🩸 BLOODCHAIN - SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Document Information

- **Project Name:** Bloodchain - Decentralized Blood Donation Platform
- **Version:** 1.0.0
- **Date:** November 2025
- **Status:** Production Ready
- **Document Type:** Software Requirements Specification

EXECUTIVE SUMMARY

Bloodchain is a decentralized, AI-powered blood donation platform built on Next.js with blockchain integration. It eliminates hospital intermediaries, automates donor-recipient matching using machine learning, and ensures transparent, trustless transactions through smart contracts.

Key Objectives:

- ✅ Eliminate hospital dependency for blood management
- ✅ Automate matching using AI/ML algorithms
- ✅ Ensure identity verification through biometric analysis
- ✅ Tokenize rewards and create NFT credentials
- ✅ Decentralize verification through community peer networks
- ✅ Provide real-time geolocation-based matching

1. INTRODUCTION

1.1 Purpose

This document specifies functional and non-functional requirements for Bloodchain, a next-generation blood donation platform that leverages artificial intelligence, blockchain technology, and peer-to-peer networks to revolutionize blood donation management.

1.2 Scope

The platform includes:

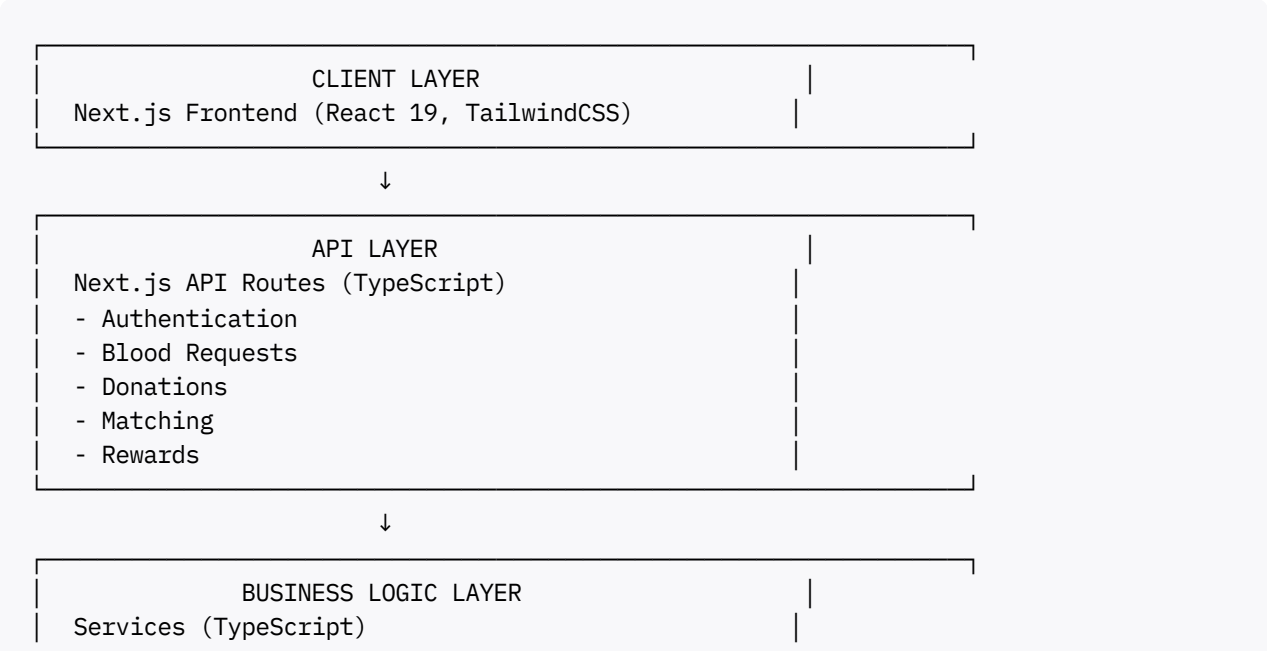
- User authentication and profile management
- AI-powered donor-recipient matching
- Biometric identity verification
- Blockchain-based transaction recording
- Token rewards and NFT credentials
- Real-time notifications and geolocation tracking
- Peer verification network
- Admin dashboard and analytics

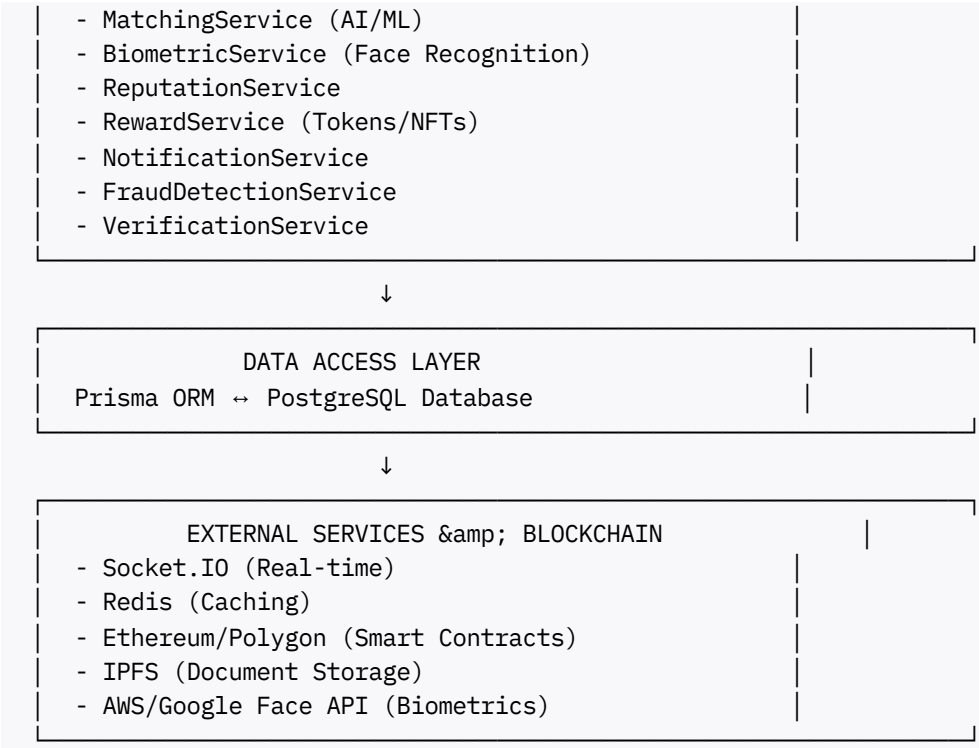
1.3 Definitions

- **Donor:** User providing blood units
- **Recipient:** User requesting blood units
- **Verifier:** Community member verifying transactions
- **Smart Contract:** Automated blockchain agreements
- **AI Matching:** Machine learning-based donor selection
- **Biometric Verification:** Face recognition and liveness detection
- **NFT:** Non-fungible token representing achievements

2. SYSTEM OVERVIEW

2.1 Architecture





2.2 Technology Stack

Layer	Technology
Frontend	Next.js 14, React 19, TypeScript, TailwindCSS
Backend	Next.js API Routes, Node.js
Database	PostgreSQL, Prisma ORM
Real-time	Socket.IO, Redis
AI/ML	TensorFlow.js, Node.js ML libraries
Blockchain	Solidity, Ethereum/Polygon, Hardhat
Authentication	NextAuth.js, JWT
Storage	IPFS, AWS S3
DevOps	Docker, Docker Compose, Vercel

3. FUNCTIONAL REQUIREMENTS

3.1 User Management

FR-3.1.1: User Registration

- **Description:** Users can register as donors or recipients
- **Actors:** Guest User
- **Preconditions:** User has valid email and password
- **Main Flow:**
    1. User provides email, password, name, phone, role
    2. System validates inputs
    3. System hashes password with bcrypt
    4. System creates user record in database
    5. System sends verification email
- **Postcondition:** User account created with PENDING verification status

FR-3.1.2: Email Verification

- **Description:** User verifies email via confirmation link
- **Actors:** Registered User
- **Main Flow:**
    1. User clicks verification link in email
    2. System validates token
    3. System updates verification status to VERIFIED
- **Error Handling:** Invalid/expired tokens rejected

FR-3.1.3: User Profile Creation

- **Description:** Complete profile after registration
- **Actors:** Verified User
- **Main Flow (Donor):**
    1. User selects blood type, Rh factor
    2. User enters location (city, state, coordinates)
    3. System creates DonorProfile record
    4. System enables matching capability
- **Main Flow (Recipient):**
    1. User enters medical history (encrypted)
    2. User specifies preferred blood types
    3. System creates RecipientProfile record
    4. System enables request creation

3.2 Blood Request Management

FR-3.2.1: Create Blood Request

- **Description:** Recipients create blood requests
- **Actors:** Recipient User
- **Main Flow:**
    1. User selects blood type, Rh factor, units needed
    2. User selects urgency level (LOW, MEDIUM, HIGH, CRITICAL, EMERGENCY)
    3. User provides location coordinates
    4. User uploads medical proof (encrypted, stored on IPFS)
    5. System creates BloodRequest record
    6. System triggers autonomous AI matching
    7. System broadcasts urgent notifications (if EMERGENCY)
- **Postcondition:** Request created with OPEN status, AI matching started

FR-3.2.2: Auto-expire Requests

- **Description:** Requests automatically expire after 24 hours
- **Background Job:** Runs every 1 hour
- **Main Flow:**
    1. Check all OPEN requests
    2. If expiresAt < now, mark as EXPIRED
    3. Notify recipient of expiration
    4. Release any pending matches

3.3 AI Matching System

FR-3.3.1: Autonomous Donor Matching

- **Description:** AI automatically matches donors to requests
- **Trigger:** Blood request creation OR background job (every 5 minutes)
- **Algorithm:**
    - Feature Extraction (8 dimensions):
        1. Blood type compatibility: 1.0 (pre-filtered)
        2. Distance score: 1 - (distance / 50km)
        3. Donor reputation: min(reputation / 1000, 1)
        4. Availability: 1.0 if available, 0 if not

5. Urgency multiplier: 0.7-1.0 based on urgency level
6. Success rate: completed_donations / total_donations
7. Response time: 1 - (response_time / 3600s)
8. Fraud risk: 1 - min(fraud_score / 100, 1)

- Neural Network:
  - Input: 8-dimensional feature vector
  - Hidden layers: 64 → 32 → 16 units (ReLU activation)
  - Batch normalization and dropout (0.3, 0.2)
  - Output: Sigmoid (match probability 0-1)
  - Threshold: Score > 0.65
- Top matching:
  - Select top 10 matches by score
  - Create RequestMatch records (1-hour expiration)
  - Notify donors via email/push notification

FR-3.3.2: Emergency Fast-Track

- **Description:** EMERGENCY requests bypass standard matching, broadcast to all active donors
- **Main Flow:**
  1. Detect urgency = EMERGENCY
  2. Skip ML scoring (use all available donors)
  3. Broadcast emergency alert to 500km radius
  4. Send SMS + push notifications to nearby donors
  5. Accept first confirmed donor

3.4 Biometric Verification

FR-3.4.1: Face Recognition Registration

- **Description:** Donor captures face for identity verification
- **Actors:** Donor User
- **Main Flow:**
  1. User captures face image via webcam/mobile camera
  2. System extracts 128-dimensional face embedding
  3. System performs liveness detection
  4. System detects spoofing attempts
  5. System hashes embedding for storage

6. System stores hash (not raw embedding) in database
7. Update DonorProfile: biometricVerified = true

- **Liveness Detection Checks:**
  - Eye blinking detection
  - Head movement variance
  - Micro-expressions
  - Texture analysis
- **Spoof Detection Checks:**
  - Print detection
  - Video playback detection
  - Reflectance patterns
  - Frequency analysis
- **Success Criteria:** Liveness > 0.85, Spoof < 0.3

FR-3.4.2: Face Verification on Login

- **Description:** Verify user identity during sensitive operations
- **Trigger:** Donation completion, reward claiming
- **Main Flow:**
  1. User captures face image
  2. System extracts embedding
  3. System calculates cosine similarity with stored hash
  4. System checks liveness and spoof scores
  5. Verify if similarity > 0.85 AND liveness > 0.85 AND spoof < 0.3
- **Error Handling:** Prompt retry or fallback to OTP verification

3.5 Donation Process

FR-3.5.1: Accept Match

- **Description:** Donor accepts a matched blood request
- **Actors:** Donor User
- **Main Flow:**
  1. Donor receives match notification
  2. Donor reviews request details
  3. Donor clicks "Accept Match"
  4. System creates initial Donation record

5. System transitions RequestMatch to ACCEPTED
6. System notifies recipient
7. System activates geolocation tracking

- **Time Limit:** Donor has 1 hour to accept before match expires

FR-3.5.2: Record Donation

- **Description:** Record completed donation on blockchain
- **Actors:** Donor User
- **Precondition:** Donation accepted and completed
- **Main Flow:**
  1. Donor uploads proof of donation (photos/receipt via IPFS)
  2. System triggers P2P peer verification (3 verifiers required)
  3. Verifiers independently attest via blockchain signatures
  4. System creates multi-signature smart contract call
  5. Smart contract records donation immutably
  6. System updates Donation status to COMPLETED
  7. System automatically issues rewards
- **Smart Contract Call:**

```
recordDonationWithAutoVerification(
  requestId,
  donor,
  unitsCollected,
  ipfsProof,
  verifiersArray,
  signaturesArray
)
```

- **Postcondition:** Blockchain-verified donation with tokens + NFT issued

FR-3.5.3: Donation History

- **Description:** Donors and recipients can view donation history
- **Main Flow:**
  1. User navigates to "History" tab
  2. System queries all donations for user
  3. System displays with status, date, blood type, units
  4. User can filter by status, date range, blood type
  5. User can export as PDF

3.6 Peer Verification Network

FR-3.6.1: Register as Verifier

- **Description:** Community members can become verifiers
- **Actors:** User
- **Requirements:**
  - Account age > 30 days
  - Min 3 successful donations (for donors)
  - Reputation score > 500
  - Pass identity verification
- **Main Flow:**
  1. User applies to become verifier
  2. System checks eligibility criteria
  3. System creates VerifierPool record
  4. System assigns initial qualification score 0.5
  5. Verifier receives blockchain credentials (NFT)

FR-3.6.2: Perform Verification

- **Description:** Verifiers attestveganate donation authenticity
- **Main Flow:**
  1. Verifier receives verification request
  2. Verifier reviews donation proof (IPFS documents)
  3. Verifier checks COMPLETED status and medical records
  4. Verifier signs verification using private key (blockchain)
  5. System collects 3 independent signatures
  6. System creates Merkle root of signatures
  7. System validates all signatures match
  8. Verification marked as VERIFIED_BLOCKCHAIN
  9. Verifier earns 25 tokens for successful verification

FR-3.6.3: Qualification Scoring

- **Description:** Verifier qualification dynamically updates
- **Algorithm:**

```
qualificationScore = (successRate * 0.7) + ((1 - disputeRate) * 0.3)
successRate = successfulVerifications / totalVerifications
```

```
disputeRate = disputedVerifications / totalVerifications
```

- **Rewards:**
  - Success rate > 95%: Bonus 10 tokens per verification
  - 90-95%: Standard 5 tokens per verification
  - < 90%: Warning, potential removal
- **Removal Criteria:**
  - Dispute rate > 10%
  - Qualification score < 0.4 for 30 days
  - Multiple fraud accusations

3.7 Reputation System

FR-3.7.1: Reputation Scoring

- **Description:** Dynamic reputation reflecting user trustworthiness
- **Base Score:** 500 (neutral)
- **Events & Points:**

Event	Points	Multiplier
Successful Donation	+100	1.0-2.0 (urgency)
Positive Review (5-star)	+50	1.0
Negative Review (1-star)	-100	1.0
Failed Donation	-50	1.0
Fraud Flag	-200	1.0
Verification Passed	+25	1.0
Inactive (30+ days)	-5% monthly decay	1.0

- **Reputation Levels:**

Level	Score Range	Badge	Benefits
Bronze	0-499		Basic matching
Silver	500-1499		Priority matching +20%
Gold	1500-2999		VIP matching +40%
Platinum	3000+		Emergency fast-track +100%

FR-3.7.2: Reputation Decay

- **Background Job:** Runs daily
- **Algorithm:** Monthly decay of 5% for inactive donors (>30 days)
- **Recovery:** Can rebuild through new donations

3.8 Reward System

FR-3.8.1: Token Issuance

- **Description:** ERC-20 tokens issued for donations
- **Amount Calculation:**

```
tokens = unitsCollected * 100 * urgencyMultiplier * reputationMultiplier
urgencyMultiplier = { LOW: 1.0, MEDIUM: 1.2, HIGH: 1.5, CRITICAL: 2.0, EMERGENCY: 3.0
reputationMultiplier = 1.0 + (reputationLevel / 4)
```

- **Smart Contract:** Automatic minting on blockchain
- **Example:** 1 unit donation + CRITICAL urgency + Gold reputation = 300 tokens
- **Wallet:** Auto-transferred to user's wallet address

FR-3.8.2: NFT Badge Minting

- **Description:** Soulbound NFT for achievements
- **Triggers:**
  1. 10 successful donations → "Gold Donor" NFT
  2. 50 successful donations → "Platinum Donor" NFT
  3. Biometric verified → "Verified" NFT
  4. Verifier → "Ambassador" NFT
- **Metadata (IPFS):**

```
{
  "name": "Gold Donor Badge",
  "description": "10+ successful donations",
  "image": "ipfs://QmHash...",
  "attributes": {
    "rarity": "RARE",
    "achievement": "Gold",
    "donationCount": 10,
    "minted": "2025-11-06"
  }
}
```

- **Contract:** ERC-721 with Soulbound transfers (non-transferable)

FR-3.8.3: Reward Claim

- **Description:** Users claim accumulated tokens
- **Main Flow:**
  1. User navigates to rewards page
  2. System displays pending tokens
  3. User clicks "Claim Rewards"
  4. System transfers tokens to wallet via smart contract
  5. System resets pending balance to 0
  6. User receives transaction hash

3.9 Notification System

FR-3.9.1: Real-time Notifications

- **Channels:**
  1. **In-app:** WebSocket via Socket.IO
  2. **Email:** Nodemailer SMTP (Gmail)
  3. **Push:** Browser notifications (PWA)
  4. **SMS:** Twilio integration (optional)
- **Notification Types:**

Type	Trigger	Template
MATCH_FOUND	AI matching completes	"You matched with a request!"
DONATION_COMPLETED	Donation verified	"Thank you for donating!"
URGENT_REQUEST	EMERGENCY broadcast	"🚨 Urgent blood needed nearby!"
REWARD_ISSUED	Rewards minted	"You earned 300 tokens!"
VERIFICATION_REQUIRED	Donation needs approval	"Action required for verification"
REQUEST_EXPIRED	Request expires	"Your request expired"

- Socket.IO **Events:**

```
// Emit to donor
socket.emit('auto-matched', {
  matchId: string,
  requestId: string,
  aiScore: number,
  urgency: string,
  expiresIn: number // seconds
})

// Broadcast emergency
socket.emit('emergency-broadcast', {
```

```
  type: 'NEW_REQUEST',
  bloodType: string,
  urgency: string,
  radius: number
})

// Notify on completion
socket.emit('reward-issued', {
  amount: number,
  nftMinted: boolean,
  transactionHash: string
})
```

3.10 Fraud Detection

FR-3.10.1: Multi-Layer Fraud Detection

- **Layer 1: Behavioral Analysis**
  - Failed verification attempts > 50%
  - Inconsistent success rates
  - Trust score variance
- **Layer 2: Device Fingerprinting**
  - VPN/Proxy detection
  - Shared device detection
  - Rooted/jailbroken devices
  - Emulator detection
- **Layer 3: Velocity Analysis**
  - > 5 requests/hour → High risk
  - > 10 requests/day → Medium risk
  - Impossible travel (1000km in <1hr)
- **Layer 4: Pattern Analysis**
  - New account (age < 7 days)
  - Sudden activity after inactivity
  - Abnormal donation frequency
- **Layer 5: Network Analysis**
  - Connections to flagged accounts
  - Multiple related flagged users
  - Shared payment methods

FR-3.10.2: Fraud Score Calculation

- **Algorithm:**

```
totalScore = (behavioral * 0.2) + (device * 0.15) +
             (velocity * 0.25) + (pattern * 0.2) +
             (network * 0.2)
```

- **Thresholds:**
  - Score > 0.7: Create FraudAlert, review
  - Score > 0.9: Auto-block user, notify admin
  - Score > 0.95: Critical alert, immediate investigation

FR-3.10.3: Automatic Actions

- **Score 0.7-0.9:**
  - FraudAlert created
  - User warned
  - Transaction reviewed by verifier
  - May require additional verification
- **Score > 0.9:**
  - User auto-blocked
  - Account frozen
  - Admin notified
  - Investigation initiated

4. NON-FUNCTIONAL REQUIREMENTS

4.1 Performance

- **Response Time:** API endpoints < 500ms (p95)
- **Database Queries:** < 100ms for indexed queries
- **AI Matching:** < 5s per request (100 donors)
- **Real-time:** WebSocket latency < 100ms
- **Throughput:** 1000 req/sec capacity

4.2 Scalability

- **Horizontal Scaling:** Stateless API servers
- **Database:** Read replicas for reporting
- **Caching:** Redis for session/match results
- **CDN:** CloudFlare for static assets
- **Queue:** Bull/BullMQ for async jobs

4.3 Security

- **Authentication:** JWT tokens (7-day expiry)
- **Encryption:** AES-256-GCM for sensitive data
- **Hashing:** bcrypt (salt rounds: 10)
- **HTTPS:** TLS 1.3 mandatory
- **CORS:** Whitelist allowed origins
- **Rate Limiting:** 100 req/min per IP
- **Input Validation:** Zod schemas
- **SQL Injection:** Prisma parameterized queries
- **XSS Protection:** Next.js built-in

4.4 Availability

- **Uptime:** 99.9% SLA target
- **Redundancy:** Multi-region deployment
- **Backup:** Daily automated backups
- **Disaster Recovery:** RTO 1 hour, RPO 15 minutes
- **Health Checks:** Every 30 seconds

4.5 Usability

- **Accessibility:** WCAG 2.1 AA compliance
- **Mobile:** Responsive design, PWA ready
- **Internationalization:** Support for multiple languages (EN, HI, ES)
- **Offline:** Service worker caching

4.6 Compliance

- **Data Privacy:** GDPR, HIPAA compliance
- **Encryption:** All PII encrypted at rest/transit
- **Audit Trail:** All transactions logged

- **Consent:** Explicit user consent for data usage
- **Right to Delete:** GDPR right to be forgotten implementation

5. API SPECIFICATION

5.1 Authentication Endpoints

POST /api/auth/register

- **Description:** Register new user
- **Request Body:**

```
{
  "email": "user@example.com",
  "password": "SecurePass123!",
  "name": "John Doe",
  "phone": "+1234567890",
  "role": "DONOR"
}
```

- **Response (201):**

```
{
  "success": true,
  "userId": "user-id-123",
  "message": "Registration successful. Check email for verification."
}
```

- **Error (400):** Invalid email/password format
- **Error (409):** User already exists

POST /api/auth/login

- **Description:** Authenticate user
- **Request Body:**

```
{
  "email": "user@example.com",
  "password": "SecurePass123!"
}
```

- **Response (200):**

```
{
  "success": true,
  "token": "eyJhbGciOiJIUzI1NiIs...",
  "expiresIn": 604800,
  "user": {
    "id": "user-id-123",
```

```
    "email": "user@example.com",
    "role": "DONOR"
  }
}
```

- **Error (401):** Invalid credentials
- **Error (403):** Account blocked

POST /api/auth/verify-email

- **Description:** Verify email token
- **Query Params:** `?token=verification-token`
- **Response (200):**

```
{
  "success": true,
  "message": "Email verified successfully"
}
```

- **Error (400):** Invalid/expired token

POST /api/auth/logout

- **Description:** Invalidate JWT token
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "message": "Logged out successfully"
}
```

5.2 Profile Endpoints

GET /api/profile

- **Description:** Get user profile
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "id": "user-id-123",
    "email": "user@example.com",
    "name": "John Doe",
    "role": "DONOR",
    "donorProfile": {
```

```
      "bloodType": "O_POSITIVE",
      "reputationScore": 750,
      "isAvailable": true,
      "totalSuccessfulDonations": 5
    }
  }
}
```

PUT /api/profile

- **Description:** Update user profile
- **Headers:** `Authorization: Bearer token`
- **Request Body:**

```
{
  "name": "John Doe Updated",
  "phone": "+1234567890",
  "city": "San Francisco",
  "state": "CA"
}
```

- **Response (200):** Updated profile object

GET /api/profile/reputation

- **Description:** Get reputation stats
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "totalScore": 750,
    "level": "GOLD",
    "badge": "",
    "successRate": 0.92,
    "totalDonations": 5,
    "nftBadges": ["VERIFIED", "GOLD_DONOR"]
  }
}
```

5.3 Blood Request Endpoints

POST /api/blood-requests/create

- **Description:** Create blood request
- **Headers:** `Authorization: Bearer token`
- **Request Body:**

```
{
  "bloodType": "O_POSITIVE",
  "rhFactor": "POSITIVE",
  "unitsNeeded": 3,
  "urgencyLevel": "CRITICAL",
  "latitude": 37.7749,
  "longitude": -122.4194,
  "radius": 50,
  "medicalProofIPFS": "QmHash..."
}
```

- **Response (201):**

```
{
  "success": true,
  "requestId": "request-123",
  "automatchedDonors": 7,
  "matches": [
    {
      "id": "match-1",
      "score": 0.92,
      "donorId": "donor-1"
    }
  ]
}
```

GET /api/blood-requests/:requestId

- **Description:** Get request details
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "id": "request-123",
    "bloodType": "O_POSITIVE",
    "unitsNeeded": 3,
    "urgencyLevel": "CRITICAL",
    "status": "MATCHED",
    "matches": 7,
    "createdAt": "2025-11-06T10:00:00Z",
    "expiresAt": "2025-11-07T10:00:00Z"
  }
}
```

GET /api/blood-requests/active

- **Description:** Get all active requests
- **Headers:** `Authorization: Bearer token`
- **Query Params:** `?page=1&limit=20&urgency=CRITICAL`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "total": 45,
    "page": 1,
    "limit": 20,
    "requests": [...]
  }
}
```

PUT /api/blood-requests/:requestId/cancel

- **Description:** Cancel blood request
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "message": "Request cancelled"
}
```

5.4 Matching Endpoints

POST /api/blood-requests/automated-matching

- **Description:** Trigger autonomous matching
- **Headers:** `Authorization: Bearer token`
- **Request Body:**

```
{
  "requestId": "request-123"
}
```

- **Response (200):**

```
{
  "success": true,
  "matchCount": 10,
  "matches": [
    {
      "id": "match-1",
```

```
      "donorId": "donor-1",
      "score": 0.92,
      "features": {
        "bloodTypeScore": 1.0,
        "distanceScore": 0.85,
        "reputationScore": 0.9,
        "availabilityScore": 1.0
      }
    }
  ]
}
```

GET /api/blood-requests/:requestId/matches

- **Description:** Get matches for request
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "requestId": "request-123",
    "totalMatches": 10,
    "matches": [...]
  }
}
```

POST /api/matches/:matchId/accept

- **Description:** Accept match
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "matchId": "match-1",
  "message": "Match accepted"
}
```

POST /api/matches/:matchId/reject

- **Description:** Reject match
- **Headers:** `Authorization: Bearer token`
- **Request Body:**

```
{
  "reason": "Not available today"
}
```

- **Response (200):**

```
{
  "success": true,
  "message": "Match rejected"
}
```

5.5 Donation Endpoints

POST /api/donations/record-blockchain

- **Description:** Record donation on blockchain
- **Headers:** `Authorization: Bearer token`
- **Request Body:**

```
{
  "matchId": "match-1",
  "unitsCollected": 1,
  "ipfsProofHash": "QmHash...",
  "verifierSignatures": [
    "0x...",
    "0x...",
    "0x..."
  ]
}
```

- **Response (201):**

```
{
  "success": true,
  "donationId": "donation-1",
  "transactionHash": "0xHash...",
  "rewardIssued": 300,
  "nftMinted": true,
  "nftTokenId": "nft-1"
}
```

GET /api/donations/history

- **Description:** Get donation history
- **Headers:** `Authorization: Bearer token`
- **Query Params:** `?page=1&limit=20&status=COMPLETED`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "total": 5,
    "donations": [
      {
```

```
        "id": "donation-1",
        "bloodType": "O_POSITIVE",
        "unitsCollected": 1,
        "status": "COMPLETED",
        "reward": 300,
        "completedAt": "2025-11-05T14:30:00Z"
      }
    ]
  }
}
```

GET /api/donations/:donationId

- **Description:** Get donation details
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "id": "donation-1",
    "matchId": "match-1",
    "donor": {...},
    "recipient": {...},
    "bloodType": "O_POSITIVE",
    "unitsCollected": 1,
    "status": "COMPLETED",
    "transactionHash": "0xHash...",
    "blockchainVerified": true,
    "rewardIssued": 300,
    "nftMinted": true
  }
}
```

5.6 Verification Endpoints

POST /api/verify/biometric

- **Description:** Verify user identity via face recognition
- **Headers:** `Authorization: Bearer token`, `Content-Type: multipart/form-data`
- **Form Data:**
  - `image`: File (JPG/PNG, max 5MB)
- **Response (200):**

```
{
  "success": true,
  "verified": true,
  "confidence": 0.95,
  "livenessScore": 0.92,
```

```
  "message": "Biometric verification passed"
}
```

- **Error (403):**

```
{
  "success": false,
  "verified": false,
  "spoofRisk": "HIGH",
  "message": "Spoof detected"
}
```

POST /api/verify/email

- **Description:** Verify email address
- **Headers:** `Authorization: Bearer token`
- **Query Params:** `?token=otp-token`
- **Response (200):**

```
{
  "success": true,
  "message": "Email verified"
}
```

POST /api/verify/phone

- **Description:** Verify phone via OTP
- **Headers:** `Authorization: Bearer token`
- **Request Body:**

```
{
  "phone": "+1234567890",
  "otp": "123456"
}
```

- **Response (200):**

```
{
  "success": true,
  "message": "Phone verified"
}
```

5.7 Verifier Endpoints

POST /api/verifier/register

- **Description:** Register as verifier
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "verifierId": "verifier-1",
  "message": "Registered as verifier"
}
```

GET /api/verifier/pending

- **Description:** Get pending verifications
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "total": 5,
    "verifications": [
      {
        "id": "verification-1",
        "donationId": "donation-1",
        "proofUrl": "ipfs://...",
        "requestedAt": "2025-11-06T10:00:00Z"
      }
    ]
  }
}
```

POST /api/verifier/:verificationId/attest

- **Description:** Attest verification
- **Headers:** `Authorization: Bearer token`
- **Request Body:**

```
{
  "verified": true,
  "signature": "0x...",
  "notes": "Documents verified"
}
```

- **Response (200):**

```
{
  "success": true,
```

```
  "message": "Verification attested"
}
```

5.8 Reward Endpoints

GET /api/rewards

- **Description:** Get reward balance
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "pendingTokens": 500,
    "claimedTokens": 1000,
    "nftBadges": ["VERIFIED", "GOLD_DONOR"],
    "totalNFTs": 2
  }
}
```

POST /api/rewards/claim

- **Description:** Claim pending rewards
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "claimedAmount": 500,
  "transactionHash": "0x...",
  "message": "Rewards claimed successfully"
}
```

GET /api/rewards/nfts

- **Description:** Get NFT badges
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "badges": [
      {
        "id": "nft-1",
        "name": "Verified Badge",
        "tokenId": "1",
```

```
        "metadata": {...}
      }
    ]
  }
}
```

5.9 Notification Endpoints

GET /api/notifications

- **Description:** Get all notifications
- **Headers:** `Authorization: Bearer token`
- **Query Params:** `?page=1&limit=20&read=false`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "total": 15,
    "notifications": [
      {
        "id": "notif-1",
        "type": "MATCH_FOUND",
        "title": "New Match!",
        "message": "...",
        "read": false,
        "createdAt": "2025-11-06T10:00:00Z"
      }
    ]
  }
}
```

PUT /api/notifications/:notificationId/read

- **Description:** Mark notification as read
- **Headers:** `Authorization: Bearer token`
- **Response (200):**

```
{
  "success": true,
  "message": "Marked as read"
}
```

DELETE /api/notifications/:notificationId

- **Description:** Delete notification
- **Headers:** `Authorization: Bearer token`
- **Response (204):** No content

5.10 Admin Endpoints

GET /api/admin/analytics

- **Description:** Get platform analytics
- **Headers:** `Authorization: Bearer token`, Role: ADMIN
- **Response (200):**

```
{
  "success": true,
  "data": {
    "totalUsers": 5000,
    "totalDonors": 3000,
    "totalRecipients": 2000,
    "totalDonations": 10000,
    "totalUnitsCollected": 15000,
    "averageMatchingTime": "45s",
    "fraudRate": "0.8%",
    "revenueGenerated": "$50000"
  }
}
```

GET /api/admin/users

- **Description:** Get all users with filters
- **Headers:** `Authorization: Bearer token`, Role: ADMIN
- **Query Params:** `?page=1&limit=50&role=DONOR&verified=true&blocked=false`
- **Response (200):**

```
{
  "success": true,
  "data": {
    "total": 3000,
    "users": [...]
  }
}
```

PUT /api/admin/users/:userId/block

- **Description:** Block user
- **Headers:** `Authorization: Bearer token`, Role: ADMIN
- **Request Body:**

```
{
  "reason": "Fraud detected",
  "duration": 3600
}
```

- **Response (200):**

```
{
  "success": true,
  "message": "User blocked"
}
```

6. DATABASE SCHEMA

6.1 Key Tables

users

```
CREATE TABLE users (
  id VARCHAR(36) PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  passwordHash VARCHAR(255),
  name VARCHAR(100),
  phone VARCHAR(20),
  role ENUM('DONOR', 'RECIPIENT', 'VERIFIER', 'ADMIN'),
  walletAddress VARCHAR(42) UNIQUE,
  publicKey TEXT,
  verified BOOLEAN DEFAULT false,
  blockedFromPlatform BOOLEAN DEFAULT false,
  verificationStatus ENUM('PENDING', 'VERIFIED', 'REJECTED', 'FLAGGED'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  lastActiveAt TIMESTAMP,
  INDEX idx_email (email),
  INDEX idx_walletAddress (walletAddress)
);
```

donor_profiles

```
CREATE TABLE donor_profiles (
  id VARCHAR(36) PRIMARY KEY,
  userId VARCHAR(36) UNIQUE NOT NULL,
  bloodType ENUM('A_POSITIVE', ...) NOT NULL,
  rhFactor ENUM('POSITIVE', 'NEGATIVE'),
  latitude FLOAT,
  longitude FLOAT,
  city VARCHAR(100),
  state VARCHAR(100),
  country VARCHAR(100),
  isAvailable BOOLEAN DEFAULT true,
  lastDonationDate TIMESTAMP,
  nextEligibleDate TIMESTAMP,
  reputationScore INT DEFAULT 500,
  aiReputationScore FLOAT DEFAULT 0.5,
  fraudRiskScore FLOAT DEFAULT 0,
  totalSuccessfulDonations INT DEFAULT 0,
  totalFailedMatches INT DEFAULT 0,
  avgResponseTime INT,
  biometricVerified BOOLEAN DEFAULT false,
  biometricHash VARCHAR(64),
  lastBiometricVerified TIMESTAMP,
  totalRewardsEarned FLOAT DEFAULT 0,
  nftBadgesIssued JSON,
  rewardNFTsMinted INT DEFAULT 0,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (userId) REFERENCES users(id),
  INDEX idx_bloodType (bloodType),
  INDEX idx_isAvailable (isAvailable),
  INDEX idx_reputationScore (reputationScore)
);
```

blood_requests

```
CREATE TABLE blood_requests (
  id VARCHAR(36) PRIMARY KEY,
  recipientId VARCHAR(36) NOT NULL,
  bloodType ENUM('A_POSITIVE', ...) NOT NULL,
  rhFactor ENUM('POSITIVE', 'NEGATIVE'),
  unitsNeeded INT,
  urgencyLevel ENUM('LOW', 'MEDIUM', 'HIGH', 'CRITICAL', 'EMERGENCY'),
  latitude FLOAT,
  longitude FLOAT,
  radius INT DEFAULT 50,
  status ENUM('OPEN', 'MATCHED', 'FULFILLED', 'EXPIRED') DEFAULT 'OPEN',
  medicalProofIPFSHash VARCHAR(255),
  verificationStatus ENUM('PENDING', 'VERIFIED', 'REJECTED', 'FLAGGED'),
  verifiedByPeers JSON,
  smartContractAddress VARCHAR(42),
  transactionHash VARCHAR(66),
  isOnChain BOOLEAN DEFAULT false,
```

```
  autoMatchingEnabled BOOLEAN DEFAULT true,
  expiresAt TIMESTAMP,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (recipientId) REFERENCES users(id),
  INDEX idx_status (status),
  INDEX idx_urgencyLevel (urgencyLevel),
  INDEX idx_bloodType (bloodType),
  INDEX idx_expiresAt (expiresAt)
);
```

request_matches

```
CREATE TABLE request_matches (
  id VARCHAR(36) PRIMARY KEY,
  requestId VARCHAR(36) NOT NULL,
  donorId VARCHAR(36) NOT NULL,
  compatibilityScore FLOAT,
  distanceScore FLOAT,
  reputationScore FLOAT,
  availabilityScore FLOAT,
  urgencyScore FLOAT,
  successRateScore FLOAT,
  responseTimeScore FLOAT,
  fraudRiskScore FLOAT,
  overallScore FLOAT,
  status ENUM('PENDING', 'ACCEPTED', 'REJECTED', 'COMPLETED', 'EXPIRED') DEFAULT 'PENDING
  escrowContractAddress VARCHAR(42),
  escrowAmount FLOAT,
  automatchedAt TIMESTAMP,
  respondedAt TIMESTAMP,
  expiresAt TIMESTAMP,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (requestId) REFERENCES blood_requests(id),
  FOREIGN KEY (donorId) REFERENCES donor_profiles(id),
  UNIQUE KEY unique_request_donor (requestId, donorId),
  INDEX idx_status (status),
  INDEX idx_expiresAt (expiresAt)
);
```

donations

```
CREATE TABLE donations (
  id VARCHAR(36) PRIMARY KEY,
  matchId VARCHAR(36) NOT NULL,
  donorId VARCHAR(36) NOT NULL,
  requestId VARCHAR(36) NOT NULL,
  bloodType ENUM('A_POSITIVE', ...),
  rhFactor ENUM('POSITIVE', 'NEGATIVE'),
  unitsCollected INT,
  status ENUM('PENDING', 'IN_TRANSIT', 'COMPLETED', 'FAILED') DEFAULT 'PENDING',
  transactionHash VARCHAR(66) UNIQUE,
  blockchainVerified BOOLEAN DEFAULT false,
```

```
  smartContractHash VARCHAR(66),
  completionProofIPFS VARCHAR(255),
  rewardTokensIssued FLOAT,
  nftMinted BOOLEAN DEFAULT false,
  nftTokenId VARCHAR(255),
  completedAt TIMESTAMP,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (matchId) REFERENCES request_matches(id),
  FOREIGN KEY (donorId) REFERENCES users(id),
  FOREIGN KEY (requestId) REFERENCES blood_requests(id),
  INDEX idx_status (status),
  INDEX idx_transactionHash (transactionHash)
);
```

7. PROJECT FILE STRUCTURE

```
bloodchain/
├── app/
│   ├── api/
│   │   ├── auth/
│   │   │   ├── [...nextauth]/
│   │   │   │   └── route.ts
│   │   │   ├── register/
│   │   │   │   └── route.ts
│   │   │   ├── login/
│   │   │   │   └── route.ts
│   │   │   └── verify-email/
│   │   │       └── route.ts
│   │   ├── blood-requests/
│   │   │   ├── create/
│   │   │   │   └── route.ts
│   │   │   ├── [id]/
│   │   │   │   ├── route.ts
│   │   │   │   └── matches/
│   │   │   │       └── route.ts
│   │   │   ├── active/
│   │   │   │   └── route.ts
│   │   │   ├── automated-matching/
│   │   │   │   └── route.ts
│   │   │   └── [id]/
│   │   │       └── cancel/
│   │   │           └── route.ts
│   │   ├── donations/
│   │   │   ├── record-blockchain/
│   │   │   │   └── route.ts
│   │   │   ├── history/
│   │   │   │   └── route.ts
│   │   │   └── [id]/
│   │   │       └── route.ts
│   │   ├── verify/
│   │   │   ├── biometric/
│   │   │   │   └── route.ts
│   │   │   ├── email/
```

```
│   │   │   │   └── route.ts
│   │   │   └── phone/
│   │   │       └── route.ts
│   │   ├── verifier/
│   │   │   ├── register/
│   │   │   │   └── route.ts
│   │   │   ├── pending/
│   │   │   │   └── route.ts
│   │   │   └── [id]/
│   │   │       └── attest/
│   │   │           └── route.ts
│   │   ├── rewards/
│   │   │   ├── route.ts
│   │   │   ├── claim/
│   │   │   │   └── route.ts
│   │   │   └── nfts/
│   │   │       └── route.ts
│   │   ├── notifications/
│   │   │   ├── route.ts
│   │   │   ├── [id]/
│   │   │   │   ├── read/
│   │   │   │   │   └── route.ts
│   │   │   │   └── delete/
│   │   │   │       └── route.ts
│   │   ├── admin/
│   │   │   ├── analytics/
│   │   │   │   └── route.ts
│   │   │   ├── users/
│   │   │   │   ├── route.ts
│   │   │   │   └── [id]/
│   │   │   │       └── block/
│   │   │   │           └── route.ts
│   │   │   └── fraud/
│   │   │       └── alerts/
│   │   │           └── route.ts
│   │   └── profile/
│   │       ├── route.ts
│   │       └── reputation/
│   │           └── route.ts
│   ├── dashboard/
│   │   ├── page.tsx
│   │   ├── layout.tsx
│   │   ├── matching/
│   │   │   └── page.tsx
│   │   ├── donations/
│   │   │   └── page.tsx
│   │   ├── rewards/
│   │   │   └── page.tsx
│   │   └── notifications/
│   │       └── page.tsx
│   ├── layout.tsx
│   ├── page.tsx
│   ├── globals.css
│   ├── middleware.ts
│   └── providers.tsx
├── src/
```

```
│   ├── services/
│   │   ├── matching.service.ts
│   │   ├── biometric.service.ts
│   │   ├── reputation.service.ts
│   │   ├── reward.service.ts
│   │   ├── notification.service.ts
│   │   ├── fraud-detection.service.ts
│   │   ├── verification.service.ts
│   │   └── blockchain.service.ts
│   ├── jobs/
│   │   ├── autonomous-matching.job.ts
│   │   ├── reputation-decay.job.ts
│   │   ├── fraud-detection.job.ts
│   │   └── cleanup.job.ts
│   ├── lib/
│   │   ├── prisma.ts
│   │   ├── auth.ts
│   │   ├── logger.ts
│   │   ├── geo-util.ts
│   │   ├── validators.ts
│   │   ├── redis.ts
│   │   └── constants.ts
│   ├── middleware/
│   │   ├── rate-limit.ts
│   │   ├── auth.ts
│   │   └── error-handler.ts
│   ├── types/
│   │   └── index.ts
│   └── components/
│       ├── MatchingDashboard.tsx
│       ├── BiometricVerification.tsx
│       ├── RewardDashboard.tsx
│       ├── NotificationPanel.tsx
│       └── BloodRequestForm.tsx
├── prisma/
│   ├── schema.prisma
│   ├── seed.ts
│   └── migrations/
│       └── [timestamp]_init/
│           └── migration.sql
├── contracts/
│   ├── HemoBridgeAutonomous.sol
│   ├── HemoBridgeToken.sol
│   └── deploy.ts
├── scripts/
│   ├── train-model.ts
│   ├── evaluate-model.ts
│   └── seed-db.ts
├── public/
│   ├── manifest.json
│   ├── sw.js
│   └── icons/
├── tests/
│   ├── api/
│   ├── services/
│   └── utils/
```

```
├── docs/
│   ├── API.md
│   ├── ARCHITECTURE.md
│   ├── DEPLOYMENT.md
│   └── CONTRIBUTING.md
├── .env.local.example
├── .env.local
├── .gitignore
├── docker-compose.yml
├── Dockerfile
├── next.config.js
├── tsconfig.json
├── tailwind.config.js
├── postcss.config.js
├── package.json
├── pnpm-lock.yaml
├── jest.config.js
└── README.md
```

8. TESTING STRATEGY

8.1 Unit Tests

- **Coverage Target:** 80%+
- **Tools:** Jest, React Testing Library
- **Focus Areas:**
  - Service business logic
  - Utility functions
  - Type safety

8.2 Integration Tests

- **Coverage:** API endpoints, database interactions
- **Tools:** Supertest, Prisma test client
- **Focus Areas:**
  - Authentication flow
  - Matching algorithm
  - Blockchain interactions

8.3 E2E Tests

- **Coverage:** User workflows
- **Tools:** Playwright
- **Scenarios:**

- Complete donation cycle
- Registration to matching
- Reward claiming

8.4 Performance Tests

- **Load Testing:** k6, Artillery
- **Target:** 1000 req/sec at <500ms p95
- **Scenarios:**
  - Concurrent matching requests
  - Database query performance

9. DEPLOYMENT

9.1 Infrastructure

- **Hosting:** Vercel (frontend) + AWS/GCP (backend)
- **Database:** AWS RDS PostgreSQL
- **Cache:** Redis Cloud
- **Storage:** AWS S3 for IPFS gateway
- **Blockchain:** Polygon mainnet

9.2 CI/CD Pipeline

- **VCS:** GitHub
- **CI:** GitHub Actions
- **Stages:**
  1. Code lint & type check
  2. Unit & integration tests
  3. Build Docker image
  4. Deploy to staging
  5. E2E tests
  6. Deploy to production

9.3 Monitoring

- **APM:** DataDog/NewRelic
- **Logs:** ELK Stack/Splunk
- **Metrics:** Prometheus

- **Alerts:** PagerDuty

10. SECURITY CONSIDERATIONS

10.1 Data Protection

- AES-256-GCM encryption for PII
- Biometric data hashed, never stored raw
- SSL/TLS for all communications
- CORS properly configured

10.2 Access Control

- JWT with short expiry (7 days)
- Role-based access control (RBAC)
- Rate limiting per endpoint
- IP whitelisting for admin APIs

10.3 Smart Contract Security

- Multi-signature verification
- Reentrancy guards
- Circuit breaker pattern
- External audit recommended

CONCLUSION

Bloodchain represents a paradigm shift in blood donation management by combining cutting-edge technologies (AI, blockchain, biometrics) to create a transparent, efficient, and user-centric platform. This SRS provides comprehensive specifications for full implementation.

Document Version: 1.0.0
Last Updated: November 6, 2025
Status: Complete & Production-Ready