IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 19, April 2026
Impact Factor: 8.2

Blood Donation Portal Using Python

Jidon R and Avinash R
Department of Computer Applications,
VELS Institute of Science, Technology and Advanced Studies (VISTAS), Chennai, India
23105410@vistas.ac.in; 23105403@vistas.ac.in

Abstract: *In the modern healthcare sector, emergency response units face persistent challenges in maintaining accurate and accessible records for life-saving blood donations. Traditional manual donor coordination is prone to human error, leading to critical delays in response times and a lack of transparency during medical crises. To address these limitations, this project introduces the "HEART" (Hemorrhage Emergency Assistance & Response Technology) framework — a digital ecosystem designed to prioritize operational efficiency and donor-recipient integrity through a synchronized web architecture. The system replaces outdated local contact lists with a real-time, Google Sheets-driven identification and notification framework. Developed using Python Flask, CSS3, and Pandas-based data structures, the project leverages cloud-based CSV integration for instant donor record synchronization. Empirical evaluation confirms sub-2-second data latency, a 99% email delivery rate, and broad cross-browser UI compatibility. The application demonstrates an efficient integration of modern web technologies to solve real-world logistical challenges in blood-supply environments.*

Keywords: Blood Donation Portal; Emergency Response; Google Sheets Integration; Flask Framework; Glassmorphism UI; SMTP Automation; Medical Logistics; Real-time Synchronization

I. INTRODUCTION

The availability of blood is a critical factor in emergency medical care, yet the process of connecting hospitals with eligible donors remains largely fragmented. The World Health Organization estimates that approximately 118.5 million blood donations are collected globally each year, yet demand often outstrips supply at the local level due to systemic coordination failures rather than absolute scarcity. While digital transformation has penetrated many sectors of healthcare, blood bank management continues to rely on static registers, manual telephone outreach, and batch-updated legacy databases.

This paper proposes the HEART framework — a real-time blood donation portal designed to provide accurate and efficient identification of compatible donors, significantly reducing response times during medical emergencies. The system integrates cloud-hosted Google Sheets as a live data backend, Python Flask as the application server, and automated SMTP email notifications to create a fully functional donor-matching pipeline without the infrastructure overhead of a traditional relational database system.

Existing blood bank portals face three recurrent failure modes: (i) stale donor records arising from infrequent batch synchronization; (ii) manual notification workflows that introduce human latency; and (iii) high deployment complexity that limits adoption in under-resourced hospitals. The HEART framework resolves all three by leveraging universally accessible cloud infrastructure (Google Sheets), an industry-standard lightweight web framework (Flask), and configurable automated alerts (SMTP).

The framework was developed as a final-year undergraduate capstone project at Vels Institute of Science, Technology and Advanced Studies, Chennai. Beyond its immediate deployment utility, the system serves as a pedagogical reference implementation for students learning full-stack Python web development and data engineering with Pandas. The remainder of this paper is organized as follows: Section II reviews related literature; Section III describes the proposed system; Section IV details implementation; Section V presents results; Section VI discusses applications; and Section VII concludes.

DOI: 10.48175/IJARSCT-34441

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 19, April 2026 Impact Factor: 8.2

India presents a particularly acute case study for blood management challenges. The national demand for blood is approximately 12 million units per year, yet only 9 million units are collected annually, leaving a recurring deficit of 3 million units. A significant proportion of this shortfall arises not from absence of willing donors, but from the inability of health facilities to locate and mobilize compatible donors quickly enough during emergencies. Road trauma, obstetric hemorrhage, and surgical complications frequently require transfusions within a 30-minute critical window — a timeframe that manual outreach systems routinely fail to meet.

Digital health interventions have demonstrated measurable impact in related domains. Telemedicine platforms reduced specialist consultation waiting times by over 60% in rural health centres across India. Mobile-based appointment systems improved vaccination coverage rates by 23% in controlled trials conducted across Tamil Nadu districts. These precedents establish strong evidence that technology-enabled coordination improvements translate directly to improved health outcomes, providing a compelling motivational basis for the HEART framework.

The specific contributions of this work are threefold. First, a cloud-integrated donor-matching algorithm that operates without a dedicated database server, reducing deployment barriers for under-resourced facilities. Second, an automated multi-recipient SMTP notification engine with delivery tracking and configurable retry logic. Third, a Glassmorphism-styled, mobile-responsive user interface that achieves Google Lighthouse performance scores above 88 on low-powered mobile devices — the dominant access channel in the target deployment environment across semi-urban India.

II. LITERATURE REVIEW

Blood bank digitization has been an active research area for over two decades, with solutions ranging from simple spreadsheet-based registers to enterprise-level health information systems. Vijayalakshmi et al. [1] surveyed fifteen existing blood bank management systems and concluded that the majority relied on proprietary SQL schemas that required dedicated database administrators, creating a significant operational burden for smaller hospitals. Their work highlighted the need for lightweight, maintainable alternatives that can be deployed without specialized IT staff.

Grinberg [2] demonstrated Flask's versatility in rapid-prototyping medical web applications, establishing it as a mature choice for healthcare portals that demand low latency with minimal server footprint. McKinney [3] formalized Pandas as a statistical computing framework, and subsequent research has confirmed its suitability for in-memory data pipelines processing record sets up to several hundred thousand rows — well within the requirements of regional donor registries.

Singh and Kumar [4] explored SMS-based notification systems for blood requests and found that automated outreach reduced average donor response time by 47% compared to manual telephone campaigns. However, their system required a proprietary SMS gateway with recurring subscription costs, limiting its applicability in resource-constrained settings. The HEART framework achieves comparable automation using SMTP email, which is universally accessible through free provider accounts.

Google Sheets as a backend has been explored in several IoT and logistics contexts. Patil et al. [5] demonstrated a real-time sensor data pipeline using Google Sheets CSV export and Python, achieving update latencies under 3 seconds on standard broadband connections. The HEART system adapts this pattern to donor record management, substituting sensor telemetry with structured donor registration data. The approach eliminates the need for database provisioning while maintaining data accessibility through familiar spreadsheet interfaces.

Glassmorphism as a UI design paradigm was formally articulated by Michal Malewicz in 2020 and has gained traction in healthcare UX design for its emphasis on visual clarity and depth cues that reduce cognitive load during high-stress clinical interactions. Raza et al. [6] found that clinicians preferred Glassmorphism-styled dashboards over flat design counterparts in usability studies, attributing the preference to improved figure-ground separation in high-ambient-light environments such as operating theatres.

The use of serverless and spreadsheet-backed architectures in mission-critical applications has been explored in disaster relief logistics, supply chain management, and community health surveillance. Alibek et al. [7] documented a COVID-19 contact tracing system built on Google Sheets that was deployed across 14 health districts in Kazakhstan within 48

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 19, April 2026
Impact Factor: 8.2

hours of regulatory approval, demonstrating the viability of spreadsheet-backed systems in urgent public health contexts. This precedent directly informs the HEART design philosophy of prioritising rapid deployment over architectural perfection.

Prior work on automated donor notification has been constrained by the high cost and complexity of SMS gateway integration. Emerging research on SMTP-based notification systems indicates that email remains a high-penetration communication channel in urban and semi-urban Indian populations, with smartphone email client usage exceeding 78% among adults aged 18-45 — precisely the core donor demographic. This demographic reality makes email-based notification a pragmatic and cost-effective alternative to proprietary SMS solutions, reinforcing the HEART system's design choice.

A notable gap in the existing literature is the absence of end-to-end open-source implementations that combine live data synchronization, automated notification, and mobile-responsive UI in a single deployable package. Most published systems describe either the data layer or the notification component in isolation, leaving practitioners to integrate them manually. The HEART framework addresses this integration gap by providing a complete, tested, and documented reference implementation released under the MIT licence.

III. PROPOSED SYSTEM

A. System Architecture

The HEART system is structured as a four-module web application, each module responsible for a distinct functional domain. The modular decomposition ensures that individual components can be upgraded, replaced, or extended without disrupting the remainder of the system. Communication between modules follows a request-response paradigm mediated by Flask's URL routing layer.

The User Interface Module provides an "Obsidian" Glassmorphism-themed frontend for donor registration and blood group search. The interface is constructed with HTML5 semantic markup, CSS3 animations, and vanilla JavaScript for client-side form validation. The design prioritizes mobile responsiveness through CSS Grid and Flexbox layouts, targeting the low-powered Android devices prevalent in Indian clinical settings.

The Data Synchronization Module fetches the latest donor CSV from a publicly accessible Google Sheets export URL upon each incoming query. A Pandas DataFrame is instantiated in-memory from the CSV payload, and subsequent filtering, deduplication, and compatibility matching operations are executed as vectorized DataFrame operations. This approach eliminates round-trip database query latency and allows the system to function on servers without persistent storage.

The Notification Engine dispatches personalized SMTP email alerts via Flask-Mail whenever a compatible donor is identified. Each notification includes the requesting hospital's name, address, contact number, urgency level, and a one-click opt-out link. The engine implements exponential backoff retry logic with a maximum of three delivery attempts to accommodate transient SMTP server failures.

The Hospital Network Module maintains a static JSON registry of regional medical centers, searchable by name, city, and pincode. Hospital records include contact details and the current blood inventory status, which is updated by authorized hospital administrators through a protected Flask endpoint authenticated via session-based tokens.

C. Data Flow and State Management

The HEART system follows a stateless server architecture: each HTTP request is processed independently, with all session state maintained in Flask's server-side session store (cookie-backed, cryptographically signed). The donor DataFrame is the only application-level state, held in a module-level cache variable that is thread-safe due to Python's Global Interpreter Lock (GIL) for CPython. This architectural choice trades memory efficiency for query speed, a reasonable trade-off given that regional donor datasets are unlikely to exceed 100,000 records in the foreseeable deployment horizon.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 19, April 2026 Impact Factor: 8.2

Incoming HTTP requests follow a standardized processing pipeline. The routing layer dispatches the request to the appropriate Flask view function based on URL pattern and HTTP method. The view function validates request parameters, triggers a cache refresh if the TTL has expired, executes the matching algorithm on the cached DataFrame, dispatches SMTP notifications, appends an audit record, and returns a JSON response to the client. The entire pipeline executes synchronously within a single request-response cycle, ensuring deterministic latency without the complexity of asynchronous task queues.

D. Frontend Architecture

The Glassmorphism UI is implemented as a set of server-rendered Jinja2 templates served by Flask. The design employs a frosted-glass aesthetic achieved through CSS backdrop-filter: blur() and semi-transparent background-color values, creating visual depth that aids element separation on complex backgrounds. All interactive elements — search forms, registration modals, and result cards — are styled with consistent 12px border-radius and 1px rgba border to reinforce the design language.

Animate.css 4.1 provides entrance animations for search results, applying fadeInUp transitions to result cards as they are dynamically injected into the DOM via JavaScript fetch() calls. This progressive rendering approach prevents layout shifts and provides users with immediate visual feedback that the query is being processed, improving perceived responsiveness. Form validation is implemented client-side using the Constraint Validation API with custom error message styling consistent with the Glassmorphism design system.

The operational workflow proceeds through five sequential stages. During System Initialization, the Flask server fetches the current donor CSV from Google Sheets and caches it as a Pandas DataFrame. The cache is invalidated on a configurable time-to-live (default: 60 seconds) to balance freshness with server load.

In the Donor Discovery phase, hospital staff submit a blood group query and city preference through the web interface. The Matching phase executes a two-pass DataFrame filter: the first pass applies the ABO/Rh compatibility matrix to identify compatible blood groups; the second pass restricts results to donors within the specified city. Results are ranked by last donation date to prioritize long-tenure donors.

The Notification phase automatically dispatches SMTP email alerts to all matched donors within the city, with a configurable upper limit (default: 10 donors) to prevent inbox flooding. The Audit phase records each query-notification cycle — including timestamp, requesting hospital, blood group sought, and number of donors contacted — to a time-stamped audit CSV for regulatory compliance and longitudinal analytics.

Fig. 1. HEART system architecture: four-module decomposition and inter-module data flows

Fig. 2. Operational workflow: five-stage process from blood request initiation to audit logging

IV. IMPLEMENTATION

A. Technology Stack

Python 3.10 serves as the primary runtime. Flask 2.3 provides WSGI routing and session management. Pandas 2.0 handles all DataFrame operations. Flask-Mail 0.9 integrates with standard SMTP servers. The frontend uses HTML5, CSS3 (Glassmorphism), and vanilla JavaScript. Google Sheets CSV export endpoints provide the live data backbone. Animate.css 4.1 supplies entrance animations for UI components. No relational database system is required; all persistent state is stored in Google Sheets and the audit CSV.

B. Blood Group Compatibility Matrix

The matching algorithm encodes ABO/Rh compatibility as a Python dictionary keyed by recipient blood group, with values listing all compatible donor blood groups. For example, a recipient with blood group AB+ can receive from all eight ABO/Rh types, while an O− recipient can only receive O− donations. This matrix is applied as a Pandas isin() filter on the donor DataFrame, guaranteeing medically correct matches without external library dependencies.

A secondary compatibility tier handles component-specific requirements: packed red cells, fresh frozen plasma, and platelet concentrate each have distinct compatibility rules that are encoded as separate filter passes. This extensibility

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 19, April 2026
Impact Factor: 8.2

ensures the system can evolve from whole-blood matching to component-therapy matching without architectural changes.

C. Dataset

TABLE I. BLOOD DONOR DATASET SUMMARY

Dataset Type	Description	Records
Common Groups	A+, B+, O+, AB+ — high-volume donor pool	20+
Rare Groups	A2B1+, O−, AB− — critical shortage categories	3+
Total Donors	All ABO/Rh types synced from Google Sheets CSV	30+

The dataset is maintained as a Google Sheet with columns for donor ID, full name, blood group, ABO/Rh subtype, contact email, city, last donation date, and availability flag. The sheet is published as a CSV endpoint via Google Sheets' built-in export feature. No API key is required for publicly shared sheets, eliminating credential management overhead. Dataset integrity is enforced through Google Forms-based donor registration, which applies server-side input validation before writing to the sheet.

D. Security Considerations

SMTP credentials and Google Sheets URL are stored in OS environment variables and loaded at runtime via Python's os.environ module, ensuring they are never committed to the source repository. Hospital administrator endpoints are protected by Flask session tokens issued upon username/password authentication against a locally stored bcrypt-hashed credential file. Donor personal data (name, email) is never exposed in client-side responses; the matching output displays only donor blood group and city to hospital staff, with email dispatch handled server-side.

E. Donor Registration Flow

Prospective donors access the registration portal via a publicly accessible URL. The registration form collects name, email address, blood group (selected from a validated dropdown), city, and voluntary availability status. Client-side JavaScript validates email format and ensures all mandatory fields are populated before submission. On the server side, Flask validates the POST payload using WTForms validators, appending validated records to the Google Sheet via the Sheets API write endpoint. Duplicate email addresses are rejected through a server-side uniqueness check against the current DataFrame.

The registration confirmation email includes a personalized acknowledgment, the donor's assigned unique ID, and a one-click link to update availability status. This opt-in/opt-out mechanism is essential for maintaining dataset accuracy: donors who are temporarily ineligible (due to recent donation, travel, or illness) can mark themselves unavailable, preventing spurious notifications and preserving donor trust. Availability status changes are reflected in the Google Sheet immediately, ensuring the live matching engine always operates on current data.

F. Error Handling and Resilience

The HEART system implements a three-tier error handling strategy. At the network layer, failed Google Sheets CSV fetch requests are retried up to three times with exponential backoff (1 s, 2 s, 4 s delays). If all retries fail, the system serves the most recently cached DataFrame, which may be up to the configured cache TTL (60 s) old, and logs a warning to the audit file. At the application layer, malformed CSV rows (missing columns, invalid blood group values) are filtered out during DataFrame construction and flagged in a separate error log for administrator review. At the

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 19, April 2026
Impact Factor: 8.2

notification layer, SMTP delivery failures trigger retry attempts using the backoff strategy described above, with undeliverable addresses quarantined after three consecutive failures to prevent repeated futile delivery attempts.

V. RESULTS AND DISCUSSION

The HEART system was evaluated across four dimensions: data synchronization latency, notification delivery reliability, user interface performance, and comparative feature parity. All benchmark tests were conducted on a consumer-grade laptop (Intel Core i5-1135G7, 8 GB RAM, 50 Mbps broadband) simulating a typical small-hospital IT environment. One hundred query-notification cycles were executed across five distinct time windows to capture variability introduced by Google Sheets server load.

TABLE II. SYSTEM PERFORMANCE EVALUATION

Metric	Value	Remarks
Data Sync Latency	< 2 s	Cloud CSV polling
Email Delivery Rate	99 %	SMTP via Flask-Mail
UI Load Time	< 1.5 s	CSS3 Glassmorphism
Concurrent Users	50+	Flask threaded mode
Matching Accuracy	100 %	ABO/Rh matrix

Data synchronization latency was measured as the elapsed time between a query submission and the return of a filtered donor list. The mean latency across 100 cycles was 1.73 seconds (σ = 0.21 s), with a 99th-percentile value of 1.98 seconds. No cycle exceeded the 2-second threshold. The primary latency contributor was Google Sheets CSV export (~1.2 s on average), with Pandas filtering accounting for the remaining ~0.5 s.

Email delivery was tested by dispatching 500 notifications to a pool of ten controlled test email accounts across Gmail, Outlook, and Yahoo Mail. The overall delivery rate was 99.0%, with five messages classified as spam by the Yahoo Mail filter. All Gmail and Outlook deliveries were confirmed in the primary inbox. The 1% spam-classification rate was mitigated in the production deployment by registering the sending domain with major email reputation services (SPF, DKIM records).

UI performance was assessed using Google Lighthouse on Chrome 124 across three device profiles: desktop (1920×1080), tablet (768×1024), and mobile (390×844). Performance scores were 94, 91, and 88 respectively, reflecting minor render-blocking from Animate.css on mobile. This was resolved in a subsequent patch by lazy-loading the animation library.

TABLE III. COMPARISON WITH EXISTING BLOOD BANK SYSTEMS

Feature	HEART	eBloodBank	LifeCare	BloodConnect
Live Sync	Yes	No	Partial	No
Auto Notify	SMTP	SMS only	None	Manual
DB Required	No	MySQL	PostgreSQL	SQLite
Deploy Time	< 30 min	> 4 hrs	> 3 hrs	> 2 hrs
Mobile UI	Yes	No	Basic	No

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 19, April 2026 Impact Factor: 8.2

Comparative analysis (Table III) confirms that HEART is the only system among those surveyed to achieve real-time data synchronization without requiring a dedicated database server. The 30-minute deployment time represents a 4–8× reduction compared to competing systems, making it tractable for deployment by hospital IT officers without specialist database administration skills.

VI. APPLICATIONS

A. Software Applications

The HEART framework can be extended to adjacent healthcare coordination tasks with minimal code modifications. By substituting the blood group compatibility matrix with organ compatibility rules (HLA typing, cross-match results), the system can serve as a lightweight organ donor registry for local transplant programs. Integration with hospital management systems via Flask's RESTful API endpoints enables automated blood request escalation without human intervention, reducing the administrative burden on nursing staff during emergencies.

The SMTP notification engine can be replaced with an SMS gateway (e.g., Twilio, MSG91) to accommodate donors without reliable email access, broadening reach in semi-urban and rural settings. A Progressive Web App (PWA) wrapper around the existing frontend would enable offline donor registration with background sync, addressing connectivity gaps in areas with intermittent internet access.

B. Practical Healthcare Uses

Beyond emergency transfusion support, the portal can facilitate scheduled blood donation drives by identifying donors whose last donation date exceeds the minimum inter-donation interval (56 days for whole blood, 112 days for double red cells). Automated pre-drive notification campaigns can be dispatched up to one week in advance, improving drive attendance rates. Hospitals conducting pre-operative planning can use the system to pre-screen and reserve donors for elective surgeries requiring rare blood types, reducing last-minute shortages and associated surgical cancellations.

The audit log generated by the HEART system provides a longitudinal record of donor engagement that can inform blood bank inventory management. Analysis of historical query patterns enables blood banks to anticipate demand spikes — for example, before major festivals or sporting events — and proactively recruit donors through targeted campaigns.

C. Regional Health Administration

District and state-level health administrators can deploy HEART as a federated network of facility-level instances, each managing its own donor registry while exposing a standardized REST API endpoint for cross-facility queries. When a facility exhausts its local donor pool for a rare blood type, the system can automatically escalate the query to neighbouring facilities' HEART instances, effectively creating a regional blood donor network without the infrastructure overhead of a centralized national registry.

The audit logs generated across federated instances can be aggregated at the district health office level to produce heatmaps of blood demand by type, location, and time period. These analytics dashboards, generated using Pandas and Matplotlib, provide public health planners with actionable data for targeted donor recruitment campaigns, mobile donation camp scheduling, and inter-facility blood transfer logistics optimization.

D. Disaster and Mass Casualty Response

The HEART system's sub-2-second query response makes it particularly well-suited to mass casualty incident (MCI) management, where multiple simultaneous transfusion requests across different blood groups must be processed concurrently. The system's multi-match capability — returning up to a configurable number of donors per query — allows incident command centres to mobilize donor pools in parallel rather than sequentially. Integration with the National Disaster Management Authority's emergency alert infrastructure would further amplify the system's reach during large-scale events such as industrial accidents, train derailments, or natural disasters.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 19, April 2026
Impact Factor: 8.2

E. Academic and Research Applications

The HEART system serves as a pedagogical reference implementation for undergraduate students studying Python web development, Pandas data engineering, and cloud-integrated application design. Its modular architecture, comprehensive inline documentation, and zero-cost infrastructure requirements make it suitable as a capstone project template in resource-limited academic settings. The matching algorithm and dataset schema are also applicable to published research in healthcare informatics, humanitarian logistics optimization, and emergency response system design.

VII. CONCLUSION

This paper presented the HEART (Hemorrhage Emergency Assistance & Response Technology) framework, a cloud-integrated web portal for real-time blood donor identification and automated notification. By combining Python Flask, Google Sheets as a live backend, Pandas for in-memory data processing, and SMTP-based notification, the system addresses the critical latency and transparency gaps that characterize conventional blood bank management. Empirical evaluation confirmed a mean data synchronization latency of 1.73 seconds, a 99% email delivery rate, and Lighthouse performance scores above 88 across all device profiles.

Comparative benchmarking against three existing open-source blood bank systems demonstrated that HEART achieves superior real-time data fidelity and significantly lower deployment complexity, making it accessible to hospitals without dedicated database infrastructure. The absence of a relational database requirement reduces the total cost of ownership to near-zero for institutions with existing Google Workspace accounts.

A. Latency Breakdown Analysis

A detailed latency breakdown revealed three contributors: (i) Google Sheets CSV export, which accounted for 69% of total query latency (mean 1.19 s); (ii) Pandas DataFrame construction from the CSV payload, accounting for 17% (mean 0.29 s); and (iii) ABO/Rh compatibility filtering and city-level secondary filter, accounting for the remaining 14% (mean 0.24 s). The dominant contributor — Google Sheets export latency — is outside the system's direct control, but is amenable to partial mitigation through caching. With a 60-second cache TTL, repeat queries within the cache window experience a mean latency of 0.24 s (filtering only), representing an 86% latency reduction for high-frequency query patterns typical of major emergency departments.

B. Scalability Assessment

Scalability was evaluated by progressively increasing the simulated donor registry size from 30 records to 10,000 records using synthetically generated data conforming to the live dataset schema. Pandas filtering latency remained below 0.5 seconds for all dataset sizes up to 10,000 records, confirming that the in-memory DataFrame approach is adequate for regional-scale deployments. For national-scale registries exceeding 100,000 records, the authors recommend migrating the data layer to a lightweight SQLite or PostgreSQL backend while retaining all other system components unchanged — a migration requiring modification of only the data access layer.

Concurrent user load testing, conducted using Locust 2.0 with 50 simulated simultaneous users submitting queries, revealed no request failures and a 95th-percentile response time of 2.3 seconds. Flask's threaded request handling was sufficient for this load level. For higher concurrency requirements (100+ simultaneous users), deployment behind a Gunicorn WSGI server with four worker processes is recommended, which preliminary testing indicated would support up to 200 concurrent users without degradation.

Future work will explore three primary enhancement vectors. First, integration with national blood bank registries — specifically, India's eRaktKosh platform — to broaden donor discovery beyond locally registered individuals. Second, machine learning-based demand forecasting using historical audit log data to pre-position blood stock across hospital networks before anticipated shortages. Third, Progressive Web App capabilities to support fully offline donor registration in areas with intermittent connectivity, with background synchronization upon network restoration.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 19, April 2026 Impact Factor: 8.2

Additional planned enhancements include multi-language UI support to improve accessibility across Tamil Nadu's linguistically diverse user population, integration with the Aadhaar-linked DigiLocker system for donor identity verification, and a real-time dashboard for district blood bank administrators to monitor donation activity and inventory levels across all affiliated facilities. The framework is released as open-source software under the MIT licence to encourage community-driven development and wider deployment across resource-constrained medical environments globally.

ACKNOWLEDGMENT

The authors thank Vels Institute of Science, Technology and Advanced Studies (VISTAS), Chennai, for providing laboratory facilities, computational resources, and academic guidance throughout this project. Special thanks to the Department of Computer Science faculty — particularly the project coordinator and external reviewers — for their constructive feedback during system design, implementation, and evaluation. The authors also acknowledge the volunteer donors who participated in the usability testing phase and provided invaluable feedback on the registration workflow and notification design.

REFERENCES

[1]. S. Vijayalakshmi, R. Priya, and M. Anand, "A Survey on Blood Bank Management Systems: Challenges and Solutions," Int. J. Health Informatics, vol. 12, no. 3, pp. 45–58, 2021.

[2]. M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.

[3]. W. McKinney, "Data Structures for Statistical Computing in Python," in Proc. 9th Python in Science Conf. (SciPy), Austin, TX, USA, 2010, pp. 51–56.

[4]. A. Singh and R. Kumar, "SMS-Based Automated Donor Notification System for Emergency Blood Requirements," in Proc. IEEE Int. Conf. Healthcare Informatics (ICHI), Orlando, FL, USA, 2022, pp. 210–218.

[5]. V. Patil, S. Desai, and A. Kulkarni, "Real-Time IoT Data Pipeline Using Google Sheets and Python," in Proc. Int. Conf. Internet of Things (IoT), Pune, India, 2023, pp. 87–93.

[6]. M. Raza, T. Hussain, and F. Ali, "Glassmorphism vs. Flat Design: A Usability Study in Clinical Dashboard Environments," J. Usability Studies, vol. 18, no. 2, pp. 34–49, 2023.

[7]. D. Alibek, A. Seitkali, and N. Ospanova, "Rapid Deployment of a Spreadsheet-Backed Contact Tracing System During the COVID-19 Pandemic," J. Public Health Informatics, vol. 13, no. 1, e5, 2021.

[8]. [Flask Documentation, "Flask Web Framework," Pallets Projects, 2024. [Online]. Available: https://flask.palletsprojects.com

[9]. Google Developers, "Google Sheets API v4 Reference," 2024. [Online]. Available: https://developers.google.com/sheets/api

[10]. Animate.css Contributors, "Animate.css — A Cross-browser Library of CSS Animations," 2024. [Online]. Available: https://animate.style

[11]. National Blood Transfusion Council of India, "Annual Report on Blood Services," Ministry of Health and Family Welfare, New Delhi, India, 2023.

[12]. M. Malewicz, "Glassmorphism in User Interfaces," UX Collective, 2020. [Online]. Available: https://uxdesign.cc/glassmorphism-in-user-interfaces