings`.

2. \*\*System Match & Notify:\*\*

\* The system’s \*\*Rules Engine\*\* identifies the best potential donors based on: blood type needed, eligibility status, last donation date, and recent health screening status (if available).

\* The \*\*Notification Service\*\* sends a targeted push notification/SMS to the selected donor pool, informing them of the urgent need.

3. \*\*Donor Action:\*\*

\* The Donor receives the notification and views the need in their mobile app.

\* The Donor must complete/pass the in-app \*\*Eligibility Quiz\*\* if they haven't recently.

\* If eligible, the Donor is encouraged to visit "The Blood Bank" for an on-site screening and donation.

4. \*\*On-Site Process (Walk-in, No Appointment):\*\*

\* The Donor arrives at the Blood Bank.

\* A Blood Bank Staff checks in the donor by looking up their profile.

\* The staff conducts the full \*\*On-Site Health Screening\*\* (questionnaire, vitals, hemoglobin).

\* If the donor passes, the \*\*Blood Donation\*\* proceeds.

5. \*\*Inventory Management:\*\*

\* A new `BloodBags` record is created in the system for the collected unit, with a status of 'Pending Testing'.

\* After testing, the status is updated to 'Available'.

6. \*\*Request Fulfillment & Cross-Match:\*\*

\* A Blood Bank Staff reviews the pending Hospital Request.

\* They select the most appropriate 'Available' `BloodBag` based on system rules (FEFO, patient special requirements).

\* They perform the cross-match test.

\* They \*\*Upload/Confirm the Cross-match Report\*\* within the fulfillment workflow. On 'Pass', the `BloodBag` status becomes 'Reserved' or 'Crossmatched'.

\* The `BloodBag` is allocated to the request.

7. \*\*Issuance & Completion:\*\*

\* The allocated `BloodBag` is physically issued to the hospital. The status is updated to 'Issued'.

\* The Hospital is notified that the blood is on its way.

\* The `HospitalRequest` is marked as 'Fulfilled'. All parties see the final status.

\*\*3. System Architecture\*\*

\* \*\*Layers & Components:\*\*

\* \*\*API Gateway (Node.js/Express or similar):\*\* Single entry point for all client requests.

\* \*\*Auth Service:\*\* Manages authentication (JWT-based) and authorization for all users\_TIMESTAMP,

`UpdatedAt` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

) ENGINE=InnoDB;

CREATE INDEX `idx\_users\_email` ON `Users`(`Email`);

-- =================================================================

-- PROFILE TABLES (For role-specific data)

-- =================================================================

-- Table: DonorProfiles (1-to-1 with Users, for donor-specific info)

CREATE TABLE `DonorProfiles` (

`UserID` INT PRIMARY KEY, -- This is both PK and FK

`BloodType` ENUM('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-') NULL,

`DateOfBirth` DATE NOT NULL,

`Gender` ENUM('Male', 'Female') NOT NULL,

`LastDonationDate` DATE NULL,

`NextEligibleDonationDate` DATE NULL,

`IsEligible` BOOLEAN DEFAULT TRUE,

`Notes` TEXT,

`UpdatedAt` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (`UserID`) REFERENCES `Users`(`UserID`) ON DELETE CASCADE

) ENGINE=InnoDB;

-- Table: StaffDetails (1-to-1 with Users, for staff-specific info)

CREATE TABLE `StaffDetails` (

`UserID` INT PRIMARY KEY, -- This is both PK and FK

`HospitalID` INT NULL, -- Set if role is 'hospital\_staff'

`BankID` INT NULL, -- Set if role is 'blood\_bank\_staff'

`JobTitle` VARCHAR(100),

`IsAdmin` BOOLEAN DEFAULT FALSE, -- To give special permissions to some staff

`UpdatedAt` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (`UserID`) REFERENCES `Users`(`UserID`) ON DELETE CASCADE,

FOREIGN KEY (`HospitalID`) REFERENCES `Hospitals`(`HospitalID`) ON DELETE SET NULL,

FOREIGN KEY (`BankID`) REFERENCES `BloodBanks`(`BankID`) ON DELETE SET NULL

) ENGINE=InnoDB;

-- =================================================================

-- OPERATIONAL TABLES

-- =================================================================

-- Table: HealthScreenings (No AppointmentID)

CREATE TABLE `HealthScreenings` (

`ScreeningID` INT AUTO\_INCREMENT PRIMARY KEY,

`DonorID` INT NOT NULL,

`BankID` INT NOT NULL,

`StaffID` INT NOT NULL,

`ScreeningDate` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

`Weight` REAL,

`BloodPressure` VARCHAR(20),

`HemoglobinLevel` REAL,

`Temperature` REAL,

`QuestionnaireSummary` JSON NULL,

`DeferralReason` VARCHAR(255) NULL,

`IsEligibleOnDay` BOOLEAN NOT NULL,

FOREIGN KEY (`DonorID`) REFERENCES `Users`(`UserID`),

FOREIGN KEY (`BankID`) REFERENCES `BloodBanks`(`BankID`),

FOREIGN KEY (`StaffID`) REFERENCES `Users`(`UserID`)

) ENGINE=InnoDB;

-- Table: BloodBags (No AppointmentID)

CREATE TABLE `BloodBags` (

`BagID` VARCHAR(50) PRIMARY KEY,

`DonorID` INT NOT NULL,

`HealthScreeningID` INT NOT NULL,

`BankID` INT NOT NULL,

`BloodType` ENUM('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-') NOT NULL,

`ComponentType` VARCHAR(50) NOT NULL DEFAULT 'Whole Blood',

`CollectionDate` TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

`ExpiryDate` DATE NOT NULL,

`Status` ENUM('Pending Testing', 'Available', 'Reserved', 'Crossmatched', 'Issued', 'Used', 'Discarded in the single `Users` table.

\* \*\*Core Services (Microservice approach):\*\*

\* \*\*User & Donor Service:\*\* Manages user profiles, eligibility quiz results, and donation history.

\* \*\*Inventory Service:\*\* Manages the `BloodBags` table and its entire lifecycle.

\* \*\*Request Service:\*\* Manages `BloodRequests` and their status changes.

\* \*\*Screening Service:\*\* Manages the logic and state of on-site health screenings and `DeferralCriteria`.

\* \*\*Rules & Notification Service:\*\* Contains the logic for matching donors and triggering notifications.

\* \*\*Databases:\*\*

\* \*\*PostgreSQL or MySQL:\*\* Primary database for all transactional data.

\* \*\*Redis (Optional but Recommended):\*\* For caching frequently accessed data and managing notification queues.

\* \*\*Frontend:\*\*

\* \*\*Web Dashboard (React + TypeScript):\*\* Single-page application for Hospital and Blood Bank Staff.

\* \*\*Mobile App (React Native):\*\* Cross-platform mobile app for Donors.

\* \*\*CI/CD & Infrastructure:\*\*

\* \*\*Docker\*\* containers for all services.

\* \*\*Kubernetes\*\* (e.g., EKS, GKE) for container orchestration.

\* \*\*GitHub Actions\*\* for automated build, test, and deployment pipelines.

\*\*4. Development Roadmap\*\*

1. \*\*Sprint 0 – Foundations:\*\*

\* Setup Git repo, CI/CD pipeline, and project boards.

\* Deploy the \*\*final, unified database schema\*\* to a development environment.

\* Implement user authentication service (Register/Login for all roles using the single `Users` table).

\* Populate `AppSettings` and `DeferralCriteria` with initial data.

2. \*\*Sprint 1 – Donor & Hospital MVP:\*\*

\* \*\*Donor App:\*\* Implement Profile Management, the full Eligibility Quiz flow, and the "View Blood Needs" dashboard.

\* \*\*Hospital App:\*\* Implement the "Submit Blood Request" form and the "View Request Status" dashboard.

3. \*\*Sprint 2 – Blood Bank Core Workflow:\*\*

\* \*\*Blood Bank App:\*\* Implement Donor Check-in (walk-in), the full On-Site Health Screening workflow, and the creation of new `BloodBags` in inventory upon successful donation.

4. \*\*Sprint 3 – Fulfillment & Cross-Match:\*\*

\* \*\*Blood Bank App:\*\* Implement the "Review Hospital Requests" queue. Build the workflow to select an available `BloodBag`, mark it as cross-matched, and fulfill the request.

5. \*\*Sprint 4 – Rules Engine & Notifications:\*\*

\* Implement the backend \*\*Rules Engine\*\* that finds eligible donors based on low stock or new requests.

\* Connect the engine to a \*\*Notification Service\*\* to send alerts to the Donor mobile app.

6. \*\*Sprint 5 – Polish & Testing:\*\*

\* End-to-end testing of all user journeys.

\* UI/UX polish and refinement.

\* Security audit and performance optimization.

---

\*\*5. Detailed Use Cases (Ultra-Concise)\*\*

\*(The concise use cases from the previous answer remain valid, as they focus on the UI flow which is unaffected by the backend database structure change.)\*

---