

Blood Bank Management and Request System for College Community

Shubham Mishra¹

¹ B.Tech Final Year, Dept. of Artificial Intelligence and Machine Learning, AIMT Lucknow, India

Guide: Dr. Krishna Nand Mishra, Asst. Professor, Dept. of AIML, AIMT Lucknow

Abstract - Blood availability during emergencies is something that most college campuses are completely unprepared for. Students and staff often struggle to find compatible donors in time, and there is rarely any organized system to track who has donated, who can donate, or who is in need. This paper describes a web-based Blood Bank Management and Request System that we built specifically to address this problem within a college community setting. The system lets donors register their blood type, log donations, and track their pledge status; it lets requesters search for compatible donors and submit emergency requests; and it gives administrators a dashboard to manage everything end-to-end. What makes this system different from generic blood bank tools is its built-in "receive-one, donate-one" pledge rule — if someone receives blood through the platform, they are expected to give back one unit or arrange for someone to do so on their behalf. This creates a self-sustaining cycle of donation culture within the institution. We built the platform using React.js for the frontend, Node.js and Express.js on the backend, and MySQL as the database, containerized with Docker for easy deployment. Testing across 47 functional test cases gave us a 97.9% pass rate. Under a load of 50 simultaneous users, all API endpoints responded within 500ms. A user acceptance study with 20 student volunteers returned an average satisfaction score of 4.4 out of 5. The system also passed security checks for SQL injection, XSS, CSRF, and brute-force attacks. Overall, we believe this system is a practical, affordable solution that any Tier-3 engineering college in India can deploy without needing expensive enterprise infrastructure.

Keywords: Blood Bank Management, Web Application, Donor Registry, Request Processing, Pledge System, Healthcare Information System, College Community, MySQL, Node.js, React.js, Role-Based Access Control, Docker.

I. INTRODUCTION

Blood is one of those resources that simply cannot be manufactured or substituted — and yet, in most Indian college campuses, there is no organized way to manage it. When a student or staff member faces a medical emergency that requires blood, the usual process is a frantic round of phone calls and WhatsApp messages, hoping that someone in the contact list happens to share the right blood type and is available to donate. It is reactive, unreliable, and often too slow.

This problem is not just an inconvenience. According to the National Blood Transfusion Council, India consistently falls short of its voluntary donation targets, particularly in non-metropolitan areas. NASSCOM has separately noted that over 60% of engineering graduates from Tier-3 colleges lack exposure to professionally managed digital systems — which suggests that building one could serve both a humanitarian and an educational purpose simultaneously.

We designed the Blood Bank Management and Request System to fill this gap at the institutional level. The idea is straightforward: give every member of a college community — students, faculty, and staff — a single platform where they can register as donors, search for blood by type, submit requests in emergencies, and track whether they have honoured their pledge to give back. The pledge rule is central to the system. Anyone who receives blood through the platform is obligated to either donate one unit themselves within 90 days or arrange for someone else to do so. Without this constraint, a request-only system would quickly exhaust its donor pool.

The key contributions of this work are as follows:

- A role-based web application with separate interfaces for Donors, Requesters, and Administrators, each with tailored functionality and access control.
- A searchable donor registry that supports filtering by blood type, department, and availability status for fast, targeted lookups.
- A request processing workflow that moves through defined states — PENDING, APPROVED, FULFILLED, REJECTED — with email notifications at each transition.
- An automated pledge enforcement subsystem that tracks obligation deadlines, sends reminders, and restricts repeat offenders from placing new requests.
- A real-time admin dashboard providing inventory-level visibility across all eight blood types registered in the system.

The rest of this paper is organized as follows. Section II reviews related work. Section III covers system design and requirements. Section IV describes the architecture. Section V explains the implementation of each module. Section VI presents our testing and evaluation results. Section VII discusses what worked, what did not,

and what we would do differently. Section VIII concludes with directions for future development.

II. RELATED WORK

To situate our contribution, we reviewed prior work across four areas: hospital-grade blood bank systems, community donation platforms, access control in healthcare applications, and the role of behavioral incentives in health technology.

A. Blood Bank Information Systems

Much of the existing work on blood bank digitization targets hospital environments with full clinical workflows. Ramesh et al. [1] built a centralized database system for hospital networks and demonstrated measurable reductions in wastage through synchronized inventory. Bharathi and Babu [2] went further by integrating mobile-based blood request flows directly into hospital management systems in Indian cities, showing a 34% faster request fulfillment rate. These are impressive results, but the systems themselves are too heavyweight for a college setting — they assume dedicated IT staff, medical-grade validation processes, and infrastructure that simply does not exist in most Tier-3 institutions.

B. Community-Based Donor Management Platforms

A smaller body of work focuses specifically on community-driven donation models, which are more directly relevant to our work. Arora et al. [3] studied voluntary donation programs at universities and found that peer-driven campaigns outperformed general public outreach by as much as 48% in terms of participation. Singh et al. [4] showed that a simpler, mobile-friendly registration process meaningfully increased first-time donor sign-ups. What neither of these systems includes, however, is any kind of structural obligation for recipients — they rely entirely on goodwill. Our pledge mechanism tries to go one step further by making reciprocity a system requirement rather than a social norm.

C. Healthcare Web Application Security and Access Control

Because our system handles personal health information — blood type, donation history, contact details — security was a non-negotiable design concern from the start. The role-based access control model we adopted follows the framework formalized by Sandhu et al. [5], which partitions data and operations according to organizational roles. Zhang and Joshi [6] applied this in electronic health record systems and found that RBAC reduced unauthorized access incidents by 91% relative to discretionary models. We implement a three-role variant — Donor, Requester, Administrator — enforced through JWT-based middleware on every protected API route.

D. Gamification and Behavioral Incentives in Health Platforms

The "receive-one, donate-one" pledge rule in our system is partly inspired by work on behavioral incentives in digital health platforms. Deterding et al. [7] defined gamification broadly as the use of game-design elements in non-game contexts to shape behavior — our pledge tracker fits this description. More specifically, Godin et al. [8] found that social reciprocity is one of the strongest predictors of repeat blood donation. Rather than just hoping users internalize this norm, we decided to build it into the platform as a hard constraint — if a user ignores two pledge deadlines, their ability to submit future requests is suspended until they comply.

III. SYSTEM DESIGN AND REQUIREMENTS

A. Requirements Specification

Before writing a single line of code, we conducted structured interviews with 45 students and 12 faculty members at AIMT Lucknow to understand what they actually needed — not just what we assumed they needed. We also reviewed published requirements from existing blood bank systems to avoid reinventing solutions to already-solved problems.

The resulting requirements break into three user categories with distinct needs. Donors need to register their blood type, log donations, check their availability status, and see whether any active pledges are outstanding. Requesters need to find compatible blood quickly, submit requests with urgency levels, and track request status without having to chase an administrator. Administrators need to approve or reject requests, validate donor records, view inventory summaries by blood type, and generate activity reports. On the non-functional side, we set a hard requirement of sub-500ms API response time under 50 concurrent users — a load level consistent with peak usage at a mid-sized college.

Table - I: System Requirements Summary

Requirement Category	Count	Priority
Functional (Donor)	8	High
Functional (Requester)	6	High
Functional (Admin)	9	High
Non-Functional (Performance)	3	Medium
Non-Functional (Security)	4	High
Non-Functional (Usability)	3	Medium

B. Database Schema Design

The database has seven core tables: Users, Donors, BloodRequests, Inventory, PledgeRecords, Notifications, and AuditLogs. The Users table carries a role column with three possible values — DONOR, REQUESTER, ADMIN — which the application layer reads to determine what a logged-in user is allowed to see and do. PledgeRecords link bidirectionally to BloodRequests (the event that triggers a pledge) and to Donors (the account that must fulfil it). Inventory is not a static table but an aggregate view computed from active donor profiles and open pledges, refreshed by database triggers on every relevant insert or update.

IV. SYSTEM ARCHITECTURE

We chose a three-tier architecture — presentation, business logic, and data — because it gives us clean separation between what the user sees, what the server computes, and what gets stored. Each layer can be updated or scaled without necessarily touching the others, which matters for a student-built project that will evolve over time.

A. Presentation Layer

The frontend is a React.js (v18) single-page application using React Router v6 for navigation and Tailwind CSS v3.4 for styling. We picked Tailwind because it produces consistent, mobile-responsive layouts without requiring custom CSS for every component. Axios v1.6 handles HTTP calls to the backend. Each user role gets a

different dashboard layout — the role is read from the JWT payload after login, and the appropriate UI is rendered accordingly. The server still validates authorization independently on every API call, so client-side rendering is cosmetic, not a security boundary.

B. Application Layer

The backend runs on Node.js v20 LTS with Express.js v4.18. We defined eleven route groups covering authentication, donor operations, requests, inventory queries, pledge management, and admin functions. Passwords are hashed with bcrypt at a work factor of 12 before being stored — we never persist plaintext credentials. Sessions are stateless: login returns a signed JWT containing userId, role, and a 24-hour expiry, with refresh token rotation handling re-authentication without full re-login.

C. Data Layer

We chose MySQL v8.0 as our primary database because ACID compliance matters here — a blood request fulfillment and its associated pledge creation need to either both succeed or both fail; partial writes are not acceptable. Sequelize ORM v6.35 handles all database interactions, providing model-level validation and a migration system that makes schema changes trackable and reversible. Connection pooling is configured with a minimum of 2 and maximum of 10 connections to handle concurrent request spikes without exhausting database resources.

D. Deployment Infrastructure

All three services — React frontend, Node.js backend, and MySQL — run in Docker containers orchestrated with Docker Compose. Nginx v1.25 sits in front as a reverse proxy, load balancer, and SSL terminator using Let's Encrypt certificates. The frontend is served as a static build via the Nginx container. This setup means the entire application can be deployed on a single cloud VM with one command, which is an important practical consideration for a college institution with limited IT staff.

V. IMPLEMENTATION MODULES

A. User Authentication and Role Management Module

Registration collects email, full name, contact number, blood type (donors only), password, and role selection. The password is hashed before the record is inserted — the raw password never touches the database. Login returns a JWT signed with a server-side secret; the token payload carries userId, role, and expiry. Every protected route runs two middleware checks: first, verify the token signature; second, confirm the role matches what the route permits. We rate-limit authentication endpoints to five attempts per IP per 15 minutes using express-rate-limit, which prevents brute-force attacks without disrupting normal use.

B. Donor Registry and Search Module

The donor search interface lets users filter by blood type (all eight ABO-Rh combinations), department, availability status, and time since last donation. Queries are built dynamically using Sequelize's Op.and and Op.or chaining so we never expose raw SQL to user input. Results are paginated server-side at 20 records per page by default. One thing worth highlighting: donor availability automatically flips to 'Unavailable' for 90 days after a logged donation event. This is not just a courtesy flag — it enforces the WHO-recommended safe interval between donations at the data layer, not just the UI.

C. Blood Request Processing Module

When a requester submits a blood request, they specify blood type, urgency level (Routine, Urgent, or Emergency), number of units, and any context notes the administrator might need. The request enters a PENDING state immediately. From there, an administrator can approve, reject, or mark it fulfilled. Each state transition fires an email via Nodemailer v6.9 through Gmail SMTP, so the requester is never left wondering what happened to their request. The admin-side queue supports filtering by status, urgency level, and blood type, which makes it manageable even when multiple requests are open simultaneously.

D. Pledge Enforcement Module

This module is the part of the system we are most proud of — and the part that has no direct equivalent in any related work we reviewed. When a blood request is marked FULFILLED, a PledgeRecord is automatically created and linked to the requester's account with a 90-day deadline. A cron job running daily at midnight IST scans for approaching and overdue pledges and fires reminder notifications. If a user lets two consecutive pledges expire without fulfilling them, the system adds a middleware check on their request submission endpoint that blocks new requests and redirects them to their pledge queue. Self-reported donation fulfilment updates the PledgeRecord status — a limitation we acknowledge, and one we plan to address through integration with external donor verification systems in a future version.

VI. EXPERIMENTAL RESULTS

A. Functional Testing Results

We wrote 47 test cases covering every feature in the system and ran them against a staging environment seeded with 500 synthetic donor profiles and 200 blood request records. The results are summarised in Table II.

Table - II: Functional Testing Summary

Module	Test Cases	Passed	Pass Rate
Authentication	8	8	100%
Donor Registry	10	10	100%
Blood Request	12	11	91.7%
Pledge System	9	9	100%
Admin Dashboard	8	8	100%
Overall	47	46	97.9%

The one failure was in the blood request module: under rapid-fire concurrent submissions from the same account, duplicate requests occasionally slipped through within a 500ms window before the uniqueness check fired. This is a known race condition. The fix — wrapping the check and insert in a database transaction with a row-level lock — is straightforward and is scheduled for the next release.

B. Performance Benchmarks

Load testing ran through Apache JMeter v5.6 with 50 virtual users sustained over five-minute periods. Every endpoint cleared the sub-500ms requirement we set at the requirements stage, as shown in Table III.

Table - III: API Endpoint Performance Benchmarks

API Endpoint	Avg. Response	95th Percentile
POST /api/auth/login	187 ms	312 ms
GET /api/donors/search	234 ms	445 ms
POST /api/requests/create	198 ms	389 ms
GET /api/inventory/summary	156 ms	298 ms
POST /api/pledges/fulfill	221 ms	412 ms

The donor search endpoint was the slowest, as expected — it builds a dynamic multi-parameter query at runtime. Adding a Redis caching layer for frequently-searched blood type combinations is an obvious next optimization that we did not have time to implement in this version.

C. Security Testing Results

We ran automated and manual security tests using SQLMap v1.7, OWASP ZAP, and custom scripts written to target our specific endpoints. Table IV summarises the outcomes.

Table - IV: Security Testing Results

Security Test	Method	Result
SQL Injection	SQLMap v1.7	Not Vulnerable
XSS Attack	OWASP ZAP	0 Issues Found
CSRF Protection	Manual Testing	Tokens Validated
JWT Tampering	Custom Script	Rejected (401)
Brute Force Auth	Custom Script	Blocked (429)

D. User Acceptance Testing

Twenty student volunteers from AIMT — split across Donor (n=10), Requester (n=7), and Administrator (n=3) roles — used the system over a five-day evaluation period with synthetic test data. They rated the system on five criteria after completing a set of prescribed tasks.

Table - V: UAT Results Summary

Evaluation Criterion	Score (1–5)
Ease of Registration	4.6
Donor Search Usability	4.4
Request Form Clarity	4.3
Admin Dashboard Utility	4.5
Pledge System Understanding	4.1
Overall Satisfaction	4.4

The pledge module scored lowest, which is understandable — it is the most conceptually novel part of the system and requires users to understand an obligation they have not encountered in other apps. The feedback suggested that clearer onboarding copy and a visual timeline of pledge deadlines would help. These are UI changes, not architectural ones, and we plan to incorporate them.

E. System Reliability Metrics

Table - VI: System Reliability Metrics (7-Day Evaluation)

Metric	Value
System Uptime	99.3%
Mean Time Between Failures	56.2 hours
Mean Time to Recovery	4.7 minutes
Total Successful API Calls	14,823
Overall Error Rate	0.71%

F. Key System Capabilities

Table - VII: Key System Capability Summary

Capability	Metric	Significance
Pledge Enforcement	100% test pass rate	Core novel feature
Role-Based Access Control	3 roles, 11 API groups	Security foundation
Blood Type Coverage	8 blood types	Full ABO+Rh coverage
Donor Auto-Availability Lock	90-day automatic block	WHO safety compliance
Concurrent User Support	50 users — stable	Institutional-scale load

VII. DISCUSSION

A. Performance Analysis

The 97.9% functional test pass rate is a result we are reasonably satisfied with. The one failure is a known concurrency edge case rather than a logic bug, and it occurs only under simultaneous duplicate submissions from the same account — a scenario that is unusual in normal use. The pledge system, which is the most complex module in terms of state management and scheduling, passed all nine of its test cases cleanly, which validates the core design.

On performance, the donor search endpoint's 234ms average response time is acceptable but not ideal. For a campus with several thousand registered donors, the dynamic query approach will start to strain as the dataset grows. Redis caching for static lookups like blood type frequency is a natural next step.

B. Comparative Analysis with Existing Systems

Table - VIII: Comparative Feature Analysis

Feature	This System	Hospital Systems	Generic Portals
College-Community Scope	Yes	No	No
Free & Open Deployable	Yes	No	No
Pledge Enforcement	Yes	No	No
Mobile Responsive	Yes	Partial	Partial
Admin Dashboard	Yes	Yes	Yes
Real-time Inventory	Yes	Yes	Limited
Student-Centric Design	Yes	No	No

C. Limitations

We want to be honest about where this system falls short. First, there is no integration with any external blood bank network — if the institutional supply runs dry, the platform cannot route a request elsewhere. Second, pledge fulfilment relies on self-reporting, which means a determined bad actor could mark a pledge complete without actually donating. Third, the notification system uses email, and students are notoriously bad at checking email; push notifications via Firebase would be more effective. Fourth, the system does not implement blood compatibility logic — it treats blood type as an exact match filter rather than using medical compatibility rules (e.g., O− as a universal donor). Adding this would make the search significantly more useful in practice. Fifth, our test dataset was entirely synthetic, so real-world edge cases from actual institutional data will inevitably surface bugs that our tests did not catch.

VIII. CONCLUSION

We built the Blood Bank Management and Request System because we genuinely believe that college campuses in India need it, and existing solutions are too expensive, too complex, or simply not designed with students in mind. The system we have delivered is a working, tested, secure, and deployable web application that covers the core use cases: donor registration, blood search, request processing, admin oversight, and pledge enforcement.

The numbers came out well — 97.9% test pass rate, sub-500ms response times under realistic load, 4.4/5 satisfaction in user testing, and clean security evaluation results. But the number we are most interested in is the pledge fulfilment rate once the system is deployed with real users, because that is the metric that will determine whether the donation cycle we designed for is actually sustainable.

Going forward, the most impactful additions would be real-time integration with state-level blood bank data through the NBTC API, blood compatibility logic in the search module, a React Native mobile app, Firebase push notifications, and eventually a machine learning layer that forecasts blood demand by type ahead of known high-risk periods like exam weeks and sports events. We hope this work serves as a useful foundation for that future development.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Dr. Krishna Nand Mishra, Assistant Professor, Department of Artificial Intelligence and Machine Learning, Ambalika Institute of Management and Technology, Lucknow, for his patient guidance, critical feedback, and consistent encouragement throughout this project. Without his direction this system would have been a much lesser thing. I also thank the management of AIMT for providing the infrastructure and environment needed to carry out this work. The open-source community behind React.js, Node.js, Express.js, MySQL, Sequelize, and Docker deserves acknowledgment as well — this project stands on the shoulders of their collective work.

REFERENCES

- [1] S. Ramesh, K. Srinivasan, and R. Prabha, "Design and Implementation of a Centralized Blood Bank Database Management System," *International Journal of Computer Applications*, vol. 97, no. 12, pp. 1–6, 2014.
- [2] M. Bharathi and S. Babu, "Mobile-Based Blood Request Management System for Hospitals," in *Proc. IEEE International Conference on Smart Structures and Systems*, Chennai, 2019, pp. 1–5.
- [3] S. Arora, S. Dembla, and V. Jain, "Voluntary Blood Donation in University Settings: A Study of Incentive Structures," *Journal of Blood Medicine*, vol. 11, pp. 89–97, 2020.
- [4] P. Singh, A. Sharma, and R. Verma, "Impact of Web-Based Donor Registration on First-Time Blood Donation Participation," *Transfusion Medicine*, vol. 30, no. 4, pp. 298–305, 2020.
- [5] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.
- [6] R. Zhang and R. Joshi, "RBAC in Healthcare: A Comparative Analysis of Access Control Models for EHR Systems," *Journal of Medical Systems*, vol. 35, no. 5, pp. 1249–1261, 2011.
- [7] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From Game Design Elements to Gamefulness: Defining Gamification," in *Proc. MindTrek*, 2011, pp. 9–15.
- [8] G. Godin, P. Sheeran, and M. Conner, "Determinants of Repeated Blood Donation Among New and Experienced Donors," *Transfusion*, vol. 45, no. 11, pp. 1607–1615, 2005.
- [9] A. Gupta, R. Agrawal, and M. Mehta, "Design of a Student Blood Donation Portal: A Case Study," in *Proc. IEEE ICACCI*, 2017, pp. 1–5.
- [10] V. Srikantaiah, "Emerging Trends in Blood Bank Management Systems Using Cloud Computing," *International Journal of Engineering Trends and Technology*, vol. 65, no. 3, pp. 121–126, 2018.
- [11] World Health Organization, "Blood Safety and Availability," WHO Fact Sheet, Geneva, 2023. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/blood-safety-and-availability>
- [12] National Blood Transfusion Council, "Annual Report on Blood Transfusion Services in India 2022–23," Ministry of Health and Family Welfare, New Delhi, 2023.
- [13] R. Pressman, *Software Engineering: A Practitioner's Approach*, 9th ed. McGraw-Hill Education, 2019.
- [14] A. Banks and E. Porcello, *Learning React: Modern Patterns for Developing React Apps*, 2nd ed. O'Reilly Media, 2020.
- [15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.