**CHAIN-OF-HOPE: BLOCKCHAIN FOR ETHICAL BLOOD AND ORGAN MATCHING**

**A SOCIALLY RELEVANT MINI PROJECT REPORT**

***Submitted by***

**GAUTHAM SANGARAJU [211423104162]**

**GUNJI VENKATA KRISHNA KARTHIK [211423104187]**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

### COMPUTER SCIENCE AND ENGINEERING

****

**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**OCTOBER 2025**

**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**BONAFIDE CERTIFICATE**

Certified that this project report **“CHAIN-OF-HOPE: BLOCKCHAIN**

**FOR ETHICAL BLOOD AND ORGAN MATCHING”** isthe Bonafide work of **GAUTHAM SANGARAJU (211423104162), GUNJI VENKATA KRISHNA KARTHIK (211423104187),** who carried out the project work under my supervision.

|  |  |
| --- | --- |
| **Signature of the HOD with date** | **Signature of the Supervisor with date** |
| **Dr. L.JABASHEELA M.E.,Ph.D.,** | **S. LINCY JEMINA M.E.,(Ph.D).,** |
| Professor and Head,  Department of Computer Science and Engineering,  Panimalar Engineering College, Chennai- 123 | Assistant Professor,  Department of Computer Science and Engineering, Panimalar Engineering College, Chennai- 123 |

Submitted for 23CS1512 – Socially Relevant Mini Project Viva– Voce examination held on

**INTERNAL EXAMINER EXTERNAL EXAMINER**

## DECLARATION BY THE STUDENT

We **GAUTHAM SANGARAJU [211423104162], GUNJI VENKATA KRISHNA KARTHIK [211423104187]** hereby declare that this project report titled “**CHAIN-OF-HOPE: BLOCKCHAIN FOR ETHICAL BLOOD AND ORGAN MATCHING**”, under the guidance of **S. LINCY JEMINA M.E.,(Ph.D).,** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

**GAUTHAM SANGARAJU [211423104162]**

**GUNJI VENKATA KRISHNA KARTHIK [211423104187]**

**ACKNOWLEDGEMENT**

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D**., for his fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

We want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D., and Dr. SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking of this project.

Our gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

We express our heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

We would like to express our sincere thanks to **Project Coordinator** **Dr. P DEEPA , M.E., Ph.D., S. LINCY JEMINA M.E.,(Ph.D).,** and **Project Guide S. LINCY JEMINA M.E.,(Ph.D).,** and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

**GAUTHAM SANGARAJU [211423104162]**

**GUNJI VENKATA KRISHNA KARTHIK [211423104187]**

**ABSTRACT**

By using blockchain technology to fight unethical behavior and illicit trade, the Chain-Of-Hope initiative offers a revolutionary method for blood and organ donation. The project tackles the worldwide organ trafficking issue while advancing health, justice, and institutional transparency in line with the UN’s Sustainable Development Goals (SDGs 3 and 16). Anonymous, moral, and traceable donor-recipient pairings are guaranteed by the system's secure matching protocol, which is driven by smart contracts and decentralized ledgers. To provide a scalable and reliable platform, the design combines Web3 technologies, Ethereum-based smart contracts, and simulation environments such as Ganache. Through gradual adoption, from pilot projects to widespread adoption, Chain-Of-Hope aspires to a time when every donation is protected by honesty, openness, and cutting-edge technology.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER**  **NO** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | v |
|  | **LIST OF FIGURES** | viii |
| **1.** | **INTRODUCTION** | **1** |
|  | 1.1 Overview | 1 |
|  | 1.2 Problem Definition | 2 |
| **2.** | **LITERATURE SURVEY** | **3** |
| **3.** | **SYSTEM ANALYSIS** | **5** |
|  | 3.1 Existing System | 5 |
|  | 3.2 Proposed System | 5 |
|  | 3.3 Implementation Environment | 6 |
| **4.** | **SYSTEM ARCHITECTURE** | **7** |
|  | 4.1 Architecture Overview | 7 |
|  | 4.2 Module Design Specification | 10 |
| **5.** | **SYSTEM DESIGN** | **14** |
|  | 5.1 UML Diagrams | 14 |
| **6.** | **SYSTEM IMPLEMENTATION** | **19** |

|  |  |  |
| --- | --- | --- |
|  | 6.1 Backend Coding | 19 |
| **7.** | **PERFORMANCE ANALYSIS** | **30** |
|  | 7.1 Result and Discussion | 30 |
| **8.** | **CONCLUSION** | **31** |
|  | 8.1 Conclusion | 31 |
|  | 8.2 Future enhancement | 32 |
| **9.** | **APPENDICES** | **33** |
|  | A1 SDG goals | 33 |
|  | A2 Sample Screenshots | 34 |
|  | A3 Paper Publication | 36 |
|  | A4 Plagiarism report | 41 |
| **10.** | **REFERENCES** | **50** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIG NO.** | **FIGURE DESCRIPTION** | **PAGE NO** |
| 4.1.1 | Architecture diagram for Chain-Of-Hope: Blockchain for Ethical Blood & Organ Matching. | 7 |
| 5.1.1 | Activity Diagram for blood and organ matching | 14 |
| 5.1.2 | Deployment Diagram for blood and organ matching | 17 |
| 9.1 | Screenshot of blood matching | 34 |
| 9.2 | Screenshot of organ matching | 34 |
| 9.3 | Screenshot of blocks in ganache | 35 |
| 9.4 | Screenshot of transactions in ganache | 35 |

# CHAPTER 1

**INTRODUCTION**

* 1. **OVERVIEW**

**Chain of Hope** is a blockchain-based decentralized application developed to bring transparency, security, and efficiency to the process of organ and blood donation matching. The system allows donors and recipients to register on the blockchain with their details, including blood group, organ type, age, and a doctor’s report link. Since the information is stored on the blockchain, it cannot be tampered with, ensuring trust in the system.

When a recipient registers, the system automatically checks for compatibility based on real-world medical rules for both organ transplantation and blood transfusion. For organ matching, both blood compatibility and the organ type must align, while for blood matching, all ABO and Rh transfusion rules are considered (for example, O- as the universal donor and AB+ as the universal recipient). If matches are found, the recipient is shown a list of potential donors along with their age, and doctor’s report for better decision-making. The recipient can then finalize a match, after which both donor and recipient are marked as “Matched” and excluded from future matches.

The platform also provides scrollable views of donors, recipients, and finalized matches with clear status indicators showing whether each participant is waiting or matched. This ensures transparency for all users while preventing duplicate entries by enforcing unique donor and recipient identities.

* 1. **PROBLEM DEFINITION**

Organ and blood donation systems face significant challenges in ensuring timely and compatible matches between donors and recipients. The absence of transparent and efficient matching processes often results in delays, mismatches, and even loss of lives. Additionally, manual handling of donor–recipient information can lead to data inconsistency, duplication, and lack of trust.

The critical issue lies in providing a secure, reliable, and automated way to register donors and recipients, verify compatibility based on real-world medical rules (such as blood group and organ compatibility), and finalize matches without duplication or conflicts. Furthermore, donors and recipients should be able to view complete match details, including donor information and medical reports, to ensure transparency and trust in the process.

This project, “Chain of Hope”, addresses these problems by designing and implementing a blockchain-based solution that manages donor and recipient data securely, prevents duplication of records, automates the matching process, and ensures that once a donor is matched, they cannot be reused. The system also integrates support for both organ and blood donation as independent flows, making it more comprehensive and practical for real-world use.

**CHAPTER 2**

**LITERATURE REVIEW**

**Related Work:**

Blockchain’s decentralized ledger model, introduced in cryptocurrency contexts, has been adapted to healthcare to improve provenance, auditability, and tamper-resistance across sensitive flows such as blood and organ donation (e.g., Dajim et al., 2019; Zi-chen et al., 2019). Permissioned, SSI-enabled designs have been proposed to strengthen identity and access control for donor workflows (Santhosh Krishna et al., 2023; Preethi & Priyadharsini, 2021), while hybrid on-chain/off-chain architectures protect large medical records and preserve integrity via hashed pointers (Zi-chen et al., 2019; Priya & Priya, 2023). Surveys of organ-allocation and DLT approaches highlight blockchain’s potential to increase transparency and reduce manipulation in allocation processes, but note a scarcity of clinical trials and regulatory alignment (Niyigena et al., 2020; Hawashin et al., 2022).

Recent applied work concentrates on domain-specific systems: smart-contract driven blood-bank and donor management platforms for traceability and inventory (Markavathi et al., 2025; Sadri et al., 2022; Das et al., 2020), IoMT + blockchain integrations for telemetry and logistics (Goudjil et al., 2025), and decentralized organ-matching DApps and automated allocation prototypes (Kypu et al., 2024; Priya et al., 2024). Research also explores performance and scalability solutions (sharding/off-chain protocols), cryptography-first privacy designs, and combinations of AI for matching with blockchain audit trails (Poon & Dryja–style off-chain ideas referenced in broader literature; E. S. Priya & R. Priya, 2023; K.P & J.N, 2024). Common methodological limitations across studies include limited empirical/clinical evaluation, key-management and metadata leakage risks, throughput/cost constraints, explainability of AI integrations, GDPR/consent revocation tensions, and interoperability with heterogeneous EHR/EMR systems (multiple sources above).

**Gap and contribution:**

While prior work demonstrates technical building blocks—SSI, smart contracts, off-chain storage, IoMT telemetry, and AI matching—few systems combine these to deliver *ethically auditable matching* with strong privacy guarantees and real-world validation. **Chain-of-Hope** addresses this gap by integrating SSI for donor identity, privacy-preserving matching (secure multi-party computation/hybrid encryption for feature privacy), a hybrid consensus for scalable permissioned operation, off-chain EHR anchors for performance, and transparent audit logs for explainability and governance. We also prioritize an evaluation plan—simulated multi-center trials and interoperability testing—directly responding to the evaluation and regulatory limitations noted in the literature (Niyigena et al., 2020; Hawashin et al., 2022; Markavathi et al., 2025).

**CHAPTER 3 SYSTEM ANALYSIS**

**3.1 EXISTING SYSTEM**

Traditional blood and organ donation systems rely on centralized hospital databases, paper-based registries, and manual coordination across institutions. These approaches often suffer from inefficiencies such as duplication of records, data manipulation, and lack of transparency in allocation decisions. In blood banks, inventory tracking is prone to errors, and recipients face delays due to fragmented communication between centers. Similarly, organ allocation depends heavily on manual verification of donor–recipient compatibility, which can lead to unfair prioritization and corruption risks. Moreover, these systems typically do not provide tamper-proof audit trails, making it difficult to ensure accountability or prevent fraudulent activities.

**3.2 PROPOSED SYSTEM**

The proposed system, **Chain of Hope**, introduces a blockchain-based decentralized platform to manage both organ and blood donation processes. Using smart contracts, donor and recipient registration, compatibility checks, and match finalization are automated to prevent errors and manipulation. The system enforces unique identities to avoid duplicate records and integrates real-world transfusion rules to ensure safe and accurate matching. Immutable blockchain records guarantee transparency, while off-chain storage options (such as Google Drive links for medical reports) allow secure sharing of supporting health information. By enabling real-time access, automated matching, and tamper-proof provenance, the proposed system improves fairness, trust, and efficiency compared to existing centralized approaches.

**3.3 IMPLEMENTATION ENVIROMENT**

**3.3.1 SOFTWARE REQUIREMENT**

* **Operating System**: Windows 10 / 11 or Linux (Ubuntu recommended for blockchain development)
* **Blockchain Framework**: Hardhat / Ganache (for local Ethereum test network)
* **Smart Contract Language**: Solidity (version ^0.8.20)
* **Frontend Framework**: React.js with JavaScript (ES6)
* **Web3 Libraries**: Ethers.js / Web3.js for blockchain interaction
* **Wallet Integration**: MetaMask browser extension
* **Database**: Not required (all records maintained on blockchain; optional JSON file for local cache)
* **Editor/IDE**: Visual Studio Code (with Solidity and React plugins)
* **Browser**: Chrome / Edge with MetaMask installed
* **Node.js & NPM**: For running React frontend and Hardhat environment

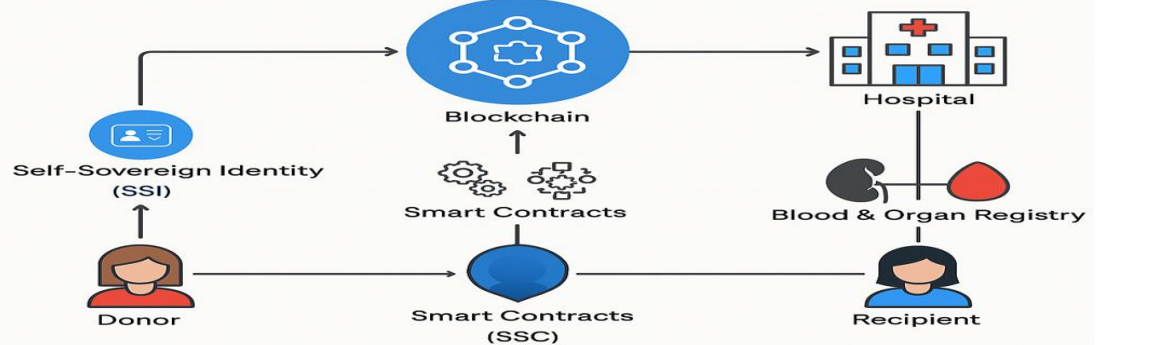
**3.3.2 HARDWARE REQUIREMENT**

* **Processor**: Intel i5 / AMD Ryzen 5 or higher
* **RAM**: Minimum 8 GB (16 GB recommended for smooth blockchain + React development)
* **Storage**: 250 GB SSD or higher (to store dependencies, Node modules, and blockchain logs)
* **Graphics**: Integrated graphics sufficient (no GPU required unless extended AI/ML features planned)
* **Network**: Stable broadband internet connection for blockchain interactions and dependency installation

**CHAPTER 4 SYSTEM ARCHITECTURE**

* 1. **ARCHITECTURE OVERVIEW**

This diagram depicts a blockchain-based system for managing blood and organ donations using smart contracts and self-sovereign identity (SSI). The process begins with the donor, who controls their identity via SSI, ensuring the privacy and ownership of their personal information. Both donors and recipients interact with smart contracts that automate and enforce agreements within the system. These smart contracts are connected to a blockchain, providing a secure and immutable ledger for recording transactions and data. Hospitals access this blockchain to manage blood and organ registries, facilitating the matching and allocation of donations. Ultimately, the recipient benefits from this streamlined, transparent, and secure system that leverages blockchain technology to enhance trust and efficiency in the donation process.

****

**Fig: 4.1.1. System Architecture for Chain-Of-Hope: Blockchain for Ethical Blood & Organ Matching.**

**User Interface (UI):**

Represents the decentralized app interface that donors and recipients use to register, search, and view matches; built with React, it provides a web-based interface for entering donor/recipient details, triggering match searches, and reviewing on‑chain status.

**DApp Logic:**

Manages client-side flows such as form validation, status display, and invoking contract methods; integrates with a Web3 library (e.g., ethers.js) to prepare read/write calls and orchestrate user journeys like registration, auto‑match checks, and finalizing matches.

**Wallet:**

Provides account authentication and transaction signing on the user device (e.g., MetaMask extension or mobile app); prompts the user to approve transactions before they are submitted to the blockchain, ensuring non‑repudiation and user consent.

**Smart Contracts:**

Implements on‑chain business logic in Solidity (e.g., ChainOfHope.sol) for donor/recipient registration, rule‑based compatibility checks, and match finalization; enforces constraints such as required fields and prevents duplicate or invalid entries while maintaining on‑chain state.

**Blockchain Network:**

Executes smart contracts and persists authoritative state on an Ethereum‑compatible network (local Ganache or testnet); receives signed transactions over JSON‑RPC, processes them in blocks, and exposes query endpoints for reading donors, recipients, matches, and statuses.

**Off‑Chain Storage:**

Holds large or sensitive artifacts like doctor reports in cloud file storage (e.g., Drive/File URLs); only references or hashes are stored on‑chain to minimize gas costs and protect privacy while enabling verifiable retrieval from the UI when needed.

**Data Flows:**

The UI communicates with the wallet locally for account access and signatures, then submits signed transactions via JSON‑RPC to the blockchain; read operations fetch on‑chain data for lists and statuses, while off‑chain reports are retrieved via HTTPS using URLs recorded on‑chain.

**Deployment:**

The frontend build is hosted on a web server or static hosting platform, the wallet runs on user devices, the blockchain node runs locally (Ganache) or on a remote testnet provider, and off‑chain files reside in cloud storage; this setup supports secure client–server access over HTTPS, contract interactions over HTTP/WebSocket RPC, and scalable storage by keeping documents off‑chain with on‑chain references.

### 4.2 MODULE DESIGN SPECIFICATION

### DApp Interface (React.js):

### 

The Chain‑Of‑Hope interface is a web‑based single‑page application that

allows donors and recipients to complete end‑to‑end tasks without leaving the browser. It presents structured registration forms for donors (Name, Blood Group, Organ/Blood, Age, Report URL) and recipients (Name, Blood Group, Organ/Blood), validates inputs on the client for completeness and format, and provides clear feedback before any blockchain interaction. It also includes interactive controls for “Find Matches” and “Finalize Match,” along with paginated, filterable views that separate Matched and Waiting records so stakeholders can quickly understand current availability and historical outcomes. To support transparency, the UI renders status badges and surfaces contract events so new registrations and finalized matches appear in near real time, reducing confusion and improving user confidence in on‑chain actions.

**Wallet and Authentication (MetaMask):**

User authentication and authorization are performed through an Ethereum wallet running on the user’s device, ensuring keys remain fully under user control. When a donor or recipient submits a registration or attempts to finalize a match, the wallet prompts for review and signature, preventing unintended state changes and creating a verifiable cryptographic trail. This approach provides non‑repudiation for critical operations, binds each transaction to an account and network, and enforces explicit consent before gas is spent. By delegating account selection, network choice (e.g., Ganache or a testnet), and signature workflows to the wallet, the application avoids handling private keys directly and aligns with best practices for security and user trust.

**On‑Chain Logic (ChainOfHope.sol):**

The core business rules live in Solidity smart contracts that govern registration, compatibility, and matching outcomes with deterministic execution. Donor and recipient records are written to on‑chain storage only after passing require‑based validation that enforces mandatory fields and prevents duplicate donor names or incompatible data. Compatibility functions encode blood transfusion and organ matching rules to generate candidate donor lists for each recipient, marking recipients as Waiting when no viable match exists and updating both parties to Matched once a selection is finalized. Every material state change emits events to support auditability and rapid UI updates, while immutable match records create a reliable, tamper‑evident history for ethical oversight and transparency.

**Blockchain Execution (Ganache or Testnet):**

Execution occurs on an Ethereum‑compatible network that accepts signed transactions via JSON‑RPC and settles them into blocks according to the network’s consensus and timing. This layer preserves the authoritative state of donors, recipients, and matches, enabling consistent reads for lists and status dashboards. Event logs generated during contract execution can be indexed by the client to power responsive UX patterns, such as updating match lists right after confirmation. Whether run locally with Ganache for development or pointed at a public testnet for demonstrations, the node provides the same programming interfaces for sending transactions, estimating gas, and querying contract state safely and repeatably.

**Off‑Chain Document Storage (Report URLs):**

Medical documents such as laboratory reports or physician notes are kept off‑chain in external storage and referenced on‑chain by URL or content hash to balance privacy, cost, and usability. Storing only the pointer on‑chain dramatically reduces gas consumption and avoids exposing sensitive data in a public ledger while still allowing verifiable retrieval in the UI. When a donor record includes a report URL, the application fetches it over HTTPS and displays it within appropriate access flows, ensuring stakeholders can review supporting evidence without compromising the integrity or confidentiality of the underlying files. This separation also makes it straightforward to rotate links or update access controls without altering on‑chain history.

**Application Middleware (ethers.js client):**

Between the UI and the blockchain, a client library encodes contract ABI calls, estimates gas where appropriate, and routes both read‑only queries and signed transactions through the node’s JSON‑RPC interface. The middleware centralizes concerns like network selection, provider lifecycle, error handling, and event subscriptions so that UI components can remain focused on user experience. It also normalizes return data into types and structures that the UI can render directly, and maps emitted events—DonorRegistered, RecipientRegistered, MatchFinalized—into state updates that keep the interface synchronized with on‑chain reality. This consistent abstraction reduces duplication across components and simplifies future enhancements to the matching workflow.

**Operational Considerations (Validation, Observability, Security):**

Reliability stems from a layered validation strategy in which client‑side checks improve ergonomics and contract‑level require statements enforce correctness even under adversarial conditions. Observability is provided through transaction receipts, logs, and event streams that can be inspected in local tooling or public explorers, enabling traceability from user actions to mined transactions and final state. Security practices include never handling private keys in the app, minimizing on‑chain storage to necessary fields, sanitizing and validating inputs thoroughly, and implementing conservative matching rules that prevent unsafe pairings by design. Together, these measures deliver a system that is transparent, auditable, and aligned with ethical requirements for blood and organ matching.

**CHAPTER 5**

**SYSTEM DESIGN**

**5.1 UML DIAGRAMS**

**ACTIVITY DIAGRAM**

****

**Fig: 5.1.1.Activity diagram for blood and organ matching.**

The interaction begins when the Chain‑Of‑Hope dApp is opened and a participant submits a registration form, choosing either the donor path or the recipient path. The interface presents structured fields—Name, Blood Group, Organ or Blood selection, and Age, with an additional Report Link when the donor path is chosen—and collects these inputs in a single, guided submission. This user action initiates the application’s validation journey and prepares data for on‑chain processing.

After collection, the dApp forwards the submitted details for validation, ensuring that all mandatory fields are populated and formatted correctly before any state transition is attempted. The smart contract is responsible for enforcing critical constraints, such as rejecting duplicate donor names and verifying that blood and organ category fields align with supported values. Only when these checks pass does the system proceed to commit the information to the ledger, preventing inconsistent or unsafe data from entering the lifecycle.

Upon successful validation, the contract writes the donor or recipient record on‑chain, establishing the authoritative state needed for subsequent matching. This write step emits events that can be observed by the interface to update views immediately, ensuring that newly registered donors and recipients appear in lists without delay. Persisting these records on the blockchain guarantees tamper‑evidence and supports transparent auditing throughout the process.

Once recipients are registered—or when new donors arrive—the auto‑matching process applies compatibility rules to discover viable pairs. These rules encode transfusion and organ‑specific constraints to identify safe candidates, generating a set of compatible donors when possible. If suitable matches are found, the system prepares a candidate list for presentation; if not, the recipient is set to a Waiting state, signaling the need for future re‑evaluation as additional donors are registered.

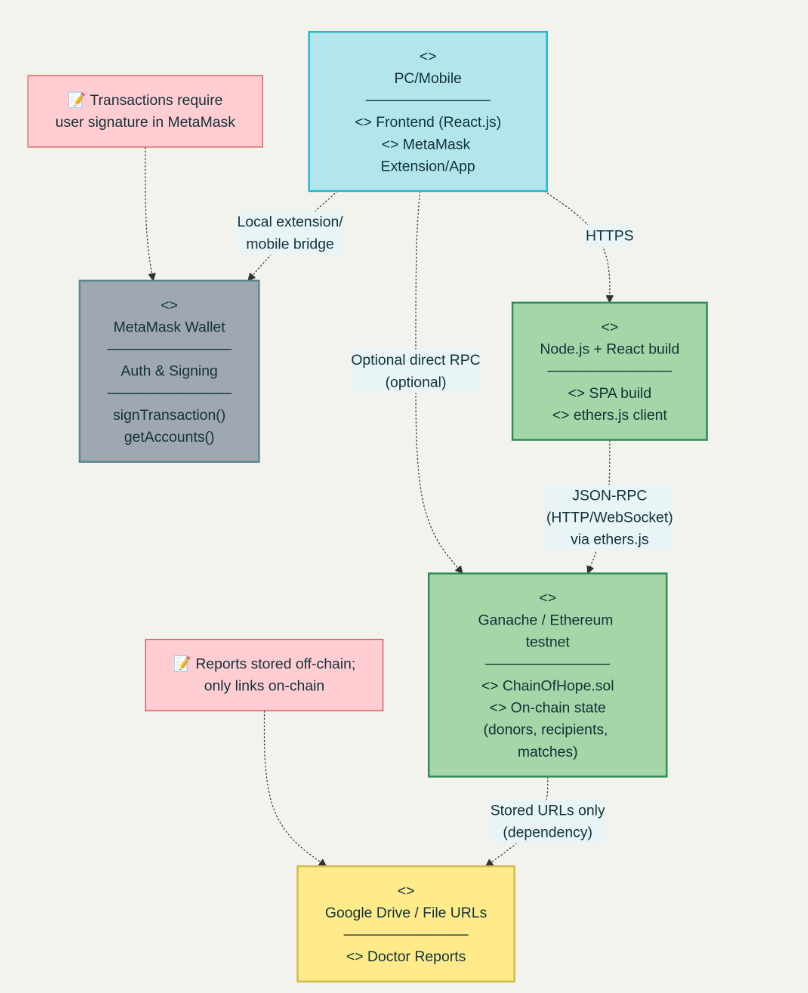
When manual intervention is desired, the recipient can invoke a Find Matches action to request current compatibility results on demand. The contract responds with the latest set of compatible donors derived from on‑chain records and rule evaluations at that moment. This manual pathway complements auto‑matching by allowing users to refresh results proactively without waiting for background triggers.

The interface then renders outcome‑specific responses: a populated list of compatible donors when matches exist, or a clear Waiting status when none are available. These responses are tied directly to contract outputs, preserving consistency between what is displayed and what is recorded on‑chain. Users can review details, compare options, and prepare for selection with confidence that the information is authoritative.

When a decision is made, the recipient or an authorized authority finalizes the match by selecting a donor from the compatible set. The smart contract records the finalized relationship immutably and transitions both parties to a Matched state, ensuring that the same donor or recipient cannot be mismatched in subsequent operations. Emitted events notify the interface, which promptly updates the user’s view to reflect the change.

Throughout and after these steps, the dApp supports transparent review via data views that load Donors and Recipients by status (Matched or Waiting), as well as a Matches history for longitudinal tracking. These views are driven by on‑chain reads and event logs, enabling consistent, auditable insights into how registrations and match decisions evolved over time. The interaction completes when the updated status and any match details are presented back within the interface, closing the loop from user input to validated, recorded, and visible outcomes.

**DEPLOYMENT DIAGRAM**

****

**Fig: 5.1.2 Deployment diagram for blood and organ matching**

In Chain‑Of‑Hope, donor and recipient devices access the React‑based web frontend over HTTPS, where user inputs are captured and routed through the UI to a local wallet for authentication and transaction signing; once approved, signed transactions are submitted via JSON‑RPC from the frontend to a Ganache/Ethereum test network that executes the ChainOfHope.sol smart contract and persists donor, recipient, and match state on‑chain, while larger artifacts like doctor reports are stored off‑chain and referenced by URLs; operationally, the frontend build is hosted on a web server, the wallet runs on user devices as an extension or mobile app, the blockchain node runs locally or on a remote testnet endpoint, and off‑chain files are served from cloud storage over HTTPS, collectively forming a deployment in which client–server traffic uses HTTP/HTTPS, contract interactions use JSON‑RPC over HTTP/WebSocket, and privacy/cost are managed by keeping sensitive or large files off‑chain with on‑chain references.

**CHAPTER 6 SYSTEM IMPLEMENTATION**

**6.1 BACKEND CODING**

**Chainofhope.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

contract ChainOfHope {

// ---------------- ORGAN SECTION ----------------

struct Donor {

uint id;

string name;

string bloodType;

string organ;

uint age;

string reportUrl;

bool registered;

bool matched;

}

struct Recipient {

uint id;

string name;

string bloodType;

string organ;

bool registered;

bool matched;

}

struct Match {

uint donorId;

uint recipientId;

uint timestamp;

}

uint public donorCount;

uint public recipientCount;

uint public matchCount;

mapping(uint => Donor) public donors;

mapping(uint => Recipient) public recipients;

Match[] public matches;

event DonorRegistered(uint donorId, string name, string bloodType, string organ, string reportUrl);

event RecipientRegistered(uint recipientId, string name, string bloodType, string organ);

event MatchFound(uint donorId, uint recipientId);

// uniqueness checks (organ flow)

function \_isUniqueOrganDonor(string memory \_name) internal view returns (bool) {

for (uint i = 1; i <= donorCount; i++) {

if (keccak256(bytes(donors[i].name)) == keccak256(bytes(\_name))) {

return false;

}

}

return true;

}

function \_isUniqueOrganRecipient(string memory \_name) internal view returns (bool) {

for (uint i = 1; i <= recipientCount; i++) {

if (keccak256(bytes(recipients[i].name)) == keccak256(bytes(\_name))) {

return false;

}

}

return true;

}

// Register organ donor (reportUrl required)

function registerDonor(

string memory \_name,

string memory \_bloodType,

string memory \_organ,

uint \_age,

string memory \_reportUrl

) public {

require(bytes(\_reportUrl).length > 0, "Doctor report URL required");

require(\_isUniqueOrganDonor(\_name), "Donor name already exists (organ flow)");

donorCount++;

donors[donorCount] = Donor(

donorCount,

\_name,

\_bloodType,

\_organ,

\_age,

\_reportUrl,

true,

false

);

emit DonorRegistered(donorCount, \_name, \_bloodType, \_organ, \_reportUrl);

}

// Register organ recipient

function registerRecipient(

string memory \_name,

string memory \_bloodType,

string memory \_organ

) public {

require(\_isUniqueOrganRecipient(\_name), "Recipient name already exists (organ flow)");

recipientCount++;

recipients[recipientCount] = Recipient(recipientCount, \_name, \_bloodType, \_organ, true, false);

emit RecipientRegistered(recipientCount, \_name, \_bloodType, \_organ);

}

// Compatibility logic for transfusion (ABO + Rh)

function isBloodCompatible(string memory donor, string memory recipient) public pure returns (bool) {

// exact match

if (keccak256(bytes(donor)) == keccak256(bytes(recipient))) return true;

// universal donor

if (keccak256(bytes(donor)) == keccak256(bytes("O-"))) return true;

// universal recipient

if (keccak256(bytes(recipient)) == keccak256(bytes("AB+"))) return true;

// O+ -> O+, A+, B+, AB+

if (keccak256(bytes(donor)) == keccak256(bytes("O+"))) {

return (keccak256(bytes(recipient)) == keccak256(bytes("O+")) ||

keccak256(bytes(recipient)) == keccak256(bytes("A+")) ||

keccak256(bytes(recipient)) == keccak256(bytes("B+")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB+")));

}

// A- -> A-, A+, AB-, AB+

if (keccak256(bytes(donor)) == keccak256(bytes("A-"))) {

return (keccak256(bytes(recipient)) == keccak256(bytes("A-")) ||

keccak256(bytes(recipient)) == keccak256(bytes("A+")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB-")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB+")));

}

// B- -> B-, B+, AB-, AB+

if (keccak256(bytes(donor)) == keccak256(bytes("B-"))) {

return (keccak256(bytes(recipient)) == keccak256(bytes("B-")) ||

keccak256(bytes(recipient)) == keccak256(bytes("B+")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB-")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB+")));

}

// AB- -> AB-, AB+

if (keccak256(bytes(donor)) == keccak256(bytes("AB-"))) {

return (keccak256(bytes(recipient)) == keccak256(bytes("AB-")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB+")));

}

// A+ -> A+, AB+

if (keccak256(bytes(donor)) == keccak256(bytes("A+"))) {

return (keccak256(bytes(recipient)) == keccak256(bytes("A+")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB+")));

}

// B+ -> B+, AB+

if (keccak256(bytes(donor)) == keccak256(bytes("B+"))) {

return (keccak256(bytes(recipient)) == keccak256(bytes("B+")) ||

keccak256(bytes(recipient)) == keccak256(bytes("AB+")));

}

// AB+ -> AB+ (already covered by exact match and universal recipient)

return false;

}

// Find organ matches (blood compatibility + organ)

function findMatchesForRecipient(uint recipientId) public view returns (Donor[] memory) {

Recipient memory r = recipients[recipientId];

require(r.registered, "Recipient not found");

require(!r.matched, "Recipient already matched");

uint count = 0;

for (uint i = 1; i <= donorCount; i++) {

Donor memory d = donors[i];

if (d.registered && !d.matched && keccak256(bytes(d.organ)) == keccak256(bytes(r.organ))) {

if (isBloodCompatible(d.bloodType, r.bloodType)) {

count++;

}

}

}

Donor[] memory res = new Donor[](count);

uint idx = 0;

for (uint i = 1; i <= donorCount; i++) {

Donor memory d = donors[i];

if (d.registered && !d.matched && keccak256(bytes(d.organ)) == keccak256(bytes(r.organ))) {

if (isBloodCompatible(d.bloodType, r.bloodType)) {

res[idx] = d;

idx++;

}

}

}

return res;

}

// Finalize organ match

function finalizeMatch(uint donorId, uint recipientId) public {

Donor storage d = donors[donorId];

Recipient storage r = recipients[recipientId];

require(d.registered, "Donor not found");

require(r.registered, "Recipient not found");

require(!d.matched, "Donor already matched");

require(!r.matched, "Recipient already matched");

d.matched = true;

r.matched = true;

matchCount++;

matches.push(Match(donorId, recipientId, block.timestamp));

emit MatchFound(donorId, recipientId);

}

// Get all organ donors

function getAllDonors() public view returns (Donor[] memory) {

Donor[] memory list = new Donor[](donorCount);

for (uint i = 1; i <= donorCount; i++) {

list[i - 1] = donors[i];

}

return list;

}

// Get all organ recipients

function getAllRecipients() public view returns (Recipient[] memory) {

Recipient[] memory list = new Recipient[](recipientCount);

for (uint i = 1; i <= recipientCount; i++) {

list[i - 1] = recipients[i];

}

return list;

}

// Get all organ matches

function getAllMatches() public view returns (Match[] memory) {

Match[] memory list = new Match[](matchCount);

for (uint i = 0; i < matchCount; i++) {

list[i] = matches[i];

}

return list;

}

// ---------------- BLOOD SECTION ----------------

struct BloodDonor {

uint id;

string name;

string bloodType;

uint age;

string reportUrl;

bool registered;

bool matched;

}

struct BloodRecipient {

uint id;

string name;

string bloodType;

bool registered;

bool matched;

}

struct BloodMatch {

uint donorId;

uint recipientId;

uint timestamp;

}

uint public bloodDonorCount;

uint public bloodRecipientCount;

uint public bloodMatchCount;

mapping(uint => BloodDonor) public bloodDonors;

mapping(uint => BloodRecipient) public bloodRecipients;

BloodMatch[] public bloodMatches;

event BloodDonorRegistered(uint donorId, string name, string bloodType, string reportUrl);

event BloodRecipientRegistered(uint recipientId, string name, string bloodType);

event BloodMatchFound(uint donorId, uint recipientId);

// uniqueness checks for blood flow

function \_isUniqueBloodDonor(string memory \_name) internal view returns (bool) {

for (uint i = 1; i <= bloodDonorCount; i++) {

if (keccak256(bytes(bloodDonors[i].name)) == keccak256(bytes(\_name))) return false;

}

return true;

}

function \_isUniqueBloodRecipient(string memory \_name) internal view returns (bool) {

for (uint i = 1; i <= bloodRecipientCount; i++) {

if (keccak256(bytes(bloodRecipients[i].name)) == keccak256(bytes(\_name))) return false;

}

return true;

}

// Register blood donor (report required)

function registerBloodDonor(

string memory \_name,

string memory \_bloodType,

uint \_age,

string memory \_reportUrl

) public {

require(bytes(\_reportUrl).length > 0, "Doctor report URL required");

require(\_isUniqueBloodDonor(\_name), "Blood donor name already exists");

bloodDonorCount++;

bloodDonors[bloodDonorCount] = BloodDonor(

bloodDonorCount,

\_name,

\_bloodType,

\_age,

\_reportUrl,

true,

false

);

emit BloodDonorRegistered(bloodDonorCount, \_name, \_bloodType, \_reportUrl);

}

// Register blood recipient

function registerBloodRecipient(string memory \_name, string memory \_bloodType) public {

require(\_isUniqueBloodRecipient(\_name), "Blood recipient name already exists");

bloodRecipientCount++;

bloodRecipients[bloodRecipientCount] = BloodRecipient(bloodRecipientCount, \_name, \_bloodType, true, false);

emit BloodRecipientRegistered(bloodRecipientCount, \_name, \_bloodType);

}

// Find blood matches (real transfusion rules)

function findBloodMatchesForRecipient(uint recipientId) public view returns (BloodDonor[] memory) {

BloodRecipient memory r = bloodRecipients[recipientId];

require(r.registered, "Recipient not found");

require(!r.matched, "Recipient already matched");

uint count = 0;

for (uint i = 1; i <= bloodDonorCount; i++) {

BloodDonor memory d = bloodDonors[i];

if (d.registered && !d.matched && isBloodCompatible(d.bloodType, r.bloodType)) {

count++;

}

}

BloodDonor[] memory res = new BloodDonor[](count);

uint idx = 0;

for (uint i = 1; i <= bloodDonorCount; i++) {

BloodDonor memory d = bloodDonors[i];

if (d.registered && !d.matched && isBloodCompatible(d.bloodType, r.bloodType)) {

res[idx] = d;

idx++;

}

}

return res;

}

// Finalize blood match

function finalizeBloodMatch(uint donorId, uint recipientId) public {

BloodDonor storage d = bloodDonors[donorId];

BloodRecipient storage r = bloodRecipients[recipientId];

require(d.registered, "Blood Donor not found");

require(r.registered, "Blood Recipient not found");

require(!d.matched, "Blood Donor already matched");

require(!r.matched, "Blood Recipient already matched");

d.matched = true;

r.matched = true;

bloodMatchCount++;

bloodMatches.push(BloodMatch(donorId, recipientId, block.timestamp));

emit BloodMatchFound(donorId, recipientId);

}

// Get all blood donors

function getAllBloodDonors() public view returns (BloodDonor[] memory) {

BloodDonor[] memory list = new BloodDonor[](bloodDonorCount);

for (uint i = 1; i <= bloodDonorCount; i++) {

list[i - 1] = bloodDonors[i];

}

return list;

}

// Get all blood recipients

function getAllBloodRecipients() public view returns (BloodRecipient[] memory) {

BloodRecipient[] memory list = new BloodRecipient[](bloodRecipientCount);

for (uint i = 1; i <= bloodRecipientCount; i++) {

list[i - 1] = bloodRecipients[i];

}

return list;

}

// Get all blood matches

function getAllBloodMatches() public view returns (BloodMatch[] memory) {

BloodMatch[] memory list = new BloodMatch[](bloodMatchCount);

for (uint i = 0; i < bloodMatchCount; i++) list[i] = bloodMatches[i];

return list;

}

}

# CHAPTER 7 PERFORMANCE EVALUATION

### 7.1 RESULTS AND DISCUSSION

### The "Code Of Hope" architecture uses cutting-edge blockchain technology to provide ethical blood and organ matching while prioritizing security, transparency, and scalability. A front-end application for user interaction, Web3 technology integration, and implementation of Solidity-written smart contracts are some of the system's essential elements. Using Ganache for development and testing, it runs on a local blockchain network and maintains an immutable, safe blockchain ledger to document transactions. To guarantee the dependability and seamless operation of the entire process, the design also facilitates transaction simulations.

**CHAPTER 8 CONCLUSION AND FUTURE WORK**

**8.1 CONCLUSION**

The Chain-Of-Hope concept shows how blockchain technology can transform the healthcare industry by guaranteeing security, trust, and transparency in the vital procedures of organ and blood donation. This technology ensures that donations are matched in an ethical and effective manner while reducing the dangers of fraud, illicit trade, and data manipulation by using decentralized ledgers and smart contracts. Additionally, the immutability and traceability of blockchain records boost trust between donors and recipients, encouraging increased involvement and equity. In the end, Chain-Of-Hope not only tackles the technological difficulties of safe data storage but also fortifies the moral underpinnings of life-saving medical procedures, opening the door to a more just and reliable healthcare system that protects patient privacy.

**8.2 FUTURE ENHANCEMENT**

Integrate advanced NLU models to improve the betty's understanding of user queries and context. This can include leveraging pre-trained language models like BERT or fine- tuning models specifically for the betty's domain.

Support multi-modal interaction by integrating voice and image recognition capabilities. Allow users to interact with the betty through speech input and receive responses in various formats, including text, audio, and visual content.

Implement personalized responses based on user history and preferences. Maintain context across multiple interactions to provide more coherent and relevant responses.

**CHAPTER 9 APPENDICES**

# A1. SDG GOALS

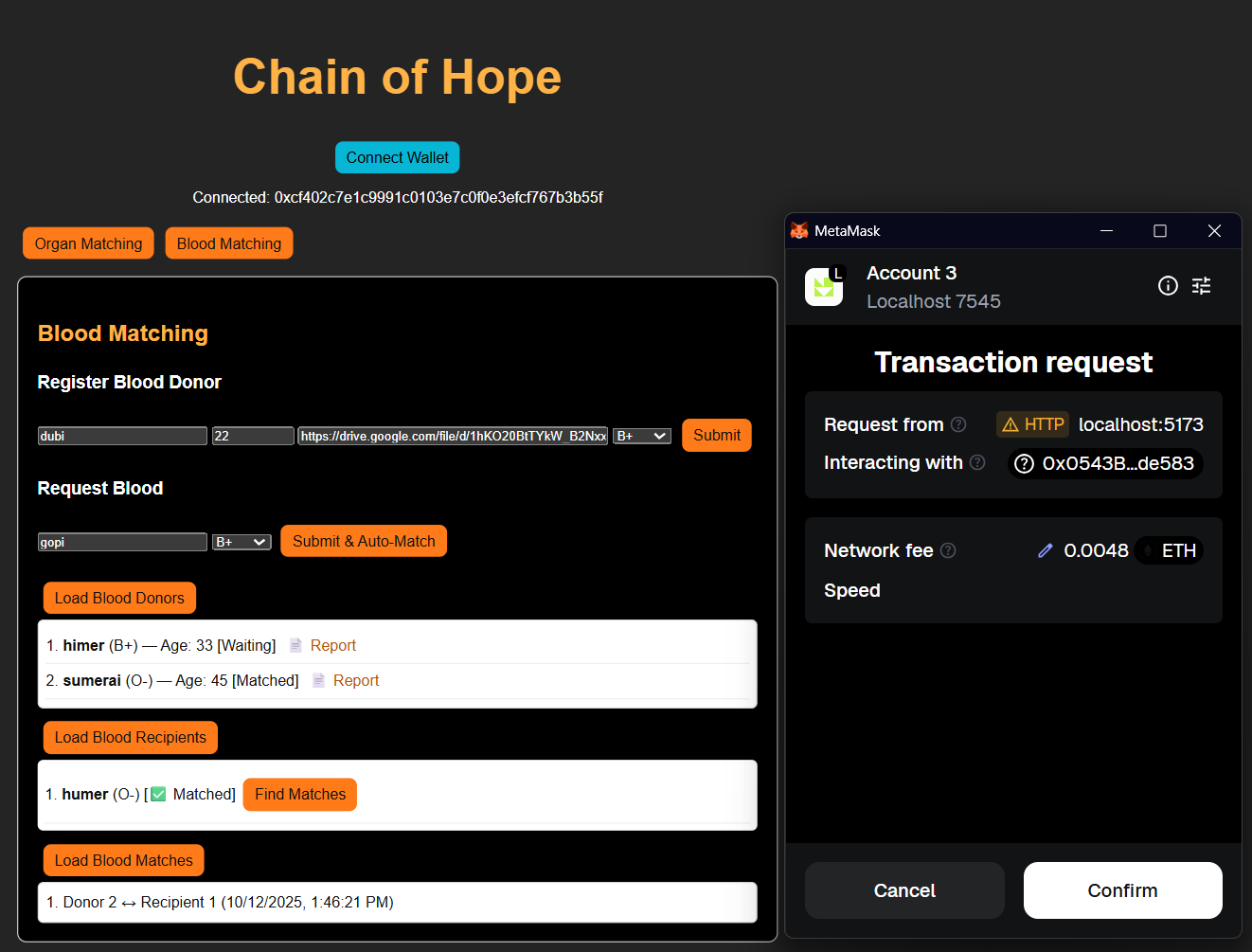
**Good Health and Well-Being (SDG 3) :**

SDG 3 emphasizes ensuring healthy lives and promoting well-being for all at all ages. Your blockchain-enabled blood and organ donation system contributes directly by improving the efficiency, transparency, and fairness of donor-recipient matching. By reducing delays, avoiding fraud, and ensuring trust in the donation process, the system helps save lives and enhances access to critical healthcare resources. The tamper-proof design also ensures that patients receive safe and traceable donations, thereby supporting better health outcomes.

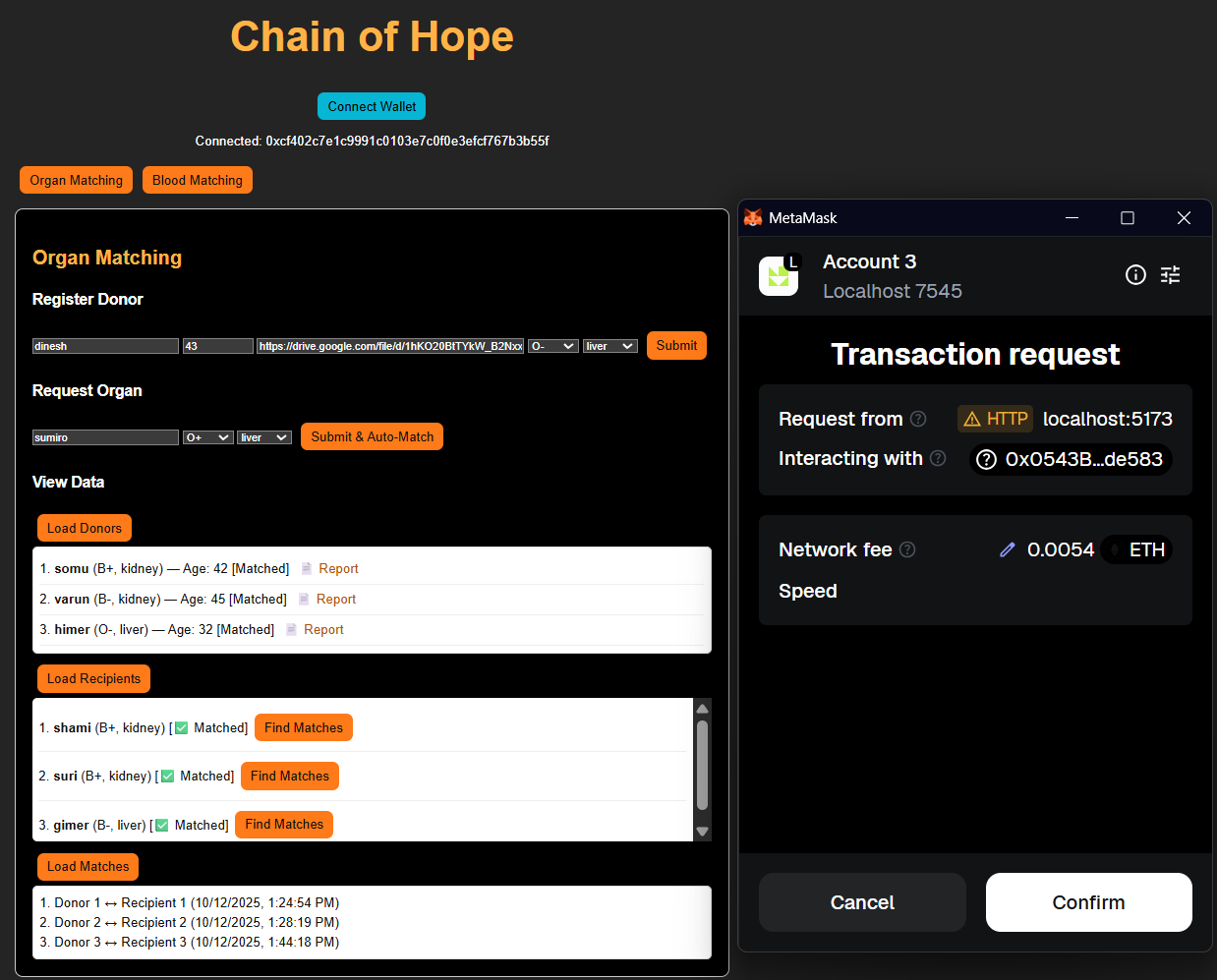
**Peace, Justice and Strong Institutions (SDG 16) :**

SDG 16 focuses on promoting strong institutions, justice, and accountability. Your project addresses this by leveraging blockchain’s transparency and immutability to eliminate manipulation in donor allocation and prevent corruption in donation workflows. The system creates a trusted, auditable record of registrations, matches, and allocations, ensuring fairness and accountability in healthcare governance. By doing so, it builds trust among citizens and institutions, aligning with the goal of strong, transparent systems.

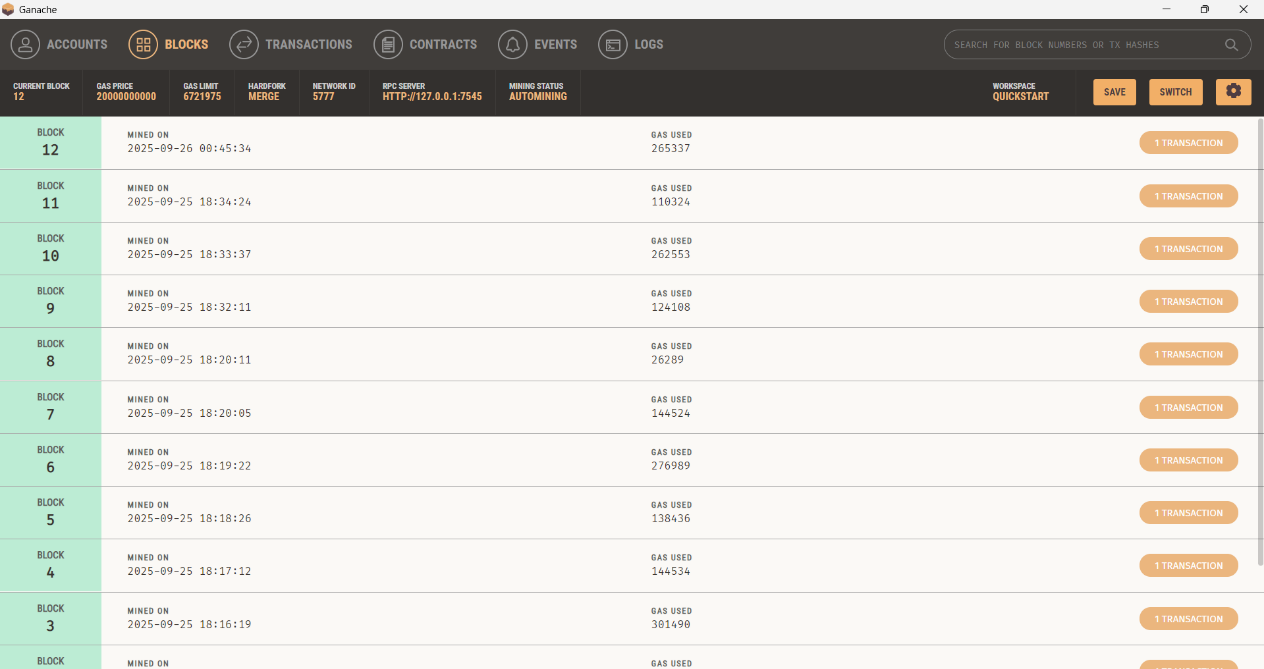
# A2. SCREENSHOTS

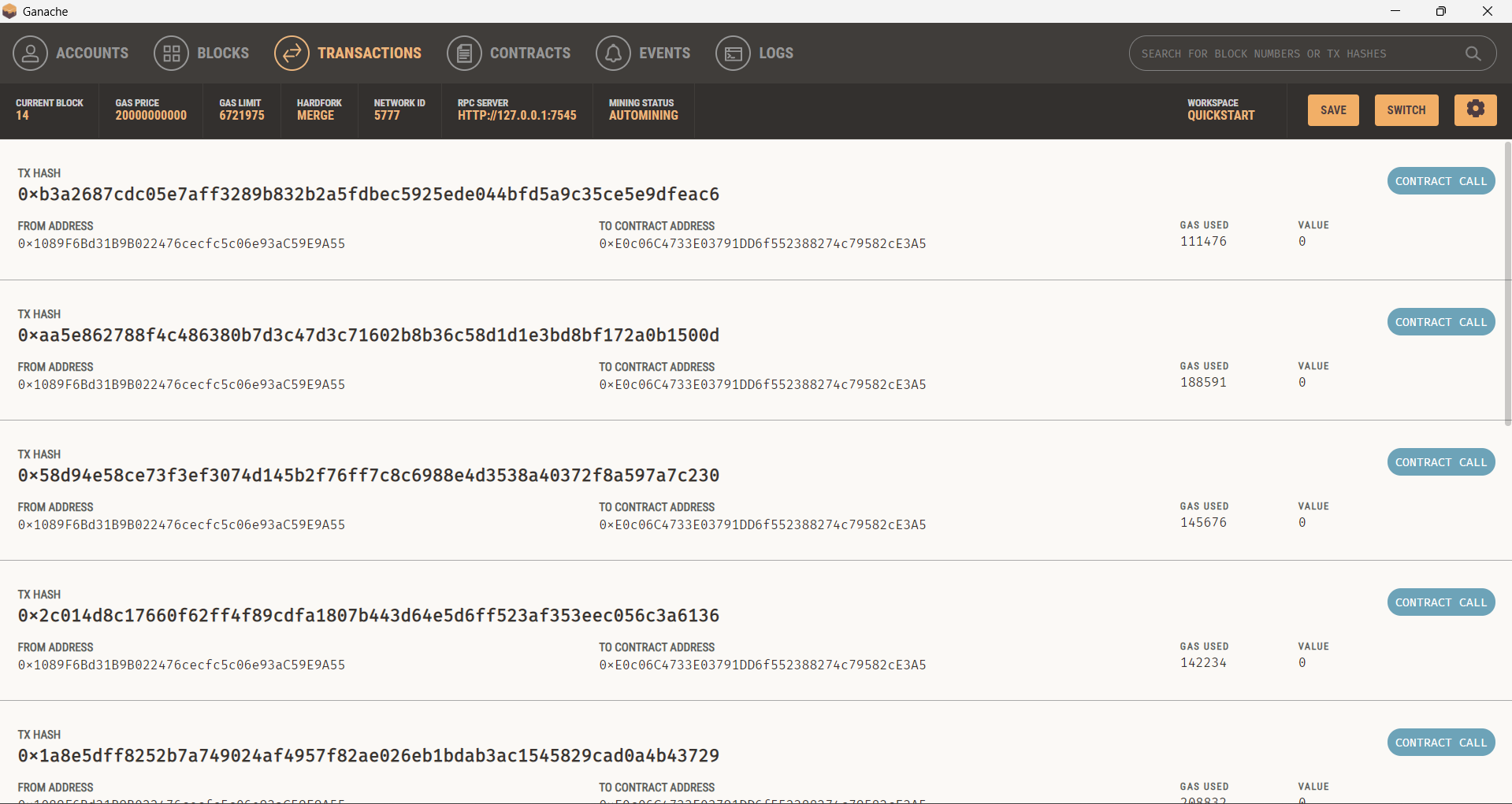


**Fig:A.9.1.Screenshot of blood matching**

****

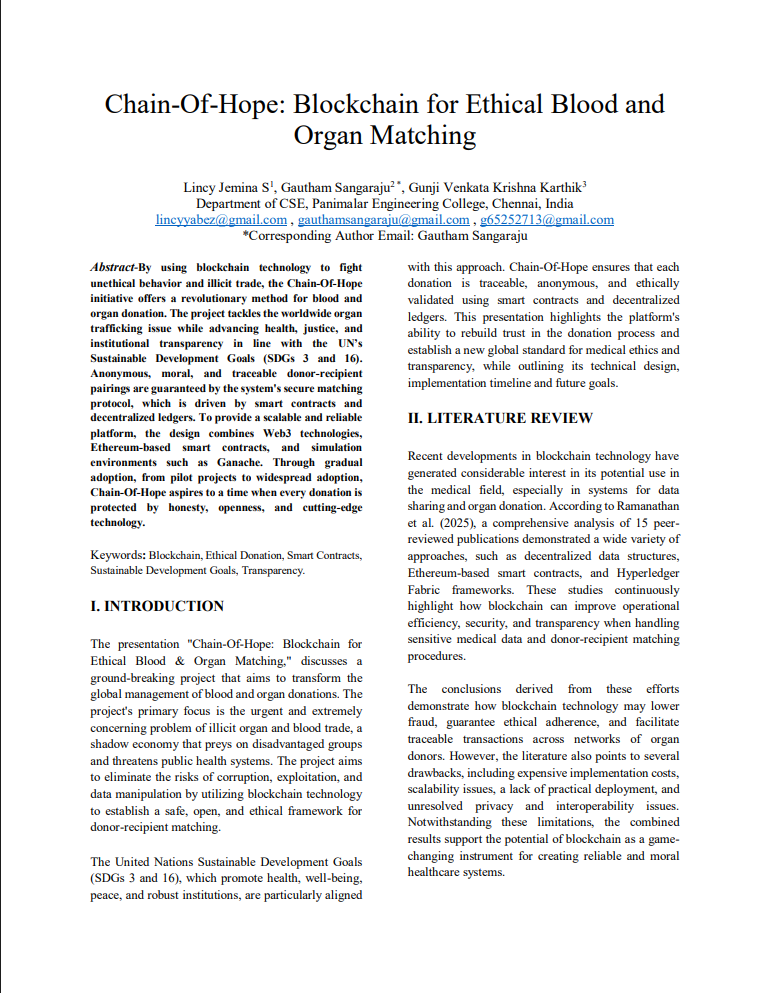
**Fig:A.9.2.Screenshot of organ matching**



**Fig:A.9.3. Screenshot of blocks in ganache**

**Fig A.9.4. Screenshot of transactions in ganache**

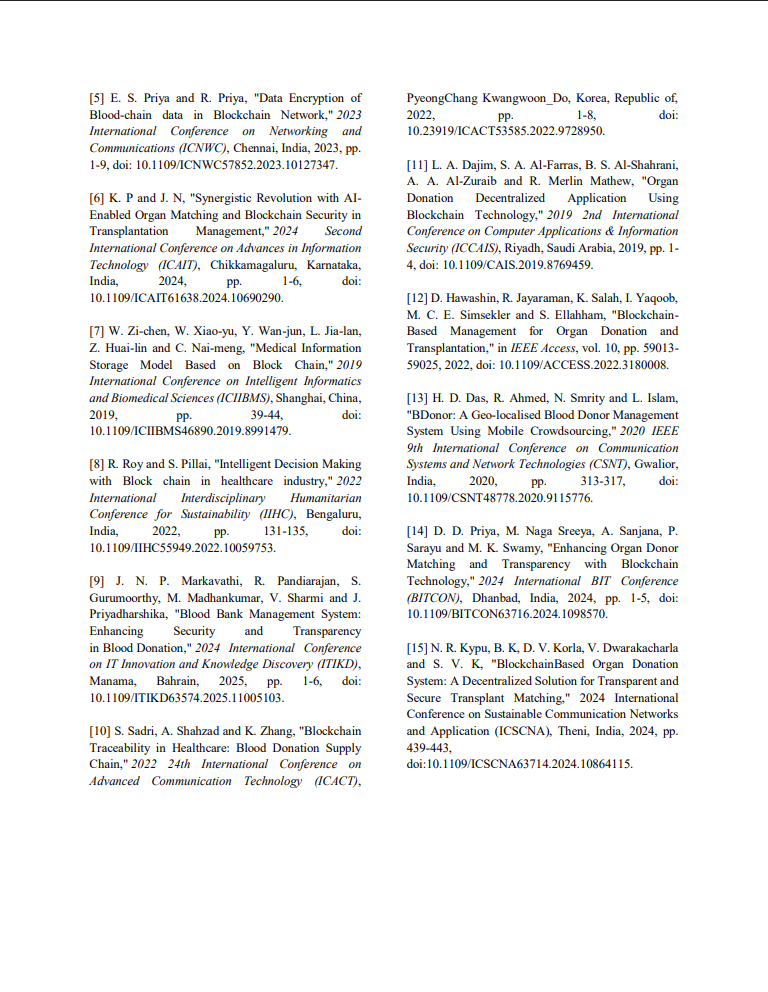
# A3. PAPER PUBLICATION

****

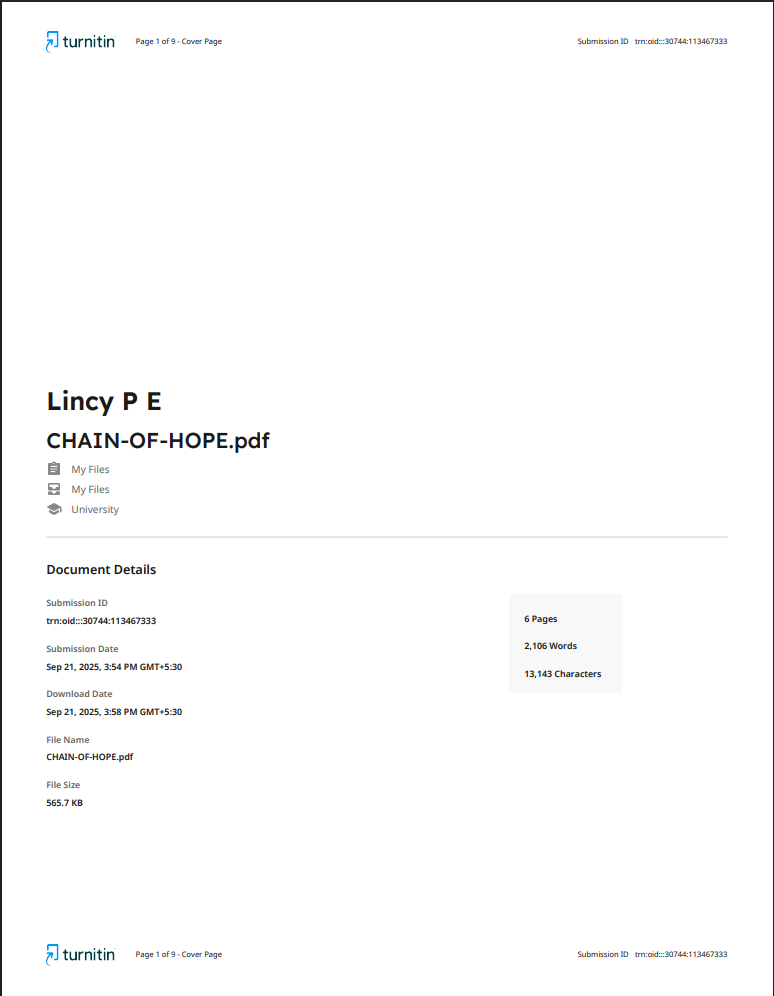
****

****

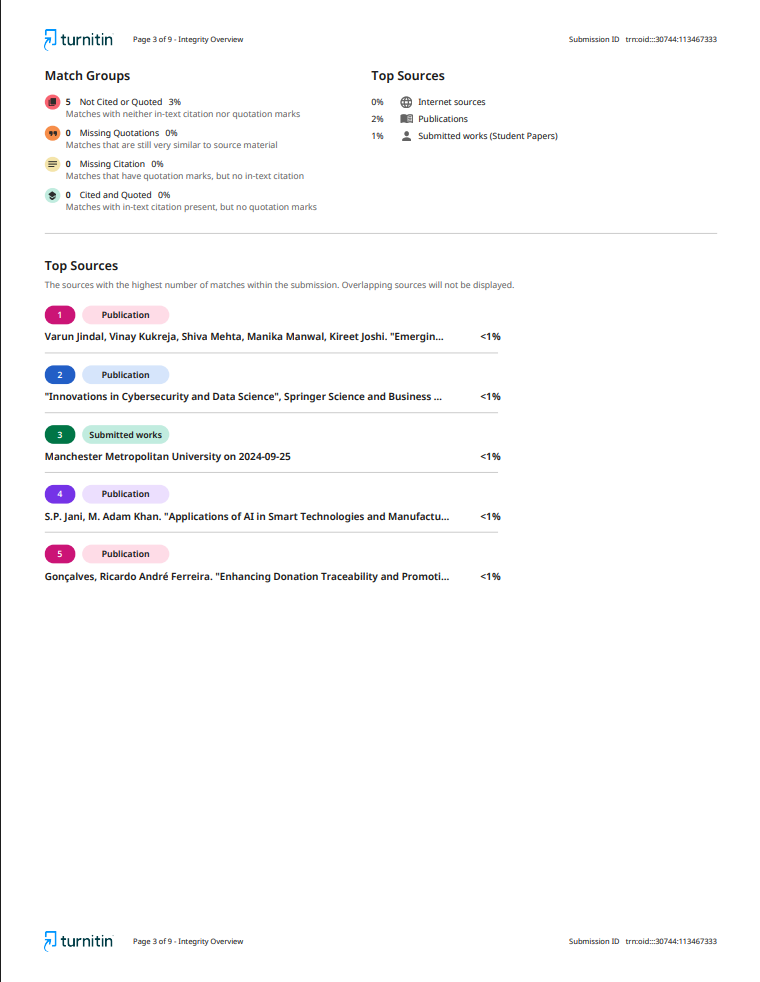
****

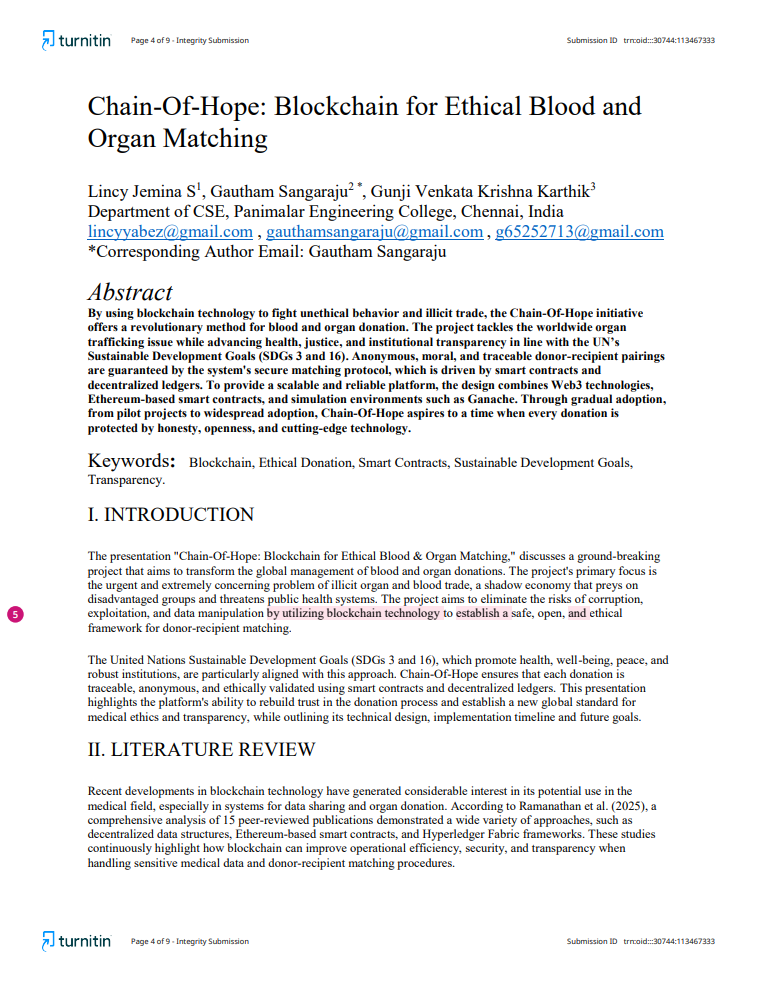
****

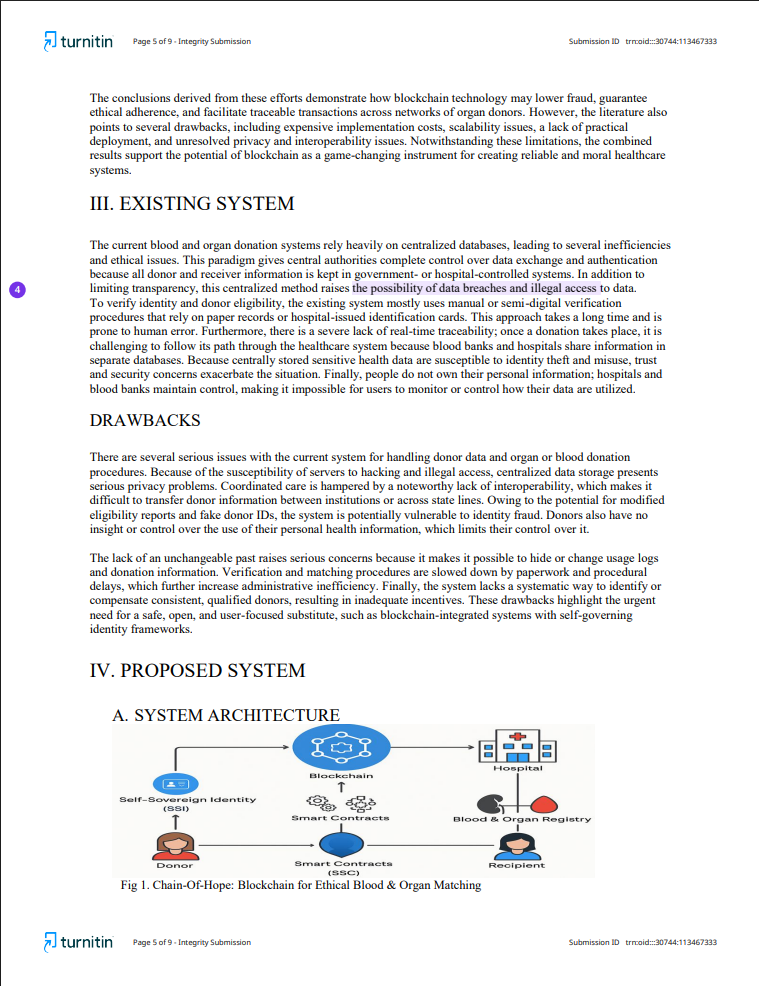
**A4. PLAGIARISM REPORT**

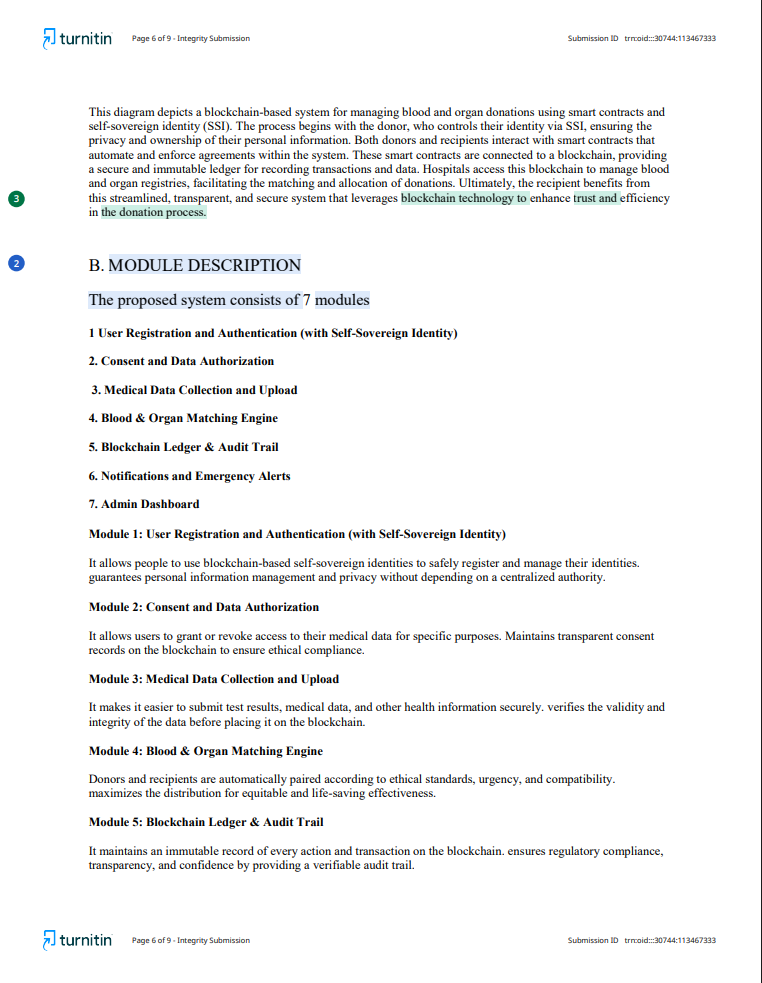
****

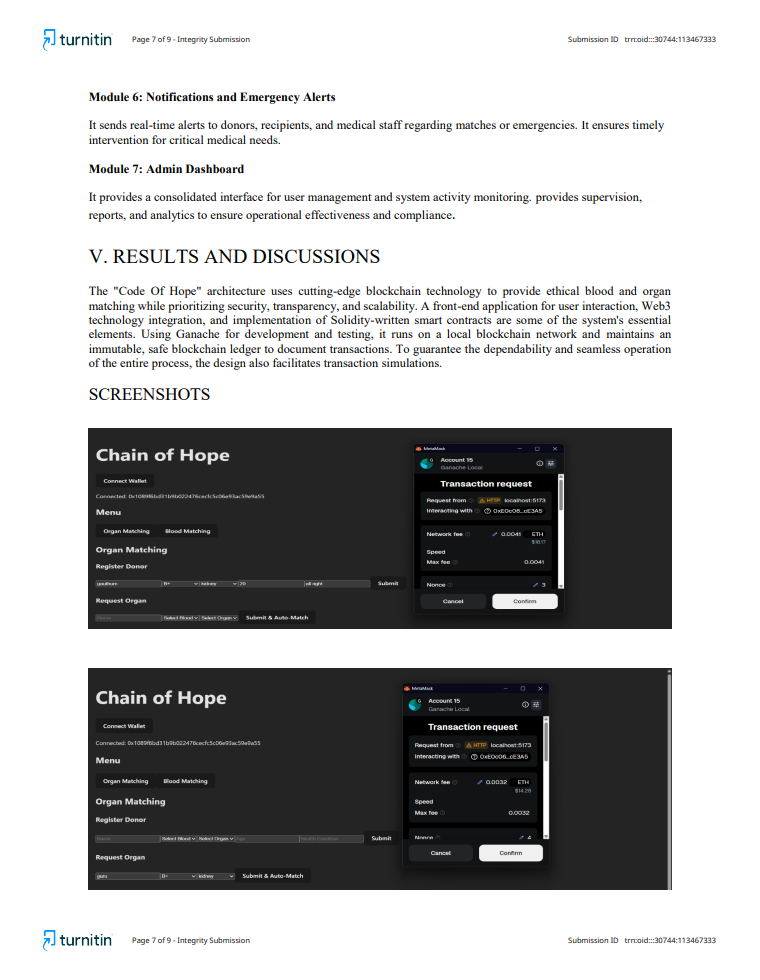
****

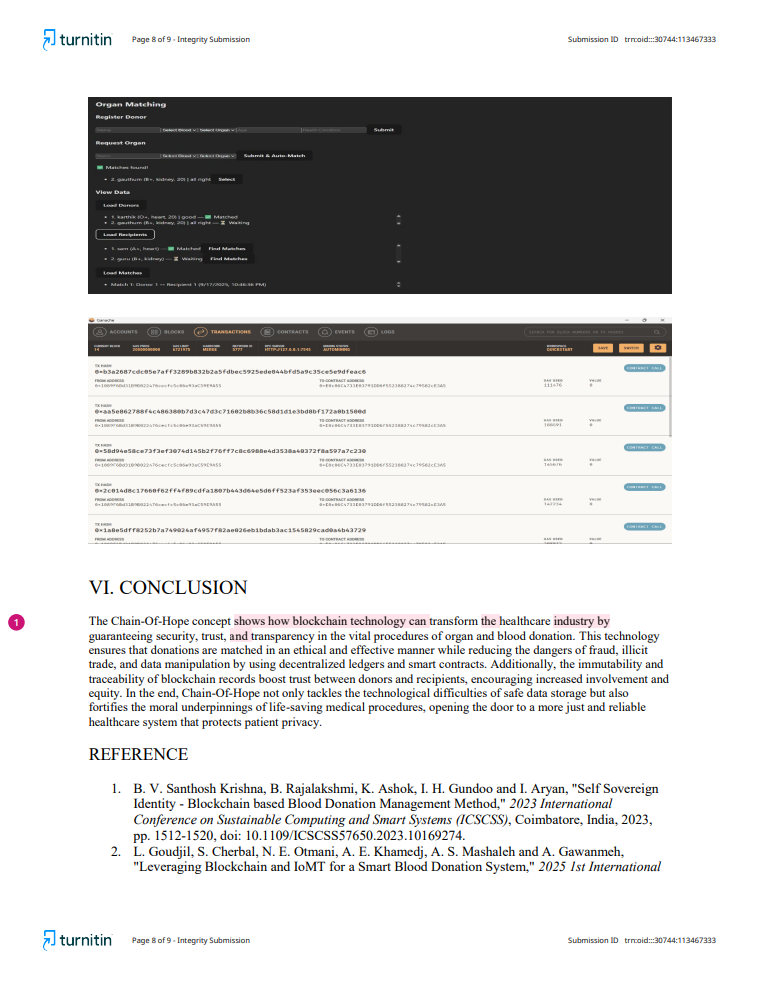
****

****

****

****

****

****

****

**CHAPTER 10**

**REFERENCES**

[1] B. V. Santhosh Krishna, B. Rajalakshmi, K. Ashok, I. H. Gundoo and Aryan, "Self Sovereign Identity - Blockchain based Blood Donation Management Method," *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, Coimbatore, India, 2023, pp. 1512-1520, doi: 10.1109/ICSCSS57650.2023.10169274.

[2] L. Goudjil, S. Cherbal, N. E. Otmani, A. E. Khamedj, A. S. Mashaleh and A. Gawanmeh, "Leveraging Blockchain and IoMT for a Smart Blood Donation System," *2025 1st International Conference on Computational Intelligence Approaches and Applications (ICCIAA)*, Amman, Jordan, 2025, pp. 01-06, doi: 10.1109/ICCIAA65327.2025.11013492.

[3] C. Niyigena, S. Seol and A. Lenskiy, "Survey on Organ Allocation Algorithms and Blockchain-based Systems for Organ Donation and Transplantation," *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South), 2020, pp. 173-178, doi: 10.1109/ICTC49870.2020.9289421.

[4] S. Preethi and C. Priyadharsini, "A Literature Review on Secured Data Management using Block Chain Technology in Healthcare Sector," *2021 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*, Pune, India, 2021, pp. 1-5, doi: 10.1109/SMARTGENCON51891.2021.9645826.

[5] E. S. Priya and R. Priya, "Data Encryption of Blood-chain data in Blockchain Network," *2023 International Conference on Networking and Communications (ICNWC)*, Chennai, India, 2023, pp. 1-9, doi: 10.1109/ICNWC57852.2023.10127347.

[6] K. P and J. N, "Synergistic Revolution with AI-Enabled Organ Matching and Blockchain Security in Transplantation Management," *2024 Second International Conference on Advances in Information Technology (ICAIT)*, Chikkamagaluru, Karnataka, India, 2024, pp. 1-6, doi: 10.1109/ICAIT61638.2024.10690290.

[7] W. Zi-chen, W. Xiao-yu, Y. Wan-jun, L. Jia-lan, Z. Huai-lin and C. Nai-meng, "Medical Information Storage Model Based on Block Chain," *2019 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Shanghai, China, 2019, pp. 39-44, doi: 10.1109/ICIIBMS46890.2019.8991479.

[8] R. Roy and S. Pillai, "Intelligent Decision Making with Block chain in healthcare industry," *2022 International Interdisciplinary Humanitarian Conference for Sustainability (IIHC)*, Bengaluru, India, 2022, pp. 131-135, doi: 10.1109/IIHC55949.2022.10059753.

[9] J. N. P. Markavathi, R. Pandiarajan, S. Gurumoorthy, M. Madhankumar, V. Sharmi and J. Priyadharshika, "Blood Bank Management System: Enhancing Security and Transparency in Blood Donation," *2024 International Conference on IT Innovation and Knowledge Discovery (ITIKD)*, Manama, Bahrain, 2025, pp. 1-6, doi: 10.1109/ITIKD63574.2025.11005103.

[10] S. Sadri, A. Shahzad and K. Zhang, "Blockchain Traceability in Healthcare: Blood Donation Supply Chain," *2022 24th International Conference on Advanced Communication Technology (ICACT)*, PyeongChang Kwangwoon\_Do, Korea, Republic of, 2022, pp. 1-8, doi: 10.23919/ICACT53585.2022.9728950.

[11] L. A. Dajim, S. A. Al-Farras, B. S. Al-Shahrani, A. A. Al-Zuraib and R. Merlin Mathew, "Organ Donation Decentralized Application Using Blockchain Technology," *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, Riyadh, Saudi Arabia, 2019, pp. 1-4, doi: 10.1109/CAIS.2019.8769459.

[12] D. Hawashin, R. Jayaraman, K. Salah, I. Yaqoob, M. C. E. Simsekler and S. Ellahham, "Blockchain-Based Management for Organ Donation and Transplantation," in *IEEE Access*, vol. 10, pp. 59013-59025, 2022, doi: 10.1109/ACCESS.2022.3180008.

[13] H. D. Das, R. Ahmed, N. Smrity and L. Islam, "BDonor: A Geo-localised Blood Donor Management System Using Mobile Crowdsourcing," *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*, Gwalior, India, 2020, pp. 313-317, doi: 10.1109/CSNT48778.2020.9115776.

[14] D. D. Priya, M. Naga Sreeya, A. Sanjana, P. Sarayu and M. K. Swamy, "Enhancing Organ Donor Matching and Transparency with Blockchain Technology," *2024 International BIT Conference (BITCON)*, Dhanbad, India, 2024, pp. 1-5, doi: 10.1109/BITCON63716.2024.1098570.

[15] N. R. Kypu, B. K, D. V. Korla, V. Dwarakacharla and S. V. K, "Blockchain-Based Organ Donation System: A Decentralized Solution for Transparent and Secure Transplant Matching," *2024 International Conference on Sustainable Communication Networks and Application (ICSCNA)*, Theni, India, 2024, pp. 439-443, doi: 10.1109/ICSCNA63714.2024.10864115.