# Blood Bank Management System - Requirements Analysis

## 1. Functional Requirements

### 1.1 User Management

* Users can register as donors using national ID.
* Admins can log in using email and password.
* Authenticated users receive JWT access tokens.
* Role-based access control to secure endpoints.

### 1.2 Donor Module

* Donors can log in and view their donation history.
* Donors can submit a blood donation.
* Validations on donation:
  + Virus test must be negative.
  + At least 3 months since last donation.

### 1.3 Donation Management

* Stores donations with metadata (blood type, city, expiration, etc.).
* Marks donation as used when matched to hospital request.
* User can get all his donations and admin can get all donations (with the same endpoint)

### 1.4 Hospital Requests

* Hospitals can submit blood requests specifying:
  + Blood type
  + Quantity
  + City
  + Patient status (Normal, Urgent, Immediate)
* Requests saved in DB until count reaches 10.

### 1.5 Request Fulfillment

* Once 10+ unfulfilled requests exist:
  + System prioritizes based on urgency and distance.
  + Matches available, valid donations.
  + Fulfills request by marking donations as used.
  + Marks request as fulfilled.

### 1.6 Admin Management

* Admins can create new admin accounts.
* Admins can view donor lists.

### 1.7 Notifications

* Donors receive rejection email if their donation is declined.

### 1.8 Health Check

* Root endpoint returns welcome message.

## 2. Non-Functional Requirements

### 2.1 Performance

* Donation matching optimized using greedy algorithm.
* System handles up to thousands of donors and requests.

### 2.2 Security

* JWT-based authentication.
* Role-based access guards.
* Passwords hashed using bcrypt.

### 2.3 Maintainability

* Clean modular architecture (NestJS).
* Separation of concerns: services, controllers, DTOs.
* Strong typing with TypeScript.

### 2.4 Testability

* Unit tests written with Jest.
* Services are isolated and tested with mocks.

### 2.5 Scalability

* Can be containerized and scaled with Docker/Kubernetes.
* Logic prepared to handle concurrent requests.

### 2.6 Reliability

* Critical validations on donation ensure data integrity.
* Fulfillment logic prevents overuse of donations.

## 4. Assumptions

* Hospitals are external and don’t need login.
* All cities are treated as flat distances (same vs different only).

This document outlines the system boundaries and ensures the solution meets both user and system expectations.